Unidade 1 - Representação da informação

1.1 - Primeiras palavras

Para que se possa ter domínio sobre como as informações são manipuladas por computadores, é preciso, inicialmente, ter uma boa noção de como elas são representadas.

A representação de informações está, em um primeiro momento, diretamente relacionada a códigos. Por exemplo, convencionamos que o símbolo **A** representa o fonema que todos conhecem (existente, por exemplo, no início da palavra *abacate*). Para que os códigos funcionem, é preciso que todos que compartilhem tal código conheçam sua representação. Cada um consegue fazer a leitura deste texto porque conhece o código de representação das letras, o código de combinação das letras para a formação de palavras, o código usado para dar significado a cada palavra e os códigos de sintaxe e semântica que permitem dar significado a este conjunto de letras, espaços e símbolos de pontuação.

Para que este conceito fique mais claro, um exemplo de representação numérica pode ser utilizado. Esta representação é a clássica numeração em números romanos. O código romano que indica que o valor 1 é representado por I, o valor 5 por V, o 10 por X, o 50 por L, o 100 por C, o quinhentos por D e o mil por M. Na formação de outros valores, combinações precisas são usadas: III para o 3, IV para o 4, IX para o 9 etc., o que é estabelecido por regras de soma (XI é X somado a I, ou seja, 11) e subtração (XL é X subtraído de L, ou seja, 40). Sabendo-se o código (símbolos e regras usadas para a interpretação), fica "fácil" dizer que MCMLXXI representa 1981. (Fica aqui um desafio ao leitor para dizer como os romanos indicavam o número zero.)

Um segundo exemplo relativo a valores numéricos é o próprio sistema decimal, amplamente utilizado por todos. Os símbolos incluem os dígitos de 0 a 9 e as regras de representação assumem grandezas diferentes dependendo da posição do dígito do número. Assim, o dígito 8 em 18 significa 8 unidades, enquanto o mesmo dígito em 81 significa 8 dezenas. O uso de uma vírgula para partes decimais também pode ser empregado, atribuindo potências negativas de 10 ao significado do dígito, de forma que, por exemplo, o dígito 8 em 1,48 passa a indicar 8 centésimos. E isso sem considerar números positivos ou negativos.

Fugindo dos exemplos numéricos, existe a representação alfabética, usada para a composição de palavras. As letras de A a Z são usadas para formar palavras e a justaposição delas determinam como devem ser lidas (e.g., CA para "ká" e CE para "cê"). Além destes, outros símbolos como acentos e cedilhas podem ser usados para modificar como a leitura é feita. Um dos problemas do uso das letras em palavras é que o som associado às combinações não é preciso. Os casos mais clássicos são as dificuldades com G e J (berinjela ou berinjela?) e com o X (chícara ou xícara?). Ou, ainda, quantas pessoas sabem que a pronúncia correta de *sintaxe* é "sintas*s*e" e não "sinta*c*se"? O problema de codificações abertas é a imprecisão na interpretação, o que não pode ser tolerado no armazenamento de dados.

Outras formas de codificação da informação são os números em base 2, os números em hexadecimal, os alfabetos grego e cirílico, as placas de trânsito, o código Morse e a escrita em Braile, por exemplo.

Em um segundo momento, para que uma informação possa ser representada, é preciso salientar que existem alguns elementos importantes:

1. Um conjunto de símbolos deve ser escolhido
2. Regras para combinar os símbolos devem ser estabelecidas
3. Os agentes que vão representar ou interpretar estes símbolos devem conhecer e concordar com a codificação

Limitar o conjunto de símbolos e conhecê-lo por completo é importante para a interpretação. As regras definem como as associações de símbolos devem ser interpretadas e em quais contextos se aplica. Finalmente, quem vai armazenar e quem vai recuperar as informações devem ser conhecedores dos símbolos, regras e contexto, para que a informação que é armazenada seja a mesma que é recuperada

1.2 - Representação básica da informação

Nos computadores, há restrições severas sobre o que usar para representar as informações usando códigos. Assim, como os computadores apenas possuem **bits**, estes passam a ser a única forma de representar as informações. É sabido, também, que cada bit representa apenas 0 ou 1 (desligado ou ligado, apagado ou aceso, ausente ou presente etc.). Resta, para qualquer representação, portanto, apenas os códigos binários, ou seja, formados por zeros e uns.

