

Fonte: <http://www.techspot.com/article/904-history-of-the-personal-computer-part-5/>

Datapath e Controle Pipelined para MIPS

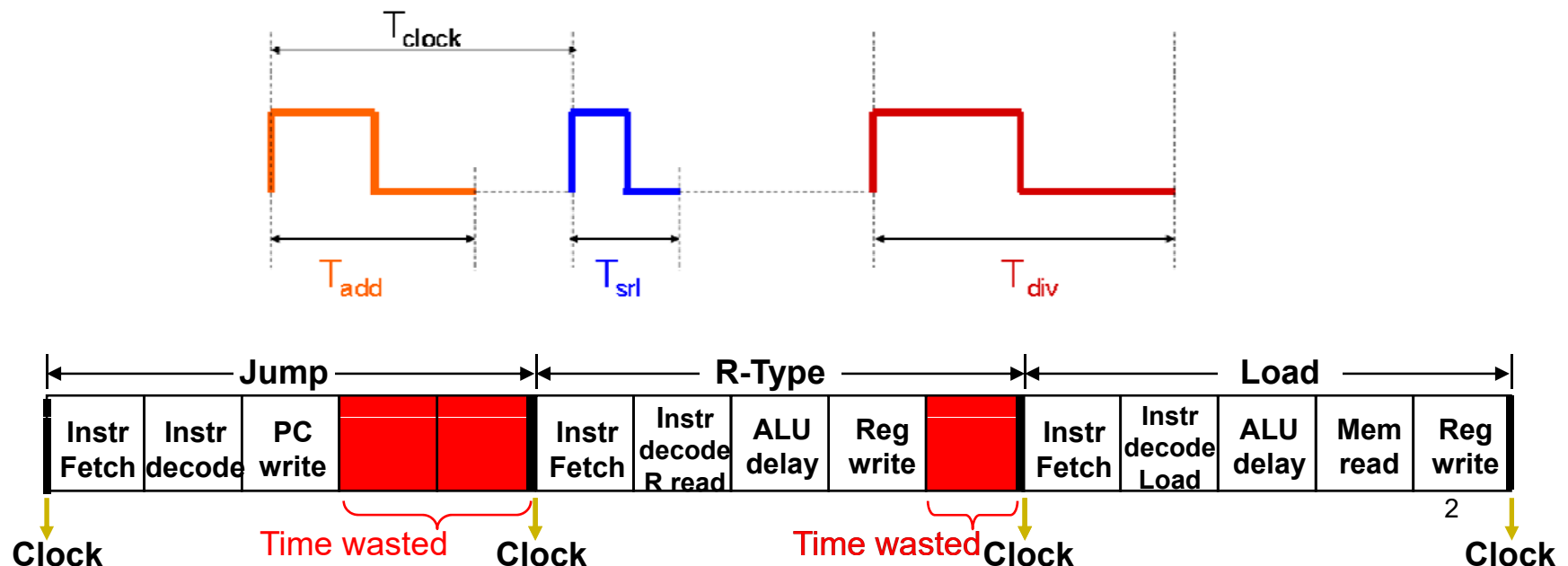
Luciano de Oliveira Neris

luciano@dc.ufscar.br

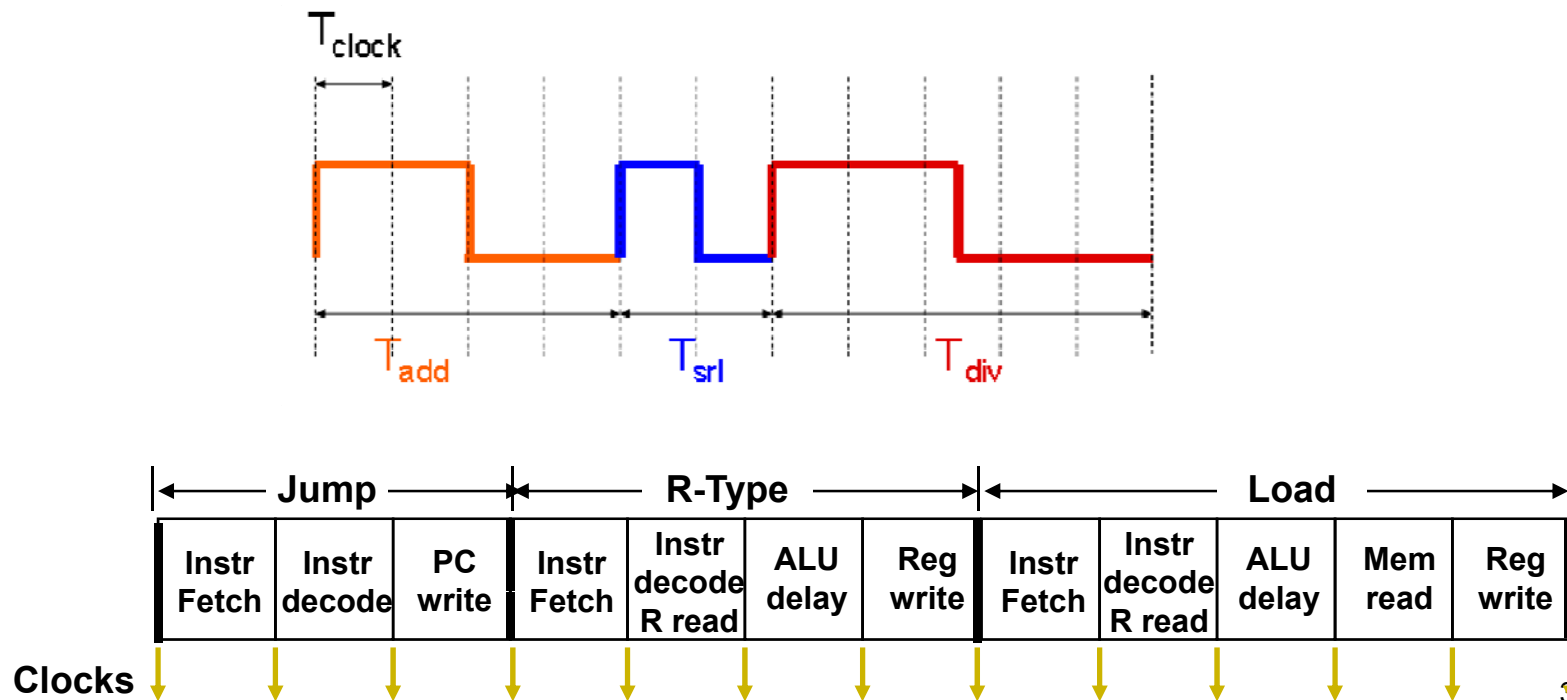
Adaptado de slides do prof. Marcio Merino Fernandes

Datapath Monociclo

- Cada instrução é executada em um 1 ciclo de clock
- Ciclo de clock deve ser longo o suficiente para executar a instrução mais longa
- Desvantagem: velocidade global limitada à velocidade da instrução mais lenta



- Quebra o ciclo de execução em vários passos
- Executa cada passo em um ciclo de clock
- Cada instrução usa apenas o número de ciclos que ela necessita



4

Pipeline

Lavanderia

4 Sacolas de roupa suja - A,B,C,D

Tarefa p/ cada uma delas:

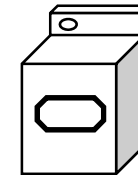
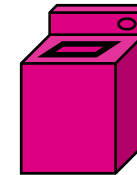
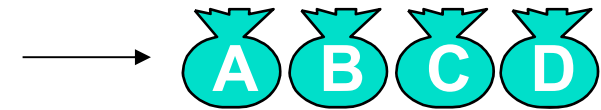
- lavar, secar, passar, guardar

