É fácil projetar um sistema para situações onde tudo dá certo [...] O difícil é projetar para quando as coisas dão errado (Don Norman, The Design of Everyday Things).

Hazards

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

Hazards

Ao implementar o conceito de pipeline, criamos diversos problemas e complexidades.

Dentre os problemas, estão os **hazards**.

Risco ou perigo.

Impedem que a próxima instrução execute no ciclo de clock seguinte.

Hazards Estruturais

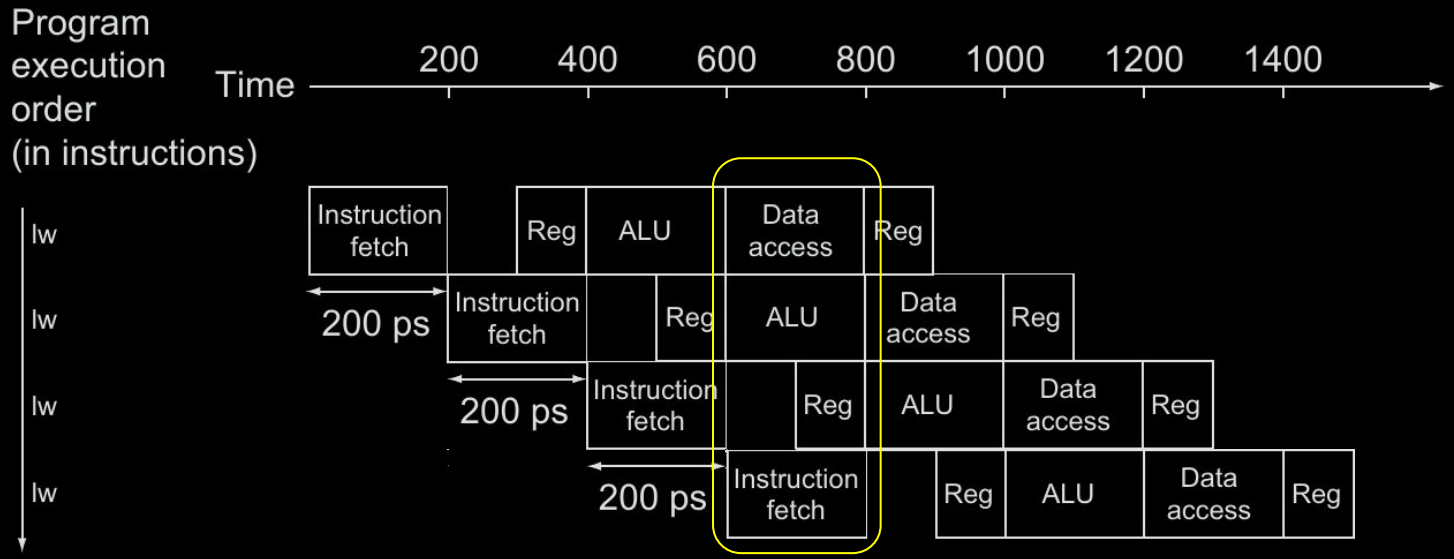
Em um **hazard estrutural** o hardware não pode manter duas instruções no pipeline, pois elas estão competindo por algum componente.

Exemplo: memória única, para dados e instruções.

Enquanto uma instrução está sendo carregada, outra está tentando escrever na memória ao mesmo tempo.

Exemplo de hazard estrutural

Se tivéssemos uma única memória, duas instruções poderiam competir pelo acesso.



Hazards Estruturais

Nossa implementação do RISC-V não sofre com hazards estruturais.

Os componentes que poderiam conflitar em determinado instante já estão duplicados (ex.: memórias separadas).

Um hazard estrutural muitas vezes é corrigido criando-se cópias das unidades funcionais.

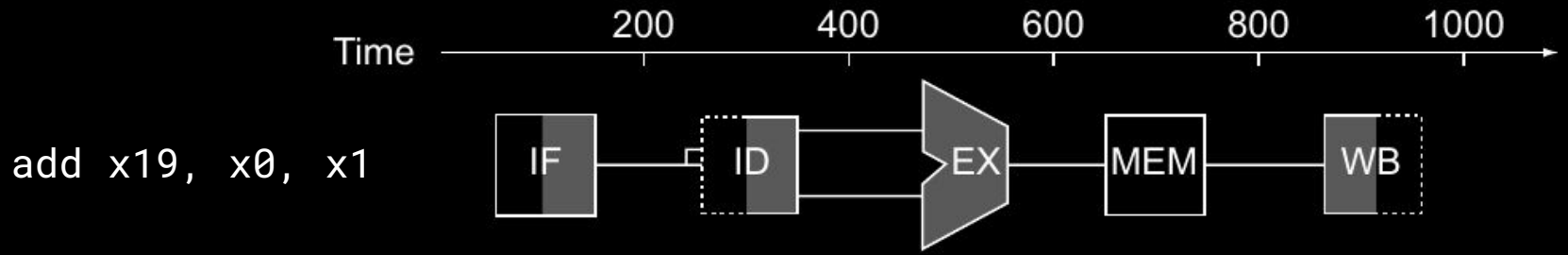
> Instruções em diferentes estágios do pipeline utilizam diferentes cópias da unidade.

