Arquitetura de Computadores I

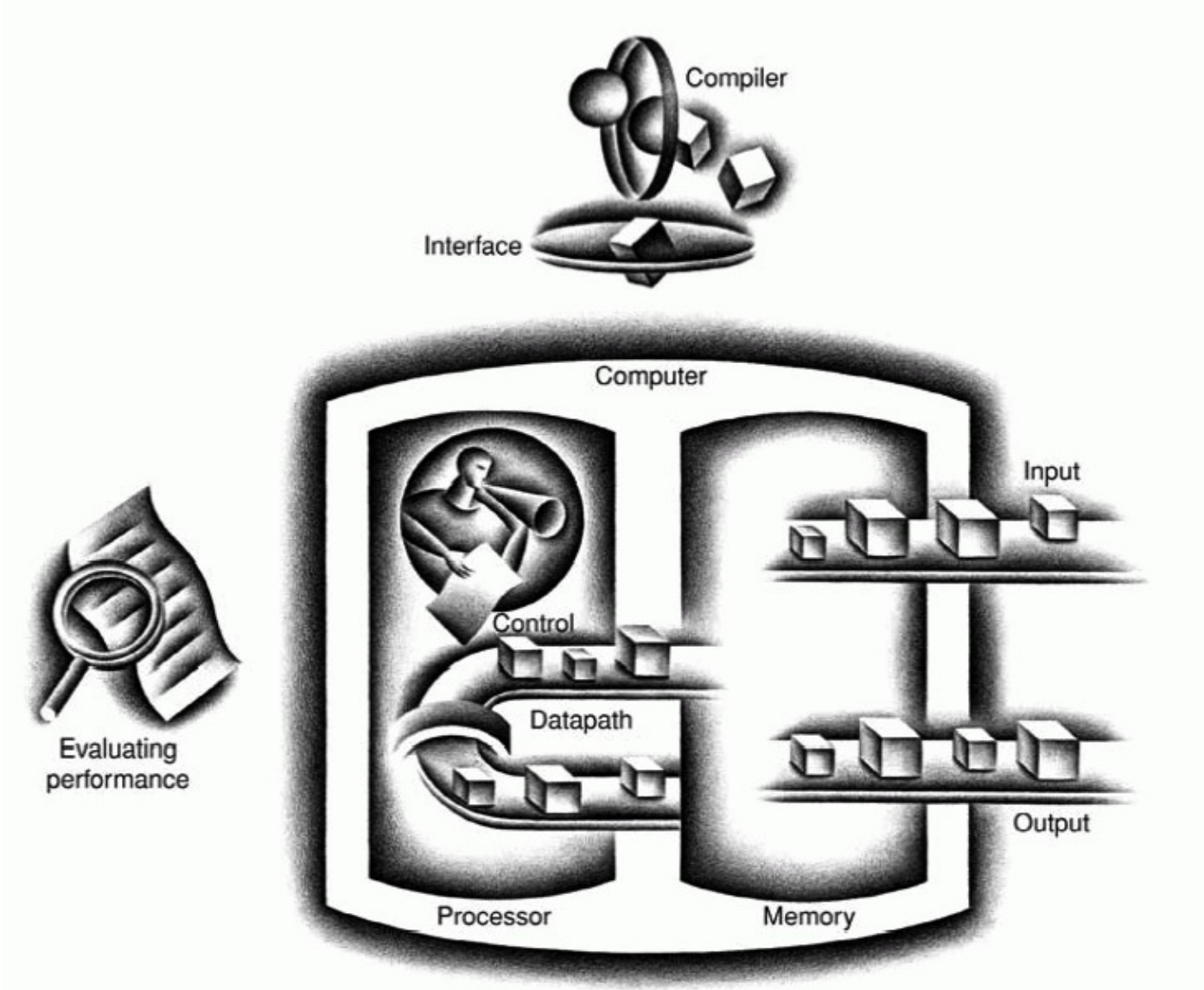
Cap. 06 – Pipeline

Prof. M.Sc. Bruno R. Silva

Plano de aula

- Visão geral de pipelining
- Um caminho de dados usando pipelie
- Controle de um pipeline
- Hazards de dados e forwarding
- Hazards de dados e stals
- Hazards de desvio

A Grande Figura!



