MC542

Organização de Computadores
Teoria e Prática

2006

Prof. Paulo Cesar Centoducatte

ducatte@ic.unicamp.br

www.ic.unicamp.br/~ducatte

MC542

Arquitetura de Computadores

Processador MIPS
Pipeline (continuação)

"Computer Organization and Design:
The Hardware/Software Interface" (Capítulo 6)

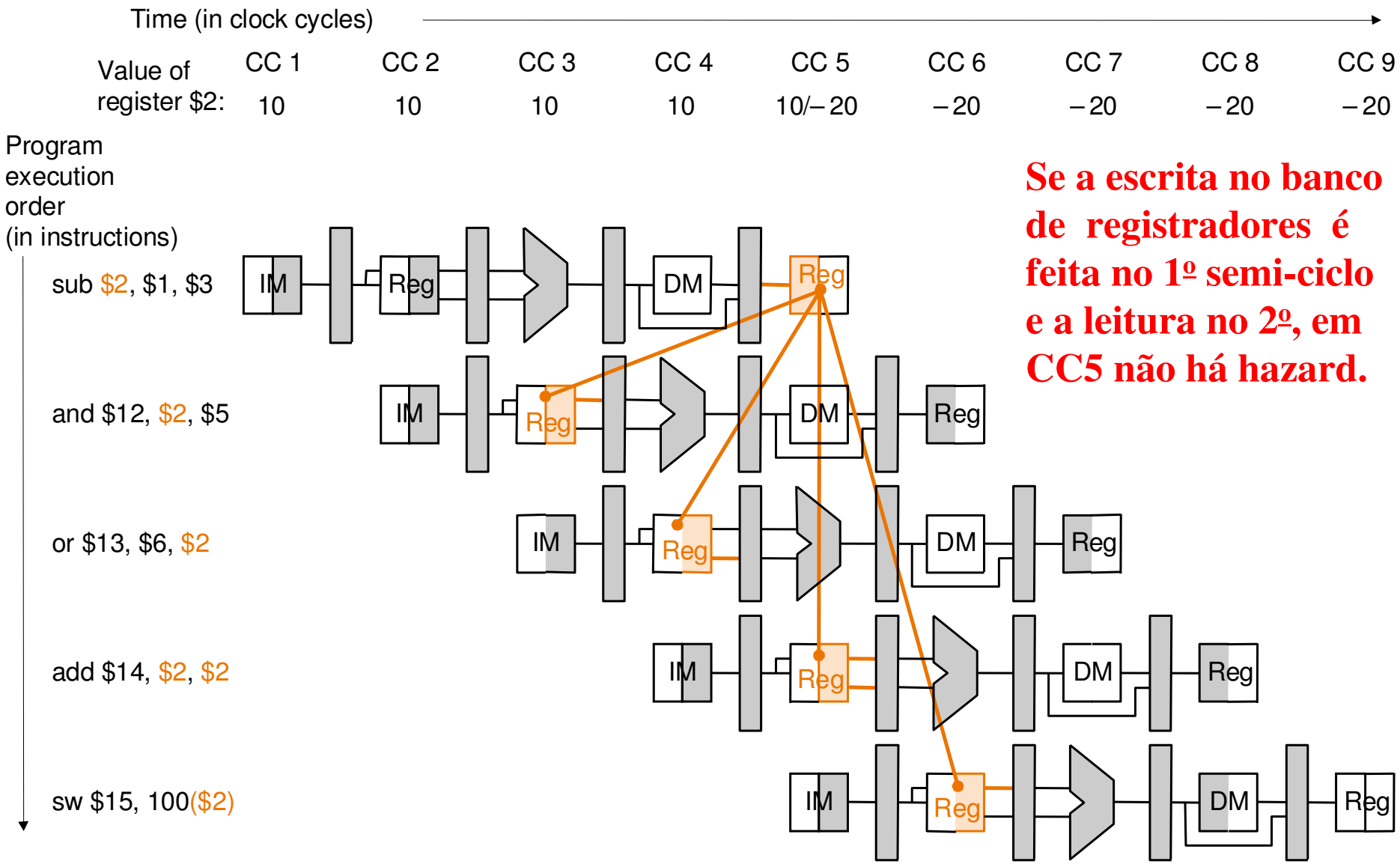
Sumário

- Dependências
  - Soluçãopor Softwarwe
  - Solução por Hardware
- Forwarding
- Condições para determinar se há Hazard de dados
- Datapath sem Forwarding
- Datapath com Forwarding
- Detecção e Controle de Hazard
- Datapath Modificado para Fowarding
- Exemplo de Execução de instruções com Forwarding

Dependências

- Problema com iniciar a execução de uma instrução antes do término da anterior
  - Exemplo: Instruções com dependências

```
sub     $2,  $1, $3      #  reg $2 modificado
and     $12, $2, $5      #  valor (1° operando) de $2
                         # depende do sub
or      $13, $6, $2      #  idem (2° operando)
add     $14, $2, $2      #  idem (1° e 2° operando)
sw      $15, 100($2)     #  idem (base do endereçamento)
```



Solução por Software

- Compilador garante um código sem Hazard inserindo nops
  - Onde inserir os nops?

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

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

Problema: reduz o desempenho

Solução por Hardware

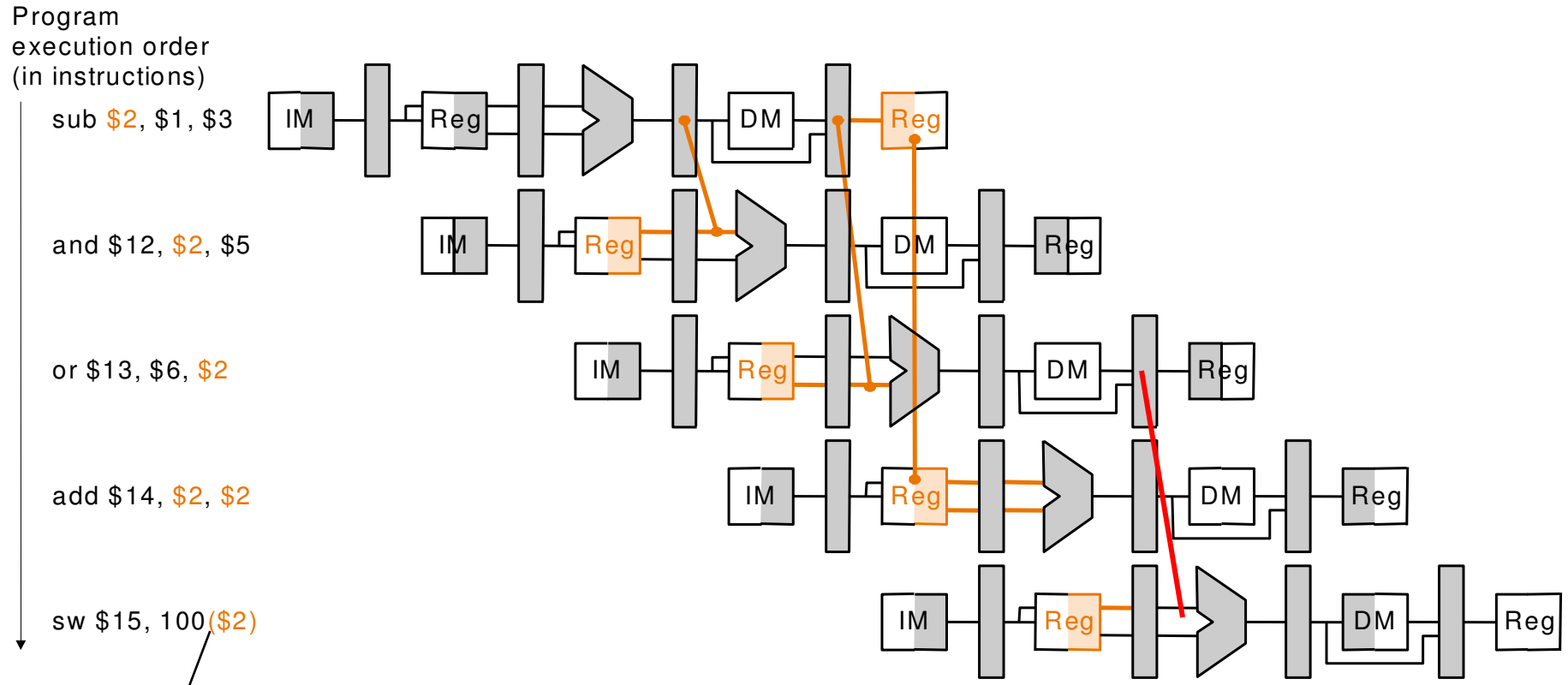
- Usar o resultado assim que calculado, não esperar que ele seja escrito.

- Neste caso é necessário mecanismo para detectar o hazard. Qual a condição para que haja hazard?

Quando uma instrução tenta ler um registrador (estágio EX) e esse registrador será escrito por uma instrução anterior no estágio WB

Forwarding

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2 :	10	10	10	10	10/– 20	– 20	– 20	– 20	– 20
Value of EX/MEM :	X	X	X	– 20	X	X	X	X	X
Value of MEM/WB :	X	X	X	X	– 20	X	X	X	X



O que ocorre se $13 no lugar de $2

Condições que determinam Hazards de dados

- EX/MEM
  - EX/MEM.RegisterRd = ID/EX.RegisterRs
  - EX/MEM.RegisterRd = ID/EX.RegisterRt
- MEM/WB
  - MEM/WB.RegisterRd = ID/EX.RegisterRs
  - MEM/WB.RegisterRd = ID/EX.RegisterRt

Condições que determinam Hazards de dados

- Exemplo:

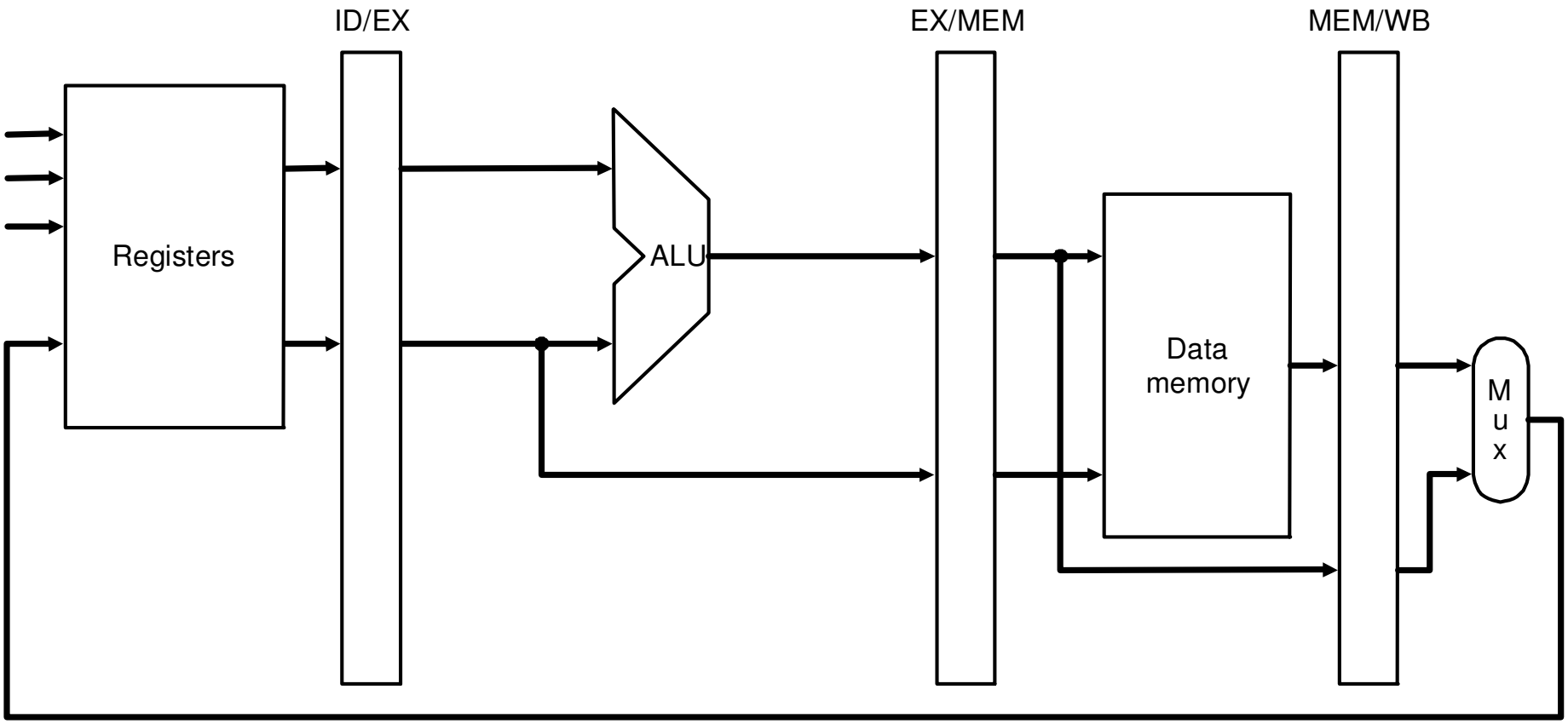
```
sub     $2, $1, $3        #  reg $2 modificado
and     $12,  $2, $5      #  valor de $2  depende do sub
or      $13, $6, $2       #  idem (2º operando)
add     $14,$2,  $2       #  idem (1º e 2º operandos)
sw      $15, 100($2)      #  idem (base do endereçamento)
```

