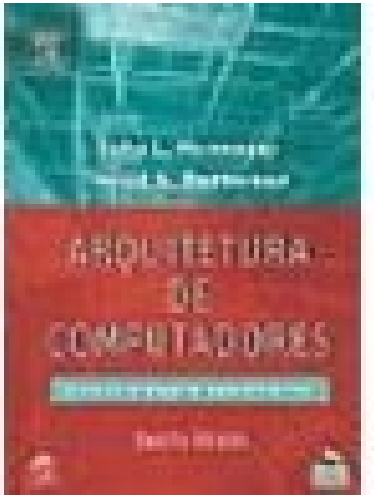
Instituto de Matemática
Departamento de Ciência da Computação

MATA48
Arquitetura de Computadores

Pipelining

Prof. Marcos E. Barreto

Tópicos

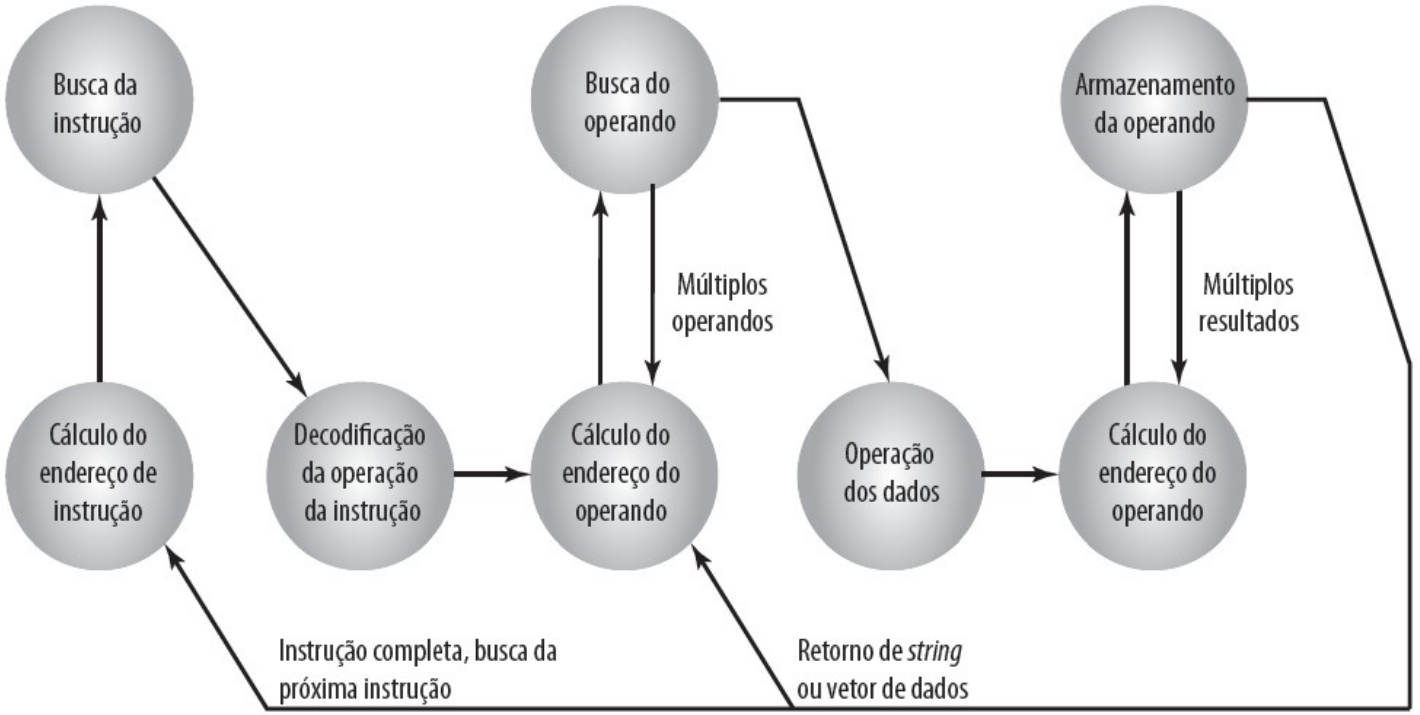


- Contextualização
- Desempenho do pipelining
- Penalidades (*hazards*) de pipelining

- Referências:
  - PATTERSON, D; HENNESSY, J. Organização e projeto de computadores – a interface hardware/software. 3 ed. 2005 – cap 6.
  - HENNESSY, J.; PATTERSON, D.; Arquitetura de computadores – uma abordagem quantitativa. 4 ed. 2008 – apêndice A.

Contextualização

- **Pipelining**
  - Técnica de implementação pela qual várias instruções são sobrepostas em execução.
  - Tira proveito do paralelismo existente entre as etapas do ciclo de instrução.



Contextualização (2)

Pipelining RISC (*Reduced Instruction Set Computers*)

Cada instrução utiliza no máximo 5 ciclos de clock.

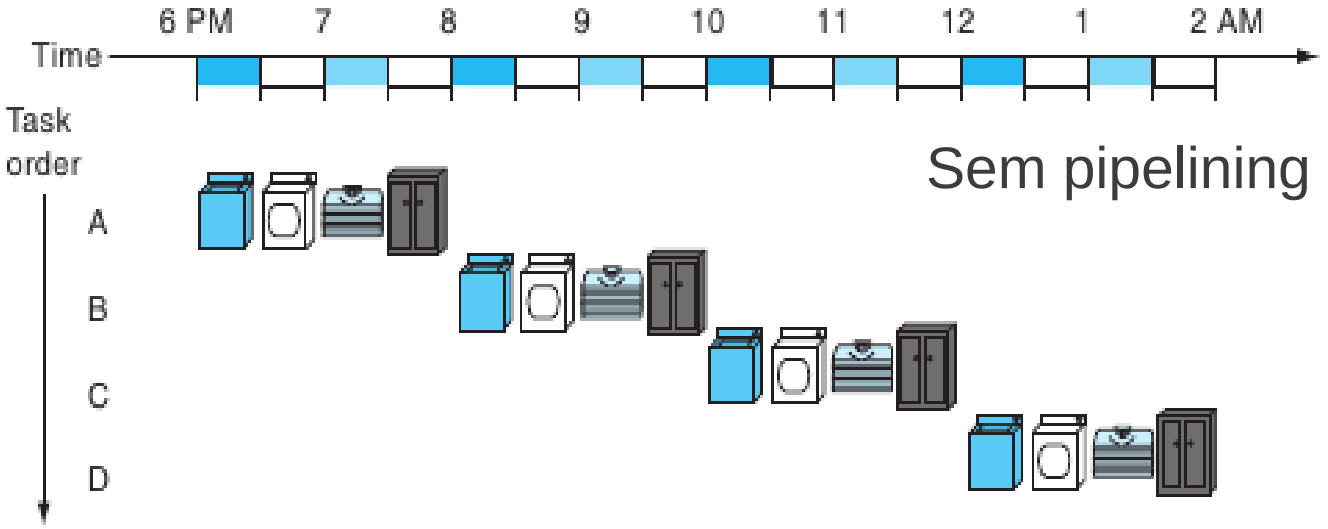
	Clock number								
Instruction number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Instruction *i*	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	ID	EX	MEM	WB			
Instruction *i* + 2			IF	ID	EX	MEM	WB		
Instruction *i* + 3				IF	ID	EX	MEM	WB	
Instruction *i* + 4					IF	ID	EX	MEM	WB

Figure A.1 Simple RISC pipeline. On each clock cycle, another instruction is fetched and begins its 5-cycle execution. If an instruction is started every clock cycle, the performance will be up to five times that of a processor that is not pipelined. The names for the stages in the pipeline are the same as those used for the cycles in the unpipelined implementation: IF = instruction fetch, ID = instruction decode, EX = execution, MEM = memory access, and WB = write back.

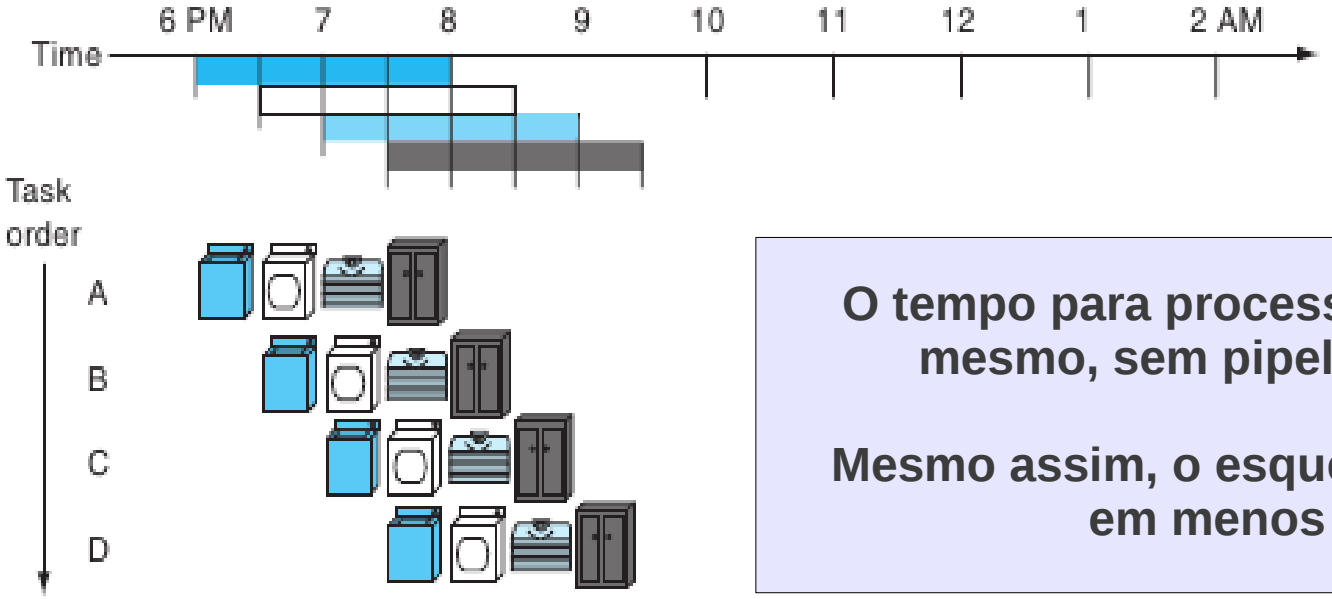
Contextualização (3)

- **Pipelining**
  - Analogia: lavanderia.

