Especulação para "Resolver" Alias Dependency

Luciana B. Carvalho – RA: 981561

MO401 – Arquitetura de Computadores I

Campinas, 14 de junho de 2010

Introdução

- Alias dependency
- ILP (Instruction-Level Parallelism)
- Especulação
  - Por hardware
  - Por software

Especulação por Hardware

- Algoritmo de Tomasulo especulativo
- Processadores: Pentium III/4, MIPS R10K, Alpha 21264, HP PA8500 e IBM RS64III
- Dependence Prediction
  - Blind
  - Wait
  - Store Sets
    - Melhor desempenho

Especulação por Hardware

- Address Prediction
  - Last Address Prediction
  - Stride Prediction
  - Context Prediction
  - Hybrid Prediction

Especulação por Hardware

- Value Prediction
  - Last Value Prediction
  - Stride Prediction
  - Context Prediction
  - Hybrid Prediction

Especulação por Hardware

- Memory Renaming

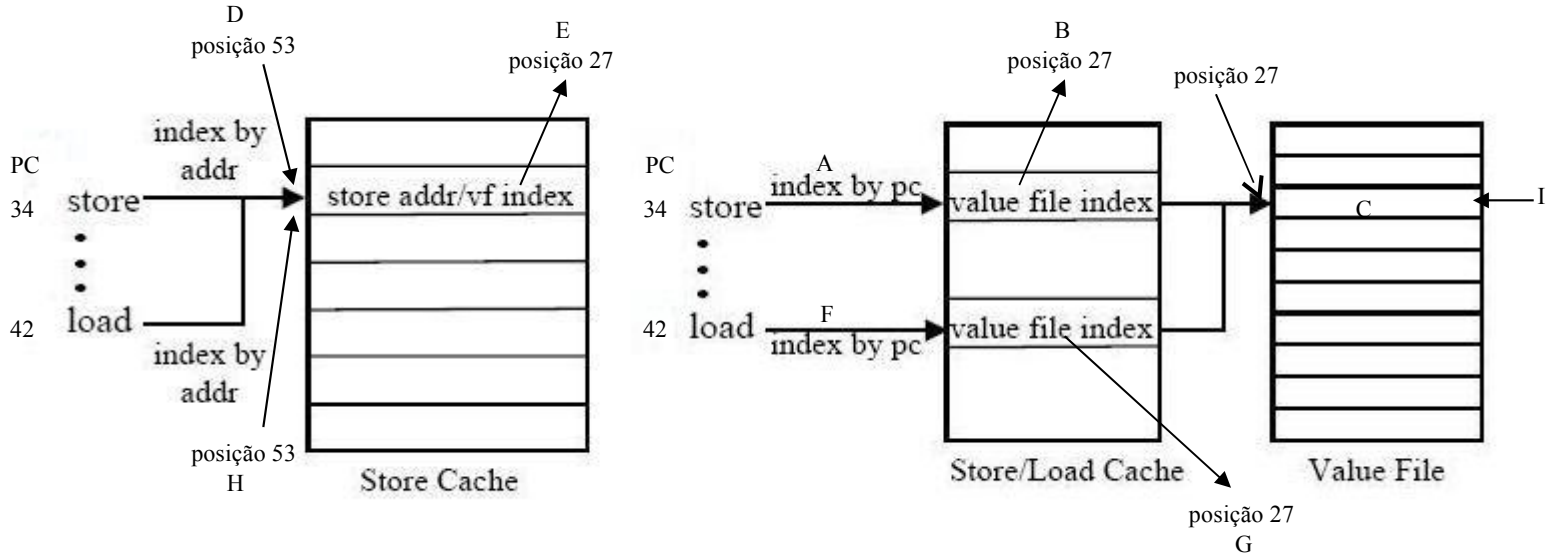

Figura 1. Memory renaming quando o load depende do store anterior [2].

