ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

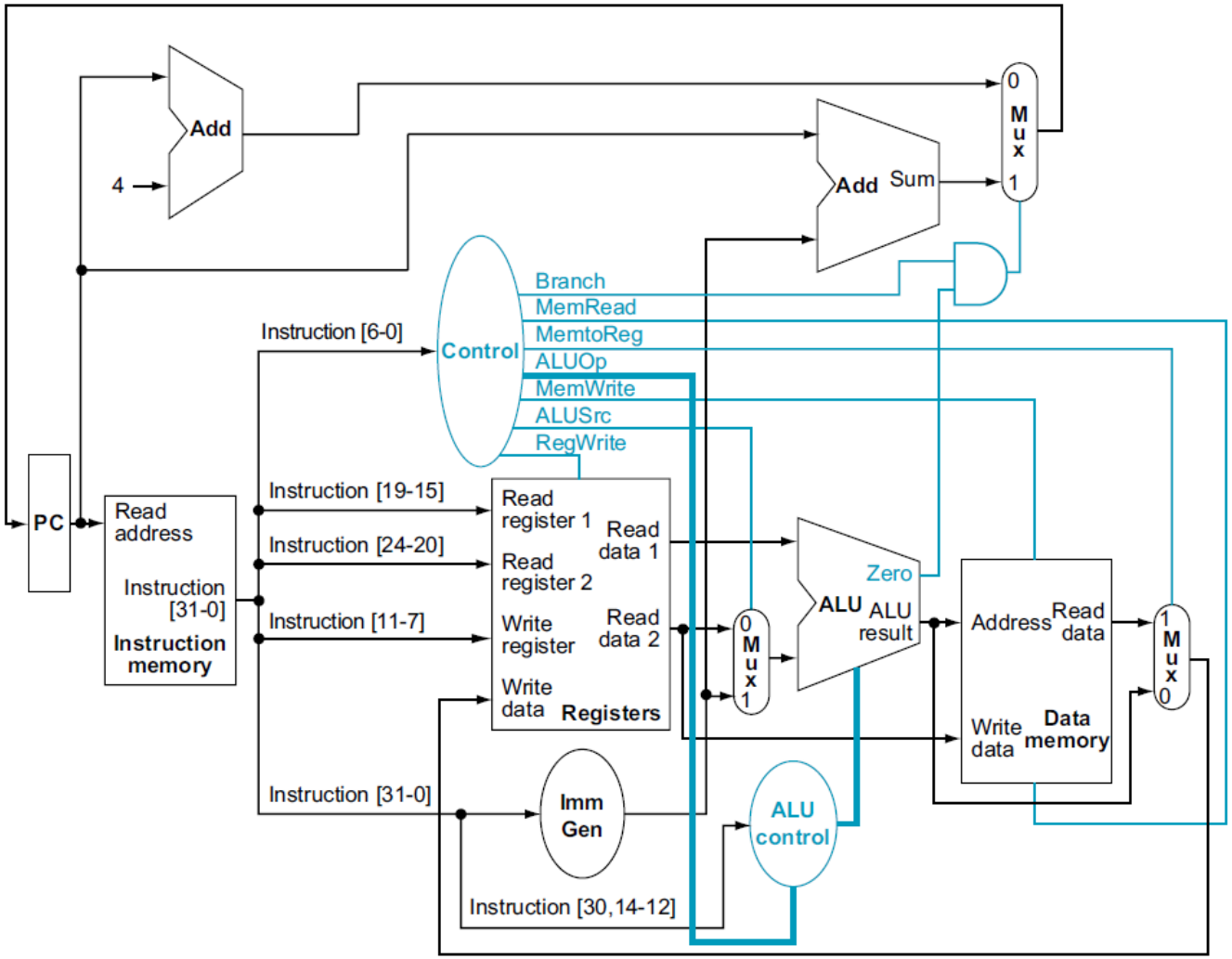
Arquitetura RISC-V **Pipelined**

Profª. Fabiana F F Peres

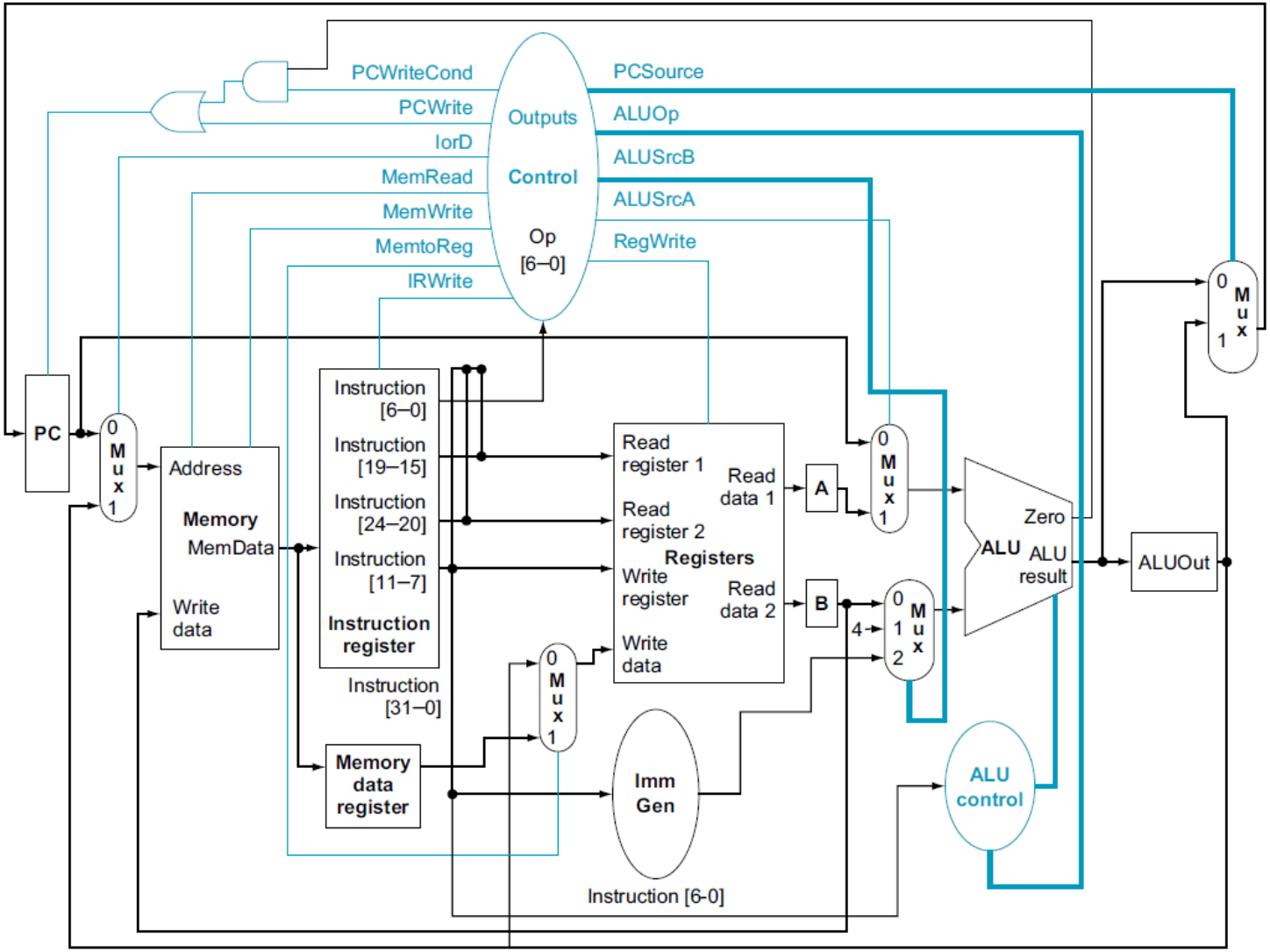
Apoio: Camile Bordini

Monociclo (revisão)

- Executa toda instrução em um ciclo de clock;



Multiciclo (revisão)



Multiciclo (revisão)

(características)

•Cada um dos passos, tomará um ciclo de clock;

•Permite que as unidades funcionais possam ser usadas mais de uma vez pela mesma instrução contanto que seja em ciclos de clock diferentes;

•Reduz o hardware;

•Permite que as instruções tomem diferentes números de ciclos de clock;

Multiciclo (revisão)

(passos para execução das instruções)

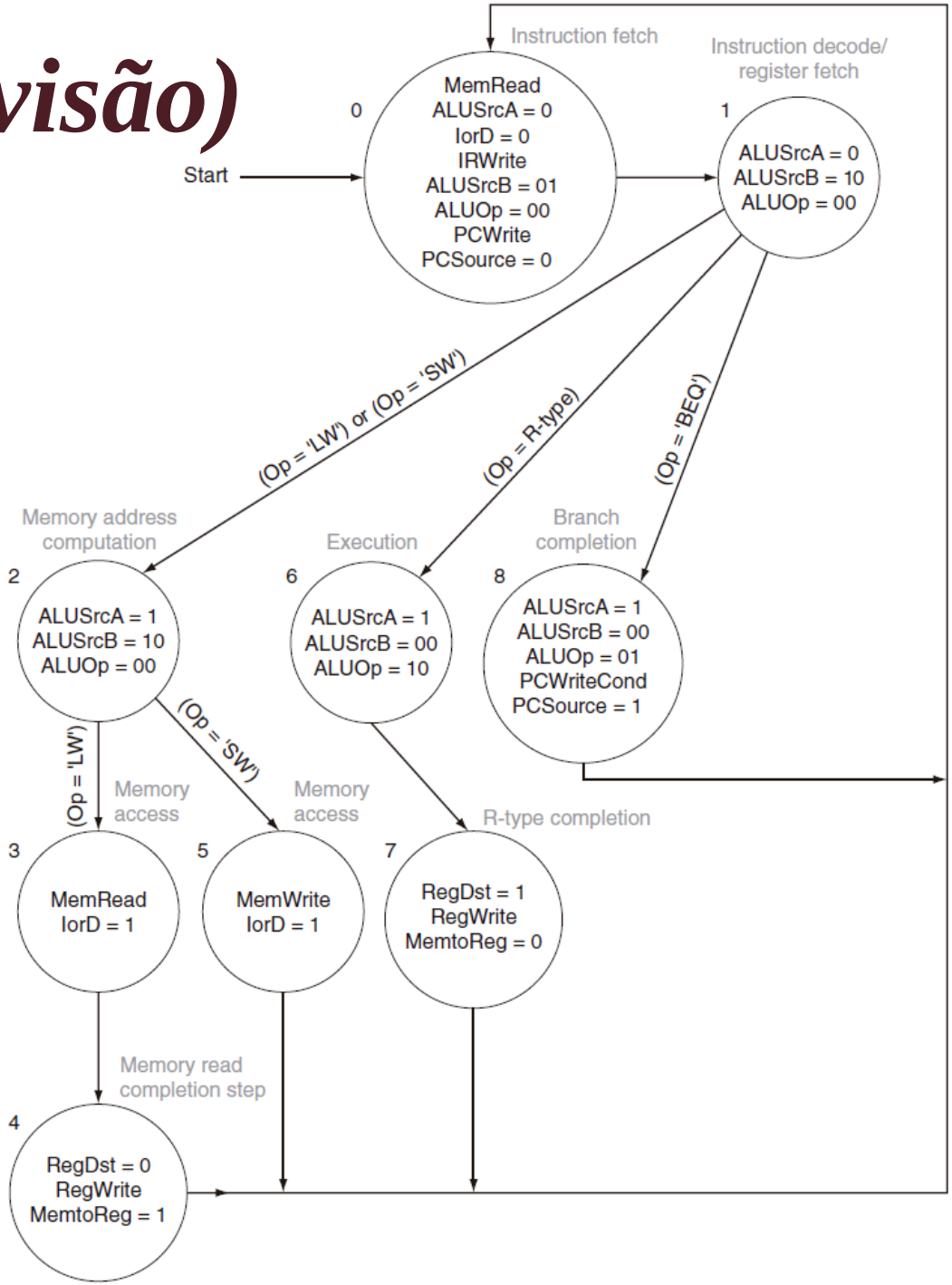
1. Buscar instrução da memória e incrementar PC;
2. Decodificar a instrução, ler registradores e calcular endereço de desvio condicional;
3. Depende da instrução que está sendo executada:
   - Load ou Store: calcular endereço da memória;
   - Formato R: realizar a operação logica ou aritmética requisitada;
   - Beq: realizar a comparação entre o conteudo dos dois registradores e realiza o desvio se a cond. for aceita;
   - Jump: atualizar o PC com o endereço de desvio;