Quatro estágios:
1. lavar
2. secar
3. passar
4. guardar



Sem pipelining



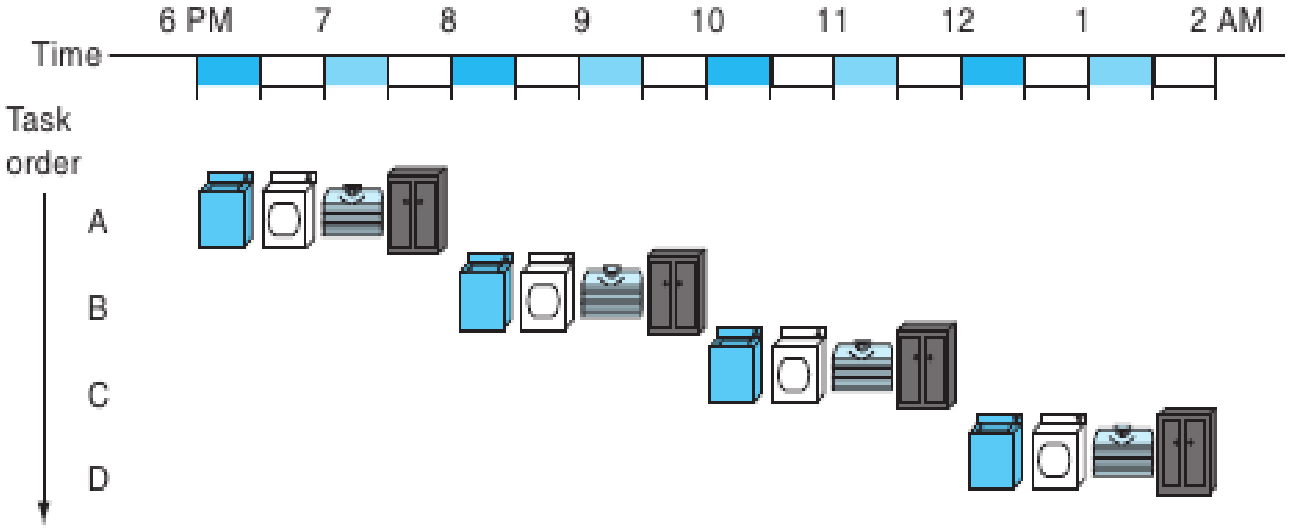
Com pipelining

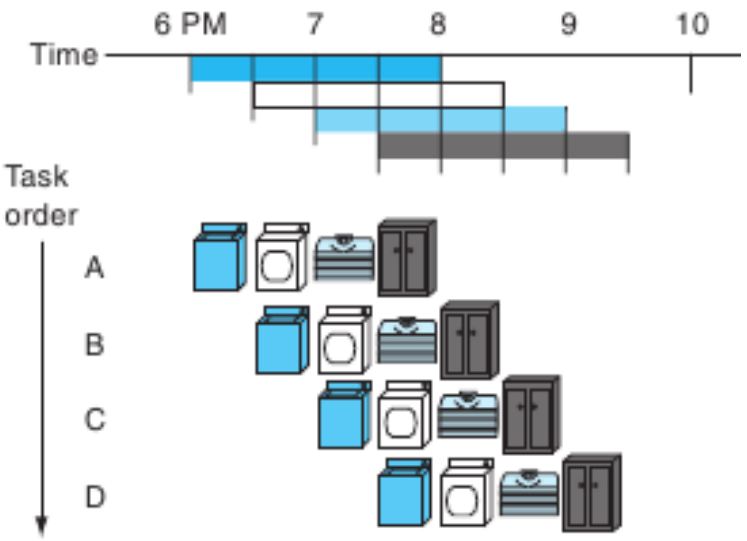
O tempo para processar uma trouxa de roupa é o mesmo, sem pipelining ou com pipelining.

Mesmo assim, o esquema com pipelining termina em menos tempo. Por quê?

Desempenho do pipelining (1)

- O pipelining é mais rápido porque muitas trouxas de roupa são processadas dentro da mesma unidade de tempo (ex. hora).
- O **pipelining melhora a vazão do sistema**, sem melhorar o tempo para que uma única trouxa de roupa seja processada.
- No exemplo: 4 trouxas (com pipelining) levam o tempo de 2 trouxas (sem pipelining).
- Se todos os estágios do pipelining tiverem o mesmo tempo de execução e estiverem todos ocupados, então o ganho de vazão será igual ao número de estágios do pipelining.





Desempenho do pipelining (2)

- A vazão (*throughput*) do pipelining de instruções é determinada pela frequência com que uma instrução sai do pipelining.

- Ciclo do processador: tempo gasto para avançar uma instrução um estágio no pipelining.
  - Determinado pelo estágio mais lento.
  - Geralmente, ciclo de processador = ciclo de clock.

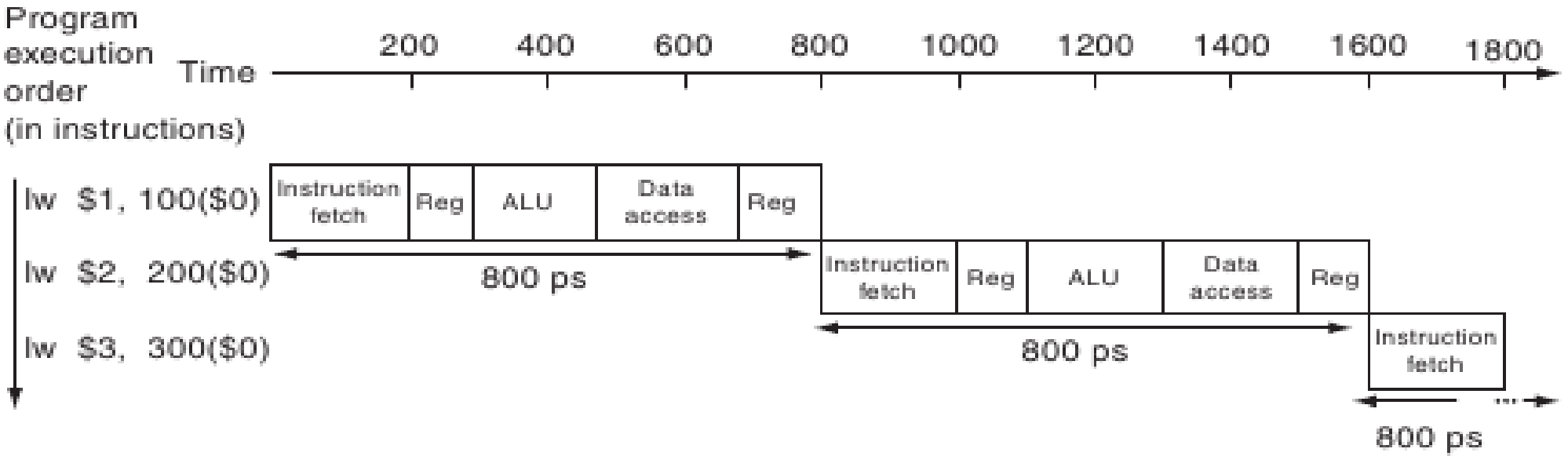
- Objetivo: balancear o tempo de cada estágio do pipelining.

$$Tempo\ entre\ instruções\left(com\ pipelining\right)=\frac{\left(Tempo\ entre\ instruções\left(sem\ pipelining\right)\right)}{\left(Número\ de\ estágios\ do\ pipelining\right)}$$

Desempenho do pipelining (3)

- **Exemplo de desempenho sem pipelining X com pipelining**
  - Sem pipelining: cada instrução leva um ciclo de clock, “esticado” para suportar a instrução mais lenta => lw (800 ps).

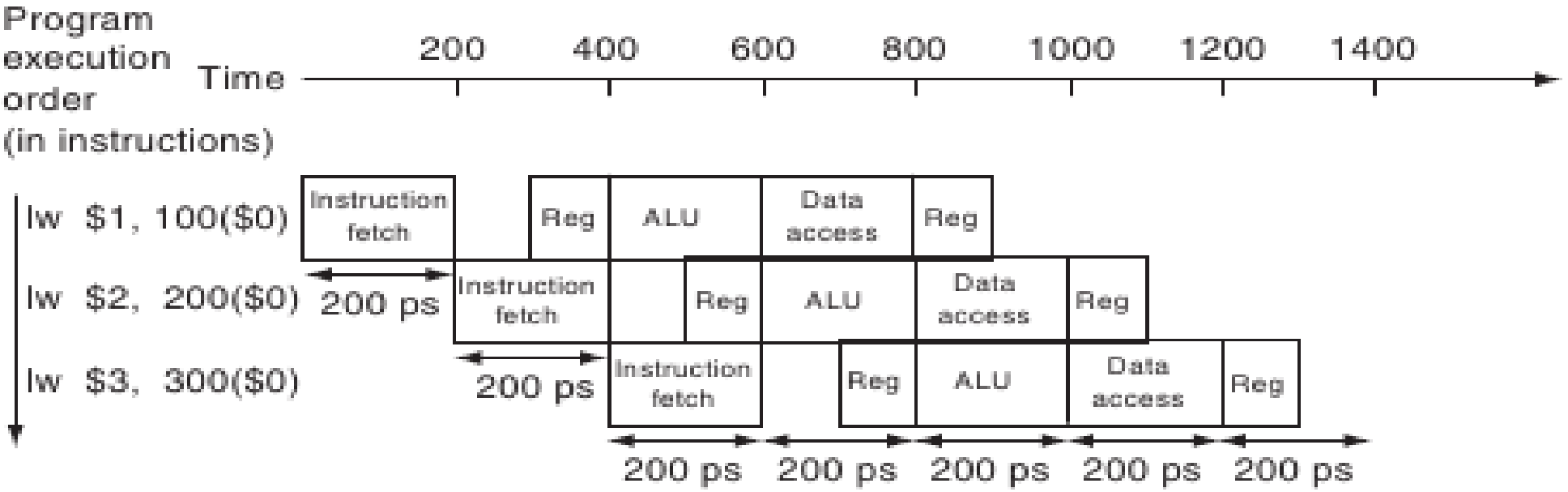
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps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



