Unidade IV

7 BARRAMENTOS, TIPOS DE TRANSMISSÃO E DISPOSITIVOS DE ENTRADA/SAÍDA (E/S)

7.1 Barramentos do computador

Um barramento pode ser definido como um fio de cobre que tem como função realizar um "caminho" elétrico entre vários dispositivos internos ou externos ao computador, como pode ser observado na figura a seguir.

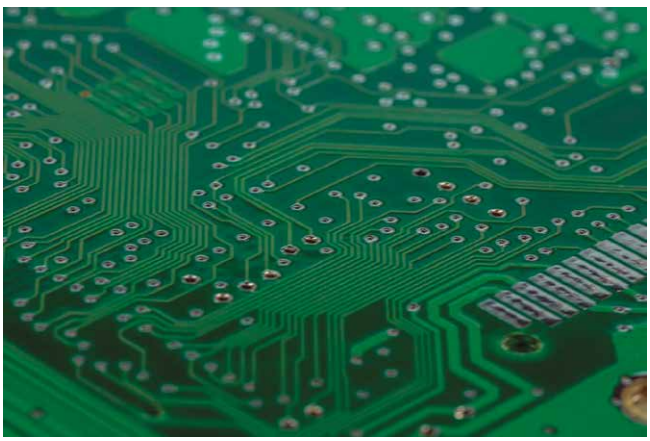

Figura 162 – Configuração de fios de cobre em um barramento da placa-mãe

Observação

Os barramentos geralmente são diferenciados de acordo com sua finalidade, como para transporte de dados entre a ULA e a UC, ou entre a CPU e a memória principal, ou mesmo entre outros dispositivos de E/S como disco rígido, CD, DVD, *pen drive* etc.

Os primeiros PCs possuíam um barramento externo, conhecido também como barramento do sistema, que consistia em 50 a 100 fios paralelos dentro da placa-mãe, constituídos por conectores com intervalos de distanciamento regulares ligados à memória principal e outras placas de E/S como conexões de rede, impressora etc. (TANENBAUM; AUSTIN, 2013).

Em diagramas encontrados em livros didáticos, o barramento sempre é representado por setas largas e sombreadas. A distinção entre as setas e uma linha reta apenas cortada por um pequeno segmento

de reta inclinada com valores de *bits* é quase imperceptível. Quando os *bits* de dados ou endereços, por exemplo, são utilizados na figura, geralmente denota-se com a representação de um segmento de reta diagonal. Entretanto, para *bits* de controle usa-se, geralmente, setas mais largas e sombreadas, como esquematizado na figura a seguir.

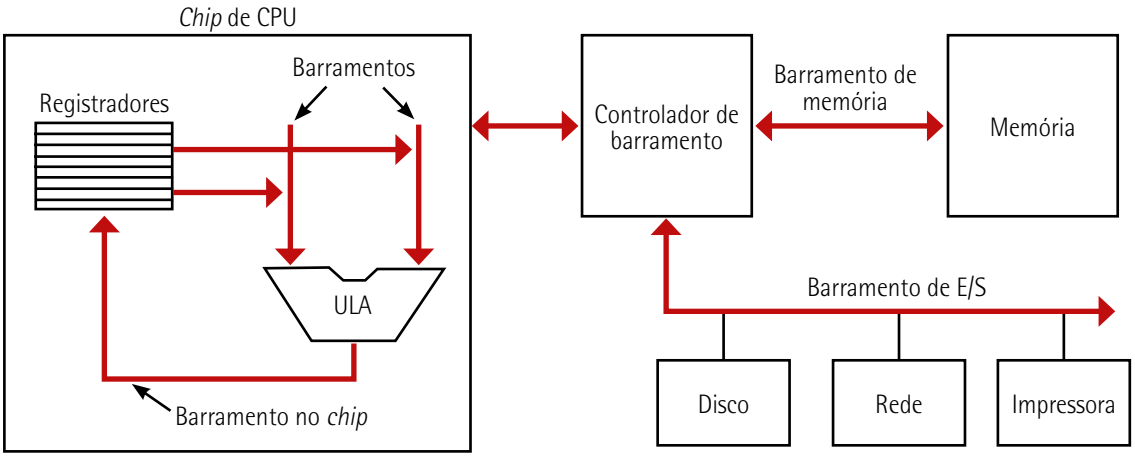

Figura 163 – Organização interna de um computador e seus barramentos

Para que os barramentos sigam um certo padrão, os projetistas de computador devem seguir um protocolo de barramentos, que irá determinar especificações mecânicas (tensões mecânicas) e elétricas (temporização, largura de banda) na fabricação das placas. Existem variedades de barramentos para uso em computadores, entre os mais conhecidos estão: Omnibus (PDP-8), Unibus (PDP-11), Multibus (8086), barramento IBM PC (PC/XT), barramento ISA (PC/AT), barramento SCSI, entre outros. Outra característica de funcionamento dos barramentos está relacionada à transferência ativa ou passiva de dados/instruções. Os barramentos ativos são conhecidos como mestres, enquanto os passivos são denominados escravos.

Quando o processador ordena que um controlador de disco realize a leitura ou escrita de um bloco de palavras, a CPU está agindo como um mestre e o dispositivo que atua obedecendo a operação como um escravo. Os sinais em binário, emitidos pelos dispositivos do computador, geralmente são fracos e não conseguem energizar o barramento, principalmente se ele for relativamente longo e com muitos dispositivos conectados nele. Assim, a maioria dos barramentos mestres estão conectados por um *chip* denominado controlador de barramento, que opera como um amplificador digital de sinais elétricos. De forma semelhante, os dispositivos que operam como escravos estão conectados em um *chip* denominado transceptor de barramento. As interfaces de barramento possuem três estados diferentes, que permitem que se desconectem quando for necessário ou então se conectem (coletor aberto) (TANENBAUM; AUSTIN, 2013).

7.2 Largura de barramento

Em se tratando de memórias de grande capacidade, os barramentos necessitam de muitas linhas de endereços. O problema é que barramentos muito largos necessitam obrigatoriamente de mais espaço físico, mesmo utilizando fios estreitos, além de necessitar de conectores maiores. Esses fatores

podem encarecer o barramento, obrigando os fabricantes a produzirem barramentos que levem em consideração o tamanho máximo de memória a ser utilizada a um dado barramento específico. Um sistema utilizando barramento de endereçamento de 64 linhas e 2^{32} bytes de memória principal irá custar mais do que um com 32 linhas e os mesmos 2^{32} bytes. O computador IBM PC com processador Intel 8088, esquematizado na figura a seguir (A) possuía um barramento de endereços de 20 bits que possibilitavam o endereçamento de 1 MB de memória.

Observação

A largura do barramento é outro parâmetro que deve ser considerado no projeto. Quanto mais linhas de endereços tiver no barramento, mais capacidade de memória *cache* ou memória principal e processador podem ser endereçados de forma direta. Portanto, se um barramento possuir n linhas de endereços, então a CPU poderá utilizá-la para endereçar 2n localizações diferentes na memória.

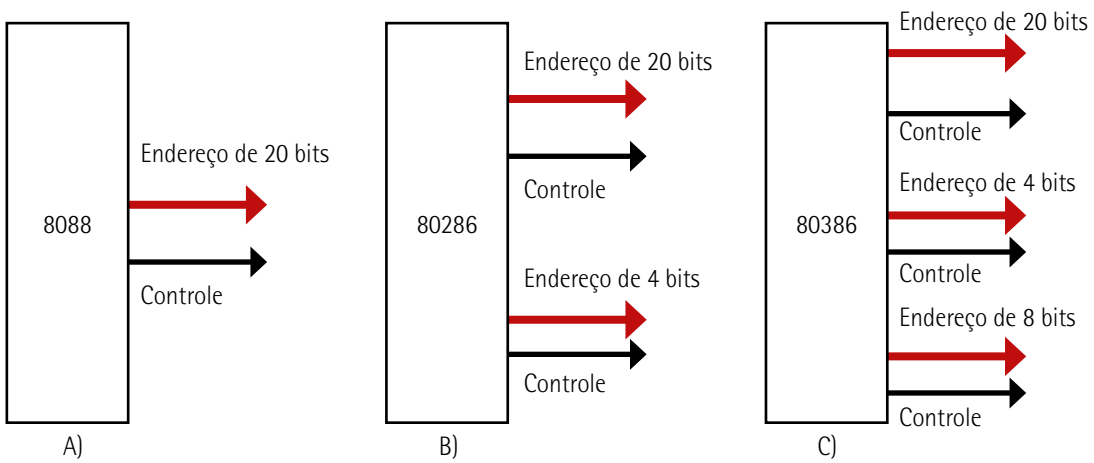

Figura 164 – A) barramento de endereço e controle no processador Intel 8088; B) no Intel 80286; e C) no Intel 80386

Com a evolução do 8088, a Intel lançou o 80286 e decidiu aumentar o espaço de endereçamento para 16 MB, de modo que foi necessário adicionar mais quatro linhas de barramento sem alterar as vinte linhas originais para facilitar a compatibilidade, como mostra a figura anterior (B). Já no lançamento do Intel 80386, oito linhas de endereço precisaram ser adicionadas, além de outras linhas de controle, originando o barramento conhecido como EISA, como mostra a figura anterior (C).

Lembrete

Em projetos atuais, não é apenas o número de linhas de endereços que tende a crescer, mas também o número de linhas para transmissão de dados.

Existem duas maneiras básicas para aumentar a largura de banda para transmissão de dados no barramento (TANENBAUM; AUSTIN, 2013):

- Redução do tempo para realizar a transmissão de dados, o que implicaria mais transferências por segundo.

- Aumento da largura de dados, possibilitando mais *bits* em cada transferência.

Assim, aumentar a aceleração do barramento é possível, porém difícil, pois os sinais geralmente trafegam com diferentes velocidades no barramento, problema conhecido na literatura como atraso diferencial do barramento. Esse atraso pode complicar a evolução dele, pois quanto mais rápido for o barramento, mais provável e sério será o atraso diferencial. Outro problema relacionado ao barramento está associado à compatibilidade da aceleração. Placas e circuitos impressos mais antigos foram projetados para barramentos mais lentos e podem não funcionar com um novo. Para contornar um problema desses, os projetistas de computadores optam por utilizarem um barramento multiplexado, em que, em vez de linhas de endereços e dados separados, podem ser anexadas 32 linhas de endereços e dados juntos. Então pode-se inicialmente utilizar o barramento somente para endereços e depois somente para dados. Os barramentos também podem ser divididos em duas categorias: síncronos ou assíncronos, de acordo com sua dependência do *clock*.

7.3 Barramentos síncronos

Os barramentos síncronos possuem uma linha controlada por um oscilador de cristal, responsável pelo sinal de *clock*, que consiste em uma onda quadrada gerada a partir de um sinal analógico, como o da figura a seguir, que possui frequências geralmente entre 5 MHz e 133 MHz.

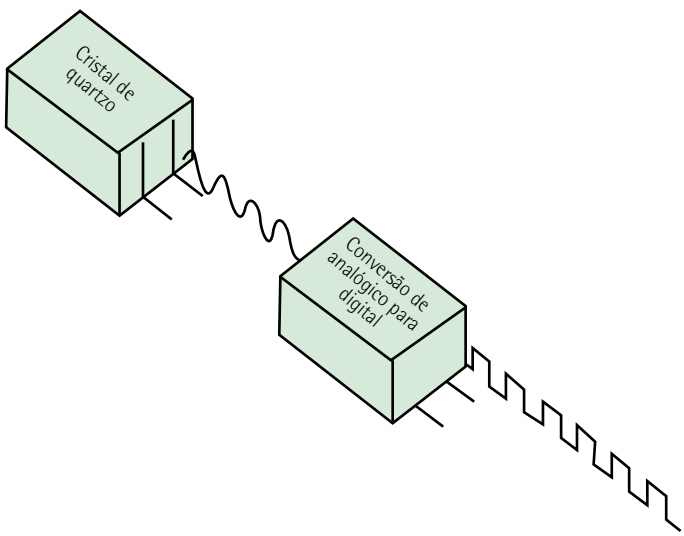

Figura 165 – Geração de um sinal de *clock* e sua conversão analógica/digital

Utilizando como base de exemplo um barramento síncrono, considere o diagrama temporal da figura a seguir. Nesse exemplo, utiliza-se um *clock* de 100 MHz, resultando em um ciclo de barramento de

10 nanossegundos, e tem-se as linhas de *clock*, ADDREES, DATA, $\overline{\text{MREQ}}$, $\overline{\text{RD}}$ e $\overline{\text{WAIT}}$ representadas na mesma escala de tempo.

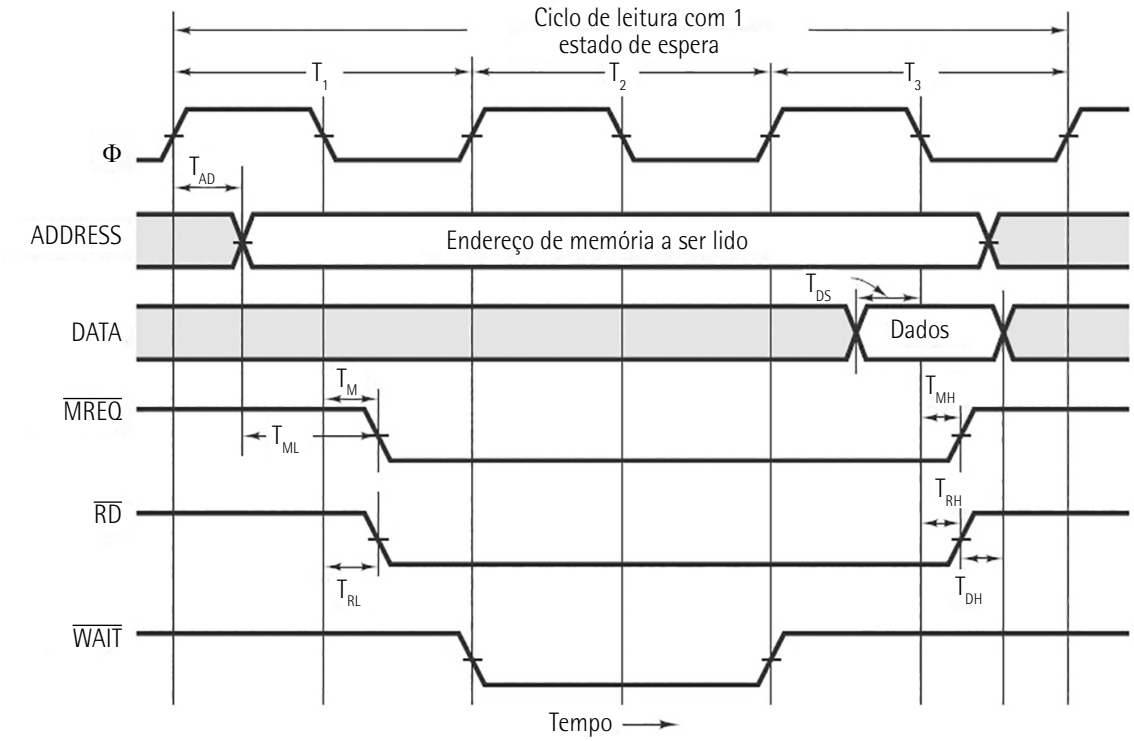

Figura 166 – Temporização de leitura em um barramento síncrono

Utilizando esse tipo de barramento, leva-se 15 nanossegundos para que um endereço seja lido na memória. No geral, essa leitura ocorre durante 3 ciclos de barramento, em que o primeiro ciclo começa na borda ascendente de T_1 e o terceiro ciclo termina na borda ascendente de T_4. No exemplo da figura anterior, nenhuma das bordas ascendentes ou descendentes foi colocada na linha vertical, pois nenhum sinal elétrico poderá trocar seu valor no tempo zero, de modo que se pode admitir que se leva 1 nanossegundo para o sinal mudar. O início de T_1 é definido através da borda ascendente do *clock*, e no meio do caminho de T_1 o processador insere o endereço da palavra desejada nas linhas de endereço. O endereço não é constituído de um único valor e é mostrado na figura como duas linhas que se cruzam no instante da mudança de endereço. Assim, é possível observar que o conteúdo das linhas de dados não é significativo até grande parte de T_3. Após as linhas de endereços acomodarem seus novos valores, $\overline{\text{MREQ}}$ e $\overline{\text{RD}}$ são ativados. No primeiro indica-se que está sendo ativada (*bit* 0) a memória e não uma comunicação de E/S, e o segundo é ativado (*bit* 0) para leitura e negado (*bit* 1) para realização da operação de escrita. Dessa forma, como a memória leva cerca de 15 nanossegundos após o endereço ficar estável, ela não poderá entregar os dados requisitados durante o tempo T_2.

7.4 Barramentos assíncronos

Os barramentos do tipo assíncrono não possuem um *clock* mestre e, embora sejam fáceis para trabalhar devido a seus intervalos discretos de tempo, eles possuem alguns problemas. Por exemplo,

como os ciclos de *clock* não podem ser fracionados, ainda que o processador e a memória realizem uma transferência de dados em 3,1 ciclos, eles deverão ser prolongados até 4,0. Outro problema também relacionado com os ciclos de leitura/escrita consiste em que barramentos construídos especificamente para esse tipo de comunicação dificilmente são aproveitados em avanços de tecnologia. Nos barramentos assíncronos, como o da figura a seguir, em vez de vincular todas as operações ao *clock*, quando o mestre ativar o endereço $\overline{\text{MREQ}}$ e $\overline{\text{RD}}$ na sequência, ele ativa um sinal especial denominado $\overline{\text{MSYN}}$ ou *master synchronization*. Dessa forma, quando o escravo verifica o sinal, ele irá realizar o trabalho de forma mais rápida, concluindo a fase ativando o $\overline{\text{SSYN}}$ ou *slave synchronization*. Em resumo:

- $\overline{\text{MSYN}}$ é ativado.
- $\overline{\text{SSYN}}$ é ativado em resposta a $\overline{\text{MSYN}}$.
- $\overline{\text{MSYN}}$ é negado em resposta a $\overline{\text{SSYN}}$.
- $\overline{\text{SSYN}}$ é negado em resposta à negação de $\overline{\text{MSYN}}$.

