

Introdução à Arquitetura de Computadores

Auto25

μArquitetura MIPS Multicycle: I

Performance Single-cycle

Caminho Crítico de Execução:

Período Mínimo de Clock

Tempo de Execução

μArquitetura Multicycle (MC)

Limitações do Single-cycle

Multicycle versus Single-cycle

Von Neumann: Memória única

Reutilização da ALU

Datapath Multicycle

Elementos de Estado

Execução de instruções:

Acesso à Memória: *lw* e *sw*

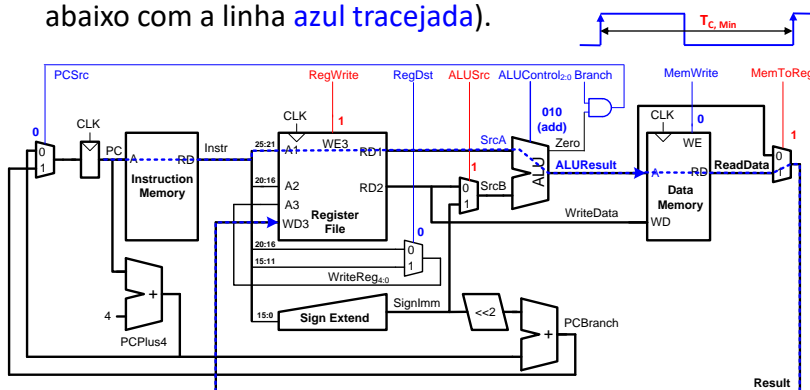
tipo-R (*add*) e *branch* (*beq*)

A. Nunes da Cruz / DETI - UA

Junho / 2021

Performance SC (1) - Caminho Crítico e $T_{C,Min}$ (1) - *lw*

- Num processador *Single-cycle* todas as instruções executam num único ciclo de *clock*.
- O período mínimo do *clock* ($T_{C,Min}$) é limitado pelo caminho crítico de execução da instrução *lw* (ilustrado no diagrama abaixo com a linha azul tracejada).



$T_{C,Min}$ limitado pelo caminho crítico (*lw*)

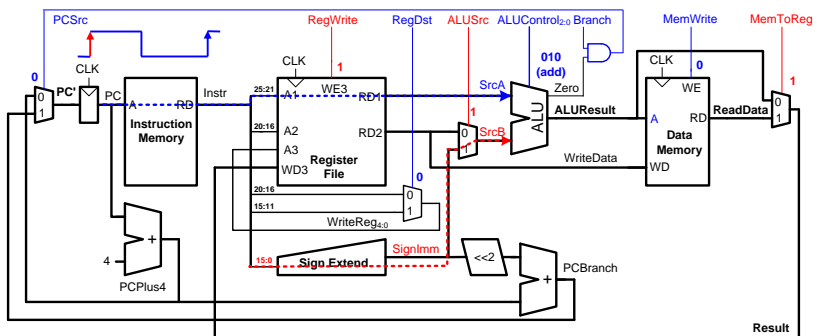
© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

1/26

Performance SC (2) - Cam. Crítico: T_{C1} (2) - Fetch+Decode

1. No flanco **ascendente** do **clock**, **PC** carrega um novo endereço.
2. A instrução é lida da Memória de Instruções.
3. O operando **SrcA** é lido do Banco de Registos e **em paralelo** o valor **imediato** é estendido em sinal e selecionado, pelo multiplexer **ALUSrc**, como **SrcB** da ALU.



T_{C1} : CLK to InstrMem to RegisterFile to ALU

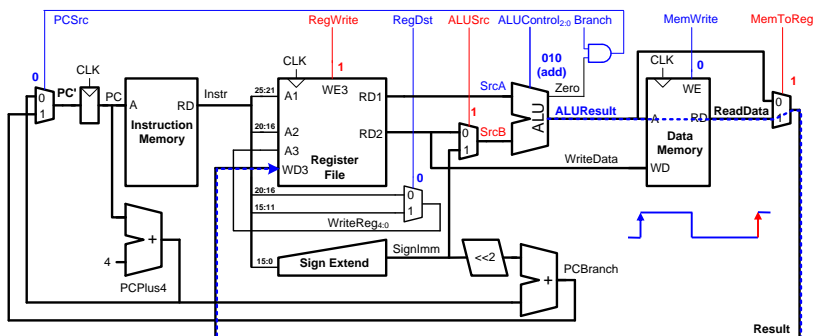
© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

2/26

Performance SC (3) - Cam. Crítico: T_{C2} (3) - Exec+Data+Wb

4. A ALU soma **SrcA** com **SrcB** e calcula o endereço (**ALUResult**).
5. Deste endereço é lido **ReadData** da Memória de Dados.
6. O multiplexer **MemToReg** seleciona **ReadData**.
7. O **Result** apresenta-se à entrada do RF, respeitando o tempo de **setup**, para ser escrito **no próx. flanco ascendente** do **clock**.