Visão Geral

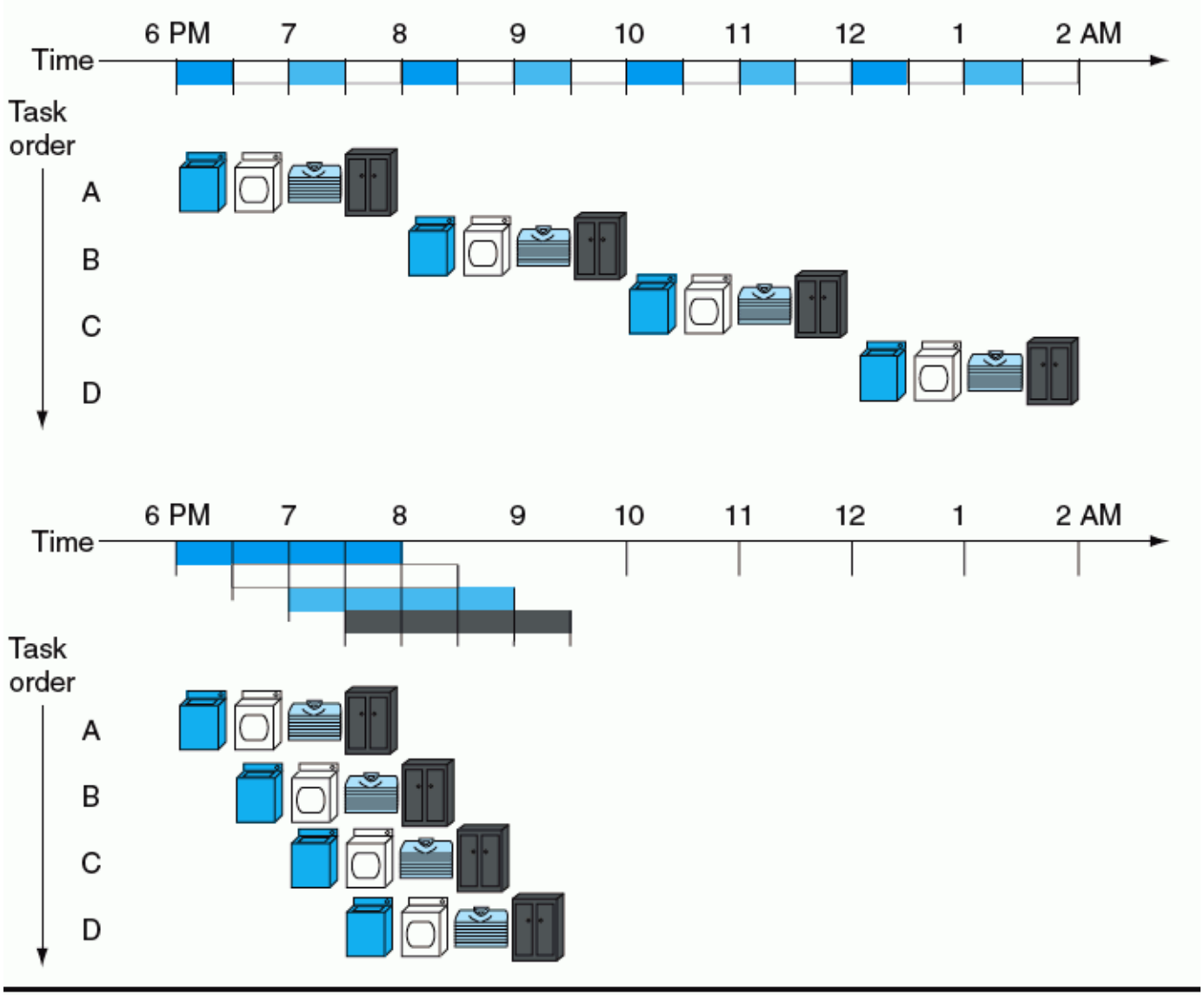
Pipelining é uma técnica de implementação em que várias instruções são sobrepostas na execução.

Etapas para se Lavar roupas

1. Colocar a trouxa suja de roupas na lavadora
2. Quando a lavadora terminar, colocar a trouxa molhada na secadora (se houver)
3. Quando a secadora terminar, colocar a trouxa seca na mesa e passar
4. Quando terminar de passar, pedir ao seu colega de quarto para guardar as roupas

Será que faz sentido esperar seu colega terminar, para só então começar a lavar a próxima trouxa de roupas?

Visão Geral


6 PM
7
8
9
10
11
12
1
2 AM
Time
Task order
A
B
C
D
6 PM
7
8
9
10
11
12
1
2 AM
Time
Task order
A
B
C
D

Visão Geral

- O paradoxo da técnica de pipelining é que o tempo desde a colocação de uma única trouxa de roupas suja na lavadora até que ela seja secada, passada e guardada não é mais curto para a técnica de pipelining.
  - A grande sacada é que tudo está operando em paralelo

A técnica pipeline melhora a vazão do sistema, sem melhorar o tempo para concluir uma única tarefa

- Se todos os estágios levarem aproximadamente o mesmo tempo e houver trabalho suficiente para realizar, então o ganho de velocidade devido à tecnica de pipelining será **igual ao número de estágios do pipeline**.

