Processadores Superescalares

Paralelismo no Nível de Instruções

Tradução, revisão e adaptação por Paulo S. L. de Souza
Adaptação 2 Regina Helena Carlucci Santana

Processadores Superescalar

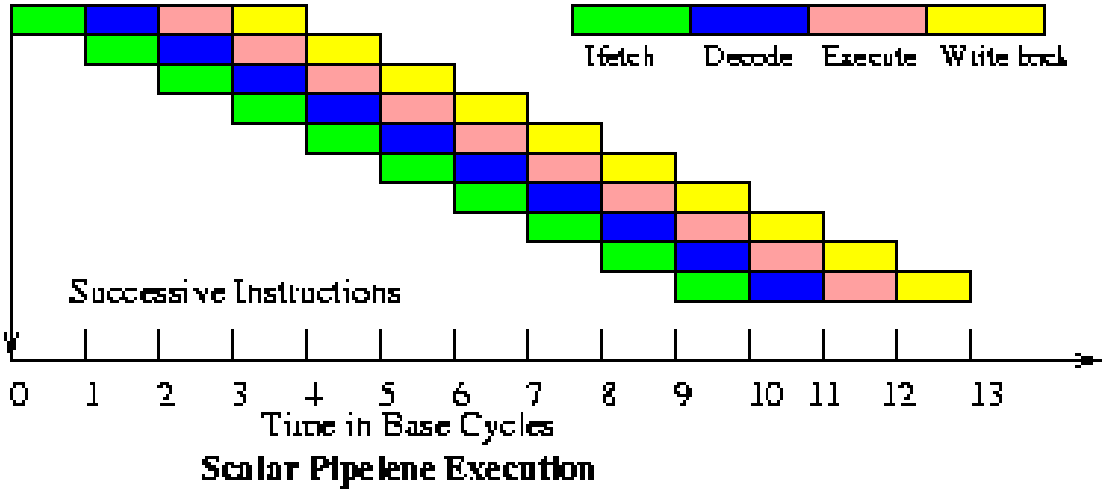
- Processadores Escalares
- Processadores Superescalares
- Processadores pipelined
- Processadores Superpipelined
- Processadores VLIW

Processadores Superscalar

- Processadores Escalares
  - Uma instrução por Ciclo
  - Uma instrução terminada por ciclo
- Processadores Superescalares
  - Múltiplas instruções consideradas em um único ciclo
  - Múltiplas instruções podem ser terminadas em um ciclo
  - Desenvolvidos como uma alternativa a processadores vetoriais

Processadores Superscalar

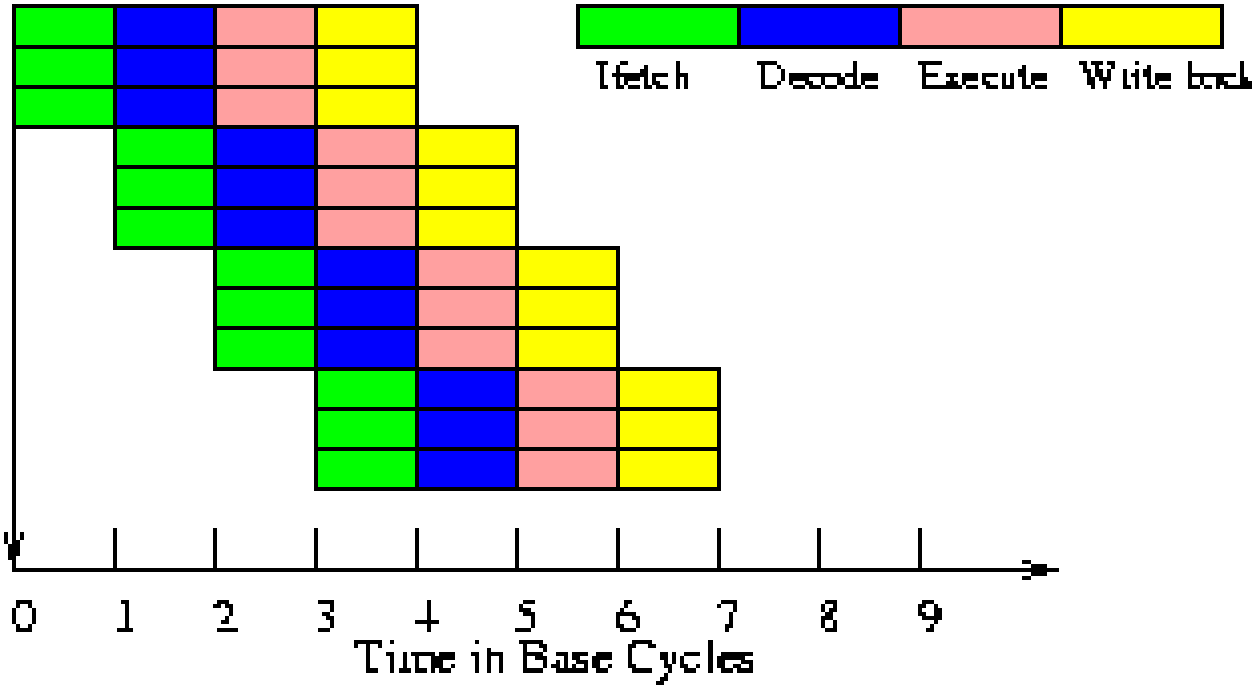
Processamento pipeline de 4 estágios



Scalar Pipelene Execution

Processadores Superscalar

Processamento superscalar com grau = 3 e com pipeline de 4 estágios



A Superscalar Processor of Degree 3

Processadores Superscalar

Processador escalar com grau m - **pode** finalizar até m instruções por ciclo

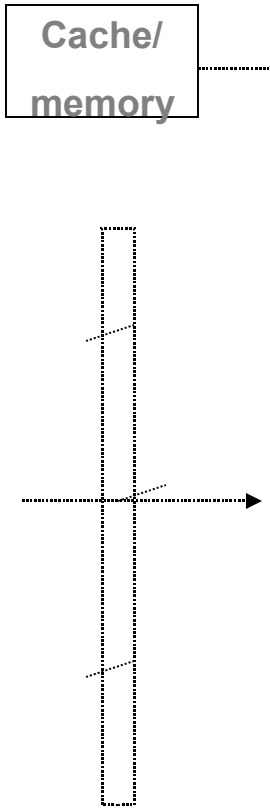
Depende da dependência entre dados e conflito por recurso

Processadores VLIW

VLIW – Very Large Instruction Word

- Explora paralelismo em nível de instrução
- Instruções de 128-1024 bits
- Cada instrução consiste de múltiplas instruções independentes
- Diversas unidades funcionais interligadas por um único registrador compartilhado

VLIW X Superescalar

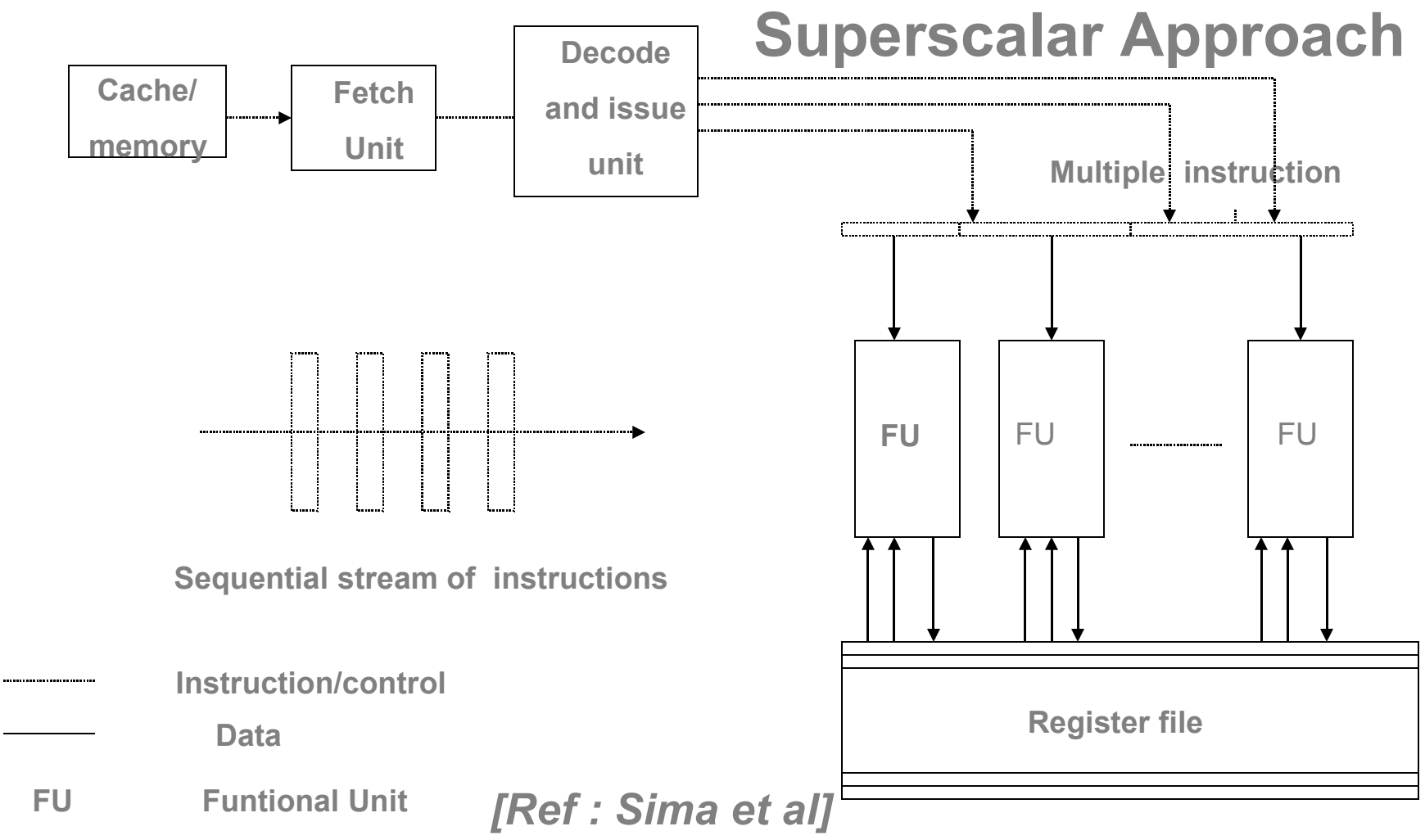


multi-operation instruction

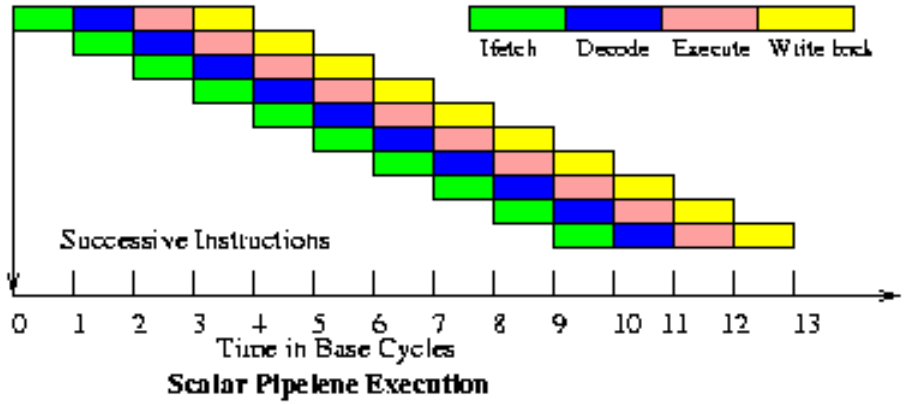
[Ref : Sima et al]

