SSC0510
Arquitetura de Computadores

6ª Aula – Pipeline

Profa. Sarita Mazzini Bruschi

sarita@icmc.usp.br

Pipeline

- Dependências ou Conflitos (*Hazards*)
  - Conflitos Estruturais
    - Pode haver acessos simultâneos à memória feitos por 2 ou mais estágios.
  - Dependências de Dados
    - As instruções dependem de resultados de instruções anteriores, ainda não completadas.
  - Dependências de Controle
    - A próxima instrução não está no endereço subsequente ao da instrução anterior.

Pipeline
Dependência de Dados

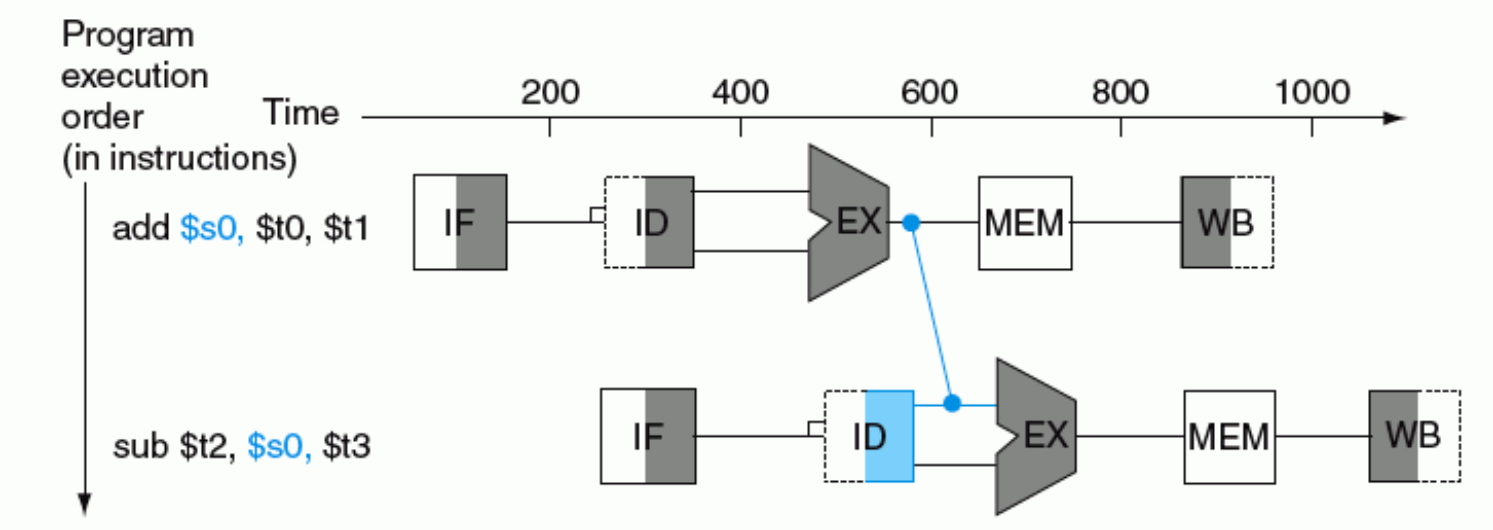
- Problema: uma instrução faz uso de um operando que vai ser produzido por uma outra instrução que ainda está no pipeline.
- A execução da instrução seguinte depende de operando calculado pela instrução anterior.
- Tipos de dependências de dados:
  - Dependência verdadeiras
  - Dependências falsas
    - Antidependência
    - Dependência de saída

Pipeline
Dependência de Dados

- Dependências Verdadeiras (Direta):
  - O pipeline precisa ser parado durante certo número de ciclos (*interlock*);
  - Inserção de instruções de “nop” ou escalonamento adequado das instruções pelo compilador;
  - O adiantamento (*bypassing* ou *forwarding*) dos dados pode resolver em alguns casos.
- Dependências Falsas:
  - Não é um problema em pipelines onde a ordem de execução das instruções é mantida;
  - Problema em processadores superescalares;
  - A renomeação dos registradores é uma solução usual para este problema.

