

Organização e Arquitetura de computadores

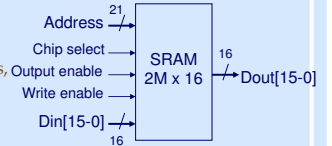
Explorando a Hierarquia de Memória

Prof. Dr. Luciano José Senger

Revisão das tecnologias de memória

- Caches utilizam a tecnologia **SRAM** pelo desempenho

- Baixa densidade (6 transistor cells), consumo elevado, caras, rápidas
- estáticas: enquanto existir alimentação de energia, o conteúdo é preservado

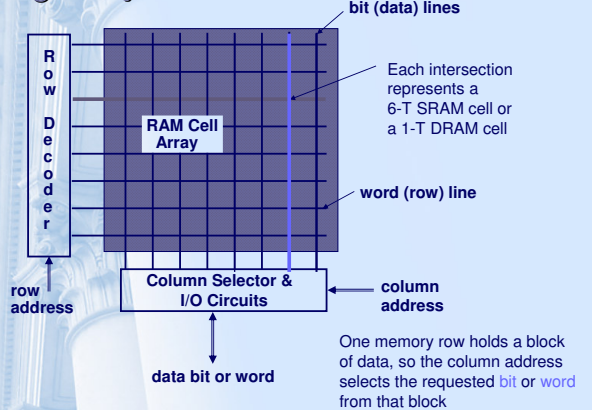


- Memória principal utiliza **DRAM** pelo tamanho (densidade)
 - Alta densidade (1 transistor cells), baixo consumo, baratas, rápidas
 - Dinâmicas: necessitam ser "refreshed" regularmente (~ cada 8 ms)
 - 1% a 2% dos ciclos ativos de uma DRAM
 - Endereços são organizados em 2 partes (row and column)
 - RAS or Row Access Strobe triggering row decoder
 - CAS or Column Access Strobe triggering column selector

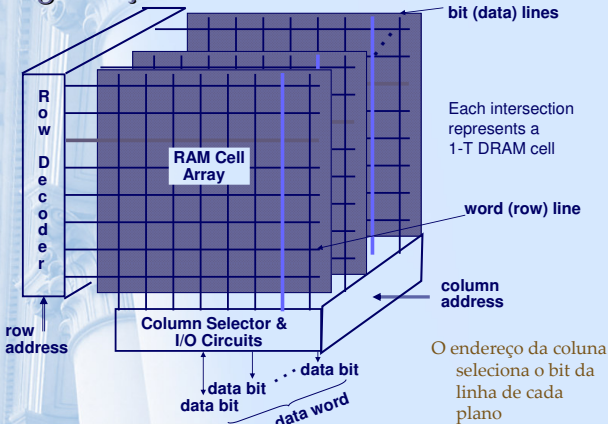
Revisão das tecnologias de memória

- Medidas de desempenho
 - Latência: tempo para acessar uma palavra
 - Tempo de acesso (Access time): tempo entre a requisição e a disponibilidade dos dados (ou uma escrita)
 - Tempo de ciclo (Cycle time): tempo entre requisições
 - Tempo de ciclo > tempo de acesso
 - Tempos de acesso típicos para SRAMS são de 2 a 4ns
 - Largura de banda (Bandwidth): quantos dados podem ser fornecidos ao processador por unidade de tempo
 - Largura do canal de dados * taxa a ser utilizada

Organização clássica de uma RAM

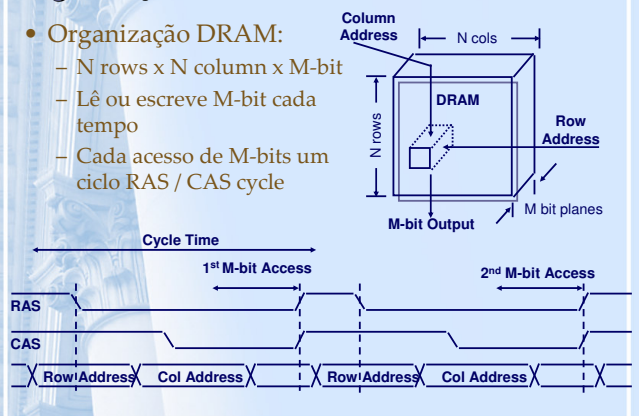


Organização clássica de uma DRAM



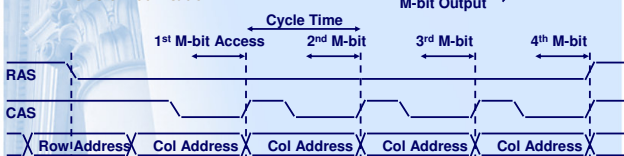
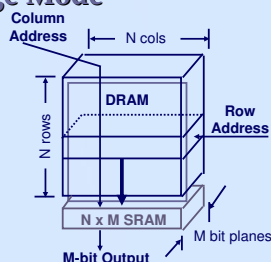
Organização clássica de uma RAM

