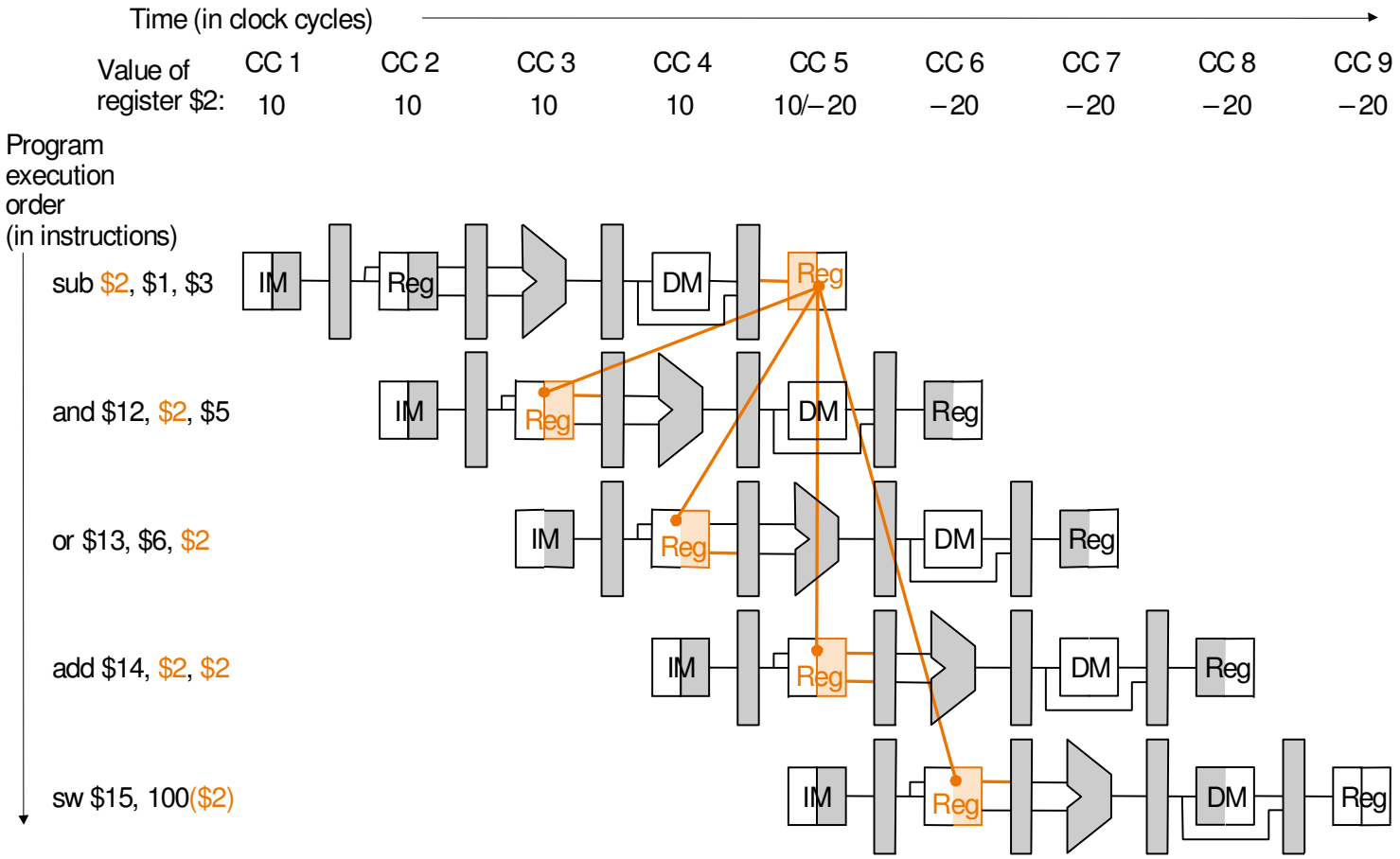
Capítulo 6
Hazards

Dependências de Dados

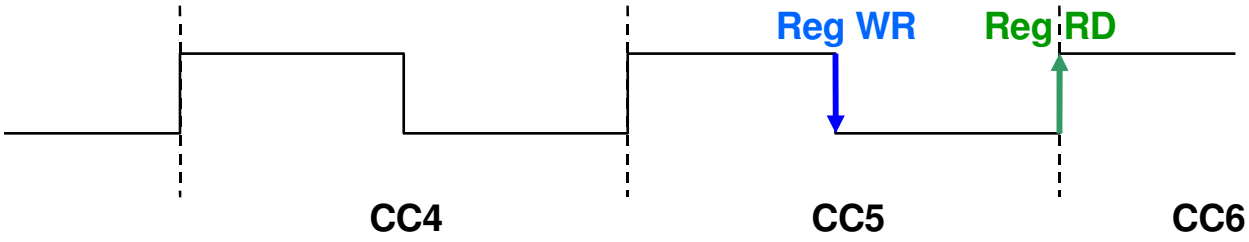
- **Problema: iniciar uma instrução antes da anterior ter finalizado**
  - **dependências que "voltam no tempo" são *hazards* de dados**
  - **qual instrução receberá o valor errado de $2 (velho 10, novo -20)**



Erro devido à dependência de dados

```
sub    $2, $1, $3
and    $12, $2, $5
or     $13, $6, $2
add    $14, $2, $2
sw     $15, 100($2)
```

- **and e or lêem resultado errado (velho 10)**
- **store está claramente à direita (depois no tempo) e lê resultado certo (-20)**
- ***hazard* no add pode ser evitado se a escrita no banco de registradores for feita (em CC5) na metade do ciclo (borda de descida)**



