Sistemas de Tempo Real:

Revisão sobre Arquitetura de Processadores

Renan Augusto Starke & Rômulo Silva de Oliveira
Departamento de Automação e Sistemas – DAS – UFSC

romulo@das.ufsc.br
http://www.das.ufsc.br/~romulo
Setembro/2011

Referências

- Computer Systems: A Programmer's Perspective.
  - BRYANT, R. E.; HALLARON, D. R. O.
  - New Jersey: Prentice Hall, 2003. ISBN 0-13-034074-X.
- Modern Processor Design: Fundamentals of Superscalar Processors.
  - SEHN, J. P.; LIPASTI, M. H.
  - New York: McGraw-Hill, 2005. ISBN 0-07-059033-8.

Introdução 1/7

- Três níveis básicos de abstração
  - Arquitetura
  - Implementação
  - Realização

Introdução 2/7

- A arquitetura é conhecida como a arquitetura de conjunto de instruções: Instruction Set Architecture (ISA)
- Especifica o conjunto de instruções que caracterizam o comportamento fundamental do processador
- Todo software deve ser codificado em uma ISA correspondente a tal máquina
- Exemplos de ISA:
  - Intel IA32
  - AMD x86_64
  - PowerPC
  - ARM

Introdução 3/7

- Uma implementação especifica o projeto da arquitetura, referenciado usualmente como microarquitetura
- Exemplos de implementações:
  - PowerC 604
  - Intel P6
- Atributos como projeto do pipeline, memórias cache e previsão de fluxo (brach prediction) fazem parte da implementação

Introdução 4/7

- A realização de uma implementação consiste no encapsulamento físico do projeto
- No caso dos microprocessadores, esse encapsulamento geralmente é um chip
- As realizações podem variar em termos de
  - Frequência
  - Capacidade de memória cache
  - Barramentos
  - Tecnologia de fabricação e empacotamento
- Atributos relacionados com a realização incorporam o die size, encapsulamento, potência, refrigeração e confiabilidade

Introdução 5/7

- O ISA é um contrato entre o software e o hardware
- Torna o desenvolvimento de programas e máquinas uma tarefa independente
- Programas escritos em dada ISA podem rodar em diversas implementações
  - com diferentes níveis de desempenho
  - aumentando a longevidade do software
- ISAs com uma grande quantidade de software já desenvolvidos tendem a ficar no mercado por um bom tempo
  - Exemplo: o ISA Intel IA32

Introdução 6/7

- O ISA também serve como especicação de processadores
- Um ISA define um conjunto de instruções de máquina
  - Operador e um ou mais operandos
- ISAs têm sido diferenciados de acordo com número de operandos que podem ser explicitamente especificados em cada instrução
  - Pode usar um acumulador como operando implícito
  - Pode usar uma pilha para conter os operandos
  - Pilhas são usadas em unidades de ponto flutuante: Floating Point Units (FPU)
- ISAs modernas consideram que os operandos estão gravados em registradores especiais formando o multientry register file do processador
- Todas as operações aritméticas e lógicas são realizadas sobre estes registradores
- Instruções tipo load/store são responsáveis por mover os operandos dos registradores para a memória principal e vice-versa

Introdução 7/7

- O objetivo principal no desenvolvimento de microarquiteturas é aumentar o desempenho
  - Recentemente também reduzir o consumo de energia

$$\frac{1}{desempenho} = \frac{tempo}{programa} = \frac{instrucoes}{programa} \times \frac{ciclos}{instrucao} \times \frac{tempo}{ciclo}$$

- Quanto tempo leva para executar o programa ?
  - Número de instruções necessárias para executar o programa
  - Quandos ciclos de máquina são consumidos por cada instrução
  - Quanto demora cada ciclo de maquina
- Aumentar o desempenho não é uma tarefa trivial
  - Os três termos não são independentes
  - Existem interações complexas entre eles