- Organização DRAM:
 - N rows x N column x M-bit
 - Lê ou escreve M-bit cada tempo
 - Cada acesso de M-bits um ciclo RAS / CAS cycle



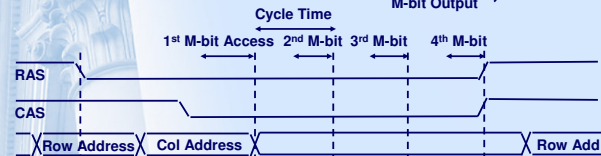
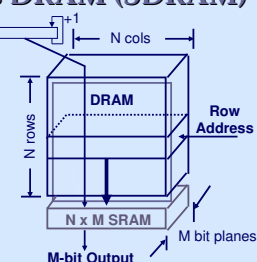
Operação DRAM em Page Mode

- Page Mode DRAM
 - N x M SRAM to save a row
- Após uma linha ser lida dentro do registrador RAS
 - Apenas o CAS é necessário para acessar os outros M-bits daquela linha
 - RAS permanece com o mesmo endereço enquanto o endereço de CAS é modificado

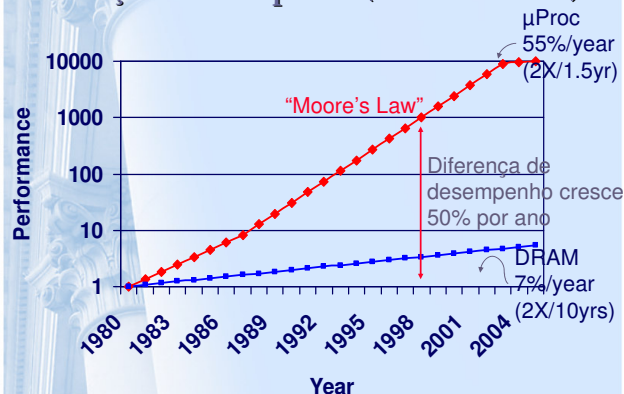


Operação da Synchronous DRAM (SDRAM)

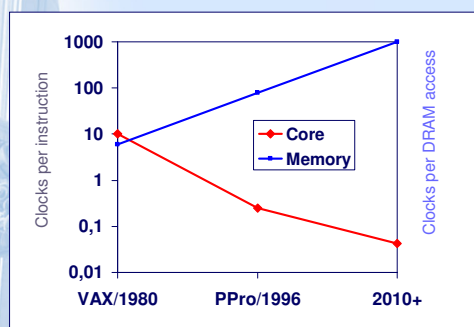
- Após uma linha ser marcada no RAS
 - CAS marcado com o início da rajada
 - A transferência ocorre em rajada com uma série de endereços sequenciais de uma linha
 - Quando DDR, transferência de dados na subida e na descida do clock



Diferença de desempenho (UCPxMemória)

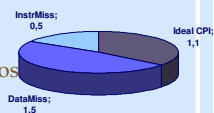


Diferença de desempenho (UCPxMemória)



Impacto no desempenho da computação

- Suponha um processador
 - CPI = 1.1
 - 50% arith/logic, 30% ld/st, 20% desvios
- e que 10% das operações em memória gastam 50 ciclos
- CPI = ideal CPI + espera por memória
 - = 1.1(cycle) + (0.30 (datamemops/instr) x 0.10 (miss/datamemop) x 50 (cycle/miss))
 - = 1.1 cycle + 1.5 cycle = 2.6
- então 58% do tempo do processador é gasto esperando pela resposta da memória!

