Capítulo 4

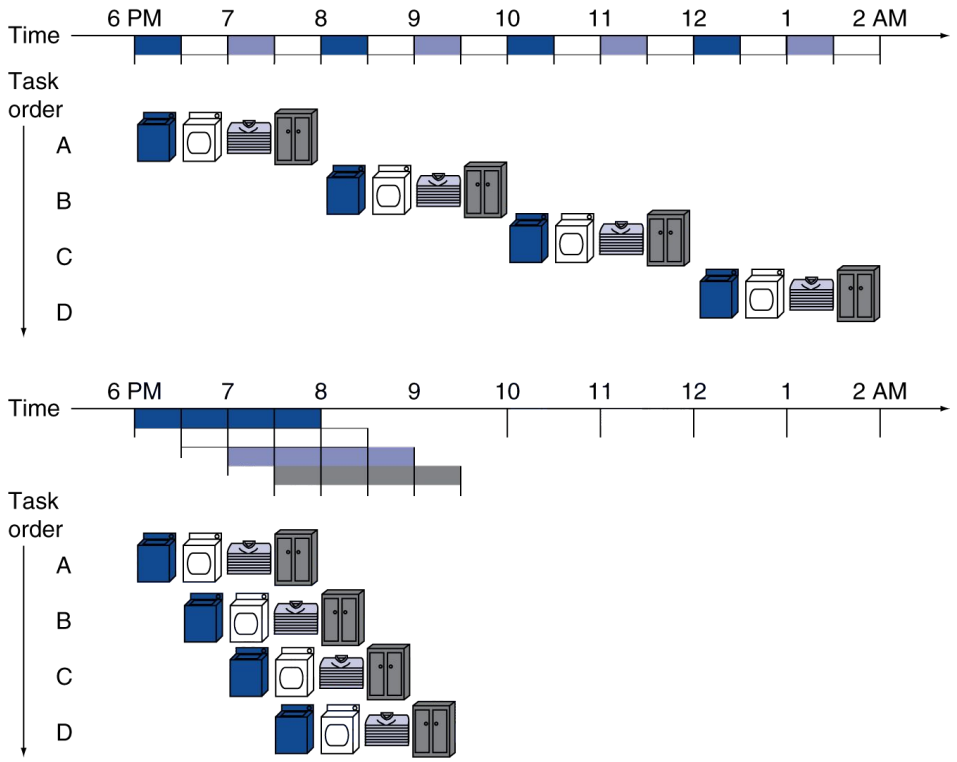
O Processador

Problemas de desempenho

- O maior atraso determina o período do clock
  - Caminho crítico: instrução load
  - Memória de instrução →leitura registradores → ULA → memória de dados → escrita registradores
- Não é viável variar o período para instruções diferentes
- Viola o princípio de design
  - Tornando o caso comum rápido
- Vamos melhorar o desempenho através do pipelining

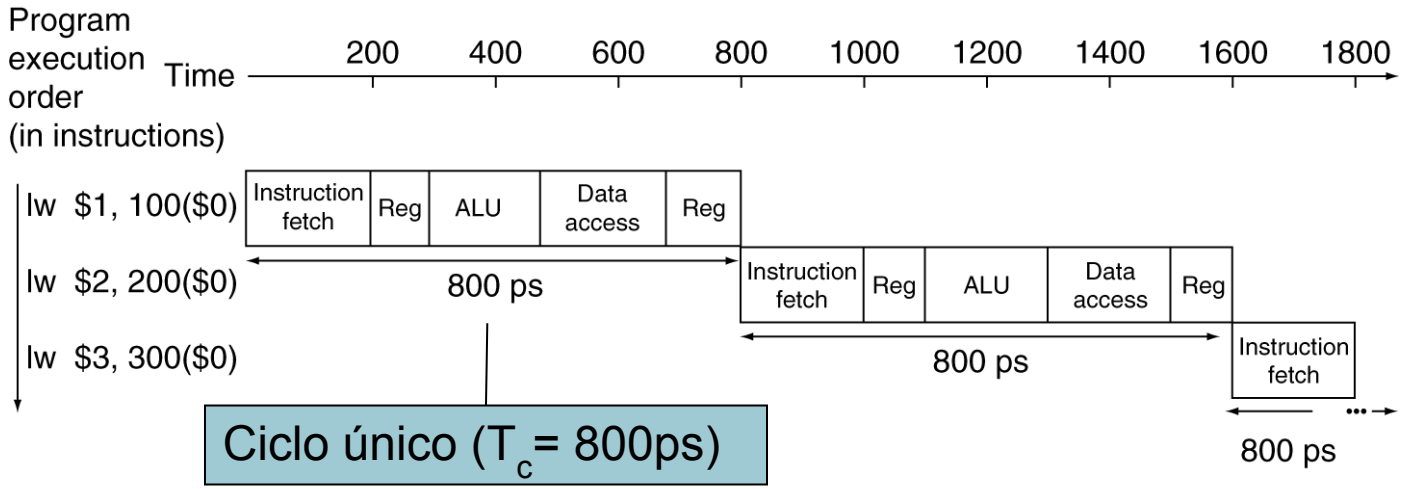
Analogia de Pipelining

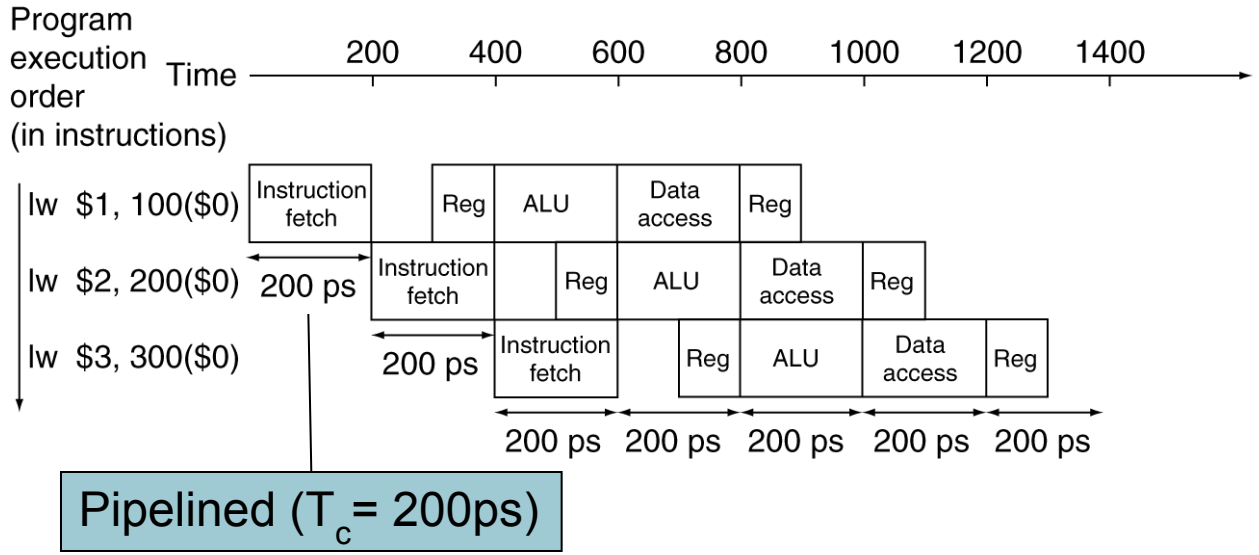
- Lavanderia em pipeline: execução sobreposta
  - Paralelismo melhora desempenho