Especulação por Hardware

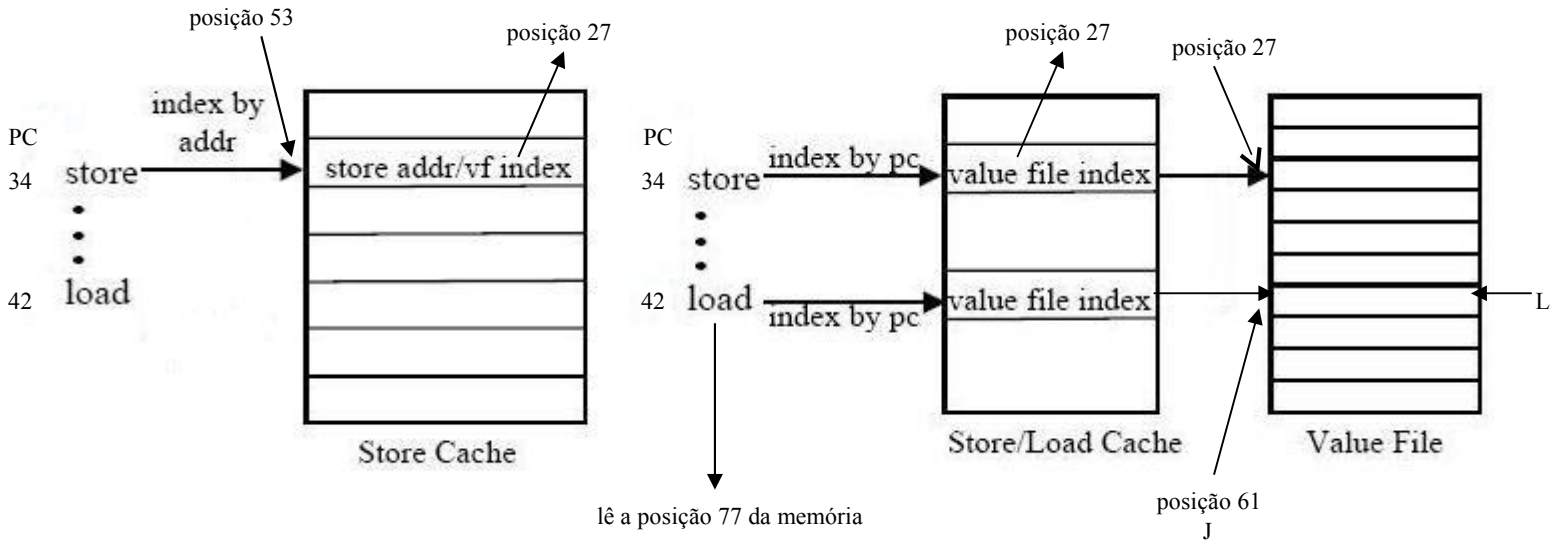

Figura 2. Memory renaming quando o load é independente do store anterior [2].

Especulação por Hardware

- Resultados
  - hybrid value prediction ~ store set dependence prediction > memory renaming
  - store set dependence prediction e hybrid value prediction > store set dependence predicton e hybrid address prediction

Especulação por Hardware

- quatro técnicas (store set dependence prediction, hybrid address prediction, hybrid value prediction e memory renaming) < store set dependence prediction e hybrid value prediction
- Mais promissoras: store set dependence prediction*, hybrid address prediction e hybrid value prediction

Especulação por Software

- Compilador
- speculation check
- mispeculation recovery
- Processadores VLIW: Trimedia e i860
- Processadores EPIC: Itanium e Itanium 2
- General Compiler Framework [4]

