ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Arquitetura RISC-V **Pipelined**

Profª. Fabiana F F Peres

Apoio: Camile Bordini

Pipeline

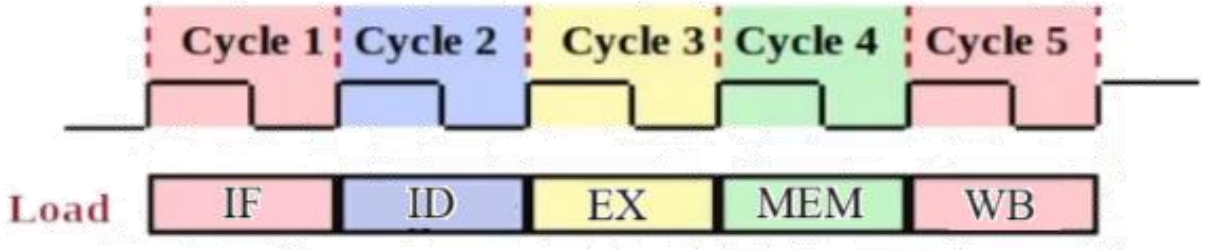
(via de dados)

- *Pipelining* de cinco estágios

- Durante os ciclos de clock, cinco instruções estarão em execução

- Divisão da via de dados em cinco partes, onde cada uma delas terá o nome do estágio correspondente

Pipeline



1. Busca da **instrução** e incremento do PC → **IF: *Intruction Fetch***

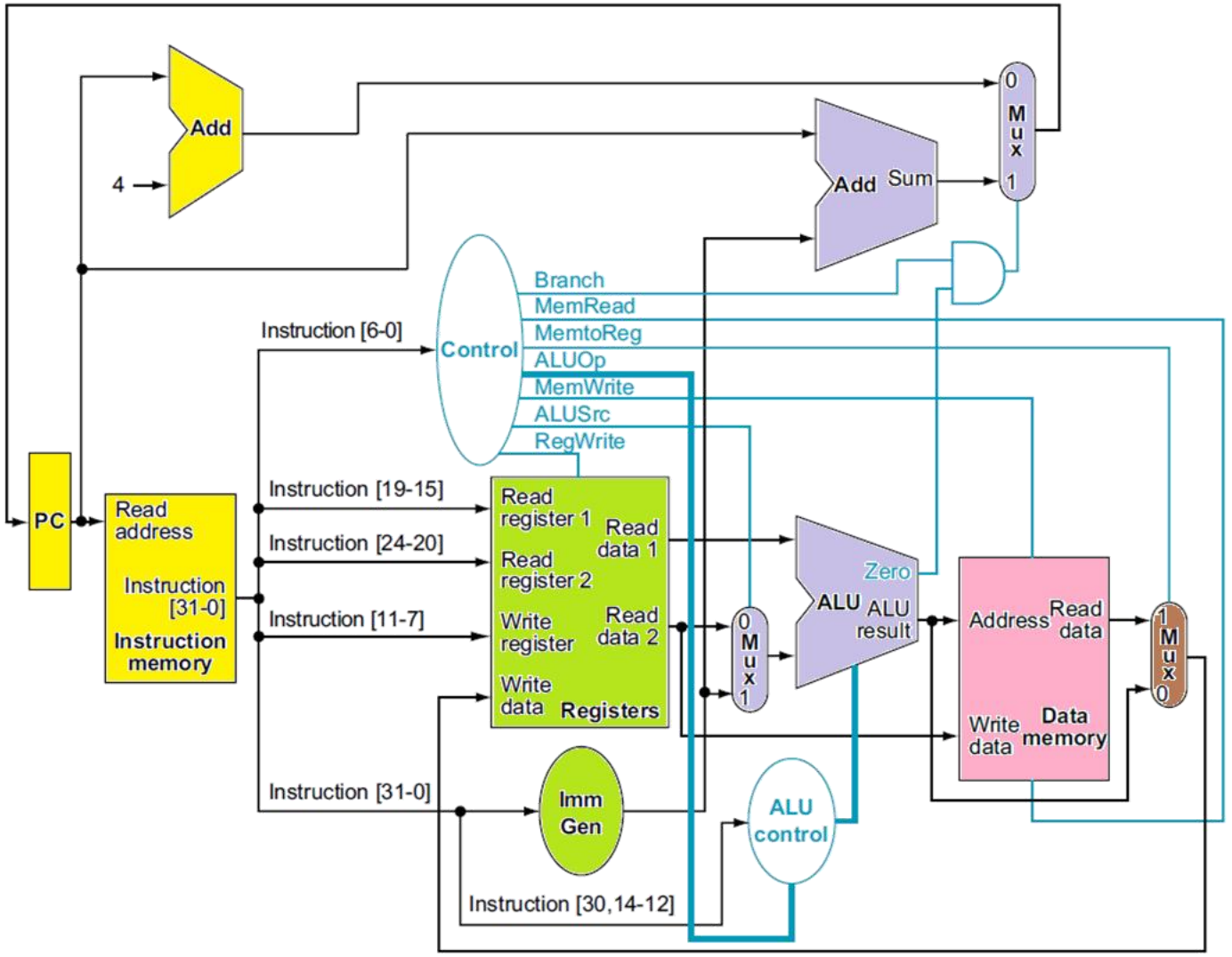
2. Acesso ao banco de registradores e **decodificação** da instrução → **ID: *Intruction Decode***

3. **Execução** da operação ou cálculo do endereço de memória → **EX: *Execução***

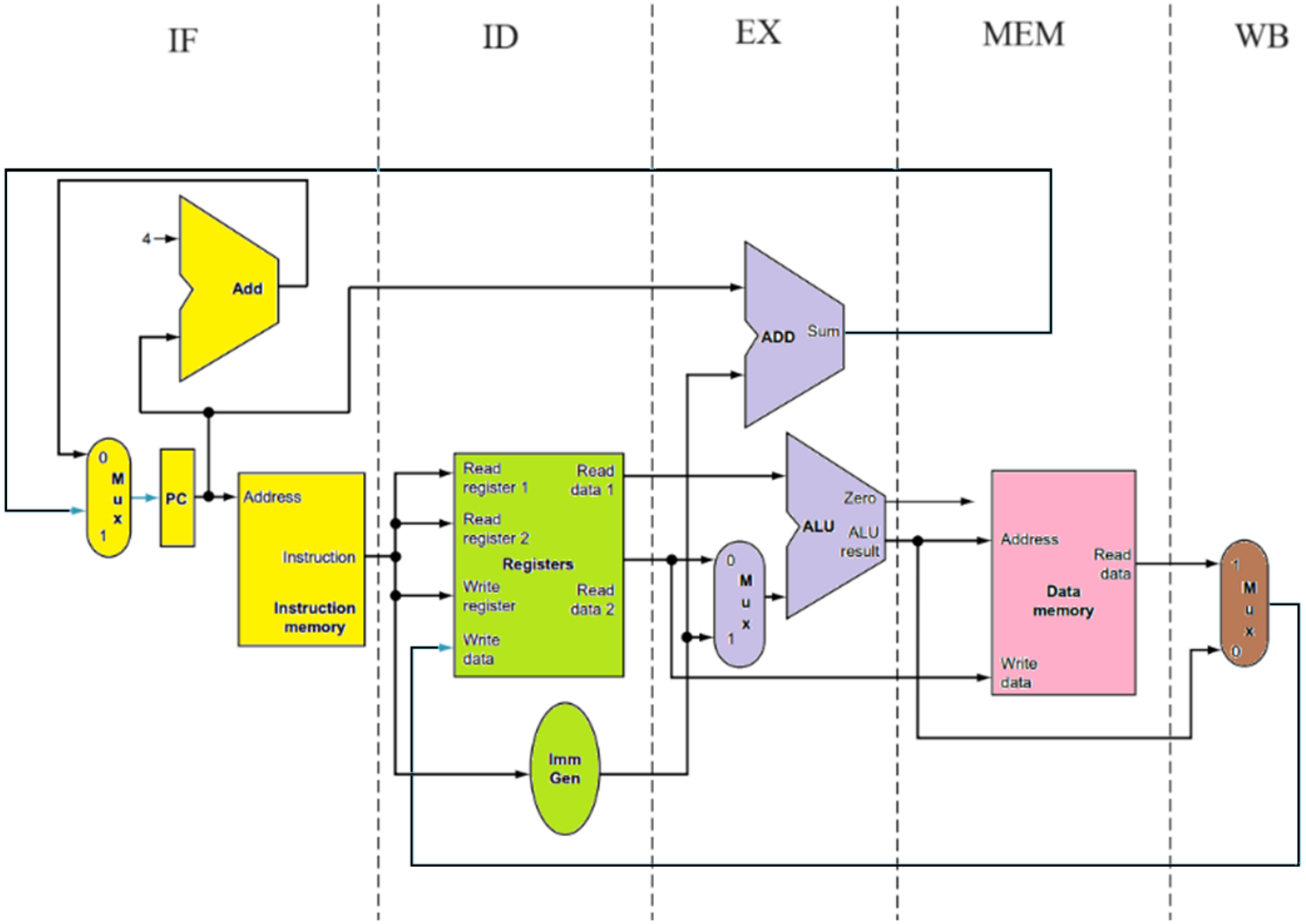
4. Acesso à **memória** de dados (para leitura ou escrita), se necessário. → **MEM: *Memória***

5. **Escreve** o resultado dentro de um **registrador**, se necessário → **WB: *Write Back***

