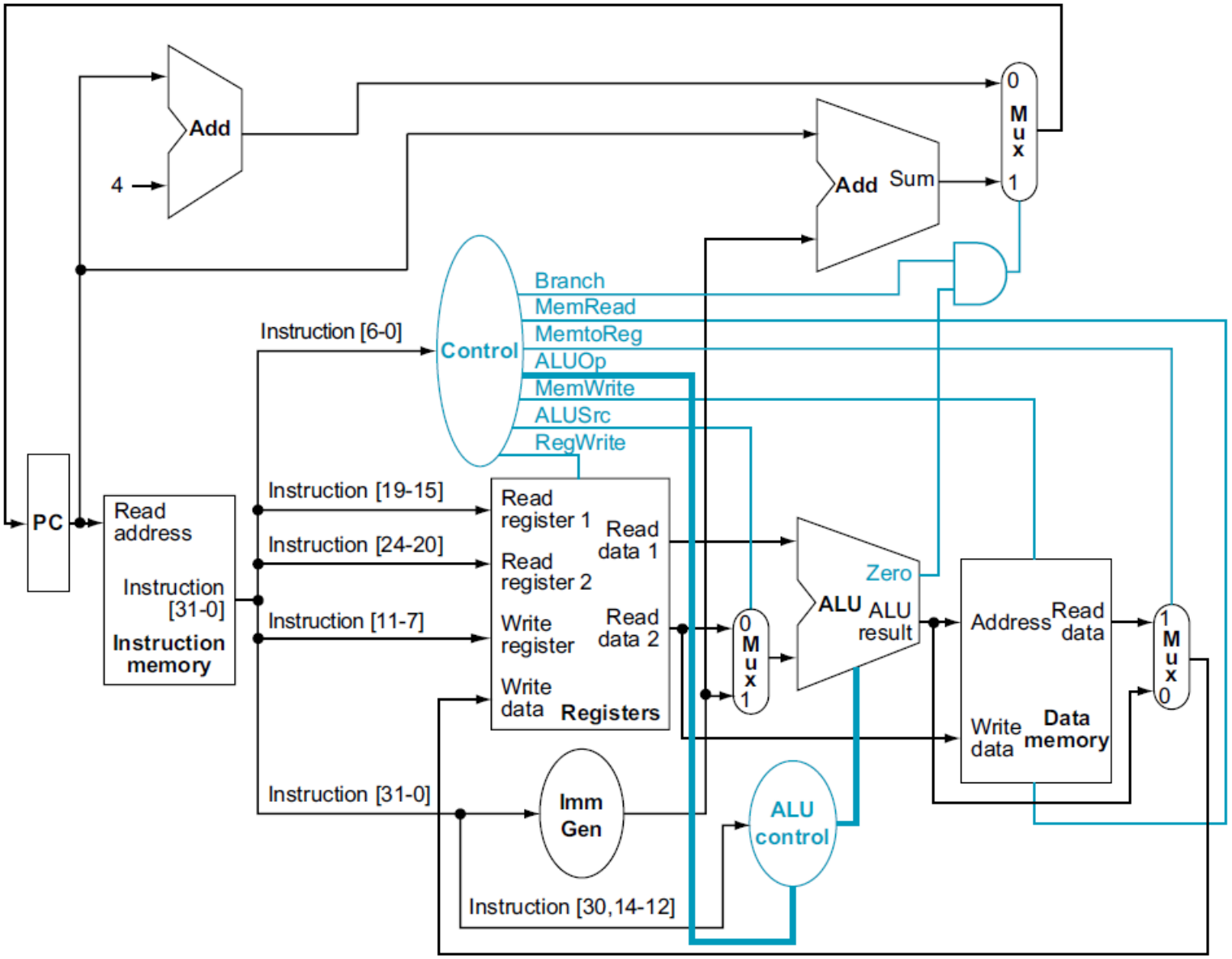
ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Arquitetura RISC-V **Multiciclo**

Profª. Fabiana F F Peres

Apoio: Camile Bordini

Lembrando: Via de Dados Monociclo



Problemas da Monociclo

- **Quais as devantagens da Monociclo?**
  - CPI = 1
  - Tempo de ciclo determinado pela instrução de maior duração (**load**)
  - Perda de área para hardware
    - Funcionalidades realizadas mais de uma vez no mesmo ciclo feitas em unidades computacionais diferentes
    - Ex: ULA, somador do PC+4, somador do PC+desvio

Via de Dados Multiciclo

Portanto, qual é a ideia da Multiciclo?

- Dividir as instruções em **ETAPAS**
  - Cada **etapa** leva um ciclo de clock
  - Necessário balancear a quantidade de trabalho realizada em cada etapa/ciclo para que levem o mesmo tempo
  - O tempo do ciclo é dado pela etapa mais lenta
  - Restringir cada etapa a usar no máximo uma grande unidade funcional para que elas não precisem ser duplicadas

Passos para execução das instruções

1. Buscar instrução da memória e incrementar PC;
2. Decodificar a instrução, ler registradores e calcular endereço de desvio condicional;
3. **Dependendo da instrução** que está sendo executada:
   - ***Load* ou *Store*:** calcular endereço da memória;
   - ***Tipo R*:** realizar a operação logica ou aritmética requisitada;
   - ***Beq*:** realizar a comparação entre o conteudo armazenado em dois registradores e realizar o desvio se a condição for aceita (se o conteúdo dos dois reristradores forem iguais);
4. **Dependendo da instrução** que está sendo executada:
   - ***Load*:** Lê o dado da memória;
   - ***Store*:** Escreve o dado na memória;
   - ***Tipo R*:** Escreve o resultado da operação no banco de reg.;
5. **Instrução** que está sendo executada:
   - ***Load*:** Escreve o dado lido no banco de registradores;

Via de Dados Multiciclo

Dividir as instruções em **ETAPAS:**

- **Cada um dos passos acima**, tomará um ciclo de clock
- Permite que as unidades funcionais possam ser usadas mais de uma vez pela mesma instrução contanto que seja em ciclos de clock diferentes
- Reduz o hardware necessário
- Permite que as instruções tomem diferentes números de ciclos de clock

Questão: quantos ciclos de clock as instruções **Load**, **Store**, **Tipo-R** e **Beq** necessitam?

Via de Dados Multiciclo

(diferenças entre a via de dados monociclo e a multiciclo)

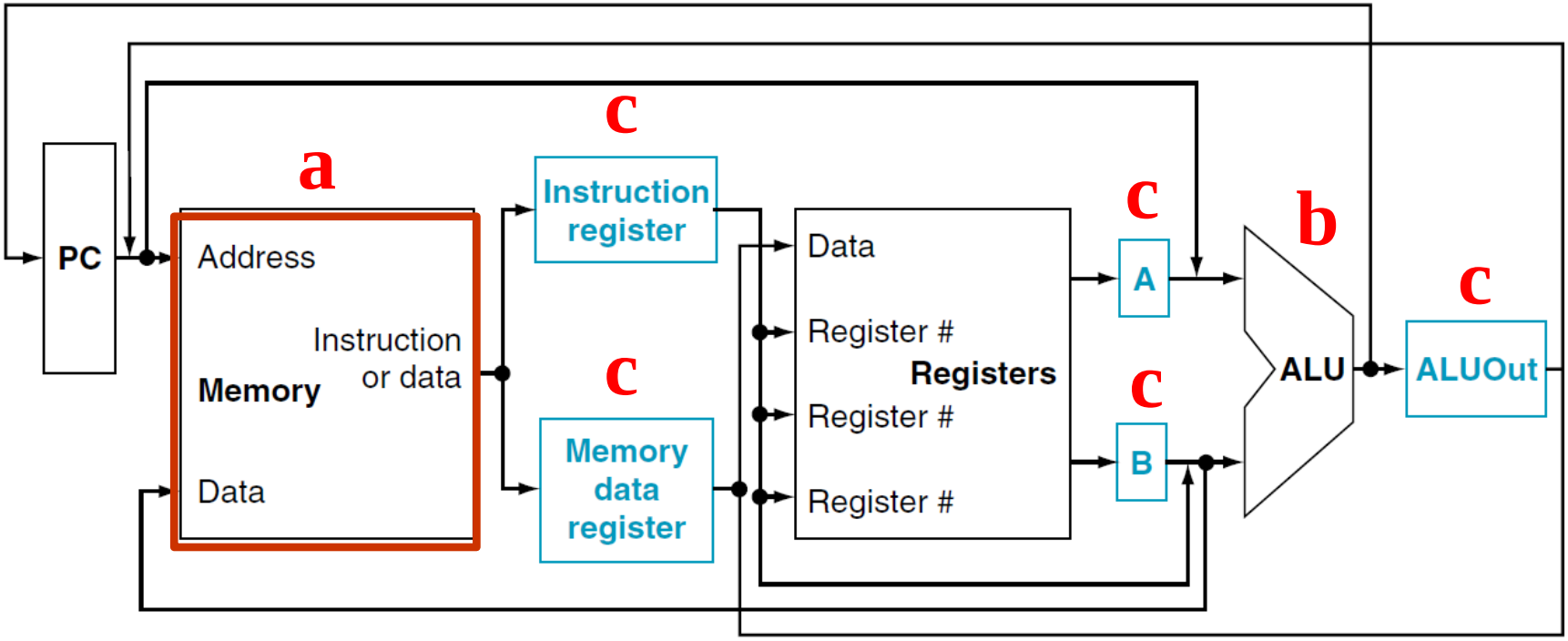
a. Unidade de **memória de instrução** e de **dados** é substituída por uma **única unidade de memória**;

b. A **ALU** e os **dois somadores** são substituídos por uma **única ALU**;

c. São adicionados alguns **registradores auxiliares** depois de algumas unidades funcionais com a função de guardar a saída daquela unidade para ser utilizado num ciclo de clock subseqüente, se necessário;

Via de Dados Multiciclo

(via de dados multiciclo)



Via de Dados Multiciclo

Entre as **ETAPAS:**

- Ao final de uma etapa: armazenar dados que serão usados na etapa seguinte de uma **mesma instrução**
  - **Registradores auxiliares**
- Dados usados em **instruções posteriores** são armazenados em elementos de estado visíveis (banco de registradores, PC, memória)

Via de Dados Multiciclo

(registradores adicionais)

• **IR (*Instruction Register*):** utilizado para guardar a saída da **memória** (quando a leitura é de uma **INTRUÇÃO**)

• **MDR (*Memory Data Register*):** utilizado para guardar a saída da **memória** (quando a leitura é de um **DADO**)

•**A e B:** utilizados para guardar o valor dos operandos lidos do banco de registradores (*Register File*)

•**ALUOut:** utilizado para guardar o resultado da operação realizada pela ALU

Questão: por que são necessários registradores adicionais?

