



Capítulo 4

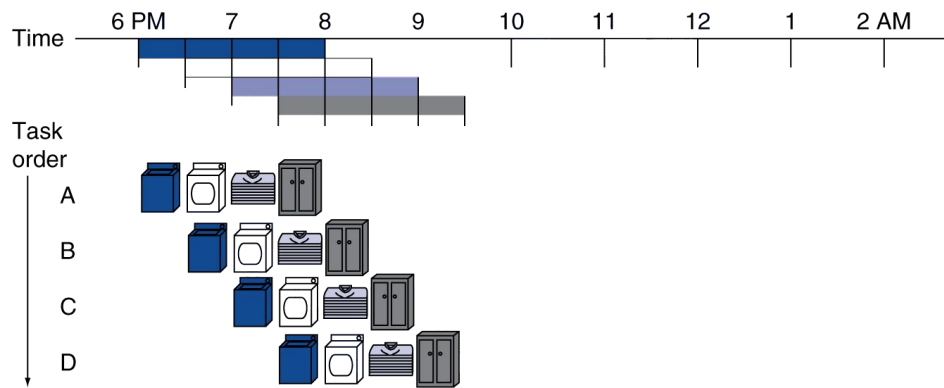
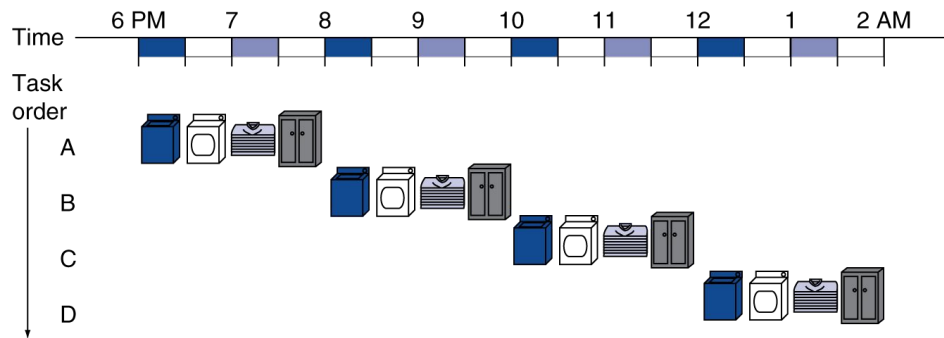
O Processador

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:
 - $\text{Speedup} = 8 / 3,5 = 2,3$

Pipeline MIPS

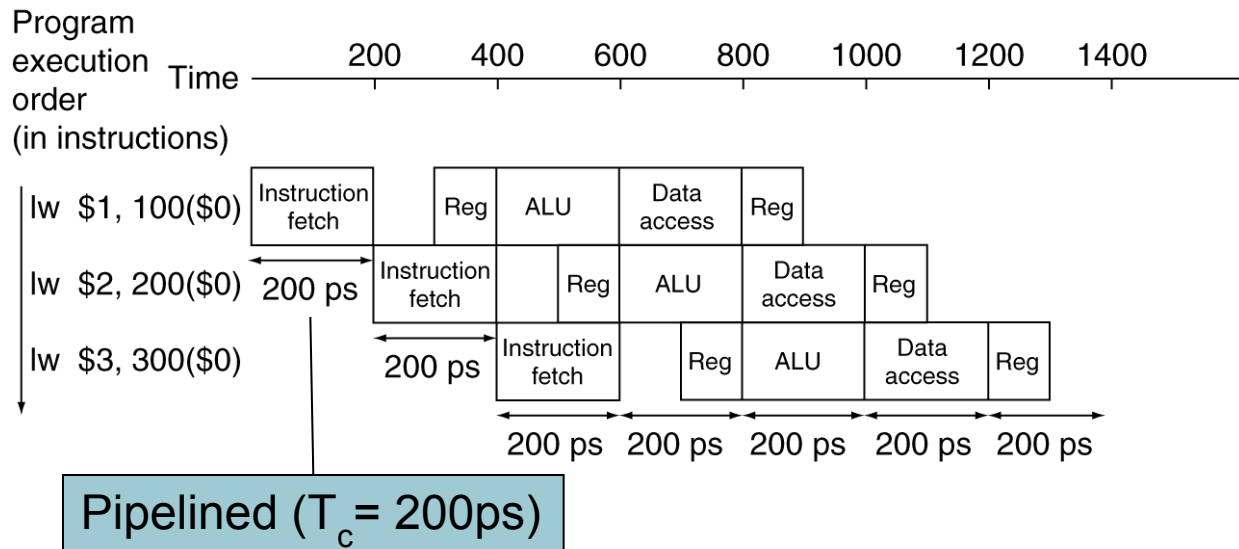
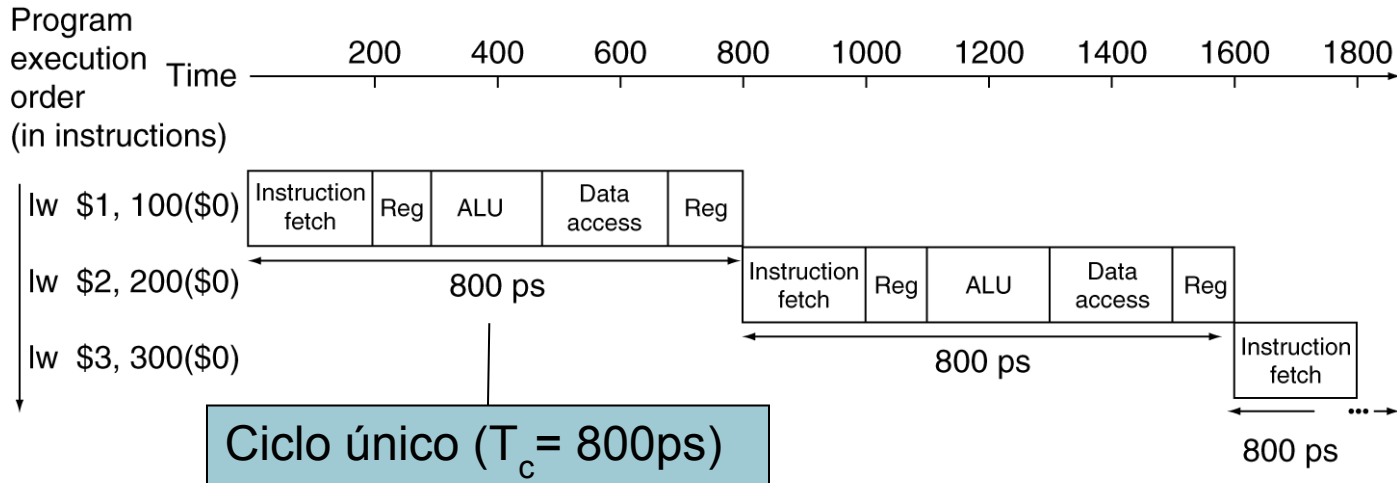
1. **IF: Instruction Fetch**
 - Busca instrução da memória
2. **ID: Instruction Decode**
 - Lê os registradores enquanto a instrução é decodificada
3. **EX: Execute**
 - Executa a operação ou calcular um endereço
4. **MEM: Memory**
 - Acessa um operando na memória de dados
5. **WB: Write result Back**
 - 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
 - $$\text{Tempo entre as instruções}_{\text{com pipeline}} = \frac{\text{Tempo entre as instruções}_{\text{sem pipeline}}}{\text{Número de estágios}}$$
- 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