- Quatro tarefas:
  - Speedup = 8 / 3,5 = 2,3

Desempenho Pipeline





Pipeline MIPS

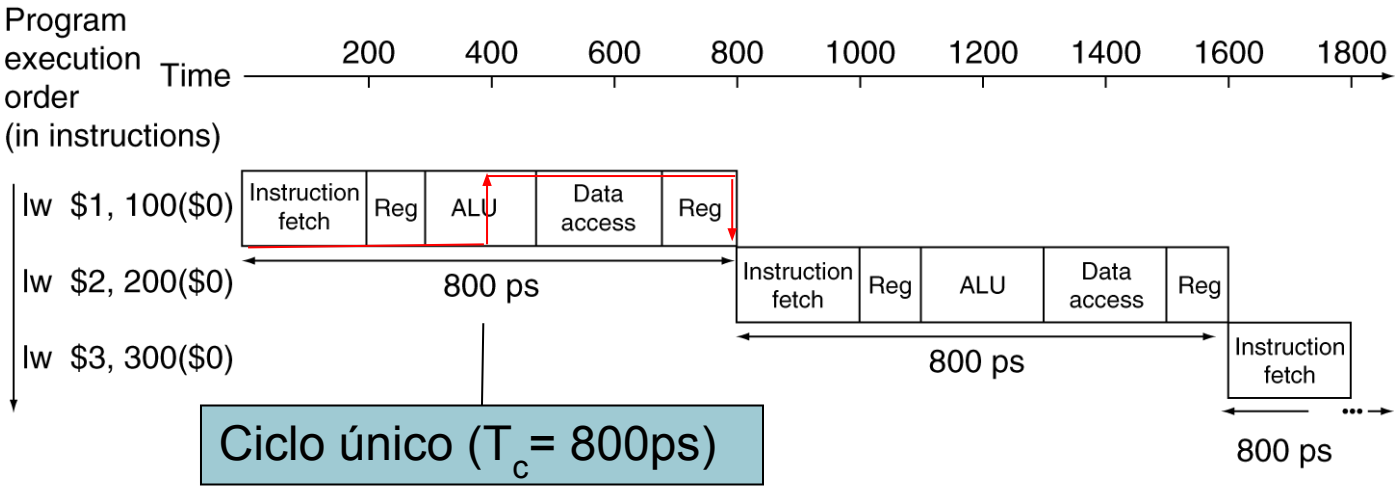
1. **IF**: **I**nstruction **F**etch
   - Busca instrução da memória
2. **ID**: **I**nstruction **D**ecode
   - Lê os registradores enquanto a instrução é decodificada
3. **EX**: **E**xecute
   - Executa a operação ou calcular um endereço
4. **MEM**: **Mem**ory
   - Acessa um operando na memória de dados
5. **WB**: **W**rite result **B**ack
   - Escreve o resultado em um registrador

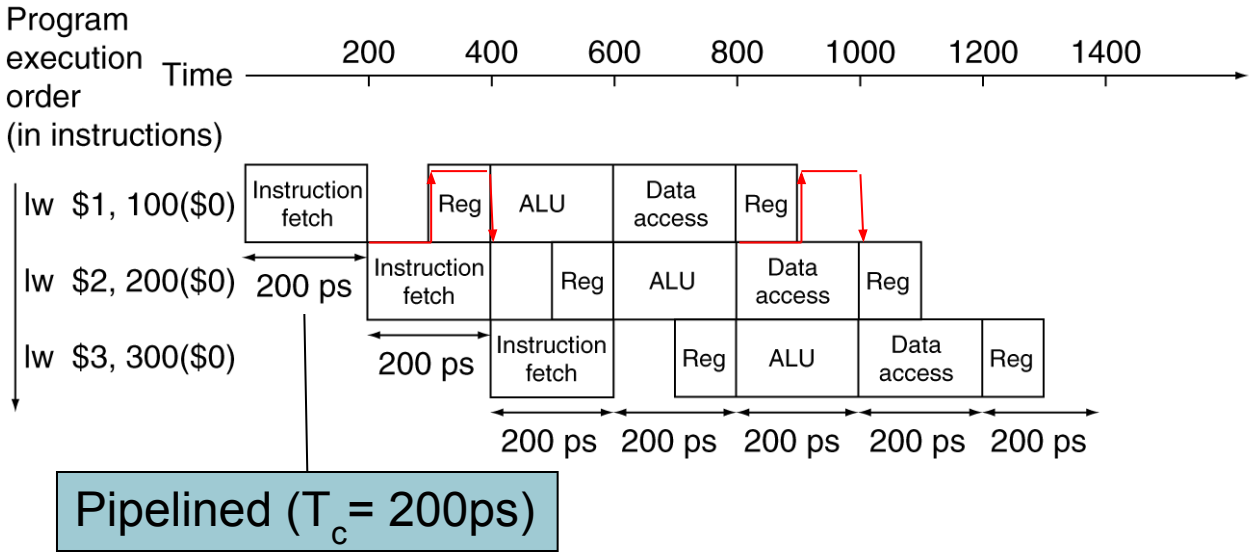
Desempenho do Pipeline

- Suponha que o tempo para estágios seja
  - 100ps para leitura ou gravação de registrador
  - 200ps para outros estágios
- Compare o caminho de dados em pipeline com o caminho de dados de ciclo único

Instrução	Busca Instrução	Leitura Reg.	Operação ULA	Acesso Dados	Escrita Reg.	Tempo Total
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Desempenho Pipeline





Pipeline Speedup

- Se todos os estágios estiverem equilibrados
  - Todos levam o mesmo tempo
  - Tempo entre as instruções$_{\text{com pipeline}}$ =

$$\frac{\textbf{Tempo entre as instruções}_{\textbf{sem pipeline}}}{\textbf{Número de estágios}}$$

- Se não equilibrado, speedup é menor
- Speedup devido ao aumento da vazão de instruções (throughput)
  - A latência (tempo para cada instrução) não diminui

Pipelining e Projeto de ISA

- ISA do MIPS projetado para pipelining
  - Todas as instruções possuem 32 bits
    - Mais fácil de buscar e decodificar em um ciclo
    - x86: instruções de 1 a 17 bytes
  - Poucos e regulares formatos de instrução
    - Decodifica e lê registradores em um passo
  - Endereçamento Load/store
    - Calcula o endereço no estágio 3, acessa memória no estágio 4
  - Alinhamento de operandos de memória
    - O acesso à memória leva apenas um ciclo

Hazards

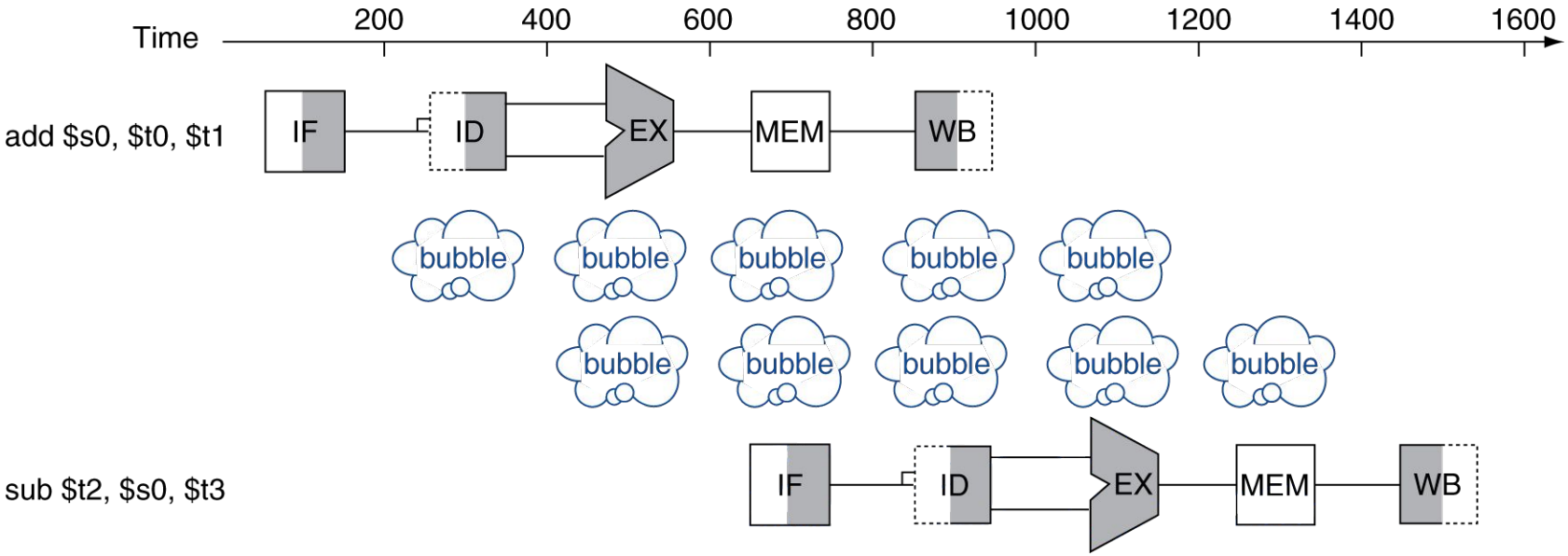
- Situações que impedem o início da próxima instrução no próximo ciclo
- Conflitos estruturais (Structure Hazards)
  - Um recurso necessário está ocupado
- Conflitos de dados (Data Hazards)
  - Precisa aguardar instruções anteriores para concluir a leitura/escrita de dados
- Conflitos de controle (Control hazards)
  - Decisão de controle depende de instrução prévia

