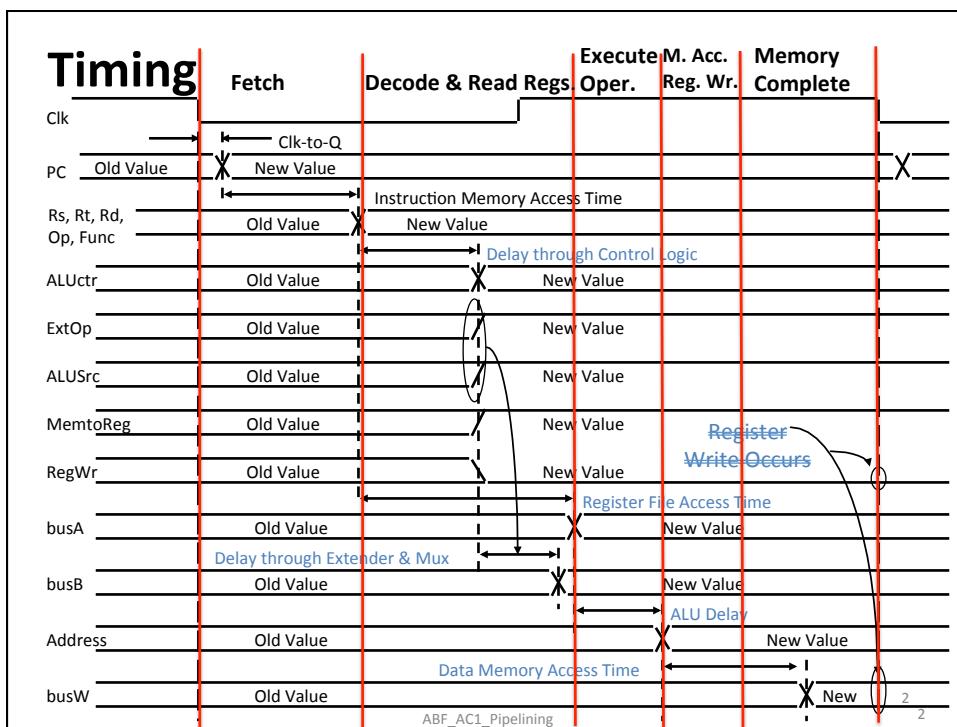


Pipelining

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

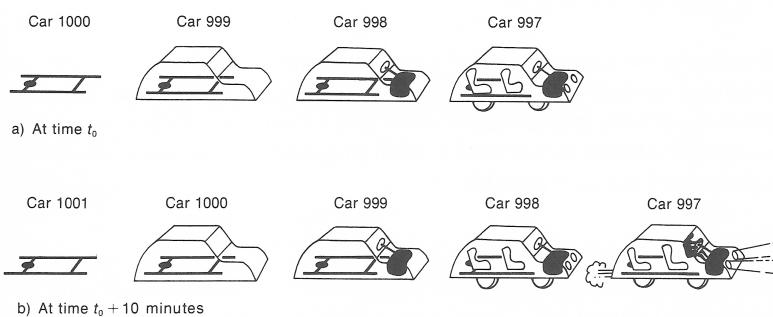
2. Pipelined:

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

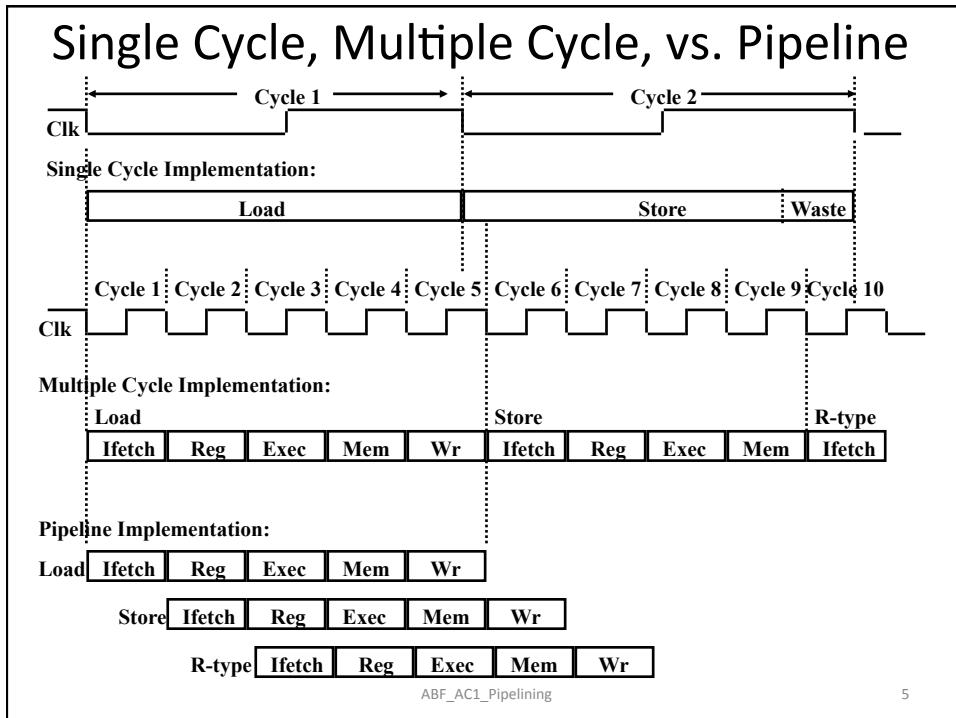


Montar um carro completo: 5×10 minutos = 50 minutos (**Latência**)
 Linha de montagem: um novo carro a cada 10 minutos (**Throughput**)

$$\text{Speedup}_{\text{pipeline/singlecycle}} = 5$$

ABF_AC1_Pipelining

4



Tempos com a tecnologia atual

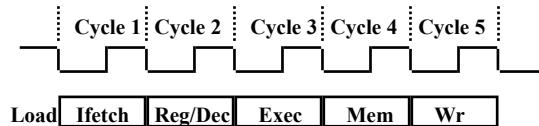
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, AND, OR, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps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

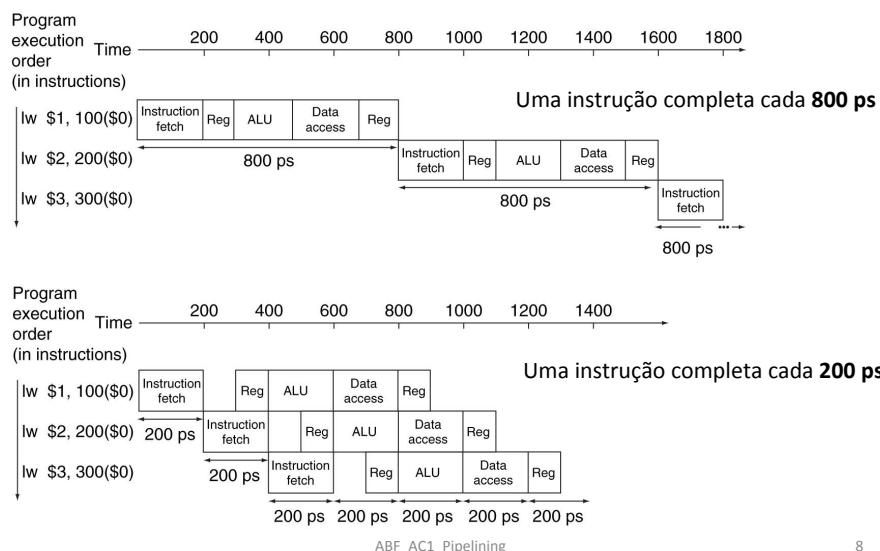


- Ifetch: Instruction Fetch
 - Fetch da instrução da Instruction Memory
- Reg/Dec: Registers Fetch e Instruction Decode
- Exec: Cálculo do endereço de memória
- Mem: Ler o dado da Data Memory
- Wr: Escrever o dado no banco de registros

ABF_AC1_Pipelining

7

Multicycle vs. Pipelined



ABF_AC1_Pipelining

4

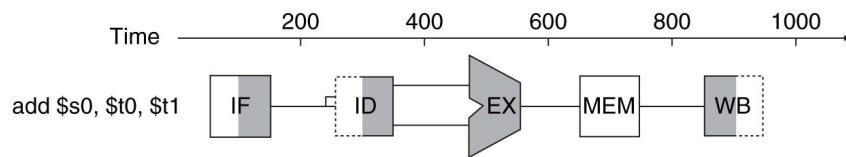
ISA e Pipelining - MIPS

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 15 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

Representação do Pipeline

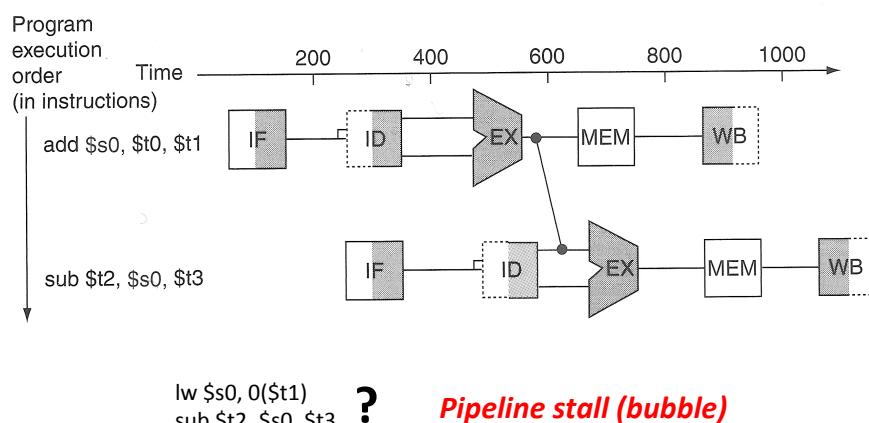


Sombreado – componente ativa
 Leitura dos registos ou da memória na 2^a parte do ciclo
 Escrita na 1^a parte do ciclo
 MEM stage – inativo (não há acesso à memória de dados)

