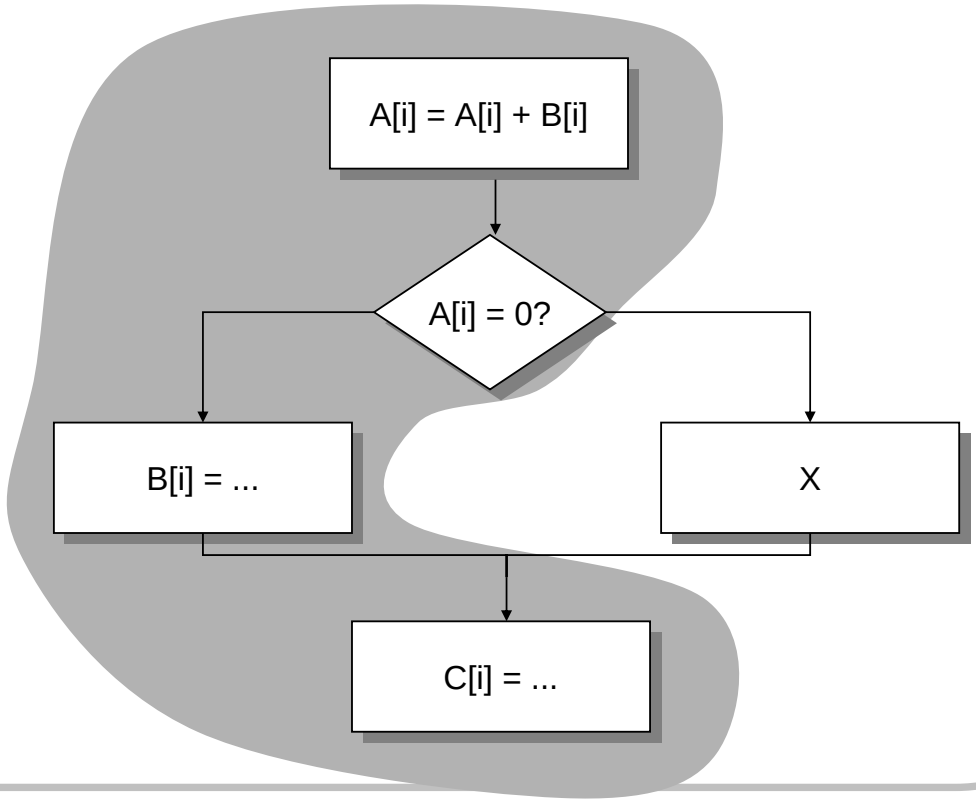
Aula 12: Término de Pipelines Avançados Hierarquia de Memória — Motivação, Definições, Quatro Perguntas Fundamentais

Trace Scheduling

- Permite encontrar paralelismo cruzando branches de IFs vs. branches de LOOPs
- Dois passos:
  - *Trace Selection*
    - » Encontre sequência mais provável de blocos básicos (*trace*) a partir de sequência estaticamente prevista de código sequencial
  - *Trace Compaction*
    - » Comprima *trace* no menor número de instruções VLIW o possível
    - » Necessita de manter informações para recuperação em caso de previsão errônea

Seleção de *Trace*



Suporte em HW para Mais ILP

- *Speculation*: permite uma instrução executar sem nenhuma consequência, mesmo se branch não for tomado (“HW undo”)
- Geralmente combinado com escalonamento dinâmico
- Tomasulo: separa bypass especulativo dos resultados do bypass real dos resultados
  - Quando instr. não for mais especulativa, escreva os resultados (*instruction commit*)
  - Executa fora de ordem, mas *commit* em ordem

Suporte em HW para Mais ILP

- Necessita de buffer em HW para resultados de instruções *uncommitted*: *reorder buffer*
  - *Reorder buffer* pode ser fonte de operandos
  - Quando instr. *commit*, resultado pode ser encontrado nos registradores
  - 3 campos: tipo instr, destino, valor
  - Use número do *reorder buffer* ao invés do número da *reservation station*

