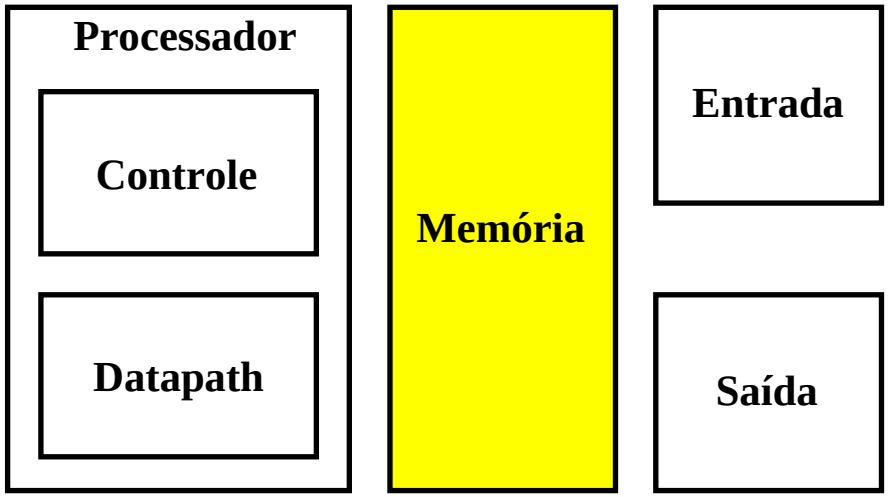
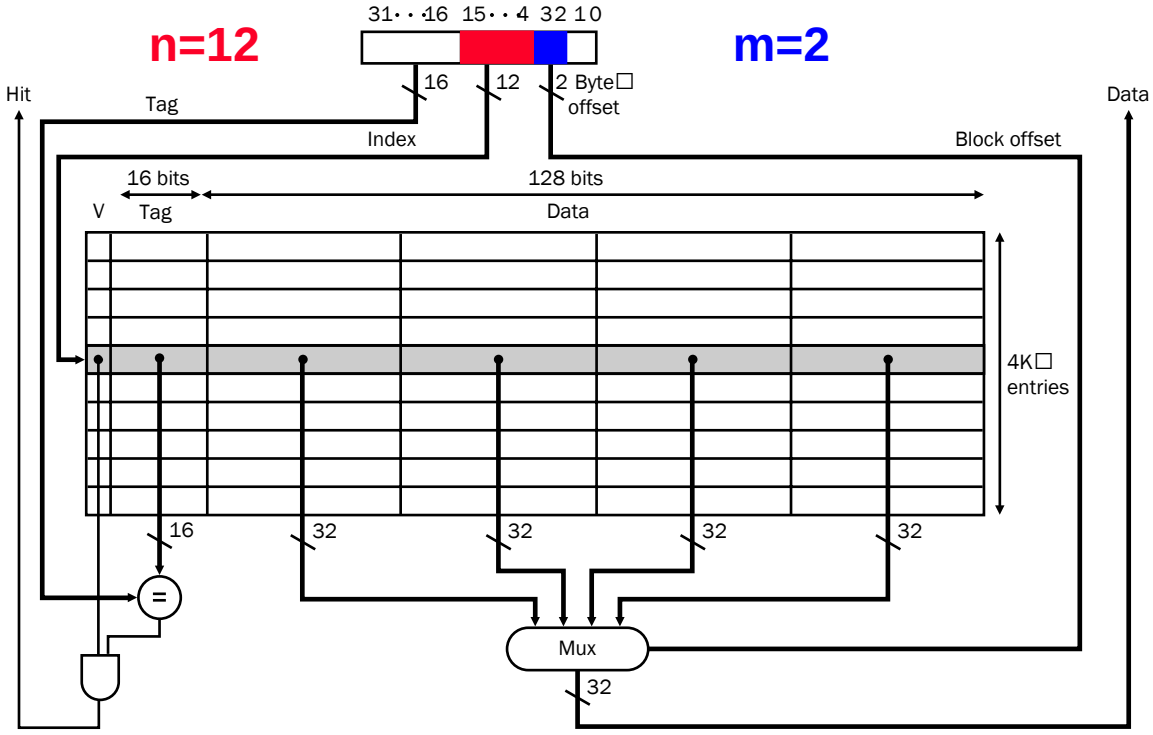
Cache: mapeamento e consistência



Explorando a Localidade Espacial

- **Cache com m=0 só explora localidade temporal**
- **Idéia-chave: bloco > 1 palavra**
  - **Fracasso: busca de palavras <u>adjacentes</u>**



Mapeamento:

