Aula 4: Processador - Caminho de Dados e Controle - Parte III

Professor(a): Virgínia Fernandes Mota
http://www.dcc.ufmg.br/∼virginiaferm

OCS (TEORIA) - SETOR DE INFORMÁTICA

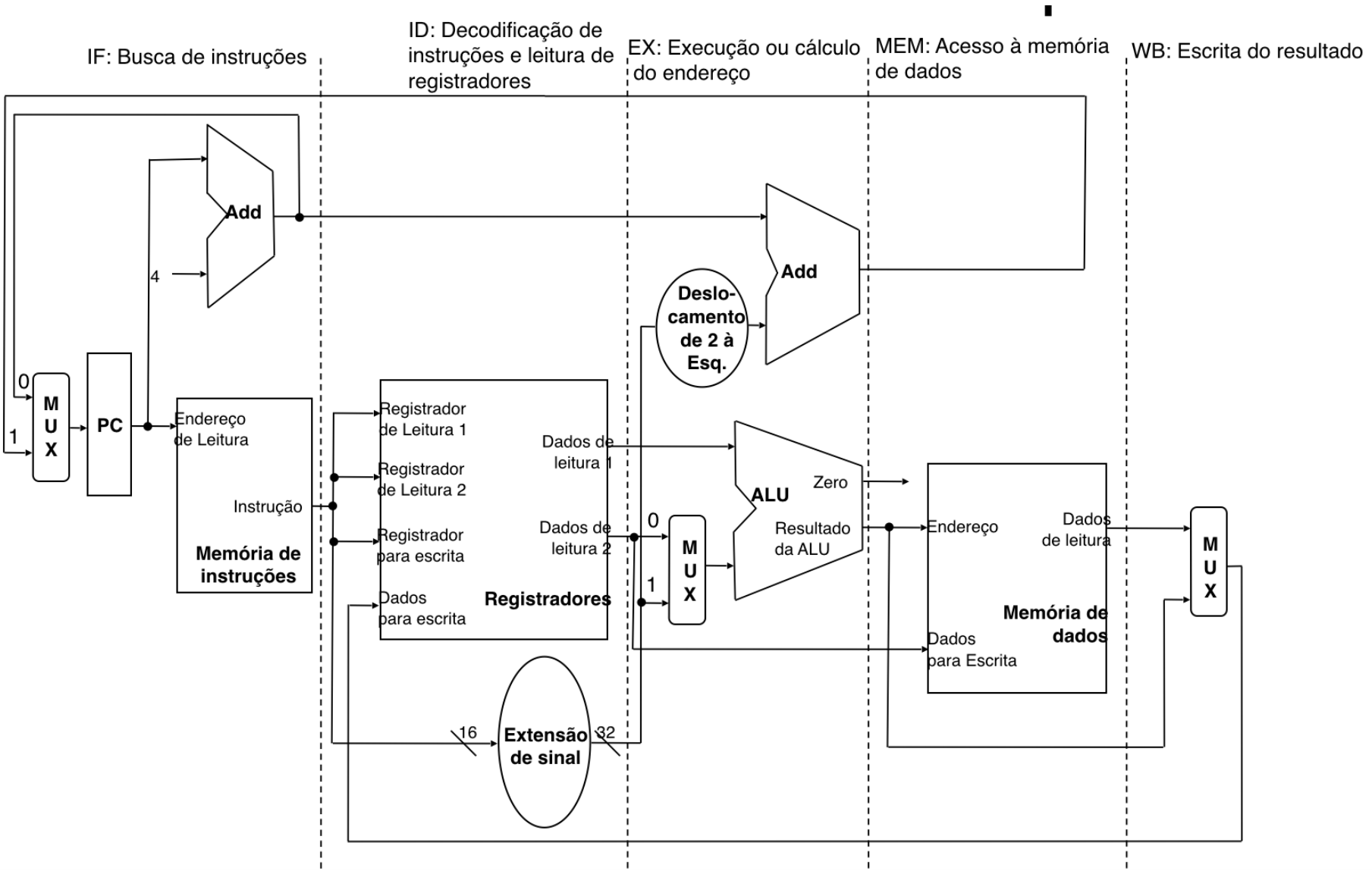
Caminho de Dados e Controle

- Caminho de dados usando Pipeline
- Controle de um Pipeline
- Hazard real

Um caminho de dados usando Pipeline

- Instrução dividida em cinco estágios: Pipeline de cinco estágios
  - IF (Instruction Fetch): Busca de instruções
  - ID (Instruction Decode): Decodificação de instruções e leitura de registradores
  - EX: Execução ou cálculo do endereço
  - MEM: Acesso à memória de dados
  - WB (Write Back): Escrita do resultado

Um caminho de dados usando Pipeline



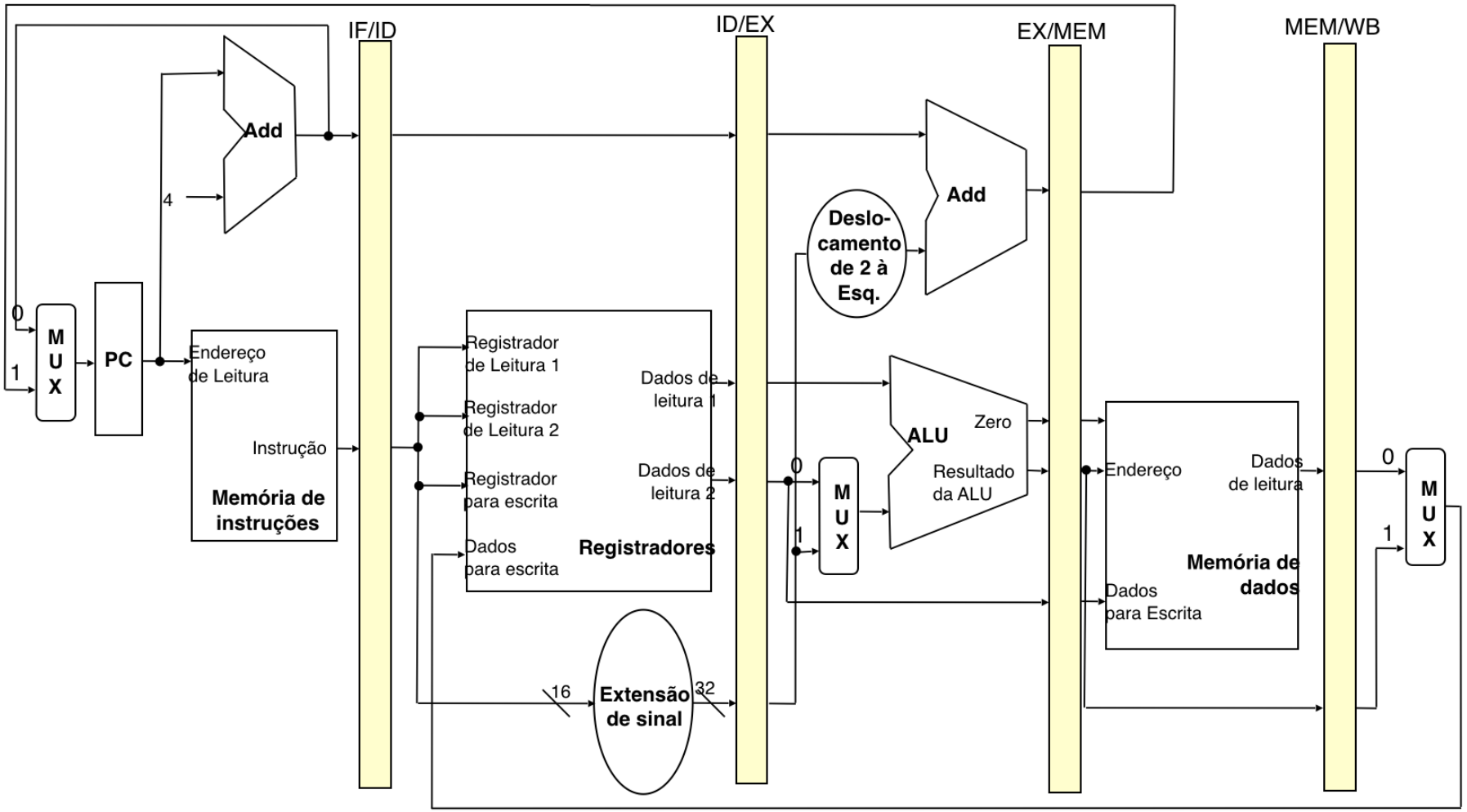
Um caminho de dados usando Pipeline

- Instruções se movem da esquerda para direita enquanto completam operação
- Duas exceções
  - Escrita de resultado no banco de registradores
  - Seleção do próximo valor do PC
- Fluxo da direita para esquerda não afeta instrução atual
  - Somente instrução seguinte

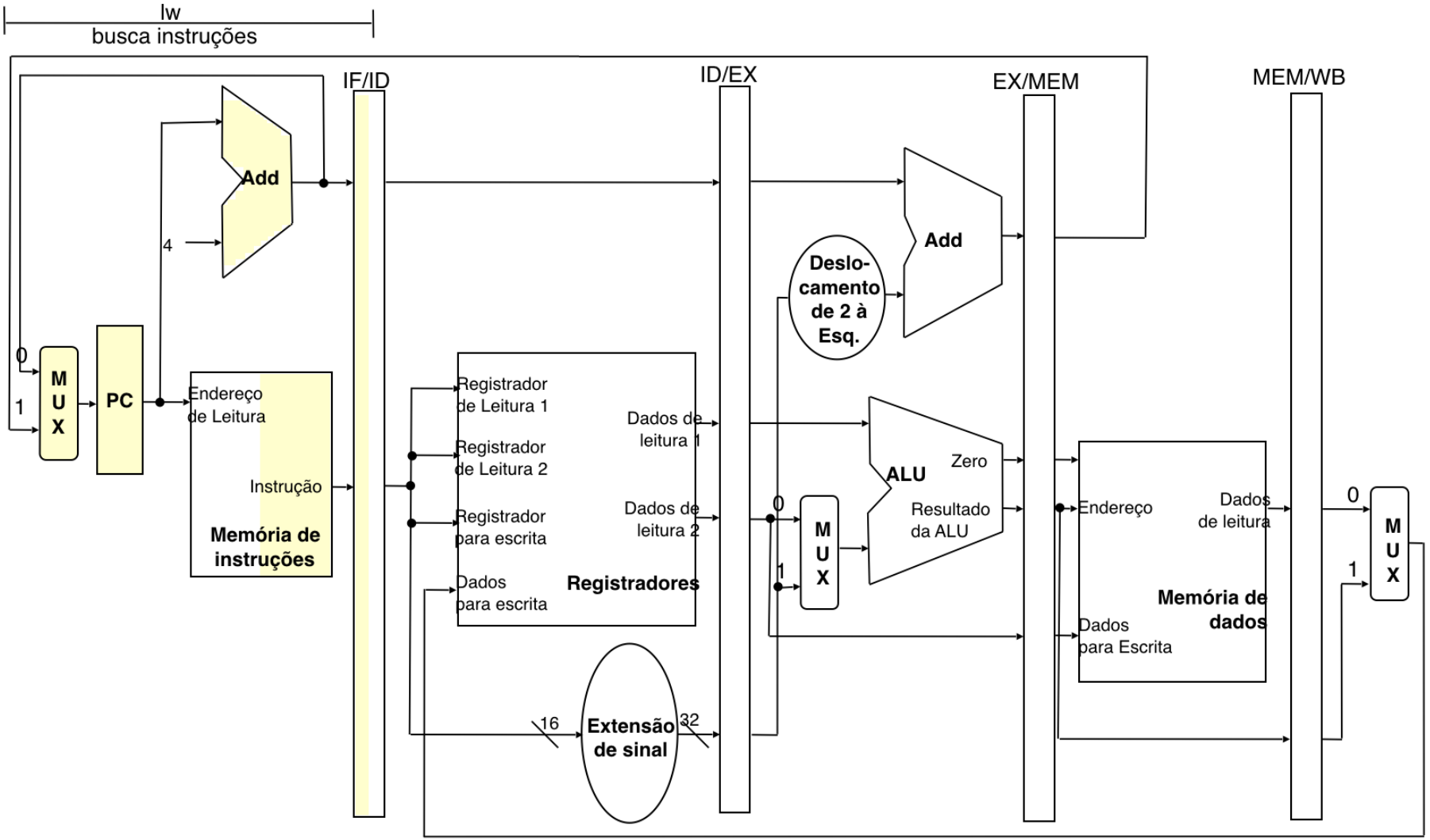
Um caminho de dados usando Pipeline

- Para reter o valor de uma instrução individual para seus outros quatro estágios
  - Registradores
  - Instruções avançam entre registradores
  - Registradores recebem o nome dos estágios separados por esses registradores
  - Precisam ser grandes o suficiente para armazenar todos os dados correspondentes às linhas que passam por eles