Solução de software

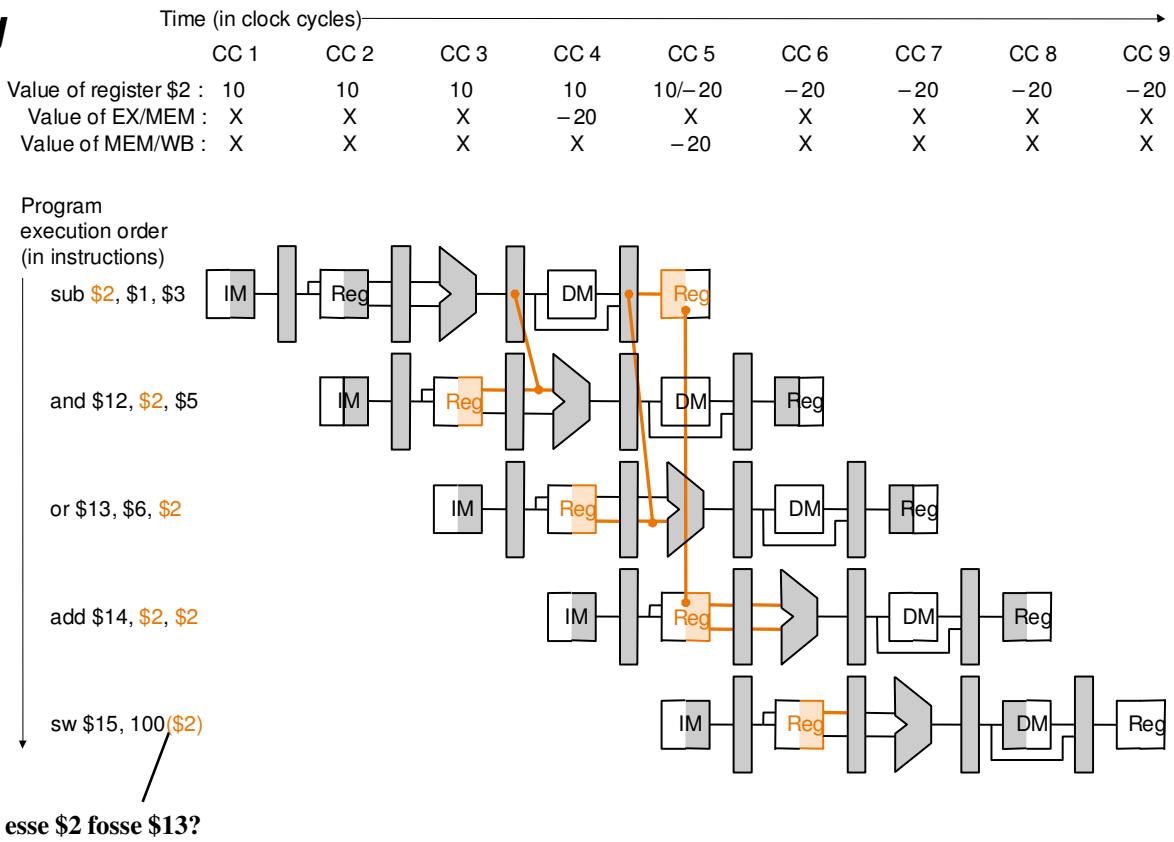
- **O compilador deve garantir a ausência de *hazards:* inserção de *nops***
- **Onde inserir os "nops" ?**
- **Quantos?**

```
sub    $2, $1, $3
and    $12, $2, $5
or     $13, $6, $2
add    $14, $2, $2
sw     $15, 100($2)
```

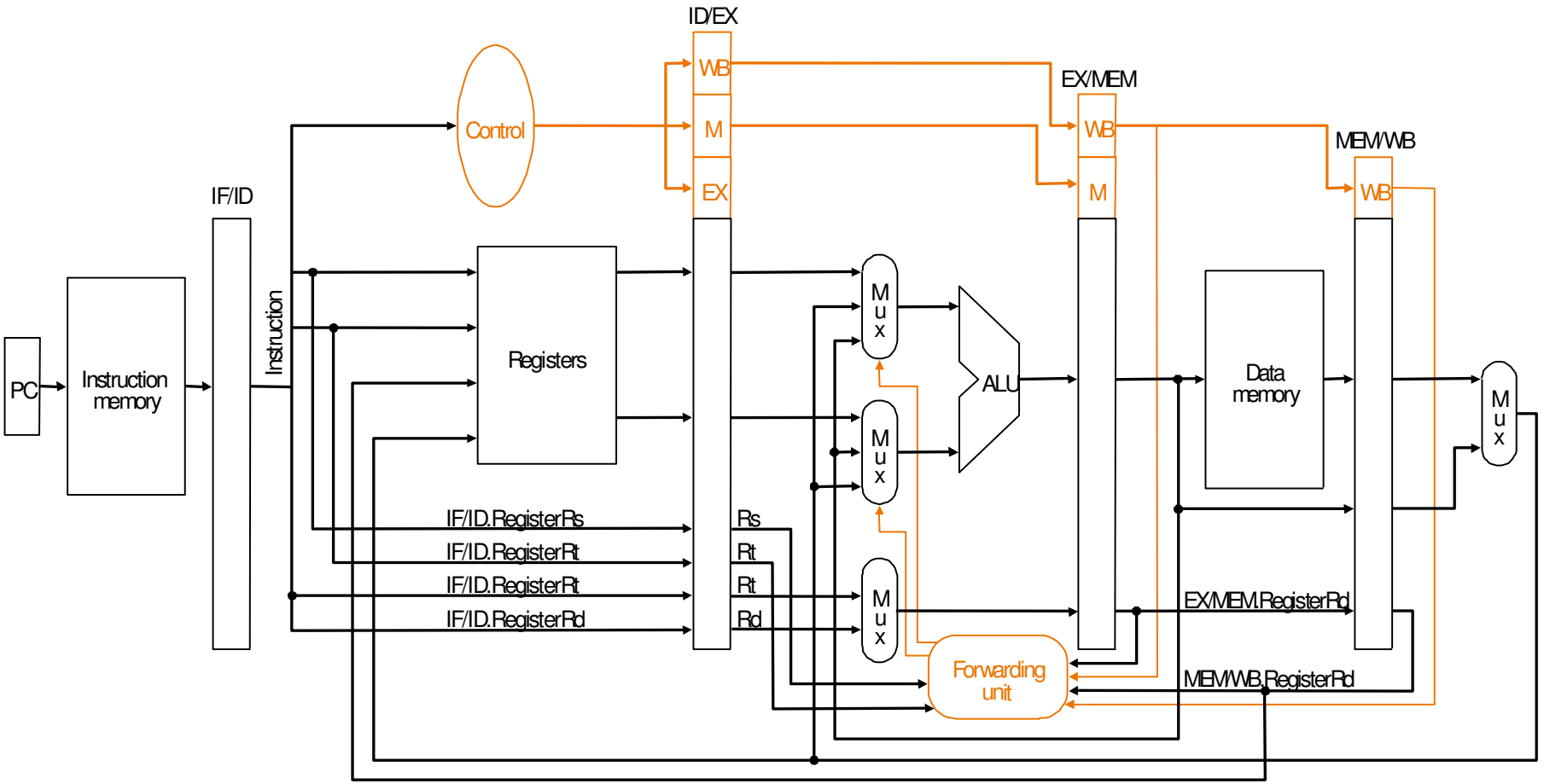
- **Problema: essa abordagem afeta o desempenho!**