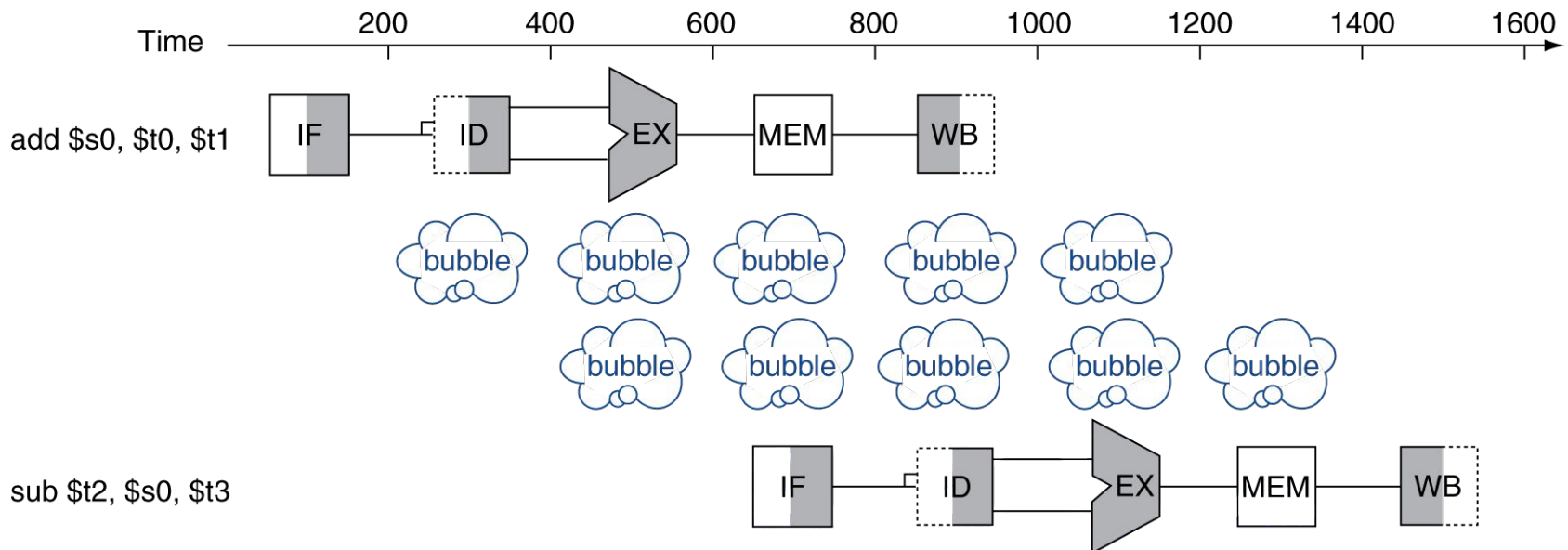
- Situações que impedem o início da próxima instrução no próximo ciclo
- Conflitos estruturais (Structure Hazards)
 - Um recurso necessário está ocupado
- Conflitos de dados (Data Hazards)
 - Precisa aguardar instruções anteriores para concluir a leitura/escrita de dados
- Conflitos de controle (Control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