Multiciclo (revisão)

(passos para execução das instruções)

4. Depende da instrução que está sendo executada:
   - Load: Lê o dado da memória;
   - Store: Escreve o dado na memória;
   - Formato R: Escreve o resultado da operação no conjunto de registradores;
5. Depende da instrução que está sendo executada:
   - Load: Escreve o dado lido no conjunto de registradores;

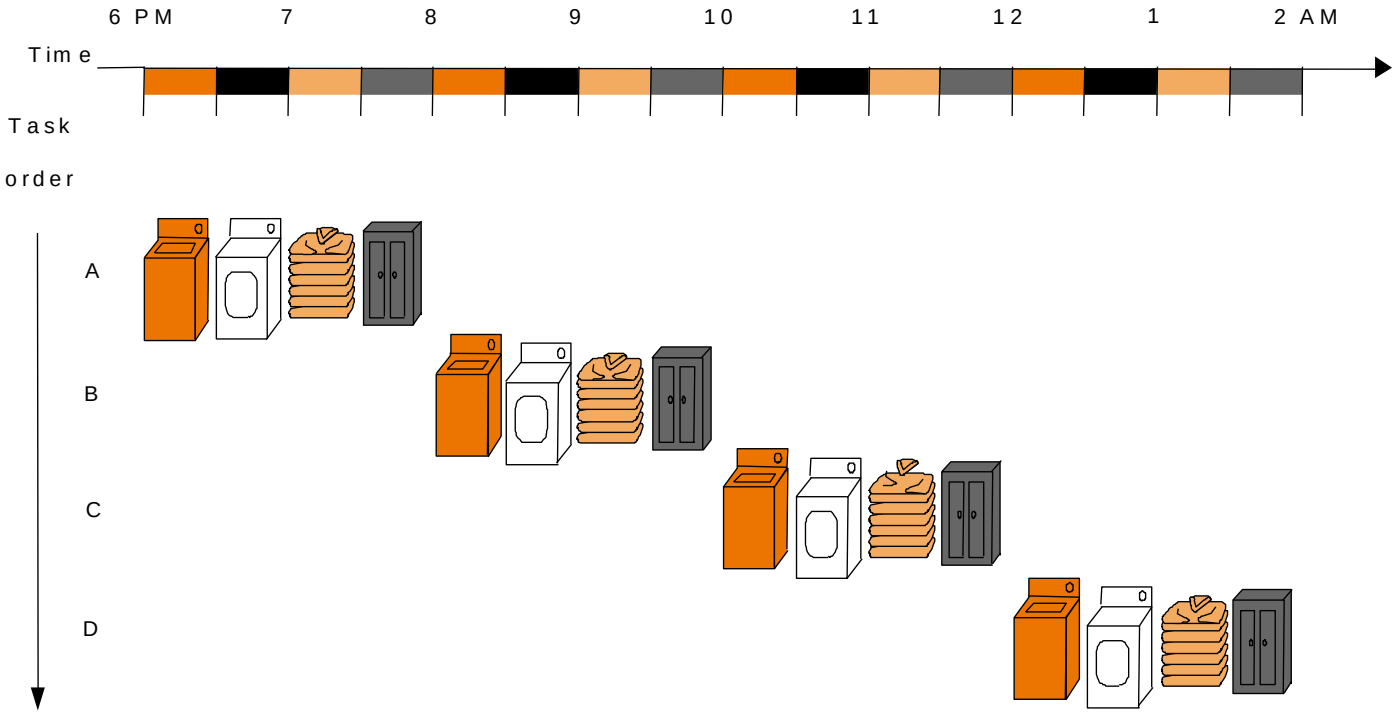
Multiciclo (revisão)



Multiciclo

(analogia)

- **Instrução**: “lavar roupa”;
- **Etapas:** lavar, secar, passar e guardar;
- Execução das tarefas **de forma sequencial** (leva 8h):



Pipeline

(definição)

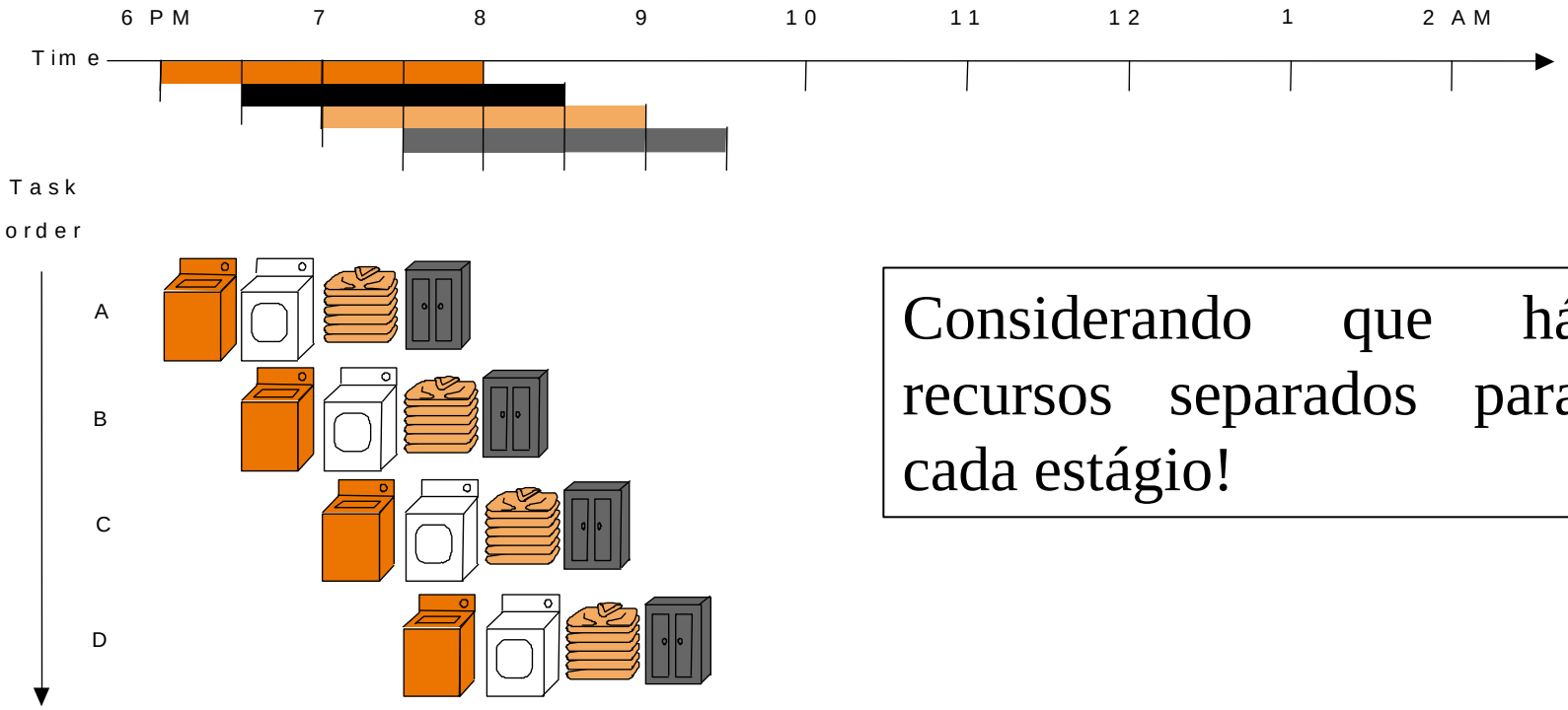
- *Pipelining* é uma técnica de implementação onde várias instruções são executadas simultaneamente

- Atualmente o pipeline é quase universal

Pipeline

(analogia)

- **Instrução**: “lavar roupa”;
- **Etapas**: lavar, secar, passar e guardar;
- Execução das tarefas **de forma paralela** (leva 3,5h):



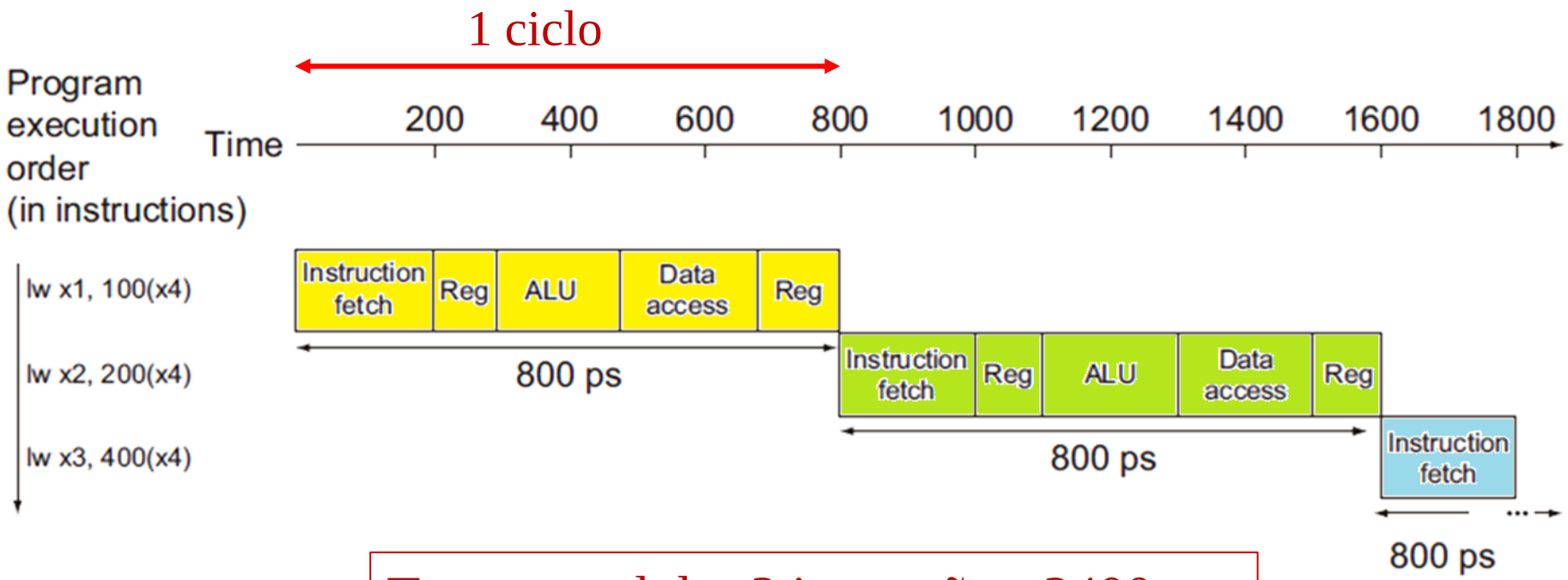
Considerando que há recursos separados para cada estágio!

