Unidade 1: Introdução à Organização de Computadores

Prof. Daniel Caetano

Objetivo: Apresentar o que é um computador e o que são dados, a evolução histórica dos computadores e os diferentes componentes que constituem um computador.

Bibliografia:

- MONTEIRO, M.A. **Introdução à Organização de Computadores**. 5ª. Ed. Rio de Janeiro: LTC, 2008.
- MURDOCCA, M. J; HEURING, V.P. **Introdução à Arquitetura de Computadores**. S.I.: Ed. Campus, 2000.
- TANENBAUM, A. S. **Sistemas Operacionais Modernos**. 2ª.Ed. São Paulo: Prentice Hall, 2003.

INTRODUÇÃO

Atualmente é bastante incomum encontrar pessoas que jamais tenham visto ou operado um computador. Entretanto, para a maioria das pessoas o computador não passa de uma "caixa preta", isto é, um aparato "mágico" que produz resultados interessantes. O objetivo do curso de Arquitetura e Organização de Computadores é desvendar essa "caixa preta", apresentando os fundamentos que tornam possível seu funcionamento.

A compreensão dos computadores modernos sem conhecer a sua origem, entretanto, é muito mais árdua. Assim, esta primeira aula aborda a origem dos computadores e sua evolução, além de dar uma visão geral de toda a transição que ocorre entre aquilo que escrevemos na forma de um programa - o código - e aquilo que o computador realmente entende - sinais elétricos.

Finalmente, é apresentada uma breve introdução aos sistemas de numeração mais utilizados na informática, desvendando as razões pelas quais eles são utilizados.

1. CONCEITOS DE PROCESSAMENTO DE DADOS

O primeiro conceito que iremos abordar é o conceito de **computador**. O termo “computador” surgiu como o nome de uma profissão: eram os profissionais que realizavam cálculos complexos para aplicações militares como lançamento de mísseis. Posteriormente essa atividade passou a ser executada por equipamentos que acabaram herdando esse nome: computadores.

Resumidamente, um computador é uma máquina capaz de coletar dados, processar e gerar resultados a partir dos dados coletados e de uma maneira sistemática.

Essa definição, no entando, nos leva a uma outra questão: o que é um **dado**?

Um dado é, em essência, um valor que tem algum significado. Por exemplo: 10 é apenas um número, mas se dissermos que esse 10 é uma idade, em anos, ele pode ser chamado de dado. Por outro lado, nós é que precisamos saber o que esse número significa; o computador, nem sempre.

Mas... se dados são valores, o que seria o **processamento**? Processar, no caso dos computadores, é o ato de realizar operações com os dados. Por exemplo: somar o valor 2 com o valor 3 é um processamento, cujo resultado é o valor 5.

Uma dúvida frequente é como diferenciar dado de informação. Um dado é, como já dito, um valor com significado; a informação, porém, é um dado examinado dentro de um contexto. “Idade: 10 anos” é apenas um dado. “Minha idade é 10 anos” é uma informação, porque não apenas transmitimos um valor, mas associamos esse valor a uma pessoa, o que permite inferir que essa pessoa nasceu há 10 anos. Contextualizamos a idade: idade de quê/quem?

Em geral os computadores recebem dados, mas os organizam e processam de maneira contextualizada, produzindo informações como resultado.

Para que o processamento dos dados possa ser realizado, contamos com dois componentes principais: o **hardware** e o **software**.

O nome “hardware” é dado aos componentes físicos de um computador, como processador, memória, gabinete etc. O nome “software”, por outro lado, são conjuntos de instruções e dados que coordenam o funcionamento do hardware; o software é como se fosse uma “receita” que o computador segue para funcionar.

Quando pensamos em “hardware”, pensamos em notebook, tablet, telefone. Quando pensamos em “software”, pensamos em Linux, Windows, Office, iTunes.

Todos estamos acostumados a esses equipamentos e seus programas... mas será que os computadores sempre foram assim?

2. EVOLUÇÃO DOS COMPUTADORES

A evolução dos equipamentos conhecidos hoje como "computadores" pode ser dividida duas grandes etapas: uma, inicial, envolve os equipamentos mecânicos e eletro-mecânicos. A segunda, mais recente, envolve os equipamentos eletrônicos e de estado sólido. Esta segunda etapa apresentou tantas transformações que acabou sendo dividida em diversas fases ou gerações.

2.1. Equipamentos Mecânicos e Eletro-Mecânicos

Esse universo eletrônico que hoje conhecemos como "mundo da informática" teve seu início em épocas bem mais precárias, há muitos séculos.

Tudo começou com os babilônios, por volta de 500a.C., com a invenção do ábaco, que era uma ferramenta de cálculo manual. A primeira evolução do ábaco só veio em 1642, com a invenção da Pascalene, pelo físico e matemático Blaise Pascal. A Pascalene era um equipamento mecânico capaz apenas de realizar somas e subtrações. A evolução destes dispositivos foi muito lenta e eles eram pouco usados, devido ao uso limitado e desajeitado.

Foi apenas com a invenção do motor elétrico, já no fim do século XIX, foi possível construir máquinas mecânicas muito mais complexas e "rápidas". Uma das primeiras máquinas deste tipo foi usada com o propósito de realizar a contabilização do censo dos Estados Unidos da América. Esta máquina foi projetada por Hermann Hollerith, fundador da IBM e também criador da máquina que realizava o cálculo do pagamento dos funcionários, produzindo um pequeno resumo de contabilidade que recebeu o seu nome, sendo chamado até hoje de "Olerite".

2.2. Equipamentos Eletrônicos

Como os equipamentos existentes até o fim do seculo XIX não eram adequados para resolver grandes problemas, todos os problemas mais complexos precisavam ser solucionados por seres humanos.

Os militares, por exemplo, que frequentemente precisavam de soluções sistematizadas para problemas complexos - fosse para distribuir produtos ou para construir edificações -, usavam profissionais específicos da área de cálculo e lógica, que compunham uma espécie de linha de produção de soluções de problemas. Haviam os profissionais que, em tempos de guerra, estabeleciam a lógica de solução para os problemas de distribuição de armas e suprimentos; a atividade exercida por eles ficou conhecida como "Logística". Entretanto, os "logísticos" só descreviam os procedimentos de solução, que precisavam ter seus resultados computados para que pudessem ser postos em prática. Os profissionais que realizavam os cálculos eram chamados de "Computadores".

Esta organização era suficiente para todas as necessidades até que, na segunda guerra mundial, os engenheiros alemães criaram máquinas complexas de criptografia de dados, chamadas "Enigma". As forças militares aliadas, incluindo o exército norte-americano, eram capazes de captar as mensagens transmitidas pelos alemães mas, como estas estavam codificadas, não era possível compreendê-las. Percebendo que era fundamental decifrar tais mensagens para a vitória e, verificando que a decodificação pelos computadores humanos era impossível, engenheiros foram chamados para que fossem propostas máquinas capazes de decifrar as mensagens codificadas pelo Enigma.

Das pesquisas nesta área, deu-se início aos primeiros equipamentos eletrônicos de computação, substitutos dos computadores humanos.

2.2.1. Primeira Fase

A primeira fase (ou primeira geração) dos computadores ocorreu aproximadamente durante o período entre 1940 e 1955, quando surgiram os primeiros computadores digitais usados durante a segunda guerra mundial.

O ENIAC (*Eletronic Numerical Integrator and Computer*) foi o primeiro computador de propósito geral, desenvolvido para cálculo de Balística. Era gigantesco e tinha uma capacidade de processamento em torno de 5000 adições por segundo, valor este muito inferior ao de uma calculadora atual da Hewlett-Packard. Outro computador desenvolvido à mesma época foi o Colossus, este de propósito específico: decodificar as mensagens criadas pela máquina Enigma (e posteriormente Enigma 2) dos Alemães.

Estes primeiros computadores eram baseados em válvulas e programados com o método chamado *wire-up*, isto é, conectando fisicamente, com fios, diversos polos, fechando bits de forma a construir um programa. A programação era, assim, feita diretamente em linguagem de máquina e o equipamento não oferecia qualquer recurso para facilitar este trabalho de programação.

Outros grandes computadores construídos na época incluem o EDVAC, o IAS e o UNIVAC I, construído pela IBM para a computação do censo americano de 1950.

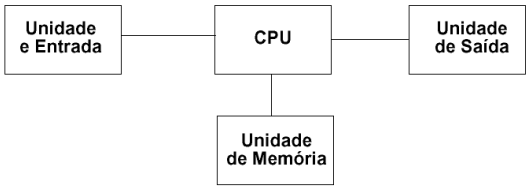
2.3. Equipamentos de Estado Sólido

Os equipamentos eletrônicos produzidos na primeira fase possuíam diversas limitações. Para começar, consumiam uma quantidade monstruosa de energia. Adicionalmente, eram enormes, esquentavam demasiadamente, demoravam horas para poderem ser usados a partir do momento em que eram ligados e, para finalizar, alguns cálculos tinham de ser repetidos diversas vezes, pois a confiabilidade dos resultados não era exatamente alta.

Uma mudança radical neste cenário só foi possível com a criação dos transístores, que permitiram a eliminação das válvulas e, com elas, a maior parte dos transtornos mencionados acima. Entretanto, o transístor, por si só, não eliminava uma das principais limitações destes equipamentos: a programação física através de fios. Como a memória dos computadores eletrônicos era muito pequena, apenas os dados do processamento eram armazenados nela.

Com a possibilidade de memórias maiores - seja pelo uso de transístores ou pelas novas "fitas magnéticas", este problema foi resolvido com o conceito de *software*, isto é, um programa armazenado em memória, conforme descrito por John Von Neumann.

Na proposta de Von Neumann, todo equipamento de computação deveria possuir quatro componentes: memória, unidade de processamento, dispositivos de entrada e dispositivos de saída, conforme apresentado na figura a seguir. Adicionalmente, um certo conjunto de dados armazenado na memória deve ser interpretado pela CPU como *instruções*, eliminando a necessidade de fios para a programação do equipamento.



2.3.1. Segunda Fase

A segunda fase ocorreu aproximadamente entre 1955 e 1965, e a grande inovação era o uso de transístores, o que permitiu uma grande redução no tamanho dos equipamentos e aumento de sua velocidade, além do aumento da **confiabilidade** do processamento. Também é desta época o surgimento das memórias magnéticas, que permitiram um aumento na capacidade e velocidade do armazenamento.

Nesta fase surgiram as primeiras linguagens e compiladores e, pela primeira vez, surgiu o conceito de sistema operacional como um software para automatizar todas as tarefas repetitivas utilizadas por diversos softwares e que, até então, eram realizadas manualmente (processamento *batch*). Originalmente, sempre que se desejasse executar um programa, o programador deveria inserir este programa no equipamento (através de um cartão perfurado), este programa seria executado e finalmente o resultado seria impresso. Entretanto, em geral este processamento durava horas e era comum que se passassem horas até alguém perceber que o processamento havia finalizado. Isso fazia com que o equipamento ficasse ocioso mesmo que muitas tarefas ainda estivessem por ser executadas, já que o equipamento dependia que um ser humano fosse até ele e o alimentasse com um novo programa.

Nesta geração, então, passou a ser possível introduzir diversos programas e o "sistema operacional" existente era capaz de executá-los em seqüência, ou seja, assim que um terminava, ele iniciava o seguinte e assim por diante, eliminando o problema da ociosidade. Ainda no final desta fase a IBM criou o conceito de "canal", hoje comumente chamado de "DMA", que permitia a escrita direta entre dispositivos sem a necessidade de intervenção da CPU.

2.3.2. Terceira Fase

Esta fase ocorreu mais ou menos no período de 1965 a 1980, e foi marcada pela utilização dos circuitos integrados, que reduziram tamanho e custos e ampliaram enormemente a capacidade de armazenamento, processamento e confiabilidade dos computadores. Nesta época surgiu o conceito de família de processadores (IBM Série 360), em que vários equipamentos, com dispositivos diferentes conectados, eram compatíveis entre si. Também surgiram computadores de custo menor, como o PDP-8 da DEC.

Com o grande aumento de recursos destes equipamentos, os novos sistemas operacionais (como o OS/360) traziam novas possibilidades de gerência de processamento, permitindo que enquanto quando um programa esperava pela entrada de dados do usuário, outro fosse processado. Esta tecnologia ficou conhecida como *multiprogramação* e é uma técnica básica envolvida na *multitarefa cooperativa*. Nesta geração também passou a existir uma "interação online"; foram criados os primeiros terminais de vídeo e teclados para comunicação com o software *durante sua execução*.

Ainda nesta geração, a multiprogramação evoluiu de maneira a melhorar os tempos de resposta na interação com os usuários, desemvolvendo o conceito de *time-sharing*, isto é, cada processo compartilha a CPU por um intervalo de tempo. Este conceito é a base da *multitarefa preemptiva*.

Surgiu nesta fase, ainda, o sistema operacional UNIX, concebido inicialmente para o computador PDP-7, desenvolvido em Linguagem C, e tornou-se bastante conhecido por sua portabilidade. Outras grandes novidades desta época foram os computadores de 8 bits da Apple e o sistema operacional CP/M (Control Program Monitor).

Vale ressaltar que, nesta geração, houve a criação do padrão POSIX (Portable Operating System IX), que definiu uma interface mínima que sistemas UNIX devem suportar.

