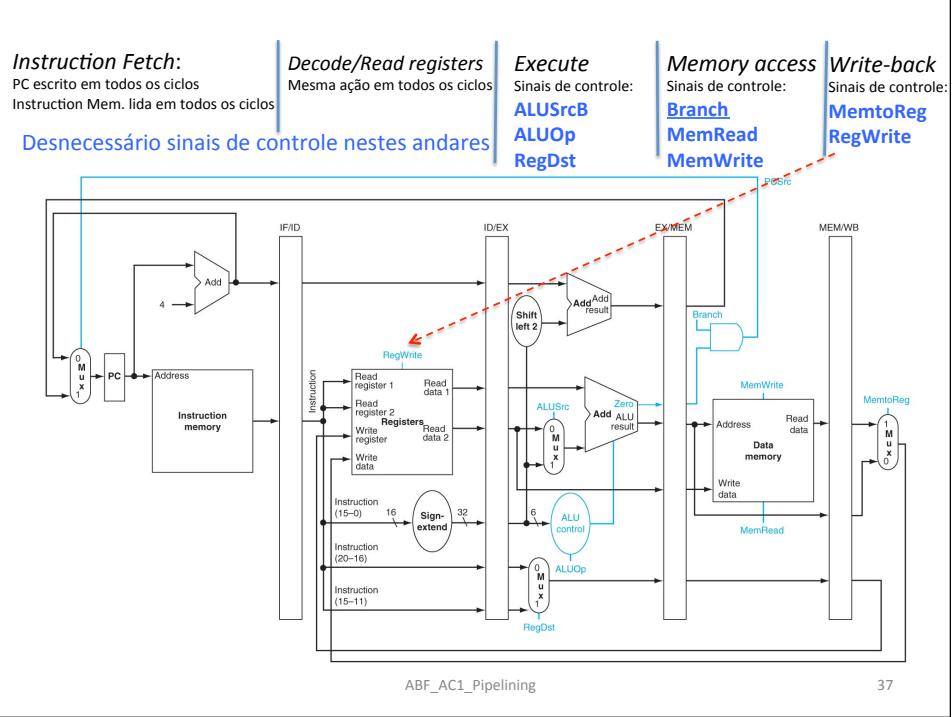


2. Pipelined Control

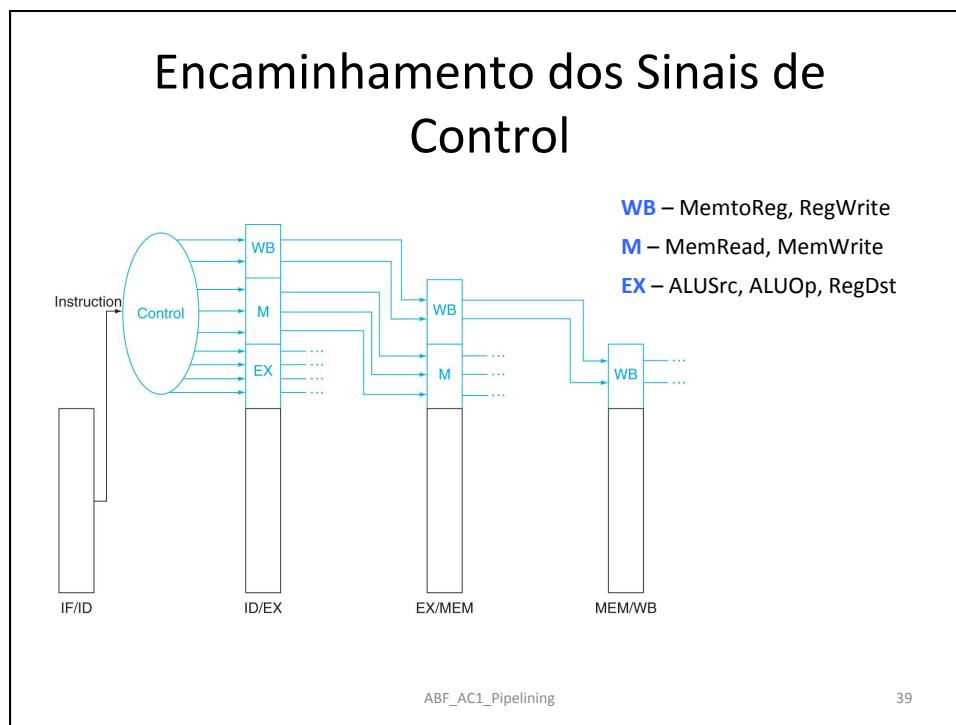
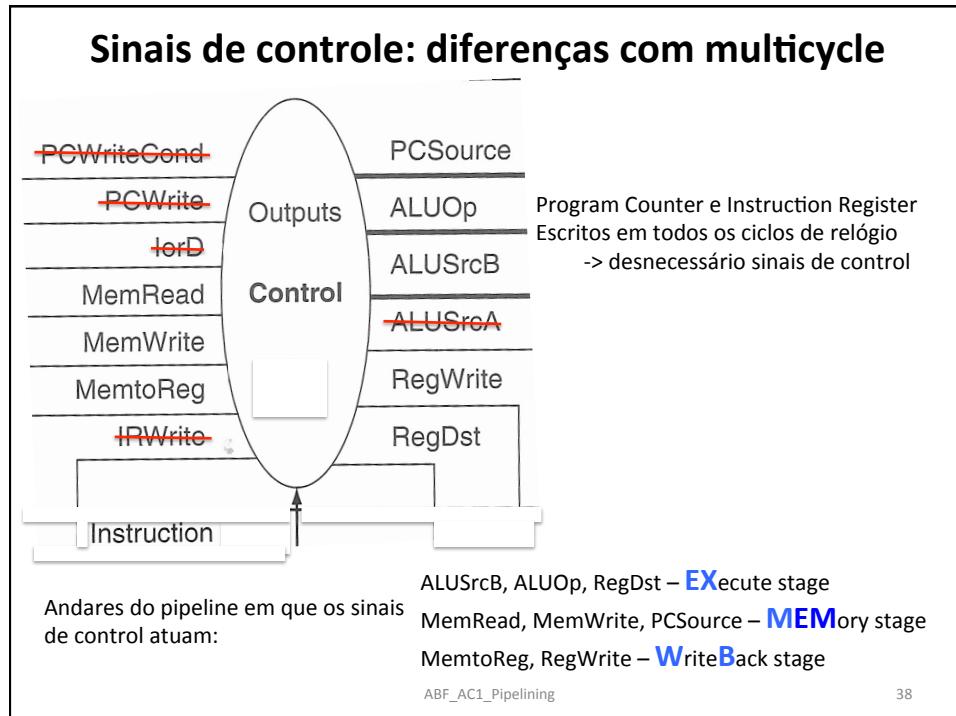
ABF_AC1_Pipelining

