ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

CAPÍTULO 3 - EM QUE CONSISTE O PROCESSAMENTO DAS INSTRUÇÕES E COMO O PROJETO DA MEMÓRIA PODE AUXILIAR?

Fernando Cortez Sica

Introdução

Você chegou até aqui vendo muito conteúdo interessante sobre o funcionamento e a concepção básica dos computadores. Que tal falarmos agora sobre o processamento das instruções e alguns itens relacionados à memória?

Neste capítulo, vamos estudar essas questões. Começaremos com a unidade de controle, para que possamos ter a resposta para o possível questionamento: como o processador efetivamente processa as instruções? Neste ponto, abordaremos o que vêm a ser as micro-operações e como elas são abstraídas e tratadas pelo processador.

A essa altura, poderá estar se formando, em sua mente, mais uma pergunta: se existem as micro-operações, o tratamento de interrupções também é abstraído como uma sequência de instruções divididas, ainda, em micro-operações? Assim, para encontrar as respostas, abordaremos a questão das interrupções apresentando um fluxo de execução realizado pelo processador.

Também vamos estudar alguns conceitos relacionados à interconexão. Assim, o processador conseguirá buscar, nos demais módulos pertencentes ao computador, as informações e instruções que deverão ser manipuladas.

Por fim, serão abordados aspectos do sistema de memória. O sistema de memória é alvo de muitas pesquisas pelo fato de ser um grande gargalo que impacta a performance computacional. Então, poderá surgir mais uma questão: como atuar no projeto do sistema de memória para que se possa melhorar o seu desempenho? A resposta está nos quesitos ligados à hierarquia de memória e à memória *cache*.

A partir dos pontos abordados neste capítulo, você terá condições de responder às indagações inerentes à performance computacional e, também, favorecer o desenvolvimento de sistemas computacionais ainda mais eficientes.

Vamos em frente? Bons estudos!

3.1 Unidade de Controle (UC)

A UC tem como principal objetivo gerenciar todas as etapas de processamento de uma instrução sincronizando os submódulos internos do processador por intermédio da geração de sinais de controle. No entanto, cabe a pergunta: como a UC consegue fazer isso?

A resposta inicia na fase da implementação da UC, concebida pela metodologia conhecida como implementação por *hardware* (*hardwired*). Nesta técnica, as etapas de processamento de uma instrução são mapeadas pelos circuitos de eletrônica digital, em que um conjunto de portas lógicas consegue decodificar a instrução em questão, gerando os sinais de controle.

A partir deste ponto, você pode ter, em sua mente, outra pergunta: quais etapas de instrução são essas? Uma instrução, para que se possa aproveitar melhor os recursos computacionais e, também, para que os submódulos sejam mais simples, rápidos e otimizados, é quebrada em várias micro instruções (ou micro-operações). A execução de cada micro-operação ficará sob a responsabilidade de um módulo de *hardware* específico.

VOCÊ QUER LER?

Os processadores da Intel, como é o caso do Core i7, são tidos como processadores híbridos, pois mantêm, em sua arquitetura e de forma fortemente acoplada, núcleos baseados no padrão CISC (*Complex Instruction Set Computer*) e no padrão RISC (*Reduced Instruction Set Computer*) (RAISCH, 2010). Leia mais sobre os processadores: <https://www.ibm.com/developerworks/community/blogs/tlcbr/entry/computadores_hibridos_a_proxima_fronteira_da_computacao?lang=en>.

Essa divisão da instrução em etapas menores favorece o *pipeline*, por meio do qual consegue-se antecipar, mais facilmente, o início de uma nova instrução.

3.1.1 Micro-operações

Sabemos que cada instrução é dividida em várias operações mais simples denominadas como micro-operações. Mas, não somente a fase de execução propriamente dita é dividida em micro-operações, também, todas as etapas de execução da instrução, ou seja, todo o ciclo de instrução.

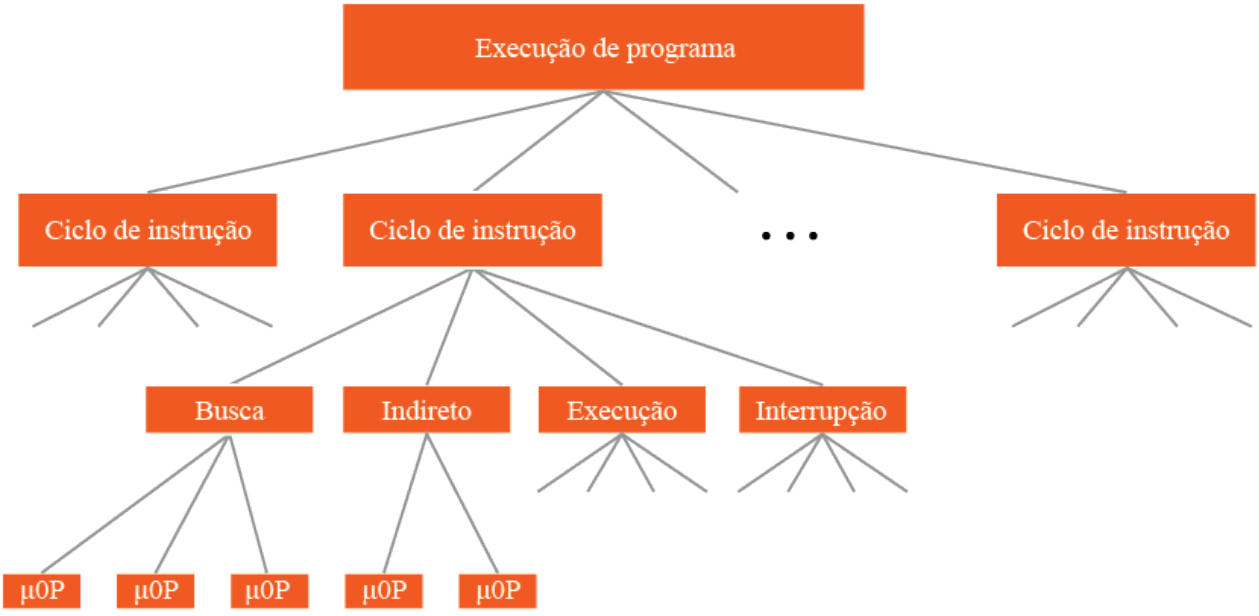


Figura 1 - Divisão das etapas de execução em micro-operações, sendo que cada micro-operação é executada em um pulso de máquina.
Fonte: STALLINGS, 2010, p. 462.

Na figura acima podem ser notadas as micro-operações em todo o ciclo de instrução. A etapa de busca da instrução poderá ser, então, dividida em micro-operações, tais como:

- instanciação do registrador MAR (*Memory Address Register*), com o valor contido em PC (*Program Counter*);

- ativação do sistema de memória, para acessar o endereço cuja localização está constando no MAR. O valor retornado pela memória será salvo no registrador MBR (*Memory Buffer Register*);
- incrementação do valor do registrador PC. Essa micro-operação poderá ser realizada concomitantemente à micro-operação descrita anteriormente;
- cópia do valor buscado da memória, que foi salvo no registrador MBR para o registrador IR (*Instruction Register*).

Ainda em relação à figura acima, o mesmo processo de divisão de uma fase dentro do ciclo de instrução pode ser aplicado ao ciclo indireto. O ciclo indireto ocorre na fase de busca dos operandos. Nessa ocasião, o ciclo pode ser dividido nas micro-operações abaixo listadas:

- registrador MAR recebe o campo do IR, que contém o endereço de memória, o qual contém a referência para que seja acessada a informação requerida;
- acessa-se a memória na posição MAR para coletar a referência de memória e armazená-la no MBR;
- atualiza-se o campo de IR para que se possa, na próxima fase, acessar a localização definitiva de memória que contém a informação a ser manipulada.

VOCÊ O CONHECE?

O cientista da computação Maurice Vincent Wilkes (1913-2010), foi o responsável pelo projeto do computador EDSAC, em 1949, quando a ideia de micro-operações foi concebida. Mais tarde, essa ideia foi utilizada pela empresa americana IBM em sua série 360, no ano de 1964 (UFRGS, 2010). Em 1967, Wilkes recebeu o Prêmio Turing, por essa e outras contribuições. Para conhecer mais sobre a trajetória de Wilkes acesse: <http://www.ufrgs.br/mvs/Periodo02-1949-ElectronicDelay.html>.