- Lavar gasta 30 minutos

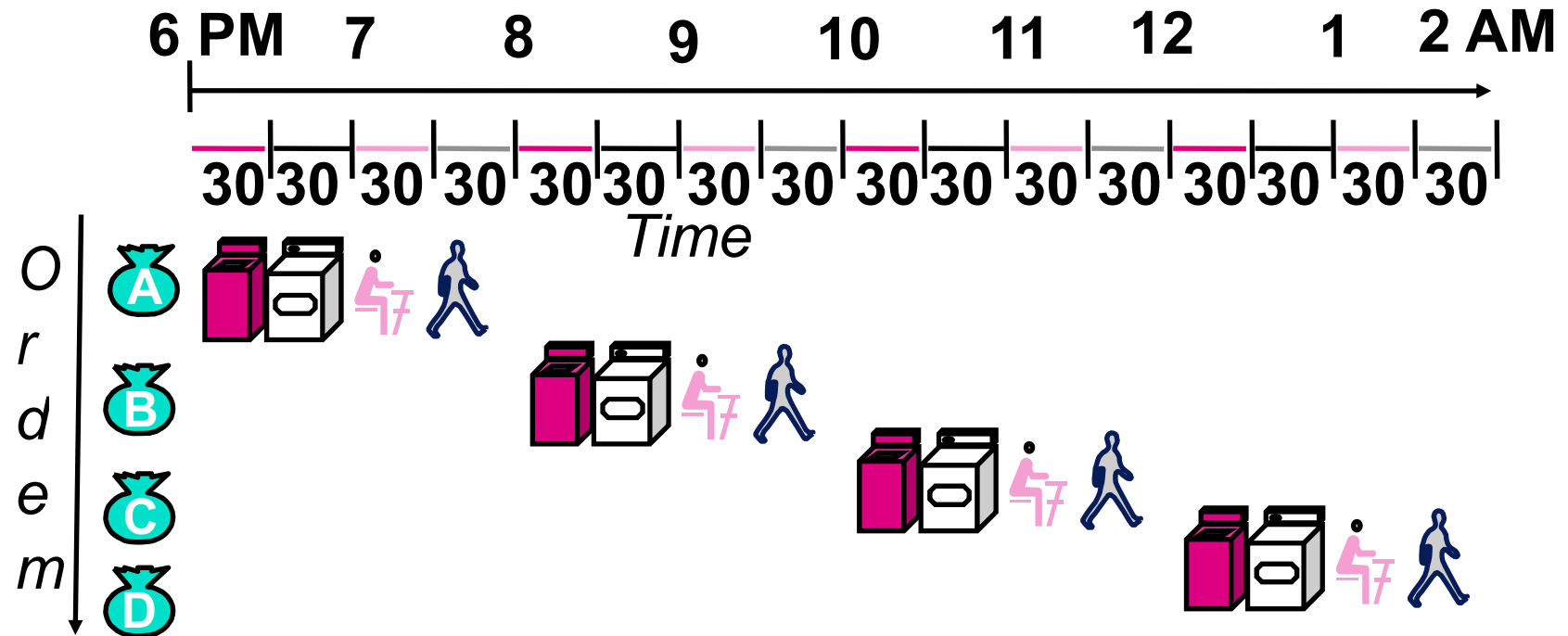
- Secar gasta 30 minutos

- Passar gasta 30 minutos

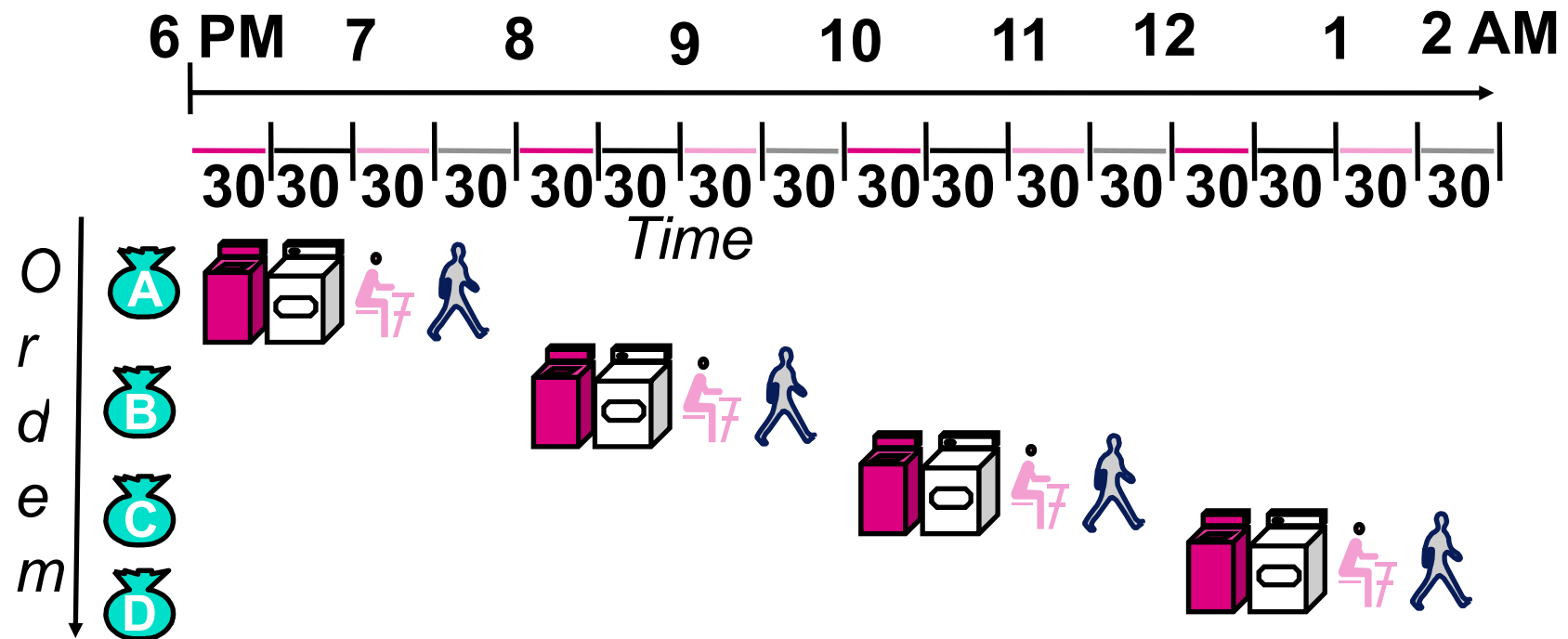
- Guardar gasta 30 minutos



Lavanderia Sequential

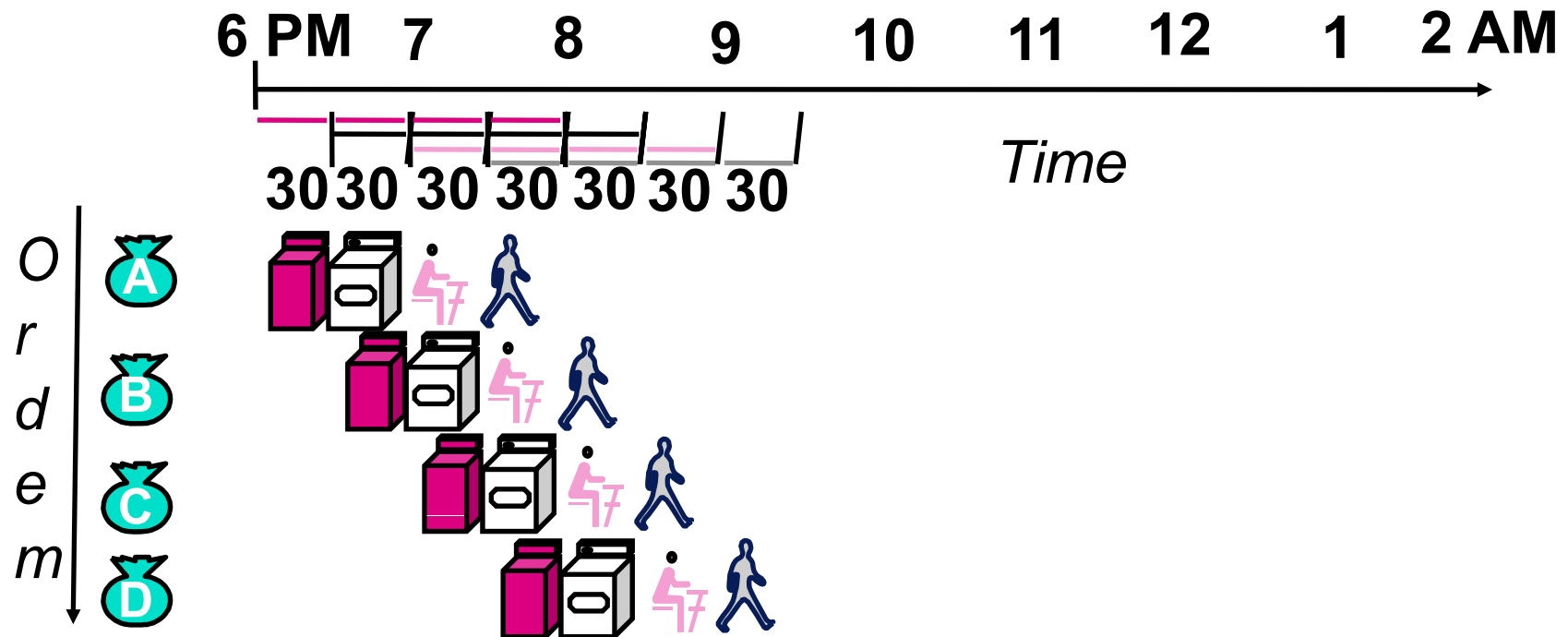


Lavanderia Sequencial

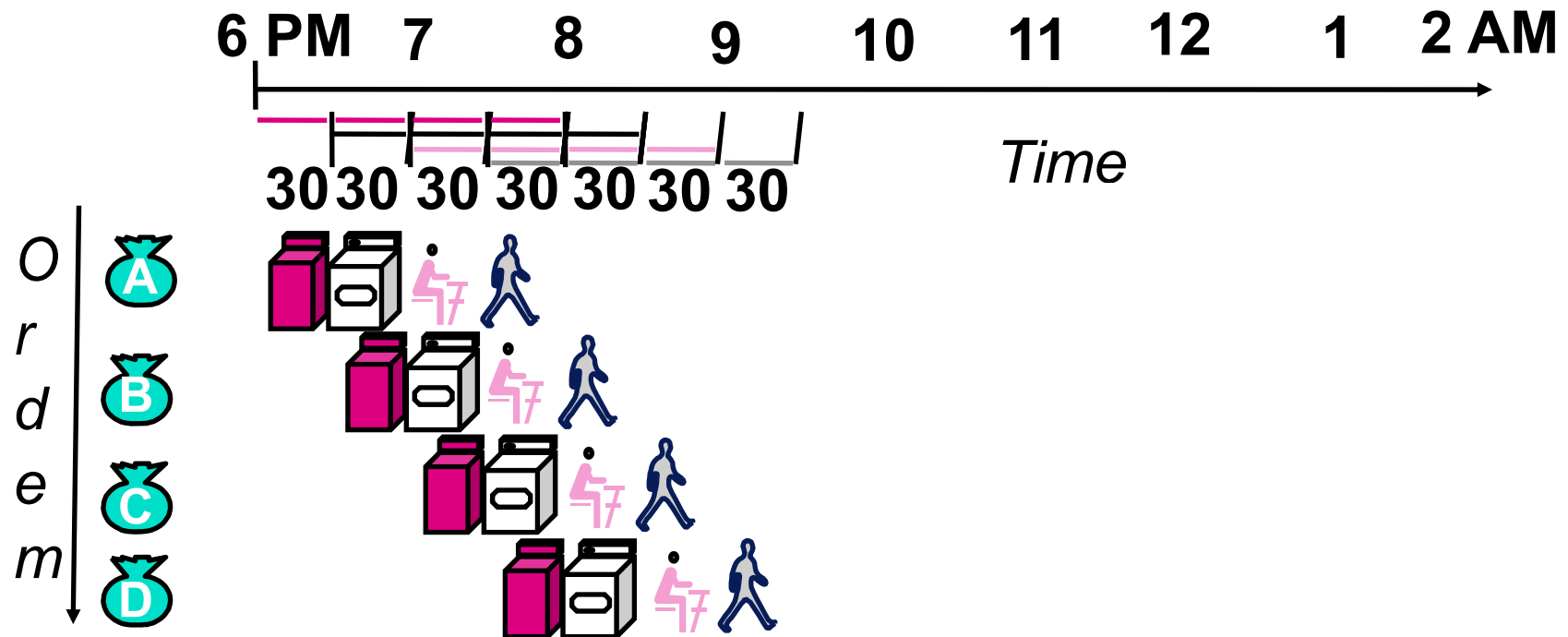


Lavanderia sequencial gasta 8 horas p/ 4 cargas de roupas

Lavanderia em Pipeline

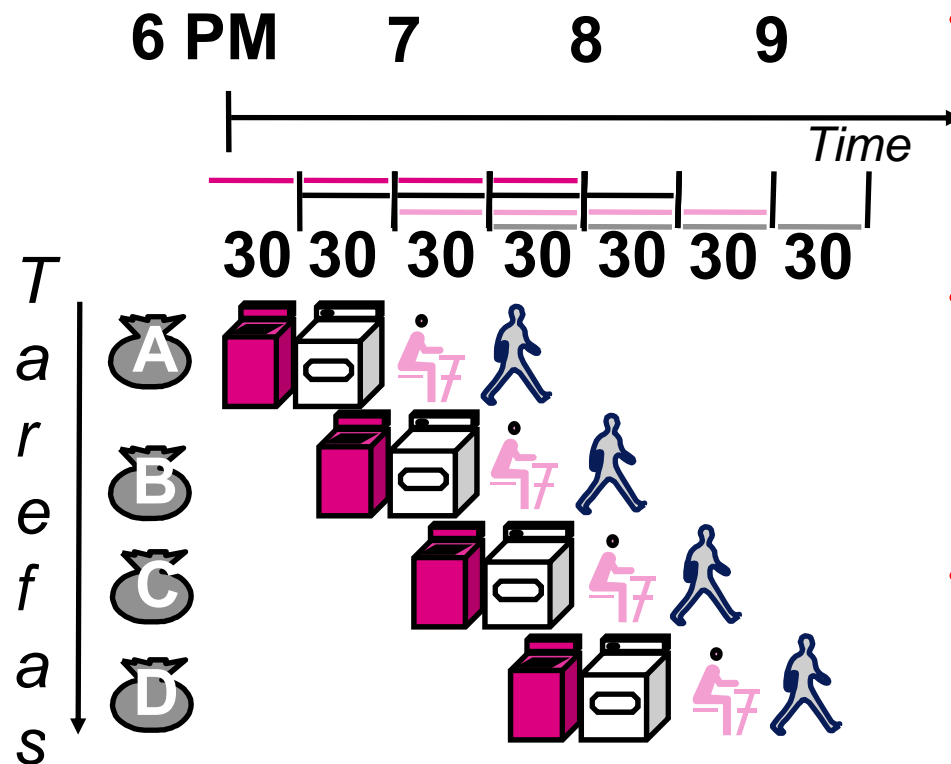


Lavanderia em Pipeline



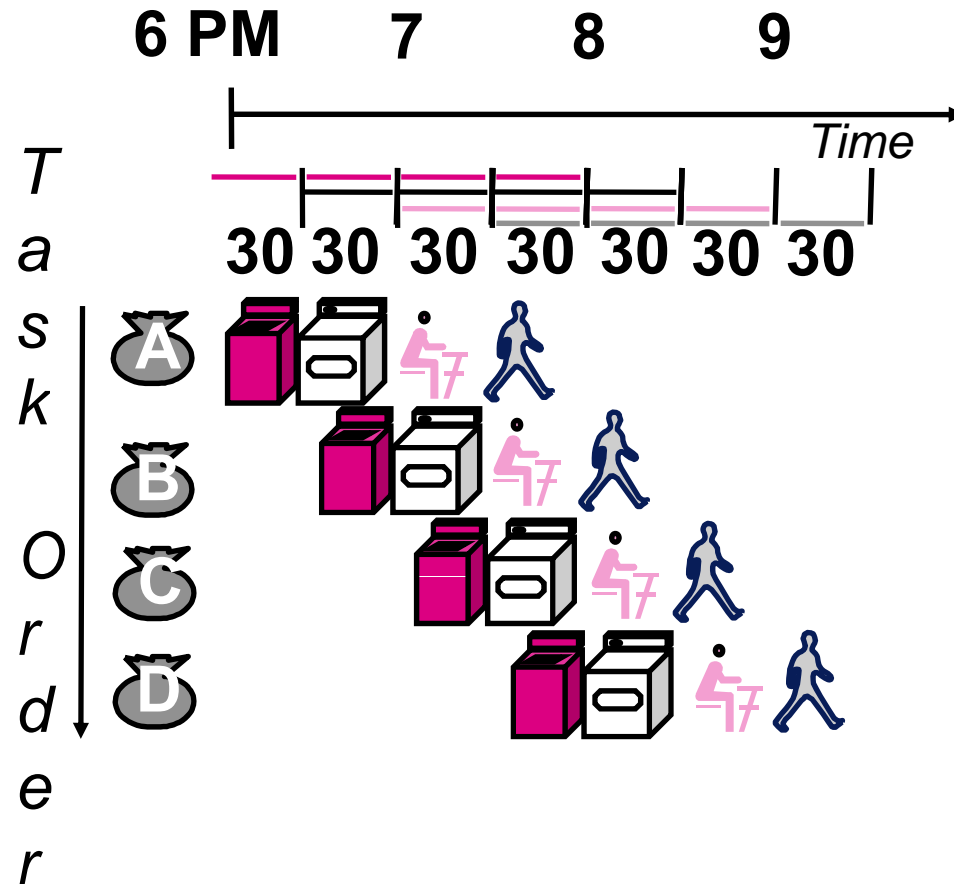
Lavanderia em pipeline gasta 3.5 horas p/ 4 cargas de roupas

Pipeline: Conclusões (1/2)



- Pipelining não melhora a **latência** de uma única tarefa, mas sim a **produção** do trabalho completo.
- **Multiplas** tarefas são executadas simultaneamente, usando **recursos diferentes**.
- Speedup Potencial: = **Número de estágios do pipeline**

Pipeline: Conclusões (2/2)



- Suponha lavar passe a gastar apenas 20 minutos. Quanto melhoraria o pipeline completo?
- Nada - a taxa do pipeline é limitada pelo estágio mais lento.
- O tempo de execução de cada estágio deve ser balanceado.

Pipelining

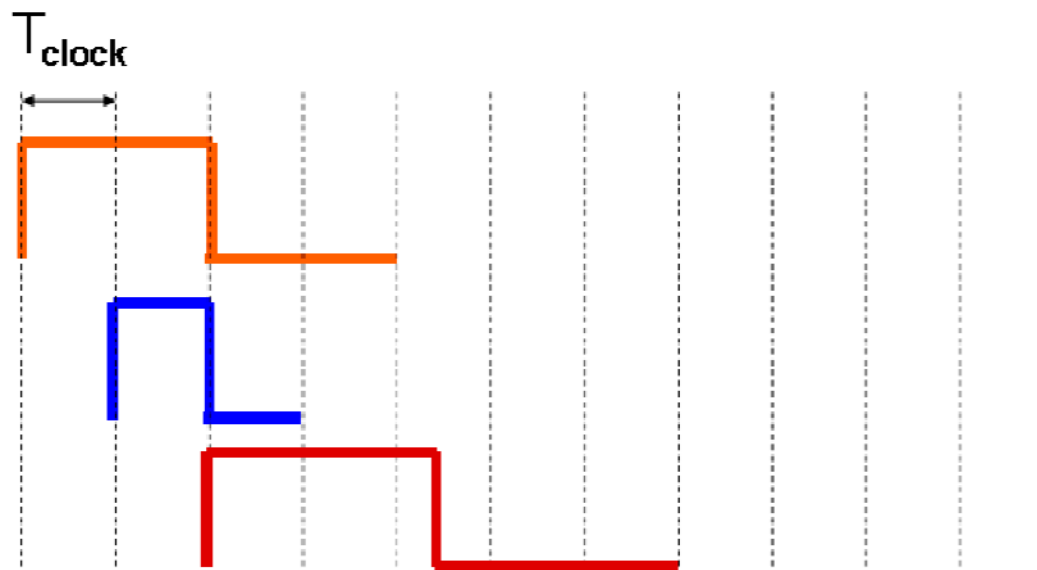
- Tecnologia chave p/ alcançar alto desempenho
- Pipelining é uma técnica que permite a intercalação de estágios diferentes na execução de várias instruções simultaneamente.
 - ▣ O hardware processa mais de uma instrução de cada vez, sem aguardar o término de uma instrução para iniciar a próxima instrução.
- Possível devido aos seguintes fatos:
 - ▣ A execução de uma instrução pode ser subdividida em vários estágios
 - ▣ Duas ou mais instruções podem estar em estágios diferentes

Pipelining

- Pode ser visto como uma "linha de montagem", constituída por vários passos (estágios);
- Cada estágio é conectado ao seguinte, análogo a uma canalização (**cano= pipe**)
- Instruções entram no início da canalização (**pipelining**), e saem no final

Arquitetura tipo Pipelined

- Cada instrução é executada em múltiplos ciclos
- Executa uma etapa de cada instrução em cada ciclo
- Processa múltiplas instruções em paralelo



Desempenho do Pipelining

- Pipelining reduz o **tempo médio** na execução de um conjunto de instruções → **Diminui o CPI médio**
- Pipelining permite a implementação de estágios mais curtos, diminuindo o tempo de ciclo da máquina → **Aumenta o clock rate.**

$$\text{CPU time} = \frac{\text{IC} \times \text{CPI}}{\text{Clockrate}}$$

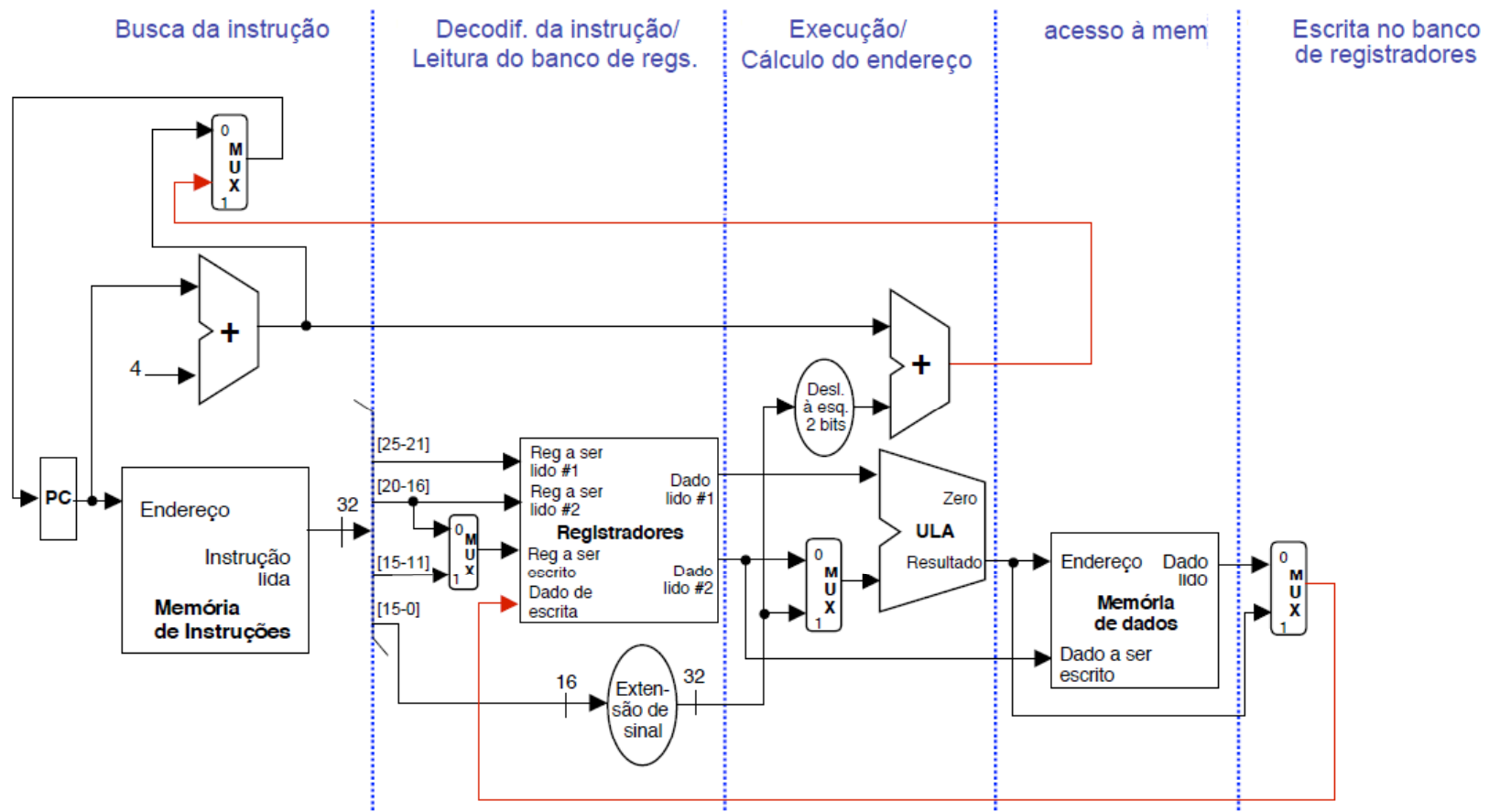
- O uso do Pipelining é totalmente **transparente ao programador/compilador** → Técnica altamente bem sucedida!

Pipelining para MIPS

- Arquiteturas RISC (ex: MIPS) se adequam muito bem à organização em pipelining, devido principalmente a:
 - ▣ Predominância de operandos do tipo registrador
 - ▣ Poucos formatos de instrução, condificação fixa (ex: 32 bits)
- Podemos estender a organização monociclo para operar no modo pipelining.
- Isso pode ser feito acrescentando-se **registadores temporários** entre os diversos estágios do pipeline.

Pipelining para MIPS

□ Monociclo



Pipelining para MIPS

- Solução clássica p/ MIPS: 5 estágios de 1 ciclo cada
 - Instruction fetch (**IF**) (busca)
 - Instruction decode (**ID**) (decodificação)
 - Execution (**EX**) (execução)
 - Memory access (**MEM**) (acesso à memória)
 - Write back (**WB**) (escrita no banco registradores)

IF: Instruction Fetch

- **IF**: Busca a instrução da memória, armazenando no Registrador de Instrução (IR), atualizando o PC p/ PC+4.

ID: Instruction Decode

- **ID**: Envia o OP-CODE p/ a unidade de controle, lê o valor dos registradores determinados na instrução (quando houver).

EX: Execution

- **EX:** Se instrução = load ou store
 - ALU calcula o endereço de memória a ser acessado
- **EX:** Se instrução = R-type (add, sub, etc)
 - ALU executa operação c/ os registradores lidos no estágio ID
- **EX:** Se instrução = branch (beq)
 - ALU efetua o teste (=) entre os dois registradores lidos no estágio ID
- **EX:** Se instrução = jump
 - PC é substituído pelo endereço do desvio
- Além disso, vários sinais de controle são ativados

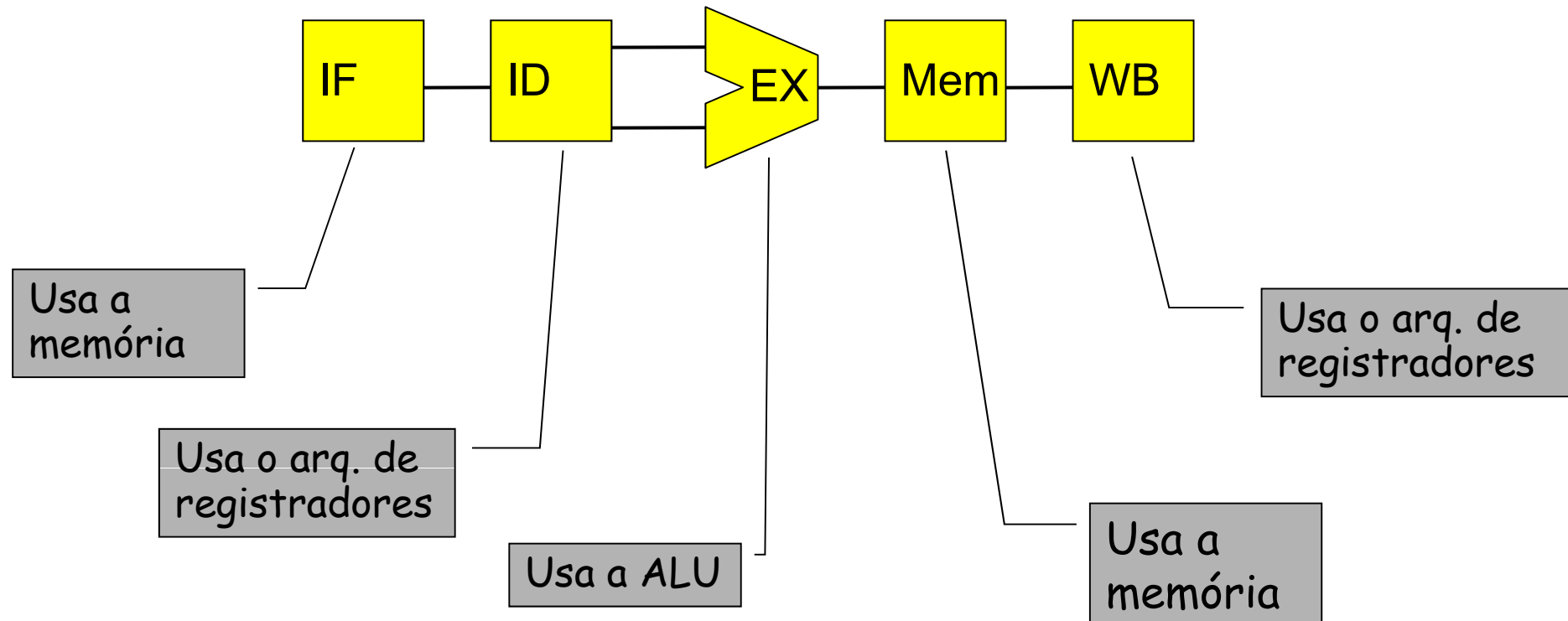
MEM: Memory

- **MEM**: Se instrução = load:
 - Um dado é lido da memória e armazenado em um registrador temporário
- **MEM**: Se instrução = store:
 - Um dado é escrito na memória
- **MEM**: Se instrução = R-type:
 - Armazena o resultado da ALU em um registrador temporário
- Além disso, vários sinais de controle são ativados

WB: Write-back

- **WB:** Armazena no arquivo de registradores resultados eventualmente obtidos em estágios anteriores:
 - Dado lido da memória por uma instrução load
 - Resultados da ALU p/ instruções R-type

Pipelining para MIPS



Pipelining para MIPS

- A organização de um processador em pipeline **não altera** o tempo de execução de uma única instrução, **mas sim** a taxa ou vazão de instruções (**throughput**) que o processador consegue atingir.
- O pipeline aumenta o desempenho por meio do aumento do throughput das instruções, ou seja, aumentando o número de instruções executadas na unidade de tempo (e não por meio da diminuição do tempo de execução de uma instrução individual).
- Throughput: trata-se de uma métrica essencial, uma vez que os programas exigem, de uma maneira geral, a execução de milhões a bilhões de instruções.

Pipelining para MIPS

□ Monociclo x Pipeline

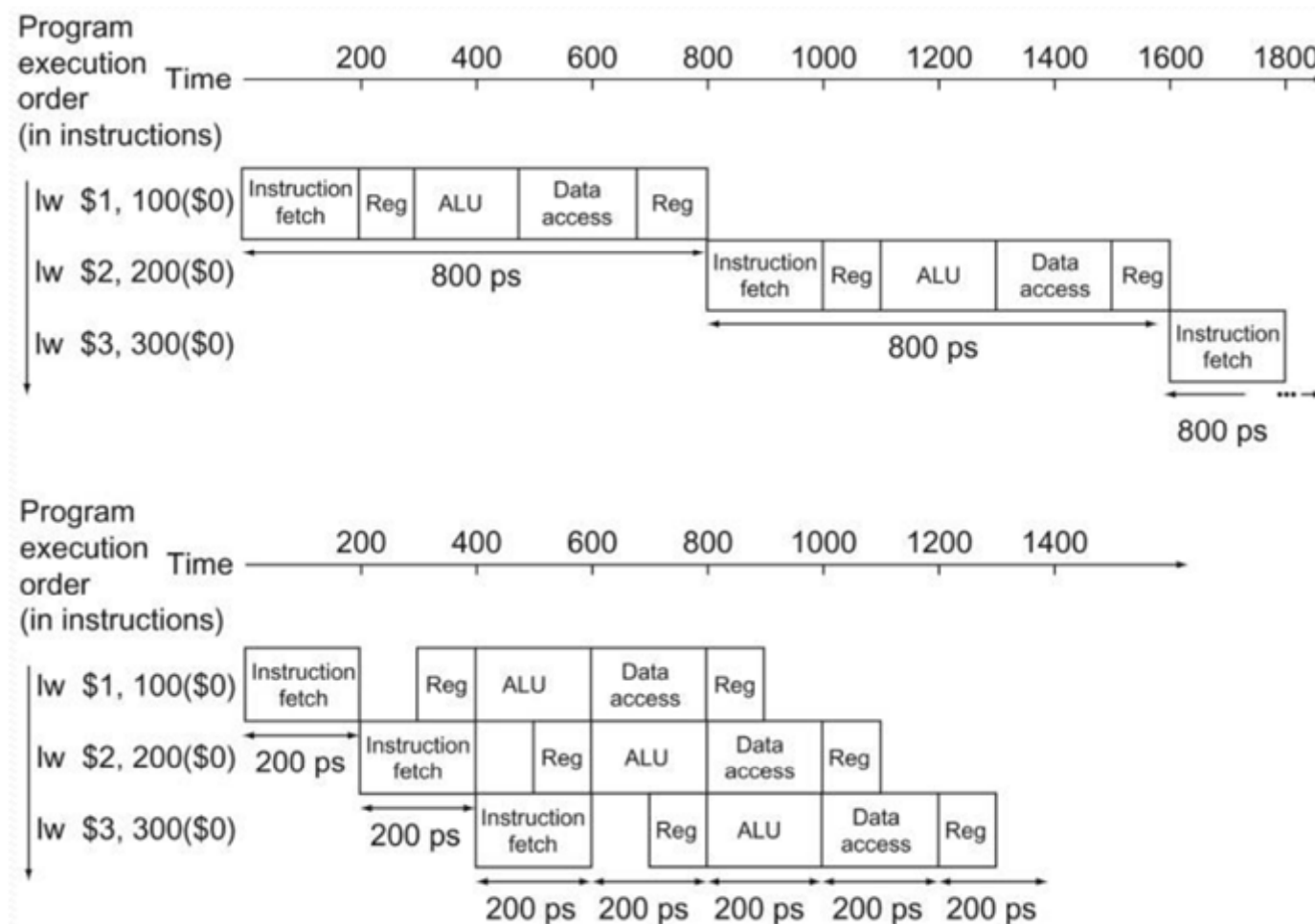
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, AND, OR, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

Pipelining para MIPS

- Monociclo x Pipeline
 - ▣ Monociclo: período mínimo de clock = 800ps
 - ▣ Pipeline: todos os estágios consomem um ciclo.
 - Logo, o período do clock deve ser longo o suficiente para acomodar a operação mais longa (200ps)