Operação de um barramento assíncrono:

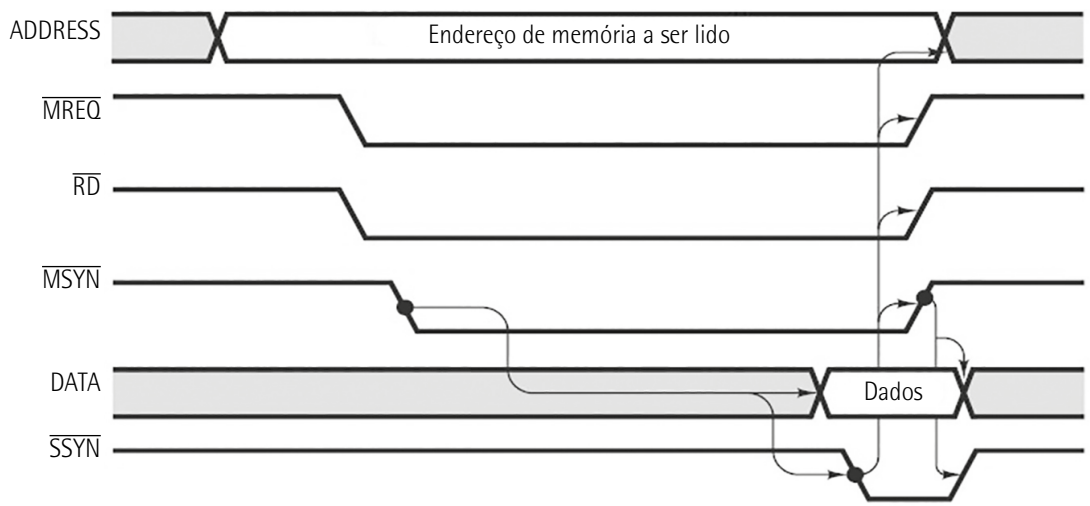

Figura 167 – Temporização de leitura em um barramento assíncrono

7.5 Barramento ISA

O barramento do tipo ISA (*Industry Standard Architecture*), desenvolvido em meados dos anos 1980 para funcionar em computadores pessoais da IBM PC/AT, equipados com processadores Intel 80286, era uma estrutura típica de 8 bits que foi projetada para substituir o padrão já ultrapassado VESA (*Video Electronics Standards Association*). Orginalmente o barramento padrão ISA operava em frequência de 4,77 MHz, mas que logo foi substituído por uma frequência mais alta de 8,33 com o lançamento dos modelos de computadores PC/XT de 16 bits. O padrão ISA era dividido em duas partes, sendo a primeira contendo pinos utilizados por placas que suportam apenas 8 bits, e a segunda com uma possibilidade de

extensão que adiciona pinos extras. Outra característica do barramento ISA é a grande quantidade de contatos elétricos, que chegam a 98, como mostra a figura a seguir, de modo que somente 16 trilhas são para dados e o restante utilizadas para realizar o endereçamento. Se comparada com os barramentos atuais, o ISA é considerado muito lento para operações que envolvem processamento de alto desempenho, como os da placa-mãe do computador, porém esse tipo de barramento ainda é utilizado principalmente pela facilidade de operação, e muitos desenvolvedores na área de automação industrial ou robótica ainda utilizam esse padrão.

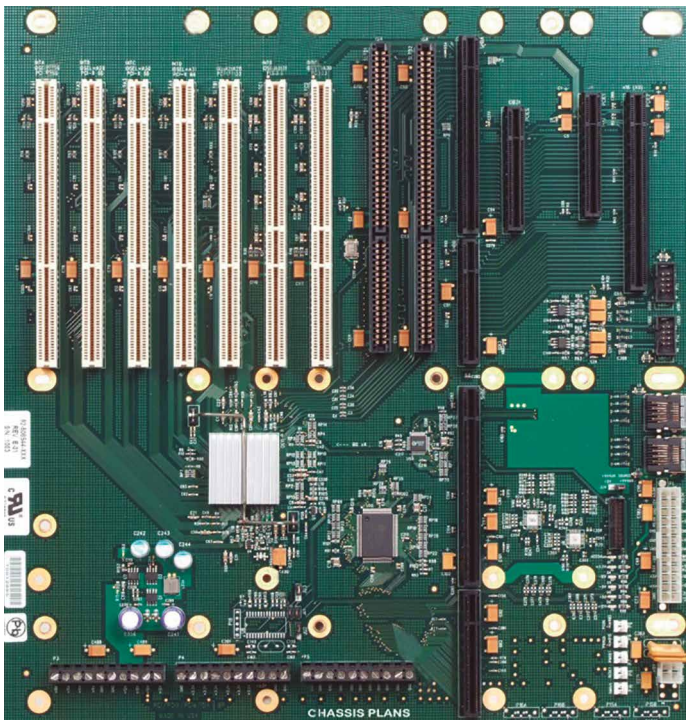

Figura 168 – Barramento padrão ISA

7.6 Barramento PCI

O barramento PCI (*peripheral interconnect bus* ou barramento de interconexão de componente periférico), visualizado na figura a seguir, foi desenvolvido em 1990 em substituição ao já ultrapassado barramento EISA (*Extended Industry Standard Architecture*). Originalmente o PCI possuía a capacidade de transferência de 32 bits por ciclo, barramento compatível a um *clock* de 33 MHz e largura de banda de 133 MB/s. Em 1993 foi lançada a segunda versão do padrão PCI (PCI 2.0), e em 1995 o PCI 2.1. O barramento do tipo PCI possui capacidade de funcionamento em uma frequência até 66 MHz, manipulando transferências em 64 bits a uma largura de banda total, atingindo 528 MB/s. Devido a essa taxa elevada de transferência de dados/instruções, foi possível melhorar o desempenho nas transmissões de áudio e principalmente de vídeo em alta qualidade.

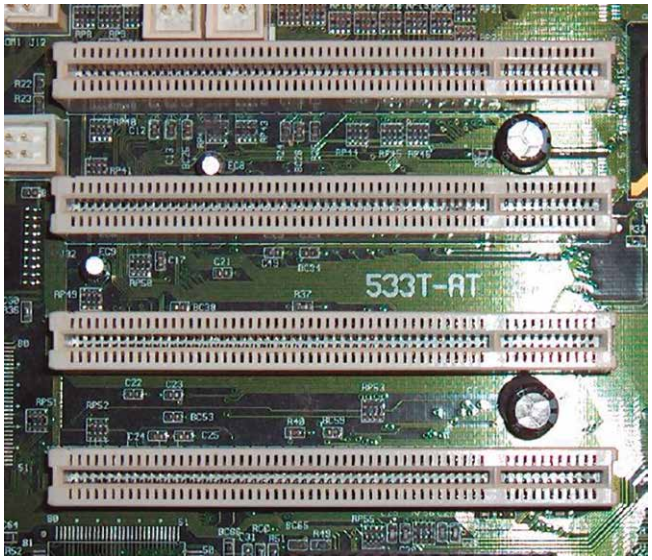

Figura 169 – Barramento PCI

A melhora do padrão PCI em comparação a padrões anteriores também se deve a dois *chips* da Intel, que trabalham como uma espécie de ponte que conecta o barramento PCI ao processador, à memória principal e ao restante do barramento. O barramento PCI foi desenvolvido para operar em 32 bits ou 64 bits, de modo que o padrão 64 bits pode comportar o funcionamento em 32 bits, mas o contrário não é possível. Além disso, o padrão PCI também suporta operar em 5 volts e 3,3 volts, como os barramentos mais modernos na atualidade.

7.7 Barramento AGP

No final dos anos 1990, o barramento padrão ISA estava praticamente inutilizado, portanto novos projetos, além do aprimoramento do padrão PCI, estavam em desenvolvimento. Um novo tipo de barramento utilizado para comandar a placa gráfica foi fabricado pela Intel, denominado AGP (*accelerated graphics port bus* ou barramento de porta gráfica acelerada), como mostra a figura a seguir. A primeira versão do AGP foi a 1.0 e funcionava a uma largura de banda de 264 MB/s e ficou conhecida como 1x. Embora um pouco mais lento do que o já existente PCI, ele era capaz de controlar de forma eficiente a placa gráfica.

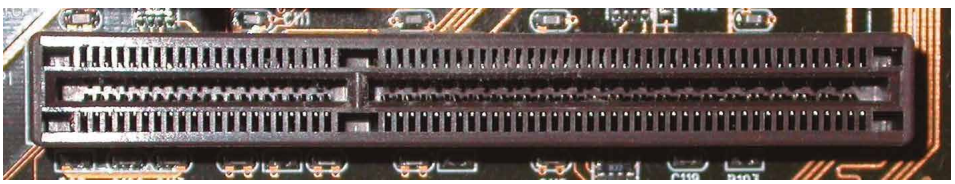

Figura 170 – Barramentos padrão ISA, PCI e AGP

7.8 Barramento PCI Express

Mesmo após sua evolução para o AGP 3.0 de alto desempenho, o barramento AGP foi superado em desempenho por um novo tipo, denominado PCI Express (figura a seguir), que possui a incrível capacidade de largura de banda de 16 GB/s de dados por enlaces de barramento serial de alta velocidade.

Uma das soluções da tecnologia PCI Express, ou simplesmente PCIe, é eliminar o barramento paralelo constituído de muitos mestres e escravos e utilizar um projeto baseado em conexões seriais ponto a ponto de alto desempenho. Essa solução apresenta uma transição radical na tradição do barramento ISA/EISA/PCI e se baseia em algumas ideias do universo de rede Ethernet comutada. A ideia se baseia em uma estrutura com um computador constituído de um conjunto de *chips*, memória e controladores de E/S que necessitam ser interconectados. O PCIe irá fornecer um sistema comutador de uso geral para conectar os *chips* utilizando ligações seriais, como ilustra o esquema da figura a seguir.

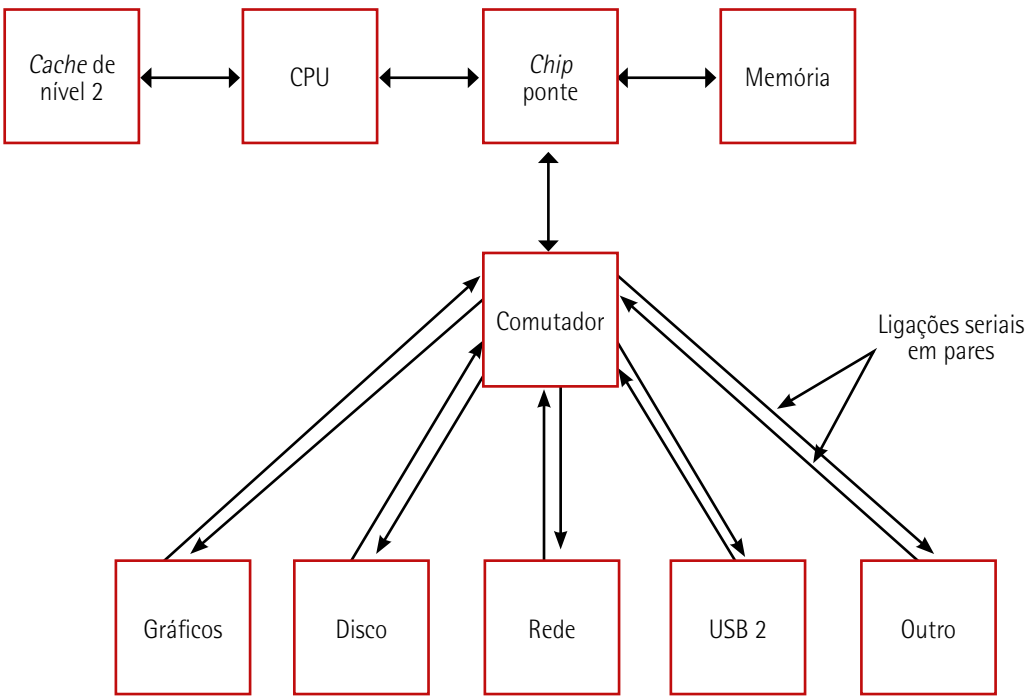

Figura 171 – Sistema de barramento PCIe

Como mostra a figura, o processador, a memória e a *cache* estão conectados a um *chip* através de ligações seriais em pares. Cada conexão é constituída por um par de canais unidirecionais onde um liga o comutador e/ou outro é oriundo dele. Cada canal é composto por dois fios de barramento, um para o sinal e outro para o aterramento, para diminuir alguma possibilidade de ruído durante a transmissão de alta velocidade. A arquitetura PCIe também possui três pontos de diferença em relação ao barramento PCI antigo, um deles é o comutador centralizado e o outro é a utilização de conexões seriais ponto a ponto estreitas. A terceira diferença é um pouco mais sutil e trata-se de um modelo conceitual de mestre de barramento encontrado no PCI, que emite um comando a um escravo que envia um pacote de dados a um outro dispositivo. Dessa forma, é comum se referir ao barramento PCIe como uma miniatura de rede de comunicação de comutação de pacotes.

7.9 Barramento serial universal USB

Embora os barramentos PCI e PCIe sejam muito eficientes e rápidos para anexar dispositivos de alto desempenho, eles são muito caros para ser empregados em dispositivos de E/S, que operam a baixa velocidade. Além da questão do custo, alguns barramentos como ISA têm a necessidade de que sejam adaptados alguns comutadores e pontes na placa, para assegurar que o barramento terá êxito de execução em versões mais recentes e para que nenhum ajuste adicional seja necessário.

Observação

Para resolver pendências relacionadas ao desempenho, em 1993 sete empresas de tecnologia (Compaq, DEC, IBM, Intel, Microsoft, NEC e Northern Telecom) se juntaram para buscar uma solução de unificação para a anexação de uma gama variada de dispositivos de E/S.

O padrão resultante dessa pesquisa em conjunto foi lançado em 1998 e ficou conhecido como USB (*universal serial bus* – barramento serial universal), e desde então é amplamente utilizado em computadores, celulares, *tablets*, *smart* TVs e sistemas embarcados em geral (TANENBAUM; AUSTIN, 2013). Ao se juntarem, as sete empresas tomaram como primordiais na elaboração do projeto as seguintes características:

- Os usuários não necessitarão de ajustes de comutadores ou pontes para que o USB funcione para diferentes tipos de dispositivos.
- Os usuários não necessitarão abrir a torre para instalar novos dispositivos de E/S.
- Apenas um tipo de cabo e conector será utilizado como padrão para esse barramento (figura a seguir).
- A energia de alimentação deverá ser fornecida pelo próprio barramento.
- O limite para 127 dispositivos é poderem ser conectados a um único computador utilizando o padrão USB.
- O sistema deverá ser capaz de suportar dispositivos que operem em tempo real, como, por exemplo, telefone, televisão etc.
- Novos dispositivos poderão ser instalados mesmo com o computador em funcionamento.
- Não será necessária a reinicialização do computador após a instalação do novo dispositivo.
- A produção desse novo barramento e todos os dispositivos de E/S baseados nele não deverá ter um custo alto.

Figura 172 – Conector do tipo USB (barramento serial universal)

O padrão USB atende a todos esses requisitos e é largamente utilizado em dispositivos como teclado, *mouse*, câmeras fotográficas, *scanners*, telefones digitais, entre outros. A primeira versão do padrão USB ficou conhecida como 1.0 e possui uma largura de banda de 1,5 Mbps, suficiente para atender teclados e *mouses*. Já a versão 1.1 opera em 12 Mbps, capaz de operar impressoras, câmeras digitais e outros dispositivos de multimídia. Já a versão 2.0 possui suporte para dispositivos com a capacidade de transferência de 480 Mbps, capaz de operar *drives* de disco rígido externo, *webcams* de alta definição e interfaces de redes ethernet. A versão 3.0 alcança incríveis velocidades de transferência de 5 Gbps e pode ser utilizada em dispositivos de alta *performance* e ultrabanda larga, como memórias externas SSD.

7.10 Tipos de transmissão

Existem dois modos básicos de realizar uma transmissão/recepção entre os periféricos/interfaces, o barramento e o processador/memória principal denominados: serial e paralela (MONTEIRO, 2019).