Como mencionado, essa divisão em micro-operações em muito facilita a técnica de *pipeline*, sendo que o tempo para a execução de uma micro-operação é muito menor em relação ao tempo para processar toda a instrução. Por consequência, os módulos responsáveis pela execução da micro-operação tornam-se disponíveis mais rapidamente, favorecendo, portanto, o início de execução da micro-operação referente à instrução seguinte.

3.1.2 Ciclos de interrupção, execução e instrução

A interação entre os dispositivos de E/S (entrada e saída ou I/O – *input/output*) é feita mediante o envio de sinais de interrupção. Uma interrupção é tratada de acordo com uma sequência de instruções análoga à de um processo (programa em execução). Sendo assim, o processo sob processamento deverá ser interrompido para que sejam carregadas, no *pipeline*, as instruções relativas à interrupção. No entanto, se uma interrupção, em nível do ciclo de instruções, comporta-se como um processo, então ela também é quebrada em micro-operações? A resposta é sim. Não somente o código, relativo à interrupção em si, é dividido em micro-operações, mas, também, a troca de contexto. Salvar contexto denota a operação de armazenar, na memória (estrutura em pilha), informações tais como o valor do registrador PC, para que, ao término da interrupção, seja possível retornar exatamente ao ponto de parada do programa. As micro-operações para o atendimento da interrupção podem ser expressas na sequência de micro-operações descritas a seguir:

- registrador MBR recebe o valor do PC para que seja empilhado;
- registrador MAR é instanciado com a localização do topo do segmento de pilha (*stack segment*);
- PC é setado com o endereço da rotina de tratamento da interrupção;
- a memória é acessada para efetivar o empilhamento do PC (contido no MBR).

Na implementação do tratamento de interrupção, o seu término é marcado por uma instrução do tipo IRET (*Interrupt Return*). As micro-operações relativas ao IRET são parecidas à sequência acima, porém ocorre o desempilhamento do registrador PC.

Até o momento, foram referenciadas etapas fixas. Mas você deve estar se perguntando: tudo é assim tão fixo e previsível? As micro-operações para a fase de execução são também fixas como nas etapas de busca da instrução, ciclo indireto e tratamento da interrupção independentemente do tipo e funcionalidade da instrução? Cada instrução pertence a um grupo distinto e apresenta uma funcionalidade também distinta. Então, é de se esperar que a sequência de micro-operações varie de instrução para instrução – mesmo entre instruções da mesma classe (mesmo formato de instrução).

VOCÊ SABIA?

Você sabia que o processador Intel Core i7, Nehalem, possui um conjunto de instruções composto por macro-instruções CISC (*Complex Instruction Set Computer*) que são traduzidas em micro-operações RISC (*Reduced Instruction Set Computer*)? E que, para executar as micro-operações, o i7 apresenta um *pipeline* com profundidade de 14 estágios? Detalhes dessas afirmativas você poderá encontrar em Hennessy (2014).

Tomemos, para ilustrar a divisão da etapa de execução em micro-operações, duas instruções hipotéticas quaisquer: a primeira trata de uma escrita na memória do tipo STORE posição, e, a segunda, um desvio condicional do tipo JZ offset (*jump* caso zero).

Para a instrução do tipo STORE posição, vamos supor que o valor a ser salvo na memória, encontra-se no registrador acumulador. Nesse caso, a instrução poderá ser decomposta nas seguintes micro-operações:

- salvar o valor do acumulador no registrador MBR;
- carregar, no registrador MAR, o campo de endereço presente no registrador IR;
- acessar a memória para efetivar a gravação.

Já com relação à instrução JZ offset, vamos supor que a operação de comparação (ou subtração), que servirá de base à instrução de desvio condicional já tenha ocorrido. Dessa forma, a instrução JZ offset poderá ser decomposta nas seguintes micro-operações:

- carregar, em algum registrador *R*, o valor contido no IR (*offset*);
- somar, ao registrador *R*, o valor de PC;
- caso *flag Z* seja igual a zero, então PC recebe o valor de *R*.

Apesar de, em ambos os exemplos, as instruções terem sido divididas, coincidentemente, em quatro micro-operações, na realidade, o número de micro-operações pode se diferenciar nas diferentes instruções.

Convém lembrar que o número de micro-operações está relacionado à implementação do processador (organização). Sendo assim, processadores da mesma família (apresentando a mesma arquitetura) podem variar em relação ao número e à forma das micro-operações.

Para controlar a sequência de operações das micro-operações, a unidade de controle manipula um registrador especial para tal função: o ICC (*instruction cycle code*). Esse registrador é instanciado ao longo da execução das micro-operações para armazenar o estado corrente e, consequentemente, determinar a próxima micro-operação a ser realizada. A figura a seguir ilustra um fluxograma da execução das micro-operações a partir do valor instanciado no registrador ICC.

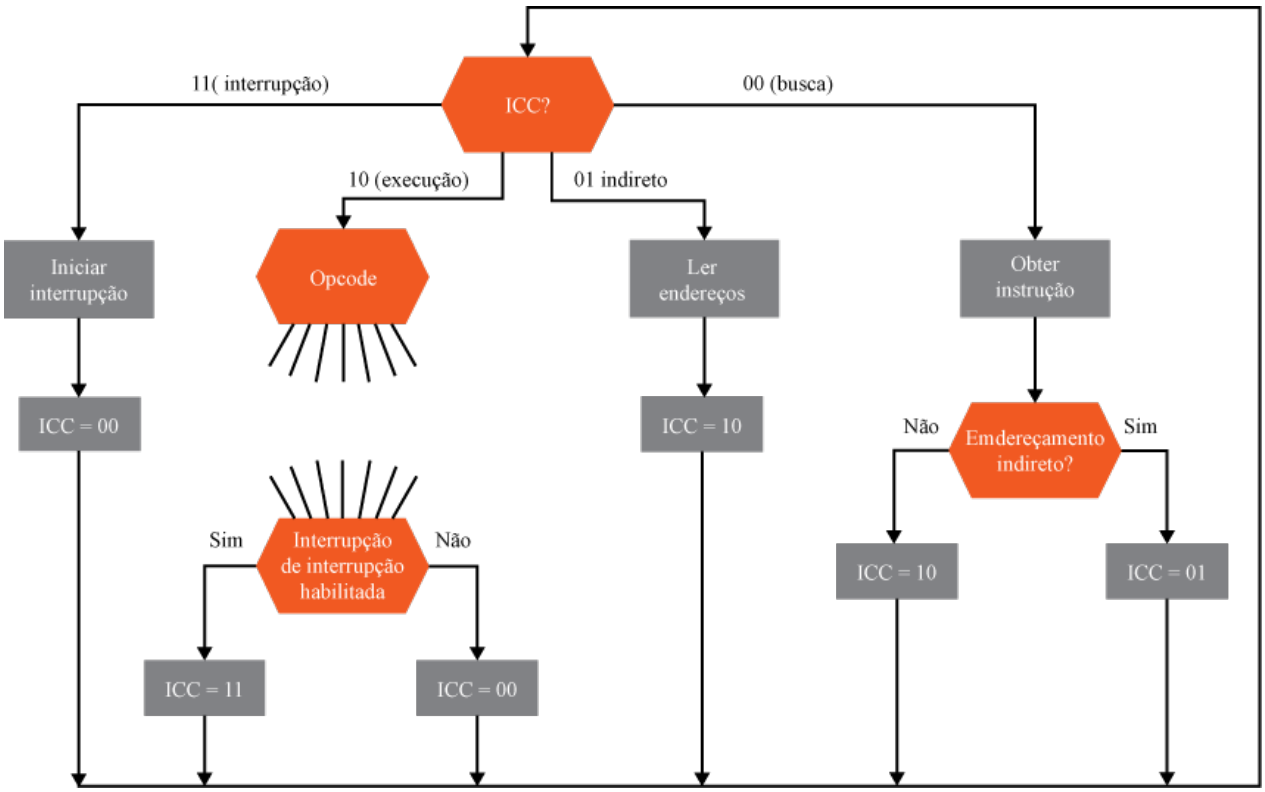


Figura 2 - Fluxograma da execução das micro-operações a partir do valor instanciado no registrador ICC.
Fonte: STALLINGS, 2010, p. 467.