Cabe, agora, definir como cada código é associado a um significado.

Uma primeira regra, já de longa data, é agrupar os bits em conjuntos de 8, chamando a isso **byte**. Como cada byte tem 8 bits e cada bit assume apenas 2 valores, são possíveis exatas 256 combinações para cada byte, indo de 00000000 a 11111111.

A representação de informação usando os bytes fica, assim, condicionada a:

1. Definir os 256 possíveis bytes como o conjunto de símbolos que podem ser usados
2. Utilizar conjuntos de um ou mais bytes para combinar os símbolos básicos e definir novos símbolos
3. Definir, em comum acordo com o agende que vai guardar a informação e o agente que vai interpretá-la, o que cada combinação de símbolos significa

Alguns poucos exemplos serão dados, na sequência, para mostrar como representar alguns tipos de informação.

Números inteiros sem sinal

Uma das formas de representar um número inteiro positivo usando os bytes como símbolos é fazer a associação direta entre a representação binária do número e os bits dos bytes. Desta forma, o número 33, que tem representação binária 100001_2, é representado pelo byte 00100001. Da mesma forma, 66 é associado ao byte 01000010, enquanto 255 ao byte 11111111.

De posse da regra e concordando com a representação, se houver um byte com o valor 10001010 e ele representar um valor inteiro positivo, pode-se saber, com segurança, que o valor representado é o 138.

A Figura 1-1 mostra um exemplo de quatro bytes, cujos valores representam, segundo a codificação convencionada, os valores inteiros 8, 1, 127 e 10, respectivamente. Neste exemplo, são quatro valores distintos, cada um deles formado por apenas um byte e usando a convenção da representação binária sem sinal para associar os valores a cada byte em particular.

10001000	00000001	10000001	00001010

Figura 1-1 Quatro bytes, cada um representando um valor inteiro sem sinal distinto. Na ordem: 136, 1, 127 e 10

Com esta representação usando apenas um byte, é possível apenas representar os números inteiros de 0 a 255. Ampliar estes valores segue a mesma regra usada nos números decimais: se valores maiores precisam de mais dígitos, valores maiores precisam de mais bytes.

Outra representação para números inteiros sem sinal, bastante comum em computação, usa conjuntos de quatro bytes. Alinhados, os quatro bytes representam 32 bits e permitem 2^{32} combinações, ou seja, valores de 0 a 4.294.967.295. Deve-se notar que a ordem em que os bytes são considerados também é importante e esta escolha também faz parte da regra de codificação. Usando quatro bytes para representar um único valor, o 1.140.899.978, os bytes ficariam como os representados na Figura 1-2.

10001000	00000001	10000001	00001010

Figura 1-2. Quatro bytes que, juntos, representam um único valor inteiro sem sinal, de 32 bits. O valor binário é o $10001000000000110000001000010 10_2$, ou seja, 1.140.899.978 em decimal.

Em C ou C++, estes dois tipos de representação apresentados corresponderiam ao *unsigned char* (quando usado como inteiro) e ao *unsigned int*, respectivamente.

Um ponto importante a ser observado é que as representações têm restrições. O número inteiro 300 não pode ser representado segundo as regras definidas para uso de apenas um byte. Mas estas limitações estão presentes em várias outras codificações, ou alguém sabe como representar 2,75 em números romanos?

Números inteiros com sinal

A representação de números negativos em bytes utiliza, em sua forma mais comum, o primeiro bit para indicar o sinal. Se seu valor for 0, o número é positivo, se for 1, negativo. Porém, como um dos bits é usado para o sinal, resta um bit a menos para representar o valor em si.