36

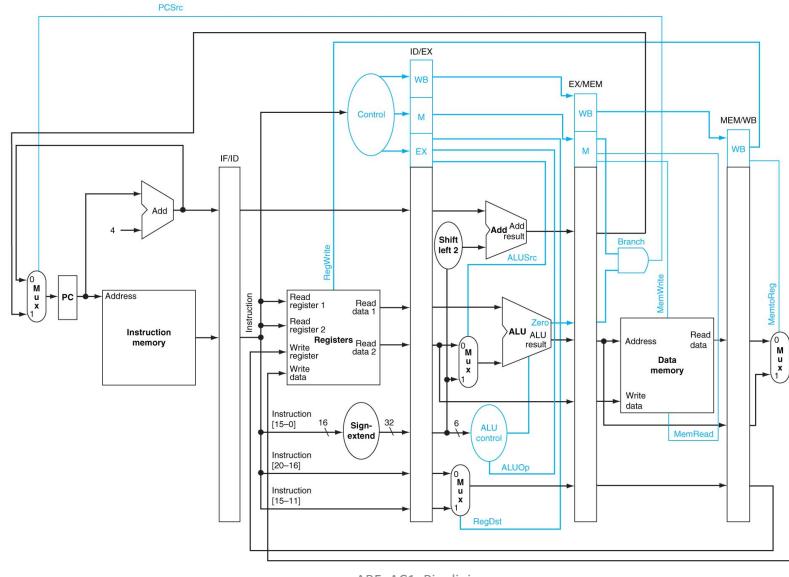


ABF_AC1_Pipelining

37

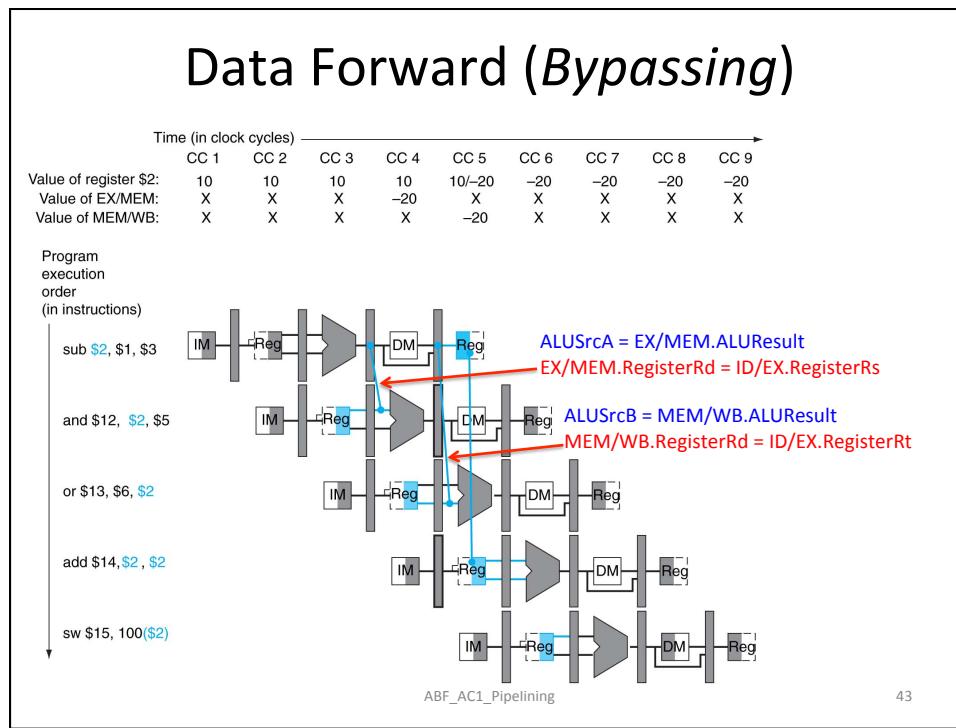
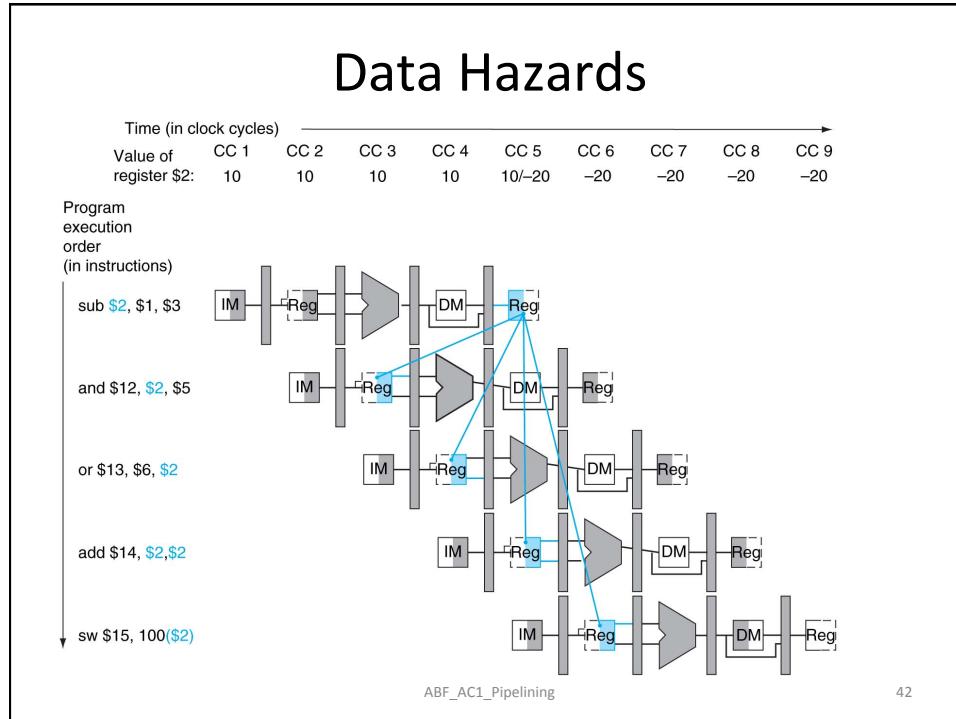


