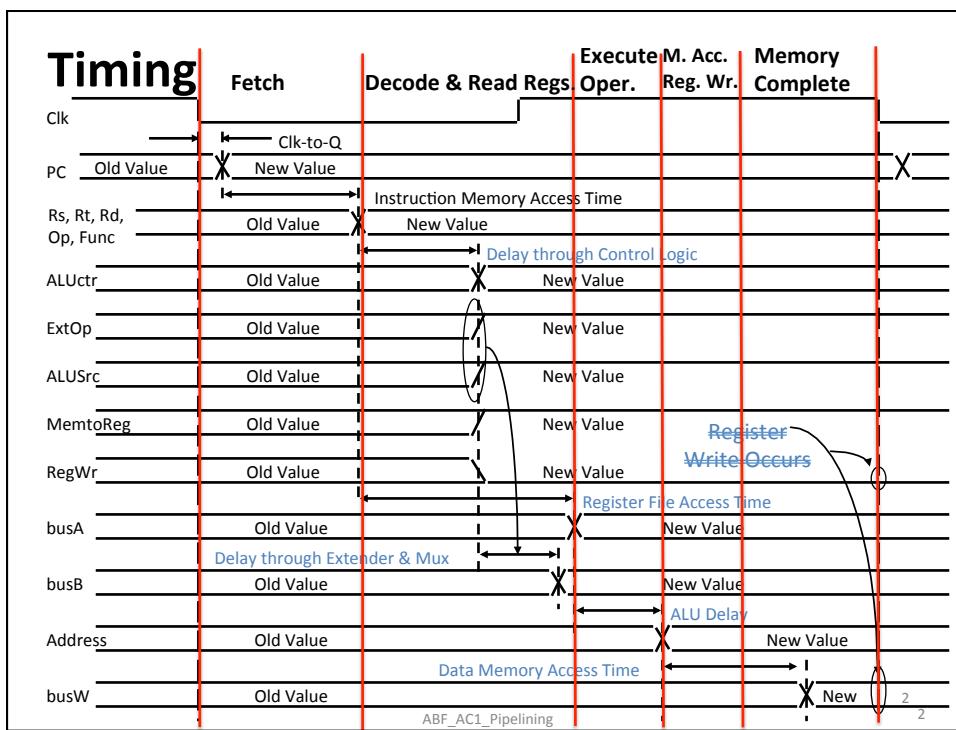


Pipelining

António de Brito Ferrari

ferrari@ua.pt



Melhorar Single-Cycle CPU

1. Multi-cycle:

1. Divisão da execução da instrução em 5 ciclos de relógio
2. Economia de meios – atualização do PC calculada pela ALU, poupando um somador dedicado - a mesma componente do datapath reutilizada em diferentes ciclos para realizar operações diferentes

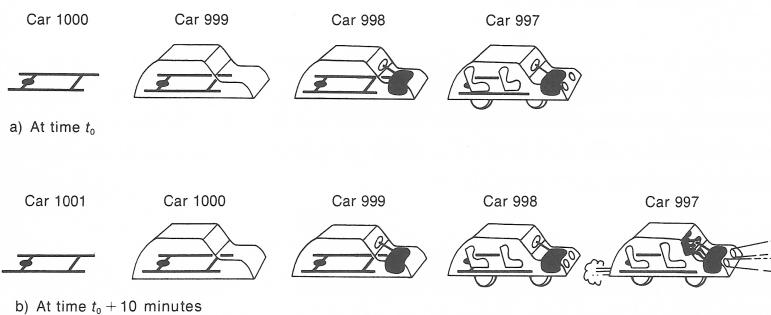
2. Pipelined:

1. Divisão da execução da instrução em 5 ciclos de relógio
2. Cada componente do datapath usada apenas num dos ciclos
3. Manter todas as componentes do datapath ativas em todos os ciclos – **princípio da cadeia de montagem (assembly line)**

ABF_AC1_Pipelining

3

Linha de Montagem: pipelined execution

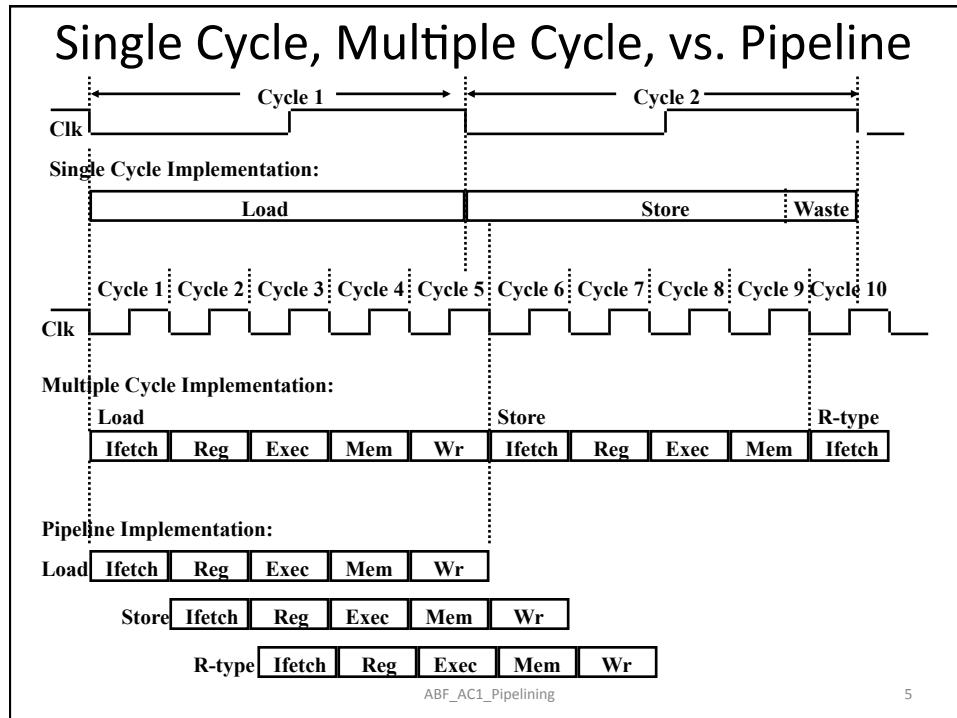


Montar um carro completo: 4×10 minutos = 40 minutos (**Latência**)
 Linha de montagem: um novo carro a cada 10 minutos (**Throughput**)

$$\text{Speedup}_{\text{pipeline/multicycle}} = 4$$

ABF_AC1_Pipelining

4



Tempos com a tecnologia atual

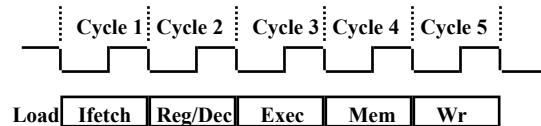
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, AND, OR, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

Tempos de cada componente da datapath (não tendo em conta os tempos da unidade de controle e dos multiplexers)

ABF_AC1_Pipelining

6

Execução de Load

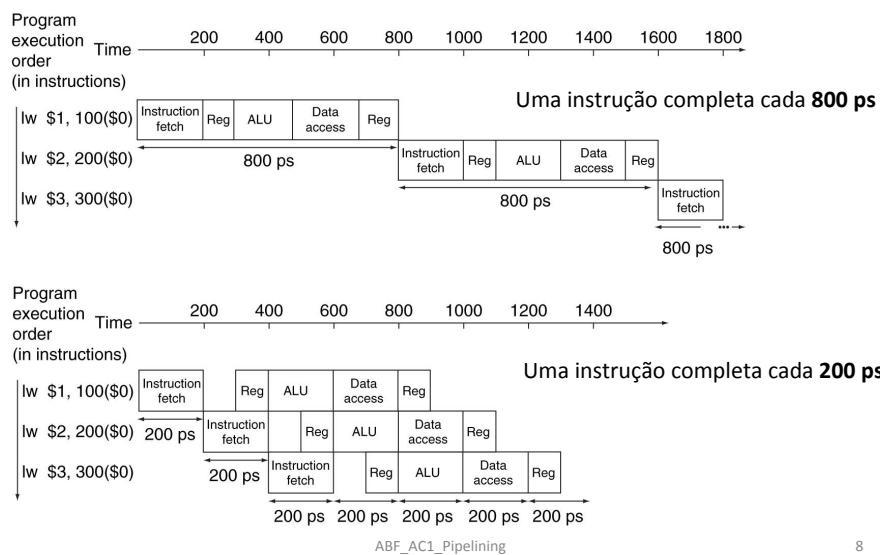


- Ifetch: Instruction Fetch
 - Fetch da instrução da Instruction Memory
- Reg/Dec: Registers Fetch e Instruction Decode
- Exec: Cálculo do endereço de memória
- Mem: Ler o dado da Data Memory
- Wr: Escrever o dado no banco de registros

ABF_AC1_Pipelining

7

Multicycle vs. Pipelined



ABF_AC1_Pipelining

4

ISA e Pipelining - MIPS

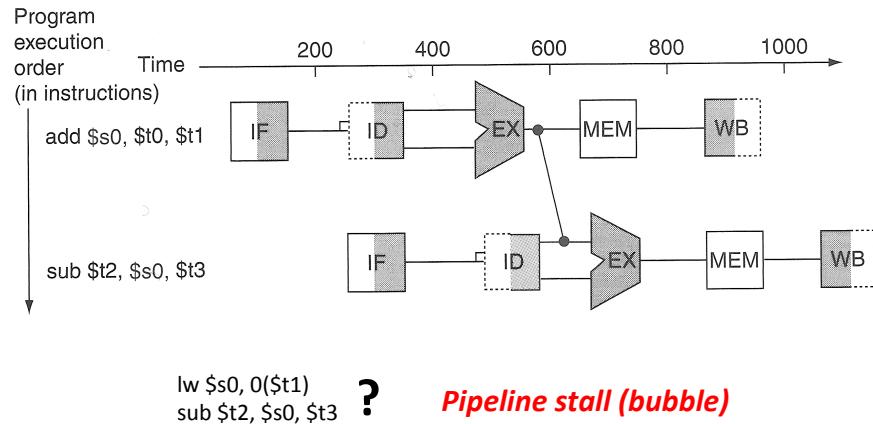
1. Todas as instruções com o mesmo número de bits (32) – facilita Ifetch num único ciclo seguido de um ciclo para Idecode
 - IA-32: comprimento das instruções varia entre 1 e 17 bytes – dificulta pipelining
2. Poucos formatos de instrução. Registos dos operandos sempre na mesma posição – leitura dos registos pode-se fazer logo após o fetch da instrução (se assim não fosse a leitura dos registos apenas se podia fazer após a descodificação da instrução obrigando a um pipeline de 6 andares)
3. Operandos em memória só nas instruções de *load* e *store*. Fase *Execute* a seguir à leitura de registos permite executar a operação ou calcular endereço de memória
 - IA-32: operandos em memória – necessário introduzir fases *MemAddr*, *MemAccess* e *Execute*
4. Operandos alinhados em memória – dados podem ser transferidos num único acesso à memória