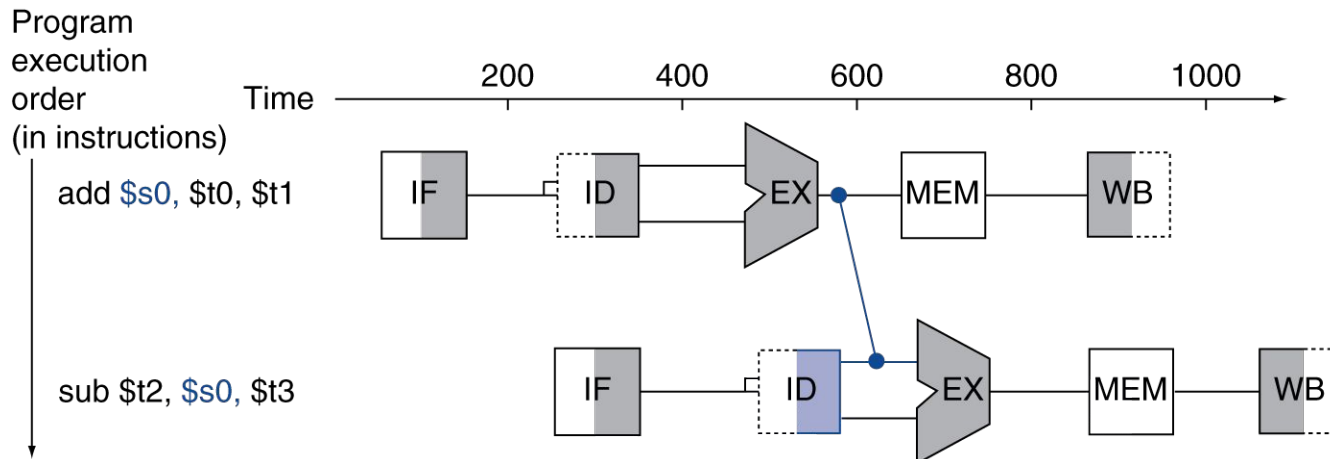
Hazards de Dados

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



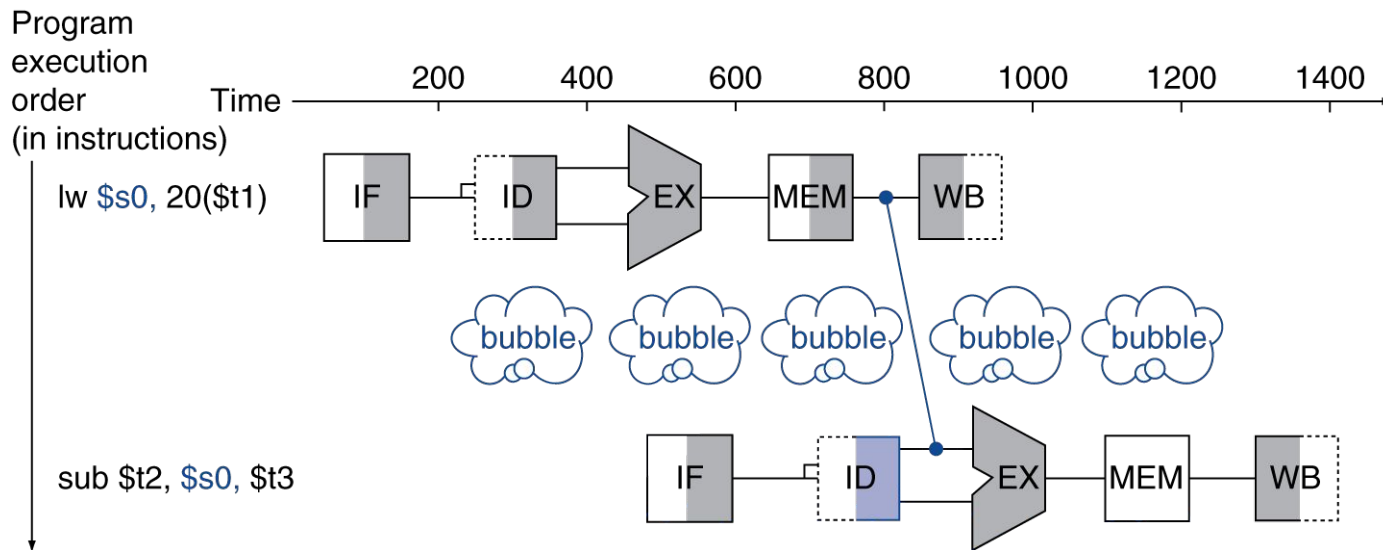
Encaminhamento (Forwarding)