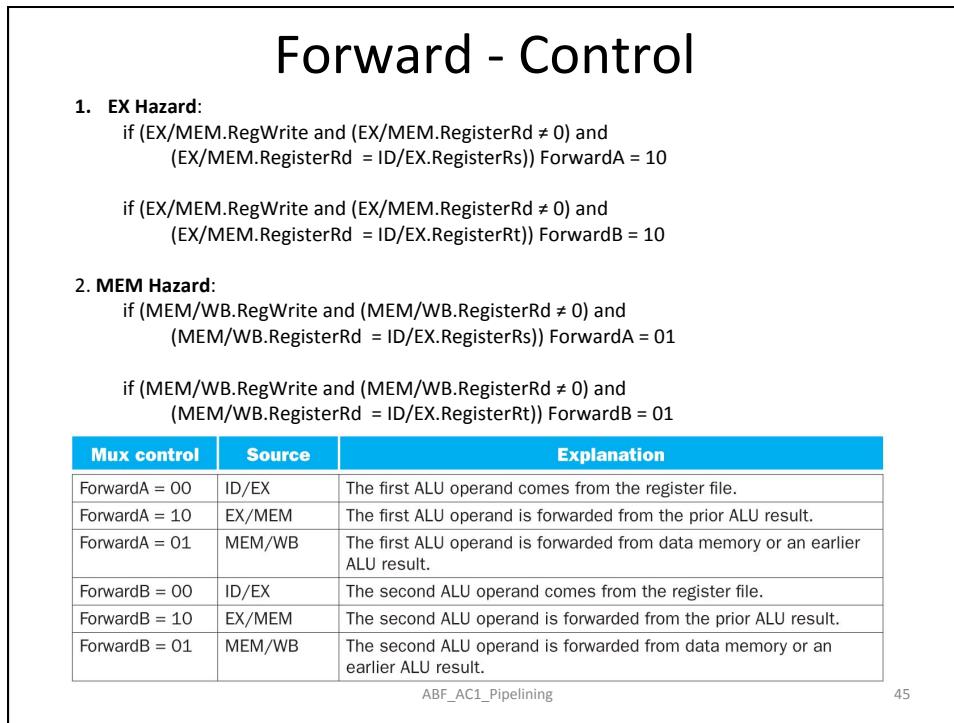
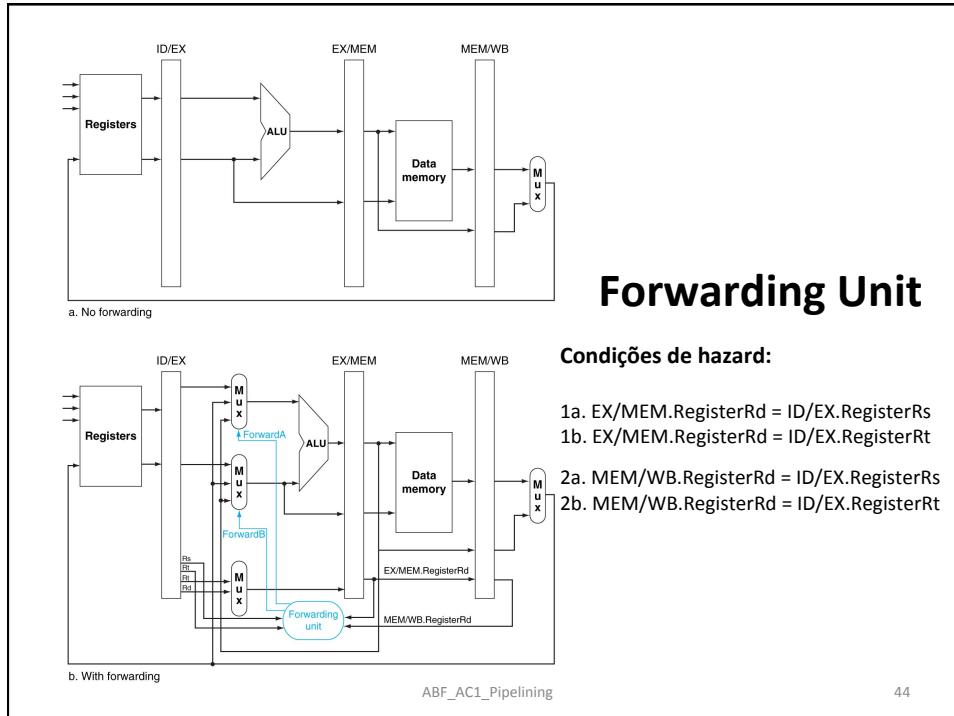
Pipelined Datapath + Control



40

3. Pipeline Hazards





Forward Control (2)

```

add $1, $1, $2          Resultado do 2º add
add $1, $1, $3          Resultado do 1º add
add $1, $1, $4 # (EX/MEM.RegisterRd = ID/EX.RegisterRs) and (MEM/WB.RegisterRd = ID/EX.RegisterRs)

```

Na execução da 3ª instrução tem de se fazer o forward do resultado do 2º add
(forward de EX/MEM tem prioridade sobre forward de EM/WB):

MEM Hazard corrigido:

```

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and
    not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and
        (EX/MEM.RegisterRd = ID/EX.RegisterRs)) and
    (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

```

```

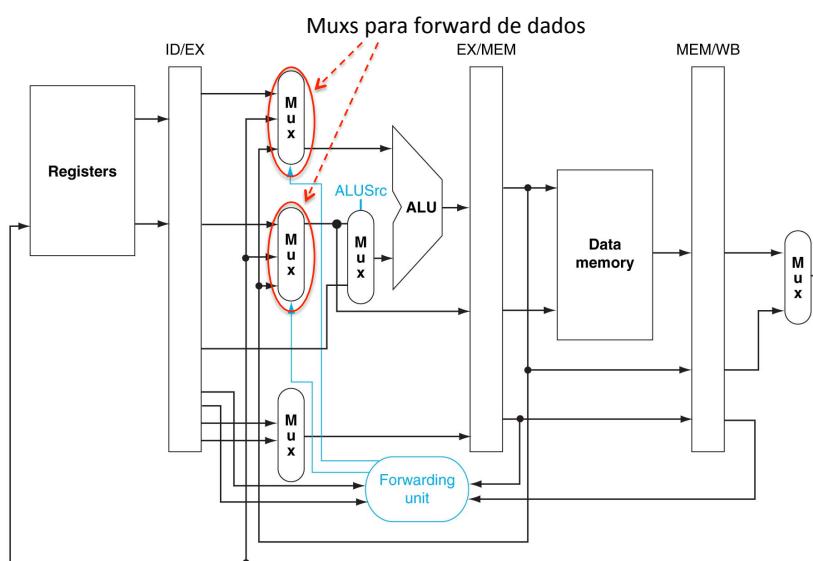
if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and
    not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and
        (EX/MEM.RegisterRd = ID/EX.RegisterRt)) and
    (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01

