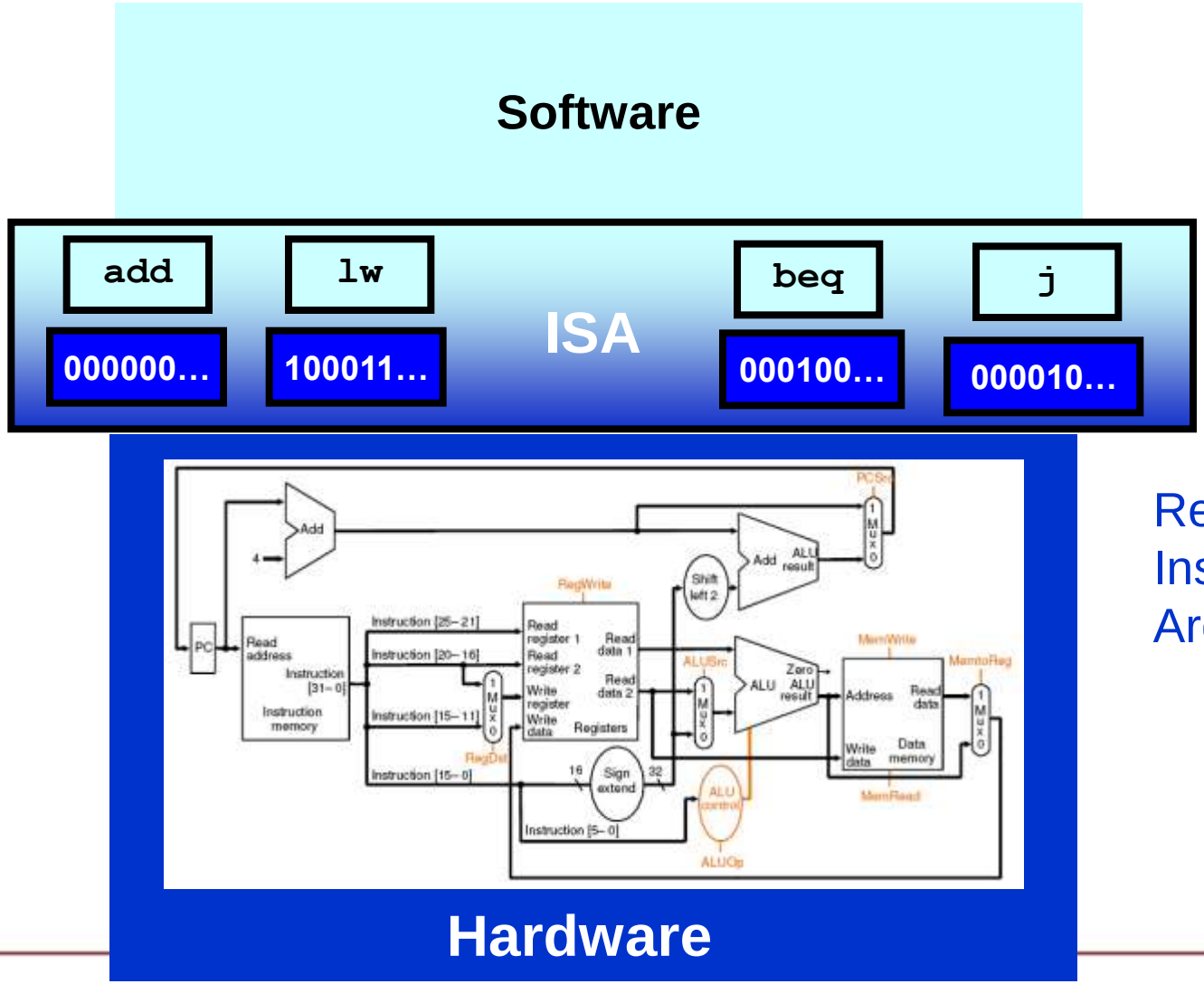
Infraestrutura de Hardware

Implementação Multiciclo de um Processador Simples

Perguntas que Devem ser Respondidas ao Final do Curso

- Como um programa escrito em uma linguagem de alto nível é entendido e executado pelo HW?
- **Qual é a interface entre SW e HW e como o SW instrui o HW a executar o que foi planejado?**
- O que determina o desempenho de um programa e como ele pode ser melhorado?
- Que técnicas um projetista de HW pode utilizar para melhorar o desempenho?