Na figura acima, as ações são representadas pelos retângulos, e os hexágonos representam as tomadas de decisão em função do valor do registrador ICC. Na figura, foram convencionados os seguintes valores para ICC: 00 = busca; 01 = indireto; 10 = execução; 11 = interrupção. Dessa forma, foram representadas todas as etapas inerentes ao ciclo de instrução. Com base no valor do ICC, o *hardware* pode acionar os submódulos correspondentes e rotear as informações por entre eles.

3.2 Função do computador e interconexão

Você já deve estar familiarizado com o fato de que o computador tem por finalidade executar instruções pertencentes a um programa (programas do usuário, do próprio sistema operacional ou referentes ao tratamento das interrupções). Todas as instruções estão dentro de uma das seguintes classes: processador-memória, processador-E/S, processamento propriamente dito e controle.

Então, as micro-operações devem, também, abranger as funcionalidades básicas do computador. Segundo Stallings (2010), as funcionalidades das micro-operações estão classificadas da seguinte forma:

- movimentações de dados do tipo registrador-registrador;
- movimentações de dados entre registrador e barramento (para interfacear, por exemplo, o sistema de memória);
- efetuar operações lógicas ou aritméticas.

Como você pode ter notado, essas funcionalidades das micro-operações são evocadas para o tratamento das interrupções e, por diversas vezes, acessam os barramentos do sistema para completar as suas ações. Diante desse fato, vamos, agora, dar um foco nas interrupções e nos barramentos do sistema computacional.

3.2.1 Interrupções: sequenciais, múltiplas e aninhadas

Interrupções são eventos utilizados para interfacear os dispositivos de E/S com o processador. Elas podem ocorrer a partir da evocação do dispositivo (por exemplo, quando é pressionada uma tecla do teclado) ou a evocação pode partir do *software* (por exemplo, o programa necessita salvar uma informação em um arquivo).
Em ambos os casos, elas deverão ser tratadas, interrompendo o fluxo de processamento dos aplicativos abertos. A figura abaixo ilustra o tratamento das etapas de execução de uma instrução envolvendo a manipulação de interrupções.

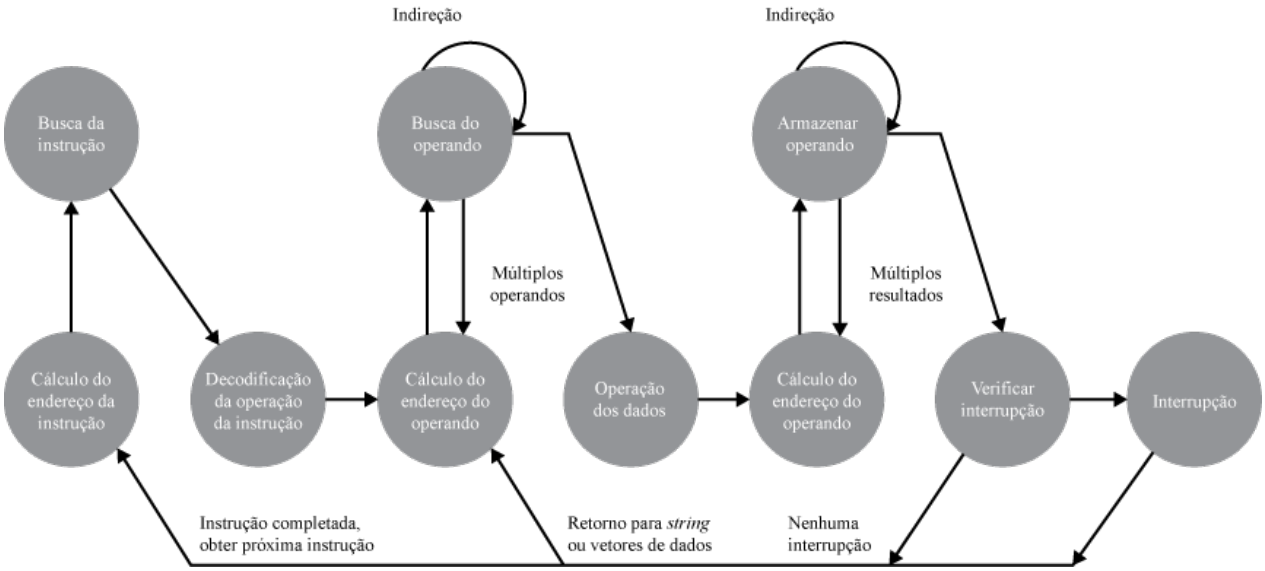


Figura 3 - Sequência de etapas para execução das instruções, sendo que, após cada instrução, há a verificação da existência de solicitações de interrupções.
Fonte: STALLINGS, 2010, p. 363.

Na figura acima, nota-se que, após a execução de uma instrução ser completada, há a verificação, se há alguma interrupção enfileirada pendente a ser executada. Nesse caso, ela entraria como um processo normal, ou seja, as instruções vinculadas ao tratamento da interrupção entrariam no fluxo do *pipeline*.
Quando um fluxo de execução é adicionado ao *pipeline*, o que acontece quando alguma outra interrupção ocorre? Quando esse cenário ocorre, temos duas maneiras de realizar o tratamento: na primeira forma, assim que o tratamento da interrupção for iniciado, poderemos desabilitar novas interrupções ou permitir que as próprias interrupções sejam interrompidas para o atendimento de outras. A figura abaixo mostra a diferença entre esses dois enfoques.

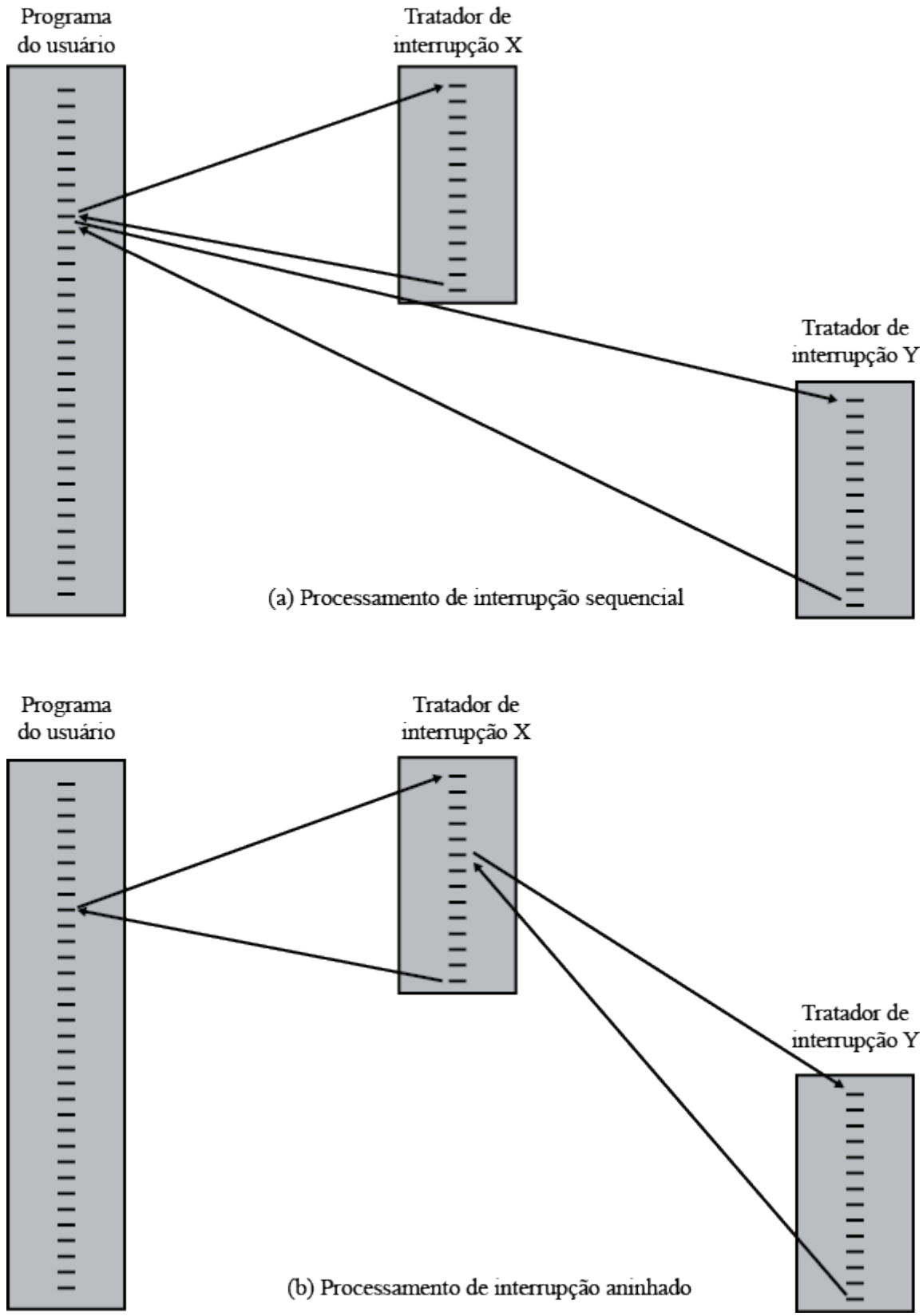


Figura 4 - Tratamento de interrupções com a desabilitação de novas interrupções (a) e tratamento de novas interrupções de forma aninhada (b).
Fonte: STALLINGS, 2010, p. 65.