```

ABF_AC1_Pipelining

46

Datapath completado



ABF_AC1_Pipelining

47

Execução no pipeline com forwarding

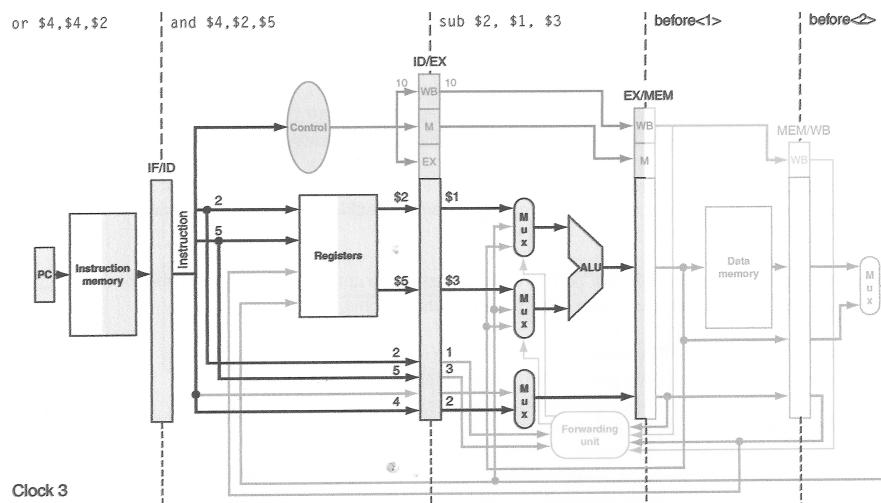
Exemplo:

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

ABF_AC1_Pipelining

48

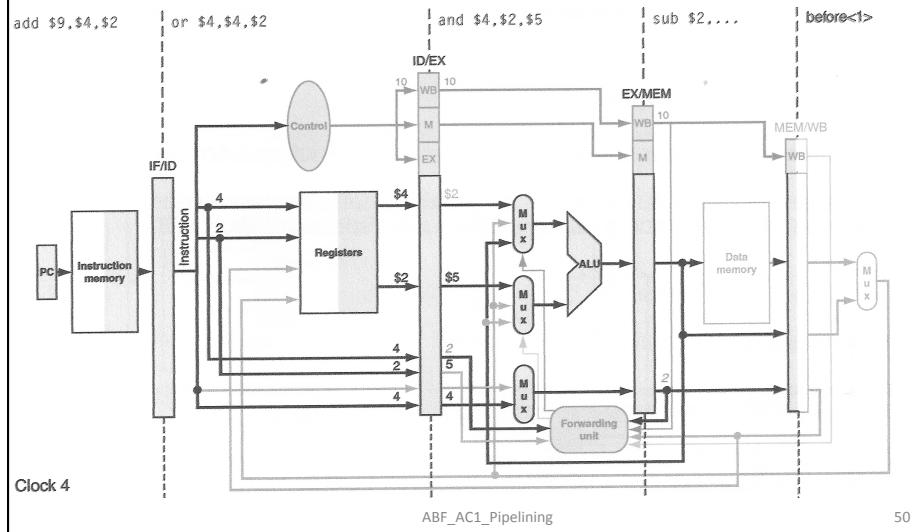
Execução no pipeline com forwarding 3º ciclo de relógio



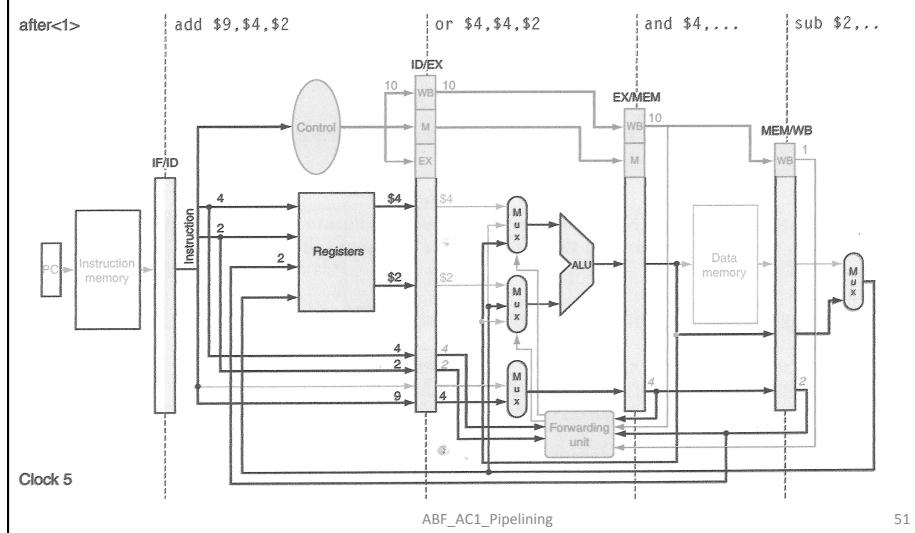
ABF_AC1_Pipelining

49

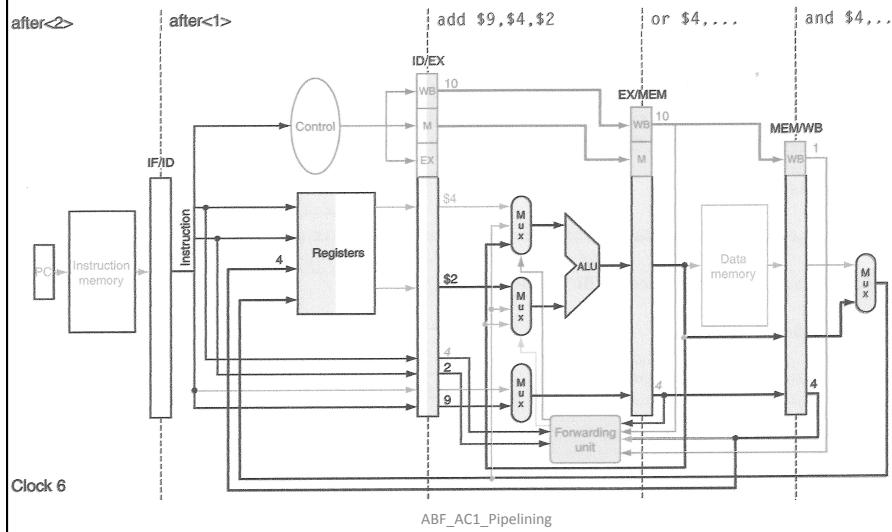
Execução no pipeline com forwarding 4º ciclo de relógio