T_{C2} : ALU to DataMem to Mux to RegisterFile

$T_{C1} + T_{C2} = T_{C,Min} \rightarrow$

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

3/26

Performance SC (4) - Caminho Crítico e $T_{C,Min}$

- Período Mínimo de Clock, $T_{C,Min}$:

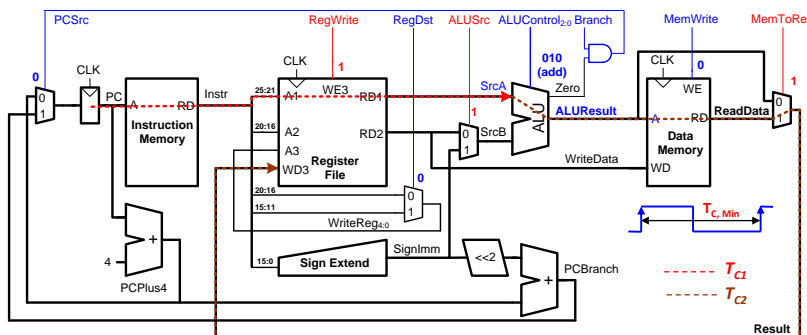
$$T_{C,Min} = t_{pcq_PC} + t_{mem} + \overbrace{\max(t_{Rfread}, t_{sext} + t_{mux})}^{T_{C1}} + t_{ALU} + t_{mem} + \overbrace{t_{mux} + t_{RFsetup}}^{T_{C2}}$$

Em geral, os limites são impostos pelas Memórias, Banco de Registos e ALU; assim, a expressão anterior pode ainda ser simplificada:

$$T_{C,Min} = t_{pcq_PC} + 2t_{mem} + t_{Rfread} + t_{ALU} + t_{mux} + t_{RFsetup}$$

→ Para uma instrução Tipo R o tempo era mais pequeno

Existem instruções mais rápidas e outras mais lentas

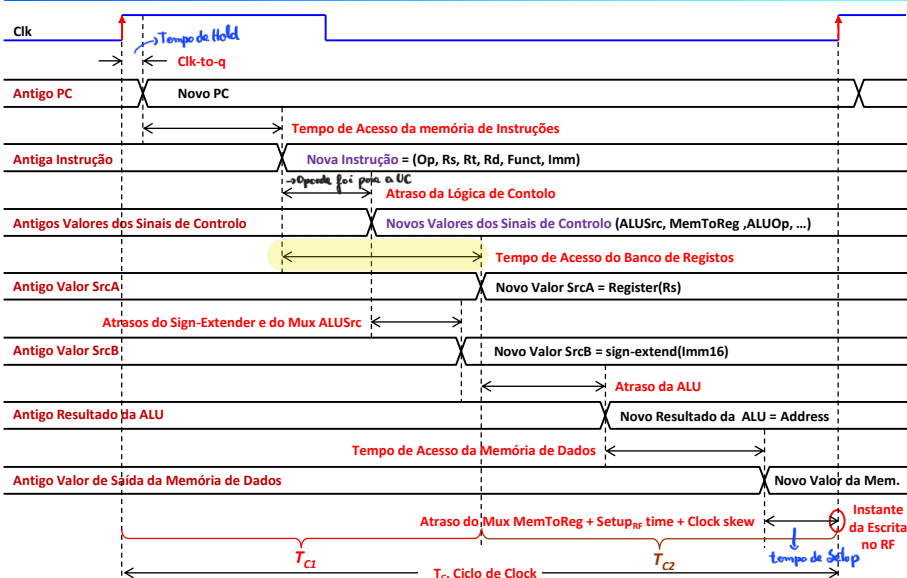


© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

4/26

Performance SC (5) - Caminho Crítico e $T_C(5)$: $1w$ timing



© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

5/26

Performance SC (6) - Tempo de Execução (1) - Definição

Tempo de Execução de um Programa

$$\begin{aligned}\text{Tempo de Execução} &= \# \text{Instruções} \times (\text{Ciclos/Instrução}) \times (\text{Segundos/Ciclo}) \\ &= \# \text{Instruções} \times \text{CPI} \times T_c\end{aligned}$$

$$T_{\text{EXEC}} = \frac{\text{Segundos}}{\text{Programa}} = \frac{\# \text{Instruções}}{\text{Programa}} \times \frac{\text{Ciclos}}{\text{Instrução}} \times \frac{\text{Segundos}}{\text{Ciclo}}$$

