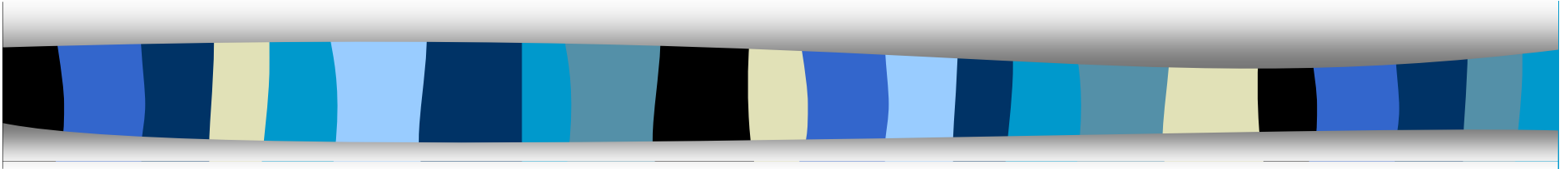


“Arquitetura e Organização de Computadores I – Aula_07 – *Pipeline* – Parte 1”



Prof. Dr. Emerson Carlos Pedrino
DC/UFSCar
São Carlos





Objetivos

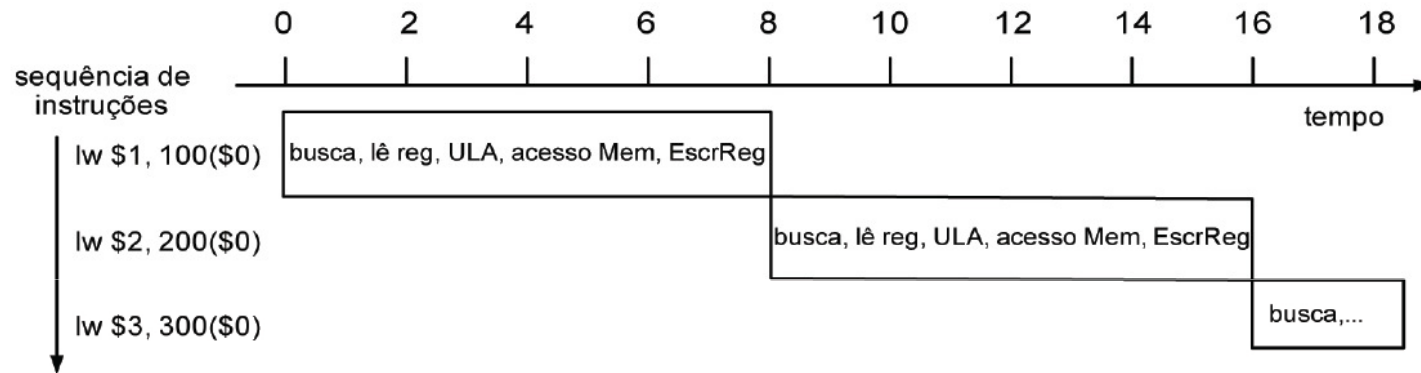
- Descrição dos princípios de *pipeline* (Parte 1).
- Outros mecanismos de paralelização aplicados em computadores (Parte 2).



MIPS Monociclo

- Todas as instruções são executadas em um ciclo, cujo período de tempo é calculado em função da instrução mais demorada.
- Instrução mais simples: neste caso sobrar um tempo de ociosidade até terminar um ciclo de *clk*.

Sequência de execução de instruções no MIPS monociclo



- Ex: tempos de ciclo:
 - busca, acesso à memória e uma operação na ULA: 2 ns.
 - leitura e escrita no banco de registradores: 1 ns.
 - 1 ciclo de instrução: 8ns, e 3 instruções: 24 ns.



MIPS Multiciclo

- Uma instrução: implementada em vários ciclos, com períodos menores em relação ao monociclo.
- Uso de mais ciclos para instruções mais demoradas.
- Uso de menos ciclos para instruções mais simples.

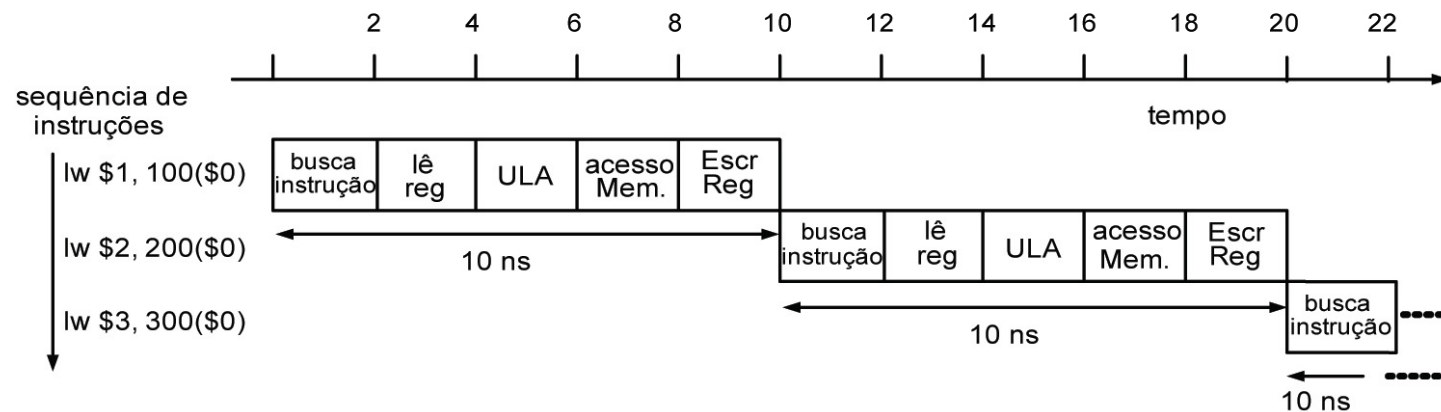


MIPS Multiciclo

- Em um determinado ciclo apenas uma parte do processador funciona, enquanto outras ficam ociosas.
- Tempo de ciclo: igual ao da etapa mais demorada do monociclo: 2 ns.
 - Ex: tempo de execução de uma instrução *load-word* = 10 ns. O MIPS monociclo leva vantagem no tempo de leitura e escrita de registradores (1 ns).