Em relação à figura acima, temos o tratamento de interrupções sequencialmente (a). Nesse tipo de enfoque, ocorre a desabilitação de interrupções, para que as novas que ocorrerem sejam apenas enfileiradas, de modo que o tratamento ocorra somente após o término da atual. Por sua vez, no tratamento aninhado (b), uma interrupção poderá ser descontinuada, para que seja tratada uma com maior prioridade.
Em ambos os casos (sequencial ou aninhado), ocorrerá a interrupção da execução dos processos do usuário para que as interrupções sejam tratadas pelo *pipeline*.

3.2.2 Barramentos: de endereço, dados e controle

Mais uma vez retomando as classes das instruções, você pode notar que existem instruções que dependem de uma estrutura de comunicação com os demais módulos do computador (memória e dispositivos de E/S), para que possam ser processadas. Essa estrutura de comunicação é provida pelos barramentos. A figura a seguir, ilustra, em linhas gerais, as interfaces requeridas pelos módulos do computador e a estrutura básica de uma rede de interconexão (barramento).

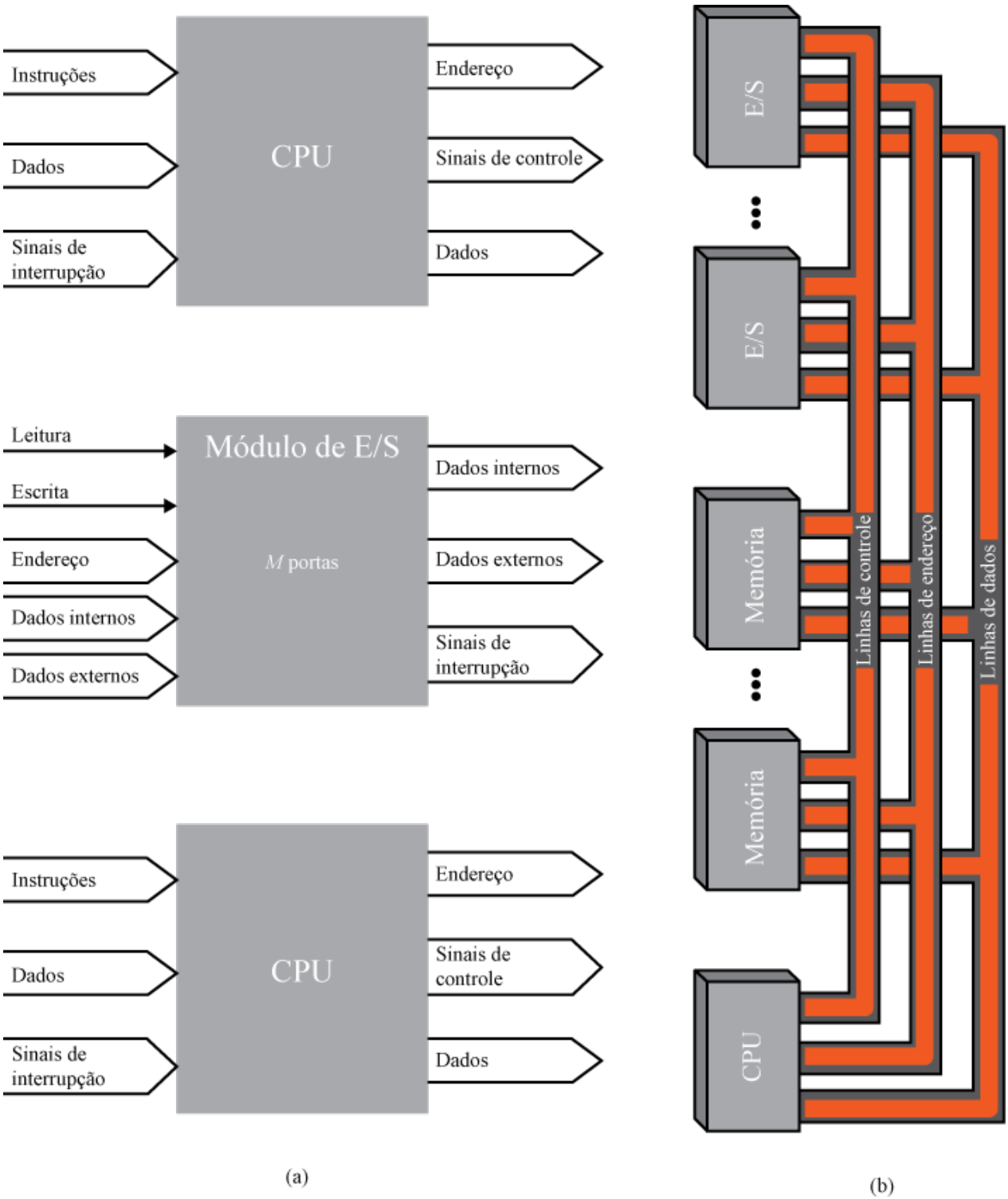


Figura 5 - Interfaces requeridas pelos módulos do computador (a) e a estrutura básica de um barramento (b).
Fonte: Adaptada de STALLINGS, 2010, p. 67-69.

Como você pode notar na figura acima, as interfaces dos módulos do computador, em (a), correspondem, em suma, às linhas de dados, endereços e de sinais de controle. Sendo assim, um barramento deve ter, em sua funcionalidade básica, vias para o tráfego de tais sinais (exibidos em (b), destacados em vermelho):

- **vias de dados**: as vias de dados são responsáveis pelo tráfego, em paralelo, das informações a serem processadas, armazenadas ou enviadas aos dispositivos de E/S;

- **vias de endereçamento**: tem por objetivo transmitir endereços de memória ou de módulos de E/S;
- **vias de sinais de controle**: como sinais de controle podemos ter (STALLINGS, 2010):

- leitura/escrita de memória: define a operação a ser realizada quando o acesso à memória for solicitado;
- leitura/escrita de E/S: define a operação a ser realizada quando o acesso a algum dispositivo de E/S for solicitado;
- ACK de transferência (*acknowledgment*): o ACK é um sinal de reconhecimento cuja função é avisar o término, de forma satisfatória, da operação de transferência;
- **solicitação de barramento**: sinal gerado por algum módulo para que use o barramento;
- concessão de barramento: resposta à solicitação de barramento, outorgando ao módulo, o direito do uso do barramento;
- requisição de interrupção: sinal que indica a pendência de alguma interrupção;
- ACK de interrupção: reconhecimento da interrupção que estava pendente;
- clock: pulsos de sincronização;
- reset: efetua a reinicialização dos módulos.

Em alguns modelos de barramentos, como o PCI (*Peripheral Component Interconnect*), as vias de endereçamento e de dados são multiplexadas. Em canais multiplexados, as vias podem atender às funcionalidades de transporte de forma alternada, ou seja, em certos momentos, transmite-se endereços e, em outros, os dados são trafegados.

VOCÊ SABIA?

