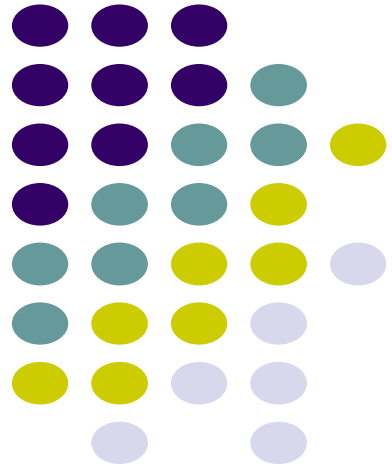
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

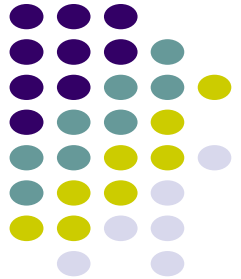
Arquitetura e Organização de Computadores

Melhorando o Desempenho com Pipeline - Parte I



Prof. Sílvio Fernandes

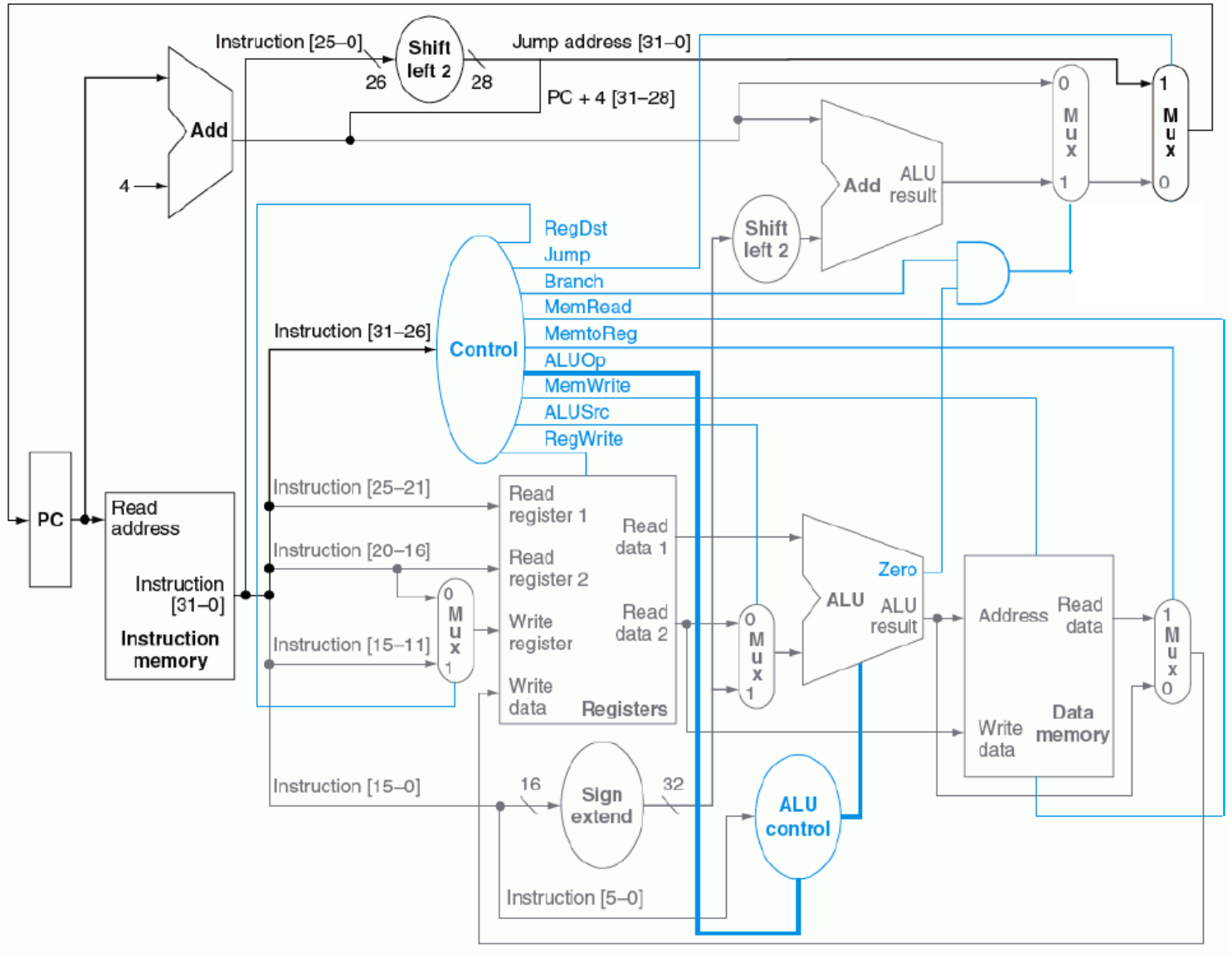
Por que uma implementação de ciclo único não é usada hoje?



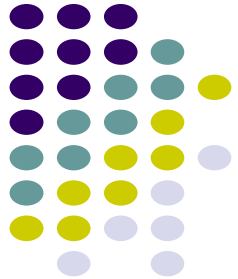
- Porque é ineficiente
  - O ciclo de clock precisa ter a mesma duração para cada instrução nesse projeto de ciclo único e o CPI será 1
  - É claro que o ciclo de clock é determinado pelo caminho mais longo possível na máquina
  - Esse caminho, é quase certamente, uma instrução load, que usa 5 unidades funcionais em série
  - O desempenho de uma implementação não será muito bom, já que várias das classes de instrução poderiam ficar em um ciclo de clock mais curto

Desempenho das máquinas de ciclo único

- Considerando o caminho de dados

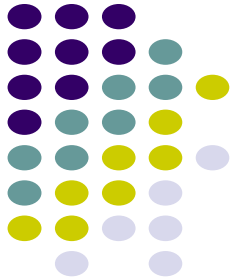


Desempenho das máquinas de ciclo único



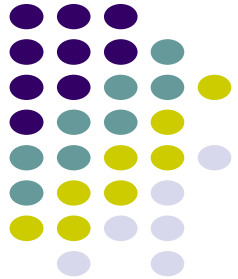
- Suponha que os tempos de operação para as principais unidades funcionais nessa implementação sejam os seguintes
  - Unidades de memória: 200 picossegundos (ps)
  - ALU e somadores: 100 ps
  - Banco de registradores (leitura ou escrita): 50 ps
- Considerando que os multiplexadores, a unidade de controle, os acessos do PC, a unidade de extensão de sinal e os fios não possuem atraso, qual das seguintes implementações seria mais rápida e por quanto?