VLIW Approach

VLIW X Superescalar

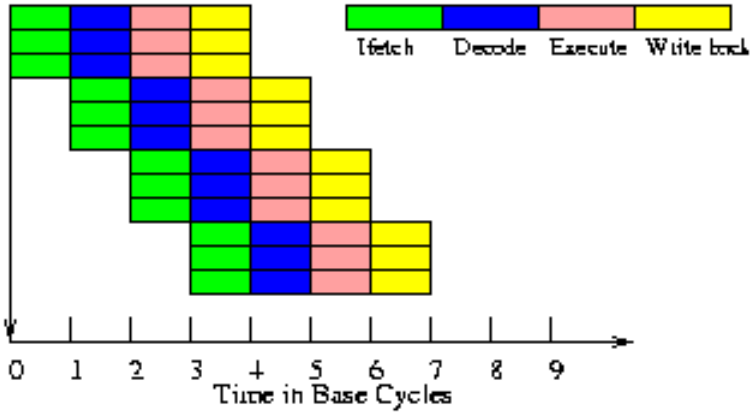


Tempos para execução de Instruções

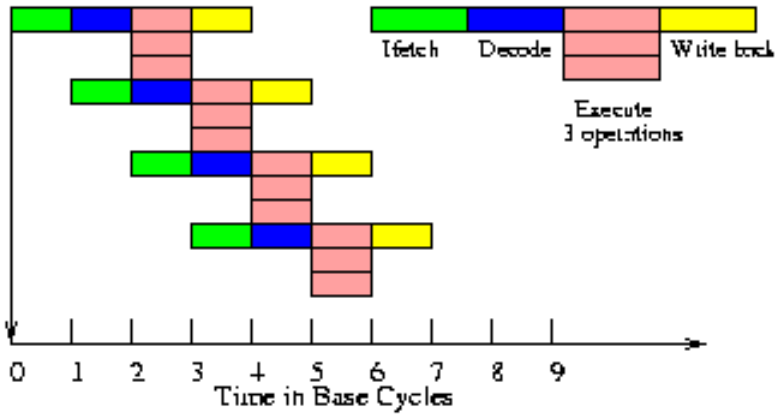


Scalar Pipelene Execution

[Ref : Hwang et al]]



A Superscalar Processor of Degree 3



VLIW Execution with Degree 3

Processadores VLIW X Superescalar

Porque VLIW é menos popular que superescalar

- Compatibilidade entre códigos binários
- Mesmo compilador pode ser utilizado para escalar e superescalar

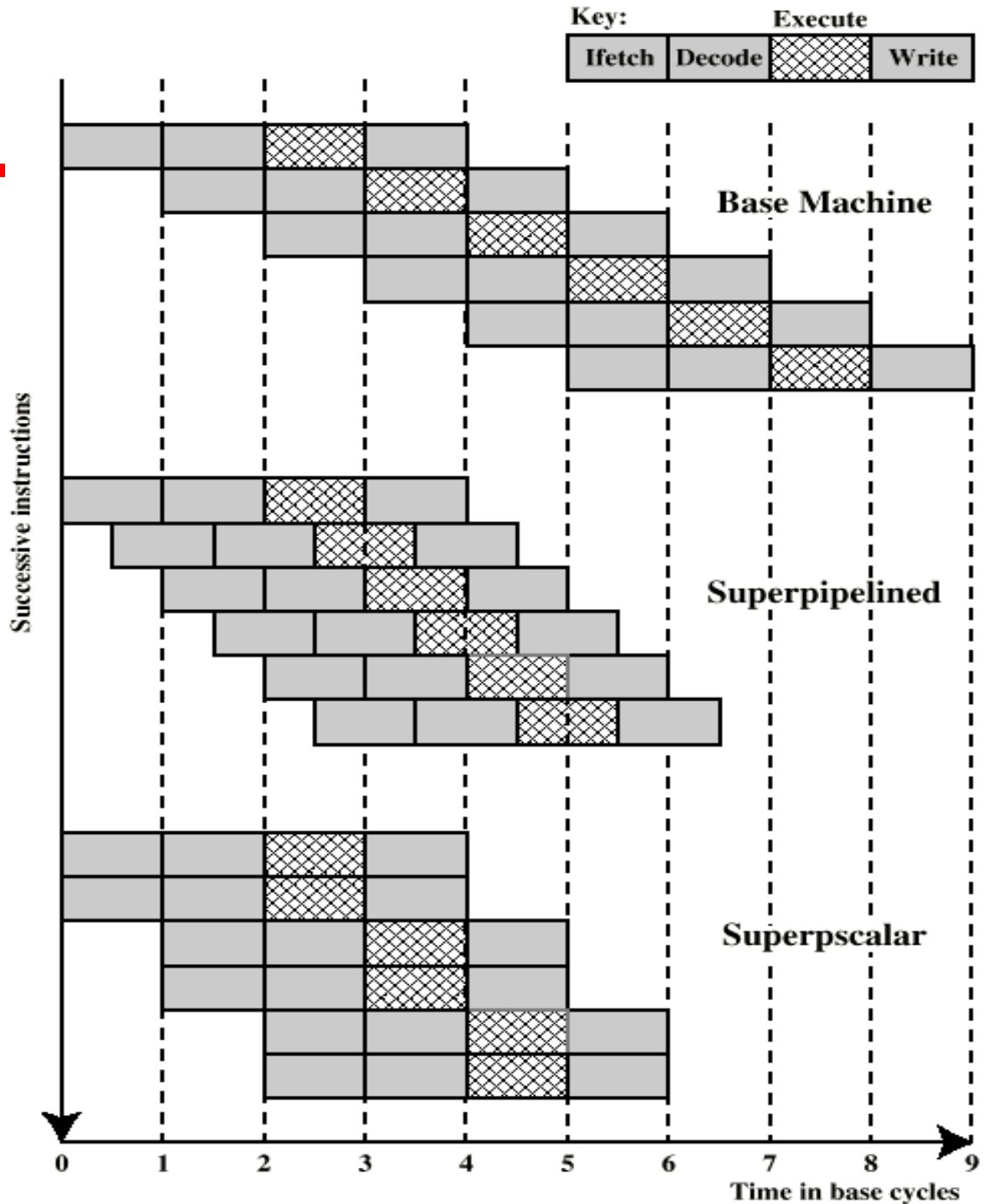
Exemplo de Superescalar: IBM RS/6000

Exemplo de VLIW:Crusoe TM 5400

Superpipeline

- É um *pipeline* com muitos estágios
- Estágios necessitam tempos de ciclo menores
  - normalmente menos que a metade
- Velocidade interna de *clock* duplicada
  - executa duas "atividades"por ciclo de *clock* externo
- Superescalar permite executar a busca em paralelo

Superescalar x superpipeline



Pontos importantes

- Processador superescalar:
  - emprega vários *pipelines* de instrução independentes
  - cada pipeline com seus estágios, executando instruções diferentes simultaneamente
  - novo nível de paralelismo: diversos fluxos de instrução cada vez
  - é o paralelismo no nível de instruções

- Processador precisa buscar várias instruções:
  - instruções próximas que sejam independentes e possam ser executadas em paralelo (ao mesmo tempo)
  - problemas com a dependência de dados
  - identificadas dependências, execução pode ser feita fora de ordem

- Dependências de dados podem ser solucionadas com:
  - registradores adicionais e renomeação de referências a registradores

- Dependências de controle
  - RISCs puros empregam desvio atrasado
  - técnica não é adequada para processadores superescalares
    - são usados métodos tradicionais de previsão de desvio

O que significa superescalar?

