Aula25

µArquitetura MIPS Multicycle: I

Performance Single-cycle

- Caminho Crítico de Execução:
  - Período Mínimo de Clock
- Tempo de Execução

µArquitetura Multicycle (MC)

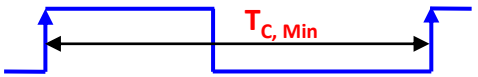
- Limitações do *Single-cycle*
- *Multicycle* versus *Single-cycle*
  - *Von Neumann: Memória única*
  - *Reutilização da ALU*

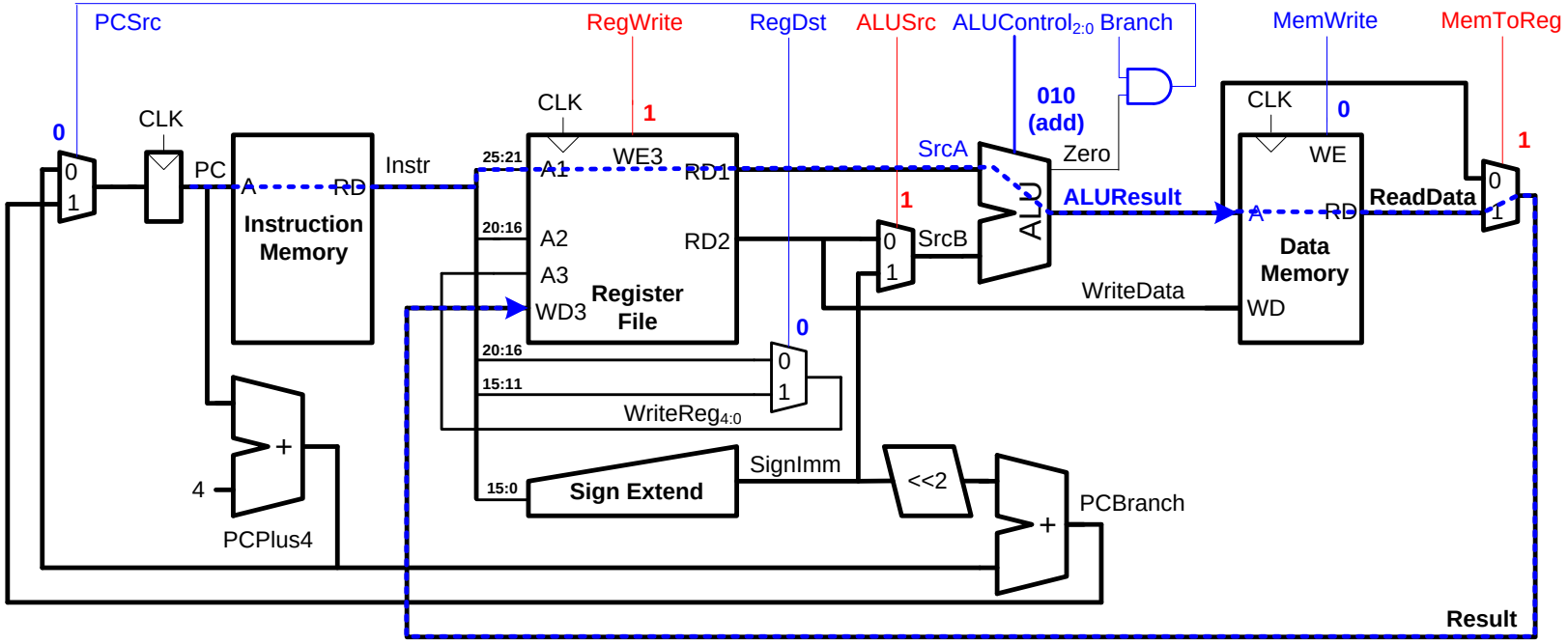
Datapath MC

- Elementos de Estado
- Execução de instruções:
  - Acesso à Memória: lw e sw
  - tipo-R (add) e *branch* (beq)

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).

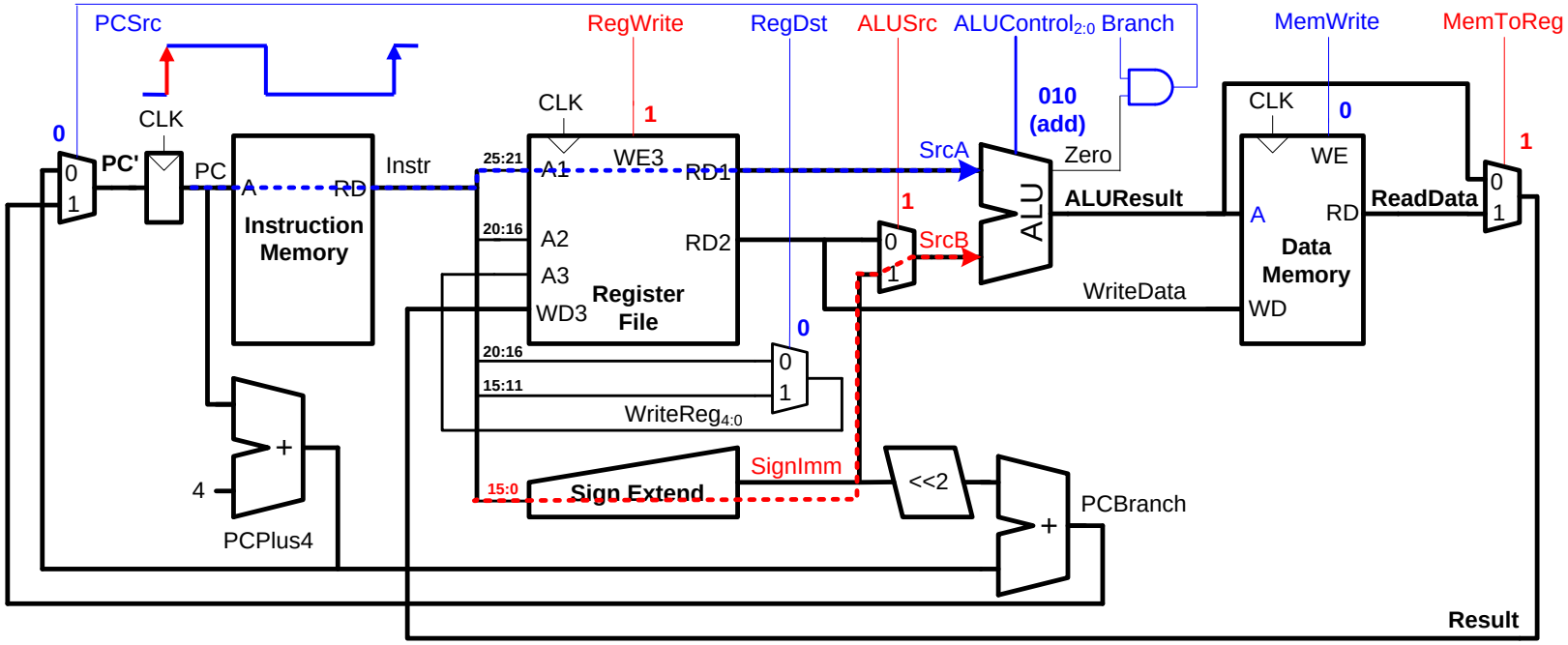




$T_{C,Min}$ limitado pelo caminho crítico (*lw*)