Pipeline 1/18

- Técnica poderosa para aumentar o desempenho do sistema
- Intel i486 foi a primeira implementação da arquitetura IA32 com pipeline
- A técnica do pipeline particiona o sistema entre múltiplos estágios
  - adicionando buffer entre eles
- A computação original é decomposta em k estagios
- Uma nova instrução pode ser inicializada assim que a anterior atravessar o primeiro estágio
- Ao invés de iniciar uma instrução a cada D unidades de tempo, inicializa-se uma a cada D/k unidades de tempo
  - Os processamentos das k instruções são sobrepostos pelo pipeline

Pipeline 2/18

- A princípio, o ganho de desempenho é proporcional ao comprimento do pipeline
  - Quanto mais estágios, maior desempenho
- Porém, existem limitações físicas relacionadas a frequência as quais determinam a quantidade de estágios que podem ser utilizados

Pipeline 3/18

- O aumento de desempenho em k consiste em três idealismos:
- Sub-computações uniformes
  - A computação a ser realizada é dividida em sub-computações com latências iguais
- Computações idênticas
  - A mesma computação é realizada repetidamente em uma grande quantidade de dados
- Computações independentes
  - As computações são independentes entre si

Pipeline 4/18

- As sub-computações uniformes mantem o idealismo que cada estágio possui o mesmo tempo de execução, ou seja, T/k
- Porém é impossível particionar computações em estágios perfeitos e balanceados
- Exemplo:
- Uma operação de 400ns é dividida em 3 estágios de 125ns, 150ns e 125ns
- Como a frequência do pipeline é determinada pelo estágio mais longo, esta será limitada pelos 150ns
- O primeiro e o terceiro estágio possuem uma ineficiência de 25ns
- E o tempo total da operação utilizando o pipeline será de 450ns ao invés de 400ns

Pipeline 5/18

- As computações idênticas assumem que todos os estágios do pipeline são sempre utilizados em todos os conjuntos de dados
- Esta suposição não é válida quando há execução de múltiplas funções, onde nem todos os estágios são necessários para suporta-las
- Como nem todos os conjuntos de dados precisarão de todos os estágios, estes ficaram esperando
  - mesmo que não precisem executar tal subfunção
  - devido a característica síncrona do sistema

Pipeline 6/18

- As computações independentes assumem que não há dependência de dados ou controle entre qualquer par de computações
- Esta suposição permite que o pipeline opere em fluxo contnuo
  - Nenhuma operação precisa esperar a completude da operação anterior
- Em alguns sistemas esta suposição é válida
- Mas para pipelines de instruções uma operação poderá precisar do resultado da anterior para prosseguir
- Se esta situação ocorre e as duas operações encontram-se em processamento,
  - Uma operação deverá esperar o resultado da anterior
  - Isto ocasiona uma flutuação do pipeline (pipeline stall)
  - Quando isto ocorre, todos os estágios anteriores também deverão esperar, ocasionando uma série de estágios ociosos

Pipeline 7/18

- O desenvolvimento de processadores com pipeline lidam com 3 desafios:
- Sub-computações uniformes: requer balanceamento dos estágios
- Computações idênticas: requer unificação dos tipos de instrução
- Computações independentes: minimização das flutuações

Pipeline 8/18

- Um ciclo de instrução básico pode ser dividido em cinco subcomputações genéricas:

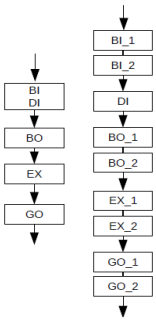
1. busca de instruçõe (BI)	instruction fetch
2. decodificação da instrução (DI)	instruction decode
3. busca dos operandos (BO)	operand's fetch
4. execução da instrução (EX)	instruction execution
5. gravação dos operandos (GO)	operand store

Pipeline 9/18

- As subcomputações genéricas correspondem a uma partição natural de um ciclo de instrução
  - Forma um pipeline genérico de cinco estágios
- Ainda múltiplas subcomputações com baixa latência podem formar um unico estágio
- ou ainda alguns dos cinco estagios genéricos também podem ser agrupados formando um melhor balanceamento do pipeline

Pipeline 10/18

- Exemplo



Pipeline 11/18

- Suponha que a computação de 5 estágios genéricos leve 300ns
- Suponha que o tempo de ciclo do pipeline de 4 estágios seja 80ns
- Suponha que o tempo de ciclo do pipeline de 9 estágios seja 40ns