- Assim a lavanderia em pipeline é potencialmente quatro vezes mais rápida do que a sema pipeline.
  - 20 trouxas levariam 5 vezes o tempo de 1 trouxa
  - Na lavagem sequencia, 20 trouxas levariam 20 vezes o tempo de 1 trouxa
  - Por que então no exemplo anterior o ganho foi apenas de 2,3 vezes?

Visão Geral

- As instruções MIPS normalmente exigem 5 etapas
  1. Buscar instrução da memória
  2. Ler registradores enquanto a instrução é decodificada
  3. Executar a operação ou calcular o endereço
  4. Acessar um operando na memória de dados
  5. Escrever o resultado em um registrador

- Logo, vamos implementar um caminho de dados multiciclo em pipeline
  - Para simplificar a implementação, vamos concentrar a atenção em apenas 8 instruções: `lw, sw, add, sub, and, or, slt e beq`.

Visão Geral

Desempenho de ciclo único versus desempenho com pipeline

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

FIGURE 6.2 Total time for each instruction calculated from the time for each component. This calculation assumes that the multiplexors, control unit, PC accesses, and sign extension unit have no delay.

- **O projeto de ciclo único precisa contemplar a instrução mais lenta (lw com 800 ps)**
- **No projeto em pipeline, todos os estágios utilizam um único de ciclo de clock e portanto precisa ser grande o suficiente para acomodar a operação mais lenta (200 ps)**

