Paralelismo ao Nível das Instruções

Luís Nogueira

luis@dei.isep.ipp.pt

Departamento Engenharia Informática

Instituto Superior de Engenharia do Porto

Como melhorar a performance?

- Equação de performance

$$\text{tempo CPU} = \frac{\text{número de instruções}}{\text{IPC * frequência relógio}}$$

- Alternativas
  - Diminuir número de instruções do programa
  - Aumentar frequência de relógio
  - Aumentar número de instruções executadas por ciclo (IPC)

Mais do que apenas MHz

	SPECint95	SPECfp95
195 MHz MIPS R10000	11.0	17.0
400 MHz Alpha 21164	12.3	17.2
300 MHz UltraSPARC	12.1	15.5
300 MHz Pentium-II	11.6	8.8
300 MHz PowerPC G3	14.8	11.4
135 MHz IBM POWER2	6.2	17.6

- IBM POWER2 a 135 MHz supera Alpha 21164 a 400 MHz em vírgula flutuante!
- Como é possível?
  - A frequência de relógio não é tudo
  - Interessa quanto trabalho é efectuado por ciclo de relógio

Como aumentar o IPC?

- Vamos analisar duas abordagens
  - Tentar aumentar a frequência interna das fases
    - Superpipelining
  - Tentar executar instruções em paralelo
    - Arquitecturas multiple-issue

Superpipelining

- Pipeline simples
  - Ciclo de relógio limitado pela fase mais lenta
- Subdivisão em fases mais simples
  - Maior frequência de relógio
  - Pipeline mais profundo
- Cada instrução demora mais ciclos de relógio
  - Ciclo de relógio menor
  - Pipeline cheia $\rightarrow$ 1 instrução por ciclo
  - Aumento de performance!

Superpipelining

- Frequência interna dos estágios
  - Múltiplo da frequência de relógio
- Exemplo
  - Frequência de relógio 100 MHz
  - Frequência interna 200 MHz

```
IF (10ns)   →  IF1 (5ns) + IF2(5ns)
ID (10ns)   →  ID1 (5ns) + ID2(5ns)
EX (10ns)   →  EX1 (5ns) + EX2(5ns)
MEM (10ns)  →  MEM1 (5ns) + MEM2(5ns)
WB (10ns)   →  WB1 (5ns) + WB2(5ns)
```

Superpipelining

- Muito usado em unidades funcionais especializadas
  - Pentium 4 leva técnica ao extremo (20 fases)
- Processadores CISC normalmente possuem pipelines mais profundos
  - Trabalho extra na descodificação de instruções

Superpipelining - Limitações

- Mais instruções no pipeline
  - Maior dependências entre instruções
- Velocidade dos circuitos
- Aumenta exigência de largura de banda para a memória
  - CPU necessita de mais instruções por unidade de tempo

Arquitecturas multiple-issue

- Replicar unidades funcionais
  - Executar várias instruções em paralelo
- Como distribuir n instruções por k unidades funcionais?
  - Sabendo que existem dependências entre as instruções
- Static multiple-issue
  - Decisões efectuadas à priori pelo compilador
- Dynamic multiple-issue
  - Decisões efectuadas durante a execução pelo hardware

Arquitecturas multiple-issue

- Performance máxima do pipeline
  - Execução de blocos de instruções sequenciais
- Instruções de salto alteram fluxo sequencial
- Bloco básico
  - Segmento de código sequencial limitado por instruções de salto
  - Em média possui apenas 5 instruções

Arquitecturas multiple-issue

- Instruções num bloco básico podem ainda conter dependências
  - Entre elas ou com instruções noutro bloco
- Limitações para extrair máximo paralelismo ao nível das instruções
- Execução especulativa é vital para arquitecturas multiple-issue
  - Performance depende da eficácia do previsor de saltos

Execução especulativa

- Compilador ou CPU tentam “adivinhar” resultado do salto
  - Execução continua no endereço previsto
  - Pipeline executa blocos de instruções sequenciais
  - Fluxo de execução especulado com base na previsão de saltos
- Previsão pode estar errada
  - Método para verificar se a previsão foi bem sucedida
  - Método para descartar efeitos de instruções que não deviam ter sido executadas (má previsão)