Via de Dados Multiciclo

(mudanças nos multiplexadores)

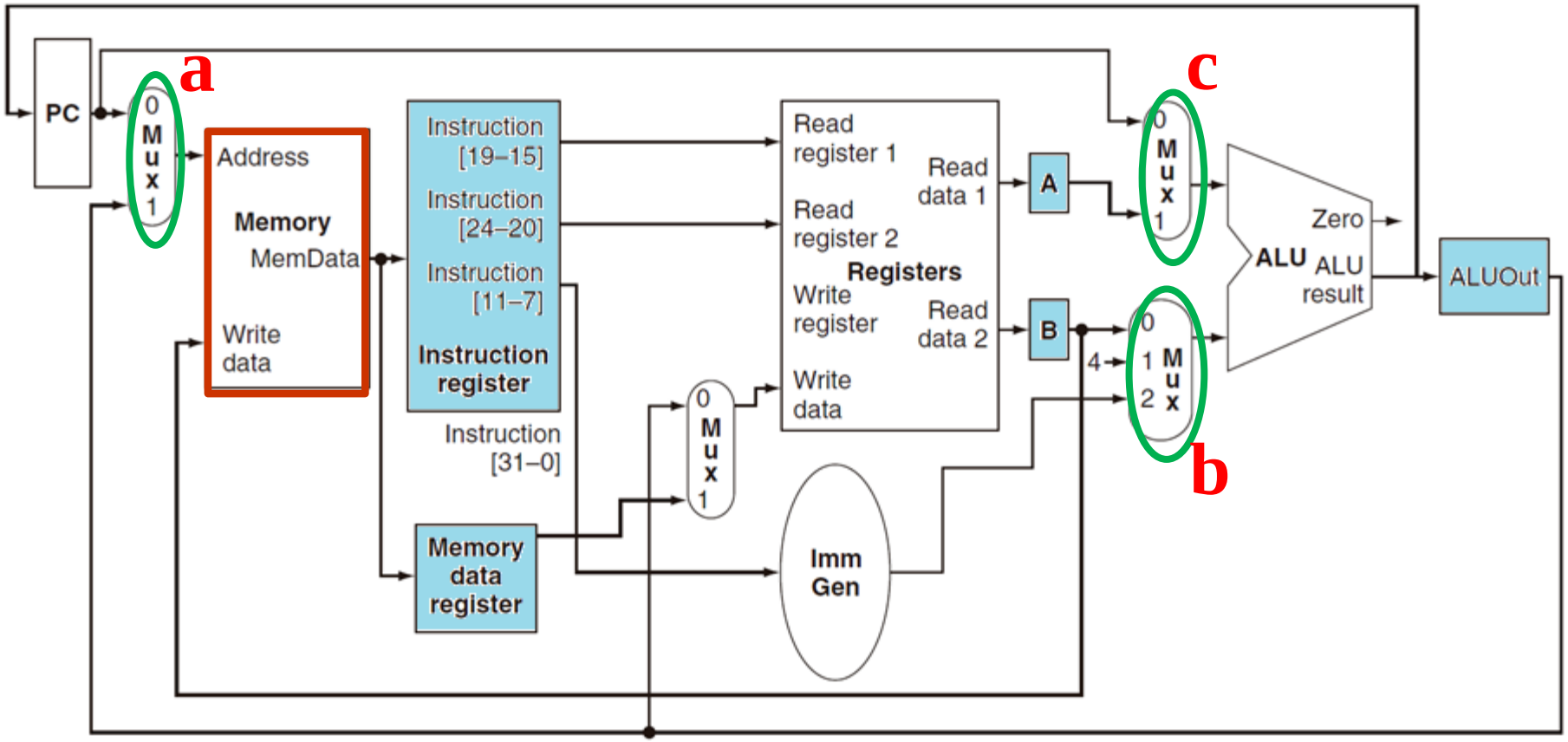
a. Precisamos de um **multiplexador** na **entrada da memória** para selecionar se o novo endereço que será acessado será: **PC+4** ou um outro endereço calculado pela ALU (**seguimento+offset**)

b. O **multiplexador** encontrado na segunda entrada da ALU deverá ser **expandido**, visto que agora esta será a única ALU e ela deverá acomodar todas as operações realizadas na via de dados

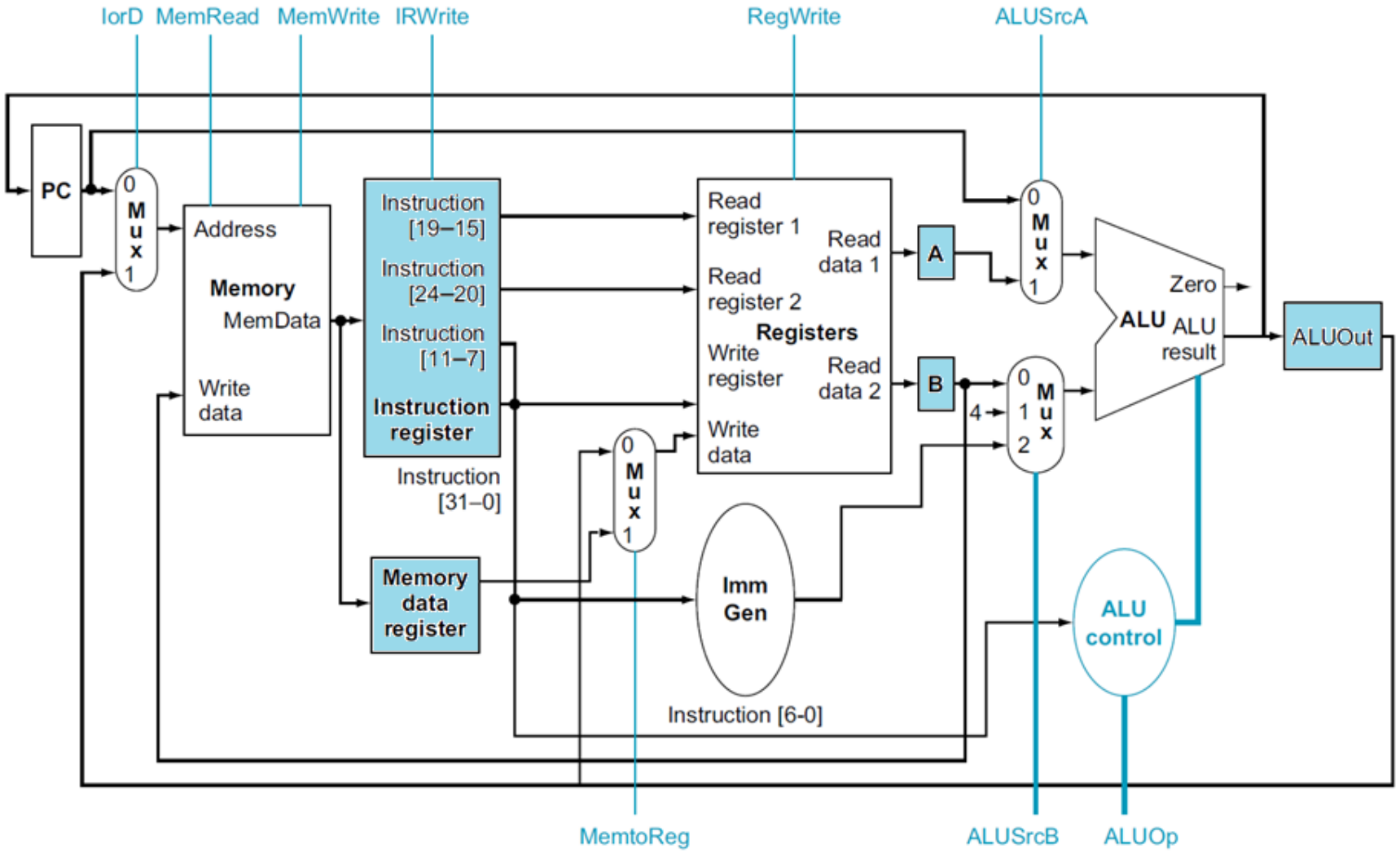
c. Um **novo multiplexador** deverá ser adicionado na primeira entrada da ALU

Via de Dados Multiciclo

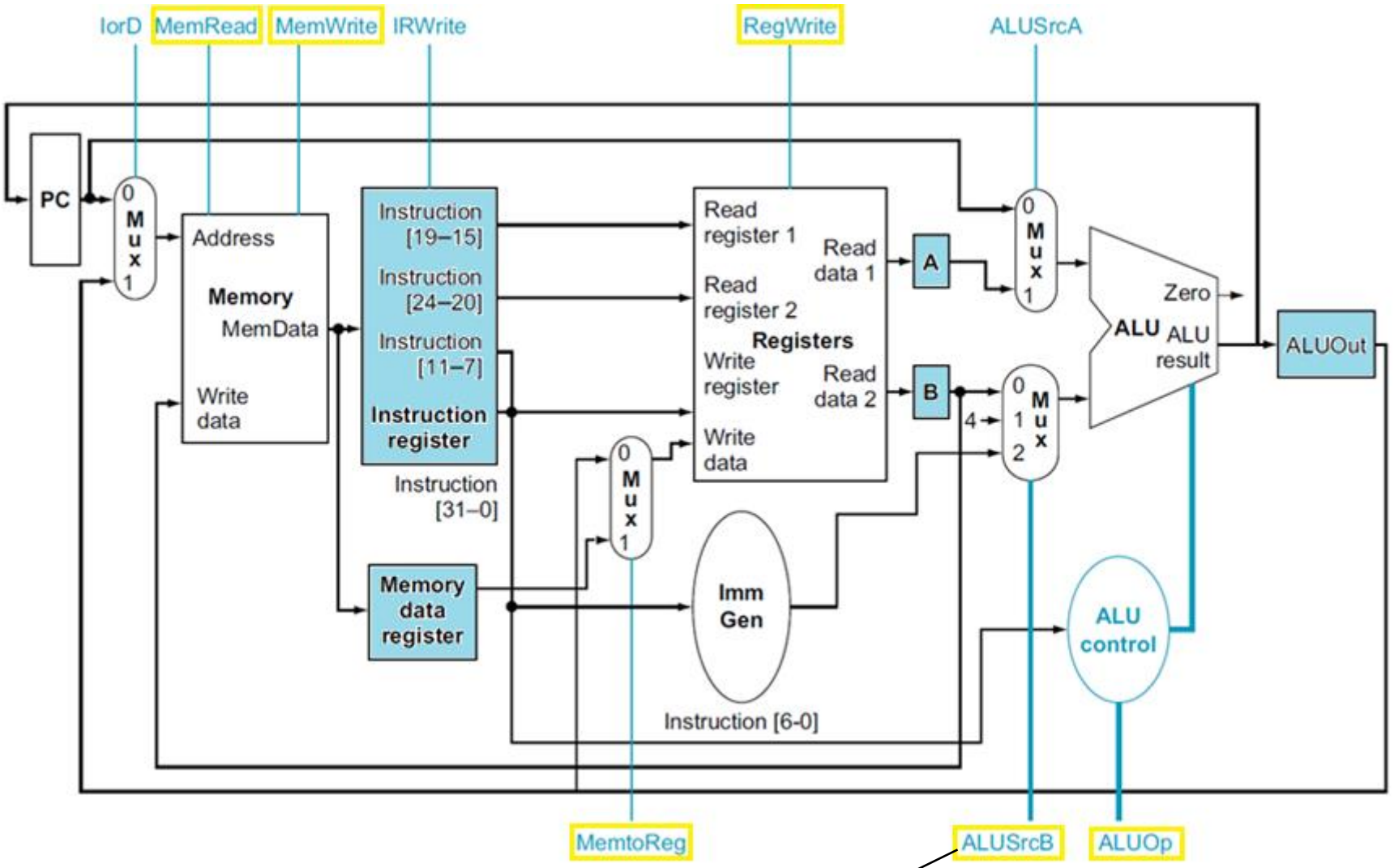
(via de dados multiciclo)



Sinais de controle – quais já existiam na Mono?