Visão Geral

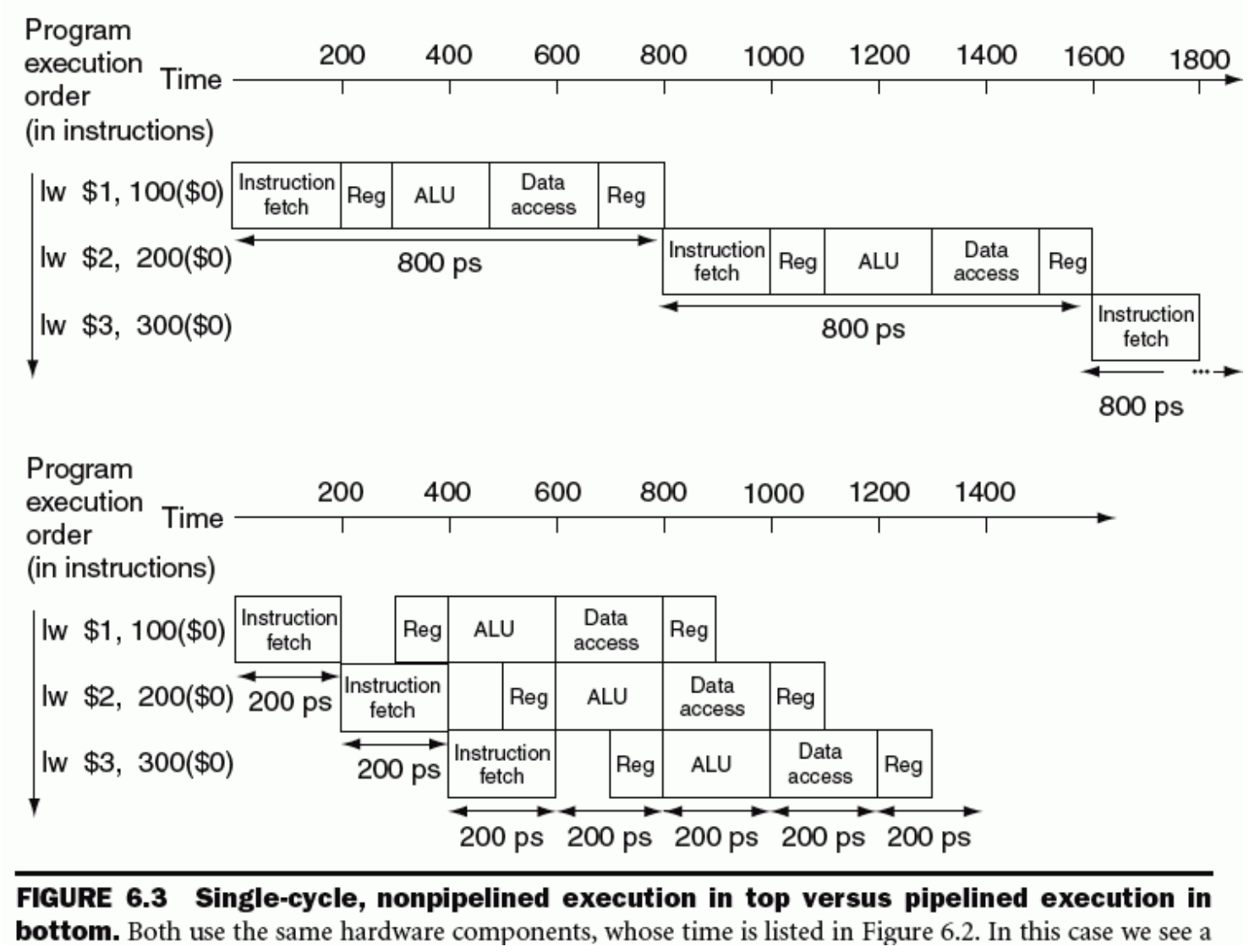


FIGURE 6.3 Single-cycle, nonpipelined execution in top versus pipelined execution in bottom. Both use the same hardware components, whose time is listed in Figure 6.2. In this case we see a

Visão Geral

- O tempo entre a primeira e quarta instrução no projeto sem pipeline é 3 x 800ps = 2400ps
- O tempo entre a primeira e a quarta instrução é de 3 x 200ps = 600ps
- Se os estágios forem perfeitamente balanceados, então o tempo entre as instruções no processador com pipeline – assumindo condições ideais – é igual a

Tempo entre instruções com pipeline = T. entre instruções sem pipeline / número de estágios do pipe

Sob condições ideais e com uma grande quantidade de instruções, o ganho de velocidade com a técnica pipeline é aproximadamente igual ao número de estágios do pipe.

Visão Geral

- Entretanto, o exemplo mostra que estágios podem ser mal balanceados.
- Assim o tempo por instrução no processador com pipeline será superior ao mínimo possível e o ganho de velocidade será menor que o número de estágios do pipeline
- Além do mais, até mesmo a afirmação de uma melhoria de 4 vezes para o exemplo anterior não está refletida no tempo de execução total. (1400ps versus 2400ps)
  - Isso acontece porque o número de instruções não é grande

Visão Geral

- O que aconteceria se aumentássemos o número de instruções para por exemplo 1.000.003?
  - O tempo de execução total seria 1.000.000 x 200ps + 1400ps = 200.001.400ps.
  - No exemplo sem pipeline seria 1.000.000 x 800ps + 2400ps = 800.002.400ps
  - Sob as condições acima, a razão entre os tempos de execução total para programas reais nos processadores sem e com pipeline é próximo à razão de tempos entre as instruções
- A técnica de pipeline melhora o desempenho aumentando a vazão de instruções, em vez de diminuir o tempo de execução de uma instrução individual.
- A vazão é importante, pois os programas reais executam bilhões de instruções

Projetando instruções para pipelining

- O projeto do conjunto de instruções do MIPS foi pensado para execução em pipeline.

- Todas as instruções têm o mesmo tamanho
  - Isso torna mais fácil buscar instruções no primeiro estágio do pipeline e decodificá-las no segundo estágio

- O MIPS tem apenas alguns poucos formatos de instrução, com os campos de registrador de origem localizados no mesmo lugar em cada instrução.
  - Essa simetria significa que o segundo estágio pode começar a ler o banco de registradores ao mesmo tempo em que o hardware está determinando que tipo de instrução foi lida.

- Os operandos em memória só aparecem em loads ou estores no MIPS
  - Isso significa que podemos usar o estágio de execução para calcular o endereço de memória e depois acessar a memória no estágio seguinte..

Pipeline Harzards

- São situações no pipelining em que a próxima instrução não pode ser executada no ciclo de clock seguinte.

1. Harzards estruturais

1. Harzards de dados

1. Harzards de controle

Harzards de estruturais – visão geral

- Significa que o hardware não pode admitir a combinação de instruções que queremos executar no mesmo ciclo de clock.

- No exemplo da lavanderia, aconteceria:
  - se usássemos uma combinação lavadora-secadora no lugar de lavadora e secadora separadas,
  - ou se nosso colega estivesse ocupado com alguma outra coisa e não pudesse guardar as roupas

- Sem duas memórias, nosso pipeline no MIPS teria um hazard estrutural caso tivéssemos uma quarta instrução no pipeline da figura 6.3

Harzard de dados – visão geral

- Ocorrem quando o pipeline precisa ser interrompido porque uma etapa precisa esperar até que outra seja concluída.

- Ele surge quando uma instrução depende de uma anterior que ainda está no pipeline.