- A latência do pipeline de 4-estágios é de 80ns*4 = 320ns
- A latência do pipeline de 9-estágios é de 40ns*9 = 360ns

- Deste ponto de vista, parece que o pipeline de 4-estágios é mais eficiente
- Porém em termos de throughtput, tem-se 300ns/40ns = 7,5
- Contra 300ns/80ns = 3,75

- Deve-se considerar os vários fatores no projeto de um pipelines

Pipeline 12/18

- Para realizar uma computação, precisa-se de três tarefas genéricas:
  - Operações aritméticas
  - Movimentação de dados
  - Sequenciamento de instruções

- Em uma arquitetura RISC moderna, o conjunto de instruções emprega tipos de instruções dedicadas para cada uma das tarefas genéricas:
  - Instruções da Unidade lógica e aritmética (ULA)
    - Restritas aos registradores
  - Instruções de memória (load/store): movimentação de dados
    - Únicas que podem acessar a memória
  - Instruções de fluxo (branch): para o manipulação do fluxo de controle

Pipeline 13/18

- Especificação das instruções aritméticas/ponto flutuante

Subcomputação	Instrução Inteira	Instruçao ponto-flutuante
BI	Buscar (acessar I-cache)	Buscar (acessar I-cache)
DI	Decodificar	Decodificar
BO	Acessar registradores	Acessar Registrados PF
EX	Realizar operação	Realizar operação PF
GO	Regravar nos registradores	Regravar nos registradores

Pipeline 14/18

- Especificação das instruções de acesso à memória

Subcomputação	Instrução *Load*	Instrução *store*
BI	Buscar (acessar I-cache)	Buscar (acessar I-cache)
DI	Decodificar	Decodificar
BO	Acessar registradores (endereço base) Gerar endereço efetivo (base + deslocamento) Acessar memória (acessar D-cache)	Acessar Registradores (operando e endereço base)
EX		
GO	Regravar nos registradores	Gerar endereço efetivo (base + deslocamento) Acessar memória (acessar D-cache)

Pipeline 15/18

- Especificação das instruções de controle de fluxo

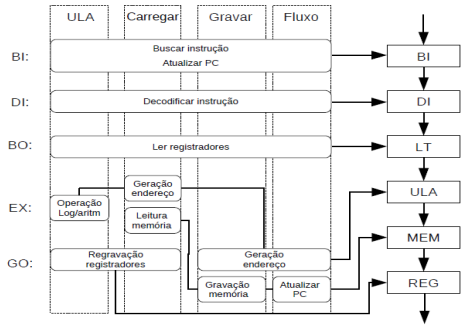
Subcomputação	Salto incondicional	Salto condicional
BI	Buscar (acessar I-cache)	Buscar (acessar I-cache)
DI	Decodificar	Decodificar
BO	Acessar registradores (endereço base) Gerar endereço efetivo (base + deslocamento)	Acessar Registradores (endereço base) Gerar endereço efetivo (base + deslocamento)
EX		Avaliar condição
GO	Atualizar PC com endereço alvo	Se condição for verdadeira atualizar PC com endereço alvo

Pipeline 16/18

- A analise das especificações das instruções proporciona uma verificação da semântica de cada instrução
- Apesar delas compartilharem subcomputações em termos de busca e decodicação, elas necessitarão de recursos diferenciados

- Exemplo de um pipeline genérico de seis estágios
  - Na coluna da esquerda as cinco tarefas básicas
    - Busca (BI)
    - Decodificação de instrução (DI)
    - Busca dos operandos (BO)
    - Execução (EX)
    - Gravação dos operandos (GO)
  - A direita um possível pipeline compartilhando as unidades
  - Na parte superior, os 4 tipos de instruções

Pipeline 17/18



Pipeline 18/18

- Os seis diferentes estágios significam que haverá seis instruções ao mesmo tempo em execução
- O register file necessita suportar
  - duas leituras (dois operandos no estágio LT)
  - e uma escrita (estágio REG)
  - simultaneamente

- A I-cache deverá suportar uma leitura a cada ciclo
- A D-cache deverá suportar uma leitura ou escrita (estágio MEM)