Forwarding (algumas vezes: bypassing)

- **Usar resultados temporários**
  - **Não esperar que os resultados sejam escritos no banco de registradores**
  - **Unidade de *forwarding* para manipular leitura/escrita para um mesmo registrador**
  - **ULA *forwarding***



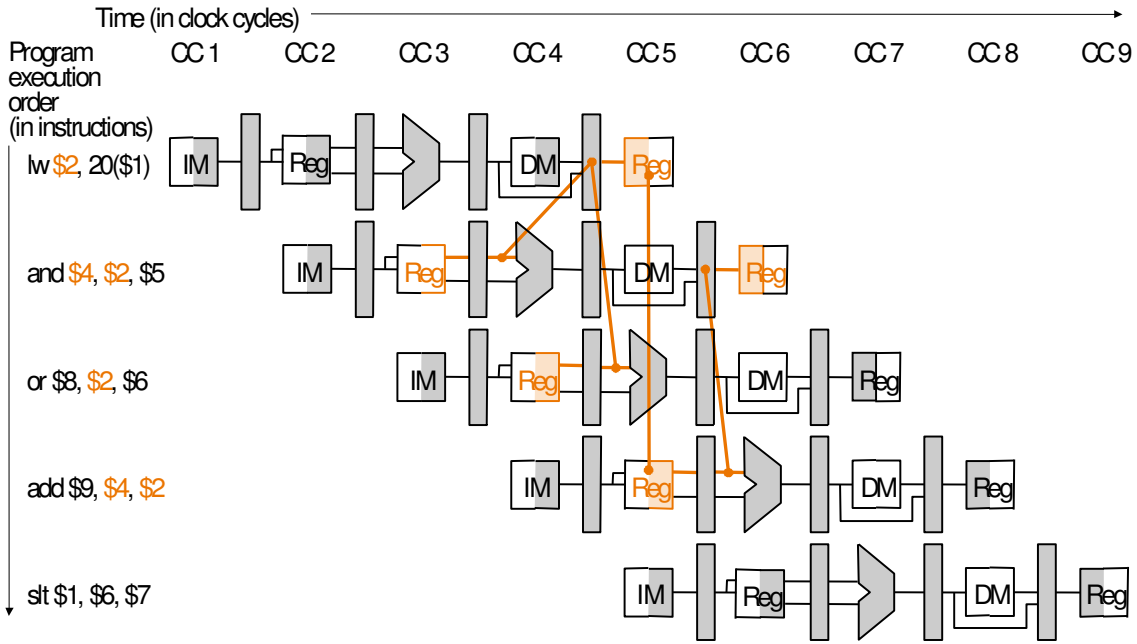
Forwarding



Forward não é válido para todos os casos

- ***Load*** **(lw) ainda pode causar um** ***hazard***
  - **Uma instrução, após um** ***load*****, tenta ler um registrador que é o alvo (destino) do** ***load***