Pipeline: limitações ao desempenho

- Ideal: *speedup = no. de andares do pipeline*
- Real - obstáculos ao aumento do desempenho (**Hazards**):
 - Estruturais – o h/w não permite a combinação de algumas instruções no mesmo ciclo de relógio
 - Data Hazards – a admissão de novas instruções no pipeline tem de ser parada porque a execução numa das fases tem de ser parada à espera que outra fase complete para estarem disponíveis os dados de que precisa
 - Exemplo: add \$s0, \$t0, \$t1
sub \$t2, \$s0, \$t3

Data Hazzards

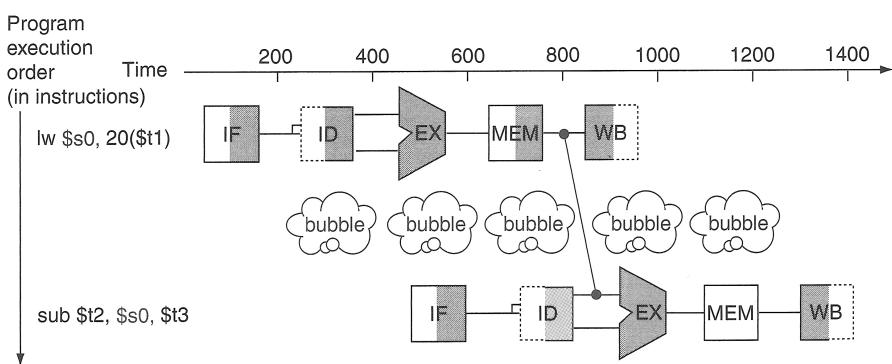
Solução: Forwarding



ABF_AC1_Pipelining

11

Data Hazards (2)



Software: Reordenação do código

ABF_AC1_Pipelining

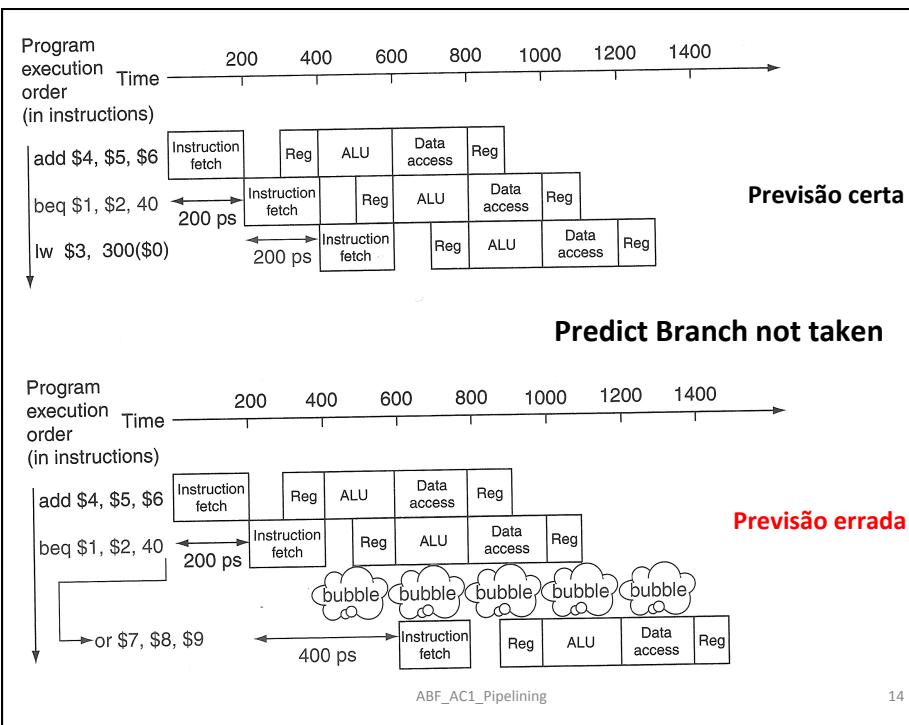
12

Control Hazards

- Branch: necessário fazer o fetch da instrução seguinte quando o resultado do branch ainda não foi calculado
 - Branch Prediction

ABF_AC1_Pipelining

13



14

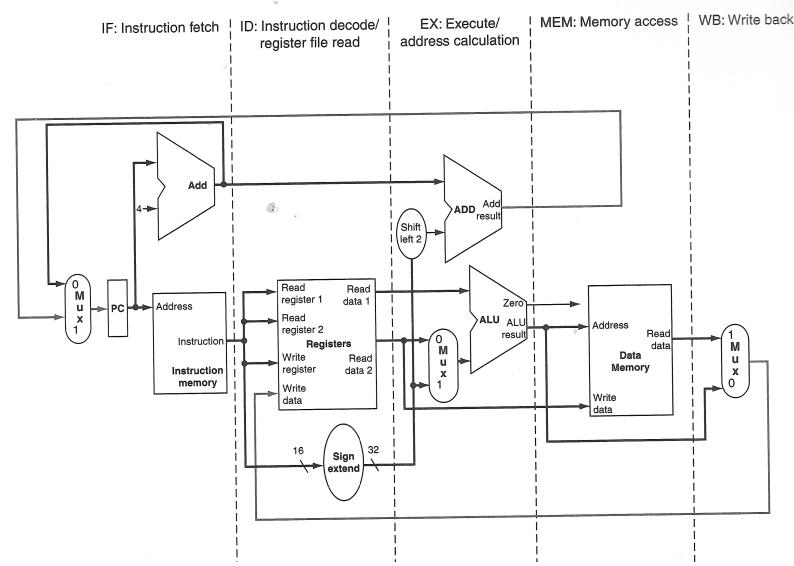
Branch Prediction

- Formas mais elaboradas:
 - Dynamic Branch Prediction – baseada no histórico do comportamento do branch
 - Consegue-se atingir 90% de previsões certas
- Alternativa: *delayed branch* (MIPS)

ABF_AC1_Pipelining

15

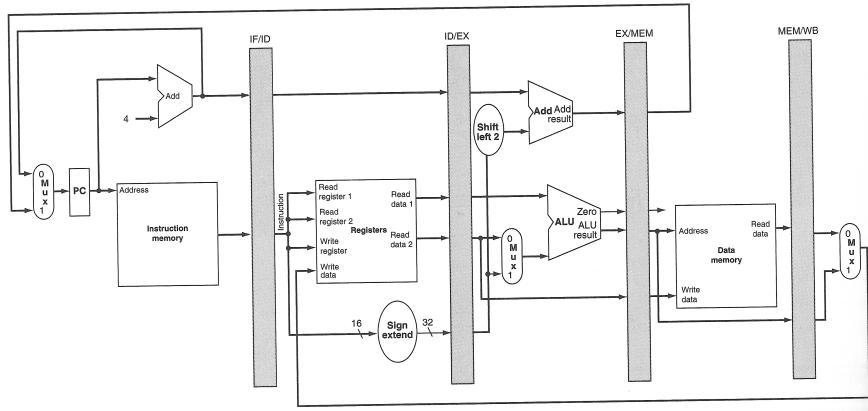
Single-Cycle Datapath



ABF_AC1_Pipelining

16

Versão pipelined do Datapath

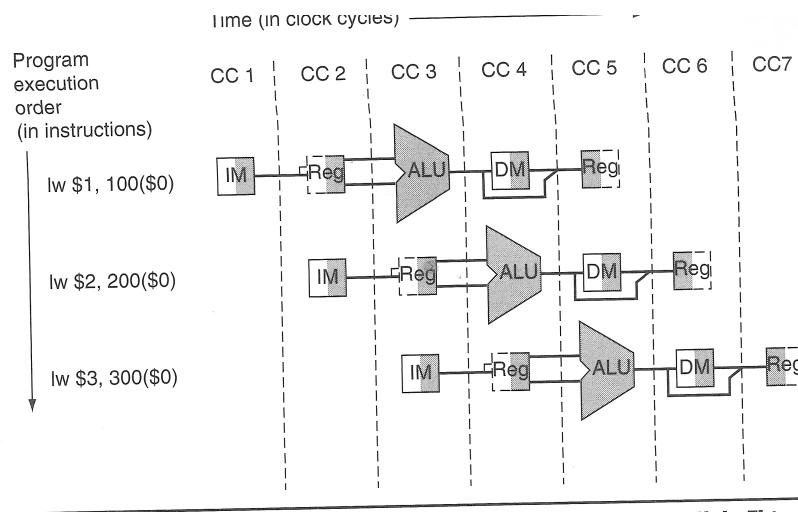


Divisão do datapath em andares – cada andar executa uma fase das instruções
Registros IF/ID, ID/EX, EX/MEM e MEM/WB armazenam os resultados obtidos no andar a Montante e fornecem-os ao andar seguinte

