Organização e Arquitetura de Computadores I

Pipeline

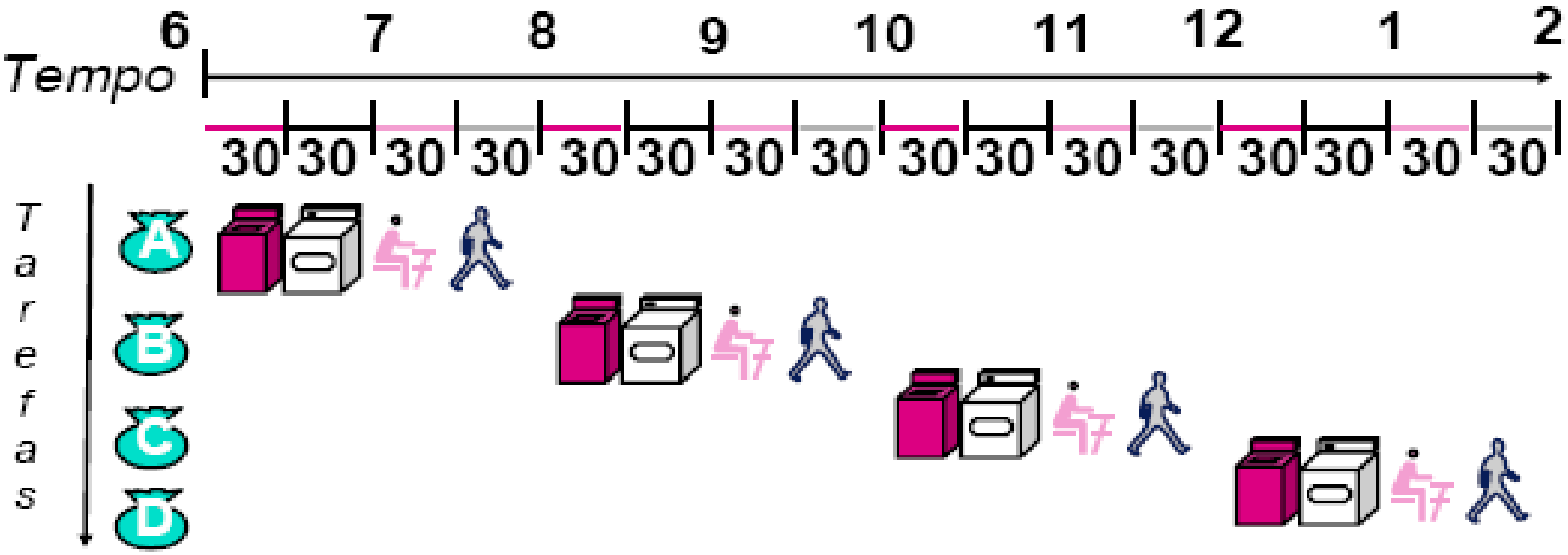
Sumário

- Pipeline
- Pipeline Hazards:
  - Hazards Estruturais
  - Hazards de Dados
  - Hazards de Controle
- Caminho de Dados usando Pipeline
- Representação Gráfica do Pipeline
- Forwarding

Pipeline

- *Pipelining* é uma técnica de implementação em que várias instruções são sobrepostas na execução.
- Para exemplificar o funcionamento do *pipelining* será utilizada uma analogia com o processo de lavar roupas.
- A técnica utilizada é a descrita abaixo:
  1 Colocar a trouxa com roupa suja na lavadora;
  2 Ao término da lavagem, colocar a roupa lavada na secadora;
  3 Ao término da secadora, passar a roupa;
  4 Após passar a roupa, guardá-la.

Pipeline

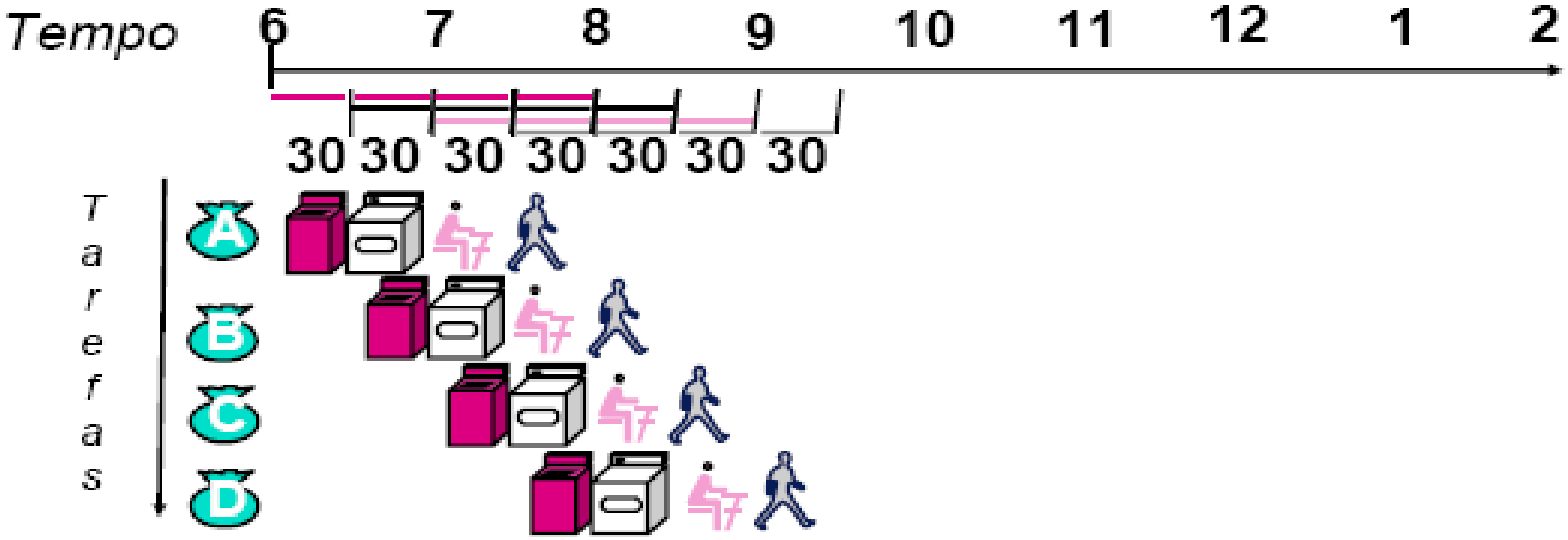


Processo de lavagem de roupas sem *pipeline*

Pipeline

- A mesma técnica com *pipeline* leva muito menos tempo.
- Ao término da lavagem da primeira trouxa e ela for levada para a secadora, a segunda trouxa será colocada na máquina de lavar roupas e continua para cada etapa... Até que passamos a ter tarefas sendo feitas simultaneamente.
- Na analogia, cada tarefa corresponderia a um estágio do *pipeline*.
- Desde que haja recursos separados para cada estágio, podemos usar um *pipeline* para as tarefas.

Pipeline



Processo de lavagem de roupas com *pipeline*

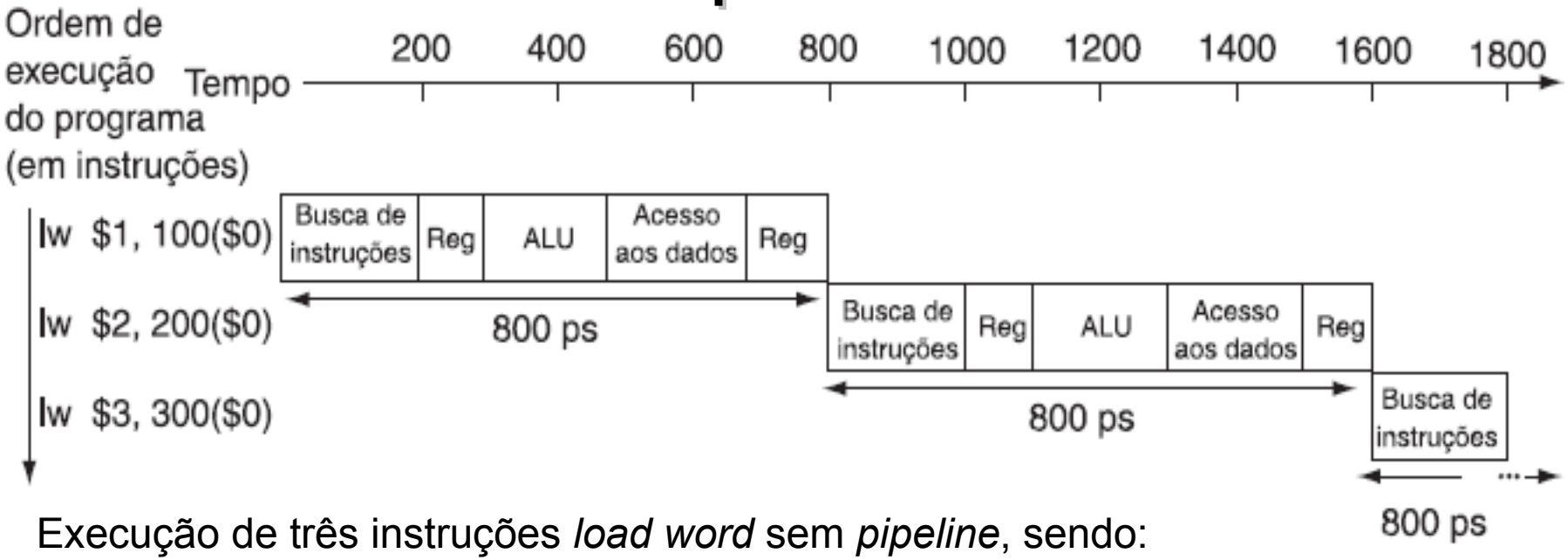
Pipeline

- A técnica melhora a vazão do sistema sem melhorar o tempo de conclusão de uma única instrução. Sendo assim, a melhora no desempenho ocasionada pela utilização da técnica de *pipelining*, só é notada quando temos muitas instruções a serem executadas.