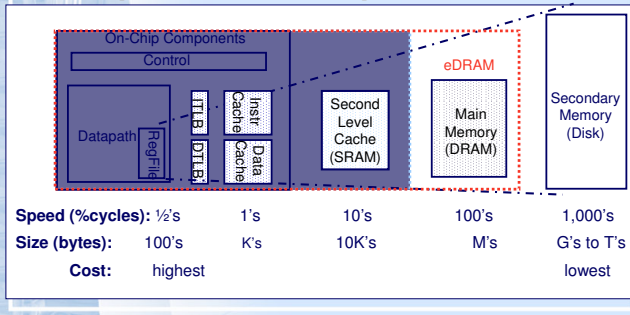


Objetivo de uma hierarquia de memória

- Princípio da localidade
 - Programas acessam uma parte relativa pequena do seu espaço de endereçamento em qualquer instante de tempo
 - Dois tipos
 - Localidade temporal: se um item é referenciado, ele tenderá a ser referenciado novamente em breve
 - Localidade espacial: se um item é referenciado, os itens cujos endereços estão próximos tenderão a ser referenciados em breve
 - Para tirar vantagem dos princípios de localidade, deve-se implementar a memória do computador como uma **hierarquia de memória**
 - Uma hierarquia de memória consiste em múltiplos níveis de memória com diferentes velocidades e tamanhos

Uma hierarquia de memória típica

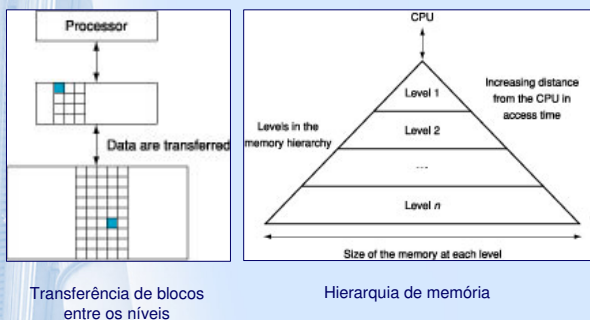
- Hierarquia de memória
 - Os dados são copiados apenas entre os níveis adjacentes
 - Bloco** consiste em uma unidade mínima de informação que pode estar presente ou ausente na hierarquia de dois níveis



Hierarquia de memória

- Se os dados requisitados pelo processador aparecem em algum bloco do nível superior, isso é chamado **acerto**
 - Se os dados não forem encontrados, isso é chamado de **falha**
 - A **taxa de acertos** é a fração dos acessos a memória encontrados no nível superior
 - A **taxa de falhas** (1 - taxa de acertos) é a proporção de acessos à memória não encontrados no nível superior
 - O **tempo de acerto** é o tempo para ter acesso ao nível adjacente da hierarquia
 - A **penalidade de falha** é o tempo para substituir um bloco do nível superior pelo bloco correspondente do nível inferior
 - Tempo de acerto \ll tempo para acesso ao próximo nível

Hierarquia de memória



Como a hierarquia de memória é gerenciada

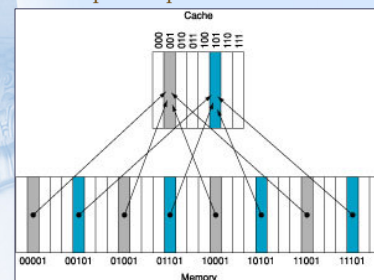
- registradores \leftrightarrow memória
 - Pelo compilador/programador
- cache \leftrightarrow memória principal
 - Pelo controlador de cache (hardware)
- Memória principal \leftrightarrow disco
 - Pelo sistema operacional, através da memória virtual
 - O mapeamento de endereços virtuais para endereços físicos é assistido pelo hardware (TLB)
 - Pelo programador (arquivos)

Memória Cache

- O hardware deve ser implementado para responder duas questões
 - Q1: Como saber se os itens procurados estão na memória cache?
 - Q2: Se estiver, como encontrá-lo
- A implementação da memória cache pode ser realizada de diferentes formas para responder essa pergunta
 - A forma de implementação afeta diretamente o desempenho
- Mapeamento direto
 - Para cada item de dados no nível mais baixo, existe apenas uma localização na cache onde ele pode estar
 - Uma quantidade de itens de dados do nível inferior deve **compartilhar** localizações no nível mais alto
 - Mapeamento de endereço
(endereço do bloco) módulo (número de blocos na cache)
 - Inicialmente, serão considerados tamanho de blocos igual a uma palavra

Memória cache com mapeamento direto

- Exemplo
 - Cache diretamente mapeada com oito entradas mostrando os endereços das *words* de memória entre 0 e 31 que são mapeadas para os mesmos locais na cache