- #Instruções/Programa: Número de Instruções para completar a tarefa
- Ciclos/Instrução: **CPI**: Ciclos/Instrução (=1 no Single-cycle!)
- Segundos/Ciclo: **T_c**, Período de Clock (1/Frequência)
- Segundos/Programa : **T_{EXEC}**: Tempo de espera para executar a tarefa

Performance SC (6) - Tempo de Execução (2) - Exemplo

Elemento	Parâmetro	Atraso (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	t_{RFsetup}	20

$$\begin{aligned}T_{c,\text{Min}} &= t_{pcq_PC} + 2t_{\text{mem}} + t_{\text{RFread}} + t_{\text{ALU}} + t_{\text{mux}} + t_{\text{RFsetup}} \\ &= [30 + 2(250) + 150 + 200 + 25 + 20] \text{ ps} \\ &= 925 \text{ ps } (F_{c,\text{Max}} = 1.08 \text{ GHz})\end{aligned}$$

Em geral, estes atrasos são dependentes da tecnologia usada (e.g., CMOS, BiCMOS) para implementar a lógica.

Performance SC (7) - Tempo de Execução (3) - Exemplo

Qual o tempo de execução dum programa com 100 mil milhões de instruções, num CPU *Single-cycle*?

$$\begin{aligned}
 \text{Tempo de Execução} &= \overset{100 \text{ mil milhões}}{\# \text{Instruções}} \times \text{CPI} \times T_c \\
 &= 0.1 \times 10^{12} \times (1) \times 925 \times 10^{-12} \text{ s} \\
 &= 92.5 \text{ segundos}
 \end{aligned}$$

$$\begin{aligned}
 \text{CPI} &= 1 \\
 T_c &= 925 \text{ ps}
 \end{aligned}$$

Multicycle MIPS (1) - Limitações Single-cycle

Basicamente, **três** pontos fracos:

1. Requer um período de *clock* suficientemente longo, para acomodar a instrução com o maior tempo de execução (*lw*), não obstante a maioria das instruções ter um tempo de execução mais curto.

2. Usa dois somadores e uma ALU.

Específicos para calcular: $\rightarrow (PC+4)$
 $\rightarrow BTA$

3. Usa memórias separadas para instruções e dados (i.e., a arquitetura de processador é do tipo *Harvard*).

Hoje em dia, a maioria dos computadores usa uma memória única para instruções e dados (i.e., arquitetura *Von Neumann*).

Single cycle:

$$t_{pcq-PC} + t_{mem} + t_{RF} + t_{ALU} + t_{mem} + t_{mux} + t_{RFSetup}$$

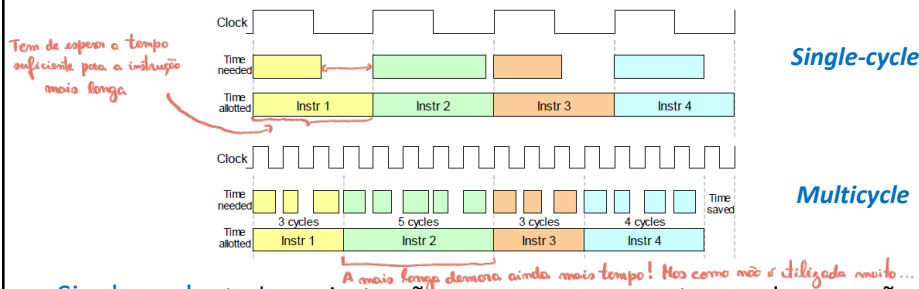
ou

$$t_{SE} + t_{mux}$$

Multi cycle: Separámos a execução!