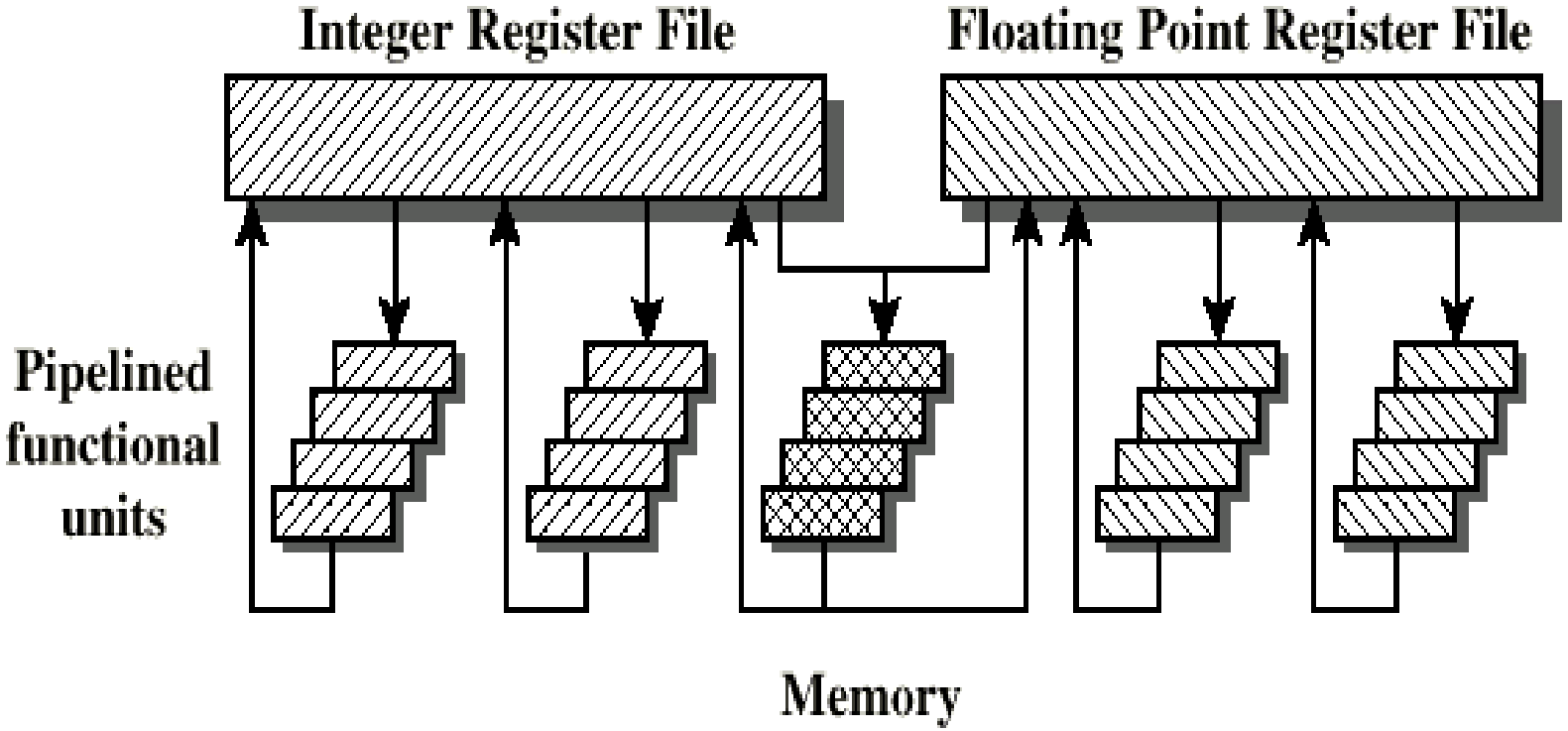
- Instruções comuns
  - aritméticas, *load/store* e desvios condicionais podem ser iniciados e executados independentemente
- Aplicável igualmente a RISC e CISC
  - na prática é implementado usualmente em máquinas RISC

Por que superescalar?

- Maioria das operações realizadas em valores escalares
- Melhora desempenho dessas operações para melhorar o desempenho total
- Proposta foi aceita rapidamente
  - máquinas superescalares comerciais surgiram apenas +/- 2 anos após o surgimento do termo superescalar
  - máquinas RISCs comerciais levaram +/- 8 anos

Organização superescalar genérica

- Várias unidades funcionais organizadas em *pipeline*
- Ex.: duas op com inteiros
  duas op com ponto flutuante
  uma op de carga/armazenamento



Limitações

- Paralelismo no nível de instrução
  - nível que as instruções podem ser executadas em paralelo
- Otimizações baseadas:
  - no compilador e em técnicas de hardware
- Limitações intrínsecas:
  - dependência de dados verdadeira (escrita-leitura)
  - dependências de desvio
  - conflitos no uso de recursos
  - dependência de saída (escrita-escrita)
  - antidependência (leitura-escrita)

Dependências de dados verdadeira

- Outros nomes:
  - dependência de fluxo ou escrita-leitura

```
ADD r1, r2          #(r1 := r1+r2;)
MOVE r3,r1          #(r3 := r1;)
```

- Considerando uma máquina superescalar **grau 2**:
  - instruções podem ser obtidas e decodificadas em paralelo
  - segunda instrução não pode ser executada enquanto a primeira não terminar

Dependências de desvio

- Não pode executar instruções posteriores a um desvio condicional em paralelo com instruções anteriores ao desvio

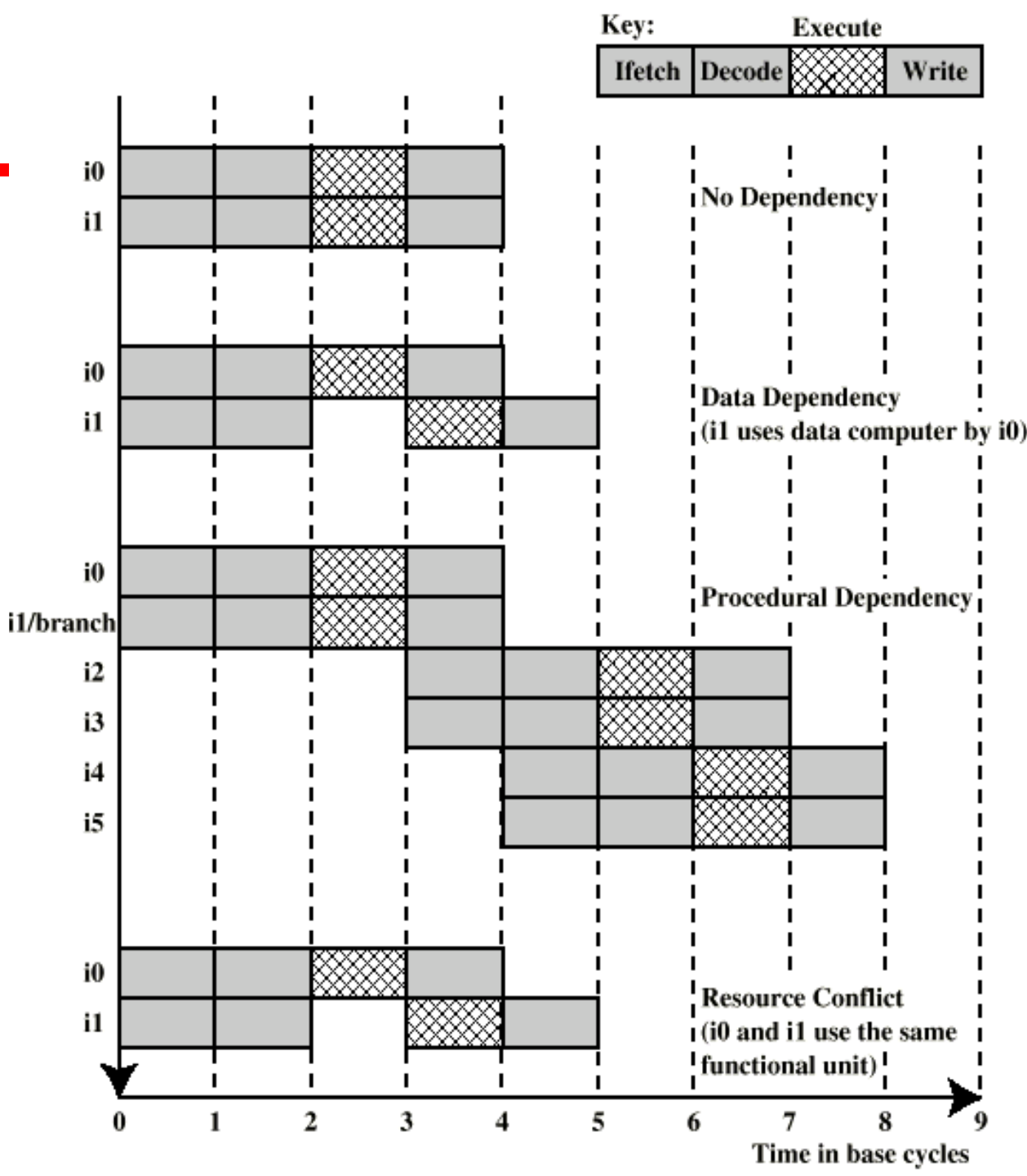
- Tamanho variável das instruções é um agravante:
  - se o tamanho das instruções não é fixo
    - instruções precisam ser decodificadas parcialmente
      - determina quantos acessos posteriores serão necessários
    - isto impede buscas simultâneas
  - mais uma razão para usar instruções de tamanho fixo
    - máquinas RISC

Conflitos no uso de recursos

- Ocorre quando instruções tentam acessar o mesmo dispositivo ao mesmo tempo
  - ex.: duas instruções aritméticas

- Uma possível solução:
  - duplicar recursos com conflito
    - ex.: ter duas unidades aritméticas

Efeito das dependências



Questões de projeto

- Paralelismo no nível de instruções & de máquina

- Paralelismo no nível de instruções
  - ocorre quando instruções em seqüência são independentes e a execução pode ser sobreposta
  - dependências de desvio e de dados determinam a possibilidade ou não

- Paralelismo na máquina
  - habilidade de obter vantagem do paralelismo no nível de instrução
  - determinado pelo número de *pipelines* em paralelo
    - número de instruções buscadas e executadas ao mesmo tempo

Políticas para iniciar instruções

- Para aproveitar o paralelismo de máquina
  - UCP deve identificar paralelismo no nível de instruções
    - coordenar: buscas, decodificações e execuções
- Para isso são usadas políticas de iniciação de instrução
- As políticas essencialmente:
  - examinam algumas instruções à frente
  - tentam localizar instruções para iniciar no *pipeline*
- Também determinam a ordem para:
  - buscar/executar instruções (iniciar instruções)
  - entregar as instruções executadas
    - ou seja, efetivar a alteração em registradores e na memória

Iniciação ordenada e entrega ordenada

- Busca e executa instruções na ordem que elas ocorrem
- Não é muito eficiente
- Podem buscar mais de uma instrução
- Se necessário as instruções devem sofrer paradas
  - tratamento de dependências

- Exemplo considerado:

  I1 necessita dois ciclos na fase de execução

  I3 e I4 concorrem pela mesma unidade funcional

  I5 depende do valor produzido por I4

  I5 e I6 concorrem pela mesma unidade funcional

Diagrama para iniciação ordenada e entrega ordenada

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
I3	I4	I1					3
	I4			I3	I1	I2	4
I5	I6			I4			5
	I6		I5		I3	I4	6
			I6				7
					I5	I6	8

I1 necessita dois ciclos na fase de execução
I3 e I4 concorrem pela mesma unidade funcional
I5 depende do valor produzido por I4
I5 e I6 concorrem pela mesma unidade funcional

Iniciação ordenada e entrega desordenada

- Dependência de saída

```
R3:= R3 + R5;        # I1
R4:= R3 + 1;         # I2
R3:= R5 + 1;         # I3
R7:= R3 op R4        # I4
```