Hazards Estrutural

- Conflito pelo uso de um recurso
- No pipeline do MIPS com uma única memória (instruções/dados)
  - Load/store requer acesso a dados
  - A busca de instruções teria que parar para esse ciclo
    - Causaria uma "bolha" no pipeline
- Portanto, os caminhos de dados em pipeline requerem memórias separadas de instruções/dados
  - Ou separar caches de instruções/dados

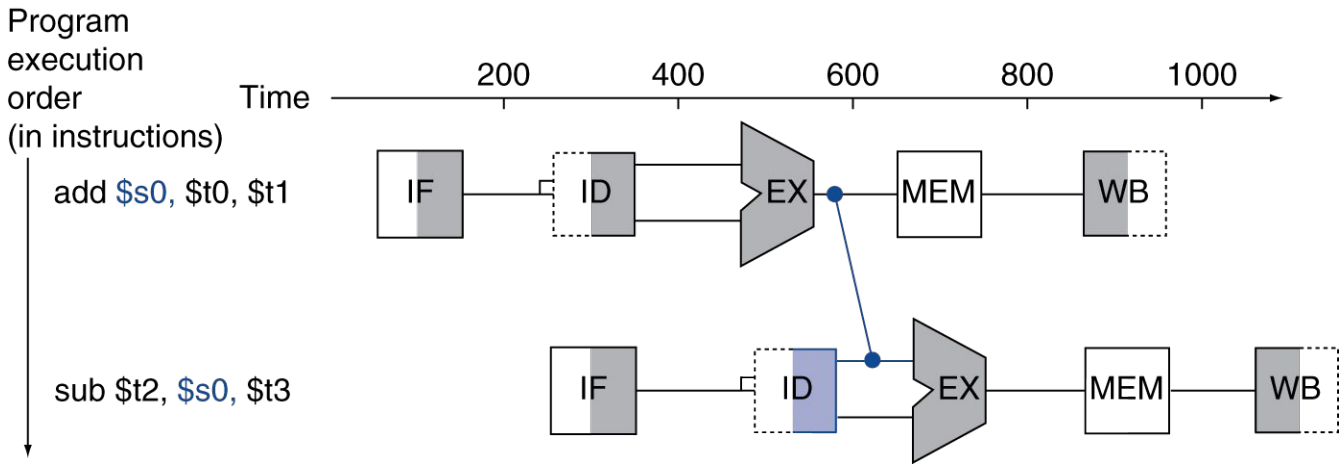
Hazards de Dados

- Uma instrução depende da conclusão do acesso a dados por uma instrução anterior
  - `add $s0, $t0, $t1`
    `sub $t2, $s0, $t3`



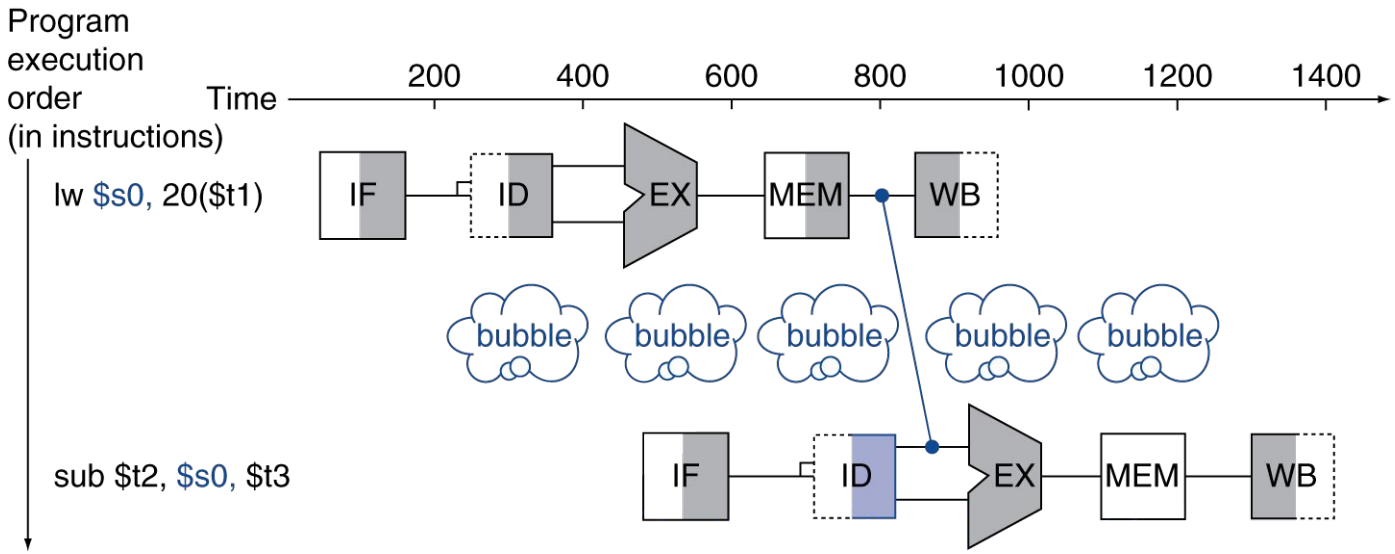
Encaminhamento (Forwarding)

- Utiliza o resultado quando é calculado
  - Não espere pelo armazenamento no registrador
  - Requer conexões extras no caminho de dados



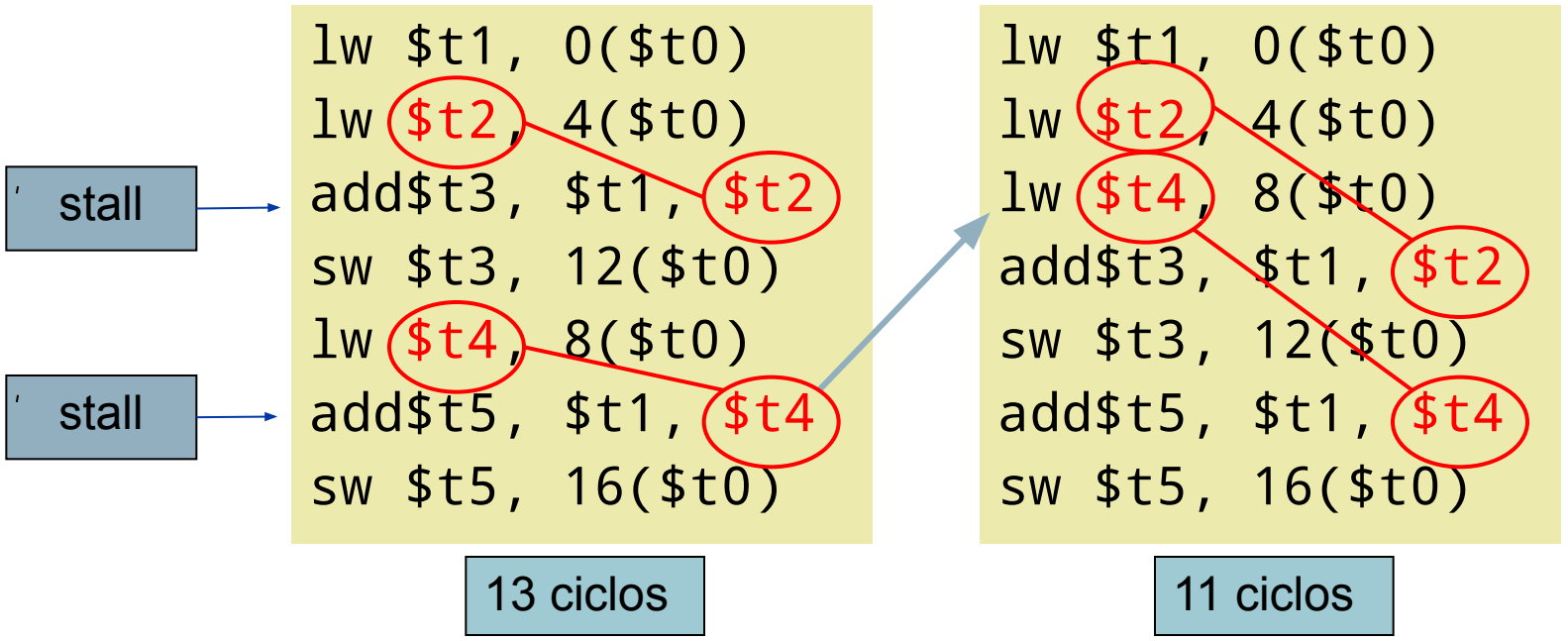
Hazard de Dados Uso Load

- Nem sempre é possível evitar paradas com fowarding
  - Se o valor não estiver disponível quando necessário
  - Não é possível voltar no tempo!