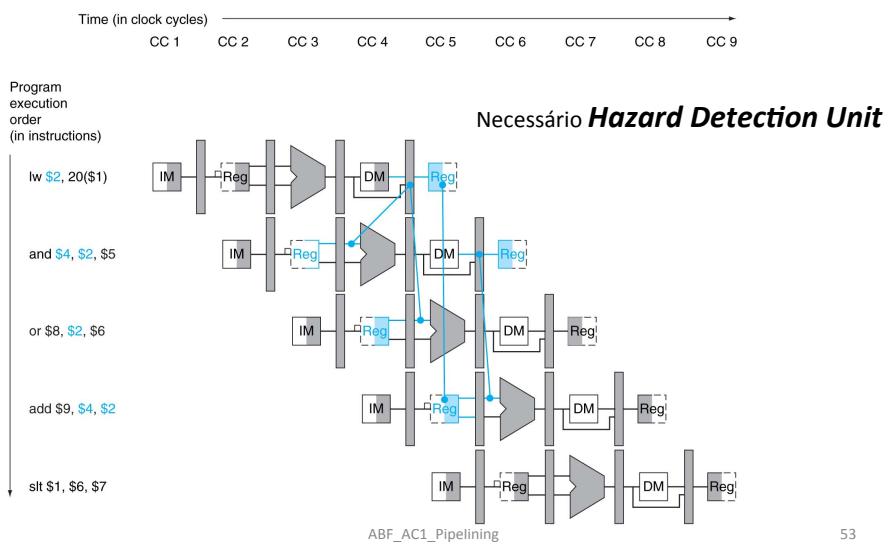
Execução no pipeline com forwarding 5º ciclo de relógio



Execução no pipeline com forwarding 6º ciclo de relógio



Data Hazards não resolueis sem stall (causados por instruções de *load*)



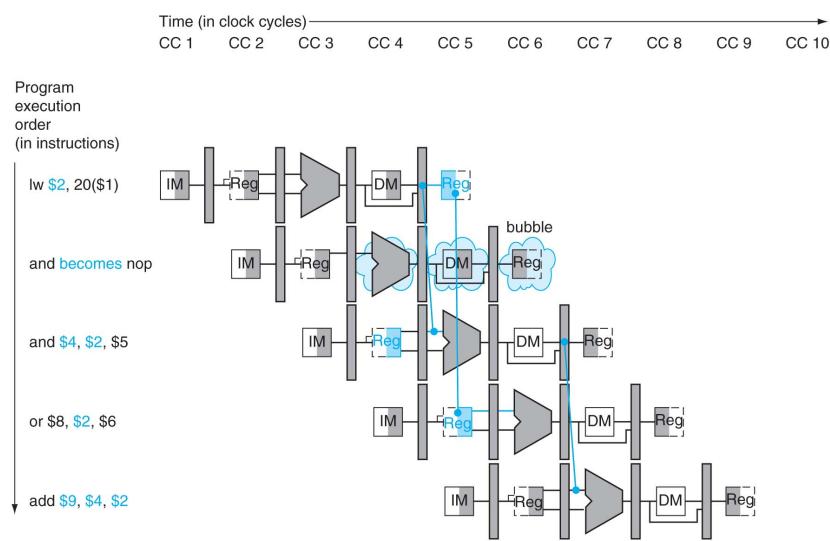
Hazard Detection Unit

- ```
if (ID/EX.MemRead and
 ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
 (ID/EX.RegisterRt = IF/ID.RegisterRt)))
stall the pipeline
• Stall the pipeline:
 ➤ Manter o valor do PC (leitura repetida da mesma
 instrução)
 ➤ Manter o conteúdo do registo IF/ID
• Introduzir nop:
 – Colocar a zero os campos EX, MEM e WB do registo
 ID/EX do pipeline
```