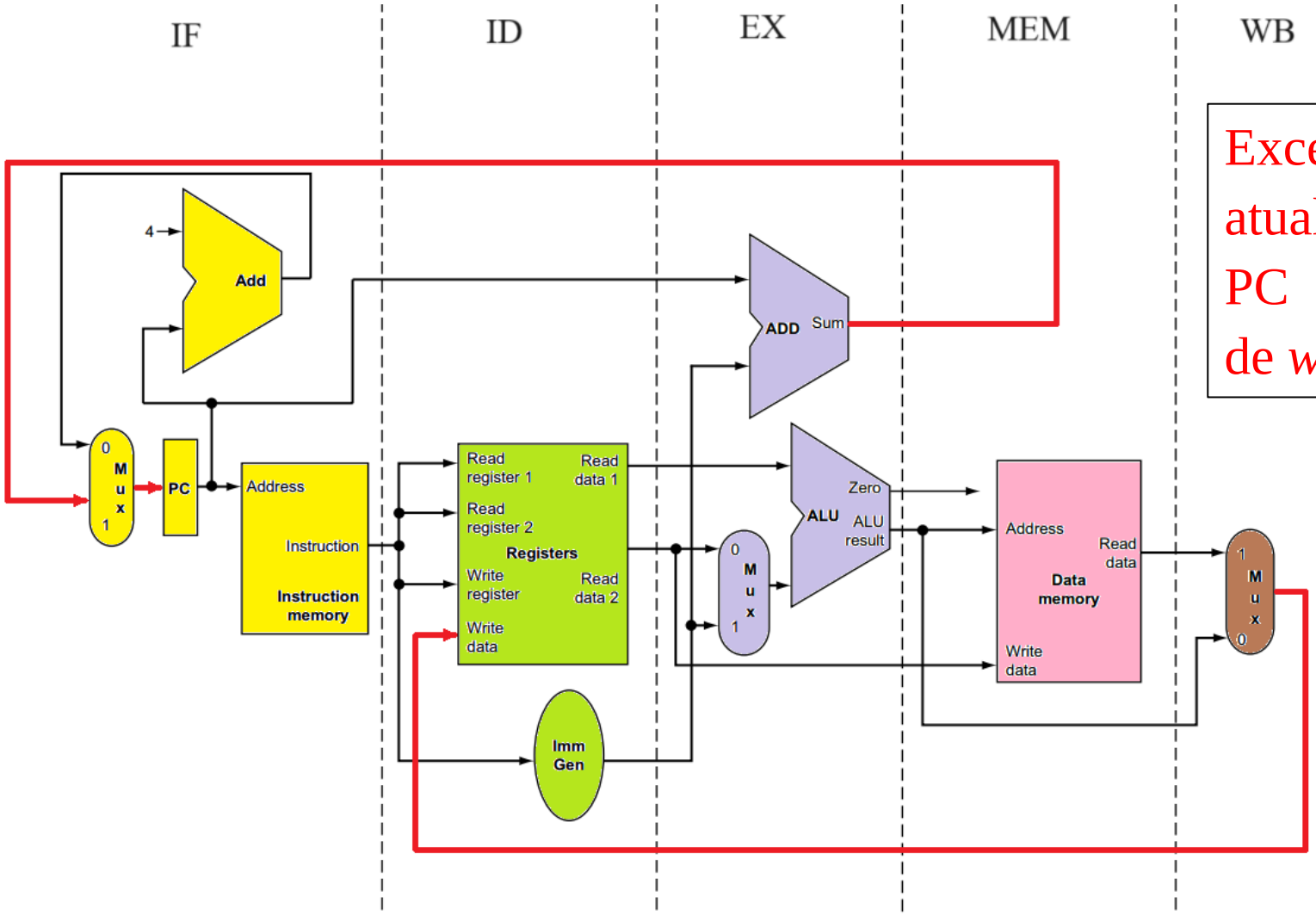
•Vamos lembrar da via de dados Monociclo, se inseríssemos os estágios do *pipeline*, como ficaria?



•Cada etapa pode ser mapeada por um caminho da esquerda para a direita na Monociclo. Mas <u>todos</u> os fluxos vão nesse sentido?



•Cada etapa pode ser mapeada por um caminho da esquerda para a direita na Monociclo. Mas <u>todos</u> os fluxos vão nesse sentido?



Exceto a atualização do PC e a etapa de *write-back*

Pipeline de 5 estágios

•A figura anterior corresponde aproximadamente como o caminho de dados é desenhado

•Instruções e dados movem-se geralmente da esquerda para a direita através dos 5 estágios

•As 2 exceções a esse fluxo:

- A seleção do próximo valor do PC, escolhendo entre o PC incrementado e o endereço da *branch* do estágio MEM
- O estágio de *write-back*, que coloca o resultado de volta no banco de registradores no meio do caminho de dados

Pipeline de 5 estágios

•*Questão*: Como os dados que fluem da direita para a esquerda afetam a instrução atual?

–Esses movimentos reversos de dados influenciam apenas instruções posteriores no pipeline, não na atual.

•Que tipo de conflitos as 2 exceções podem gerar?

- Seleção do próximo valor do PC: **Conflitos de controle**
- Estágio de *write-back*: **Conflito de dados**

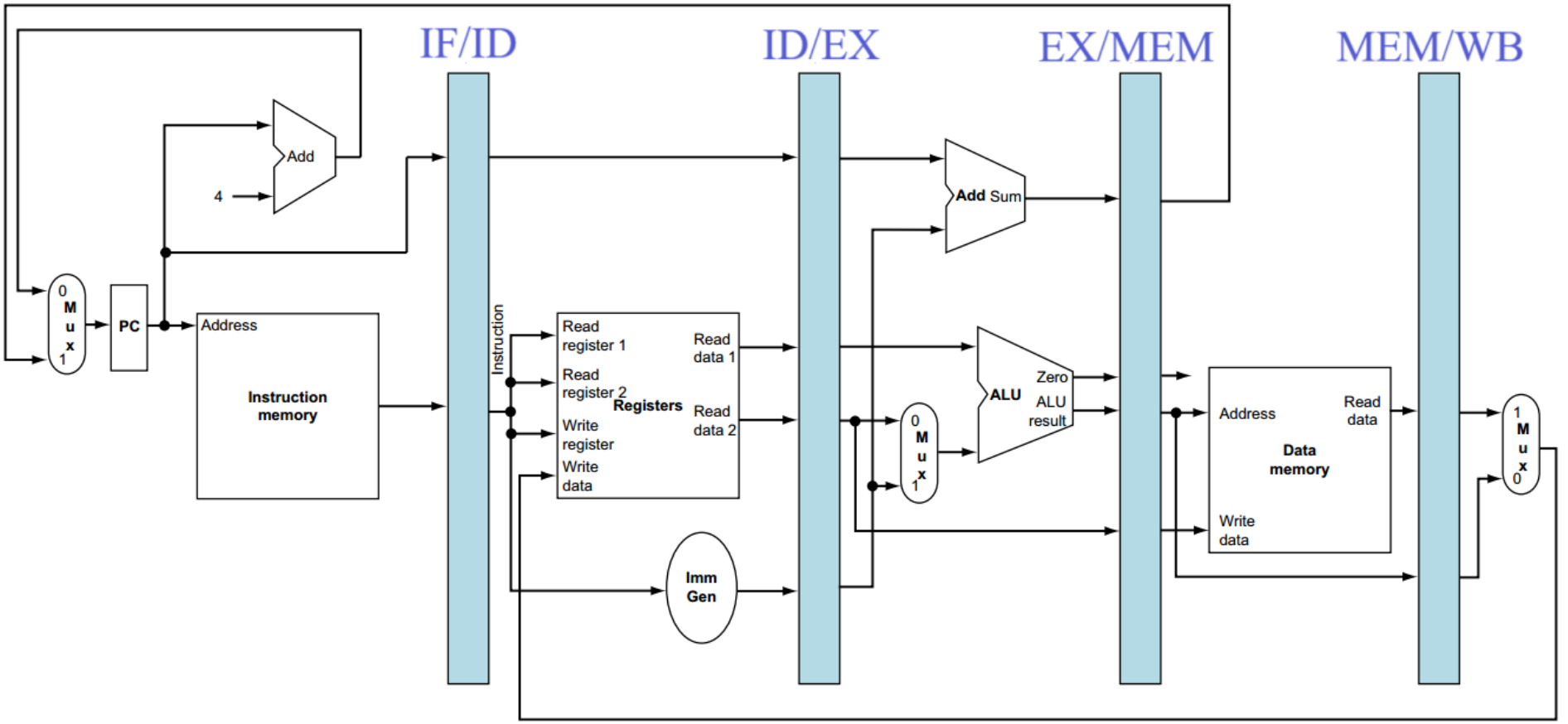
Pipeline de 5 estágios

•Vamos analisar, por exemplo, a memória de instruções: ela é usada apenas em um dos 5 estágios de uma instrução "**x**" qualquer. E por esse motivo, ela pode ser compartilhada com outras instruções "**y**", "**z**", durante os outros 4 estágios de "**x**"

•Mas para que o valor lido da memória de instruções pela instrução "**x**" possa ser usado nos seus outros 4 estágios, este valor deve ser salvo em algum lugar para que não seja sobrescrito no próximo ciclo!

•Argumentos semelhantes se aplicam a todos os estágios do pipeline, portanto devemos colocar **registradores** onde quer que haja linhas divisórias entre os estágios!

Registradores Pipeline



IF/ID, ID/EX, EX/MEM e MEM/WB

Registradores Pipeline

•As instruções avançam durante cada ciclo de um **banco de registradores de *pipeline*** para o próximo

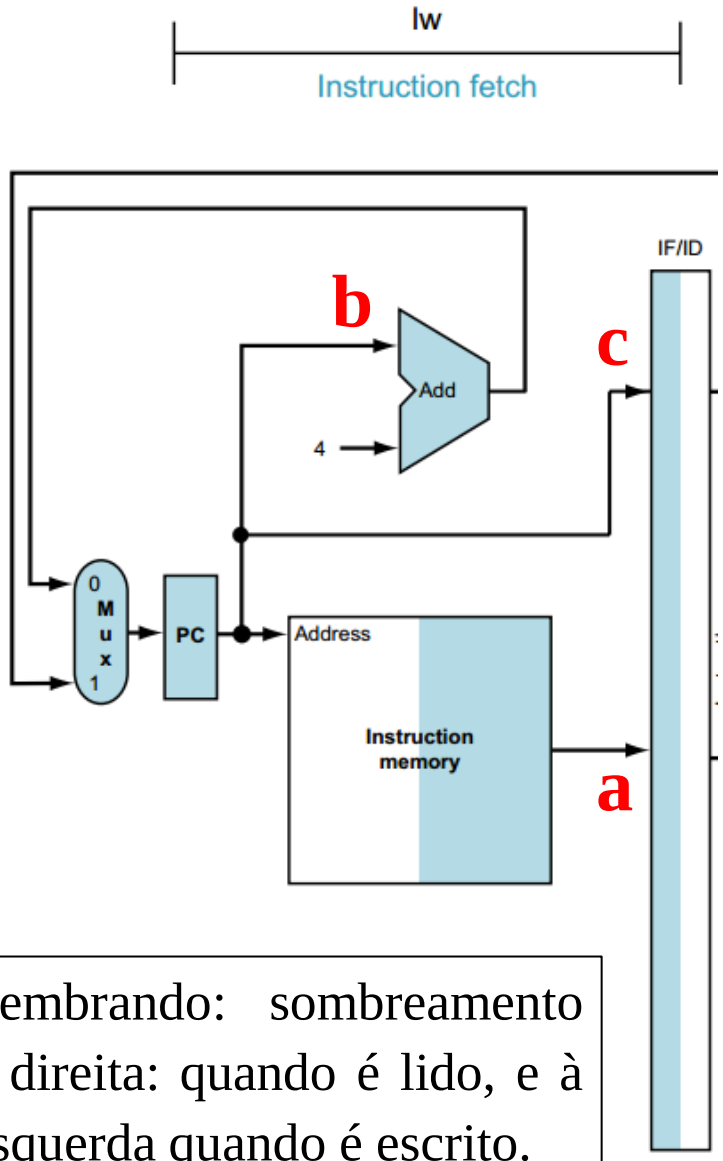
•Esses registradores são nomeados pelos nomes dos estágios que os separam