Reordenação de Código para Evitar Stalls

- Reordene o código para evitar o uso do resultado de load na próxima instrução
- Código C para A = B + E; C = B + F;



Reordenação de Código para Evitar Stalls

13 ciclos

	1	2	3	4	5	6	7	8	9	10	11	12	13
lw $t1,0($t0)	IF	ID	EX	MEM	WB								
lw **$t2**,4($t0)		IF	ID	EX	MEM	WB							
nop			IF	ID	EX	MEM	WB						
add $t3,$t1,**$t2**				IF	ID	EX	MEM	WB					
sw $t3,12($t0)					IF	ID	EX	MEM	WB				
lw **$t4**,8($t0)						IF	ID	EX	MEM	WB			
nop							IF	ID	EX	MEM	WB		
add $t5,$t1,**$t4**								IF	ID	EX	MEM	WB	
sw $t5,16($t0)									IF	ID	EX	MEM	WB

Reordenação de Código para Evitar Stalls

11 ciclos

	1	2	3	4	5	6	7	8	9	10	11	12	13
lw $t1,0($t0)	IF	ID	EX	MEM	WB								
lw **$t2**,4($t0)		IF	ID	EX	MEM	WB							
lw **$t4**,8($t0)			IF	ID	EX	MEM	WB						
add $t3,$t1,**$t2**				IF	ID	EX	MEM	WB					
sw $t3,12($t0)					IF	ID	EX	MEM	WB				
add $t5,$t1,**$t4**						IF	ID	EX	MEM	WB			
sw $t5,16($t0)							IF	ID	EX	MEM	WB		

Hazards de Controle

- O desvio determina o fluxo de controle
  - A busca da próxima instrução depende do resultado do desvio
  - O pipeline nem sempre pode buscar as instruções corretas
    - Ainda trabalhando no estágio ID do desvio
- No pipeline do MIPS
  - Precisa comparar registradores e calcular o endereço destino no início do pipeline
  - Adicione hardware para fazê-lo no estágio ID (Instruction Decode)

Hazards de Controle

- Desvios no pipeline
  - O que acontece quando o desvio é tomado nesta sequência de instruções, considerando que movemos a execução do desvio para o estágio ID:

```
36 sub $10, $4, $8
40 beq $1,  $3, 7  #desvio relativo ao PC
44 and $12, $2, $5 #40 + 4 + 7 * 4 = 72
48 or  $13, $2, $6
52 add $14, $4, $2
56 slt $15, $6, $7
. . .
72 lw  $4,  50($7)
```

Hazards de Controle

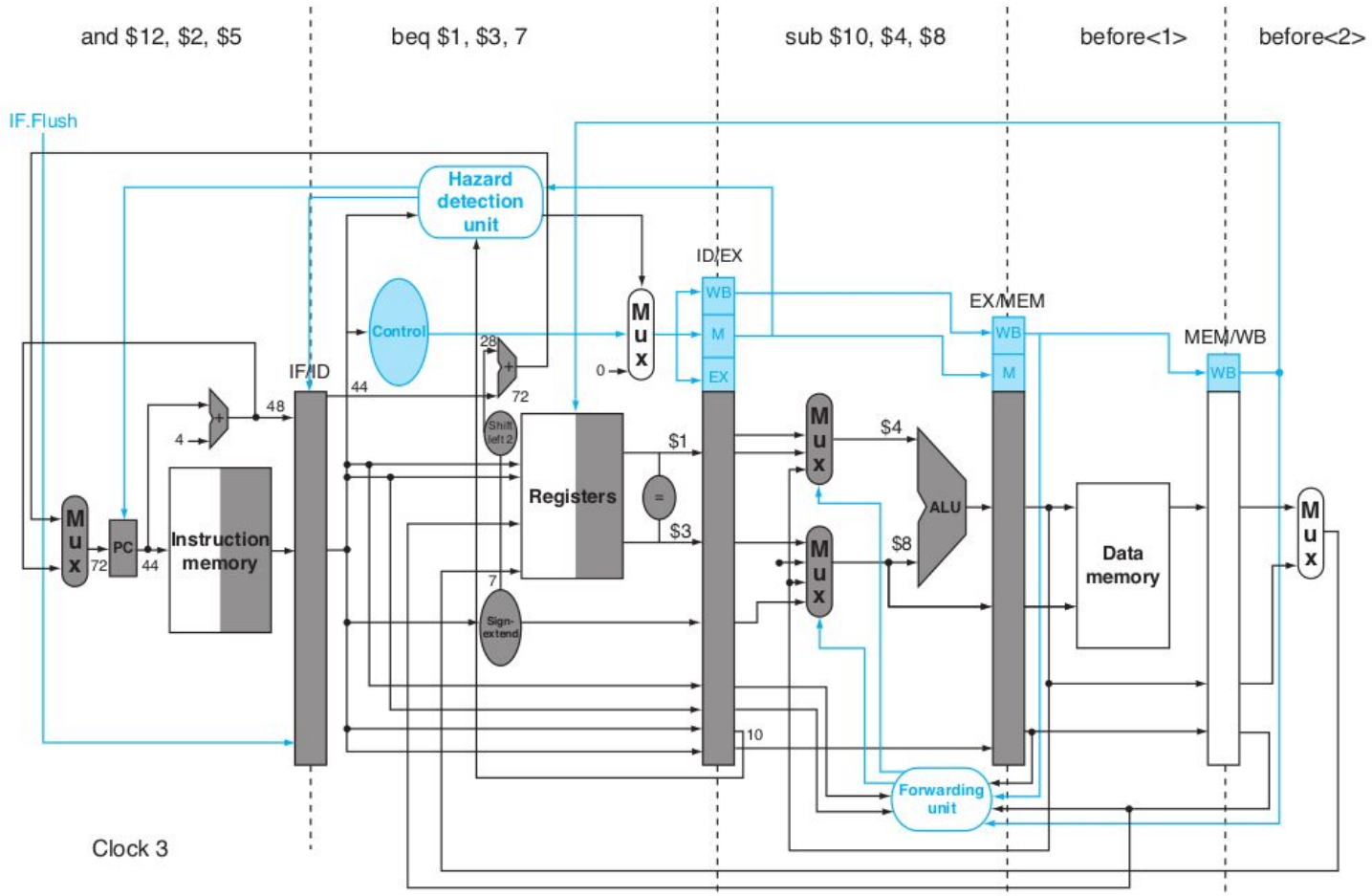


FIGURE 4.62 The ID stage of clock cycle 3 determines that a branch must be taken, so it selects 72 as the next PC address and zeros the instruction fetched for the next clock cycle. Clock cycle 4 shows the instruction at location 72 being fetched and the single bubble or nop instruction in the pipeline as a result of the taken branch. (Since the nop is really sll $0 , $0 , 0 , it's arguable whether or not the ID stage in clock 4 should be highlighted.).

Hazards de Controle

clock	1	2	3	4	5	6	7	8	9	10	11	12	13
sub $10, $4, $8	IF	ID	EX	MEM	WB								
beq $1, $3, 7		IF	ID	EX	MEM	WB							
and $12, $2, $5			IF	ID	EX	MEM	WB						