Desempenho das máquinas de ciclo único



1. Uma implementação em que toda instrução opera em 1 ciclo de clock em uma duração fixa
2. Uma implementação em que toda instrução é executada em 1 ciclo de clock usando um clock de duração variável, que, para cada instrução, tem apenas a duração necessária. (**Esse método não é incrivelmente prático, mas permitirá ver o que está sendo sacrificado quando todas as instruções precisam ser executadas com um clock único de mesma duração**)

Desempenho das máquinas de ciclo único

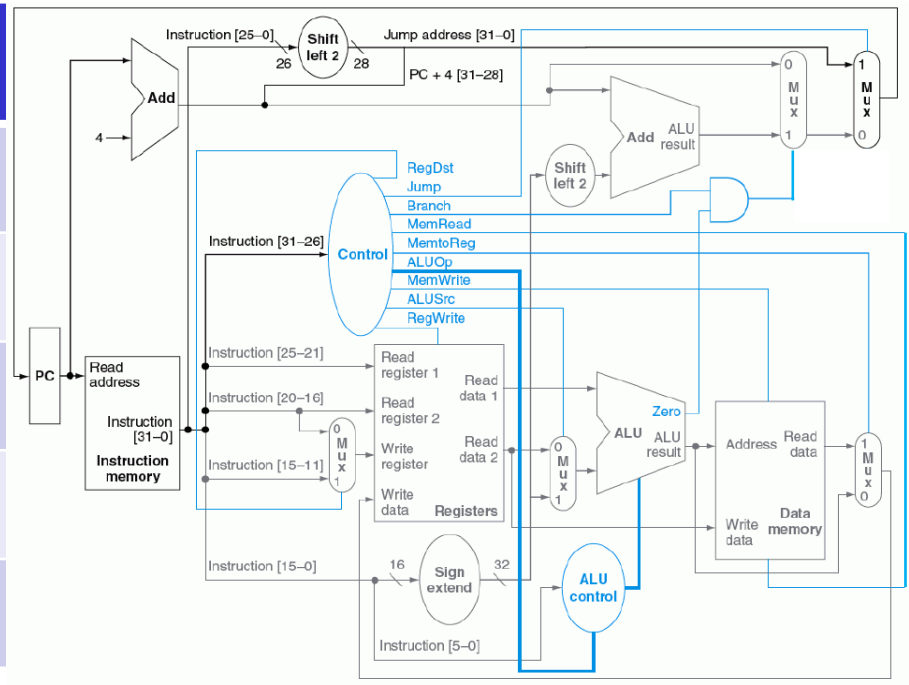


- Para comparar o desempenho, considere o seguinte mix de instruções: 25% loads, 10% stores, 45% instruções da ALU, 15% desvios e 5% jumps
- Vamos começar comparando os tempos de execução da CPU
  - Tempo CPU = Contagem instruções x CPI x Tempo do ciclo de clock
- Como CPI é 1, podemos simplificar
  - Tempo CPU = Contagem instruções x Tempo do ciclo de clock

Desempenho das máquinas de ciclo único

- Resposta:
  - Precisamos encontrar o tempo de ciclo de clock para as 2 implementações, já que a contagem de instruções e CPI são iguais

Classe de instrução	Unidades funcionais usadas pela classe de instrução				
Tipo R	Busca de instrução	Acesso a registrador	ALU	Acesso a registrador	
Load Word	Busca de instrução	Acesso a registrador	ALU	Acesso à memória	Acesso a registrador
Store Word	Busca de instrução	Acesso a registrador	ALU	Acesso à memória	
Branch	Busca de instrução	Acesso a registrador	ALU		
Jump	Busca de instrução				



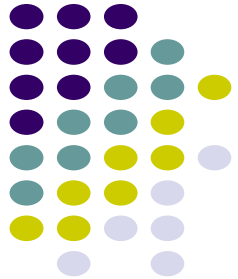
Desempenho das máquinas de ciclo único

- Resposta:
  - Usando esses caminhos críticos, podemos calcular o tamanho exigido para cada classe de instrução:

Classe de instrução	Memória de instrução	Leitura de registrador	Operação ALU	Memória de dados	Leitura de registrador	Total
Tipo R	200	50	100	0	50	400 ps
Load Word	200	50	100	200	50	600 ps
Store Word	200	50	100	200		550 ps
Branch	200	50	100	0		350 ps
Jump	200					200 ps

Para ciclo único essa é a duração

Desempenho das máquinas de ciclo único



- Resposta:
  - Já uma máquina com um clock variável terá um ciclo de clock que varia entre 200ps e 600ps.
  - Podemos encontrar a duração média do ciclo de clock usando essas informações e a distribuição da frequência das instruções
  - **Ciclo de clock da CPU** = 600 x 25% + 550 x 10% + 400 x 45% + 350 x 15% + 200 x 5% = 447,5 ps

Desempenho das máquinas de ciclo único

- Resposta:

$$\frac{Desempenho_{clock\ variável}}{Desemepnho_{clock\ único}} = \frac{Tempo\ de\ Execuçãoclock_{\ único}}{Tempo\ de\ Execuçãoclock_{\ variável}} =$$

$$\frac{CI\ x\ Ciclo\ de\ clock\ da\ CPUclocl_{\ variável}}{CI\ x\ Ciclo\ de\ clock\ da\ CPUclock_{\ único}} = \frac{Ciclo\ de\ clock\ da\ CPU_{único}}{Ciclo\ de\ clock\ da\ CPUvari_{ável}}$$

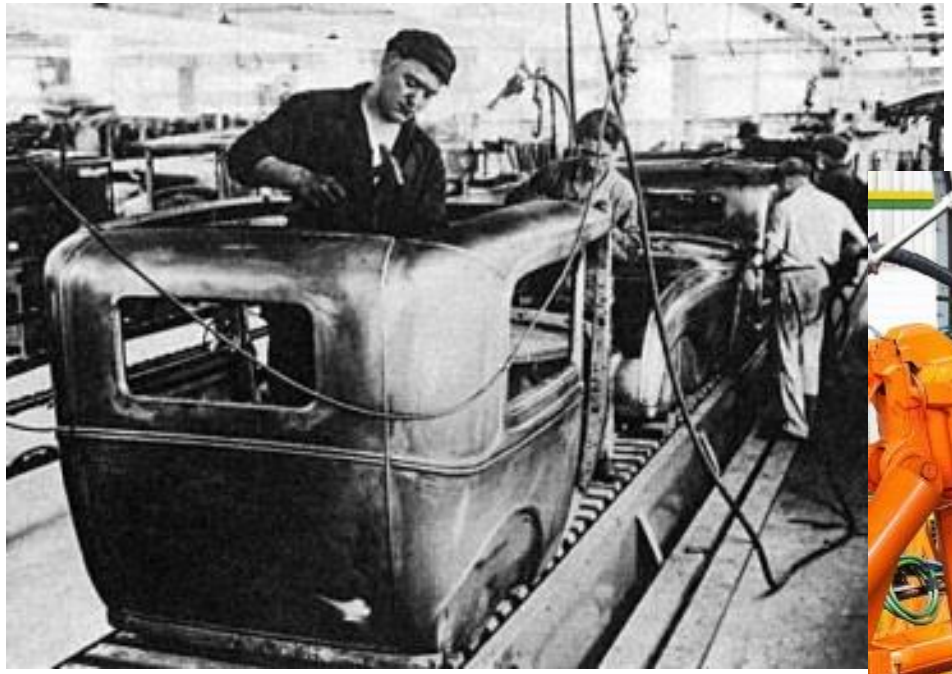
$$= \frac{600}{447{,}5} = 1{,}34$$

Infelizmente o ciclo variável não é viável

Visão geral de pipelining

Imagens do Filme “Tempos Modernos”

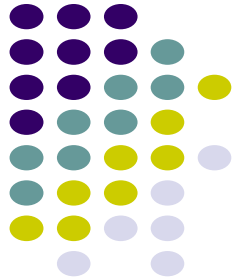
Visão geral de pipelining



Linha de Montagem da Ford de 1913

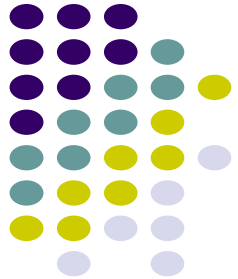
Linha de Montagem da Ford de 2013

Visão geral de pipelining



- **Pipelining** é uma técnica de implementação em que várias instruções são sobrepostas na execução.
- Qualquer um que tenha lavado muitas roupas intuitivamente já usou pipelining
- SEM pipelining para lavar roupas seria:
  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
- Quando seu colega terminar, comece novamente com a próxima trouxa suja