Tempo das instruções

- **Monociclo**
  - **Tempo do ciclo** = tempo total da instrução mais longa
- **Multiciclo**
  - **Tempo do ciclo** = tempo total da etapa mais longa
  - Instruções executam apenas etapas necessárias (nº de ciclos entre 3 e 5)
- **Pipeline**
  - **Tempo do ciclo** = tempo total da etapa mais longa
  - Instruções executam todas as etapas, até mesmo as desnecessárias (nº de ciclos fixo: 5)

Tempo das instruções - Monociclo

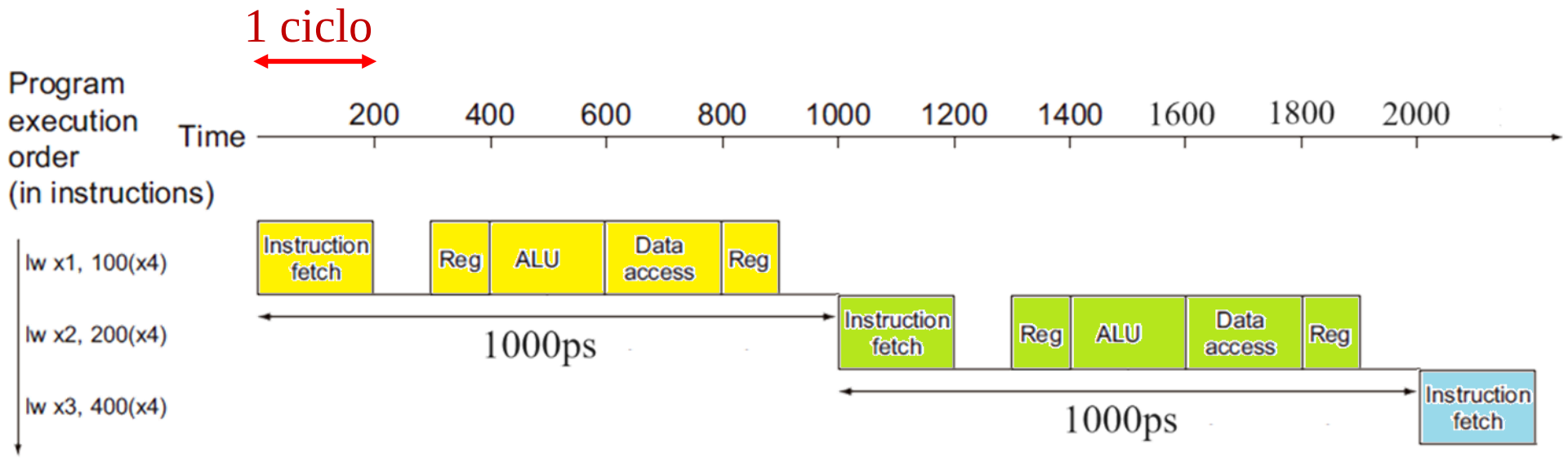
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps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



Tempo total das 3 instruções: 2400ps

Tempo das instruções - Multiciclo

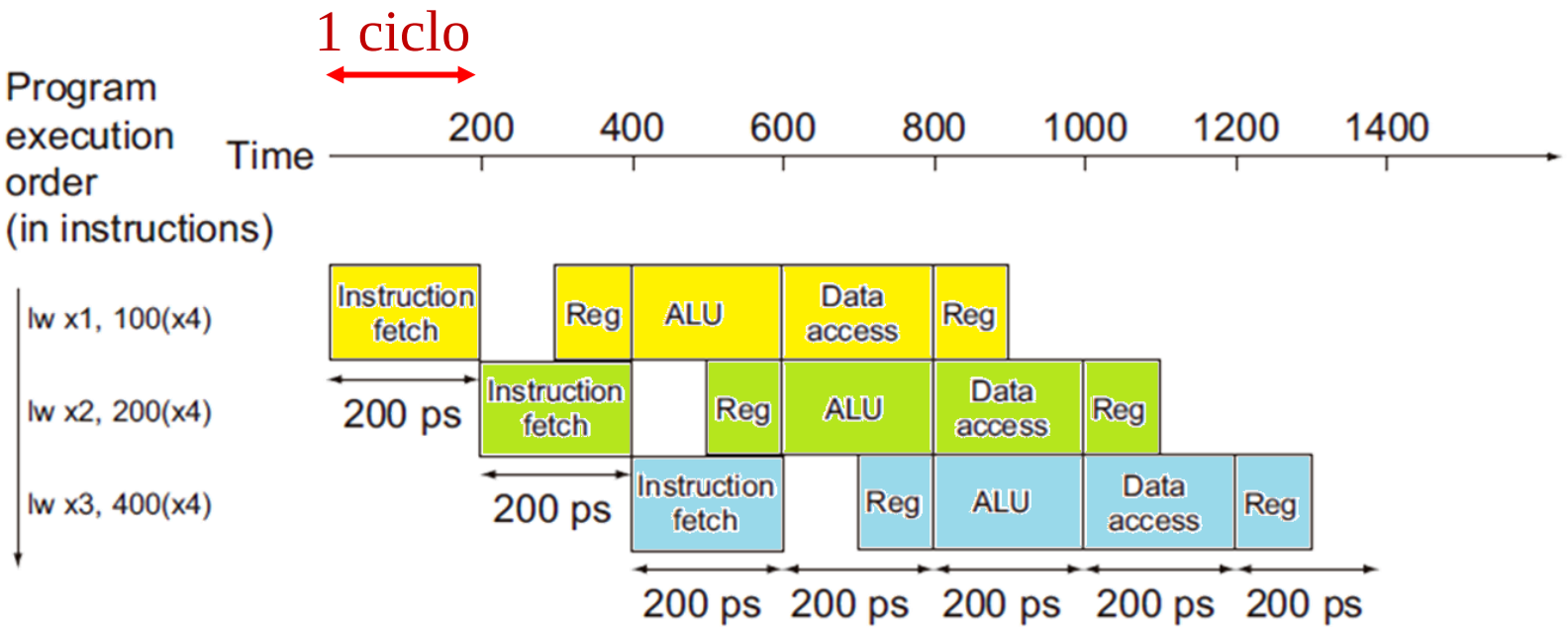
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps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



