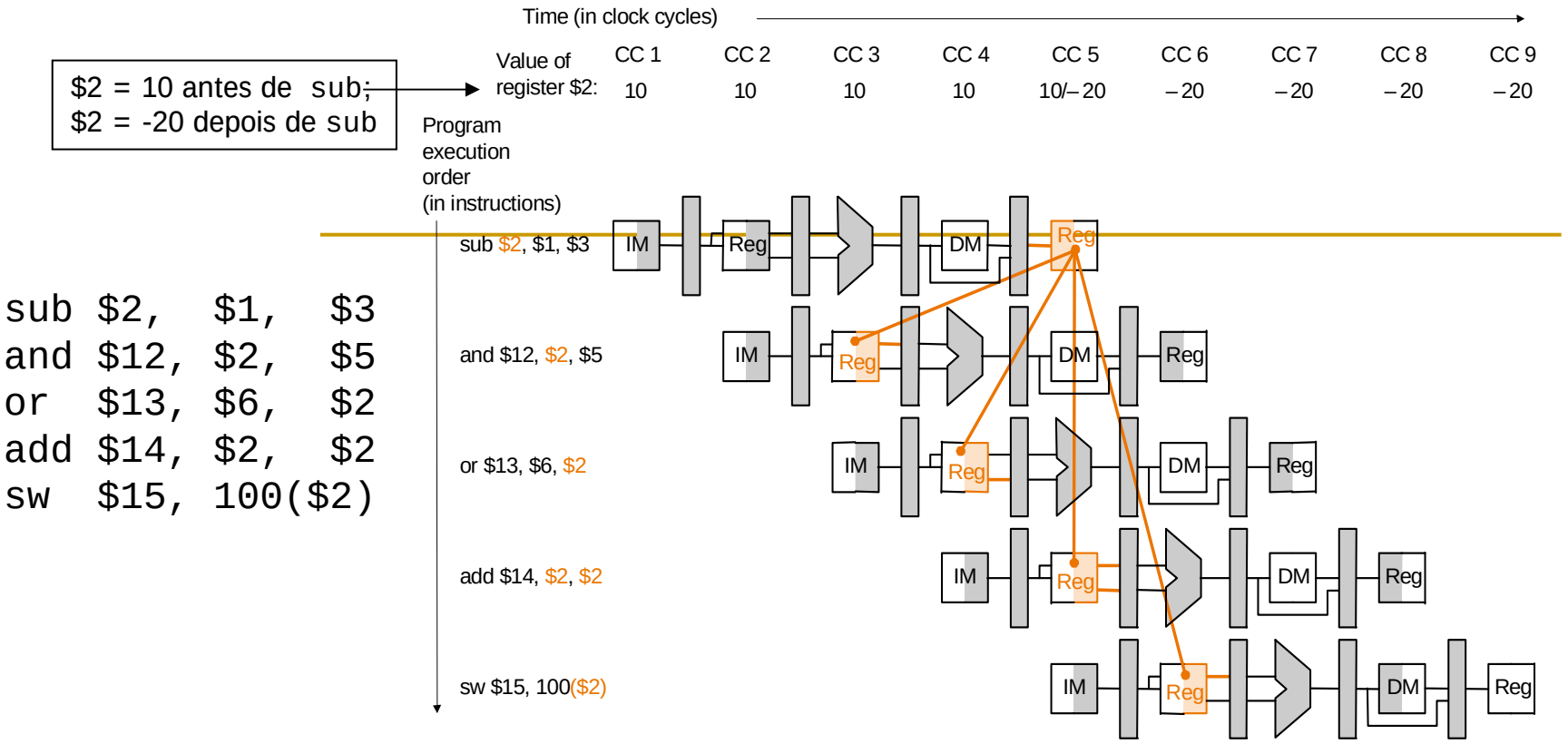
Arquiteturas de Computadores

Pipeline

Fontes dos slides: Patterson & Hennessy book website (copyright Morgan Kaufmann) e Dr. Sumanta Guha

Conflitos de Dados e Encaminhamento

- Problema que ocorre quando começa uma instrução sem acabar a precedente:
  - Dependência de dados que devem vir para trás– chamados conflitos de dados




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

Solução de Software

- Compilador garante que nunca haverá conflito de dados!
  - *rearrumando as instruções para inserir instruções independentes* entre as instruções que poderiam ter conflito de dados entre elas
  - ou, se a rearrumação não é possível, inserir nops

```
sub   $2,  $1, $3          sub   $2,  $1, $3
lw    $10, 40($3)          nop
slt   $5,  $6,  $7         nop
and   $12, $2,  $5    or   and   $12, $2,  $5
or    $13, $6,  $2         or    $13, $6,  $2
add   $14, $2,  $2         add   $14, $2,  $2
sw    $15, 100($2)         sw    $15, 100($2)
```

- Estas soluções podem nem sempre ser possíveis, e nops atrasam a execução

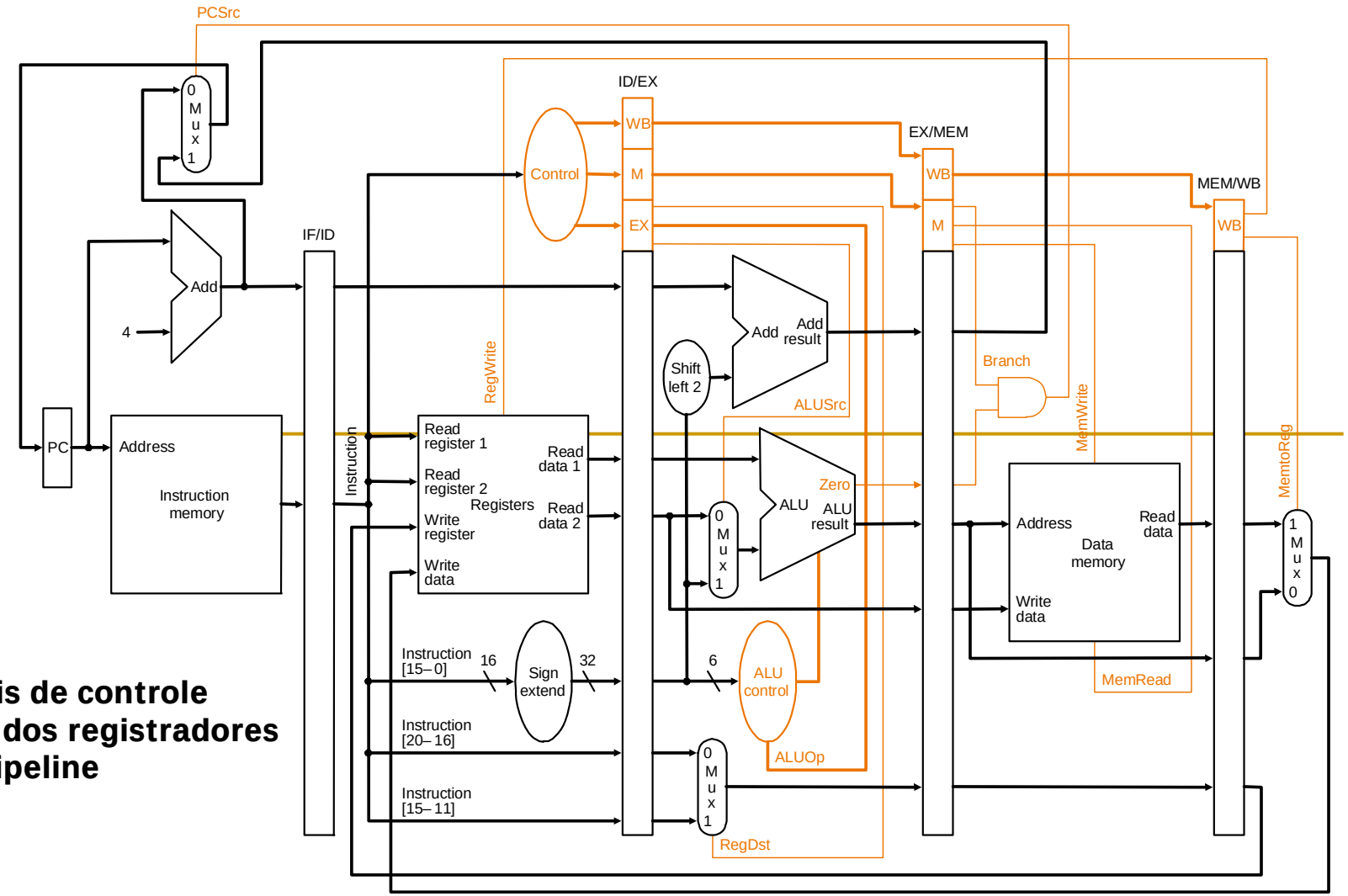
MIPS: nop = “no operation” = 00…0 (32bits) = sll $0, $0, 0

Solução de Hardware: Encaminhamento

Ideia: *usar dados intermediários*, não esperar que o resultado seja escrito no registrador. Dois passos:

1. *Detectar* conflito de dados
2. *Encaminhar dado intermediário* para resolver o problema