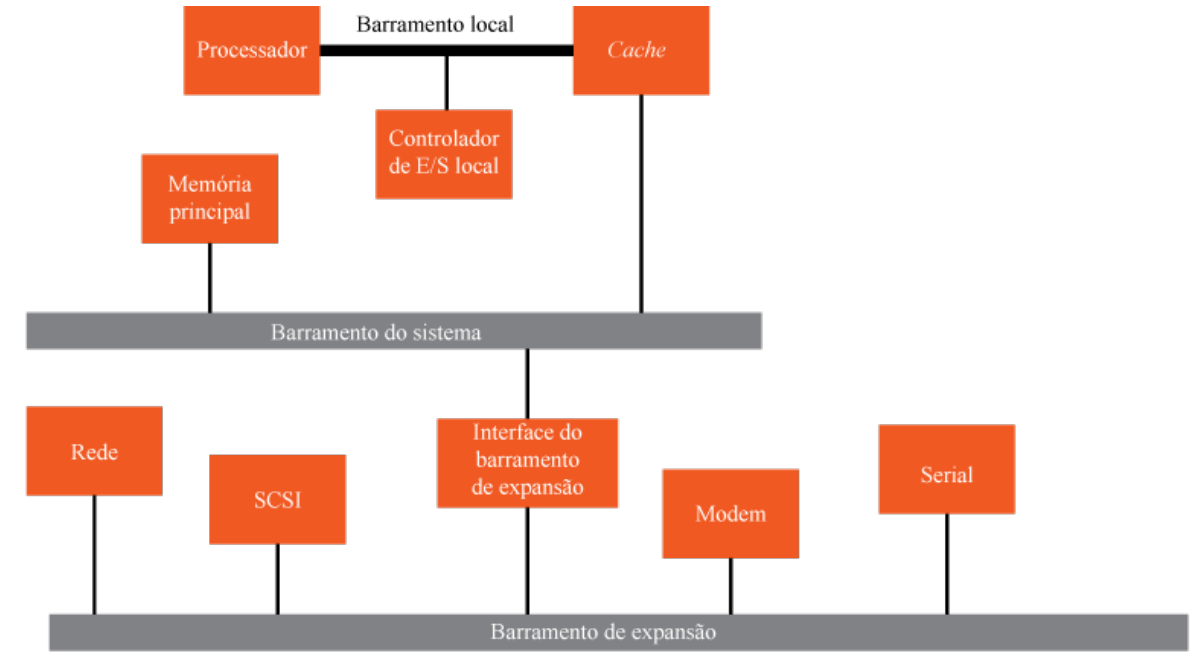
Barramentos não são específicos para realizar a interconexão apenas de módulos internos do computador. Barramentos também são encontrados em sistemas embarcados como o I^2C e o *wishbone*. Também são encontrados em máquinas do tipo HPC (*High Performance Computer*). A especialista em Teleinformática e Redes de Computadores, Kopp (2012), aborda a construção de um HPC: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/802/1/CT_TELEINFO_XX_2012_04.pdf>.

Antes de prosseguirmos abordando barramentos, convém mencionarmos algumas métricas e conceitos associados a eles:

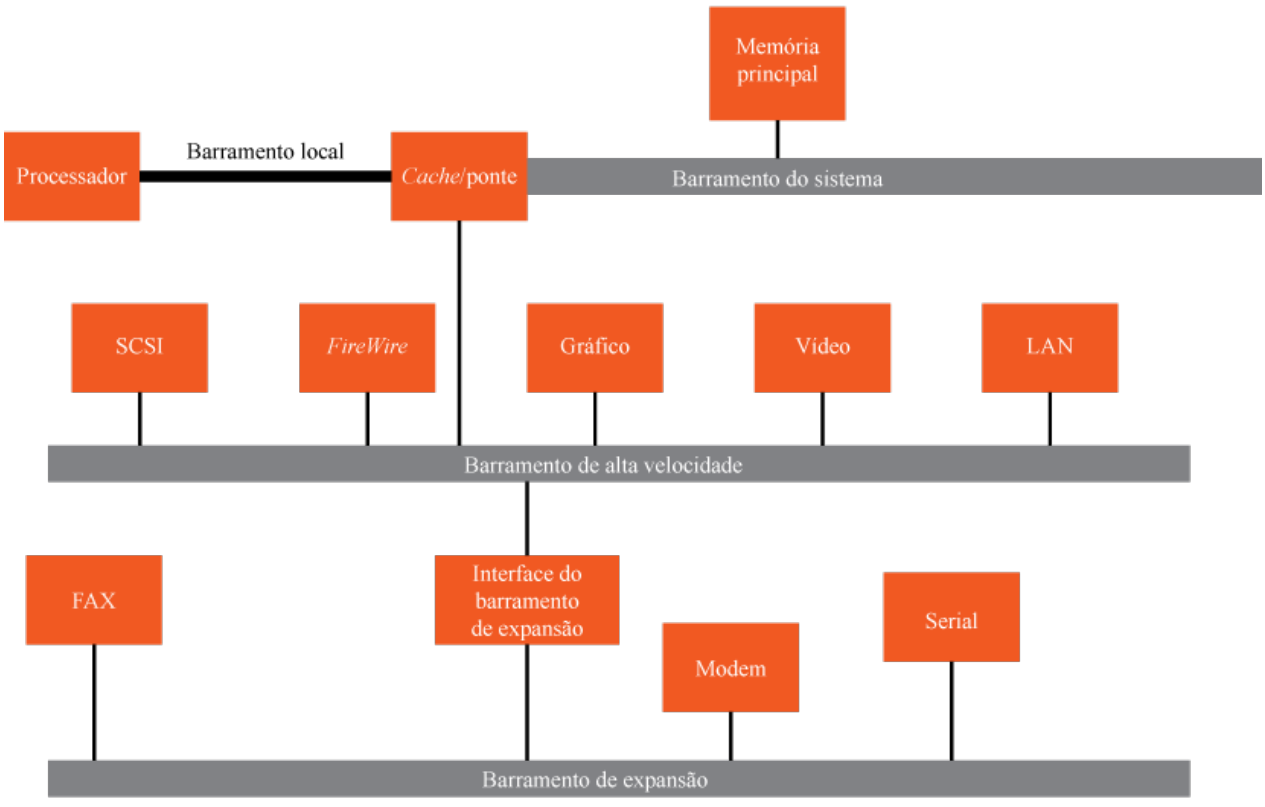
- **largura do barramento**: referencia-se à quantidade de vias paralelas para o tráfego de informações. A largura do barramento está associada ao tamanho da palavra usada pelo sistema computacional em questão – por exemplo, uma largura de 64 *bits* ou de 32 *bits*;
- **topologia**: a topologia está associada ao arranjo físico para a ligação dos módulos. Como topologias mais comuns, encontramos: barramento, anel e linha ponto a ponto;
- **frequência de trabalho**: relativo à frequência, em Hertz (Hz), do sinal de *clock* usado para a sincronização dos módulos;
- **vazão, largura de banda ou *throughput***: a largura de banda é o índice que indica a quantidade de *bits* transferidos dentro do intervalo de um segundo (bps);
- **sistema de coordenação**: indica como é realizada a coordenação pelo uso do barramento. O sistema pode ser centralizado onde há um árbitro de barramento, cuja função é receber as requisições e, usando um critério de tempo e/ou prioridade, fornecer, a um módulo requisitante, a permissão de uso do barramento. Um mecanismo bastante usado para solicitar e obter acesso ao barramento é denominado "*handshake*". Por sua vez, denomina-se controle distribuído quando não há um coordenador. Nesse caso, os próprios módulos decidem entre si qual será o detentor de uso do barramento. Para tanto, usa-se a

troca de mensagens ou a passagem de "*tokens*". Em sistemas ponto a ponto, podemos, ainda, encontrar solicitações baseadas em "*daisy-chain*".

Um computador pode ser dotado de barramentos de vários níveis, atendendo a um conjunto de módulos ou dispositivos. A utilização de uma hierarquia de barramentos permite uma utilização mais eficiente evitando-se que um tipo de fluxo não impacte negativamente em algum outro. Como exemplo, podemos ver a topologia das máquinas usuais atuais, na figura a seguir, em dois níveis de abstração: a configuração de uma máquina dita como convencional (a); e uma configuração de alto desempenho (b).



(a) Arquitetura de barramento tradicional



(b) Arquitetura de alto desempenho

Figura 6 - Configurações de um barramento: em (a), tem-se uma configuração básica e, em (b), uma configuração de alto desempenho.
Fonte: STALLINGS, 2010, p. 71.

Na figura acima, temos uma dissociação dos barramentos em função do tipo de comunicação requerida. Nota-se que o barramento do sistema tem por objetivo interligar apenas o processador ao sistema de memória (memória principal e memória *cache*) devido à alta demanda de uso. Os demais módulos estão sendo interligados por barramento de alta velocidade ou barramento de expansão. No caso, nota-se, na porção denotada por (b), que houve uma incorporação de um barramento de alto desempenho, atendendo os dispositivos mais críticos (como a placa de vídeo). Nesse caso, o tráfego do barramento de expansão fica dissociado em relação ao tráfego do barramento de alta velocidade.

