Organização de Computadores Digitais - 5954008

1. Informação Digital

Prof. Luiz Otavio Murta Jr

Local: Depto. de Computação e Matemática (FFCLRP/USP)

Principais Tópicos

2. Revisão de Lógica Digital

- 2.1. Álgebra Booleana
- 2.2. Portas Lógicas
- 2.3. Circuitos Combinatórios
- 2.4. Circuitos Sequenciais

2.3.4. Cache

- Historicamente, as CPUs sempre foram mais rápidas do que as memórias.
- Conforme memórias melhoraram as CPUs também se aperfeiçoaram, mantendo o desequilíbrio.
- Na verdade, à medida que fica possível colocar cada vez mais circuitos em um chip, os projetistas estão usando essas novas facilidades no paralelismo (*pipelining*) e em operação superescalar, fazendo as CPUs ficarem ainda mais velozes.
- Projetistas de memória costumam usar nova tecnologia para aumentar a capacidade de seus chips, e não a velocidade, portanto, parece que os problemas estão piorando com o passar do tempo.

2.3.4. Cache

- Na prática, o significado desse desequilíbrio é que, após emitir uma requisição de memória, a CPU não obterá a palavra de que necessita por muitos ciclos de CPU.
- Quanto mais lenta a memória, mais ciclos a CPU terá de esperar.
- Como já destacamos, há duas maneiras de tratar desse problema.
- O modo mais simples é somente iniciar READs (leituras) de memória quando elas forem encontradas, mas continuar executando e bloquear a CPU se uma instrução tentar usar a palavra de memória antes de ela chegar.
- Quanto mais lenta a memória, maior será a frequência desse problema e maior será a penalidade quando isso, de fato, ocorrer.

2.3.4. Cache

- Por exemplo, se uma instrução em cinco toca na memória e o tempo de acesso à memória for de cinco ciclos, o tempo de execução será o dobro daquele que teria sido na memória instantânea.
- Mas, se o tempo de acesso for de 50 ciclos, então o tempo de execução será elevado por um fator de 11 (5 ciclos para executar instruções mais 50 ciclos para esperar pela memória).
- A outra solução é ter máquinas que não ficam bloqueadas, mas, em vez disso, exigem que o compilador não gere código para usar palavras antes que elas tenham chegado.
- O problema é que é muito mais fácil falar dessa abordagem do que executá-la.

2.3.3. Cache

- Muitas vezes, não há nada mais a fazer após um LOAD (carregar), portanto, o compilador é forçado a inserir instruções NOP (nenhuma operação), que nada mais fazem do que ocupar um intervalo (*slot*) e gastar tempo.
- Com efeito, essa abordagem é um bloqueio de software em vez de um bloqueio de hardware, mas a degradação do desempenho é a mesma.
- Na verdade, o problema não é tecnológico, mas econômico.
- Os engenheiros sabem como construir memórias tão rápidas quanto as CPUs, mas para que executem a toda velocidade, elas têm de estar localizadas no chip da CPU
  - (porque passar pelo barramento para alcançar a memória é uma operação muito lenta).

2.3.4. Cache

- Instalar uma memória grande no chip da CPU faz com que esta fique maior e, portanto, mais cara. Ainda que o custo não fosse uma questão a considerar, há limites de tamanho para um chip de CPU.
- Assim, a opção se resume a ter uma pequena quantidade de memória rápida ou uma grande quantidade de memória lenta.
- O que nós gostaríamos de ter é uma grande quantidade de memória rápida a um preço baixo.
- O interessante é que há técnicas conhecidas para combinar uma pequena quantidade de memória rápida com uma grande quantidade de memória lenta para obter (quase) a velocidade da memória rápida e a capacidade da memória grande a um preço módico.

2.3.4. Cache

- A memória pequena e rápida é denominada cache (do francês cacher, que significa “esconder” e se pronuncia “késh”).
- Em seguida, descreveremos brevemente como as caches são usadas e como funcionam.
- A ideia básica de uma cache é simples: as palavras de memória usadas com mais frequência são mantidas na cache.
- Quando a CPU precisa de uma palavra, ela examina em primeiro lugar a cache.
- Somente se a palavra não estiver ali é que ela recorre à memória principal.
- Se uma fração substancial das palavras estiver na cache, o tempo médio de acesso pode ser muito reduzido.

2.3.4. Cache

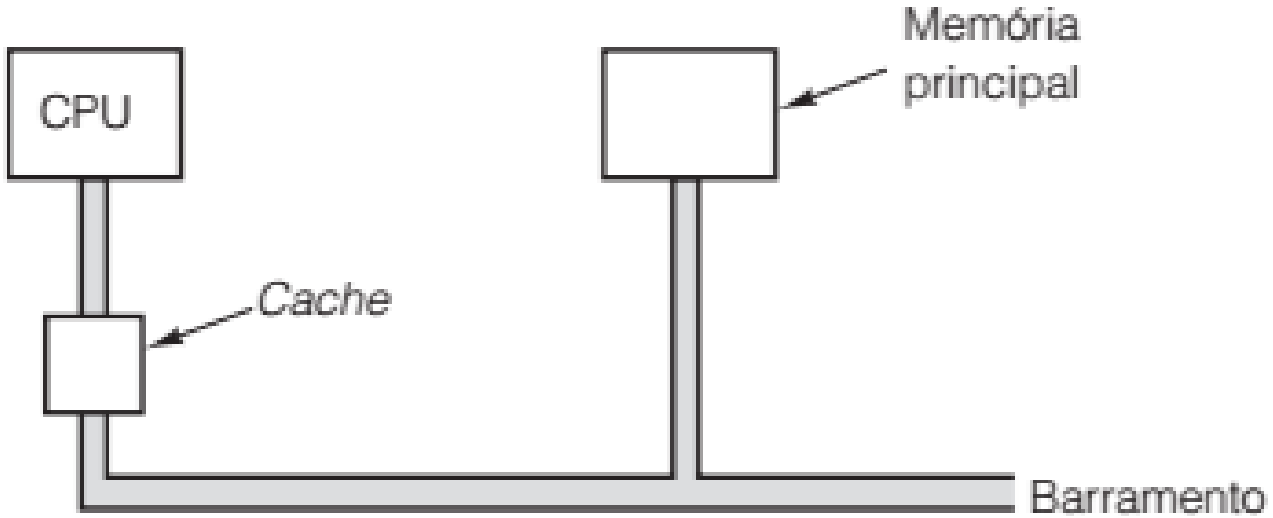
- Assim, o sucesso ou o fracasso depende da fração das palavras que estão na cache.
- Há anos todos sabemos que programas não acessam suas memórias de forma totalmente aleatória.
- Se uma dada referência à memória for para o endereço A, é provável que a próxima estará na vizinhança geral de A. Um exemplo simples é o próprio programa.
- Exceto quando se trata de desvios e de chamadas de procedimento, as instruções são buscadas em localizações consecutivas da memória.
- Além do mais, grande parte do tempo de execução de um programa é gasto em laços, nos quais um número limitado de instruções é executado repetidas vezes.