Controle do Pipeline



Sinais de controle vem dos registradores de pipeline

Detecção de conflito

- Condições de conflito:

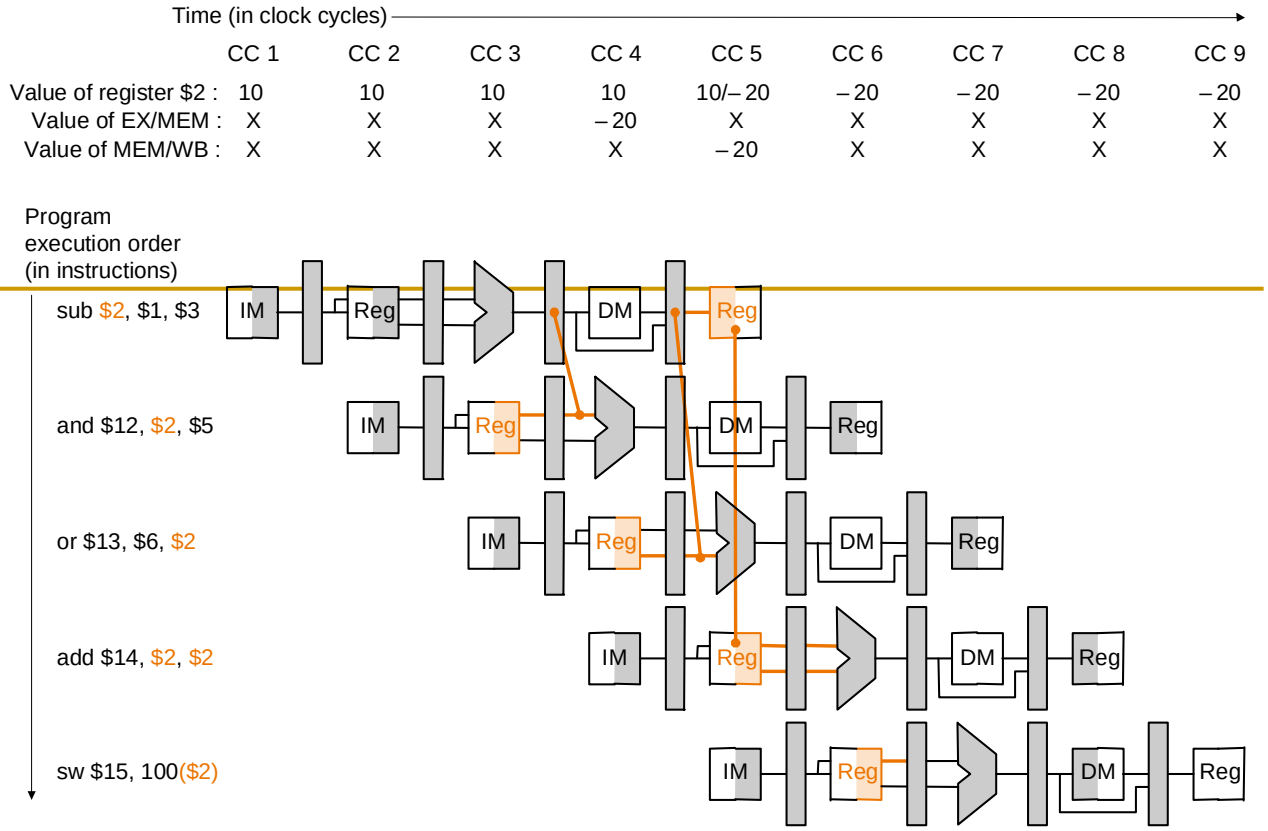
1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

  - No exemplo anterior, primeiro conflito entre `sub $2, $1, $3` e `and $12, $2, $5` é detectado quando `and` está no estágio EX e `sub` está no estágio MEM porque
    - `EX/MEM.RegisterRd = ID/EX.RegisterRs = $2 (1a)`

- Deve haver encaminhamento:
  - *Se a instrução anterior for escrever em registrador* – se não, não necessita encaminhamento, mesmo se alguma das condições acima ocorrer
  - *Se o destino de escrita da instrução anterior é $0* – neste caso não precisa encaminhamento ($0 é sempre 0 e nunca é sobreescrito)

Encaminhamento de dados

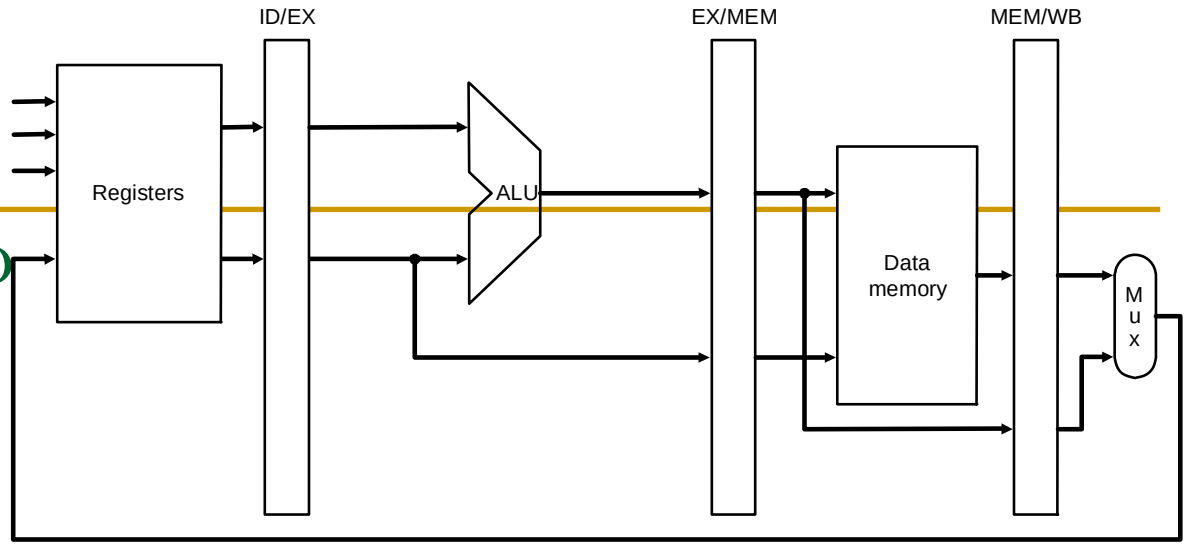
- Ideia:
  - *Permitir entradas na ALU não apenas ID/EX, mas também de registradores de pipeline mais para frente but also later pipeline registers*, e
  - *usar multiplexadores e sinais de controle para escolher as entradas apropriadas para* ALU



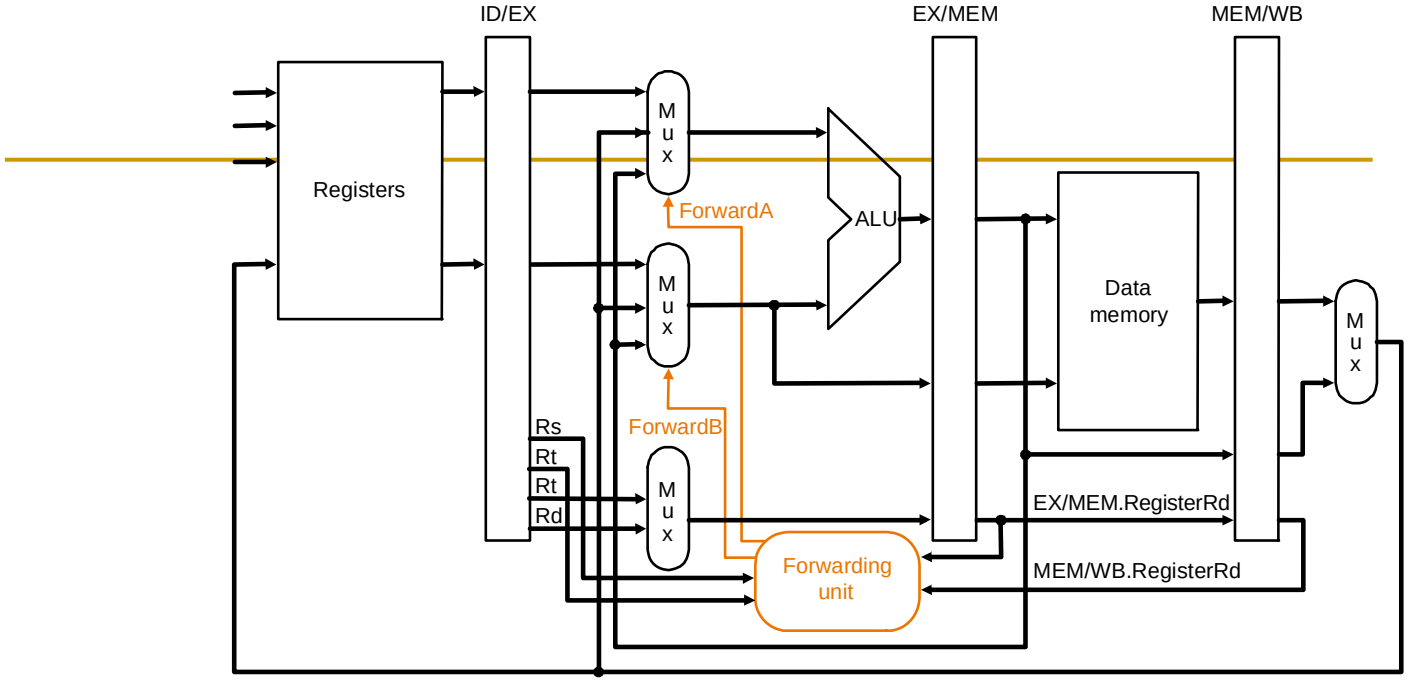

```
sub $2,   $1,   $3
and $12,  $2,   $5
or  $13,  $6,   $2
add $14,  $2,   $2
sw  $15,  100($2)
```