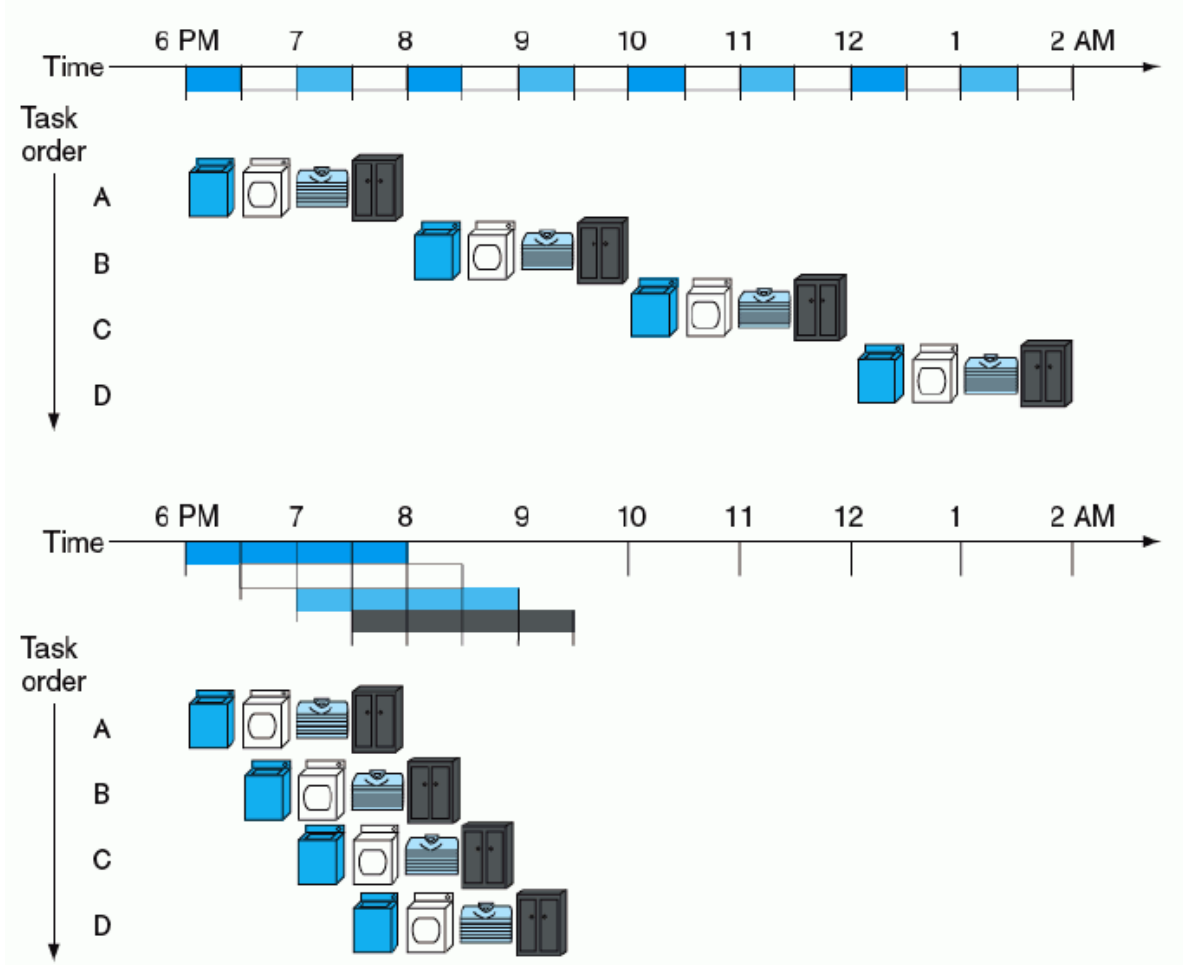
Visão geral de pipelining



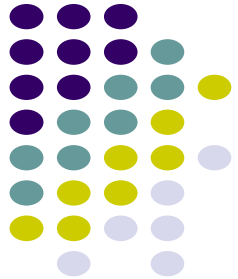
- A técnica COM pipeline leva muito menos tempo
  - Assim que a lavadora terminar com a 1ª trouxa e ela for colocada na secadora, você carrega a lavadora com a 2ª trouxa suja
  - Quando a 1ª trouxa estiver seca, você coloca na tábua para começar a passar e dobrar, move a trouxa molhada para a secadora e a próxima trouxa suja para a lavadora
  - Em seguida, você pede a seu colega para guardar a 1ª remessa, começa a passar e dobrar a 2ª, a secadora está com a 3ª remessa e você coloca a 4ª na lavadora

Visão geral de pipelining

- A técnica COM pipeline leva muito menos tempo

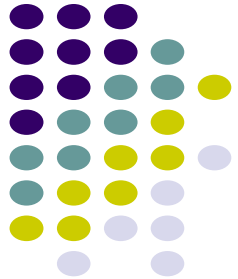


Visão geral de pipelining



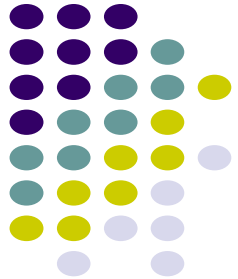
- A técnica de pipeline **não** diminuirá o tempo para concluir **uma** trouxa de roupas, mas, quando temos **muitas** trouxas para lavar, a melhoria na **vazão diminui** o tempo total para concluir o trabalho
- Se todos os estágios levarem aproximadamente o mesmo tempo e houver trabalho suficiente para realizar, então o ganho de velocidade devido à técnica de pipelining será igual ao número de estágios do pipeline

Visão geral de pipelining



- Os mesmos princípios se aplicam a processadores
- As instruções MIPS normalmente exigem 5 estágios
  1. Buscar instrução da memória (IF)
  2. Ler registradores enquanto a instrução é decodificada (ID)
  3. Executar a operação ou calcular um endereço (EX)
  4. Acessar um operando na memória de dados (MEM)
  5. Escrever o resultado em um registrador (WB)

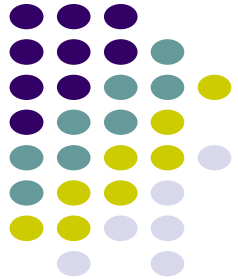
Desempenho de ciclo único *versus* desempenho com pipeline



- EXEMPLO
  - Os tempos de operação para as principais unidades funcionais neste exemplo são de:
    - 200 ps para acesso à memória
    - 200 ps para operação com ALU
    - 100 ps para leitura e escrita de registradores

Classe de instrução	Busca de instruções	Leitura de registradores	Operação da ALU	Acesso a dados	Escrita de registradores	Tempo Total
Load word	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word	200 ps	100 ps	200 ps	200 ps		700 ps
Formato R	200 ps	100 ps	200 ps		100 ps	600 ps
Branch	200 ps	100 ps	200 ps			500 ps