Na prática, você poderá encontrar essa hierarquia de barramentos nos *chipsets* (conjunto de circuitos integrados controladores do barramento) atuais. Em tais *chipsets* podemos encontrar a "ponte norte" (*northbridge*) como elementos de interligação do processador com o sistema de memória e a placa de vídeo (por exemplo, *AGP* ou *PCI Express*). Por sua vez, os demais módulos (placa de vídeo *onboard*, controladoras de disco, etc.) são interligados pela "ponte sul" (*southbridge*).

VOCÊ QUER LER?

Quer saber mais sobre *chipsets* e barramentos? Recomendamos o artigo de Torres (2012), no endereço: <https://www.clubedohardware.com.br/artigos/placas-mae/tudo-o-que-voc%C3%AA-precisa-saber-sobre-chipsets-r34158/?nbcpage=1>.

Já que existe uma grande preocupação no transporte de informações entre o processador e a memória, dedicando, para isso, um barramento (ponte norte), então a memória também deve ser eficiente para que o tempo de resposta diante das requisições do processador seja mínimo. Veremos, a seguir, os aspectos inerentes à memória para que se possa garantir a eficiência no armazenamento e na recuperação das informações solicitadas pelo processador.

3.3 Visão geral do sistema de memória

Até o momento, você viu, neste capítulo, aspectos relacionados com o processamento em si e sobre o transporte das informações (dados, instruções e sinais de controle) pelos barramentos. Mas, como se realiza o armazenamento e recuperação das informações? Veremos esses pontos nesta seção.

3.3.1 Características dos Sistemas de Memória

Inicialmente, pode-se falar que as características da memória variam em função da sua funcionalidade e de sua localização. Porém algumas métricas e conceitos podem ser aplicados em qualquer que seja o nível de memória dentro de sua hierarquia. Com isso, segundo (STALLINGS, 2010), uma memória em específico pode ser classificada de acordo com a sua unidade de transferência, método de acesso, desempenho e tecnologia de fabricação.

Define-se unidade de transferência como sendo a forma com que a informação é tratada. A informação pode ser tratada como sendo uma palavra de *bits* ou um bloco (contendo várias palavras). A palavra geralmente relaciona-se com a largura do barramento (por exemplo, 64 *bits*) para que as informações não precisem passar por um processo de serialização ou paralelização, ou, ainda, não sejam necessários vários pulsos (ciclos) de *clock* para ler ou gravar informações frente à memória.

Alguns níveis de memória, por exemplo, a memória externa (discos rígidos), manipulam blocos de informações. Nesse tipo de memória, consegue-se gravar ou ler todo o bloco, gastando-se apenas um ciclo de memória. Essa quantidade de *bits* que poderão ser gravados ou recuperados de uma só vez se denomina unidade de transferência.

Além da unidade de transferência, um outro aspecto a ser considerado relaciona-se com as unidades endereçáveis. A quantidade de unidades endereçáveis é proporcional à quantidade de *bits* que compõem o endereço. Por exemplo, em um sistema cujo tamanho da palavra é de 16 *bits*, teremos 2^{16} linhas de endereçamento de memória (unidades endereçáveis). Caso a largura da memória seja também de 16 *bits* então, teremos: 2^{16} = 65.536 (64 k) unidades endereçáveis. Cada unidade compreende 16 *bits* (2 *bytes*), então a capacidade total da memória é de 64k * 2 *bytes* = 128 k *bytes*.

Um outro aspecto a ser considerado relaciona-se ao método de acesso. Método de acesso refere-se a como o item deverá ser localizado na memória. Podemos classificar o método de acesso em quatro tipos:

- **acesso sequencial**: para acessarmos uma informação (no caso denominado como registro), devemos percorrer, sequencialmente, todos os registros anteriores até chegarmos no demandado. Como exemplo de memória que utiliza o acesso sequencial, poderemos relacionar as unidades de fitas magnéticas;
- **acesso direto**: esse método de acesso é utilizado, por exemplo, nas unidades de discos. Para se acessar um registro, primeiro pode-se posicionar o dispositivo de leitura e gravação em uma região específica para que, em seguida, seja possível realizar um acesso sequencial dentro dessa região. Como é necessário um acesso sequencial, não se pode precisar o tempo gasto para acessar um item;
- **acesso aleatório**: em memórias com acesso aleatório, como as memórias principais (memórias RAM – *Random Access Memory*), a unidade endereçável pode ser acessada diretamente. Dessa forma, o tempo de acesso é constante e determinista, independentemente da posição na qual se encontra o item a ser manipulado;
- **memória associativa**: na técnica de memória associativa, usada, por exemplo, em alguns modelos de memória *cache*, o item não tem um endereçamento explícito, ou seja, pode estar posicionado em qualquer linha da memória. O processo de busca do item é realizado paralelamente em todas as linhas, comparando-se os seus conteúdos com um padrão de *bits* que representa a informação demandada.

A tecnologia empregada para o sistema de memória tem um forte impacto sobre a forma de manipulação das informações, custos e a performance computacional. Além de impactar diretamente na volatilidade da memória, as tecnologias de fabricação influenciam na densidade de *bits* (número de *bits* armazenados por uma área). Podemos falar sobre dois grandes tipos principais: as memórias do tipo DRAM (*Dynamic RAM*) e as SRAM (*Static RAM*).

As memórias DRAM são baseadas na utilização de capacitores como elementos básicos de armazenamento. Porém, os capacitores se descarregam com o tempo, sendo necessário um ciclo de realimentação para reativar a sua carga (*refresh*). Durante o ciclo de *refresh*, a memória fica bloqueada para leituras e escritas, deixando-a mais lenta em relação à SRAM.

Por sua vez, as SRAM usam componentes semicondutores para o armazenamento de *bits*. Isso faz com que as memórias se tornem mais rápidas em relação à DRAM (pois, além da tecnologia mais rápida, não é necessário o ciclo de *refresh*). Porém, ocupa um espaço muito maior para armazenar cada *bit* da informação.

Todos os aspectos aqui apresentados influenciam no desempenho do sistema de memória. O desempenho de uma memória está relacionado a três métricas:

- **tempo de acesso**: o tempo de acesso ou latência representa o tempo, desde a chegada da requisição ao sistema de memória, para que a informação fique disponível para a transferência;
- **tempo de ciclo de memória**: período no qual possa ser tratada uma segunda solicitação sem causar interferências ou danos às informações manipuladas na requisição anterior;
- **taxa de transferência**: tempo gasto para a completa transferência dos *n bits* demandados.

Essas métricas, conceitos e tecnologias podem ser empregados no sistema de memória, variando dentro de seu nível hierárquico.

3.3.2 Hierarquia de memória e princípios da memória cache

Até o momento, citamos alguns exemplos de locais de armazenamento dentro do sistema de memória. Mas, como seria o fluxo de informações nesses locais, já que temos funcionalidades distintas? Lembramos que tudo a ser processado (as informações e as próprias instruções) deverá ser carregado para os registradores do processador. E, também, os arquivos executáveis que se encontram no disco rígido precisam ser transferidos para a memória principal para que sejam executados. Diante dessas afirmativas, temos a seguinte constituição da hierarquia de memória:

- **registradores**: nível mais alto da hierarquia de memória. São os elementos de memória internamente ao processador, baseados em componentes semicondutores;
- **memória *cache***: nível imediatamente abaixo dos registradores. Por se tratar de uma memória de alta velocidade, tem como objetivo diminuir os acessos à memória principal. A memória *cache* pode, ainda, apresentar três níveis: L1 (*level* 1), L2 e L3. O nível L1, localizado dentro do núcleo do processador, é sincronizado com a mesma frequência do núcleo. Geralmente o nível L1 é dividido entre L1 de instruções e L1 de dados. O nível L2 encontra-se fora do núcleo, porém dentro do encapsulamento do processador. Trabalha com a metade da frequência do núcleo. Por último, a *cache* L3 está localizada na placa-mãe, operando na mesma frequência do barramento. A *cache* é implementada, assim como os registradores, com componentes baseados em semicondutores;
- **memória principal**: componente de memória responsável por conter os processos em execução. Junto com a memória secundária, forma a memória virtual (TANENBAUM, 2013). A memória principal é implementada, geralmente, por componentes capacitivos;
- **memória secundária**: representa o nível mais baixo da memória (mais distante do processador). A memória secundária pode ser exemplificada pelo disco rígido – dispositivo que usa o meio magnético como forma de armazenamento.

