Introdução à Arquitetura de Computadores

Aula25

µArquitetura MIPS Multicycle: I

Performance Single-cycle

Caminho Crítico; Tempo de Execução

Arquitetura Multicycle (MC)

Limitações do *datapath Single-cycle*

Multicycle versus *Single-cycle*

Datapath MC

Elementos de Estado: Memória Única

Execução de instruções:

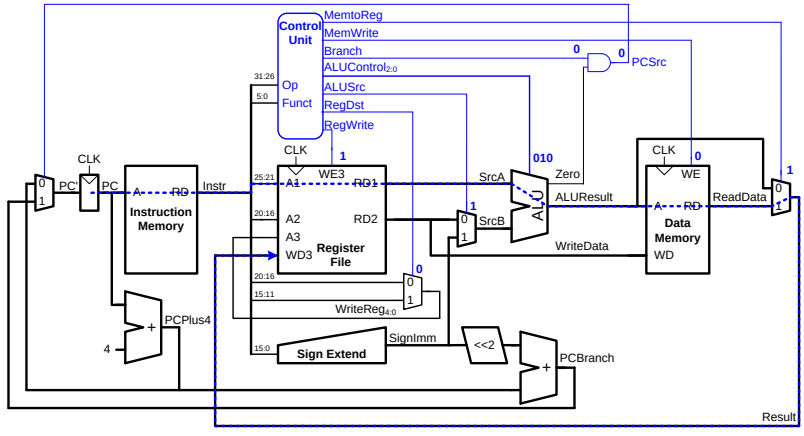
Acesso à Memória: lw, sw,

Tipo-R e

Branch (beq)

Performance SC (1) - Caminho Crítico: T_C (1)

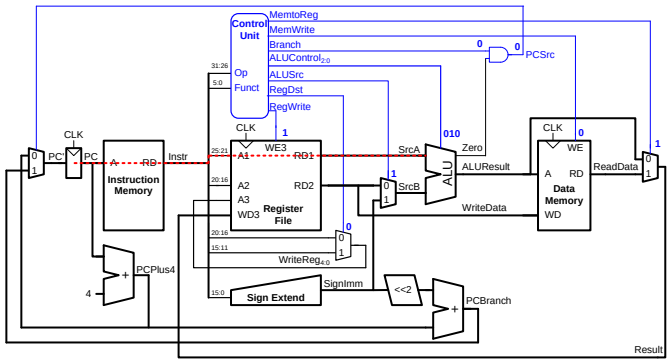
Num processador *single-cycle* todas as instruções executam num ciclo de *clock*.
O período mínimo do *clock* (T_C) é limitado pelo caminho crítico (*critical path*) da instrução `lw` (ilustrado no diagrama abaixo com a linha azul tracejada).



T_C limitado pelo caminho crítico (*lw*)

Performance SC (2) - Caminho Crítico: T_C (2)

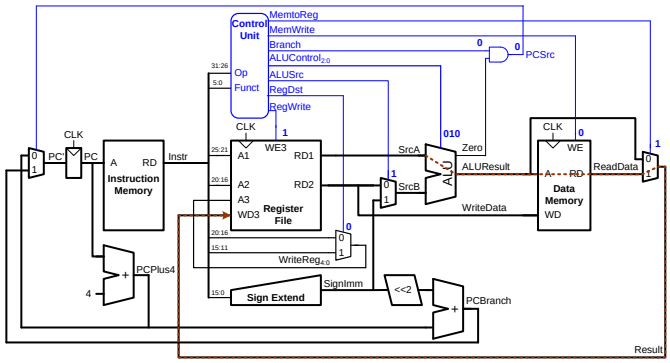
1. PC carrega um novo endereço no flanco ascendente do *clock*.
2. A instrução é lida da Memória de Instruções.
3. O operando SrcA é lido do Banco de Registos (e simultaneamente, não tracejado no diagrama, o valor imediato é estendido em sinal e ALUSrc seleciona-o como SrcB no multiplexer).



