

Parte 4

Memória

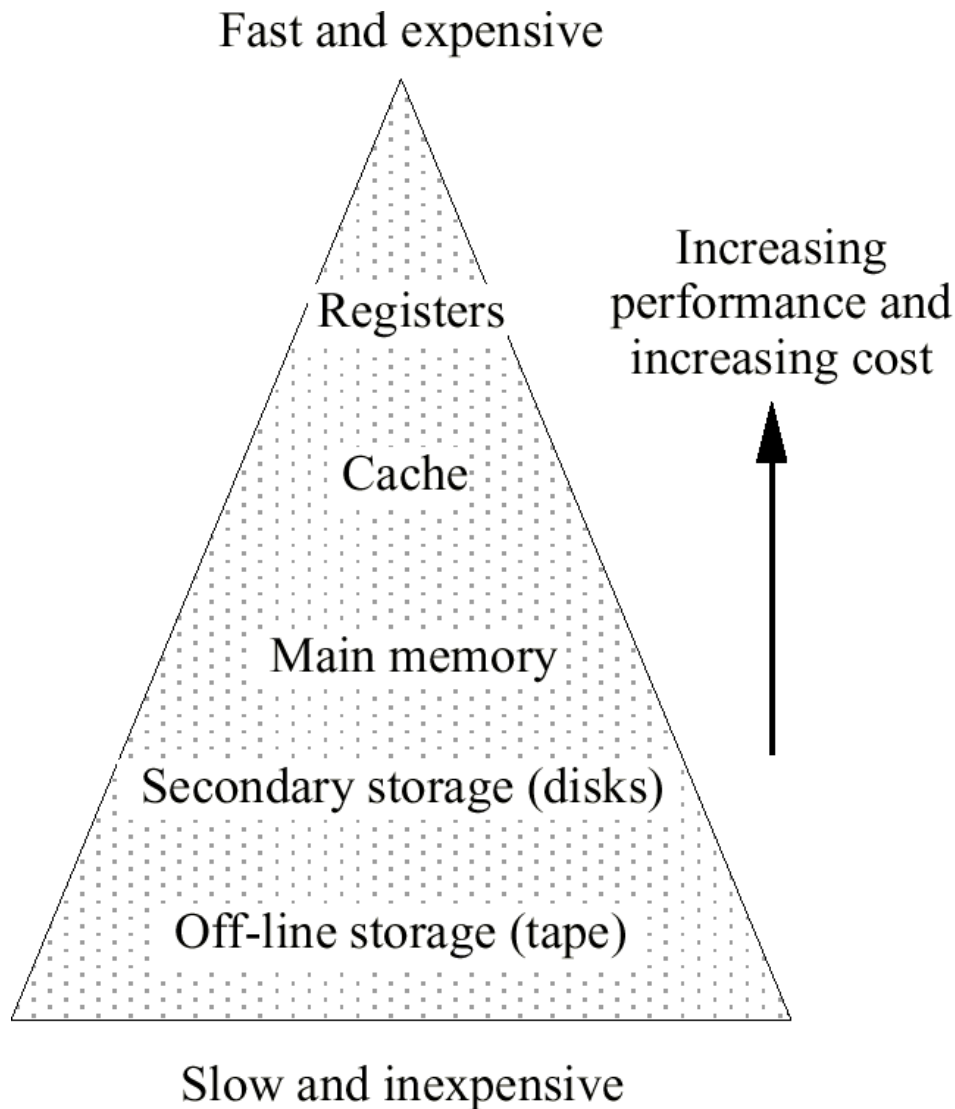
Bibliografia

- [1] Miles J. Murdocca e Vincent P. Heuring, “Introdução à Arquitetura de Computadores”
- [2] Notas de Aula do Prof. Marcelo Rubinstein
- [3] Notas de Aula do Prof. Brafman
- [4] Patterson
- [5] Andrew S. Tanenbaum, “Modern Operating Systems”

Histórico

- **CPU**
 - Velocidade de processamento dobra a cada 18 meses
- **Memória**
 - Tamanho quadruplica a cada 36 meses
 - Mas velocidade aumenta em 10% ao ano
- **Soluções de arquitetura tentam “diminuir” a diferença**
 - Computador possui uma combinação de diferentes tipos de memória

A Hierarquia de Memória



- **registrador**
 - **velocidade da CPU**
 - **grandes**
 - **alto consumo**
- **discos rígidos**
 - **tempo de acesso muito longo**
 - **custo por bit baixo**

Propriedades da hierarquia de memória no fim dos anos 90

Tipo de memória	Tempo de acesso	Custo por MB	Quantidade típica usada	Custo típico
Registradores	1 ns	Alto	1 KB	-
Cache	5 – 20 ns	US\$100	1 MB	US\$100
Memória principal	60 – 80 ns	US\$1,10	64 MB	US\$70
Memória em disco	10 ms	US\$0,05	4 GB	US\$200

Memória de Acesso Aleatório

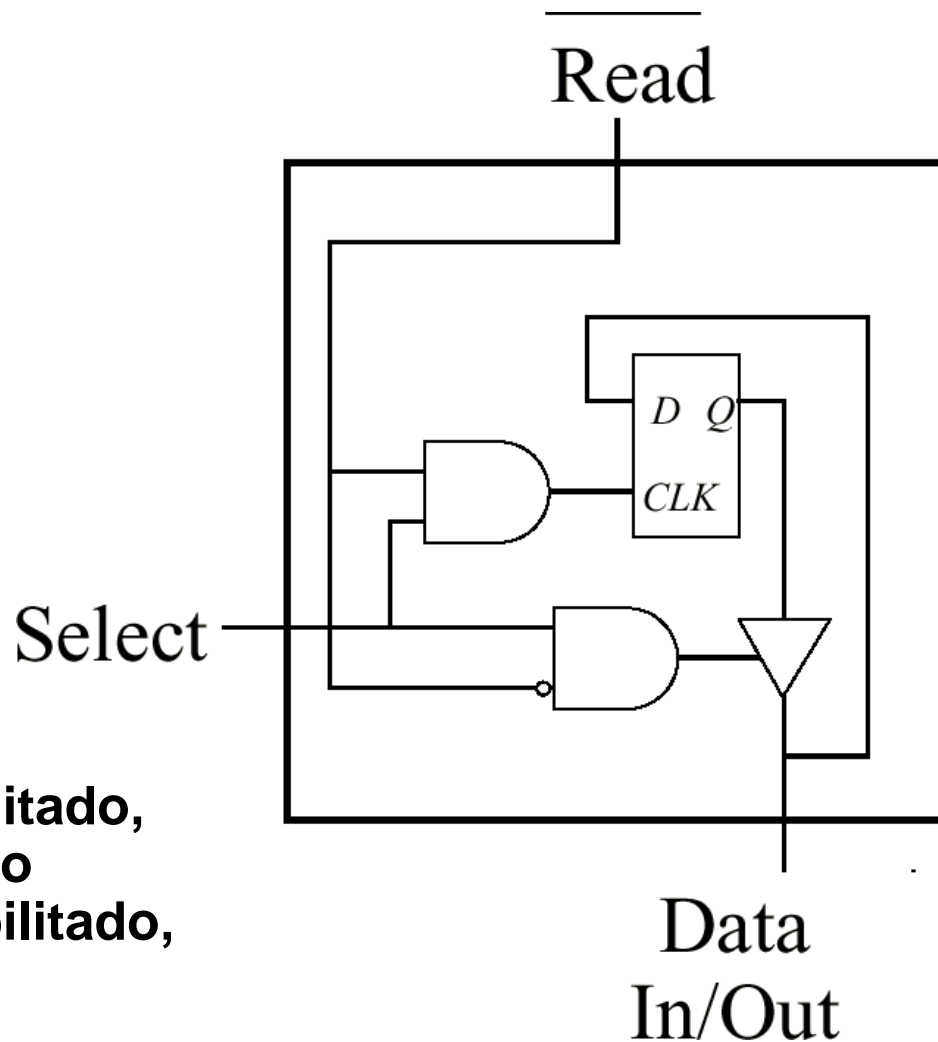
- **RAM (*Random Access Memory*)**
 - **acesso aleatório**
 - qualquer posição de memória acessada na mesma quantidade de tempo
- **RAMs estáticas (SRAM)**
 - **baseadas em flip-flops**
 - conteúdo persiste enquanto circuito alimentado
 - mais rápidas
- **RAMs dinâmicas (DRAM)**
 - **baseadas em capacitores**
 - carga deve ser restaurada periodicamente
 - menores, mais econômicas

Comportamento Funcional de uma Célula RAM

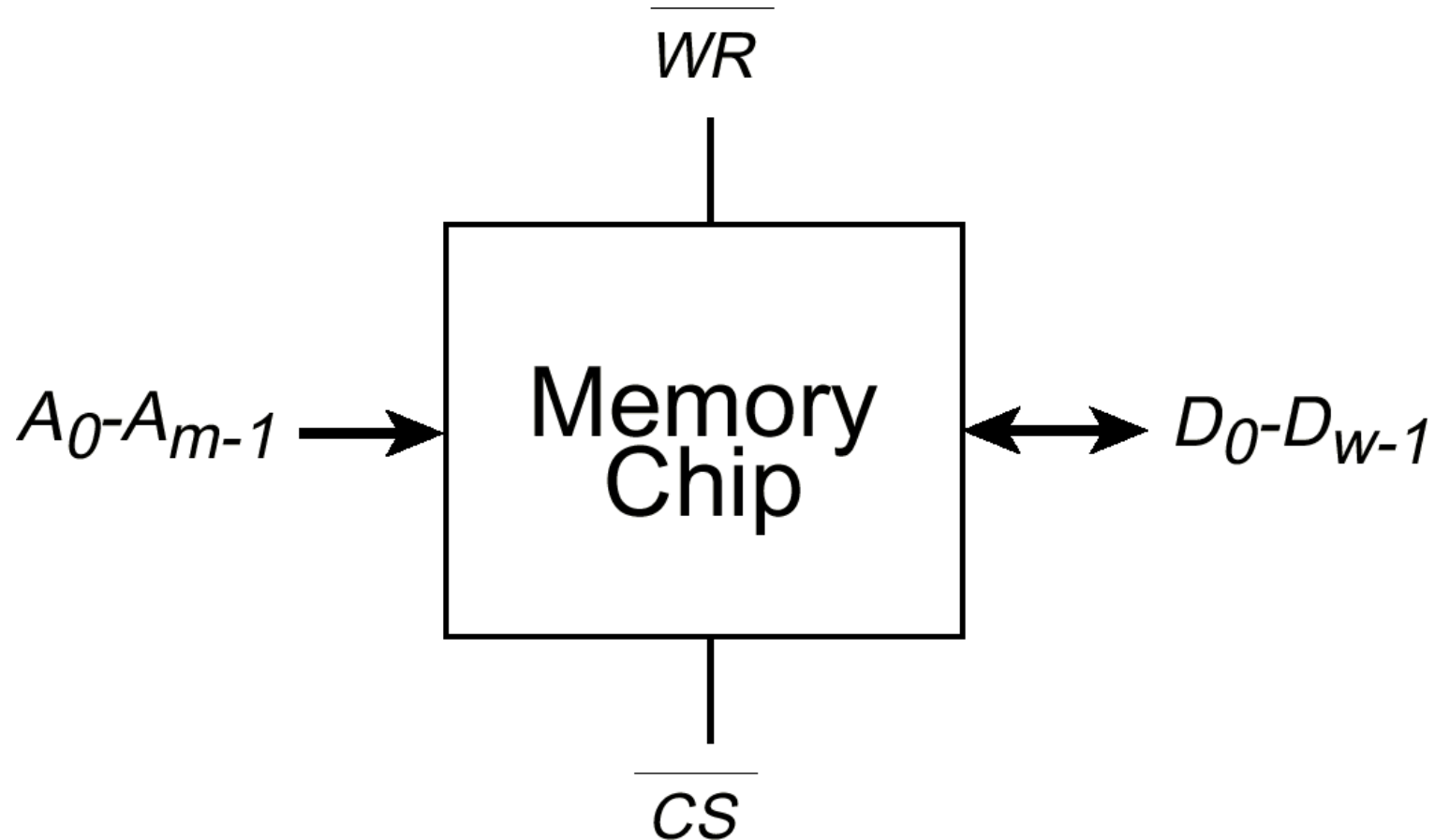
(Existem diversas implementações físicas)

Linhas permitem selecionar, ler e escrever a célula

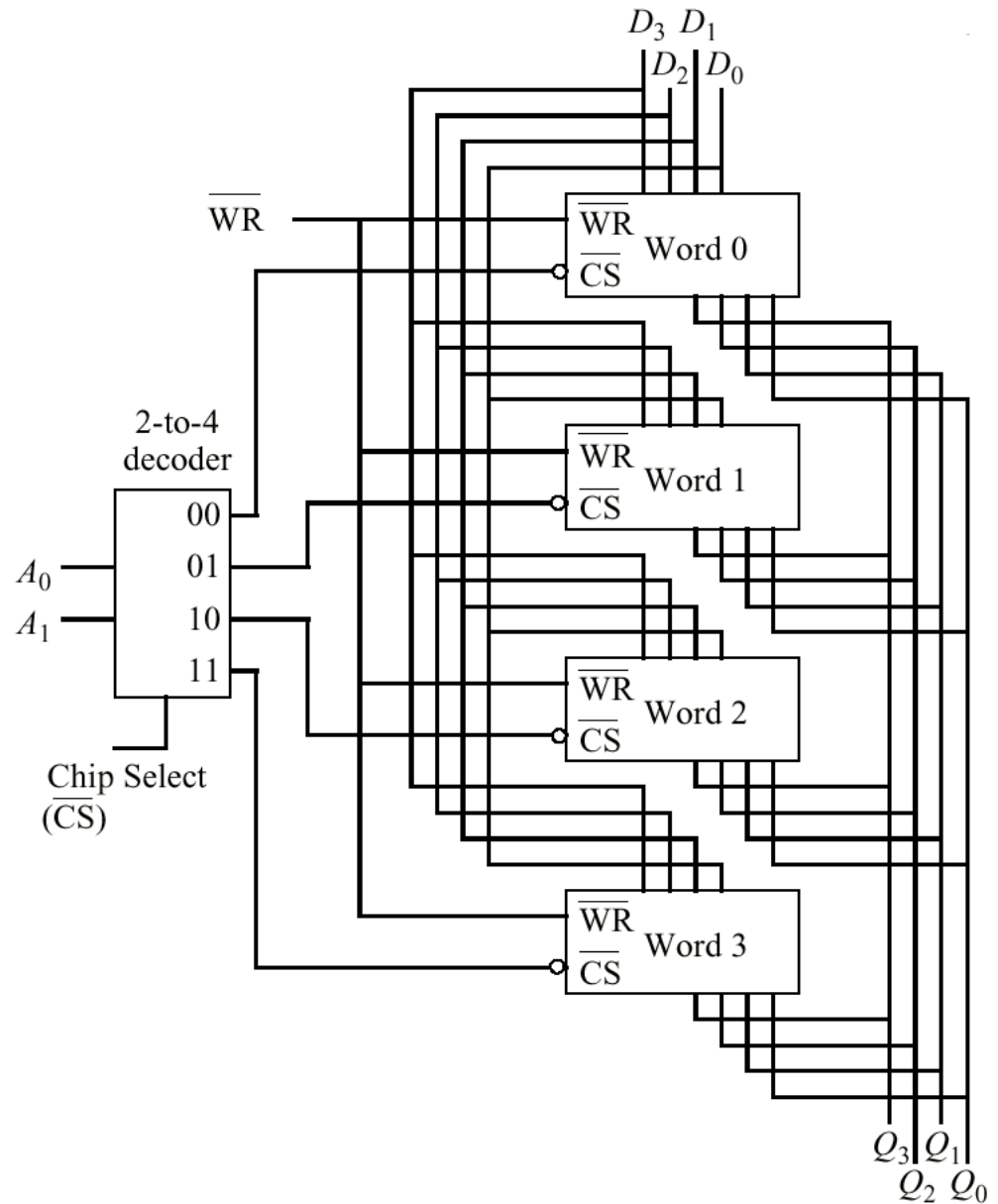
Leitura: tri-state habilitado, clock desabilitado
Escrita: tri-state desabilitado, clock vai pra 1



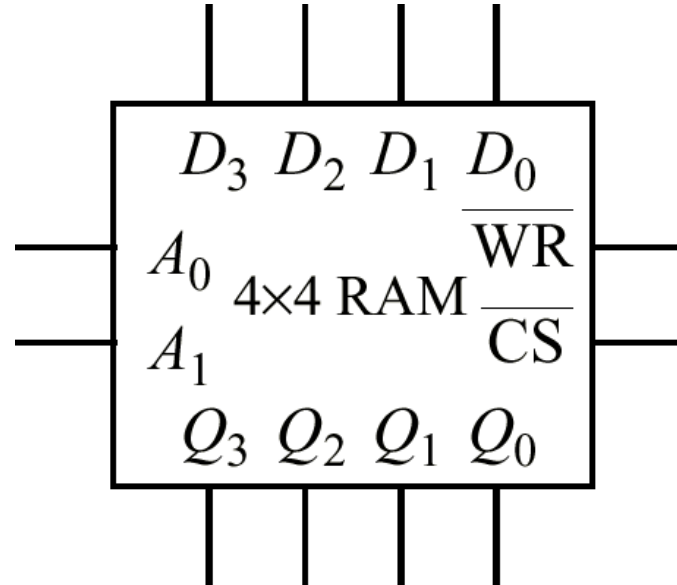
Pinagem Simplificada de um Chip RAM



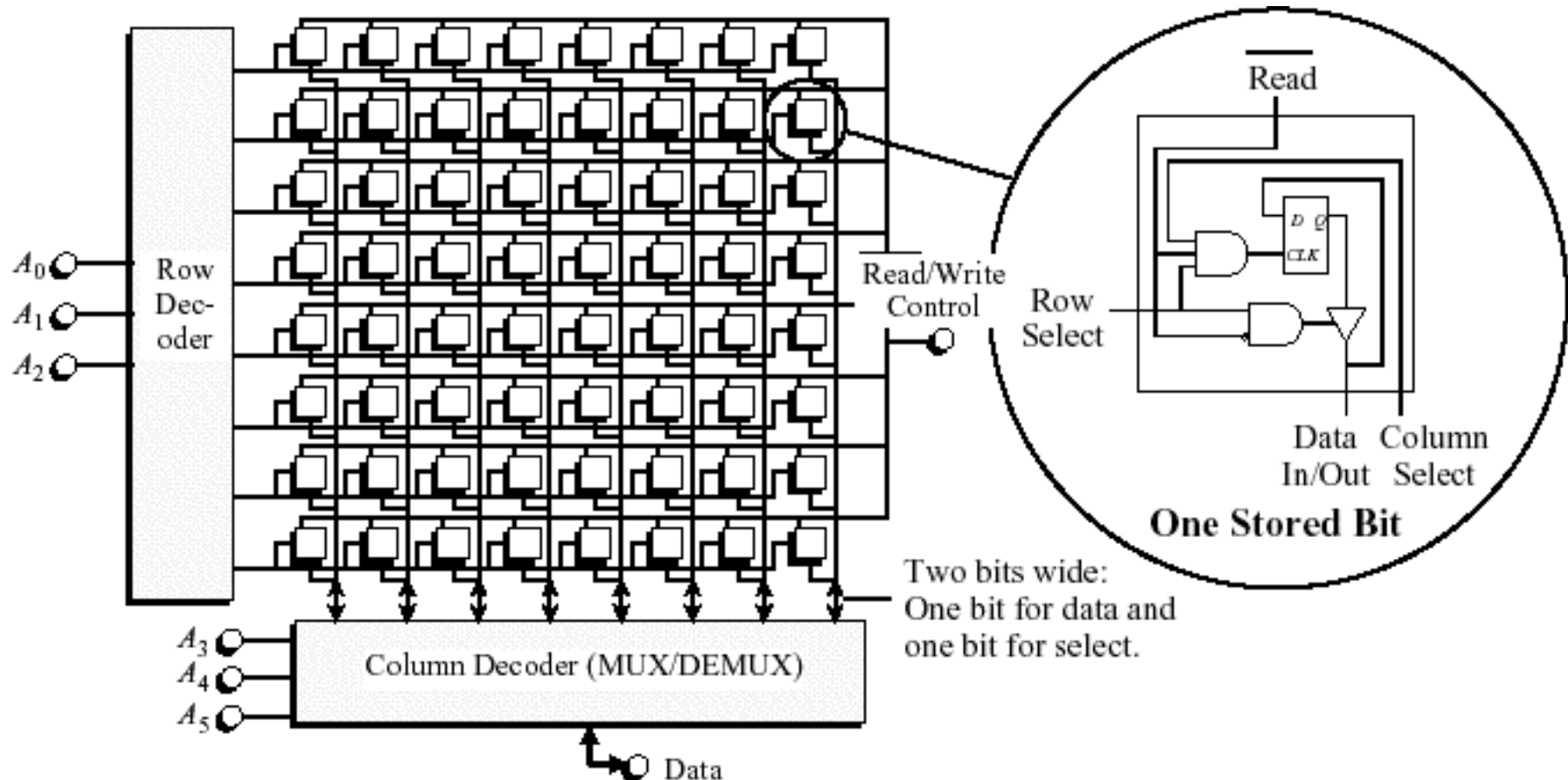
Uma memória de 4 palavras de 4 bits, organizada em 2D