Na medida em que se aumenta o nível, ou seja, aproxima-se do processador, tem-se o incremento da performance computacional, diminuição da densidade (capacidade de armazenamento de menos *bits* por área de superfície), menor capacidade de armazenamento, maior custo por *bit*.

Dentre os níveis da hierarquia de memória, daremos maior foco, nesse momento, à memória *cache* devido a sua importância para o aumento da performance computacional do sistema.

Como mencionado antes, o objetivo da cache é diminuir o número de acessos à memória principal. Com isso, atenuam-se os atrasos frente à busca ou gravação das informações junto ao sistema de memória.

O funcionamento da memória *cache* segue dois princípios: o princípio da localidade espacial e o princípio da localidade temporal. De acordo com Hennessy (2014), eles se definem da seguinte forma:

- **princípio da localidade espacial**: quando um processo usa alguma informação, a probabilidade de usar outras informações próximas é alta;
- **princípio da localidade temporal**: a chance de uma informação utilizada recentemente ser utilizada em um futuro próximo é elevada.

Então, a memória *cache*, usando os princípios dessas duas localidades, armazena as informações "estratégicas" – tentando-se, dessa forma, evitar acessos desnecessários à memória principal.

VOCÊ QUER VER?

No vídeo "Políticas de Atualização da Cache", você pode conhecer um pouco mais sobre o funcionamento da memória *cache*, a partir dos algoritmos de substituição (UNB-GAMA, 2015): <https://www.youtube.com/watch?v=1wh5gczKNbc>.

Agora, já temos informações suficientes para nos perguntar: como a *cache* é estruturada e organizada? Como as informações são armazenadas e buscadas? A seguir, responderemos a essas perguntas e algumas outras que poderiam aparecer em sua mente.

3.4 Elementos do projeto da memória *cache*

Para entender como ocorre a manipulação da memória *cache* e determinar seu funcionamento, vamos nos guiar por algumas questões.

- Aonde colocar o item dentro da *cache*?
- Como localizar a item dentro da *cache*?
- Como realizar as trocas dos blocos armazenados na *cache*?
- Quais ações ocorrem quando o conteúdo de um bloco armazenado na *cache* é atualizado?

As três primeiras questões serão respondidas quando abordarmos os tipos de mapeamentos. Por sua vez, a última questão será tratada na política de escrita.

3.4.1 Mapeamento associativo e direto

Antes de prosseguirmos na descrição do tipo de mapeamento, convém mencionar que a *cache* manipula blocos de informações (compostos por vários dados ou várias instruções vizinhas).

O modelo de mapeamento refere-se à localização do bloco dentro da *cache*. No mapeamento associativo, um bloco não tem uma posição específica dentro da *cache*, ou seja, poderá estar em qualquer linha endereçável. Com essa estratégia, a *cache* torna-se mais poderosa, pois permite-se uma maior reutilização das informações armazenadas. Porém o custo por *bit* é muito dispendioso pelo fato de que, em cada linha, deve ser associado um circuito de comparação. A função desse circuito de comparação consiste em procurar o item solicitado pelo processador comparando-se o endereço requisitado pelo processador com o conteúdo armazenado – figura abaixo, item (a).

Para localizar um item solicitado pelo processador, o sistema de gerenciamento da memória *cache* acompanha o seguinte fluxo:

- ao receber a demanda realizada pelo processador, todos os circuitos comparadores realizam a comparação do valor recebido do processador com todos os conteúdos localizados no campo "*tag*";
- o comparador que retornar "*true*" como resultado da comparação envia um sinal a um circuito de seleção para que o campo "valor" seja acessado;
- será, também, acessado o campo "válido" da linha selecionada para que, caso o conteúdo seja válido, o conteúdo seja despachado para o processador, denotando, nesse caso, o evento "*hit*" (*hit* = conteúdo encontrado na *cache*). Caso o item não seja encontrado, ocorrerá o evento "*miss*", fazendo com que seja ativado o nível de memória imediatamente inferior.

Convém salientar que esse procedimento é executado no caso de leituras na cache. A posição do item no processo de gravação em memória associativa segue apenas dois critérios:

- caso exista alguma posição vaga, escreva o item nesta posição;
- caso contrário, executa-se a política de troca de bloco, que será descrita adiante.

Por sua vez, o mapeamento direto vem em direção contrária aos custos e da complexidade do *hardware* – figura a seguir, item (b). A sua estratégia de busca segue os seguintes passos:

- ao receber o endereço do processador, acessa-se a linha que corresponda à operação de resto da divisão do endereço pela quantidade de linhas da *cache*. Em termos práticos, a operação de divisão é realizada, simplesmente, coletando-se os *n bits* menos significativos do endereço. Por exemplo, supondo que o processador gerou o endereço 35 e que a *cache* possua oito linhas. Nesse caso, obtém-se a linha da *cache* pelos $\log_2^8 = 3$ *bits* menos significativos. O valor $35_{(10)}$ poderá ser expresso como $00100011_{(2)}$. Nesse caso, a linha a ser acessada é a 3 ($011_{(2)} = 3_{(10)}$). Como todo bloco a ser alocado nesta linha contém os mesmos *n bits* menos significativos, então, o campo "*tag*" precisará armazenar apenas os (*k*–*n*) *bits* mais significativos, sendo *k* o tamanho da palavra manipulada pelo processador;
- na mesma linha, acessa-se o campo "válido". Caso o item seja válido, então ocorre o evento "*hit*", caso contrário, o evento "*miss*" (de forma análoga ao que foi descrito anteriormente).

No caso do mapeamento direto, quando é preciso escrever o item na *cache*, ocorre a substituição do conteúdo original pelo novo, demandado pelo processador.

O mapeamento direto tem uma grande desvantagem quando, por exemplo, dois itens que possuam o mesmo padrão de localização, estejam sendo demandados de forma alternada – como por exemplo, dois itens acessados durante um laço de repetição. Neste caso, sempre ocorrerá o evento "*miss*" pois um item sempre substituirá o outro.

Para atenuar esse problema, existe um terceiro mapeamento que, na verdade, é a fusão dos dois primeiros: trata-se do conjunto associativo. No conjunto associativo, item (c) da figura abaixo, ocorre uma replicação das colunas do mapeamento direto. Nessa abordagem, ao receber uma demanda do processador, a sistema de *cache* acessa a linha correspondente como no mapeamento direto. Com a linha selecionada, aplica-se, para a leitura ou para a escrita do bloco, os passos como descritos no mapeamento associativo. Esse tipo de mapeamento é o mais usado nos processadores atuais. Por exemplo, o Intel Core i7 Haswell utiliza a configuração "*8-way set associative*" (replicação de colunas com fator 8) em suas *caches* L1 e L2 (TELES, 2004).

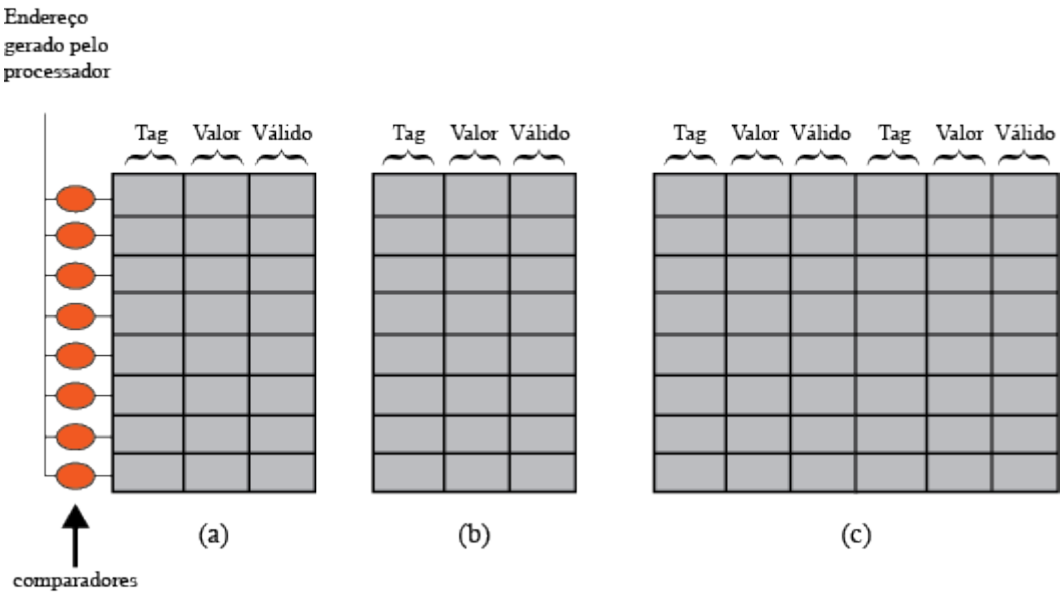


Figura 7 - Formas de mapeamento da memória cache, sendo que em (a) temos o mapeamento associativo, em (b), o mapeamento direto e, por fim, em (c), o conjunto associativo.
Fonte: Elaborada pelo autor, 2018.