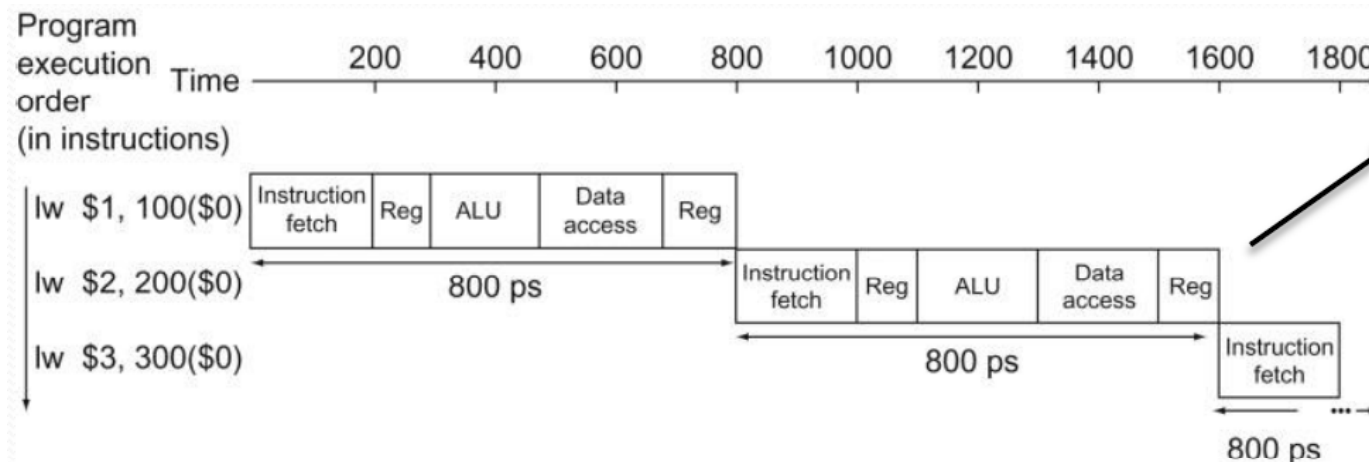
Pipelining para MIPS

□ Monociclo x Pipeline

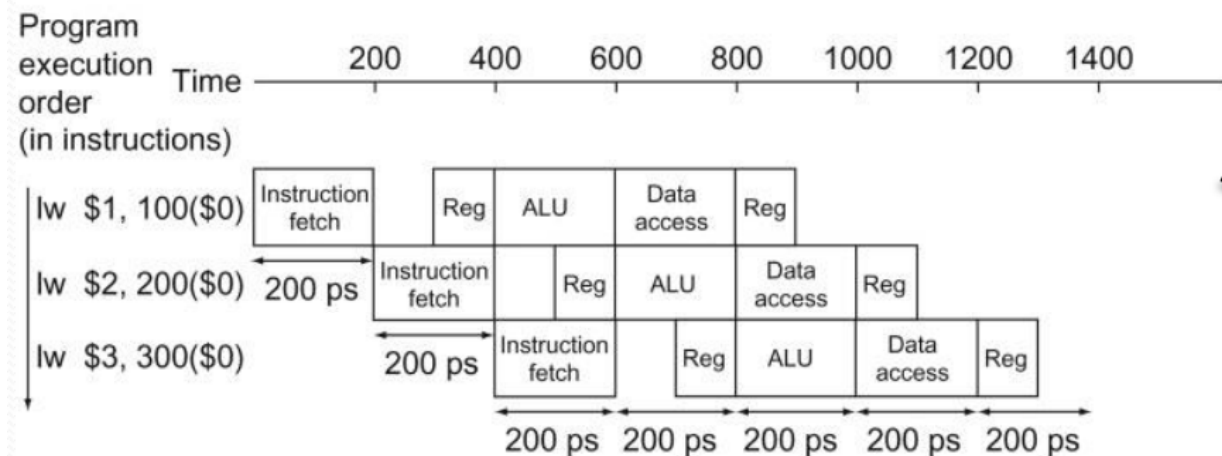


Pipelining para MIPS

□ Monociclo x Pipeline



O tempo entre a primeira instrução *load* e a quarta será de $3 \times 800 \text{ ps} = 2400 \text{ ps}$



Com *pipeline*, a quarta instrução começará a ser executada 600 ps após a primeira

Arquitetura tipo Pipelined

- O conjunto de instruções MIPS foi planejado visando uma implementação "eficiente" com pipeline.
- Todas as instruções possuem o mesmo tamanho (32 bits):
 - Facilita a busca e a decodificação, que podem ser feitas em um ciclo cada (no 1º e 2º estágios da pipeline, respectivamente).
- Os campos dos registradores que fornecem operandos estão localizados no mesmo lugar em cada instrução.
 - Esta simetria abre a possibilidade de o segundo estágio da pipeline iniciar a leitura do arquivo de registrador ao mesmo tempo em que o hardware está identificando a instrução recebida.

Arquitetura tipo Pipelined

- O conjunto de instruções MIPS foi planejado visando uma implementação "eficiente" com pipeline.
- ▣ Operandos em memória somente aparecem nas instruções load e store:
 - Esta restrição de projeto possibilita que o 3º estágio (execução) seja utilizado para o cálculo do endereço de memória e a referida posição seja acessada no estágio seguinte.
 - Se fosse permitida a especificação de operandos em memória, como acontece na arquitetura x86, os estágios 3 e 4 seriam expandidos em: (1) estágio de endereço, (2) estágio de acesso à memória e (3) estágio de execução.

Arquitetura tipo Pipelined

- O conjunto de instruções MIPS foi planejado visando uma implementação "eficiente" com pipeline.
- ▣ Alinhamento dos operandos em memória:
 - No MIPS, as palavras sempre iniciam em endereços de memória que são múltiplos de 4.
 - Por causa desta restrição (ou garantia), um acesso à memória pode ser feito em apenas um estágio.

Pipelining para MIPS - Exemplo

- Dado os 5 estágios anteriores, em regime:
 - ▣ uma nova instrução é iniciada a cada ciclo
 - ▣ uma instrução termina a execução a cada ciclo
 - ▣ logo, a produção (throughput) é proporcional ao estágio mais lento.
 - ▣ Infelizmente existem fatores que dificultam atingir o desempenho máximo (hazards)

Instruction	clock cycle								
	1	2	3	4	5	6	7	8	9
i	IF	ID	EX	MEM	WB				
i+1		IF	ID	EX	MEM	WB			
i+2			IF	ID	EX	MEM	WB		
i+3				IF	ID	EX	MEM	WB	
i+4					IF	ID	EX	MEM	WB

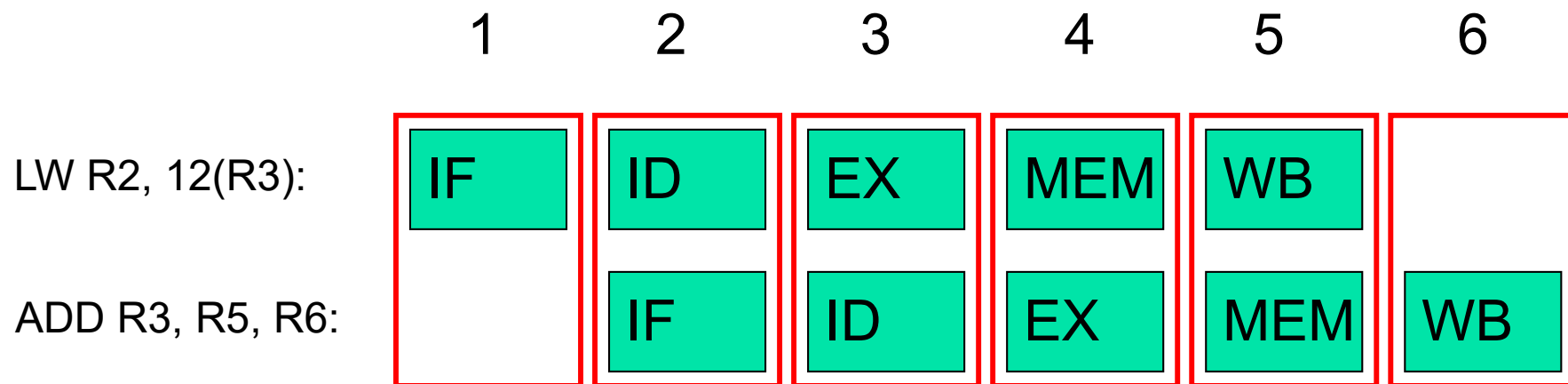
Exemplo

- Considere o seguinte código (instruções I1 e I2)

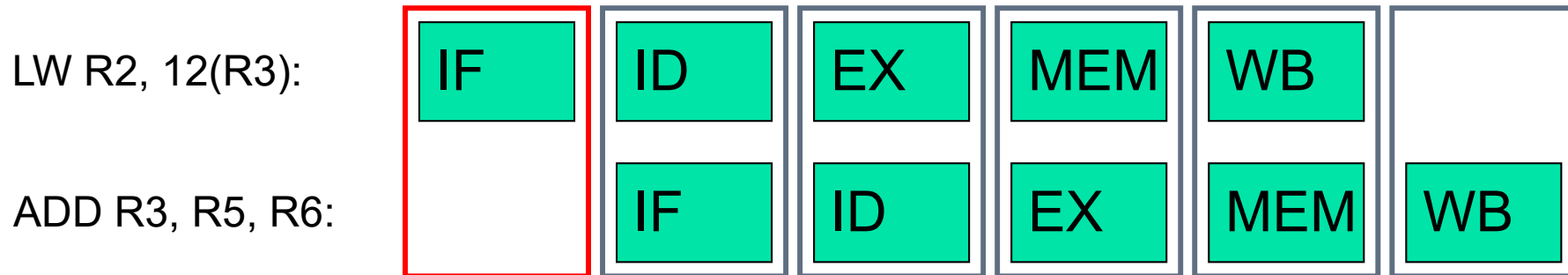
I1: LW R2, 12(R3)

I2: ADD R4, R5, R6

- Execução total= 6 estágios



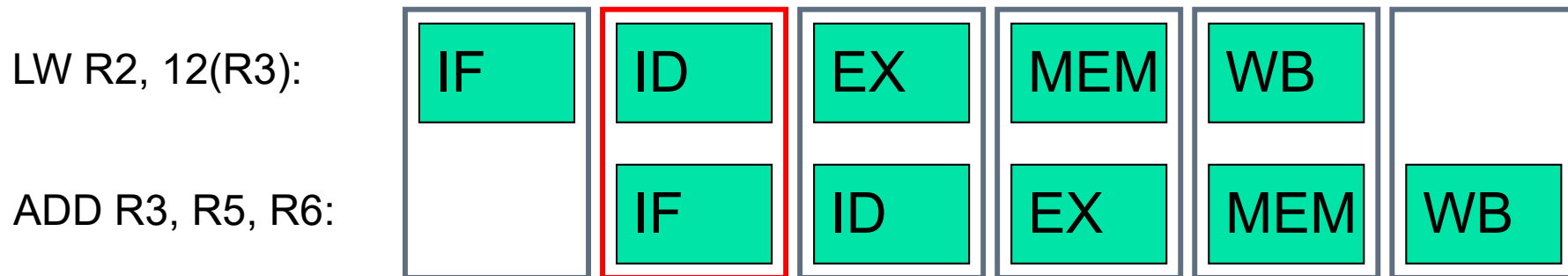
Exemplo



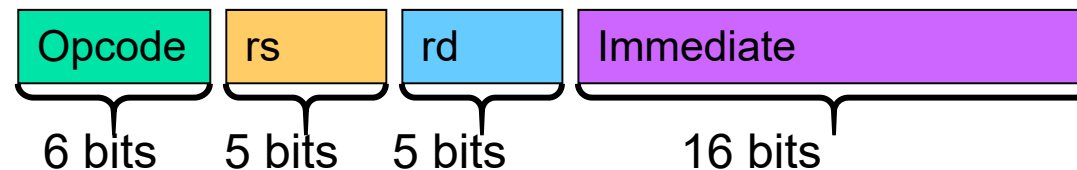
- **Passo 1:** I1 em IF, I2 não emitida
 - ▣ Envia o PC p/ memória para buscar I1
 - Obtém I1 um ciclo depois
 - ▣ $PC = PC + 4$ (shift-left 2)
 - Atualizado no mesmo ciclo

Issued= emitada (enviada para execução)

Exemplo



□ Segundo passo: I1 em ID, I2 em IF



□ I1:

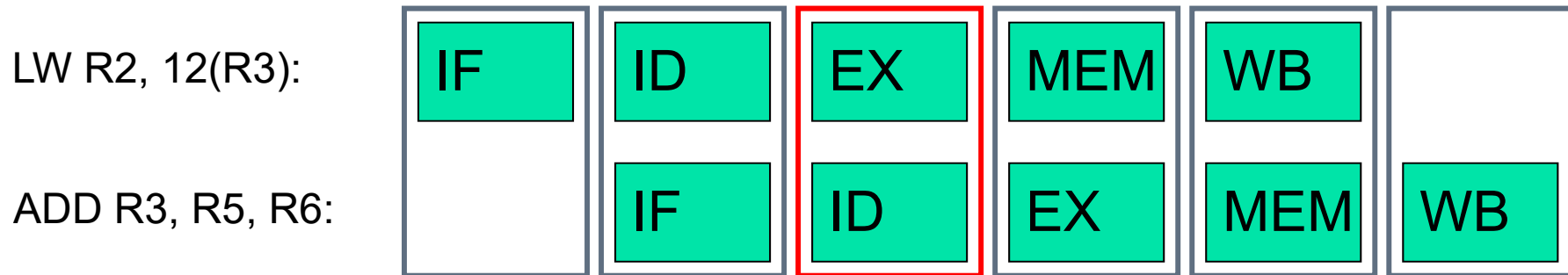
Decodifica a instrução

- Lê o conteúdo do registrador R3
- Obtém a constante 12 (imediato), e faz a extensão de sinal

□ I2:

- Envia o PC p/ memória para buscar I2
- $PC = PC + 4$

Exemplo

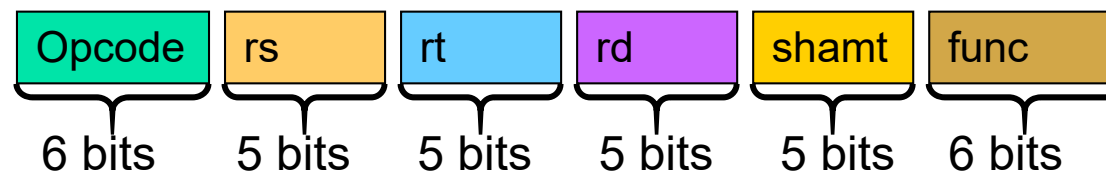


Terceiro Passo: I1 em EX, I2 em ID

I1:

- Soma o conteúdo de R3 c/ a constante 12, calculando endereço de acesso à memória.

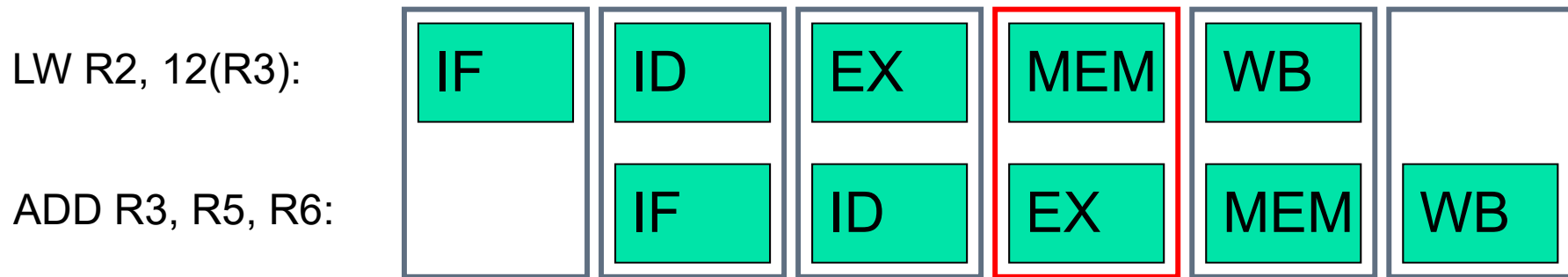
I2:



Decodifica a instrução

- Lê o conteúdo dos registradores R5 e R6

Exemplo



□ **Quarto Passo:** I1 em MEM, I2 em EX

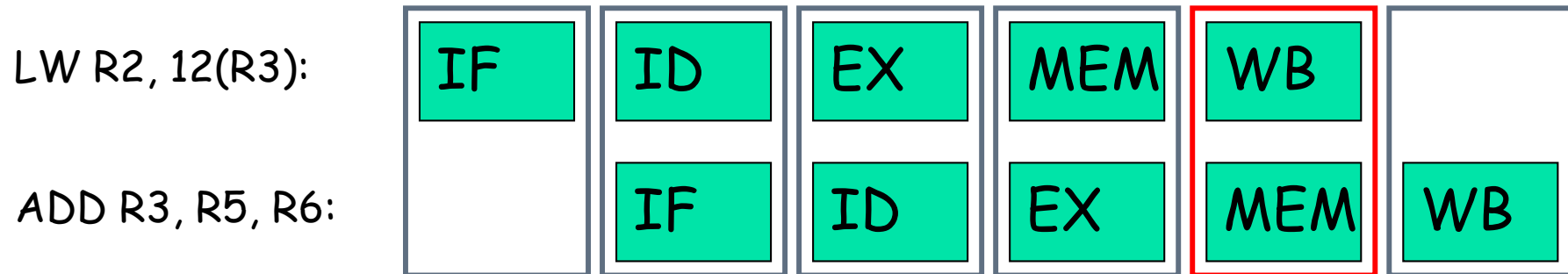
▣ I1:

- Envia o endereço calculado em EX p/ a Memória
- Lê a double word da memória

▣ I2:

- Soma o conteúdo de R5 c/ R6

Exemplo



□ **Quinto Passo:** I1 em WB, I2 em MEM

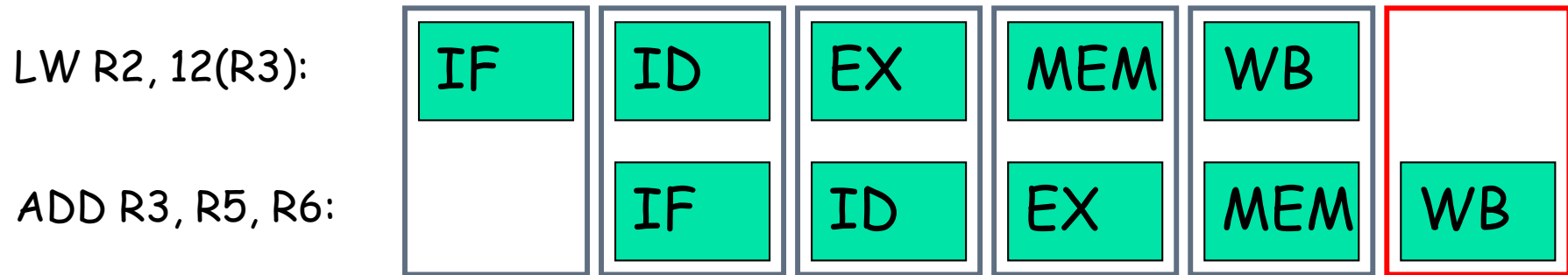
▣ I1:

■ Grava o dado lido da memória no registrador R2

▣ I2:

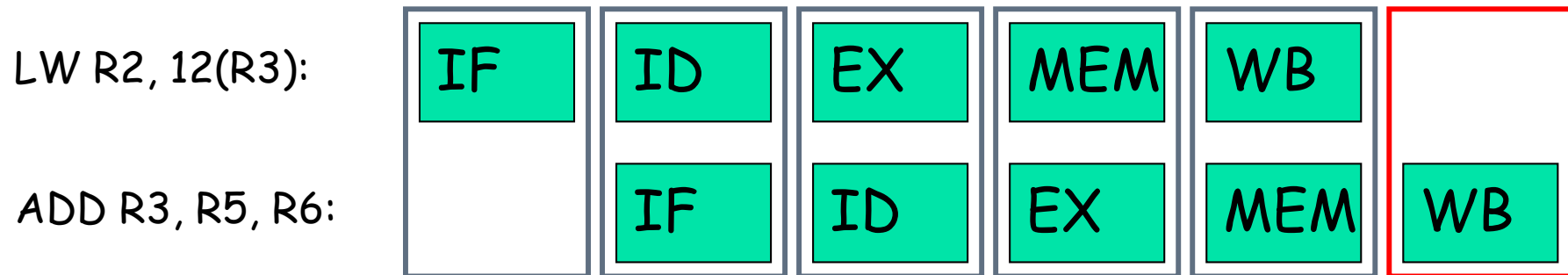
■ Nada a fazer (instrução add não acessa a memória)

Exemplo



- **Sexto Passo:** I1 terminada, I2 in WB
 - ▣ I2:
 - Grava o valor calculado em EX no registrador R3

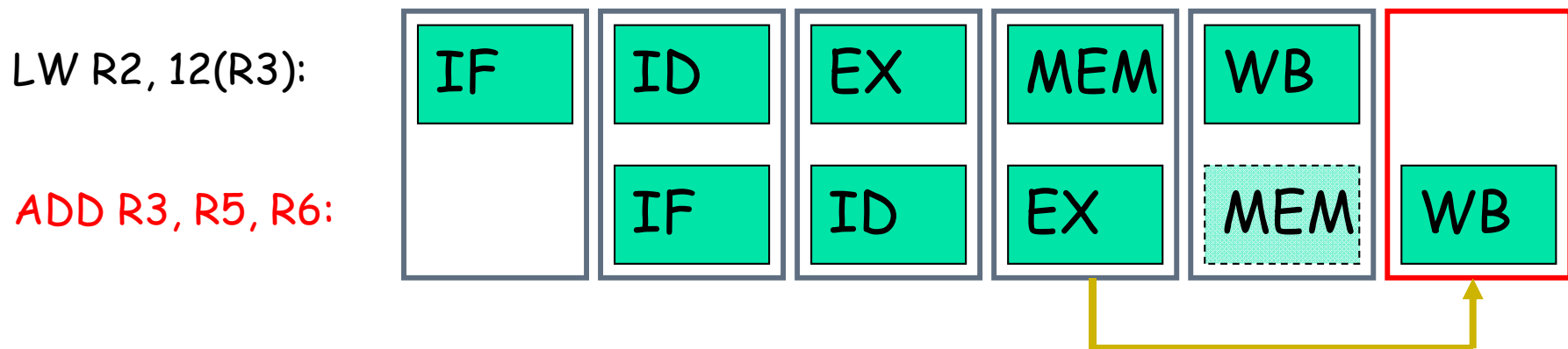
Exemplo



- **Sexto Passo:** I1 terminada, I2 in WB
 - ▣ I2:
 - Grava o valor calculado em EX no registrador R3
- **Fim da execução:** Duas instruções foram executadas em 6 ciclos de clock
 - ▣ **CPI médio:** 3 ciclos de clock
 - ▣ **Duração de cada instrução:** 5 ciclos de clock

Pipelining e Registradores

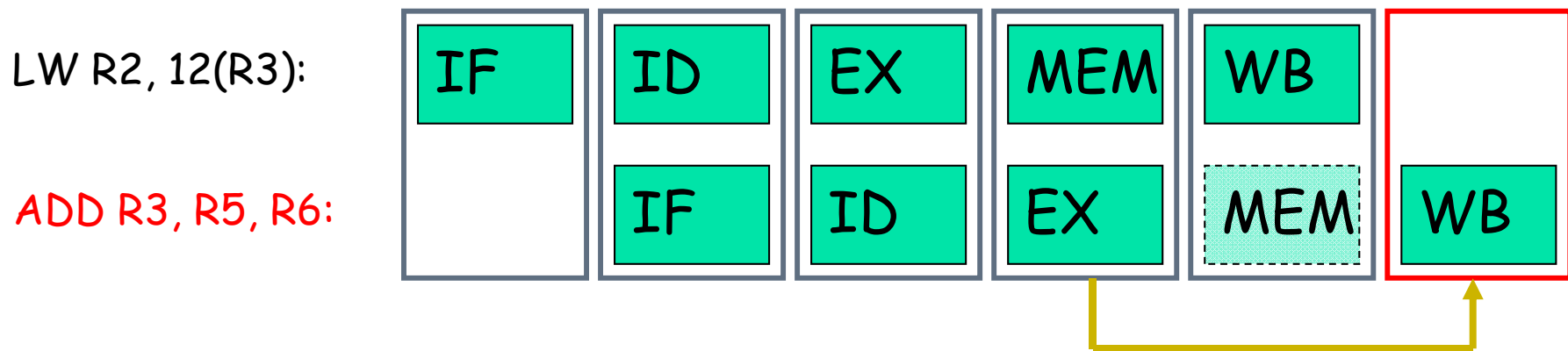
- Às vezes um valor é calculado em um estágio, mas usado em um outro estágio **não adjacente**.