T_{C1} : CLK to Instruction_Mem to Register_File to ALU

Performance SC (3) - Caminho Crítico: T_C (3)

4. A ALU soma SrcA com SrcB para calcular 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 Banco de Registos, respeitando o tempo de *setup*, para ser escrito no próximo flanco ascendente do *clock*.



T_{C2} : ALU to Data_Mem to Mux to Register_File

Performance SC (4) - Caminho Crítico: T_C (4)

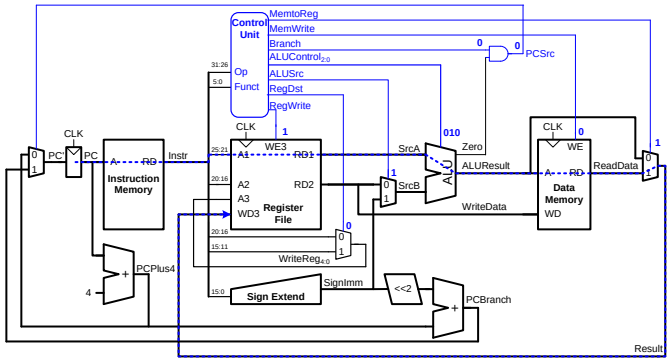
- Caminho Crítico *Single-cycle*:

  $T_c = t_{pcq\_PC} + t_{mem} + \max(t_{RFread}, t_{sext} + t_{mux}) + t_{ALU} + t_{mem} + t_{mux} + t_{RFsetup}$

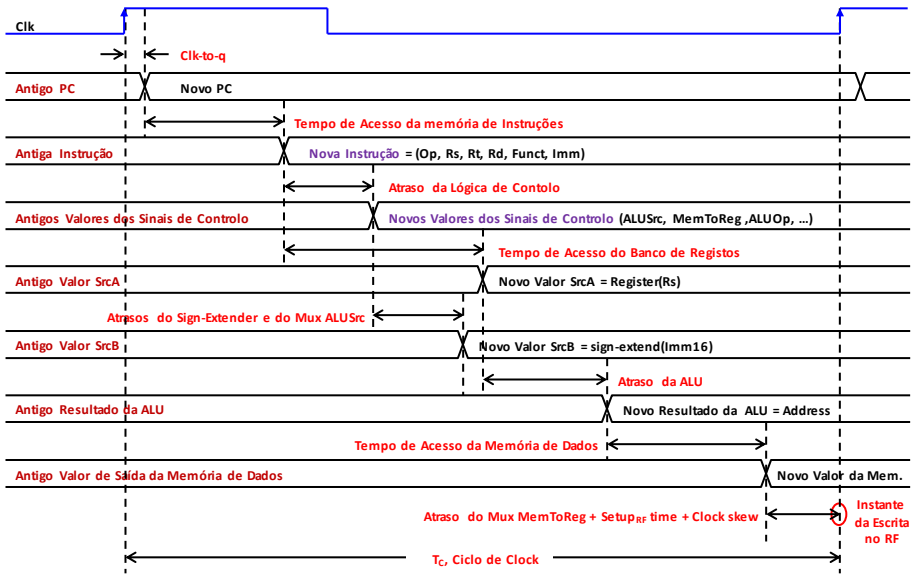
- Na maioria das implementações os limites são impostos por:

  Memórias, Banco de Registos e ALU.

  $T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{ALU} + t_{mux} + t_{RFsetup}$



Performance SC (5) - Caminho Crítico: LW Timing



Performance SC (5) - Tempo de Execução (1)

Tempo de Execução de um Programa

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

= #Instruções x CPI x T_C

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

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

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

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

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

Estes atrasos são, em geral, dependentes da tecnologia usada para implementar a lógica.

Performance SC (6) - Tempo de Execução (3)

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

Multicycle MIPS (1) - Limitações Single-cycle

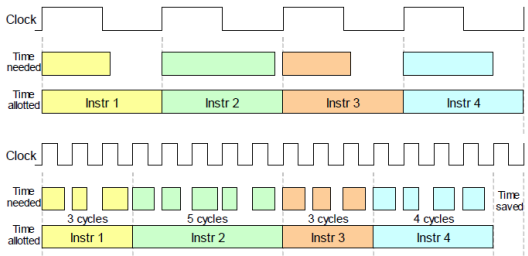
Limitações do *Single-cycle*

Basicamente, três pontos fracos:

1. Requere um período de *clock* suficientemente longo para acomodar a instrução com o tempo de execução mais longo (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 (*Harvard*). Hoje em dia, a maioria dos computadores usa uma única memória (*Von Neumann*) para instruções e dados.