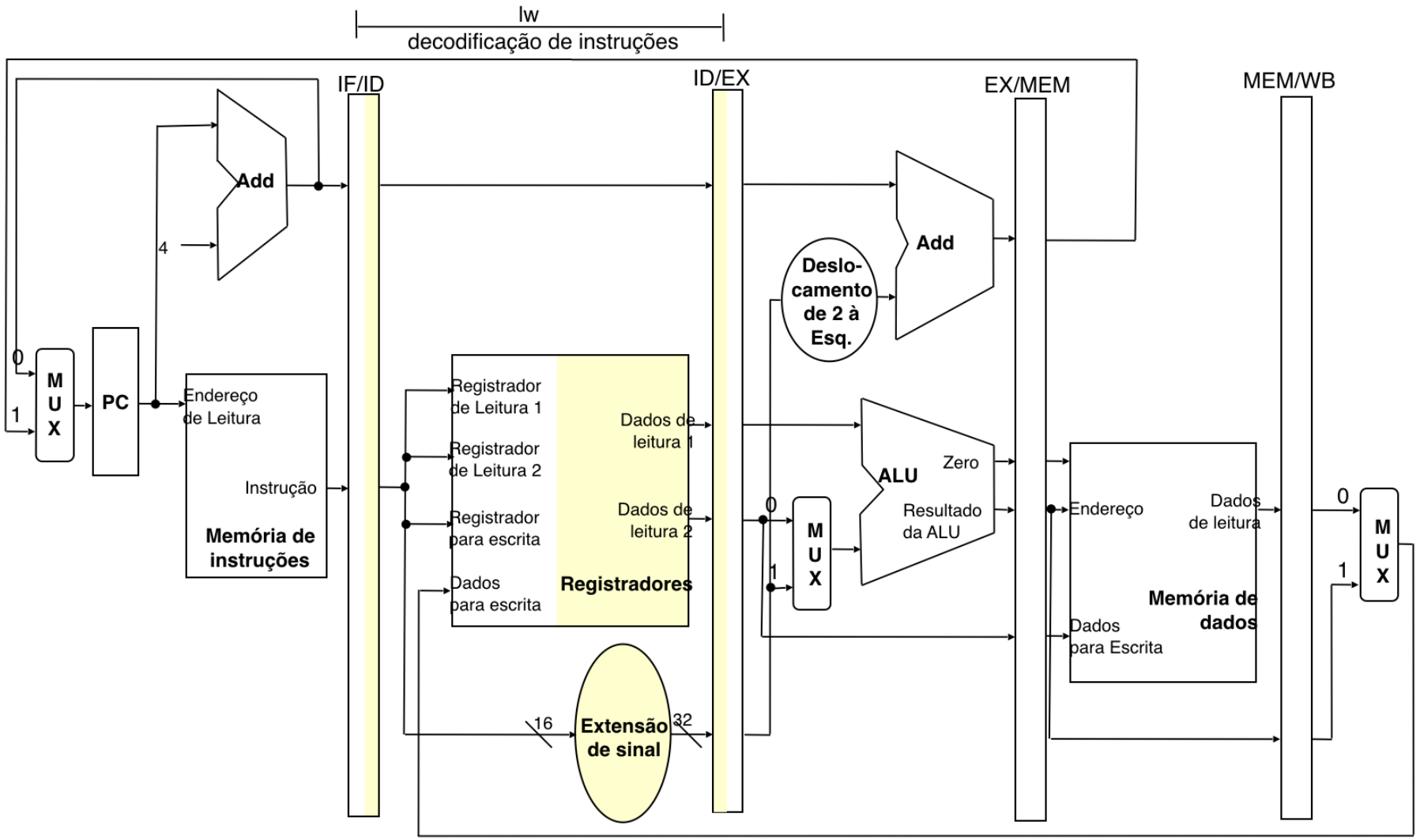
Um caminho de dados usando Pipeline



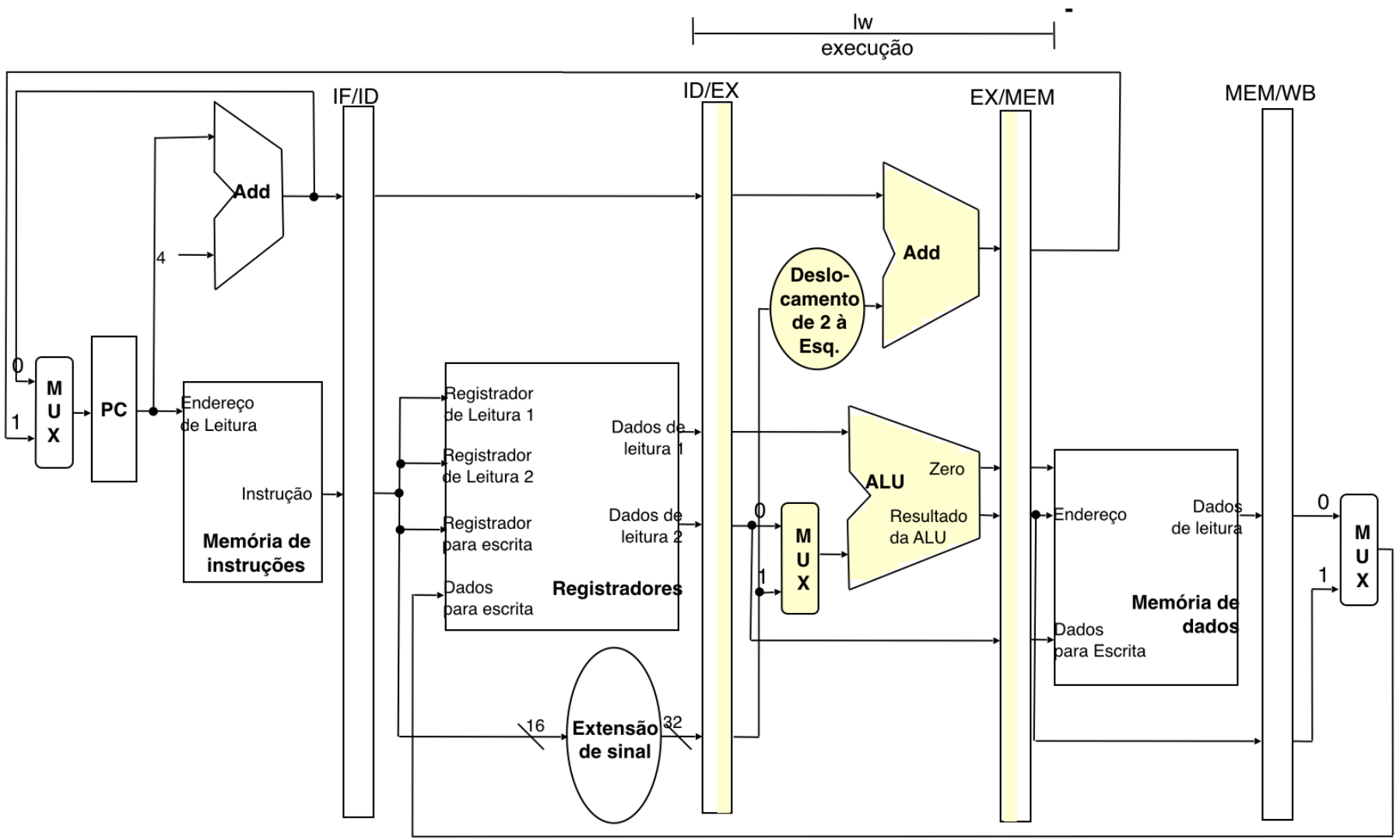
Um caminho de dados usando Pipeline



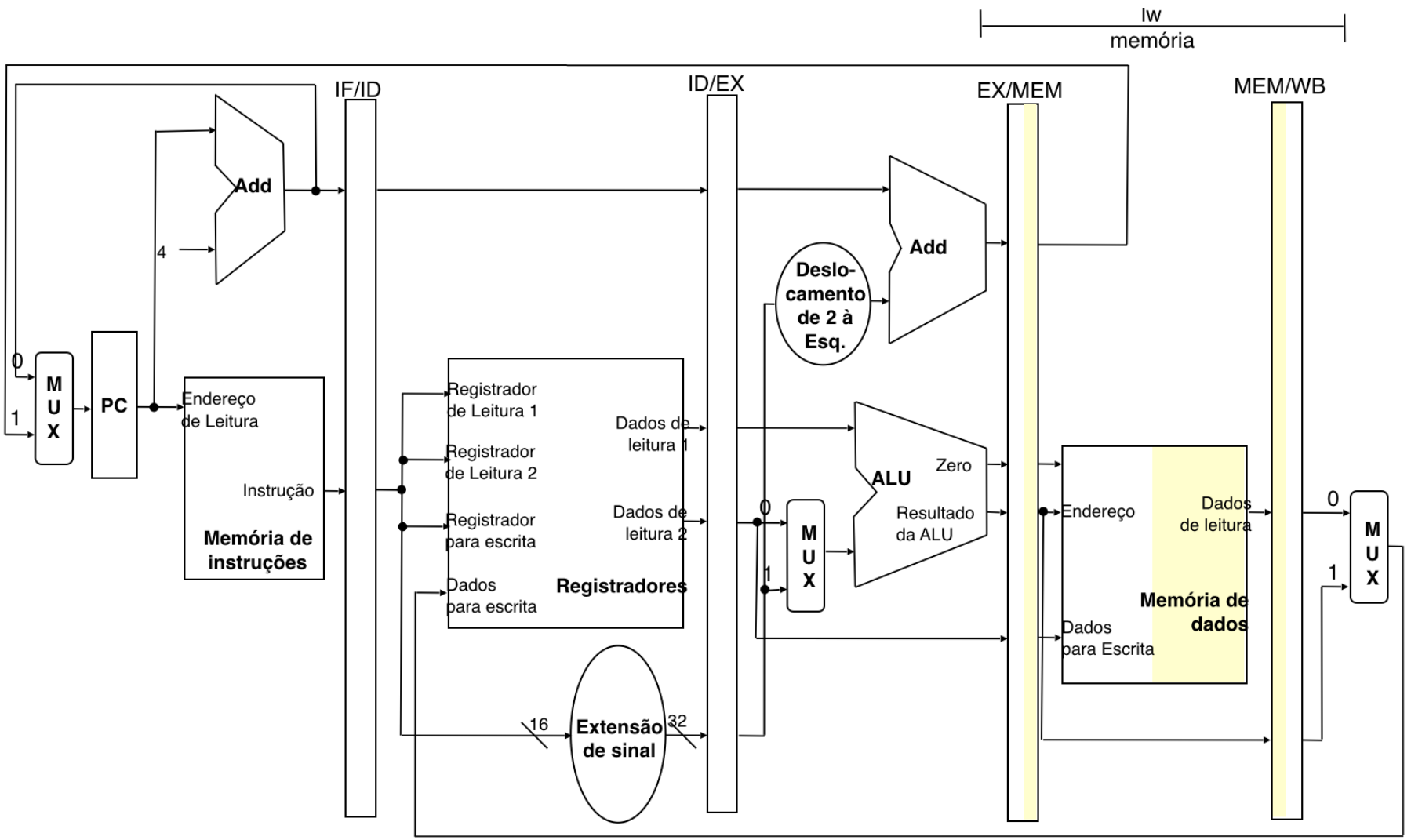
Um caminho de dados usando Pipeline



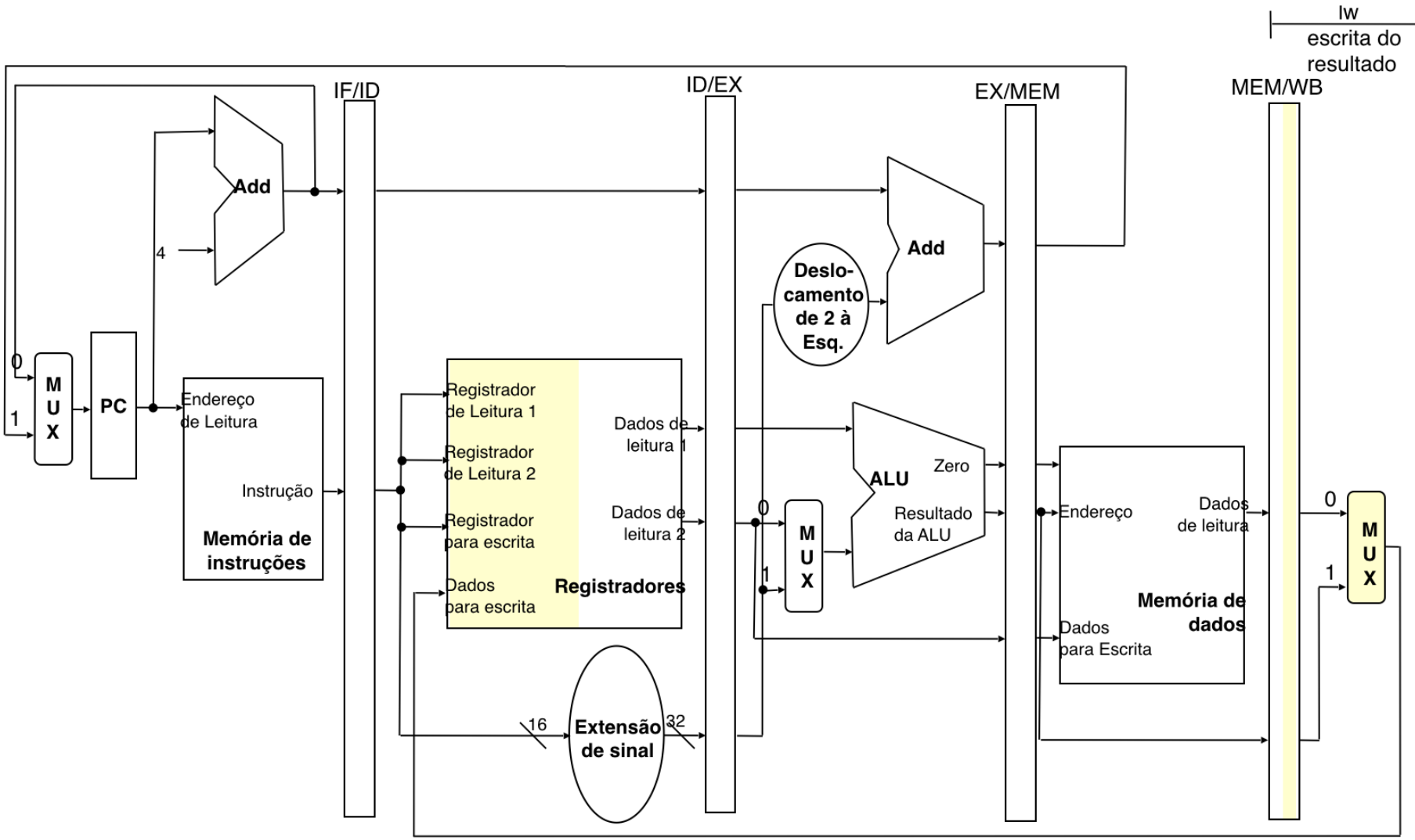
Um caminho de dados usando Pipeline



Um caminho de dados usando Pipeline



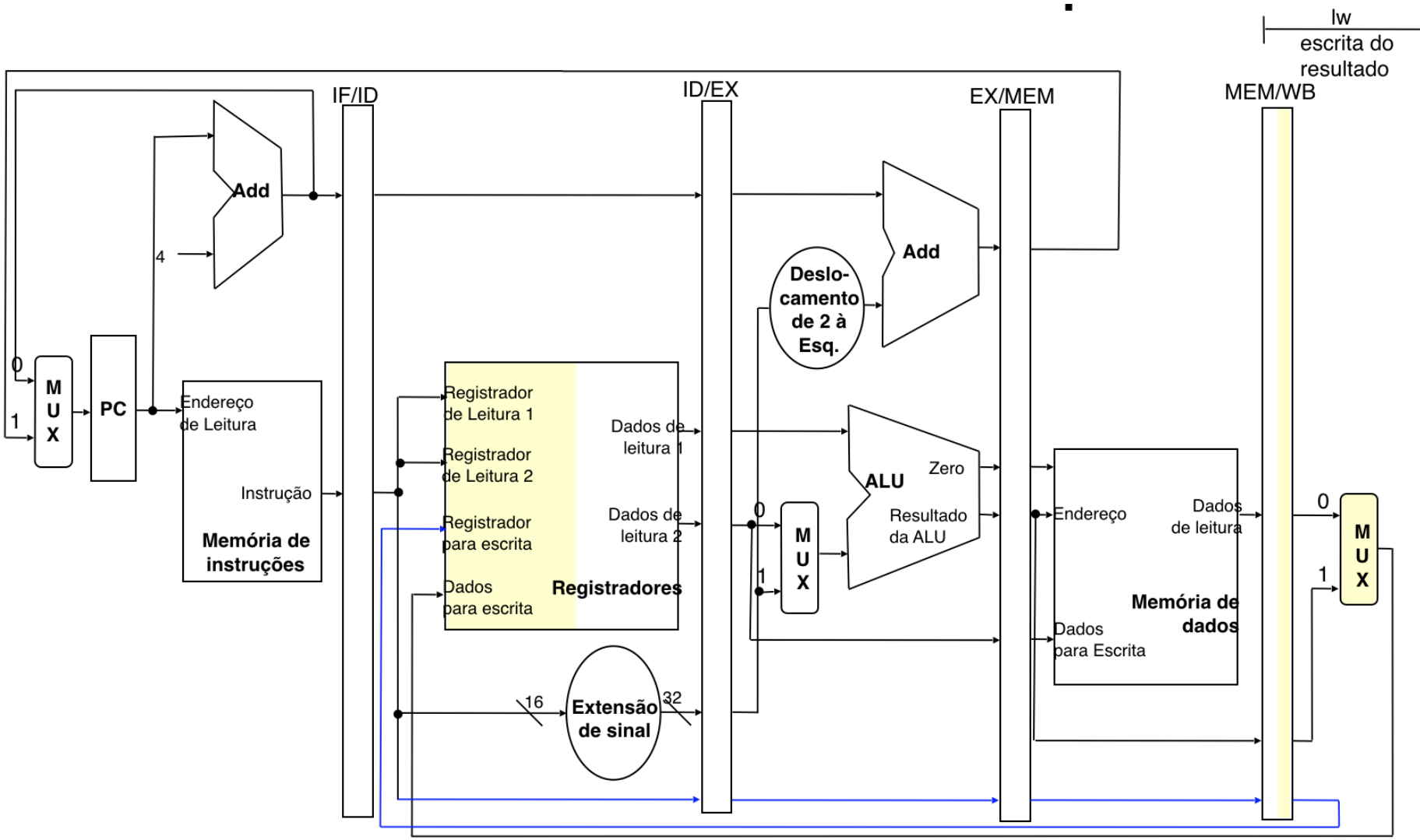
Um caminho de dados usando Pipeline