- **Pergunta:** como esse valor é comunicado do estágio EX para o estágio WB ?

Pipelining e Registradores

- Às vezes um valor é calculado em um estágio, mas usado em um outro estágio **não adjacente**.

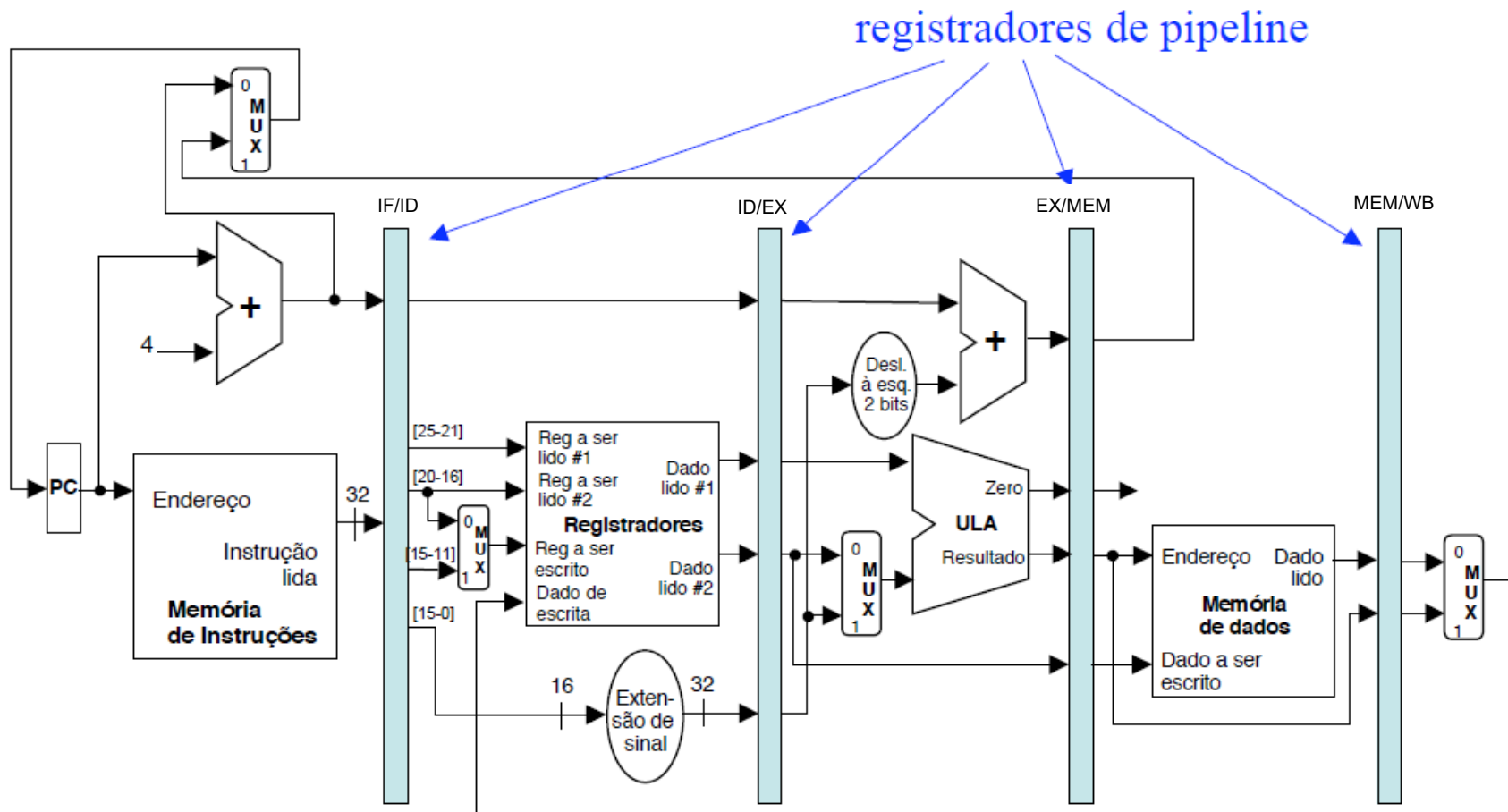


- **Pergunta:** como esse valor é comunicado do estágio EX para o estágio WB ?
- **Resposta:** Usando registradores de pipeline

Registradores de Pipelining

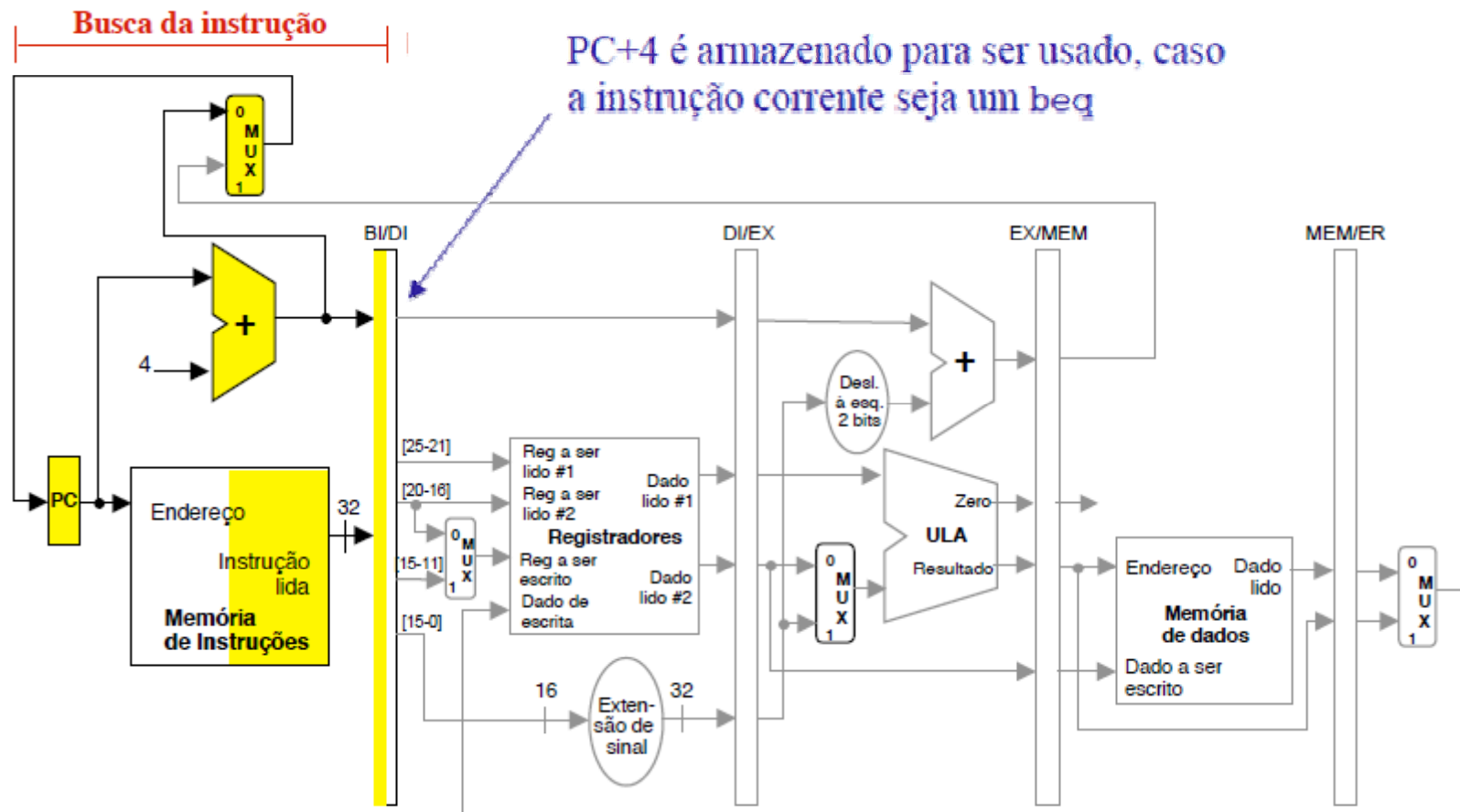
- A técnica descrita anteriormente necessita de **registradores de pipelining**
- Para o datapath MIPS descrito, temos **4 grupos de registradores de pipelining**.
- Cada grupo é usado para armazenar a saída de um estágio, e passá-la à entrada do estágio seguinte.
- Podemos identificar esses grupos de acordo c/ os estágios adjacentes:
 - ▣ Grupo 1: IF/ID
 - ▣ Grupo 2: ID/EX
 - ▣ Grupo 3: EX/MEM
 - ▣ Grupo 4: MEM/WB
- Dessa forma, toda informação obtida ou computada é guardada para ser usada no contexto da instrução em questão.