2.3.3. Quarta Fase

Nesta fase, que durou toda a década de 1980, a integração em larga escala (LSI e VLSI) permitiram uma redução substancial no tamanho e no preço dos equipamentos. Com isso houve o surgimento de diversos computadores menores mas muito potentes (variando desde os mais simples como o MSX até os mais poderosos IBM-PCs), ambiente no qual surgiu o DOS (Disk Operating System), base dos "computadores pessoais" do período. Estes equipamentos tinham processamento relativamente limitado e, portanto, o DOS não suportava muitas das características de multiprogramação, *time-sharing* e outros.

No campo dos computadores de grande porte, surgiu o sistema VMS (Virtual Machine System) que, implementando todos os recursos concebidos até então, criou oficialmente o conceito de *multitarefa* em um sistema monousuário.

Nesta fase surgiram os computadores capazes de *multiprocessamento*, com várias CPUs em paralelo, e os primeiros sistemas capazes de lidar com este tipo de característica também surgiram. Nesta fase houve proliferação das LANs e WANs, com o surgimento de diversos protocolos de comunicação e uma grande aceitação do protocolo TCP/IP.

Alguns autores (como Tanenbaum, 2003) não consideram a quinta e sextas fases, colocando os avanços posteriores ao da quarta fase dentro da própria quarta fase. Por questões didáticas, neste trabalho foi feita a opção pela separação.

2.3.4. Quinta Fase

A quinta fase compreendeu basicamente a década de 1990, sendo a tônica principal o aumento da capacidade de processamento e armazenamento em proporções não previstas anteriormente, possibilitando aplicação de inteligência artificial, bancos de dados e mutlimídia em praticamente qualquer tipo de aplicação, tornando-as muito mais complexas.

Nesta fase os computadores se tornaram muito baratos e passaram a fazer parte da vida de praticamente todas as pessoas. O conceito de *processamento distribuído* passou a fazer parte das pesquisas e a *multitarefa* veio para os computadores pessoais.

2.3.5. Sexta Fase

A sexta fase teve início juntamente com o século XXI e ainda não foi finalizada. As inovações trazidas são conhecidas pela maioria das pessoas, como a ubiquidade do acesso à

rede, com redes sem fio e internet, com um aparente limite físico estabelecido da capacidade de processamento de uma unidade central de processamento e o *multiprocessamento* chegando aos computadores pessoais a baixos preços.

A quantidade de memória e velocidade de comunicação das redes permitem que grandes massas de dados sejam processadas e transmitidas, possibilitando video-conferências a um baixo custo. O processamento distribuído tornou-se uma realidade comum, embora ainda explorada apenas por aplicações científicas. A computação móvel tornou-se uma realidade, com a proliferação dos laptops e palmtops, levando os recursos computacionais a qualquer lugar.

3. NÍVEIS DE MAQUINA

Na seção anterior foi possível verificar como os equipamentos foram evoluindo ao longo do tempo. Entretanto, a partir das máquinas de estado sólido, estabeleceu-se um paradigma de organização que facilita o estudo destes equipamentos.

Cada um dos níveis desta organização é denominado "nível de máquina" e, dentro de certos limites, cada um destes níveis de abstração pode ser estudado independentemente. Nestes termos, podem ser definidos 7 níveis de uma máquina, do mais alto para o mais baixo:

1. Programas Aplicativos
2. Linguagens de Alto Nível
3. Linguagem Assembly / de Máquina
4. Controle Microprogramado
5. Unidades Funcionais
6. Portas Lógicas
7. Transistores e Fios

Esta "independência" é que permite, na prática, que o usuário de um software qualquer não precise conhecer programação e que um programador não precise entender de eletrônica e portas lógicas, ficando esta tarefa apenas para os Engenheiros Eletrônicos.

A compreensão destes níveis é importante para que sejam compreendidos os diferentes níveis de compatibilidade que podem existir entre dois equipamentos.

Programas Aplicativos: é o nível com que, obviamente, o usuário de computador está mais familiarizado. É neste nível que o usuário interage com o computador, usando um programa (como jogos, editores gráficos ou de texto). Neste nível, quase nada (ou nada mesmo) da arquitetura interna é visível. Neste nível, existe a compatibilidade de "usabilidade", do tipo que você espera ao executar um programa como Microsoft Office ou Firefox independente de estar executando em um PC ou Mac.

Linguagens de Alto Nível: é o nível com que lidam os programadores de linguagens como C/C++, Pascal, Java etc. O programador lida com todos os detalhes de instruções e tipos de dados da linguagem que não necessariamente têm a ver com as instruções e tipos de dados da linguagem de máquina. É interessante citar a exceção do C/C++, onde algumas vezes o programador é obrigado a lidar com características especiais da linguagem de

máquina. Por esta razão, o C/C++ às vezes é chamado, informalmente, de "a única linguagem de médio nível". Neste nível temos a chamada "compatibilidade de código fonte", em que um código escrito da maneira correta pode ser *compilado* para "qualquer" processador (ou CPU) e funcionar normalmente.

Linguagem Assembly / de Máquina: enquanto uma linguagem considerada de alto nível tem pouco a ver (ou nada a ver) com as instruções e estruturas de dados típicas de uma dada CPU, a linguagem de máquina (de baixo nível) é exatamente a linguagem desta CPU, com instruções próprias e tipos de dados intimamente ligados à forma como a CPU funciona. Estas instruções de uma CPU são chamadas de **conjunto de instruções** da máquina. Para programar neste nível, o programador precisa conhecer muito bem toda a arquitetura da máquina e também seu conjunto de instruções. Quando máquinas são compatíveis neste nível - ainda que o circuito seja completamente diferente de uma para outra, é dito que elas têm **compatibilidade binária**, pois uma é capaz de executar códigos de máquina da outra. A compatibilidade entre os diversos processadores Intel x86 e "compatíveis" vem até este nível.

Nos computadores digitais, a linguagem de máquina é composta por instruções binárias (longas seqüências de zeros e uns), também chamado de **código de máquina binário**. Entretanto, nenhum programdor com um mínimo de recursos disponíveis trabalha com tais códigos, por ser um trabalho extremamente tedioso e sujeito a erros de digitação. Ao trabalhar com programação de baixo nível é comum o uso de **montadores** (*assemblers*), que foram, certamente, um dos primeiros tipos de software escritos. Estes montadores permitem que usemos palavras chamadas **mnemônicos** para expressar instruções da CPU (LOAD, MOVE, JUMP etc.) e o trabalho destes montadores é justamente o de traduzir estes mnemônicos para códigos de máquina. O conjunto de mnemônicos cujas construções têm relação direta de um para um com a linguagem de máquina é chamada **linguagem de montagem** (linguagem *assembly*).

Controle Microprogramado: é o nível que faz a interface entre a linguagem de máquina (código de máquina) e os circuitos que realmente efetuam as operações, interpretando instrução por instrução, executando-as uma a uma. Nos processadores "compatíveis com x86", incluindo os da própria Intel, é nessa camada que é feita a "mágica" da compatibilidade. Este nível foi "criado" pela IBM com a série de computadores **IBM 360**, em meados da década de 1960. Existem duas formas de fazer a microprogramação: uma delas é através de circuitos lógicos (*hardwired*), o que é extremamente eficiente e rápido, mas de projeto bastante complexo. Uma outra solução é através do microprograma, que nada mais é que um pequeno programa escrito em uma linguagem de ainda mais baixo nível executado por um **microcontrolador**. Este microprograma é também chamado de **firmware**, sendo parte hardware e parte software.

Unidades Funcionais: a grande maioria das operações da Unidade de Controle (parte da CPU) são exatamente para mover dados para dentro e para fora das "unidades funcionais". Estas unidades têm esse nome porque executam alguma tarefa importante para o funcionamento da máquina e, dentre elas, temos os registradores da CPU (memórias internas da CPU que possuem um nome específico), a ULA (que realiza as contas de fato) e a memória principal.

Portas Lógicas, Transistores e Fios: este é o nível mais baixo que ainda remete ao funcionamento de mais alto nível. As unidades funcionais são compostas de **portas lógicas**, que por sua vez são compostas de **transistores** interconectados. Abaixo deste nível existem

apenas detalhes de implementação de circuitos (como níveis de voltagem, atrasos de sinal etc).

3. ESTRUTURA DE UM COMPUTADOR MODERNO

Um computador moderno segue um modelo descrito em 5 componentes:

Unidade Central de Processamento (UCP ou CPU, Central Processing Unit): responsável por interpretar os programas e executar operações matemáticas.

Memória: responsável por armazenar os programas, dados de entrada e resultados dos processamentos.

Dispositivos de Entrada: permitem que o usuário forneça dados para o computador.

Dispositivos de Saída: permitem que o usuário obtenha informações geradas pelo computador.

Barramento: conjunto de fios que interligam todos os componentes anteriores.

4. BIBLIOGRAFIA

MONTEIRO, M.A. **Introdução à Organização de Computadores**. 5ª. Ed. Rio de Janeiro: LTC, 2008.

MURDOCCA, M. J; HEURING, V.P. **Introdução à Arquitetura de Computadores**. S.I.: Ed. Campus, 2000.

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. 2ª.Ed. São Paulo: Prentice Hall, 2003.

Unidade 2: Fundamentos da Programação de Computadores

Prof. Daniel Caetano

Objetivo: Compreender os diferentes tipos de linguagem de programação, como as linguagens de programação de alto nível são traduzidas para o computador e, finalmente, conhecer as diferentes unidades de daods/informações usadas no computador.

Bibliografia:

- MONTEIRO, M.A. **Introdução à Organização de Computadores**. 5ª. Ed. Rio de Janeiro: LTC, 2008.
- MURDOCCA, M. J; HEURING, V.P. **Introdução à Arquitetura de Computadores**. S.I.: Ed. Campus, 2000.
- TANENBAUM, A. S. **Sistemas Operacionais Modernos**. 2ª.Ed. São Paulo: Prentice Hall, 2003.

INTRODUÇÃO

Vimos que todo o processamento de dados realizado por um computador segue uma "receita" chamada "**programa**". Um programa é uma implementação, na linguagem do computador, de um "**algoritmo**".

Poderíamos escrever nossos programas diretamente na linguagem do computador mas essa tarefa costuma ser extremamente tediosa, porque tudo teria que ser escrito em "0s" e "1s", que é a linguagem **binária** que o computador entende.

Sendo assim, costumamos programar em linguagens chamadas de "alto nível", muito parecidas com a nossa própria língua, que posteriormente é convertida para a linguagem binária dos computadores com a ajuda de alguns programas, chamados **compiladores**.

Todos esses tópicos serão abordados com mais detalhes nas próximas seções.

1. PROGRAMAÇÃO E OS ALGORITMOS

O ser humano sempre se deparou com "problemas", isto é, situações indesejadas que exigem intervenção para que uma situação desejável seja atingida. A intervenção a ser realizada depende do problema, assim como do que se considera um resultado desejável.

Frequentemente essa intervenção é uma sequência de tarefas que transformam a situação. À uma sequência de passos que visa atingir a um objetivo definido dá-se o nome de **algoritmo**.

Um exemplo clássico são as receitas de culinária:

Algoritmo 1 - Fazer um Omelete:
Passo 1: Em um prato fundo, bata 3 ovos.
Passo 2: Acrescente sal.
Passo 3: Acrescente cheiro-verde.
Passo 4: Bata mais um pouco.
Passo 5: Leve ao fogo médio em frigideira untada com manteiga.
Passo 6: Depois de dourar um lado, vire e deixe dourar o outro lado.

Algoritmo 2 - Fazer um Misto Quente
Passo 1: Pegar o presunto
Passo 2: Grelhar o presunto
Passo 3: Colocar o queijo sobre o presunto
Passo 4: Pegar duas fatias de pão de forma
Passo 5: Colocar uma fatia de pão sobre o queijo.
Passo 6: Virar e colocar a outra fatia de pão.
Passo 7: Deixe dourar ambos os lados.

Entretanto, um algoritmo precisa ser escrito em um formato, uma linguagem e com instruções que seu executor conheça e seja capaz de executar. Por exemplo: de nada adianta dar uma ordem para que uma geladeira faça café, se ela não for equipada para isso; ou pedir a um advogado que projete uma usina hidrelétrica. Da mesma forma, dar ordens em português para uma pessoa que só compreende russo não trará bons resultados.

Sendo assim, quando escrevemos um algoritmo em passos que um computador seja capaz de executar e em uma linguagem que ele seja capaz de compreender, esse algoritmo recebe o nome de **programa**.

Mas quais tarefas o computador sabe executar? E qual linguagem ele compreende?

A primeira pergunta é fácil de responder: de maneira geral, ele sabe fazer cálculos, decidir se executa ou não uma tarefa com base em algum valor, receber e armazenar dados do usuário e, finalmente, informar resultados ao usuário.

Quanto a linguagem, será necessária uma discussão mais aprofundada.

2. LINGUAGEM DE MÁQUINA E DE MONTAGEM

Assim como os seres humanos possuem sua linguagem - palavras e comandos que compreendem -, cada computador tem também a sua própria linguagem. Nós somos capazes de nos comunicar pelo som e pela escrita, gerando os mais diversos ruídos e símbolos para nos comunicarmos.

Os computadores, entretanto, são dispositivos eletrônicos, que se comunicam por meio de **fios**. Os fios, grosso modo, só permitem dois tipos de interações entre dispositivos eletrônicos: ou existe uma corrente elétrica entre eles... ou não existe essa corrente elétrica. Convencionou-se chamar o estado de passagem de corrente elétrica como “1” e o estado de não-passagem de corrente elétrica como “0”. Em outras palavras, um fio pode representar um valor “0” ou “1”, de acordo com seu estado de corrente. À quantidade de informação que

pode ser representada por um fio desse tipo dá-se o nome de **bit**, a menor unidade de informação em um computador.