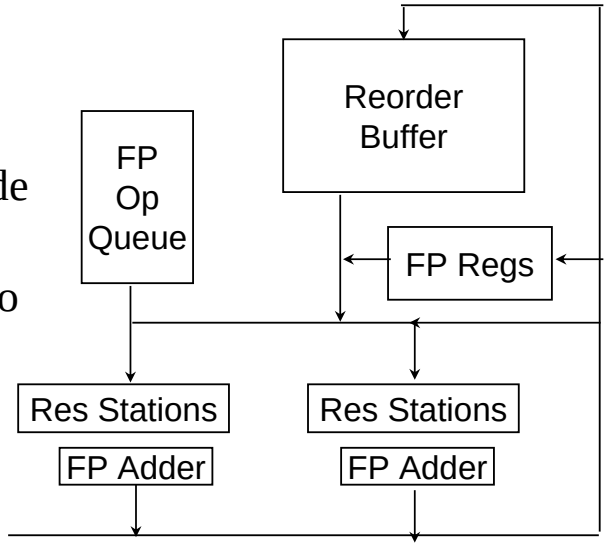


Figura 4.34, página 311

Quatro Passos do Algoritmo de Tomasulo com Especulação

1. Issue—carrega instrução da fila de instruções de FP

   Se *reservation station* ou *reorder buffer slot* livre, issue instr & envie operandos & num. *reorder buffer* para destino.

2. Execução—(EX)

   Quando ambos operandos estiverem na *reservation station*, execute;

3. Escrita de Resultado—término da execução(WB)

   Escreva no CDB para todas as FUs & reorder buffer aguardando resultado; marque *reservation station* livre.

4. Commit—grave resultado com resultado de *reorder*

   Quando instr. no topo do *reorder buffer* & resultado estiver presente, grave resultado em registrador ou memória

Limites para ILP

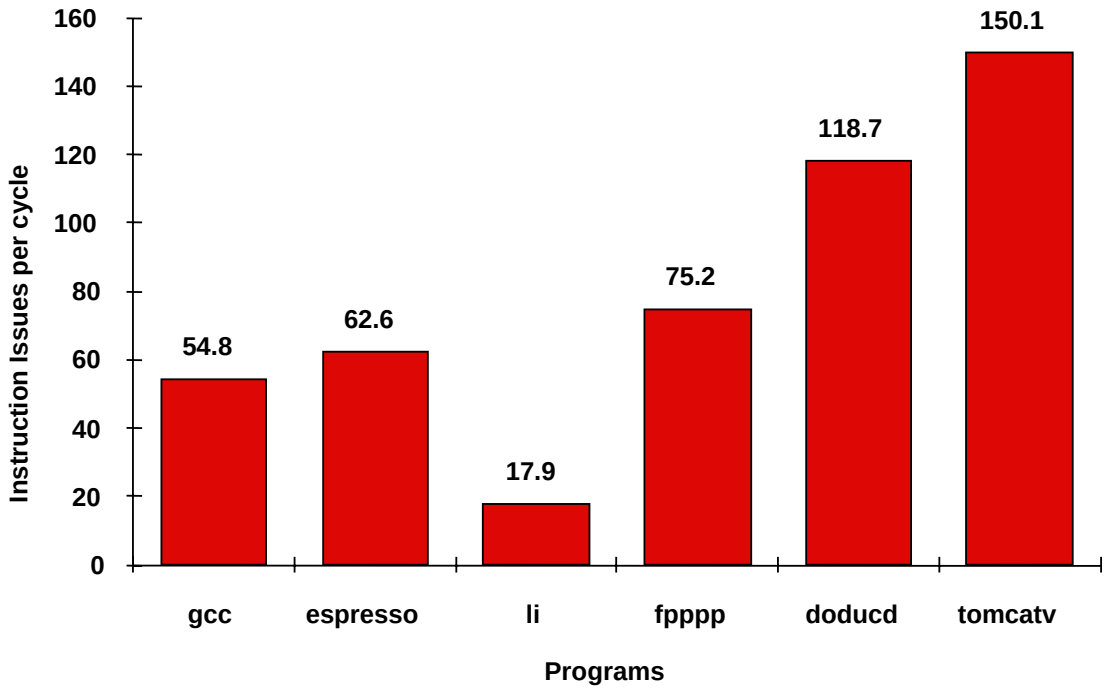
Modelo inicial de hardware; compiladores MIPS