Dependências entre pipelines movem para frente no tempo

Hardware de Encaminhamento



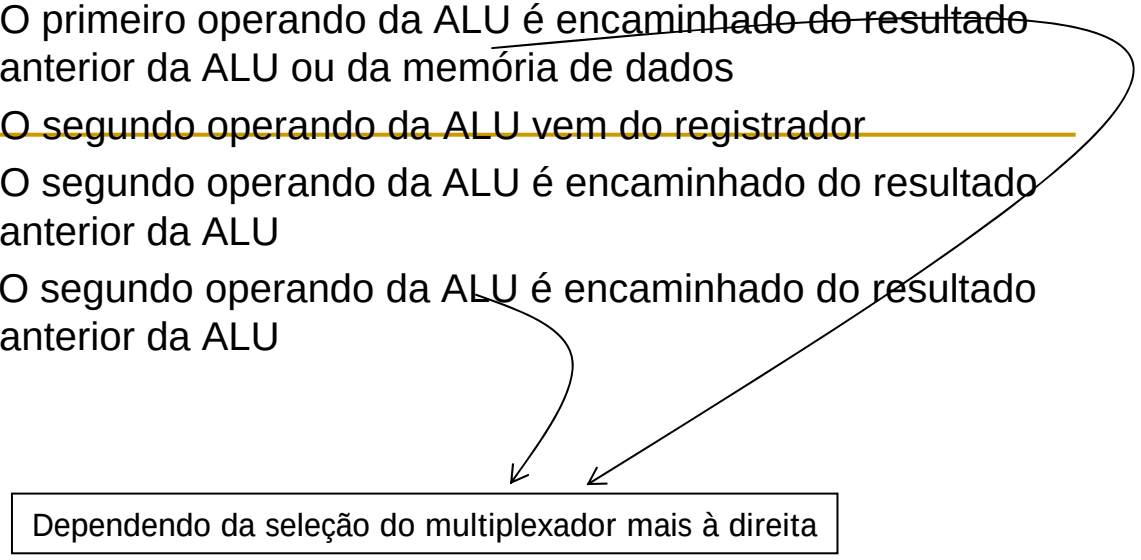
Sem hardware



Com hardware

Encaminhamento de Hardware: Controle do Multiplexador

Contr. do Mux	Fonte	Explicação
ForwardA = 00	ID/EX	O primeiro operando da ALU vem do registrador
ForwardA = 10	EX/MEM	O primeiro operando da ALU é encaminhado do resultado anterior da ALU
ForwardA = 01	MEM/WB	O primeiro operando da ALU é encaminhado do resultado anterior da ALU ou da memória de dados
ForwardB = 00	ID/EX	O segundo operando da ALU vem do registrador
ForwardB = 10	EX/MEM	O segundo operando da ALU é encaminhado do resultado anterior da ALU
ForwardB = 01	MEM/WB	O segundo operando da ALU é encaminhado do resultado anterior da ALU



Conflito de dados: Detecção e Encaminhamento

A unidade de encaminhamento determina os sinais dos multiplexadores de acordo com as seguintes regras::

1. **Conflito EX**

```
if (      EX/MEM.RegWrite                          // se existe uma escrita…
   and ( EX/MEM.RegisterRd ≠ 0 )                    // em um registrador diferente  de $0…
   and ( EX/MEM.RegisterRd = ID/EX.RegisterRs ) )  //que vai ser usado, então…
     ForwardA = 10

if (      EX/MEM.RegWrite                          // se existe uma escrita…
   and ( EX/MEM.RegisterRd ≠ 0 )                    // em um registrador diferente  de $0…
   and ( EX/MEM.RegisterRd = ID/EX.RegisterRt ) )  // que vai ser usado, então…
     ForwardB = 10
```

Conflito de dados: Detecção e Encaminhamento

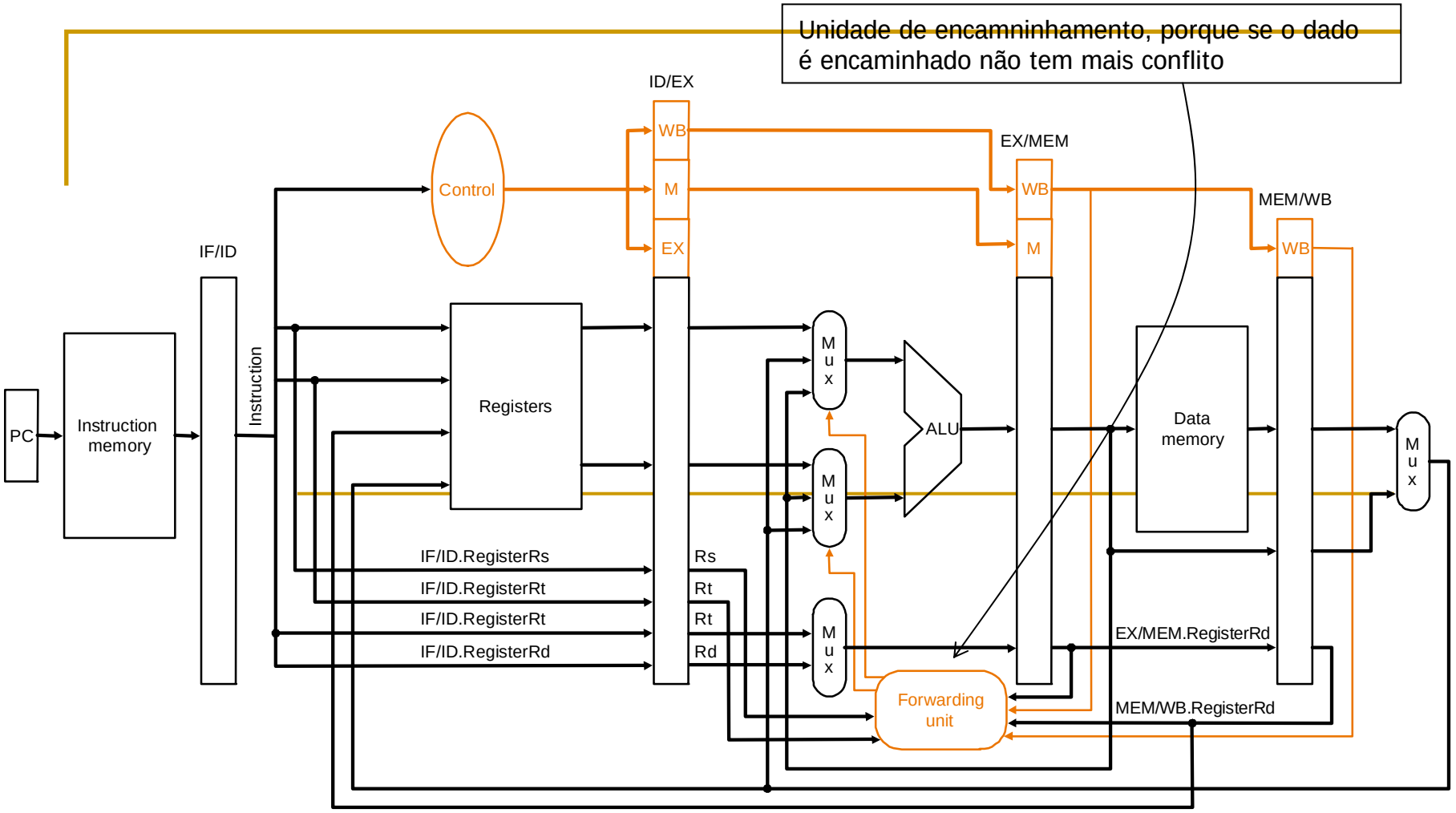
2. **Conflito MEM**

```
if (      MEM/WB.RegWrite                          // se existe uma escrita…
   and ( MEM/WB.RegisterRd ≠ 0 )                    // em um registrador diferente  de $0…
   and ( EX/MEM.RegisterRd ≠ ID/EX.RegisterRs )     // e já não existe conflito de registrador
                                                    // com registrador de pipeline anterior …
   and ( MEM/WB.RegisterRd = ID/EX.RegisterRs ) )   // mas existe com registrador pipeline posterior,
                                                       então …,
      ForwardA = 01

if (      MEM/WB.RegWrite                          // se existe uma escrita…
   and ( MEM/WB.RegisterRd ≠ 0 )                    // em um registrador diferente  de $0…
   and ( EX/MEM.RegisterRd ≠ ID/EX.RegisterRt )     // e já não existe conflito de registrador
                                                    // com registrador de pipeline anterior …
   and ( MEM/WB.RegisterRd = ID/EX.RegisterRt ) )   // mas existe com registrador pipeline posterior,
                                                       então …,
      ForwardB = 01
```