7.10.1 Transmissão serial

Na transmissão em série ou transmissão serial, o periférico é conectado ao módulo de controle ou interface de E/S através de uma linha única para realização da transmissão de dados, de maneira que essa transferência seja realizada 1 bit de cada vez, mesmo que o controlador tenha capacidade de ser conectado ao processador/memória principal através do barramento constituído de diversas linhas de transmissão e com grande largura de banda, como observado na figura:

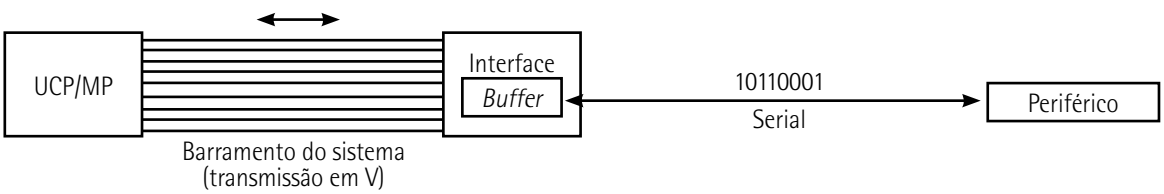

Figura 173 – Diagrama de transmissão serial entre uma interface e o periférico

A transmissão em seu início era considerada muito lenta se comparada com a transmissão paralela. Teclado, *mouse* e *modens* são dispositivos que realizam comunicação serial. Devido à transmissão serial ser realizada *bit* a *bit*, sempre será necessário que o receptor (RX) e o transmissor (TX) estejam sincronizados *bit* a *bit*, ou seja, o transmissor deverá transmitir os *bits* sempre na mesma velocidade, de modo que todos eles deverão possuir o mesmo tempo de duração. Por exemplo, se TX estiver funcionando em uma velocidade de 1 Kbps, isso significará que cada *bit* irá durar 1/1.000 segundos ou 1 milissegundo. Para que o RX seja capaz de receber todos os *bits* de forma correta, ou seja, um a um, ele precisa iniciar a sequência de *bits* e denominar qual a sua duração, conhecida como largura de *bit*. Dessa forma, se a cada 1 milissegundo TX enviar 1 bit, cujo nível de tensão alta é considerado *bit* 1 e nível de tensão baixa considerado *bit* 0, então RX deverá verificar o nível de tensão da linha recebida para identificar o *bit* como 0 ou 1.

As comunicações em série ainda podem ser subclassificadas como síncronas e assíncronas. Nas transmissões assíncronas, o processo de sincronização de RX é realizado e cada novo caractere transmitido, de modo que, antes de a transmissão ser iniciada, cada caractere será acrescido de dois pulsos, um no início e outro no final do pulso. A transmissão síncrona é considerada uma forma moderna de comunicação, se comparada com a assíncrona, pois nesse caso os dados são transmitidos de uma só vez por blocos de caracteres sem a necessidade de intervalos entre eles e sem a necessidade de um pulso indicando o início ou o fim da transmissão, reduzindo com o uso a quantidade de *bits* na linha de transmissão. Na atualidade, uma das aplicações de maior uso de cabeamento em série é sua utilização para conexão de discos rígidos ou *drives* de SSD, utilizando o cabeamento SATA (*serial advanced technology attachment*), como mostra a figura a seguir.

Figura 174 – Conexão de cabo SATA (serial ATA)

7.10.2 Transmissão paralela

A transmissão em paralelo possui a capacidade de enviar um grupo de *bits* de cada vez, em que cada *bit* é enviado por uma linha separada de transmissão, quanto maior a distância entre os dispositivos, maior será o custo, devido à quantidade de linhas utilizadas, como mostra a figura:

Figura 175 – Diagrama de transmissão paralela entre TX e RX

Lembrete

Na transmissão paralela, quanto maior a distância entre os dispositivos, maior será o custo, devido à quantidade de linhas utilizadas.

Outra característica das transmissões em paralelo é que elas costumam ser mais rápidas se comparadas com algumas conexões em série mais antigas, o que permite maiores taxas de transmissão para dados. Um problema aparente com a transmissão em paralelo é que os *bits* enviados podem não chegar ao destino exatamente no mesmo instante ao qual foram programadas para chegar. Isso ocorre devido a algumas diferenças de comprimento de cabos de que os canais de comunicação são constituídos. A solução para esse problema foi o aprimoramento da transmissão em série, pois ao transmitir *bit* a *bit*, não ocorrerá o problema de desvio conhecido como *skew*. Em consequência disso, como já abordado, em 1995 surgiu outro padrão, conhecido como USB. Entre os dispositivos que utilizavam em larga escala as conexões em paralelo estão as impressoras e as conexões entre o disco rígido e a placa-mãe, conhecidos como IDE (*integrated drive electronics*), como mostra a figura:

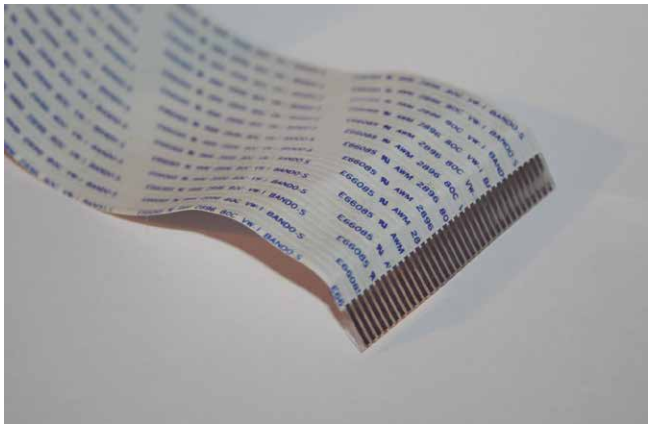

Figura 176 – Conexão paralela do tipo IDE

7.11 Módulos de E/S

Os módulos de comunicação de dispositivos de entrada e saída possuem como principais requisitos ou funções:

- Controle e temporização.
- Comunicação com a CPU.
- Comunicação com os dispositivos de E/S.
- Armazenamento temporário dos dados (*buffer*).
- Detecção de erros.

Durante a operação, o processador deve se comunicar com um ou mais dispositivos externos em diferentes tipos de padrões. Os recursos internos disponíveis à CPU, como a memória principal e o barramento do sistema, possuem a necessidade de compartilhar várias de suas atividades, o que também inclui dados oriundos de dispositivos de E/S. Dessa forma, existe a necessidade de haver um controle e a temporização, para que o fluxo de tráfego de dados seja coordenado entre os dispositivos. Por exemplo, o controle da transferência dos dados armazenados em um dispositivo externo para serem processados na CPU pode envolver as seguintes etapas (STALLINGS, 2010):

- A CPU pergunta ao módulo de E/S se o estado corresponde a "conectado".
- O módulo de E/S deverá então retornar ao estado atual do dispositivo.
- Se o dispositivo estiver no estado operacional e pronto para transmissão de dados, então a CPU solicita o início da transferência de dados através de um comando enviado ao módulo de E/S.

- O módulo de E/S obtém uma unidade de dados de 8 ou 16 bits do dispositivo externo.
- Os dados são transferidos do módulo de E/S para a CPU.

Além desses fatores, a comunicação entre os dispositivos externos com a CPU também irá envolver:

- **Decodificação de comando**: os módulos de E/S deverão aceitar os comandos da CPU que são enviados como sinais no barramento de controle.
- **Dados**: deverão ser transferidos entre a CPU e o módulo de E/S através do barramento de dados.
- **Informação de estado**: devido à baixa velocidade dos periféricos de E/S, é de extrema importância conhecer o estado atual do seu módulo. Por exemplo, se o módulo de E/S precisar enviar dados para a CPU para realizar a operação de leitura, ele pode não ser capaz de realizar a operação, pois ainda estará trabalhando atendendo ao comando de E/S anterior.
- **Reconhecimento de endereço**: da mesma forma que cada palavra contida na memória possui um certo endereço, cada dispositivo de E/S também o terá. Dessa maneira, um módulo de E/S deverá reconhecer um endereçamento exclusivo para cada dispositivo externo que ele controla.
- **Comunicação com o dispositivo**: o módulo de E/S deverá ser capaz de realizar a comunicação entre dispositivos, o que envolverá diversos sinais de controle, determinando a função que o dispositivo realizará, conforme mostra a figura:

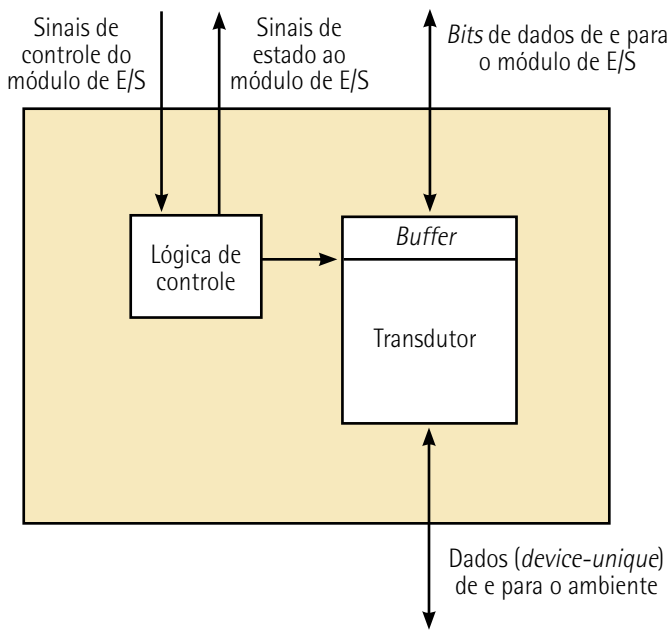

Figura 177 – Diagrama de blocos representando um dispositivo externo e seus sinais de comunicação

7.12 Interfaces e dispositivos de E/S

Os dispositivos de E/S possuem diferentes taxas de transmissão entre eles, como mostra o quadro. Basicamente, um dispositivo de E/S se comunica com o meio externo e com sua interface através do envio de *bits* e sinais de controle.

Quadro 9 – Dispositivos de E/S e velocidade de transmissão de dados

Dispositivo	Taxa de transmissão em KB/s
Teclado	0,01
Mouse	0,02
Impressora matricial	1
Modem	2 a 8
Disquete	100
Impressora a *laser*	200
Scanner	400
CD-ROM	1.000
Rede local	500 a 6.000
Vídeo gráfico	6.000
Disco rígido	2.000 a 10.000

Adaptado de: Monteiro (2019).

No processo de comunicação está envolvido o envio/recebimento de dados e sinais de controle. Embora cada dispositivo possua características de funcionamento próprios, o fluxo de informações é o mesmo para todos. A figura a seguir mostra alguns dos principais componentes de uma interface de E/S, assim como os tipos de informação e o sentido direcional para o fluxo de dados transmitidos/recebidos (RX/TX) das conexões entre os periféricos.

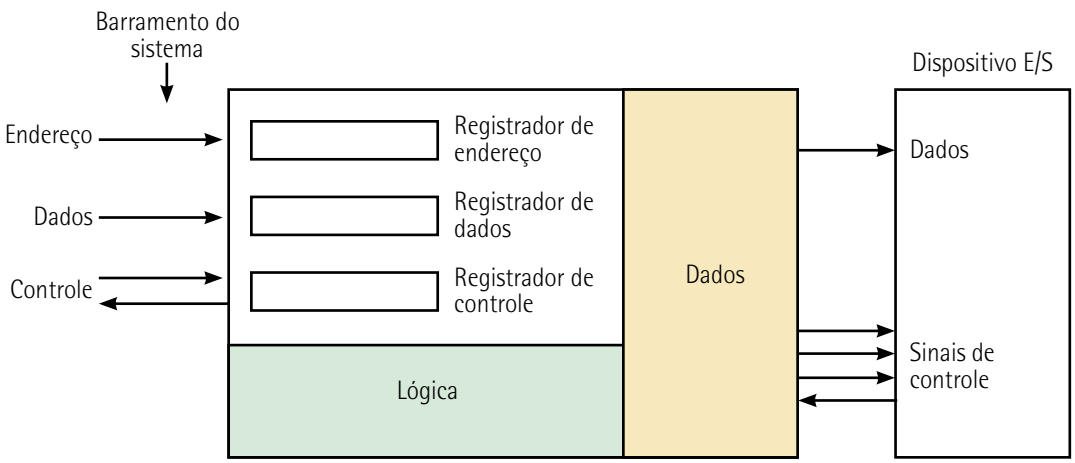

Figura 178 – Módulo de interface e conexão de E/S

Na figura, é possível também observar três diferentes áreas, que são constituídas primeiramente por registradores, que têm função de interação básica entre a interface de E/S e sua conexão com o barramento do sistema, além de controlar os registradores de dados e endereços. Na sequência, tem-se o espaço de armazenamento de dados que circulam entre os dispositivos pelo barramento durante as operações de E/S, fazendo com que a interface de E/S atue como um *buffer* para operar com diferentes velocidades em compatibilidade com os diferentes dispositivos. A terceira área é onde se localiza a lógica de funcionamento da interface, que permite a interação entre os dispositivos externos e a detecção de erros, entre outros.

7.12.1 Teclado

Um dos principais dispositivos periféricos do computador é o teclado, pois facilita que o computador possa receber comandos do ser humano. O reconhecimento de padrões realizado pelo teclado se baseia na interpretação do sinal elétrico de cada tecla ao ser pressionada pelo usuário. A figura a seguir mostra um exemplo típico de um teclado de computador.

Figura 179 – Teclado de computador.

De uma forma geral, existem basicamente três categorias de teclado em uso nos dispositivos modernos (MONTEIRO, 2019):

- **Teclados apenas numéricos**: utilizados geralmente em calculadoras de bolso ou mesa.
- **Teclados utilizados em sistemas dedicados**: consistem em teclas utilizadas somente para entrada de dados referentes à tarefa ao qual o dispositivo foi projetado (sistemas embarcados) como, por exemplo, equipamentos de ar-condicionado, micro-ondas, televisão, foguetes etc.
- **Teclado de uso geral**: constituído por teclas alfabéticas, numéricas, sinais de pontuação, operações aritméticas, entre outras de uso especial.

O funcionamento de um teclado pode ser entendido como uma chave que, ao pressioná-la, aciona-se a chave e, como consequência, o sinal elétrico naquele local é inserido pelos circuitos de controle. Abaixo das teclas do teclado, há um circuito impresso com diferentes dispositivos eletrônicos como capacitores. A tecla capacitiva funciona baseada na variação de capacitância do acoplamento entre duas placas metálicas. Essa variação sempre ocorre quando uma tecla for pressionada. Dispositivos capacitivos como o teclado são, geralmente, de baixo custo e com tamanhos relativamente pequenos, além de possuírem tempo longo de vida, com cerca de 20 milhões de pressionamentos. Um teclado de computador basicamente pode ser descrito, em termos de funcionamento, com as seguintes características:

- **Detecção do pressionamento de teclas**: um processador de 8 bits, geralmente Intel 8048 ou 8049 interno ao teclado, realiza a varredura para a detecção das teclas pressionadas.

- ***Debouncing* de pressionamento**: confirma se a tecla foi realmente pressionada, repetindo a varredura várias vezes.

- **Geração do código de identificação da tecla pressionada**: geração de uma codificação de 8 bits baseada em linhas e colunas, denominada código de varredura ou *scan code*.

- **Geração de um sinal de interrupção**: gerado a partir do pressionamento de uma tecla, fazendo com que a CPU tome providências em relação à identificação da tecla scaneada, e seu valor decodificado pelo programa.

- **Troca de sinais de interrupção entre teclado e CPU**: nessa operação, o código de varredura é transmitido para um endereço da memória principal, interpretado por um programa de E/S.

Observação

A BIOS (*basic input output system*) analisa o código recebido para verificar se uma tecla foi pressionada sozinha ou em conjunto, como, por exemplo, ALT, e aciona o código ASCII correspondente às teclas pressionadas na área de memória apropriada.

7.12.2 *Mouse*

O *mouse* (figura a seguir) é um dispositivo de entrada cujo propósito principal é o de facilitar o trabalho do usuário final em sua tarefa de comunicação com o computador. Seu nome deriva do seu formato pequeno e sua ligação com o computador por um fio (em *mouses* mais antigos) que se assemelha a uma cauda de rato. Diferentemente do teclado, em vez de o usuário ter que digitar comandos, ao utilizar o *mouse* somente é necessário o uso da coordenação motora para a movimentação. Ele trabalha basicamente como uma interligação visual do usuário com os sistemas de *hardware* e *software* do computador. O *mouse* possui alguns tipos de sensores (de acordo com o modelo), como mecânicos, ópticos e ópticos-mecânicos, para realizar a captação do movimento em uma superfície plana e transmitir as informações coletadas desses movimentos ou do acionamento dos seus botões.

Alguns programas fazem parte do funcionamento do *mouse*, como elementos de interface apontadores na tela, onde o usuário escolhe o que quer acionar e seleciona o item pressionando um dos dois botões.

Figura 180 – *Mouse* óptico sem fio

O *mouse* totalmente mecânico possui uma esfera coberta com borracha que gira, acompanhando o movimento de sua rotação. Esse movimento da esfera é então transmitido a dois roletes perpendiculares que possuem rodas de contato com o metal, e à medida que as rodas giram, os contatos tocam algumas escovas em seu interior. Já o *mouse* do tipo óptico-mecânico possui o mesmo mecanismo de esfera e rolete em seu interior, com exceção dos roletes conectados a furos em sua carcaça, onde, à medida que as rodas giram, elas podem bloquear ou permitir a passagem da luz produzida por um LED. Isso fará com que as transições entre essa passagem da luz sejam detectadas por sensores semicondutores sensíveis à luz.

7.12.3 Impressoras matriciais, jato de tinta e a *laser*

As impressoras são dispositivos básicos de saída, em que algumas das informações internas do computador são convertidas em símbolos impressos em um meio externo, como o papel. As impressoras possuem algumas caraterísticas básicas como (MONTEIRO, 2019):

- **Volume de impressão que a impressora suporta por unidade de tempo**: geralmente as impressoras possuem "vazão" de impressão e caracteres por segundo (cps), dados em linha por minuto (lpm) ou em páginas por minuto (ppm), de acordo com o tipo e modelo utilizado.

- **Tecnologia para impressão de símbolos**: divididos em matricial, jato de tinta, *laser* e térmica (uso de cera aquecida ou sublimação de tinta).