Hazard de Dados

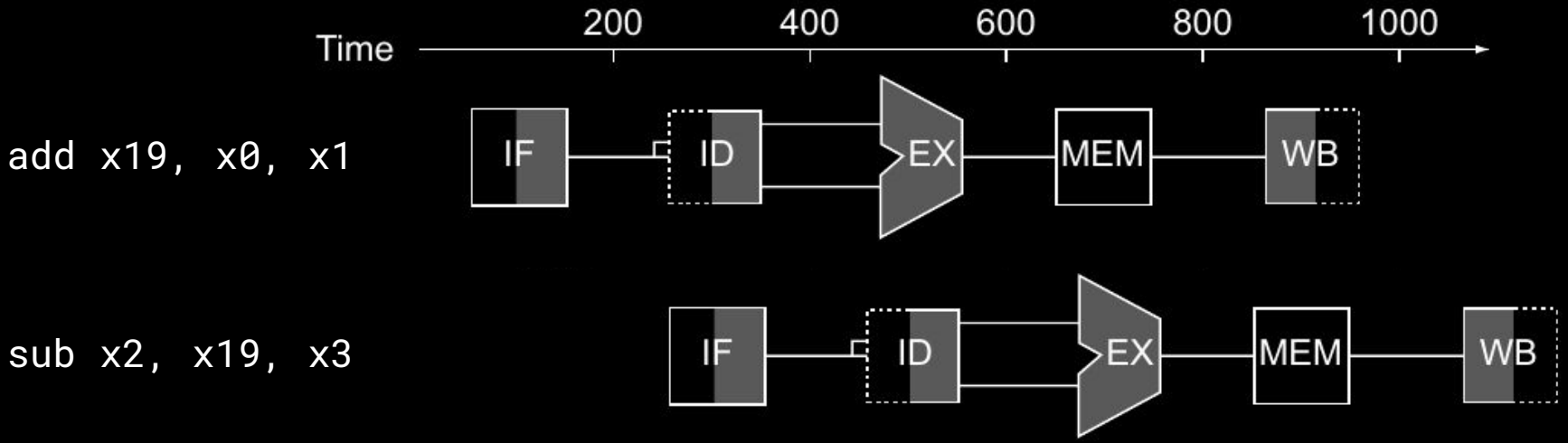
Considere as seguintes instruções:

```
add x19, x0, x1
sub x2, x19, x3
```

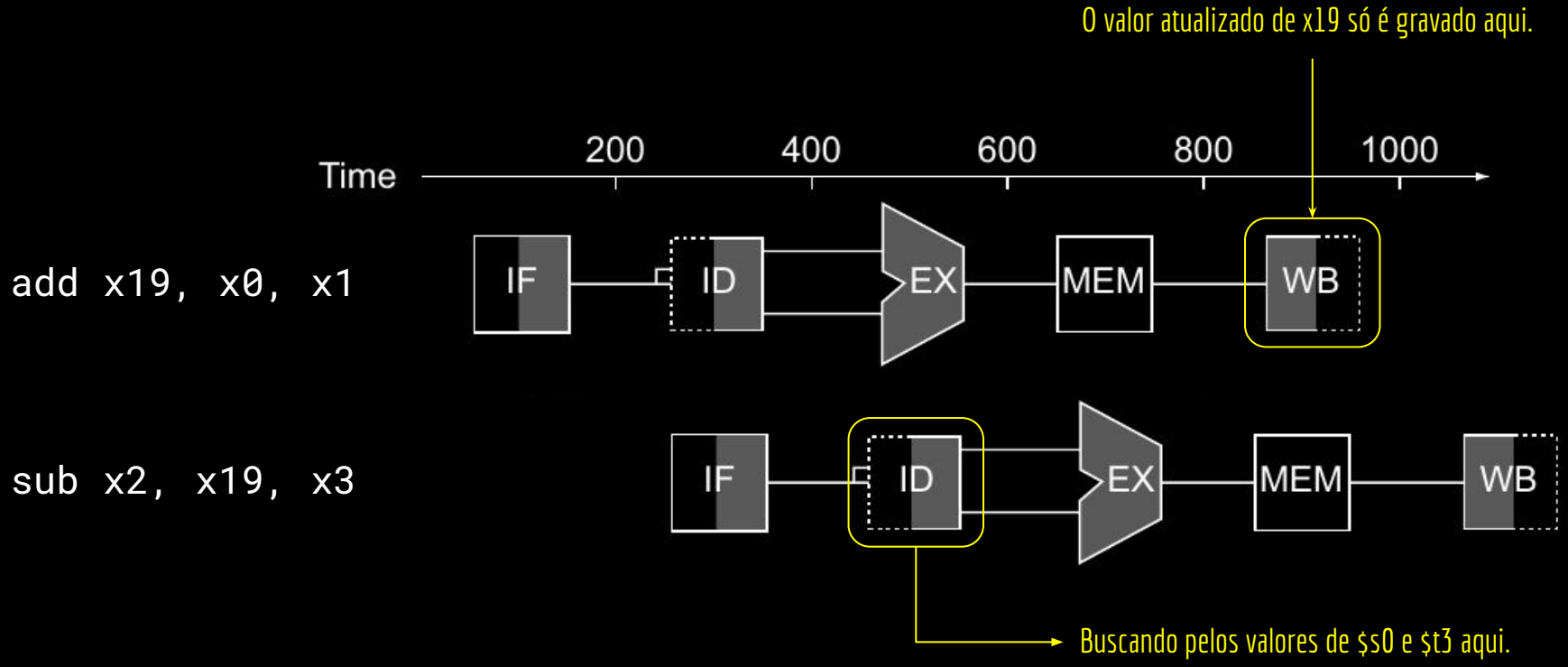
Considere diagrama da execução do `add`, e adicione o diagrama do `sub` logo abaixo (em um pipeline). Identifique o que está sendo feito em cada etapa e qual o problema que está acontecendo.



