Capítulo 5

O Sub-sistema de Entrada/Saída

Depois do processador e do sub-sistema de memória, este capítulo examina o terceiro componente na arquitetura de um computador, o sub-sistema de entrada e saída (e/s). Neste sub-sistema estão incluídas as *interface*s de e/s, através das quais os dispositivos periféricos são conectados ao sistema.

Este capítulo inicia descrevendo como processador e *interface*s de e/s se comunicam, a organização típica de uma *interface* de e/s, e como o processador exerce controle sobre um dispositivo periférico através de uma *interface* de e/s. Em seguida, são apresentadas as principais técnicas de transferência de dados em operações de e/s. Por último, são discutidos alguns aspectos relacionados com barramentos de e/s.

5.1 A Interação entre Processador e *Interface*s de E/S

Em sistemas como computadores pessoais e estações de trabalho as *interface*s de e/s estão ligadas ao processador através de barramentos de endereço, dados e controle, de maneira semelhante à conexão entre memória principal e processador. A organização típica de um computador incluindo o sub-sistema de e/s é mostrada na Figura 5.1.

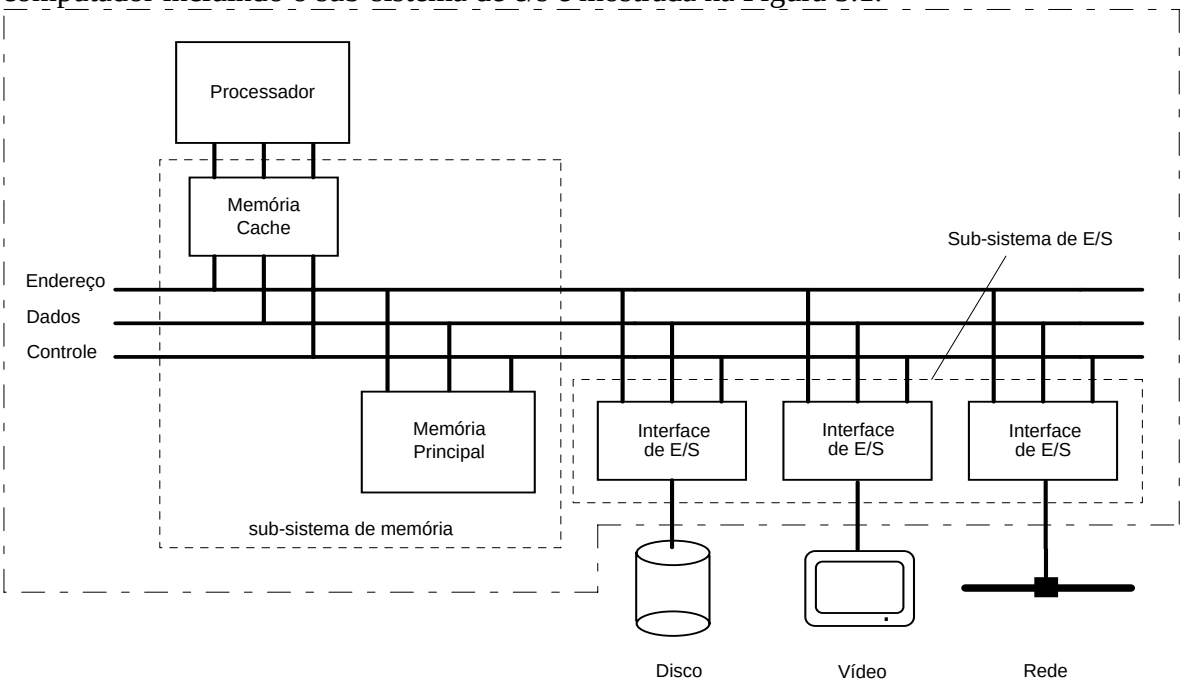

Figura 5.1. *Arquitetura de um computador incluindo o sub-sistema de e/s.*

Assim como acontece em relação à memória principal, o processador realiza acessos de leitura ou de escrita a uma *interface* de e/s. Em um acesso de leitura, o processador obtém um dado recebido do dispositivo periférico conectado à *interface*, ou então uma informação de estado sobre uma operação de e/s em andamento ou recém-completada. Em um acesso de escrita, o processador fornece à *interface* um dado que deve ser enviado ao dispositivo periférico, ou então o código de um comando que inicia uma operação de e/s ou uma operação de controle sobre o dispositivo periférico.