  - sub–and: EX/MEM.RegisterRd = ID/EX.RegisterRs = $2
  - sub-or: MEM/WB.RegisterRd = ID/EX.RegisterRt = $2
  - sub-add: não tem hazard
  - sub-sw: não tem hazard

Condições que determinam Hazards de dados

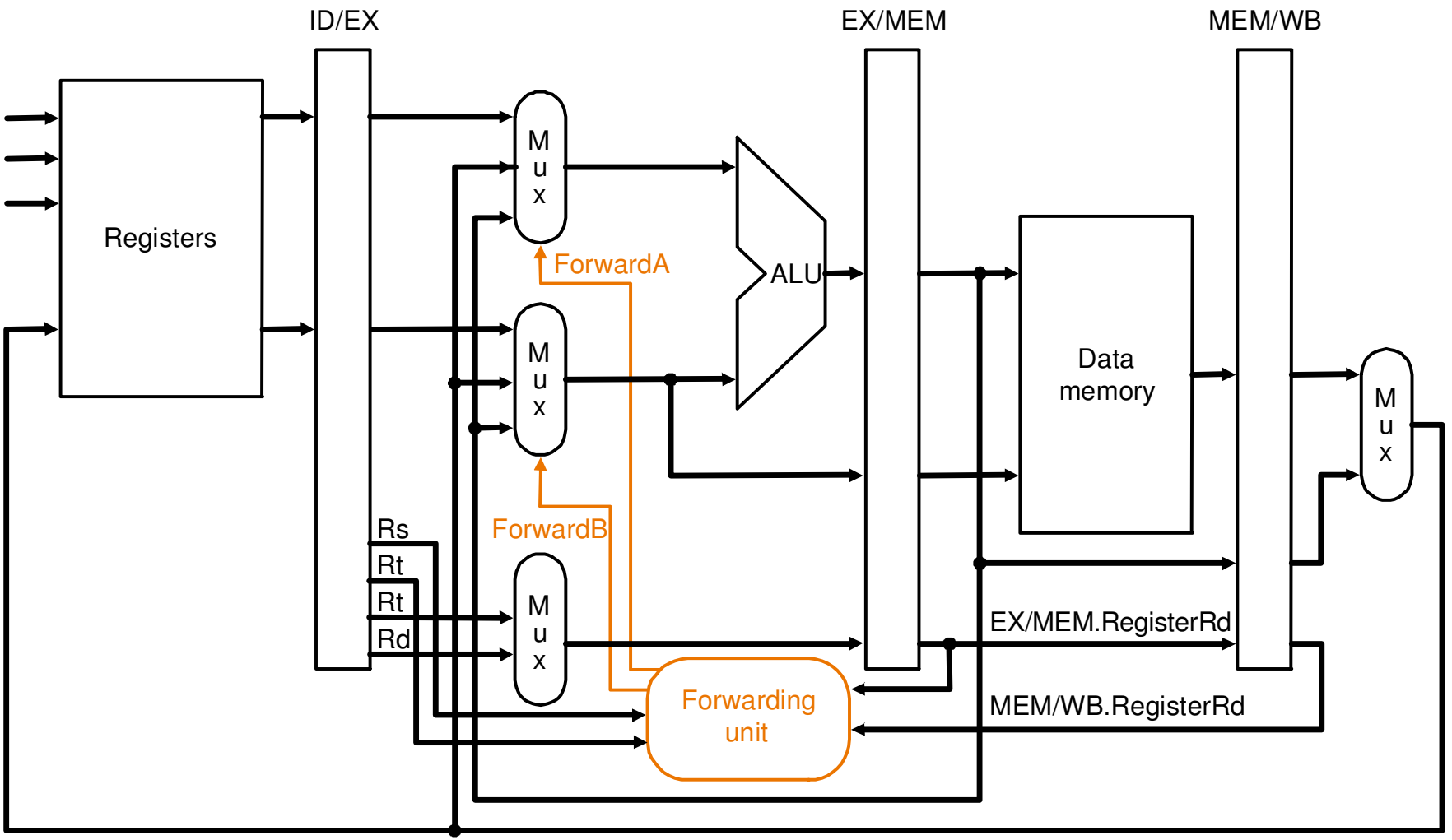
- Esta não é uma política precisa, pois existem instruções que não escrevem no register file.
  - solução seria verificar o sinal RegWrite
- MIPS usa $0 para operandos de valor 0. Para instruções onde $0 é destino?
  - sll $0, $1, 2
    - valor diferente de zero nos regs de pipeline
  - add $3, $0, $2
    - se houver fowarding, $3 terá valor errado no fim da instrução add
- Para isto temos que incluir as condições
  - EX/MEM.registerRd <> 0 (1° hazard)
  - MEM/WB.registerRd <> 0 (2° hazard)

Datapath sem Forwarding



a. No forwarding

Datapath com Forwarding (para add, sub, and e or)



b. With forwarding

Valores dos Sinais ForwardA e ForwardB

Mux Control	Source	Explanação
ForwardA = 00	ID/EX	Primeiro operando da ULA → register file
ForwardA = 10	EX/MEM	Primeiro operando da ULA → resultado anteiror da ULA
ForwardA = 01	MEM/WB	Primeiro operando da ULA é antecipado da memória de dados ou um resultado anterior da ULA
ForwardB = 00	ID/EX	Segundo operando da ULA → register file
ForwardB = 10	EX/MEM	Segundo operando da ULA → resultado anteiror da ULA
ForwardB = 01	MEM/WB	Segundo operando da ULA é antecipado da memória de dados ou um resultado anterior da ULA

Detecção e Controle de Hazard

- **O controle de fowarding será no estágio EX, pois é neste estágio que se encontram os multiplexadores de fowarding da ULA. Portanto devemos passar o número do registrador operando do estágio ID via registrador de pipeline ID/EX ➔ adicionar campo rs (bits 25-21).**

Condições para Detecção de Hazard

EX hazard

if (EX/MEM.RegWrite and (EX/MEM.RegisterRd <> 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
FowardA = 10
if (EX/MEM.RegWrite and (EX/MEM.RegisterRd <> 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
FowardB = 10

MEM hazard

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd <> 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
FowardA = 01
if (MEM/WB.RegWrite and (MEM/WB.RegisterRd <> 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
FowardB = 01

Observações:

- Não existe data hazard no estágio WB, pois estamos assumindo que o register file supre o resultado correto se a instrução no estágio ID é o mesmo registrador escrito pela instrução no estágio WB ➔ forwarding no register file
  - Na 1ª edição do livro P&H tem hazard

Observações:

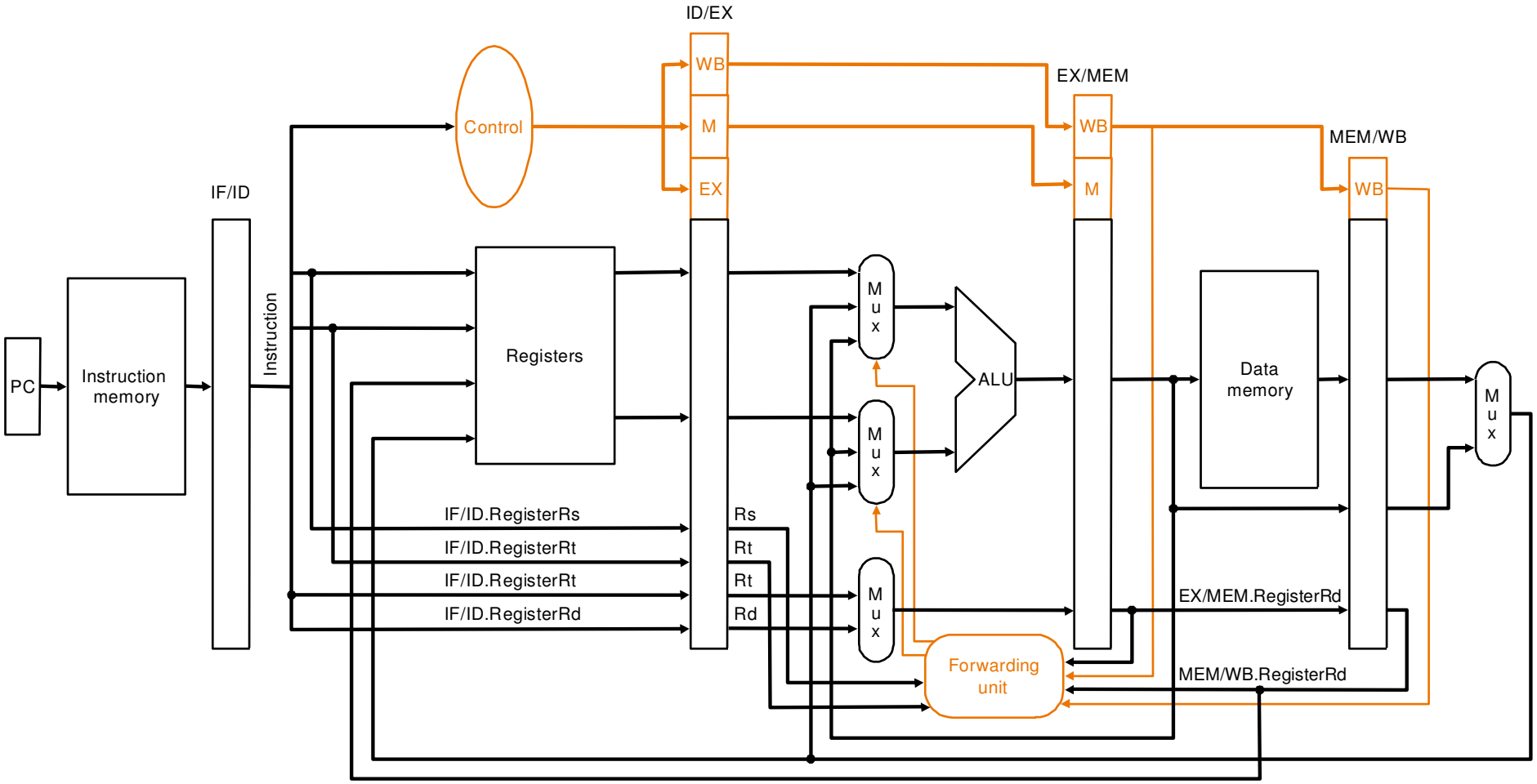
- Um data hazard potencial pode ocorrer entre o resultado de uma instrução em WB, o resultado de uma instrução em MEM e o operando fonte da instrução no estágio da ULA

```
add   $1, $1, $2
add   $1, $1, $3
add   $1, $1, $4
```

- Neste caso o resultado é antecipado do estágio MEM porque o resultado neste estágio é mais recente