- Ex. uma instrução add seguida imediatamente por uma instrução subtract que usa a soma ($s0)

```
Add $s0, $t0, $t1
Sub $t2, $s0, $t3
```

- A instrução add não escreve seu resultado até o quinto estágio, significando que teríamos de acrescentar 3 bolhas ao pipeline

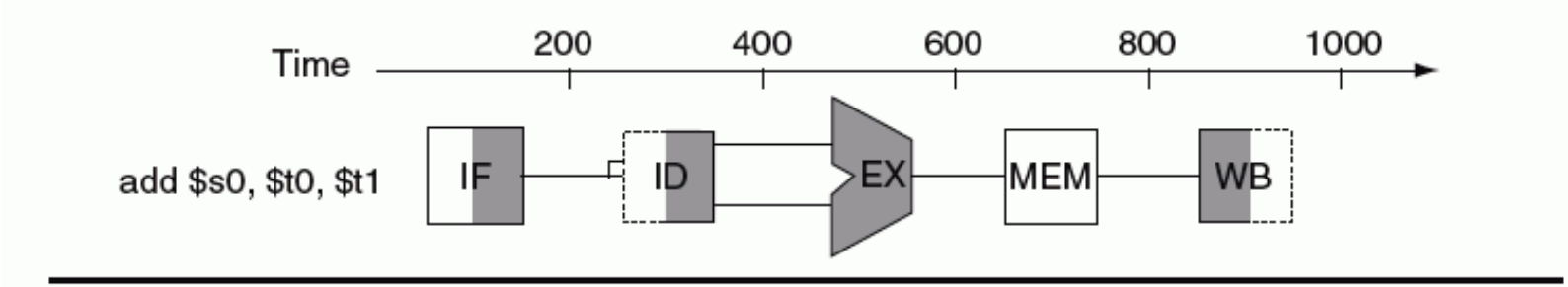
Harzard de dados – visão geral

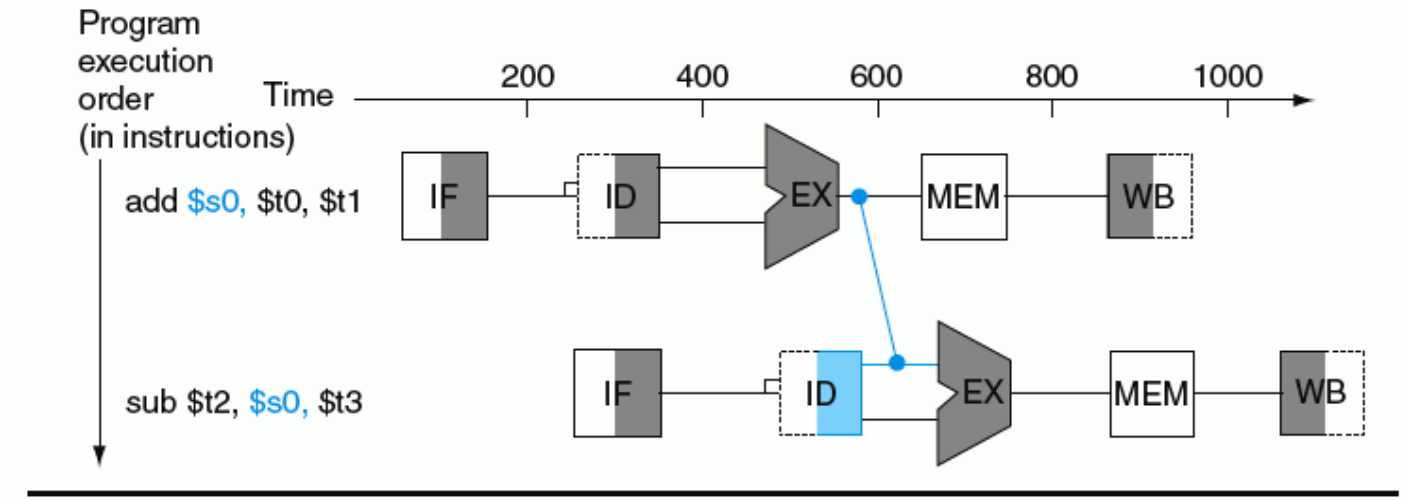
- Embora pudéssemos contar com compiladores para remover esses hazards, os resultados não seriam satisfatórios.

- A solução principal é baseada na observação de que não precisamos esperar que a instrução termine antes de tentar resolver o hazard de dados.