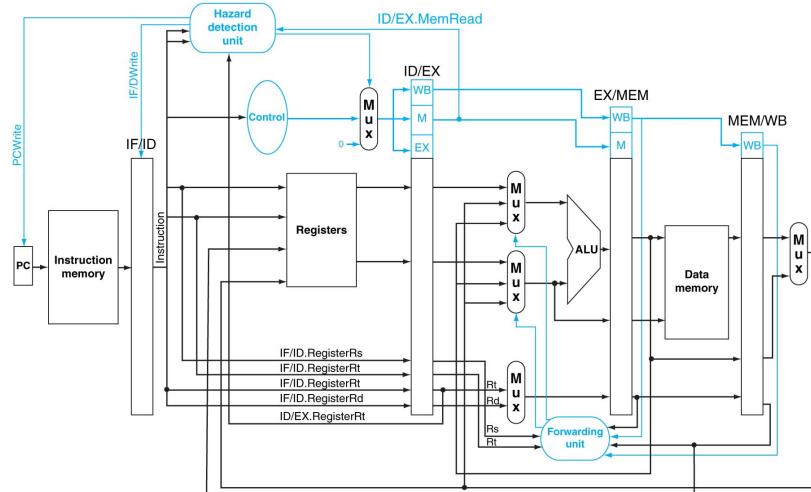
ABF\_AC1\_Pipelining

54

## Data Hazard - Load



## Datapath com Hazard Detection Unit



ABF\_AC1\_Pipelining

56

## Execução no pipeline com stalls e forwarding

Exemplo:

```

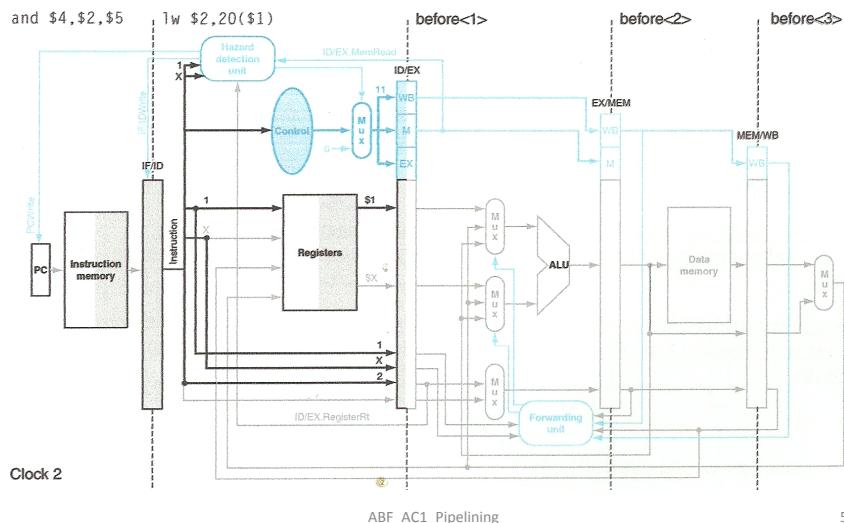
lw $2, 20($1)
and $4, $2, $5
or $4, $4, $2
add $9, $4, $2

```

ABF\_AC1\_Pipelining

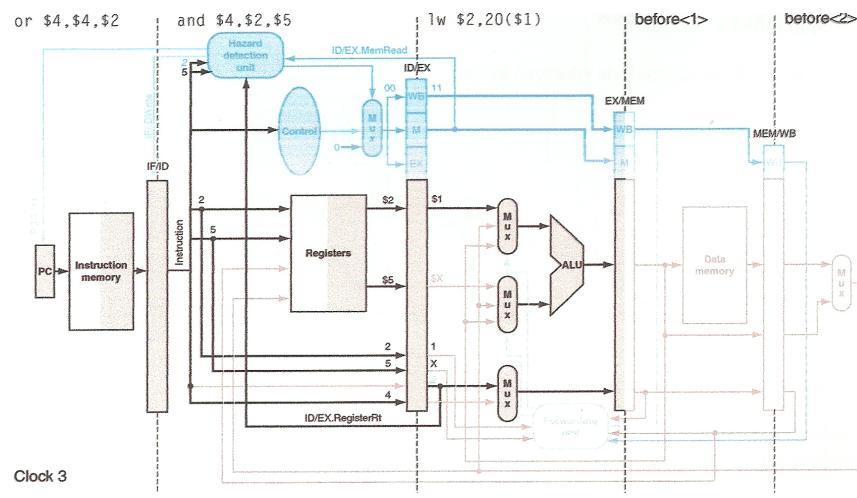
57

## Execução no pipeline com stalls e forwarding – 2º ciclo de relógio



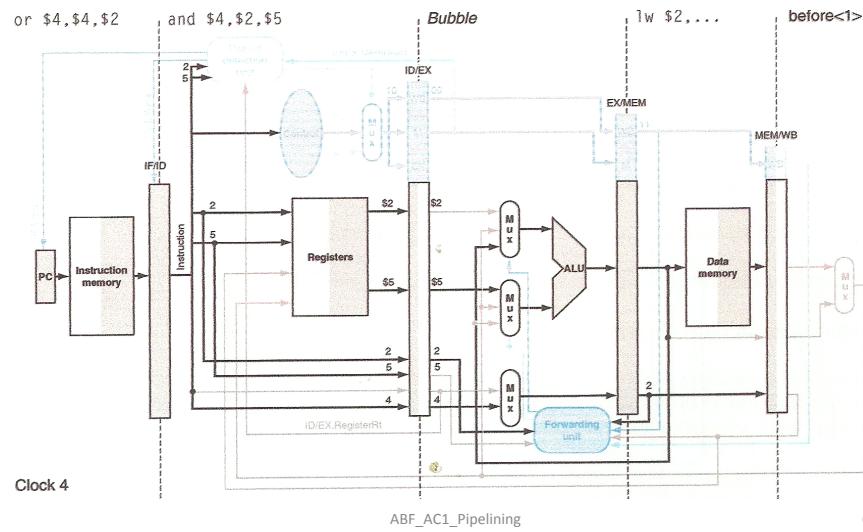
58

## Execução no pipeline com stalls e forwarding – 3º ciclo de relógio

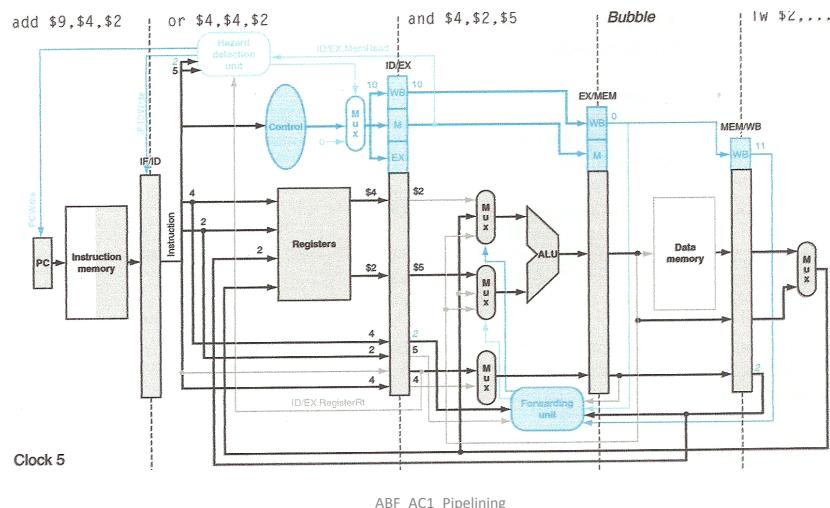


59

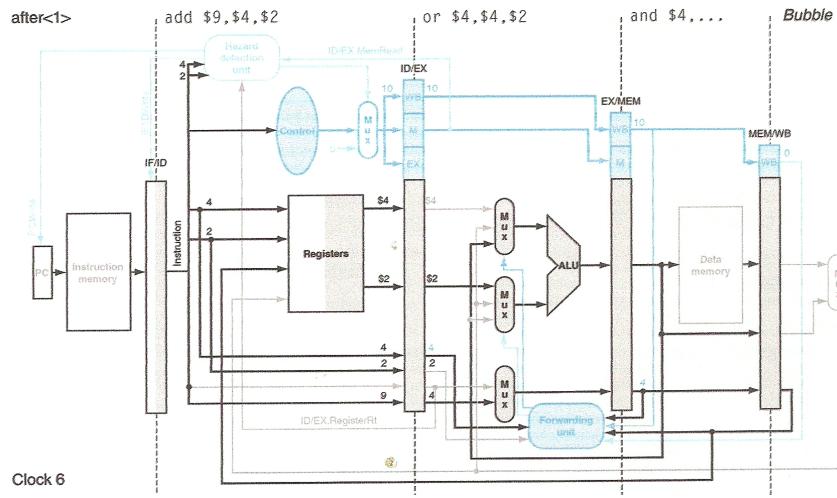
## Execução no pipeline com stalls e forwarding – 4º ciclo de relógio