ABF_AC1_Pipelining

11

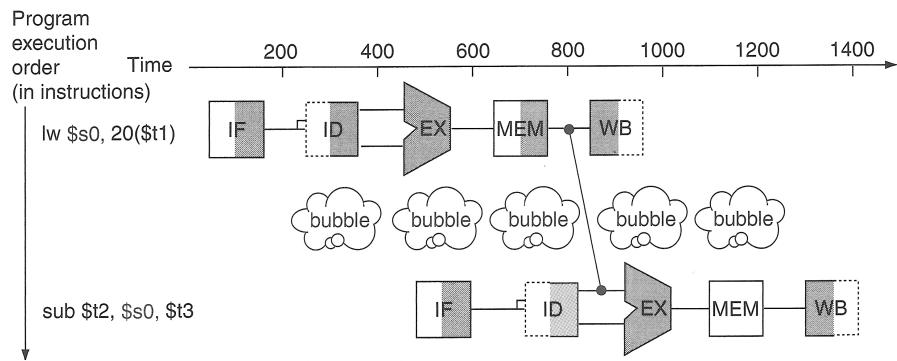
Data Hazzards Solução: *Forwarding*



ABF_AC1_Pipelining

12

Data Hazards (2)



Software: Reordenação do código

ABF_AC1_Pipelining

13

Reordenação do código

$a = b + e;$
 $c = b + f;$

lw	\$t1, 0(\$t0)
lw	\$t2, 4(\$t0)
add	\$t3, \$t1, \$t2
sw	\$t3, 12(\$t0)
lw	\$t4, 8(\$t0)
add	\$t5, \$t1, \$t4
sw	\$t5, 16(\$t0)

$a = b + e;$
 $c = b + f;$

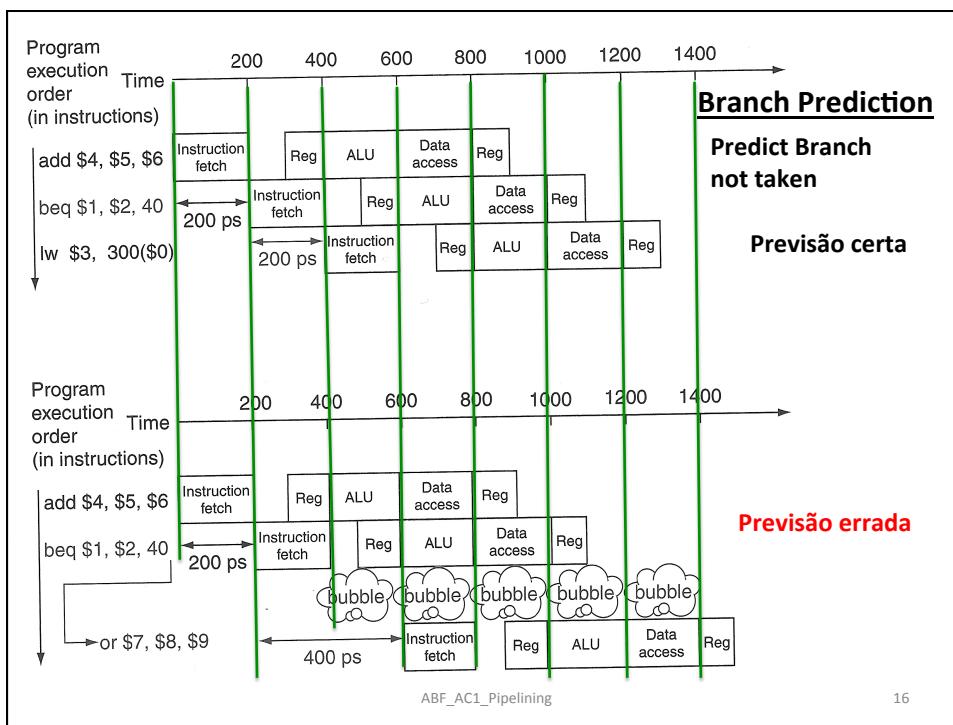
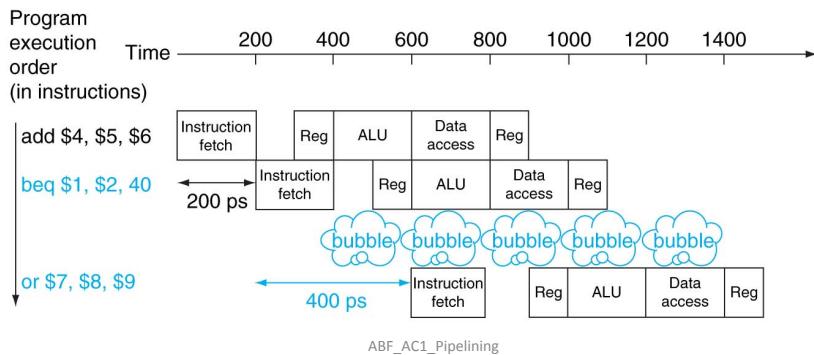
lw	\$t1, 0(\$t0)
lw	\$t2, 4(\$t0)
lw	\$t4, 8(\$t0)
add	\$t3, \$t1, \$t2
sw	\$t3, 12(\$t0)
add	\$t5, \$t1, \$t4
sw	\$t5, 16(\$t0)