Como um único fio acaba propiciando uma comunicação muito limitada, é comum usar vários fios para comunicação, cada um deles representando um bit, compondo grupos de vários bits chamados **números binários**.

A linguagem do computador - ou a linguagem de máquina - é uma linguagem em que esses números binários representam instruções; por exemplo, para armazenar o valor 10 na "variável" A, em um determinado processador chamado Z80, usamos a seguinte sequência de bits:

00111110 00001010

Obviamante escrever programas assim não é nem um pouco prático e não demorou até que alguém criasse um "nome" e uma "grafia" mais natural para os humanos para um comando desse tipo:

LOAD A,10

Muito mais fácil de ler, correto?

Sem dúvida, mas um programa escrito assim precisava antes ser "traduzido": uma enorme tabela continua todos os comandos nessa linguagem "mnemônica" (isso é, mais fácil de guardar) e sua versão em binário. O programa era contruído (ou montado) nessa linguagem mnemônica e depois era traduzido manualmente no momento de digitar no computador. Assim, essa linguagem mnemônica ficou conhecida como "lingaugem de montagem" (linguagem **assembly**).

Não demorou muito para que alguém construísse um programa que fosse capaz de ler a linguagem de montagem e gerasse o código binário para o computador; esses programas passaram a ser conhecidos como "montadores" ou **assemblers**. Esse primeiro passo foi muito importante e praticamente foi o início da atividade que se conhece hoje como "programação".

O assembler, entretanto, tinha um problema: apesar das instruções serem mais facilmente compreendidas pelo ser humano, elas eram simplesmente um "símbolo" diferente para especificar as limitadas instruções do computador. E, para piorar, cada computador/CPU possuia uma linguagem assembly própria, o que significava que um programa escrito para um computador não serviria em outro, de outro fabricante.

E com a "popularização" inicial da programação, os programadores passaram a desejar poder escrever programas de maneira mais parecida com que faziam nas deduções matemáticas: LOAD A,10 deveria ser escrito como **A = 10** e coisas mais complicadas deviam ser possíveis: **A = 10 * 17 / X**. Além disso, o ideal seria que um programa escrito deveria poder ser executado em vários computadores, de modelos diferentes.

Foi essa a motivação para a criação das **linguagens de programação de alto nível**.

3. LINGUAGENS DE PROGRAMAÇÃO DE ALTO NÍVEL

A linguagem assembly é denominada linguagem de baixo nível porque, para programar nela, o programador precisa "descer ao nível do computador", ou seja, construir um programa com ordens muito simples, que a CPU consiga compreender diretamente.

A ideia das linguagens de alto nível é que o programador possa construir programas em uma linguagem mais parecida com a usada por ele em seu dia-a-dia, sem se preocupar com as instruções que o computador compreende ou não.

Sendo assim, foram criadas linguagens como FORTRAN, COBOL, LISP, C/C++, BASIC, PASCAL, Java... E tantas outras. Cada uma delas com um propósito específico de facilitar a criação de um tipo específico de programa.

Essas linguagens são, na verdade, um "meio termo" entre a forma como os humanos se expressam e o que os computadores entendem. Elas exigem um pouco menos de esforço do ser humano quando comparado ao esforço necessário para escrever em assembly, mas ainda assim **são absolutamente incompreensíveis para o computador**.

Para que o computador seja capaz de compreender tais linguagens é necessária uma **tradução**, ou seja, é necessário que "alguém" leia o código na linguagem de alto nível e escreva um novo código em linguagem de máquina, para que o computador saiba executá-lo. Esse "alguém" que faz a tradução é um programa de computador denominado **compilador**.

Assim, o compilador é um programa que lê um código escrito em "linguagem de alto nível" e, com base nele, cria um programa em linguagem de máquina, chamado código objeto.

Em sistemas mais complexos, os programas tendem a ser compostos por várias partes menores que precisam ser conectadas para que possam funcionar. Esse trabalho final, que transforma o código objeto em um **arquivo executável** é chamado de **linker**.

4. UNIDADES DA INFORMAÇÃO

A quantidade de dados gerada e processada por um computador é, em geral, muito maior que aquela que podemos armazenar em 1 bit. Como vimos, mesmo as instruções do computador são organizadas em conjuntos de bits, sendo que alguns deles recebem nomes especiais. Os grupos mais comumente usados são:

Byte: um grupo de 8 bits, suficiente para armazenar os símbolos mais comuns da comunicação ocidental baseada em caracteres romanos.
Word: um grupo de 2 bytes (ou 16 bits). Em geral usado para armazenar números inteiros.

Os múltiplos são o kilo (K), o mega (M), o giga (G), o tera (T) e o peta (P) mas, diferentemente do sistema decimal, em que cada unidade é uma potência de 10, no sistema binário usamos potências de 2 para indicar essas magnitudes:

1 KB = 2^{10} bytes = 1024 bytes
1 MB = 2^{20} bytes = 1024 KB = 1048576 bytes
1 GB = 2^{30} bytes = 1024 MB = 1073741824 bytes
1 TB = 2^{40} bytes = 1024 GB
1 PB = 2^{50} bytes = 1024 TB

5. BIBLIOGRAFIA

MONTEIRO, M.A. **Introdução à Organização de Computadores**. 5ª. Ed. Rio de Janeiro: LTC, 2008.

MURDOCCA, M. J; HEURING, V.P. **Introdução à Arquitetura de Computadores**. S.I.: Ed. Campus, 2000.

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. 2ª.Ed. São Paulo: Prentice Hall, 2003.

Unidade 3: Sistemas de Numeração

Numerais Binários e Bases de Potência de Dois

Prof. Daniel Caetano

Objetivo: Apresentar as diferentes bases numéricas, preparando o aluno para compreender e trabalhar com lógica binária.

Bibliografia STALLINGS, 2003; MURDOCCA e HEURING, 2000.

INTRODUÇÃO

Como discutido na aula anterior, o computador trabalha com sinais elétricos do tipo "ligado" e "desligado" e, como uma consequência disso, quando usamos sinais em fios para representar números, torna-se mais natural uma representação em potências de dois.

Sendo assim, a capacidade de leitura/especificação de valores nas mais diversas bases de numeração torna-se fundamental a todo profissional que almeje trabalhar com lógica digital e a construção/manutenção de equipamentos baseados nesta.

Para propiciar familiaridade dos alunos com estas diferentes bases numércias, este será o assunto desta e das próximas aulas, antes de estudarmos o funcionamento interno dos equipamentos computacionais.

1. REPRESENTAÇÕES NUMÉRICAS

Primeiramente, é importante diferenciar o que são números do que são quantidades. A quantidade de elementos em um conjuto é um conceito abstrato oriundo da contagem dos elementos. É possível comparar quantidades - isto é, dizer se um conjunto é maior que outro - independentemente de existir um nome para essa quantidade. Por exemplo: os dois conjuntos abaixo possuem diferentes quantidades de bolinhas:

Conjunto 1	**Conjunto 2**
o o o o o	o o o o o o o o o o o o o o o

Os números são representações simbólicas convenientes para quantidades. Por exemplo: o Conjunto 1 tem 5 bolinhas e o Conjunto 2 tem 15 bolinhas. Entretanto, será que essa é a única forma de representar quantidades?

Na verdade, esta não apenas não é a única como também não foi a primeira. Esta forma de representação numérica é chamada "representação decimal com numerais indu-arábicos". Uma outra forma tradicional de representar os números é através dos

numerais romanos, que não seguem uma base numércia tradicional usando letras como I, L, C, X, M e V para representar as quantidades. Segundo a representação romana, o primeiro conjunto tem V bolinhas e o seguindo tem XV bolinhas.

Existem outras representações "não-decimais" usando númerais indú-arábicos, como a binária e a octal. Existem aquelas que usam caracteres alfanuméricos para representar as quantidades, como a hexadecimal. A tabela a seguir mostra a contagem de 0 a 15 representada em diferentes formas.

Decimal	Romana	Binária	Octal	Hexadecimal
0	-	0	0	0
1	I	1	1	1
2	II	10	2	2
3	III	11	3	3
4	IV	100	4	4
5	V	101	5	5
6	VI	110	6	6
7	VII	111	7	7
8	VIII	1000	10	8
9	IX	1001	11	9
10	X	1010	12	A
11	XI	1011	13	B
12	XII	1100	14	C
13	XIII	1101	15	D
14	XIV	1110	16	E
15	XV	1111	17	F

Observe que, em cada linha, temos diferentes representações para uma mesma quantidade!

Qual a razão para essa "confusão" toda? Bem, algumas representações, como a romana, são muito antigas e, posteriormente, foram substituídas na maioria dos usos pela numeração indú-arábica decimal. A numeração decimal, por sua vez, parece ser a mais lógica para nós, já que somos capazes de contar com as mãos até 10 elementos.

Entretanto, em alguns casos - como no caso dos computadores, temos de representar as quantidades usando apenas **fios**, pelos quais pode (ou não) passar uma corrente. Nestes casos, somos obrigados a usar vários fios para representar um número, e ele acaba sendo, obrigatoriamente, representado na forma binária.

Na ***eletrônica digital***, cada fio/conexão devem indicar apenas um de dois valores: ligado (com tensão) ou desligado (sem tensão). Essa decisão tem um impacto bastante relevante na forma com que representamos as informações em um computador: considerando que o fio é a "mão" do computador, ele considera apenas dois dígitos: 0 e 1 - diferentemente de nós, que consideramos os dígitos de 0 a 9!

Ocorre que, como a representação biária é um tanto desajeitada - devido ao grande número de dígitos - frequentemente usamos uma representação equivalente, de fácil conversão para o binário: o hexadecimal. Observe que, na representação hexadecimal, o segundo dígito só será necessário quando a representação binária tiver 5 dígitos (não aparece nessa tabela). É um considerável ganho para economia de escrita e, adicionalmente, reduz a probabilidade de erros de digitação - bastante alta quando os números envolvem apenas longas sequências de zeros e uns.

Como pode haver confusão se usarmos diferentes representações numéricas em um mesmo texto, usa-se a seguinte convenção:

Números Decimais: são escritos normalmentem, SEM zero à esquerda.
Exemplos: 5, 30, 44.
Números Binários: são escritos com o acréscimo de um "b" ao final ou com índice 2.
Exemplos: 101b, 11110b, 101100b,101100_2.
Números Hexadecimais: são escritos com o acréscimo de um "h" ao final ou com índice 16.
Exemplos: 5h, 1Eh, 2Ch, $2C_{16}$

Nota: uma forma alternativa de representar os números hexadecimais é usada nas linguagens C/C++, Java e outras. Nestas, ao invés de se acrescentar o "h" ao final, acrescenta-se o prefixo "0x" no início. Exemplos: 0x5, 0x1E, 0x2C.

2. NOTAÇÃO POSICIONAL

Um grande avanço da notação indú-arábica decimal com relação à romana é o uso de notação posicional. A notação posicional significa que a quantidade que um número representa depende da posição em que ele aparece na reprentação completa. Por exemplo: qual a quantidade representada pelo símbolo "1"? Se você respondeu "1, oras!", errou! O símbolo 1 tem diferentes significados, de acordo com a posição no número!

Consideremos os números decimais. Neste caso, se o 1 estiver na primeira casa, ele vale **uma unidade**. Se estiver na segunda casa, ele vale **uma dezena**. Se estiver na terceira casa, ele vale **uma centena**... na quarta vale **uma unidade de milhar** e assim por diante. Observe:

1 : Um
10 : Dez
100 : Cem
1000 : Mil

1101 : Mil cento e um.

Neste último caso, observe que o número pode ser construído com uma soma de suas partes: 1101 = 1000 + 100 + 1. O mesmo vale quando temos outros números:

12345 = 10000 + 2000 + 300 + 40 + 5
4532 = 4000 + 500 + 30 + 2

Observe que a posição de um número indica quantos zeros devem ser acrescentados ao seu lado para identificarmos a quantidade que ele representa. Considere este exemplo:

4356 = 4000 + 300 + 50 + 6

Se conisderarmos esse número contando suas casas da direita para a esquerda, começando em zero, teremos uma correspondência direta:

Casa	3	2	1	0
Dígito	4	5	3	2
Quantidade	4.000	500	30	2

Observe: na casa 3, a quantidade real tem 3 zeros; na casa 2, a quantidade real tem 2 zeros... e assim por diante. Na tabela abaixo, escreveremos as quantidades de uma maneira diferente:

Casa	3	2	1	0
Dígito	4	5	3	2
Quantidade	4×10^3	5×10^2	3×10^1	2×10^0

Observe que o expoente do "10" é exatamente a posição do dígito em questão. Isso ocorre porque estamos usando, para construir os números, 10 dígitos diferentes: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Sempre que não há mais dígitos em uma das posições, acrescentamos um à posição imediatamente à esquerda do número.

É por essa razão que esta representação é chamada de "decimal". A representação binária, por usa vez, usa apenas dois valores para os dígitos: 0 e 1. A representação hexadecimal usa dezesseis valores diferentes para os dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F.

3. A NOTAÇÃO BINÁRIA

Como apresentado anteriormente, acredita-se que os humanos trabalhem com números decimais por conta da quantidade de dedos que temos nas mãos; os computadores, entretanto, foram construídos com uma outra característica e, portanto, a representação mais natural neste caso é a binária.