Usando um único byte, portanto, o primeiro bit indica o sinal e os sete bits restantes são usados para os números. Sendo o primeiro bit igual a zero, os valores representados vão de 0 (00000000) a 127 (01111111). As outras 127 combinações com o primeiro bit igual a 1 representam os valores de -1 a -128.

Se forem usados quatro bytes para armazenar um valor inteiro com sinal, então, os valores possíveis se iniciam em –2.147.483.648 e vão até 2.147.483.647, dada a reserva do primeiro bit para o sinal.

Quatro bytes, cada um representando um valor inteiro com sinal, são ilustrados na Figura 1-3. O primeiro e o terceiro (da esquerda para a direita) são negativos, com o primeiro bit igual a 1; o segundo e o quarto são positivos.

10001000	00000001	10000001	00001010

Figura 1-3. Representação de quatro inteiros com sinal, cada um com um byte. Valores representados, na ordem: -120, 1, -127 e 10.

A Figura 1-4, por sua vez, mostra um único valor de quatro bytes, que é negativo por seu primeiro bit (neste caso o do primeiro byte) ser igual a 1.

10001000	00000001	10000001	00001010

Figura 1-4. Quatro bytes que, juntos, representam um único valor inteiro com sinal, de 32 bits. O valor, em decimal, é o -2.013.167.350.

Uma observação cabe aqui: a interpretação de números negativos não é direta nestes exemplos, pois foi usada a notação chamada *complemento para 2*. Isso significa que o byte 10000000 representa -128 e não "menos zero". Esta representação não será detalhada neste texto.

Em C ou C++, estes tipos corresponderiam ao *char* (quando usado como inteiro) e ao *int*, respectivamente.

Números reais

Um tipo de dados importante em computação é o que representa os números chamados com "ponto flutuante", que são os valores com parte decimal.

Valores reais podem ser representados por uma convenção não tão óbvia, mas padronizada para vários sistemas. Este é o caso da especificação IEEE 754 [1], que utiliza subconjuntos dos 32 bits para a representação.

Um valor real é codificado, usando esta especificação, dividindo-o em mantissa (m) e expoente (e). Um valor qualquer é representado no formato $m2^e$. A parte referente à mantissa utiliza 23 bits e o expoente usa 8 bits. Um bit para representação de sinal completa os 32 bits.

Numerando cada bit, da esquerda para a direita, de 0 a 31, então o bit 0 indica o sinal da mantissa, os bits de 1 a 9 representam o expoente e os demais, de 10 a 31 formam a mantissa. A Figura 1-5 mostra a combinação de bits, usando quatro bytes e a convenção da especificação IEEE, que representa o valor 100,5.

01000010	11001001	00000000	00000000

Figura 1-5. Um número em ponto flutuante usando quatro bytes consecutivos. O valor representado é 100,5.

Em C ou C++, este exemplo corresponde às variáveis do tipo *float*.

Caracteres

O uso dos bytes na representação de caracteres é feito por uma associação arbitrária entre os caracteres e as combinações de bits.

Em algum momento, convencionou-se que o desenho **b** (ou **b**) fosse usado para representar a letra "bê", sendo que qualquer outro símbolo poderia ter sido escolhido. Os gregos, por exemplo, optaram por ***B***.

A convenção mais comum de representação de caracteres em sistemas computacionais é dada pela chamada tabela ASCII (American Standard Code for Information Interchange, ou código padrão americano para intercâmbio de informações). Esta tabela define, arbitrariamente, a qual código de bits está associado cada caractere. Por exemplo, o dígito **0** é representado pelo byte 00110000, o **J** pelo 01001010, o **j** pelo 01101010 e o **%** pelo 00100101.

Além da tabela ASCII, outras tabelas também populares incluem a EBCDIC (Extended Binary Coded Decimal Interchange Code, ou código de intercâmbio estendido de decimais codificados em binário) e a UNICODE (Unicode Standard, ou padrão de "código único"). A codificação UTF-16, em particular, é uma versão da representação UNICODE de 16 bits (2 bytes) para representar caracteres.