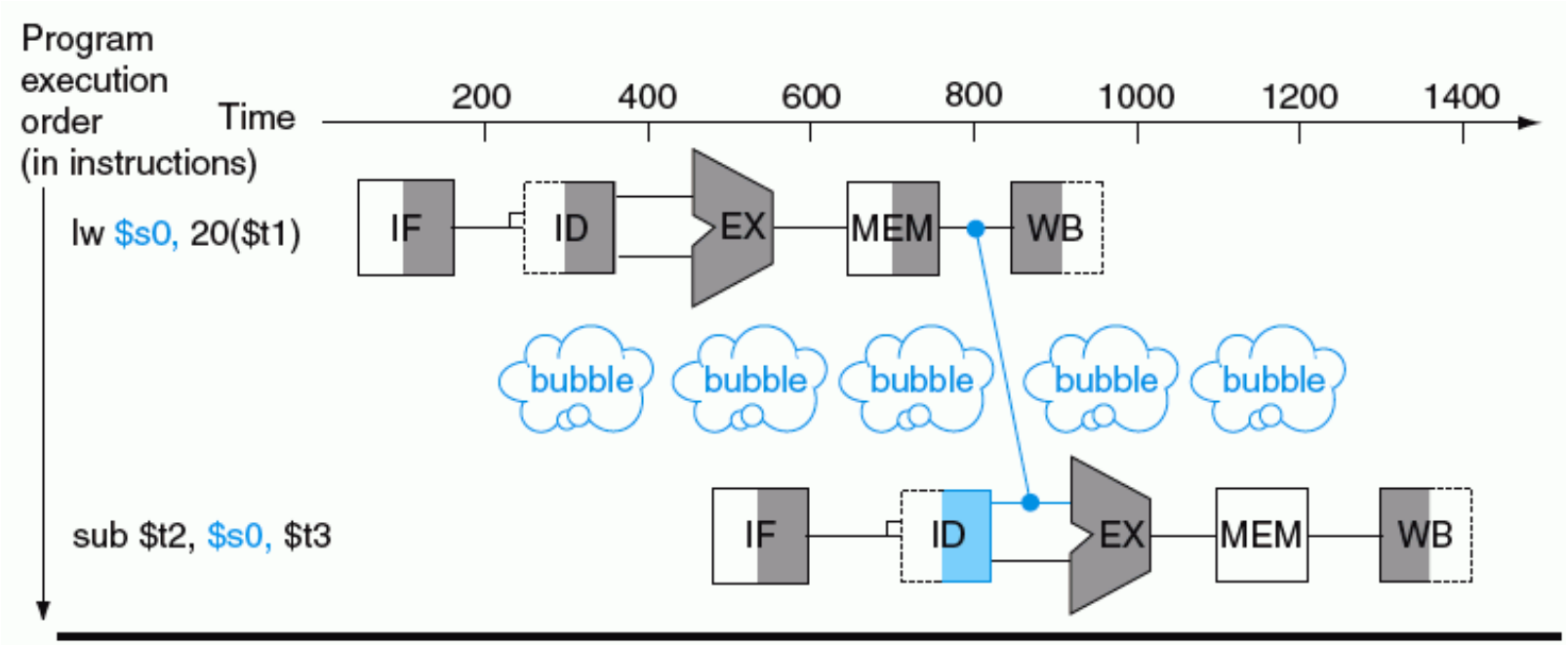
- Para a sequência de código anterior, assim que ALU cria a soma para o add, podemos fornecê-la como uma entrada para a subtração (FORWARDING ou BYPASSING ou ADIANTAMENTO de resultados)

Forwarding com duas instruções





Pipeline stalls – Bolhas



Reordenando o código para evitar pipeline stalls

Consider the following code segment in C:

```
A = B + E;
C = B + F;
```

Here is the generated MIPS code for this segment, assuming all variables are in memory and are addressable as offsets from `$t0`:

```
lw      $t1, 0($t0)
lw      $t2, 4($t0)
add     $t3, $t1,$t2
sw      $t3, 12($t0)
lw      $t4, 8($01)
add     $t5, $t1,$t4
sw      $t5, 16($t0)
```

Find the hazards in the following code segment and reorder the instructions to avoid any pipeline stalls.