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



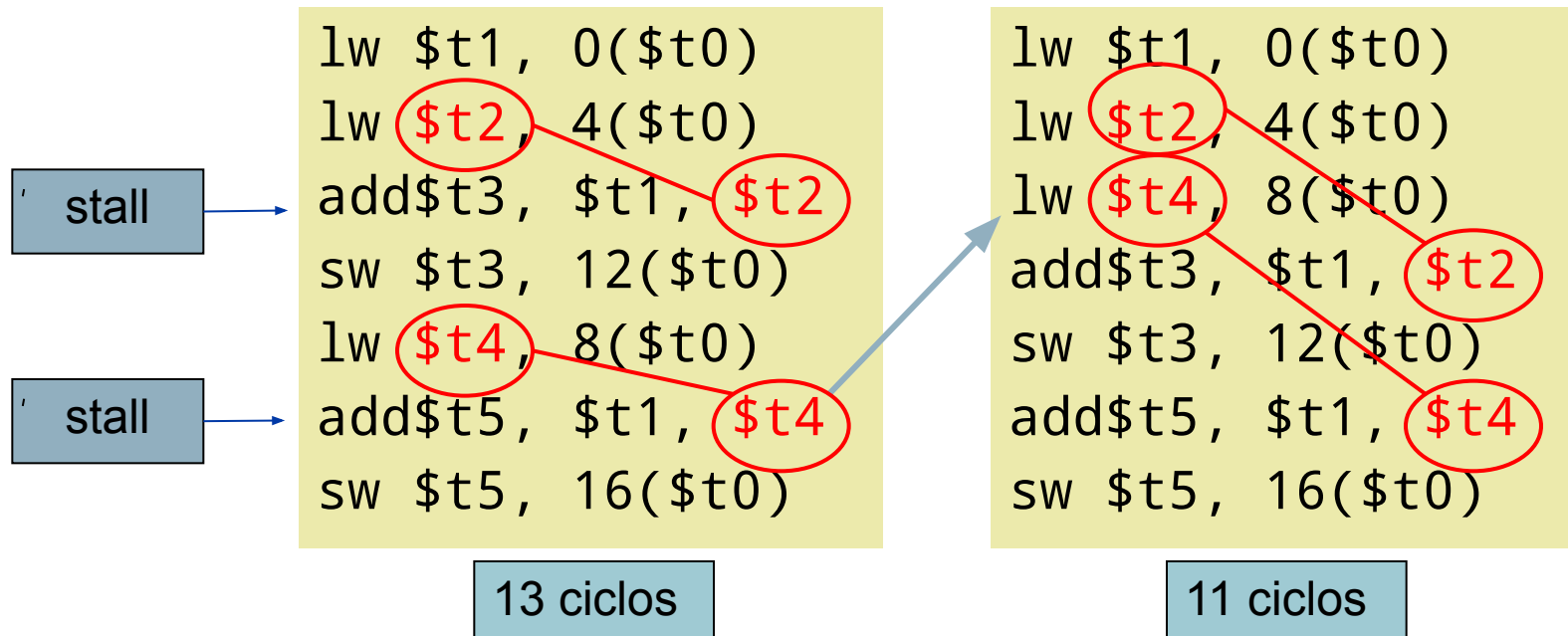
Hazard de Dados Uso Load

- Nem sempre é possível evitar paradas com forwarding
 - 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 $A = B + E$; $C = B + F$;

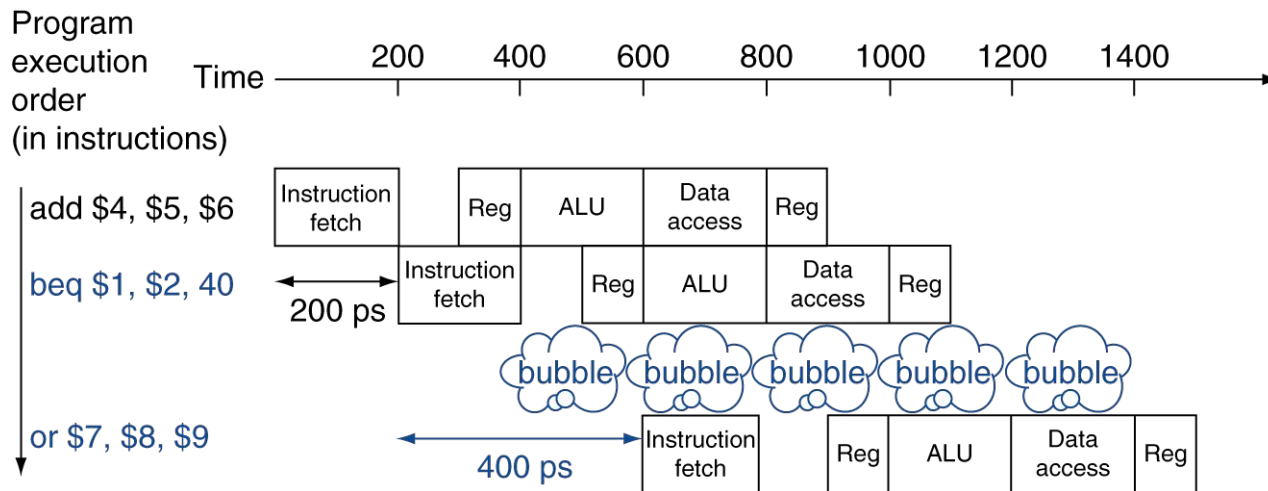


Hazards de Controle

- 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)