•Observe que não há registrador pipeline ao final de *write-back*

•Como todas as instruções devem atualizar algum estado do processador (o banco de registradores, a memória, ou o PC), um registrador pipeline pode ser visto como uma redundância para o estado do processador que é atualizado

Ex: Caminho de dados - Load

1. IF



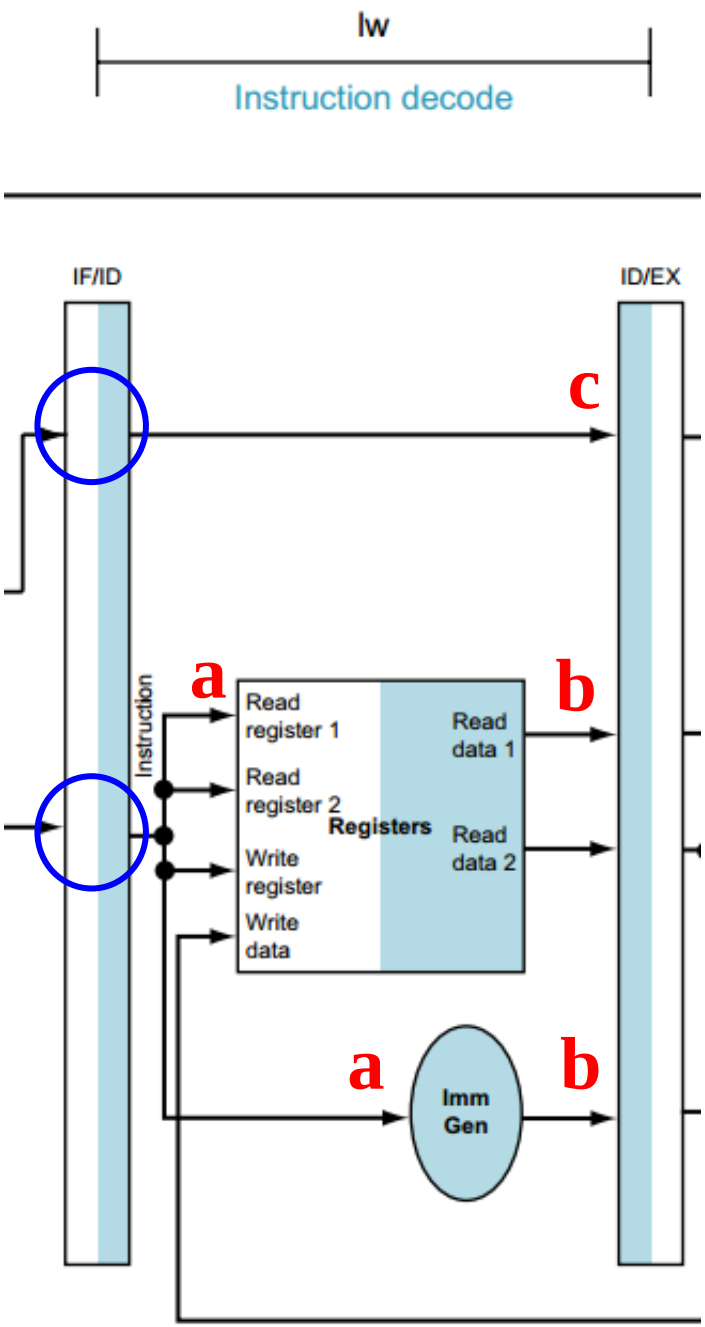
Lembrando: sombreamento à direita: quando é lido, e à esquerda quando é escrito.

1. Busca da instrução:

a. Por exemplo, uma **instrução *Load*** é lida da memória (no endereço que PC guarda), e colocada no registrador IF/ID.

b. O **endereço do PC** é incrementado em 4 e gravado de volta em PC para estar pronto para o próximo ciclo.

c. Este **PC+4** também é salvo no registrador IF/ID caso seja necessário posteriormente para uma instrução, como *Beq* (como não se sabe que tipo de instrução está sendo buscada, deve se preparar para qualquer uma, passando informações possivelmente necessárias pelo pipeline).



2. ID

2. <u>Decodificação da instrução e leitura do banco de registradores</u>:

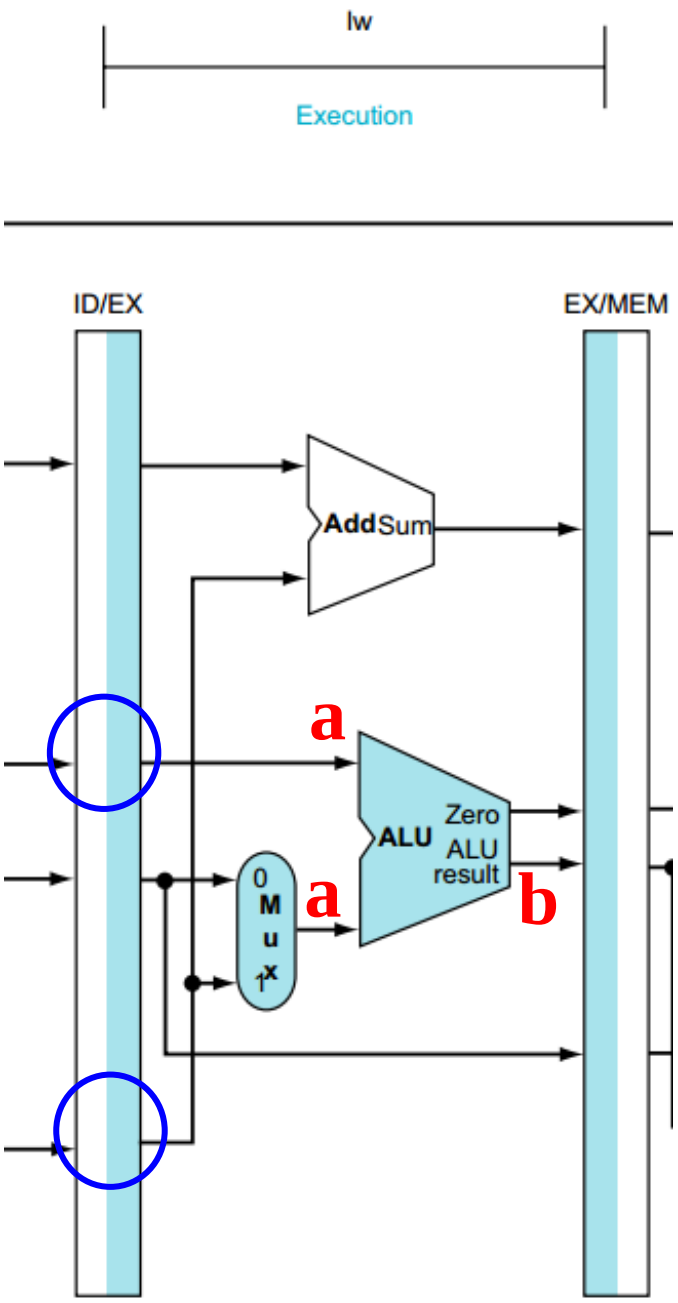
a. Parte da instrução do registrador IF/ID fornece o **campo imediato**, que é sinal-estendido para 64 bits, e o **registrador rs1** a ser lido.

b. **Os dois valores** são armazenados no registrador pipeline ID/EX,

c. juntamente com o **endereço do PC**.

Obs: Novamente transferimos tudo o que pode ser necessário para qualquer instrução durante um ciclo de clock posterior.

3. *EX*

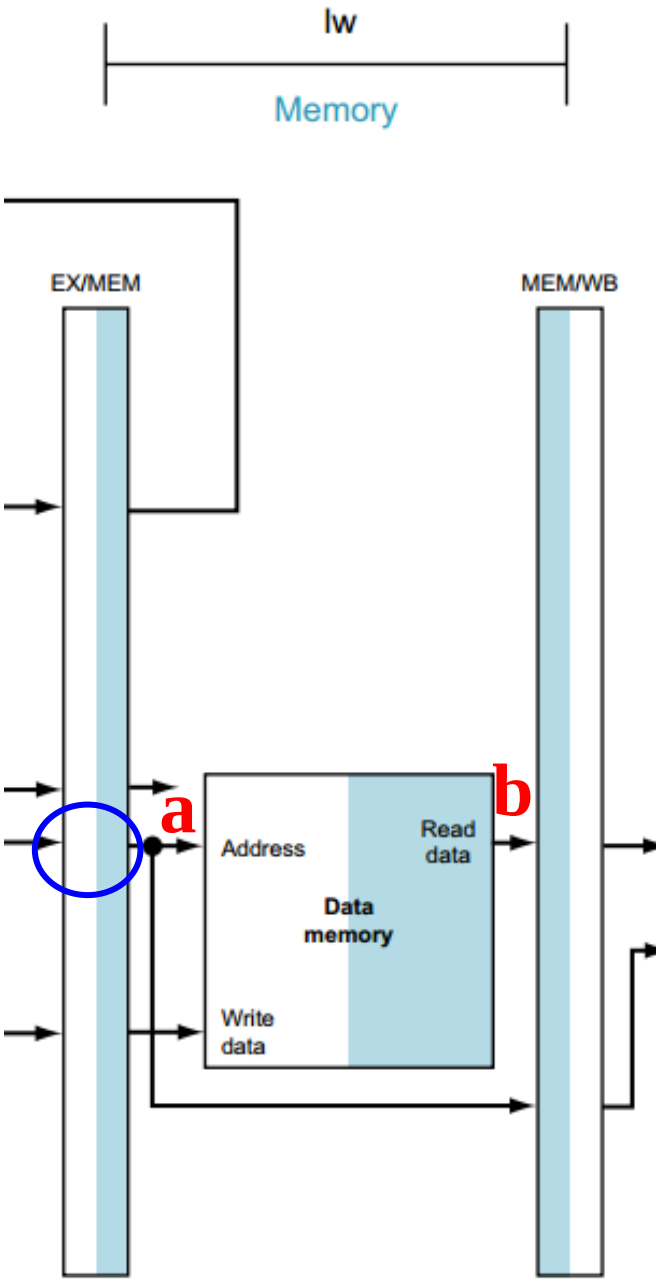


3. <u>Cálculo de execução ou endereço</u>:

a. A instrução *Load* lê o **conteúdo do registrador rs1** e o <u>adiciona</u> com o **imediato estendido** que se encontra no registrador ID/EX, usando a **ALU**.

b. Essa soma é colocada no registrador do pipeline EX/MEM.