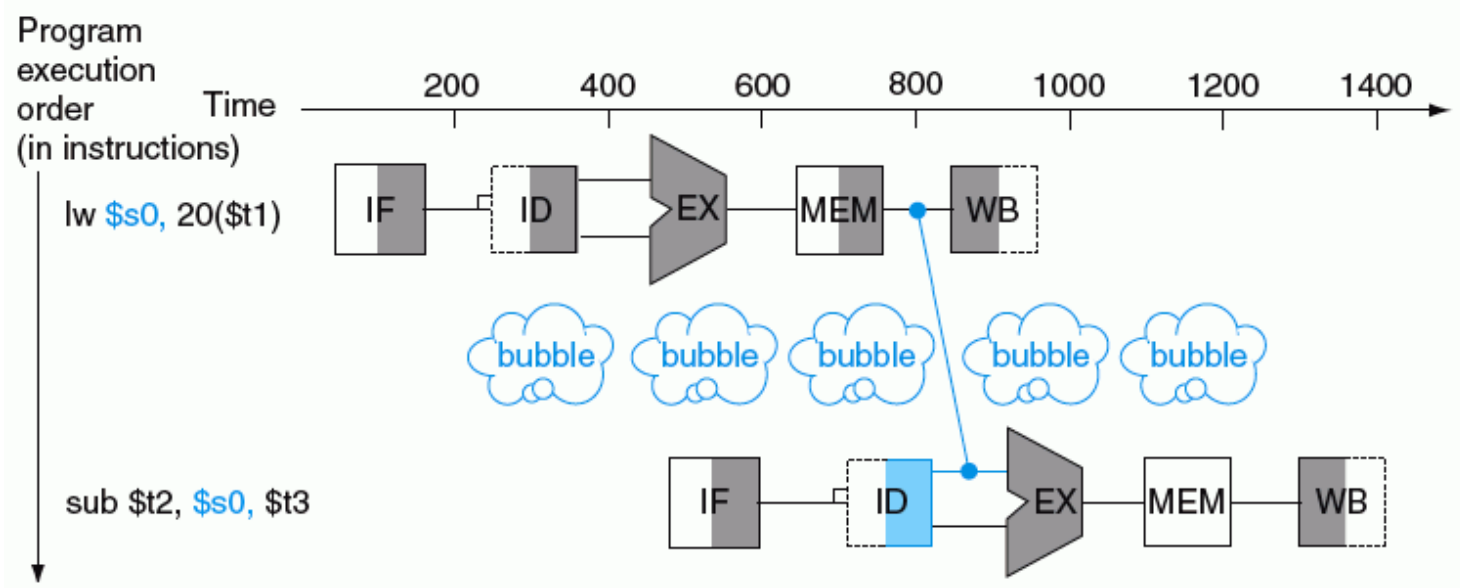
Pipeline
Dependência de Dados

- Adiantamento de Dados
  - Caminho interno dentro do pipeline entre a saída e a entrada da ULA
    - Técnica conhecida como *forwarding* ou *bypassing*
    - Evita a *parada* do pipeline utilizando *buffers* internos em vez de esperar que o elemento de dado chegue nos registradores visíveis ao programador ou na memória