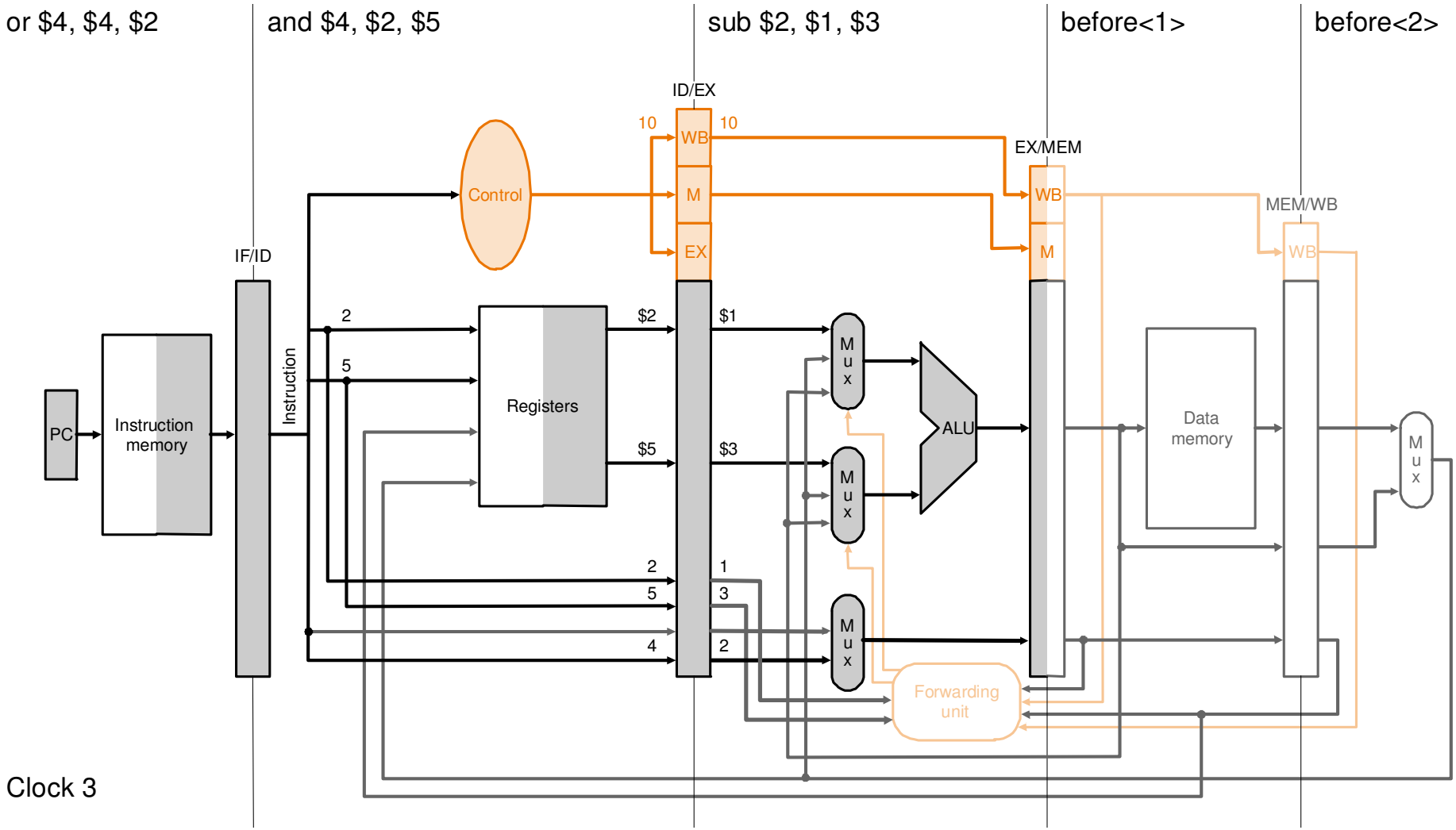
```
if (MEM/WB.RegWrite and (MEM/WB.RegisterRd <> 0 )
and (EX/MEM.RegisterRd <> ID/EX.registerRs)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) FowardA = 01

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd <> 0 )
and (EX/MEM.RegisterRd <> ID/EX.registerRt)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) FowardB = 01
```

Datapath Modificado para Fowarding



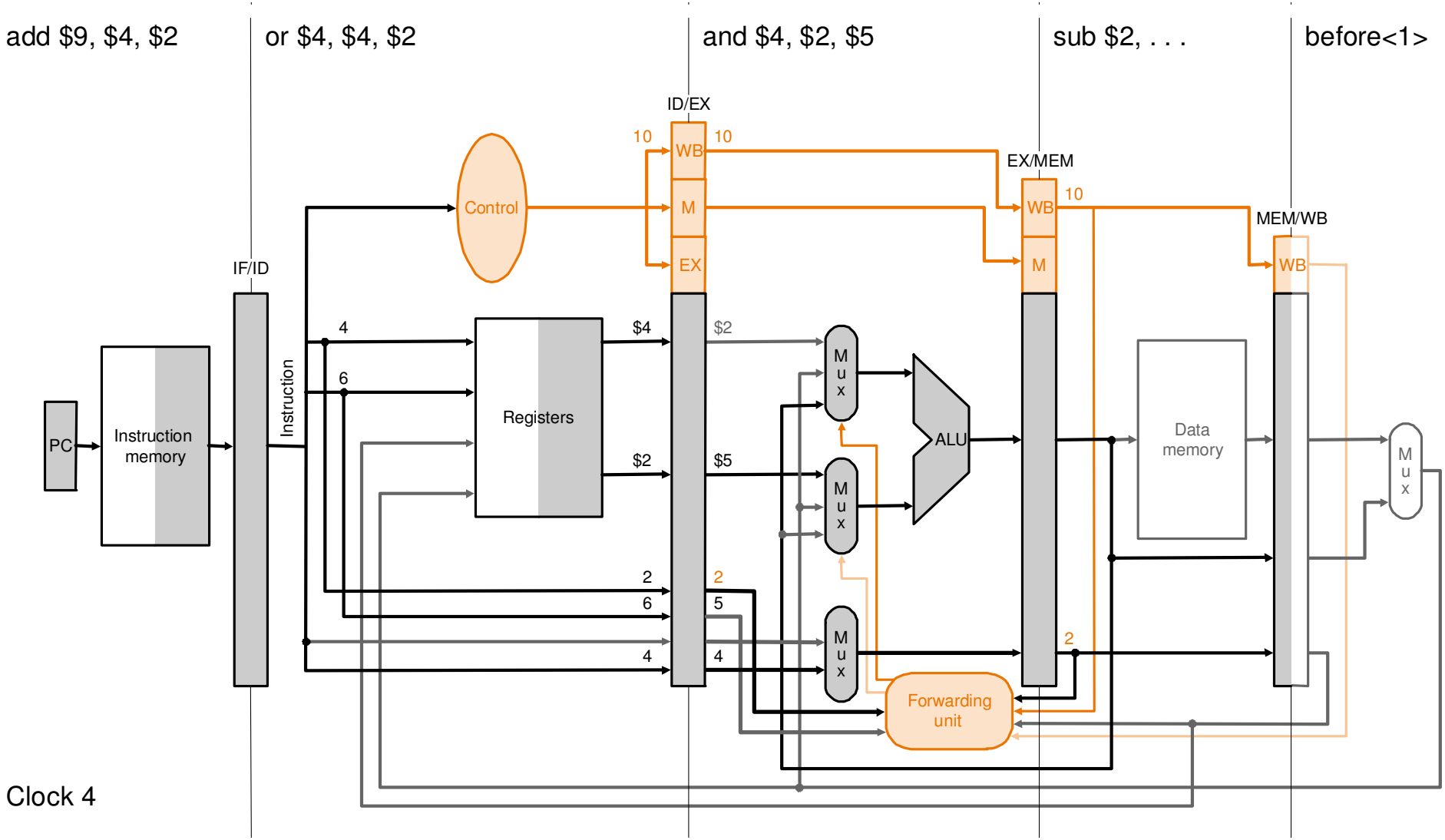
Forwarding

```
sub   $2, $1, $3
and   $4, $2, $5
or    $4, $4, $2
add   $9, $4, $2
```



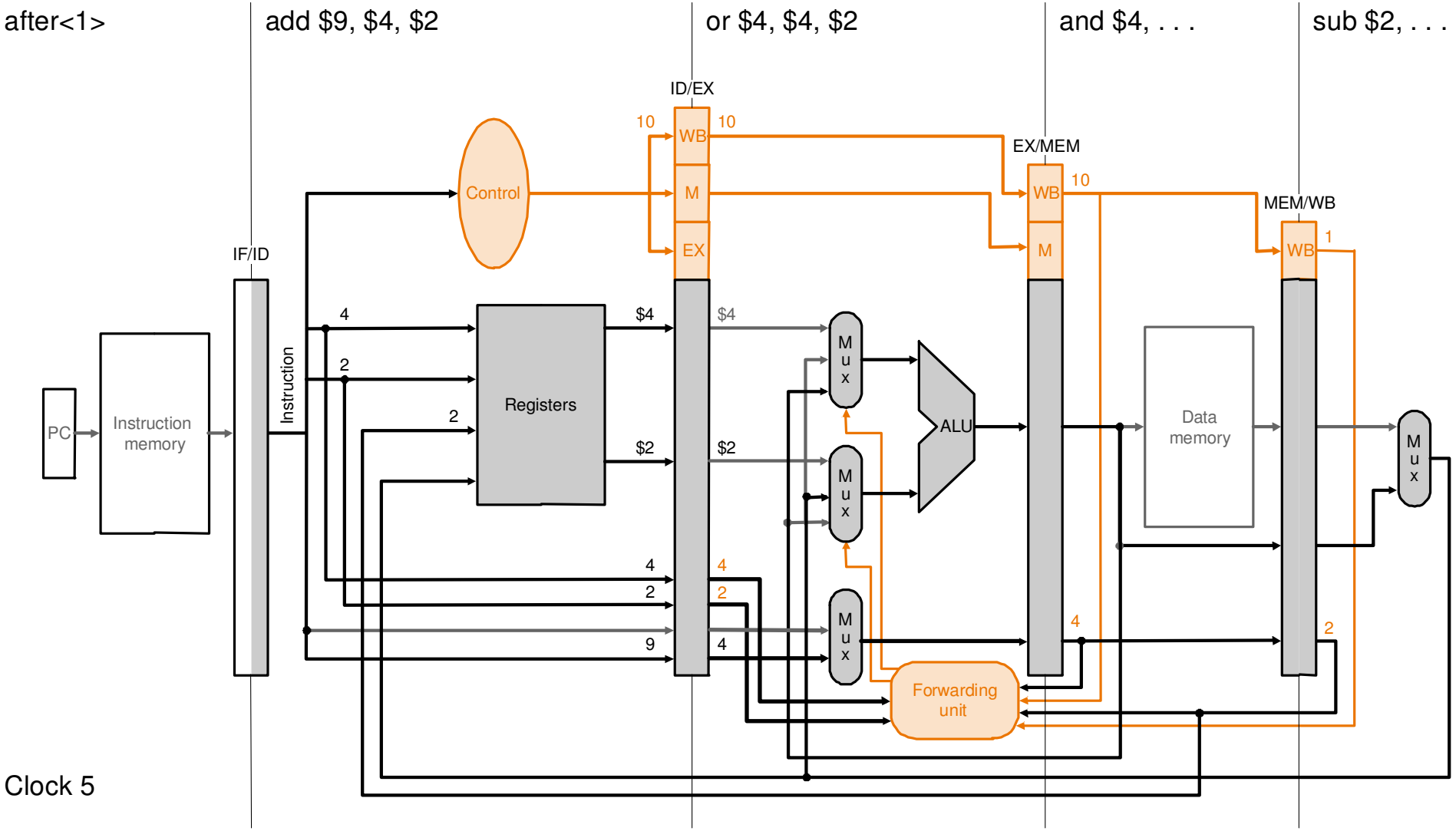
Forwarding

```
sub   $2, $1, $3
and   $4, $2, $5
or    $4, $4, $2
add   $9, $4, $2
```