Cadeias de caracteres

Cadeias de caracteres, usadas para a representação de textos, são usualmente sequências de bytes que representam caracteres. Um controle adicional deve ser acrescido à sequência para controlar quantos caracteres são válidos.

Aqui são comentadas duas formas comuns para a representação de cadeias de caracteres: a da linguagem C padrão e da linguagem Pascal.

Em C, as cadeias de caracteres são representadas como arranjos (vetores) de caracteres. São considerados válidos para o texto armazenado todos os caracteres a partir do início do arranjo até que seja encontrado um byte com valor 00000000, indicado na linguagem por '\0'. Na linguagem Pascal, opta-se por reservar o primeiro byte do tipo *string* para armazenar o número de caracteres que devem ser considerados como válidos. Como apenas um byte é usado para indicar esta quantidade, há um máximo de 255 caracteres para este tipo de dados.

A Figura 1-6 mostra uma representação de uma variável para guardar textos em C. Declarada como uma cadeia de caracteres de 6 bytes, representa os caracteres *AABC* (usando codificação ASCII). O quinto byte indica o fim da sequência válida; os caracteres seguintes são desprezados.

01000001	01000001	01000010	01000011	00000000	01000001

Figura 1-6. Representação de quatro bytes para uma variável declarada como *char[6]* e armazenando o literal *AABC*.

1.3 - Informações na memória principal

Na memória principal, as informações são armazenadas nos bytes disponíveis, usando representações como as descritas no tópico anterior, além de várias outras. A forma mais usual de definir uma área da memória para uso é por meio das variáveis dos programas; outra forma é por meio de alocação dinâmica de memória.

O tratamento de especificidades como tipos de memória, o que envolveria memórias RAM e cache, entre outras, não pertence ao contexto deste texto. Assim, salvo em ocasiões deixadas explícitas, assume-se que o termo memória apenas se refere à memória principal, ou seja, a uma memória RAM comum.

Como características da memória principal, dentro do escopo desta disciplina, destacam-se:

1. A área de memória é composta de bytes dispostos de forma sequencial, sem divisões
2. Cada byte possui seu endereço de memória
3. O tempo de acesso (consulta do conteúdo ou armazenamento de padrão de bits) a qualquer byte da memória é considerado similar, sendo feito muito rapidamente
4. A memória é considerada volátil
5. A quantidade total de bytes existentes para armazenamento (tamanho da memória) é considerada de tamanho arbitrário, porém finito

O entendimento do uso da memória passa por saber como os bytes são usados para armazenar as informações. Este conceito passa pela representação da informação, que pode ocorrer conforme os exemplos indicados no tópico anterior.

Como ilustração do conceito, pode-se considerar a declaração de variáveis em um programa escrito na linguagem C, apresentada no Algoritmo 1-1.

Algoritmo 1-1

```
1    /* exemplos de declaracoes */
2    int i;
3    char n[5];
4    float f;
```

A variável **i** é um inteiro com sinal e ocupa quatro bytes consecutivos na memória; **n** consome outros cinco bytes consecutivos, sendo usada para armazenar um caractere por byte; a variável **f**, assim como **i**, ocupa quatro bytes consecutivos e serve para armazenar um valor em ponto flutuante.

Um ponto importante é o que diferencia os quatro bytes da variável **i** dos da variável **f**. A resposta é... nada. Como ocorre com qualquer byte na memória, não existem diferenças entre um e outro. Assim, reconhecer como os bytes da variável inteira são diferenciados da variável *float* é apenas uma convenção, ou seja, é preciso saber **o que** está armazenado. No programa, os bytes reservados para a variável **i** sempre serão interpretados como um valor inteiro com sinal e seguem, assim, a interpretação adequada para seus 32 bits. No caso da variável **f**, a interpretação como um valor IEEE 754 é definida pelo tipo da variável.

