Organização e Arquitetura de Processadores

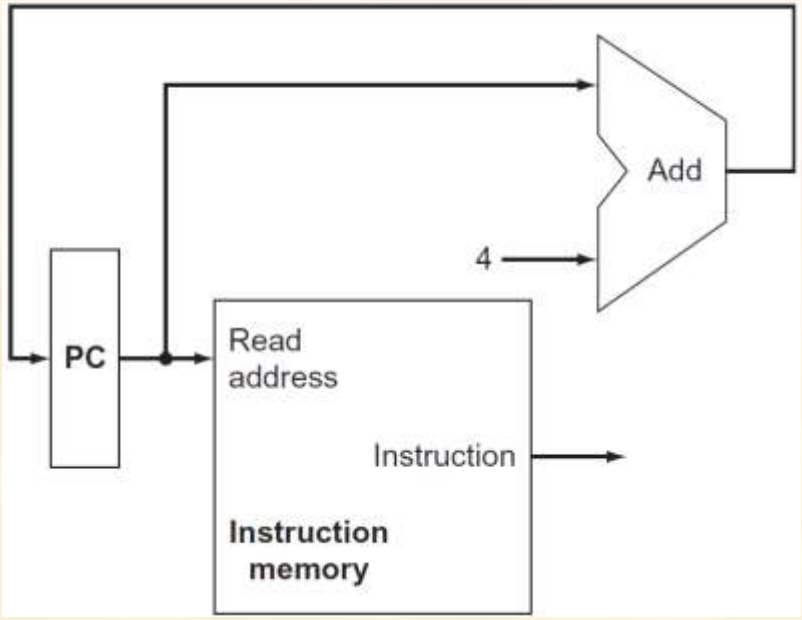
Organização do MIPS

Monociclo

Prof. Dr. César Marcon

Acesso Sequencial à Memória de Instruções

- **Elementos básicos para a busca de instruções na memória**
  - Apontador de programa
  - Memória de instruções
  - Circuito combinacional somador para gerar o próximo endereço de instrução

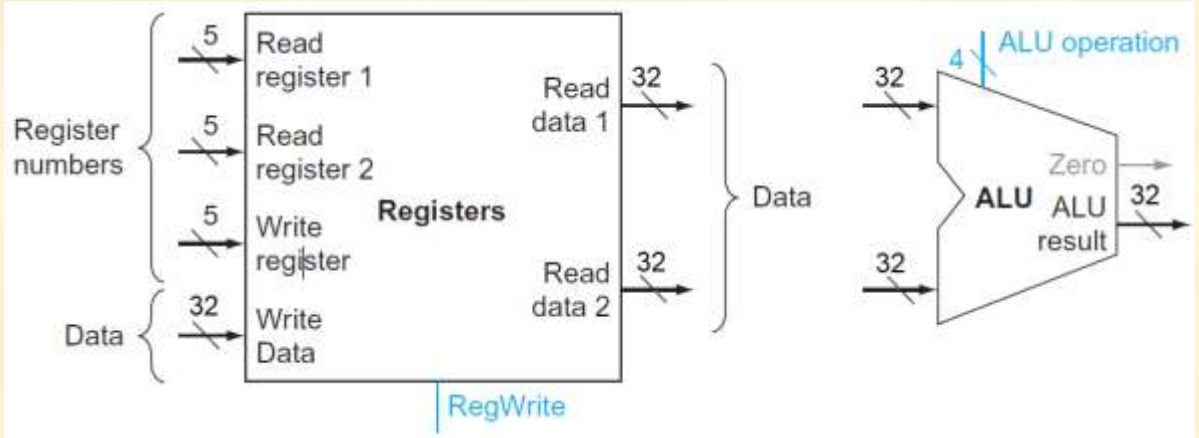


- **Circuito limitado a acessar a memória sequencialmente**

Fluxo de Endereçamento à Registrador (1)

- **Elementos básicos para instruções com formato de acesso à registrador**
  - Banco de registradores
    - 2 endereços de leitura
    - 1 endereço de escrita
    - Leituras e escrita podem ocorrer simultaneamente
    - Escrita necessita habilitar sinal **RegWrite**
    - Leitura está sempre habilitada
  - Unidade lógica e aritmética
    - Operandos de 32 bits
    - 4 bits para comandos **ALUoperation**, potencialmente $2^4 = 16$ instruções
    - Flag de **Zero** para informar resultado da operação

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

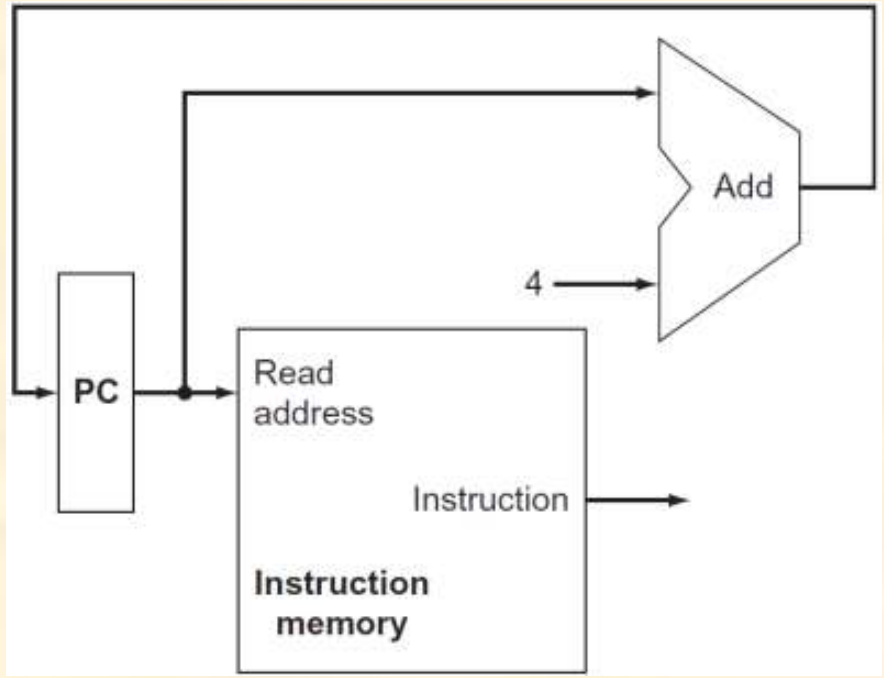


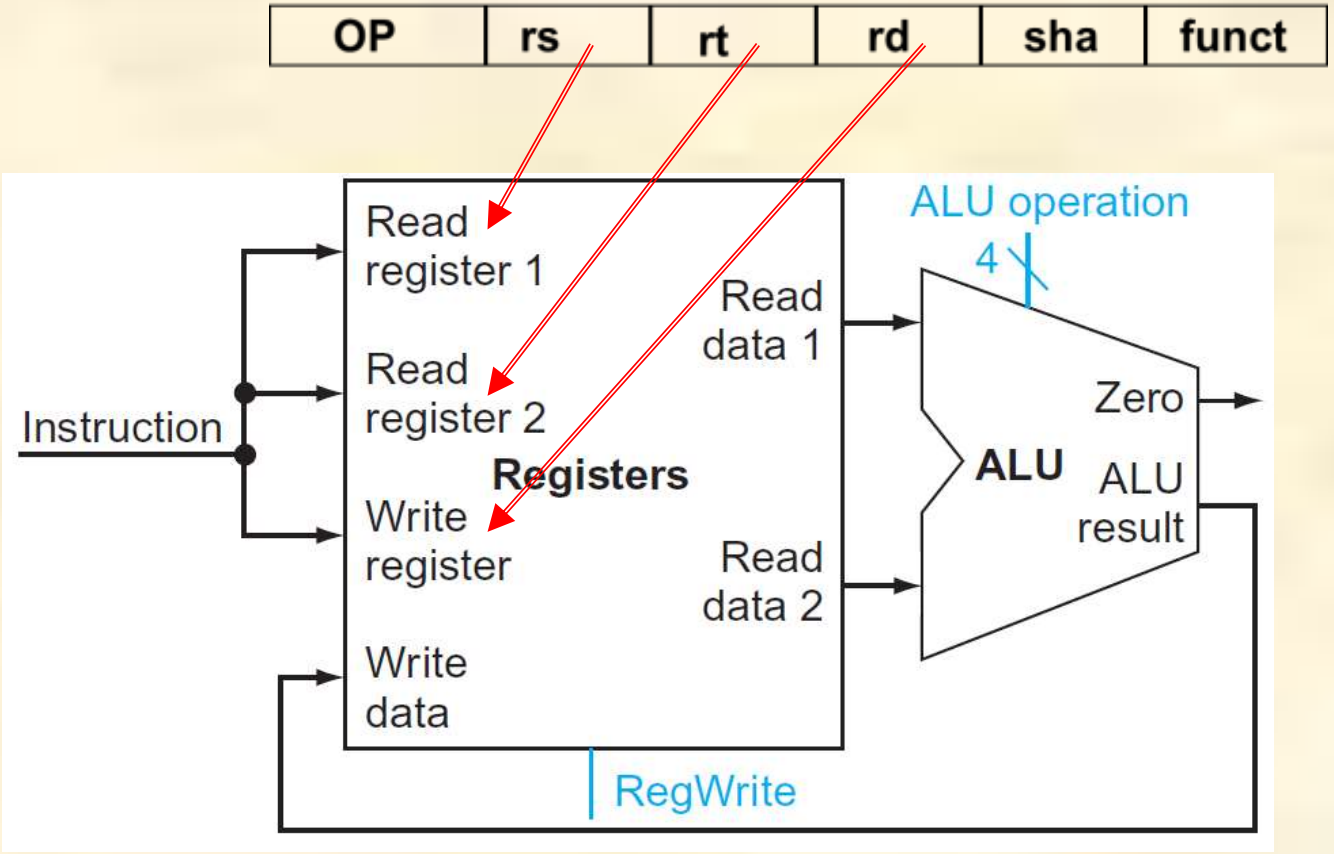
Fluxo de Endereçamento à Registrador

- **Acesso à registrador simultaneamente opera com banco de registradores, ULA e circuito de incremento do PC para busca de nova instrução**

- **Exemplo: add rd, rs, rt → rd = rs + rt**

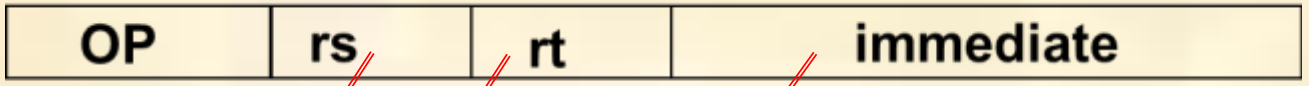
PC = PC + 4

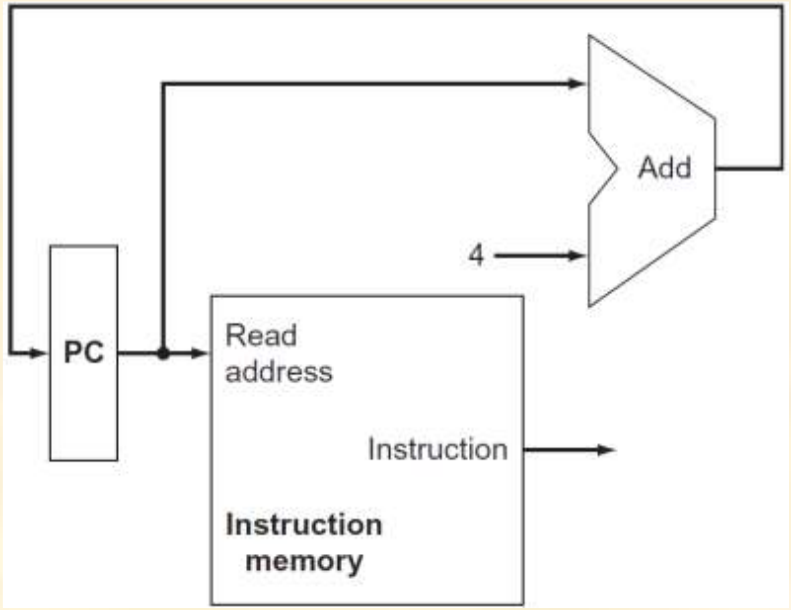




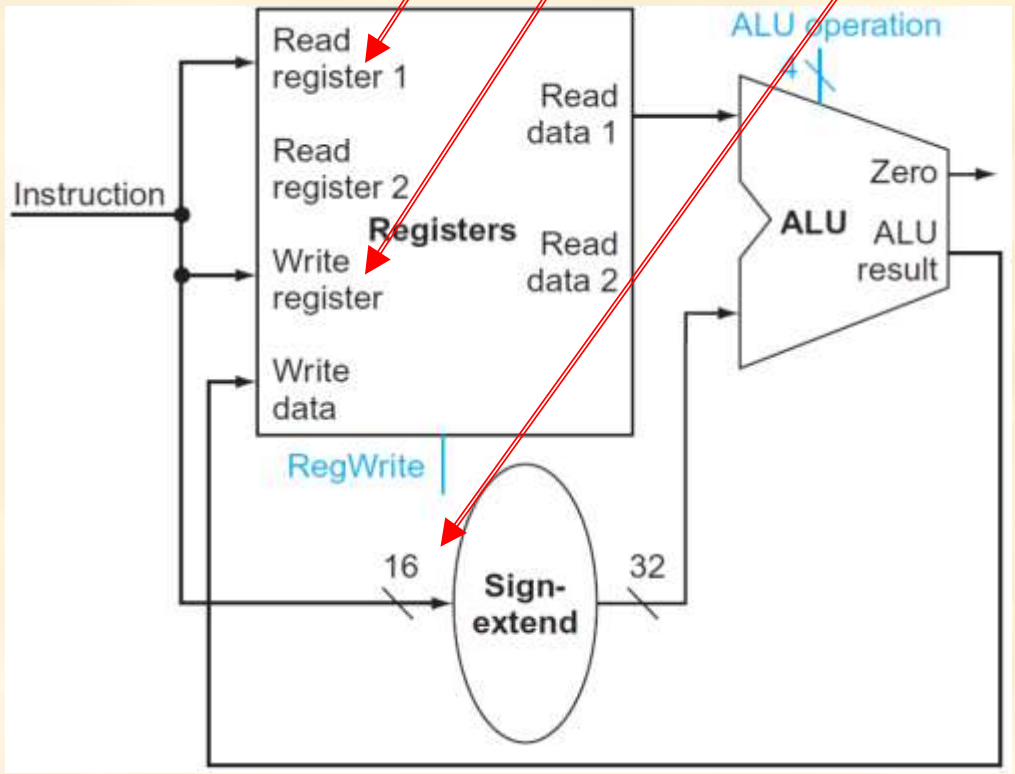
Fluxo de Endereçamento Imediato

- **Utiliza os mesmos recursos do acesso à registrador, trocando um registrador pelo imediato**
- **Exemplo: addi rt, rs, 100 → rt = rs + 100**





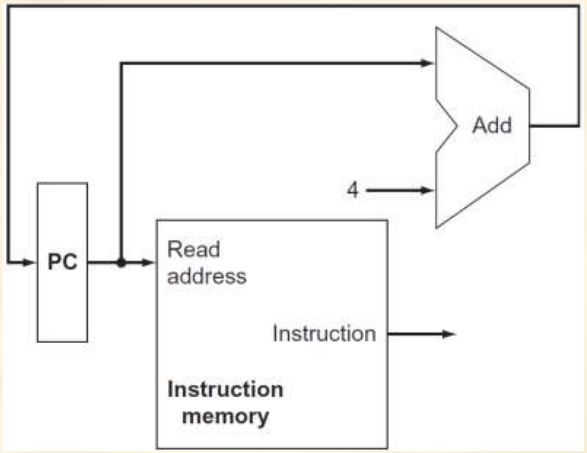
PC = PC + 4



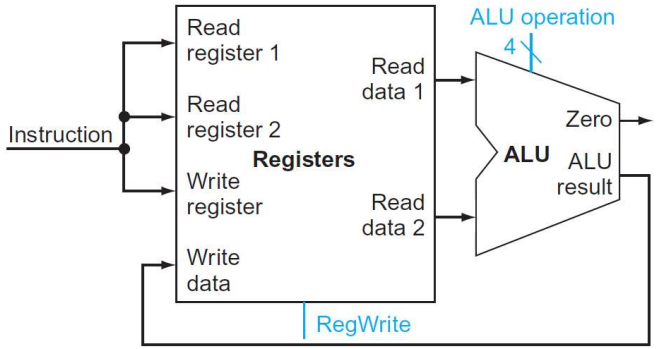
Fluxo de Endereçamento à Registrador + Imediato