Hazard de controle – visão geral

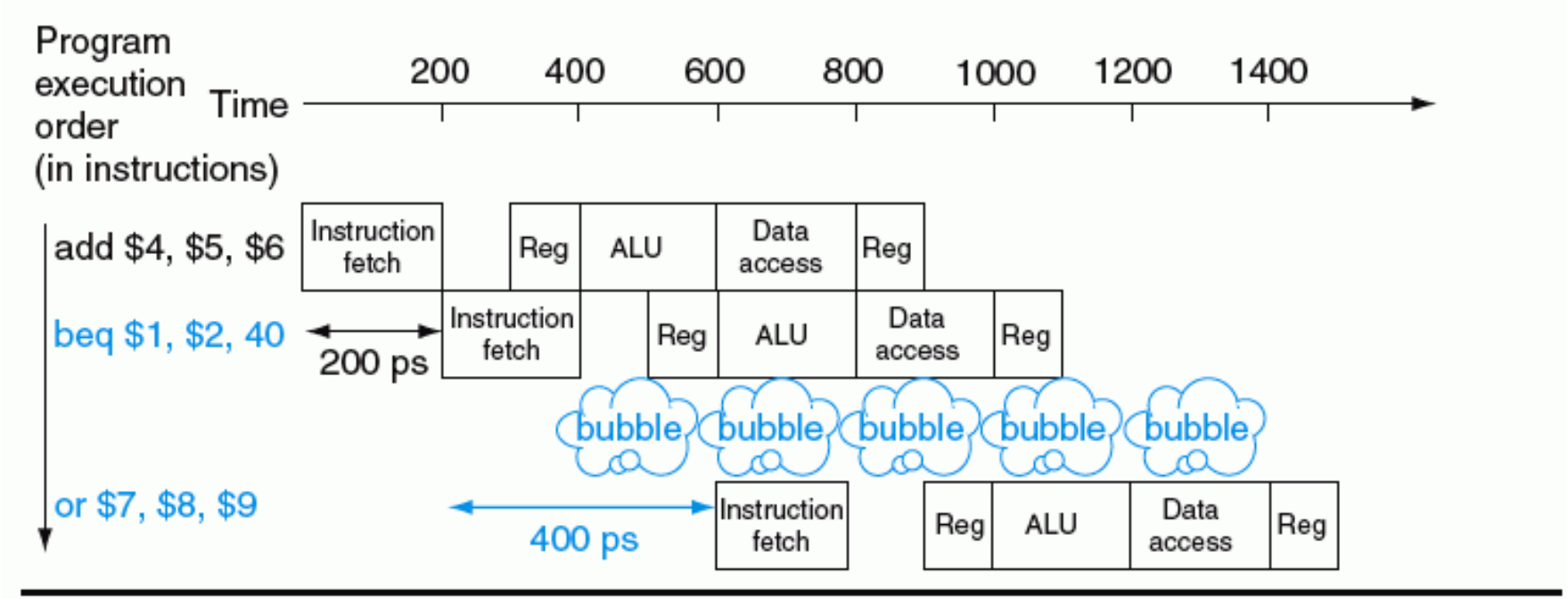
- Vem da necessidade de tomar uma decisão com base nos resultados de uma instrução enquanto outras estão sendo executadas.

- Também chamado de hazard de desvio

- Temos que buscar a instrução após o desvio no próximo ciclo de clock.
  - Contudo o pipeline possivelmente não saberá qual é a próxima instrução
  - Uma solução é ocasionar um stall no pipeline imediatamente após buscarmos um desvio e esperarmos até que o pipeline saiba o resultado do desvio para saber de que endereço apanhar a próxima instrução

Hazard de controle – visão geral

- Mesmo com hardware extra suficiente de modo que possamos testar registradores, calcular o endereço de desvio e atualizar o PC durante o segundo estágio do pipeline, teríamos a situação descrita abaixo



Desempenho do “stall no desvio”

- Exercício:
  - Estime o impacto nos ciclos de clock por instrução (CPI) do stall nos desvios. Suponha que todas as outras instruções tenha o CPI = 1 e que os desvios são 13% das instruções executadas.

- Se não pudermos resolver o desvio no segundo estágio, como normalmente acontece em pipelines maiores, então veríamos um atraso ainda maior.

- Segunda solução: Prever.
  - Uma técnica simples é sempre prever que os desvios não serão tomados.
  - Quando você estiver certo, o pipeline prosseguirá a toda velocidade
  - Somente quando os desvios são tomados é que o pipeline sofre um stall