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

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



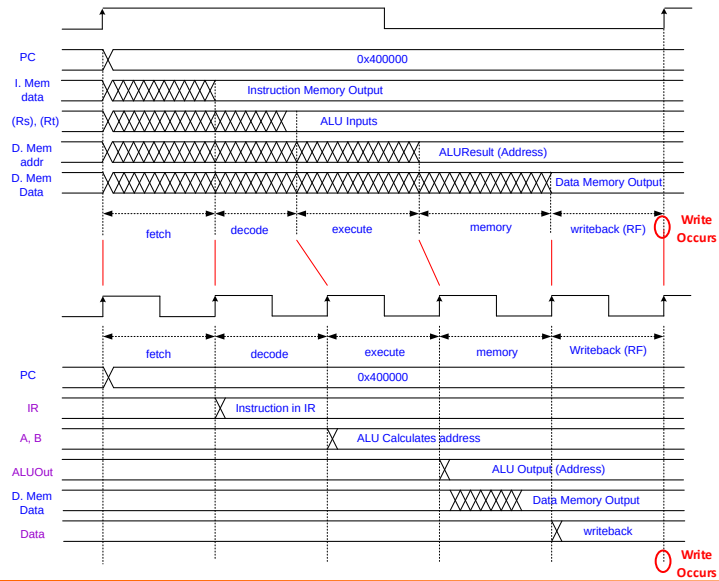
Single-cycle

Multicycle

- No Single-cycle todas as instruções ocupam o mesmo tempo de execução.
- No 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 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 *Single-cycle*

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.

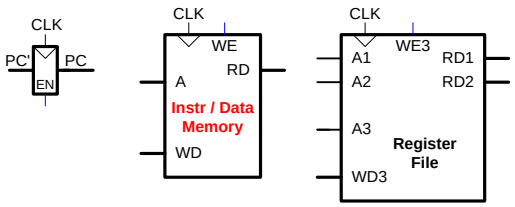
Multicycle MIPS (4) - SC vs MC: Sumário

- Microarquitectura *Single-cycle*
  - \+ simples
  - - tempo de ciclo limitado pela instrução mais longa (lw)
  - - uma ALU, dois somadores e duas memórias
- Microarquitectura *Multicycle*
  - \+ frequência de *clock* mais elevada
  - \+ as instruções mais simples executam mais rápido
  - \+ reutilização de *hardware* (caro) em ciclos distintos: 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*

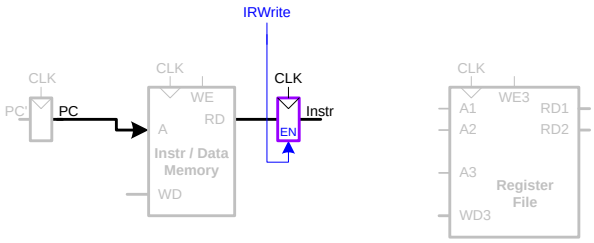
- Não era possível na implementação *Single-cycle*
  As instruções e os dados são necessários durante o mesmo ciclo de clock.
- Agora, a mesma memória pode ser (re)usada se necessário
  O *fetch* da instrução e o acesso aos dados são feitos em ciclos de clock distintos.