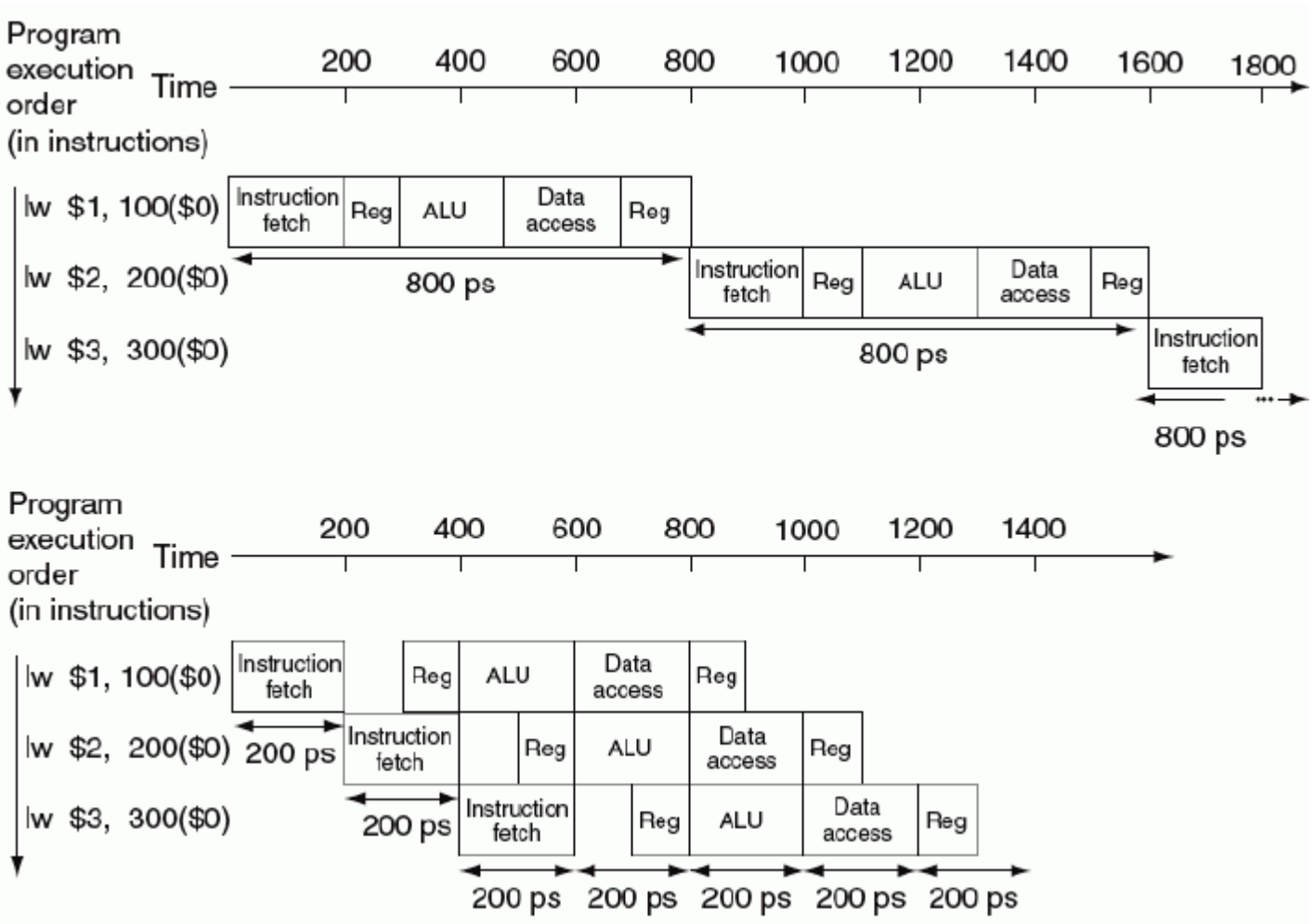
Desempenho de ciclo único *versus* desempenho com pipeline



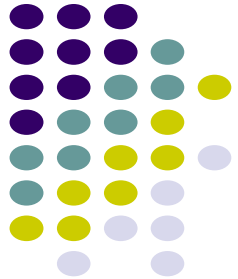
- EXEMPLO
  - O projeto de ciclo único precisa contemplar a instrução mais lenta (lw) de modo que o tempo exigido para cada instrução é 800 ps
    - O tempo entre a 1ª e 4ª instrução no projeto sem pipeline é 3 x 800 ps = 2400 ps
  - No projeto COM pipeline o ciclo de clock da execução precisa ter o ciclo no pior caso (200 ps), embora alguns estágios levem apenas 100 ps
    - O uso de pipeline oferece uma melhoria de 4 vezes: o tempo entra a 1ª e 4ª instrução é de 3 x 200 ps = 600 ps

Desempenho de ciclo único *versus* desempenho com pipeline

- EXEMPLO



Desempenho de ciclo único *versus* desempenho com pipeline

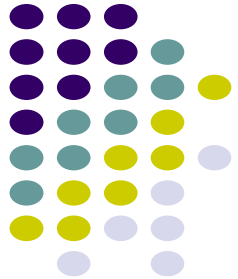


- O ganho de velocidade com o pipelining

$$Tempo\ entre\ instruçõescom_{\ pipeline} = \frac{Tempo\ entre\ instruçõessem_{\ pipeline}}{Número\ de\ estágio\ do\ pipe}$$

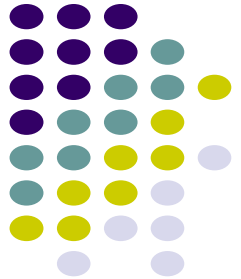
- Sob condições ideais e com uma grande quantidade de instruções, o ganho de velocidade com a técnica de pipelining é aproximadamente igual ao número do pipe; um pipeline de 5 estágios é quase 5 vezes mais rápido

Projetando conjuntos de instruções para pipelining



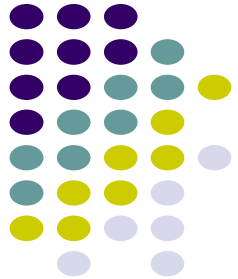
- Podemos entender melhor o projeto do conjunto de instruções MIPS, projetado para execução com pipeline
  1. Todas as instruções MIPS têm o mesmo tamanho, tornando mais fácil buscá-las no 1º estágio e decodificá-las no 2º
  2. O MIPS tem apenas alguns formatos de instrução, com os campos de registrador origem localizados no mesmo lugar; essa simetria significa que o 2º estágio pode começar a ler o banco de reg. ao mesmo tempo que o HW está determinando o tipo de instrução lida

Projetando conjuntos de instruções para pipelining



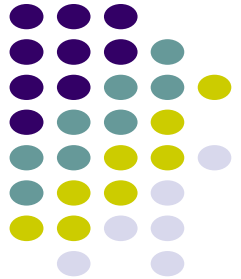
- Podemos entender melhor o projeto do conjunto de instruções MIPS, projetado para execução com pipeline
  3. Os operandos em memória só aparecem em loads ou stores no MIPS; podemos usar o estágio de execução para calcular o endereço de memória e depois acessar a memória no estágio seguinte
  4. Os operandos precisam estar aninhados na memória; não precisamos nos preocupar com uma única instrução de transferência de dados exigindo dois acessos à memória de dados.