  - I2 depende do resultado de I1 e I4 depende de I3
    - dependência de dados verdadeira
  - se I3 executar antes de I1
    - os resultados de I1 e de I4 estarão errados
    - **dependência de saída (escrita/escrita)**

Diagrama para iniciação ordenada e entrega desordenada

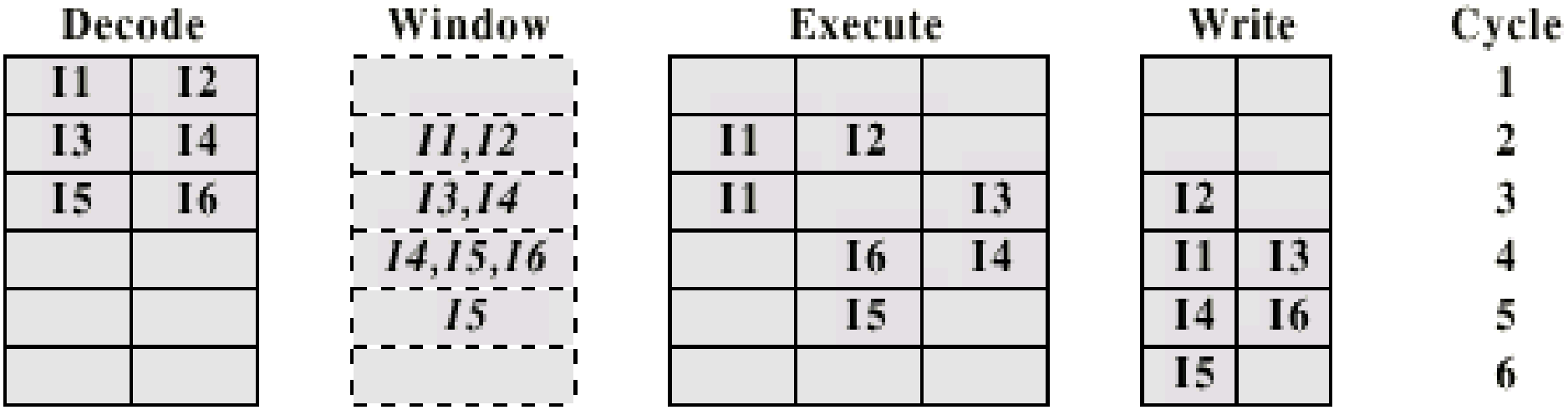
Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
	I4	I1		I3	I2		3
I5	I6			I4	I1	I3	4
	I6		I5		I4		5
			I6		I5		6
					I6		7

I1 necessita dois ciclos na fase de execução
I3 e I4 concorrem pela mesma unidade funcional
I5 depende do valor produzido por I4
I5 e I6 concorrem pela mesma unidade funcional

Iniciação desordenada e entrega desordenada

- Separa a decodificação da execução no *pipeline*
  - busca e decodifica instruções até encher o *pipeline*
- Executa uma instrução assim que uma unidade funcional torna-se disponível
- Visto que instruções estão decodificadas
  - processador pode realizar buscas antecipadas (*look ahead*)
- Área de armazenamento temporário
  - janela de instruções
    - recebe o resultado da decodificação das instruções
  - busca/decodificação continua enquanto a janela tiver espaço

Diagrama para iniciação desordenada e entrega desordenada



I1 necessita dois ciclos na fase de execução
I3 e I4 concorrem pela mesma unidade funcional
I5 depende do valor produzido por I4
I5 e I6 concorrem pela mesma unidade funcional

Antidependência

- Também conhecida como dependência leitura-escrita

```
R3:=R3 + R5;    #I1
R4:=R3 + 1;     #I2
R3:=R5 + 1;     #I3
R7:=R3 + R4;    #I4
```

  - I3 não pode finalizar antes que I2 inicie
    - I2 precisa de um valor em R3 e I3 irá mudar o valor de R3

Renomeação de registradores

- Por que ocorrem dependência de saída e antidependências?
  - conteúdo dos registradores pode não refletir a ordem correta do programa
  - política de entrega desordenada
    - não ocorrem devido ao código gerado

- Também conhecidas como **dependências falsas**

- Podem gerar paradas do *pipeline*

- Uma possível solução (**renomeação de registradores**)
  - registradores lógicos alocados durante a codificação
  - registradores físicos alocados durante a execução

Exemplo renomeação de registradores

R3:=R3 + R5;	#I1	R3b:=R3a + R5a	#I1
R4:=R3 + 1;	#I2	R4b:=R3b + 1	#I2
R3:=R5 + 1;	#I3	R3c:=R5a + 1	#I3
R7:=R3 + R4;	#I4	R7b:=R3c + R4b	#I4

- Sem *subscrito a instrução* refere-se a registradores lógicos
- Com *subscrito* o registrador físico é alocado
- Observe: R3a R3b R3c

Paralelismo de máquina

- Foram comentadas três técnicas de hardware
  - duplicação de recursos
  - execução desordenada
  - renomeação de registradores

- Smith, Johnson e Horowitz mostram a relação entre elas:
  - replicação de unidades funcionais não vale a pena sem renomeação de registradores
  - precisa de uma janela de instruções grande o suficiente
    - normalmente mais que 8

Previsão de desvio

- Ocorre com freqüência e necessita tratamento!

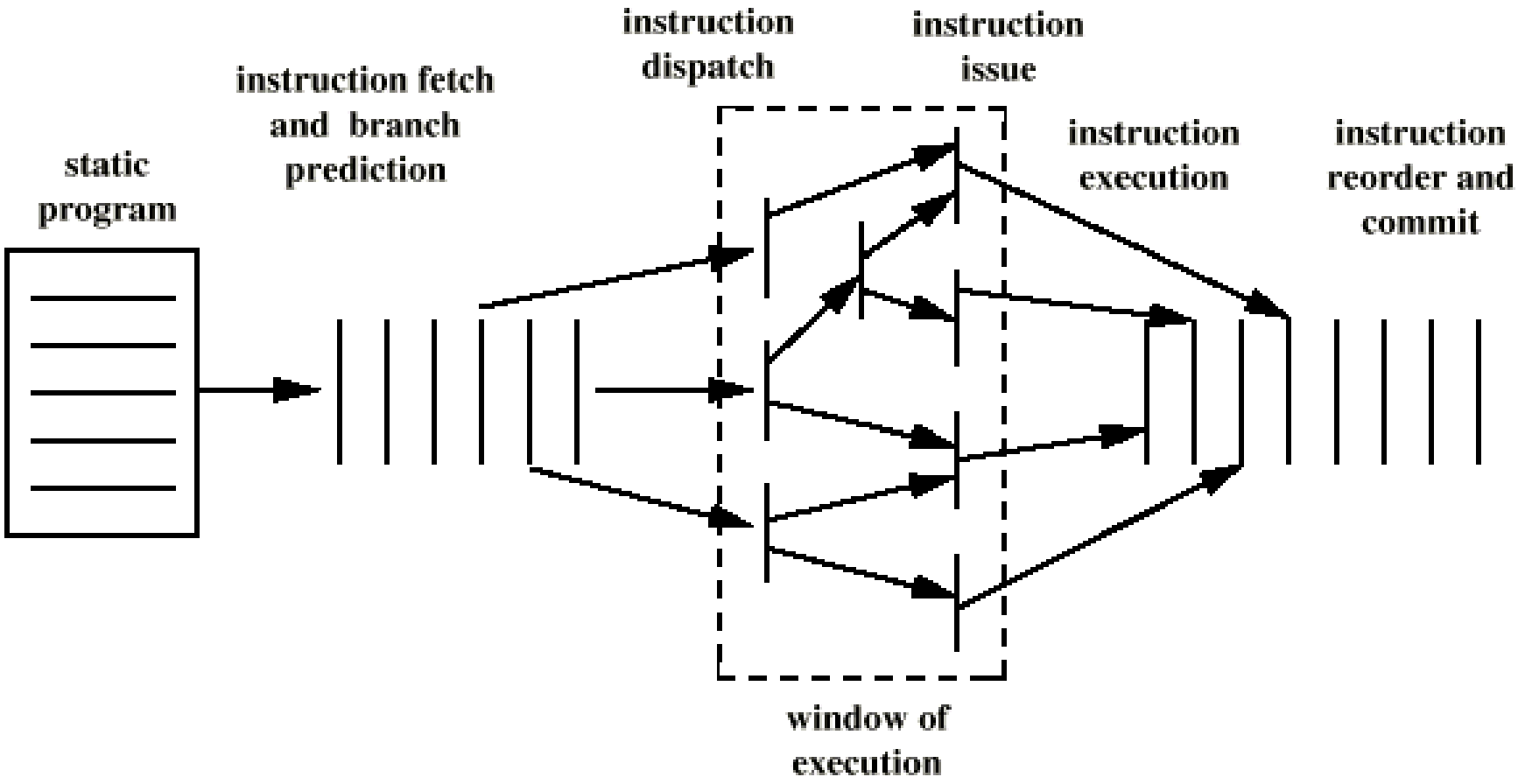
- Ex.: 80486 busca as duas instruções
  - próxima instrução (na seqüência) e a instrução alvo do desvio
  - insere dois ciclos de atraso se o desvio for realizado
    - há dois estágios entre a busca antecipada e o estágio de execução

- Outras técnicas podem ser exploradas:
  - ex.: desvio atrasado
    - inserir instruções após o desvio para que o resultado desta possa ser calculado

Desvio atrasado - RISC

- Permite calcular o resultado do desvio antes da pré-busca de instruções que não podem ser utilizadas
  - sempre executa a instrução seguinte ao desvio condicional
- Mantém o pipeline cheio enquanto está buscando a nova instrução
- Solução não é boa para processador superescalar
  - múltiplas instruções necessitam executar no *slot* de atraso
    - há várias seqüências de *pipeline*
  - gera problemas com as dependências de instrução
- UCPs superescalares optaram por técnicas antigas
  - predecessoras das máquinas RISC
  - ex.: previsão estática ou dinâmica baseada no histórico

Execução superescalar



Implementação superescalar

- Busca múltiplas instruções simultaneamente
- Lógica para determinar dependências verdadeiras
  - envolvendo valores dos registradores
- Mecanismos para comunicar estes valores
  - ex.: adiantamento de valores entre instruções
- Mecanismos para iniciar múltiplas instruções em paralelo
- Recursos para execução paralela de múltiplas instruções
  - múltiplas unidades funcionais e hierarquia de memória eficiente
- Mecanismos para confirmar resultados na ordem correta