Hazards de Controle

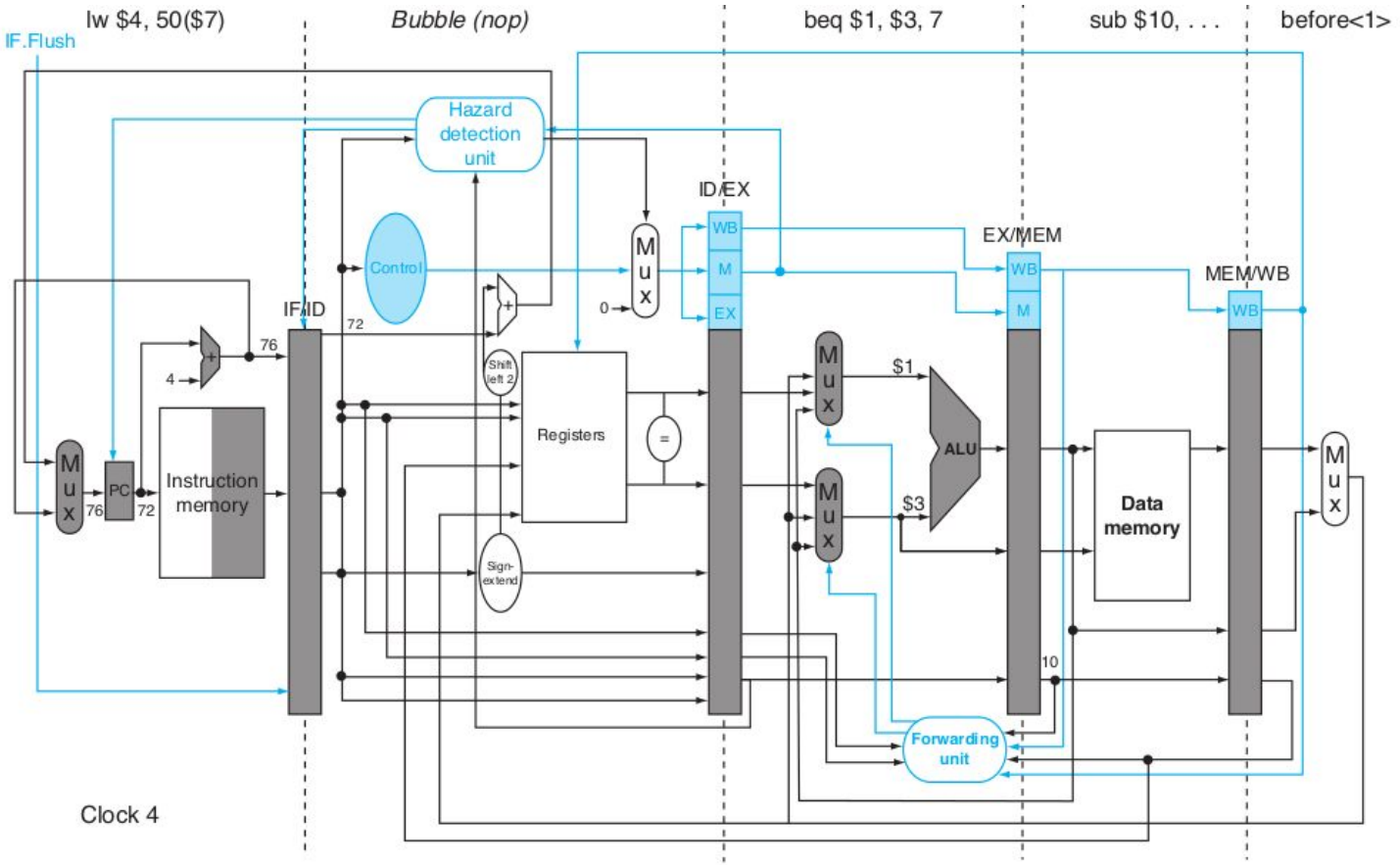


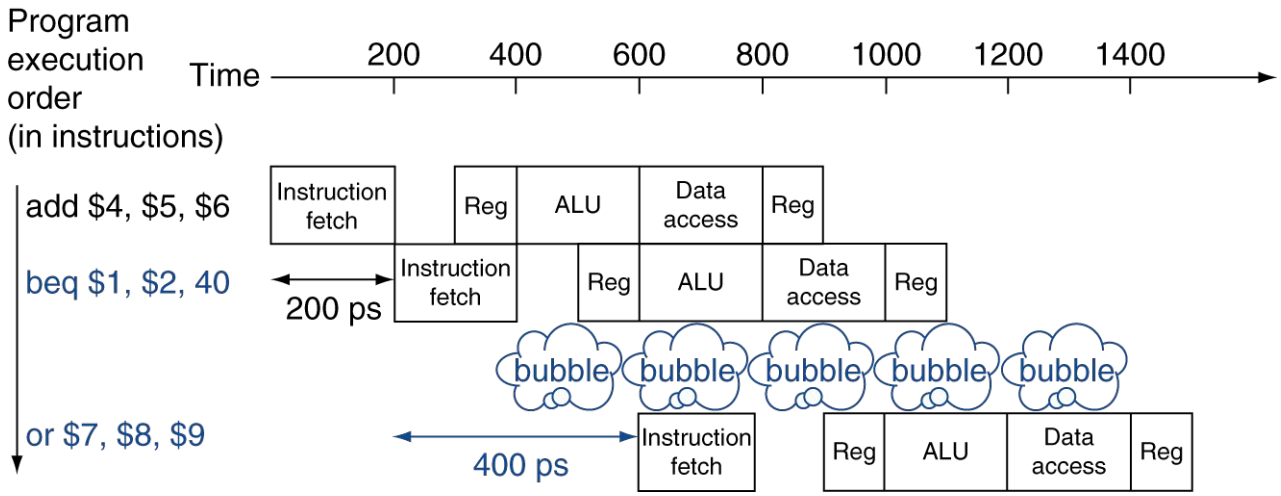
FIGURE 4.62 The ID stage of clock cycle 3 determines that a branch must be taken, so it selects 72 as the next PC address and zeros the instruction fetched for the next clock cycle. Clock cycle 4 shows the instruction at location 72 being fetched and the single bubble or nop instruction in the pipeline as a result of the taken branch. (Since the nop is really sll $0 , $0 , 0 , it's arguable whether or not the ID stage in clock 4 should be highlighted.).

Hazards de Controle

clock	1	2	3	4	5	6	7	8	9	10	11	12	13
sub $10, $4, $8	IF	ID	EX	MEM	WB								
beq $1, $3, 7		IF	ID	EX	MEM	WB							
nop			IF	ID	EX	MEM	WB						
lw $4, 50($7)				IF	ID	EX	MEM	WB					

Stall no Desvio

- Aguarde até que o resultado do desvio seja determinado antes de buscar a próxima instrução