Exemplo: MIPS Multiciclo

- Sequência de execução de instruções no MIPS multiciclo:





Pipeline (nova forma de implementação)

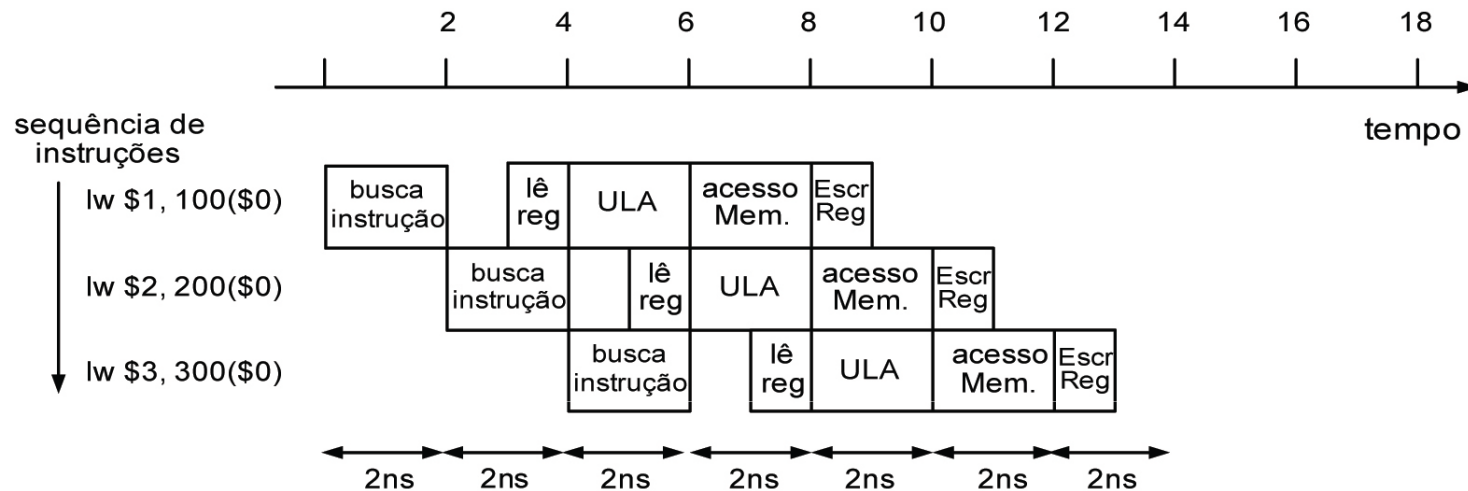
- *Pipelining*: técnica de implementação na qual várias instruções são sobrepostas na execução, semelhante a uma linha de montagem.
- Há uma analogia com um duto (*pipe*) que conduz ininterruptamente alguma matéria.
- A cada instante:
 - numa extremidade: injeção de substância.
 - em outra: ejeção da matéria.
 - e em todas as outras partes: há a condução simultânea de porções diferentes dessa matéria.



Pipeline

- **Processamento de uma instrução:** é dividido em estágios.
- Em **cada ciclo** é permitida a entrada de uma **nova instrução**, enquanto outras são executadas, só que em **estágios diferentes**.
- Em **média:** uma instrução se inicia enquanto **outra termina** sua execução, e **outras** estão em **estágios intermediários**.

Exemplo



- Tempo de execução de 3 instruções *load-word* no MIPS pipeline: 14 ns. (implementação mais eficiente)
- No monociclo: 24 ns.
- No multiciclo: 30 ns.



Pipeline em Computadores

- *Speed-up* para um programa de n instruções, num computador *pipeline* de k estágios, em relação a um computador multiciclo de k ciclos (considerando o mesmo tempo de ciclo em cada caso):
 - Observações:
 - Computador multiciclo: $\text{num_ciclos} = n \times k$.
 - Computador *pipeline*: soma do $\text{num_ciclos_instr_1} = k$, com o $\text{num_ciclo_outras_instrs} = n - 1$



Speed-up (Pipeline)

$$\text{Speed-up} = \frac{n \cdot k}{k + n - 1}.$$

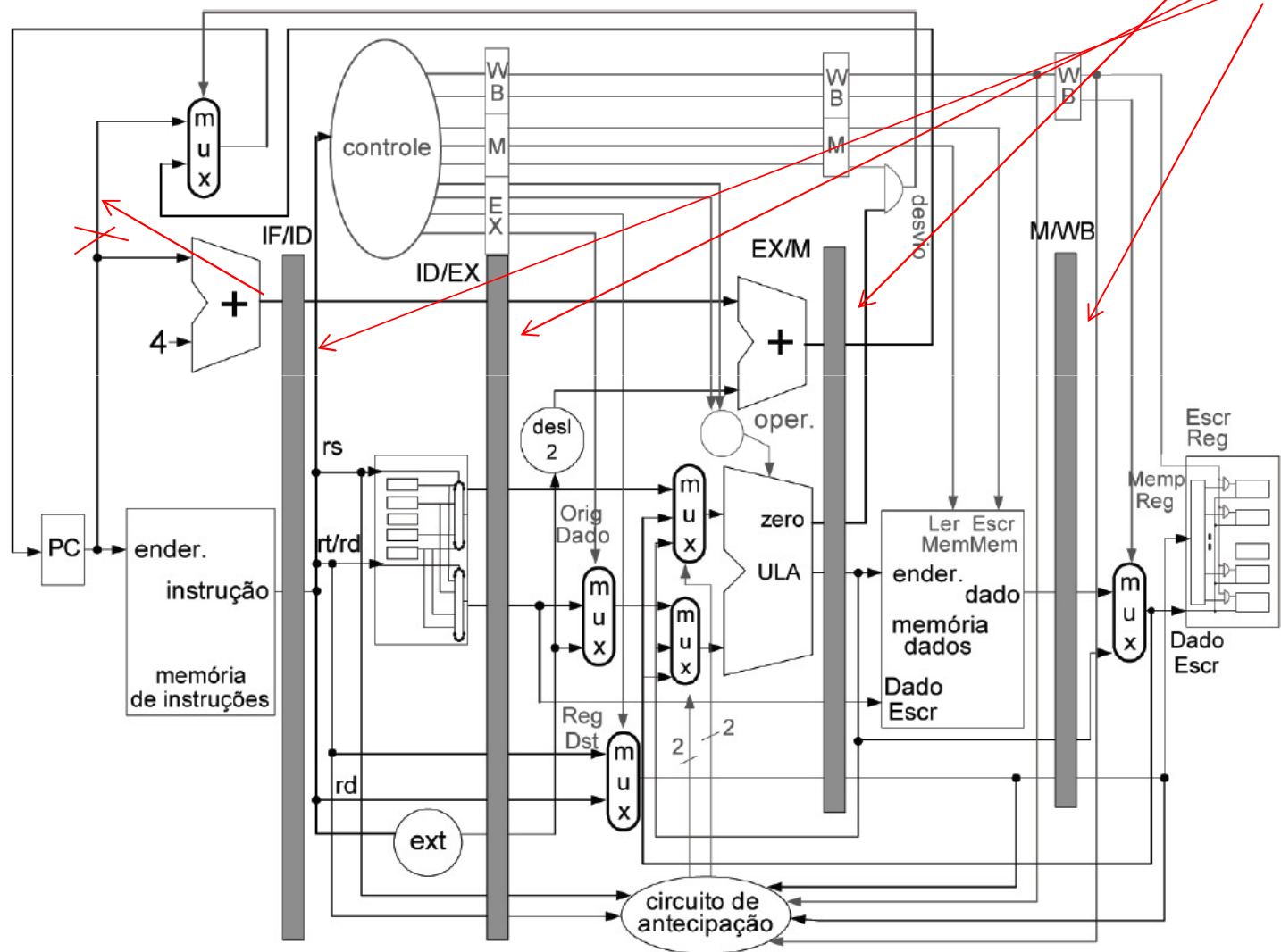
- Para *n* grande, o *speed-up* se aproxima de *k*.