Pipeline

- As instruções MIPS normalmente exigem cinco etapas:
  1. Buscar instrução da memória;
  2. Ler registradores enquanto a instrução é decodificada.
  3. Executar a operação ou calcular um endereço;
  4. Acessar um operando na memória de dados;
  5. Escrever o resultado em um registrador.
- O *pipeline* estudado terá portanto 5 estágios.

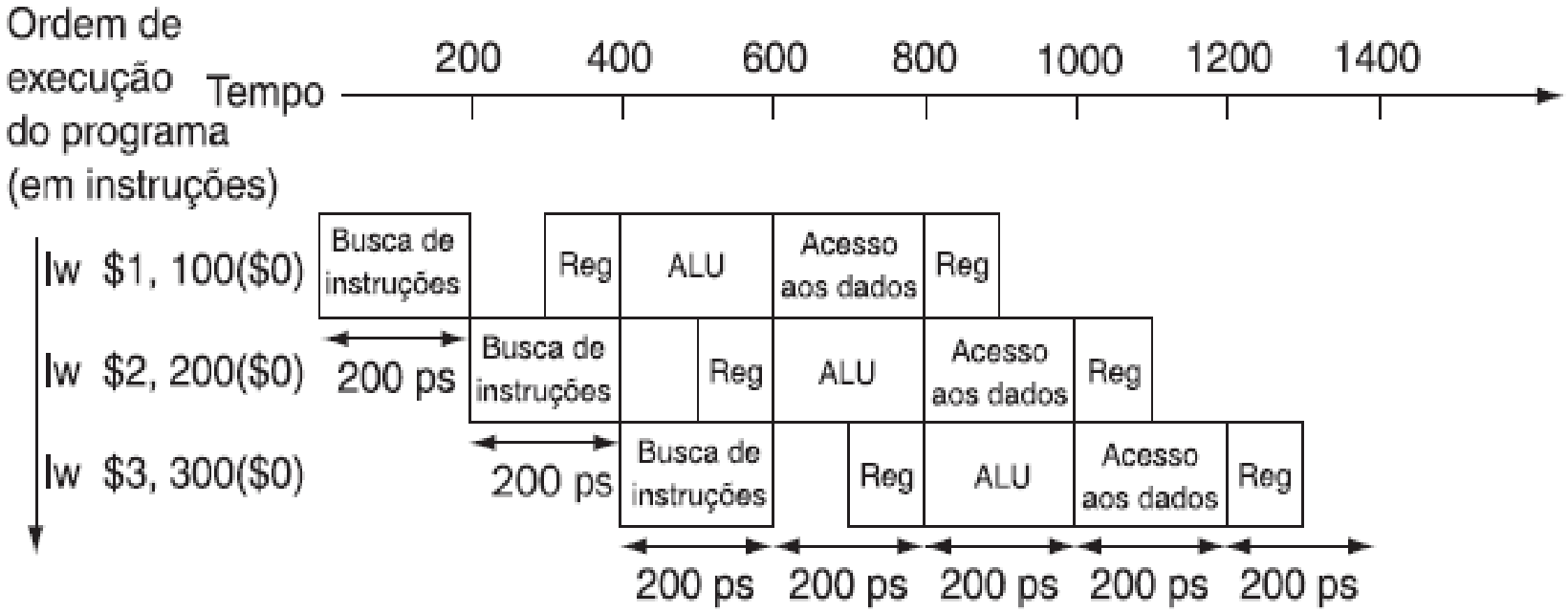
Pipeline



Execução de três instruções *load word* sem *pipeline*, sendo:

- Busca da Instrução = 200ps
- Leitura de registradores = 100ps
- Operação da ULA = 200ps
- Acesso a dados = 200ps

Pipeline



Execução de três instruções *load word* com *pipeline*

Pipeline

- Os tempos de estágio do pipeline dos computadores são limitados pelo recurso mais lento, seja a operação da ULA ou o acesso à memória.
- A técnica de *pipeline* acrescenta um certa demora na execução da instrução, ou seja, uma única instrução sendo executada em um sistema sem *pipelining* termina mais rápido do que uma executada em um sistema com *pipeling*.

Pipeline

- Facilidades à implantação do Pipeline na arquitetura MIPS:
  - Todas as instruções possuem o mesmo tamanho;
  - Poucos formatos de instrução;
  - Operandos de memória aparecem apenas em *loads* e *stores*.
- Dificuldades:
  - Riscos estruturais: apenas uma memória, por exemplo;
  - Riscos de Controle: Necessidade de se preocupar com instruções ramificadas;
  - Riscos de dados: uma instrução depende de uma instrução anterior.

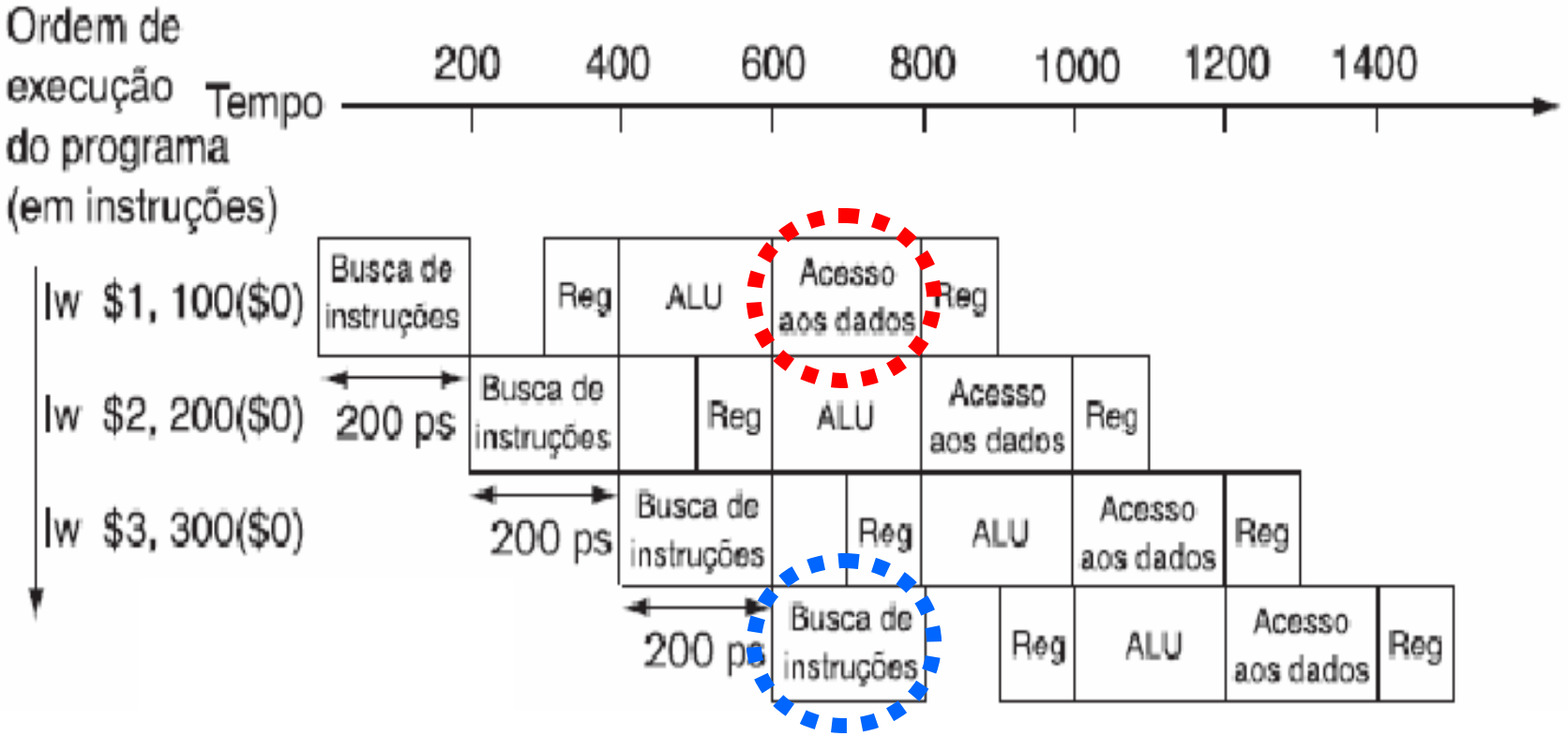
Pipeline Hazards

- *Hazard* = risco.
- Existem situações em *pipelining* em que a próxima instrução não pode ser executada no ciclo de *clock* seguinte, são elas:
  - *Hazards* estruturais;
  - *Hazards* de dados;
  - *Hazards* de controle.

Hazards Estruturais

- Significa que o *hardware* não pode admitir a combinação de instruções que queremos executar no mesmo ciclo de *clock*.
  - Suponha que tivéssemos uma única memória, em vez de duas. Se o *pipeline* tivesse uma quarta instrução, veríamos que, no mesmo ciclo de *clock* em que a primeira instrução está acessando dados da memória, a quarta instrução está buscando uma instrução dessa mesma memória. Sem duas memórias, nosso *pipeline* poderia ter um *hazard* estrutural.