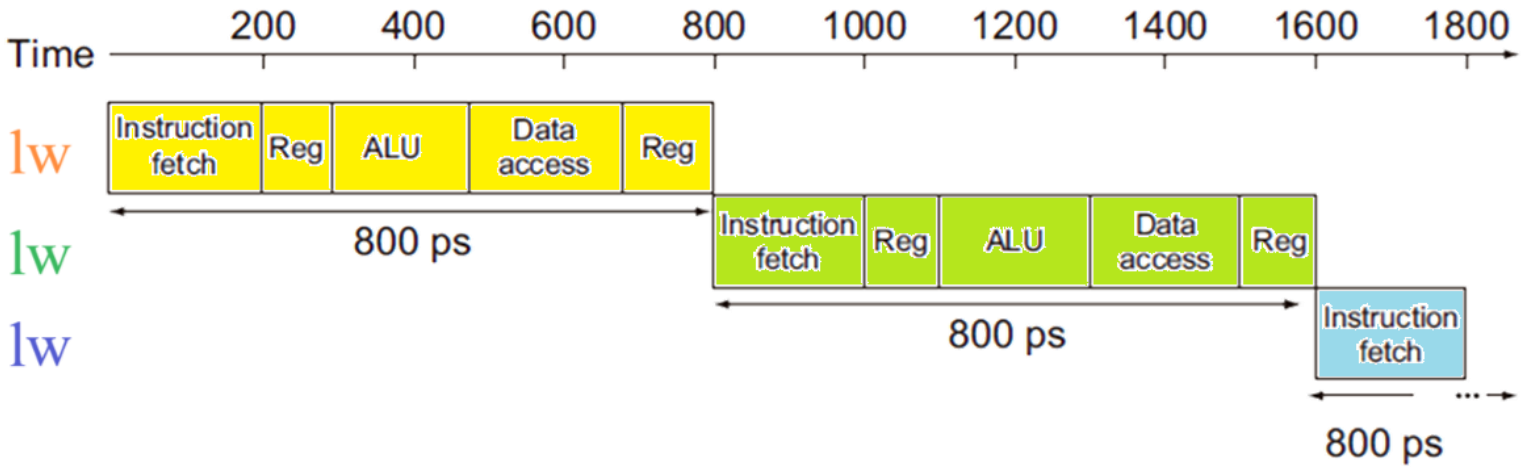
Tempo total das 3 instruções: 3000ps

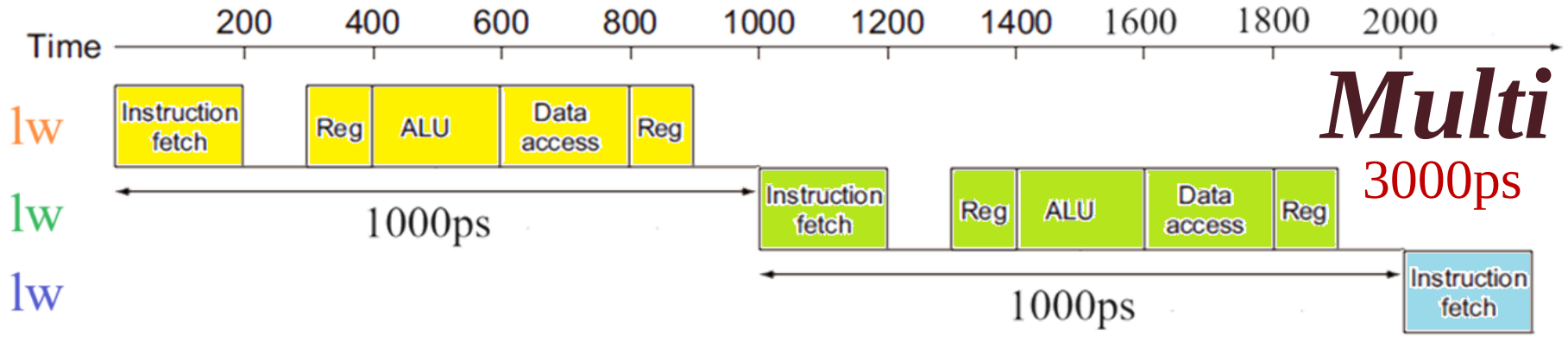
Tempo das instruções - Pipeline

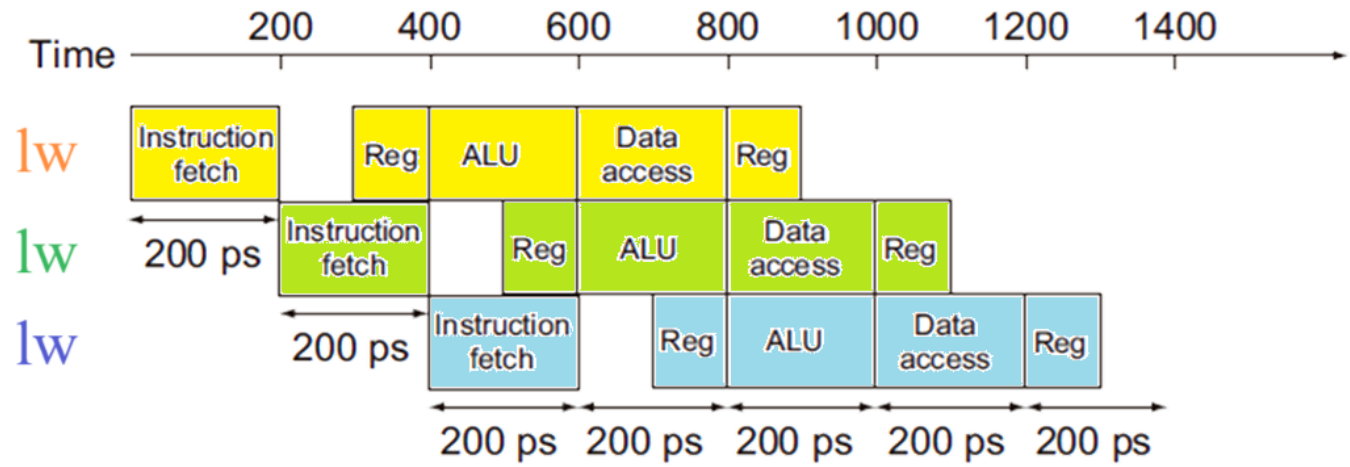
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps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

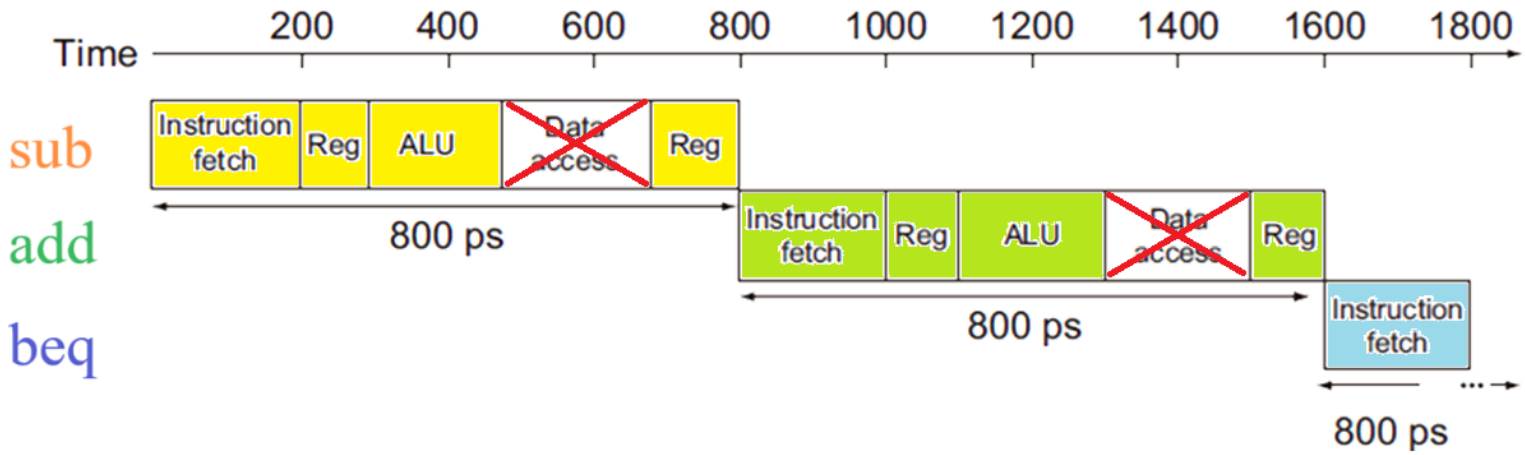


Tempo total das 3 instruções: 1400ps


Time
200
400
600
800
1000
1200
1400
1600
1800
lw
lw
lw
Instruction fetch
Reg
ALU
Data access
Reg
800 ps
800 ps
800 ps
Mono
2400ps

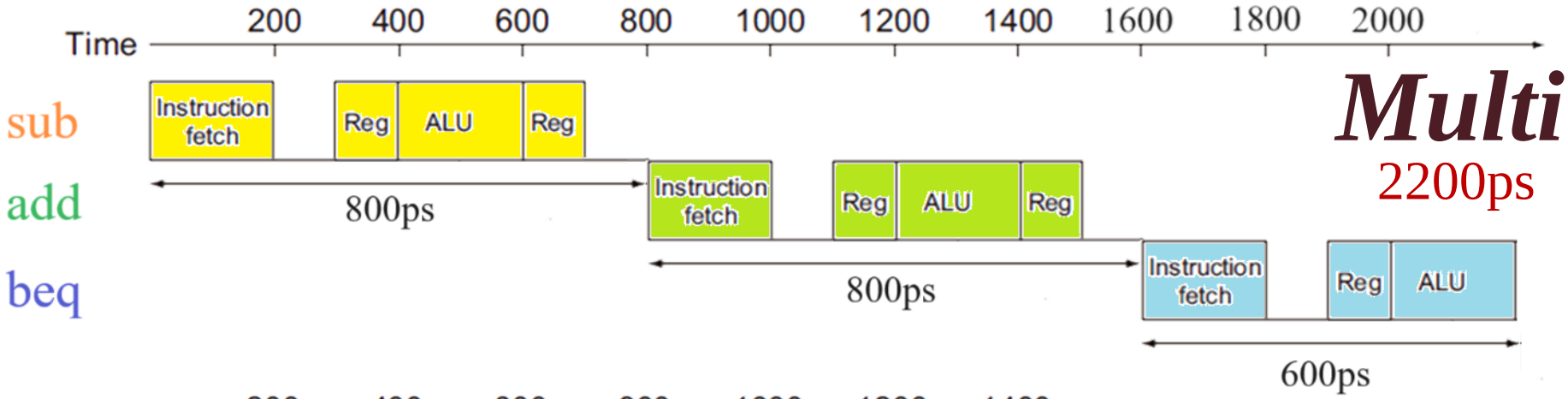

Time
200
400
600
800
1000
1200
1400
1600
1800
2000
lw
lw
lw
Instruction fetch
Reg
ALU
Data access
Reg
1000ps
1000ps
Multi
3000ps


Time
200
400
600
800
1000
1200
1400
lw
lw
lw
Instruction fetch
Reg
ALU
Data access
Reg
200 ps
200 ps
200 ps
200 ps
200 ps
200 ps
200 ps
Pipe
1400ps



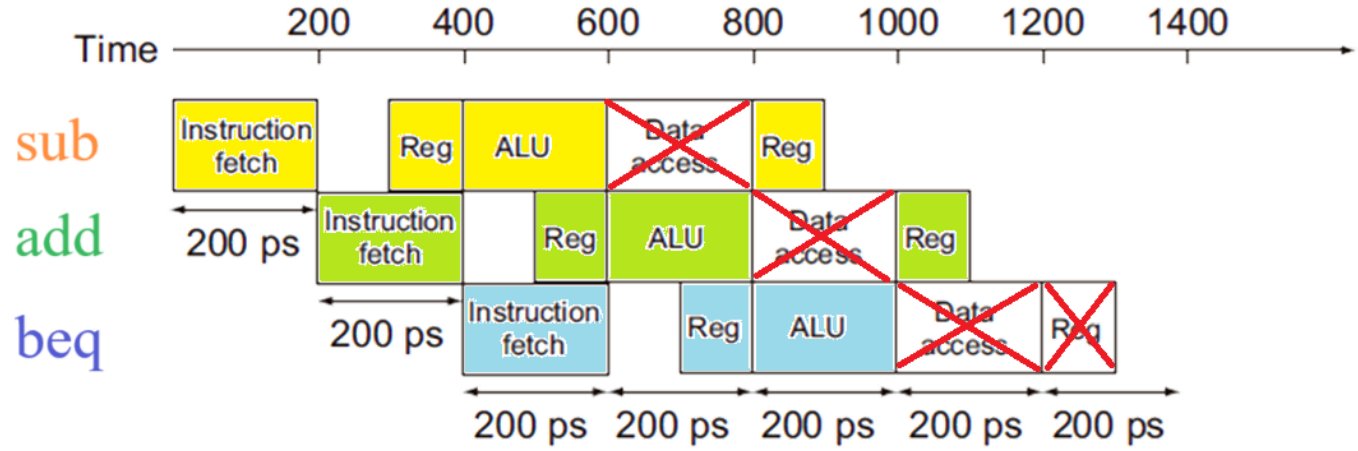
Mono

2400ps



Multi

2200ps



Pipe

1400ps

Pipeline

(definição)

- *Questão*: o tempo total de execução de uma instrução é menor no pipeline?
  - O *pipelining* não reduz o **tempo total** da execução (latência) de uma instrução isolada, e sim o **throughput** (vazão) da carga de trabalho

Pipeline de 5 estágios

*"Para um pipeline de **5 estágios**, o fluxo de instruções será **5 vezes maior** que em uma implementação sequencial"*

•Ou seja, se todas as etapas levassem aproximadamente a mesma quantidade de tempo e houver trabalho em todas as etapas, então a aceleração (*speed-up*) devido ao pipeline é igual ao número de estágios existentes

("Mundo ideal")

Pipeline de 5 estágios

•Essa reflexão sobre o ***speed-up*** do pipeline pode ser vista através de uma fórmula (assumindo as condições ideais anteriores)

$$\text{Tempo entre instruções}_{\text{com pipeline}} = \frac{\text{Tempo entre instruções}_{\text{sem pipeline}}}{\text{Número de estágios do pipe}}$$

•Como ficaria no caso do exemplo anterior com as instruções load's?

$$\text{Tempo entre instruções}_{\text{com pipeline}} = \frac{800}{5} = \text{ciclo de clock de 160ps}$$

Pipeline de 5 estágios

•***Para pensar***: uma situação em que o nº de tarefas não é grande em relação ao nº de estágios, como isso afetaria o *pipeline*, para melhor ou pior?