- Se ocorrer qualquer falta de cache (cache miss), haverá uma flutuação no pipeline
- Neste modelo, o acesso a cache possui uma latência de um ciclo
  - Com cache maior e lógica do processador complexa, manter uma latência de um ciclo torna-se difícil

Flutuações do Pipeline 1/6

- Dadas duas instruções i e j sendo j precedido de i, j pode ser dependente de i em três situações

- Instrução j requer um operando que está na imagem de i
- Dependência read-after-write (RAW) ou dependência verdadeira
- A instrução j não pode executar até a completude de i

- i : R3 <- R1 op R2
  j : R5 <- R3 op R4

Flutuações do Pipeline 2/6

- Dadas duas instruções i e j sendo j precedido de i, j pode ser dependente de i em três situações

- A instrução j irá modificar uma variável que é operando de i
- Dependência write-after-read (WAR) ou dependência anti-dados
- Requer que a instrução j não complete antes da execução de i, ou i irá utilizar o operando errado

- i : R3 <- R1 op R2
  j : R1 <- R4 op R5

Flutuações do Pipeline 3/6

- Dadas duas instruções i e j sendo j precedido de i, j pode ser dependente de i em três situações

- As instruções i e j irão modificar a mesma variável
- Dependência write-after-write ou dependência de saída
- A instrução j não pode completar antes de i

- i : R3 <- R1 op R2
  j : R3 <- R6 op R7

Flutuações do Pipeline 4/6

- Alem das dependências de dados existem as dependências de controle
- Dadas as instruções i e j, com j precedido por i, a execução de j depende do resultado de i
- As dependências de controle são resultados da estrutura de controle de fluxo do programa
- Saltos condicionais geram incerteza no sequenciamento das instruções

Flutuações do Pipeline 5/6

- Como a semântica do programa requer que as dependências sejam respeitadas
- e a execução de instruções por um pipeline pode facilmente romper o sequenciamento
- deve haver mecanismos de identicação e resolução de dependências
- Uma potencial violação das dependências é conhecida como pipeline hazzards
- As dependências verdadeiras, anti-dados e de saída devem ser identificadas e resolvidas através de mecanismos de hardware:
  Pipeline interlock
- Este mecanismo envolve a criação de flutuações no pipeline em certos estágios bem como controle da lógica em caminhos de atalho

Flutuações do Pipeline 6/6

- Um pipeline de k-estágios pode potencialmente alcançar um desempenho de um fator k em relação ao projeto normal
- Com a construção de pipelines longos as penalidades relacionadas com dependências e suas resoluções aumentam
- Penalidades envolvendo busca de operandos, memória e operações aritméticas geralmente são resolvidas por caminhos de atalho
  - Incorporam poucos ciclos
- Penalidades relacionadas com instruções de controle de fluxo podem incorporar muitos ciclos de ociosidade
- No projeto de processadores busca-se um equilibrio entre a quantidade de estágios, frequência de operação, desempenho e custo

Processadores Superescalares 1/9

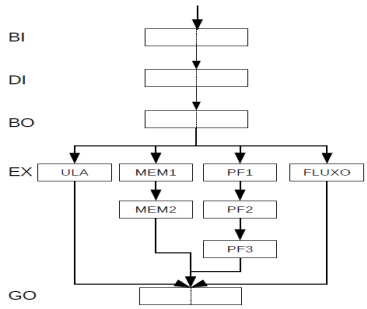
- Máquinas superescalares são capazes de avançar múltiplas instruções pelos estágios do pipeline
  - Elas incorporam múltiplas unidades funcionais
- Aumentam a capacidade de processamento concorrente a nível de instrução aumentando o throughput
- Podem executar instruções em ordem diferente do programa
- A ordem sequencial das instruções no programa implica em algumas precedências desnecessárias entre as instruções
- A capacidade de executar as instruções fora de ordem
  - alivia a imposição sequencial
  - permite mais processamento paralelo sem modicação do programa original

Processadores Superescalares 2/9

- Um pipeline de k-estágios teoricamente pode aumentar o desempenho em um fator k
  - paralelismo temporal
- De modo alternativo, pode-se atingir o mesmo fator de velocidade empregando k cópias do mesmo pipeline processando k instruções em paralelo
  - paralelismo espacial
- Processadores superescalares utilizam as duas técnicas de paralelismo