Execução especulativa - Recuperação de má previsão

- Compilador
  - Insere instruções adicionais para verificar resultado da previsão
  - Fornece rotina para descartar instruções inválidas
- Hardware
  - CPU armazena resultado das instruções até conhecer decisão de salto
  - Previsão bem sucedida $\rightarrow$ resultados escritos nos registos ou memória
  - Previsão incorrecta $\rightarrow$ instruções são descartadas

Previsão de saltos

- Previsão dinâmica
  - Baseada na história da instrução
  - Implementada em hardware
  - Adaptativa e mais eficiente
- Previsão estática
  - Baseada na estrutura do programa
  - Implementada pelo compilador
  - Limitada a instruções com comportamento previsível
  - Restantes saltos dependem de informação de profiling ou interpretação abstracta

Previsão dinâmica de saltos

- Prever dinâmicamente o resultado do salto
  - Previsão muda se o salto alterar o seu comportamento durante a execução do programa
- Branch prediction buffer
  - Esquema de 1 bit
  - Esquema de 2 bits
  - Correlacting branch predictor
  - Tournament branch predictor

Branch prediction buffer

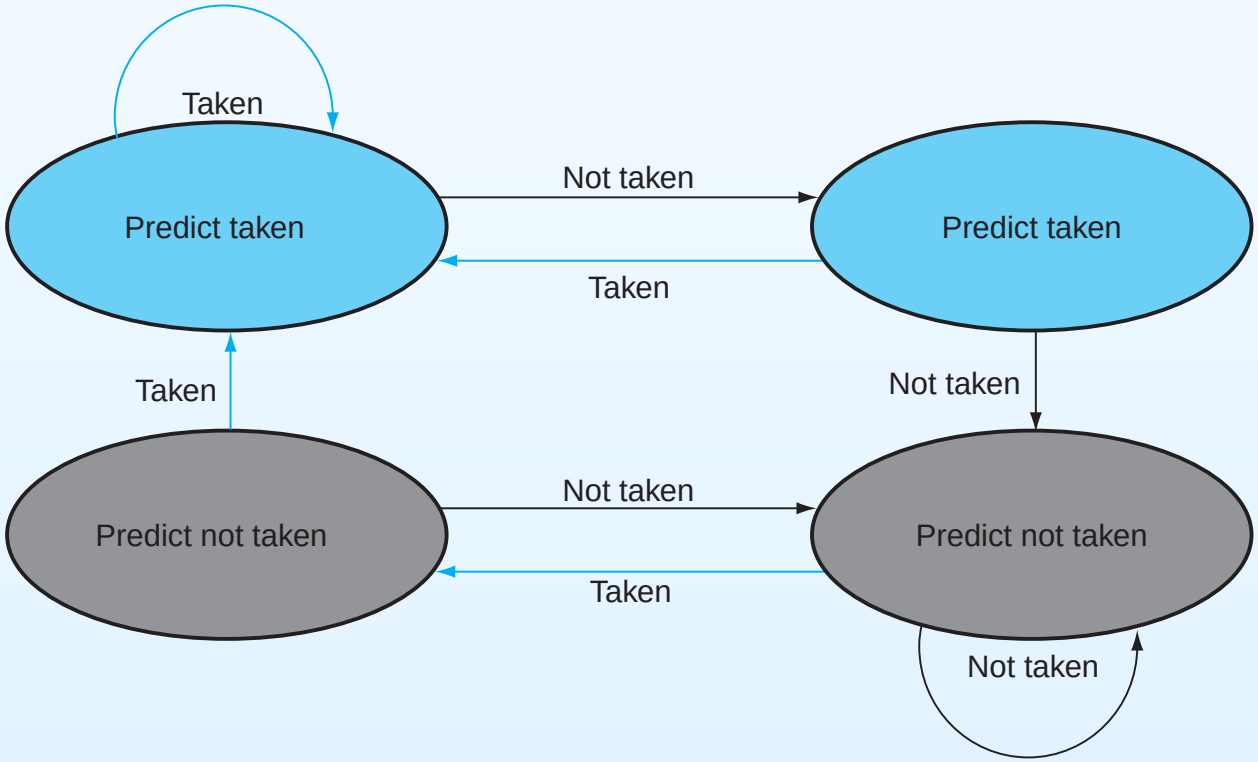
- Também designado por “branch history table”
- Pequena memória indexada pelos bits menos significativos do endereço da instrução de salto
  - Tem de ser rápida. Caso contrário era melhor esperar pela decisão
- Contém 1 ou mais bits indicando se o salto foi ou não efectuado recentemente
- Performance depende do esquema usado no previsor de saltos