Hazards Estruturais



Hazard estrutural em um *pipeline*

Hazards de Dados

- O *hazards* de dados ocorrem quando o *pipeline* precisa ser interrompido porque uma etapa precisa esperar até que outra seja concluída.
- Um exemplo desse problema é quando uma instrução depende de uma anterior que ainda está no *pipeline*.
  - Supondo que exista uma instrução add seguida imediatamente por uma instrução sub, que usa o resultado da soma na sua subtração:

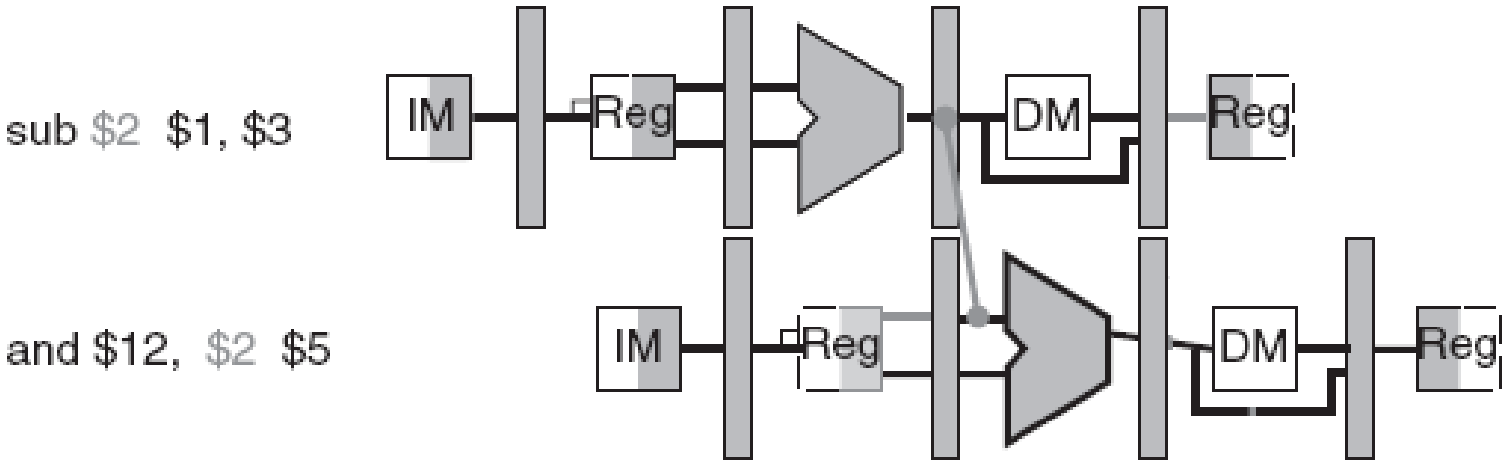
```
add     $s0, $t0, $t1
sub     $t2, $s0, $t3
```

Hazards de Dados

- Sem intervenção, um *hazard* de dados poderia prejudicar o *pipeline* severamente.
- Uma possível solução seria atrasarmos a segunda instrução para que a mesma começasse somente após o término da anterior.
- Mas não precisamos necessariamente esperar pelo fim da instrução anterior para tentar resolver o *hazard*. No caso do exemplo, Assim que a ULA gera a saída da soma, podemos fornecê-la como entrada para a subtração.

Hazards de Dados

- O acréscimo de *hardware* extra para ter o item que falta antes do previsto, diretamente dos recursos internos, é chamado de ***forwarding*** ou ***bypassing***.



Hazards de Controle

- O *hazard* de controle, também chamado de *hazard* de desvio, ocorre quando existe a necessidade de tomar uma decisão com base nos resultados de uma instrução enquanto outras estão sendo executadas.
- Ao iniciar a execução de uma instrução de desvio, o *pipeline* tem que buscar a próxima instrução. Contudo, o *pipeline* possivelmente não saberá qual deve ser a próxima instrução, pois a instrução de desvio ainda não foi concluída.

Hazards de Controle

- Ao encontrar um desvio o computador tenta prever qual será a possível decisão:
  - O desvio será tomado;
  - O desvio não será tomado.
- Uma técnica simples é **sempre prever que os desvios não serão tomados**, se coincidir o *pipeline* seguirá a toda velocidade, caso contrário o trabalho é perdido e o *pipeline* tem de recomeçar.

Hazards de Controle

- Uma versão mais sofisticada é a **previsão de desvio**, ela teria alguns desvios previstos como tomados e alguns como não tomados. Baseia-se no comportamento dos desvios ocorridos e as escolhas anteriores.
- A quantidade e o tipo de histórico mantido têm se tornado extensos, resultando em previsores de desvio dinâmicos que podem prever os desvios corretamente, com uma previsão superior a 90%.
- Quando a escolha estiver errada, o controle do *pipeline* terá de garantir que as instruções após o desvio errado não tenham efeito.

Hazards de Controle

- Existe uma outra técnica que é chamada de **decisão adiada**. Também chamada de *delayed branch* (desvio adiado) nos computadores, essa é a solução utilizada pela arquitetura MIPS.
- A decisão adiada, sempre executa a próxima instrução sequencial, com o desvio ocorrendo após esse atraso de uma instrução. O *software* MIPS colocará uma instrução imediatamente após a instrução de decisão adiada, que não é afetada pelo desvio.

Hazards de Controle

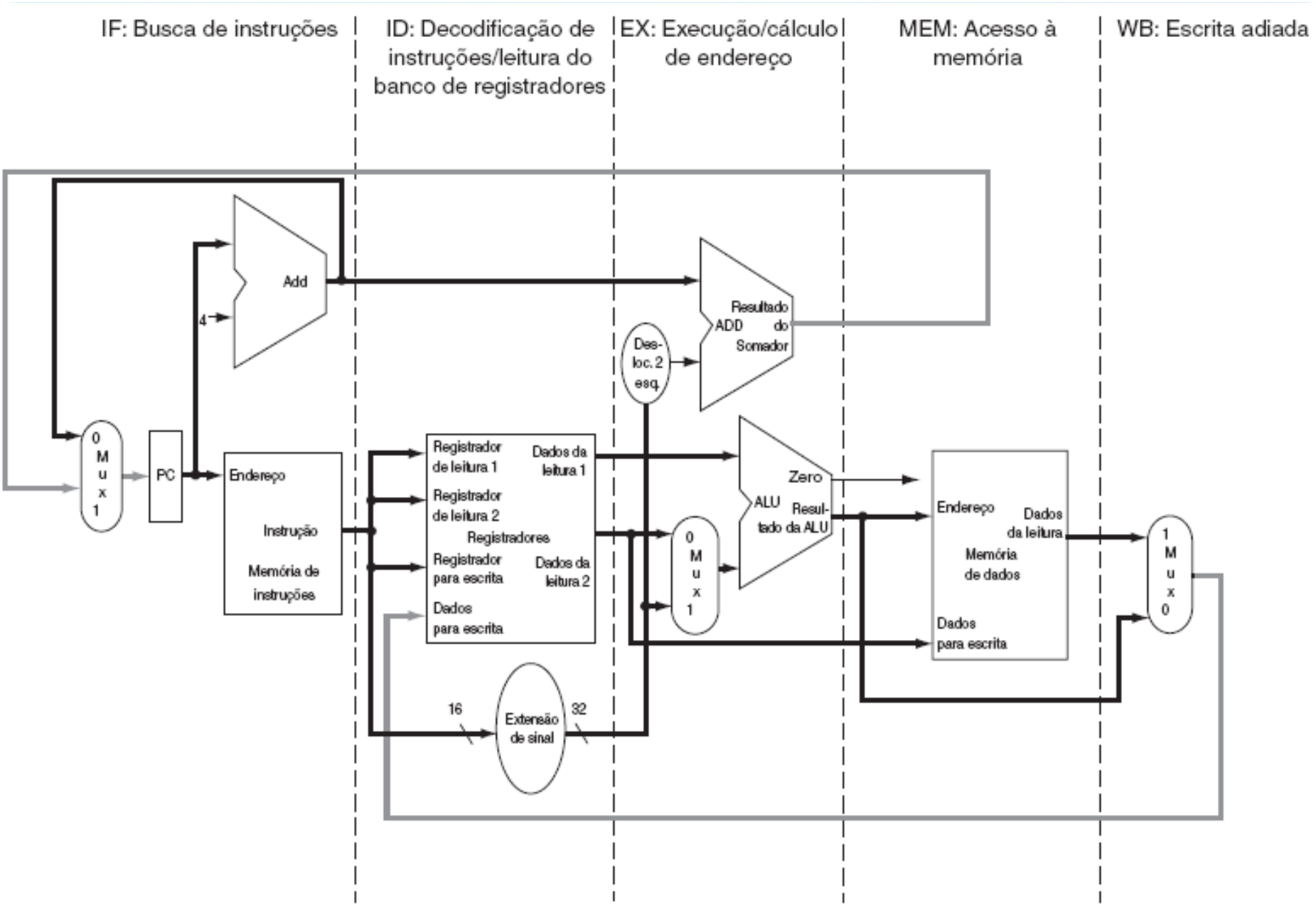
- Exemplo:

```
add $s4, $s5, $s6
beq $s1, $s2, L1
lw $s3, 3000($s3)
```

  - Após o “beq”, a arquitetura MIPS insere uma instrução que não é afetada pelo desvio (add $s4, $s5, $s6).