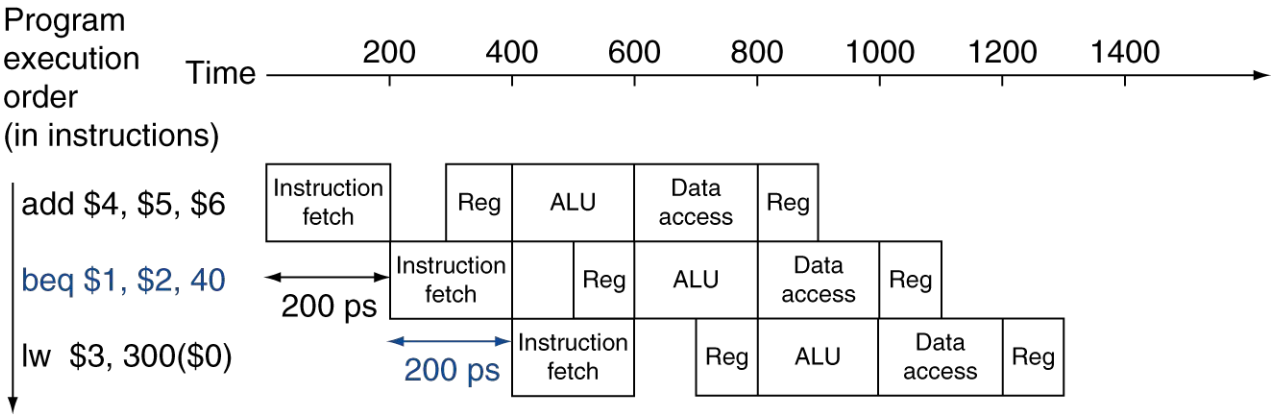


Previsão de Desvio

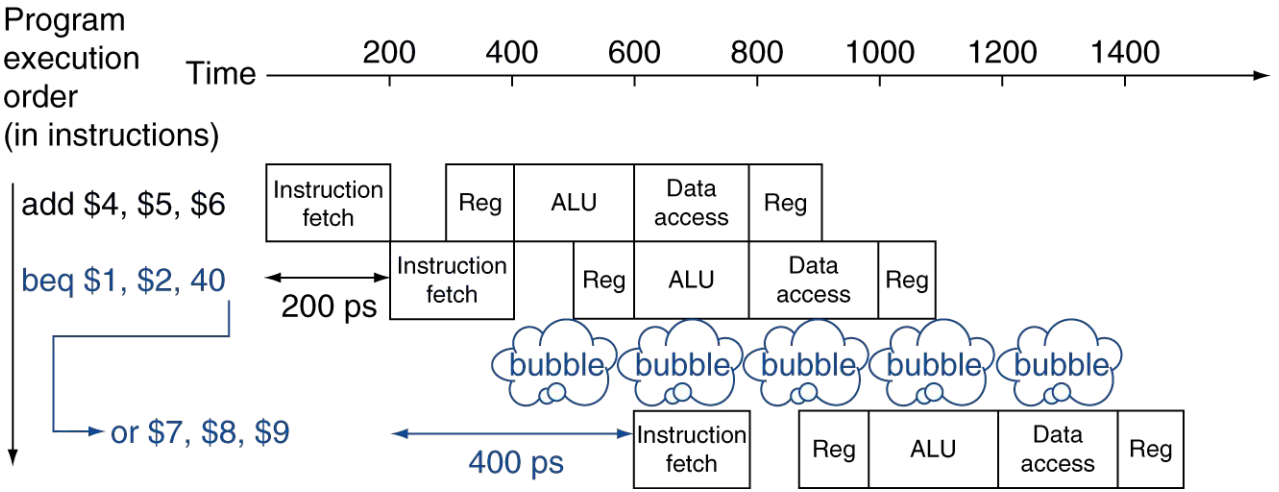
- Pipelines mais longos não conseguem determinar o resultado do desvio mais cedo
  - A penalidade de parada se torna inaceitável
- Prever o resultado do desvio
  - Parar apenas se a previsão estiver errada
- No pipeline do MIPS
  - Pode prever desvio não executado (not taken)
  - Buscar instrução depois do desvio, sem atrasos

MIPS com Previsão Não Tomado

Previsão correta



Previsão incorreta



Paralelismo Em Nível de Instrução (ILP)

- Esta seção é uma breve introdução de assuntos fascinantes, porém avançados
- No livro mais avançado, Arquitetura de Computadores: uma abordagem quantitativa, quarta edição
  - O material explicado nas próximas páginas é expandido para mais de 200 páginas (incluindo Apêndices)!

Paralelismo Em Nível de Instrução (ILP)

- Pipelining: executando várias instruções em paralelo
- Para aumentar o ILP
  - Pipeline mais profundo
    - Menos trabalho por estágio ⇒ ciclo de relógio mais curto
  - Despacho múltiplo
    - Replicar estágios do pipeline ⇒ múltiplos pipelines
    - Iniciar várias instruções por ciclo de clock
    - Ciclos de clock por instuções (CPI) < 1
      - Use Instruções por ciclo (IPC)
    - Por exemplo, despacho múltiplo de 4 vias a 4GHz
      - 16 bilhões de instruções por segundo
      - CPI de 0,25 no melhor dos casos
      - IPC de 4
    - Mas as dependências reduzem isso na prática

Paralelismo Em Nível de Instrução (ILP)

- Aumentar a profundidade do pipeline para sobrepor mais instruções
  - Usando nossa analogia da lavanderia e considerando que o ciclo da lavadora fosse maior do que os outros
  - Poderíamos dividir nossa lavadora em três máquinas que lavam, enxáguam e centrifugam, como as etapas de uma lavadora tradicional (lavadora = lavagem / enxague / centrifugar)
  - Poderíamos, então, passar de um pipeline de quatro para seis estágios

Paralelismo Em Nível de Instrução (ILP)

- Replicar componentes internos do computador de modo que ele possa iniciar várias instruções em cada estágio do pipeline
  - Uma lavanderia com despacho múltiplo substituiria nossa lavadora e secadora doméstica por, digamos, três lavadoras e três secadoras
  - Você também teria de recrutar mais auxiliares para passar e guardar três vezes a quantidade de roupas no mesmo período

Despacho Múltiplo

- Despacho múltiplo estático
  - Compilador determina quais instruções podem ser despachadas juntas
  - Empacota-as em "slots de despacho"
  - O compilador detecta e evita hazards
- Despacho múltiplo dinâmico
  - CPU examina o fluxo de instruções e escolhe instruções para despachar a cada ciclo
  - O compilador pode ajudar reordenando as instruções
  - CPU resolve hazards usando técnicas avançadas em tempo de execução

Especulação

- "Adivinhe" o que fazer com uma instrução
  - Inicie a operação o mais rápido possível
  - Verifique se o palpite estava certo
    - Nesse caso, conclua a operação
    - Caso contrário, reverta e faça a coisa certa
- Comum para despacho múltiplo estáticos e dinâmicos
- Exemplos
  - Especular sobre o resultado de um desvio
    - Reverte se o caminho percorrido for diferente
  - Especular no carregamento (load)
    - Reverte se o local for atualizado

Especulação Compilador/Hardware

- O compilador pode reordenar instruções
  - por exemplo, mover o load antes do desvio
  - Pode incluir instruções de "reparo" para recuperar de especulações incorretas
- O hardware pode aguardar instruções para executar
  - Resultados em um buffer até que não são mais especulativos
  - Esvazia buffers caso especulação incorreta

Despacho Múltiplo Estático

- O compilador agrupa instruções em "pacotes de despacho"
  - Grupo de instruções que podem ser despachadas em um único ciclo
  - Determinado pelos recursos de pipeline necessários
- Pense em um pacote de despacho como uma instrução muito longa
  - Especifica várias operações simultâneas
  - ⇒ Very Long Instruction Word (VLIW)

Escalonamento Despacho Múltiplo Estático

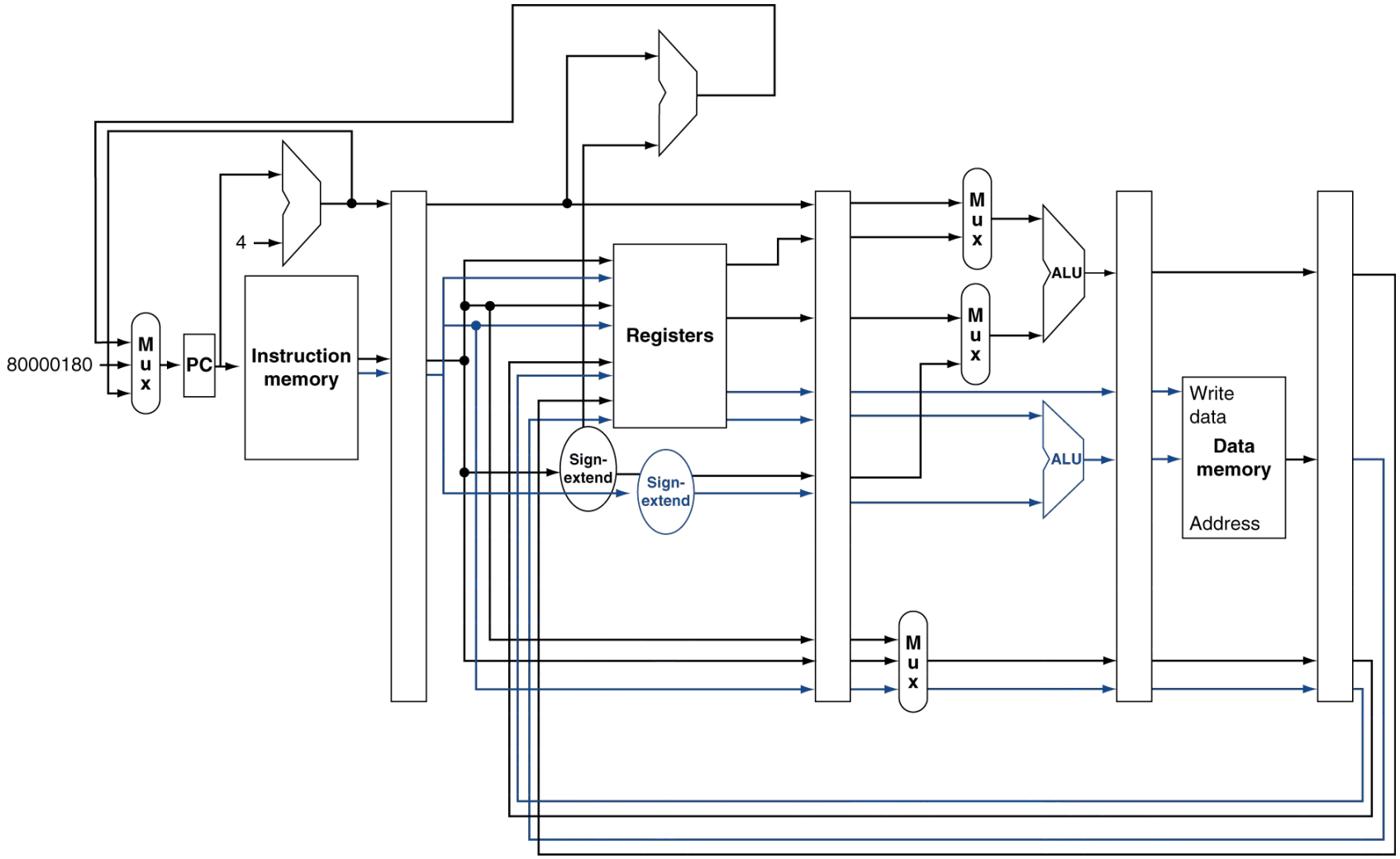
- O compilador deve remover alguns/todos hazards
  - Reordenar instruções em pacotes de emissão
  - Sem dependências em um pacote
  - Possivelmente algumas dependências entre pacotes
  - Varia entre ISAs; compilador deve saber!
  - Slot com nop, se necessário