Esquema de 1 bit

- Um bit para guardar resultados anteriores
- Prevê erradamente a entrada e saída de um ciclo
- Ciclo com n iterações
  - Saída
    - Má previsão é inevitável
    - Salto foi efectuado n vezes seguidas
  - Entrada
    - Bit indica que salto anterior não foi efectuado
    - Execução anterior de outro ciclo

Esquema de 2 bits

- Previsão tem de falhar 2 vezes antes de ser alterada
- Saltos que indicam forte probabilidade são erradamente previstos apenas uma vez



Esquemas mais complexos

- Saltos podem ter comportamentos diferentes dependentes do fluxo de execução
  - Distinguir fluxos de execução
  - Usar previsores de 2 bits para cada salto
- Correlacting branch predictor
  - Combina comportamento local de um salto com informação global
- Tournament branch predictor
  - Vários previsores de 2 bits para cada salto
  - Mecanismo de selecção escolhe que previsor usar

Arquitecturas dynamic multiple-issue

- Processadores conhecidos como “super-escalares”
- Executar várias instruções em paralelo
  - Distribuição dinâmica de instruções por hardware
  - Execução especulativa de blocos básicos
- Permite manter compatibilidade com software antigo
  - Sem necessidade de recompilação
  - IPC pode ser melhorado com compilador eficiente

Distribuição dinâmica por hardware

- Pipeline dividido em 3 unidades principais
  - Instruction Fetch and Issue Unit
  - Conjunto de várias unidades funcionais
  - Commit Unit
- Instruction Fetch and Issue Unit
  - Obtém instruções
  - Descodifica-as
  - Envia-as para a unidade funcional correspondente
- Commit Unit
  - Validação dos resultados calculados especulativamente

Distribuição dinâmica por hardware

- Reservation station
  - Buffer que armazena operandos numa unidade funcional
- Quando todos os operandos são conhecidos
  - Unidade funcional processa operandos
  - Envia resultado para
    - Reservation stations que esperam valor
    - Commit unit (forwarding)
- Instruções são executadas fora de ordem
  - Instrução bloqueada não atrasa execução da instrução seguinte