Um exemplo alternativo, sobre saber o que é a representação para interpretar corretamente um símbolo, corresponde ao símbolo X. Se se souber que é uma letra, trata-se do “xis”; se for um número

romano, é o valor 10. Ter apenas o símbolo não é suficiente para saber que informação ele representa; é requerido que se tenha em mãos também a convenção que deve ser usada na interpretação.

Assim, não é padrão que haja, na memória, alguma referência ao tipo de seu conteúdo. A função de interpretar corretamente a codificação usada para representar a informação é responsabilidade de quem usa o dado. É possível, por exemplo, que os bytes da variável inteira sejam interpretados como um *float*[*]; nesta situação, os bits seriam interpretados de forma diferente e os valores reconhecidos seriam completamente diferentes.

Esta interpretação diferenciada do mesmo padrão de bits na memória pode ser observada desde a Figura 1-1 até a Figura 1-4, nas quais o mesmo padrão de bits é interpretado de formas diferentes, dependendo apenas das convenções usadas.

1.4 - Informações na memória secundária

O termo memória secundária é aplicado a formas de armazenamento "fora" do computador. Em geral, a memória secundária se refere a um dispositivo de armazenamento em massa, como é o caso dos discos rígidos, memórias *flash* (*pen drives* e cartões de memória) e discos ópticos, entre outros.

Memórias secundárias se caracterizam com comportamento diferenciado da memória principal, justamente em função de suas características físicas e lógicas. As principais características da memória secundária incluem:

1. A área de memória é composta de bytes dispostos de forma sequencial, com divisões dependentes do dispositivo
2. Cada byte ou conjunto de bytes possui seu endereço no dispositivo
3. O tempo de acesso (consulta do conteúdo ou armazenamento de padrão de bits) a cada byte depende das características do dispositivo no qual é feito o armazenamento, sendo muito mais lento quando comparado ao da memória principal
4. O armazenamento é permanente
5. O tamanho da memória secundária é limitado e, em geral, bastante superior à capacidade de armazenamento da memória principal

Assim como na memória principal, uma codificação é escolhida para armazenar cada informação, sendo usadas as mesmas convenções empregadas na memória principal. Isso quer dizer que um valor inteiro com sinal de dois bytes é representado na memória secundária da mesma forma que na memória principal, sendo raramente necessárias conversões de representação.

A principal diferença entre a memória secundária e a primária é o tempo de acesso. Qualquer que seja a mídia usada para o armazenamento secundário, o tempo de acesso é superior. Na realidade é muito superior: enquanto o tempo de acesso à memória é da ordem de nanossegundos (10^{-9}s), um disco rígido moderno consome tempo na ordem de milissegundos (10^{-3}s); isto dá uma razão de um milhão de vezes. Outros dispositivos como CDs ou *pen drives*, embora apresentem razões de tempo diferentes das dos discos rígidos, também são significativamente diferentes da memória principal.

Um fator extremamente relevante quando se fala de memórias principal e secundária é que o processador somente tem acesso aos dados da memória principal. Ao se necessitar de um dado armazenado em disco, por exemplo, é necessário primeiro transferi-lo para a memória principal. Nesta última o dado é trabalhado e, eventualmente, transferido de volta ao disco para armazenamento.

Como o armazenamento de dados está atualmente concentrado em discos rígidos, sendo exemplo os servidores web e os de bancos de dados, o enfoque na disciplina será dado ao armazenamento secundário neste tipo de dispositivo.

1.4.1 - Discos rígidos como memória secundária

Os discos rígidos são a principal forma de armazenamento permanente de dados, estando presentes desde em pequenos notebooks até grandes servidores.

Sendo o tempo de acesso ao disco muito maior que à memória principal e dada a necessidade de que o dado esteja presente nesta última para que seja manipulado pelo computador, o tempo de acesso é a característica mais importante que deve ser considerada quando se trabalha com armazenamento secundário.

Assim, é preciso que se tenha uma boa idéia de como funcionam os discos rígidos e as razões que influenciam o tempo para se armazenar ou recuperar um dado.

[*] Isso é viável fazendo um ponteiro para *float* apontar para o endereço da variável inteira.