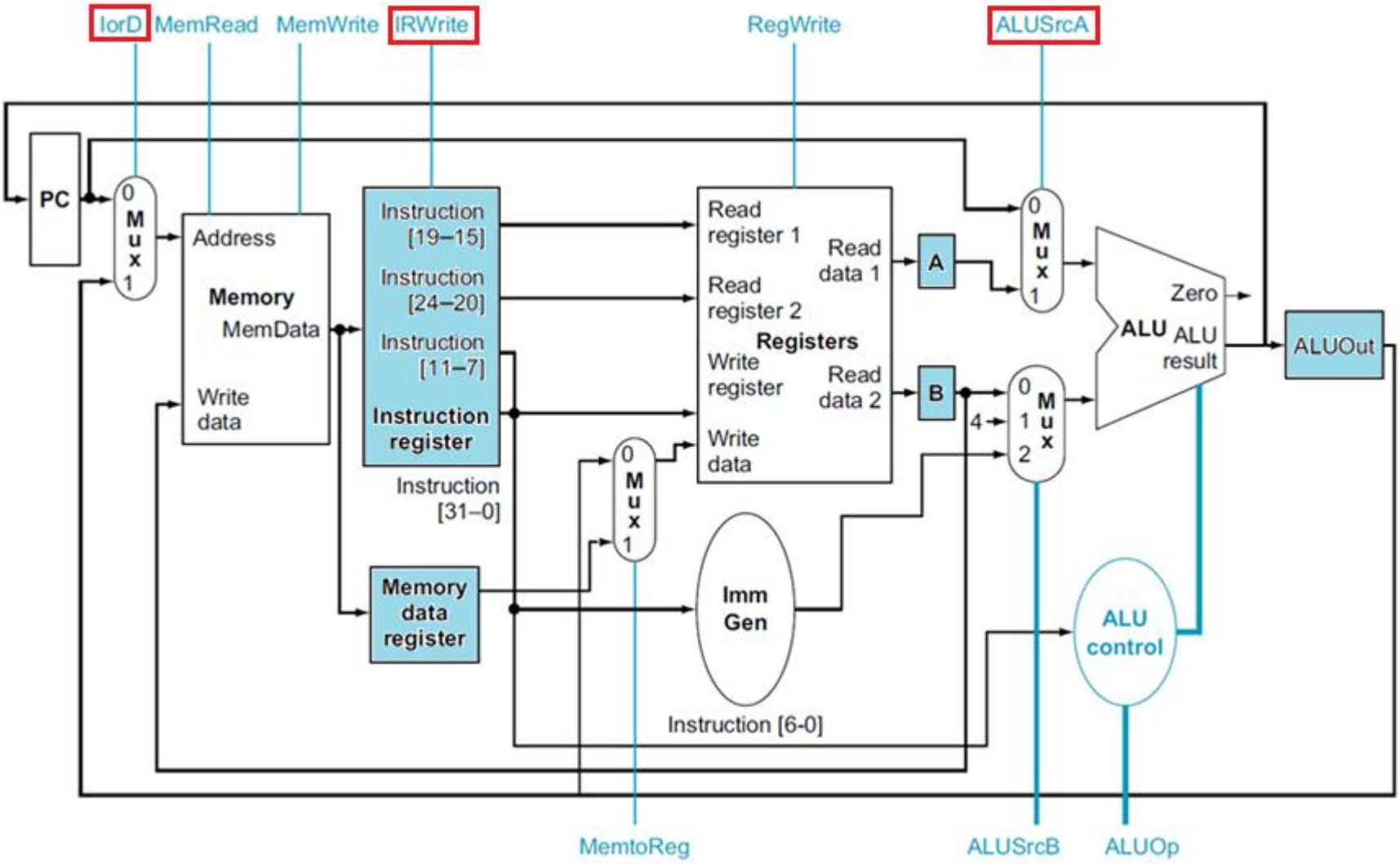


Sinais de controle – quais já existiam na Mono?



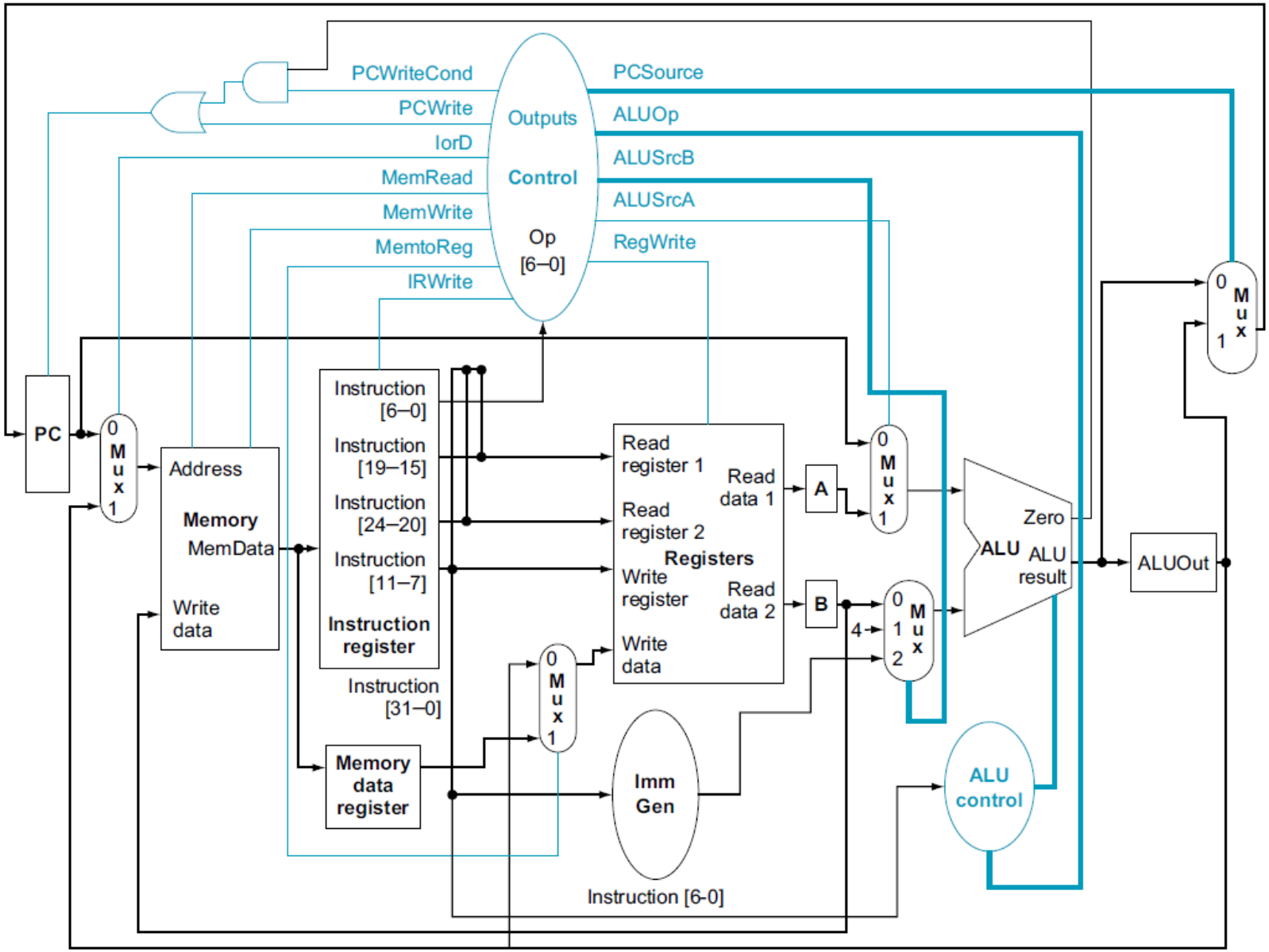
Mono: 1 bit Multi: 2 bits

Qual a necessidade dos novos sinais de controle?



Multiciclo

Via de Dados Multiciclo



Via de Dados Multiciclo

(características da execução das instruções na via de dados multiciclo)

• Balancear a quantidade de trabalho feito **em cada ciclo**, para minimizar o tempo de ciclo de clock

• **Cada passo** poderá conter no máximo: uma operação da ALU, um acesso ao *register file*, um acesso a memória

• Cada instrução necessita de **3** a **5 passos** para sua execução

Via de Dados Multiciclo

- Novo valor de PC será:
  – A saída da ALU (**PC+4**): quando a instrução não é uma condição ou a quando a condição não for aceita

  – O valor de **ALUOut** (**PC+offset**): quando a instrução for um desvio condicional aceito

- Sinais de controle:
  - Dois sinais de controle : **PCWrite** e **PCWriteCond**

Sinais de Controle

• **IorD**: como há uma única memória (dados e instruções), é necessário, a cada ciclo, selecionar se o endereço a ser lido (ou escrito) é o que **PC** armazena ou o **calculado pela ALU**;

• **IRWrite**: para o controle de escrita no IR, que deve se manter consistente até o final de uma instrução nos demais ciclos de clock. IR só deve ser sobrescrito ao ser lido uma nova instrução;

• **ALUSrcA**: como não há mais somadores adicionais, é a ALU quem fará todas as somas, portanto, o primeiro operando da ALU pode ser ou o valor de **PC** ou o do registrador **rs1**

• **ALUSrcB**: o segundo operando da ALU agora pode ser o **registrador rs2**, a **constante 4** ou a constante da instrução (**imediato** – beq, lw, sw)

Sinais de Controle

Nome do sinal	Efeito quando desativado	Efeito quando ativado
RegWrite	Nenhum	É escrito em um registrador (*Write register)* o valor existente em *Write date*
ALUSrcA	O primeiro operando da ALU é o PC	O primeiro operando da ALU vem do registrador **A**
MemRead	Nenhum	O conteúdo da **Memória** armazenado no endereço de entrada (*Address*) é colocado na saída (*Read data*)
MemWrite	Nenhum	O conteúdo da **Memória** armazenado no endereço de entrada (*Address*) é substituído pelo dado na entrada (*Write data*)
MemtoReg	O valor que será escrito no no **registrador destino** vem do **ALUOut**	O valor que será escrito no **registrador destino** vem da **Memória**
IorD	O valor de PC é usado para indicar o endereço da **memória**	O **ALUOut** é usado para indicar o endereço da **memória**;
IRWrite	Nenhum	A saída da memória é escrita no IR