Hazard de Dados



Hazard de Dados



Hazard de Dados

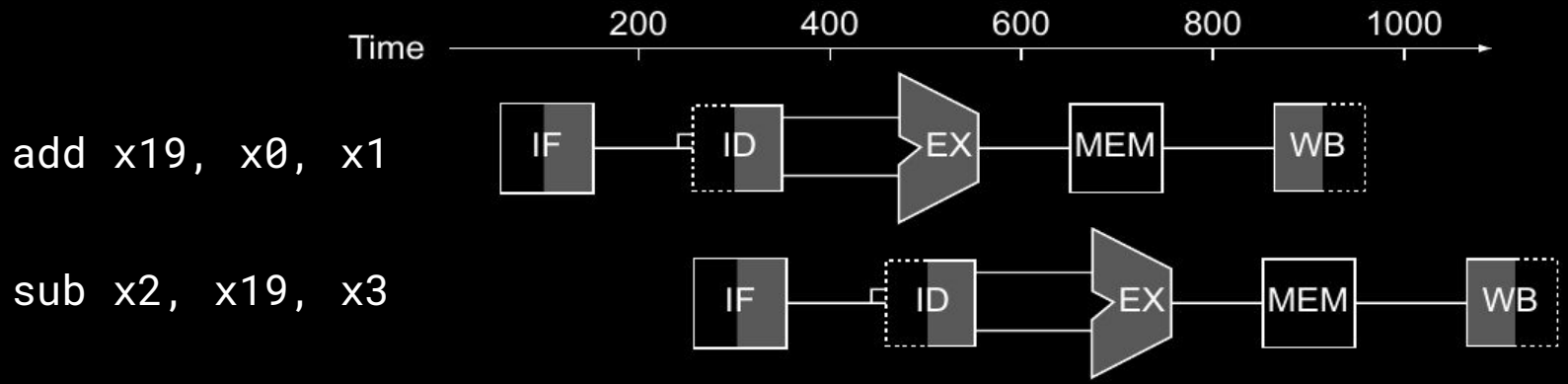
O pipeline precisa ser interrompido. Uma etapa precisa que outra seja concluída para que o dado esteja pronto.

Uma instrução depende de outra.

No circuito, o resultado é escrito no estágio **WB**.

Mas em que estágio o resultado já está pronto e só não foi gravado?

Como utilizar isso para resolver o problema?



Forwarding

Alguns hazards de dados podem ser mitigados por **forwardings**.

Conhecidos também como **bypassings**.

Ideia: “Emprestar” o resultado de uma unidade antes da operação ter sido completada (escrito na memória ou no banco de registradores).

Forwarding

Alguns hazards de dados podem ser mitigados por **forwardings**.

Conhecidos também como **bypassings**.

Ideia: “Emprestar” o resultado de uma unidade antes da operação ter sido completada (escrito na memória ou no banco de registradores).

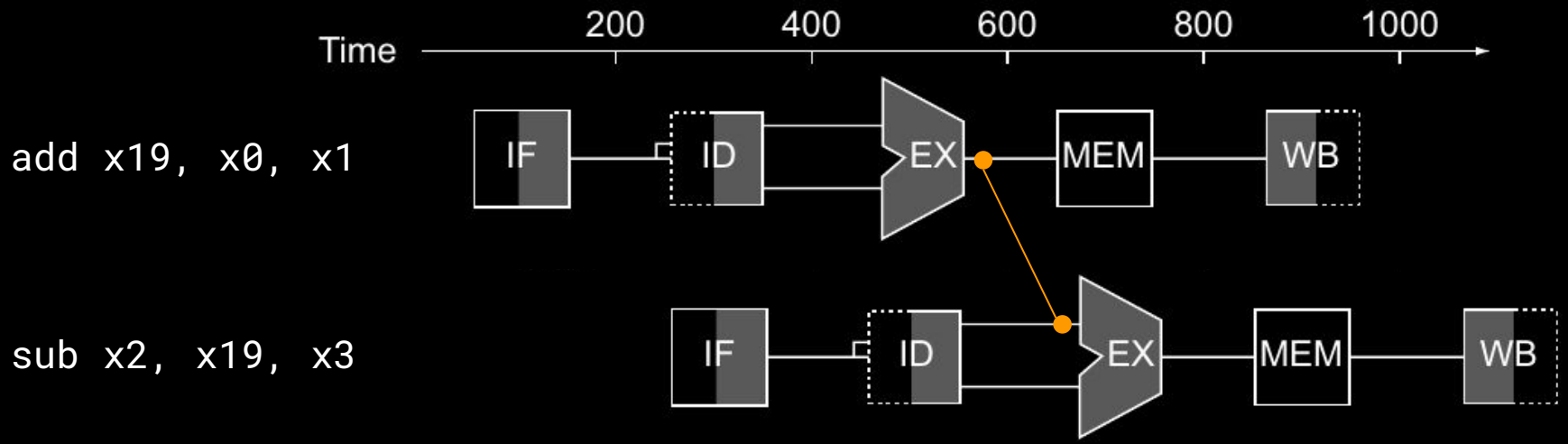
O resultado já está pronto após a unidade EX, mas não foi escrito no registrador ainda.

Tomar esse resultado, e utilizar como entrada para o EX da próxima instrução.

Vai exigir um controle mais complexo da CPU.

Necessário decidir se a entrada da ALU vem do banco de registradores, ou do resultado atual da ALU.