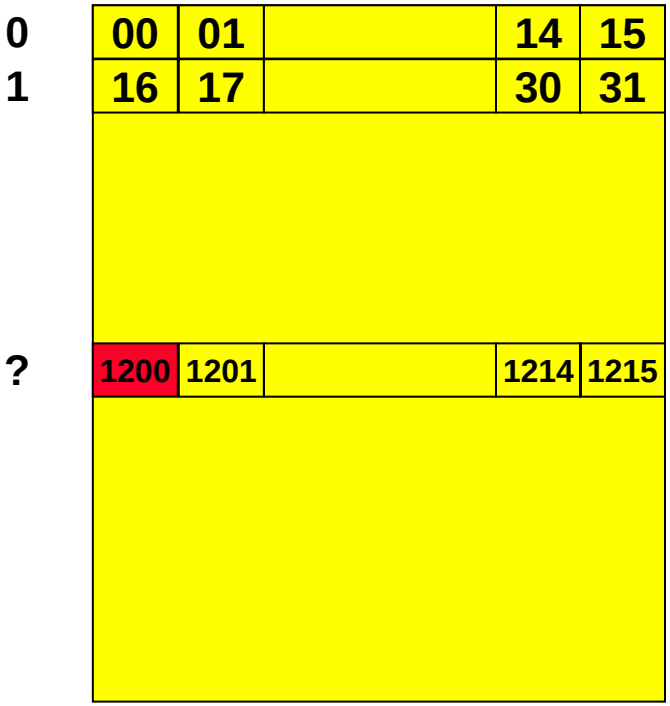
 # bloco modulo N,

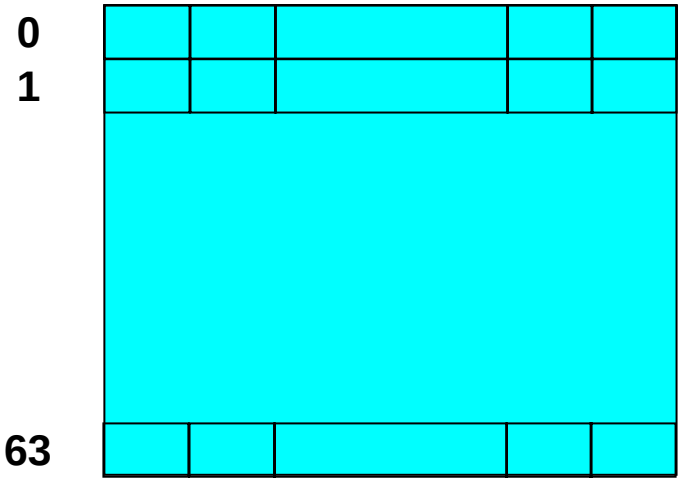
(onde N = 2^n)

$$\#\,\text{bloco} = \frac{\text{endereço do byte}}{\text{bytes por bloco}}$$

Múltiplas Palavras por Bloco

- **Exemplo de mapeamento**
  - **Cache armazena 64 blocos; bloco = 16 bytes.**
  - **Para que bloco da cache é mapeado o byte 1200 ?**

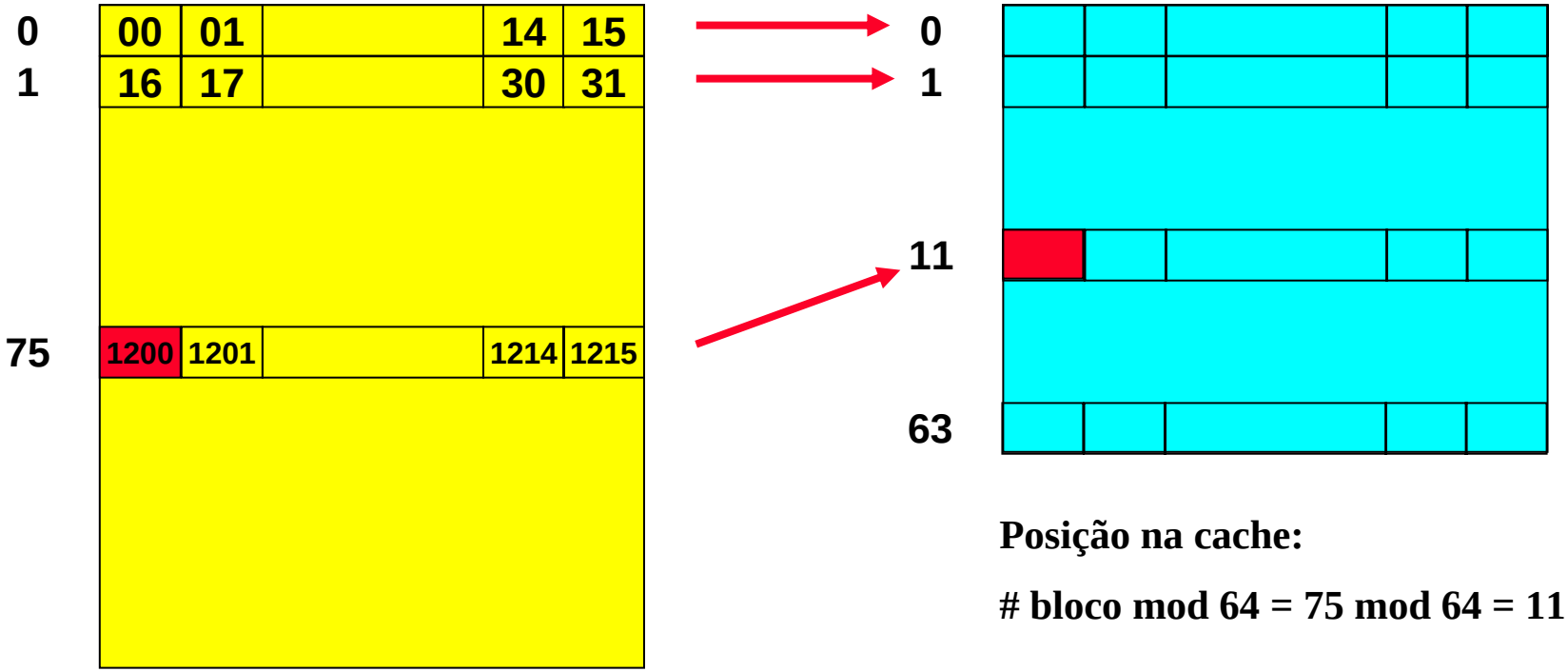




$$\#\text{bloco} = \frac{\text{endereço do byte}}{\text{bytes por bloco}} = \frac{1200}{16} = 75$$

Múltiplas Palavras por Bloco

- **Exemplo de mapeamento**
  - **Cache armazena 64 blocos; bloco = 16 bytes.**
  - **Para que bloco da cache é mapeado o byte 1200 ?**