*A arquitetura de Von Neumann substitui a de Harvard

MC Datapath (1) - lw Fetch

Passo 1: Leitura da Instrução (*Fetch*)

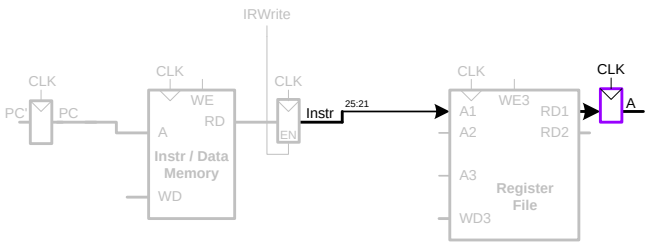


1. Usa-se um novo Registo de Instrução* (Instr) para reter a instrução por forma a poder ser usada nos ciclos seguintes.

* Registo não acessível ao programador, designado por '*non-architectural*' (contrariamente ao que acontece com os registos do RF e mesmo com o PC).
Estes registos (e são vários) 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

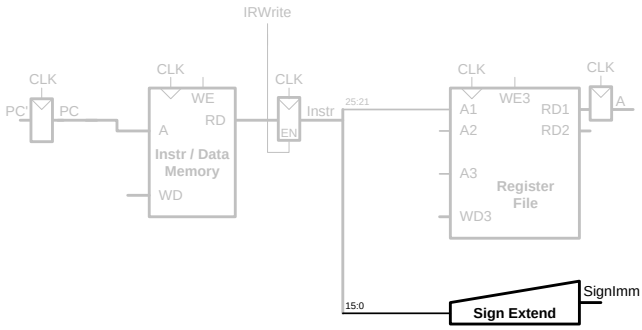
Passo 2: Leitura do operando do *Register File* (RF)



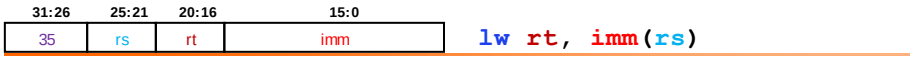
2. Insere-se mais um registo adicional (RegA), para reter o valor de rs (vai ser necessário no ciclo seguinte).

MC Datapath (3) - lw Obtenção do Offset

Passo 3: Extensão do sinal do valor Imediato

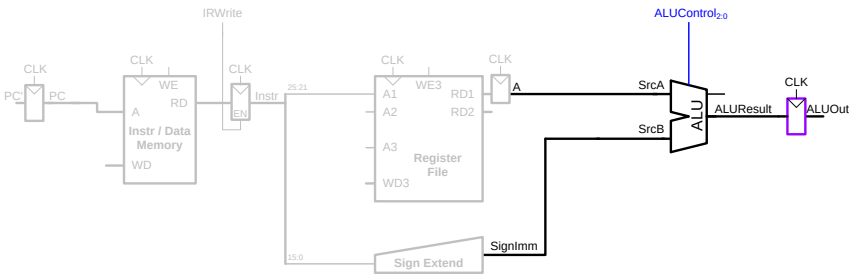


3. Converte-se o valor Imm_{16} , Instr15:0, em $SignImm_{32}$



MC Datapath (4) - lw Cálculo do Endereço

Passo 4: Cálculo do endereço de memória

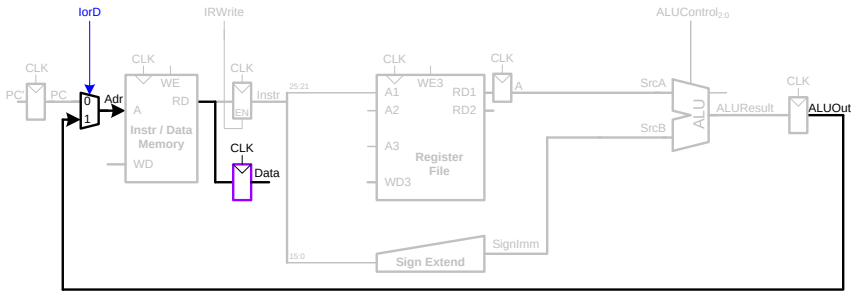


4. Na ALU soma-se o endereço base SrcA ao SrcB, para obter o endereço de memória em ALUResult (endereço = A + SignImm) .
Insere-se um novo registo na saída da ALU, ALUOut.