Hazard de Dados



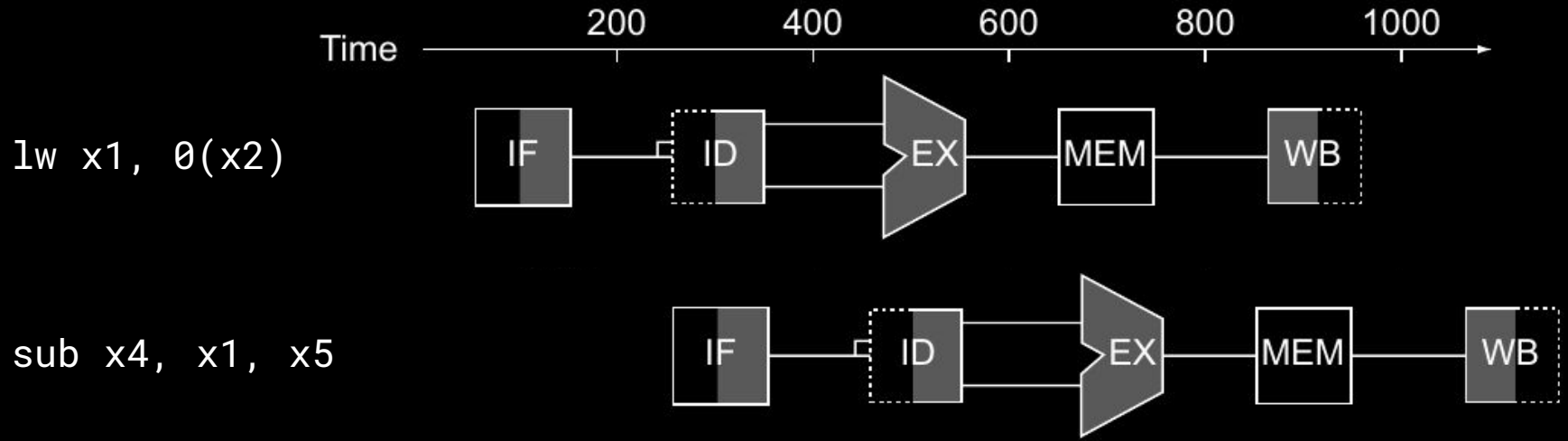
Faça você mesmo

Crie uma sequência de instruções onde o forwarding **não** vai resolver o problema.

Faça você mesmo

Crie uma sequência de instruções onde o forwarding **não** vai resolver o problema.

```
lw x1, 0(x2)
sub x4, x1, x5
```

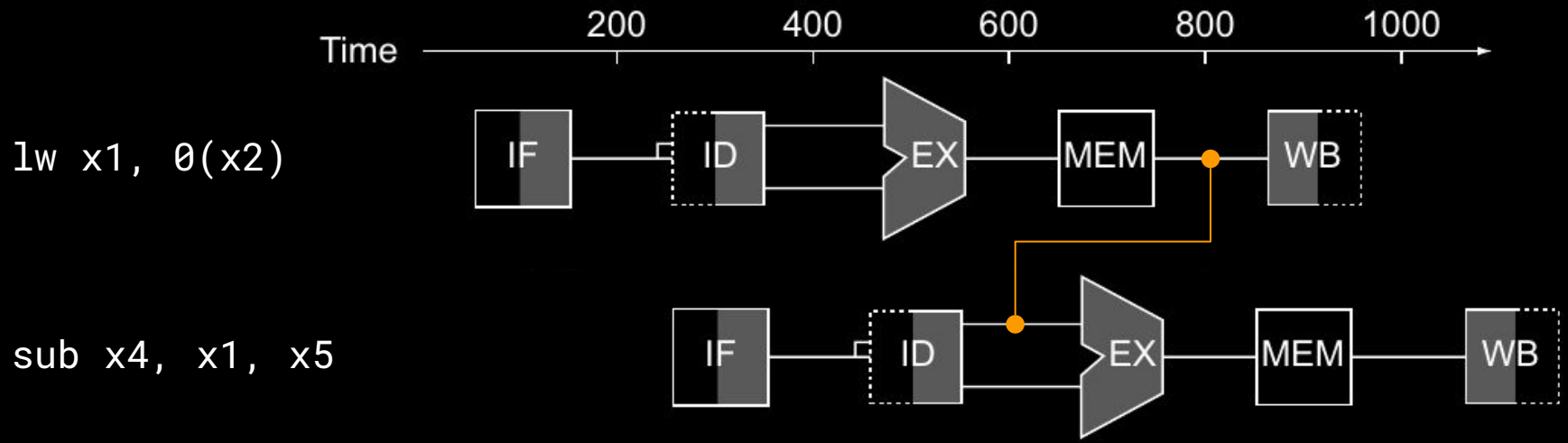


Faça você mesmo

Crie uma sequência de instruções onde o forwarding **não** vai resolver o problema.

```
lw x1, 0(x2)
sub x4, x1, x5
```

Voltar no tempo!?



Pipeline Stall

Um hazard de dados pode exigir o atraso da execução da próxima instrução.

O processador deve injetar uma **instrução bolha** entre as instruções que estão com o hazard.

A instrução bolha **não faz operação alguma**.

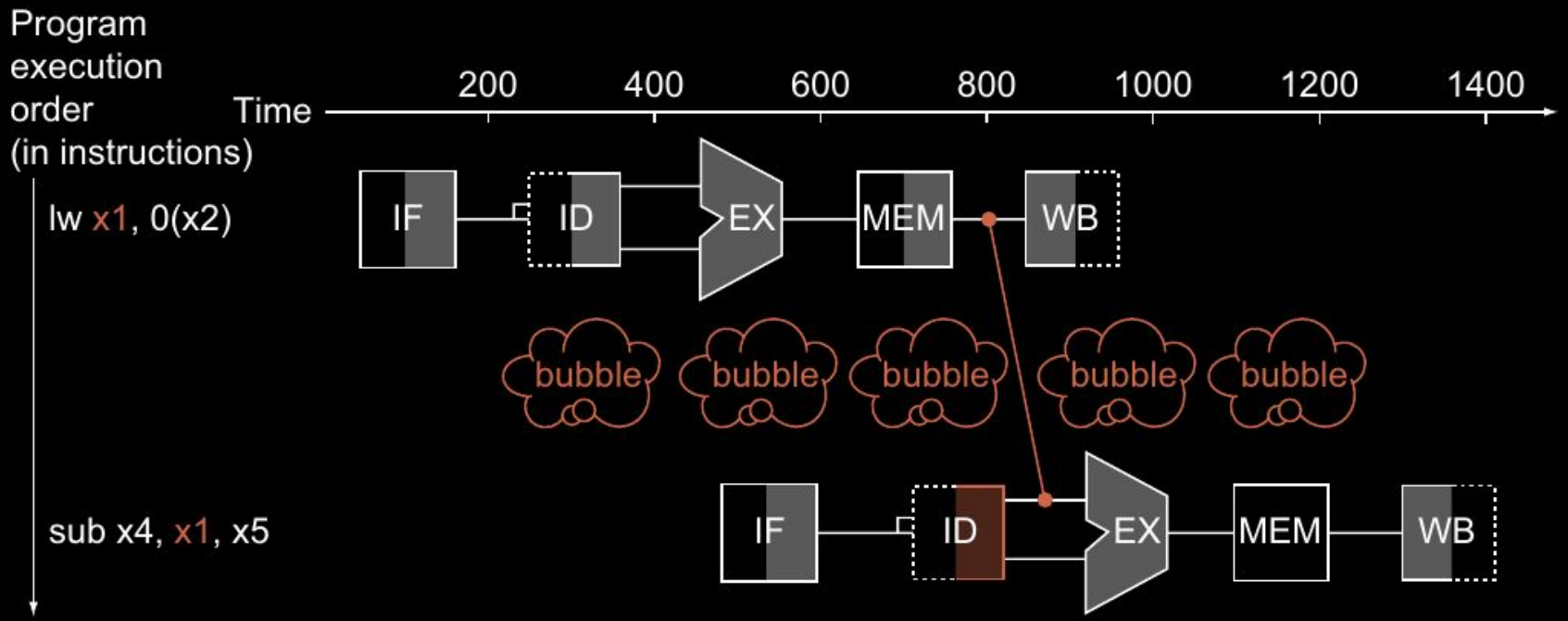
Não altera o estado dos registradores e não altera a memória de dados.