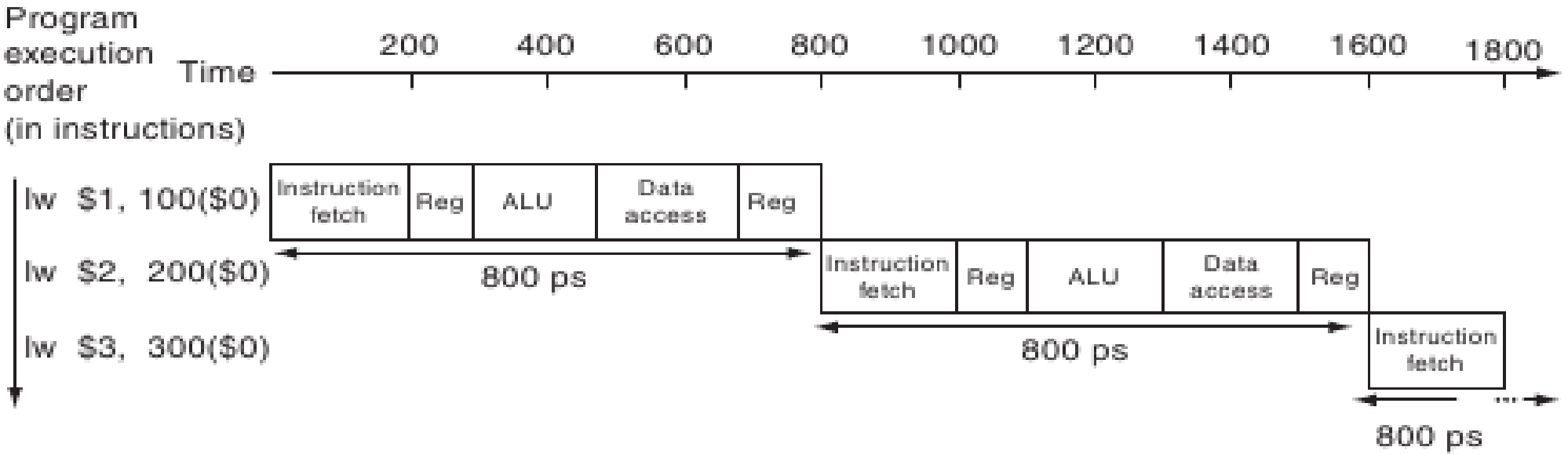
Desempenho do pipelining (4)

- **Exemplo de desempenho sem pipelining X com pipelining**
  - Com pipelining: ciclo de clock deve suportar a instrução mais lenta => 200 ps.

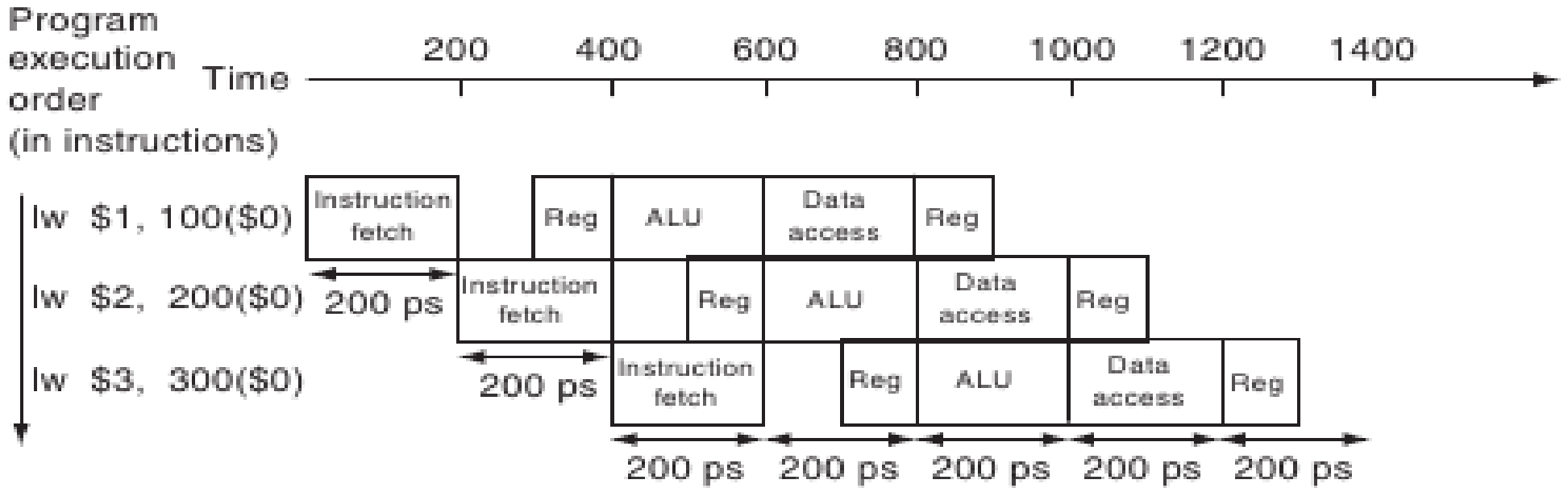
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps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



Desempenho do pipelining (5)



Qual é o ganho com o pipelining?



Desempenho do pipelining (6)

- **Exemplo de desempenho sem pipelining X com pipelining**

Processador sem pipelining:

- ciclo de clock de 1 ns; 4 ciclos para operações da ULA e desvios, com frequências de 40% e 20%, respectivamente; 5 ciclos para operações com memória, com frequência de 40%.

Tempo médio de execução da instrução = ciclo de clock x CPI médio
1 ns x ((40% + 20%) x 4 + 40% x 5)
1 ns x 4,4 => 4,4 ns

Processador com pipelining

- Sobrecarga do pipelining = 0,2 ns.
- Velocidade do estágio mais lento + sobrecarga => 1 ns + 0,2 ns

$$Ganho\ de\ velocidade = \frac{(tempo\ médio\ de\ instrução\ sem\ pipeline)}{(tempo\ médio\ de\ instrução\ com\ pipeline)} = \frac{(4{,}4\,ns)}{(1{,}2\,ns)} = 3{,}7\ vezes$$