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

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

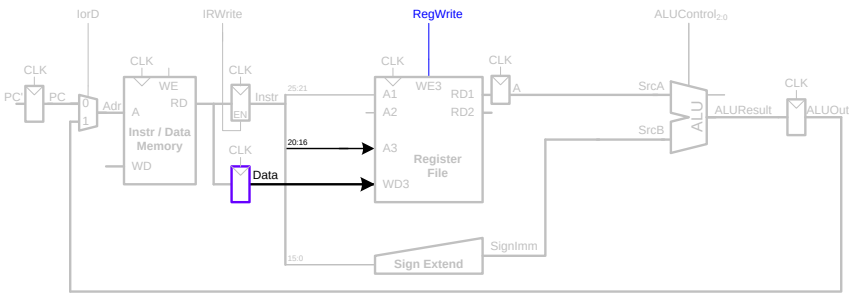
Passo 5: Leitura do valor da Memória



5. Insere-se um multiplexer em frente da entrada de endereço da memória (Adr), para selecionar o PC ou ALUOut (i.e., o *Fetch* da instrução ou o acesso a dados é controlado pelo novo sinal de controlo IorD). O valor lido (RD) é colocado noutro registo (Data).

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

Passo 6: Escrita no Banco de Registos



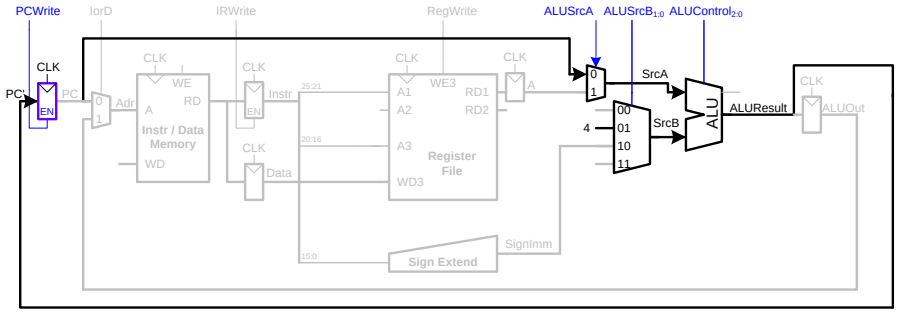
6. O valor lido da memória (Data) é escrito (RegWrite=1) no registo rt (Instr20:16) do Banco de Registos.

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

MC Datapath (7) - lw Incrementar o PC

Passo 7: Calculo do PC'

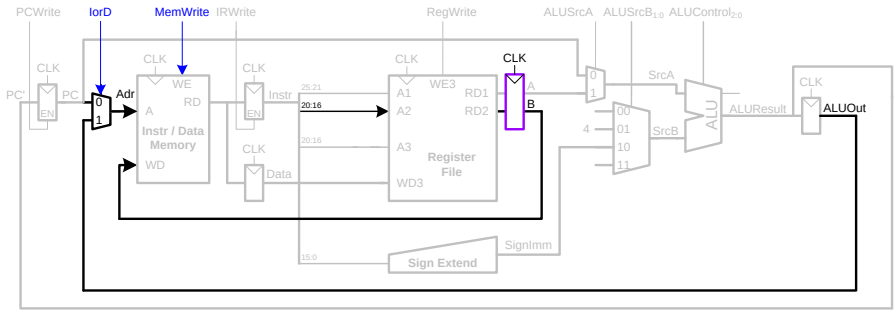
$ALUSrcB_{1:0}$: Seleciona 4 ou SignImm como SrcB (As restantes entradas serão usadas mais adiante).



7. PC' = PC + 4 . A soma é feita (re)usando a mesma ALU (tb usada no cálculo do endereço), mas em ciclos distintos.
O novo sinal de controlo PCWrite (*enable* do registo PC) possibilita a atualização do PC só em determinados ciclos.