A informação que pode ser representada por um "fio" - 0 ou 1 - é denominada **bit**, e é a menor unidade de informação de um computador. Se um processador tivesse apenas 1 bit, ele só seria capaz de representar os números 0 e 1. Mas, e se ele tiver 2 bits? A tabela abaixo mostra todas as combinações possíveis:

00 01 10 11

E se ele tiver 3 bits?

000 001 010 011 100 101 110 111

E se ele tiver 4 bits?

0000 0001 0010 0011 0100 0101 0110 0111
1000 1001 1010 1011 1100 1101 1110 1111

Observe que o número de bits representa o número de dígitos binários; além disso, considerando **n** bits, o número de combinações possíveis é dado por 2^n: um computador com 8 bits pode representar até $2^8 = 256$ números e um de 16 bits pode representar até $2^{16} = 65536$ números... e assim por diante.

Nota: como 256 variações eram suficientes para representar a maior parte das informações necessárias nos primeiros computadores, o conjunto de 8 bits ganhou um nome específico: byte. Assim, um **byte é um conjunto de 8 bits**.

3.1. Conversão de Números Binários para Decimais

Mas que números esse valores representam, em nossa notação decimal?

Existe uma regra de conversão muito simples. Lembremos como representamos o número decimal anterioremente:

Dígito	3	2	1	0
Número	1	5	3	7

$1537 = 1*10^3 + 5*10^2 + 3*10^1 + 7*10^0$

Se fizermos o mesmo com um número binário, por exemplo, 1101, teremos:

Dígito	3	2	1	0
Número	1	1	0	1

1101 binário $= 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$

Observe que os 10^n foram substituídos por 2^n; a razão para isso é que antes estávamos em uma base decimal, agora estamos em uma base binária. Façamos essa conta:

$1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 8 + 4 + 0 + 1 = 13$

Logo: 1101 binário = 13 decimal

Observe que agora temos uma confusão de representação: um número binário poderia ser lido erradamente como um número decimal. Para evitar esse problema, é usual acrescentar a letra "b" após valores binários, para evitar confusão. Em outras palavras, a representação é:

Texto	Valor (em decimal)
1101	1101
1101b	13

A tabela a seguir mostra a conversão das 16 combinações de números de 4 bits de binário para decimal:

Binário	Decimal	Binário	Decimal
0000b	0	1000b	8
0001b	1	1001b	9
0010b	2	1010b	10
0011b	3	1011b	11
0100b	4	1100b	12
0101b	5	1101b	13
0110b	6	1110b	14
0111b	7	1111b	15

3. CONVERTENDO HEXADECIMAL PARA DECIMAL

Os números hexadecimais usam 16 símbolos para cada dígito, como já visto. As quantidades de 0 a 15 serão representadas com apenas um digito em hexadecimal: 0h a Fh. A partir da quantidade 16 serão necessarios mais dígitos: 16 = 10h, 17 = 11h, 18 = 12h, 31 = 1Fh e assim por diante.

3.1. Conversão de Números Hexadecimal para Decimais

Multiplica-se cada dígito pela correspondente potência de **dezesseis**. Exemplo: converter o número 0x2F3C para decimal:

Dígito	3	2	1	0
Número	2	F	3	C

$0x2F3C=2*16^3 + 15*16^2 + 3*16^1 + 12*16^0 = 2*4096 + 15*16 + 3*16 + 12*1 = 12092$

4. BIBLIOGRAFIA

STALLINGS, W. **Arquitetura e organização de computadores**. 5ed. São Paulo: Ed. Pearson Prentice Hall, 2003.

MURDOCCA, M. J; HEURING, V.P. **Introdução à Arquitetura de Computadores**. S.I.: Ed. Campus, 2000.

Unidade 4: Conversões e Cálculos em Outras Bases

Prof. Daniel Caetano

Objetivo: Apresentar os métodos de conversão de decimal para diferentes bases e como realizar cálculos nas diferentes bases.

Bibliografia STALLINGS, 2003; MURDOCCA e HEURING, 2000.

INTRODUÇÃO

Alguns comportamentos do computador só são possíveis ao compreender como os nossos números são armazenados em sua memória e, em especial, ele realiza seus cálculos.

O objetivo desta aula é apresentar as conversões dos números decimais, usados pelos humanos, para os formatos binário e hexadecimal, usados pelos computadores. Adicionalmente, serão apresentados, simplificadamente, o processo de cálculo (soma) de números em outras bases e também a representação de caracteres.

1. CONVERSÃO DE DECIMAIS PARA BINÁRIOS

A conversão de números decimais para binários é bastante simples, sendo realizada com um processo de sucessivas divisões inteiras por dois, parando quando o divisor se tornar 0. Os restos das divisões vão compondo o valor em binário, da esquerda para a direita. Por exemplo: vamos transformar o valor 13 em sua representação binária:

13 / 2 = 6 e sobra...	1
6 / 2 = 3 e sobra...	0
3 / 2 = 1 e sobra	1
1 / 2 = 0 e sobra	1

Assim, o valor 13 é representado em binário como 1101b. Tentemos novamente com outro número maior, 118:

118 / 2 = 59 e sobra... 0
59 / 2 = 29 e sobra... 1
29 / 2 = 14 e sobra... 1
14 / 2 = 7 e sobra... 0
7 / 2 = 3 e sobra... 1
3 / 2 = 1 e sobra... 1
1 / 2 = 0 e sobra... 1

Assim, o valor 118 é representado em binário como 1110110b

2. CONVERSÃO DE DECIMAIS PARA HEXADECIMAIS

A conversão é similar à anterior, e é feita com divisões sucessivas por **dezesseis**, anotando os restos da divisão, que formam o número hexadecimal da direita para a esquerda. Exemplo: converter o valor 12092 em sua representação hexadecimal:

12092 / 16 = 755 e sobra... 12 (C)
755 / 16 = 47 e sobra... 3
47 / 16 = 2 e sobra... 15 (F)
2 / 16 = 0 e sobra... 2

Logo, 12092 = 0x2F3C

3. ARITMÉTICA EM OUTRAS BASES

Quando trabalhamos com números decimais, fazemos operações diretas. Por exemplo:

$$\begin{array}{r} {\scriptstyle 1} \\ 15 \\ \underline{+7} \\ 22 \end{array}$$

No fundo, alinhamos as casas (unidade com unidade, dezena com dezena, centena com centena...) e depois somamos uma a uma, começando com a unidade e, em seguida, partindo para a dezena e centena. Quando o resultado de uma das casas é maior do que o valor da base (no exemplo, 5 + 7 = 12), subtraímos deste resultado o valor da base (12 - 10 = 2) e, fazemos o "vai um" (representado no exemplo como o pequeno algarismo 1 sobre o 15).

Realizar a soma em outras bases é exatamente a mesma coisa. Veja em binário:

```
 1  1
 1101b          =    13
+0101b          =   +5
10010b          =    18
```

Começando da direita para a esquerda:

Primeira Casa: 1 + 1 = 2; como 2 não pode ser representado em binário, subtraímos 2 (2-2 = 0) e fazemos o vai um.

Segunda Casa: 0 + 0 = 0, somando com o "1" que veio da casa anterior 0+1 = 1.

Terceira Casa: 1 + 1 = 2. Mais uma vez não é possível representar, deixamos zero (2-2) no lugar e vai um.

Quarta Casa: A soma é 1 + 0 = 1, mas ao somar com o "1" que veio da casa anterior, 1+1 = 2, deixando zero no lugar e, mais uma vez, "vai um".

Quinta Casa: Como ela não existe nos números originais, permanece apenas o "1" que veio da casa anterior.

O processo é análogo para outras bases:

```
    1
 0x25      =     37
+0x3C      =    +60
 0x61      =     97
```

Começando da direita para a esquerda:

Primeira Casa: 5 + C(12) = 17. 17 não pode ser representado... então indicamos 17-16 = 1 e vai um

Segunda Casa: 2 + 3 = 5. Somando com o "1" que veio da casa anterior: 5 + 1 = 6.

Será que podemos aplicar a mesma lógica para a subtração? É claro que sim! Vejamos primeiro com decimais:

```
1
25
-7
18
```

Primeira Casa: temos 5 - 7; não é possível fazer, então "emprestamos um" da próxima casa, que aqui na unidade vale 10 e a nova conta é (10+5) - 7 = 8.

Segunda Casa: temos 2, mas que deve subtrair o "1" que foi emprestado pela primeira casa, então 2-1 = 1.

Vejamos agora em binário

```
 1 1 1
 1100b          =    12
-0101b          =    -5
 0111b          =     7
```

Primeira Casa: 0 - 1; não é possível. Então "emprestamos 1" da próxima casa, que aqui na primeira casa vale 2. A nova conta é, então (2+0) -1 = 1

Segunda Casa: 0-0 = 0; entretanto, precisamos descontar o 1 que foi emprestado para a primeira casa; como 0-1 não é possível, somos obrigados a emprestar 1 da terceira casa, que aqui vale 2. A nova conta é: (2+0)-1 = 1.

Terceira Casa: 1-1 = 0... mas mais uma vez é preciso descontar o 1 que foi emprestado para a casa anterior... e, para isso, é preciso emprestar 1 da quarta casa! Daí (2+0)-1 = 1.

Quarta Casa: 1-0 = 1, que descontado o 1 que foi emprestado... 0.

O mesmo pode ser aplicado para a multiplicação. Façamos direto em binário:

```
  11b     =     3
 x10b     =    x2
  00b           6
 11b+
 110b
```

A divisão fica como exercício!

4. REPRESENTAÇÃO DE CARACTERES

Até o momento vimos como armazenar números na memória. Mas como armazenar letras? Bem, este foi um problema que surgiu nos primórdios da computação e, por esta razão, existe uma solução padrão, que é a chamada Tabela ASCII (ASCII significa *American Standard for Computer Information Interchange*). A tabela ASCII relaciona cada valor numérico de um byte a cada um dos códigos visuais usados por nós na atividade da escrita. A tabela de conversão é apresentada na página seguinte (fonte: Wikipédia).

Observe, porém, que nem todos os caracteres são definidos por essa tabela: em especial, os caracteres acentuados estão faltando. Mas não são apenas estes: também não estão presentes os caracteres japoneses, chineses, russos... dentre tantos outros.

Por essa razão, atualmente existem diversas outras "tabelas de código de caracteres" ou "páginas de código de caracteres" (do inglês *codepage*), que extendem a tabela abaixo indicando os símbolos faltantes aos códigos livres (não especificados pela tabela ASCII). Entretanto, com a grande troca de arquivos entre pessoas de países diferentes, isso começou a causar alguma confusão.

Foi assim que surgiram então os códigos Unicode, que são versões alternativas e universais à tabela ASCII. O padrão UTF (Unicode Transformation Format) define várias tabelas, sendo as mais conhecidas e usadas as tabelas UTF-8 e UTF16. A tabela UTF-8 define 256 caracteres, como a tabela ASCII, mas com um padrão que tenta alocar a grande maioria dos símbolos usados pela maioria das línguas. Já o UTF-16 define 65.536 caracteres, englobando a grande maioria dos caracteres de todas as línguas. Existe ainda o padrão UTF-32, com capacidade para definir até 4 bilhões de caracteres, mas que é muito pouco usado.

Binário	Decimal	Hexa	Glifo
0010 0000	32	20	
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	-
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?

Binário	Decimal	Hexa	Glifo
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\
0101 1101	93	5D	]
0101 1110	94	5E	^
0101 1111	95	5F	_

Binário	Decimal	Hexa	Glifo
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q
0111 0010	114	72	r
0111 0011	115	73	s
0111 0100	116	74	t
0111 0101	117	75	u
0111 0110	118	76	v
0111 0111	119	77	w
0111 1000	120	78	x
0111 1001	121	79	y
0111 1010	122	7A	z
0111 1011	123	7B	{
0111 1100	124	7C	|
0111 1101	125	7D	}
0111 1110	126	7E	~

5. BIBLIOGRAFIA

STALLINGS, W. **Arquitetura e organização de computadores**. 5ed. São Paulo: Ed. Pearson Prentice Hall, 2003.

MURDOCCA, M. J; HEURING, V.P. **Introdução à Arquitetura de Computadores**. S.I.: Ed. Campus, 2000.

Unidade 8: A Unidade de Controle e o Ciclo de Instruções

Prof. Daniel Caetano

Objetivo: Apresentar as funções o mecanismo de atuação da Unidade de Controle.

Bibliografia:

- STALLINGS, W. **Arquitetura e organização de computadores**. 5ed. São Paulo: Ed. Pearson Prentice Hall, 2003.
- MURDOCCA, M. J; HEURING, V.P. **Introdução à arquitetura de computadores**. S.I.: Ed. Campus, 2000.

INTRODUÇÃO

Na aula anterior foi apresentada a Unidade Lógica Aritmética (ULA). Como vimos, a ULA é responsável por executar o processamento "de fato" de um computador, mas a ULA realiza apenas operações individuais. Para que a ULA processe seqüências de instruções, é necessário que algum dispositivo forneça tais instruções, na ordem correta.

Assim, nesta aula será continuado o estudo da Unidade Central de Processamento (CPU), apresentando a Unidade de Controle (UC) e alguns outros registradores importantes.

1. A UNIDADE DE CONTROLE

Uma das melhores analogias existentes entre a ULA e a UC é a analogia da calculadora. Enquanto a ULA é como uma calculadora simples, que executa um pequeno número de operações, a UC é como o operador da calculadora, que sabe onde buscar informações para alimentar a calculadora e também em que ordem estas infomações devem ser repassadas.