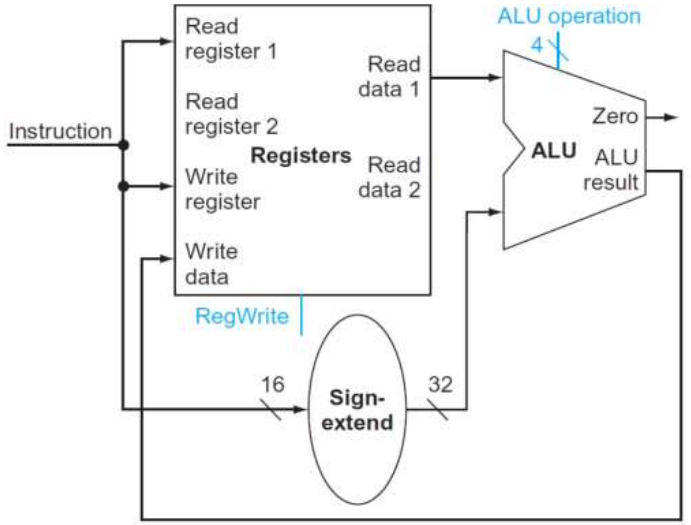
- **Adiciona multiplexador para dar suporte para selecionar o tipo de instrução**
  - Emprega o sinal **ALUSrc**

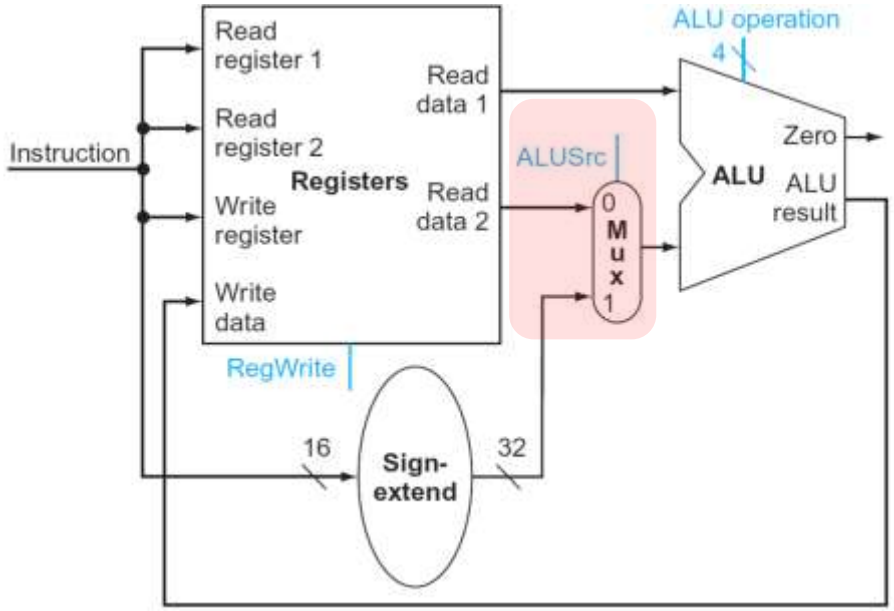
- **Exemplo:**
  - add rd, rs, rt
  - addi rt, rs, 100



PC = PC + 4

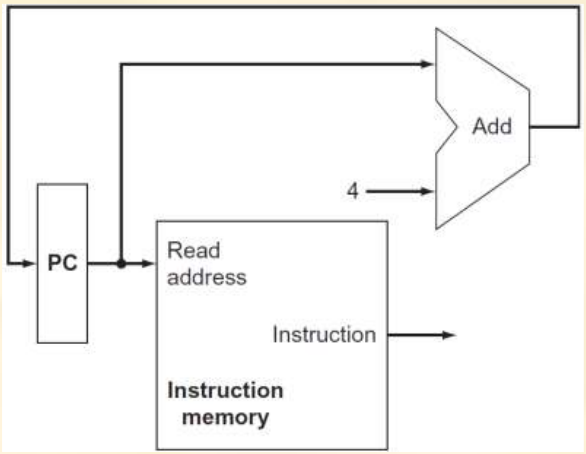




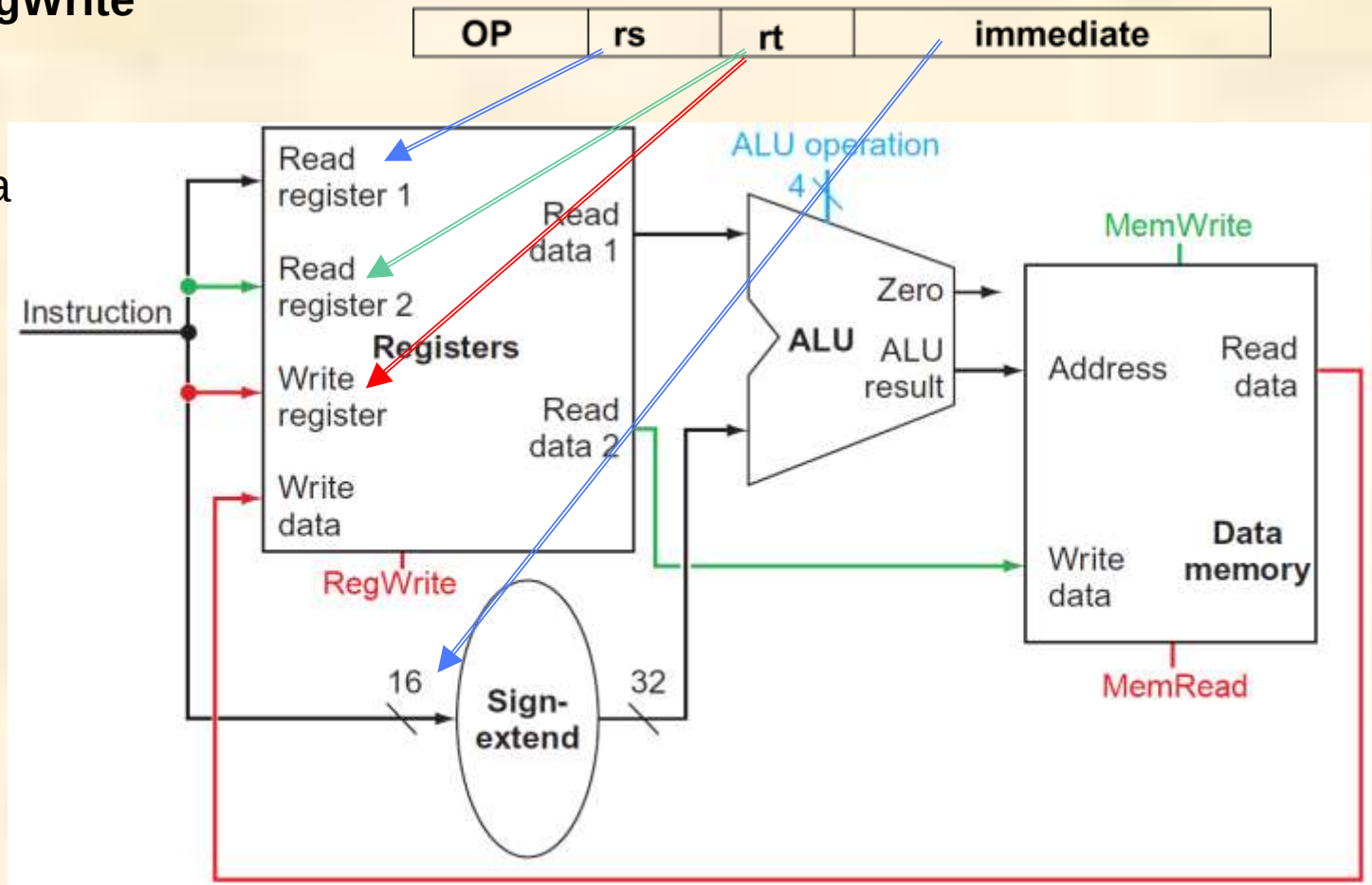


Fluxo de Endereçamento de Acesso à Memória

- **Modo de endereçamento Base-Deslocamento empregado para instruções load/store, com simultâneo incremento ao PC**
  - Emprega unidade de extensão do sinal para transformar imediato de 16 bits em 32 bits
  - Load requer os sinais de **MemRead e RegWrite**
  - Store requer o sinal de **MemWrite**
  - Load e store usam **ALUoperation** para produzir o endereço de acesso à memória
- **Exemplos:**
  - **lw rt, 12(rs)** → rt = Mem[12 + rs]
  - **sw rt, 12(rs)** → Mem[12 + rs] = rt

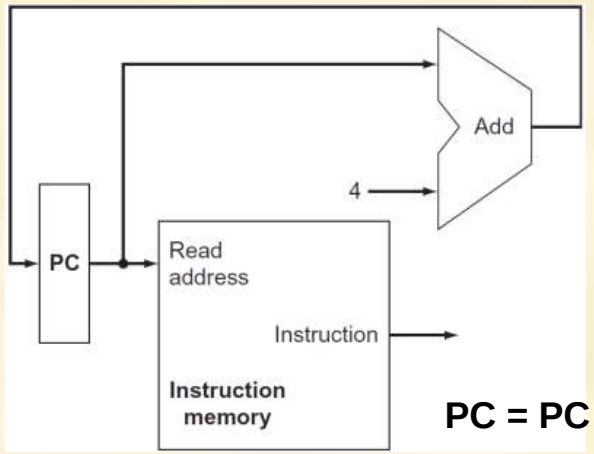


PC = PC + 4

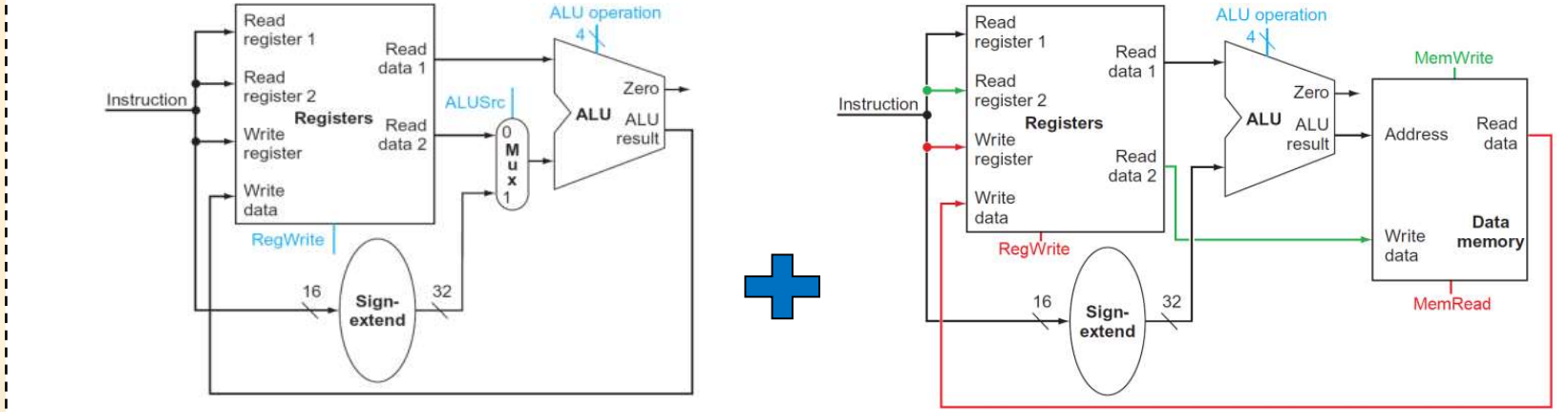


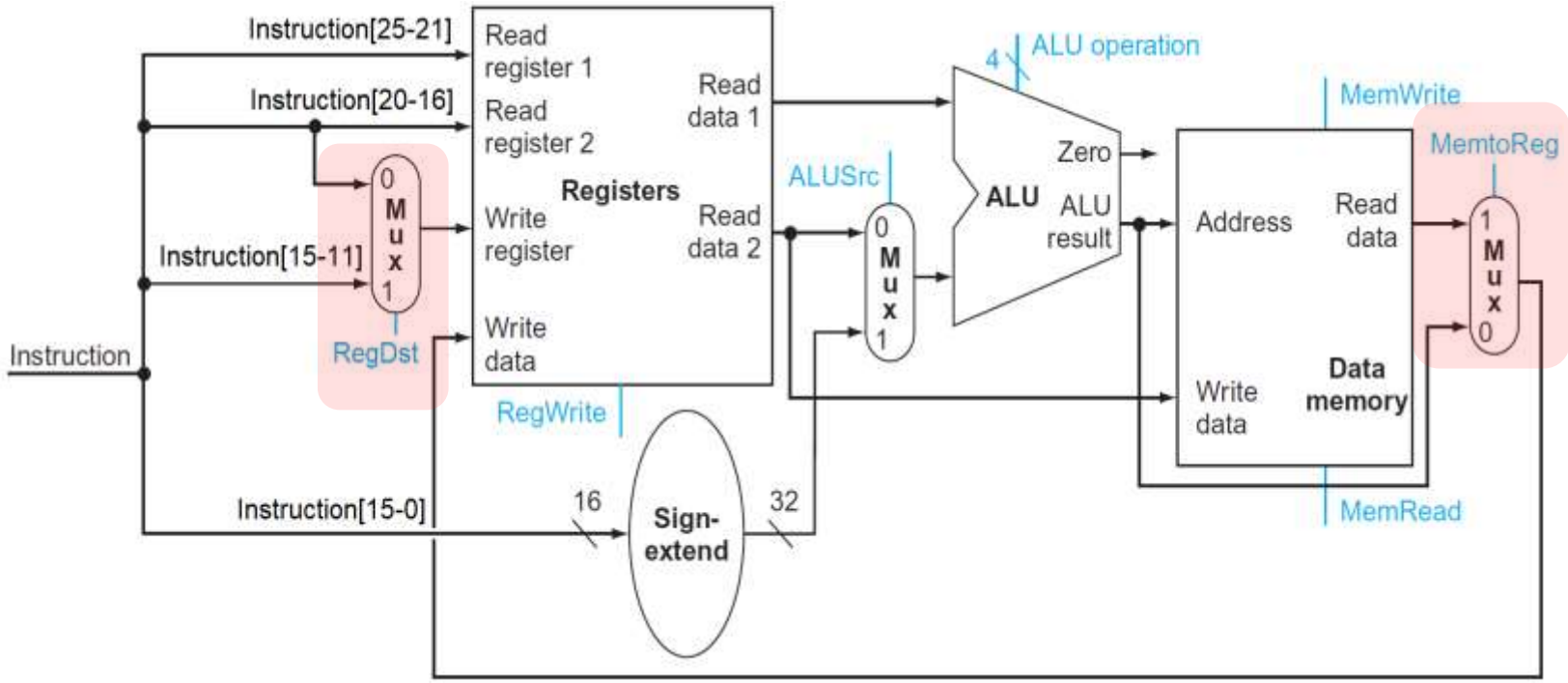
Fluxo de Endereçamento Registrador + Imediato + Memória

- **Adiciona 2 multiplexadores para suportar a seleção dos tipos de instruções**
  - Sinal **MemtoReg**
  - Sinal **RegDst**
- **Exemplo:**
  - add rd, rs, rt
  - addi rt, rs, 100
  - lw rt, 12(rs)
  - sw rt, 12(rs)