1. *Register renaming*–número infinito de registradores virtuais para evitar todos os hazards de WAW & WAR
2. *Branch prediction*–perfeito; sem previsões erradas
3. *Jump prediction*–previsões perfeitas => máquina com especulação perfeita & buffer de instruções ilimitado
4. *Análise de sinônimos de memória*–endereços são conhecidos e stores podem ser movidos antes de loads se os endereços não forem iguais

1 ciclo de latência para todas as instruções

Limite Superior para ILP

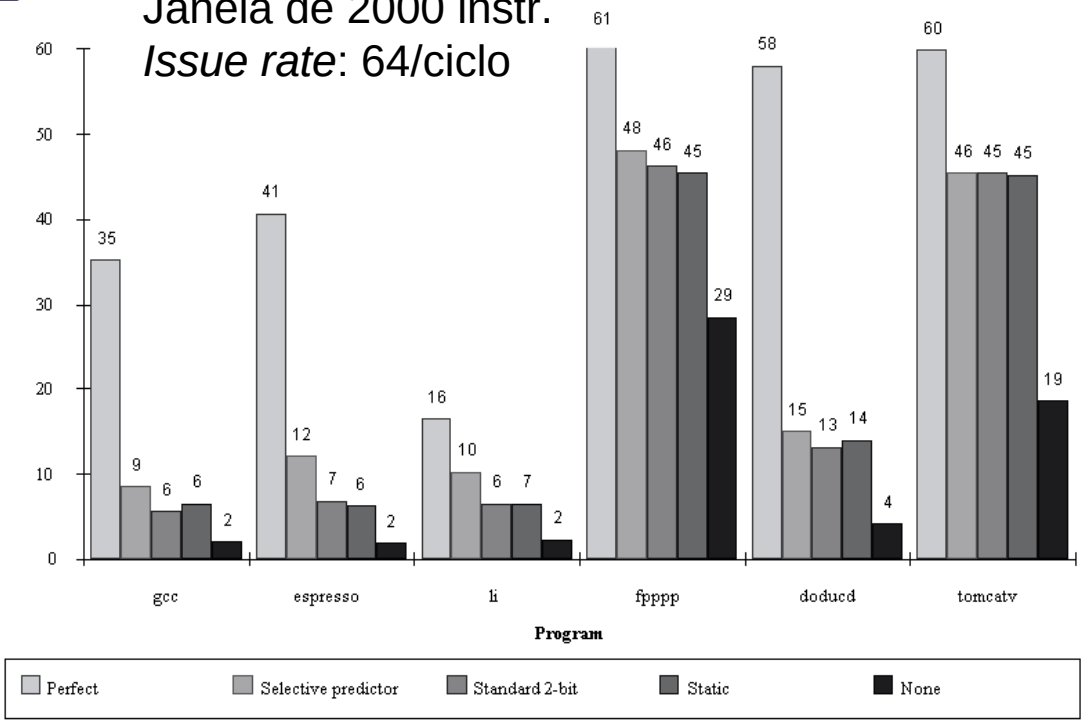
(Figure 4.38, page 319)