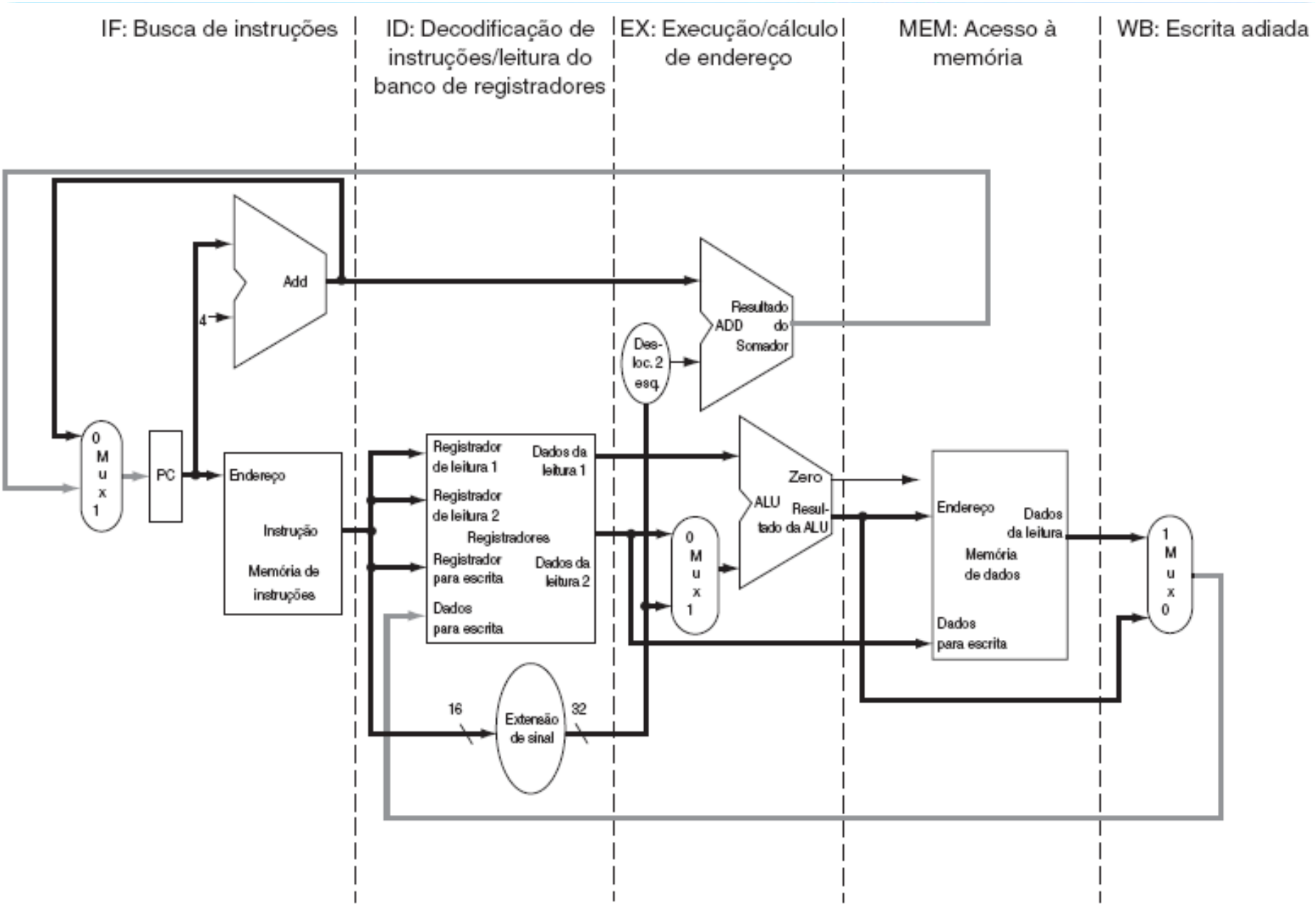
Caminho de Dados usando *Pipeline*

- A divisão de uma instrução em cinco estágios significa um *pipeline* de cinco estágios, que, por sua vez, significa que até cinco instruções estarão em execução durante qualquer ciclo de *clock*.
- São elas:
  - IF (*Instruction Fetch*): busca de instruções;
  - ID (*Instruction Decode*): Decodificação de instruções e leitura do banco de registradores;
  - EX (*Execution*): Execução ou cálculo de endereço;
  - MEM (*Memory*): Acesso à memória de dados;
  - WB (*Write Back*): Escrita do resultado.


IF: Busca de instruções
ID: Decodificação de instruções/leitura do banco de registradores
EX: Execução/cálculo de endereço
MEM: Acesso à memória
WB: Escrita adiada
Add
4
0
M
u
x
1
PC
Endereço
Instrução
Memória de instruções
Registrador de leitura 1
Registrador de leitura 2
Registradores
Registrador para escrita
Dados para escrita
Dados da leitura 1
Dados da leitura 2
16
Extensão de sinal
32
Des-loc. 2 esq.
Resultado ADD do Somador
0
M
u
x
1
ALU
Zero
Resul-tado da ALU
Endereço
Dados da leitura
Memória de dados
Dados para escrita
1
M
u
x
0

Caminho de Dados usando *Pipeline*

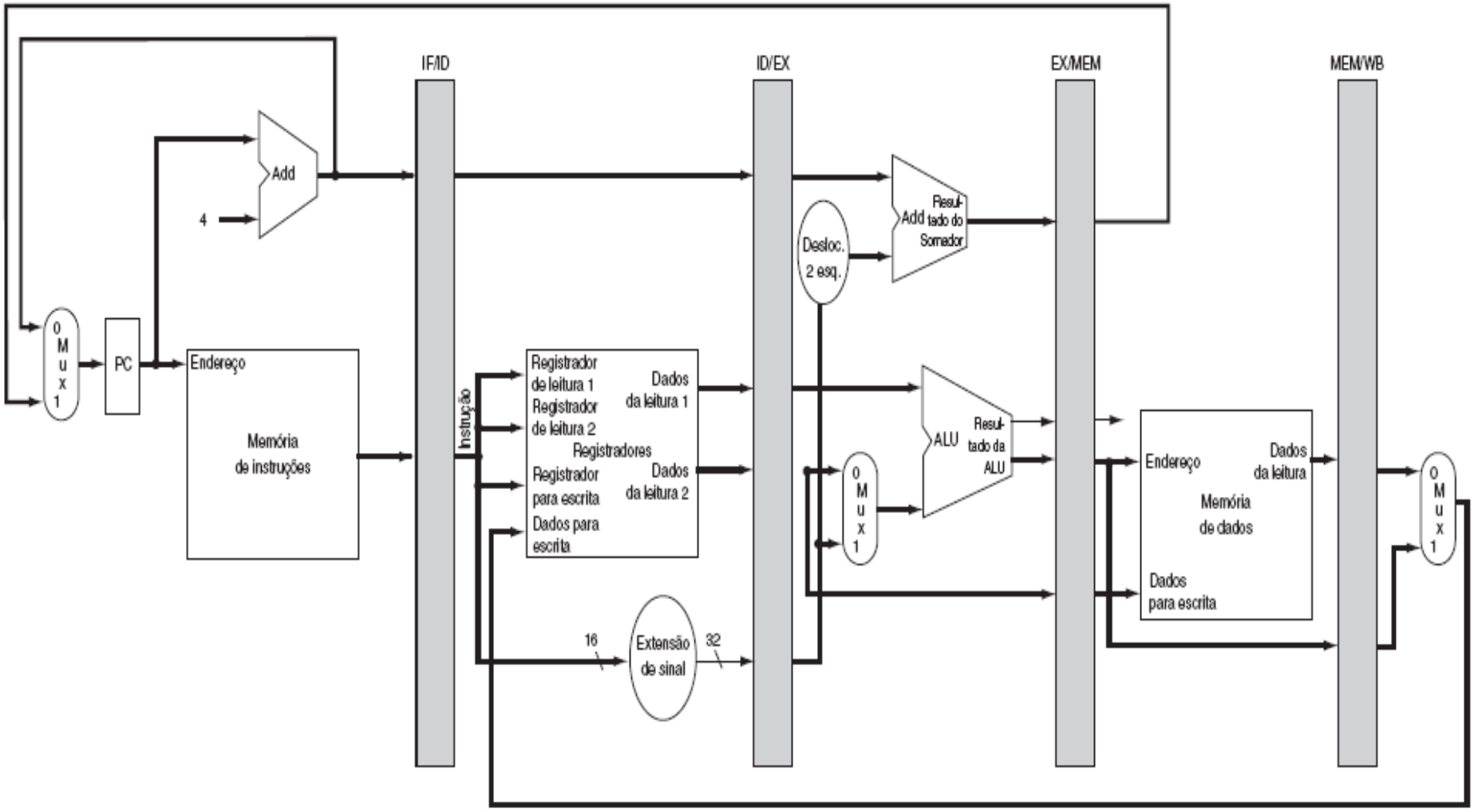
- O fluxo de informações se dá da esquerda para direita, pelos cinco estágios.
- Existem duas exceções para esse fluxo de informações da esquerda para a direita:
  - O estágio de escrita do resultado, que coloca o resultado de volta no banco de registradores, no meio do caminho de dados;
  - A seleção do próximo valor no PC, escolhendo entre o PC incrementado e o endereço de desvio do estágio MEM.
- Os dados fluindo da direita para a esquerda não afetam a instrução atual; somente as instruções seguintes no *pipeline* são influenciadas por esses movimentos de dados reversos.


IF: Busca de instruções
ID: Decodificação de instruções/leitura do banco de registradores
EX: Execução/cálculo de endereço
MEM: Acesso à memória
WB: Escrita adiada
Add
4
0
M
u
x
1
PC
Endereço
Instrução
Memória de instruções
Registrador de leitura 1
Registrador de leitura 2
Registradores
Registrador para escrita
Dados para escrita
Dados da leitura 1
Dados da leitura 2
16
Extensão de sinal
32
Des-loc. 2 esq.
Resultado ADD do Somador
0
M
u
x
1
ALU
Zero
Resul-tado da ALU
Endereço
Dados da leitura
Memória de dados
Dados para escrita
1
M
u
x
0

Caminho de Dados usando *Pipeline*

- Existe a necessidade de colocarmos registradores sempre que existam linhas divisórias entre os estágios para que as informações de um estágio não sejam perdidas ao passarmos para um outro.
- Todas as instruções avançam durante cada ciclo de *clock* de um registrador do *pipeline* para o seguinte. Os registradores recebem os nomes dos dois estágios separados por esse registrador.
- No estágio final não existe registrador de *pipeline*. Todas as instruções precisam atualizar algum estado no processador, assim um registrador de *pipeline* separado é redundante para o estado atualizado.