Nos acessos às *interface*s, o processador executa ciclos de barramento semelhantes aos descritos no capítulo anterior. Cada *interface* de e/s é identificada por um endereço único. Em um acesso de leitura, o processador coloca o endereço da *interface* no barramento de endereço e ativa um sinal de leitura. Após um certo intervalo de tempo, a *interface* coloca a informação desejada no barramento de dados. O processador finaliza o ciclo de barramento lendo a informação presente no barramento de dados e retirando o endereço e o sinal de controle.

Em um acesso de escrita, o processador coloca o endereço da *interface* e o dado nos respectivos barramentos, e ativa um sinal de escrita. A *interface* selecionada armazena a informação presente no barramento de dados. No final do ciclo de barramento, o processador retira o endereço e o dado e desativa o sinal de controle. Assim como nos ciclos de barramento com a memória, todos estes eventos são comandados pelo processador e ocorrem em sincronismo com o sinal de *clock*.

5.2 Organização de uma *Interface* de E/S

A principal função de uma *interface* de e/s é tornar transparente para o processador os detalhes de operação e controle dos dispositivos periféricos. Podemos considerar que uma *interface* de e/s é organizada em duas partes, como mostra a Figura 5.2.

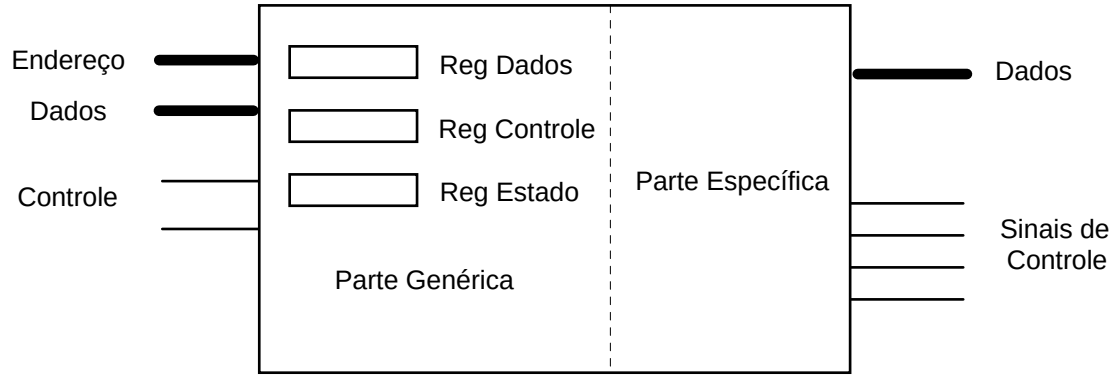

Figura 5.2. *Organização típica de uma interface de e/s.*

A parte genérica, como o próprio nome indica, é semelhante entre os diferentes tipos de *interface*s de e/s. É esta porção da *interface* que é vista pelo processador. Em geral, na parte genérica existem alguns registradores, cujo número e função depende em parte do tipo de periférico acoplado à *interface*. No entanto, como mostra a Figura 5.2, na maioria das *interfaces* a parte genérica inclui pelo menos um **registrador de dados**, um **registrador de controle** e um **registrador de estado**. O acesso a cada um destes registradores é feito pelo processador através de um endereço de e/s diferente.

O registrador de dados é usado para as transferências de dados entre o processador e o dispositivo periférico. Em uma operação de saída, o processador escreve um dado neste registrador e a *interface* se encarrega de enviá-lo para o periférico. No sentido contrário, em uma operação de entrada, a *interface* recebe um dado do periférico e o armazena no registrador de dados. O processador executa então um acesso de leitura à *interface* o obtém o dado depositado no registrador.