HW Real: Impacto de Branches

Figure 4.40, Page 323

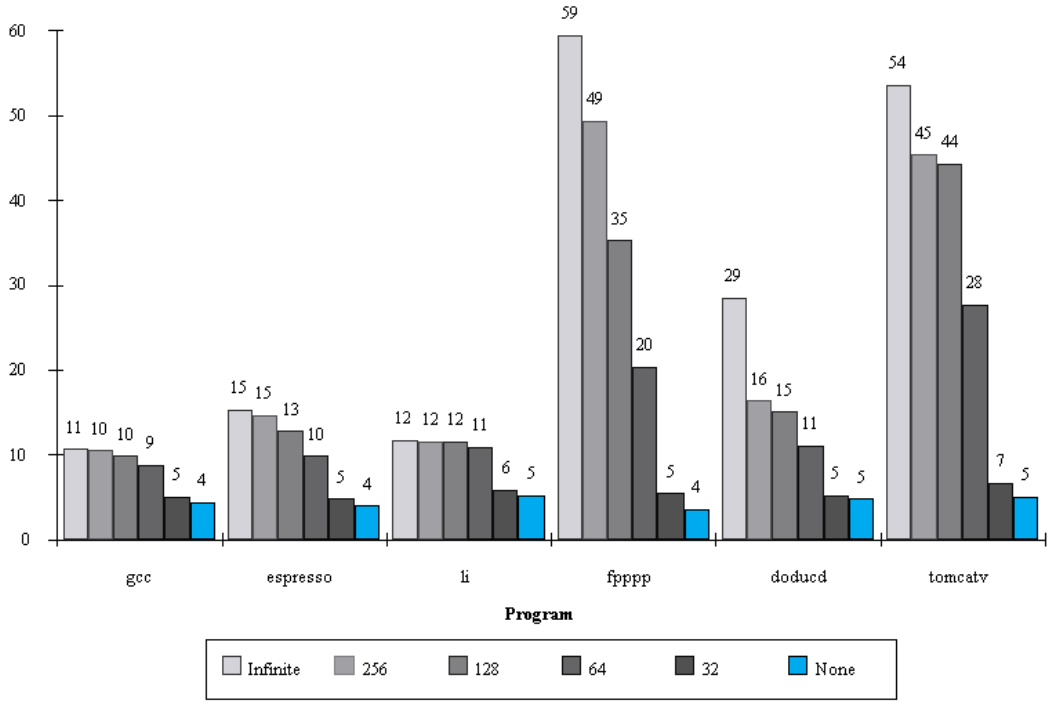
Janela de 2000 instr.
Issue rate: 64/ciclo