Esta discussão passa por entender a geometria básica dos discos rígidos, como o sistema operacional (ou um SGBD*) organiza os dados em disco e quais os principais retardos que influenciam no tempo de acesso.

Geometria dos discos rígidos

Os discos rígidos representam uma classe de armazenamento secundário que utilizam magnetismo. Vários discos metálicos giram em torno de um eixo e cabeças de leitura e gravação são usadas para obter ou guardar bits sobre a superfície destes discos (Figura 1-7).

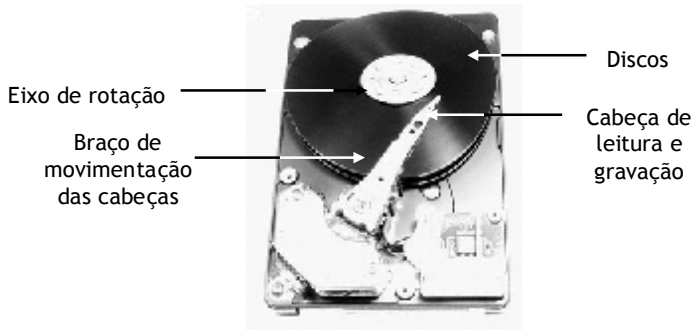

Figura 1-7. Um disco rígido (Imagem de Corbys Holding, Inc.).

Os bits são gravados ao longo de círculos concêntricos (**trilhas**), sendo que cada círculo é usualmente segmentado em trechos regulares (**setores**). A Figura 1-8 mostra uma distribuição de 9 trilhas, divididas em 8 setores, sobre uma das superfícies do disco. Os bits são gravados ao longo de cada trilha, distribuídos em seus setores.

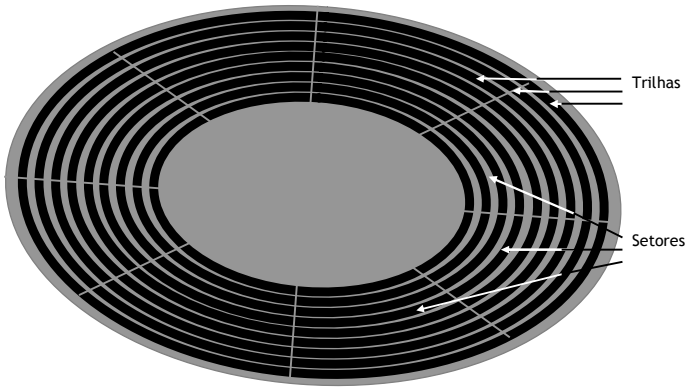

Figura 1-8. Ilustração da superfície de um dos discos, indicando as trilhas (círculos concêntricos) e os setores (segmentos das trilhas).

Um disco rígido, na prática, é constituído de vários discos que giram juntos. Cada disco permite gravação em ambas suas faces e cada uma das faces tem sua própria cabeça de leitura e gravação. As cabeças são presas a um braço (Figura 1-8), de modo que todas se movimentam como um único bloco. Enquanto o braço faz com que as cabeças se movimentem radialmente, o disco gira sob elas. As gravações e leituras ocorrem enquanto o setor de uma dada trilha está sob a cabeça de gravação.

A capacidade de um disco é função do número de superfícies, do número de trilhas, do número de setores e da capacidade, em bytes, de armazenamento de cada setor. Usualmente um setor contém uma quantidade de bytes dada como uma potência de 2. Um dos discos rígidos da Seagate, de 1 terabyte contém, por exemplo, um total de 1.953.525.168 setores de 512 bytes cada um [2].

O sistema operacional e a recuperação de dados

O sistema operacional de um computador é responsável por se organizar para localizar os dados no disco rígido. Isso significa definir forma de organizar a área de dados para boot, fazer o gerenciamento das listas com os nomes e atributos dos arquivos e estruturar como localizar tais arquivos no disco rígido. Estabelece-se, assim, um mapeamento entre as trilhas e setores existentes no disco rígido e a organização definida pelo sistema operacional.