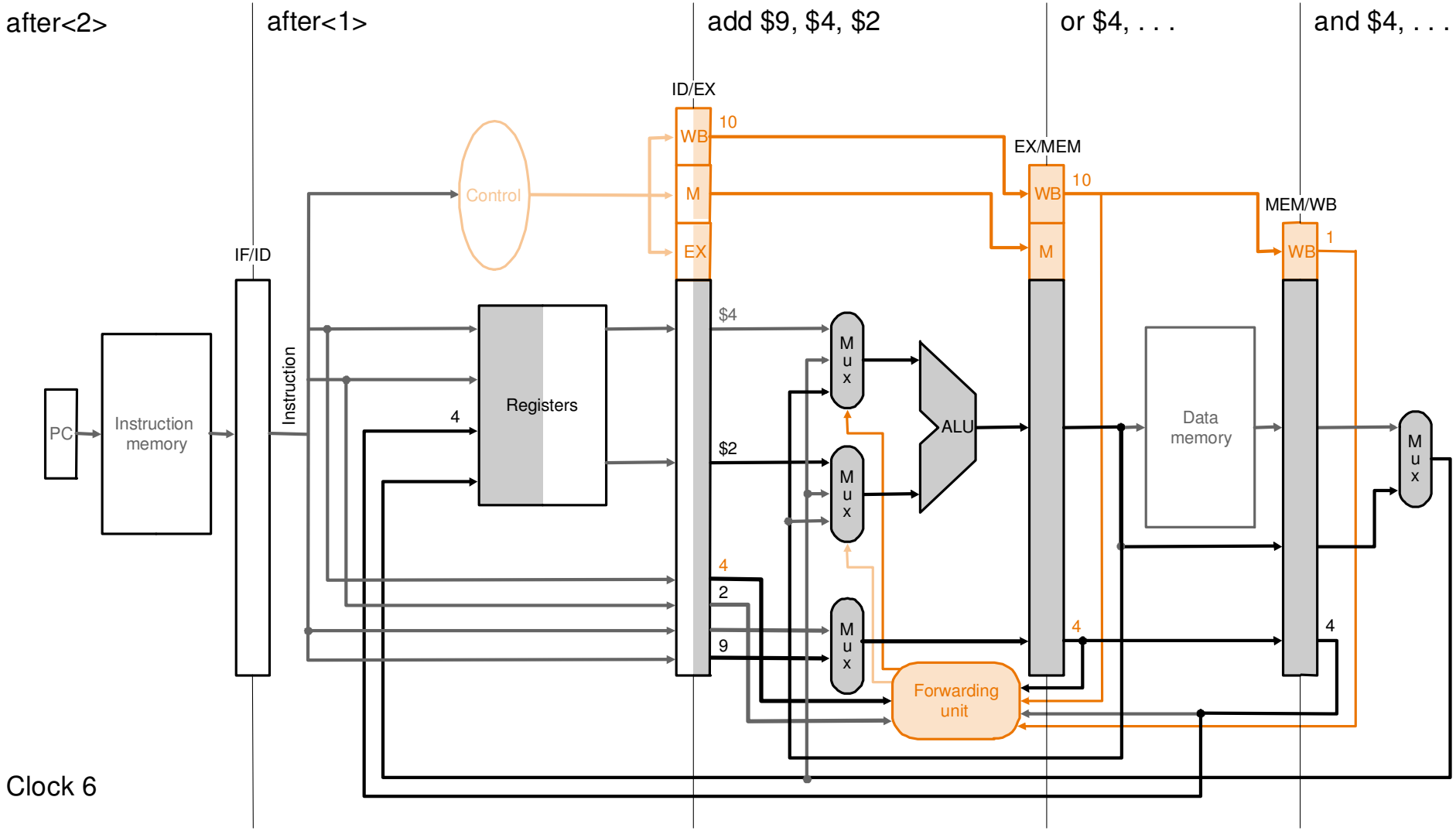
Forwarding

```
sub   $2, $1, $3
and   $4, $2, $5
or    $4, $4, $2
add   $9, $4, $2
```

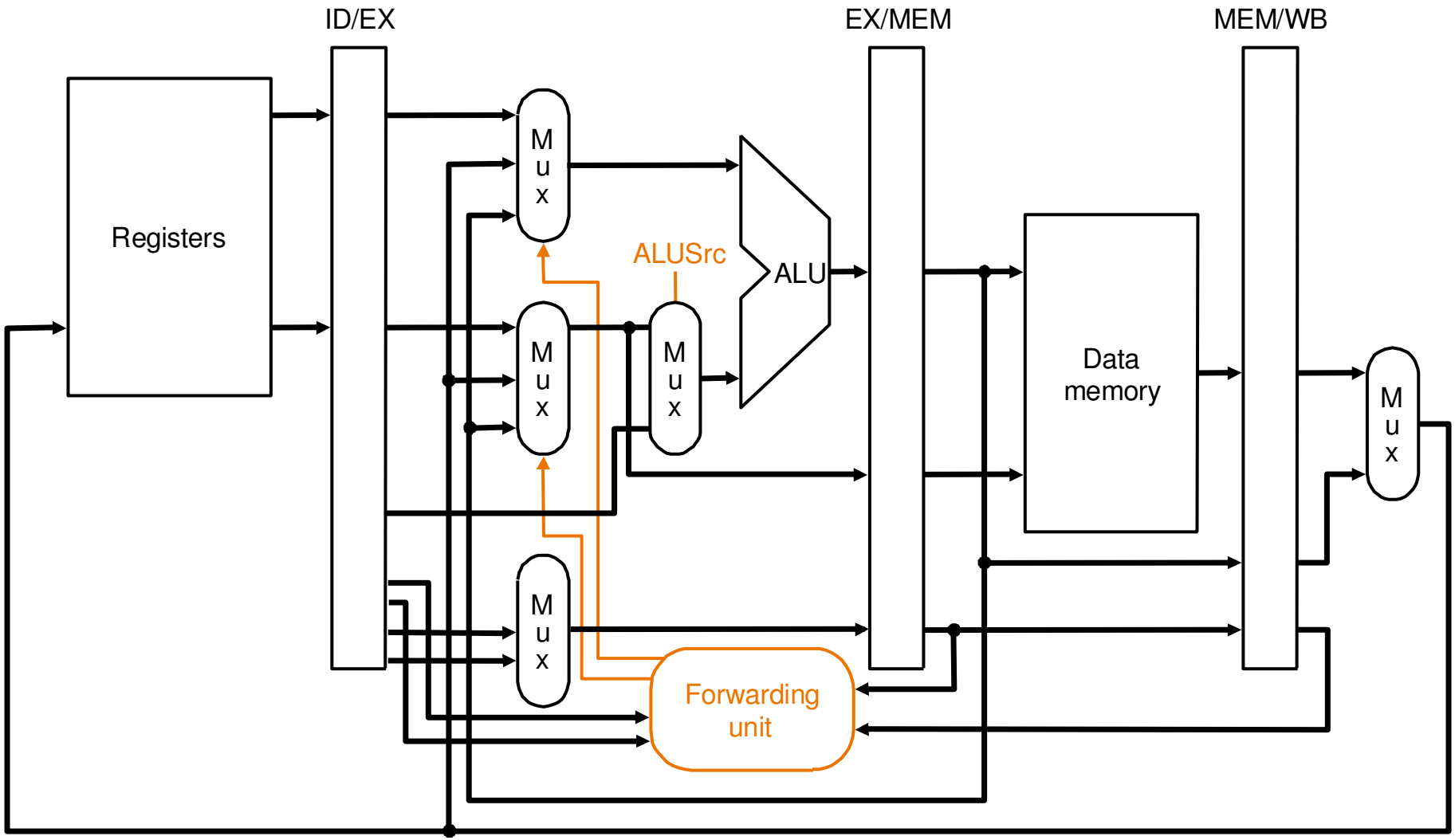


Forwarding

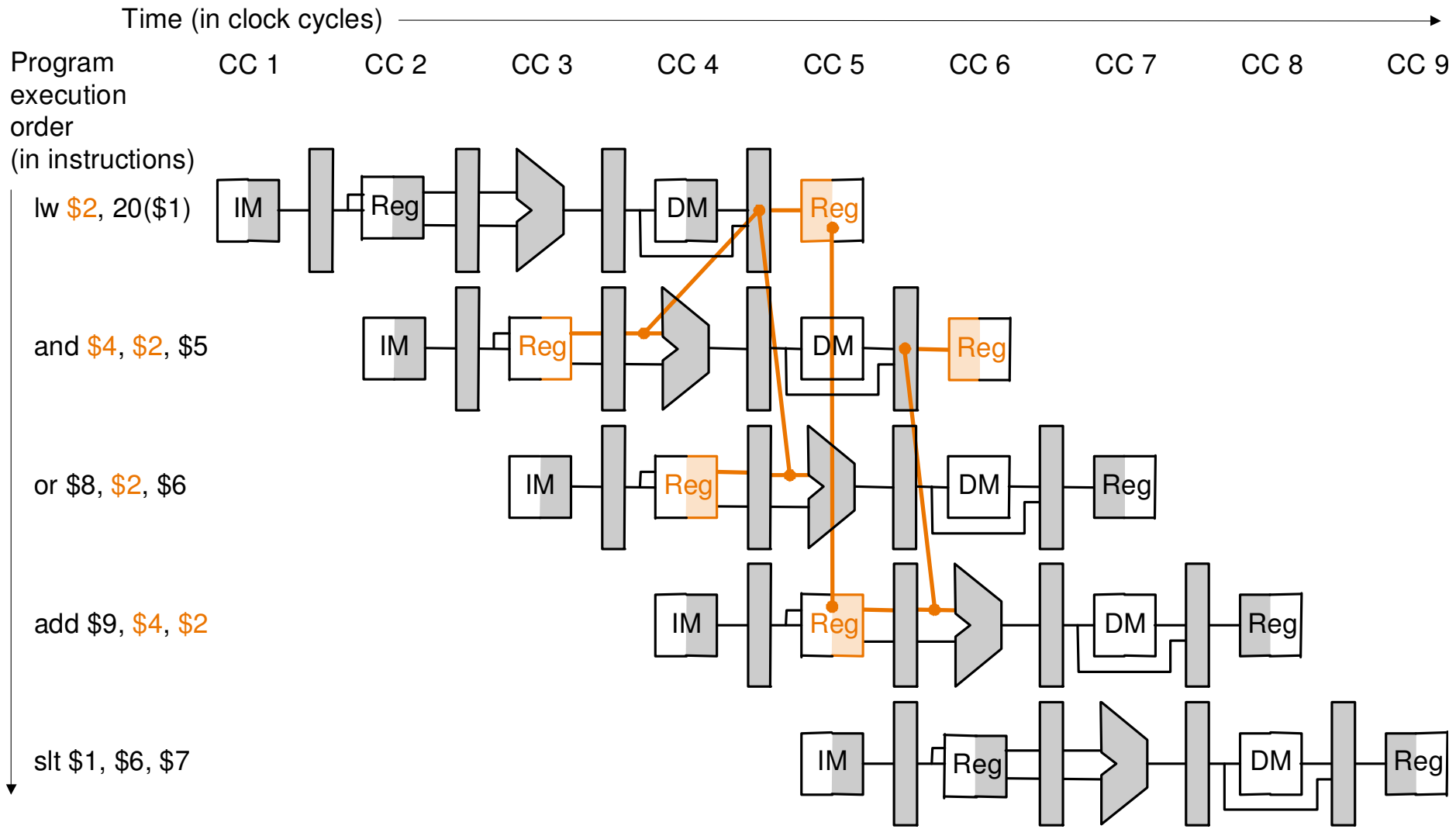
```
sub   $2, $1, $3
and   $4, $2, $5
or    $4, $4, $2
add   $9, $4, $2
```



Forwarding: Datapath para Entrada Signed-imediate Necessária para lw e sw



Data Hazard e Stalls



Data Hazards e Stalls

- Quando uma instrução tenta ler um registrador precedida por uma instrução de load, que escreve no mesmo registrador ➔ o dado tem que ser mantido (ciclo 4) enquanto a ULA executa a operação ➔ atrasar o pipeline para que a instrução leia o valor correto.