Perfect **Pick Cor. or BHT** **BHT (512)** **Profile**

HW Real: Impacto de Registradores

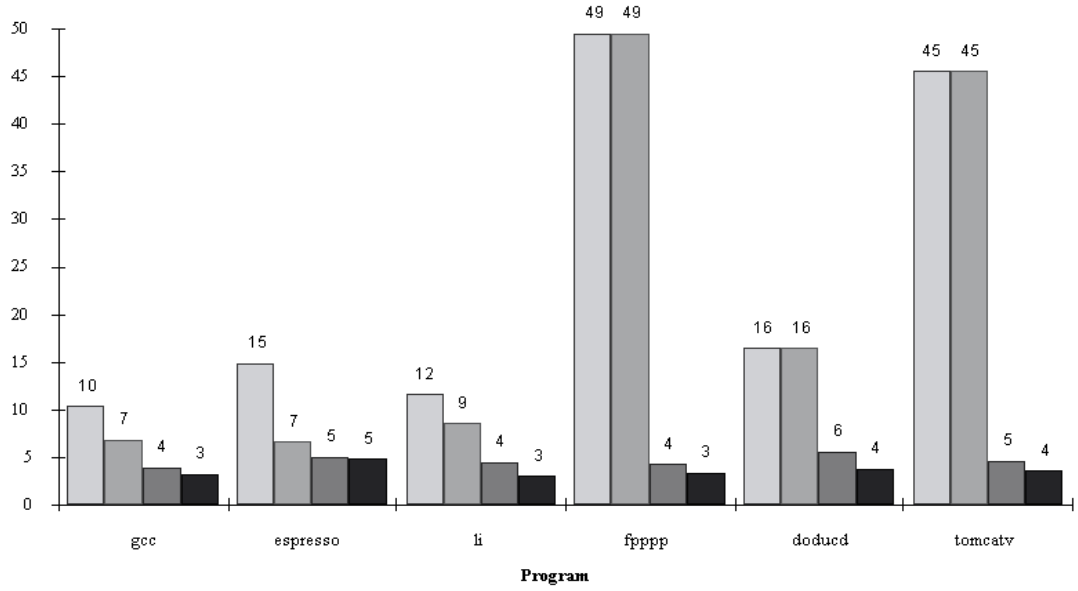
Figure 4.42, Page 325



Infinite **256** **128** **64** **32** **None**

HW Real: Impacto de Alias

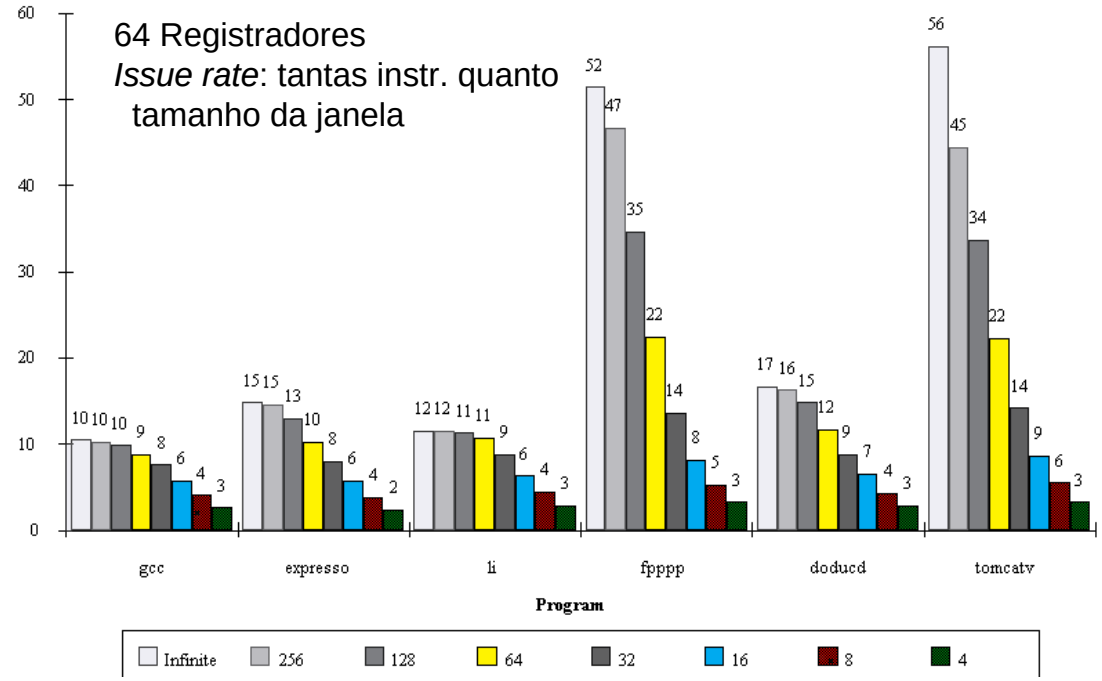
Figure 4.44, Page 328



Perfect **Global/Stack perf; heap conflicts** **Inspec. Assem.** **None**

HW Real: Impacto da Janela de Instrs

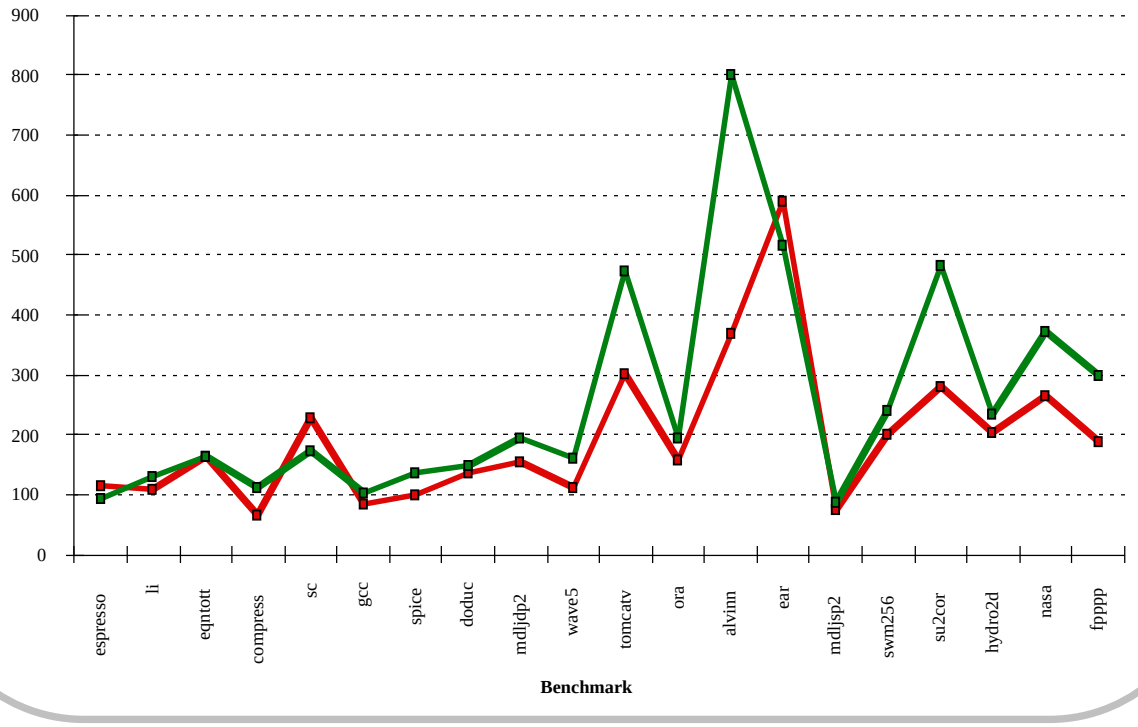
(Figure 4.48, Page 332)



Infinite **256** **128** **64** **32** **16** **8** **4**

Força Bruta vs. Rapidez

8-scalar IBM Power-2 @ 71.5 MHz (5 stage pipe)