Performance SC (2) - Cam. Crítico e T_C (2): Fetch+Decode

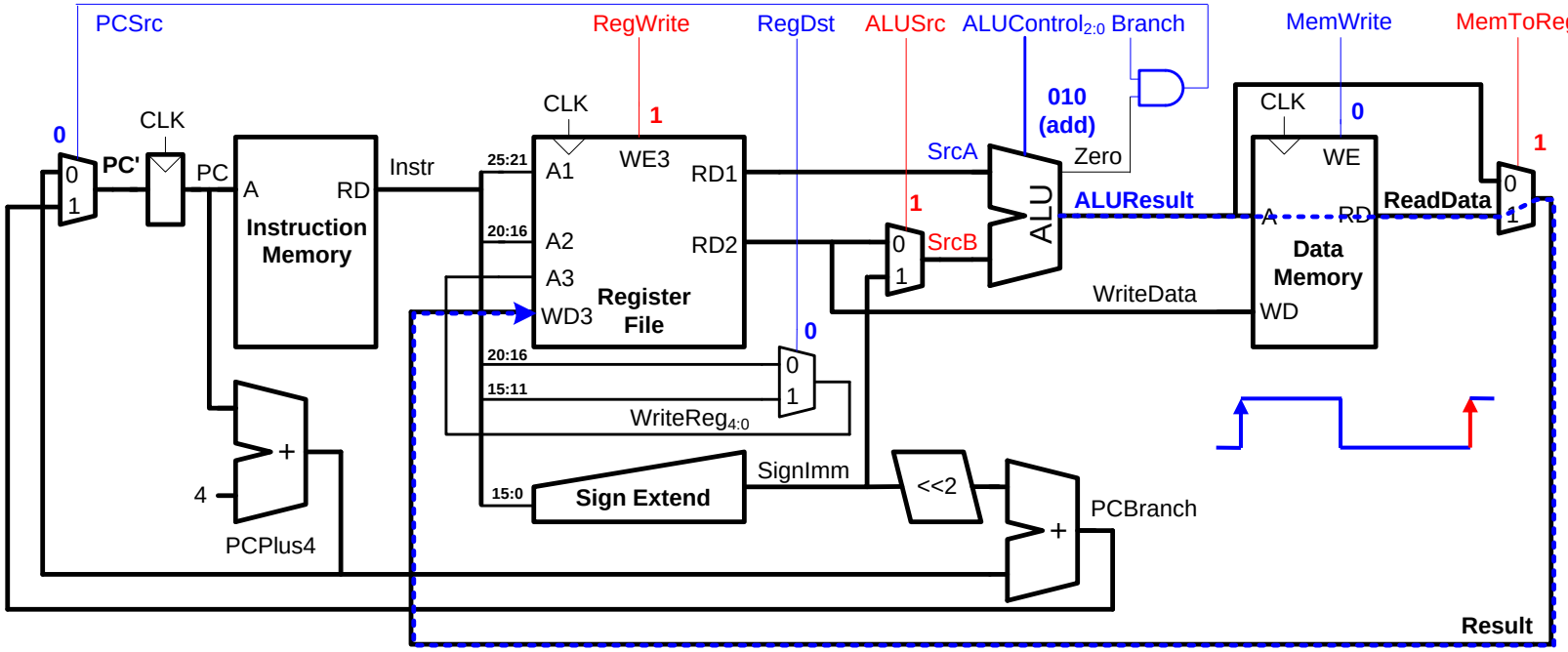
1. No flanco ascendente do *clock,* PC carrega um novo endereço.
2. A instrução é lida da Memória de Instruções.
3. O operando SrcA é lido do Banco de Registos e em paralelo o valor imediato é estendido em sinal e selecionado, pelo multi-plexer ALUSrc, como SrcB da ALU.



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

Performance SC (3) - Cam. Crítico e T_c (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

$T_{C1} + T_{C2} = T_{C,Min}$ -->

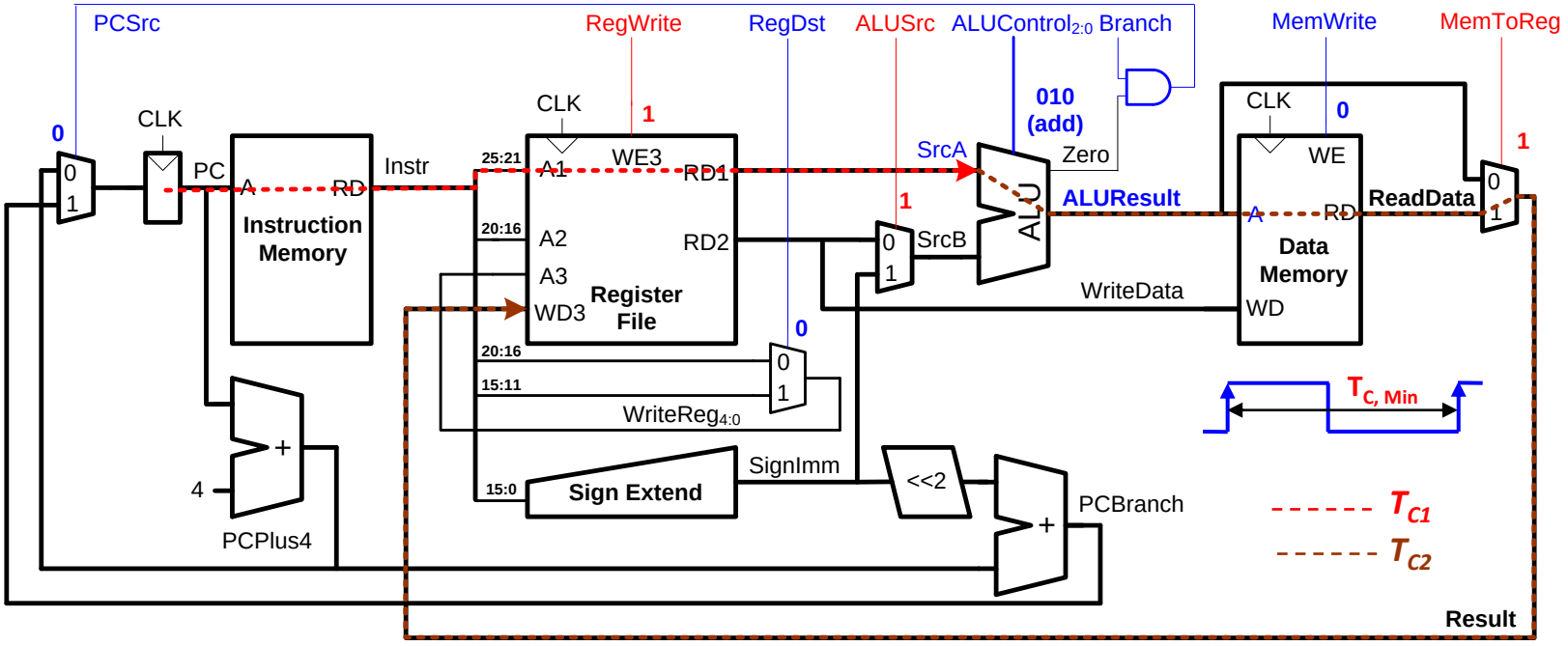
Performance SC (4) - Caminho Crítico e T_C (4) - $T_{C,Min}$

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

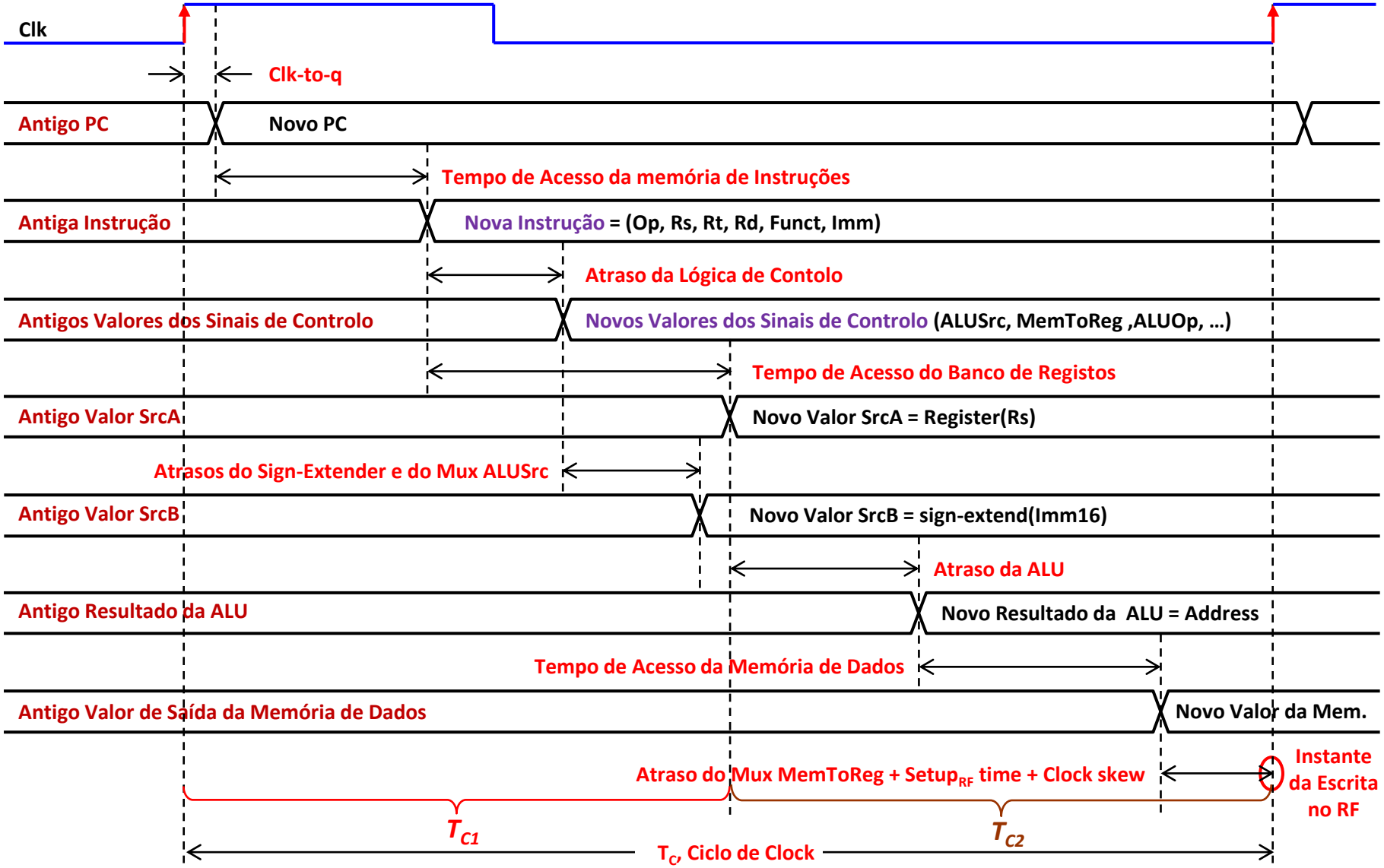
$$T_{c,Min} = \overbrace{t_{pcq\_PC} + t_{mem} + \max(t_{RFread},\ t_{sext} + t_{mux})}^{T_{C1}} + \overbrace{t_{ALU} + t_{mem} + t_{mux} + t_{RFsetup}}^{T_{C2}}$$