Pipeline *harzards*



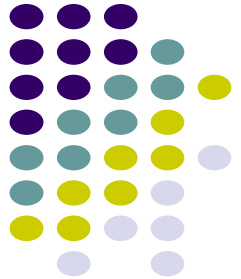
- Existem situações em pipelining em que a próxima instrução não pode ser executada no ciclo seguinte
- Esses eventos são chamados *harzards*, e existem 3 tipos diferentes
  - Harzards estruturais
  - Harzards de dados
  - Harzards de controle

Pipeline *harzards*



- Harzards estruturais
  - Significa que o HW não pode admitir a combinação de instruções que queremos executar no mesmo ciclo
    - Múltiplos acessos em uma única memória simultaneamente
    - Uso simultâneo da ALU

Pipeline *harzards*



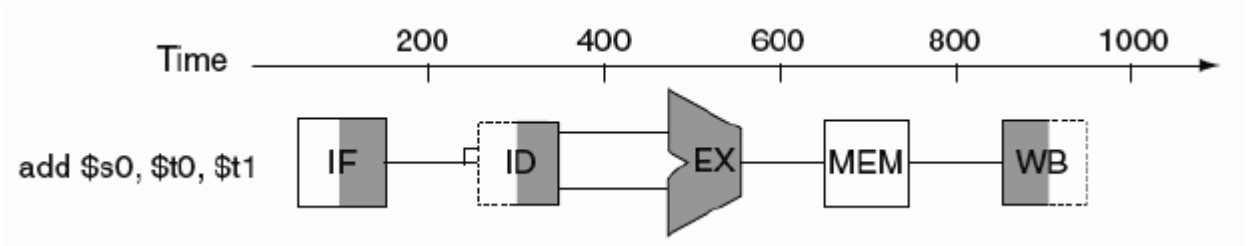
- Harzards de dados
  - Ocorrem quando o pipeline precisa ser interrompido porque uma etapa precisa esperar que os dados necessário estejam disponíveis, os quais são resultado de uma instrução em execução

    add $s0, $t0, $t1

    sub $t2, $s0, $t3
    - A solução principal é baseada na observação de que não precisamos esperar até que a instrução termine antes de tentar resolver o harzard de dados
    - O acréscimo de HW extra para ter o item antes do previsto, diretamente dos recursos internos, é chamado de **forwarding** ou **bypassing**

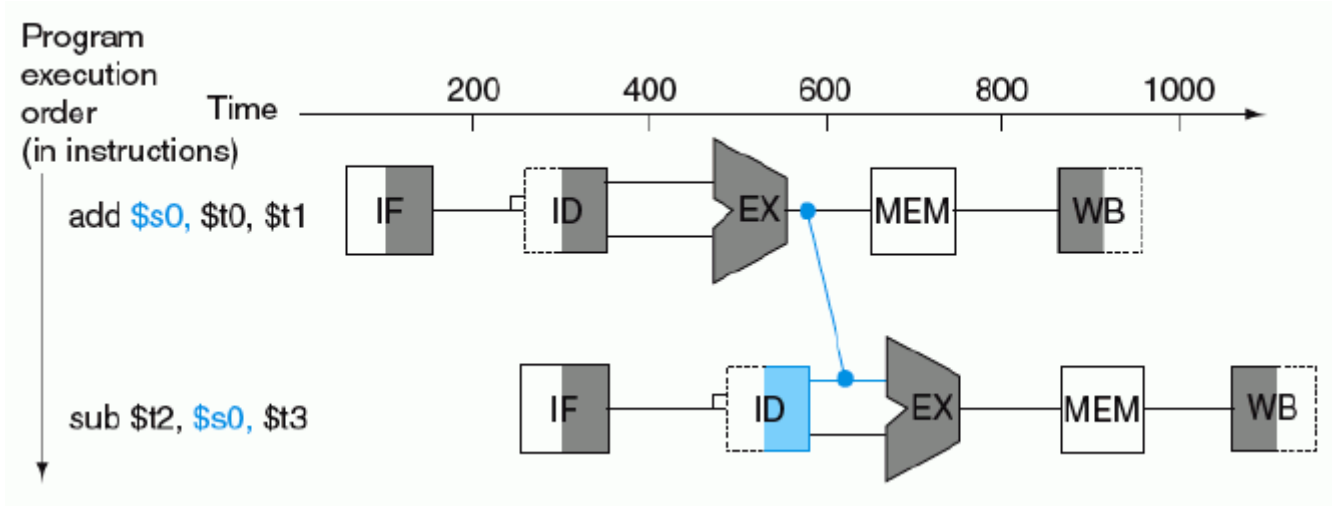
Pipeline *harzards*

- Harzards de dados
  - EXEMPLO: Para as 2 instruções anteriores, mostre graficamente quais estágios do pipeline estariam conectados pelo fowarding, considerando os 5 estágios de pipeline



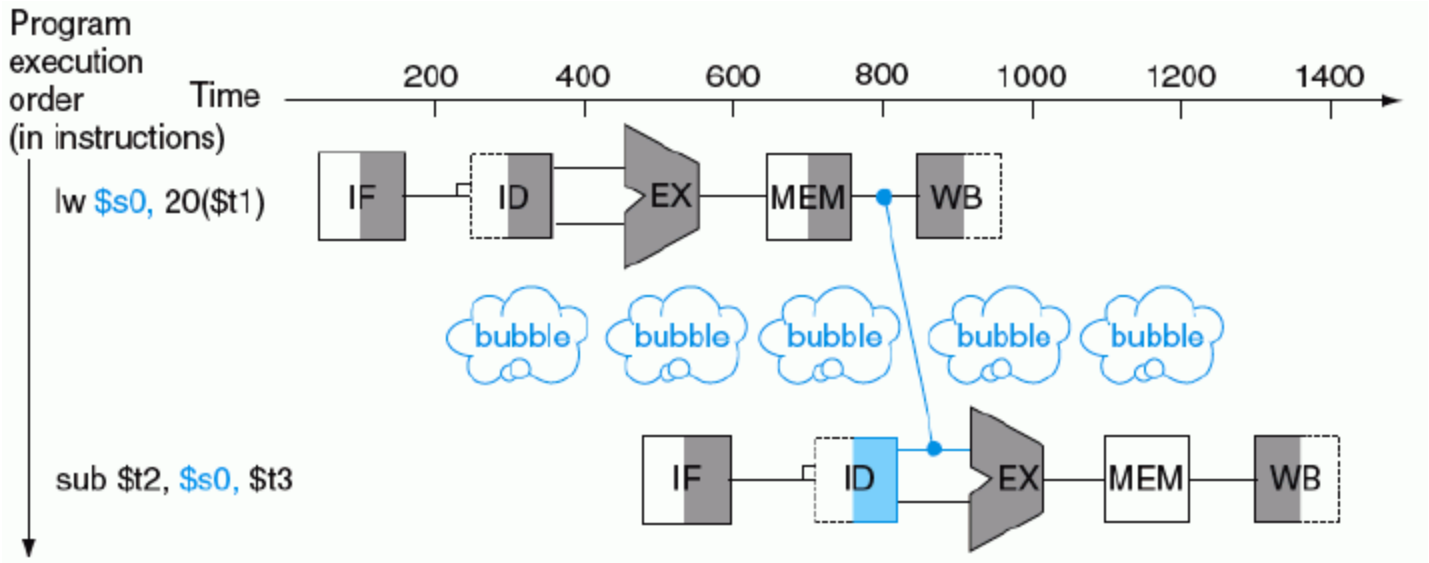
Pipeline *harzards*

- Harzards de dados
  - EXEMPLO: Resposta

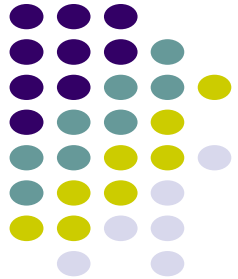


Pipeline *harzards*

- Harzards de dados
  - Essa solução não impede todos os **stalls** (também conhecido como **bolha**)



Pipeline *harzards*



- Harzards de dados
  - EXEMPLO: Considere o seguinte segmento de código em C:

    A = B + E;

    C = B+ F;