Stall no Desvio

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

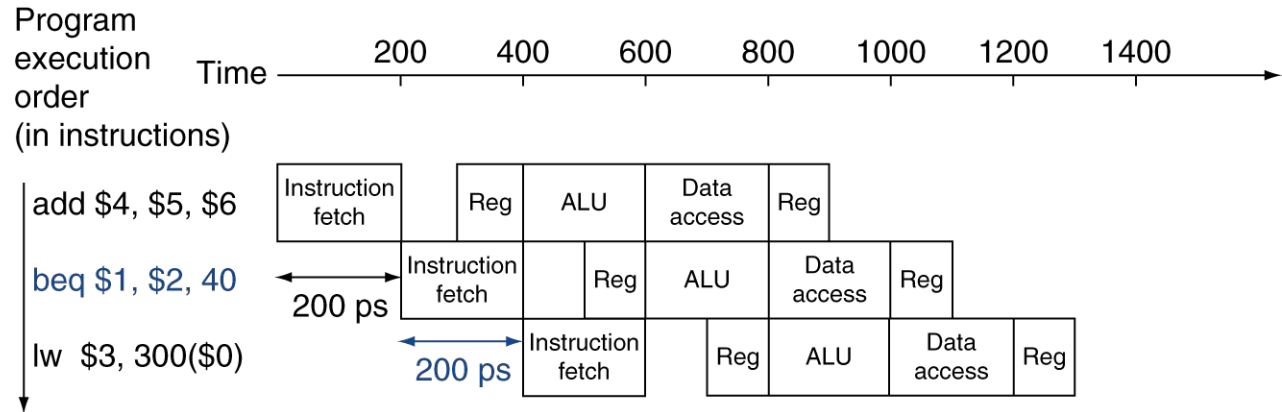


Previsão de Desvio

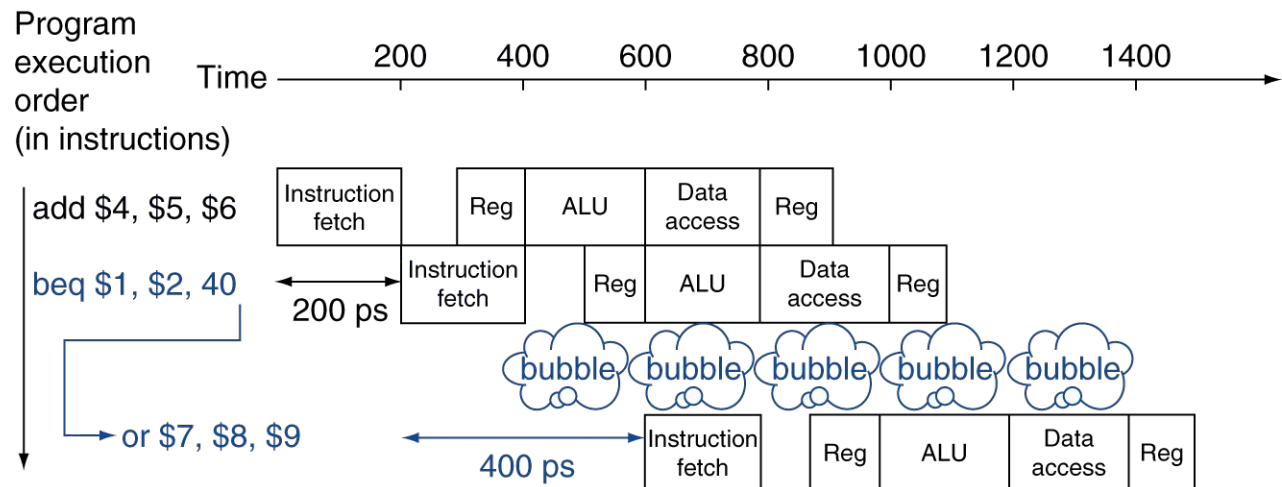
- Pipelines mais longos não conseguem determinar o resultado do desvio mais cedo
 - A penalidade de parada se torna inaceitável
- Prever o resultado do desvio
 - Parar apenas se a previsão estiver errada
- No pipeline do MIPS
 - Pode prever desvio não executado (not taken)
 - Buscar instrução depois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

Tomado

Previsão
correta



Previsão
incorreta



Previsão de Desvio Mais Realista

- Previsão de desvio estático
 - Com base no comportamento típico do desvio
 - Exemplo: desvio de loop e instrução if
 - Prever cos desvios para trás como tomados
 - Prever os desvios para frente com não tomados
- Previsão dinâmica de desvio
 - O hardware mede o comportamento real do desvio
 - por exemplo, manter o histórico recente de cada desvio
 - Suponha que o comportamento futuro continuará a tendência
 - Quando estiver errado, pare ao buscar novamente e atualize o histórico

Resumo do Pipeline

Colocando em perspectiva

- O pipelining melhora o desempenho aumentando a vazão de instruções
 - Executa várias instruções em paralelo
 - Cada instrução tem a mesma latência
- Sujeito a hazards (conflitos)
 - Estrutura, dados, controle
- O design do conjunto de instruções afeta a complexidade da implementação do pipeline

Previsão Dinâmica de Desvios

- Em pipelines mais profundos e superescalares, a penalidade de desvio é mais significativa
- Usar previsão dinâmica
 - Buffer de previsão de desvios (também conhecido como tabela de histórico de desvios)
 - Indexado por endereços de instrução de desvios recentes
 - Resultado dos desvios (tomados/não tomados)
 - Para executar um desvio
 - Verifique a tabela, espere o mesmo resultado
 - Comece a buscar do destino (tomados) ou próxima instrução (não tomado)
 - Se estiver errado, esvazie o pipeline e inverta a previsão

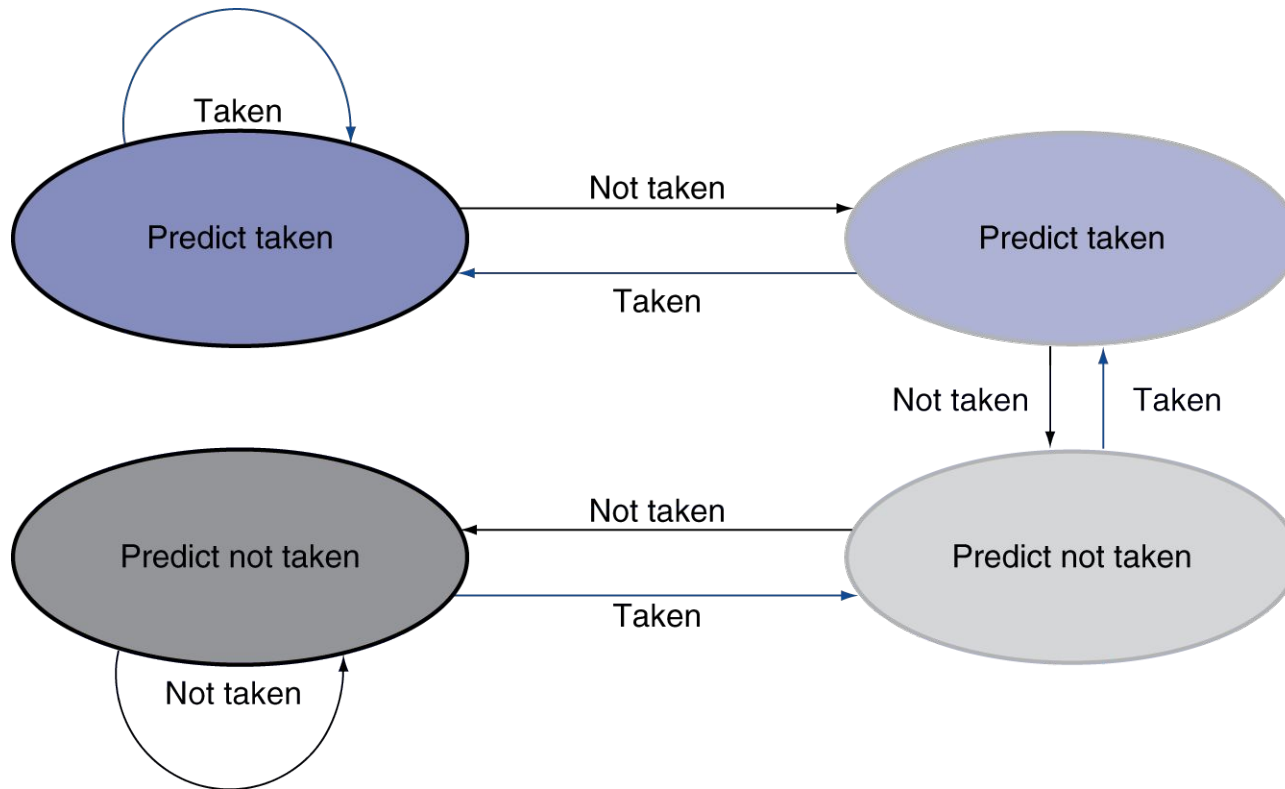
Previsão 1-Bits: Desvantagem

- Considere um desvio de loop que se desvia nove vezes seguidas, depois não é tomado uma vez
- O comportamento da previsão de estado fixo fará uma previsão errada na primeira e última iterações do loop