Processadores Superescalares 3/9

- Exemplo de pipeline superescalar



Processadores Superescalares 4/9

- No exemplo quatro unidades funcionais são implementadas e o estágio BO envia as instruções para as unidades corretas
  - Conforme o tipo de instrução
- Cada unidade funcional pode ser customizada para um tipo de instrução resultando em um hardware eficiente
- Cada tipo de instrução possui sua própria latência e utiliza todos os estágios de sua unidade funcional
- Se as dependências entre instruções são resolvidas antes do envio às unidades funcionais, não há flutuações no pipeline
- Este esquema permite controle independente e distribuído de cada sub-pipeline de execução

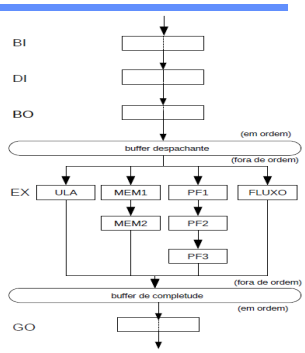
Processadores Superescalares 5/9

- Para evitar ciclos de flutuação desnecessários:
  é permitido que novas instruções sejam adiantadas
  quando há ciclos de flutuação por dependências
  ou faltas de memória
- Este adiantamento pode mudar a ordem de execução das instruções com relação ao código do programa
- A execução fora de ordem permite que estas sejam executadas assim que os operandos estejam disponveis

- Um pipeline paralelo que permite execução fora de ordem é chamado de pipeline dinâmico
  - Emprega buffers multi-entrada
  - As instruções entram e saem em ordens diferentes

Processadores Superescalares 6/9

- Exemplo de pipeline dinâmico



Processadores Superescalares 7/9

- Com um pipeline largo, o estágio de busca acessa s instruções na I-cache a cada ciclo de máquina
- A organização da I-cache deve ser larga o suficiente para que
  - cada linha armazene s instruções
  - toda a linha possa ser acessada em cada ciclo de instrução

- Contudo, não é possível garantir a banda máxima de busca
  - Desalinhamento das instruções na I-cache
  - Presença de instruções de controle de fluxo
- O problema da maximização da busca de instruções é resolvido de maneira diferente em cada microarquitetura
  - por hardware ou
  - por diretivas de alinhamento em tempo de compilação

Processadores Superescalares 8/9

- O estágio de decodificação em uma ISA RISC deve identificar dependências das instruções e verificar quais instruções podem ser despachadas em paralelo
- Deve também identificar mudanças no controle de fluxo para promover uma resposta rápida do estágio de busca

- Para uma ISA CISC a tarefa de decodificação pode ser ainda mais complexa
  - Precisa de múltiplos sub-estágios
- Pentium e AMD K5 empregam dois subestágios para decodificação de instruções do IA32
- Intel Pentium Pro precisa de um total de cinco ciclos de máquina para acessar a I-cache e decodificar uma instrução
- A utilização de instruções de tamanho variável (ISA CISC) impõe uma decodificação sequencial
  - Uma instrução deve ser decodificada e seu tamanho conhecido para que a próxima instrução possa ser identificada

Processadores Superescalares 9/9

- Na decodicação das instruções CISC,
  o decodicador traduz as instruções da arquitetura para operações de baixo nível
  - Elas são executadas pelo hardware
- Estas operações internas lembram as instruções RISC
  - Micro-instruções
  - Utilizam o modelo de memória load/store

- Na microarquitetura AMD são conhecidas como operações RISC
  - ROPs - RISC operations

- No Intel são chamadas de micro operações
  - u-ops - micro-operations

- Cada instrução IA32 é traduzida como uma ou mais micro operações
  - Na média, uma instrução possui 1,5 a 2,0 u-ops

- Para muitos processadores superescalares o hardware de decodificação é bastante complexo e requer particionamento em múltiplos estágios