2.3.4. Cache

- De modo semelhante, é provável que um programa de manipulação de matrizes fará muitas referências à mesma matriz antes de passar para outra coisa qualquer.
- A observação de que referências à memória feitas em qualquer intervalo de tempo curto tendem a usar apenas uma pequena fração da memória total é denominada princípio da localidade, e forma a base de todos os sistemas de cache.
- A ideia geral é que, quando uma palavra for referenciada, ela e algumas de suas vizinhas sejam trazidas da memória grande e lenta para a cache, de modo que, na próxima vez em que for usada, ela possa ser acessada rapidamente.
- Um arranjo comum da CPU, cache e memória principal é ilustrado na próxima figura.

2.3.4. Cache

- Se uma palavra for lida ou escrita *k* vezes em um curto intervalo de tempo, o computador precisará de 1 referência à memória lenta e *k* – 1 referências à memória rápida. Quanto maior for *k*, melhor será o desempenho global.
- **A localização lógica da cache é entre a CPU e a memória principal. Em termos físicos, há diversos lugares em que ela poderia estar localizada.**



2.3.4. Cache

- Podemos formalizar esse cálculo introduzindo
  - *c*, o tempo de acesso à cache;
  - *m*, o tempo de acesso à memória principal;
  - e *h*, a taxa de acerto, que é a fração de todas as referências que podem ser satisfeitas através da cache.
- Em nosso pequeno exemplo do parágrafo anterior,
  - $h = (k - 1)/k$.
- Alguns autores também definem a taxa de falha (na cache), que é
  - $1 - h$.
- Com essas definições, podemos calcular o tempo de acesso médio como segue:
  - tempo de acesso médio = $c + (1 - h)\, m$

2.3.4. Cache

- À medida que $h \rightarrow 1$, todas as referências podem ser satisfeitas fora da cache e o tempo de acesso médio se aproxima de *c*.
- Por outro lado, à medida que $h \rightarrow 0$, toda vez será necessária uma referência à memória,
  - portanto, o tempo de acesso se aproxima de $c + m$, primeiro um tempo para verificar a cache (sem sucesso) e então um tempo *m* para fazer a referência à memória.
- Em alguns sistemas, a referência à memória pode ser iniciada em paralelo com a busca na cache,
  - de modo que, se ocorrer uma falha na cache (*cache miss*), o ciclo da memória já terá sido iniciado.
- Contudo, essa estratégia requer que a memória possa ser interrompida se houver uma presença na cache (*cache hit*), o que torna a implantação mais complicada.

2.3.4. Cache

- Usando o princípio da localidade como guia, memórias principais e caches são divididas em blocos de tamanho fixo.
- Ao nos referirmos a esses blocos dentro da cache, eles costumam ser chamados de linhas de cache.
- Quando a busca na cache falha, toda a linha de cache é carregada da memória principal para a cache, e não apenas a palavra que se quer.
  - Por exemplo, com uma linha de cache de 64 bytes de tamanho, uma referência ao endereço de memória 260 puxará a linha que consiste nos bytes 256 a 319 para uma linha de cache.
- Com um pouco de sorte, algumas das outras palavras na linha de cache também serão necessárias em breve.

2.3.4. Cache

- Esse tipo de operação é mais eficiente do que buscar palavras individuais porque é mais rápido buscar *k* palavras de uma vez só do que uma palavra *k* vezes.
- Além disso, ter entradas de cache de mais do que uma palavra significa que há menor número delas; por conseguinte, é preciso menos memória auxiliar (overhead).
- Por fim, muitos computadores podem transferir 64 ou 128 bits em paralelo em um único ciclo do barramento, até mesmo em máquinas de 32 bits.

2.3.4. Cache

- O projeto de cache é uma questão de importância cada vez maior para CPUs de alto desempenho.
- Um aspecto é o tamanho da cache.
  - Quanto maior, melhor seu funcionamento, mas também maior é o custo.
- Um segundo aspecto é o tamanho da linha de cache.
  - Uma cache de 16 KB pode ser dividida em até 1.024 linhas de 16 bytes, 2.048 linhas de 8 bytes e outras combinações.
- Um terceiro aspecto é a maneira de organização, isto é, como ela controla quais palavras de memória estão sendo mantidas no momento.
- Um quarto aspecto do projeto é se as instruções e dados são mantidos na mesma cache ou em caches diferentes.

2.3.4. Cache

- Ter uma cache unificada (instruções e dados usam a mesma cache) é um projeto mais simples e mantém automaticamente o equilíbrio entre buscas de instruções e buscas de dados.
  - No entanto, a tendência hoje é uma cache dividida, com instruções em uma cache e dados na outra.
- Esse projeto também é denominado arquitetura Harvard
  - essa referência volta ao passado até o computador Mark III de Howard Aiken, que tinha memórias diferentes para instruções e dados.
- A força que impele os projetistas nessa direção é a utilização muito difundida de CPUs com paralelismo (*pipelined*).

2.3.4. Cache

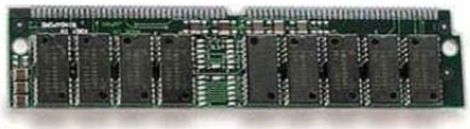
- A unidade de busca de instrução precisa acessar instruções ao mesmo tempo em que a unidade de busca de operandos precisa de acesso aos dados.
- Uma cache dividida permite acessos paralelos; uma cache unificada, não.
- Além disso, como as instruções não são modificadas durante a execução, o conteúdo da cache de instrução nunca tem de ser escrito de volta na memória.
- Por fim, um quinto aspecto é o número de caches.
  - Hoje em dia não é incomum ter chips com uma cache primária no chip, uma cache secundária fora dele, mas no mesmo pacote do chip da CPU, e uma terceira cache ainda mais distante.

2.3.5. tipos de Memórias

- Desde os primeiros dias da memória de semicondutor até o início da década 1990, a memória era fabricada, comprada e instalada como chips únicos.
- As densidades dos chips iam de 1 K bits até 1 M bits e além, mas cada chip era vendido como uma unidade separada.
- Os primeiros PCs costumavam ter soquetes vazios nos quais podiam ser ligados chips de memória adicionais, se e quando o comprador precisasse deles.
- Desde o início da década de 1990, usa-se um arranjo diferente.
- Um grupo de chips, em geral 8 ou 16, é montado em uma minúscula placa de circuito impresso e vendido como uma unidade.

2.3.5. tipos de Memórias

- Essa unidade é denominada:
  - SIMM (Single Inline Memory Module – módulo único de memória em linha),
  - ou DIMM (Dual Inline Memory Module – módulo duplo de memória em linha), dependendo se tem uma fileira de conectores de um só lado ou de ambos os lados da placa.
- Os SIMMs têm um conector de borda com 72 contatos e transferem 32 bits por ciclo de clock.
- Os DIMMs em geral têm conectores de borda com 120 contatos em cada lado da placa, perfazendo um total de 240 contatos e transferem 64 bits por ciclo de clock.
- Os mais comuns hoje são os DIMMs DDR3, que é a terceira versão das memórias de taxa dupla. Um exemplo típico de DIMM é ilustrado na figura a seguir.