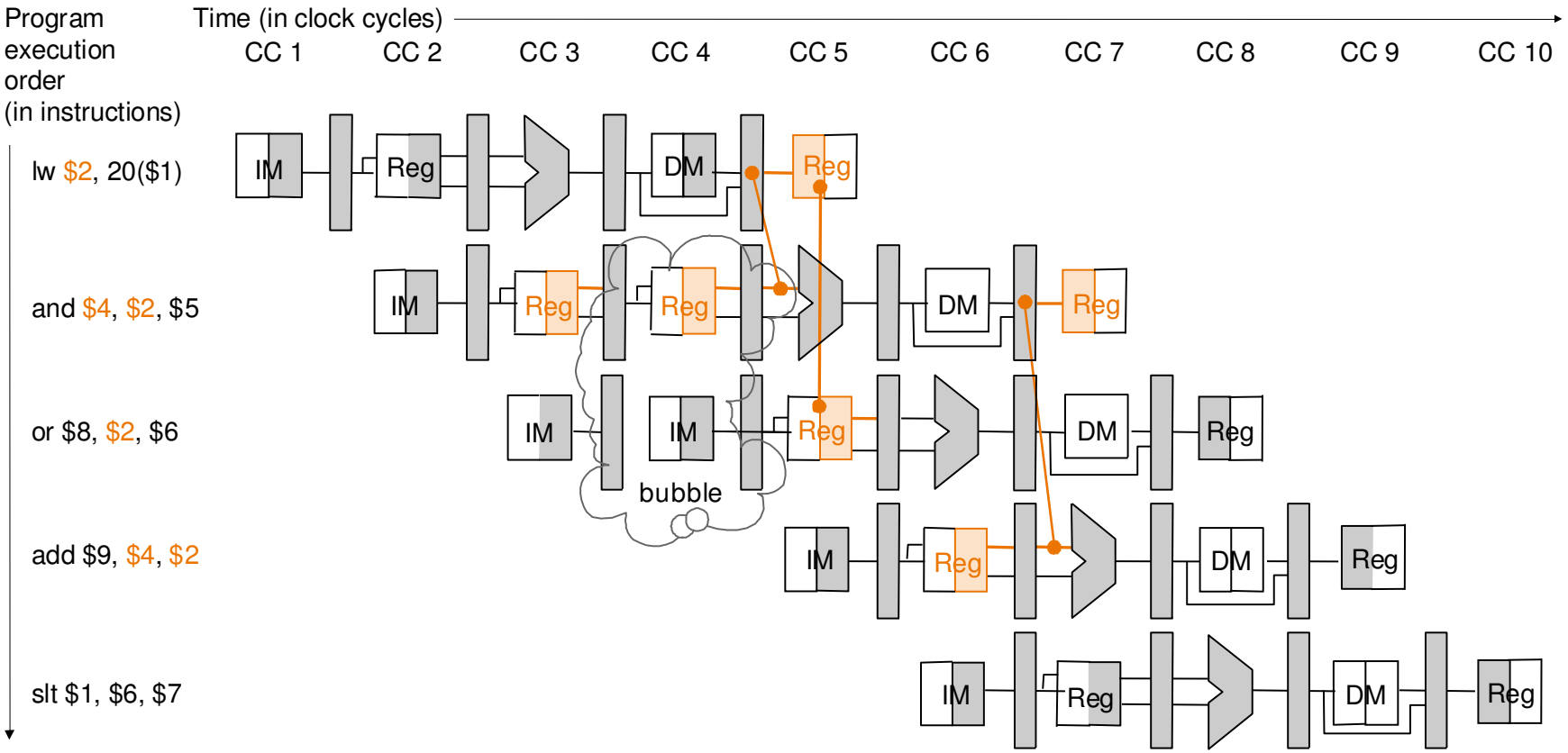
–



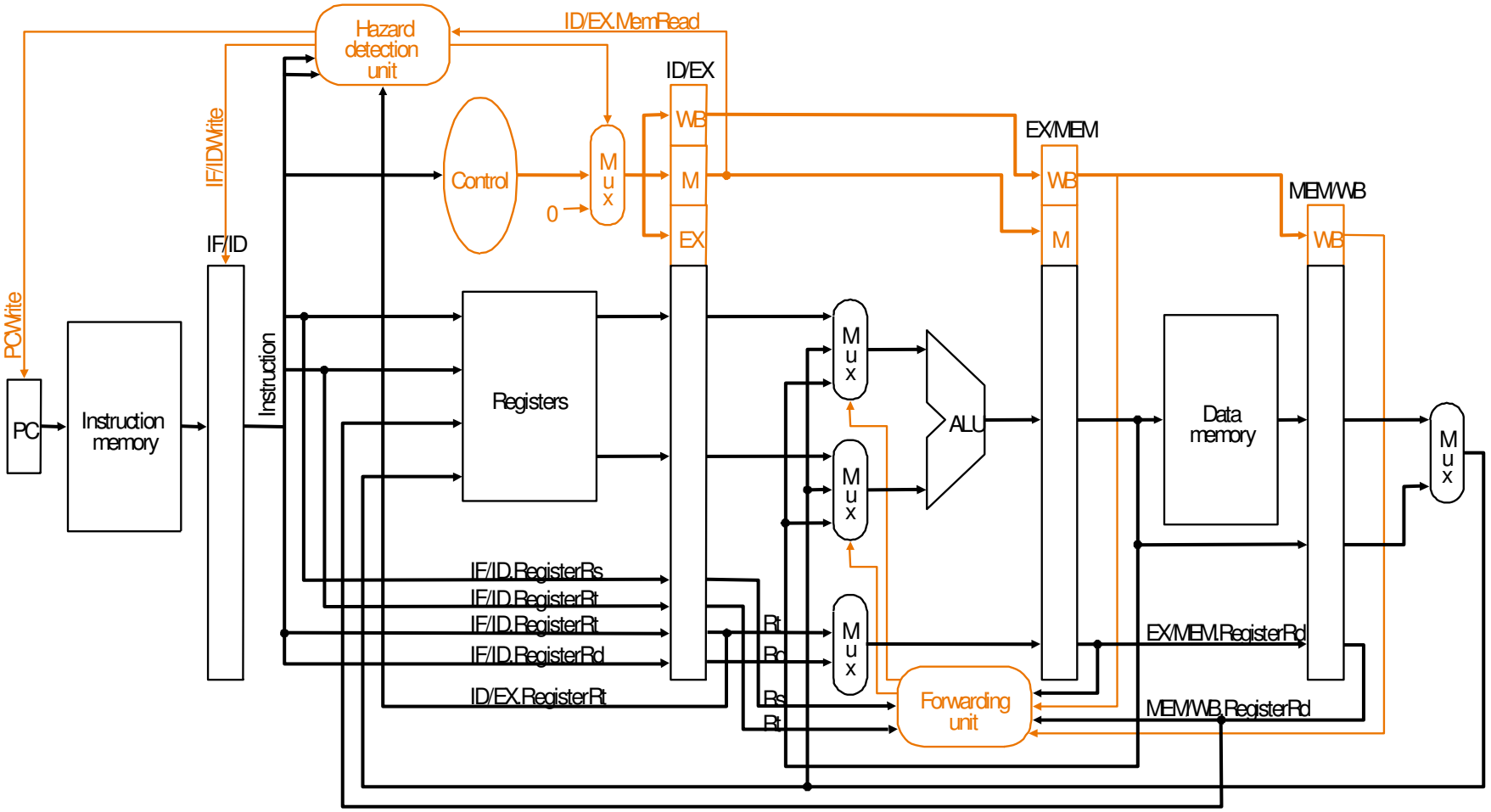
- **Uma unidade de detecção de** ***hazard*** **é necessária para travar (stall) a instrução de** ***load***