Memória cache com mapeamento direto

- Como cada local da cache pode armazenar o conteúdo de diversos locais de memória diferentes, como podemos saber se os dados na cache correspondem a uma word requisitada?
 - Inclusão de um conjunto de tags
 - Tags contém informações de endereço necessárias para identificar se uma word na cache corresponde a uma word requisitada
 - Precisamos apenas ter os dois bits mais significativos do endereço que seleciona o bloco no campo tag
 - exclui-se os bits pois eles são redundantes (ver figura)
 - Deve-se verificar também se o bloco da cache não tem informações não válidas
 - Mesmo após executar várias instruções, algumas entradas na cache podem ter informações não válidas (vazias)
 - Método mais comum é a inserção de um bit de validade

Obtendo acesso a uma memória cache

Índice	V	Tag	Dados
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. Estado inicial da cache após a inicialização

Índice	V	Tag	Dados
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	S	10 _{bin}	Memória (10110 _{bin})
111	N		

b. Após tratar uma falha no endereço (10110_{bin})

Índice	V	Tag	Dados
000	N		
001	N		
010	S	11 _{bin}	Memória (11010 _{bin})
011	N		
100	N		
101	N		
110	S	10 _{bin}	Memória (10110 _{bin})
111	N		

c. Após tratar uma falha no endereço (11010_{bin})

Índice	V	Tag	Dados
000	S	10 _{bin}	Memória (10000 _{bin})
001	N		
010	S	11 _{bin}	Memória (11010 _{bin})
011	S	00 _{bin}	Memória (00011 _{bin})
100	N		
101	N		
110	S	10 _{bin}	Memória (10110 _{bin})
111	N		

d. Após tratar uma falha no endereço (10000_{bin})

Índice	V	Tag	Dados
000	S	10 _{bin}	Memória (10000 _{bin})
001	N		
010	S	10 _{bin}	Memória (10010 _{bin})
011	S	00 _{bin}	Memória (00011 _{bin})
100	N		
101	N		
110	S	10 _{bin}	Memória (10110 _{bin})
111	N		

e. Após tratar uma falha no endereço (00011_{bin})

Índice	V	Tag	Dados
000	S	10 _{bin}	Memória (10000 _{bin})
001	N		
010	S	10 _{bin}	Memória (10010 _{bin})
011	S	00 _{bin}	Memória (00011 _{bin})
100	N		
101	N		
110	S	10 _{bin}	Memória (10110 _{bin})
111	N		

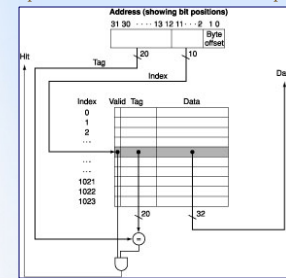
f. Após tratar uma falha no endereço (10010_{bin})

Obtendo acesso a uma memória cache

Endereço decimal da referência	Endereço binário da referência	Acerto ou falha na cache	Bloco de cache atribuído (onde foi encontrado ou inserido)
22	10110 _{bin}	falha (7.6b)	(10110 _{bin} mod 8) = 110 _{bin}
26	11010 _{bin}	falha (7.6c)	(11010 _{bin} mod 8) = 010 _{bin}
22	10110 _{bin}	acerto	(10110 _{bin} mod 8) = 110 _{bin}
26	11010 _{bin}	acerto	(11010 _{bin} mod 8) = 010 _{bin}
16	10000 _{bin}	falha (7.6d)	(10000 _{bin} mod 8) = 000 _{bin}
3	00011 _{bin}	falha (7.6e)	(00011 _{bin} mod 8) = 011 _{bin}
16	10000 _{bin}	acerto	(10000 _{bin} mod 8) = 000 _{bin}
18	10010 _{bin}	falha (7.6f)	(10010 _{bin} mod 8) = 010 _{bin}

Memória cache com mapeamento direto

- Endereço referenciado é dividido em:
 - Um índice de cache, utilizado para selecionar um bloco
 - Um campo tag, usados para ser comparado com o valor do campo tag da cache
 - Exemplo: cache com 1024 words (2**10) e tamanho de bloco de 1 palavra
 - 10 bits são usados para indexar a cache, mais 20 bits para tag