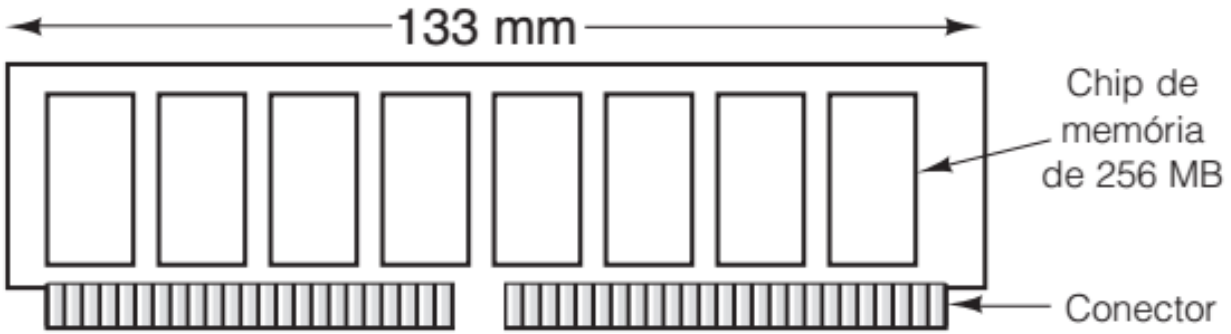


SIMM – 72 vias

DIMM – 168 vias

2.3.5. tipos de Memórias

- Visão superior de um DIMM de 4 GB, com oito chips de 256 MB em cada lado. O outro lado tem a mesma aparência.

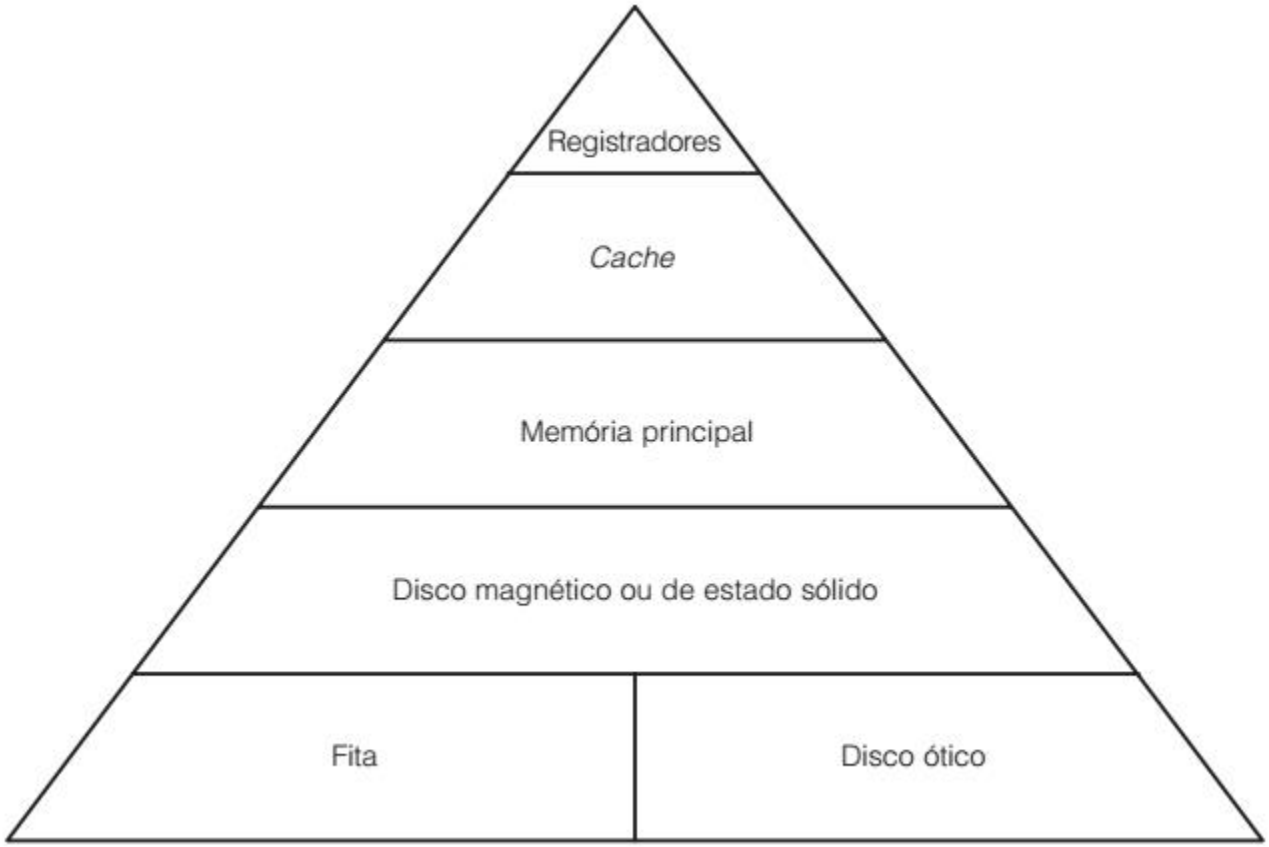


2.3.5. tipos de Memórias

- Uma configuração típica de DIMM poderia ter oito chips de dados com 256 MB cada.
- Então, o módulo inteiro conteria 2 GB.
- Muitos computadores têm espaço para quatro módulos, o que dá uma capacidade total de 8 GB se usarem módulos de 2 GB e mais, se usarem módulos maiores.
- Um DIMM fisicamente menor, denominado SO-DIMM (*Small Outline* DIMM – DIMM pequeno perfil) é usado em notebooks.
- Pode-se adicionar um bit de paridade ou correção de erro aos DIMMS, porém, visto que a taxa média de erro de um módulo é de um erro a cada dez anos, na maioria dos computadores de uso comum e doméstico, detecção e correção de erros são omitidas.

22.3.5. tipos de Memórias

- Hierarquia de memória de cinco níveis.



2.3.5. tipos de Memórias

- Os discos dos modernos computadores pessoais evoluíram daquele usado no IBM PC XT,
  - que era um disco Seagate de 10 MB controlado por um controlador de disco Xebec em um cartão de encaixe (plug-in).
- O disco Seagate tinha 4 cabeçotes, 306 cilindros e 17 setores por trilha.
  - O controlador conseguia manipular dois drives.
- O sistema operacional lia e escrevia em um disco colocando parâmetros em registradores da CPU e então chamando o BIOS (Basic Input Output System – sistema básico de entrada e saída) localizado na memória somente de leitura do PC.
- O BIOS emitia as instruções de máquina para carregar os registradores do controlador de disco que iniciava as transferências.

2.3.5. tipos de Memórias

- A tecnologia evoluiu rapidamente e passou do controlador em uma placa separada para o controlador integrado com os drives,
  - começando com drives IDE (Integrated Drive Electronics – eletrônica integrada ao drive) em meados da década de 1980.
- Contudo, as convenções de chamada do BIOS não foram alteradas por causa da compatibilidade.
- Essas convenções de chamada endereçavam setores dando seus números de cabeçote, cilindro e setor, sendo que a numeração de cabeçotes e cilindros começava em 0, e de setores, em 1.
- Essa escolha provavelmente se deveu a um erro da parte do programador original do BIOS, que escreveu sua obra-prima em assembler 8088.