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

$$T_{c,Min} = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{ALU} + t_{mux} + t_{RFsetup}$$



Performance SC (5) - Caminho Crítico e T_C (5): `lw` timing



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

Tempo de Execução de um Programa

Tempo de Execução = #Instruções x (Ciclos/Instrução) x (Segundos/Ciclo)

= #Instruções x CPI x T_C

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

- #Instruções/Programa: Número de Instruções para completar a tarefa
- Ciclos/Instrução: **CPI**: Ciclos/Instrução (=1 no Single-cycle!)
- Segundos/Ciclo: $\mathbf{T_C}$, Período de Clock (1/Frequência)
- ➢ Segundos/Programa : $\mathbf{T_{EXEC}}$: 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

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

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*?

Tempo de Execução = #Instruções x CPI x T_C

= 0.1×10^{12} x (1) x 925×10^{-12} s

= 92.5 segundos

CPI = 1

T_c = 925 ps

Limitações do *Single-cycle*

Basicamente, três pontos fracos:

1. Requere 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.

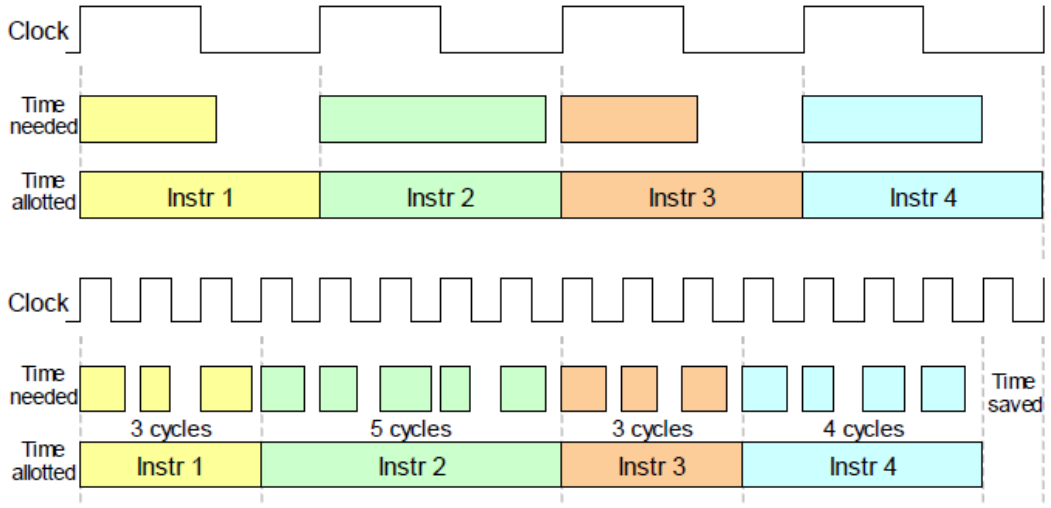
2. Usa dois somadores e uma ALU.

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*).

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

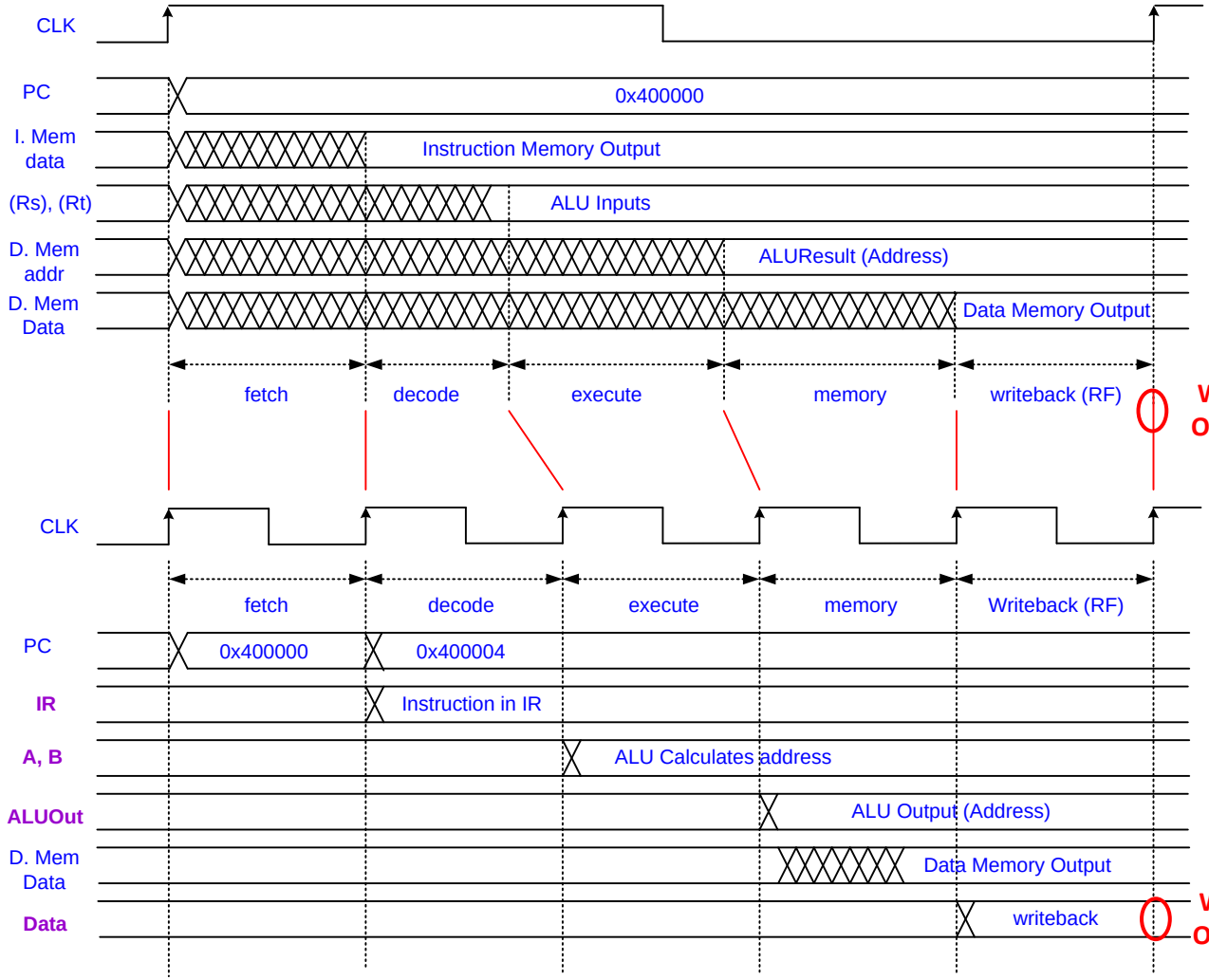
Ideia: Dividir a execução em múltiplos ciclos de *clock*



- 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... -->

Multicycle MIPS (3) - SC vs MC: `lw` timing



`lw` *timing:*
num CPU Singlecycle

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

`lw` *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.