## Execução no pipeline com stalls e forwarding – 5º ciclo de relógio



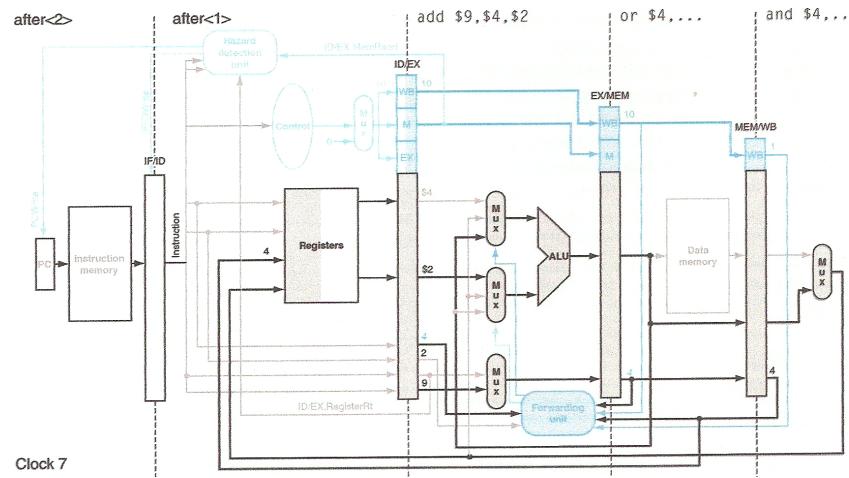
## Execução no pipeline com stalls e forwarding – 6º ciclo de relógio



ABF\_AC1\_Pipelining

62

## Execução no pipeline com stalls e forward – 7º ciclo de relógio



ABF\_AC1\_Pipelining

63

# Control Hazards

ABF\_AC1\_Pipelining

64

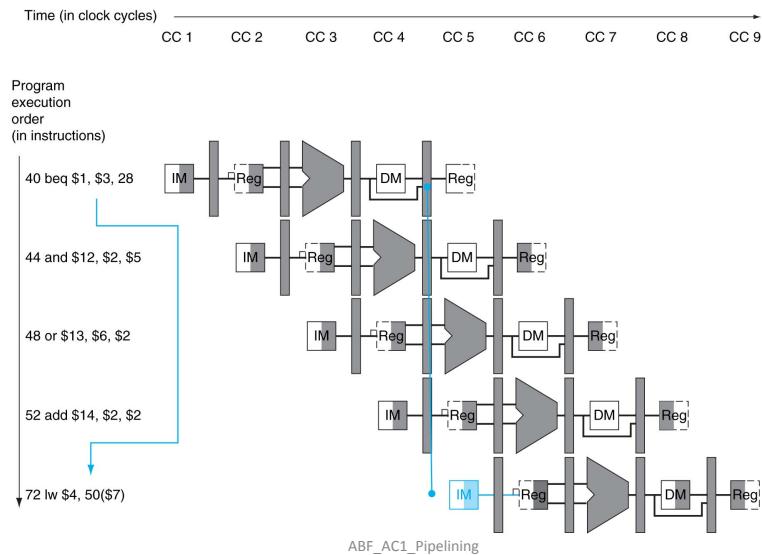
## Control Hazards

- Pipeline hazards introduzidos pelos branches
- Problema: determinar a próxima instrução de que fazer o *fetch*
- Como lidar com este tipo de hazards?
  - Prever resultado dos branch (*branch prediction*)

ABF\_AC1\_Pipelining

65

## Branch Hazard



## Static Branch Prediction

Previsão *branch not taken*:

1. Assumir que a condição de branch não se verifica (*branch not taken*) e continuar a execução
  - Se a condição de branch se verificar as instruções que estão nas fases de IF e ID têm de ser descartadas e continuar a execução na instrução alvo do branch
  - “Descartar Instruções” – colocar a zero os respetivos sinais de control
    - Semelhante ao que foi feito para os *loads*, só que agora para os andares IF, ID e EX (e não apenas para ID como no caso dos *load*)

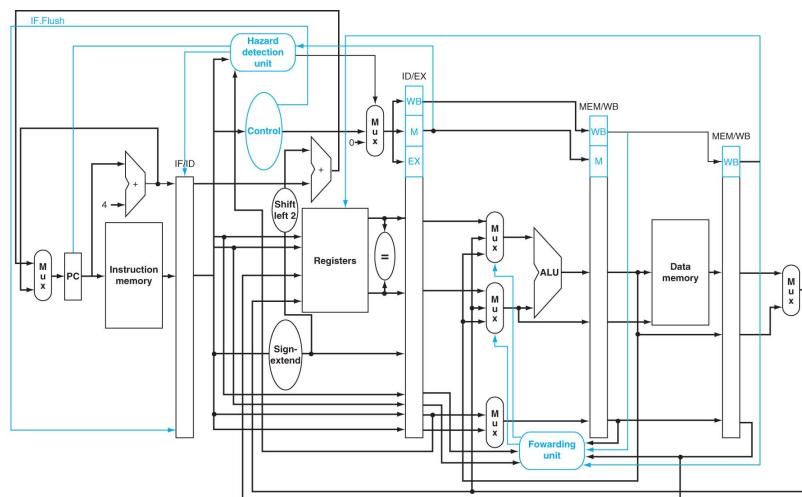
## Modificar Datapath

- Antecipar a avaliação da condição de branch e o cálculo do endereço-alvo
1. Avaliação da condição de igualdade não exige a ALU – basta XOR bit a bit os 2 registos e fazer o OR do resultado – pode ser feito no andar ID
    - introduz a necessidade de forwarding para ID stage
    - Se a instrução imediatamente anterior escrever num dos registos da condição do branch → **Stall**
  2. Cálculo do endereço-alvo: o somador dedicado pode ser colocado no andar ID