•**Lembrando**: *pipeline* é uma forma de **implementação** (não está vinculado à **arquitetura** X ou Y)

Pipeline de 5 estágios

•***Para pensar***: uma situação em que o nº de tarefas não é grande em relação ao nº de estágios, como isso afetaria o *pipeline*, para melhor ou pior?

–Para pior, quando o *pipeline* não está completamente cheio, isso afeta o desempenho (caso do 1º exemplo com só 3 load's)

- Mas programas reais executam bilhões de instruções

–Com os estágios completamente cheios na maior parte do tempo e um grande número de instruções, o aumento no rendimento deve se aproximar do nº de estágios.

Pipeline de 5 estágios

- No entanto, o que faz o *speed-up* do *pipeline* **não** aumentar proporcionalmente ao número de estágios?
  - …os estágios não serem perfeitamente equilibrados
  - …há também conflitos existentes entre as instruções (veremos mais adiante)
  - …os componentes de interligação necessários p/ implementar um *pipeline* e corrigir esses conflitos introduzem uma latência adicional.

Pipeline
+
RISC-V

Pipeline RISC-V

•Como o conjunto de instruções (ISA) foi projetado para a execução em pipelining?

a. Instruções de tamanho variável **vs.** todas instruções de mesmo tamanho?
   - Em RISC-V todas as instruções têm o mesmo tamanho: facilidade em buscá-las no 1º estágio e decodificar no 2º
   - Exemplo: nas primeiras implementações do x86 (instruções variavam de 1 à 15 bytes), o pipeline é bem mais desafiador.

Pipeline RISC-V

•Como o conjunto de instruções (ISA) foi projetado para a execução em pipelining?

b. Operandos que residem em memória em qualquer instrução **vs.** operandos residindo em memória apenas através de load e store?

- Na RISC-V só aparecem em Loads e Stores
- X86: operandos residindo em memória são possíveis em instruções de operações, isso requereria em pipeline: estágio de endereço -> estágio da memória -> estágio de execução

Pipeline RISC-V

•Como o conjunto de instruções (ISA) foi projetado para a execução em pipelining?

c. Formato das instruções irregular **vs.** regular?

- RISC-V possui apenas alguns formatos de instrução, com registradores (de origem e destino) localizados no mesmo lugar em cada instrução

Passos das instruções em RISC-V

1. Busca da **instrução** e incremento do PC

2. Acesso ao banco de registradores e **decodificação** da instrução

3. **Execução** da operação ou cálculo do endereço de memória

4. Acesso à **memória** de dados (para leitura ou escrita), se necessário.

5. **Escreve** o resultado dentro de um **registrador**, se necessário

Pipeline de 5 estágios

1. Busca da **instrução** e incremento do PC → **IF: *Intruction Fetch***

2. Acesso ao banco de registradores e **decodificação** da instrução → **ID: *Intruction Decode***

3. **Execução** da operação ou cálculo do endereço de memória → **EX: *Execução***

4. Acesso à **memória** de dados (para leitura ou escrita), se necessário. → **MEM: *Memória***

5. **Escreve** o resultado dentro de um **registrador**, se necessário → **WB: *Write Back***

Pipeline de 5 estágios

(via de dados)

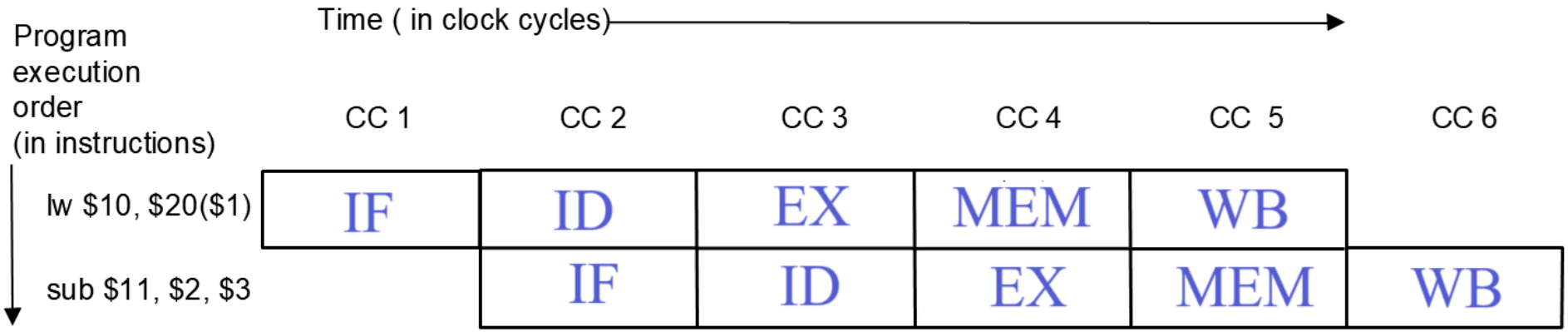
•Durante os ciclos de clock, cinco instruções estarão em execução;

•Divisão da via de dados em cinco partes, onde cada uma delas terá o nome do estágio correspondente:

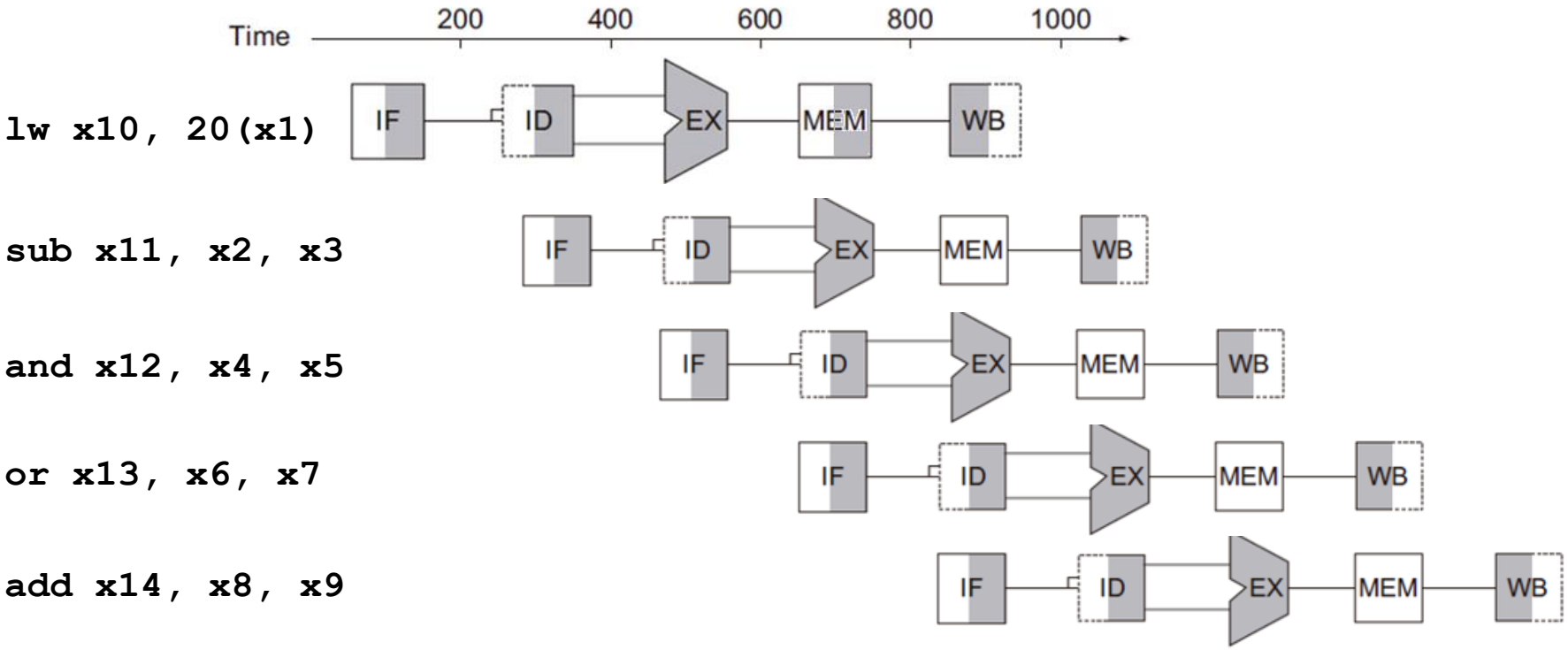
- •**IF**: busca a instrução;
- •**ID**: decodifica instrução e faz leitura de registradores;
- •**EX**: executa operação ou calcula endereço;
- •**MEM**: acessa dados da memória (se necessário);
- •**WB**: escreve resultado em registrador (se necessário).

Pipeline

(execução de instruções em paralelo)



Pipeline



- Sombreamento indicando que o elemento é utilizado pela instrução
  - Sombreamento à direita no banco de registradores ou memória: o elemento é **lido** neste estágio
  - Sombreamento à esquerda: elemento foi **escrito** neste estágio

Conflitos

Conflitos

- Há situações no pipeline que a próxima instrução não pode ser executada no ciclo de clock seguinte
- Esses eventos são chamados de **conflitos** (***hazards***)
- Há 3 tipos de conflitos:
  1. **Estruturais**
  2. **De dados**
  3. **De Controle**

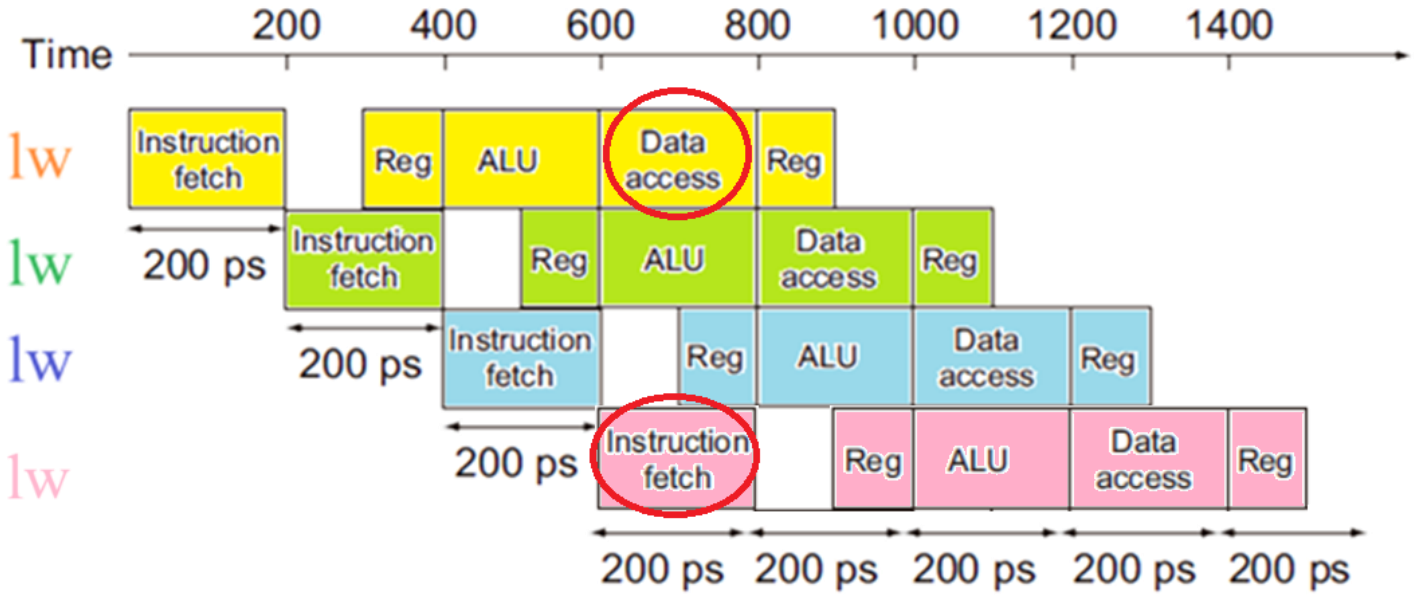
Conflitos estruturais

1. Conflitos Estruturais

- Situação onde **o hardware não suporta a combinação de instruções que se deseja executar num mesmo ciclo de clock**
  - Analogia: ocorreria um conflito estrutural se a máquina de lavar e secar fosse a mesma e quiséssemos lavar um conjunto de roupas e secar um outro conjunto
- O conjunto de instruções **RISC-V** foi projetado para ser ***pipelined***!
  - Isso torna fácil aos projetistas evitar riscos estruturais