Pipelining – mais detalhes

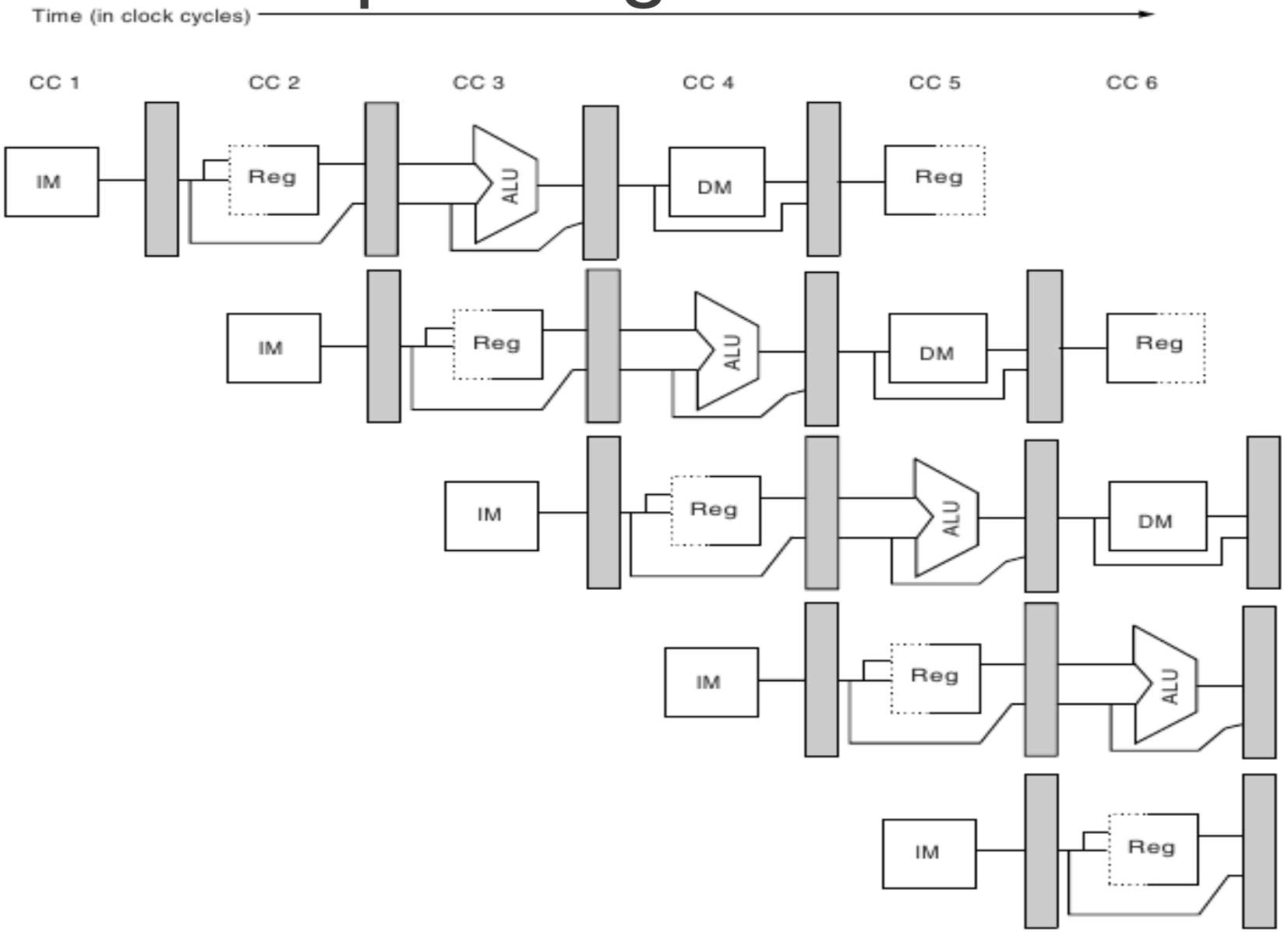


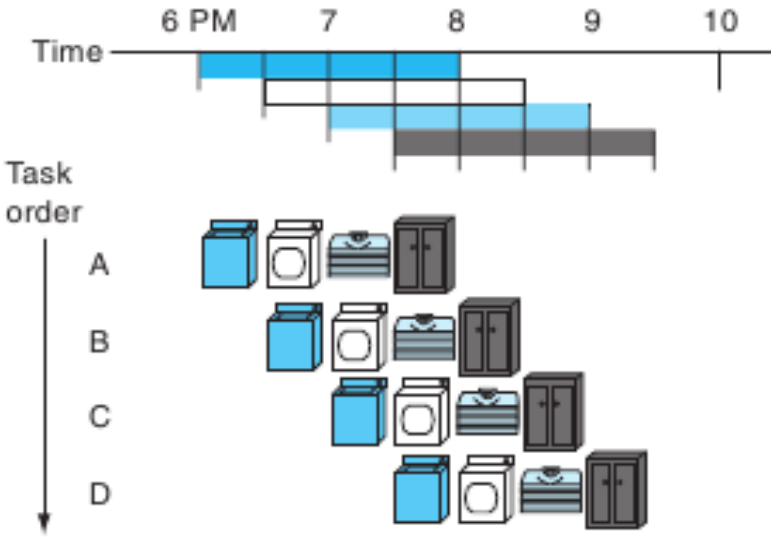
Figure A.3 A pipeline showing the pipeline registers between successive pipeline stages. Notice that the registers prevent interference between two different instructions in adjacent stages in the pipeline. The registers also play the critical role of carrying data for a given instruction from one stage to the other. The edge-triggered property of registers—that is, that the values change instantaneously on a clock edge—is critical. Otherwise, the data from one instruction could interfere with the execution of another!

Penalidades do pipelining (1)

- Existem situações em que a próxima instrução não pode ser executada no ciclo de clock seguinte.
- Essas situações são chamadas **penalidades** (*hazards*),
- 3 tipos:
  - **Penalidades estruturais**
  - **Penalidades de dados**
  - **Penalidades de controle (ou de desvio)**

Penalidades do pipelining (2)

- **Penalidades estruturais**
  - O hardware não pode admitir a combinação de instruções que dependam de um mesmo recurso do hardware no mesmo ciclo de clock.
  - Por exemplo: uma combinação lavadora-secadora em vez de lavadora separada da secadora.



- **Penalidades estruturais**

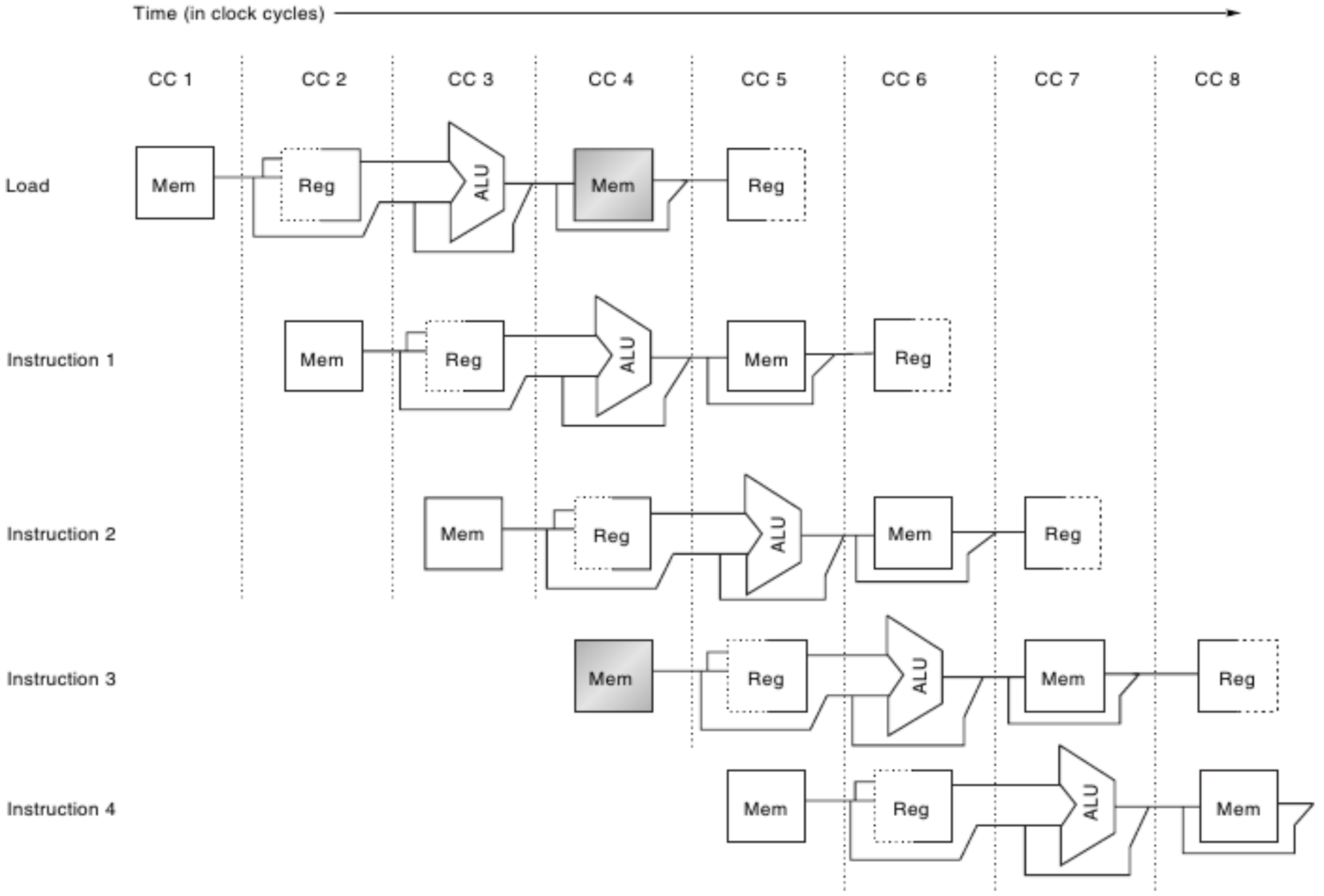


Figure A.4 A processor with only one memory port will generate a conflict whenever a memory reference occurs. In this example the load instruction uses the memory for a data access at the same time instruction 3 wants to fetch an instruction from memory.

Penalidades do pipelining (4)

- **Penalidades estruturais**

	Clock cycle number									
Instruction	1	2	3	4	5	6	7	8	9	10
Load instruction	IF	ID	EX	MEM	WB					
Instruction $i + 1$		IF	ID	EX	MEM	WB				
Instruction $i + 2$			IF	ID	EX	MEM	WB			
Instruction $i + 3$				stall	IF	ID	EX	MEM	WB	
Instruction $i + 4$						IF	ID	EX	MEM	WB
Instruction $i + 5$							IF	ID	EX	MEM
Instruction $i + 6$								IF	ID	EX

Figure A.5 A pipeline stalled for a structural hazard—a load with one memory port. As shown here, the load instruction effectively steals an instruction-fetch cycle, causing the pipeline to stall—no instruction is initiated on clock cycle 4 (which normally would initiate instruction $i + 3$). Because the instruction being fetched is stalled, all other instructions in the pipeline before the stalled instruction can proceed normally. The stall cycle will continue to pass through the pipeline, so that no instruction completes on clock cycle 8. Sometimes these pipeline diagrams are

Penalidades do pipelining (5)

- **Penalidades de dados**
  - Uma instrução depende de um dado de uma instrução anterior que ainda está no pipelining.
  - Exemplo: `add $s0, $t0, $t1; sub $t2, $s0, $t3`
  - A instrução `add` somente escreve o resultado no 5º ciclo, causando 3 ciclos ociosos (bolhas ou *stalls*) no pipelining.
  - Solução: **forwarding** ou **bypassing**