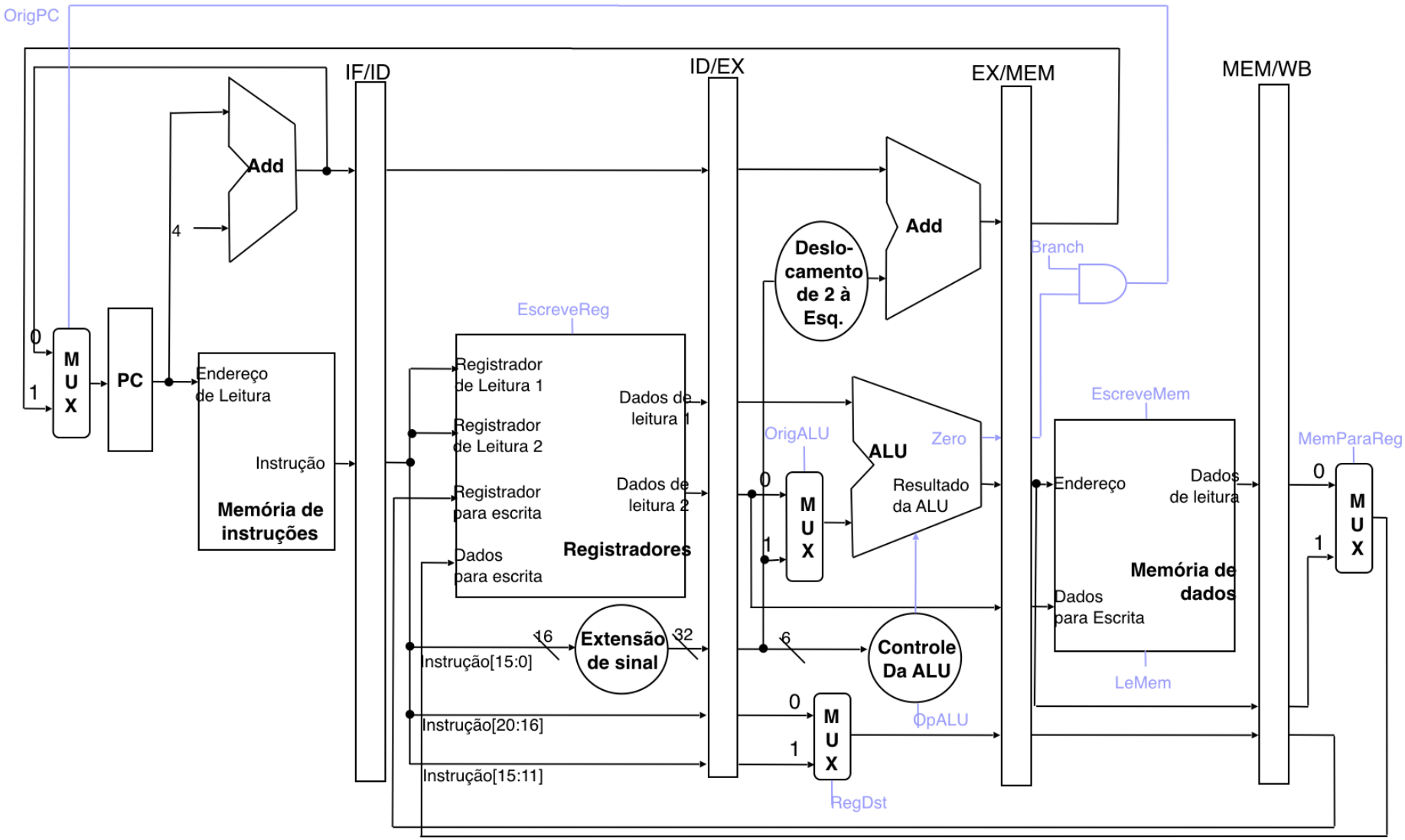
Um caminho de dados usando Pipeline

- Para passar algo de um estágio anterior do Pipeline para um posterior, informação precisa ser colocada em um registrador
  - Caso contrário, informação é perdida quando próxima instrução entrar nesse estágio
- Cada componente lógico do caminho de dados só pode ser usado dentro de um único ciclo do Pipeline
  - Caso contrário, hazard estrutural
- Pipeline anterior contêm bug! O que falta?