PC = PC + 4



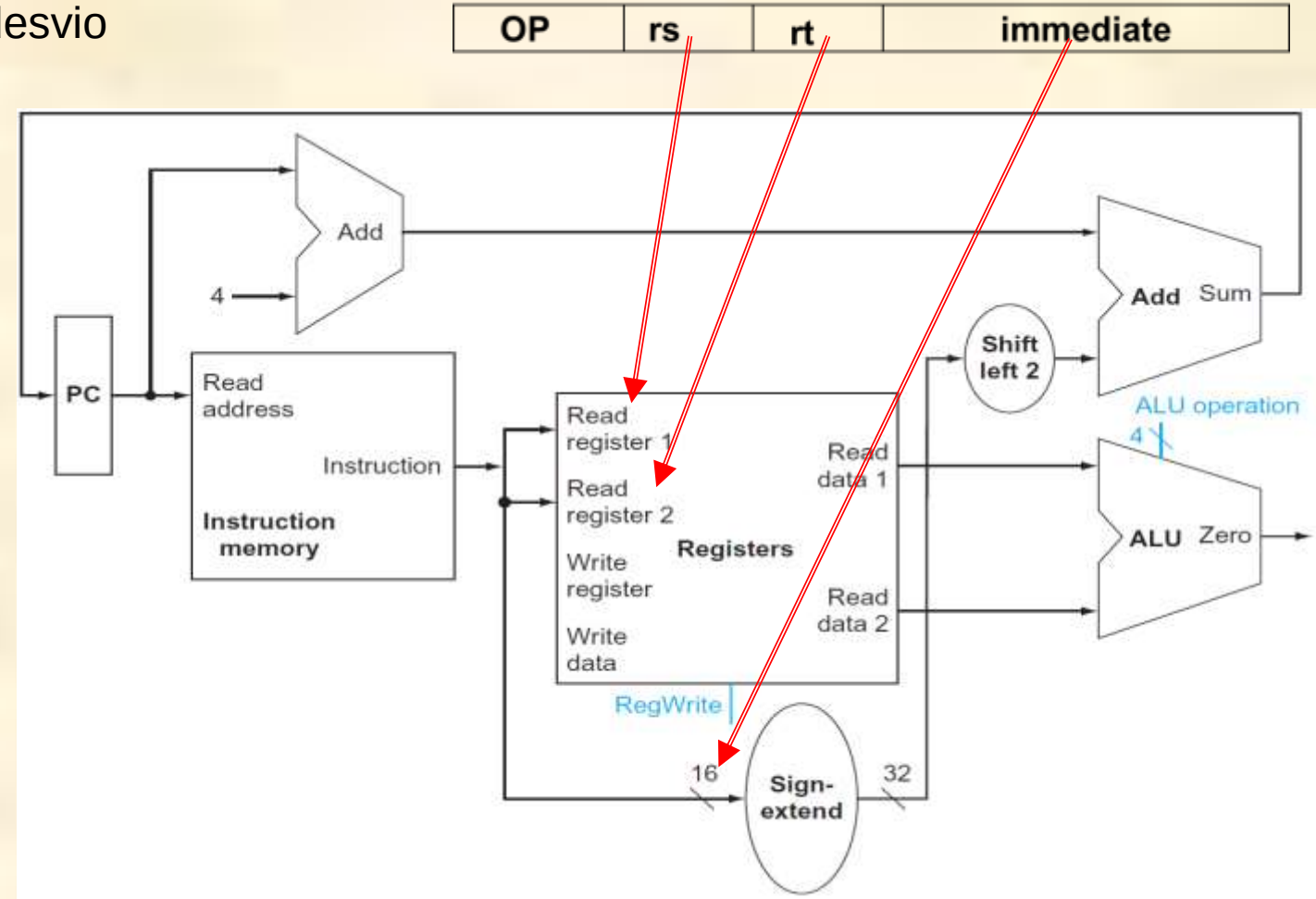


Fluxo de Desvios Condicionais (Somente Salto Tomado)

- **Modo de endereçamento Relativo ao PC**
  - Usa deslocamento de 2 bits (* 4) para aumentar o endereçamento do imediato de 2^{16} para 2^{18}
  - Requer ULA simplificada para somar PC+4 com imediato*4
  - Flag de **ZERO** com resultado do teste de desvio
- **Exemplo: beq rs, rt, imm →**

  PC = (rs == rt) ? PC+4+imm*4 : PC+4

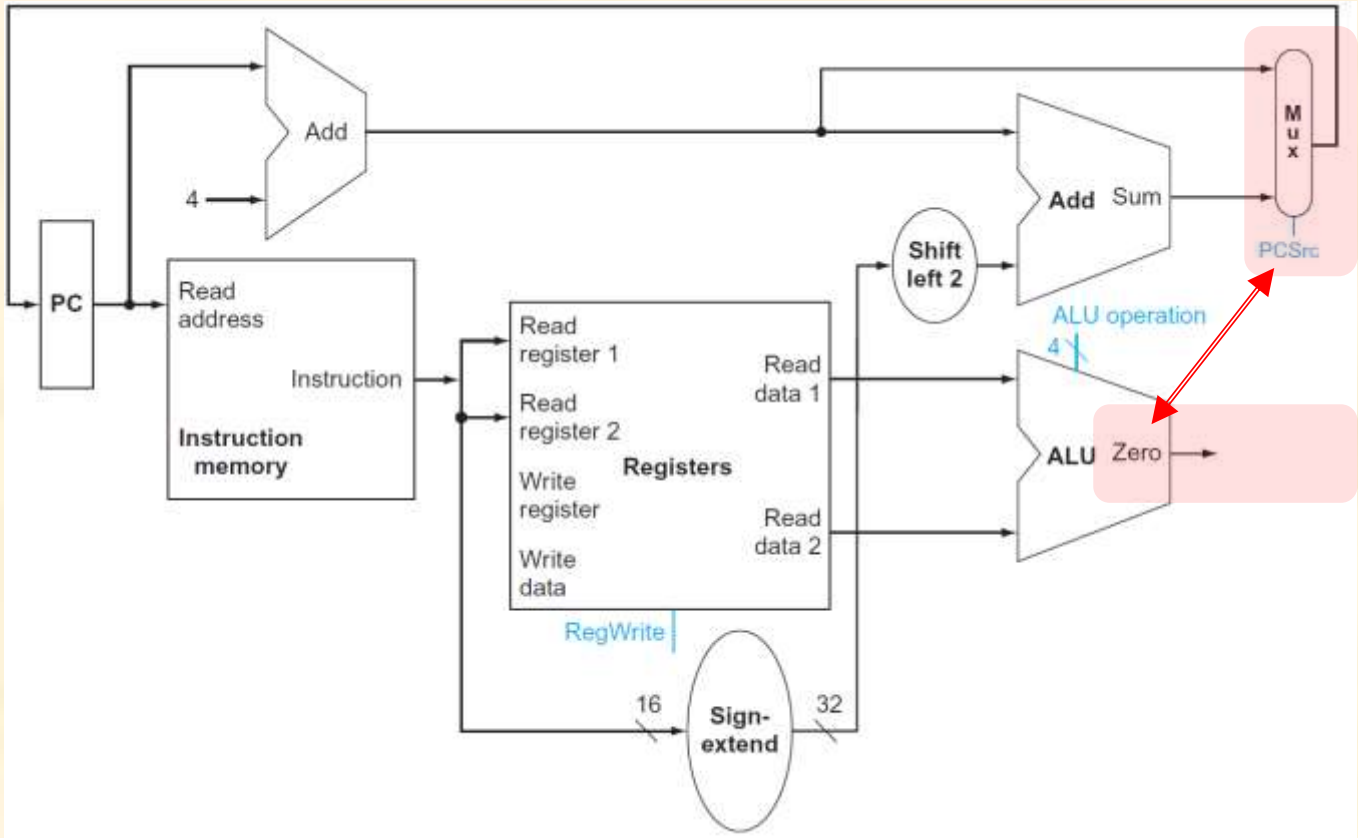
  - Supondo
    - $PC_{ATUAL}+4 = 0x2003F004$
    - imm = 0xAC52



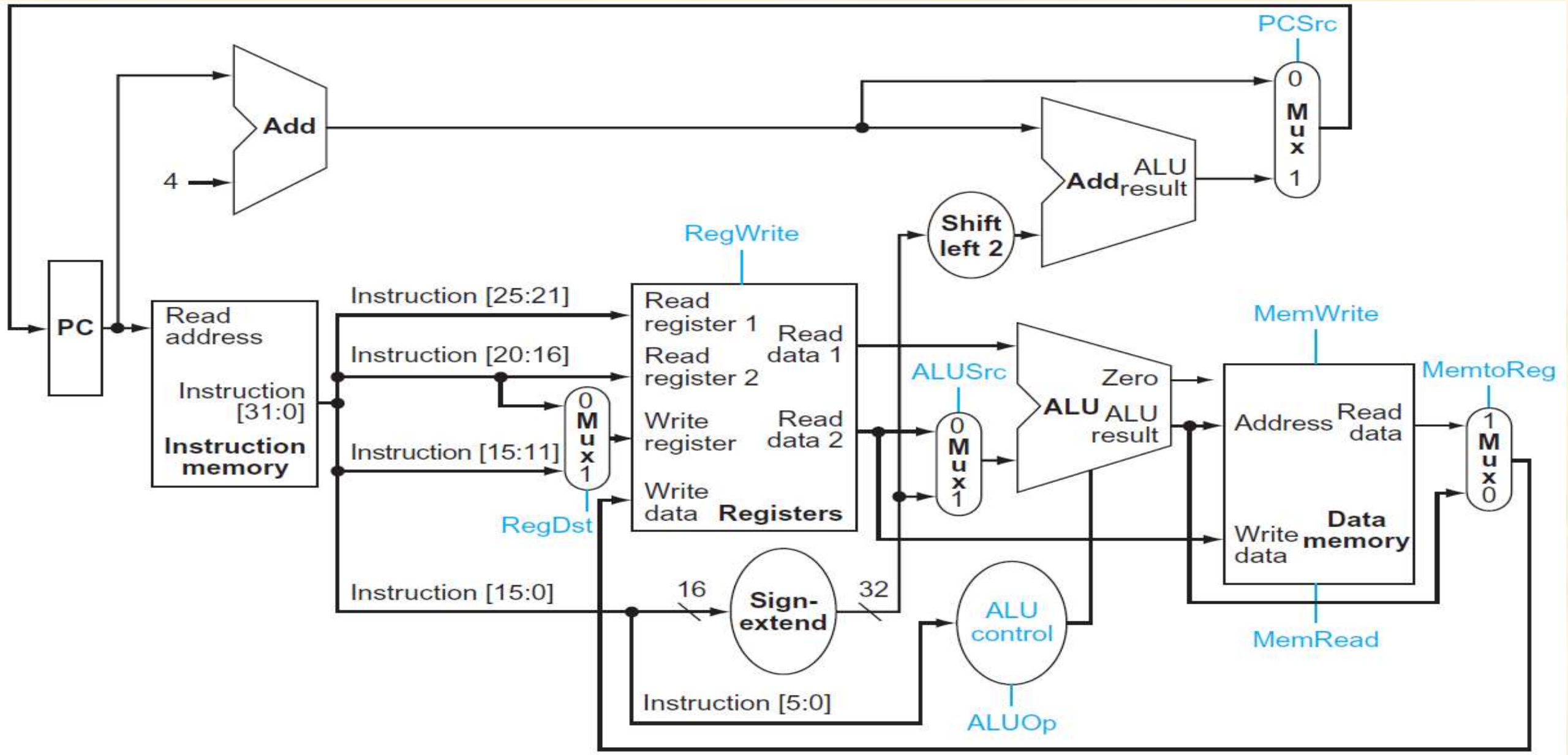
$PC_{ATUAL}+4$:	0010	0000	0000	0011	1111	0000	0000	0100
imm_{16}:					1010	1100	0101	0010
imm_{32}:	1111	1111	1111	1111	1010	1100	0101	0010
imm_{SL}:	1111	1111	1111	1110	1011	0001	0100	1000
PC_{DESVIO}:	0010	0000	0000	0010	1010	0001	0100	1100

Fluxo de Desvios Condicionais (completo)

- **A instrução de desvio condicional completa requer a adição de um multipliexador controlado pelo sinal PCSrc**
  - **PCSrc** seleciona entre PC+4+valor*4 e PC+4
  - O controle do sinal **PCSrc** depende do resultado da ULA (flag **ZERO**)

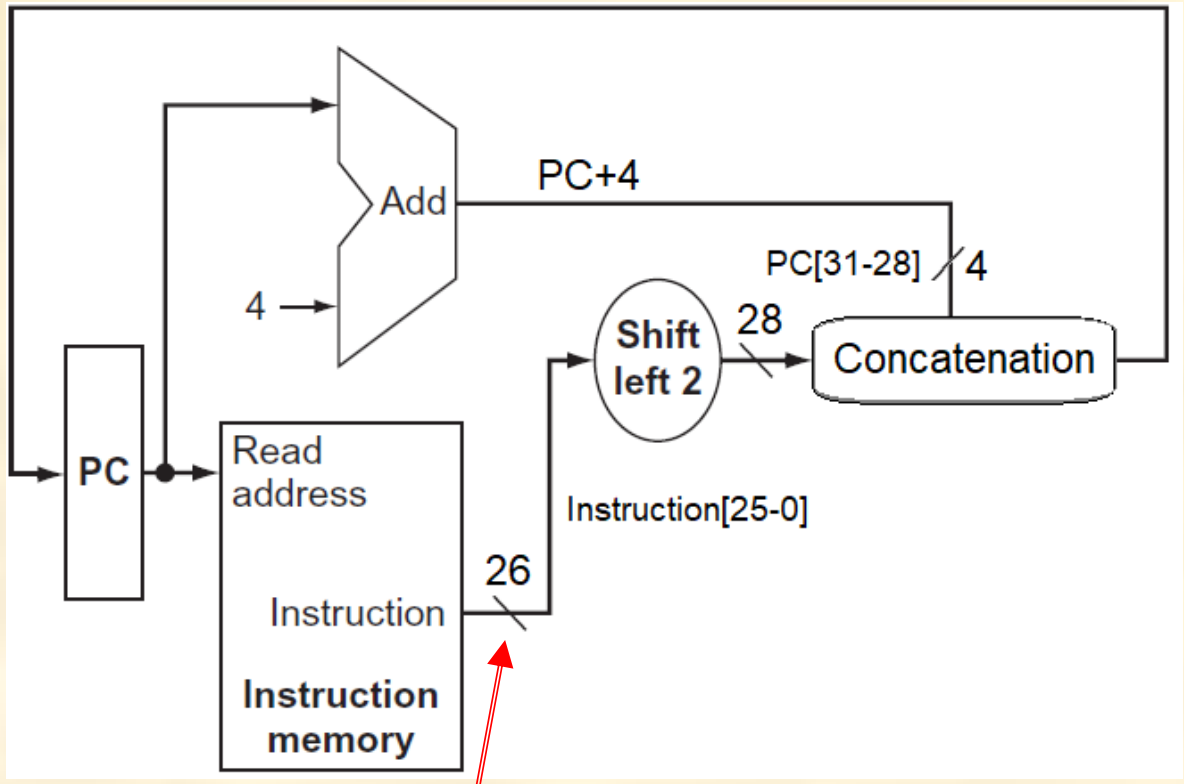


Fluxo de Dados com Modos
à Registrador + Imediato + Memória + Relativo ao PC



Fluxo de Endereçamento Direto à Imediato

- **Instruções de salto incondicionais são implementadas com endereçamento direto ao imediato; um campo de 26 bits, que com deslocamento de 2 bits, permitindo um salto de 2^{28}**
- **Exemplo: j offset**
  - Supondo
    - $PC_{ATUAL}+4 =$ 0x2003F004
    - offset = 0x2E3AC52