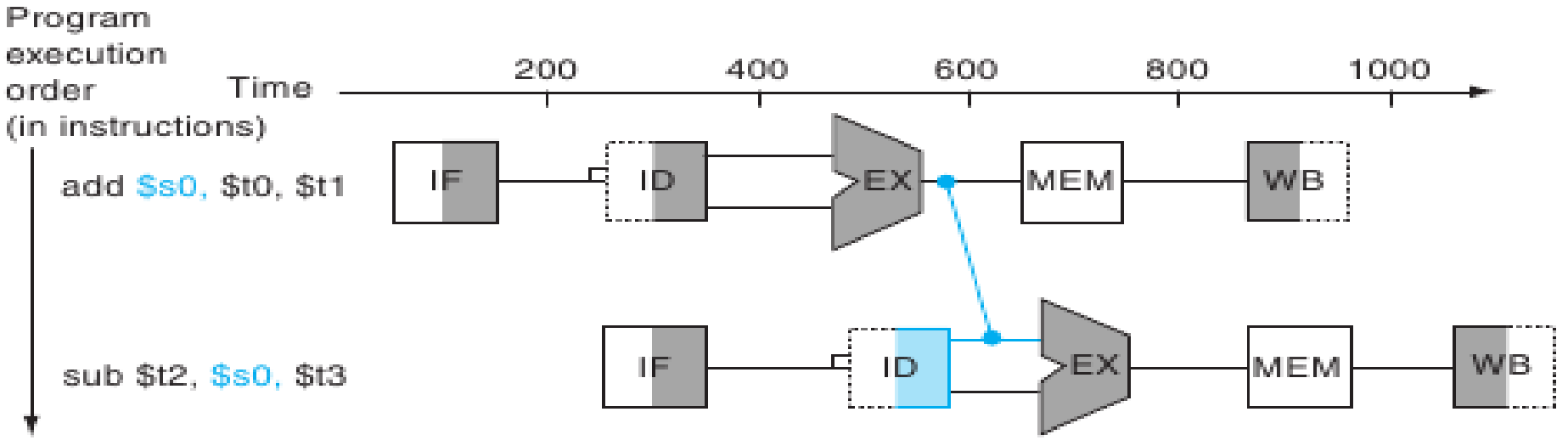
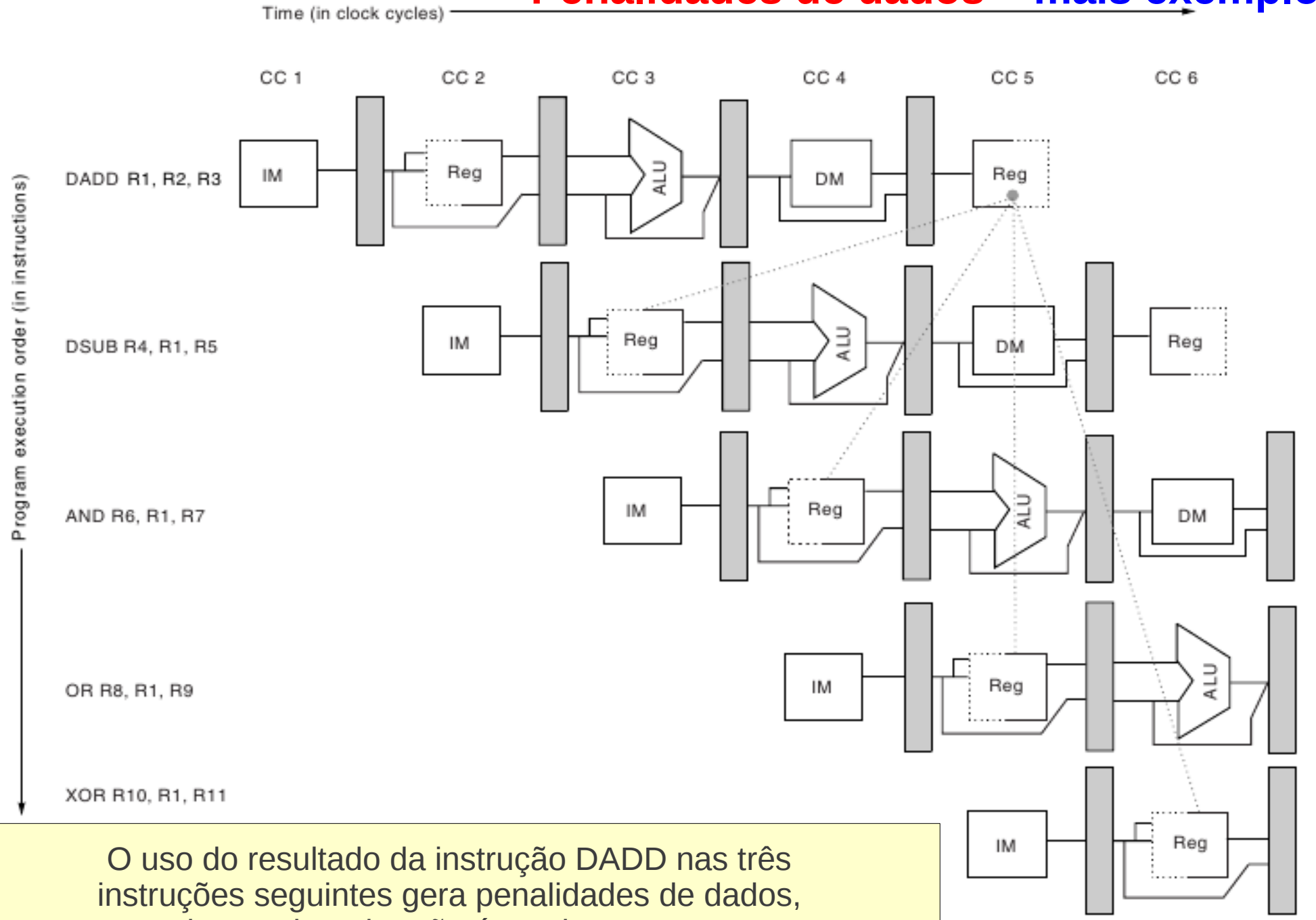


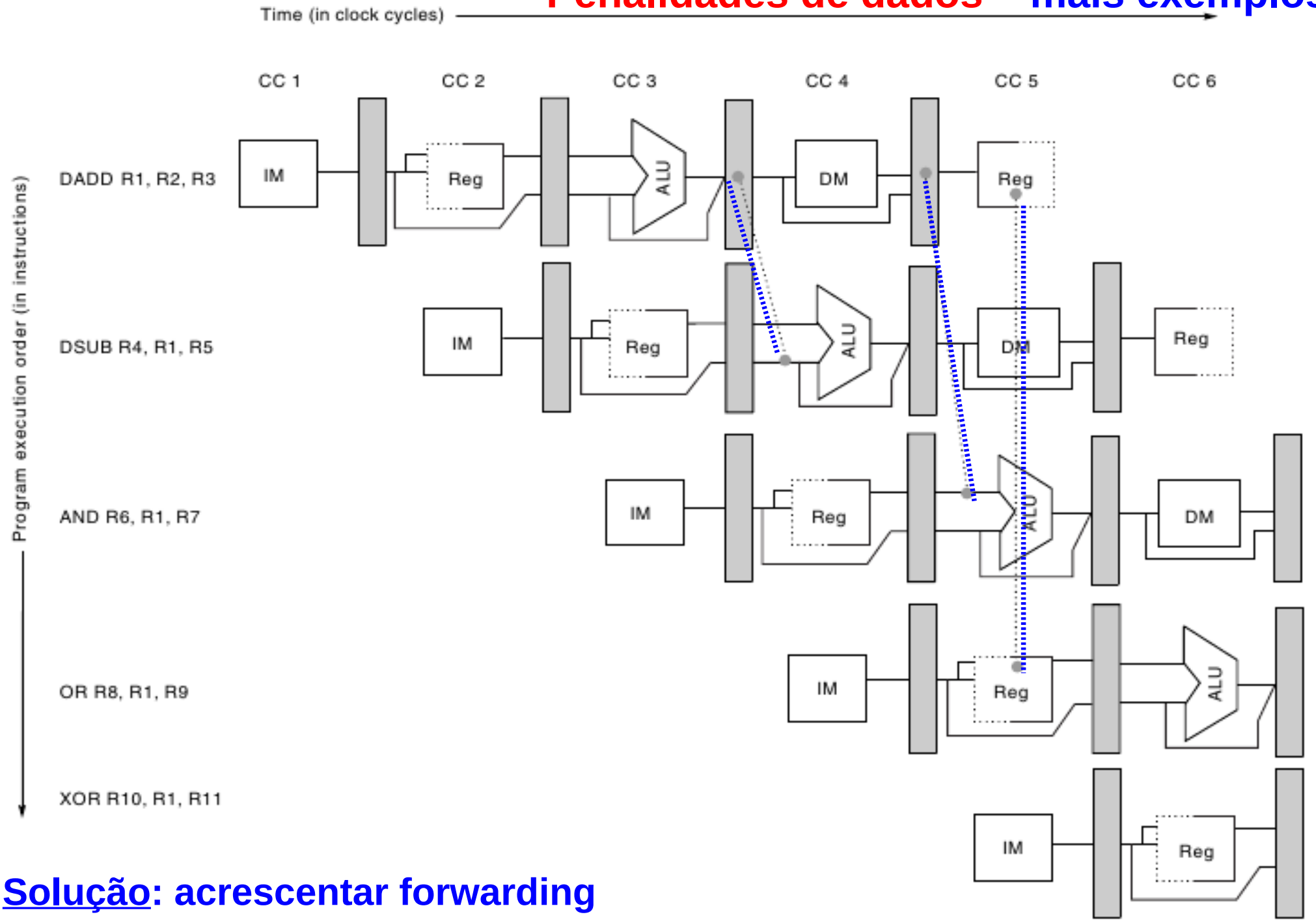
FIGURE 6.5 Graphical representation of forwarding. The connection shows the forwarding path from the output of the EX stage of `add` to the input of the EX stage for `sub`, replacing the value from register `$s0` read in the second stage of `sub`.

• Penalidades de dados – mais exemplos



O uso do resultado da instrução DADD nas três instruções seguintes gera penalidades de dados, pois o registrador não é escrito antes que estas instruções o leiam.

- **Penalidades de dados – mais exemplos**



Solução: acrescentar forwarding

Figure A.7 A set of instructions that depends on the DADD result uses forwarding paths to avoid the data hazard.

Penalidades do pipelining (8)

- **Penalidades de dados**
  - Exemplo: `lw $s0, 20($t1); sub $t2, $s0, $t3`
  - Resultado da instrução `load (lw)` só estará disponível após o 4º estágio; muito tarde para ser usado na entrada do 3º estágio da instrução `sub`.
  - Problema: **hazard de dados no uso de load**

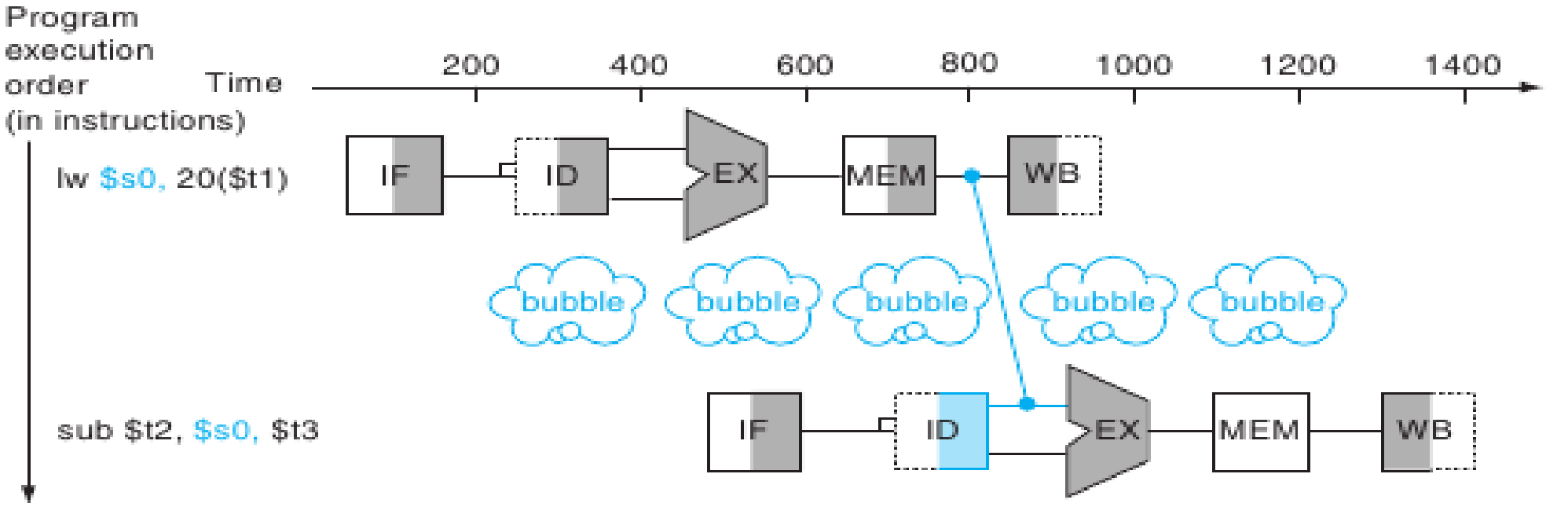


FIGURE 6.6 We need a stall even with forwarding when an R-format instruction following a load tries to use the data. Without the stall, the path from memory access stage output to execution stage input would be going backwards in time, which is impossible. This figure is actually a