Multicycle MIPS (2) - SC vs MC: Modo de Execução

Ideia: Dividir a execução em múltiplos ciclos de clock



- **Single-cycle:** todas as instruções ocupam o mesmo tempo de execução.
- **Multicycle:** as instruções ocupam um tempo de execução variável (no diagrama é usado um clock 4 vezes mais rápido). Consegue-se, deste modo, uma maior taxa-média de execução de instruções-por-segundo (ou throughput).

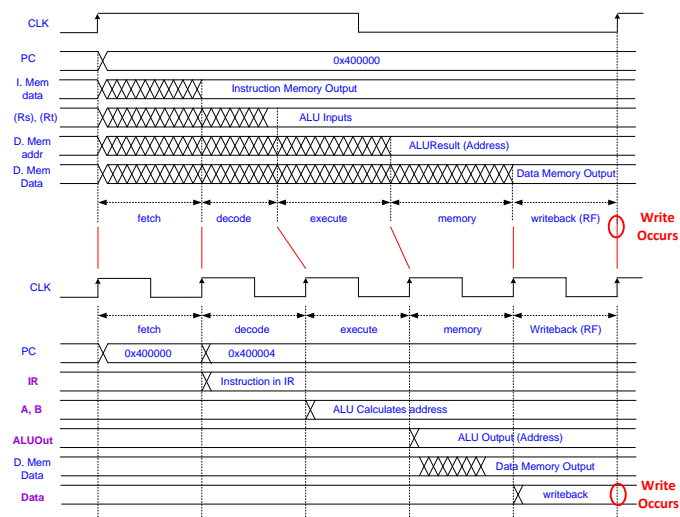
Admitamos que é possível dividir a execução (SC) em múltiplas fases (max. 5):
fetch, decode, operação na ALU, acesso à memória (dados) e escrita no RF... -->

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

10/26

Multicycle MIPS (3) - SC vs MC: 1w timing



1w timing:
 num CPU Singlecycle

Substituiu-se o longo ciclo de CLK por 5 mais curtos.

1w timing:
 num CPU Multicycle

IR, Data, A, B, e ALUOut são registos auxiliares.

Esta decomposição em ciclos mais curtos, vai permitir a execução mais rápida de instruções que não exijam as 5 fases; Exemplos: *beq* só precisa de 3 ciclos e as instruções de tipo-R só de 4 ciclos.

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

11/26

Multicycle MIPS (4) - μ Arquitecturas SC vs MC

• Single-cycle

- + simples
- tempo de ciclo limitado pela instrução mais longa (**lw**)
- usa uma ALU, dois somadores e duas memórias → Mais hardware
- + FSM mais simples (combinatória)

• Multicycle

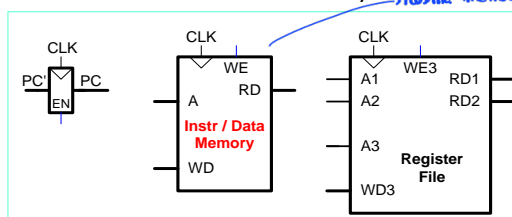
- + frequência de *clock* mais elevada
- + as instruções mais simples executam mais rápido → Principal vantagem!
- + reutilização de *hardware* (caro) em ciclos distintos:
 - usa uma única Memória e uma única ALU
- Unidade de Controlo mais complexa:
 - máquina de estados (FSM)
 - ↳ Possui a ser sequencial
 - ↳ cálculo do BTA e/ou (PC+4)

• Fases de design iguais: *datapath* + *control*

MC Datapath (1) - Elementos de Estado - Uma só Memória

Substituição das duas memórias de Instruções e Dados por uma única*

- Não era possível na implementação *Single-cycle* (As instruções e os dados são necessários durante o mesmo ciclo de clock).
- Agora, a **mesma** memória pode ser (re)usada se necessário (O *fetch* da Instrução e o acesso aos Dados são feitos em **ciclos** de clock **distintos**). → Mesma memória para instruções e dados



*A arquitetura de Von Neumann substitui a arquitetura de Harvard.