Um caminho de dados usando Pipeline



Um caminho de dados usando Pipeline



Controle de um Pipeline

Opcode	OpALU	Operação da Instr.	Campo *funct*	Ação Desejada	Entrada de Controle
LW	00	Load word	XXXXXX	Add	0010
SW	00	Store word	XXXXXX	Add	0010
BEQ	01	Branch equal	XXXXXX	Subtract	0110
TIPO R	10	Add	100000	Add	0010
TIPO R	10	Subtract	100010	Subtract	0110
TIPO R	10	And	100100	And	0000
TIPO R	10	Or	100101	Or	0001
TIPO R	10	Set on less than	101010	SLT	0111

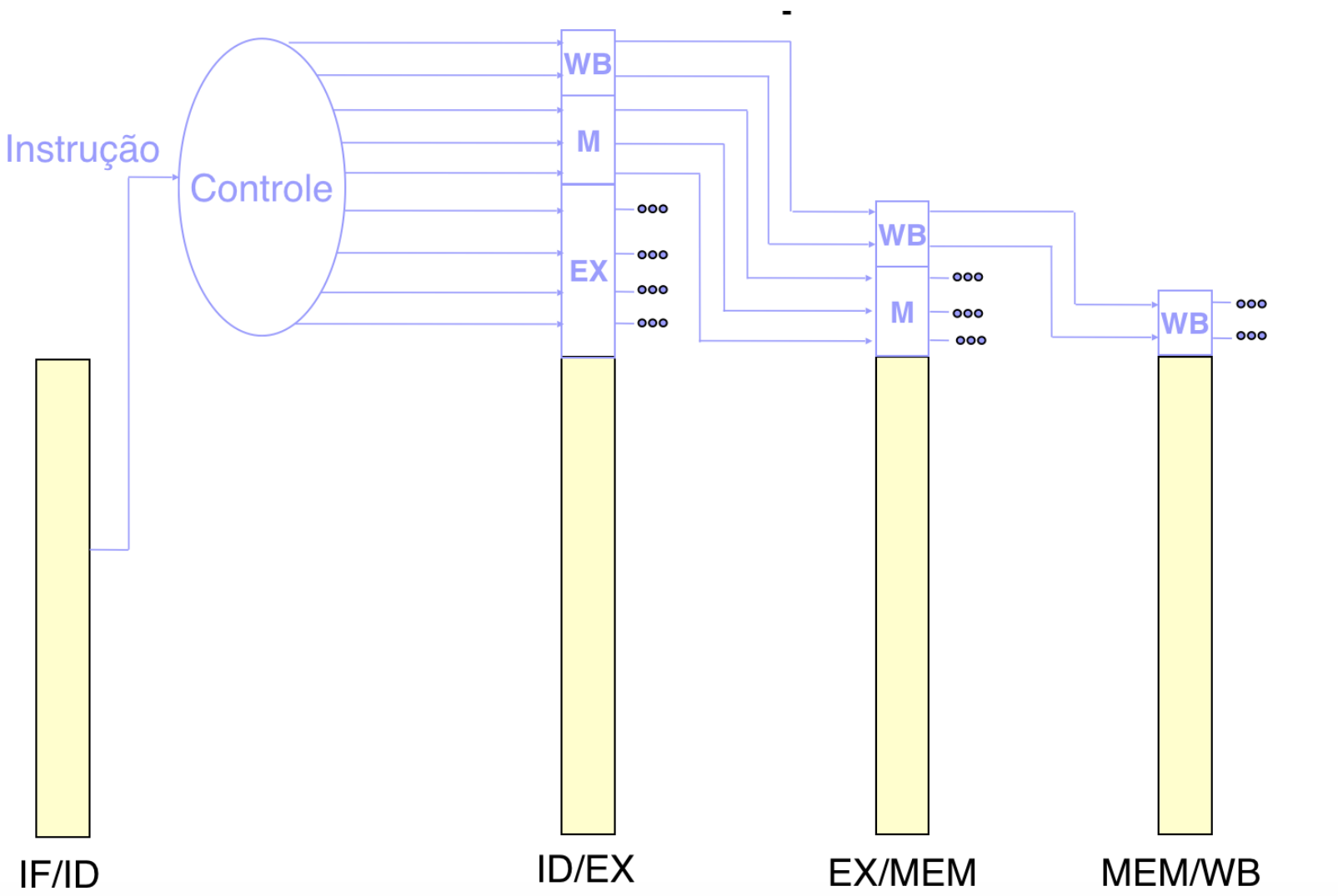
Controle de um Pipeline

Sinal	Efeito quando Inativo	Efeito quando Ativo
RegDst	Entrada para “Registrador para Escrita” vem de [20:16]	Entrada para “Registrador para Escrita” vem de [15:11]
EscreveReg	Nenhum	Registrador “Registrador para Escrita” é escrito com “Dados para Escrita”
OrigALU	Segundo operando de ALU vem do banco de registradores.	Segundo operando de ALU vem do sinal estendido.
OrigPC	PC = PC + 4	PC = desvio
LeMem	Nenhum	Valor lido de “Endereço” é colocado em “Dados da Leitura”
EscreveMem	Nenhum	Valor de “Dados para Escrita” é escrito em “Endereço”
MemParaReg	Valor de “Dados para Escrita” vem da ALU	Valor de “Dados para Escrita” vem da memória de dados

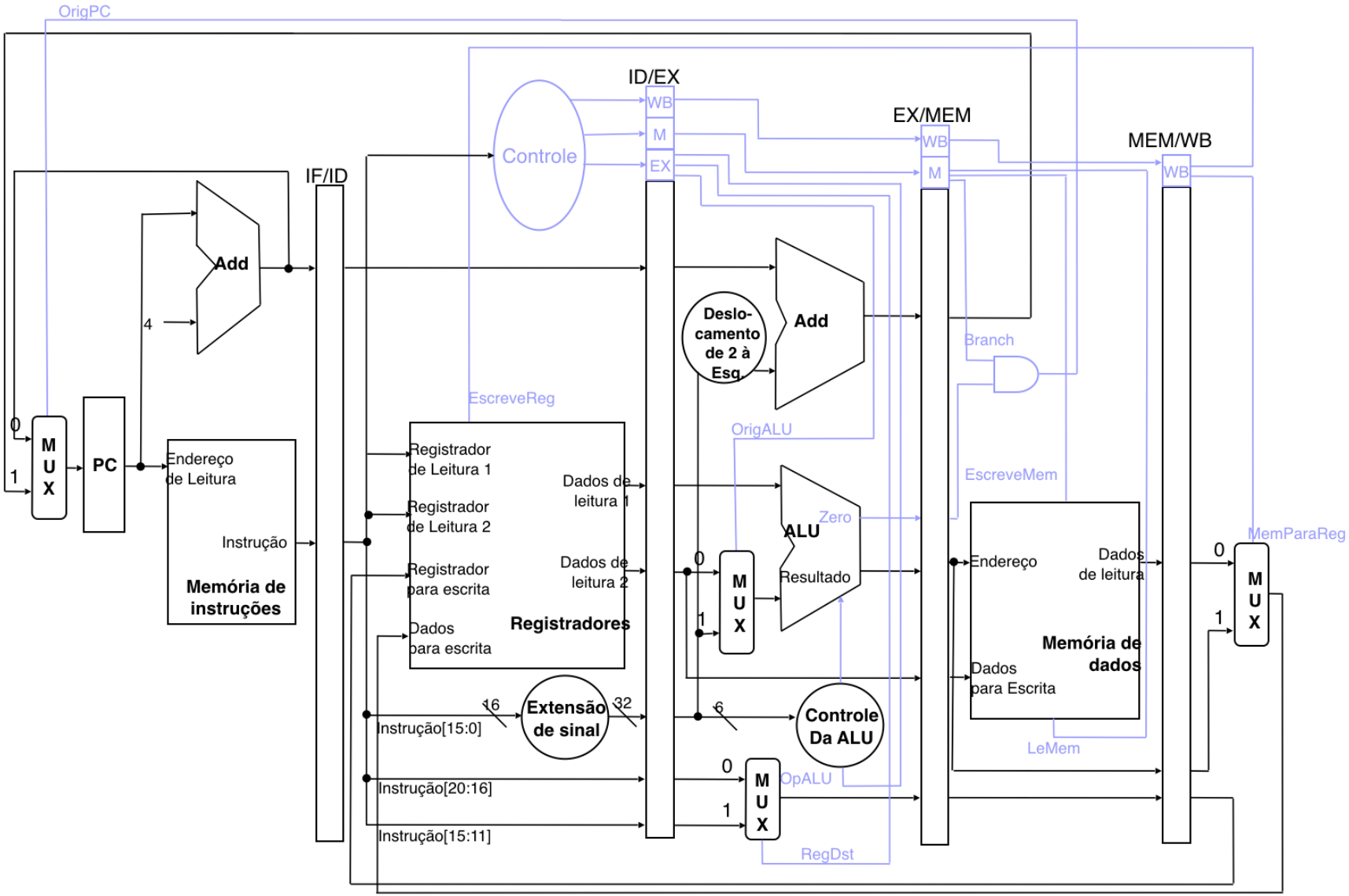
Controle de um Pipeline

Instrução	Linhas de controle do estágio de execução/ cálculo de endereço				Linhas de controle do estágio de acesso à memória			Linhas de controle do estágio de escrita do resultado	
	RegDSt	ALUOp1	ALUOp0	OrigALU	Branch	LeMem	EscreveMem	EscreveReg	MemParaReg
Tipo R	1	1	0	0	0	0	0	1	1
lw	0	0	0	1	0	1	0	1	0
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Controle de um Pipeline