- Condição de detecção de hazard para atraso no pipeline

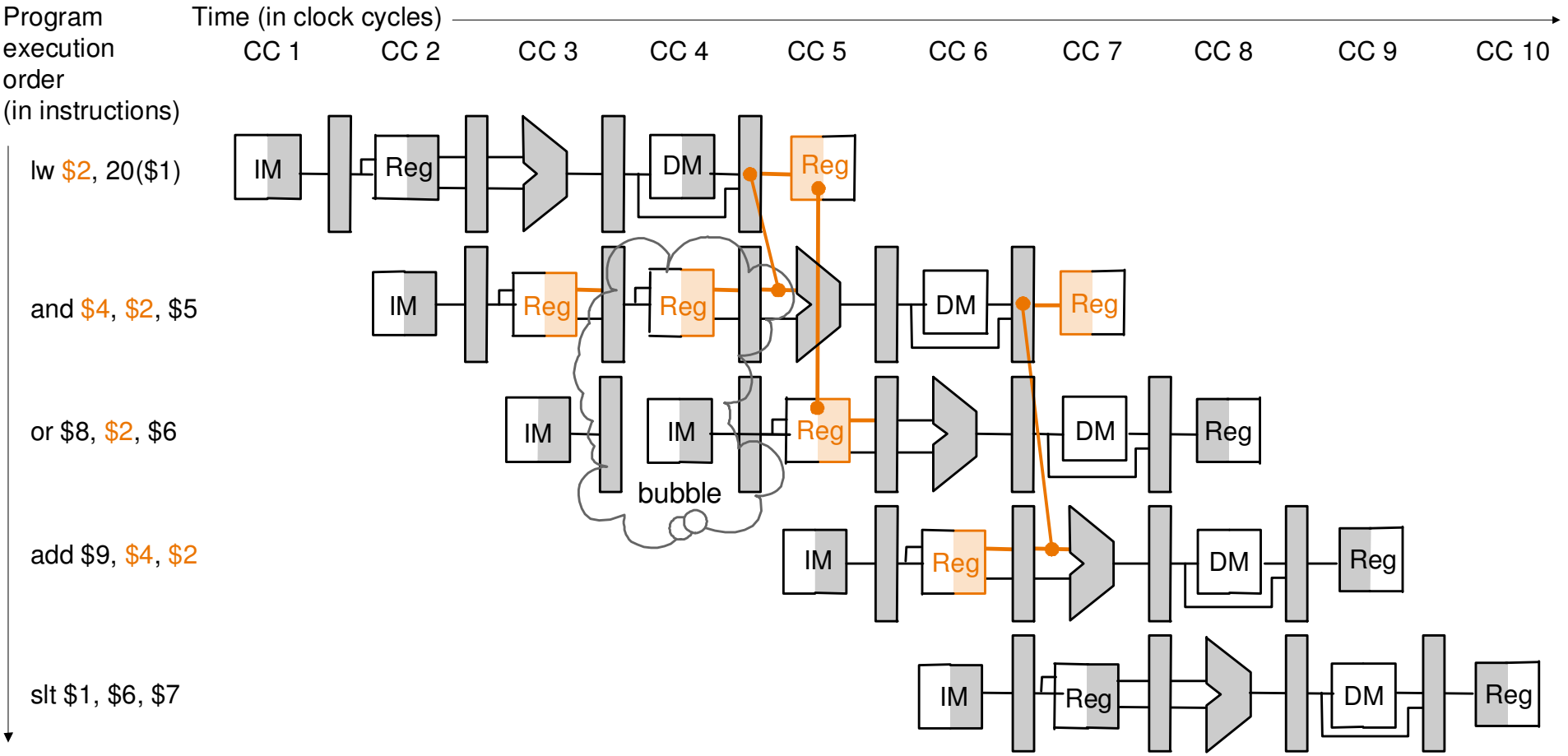
 # testa se é um load:

if (ID/EX.MemRead and

 # verifica se registrador destino da instrução load em EX é o registrador fonte da instrução em ID:

((ID/EX.RegisterRt = IF/ID.RegisterRs) or
(ID/EX.RegisterRt = IF/ID.RegisterRt)))
stall pipeline

Data Hazards e Stalls



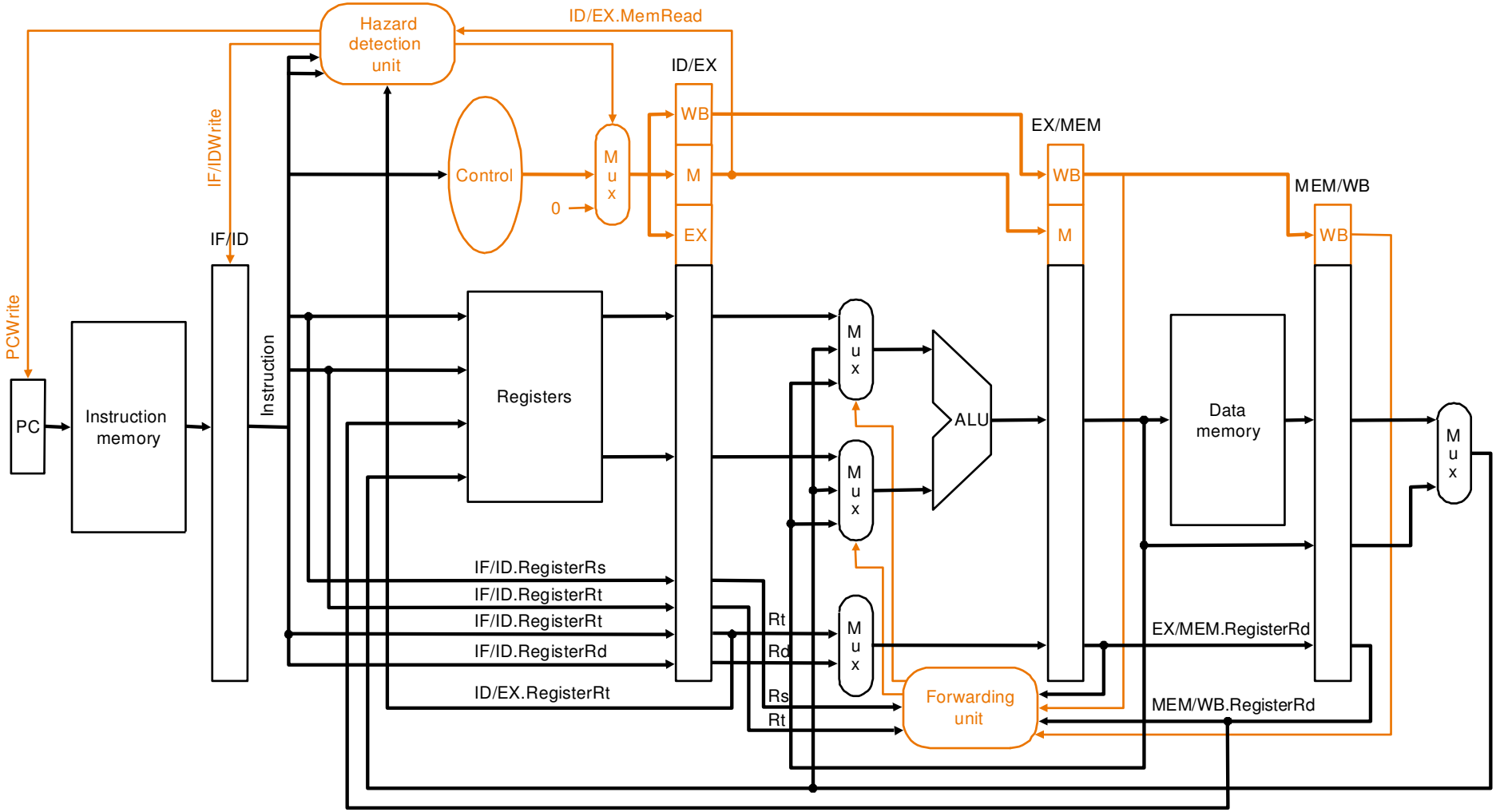
Data Hazards e Stalls

- **Se a instrução no estágio ID é atrasada, a instrução no estágio IF também deve ser atrasada**

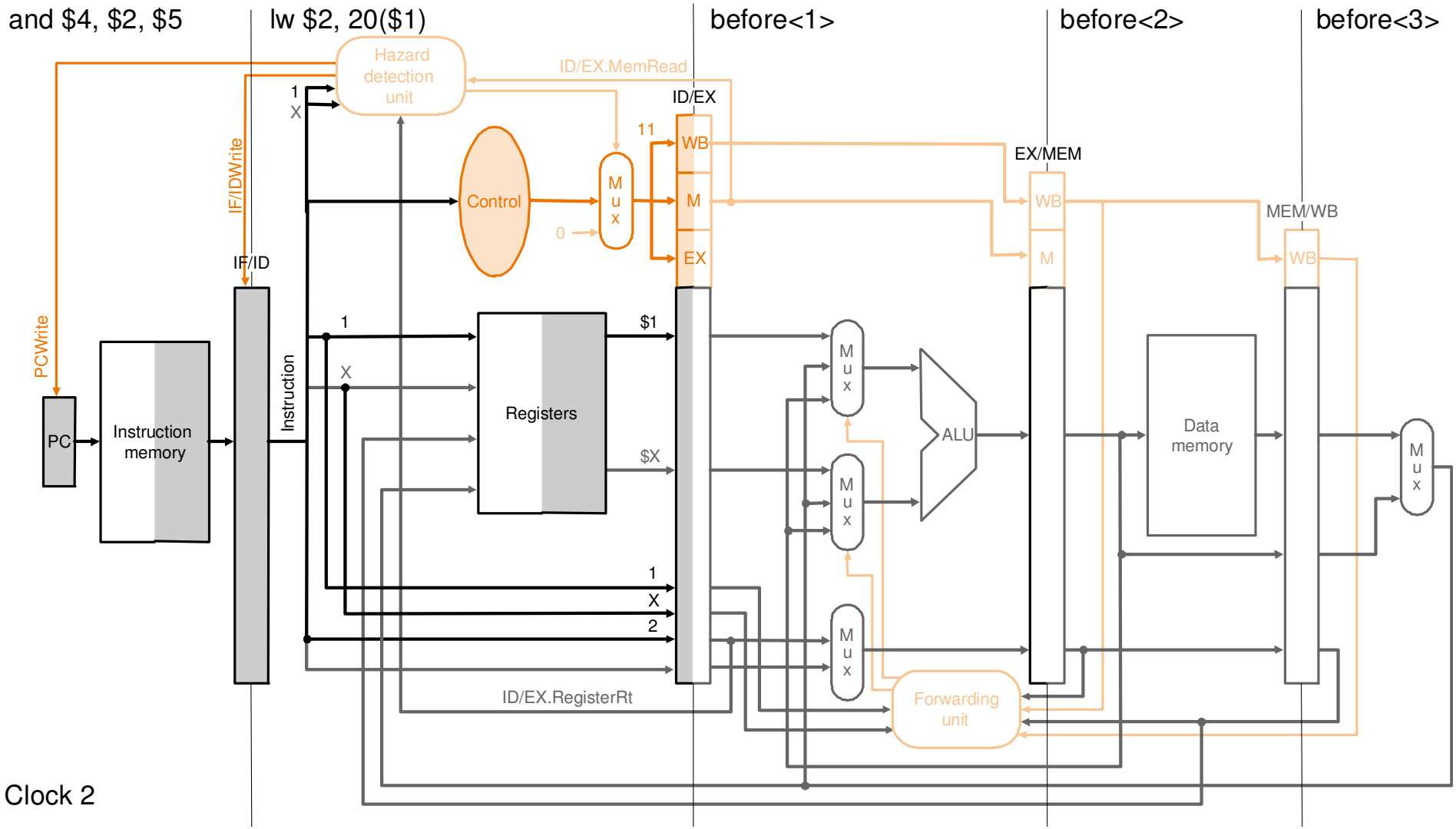
- **Como fazer?**
  Impedir a mudança no PC e no registrador IF/IF. A instrução em IF continua sendo lida e em ID continuam sendo lido os mesmos campos da instrução.