Em outras palavras, enquanto a ULA faz "partes" de um trabalho, a UC gerencia a execução destas partes, de forma que um trabalho mais complexo seja executado.

1.1. Algumas Responsabilidades da Unidade de Controle

- Controlar a execução de instruções, na ordem correta: uma vez que a ULA só cuida de executar instruções individuais, a UC tem o papel de ir buscar a próxima instrução e trazê-la para a ULA, no momento correto.

- Leitura da memória principal: Na aula anterior foi visto que a ULA não pode acessar diretamente a memória principal da máquina. A ULA só faz operações sobre os registradores, sendo que as instruções devem ser comandadas diretamente a ela. Assim, a UC tem o papel não só de buscar as instruções na memória, como também verificar se a instrução exige dados que estejam na memória. Se for o caso, a UC deve recuperar os dados na memória e colocá-los em registradores especiais e, finalmente, solicitar que ULA execute a operação sobre estes valores.

- Escrita na memória principal: Da mesma forma que a leitura, a ULA não pode escrever na memória principal da máquina. Assim, quando for necessário armazenar o resultado de uma operação na memória principal, é tarefa da UC transferir a informação de um registrador para a memória.

- Controlar os ciclos de interrupção: praticamente toda CPU atual aceita sinais de interrupção. Sinais de interrupção são sinais que indicam para a UC que ela deve parar, momentaneamente, o que está fazendo e ir executar uma outra tarefa. As razões para as interrupções são as mais diversas, como o disparo de um timer ou uma placa de rede / model solicitando um descarregamento de seu buffer.

1.2. Rotina de Operação da CPU

Em geral, é possível dizer que uma CPU tem uma seqüência de ações a executar; algumas delas são atividades da ULA, outras da UC. Esta seqüência está apresentada a seguir:

a) Busca de instrução: quando a CPU lê uma instrução na memória.
b) Interpretação de Instrução: quando a CPU decodifica a instrução para saber quais os passos seguintes necessários.
c) Busca de dados: caso seja determinado na interpretação que dados da memória ou periféricos são necessários, a CPU busca estes dados e os coloca em registradores.
d) Processamento de dados: quando a instrução requer uma operação lógica ou aritmética, ela é executada neste instante.
e) Escrita de dados: se o resultado da execução exigir uma escrita na memória ou periféricos, a CPU transfere o valor do registrador para o destino final.
f) Avaliação de Interrupções: após finalizar a execução de uma instrução, a CPU verifica se foi requisitada uma interrupção (Interrupt Request). Se sim, toma as providências necessárias. Se não, volta para **a)**.

Pelas responsabilidades da ULA e da UC, é possível perceber que a atividade **d** é executada pela ULA e todas as outras pela UC.

1.3. Registradores Usados pela UC

Assim como a ULA tem seus registradores especiais (*Acumulador* para armazenar os resultados e o *Flags* para indicar informações sobre a última operação executada), também a UC precisa de alguns registradores para funcionar correamente.

O primeiro deles vem da necessidade da UC saber onde está a próxima instrução a ser executada. Em outras aplavras, ela precisa de um registrador que indique a posição de memória em que a próxima instrução do programa estará armazenado (para que ela possa realizar a busca de instrução). Este registrador sempre existe, em todos os computadores microprocessados, mas seu nome varia de uma arquitetura para outra. Normalmente este registrador é chamado de **PC**, de Program Counter (Contador de Programa).

Sempre que é iniciado um ciclo de processamento (descrito na seção anterior), uma a UC busca a próxima instrução na memória, na posição indicada pelo PC. Em seguida, o PC é atualizado para apontar para a próxima posição da memória (logo após a instrução), que deve indicar a instrução seguinte.

Bem, como foi visto anteriormente, a UC precisa analisar esta instrução antes de decidir o que fazer em seguida. Por esta razão, costuma existir um registrador especial para armazenar a última instrução lida, chamado **IR**, de Instruction Register (Registrador de Instruções).

Para conseguir ler e escrever dados em memórias e periféricos, a UC também precisa de um contato com o barramento, o que é feito através de registradores especiais, de armazenamento temporário, chamados **MAR**, de Memory Address Register (Registro de Endereço de Memória) e o **MBR**, de Memory Buffer Register (Registro de Buffer de Memória). Assim, quando é preciso escrever na memória (ou em um periférico), a UC coloca o endereço no registrador MAR, o dado no registrador MBR e comanda a transferência pelo barramento de controle. Quando for preciso ler da memória (ou do periférico), a UC coloca o endereço no MAR, comanda a leitura pelo barramento de controle e então recupera o valor lido pelo MBR.

Adicionalmente a estes registradores, as CPUs costumam ter outros registradores que podem facilitar sua operação e mesmo sua programação. Alguns destes são os **registradores de propósito geral**, que servem para armazenar resultados intermediários de processamento, evitando a necessidade de muitas escritas e leituras da memória quando várias operações precisarem ser executadas em seqüência, a fim de transformar os dados de entrada nos dados de saída desejados. O nome destes registradores costuma ser letras diversas como B, C, D...

Existem também os **registradores de pilha**, normalmente com nomes como **SP**, de Stack Pointer (Ponteiro da Pilha) ou **BP** (Base da Pilha), que servem para que uma pilha seja usada pelo processador, na memória. Em essência, é onde o endereço de retorno é armazenado, quando um desvio é feito em linguagem de máquina; afinal, é preciso saber para onde voltar após a

realização de uma chamada de subrotina. A pilha que o processador fornece pode ser usada com outros objetivos, como passagem de parâmetros etc.

Quase todas as arquiteturas fornecem os **registradores de índices**, que são registradores que permitem acessar, por exemplo, posições de uma matriz. Ele guarda uma posição de memória específica e existem instruções que permitem acessar o n-ésimo elemento a partir daquela posição. Seus nomes variam muito de uma arquitetura para outra, como **IX**, de IndeX, **SI**, de Source Index (Índice Fonte) ou ainda **DI**, de Destination Index (Índice Destino).

Arquiteturas com segumento possuem ainda os **registradores de segmento**, que definem o endereço "zero" da memória para um determinado tipo de informação. A arquitetura x86, por exemplo, possui diversos registradores deste tipo: **CS**, de Code Segment (Segmento de Código), **DS**, de Data Segment (Segmento de Dados), **SS**, de Stack Segment (Segmento de Pilha) e **ES**, de Extra data Segment (Segmento de Dados Extra). Quando estes segmentos existem, os endereços usados nos índices, contador de programa e outros são "somados" com os endereços do segmento para que a posição real na memória seja calculada. Por exemplo:

SS = 10000h => Endereço do segmento da pilha
SP = 1500h => Endereço da pilha (dentro do segmento)
Endereço real da pilha = SS + SP = 10000h + 1500h = 11500h

Isso permite que um programa possa rodar em qualquer parte da memória, mesmo que ele tenha sido criado para ser executado no endereço 0h: basta eu indicar no registrador CS o endereço inicial de carregamento deste programa e, para todas as instruções deste programa, vai ser como se ele estivesse no endereço 0h. A CPU, com os registradores de sergento, são responsáveis pela tradução do endereço "virtual" para o endereço real.

Vale lembrar que cada arquitetura tem nomes distintos para estes resigtradores e em algumas delas existem ainda outros; além disso, alguns destes registradores até existem em algumas arquiteturas, mas não são visíveis para o programador, isto é, o registrador está lá e é usado pela CPU, mas o programador não tem acesso direto a eles (embora muitas vezes tenha acesso indireto, como sempre ocorre com os registradores MAR e MBR).

2. CICLO DE INSTRUÇÃO

Nas seções anteriores já foi descrito o ciclo de instrução, que é a seqüência de passos que a UC segue até que uma instrução seja executada. Nesta parte será apresentado um diagrama genérico que mostra todos os passos que um processador comum executa para executar suas instruções. Os principais subciclos são os de busca de instruções, busca de dados, execução e interrupção.

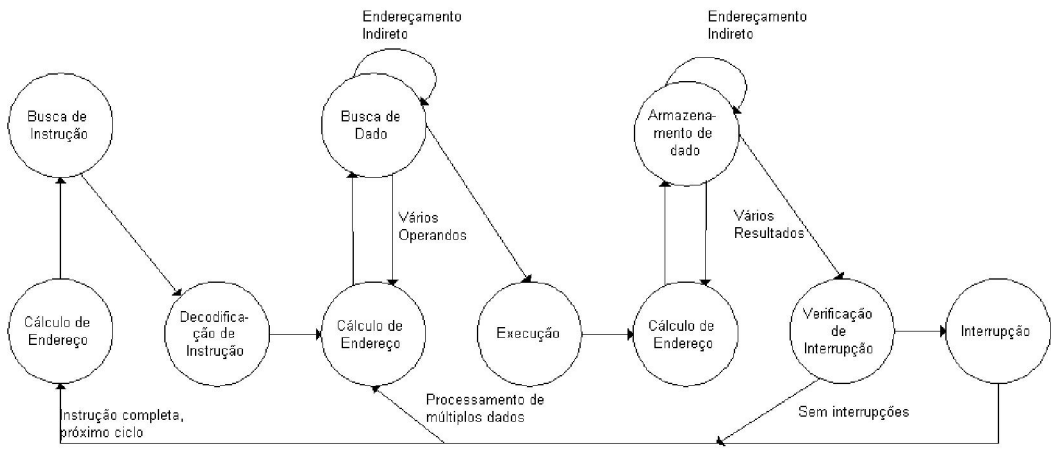


Figura 1 - Diagram de transição de estados do ciclo de instrução (STALLINGS, 2003)

Como é possível ver, o primeiro passo é a busca de instrução, onde a UC coloca o valor do PC no MAR, comanda leitura da memória e recebe o dado (que neste caso é uma instrução) pelo MBR, que em seguida copia para o IR.

Em seguida, a UC decodifica a instrução, avaliando se há a necessidade de busca de dados adicionais (por exemplo, se for uma instrução do tipo "ADD A,B", nenhum dado precisa ser buscado. Se houver a necessidade, o próximo passo é a busca do dado, onde a UC realiza o mesmo processo da leitura da instrução, mas agora para a leitura do dado. Esse processo é repetido até que todos os dados necessários tenham sido colocados em registradores.

O passo seguinte é a execução, onde a UC meramente comanda a ULA para executar a operação relevante. A ULA devolve um resultado em um registrador. Se este dado precisar ser armazenado externamente à CPU, a UC cloca este dado na MBR e o endereço destino na MAR e comanda a escrita, usando o barramento de controle. Se houver mais de um dado a armazenar, este ciclo é repetido.

Finalmente, a UC verifica se há *requisição de interrupção pendente*. Se houver, ela executa o ciclo da interrupção (que varia de arquitetura para arquitetura). Caso contrário, o funcionamento prossegue, com o cálculo do novo endereço de instrução e o ciclo recomeça.

3. A PIPELINE

A ideia de pipeline é a mesma da produção em série em uma fábrica: "quebrar" a produção de alguma tarefa em pequenas tarefas que podem ser executadas paralelamente. Isso significa que vários componentes contribuem para o resultado final, cada um executando sua parte.

Como foi possível ver pela seção anterior, existem vários passos em que a execução de uma instrução pode ser dividida. A idéia, então, é que cada um destes passos seja executado independentemente e por uma parte diferente da CPU, de forma que o processamento ocorra mais rapidamente.

Mas como isso ocorre? Imagine o processo explicado na seção 2. Nos momentos em que a comunicação com a memória é feita, por exemplo, a ULA fica ociosa. Nos momentos em que a ULA trabalha, a comunicação com a memória fica ociosa. Certamente o processamento linear não é a melhor forma de aproveitar os recursos.

Imagine então que emos duas grandes etapas (simplificando o processo explicado anteriormente): a etapa de busca e a etapa de execução. Se tivermos duas unidades na UC, uma para cuidar da busca e outra para cuidar da execução, enquanto a execução de uma instrução está sendo feita, a seguinte já pode estar sendo buscada! Observe as seqüênciasa seguir.

	SEM pipeline		COM pipeline	
Seqüência no Tempo	Busca	Execução	Busca	Execução
0	I1	-	I1	-
1	-	I1	I2	I1
2	I2	-	I3	I2
3	-	I2	I4	I3
4	I3	-	I5	I4

Observe que no tempo que foram executadas 2 instruções sem pipeline, com o pipeline de 2 níveis foram executadas 4 instruções. Entretanto, isso é uma aproximação grosseira, pois os tempos de execução de cada um destes estários é muito diferente, sendo que o aproveitamento ainda não é perfeito. Para um bom aproveitamente, precisamos dividir as tarefas em blocos que tomem mais ou menos a mesma fatia de tempo. Este tipo específico de pipeline é chamado de "Prefetch" (leitura antecipada).