vs. 2-scalar Alpha @ 200 MHz (7 stage pipe)



Conclusões Cap. 4

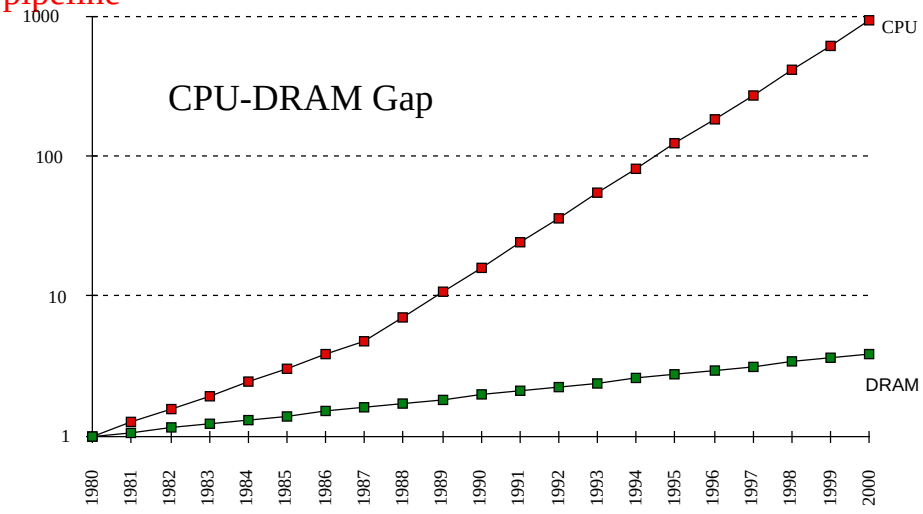
- Mais transistores gastos para manter compatibilidade do que para melhorar performance
- Processador superescalar adiciona complexidade cujo preço pode ser muito alto
  - O que é melhor superescalar 8x ou superescalar 2x com um clock mais rápido?
  - Custo do chip x performance da máquina
- Tamanho de janela é um dos obstáculos principais:
  - Janela de 32 instrs -> 900 comparações por ciclo

Hierarquia do Sistema de Memória

Cache: a safe place for hiding or storing things...