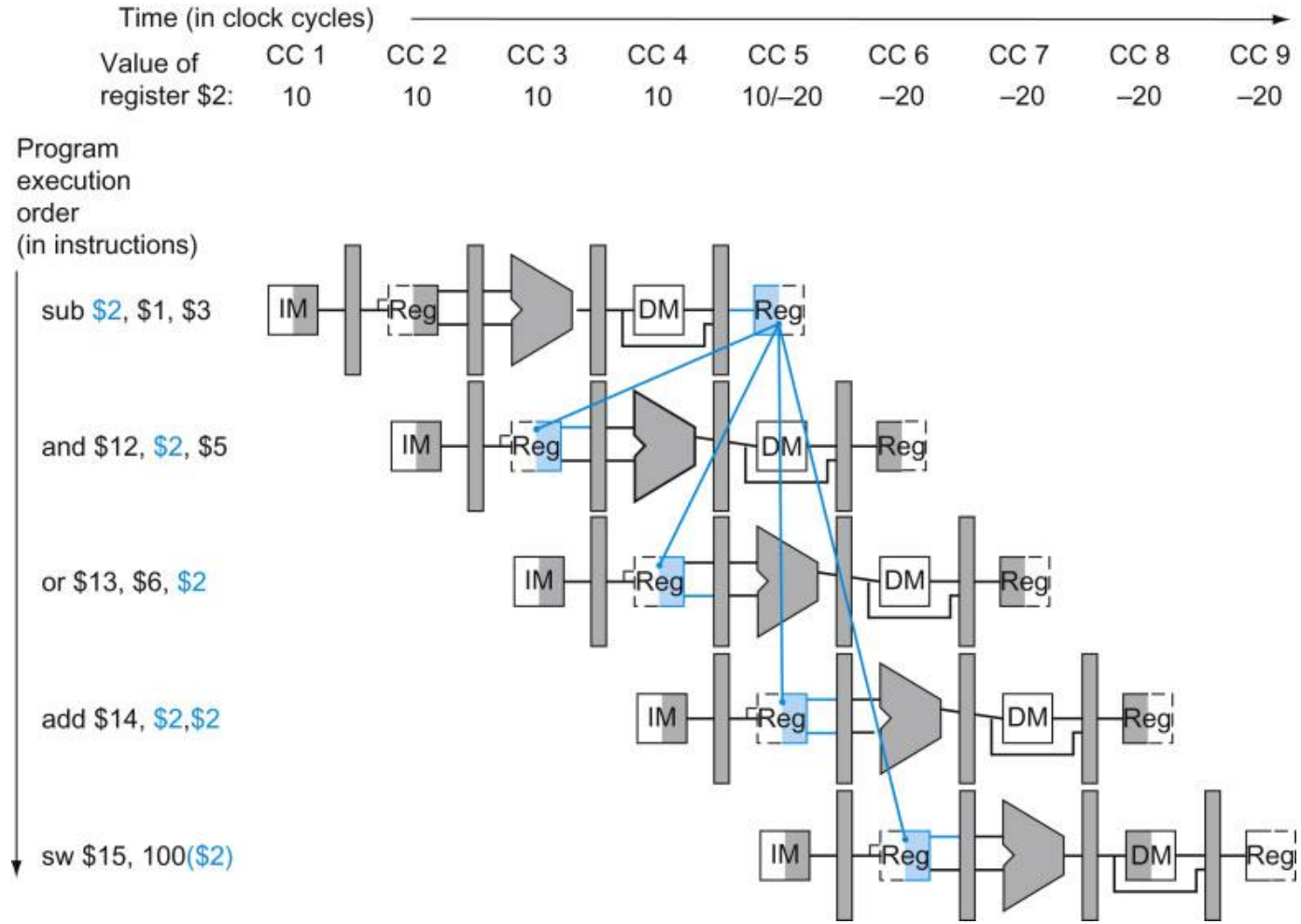
Pipeline
Dependência de Dados

- Adiantamento de Dados (cont.)
  - Em algumas situações, nem o *forwarding* pode resolver o problema de parada do pipeline



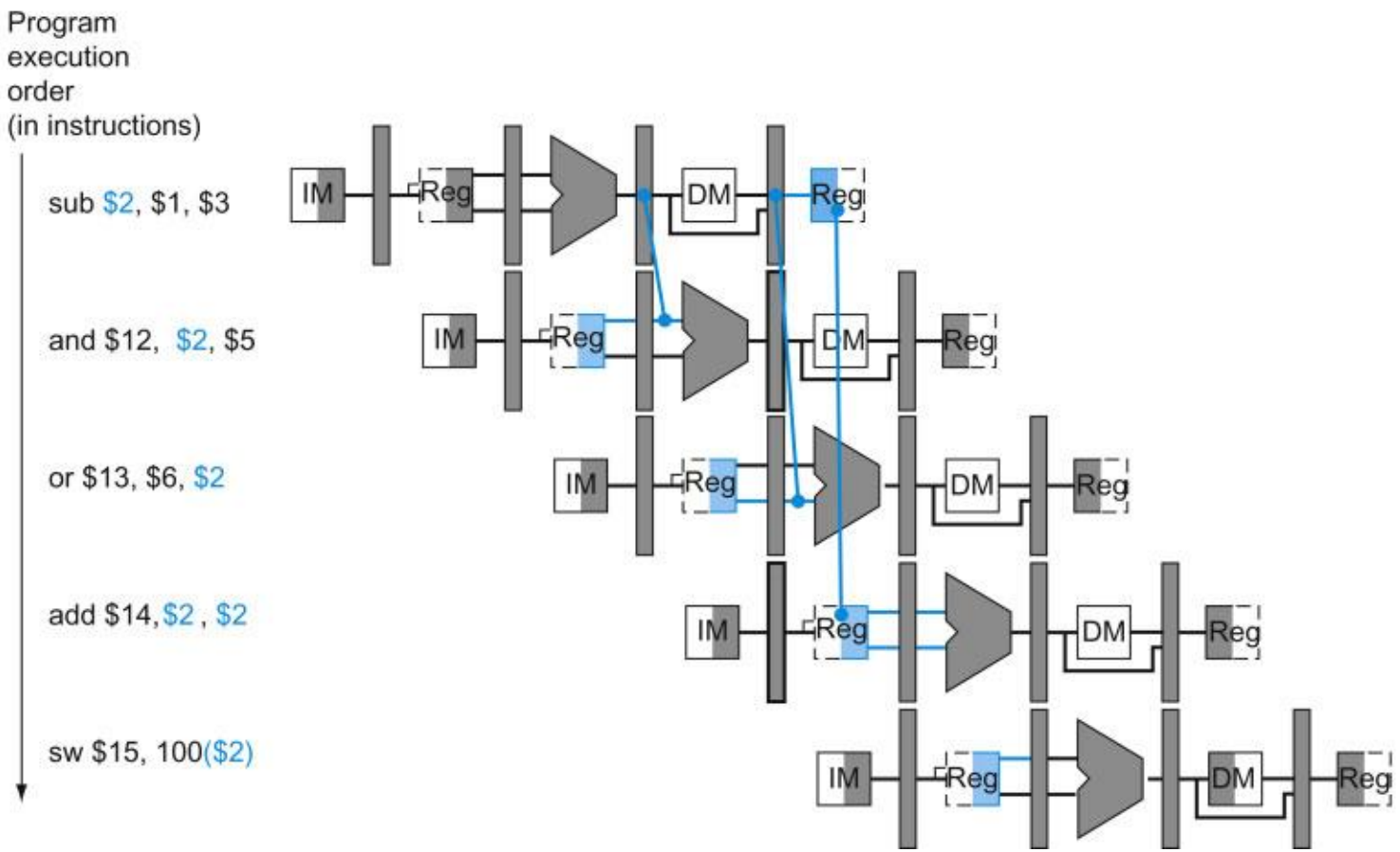
Pipeline
Dependências de Dados

```
sub    $2, $1,$3       # Register $2 written by sub
and    $12,$2,$5       # 1st operand($2) depends on sub
or     $13,$6,$2       # 2nd operand($2) depends on sub
add    $14,$2,$2       # 1st($2) & 2nd($2) depend on sub
sw     $15,100($2)     # Base ($2) depends on sub
```