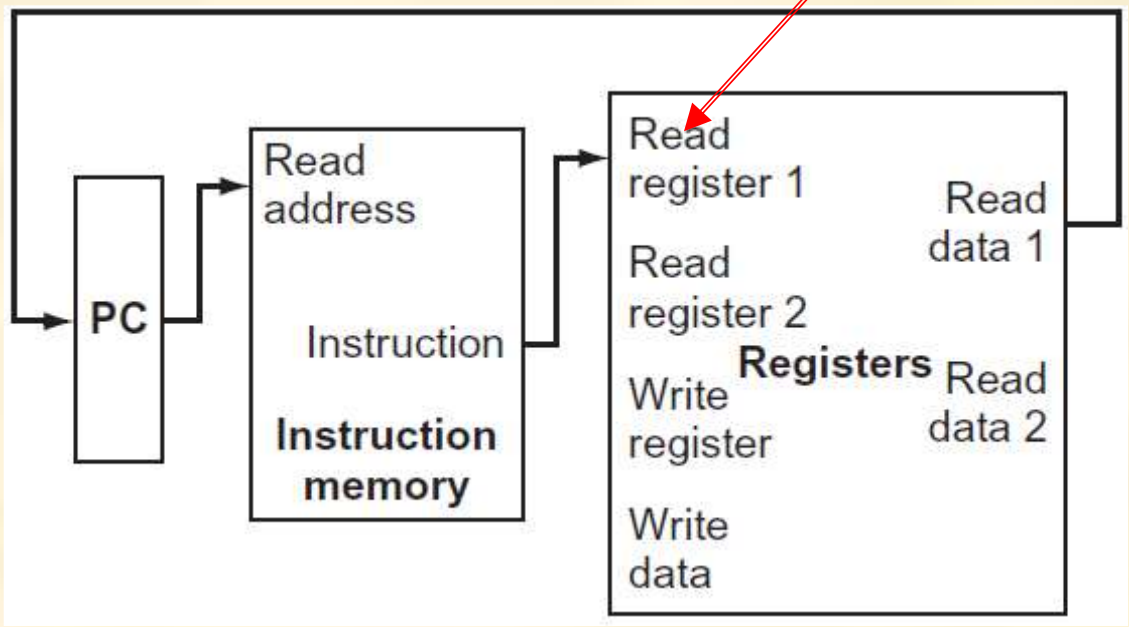


$PC_{ATUAL}+4$: 0010
$offset_{26}$: 10 1110 0011 1010 1100 0101 0010
$offset_{28}$: 1011 1000 1110 1011 0001 0100 1000
PC_{SALTO}: 0010 1011 1000 1110 1011 0001 0100 1000

Fluxo de Endereçamento Direto à Registrador

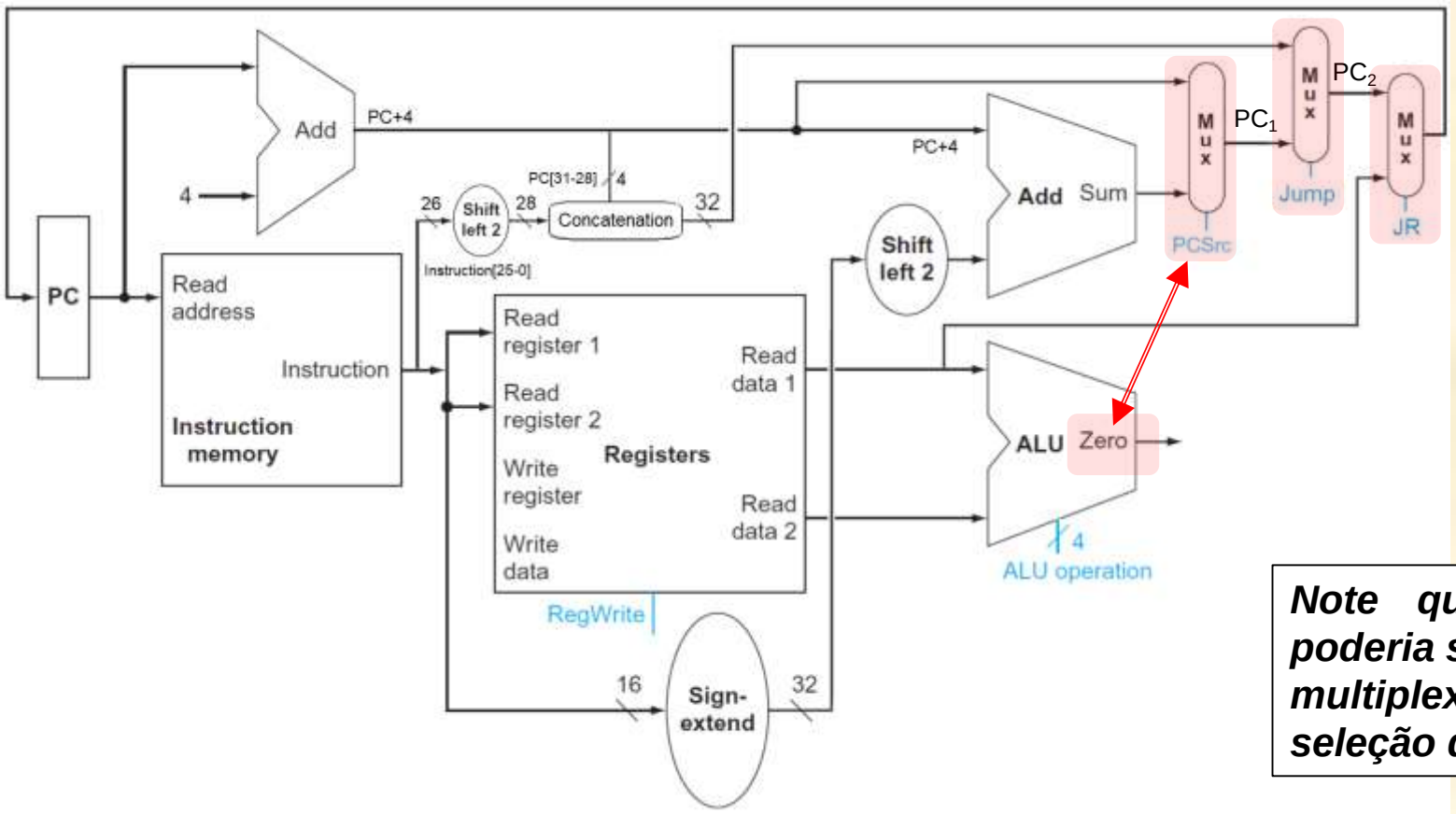
- **Instrução de salto à registrador é realizada com a cópia de um registrador para o PC; um endereçamento direto à registrador**
- **Registrador tem 32, permitindo PC se mover dentro de todo o espaço de 2^{32} endereços**
- **Utiliza instrução com formato à registrador**
- **Exemplo: jr $rs**

jr	rs	0	8



Fluxo das Instruções Sequenciais, de Saltos e de Desvios

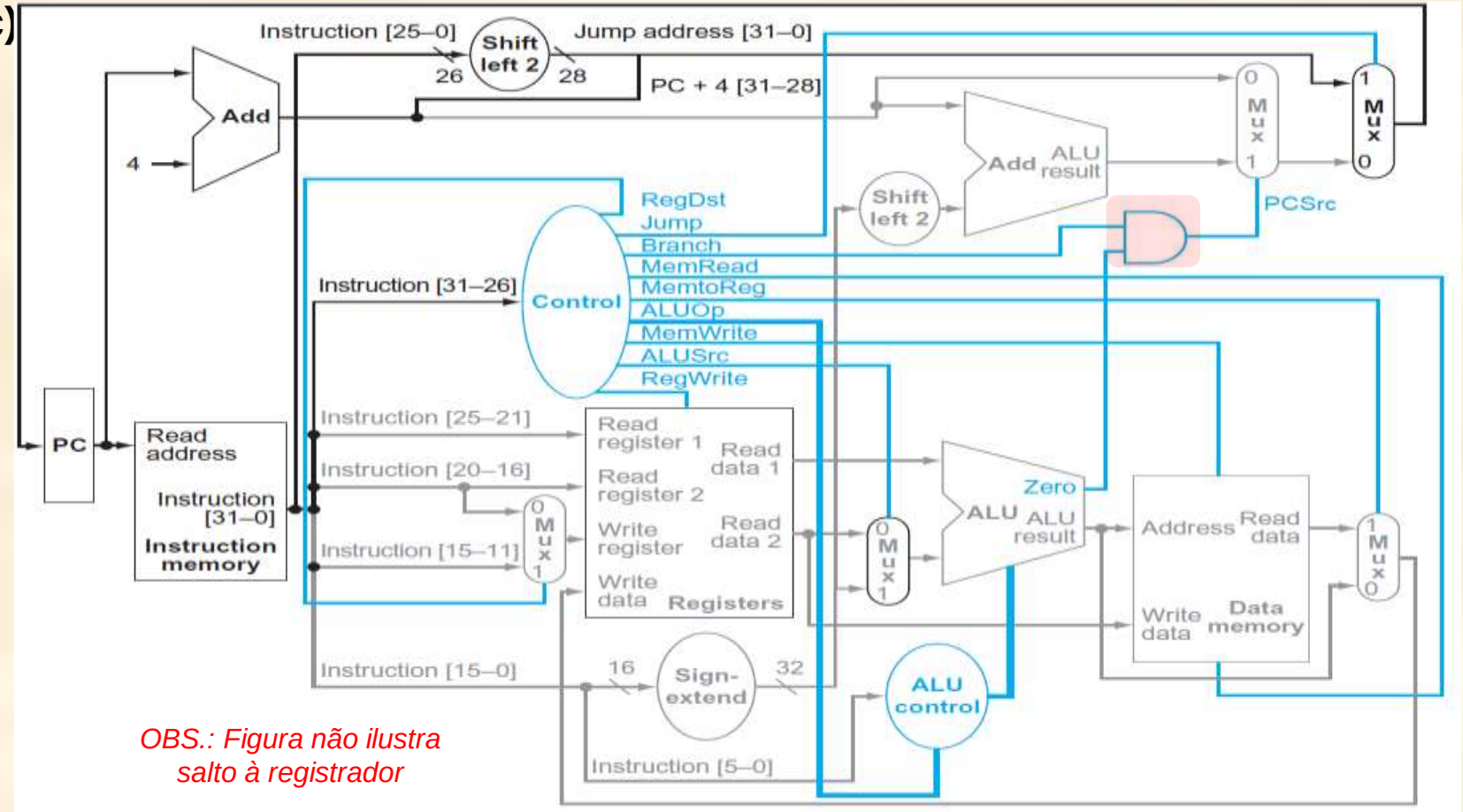
- **A carga do PC depende de três sinais que controlam multiplexadores**
  - **PCSrc** define se o PC_1 será carregado com endereço sequencial, ou de desvio condicional
  - **Jump** define se PC_2 será carregado com endereço imediato de salto ou PC_1
  - **JR** define se PC será carregado com registrador de salto ou PC_2



Note que a seleção do PC poderia ser feita com apenas um multiplexador com 4 entradas e seleção de 2 bits

Organização MIPS Monociclo Completa

- **Bloco de Controle da arquitetura monociclo é implementado com um circuito combinacional que reconhece as instruções e gerar os sinais de controle correspondentes**
- **Sinais de controle da arquitetura monociclo simplificada**
  - **ALUOp** (Sinal de 4 bits, que junto ao campo **Instruction[5-0]** da instrução define a operação da ULA)
  - **Branch** (+ **ZERO** controla **PCSrc)**
  - **JR** (Não ilustrado na figura)
  - **RegDst**
  - **Jump**
  - **MemRead**
  - **MemtoReg**
  - **MemWrite**
  - **ALUSrc**
  - **RegWrite**



OBS.: Figura não ilustra salto à registrador

Exercícios

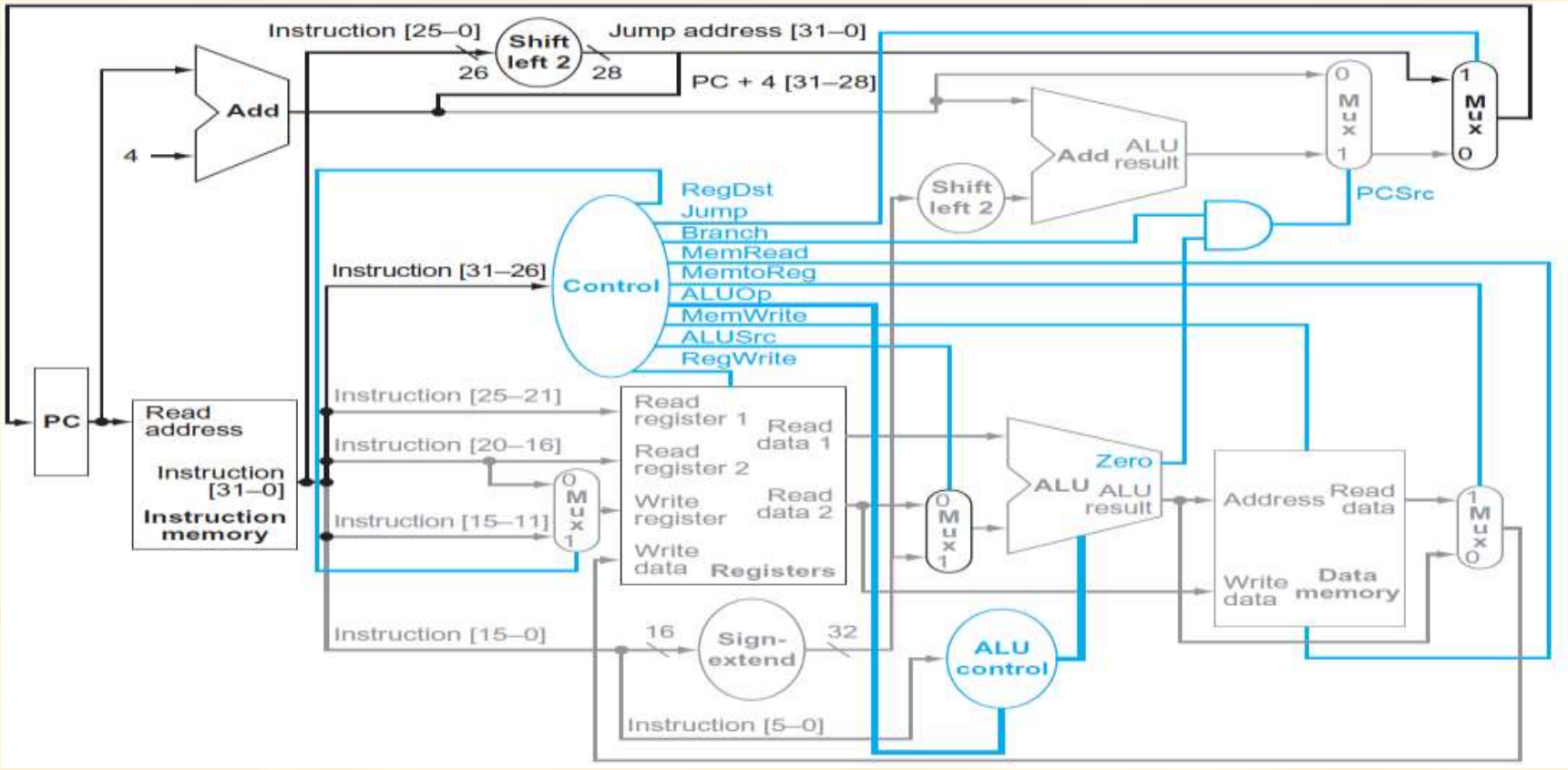
1. Descreva o efeito de manter em 0 os sinais de controle RegDst, ALUSrc, MemtoReg, PCSrc

a) Quais as instruções, se houver alguma, que continuariam a funcionar?