ABF_AC1_Pipelining

17

Execução supondo que cada instrução executa na sua cópia do datapath



ABF_AC1_Pipelining

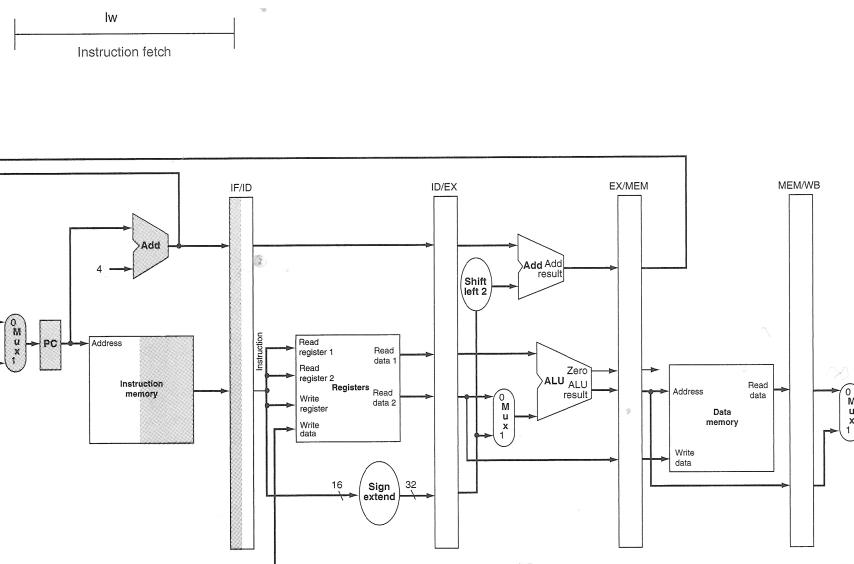
18

1. Pipelined Datapath

ABF_AC1_Pipelining

19

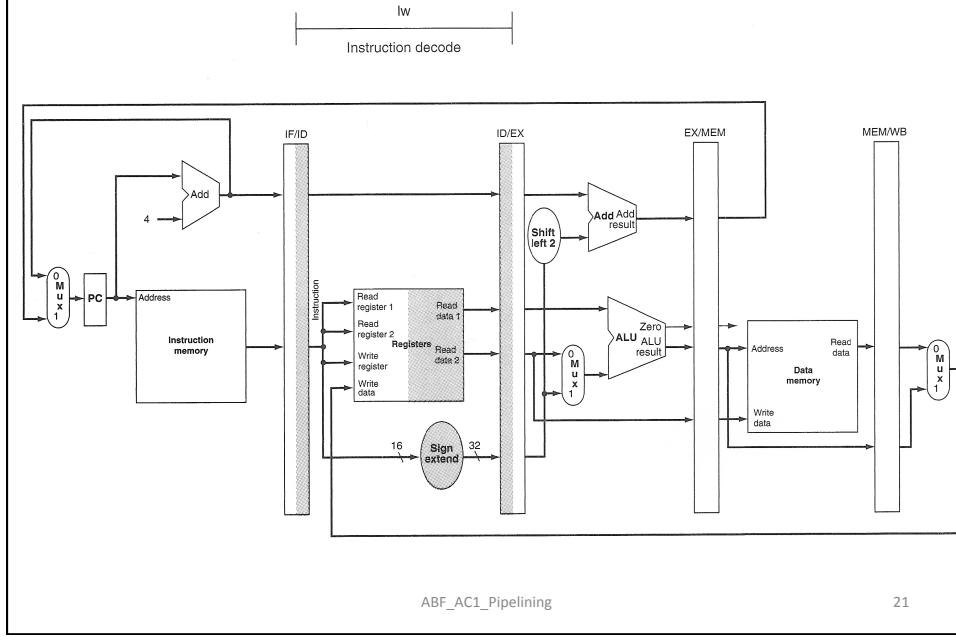
Secção ativa do Datapath durante o Fetch da instrução



ABF_AC1_Pipelining

20

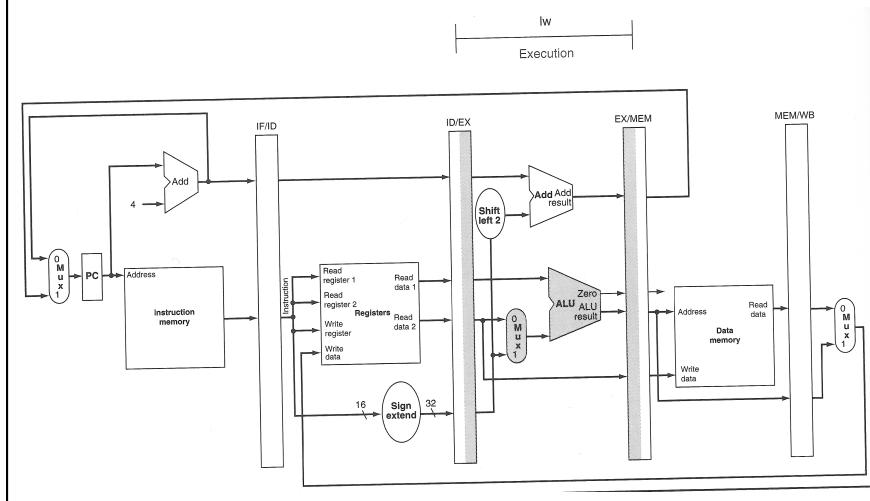
Secção ativa do Datapath durante a Descodificação da instrução e leitura dos registos



ABF_AC1_Pipelining

21

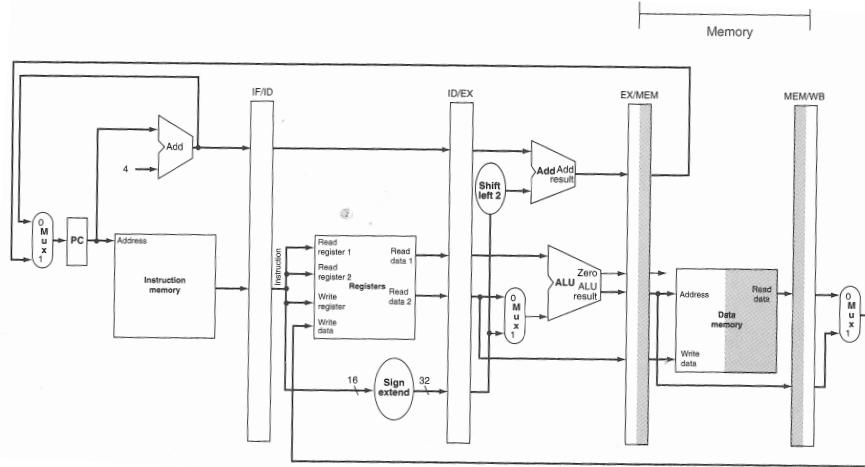
Secção ativa do datapath durante a 3^a fase de execução de Load (cálculo do endereço)



ABF_AC1_Pipelining

22

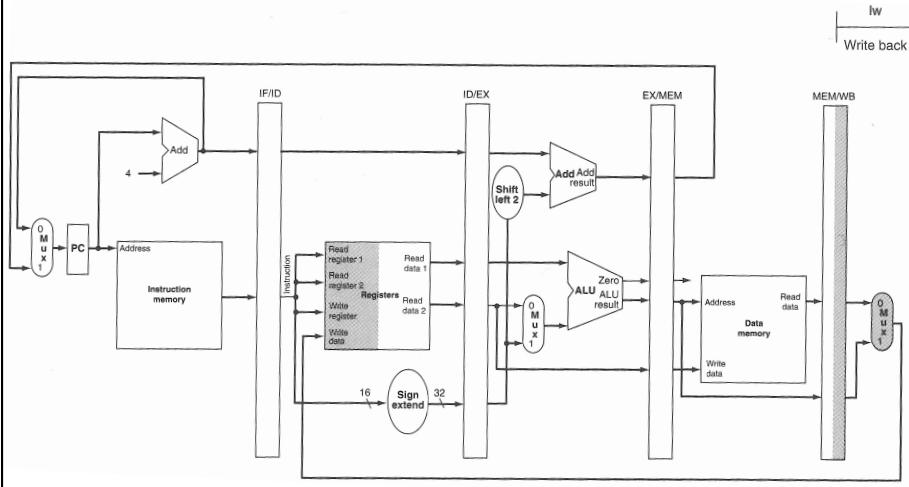
Secção ativa do datapath durante a 4^a fase de execução de Load (acesso à memória)



ABF_AC1_Pipelining

23

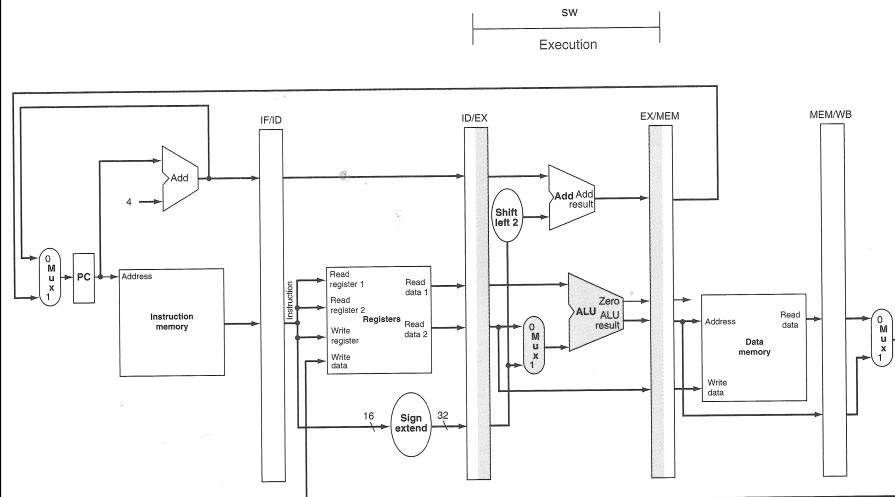
Secção ativa do datapath durante a 5^a fase de execução de Load (escrita no registo)



ABF_AC1_Pipelining

24

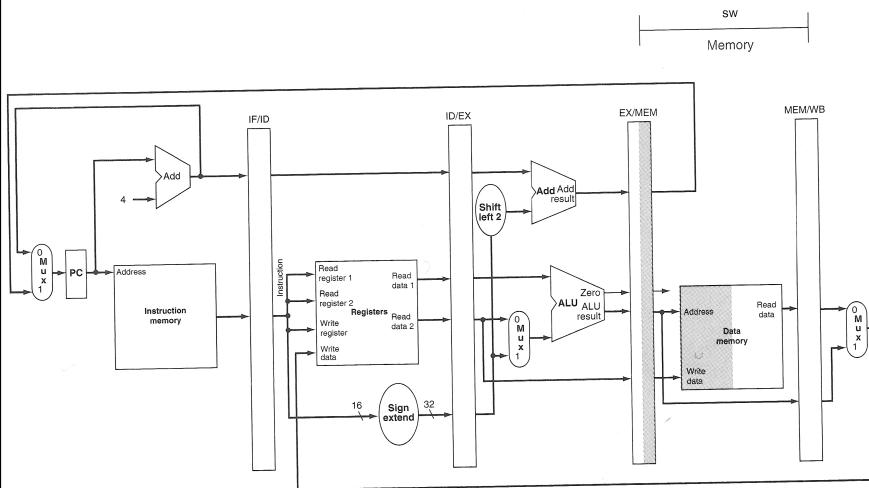
Secção ativa do datapath durante a 3^a fase de execução de Store
– conteúdo de Rt, que vai ser escrito na memória, armazenado em EX/MEM