Registradores de Pipelining



Pipelining para MIPS

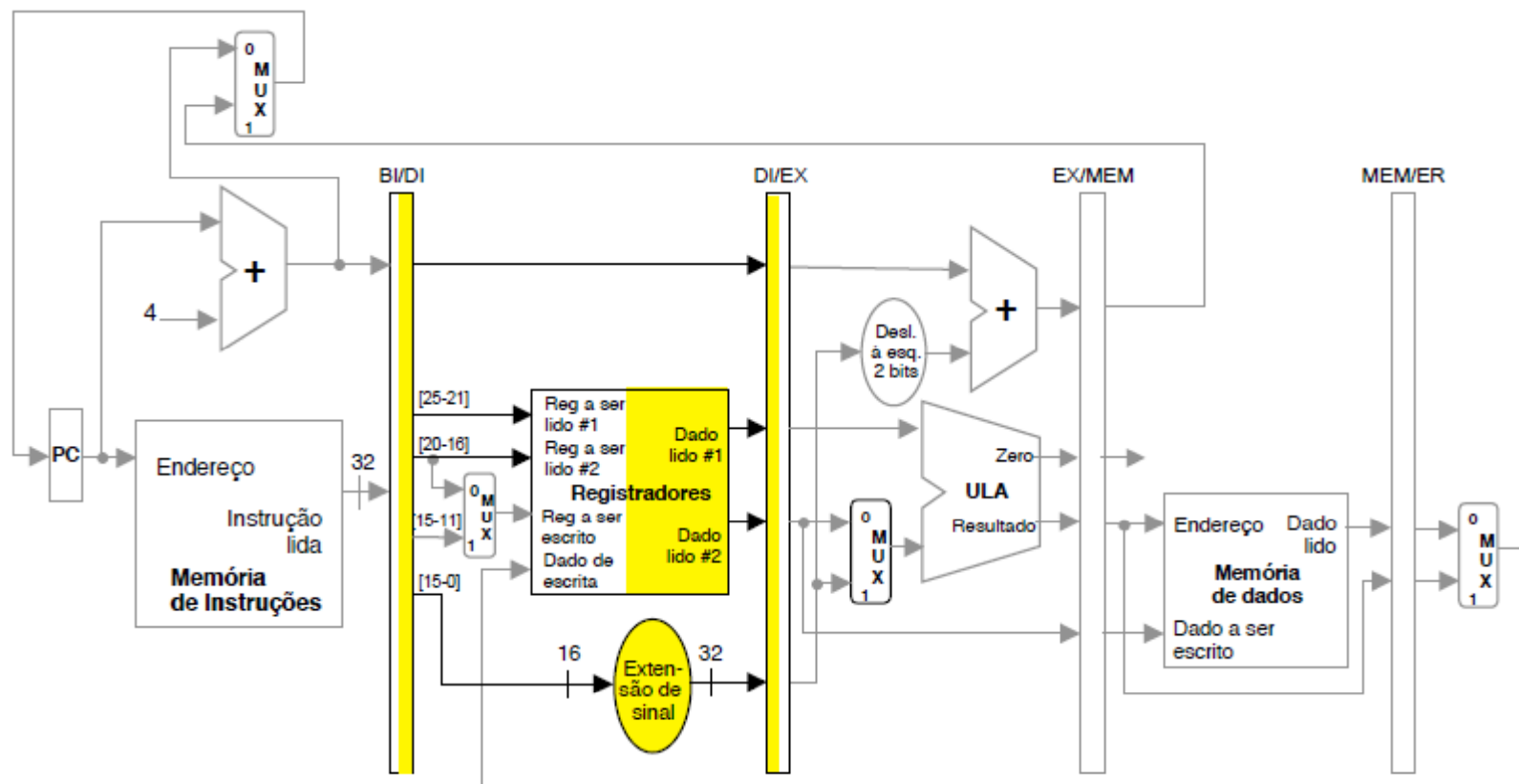
□ Executando sw



Pipelining para MIPS

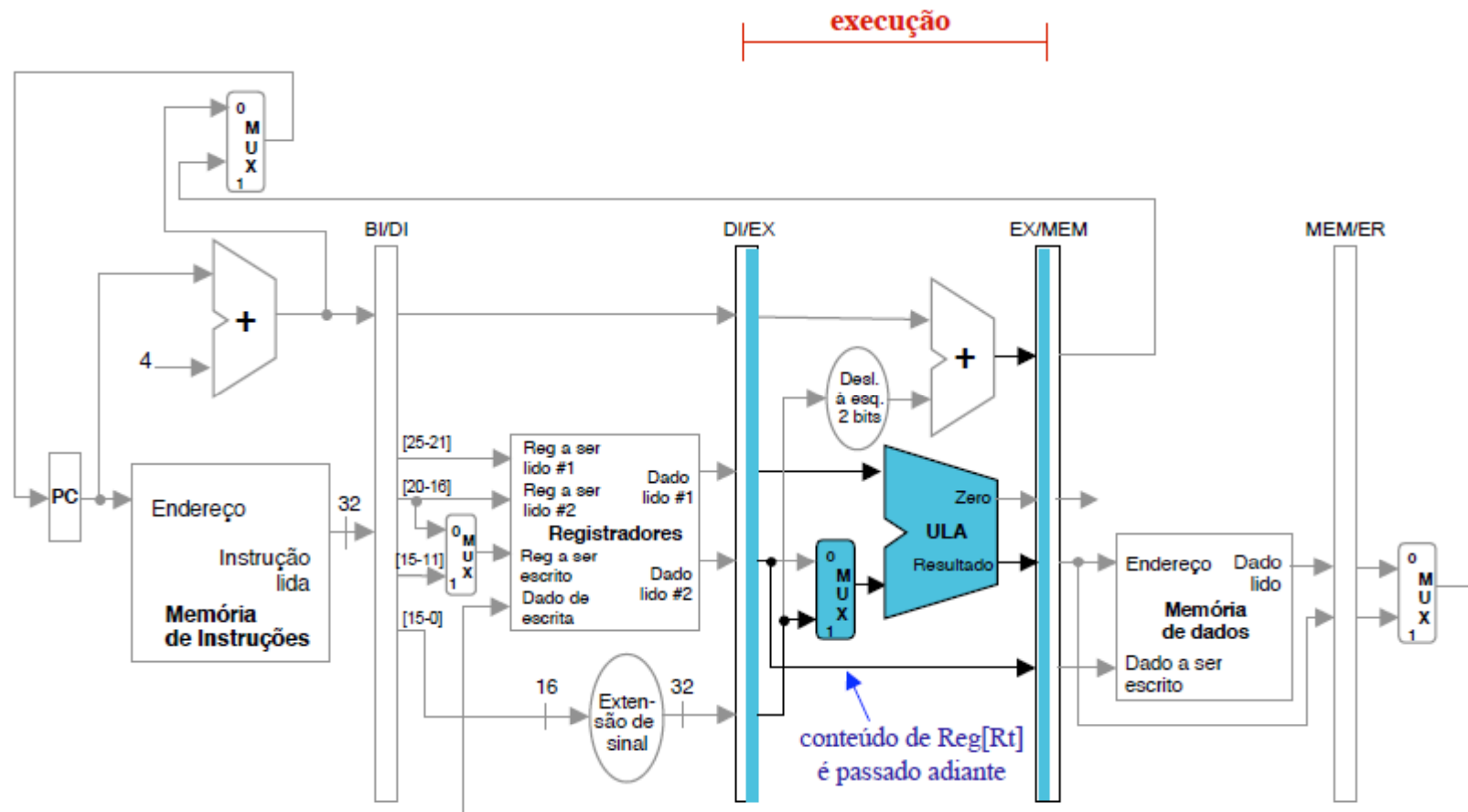
□ Executando sw

decodificação da instrução



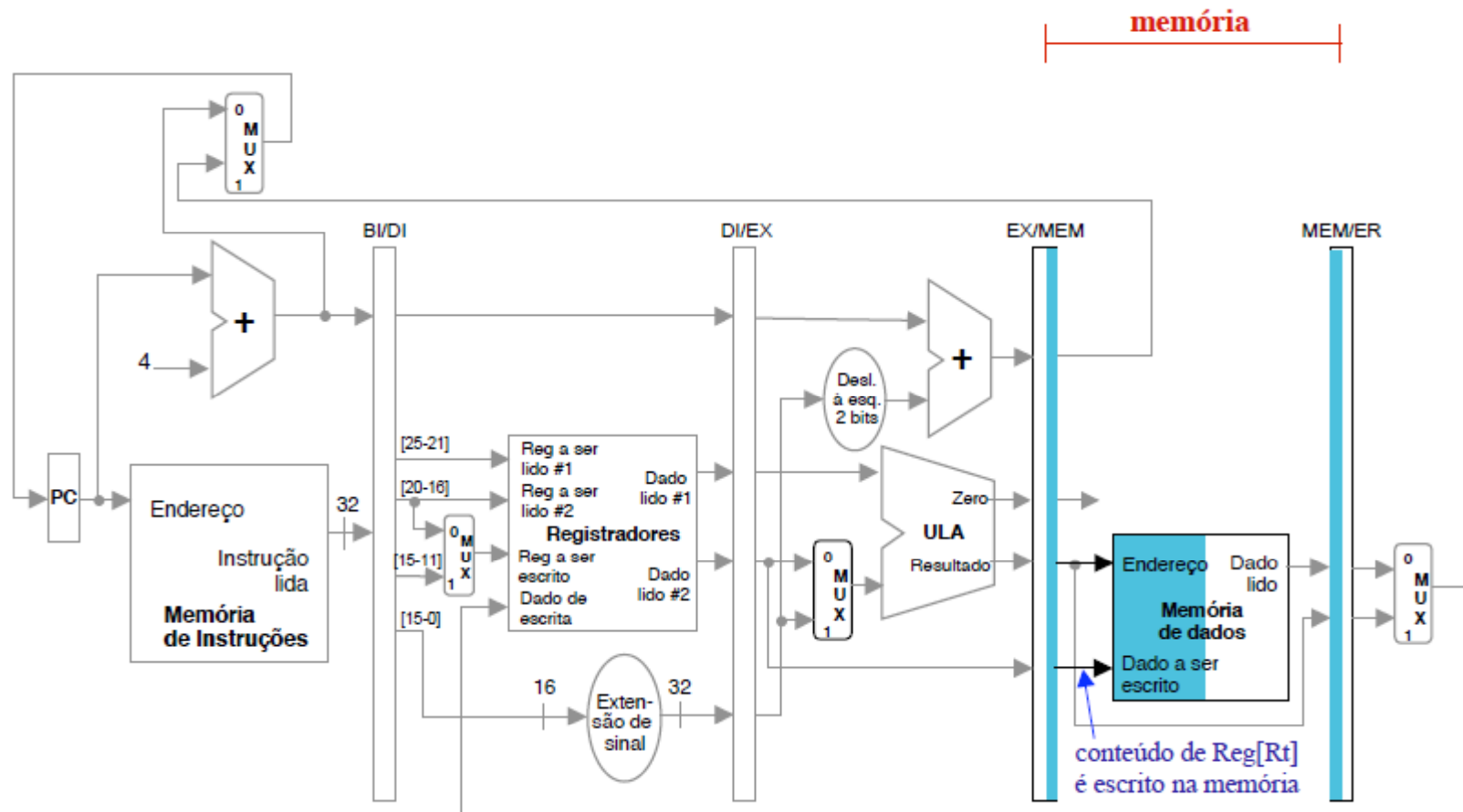
Pipelining para MIPS

□ Executando sw



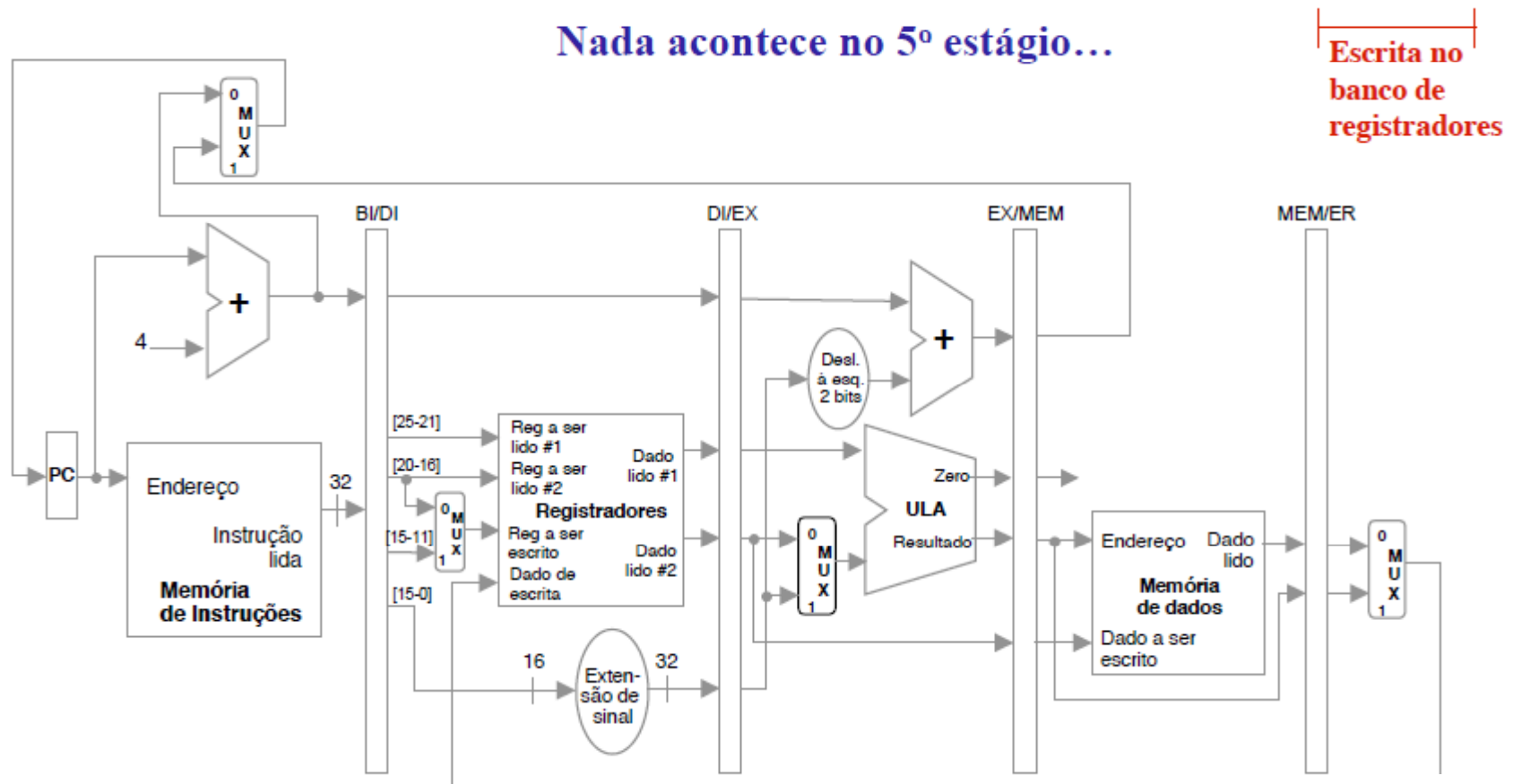
Pipelining para MIPS

□ Executando sw



Pipelining para MIPS

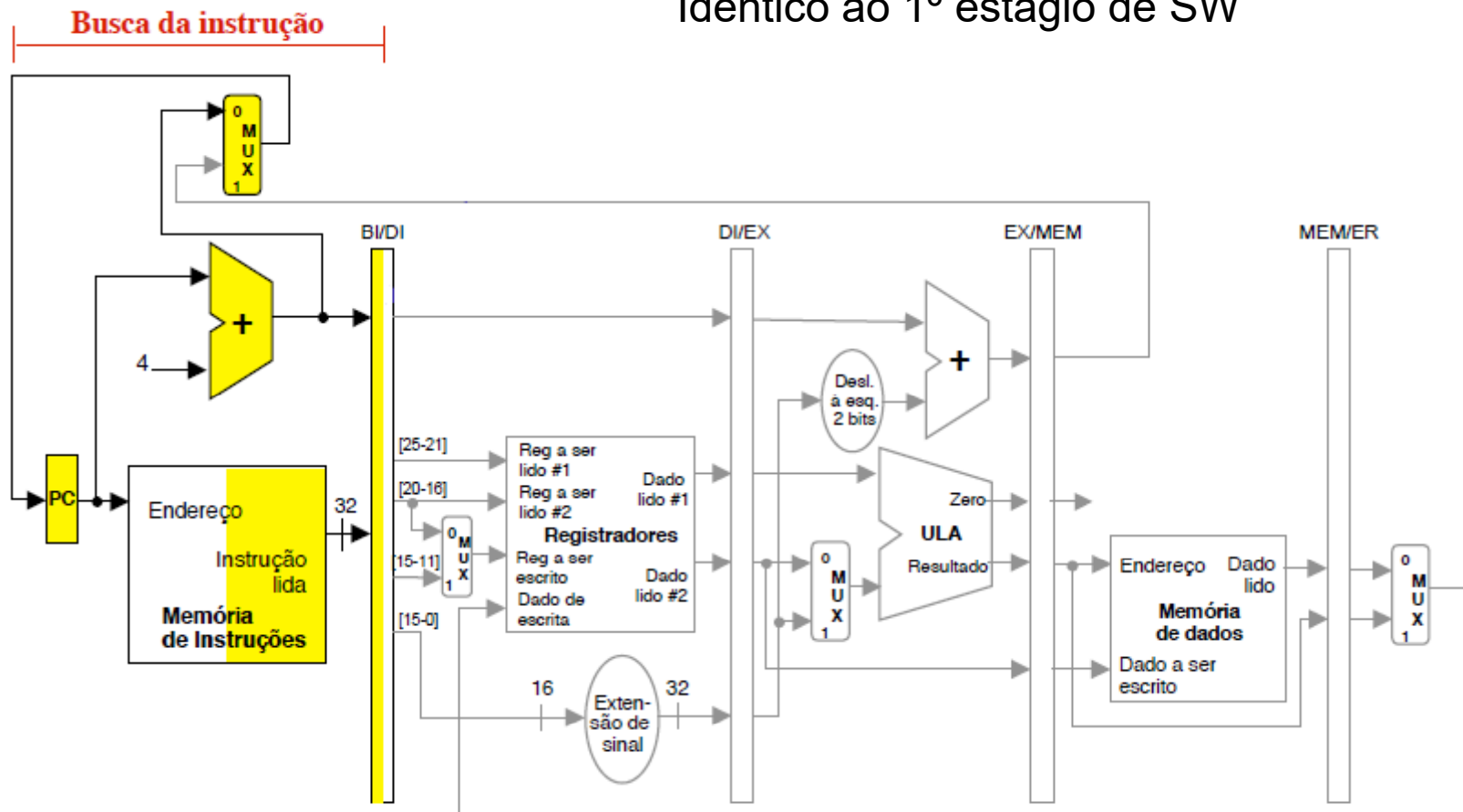
□ Executando sw



Pipelining para MIPS

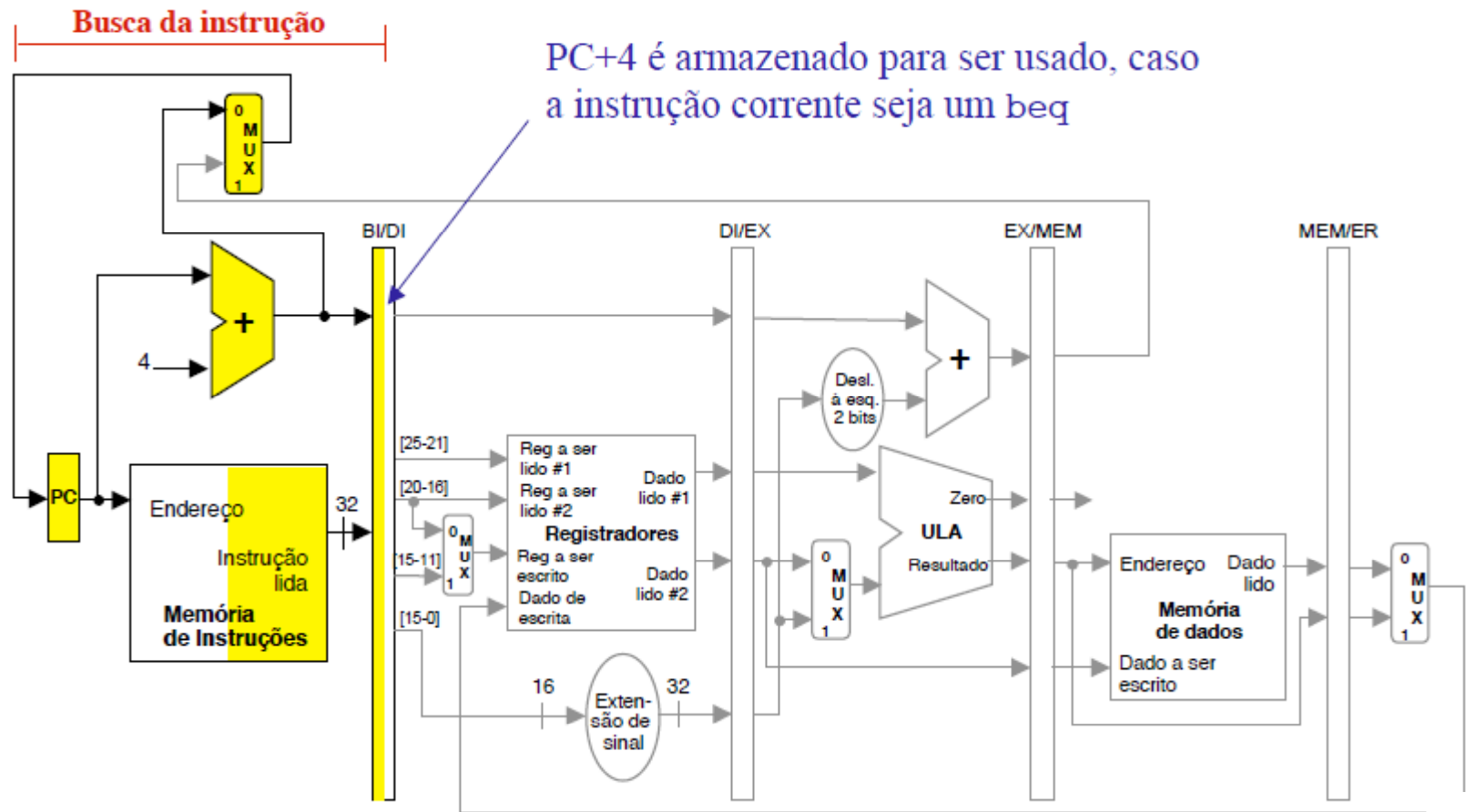
□ Executando lw

Idêntico ao 1º estágio de SW



Pipelining para MIPS

□ Executando lw

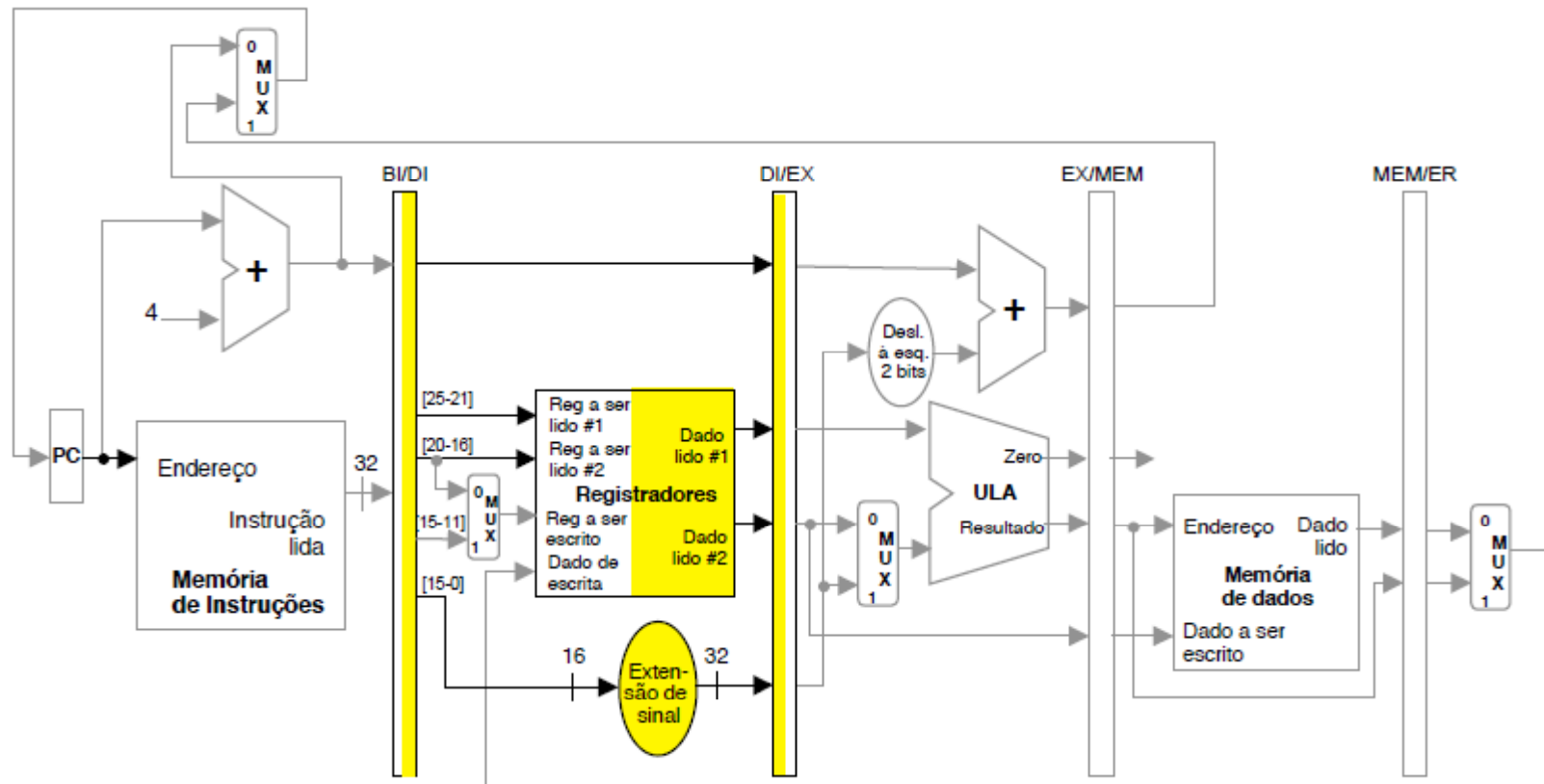