- ADDI r1, r2, imm # r1 = r2 + imm
- LW r1, imm (r2) # r1 = M[r2 + imm]
- BEQ r1, r2, desloc # $pc = $pc + 4 + desloc

b) Considere cada controle separadamente: RegDst=0, ALUSrc=0, MemtoReg=0, PCSrc=0

	[31-26]	[25-21]	[20-16]	[15-0]
addi rt, rs, imm	8	rs	rt	imm
rt ← rs + imm	6	5	5	16
lw rt, offset(rs)	0x23	rs	rt	offset
rt ← MEM[rs + offset]	6	5	5	16
beq rs, rt, target	4	rs	rt	target
$pc ← target / $pc ← $pc+4	6	5	5	16



Resposta

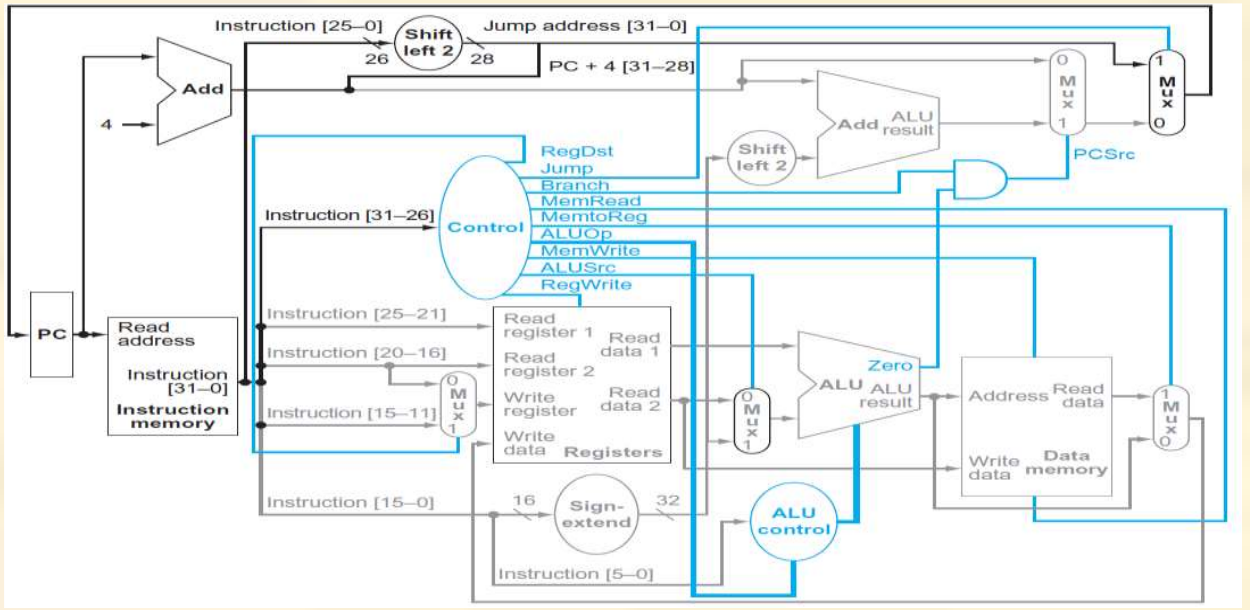
1. Descreva o efeito de manter em 0 os sinais de controle RegDst, ALUSrc, MemtoReg, PCSrc

a) Quais as instruções, se houver alguma, que continuariam a funcionar?

- ADDI r1, r2, imm # r1 = r2 + imm
- LW r1, imm (r2) # r1 = M[r2 + imm]
- BEQ r1, r2, desloc # $pc = $pc + 4 + desloc

	[31-26]	[25-21]	[20-16]	[15-0]
addi rt, rs, imm rt ← rs + imm	8 6	rs 5	rt 5	imm 16
lw rt, offset(rs) rt ← MEM[rs + offset]	0x23 6	rs 5	rt 5	offset 16
beq rs, rt, target $pc ← target / $pc ← $pc+4	4 6	rs 5	rt 5	target 16

Nenhuma irá funcionar



b) Considere cada controle separadamente: RegDst=0, ALUSr =0, MemtoReg=0, PCSrc=0

	RegDst=0	ALUSrc=0	MemtoReg=0	PCSrc=0
ADDI	Ok	Não pode somar r2 com imm	Ok	Ok
LW	Ok	Não pode somar r2 com imm	Não consegue ler a memória	Ok
BEQ	Don't care	Ok	Don't care	Não pode somar $pc+4 com desloc

Exercícios

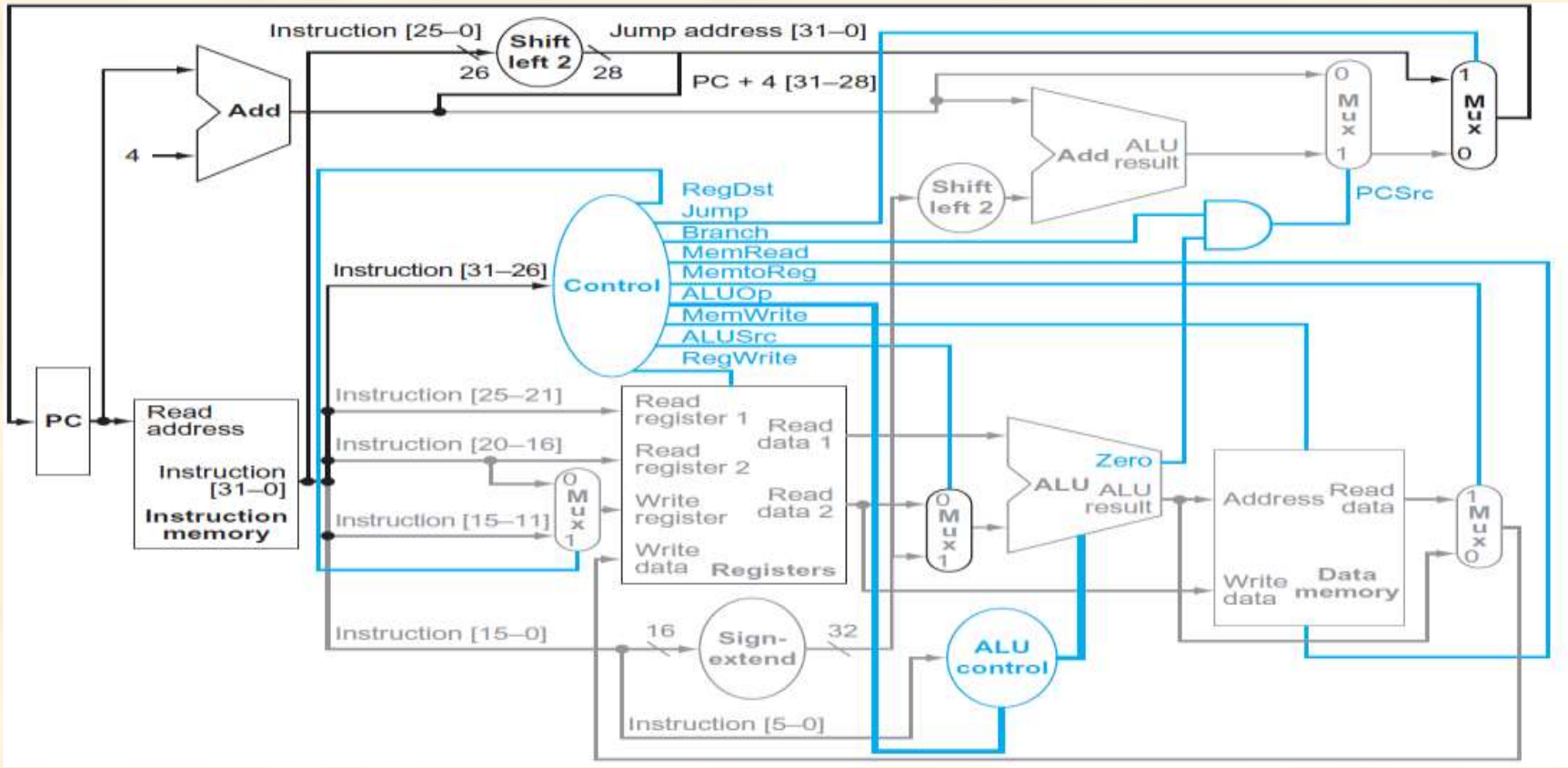
2. Descreva o efeito de manter em 1 os sinais de controle RegDst, ALUSrc, MemtoReg, PCSrc

a) Quais as instruções, se houver alguma, que continuariam a funcionar?

a) ADDI r1, r2, imm # r1 = r2 + imm
b) LW r1, imm (r2) # r1 = M[r2 + imm]
c) BEQ r1, r2, desloc # $pc = $pc + 4 + desloc

b) Considere cada controle separadamente: RegDst=1, ALUSrc=1, MemtoReg=1, PCSrc=1

	[31-26]	[25-21]	[20-16]	[15-0]
addi rt, rs, imm	8	rs	rt	imm
rt ← rs + imm	6	5	5	16
lw rt, offset(rs)	0x23	rs	rt	offset
rt ← MEM[rs + offset]	6	5	5	16
beq rs, rt, target	4	rs	rt	target
$pc ← target / $pc ← $pc+4	6	5	5	16



Resposta

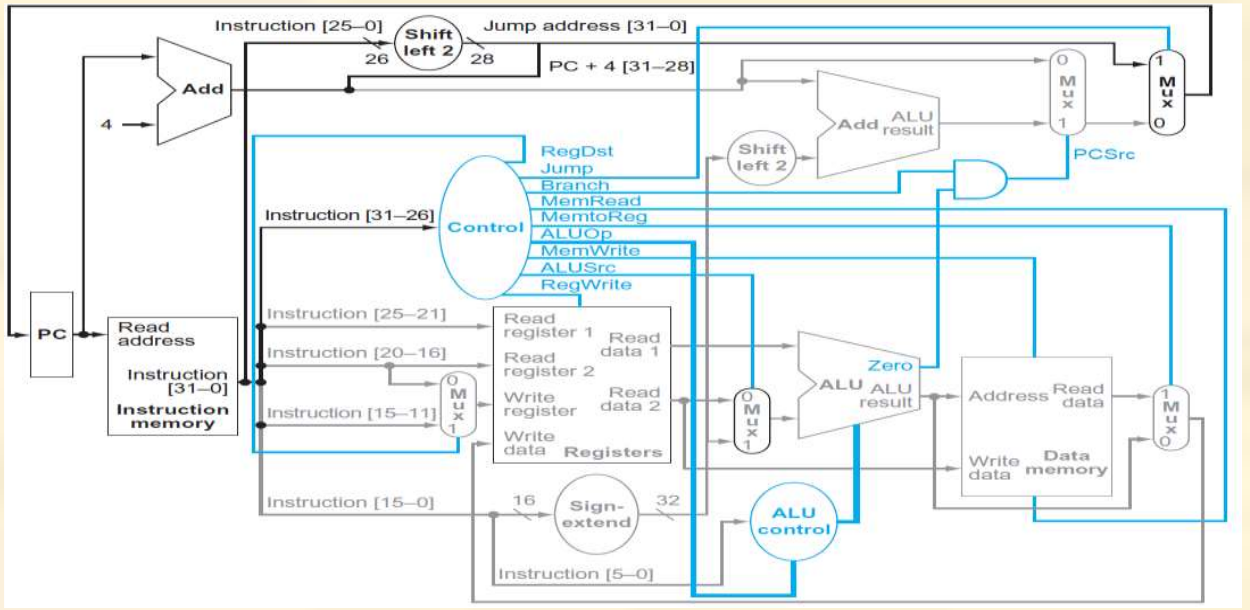
2. Descreva o efeito de manter em 1 os sinais de controle RegDst, ALUSrc, MemtoReg, PCSrc

a) Quais as instruções, se houver alguma, que continuariam a funcionar?

- ADDI r1, r2, imm # r1 = r2 + imm
- LW r1, imm (r2) # r1 = M[r2 + imm]
- BEQ r1, r2, desloc # $pc = $pc + 4 + desloc

	[31-26]	[25-21]	[20-16]	[15-0]
addi rt, rs, imm rt ← rs + imm	8 6	rs 5	rt 5	imm 16
lw rt, offset(rs) rt ← MEM[rs + offset]	0x23 6	rs 5	rt 5	offset 16
beq rs, rt, target $pc ← target / $pc ← $pc+4	4 6	rs 5	rt 5	target 16

Nenhuma irá funcionar

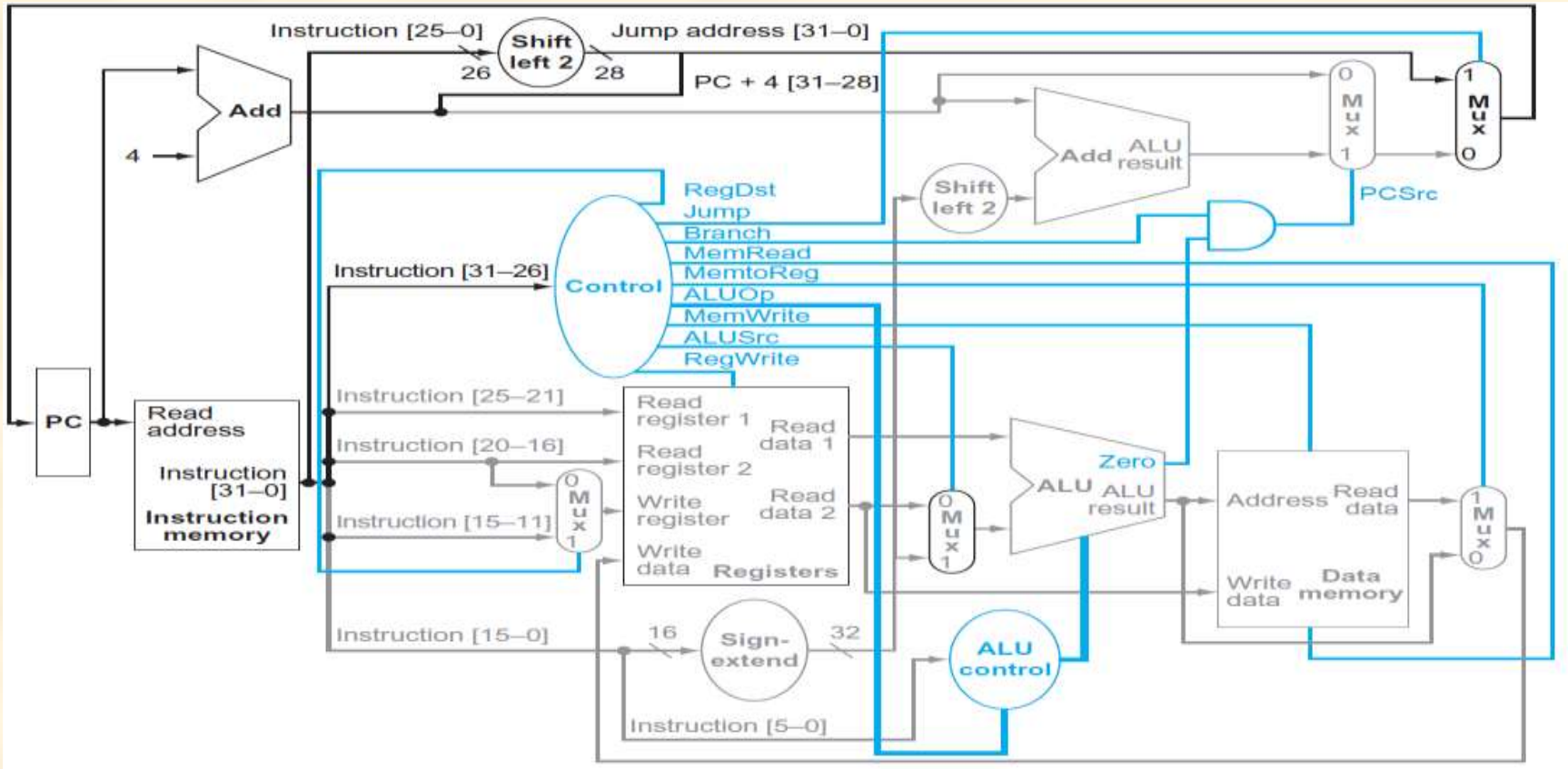


b) Considere cada controle separadamente: RegDst=1, UALFonte=1, MemParaReg=1, FontePC=1

	RegDst=1	ALUSrc=1	MemtoReg=1	PCSrc=1
ADDI	Não pode escrever no registrador destino (r1)	Ok	Não consegue ler a saída da ULA	Irá alterar erradamente o PC
LW	Não pode escrever no registrador destino (r1)	Ok	Ok	Irá alterar erradamente o PC
BEQ	don't care	Não consegue comparar os dois registradores r1 e r2	don't care	Ok