ABF_AC1_Pipelining

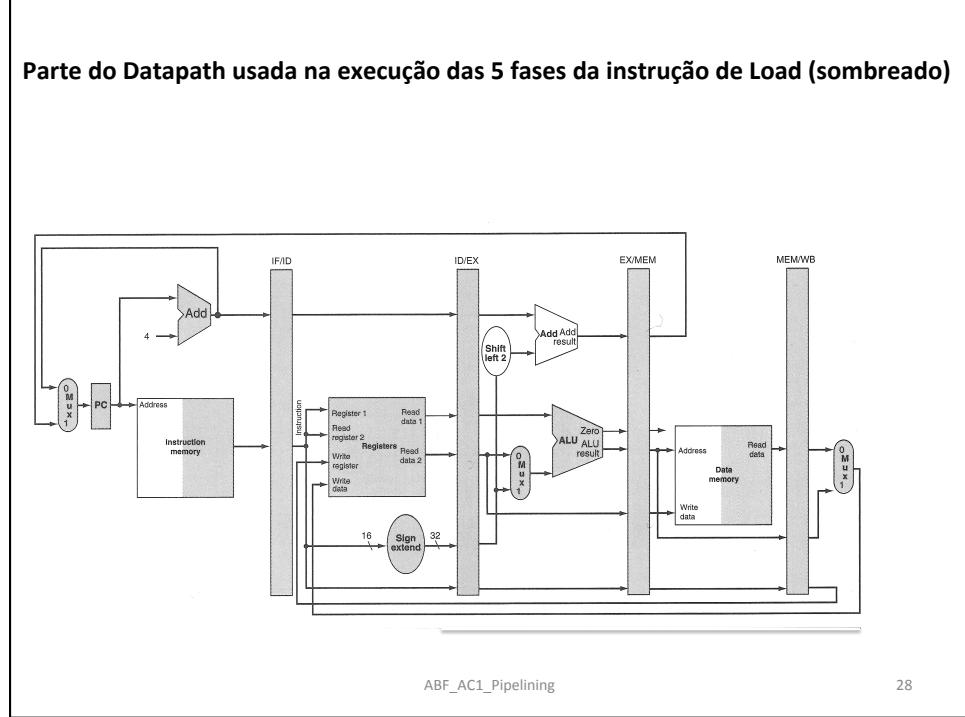
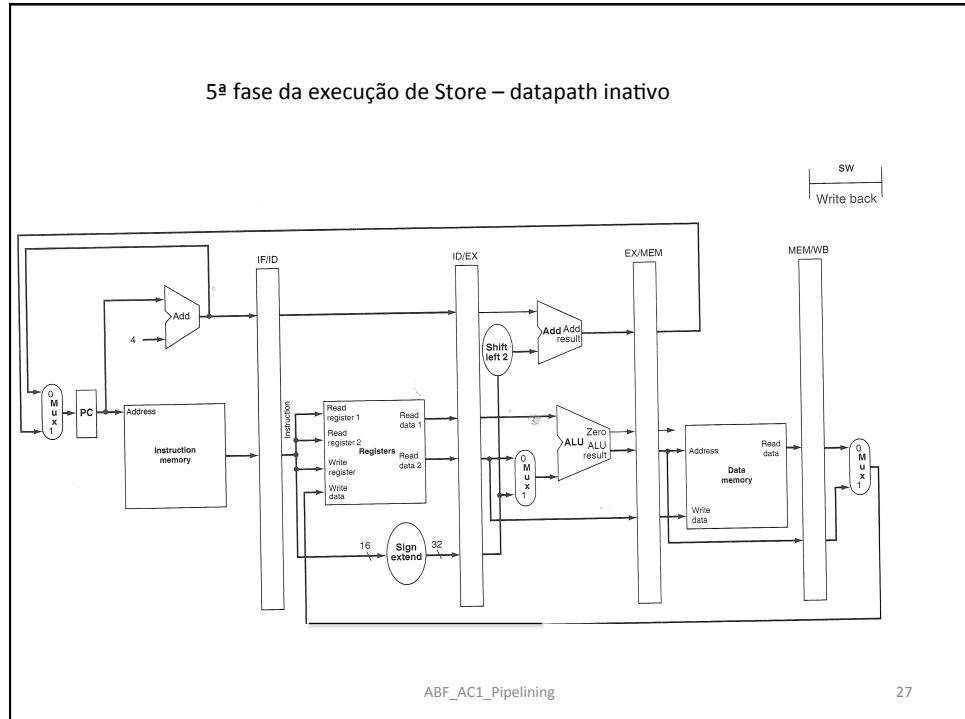
25

Secção ativa do datapath durante a 4^a fase de execução de Load (escrita em memória)

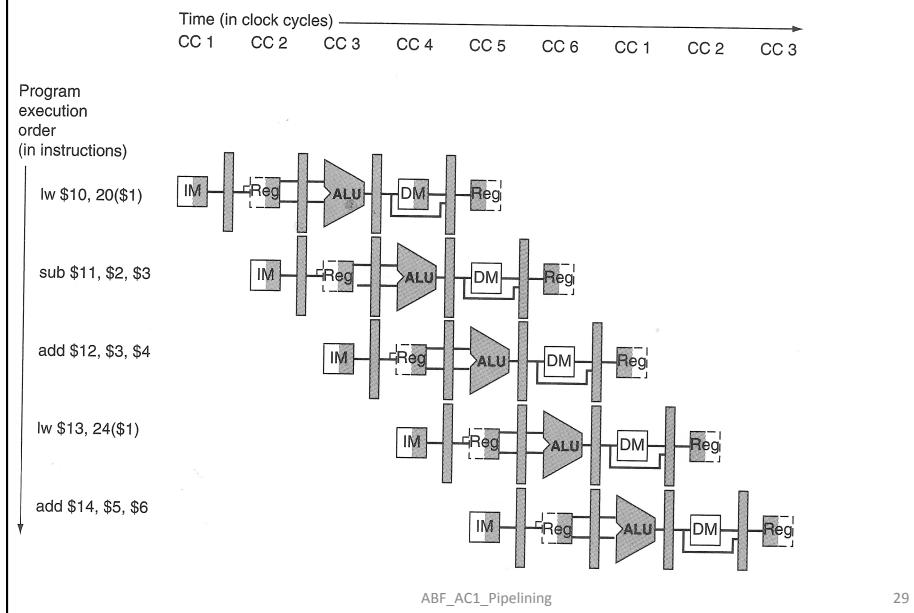


ABF_AC1_Pipelining

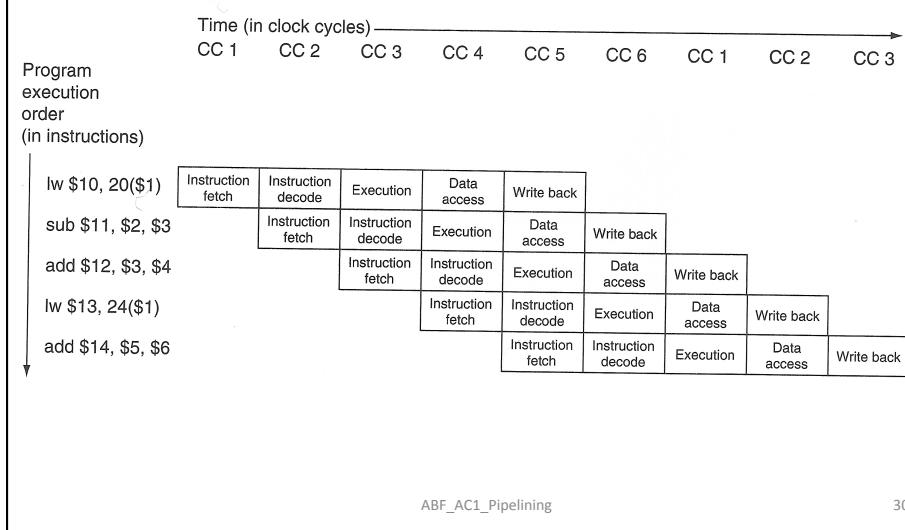
26

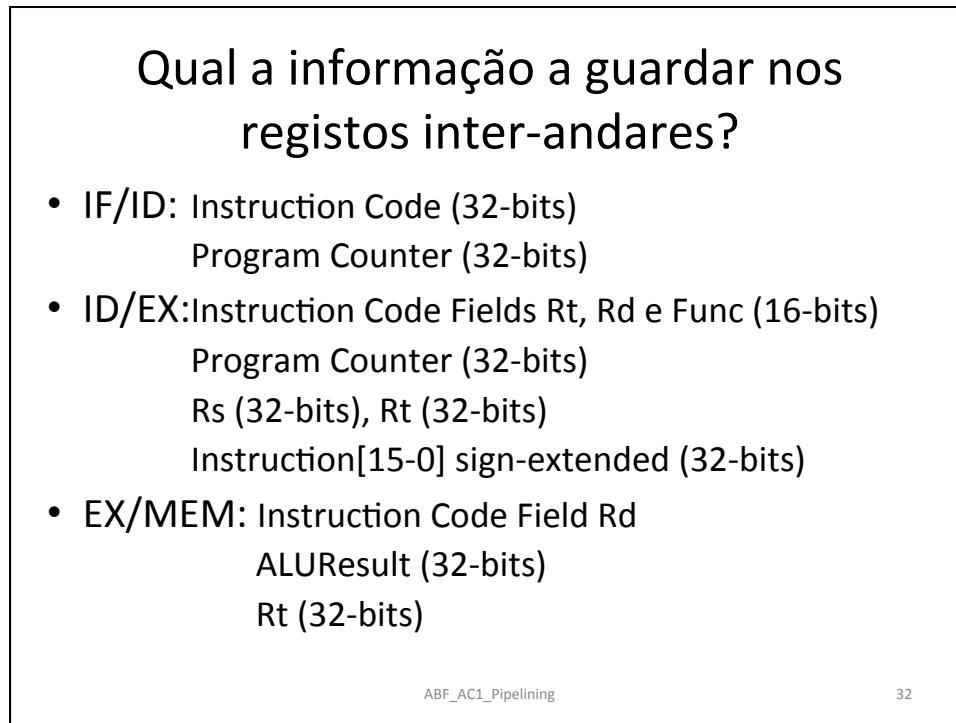
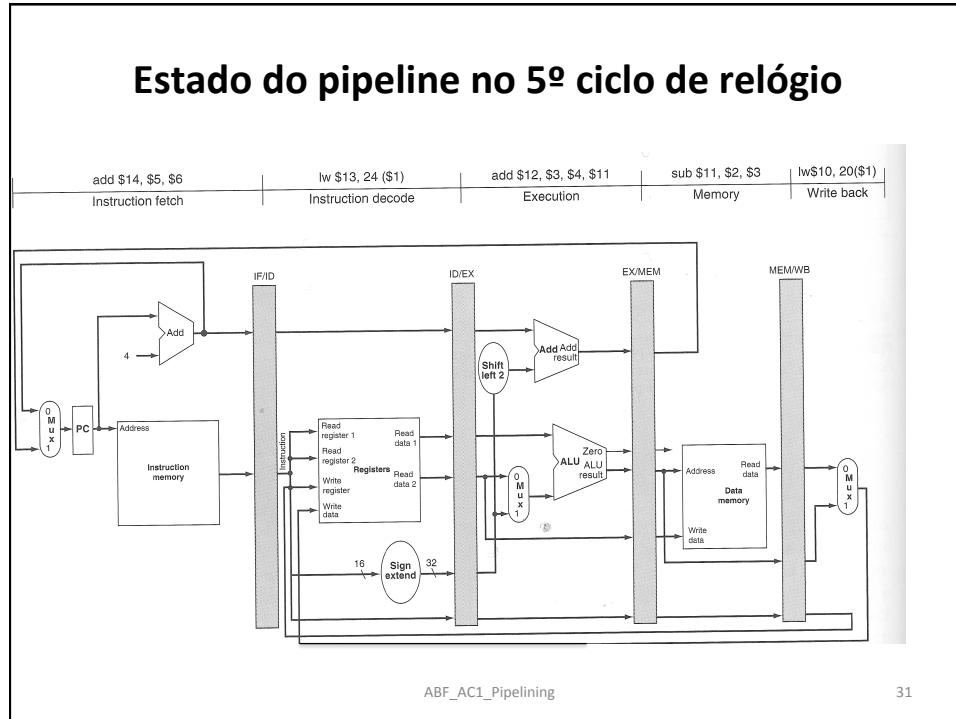