Time (in clock cycles) →

	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



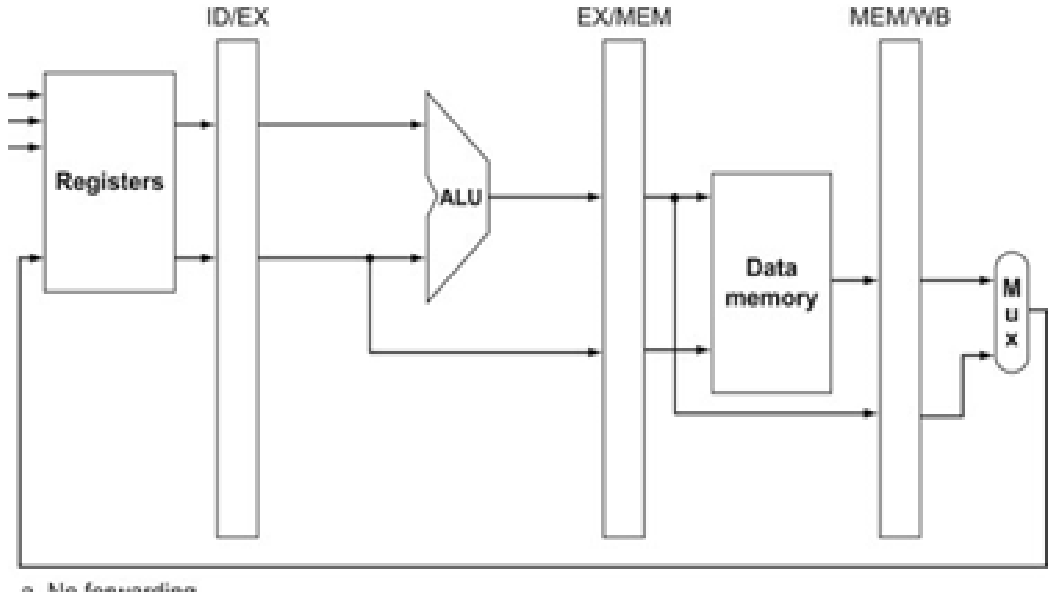
Pipeline
Dependência de Dados

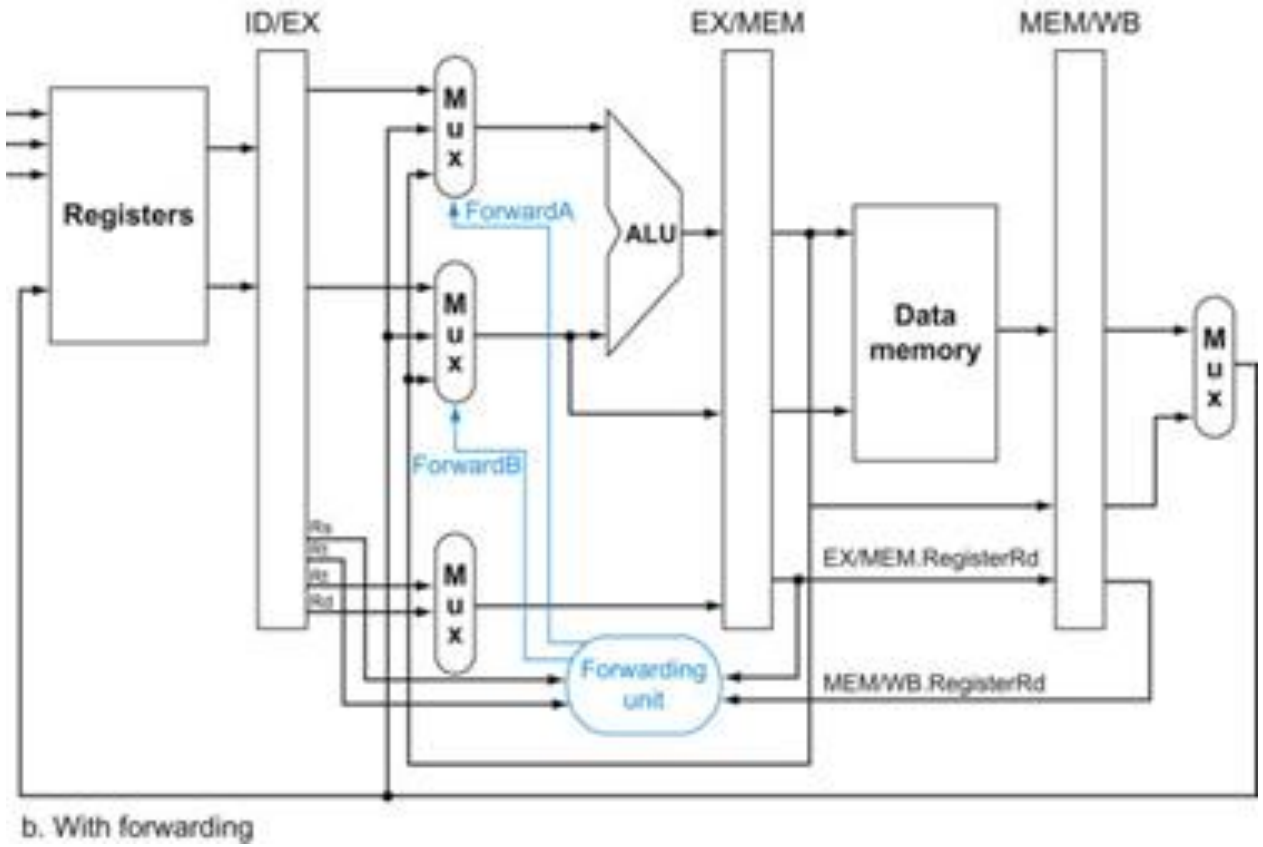
- Como implementar o adiantamento (*forwarding*)?
- Condições para que o adiantamento seja considerado:
  - Destino da instrução que está entre os ciclos de EX e MEM é igual a um dos registradores de origem da instrução que está entre os ciclos de ID e EX
    - 1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
    - 1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
  - Destino da instrução que está entre os ciclos de MEM e WB é igual a um dos registradores de origem da instrução que está entre os ciclos de ID e EX
    - 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
    - 2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