Apenas gasta tempo, para que o resultado necessário possa ser computado.

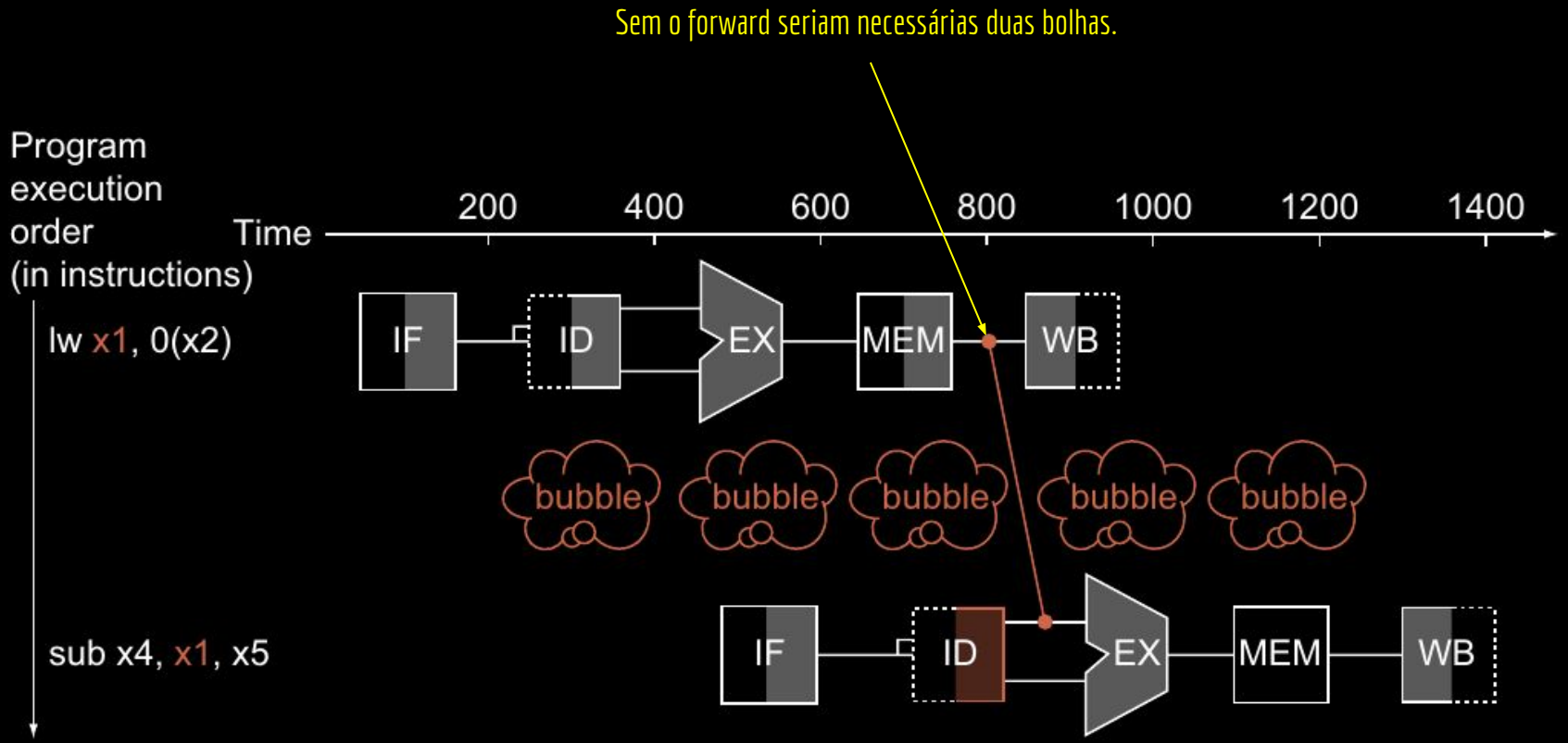
Pipeline **stall**.

Parar a “linha de montagem” por um momento para sincronizar as coisas.

Pipeline Stall



Pipeline Stall



O papel do compilador

O que o compilador ou programador podem fazer para tentar evitar um stall?

```
add x28, x29, x30
lw x31, 20(x27)
sub x5, x31, x6
```

O papel do compilador

O que o compilador ou programador podem fazer para tentar evitar um stall?

```
add x28, x29, x30
lw x31, 20(x27)
sub x5, x31, x6
```

O compilador pode reordenar as instruções ao gerar o código de máquina (ou assembly).

Nesse caso isso não afeta o resultado do programa, mas evita um stall.

```
lw x31, 20(x27)
add x28, x29, x30
sub x5, x31, x6
```

O papel do compilador

Compiladores sofisticados, como o GCC, tentam de todo modo criar uma ordem nas instruções que evita stalls.