ABF_AC1_Pipelining

14

Control Hazards

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



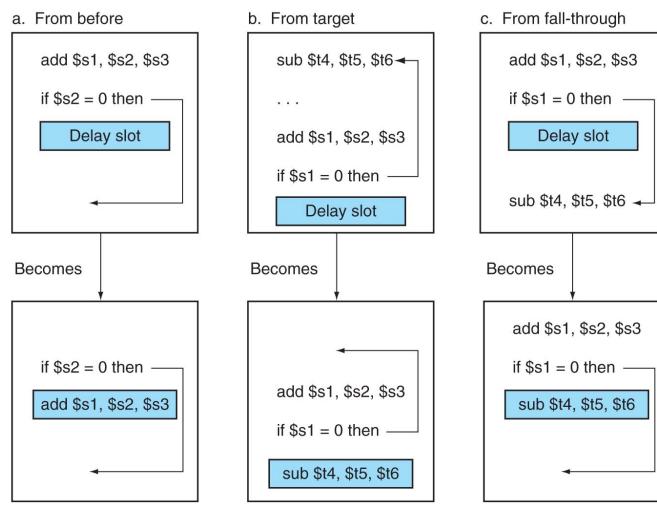
Branch Prediction

- Formas mais elaboradas:
 - Dynamic Branch Prediction – baseada no histórico do comportamento do branch
 - Consegue-se atingir 90% de previsões certas
- Alternativa: *delayed branch* (MIPS)
 - A instrução a seguir ao *branch* é sempre executada; o *branch* efetua-se com uma instrução de atraso – o *assembler* reordena as instruções de modo transparente ao utilizador