Despacho Duplo Estático MIPS

- Pacotes de dois despachos
  - Uma instrução ULA/Desvio
  - Uma instrução de load/strore
  - Alinhado de 64 bits
    - ULA/desvio, depois load/store
    - Slot com nop em instrução não utilizada

Endereço	Tipo Instrução	Estágios Pipeline						
n	ULA/desvio	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ULA/desvio		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ULA/desvio			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

Despacho Duplo Estático MIPS



Hazards - MIPS com despacho múltiplo

- Mais instruções executando em paralelo
- Hazard de dados EX
  - Encaminhamento (Forwarding) evitou stalls no despacho úncio
  - Agora não é possível usar o resultado da ULA em load/ store no mesmo pacote
    - ```
      add  $t0, $s0, $s1
      load $s2, 0($t0)
      ```
    - Divida em dois pacotes, efetivamente um stall
- Hazard no uso load
  - Ainda um ciclo usa latência, mas agora duas instruções
- Escalonamento mais agressivo necessário
      ```

Exemplo de Escalonamento

- Escalonado para o MIPS com despacho duplo

```
Loop: lw   $t0, 0($s1)       # $t0=elemento array
      addu $t0, $t0, $s2     # add escalar em $s2
      sw   $t0, 0($s1)       # resultado do store
      addi $s1, $s1,-4       # decrementa ponteiro
      bne  $s1, $zero, Loop  # desvia se $s1!=0
```

	ULA/desvop	Load/store	ciclo
Loop:	nop	lw $t0, 0($s1)	1
	addi $s1, $s1,-4	nop	2
	addu $t0, $t0, $s2	nop	3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4

Exemplo de Escalonamento

clock	1	2	3	4	5	6	7	8	9	10	11	12	13
$s1	4	4	4	4	0	0	0	0					
nop	IF	ID	EX	MEM	WB								
lw $t0, 0($s1)	IF	ID	EX	MEM	WB								
addi $s1,&s1,–4		IF	ID	EX	MEM	WB							
nop		IF	ID	EX	MEM	WB							
addu $t0, $t0, $s2			IF	ID	EX	MEM	WB						
nop			IF	ID	EX	MEM	WB						
bne $s1, $zero, Loop				IF	ID	EX	MEM	WB					
sw $t0, 4($s1)				IF	ID	EX	MEM	WB					

Desdobramento de Loop

- Replique o corpo do loop para expor mais paralelismo
  - Reduz a sobrecarga de controle de loop
- Usar diferentes registradores por replicação
  - Chamado de “renomeação de registradores”
  - Evitar "anti-dependências" ocosionadas pelo loop
    - Store seguido por um load do mesmo registrador
    - Também conhecido como “dependência de nome”
      - Reutilização de um nome de registrador

Exemplo Desdobramento Loop

	ULA/desvio	Load/store	ciclo
Loop:	addi $s1, $s1,-16	lw $t0, 0($s1)	1
	nop	lw $t1, 12($s1)	2
	addu $t0, $t0, $s2	lw $t2, 8($s1)	3
	addu $t1, $t1, $s2	lw $t3, 4($s1)	4
	addu $t2, $t2, $s2	sw $t0, 16($s1)	5
	addu $t3, $t4, $s2	sw $t1, 12($s1)	6
	nop	sw $t2, 8($s1)	7
	bne $s1, $zero, Loop	sw $t3, 4($s1)	8

clock	1	2	3	4	5	6	7	8	9	10	11	12	13
$s1	16	16	16	16	16	0	0	0					
addi $s1, $s1,-16	IF	ID	EX	MEM	WB								
lw $t0, 0($s1)	IF	ID	EX	MEM	WB								
nop		IF	ID	EX	MEM	WB							
lw $t1, 12($s1)		IF	ID	EX	MEM	WB							
addu $t0, $t0, $s2			IF	ID	EX	MEM	WB						
lw $t2, 8($s1)			IF	ID	EX	MEM	WB						
addu $t1, $t1, $s2				IF	ID	EX	MEM	WB					
lw $t3, 4($s1)				IF	ID	EX	MEM	WB					
addu $t2, $t2, $s2					IF	ID	EX	MEM	WB	IF			
sw $t0, 16($s1)					IF	ID	EX	MEM	WB	IF			
addu $t3, $t4, $s2						IF	ID	EX	MEM	WB	IF		
sw $t1, 12($s1)						IF	ID	EX	MEM	WB	IF		
nop							IF	ID	EX	MEM	WB	IF	
sw $t2, 8($s1)							IF	ID	EX	MEM	WB	IF	
bne $s1, $zero, Loop								IF	ID	EX	MEM	WB	IF
sw $t3, 4($s1)								IF	ID	EX	MEM	WB	IF

Despacho Múltiplo Dinâmico

- Processadores “superescalares”
- CPU decide se deve emitir 0, 1, 2, ... cada ciclo
  - Evitando riscos estruturais e de dados
- Evita a necessidade escalonamento no compilador
  - Embora ainda possa ajudar
  - Semântica de código garantida pela CPU