Representação Simplificada da RAM de 4 palavras de 4 bits



Organização 2,5D de uma RAM de 64 palavras por 1 bit



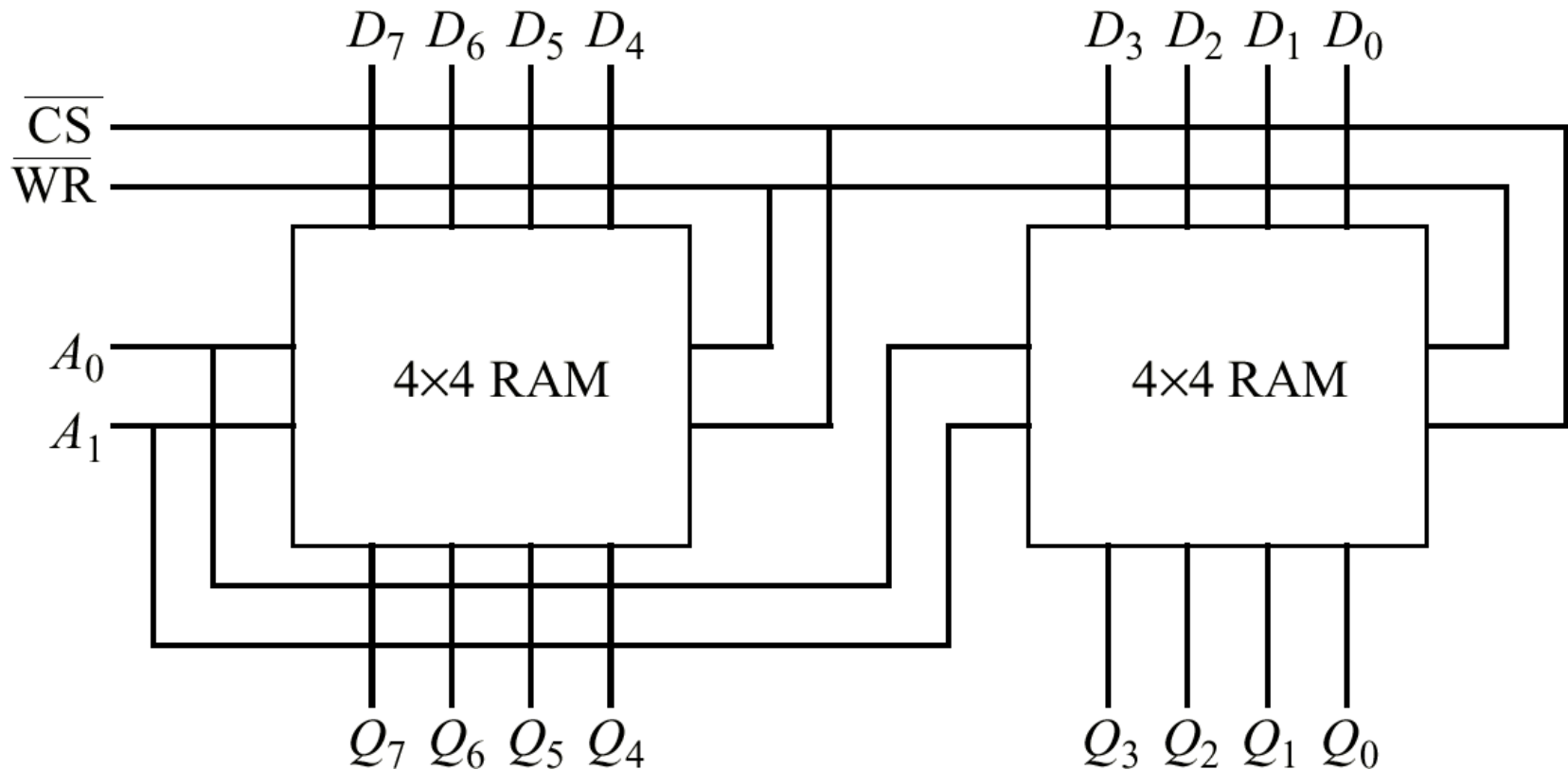
Funcionamento da Organização 2,5D

- **Leitura**
 - 1 linha inteira é selecionada, alimentada ao MUX de colunas
 - MUX seleciona um único bit para saída
- **Escrita**
 - bit a ser escrito é distribuído pelo DEMUX para a coluna-alvo
 - decodificador de linhas seleciona a linha correta

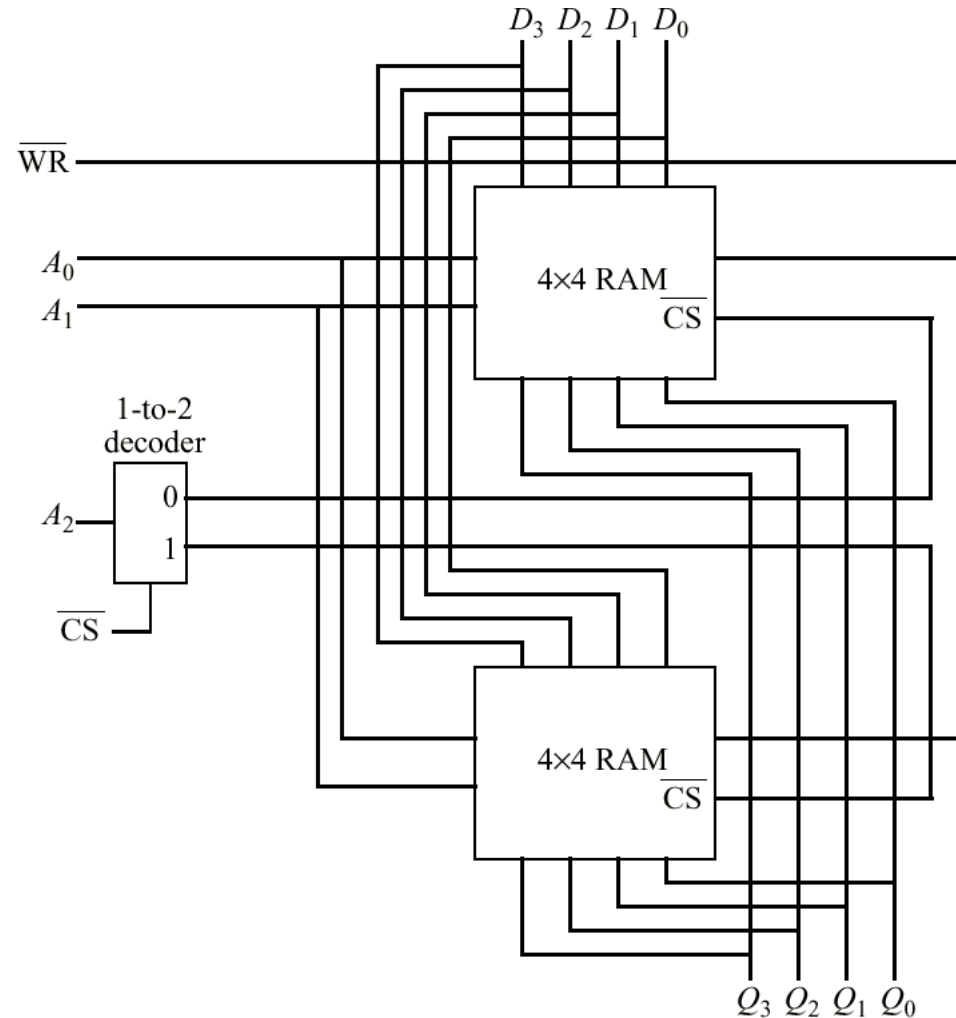
Funcionamento da Organização 2,5D

- Reduzindo o número de pinos
 - Existem somente $m/2$ pinos de endereço no chip
- $m/2$ endereços de linha são enviados com sinal $\overline{\text{RAS}}$ (*Row Address Strobe*)
 - endereço de linha decodificado e armazenado pelo chip
- $m/2$ endereços de coluna são enviados com sinal $\overline{\text{CAS}}$ (*Column Address Strobe*)

2 RAMs de 4 palavras de 4 bits usadas para criar 1 RAM de 4 palavras de 8 bits



2 RAMs de 4 palavras 4 bits formam uma RAM de 8 palavras de 4 bits



Single-In-Line Memory Module (SIMM)

PIN NOMENCLATURE	
A0-A9	Address Inputs
$\overline{\text{CAS}}$	Column-Address Strobe
DQ1-DQ8	Data In/Data Out
NC	No Connection
$\overline{\text{RAS}}$	Row-Address Strobe
V_{CC}	5-V Supply
V_{SS}	Ground
$\overline{\text{W}}$	Write Enable

- Adaptado de (Texas Instruments, *MOS*

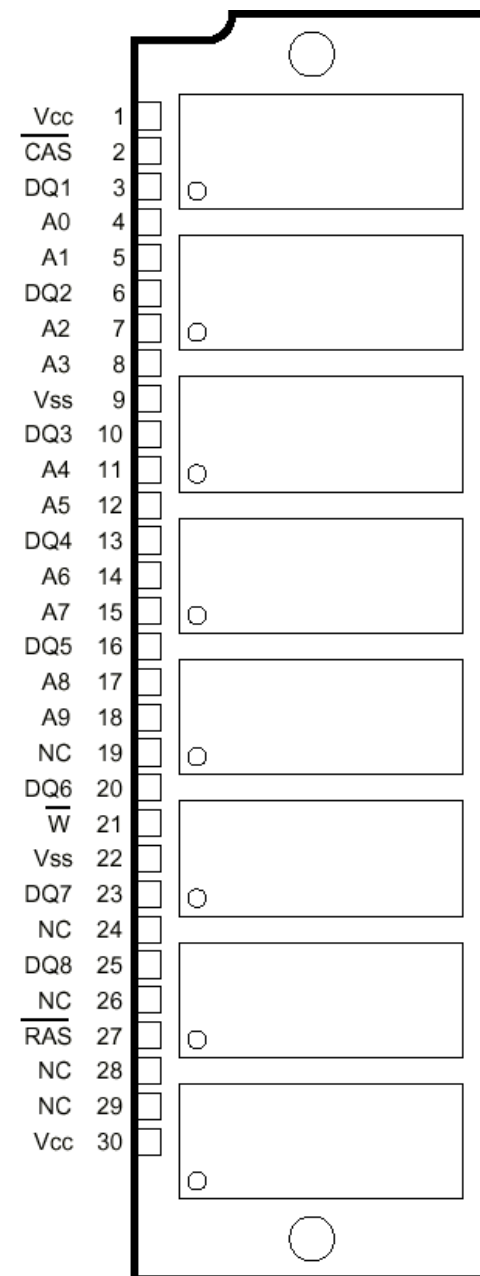
Memory: Commercial and Military

Specifications Data Book, Texas

Instruments, Literature Response

Center, P.O. Box 172228, Denver,

Colorado, 1991.)



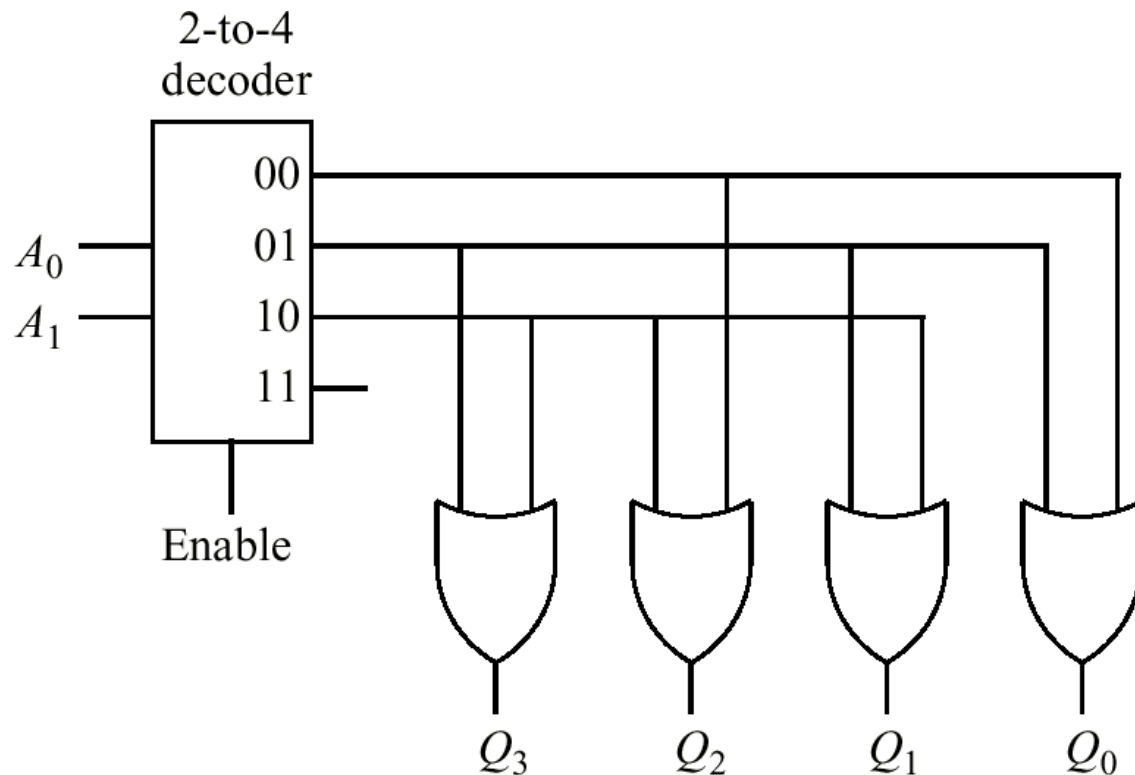
Módulo SIMM

- **Endereçamento**
 - RAS – linha
 - CAS – coluna
- **8 bits lidos ou escritos em paralelo**
 - para formar palavras de 32 bits, 4 módulos são necessários
- \overline{W} – write enable
 - ativo em zero
- **RAS**
 - também causa restauração
 - deve ser feita pelo menos a cada 8 ms para este chip

Memórias Somente de Leitura

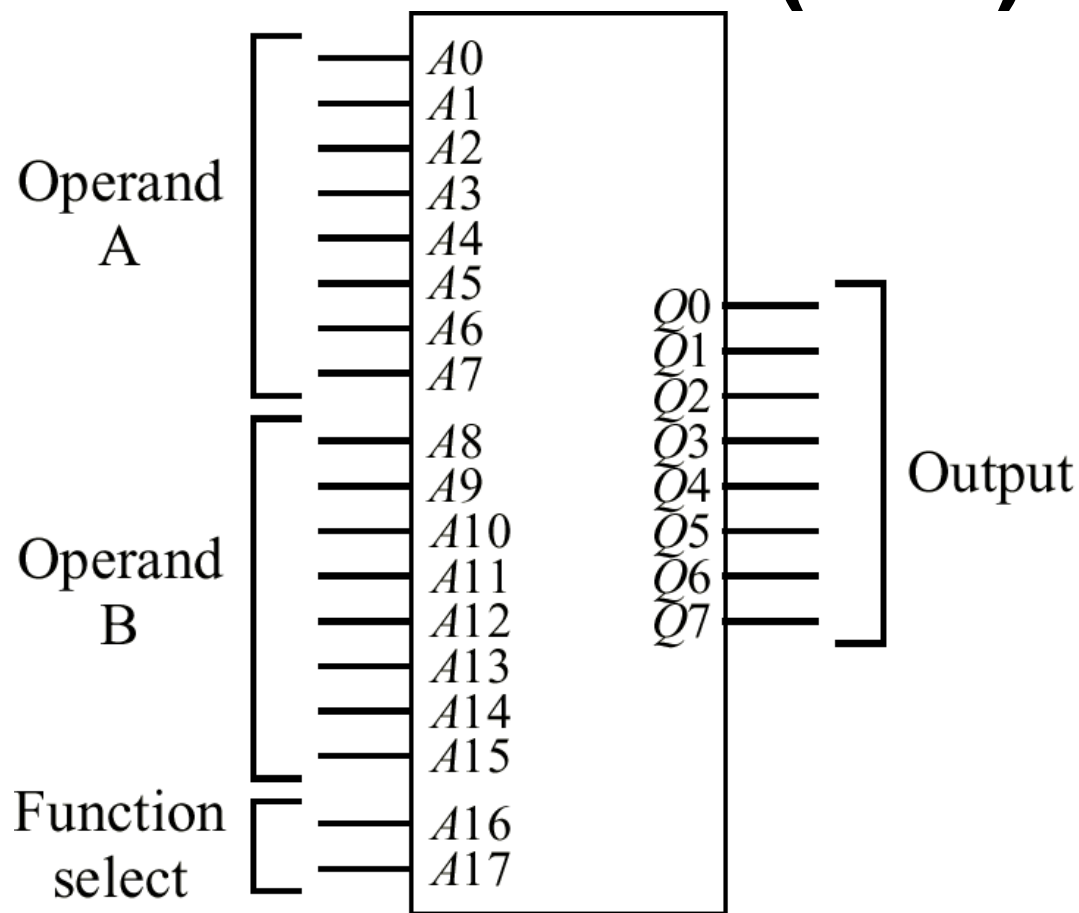
- ROM (*Read-Only Memory*)
 - Simples: decodificador, linhas de saída e portas lógicas
 - Aplicações de alto volume
- PROM (*Programmable ROM*)
 - Baixo volume e protótipos
 - Conteúdo escrito com um queimador de PROMs
- EPROM (*Erasable PROM*)
 - Apagada por luz ultra-violeta
- EEPROM (*Electrically Erasable PROM*)