Pipeline
Dependência de Dados

- No exemplo do código anterior, tem-se as seguintes dependências:
  - Entre as instruções: `sub $2,$1,$3` e `and $12,$2,$5`
    - Da numeração do slide anterior, qual é essa dependência?
  - Entre as instruções: `sub $2,$1,$3` e `or $16,$6,$2`
    - Da numeração do slide anterior, qual é essa dependência?
  - As duas dependências `sub-add` não são problema pois o banco de registradores irá fornecer os dados corretos no ciclo de decodificação
  - Não existe *hazard* entre as instruções `sub` e `sw`

Pipeline
Dependência de Dados





b. With forwarding

Pipeline
Dependência de Dados

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

Pipeline
Dependência de Dados

- Definição das condições para a detecção de dependência de dados:
  - Dependência EX:

    if (EX/MEM.RegWrite
    and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

    if (EX/MEM.RegWrite
    and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10

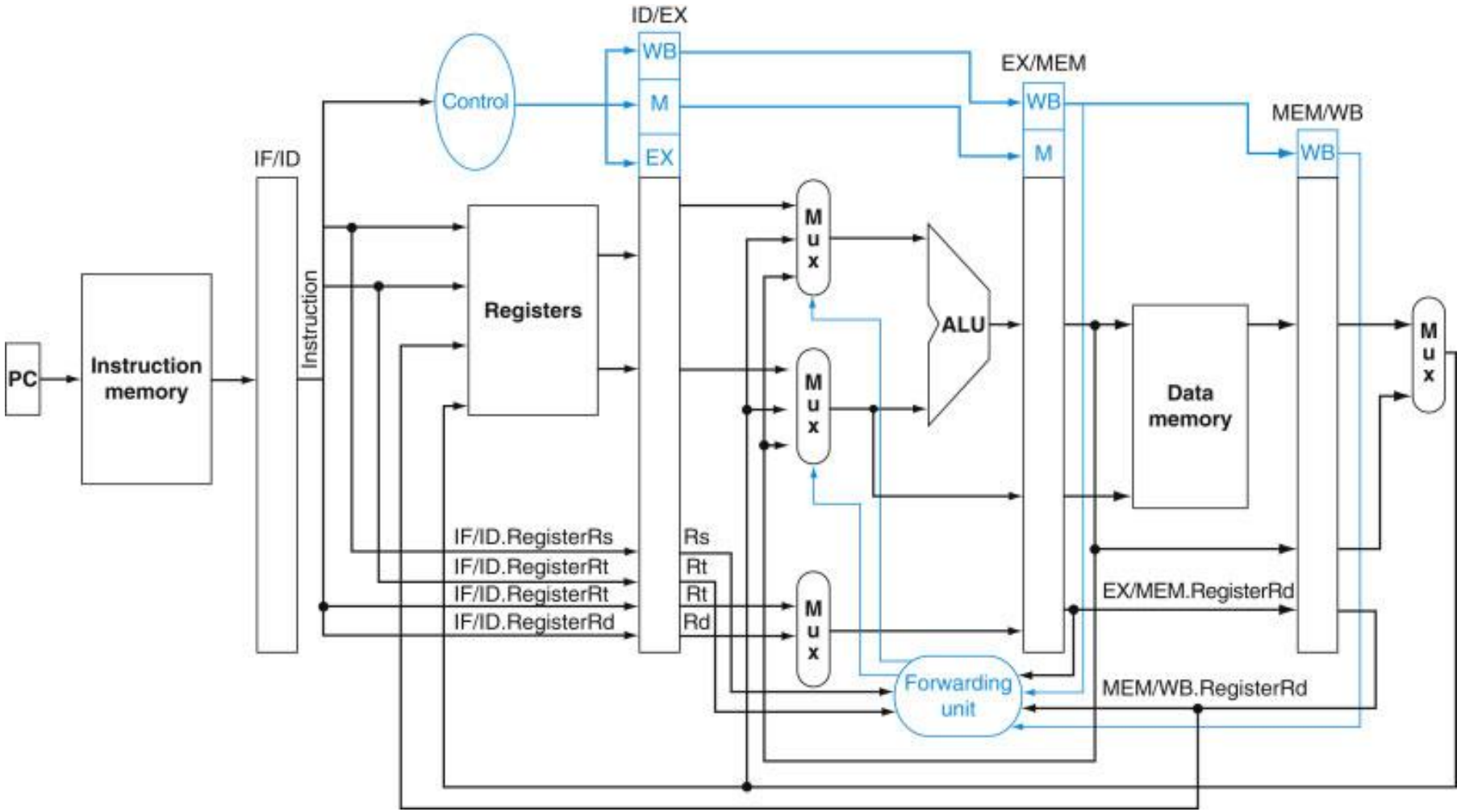
Pipeline
Dependência de Dados

- Definição das condições para a detecção de dependência de dados:
  - Dependência MEM:

    if (MEM/WB.RegWrite
    and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

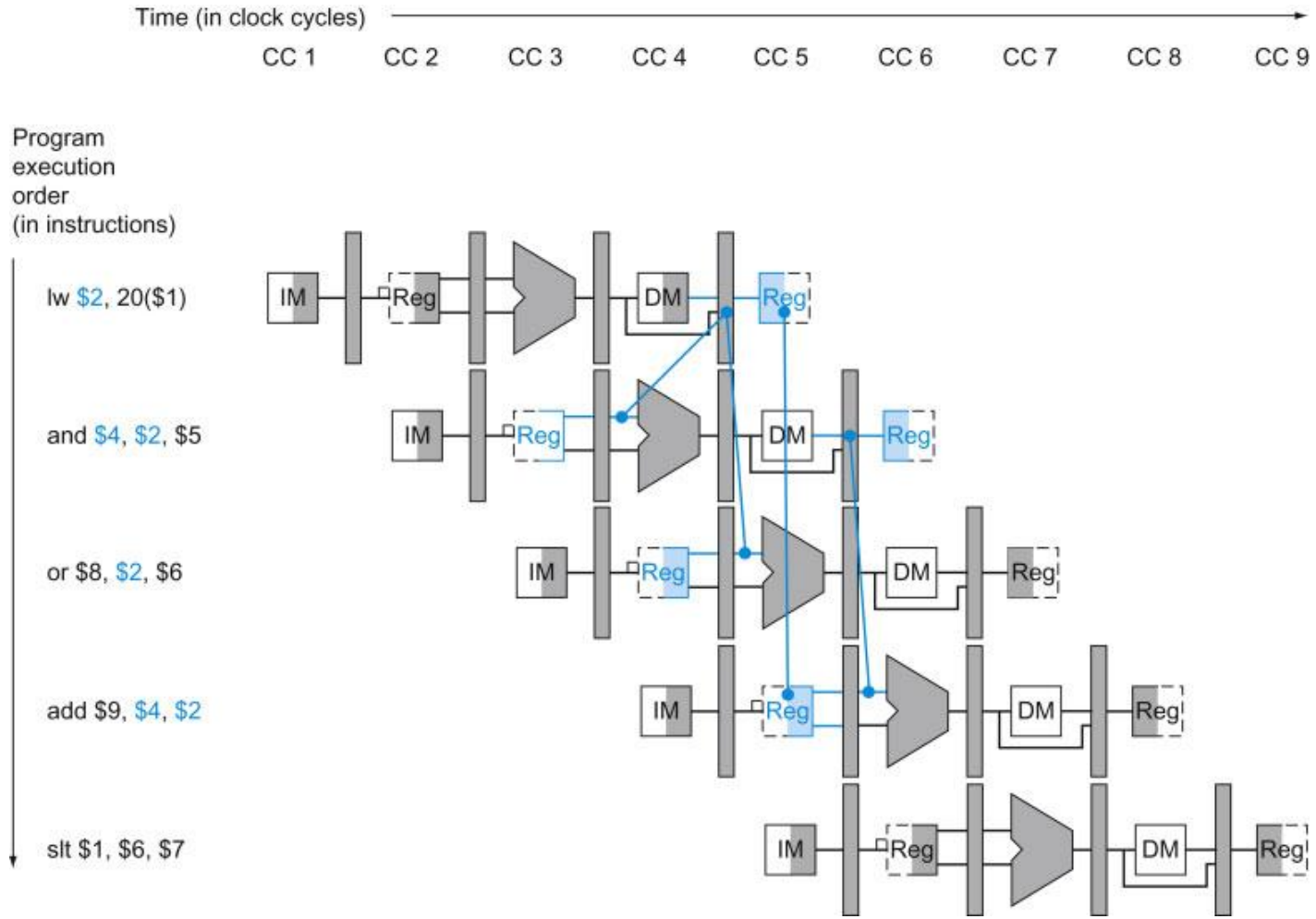
    if (MEM/WB.RegWrite
    and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01

Pipeline
Dependência de Dados



Pipeline
Dependência de Dados

- Como detectar as paradas?
  - Nem sempre é possível resolver as dependências com *forwarding*