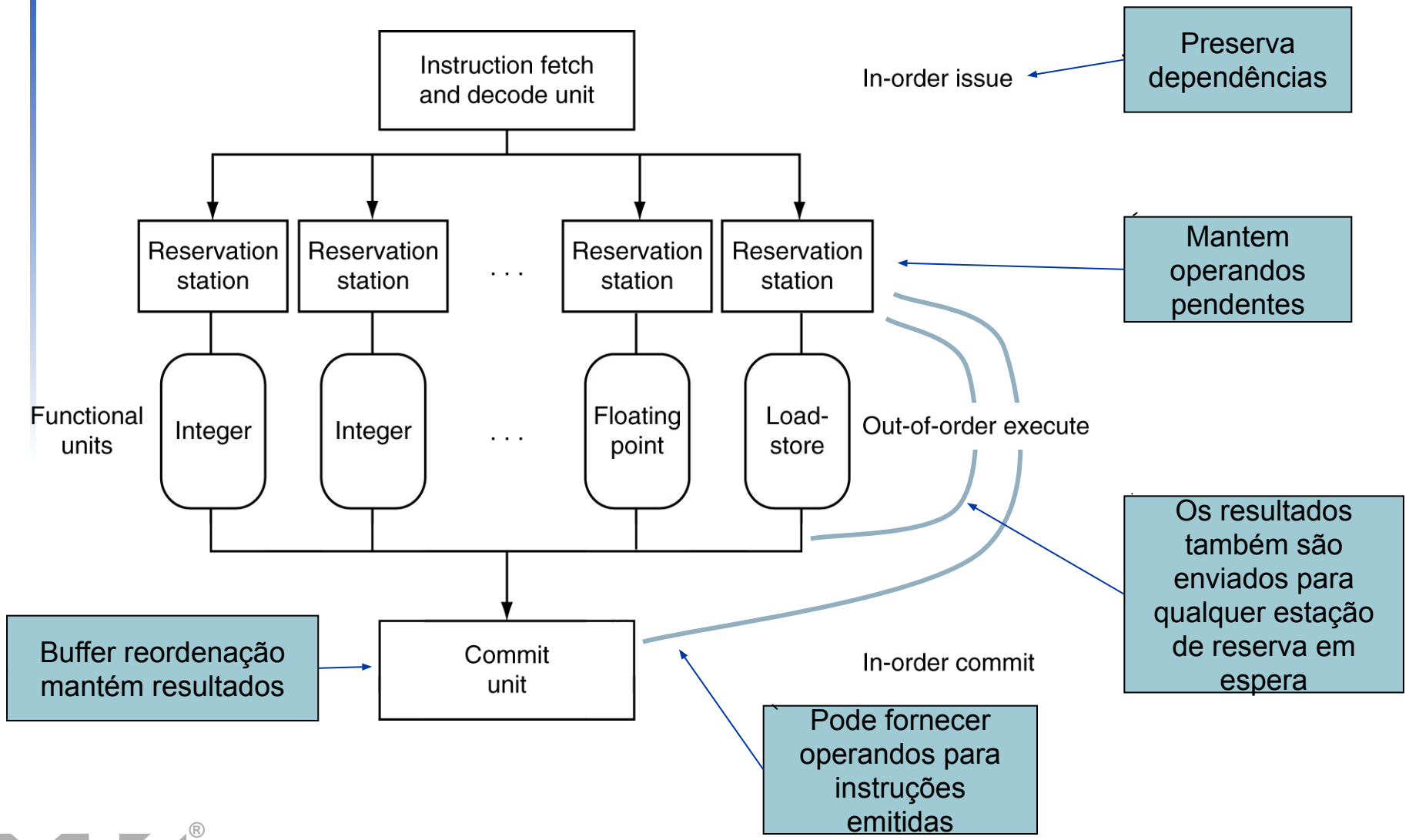
Escalonamento Dinâmico em Pipeline

- Permitir que a CPU executar instruções fora de ordem para evitar stalls
  - Mas commit o resultado nos registradores em ordem
- Exemplo

```
lw    $t0, 20($s2)
addu  $t1, $t0, $t2
sub   $s4, $s4, $t3
slti  $t5, $s4, 20
```

  - Pode iniciar sub enquanto addu está esperando por lw

CPU - Escalonamento Dinâmico



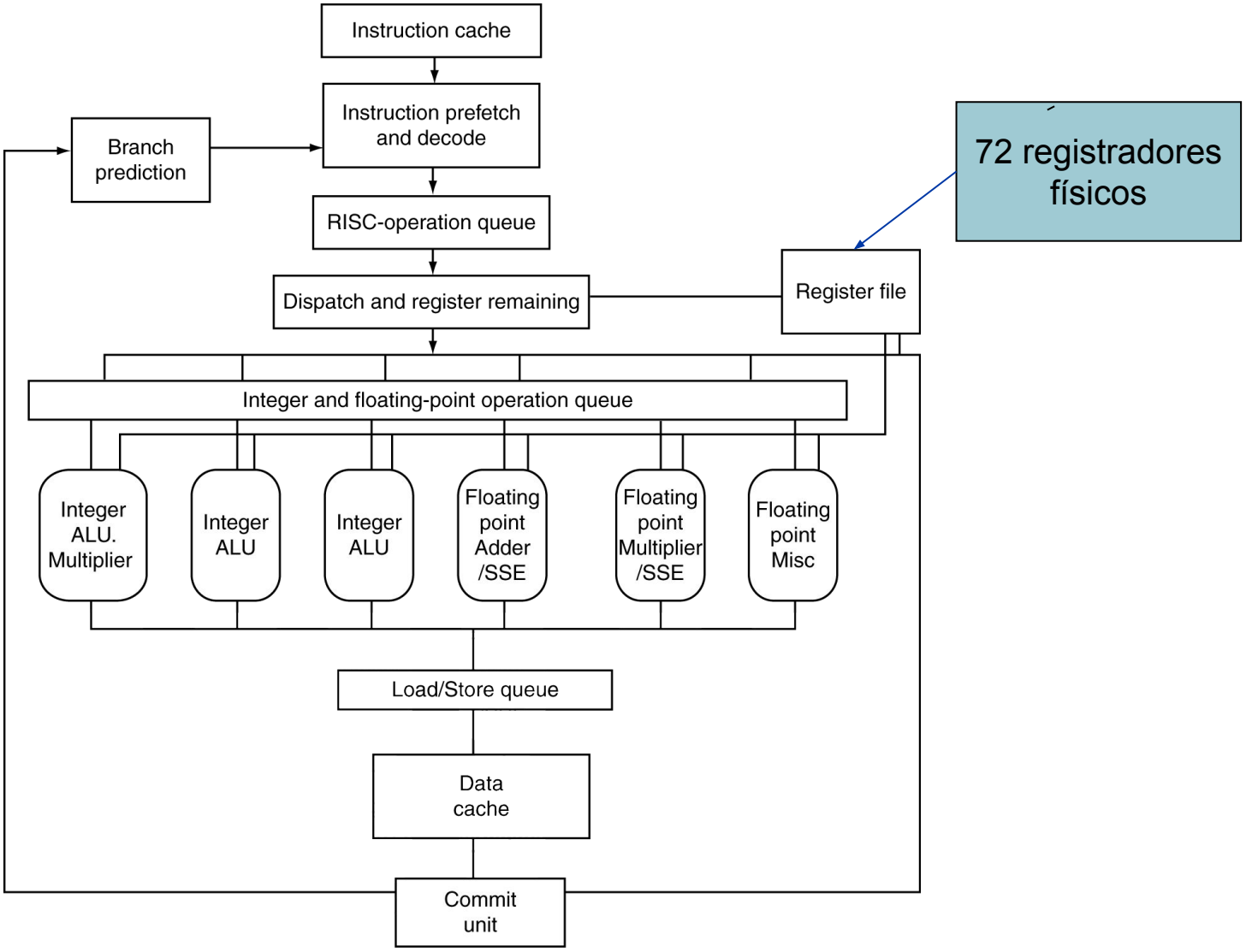
Eficiência energética

- A complexidade do despacho múltiplo dinâmico e especulação tem baixa eficiência energética
- Vários núcleos mais simples podem ser melhores

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	Power	
Intel 486	1989	25 MHz	5	1	No	1	5	W
Intel Pentium	1993	66 MHz	5	2	No	1	10	W
Intel Pentium Pro	1997	200 MHz	10	3	Yes	1	29	W
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes	1	75	W
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes	1	103	W
Intel Core	2006	2930 MHz	14	4	Yes	2	75	W
Intel Core i5 Nehalem	2010	3300 MHz	14	4	Yes	1	87	W
Intel Core i5 Ivy Bridge	2012	3400 MHz	14	4	Yes	8	77	W

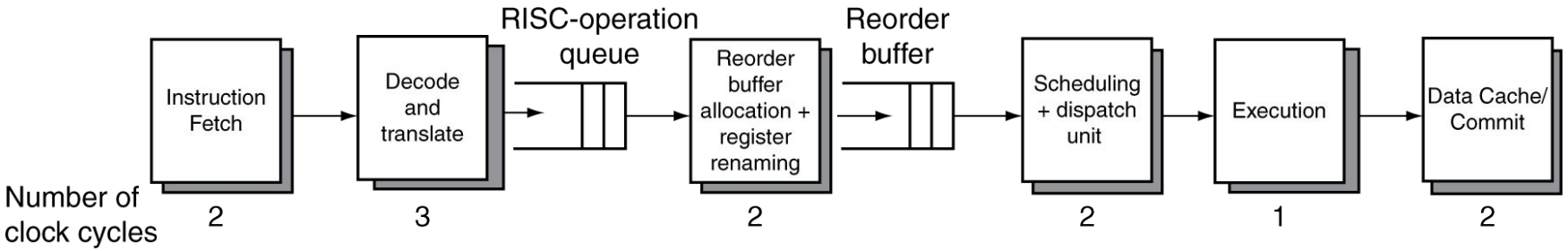
FIGURE 4.73 Record of Intel Microprocessors in terms of pipeline complexity, number of cores, and power. Th e Pentium 4 pipeline stages do not include the commit stages. If we included them, the Pentium 4 pipelines would be even deeper.

A Microarquitetura do AMD Optron X4



O Fluxo Pipeline Opteron X4

- Para instruções com inteiros



  - 12 estágios (Ponto flutuante é 5 estágios mais longo)
  - Até 106 operações RISC em andamento
- Gargalos
  - Instruções complexas com longas dependências
  - Erros de previsão do desvio
  - Atrasos de acesso à memória

Referências

- Seções 4.5, 4.10 e 4.11- Organização e Projeto de Computadores - A Interface Hardware/Software, David A. Patterson & John L. Hennessy, Campus, 4 edição, 2013.
- Seção 4.10- Computer organization and design: the hardware/software interface/David A. Patterson, John L. Hennessy, Elsevier, 5th ed, 2013.