Controle de um Pipeline



Um Pipeline mais real

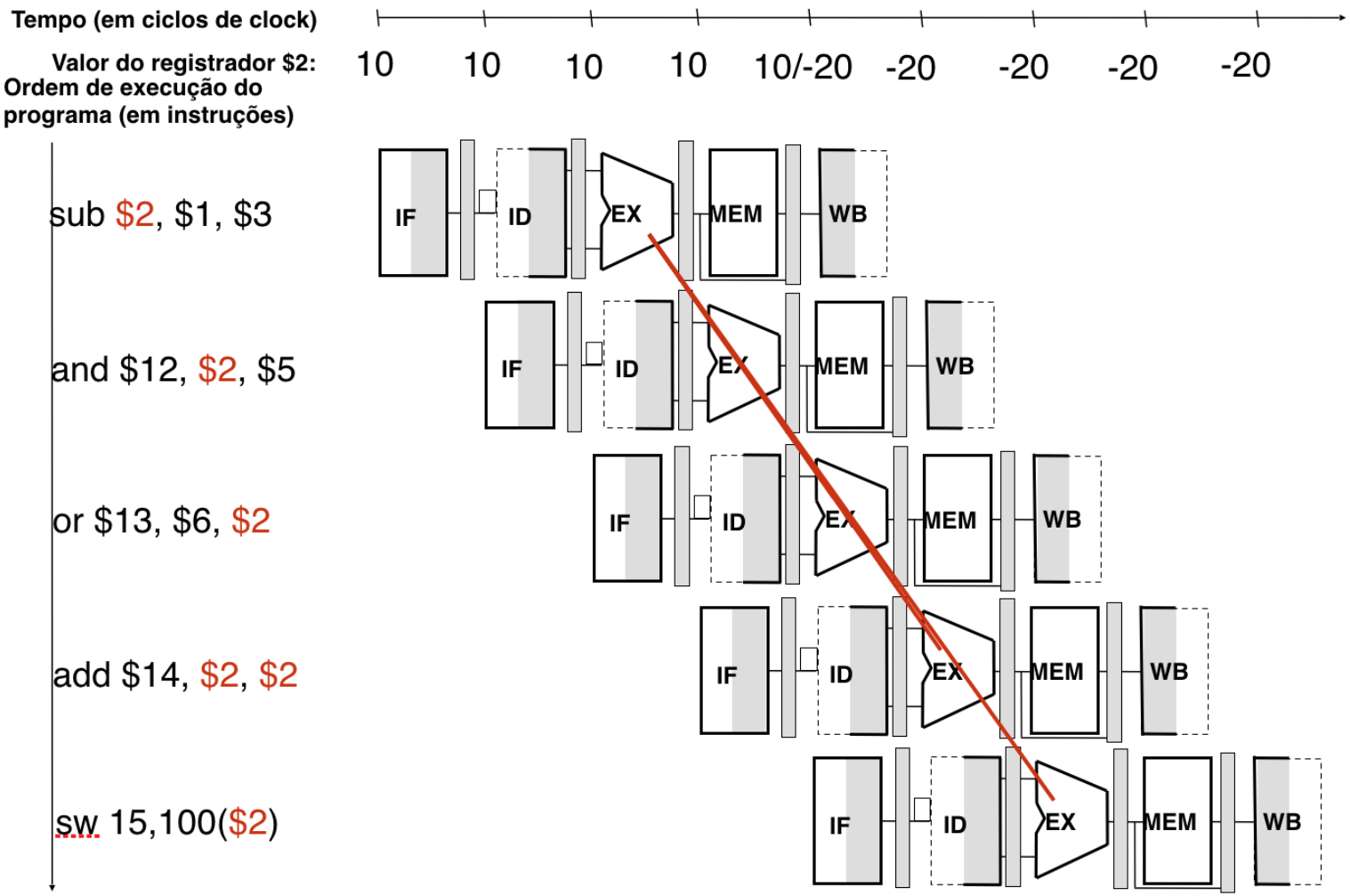
- Agora sabemos como o Pipeline funciona.
- Sabemos os problemas que podem surgir.
- Sabemos o controle dele!
- Então, o que ocorre com programas reais?

Hazard de dados e *forwarding*

```
sub $2, $1, $3 # registrador $2 escrito por sub
and $12, $2, $5 # 1o operando depende de sub
or $13, $6, $2 # 2o operando depende de sub
add $14, $2, $2 # 1o e 2o operandos dependem de sub
sw $15,100($2) # base depende de sub
```

- Como a sequência funcionaria no pipeline?
- Suponha que registrador $2 tenha 10 antes de subtract e -20 após.

Hazard de dados e *forwarding*



Hazard de dados e *forwarding*

- O que acontece quando um registrador é lido e escrito no mesmo ciclo de clock?
  - Consideraremos que a escrita está na primeira metade do ciclo de clock e a leitura está na segunda metade, de modo que a leitura fornece o que foi escrito
  - Não temos hazard de dados nessa situação
- Valores lidos para o registrador \$2 não seriam o resultado da instrução sub
  - A menos que a leitura ocorra durante o ciclo 5 ou posterior
  - add e sw $\rightarrow$ ok
  - and e or $\rightarrow$ valor incorreto
  - Ocorre quando a linhas retornam no tempo

Hazard de dados e *forwarding*

- Quando os dados estão disponíveis?
  - No final do terceiro ciclo (execução)
- Quando os dados são realmente necessários por and e or?
  - No início do quarto e quinto ciclos (execução)
- Poderemos executar esse segmento sem stall se dados sofrerem forwarding assim que estiverem disponíveis
  - Antes de estarem disponíveis para leitura no banco de registradores
- Simplificação: consideraremos apenas forwarding para uma operação no estágio EX
- Notação: "ID/EX.RegistradorRs"
  - Registrador Rs armazenado no registrador de pipeline ID/EX

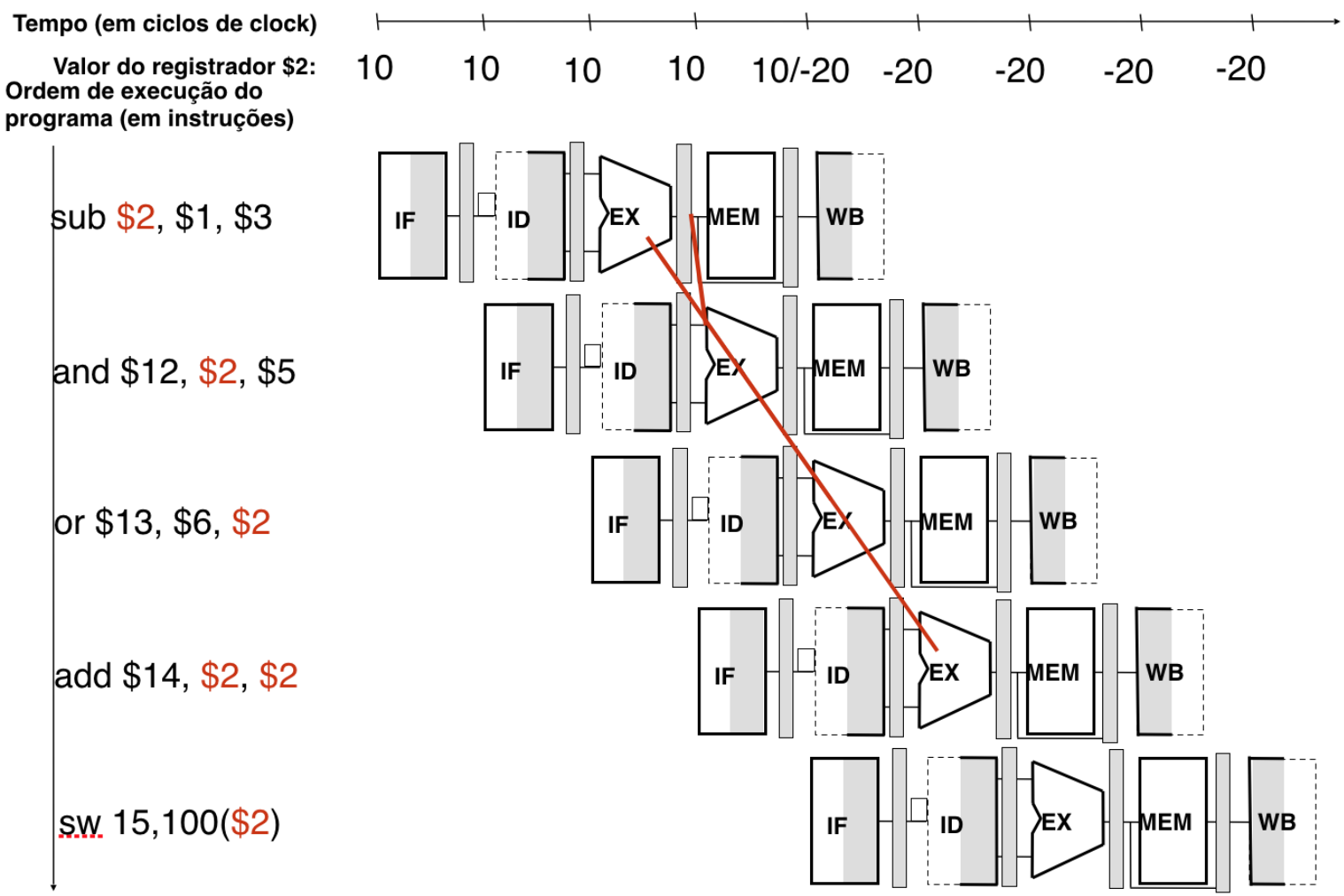
Hazard de dados e *forwarding*

- Condições para ocorrência de hazards
  1a: EX/MEM.RegistradorRd = ID/EX.RegistradorRs
  1b: EX/MEM.RegistradorRd = ID/EX.RegistradorRt
  2a: MEM/WB.RegistradorRd = ID/EX.RegistradorRs
  2b: MEM/WB.RegistradorRd = ID/EX.RegistradorRt
- Exs:
  sub $2, $1, $3 # registrador $2 (rd) escrito por sub
  and $12, $2, $5 # 1o operando (rs) depende de sub
  or $13, $6, $2 # 2o operando (rt) depende de sub
  add $14, $2, $2 # não há hazard
  sw $15,100($2) # não há hazard
  Primeiro hazard: tipo 1a; segundo hazard: tipo 2b

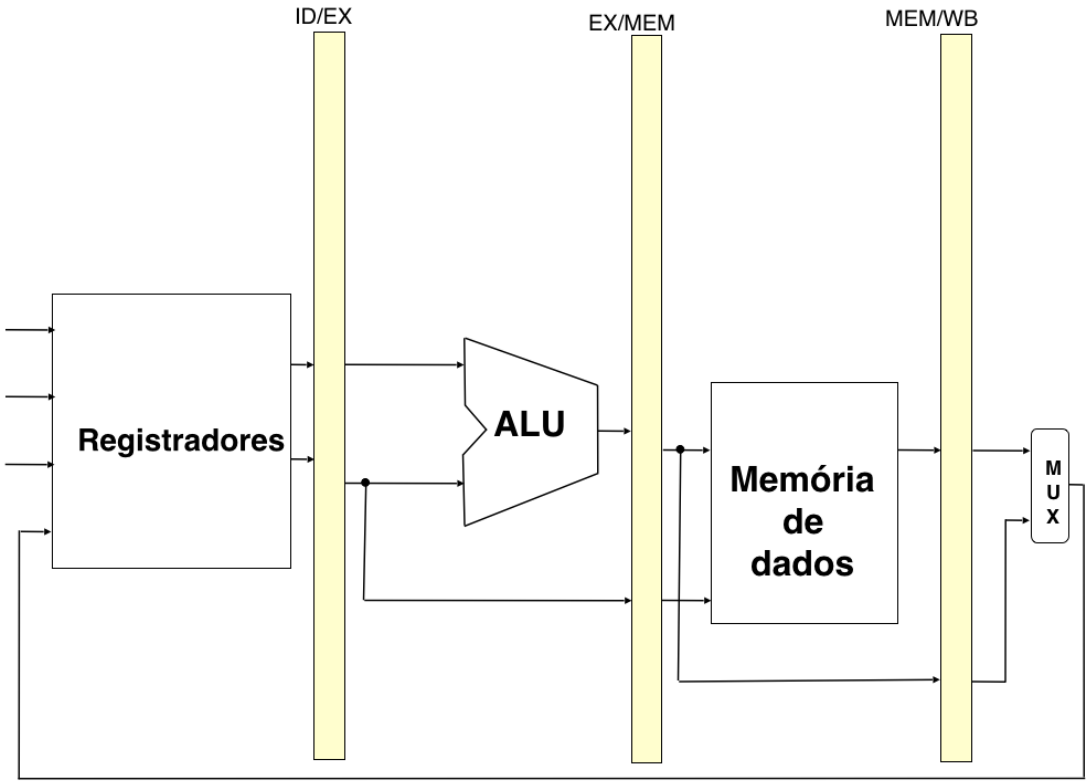
Hazard de dados e *forwarding*

- Política não exata
  - Algumas instruções não escrevem em registradores
  - MIPS: Não faz-se forwarding quando $0 como destino (rd)
- Solução:
  - Verificar se EscreveReg ativo no campo de controle WB
  - Verificar se registrador destino diferente de $0