- **Stall pipeline: mesmo efeito da instrução nop começando pelo estágio EX → desativar os 9 sinais de controle dos estágios EX, MEM e WB**

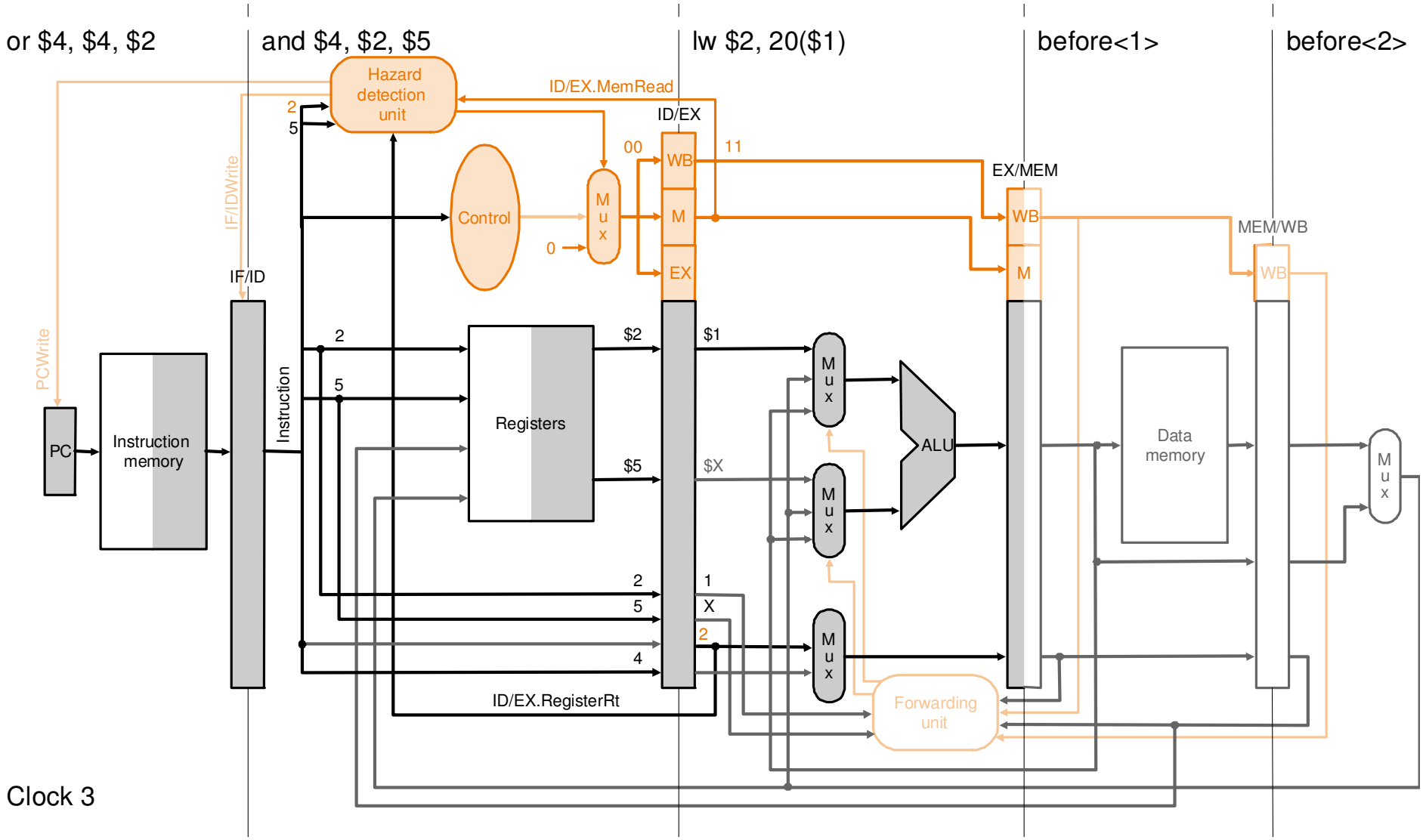
Datapath com forwarding e data hazard detection



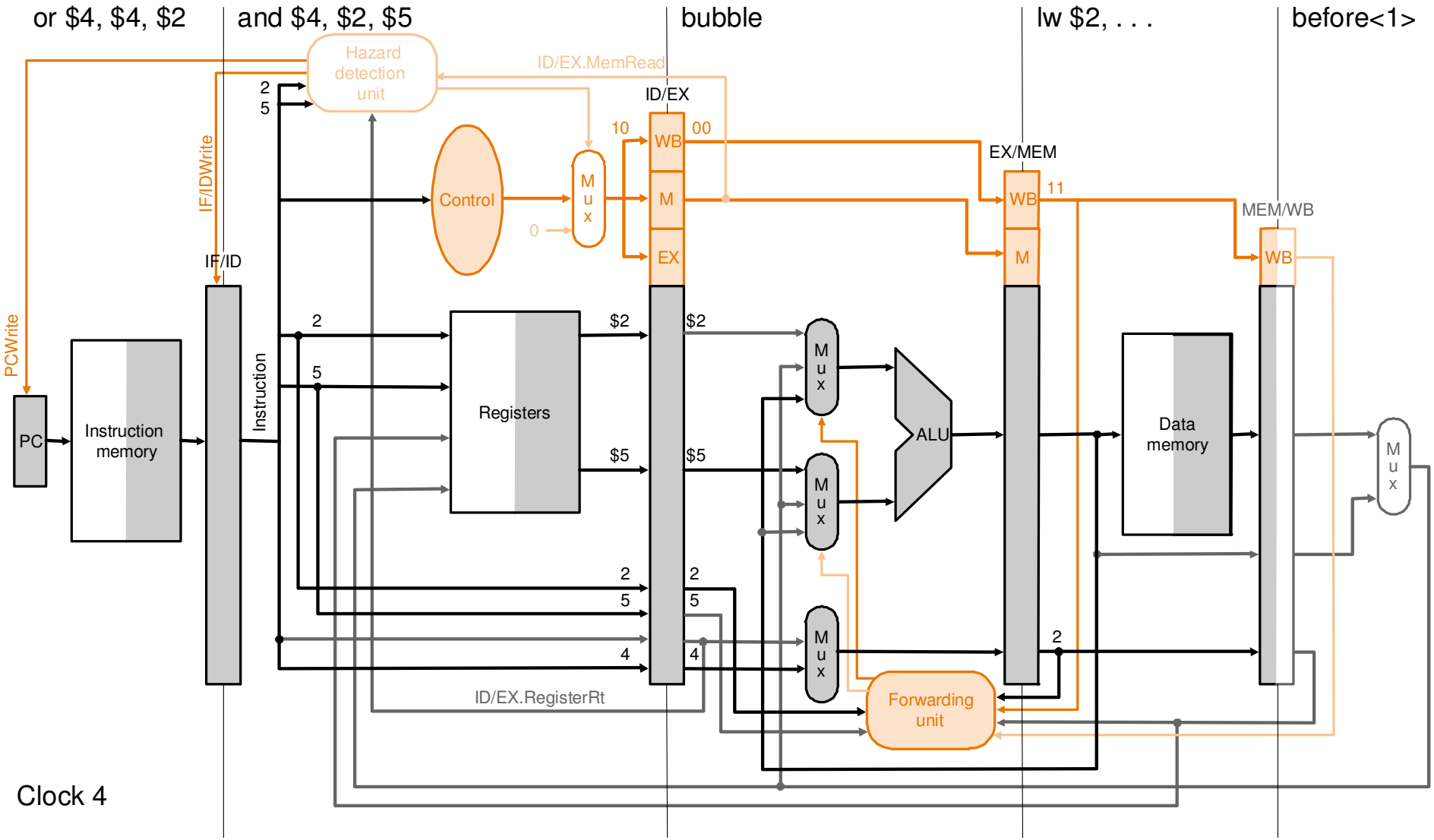
Seqüência de Execução do Exemplo Anterior



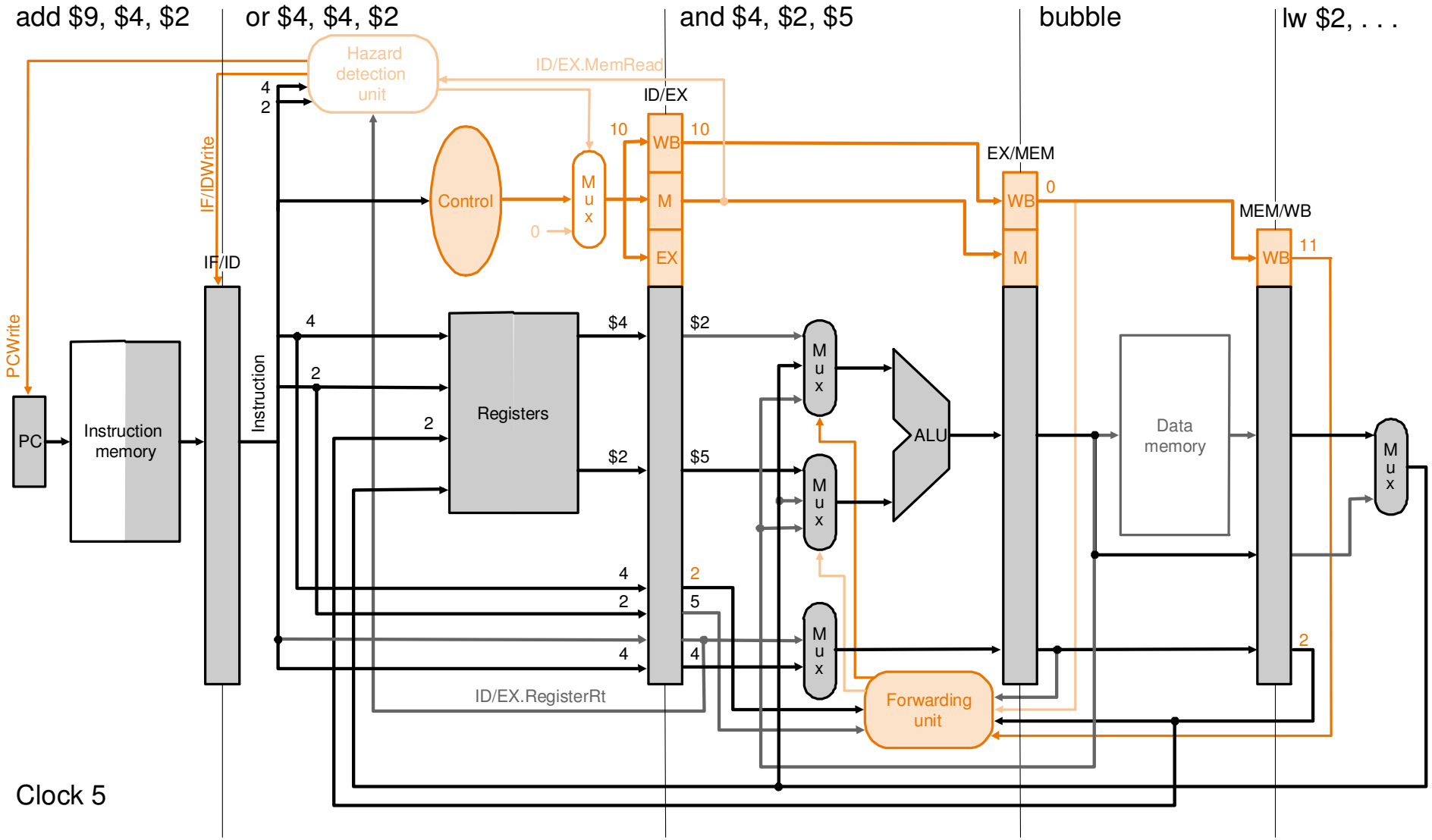
Seqüência de Execução do Exemplo Anterior