Considerações sobre o Computador *Pipeline*

- Facilidade de implementação quando (Ex: MIPS):
 - todas as **instruções** têm o **mesmo tamanho**;
 - **poucos formatos de instrução**;
 - **operandos de memória** aparecem somente em **loads** e **stores**.
- Dificuldade de implementação quando:
 - **conflitos estruturais** (ex: somente uma memória);
 - **conflitos de controle** (preocupação com instruções de desvio);
 - **conflitos de dados** (uma instrução depende de outra prévia);
 - **manipulação de exceções**.

Pipeline no MIPS





Observações

- *Latches*: guardam todos os resultados de cada estágio.
- A cada ciclo, todos os estágios operam simultaneamente. No MIPS multiciclo apenas uma parte do processador é ativa a cada ciclo.
- *Latch IF/ID*: *latch* inserido entre os estágios de busca de instrução e de decodificação.



Observações

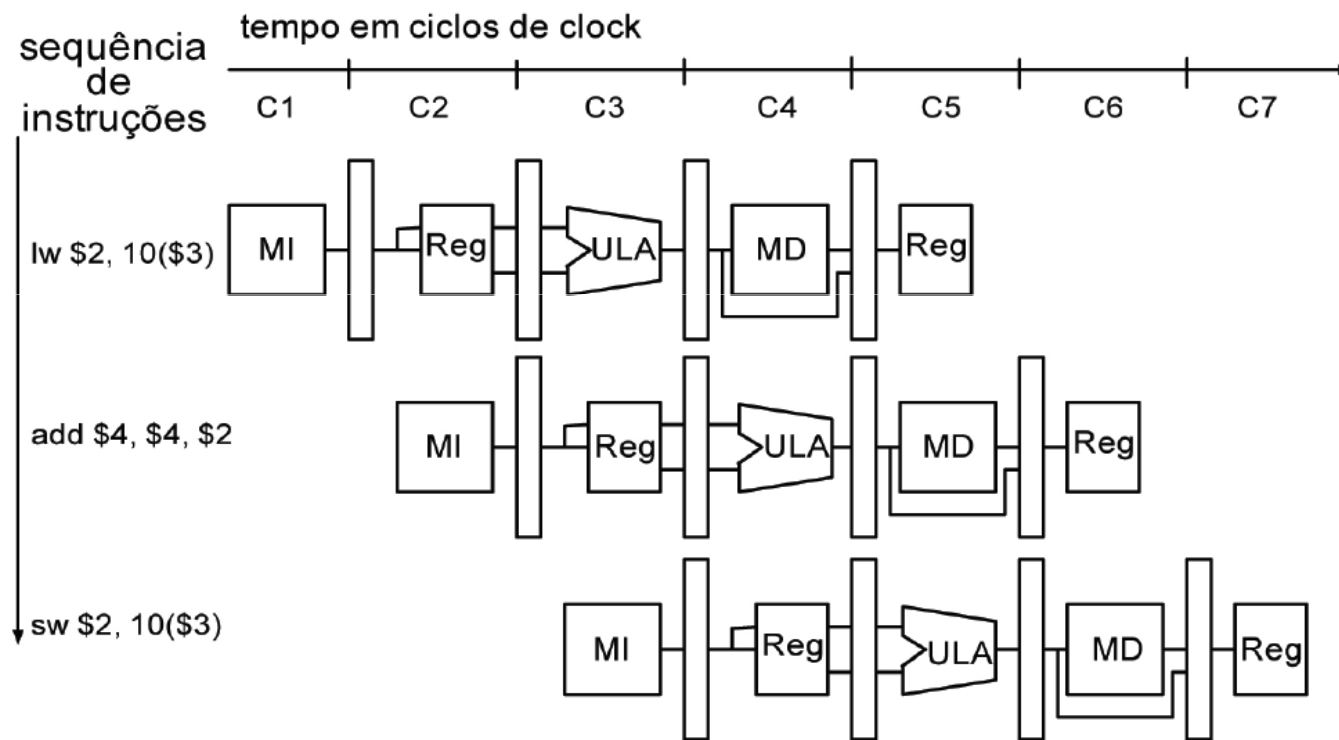
- *Latch ID/EX*: *latch* inserido entre os estágios de decodificação e de execução.
- *Latch EX/M*: *latch* inserido entre os estágios de execução e de memória de dados.
- *Latch M/WB*: *latch* inserido entre os estágios de memória e escrita no registrador (WB, *write back*).



Observações

- PC: atualizado a cada ciclo para a busca da próxima instrução sequencial usando $PC + 4$.
- Quando há uma instrução de desvio, tal endereço é calculado no estágio EX e fornecido ao multiplexador de desvio.
- Todas as instruções passam por todos os estágios do *pipeline*.

Diagrama de Tempo do Processamento no MIPS *Pipeline*



Obs.: todas as instruções usam um mesmo número de ciclos.



Controle do *Pipeline* no MIPS

- Organização diferente dos MIPS monociclo e multiciclo.
- Num determinado instante podem existir 5 instruções, cada qual necessitando de sinais de controle específicos.