Buffers de Reordenação

CPUs complexas, como as x86-64 atuais, contam ainda com circuitos especializados para reordenar o buffer de instruções.

A CPU pode, por conta própria, executar as instruções em uma ordem diferente para evitar stalls.

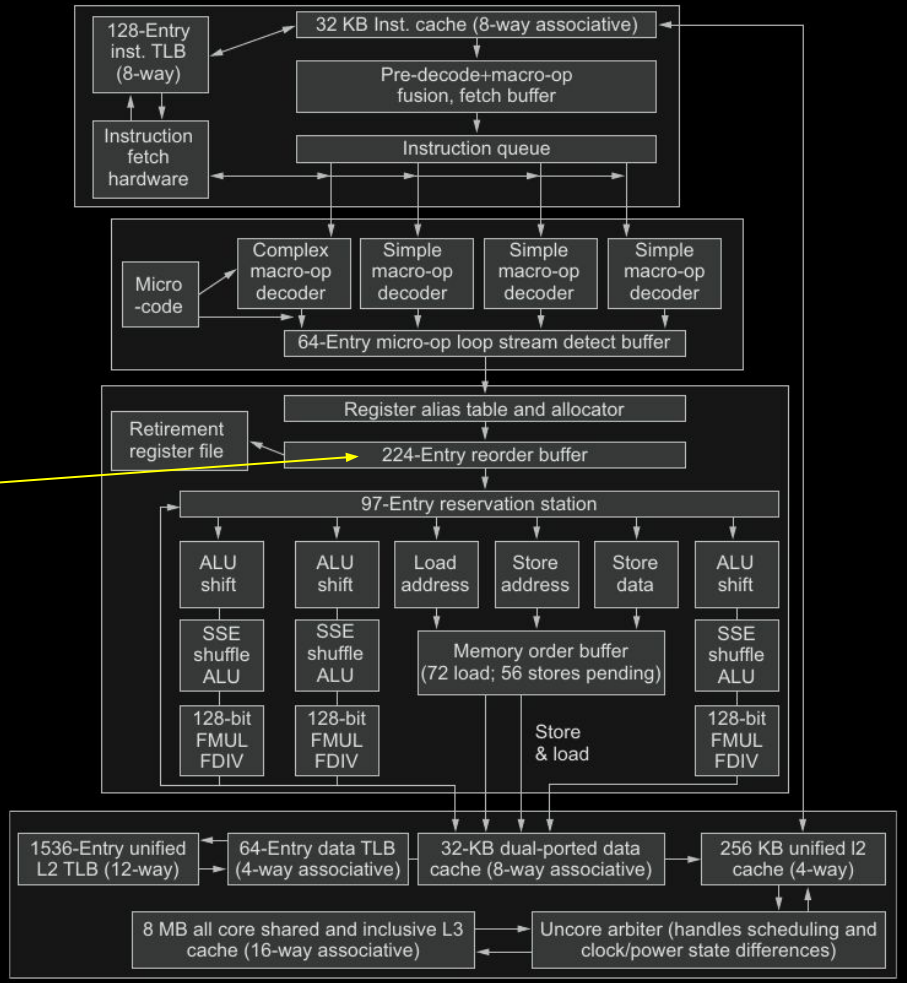
Utiliza técnicas como buffers de reordenação e renomeação de registradores.

Complexo! Veremos no decorrer da disciplina.

Exemplo

Estrutura de um Intel i7.

Buffer de reordenação de instruções para mitigar Stalls.



Patterson, Henessy, 2020.

Hazards de Controle

O pipeline funciona encadeando as instruções, uma após a outra, em uma “linha de montagem”.

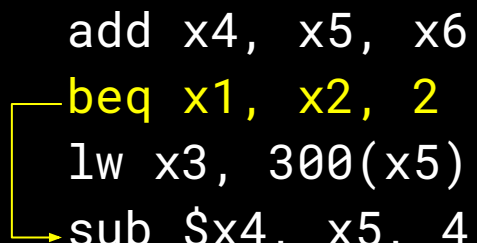
Mas e se não sabemos qual é a próxima instrução?

Em que cenário isso pode acontecer?

Hazards de Controle

O pipeline funciona encadeando as instruções, uma após a outra, em uma “linha de montagem”.

Mas e se não sabemos qual é a próxima instrução?



Hazards de Controle

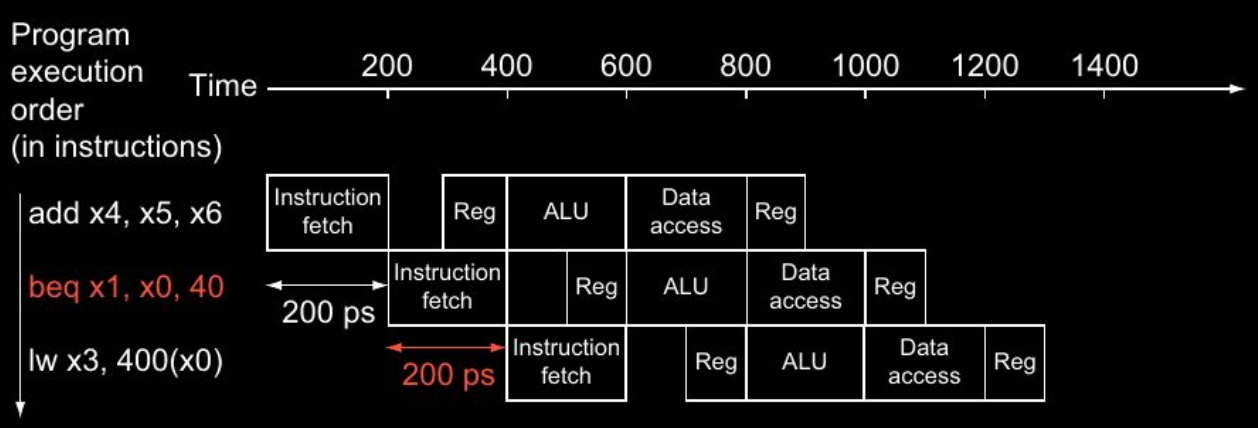
Solução simples - sempre **considerar que o desvio não foi tomado.**