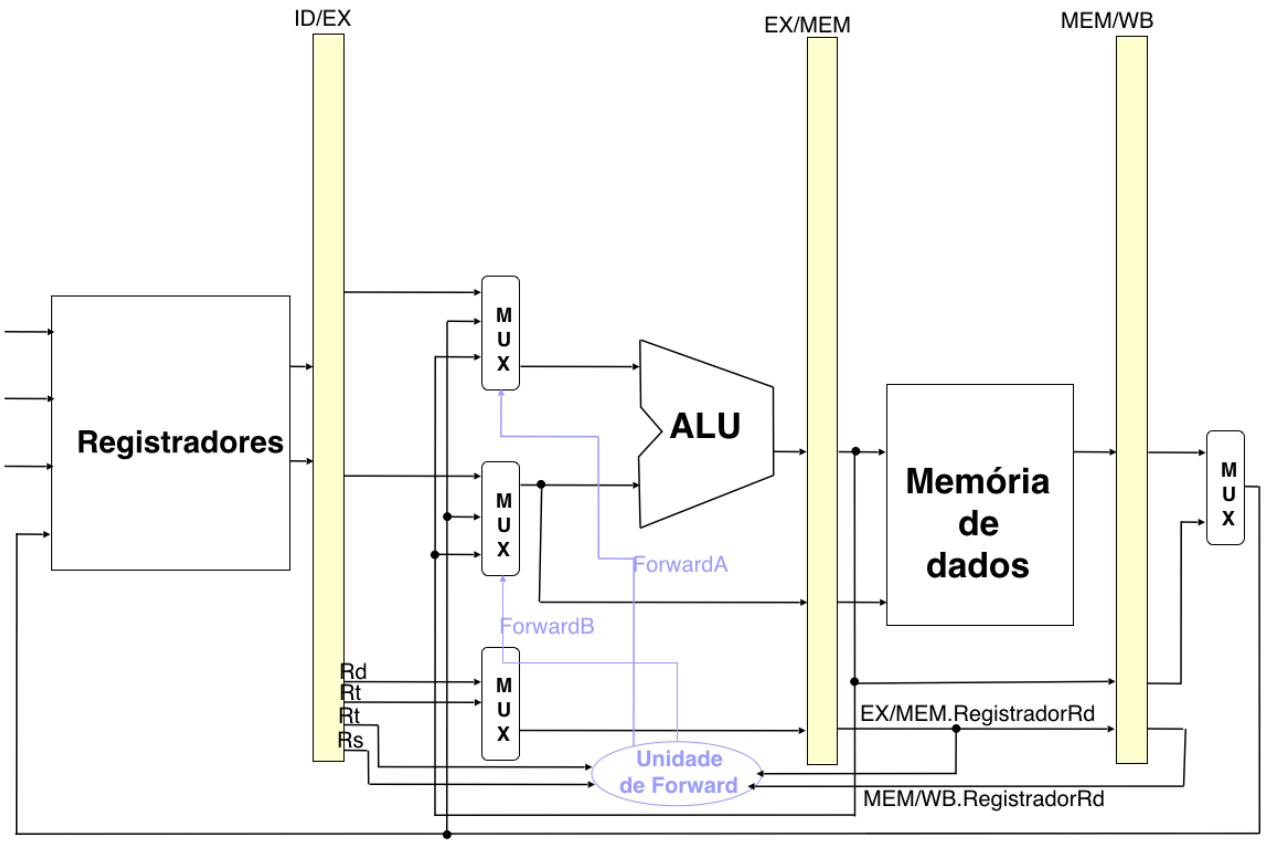
Hazard de dados e *forwarding*



Hazard de dados e *forwarding*



Hazard de dados e *forwarding*



Hazard de dados e *forwarding*

Controle do MUX	Origem	Explicação
ForwardA=00	ID/EX	Primeiro operando vem do banco de registradores
ForwardA=10	EX/MEM	Primeiro operando sofre forwarding do resultado anterior da ALU
ForwardA=01	MEM/WB	Primeiro operando sofre forwarding da memória ou de resultado anterior da ALU
ForwardB=00	ID/EX	Segundo operando vem do banco de registradores
ForwardB=10	EX/MEM	Segundo operando sofre forwarding do resultado anterior da ALU
ForwardB=01	MEM/WB	Segundo operando sofre forwarding da memória ou de resultado anterior da ALU

Hazard de dados e *forwarding*

- Condições para detectar hazards e sinais de controle para resolvê-los
- Hazard EX:
  if ((EX/MEM.EscreveReg)
  and (EX/MEM.RegistradorRd != 0)
  and (EX/MEM.RegistradorRd == ID/EX.RegistradorRs)) ForwardA = 10
  if ((EX/MEM.EscreveReg)
  and (EX/MEM.RegistradorRd != 0)
  and (EX/MEM.RegistradorRd == ID/EX.RegistradorRt)) ForwardB = 10

Hazard de dados e *forwarding*

- Condições para detectar hazards e sinais de controle para resolvê-los
- Hazard MEM:
  if ((MEM/WB.EscreveReg)
  and (MEM/WB.RegistradorRd != 0)
  and (MEM/WB.RegistradorRd == ID/EX.RegistradorRs)) ForwardA = 01
  if ((MEM/WB.EscreveReg)
  and (MEM/WB.RegistradorRd != 0)
  and (MEM/WB.RegistradorRd == ID/EX.RegistradorRt)) ForwardB = 01

Hazard de dados e *forwarding*

- Problema se resultado da instrução no estágio WB, o resultado da instrução no estágio MEM e o operando de origem da instrução no estágio ALU forem iguais!
- Ex: Soma de vetor em único registrador

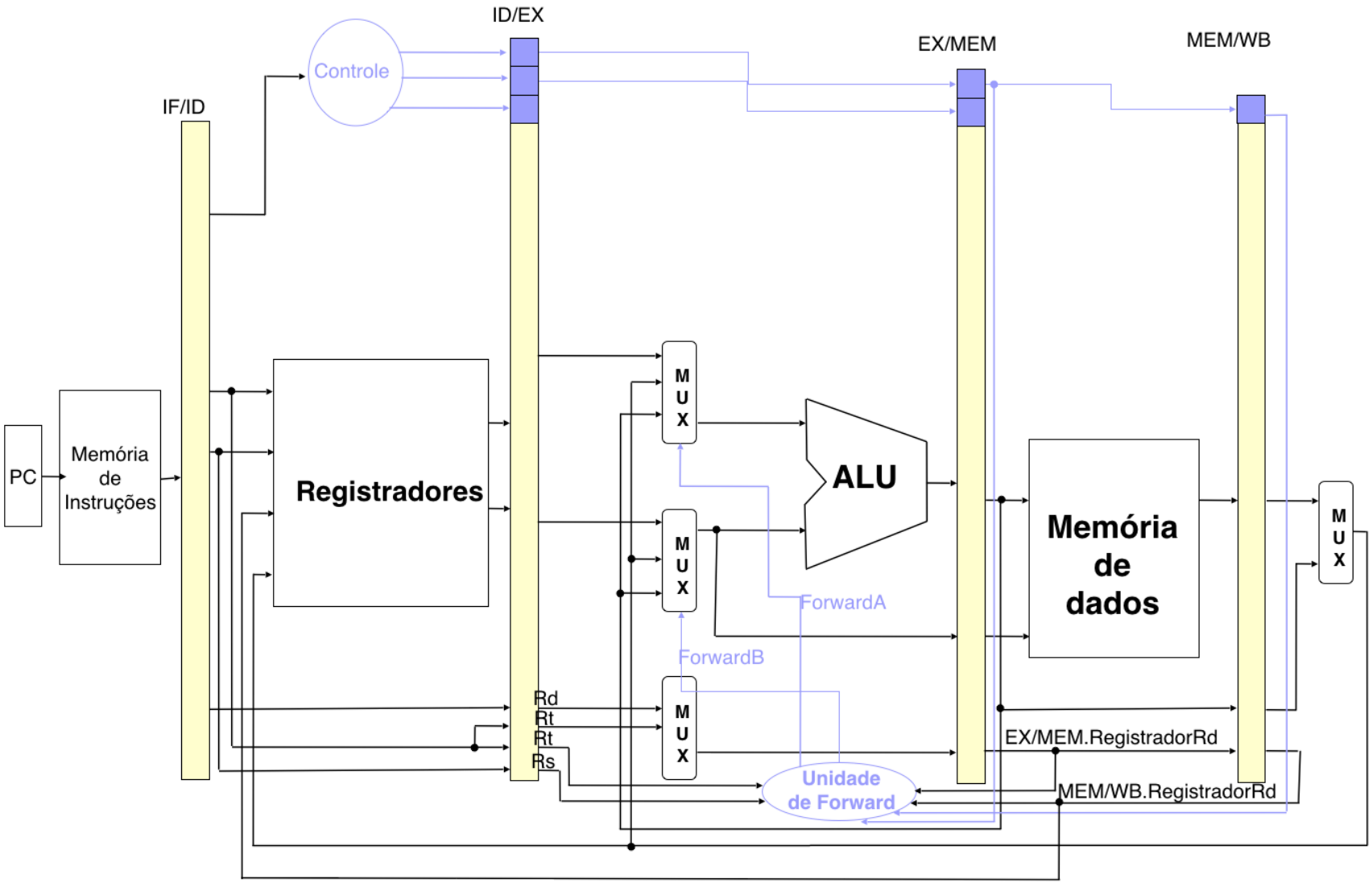
```
add $1, $1, $2
add $1, $1, $3
add $1, $1, $4
```

- Forwarding do estágio WB só feito se operando da origem da ALU diferente do registrador do estágio MEM

Hazard de dados e *forwarding*

- Neste caso resultado deve sofrer forwarding de MEM
- Mais recente que WB
- Controle de MEM:
  if ((MEM/WB.EscreveReg)
  and (MEM/WB.RegistradorRd != 0)
  and (EX/MEM.RegistradorRd != ID/EX.RegistradorRs)
  and (MEM/WB.RegistradorRd == ID/EX.RegistradorRs)) ForwardA = 01
  if ((MEM/WB.EscreveReg)
  and (MEM/WB.RegistradorRd != 0)
  and (EX/MEM.RegistradorRd != ID/EX.RegistradorRt)
  and (MEM/WB.RegistradorRd == ID/EX.RegistradorRt)) ForwardB = 01

Hazard de dados e *forwarding*



Hazard de dados e *Stalls*

- Leitura ainda pode causar um hazard:
  - Uma instrução tenta ler um registrador após uma instrução load que escreve no mesmo registrador
- Precisamos de uma unidade de detecção de hazard
  - Opera durante estágio ID, inserindo stall entre load e seu uso