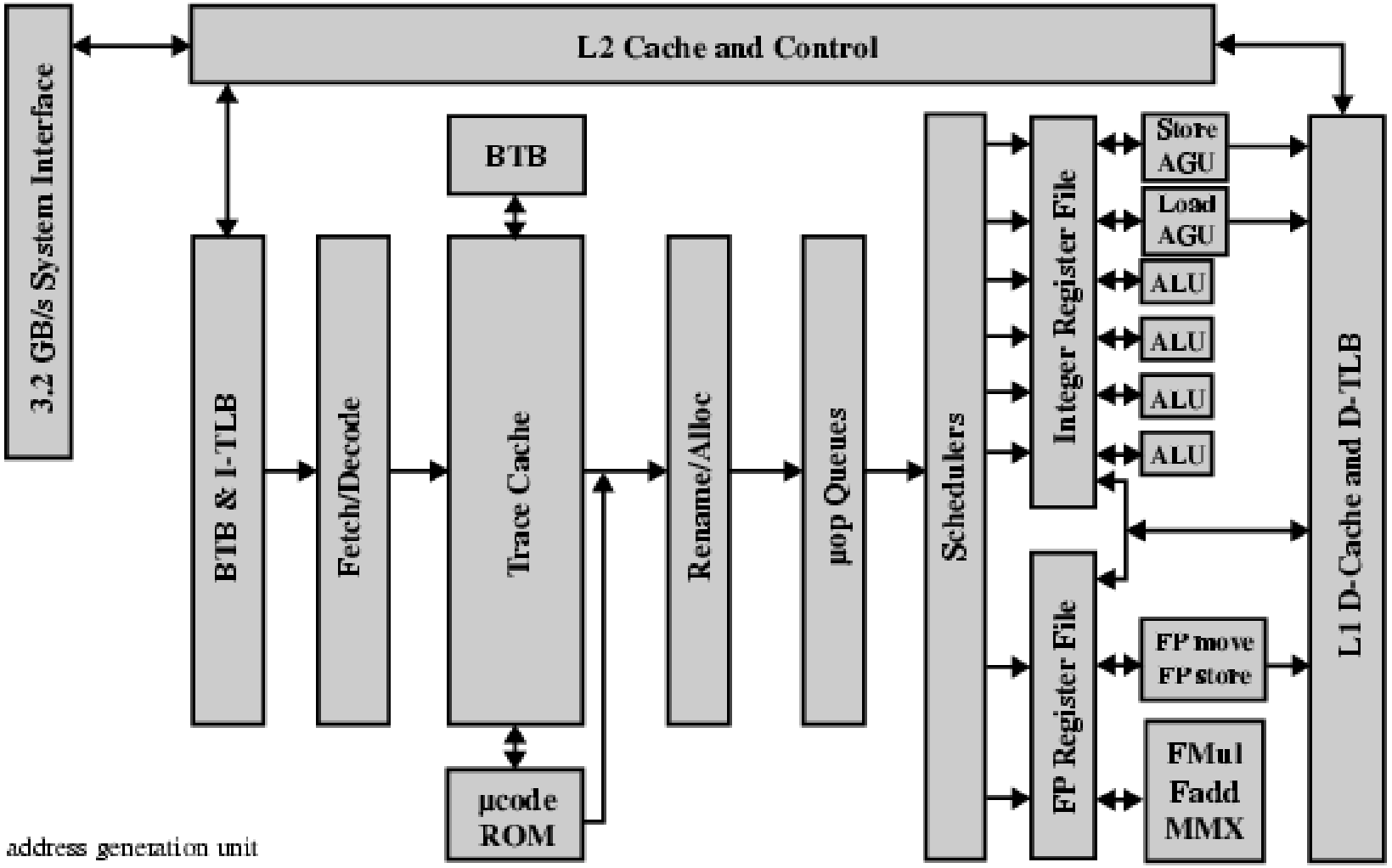
Pentium 4

- 80486 - CISC
- Pentium – alguns componentes superescalares
  - duas unidades de inteiro separadas
- Pentium Pro – projeto superescalar mais maduro
- Modelos subseqüentes refinaram e aumentaram o projeto superescalar

Operação do Pentium 4

- ***Busca instruções da memória na ordem que elas aparecem no programa***
- ***Traduz instrução em uma ou mais instruções RISC de tamanho fixo (micro-operações)***
- ***Executa micro-operações no pipeline superescalar***
  - ***micro-operações podem executar desordenadamente***
- ***Confirma resultados das micro-operações no conjunto de registradores, exatamente na ordem que as instruções ocorrem no código original***

Diagrama de bloco do Pentium 4



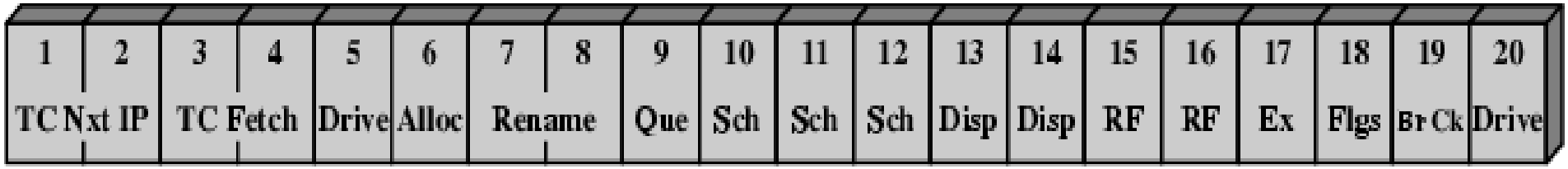
AGU = address generation unit
BTB = branch target buffer
D-TLB = data translation lookaside buffer
I-TLB = instruction translation lookaside buffer

Operação do Pentium 4

- Visão exterior CISC com interior RISC

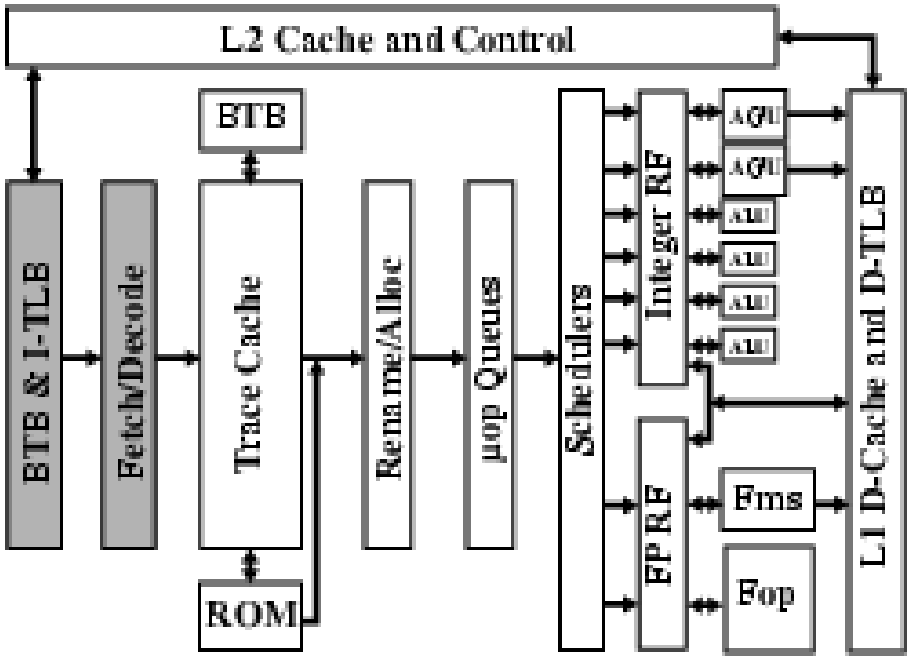
- Máquinas RISC com no mínimo 20 estágios
  - algumas micro-operações necessitam de estágios de múltiplas execuções.
  - *projeto é mais complexo que o usado no pipeline* de 5 estágios no x86 até o Pentium

Pipeline do Pentium 4

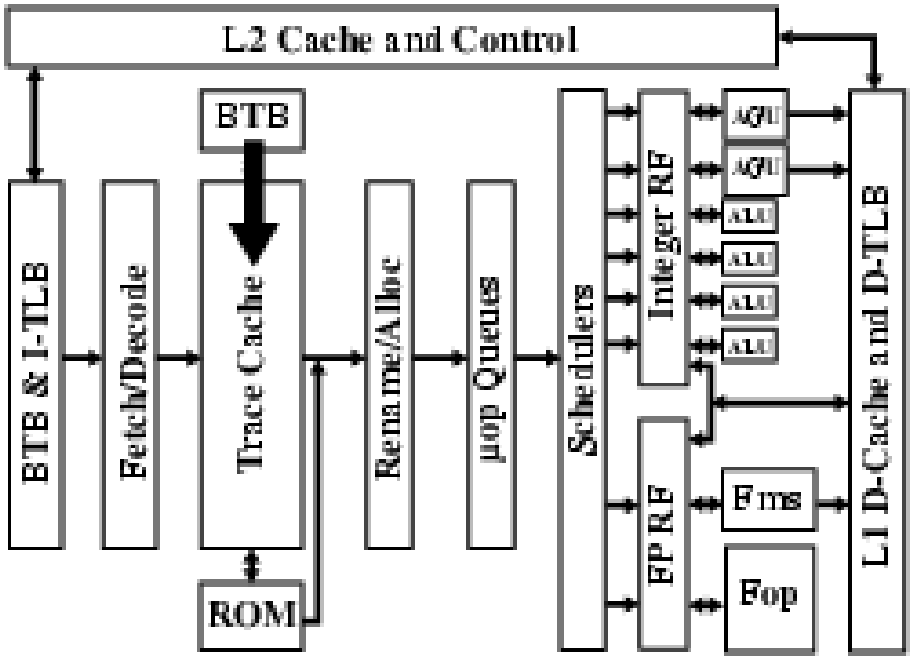


TC Next IP = trace cache next instruction pointer
TC Fetch = trace cache fetch
Alloc = allocate
Rename = register renaming
Que = micro-op queuing
Sch = micro-op scheduling
Disp = Dispatch
RF = register file
Ex = execute
Flgs = flags
Br Ck = branch check

Operação do *pipeline* no *Pentium 4* (1)