Uma ROM de 4 palavras de 4 bits



Location	Stored word
00	0101
01	1011
10	1110
11	0000

Uma ALU de 8 bits implementada com uma Lookup Table (LUT)



$A17$	$A16$	Function
0	0	Add
0	1	Subtract
1	0	Multiply
1	1	Divide

Memória Cache

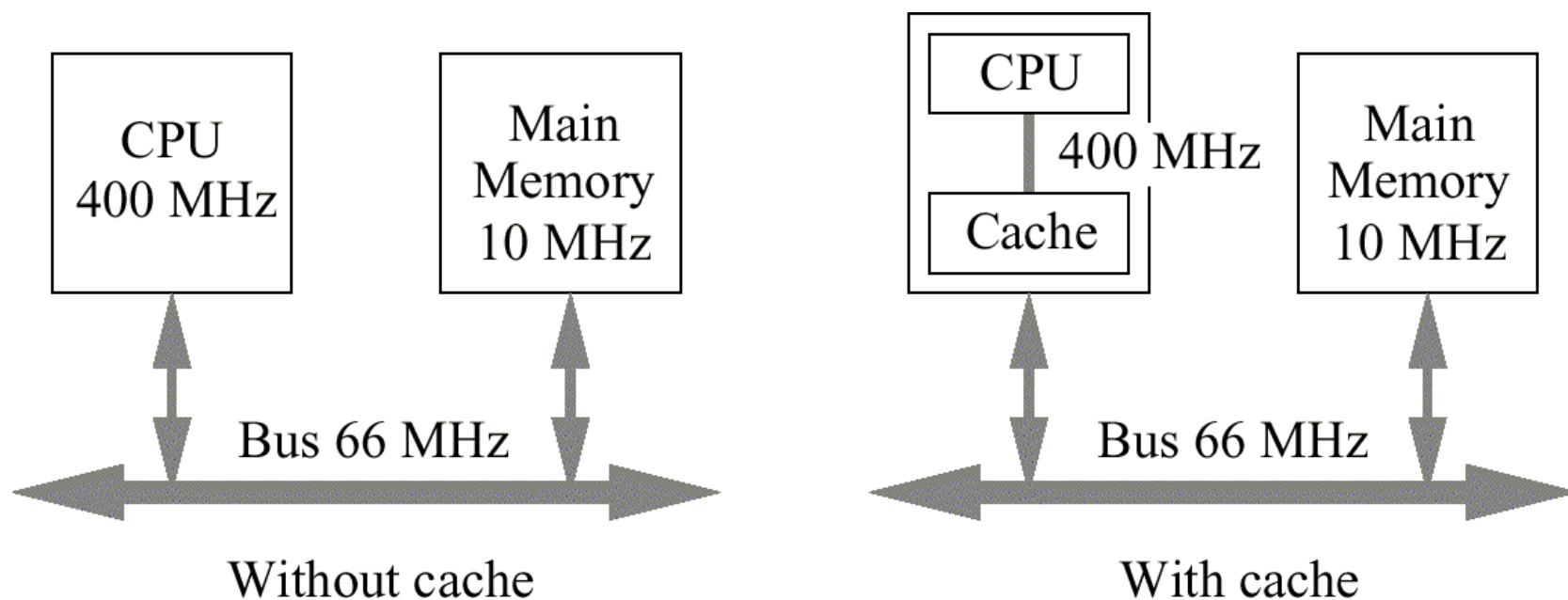
- **Princípio básico**

Na execução de um programa de computador, muitas das referências são a um pequeno conjunto de posições de memória

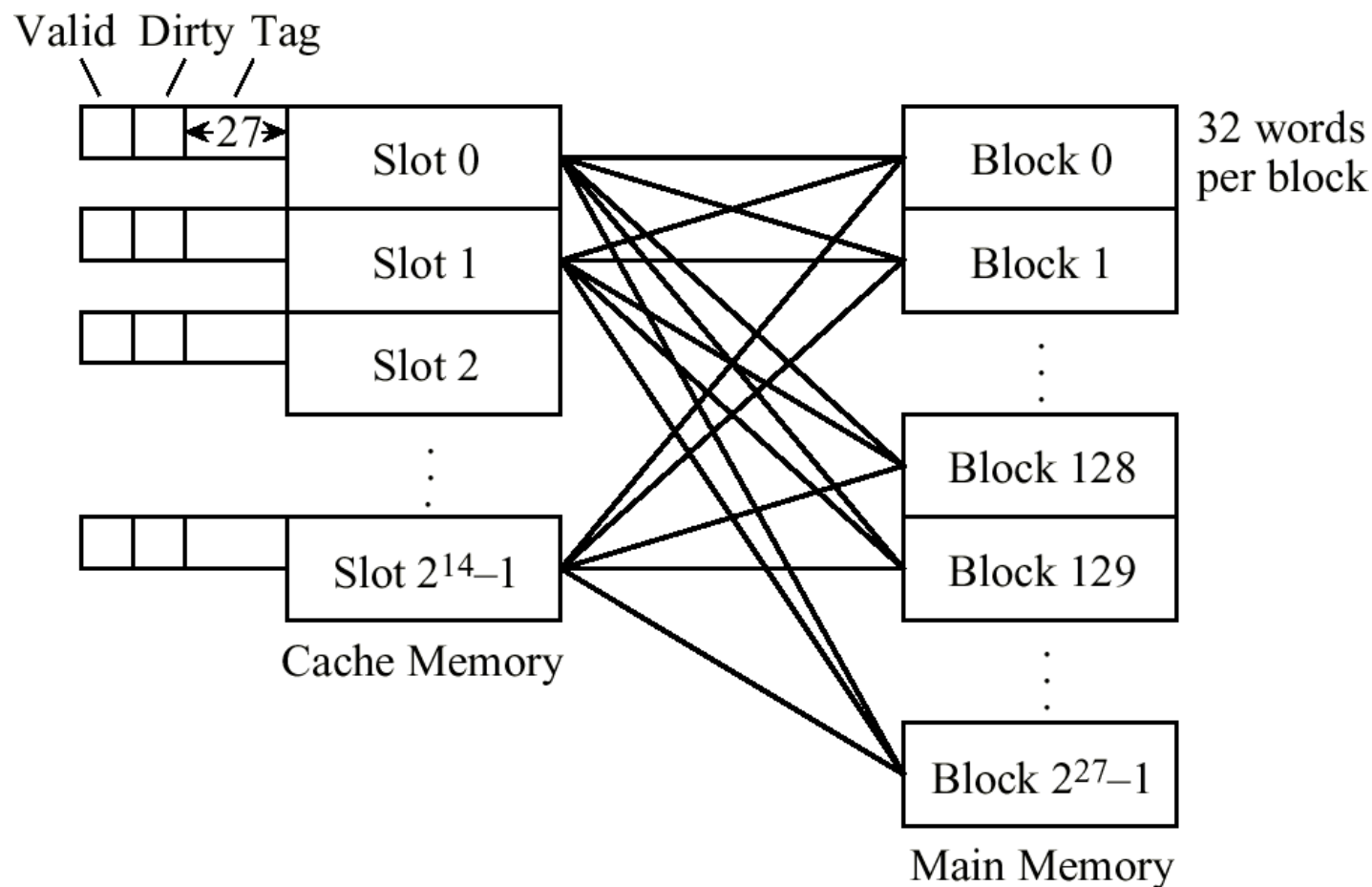
Princípio da Localidade

- **Localidade *temporal***
 - Uma posição de memória referenciada recentemente tem boas chances de ser referenciada novamente
 - iterações e recursividade
- **Localidade *espacial***
 - Uma posição de memória vizinha de uma posição referenciada recentemente tem boas chances de ser referenciada
 - dados tendem a ser armazenados em posições contíguas
- **Idéia: memória cache**
 - pequena mas muito rápida
 - colocada entre a CPU e a memória

Localização da Memória Cache no Sistema Computacional



Esquema de Mapeamento Associativo para Memória Cache



Bits de controle do cache

- **Marca**
 - 27 bits mais significativos do endereço de memória
- **Válido**
 - Indica se a fenda contém ou não um bloco do programa sendo executado
- **Sujo**
 - Indica se o bloco foi ou não modificado enquanto no cache

Operações no cache

- **Acerto (*hit*)**
 - Posição acessada está no cache
- **Erro (*miss*)**
 - Posição acessada ausente do cache
 - Buscada da memória principal
- **Ao carregar o programa**
 - Bits válidos todos zerados
 - Primeira instrução causa um *miss*

Exemplo de Mapeamento Associativo

- Considere como um acesso à posição de memória $(A035F014)_{16}$ é mapeado para o cache, para uma memória de 2^{32} palavras. A memória é dividida em 2^{27} blocos de $2^5 = 32$ palavras por bloco, e o cache consiste de 2^{14} *slots* (fendas):

Tag	Word
27 bits	5 bits

- Se a palavra referenciada está no cache, ela será achada na palavra $(14)_{16}$ de um *slot* que possui a marca $(501AF80)_{16}$, (27 bits mais significativos do endereço). Se a palavra não está no cache, então o bloco correspondente ao campo marca $(501AF80)_{16}$ é trazido da memória principal para um *slot* disponível no cache, e a referência à memória é satisfeita a partir do cache.

Tag	Word
1 0 1 0 0 0 0 0 0 0 1 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0	1 0 1 0 0

Problemas de Implementação

- Qual fenda deve ser liberada quando um novo bloco é lido?
- Pesquisa do cache
 - Campo marca do endereço referenciado deve ser comparado com os 2^{14} campos marca do cache

Políticas de Reposição em Caches Mapeados de Forma Associativa

- Quando não existem *slots* disponíveis para colocar um bloco, uma *política de reposição* é implementada. A política de reposição define que bloco deve ser liberado para dar lugar ao novo bloco.
- Políticas de reposição são usadas para esquemas de mapeamento associativo e associativo por conjunto, assim como para memória virtual.
- Least recently used (LRU)
- First-in/first-out (FIFO)
- Least frequently used (LFU)
- Random (aleatória)
- Ótima (usada para análise – gravar os acessos feitos por um programa, e escolher a melhor estratégia possível para aquela seqüência de referências à memória.)

Políticas de Reposição

- **LRU**
 - Estampilha de tempo associada a cada slot
- **LFU**
 - Contador de frequência associado a cada slot
- **FIFO**
 - Substituição dos slots de forma cíclica (round-robin)
- **Aleatória**
- **Ótima**
 - Usada para análise – gravar os acessos feitos por um programa, e escolher a melhor estratégia possível para aquela seqüência de referências à memória.

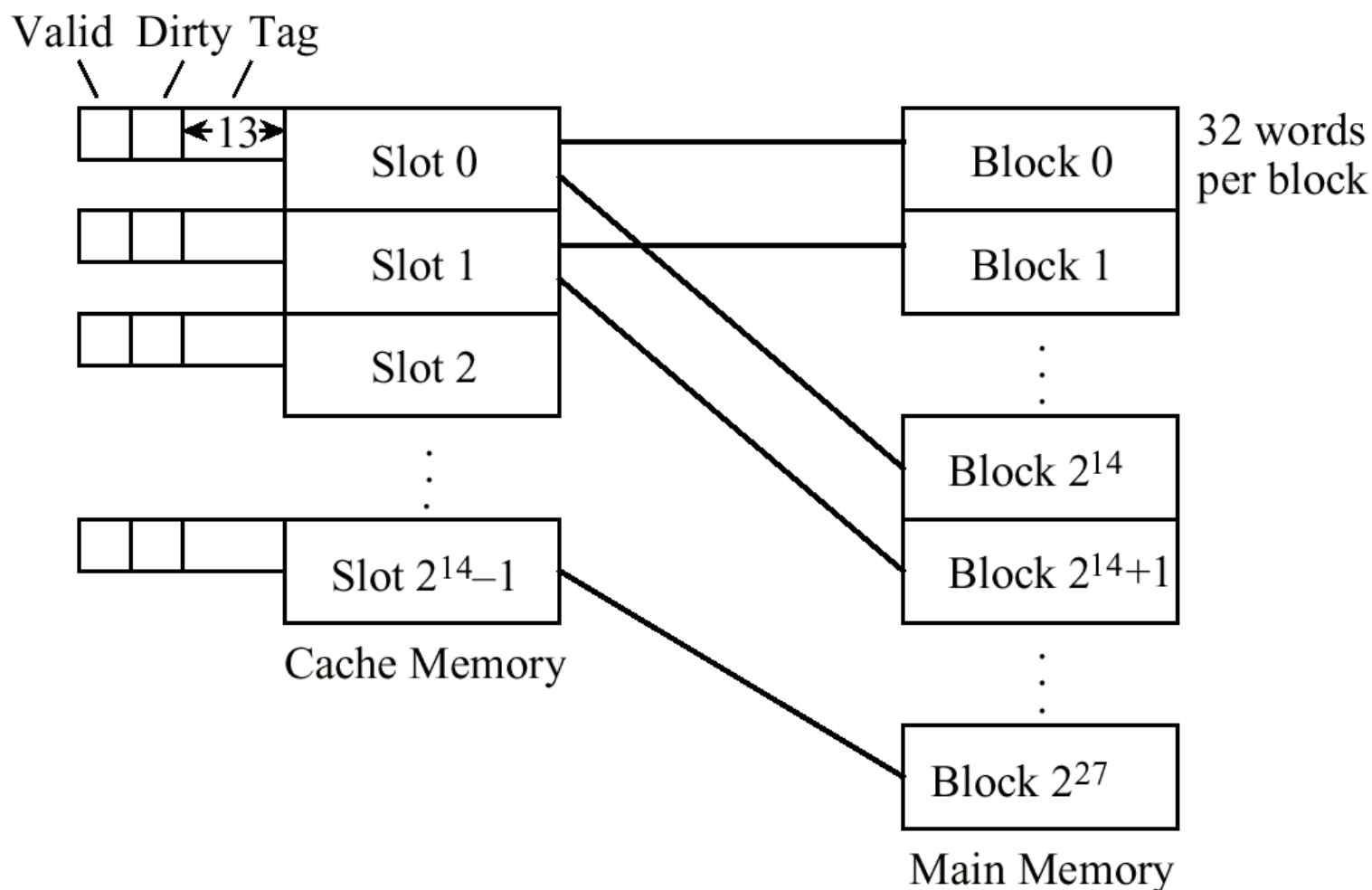
Mapeamento Associativo

- **Vantagens**
 - Qualquer bloco em qualquer fenda
- **Desvantagens**
 - Custo adicional em hardware
 - Marca deve poder ser pesquisada em paralelo
 - Memórias associativas
 - Tamanho da memória de controle do cache:
 - $(27+1+1) \times 2^{14}$ bits

Mapeamento Direto

- Cada fenda do cache corresponde a um conjunto explícito de blocos da memória
- Exemplo de memória de 2^{32} palavras
 - Cada bloco 2^5 palavras
 - Cache com 2^{14} fendas
- 2^{13} blocos da memória principal mapeados em cada fenda
 - Campo de marca de 13 bits

Mapeamento Direto para Memória Cache



Exemplo de Mapeamento Direto

- Em um cache mapeado diretamente, cada bloco da memória principal pode ser mapeado em apenas uma fenda (*slot*), mas cada fenda pode receber mais de um bloco.
- Considere como um acesso à posição de memória $(A035F014)_{16}$ é mapeado no cache para uma memória de 2^{32} palavras. A memória é dividida em 2^{27} blocos de $2^5 = 32$ palavras por bloco, e o cache possui 2^{14} *slots*:

Tag	Slot	Word
13 bits	14 bits	5 bits

- Se a palavra referenciada estiver no cache, ela será encontrada na palavra $(14)_{16}$ do *slot* $(2F80)_{16}$, que possui $(1406)_{16}$ como marca (*tag*).

Tag	Slot	Word
1 0 1 0 0 0 0 0 0 0 1 1 0	1 0 1 1 1 1 1 0 0 0 0 0 0 0	1 0 1 0 0

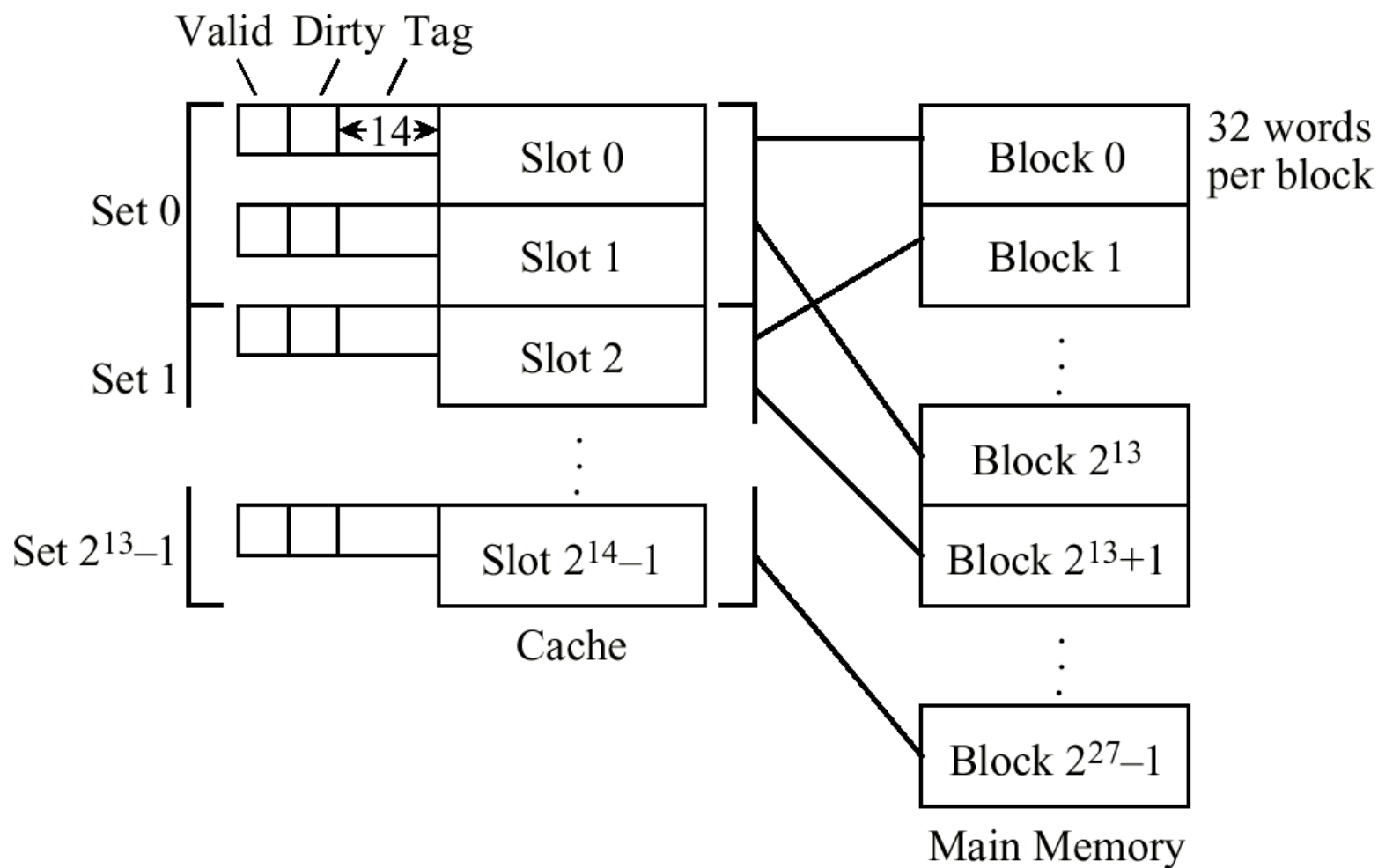
Mapeamento Direto

- **Vantagens**
 - **Memória: 13×2^{14} bits**
 - **Não há busca associativa**
 - **Campo fenda é utilizado**
- **Problema**
 - **Programa referencia posições separadas por 2^{19} palavras (tamanho do cache)**
 - **Cada referência resulta em um *miss***
 - **Somente uma parte do cache disponível será usado...**