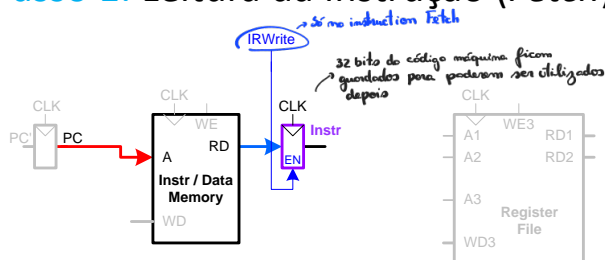
$$\begin{array}{c}
 \text{Instruction Fetch} \\
 t_{PCQ-PC} + t_{mem} + t_{RF} + t_{ALU} + t_{mem} + t_{mux} + t_{RFSetup} \\
 \text{ou} \\
 t_{SE} + t_{mux}
 \end{array}$$

1.º ciclo

Nota: calculamos o (PC+4) também. Iremos ver depois...

MC Datapath (2) - 1w Fetch - Registo Instr

Passo 1: Leitura da Instrução (Fetch)



1. Usa-se um **novo** registo de **Instrução*** para reter a instrução, por forma a poder **ser usada nos ciclos seguintes**.

*Instr - É um registo **não acessível** ao programador, designado por '**non-architectural**' (contrariamente ao que acontece com os registos do RF e mesmo com o PC). Este tipo de registos são colocados nas saídas dos elementos funcionais **para possibilitar** o acesso aos valores (endereços/dados) do respectivo elemento **nos ciclos seguintes**.

© A. Nunes da Cruz

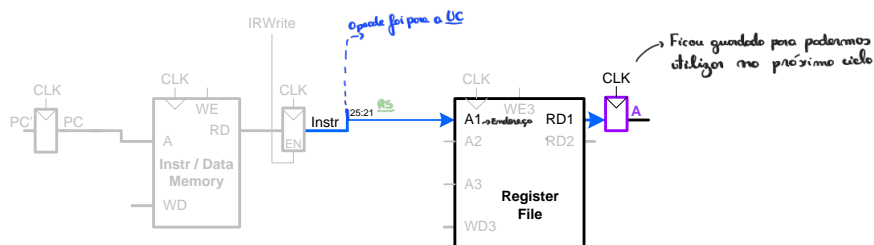
IAC - MIPS - Multicycle: Datapath

14/26

2.º ciclo

MC Datapath (3) - 1w Leitura do Operando - Registo A

Passo 2: Leitura do operando do Register File (RF)



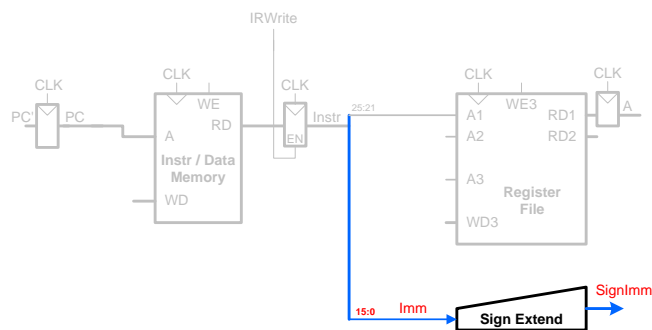
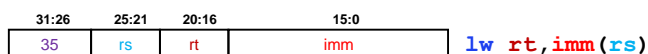
2. Insere-se mais um registo adicional (**A**), para reter o valor de **rs** (vai ser necessário no ciclo seguinte).

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

15/26

2.º ciclo

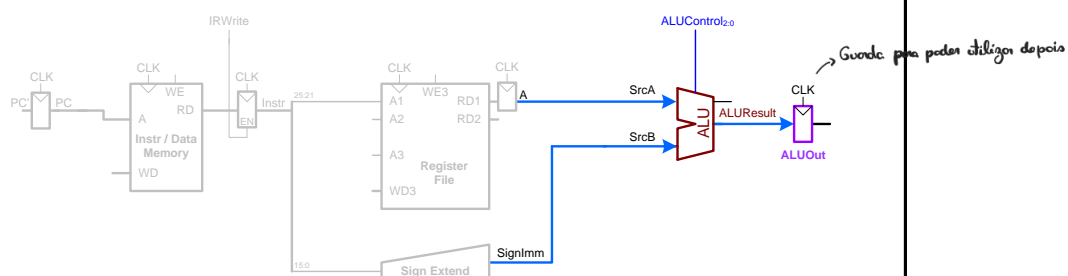
MC Datapath (4) - lw Obtenção do Offset de Endereço**Passo 3: Extensão do sinal do valor Imediato****3. Converte-se o valor Imm_{16} em $SignImm_{32}$** 

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

16/26

3.º ciclo

MC Datapath (5) - lw Cálculo do Endereço - Reg. ALUOut**Passo 4: Cálculo do endereço de memória**

4. Na ALU soma-se o endereço base $SrcA$ ao $SrcB$, para obter o endereço de memória em $ALUResult$ (Endereço = $A + SignImm$).