    Aqui está o código MIPS, supondo que todas as variáveis estejam com offset a partir de $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)
```

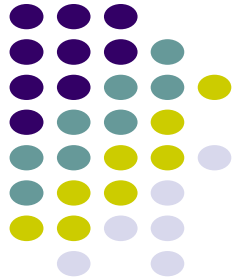
    Encontre os harzards no segmento de código e reordene as instruções para evitar quaisquer pipeline stalls.

Pipeline *harzards*

- Harzards de dados
  - EXEMPLO: Resposta

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

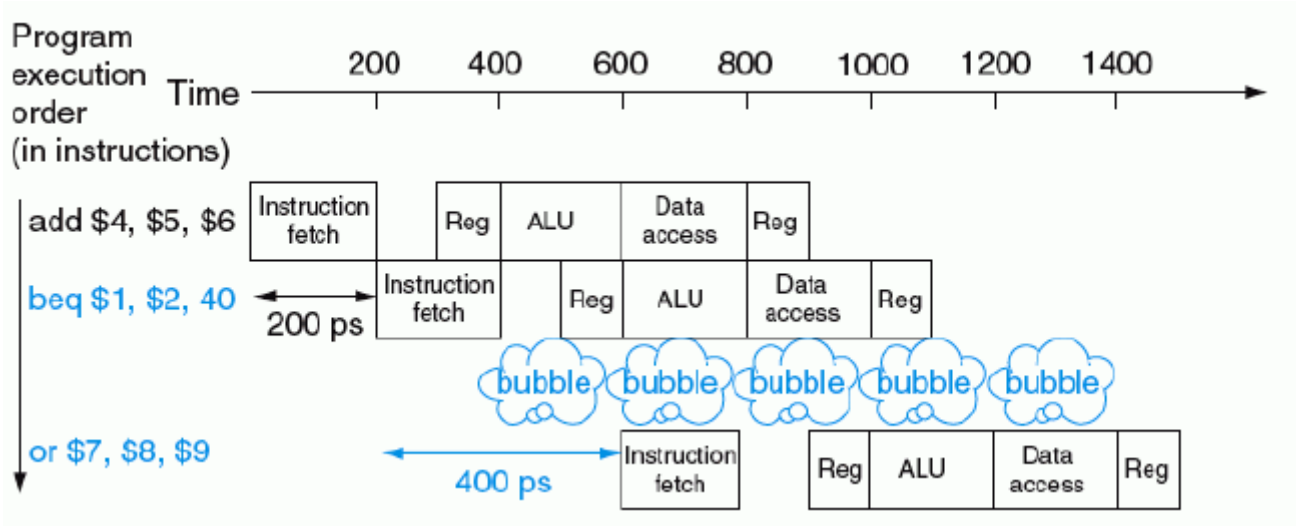
Pipeline *harzards*



- Harzards de controle
  - Vindo da necessidade de tomar uma decisão com base nos resultados de uma instrução enquanto outras estão sendo executadas
  - Observe que, em instruções de desvio, temos de começar a buscar a instrução após o desvio no próximo ciclo. Contudo, o pipeline possivelmente não saberá qual deve ser a próxima instrução, pois ele só *recebeu* da memória a instrução de desvio!
  - Considere que uma instrução lw é executada se o desvio não for tomado:

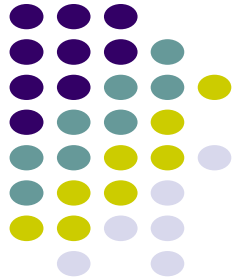
Pipeline *harzards*

- Harzards de controle



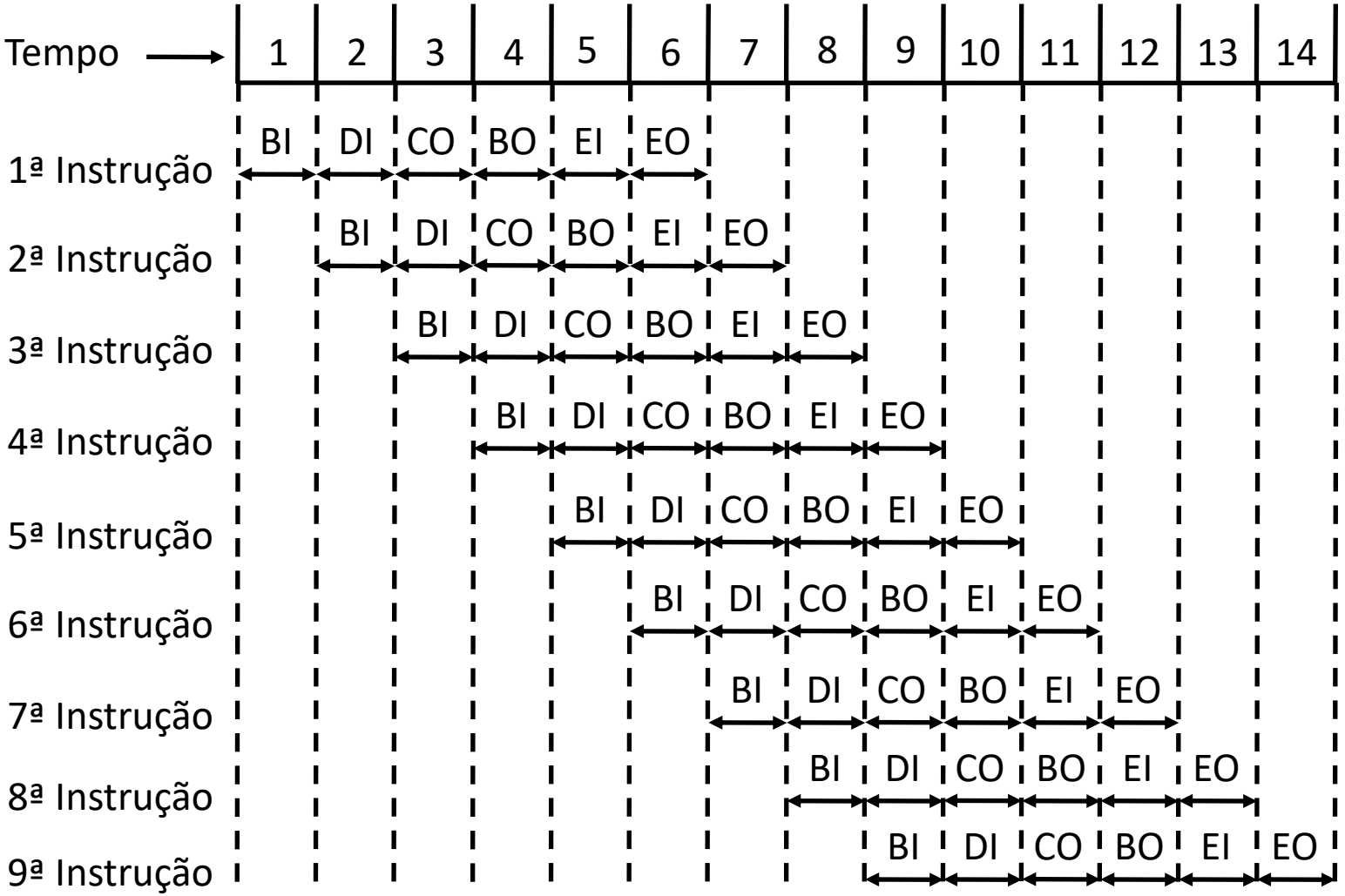
- Se não pudermos resolver o desvio no 2º estágio, então veríamos um atraso ainda maior se ocorresse um stall nos desvio
- A 2ª solução é prever

Pipeline de Instruções de Seis Estágios

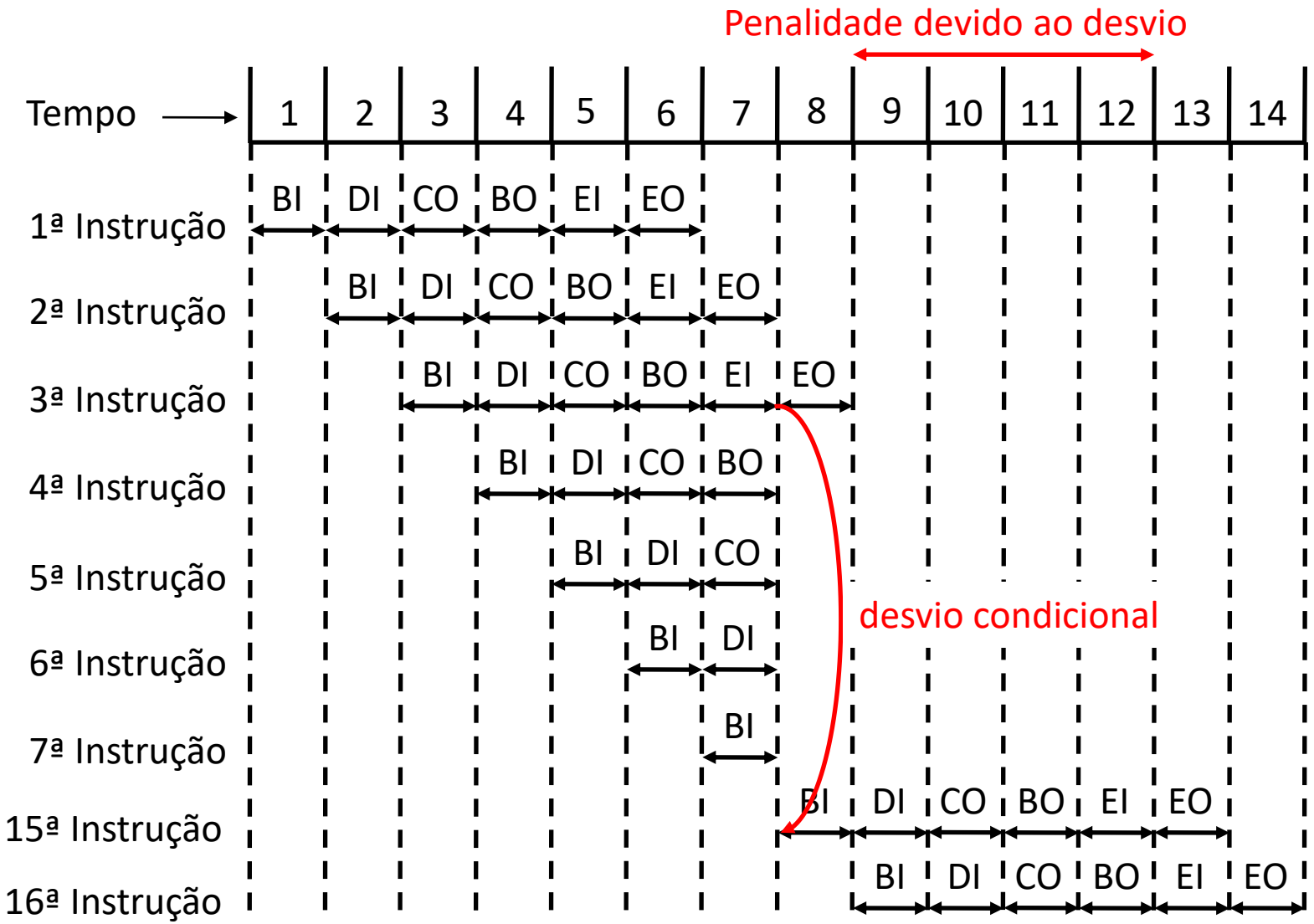


- Considerando um outro pipeline (6 estágios), podemos ver o problema do desvio
- Decomposição do processamento:
  - BI (Busca da Instrução)
  - DI (Decodificação da Instrução)