Sinais de Controle

Nome do sinal	Efeito quando é 0	Efeito quando é 1
PCWrite	None	O PC é escrito; O fonte é controlado pelo PCSource;
PCWriteCond	None	O PC é escrito se a saída do *Zero* da ALU é também ativa (1);

Nome do sinal	Valor	Efeito
ALUOp	00	A ALU realiza uma soma;
	01	A ALU realiza uma subtração;
	10	O campo *funct3* e *funct7* da instrução é que determina a operação da ALU;
ALUSrcB	00	A segunda entrada da ALU vem do **registrador B**;
	01	A segunda entrada da ALU é a **constante 4**;
	10	A segunda entrada da ALU é o ***immediate generated*** da instrução;
PCSource	0	A saída da ALU (**PC+4**) é enviada para escrita no PC;
	1	O conteúdo de **ALUOut** é enviado para escrita no PC (**endereço de desvio**);

Caminho de dados por etapa

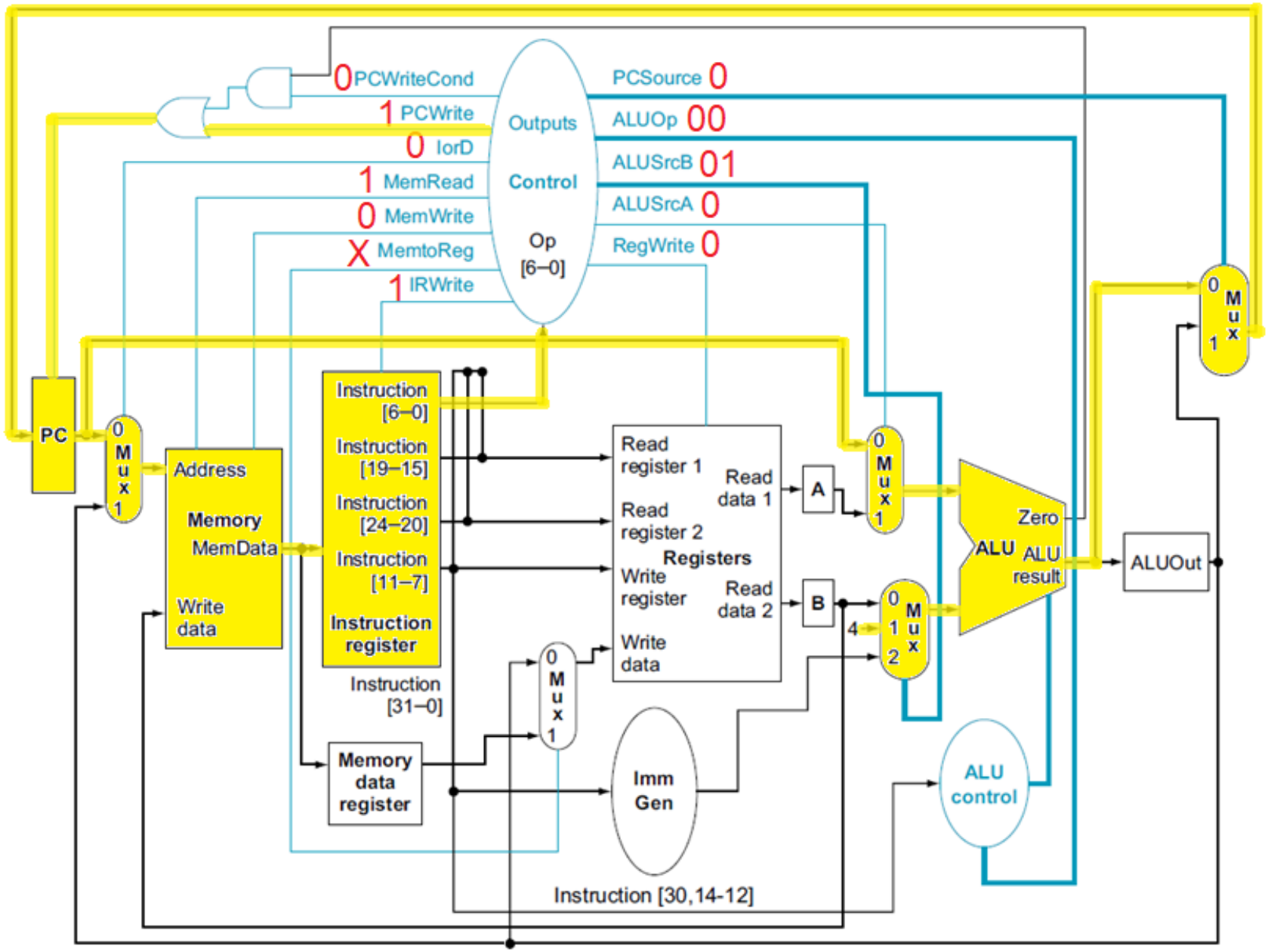
Passo 1

1. **Buscar instrução da memória e incrementar PC;**

IR = Memory[PC]
PC = PC + 4

- Operações:
  - Envia o PC para a memória com o endereço;
  - Realiza a leitura do endereço de memória especificado;
  - Escreve a instrução dentro do IR;
  - Incrementa o PC em 4.

Passo 1


0 PCWriteCond
1 PCWrite
0 IorD
1 MemRead
0 MemWrite
X MemtoReg
1 IRWrite
Outputs
Control
Op
[6–0]
PCSource 0
ALUOp 00
ALUSrcB 01
ALUSrcA 0
RegWrite 0
PC
0 M u x 1
Address
Memory
MemData
Write
data
Instruction
[6–0]
Instruction
[19–15]
Instruction
[24–20]
Instruction
[11–7]
Instruction
register
Instruction
[31–0]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
0 1 2 M u x
ALU
Zero
ALU
result
ALUOut
Imm
Gen
Instruction [30,14-12]
ALU
control

Passo 2

2. Decodificar a instrução,
ler registradores (adiantando, caso a instrução use os campos), e
calcular endereço de desvio condicional (adiantando, caso seja um desvio)

A = Reg[IR[19-15]] // registrador **A** recebe o conteúdo do registrador do *register file* contido no endereço especificado na posição [19-15] da instrução;

B = Reg[IR[24-20]] // registrador **B** recebe o conteúdo do registrador do *register file* contido no endereço especificado na posição [24-20] da instrução;

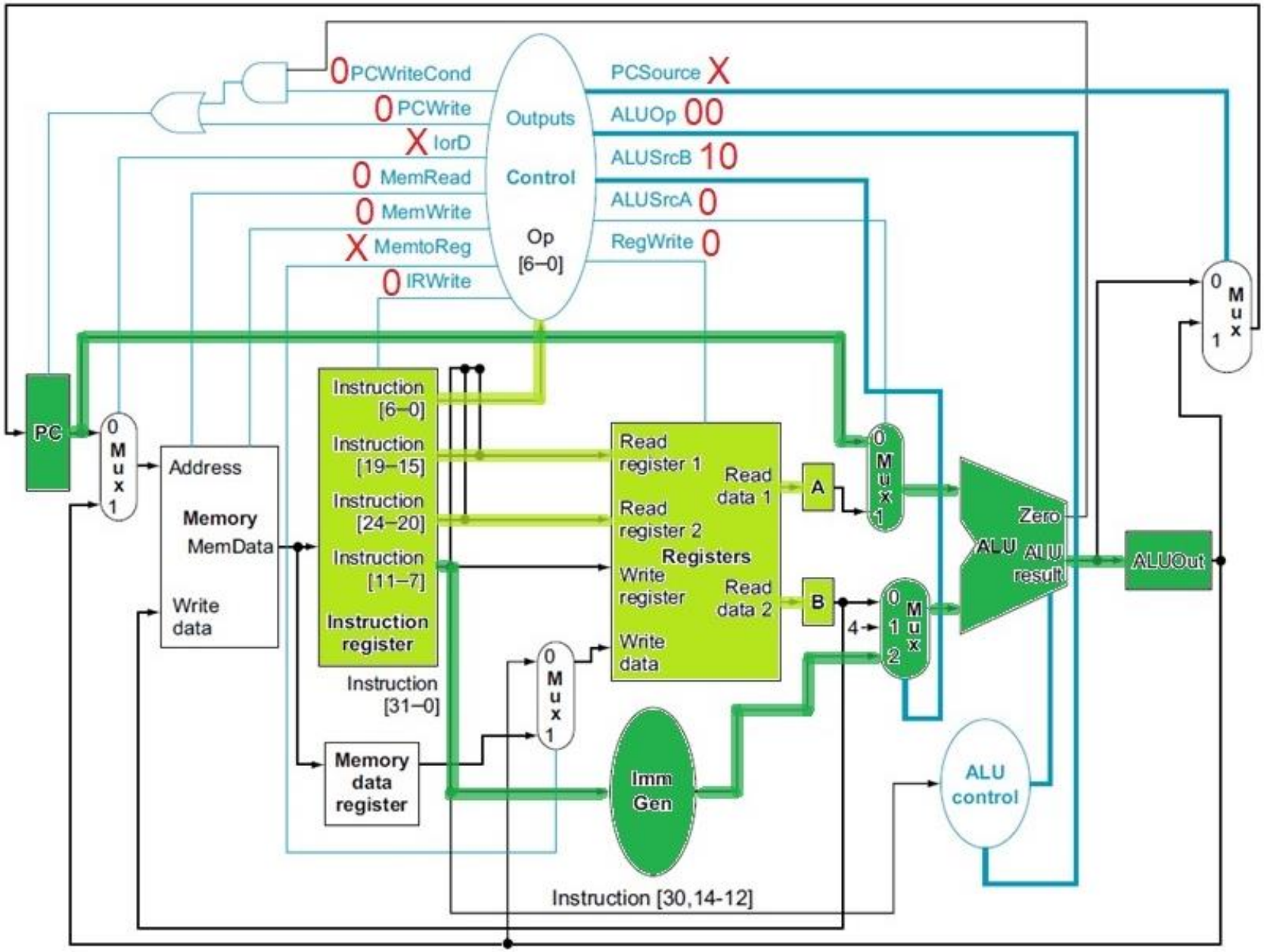
ALUOut = PC + *immediate*

Passo 2

2. Decodificar a instrução,
ler registradores (adiantando, caso a instrução use os campos), e
calcular endereço de desvio condicional (adiantando, caso seja um desvio)

- Operações:
  - Acessa o *register file* para leitura dos registradores **rs1** e **rs2**;
  - Armazena o resultado dentro dos registradores **A** e **B**;
  - Calcula o **endereço de desvio**;
  - Armazena o endereço calculado dentro do **ALUOut**;