Mapeamento Associativo por Conjunto

- Combina o mapeamento associativo e o mapeamento direto
- Idéia básica
 - A porção associativa é limitada a algumas fendas que constituem um conjunto
- Exemplo
 - Cache associativo por conjunto de dois caminhos
 - Dois blocos fazem um conjunto

Mapeamento Associativo por Conjunto para Memória Cache



Exemplo de Mapeamento Associativo por Conjunto

- Considere como um acesso à posição de memória $(A035F014)_{16}$ é mapeado para o cache para uma memória de 2^{32} palavras. A memória é dividida em 2^{27} blocos de $2^5 = 32$ palavras por bloco, existem 2 blocos por conjunto, e o cache possui 2^{14} slots:

Tag	Set	Word
14 bits	13 bits	5 bits

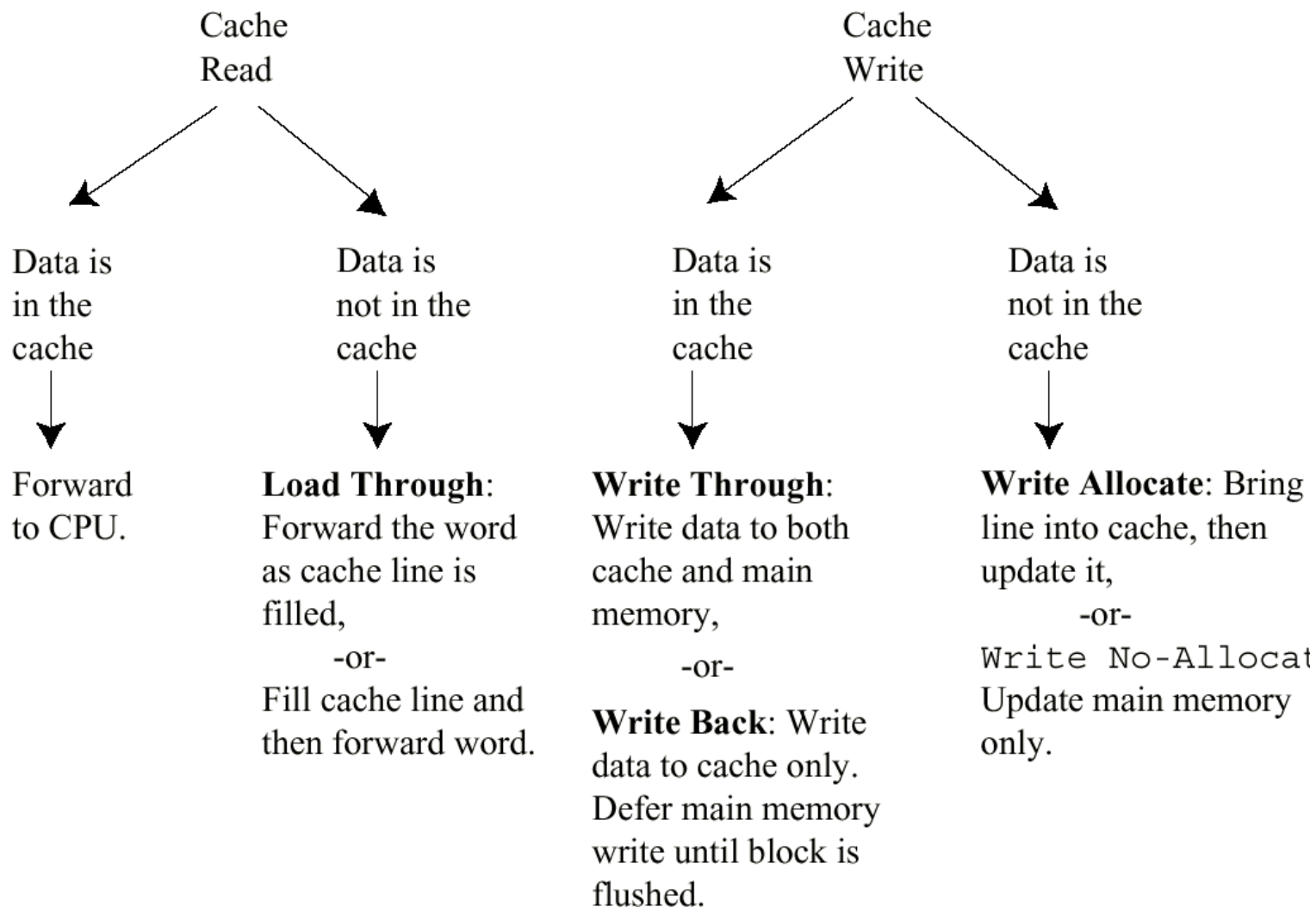
- Os 14 bits mais à esquerda formam o campo marca (*tag*), seguidos por 13 bits do campo conjunto, seguidos por 5 bits do campo palavra:

Tag	Set	Word
1 0 1 0 0 0 0 0 0 0 1 1 0 1	0 1 1 1 1 1 0 0 0 0 0 0 0 0	1 0 1 0 0

Mapeamento Associativo por Conjunto

- **Memória da marca**
 - um pouco maior que no mapeamento direto
 - menos marcas pesquisadas de forma associativa
- **Usado quase que universalmente nos processadores atuais**

Políticas de Leitura e Escrita no Cache



Split Cache

- Caches separados para instruções e dados
- Fendas de instruções
 - Nunca estão “sujas”...
- Além disso
 - Maior parte das operações é leitura (~1 escrita para 4 leituras)
 - Instruções > lidas e nunca escritas
 - Dados > dois operandos 1 resultado

Taxa de Acerto e Tempo de Acesso Efetivo

- Taxa de acerto e tempo de acesso efetivo para um cache de 1 nível:

$$\text{Hit ratio} = \frac{\text{No. times referenced words are in cache}}{\text{Total number of memory accesses}}$$

$$\text{Eff. access time} = \frac{(\# \text{ hits})(\text{Time per hit}) + (\# \text{ misses})(\text{Time per miss})}{\text{Total number of memory access}}$$

- Taxa de acerto e tempo de acesso para cache multi-nível:

$$H_1 = \frac{\text{No. times accessed word is in on-chip cache}}{\text{Total number of memory accesses}}$$

$$H_2 = \frac{\text{No. times accessed word is in off-chip cache}}{\text{No. times accessed word is not in on-chip cache}}$$

$$T_{\text{EFF}} = \frac{(\text{No. on-chip cache hits})(\text{On-chip cache hit time}) + (\text{No. off-chip cache hits})(\text{Off-chip cache hit time}) + (\text{No. off-chip cache misses})(\text{Off-chip cache miss time})}{\text{Total number of memory accesses}}$$

Exemplo de Cache Mapeado Diretamente

- Calcular a taxa de acerto e o tempo de acesso efetivo para um programa que executa das posições de memória de 48 a 95, e então executa um laço 10 vezes, das posições 15 a 31.
- O cache mapeado diretamente possui 4 fendas de 16 palavras, tempo de acerto de 80 ns, e tempo de *miss* de 2500 ns. É usada leitura-através. O cache está inicialmente vazio.

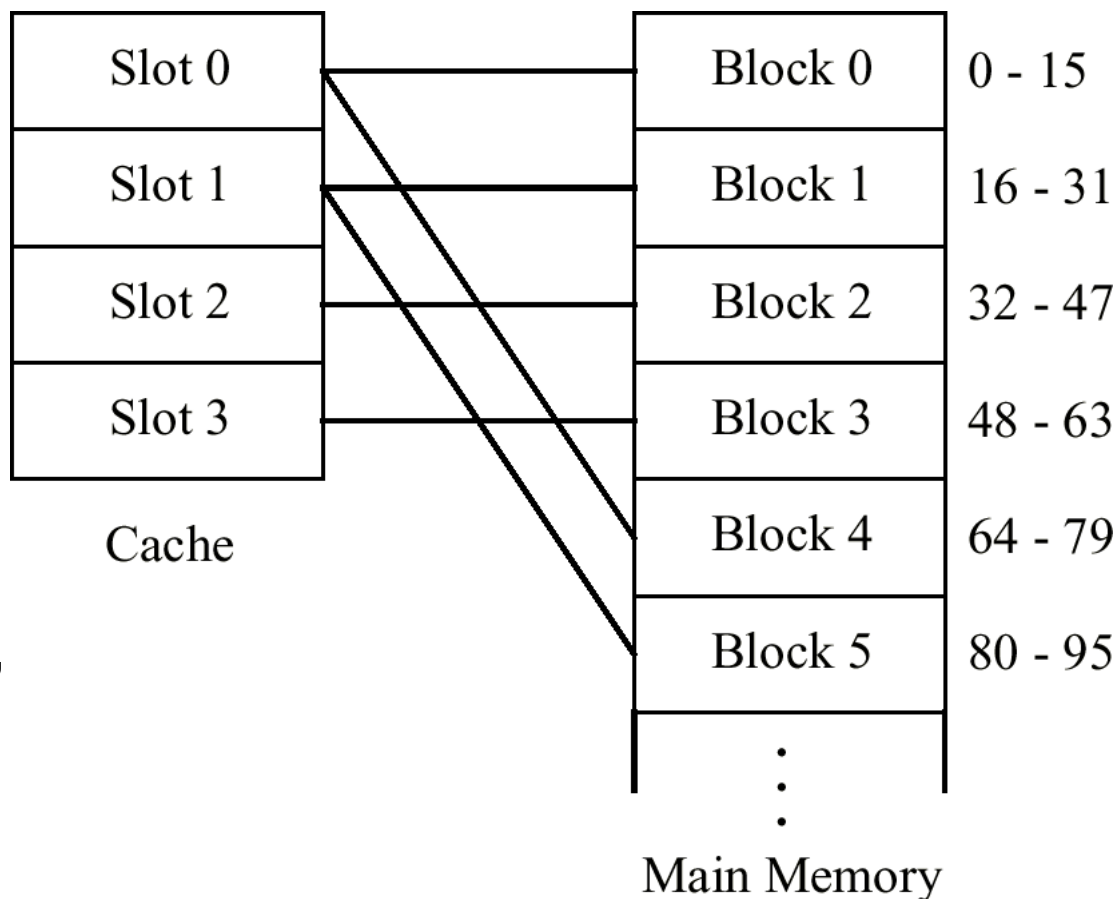


Tabela de Eventos para o Programa Exemplo

Event	Location	Time	Comment
1 miss	48	2500ns	Memory block 3 to cache slot 3
15 hits	49-63	$80\text{ns} \times 15 = 1200\text{ns}$	
1 miss	64	2500ns	Memory block 4 to cache slot 0
15 hits	65-79	$80\text{ns} \times 15 = 1200\text{ns}$	
1 miss	80	2500ns	Memory block 5 to cache slot 1
15 hits	81-95	$80\text{ns} \times 15 = 1200\text{ns}$	
1 miss	15	2500ns	Memory block 0 to cache slot 0
1 miss	16	2500ns	Memory block 1 to cache slot 1
15 hits	17-31	$80\text{ns} \times 15 = 1200\text{ns}$	
9 hits	15	$80\text{ns} \times 9 = 720\text{ns}$	Last nine iterations of loop
144 hits	16-31	$80\text{ns} \times 144 = 12,240\text{ns}$	Last nine iterations of loop
Total hits = 213 Total misses = 5			

Cálculo da Taxa de Acertos e Tempo de Acesso Efetivo para um Programa Exemplo

$$\text{Hit ratio} = \frac{213}{218} = 97.7\%$$

$$\text{EffectiveAccessTime} = \frac{(213)(80ns) + (5)(2500ns)}{218} = 136ns$$

Algoritmo *Neat Little LRU*

- Matriz com 1 linha e 1 coluna para cada fenda
- Início: todas as células em 0
- A cada acesso
 - 1's escritos na linha correspondente
 - 0's escritos na coluna correspondente
- Quando fenda é necessária, linha com mais zeros indica a fenda a liberar

Algoritmo *Neat Little LRU*

- Uma seqüência é mostrada para o algoritmo *Neat Little LRU* para um cache com 4 fendas. Blocos da memória principal são acessados na seqüência : 0, 2, 3, 1, 5, 4.

Cache slot

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Initial

	0	1	2	3
0	0	1	1	1
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Block accesses: 0

	0	1	2	3
0	0	1	0	1
1	0	0	0	0
2	1	1	0	1
3	0	0	0	0

0, 2

	0	1	2	3
0	0	1	0	0
1	0	0	0	0
2	1	1	0	0
3	1	1	1	0

0, 2, 3

	0	1	2	3
0	0	0	0	0
1	1	0	1	1
2	1	0	0	0
3	1	0	1	0

0, 2, 3, 1

	0	1	2	3
0	0	1	1	1
1	0	0	1	1
2	0	0	0	0
3	0	0	1	0

0, 2, 3, 1, 5

	0	1	2	3
0	0	1	0	1
1	0	0	0	1
2	1	1	0	1
3	0	0	0	0

0, 2, 3, 1, 5, 4

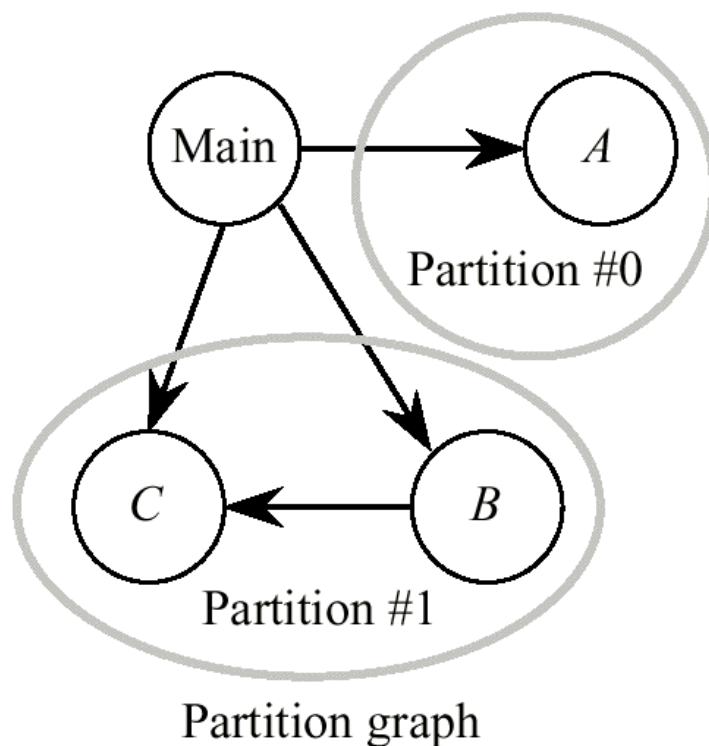
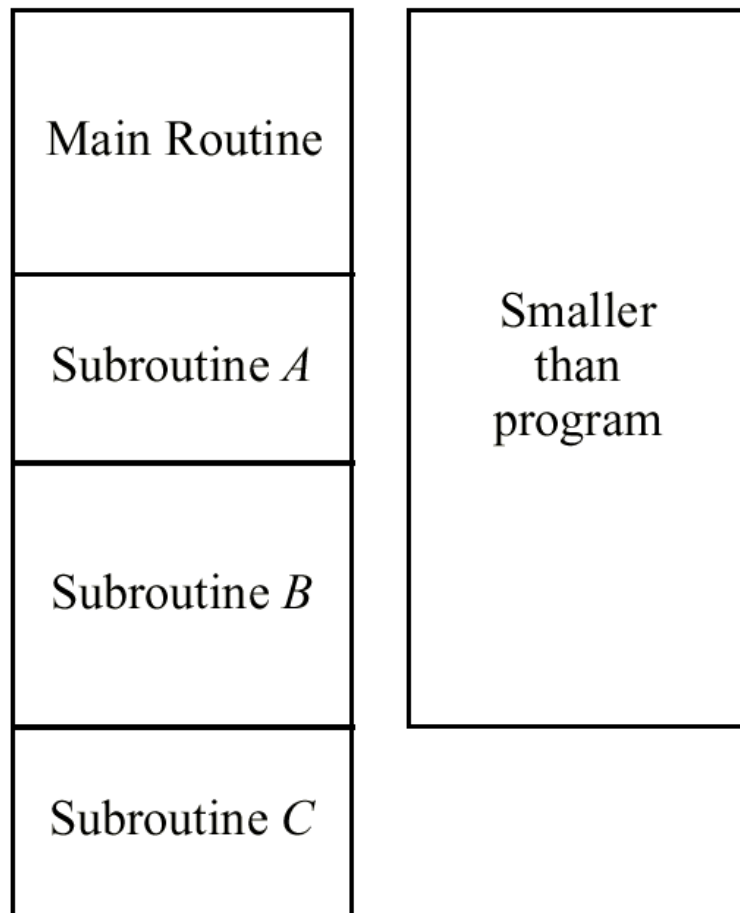
Memória Virtual

- **Problemas**
 - Um programa pode não caber na memória física disponível
 - Primeira solução: *overlays* (sobreposição)
 - Diferentes programas devem compartilhar a memória física
- **Solução: Memória Virtual**

Overlays

- **Grafo de partição para um programa com uma rotina principal e três sub-rotinas:**

Compiled program Physical Memory



Memória Virtual

- **Memória principal pode atuar como um “cache” para o armazenamento secundário (disco)**
 - **Vantagens**
 - **Permitir um compartilhamento eficiente e seguro da memória entre vários programas**
 - **Minimizar os problemas causados aos programas pela existência de uma pequena memória principal**
- **Memória principal só precisa ter as partes ativas dos muitos programas em execução**
- **Deseja-se compilar cada programa no seu próprio espaço de endereços**
 - **Tradução do espaço de endereços para o endereço físico**

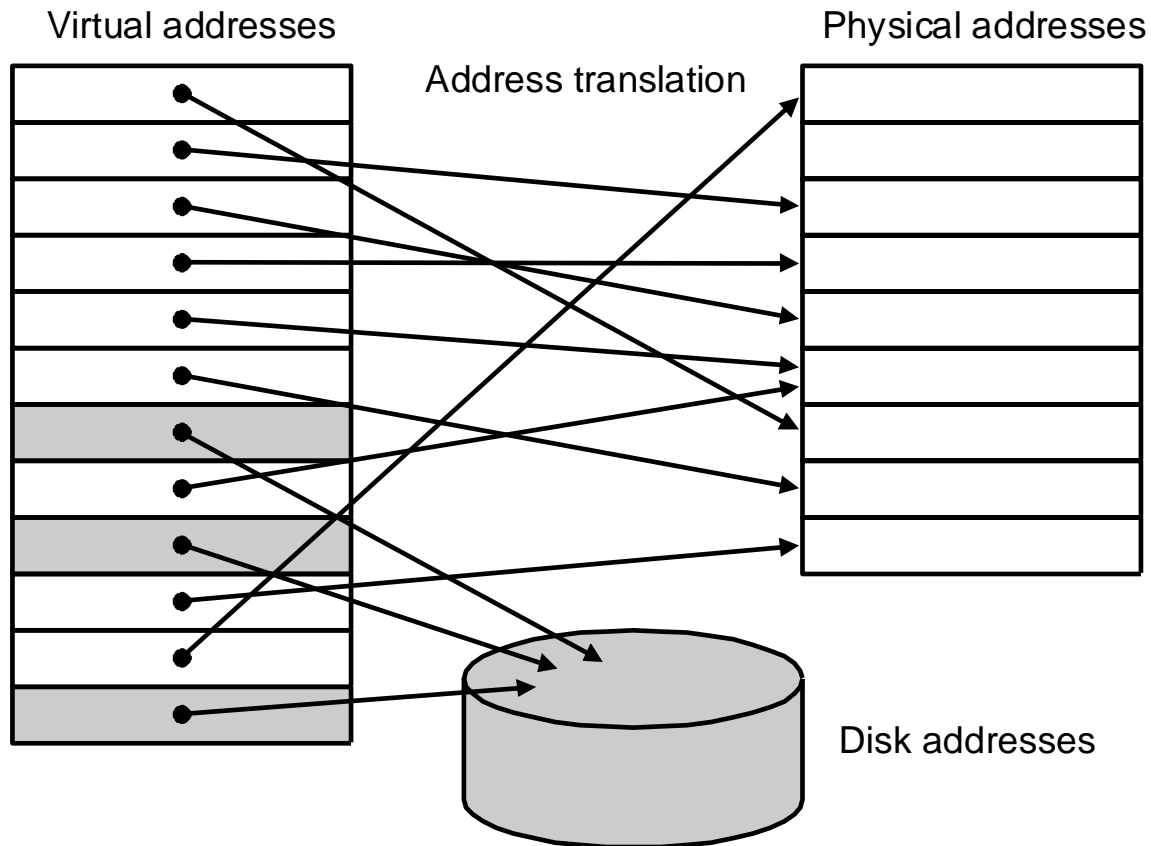
Memória Virtual (cont.)