ABF_AC1_Pipelining

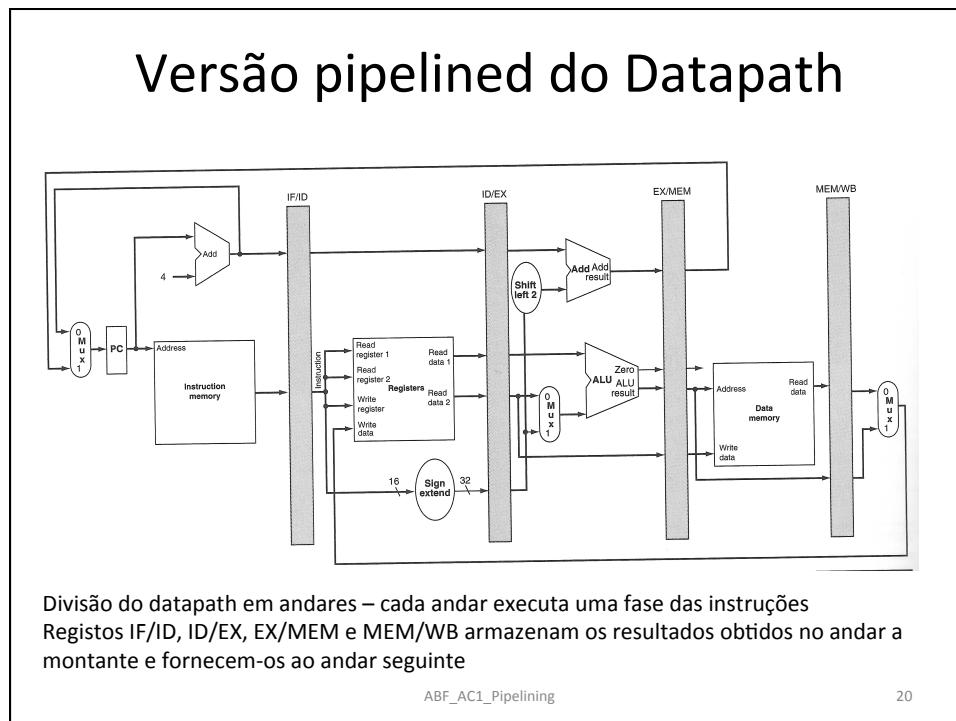
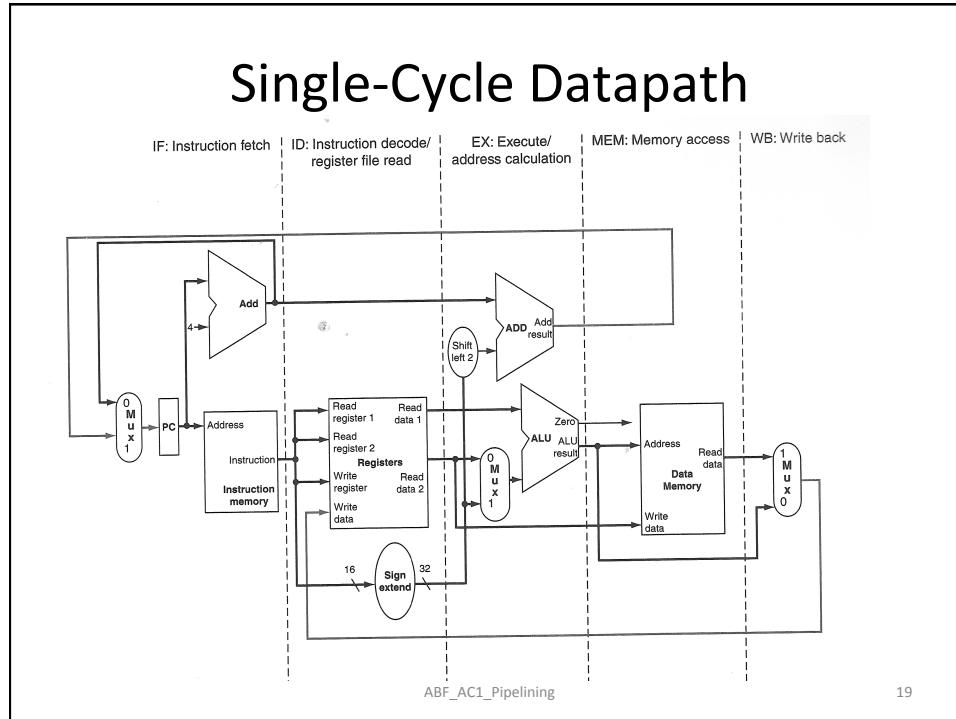
17

MIPS: *delayed branch*

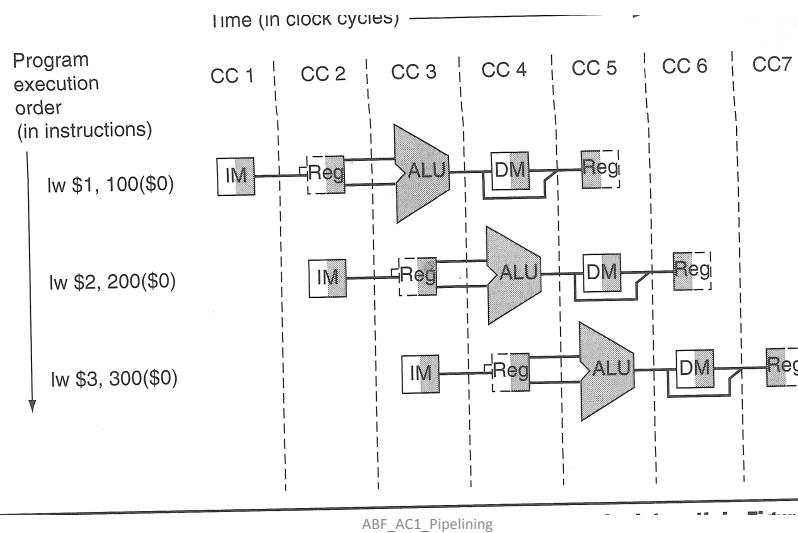


ABF_AC1_Pipelining

18

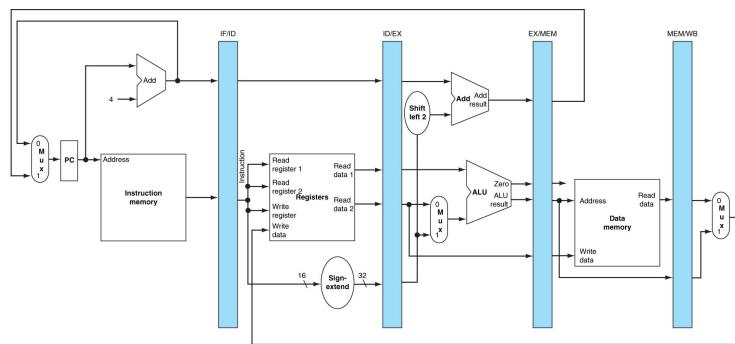


Execução supondo que cada instrução executa na sua cópia do datapath



1. Pipelined Datapath

Pipelined Datapath



Dados fluem da esquerda para a direita
(exceções: WB stage e modificação do conteúdo do PC nos branch)