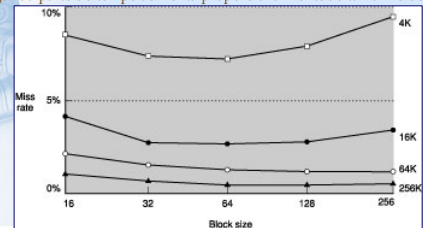


Memória cache

- Mapeando o endereço para um bloco de cache multiword
 - Considere uma cache com 64 blocos e um tamanho de bloco de 16 bytes. Para qual número de bloco o endereço 1200 é mapeado?
 - (Endereço do bloco) modulo (número de blocos na cache)
 - Onde o endereço do bloco é (endereço em bytes/bytes por bloco)
 - note, se bytes por bloco é igual a 1, (endereço em bytes)=(endereço do bloco)
 - 1200/16 = 75
 - 75 mod 64 = 11, na verdade, esse bloco mapeia todos os endereços entre 1200 e 1215
 - Blocos maiores
 - Exploram a localidade espacial para diminuir as taxas de falhas

Memória cache

- Blocos maiores, vale a pena?
 - A taxa de falhas pode subir posteriormente se o tamanho de bloco se tornar uma fração significativa do tamanho da cache, uma vez que o número de blocos que pode ser armazenado na cache se tornará pequeno e haverá uma grande competição entre blocos (é claro, mantendo o tamanho total da cache)
 - O custo da falha aumenta na medida que aumenta o tamanho do bloco (maior tempo de transferência)
 - O tempo para buscar o bloco pode ser organizado em duas partes: a latência até a primeira word e o tempo de transferência para o restante do bloco
 - Segunda parte do tempo aumenta proporcionalmente ao tamanho do bloco



Memória cache

- Blocos maiores
 - A desvantagem principal de aumentar o tamanho do bloco é que a penalidade de falha aumenta
 - Solução: reinício precoce: retornar para o processador quando a palavra solicitada estiver disponível, ao invés de esperar o bloco inteiro
 - Técnica de reinício precoce funciona bem para cache de instruções (execução sequencial é típica)
 - Não funciona bem para cache de dados (requisições são menos previsíveis que a na cache de instruções)
 - Técnica *word requisitada primeiro*: similar a reinício precoce, mas com acesso direto a *word* requisitada e então transfere o restante do bloco (duas partes: antes e depois da *word* requisitada)
 - sofre dos mesmos problemas

Tratando falhas na cache

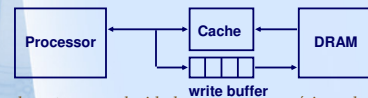
- Unidade de controle
 - Fácil para o acerto
 - Ao contrário, o processamento de uma falha cria um *stall* semelhante ao do pipeline, mas de maneira mais simples, congelando toda a execução até que a memória responda
 - Ações
 - Enviar o valor do PC original (PC atual - 4) para a memória
 - Instruir a memória principal para realizar uma leitura e esperar que a memória complete seu acesso
 - Escrever na entrada da cache (busca)
 - Reiniciar a execução da instrução na primeira etapa, o que buscará novamente a instrução, desta vez encontrando-a na cache

Tratando escritas

- Como manter a cache de dados consistente?
 - Uma operação store envia uma *word* para a memória; deve-se garantir que instruções subsequentes que necessitem dessa *word* obtenham o valor atualizado
 - Método mais simples: manter a memória sempre atualizada, sempre escrevendo os dados na memória e na cache: **write-through**
 - Outro aspecto importante é que o bloco atualizado na cache pode ser substituído por um outro bloco, perdendo assim a consistência
 - Apesar de simples, o esquema **write-through** não apresenta bom desempenho: toda escrita faz com que os dados trafeguem para a memória principal (imagina uma situação em que um programa trabalha com *load/stores* várias vezes em um vetor, antes de ter uma versão atualizada deste vetor: muitos acessos a memória desnecessários!)
 - Escritas gastam mais de 100 ciclos de máquina
 - Para o SPEC2000, 10% das operações são stores
 - Para um computador com CPI=1, tem-se $1,0 + 100 \times 10\% = 11$, reduzindo o desempenho em 11%

Tratando escritas

- Buffer de escrita
 - Permite que os dados sejam armazenados em um buffer de espera, reduzindo o custo do acesso para o processador



- Naturalmente, se a velocidade em que a memória pode completar escritas for menor do que a velocidade em que o processador está gerando escritas, nenhuma quantidade de *buffer* pode ajudar, pois as escritas estão sendo geradas mais rápido do que o sistema de memória pode aceitá-las
- **Write-back**
 - Alternativa ao **write-through**, na qual quando ocorre uma escrita, o valor recente é apenas gravado na memória cache
 - O bloco é escrito em um nível inferior da hierarquia apenas quando é substituído
 - Necessário mais um bit na cache para identificar antes do bloco ser substituído em uma falha, se este bloco deve ser escrito em memória