Técnicas de Previsão de Fluxo 1/3

- As dependências de controle induzem uma quantidade signicativa de flutuações
- Brach Prediction: técnicas de previsão do controle de fluxo
- O comportamento desse tipo de instrução é altamente previsível
- A técnica chave para minimizar as penalidades e maximizar o desempenho é especular tanto o endereço quanto a condição do salto nas instruções de controle
- Obviamente que a utilização de técnicas especulativas requerem mecanismos de recuperação de erros de previsão

Técnicas de Previsão de Fluxo 2/3

- A especulação do endereço envolve o uso de um buffer
- Branch Target Buffer (BTB)
- Armazena o endereço alvo das instruções de controle anteriores
- O BTB é uma memória cache pequena acessada durante o estágio de busca de instruções
- Cada entrada no BTB contem dois campos:
  - endereço da instrução - Branch Instruction Address (BIA)
  - endereço alvo - Branch Target Address (BTA)
- O BTB é acessado concorrentemente com a I-cache
- Quando o contador de programa (CP) bate com o BIA, o campo BTA é utilizado para a busca da próxima instrução se a condição é especulada como verdadeira para o salto

Técnicas de Previsão de Fluxo 3/3

- A versão não especulativa da execução continua para avaliação
  - Buscada da I-cache e executada normalmente
- A avaliação do endereço e condições são comparadas com a versão especulada
- Caso concordem,
  - A previsão estava correta, não há penalidade
- Caso contrário,
  - Um erro de previsão ocorreu, deve-se iniciar a recuperação
- O resultado da execução não especulativa atualiza o BTB
- Um erro de previsão é custoso
  - Ciclos perdidos no processamento de instruções não utilizadas
  - Limpeza dos efeitos das instruções falsamente previstas
- Cerca de 82,5% a 96,2% das previsões são corretas

Memória Cache 1/10

- A memoria cache foi criada para explorar as propriedades de localidade espacial e temporal
- Cria-se uma ilusão de memória rápida de grande capacidade
- O princípio básico é colocar uma memória
  - de baixa capacidade
  - rápida
  - de alto custo
- entre o processador e a memória principal
  - Memória principal mais barata e com grande capacidade
- Mesmo a memória cache de nível primário (L1) consegue satisfazer 90% das referências
  - Se está na cache é um acerto (hit)
  - Se não está na cache é uma falta (miss)

Memória Cache 2/10

- Cache é administrada em blocos ou linhas e não em bytes
  - Define a granularidade na qual a cache opera
- Cada bloco é uma sequência contínua de bytes e inicia em alinhamento comum
- O menor tamanho utilizável de um bloco possui o tamanho natural das palavras do processador
  - 4-bytes para máquinas de 32-bits
- Cache pode ter vários níveis
  - Para dar conta das grandes diferenças de custo e velocidade
- Cada nível implementa um mecanismo que permite uma procura
  - de baixa latência
  - para vericar se dado bloco está ou não contido na cache

Memória Cache 3/10

- Devido ao alinhamento dos blocos, os bits menos signicativos do endereco são sempre zero
- Se é preciso acessar um byte diferente do primeiro, os bits menos signicativos são utilizados como deslocamento para encontrar o byte correto
- Em termos de localização de blocos há três possíveis organizações
- Mapeamento direto
- Mapeamento totalmente associativo
- Mapeamento associativo por conjunto

Memória Cache 4/10

- Mapeamento direto
- Um endereço particular pode residir apenas em uma única localização na cache
- Esta localização é normalmente determinada extraindo n bits do endereço e os utilizando para índice direto das 2^n possveis localizações na cache

- Mapeamento de muitos endereços para somente um lugar na cache
  - Cada localização necessita de uma etiqueta (tag) que correspondem aos bits restantes do bloco gravado naquela localização
- Em cada procura, o hardware precisa
  - Ler a etiqueta e comparar com o endereco da referência
  - Para determinar se foi acerto ou falta

Memória Cache 5/10

- Totalmente associativa
- Permite um mapeamento de múltiplos endereços para múltiplas localizações na cache
  - Qualquer endereço de memória pode residir em qualquer localização na cache
- Todas as localizações da cache devem ser verificadas para encontrar os dados
- Cada linha da cache deve possuir uma etiqueta com o endereço dos dados que hospeda
  - Para o hardware comparar e detectar uma falta ou acerto