Representação gráfica da execução de uma sequência de 5 instruções



Outra forma de representação



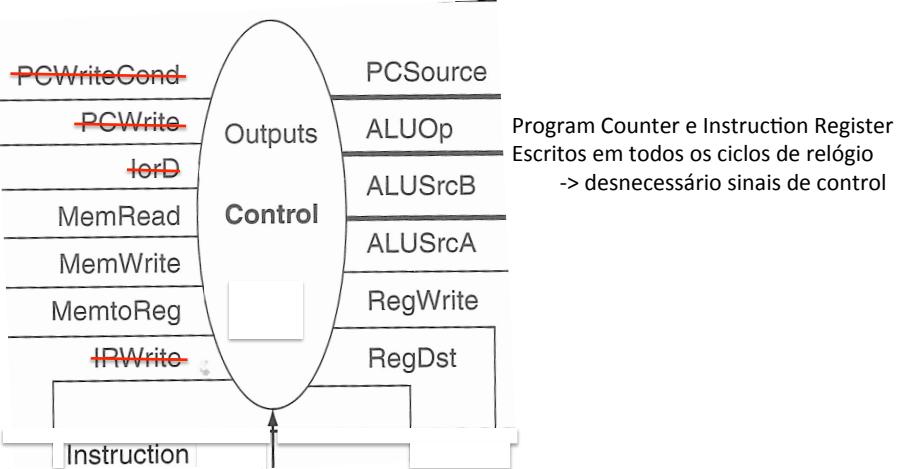


2. Pipelined Control

ABF_AC1_Pipelining

33

Sinais de controle: diferenças com multi cycle



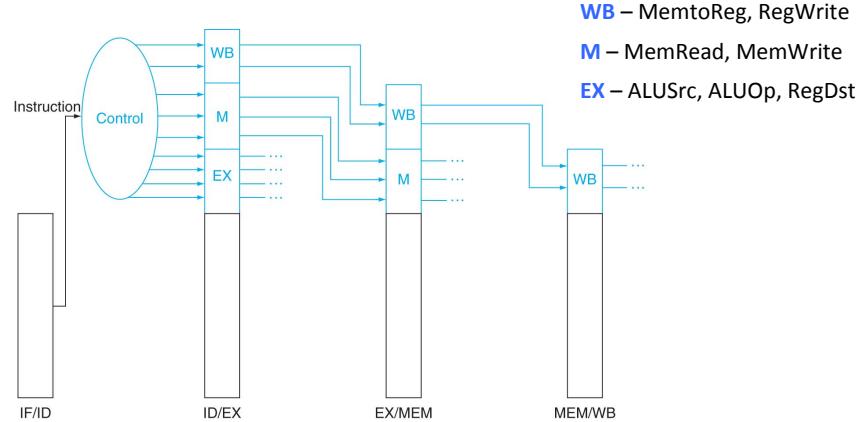
Andares do pipeline em que os sinais de controle atuam:

ALUSrcB, ALUOp, RegDst – EXecute stage
 MemRead, MemWrite – MEMory stage
 MemtoReg, RegWrite – WriteBack stage