O processador usa o registrador de controle para enviar comandos à *interface*. Este comando é enviado sob a forma de um código. Cada *interface* possui um repertório de comandos próprio. Quando o processador escreve um comando no registrador de controle, a *interface* interpreta o código do comando e executa a operação solicitada, que pode ser uma operação interna à *interface* ou sobre o periférico a ela conectado. Finalmente, o registrador de estado é usado para veicular informações gerais sobre uma operação de e/s. Tipicamente, este registrador possui *bits* para indicar o término de uma operação e para indicar condições de erro que eventualmente possam acontecer durante a operação.

A parte específica interage diretamente com o periférico, e por isso ela difere bastante entre os diferentes tipos de *interface*s. No entanto, apesar das diferenças, a parte específica na maioria das *interface*s possui dois conjuntos de sinais. Um deles é a própria via através da qual são transferidos os dados entre a *interface* e o periférico. O outro conjunto é formado pelos sinais usados no controle do periférico.

5.3 Técnicas de Transferência de Dados

Em geral, uma operação de e/s envolve a transferência de dados entre a memória e a *interface* de e/s. Existem basicamente três técnicas para realizar a transferência de dados: ***polling***, **interrupção** e **acesso direto à memória**. Cada uma destas técnicas estão descritas a seguir.

E/S com *Polling*

Na e/s com *polling*, o processador controla toda a transferência de dados entre a memória e a *interface* de e/s. Para entender como é o procedimento desta técnica, considere o exemplo de

uma operação de escrita em um setor de disco. Suponha que a *interface* controladora de disco é semelhante àquela mostrada na Figura 5.2. Normalmente, o registrador de estado possui um *bit*, chamado ***done bit***, que é desativado quando um dado é escrito no registrador de dados, sendo ativado quando este dado é escrito no setor do disco. O diagrama na Figura 5.3 mostra como acontece a escrita de um setor de disco usando-se e/s com *polling*.

Figura 5.3. *Exemplo de e/s com polling.*

Após escrever um dado no registrador de dados, o processador lê o registrador de estado e testa o *done bit*, para verificar se o mesmo já foi escrito no setor do disco. Este teste do *bit* de estado é chamado *polling*. O processador continua realizando o *polling* até encontrar o *done bit* ativado, o que indica que o dado já foi escrito no setor do disco. Quando isto acontece, e se ainda existe algum dado a ser enviado, o processador escreve o novo dado no registrador de dados e reinicia o *polling*. Este ciclo é repetido até que todos os dados tenham sido escritos no setor do disco.

A principal vantagem da e/s com *polling* é a sua simplicidade. No entanto, esta técnica possui a desvantagem de que o processador fica dedicado à operação de e/s. Isto pode ser extremamente ineficiente, sob o ponto de vista da utilização do processador. Considere uma operação de envio de um bloco de caracteres para uma impressora. O tempo de

impressão de um caracter é infinitamente maior que o tempo de execução de uma instrução. Manter o processador em *polling* durante o tempo de impressão de cada caracter é um desperdício, já que durante este intervalo de tempo o processador poderia executar alguns milhões de instruções de um outro programa. Devido ao fato que o processador fica dedicado à operação de e/s até o seu término, o uso da técnica de e/s com *polling* é restrito apenas a sistemas onde apenas um programa pode se encontrar em execução a cada instante.

E/S com Interrupção

Na e/s com *polling*, o processador fica dedicado à operação de e/s porque ele é o responsável por determinar quando um novo dado pode ser transferido entre a memória e a *interface* de e/s. O mesmo não acontece na e/s com interrupção. Nesta técnica, a *interface* é responsável por notificar o processador quando um novo dado pode ser transferido. Enquanto a e/s com *polling* é uma técnica puramente de *software*, a e/s com interrupção requer um suporte de *hardware*. A *interface* deve gerar um **sinal de interrupção**, através do qual ela notifica o processador quando uma operação de e/s foi concluída.

Considere novamente o exemplo da operação de escrita de um setor de disco. O diagrama na Figura 5.4 mostra como esta operação é realizada através de e/s com interrupção. A operação é dividida em duas fases. Na fase de disparo da operação, o processador envia para a *interface* o comando, o número da trilha e do setor. Ao final da fase de disparo, o processador passa a executar uma outra atividade qualquer, por exemplo, parte de um outro programa.