Geração e Transmissão de Sinais de Controle

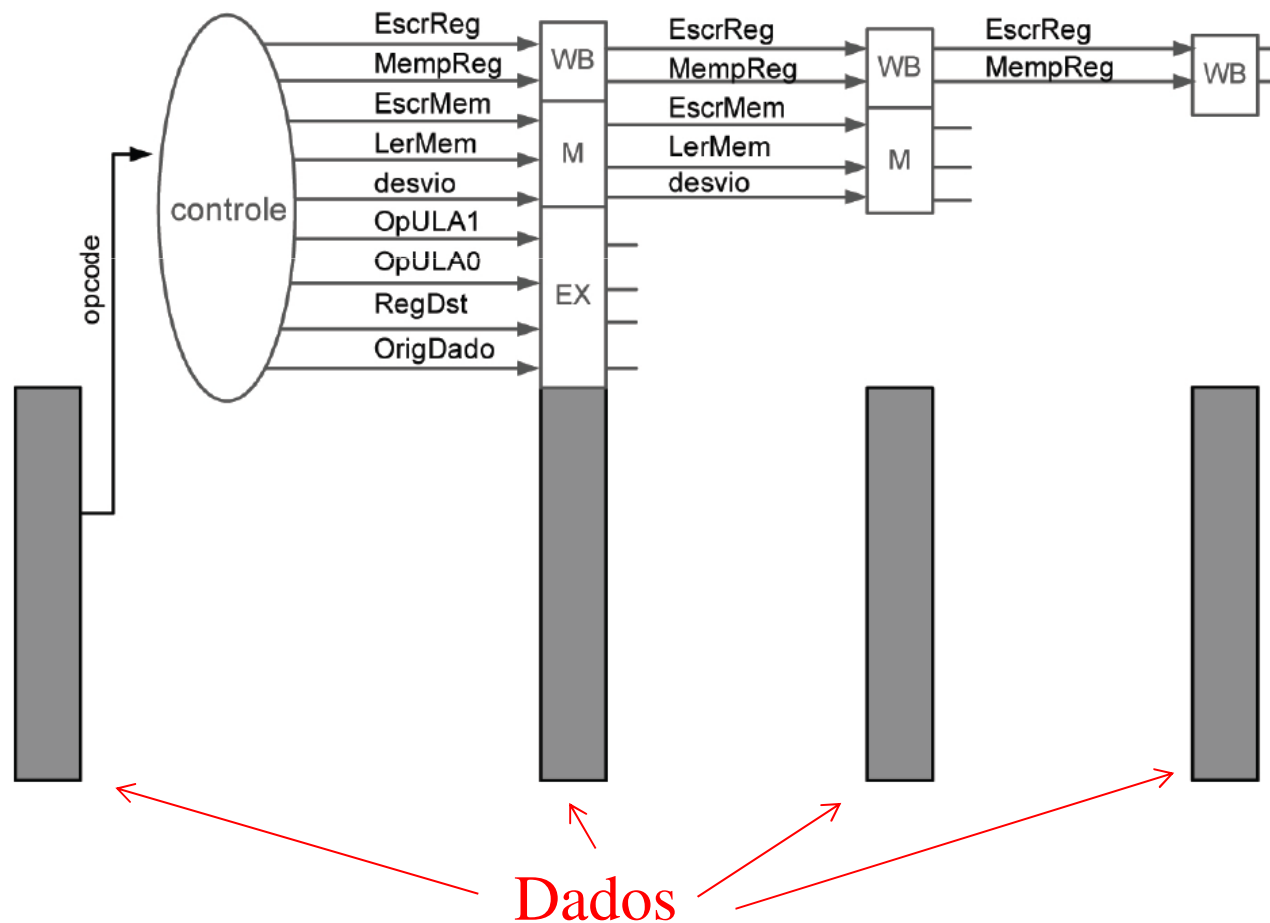
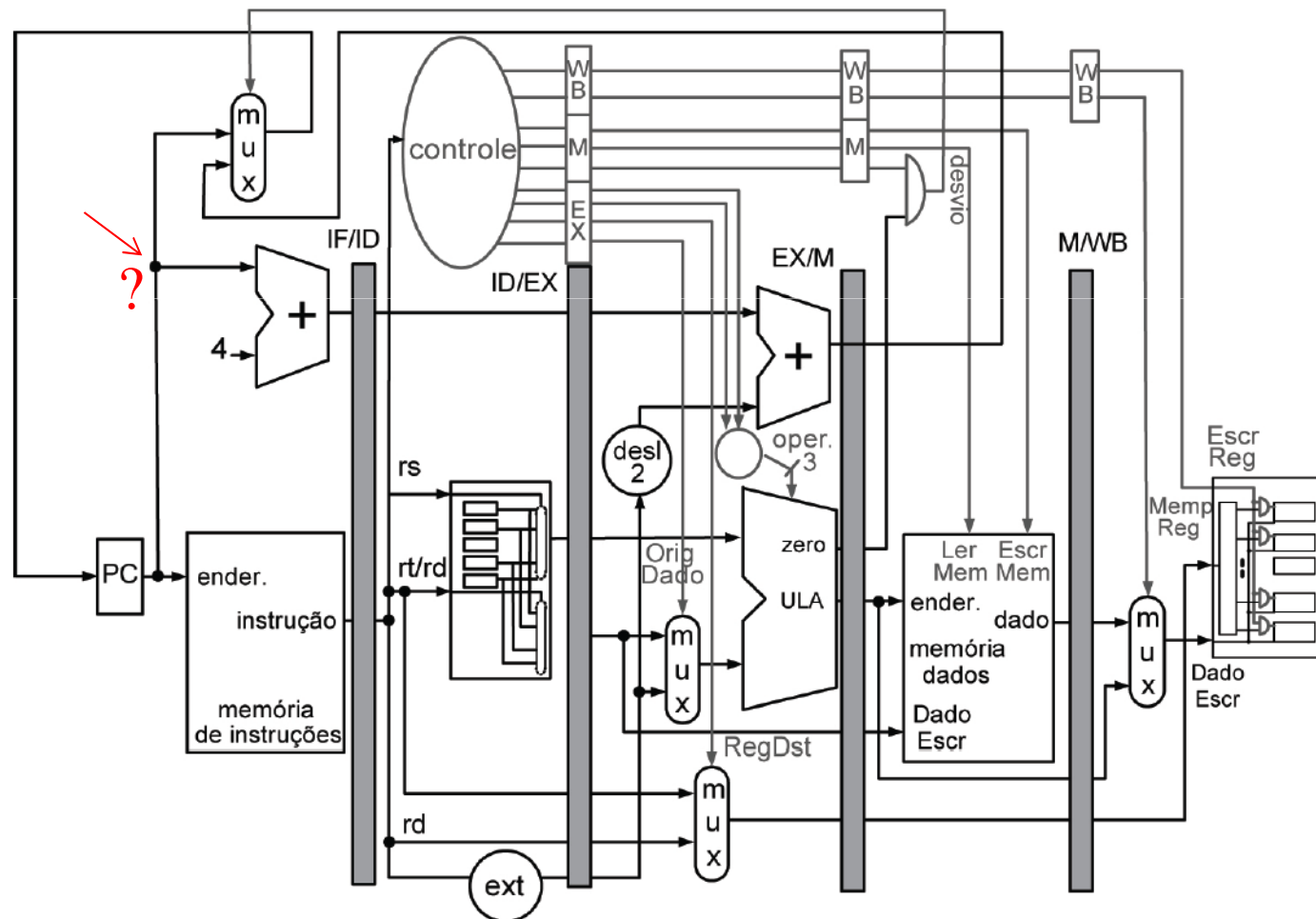