Hazard de dados e *forwarding*

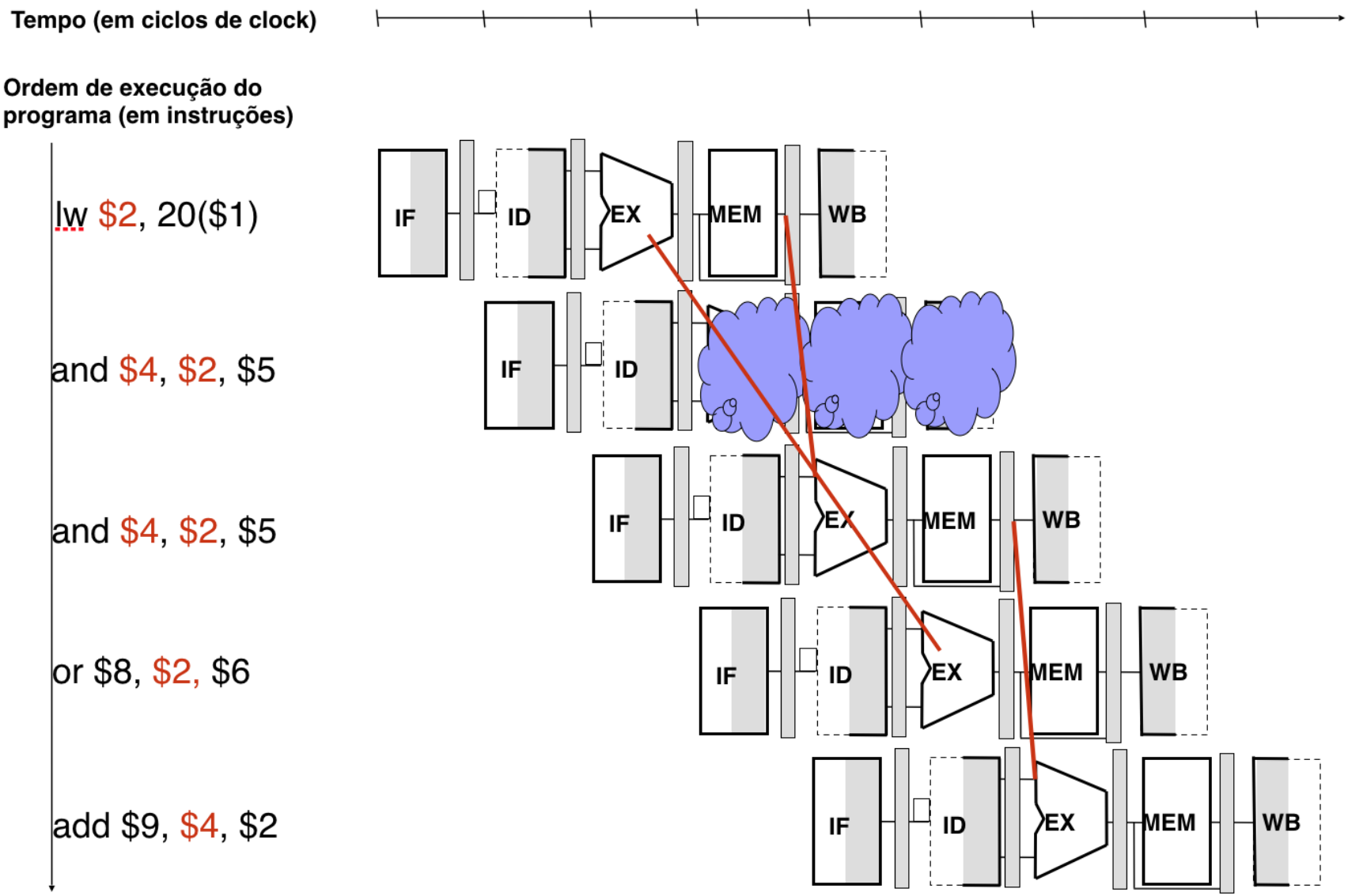
Hazard de dados e *Stalls*

- Como dependência entre load e instrução recua no tempo, hazard não pode ser resolvido por forwarding
- Detecção:
  if ((ID/EX.LeMem)
  and ((ID/EX.RegistradorRt == IF/ID.RegistradorRs) or
  (ID/EX.RegistradorRt == IF/ID.RegistradorRt))) ocasiona stall no pipeline

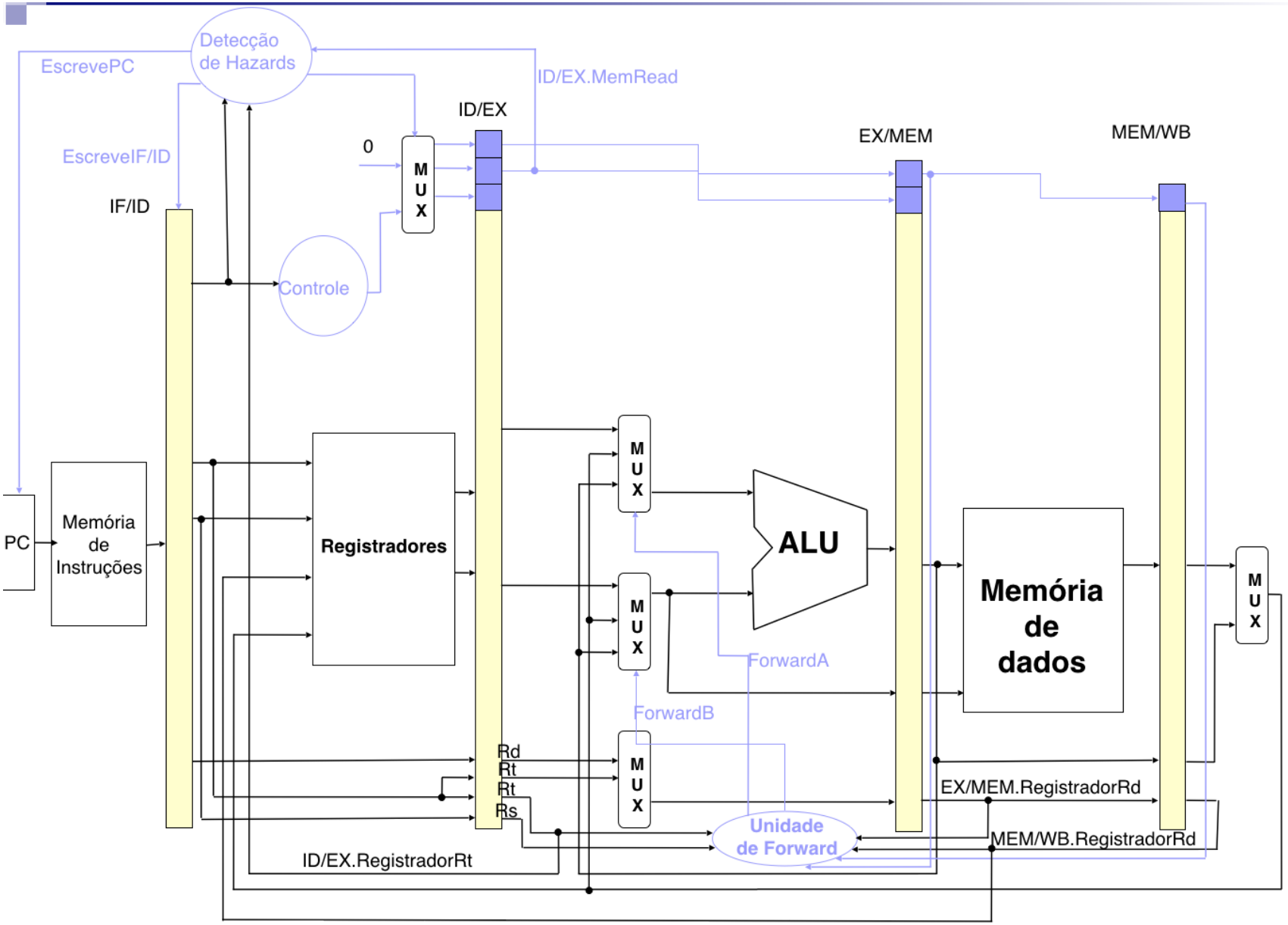
Hazard de dados e *Stalls*

- Se instrução sofre stall, precisamos evitar também que próxima instrução avance
- Não atualizar PC e IF/ID
  - Instrução no estágio IF continuará a ser lida usando o mesmo PC
  - Registradores no estágio ID continuarão a ser lidos usando os mesmos valores
- Inserir nops
  - Colocar 0 em todos os sinais de controle
  - Serão filtrados: nenhum valor modificado se todos os sinais iguais a zero

Hazard de dados e *forwarding*



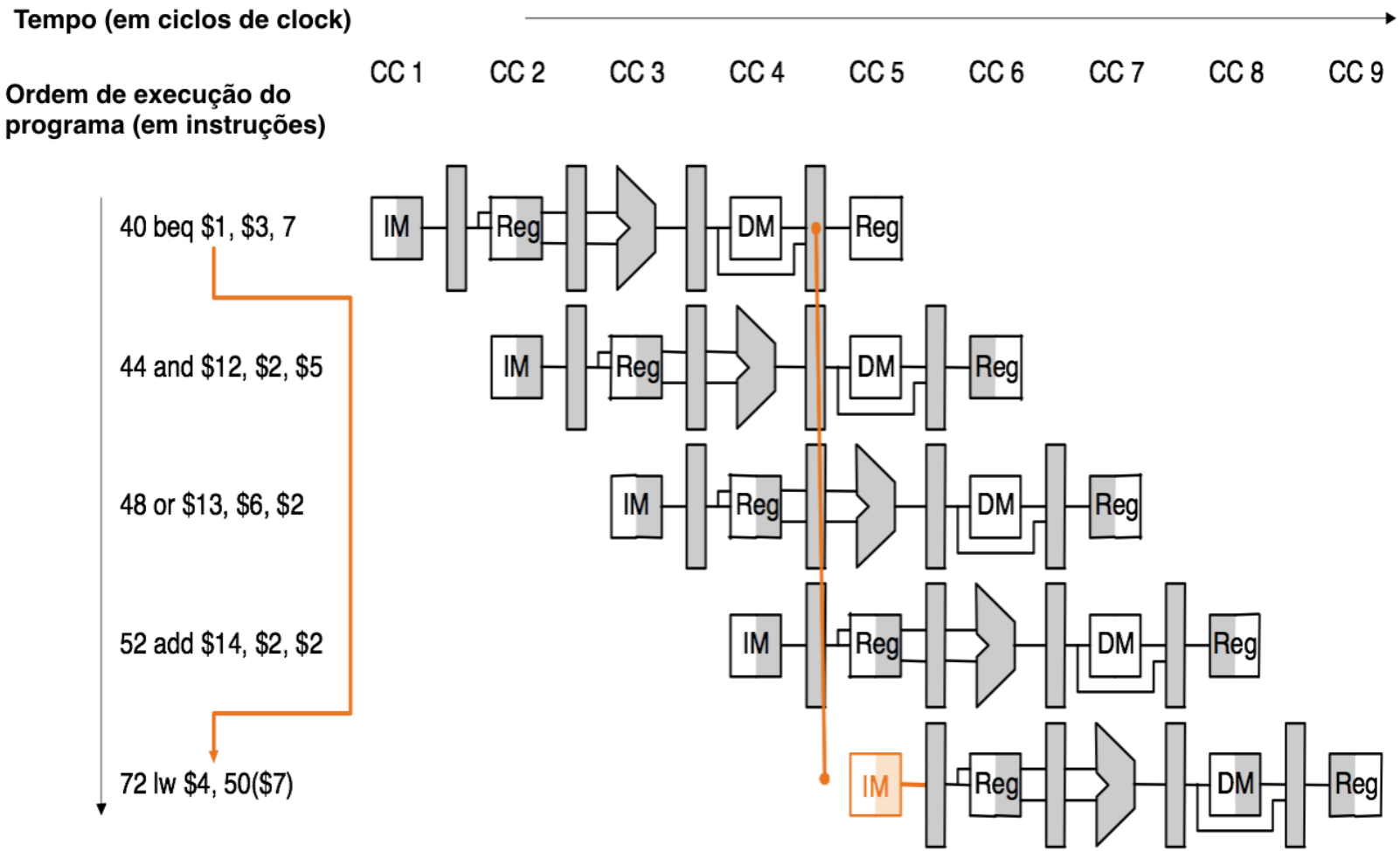
Hazard de dados e *forwarding*



Hazard de Desvio

- Uma instrução precisa ser buscada a cada ciclo para sustentar o pipeline
- A decisão do desvio não ocorre até o estágio MEM
- Conforme visto, este atraso sobre a decisão de que instrução buscar é chamado de hazard de desvio ou controle