(a) Generation of micro-ops



(b) Trace cache next instruction pointer

Transfere 64 bytes do Cache

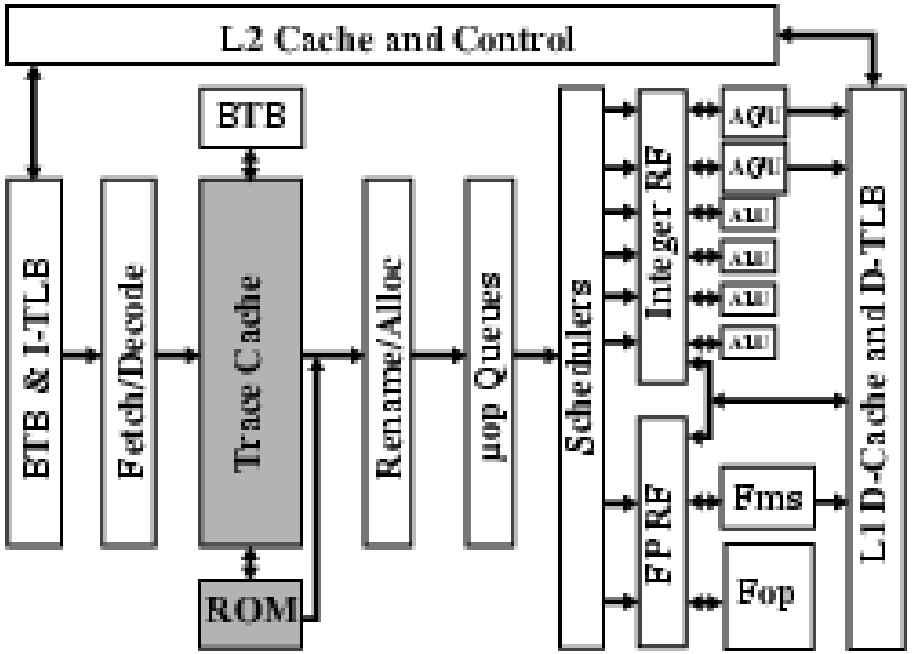
Determina limites da instrução

Gera de 1 a 4 micro operações

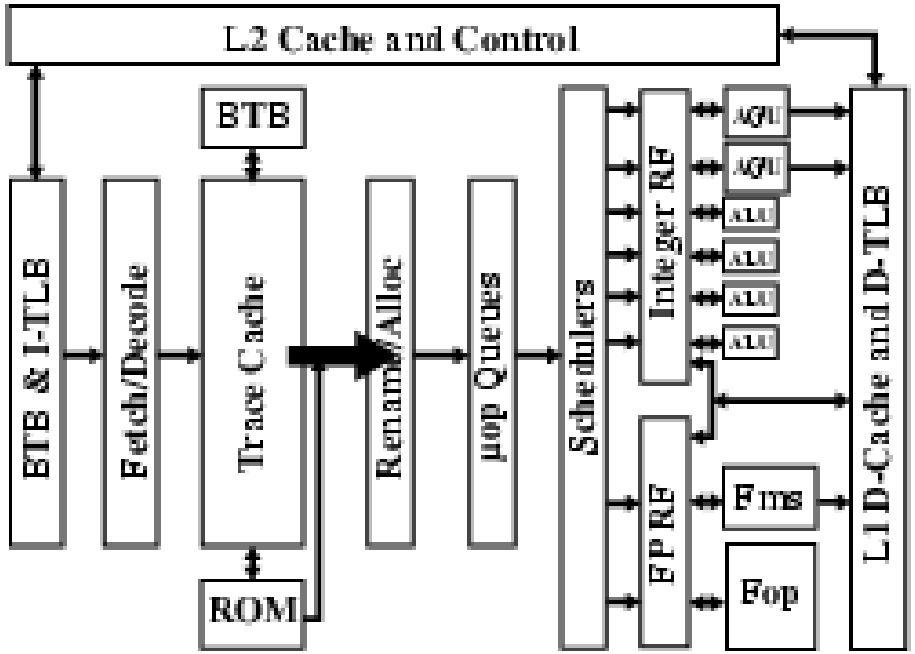
BTB – Branch Target Table

512 linhas

Operação do *pipeline* no *Pentium 4* (2)



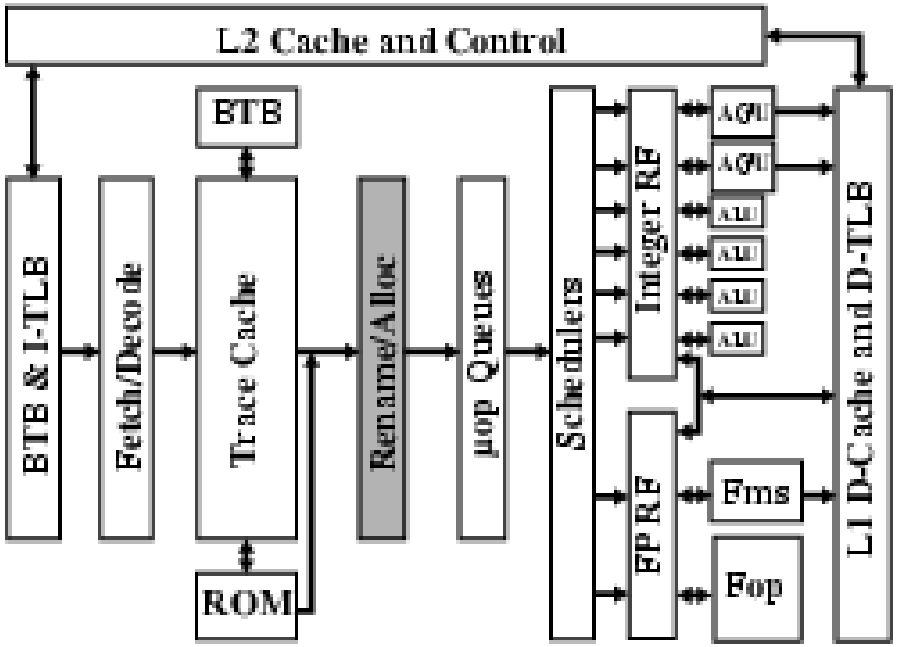
(c) Trace cache fetch



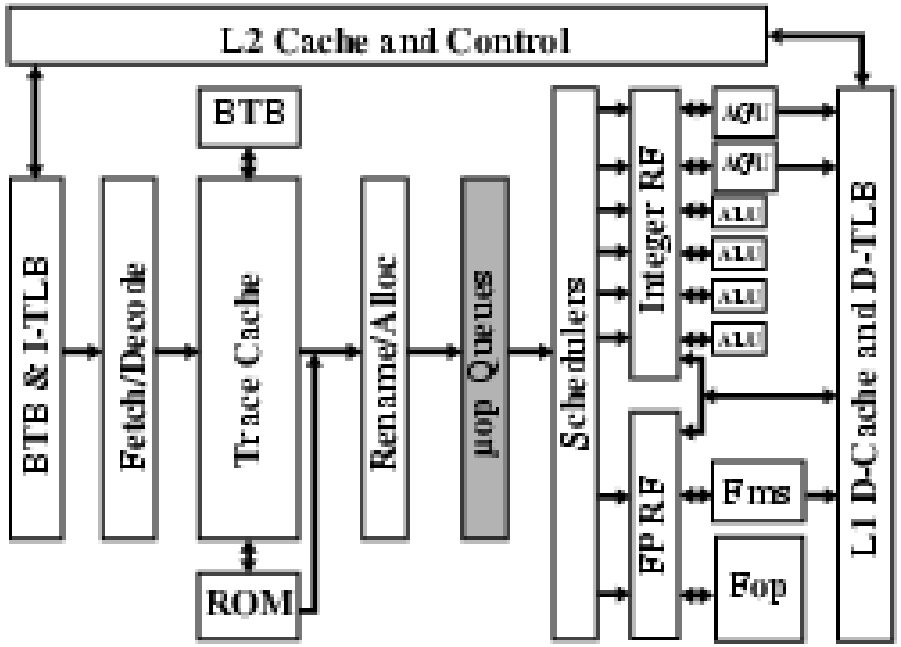
(d) Drive

ROM – Utilizada se no. De micro instr.
For maior que 4

Operação do *pipeline* no *Pentium 4* (3)



(e) Allocate; Register renaming



(f) Micro-op queuing

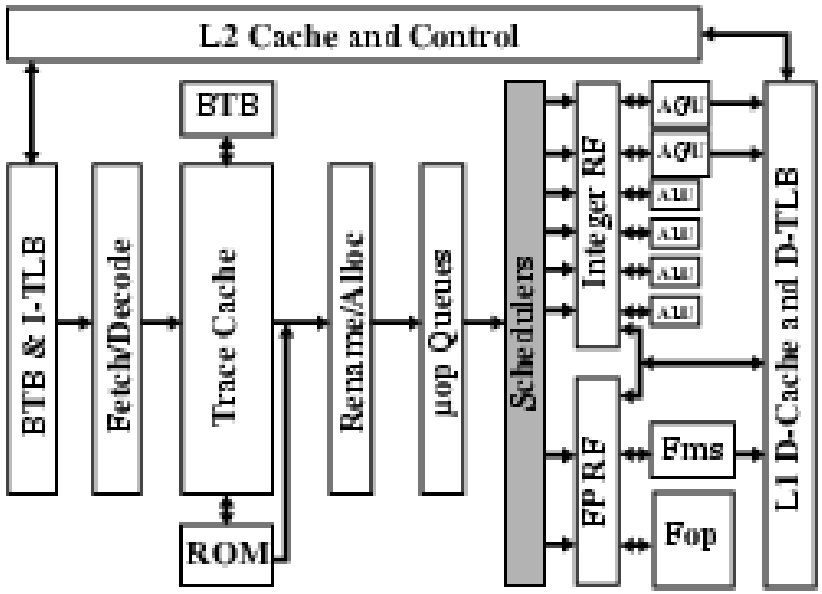
Renaming 16 registradores para 128 registradores físicos