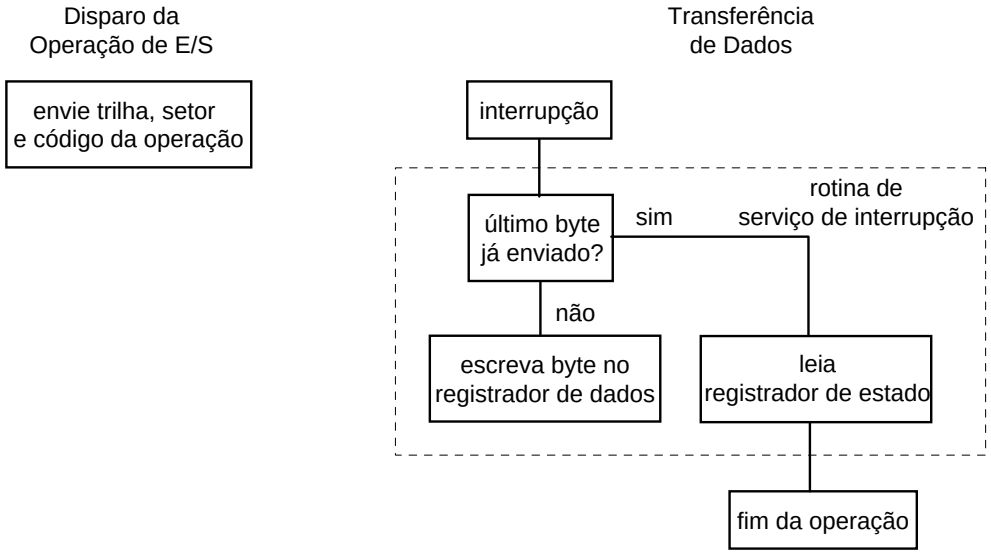

Figura 5.4. *Exemplo de e/s com interrupção.*

A *interface* inicia a fase de transferência de dados fazendo um pedido de interrupção ao processador, através do sinal de interrupção. Ao receber o pedido de interrupção, o processador suspende a execução do programa corrente e passa a executar uma rotina especial, chamada **rotina de serviço de interrupção** (também chamada *device driver* ou *device handler*). Nesta rotina, o processador verifica inicialmente se o último dado já foi enviado. Se este é o caso, o processador conclui a escrita do setor do disco lendo o registrador de estado da *interface*. Caso contrário, o processador envia um novo dado e retorna para o programa que se encontrava em execução.

Durante a fase de transferência de dados, a *interface* faz um pedido de interrupção a cada dado escrito no setor do disco. O processador responde ao pedido de interrupção executando a rotina de serviço e enviando um novo dado. Isto se repete até que todos os dados tenham sido escritos no setor do disco. Normalmente, a *interface* de disco conhece o tamanho do setor e mantém uma contagem dos dados já recebidos, de forma que ela pode determinar quando deve encerrar a seqüência de pedidos de interrupção.

Em um sistema é comum existirem várias *interface*s diferentes que fazem pedidos de interrupção ao processador. Cada *interface* deve ser atendida por uma rotina de serviço de interrupção específica para aquela *interface*. Assim, ao receber um pedido de interrupção, o processador deve determinar qual a rotina de serviço a ser executada. Além disso, quando duas ou mais *interface*s fazem pedidos de interrupção simultâneos, é necessário decidir qual o pedido de interrupção que será atendido. Estas duas funções são suportadas por um componente do sub-sistema de e/s, chamado **controlador de interrupção** (*interrupt controller*).

Como mostra a Figura 5.5, o sinal de interrupção de cada *interface* é ligado ao controlador de interrupção. O controlador de interrupção atribui um número e uma prioridade a cada um destes sinais. Quando um pedido de interrupção acontece, o controlador de interrupção envia para o processador, através do barramento de dados, o número do pedido. No caso de dois ou mais pedidos simultâneos, o controlador decide qual é o pedido com maior prioridade e envia para o processador o número correspondente.