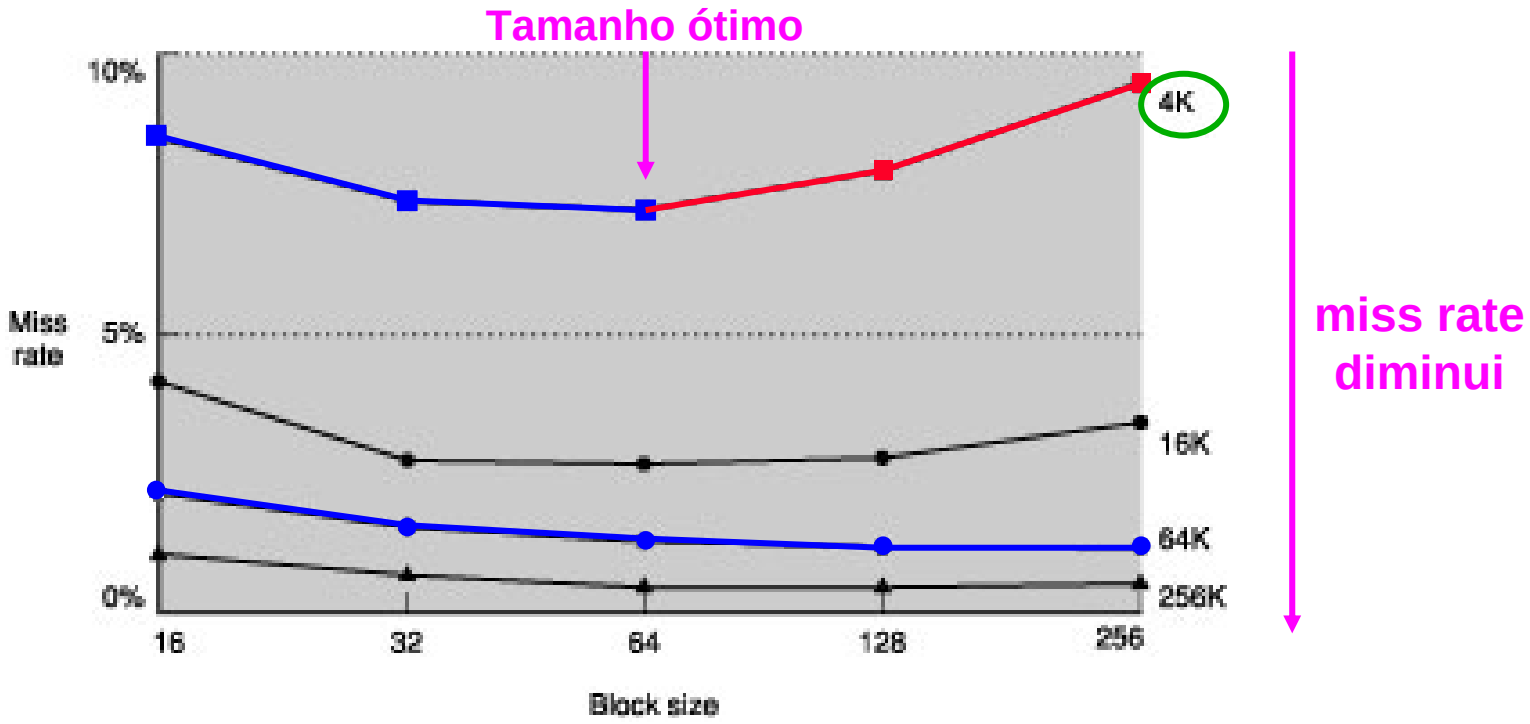


Posição na cache:

 # bloco mod 64 = 75 mod 64 = 11

$$\#\,\text{bloco} = \frac{\text{endereço do byte}}{\text{bytes por bloco}} = \frac{1200}{16} = 75$$

Impacto do tamanho do bloco



↑ bloco
captura local. espacial ↑
miss rate ↓

MAS,
capacidade fixa

↑bloco → num. blocos ↓
competição p/ bloco ↑
captura local. temporal ↓
miss rate ↑

Manipulando Fracassos na Cache

- **Unidade de controle monitora entrada “hit”**
  - **Detecta fracasso ou sucesso**
- **Sucesso**
  - **Continua execução normal**
- **Fracasso**
  - **Pausa da CPU (“stall on miss”)**
    - » **Unidade de controle da CPU “congela” registradores**
    - » **Controlador dedicado copia item requisitado p/ cache**

Diferenças no “stall”

- **Pausa devido a falta na cache**
  - **Toda a CPU sofre pausa**
    - » **Conteúdo dos registradores é “congelado”**
  - **Não requer salvamento de contexto**
    - » **Ao contrário de uma interrupção**
- **Pausa devido a hazard**
  - **Nem toda a CPU sofre pausa**
    - » **Algumas instruções continuam execução**
    - » **Para resolver o hazard**

Manipulando fracassos na cache

- **Enviar endereço para MP**
- **Comandar leitura da MP**
- **Esperar que acesso se complete**
  - **Requer múltiplos ciclos**
- **Atualizar a cache:**
  - **Bloco buscado na MP → campo de dados**
  - **MSBs do endereço → tag**
  - **Bit de validade é ativado**
- **Re-iniciar acesso à cache**
  - **Agora, item será encontrado!**

Fracasso no acesso a instrução

- **Enviar endereço para MP: atual PC-4**
- **Comandar leitura da MP**
- **Esperar que acesso se complete**
  - **Requer múltiplos ciclos**
- **Atualizar a cache:**
  - **Bloco buscado na MP → campo de dados**
  - **MSBs do endereço → tag**
  - **Bit de validade é ativado**
- **Re-iniciar acesso à cache: IF**
  - **Agora, item será encontrado! (instrução)**