1. Conflitos Estruturais

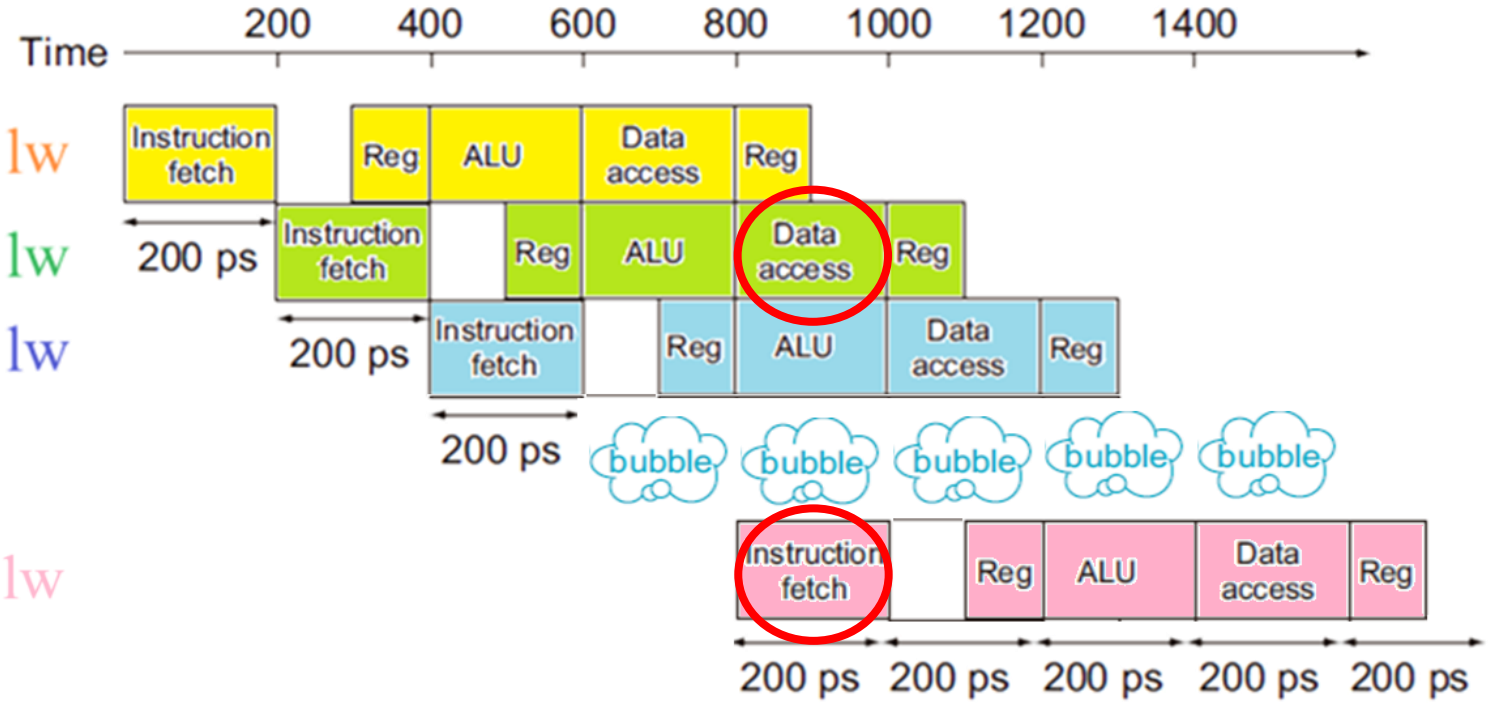
•**Suponha**: se houvesse apenas uma memória (ao invés de duas). No *pipeline* abaixo, se tivesse uma 4ª instrução **load** o que ocorreria?



•No 4º ciclo: a primeira instrução tenta acessar dados da memória e a quarta instrução estaria buscando uma instrução da mesma memória!

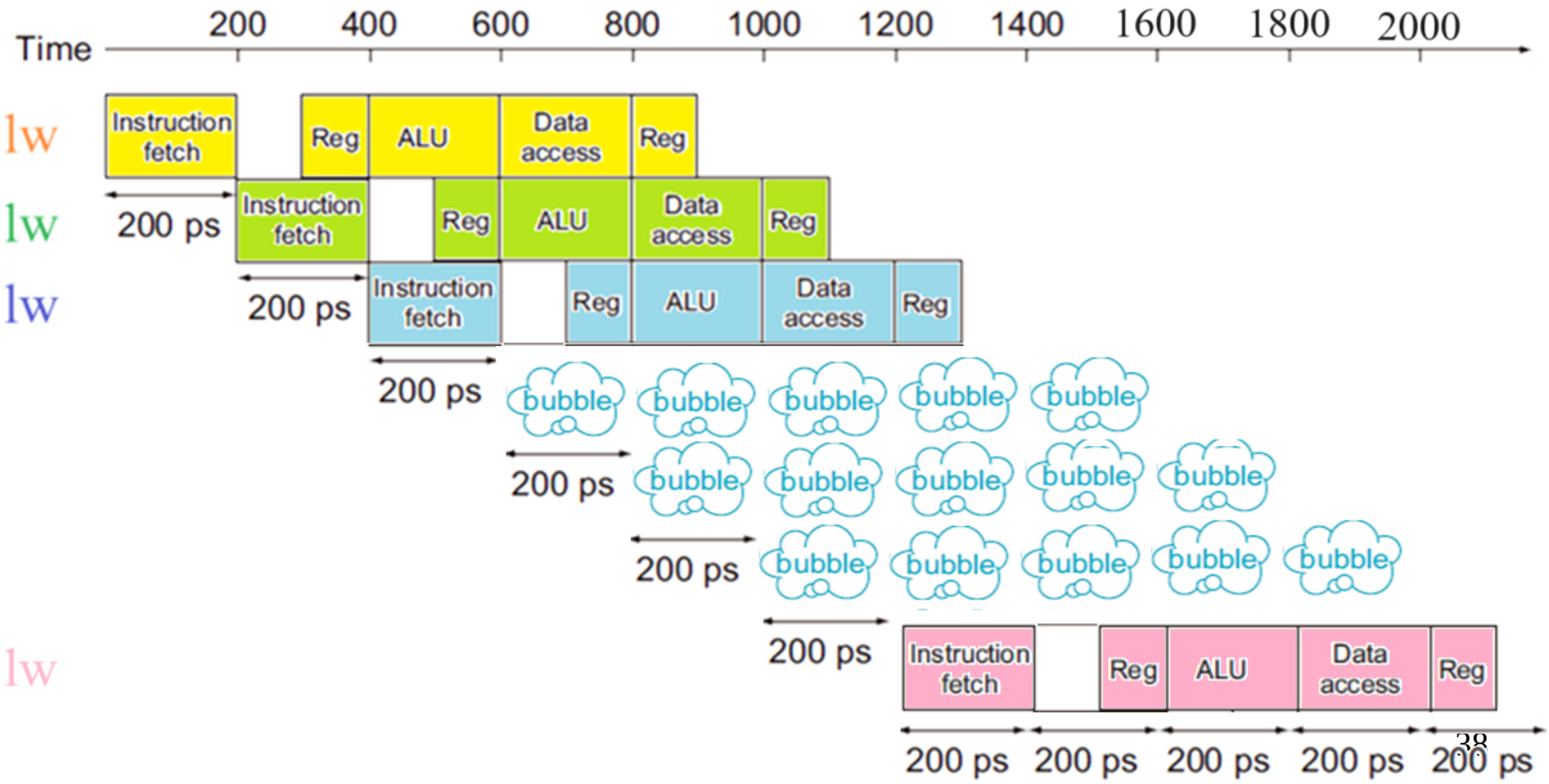
1. Conflitos Estruturais

- **Solução 1**: **inserir uma "bolha"**. Problema resolvido?
  - –Passamos a ter problema com a **load** seguinte!



1. Conflitos Estruturais

•Ou seja, a cada 3 **load's** seguidas, seria necessário 3 bolhas para se inserir uma 4ª instrução



1. Conflitos Estruturais

•**Solução 2**: **replicar unidades funcionais** para eliminar o conflito e permitir execução simultânea.

•Como decider se vale a pena replicar ou não?

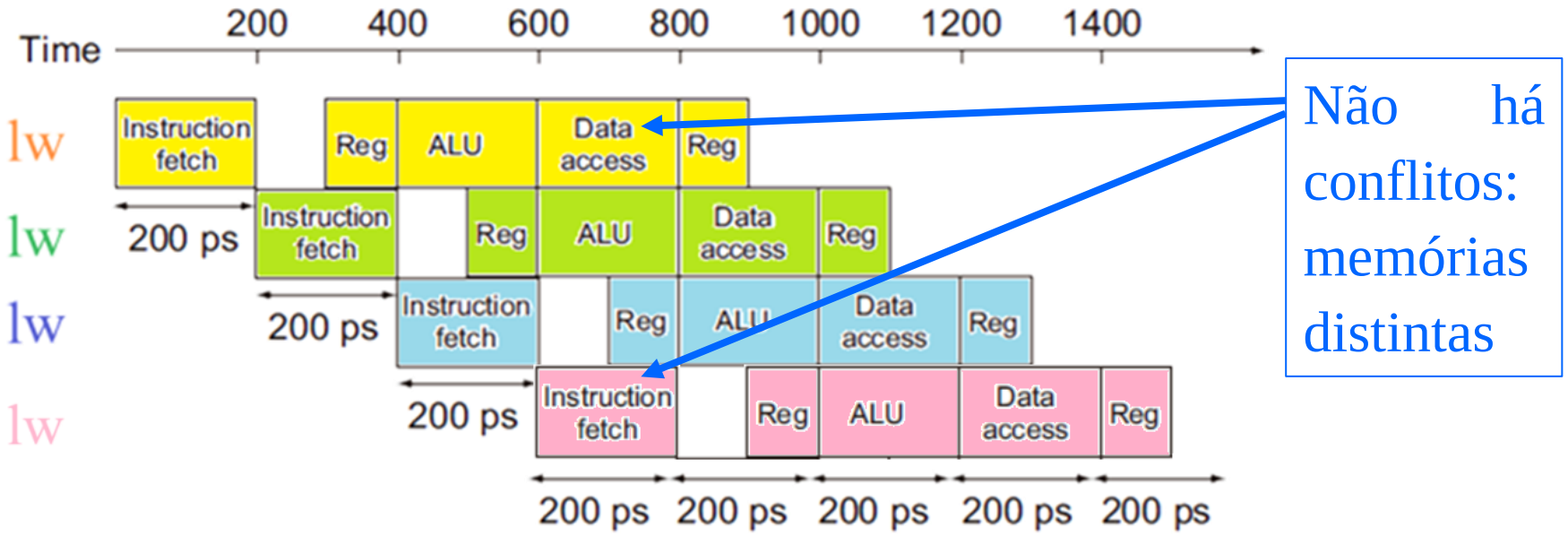
–Mais unidades funcionais encarecem o projeto

–Mais unidades funcionais podem alterar o caminho crítico do projeto, e portanto, o tempo do ciclo

•Qual a frequência da ocorrência dos conflitos estruturais daquele tipo?