Organização e Arquitetura de Computadores I



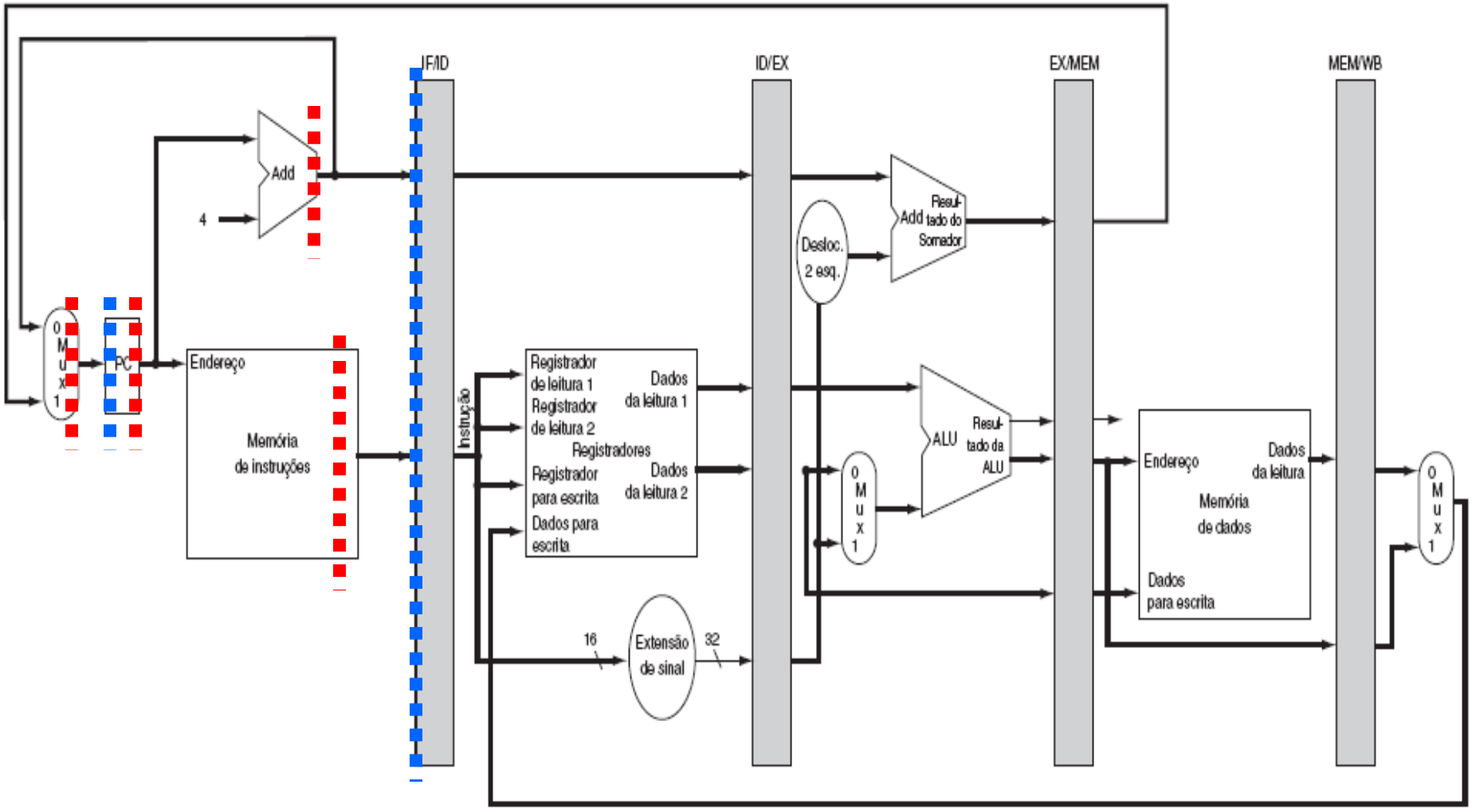
Caminho de Dados usando *Pipeline*

- Cada instrução atualiza o PC, seja incrementando-o ou atribuindo a ele o endereço de destino de um desvio. O PC pode ser considerado um registrador de *pipeline*. Mas diferente dos registradores de *pipeline*, ele é visível e no caso de uma exceção seu conteúdo precisa ser salvo.
- Os registradores de *pipeline* precisam ser grandes o suficiente para armazenar todos os dados correspondentes às linhas que passam por eles:
  - IF/ID – 64 bits;
  - ID/EX – 128 bits;
  - EX/MEM – 97 bits;
  - MEM/WB – 64 bits.

Caminho de Dados usando *Pipeline*

- **Análise do caminho tomado pela instrução *load* nos estágios do *pipeline*:**
  - **Busca de instruções**: a instrução é lida da memória usando o endereço do PC e depois colocado no registrador de *pipeline* IF/ID. O endereço do PC é incrementado de 4 e depois escrito de volta ao PC. Esse endereço incrementado também é salvo no registrador IF/ID, caso seja necessário mais tarde para uma instrução (“beq”). O controle não sabe qual é o tipo de instrução ainda.

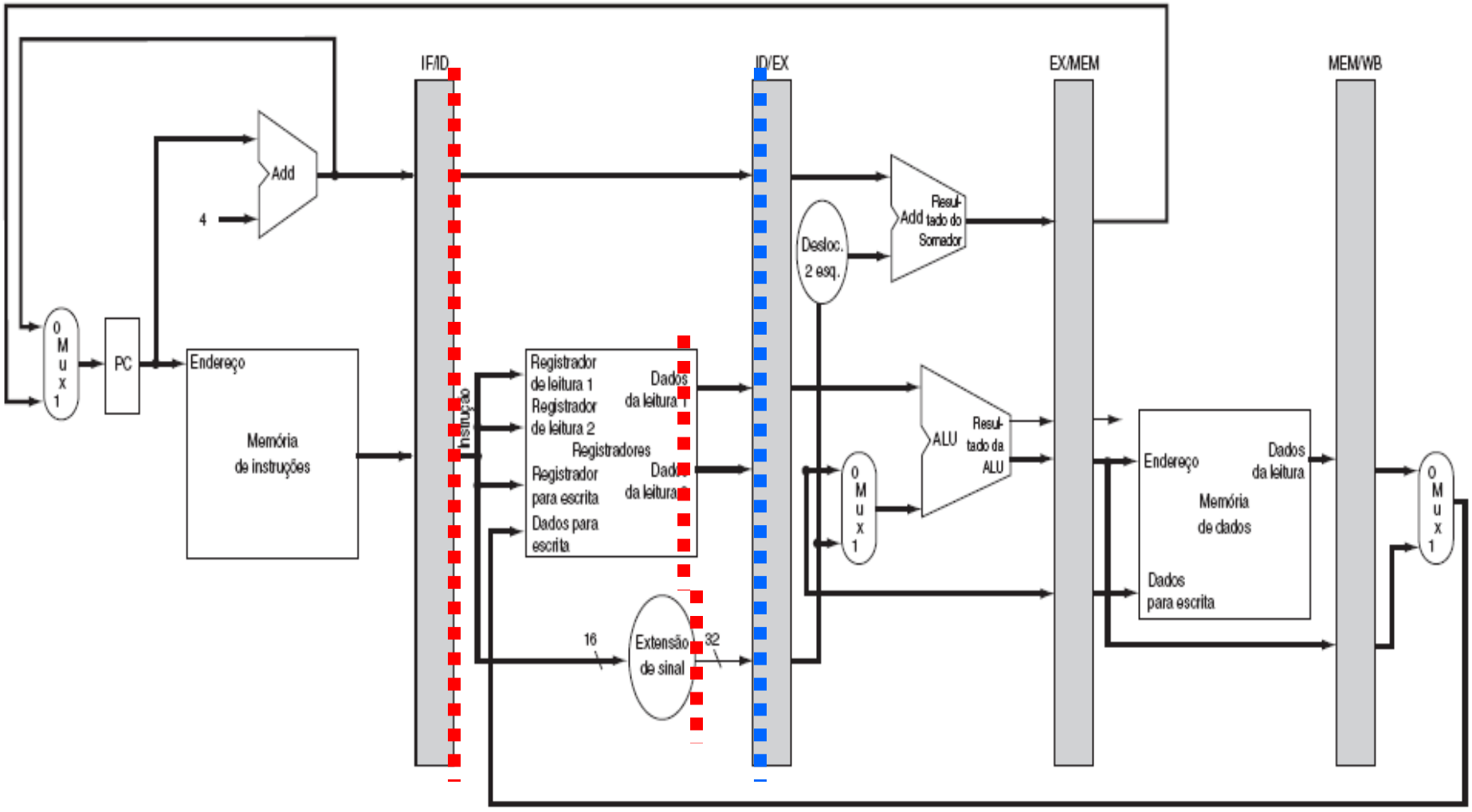
Organização e Arquitetura de Computadores I



Caminho de Dados usando *Pipeline*

- **Análise do caminho tomado pela instrução *load* nos estágios do *pipeline*:**
  - **Decodificação de instruções e leitura do banco de registradores**: o campo imm (*immediate*) de 16 bits, tem seu sinal estendido para 32 bits. São passados os endereços dos dois registradores para leitura. Todos os três valores são armazenados no registrador ID/EX.