Se quebrarmos, por exemplo, a execução em 6 etapas: Busca de Instrução (BI), Decodificação de Instrução (DI), Cálculo de Operandos (CO), Busca de Operandos (BO), Execução da Instrução (EI) e Escrita de Operando (EO), termos etapas mais balanceadas com relação ao tempo gasto. Observe na tabela abaixo o que ocorre:

	SEM Pipeline						COM Pipeline					
T	BI	DI	CO	BO	EI	EO	BI	DI	CO	BO	EI	EO
0	I1	-	-	-	-	-	I1	-	-	-	-	-
1	-	I1	-	-	-	-	I2	I1	-	-	-	-
2	-	-	I1	-	-	-	I3	I2	I1	-	-	-
3	-	-	-	I1	-	-	I4	I3	I2	I1	-	-
4	-	-	-	-	I1	-	I5	I4	I3	I2	I1	-
5	-	-	-	-	-	I1	I6	I5	I4	I3	I2	I1
6	I2	-	-	-	-	-	I7	I6	I5	I4	I3	I2
7	-	I2	-	-	-	-	I8	I7	I6	I5	I4	I3
8	-	-	I2	-	-	-	I9	I8	I7	I6	I5	I4
9	-	-	-	I2	-	-	I10	I9	I8	I7	I6	I5
10	-	-	-	-	I2	-	I11	I10	I9	I8	I7	I6
11	-	-	-	-	-	I2	I12	I11	I10	I9	I8	I7
12	I3	-	-	-	-	-	I13	I12	I11	I10	I9	I8

Basicamente, no tempo que foram executadas 2 instruções sem pipeline, foram executadas 8 instruções com pipeline. É claro que o tempo de execução de uma instrução sem pipeline neste caso de 6 estágios é aproximadamente o mesmo tempo de execução da mesma instrução sem pipeline no caso com 2 estágios; Isso ocorre porque, obviamente, cada um dos 6 estágios deste caso toma um tempo muito menor que cada um dos dois estágios do modelo anterior.

Bem, mas se o número de estágios aumenta o desempenho, porque não usar o máximo possível? Por algumas razões. Uma delas é que, a partir de um determinado número de estágios a quebra pode acabar fazendo com que dois estágios passem a gastar mais tempo de execução do que o estágio original que foi dividido. Mas esta não é a razão fundamental: existe um gargalo mais evidente no sistema de pipelines: ele pressupõe que as instruções são independentes entre si; assim, para que o pipeline tenha o desempenho apresentado, uma instrução a ser executada não pode depender do resultado das instruções anteriores.

Quando uma instrução depende do resultado das anteriores, teremos alguns estágios "esperando" a execução da outra instrução terminar para que a "cadeia de montagme possa ter prosseguimento. Vamos dar um exemplo. Imagine que A instrução I2 dependa do resultado da instrução I1 para ser executada. Então, o que vai acontecer no pipeline é descrito a seguir:

COM Pipeline

T	BI	DI	CO	BO	EI	EO
0	I1	-	-	-	-	-
1	I2	I1	-	-	-	-
2	I3	I2	I1	-	-	-
3	I4	I3	I2	I1	-	-
4	I5	I4	I3	I2	I1	-
5	I5	I4	I3	I2	-	I1
6	I6	I5	I4	I3	I2	-
7	I7	I6	I5	I4	I3	I2
8	I8	I7	I6	I5	I4	I3
9	I9	I8	I7	I6	I5	I4
10	I10	I9	I8	I7	I6	I5
11	I11	I10	I9	I8	I7	I6
12	I12	I11	I10	I9	I8	I7

Observe que, comparando com o quadro anterior, uma instrução a menos foi processada. Isso é pior ainda quando uma instrução do tipo "desvio condicional" precisa ser interpretada; isso porque a posição de leituras da próxima instrução vai depender da execução de uma instrução. Neste caso, quando ocorre este tipo de desvio, o pipeline é esvaziado e perde-se uma boa parte do desempenho.

Ocorre que a chance destes "problemas" acontecerem e os atrasos causados por eles aumentam com o número de níveis do pipeline. Desta forma, um número excessivo de níveis de pipeline podem acabar por degradas o desempenho, além de fazer com que uma CPU aqueça mais e mais. Para entender isso (o aquecimento) pense em uma fábrica: quanto mais funcionários, mais confusa é a movimentação dentro da fábrica. Nos circuitos, os níveis de pipeline fazem o papel dos funcionários da linha de montagem e, quanto maior número de movimentações internas de sinais, maior é o calor gerado.

A subdivisão excessiva dos pipelines foi o que matou a linha Intel Pentium IV, que foi abandonada. A Intel precisou retroceder sua tecnologia à do Pentium M (Pentium III móvel) e continuar o projeto em outra direção, com menos níveis de pipeline, o que deu origem aos processadores Pentium D, Core 2 Duo e os mais atuais Core i.

4. BIBLIOGRAFIA

STALLINGS, W. **Arquitetura e organização de computadores**. 5ed. São Paulo: Ed. Pearson Prentice Hall, 2003.
MURDOCCA, M. J; HEURING, V.P. **Introdução à Arquitetura de Computadores**. S.I.: Ed. Campus, 2000.

Unidade 9: Memórias

Prof. Daniel Caetano

Objetivo: Compreender os tipos de memória e como elas são acionadas nos sistemas computacionais modernos.

INTRODUÇÃO

Nas aulas anteriores foram apresentadas diversas maneiras de interpretar os bits na memória; essa compreensão é de extrema importância, mas não responde às perguntas: o que é, como funciona e como é acessada a memória?

O objetivo desta aula é apresentar uma introdução sobre os diferentes tipos de memórias existentes no computador, além de apresentar a forma com que a memória - e outros dispositivos - são acessados, através do **barramento de sistema**, que será detalhado em aulas posteriores.

1. O QUE É A MEMÓRIA?

Em palavras simples, a memória é um dispositivo físico capaz de armazenar e recuperar uma configuração elétrica em um "conjunto de fios". Uma vez que essa configuração elétrica estabelece um padrão de bits, ligados ou desligados, é possível dizer que a memória armazena e recupera **dados**.

Olhando como uma caixa preta, a memória é bastante simples. Observe a Figura 1.

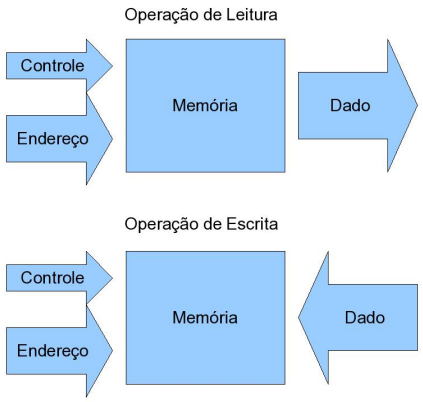


Figura 1: Operações de leitura e escrita na memória

Simplificadamente, o dispositivo memória recebe sinais de controle - que indicam se a operação é de leitura ou escrita na memória - e um endereço. Se a operação for de leitura, o dispositivo memória responde emitindo o dado armazenado no endereço em questão; se a

operação for de escrita, o dispositivo recebe o dado e o armazena na posição de memória indicada pelo endereço.

Apesar de ser um dispositivo de funcionamento aparentemente simples, as memórias são possivelmente os dispositivos com maior diversidade de implementações. Isso ocorre porque há diversas características que buscamos nas memórias como velocidade e capacidade, que não são atendidas plenamente por nenhum dos tipos de memória existente.

Há memórias que são rápidas, mas se forem desligadas perdem os dados armazenados e são muito caras; há memórias que são de velocidade média e possuem um preço razoável, mas se forem desligadas também perdem os dados. Há memórias muito baratas e que mantém a informação quando são desligadas, mas que são muito lentas...

Além disso, nem todas as memórias fornecem dados do "tamanho" que o processador "quer". O tamanho dos dados lidos pelo processador é denominado **palavra** e pode ter diferentes tamanhos, como 8, 16, 32, 64, 128bits... dentre outros. Isso é o que determina, em geral, a expressão "processador de 64 bits": o tamanho do dado que ele manipula diretamente. Algumas memórias simplesmente fornecem os dados em blocos maiores do que uma palavra, exigindo algum *"malabarismo*" para permitir seu uso direto com um dado processador.

Adicionalmente, para que se possa tirar máximo proveito de um sistema computacional, a velocidade da memória deve ser compatível com a velocidade do processador, de maneira que esse último não precise ficar esperando por respostas da memória por muito tempo. Em tese, considerando os processadores atuais, isso exigiria que toda a memória fosse muito rápida e, como consequência, os equipamentos seriam muito caros e praticamente não poderiam ser desligados.

Certamente esse não era um caminho viável e, por essa razão, cirou-se uma outra alternativa: usar diversos tipos de memória para obter o melhor desempenho ao menor custo.

2. HIERARQUIA DE MEMÓRIA

A quantidade de dados que um usuário medio armazena é gigantesca. Se considerarmos um servidor de uma grande empresa, essa quantidade de dados é ainda maior. Entretanto, a grande maioria desses dados **raramente** é usada pelo computador. Isso ocorre porque apenas um pequeno conjunto de programas e dados é usada rotineiramente.

Ainda assim, mesmo considerando os programas e dados que são processados com frequência, se medirmos a quantidade de tempo que o processador gasta com cada um destes bytes, veremos que a maior parte do tempo o computador está executando pequenos blocos de instruções e dados, realizando **tarefas repetidas**.

Observando este comportamento, os projetistas de *hardware* concluíram que poderiam equilibrar o custo de um equipamento se usassem um tipo de diferente de memória para cada tarefa:

a) Registradores e Memória Cache (Armazenamento Interno): Para armazenamento de curto prazo, de dados usados intensivamente pelo computador, adotam-se dispositivos de armazenamento volátil extremamente rápidos, mas de pequena capacidade devido ao **custo por bit ser muito alto**.

b) Memória Principal (Armazenamento Interno): Para armazenamento de médio prazo, de dados medianamente usados, adotam-se dispositivos de armazenamento volátil, cujo **custo por bit é médio**, proporcionando média capacidade com uma velocidade de acesso também média, já que estes dados são usados com alguma frequência.

c) Memória Secundária (Armazenamento Externo): Para armazenamento de longo prazo, de dados pouco usados, adotam-se dispositivos de armazenamento não volátil e cujo **custo por bit é baixo**, proporcionando grande capacidade, ainda que sejam lentos. A lentidão não é um problema, pois os dados aí contidos são pouco acessados.

d) Memória de Segurança (Armazenamento de Segurança): Para armazenamento de longuíssimo prazo, de dados que talvez nunca sejam necessários, adotam-se dispositivos de armazenamento não volátivo de **custo por bit extremamente baixo**, com enrome capacidade, ainda que extremamente lentos.

Estes quatro níveis formam a hierarquia de memória, lembrando que todos os dados úteis de um computador precisam estar armazenados na memória secundária, sendo transferidos para a memória principal na medida em que são necessários. Da mesma forma, os dados da memória principal são transferidos para o cache e para os registradores também na medida em que são necessários. A comunicação ocorre obedecendo a hierarquia, como pode ser visto na Figura 2.

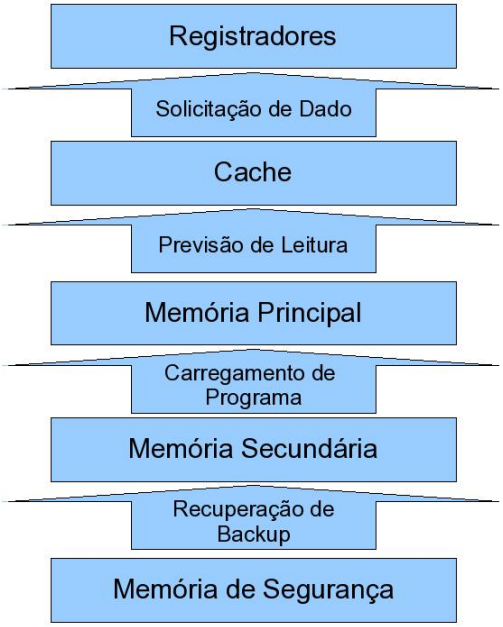


Figura 2: Situações de transferência de dados entre memórias (leitura)

3. TIPOS DE MEMÓRIA

As memórias utilizadas em cada uma destas camadas da hierarquia podem ser construídas com diferentes tecnologias. A diferenciação mais básica está entre os tipos RAM e ROM, mas existem diversos subtipos. Algumas delas estão descritas no quadro a seguir.

Tipo	Categoria	Apagamento	Escrita	Volatilidade	Palavra / Bloco	Interna / Externa	Velocidade	Usos	Custo por Bit
Memória de Acesso Aleatório Estática (SRAM)	Escrita e Leitura	Eletrica-mente	Eletrica-mente	Volátil	Bytes	Interna	Variada (pode ser tão rápida quanto o processador)	Registradores, cache, memória principal	De muito alto a alto
RAM Dinâmica (DRAM)	Escrita e Leitura	Eletrica-mente	Eletrica-mente	Volátil	Bytes	Interna	Média	Memória principal	Médio
Memória apenas de Leitura (ROM)	Apenas de leitura	Impossível	Máscaras	Não volátil	Bytes	Interna / Externa	Média para Rápida	Memória principal ou secundária	Baixo
ROM Programável (PROM)	Apenas de leitura	Impossível	Eletrica-mente	Não volátil	Bytes	Interna / Externa	Média para Rápida	Memória principal ou secundária	Baixo
PROM Apagável (EPROM)	Principal-mente de leitura	Luz UV	Eletrica-mente	Não volátil	Bytes	Interna / Externa	Rápida para leitura	Memória principal ou secundária	Baixo
EPROM Eletricamente apagável (EEPROM)	Principal-mente de leitura	Eletrica-mente	Eletrica-mente	Não volátil	Bytes ou Blocos	Interna / Externa	Rápida para leitura	Memória principal ou secundária	Médio
Memória Flash	Principal-mente de leitura	Eletrica-mente	Eletrica-mente	Não volátil	Blocos	Externa	Média para Lenta	Memória secundária	Médio
Disco Magnético	Escrita e Leitura	Magneti-camente	Magneti-camente	Não volátil	Blocos	Externa	Lenta	Memória secundária	Baixo
Disco Óptico	Leitura (e, opcio-nalmente, Escrita)	Óptica	Óptica	Não volátil	Blocos	Externa	Muito lenta	Memória secundária e de segurança	Muito baixo
Fita Magnética	Escrita e Leitura	Magneti-camente	Magneti-camente	Não volátil	Bytes	Externa (no passado, interna)	Extremamente Lenta	Memória secundária e de segurança (no passado, principal)	Extrema-mente baixo

Das memórias voláteis, qual a diferença entre a memória SRAM e DRAM? Bem, a SRAM é um dispositivo que basta estar ligado para preservar seus dados; a DRAM, por outro lado, exige que de tempos em tempos seja feita uma "simulação de leitura" em cada região da memória, para garantir que ela não seja perdida, em um processo chamado ***refresh***.