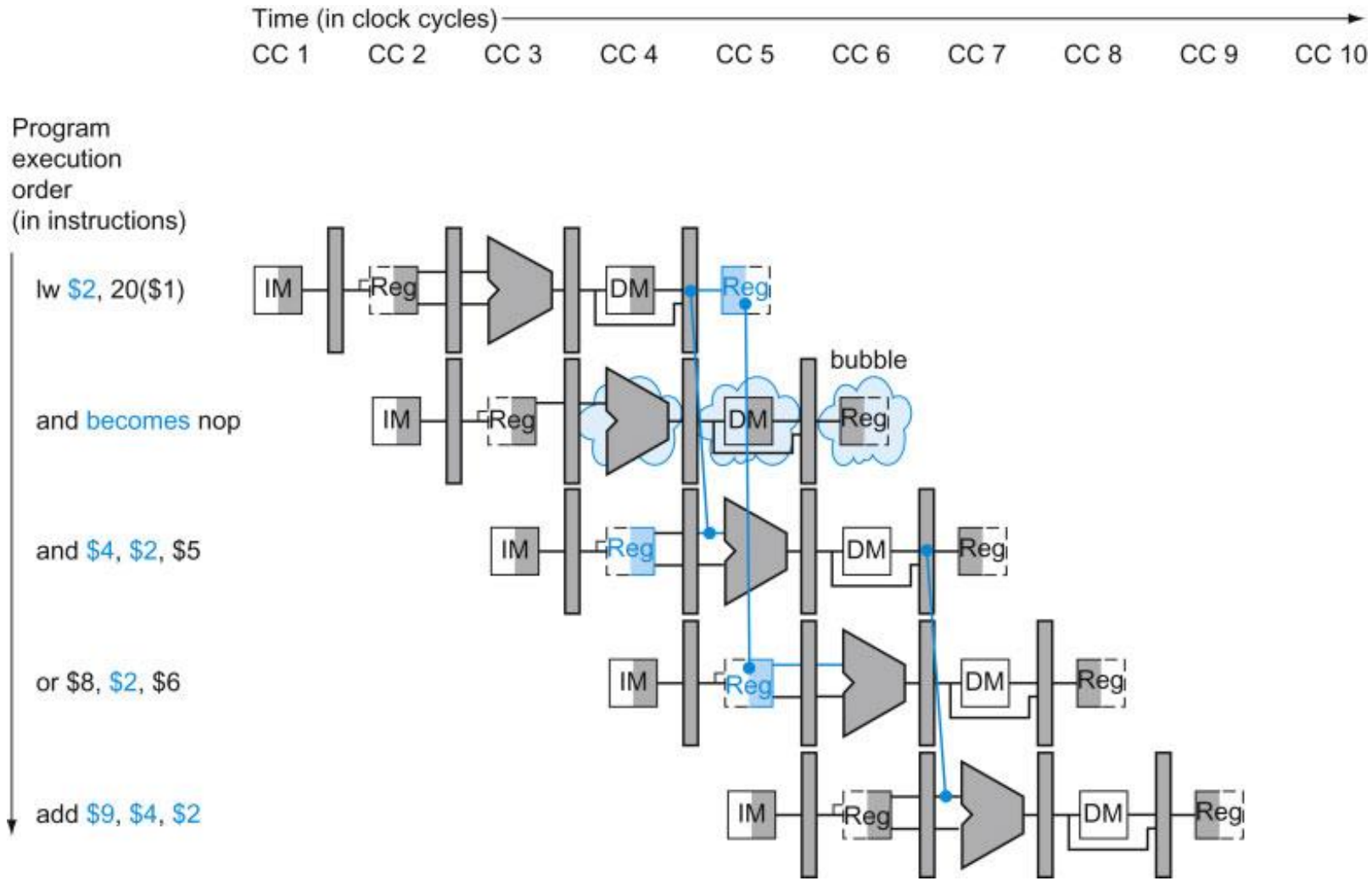


Pipeline
Dependência de Dados

- Definição de uma unidade de detecção de dependência (*Hazard Detection Unit*)
  - Caso a dependência seja detectada, deve ser inserida uma operação **nop** (no operation)