ABF_AC1_Pipelining

34

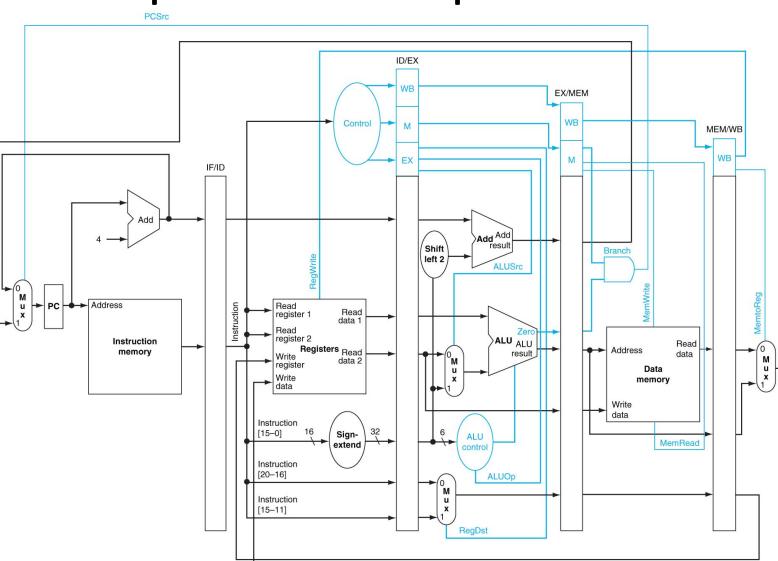
Encaminhamento dos Sinais de Control



ABF_AC1_Pipelining

35

Pipelined Datapath + Control



ABF_AC1_Pipelining

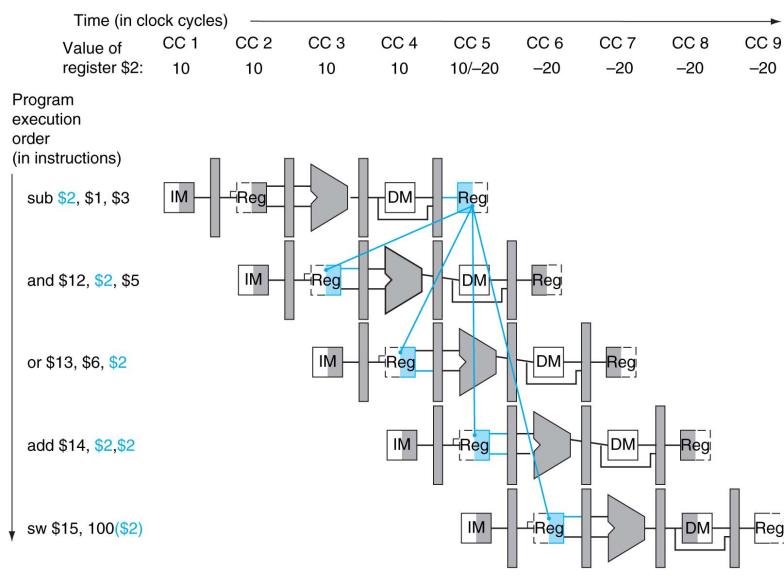
36

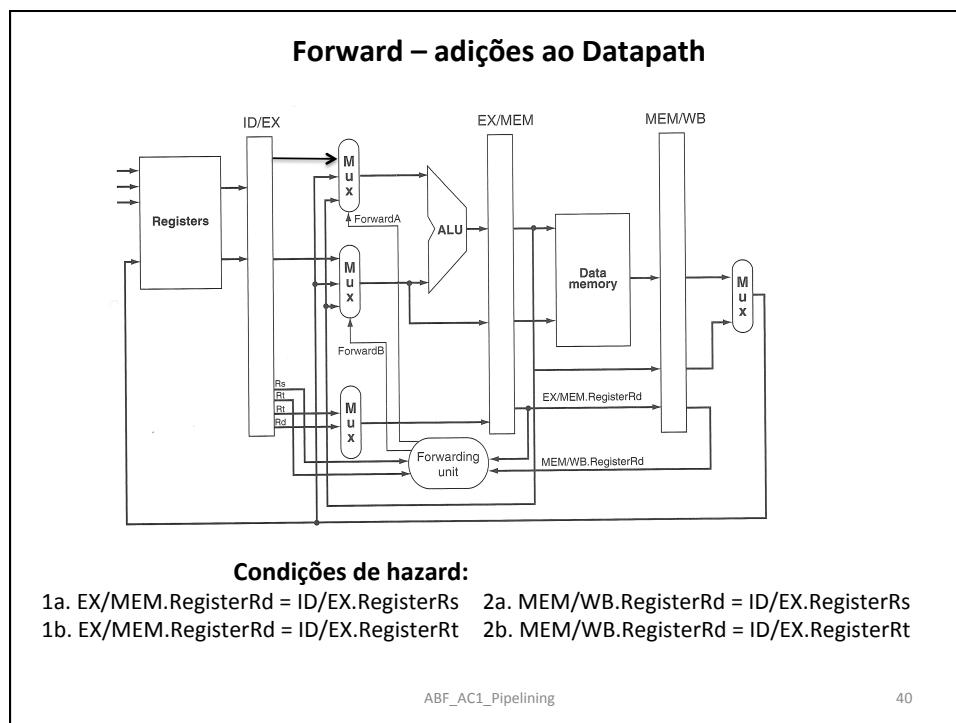
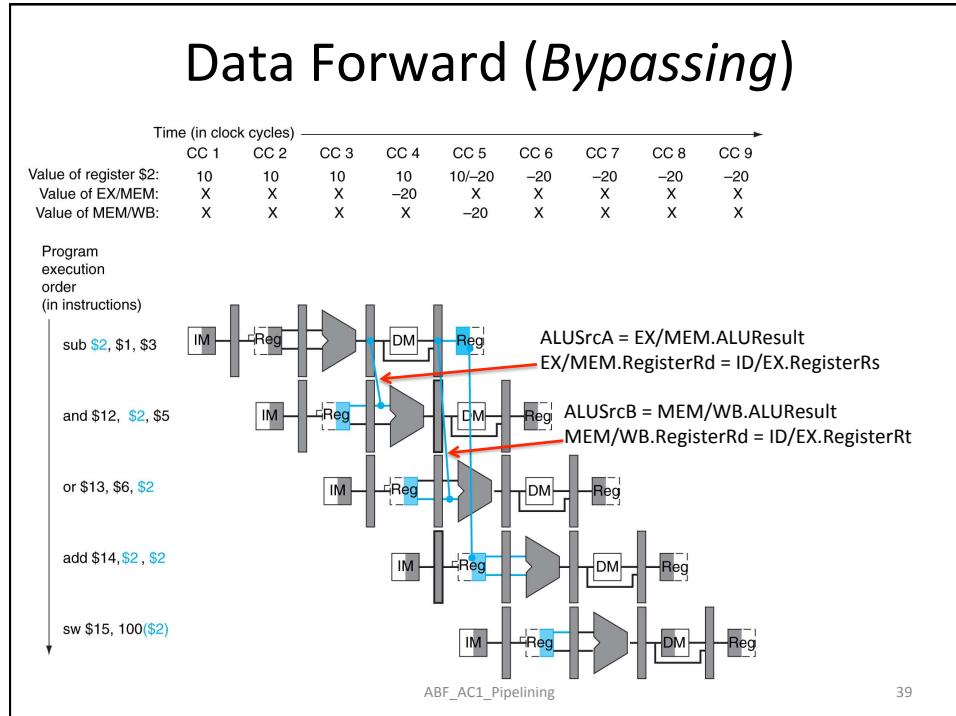
3. Pipeline Hazards

ABF_AC1_Pipelining

37

Data Hazards





Forward - Control

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.

1. EX Hazard:

```
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
```

2. MEM Hazard:

```
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
```

ABF_AC1_Pipelining

41

Forward Control (2)

add \$1, \$1, \$2

add \$1, \$1, \$2

add \$1, \$1, \$3 # (EX/MEM.RegisterRd = ID/EX.RegisterRs) and (MEM/WB.RegisterRd = ID/EX.RegisterRs)

Resultado do 2º add

Resultado do 1º add

Na execução da 3ª instrução tem de se fazer o forward do resultado do 2º add:

MEM Hazard corrigido:

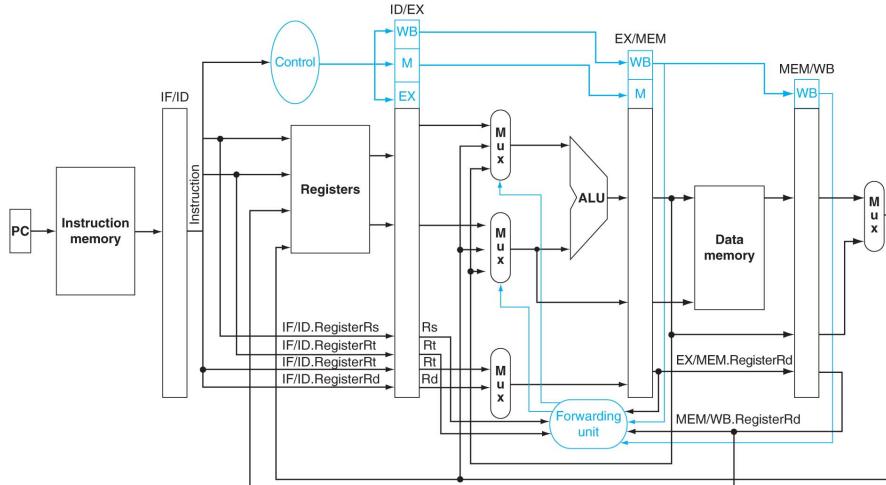
```
if not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and
       (EX/MEM.RegisterRd = ID/EX.RegisterRs)) and
       (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and
        (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01
```

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

ABF_AC1_Pipelining

42

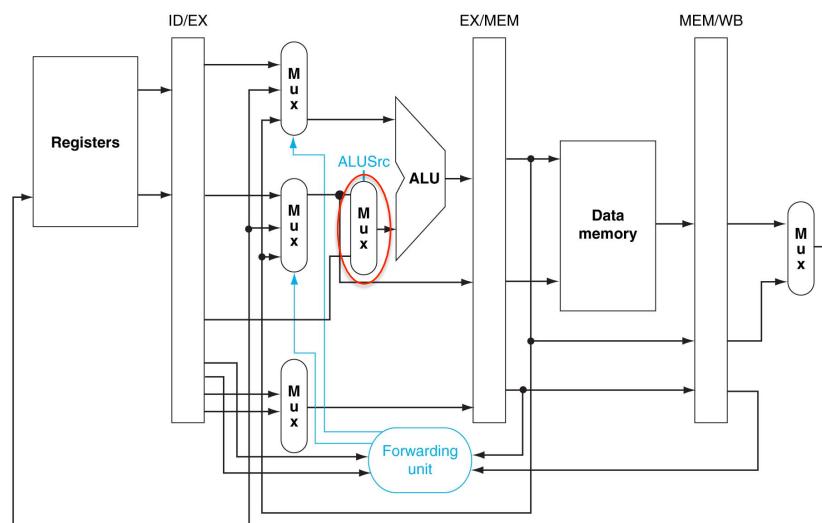
Datapath + Control com Forward



ABF_AC1_Pipelining

43

Datapath completado



ABF_AC1_Pipelining

44

Load/Store

lw \$1, 0(\$2)

sw \$1, 0(\$3)

...

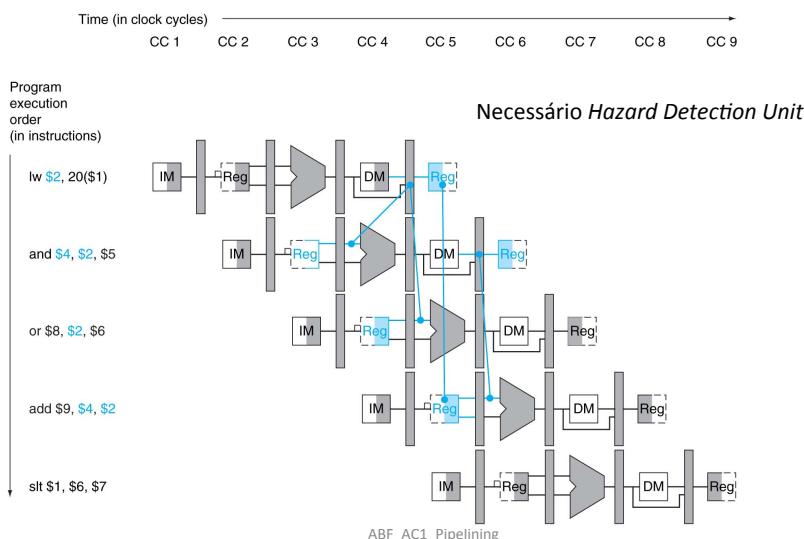
- Possível evitar *stall* do pipeline?

➤ Escreva a condição de forward e indique o reencaminhamento (forward) de dados a fazer

ABF_AC1_Pipelining

45

Data Hazards não resolueis sem stall



46

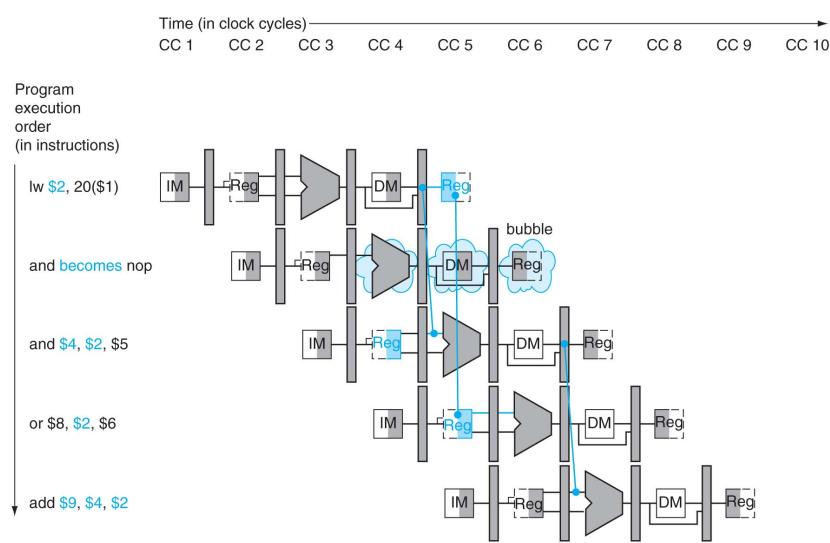
Hazard Detection Unit

- ```
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
```