➤ Insere-se um novo registo* ($ALUOut$) na saída da ALU.

*O registo $ALUOut$ é necessário devido à partilha da ALU para várias funções. Dependendo da instrução, o seu valor pode ser enviado ou para a memória (lw/sw) ou para o RF ($tipo-R$).

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

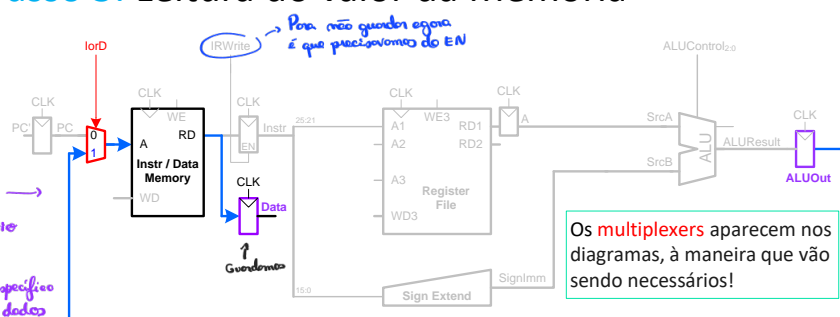
17/26

MC Datapath (6) - 1w Ler Valor da Memória - Reg. Data

Passo 5: Leitura do valor da Memória

4º ciclo

Como após os dados e as instruções estão no mesmo xiler
↓
Não temos um bloco específico para os escrever os dados



5. Insere-se um **multiplexer** em frente da entrada de endereço da memória (A), para selecionar o **PC** ou **ALUOut**.

- O **Fetch (I)** ou Acesso a **Dados (D)** é controlado pelo **novo** sinal (**lorD**).
- O valor lido (**RD**) é colocado **noutro** registo (**Data**).

Normalmente, a memória está segmentada: zona de instruções e zona de dados, etc.

© A. Nunes da Cruz

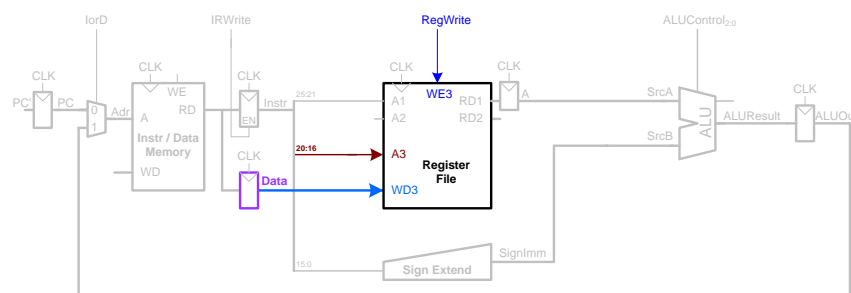
IAC - MIPS - Multicycle: Datapath

18/26

MC Datapath (7) - 1w Escrever valor no RF

Passo 6: Escrita no Banco de Registos

5º ciclo



6. O valor lido da memória (**Data**) é escrito (**RegWrite=1**) no registo **rt** (**Instr_{20:16}**) do Banco de Registos.

31:26	25:21	20:16	15:0	
35	rs	rt	imm	lw rt,imm(rs)

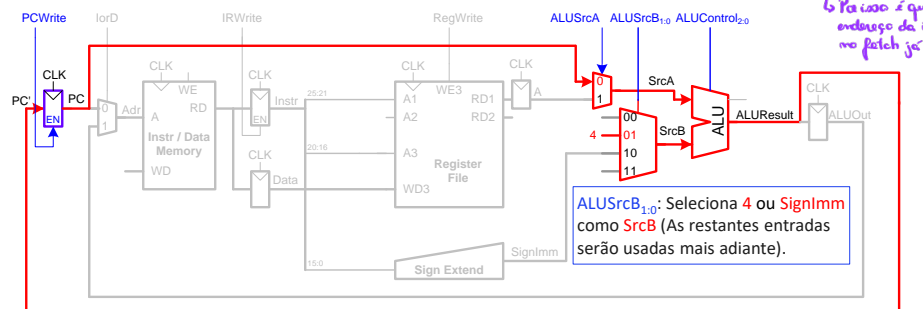
© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