Stalling

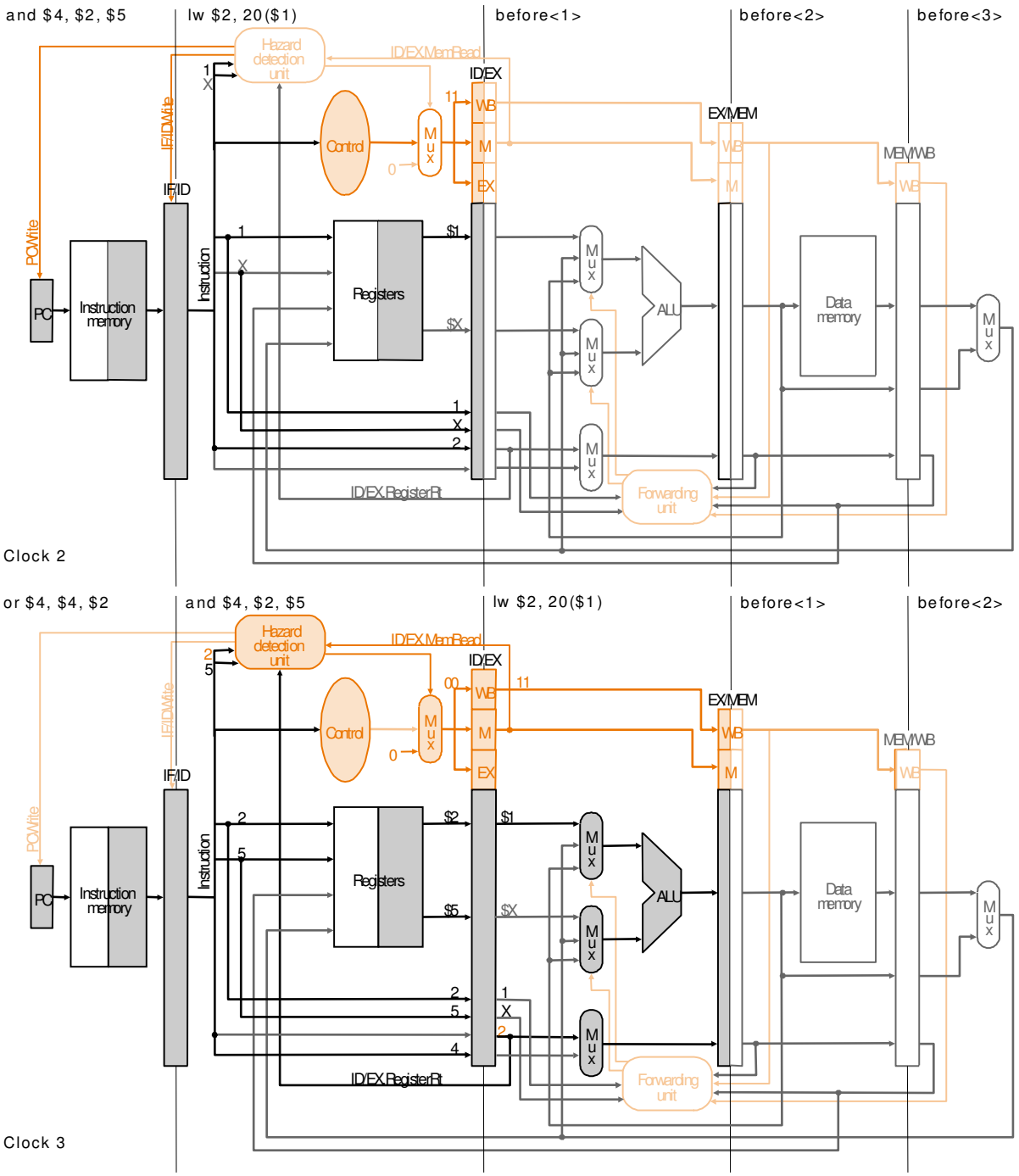
- **A idéia de *stall* consiste em "congelar" instruções em seus respectivos estágios**