Fracasso no acesso a dado

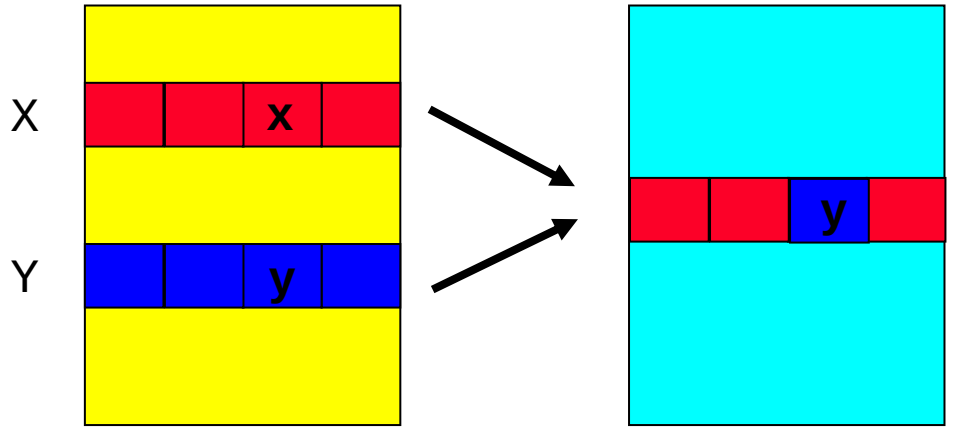
- **Enviar endereço para MP: endereço efetivo**
- **Comandar leitura da MP**
- **Esperar que acesso se complete**
  - **Requer múltiplos ciclos**
- **Atualizar a cache:**
  - **Bloco buscado na MP → campo de dados**
  - **MSBs do endereço → tag**
  - **Bit de validade é ativado**
- **Re-iniciar acesso à cache: MEM**
  - **Agora, item será encontrado! (dado)**

Manipulando escritas na cache

- **Discussão:**
  - **Store escreve só na cache (sem alterar MP)**
  - **Diferentes valores na MP e na cache**
    - » **Inconsistência**
- **Mecanismo para manter consistência**
  - **"Write-through"**
    - » **Store sempre escreve em ambas: MP e cache**
- **Integridade do bloco**
  - **Erro: simplesmente escrever item na cache**
  - **O que fazer antes de escrever na cache ?**

Múltiplas palavras por bloco

- **Hipótese: fracasso na escrita**



X mod N = Y mod N

- **Bloco inteiro <u>precisa</u> ser substituído antes de se escrever na cache**
  - **Escrita na cache pode requerer leitura na memória !?**
- **Caso particular: m=0 e mapeamento direto**
  - **Não é preciso comparar tag na escrita**
  - **Basta escrever o item (e seu tag) na cache**

Manipulando escritas na cache

- **Ineficiência do “write-through”**
  - **MP é muito mais lenta que cache**
    - » **Instruções store seriam muito mais lentas que outras**
- **Exemplo:**
  - **Tempo de acesso (MP) = 100 ciclos**
  - **SPECInt2000: 10% são stores**
  - **Ausência de hazards**
  - **CPI = 1 + 0,1 x 100 = 11**
    - » **Máquina ficaria ~10 vezes mais lenta**

Manipulando escritas na cache

- **Solução paliativa: "Write buffer"**
  - **Armazena escritas pendentes (valor, endereço)**
- **Dinâmica da escrita**
  - **Store escreve na cache e no "write buffer"**
  - **CPU continua execução (próximas instruções)**
  - **Quando uma escrita na MP se completa**
    - » **Sua entrada no write buffer é liberada**
- **Limitação**
  - **CPU sofre pausa quando "write buffer" cheio**

Manipulando escritas na cache

- **Alternativa: "write-back"**
  - **Store escreve só no bloco da cache**
  - **Bloco atualizado na MP só quando estiver na iminência de ser substituído**
- **Vantagens**
  - **Menor impacto da "bandwidth" no desempenho**
    - » **Várias escritas na MP ou cache L_{i+1} são "filtradas" na cache L_i**
  - **Menor consumo de energia**
    - » **Menor chaveamento em barramentos mais capacitivos**

Manipulando escritas na cache

- **Desvantagens**
  - **Implementação mais complexa**
    - » **Verificação de “dirty bit” no controlador de cache**
  - **Requer gerenciamento de inconsistência**
    - » **Valor em L_{i+1} ou MP inconsistente p/ E/S ou p/ outro core**
    - » **Requer mecanismo de coerência dos níveis inferiores**
  - **Pode resultar em desempenho inferior**
    - » **Em relação a “write-through + write buffer”**
    - » **Requer paradas no pipeline quando bloco é atualizado na MP**
- **Uso prático**
  - **“Write-through” combinado com “write-buffer”**
  - **“Write-back” puro**