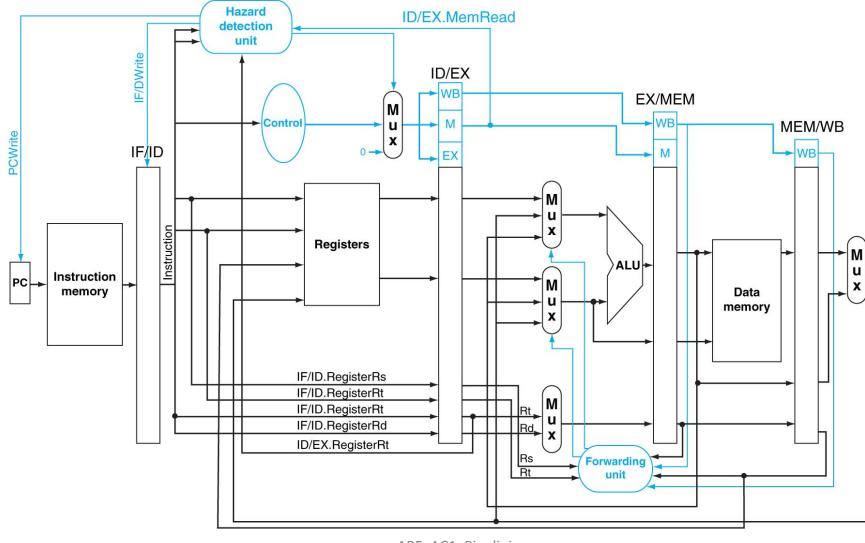
ABF\_AC1\_Pipelining

47

## Data Hazard - Load



## Datapath com Hazard Detection Unit



## Execução no pipeline com forwarding

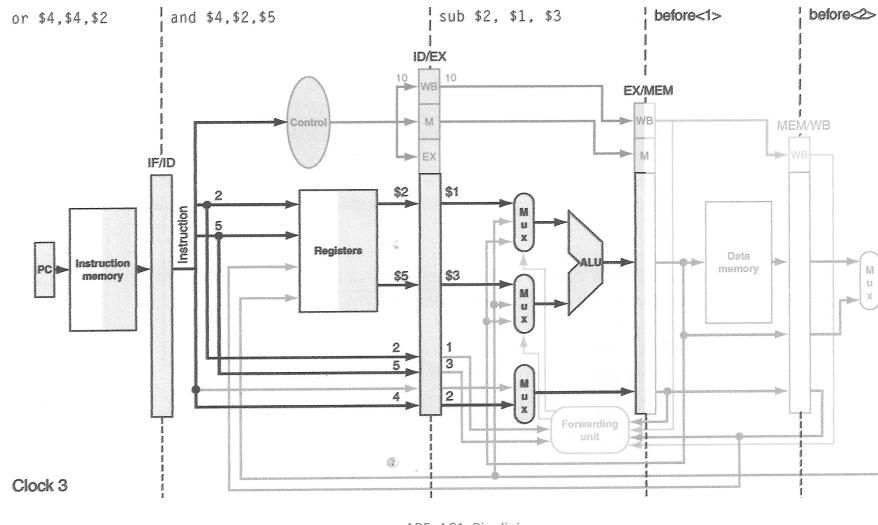
Exemplo:

```

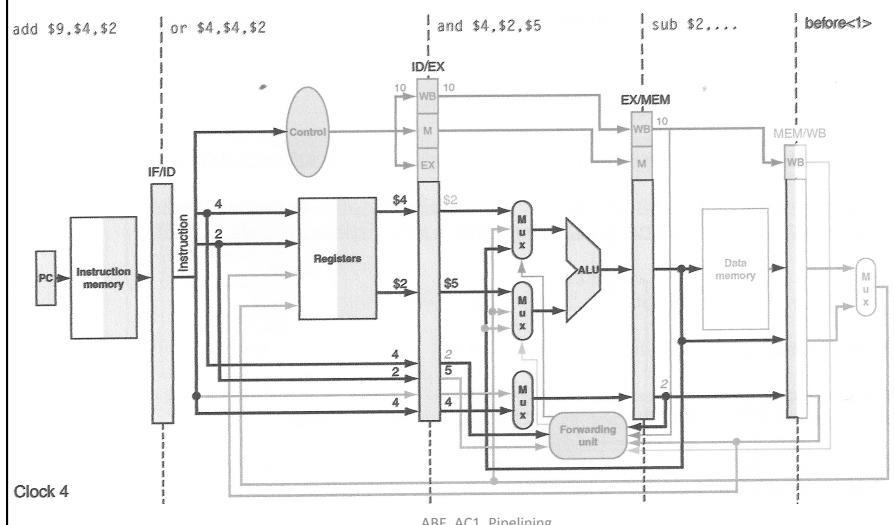
sub $2, $1, $3
and $4, $2, $5
or $4, $4, $2
add $9, $4, $2

```

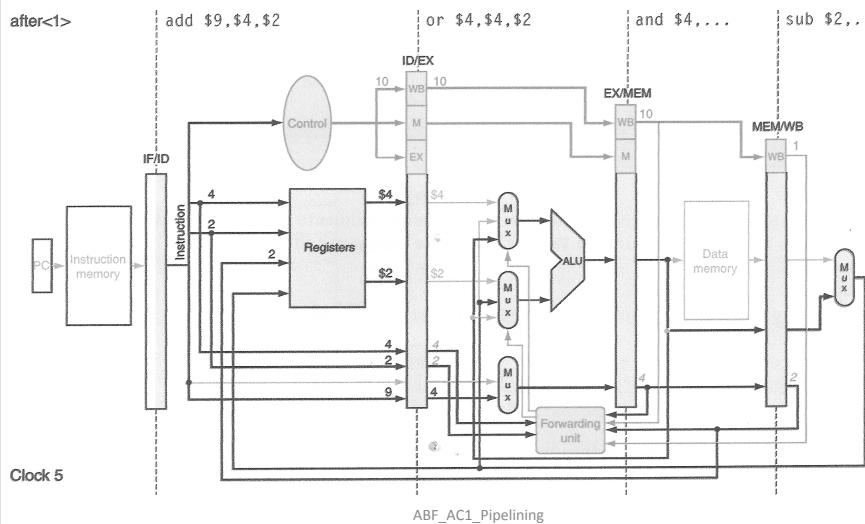
## Execução no pipeline com forwarding 3º ciclo de relógio



## Execução no pipeline com forwarding 4º ciclo de relógio

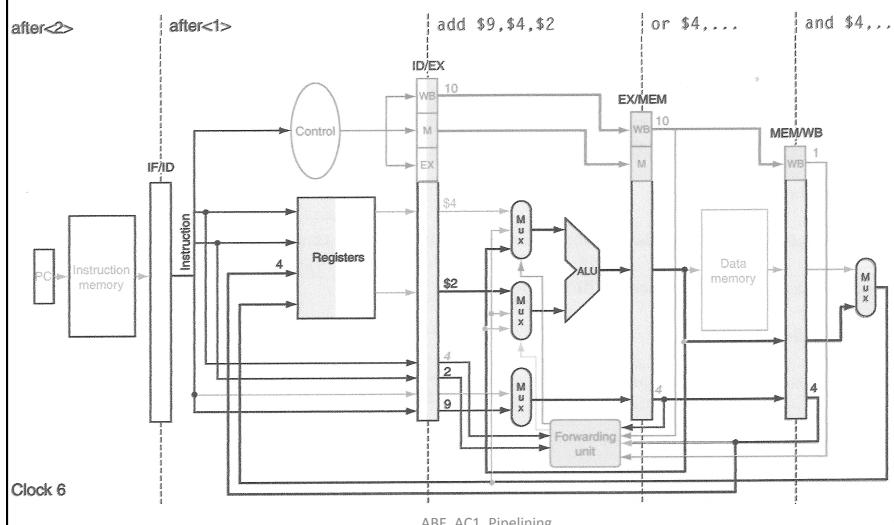


## Execução no pipeline com forwarding 5º ciclo de relógio



53

## Execução no pipeline com forwarding 6º ciclo de relógio



54

## Execução no pipeline com stalls e forward

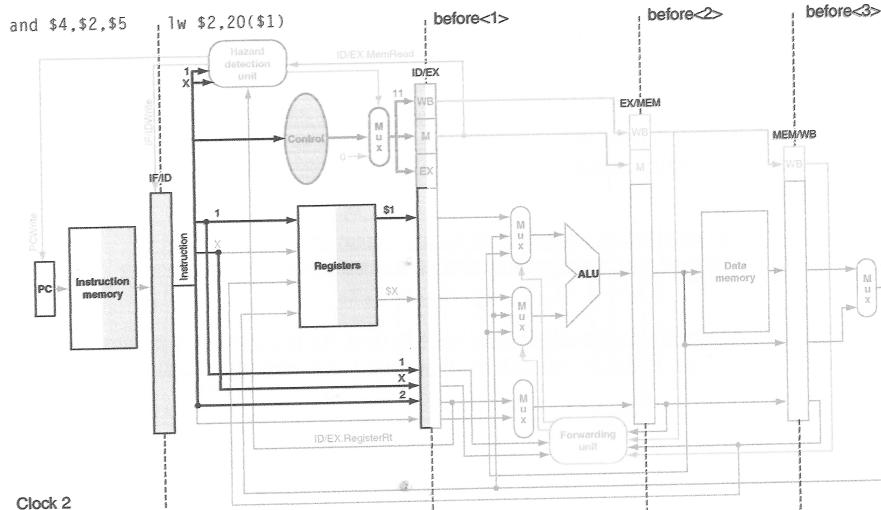
Exemplo:

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

ABF\_AC1\_Pipelining

55

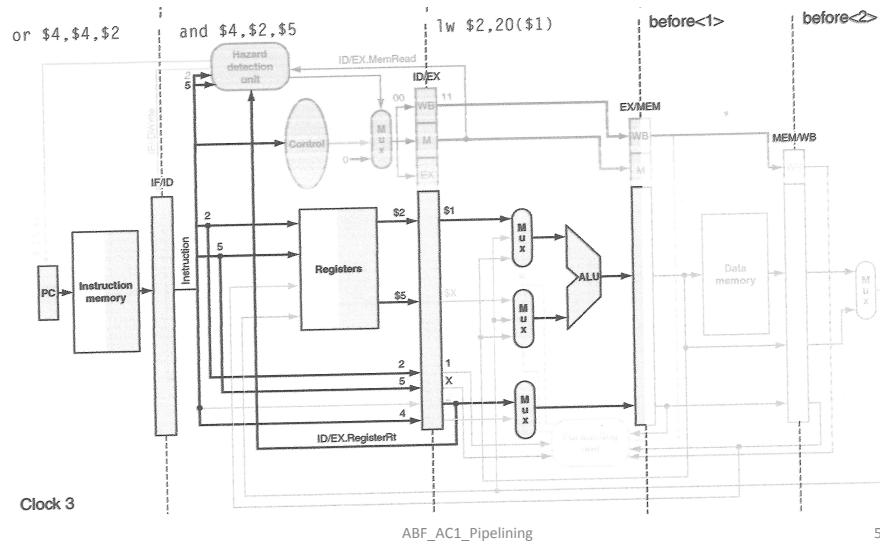
## Execução no pipeline com stalls e forward – 2º ciclo de relógio