Exercício

3. Seria possível eliminar o sinal de controle MemtoReg usando no seu lugar o sinal MemRead?



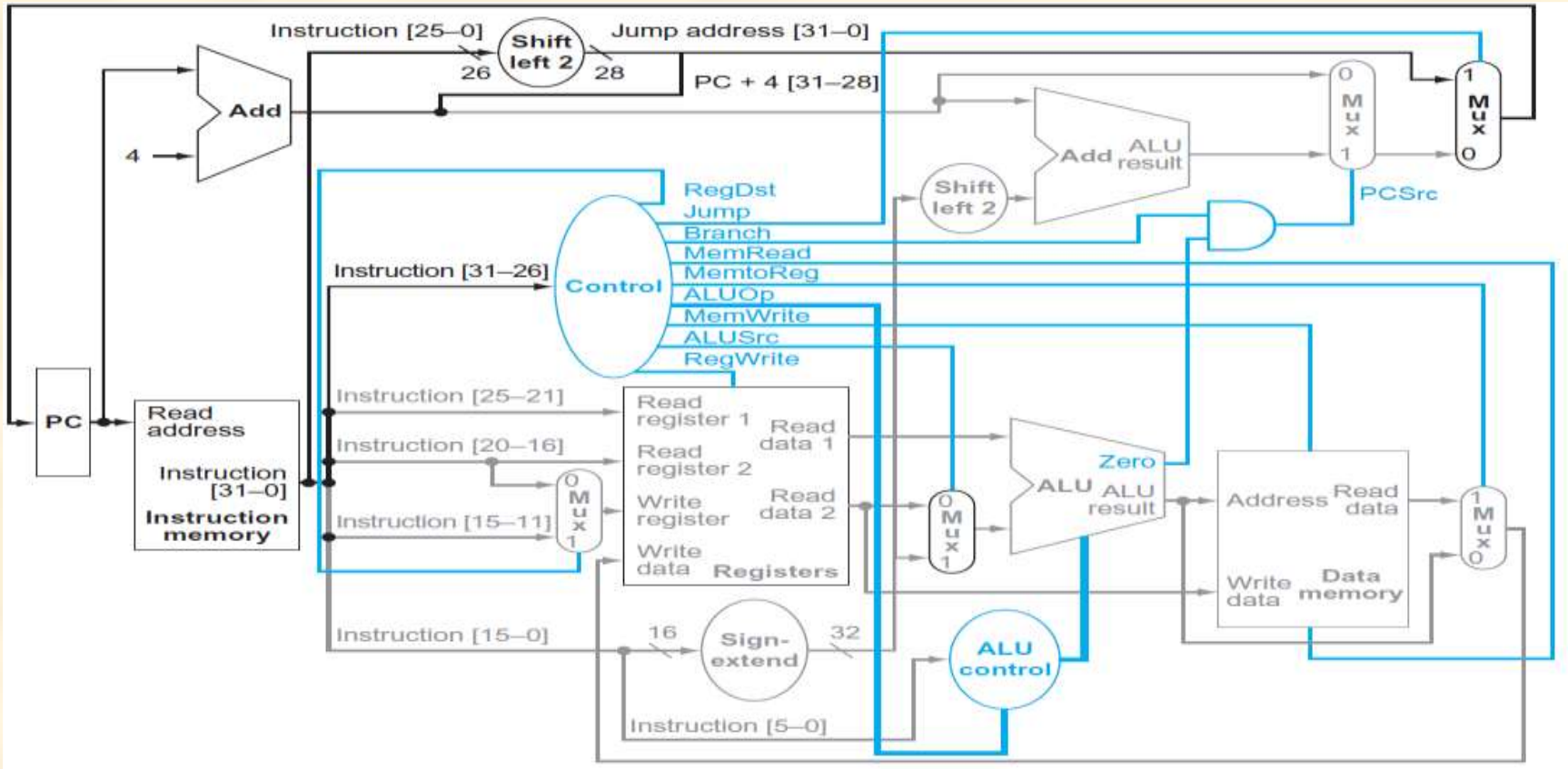
Resposta

3. Seria possível eliminar o sinal de controle MemtoReg usando no seu lugar o sinal MemRead?

O sinal **MemRead** é necessário para instruções do tipo load

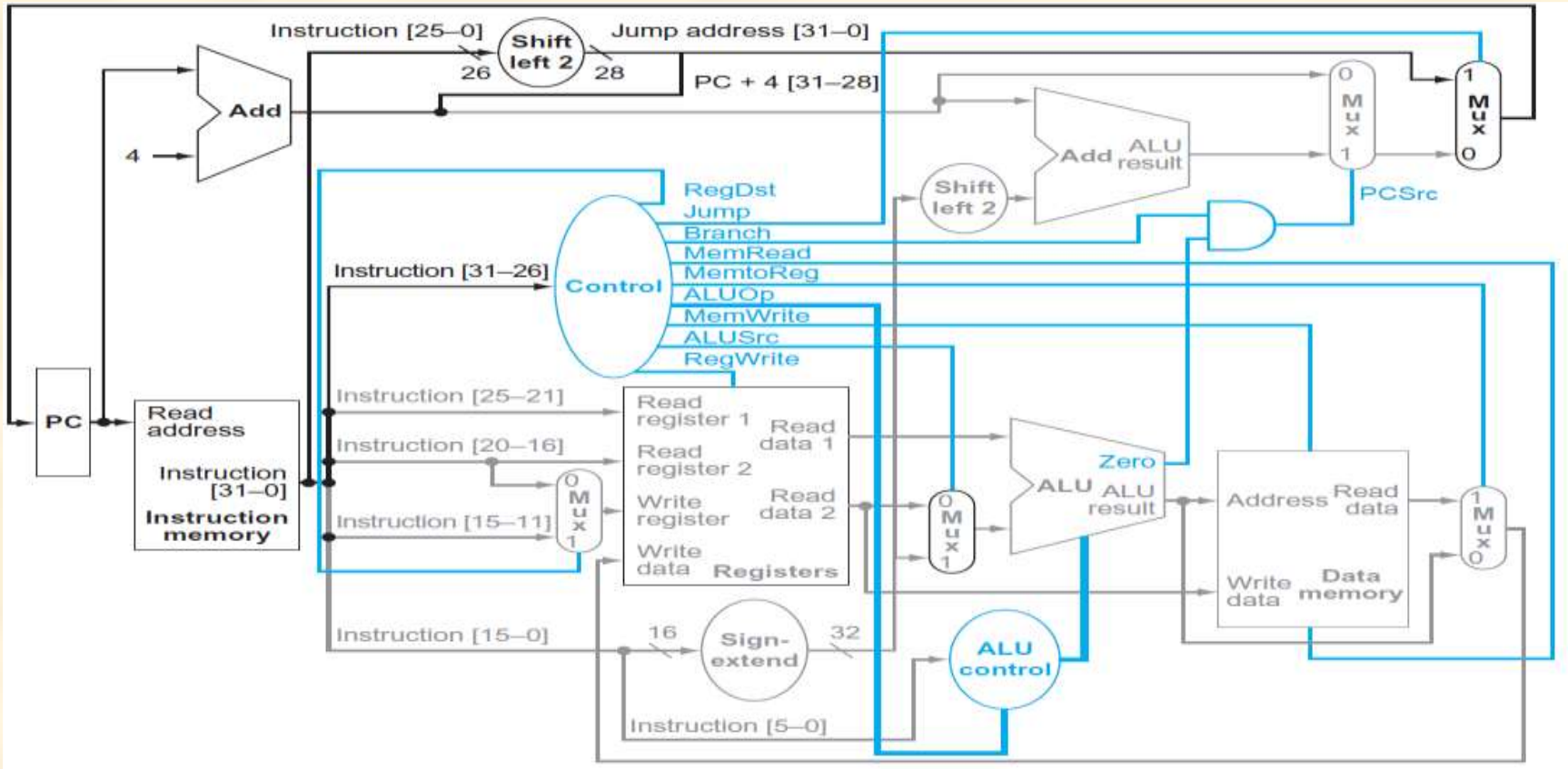
O sinal **MemtoReg** é necessário para selecionar dados provenientes da ULA ou da memória

As únicas instruções que leem a memória no MIPS são do tipo load. Neste caso, **é possível** ter um único sinal para as duas funcionalidades



Exercício

4. Seria possível eliminar o sinal de controle RegWrite usando no seu lugar o sinal RegDst?



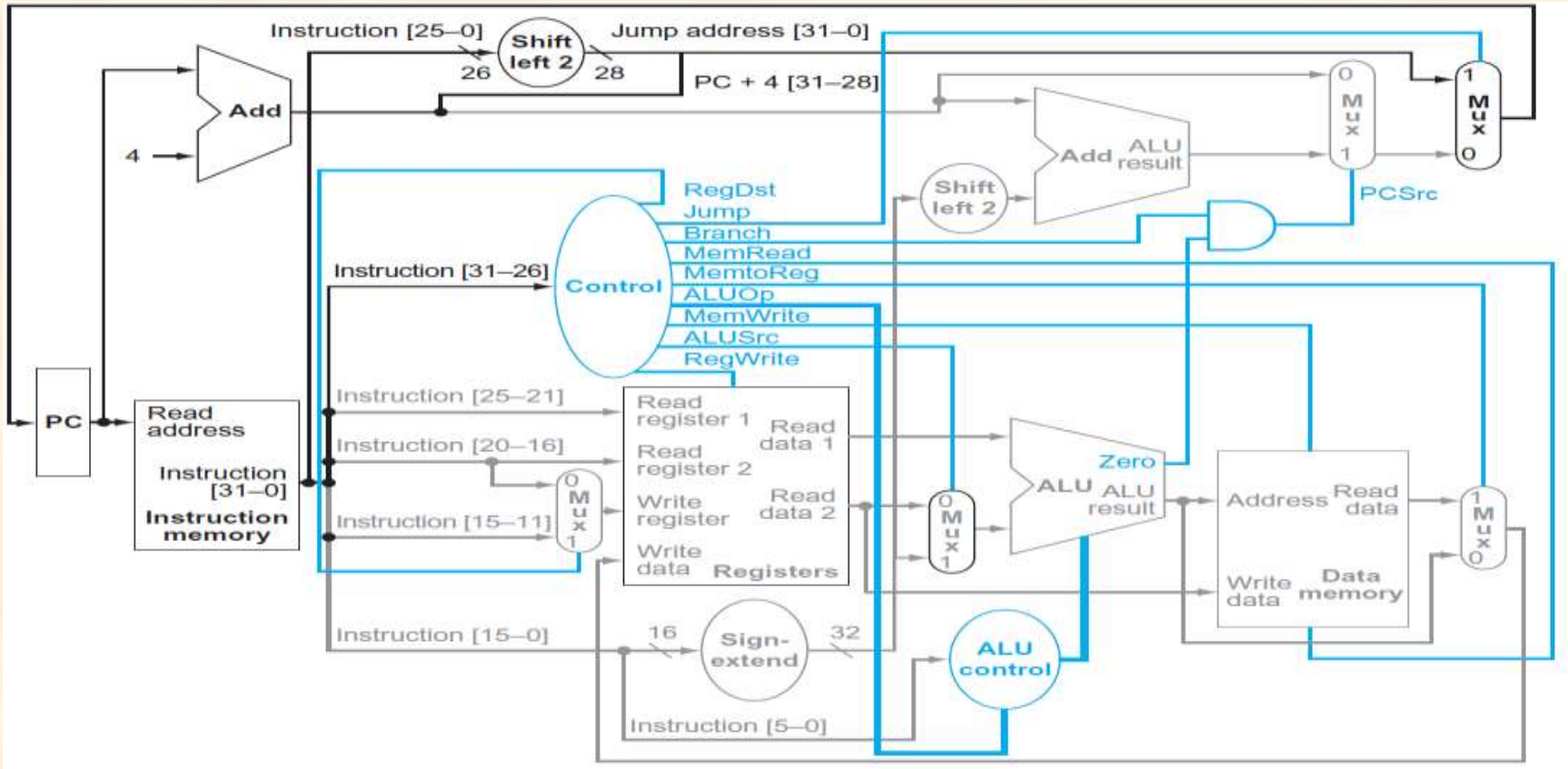
Resposta

4. Seria possível eliminar o sinal de controle RegWrite usando no seu lugar o sinal RegDst?

O sinal **RegWrite** é necessário para habilitar a escrita no banco de registradores

O sinal **RegDst** seleciona o campo de onde virá o endereço do registrador a ser escrito

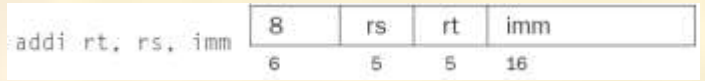
Ambos os campos são de endereço de escrita. Neste caso, **NÃO é possível** compartilhar os sinais

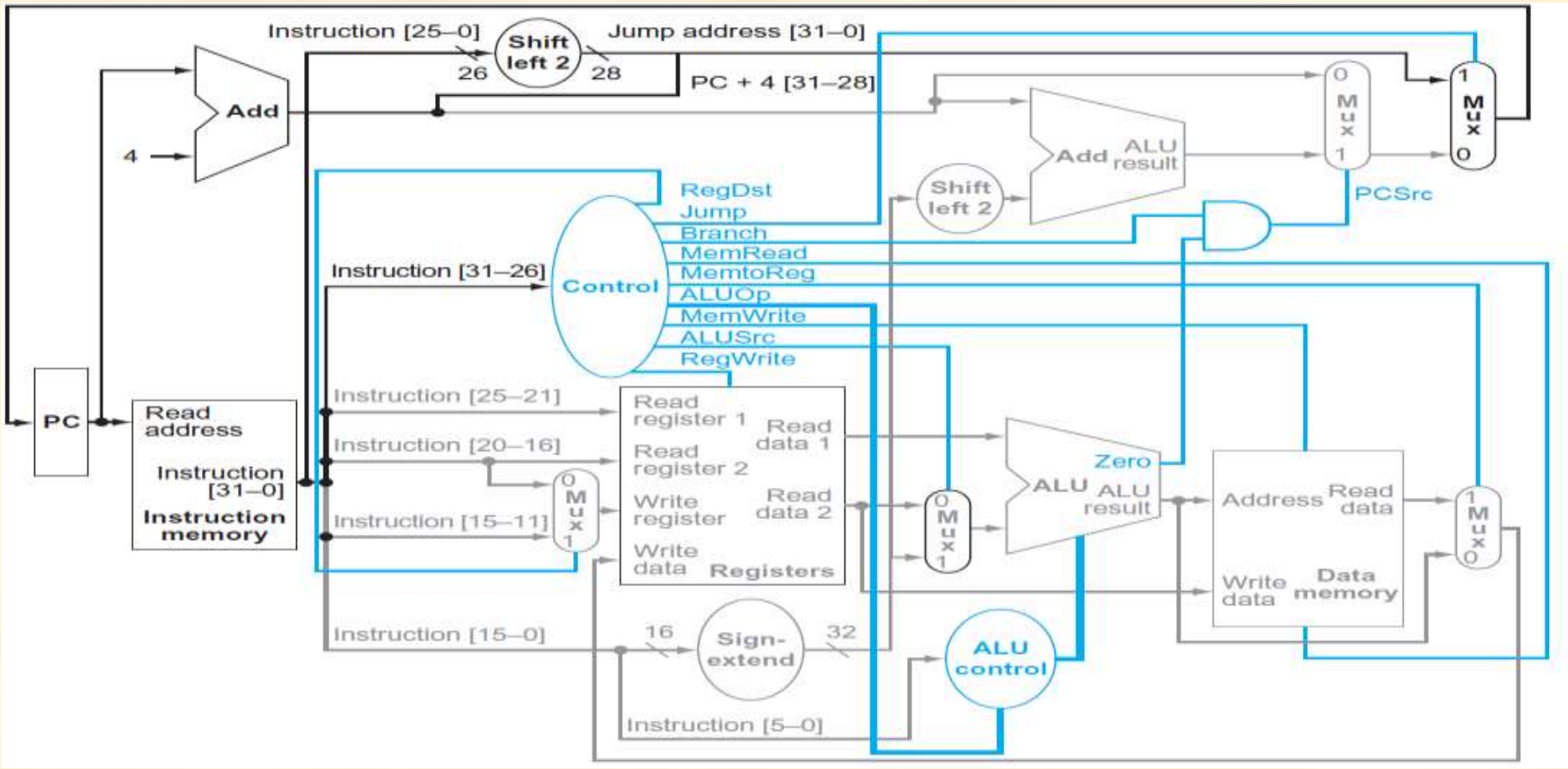


Exercício

5. Ilustre sobre a figura completa do MIPS todos os caminhos de dados e sinais de controle necessários para o funcionamento da instrução addi (soma imediata)