Este teste é necessário, por exemplo, para sequencias como add $1, $1, $2; add $1, $1, $3; add $1, $1, $4; , onde um registrador de pipeline mais à direita (EX/MEM) tem dados mais recentes

Hardware de encaminhamento com controle

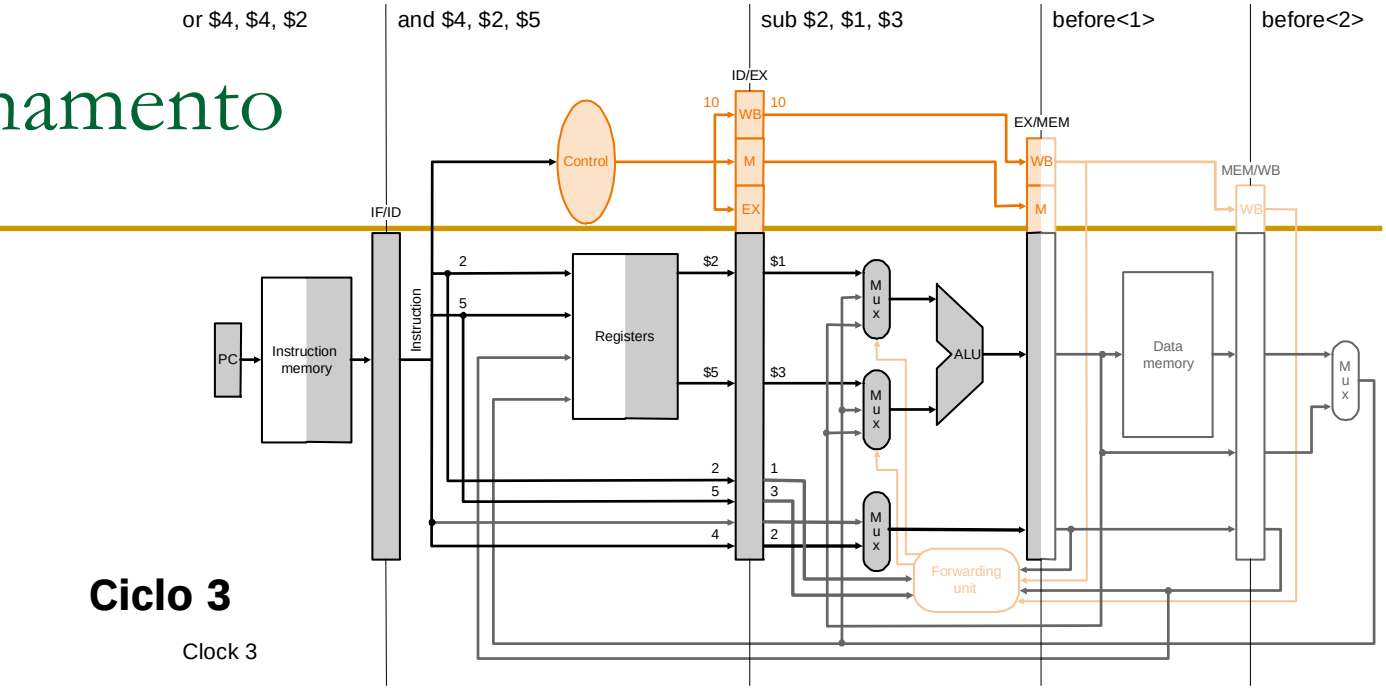


Nota: somente instruções do tipo R !

Encaminhamento

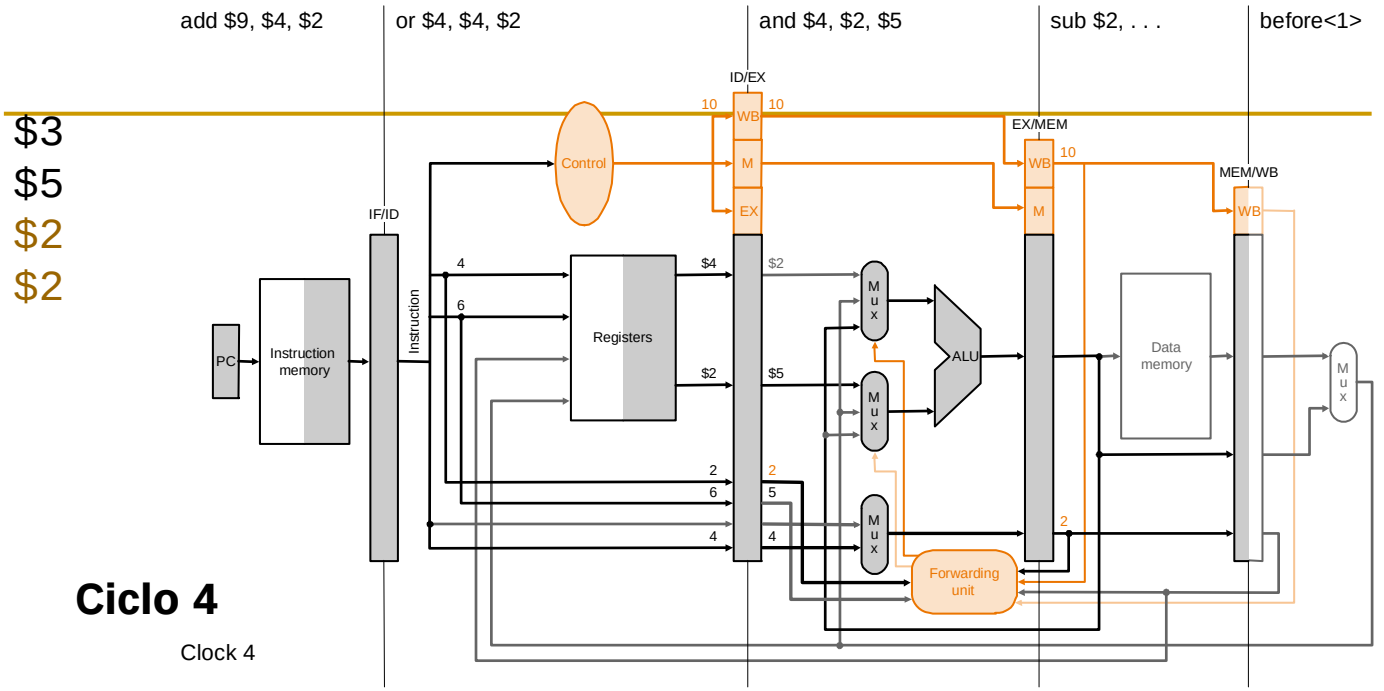
- Exemplo de execução:

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



Ciclo 3

Clock 3



Ciclo 4

Clock 4

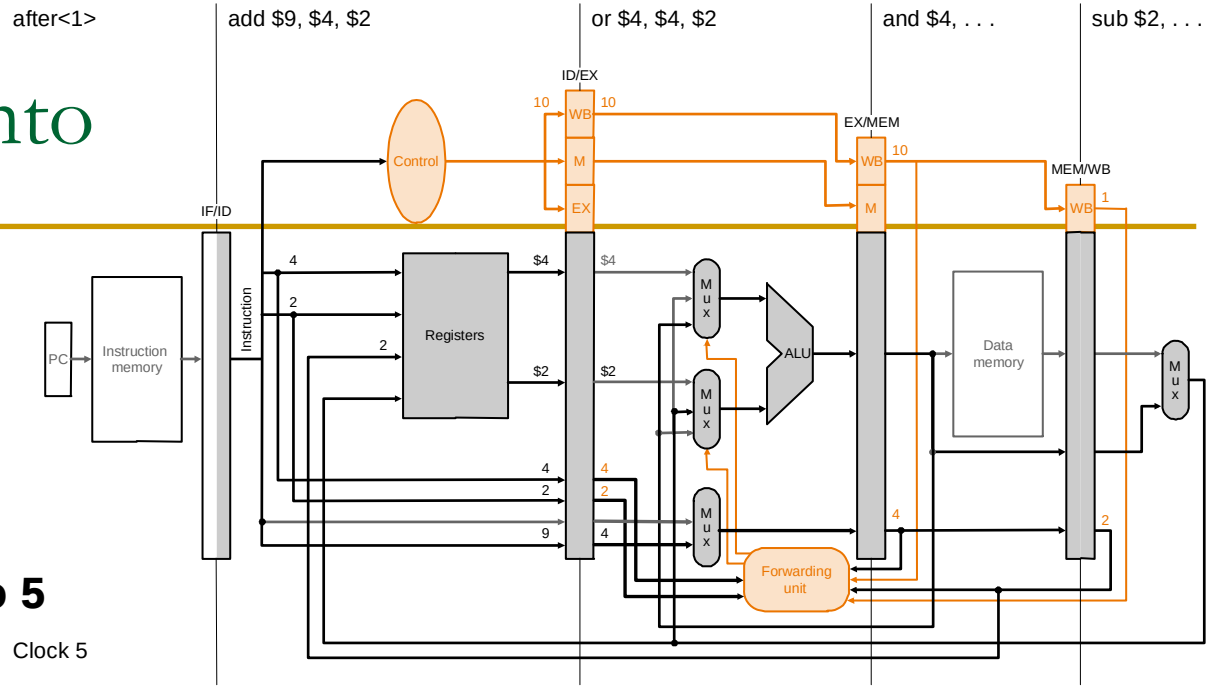
Encaminhamento

- Execution example (cont.):