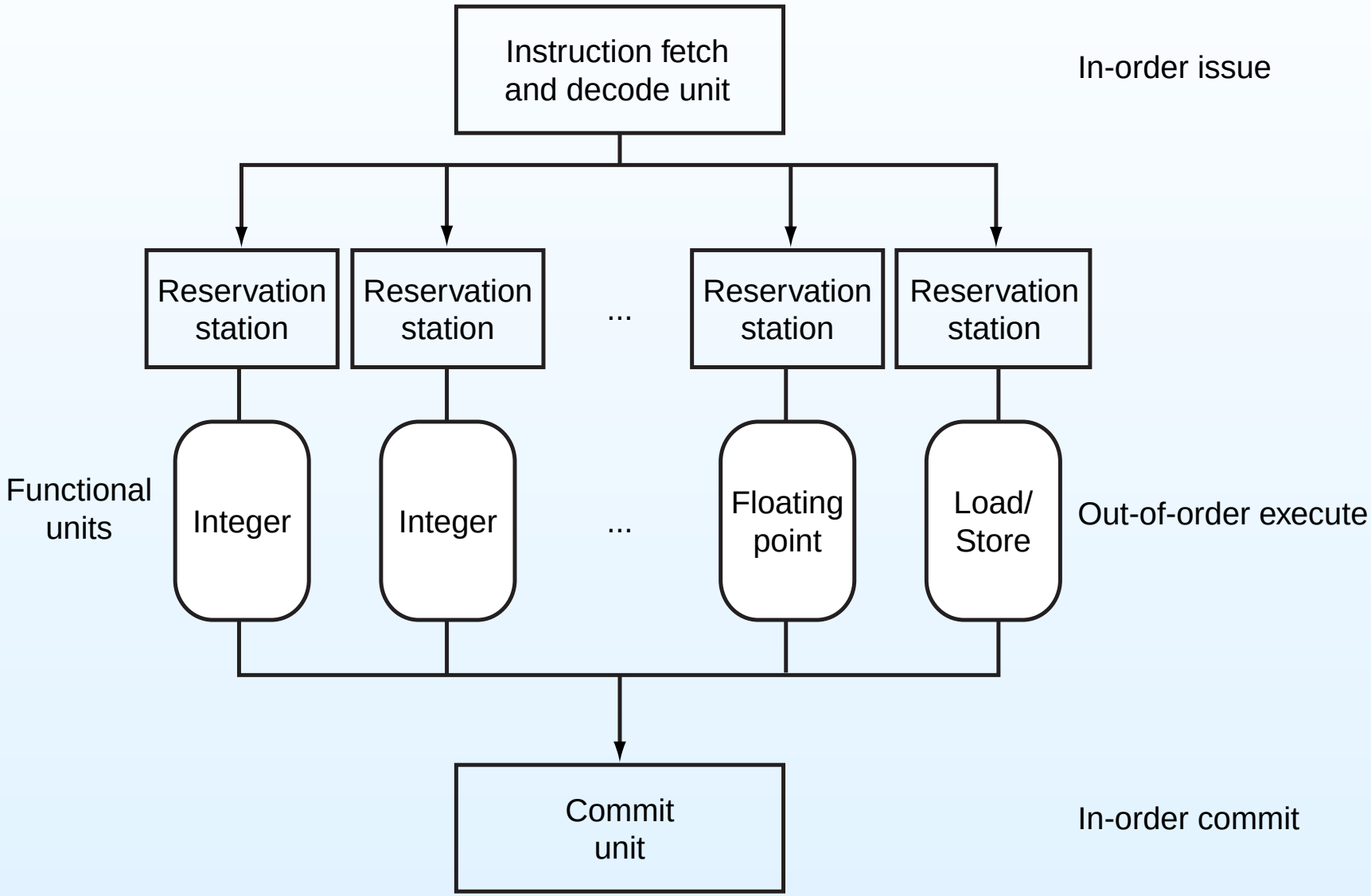
Distribuição dinâmica por hardware

- Re-order buffer
  - Buffer na Commit Unit que reordena instruções executadas fora de ordem
  - Resultados colocados em registos de rascunho (register renaming)
- Número máximo de instruções no re-order buffer
  - Número máximo de instruções que podem ser executadas especulativamente

Distribuição dinâmica por hardware

- Quando o resultado da instrução de salto é conhecido
  - Foi bem previsto?
    - Re-order buffer valida (commit) resultados
    - Valores são escritos nos registos finais
  - Foi mal previsto?
    - Resultados são descartados
- Validação dos resultados é efectuada por ordem
  - “In-order commit” preserva semântica inicial do programa