* Sistema gerenciador de banco de dados.

A forma como cada sistema operacional ou SGBD define sua organização de arquivos e dados não faz parte dos interesses deste texto. Assim, fica sugerido ao leitor que procure informações adicionais pesquisando por sistemas de arquivos como EXT3, ReiserFS, FAT32 e NTFS, entre outros.

Dentro do escopo desta disciplina, apenas algumas características desta organização são consideradas relevantes. Embora muitas outras características sejam consideradas importantes, não serão tratadas aqui por motivos de simplificação.

Os sistemas operacionais, em função do sistema de arquivos escolhido, definem um tamanho padrão para transferências de dados do disco rígido ou para ele. Este tamanho representa um **bloco** de disco, também conhecido por **cluster** ou **página** de disco.

Um bloco, usualmente dado por uma quantidade de bytes que seja potência de 2, permite ao sistema reservar espaços nas memórias cache e principal para agilizar as transferências de dados e dividir o espaço em disco para acomodar os diversos arquivos.

Para este texto, um bloco é considerado como uma quantidade fixa[*] de bytes que podem ser armazenada ou recuperada do disco rígido de cada vez. Em outras palavras, o sistema é capaz de transferir somente esta quantidade de bytes de cada vez; nem mais, nem menos. Assume-se, também, que o espaço em memória (denominado **buffer**) para realizar estas transferências seja equivalente a exatamente um bloco.

Supondo, a título de exemplo, que o sistema operacional utilize blocos de 2 KiB (2048 bytes) para seu sistema de arquivos e que já se saiba em quais setores do disco um determinado arquivo se encontre, a leitura dos dados pode ser realizada. Considerando que os dados correspondam a um texto simples que necessite 1578 bytes (entre caracteres, espaços, mudanças de linhas, tabulações e etc.), a leitura dos dados para a memória deve ser solicitada por um programa. O sistema operacional repassa ao hardware as instruções para a leitura, o que resulta na transferência de exatos 2048 bytes (um bloco) para o buffer na memória principal. Terminada esta transferência, os 1578 primeiros bytes do buffer são repassados para o programa. O sistema se encarrega do controle para que os bytes restantes no buffer não sejam considerados como dados válidos.

Sob as mesmas condições, um segundo exemplo é um programa solicitando a leitura apenas da primeira linha do mesmo arquivo texto, assumindo que tal linha tenha 56 bytes. A solicitação de leitura é passada para o sistema operacional, que executa a transferência de um bloco do disco para o buffer. A partir do buffer, os 56 primeiros bytes são passados ao programa. Se, posteriormente o programa solicitar outra linha (supondo agora outros 75 bytes), os próximos 75 bytes do buffer são repassados do buffer para o programa, não necessitando novo acesso a disco, visto que os dados já se encontram na memória. Novas solicitações do programa por dados irão consumindo os bytes do buffer, até completar os 1578 bytes válidos ali armazenados.

Havendo um arquivo que ocupe mais do que um bloco em disco rígido, uma nova leitura a disco somente ocorrerá quando os bytes do primeiro bloco já tiverem sido consumidos. Assim que o buffer é liberado, nova leitura é feita (isto é, um novo acesso a disco) e os dados continuam a ser repassados para o programa. Qualquer arquivo maior que 2048 bytes, dadas as condições do exemplo descrito anteriormente, está sujeito a esta situação.

O processo de gravação é similar. Os programas transferem dados, por meio de comandos de escrita, para o buffer do sistema operacional. O sistema decide quando estes dados são efetivamente transferidos para o disco. De qualquer forma, cada gravação de dados em disco rígido exige a transferência de um bloco completo.

Naturalmente as descrições acima são simplificações da realidade. Podem existir vários buffers (de um bloco cada um), tanto para leitura quanto para escrita, pode haver memórias cache tanto no circuito do disco rígido quanto na placa mãe e, além disso, o sistema não cuida apenas de um único programa e um único arquivo, mas de vários de ambos simultaneamente. Para justificar as organizações de dados da parte de arquivos, entretanto, esta simplificação apresentada é bastante útil.