Previsão 2-Bits

- Apenas mude a previsão em duas previsões erradas sucessivas



Paralelismo Em Nível de Instrução (ILP)

- Pipelining: executando várias instruções em paralelo
- Para aumentar o ILP
 - Pipeline mais profundo
 - Menos trabalho por estágio \Rightarrow ciclo de relógio mais curto
 - Despacho múltiplo
 - Replicar estágios do pipeline \Rightarrow múltiplos pipelines
 - Iniciar várias instruções por ciclo de clock
 - $CPI < 1$, use Instruções por ciclo (IPC)
 - Por exemplo, despacho múltiplo de 4 vias a 4GHz
 - 16 BIPS, pico de $CPI = 0,25$, pico de $IPC = 4$
 - Mas as dependências reduzem isso na prática

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

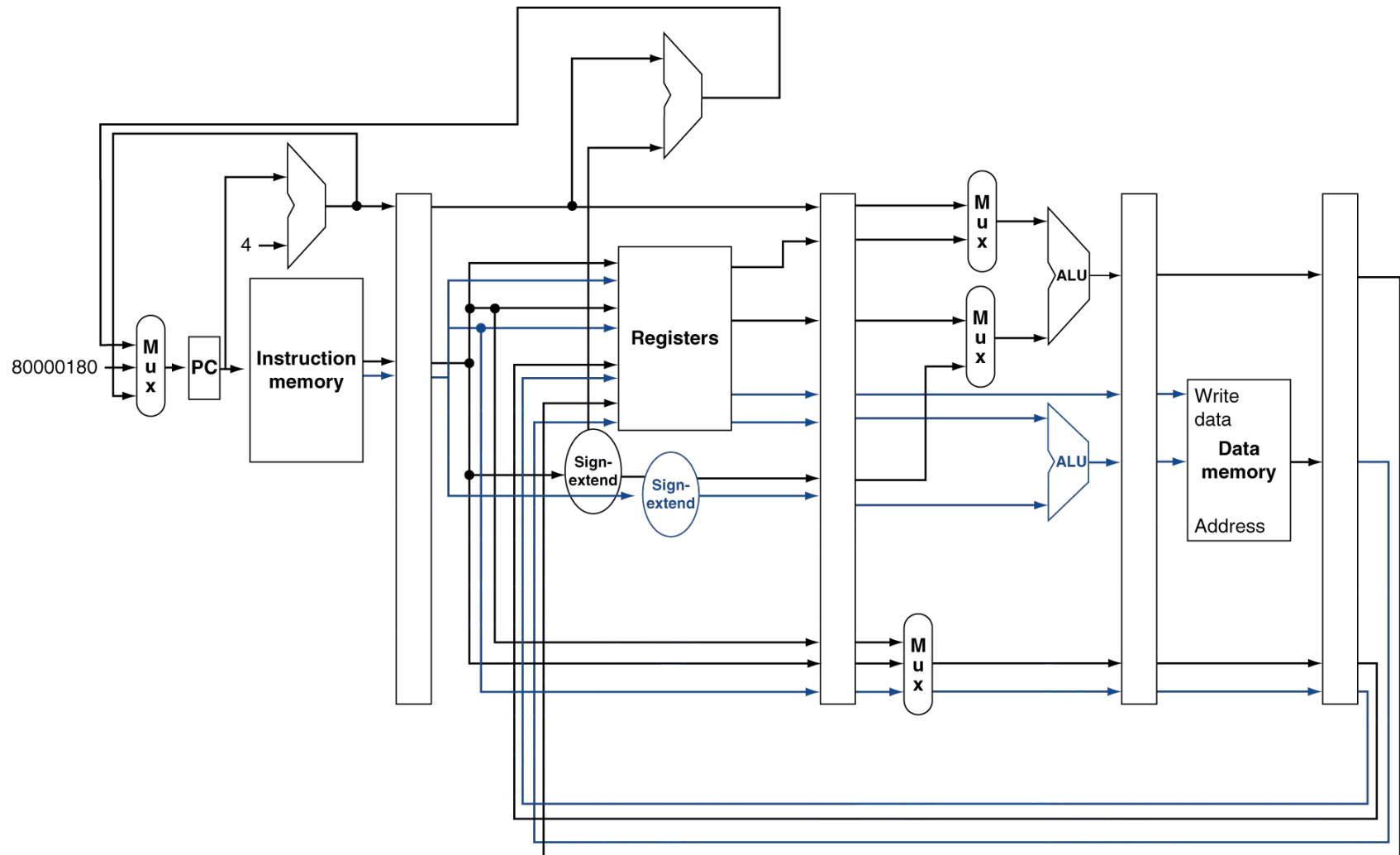
- O compilador deve remover alguns/todos hazards
 - Reordenar instruções em pacotes de emissão
 - Sem dependências em um pacote
 - Possivelmente algumas dependências entre pacotes
 - Varia entre ISAs; compilador deve saber!
 - Slot com nop, se necessário