Hazard de Desvio



Hazard de Desvio

- Considere que o desvio não foi tomado
- Podemos considerar que desvio não será tomado
- Caso seja tomado, precisamos descartar instruções buscadas e decodificadas
  - Execução continua no destino do desvio
- Se desvio tomado na metade das vezes e descartar instruções for barato
  - Otimização reduz pela metade custo do hazard de desvio
- Como descartar instruções?
  - Alteramos valor de controles para zero
  - Fazer flush nas instruções nos estágios IF, ID e EX do pipeline quando instrução de desvio atinge MEM

Hazard de Desvio

- **Reduzindo atraso nos desvios**
- PC do desvio selecionado no estágio MEM
- Podemos mover a execução do desvio para um estágio anterior do pipeline (p.ex., ID)
  - Muitos desvios contam com testes simples
  - Não exigem operação completa na ALU
  - Podem ser feitas com portas lógicas
  - Necessário:
    - Cálculo do endereço do desvio
    - Avaliar decisão do desvio

Hazard de Desvio

- **Cálculo do endereço do desvio**
- Temos PC
- Campo imediato no registrador IF/ID
- Logo basta mover somador do desvio de EX para ID
- Cálculo do endereço destino feito para todas as instruções $\rightarrow$ Mas só usado quando necessário

Hazard de Desvio

- **Avaliar decisão do desvio**
- beq: comparação de dois registradores, para verificar se são iguais
- Igualdade pode ser testada com OU exclusivo dos bits dos registradores, e depois OR de todos os resultados
- Necessário também hardware adicional de forwarding e detecção de hazard $\rightarrow$ Nova lógica
- Flushing das instruções no estágio IF feito com linha de controle que zera campo de instrução no registrador IF/ID
  - Apagar instrução a transforma em um nop

Hazard de Desvio

- **Previsão dinâmica de desvios**
- Penalidade dos desvios aumenta conforme profundidade do pipeline $\rightarrow$ Mais instruções perdidas
- Em pipelines agressivos, previsão estática desperdiça muito desempenho
- Tentaremos prever o comportamento do desvio durante a execução do programa

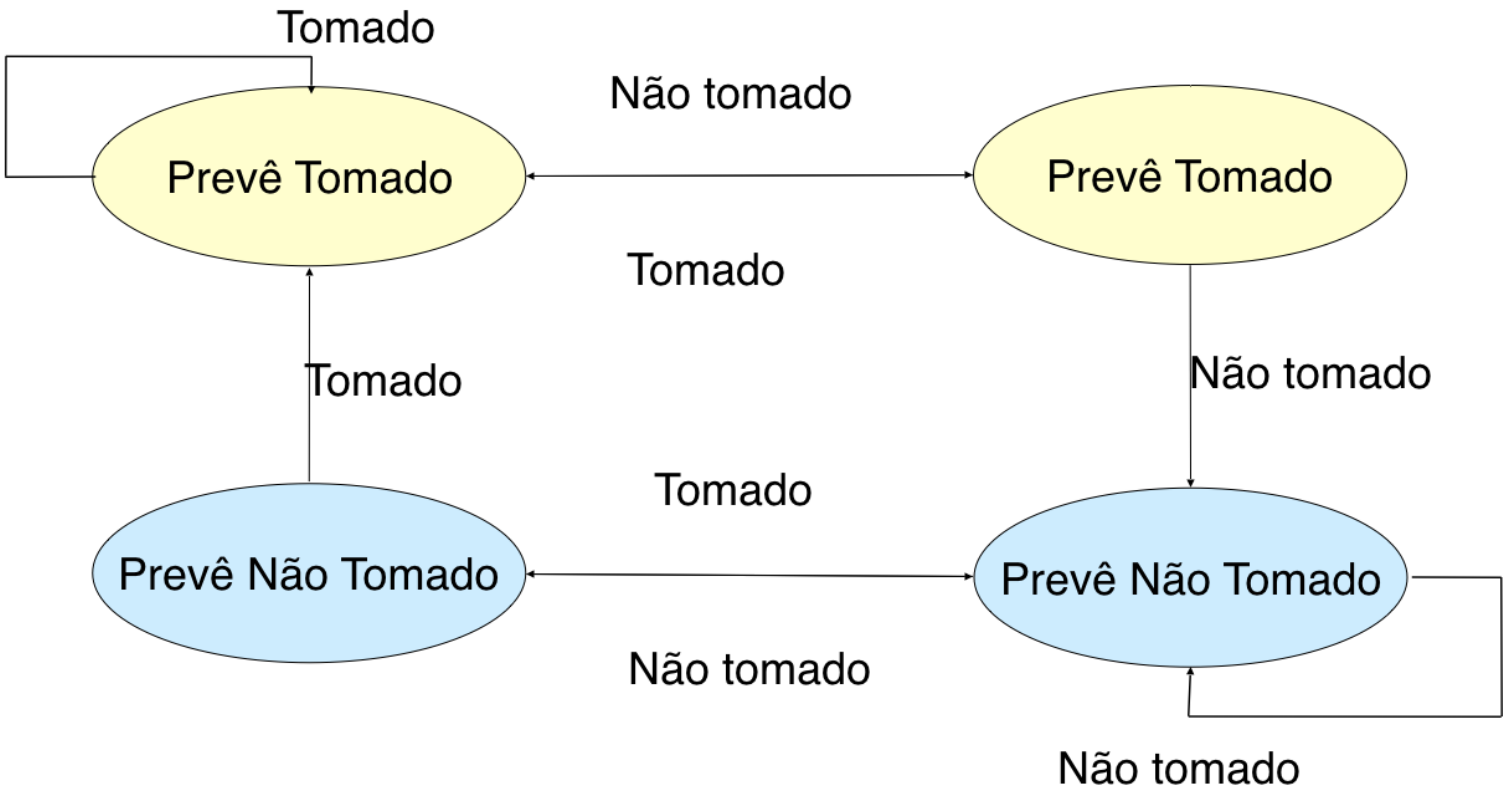
Hazard de Desvio

- Técnica: pesquisar o endereço da instrução para ver se desvio foi tomado da última vez
- Caso tenha sido, busca-se instruções a partir do mesmo lugar da última vez
- Tabela de histórico de desvios (ou buffer de previsão de desvios) usado na implementação
  - Pequena memória indexada pela parte menos significativa do endereço da instrução de desvio
  - Contém um bit dizendo se desvio foi tomado recentemente ou não

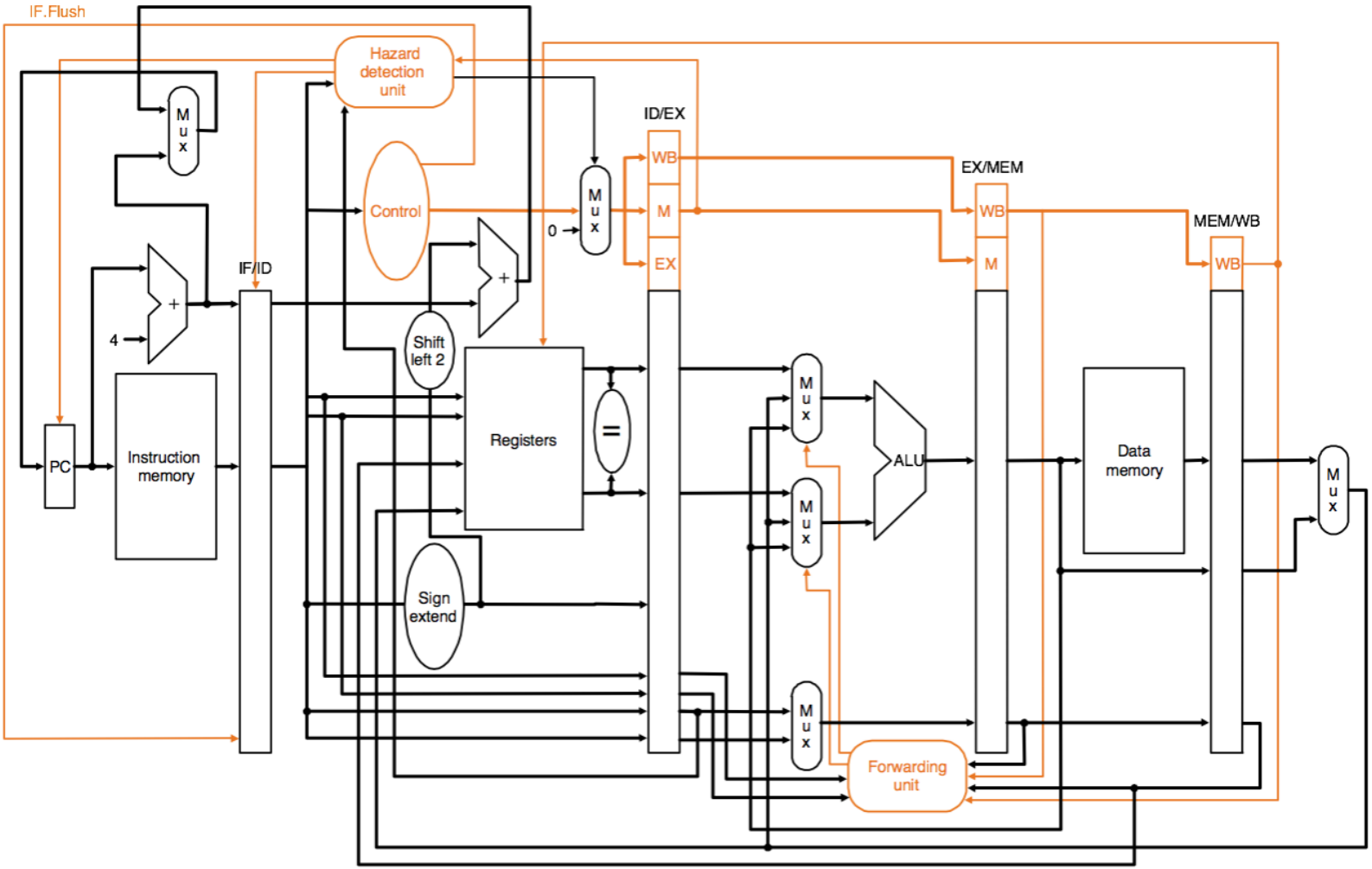
Hazard de Desvio

- Esquema de 1 bit tem problema de desempenho
  - Mesmo que desvio quase sempre tomado, provavelmente faremos uma previsão incorreta duas vezes
- Esquema com dois bits utilizado
  - Previsão precisa estar errada duas vezes para ser alterada

Hazard de Desvio



Hazard de Desvio



Próxima aula

Processador - Caminho de Dados e de Controle - Parte IV