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

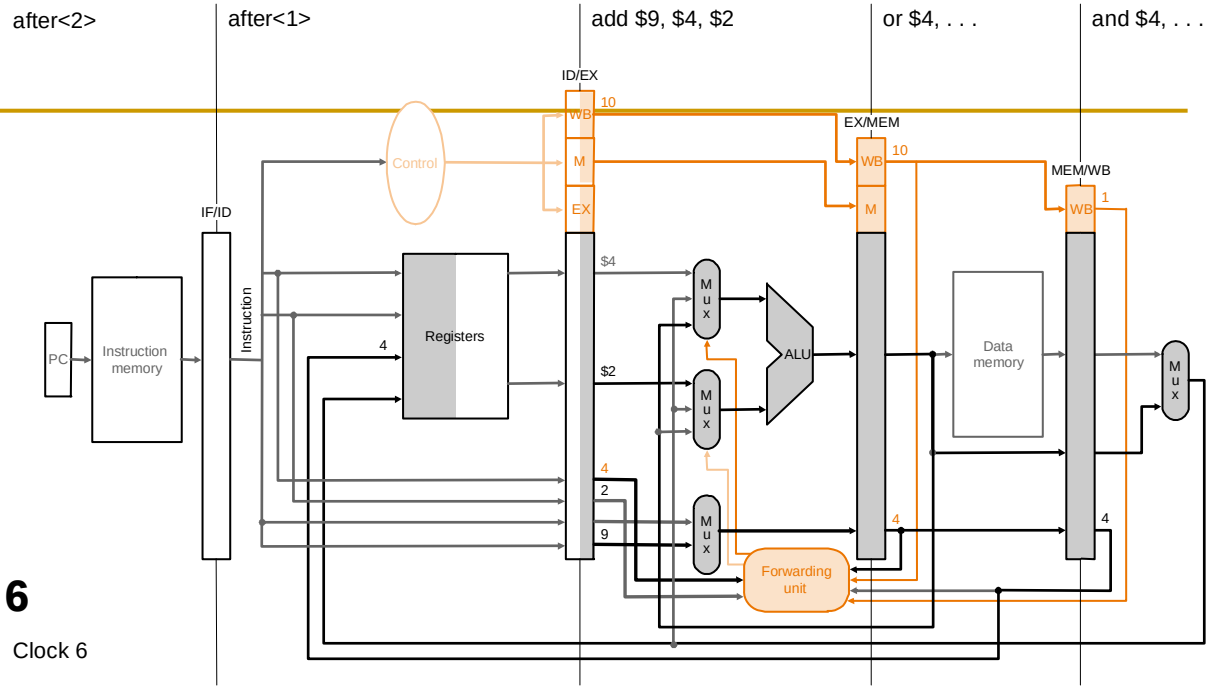
Ciclo 5

Clock 5



Ciclo 6

Clock 6

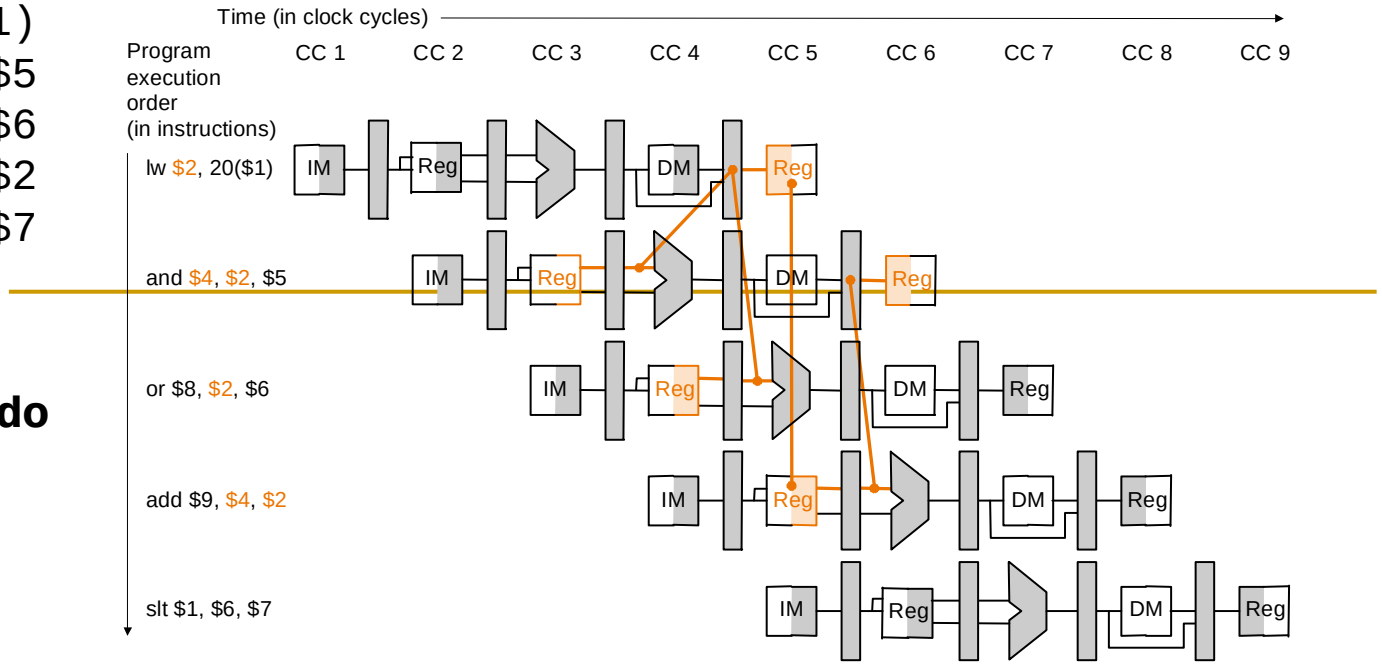


Conflito de dados e Paradas

- LW pode ainda assim causar conflito de dados:
  - Uma instrução tenta ler um registrador depois que uma instrução lw escreve no mesmo registrador

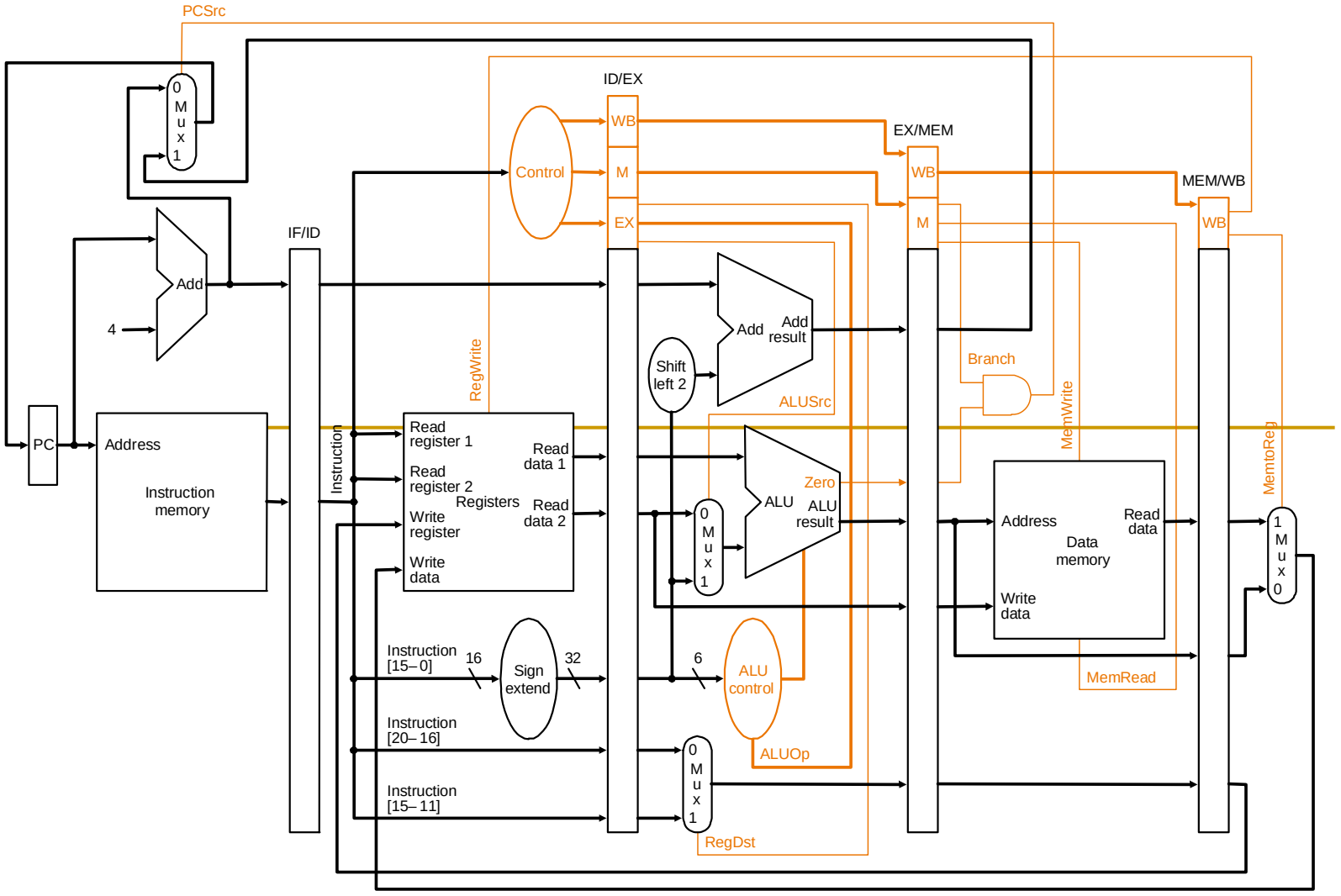
```
lw  $2, 20($1)
and $4, $2, $5
or  $8, $2, $6
add $9, $4, $2
Slt $1, $6, $7
```