Passo 2


0 PCWriteCond
0 PCWrite
X IorD
0 MemRead
0 MemWrite
X MemtoReg
0 IRWrite
Outputs
Control
Op
[6–0]
PCSource X
ALUOp 00
ALUSrcB 10
ALUSrcA 0
RegWrite 0
PC
0 M u x 1
Address
Memory
MemData
Write
data
Instruction
[6–0]
Instruction
[19–15]
Instruction
[24–20]
Instruction
[11–7]
Instruction
register
Instruction
[31–0]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
0 1 2 M u x
0 M u x 1
Imm
Gen
ALU
control
Zero
ALU
ALU
result
ALUOut
0 M u x 1
Instruction [30,14-12]

Passo 3

3. **Dependendo da instrução** que está sendo executada:
   - ***Load* ou *Store*:** calcular endereço da memória;
   - ***Tipo R*:** realizar a operação logica ou aritmética requisitada;
   - ***Beq*:** *término da instrução:* realizar a comparação entre o conteúdo armazenado em dois registradores e realizar o desvio se a condição for aceita (se o conteúdo dos dois reristradores forem iguais);

Passo 3

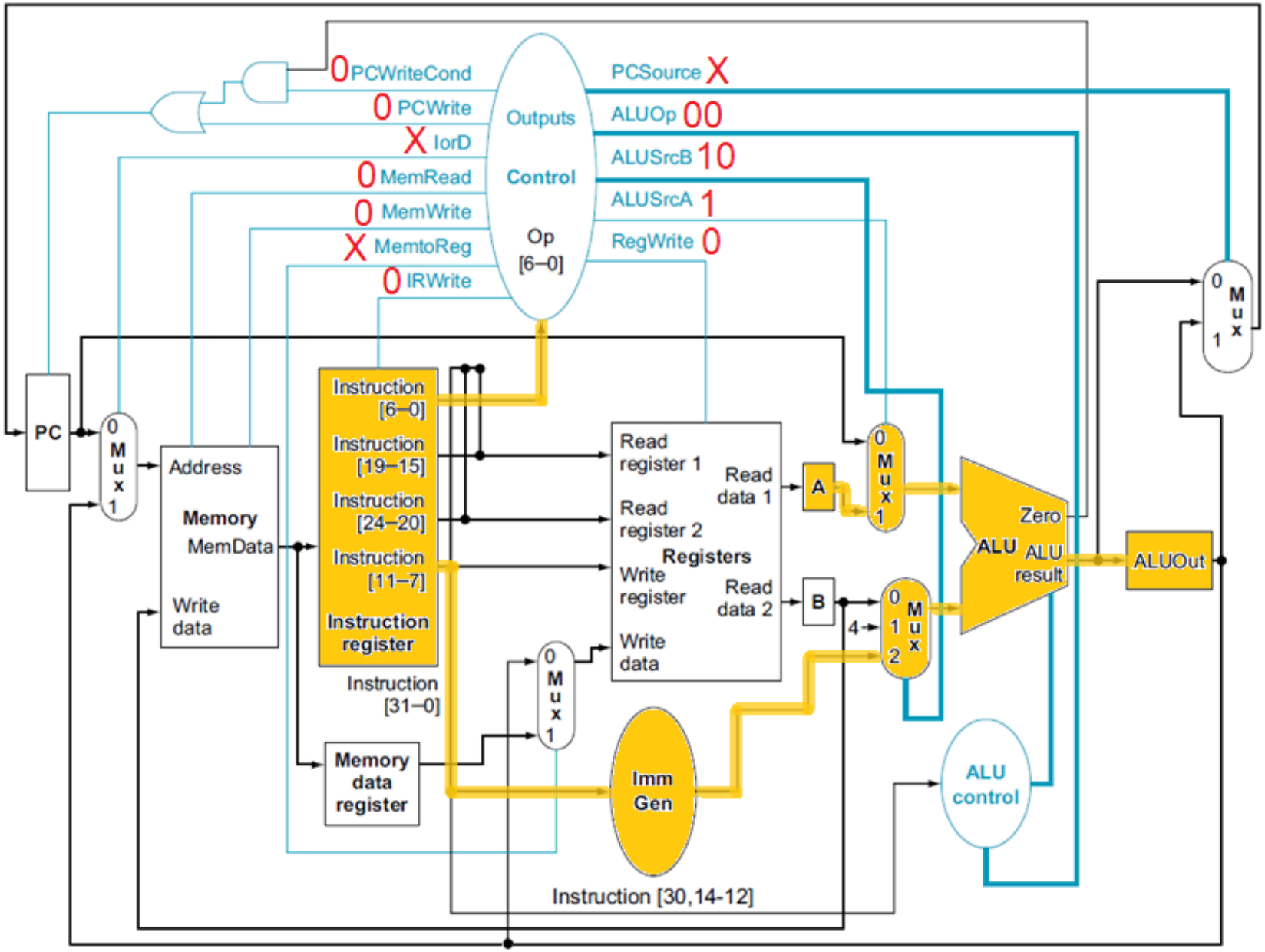
3. **Dependendo da instrução** que está sendo executada:

- Se: ***Load* ou *Store*:** calcular endereço da memória;

ALUOut = A + *immediate*

- Operações:
  - A ALU soma o conteúdo do regsitrador **A** com o *offset* com sinal estendido para calcular o endereço de memória;

Passo 3 – Load ou Store



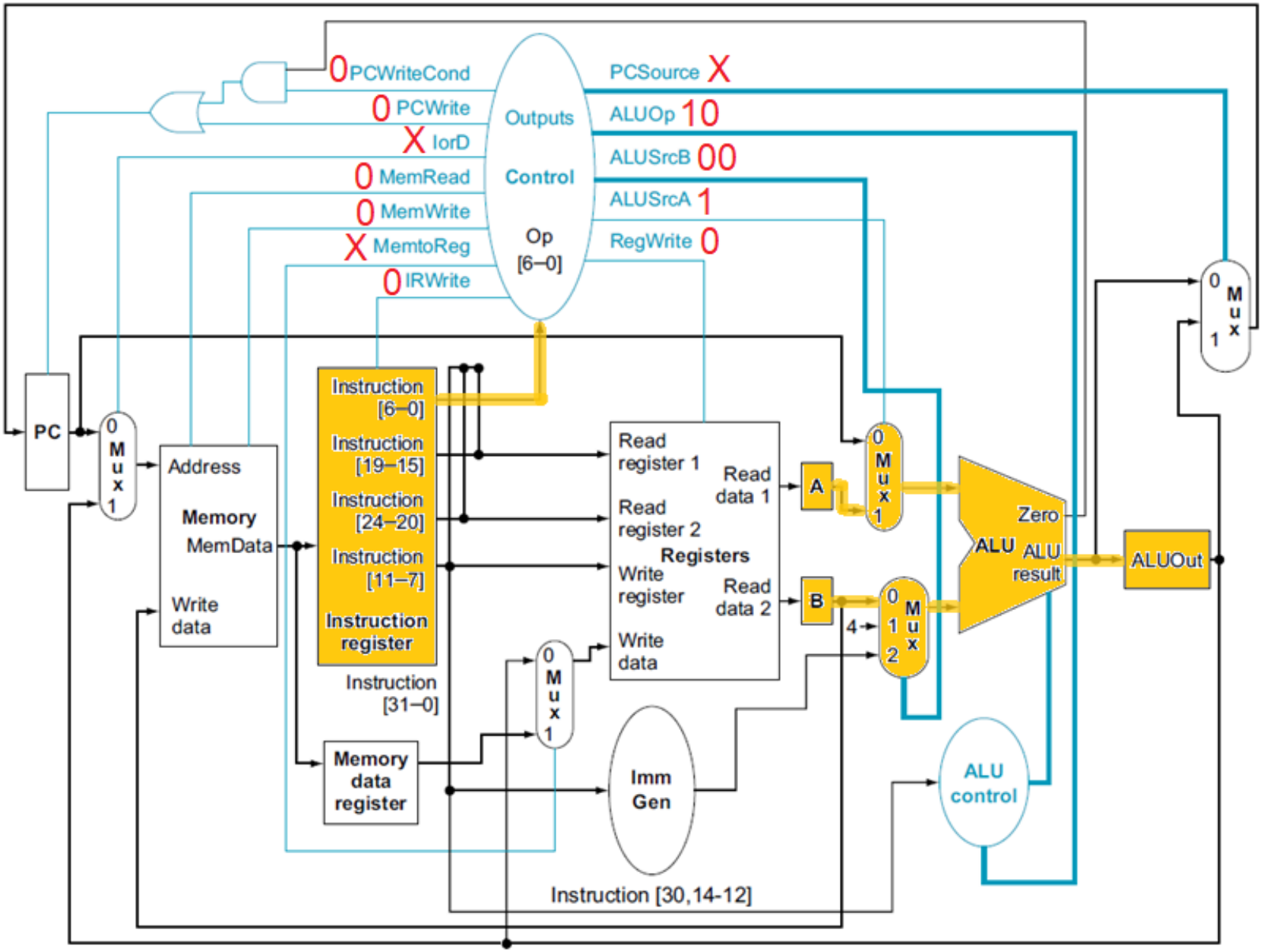
Passo 3

- Se: ***Tipo R*:** realizar a operação logica ou aritmética requisitada;

ALUOut = A op B

  - Operações:
    – A ALU realiza a operação especificada pelo codigo da função com os dois valores lidos do *register file*;