Adicionalmente, além do sistema operacional, alguns SGBDs podem controlar diretamente uma partição do disco rígido e manter sua própria estrutura de mapeamento de dados. Nestes casos, o SGBD retira do sistema operacional este controle. Os conceitos de blocos e de sua utilização, porém, permanecem inalterados.

Tempos de acesso para armazenamento e recuperação

Mesmo com o progresso que experimenta a tecnologia, o acesso a um disco rígido é muito limitado por suas características mecânicas. O envolvimento de movimentos de partes mecânicas, como o braço das cabeças de leitura e gravação, acrescenta latências importantes, como quebrar a inércia do braço parado, acelerá-lo e movimentá-lo em direção à trilha correta, desacelerá-lo até parar sobre a trilha desejada para, finalmente, aguardar a informação passar sob a cabeça e realizar a transferência de dados.

Dos diversos fatores que influenciam o tempo e geram **atrasos**, ou **latências**, são elencados três:

[*] Alguns sistemas de arquivos permitem blocos de tamanhos variados. Estes não serão considerados nesta disciplina.

1. Latência de busca
2. Latência de rotação
3. Latência de transferência

Todo dado que tem que ser transferido do disco ou para ele depende da cabeça de leitura e gravação estar posicionada sobre o dado. Isto requer, em um primeiro momento, mover o braço de forma a posicionar as cabeças sobre a trilha correta. O tempo despedido para fazer esta movimentação gera a **latência de busca**. Quanto mais longe da trilha correta estiver o braço, maior o tempo necessário para acertar a posição. Em sistemas multitarefas, as várias requisições simultâneas de acesso são ordenadas para minimizar a movimentação do braço e, assim, reduzir o tempo de seu deslocamento. Portanto, não importa a ordem em que as solicitações de acesso chegam, pois o atendimento a cada uma delas segue o menor necessidade de movimentação do braço.

A **latência de rotação** é o tempo necessário para que, estando a cabeça de leitura e gravação sobre a trilha correta, o início dos dados passe sob ela. Em algumas situações os bytes poderão estar quase que imediatamente disponíveis ou, no pior caso, uma rotação completa tem que ser aguardada.

Estando o posicionamento correto, é preciso agora transferir os dados, ou seja, gravar o bloco no disco ou recuperá-lo. O tempo gasto na transferência em si é chamado **latência de transferência** e depende do número de bytes transferidos, da quantidade de bytes contidos na trilha e da velocidade de rotação do disco.

1.5 - Considerações finais

Nesta primeira unidade foram vistos vários conceitos importantes:

1. Representação da informação
2. Características da memória principal
3. Características da memória secundária e acesso a discos rígidos

Uma questão essencial é lembrar que, para representar uma informação útil, é preciso um código. De posse do código, é possível avaliar um conjunto de símbolos e interpretá-lo para obter a informação representada. Nos computadores, os bits representam os dados básicos e são organizados em bytes. Sabendo-se o que um dado conjunto de bytes armazena, torna-se viável analisar o padrão de bits que ele possui e interpretar a informação nele contida.

A memória principal é uma coleção finita de bytes e é usada para armazenar as diversas informações. Tal tipo de armazenamento é rápido, porém bastante limitado em tamanho. O custo relativo deste tipo de memória é um fator restritivo para seu tamanho.

Memória secundária é um termo que se associa a formas de armazenamento de grande capacidade, porém com acesso muito mais lento que aquele à memória principal. Discos rígidos são exemplos de dispositivos de armazenamento secundário, cujas características mecânicas retardam o acesso a um dado conjunto de bytes.

O sistema operacional ou um SGBD são responsáveis por organizar os dados dos discos rígidos e definem, para se ter acesso, uma estruturação baseada em blocos. Sendo os blocos as unidades de transferência de dados, entender como todo o processo de gravação ou leitura de dados ocorre permite projetar estratégias inteligentes para se obter desempenho de acesso à memória secundária.