Pipelining para MIPS

□ Executando lw

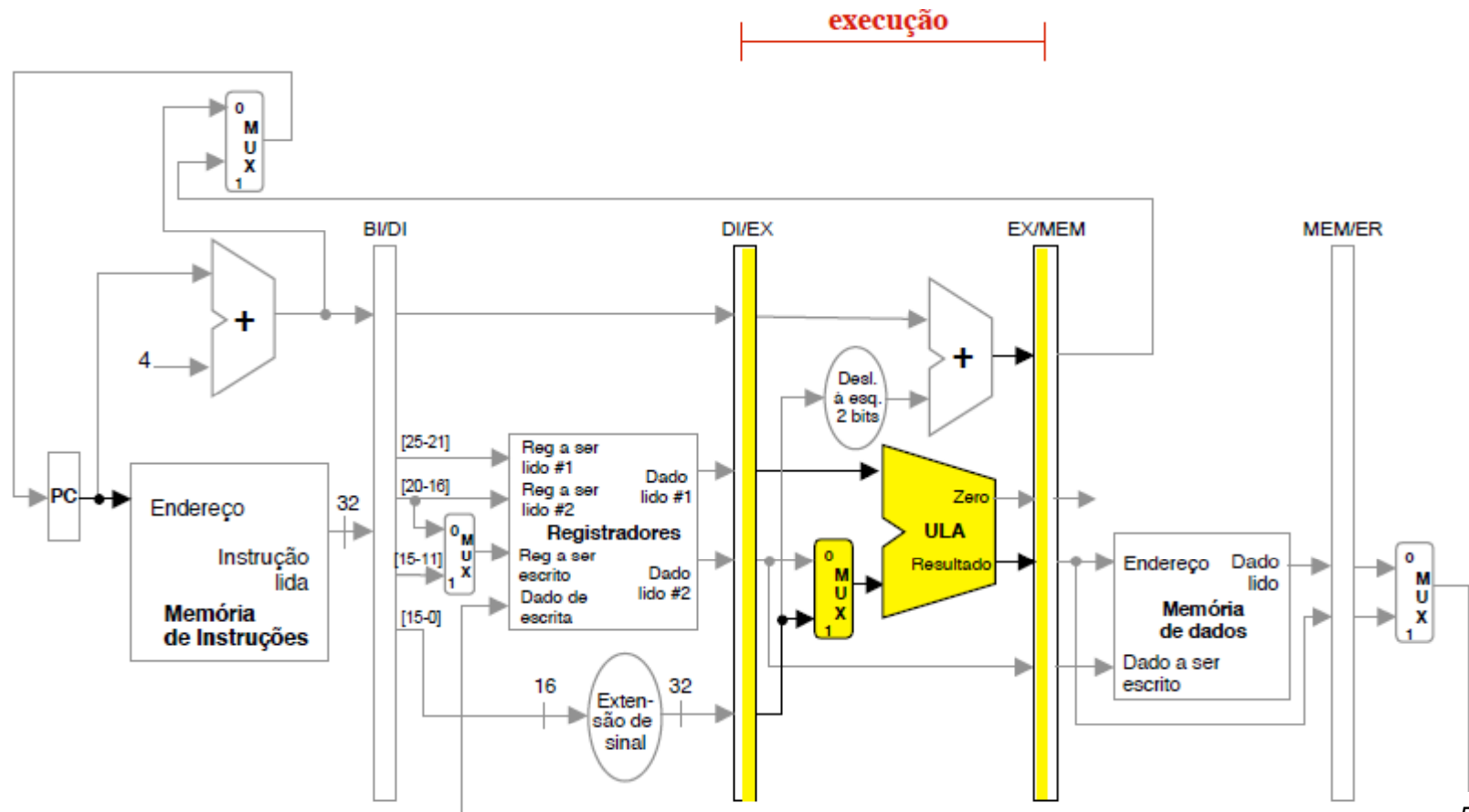
Idêntico ao 2º estágio de SW

decodificação da instrução



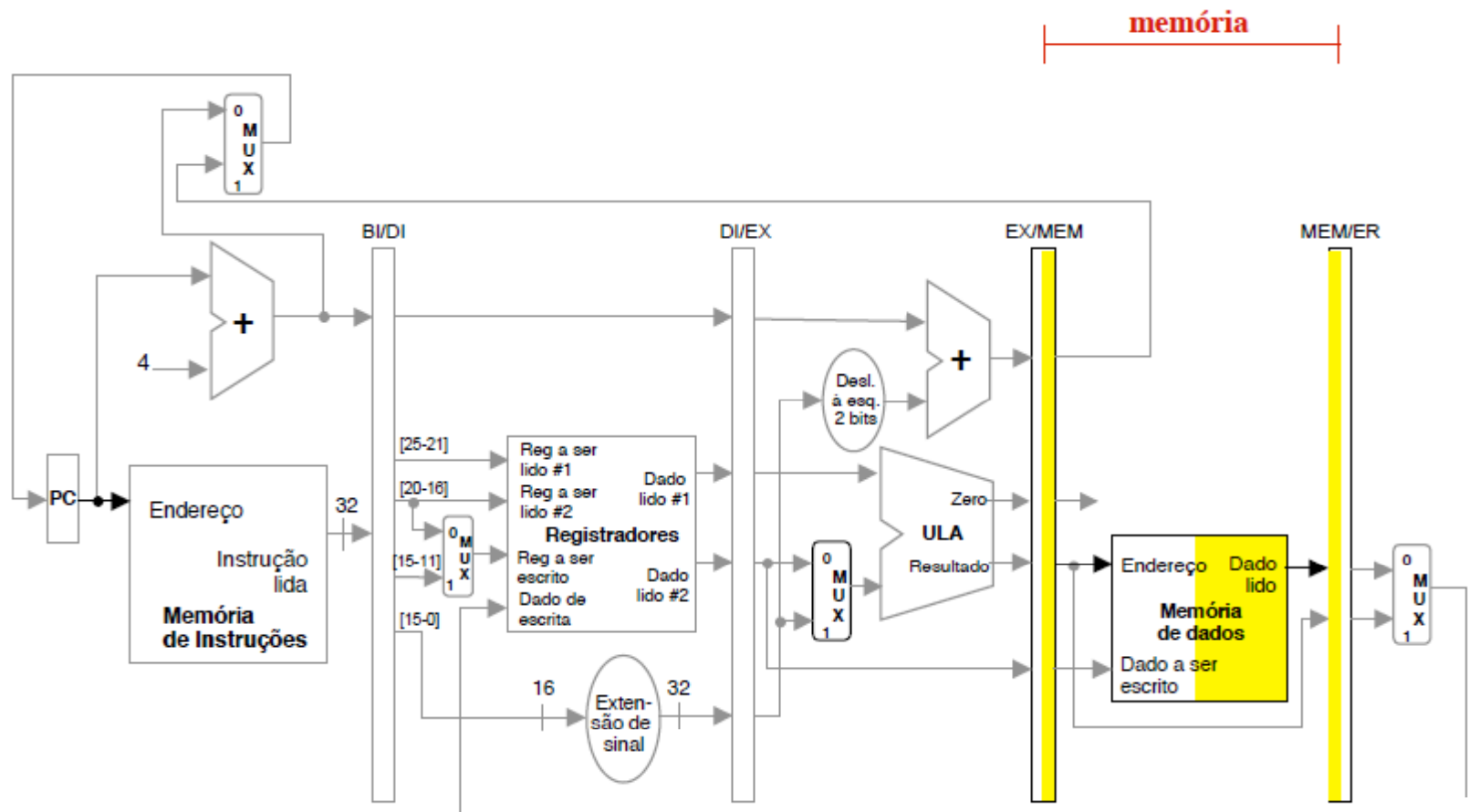
Pipelining para MIPS

□ Executando lw



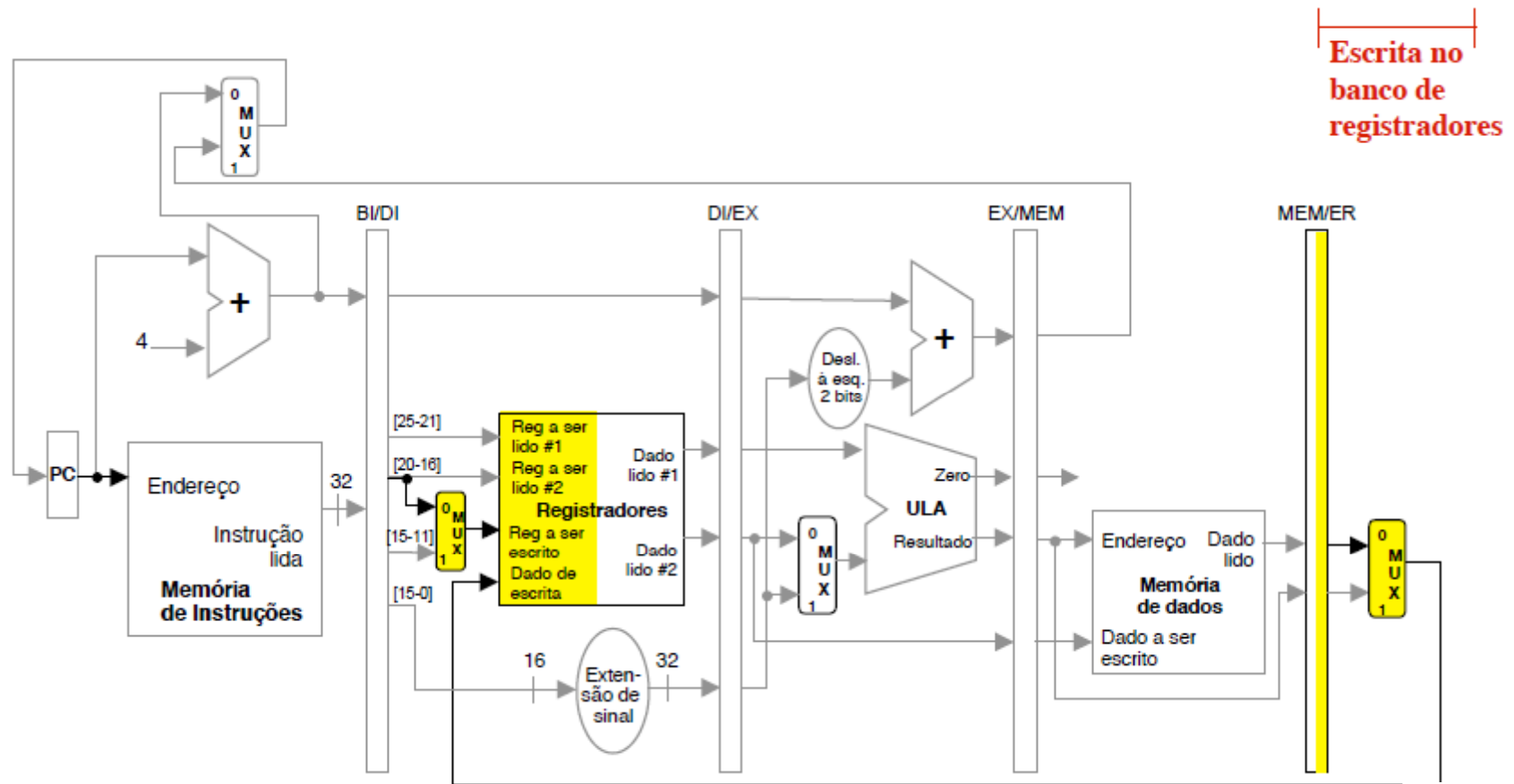
Pipelining para MIPS

□ Executando lw



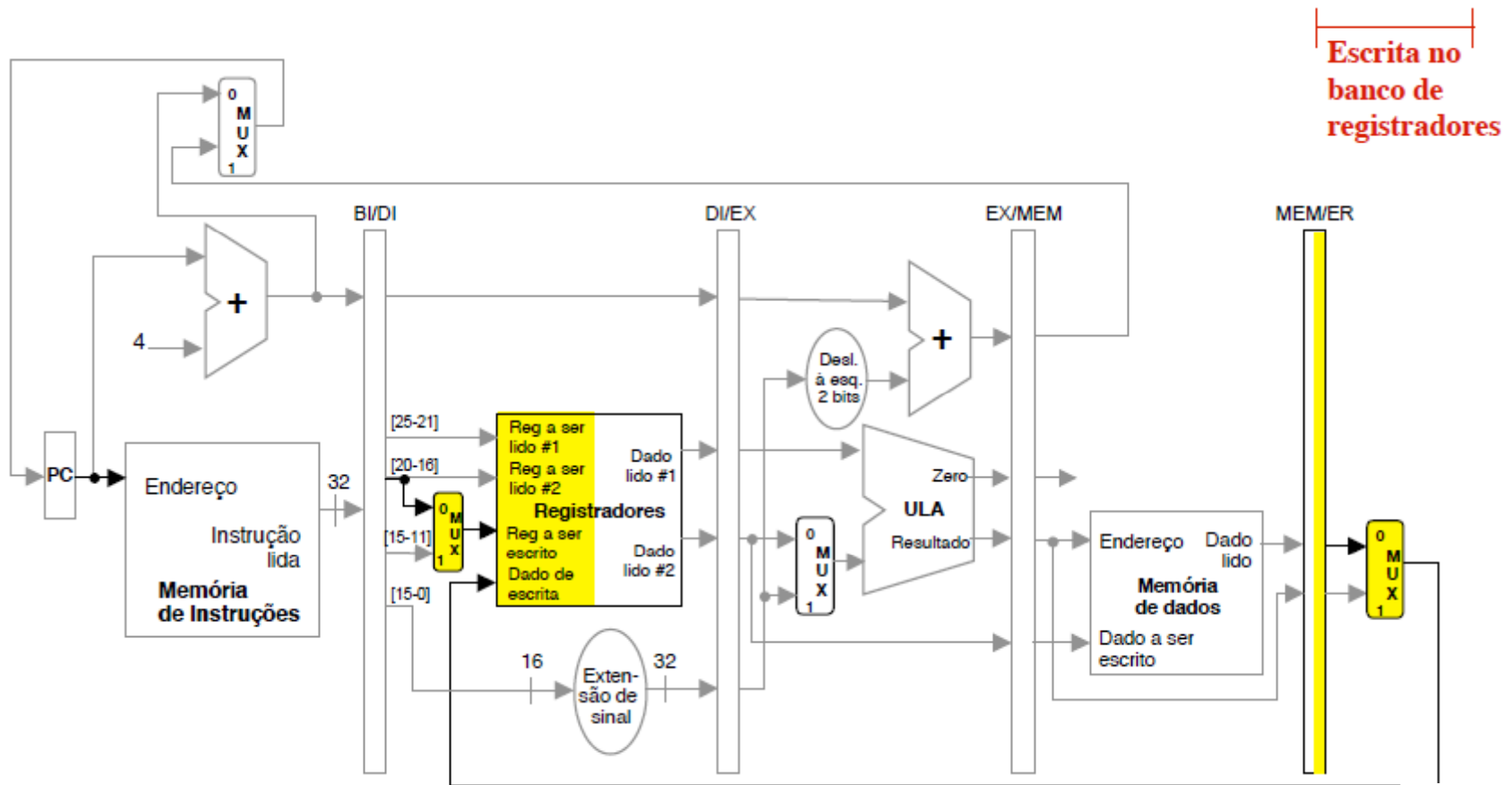
Pipelining para MIPS

□ Executando lw



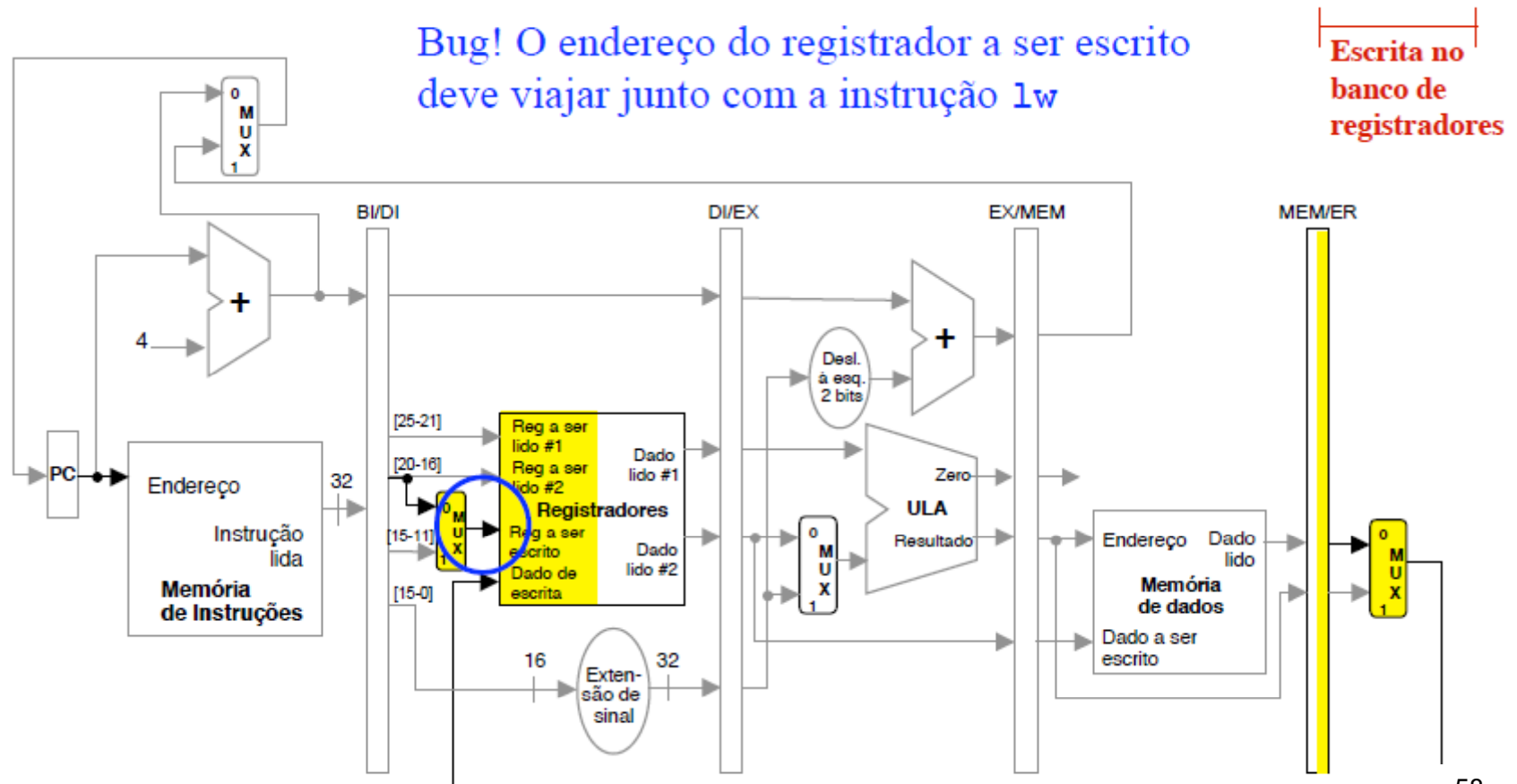
Pipelining para MIPS

□ Executando lw (BUG)



Pipelining para MIPS

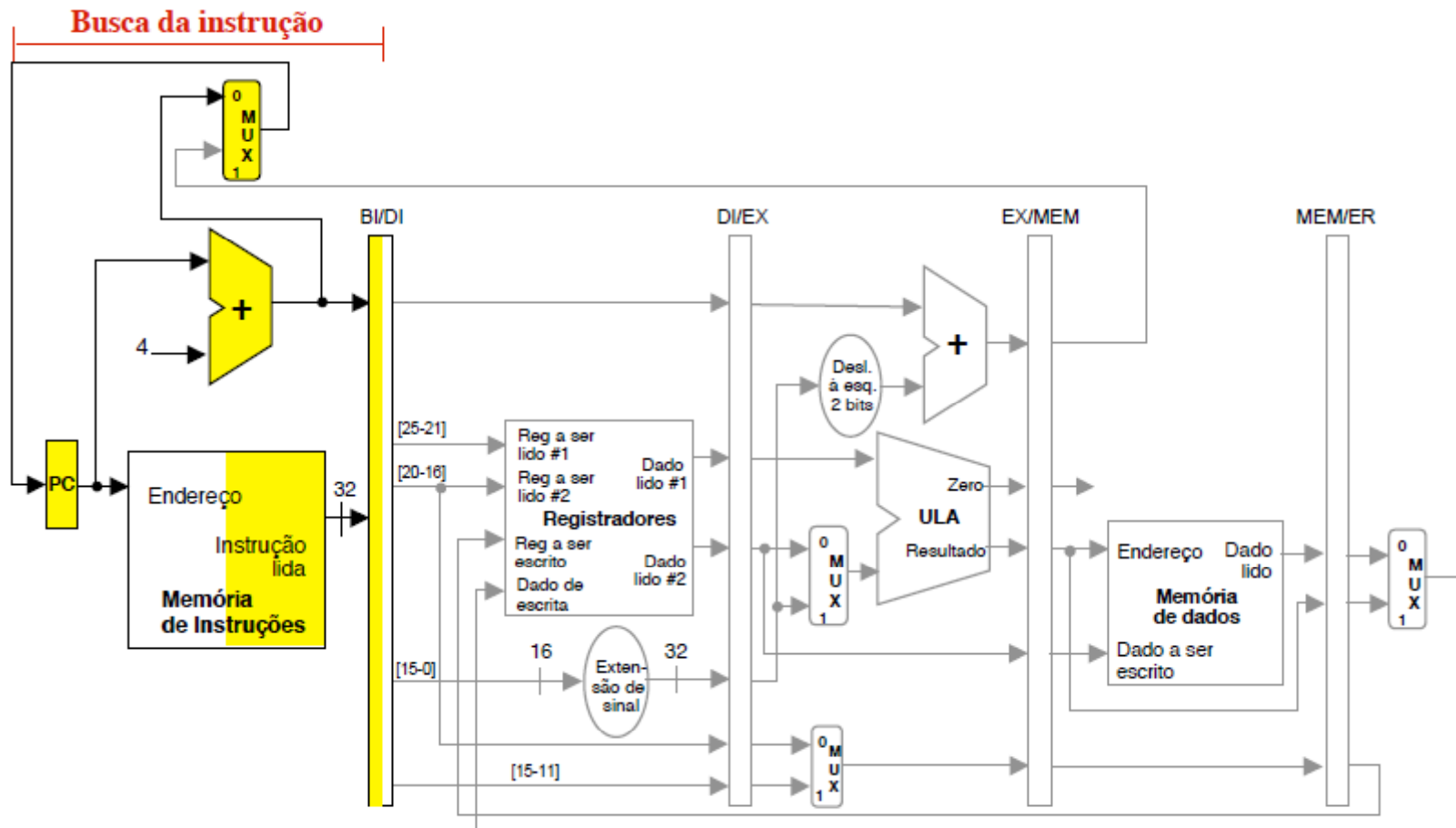
□ Executando lw





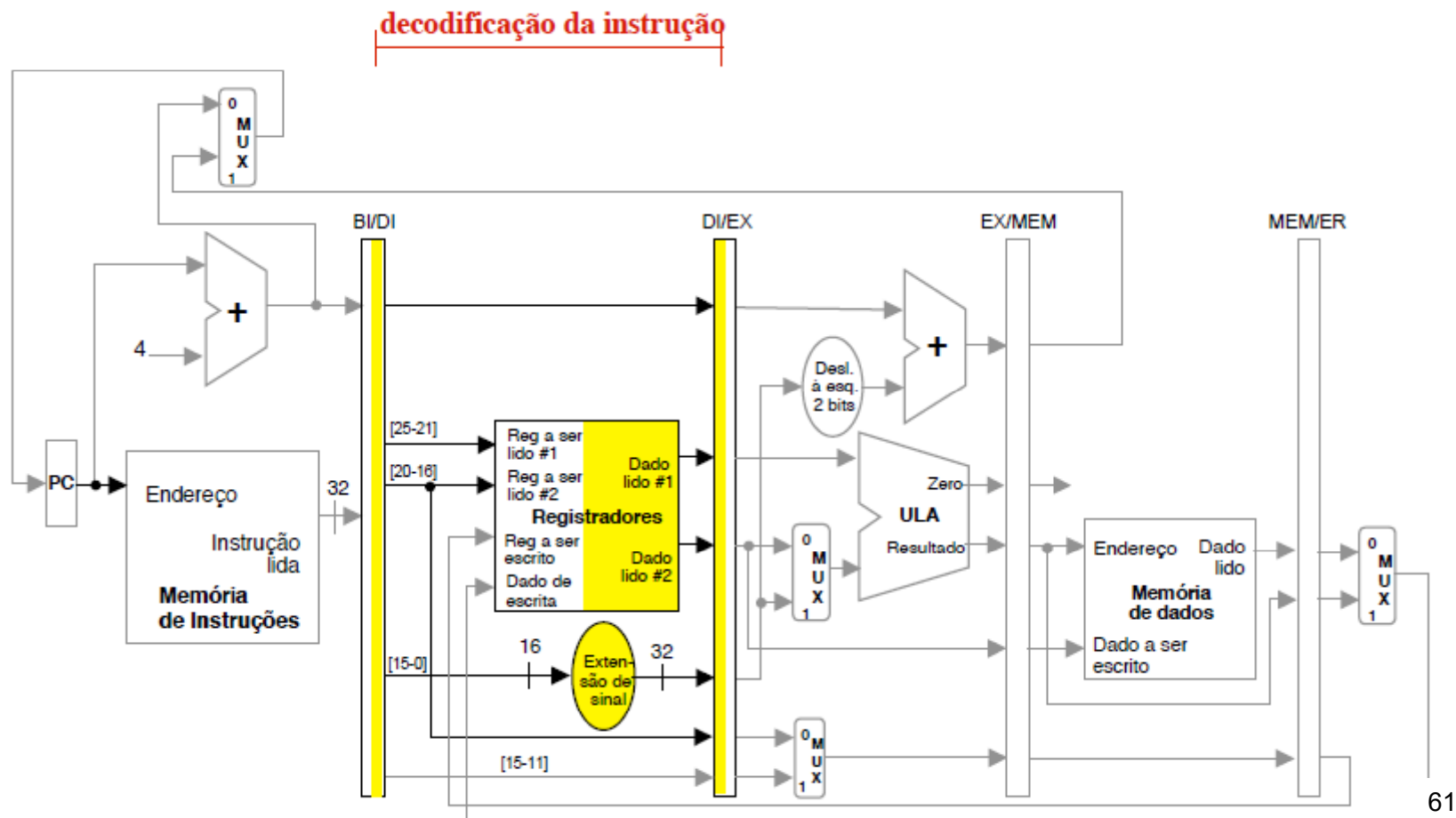
Pipelining para MIPS

□ Executando lw (diagrama corrigido)