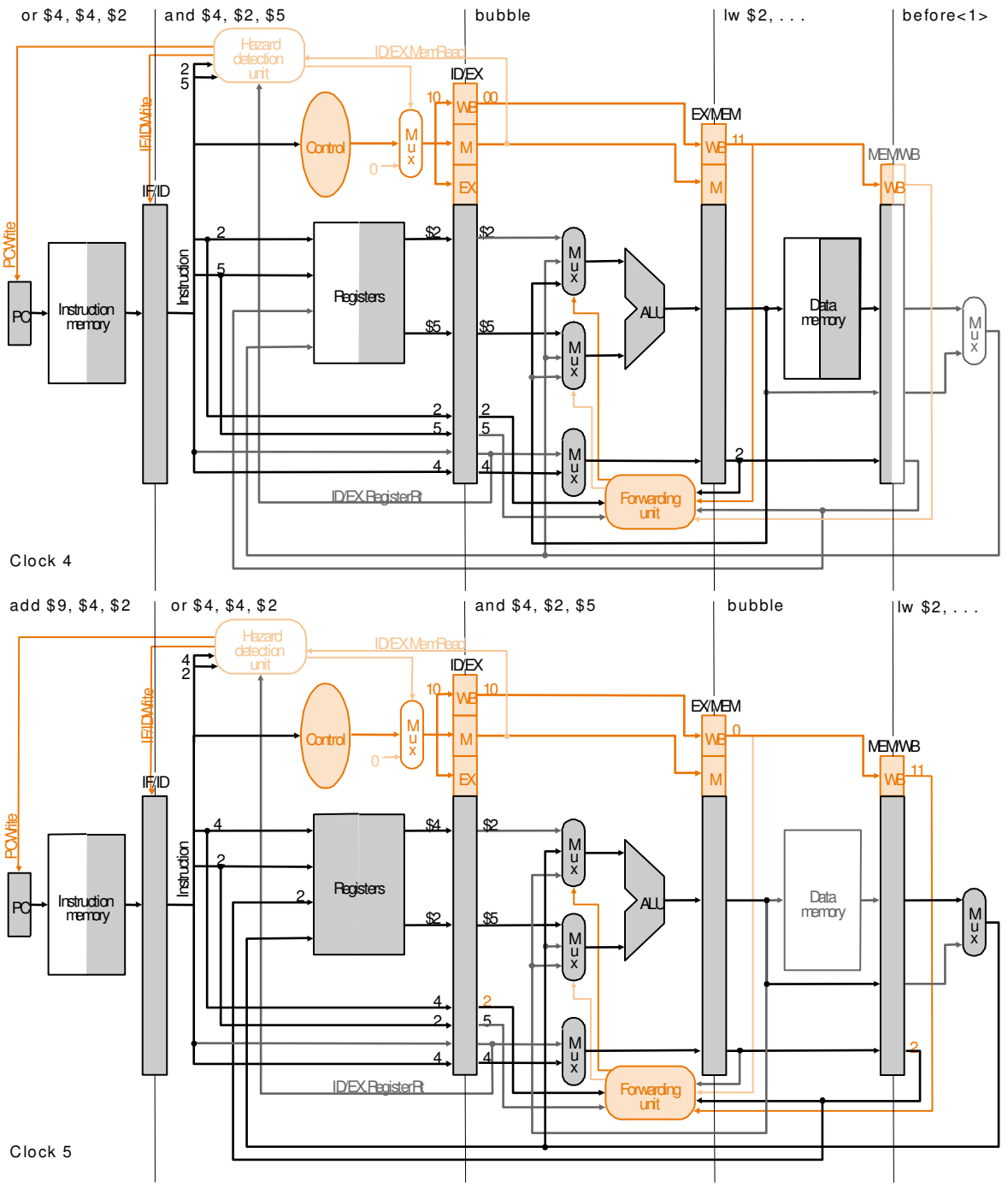
Unidade de detecção de *hazards*



Exemplo pag 493

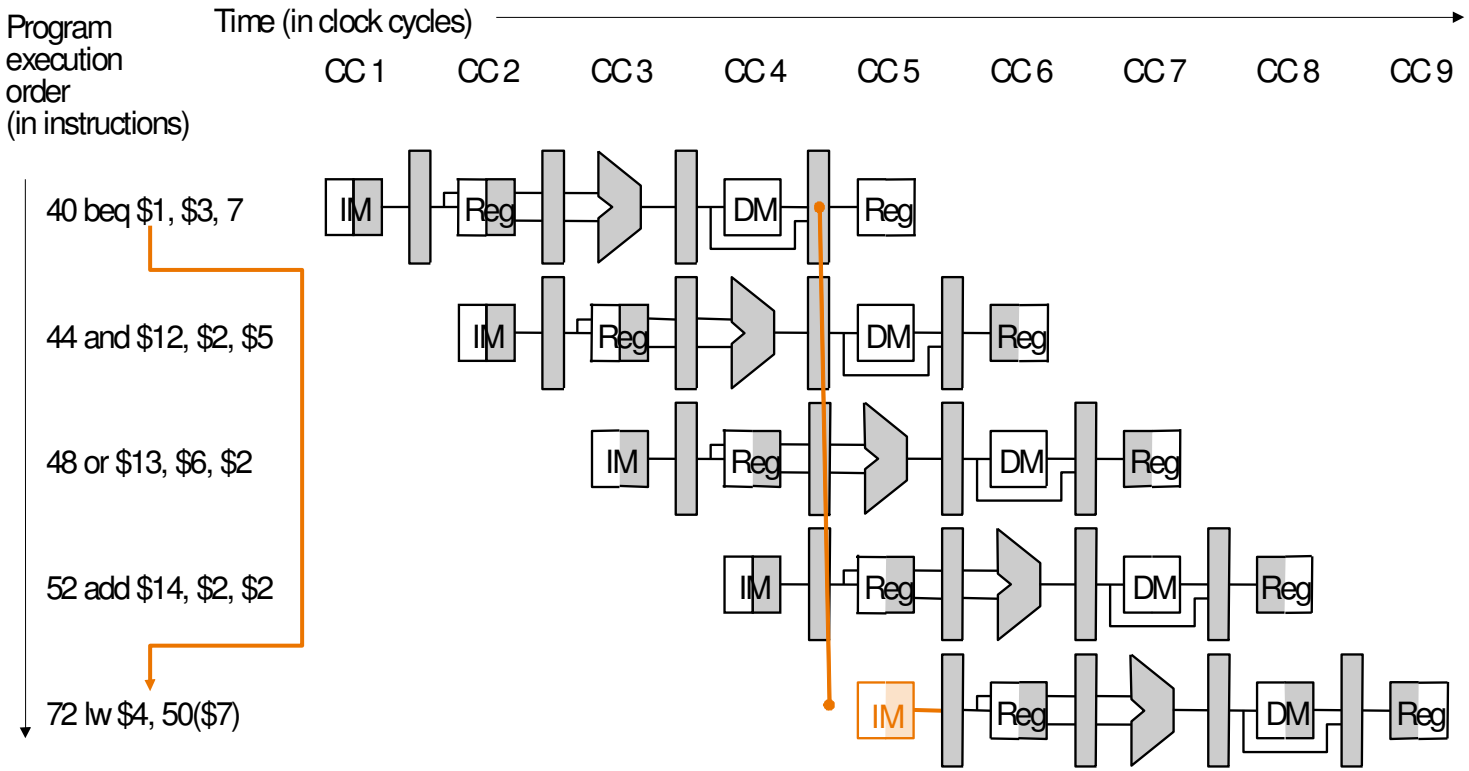


Exemplo pag 493



Hazards de desvio

- **Quando o desvio é decidido, outras instruções estão no pipeline**

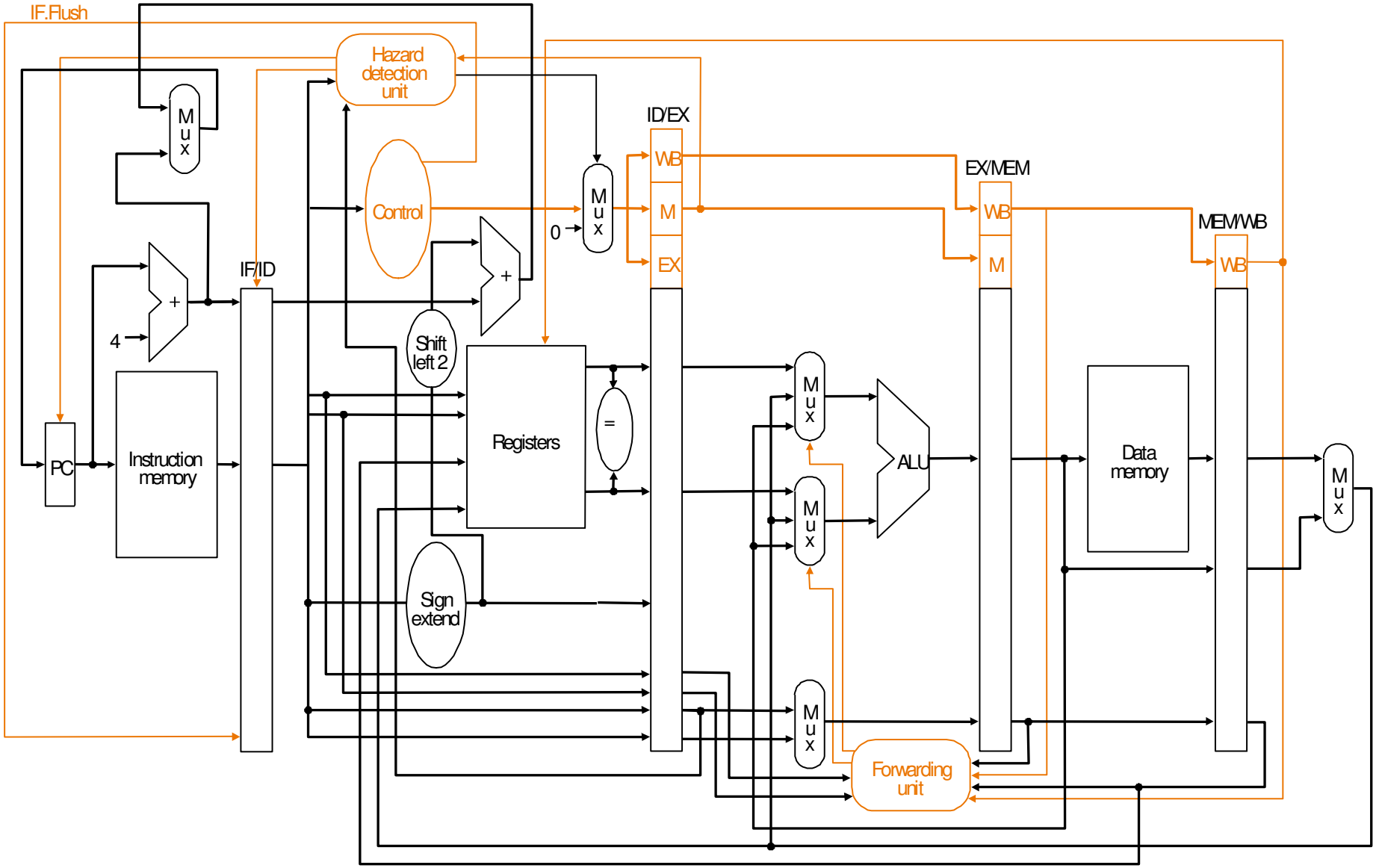


- **Nesse caso, considera-se que o desvio não será tomado (*branch not taken)* uma vez que o *stall* afeta o desempenho consideravelmente**
- **É necessário mais um hardware para limpar (flush) as instruções, caso elas não sejam as instruções corretas (alvos do desvio)**

Diminuindo a penalidade do "*branch taken*"

- **no esquema anterior**
  - **decisão só é tomada no estágio MEM**
  - **caso "*branch taken*" é necessário limpar (flush) os estágios IF, ID e EX**