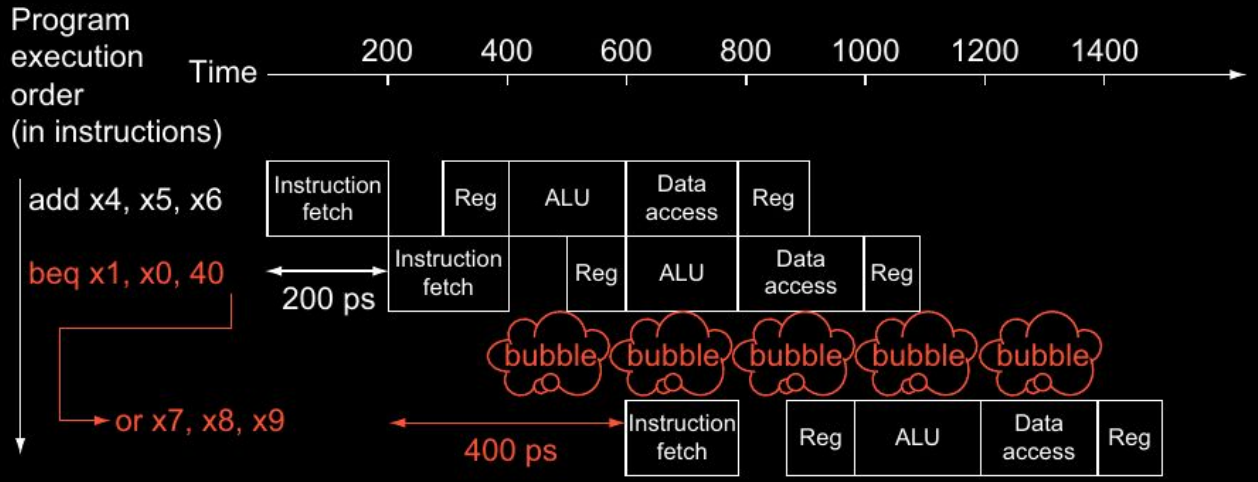
Quando a instrução de desvio terminar de executar:

- Se o desvio realmente não devia ser tomado, tudo continua normalmente.
- Se descobrirmos que o desvio deveria ser tomado.
  - **Descartar** as instruções que estão no pipeline, e carregar as instruções a partir do endereço correto.
  - **Injetar bolhas no meio do caminho!**

Exemplo

```
add x4, x5, x6
beq x1, x2, 2
lw x3, 300(x0)
or x7, x8, x9
```





Hazards de Controle

O custo de uma previsão incorreta pode ser excessivamente alto em uma CPU de pipeline profundo.

Considere o custo de uma previsão de branch incorreta em um Pentium 4 Prescott!

Várias instruções podem estar no pipeline.

Serão descartadas!

Note que depois do Pentium 4, o número de estágios no pipeline reduziu.

Um pipeline profundo é ideal se conseguimos mantê-lo cheio.

Mas é complexo, e stalls custam caro.

Difícil manter o pipeline sempre cheio.

Microprocessador	Ano	Clock	Estágios Pipeline	Tamanho Despacho	Fora de ordem?	CPUs por Chip	Potência
486	1989	0,025 GHz	5	1	Não	1	5 W
Pentium	1993	0,066 GHz	5	2	Não	1	10 W
Pentium Pro	1997	0,2 GHz	10	3	Sim	1	29 W
Pentium 4 Willamette	2001	2 GHz	22	3	Sim	1	75 W
Pentium 4 Prescott	2004	3,6 GHz	31	3	Sim	1	103 W
Intel Core	2006	3 GHz	14	4	Sim	2	75 W
Core i7 Nehalem	2008	3,6 GHz	14	4	Sim	2-4	87 W
Core Westmere	2010	3,73 GHz	14	4	Sim	6	130 W
Core i7 Ivy Bridge	2012	3,4 GHz	14	4	Sim	6	130 W
Core Broadwell	2014	3,7 GHz	14	4	Sim	10	140 W
Core i9 Skylake	2016	3,1 GHz	14	4	Sim	14	165 W
Intel Ice Lake	2018	4,2 GHz	14	4	Sim	16	185 W
Intel Raptor Cove	2022	6.2 GHz	12	-	Sim	24	-
Intel Lion Cove	2024	-	10	-	Sim	-	-

Hazards de Controle

Podemos melhorar através de um sistema que tenta aprender se os desvios estão sendo tomados ou não.

Buffer de Previsão de Desvios

Um **buffer de previsão de desvios** é uma pequena memória, que contém uma **tabela com o endereço da instrução**, e um **bit indicando se o desvio foi tomado ou não a última vez** que executamos a instrução nesse endereço.

Especialmente útil em loops.

Depois de calcular se o endereço realmente foi tomado ou não, podemos **atualizar o valor no buffer**.

```
for(int i=0; i < 10; ++i){
    //faz algo
}
```

Buffer de Previsão de Desvios

O buffer é pequeno, e obviamente não podemos armazenar o endereço de todas as instruções.

Solução: utilizar os bits mais baixos do endereço de memória para endereçar o buffer.

Muitas instruções vão compartilhar o mesmo local no buffer.

Pode acontecer de verificarmos o buffer para uma instrução, mas o bit se refere a alguma outra.

Ideia **similar** a de uma memória cache.

Exemplo – Buffer com capacidade para 8 instruções

Endereço (binário)	Instrução
… 0000 0000 0000 0000	Instrução 1
… 0000 0000 0000 0001	~~Instrução 2~~
… 0000 0000 0000 0010	Instrução 3
… 0000 0000 0000 0011	Instrução 4
… 0000 0000 0000 0100	Instrução 5
… 0000 0000 0000 0101	Instrução 6
… 0000 0000 0000 0110	Instrução 7
… 0000 0000 0000 0111	Instrução 8
… 0000 0000 0000 1000	Instrução 9
… 0000 0000 0000 1001	~~Instrução 10~~
… 0000 0000 0000 1010	Instrução 11
… 0000 0000 0000 1011	Instrução 12
…	…

Instruções competindo pelo mesmo lugar no buffer.