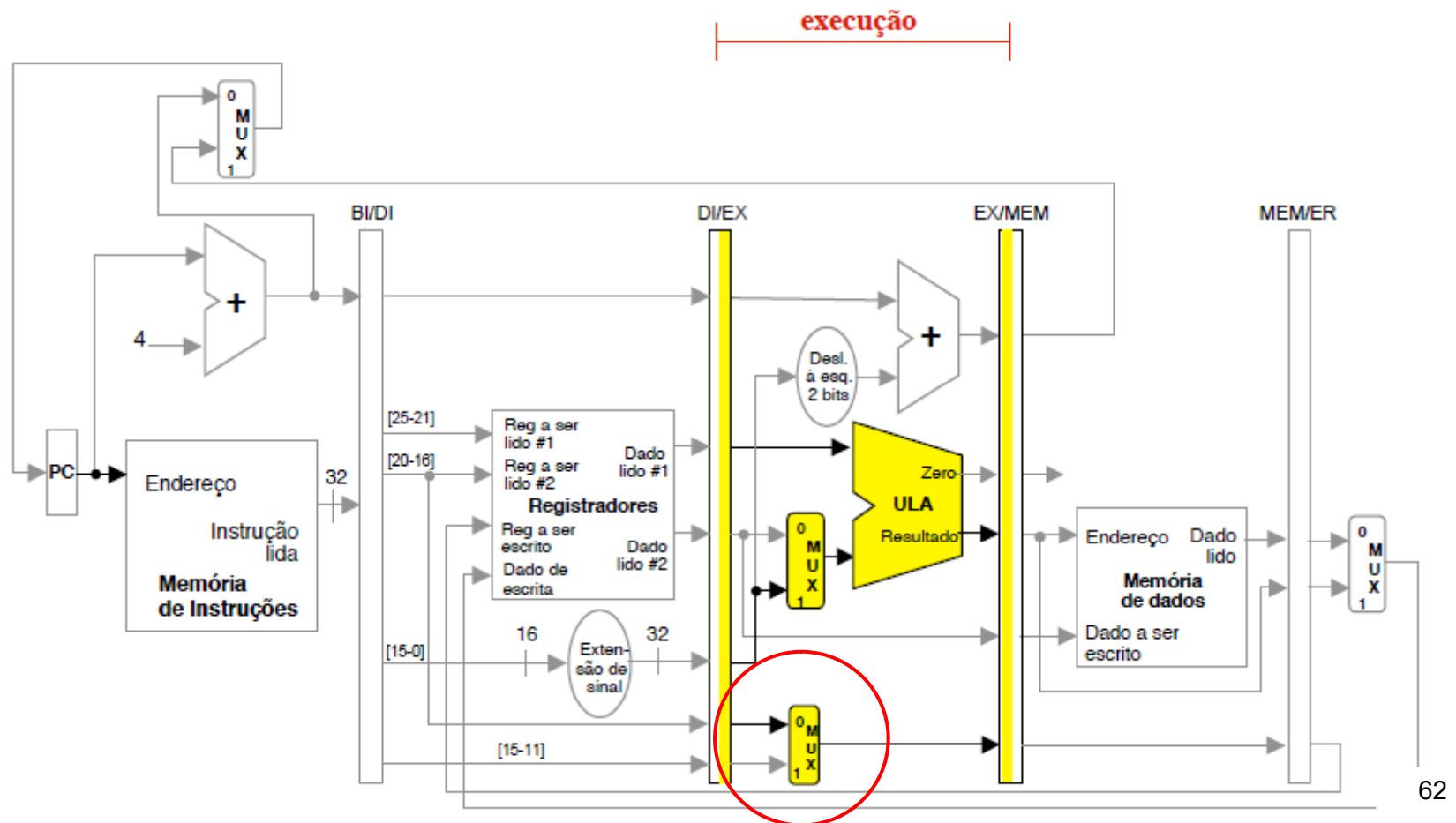
Pipelining para MIPS

□ Executando lw (diagrama corrigido)



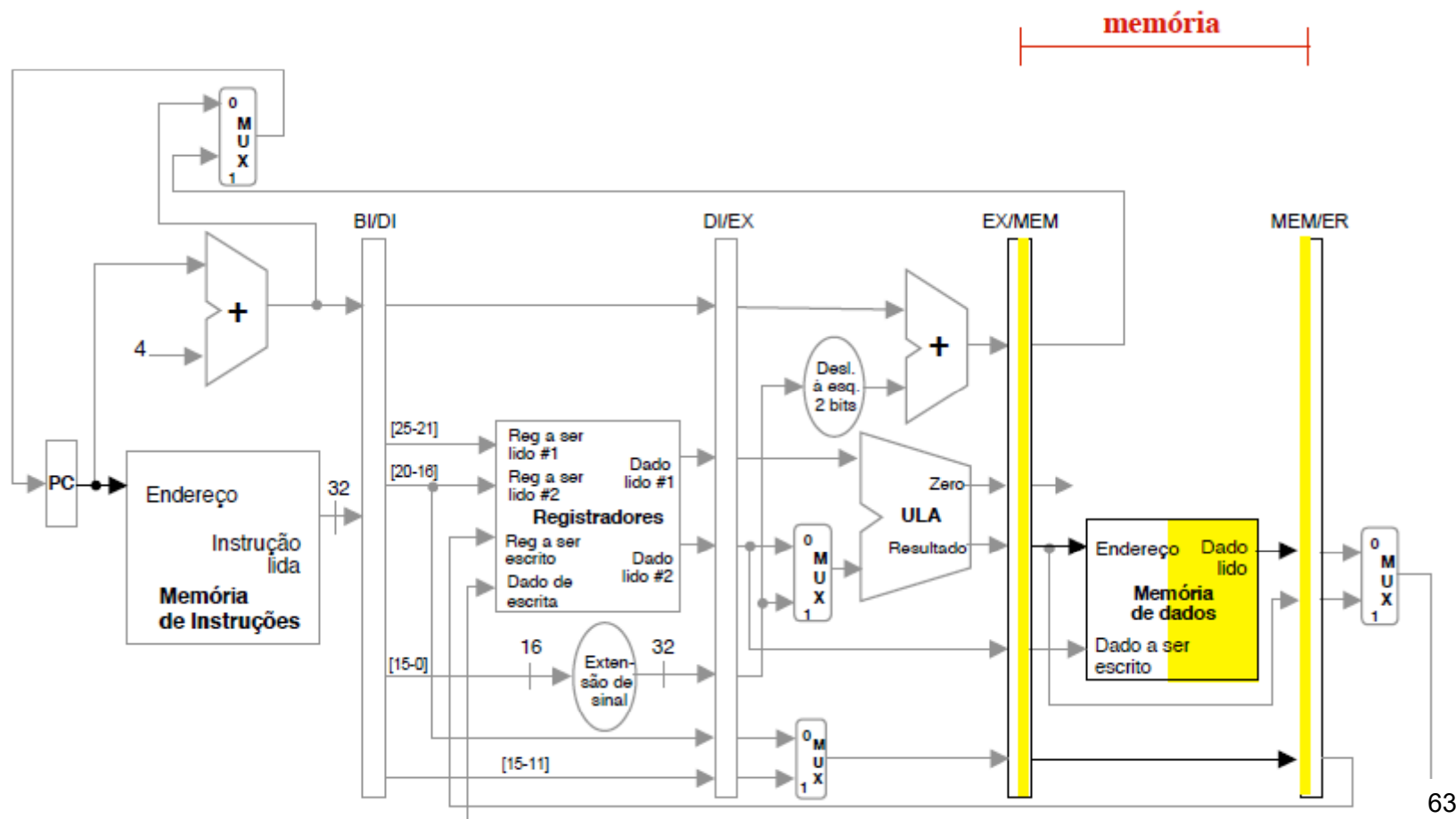
Pipelining para MIPS

□ Executando lw (diagrama corrigido)



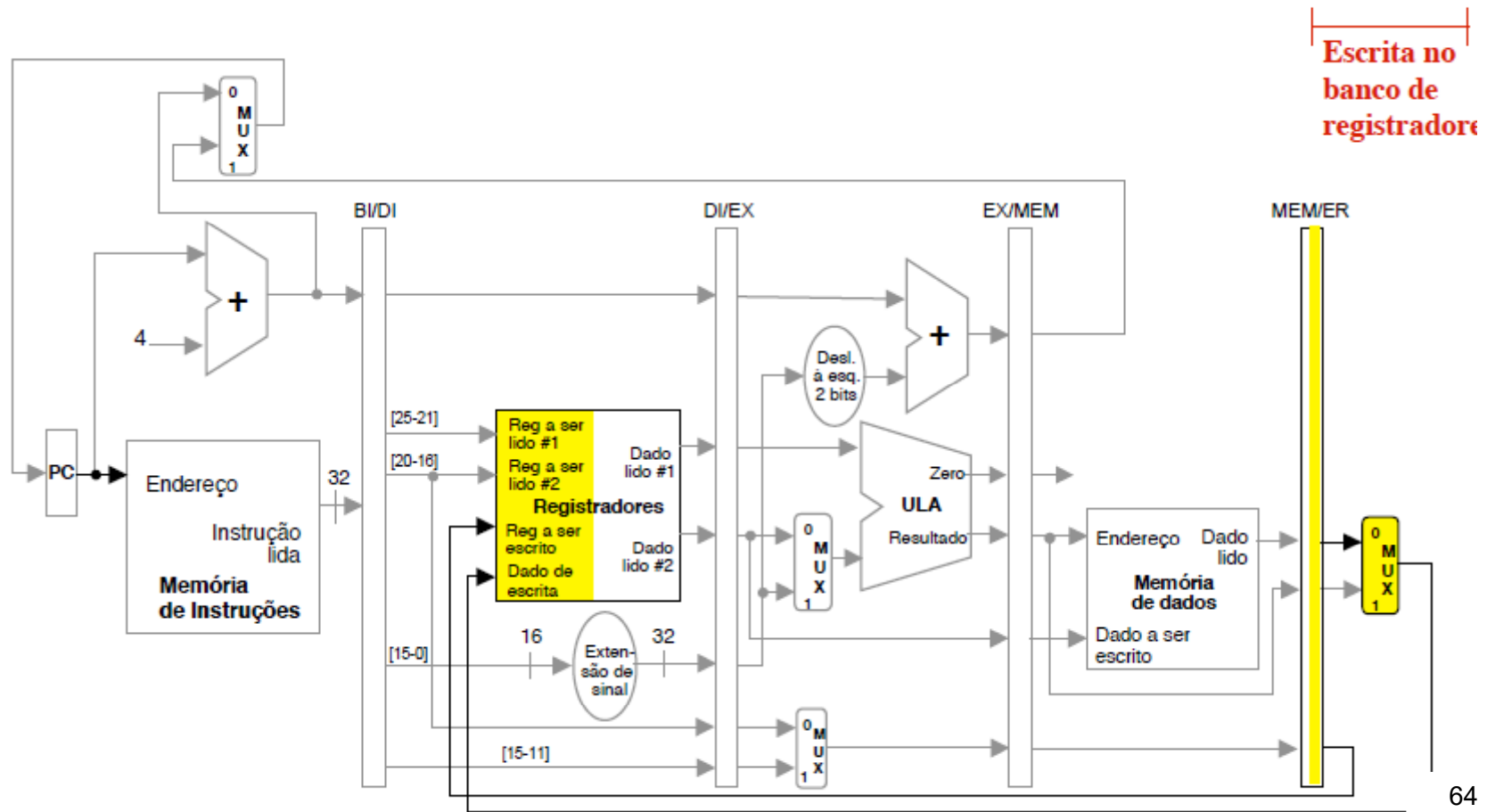
Pipelining para MIPS

□ Executando lw (diagrama corrigido)



Pipelining para MIPS

□ Executando lw (diagrama corrigido)



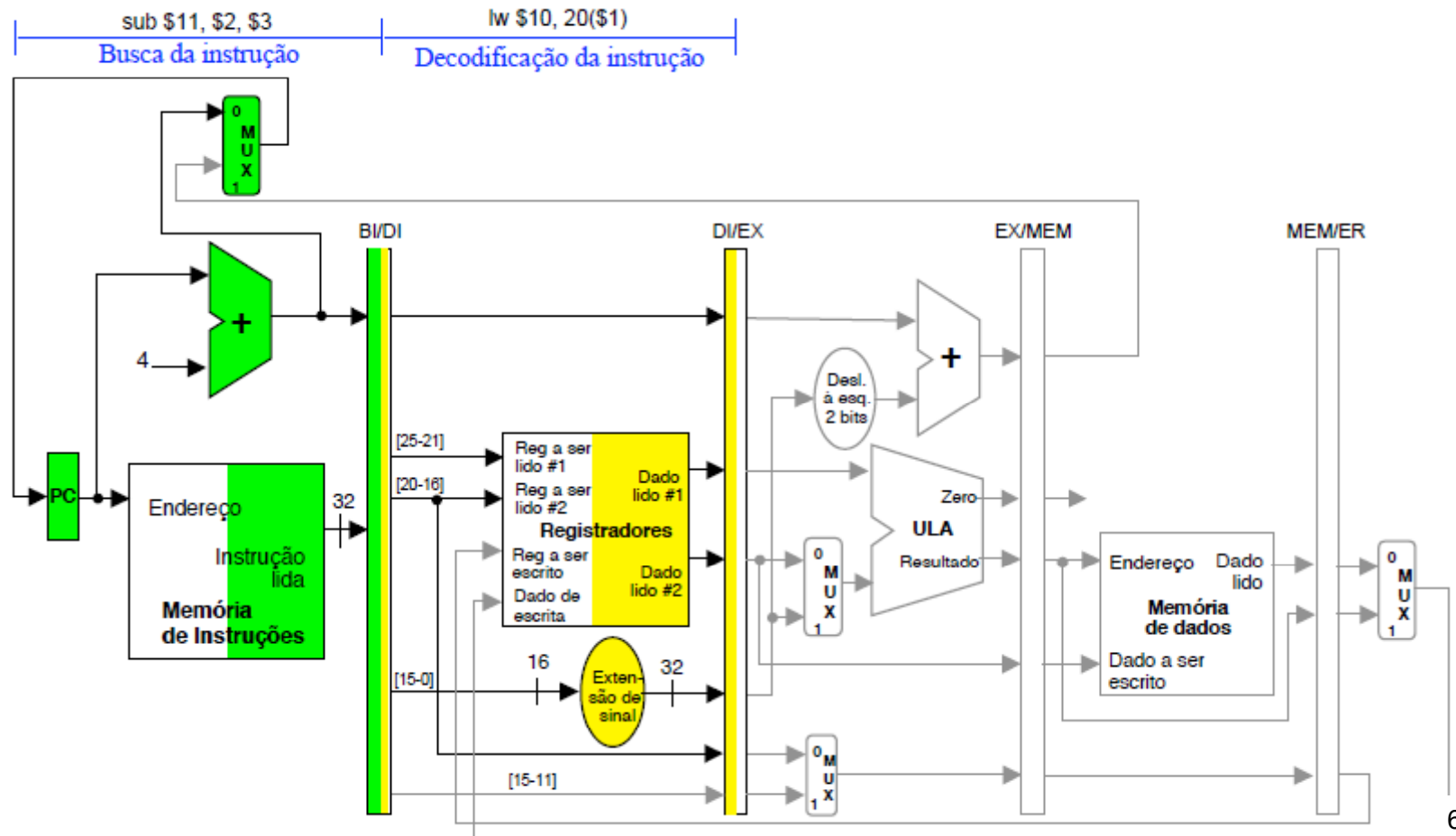
□ Executando lw \$10, 20(\$1)
sub \$11, \$2, \$3

```
lw $10, 20($1)
sub $11, $2, $3
```