- **Penalidades de dados** - **hazard de dados no uso de load**

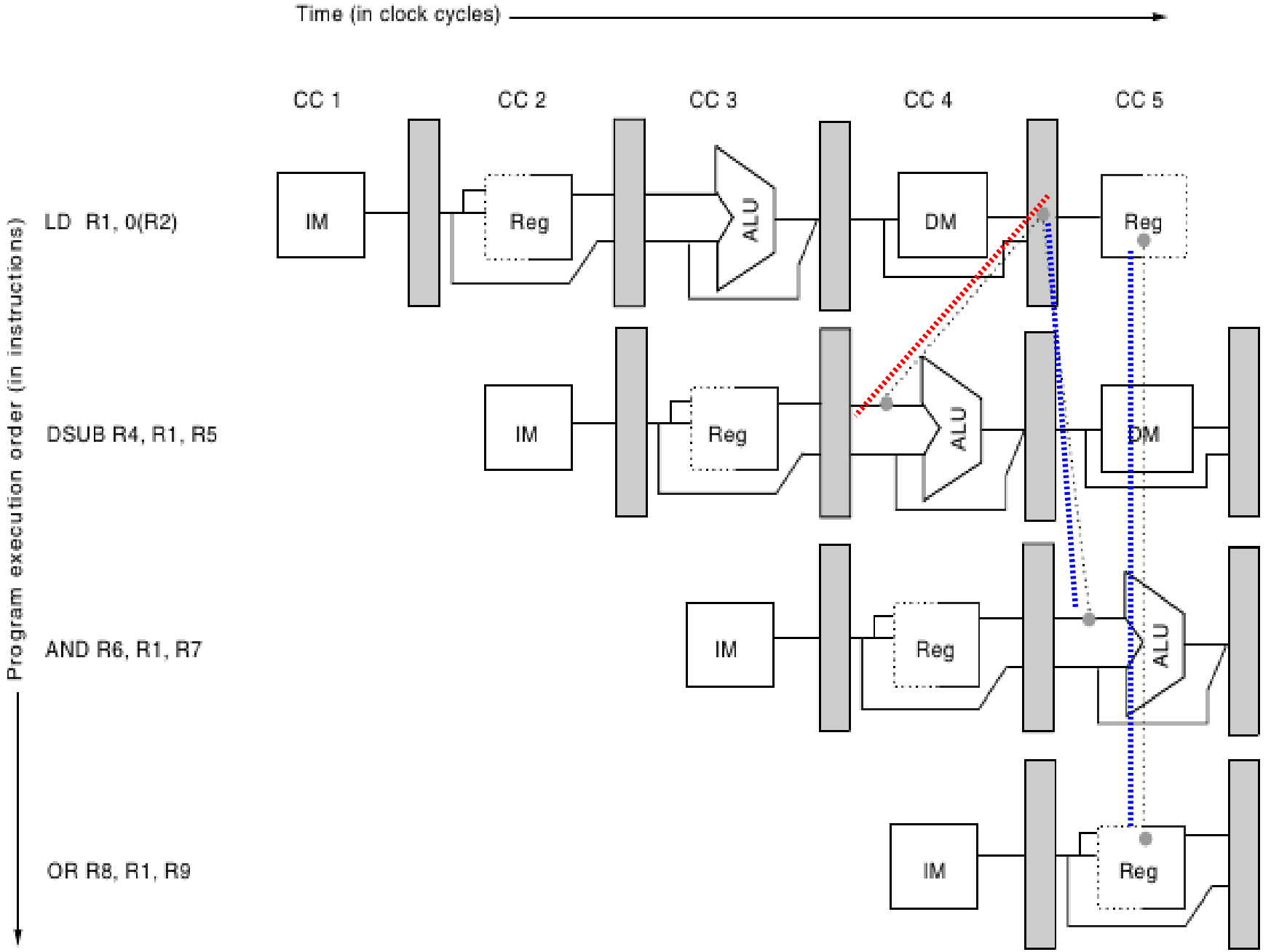


Figure A.9 The load instruction can bypass its results to the AND and OR instructions, but not to the DSUB, since that would mean forwarding the result in "negative time."

- **Penalidades de dados - hazard de dados no uso de load**

A instrução load tem um atraso de latência que não pode ser eliminado pelo forwarding.

Solução: acréscimo de hardware (*interlock pipelining*) para preservar a ordem na execução das instruções.

LD	R1,0(R2)	IF	ID	EX	MEM	WB				
DSUB	R4,R1,R5		IF	ID	EX	MEM	WB			
AND	R6,R1,R7			IF	ID	EX	MEM	WB		
OR	R8,R1,R9				IF	ID	EX	MEM	WB	

LD	R1,0(R2)	IF	ID	EX	MEM	WB				
DSUB	R4,R1,R5		IF	ID	stall	EX	MEM	WB		
AND	R6,R1,R7			IF	stall	ID	EX	MEM	WB	
OR	R8,R1,R9				stall	IF	ID	EX	MEM	WB

Figure A.10 **In the top half, we can see why a stall is needed: The MEM cycle of the load produces a value that is needed in the EX cycle of the DSUB, which occurs at the same time.** This problem is solved by inserting a stall, as shown in the bottom half.

Penalidades do pipelining (11)

- **Penalidades de controle (ou de desvio)**
  - Necessidade de tomar uma decisão com base no resultado de uma instrução que ainda não terminou de ser executada.
  - **Instruções de desvio**

```
I1. IF (X = 10)
I2. THEN Z := 100
I3. ELSE Z := 200;
```

  - Qual valor deve ser atribuído a Z (`I2` ou `I3`)?

O pipelining deve buscar a próxima instrução após o desvio (IF).
Porém, ele não sabe qual instrução deve buscar (THEN ou ELSE)!
Ele somente recebeu a instrução de desvio, mas ainda não a executou!
Logo, o resutado (X = 10) ainda não é conhecido.

Penalidades do pipelining (12)

- **Penalidades de controle (ou de desvio)**

Branch instruction	IF	ID	EX	MEM	WB	
Branch successor		IF	IF	ID	EX	MEM	WB
Branch successor + 1				IF	ID	EX	MEM
Branch successor + 2					IF	ID	EX

Figure A.11 A branch causes a 1-cycle stall in the five-stage pipeline. The instruction after the branch is fetched, but the instruction is ignored, and the fetch is restarted once the branch target is known. It is probably obvious that if the branch is not taken, the second IF for branch successor is redundant. This will be addressed shortly.

Penalidades do pipelining (13)

- **Penalidades de controle (ou de desvio)**
  - Solução 1: acrescentar bolhas (*stalls*) no pipeline logo após a instrução de desvio, para dar tempo do pipelining descobrir o endereço da próxima instrução (THEN ou ELSE)?