Na figura acima, você pode notar o campo "*tag*" que corresponde à referência do endereço do bloco armazenado. O campo "válido" tem por objetivo informar se o conteúdo da linha não pertence a algum processo já finalizado ou algum valor aleatório produzido no momento da iniciação do sistema.

3.4.2 Política de escrita

Ao se falar sobre política de escrita, podemos referenciar a dois aspectos: quando um item novo precisa ser gravado ou quando o item é alterado na memória *cache*.
Quando um item novo é demandado pelo processador e acontece um evento do tipo "*miss*", então significa que o bloco precisa ser lido dos níveis inferiores de memória e gravado na *cache*. Sendo assim, os mapeamentos associativos e o conjunto associativo executam o seguinte algoritmo:

- caso exista posição disponível, salve o item nesta posição;
- caso contrário, algum bloco previamente carregado deverá deixar a *cache*. Mas qual item escolher? Adota-se um critério de escolha retirando-se, por exemplo, o bloco que teve o acesso mais remotamente (LRU – *Least Recently Used*), o bloco usado com menos frequência (LFU – *Least Frequently Used*) ou o bloco mais antigo (FIFO – *First In/First Out*).

Como mencionamos, o outro aspecto relacionado à política de escrita refere-se quando há a atualização no valor do conteúdo do bloco presente na *cache*. Nesse caso, o sistema de memória pode adotar dois critérios:

- *write back*: nesse tipo de política, qualquer alteração na *cache* é repassada aos níveis inferiores de memória, deixando todos os níveis sincronizados com o mesmo valor de conteúdo;
- *write through*: na estratégia de *write through*, ocorre a atualização do nível inferior de memória apenas quando o item deixa a *cache* para ceder espaço a um novo item demandado pelo processador.

CASO

Atualmente, em função do aparecimento dos processadores *multicore* (vários núcleos), apenas a política de escrita *write through* não basta. Supondo a existência de dois núcleos: N_1 e N_2. Suponha que um bloco B será usado, inicialmente, por N_1, que fará, em seguida, uma atualização de valor. Usando *write through*, a modificação realizada em N_1 irá refletir, prontamente, nos níveis inferiores de memória. Nesse momento, suponha que o núcleo N_2 utilizará o mesmo bloco B – coletará e armazenará em sua *cache* L_1 o mesmo valor armazenado nos níveis inferiores na *cache* L_1 de N_1. Porém, em N_2, ao modificar o valor de B, fará modificar apenas o valor nos níveis inferiores, permanecendo o valor na L_1 de N_1 inalterado. Por consequência, as *caches* L_1 de N_1 e N_2 ficarão incoerentes. Diante desse fato, foram criados protocolos de coerência de *cache* tais como o protocolo MESI (*Modified Exclusive Shared Invalid*) e MOESI (*Modified Owner Exclusive Shared Invalid*). O Intel Core i7 usa uma variação do MESI (que fora introduzido no Pentium) chamada MESIF (*Modified Exclusive Shared Invalid Forward*).

Você pode estar se perguntando: quando se deve usar uma ou outra política de escrita? Para responder a essa pergunta, abstraia um ambiente que apresenta apenas um processador que tenha um núcleo. Neste caso, não há nenhuma possibilidade de que o valor de um bloco seja alterado por outros nós de processamento. Sendo assim, neste ambiente não é necessário o *write through*. Agora abstraia um ambiente que um certo item possa ser buscado por vários núcleos. O *write through* ajuda a manter a coerência dos valores armazenados na hierarquia de memória.

Síntese

Chegamos ao final de mais uma etapa em nosso caminho de conhecimento. Neste capítulo, passamos pelos aspectos de processamento das instruções, mais especificamente sobre a unidade de controle do processador e a divisão das instruções em micro-operações. Pudemos ver, também, a parte de interconexão dos módulos do computador, representada pelo barramento e, ao final, os conceitos acerca do sistema de memória. Dentro do sistema de memória, pudemos dar um enfoque maior no sistema de memória *cache*.

Esperamos que este conteúdo lhe dê condições de tomar decisões nos quesitos relacionados ao parque de informática da empresa e implementar soluções de forma mais eficiente.

Neste capítulo, você teve a oportunidade de:

- identificar e descrever aspectos referentes à execução das instruções, sob o ponto de vista da unidade de controle, e às micro-operações;
- descrever e interpretar os mecanismos para o tratamento das interrupções;
- situar, esquematizar as funcionalidades e os conceitos relacionados aos barramentos de interconexão do computador;
- definir, identificar e reproduzir as técnicas e os modelos usados pelo sistema de memória – mais especificamente, de memória *cache*.

Bibliografia

HENNESSY, J. L.; PATTERSON, D. A. **Arquitetura de Computadores**: uma Abordagem Quantitativa. 5.ed. Rio de Janeiro: Campus, 2014.

KOPP, E. M. **Construção de um Cluster HPC para Simulações de CFD**. Monografia (Especialização em Teleinformática e redes de computadores) – Programa de Pós-Graduação em Tecnologia, Universidade Tecnológica Federal do Paraná. Curitiba, 2012. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/*bits*tream/1/802/1/CT_TELEINFO_XX_2012_04.pdf>. Acesso em: 03/07/2018.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. **Segundo Período - *Electronic Delay***. Museu Virtual do Sintetizador – Universidade Federal do Rio Grande do Sul (MVS-UFRGS). Publicado em fevereiro de 2010. Disponível em: <http://www.ufrgs.br/mvs/Periodo02-1949-ElectronicDelay.html>. Acesso em: 03/07/2018.

RAISCH, D. **Computadores Híbridos, a próxima fronteira da computação**. Portal IMB Community, publicado em 16/09/2010. Disponível em: <https://www.ibm.com/developerworks/community/blogs/tlcbr/entry/computadores_hibridos_a_proxima_fronteira_da_computacao?lang=en>. Acesso em: 03/07/2018.

STALLINGS, W. **Arquitetura e Organização de Computadores**. 8. ed. São Paulo: Pearson Prentice Hall, 2010. Disponível na Biblioteca Virtual Laureate: <https://laureatebrasil.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_198689_1&content_id=_4122211_1&mode=reset>. Acesso em: 21/06/2018.

TANENBAUM, A. S. **Organização Estruturada de Computadores**. 6. ed. São Paulo: Pearson Prentice Hall, 2013. Disponível na Biblioteca Virtual Laureate: <https://laureatebrasil.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_198689_1&content_id=_4122211_1&mode=reset>. Acesso em: 21/06/2018.

TELES, B. **O Processador Intel Core i7**. Universidade Estadual de Campinas. Instituto de Computação. 2004. Disponível em: <http://www.ic.unicamp.br/~ducatte/mo401/1s2009/T2/042348-t2.pdf>. Acesso em: 03/07/2018.

TORRES, G. **Tudo o que você precisa saber sobre chipsets**. Portal Clube do Hardware, publicado em 02/08/2012. Disponível em: <https://www.clubedohardware.com.br/artigos/placas-mae/tudo-o-que-voc%C3%AA-precisa-saber-sobre-chipsets-r34158/?nbcpage=1>. Acesso em: 03/07/2018.

UNIVERSIDADE DE BRASÍLIA – FACULDADE DO GAMA (UNB-GAMA). **Políticas de Atualização da Cache**. Canal FAC UNB, YouTube, publicado em 17/06/2015. Disponível em <https://www.youtube.com/watch?v=1wh5gczKNbc>. Acesso em: 03/07/2018.