19/26

MC Datapath (8) - **lw** - Cálculo de PC'

Passo 7: Cálculo do PC'

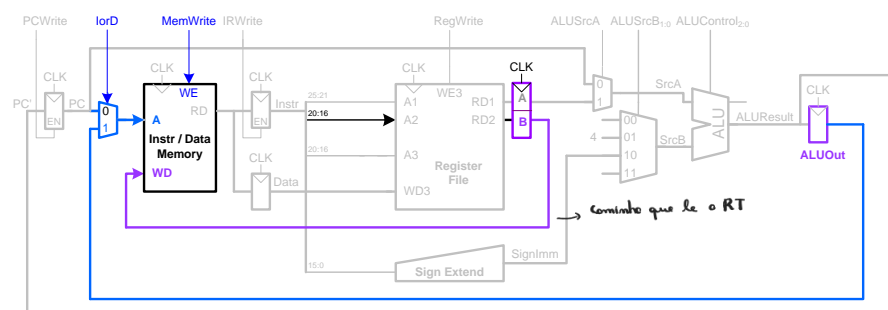


7. PC' = PC + 4. A soma é feita usando a mesma ALU (já usada no cálculo do endereço), mas em ciclos distintos.

- O novo sinal de controlo PCWrite (Enable do registo PC) permite a atualização do PC só em determinados ciclos de clock.

MC Datapath (9) - **sw** versus **lw** - Registo B

Passo 5: Escrita do valor de rt (B) na Memória



5. Comparando **sw** com **lw**, e neste mesmo ciclo de clock:

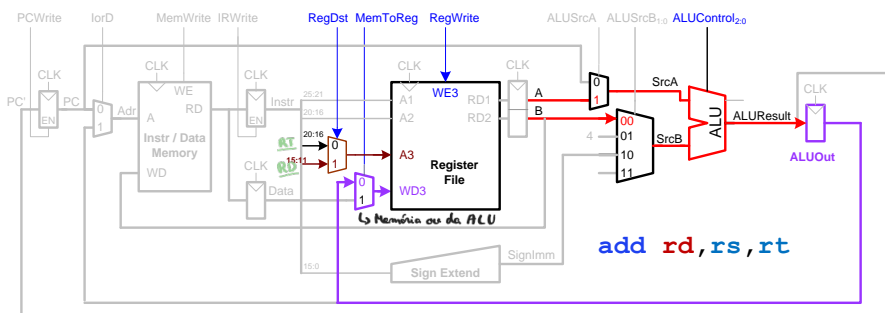
- O valor de **rt** está no novo registo B (também foi lido no Passo 2)
- Escreve B na memória fazendo MemWrite = 1 com A = ALUOut

31:26	25:21	20:16	15:0	
43	rs	rt	imm	sw rt, imm(rs)

$$\underbrace{t_{PCQ-PC} + t_{mem}}_{\text{Instruction Fetch}} + \underbrace{t_{RF}}_{\text{Instruction Decode}} + \underbrace{t_{ALU}}_{\text{Execute}} + \underbrace{t_{mem}}_{\text{Memory}} + t_{mux} + \cancel{t_{RF} \text{ setup}} \quad \text{or} \quad t_{SE} + t_{mux}$$

MC Datapath (10) - tipo-R

- Usa os registos **rs** e **rt** como operandos



1. Executa a operação aritmética/lógica (resultado em **ALUOut**)
2. Escreve o resultado no registo **rd** (RF), **RegDst=1**.
 - O multiplexer **MemToReg** está, agora, em frente de **WD3** do RF.

© A. Nunes da Cruz

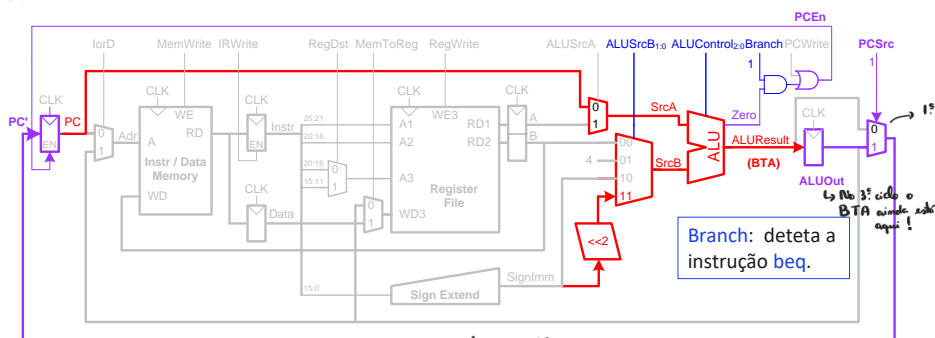
IAC - MIPS - Multicycle: Datapath

22/26

MC Datapath (11) - branch

1. (PC + 4)