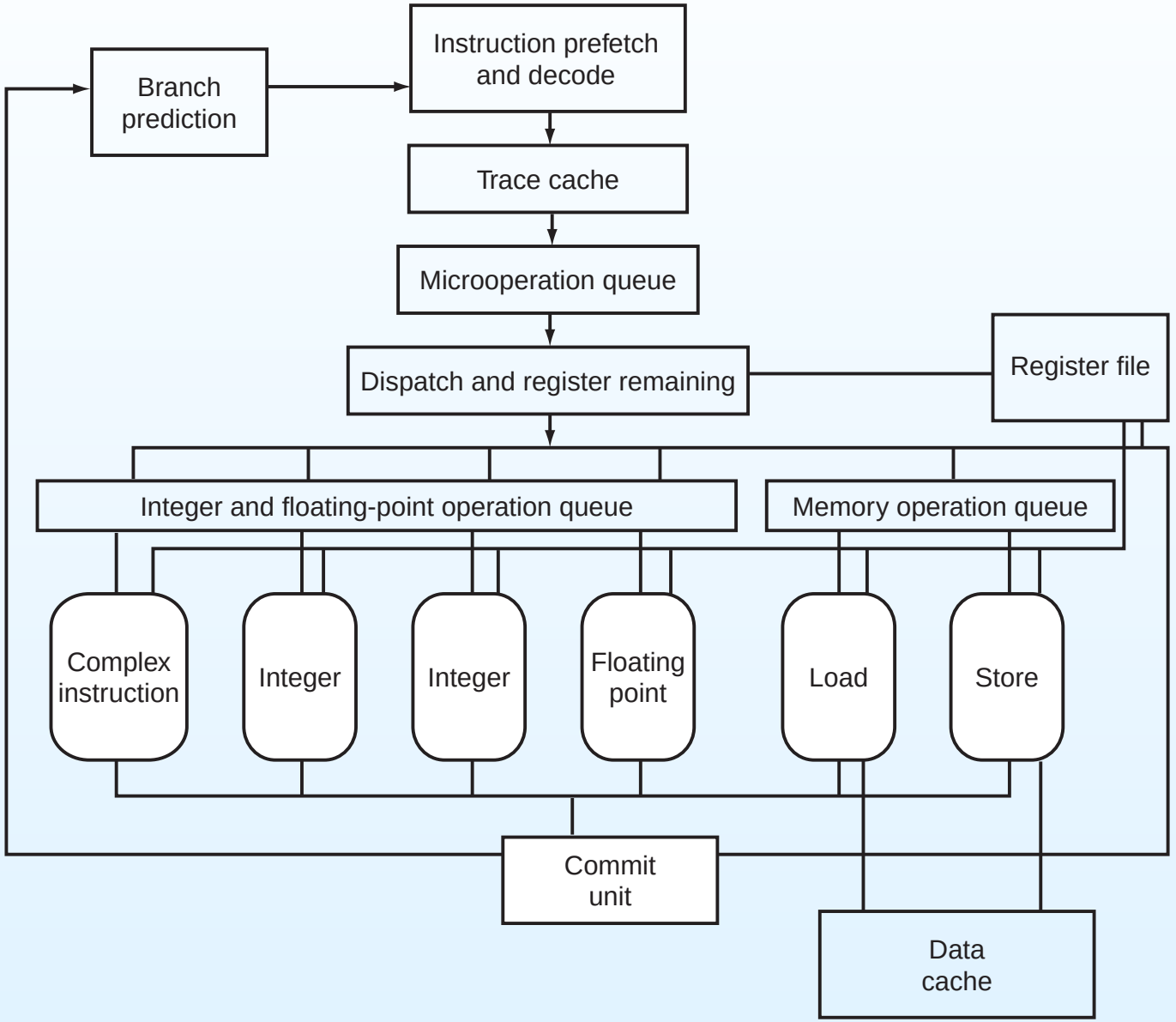
Distribuição dinâmica por hardware



Pentium 4

- Tradução de instruções IA-32 em micro-instruções
  - Micro-instruções executadas num superpipeline (20 fases)
- Distribuição dinâmica por 7 unidades funcionais
  - Execução fora de ordem
- Register renaming
  - 8 registos IA-32 $\rightarrow$ 128 registos rascunho

Pentium 4


Branch prediction
Instruction prefetch and decode
Trace cache
Microoperation queue
Dispatch and register remaining
Register file
Integer and floating-point operation queue
Memory operation queue
Complex instruction
Integer
Integer
Floating point
Load
Store
Commit unit
Data cache