- Permite a um único programa ultrapassar o tamanho da memória primária
 - Antigamente usavam-se *overlays*
 - Memória virtual automaticamente gerencia a memória principal e a secundária
- Bloco → página
- CPU produz um endereço virtual que é traduzido para um endereço físico (por *hardware* e *software*)
 - Páginas físicas podem ser compartilhadas, logo dois programas diferentes podem compartilhar dados ou código

Memória Virtual (cont.)

Técnica da memória virtual (fonte: Patterson)

(COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED.)



Memória Virtual (cont.)

- Analogia
 - Endereço virtual → título de um livro
 - Endereço físico → localização do livro na biblioteca
- Simplifica a carga dos programas para execução a partir da relocação
 - Relocação mapeia os *endereços virtuais* usados por um determinado programa em *endereços físicos* antes de tais endereços serem usados
 - Carrega-se o programa em qualquer lugar na memória principal
 - Sistemas operacionais só precisam encontrar um número suficiente de páginas (não necessariamente contíguas)

Memória Virtual (cont.)

- Endereço dividido em número de página virtual e deslocamento na página
- Exemplo de mapeamento de um endereço virtual em um endereço físico

Tamanho da página: $2^{12} = 4 \text{ kB}$

Memória principal: $2^{30} = 1 \text{ GB}$

Espaço de endereçamento virtual: $2^{32} = 4 \text{ GB}$

Memória Virtual (cont.)

- A memória virtual é armazenada em uma imagem no disco rígido. A memória física mantém um pequeno número de páginas *virtuais* em quadros de páginas (*page frames*) físicos.
- Um mapeamento entre as memórias física e virtual:

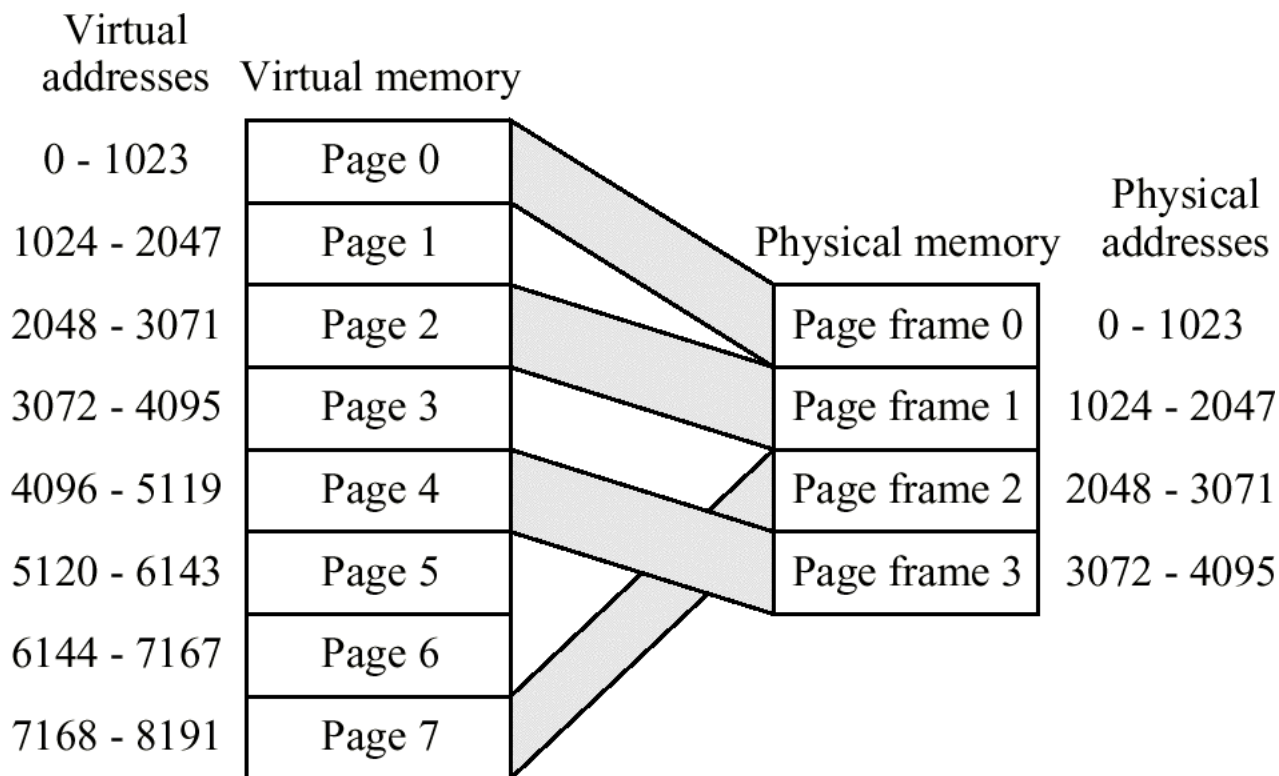


Tabela de Páginas

- A tabela de páginas armazena o mapeamento entre a memória virtual e a memória física.

Page #	Present bit	Page frame	
	Disk address		
0	1	01001011100	00
1	0	11101110010	xx
2	1	10110010111	01
3	0	00001001111	xx
4	1	01011100101	11
5	0	10100111001	xx
6	0	00110101100	xx
7	1	01010001011	10

Present bit:

0: Page is not in
physical memory

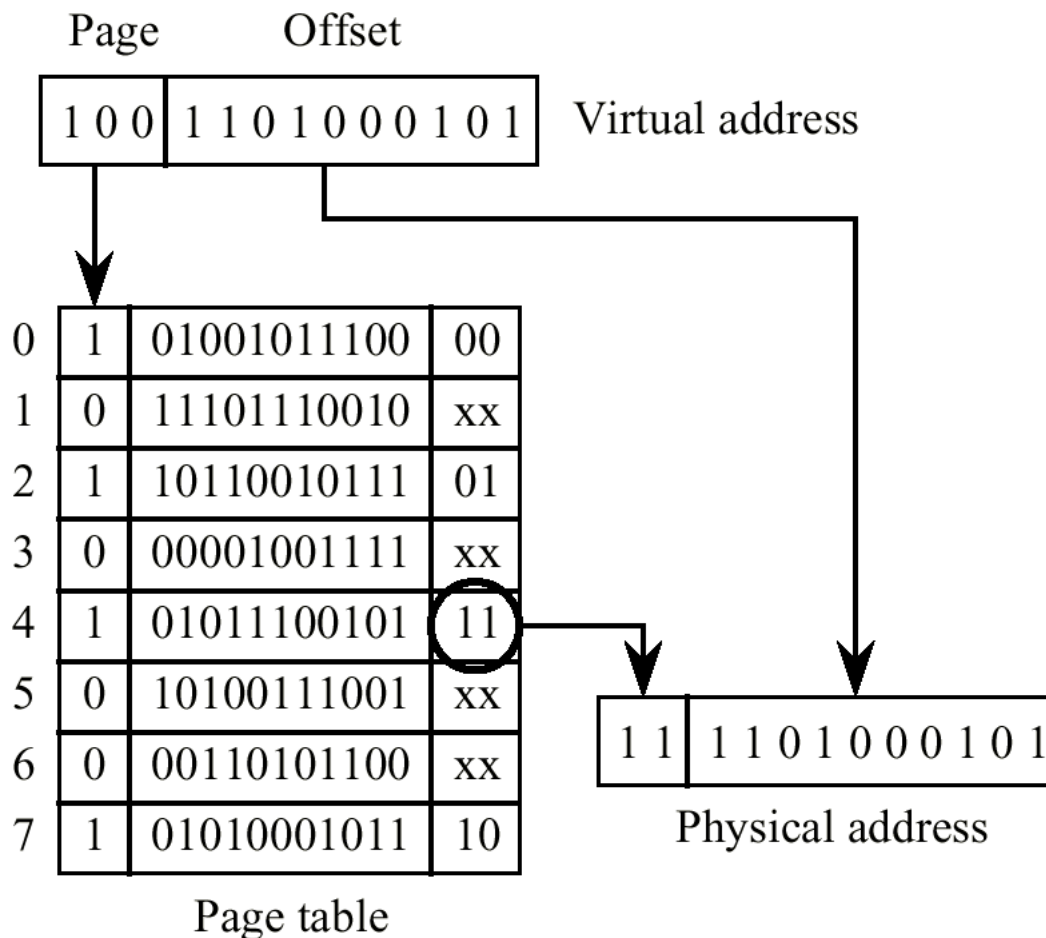
1: Page is in physical
memory

Tabela de Páginas

- Número de linhas = número de páginas *virtuais*
- Endereços no disco
 - Atribuídos pelo S.O. aos blocos do disco
- Campo *frame*
 - Que quadro (página física) contém a página virtual

Usando a Tabela de Páginas

- Um endereço virtual é traduzido em um endereço físico:



Usando a Tabela de Páginas (cont.)

- A configuração da tabela de páginas se modifica com a execução do programa.
- Inicialmente, a tabela de páginas está vazia. Na configuração final, quatro páginas estão na memória física.

0	0	01001011100	xx
1	1	11101110010	00
2	0	10110010111	xx
3	0	00001001111	xx
4	0	01011100101	xx
5	0	10100111001	xx
6	0	00110101100	xx
7	0	01010001011	xx

After
fault on
page #1

0	0	01001011100	xx
1	1	11101110010	00
2	1	10110010111	01
3	0	00001001111	xx
4	0	01011100101	xx
5	0	10100111001	xx
6	0	00110101100	xx
7	0	01010001011	xx

0	0	01001011100	xx
1	1	11101110010	00
2	1	10110010111	01
3	1	00001001111	10
4	0	01011100101	xx
5	0	10100111001	xx
6	0	00110101100	xx
7	0	01010001011	xx

After
fault on
page #3

0	0	01001011100	xx
1	0	11101110010	xx
2	1	10110010111	01
3	1	00001001111	10
4	1	01011100101	11
5	1	10100111001	00
6	0	00110101100	xx
7	0	01010001011	xx

Memória Virtual (cont.)

- **Falta de página:** dado não está na memória, recuperá-lo do disco (consome milhões de ciclos de clock)
 - Usa-se páginas grandes para amortizar o tempo de acesso (16 kB por ex.)
 - Pode ser tratada por *software* pois o tempo de acesso ao disco é enorme
 - Utiliza-se algoritmos mais eficientes para a escolha de como colocar as páginas
 - Usa-se a contra-escrita (*write-back*) pois as escritas no disco demoram muito

Memória Virtual (cont.)

- **Esquema de associatividade total para colocar blocos na memória principal**
 - **Páginas localizadas por uma tabela de páginas que indexa totalmente a memória principal**
 - Tabela está na memória principal, é indexada com o número da página virtual e contém o respectivo número de página física
 - Cada programa possui a sua tabela de páginas
 - Tabela pode conter entradas para páginas que não estão na memória principal
 - ***Hardware* possui um *registrador da tabela de páginas* usado para apontar o endereço inicial da tabela na memória principal**

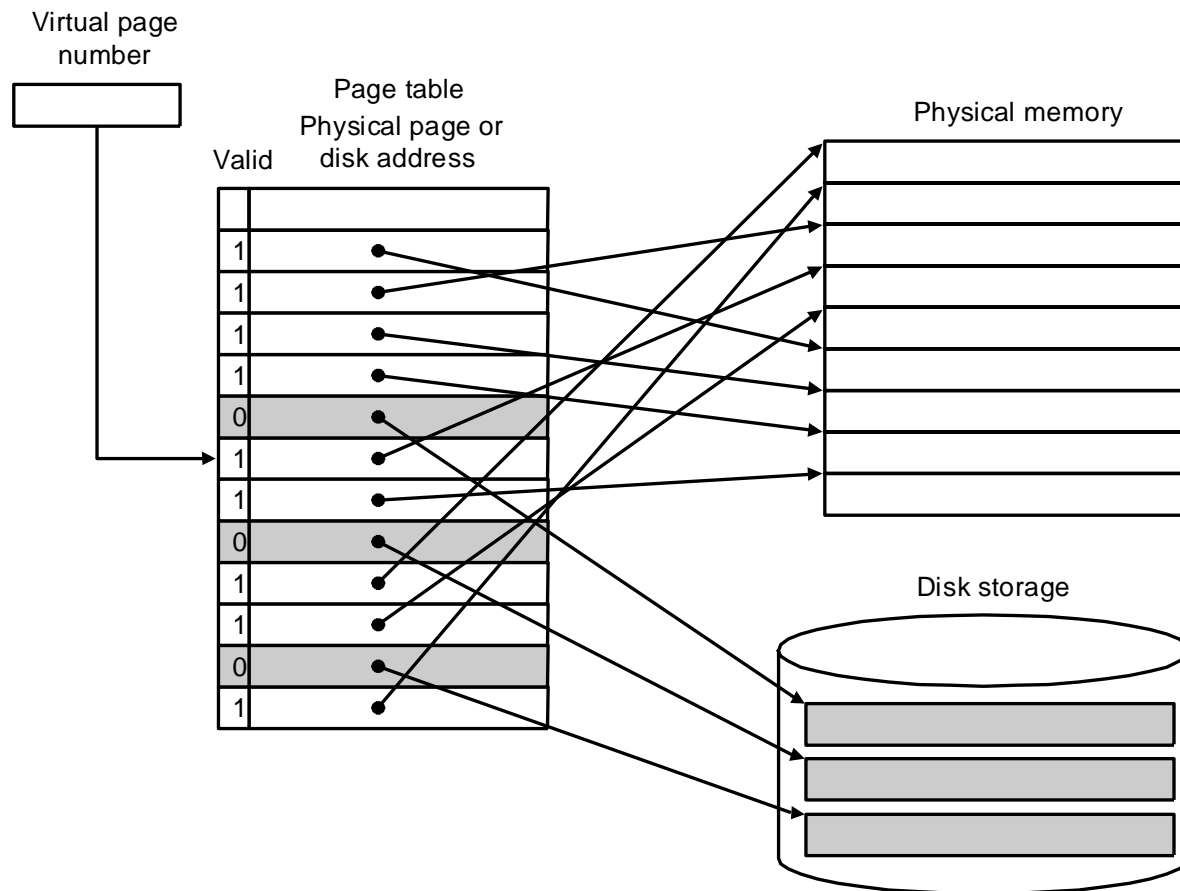
Memória Virtual (cont.)

- **Esquema de associatividade total para colocar blocos na memória principal**
 - **Bit de residência**
 - **Se 0, a página não está na memória principal**
 - **Rótulos não são necessários pois a tabela de páginas contém um mapeamento para cada possível página virtual**

Tabela de Páginas Única

Para endereço físico e endereço em disco (fonte: Patterson)

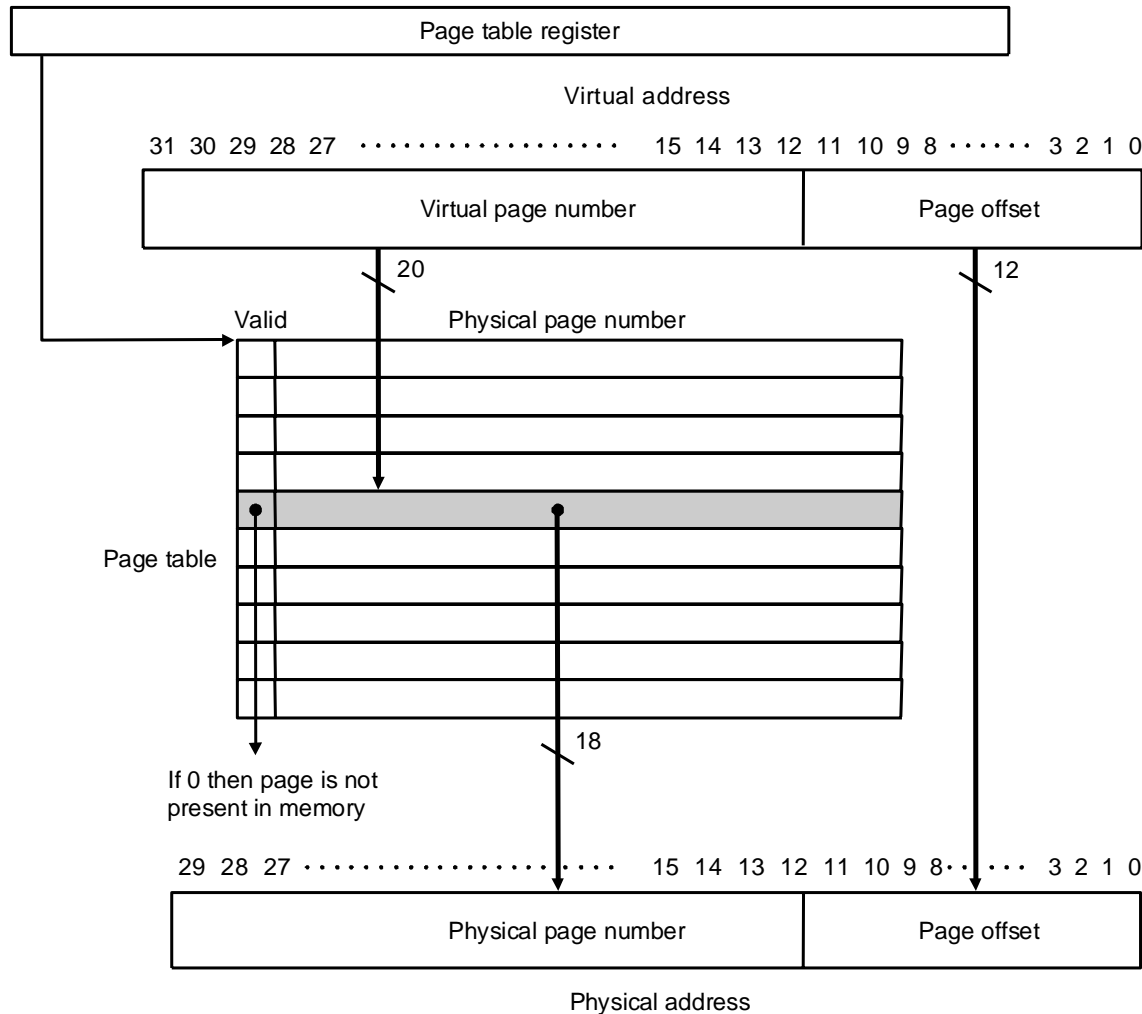
(COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED.)