MC Datapath (8) - sw versus lw

Passo 5: Escrita do valor de rt na Memória



Comparando **sw** com **lw** (e neste mesmo ciclo de clock):

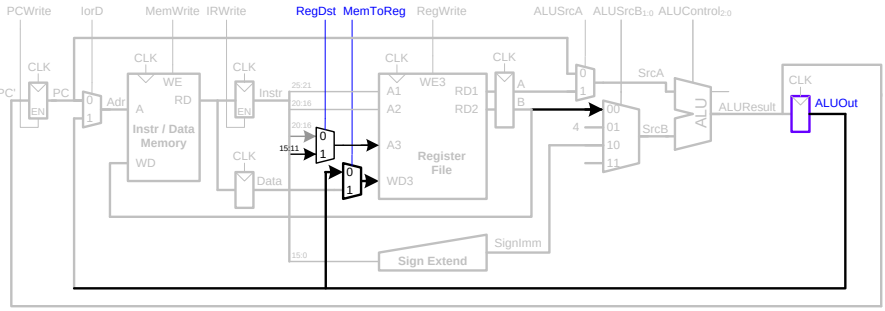
- O valor de rt está no novo registo RegB (tb foi lido no Passo 2)
- Escreve B na memória fazendo MemWrite =1 com Adr =ALUOut

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 `add rd, rs, rt`

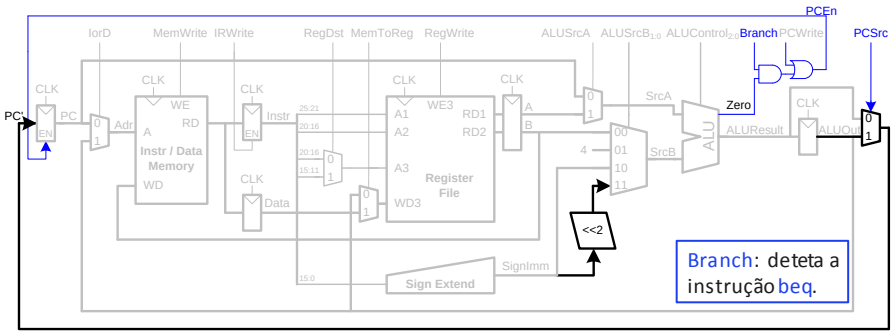


- Executa a operação aritmética/lógica (resultado em ALUOut)
- Escreve o resultado no registo rd (RF), RegDst=1
- O multiplexer MemToReg está, agora, em frente de WD3 do Banco de Registos.

MC Datapath (10) - Branch

`beq rs,rt,imm`

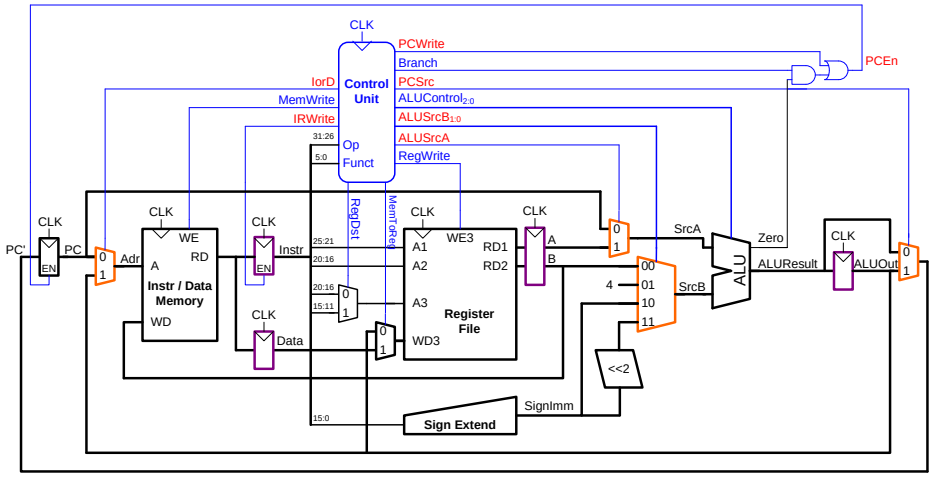
- Determina se o valor dos registos rs e rt é igual (Zero)



- Calcula o endereço-alvo (reusando a ALU):
  BTA = $(\text{SignImm}_{32} << 2) + (\text{PC}+4)$
- O novo multiplexer PCSrc, escolhe o signal que atualiza o valor de PC'. O PC tanto pode ser atualizado (PCEn) pelo PCWrite como devido a um *Branch*.

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

MC Datapath (11) - Completo*



*Mas, com um *instruction set* limitado a: lw/sw, tipo-R e beq