4. MEM



4. <u>Acesso à memória</u>:

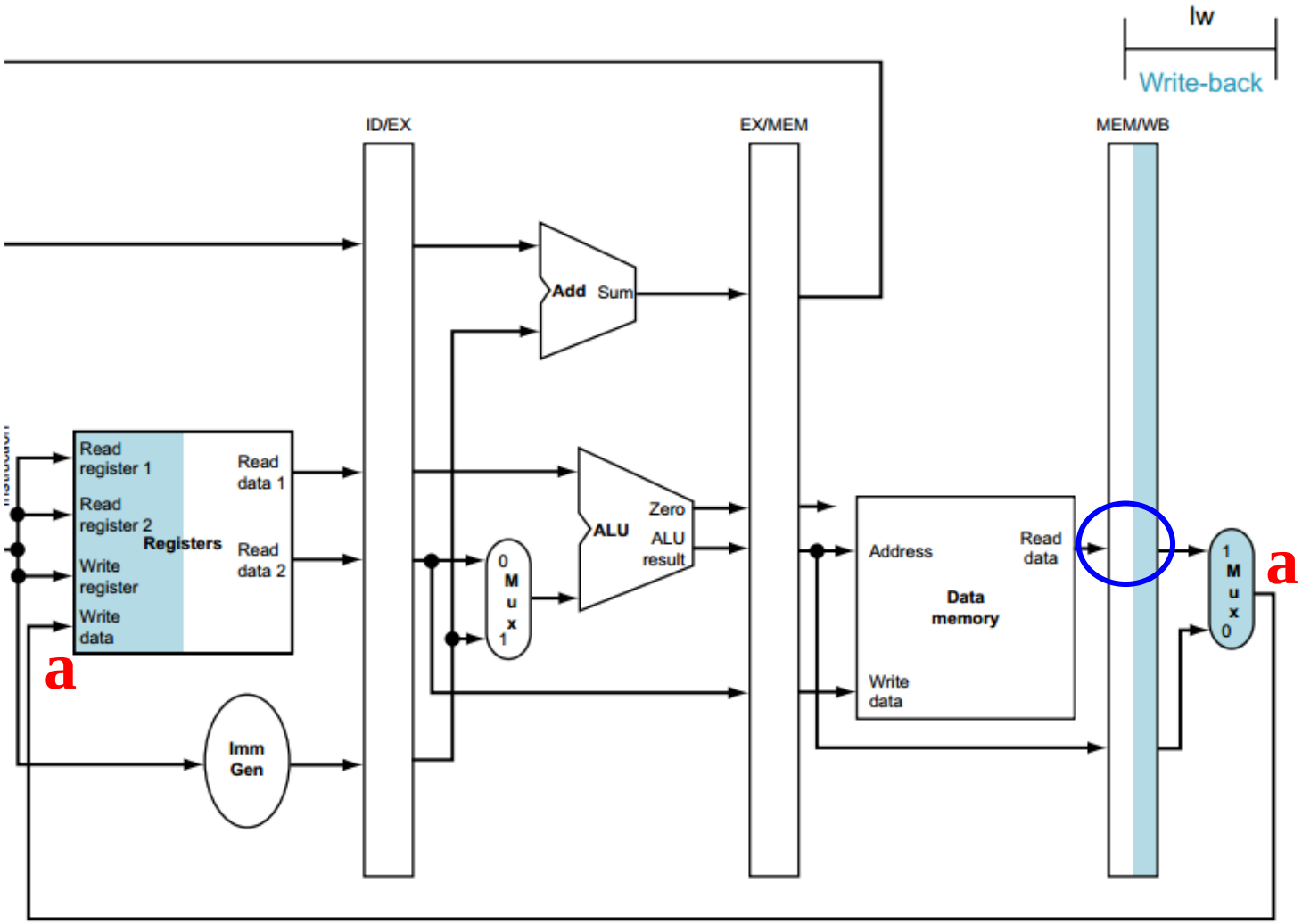
a. A instrução *Load* **lê a memória de dados** (usando o endereço do registrador EX/MEM salvo na etapa anterior)

b. E carrega os dados para o registrador pipeline MEM/WB.

5. WB

Obs: há um bug nessa figura!

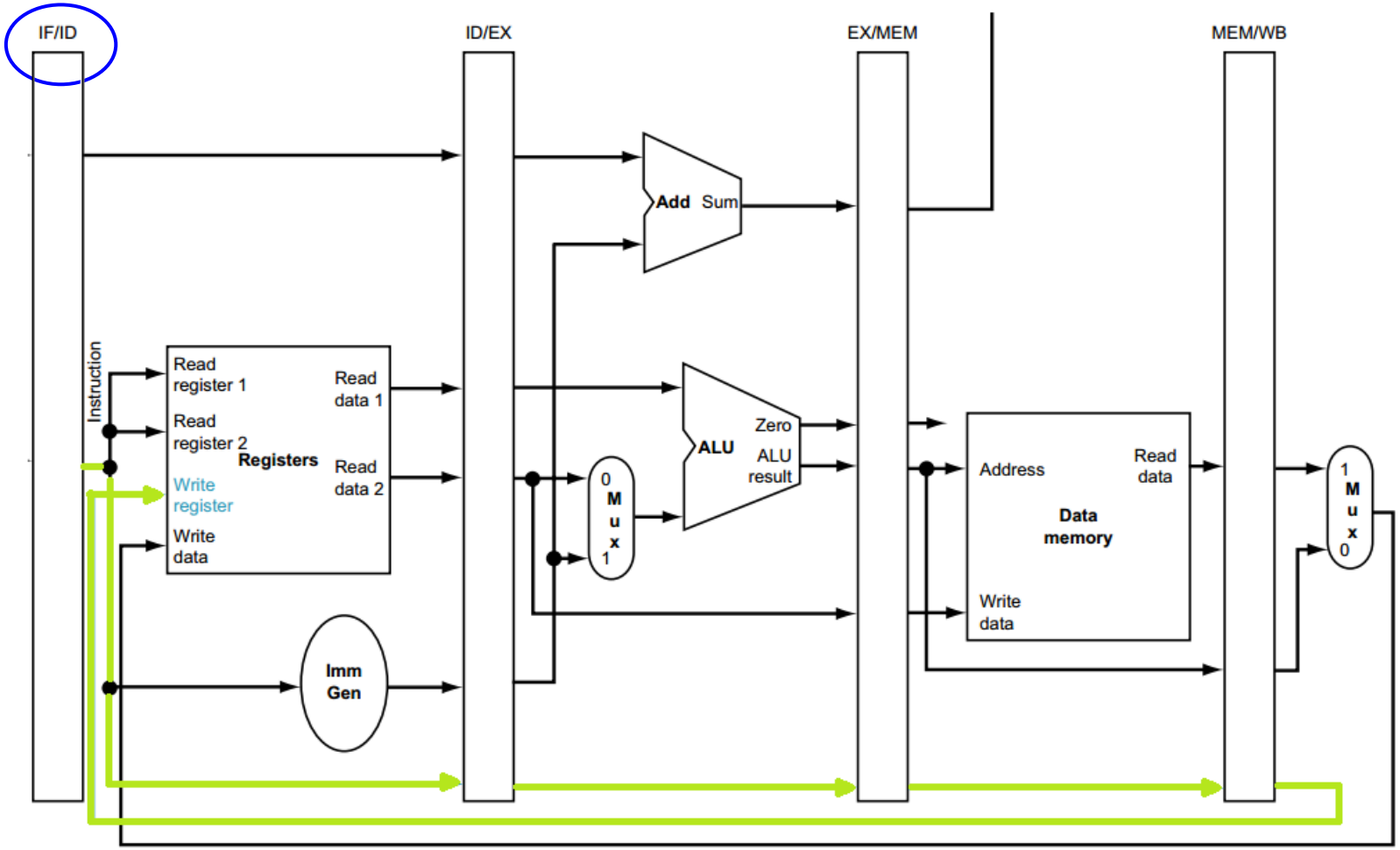
Qual?



5. <u>Write-back</u>:

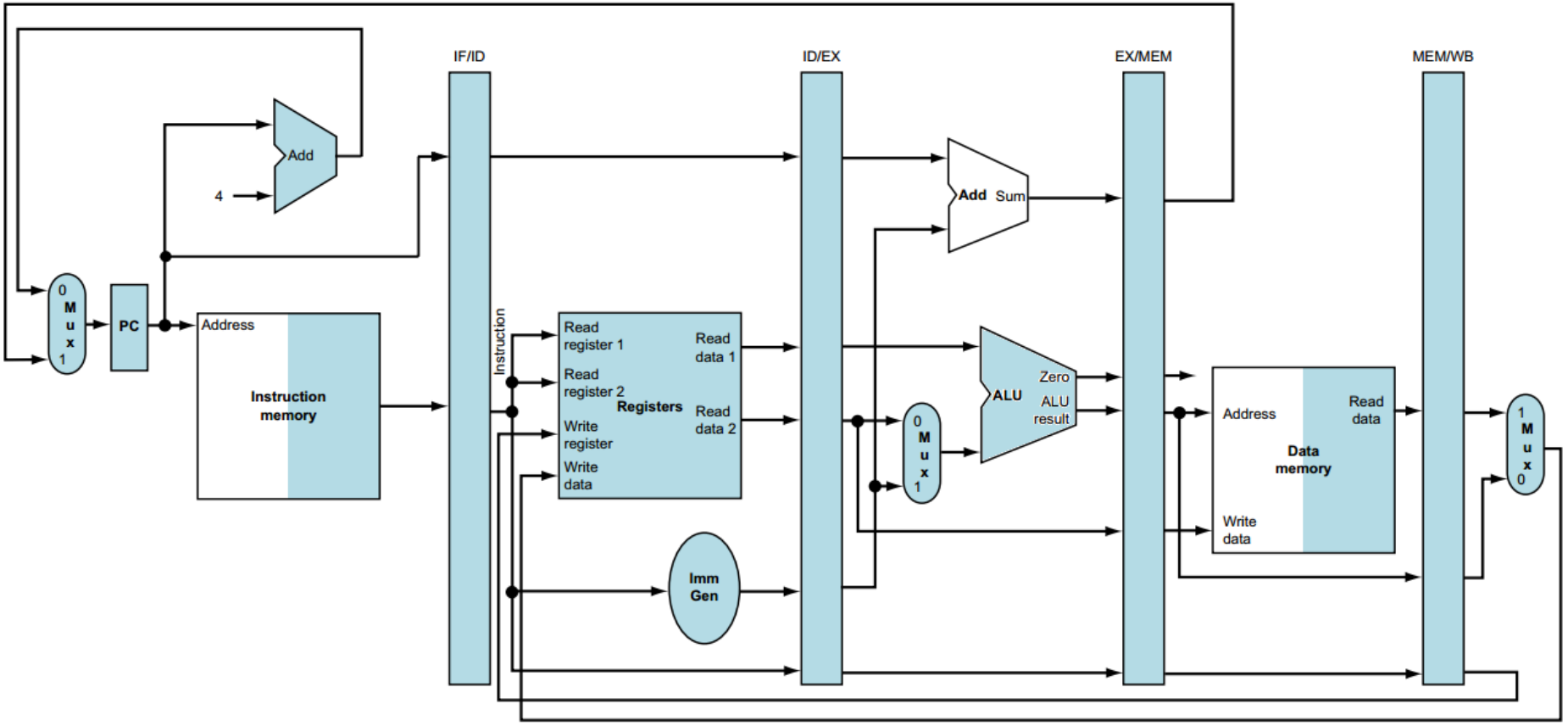
a. A etapa final da *Load* consiste em ler os dados do registrador MEM/WB e **gravá-los devolta no banco de registradores**.