As impressoras de impacto, como as matriciais (figura a seguir), já foram muito populares no mercado.

Figura 181 – Impressora do tipo matricial

Seu mecanismo de impressão se baseia em um dispositivo com um conjunto de martelos ou agulhas, utilizados para pressionar uma fita com tinta, imprimindo o símbolo no papel posicionado atrás da fita.

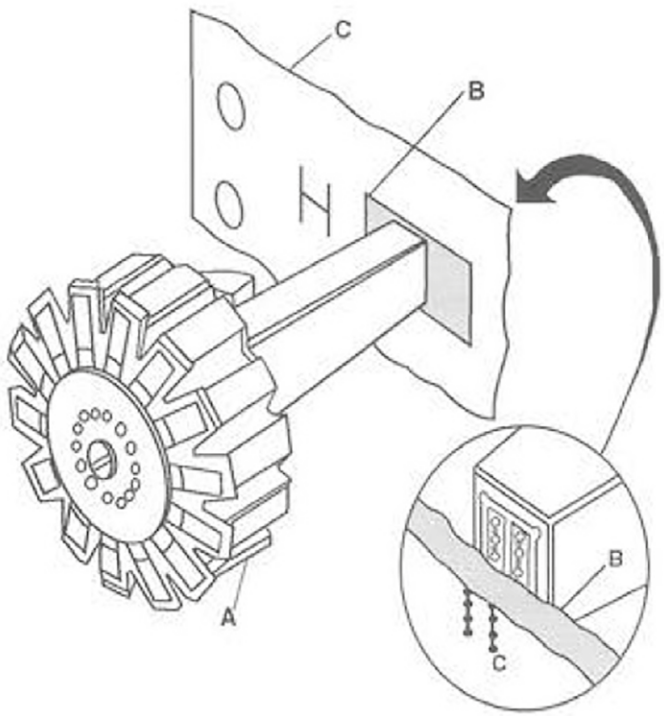

Figura 182 – Mecanismo de impressão em uma impressora matricial

O item A da figura mostra o mecanismo de impressão constituído pela cabeça de impressão, o item B se refere à fita contendo tinta, e o item C, o papel onde os símbolos são impressos. A geração de pontos no papel tem seu início com a cabeça de impressão, constituída por um tubo contendo agulhas e ligado a uma bobina, e uma corrente elétrica energiza a bobina que impactará nas agulhas, pressionando a fita, impregnando o papel. Dessa maneira, a cabeça imprimirá simultaneamente os n pontos de uma coluna e os n pontos da coluna seguinte, sucessivamente, até formar todo o caractere desejado e completar a linha.

Já as impressoras do tipo jato de tinta ou *ink-jet* (figura a seguir) são mais populares e utilizadas atualmente, principalmente pelo seu baixo custo. Seu funcionamento envolve a produção de caracteres em forma de uma matriz de pontos em uma folha de papel. Esse processo se assemelha com as impressoras matriciais e a diferença está na técnica utilizada para a criação dos pontos no papel.

Figura 183 – Impressora do tipo jato de tinta

Para impressoras a jato de tinta, a criação do ponto é o resultado de uma gota de tinta depositada no papel na região determinada por coordenadas x e y. O mecanismo que realiza a impressão é constituído por uma certa quantidade de tubos pequenos contendo um bico injetor para permitir a saída das gotículas de tinta. Tecnologias atuais são baseadas na projeção de gota a gota por demanda ou *drop-on-demand bubble jet*, com modelos que possuem de 128 a 256 bicos injetores. Esse processo consiste na passagem de uma corrente elétrica através de uma resistência que aquecerá a tinta, facilitando sua saída pelo bico do tubo. Esse processo ocorrerá milhares de vezes por segundo durante toda a etapa de impressão. Em relação às cores de impressão, algumas impressoras só trabalham com tinta preta e são conhecidas como monocromáticas, enquanto outras impressoras jato de tinta imprimem colorido por meio do emprego de três ou quatro tubos de tinta (magenta, ciano, amarela e preta) formando as cores necessárias para a impressão.

Como uma impressora a jato de tinta possui diversos bicos injetores, ela produzirá uma matriz de pontos muito mais densa em comparação com as impressoras matriciais, que possuem geralmente 24 agulhas. Os valores típicos de impressão de tinta estão na faixa de 300 x 300 pontos por polegada ou ppi (*point per inch*).

As impressoras a *laser*, como a da figura a seguir, possuem como características uma alta qualidade da imagem, excelente flexibilidade, além de velocidade de impressão maior se comparada aos outros modelos.

Figura 184 – Impressora do tipo *laser*

A tecnologia empregada nesse tipo de impressora consiste em um tambor fotossensível rotativo de precisão onde, no início de cada ciclo, receberá uma carga de cerca de 1.000 volts. Na sequência, a luz de um *laser* passa pelo comprimento do tambor a fim de refleti-lo como um espelho octogonal e rotativo, como mostra a figura a seguir. O feixe de luz será modulado para produzir um padrão de pontos claros e escuros no papel. Após uma linha de pontos ser pintada, o tambor gira uma fração de um grau, a fim de permitir que a próxima linha também seja pintada. Na sequência, de acordo com a rotação, a primeira linha de pontos chega ao reservatório (tôner) que contém um pó negro eletrostaticamente sensível à corrente elétrica. Então o tôner é atraído pelos pontos carregados, formando uma imagem produzida naquela linha de forma que o tambor revestido pelo pó seja pressionado contra o papel, transferindo-o para ele. O papel passa pelos rolamentos que estão aquecidos, fundindo-se com o tôner, fixando a imagem no papel.

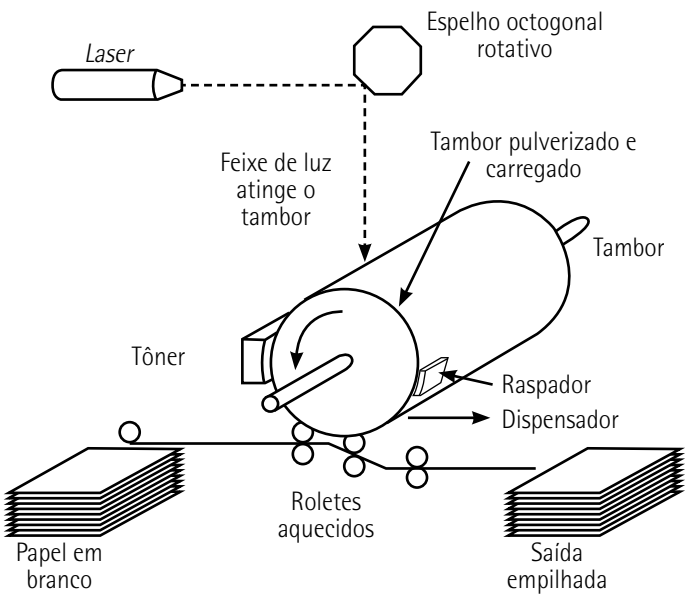

Figura 185 – Mecanismo de impressão em uma impressora a *laser*

Após a impressão no papel, o tambor será descarregado e raspado para eliminar qualquer resíduo de pó a fim de prepará-lo para o recebimento de uma nova carga elétrica e um novo revestimento para a próxima impressão de uma página.

7.12.4 Monitores de vídeo

Entre todos os periféricos de um computador, os monitores se tornaram os mais populares, além é claro de muito necessários. É difícil de acreditar, mas os primeiros computadores não possuíam monitores de vídeo, mas lâmpadas que acendiam e apagavam na frente do painel da máquina, em um padrão que representava valores em binário. Os monitores de vídeo (figura a seguir), também conhecidos como *displays*, pois a tradução se refere ao elemento de visualização, possuem diversas tecnologias e características físicas para exibição de uma imagem ou vídeo, como (MONTEIRO, 2019):

- tubo de raios catódicos ou CRT (*cathode-ray tube*);
- diodos emissores de luz ou TV de LED (*light emitting diodes*);
- monitores de cristal líquido ou LCD (*liquid-crystal display*);
- monitores com painel estreito ou VPE (*flat panel display*);

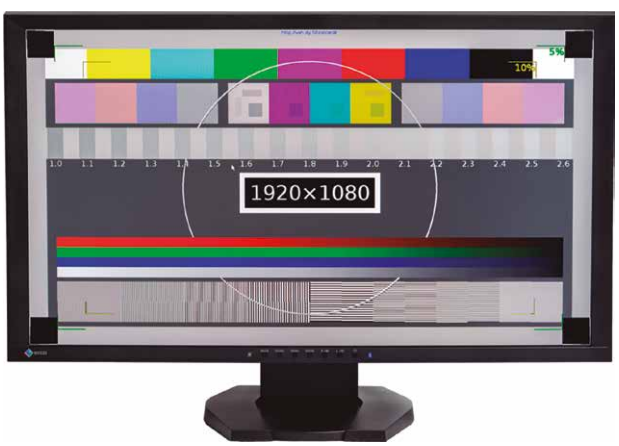

Figura 186 – Monitor de vídeo LED

Essas são algumas tecnologias utilizadas em monitores de vídeo, porém existem ainda muitas outras, como as de gás plasma e as eletroluminescentes. Apesar dessas novas tecnologias, os monitores do tipo CRT prevaleceram durante muitos anos como padrão em empresas ou mesmo nas residências. A figura a seguir mostra de forma esquemática o funcionamento de um monitor de tubo baseado em raios catódicos, que possui os seguintes elementos:

- Um cátodo, também conhecido como canhão de elétrons.
- Um ânodo formado pela tela frontal do vídeo, coberta com o elemento químico fósforo.

- Um par de bobinas defletoras de feixe na horizontal e na vertical para direcionar o feixe de elétrons.

A partir desses itens, é possível então explicar o funcionamento de um monitor de tubo na seguinte sequência:

- O canhão de elétrons emite um feixe concentrado que caminha velozmente para a tela frontal de fósforo, iluminando-a.

- Durante o caminho para a tela, o feixe de elétrons sofrerá uma deflexão que produzirá um ponto brilhante no local desejado da tela.

- A deflexão do feixe de elétrons causará uma varredura nas coordenadas X e Y da tela, denominada rastro ou *raster-scan*.

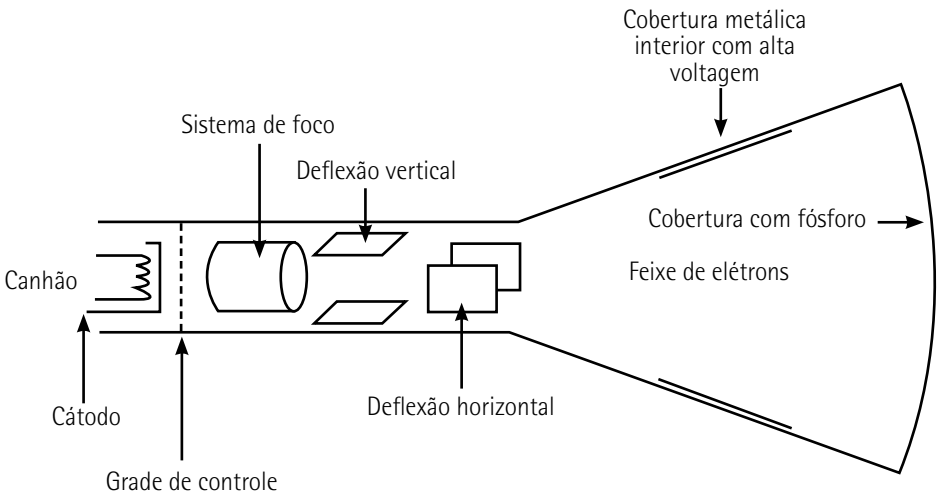

Figura 187 – Tubo de raios catódicos

Saiba mais

Você pode ler mais sobre monitores LCD em:

NOVO processador tem 100 milhões de transistores. *Inovação Tecnológica*, 4 dez. 2002. Disponível em: https://bit.ly/2PXsWwt. Acesso em: 10 mar. 2021.

ULTRADOWNLOADS. Como funcionam os monitores LCDs?. *Canaltech*, 3 abr. 2012. Disponível em: https://bit.ly/3taLcAw. Acesso em: 11 mar. 2021.

8 ARQUITETURA RISC, PROCESSADORES SUPERESCALARES, *MULTITHREADING* E *MULTICORE*

8.1 Arquitetura RISC

As arquiteturas de computadores do tipo RISC (*reduced instruction set computer* ou computador com um conjunto reduzido de instruções) foram um grande avanço no desenvolvimento das arquiteturas dos processadores modernos. Essa arquitetura trouxe muitas novas questões em seu projeto como as que veremos a seguir (MONTEIRO, 2019).

Menor quantidade de instruções e todas com largura fixa

Entre as principais características do sistema RISC está a sua tendência natural de possuir um conjunto de instruções menor se comparado com os das máquinas CISC (*complex instrucion set computer* ou computador com conjunto complexo de instruções) com a mesma configuração de *hardware*. Essa arquitetura teve seu início com a família Spark da SUN Microsystems, por volta da década de 1980, e possuía cerca de 50 instruções. Como comparação, os VAX-11/780 eram totalmente CISC e possuíam até 300 instruções, já o Intel 80486 foi lançado com a capacidade de 147 instruções. Vale ressaltar que uma menor quantidade de instruções, e cada uma delas possuindo uma execução otimizada, resultará em um sistema com melhor desempenho geral, mesmo se houver menos quantidades de instruções, o que pode ocasionar programas um pouco mais longos se houvesse mais instruções. Outras vantagens podem ser obtidas utilizando-se um conjunto reduzido de instruções como:

- Uma menor quantidade de transistores no *chip* do processador, obtendo-se assim um menor espaço físico para a implementação do VLSI (*very large scale integration* – integração em escala muito grande) e, consequentemente, reduzindo-se o custo.

- Redução na complexidade do compilador ou decodificador de instruções, com uma menor largura de entrada e saída, reduzindo-se então o tempo total para a decodificação das instruções.

Como as instruções serão menores, também haverá uma menor quantidade de *bits* no campo do código da operação ou *opcode* da instrução, o que reduzirá o tamanho dos programas.

Execução otimizada de chamada: funções

Outra característica de suma importância nas arquiteturas RISC, que se difere das arquiteturas CISC, refere-se ao modo de realizar chamadas para as rotinas de parâmetros. Algumas pesquisas mostraram que os programas utilizam muito as chamadas funções, consumindo um tempo razoável do processador, mas utilizando usualmente poucos dados no processo, consumindo muito processamento na transferência.

Para as máquinas CISC, a chamada rotina de funções conduz a operações que envolvem escrita/leitura constantes na memória principal, a fim de passar os parâmetros que serão lidos ou gravados. Já para máquinas com arquitetura do tipo RISC, essa transferência ocorre basicamente no processador, de

forma que para que não se perca o desempenho, utiliza-se uma quantidade maior de registradores do que os usualmente encontrados nas arquiteturas CISC.

Menor quantidade de modos de endereçamento

Mais um fator preponderante na arquitetura RISC está relacionado aos modos de endereçamento de memória pelo conjunto de instruções. Máquinas CISC tendem a uma quantidade maior de endereçamentos, de modo que uma operação simples para soma de dois valores armazenados em registradores possa ocorrer de forma mais rápida, a menos que um dos operandos utilizados na soma esteja na memória e o outro operando no registrador, ou pior, com os dois operandos contidos na memória, o que tornará o processo mais demorado. Isso devido a esse problema apresentado.

As arquiteturas RISC buscam por soluções mais simples, e uma delas ficou conhecida como *load/store* (*load* – transfere o dado oriundo da memória para o registrador específico do processador, *store* – operação contrária ao *load*, que irá transferir um dado do registrador específico para algum endereço de memória). Embora pareça simples, essa técnica de implementação de execução de instruções diminui drasticamente os ciclos de *clock* necessários para o processamento da operação.

Modos de execução baseados no uso de *pipeline*

Como já abordado anteriormente, técnicas de *pipeline* tendem a um desempenho elevado na execução de instruções em paralelo. Porém, para que se extraia o seu melhor desempenho, é preciso que as instruções possuam formato e complexidade similares, ocasionando um tempo redundante em todas as etapas, equilibrando o tempo de execução em cada uma delas.

Execução de cada instrução em um ciclo de *clock*

Computadores do tipo CISC, em geral, levam um tempo maior para a execução de instruções devido, entre outros fatores, ao uso de um microcódigo, que implicará a interpretação de cada uma das micro-operações envolvidas no processo, que causará um atraso total na execução das instruções. Em arquiteturas RISC, por outro lado, ao utilizarem poucas instruções e todas elas mais simples, não haverá a necessidade do uso de microcódigos.

8.1.1 Arquitetura RISC da IBM

A arquitetura do tipo RISC foi desenvolvida na década de 1980 pela IBM no laboratório de pesquisas Thomas J. Watson em Nova Iorque (TANEMBAUM; AUSTIN, 2013). O projeto tinha o nome de IBM 801 (figura a seguir), devido ao nome do edifício onde a pesquisa estava ocorrendo. A IBM tinha como objetivo desenvolver um processador para o gerenciamento de uma central telefônica de grande porte, capaz de manipular, simultaneamente, 300 chamadas por segundo. Como requisito inicial, o 801 da IBM deveria completar a execução de uma instrução a cada ciclo de *clock*, além de possuir um desempenho de 12 MIPS, de modo que seria capaz de executar até 20.000 instruções por chamadas. Embora seu projeto inicial fosse o emprego do 801 na central telefônica, ele se mostrou muito popular no emprego do processamento de grandes *mainframes*.