`addi rt, rs, operando:` `rt = rs + operando`

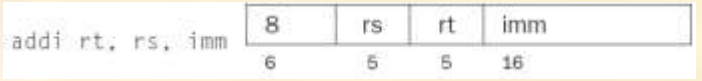


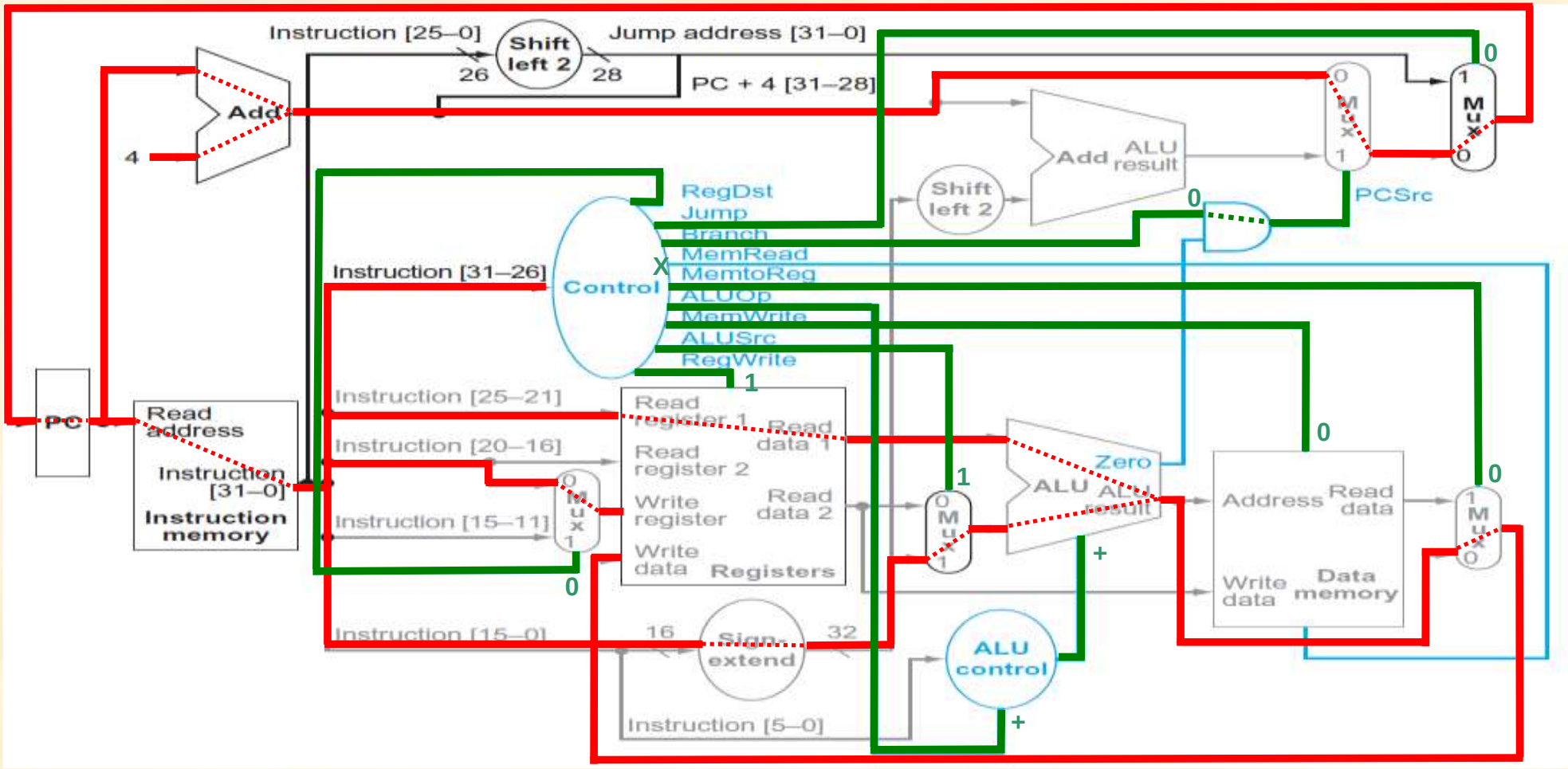


Resposta

5. Ilustre sobre a figura completa do MIPS todos os caminhos de dados e sinais de controle necessários para o funcionamento da instrução addi (soma imediata)

`addi rt, rs, operando:` `rt = rs + operando`



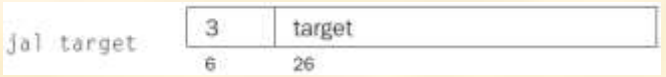


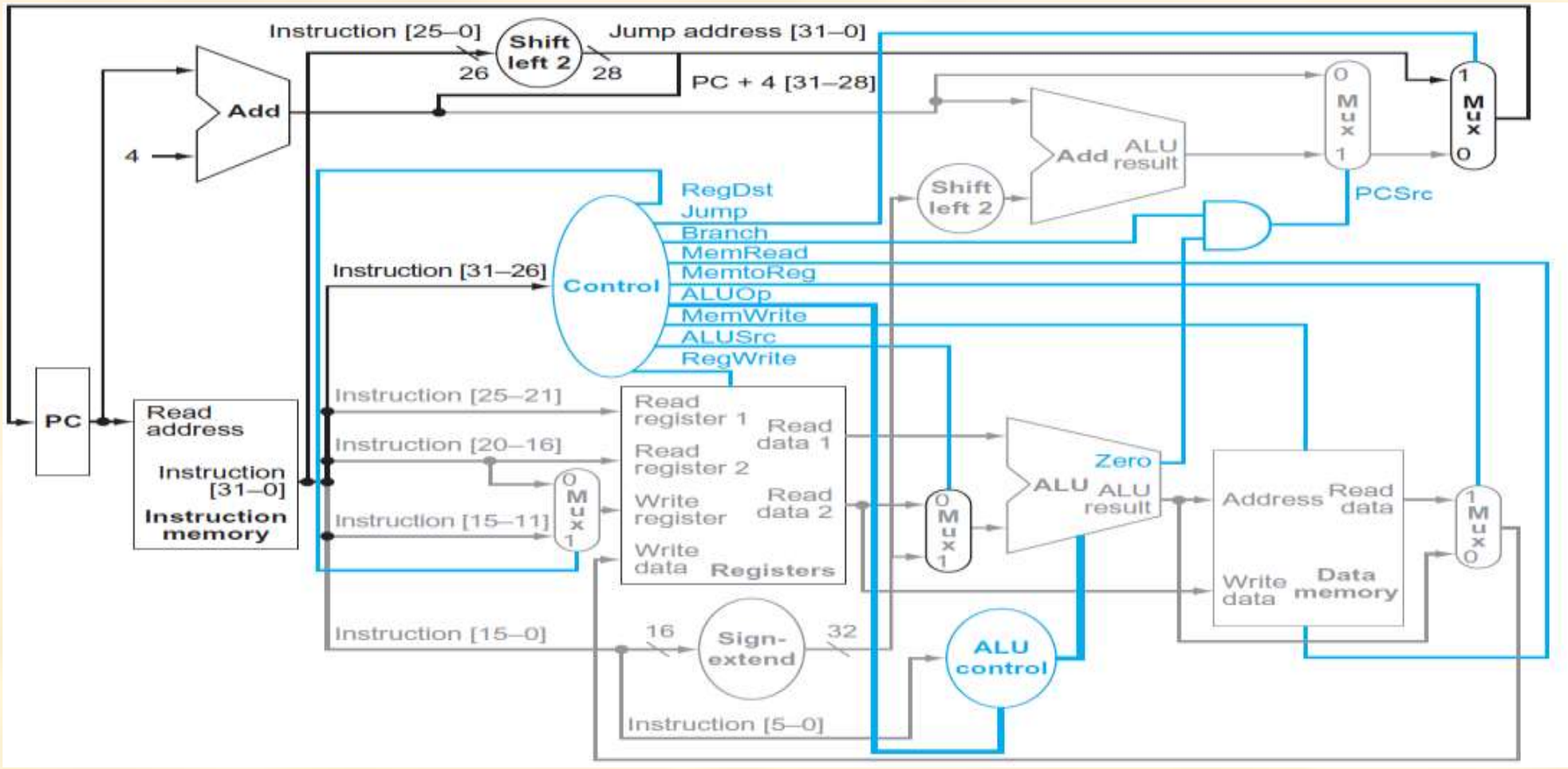
Exercício

6. Semelhante ao Exercício 5, mas a instrução a ser acrescentada é a jal (jump and link)

`jal endereço:` `$pc = endereço`

`$ra = $pc + 4`

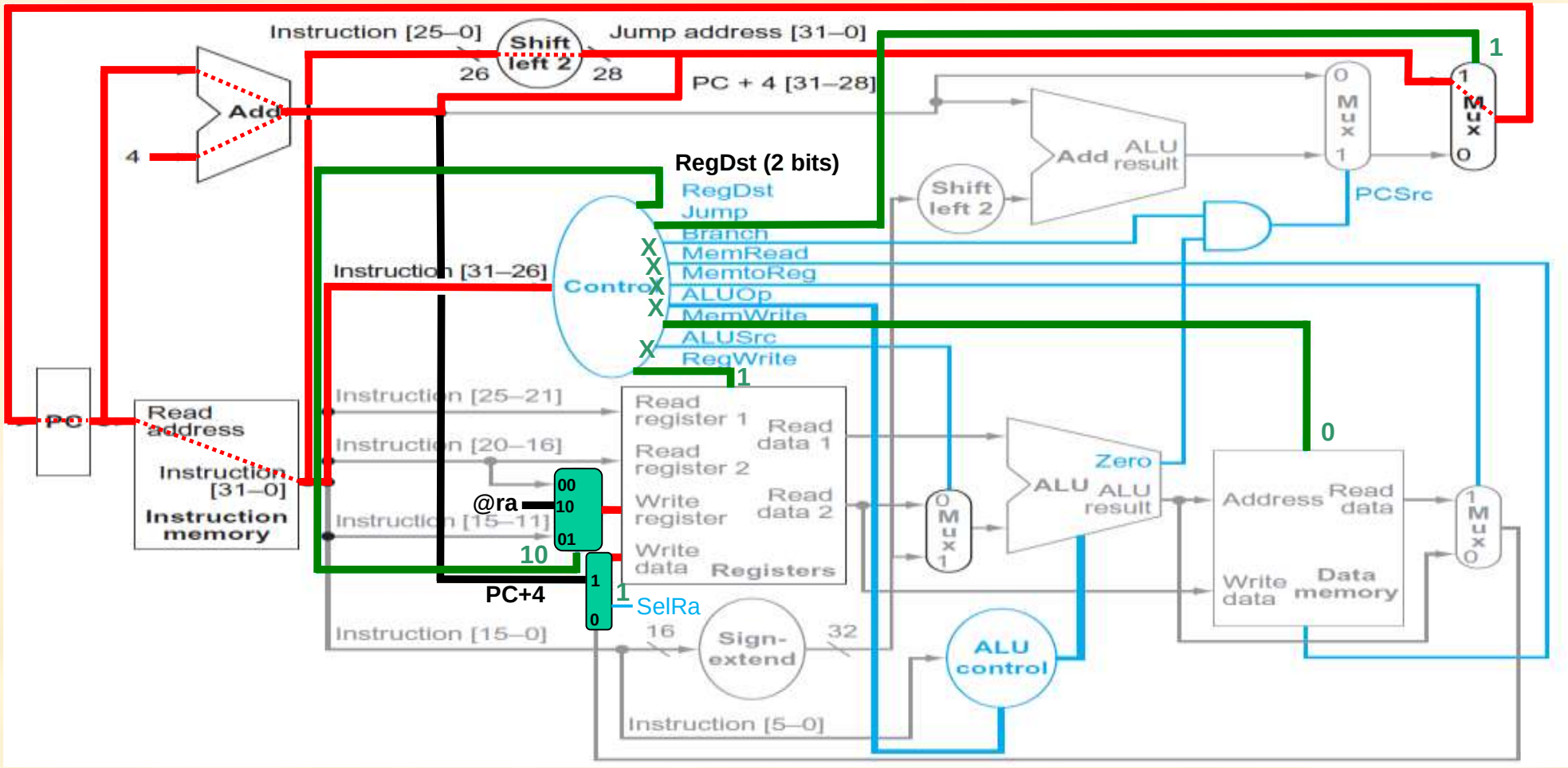




Resposta

6. Semelhante ao Exercício 5, mas a instrução a ser acrescentada é a jal (jump and link)

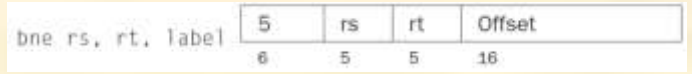
```
jal endereço:    $pc = endereço
                 $ra = $pc + 4
```

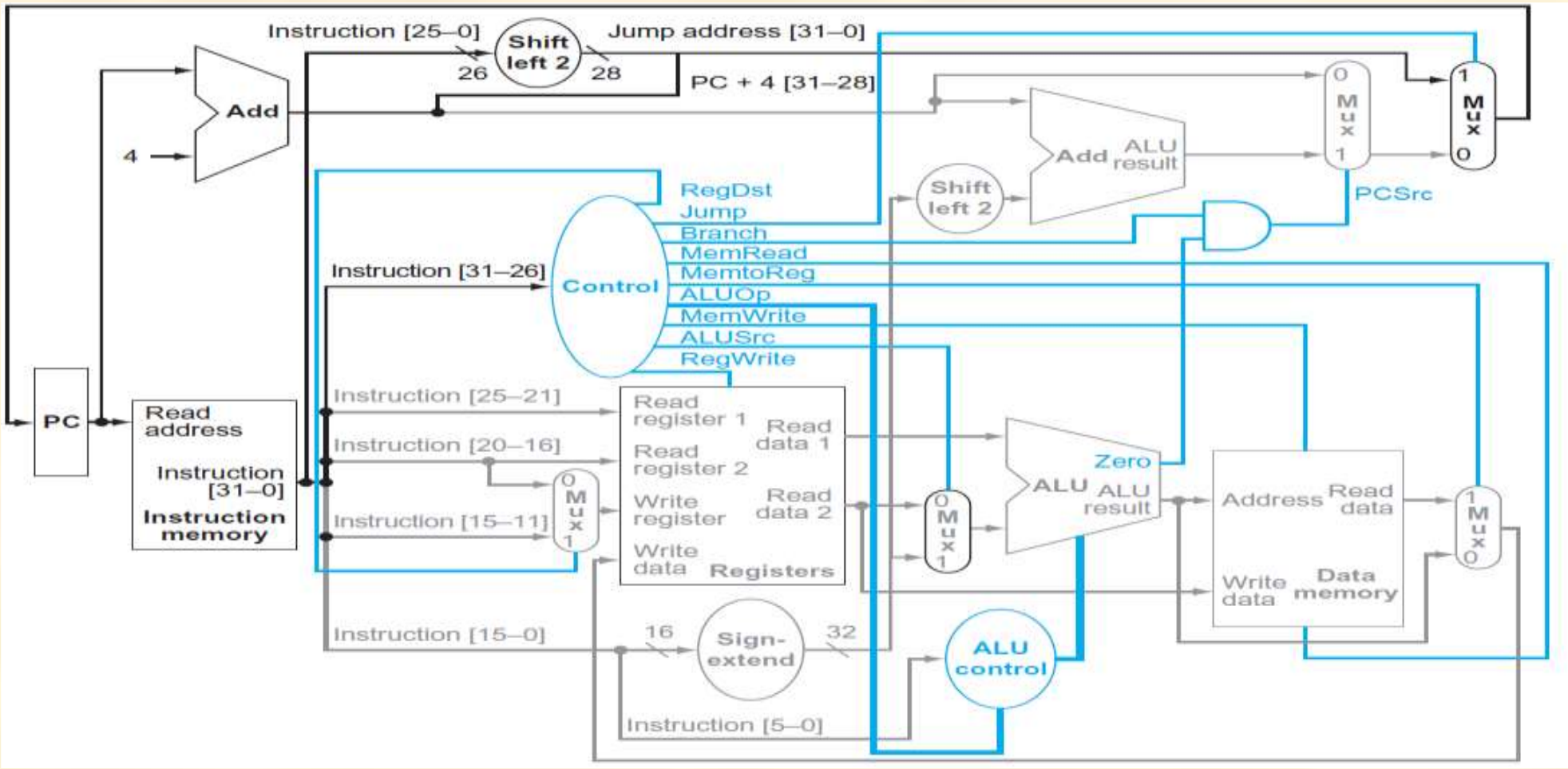


Exercício

7. Semelhante ao Exercício 5, agora para a instrução bne (branch if not equal)

```
bne rs, rt, label:   se(rs != rt) $pc = label + $pc + 4
                     senão        $pc = $pc + 4
```