Quando a Load estava em IF/ID é que deve ser fornecido o endereço de rd



O **endereço do registrador de destino (rd)** deve vir de MEM/WB junto com os **dados**. Para isso ele deve ir sendo passado desde o estágio **ID** do *pipeline* até atingir esse registrador, (adicionando mais cinco bits aos últimos três bancos de registradores do *pipeline*)

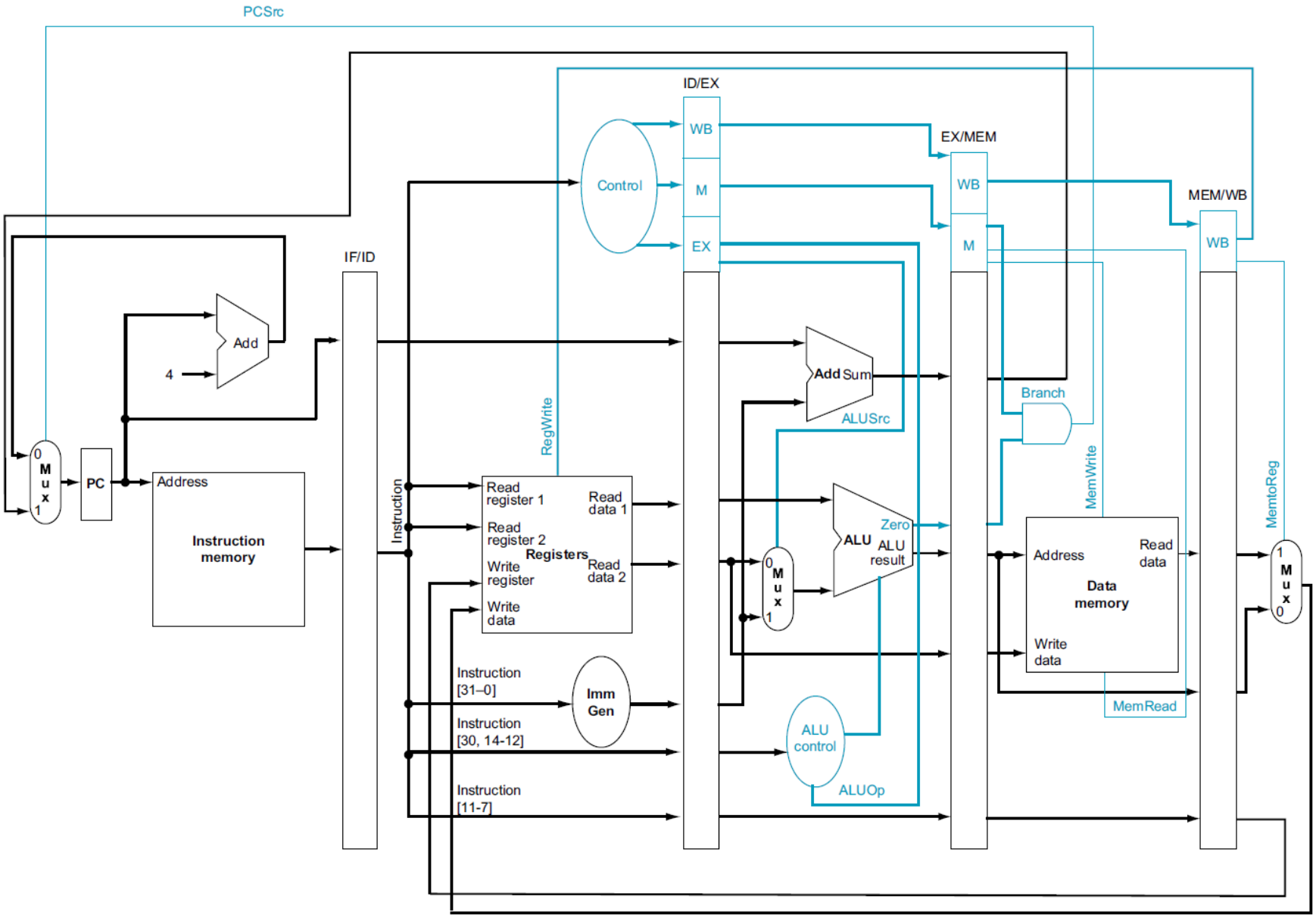
Load em resumo



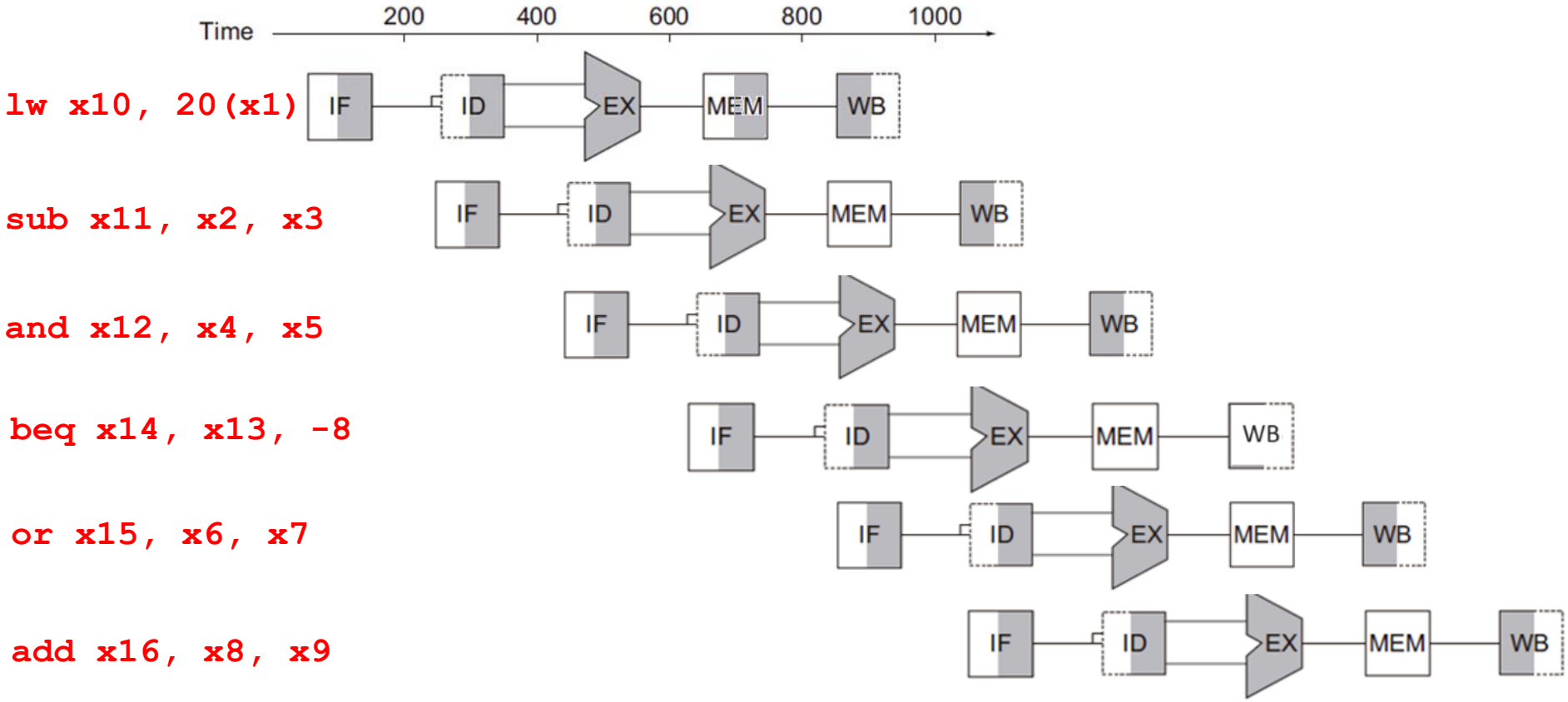
Porção do caminho de dados que é utilizado em todos os 5 estágios da instrução *Load*