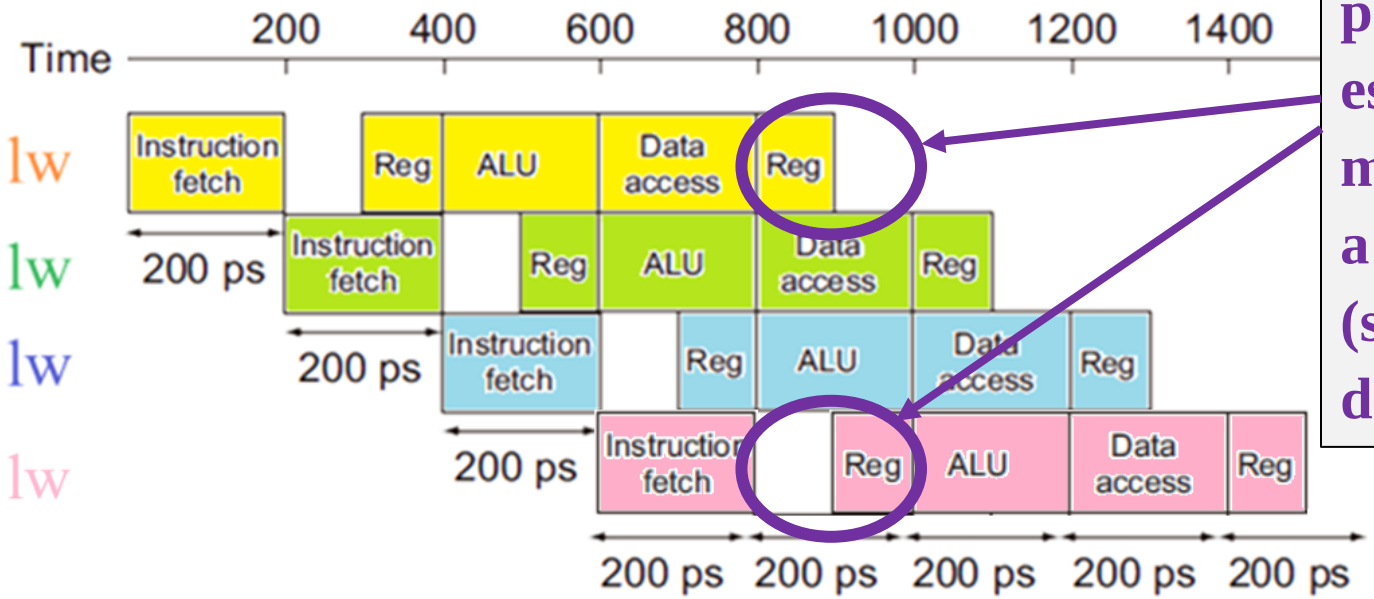
1. Conflitos Estruturais

•*Exemplo*: com duas portas de leitura na memória: uma para instruções e outra para dados (= Monociclo). Desta forma são consideradas unidades funcionais distintas.



1. Conflitos Estruturais

•**Questão**: e no caso do banco de registradores sendo utilizado tanto pela 1ª instrução quanto pela 4ª, há conflito estrutural?



Não há problemas, pois a etapa 5 escreve (primeira metade do ciclo) e a etapa 2 lê (segunda metade do ciclo)

Conflitos de dados

2. Conflitos de Dados

•Conflitos de dados ocorrem quando **o pipeline precisa ser paralisado porque uma etapa precisa esperar outra completar**

–Analogia: você encontra uma meia sem par na hora de dobrar, e a outra ainda está sendo lavada. Ela precisa passar pelas etapas de secar e passar antes de ter acesso.

•No *pipeline*: dependência de uma instrução por outra que ainda está no *pipeline*. Exemplos?

2. Conflitos de Dados

•*Exemplo*: uma instrução `add` seguida por uma `sub` que usa a soma anterior:

```
add x19, x0, x1

sub x2, x19, x3
```

•A `add` não escreve o resultado enquanto não chega no 5º estágio. Soluções?

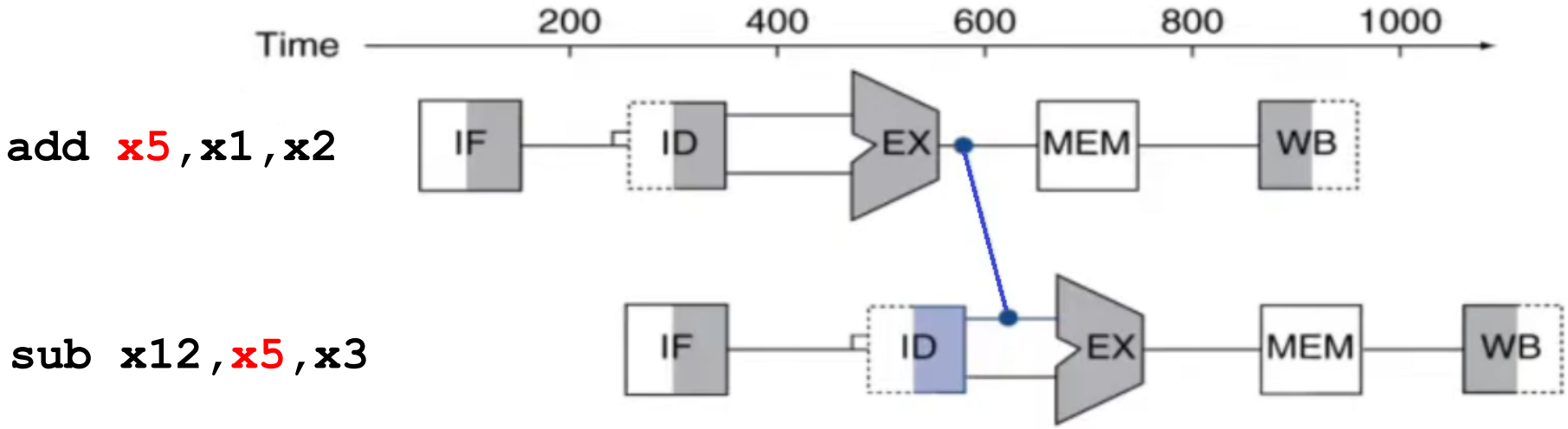
–Desperdiçar 3 ciclos de clock no pipeline (*stall* no pipe)

–Não esperar a conclusão da `add` para resolver o conflito: assim que a ULA calcular a soma, fornecê-la como entrada para a `sub`. -> Necessário adicionar hardware extra

–Confiar nos compiladores para remover esses perigos

2. Conflitos de Dados

•**Solução 1**: encaminhar o dado, se possível (***forwarding***)



•Saída da ULA é encaminhada para entrada da propria ULA, sem precisar salvar em registrador antes

•Pois a **sub** precisa do dado atualizado no início da 3ª etapa

2. Conflitos de Dados

•**Solução 2**: solução em software pelo compilador: reordenar o Código para evitar bolhas

- Exemplo:

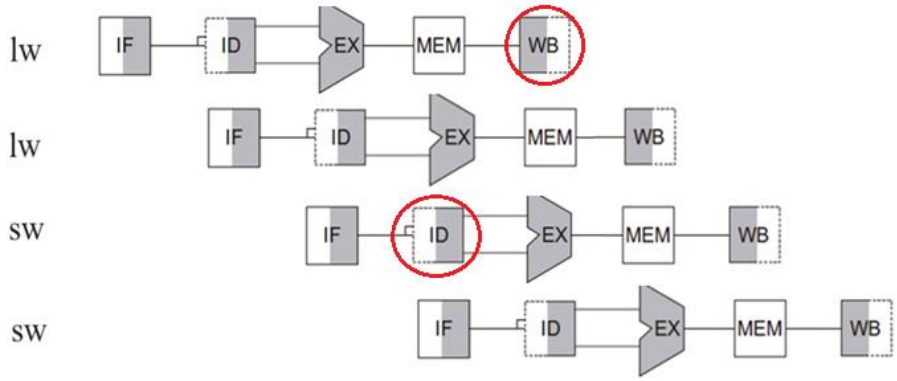
```
lw x2, 0(x1)
lw x4, 4(x1)
sw x2, 0(x1)
sw x3, 4(x1)
```

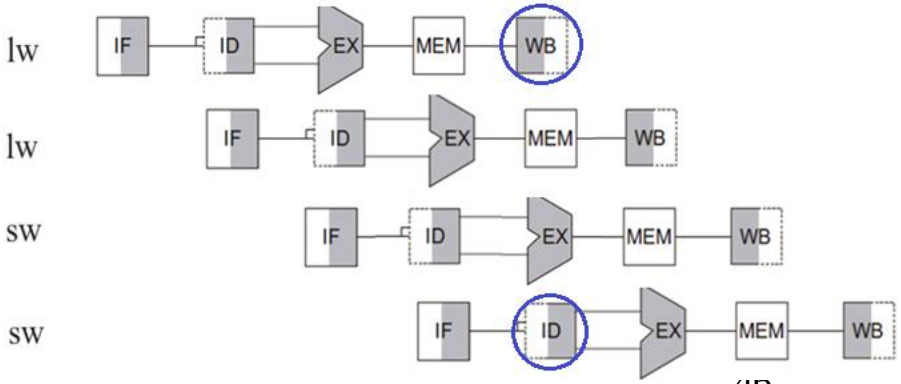
hazard

- Código reordenado:

```
lw x2, 0(x1)
lw x4, 4(x1)
sw x3, 4(x1)
sw x2, 0(x1)
```

Instruções trocadas





Conflitos de controle

3. Conflitos de Controle

•Necessidade de tomar a decisão com base nos resultados de uma instrução, enquanto outras estão sendo executadas ainda no pipeline (branches e jumps);

–Analogia: a lavanderia precisa limpar os uniformes de um time de futebol, mas como está muito suja a roupa, é preciso determinar o tipo do sabão e a temperatura da água ideais. Depois da primeira roupa seca, é possível avaliar o resultado e mudar as configurações e lavar novamente (*stall*)