Exemplo de Memória Virtual

(fonte: Patterson)

(COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED.)



Detalhes da Paginação

- Estado de um programa é especificado através da tabela de páginas, do PC e dos registradores
- Falta de página
 - Se o bit de residência é 0
 - Sistema operacional usa o mecanismo de exceção para transferir a página
 - Espaço em disco para todas as páginas virtuais de um processo
 - Estrutura de dados para controlar onde cada página virtual está armazenada no disco
 - Estrutura de dados para controlar o espaço físico pelos diversos processos
 - Substituição de página com LRU (*Less Recently Used*)
 - Bit de uso ou de referência

Acesso às Páginas

- **Escrita das páginas**
 - **Usa-se contra escrita: páginas são escritas na memória principal e copiadas para o disco no momento em que forem substituídas**
 - **Bit de modificação na tabela de páginas setado na primeira vez em que a página foi escrita (modificada)**
- **Cada acesso à memória gera dois acessos**
 - **Um para obter o endereço físico (da tabela de páginas)**
 - **Outro para obter a informação (da memória física)**

Buffer de Previsão da Tradução (TLB - *Translation Lookaside Buffer*)

- Para melhorar o desempenho, deve-se apostar nas localidades temporal e espacial das referências à tabela de páginas
- TLB (*Translation-Lookaside Buffer*): cache que armazena as traduções de endereços mais recentes
 - Na analogia, é um pedaço de papel onde escreve-se a posição de um conjunto de livros
 - Rótulo de cada entrada do TLB guarda uma parte do número da página virtual e no campo informação daquela entrada é guardado o número do endereço físico
 - Inclui os bits de residência e de modificação

Exemplo de TLB

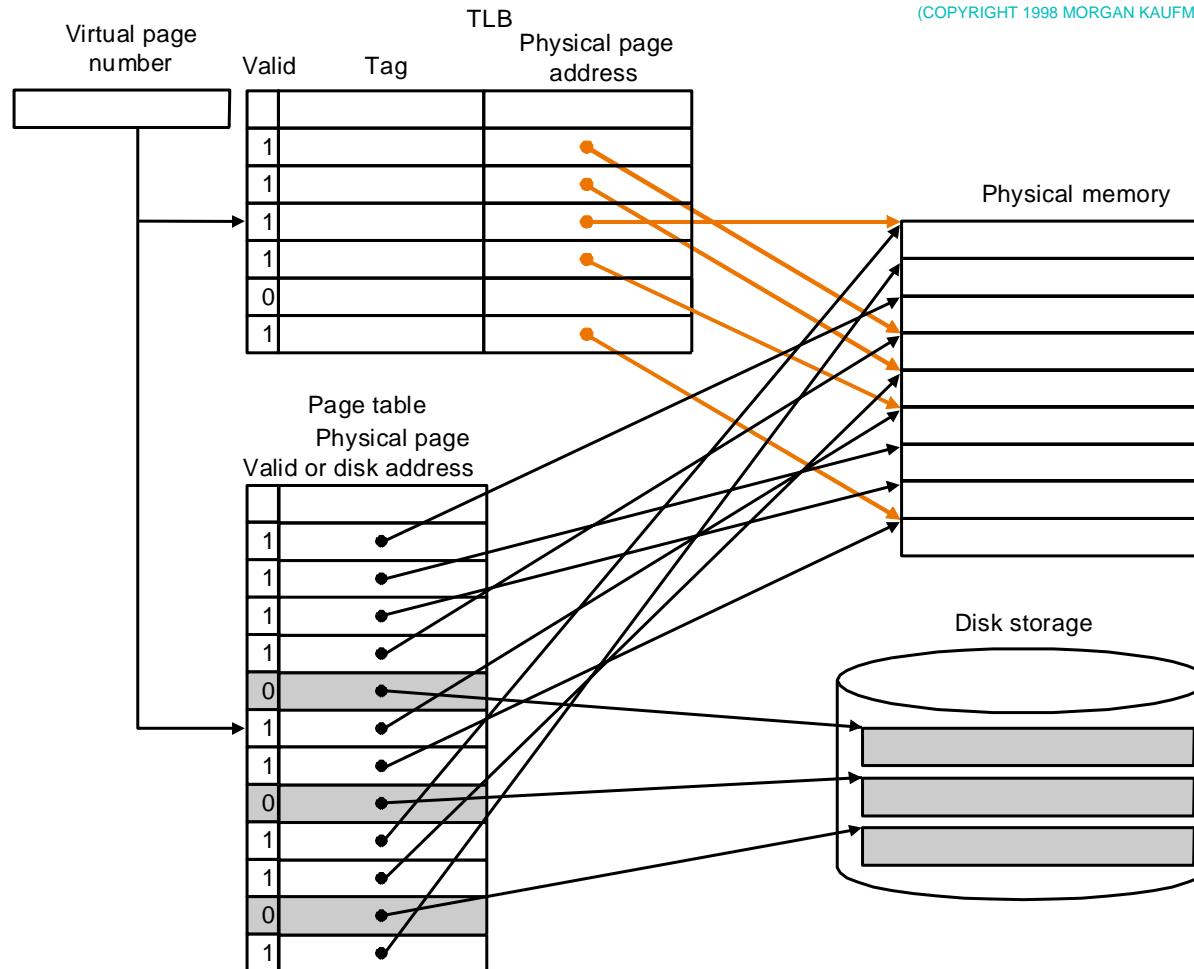
- Um TLB que mantém 8 entradas para um sistema com 32 páginas virtuais e 16 quadros de página (*page frames*).

Valid	Virtual page number	Physical page number
1	0 1 0 0 1	1 1 0 0
1	1 0 1 1 1	1 0 0 1
0	- - - - -	- - - -
0	- - - - -	- - - -
1	0 1 1 1 0	0 0 0 0
0	- - - - -	- - - -
1	0 0 1 1 0	0 1 1 1
0	- - - - -	- - - -

TLB (cont.)

(fonte: Patterson)

(COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED.)

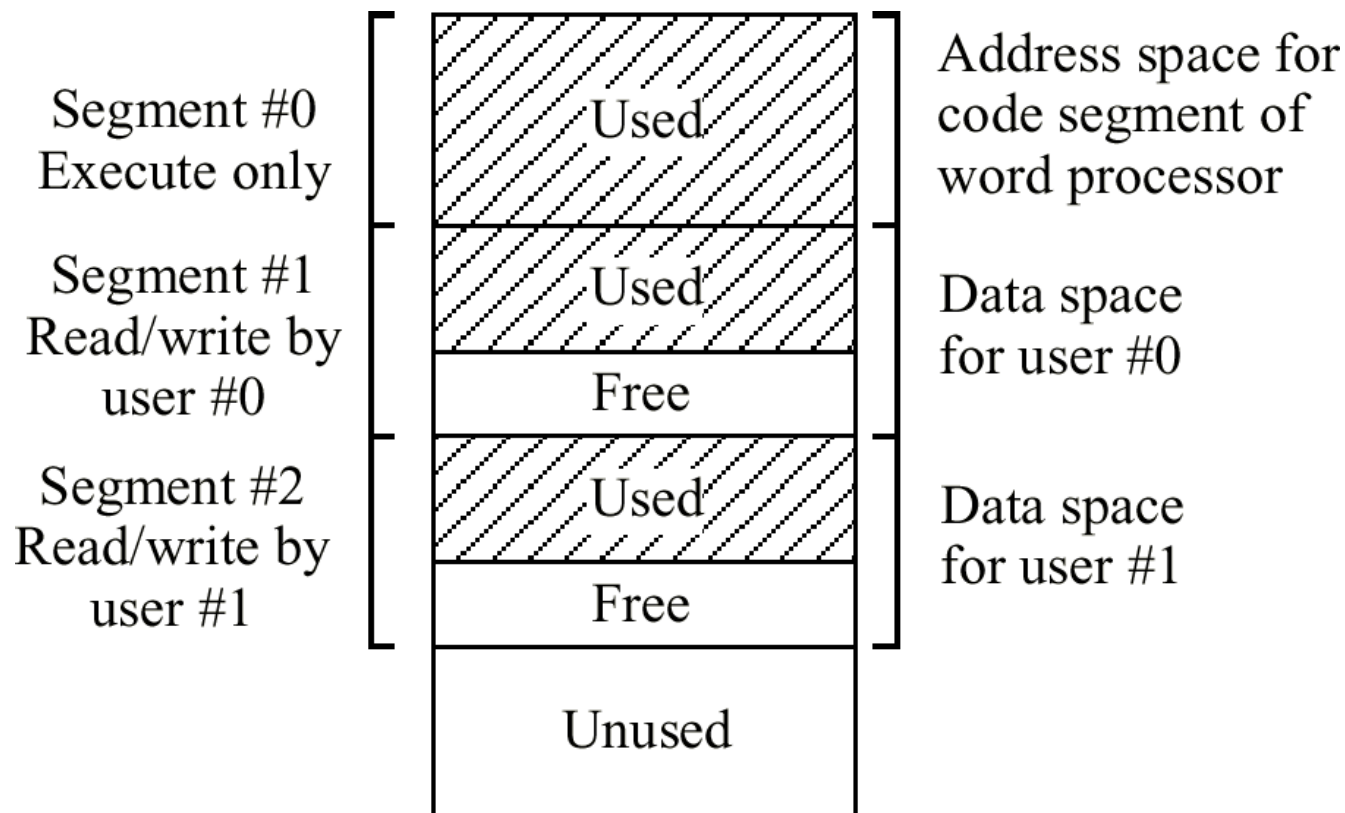


TLB (*cont.*)

- **TLB**
 - Falta devido a uma falta no acesso ao TLB ou a uma falta de página
 - Se a página existir na memória principal, o processador coloca no TLB as informações necessárias para efetuar a tradução, tentando usar o TLB novamente
 - Se a página não existir na memória, a falta no TLB indica uma falta de página
 - Usa-se contra escrita (*write-back*)

Segmentação

- A segmentação da memória permite *por exemplo* a dois usuários *compartilhar* o mesmo código de um processador de textos, com diferentes espaços de dados:



Segmentação

- **Programas possuem regiões de memória que podem variar**
 - **Ex. compilador**
 - **código-fonte, tabela de constantes, pilha...**
- **Segmentos**
 - **Espaços de endereçamento virtual distintos**
 - **Facilitam o gerenciamento destas regiões**
 - **(aumento e diminuição de tabelas)**

Segmentos

- **Corresponde a um espaço de endereçamento virtual separado**
- **Podem crescer e diminuir sem afetar uns aos outros**
- **O programador deve especificar endereços na memória de duas dimensões**
 - **“número” do segmento**
 - **endereço dentro do segmento**

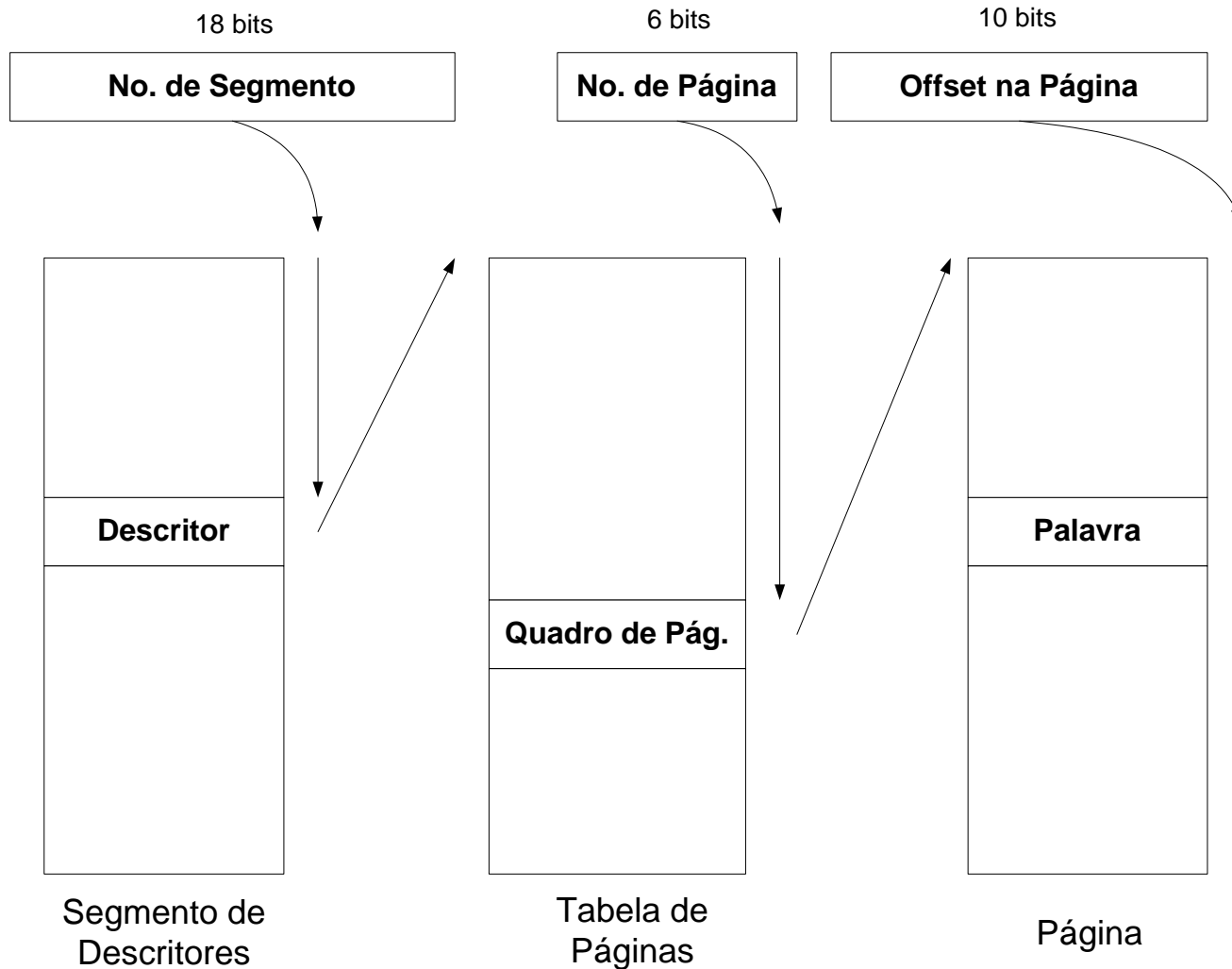
Vantagens da Segmentação

- **Facilidade de programação**
- **Modularidade (compilação)**
- **Proteção**
- **Compartilhamento**

	Paginação	Segmentação
Programador precisa estar consciente da técnica?	Não	Sim
Quantos espaços de endereçamento linear?	1	Vários
O espaço de endereçamento total pode exceder a memória física?	Sim	Sim
Os procedimentos e dados podem ser diferenciados e protegidos separadamente?	Não	Sim
Tabelas de tamanho variável suportadas facilmente?	Não	Sim
Compartilhamento de funções facilitado?	Não	Sim
Para que a técnica foi criada?	Utilizar espaço de endereçamento maior sem ter que acrescentar memória física	Permitir que programas e dados possam ser divididos em espaços de endereçamento lógico diferentes, facilitando o compartilhamento e a proteção

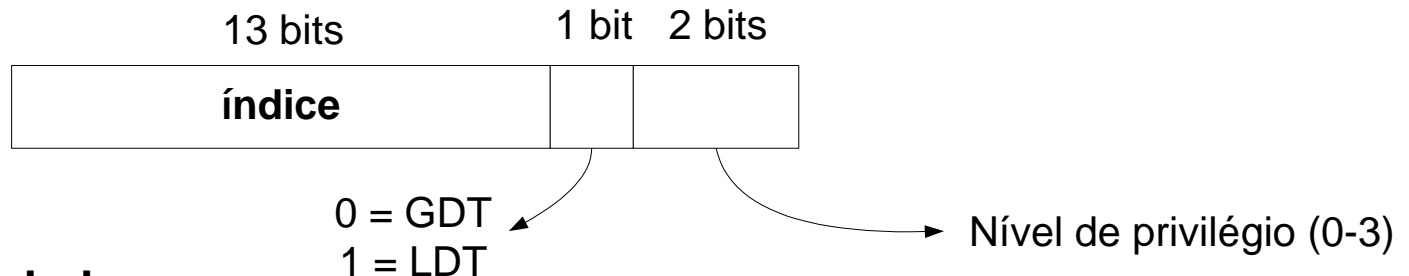
Exemplo de Segmentação - MULTICS

Endereço virtual de 34 bits



Segmentação - 80386

- 16k segmentos, cada um contendo até 1 bilhão de palavras de 32 bits
- Acesso ao segmento
 - Primeiro, um seletor é carregado em um dos (6) registradores de segmento da CPU
 - Seletor (16 bits)



- Há duas tabelas
 - GDT – *Global Descriptor Table*
 - Única – descreve segmentos do sistema
 - LDT – *Local Descriptor Table*
 - 1 por processo – segmentos locais

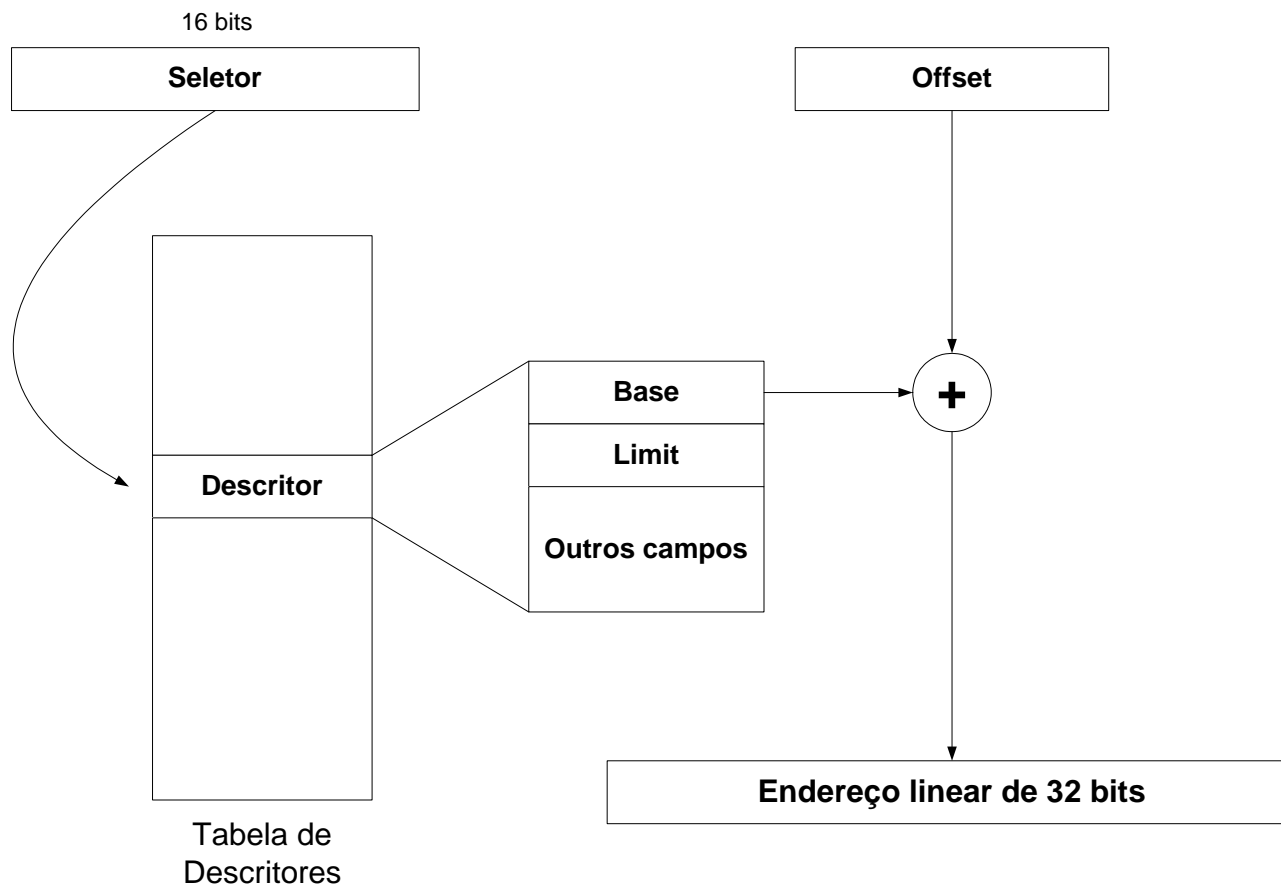
Segmentação – 80386 (cont.)