ABF_AC1_Pipelining

23

Execução de lw

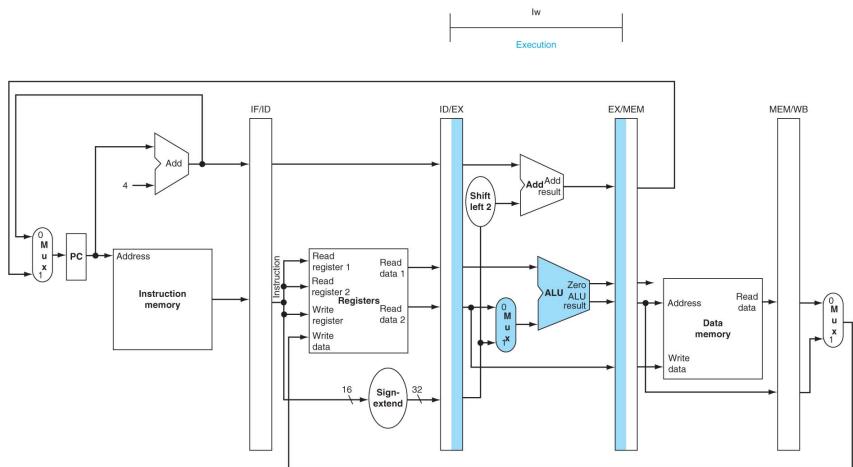
1ª fase:
Instruction Fetch

2ª fase:
Instruction Decode

ABF_AC1_Pipelining

24

IW – 3^a fase: Execute (cálculo do endereço de memória)

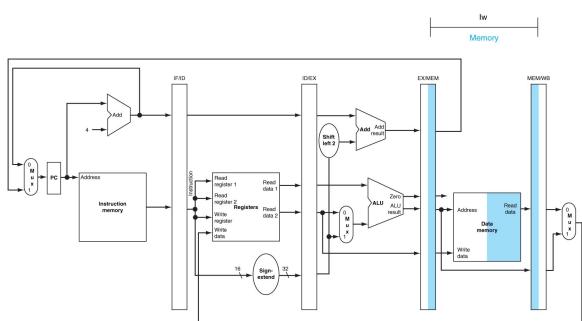


ABF_AC1_Pipelining

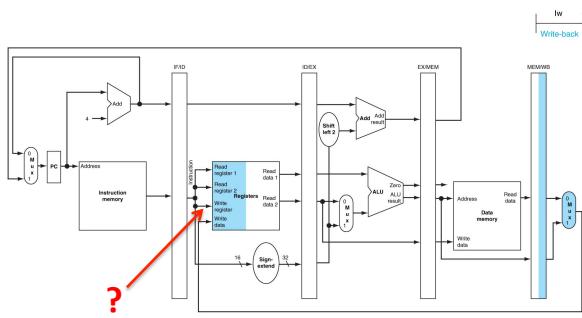
25

IW

4^a fase:
Memory Acess



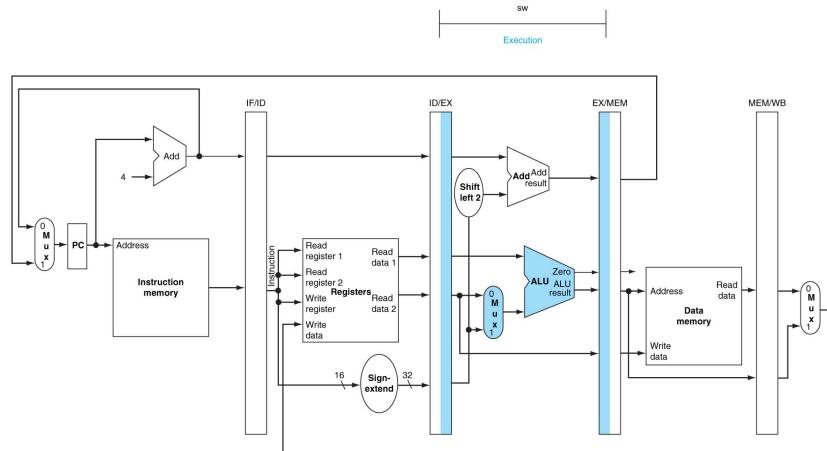
5^a fase:
Write-back



ABF_AC1_Pipelining

26

SW – 3ª fase: Execute (cálculo do endereço de memória)