Memória Cache 6/10

- Associatividade por conjunto
- Existe um mapeamento de múltiplos endereços para algumas localizações na cache
- Em cada procura, um subconjunto de bits de endereço são utilizados para geração do índice
  - Como no caso de mapeamento direto
- Contudo, o índice corresponde a um conjunto de entradas que são procuradas em paralelo em busca da etiqueta correta
  - Existe associatividade por conunto

Memória Cache 7/10

- Cada nível de cache possui uma capacidade finita
- Deve haver uma política para despojar ocupantes atuais possibilitando espaço para blocos com referências mais recentes
- Política de substituição: algoritmo para identicar um candidato para ser despojado

- Em mapeamento direto o problema é trivial
  - Somente um lugar para cada endereço

- Na associatividade completa ou por conjunto, precisa escolher
  - Um novo bloco pode ser colocado em várias possíveis localizações
- Três políticas básicas:
  - FIFO (primeiro a entrar, primeiro a sair)
  - LRU (menos recentemente utilizado)
    - Aproximada por não-mais-recentemente-utilizado (not-most-recently-used, NMRU)
  - Aleatoriamente

Memória Cache 8/10

- A utilização de memória cache implica em múltiplas cópias do bloco

- Enquanto há apenas leitura não ocorre nenhum problema

- Para escrita deve haver mecanismos para atualização dos blocos nos demais níveis da memória

- Existem basicamente dois mecanismos conhecidos como write-through e write-back

Memória Cache 9/10

- O mecanismo write-through simplesmente propaga a atualização para os níveis inferiores
- É fácil implementar
- Não existe ambiguidade sobre a versão dos dados

- Exige muita demanda de barramento
- Disparidade da frequência entre o processador e a memória principal torna impossível a utilização em todos os níveis da hierarquia

- Este mecanismo ainda deve especicar se vai ou não alocar espaço na cache quando ocorre uma falta em escrita de um bloco

- A política write-allocate implica em alocar o bloco na cache
- A política write-no-allocate evita a instalação do bloco na cache
  - Realiza esta operação apenas em faltas de leitura

Memória Cache 10/10

- O mecanismo write-back prorroga a atualização das outras cópias até quando precisa manter a ordem de correção
- Uma ordem de prioridade implícita é utilizada para encontrar a cópia mais atualizada do bloco
  - somente esta cópia está atualizada
- Se um bloco é encontrado no nível mais elevado da cache
  - esta copia é atualizada
  - as cópias dos níveis mais baixos permanecem desatualizadas
- Se uma copia é apenas encontrada em nível mais baixo
  - Esta é promovida ao topo e é ali atualizada
- As cópias atualizadas são marcadas com um dirty bit
  - Indica que os outros níveis possuem cópias desatualizadas
- Quando um bloco sinalizado é substituido, este é atualizado para o nível inferior
- A cópia do nível inferior é agora marcada com dirty bit
  - Caso despejado, precisa atualizar os níveis ainda mais inferiores

Paginação 1/4

- O mapeamento da memória lógica para a física deve ser gravado em uma memória de tradução
- O sistema operacional é responsável por atualizar esse mapeamento quando necessário
- O processador deve acessar a memória de tradução para determinar o endereço físico de cada acesso lógico que ocorre
- Cada entrada de tradução deve conter
  - o endereço lógico
  - o endereço físico
  - bits de permissão para leitura, escrita e execução
  - bits de referência e modicação
  - bit de inibição de cache (acesso via barramento sempre)