Passo 3 – Tipo-R



Passo 3

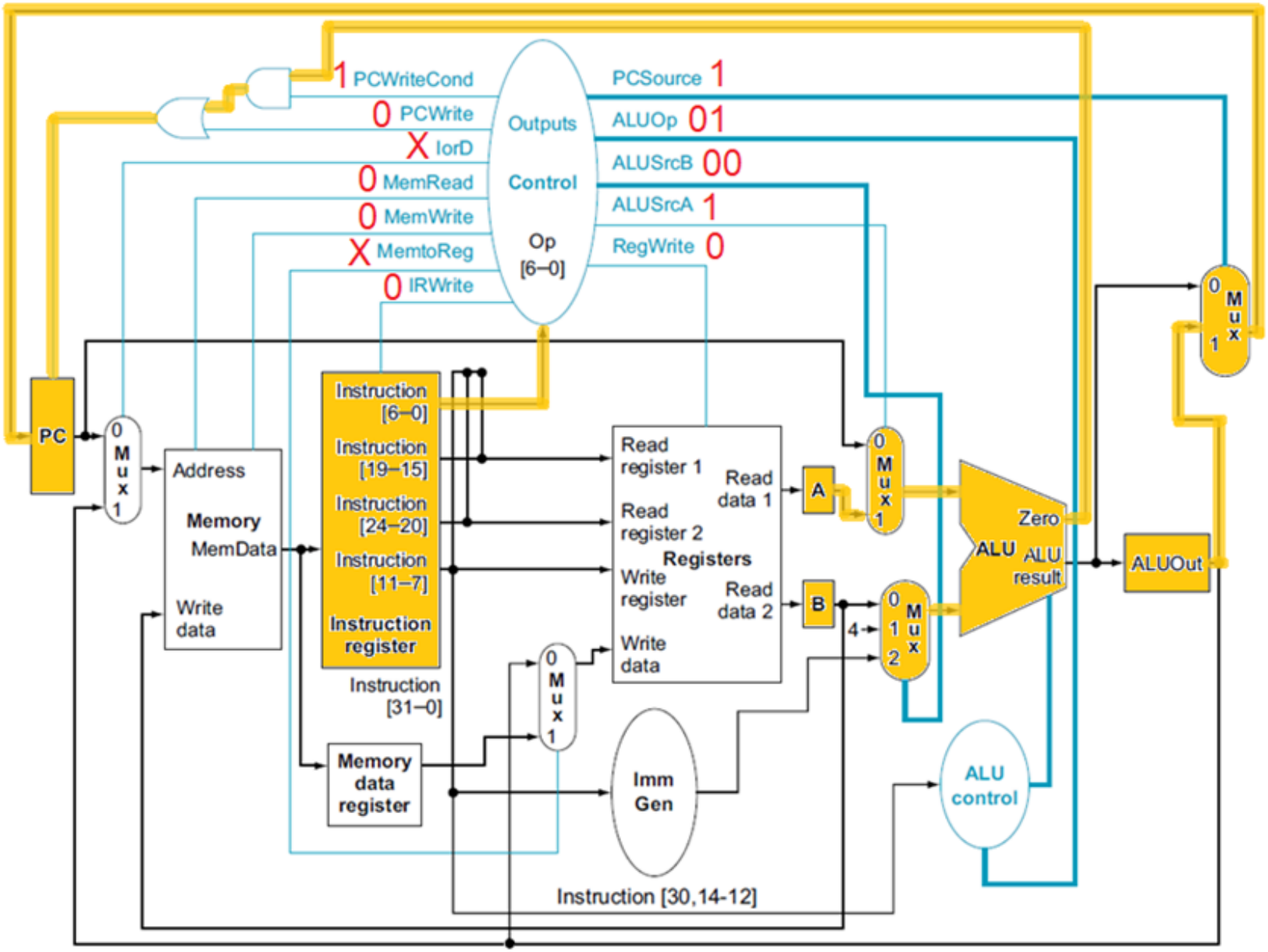
- Se: ***Beq:*** *término da instrução:* realizar a comparação entre o conteúdo armazenado em dois registradores e realizar o desvio se a condição for aceita (se o conteúdo dos dois reristradores forem iguais);

if(A==B) PC = ALUOut

Operações:

- A ALU realiza uma comparação entre o valor do registrador **A** e **B**;
- O sinal ***Zero*** da ALU é utilizado para determinar se o desvio irá ou não acontecer;

Passo 3 – Beq (finalizada)



Passo 4

4. **Dependendo da instrução** que está sendo executada:
   - ***Load*:** Lê o dado da memória;

MDR = Memory[ALUOut]

MDR irá receber o conteúdo da memória na posição seguimento + offset

   - ***Store*:** Escreve o dado na memória;

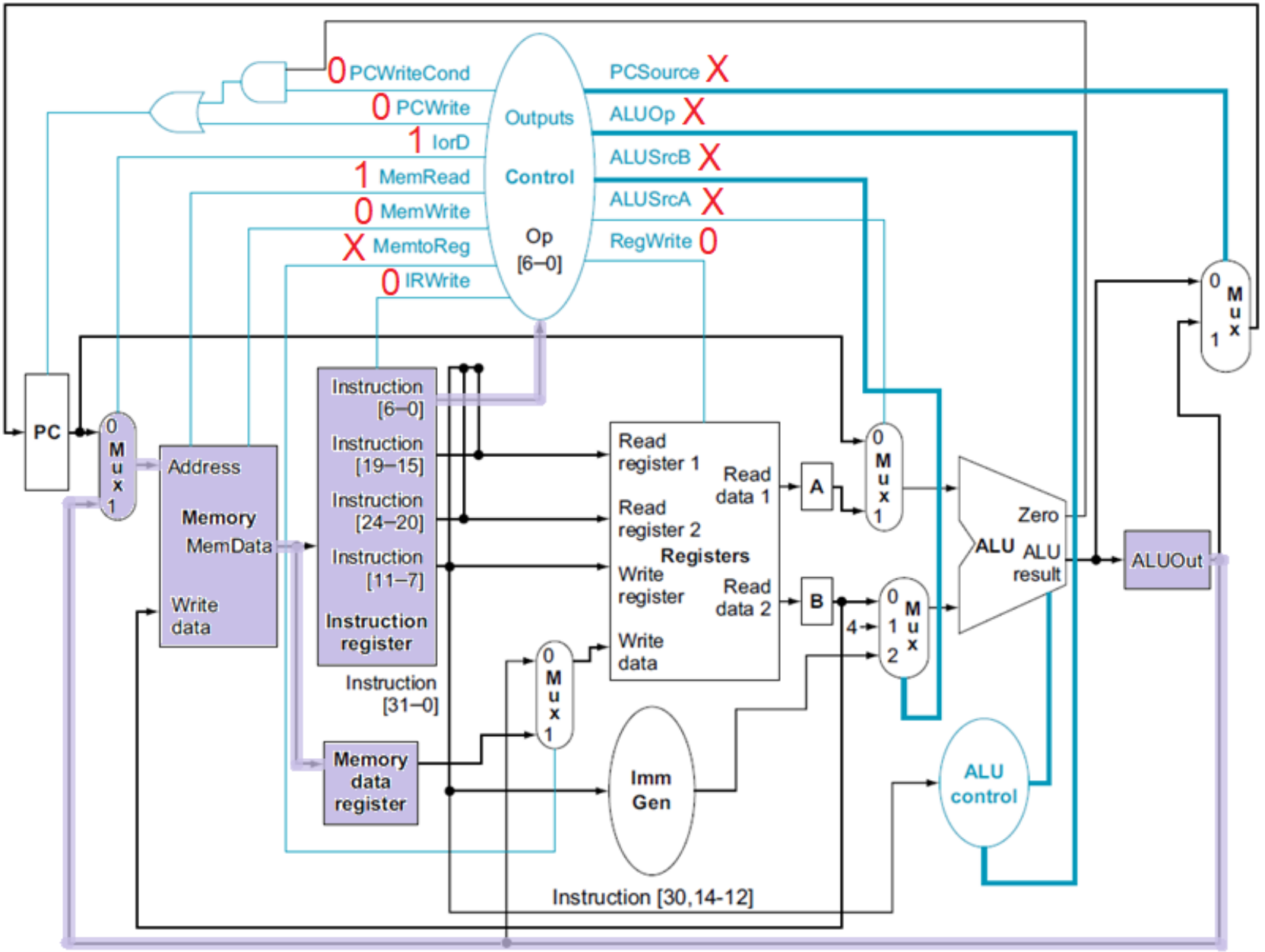
Memory[ALUOut] = B

A posição seguimento + offset da memória irá receber o conteúdo do registrador B

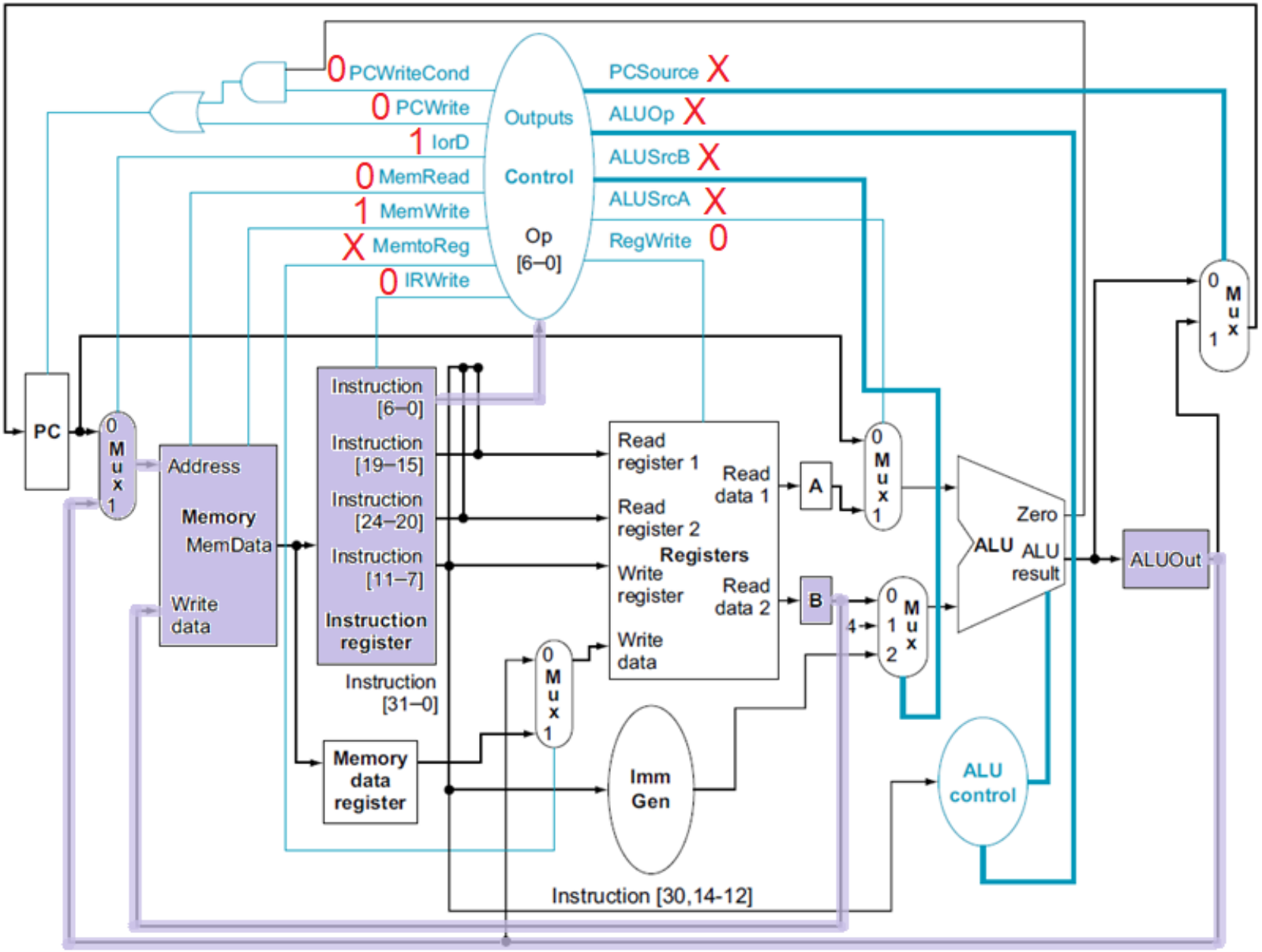
   - ***Tipo R*:** Escreve o resultado da operação no banco de reg.;