Diagrama do MIPS *Pipeline* Completo

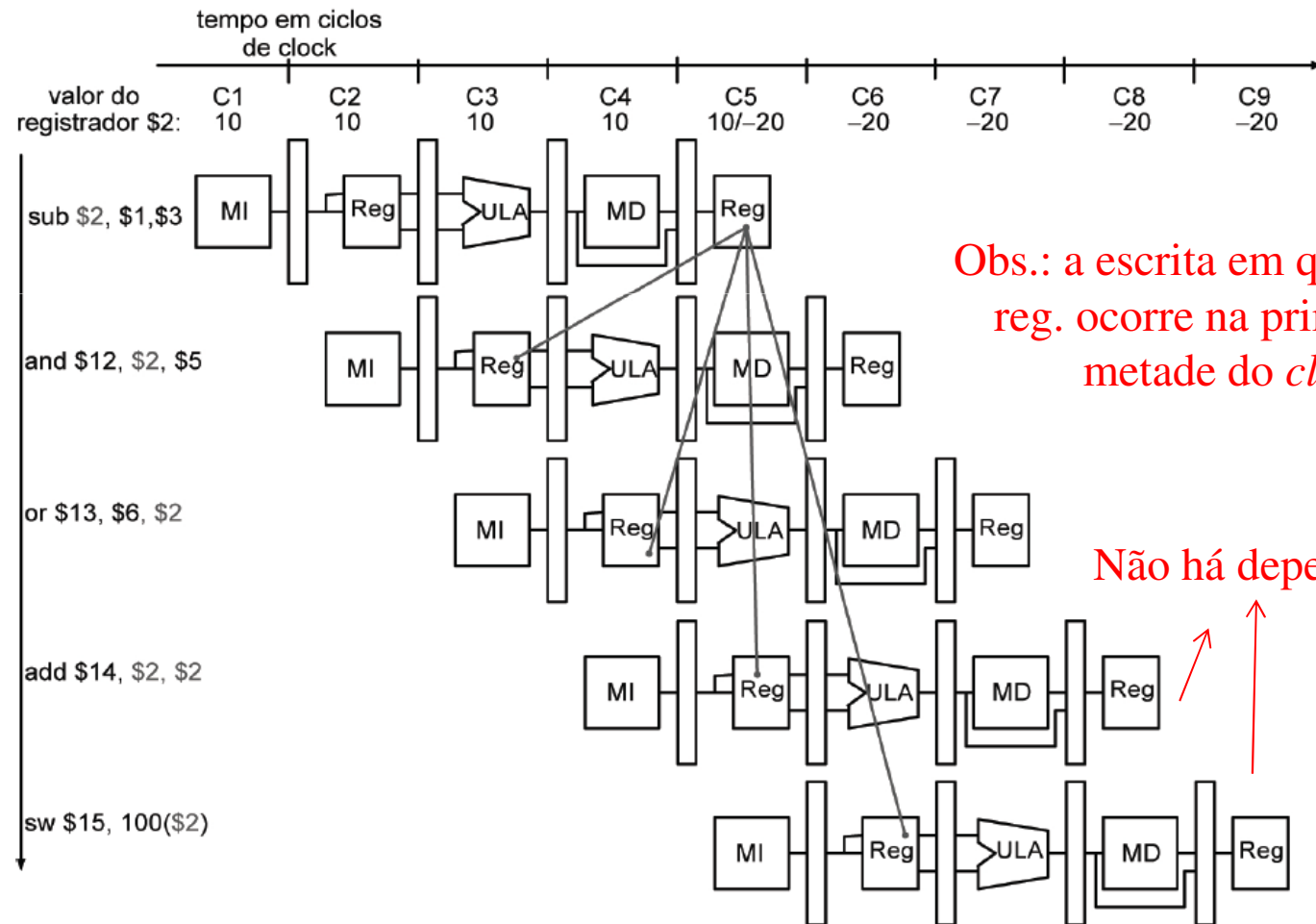




Problemas Relacionados à Sobreposição de Instruções em *Pipeline*

- *Hazards* (Riscos).
- Primeiro Problema (dependência de dados): uma instrução escreve num registrador que é usado por instruções seguintes.
- Escrita: ocorre no último estágio do *pipeline*. Assim, se a próxima instrução depender desse registrador para leitura, o mesmo ainda não estará escrito.

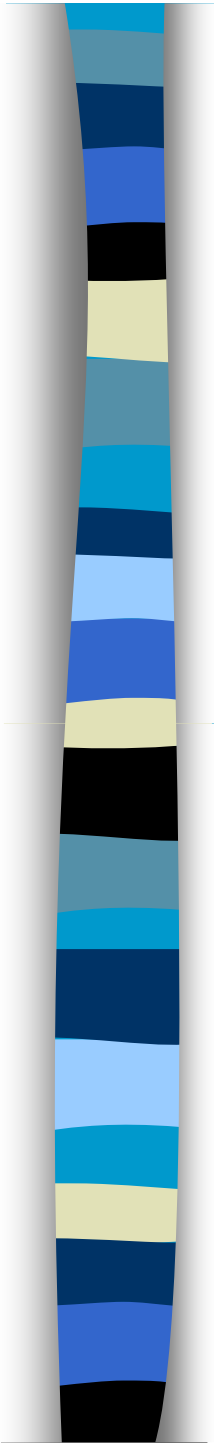
Exemplo: Dependência de Dados no Pipeline





Solução do Problema usando Antecipação

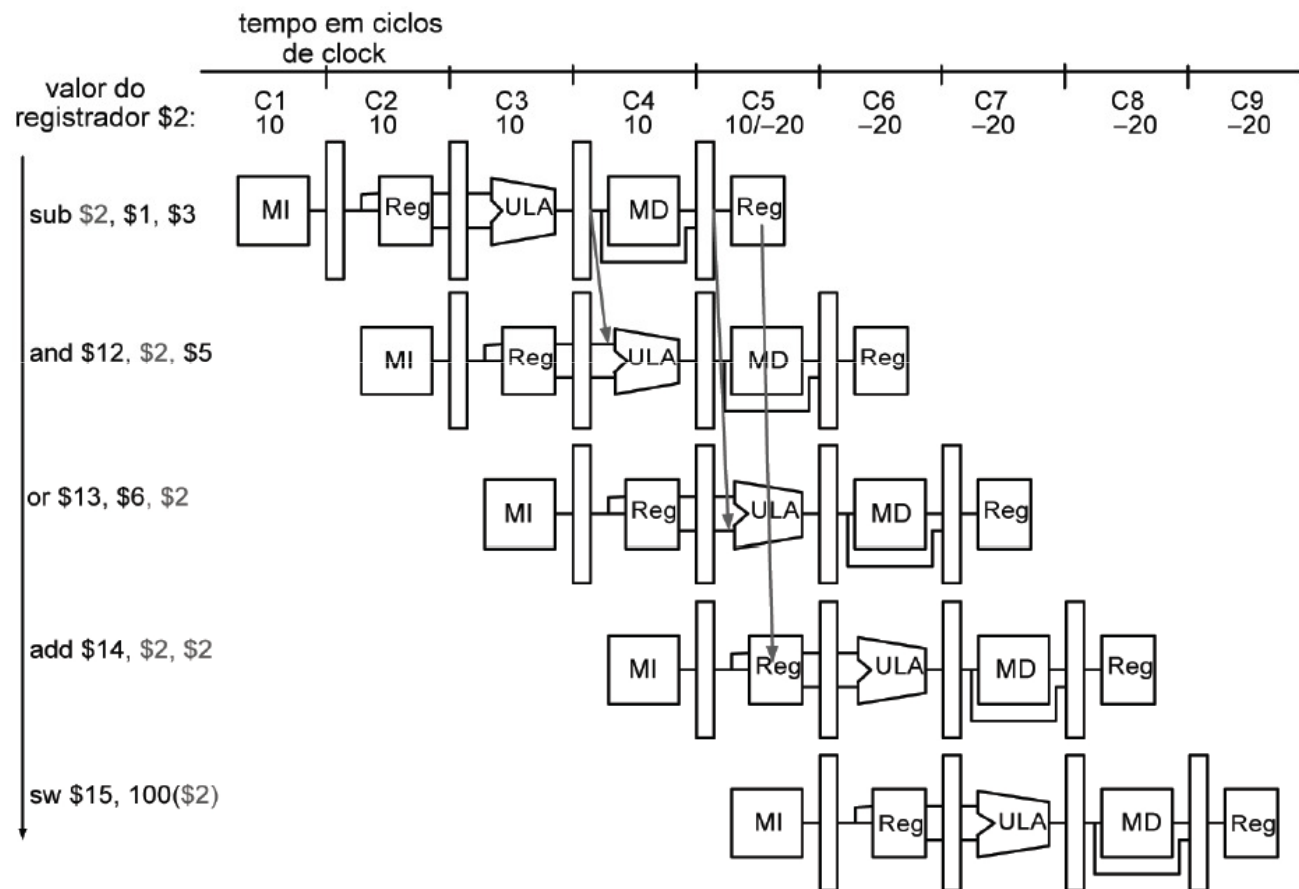
- **Solução:** através de um **circuito de antecipação de dados.**
- **Obs.:** apesar de o registrador de destino ainda não estar escrito no tempo em que a instrução posterior necessita do dado, este já está calculado pela ULA.