Como a maioria das memórias permite apenas **um acesso** por vez, isto é, ela só permite que um endereço de memória seja acessado de cada vez, durante o momento em que o ***refresh*** está sendo executado a memória DRAM fica **indisponível** para o processamento e, por essa razão, ela acaba sendo, no geral, mais lenta que a SRAM.

Projetar circuitos com SRAM é muito mais simples do que com DRAM; entretanto, a diferença de preço entre ambas faz com que, em geral, se a velocidade de uma SRAM não é necessária, os projetistas adotem o uso de DRAMs.

4. ACESSO A MEMÓRIA

Desde o início do curso é comentado que as partes do computador se comunicam por "fios". De fato, é isso que ocorre, embora esses "fios" muitas vezes sejam trilhas minúsculas em uma placa de circuito impresso.

Por exemplo, no caso da memória, apresentamos na Figura 1 "setas" que indicavam o fluxo de informações para a memória. Na prática, essas "setas" são fios ou trilhas, pelos quais trafegam sinais elétricos: a presença de sinal é interpretada como "1" e a ausência de sinal é interpretad como "0", formando os padrões desejados. Veja o exemplo da Figura 3.

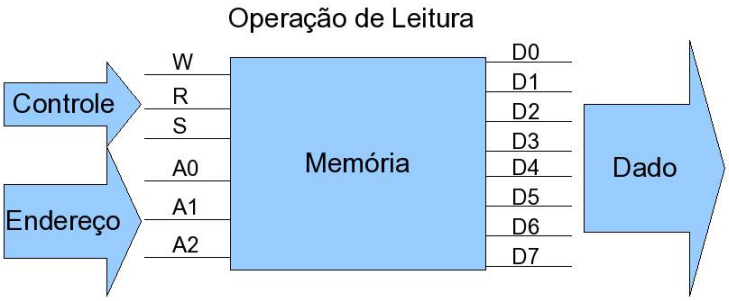


Figura 3: Nomenclatura dos "fios" que ligam a memória ao sistema

Nesta figura, os "fios" do controle foram nomeados de W (Write), R (Read) e S (Select). Estes fios controlam o funcionamento da memória. Sempre que a memória estiver sendo usada, S (ou MREQ) deverá estar com sinal em 1, indicando que a memória está selecionada. Quando S (ou MREQ) estiver em 1, os valores de W e R indicarão qual é a operação solicitada.

Quando a memória for usada para escrita, o sinal de W deve estar em 1 e R em 0; quando a memória for usada para leitura, R deverá estar em 1 e W em 0. O comportamento se S (MREQ), R e W estiverem todos simultaneamente em 1 é indefinido.

Os "fios" de endereço foram nomeados de A0 a A2 (A vem de Address). Isso significa que essa memória tem 3 bits de endereçamento, permitindo acesso a 8 dados (2^3). Em outras palavras, **temos 8 posições de memória**.

Os "fios" para representar os dados foram nomeados de D0 a D7 (D vem de Data). Isso significa que cada uma das posições de memória tem 8 bits (1 byte), proporcionando o armazenamento de 256 valores (2^8) diferentes em cada posição de memória.

Assim, como temos 8 posições de 1 byte cada, esta é uma memória de 8 bytes.

Vamos analisar agora como funcionaria a memória no momento de uma leitura. Num primeiro momento, o circuito do computador irá indicar nos "fios" do controle que ele deseja ler a memória e, nos "fios" de endereço, vai indicar que deseja ler um determinado endereço. Suponhamos que o endereço a ser lido seja o endereço 6, ou seja, 110b. Observe a configuração na Figura 4.

Figura 4: Configuração para a leitura do endereço de memória 6 (110b)

O circuito do computador coloca essa configuração elétrica nos "fios" de entrada da memória e, alguns instantes depois, a memória configura eletricamente os "fios" de dados (D0 a D7) com a informação que nela está armazenada, conforme indicado na Figura 5.

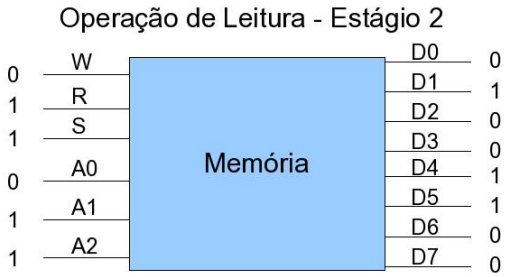


Figura 5: Resposta da memória à leitura do endereço de memória 6 (110b)

Ou seja: o dado armazenado na posição 6 da memória é: 00110010b, ou, interpretando como um decimal inteiro, 50.

Por terem funções **muito** distintas, cada um destes conjuntos de fios recebem nomes específicos. Os "fios" que controlam os dispositivos ligados ao computador são chamados de **barramento de controle**. Os "fios" que configuram endereços de memória e outros dispositivos são chamados de **barramento de endereços** e, finalmente, os "fios" que servem para a troca de dados entre os vários dispositivos são chamados de **barramento de dados**.

O exemplo desta aula foi feito com uma unidade de memória mas, de maneira geral, o procedimento é parecido para qualquer dispositivo que seja ligado no sistema computacional moderno.

Isso ocorre porque os computadores são projetados segundo um paradigma de arquitetura denominado "barramento de sistema", que será visto na aula que vem.

6. BIBLIOGRAFIA

MURDOCCA, M. J; HEURING, V.P. **Introdução à arquitetura de computadores**. S.I.: Ed. Campus, 2000.

STALLINGS, W. **Arquitetura e organização de computadores**. 5ed. São Paulo: Ed. Pearson Prentice Hall, 2003.

Unidade 10: Dispositivos de Entrada e Saída

Prof. Daniel Caetano

Objetivo: Conhecer alguns dos dispositivos de entrada e saída mais comuns, além de compreender a lógica de comunicação entre os dispositivos de E/S mais comuns.

Bibliografia:

- MURDOCCA, M. J; HEURING, V.P. **Introdução à arquitetura de computadores**. S.I.: Ed. Campus, 2000.
- STALLINGS, W. **Arquitetura e organização de computadores**. 5ed. São Paulo: Ed. Pearson Prentice Hall, 2003.

INTRODUÇÃO

Grande parte da funcionalidade de um computador se deve à sua capacidade de se comunicar com dispositivos de entrada e saída. De fato, é usual que se dê o nome de "computador" apenas a dispositivos que realizem operações e que possuam pelo menos uma unidade de entrada e uma de saída.

Uma vez que os dispositivos existentes são os mais variados e, em geral, possuem uma velocidade de comunicação **muito inferior** à da memória, com **tamanhos de palavras** usualmente **distintos** daqueles trabalhados pelo computador, não é praticável instalá-los no mesmo barramento de alta velocidade da memória, o que faz com que normalmente possuam um barramento diferenciado, acessado através de uma das pontes, como visto em aulas anteriores.

Além disso, a maneira com que as informações destes dispositivos são transferidas para a memória pode variar em nível de complexidade, do mais simples e lento ao mais complexo e eficiente. Nesta aula serão apresentados os três modos de comunicação existentes nos computadores modernos. A maior diferença entre os três é o nível de interferência da CPU no processo de comunicação de um dado dispositivo com a memória.

No caso de maior intervenção da CPU no processo, é dado o nome de **Entrada e Saída Programada** (ou ***polling***). No caso de menor intervenção, é dado o nome de **Acesso Direto à Memória** (ou ***Direct Memory Access***, **DMA**). No caso intermediário, é dado o nome de **Entrada e Saída Controlada por Interrupção**.

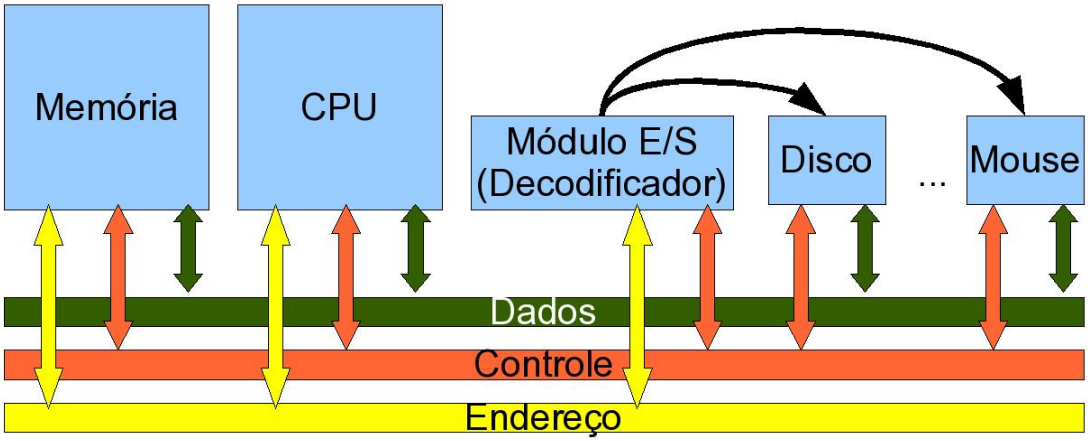
Após a visão geral sobre os métodos de transferência de dados entre dispositivos e memória, serão comentados alguns dos dispositivos mais comuns atualmente.

1. COMO SE ATIVA UM DISPOSITIVO DE E/S?

Um dispositivo de E/S, em certo aspecto, é bastante difernente da memória. Enquanto a memória recebe um sinal de ativação e ela é preparada para responder a todos os endereços (recebe os sinais A0 a AN), os dispositivos de E/S tem um funcionamento diferenciado, usualmente tendo apenas um (ou alguns poucos) sinal de ativação (A0).

Para diferenciar entre esses comportamentos, o processador tem, usualmente, formas alternativas para ativar o uso da a memória ou dos dispositivos de E/S. Em geral, o processador central tem um pino chamado MREQ (do barramento de controle) que é ligado diretamente ao seletor de ativação da memória e, no caso da memória, o mesmo ocorre com o barramento de endereços inteiro.

No caso dos dispositivos de E/S, a CPU os ativa por meio de um pino chamado IORQ (também do barramento de controle), que junto com o barramento de endereços aciona um (ou vários) dispositivo chamado **decodificador** que, dependendo da configuração do endereço, acionará um dispositivos adequado. Observe a figura abaixo, simplificada com um único barramento:



Note que este desenho é um exemplo, apenas para indicar como o **módulo de E/S** "interfere" na ativação do dispositivo, em uma funcionalidade chamada de **decodificação de endereço de E/S**.

Alguns dispositivos, como o vídeo e o *hard disk*, não possuem qualquer contato direto com o barramento do sistema, ficando completamente interligados ao módulo de E/S, que assume a responsabilidade de gereciar os três barramentos. Esse tipo de móldulo de E/S é chamado **interface controladora de dispositivo**. Cada dispositivo pode ter seu próprio controlador ou é possível ter um controlador para vários dispositivos.

Como exemplo desses módulos E/S temos a placa de vídeo ou a placa SCSI/SATA. Em alguns casos, o módulo decodificador está fisicamente integrado ao dispositivo, de maneira que não é possível observá-lo.

Note que o que se chama de **interface controladora** não é apenas um seletor e decodificador de endereços; usualmente ela tem muito mais recursos, permitindo, por

exemplo, que um determinado dispositivo "tome o controle" dos barramentos temporariamente, permitindo que o dispositivo converse diretamente com a memória (comum) ou com outros dispositivos (incomum).

A partir da próxima seção estudaremos quais são as formas com que a comunicação com o dispositivo pode ocorrer.

2. ENTRADA E SAÍDA POR *POLLING*

No esquema chamado *polling*, a CPU é responsável por todo o controle de transferências de dados de dispositivos. Isso significa que ela é responsável não só pela transferência de informações em si, mas também pela verificação constante dos dispositivos, para saber se algum deles tem dados a serem transferidos.

Isso significa que, de tempos em tempos, a CPU faz a seguinte 'pergunta', seqüencialmente, a todos os dispositivos conectados: "Você tem dados para serem transferidos para a memória?".

Quando algum dispositivo responder "sim", a CPU faz a transferência e continua perguntando aos outros dispositivos em seguida. Quando nenhum dispositivo necessitar de transferências, a CPU volta a fazer o que estava fazendo antes: executar um (ou mais) programas. Depois de algum tempo, ela volta a realizar a pergunta para todos os dispositivos novamente.

> Uma analogia que costuma ajudar a compreender a situação é a do garçom e a do cliente em um restaurante. No sistema de *polling* o garçom é a CPU e o cliente é o dispositivo. O cliente tem que esperar pacientemente até o garçom resolver atendê-lo e, enquanto o garçom atende a um cliente, ele não pode realizar qualquer outra tarefa. Depois que o cliente fez o pedido, ele ainda tem que esperar, pacientemente, o garçom trazer a comida.