Reg[IR[11-7]] = ALUOut

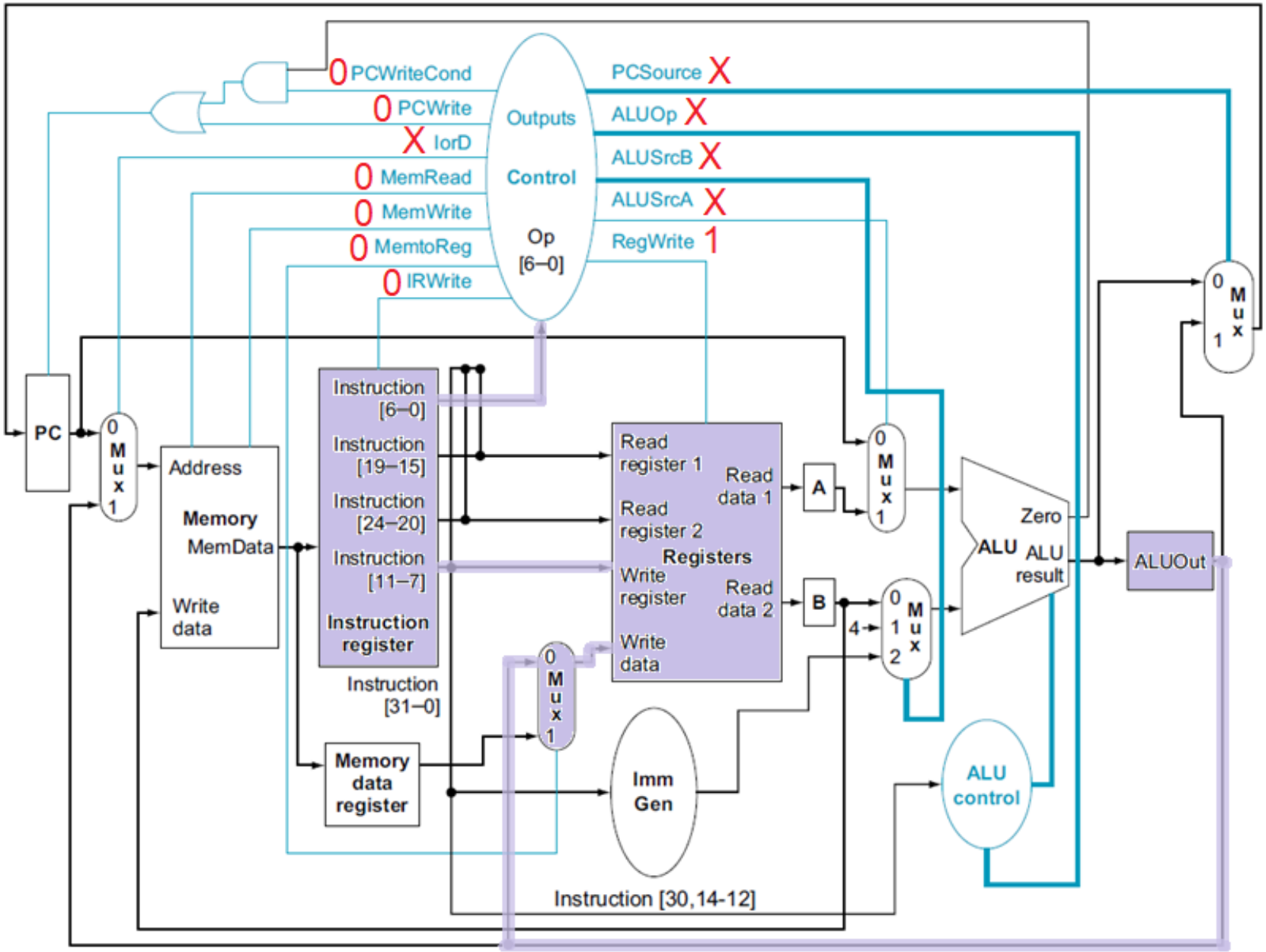
Passo 4 – Load


0 PCWriteCond
0 PCWrite
1 IorD
1 MemRead
0 MemWrite
X MemtoReg
0 IRWrite
Outputs
Control
Op
[6–0]
PCSource X
ALUOp X
ALUSrcB X
ALUSrcA X
RegWrite 0
PC
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction
[6–0]
Instruction
[19–15]
Instruction
[24–20]
Instruction
[11–7]
Instruction
register
Instruction
[31–0]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
ALU
Zero
ALU
result
ALUOut
Imm
Gen
ALU
control
Instruction [30,14-12]

Passo 4 – Store (finalizada)



Passo 4 – Tipo-R (finalizada)



Passo 5

5. **Instrução** que está sendo executada:
   - ***Load*:** Escreve o dado lido no banco de registradores

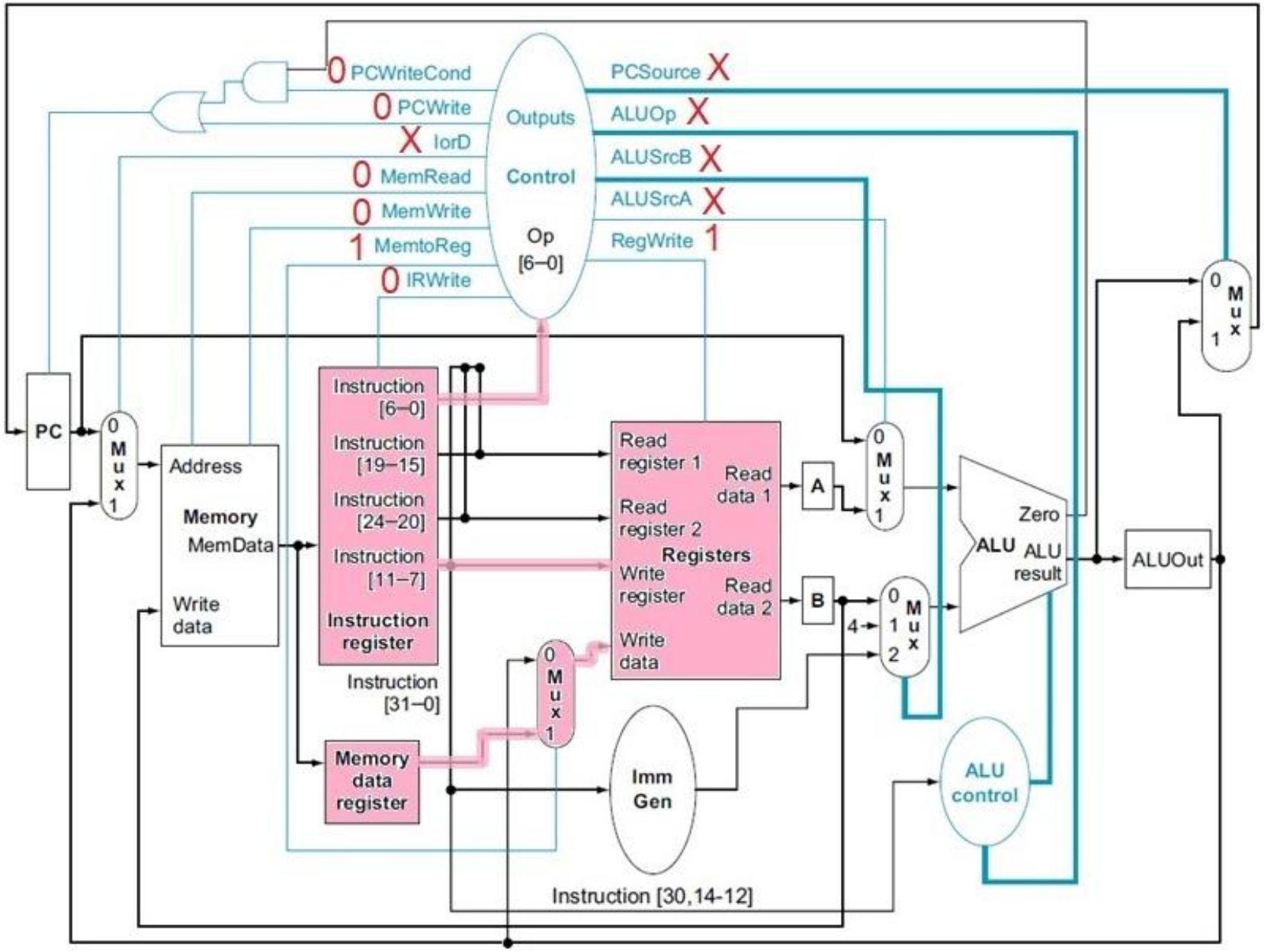
Reg[IR[11-7]] = MDR

- *Obs*: não há impedimento no caminho de dados (ou de controle) para que esse Passo 5 fosse eliminado e suas ações fossem transferidas para o Passo 4.

(IR[11-7] = Memory[ALUOut])

- Mas não é feito dessa forma pois: o acesso à memória é a operação mais lenta, portanto o Passo 4 é o que determina o tamanho do ciclo. Se fosse dessa forma, aumentaríamos o tamanho do ciclo para todos os passos!

Passo 5 – Load (finalizada)