Despacho Duplo Estático MIPS

- Pacotes de dois despachos
 - Uma instrução ULA/Desvio
 - Uma instrução de load/store
 - 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 de baias evitadas com problema único
 - 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

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" ocasionadas 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

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

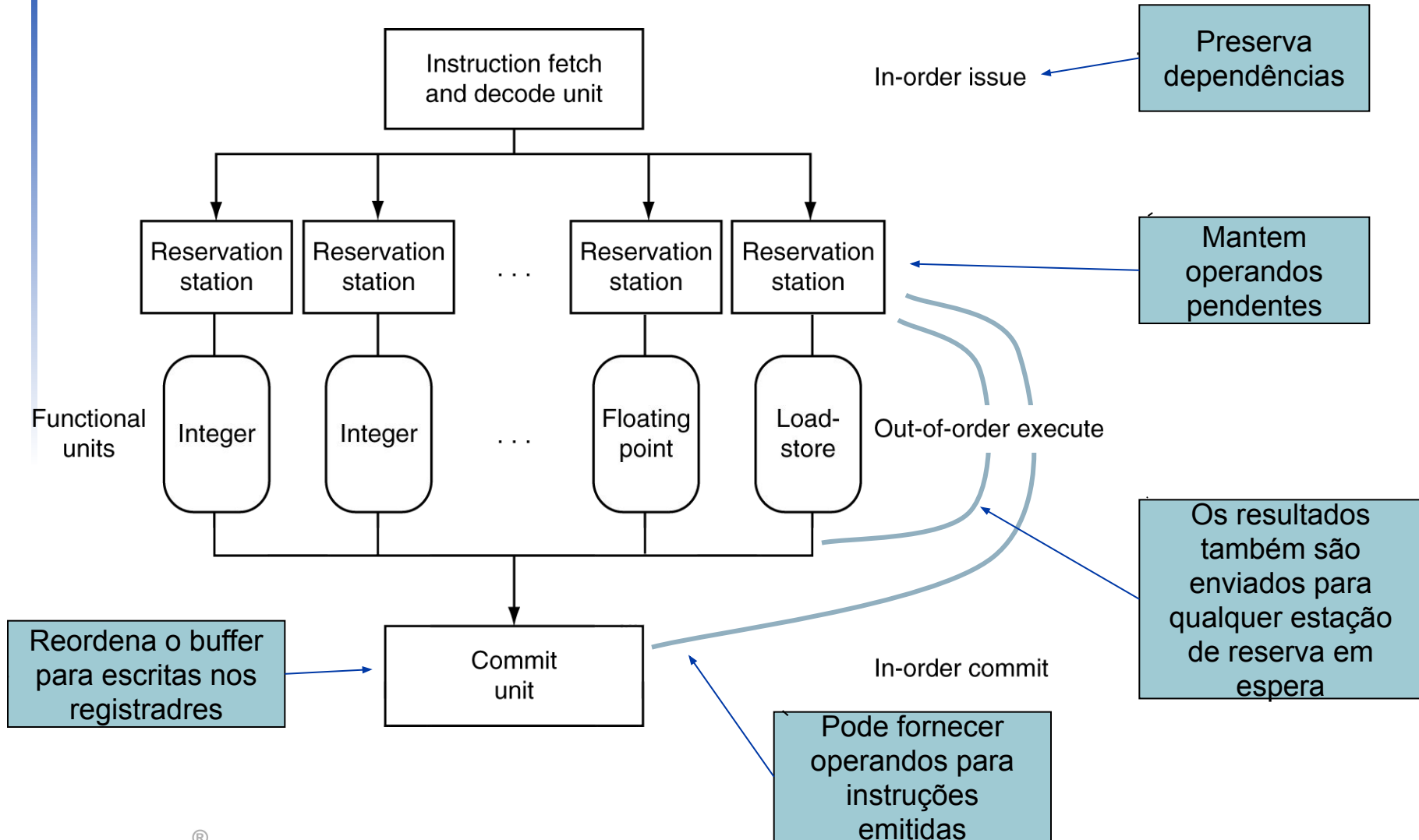
- Permitir que a CPU executar instruções fora de ordem para evitar stalls
 - Mas commit o resultado nos registradores em ordem

- 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



Renomeação de Registradores

- Estações de reserva e buffer de reordenamento fornecem renomeação de registradores
- Em emissão de instrução para a estação de reserva
 - Se o operando estiver disponível no banco de registradores ou no buffer de reordenação
 - Copiado para a estação de reserva
 - Não é mais necessário no registrador; pode ser sobrescrito
 - Se o operando ainda não estiver disponível
 - Será fornecido à estação de reserva por uma unidade funcional
 - A atualização no registrador pode não ser necessária

Especulação

- Prever desvio e continuar despachando
 - Como as instruções possuem um commit em ordem, sabemos se o desvio foi previsto corretamente ou não antes que quaisquer instruções do caminho previsto tenham seus resultados atualizados pelas unidades de commit
- Especulação nos endereços de load
 - Permitindo uma reordenação load-store e usando a unidade de commit para evitar a especulação incorreta

Por Que Fazer Agendamento dinâmico?