Não é difícil ver que este sistema tem **três** problemas fundamentais:

a) A CPU gasta uma parcela considerável de tempo de processamento só para verificar se algum dispositivos tem dados a serem transferidos para a memória.
b) A CPU gasta uma parcela considerável de tempo de processamento apenas para transferir dados de um dispositivo para a memória.
c) Se um dispositivo precisar de um atendimento "urgente" (porque vai perder dados se a CPU não fizer a transferência imediata para a memória, para liberar espaço no dispositivo), não necessariamente ele terá.

A primeira questão é considerada um problema porque muitas vezes a CPU perde tempo perguntando para todos os dispositivos e nenhum deles tem qualquer dado a ser transferido. É tempo de processamento totalmente disperdiçado.

A segunda questão é considerada um problema porque cópia de dados é uma tarefa que dispensa totalmente a capacidade de processamento: é uma tarefa que mobiliza toda a

CPU mas apenas a Unidade de Controle estará trabalhando - e realizando um trabalho menos nobre. Adicionalmente, para a transferência de um dado do dispositivo para a memória, tendo que passar pela CPU, o barramento é ocupado duas vezes pelo mesmo dado, ou seja, são realizadas duas transferências: dispositivo=>CPU e depois CPU=>Memória. Isso acaba sendo um "retrocesso", transformando o barramento em um modelo de Von Neumann simples.

A terceira questão é considerada um problema porque, eventualmente, dados serão perdidos. Adicionalmente, se o dispositivo em questão precisar **receber** dados para tomar alguma atitude, problemas mais sérios podem ocorrer (como uma prensa hidráulica controlada por computador causar a morte de uma pessoa por falta de ordens em tempo hábil).

Por outro lado, o sistema de transferência por *polling* é de implementação muito simples, o que faz com que muitas vezes ele seja usado em aplicações onde os problemas anteriormente citados não são totalmente relevantes.

3. ENTRADA E SAÍDA POR INTERRUPÇÃO

No esquema chamado de entrada e saída por interrupção, a CPU fica responsável apenas pelas transferências em si. Isso significa que ela não tem que verificar os dispositivos, para saber se há dados a serem transferidos.

Mas se a CPU não faz a verificação, como ela vai perceber quando uma transferência precisa ser feita? Simples: o dispositivo dispara um sinal do barramento de controle chamado "Interrupção" (chamado de IRQ - *Interrupt ReQuest*). Quando a CPU percebe este sinal, ela sabe que algo precisa ser feito com algum dispositivo; normalmente uma transferência de dados (seja de entrada ou saída).

> Voltando a analogia do restaurante, o sistema com interrupções seria o fato de o cliente possuir uma sineta que, ao tocar, o garçom viria o mais rapidamente possível para atender ao cliente

Há sistemas em que há mais dispositivos que interrupções. Neste caso, o sistema ainda terá que fazer *polling* para saber qual foi o dispositivo que solicitou atenção; entretanto, o polling será feito somente quando ***certamente*** um dispositivo precisar de atenção da CPU (uma entrada ou saída de dados). Desta forma, elimina-se o problema de tempo perdido fazendo *pollings* quando nenhum dispositivo precisa de transferências.

A forma mais eficiente, entretanto, é quando temos pelo menos uma interrupção por dispositivo, de forma que a CPU saiba sempre, exatamente, qual é o dispositivo que está solicitando atenção e nenhum tipo de *polling* precise ser feito.

Essa característica resolve os problemas a) e c) existentes no sistema de *polling* puro, embora o problema b), relativo ao tempo de CPU gasto com as transferências em si, ainda esteja presente.

Entretanto, sempre que lidamos com sinais no barramento, temos que levantar uma questão: e se dois ou mais dispositivos solicitarem uma interrupção ao mesmo tempo? Normalmente os sistemas com várias interrupções possuem vários **níveis de prioridade de interrupção** (*interrupt level*). A CPU sempre atenderá a interrupção de maior prioridade primeiro (normalmente de "número" menor: IRQ0 tem mais prioridade que IRQ5).

Existe um outro problema também: quando a CPU recebe uma IRQ, ela **sempre** para o que está fazendo, momentaneamente, para realizar outra atividade qualquer. Ao finalizar esta atividade, ela volta ao que estava fazendo antes da IRQ ocorrer. Entretanto, há alguns processos em que talvez o programador não queria interrupções - em aplicações onde o controle de tempo é crítico, as interrupções podem causar problemas graves. Nestes casos, o programador pode usar uma instrução em linguagem de máquina que desliga as interrupções (normalmente chamada DI, de *Disable Interrupts*). Obviamente ele precisa ligá-las novamente depois (usando uma instrução normalmente chamada EI, de *Enable Interrupts*).

Entretanto, ao desligar as interrupções, o programador pode, potencialmente, causar um dano grave ao funcionamento do sistema operacional, por exemplo. Os sistemas operacionais modernos usam a interrupção para realizar a troca de aplicativos em execução, na chamada "multitarefa preemptiva". Por esta razão, as arquiteturas modernas possuem pelo menos uma interrupção chamada de Interrupção Não Mascarável (NMI, de *Non Maskable Interrupt*), que não pode ser desligada, nunca.

4. ENTRADA E SAÍDA POR DMA

No esquema chamado de entrada e saída por DMA (Acesso Direto à Memória), a CPU fica responsável apenas por coordenar as transferências. Isso significa que ela não tem que verificar os dispositivos, para saber se há dados a serem transferidos e nem mesmo transferir estes dados.

Mas se a CPU não faz a verificação, como ela vai perceber quando uma transferência precisa ser feita? Como já foi visto, pela interrupção (assim, DMA pressupõe interrupções). Mas se a CPU não faz a transferência, como os dados vão parar na memória? Simples: a CPU comanda um dispositivo responsável pela transferência, normalmente chamado simplesmente de DMA. Quando a CPU perceber o sinal de IRQ, ela verifica qual a transferência a ser feita e comanda o DMA, indicando o dispositivo origem, a posição origem dos dados, a posição destino dos dados e o número de bytes a transferir. O circuito do DMA fará o resto. Quando ele acabar, uma outra interrupção será disparada, informando que a cópia foi finalizada.

> Voltando a analogia do restaurante, o sistema com DMA seria como se, ao cliente tocar a sineta, o garçom (CPU) mandasse um outro garçom auxiliar (DMA) para fazer o que precisa ser feito, e o garçom "chefe" (CPU) continuasse a fazer o que estava fazendo antes. Ao terminar seu serviço, o garçom auxiliar (DMA) avisa ao garçom chefe (CPU) que está disponível novamente.

Este sistema resolve todos os problemas a), b) e c) apresentados anteriormente.

5. DISPOSITIVOS PRINCIPAIS DE ENTRADA E SAÍDA

Os dispositivos de entrada e saída dos computadores atuais são de conhecimento geral. Entretanto, veremos alguns detalhes de seu funcionamento interno.

5.1. HardDisks

Os harddisks são, em essência, muito similares aos disk-drives e disquetes comuns; entretanto, existem diferenças.

Dentro de um harddisk existem, normalmente, vários discos (de alumínio ou vidro) fixos a um mesmo eixo, que giram em rotações que variam de 3000 a 1000 rpm (rotações por minuto). Estes discos são, assim como os disquetes, cobertos por óxido de ferro, que pode ser regionalmente magnetizado em duas direções (indicando 0 ou 1). Estas regiões são acessíveis na forma de setores de uma trilha do disco. O tamanho de bits disponíveis em cada setor varia de caso para caso, mas 512 bytes é um valor comum.

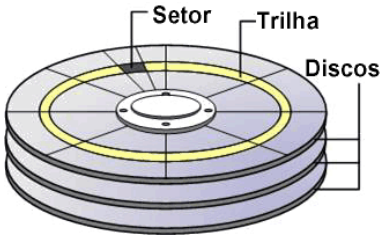


Figura 1: Discos, trilhas e setores

Para ler estes bits de um setor, existe normalmente uma cabeça de leitura para cada face de disco. No caso acima, seriam, normalmente, 6 cabeças de leitura, presas a um mesmo eixo. Apenas uma funciona por vez.

Quando é necessário ler um dado do disco, é preciso descobrir em que setor(es) ele está e ler este(s) setor(es). Um sistema de arquivos é exatamente uma maneira ordenada de guardar arquivos em diversos setores e o sistema operacional faz a parte mais chata de localização, fazendo com que os programadores possam lidar diretamente com o conceito lógico de "arquivos".

5.2. Discos Ópticos

Os discos ópticos são discos plásticos cobertos de uma camada de alumínio que é "entalhado" para representar os 0s e 1s: 1 quando não houver uma concavidade e 0 quando houver. Para a leitura há uma cabeça que é capaz de disparar um feixe de laser e também é capaz de recebê-lo de volta. O feixe de laser é disparado contra a superfície "entalhada" do disco; se encontrar uma concavidade, sofre um tipo de deflexão e a lente detecta esta deflexão como sendo um 0. Se não encontrar a concavidade, a deflexão será diferente, e a lente detectará esta outra deflexão como 1.

Os dados não são colocados em um disco óptico em um formato de trilhas e setores. Os dados em um disco óptico são colocados em forma espiral, que vai do interior para a borda do disco. O disco **não** gira com velocidade constante: gira mais lentamente à medida em que a cabeça se afasta do centro do disco.

5.3. Teclados

Independente da posição das teclas de um teclado, em geral elas são compreendidas pelo computador na forma de uma matriz com um determinado número de linhas e colunas ou simplesmente numeradas (juntando o número da linha e da coluna). Por exemplo: se a tecla "A" fica na linha 4 e coluna 2, então o número da tecla poderia ser 402, por exemplo.

Entretanto, a maioria dos teclados consegue enviar apenas um número de 7 ou 8 bits para o computador (0 a 127 ou 0 a 255), fazendo com que uma numeração tão simples não seja possível, mas a idéia permanece. Ainda assim, as teclas especiais que modificam o significado das teclas (como shift, alt, ctrl, etc) fariam com que 7 (ou 8) bits jamais fossem suficientes para transmitir todas as possíveis combinações. Por esta razão, alguns dos valores de 7 (ou 8) bits são separados para indicar o pressionamento de teclas especiais, de forma que os valores são transmitidos separadamente.

A função do "mapa de teclado" que é selecionado ao configurar um sistema operacional é justamente a de "mapear" cada número que o teclado irá enviar ao computador a uma letra correta. É por esta razão que se um teclado US-International for configurado com um mapa ABNT-2 as teclas não se comportarão como o esperado.

Mais uma vez o sistema operacional faz a maior parte do trabalho para o programador, que recebe diretamente o valor correto da tecla (letra ou número), ao invés de ter que lidar com números de teclas no teclado.

5.4. Mouses Ópticos

Os mouses ópticos são dispositvos extremamente complexos. Sua função é emitir uma luz sobre uma superfície e digitalizar a imagem da superfície onde ele se encontra. O mouse realiza essa operação milhares de vezes por segundo, comparando as imagens coletadas para detectar qual foi o **deslocamento** entre elas.

Depois de calculado este deslocamento, o mouse converte esse valor em um deslocamento proporcional que será enviado ao computador, que moverá o ponteiro do mouse na tela. É importante notar que o deslocamento na tela será tão maior quanto mais brusco for o movimento do mouse (um deslocamento rápido do mouse causa maior deslocamento na tela que o mesmo deslocamento de mouse feito lentamente).

5.5. Monitores de Vídeo LCD

Os monitores de vídeo do tipo LCD são compostos de 3 partes:

a) um papel de fundo, onde são pintados todos os pixels, cada um deles divididos verticalmente em 3 cores: vermelho, verde e azul (RGB, Red Green Blue).

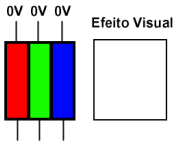
b) um sistema de iluminação (que deve iluminar uniformemente o papel de fundo)

c) uma tela de cristal líquido

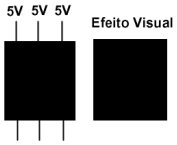
A tela de cristal líquido é composta por uma matriz de elementos de cristal líquido. O cristal líquido tem uma propriedade que é a de impedir que a luz passe em uma dada direção quando colocado sobre uma diferença de potencial suficientemente alta. A quantidade de luz que o cristal líquido deixa passar naquela direção é proporcional à diferença de potencial (campo elétrico). Pequena diferença de potencial faz com que muita luz passe; grande diferença de potencial faz com que pouca luz passe.

Cada pixel tem, então, três células de cristal líquido: uma para a região vermelha do papel, outra para a região verde do papel e outra para a região azul do papel. As cores são compostas variando o campo elétrico em cada célula de cristal líquido. Exemplos:

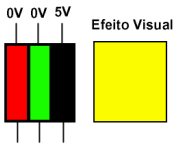
1) Campo baixo em R, G e B: cor branca (todas as cores "acesas")



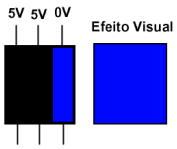
2) Campo alto em R, G, B: cor preta (todas as cores "apagadas")



3) Campo baixo em R e G e alto em B: cor amarela (só vermelho e verde "acesos")



4) Campo baixo em B e alto em R e G: cor azul (só azul "aceso")



As variações de intensidade e matiz das cores são obtidas variando a diferença de potencial em cada uma das células de cristal líquido.

6. BIBLIOGRAFIA

MURDOCCA, M. J; HEURING, V.P. **Introdução à arquitetura de computadores**. S.I.: Ed. Campus, 2000.

STALLINGS, W. **Arquitetura e organização de computadores**. 5ed. São Paulo: Ed. Pearson Prentice Hall, 2003.