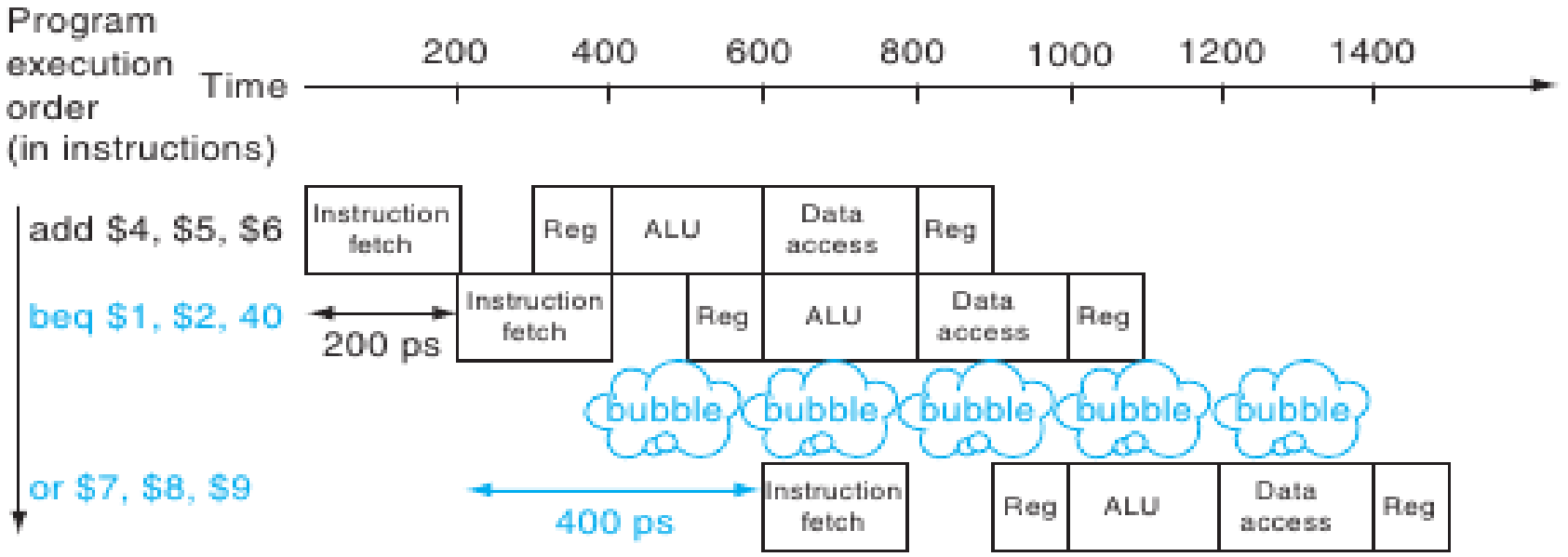
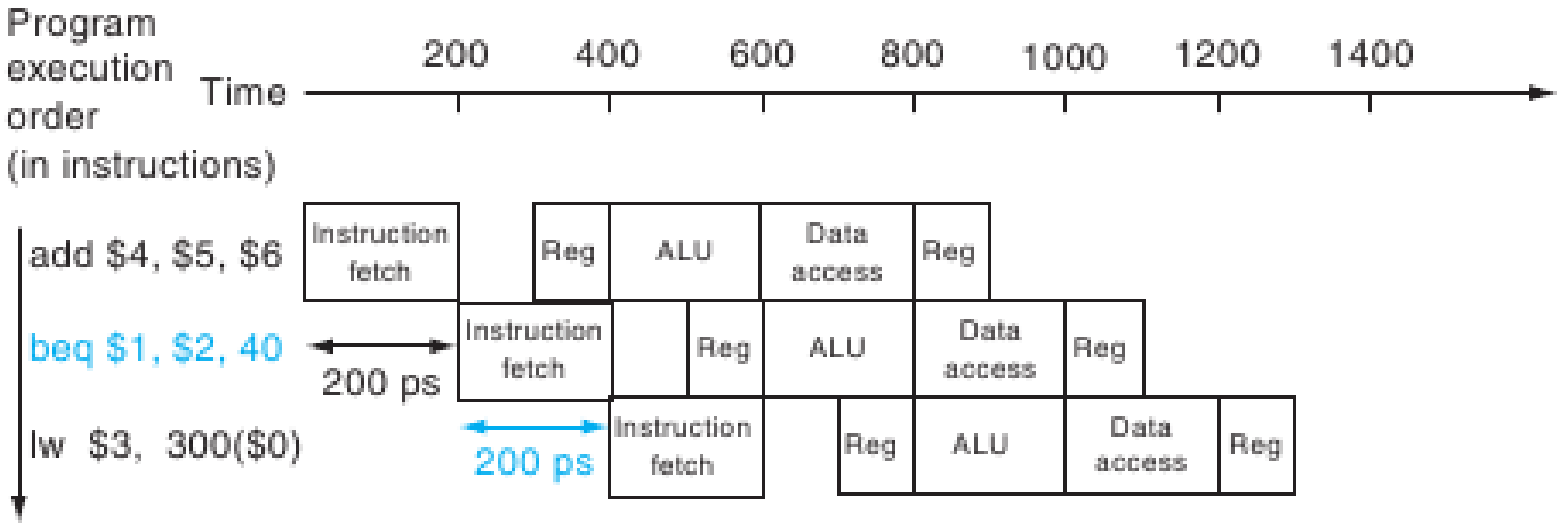


FIGURE 6.7 Pipeline showing stalling on every conditional branch as solution to control hazards. There is a one-stage pipeline stall, or bubble, after the branch. In reality, the process of creating a

Penalidades do pipelining (14)

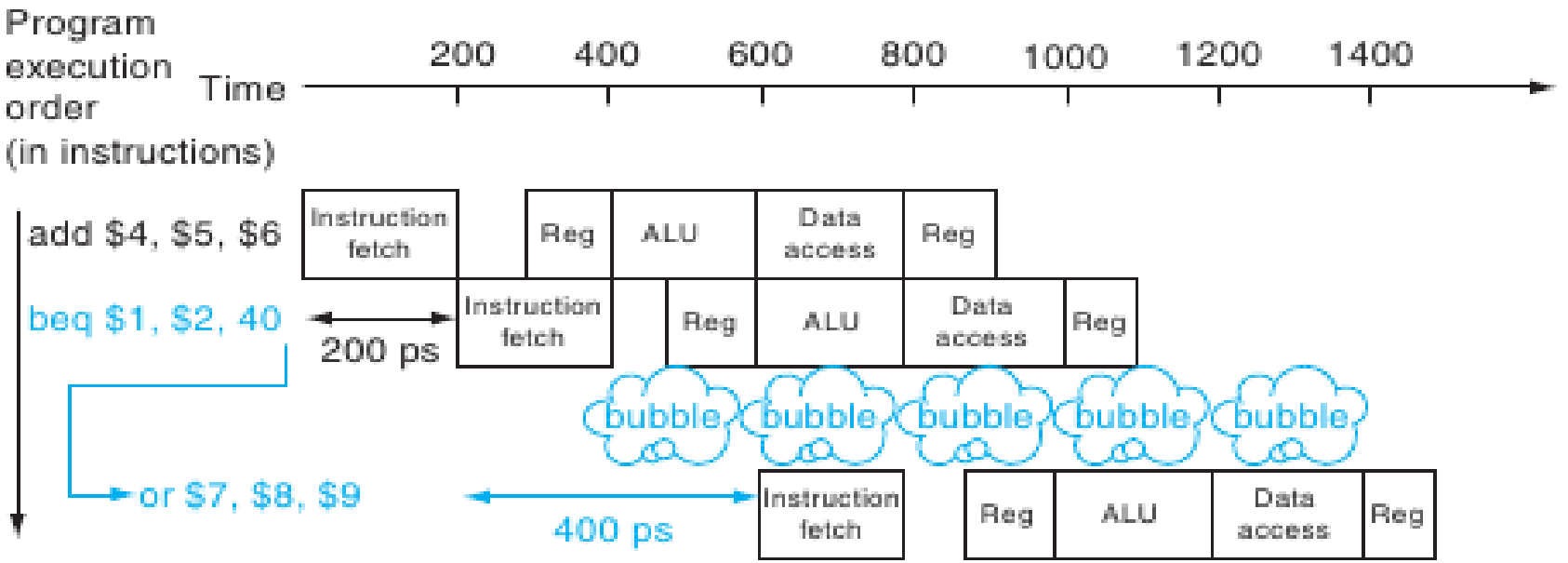
- **Penalidades de controle (ou de desvio)**
  - Solução 2: se o desvio não puder ser resolvido no 2º estágio, então deve-se tentar **prever** qual será o resultado da instrução de desvio
  - => **previsão de desvios** (*branch prediction*).
  - Uma técnica simples assume que desvios **nunca são tomados**.



Penalidades do pipelining (15)

- **Penalidades de controle (ou de desvio)**
  - Solução 2: **previsão de desvios** (*branch prediction*).
  - Uma técnica simples assume que desvios **nunca são tomados**.

Se a previsão de desvio não tomado estiver errada, o pipelining deverá ser esvaziado (a instrução `lw $3, 300($0)` deve ser descartada) e a próxima instrução (do desvio tomado) deve ser buscada (`or $7, $8, $9`)



Penalidades do pipelining (16)

- **Penalidades de controle (ou de desvio)**
  - Solução 2: **previsão de desvios** (*branch prediction*).
  - **Desvios não tomados X desvios tomados**.
    - **Esvaziamento do pipelining**

Untaken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	ID	EX	MEM	WB			
Instruction $i + 2$			IF	ID	EX	MEM	WB		
Instruction $i + 3$				IF	ID	EX	MEM	WB	
Instruction $i + 4$					IF	ID	EX	MEM	WB

Taken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	**idle**	**idle**	**idle**	**idle**			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target + 2					IF	ID	EX	MEM	WB

Figure A.12 The predicted-not-taken scheme and the pipeline sequence when the branch is untaken (top) and taken (bottom). When the branch is untaken, determined during ID, we have fetched the fall-through and just continue. If the branch is taken during ID, we restart the fetch at the branch target. This causes all instructions following the branch to stall 1 clock cycle.

Penalidades do pipelining (17)

- **Penalidades de controle (ou de desvio)**
  - Solução 2: **previsão de desvios** (*branch prediction*).
  - Uma técnica mais sofisticada assume que **certos desvios serão sempre tomados e outros não**.
  - Por exemplo: final de um laço (*loop*) sempre retorna para o início do laço => qualquer retorno para uma instrução anterior é assumida como desvio sempre tomado.
  - Esta técnica usa um **previsor de hardware dinâmico**, que baseia-se no histórico de cada desvio como tomado ou não, de modo a decidir sobre um novo desvio com base nesse comportamento recente.
  - Previsores dinâmicos por hardware costumam apresentar uma precisão de cerca de 90%, baseados em grandes bases de históricos de desvios.

- **Penalidades de controle (ou de desvio)**
  - Solução 2: **previsão de desvios** (*branch prediction*).
  - Técnica: **certos desvios serão sempre tomados e outros não**.
  - Máquina de estados para um esquema de previsão com 2 bits