Na sequência, o processador do 801 passou a ser chamado ROMP (Research/OPD microprocessor ou microprocessador do departamento da IBM), em Austin, Texas (Office Products Division). Uma evolução do IBM 801 foi o *mainframe* RS/6000 (figura a seguir), que se tornou um sucesso de vendas devido ao seu alto desempenho.

Figura 188 – Mainframe IBM RS/6000

O RS/6000 possuía entre as suas principais características:

- Foi desenvolvido para ser utilizado também como estação de trabalho ou minicomputador.
- Tamanho fixo de instruções com 4 bytes.
- Acesso à memória principal utilizando o esquema *load*/*store*.
- Possuía 32 registradores de 32 bits para realizar o endereçamento de memória.
- Possuía 32 registradores de 64 bits para a realização de operações em ponto flutuante.
- Utilizava somente a ULA e sem acesso à memória para operações de lógica e aritmética.
- Uso de *pipeline* em unidades independentes de execução, além de quatro modos de endereçamento.
- Possuía um conjunto de 183 instruções, com nove formatos diferentes além de quatro modos para o endereçamento.
- Endereços de 32 bits, permitindo um endereçamento de memória de até 4 GB.
- *Caches* L1, uma para dados e outra para instruções.

Lembrete

O *pipeline* serve para executar instruções em paralelo e geralmente é utilizado quando se quer melhorar o desempenho de processamento.

8.1.2 *Pipeline* na arquitetura RISC

Para máquinas do tipo RISC, a maioria das instruções é do tipo registrador-para-registrador, envolvendo apenas dois estágios de *pipeline*, como segue:

- I: busca a instrução.
- E: executa a instrução (efetua uma operação entre a ULA e os registradores).

Em operações de carregamento e armazenamento em memória principal, serão necessários três estágios de *pipeline*, como segue:

- I: busca a instrução.
- E: executa a instrução (calcula um endereço da memória principal).
- D: operação de armazenamento entre o registrador e a memória ou vice-versa.

A figura a seguir mostra um exemplo para o tempo decorrido de uma sequência de instruções sem *pipeline* executada sequencialmente. Nesse exemplo, observa-se claramente um desperdício de tempo de execução de instruções visto que elas são executadas em série, somente paralelizando as instruções ao término das instruções anteriores.

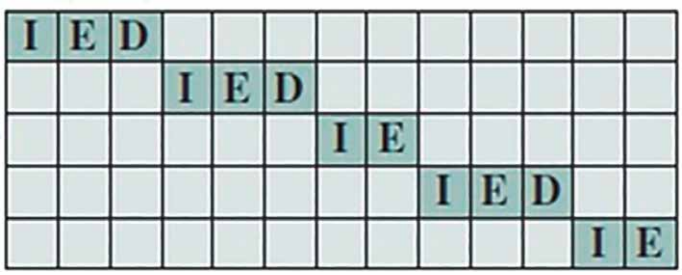

Figura 189 – Execução de instruções sequencialmente

Já a figura a seguir mostra uma pequena melhoria no processamento paralelo, visto que, neste exemplo, utiliza-se um *pipeline* de dois estágios, sendo eles I e E executados de forma simultânea.

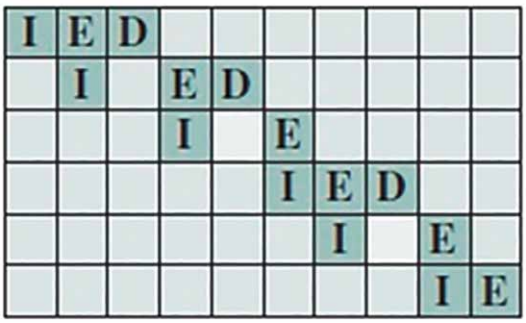

Figura 190 – *Pipeline* de dois estágios

Nesse exemplo, os dois primeiros estágios do *pipeline* serão para a realização da busca da instrução e um estágio para a execução, além do armazenamento em memória da instrução resultante, o que incluirá operações de registrador para a memória ou memória para o registrador. Como pode ser observado na figura, o estágio de busca de instrução seguinte poderá ser executado em paralelo, juntamente à primeira parte do estágio, que irá realizar a operação execução/memória. Porém, o estágio que realiza a execução/memória da segunda instrução deverá ser atrasado, até que a primeira instrução possa esvaziar o segundo estágio. Mesmo com essa penalidade, o *pipeline* de dois estágios pode aumentar em até duas vezes a taxa de execução, se comparado com um sistema de execução sequencial. Ainda assim, dois problemas impedem que seja obtido o aumento máximo na velocidade do *pipeline*, sendo o primeiro em relação ao acesso à memória, que, como se sabe, nessa situação será única, e será utilizado apenas um acesso por estágio. O segundo problema envolve o fato de que, quando ocorre um desvio na instrução, o fluxo sequencial de execução também será interrompido, de forma que, para acomodar isso utilizando-se a menor quantidade de estágios possíveis, insere-se uma instrução a mais no ciclo de estágios, o que, claramente, irá resultar em um ciclo de *pipeline* mais longo e demorado.

Devido a esses problemas mencionados, o *pipeline* ainda pode ser melhorado, permitindo, por exemplo, dois acessos simultâneos à memória para cada estágio. Isso resultará na situação em que três instruções poderão ser sobrepostas, obtendo um melhoramento com fator 3. Essas melhorias podem ser visualizadas na figura a seguir.

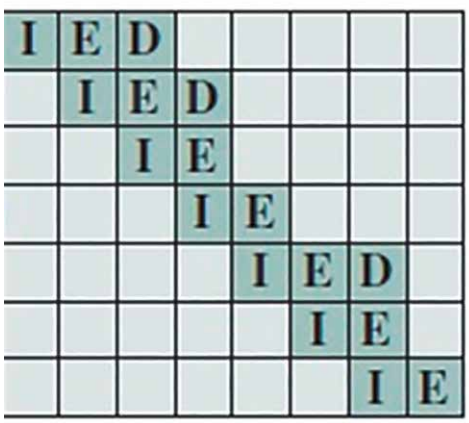

Figura 191 – *Pipeline* de três estágios

Da mesma forma, se alguma instrução de desvio entrar no *pipeline*, ocorrerá que a aceleração máxima nunca seja alcançada. Além disso, algumas dependências de dados também poderão afetar esse circuito. Se alguma instrução necessitar alterar algum operando obtido da instrução anterior, ocorrerá um atraso, de forma que, novamente, uma instrução adicional será inserida no *pipeline*. No de três estágios, assim como nos anteriores, é de interesse que todos os estágios possuam o mesmo tempo de duração, ou seja, teoricamente o mesmo tamanho. Como o estágio E geralmente envolve alguma operação de ULA, ele será sempre mais demorado; dessa forma, pode-se dividir o estágio de execução em dois subestágios:

- E_1: realiza a leitura no banco de registradores.
- E_2: realiza a operação na ULA e a escrita em um registrador.

Como as instruções na arquitetura RISC são mais simples e regulares, um projeto com três ou quatro estágios será facilmente implantado. A figura a seguir mostra o resultado de um *pipeline* com quatro estágios.

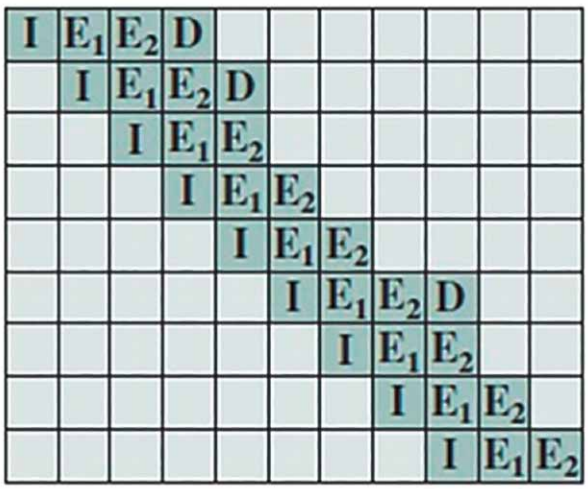

Figura 192 – *Pipeline* de quatro estágios

8.2 Processadores superescalares e *superpipeline*

Uma das grandes vantagens da implementação de *pipelines* múltiplos é justamente o aumento no nível de paralelismo de instruções, o que possibilita múltiplos fluxos que podem ser processados simultaneamente. Em vez da busca de somente uma instrução por vez, como nos sistemas convencionais, um processador superescalar possui como tarefa buscar múltiplas instruções simultaneamente, realizando, na sequência, uma tentativa de localização das próximas instruções, para que, independentemente, possa executá-las em paralelo. Entretanto, assim como em um sistema de *pipeline* comum, se a entrada de alguma operação dada por uma instrução for dependente da saída da instrução anterior, então a instrução seguinte não poderá completar a execução da tarefa ao mesmo tempo ou mesmo antes da instrução anterior. Dessa forma, uma vez que todas as dependências no processo sejam identificadas,

o processador poderá executar, e até mesmo completar, todas as instruções em uma ordem que seja diferente do código de máquina originalmente programado.

Observação

Um processador superescalar pode ser definido como aquele que possui múltiplos e independentes *pipelines* de instruções. Assim, uma implementação do tipo superescalar, implementada na arquitetura de um processador que possui instruções do tipo aritméticas de inteiros, de ponto flutuante, leitura/escrita e desvios condicionais, pode realizar essas tarefas independentes de forma simultânea (STALLINGS, 2010).

Uma forma de contornar tal problema seria o processador eliminar algum tipo de dependência de dados ou instruções desnecessárias, utilizando-se para isso de registradores adicionais, renomeando assim as referências obtidas dos registradores no código original. O projeto de processadores superescalares foi implementado em 1987, logo após o desenvolvimento da arquitetura RISC, embora a abordagem superescalar também possa ser implementada nas arquiteturas CISC. A figura a seguir mostra um comparativo entre uma organização escalar simples e uma organização de processamento superescalar.

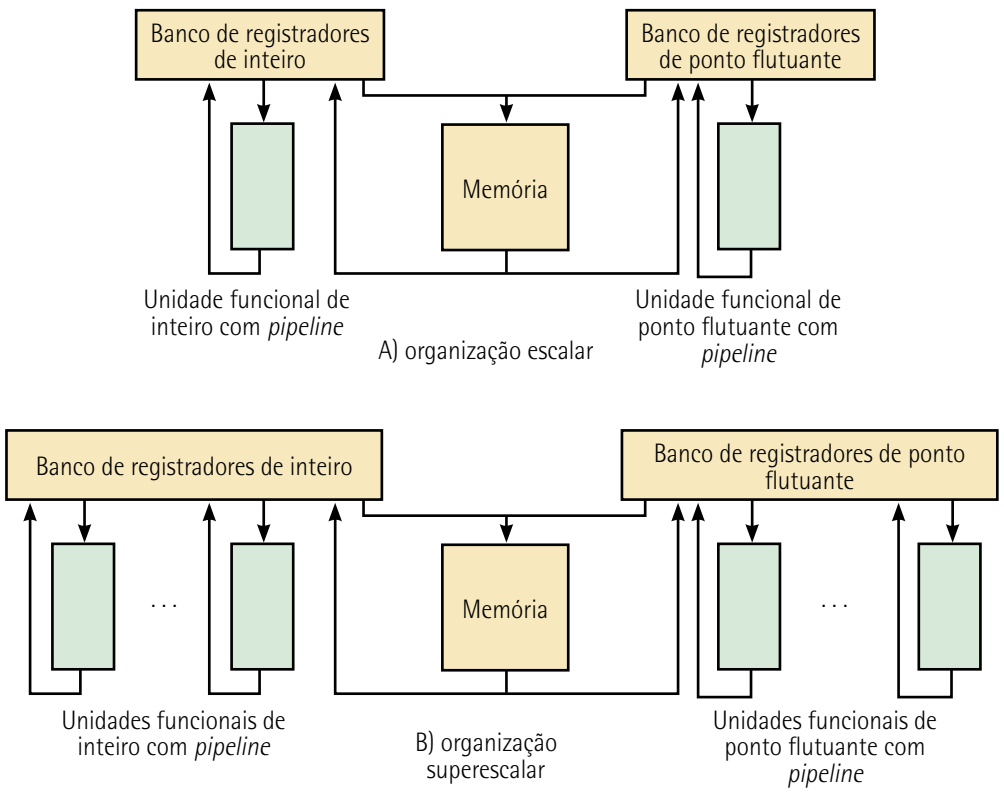

Figura 193 – A) organização escalar; e B) organização superescalar

Em uma organização escalar convencional (figura 193 A), existe uma unidade funcional em um *pipeline* único para realizar operações inteiras e outro *pipeline* para realizar operações de ponto flutuante. Nesse tipo de organização, o paralelismo será obtido quando se habilitam múltiplas instruções em diferentes estágios do *pipeline* de uma só vez. Já em uma organização superescalar, existem várias unidades funcionais, em que cada uma será implementada com um único *pipeline*. Cada uma dessas unidades implementará um grau de paralelismo em função da estrutura. Assim, será de responsabilidade do *hardware* e do compilador garantir que a execução em paralelo não viole algum propósito do programa original.

8.2.1 Comparativo entre processadores superescalares e *superpipeline*

A evolução da organização de *pipeline* simples ficou conhecida como *superpipeline* e foi desenvolvida em 1988. A sua organização trabalha com o fato de que possam se realizar vários estágios de *pipeline*, executando tarefas que irão requerer metade do tempo de um ciclo de *clock*. Dessa forma, a velocidade de *clock* possibilitará que duas tarefas sejam realizadas em apenas um único ciclo. A figura a seguir mostra um comparativo entre as abordagens de *pipeline* simples, superescalar e *superpipeline*, todas considerando quatro estágios: buscar, decodificar, executar e escrever.

Na figura, observa-se que, ainda que as várias instruções possam ser executadas concorrentemente, somente uma delas estará em seu estágio de execução. A parte seguinte da figura mostra uma implementação do *superpipeline*, que é capaz de executar dois estágios de *pipeline* em um único ciclo de *clock*. Dessa forma, observa-se que as funções executadas em cada estágio poderão ser divididas em duas partes, não ocorrendo sobreposição, e cada uma delas poderá ser executada na metade de um ciclo de *clock*. Por fim, na parte mais baixa da figura, é possível observar uma implementação superescalar, capaz de executar duas instâncias dentro de cada estágio em paralelo. É válido observar que tanto a implementação da organização *superpipeline* como a organização superescalar devem possuir o mesmo número de instruções executando ao mesmo tempo, em um determinado estado. Assim, o processador de *superpipeline* ficará atrasado em relação ao processador superescalar no início de cada programa ou quando houver algum desvio.

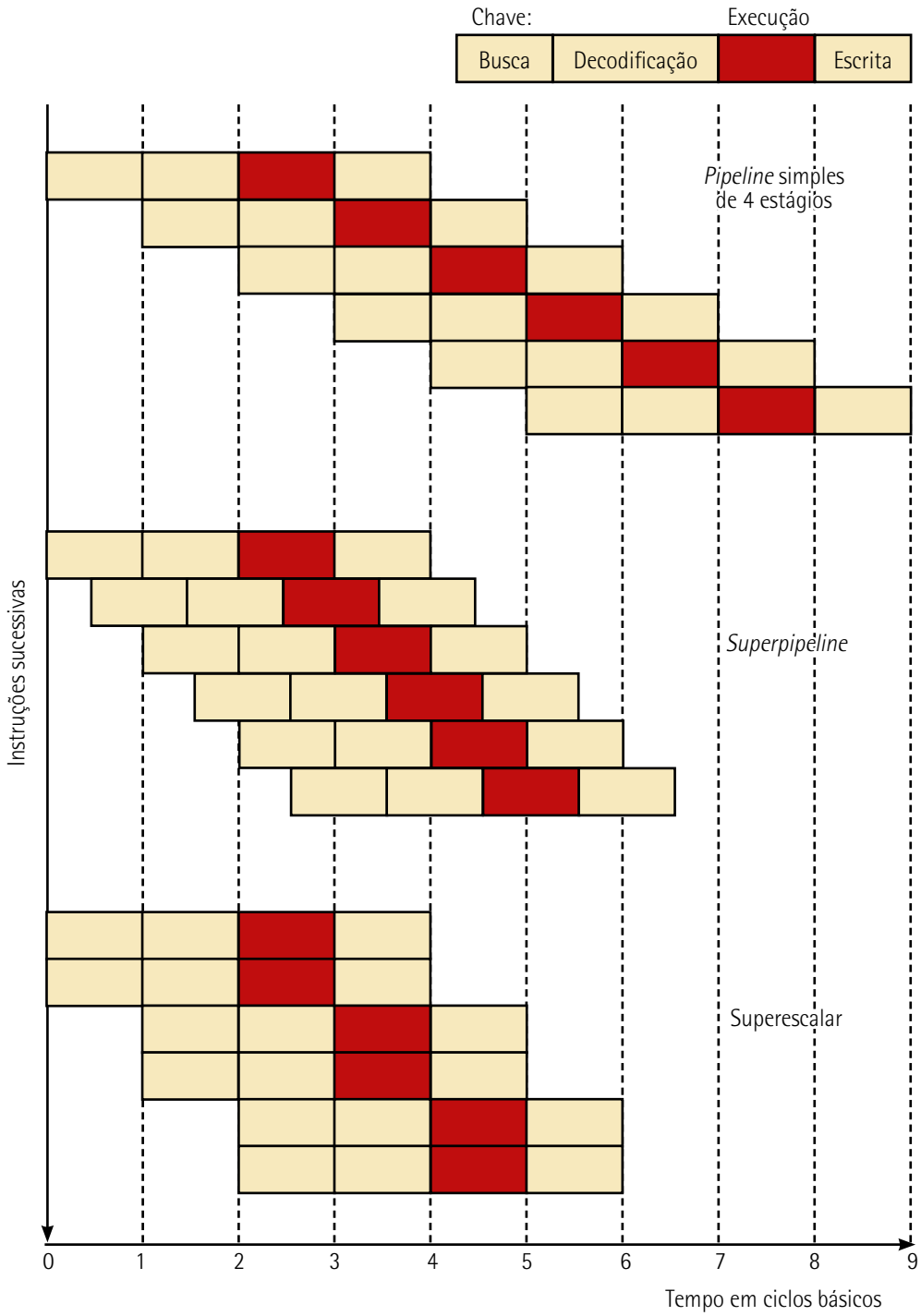

Figura 194 – Comparação entre as abordagens de *pipeline* simples, *superpipeline* e superescalar

8.2.2 Limitações

O paralelismo em nível de instruções também pode conter algumas limitações ao executar tarefas em paralelo. Assim, é necessária uma combinação ótima entre as técnicas de implementação de paralelismo em *hardware* em conjunto com o compilador, a fim de maximizar o paralelismo das instruções. Algumas limitações no paralelismo em nível de instrução podem ser listadas:

- **Dependência de dados verdadeira**: quando uma segunda instrução só pode ser executada ao término de execução da primeira instrução, devido ao compartilhamento dos dados ou mesmo do uso do resultado da primeira operação pela segunda operação.