Solução do Problema usando Antecipação (ver exemplo anterior)

- Antecipação do dado -> saída da ULA (em relação à *sub*) -> entrada da ULA (em relação à *and*).
- Antecipação do dado -> saída do estágio de memória (em relação à *sub*) -> entrada da ULA (em relação à terceira instrução).
- Obs.: para a quarta instrução não existe necessidade de antecipação.

Exemplo





Obrigações do Circuito de Antecipação

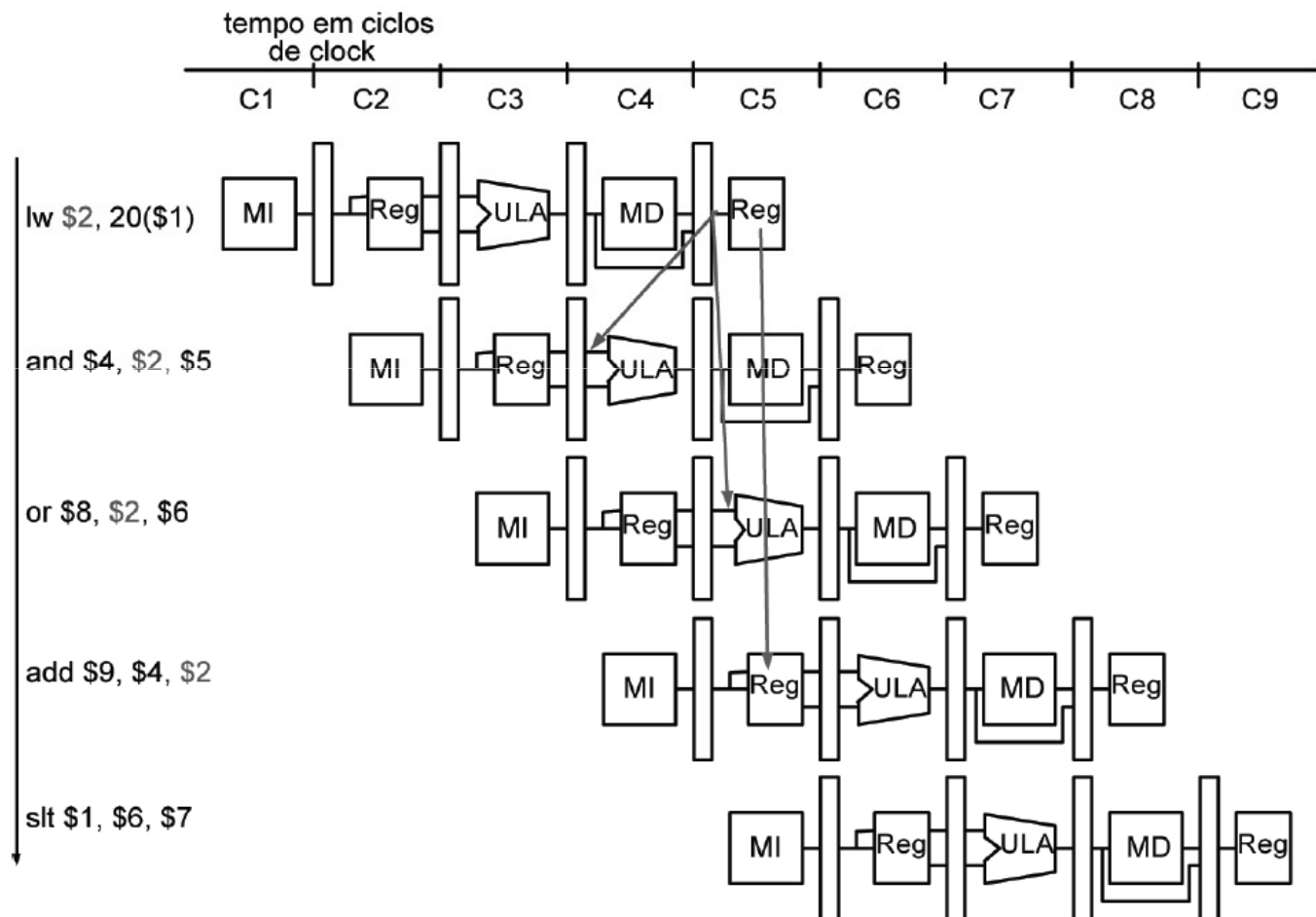
- Interceptação dos dados -> ocorre no estágio de execução.
- *Verificar se uma ou ambas as instruções anteriores escrevem num *reg*, percebendo os sinais de controle de *EscrReg* respectivos.
- **Se * afirmativa, verificar se há coincidência entre o *RegDst* a ser escrito e o número do *reg* usado em *EX*.
- Se ** afirmativa, o referido dado deve ser antecipado através de multiplexadores nas entradas da ULA.



Dependência de Dado Escrito por *load-word*

- Nem sempre é possível antecipar um dado através do circuito de antecipação.
- Instrução *load-word*: **dado** disponível apenas no **estágio M**. Se a instrução subsequente depender deste, não há como antecipá-lo para a entrada da ULA.