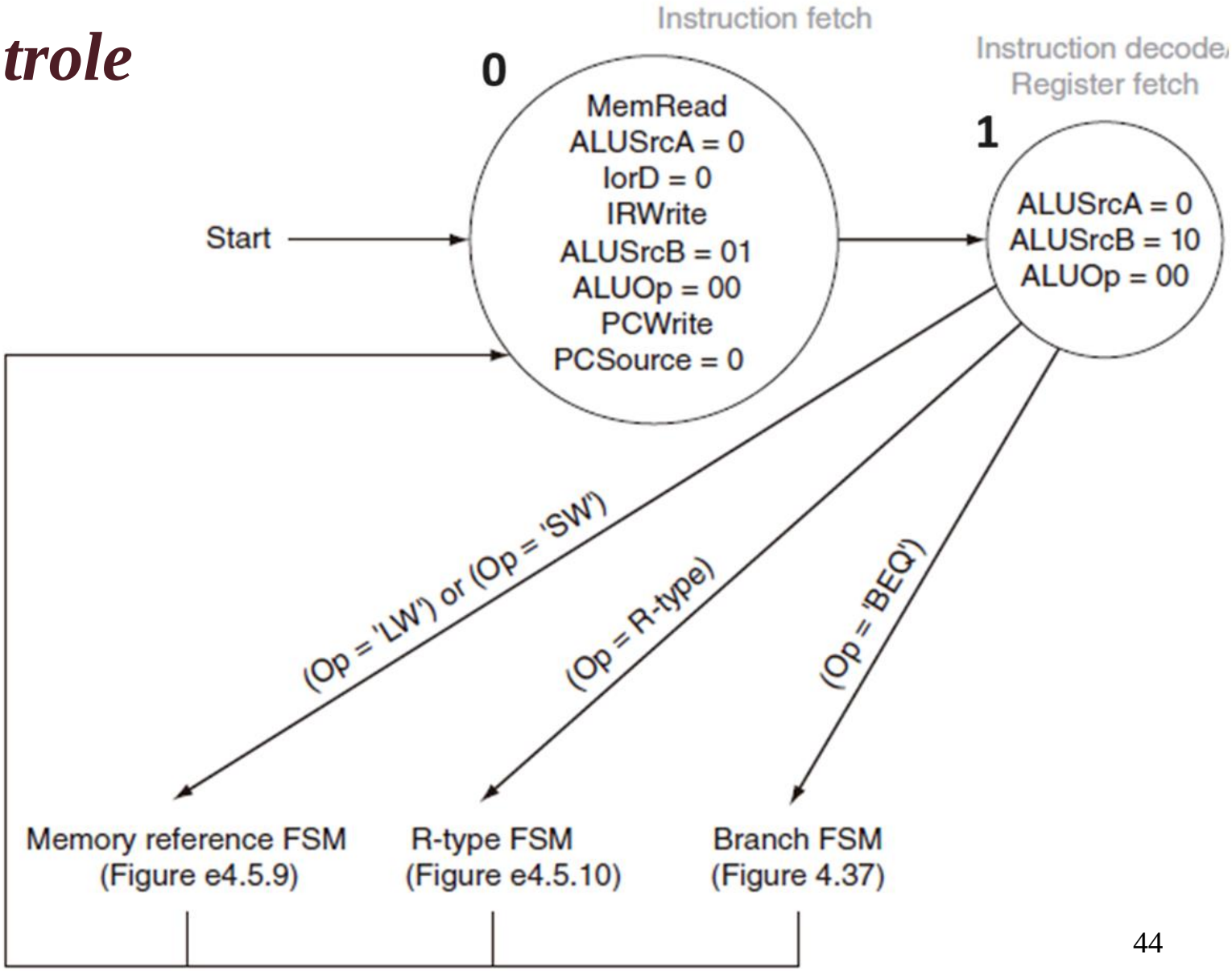


Em Resumo

Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches
Instruction fetch	IR <= Memory[PC] PC <= PC + 4		
Instruction decode/register fetch	A<= Reg [IR[19:15]] B <= Reg [IR[24:20]] ALUOut <= PC + immediate		
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + immediate	if (A == B) PC <= ALUOut
Memory access or R-type completion	Reg [IR[11:7]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B	
Memory read completion		Load: Reg[IR[11:7]] <= MDR	

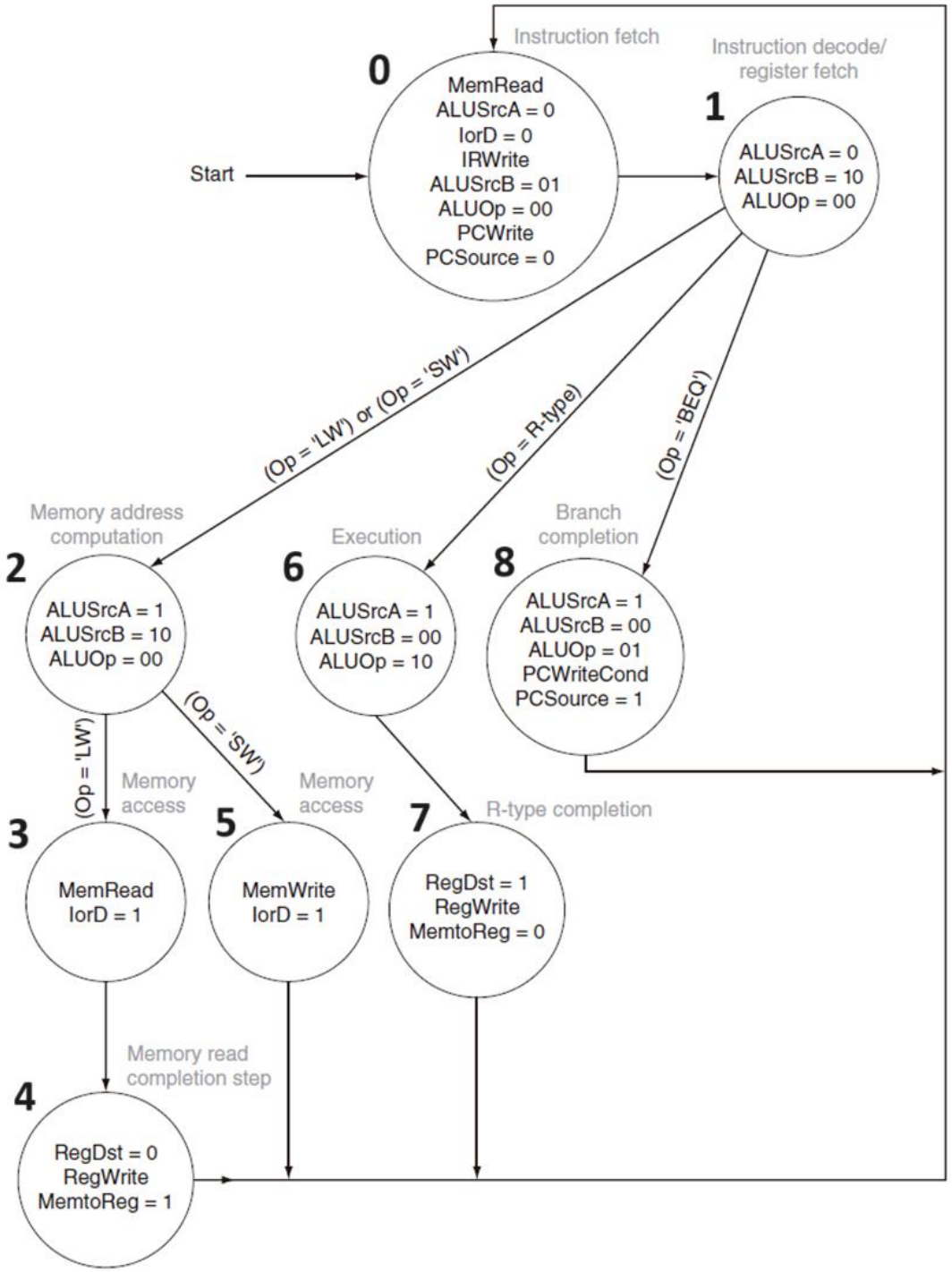
Unidade de Controle

Unidade de controle

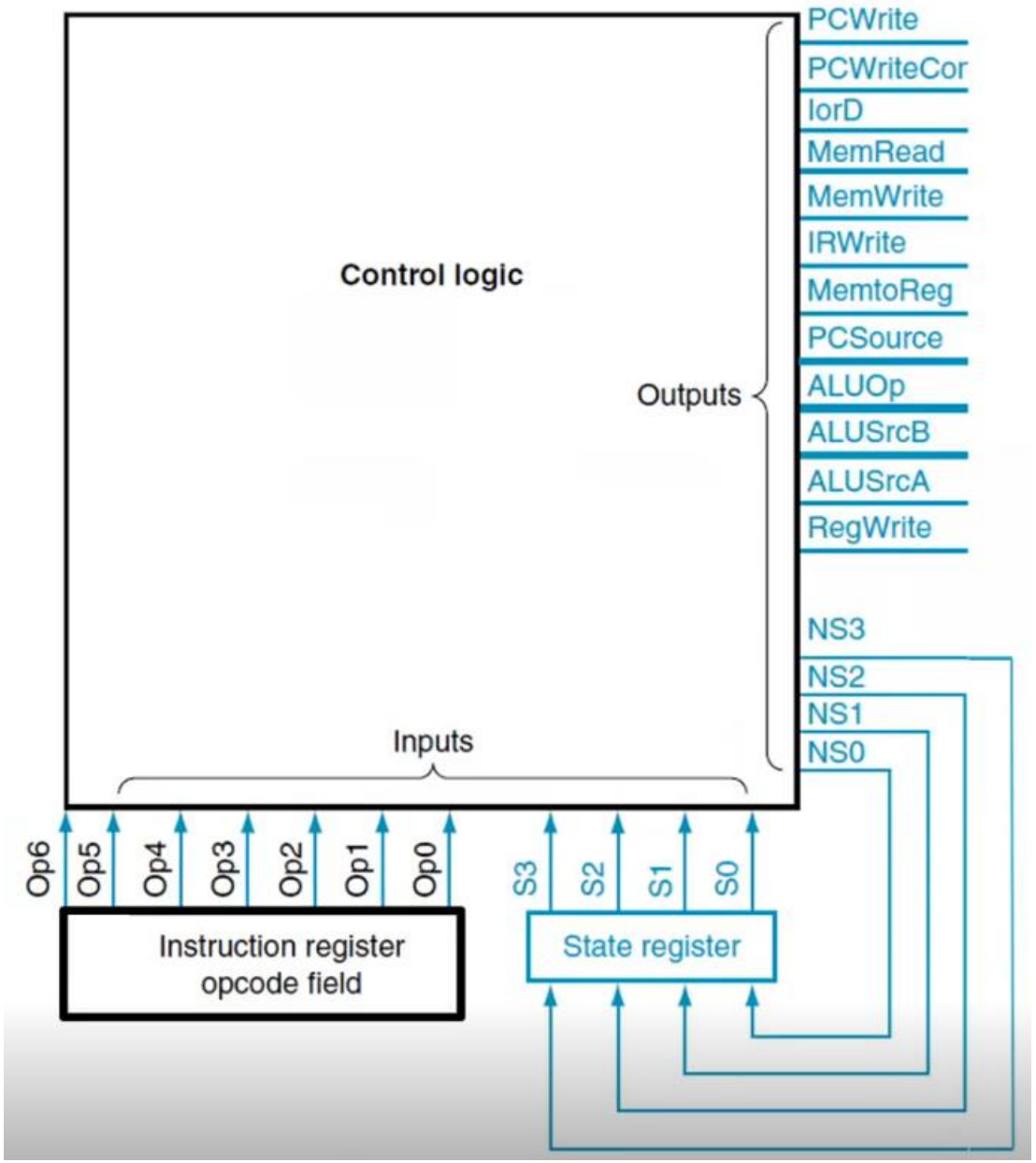


Unidade de controle

- Uma máquina de estados finita
- Ela identifica as configurações de todos os ciclos de clock
- Quantos estados possíveis?



Unidade de Controle - Multiciclo



- **NS0** à **NS3**: 4 bits referentes à identificação do próximo estado, baseado:
  - No estado atual (S0 à S3)

    ou
  - No estado atual + opcode

- ***State register***: registrador que guarda qual é o próximo estado

Unidade de controle

(objetivo)

•Para especificar os **sinais de controle** para a via de dados: depende somente dos **bits dos estados**

•Para especificar o **próximo estado** a fim de continuar a execução de uma determinada instrução: depende do **estado corrente** e do ***opcode***

•Para implementar isso há duas técnicas:

- Baseada na máquina finita de estados (que pode ser implementada em **ROM** ou utilizando uma **PLA**)
- Microprograma

Implementações da Máquina Finita de Estados

•**ROM** (Read Only Memory)

•**PLA**(Programmable Logic Array): Estrutura lógica composta de um conjunto de entradas e complementos correspondentes destas entradas e dois estágio de lógica: a primeira gera o produto dos termos da entrada e a segunda gera a soma do produto destes termos. Implemanta uma função lógica da soma dos produtos.

PLA

- Vantagens:
  - Reduz a quantidade de controle a ser armazenado
- Desvantagens:
  - Complexo para implementar