Arquitecturas static multiple-issue

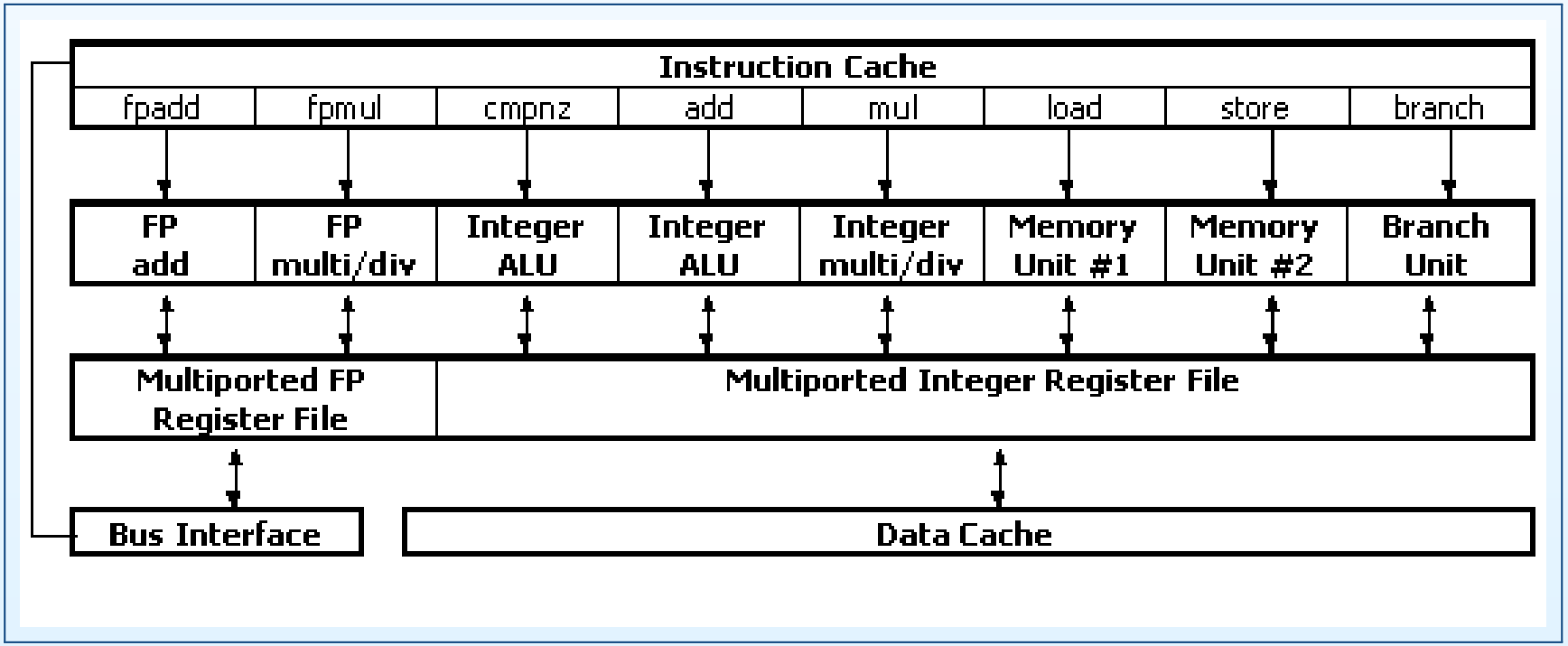
- Transferir para o compilador
  - Análise de dependências
  - Distribuição das instruções pela unidades funcionais
- Pacotes de n instruções independentes
  - Enviadas directamente para as unidades funcionais
- Pacote pode ser visto como uma única grande instrução
  - Very Large Instruction Word (VLIW)
- Compromisso entre
  - Eficiência da distribuição das instruções
  - Simplicidade da sua descodificação