Reordena micro instruções

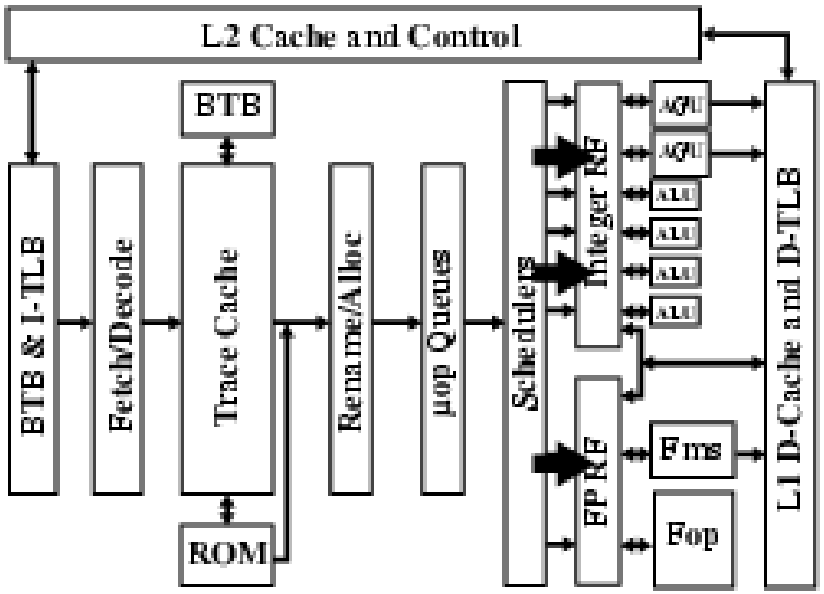
Gera duas filas:

Operações com ou sem acesso a memória

Operação do *pipeline* no *Pentium 4* (4)



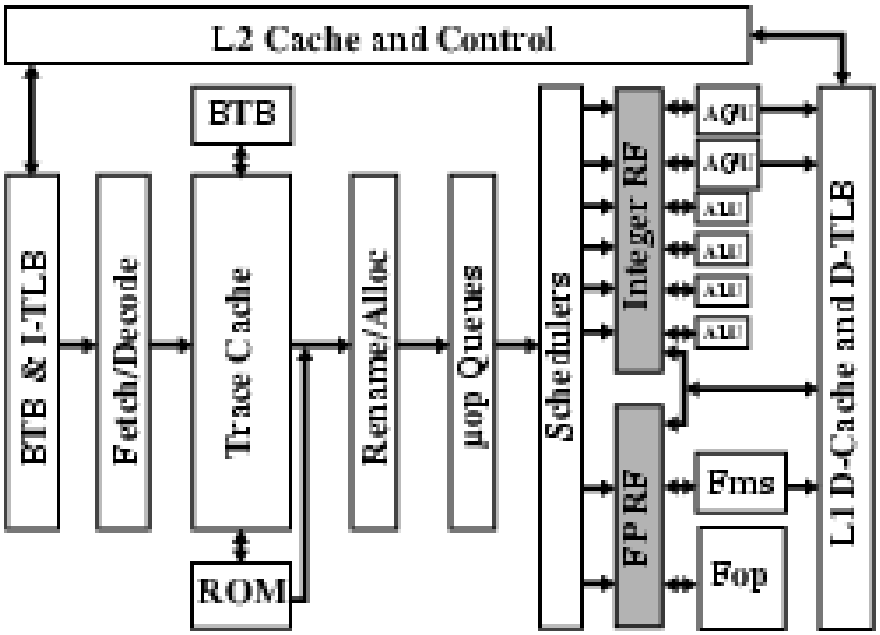
(g) Micro-op scheduling



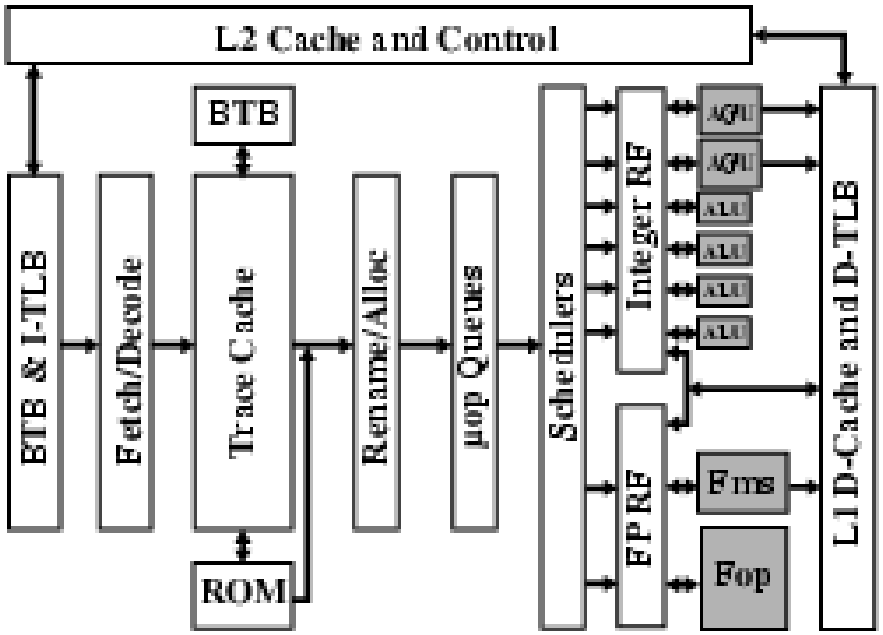
(h) Dispatch

Ativa execução de até 6 micro operações

Operação do *pipeline* no *Pentium 4* (5)

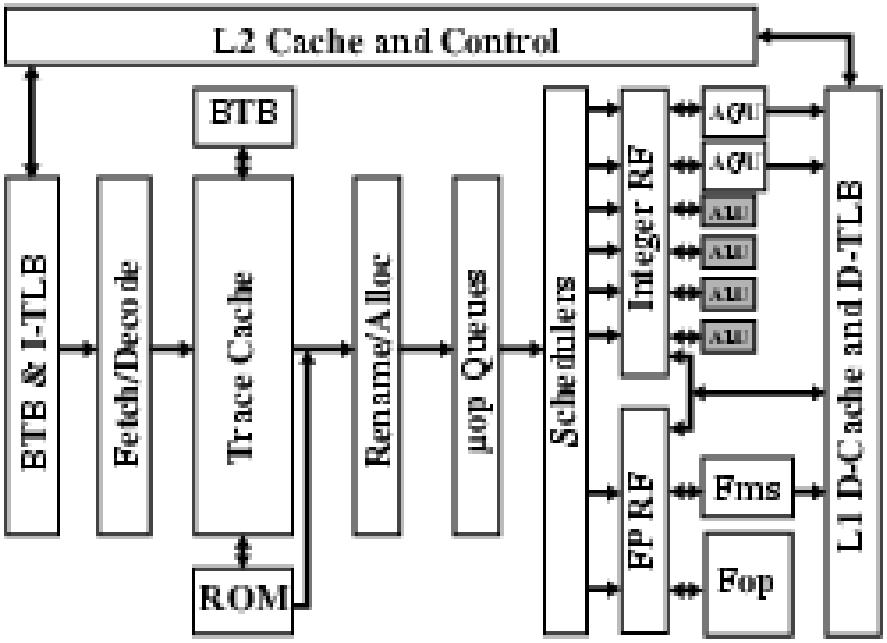


(i) Register file

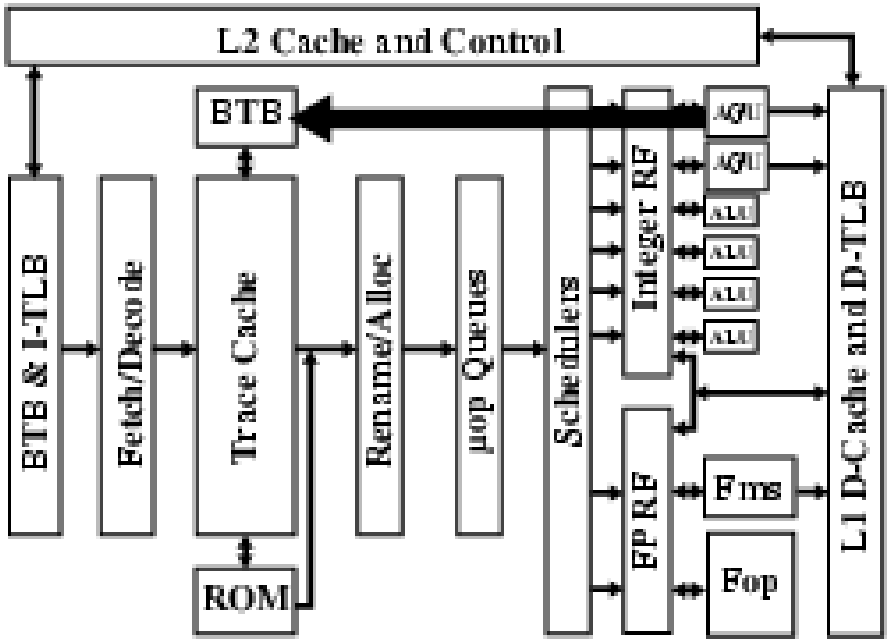


(j) Execute; flags

Operação do *pipeline* no *Pentium 4* (6)