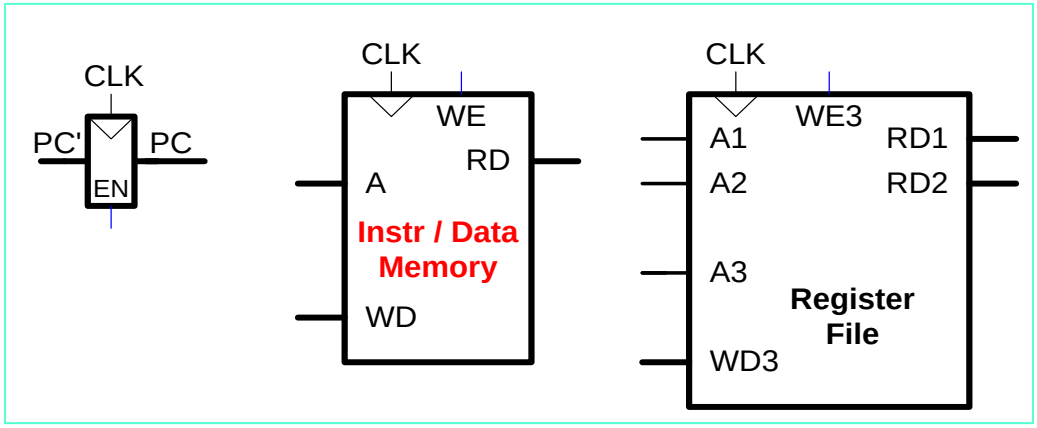
Multicycle MIPS (4) - µArquitecturas SC vs MC

- *Single-cycle*
  - \+ simples
  - \- tempo de ciclo limitado pela instrução mais longa (lw)
  - \- usa uma ALU, dois somadores e duas memórias
- *Multicycle*
  - \+ frequência de *clock* mais elevada
  - \+ as instruções mais simples executam mais rápido
  - \+ reutilização de *hardware* (caro) em ciclos distintos:
    usa uma única Memória e uma única ALU
  - \- Unidade de Controlo mais complexa:
    máquina de estados (FSM)
- Fases de *design* iguais: *datapath* + *control*

MC Elementos de Estado - Uma só Memória

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

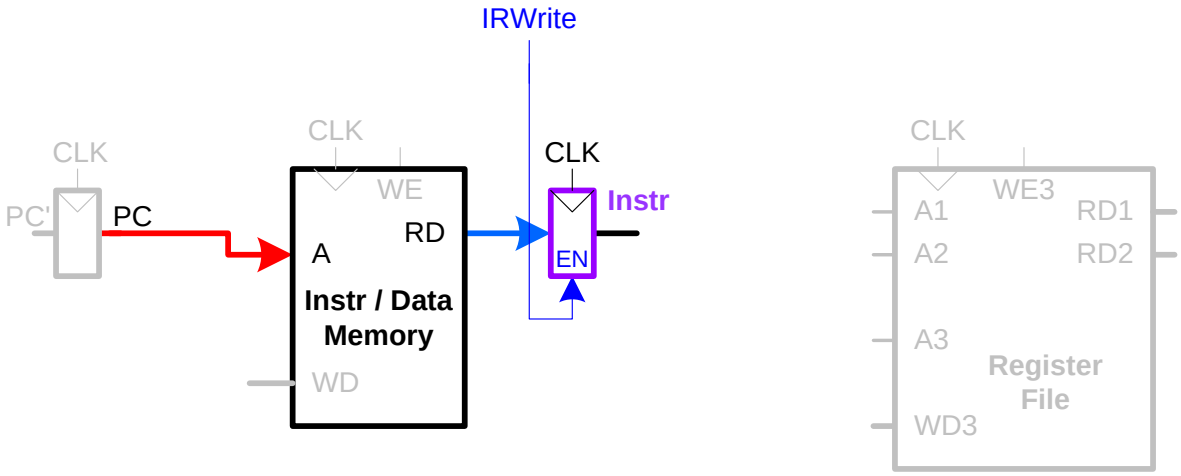
- Não era possível na implementação *Single-cycle* (As instruções e os dados são necessários durante o mesmo ciclo de clock).
- Agora, a mesma memória pode ser (re)usada se necessário (O *fetch* da Instrução e o acesso aos Dados são feitos em ciclos de clock distintos).



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

MC Datapath (1) - lw Fetch - Registo Instr

Passo 1: Leitura da Instrução (*Fetch*)

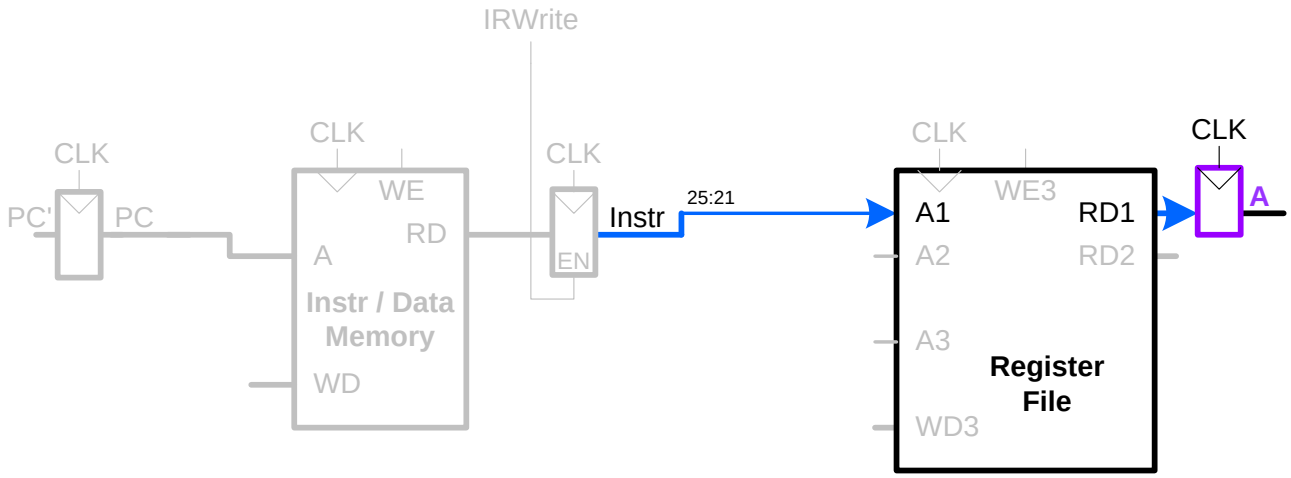


1. Usa-se um novo registo de Instrução* para reter a instrução, por forma a poder ser usada nos ciclos seguintes.

*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.

MC Datapath (2) - `lw` Leitura do Operando - Registo A

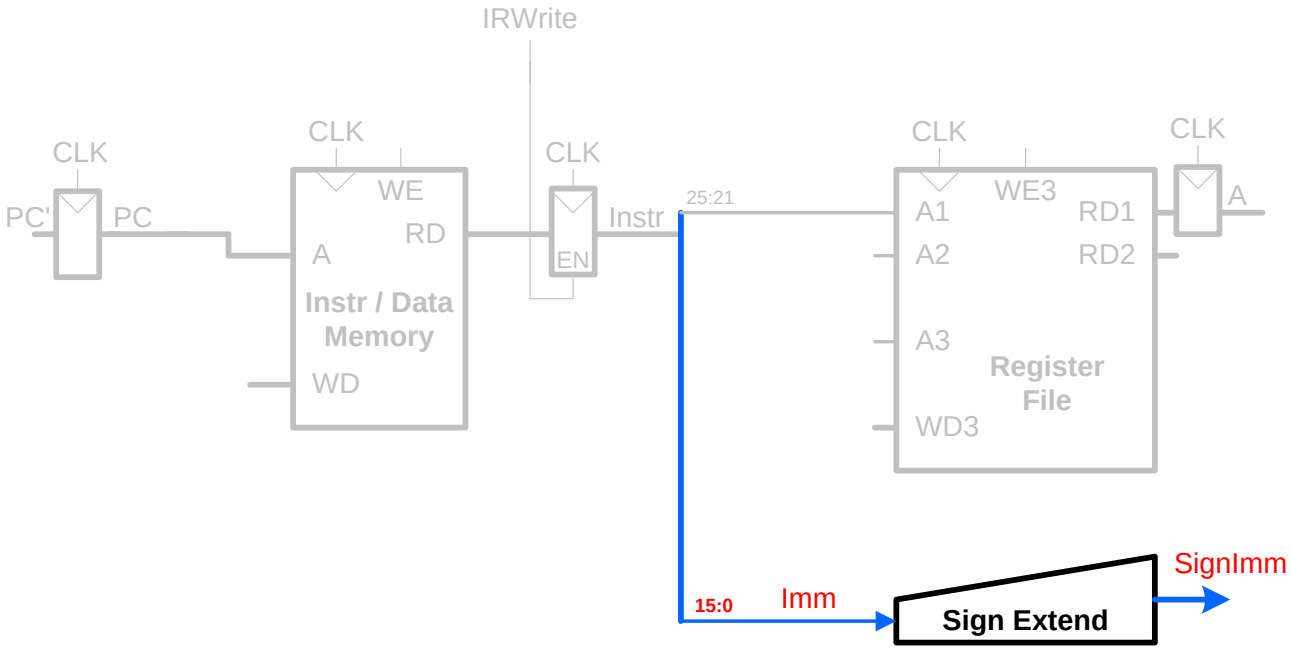
Passo 2: Leitura do operando do *Register File* (RF)



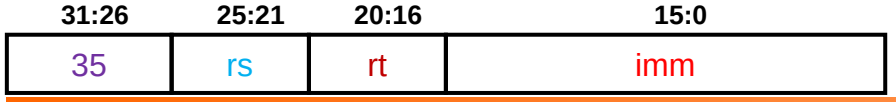
2. Insere-se mais um registo adicional (A), para reter o valor de rs (vai ser necessário no ciclo seguinte).