  - **3 ciclos de clock perdidos**
- **para diminuir a penalidade:**
  - **antecipar a decisão do estágio MEM para o estágio ID**
  - **economia de dois ciclos clocks**
  - **Limpar somente instrução sendo lida da memória (*fetched*)**
- **mudanças no hardware:**
  - **cálculo do endereço do desvio (PC + offset<<2)**
  - **comparação dos registradores**
    - 32 **XORs** com **or** na saída
    - mais rápido do que a ALU

Limpando instruções



Funcionamento do *branch taken* com mudança na decisão

MEM -->ID

(fig 6.52)

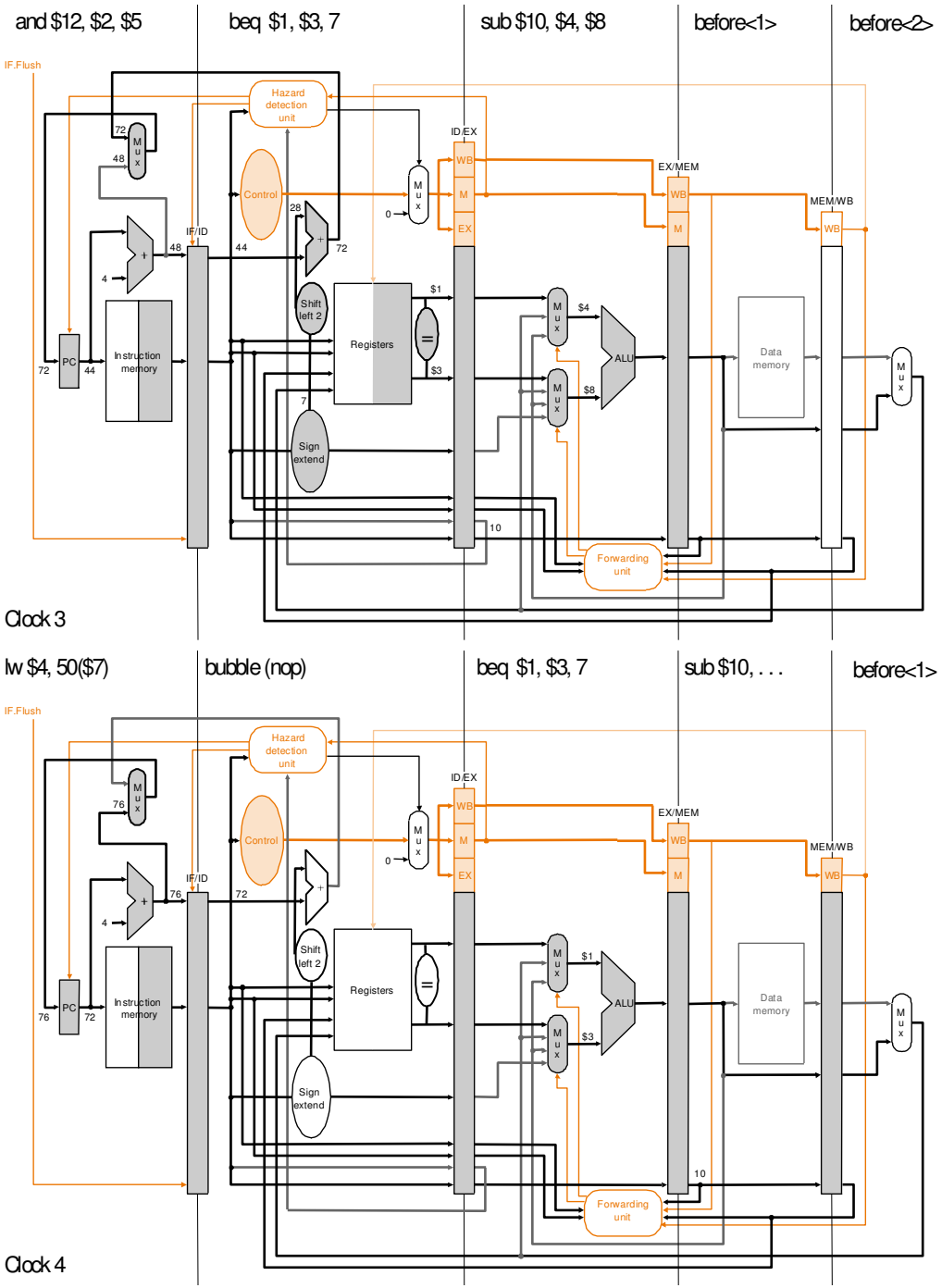
40 beq $1, $3, 7

44 and $12, $2, $5

48 or $13, $6, $2

52 add $14, $2, $2

72 lw $4, 50($7)



Melhorando o desempenho...