Não é possível encaminhar dado para o passado



  - então, neste caso a unidade de detecção de conflito tem que parar o pipeline depois da instrução lw

Controle de pipeline original



Lógica para detectar conflito de dados que causa parada

- A unidade deve implementar o seguinte teste:

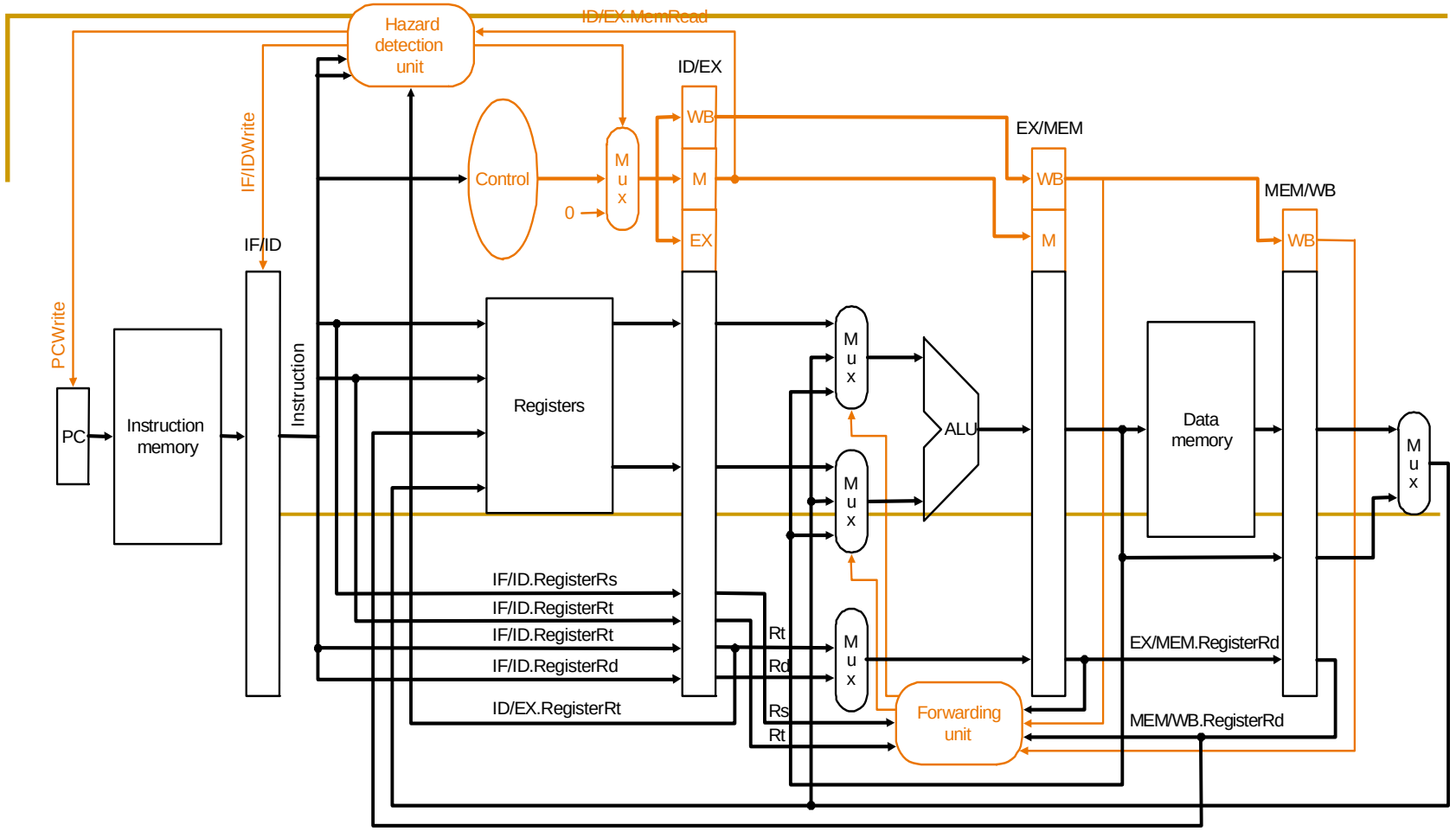
```
if ( ID/EX.MemRead                          // se a instrução do estágio EX stage é lw…
    and ( ( ID/EX.RegisterRt = IF/ID.RegisterRs )      // e o registrador destino
      or  ( ID/EX.RegisterRt = IF/ID.RegisterRt ) ) )  // corresponde ao registrador fonte
                                            // da instrução do estágio ID então …
```

para pipeline

Mecanismos de parada

- Se a condição de parada é verificada, então o *pipeline deve ser parado somente 1 ciclo de relógio* depois de lw, porque depois a unidade de encaminhamento resolve a dependência
- O que o hardware faz para parar 1 ciclo:
  - *Não deixa o registrador IF/ID trocar os bits* (*desabilita write*!) –a instrução que está no estágio ID será repetida
  - então, a instrução logo atrás no estágio IF também tem que ser parada – então hardware não deixa conteúdo de PC mudar (*desabilita write*!) – a instrução que está no estágio IF será repetida
  - *Coloca todos os campos de controle de EX, MEM e WB no registrador de pipeline em 0*, de modo que a instrução logo atrás de lw se torne nop – uma *bolha* foi inserida no pipeline
    - Note que não podemos tornar esta instrução uma nop zerando todos os bits da instrução – lembre nop = 00…0 (32 bits) – porque ela já foi decodificada e os sinais de controle já foram gerados