2.3.5. tipos de Memórias

- Com 4 bits para o cabeçote, 6 bits para o setor e 10 bits para o cilindro, o drive máximo podia ter 16 cabeçotes, 63 setores e 1.024 cilindros, para um total de 1.032.192 setores.
- Esse drive máximo tinha uma capacidade de 504 MB, o que devia parecer uma infinidade naquela época, porém, agora, decerto não.
  - (Hoje você criticaria uma nova máquina que não pudesse manipular drives maiores do que 1.000 TB?)
- Infelizmente, não passou muito tempo e apareceram drives acima de 504 MB, mas com a geometria errada
  - (por exemplo, 4 cabeçotes, 32 setores e 2.000 cilindros totalizam 256.000 setores).
- O sistema operacional não conseguia endereçá-los de modo algum, por causa das convenções de chamada do BIOS há muito cristalizadas.

2.3.5. tipos de Memórias

- O resultado é que os controladores de disco começaram a mentir, fingindo que a geometria estava dentro dos limites do BIOS embora, na verdade, estivesse remapeando a geometria virtual para a geometria real.
- Embora essa técnica funcionasse, causava grandes estragos nos sistemas operacionais que posicionavam dados cuidadosamente para minimizar tempos de busca.
- Com o tempo, os drives IDE evoluíram para drives EIDE
  - (*Extended* IDE – IDE estendido),
- que também suportavam um segundo esquema de endereçamento denominado LBA
  - (*Logical Block Addressing* – endereçamento de blocos lógicos),
- que numera os setores começando em 0 até um máximo de $2^{28} - 1$.

E-IDE

2.3.5. tipos de Memórias

- Esse esquema requer que o controlador converta endereços LBA para endereços de cabeçote:
  - setor e cilindro, mas ultrapassa o limite de 504 MB.
- Infelizmente, ele criava um novo gargalo a $2^{28} \times 2^{9}$ bytes (128 GB).
- Em 1994, quando foi adotado o padrão EIDE, ninguém poderia imaginar discos de 128 GB.
- Comitês de padronização, assim como os políticos, têm tendência de empurrar problemas para que o próximo comitê os resolva.
- Drives e controladores EIDE também tinham outras melhorias.
- Por exemplo, controladores EIDE podiam ter dois canais, cada um com um drive primário e um secundário.

2.3.5. tipos de Memórias

- Esse arranjo permitia um máximo de quatro drives por controlador.
- Drives de CD-ROM e DVD também eram suportados, e a taxa de transferência aumentou de 4 MB/s para 16,67 MB/s.
- Enquanto a tecnologia de disco continuava a melhorar, o padrão EIDE continuava a evoluir,
  - mas, por alguma razão, o sucessor do EIDE foi denominado ATA-3 (AT *Attachment*), uma referência ao IBM PC/AT (onde AT se referia à então "tecnologia avançada" – *Advanced Technology* – de uma CPU de 16 bits executando em 8 MHz).
- Na edição seguinte, o padrão recebeu o nome de ATAPI-4 (ATA *Packet Interface* – interface de pacotes ATA) e a velocidade aumentou para 33 MB/s.
- Com o ATAPI-5, ela alcançou 66 MB/s.

2.3.5. tipos de Memórias

- Nessa época, o limite de 128 GB imposto pelos endereços LBA de 28 bits estava ficando cada vez mais ameaçador, portanto, o ATAPI-6 alterou o tamanho do LBA para 48 bits.
- O novo padrão entrará em dificuldade quando os discos chegarem a $2^{48} \times 2^{9}$ bytes (128 PB).
- Com um aumento de capacidade de 50% ao ano, o limite de 48 bits deverá durar até mais ou menos 2035.
- A melhor aposta é que o tamanho do LBA alcance 64 bits.
- O padrão ATAPI-6 também aumentou a taxa de transferência para 100 MB/s e atacou a questão do ruído do disco pela primeira vez.
- O padrão ATAPI-7 é uma ruptura radical com o passado.

2.3.5. tipos de Memórias

- Em vez de aumentar o tamanho do conector do drive (para aumentar a taxa de dados),
  - esse padrão usa o que é chamado ATA serial para transferir 1 bit por vez por um conector de 7 pinos a velocidades que começam em 150 MB/s e que, com o tempo, espera-se que alcancem 1,5 GB/s.
- Substituir o atual cabo plano de 80 fios por um cabo redondo com apenas alguns milímetros a mais de espessura melhora o fluxo de ar dentro do computador.
- Além disso, o ATA serial usa 0,5 volt para sinalização (em comparação com os 5 volts dos *drivers* ATAPI-6), o que reduz o consumo de energia.
- É provável que, hoje, todos os computadores novos usam interface ATA serial (SATA).

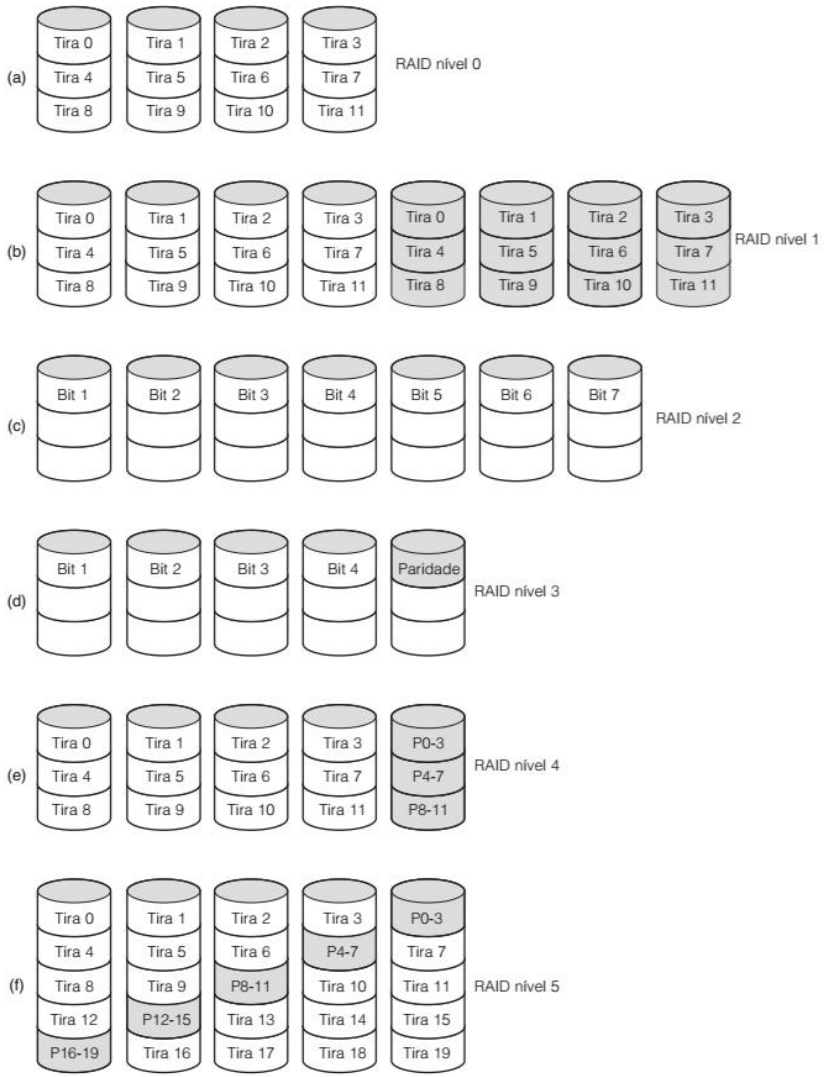
Cabo SATA

2.3.5. tipos de Memórias

- O desempenho da CPU vem tendo aumento exponencial na última década e dobra a cada 18 meses mais ou menos.
- O mesmo não acontece com o desempenho do disco. Na década de 1970, os tempos médios de busca em discos de minicomputadores eram de 50 a 100 ms.
  - Agora, são de 10 ms.
- Na maioria das indústrias técnicas
  - (por exemplo, automóveis ou aviação),
- um fator de 5 a 10 de melhoria de desempenho em duas décadas seria uma grande notícia, mas na indústria de computadores isso é constrangedor.
- Assim, a lacuna entre o desempenho da CPU e o do disco ficou cada vez maior com o passar do tempo.