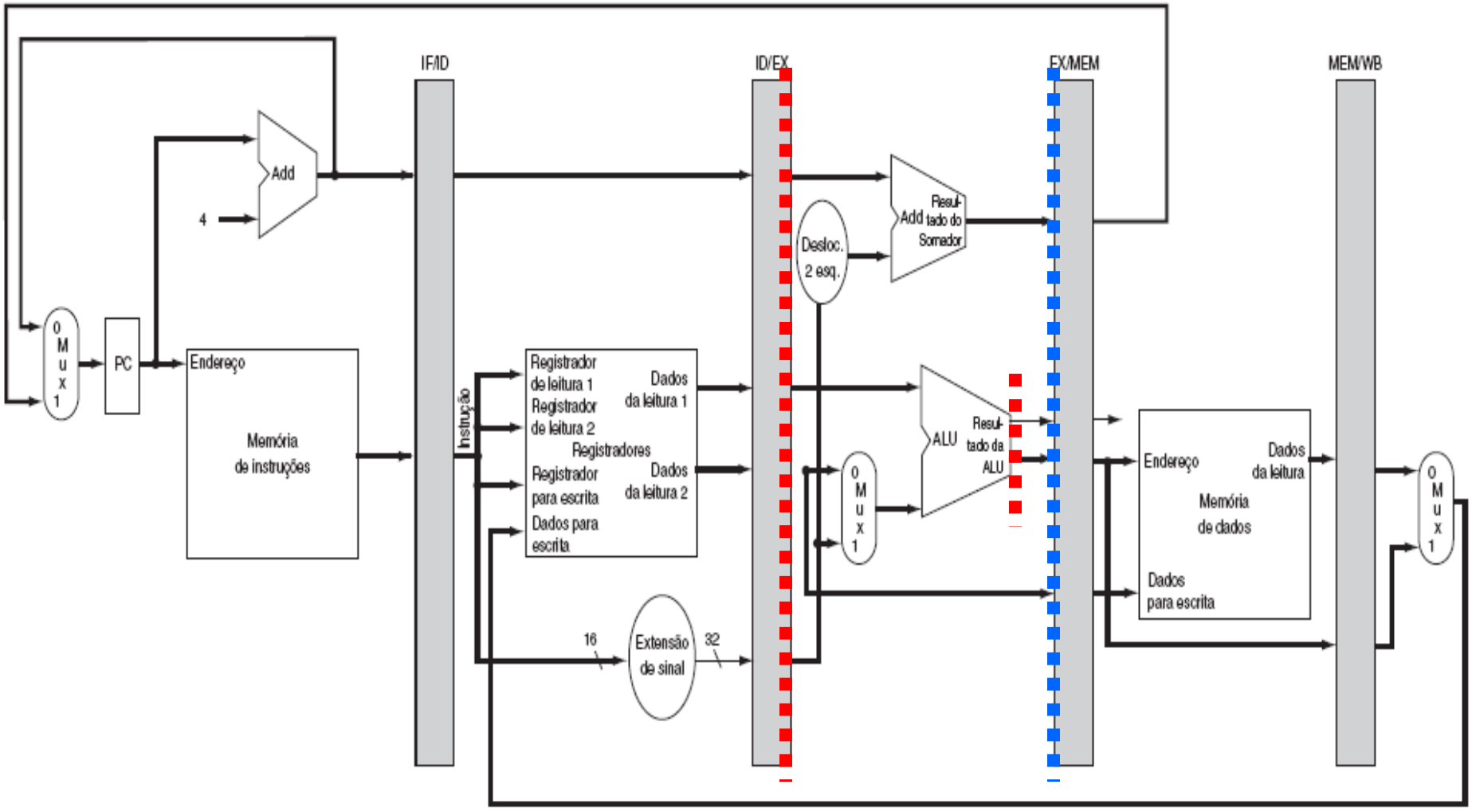
Organização e Arquitetura de Computadores I



Caminho de Dados usando *Pipeline*

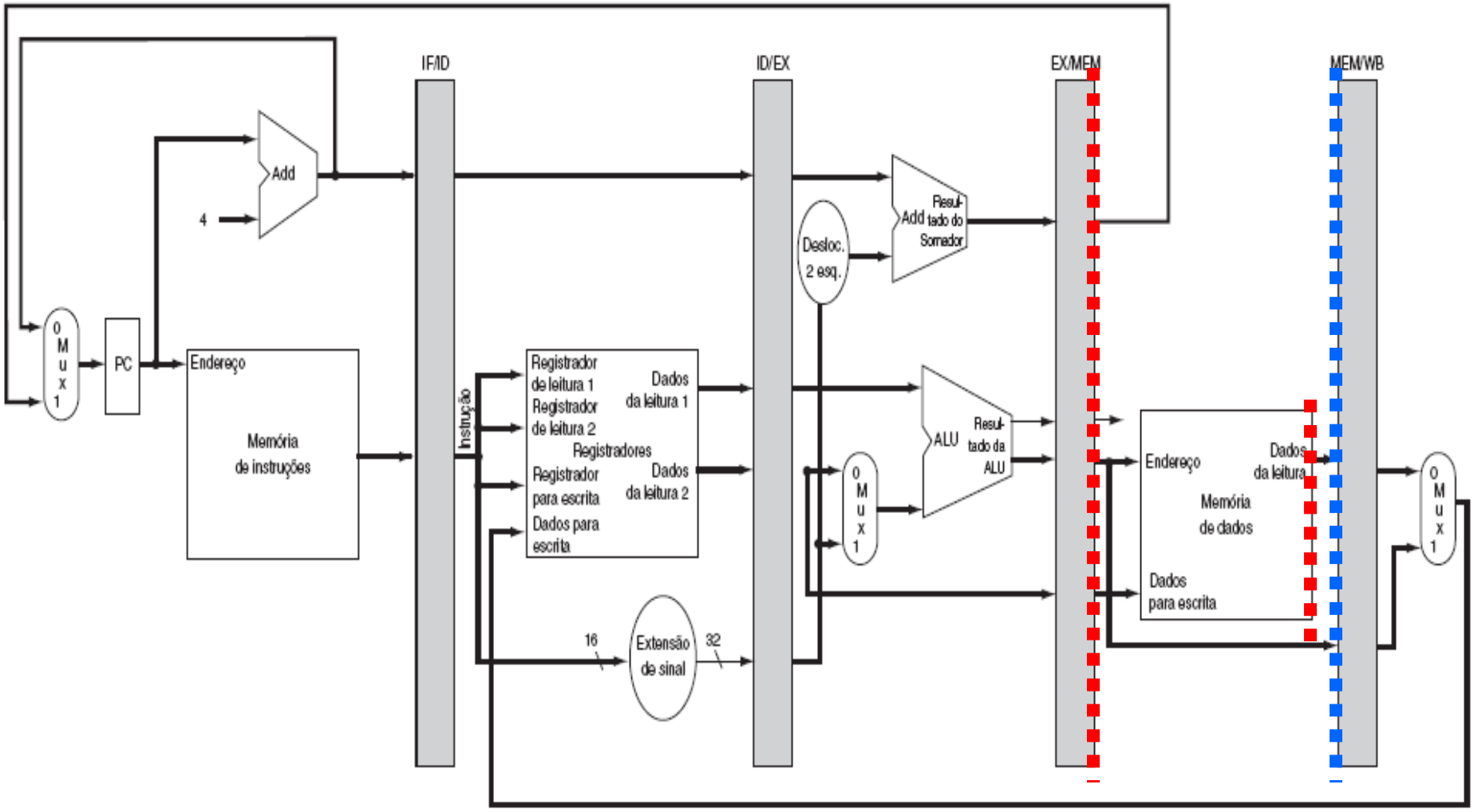
- **Análise do caminho tomado pela instrução *load* nos estágios do *pipeline*:**
  - **Execução ao cálculo de endereço**: a instrução lê o conteúdo do registrador de leitura 1 e o deslocamento com sinal estendido do registrador ID/EX e os soma usando a ULA, o resultado vai para EX/MEM.