MC Datapath (3) - `lw` Obtenção do Offset de Endereço

Passo 3: Extensão do sinal do valor Imediato



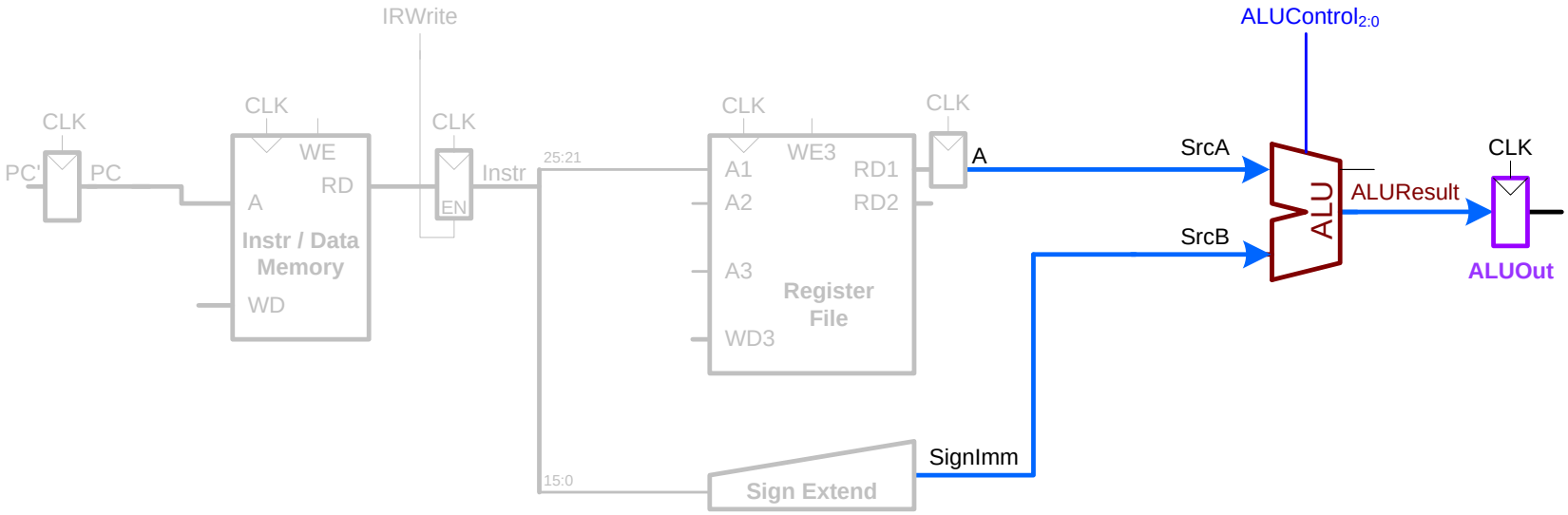
3. Converte-se o valor Imm_{16} em $SignImm_{32}$




```
lw rt,imm(rs)
```

MC Datapath (4) - lw Cálculo do Endereço - Reg. ALUOut

Passo 4: Cálculo do endereço de memória



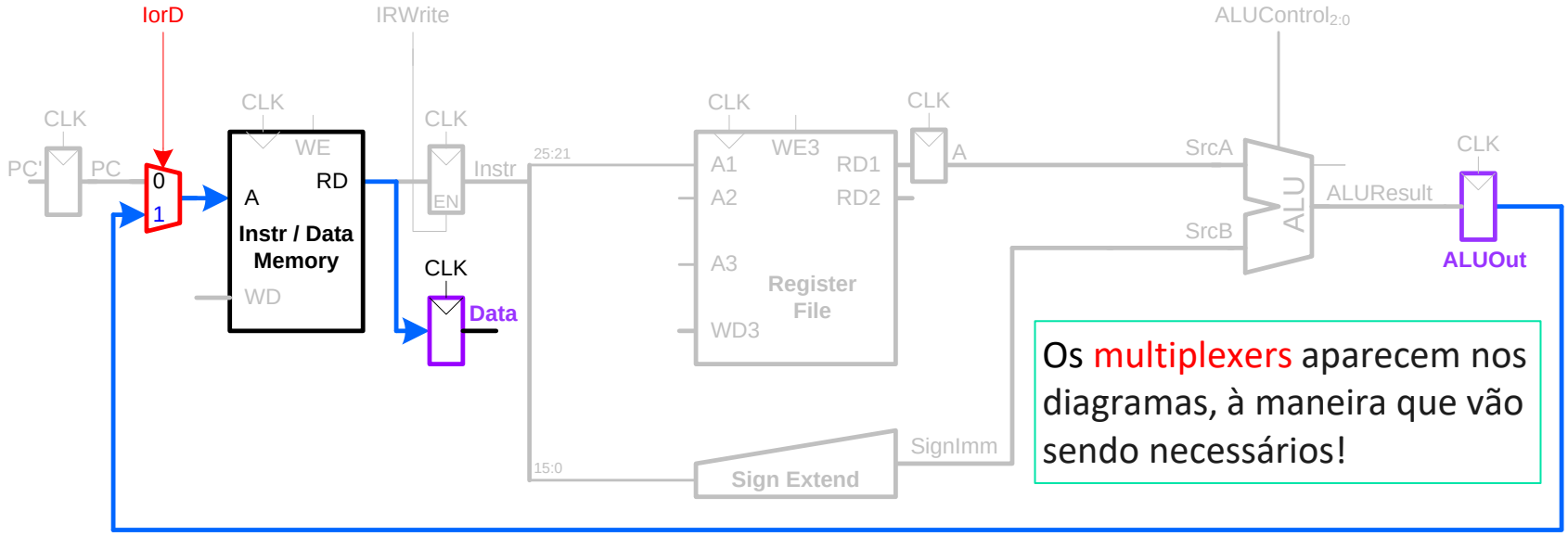
4. Na ALU soma-se o endereço base SrcA ao SrcB, para obter o endereço de memória em ALUResult (Endereço = A + SignImm).

- 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).

MC Datapath (5) - `lw` Ler Valor da Memória - Reg. Data

Passo 5: Leitura do valor da Memória



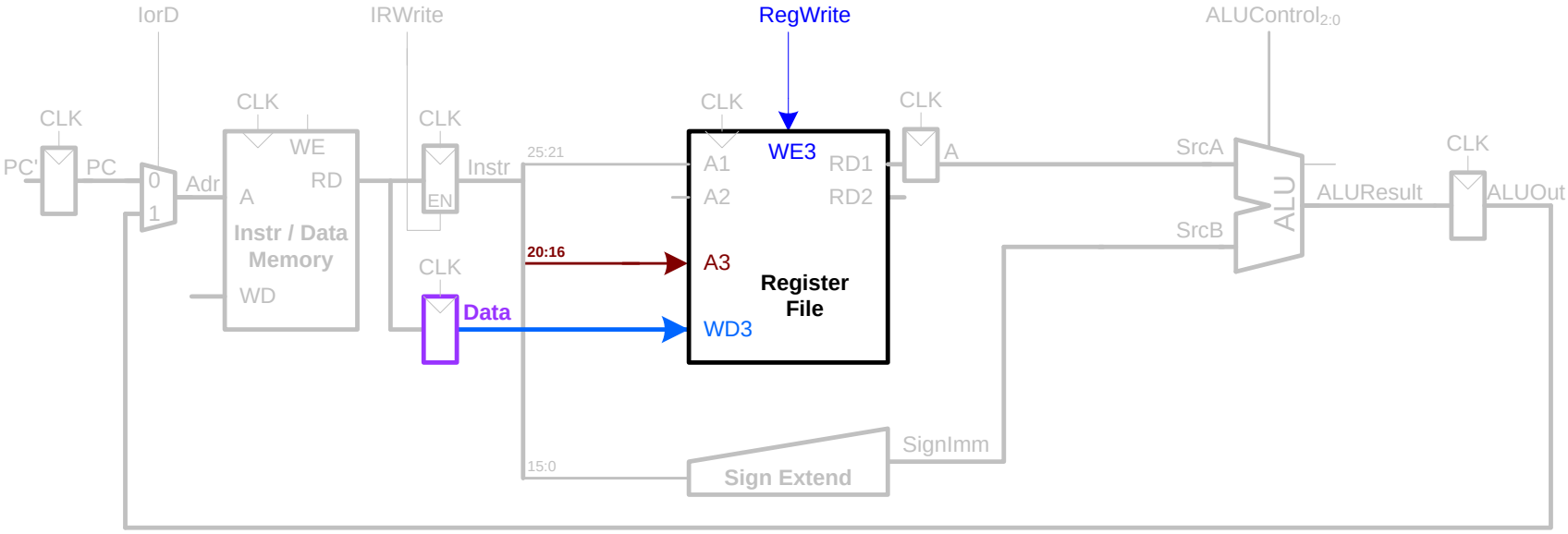
5. Insere-se um multiplexer em frente da entrada de endereço da memória (A), para selecionar o PC ou ALUOut.