Sinais de Controle

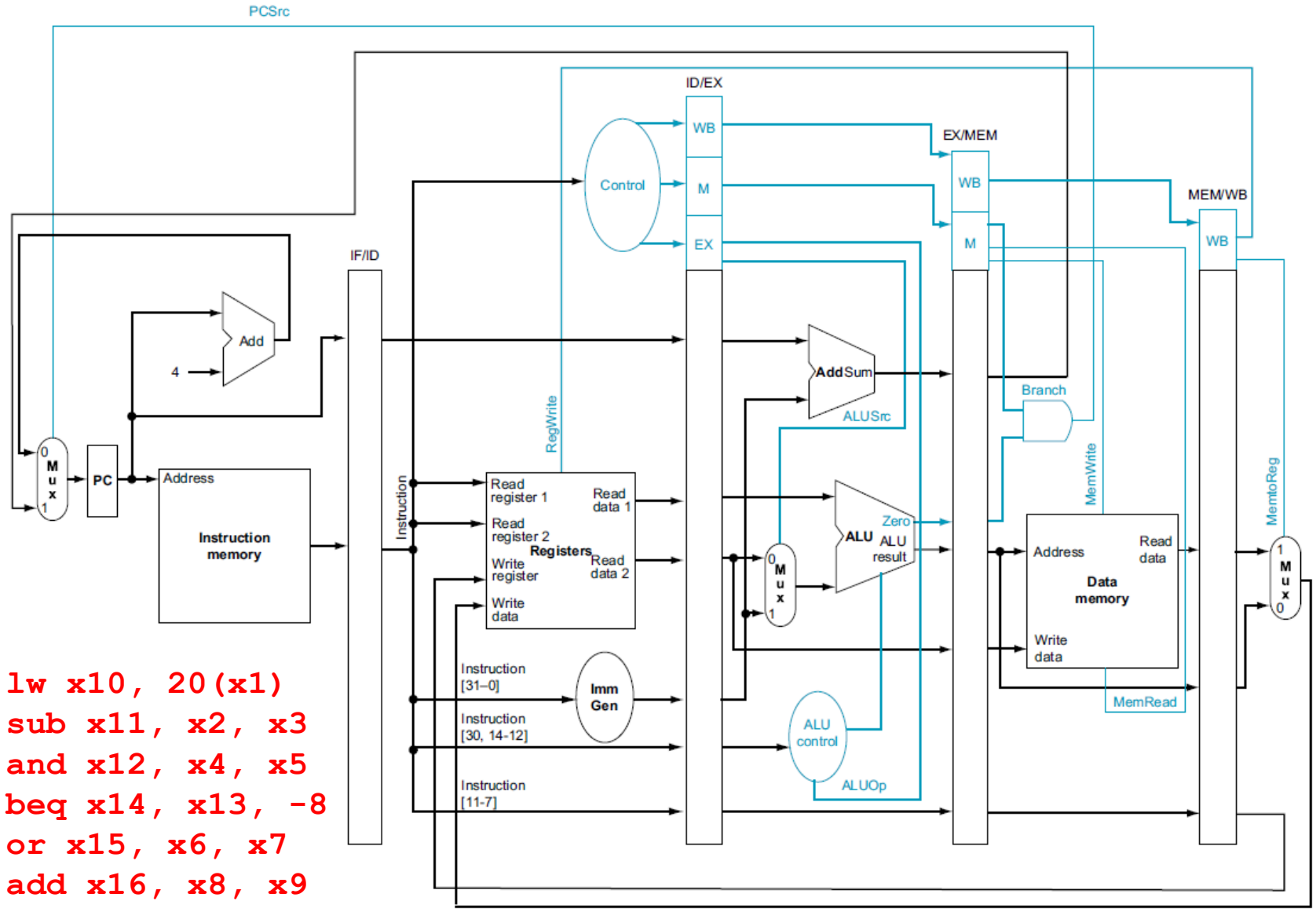
Pipeline com sinais de controle



Pipeline

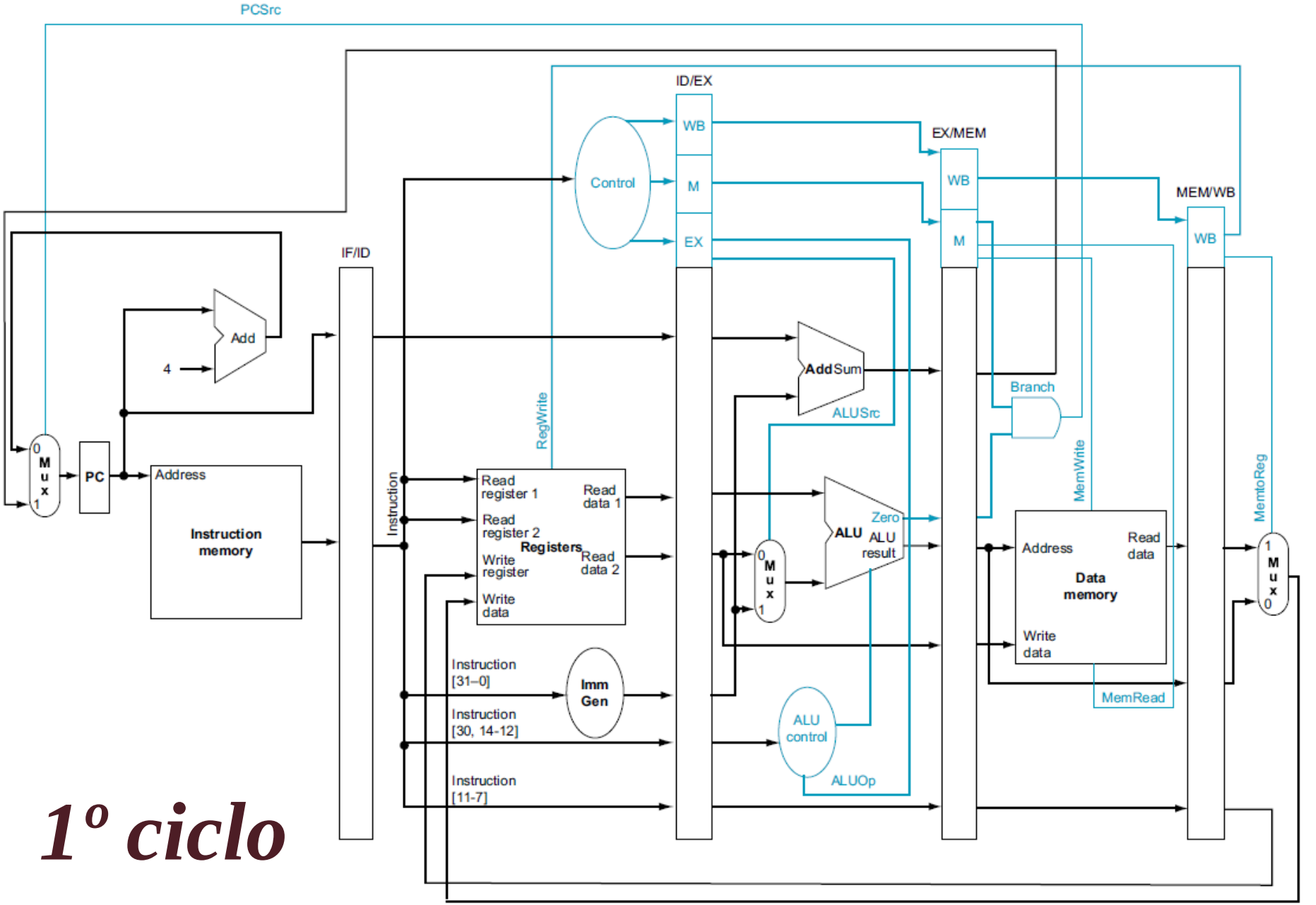

Time
200
400
600
800
1000
lw x10, 20(x1)
sub x11, x2, x3
and x12, x4, x5
beq x14, x13, -8
or x15, x6, x7
add x16, x8, x9
IF
ID
EX
MEM
WB

Ex 1: Considerando que deseja-se executar a sequência de instruções abaixo, identifique onde cada instrução estará no quinto ciclo de clock.



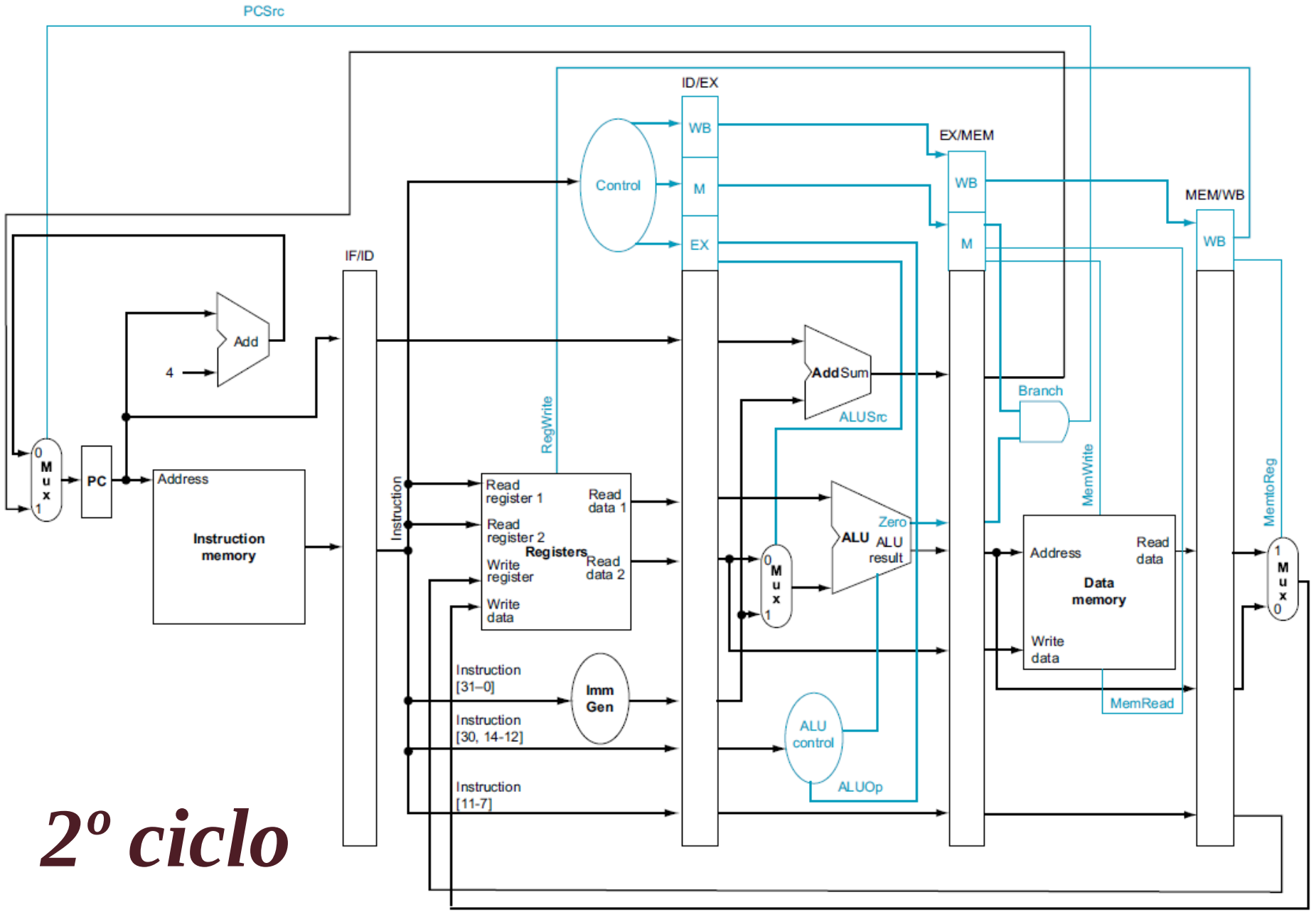

```
(8000)lw x10, 20(x1)
(8004)sub x11, x2, x3
(8008)and x12, x4, x5
(8012)beq x14, x13, -8
(8016)or x15, x6, x7
(8020)add x16, x8, x9
```