Organização e Arquitetura de Computadores I



Caminho de Dados usando *Pipeline*

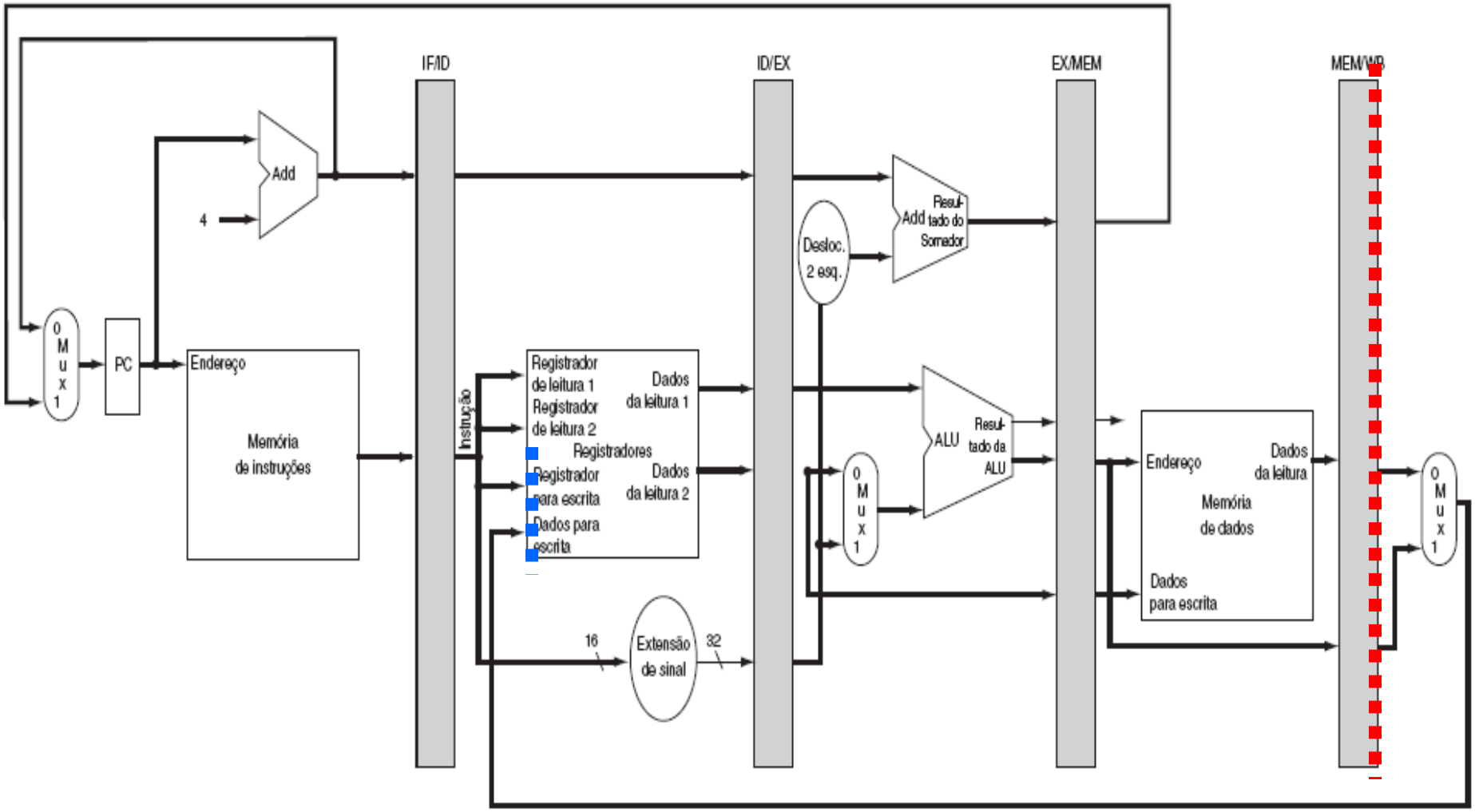
- **Análise do caminho tomado pela instrução *load* nos estágios do *pipeline*:**
  - **Acesso à memória**: O endereço de memória é lido de EX/MEM e usado para ler a memória de dados e os mesmos são armazenados no registrador MEM/WB.


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Endereço
Memória de instruções
Instrução
Registrador de leitura 1
Registrador de leitura 2
Registradores
Registrador para escrita
Dados para escrita
Dados da leitura 1
Dados da leitura 2
16
Extensão de sinal
32
Desloc. 2 esq.
Add
Resul-tado do Somador
ALU
Resul-tado da ALU
Endereço
Dados da leitura
Memória de dados
Dados para escrita

Caminho de Dados usando *Pipeline*

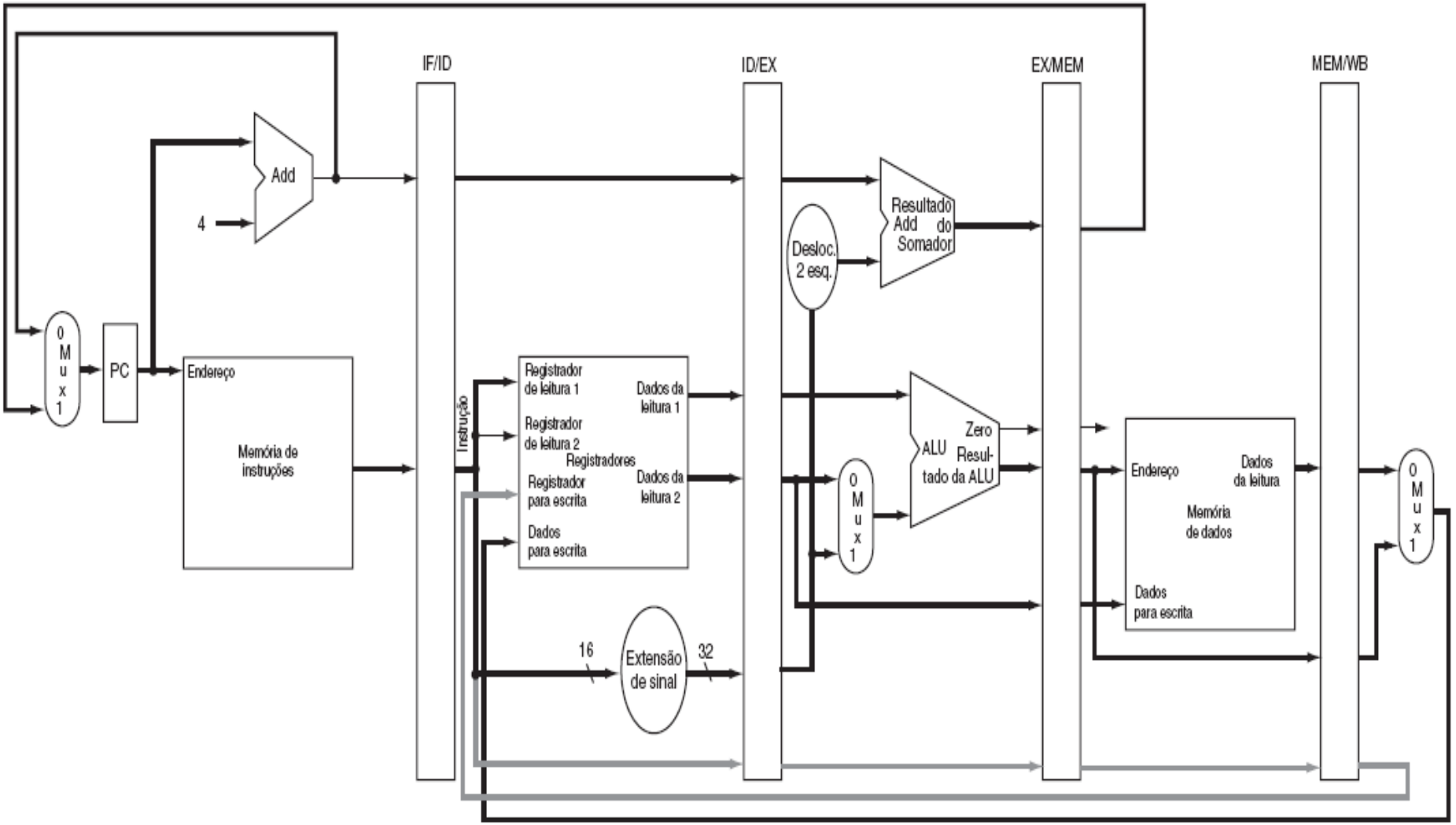
- **Análise do caminho tomado pela instrução *load* nos estágios do *pipeline*:**
  - **Escrita do resultado**: os dados são lidos do registrador MEM/WB e escritos no banco de registradores.

Organização e Arquitetura de Computadores I



Caminho de Dados usando *Pipeline*

- Existe um erro no projeto da instrução *load*. Não é determinado qual registrador fornece o endereço do registrador de destino da referida instrução. O certo seria esse endereço ser preservado para uso posterior. Ou melhor, preservar parte da instrução que seja necessária nos estágios posteriores do *pipeline*.