Especulação por Software

- Regra heurística simples

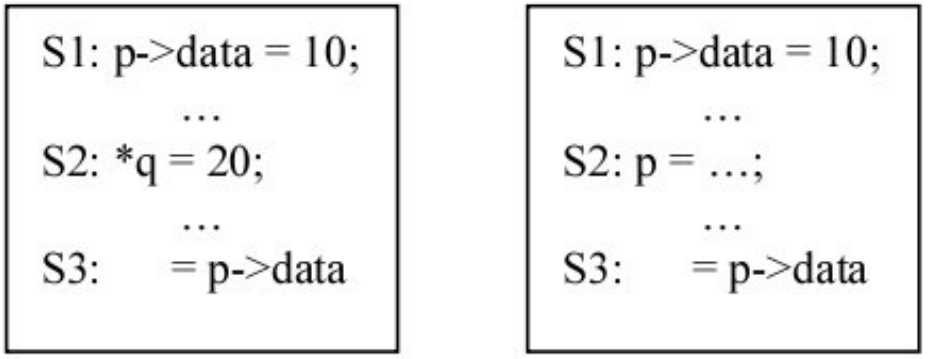

Figura 3. Exemplos da heurística utilizada para determinar se dois acessos à memória são dependentes (a) ou independentes (b) [4].

Especulação por Software

Case 1:	Case 4:	Case 6:
Before DSCM: st [r1] = r2 ld r3 =[r4] After DSCM: S1: flag =1 S2: ld r3 = [r4] S3: if overlap(r1, r4) flag =0 S4: st [r1]= r2 S5: if (flag == 0) S6: ld r3=[r4]	Before DSCM: st [r1] = r2 ld r3 = [r4] After DSCM: S1: flag =1 S2: if overlap(r1,r4) flag =0 S3: if (flag==0) S4: st[r1] = r2 S5: ld r3 = [r4] S6: if (flag == 1) S7: st [r1] = r2	Before DSCM: st [r1] = r2 st [r3] = r4 After DSCM: S1: flag =1 S2: if overlap(r1, r3) flag =0 S3: st [r3] = r4 S4: if (flag == 1) S5: st [r1] = r2

Figura 4. Exemplos onde a especulação possibilita a movimentação de instruções [4].

Especulação por Software

- Arquitetura IA64 (processador Itanium 2)
  - instruções que dão suporte a especulação
    - ld.a (advanced load)
    - chk.a (advanced load check)
  - predication
    - instruções predicadas

Conclusão

- Especulação para “resolver” alias dependency
- Pesquisa futura
  - Especulação por hardware e por software

Referências

[1] August, D., Connors, D., Mahlke, S., Sias, J., Crozier, K., Cheng, B., Eaton, P., Olaniran, Q., Hwu, W. 1998. *Integrated Predicated and Speculative Execution in the IMPACT EPIC Architecture*. *Proceedings of the 25th Annual International Symposium on Computer Architecture* (ISCA'98), 227-237.

[2] Calder, B., Reinman, G. 2000. *A Comparative Survey of Load Speculation Architectures*. *Journal of Instruction-Level Parallelism*, 1-39. University of California, San Diego.

[3] Chrysos, G., Emer, J. 1998. *Memory Dependence Prediction using Store Sets*. *25th Annual International Symposium on Computer Architecture*, 142-153.

[4] Dai, X., Zhai A., Hsu W., Yew P. 2005. *A General Compiler Framework for Speculative Optimizations Using Data Speculative Code Motion*. *International Symposium on Code Generation and Optimization* (CGO), 280-290.

[5] Embry, A., Towles, B., Erez M., 2000. *Memory disambiguation and speculation. Processor Architecture – Advanced Computer Organization. Lecture* #9, 1-4. Stanford University.

[6] Hennessy, J., Patterson, D. 2003. *Computer Architecture: A Quantitative Approach*. 3ª edição, capítulo 3 – *Instruction-Level Parallelism and its Dynamic Exploitation*, capítulo 4 – *Exploiting Instruction Level Parallelism with Software Approaches*.

[7] Itanium from Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Itanium

Data do último acesso: 24/05/2010.

[8] Önder, Soner. *Cost Effective Memory Dependence Prediction using Speculation Levels and Color Sets*. 2002. *International Conference on Parallel Architectures and Compilation Techniques (PACT)*.

[9] Patterson, D. e Hennessy, J. 2007. *Computer Organization and Design – The Hardware/Software Interface*. 3ª edição, capítulo 6 – *Enhancing Performance with Pipelining*, 434-435.