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

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

MC Datapath (6) - `lw` Escrever valor no RF

Passo 6: Escrita no Banco de Registos



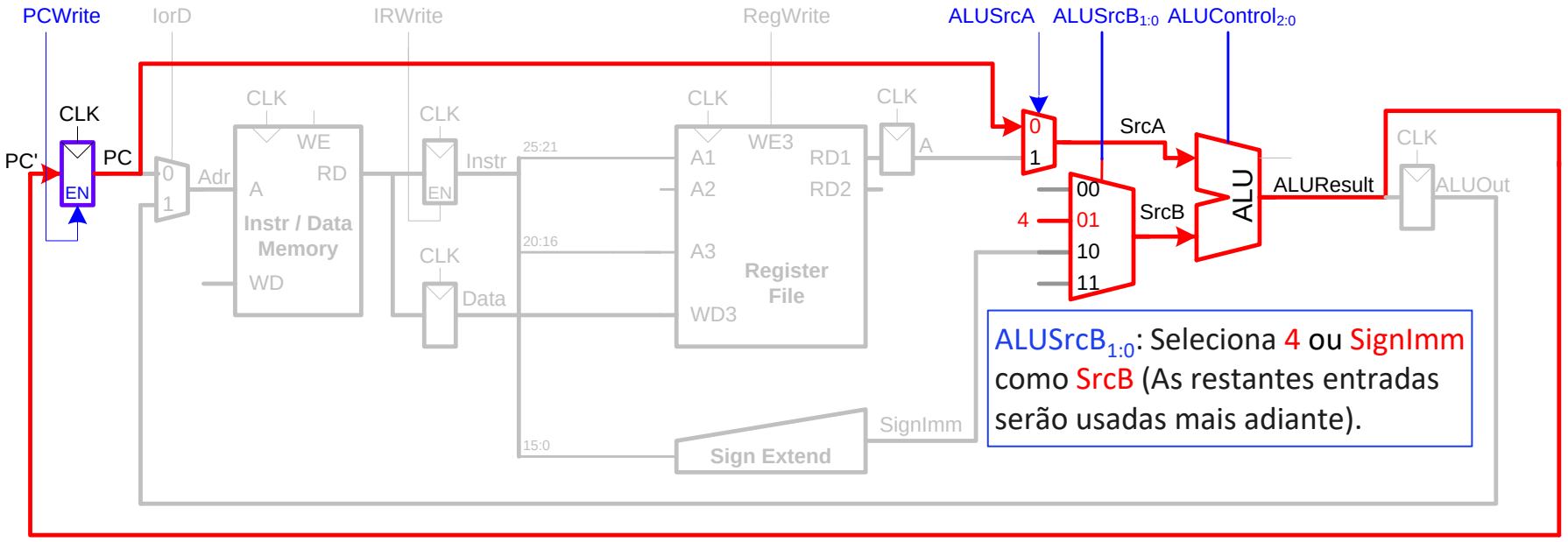
6. O valor lido da memória (Data) é escrito (RegWrite=1) no registo rt ($Instr_{20:16}$) do Banco de Registos.

31:26	25:21	20:16	15:0
35	rs	rt	imm

`lw rt,imm(rs)`

MC Datapath (7) - lw - Cálculo de PC'

Passo 7: Calculo do PC'

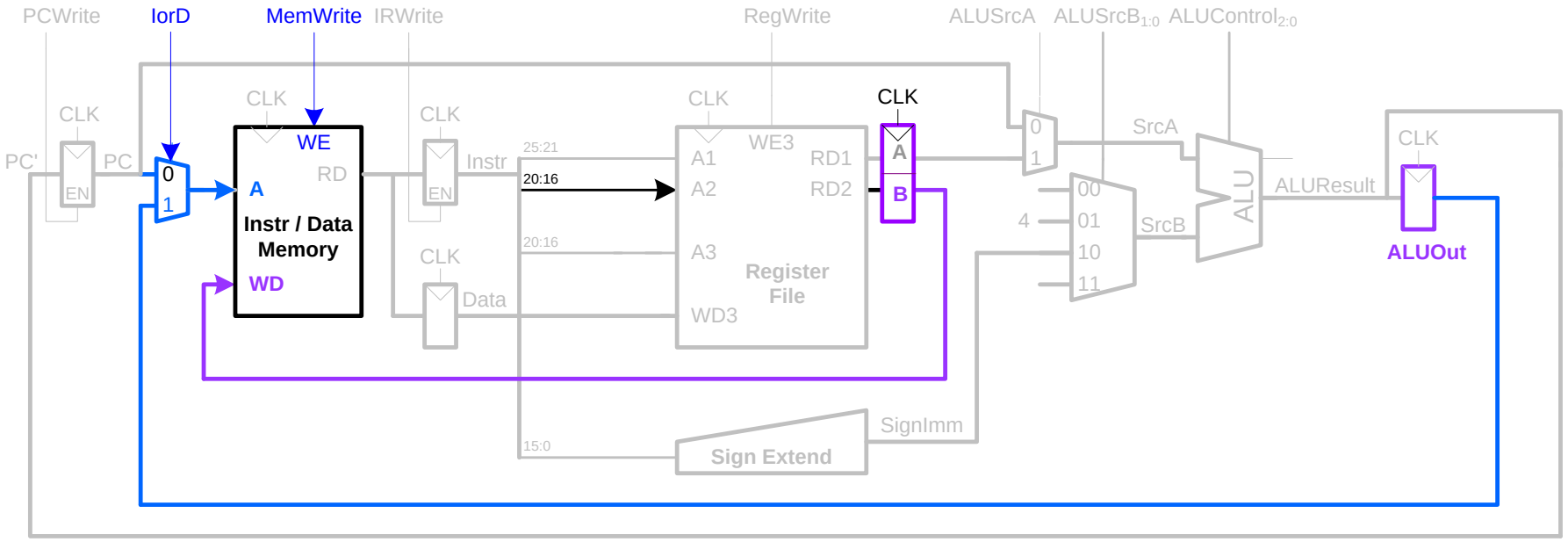


7. PC' = PC + 4. A soma é feita usando a mesma ALU (já usada no cálculo do endereço), mas em ciclos distintos.

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

MC Datapath (8) - 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:

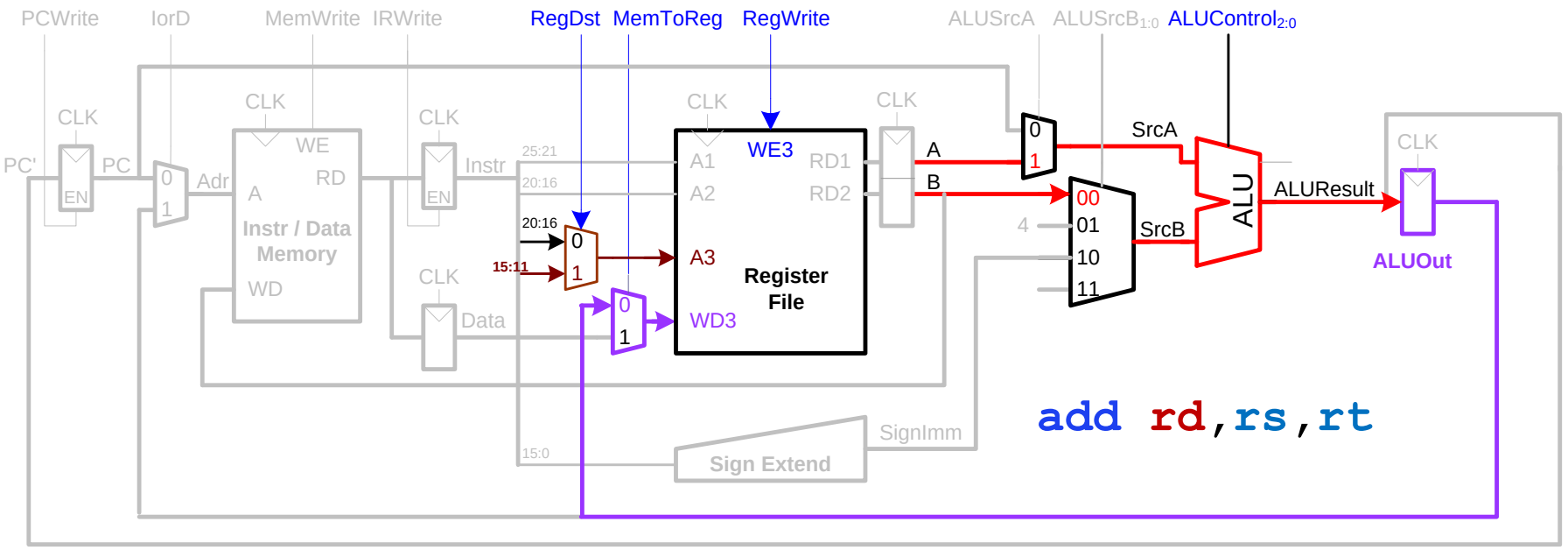
- O valor de rt está no novo registo B (também foi lido no Passo 2)
- Escreve B na memória fazendo MemWrite =1 com A=ALUOut