(k) Branch check

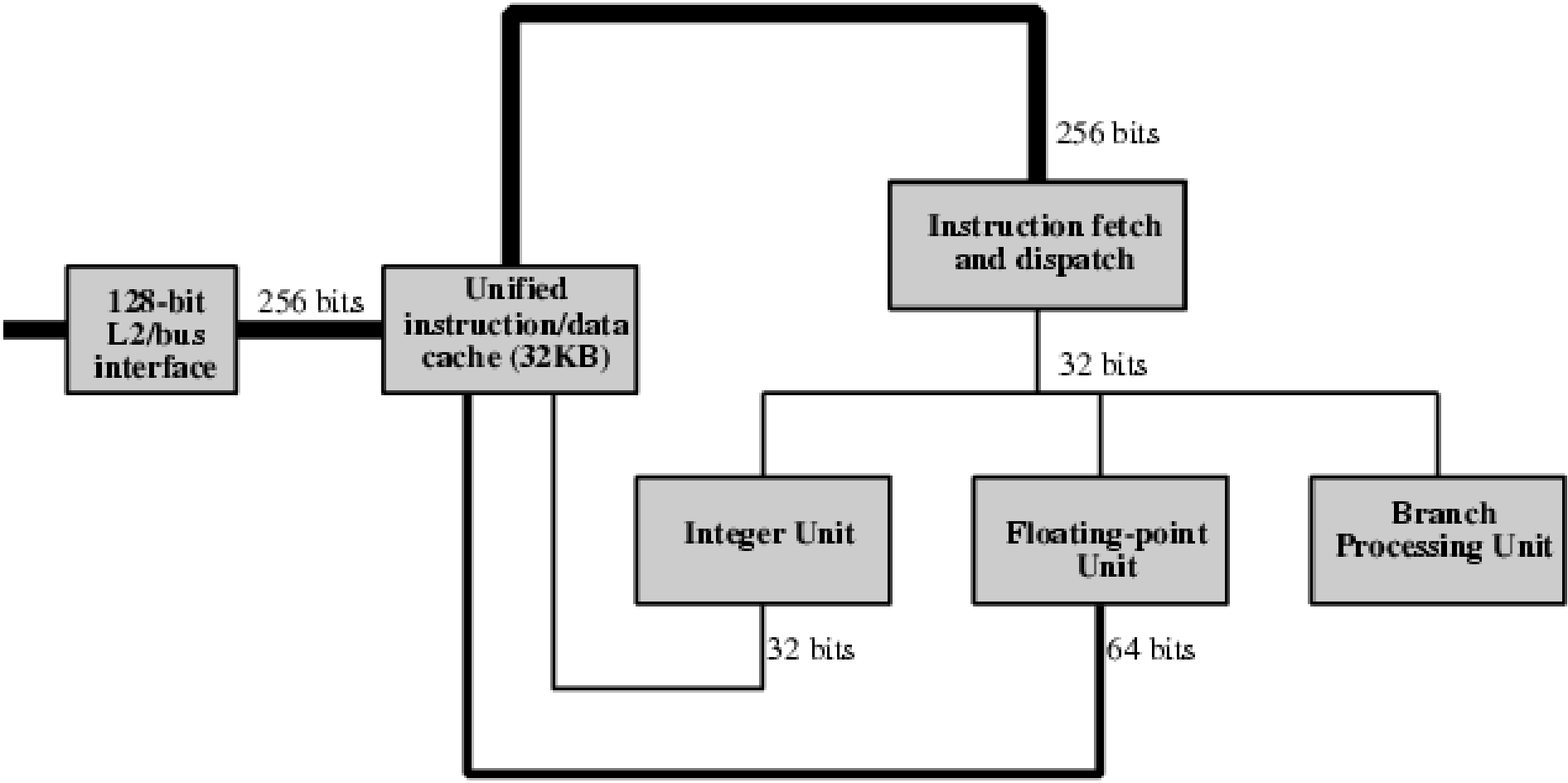


(l) Branch check result

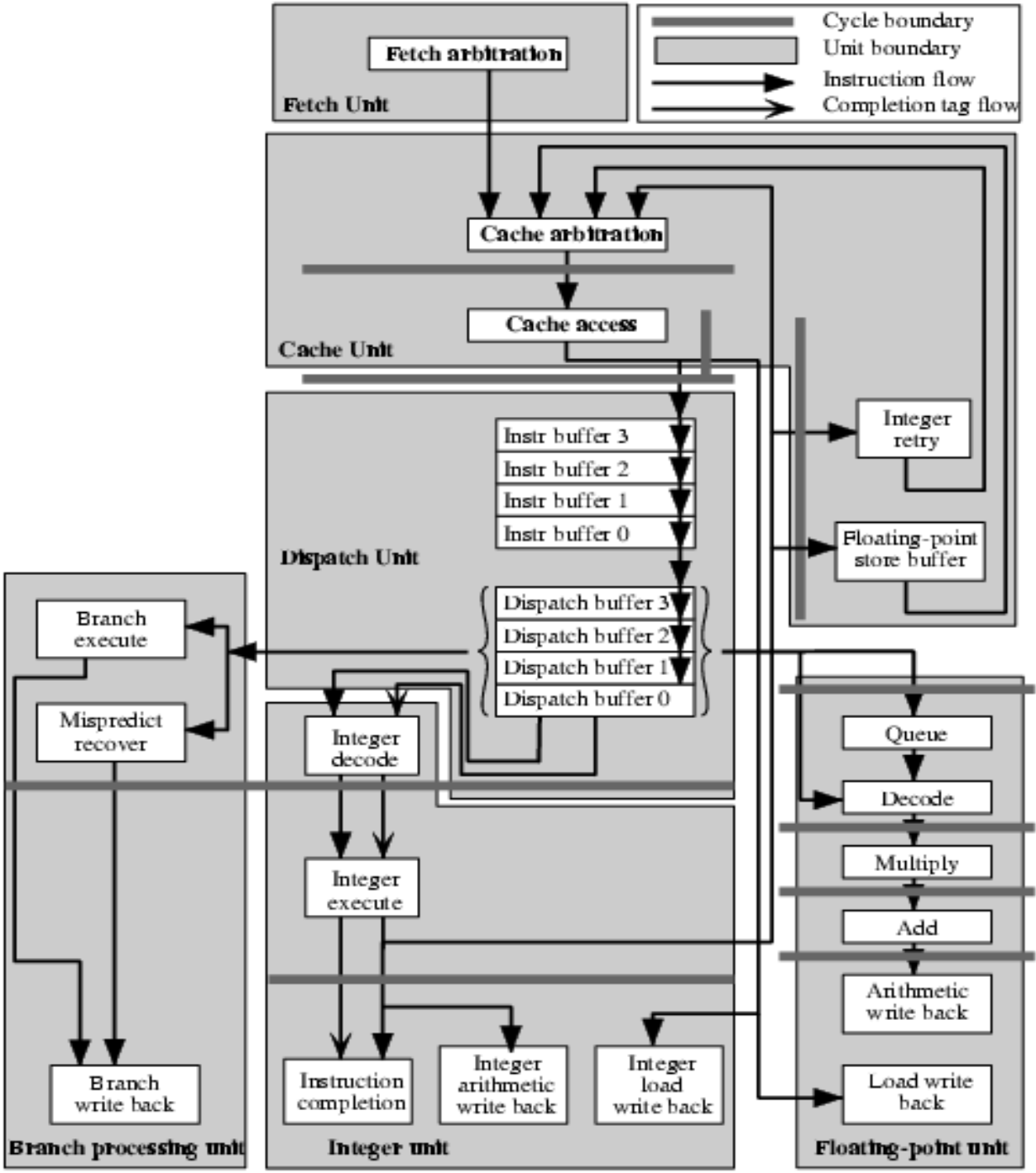
PowerPC

- Descendente direto do *IBM 801, RT PC* e *RS/6000*
  - —todas máquinas RISC
- *RS/6000* foi o 1o. processador superescalar
- Projeto superescalar do *PowerPC 601* é similar ao projeto do *RS/6000*
- Projetos posteriores estenderam conceito de superescalar

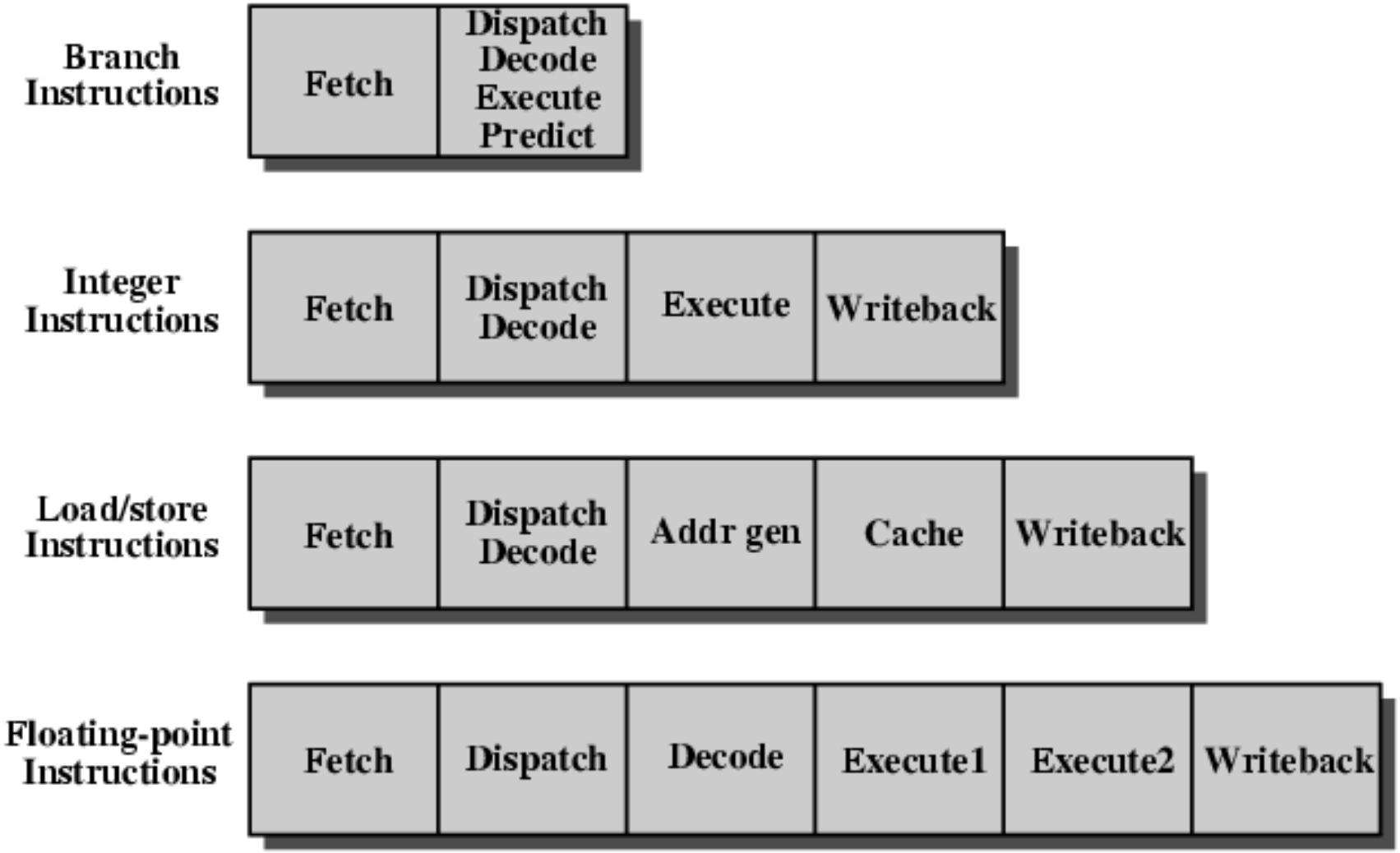
Visão Geral do *PowerPC 601*



Estrutura *PowerPC 601*



Pipeline do *PowerPC 601*



Leitura Recomendada

- Capítulo 14 do Stallings
- *Web sites* dos fabricantes