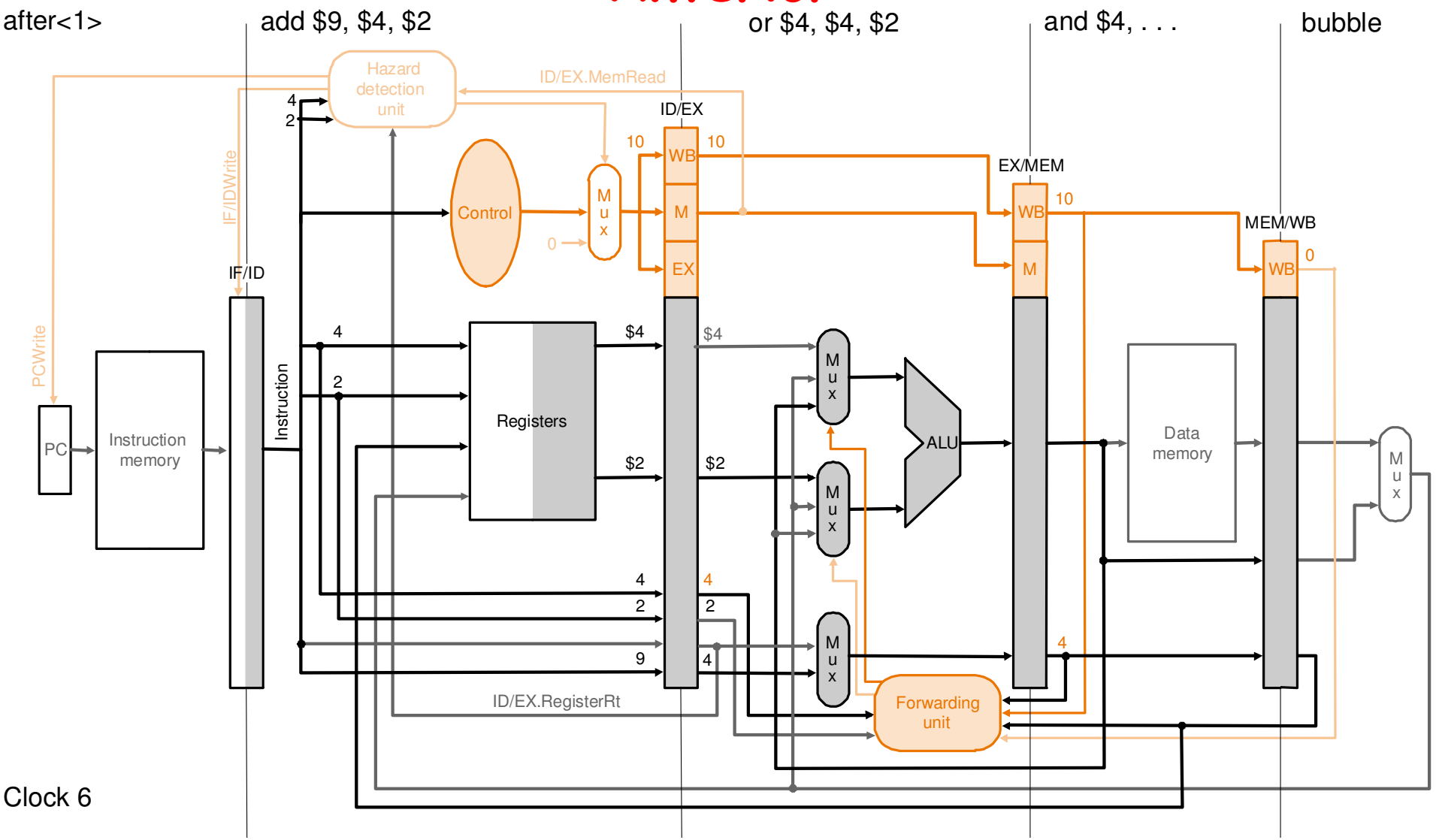
Seqüência de Execução do Exemplo Anterior



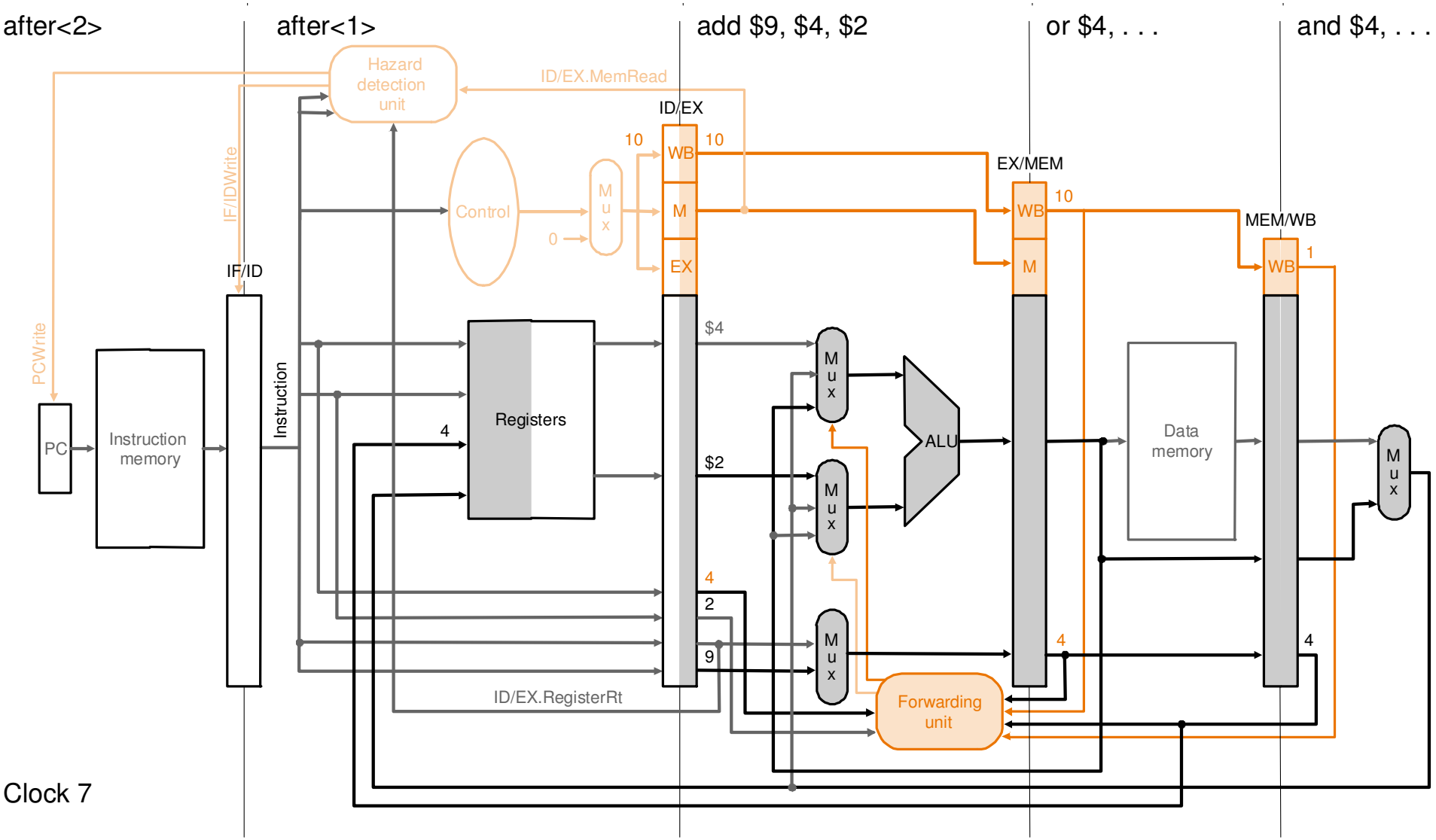
Seqüência de Execução do Exemplo Anterior



Seqüência de Execução do Exemplo Anterior

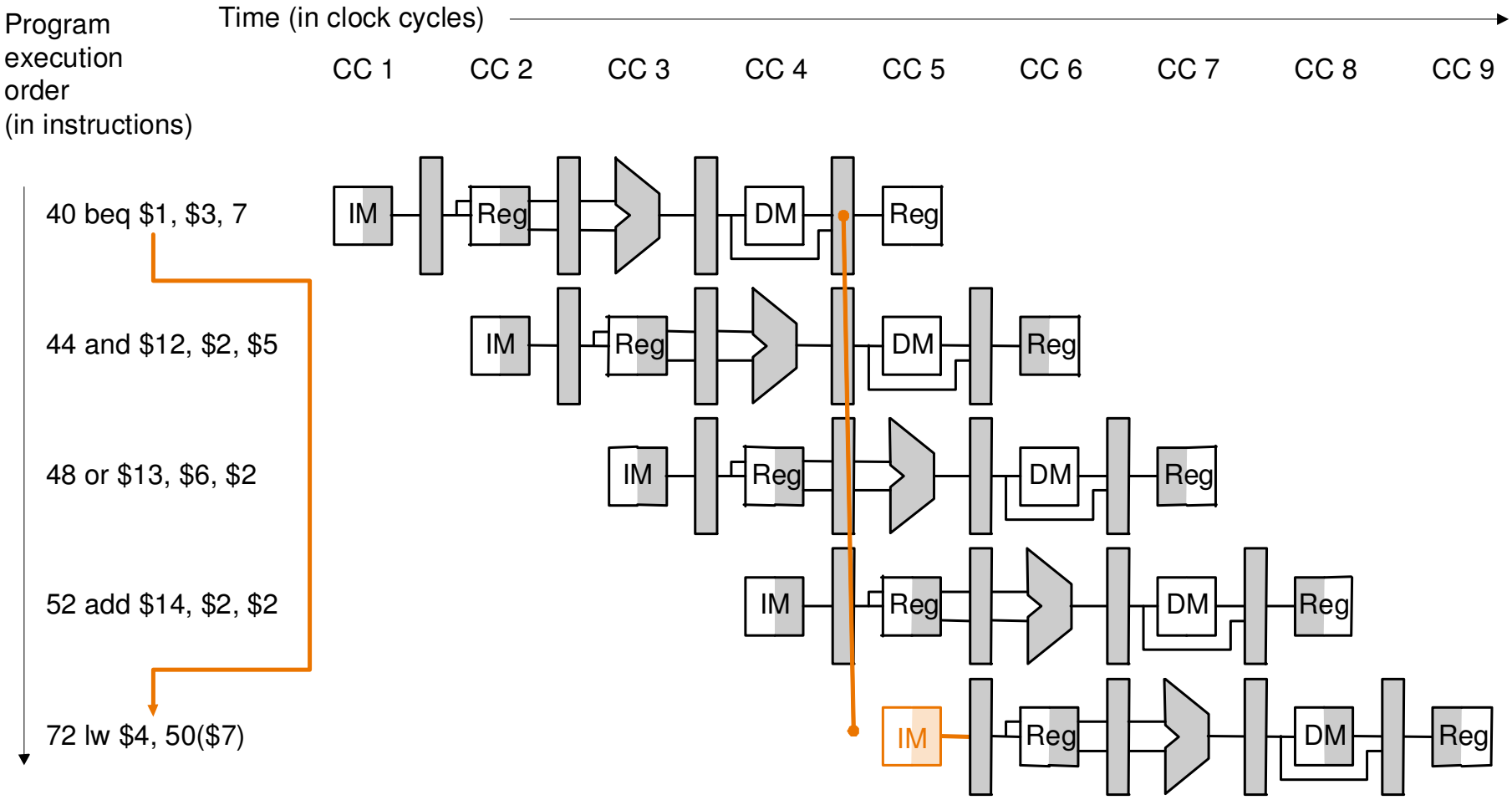


Seqüência de Execução do Exemplo Anterior



Branch Hazards

- **Devemos ter um fetch de instrução por ciclo de clock, decisão: qual caminho de um branch deve ocorrer até o estágio MEM.**