31:26	25:21	20:16	15:0
43	rs	rt	imm

`sw rt,imm(rs)`

MC Datapath (9) - tipo-R

- Usa os registos rs e rt como operandos

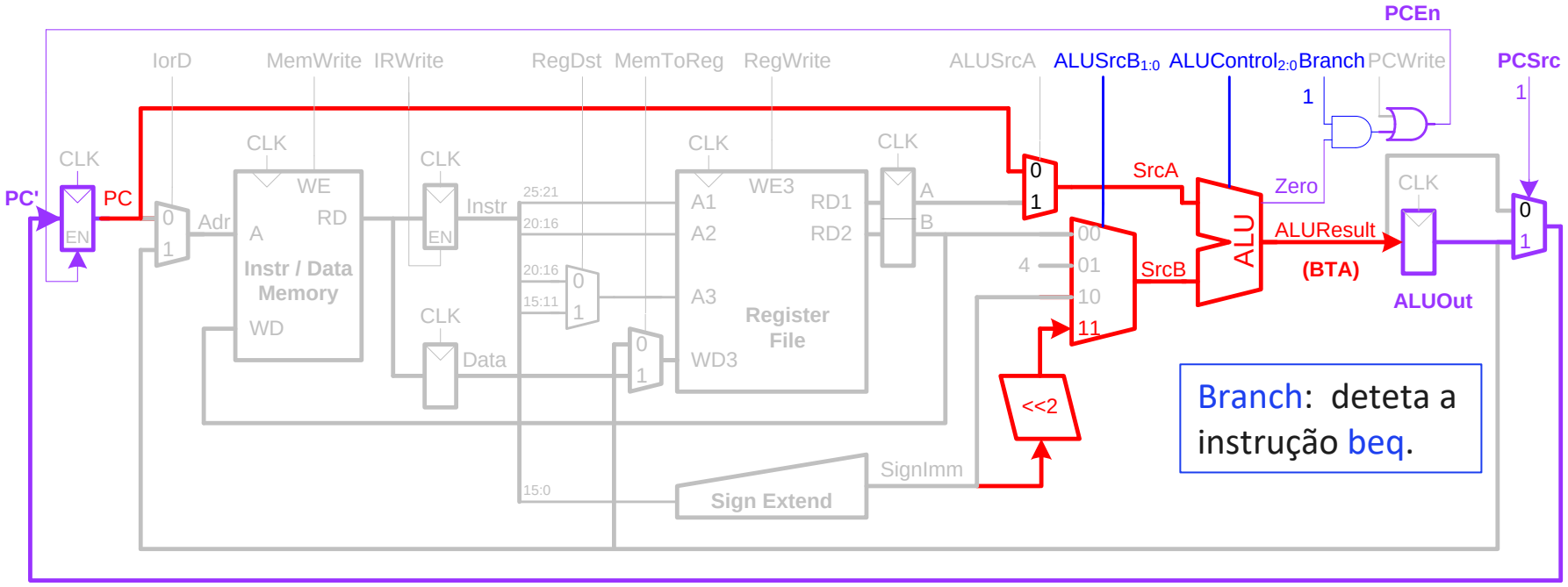


1. Executa a operação aritmética/lógica (resultado em ALUOut)
2. Escreve o resultado no registo rd (RF), RegDst=1.
   - O multiplexer MemToReg está, agora, em frente de WD3 do RF.

MC Datapath (10) - branch

`beq rs,rt,imm`

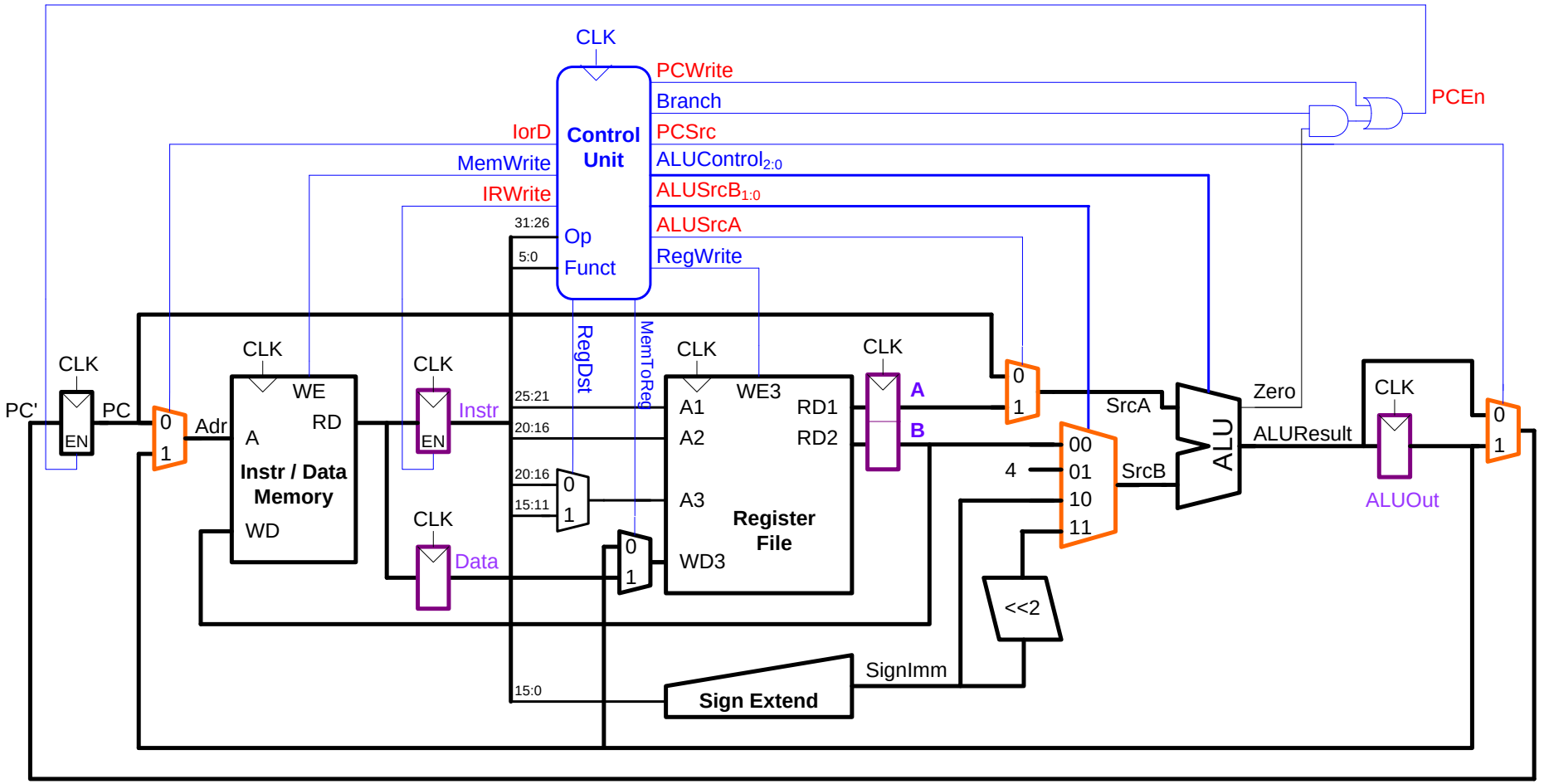
1. Calcula o endereço-alvo: BTA = ($SignImm_{32}$ << 2) + (PC+4)



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'.