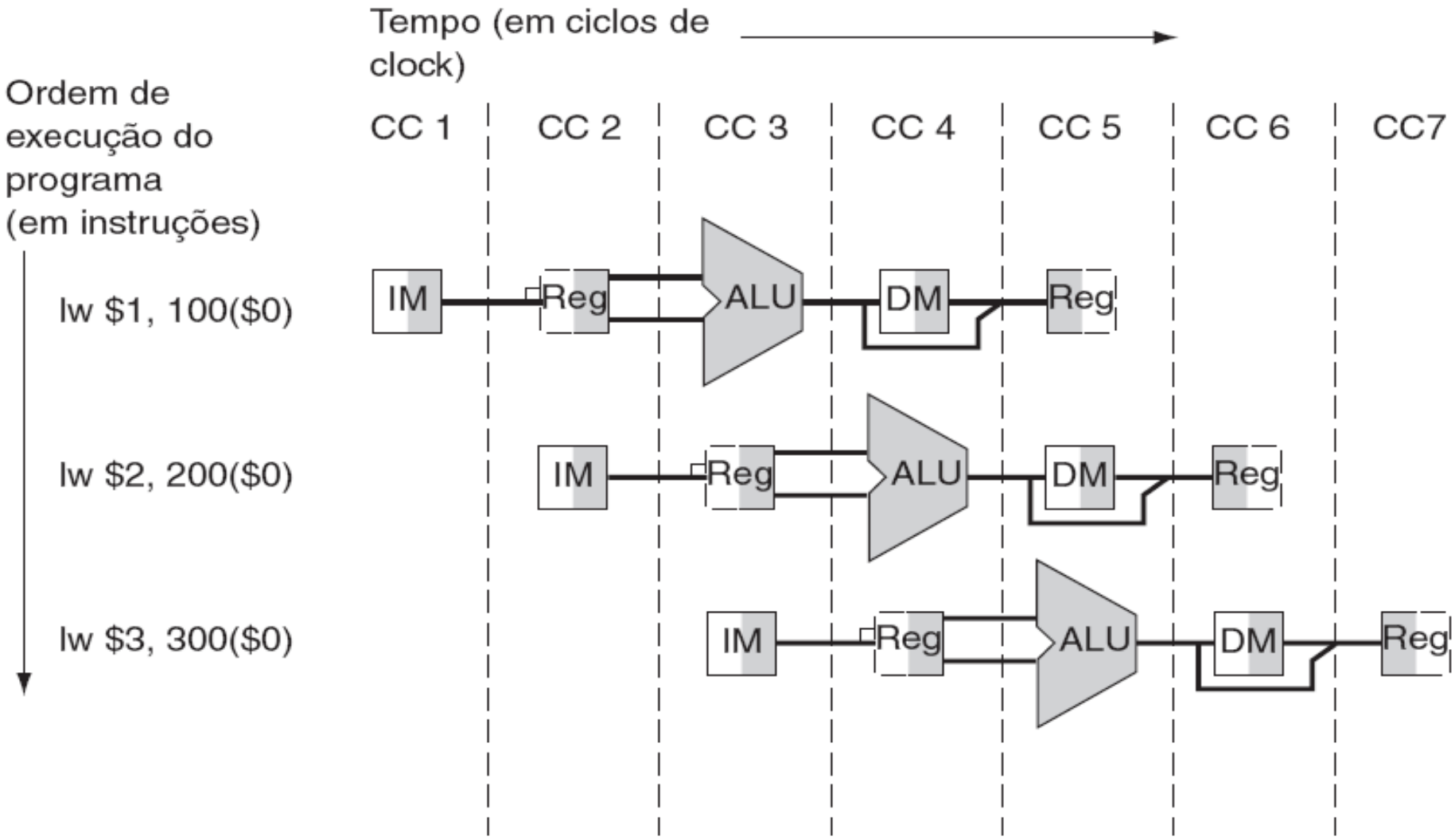

IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0 M u x 1
PC
Endereço
Memória de instruções
Instrução
Registrador de leitura 1
Registrador de leitura 2
Registradores
Registrador para escrita
Dados para escrita
Dados da leitura 1
Dados da leitura 2
16
Extensão de sinal
32
Desloc. 2 esq.
Resultado Add do Somador
ALU
Zero
Resul-tado da ALU
Endereço
Dados da leitura
Memória de dados
Dados para escrita

Representação Gráfica do *Pipeline*

- *Pipelining* pode ser difícil de entender, pois muitas instruções estão executando simultaneamente em um único caminho de dados em cada ciclo de *clock*. Para auxiliar existem dois estilos básicos de figuras de *pipeline*:
  - Diagramas de *pipeline* com múltiplos ciclos de *clock*;
  - Diagramas de *pipeline* com único ciclo de *clock*.

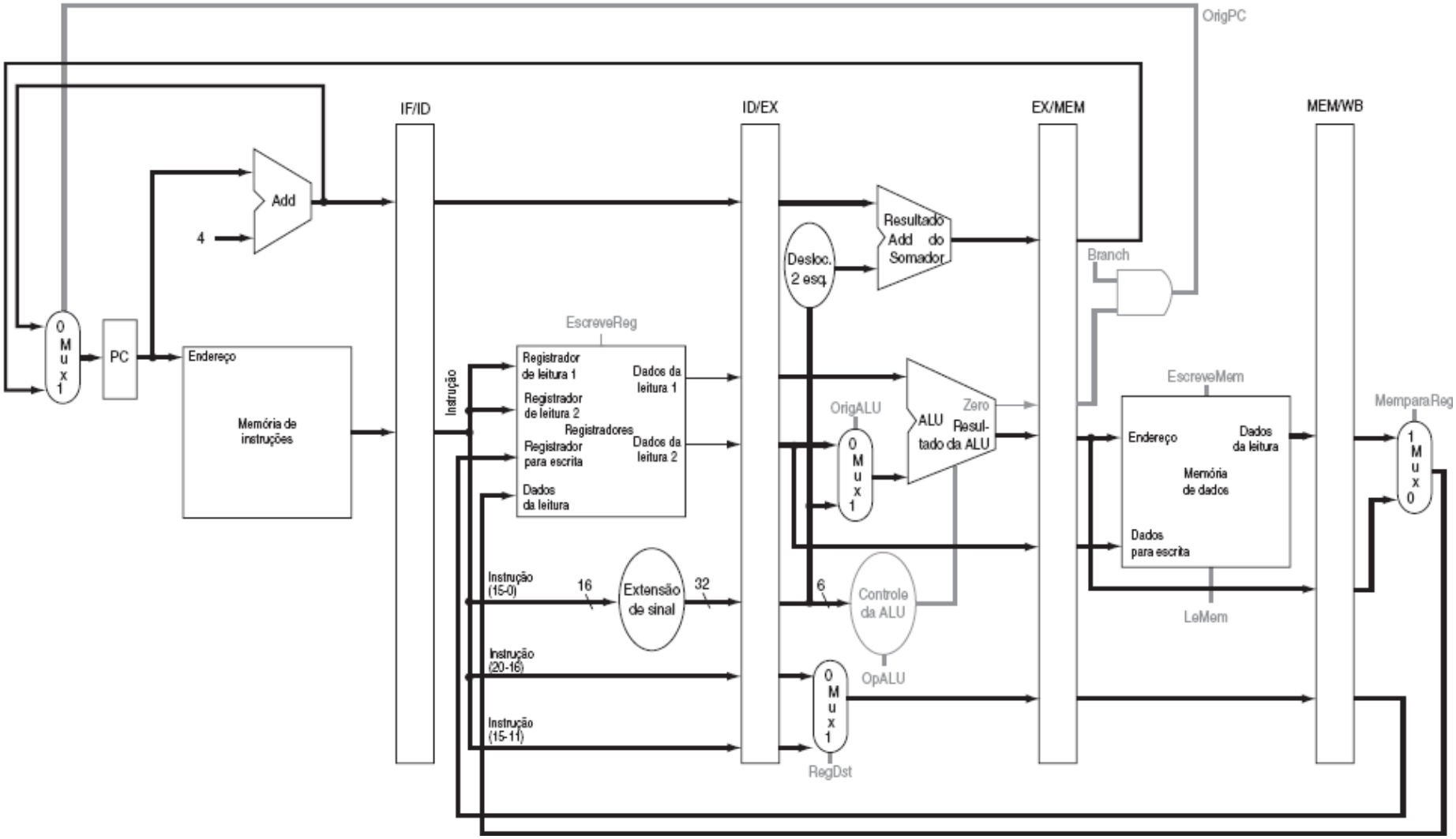
Representação Gráfica do *Pipeline*

- Os diagramas com múltiplos ciclos de clock são mais simples, mas não contêm todos os detalhes.
- Os diagramas de ciclo de *clock* mostram o estado do caminho de dados inteiro durante um único ciclo de *clock*, e normalmente todas as cinco instruções no *pipeline* são identificadas por rótulos acima de seus respectivos estágios do *pipeline*.


Tempo (em ciclos de clock)
Ordem de execução do programa (em instruções)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC7
lw $1, 100($0)
lw $2, 200($0)
lw $3, 300($0)
IM
Reg
ALU
DM
Reg

Controle de um *Pipeline*

- O esquema a ser visto, toma emprestado a lógica de controle do caminho de dados multiciclo para:
  - A origem do PC;
  - O número do registrador destino;
  - O controle da ULA.
- Os bits da lógica de controle da ULA precisam ser incluídos no registrador de *pipeline* ID/EX, eles são retirados do campo imm (*immediate*) com sinal estendido (os 6 bits menos significativos).


OrigPC
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0 M u x 1
PC
Endereço
Memória de instruções
Instrução
EscreveReg
Registrador de leitura 1
Registrador de leitura 2
Registradores
Registrador para escrita
Dados da leitura
Dados da leitura 1
Dados da leitura 2
Desloc. 2 esq.
Resultado Add do Somador
Branch
OrigALU
ALU
Zero
Resul-tado da ALU
EscreveMem
Endereço
Dados da leitura
Memória de dados
Dados para escrita
LeMem
MemparaReg
1 M u x 0
Instrução (15-0)
16
Extensão de sinal
32
6
Controle da ALU
OpALU
Instrução (20-16)
Instrução (15-11)
RegDst

Controle de um *Pipeline*

Instrução	Linhas de controle do estágio de cálculo de endereço/execução				Linhas de controle do estágio de acesso à memória			Linhas de controle do estágio de escrita do resultado	
	RegDst	OpALU1	OpALU0	OrigALU	Branch	LeMem	Escreve Mem	Escreve Reg	Mem para Reg
Formato 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

Instrução	RegDst	OrigALU	Mempara Reg	Escreve Reg	Le Mem	Escreve Mem	Branch	ALUOpl	ALUOp0
formato R	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Controle de um *Pipeline*

- Para especificar o controle para o *pipeline*, só precisamos definir os valores de controle durante cada estágio do *pipeline*. Como cada linha de controle está associada a um componente ativo em apenas um estágio do pipeline, podemos dividir as linhas de controle em cinco grupos, de acordo com o estágio do *pipeline*:

Controle de um *Pipeline*

- **Busca de instruções**: os sinais de controle para ler a memória de instruções e escrever o PC sempre são ativados;
- **Decodificação de instruções/leitura do banco de registradores**: como no estágio anterior, a mesma coisa acontece em cada ciclo de *clock*, de modo que não existem linhas de controle opcionais a definir;
- **Execução/cálculo de endereço**: os sinais a serem definidos são RegDst, OpALU e OrigALU;

Controle de um *Pipeline*

- **Acesso à memória**: as linhas de controle definidas nesse estágio são Branch, LeMem e EscreveMem. Esses sinais são definidos pelas instruções *branch if equal*, *load* e *store*, respectivamente.
- **Escrita do resultado**: as duas linhas de controle são MemparaReg e EscreveReg.

- Como é percebido, os valores e significados são os mesmos, mas agora as nove linhas de controle estão agrupadas por estágio do *pipeline*.

Controle de um *Pipeline*

- A maneira mais fácil para implementar o controle é estender os registradores do *pipeline* para incluir as informações de controle.

Controle de um *Pipeline*