Quem se Preocupar com a Hierarquia de Memória?

- Somente vimos processadores até agora...
  - Custo/performance de CPU cost/performance, ISA, Execução com pipeline



- 1980: não haviam caches em processadores (386)
  1995: caches de 2 níveis no PC (486,Pentium)

Princípios Gerais

- Localidade
  - *Localidade temporal*: dados serão referenciados novamente
  - *Localidade espacial*: itens serão referenciados na vizinhança
- Localidade + HW menor é mais rápido == hierarquia de memória
  - *Níveis*: cada menor e mais rápida é mais cara que a de nível mais baixo
  - *Inclusive*: dado encontrado no topo também é encontrado no nível mais baixo
- Definições
  - *Nível mais alto* é mais próximo ao processador
  - *Bloco*: unidade mínima de dados (também conhecido como *linha*)
  - Endereço = *endereço do bloco* + *offset dentro do bloco*
  - *Hit time*: Tempo para acessar dado no nível mais alto

Princípios Gerais de Caches

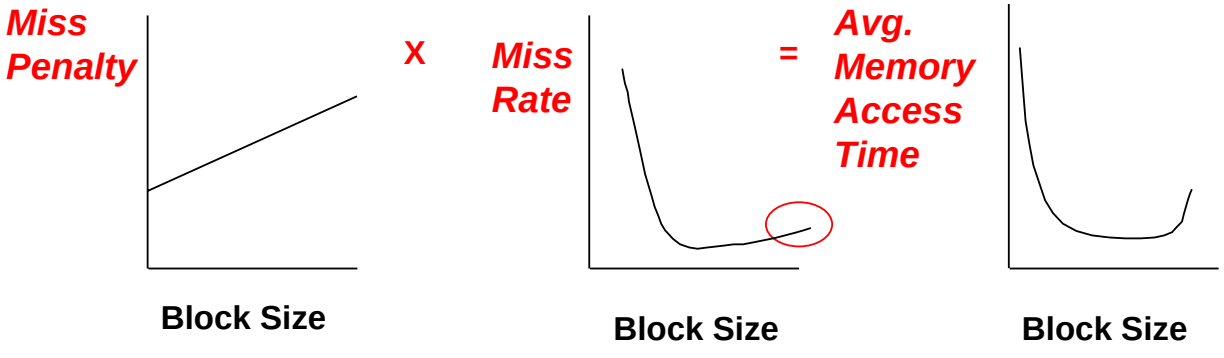
P	***400MHz***
L1	***400MHz***
L2	***100MHz***
M	***50-100MHz***
D	***10KHz***

Medições em Caches

- *Hit rate*: fração achada no nível
  - Tão alta que as vezes usamos *Miss rate*
- Tempo médio de acesso à memória = *Hit time* + *Miss rate* x *Miss penalty* (ns ou clocks)
- *Miss penalty*: tempo requerido para trocar um bloco vindo de um nível mais baixo, incluindo tempo até carregar dado na CPU
  - *tempo de acesso*: tempo para nível mais baixo = *f*(latência do nível mais baixo)
  - *tempo de transferência*: tempo para transferir bloco = *f*(BW nível + alto e + baixo, tamanho do bloco)