–Prever/supor que será necessário produtos adicionais de fato (*previsão de desvio*)

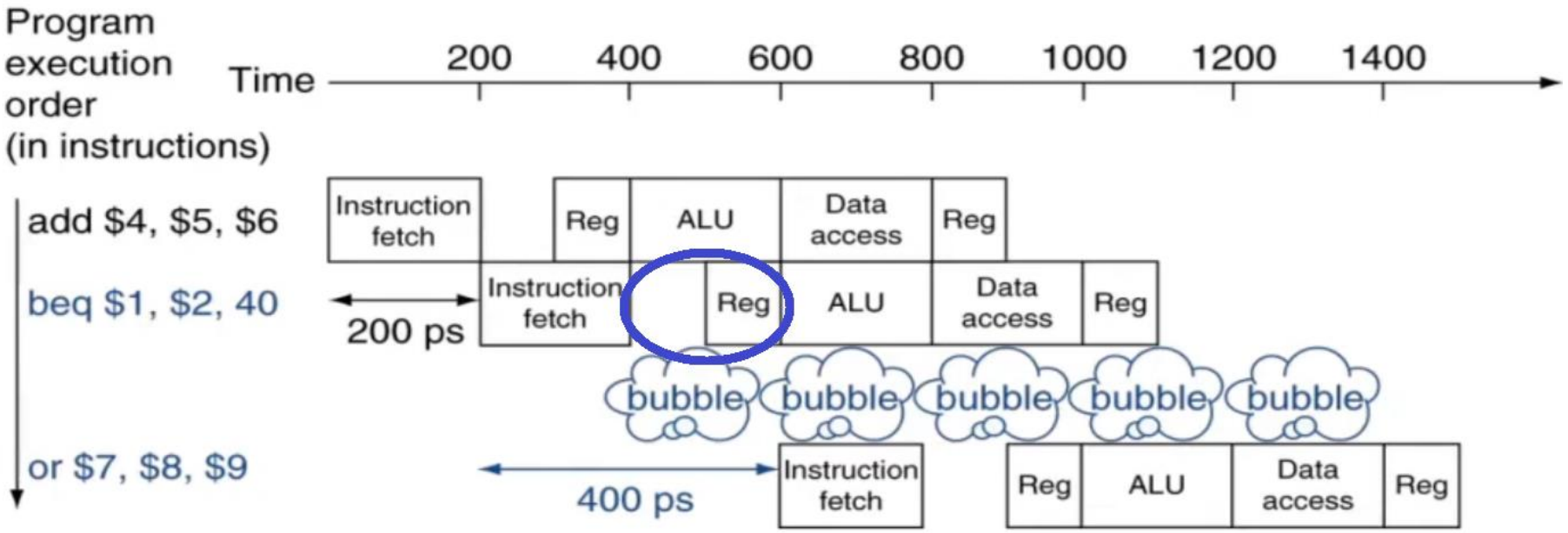
–Ou colocar roupas que não sejam de futebol até que a decisão seja tomada (*delayed branch*)

3. Conflitos de Controle

•Com a adição de hardware: cálculos de desvio passam a ser executadas no 2º estágio (ID)

–Mesmo assim o problema permanece

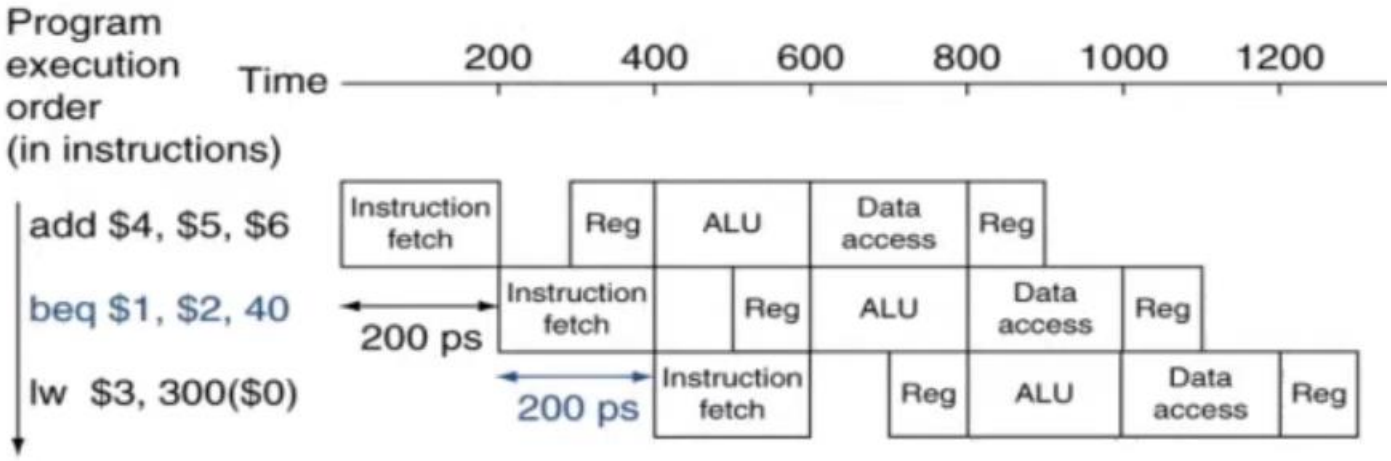
•**Solução 1**: necessário 1 bolha (**stall**)



3. Conflitos de Controle

•**Solução 2**: **previsão de desvio** – ex: supor que o desvio nunca será tomado

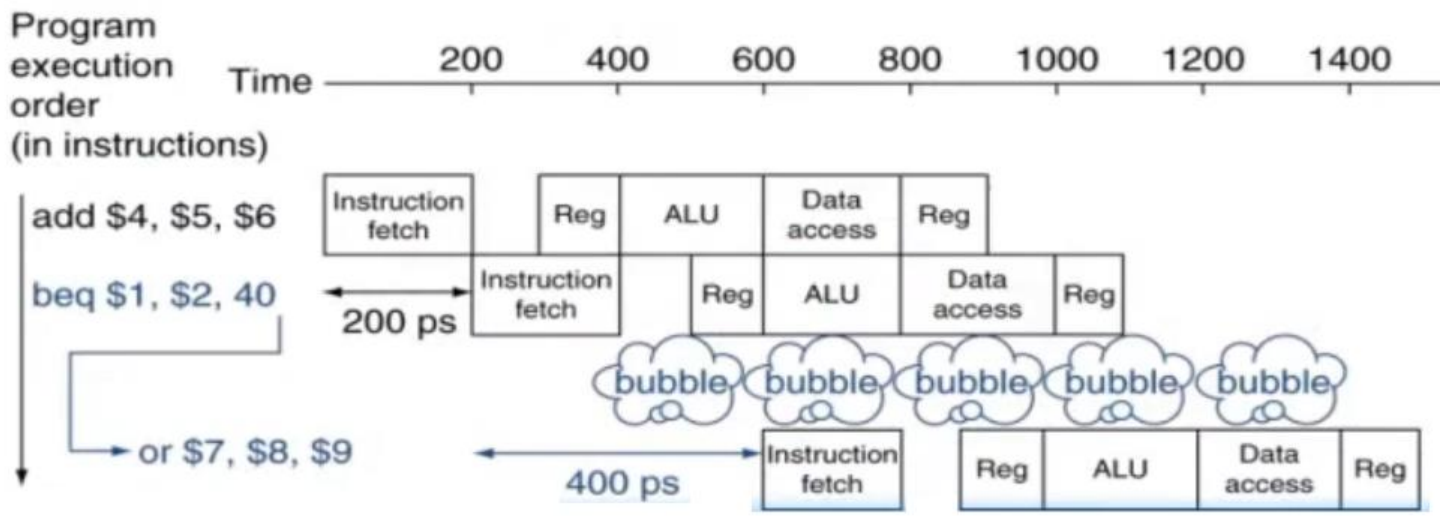
Previsão correta



•Se a previsão for correta, não é necessário bolhas

3. Conflitos de Controle

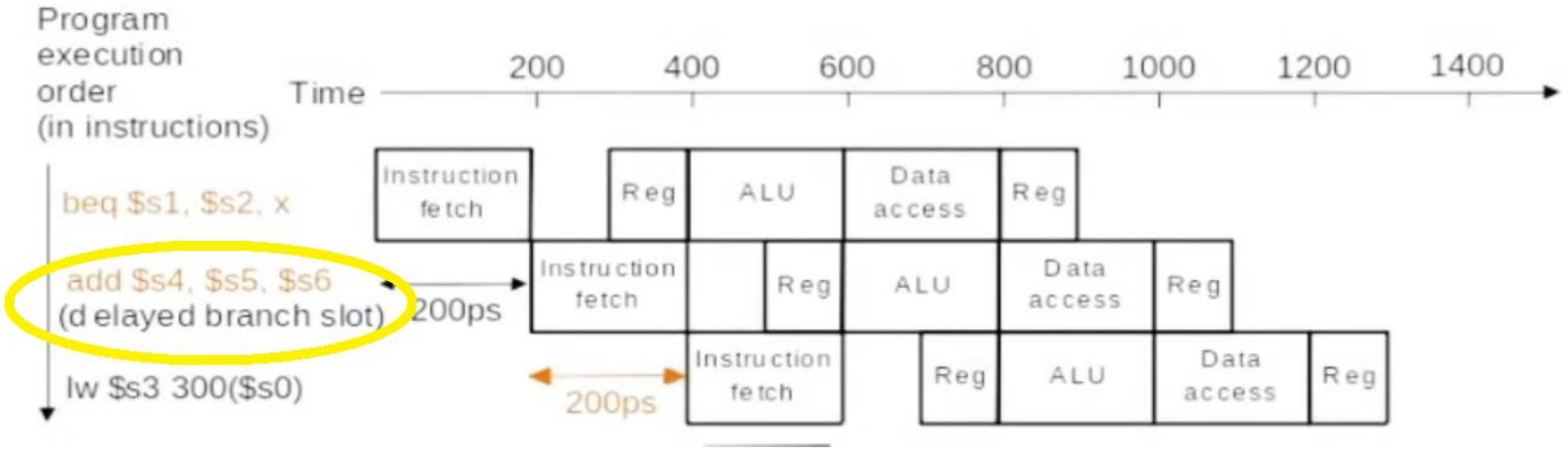
Previsão incorreta



•Se a previsão for incorreta – necessário bolha para anular a instrução **lw** anterior

3. Conflitos de Controle

•**Solução 3**: **delayed branch** – colocar uma instrução (independente) que era anterior ao branch para executar logo depois do branch, preenchendo a bolha (*delay slot*) – Feito pelo compilador



•Ex: **add** era uma instrução anterior à **beq**, mas foi colocada após, fazendo a vez de uma bolha

Referências

- PATTERSON, David A; HENNESSY, John L; Computer Organization and Design – The hardware/software interface RISC-V edition; Elsevier – Morgan Kaufmann/ Amsterdam.
- PATTERSON, David; Waterman, Andrew; The RISC-V reader: an open architecture atlas; First edition. Berkeley, California: Strawberry Canyon LLC, 2017.