2.3.5. tipos de Memórias

RAIDs níveis 0 a 5.
Os drives de backup e,
paridade estão sombreados.



2.3.5. tipos de Memórias

- Como vimos, muitas vezes é usado processamento paralelo para acelerar o desempenho da CPU.
- Ao longo dos anos, ocorreu a várias pessoas que a E/S paralela também poderia ser uma boa ideia.
- Em seu artigo de 1988, Patterson et al. sugeriram seis organizações específicas de disco que poderiam ser usadas para melhorar o desempenho, a confiabilidade do disco, ou ambos (Patterson et al., 1988).
- Essas ideias logo foram adotadas pela indústria e deram origem a uma nova classe de dispositivos de E/S, denominados RAID.
- Patterson et al. definiram RAID como *Redundant Array of Inexpensive Disks* (arranjo redundante de discos baratos), mas a indústria redefiniu o I como “independente” em vez de barato (*inexpensive*) – talvez para que pudessem usar discos caros?

2.3.5. tipos de Memórias

- Já que também era preciso ter um vilão
  - (como no caso RISC versus CISC, também devido a Patterson),
  - nesse caso o bandido era o SLED (*Single Large Expensive Disk* – disco único grande e caro).
- A ideia fundamental de um RAID é instalar uma caixa cheia de discos próxima ao computador,
  - em geral um grande servidor, substituir a placa do controlador de disco por um controlador RAID,
  - copiar os dados para o RAID e então continuar a execução normal.
- Em outras palavras, um RAID deveria parecer um SLED para o sistema operacional, mas ter melhor desempenho e melhor confiabilidade.

2.3.5. tipos de Memórias

- Uma vez que discos SCSI têm bom desempenho, baixo preço e a capacidade de ter até 7 drives em um único controlador (15 para o wide SCSI),
  - é natural que a maioria dos RAIDs consista em um controlador RAID SCSI mais uma caixa de discos SCSI que parecem para o sistema operacional como um único disco grande.
- Portanto, não é preciso alterar software para usar o RAID, um ótimo argumento de venda para muitos administradores de sistemas.
- Além de parecerem um disco único para o *software*,
  - há uma propriedade comum a todos os RAIDs,
  - que é a distribuição dos dados pelos drives para permitir operação paralela.

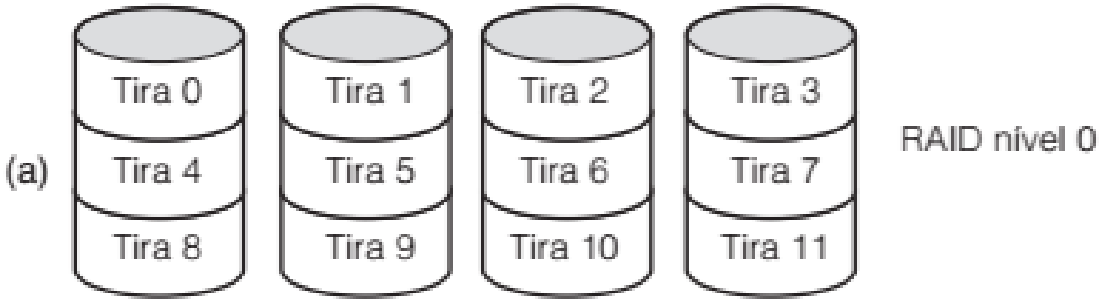
2.3.5. tipos de Memórias

- Patterson et al. definiram vários esquemas diferentes para fazer isso e, agora, eles são conhecidos como RAID nível 0 até RAID nível 5.
- Além disso, há alguns outros níveis menos importantes que não discutiremos.
- O termo “nível” é, de certa maneira, uma denominação imprópria,
  - não há nenhuma hierarquia envolvida;
  - há seis diferentes organizações possíveis,
  - cada qual com uma mistura diferente de características de confiabilidade e desempenho.
- O RAID nível 0 é ilustrado na figura consiste em ver o disco virtual simulado pelo RAID como se fosse dividido em tiras de k setores cada: os setores 0 a k – 1 são a tira 0, os setores k a 2k – 1 são a tira 1 e assim por diante.

2.3.5. tipos de Memórias

RAIDs nível 0.

Os drives de backup e,
paridade estão sombreados.



2.3.5. tipos de Memórias

- Para k = 1, cada tira é um setor; para k = 2, uma tira são dois setores etc.
- A organização RAID nível 0 escreve tiras consecutivas nos drives por alternância circular, como demonstrado na figura para um RAID com quatro drives de disco.
- Essa distribuição de dados por múltiplos drives é denominada *striping* (ou segmentação).
  - Por exemplo, se o *software* emitir um comando para ler um bloco de dados que consiste em quatro tiras consecutivas e começa na borda da tira, o controlador RAID o subdividirá em quatro comandos separados, um para cada disco, e fará com que eles funcionem em paralelo.
- Assim, temos E/S paralela sem que o software saiba disso.

2.3.5. tipos de Memórias

- O RAID nível 0 funciona melhor com requisições grandes; quanto maiores, melhor.
- Se uma requisição for maior do que o número de drives vezes o tamanho da tira, alguns drives receberão múltiplas requisições, de modo que, quando terminam a primeira, iniciam a segunda.
  - Cabe ao controlador dividir a requisição e alimentar os comandos adequados aos discos adequados na sequência certa e então agrupar os resultados na memória corretamente.
- O desempenho é excelente e a execução é direta.
- O RAID nível 0 funciona pior com sistemas operacionais que costumam requisitar dados a um setor por vez.
- Os resultados serão corretos, mas não há paralelismo e, por conseguinte, nenhum ganho de desempenho.

2.3.5. tipos de Memórias

- Outra desvantagem dessa organização é que a confiabilidade é potencialmente pior do que ter um SLED.
- Se um RAID consistir em quatro discos, cada um com um tempo médio de falha de 20 mil horas,
  - mais ou menos uma vez a cada 5 mil horas um drive falhará e haverá perda total de dados.
- Um SLED com um tempo médio de falha de 20 mil horas seria quatro vezes mais confiável.
- Como não há nenhuma redundância presente nesse projeto, na realidade ele não é um RAID verdadeiro.
- A próxima opção, RAID nível 1, mostrada na figura, é um RAID verdadeiro.
- Ele duplica todos os discos, portanto, há quatro discos primários e quatro de backup.

2.3.5. tipos de Memórias

RAIDs nível 1.