  - CO (Cálculo dos Operandos)
  - BO (Busca dos Operandos)
  - EI (Execução da Instrução)
  - EO (Escrita dos Operandos)

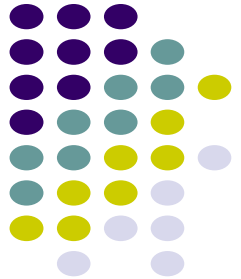
Pipeline de Instruções de Seis Estágios



Pipeline de Instruções de Seis Estágios



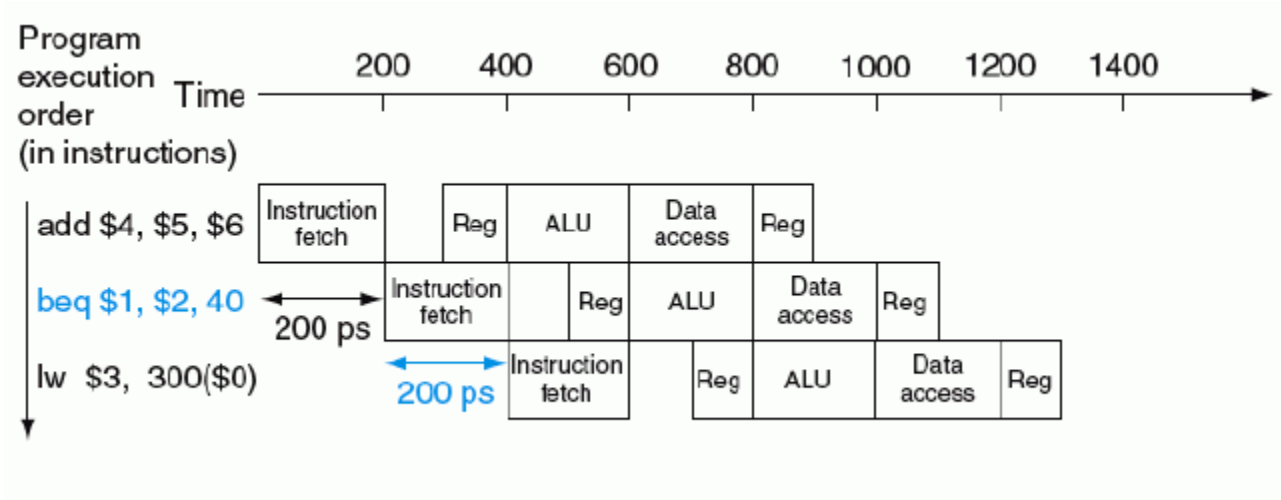
Pipeline *harzards*



- Harzards de controle
  - Os computadores realmente utilizam a *previsão* para tratar dos desvios
  - Uma técnica simples é prever que os **desvios nãos serão tomados**
    - Quando você estiver certo, o pipeline prosseguirá a toda velocidade
    - Somente quando os desvios são tomados é que o pipeline sobre um stall

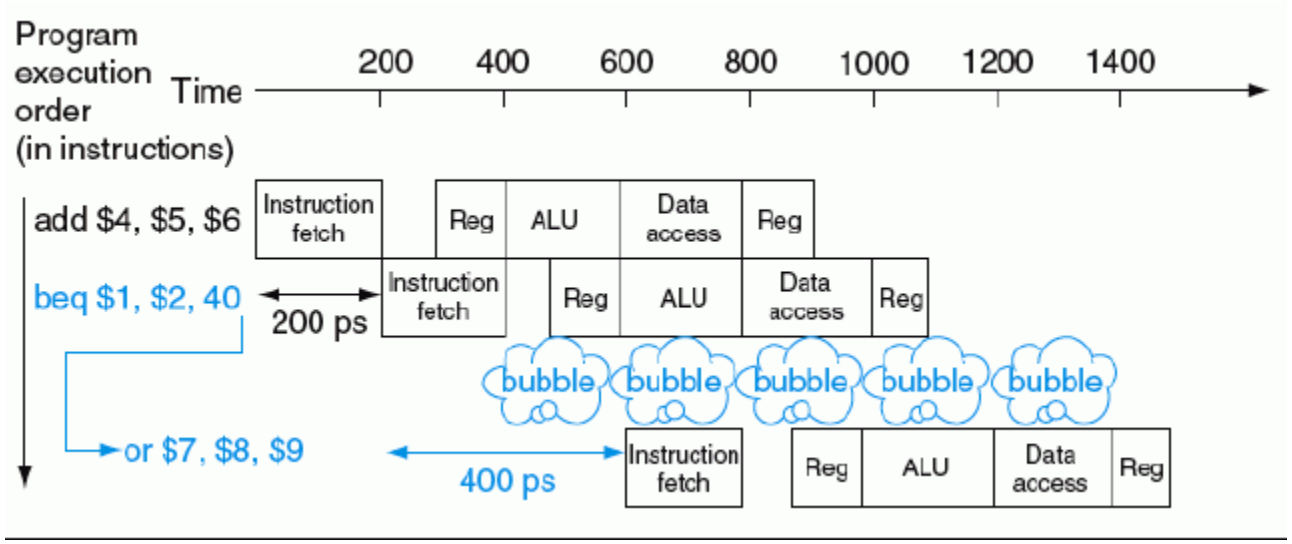
Pipeline *harzards*

- Harzards de controle
  - Previsão de desvios não tomados
    - Com desvio não tomado

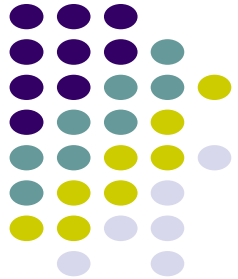


Pipeline *harzards*

- Harzards de controle
  - Previsão de desvios não tomados
    - Com desvio tomado



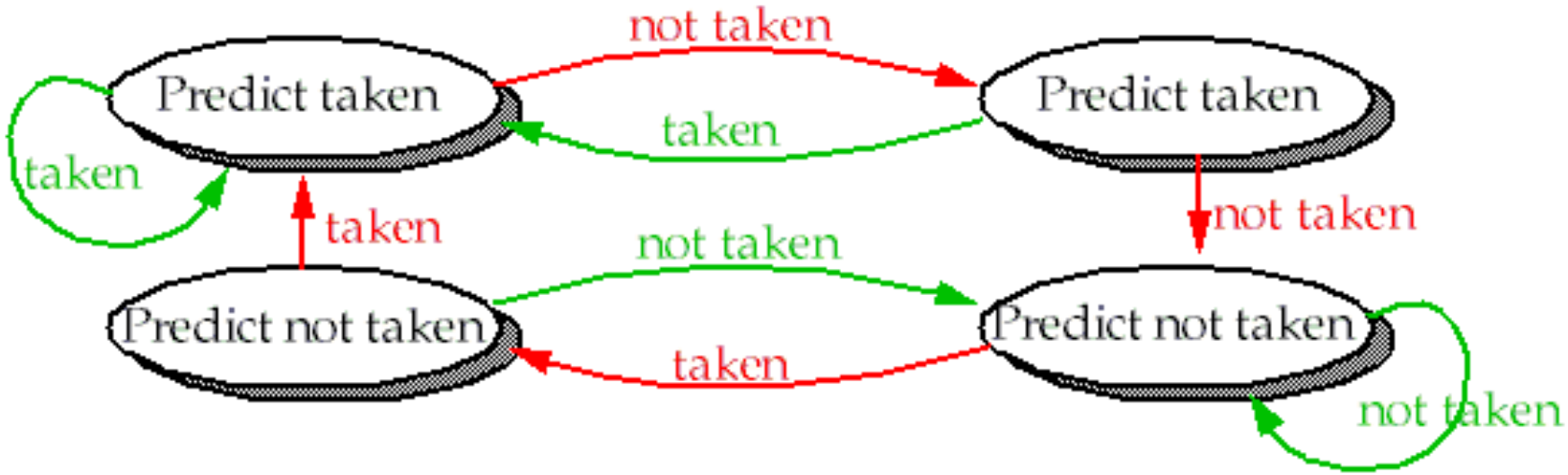
Pipeline *harzards*



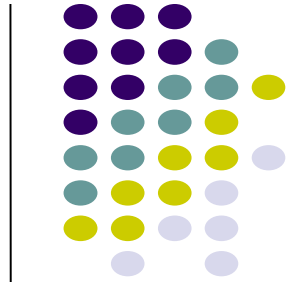
- Harzards de controle
  - Uma versão mais sofisticada da **previsão de desvio** teria alguns desvios previstos como tomados e alguns como não tomados
  - Essas técnicas rígidas para desvio contam com o comportamento estereotipado e não são responsáveis pela individualidade de uma instrução de desvio específica
  - Previsores de HW *dinâmico*, ao contrário, fazem suas escolhas dependendo do comportamento de cada desvio, e podem mudar as previsões para um desvio durante a vida do programa