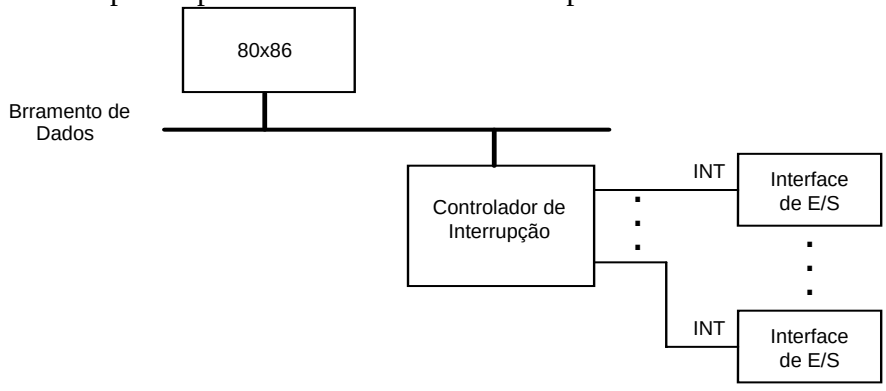

Figura 5.5. *O controlador de interrupção*.

O processador usa o número recebido do controlador para indexar uma tabela armazenada na memória, chamada **tabela de vetores de interrupção** (*interrupt vector table*). Cada entrada desta tabela contém o ponteiro, ou **vetor**, para uma rotina de serviço. Ao receber um número de interrupção *n*, o processador lê o vetor contido na posição *n* da tabela e passa a executar a rotina de serviço de interrupção apontada por este vetor.

Na e/s com interrupção, o processador não fica dedicado à operação de e/s. O processador é alocado somente quando realmente deve ser transferido um dado entre a memória e a *interface*, resultando em uma utilização mais eficiente do processador. No entanto, esta técnica apresenta uma desvantagem quanto à velocidade de transferência dos dados. Note que a transferência de um dado envolve a arbitração pelo controlador de interrupção, a comunicação entre o controlador e o processador, o acesso à memória para a leitura do vetor de interrupção e finalmente o desvio para a rotina de serviço. Todas estas etapas acrescentam um retardo antes que o dado seja realmente transferido. Este retardo é chamado de **tempo de latência de interrupção** (*interrupt latency time*).

Em alguns tipos de periféricos, a taxa de transferência de dados entre o periférico e a *interface* é muito alta, ou em outras palavras, o intervalo de tempo entre a transferência de dois dados consecutivos entre o periférico e a *interface* é muito pequeno. Devido ao tempo de latência, o intervalo de tempo entre acessos do processador à *interface* pode tornar-se maior que o intervalo de tempo com que os dados chegam à *interface*. Se isto acontece, um novo dado chega à *interface* antes que o processador leia o dado anterior, e assim o dado anterior é perdido.

Na realidade, o que contribui para aumentar o tempo de latência é o fato de que o processador ainda é o responsável por controlar a transferência de dados. Para atender periféricos com alta taxa de transferência, usa-se a técnica de e/s com acesso direto à memória, onde o processador não participa da fase de transferência de dados. Esta técnica é analisada a seguir.

E/S com Acesso Direto à Memória

Na e/s com DMA (*Direct Memory Access*) um componente do sub-sistema de e/s, chamado **controlador de DMA,** é responsável por transferir os dados entre a memória e a *interface* de e/s.

A Figura 5.6 mostra como o controlador de DMA é ligado ao resto do sistema.