ABF\_AC1\_Pipelining

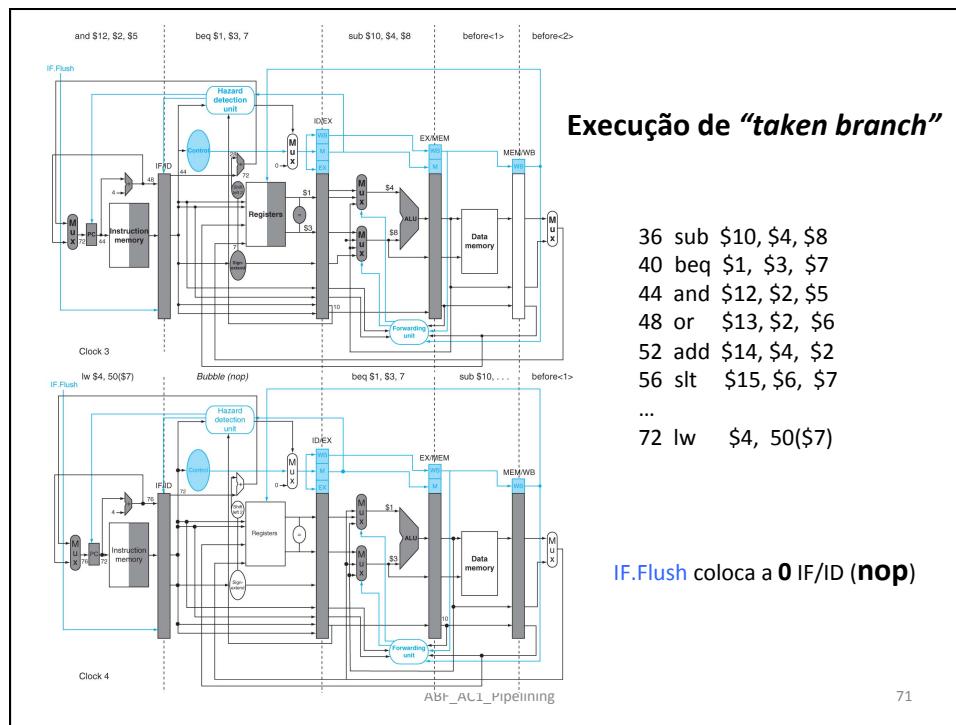
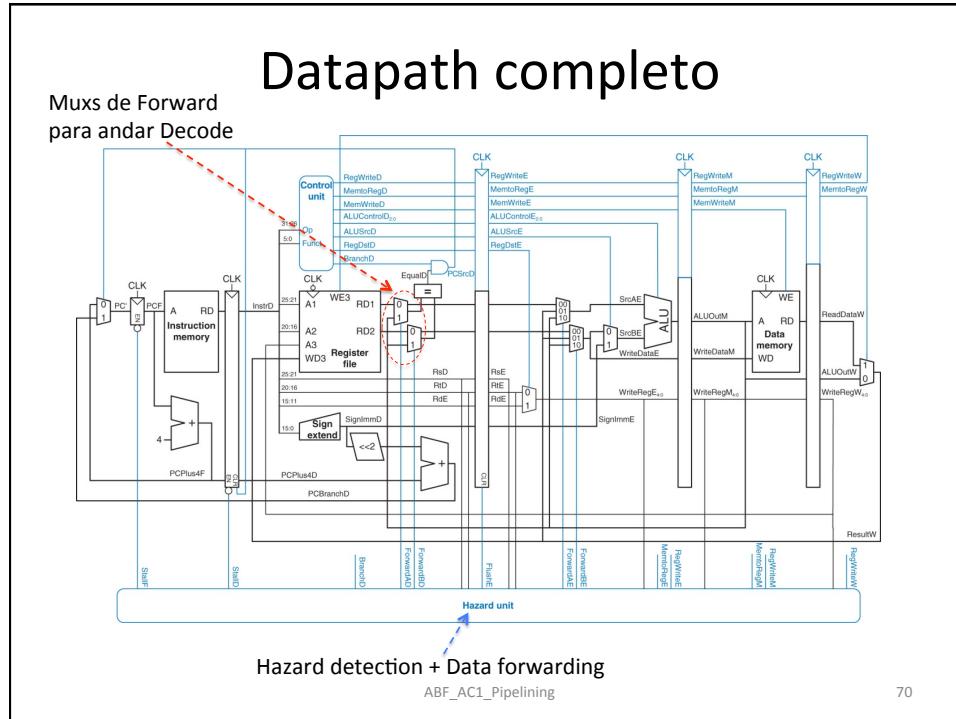
68

## Datapath + Control: final



ABF\_AC1\_Pipelining

69



## Dynamic Branch Prediction

- *Branch taken (not taken)* – previsão estática, independente do comportamento real do branch (p.ex. quando o *branch* corresponde à execução de um ciclo a previsão é, a maior parte das vezes, errada)
- Alternativa: previsão dinâmica, baseada na história da execução da instrução
  - *Branch Prediction Buffer (Branch History Table)* pequena memória acedida pelos bits menos significativos do endereço da instrução de branch em que cada posição da memória tem um bit que indica se o branch foi ou não “taken” na anterior execução.

## Pipelining (sumário)

- Pipelining é um conceito fundamental
  - Multiplas etapas usam recursos distintos
  - Instrução seguinte inicia-se enquanto se trabalha na instrução atual
  - Limitado pelo tempo exigido pelo andar mais longo (e por enchimento/esvaziamento do pipeline)
  - Importante detetar e resolver hazards para otimizar desempenho

# Pipelining (sumário)

- Fatores que facilitam pipelining
  - Todas as instruções do mesmo comprimento
  - Poucos formatos de instrução
  - Operandos em memória apenas em loads e stores
- O que dificulta pipelining?
  - structural hazards: se só existisse uma memória...
  - control hazards: instruções de branch
  - data hazards: uma instrução depende de outra anterior
- Construiu-se um pipeline simples (sem structural hazards) e viu-se como tratar os hazards
- Por tratar:
  - Exceções (exception handling – a tratar em AC II)
  - Técnicas usadas para aumentar o desempenho em processadores mais recentes: multiple issue (*superscalar architectures*) out-of-order execution, etc.