Tamanho do Bloco vs. Performance de Caches

- Aumento do tamanho do bloco geralmente causa aumento do tempo de acesso médio

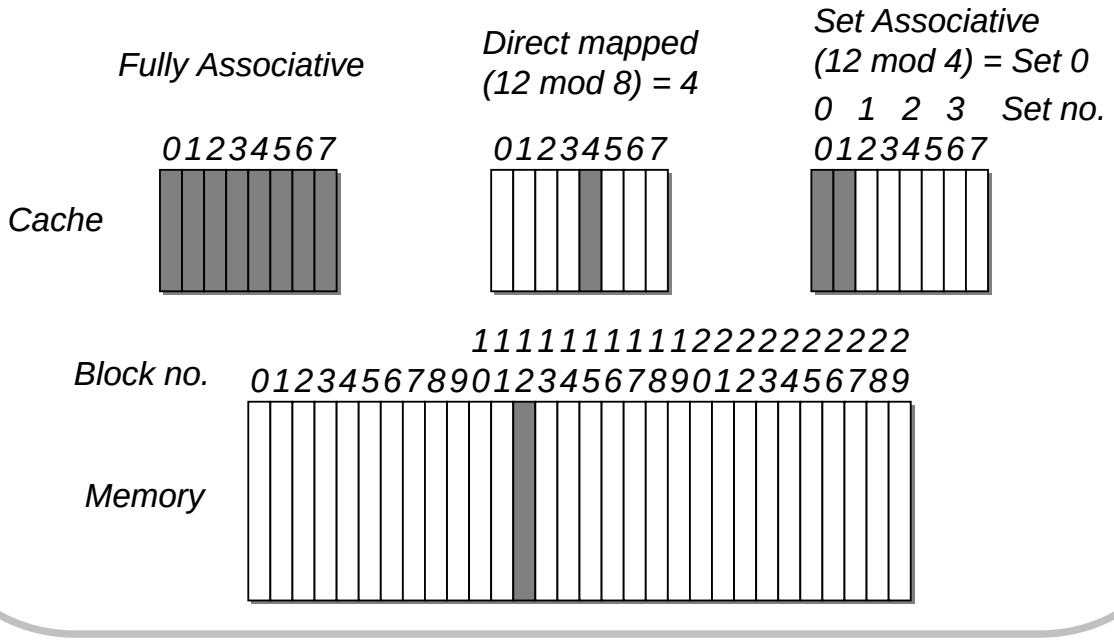


Quatro Perguntas Básicas sobre Hierarquia de Memória

- Q1: Onde o bloco vai ser colocado na memória de nível mais alto? *(Block placement)*
- Q2: Como bloco é encontrado na memória de nível mais alto? *(Block identification)*
- Q3: Quais blocos serão trocados em um miss? *(Block replacement)*
- Q4: O que acontece em uma escrita? *(Write strategy)*

Q1: Onde Bloco Deve Ser Colocado no Nível Mais Alto?

Onde bloco 12 deve ser colocado?



Q2: Como o Bloco é Encontrado no Nível Mais Alto?

- Tag para cada bloco
  - Não é necessário checar índice ou offset
- Aumento de associatividade reduz índice, aumenta tag

Block Address		Block offset
Tag	Index	

Q3: Qual Bloco Será Trocado Durante um Miss?

- Fácil de decidir para caches de mapeamento direto
- Para *Set Associative* ou *Fully Associative*:
  - Aleatório (associatividade alta)
  - LRU (associatividade baixa)

Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64 KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: O Que Acontece Durante uma Escrita?

- *Write through*: A informação é escrita tanto para o bloco da cache quanto para a memória de nível mais baixo.
- *Write back*: A informação é escrita somente para o bloco da cache. Este bloco é escrito na memória quando ele for trocado.
  - Precisa de *dirty bit*
- Vantagens e Desvantagens:
  - WT: miss de leitura não resulta em escrita na memória
  - WB: reduz BW para memória de nível mais baixo
- WT está sempre combinada com *write buffers* de forma a não precisarmos esperar pelo nível mais baixo de memória

Q4: O Que Acontece Durante uma Escrita?

- *Write allocate (fetch on write)* : bloco é trazido para cache se ocorrer um miss de escrita, seguido por uma escrita com hit.
- *No-write allocate (write around)*: bloco é modificado no nível mais baixo somente e não é carregado na cache.
- WB é geralmente utilizado com *write allocate*
- WT é geralmente utilizado com *no-write allocate*

Resumo

- Gap entre CPU e memória é um dos maiores obstáculos para a performance de sistemas de computação
- Para melhorarmos a relação custo/benefício, utilizamos o princípio da localidade
- Para qualquer nível de memória, queremos saber a resposta para os 4 Qs