- **Dependência procedural**: quando há algum desvio na sequência de execução das instruções, complicando a operação do *pipeline*.

- **Conflitos de recursos**: quando há uma competição de duas ou mais instruções pelos mesmos recursos (memória, processador, dado, instrução) simultaneamente.

- **Dependência de saída**: quando duas ou mais instruções necessitam armazenar seu resultado em um mesmo endereço e ao mesmo tempo.

- **Antidependência**: quando uma instrução, ao utilizar um local ou operando, entra em conflito com a instrução seguinte que também tenta utilizar o mesmo local ou operando.

A figura a seguir ilustra a execução de instruções em paralelo, utilizando um *pipeline* de quatro estágios, que compara situações sem dependência com as dependências de dados, procedural e conflito de recursos. Como se observa, em um sistema paralelo sem dependências, as instruções podem ser executadas simultaneamente sem nenhum tipo de atraso. Quando há uma dependência de dados, o estágio de execução terá um atraso em um ciclo de *clock*, aguardando a obtenção do resultado da primeira instrução logo acima. Na dependência procedural há um desvio de instruções, que deverão ser executadas primeiramente deixando o fluxo para execução das instruções atrasadas, aguardando o estágio de "executar" finalizar. Quando há algum conflito de recurso, novamente o estágio "executar" na primeira operação deve ser finalizado para que o processamento paralelo possa utilizar o dado resultante, ou mesmo o uso de memória/processador para que seu processo seja de fato executado e escrito.

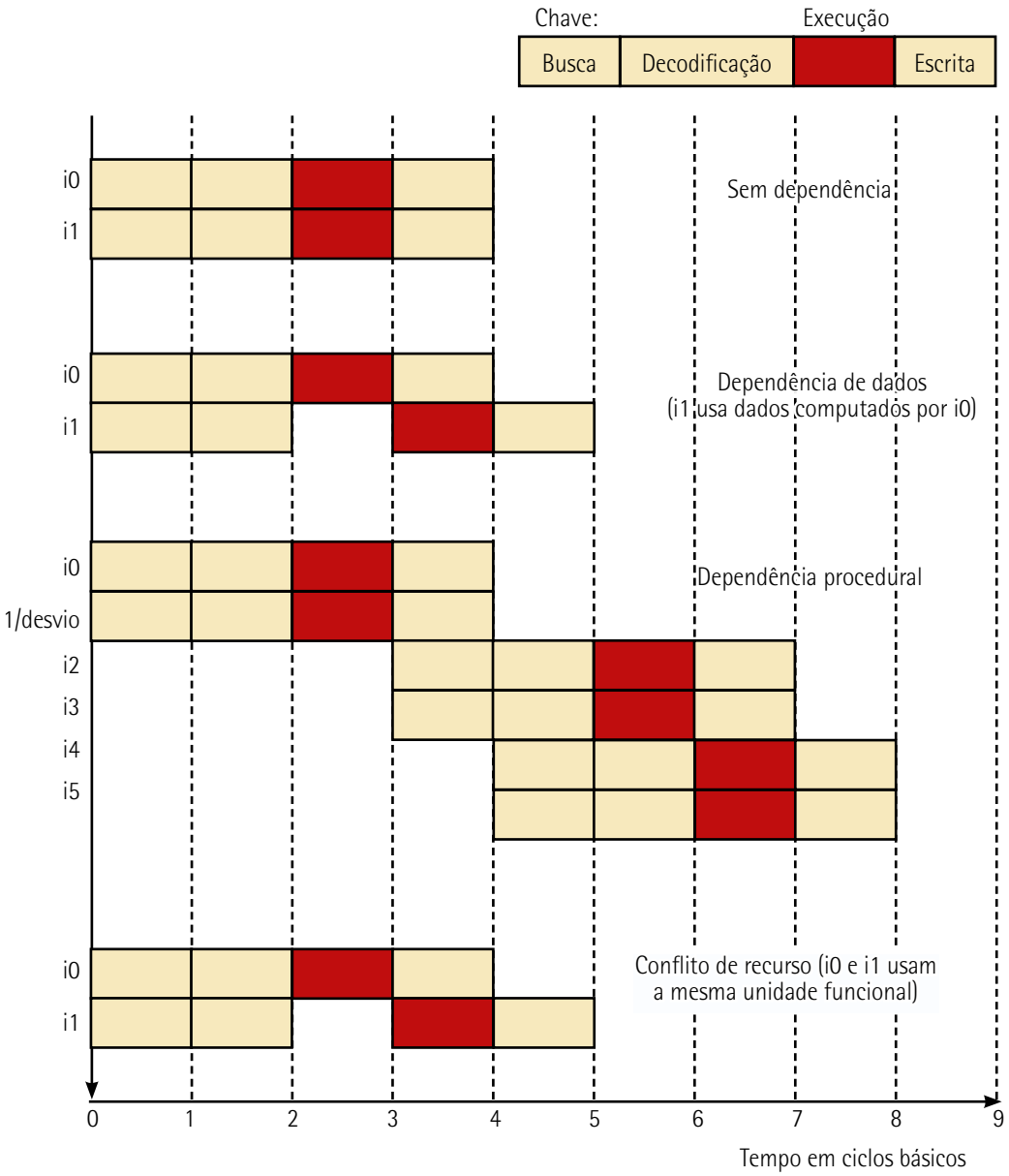

Figura 195 – Efeito de dependências em um sistema de processamento superescalar

8.3 *Multithreading*

Uma abordagem utilizada como alternativa para que se permita um número maior no paralelismo, se comparado com os esquemas superescalar ou *superpipeline*, é conhecida como *multithreading*.

No *multithreading*, os vários ciclos de instruções são divididos em ciclos menores, também conhecidos como *threads*, onde cada *thread* poderá ser executado em paralelo. Além disso, ele é subdividido entre implícito e explícito.

Na abordagem de *multithreading* explícito é de suma importância que o processador disponibilize um contador de programa (*program conter* – PC), para que cada *thread* em execução possa ser executado concorrentemente. Já o *multithreading* implícito se refere a uma execução concorrente de vários *threads* retirados de um único programa, que, originalmente, estava sendo executado de forma sequencial. Dessa forma, os *threads* implícitos também podem ser definidos de maneira estática pelo próprio compilador do *software* utilizado, ou de forma dinâmica pelo *hardware* do computador.

O *multithreading* pode se basear no uso de várias técnicas já conhecidas, a fim de otimizar a execução de um *thread* singular, como, por exemplo, a previsão de desvio, renomeação de registradores ou mesmo técnicas de processamento superescalar. As diversas técnicas de *multithreading* explícitos podem ser divididas entre:

- ***Multithreading* intercalado, também conhecido como *multithreading* de granularidade fina**: nesse esquema, o processador pode operar dois ou mais contextos de *thread* simultaneamente, alternando entre eles a cada ciclo de *clock*. Se algum *thread* for bloqueado devido a alguma dependência de dados ou mesmo por falta de recursos como memória disponível, ela será alternada para um que esteja pronto para ser executado.

- ***Multithreading* bloqueado, também conhecido como *multithreading* de granularidade grossa**: nesse esquema, as instruções de um *thread* serão executadas de forma progressiva até que algum novo evento ocorra, causando um atraso. Esse evento induzirá uma troca de *thread* e só será eficiente se o processador executar as tarefas de forma ordenada e se possível sem nenhuma falha de *cache*.

- ***Multithreading* simultâneo (SMT)**: nesse tipo, as instruções a serem executadas são enviadas de forma simultânea a partir de vários *threads* para as unidades de execução de um processador superescalar. Assim, combinará a capacidade de envio de instruções superescalares através do uso de múltiplos contextos de *thread*.

- ***Chip* multiprocessado**: nesse caso, o processador inteiro será replicado em um único *chip*, de forma que cada processador irá operar *threads* separados. Uma das grandes vantagens dessa abordagem é que sua área lógica disponível no *chip* será utilizada eficientemente sem a dependência do crescimento da complexidade no projeto do *pipeline*.

A figura a seguir faz um comparativo entre algumas abordagens de arquiteturas de *pipeline* envolvendo o uso de *multithreading* e as compara com outras abordagens também de *pipeline*, mas que não usam o *multithreading*.

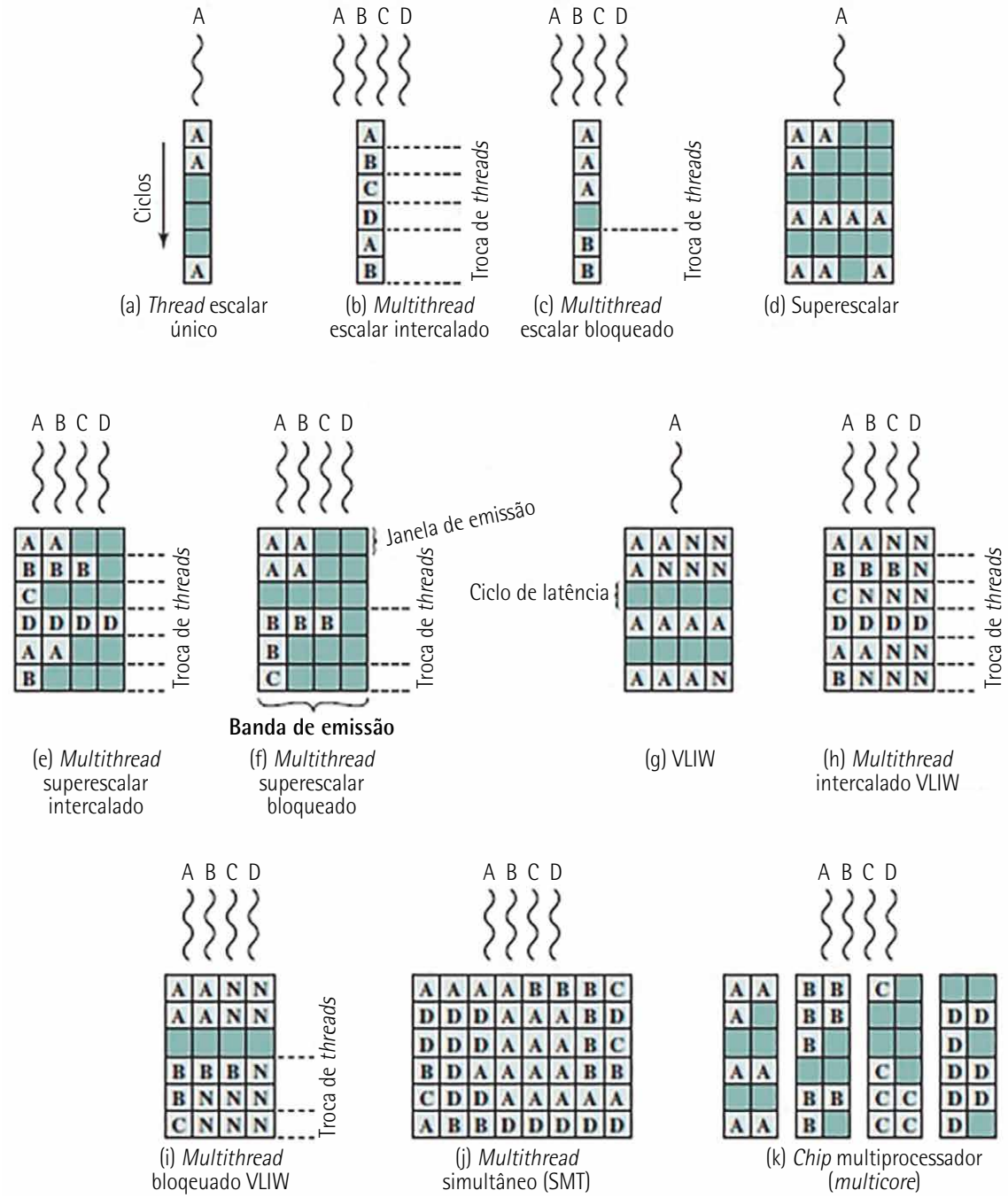

Figura 196 – Diferentes tipos de abordagens para execução de múltiplos *threads*

As linhas horizontais da figura representam os *slots* de envio de instruções para o ciclo de execução, ou seja, dessa forma a largura de cada linha irá corresponder ao número máximo de instruções que serão emitidas a cada ciclo de *clock*. Já as linhas levemente curvadas acima de cada item representam os *threads* únicos ou múltiplos de instruções sendo adicionados no esquema. Na vertical, é representada a sequência de tempo de cada ciclo de *clock*. Os *slots* vazios, sombreados na figura, representam *slots* de execução que não são utilizados no processo de *pipeline*. Cada um dos processos é definido a seguir:

- ***Thread* escalar única**: tipo de *pipeline* simples, encontrado tanto em máquinas RISC quanto CISC tradicionais sem o uso de *multithreading*.

- ***Multithread* escalar integrado**: a abordagem *multithreading* simples e fácil de ser implementada. No *multithread* escalar intercalado, uma troca de *thread* para outro ocorrerá a cada ciclo de *clock* e os múltiplos estágios de *pipeline* poderão ser mantidos ocupados ou parcialmente ocupados.

- ***Multithread* escalar bloqueado**: nesse tipo de *multithread*, apenas um único *thread* será executado, até que ocorra um evento que causasse algum atraso no processo de *pipeline*, sendo que nesse momento o processador realizaria a troca de um *thread* para outro.

- **Superescalar**: a abordagem superescalar básica não possui *multithread* e somente alguns ciclos de *slots* de envio serão utilizados, ocasionando o que também é conhecido como perda horizontal. Em outros ciclos de instruções nenhum *slot* de envio será utilizado e esses ciclos são conhecidos como perda vertical.

- ***Multithread* superescalar intercalado**: durante a execução dos ciclos de *clock* são emitidas diversas instruções, o tanto quanto forem possíveis naquele tempo determinado, de forma que um único *thread* seja capaz de processá-lo. Logicamente isso ocasiona atrasos potenciais, pois haverá uma troca de *threads* que não serão possíveis de serem executados e serão eliminados. Entretanto, o número de instruções que são enviadas para qualquer ciclo pode ser limitado de acordo com as suas dependências, seja de dados ou recursos dentro daquele *thread*.

- ***Multithread* superescalar bloqueado**: nessa situação, as instruções de apenas um *thread* podem ser emitidas durante o ciclo e, como consequência, o *multithread* bloqueado poderá ser utilizado.

- ***Slot* de envio ou VLIW (palavra de instrução muito longa, do inglês *very long instruction word*)**: o emprego de uma VLIW é geralmente elaborado em conjunto com o compilador, que coloca operações que poderão ser executadas em paralelo, numa mesma palavra. Em máquinas VLIW simples não é possível preencher uma palavra de modo completo com instruções que ainda serão emitidas em paralelo.

- **VLIW *multithread* intercalado**: oferece uma eficácia semelhante àquela do *multithread* intercalado para uma arquitetura superescalar.

- ***Multithread* VLIW bloqueado**: oferece uma eficácia semelhante àquela do *multithread* bloqueado para uma arquitetura superescalar.

- ***Multithreading* simultâneo**: esquema capaz de emitir oito instruções simultâneas. Nele, se um *thread* possuir um alto grau de paralelismo em nível de instruções, ele pode ser capaz de preencher todos os *slots* horizontais.

- ***Chip multicore***: combina dois ou mais processadores em um único *chip*.

Saiba mais

Descubra mais sobre os processadores *multithreading* e *hyperthreading* em:

DESCOBERTA vulnerabilidade nos processadores com *hyper-threading*. *Inovação Tecnológica*, 18 maio 2005. Disponível em: https://bit.ly/3vhZnWC. Acesso em: 5 mar. 2021.

8.4 *Multicore*

Há pelo menos uma década o uso de *chips* com um único processador atingiu o seu limite de desempenho e de execução de instruções em paralelo. Através do desenvolvimento da arquitetura *multicore* foi possível explorar e melhorar o processamento *multithreading* em vários núcleos do processador. Assim, fica claro que o uso de uma arquitetura que oferece vários núcleos em vez de um único também aumentará o número de níveis de memória *cache*, aumentando com isso o desempenho geral de processamento.