ABF\_AC1\_Pipelining

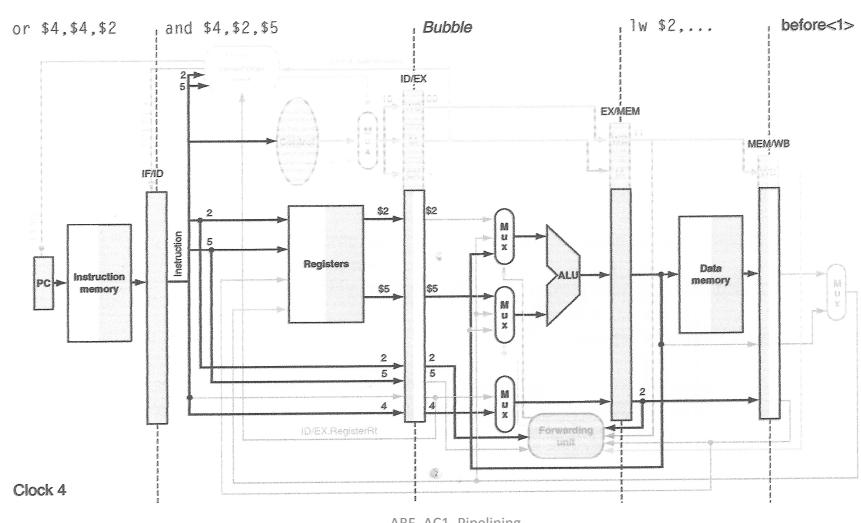
56

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



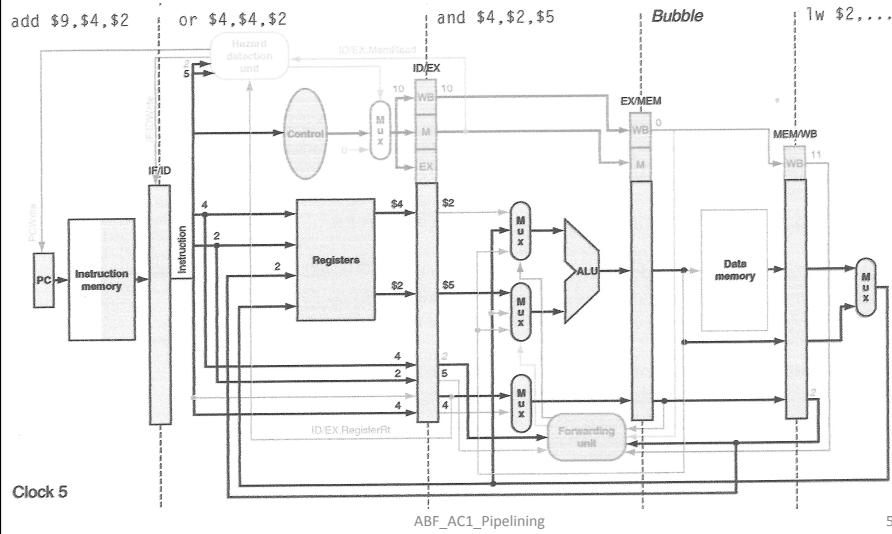
57

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

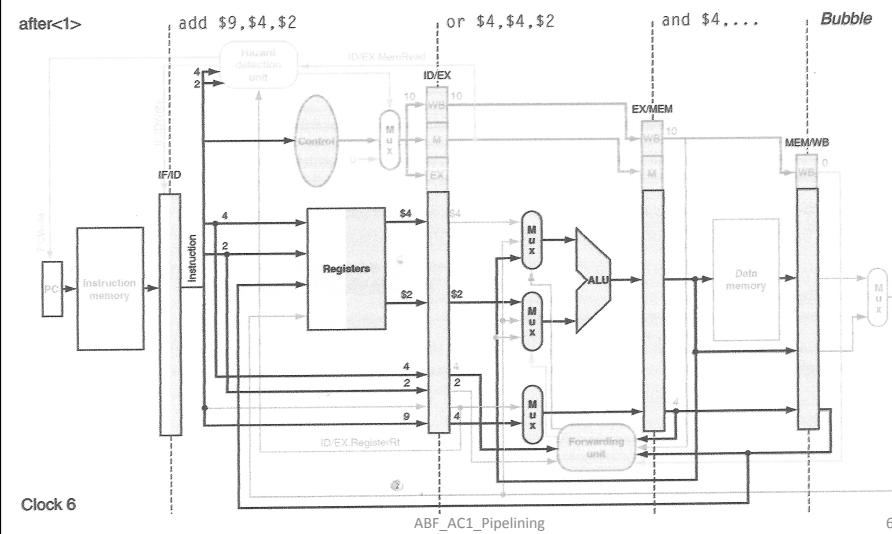


58

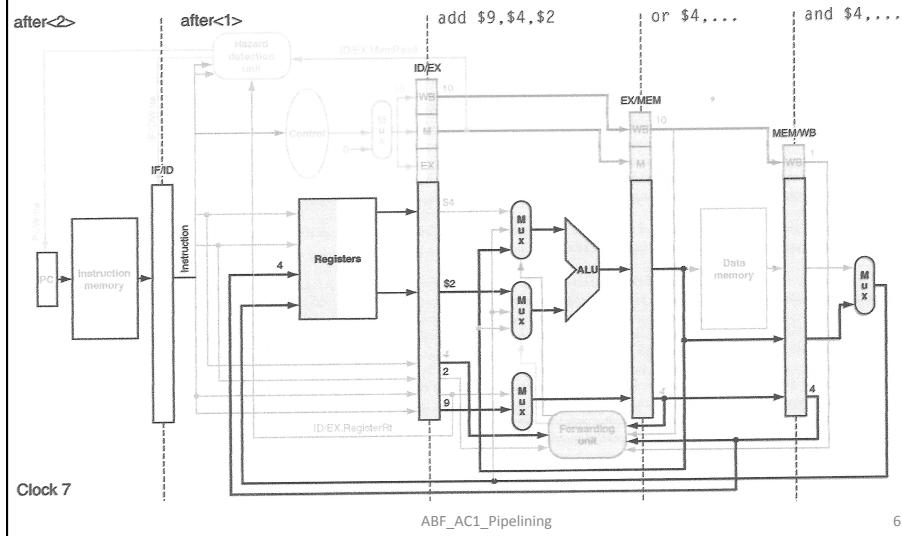
## Execução no pipeline com stalls e forward – 5º ciclo de relógio



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



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



61

## Control Hazards

## Control Hazards

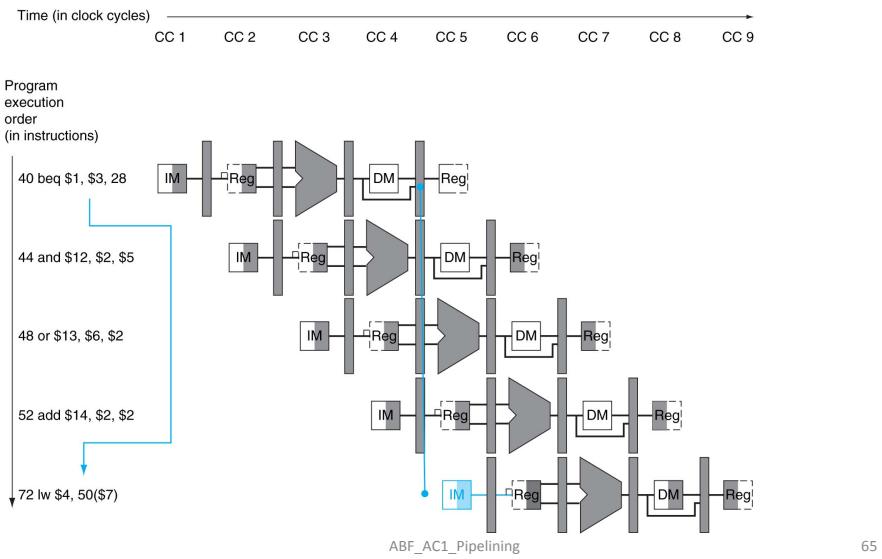
- Pipeline hazards introduzidos pelos branches
- Problema: determinar a próxima instrução de que fazer o *fetch*
- Como lidar com este tipo de hazards?
- Prever resultado dos branch (*branch prediction*)

## 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

## Branch Hazard



## Branch Prediction (2)

- *Branch 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.

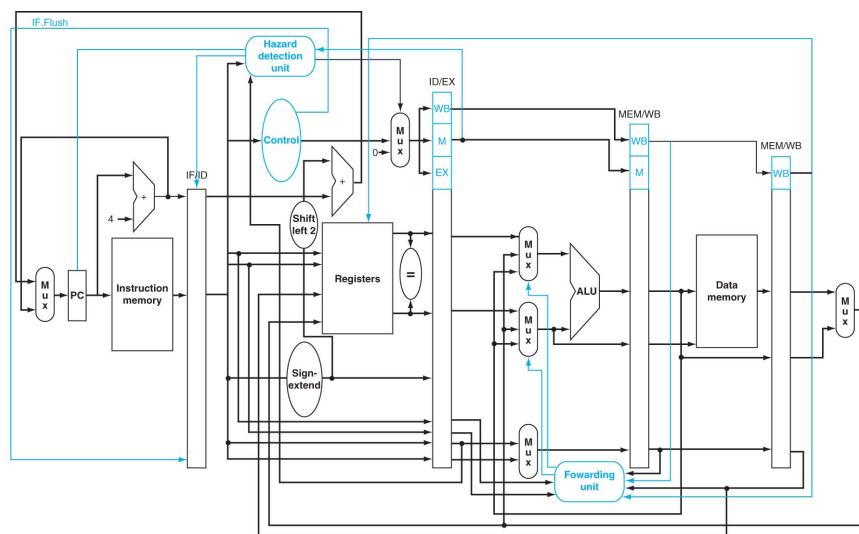
## Modificar Datapath

- 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
  2. Cálculo do endereço-alvo: o somador dedicado pode ser colocado no andar ID

ABF\_AC1\_Pipelining

67

## Datapath + Control: final



ABF\_AC1\_Pipelining

68

## Pipelining - sumário

- Fatores que facilitam pipelining
  - Todas as instruções do mesmo comprimento
  - Poucos formatos de instrução
  - Operands 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 e viu-se como tratar os hazards
- Por tratar:
  - exception handling
  - Técnicas usadas para aumentar o desempenho em processadores mais recentes: out-of-order execution, etc.

## Pipelining – sumário (cont.)

- 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)
  - Detetar e resolver hazards