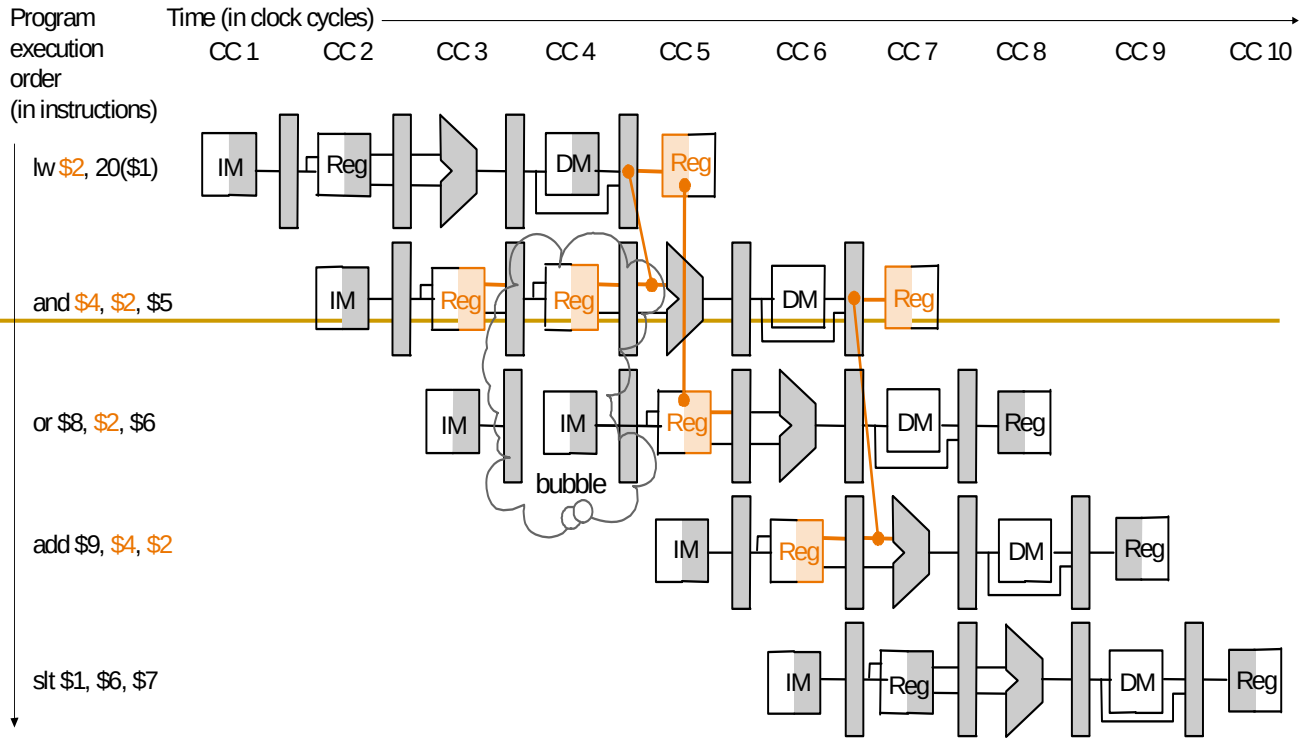
Unidade de detecção de conflito



Parada resolve conflito

- Mesma execução anterior com a inclusão da bolha:

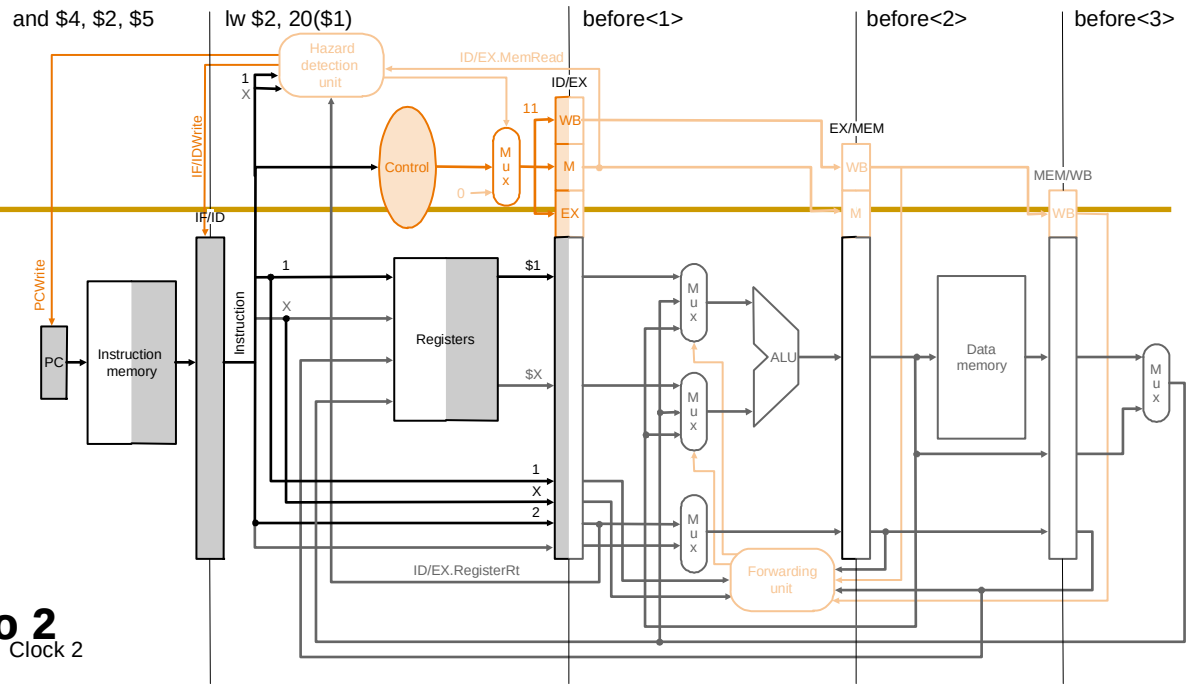
```
lw  $2, 20($1)
and $4, $2, $5
or  $8, $2, $6
add $9, $4, $2
Slt $1, $6, $7
```



Parada

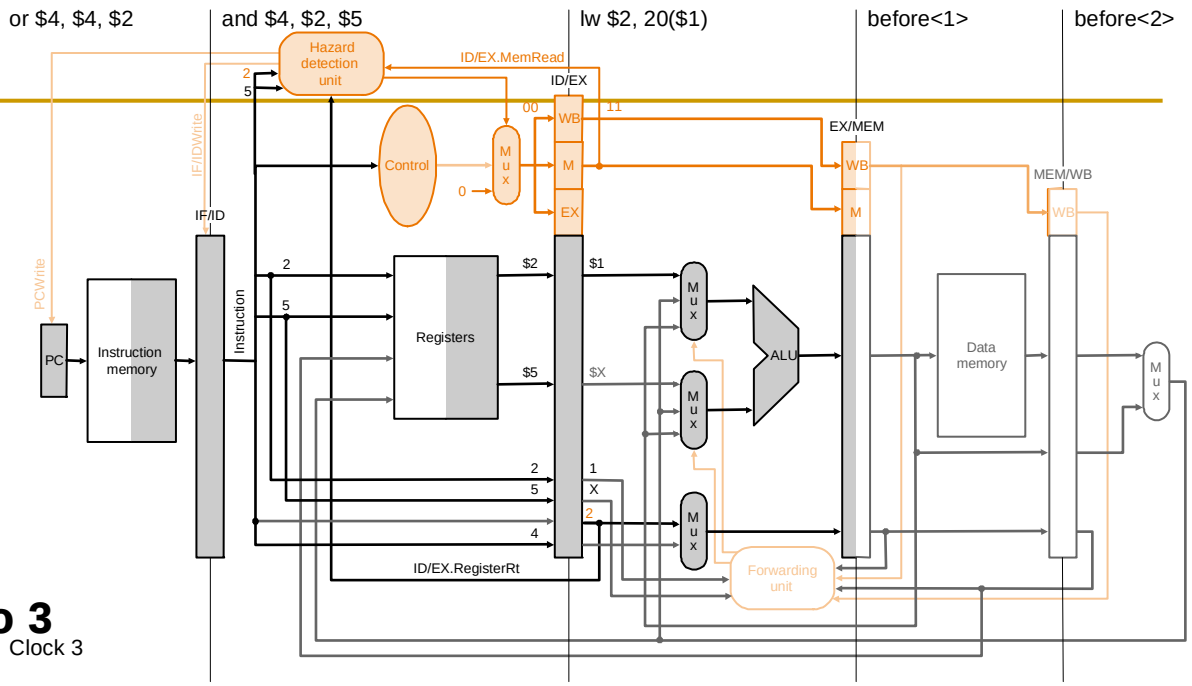
- Exemplo de execução:

```
lw  $2, 20($1)
and $4, $2, $5
or  $4, $4, $2
add $9, $4, $2
```