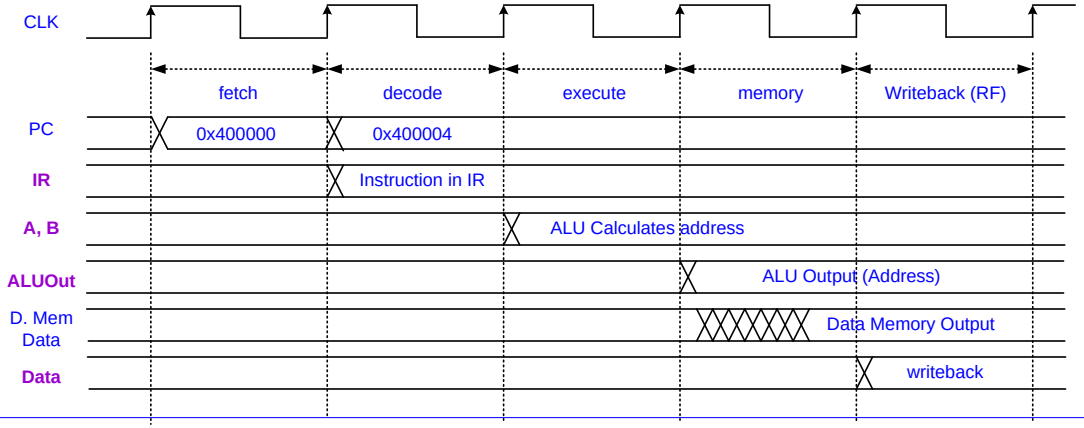
As duas operações (BTA e comparação de rs e rt) são feitas em ciclos diferentes! Ver FSM (beq).

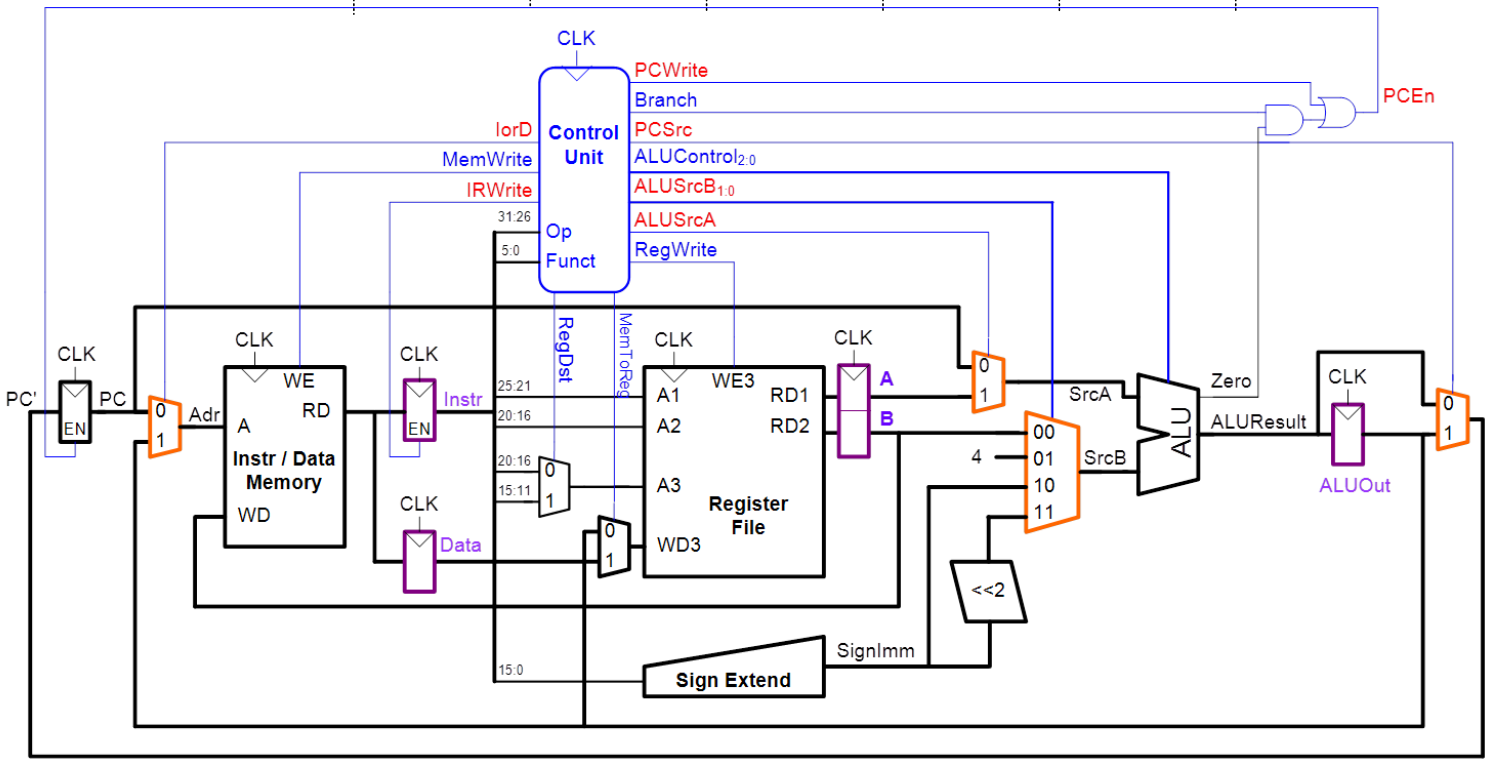
MC Datapath (11) - Completo*



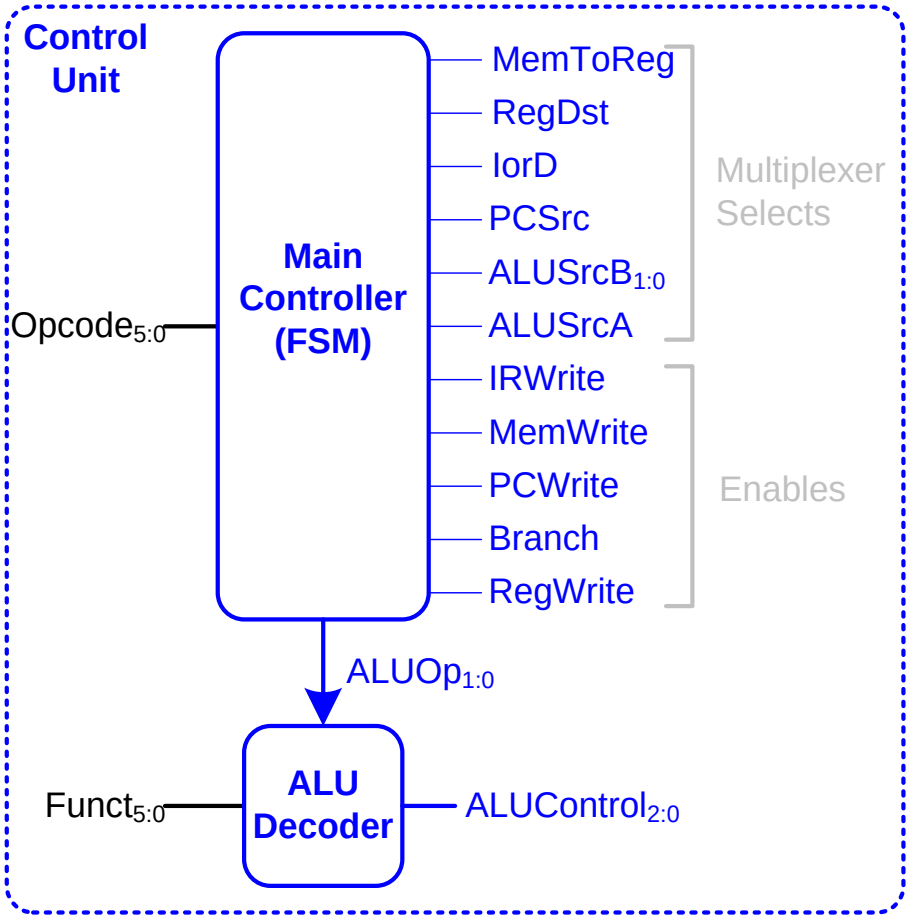
*Mas com um ISA limitado a: lw, sw, tipo-R e beq.

MC Datapath (12) - Completo - com `lw` timing





A seguir: Unidade de Controlo MC: FSM



Z - Programa das Teóricas

Datapath do µP MIPS: *Multicycle*

Aula	Data	Descrição
25	22/Mai	*Multicycle*: Limitações das uArquiteturas *Single-cycle*; Versão de referência duma uArquitetura *Multicycle*; Exemplos do processamento das instruções numa uArquitetura *Multicycle.*
26	27/Mai (4ª)	Unidade de Controlo *para datapath Multicycle*: Diagrama de estados. Sinais de controlo e valores do *datapath Multicycle*. Exemplos da execução sequencial de algumas instruções *no datapath multicycle*.
27	29/Mai	Resolução de problemas sobre uArquiteturas *Multicycle*.

Comunicação do µP com o exterior (I/O)

Aula	Data	Descrição
28	3/Jun	**VII -** Comunicação com o exterior: Entrada e saída de dados (I/O) Acesso a dispositivos de I/O: I/O Memory-Mapped Hardware e Software (Asm). Dispositivos de I/O Embedded uControlador PIC32 (MIPS): I/O Digital: Switches e LEDs I/O Série : SPI e UART.
29	5/Jun	I/O sob interrupção: Timers e Interrupções I/O analógico: ADC e DAC; Outros periféricos: LCD e Bluetooth.