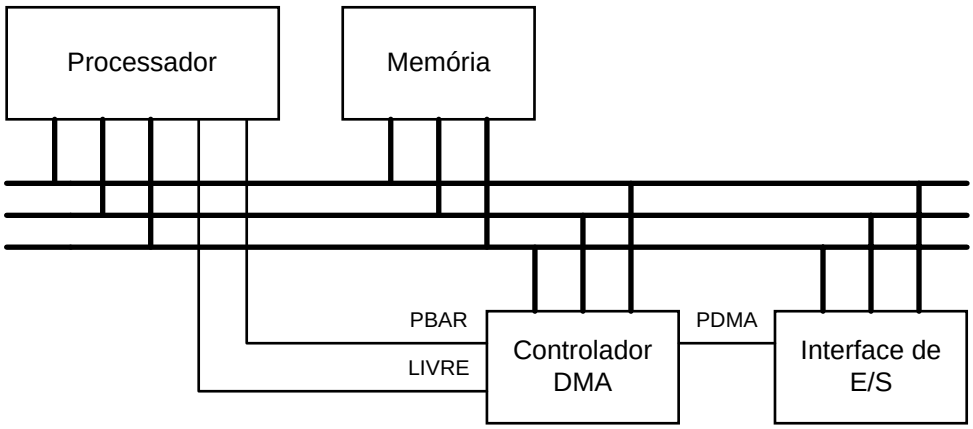

Figura 5.6. *Sistema com controlador de DMA.*

Considere novamente o exemplo da operação de escrita de um setor de disco. Na fase de disparo da operação, o processador informa ao controlador de DMA o número de dados a serem transferidos, o endereço do primeiro dado e o sentido da transferência (no caso do exemplo, o sentido de transferência é da memória para a *interface* de e/s). Em seguida, o processador envia para a *interface* controladora de disco o número de trilha, o número de setor e o comando da operação.

O processador participa apenas da fase de disparo. Na fase de transferência de dados, o controlador de DMA assume o controle dos barramentos para realizar a transferência entre a memória e a *interface*. Para tanto, o controlador de DMA coloca o processador em um estado, chamado *hold state*, no qual o processador fica impedido de iniciar ciclos de barramento. Mais detalhadamente, a fase de transferência de dados envolve os seguintes passos:

1. após receber o comando do processador, a *interface* de disco faz um **pedido de DMA** ao controlador de DMA através do sinal PDMA. Por sua vez, o controlador faz um pedido de barramento ao processador, através do sinal PBAR. Ao liberar os barramentos, o processador responde ativando o sinal LIVRE, indicando ao controlador de DMA que este já pode usar os barramentos.
2. controlador de DMA coloca no barramento de dados o endereço do primeiro dado e ativa o sinal de leitura de memória. A memória responde colocando o dado endereçado no barramento de dados. O controlador de DMA ativa o sinal de escrita em *interface* de e/s, fazendo com que a *interface* de disco capture o dado presente no barramento de dados.
3. ao escrever o dado no setor do disco, a *interface* faz um novo pedido de DMA. O controlador de DMA inicia uma nova transferência, colocando o endereço do próximo dado no barramento de endereço e ativando os sinais de controle apropriados. Este

passo se repete até que todos os dados tenham sido transferidos. Ao concluir a última transferência, o controlador de DMA retira o pedido de barramento, permitindo que o processador volte à operação normal.

Note que na e/s com DMA a transferência de cada dado envolve apenas uma leitura de memória e uma escrita de *interface* de e/s, realizadas pelo próprio controlador de DMA. A e/s com DMA efetivamente elimina o tempo de latência associado a cada dado transferido, que existe na e/s com interrupção. Isto permite que a e/s com DMA atinja taxas de transferência bem maiores que as técnicas de e/s que envolvem o controle do processador.

Em geral, é possível ter várias *interface*s de e/s operando com a técnica de acesso direto à memória. Para tanto, o controlador de DMA possui várias entradas para pedido de DMA. O controlador de DMA associa a cada uma destas entradas um conjunto independente de registradores para armazenar o número de dados a serem transferidos, o endereço inicial e o sentido da transferência. Um grupo de sinais de controle com seus respectivos registradores formam o chamado **canal de DMA**. O controlador de DMA se encarrega de arbitrar entre *interface*s que fazem pedidos de DMA simultâneos, usando um esquema de prioridades atribuídas aos canais de DMA.

5.4 Padrões de Barramentos

A Figura 5.1 mostra a arquitetura de sistema típica de computadores, onde processador, memória principal e *interface*s de e/s estão interligados através de um conjunto de três barramentos. Estes barramentos são chamados coletivamente de **barramento de sistema**. Algumas características do barramento de sistema, tais como largura do barramento de endereço e do barramento de dados, são determinadas pelo processador. Outras características estão relacionadas com o sub-sistema de e/s, como por exemplo, o número de sinais de interrupção e o número de canais de DMA disponíveis no barramento de controle.