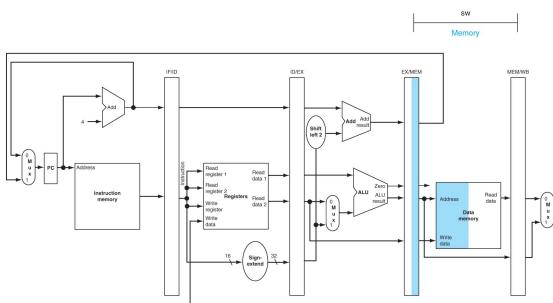


ABF_AC1_Pipeline

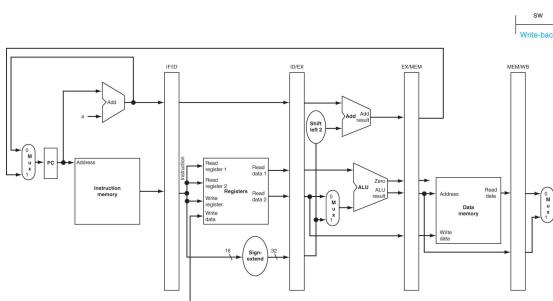
27

SW

4ª fase:
Memory Access



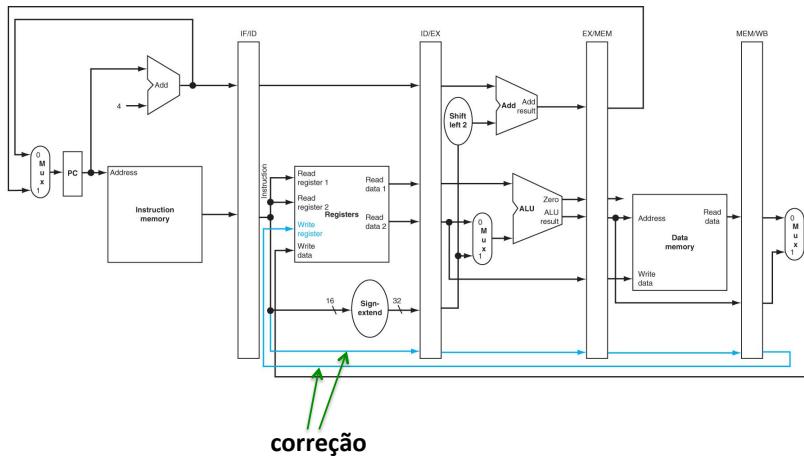
5ª fase:
Write-back
(nenhuma componente
do datapath ativa)



ABF_AC1_Pipeline

28

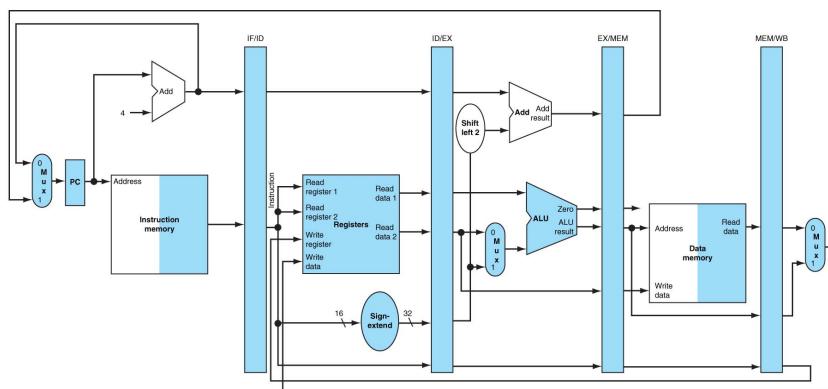
Datapath corrigido



ABF_AC1_Pipelining

29

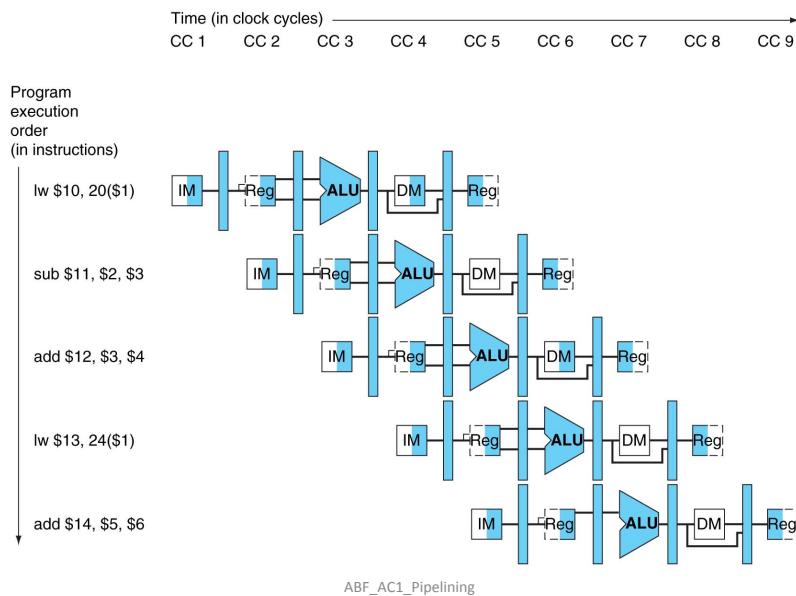
Parte do Datapath usada na execução de **Iw** (no conjunto das 5 fases)