Buffer	
Endereço	Desviar?
000	1
001	0
010	0
011	1
100	1
101	0
110	1
111	0

Melhorando

Quais os erros que o buffer de predição vai cometer com a estrutura a seguir?

```
for(int i=0; i < 10; i++){
    for(int j=0; j < 10; j++){
        //faz algo
    }
}
```

Melhorando

Quais os erros que o buffer de predição vai cometer com a estrutura a seguir?

```
for(int i=0; i < 10; i++){
    for(int j=0; j < 10; j++){
        //faz algo
    }
}
```

Toda vez que esse loop acaba, o preditor erra. A tabela é atualizada, informando que o desvio foi tomado.

Mas o loop interno é executado novamente por conta do loop externo, e quando entrar no loop interno novamente, a predição na tabela estará incorreta.

Preditor de 2 bits

Utilizar 2 bits no buffer.

A predição muda quando há uma confirmação na mudança do comportamento.

Utilização de uma máquina de estados simples.

Buffer	
Endereço	Desviar?
000	01
001	00
010	00
011	11
100	11
101	00
110	10
111	01

Outros esquemas de previsão

Buffers de previsão que utilizam mais de 2 bits.

Delayed Slots.

Pipelines rasos e com decisões sobre desvios tomadas no início do pipeline.

Um bom exemplo é o processador RISC-V sendo analisado em aula.

Obs.: Processadores RISC-V não usam delayed slots, mas processadores MIPS às vezes, sim.

Preditores de correlação.

Combinam buffers de predição com informação global sobre os comportamentos dos branches do programa.

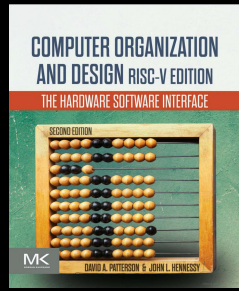
Preditor de torneio.

Usam múltiplos preditores que “competem” entre si. O preditor de torneio escolhe o preditor com o melhor resultado no momento.

Outros esquemas de previsão

Esquemas atuais conseguem uma taxa de acertos de cerca de 90% (Patterson, Henessy; 2020).

Patterson, Hennessy. Computer Organization and Design RISC-V Edition: The Hardware Software Interface. 2020.



Exercícios

1. Foi demonstrado que uma das entradas da unidade EX pode vir emprestada da etapa EX anterior para as instruções:

   ```
   add s0, t0, t1
   sub t2, s0, t3
   ```

   Existe a possibilidade das duas entradas EX precisarem de um forwarding? Se sim, faça um código em Assembly que ilustra esse cenário, e desenhe o gráfico com os forwardings.
2. Para as sequências a seguir, indique se acontecerá um stall, se stalls podem ser evitados via forwarding, ou se a execução não gera stalls e não requer forwardings.

```
lw  x5,0(x5)
add  x6,x5,x5
```

```
add  x6,x5,x5
addi  x7,x5,5
addi  x9,x6,5
```

```
addi  x6,x5,1
addi  x7,x5,2
addi  x8,x5,2
addi  x8,x5,4
addi  x10,x5,5
```

Exercícios

3. Considerando que o buffer de predição de desvios começa com todas suas entradas em 0, significando que o desvio não deve ser tomado, e as instruções dos trechos a seguir cabem no buffer sem colisões de endereço (não existem duas instruções competindo pela mesma entrada na tabela), qual a taxa de acertos para um buffer de predição de 1 bit, e 2 bits, para os seguintes trechos?

a)
```
for(int i=0; i < 100; i++){
    //faz algo
}
```

b)
```
for(int i=0; i < 10; i++){
    for(int j=0; j < 10; j++){
        //faz algo
    }
}
```

Exercícios

4. Assuma que as variáveis do trecho programa em C a seguir são do tipo inteiro e ocupam 4 bytes. A variável *a* está no endereço apontado por x31. A variável *b*, está no endereço apontado por x31 somado com 4. A variável *c*, está no endereço apontado por x31 somado com 8...

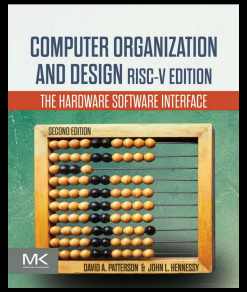
```
a = b + e;
c = b + f;
```

a. Traduza o trecho para assembly do RISC-V
b. Reordene as instruções de (a) para evitar stalls.

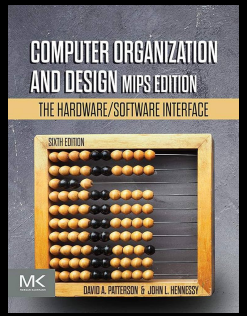
Resposta no livro base da disciplina.

Referências

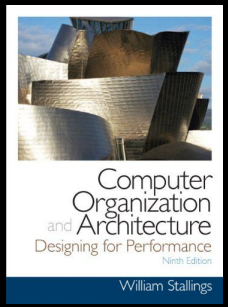
Patterson, Hennessy. Computer Organization and Design RISC-V Edition: The Hardware Software Interface. 2020.



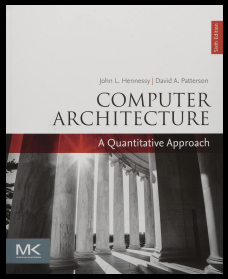
Patterson, Hennessy. Computer Organization and Design MIPS Edition: The Hardware/Software Interface. 2020.



Stallings, W. Organização de Arquitetura de Computadores. 10a Ed. 2016.



Hennessy, Patterson. Arquitetura de Computadores: uma abordagem quantitativa. 2019.



Licença