Pipeline *harzards*

- Harzards de controle
  - Preditor de salto dinâmico

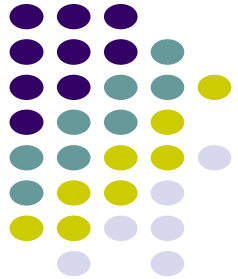


Pipeline *harzards*



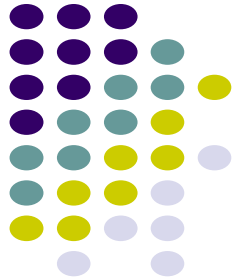
- Harzards de controle
  - Uma técnica popular para previsão dinâmica de desvio é manter um histórico de cada desvio como tomado ou não tomado, e depois usar o comportamento passado recente para prever o futuro
    - Essa técnica pode prever corretamente, com uma precisão superior a 90%

Pipeline



- A técnica de pipelining melhora a *vazão* de instruções, e não o *tempo de execução* ou *latência* das instruções individualmente

Verifique você mesmo

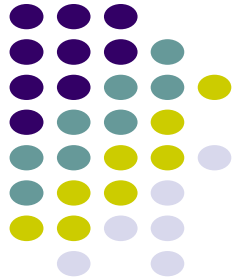


- Para cada sequência de código a seguir, indique se ela deverá sofrer stall, pode evitar stalls usando apenas forwarding, ou pode ser executada sem stall ou forwarding:

Sequence 1	Sequence 2	Sequence 3

lw $t0,0($t0)
add $t1,$t0,$t0
	add $t1,$t0,$t0
	addi $t2,$t0,#5
	addi $t4,$t1,#5
		addi $t1,$t0,#1
		addi $t2,$t0,#2
		addi $t3,$t0,#2
		addi $t3,$t0,#4
		addi $t5,$t0,#5

Verifique você mesmo



- Resposta
  1. Stall no resultado de LW
  2. Fazer forwarding com o resultado de ADD
  3. Não é necesário forwarding nem stall

Referências

- PATTERSON, D. A. ; HENNESSY, J.L. Organização e projeto de computadores – a interface hardware software. 3. ed. Editora Campus, 2005.
- STALLINGS, W. Arquitetura e organização de computadores: projeto para o desempenho. 8. ed. Prentice Hall, 2009.