Previsão de desvios

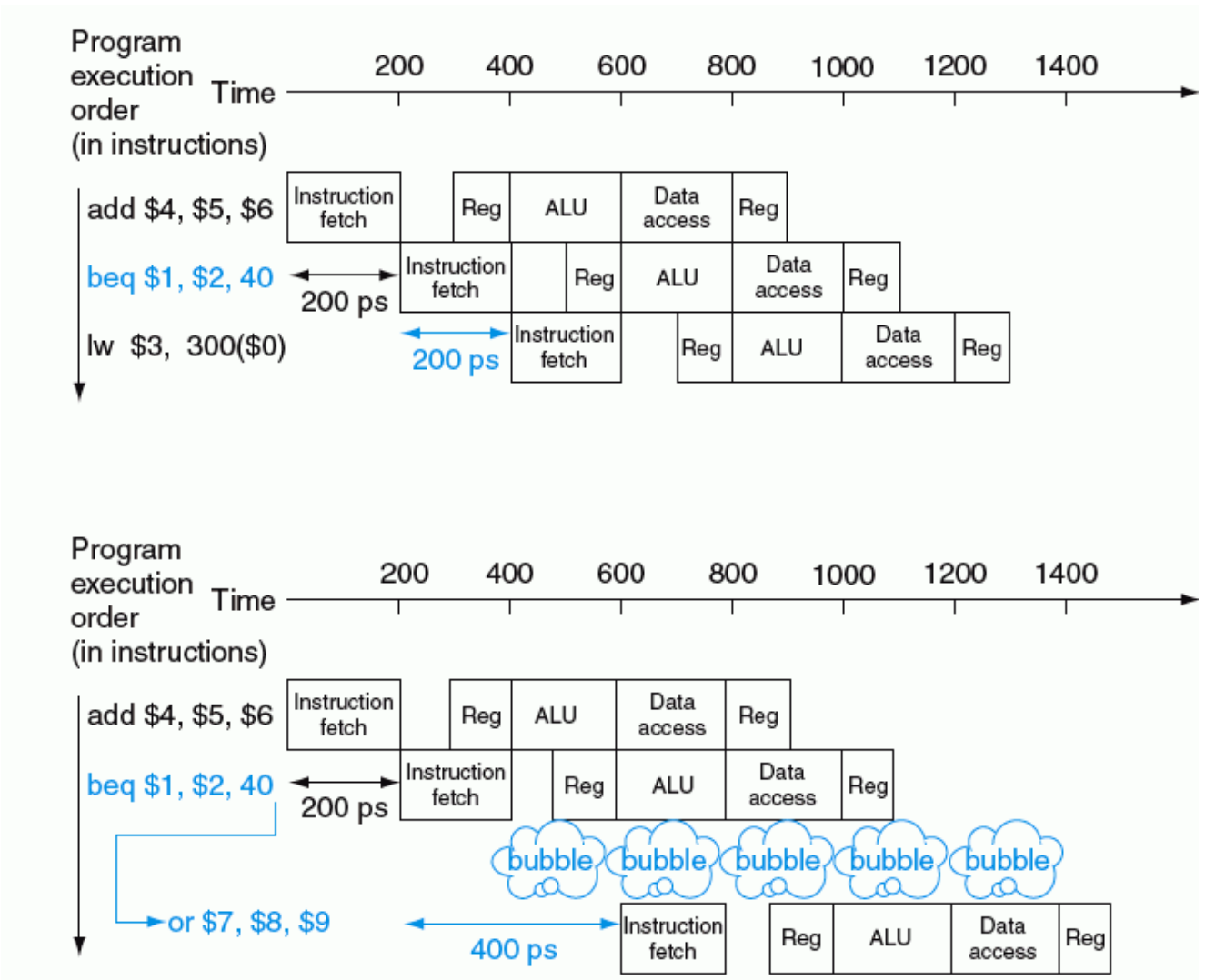


FIGURE 6.8 Predicting that branches are not taken as a solution to control hazard.

Previsão de desvios

- Uma versão mais sofistica é prever desvios para endereços anterioes como tomados (loops fazem isso).
- Outra opção é utilizar previsores de hardware dinâmicos que fazem sua escolha dependendo do comportamento anterior dos desvios.
- Em sistemas reais, previsores de desvio dinâmicos possuem uma precisão superior à 90%
- Quando a previsão estiver errado, a unidade de controle precisa garantir que a instrução buscada incorretamente, não tenha efeito, e reiniciar o pipeline no endereço de desvio apropriado.
- Pipelines mais longos aumentam o problema, aumentando o custo do erro de previsão

Previsão de desvios

- Terceira solução: Previsão adiada
  - Também chamado de ***delay branch*** nos computadores, sempre executa a próxima instrução sequencial, com o desvio ocorrendo após esse atraso de uma instrução.
  - O montador MIPS colocará uma instrução imediatamente após a instrução de desvio, que não é afetada pelo desvio.
  - Em nosso exemplo, a instrução add antes do desvio não afeta o desvio e pode ser movida para depois do desvio, para ocultar totalmente o atraso do desvio

Referências

- Hennessy, J. L., Patterson, D. A. *Organização e projeto de computadores: A Interface hardware/software*. 3 ed, Rio de Janeiro, LTC, 2005.