Very Large Instruction Word

- Distribuição das instruções é trivial
  - Lógica de controlo simplificada
- Sucessor natural do RISC
  - Move complexidade do hardware para o compilador
- Compilador tem de ser sofisticado
  - Loop unrolling, register renaming, ...
  - Agrupa instruções sem dependências num pacote
  - Se não existe paralelismo suficiente slots ficam vazios
  - Execução especulativa usada agressivamente

Very Large Instruction Word

- Exemplo de pacote com 8 instruções
  - 2 fp + 2 int + 2 lw/sw + 1 salto
- Pacote sem dependências executado em paralelo



Limitações das arquitecturas VLIW

- Depende de um compilador poderoso
- Natureza estática das optimizações do compilador
- Requisitos exigentes de largura de banda para a memória
- Tamanho dos binários é elevado e incompatível entre diferentes implementações

Very Large Instruction Word

- Primeiros protótipos nos anos 80
  - Cydrome Cydra-5
- Contemporâneos
  - Transmeta Crusoe / Efficeon (2ª geração)
  - Intel IA-64 Explicitly Parallel Instruction Computer (EPIC)
  - NXP Trimedia
  - Texas Instruments C6000

Transmeta Crusoe

- Camada de software isola o hardware
- Recompilação dinâmica de código binário
- Em tempo de execução compilador gera código nativo
  - Code Morphing
- Agrupa instruções nativas em pacotes VLIW de 128 bits
  - Executa até 4 IPC

Transmeta Crusoe

- Após algumas iterações
  - Código nativo mantido em cache
  - Optimizado com base em informação recolhida dinâmicamente
- Pode executar binários escritos em qualquer ISA
  - Apenas necessita versão adequada de software de morphing
- Destina-se principalmente a sistemas embebidos
  - Ajuste dinâmico da frequência do relógio
  - Baixas necessidades energéticas

Paralelismo ao nível das instruções - Resumo

Superpipelining

- Pipeline mais profunda e maior frequência interna dos estágios
- Exige maior velocidade da memória
- Aumentam problemas originados por
  - Dependências entre instruções
  - Latência no acesso à memória
  - Instruções de salto
- Técnica limitada pela velocidade dos circuitos
  - Necessidade de armazenar valores intermédios nos registos do pipeline

Paralelismo ao nível das instruções - Resumo

Arquitecturas multiple-issue

- Replicar componentes funcionais
- Executar instruções em paralelo no mesmo ciclo de relógio
- Recorrem à execução especulativa
- Exige bom previsor de saltos
  - Má previsão $\rightarrow$ perda de performance
- Duas abordagens para a distribuição das instruções
  - Static multiple-issue (compilador)
  - Dynamic multiple-issue (hardware)

Paralelismo ao nível das instruções - Resumo

Dynamic multiple-issue

- Instruction Fetch and Issue Unit
  - Obtém, descodifica e distribui instruções
  - Processadores CISC actuais descodificam instruções complexas em conjunto de instruções simples
- Execução fora de ordem
  - Operandos armazenados em *reservation stations*
  - Forwarding para unidades que esperam valores
  - Envio de resultados para *commit unit*

Paralelismo ao nível das instruções - Resumo

Dynamic multiple-issue

- Commit Unit
  - *Register renaming* cria registos temporários
  - *Re-order buffer* preserva semântica original do programa
  - Instrução de salto foi bem prevista
    - Validação (commit) dos resultados
    - Registos temporários $\rightarrow$ registos do CPU
  - Instrução de salto mal prevista
    - Resultados descartados

Paralelismo ao nível das instruções - Resumo

Static multiple-issue

- Transferir para o compilador
  - Análise de dependências
  - Distribuição das instruções pela unidades funcionais
- Very Large Instruction Word (VLIW)
  - Pacotes de n instruções sem dependências
  - Pacote pode ter slots vazios
- Reduz substancialmente a complexidade do hardware
- Exige compilador poderoso

Paralelismo ao nível das instruções - Resumo

Dynamic Multiple-issue vs Static Multiple-issue