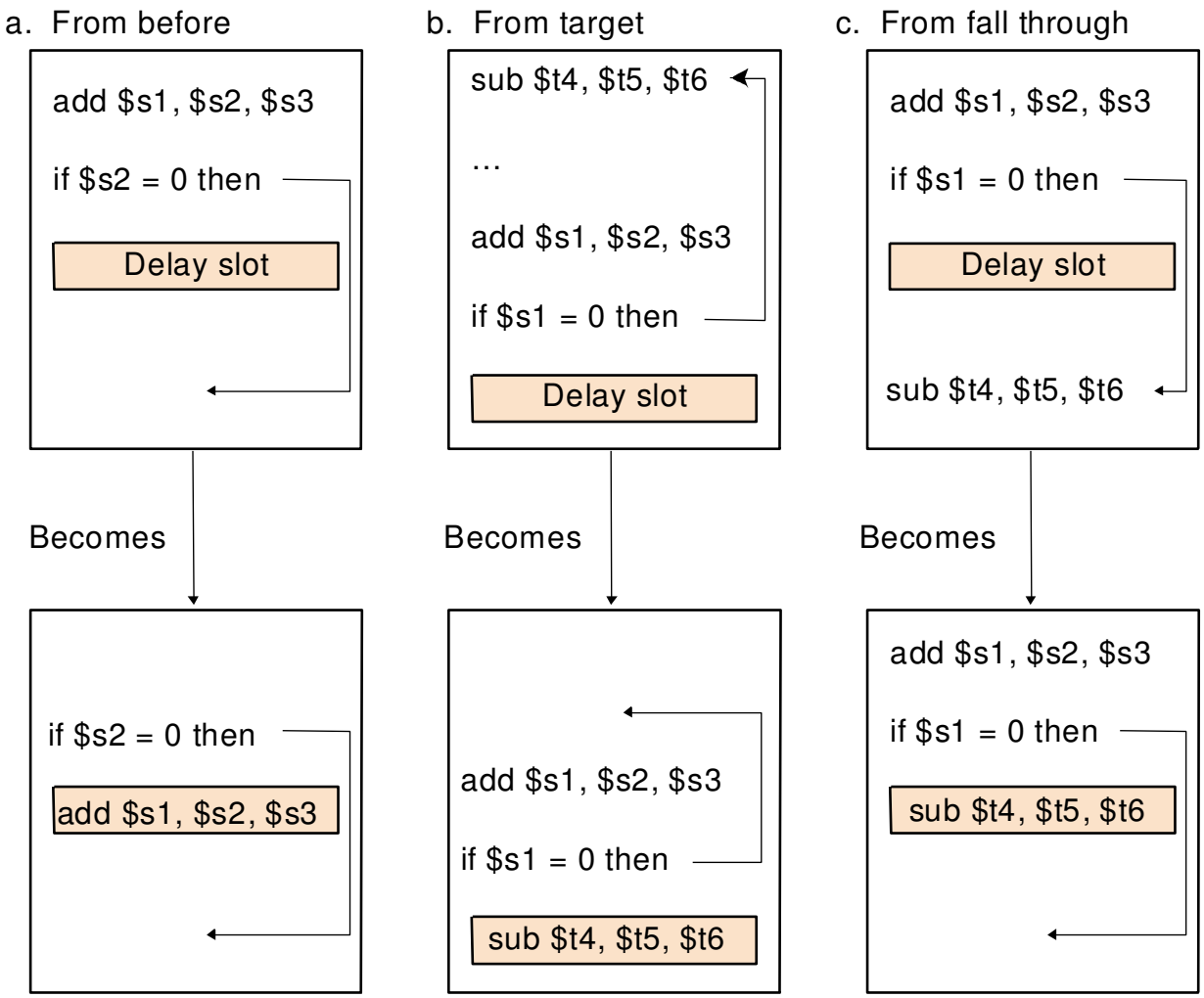
- **Tente evitar os *stalls*! Ex: reordene as instruções a seguir:**

```
lw $t0, 0($t1)
lw $t2, 4($t1)
sw $t2, 0($t1)
sw $t0, 4($t1)
```

- **Adicione um *slot* de atraso do desvio "*branch delay slot*"**
  - **A próxima instrução depois do desvio é sempre executada**
  - **Depende do compilador preencher "fill" o slot com alguma instrução útil**
- **Previsão de desvios (*Branch Prediction*)**
  - **tentar acertar se o desvio será tomado ou não**
  - **em *loops*, desvio é tomado a maior parte das vezes**
  - **Necessidade de uma *branch history table***

- **Em máquinas superscalarers: mais de uma instrução é iniciada no mesmo ciclo**

Delay slot



Escalonamento Dinâmico

- **O hardware realiza o escalonamento (“scheduling”)**
  - **Hardware tenta encontrar instruções para executar**
  - **Execução fora-de-ordem é possível**
  - **Execução especulativa e previsão dinâmica de desvios**
  - **TODOS os processadores modernos possuem algum(ns) desse(s) recurso(s) e, por isso, são bastante complicados!**

  - **DEC Alpha 21264: 9 estágios de pipeline, 6 instruções podem ser despachadas simultaneamente**
  - **PowerPC e Pentium: possuem uma *branch history table***
  - **A tecnologia existente no compilador está cada vez mais importante**

Comparação de desempenho

- **para monociclo, multiciclo e pipeline**
- **gcc: lw (23%), sw (13%), beq (19%), j (2%), resto (43%)**
- **pipeline:**
  - **2ns (memória e ALU) e 1ns para o registrador (RD ou WR)**
  - **1/2 dos lw seguidos por instruções que usam o resultado**
  - **1/4 dos beq são errados (1 clock perdido)**
  - **jumps perdem um ciclo**
- **pipeline:**
  - **lw: 1 clock sem hazard e 2 clock com hazard; média = 1.5**
  - **beq: 1 clock se OK (3/4) e 2 clocks se não OK (1/4); média = 1.25**
  - **jump: 2 clocks**
  - **demais instruções: 1 clock**
    - CPI = 1.5*0.23+1*0.13+1*0.43+1.25*0.19+2*0.02 = 1.18
    - clock = 2ns; tempo médio de execução = 2*1.18=**2.36**ns
    - multiciclo; 4.02*2=**8.08**ns; monociclo = **8**ns
    - **pipeline é 3.4 vezes mais rápido do que monociclo ou multiciclo**

Circuito completo