Exemplo

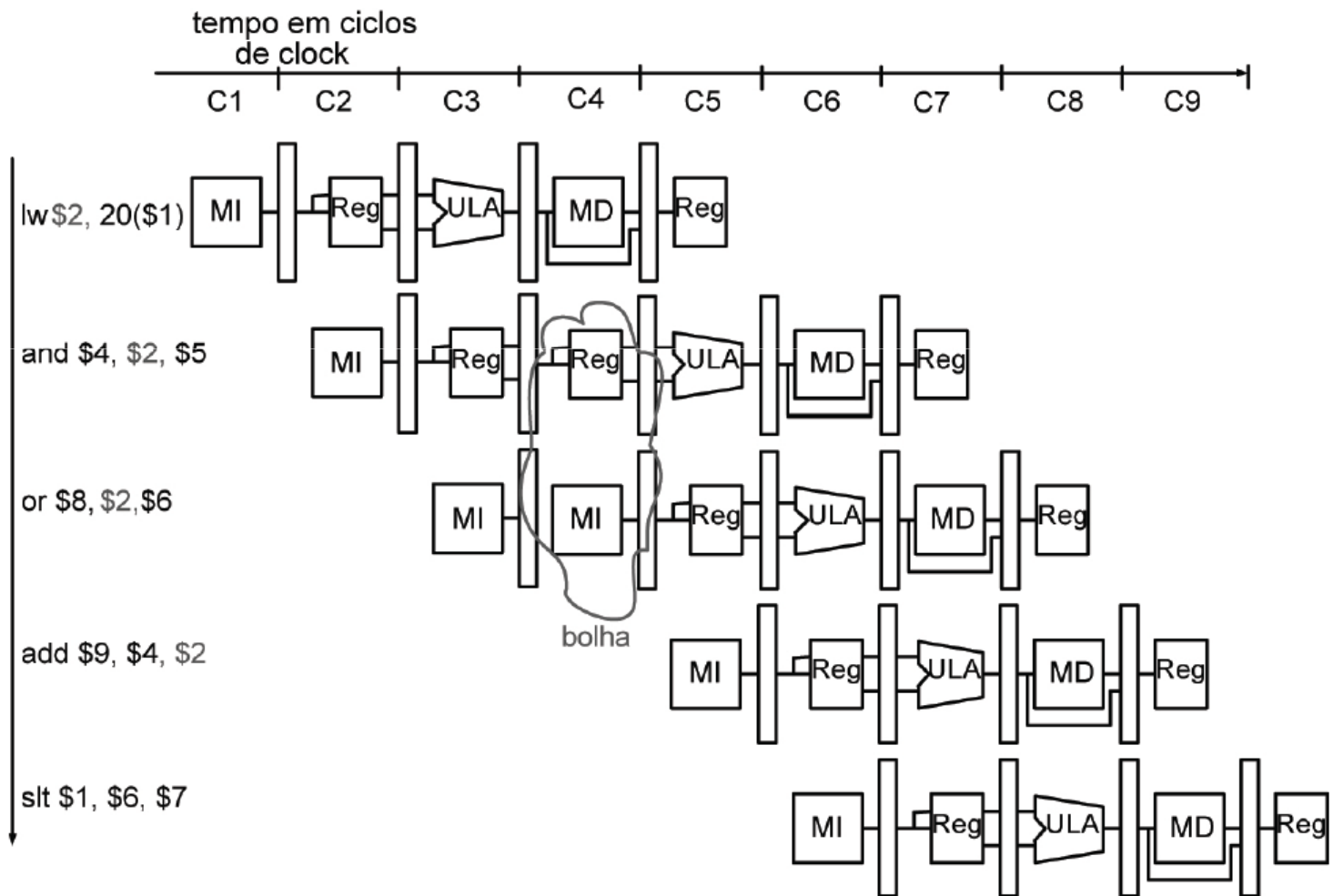


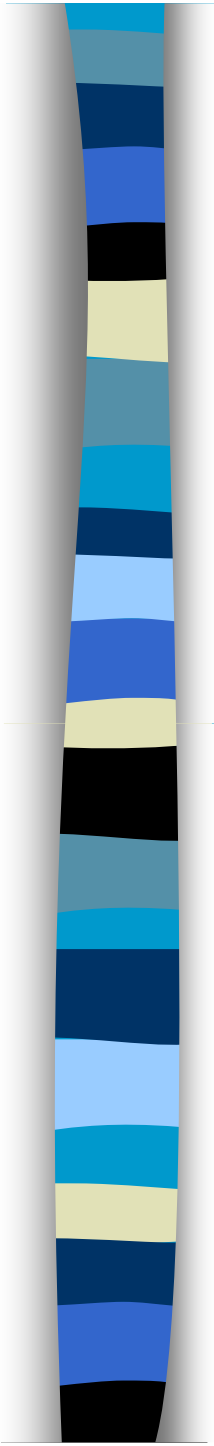


Solução

- Paralisar um ciclo (*stall*), as instruções seguintes à *load-word*.
- A **parada** corresponde à **inserção de uma “bolha”** no diagrama, **repetindo os mesmos estágios para as duas instruções**. Assim, é possível antecipar os dados após a leitura da memória.

Exemplo

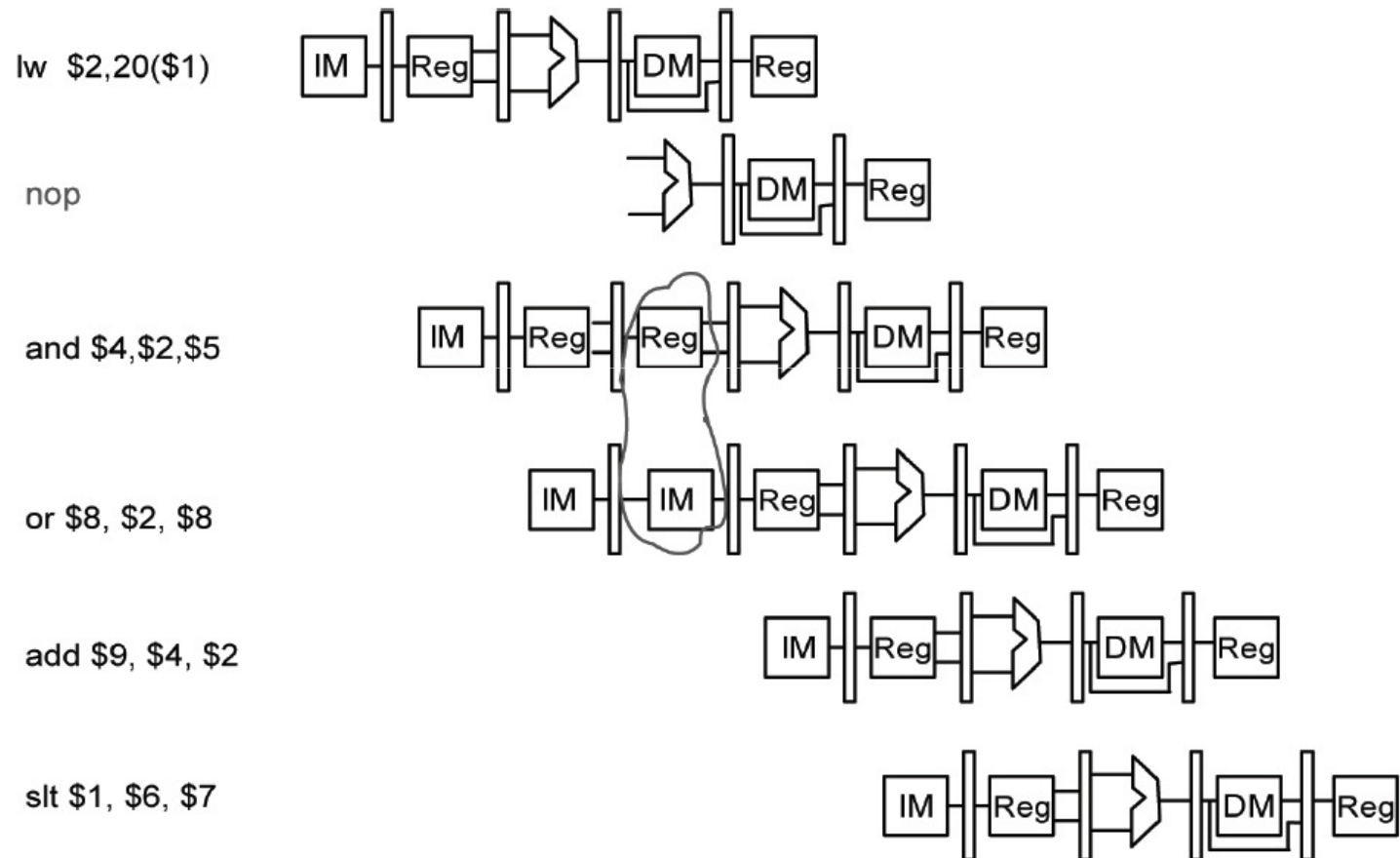




Solução usando nop (*no operation*)

- Quando se faz uma inserção de parada, é necessário zerar os sinais de controle para o estágio subsequente à instrução *load-word*, para garantir que nada será feito nos estágios logo após a instrução.