Ciclo 2

Clock 2

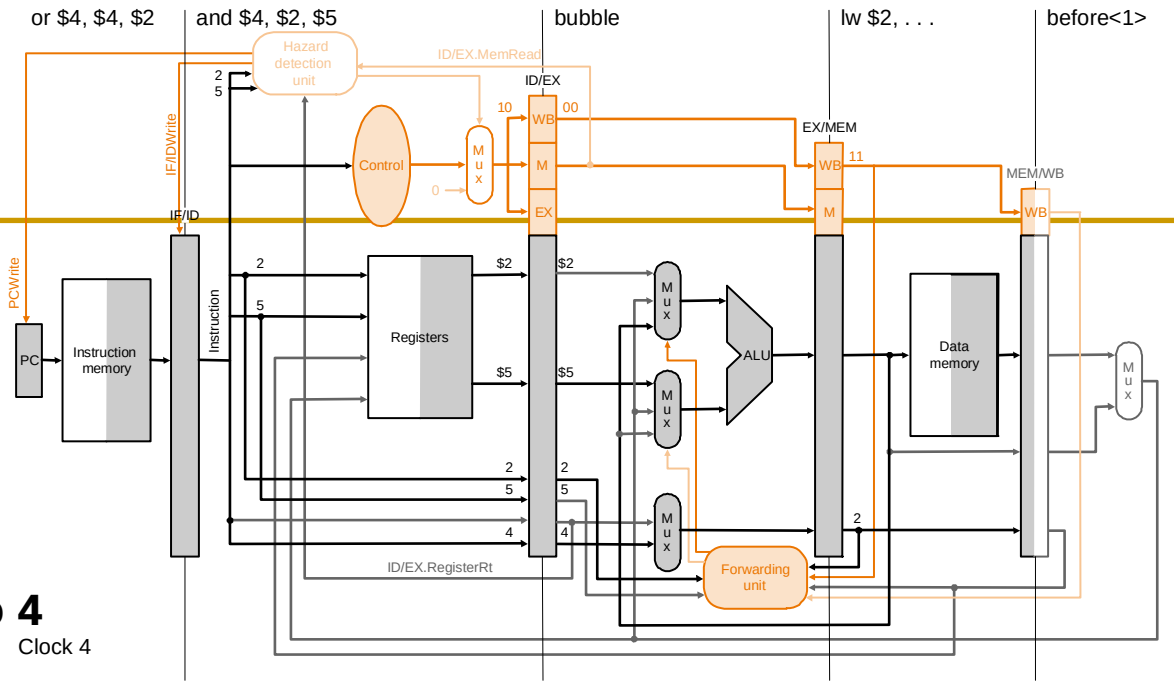


Ciclo 3

Clock 3

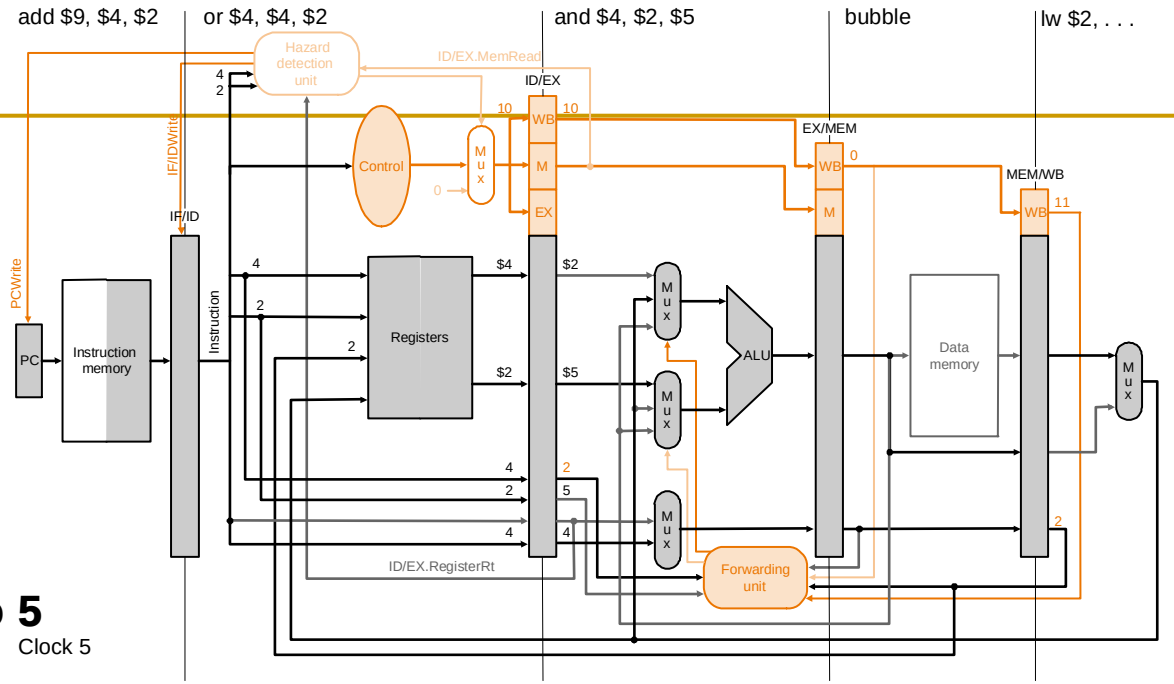
Parada

```
lw  $2, 20($1)
and $4, $2, $5
or  $4, $4, $2
add $9, $4, $2
```



Ciclo 4

Clock 4

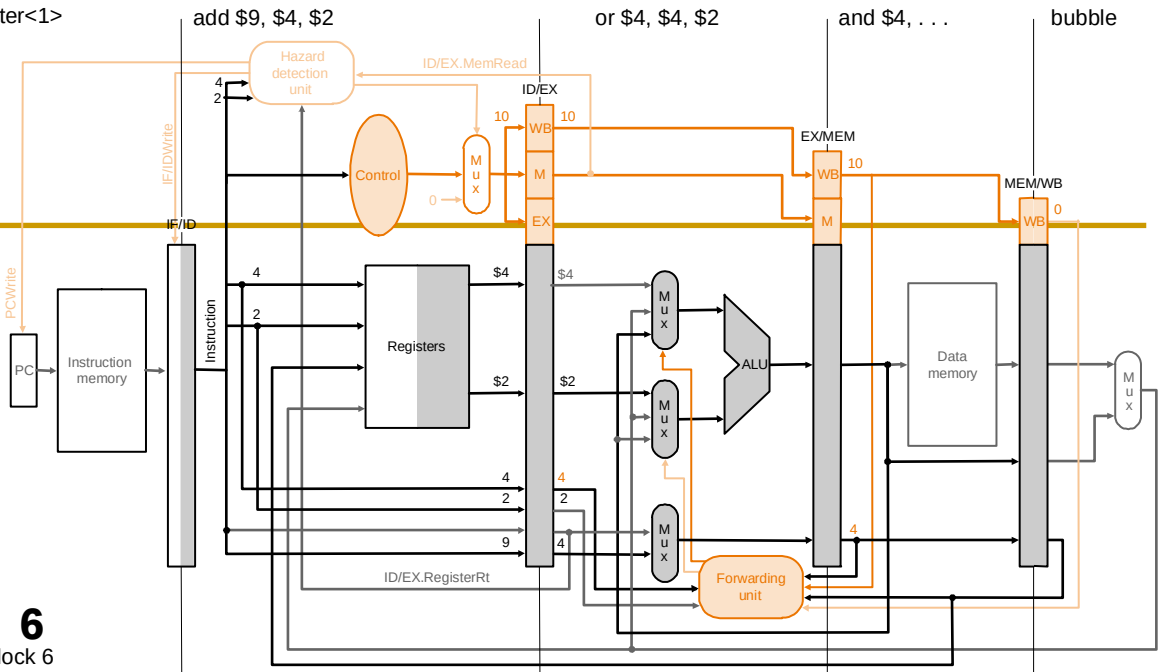


Ciclo 5

Clock 5

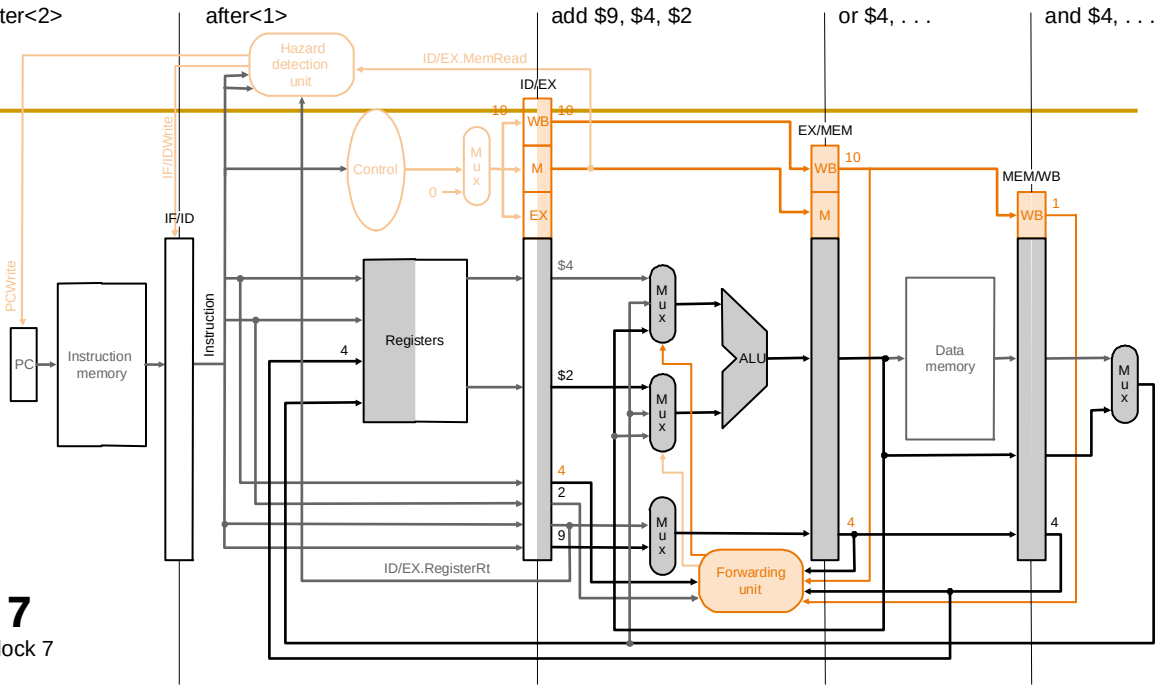
Parada

```
lw  $2, 20($1)
and $4, $2, $5
or  $4, $4, $2
add $9, $4, $2
```



Ciclo 6

Clock 6



Ciclo 7

Clock 7