Os drives de backup e,
paridade estão sombreados.

2.3.5. tipos de Memórias

- Para uma escrita, cada tira é escrita duas vezes.
- Para uma leitura, qualquer das duas cópias pode ser usada, distribuindo a carga por mais drives.
- Por conseguinte, o desempenho da escrita não é melhor do que o de um único drive, mas o de leitura pode ser duas vezes melhor.
- A tolerância a falhas é excelente: se um drive falhar, basta usar a outra cópia em seu lugar.
- A recuperação consiste na simples instalação de um novo drive e em copiar todo o drive de backup para ele.
- Ao contrário dos níveis 0 e 1, que trabalham com tiras de setores, o RAID nível 2 trabalha por palavra, possivelmente até por byte.

2.3.5. tipos de Memórias

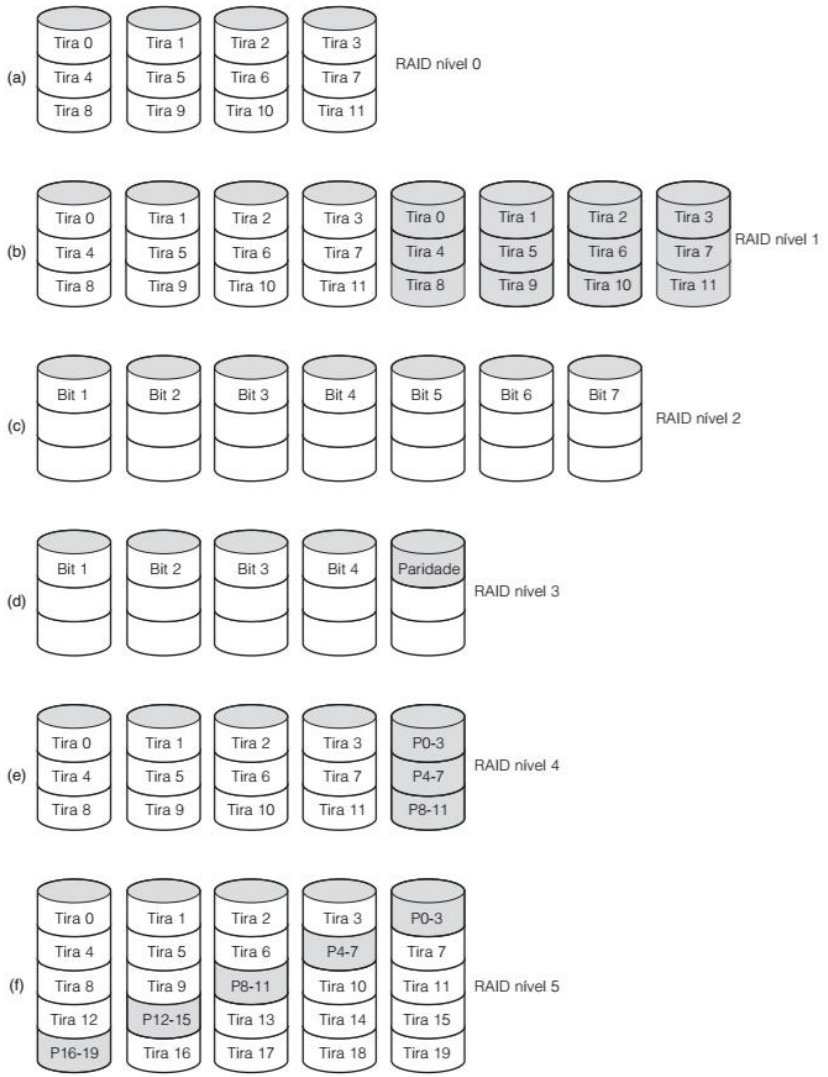
RAIDs nível 2.
Os drives de backup e,
paridade estão sombreados.

2.3.5. tipos de Memórias

- Imagine dividir cada byte do disco virtual único em um par de nibbles de 4 bits e então acrescentar um código de Hamming a cada um para formar uma palavra de 7 bits, dos quais os bits 1, 2 e 4 fossem de paridade.
- Imagine ainda que a posição do braço e a posição rotacional dos sete drives da figura fossem sincronizadas.
- Então, seria possível escrever a palavra de 7 bits codificada por Hamming nos sete drives, um bit por drive.

2.3.5. tipos de Memórias

RAIDs níveis 0 a 5.
Os drives de backup e,
paridade estão sombreados.

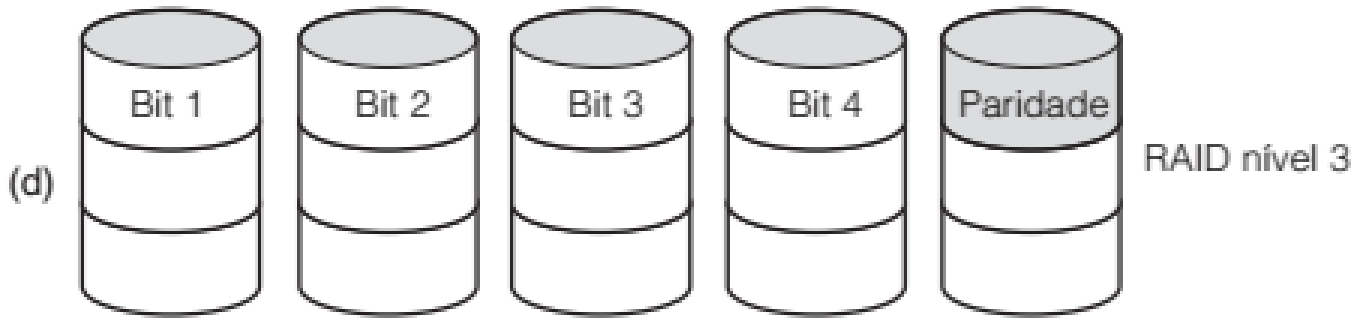


2.3.5. tipos de Memórias

- O RAID nível 3, ilustrado na figura, é uma versão simplificada do RAID nível 2.
- Nesse arranjo, um único bit de paridade é computado para cada palavra de dados e escrito em um drive de paridade.
- Como no RAID nível 2, os drives devem estar em exata sincronia, uma vez que palavras de dados individuais estão distribuídas por múltiplos drives.
- RAIDs níveis 4 e 5 de novo trabalham com tiras, e não com palavras individuais com paridade, e não requerem drives sincronizados.

2.3.5. tipos de Memórias

RAIDs nível 3.
Os drives de backup e,
paridade estão sombreados.