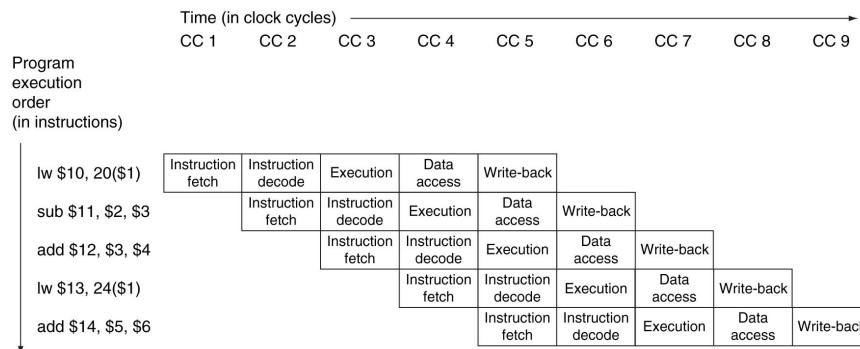
ABF_AC1_Pipelining

30

Pipeline: diagrama “*multiple-clock-cycle*”



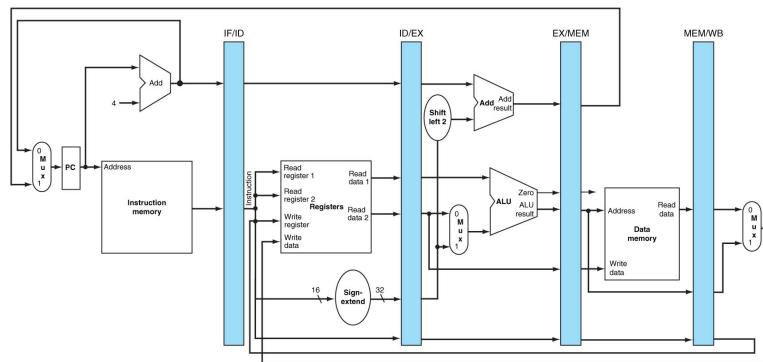
Pipeline: representação “*multiple-clock-cycle*” tradicional



Pipeline: representação “single-clock-cycle”

add \$14, \$5, \$6 lw \$13, 24 (\$1) add \$12, \$3, \$4 sub \$11, \$2, \$3 lw \$10, 20(\$1)

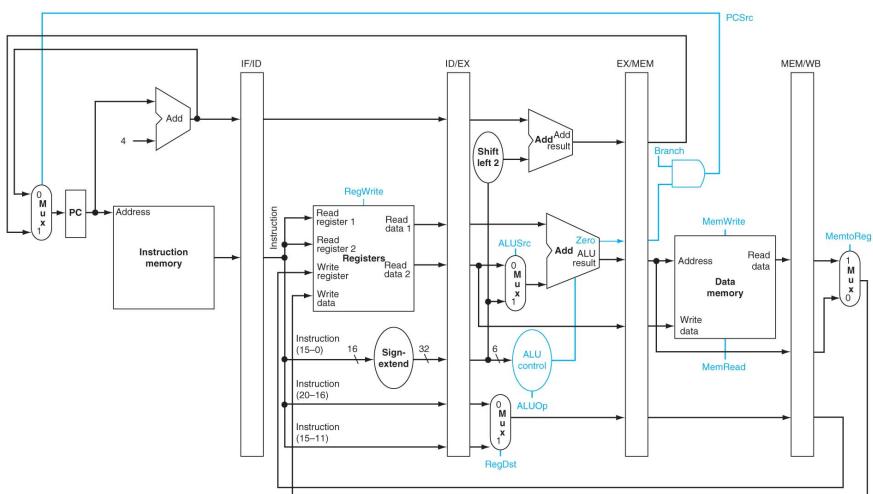
Instruction fetch Instruction decode Execution Memory Write-back



ABF_AC1_Pipelining

33

Datapath com os sinais de controle identificados



ABF_AC1_Pipelining

34

Qual a informação a guardar nos registos inter-andares?

- IF/ID: Instruction Code (32-bits)
Program Counter (32-bits)
- ID/EX:Instruction Code Fields Rt, Rd e Func (16-bits)
Program Counter (32-bits)
Rs (32-bits), Rt (32-bits)
Instruction[15-0] sign-extended (32-bits)
- EX/MEM: Instruction Code Field Rd/Rt (5-bits)
ALUResult (32-bits)
Rt (32-bits)