Exemplo real

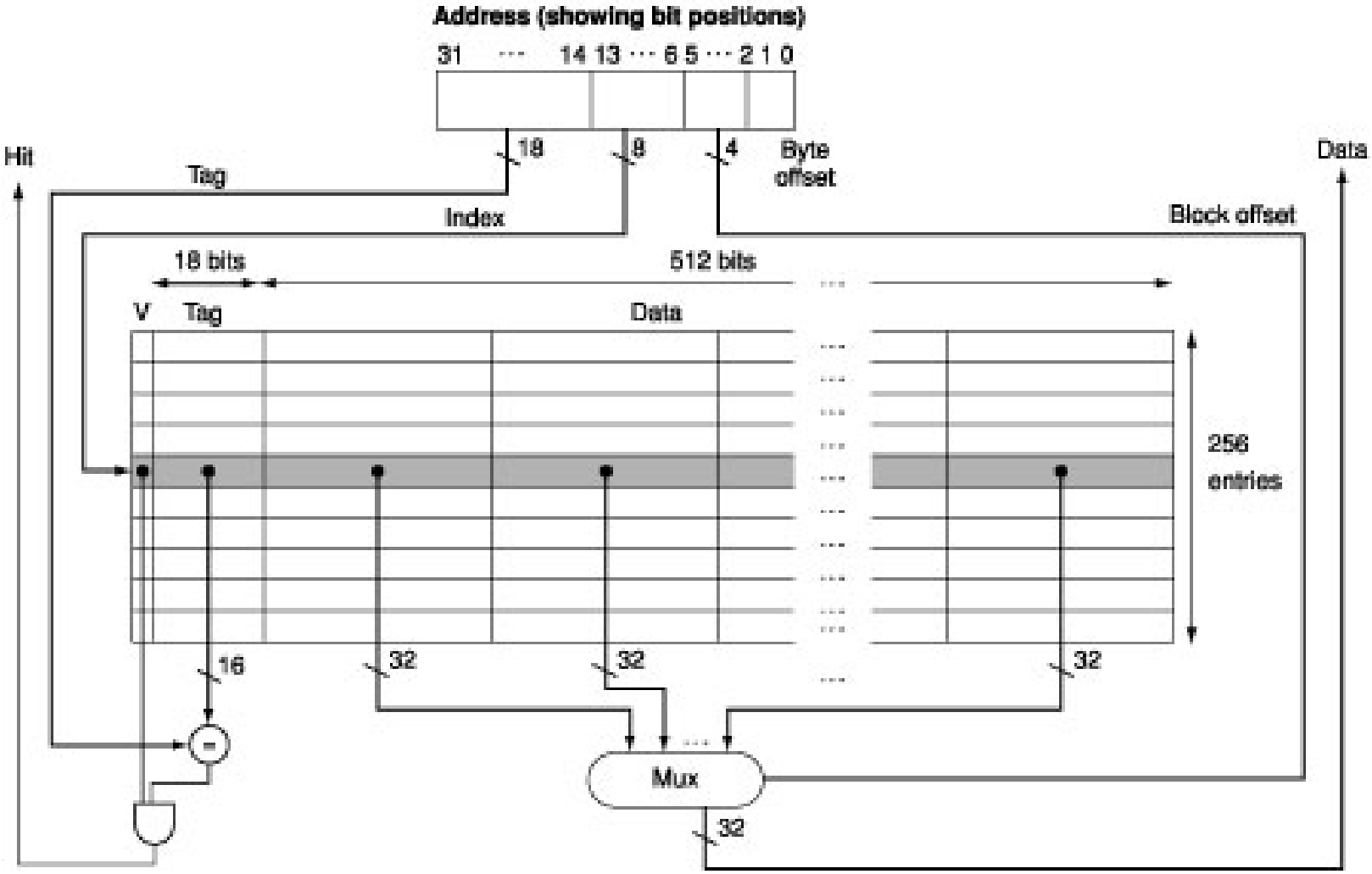
- **Instrinsity FastMATH**
  - **Processador embarcado = MIPS + cache**
- **Pipeline**
  - **12 estágios**
  - **Pode requisitar 1 instrução e 1 dado simultâneos**
    - » **Caches separadas: I-cache e D-cache**
    - » **Sinais independentes para read e write em cada cache**
- **Cache**
  - **16KB (cada)**
  - **Bloco: 16 palavras**

#Blocos = 4K palavras/16 palavras =256

n = 8 (índice)

m = 4 (word offset)

FastMATH: estrutura



FastMATH: comportamento

- **Leitura**
  - **Enviar endereço à cache apropriada**
    - » **PC (I-cache) ou ALU (D-cache)**
  - **Sucesso: palavra requisitada disponível**
    - » **Selecionada pelo MUX**
  - **Fracasso: endereço enviado a MP**
    - » **Bloco escrito na cache e, só então, lido**
- **Escrita**
  - **Mecanismo escolhido pelo sistema operacional**
    - » **“Write through” + “write buffer” de 1 entrada**
    - » **“Write back”**

FastMATH: desempenho da cache

- **SPEC2000 benchmarks**

Instruction miss rate	Data miss rate	Effective combined miss rate
0.4%	11.4%	3.2%

- **Discussão:**
  - **Quem tem maior localidade, I ou D-cache?**
  - **Qual tem menor taxa de fracassos ...**
    - » **Cache unificada ou caches separadas?**
  - **Unificada (m = 3,18%); separadas (m = 3,2%)**
    - » **Separadas resultam em maior "bandwidth"**

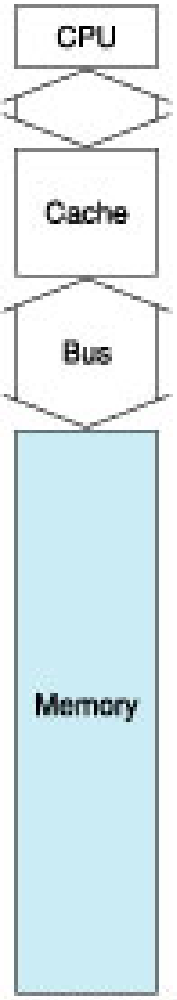
Projeto do sistema de memória

- **Latência**
  - **Tempo de acesso ao dado**
- **“Bandwidth”**
  - **Número de bytes acessados simultaneamente**
- **Memória principal**
  - **Construída com DRAMs**
  - **Para diminuir a penalidade de fracasso**
    - » **Difícil reduzir a latência da primeira palavra**
    - » **Mais prático aumentar o “bandwidth”**

Projeto do sistema de memória

- **Relógio do barramento**
  - **Normalmente bem mais lento que o da CPU**
    - » **Por exemplo, 10 vezes mais lento**
  - **Ciclo = ciclo do barramento da memória**
- **Hipótese:**
  - **1 ciclo para enviar endereço**
  - **15 ciclos para cada acesso à DRAM**
  - **1 ciclo para enviar uma palavra de dados**