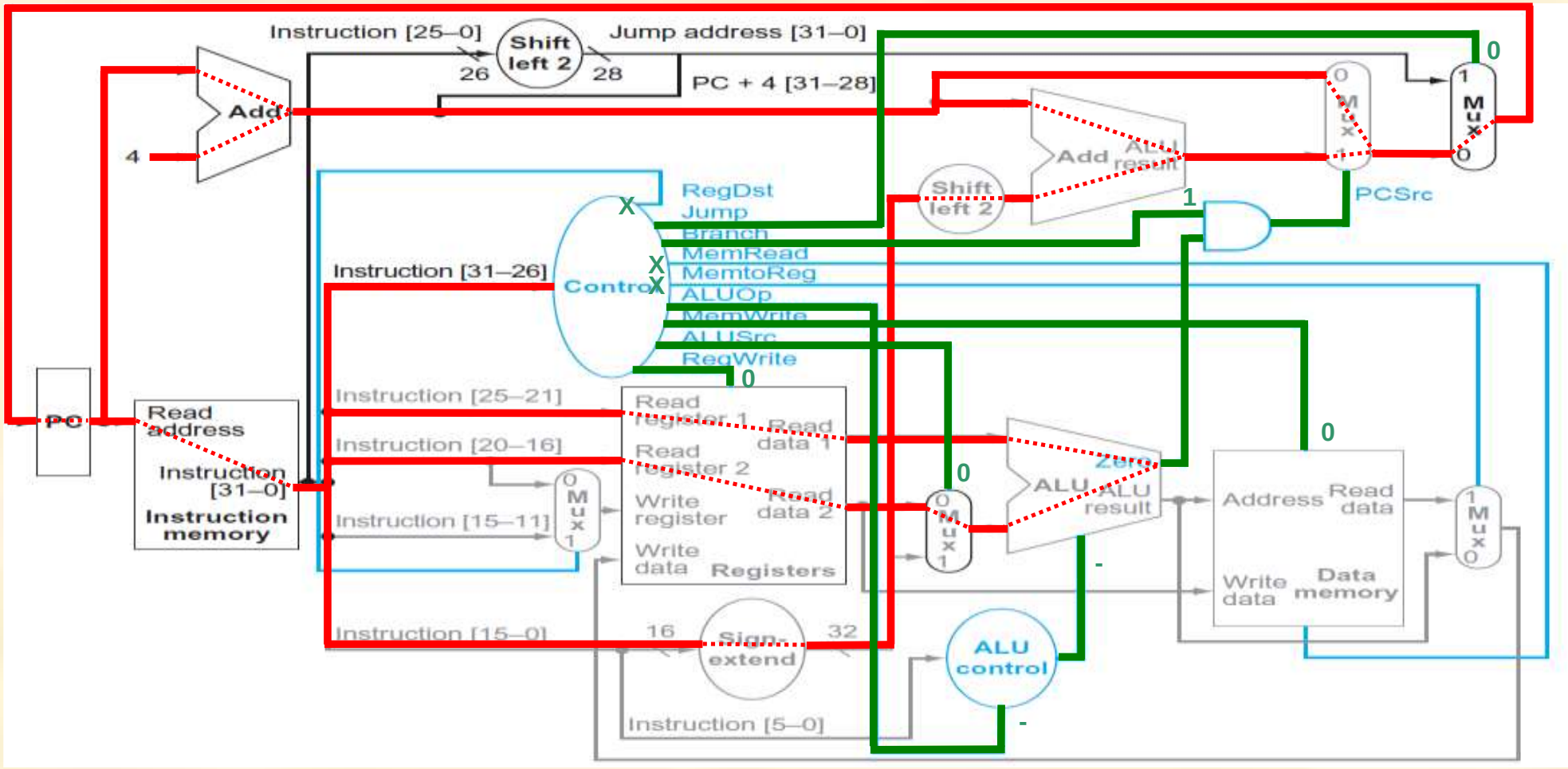


Resposta

7. Semelhante ao Exercício 5, agora para a instrução bne (branch if not equal)

```
bne rs, rt, label:   se(rs != rt)  $pc = label + $pc + 4
                     senão         $pc = $pc + 4
```

bne rs, rt, label	5	rs	rt	Offset
	6	5	5	16



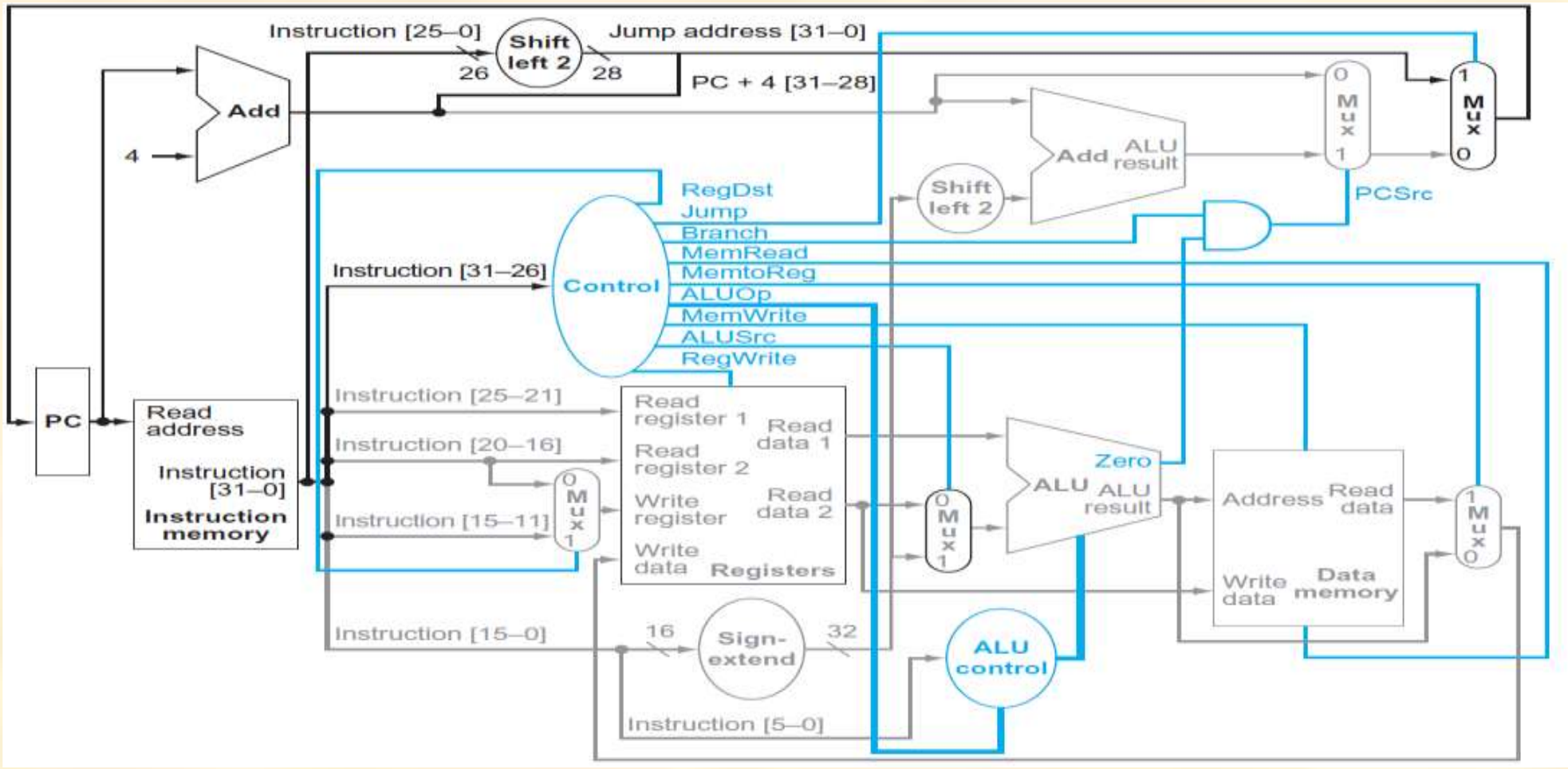
Exercício

8. Semelhante ao Exercício 5, mas a instrução a ser acrescentada é uma variante da lw (load word)

`lw rd, rs, rt:` `rd = M[rs + rt]`

lw rd, rt(rs)

??	rs	rt	rd	0	??
6	5	5	5	5	6



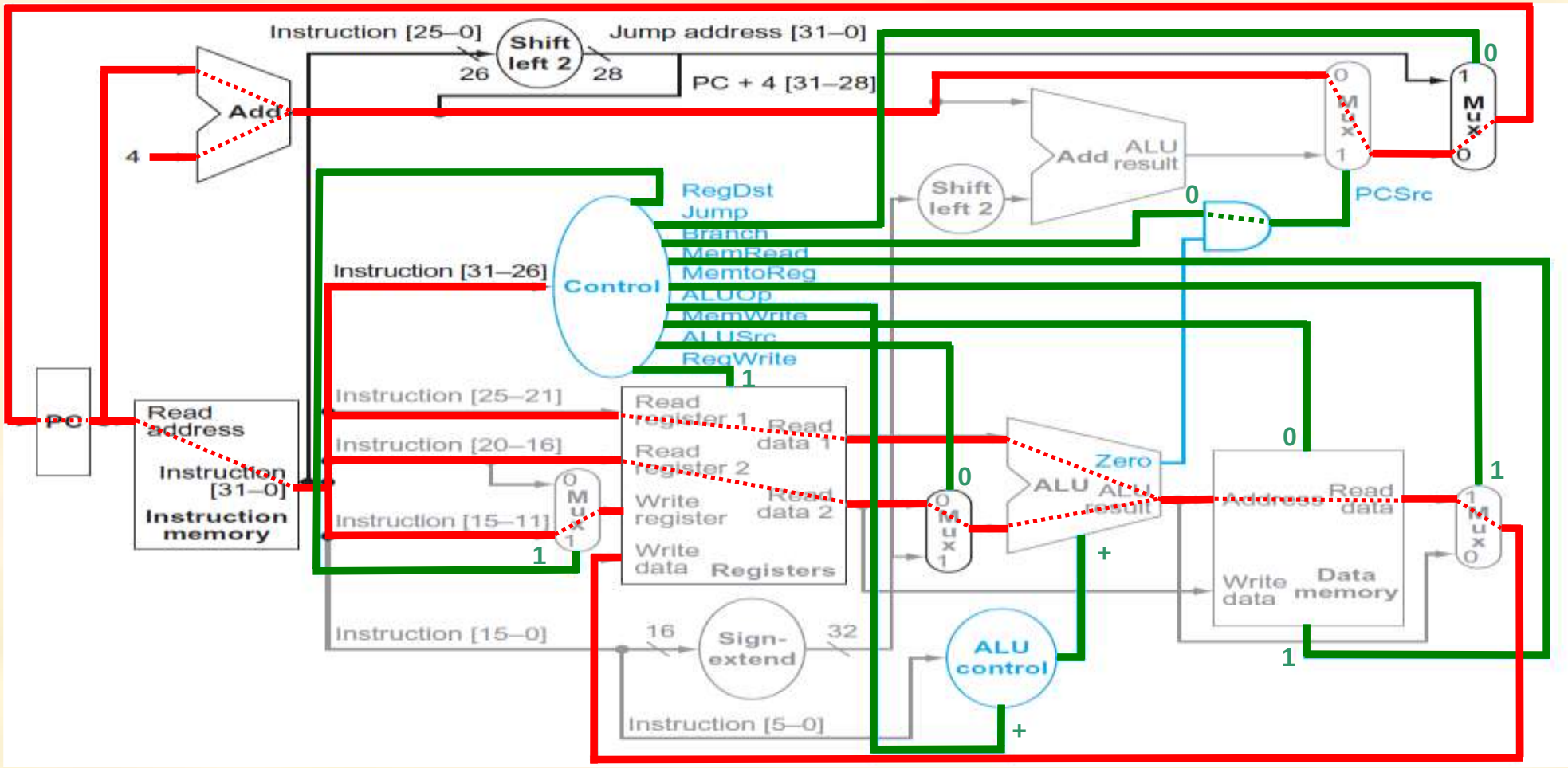
Resposta

8. Semelhante ao Exercício 5, mas a instrução a ser acrescentada é uma variante da lw (load word)

`lw rd, rs, rt:` `rd = M[rs + rt]`

lw rd, rt(rs)

??	rs	rt	rd	0	??
6	5	5	5	5	6



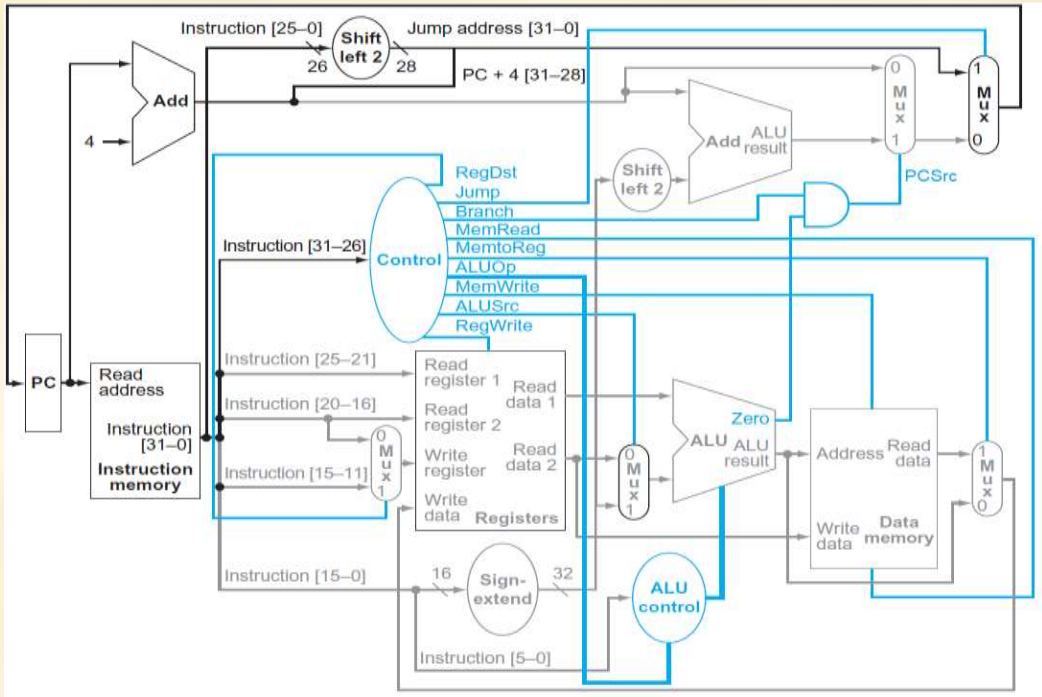
Exercício

9. Dada a instrução swap descrita abaixo

```
swap rs, rt:      rs = rt
                  rt = rs
```

a) Explique porque não é possível implementar a mesma sem modificar o banco de registradores

b) Implemente a pseudo-instrução swap com instruções da arquitetura alvo



Resposta

9. Dada a instrução swap descrita abaixo

```
swap rs, rt:      rs = rt
                  rt = rs
```

a) Explique porque não é possível implementar a mesma sem modificar o banco de registradores

A instrução swap requer duas leituras de registrador e duas escritas simultâneas; contudo, o banco de registradores somente aceita uma escrita por ciclo de relógio!

b) Implemente a pseudo-instrução swap com instruções da arquitetura alvo

```
add $at, $ZERO, $rt
add $rt, $ZERO, $rs
add $rs, $ZERO, $at
```