IF: `lw x10,20(x1)` | ID: <nop> | EX: <nop> | MEM: <nop> | WB:<nop>



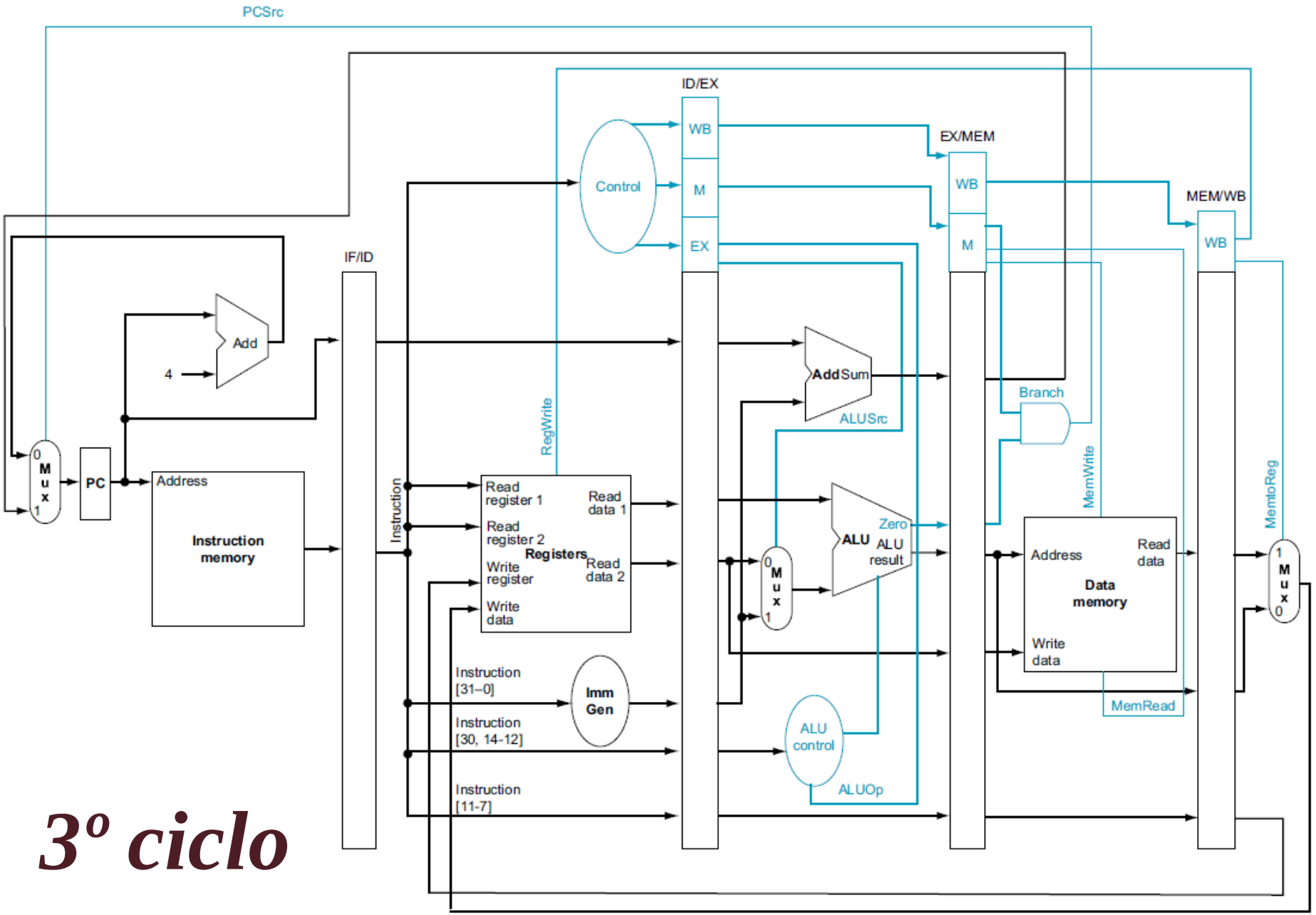
1º ciclo

IF: `sub x11,x2,x3` | ID: `lw x10,20(x1)` | EX:<nop> | MEM:<nop> | WB:<nop>



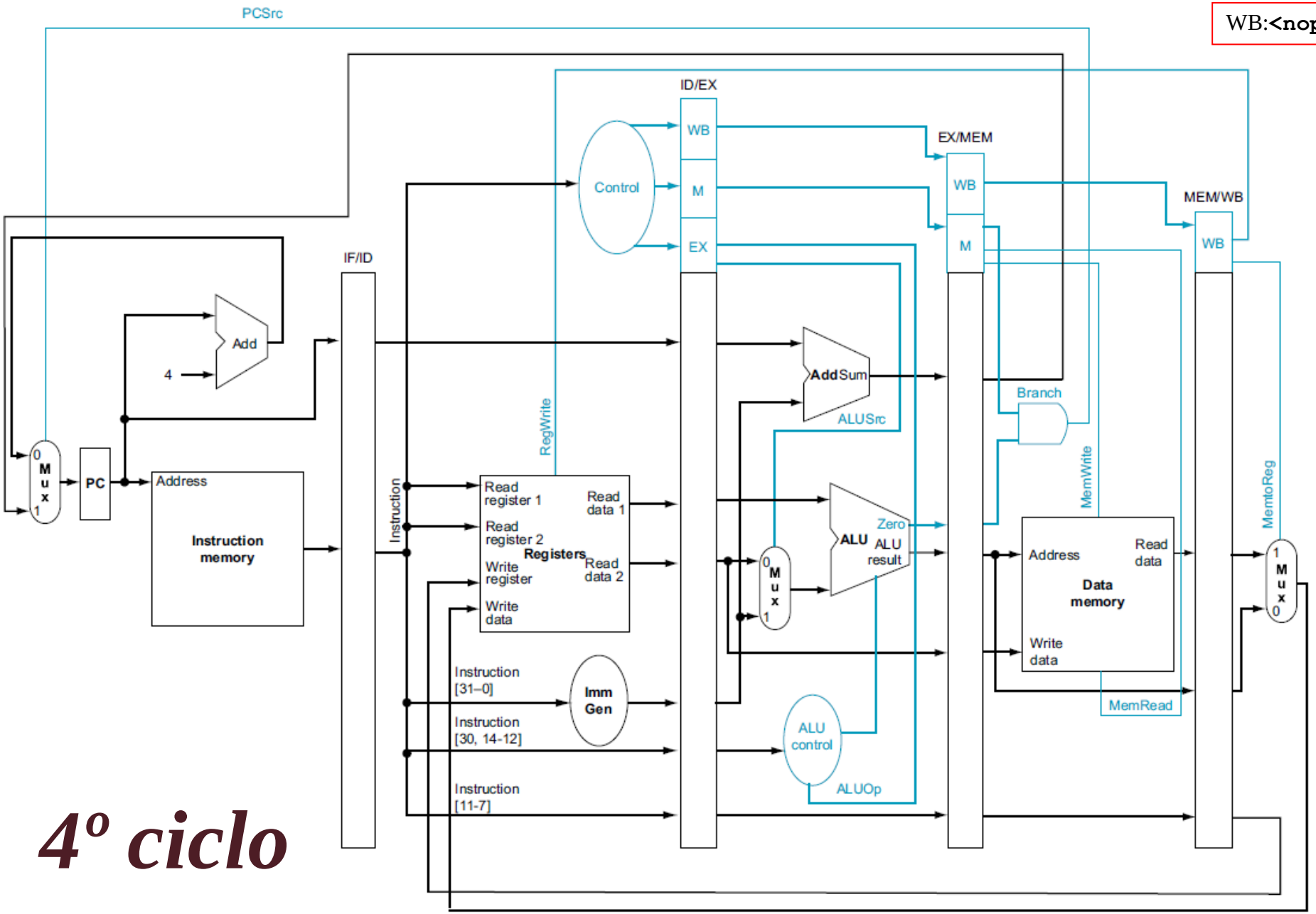
2º ciclo

IF: `and x12,x4,x5` | ID: `sub x11,x2,x3` | EX: `lw x10,20(x1)` | MEM:<nop> | WB:<nop>



3º ciclo

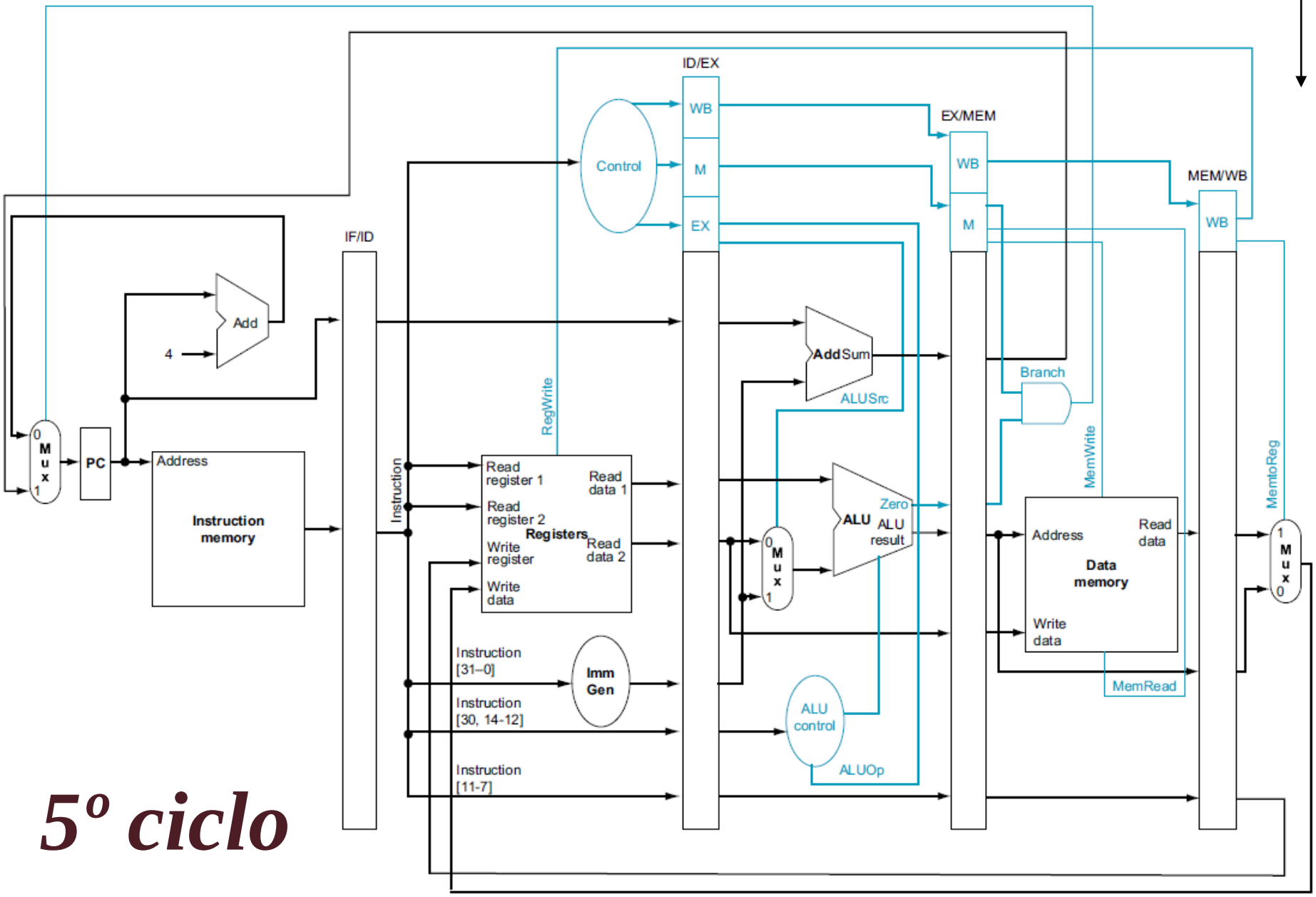
IF: `beq x14,x13,-8` | ID: `and x12,x4,x5` | EX: `sub x11,x2,x3` | MEM: `lw x10,20(x1)` | WB:`<nop>`