Exemplo

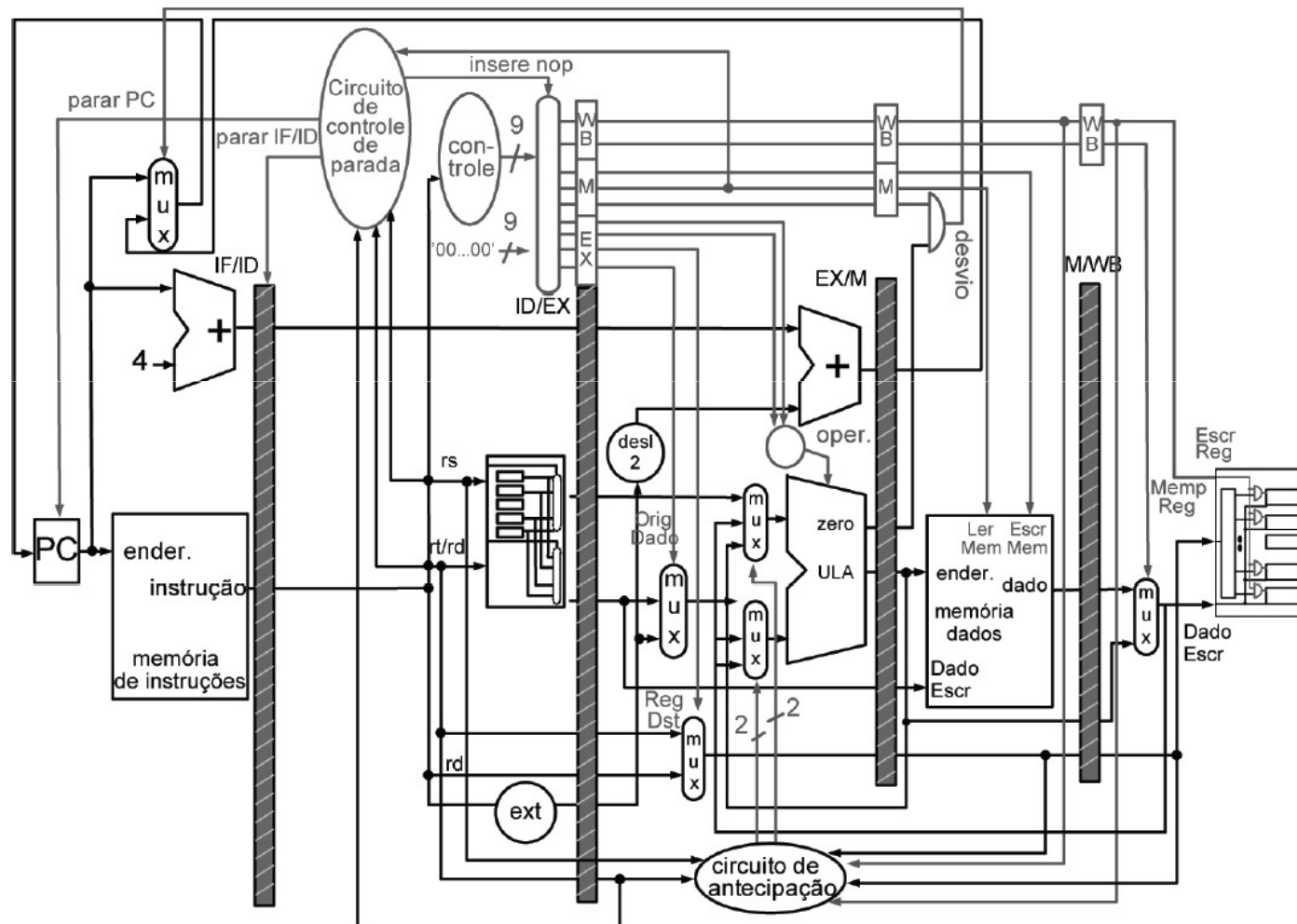




Observações

- **Detecção do problema** -> feita no **estágio ID**, onde a instrução subsequente se encontra quanto a instrução *load-word* está em EX.
- **LerMem ativo em EX** -> instrução *load-word*.
- O **circuito de controle de parada verifica** se um dos **regs em ID coincide** com o **reg destino, caso afirmativo**, esse circuito se encarrega de **parar os estágios IF e ID**, além de **inserir os sinais de controle '00...00'** através de um MUX após o circuito de controle para inserir *nop* no estágio subsequente.

Circuito de Controle de Parada





Exercícios (Parte 1)

- 1. Ana, Beto, Catarina e Davi possuem roupas sujas a serem lavadas, secadas, passadas e guardadas. O lavador, o secador, o passador e o guardador levam 30 minutos em sua tarefa. Quanto tempo levará a lavagem sequencial para as 4 trouxas de roupas? Quanto tempo levará a lavagem com *pipeline*?



Exercícios (Parte 1)

- 2. Verifique se as afirmações a seguir são verdadeiras ou falsas.
 - A técnica de *pipelining* melhora a vazão do sistema de lavanderia sem melhorar o tempo para concluir uma única trouxa de roupa. Logo, a melhoria na vazão diminui o tempo para concluir o trabalho.
 - Tudo está trabalhando em paralelo, de modo que mais trouxas são terminadas por hora.



Exercícios (Parte 1)

- 3. Prove que o ganho de velocidade devido à técnica de *pipelining* será igual ao número de estágios do *pipeline* no exemplo anterior! Dica: utilize um número grande de trouxas para provar!



Exercícios (Parte 1)

- 4. Dar exemplos de *Hazards* Estruturais.



Exercícios (Parte 1)

- 5. Mostre como seria a representação gráfica de um *pipeline* de 5 estágios para resolver o problema a seguir.
 - add \$s0,\$t0,\$t1
 - sub \$t2,\$s0,\$t3

Exercícios (Parte 1)

- 6. Considere o código MIPS dado a seguir:

```
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)
```

- Mostre como reordenar as instruções dadas para evitar os “hazards” dos *adds* no segmento de código dado. Despreze os demais “hazards”.