- Reg. CS – armazena o seletor pro segmento de código
- Reg. DS – armazena o seletor pro segmento de dados
- No seletor, o *índice* indica a entrada na GDT ou LDT
 - 13 bits – 8k descritores de segmento
- No momento em que o *seletor* é carregado no registrador de segmento, a CPU busca o *descritor* correspondente na LDT (GDT)

Segmentação – 80386 (cont.)

- Cada descritor possui 8 bytes que incluem
 - **Base** – 32 bits – início do segmento
 - **Limit** – 20 bits – tamanho do segmento
(deveria ser 32 bits, usado, então, em conjunto com **G**)
 - **G = 0** – **Limit** em bytes; **G = 1** - **Limit** em páginas (de tam. 4k)
 - **P** – segmento presente/ausente
 - **DPL** – nível de privilégio
 - **Type** – tipo de segmento e proteção

Segmentação 80386

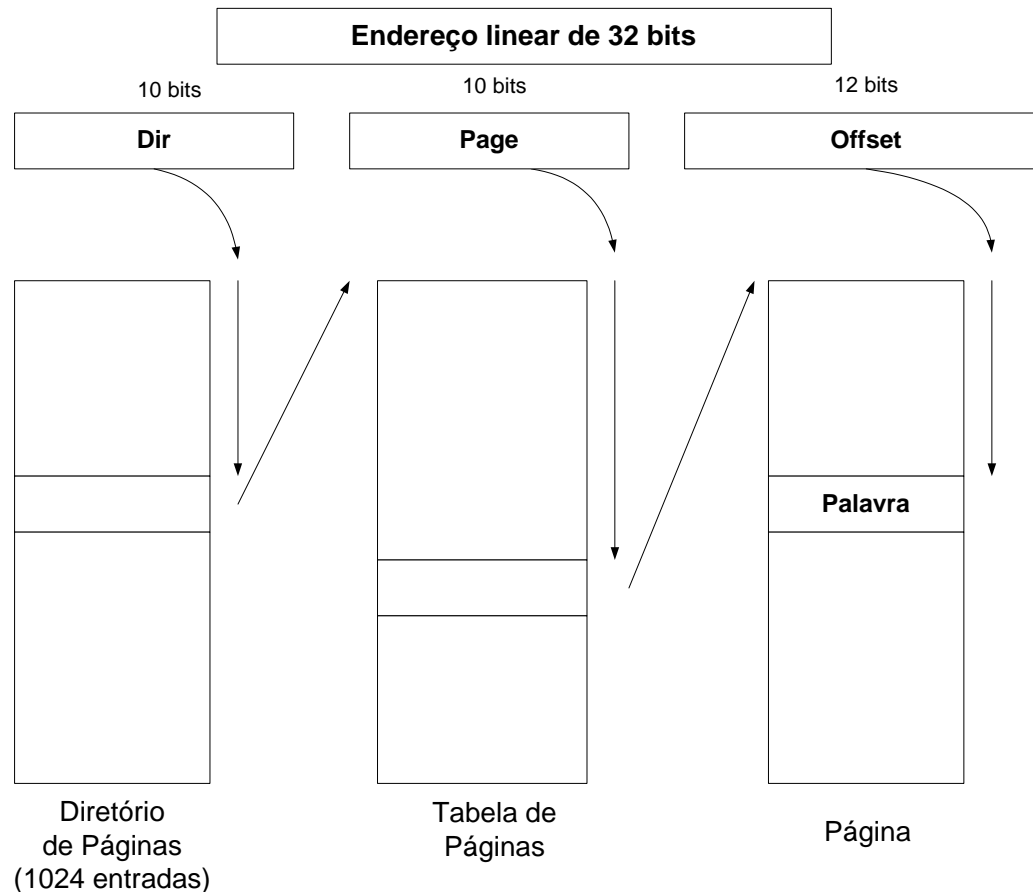


- 32 bits de *Base* somados ao *Offset* dado pela instrução formam *endereço linear de 32 bits*
- Este é o endereço físico, se não houver paginação

Paginação no 80386

- Espaço de endereçamento virtual de 32 bits
- Páginas de 4k
 - Significaria Tabela de Páginas com 1 milhão de entradas...
- Paginação feita em 2 níveis
 - Endereço linear dividido em 3 campos
- Entradas na Tabela de Páginas – 32 bits
 - Dos quais 20 bits contêm o no. do quadro de página
 - Outros 12 bits de controle usados pelo S.O.

Paginação no 80386 (cont.)



- Cada tabela de página corresponde a 4M de memória física
- Segmento < 4M \Rightarrow 1 entrada no diretório de páginas
 - Overhead para segmentos curtos = 2 entradas em vez de 1 milhão

Segmentação x Paginação no 80386

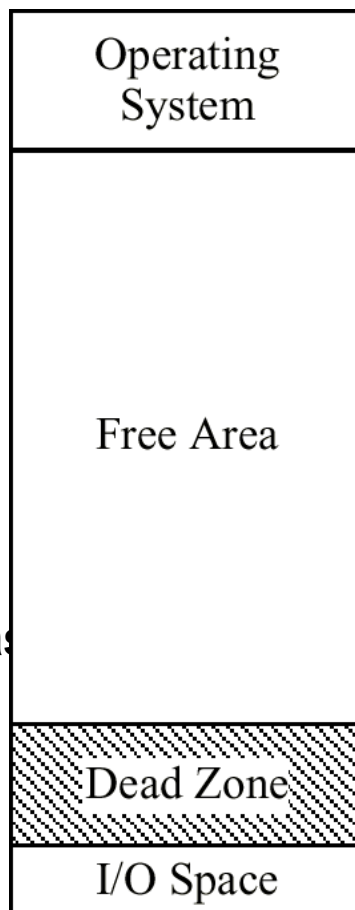
- **Segmentação pura**
 - Paginação pode ser desabilitada por um bit em um registrador de controle global
- **Paginação pura**
 - Todos os registradores de segmento apontam para o mesmo seletor:
 - cuja Base é 0
 - cujo Limit é o tamanho máximo

Fragmentação

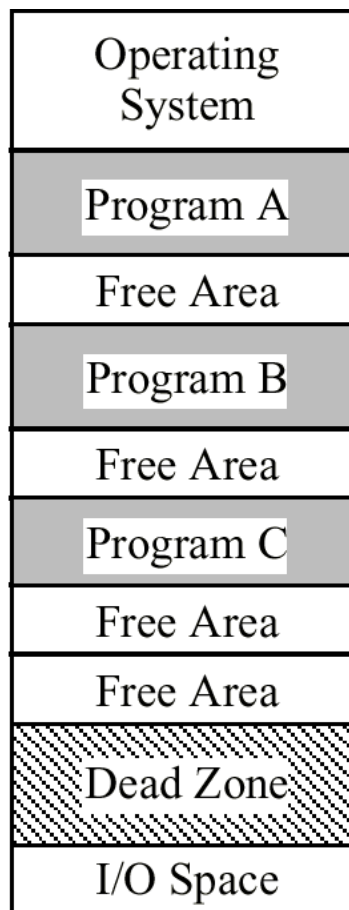
(a) Área de memória livre após inicialização;

(b) depois da fragmentação;

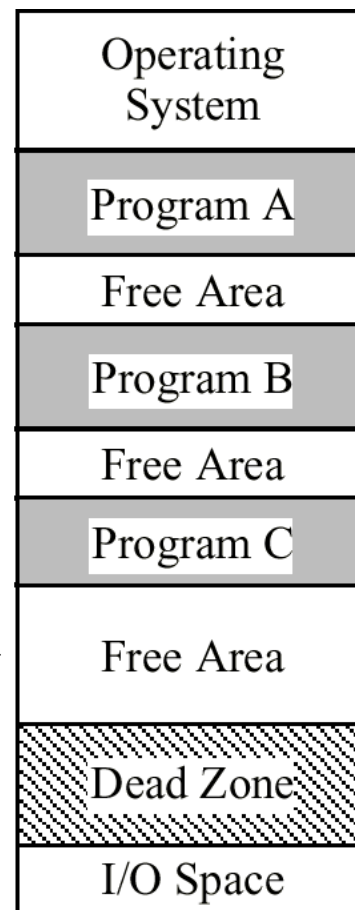
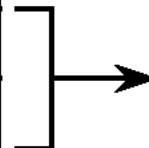
(c) depois da fusão de 2 áreas livres.



(a)



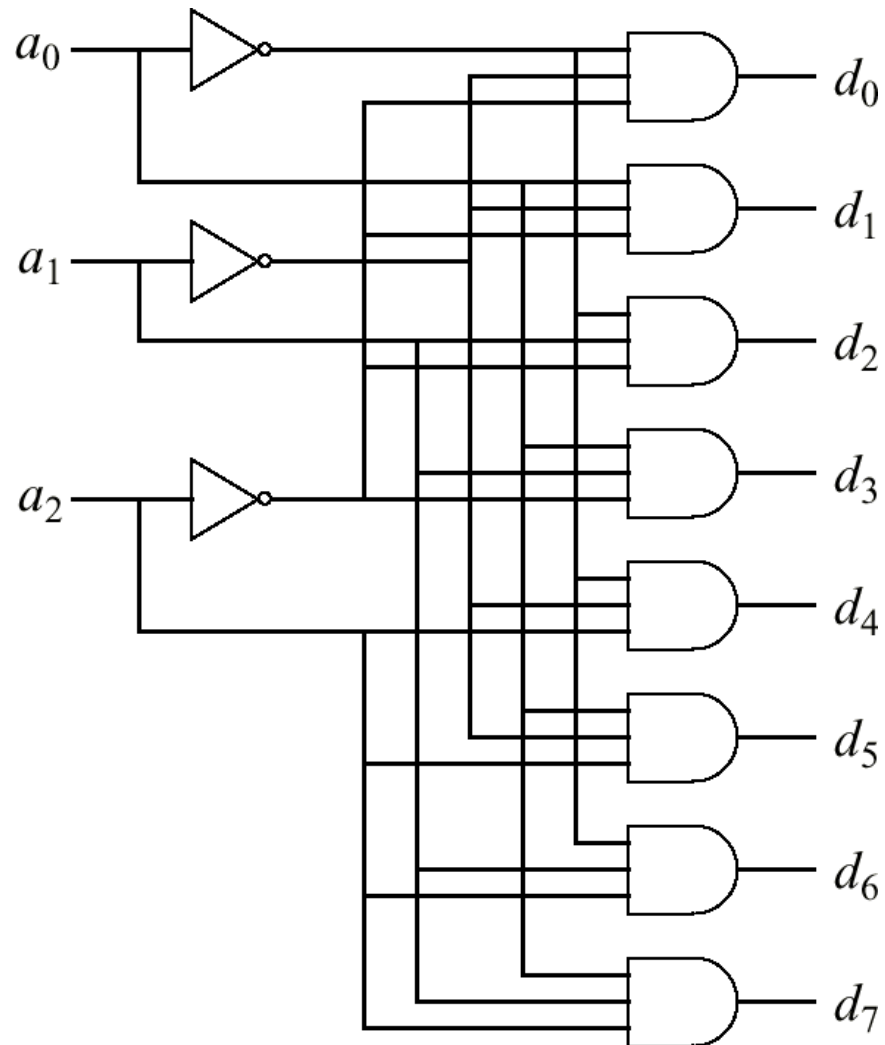
(b)



(c)

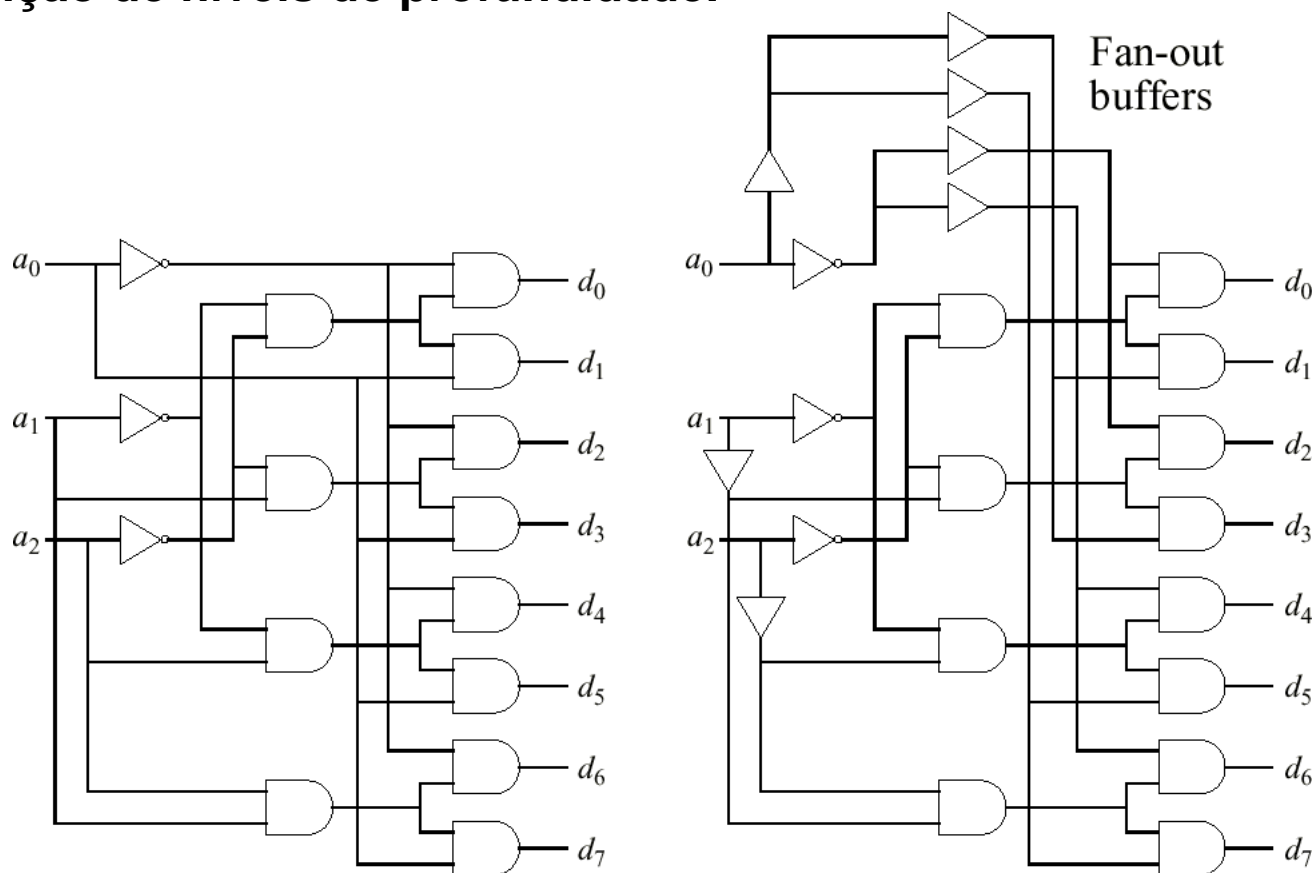
Decodificador de 3 Variáveis

- Um decodificador convencional não escala bem para tamanhos maiores porque cada linha de endereço controla o dobro de portas lógicas para cada linha de endereço adicionada.
- Cada porta AND: fan-in = N
- Cada linha de endereço: fan-out = 2^N
- Profundidade = 2 portas



Decodificador em Árvore

- Um decodificador em árvore é mais facilmente estendido para grandes tamanhos porque o fan-in e o fan-out são tratados pela adição de níveis de profundidade.