Projeto do sistema de memória



a. One-word-wide memory organization

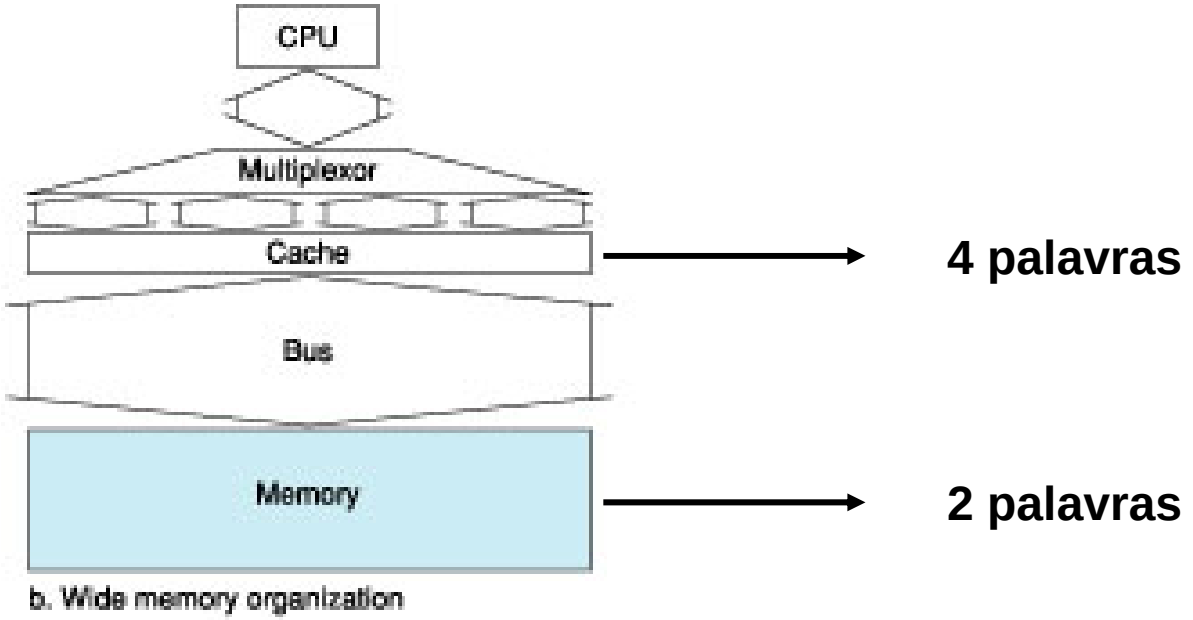
Cenário 1:

Cache: bloco com 4 palavras

MP: 1 banco de memória com largura de 1 palavra

Penalidade = 1 + 4 x 15 + 4 x 1 = 65 ciclos

Projeto do sistema de memória



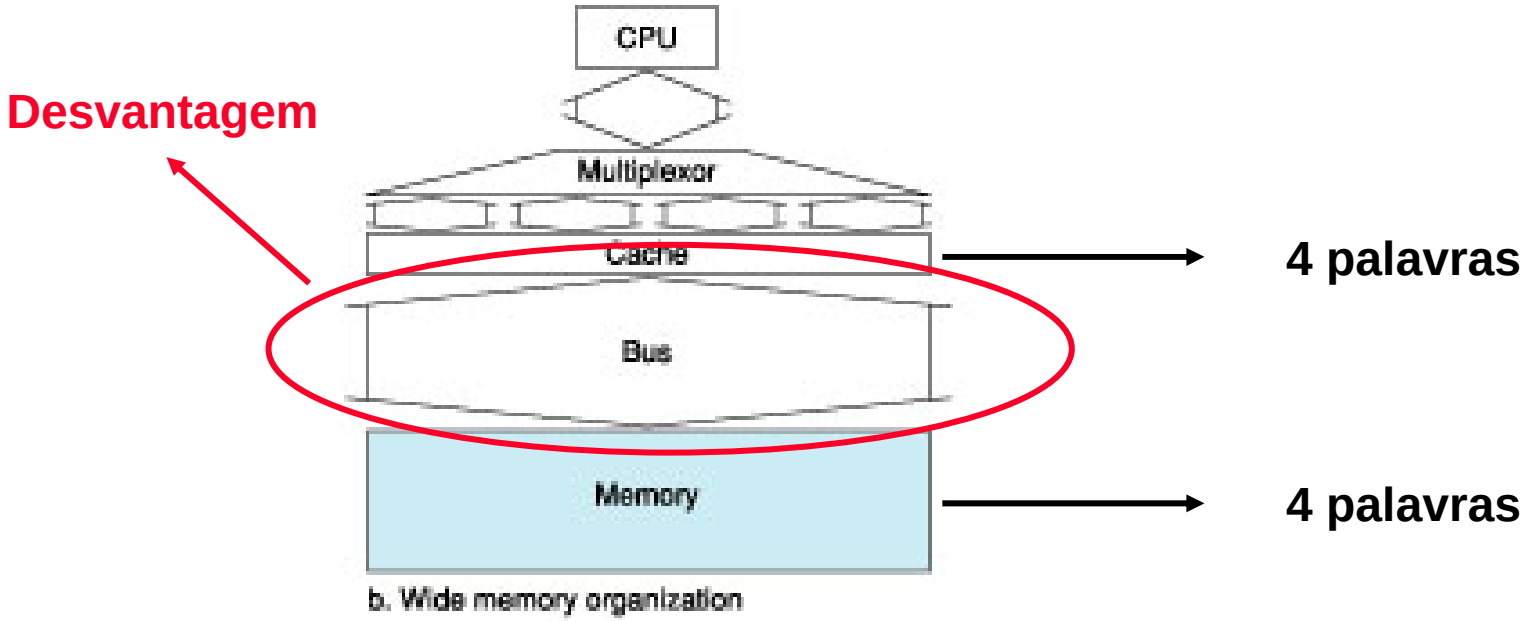
Cenário 2a:

Cache: bloco com 4 palavras

MP: 1 banco de memória com largura de 2 palavras

Penalidade = 1 + 2 x 15 + 2 x 1 = 33 ciclos

Projeto do sistema de memória



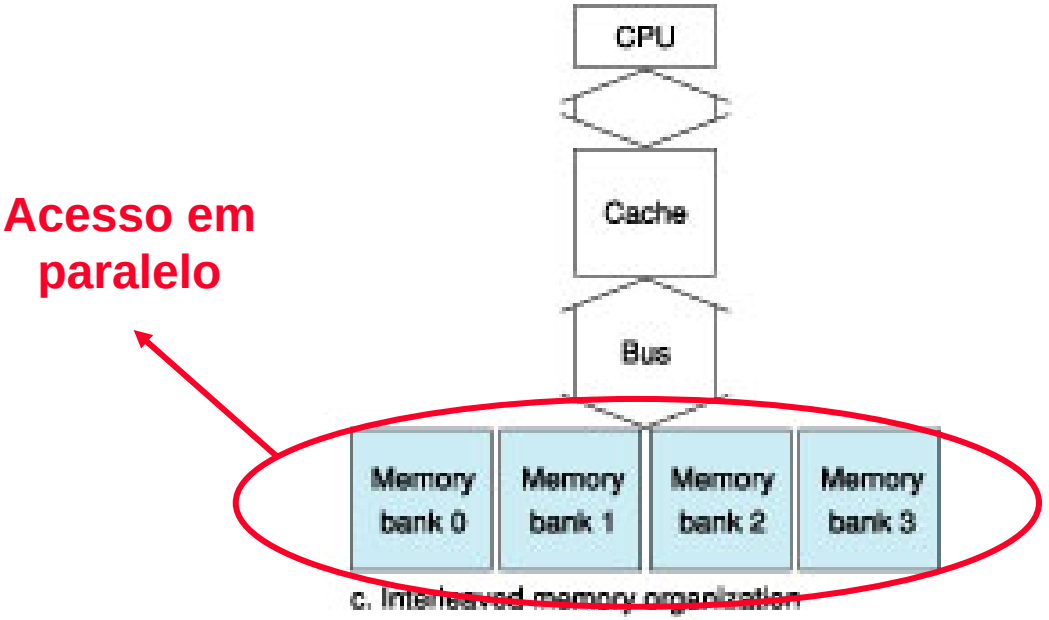
Cenário 2b:

Cache: bloco com 4 palavras

MP: 1 banco de memória com largura de 4 palavras

Penalidade = 1 + 1 x 15 + 1 x 1 = 17 ciclos

Projeto do sistema de memória



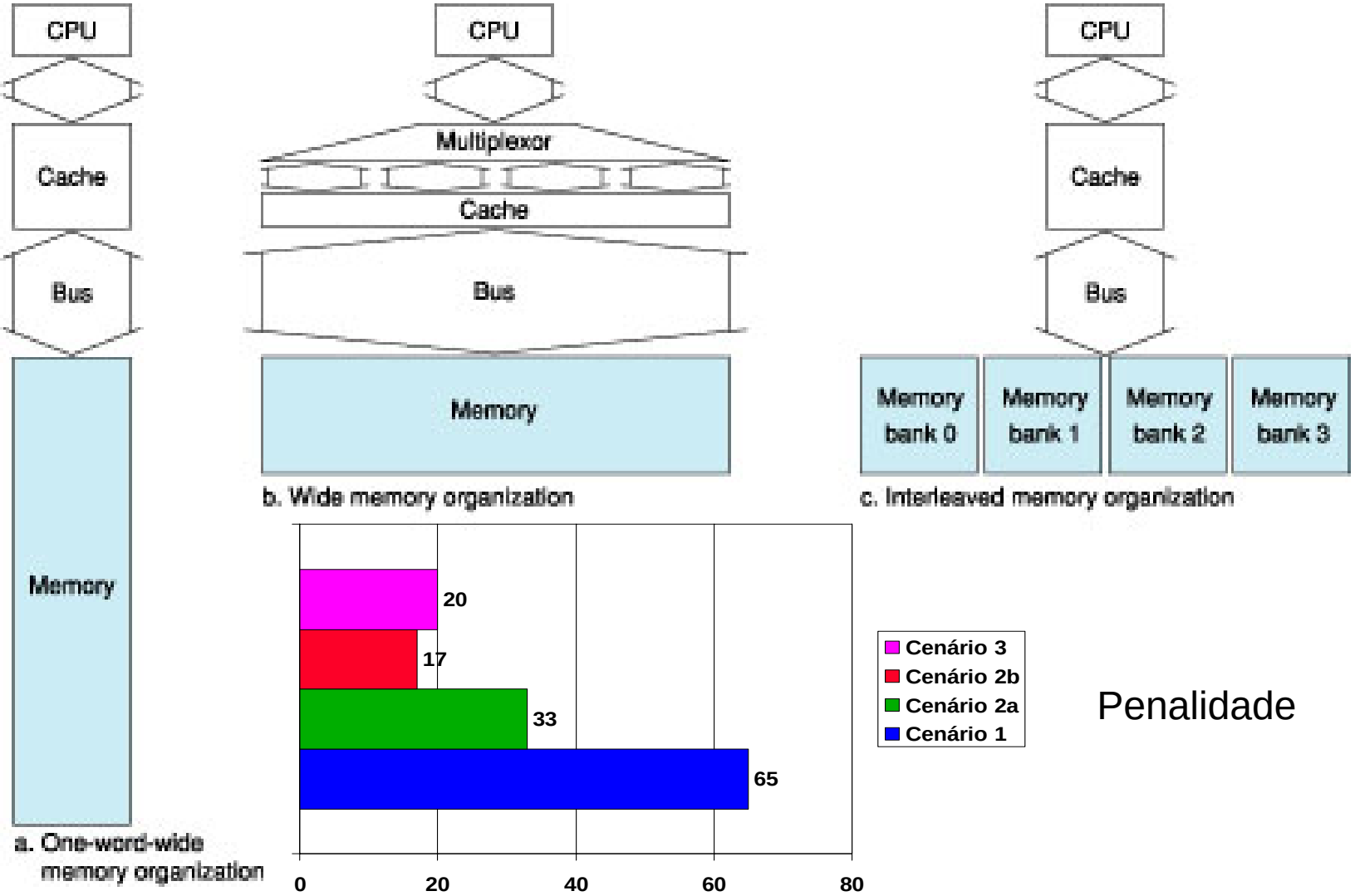
Cenário 3:

Cache: bloco com 4 palavras

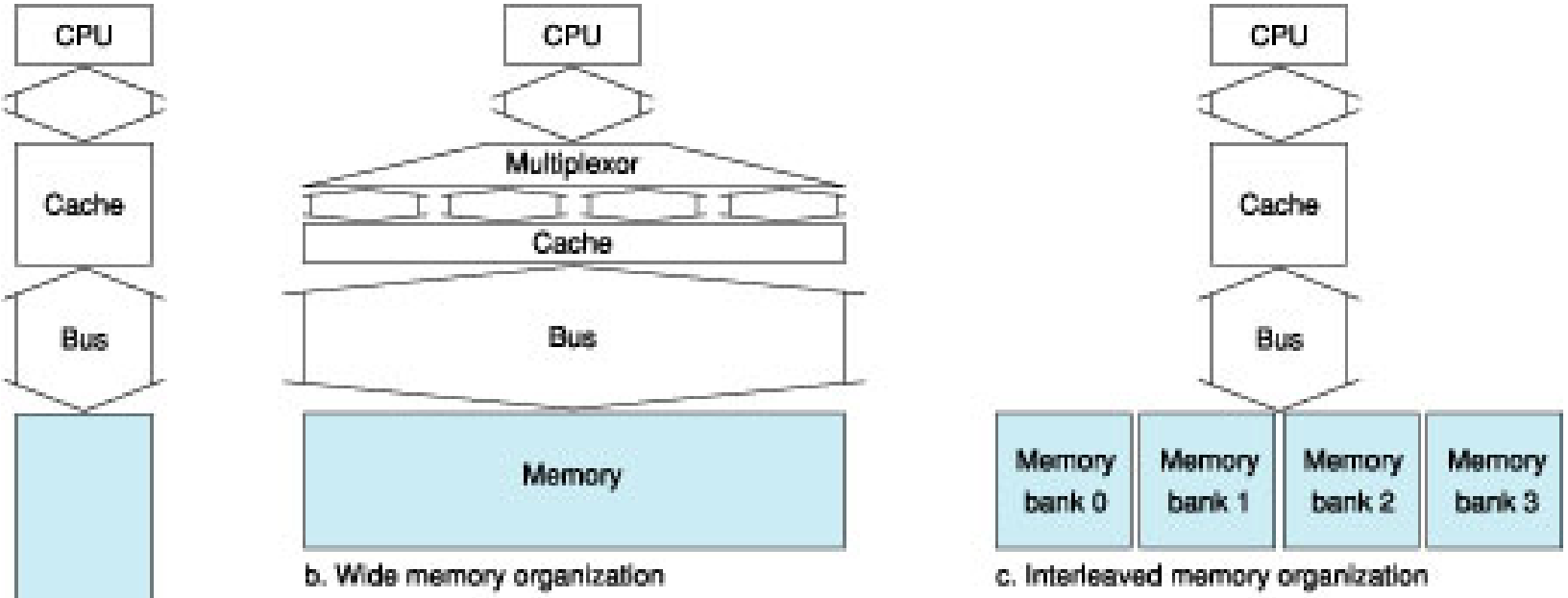
MP: 4 bancos de memória com largura de 1 palavra

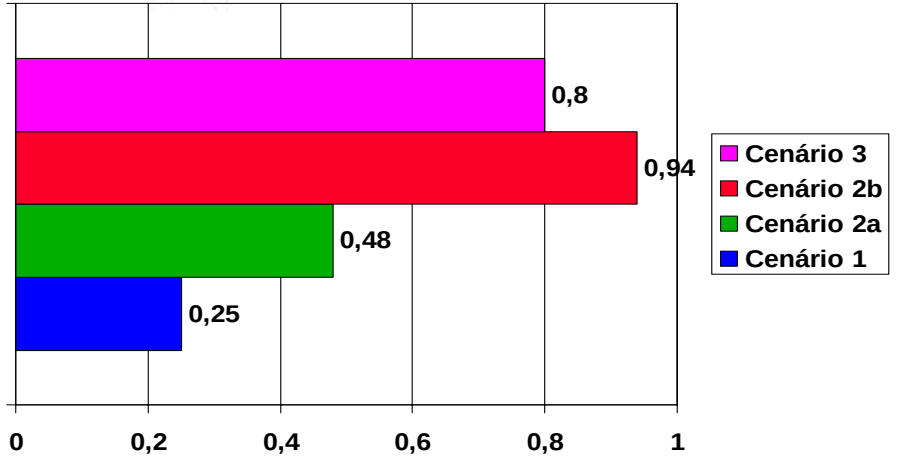
Penalidade = 1 + 1 x 15 + 4 x 1 = 20 ciclos

Projeto do sistema de memória



Projeto do sistema de memória





Número de bytes transferidos por ciclo

Conclusões: como melhorar desempenho?

- **Diminuir “miss rate”**
  - **Aumentando o tamanho do bloco**
    - » **Até um limite que depende do tamanho da cache**
- **Aumentar “bandwidth”**
  - **Caches separadas para dados e instruções**
- **Diminuir “miss penalty”**
  - **Projeto adequado do sistema de memória**
    - » **Largura do barramento, memória entrelaçada**
- **Sintonizar com aplicação-alvo**
  - **Escolhendo melhor política de consistência**
    - » **“Write-through+Write buffer” x “Write back”**