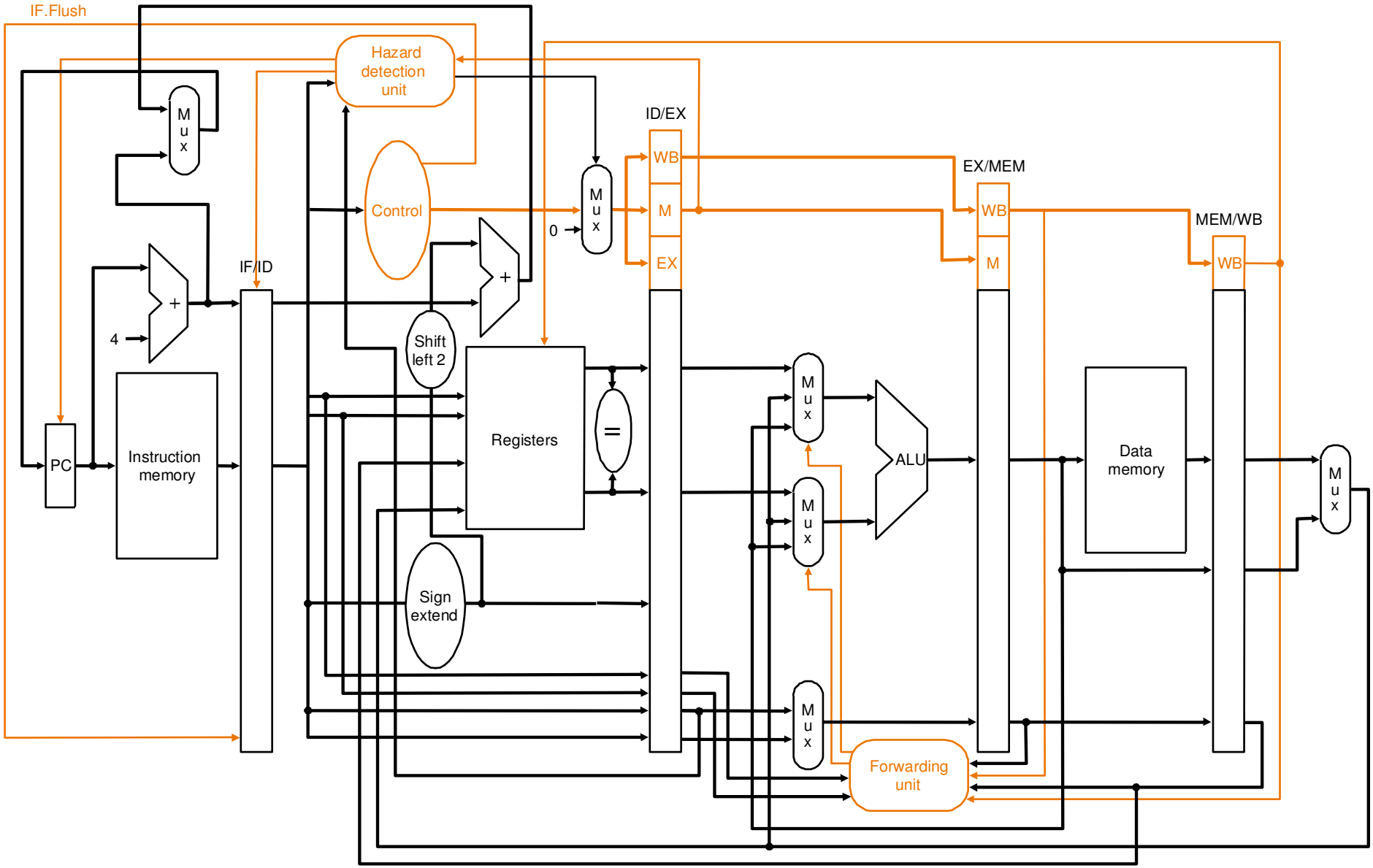
Branch Hazards

- **Tês esquemas para resolver control hazard :**
  - **Branch-delay Slots**
  - **Assume Branch Not Taken**
  - **Dynamic Branch Prediction**
- **Assume Branch Not Taken**
  - **continua a execução seqüencialmente e se o branch for tomado, descarta as instruções entre a instrução de branch e a instrução no endereço alvo, fazendo seus sinais de controle iguais a zero**

Branch Hazards

- **Redução do atraso de branches**
  - **reduzir o custo se o branch for tomado**
  - **adiantar a execução de uma instrução de branch.**
  - **O next PC para uma instrução de branch é selecionado no estágio MEM**
  - **executar o branch no estágio ID (apenas uma instrução será descartada)**
  - **deslocar o cálculo do endereço de branch (branch adder) do MEM para o ID e comparando os registradores lidos do register file.**

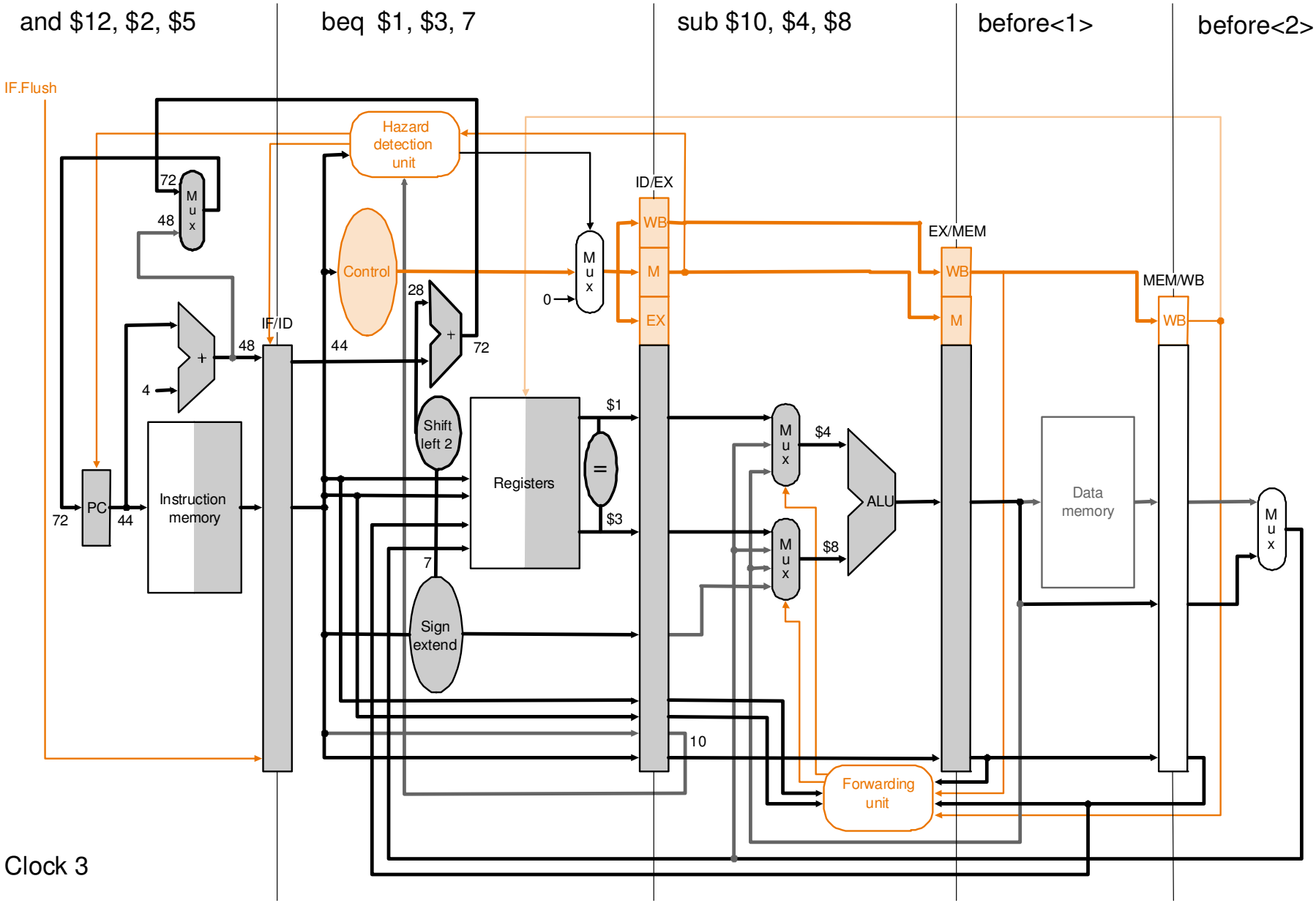
Branch Hazards



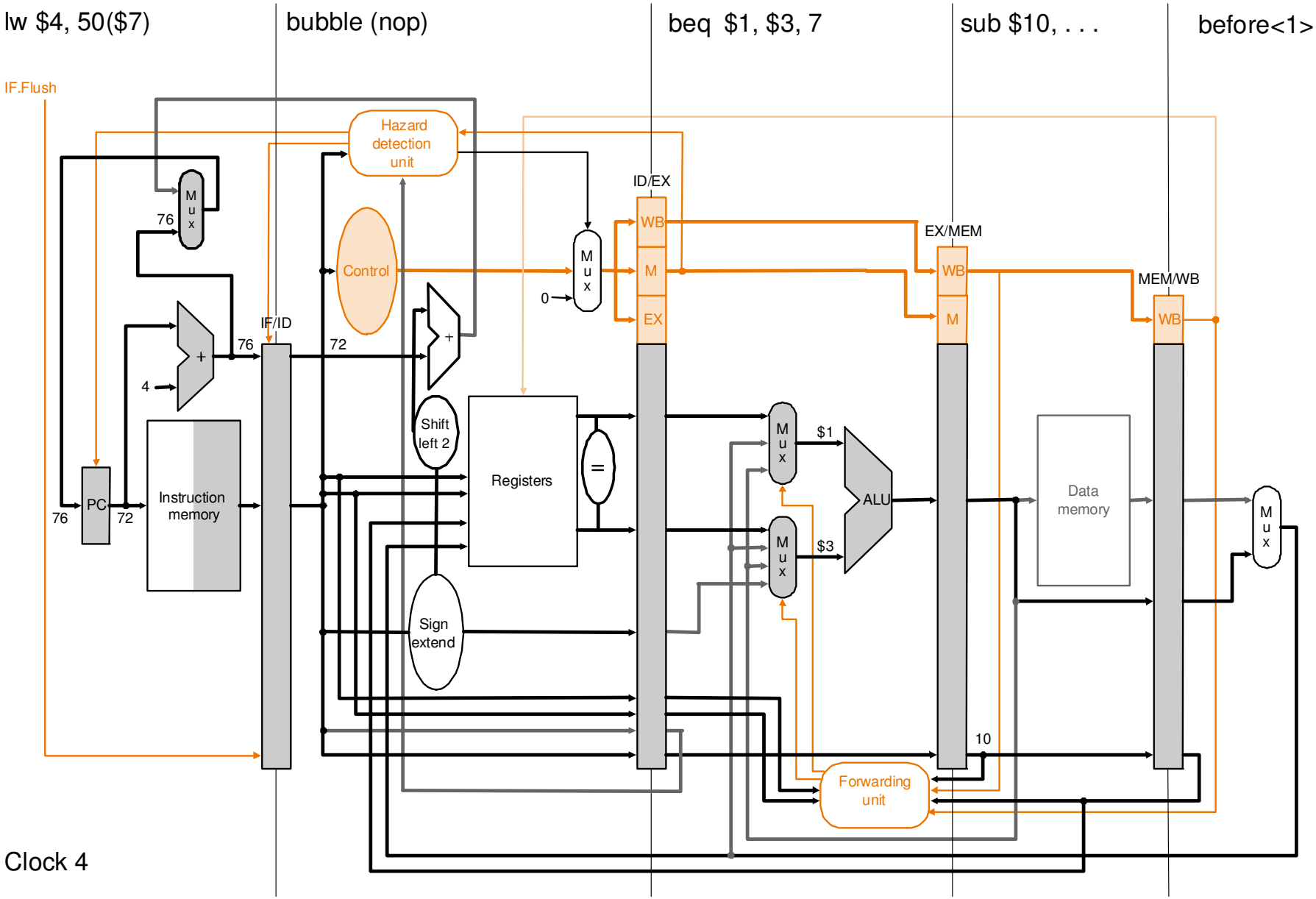
Branch Hazards (exemplo)

```
36  sub  $10, $4, $8
40  beq  $1,   $3, 7      # PC ← 40 + 4 +7*4 = 72
44  and  $12, $2, $5
48  or    $13, $2, $6
52  add  $14, $4, $2
56  slt   $15, $6,  $7
....        ............
....        ............
72  lw  $4,     50($7)
```

Branch Hazards (exemplo)



Branch Hazards (exemplo)



Dymanic Branch Prediction

- Branch not taken
  - forma de branch predicton que assume que o branch não será tomado.
- Dymanic Branch Prediction
  - descobre se o branch foi tomado ou não na última vez que foi executado e faz o fetch das instruções pelo mesmo local da última vez.

Dymanic Branch Prediction

- Implementação
  - branch prediction buffer
  - branch history table

- Branch prediction buffer: pequena memória indexada por bits menos significativos do endereço da instrução de branch. Ela contém um bit que diz se o branch foi recentemente tomado ou não (Neste esquema não sabemos se a previsão é correta ou não, pois este buffer pode ser alterado por outra instrução de branch que tem os mesmos bits menos significativos de endereço).