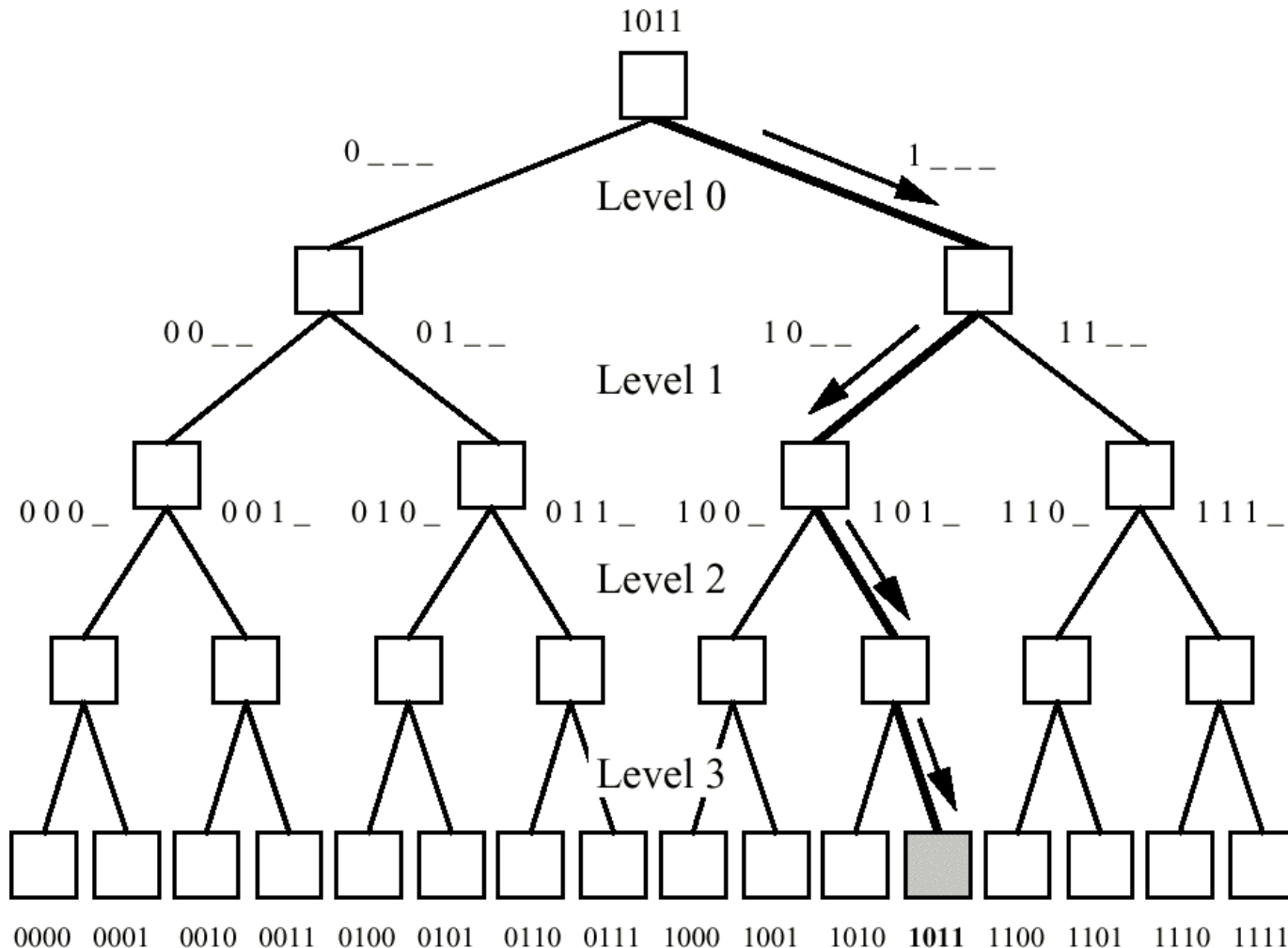


(a)

(b)

Decodificação em Árvore – Um Nível de Cada Vez


- Uma árvore de decodificação para uma RAM de 16 palavras:



Memória Endereçável por Conteúdo – Endereçamento

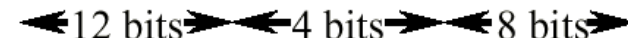
- Relações entre a memória de acesso aleatório e a memória endereçável por conteúdo:

Address	Value
0000A000	0F0F0000
0000A004	186734F1
0000A008	0F000000
0000A00C	FE681022
0000A010	3152467C
0000A014	C3450917
0000A018	00392B11
0000A01C	10034561



Random access memory

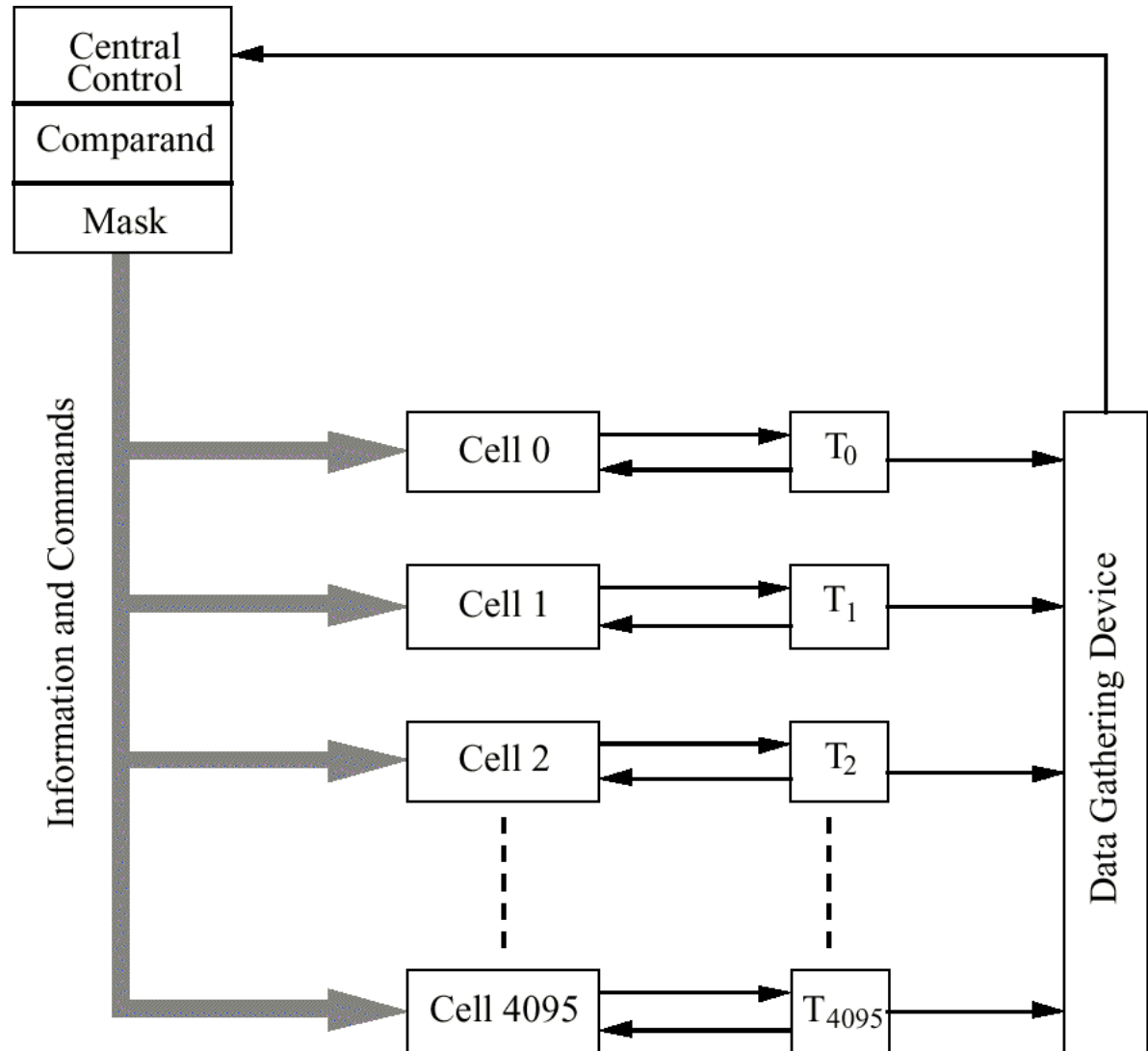
Field1	Field2	Field3
000	A	9E
011	0	F0
149	7	01
091	4	00
000	E	FE
749	C	6E
000	0	50
575	1	84



Content addressable memory

Visão Geral da CAM

- Fonte: (Foster, C. C., *Content Addressable Parallel Processors*, Van Nostrand Reinhold Company, 1976.)



Sub-árvores de Endereçamento para uma CAM

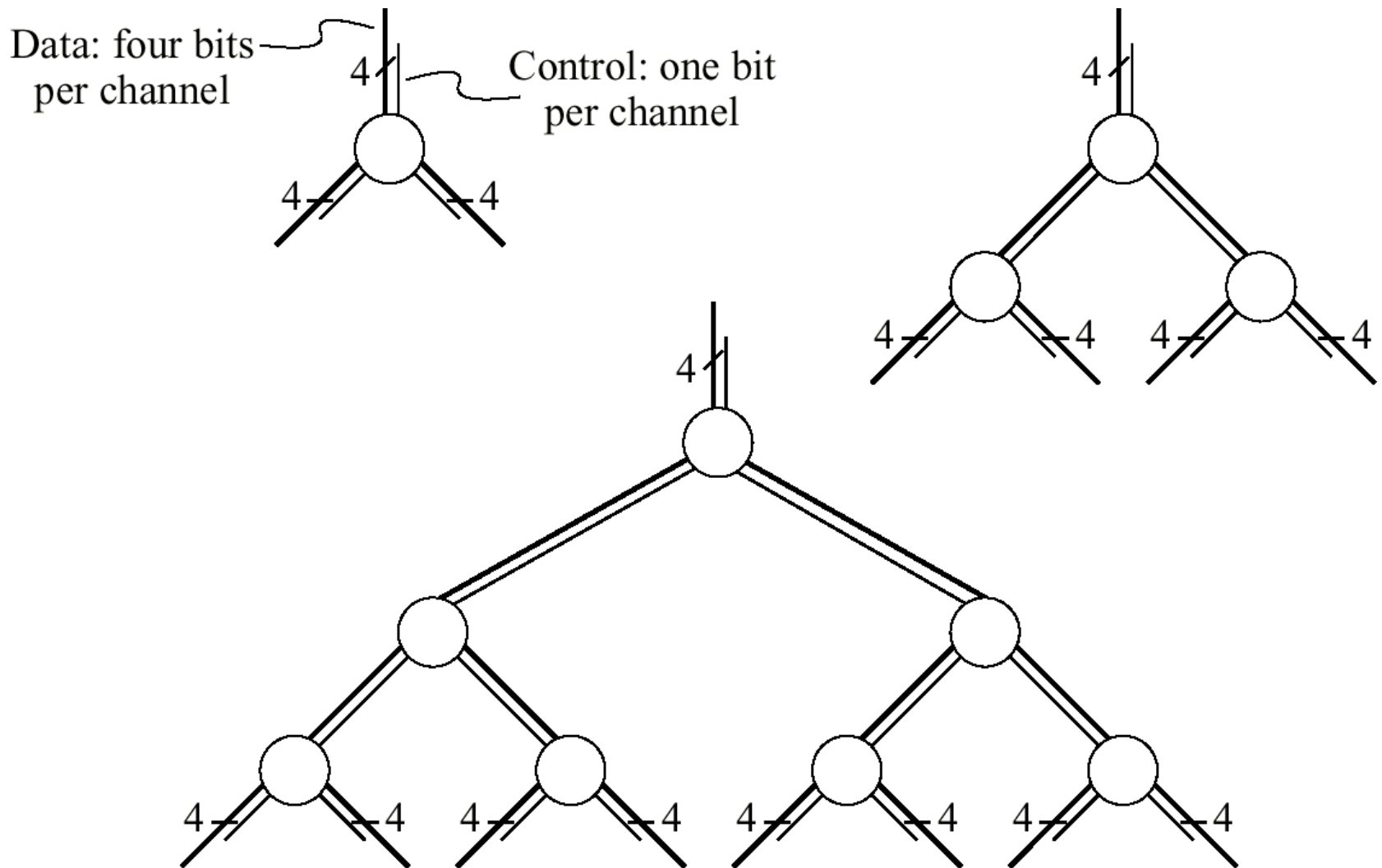
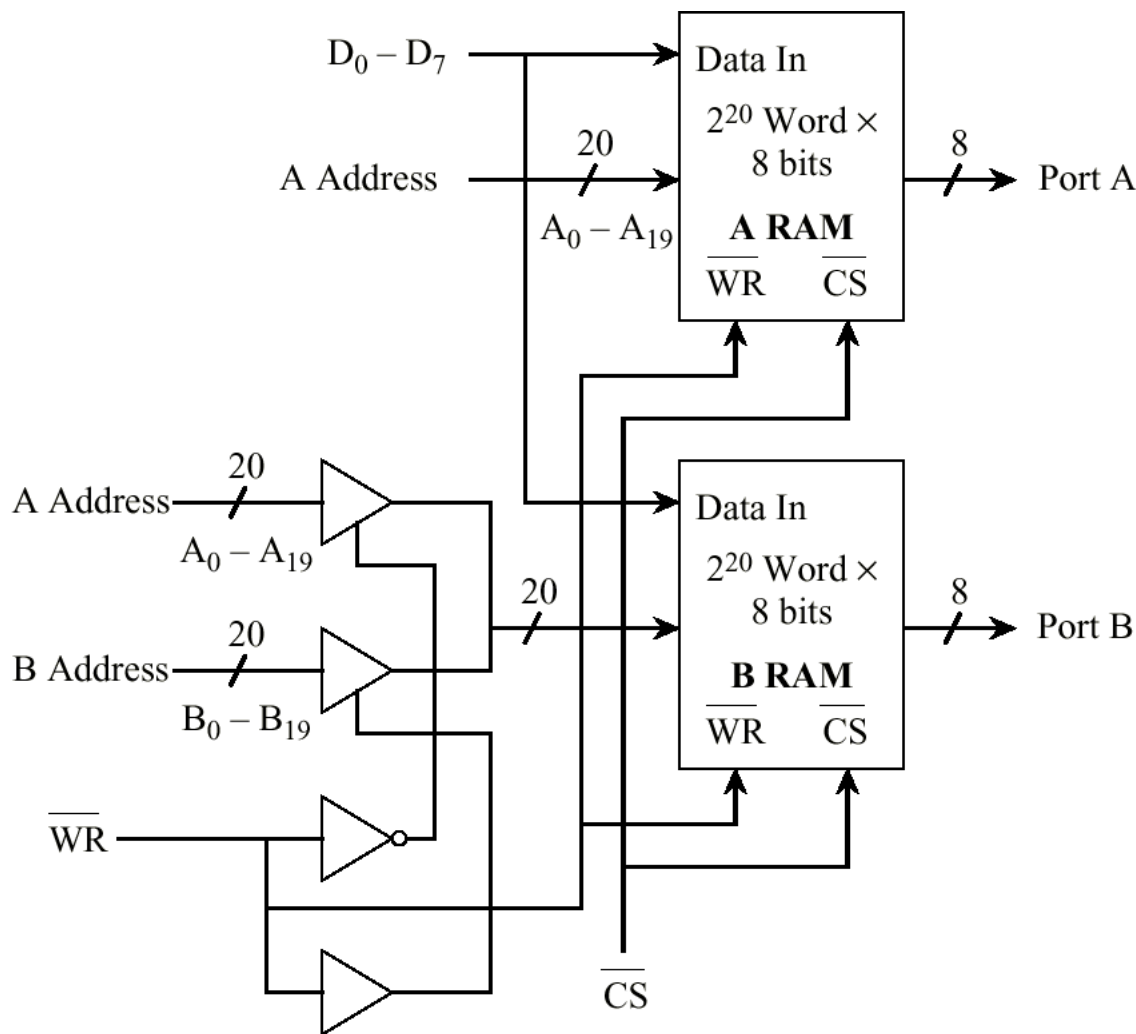
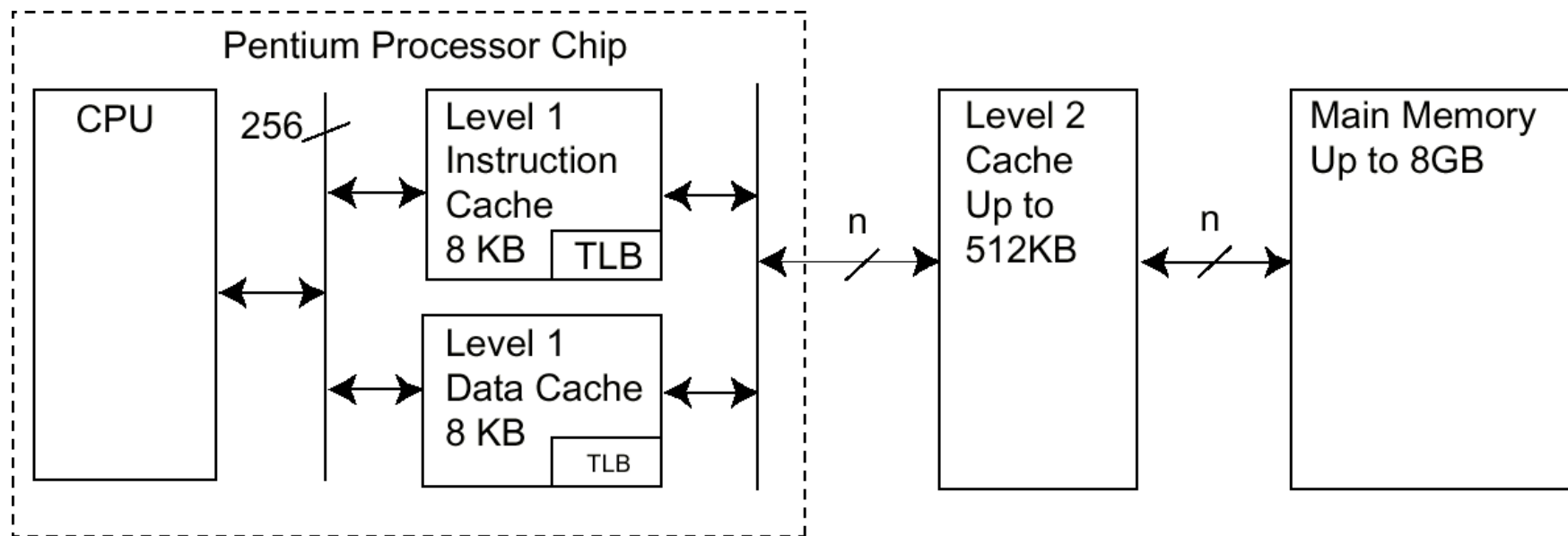


Diagrama de Blocos de uma RAM Dual-Read



O Sistema de Memória do Pentium da Intel



O Sistema de Memória do Pentium da Intel

- 2 caches L1 dentro do chip
 - Associativos por conjunto de 2 caminhos
 - I-cache – instruções
 - D-cache – dados
 - Escrita-de-volta ou escrita-através (acerto)
 - Escreva-não-aloca (miss)

O Sistema de Memória do Pentium da Intel

- Cada cache L1 possui um TLB
 - TLB do D-cache
 - Associativo por conjunto de 4 caminhos
 - 64 entradas
 - 2 portas (2 traduções simultâneas são possíveis)
 - TLB do I-cache
 - Associativo por conjunto de 4 caminhos
 - 32 entradas
- Cache L2
 - Associativo por conjunto de dois caminhos
 - 256 ou 512KB de tamanho
- Barramento de dados (n na figura)
 - 32, 64 ou 128 bits

O Protocolo MESI

- Gerencia o acesso à memória em sistemas multiprocessados
- Implementado pelo D-cache e cache L2, se presente
- Cada linha do D-cache pode estar em um de 4 estados
 - M – *modificado* – o conteúdo foi modificado e é diferente da memória
 - E – *exclusivo* – o conteúdo não foi modificado e é idêntico ao da memória principal
 - S – *compartilhado* – a linha é, ou pode ser, compartilhada com outra linha de cache em outro processador
 - I – *inválido* – a linha não está no cache

O Protocolo MESI

Estado da Linha de Cache	M Modificado	E Exclusivo	S Compartilhado	I Inválido
Linha válida?	Sim	Sim	Sim	Não
Cópia na memória é	Obsoleta	Válida	Válida	-
Cópias existem em outros caches?	Não	Não	Talvez	Talvez
Uma escrita nesta linha	Não vai ao barramento	Não vai ao barramento	Talvez vá ao barramento	Vai direto ao barramento

Paginação e Segmentação no Pentium

- ***Memória não-segmentada, não paginada***
 - Endereço virtual = endereço físico
 - Aplicações de alto desempenho que não têm grande complexidade
- ***Memória não-segmentada, paginada***
 - Espaço de endereçamento linear maior que espaço físico
 - Um TLB é usado no Pentium em conjunto com cache L1
- ***Memória segmentada, não-paginada***
 - Aplicações complexas com seções de dados que crescem dinamicamente
 - Rápida: não há paginação
- ***Memória segmentada e paginada***
 - Tabela de páginas, registradores de mapeamento de segmentos e um TLB
 - Múltiplos espaços de endereçamento

Memória Rambus

- Rambus technology on the Nintendo 64 motherboard (top left and bottom right) enables cost savings over the conventional Sega Saturn motherboard design (bottom left). (Photo source: Rambus, Inc.)