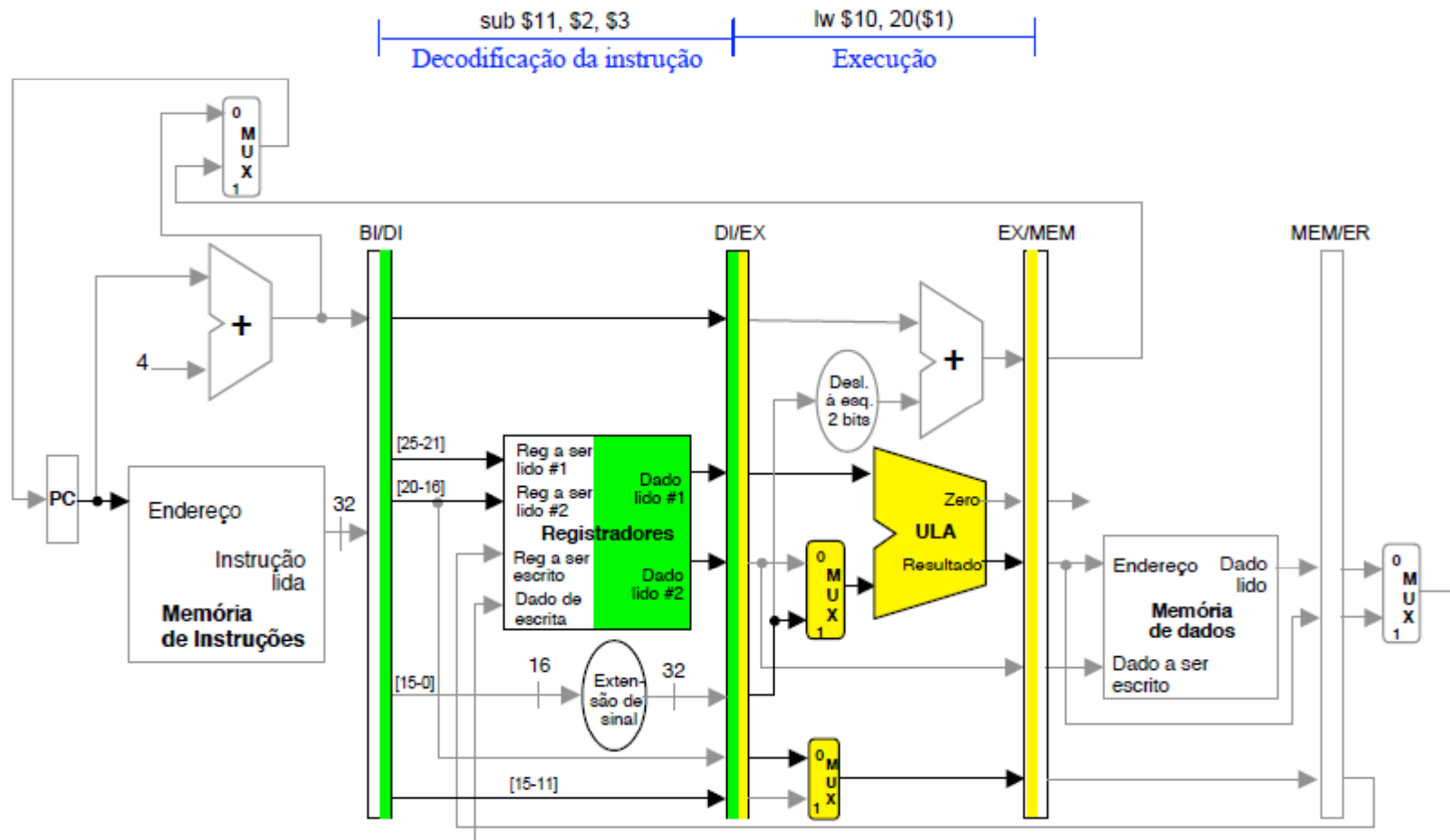
Pipelining para MIPS

□ Executando lw \$10, 20(\$1)
sub \$11, \$2, \$3



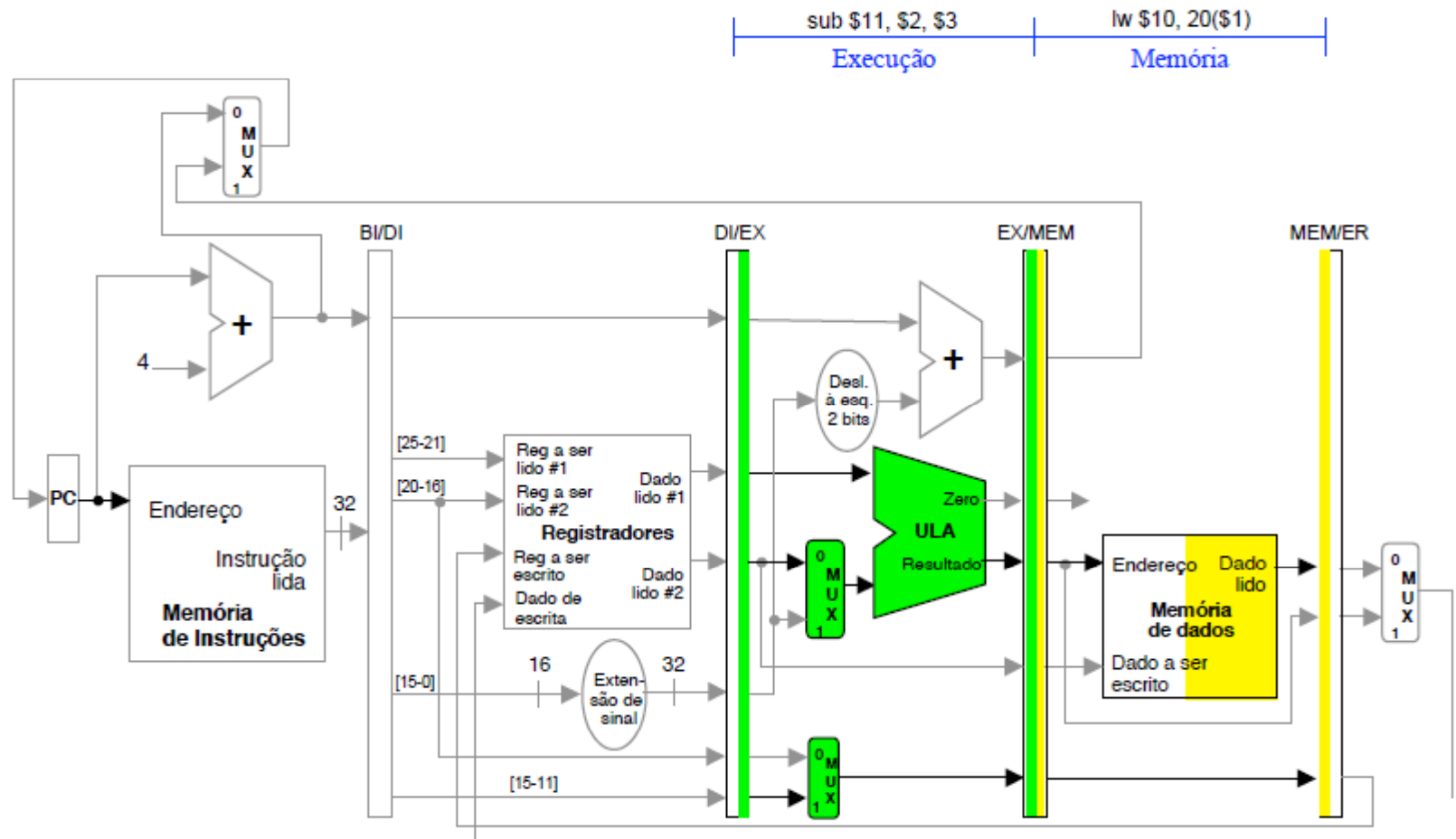
Pipelining para MIPS

□ Executando `lw $10, 20($1)` `sub $11, $2, $3`



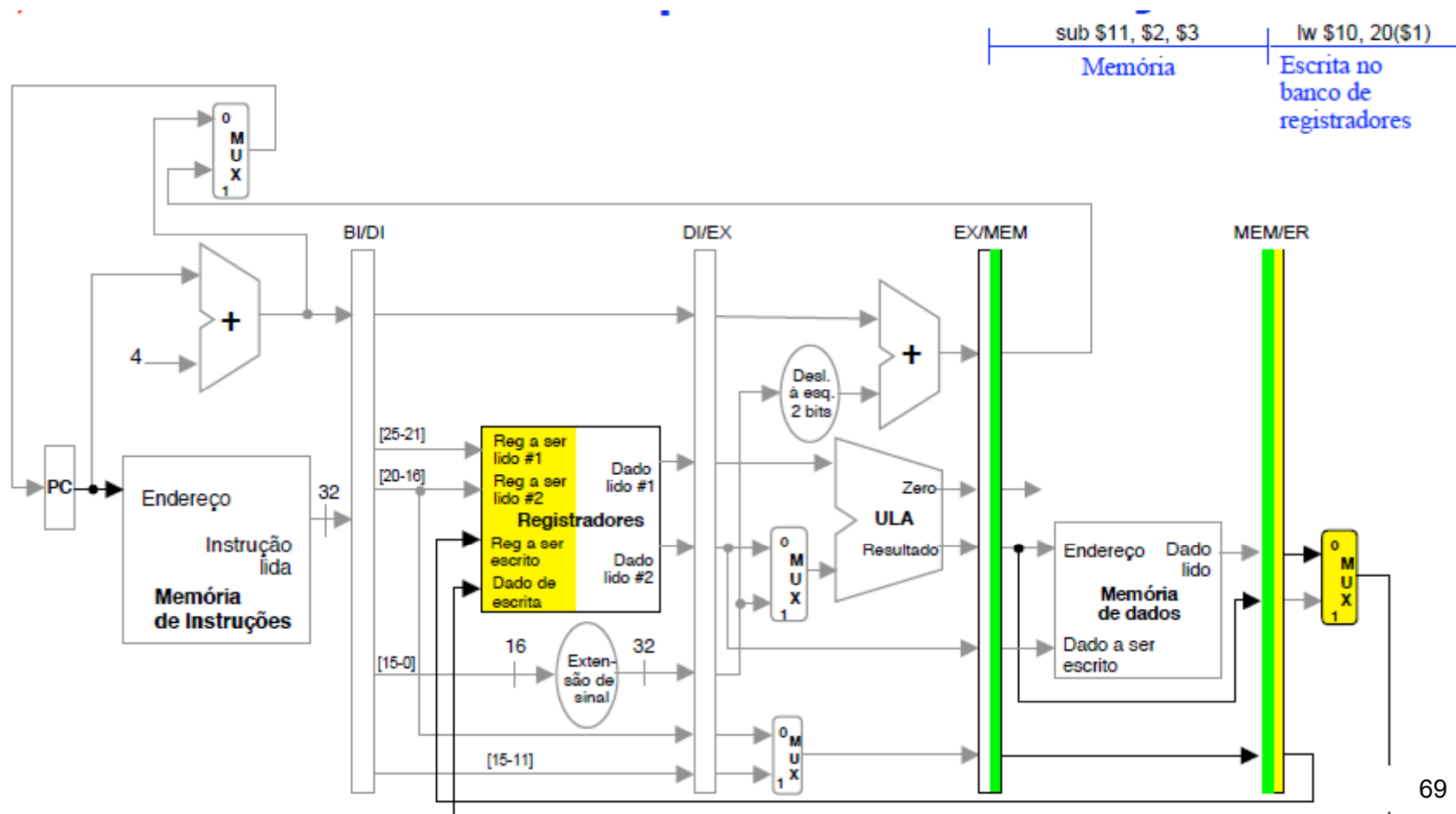
Pipelining para MIPS

□ Executando `lw $10, 20($1)`
`sub $11, $2, $3`



Pipelining para MIPS

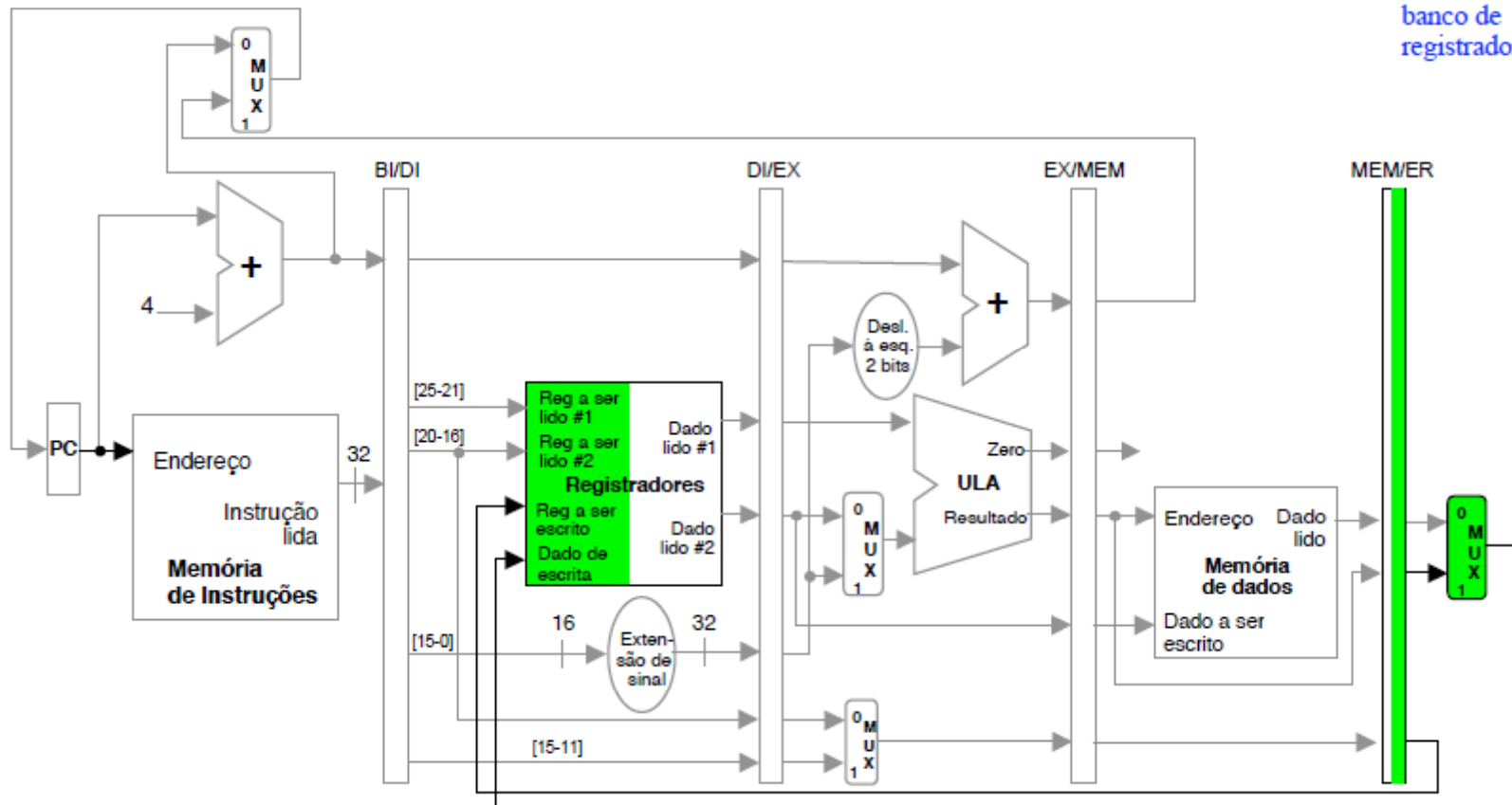
□ Executando `lw $10, 20($1)`
`sub $11, $2, $3`



Pipelining para MIPS

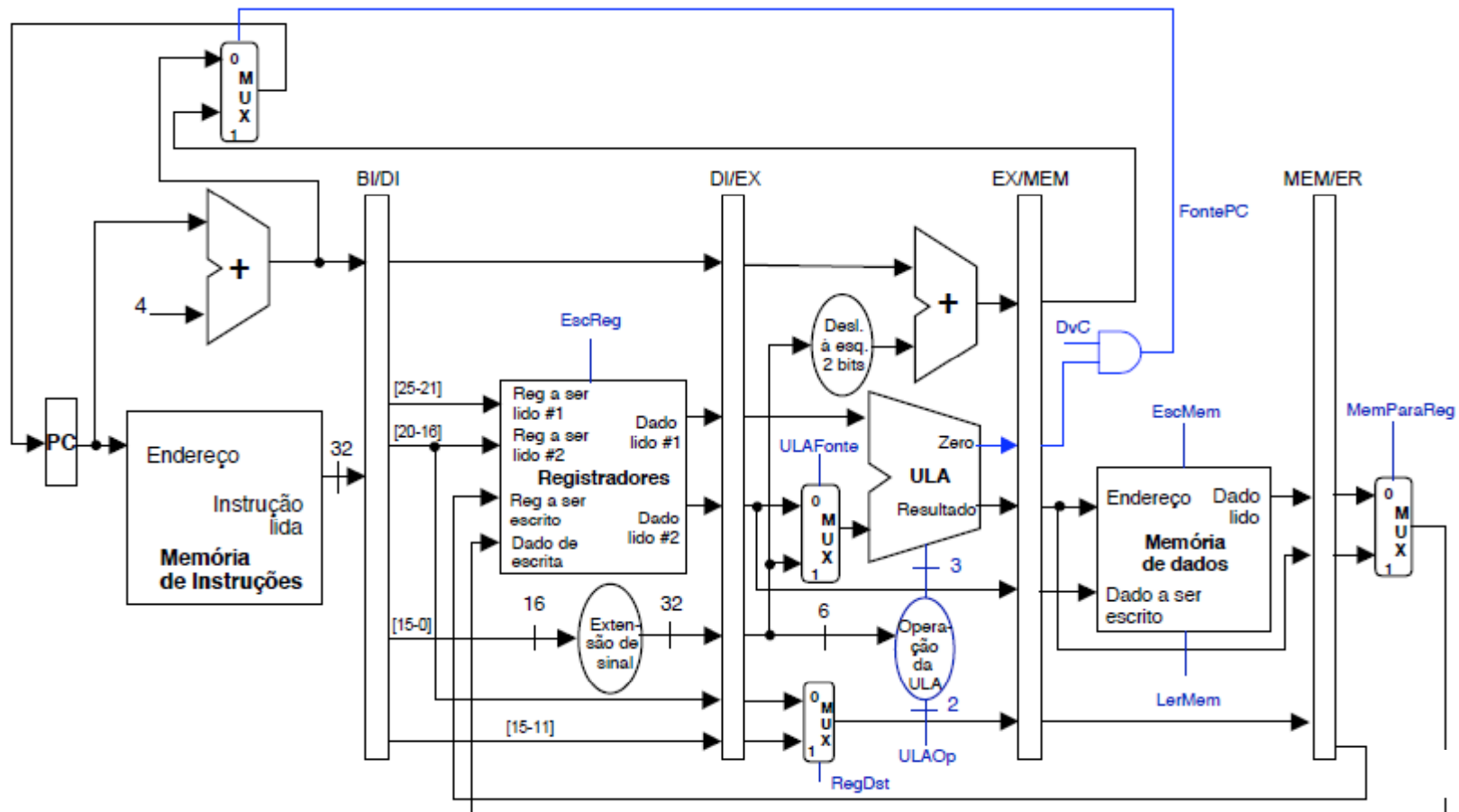
□ Executando `lw $10, 20($1)`
`sub $11, $2, $3`

`sub $11, $2, $3`
Escrita no
banco de
registradores



Pipelining para MIPS

□ Sinais de Controle



Pipelining para MIPS

□ Unidade de Controle

- ▣ **IF**: os sinais de controle para ler a instrução e para escrever em PC estão sempre ativos.
- ▣ **ID**: não há linhas de controle opcionais, pois a operação neste estágio sempre é a mesma, independentemente da instrução.
- ▣ **EX**: os sinais que precisam ser ajustados são RegDst, ALUOp e ALUSrc.
- ▣ **MEM**: as linhas de controle deste estágio são Branch, MemRead e MemWrite.
- ▣ **WB**: as linhas de controle deste estágio são MemtoReg, que decide entre enviar o resultado da ALU ou um valor de memória para o arquivo de registradores, e RegWrite, que escreve o valor selecionado.

Pipelining para MIPS

□ Unidade de Controle

- É possível aproveitar ao máximo os sinais de controle do MIPS monociclo, e a mesma lógica de controle para:
 - A ULA
 - O desvio condicional
 - O multiplexador que controla a fonte do dado do registrador destino
 - E demais linhas apresentadas anteriormente
- Os sinais de controle são essencialmente os mesmos do MIPS monociclo
- A única particularidade é que eles precisam “**viajar**” pelos estágios juntamente com a instrução

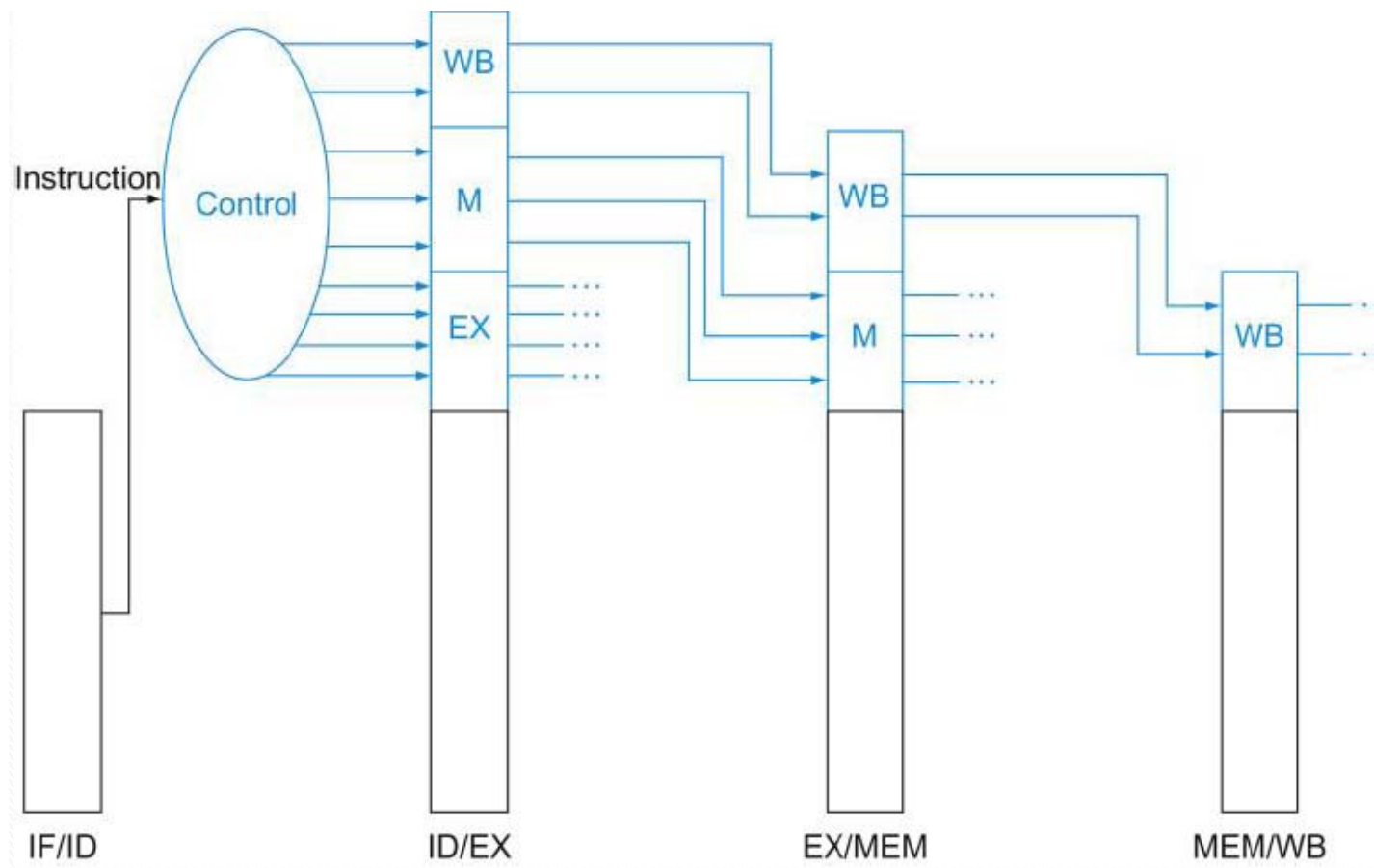
Pipelining para MIPS

□ Unidade de Controle

	Sinais de Controle a serem usados no estágio de execução/cálculo do endereço				Sinais de Controle a serem usados no estágio de acesso à memória			Sinais de Controle a serem usados no estágio de escrita no banco de registradores	
instrução	RegDst	ULAOp1	ULAOp0	ULAFonte	DvC	LerMem	EscMem	EscReg	MemParaReg
Tipo R	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

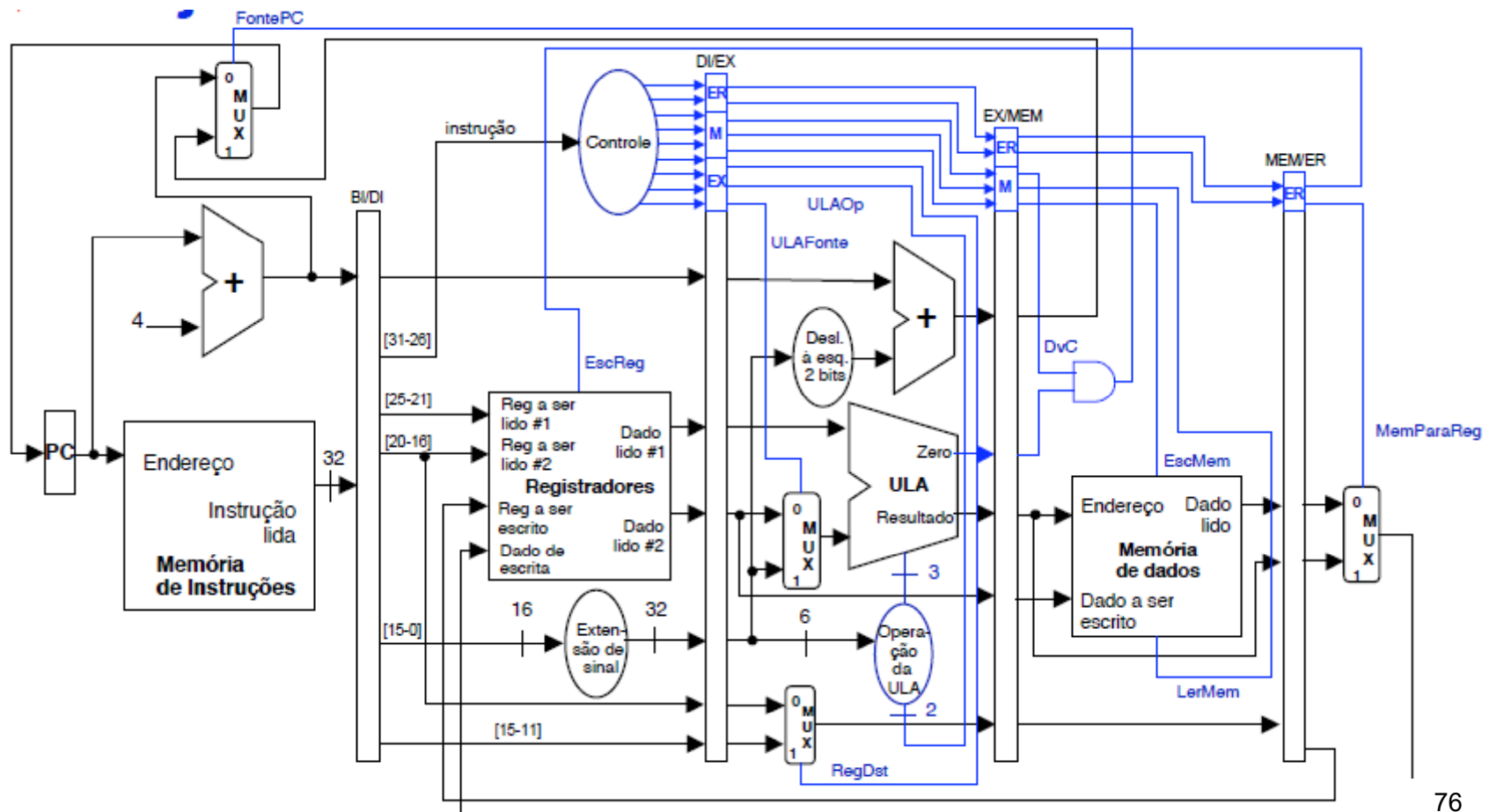
Pipelining para MIPS

□ Sinais de Controle



Pipelining para MIPS

□ Sinais de Controle



Conclusão

- Pipelining: forma efetiva de se melhorar o desempenho de microprocessadores.
 - ▣ Diminui o CPI
 - ▣ Diminui o tempo de um ciclo de clock
- Arquiteturas RISC: facilitam a implementação de pipelining;
- Uso generalizado em toda arquitetura projetada nos últimos 20 anos;
- Problemas associados com pipelining
 - ▣ Structural Hazards
 - ▣ Data Hazards
 - ▣ Controle mais complexo