Tratando escritas

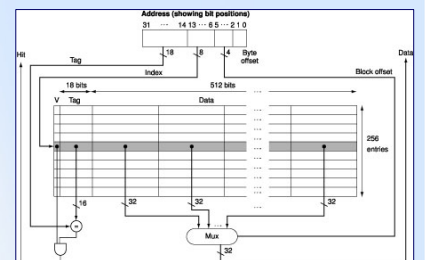
- Escritas trazem várias complicações
 - Como tratar uma falha, na escrita, em uma cache **write-through**?
 - Buscar na falha: aloca um bloco de cache para o endereço que falhou e busca o restante do bloco para a cache antes de escrever a *word* e continuar a execução
 - Não buscar na escrita: alocar o bloco na cache não buscar os dados
 - Write-around: dados escritos na cache antes de chegar a memória (desrespeitam a "atomicidade" do bloco)
 - A motivação é que as vezes os programas escrevem várias *words* antes de efetuar leituras; mas tem-se várias complicações
 - Em esquema **write-through**

Um cache de exemplo: o processador Intrinsity FastMath

- Intrinsity FastMath
 - Microprocessador embutido veloz (~2,5 G) que usa a arquitetura MIPS e uma implementação de cache simples
 - Pipeline de 12 estágios
 - Caches de instruções e de dados separadas
 - Cada cache tem 16KB, ou 4K words, com blocos de 16 words
 - Write-through e write-back: o sistema operacional pode escolher a política de atualização



- Cache dividida
 - Taxa de acertos melhor
 - Uma cache é direcionada para os dados e outra para as instruções



Projetando o sistema de memória integrados com caches

- DRAM

- Embora seja difícil reduzir a latência para buscar a primeira word da memória, podemos reduzir a penalidade de falha se aumentarmos a largura de banda da memória para a cache
 - Essa redução permite que blocos maiores possam ser empregados, sem aumentar a penalidade de falha, deixando a penalidade similar a um bloco menor

- Processador se comunica com a memória com barramento

- Conjunto hipotético

- 1 ciclo de clock de barramento de memória para enviar o endereço
- 15 ciclos de clock de barramento de memória para cada acesso a DRAM iniciado
- 1 ciclo de clock de barramento para enviar uma word de dados

- Se tivermos um bloco de cache de quatro words e um banco de DRAMS com largura de uma word, a penalidade de falha seria:

- $1 + (4 \times 15) + (4 \times 1) = 65$ ciclos de clock de barramento

- $(4 \times 4) / 65 = 0,25$ bytes transferidos por ciclo de clock de barramento

Projetando o sistema de memória integrados com caches

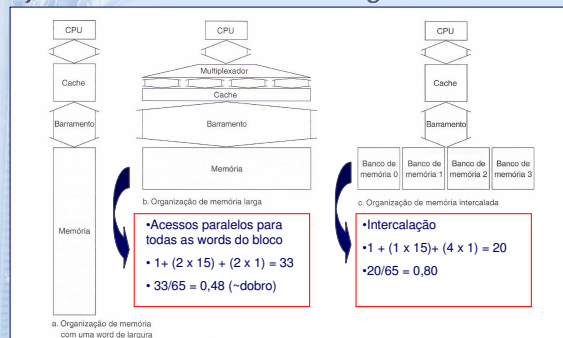


FIGURA 7.11 O principal método para obter largura de banda de memória mais alta é aumentar a largura física ou lógica do sistema de memória. Nesta figura, a largura de banda da memória é melhorada de duas maneiras. O projeto mais simples (a) usa uma memória na qual todos os componentes possuem uma word de largura; (b) mostra uma memória, um barramento e uma cache mais largos; enquanto (c) mostra um barramento e uma cache mais estreitos com uma memória intercalada. Em (b), a lógica entre a cache e o processador consiste em um multiplexador usado em leitura e lógica de controle para atualizar as words apropriadas da cache nas escritas.

Leituras recomendadas

- Patterson, David A. e Hennessy, John L. **Organização e Projeto de Computadores: A Interface Hardware/Software.** Ed. LTC, 332. Ed., 2004, Rio de Janeiro. Capítulo 7
- Slides de arquitetura de computadores
 - www.cse.psu.edu/~mji
- Tutorial sobre memória
 - http://www.corsairmemory.com/corsair/products/tech/memory_basics/153707/main