2. Calcula o endereço-alvo: $BTA = (SignImm_{32} \ll 2) + (PC+4)$



2. Compara o valor dos registos **rs** e **rt** (**Zero**=1 se igual) e faz **PCSrc**=1 (A comparação reutiliza a ALU num ciclo posterior; **não está visível** no diagrama).
- O PC tanto pode ser atualizado (**PCEn**) por **PCWrite** ou por **Branch**.
 - O **novo** multiplexer **PCSrc** escolhe o valor de **PC'**.

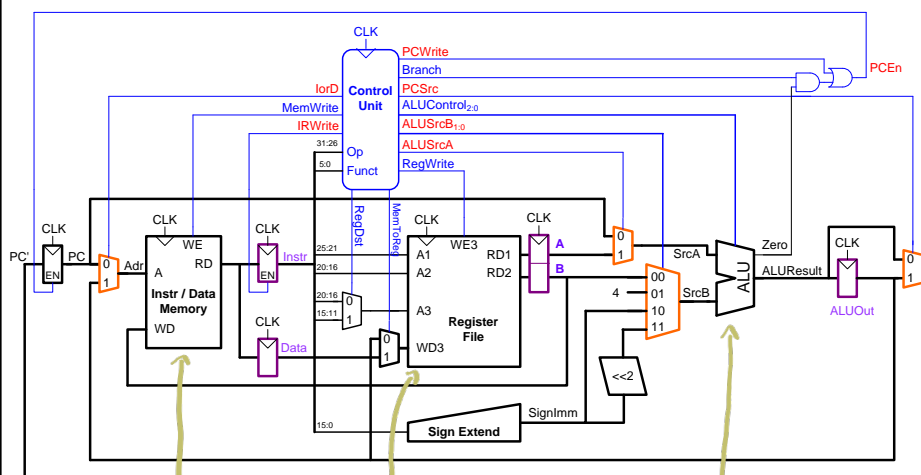
As duas operações (BTA e comparação de r_s e r_t) são feitas em ciclos diferentes! Ver FSM (beq).

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

23/26

MC Datapath (12) - Completo* *lw, sw, Tipo R, beq*



↳ Só existem 3 grandes blocos!

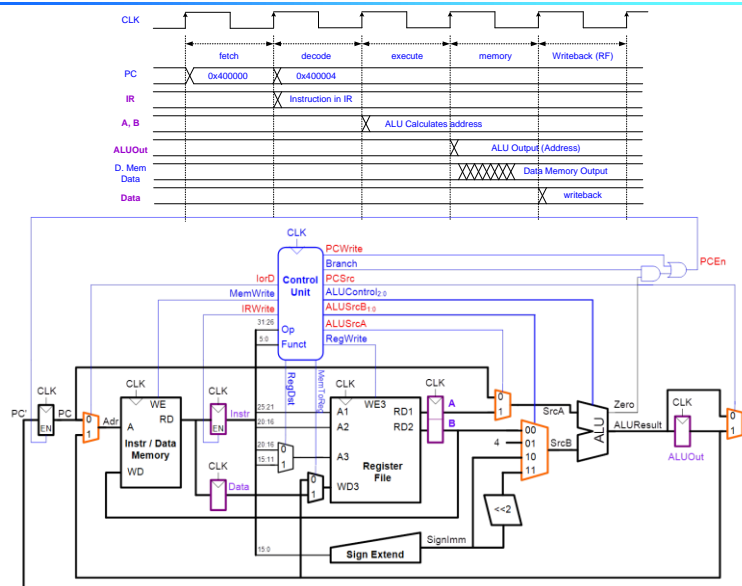
*Mas com um ISA limitado a: lw, sw, tipo-R e beq.

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

24/26

MC Datapath (13) - Completo - com *lw* timing



© A. Nunes da Cruz

IAC - MIPS - Multicycle: Datapath

25/26

A seguir: Unidade de Controlo MC: FSM