Um processador *multicore* combinará dois ou mais processadores em um único *chip* de silício. Essas mudanças na organização do projeto dos processadores, aumentando o nível de paralelismo no *chip*, facilitou a execução de instruções simultâneas, durante o mesmo ciclo de *clock*. O aumento no número de núcleos em um processador aumenta consequentemente o desempenho do *pipeline*, do processamento superescalar e do *multithreading* simultâneo, pois para cada núcleo haverá no mínimo uma *cache* L1 e uma *cache* L2 compartilhada, de acordo com o modelo do processador, como pode ser observado na figura a seguir.

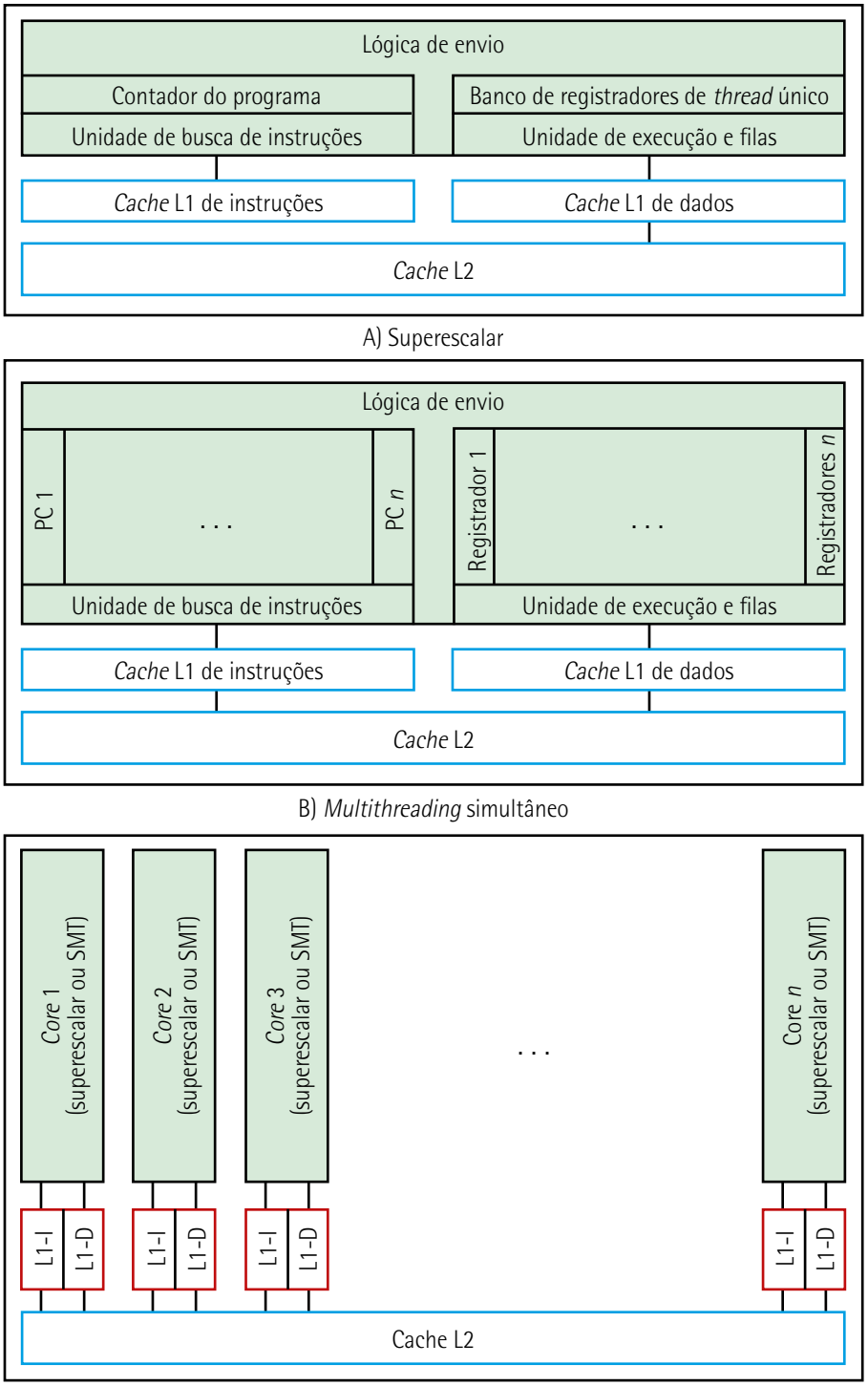

Figura 197 – Organização *multicore*

As principais variáveis envolvidas na organização *multicore* são:

- o número de núcleos no *chip*;
- o número de níveis de memória *cache*;
- a quantidade de memória *cache* a ser compartilhada.

A figura a seguir mostra algumas alternativas na organização da arquitetura *multicore* em que podem ser encontradas *cache* L1 dedicada, *cache* L2 dedicada, *cache* L2 compartilhada e *cache* L3 compartilhada.

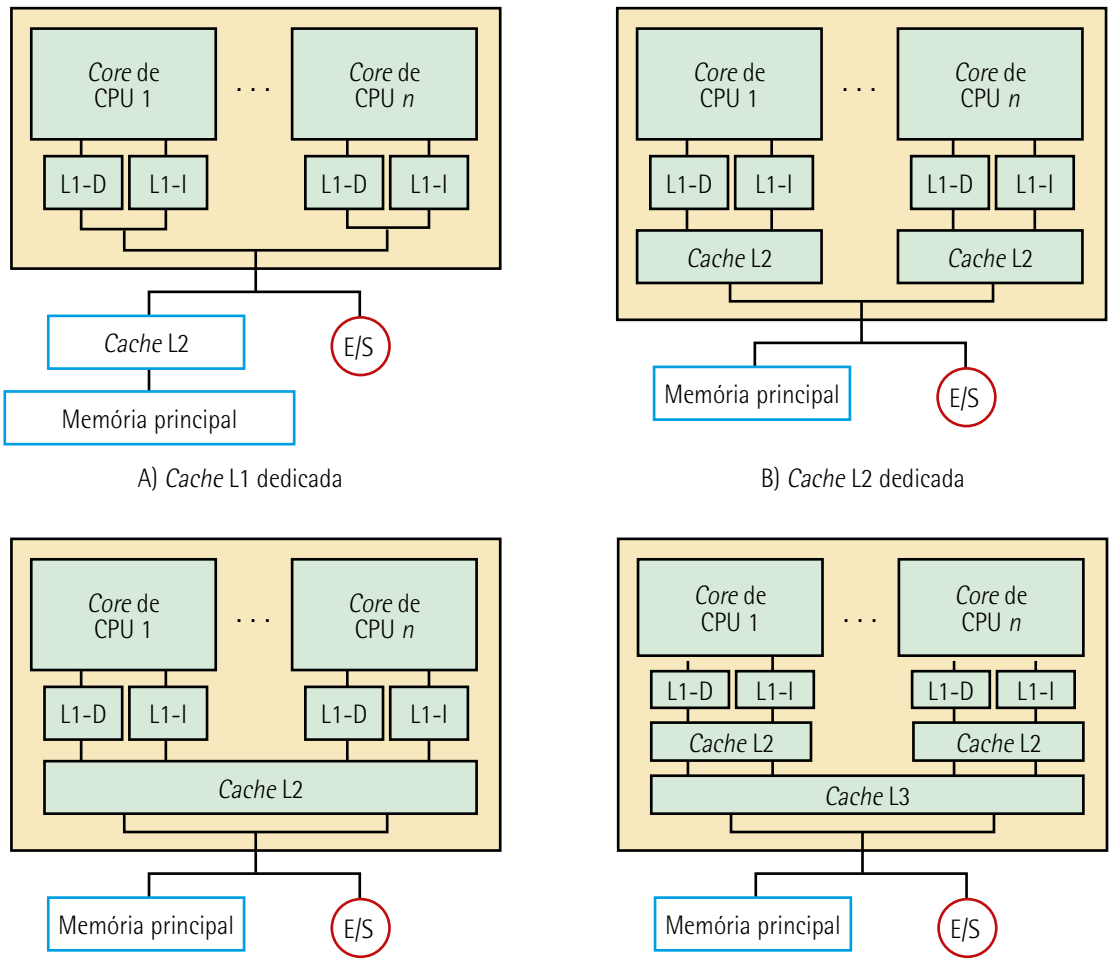

Figura 198 – Organização *multicore*

Uma organização de *cache* L1 dedicada geralmente é dividida entre *cache* de dados e *cache* de instruções e um exemplo típico dessa organização é o processador ARM 11. Em situações de *cache* L2 dedicada há uma área disponível no *chip* que permite o compartilhamento da *cache*. Um exemplo desse

tipo de organização com *cache* L2 dedicada é o processador AMD Opteron. Arquiteturas contendo *cache* L2 e L3 compartilhadas podem conter várias vantagens em relação a *caches* dedicadas, como:

- O uso de interferência construtiva que pode reduzir as taxas gerais de falhas de *cache*, ou seja, se um *thread* em um determinado núcleo acessar uma posição da memória principal, tal operação trará um quadro contendo a posição buscada na *cache* compartilhada. Se outro *thread* em outro núcleo acessar esse mesmo endereço buscando o mesmo bloco logo em seguida, as posições de memória já estarão liberadas disponíveis para uso do *chip*.

- A comunicação entre os pensadores será mais fácil de ser implementada devido às posições de memória compartilhadas.

Está claro que quando se aumenta o poder de processamento de um processador aumentando-se o número de núcleos também é relevante levar em consideração questões relacionadas ao *software*. Arquitetura *multicore* traz diversos benefícios de desempenho e é uma forma de analisar um primeiro comparativo entre o uso de vários núcleos através da fórmula conhecida como lei de Amdahl, que afirma que o ganho (unidade adimensional) na velocidade será igual ao tempo para executar um programa utilizando um único processador pelo tempo de execução desse mesmo programa utilizando N processadores (STALLINGS, 2010). Em termos matemáticos podemos expressar da seguinte forma:

$$\text{Ganho} = \frac{1}{(1 - f_{\text{restante}}) + \frac{f_{\text{utilizada}}}{N}}$$

Onde N é o número de núcleos e f é a fração do tempo de execução para um certo código de programa. Dessa forma, a equação mostra que quando f é pequeno, o uso de processadores em paralelo tem pouco efeito, mas quando N se aproxima de infinito, o ganho é ilimitado. Por exemplo: uma operação de *software* utiliza 40% do tempo em operações de ponto flutuante em uma máquina de seis núcleos. Qual será o ganho resultante?

$$\text{Ganho} = \frac{1}{(1 - 0{,}60) + \frac{0{,}40}{6}} \rightarrow \text{Ganho} = 2{,}142$$

Agora em um novo exemplo: uma operação de *software* utiliza 55% do tempo em operações lógicas em uma máquina de quatro núcleos. Qual será o ganho resultante?

$$\text{Ganho} = \frac{1}{(1 - 0{,}45) + \frac{0{,}55}{2}} \rightarrow \text{Ganho} = 1{,}212$$

Saiba mais

Descubra mais sobre processadores *multicore* em:

NOVO *hardware* acelera comunicação em processadores multicore. *Inovação Tecnológica*, 11 fev. 2011. Disponível em: https://bit.ly/3ciYjJm. Acesso em: 5 mar. 2021.

Resumo

Nesta unidade, foram apresentados conceitos sobre os dispositivos de E/S, como impressoras, monitores, *mouse*, teclado, entre outros. Também foi possível aprender como se dão as comunicações entre os diferentes dispositivos periféricos com o computador.

Também foi possível abordar conceitos específicos sobre as diferentes tecnologias envolvendo monitores de computadores LCD, LED etc.

De forma abrangente, foram discutidos alguns conceitos referentes ao processamento paralelo além do anteriormente visto *pipeline*. Foi possível visualizar melhorias implementadas com o *superpipeline* e o processamento superescalar, que ainda é empregado nos computadores contemporâneos.

Por fim, foi abordado o emprego de múltiplos núcleos nos processadores atuais e qual é o ganho em desempenho que esses vários núcleos podem oferecer, operando em conjunto com *softwares* capazes de processar paralelamente.

Exercícios

Questão 1. Considere a arquitetura de um computador hipotético, no qual são utilizados diversos barramentos diferentes para a comunicação entre as suas diversas partes. Suponha que a largura de banda no barramento que conecta a CPU com a memória esteja saturada, ou seja, que a CPU precise ficar constantemente esperando os dados virem da memória, a uma velocidade muito menor do que a sua capacidade de processamento seria capaz de lidar. Esse problema é chamado de "gargalo de von Neumann", uma vez que está profundamente relacionado a essa arquitetura.

Nesse cenário, avalie as afirmativas a seguir.

I – É possível melhorar o desempenho do computador em análise trocando a CPU atual por uma mais rápida, independentemente da tarefa que será executada.

II – Uma possível forma de melhorar a *performance* dessa arquitetura é utilizando um barramento que tenha maior largura de dados, ou seja, usando mais linhas em paralelo. Obviamente, isso requer uma mudança no projeto, o que pode causar problemas de compatibilidade com dispositivos desenvolvidos anteriormente à modificação.

III – Uma possível forma de melhorar a *performance* dessa arquitetura é aumentar a aceleração do barramento, ou seja, reduzir o tempo para a transmissão dos dados nesse barramento. Essa mudança também envolveria uma mudança no projeto original, o que pode causar problemas de compatibilidade.

É correto o que se afirma apenas em:

A) I.

B) II.

C) III.

D) I e II.

E) II e III.

Resposta correta: alternativa E.

Análise das afirmativas

I – Afirmativa incorreta.

Justificativa: se o principal gargalo do sistema está na velocidade do barramento, o aumento da velocidade da CPU não deve melhorar o desempenho e pode até causar problemas.

II – Afirmativa correta.

Justificativa: aumentar o número de sinais que operam em paralelo pode ser uma forma de aumentar a velocidade do barramento, mas isso à custa de aumento na complexidade da placa e dos conectores.

III – Afirmativa correta.

Justificativa: ao reduzir o tempo para realizar a transmissão de dados, aumenta-se a velocidade do barramento. Contudo, o aumento dessa velocidade pode implicar uma série de dificuldades técnicas, especialmente com relação ao atraso diferencial do barramento. Além disso, podem ocorrer problemas de compatibilidade com dispositivos anteriores.

Vale notar que uma maneira de aliviar o "gargalo de von Neumann" é a utilização de memórias *cache*. Parte da informação é copiada diretamente para dentro da CPU, para essas memórias, de forma que o barramento não precisa ser constantemente acessado para a manipulação de informações. Contudo, essas memórias costumam ser bem menores do que o valor total de memória RAM disponível em uma máquina, o que limita a quantidade de informação que pode ser manipulada nessa situação. Além disso, os valores dessas memórias devem ser mantidos em "sincronismo", o que pode ser um problema mais complexo em sistemas que utilizam vários processadores (ou núcleos) com memórias *cache* separadas. Esses problemas devem ser atacados com o emprego de protocolos (ou mecanismos) de coerência.

Questão 2. Considere as afirmativas a seguir sobre as arquiteturas RISC e CISC.

I – Computadores que utilizam a arquitetura RISC possuem um conjunto de instruções muito maior (ou seja, com muito mais instruções) do que os computadores que utilizam a arquitetura CISC.

II – Computadores que utilizam a arquitetura CISC costumam possuir mais modos de endereçamento que computadores que utilizam a arquitetura RISC.

III – A largura das instruções em um computador que utiliza a arquitetura RISC costuma ser fixa.

É correto o que se afirma apenas em:

A) I.

B) II.

C) III.

D) I e II.

E) II e III.

Resposta correta: alternativa E.

Análise das afirmativas

I – Afirmativa incorreta.

Justificativa: o próprio nome RISC é uma sigla que significa *reduced instruction set computer*, ou computador com conjunto de instruções reduzido. Na arquitetura RISC, o conjunto de instruções tende a ser muito menor que no caso CISC, uma sigla que significa *complex instruction set computer* ou computador com um conjunto complexo de instruções. Nesse caso, "complexo" significa tanto um número maior de instruções quanto instruções de natureza mais complexa do que uma máquina RISC.

II – Afirmativa correta.

Justificativa: outra característica da arquitetura RISC, além de um conjunto de instruções menores, é um número inferior de modos de endereçamento.

III – Afirmativa correta.

Justificativa: além de um número menor de instruções (em comparação com a arquitetura CISC), na arquitetura RISC, as instruções costumam ter largura fixa, o que simplifica alguns aspectos da construção do microprocessador e aumenta o seu desempenho.

FIGURAS E ILUSTRAÇÕES

Figura 1

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 8.

Figura 2

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 9.

Figura 3

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 9.

Figura 4

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 9.

Figura 5

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. Adaptada.

Figura 6

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 4.

Figura 7

ELECTRONIC-2633617_960_720.JPG. Disponível em: https://bit.ly/3bxUQak. Acesso em: 9 mar. 2021.

Figura 8

HOLE-KARTENLOCHER-62897_960_720.JPG. Disponível em: https://bit.ly/3cejDQ9. Acesso em: 9 mar. 2021.

Figura 9

ENIAC.JPG. Disponível em: https://bit.ly/3bAOJm1. Acesso em: 9 mar. 2021.

Figura 10

VACUUMTUBE1.JPG. Disponível em: https://bit.ly/38poxbS. Acesso em: 9 mar. 2021.

Figura 11

A_REPLICA_OF_THE_FIRST_WORKING_TRANSISTOR_02.JPG. Disponível em: https://bit.ly/3v45iyl. Acesso em: 9 mar. 2021.

Figura 12

NASACOMPUTERROOM7090.NARA.JPG. Disponível em: https://bit.ly/38pMyPY. Acesso em: 10 mar. 2021.