Paginação 2/4

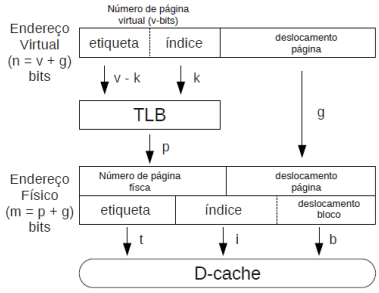
- As memórias de tradução são chamadas de tabelas de páginas
- Podem ser organizadas em tabelas de páginas diretas
  - Forward page tables
  - Tabelas de páginas diretas são mais simples
  - Possui uma entrada para cada bloco do espaço de endereçamento lógico
  - Estrutura grande, com muitas entradas não utilizadas
  - As tabelas diretas são implementadas em múltiplas seções
- Ou tabelas de páginas invertidas
  - Inverted page tables ou hashed page tables
  - Existem apenas entradas suficientes para mapear todos os endereços da memória física
  - Pode ser armazenada mais confortavelmente
  - É basicamente uma tabela hash onde o endereço lógico é a chave
  - No caso de memória virtual, ela não comporta os endereços de disco

Paginação 3/4

- Estrutura chamada de translation lookaside buffer (TLB) é usada
- TLB contem um número pequeno de entradas de páginas
  - Tipicamente 64 a 256
  - Organizadas por associatividade completa
- O processador provê um rápido hardware de pesquisa associativa para converter referências à memória
- Faltas no TLB resultam em acesso à tabela de páginas
- Arquiteturas MIPS, Alpha e Sparc empregam um software TLB miss handler
- Arquiteturas PowerPC e Intel IA-32 empregam um hardware TLB miss handler
  - Incluem uma máquina de estados para acesso à memória por hardware para procura da tabela de páginas

Paginação 4/4

- A cache de dados armazena porções da memória principal
- A TLB armazena porções da tabela de páginas



Coerência de Cache em Multiprocessadores 1/4

- Atualmente protocolos de atualização de cache não são mais utilizados devido o consumo excessivo de banda nas interconexões
- Para sistemas multiprocessados com memória compartilhada são utilizados protocolos de invalidação

- Neste protocolo apenas um processador pode escrever em uma linha de cache a qualquer momento
- Esta política é reforçada certicando-se que a linha que será modificada seja a única cópia no sistema
  - invalidando as outras presentes nos outros processadores

Coerência de Cache em Multiprocessadores 2/4

- O protocolo de invalidação requer no mínimo dois estados para cada linha de cache: modicado (M) e inválido (I)
- No estado inválido, o endereço requisitado não está presente
  - Deve ser buscado da memória
- No estado modificado, o processador sabe que a sua cópia é exclusiva no sistema
  - Pode realizar leituras e escritas na linha

- Obviamente que cada linha modificada deve ser regravada na memória
- Um otimização simples incorpora um dirty bit no estado da linha da cache
  - Linhas que são exclusivas do processador (conhecido como estado E)
  - Linhas que são exclusivas e estão desatualizadas após uma escrita (estado M)
- Este protocolo é conhecido como MEI

Coerência de Cache em Multiprocessadores 3/4

- Pelo protocolo MEI não é permitido que linhas de cache existam em mais de um processador ao mesmo tempo
- Para resolver este problema e permitir leituras simultâneas, é incluído o estado compartilhado: estado S (Shared)
- Este estado indica que uma ou mais cópias remotas existem
- Para modicar uma linha no estado S
  - Primeiramente modifica-se o estado para M, invalidando as cópias remotas

- Protocolo MESI

Coerência de Cache em Multiprocessadores 4/4

- Aprimoramento do protocolo MESI
- Adição do estado owned (O)
- Resulta no protocolo MOESI

- O estado O é atingido quando acontece uma leitura remota de um bloco (dirty) no estado M
- O estado O signica que múltiplas cópias válidas do bloco existem
  - o requisitor remoto recebeu uma cópia válida satisfazendo a leitura
  - enquanto o processador local também mantem uma cópia
- É diferente do estado convencional S pois os dados não foram atualizados na memória
- Estado também conhecido como shared-dirty
  - o bloco é compartilhado, mas aguarda a atualização na memória
- Os dois processadores passam de O para S quando a memória for atualizada

Comentário Final

- Estes são apenas alguns dos elementos da arquitetura de um processador contemporâneo
- São aqueles que mais dificultam a obtenção de WCET

- Se não tiver:
  - Pipeline (fora de ordem, previsão de fluxo, flutuações)
  - Cache
  - Paginação
- Fica muito mais fácil
  - Mas não é realista

- Aula baseada em livros da graduação
  - Complexidade dos detalhes é muito maior

Sumário