Os três tipos de barramento são denominados processador-memória, entrada e saída e *backplane*. Os barramentos processador-memória são curtos, em geral extremamente rápidos, e têm uma relação muito forte com o sistema de memória, de modo a maximizar a banda passante memória-processador. Os barramentos de entrada e saída, em contrapartida, são mais longos, podem ter muitos dispositivos conectados a ele, e precisam atender a uma ampla faixa de bandas passantes relativas aos dispositivos periféricos. Esses barramentos não necessariamente têm interface direta com a memória. Os barramentos do *backplane* são projetados de modo a permitir que processadores, memória e dispositivos de entrada e saída possam coexistir num único barramento. Eles balanceiam as demandas de comunicação processador-memória com as demandas de dispositivos de entrada e saída-memória.

Os barramentos processador-memória são projetados especificamente para um determinado projeto, enquanto os barramentos de entrada e saída e *backplane* podem ser usados para várias máquinas, por que na maioria das vezes são padronizados.

Existem diferentes esquemas de implementação da comunicação no barramento: o síncrono e o assíncrono. Um barramento síncrono deve incluir um *clock* em suas linhas de controle e um protocolo de comunicação relacionado com o *clock*. Um barramento assíncrono não inclui um *clock.* Por isso, ele pode acomodar uma grande quantidade de dispositivos. Para coordenar a transmissão de dados, o barramento assíncrono também deve utilizar um protocolo de comunicação.

5.5 Projeto de um Sistema de Entrada e Saída

Existem dois tipos principais de restrições que devem ser consideradas num projeto de um sistema de entrada e saída, a latência e a banda passante. Em ambos os casos o padrão do tráfego de dados afeta o projeto e a análise. A latência deve ficar dentro de limites aceitáveis para se completar uma operação de entrada e saída. Com relação à banda passante, dada uma carga de trabalho o sistema deve se manter balanceado.

A abordagem geral do projeto de um sistema de entrada e saída é a seguinte:

1. Encontre o componente do sistema de entrada/saída que impõe a maior restrição ao projeto. Dependendo da carga de trabalho, este componente pode ser qualquer um, processador, memória, barramentos ou os próprios dispositivos;
2. Configure o componente de modo que ele sustente a banda passante desejada; e
3. Determine as necessidades dos demais componentes do sistema e configure-os para sustentar a banda passante.

5.6 Medidas de Desempenho de Mecanismos de Entrada e Saída

A avaliação de um sistema de entrada e saída deve considerar diversos fatores. O desempenho é um desses fatores. O desempenho é medido pela taxa na qual o sistema pode transferir dados e pela latência. A taxa de transferência depende da freqüência do *clock*, e normalmente é expressa em MB/s. O desempenho depende de todos os elementos do sistema que estão entre o dispositivo e a memória, incluindo o sistema operacional. Existem vários exemplos de *benchmarks* para a determinação do desempenho.

Benchmarks para E/S nos Supercomputadores

A entrada e saída dos dados nos supercomputadores são dominadas pelo acesso a arquivos grandes armazenados em disco. Além disso, as aplicações podem ser executadas em *batch* durante horas. Nessas situações, o comportamento da entrada e saída é marcado por uma leitura inicial de dados seguida de escritas. São realizadas muito mais escritas do que leituras. A medida de entrada e saída de um supercomputador é o *throughput* o número de *bytes* por segundo entre a memória principal e o disco.

Benchmarks para E/S de Sistemas para Processamento de Transações

Aplicações de processamento de transações precisam de um bom tempo de resposta, medida com base no *throughput*. A maioria dos acessos de e/s é para uma pequena quantidade de dados. Essas aplicações envolvem modificações em grandes bases de dados. Por este motivo, as aplicações para processamento de transações está relacionada a com a taxa de e/s, medida como o número de acessos ao disco por segundo, em oposição à taxa de dados medida em *bytes* por segundo.

Benchmarks para E/S de Sistemas de Sistemas de Arquivos

Os sistemas de arquivos são armazenados em disco e têm um padrão de acesso diferente.Os acessos podem ser a arquivos grandes ou pequenos, seqüenciais ou não, de leitura ou de escrita. Além disso, todos os acessos podem ser feitos com diferentes porcentagens. As medidas levaram a criação de *benchmarks* sintéticos para sistema de arquivos. Estes *benchmarks* tentam reproduzir a diversidade das cargas de trabalho existentes.