Figura 13

IBM_PC_5150.JPG. Disponível em: https://bit.ly/3bBkhYC. Acesso em: 7 fev. 2021.

Figura 14

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 34.

Figura 15

TANENBAUM, A. S. *Sistemas operacionais modernos*. 3. ed. São Paulo: Pearson Education, 2010. p. 5.

Figura 16

TANENBAUM, A. S. *Sistemas operacionais modernos*. 3. ed. São Paulo: Pearson Education, 2010. p. 6.

Figura 17

ILVA, A. W. S. *Conjuntos numéricos na educação básica*: dos naturais aos complexos e suas aplicações matemáticas. 2019. Dissertação (Mestrado em Matemática) – Universidade Estadual do Maranhão, São Luís, 2019. p. 17.

Figura 18

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 19

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 20

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. p. 51.

Figura 21

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. p. 51.

Figura 22

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. p. 52.

Figura 23

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 24

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. p. 54.

Figura 25

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. p. 54.

Figura 26

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. p. 55.

Figura 27

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 28

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. p. 57.

Figura 29

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. p. 57.

Figura 30

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. p. 57.

Figura 31

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 32

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 33

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 34

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 35

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 36

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 37

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 38

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 39

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 40

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 41

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 42

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 43

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011. Adaptada.

Figura 45

CORE_I7_920_QUAD_FRONT_AND_BACK.JPG. Disponível em: https://bit.ly/38rqBjM. Acesso em: 10 mar. 2021.

Figura 47

ILICONCRODA.JPG. Disponível em: https://bit.ly/3clxYtY. Acesso em: 10 mar. 2021.

Figura 48

UASLP_IICO_01.JPG. Disponível em: https://bit.ly/38pNgww. Acesso em: 10 mar. 2021.

Figura 49

WAFER_2_ZOLL_BIS_8_ZOLL.JPG. Disponível em: https://bit.ly/3bz6C4z. Acesso em: 10 mar. 2021.

Figura 50

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 147.

Figura 51

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 160.

Figura 52

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 161.

Figura 53

CARTER, N. *Arquitetura de computadores*. Porto Alegre: Bookman, 2002. p. 49.

Figura 54

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 164.

Figura 55

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 172.

Figura 56

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 42.

Figura 57

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. Adaptada.

Figura 58

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 175.

Figura 59

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 14.

Figura 60

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 15.

Figura 61

MISHRA, N. Difference between Von Neumann and Harvard Architecture. *The Crazy Programmer*, [s.d.]. Adaptada. Disponível em: https://bit.ly/3tdr6WB. Acesso em: 10 mar. 2021.

Figura 62

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 16.

Figura 63

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. Adaptada.

Figura 64

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. Adaptada.

Figura 65

INTEL_4004.JPG. Disponível em: https://bit.ly/2ODJXuW. Acesso em: 10 mar. 2021.

Figura 66

INTEL_8008.JPG. Disponível em: https://bit.ly/30sKnXw. Acesso em: 10 mar. 2021.

Figura 67

MITSUBISHI_M5L8080AP_1.JPG. Disponível em: https://bit.ly/3t8fkfX. Acesso em: 10 mar. 2021.

Figura 68

KL_INTEL_D8086.JPG. Disponível em: https://bit.ly/2OnHGUC. Acesso em: 10 mar. 2021.

Figura 69

PANNAIN, R.; BEHRENS, F. H; PIVA JR., D. *Organização básica de computadores e linguagem de montagem*. São Paulo: Elsevier, 2012. p. 243.

Figura 70

PANNAIN, R.; BEHRENS, F. H; PIVA JR., D. *Organização básica de computadores e linguagem de montagem*. São Paulo: Elsevier, 2012. p. 246.

Figura 71

KL_IBM_80286.JPG. Disponível em: https://bit.ly/2N5mkuF. Acesso em: 10 mar. 2021.

Figura 72

800PX-KL_INTEL_I386EX.JPG. Disponível em: https://bit.ly/30yfvoK. Acesso em: 7 fev. 2021.

Figura 73

Intel_80486DX2_top.jpg. Disponível em: https://bit.ly/3l3aAFD. Acesso em: 10 mar. 2021.

Figura 74

KL_INTEL_PENTIUM_120.JPG. Disponível em: https://bit.ly/3exQBO3. Acesso em: 7 fev. 2021.

Figura 75

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 56.

Figura 76

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 362.

Figura 77

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 59.

Figura 78

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 285.

Figura 79

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 285.

Figura 80

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 287.

Figura 81

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 288.

Figura 82

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 289.

Figura 83

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 321.

Figura 84

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 61.

Figura 85

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 61.

Figura 86

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 62.

Figura 87

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 62.

Figura 88

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 62.

Figura 89

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 64.

Figura 90

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 65.

Figura 91

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 65.

Figura 92

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 66.

Figura 93

ARTHUR, C. Apple faces its "Nike moment" as ABC Nightline goes inside Foxconn. *The Guardian*, 20 fev. 2012. Disponível em: https://bit.ly/3ckx1Cd. Acesso em: 17 out. 2020.

Figura 94

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 365.

Figura 95

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 365.

Figura 96

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 366.

Figura 97

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 367.

Figura 98

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 368.

Figura 99

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 370.

Figura 100

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 370.

Figura 101

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 371.

Figura 102

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 371.

Figura 103

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 372.

Figura 104

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 375.

Figura 105

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 375.

Figura 106

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 290.

Figura 107

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 290.

Figura 108

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 290.

Figura 110

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 79.

Figura 111

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 82.

Figura 112

TWO-REELS-OLD-AUDIO-MAGNETIC-TAPES-ISOLATED_124717-345.JPG. Disponível em: https://bit.ly/3qF4jRv. Acesso em: 23 fev. 2021.

Figura 113

HARD-DISK-DRIVE-DISKDRIVE_19-112708.JPG. Disponível em: https://bit.ly/3esywB6. Acesso em: 23 fev. 2021.

Figura 114

MEMORY-CHIPS-57269_960_720.JPG. Disponível em: https://bit.ly/3chAphf. Acesso em: 23 fev. 2021.

Figura 115

MICROCHIPS-4924170_960_720.JPG. Disponível em: https://bit.ly/3chP3VP. Acesso em: 25 fev. 2021.

Figura 116

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 107.

Figura 117

NULL, L.; LOBUR, J. *Princípios básicos de arquitetura e organização de computadores*. Porto Alegre: Bookman, 2010. p. 313.

Figura 118

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 96.

Figura 119

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 96.

Figura 120

CARTER, N. *Arquitetura de computadores*. Porto Alegre: Bookman, 2002. p. 159.

Figura 121

CARTER, N. *Arquitetura de computadores*. Porto Alegre: Bookman, 2002. p. 159.

Figura 122

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 99.

Figura 123

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 99.

Figura 124

CARTER, N. *Arquitetura de computadores*. Porto Alegre: Bookman, 2002. p. 162.

Figura 125

CARTER, N. *Arquitetura de computadores*. Porto Alegre: Bookman, 2002. p. 163.

Figura 126

800PX-AMIGA_1200_KICKSTART_3.0_ROMS.JPG. Disponível em: https://bit.ly/3ezSGJc. Acesso em: 25 fev. 2021.

Figura 127

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 114.

Figura 128

MEMORIA_EPROM_2732_V1.JPG. Disponível em: https://bit.ly/3t6DCab. Acesso em: 26 fev. 2021.

Figura 129

COMPUTER-ACCESSORIES-1841254_960_720.JPG. Disponível em: https://bit.ly/3cmtPGa. Acesso em: 26 fev. 2021.

Figura 130

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 129.

Figura 131

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 130.

Figura 132

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 130.

Figura 133

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 96.

Figura 134

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 102.

Figura 135

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 104.

Figura 136

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 142.

Figura 137

800PX-RAMBUS-MEMORY.JPG. Adaptada. Disponível em: https://bit.ly/3rx9WCG. Acesso em: 1º mar. 2021.

Figura 138

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 143.

Figura 139

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 144.

Figura 140

EDWARDS, E.; LEUNG, K. Memory. *Introduction to computer architecture*. [s.d.]. Disponível em: https://bit.ly/3bweOCt. Acesso em: 1º mar. 2021.

Figura 141

THE FIRST Disk Drive: RAMAC 350. *Computer History Museum*. [s.d.]. Disponível em: https://bit.ly/3t4j9mr. Acesso em: 1º mar. 2021.

Figura 142

ALECRIM, E. O que é HD (disco rígido): características e como funciona. Tecnologia. *Infowester*, 23 ago. 2020. Disponível em: https://bit.ly/3qJkyxj. Acesso em: 2 mar. 2021.

Figura 143

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 318.

Figura 144

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 151.

Figura 145

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 319.

Figura 146

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 154.

Figura 147

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 156.

Figura 148

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. Adaptada.

Figura 149

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. Adaptada.

Figura 150

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. Adaptada.

Figura 151

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. Adaptada.

Figura 152

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. Adaptada.

Figura 153

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. Adaptada.

Figura 154

SD_IMGP5281_SMIAL_WP.JPG. Disponível em: https://bit.ly/2PUtpzp. Acesso em: 2 mar. 2021.

Figura 155

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 77.

Figura 156

DISCS-1344774_960_720.JPG. Disponível em: https://bit.ly/3qCVurJ. Acesso em: 3 mar. 2021.

Figura 157

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 79.

Figura 158

COMPARISON_CD_DVD_HDDVD_BD.SVG. Disponível em: https://bit.ly/3qGLxJv. Acesso em: 3 mar. 2021.

Figura 159

DISK-148609_960_720.PNG. Disponível em: https://bit.ly/30uNNJs. Acesso em: 3 mar. 2021.

Figura 160

TAPE-5723260_960_720.JPG. Disponível em: https://bit.ly/2NdHV4k. Acesso em: 3 mar. 2021.

Figura 161

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 319.

Figura 162

BOARD-564809_960_720.JPG. Disponível em: https://bit.ly/3l4NZIG. Acesso em: 3 mar. 2021.

Figura 163

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 148.

Figura 164

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 150.

Figura 165

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 39.

Figura 166

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 152.

Figura 167

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 154.

Figura 168

PICMG-BACKPLANE-DETAILS.JPG. Adaptada. Disponível em: https://bit.ly/3qGfiKC. Acesso em: 4 mar. 2021.

Figura 169

BUSES_PCI.JPG. Disponível em: https://bit.ly/30xVAWN. Acesso em: 4 mar. 2021.

Figura 170

AGP-BUS.JPG. Disponível em: https://bit.ly/3rDBy9n. Acesso em: 10 mar. 2021.

Figura 171

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 177.

Figura 172

UNIVERSAL-1823_960_720.JPG. Disponível em: https://bit.ly/3v98hpf. Acesso em: 4 mar. 2021.

Figura 173

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 345.

Figura 174

658PX-SERIAL_ATA_HARD_DISK_CONNECTED.JPG. Adaptada. Disponível em: https://bit.ly/3t7rx4y. Acesso em: 5 mar. 2021.

Figura 175

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 350.

Figura 176

FLAT-FLEX-CABLE.JPG. Disponível em: https://bit.ly/2OLILp3. Acesso em: 10 mar. 2021.

Figura 177

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 178.

Figura 178

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 342.

Figura 179

COMPUTER-1869236_960_720.JPG. Disponível em: https://bit.ly/38u3MMm. Acesso em: 5 mar. 2021.

Figura 180

MOUSE-74533_960_720.JPG. Disponível em: https://bit.ly/3qCAscv. Acesso em: 5 mar. 2021.

Figura 181

EPSON_DOT_MATRIX_PRINTER_DSC_0091.JPG. Disponível em: https://bit.ly/3qBxa9z. Acesso em: 5 mar. 2021.

Figura 182

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 362.

Figura 183

CANON_S520_INK_JET_PRINTER.JPG. Disponível em: https://bit.ly/2PUHw7R. Acesso em: 5 mar. 2021.

Figura 184

HP_LASERJET_1020_PRINTER.JPG. Disponível em: https://bit.ly/2ODkH7X. Acesso em: 5 mar. 2021.

Figura 185

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013. p. 96.

Figura 186

800PX-EIZO_FORIS_FG2421_VGA_COMPUTER_MONITOR_DISPLAYING_TEST_PATTERN.PNG. Disponível em: https://bit.ly/3I7CMar. Acesso em: 5 mar. 2021.

Figura 187

MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2019. p. 354.

Figura 188

IBM_RS6000_H70.JPG. Disponível em: https://bit.ly/3qw80sU. Acesso em: 5 mar. 2021.

Figura 189

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. Adaptada.

Figura 190

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. Adaptada.

Figura 191

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. Adaptada.

Figura 192

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. Adaptada.

Figura 193

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 493.

Figura 194

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 494.

Figura 195

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 495.

Figura 196

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 537.

Figura 197

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 561.

Figura 198

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010. p. 567.

REFERÊNCIAS

Textuais

ANIDO, R. *Linguagens de montagem*. 9. ed. Rio de Janeiro: Elsevier, 2016.

BEGGIORA, H. O que é memória *cache*? Entenda sua importância para o PC. *TechTudo*, 21 out. 2016. Disponível em: https://glo.bo/3qC9SAn. Acesso em: 25 fev. 2021.

CARTER, N. *Arquitetura de computadores*. Porto Alegre: Bookman, 2002.

CLEMENTS, A. *Computer organization and architecture*: therms and variations. Stamford: Cengage, 2014.

CRIADO disco óptico que armazena dados em cinco dimensões. *Inovação Tecnológica*, 22 maio 2009. Disponível em: https://bit.ly/3t7g6d6. Acesso em: 3 mar. 2021.

DELGADO, J.; RIBEIRO, C. *Arquitetura de computadores*. Rio de Janeiro: LTC, 2014.

DESCOBERTA vulnerabilidade nos processadores com *hyper-threading*. *Inovação Tecnológica*, 18 maio 2005. Disponível em: https://bit.ly/3vhZnWC. Acesso em: 5 mar. 2021.

FREIBERGER, P. A.; SWAINE, M. R. Von Neumann machine. *Encyclopedia Britannica*, 14 nov. 2016. Disponível em: https://bit.ly/3byu6GW. Acesso em: 3 fev. 2021.

GARRETT, F. O que é memória ROM? *TechTudo*, 14 out. 2015. Disponível em: https://glo.bo/3qH7YOJ. Acesso em: 26 fev. 2021.

IDOETA, I. V.; CAPUANO, F. G. *Elementos de eletrônica digital*. 40. ed. São Paulo: Érica, 2011.

JORDÃO, F. Tabela de processadores: Intel. *Tecmundo*, 2 jul. 2013. Disponível em: https://bit.ly/3t7kkS4. Acesso em: 3 fev. 2021.

MONTEIRO, M. A. *Introdução à organização de computadores*. 5. ed. Rio de Janeiro: LTC, 2019.

NANOFABRICAÇÃO: refazendo a matéria átomo por átomo. *Inovação Tecnológica*, 1º ago. 2016. Disponível em: https://bit.ly/3rCZbPl. Acesso em: 2 fev. 2021.

NOVO *hardware* acelera comunicação em processadores multicore. Inovação Tecnológica, 11 fev. 2011. Disponível em: https://bit.ly/3ciYjJm. Acesso em: 5 mar. 2021.

NOVO processador tem 100 milhões de transistores. *Inovação Tecnológica*, 4 dez. 2002. Disponível em: https://bit.ly/2OqqMVr. Acesso em: 1º fev. 2021.

NULL, L.; LOBUR, J. *Princípios básicos de arquitetura e organização de computadores*. Porto Alegre: Bookman, 2010.

PANNAIN, R.; BEHRENS, F. H; PIVA JR., D. *Organização básica de computadores e linguagem de montagem*. São Paulo: Elsevier, 2012.

PATTERSON, D. A.; GIBSON, G. A.; KATZ, R. H. A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD Record*, jun. 1988.

PROCESSAMENTO paralelo chega aos computadores de mesa. *Inovação Tecnológica*, 27 jun. 2017. Disponível em: https://bit.ly/3qDoC23. Acesso em: 7 fev. 2021.

ROTMAN, D. We're not prepared for the end of Moore's Law. *MIT Technology Review*, 24 fev. 2020. Disponível em: https://bit.ly/38t8IpY. Acesso em: 3 fev. 2020.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Fundamentos de sistemas operacionais*. 9. ed. Rio de Janeiro: LTC, 2015.

SODERSTROM, T. How to choose the right memory: a 2020 guide to DRAM. *Tom's Hardware*, 5 jun. 2020. Disponível em: https://bit.ly/3bE9fSK. Acesso em: 1º mar. 2021.

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo: Pearson, 2010.

TANENBAUM, A. S.; AUSTIN, T. *Organização estruturada de computadores*. São Paulo: Pearson, 2013.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*: princípios e aplicações. São Paulo: Pearson, 2011.

ULTRADOWNLOADS. Como funcionam os monitores LCDs?. *Canaltech*, 3 abr. 2012. Disponível em: https://bit.ly/3taLcAw. Acesso em: 11 mar. 2021.

ULTRAMAGNETRON: gravação magnética à velocidade da luz. *Inovação Tecnológica*, 18 jun. 2014. Disponível em: https://bit.ly/3v8jcPQ. Acesso em: 3 mar. 2021.

VIANA, A. L. S. Um pouco sobre RAID. Configuração via *software* e via *hardware*. *DevMedia*, 2012. Disponível em: https://bit.ly/3rF35Hl. Acesso em: 2 mar. 2021.

Interativa

Informações:
www.sepi.unip.br ou 0800 010 9000