4º ciclo

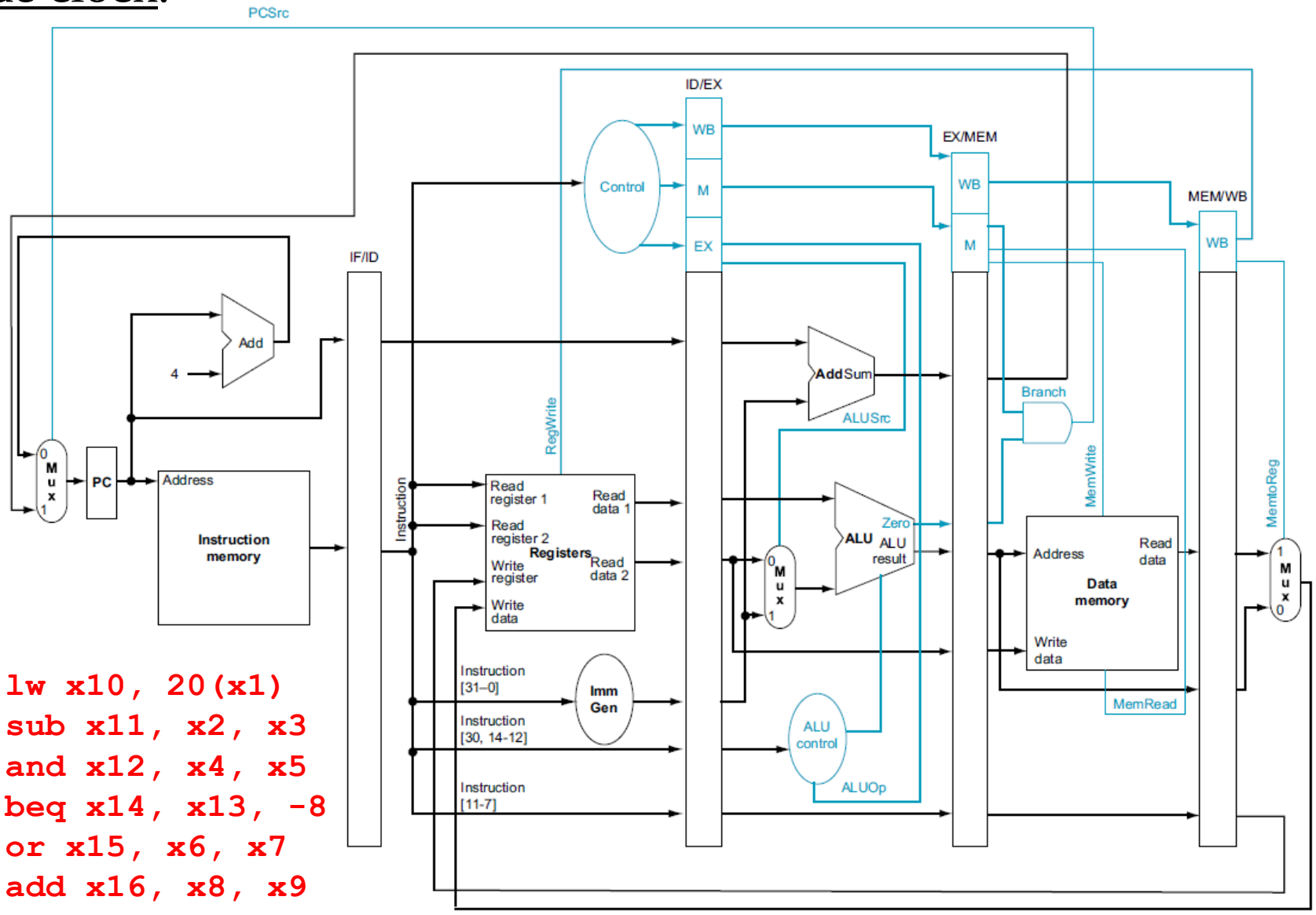
WB: `lw x10,20(x1)`

IF: `or x15,x6,x7` | ID: `beq x14,x13,-8` | EX: `and x12,x4,x5` | MEM: `sub x11,x2,x3`



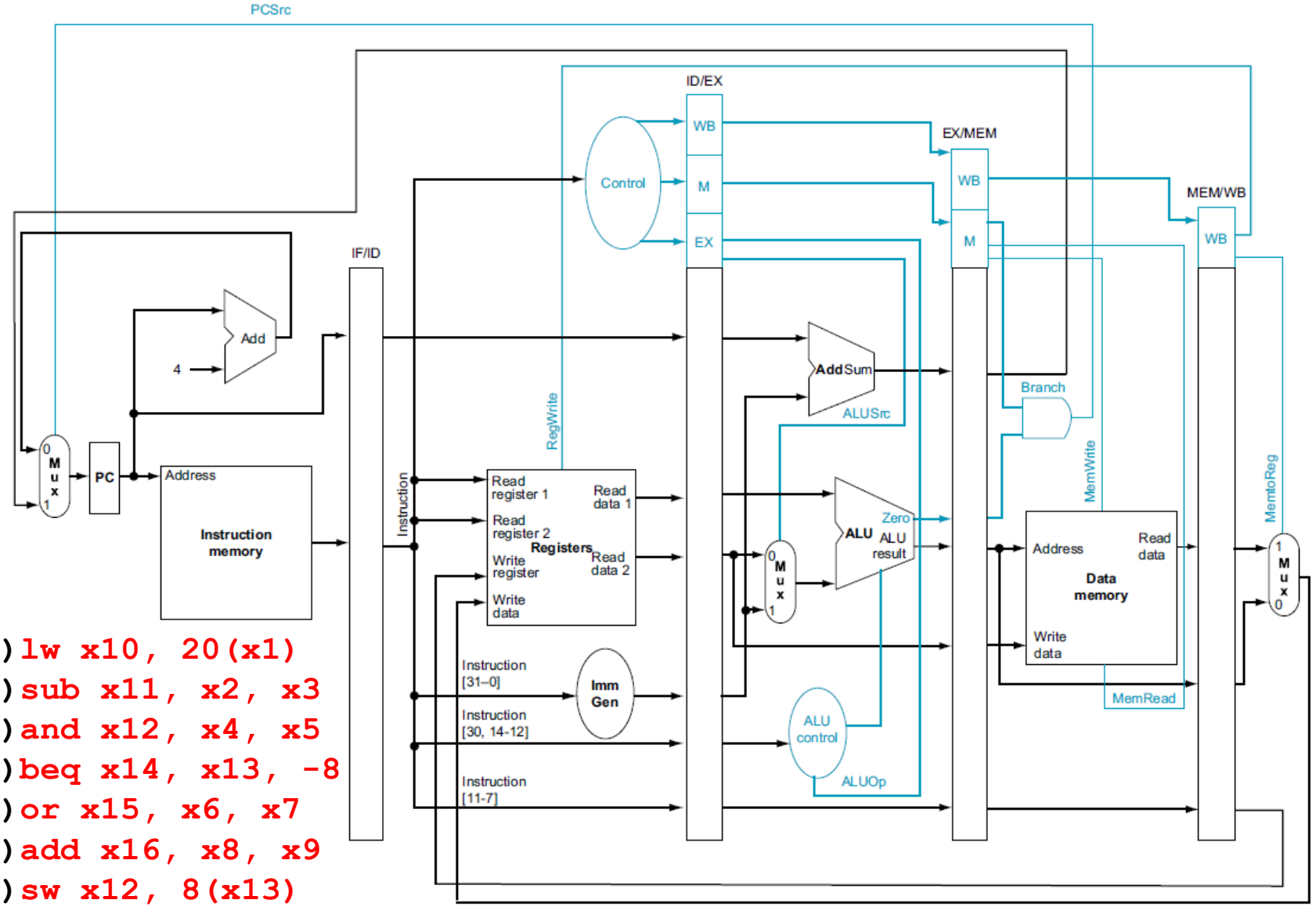
5º ciclo

Ex 2: Identifique as informações possíveis que estarão trafegando nos barramentos e nos sinais de controle da via de dados durante o quinto ciclo de clock.




```
(8000) lw x10, 20(x1)
(8004) sub x11, x2, x3
(8008) and x12, x4, x5
(8012) beq x14, x13, -8
(8016) or x15, x6, x7
(8020) add x16, x8, x9
```

Ex 3: Identifique quantos ciclos de clock são necessários para executar a seguinte sequencia de instruções.




```
(8000)lw x10, 20(x1)
(8004)sub x11, x2, x3
(8008)and x12, x4, x5
(8012)beq x14, x13, -8
(8016)or x15, x6, x7
(8020)add x16, x8, x9
(8024)sw x12, 8(x13)
(8028)sw x13, 8(x1)
```

Referências

- PATTERSON, David A; HENNESSY, John L; Computer Organization and Design – The hardware/software interface RISC-V edition; Elsevier – Morgan Kaufmann/ Amsterdam.
- PATTERSON, David; Waterman, Andrew; The RISC-V reader: an open architecture atlas; First edition. Berkeley, California: Strawberry Canyon LLC, 2017.