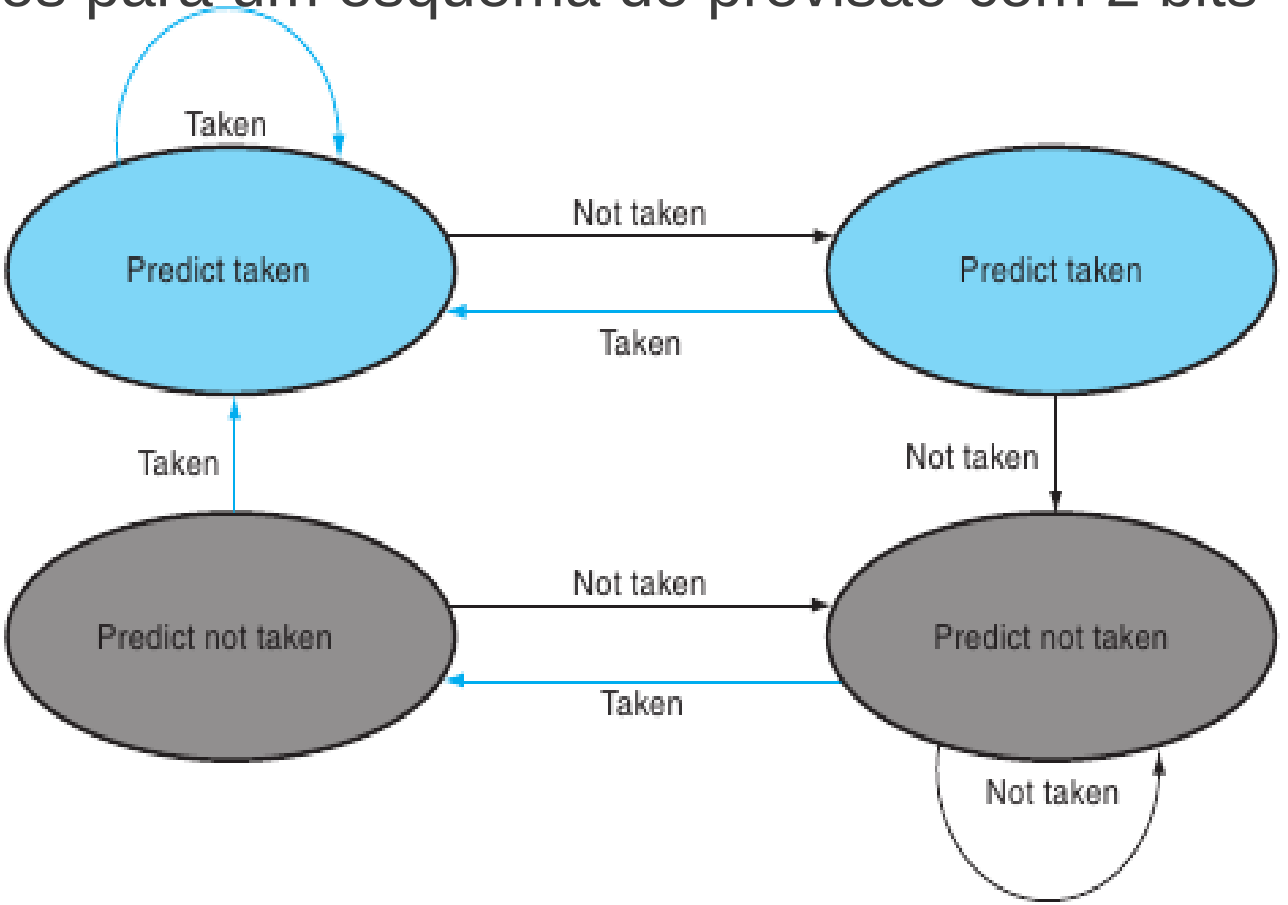


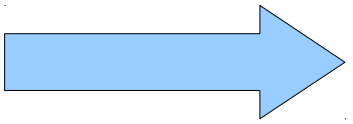
FIGURE 6.39 The states in a 2-bit prediction scheme. By using 2 bits rather than 1, a branch that strongly favors taken or not taken—as many branches do—will be mispredicted only once. The 2 bits are used to encode the four states in the system. The two-bit scheme is a general instance of a counter-based predictor, which is incremented when the prediction is accurate and decremented otherwise, and uses the midpoint of its range as the division between taken and not taken.

Penalidades do pipelining (19)

- **Penalidades de controle (ou de desvio)**
  - Solução 2: **previsão de desvios** (*branch prediction*).
  - Outra técnica sofisticada: **desvio adiado** (*delayed branch*).
    - Sempre executa a próxima instrução sequencial, com o desvio ocorrendo após esse atraso de uma instrução.
    - O montador reordena as instruções, de modo a executar uma instrução que não seja afetada pelo desvio imediatamente após a instrução de desvio.

Código MIPS

```
add $4, $5, $6
beq $1, $2, 40
or  $7, $8, $9
```



Código MIPS

```
beq $1, $2, 40
add $4, $5, $6
or  $7, $8, $9
```

- Solução 2: **previsão de desvios** => **desvio adiado**
  - Ciclo de execução com atraso de um ciclo

    ```
    instrução de desvio;
    sucessor na sequência;
    alvo do desvio se for tomado
    ```

  - Sucessor na sequência está no slot de atraso de desvio: instrução é executada independente do desvio ser tomado ou não.

Taken branch instruction	IF	ID	EX	MEM	WB				
Branch delay instruction (i + 1)		IF	ID	EX	MEM	WB			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target + 2					IF	ID	EX	MEM	WB

Figure A.13 The behavior of a delayed branch is the same whether or not the branch is taken. The instructions in the delay slot (there is only one delay slot for MIPS) are executed. If the branch is untaken, execution continues with the instruction after the branch delay instruction; if the branch is taken, execution continues at the branch target. When the instruction in the branch delay slot is also a branch, the meaning is unclear: If the branch is not taken, what should happen to the branch in the branch delay slot? Because of this confusion, architectures with delay branches often disallow putting a branch in the delay slot.

Uso do slot de atraso

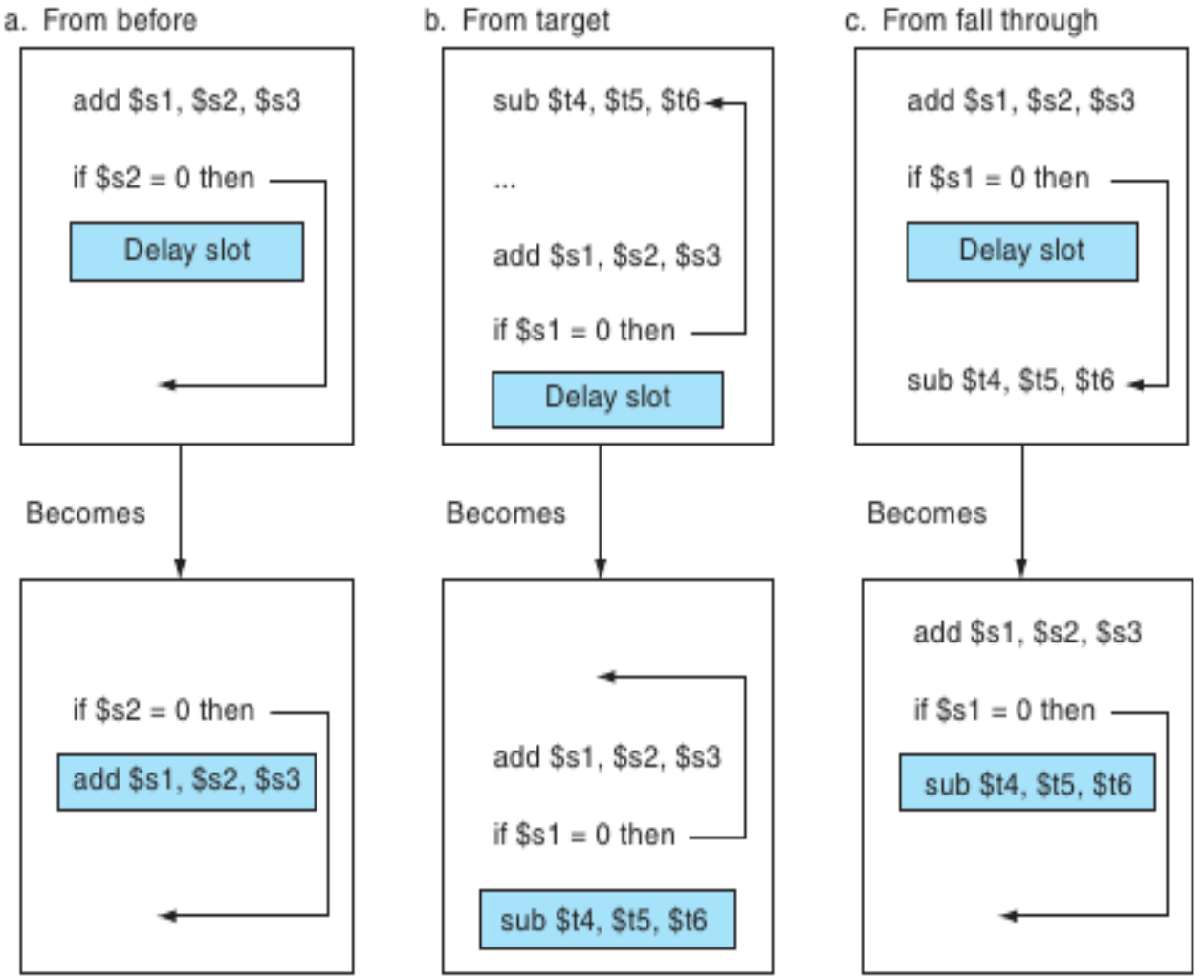


FIGURE 6.40 Scheduling the branch delay slot. The top box in each pair shows the code before scheduling; the bottom box shows the scheduled code. In (a), the delay slot is scheduled with an independent instruction from before the branch. This is the best choice. Strategies (b) and (c) are used when (a) is not possible. In the code sequences for (b) and (c), the use of `$s1` in the branch condition prevents the `add` instruction (whose destination is `$s1`) from being moved into the branch delay slot. In (b) the branch-delay slot is scheduled from the target of the branch; usually the target instruction will need to be copied because it can be reached by another path. Strategy (b) is preferred when the branch is taken with high probability, such as a loop branch. Finally, the branch may be scheduled from the not-taken fall-through as in (c). To make this optimization legal for (b) or (c), it must be OK to execute the `sub` instruction when the branch goes in the unexpected direction. By "OK" we mean that the work is wasted, but the program will still execute correctly. This is the case, for example, if `$t4` were an unused temporary register when the branch goes in the unexpected direction.

Um pequeno exercício

- **Reordenação de instruções para evitar penalidades de dados**
  - Exemplo:

Código em C

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

· Quais são as penalidades existentes?

· Como reordenar o código para eliminar tais penalidades?

Código MIPS

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

Outro pequeno exercício

- Para cada sequência da tabela, indique i) as dependências existentes (ou seja, se ela sofre alguma bolha), ii) se as bolhas podem ser evitadas usando forwarding ou iii) se ela pode executar sem nenhum stall nem forwarding.

- DICA: usar o simulador WebSimple

Sequence 1	Sequence 2	Sequence 3
lw $t0,0($t0)	add $t1,$t0,$t0	addi $t1,$t0,#1
add $t1,$t0,$t0	addi $t2,$t0,#5	addi $t2,$t0,#2
	addi $t4,$t1,#5	addi $t3,$t0,#2
		addi $t3,$t0,#4
		addi $t5,$t0,#5