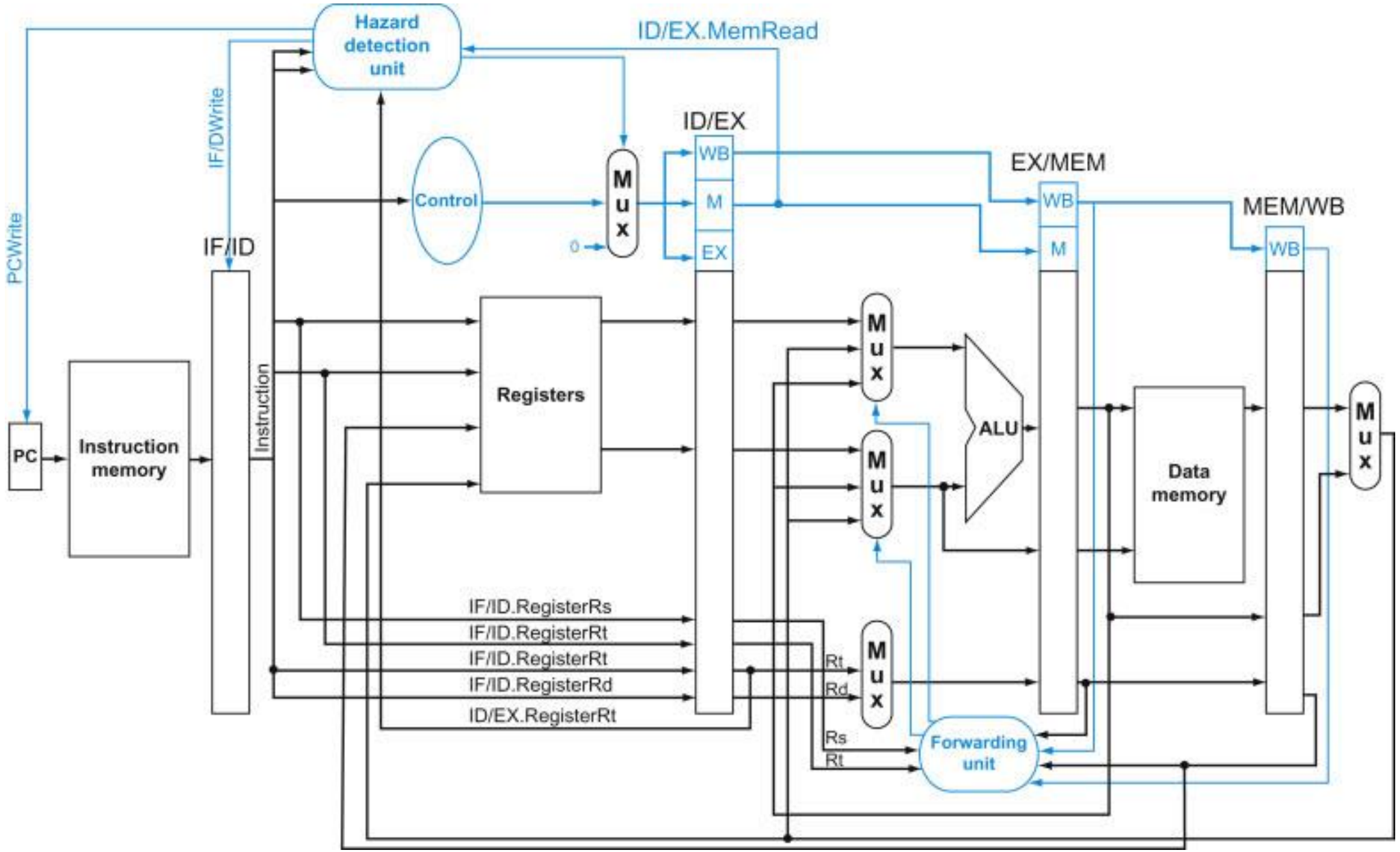
```
if (ID/EX.MemRead and                                   // leitura da memória
        ((ID/EX.RegisterRt = IF/ID.RegisterRs) or       // Destino (EX) == Origem 1 (IF)
                (ID/EX.RegisterRt = IF/ID.RegisterRt)))  // Destino (EX) == Origem 2 (IF)
        stall the pipeline
```

Pipeline
Dependência de Dados



Pipeline
Dependência de Dados

- Como tratar a parada, caso ela seja detectada, no estágio ID?
  - Instruções nos estágios ID e IF devem parar
    - Basta impedir que o registrador PC e os registradores de pipeline IF/ID sejam alterados, preservando os seus conteúdos no próximo ciclo
  - Como a instrução está num pipeline, ela caminha para o estágio EX, que precisa executar algo, mas não pode ser a instrução pois não tem os dados necessários
    - Executa uma instrução que não tem efeito: **nop**
    - Para inserir essa bolha no pipeline basta desativar todos os sinais de controle dos estágio EX, MEM e WB


Hazard detection unit
ID/EX.MemRead
IF/DWrite
PCWrite
ID/EX
WB
M
EX
EX/MEM
WB
M
MEM/WB
WB
Control
M u x
0
IF/ID
Instruction
PC
Instruction memory
Registers
M u x
ALU
Data memory
M u x
IF/ID.RegisterRs
IF/ID.RegisterRt
IF/ID.RegisterRt
IF/ID.RegisterRd
ID/EX.RegisterRt
Rt
Rd
Rs
Rt
Forwarding unit

Exercício

Considere o seguinte trecho de código

```
I0. lw $s2, 0($s1)
I1. lw $s1, 40($s6)
I2. sub $s6, $s1, $s2
I3. add $s6, $s2, $s2
I4. or $s3, $s6, $zero
I5. sw $s6, 50($s1)
```

Exercício

a) Quais são as dependências de dados? E os *hazards*? (Dependência é quando o dado de uma instrução depende de outra instrução. *Hazard* é uma situação na qual a execução do pipeline é impedida de continuar, causando uma parada.)

b) Assuma um pipeline MIPS de 5 estágios sem *forwarding*, e que cada estágio demora 1 ciclo. Ao invés de inserir operações *nops*, você deixar o processador parar quando se tem *hazards*. Quantos ciclos o processador para? Qual o tamanho de cada parada, em ciclos? Qual o tempo de execução (em ciclos) do programa inteiro?

c) Assuma um pipeline MIPS de 5 estágio com *forwarding* total. Escreva o programa com *nops* para eliminar os *hazards*.