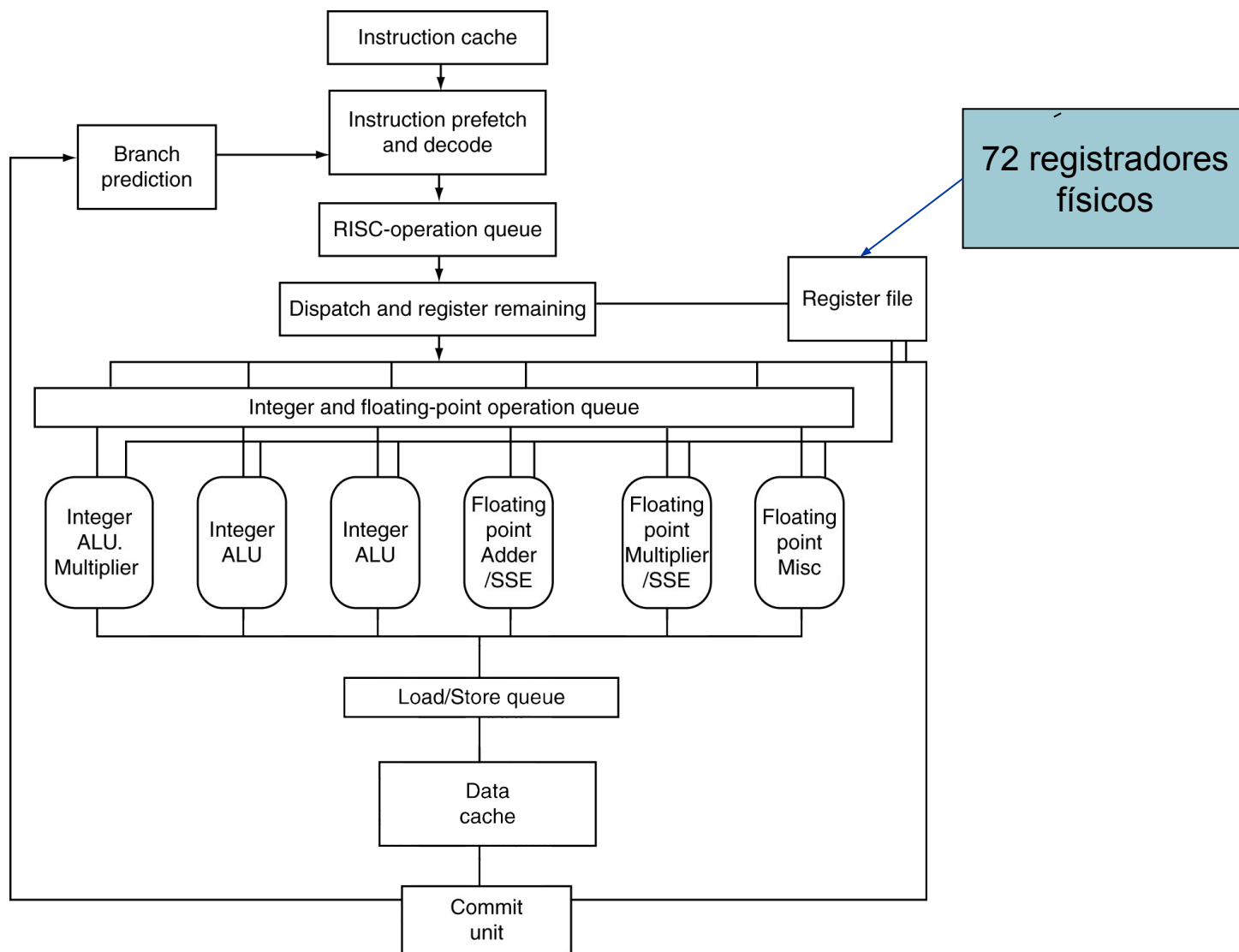
- Por que não deixar o compilador escalonar o código?
- Nem todos os stalls são previsíveis
 - Por exemplo, falhas de cache
- Nem sempre é possível escalonar perto dos desvios
 - O resultado do desvio é determinado dinamicamente
- Diferentes implementações de uma ISA têm diferentes latências e hazards

Eficiência energética

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

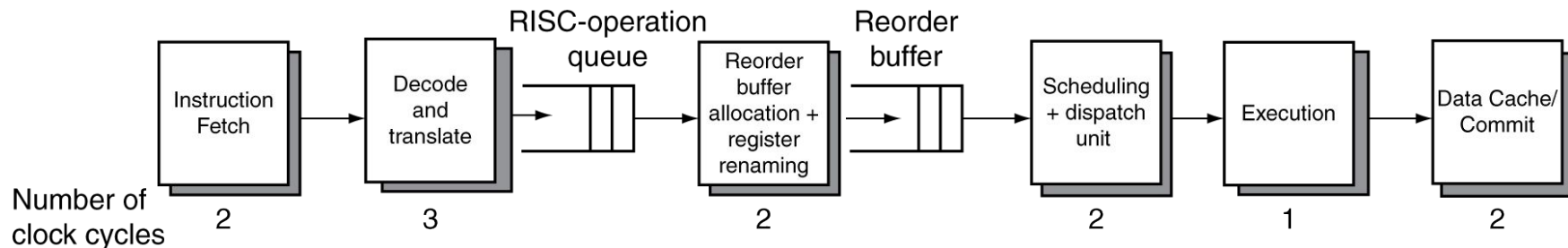
Microprocessador	Ano	Taxa Clock	Estágios Pipeline	Largura despacho	Fora de ordem/ especulação	Cores/c hip	Potência
i486	1989	25MHz	5	1	No	1	5W
Pentium	1993	66MHz	5	2	No	1	10W
Pentium Pro	1997	200MHz	10	3	Yes	1	29W
P4 Willamette	2001	2000MHz	22	3	Yes	1	75W
P4 Prescott	2004	3600MHz	31	3	Yes	1	103W
Core	2006	2930MHz	14	4	Yes	2	75W
UltraSparc III	2003	1950MHz	14	4	No	1	90W
UltraSparc T1	2005	1200MHz	6	1	No	8	70W

A Microarquitetura do AMD Opteron 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

Comentários Finais

- ISA influencia o design do caminho de dados e controle
- O caminho de dados e o controle influenciam o design do ISA
- O pipelining melhora o rendimento da instrução usando paralelismo
 - Mais instruções concluídas por segundo
 - Latência para cada instrução não reduzida
- Riscos: estruturais, dados, controle
- Problemas múltiplos e programação dinâmica (ILP)
 - Dependências limitam o paralelismo alcançável
 - A complexidade leva ao muro da potência.