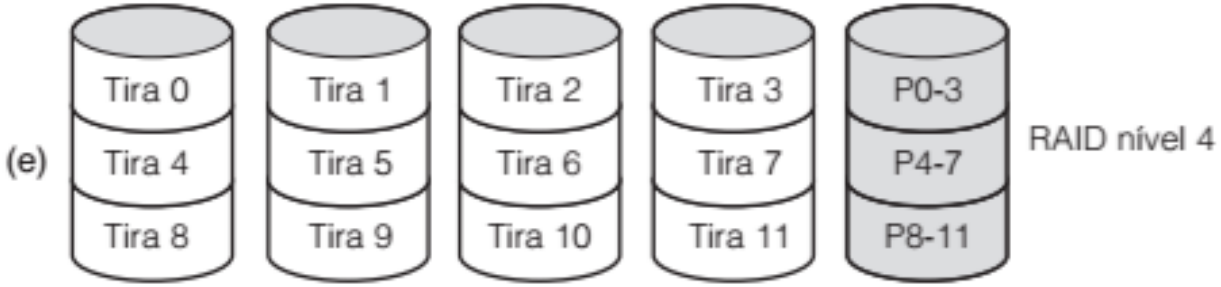
2.3.5. tipos de Memórias

- O RAID nível 4 [veja a figura] é como o RAID nível 0, com paridade tira por tira escrita em um drive extra.
- Por exemplo, se cada tira tiver k bytes de comprimento, todas as tiras passam por uma operação de EXCLUSIVE OR, resultando em uma tira de paridade de k bytes de comprimento.
- Se um drive falhar, os bytes perdidos podem ser recalculados com base no drive de paridade.

2.3.5. tipos de Memórias

RAIDs nível 4.

Os drives de backup e,
paridade estão sombreados.



2.3.5. tipos de Memórias

- O RAID nível 4 [veja a figura] é como o RAID nível 0, com paridade tira por tira escrita em um drive extra.
- Por exemplo, se cada tira tiver k bytes de comprimento, todas as tiras passam por uma operação de EXCLUSIVE OR, resultando em uma tira de paridade de k bytes de comprimento.
- Se um drive falhar, os bytes perdidos podem ser recalculados com base no drive de paridade.

2.3.5. tipos de Memórias

- Esse projeto protege contra a perda de um drive, mas seu desempenho é medíocre para pequenas atualizações.
- Se um setor for alterado, é necessário ler todos os drives para recalcular a paridade que, então, precisará ser reescrita.
- Como alternativa, ele pode ler os velhos dados de usuário e os velhos dados de paridade e recalcular nova paridade, e partir deles.
- Mesmo com essa otimização, uma pequena atualização requer duas leituras e duas escritas, o que é, claramente, um mau arranjo.

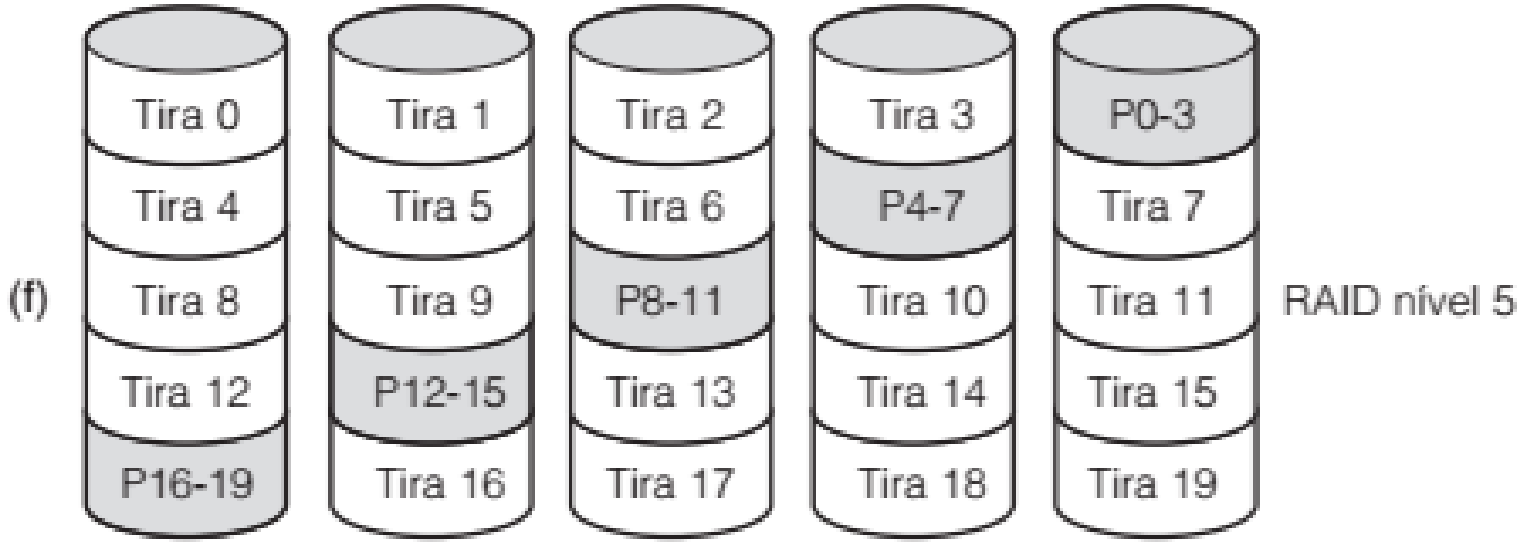
2.3.5. tipos de Memórias

- Como consequência da carga pesada sobre o drive de paridade, ele pode se tornar um gargalo.
- Esse gargalo é eliminado no RAID nível 5 distribuindo os bits de paridade uniformemente por todos os drives, por alternância circular, conforme mostra a figura.
- Contudo, no evento de uma falha de drive, a reconstrução do drive danificado é um processo complexo.

2.3.5. tipos de Memórias

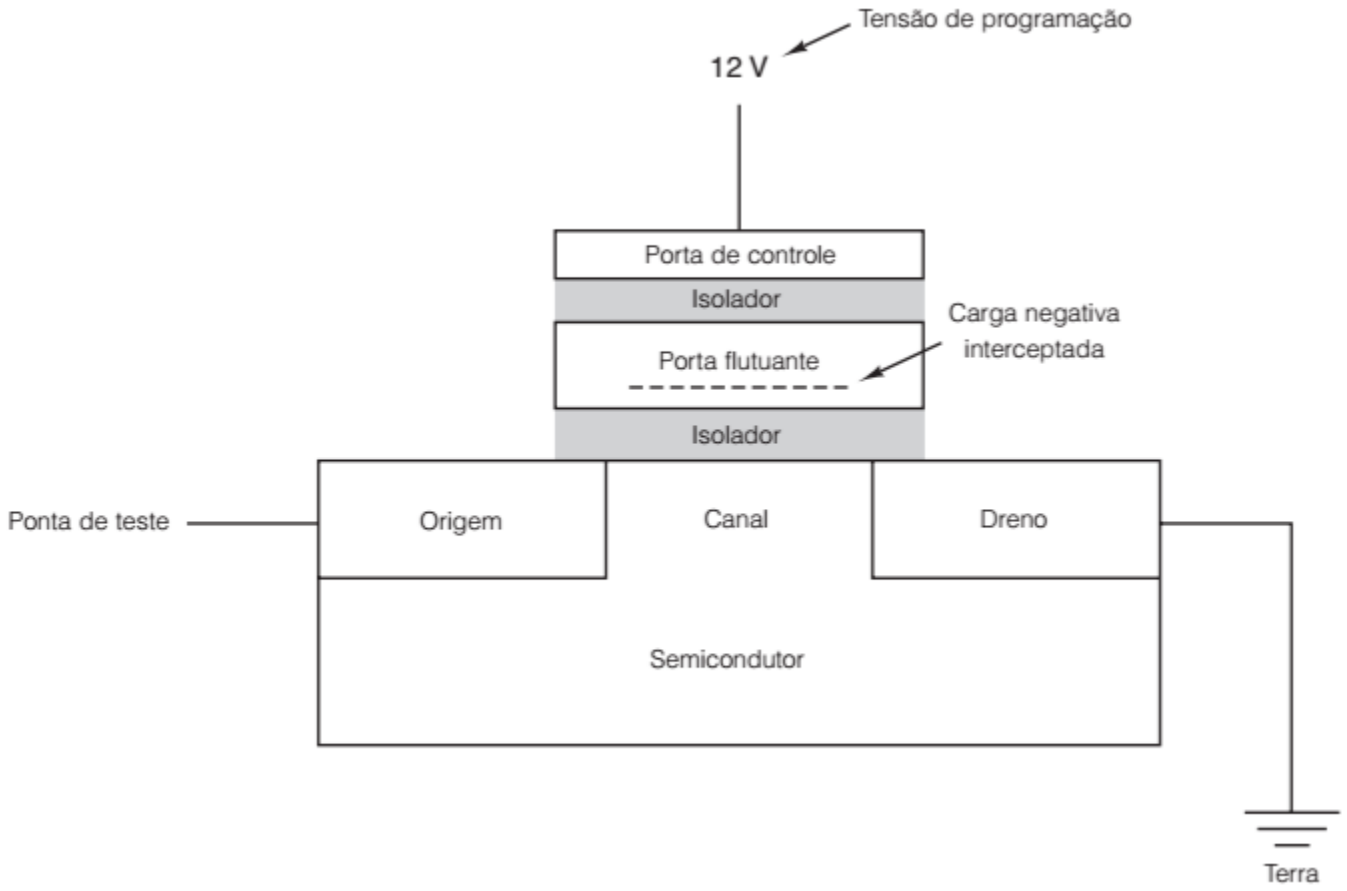
RAIDs nível 5.

Os drives de backup e,
paridade estão sombreados.



2.3.5. tipos de Memórias

- Uma célula de memória flash.



2.3.5. tipos de Memórias

- Os discos flash são compostos de muitas células de memória flash em estado sólido.
- As células da memória flash são feitas de um único transistor flash especial.
- Uma célula de memória flash aparece na figura.
- Embutido no transistor há uma porta flutuante que pode ser carregada e descarregada usando altas voltagens.
- Antes de ser programada, a porta flutuante não afeta a operação do transistor, atuando como um isolador extra entre a porta de controle e o canal do transistor.
- Se a célula flash for testada, ela atuará como um transistor simples.

2.3.5. tipos de Memórias

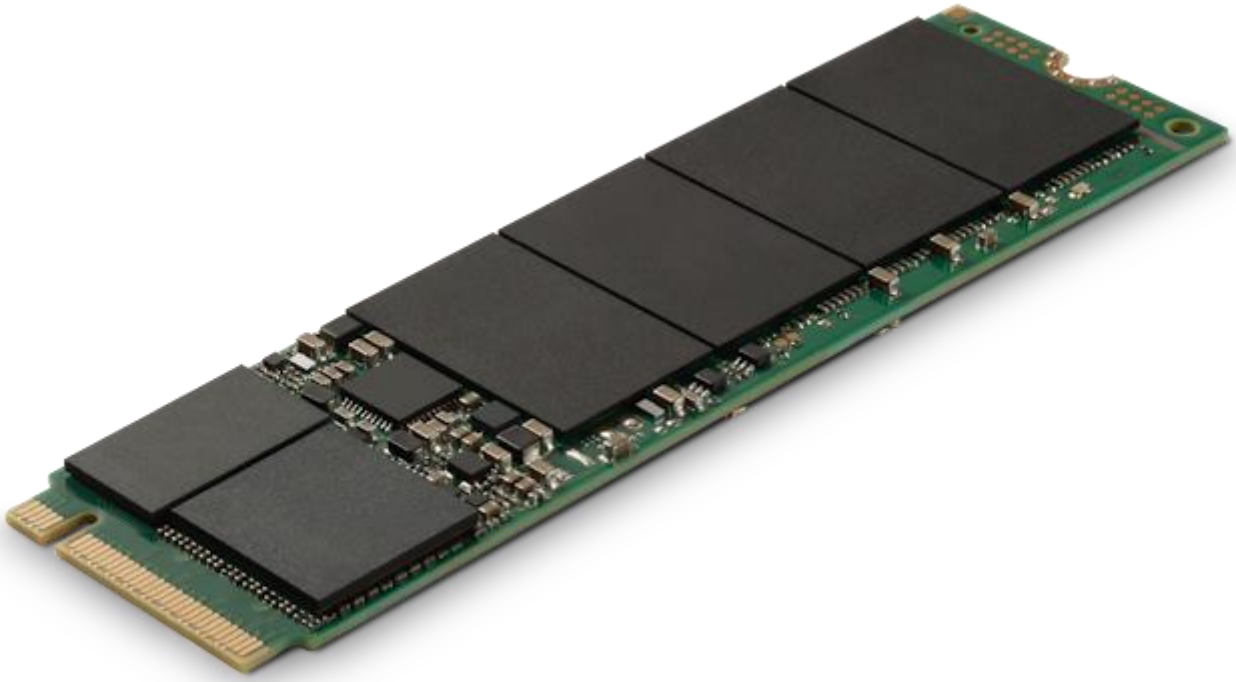
- Para programar uma célula de bit flash, uma alta tensão
  - (no mundo dos computadores, 12 V é uma alta tensão)
- é aplicada à porta de controle, que acelera o processo de injeção de portadora quente na porta flutuante.
- Os elétrons são embutidos na porta flutuante, que coloca uma carga negativa interna no transistor flash.
- A carga negativa embutida aumenta a tensão necessária para ligar o transistor flash e, testando se o canal liga ou não com uma tensão alta ou baixa,
  - é possível determinar se a porta flutuante está carregada ou não, resultando em um valor 0 ou 1 para a célula flash.
- A carga embutida permanece no transistor, mesmo que o sistema perca a alimentação, tornando a célula de memória flash não volátil.

2.3.5. tipos de Memórias

- Visto que os SSDs são basicamente memória, eles possuem desempenho superior aos discos giratórios, com tempo de busca zero.
- Enquanto um disco magnético típico pode acessar dados em até 100 MB/s, um SSD pode operar duas a três vezes mais rápido.
- E como o dispositivo não possui partes móveis, ele é muito adequado para uso em notebooks, onde trepidações e movimentos não afetarão sua capacidade de acessar dados.

2.3.5. tipos de Memórias

- Drive SSD.



2.3.5. tipos de Memórias

- Comparação entre HD e SSD.

2.3.5. tipos de Memórias

- A desvantagem dos SSDs, em comparação com discos magnéticos, é o seu custo. Enquanto os discos magnéticos custam centavos de dólar por gigabyte, um SSD típico custará de um a três dólares por gigabyte, tornando seu uso apropriado apenas para aplicações com drive menor ou em situações em que o custo não é um problema.
- O custo dos SSDs está caindo, mas ainda há um longo caminho até que alcancem os discos magnéticos baratos.
- Assim, embora os SSDs estejam substituindo os discos magnéticos em muitos computadores, talvez ainda leve algum tempo antes que o disco magnético siga o caminho da extinção.