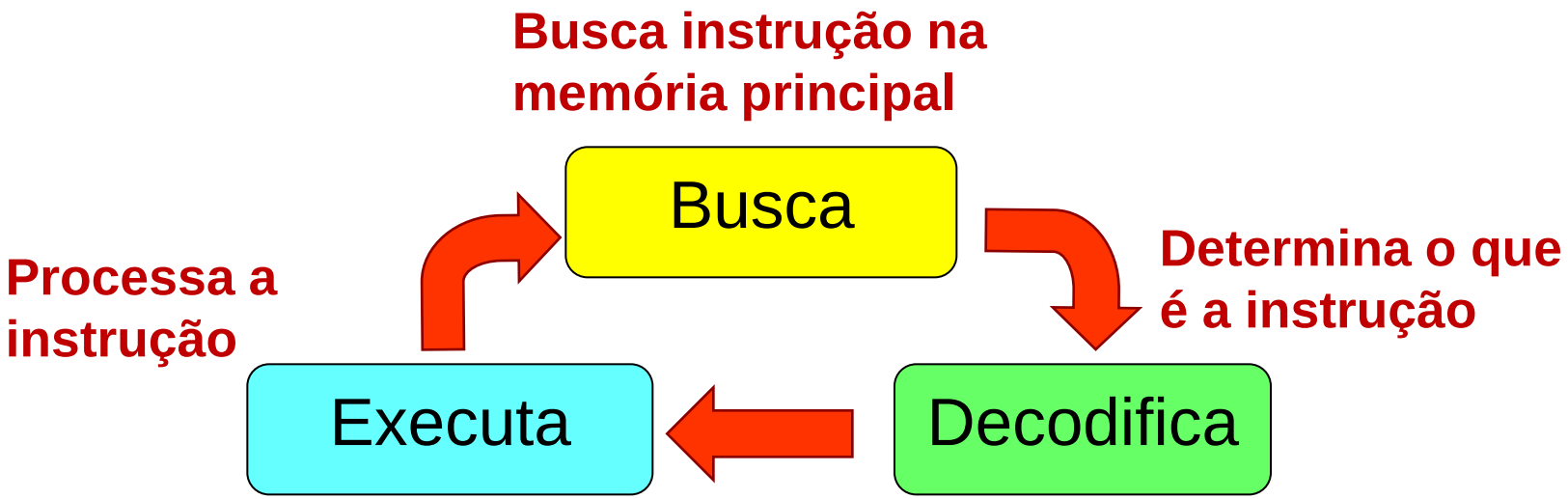
Interface HW/SW: ISA



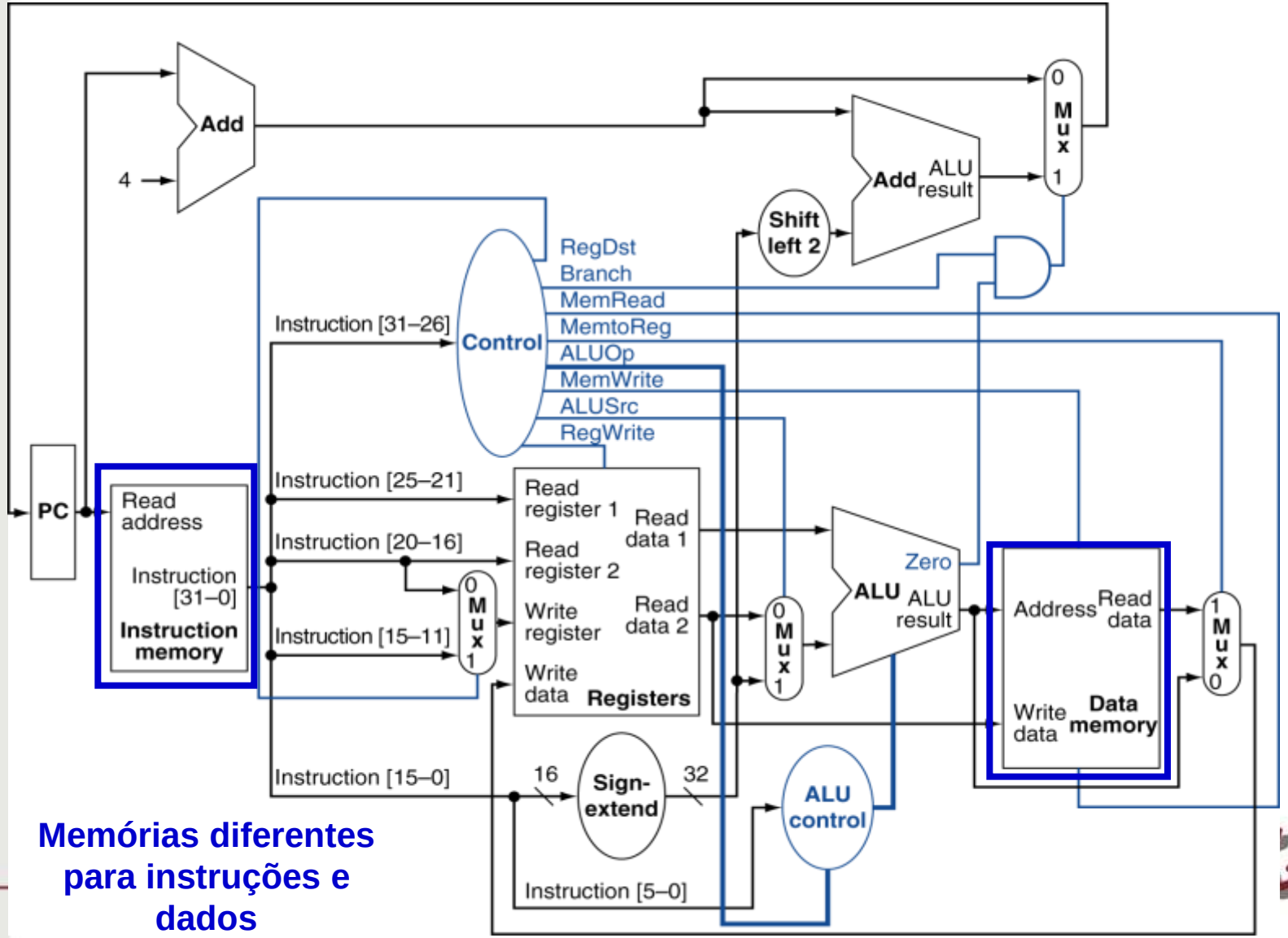
Repertório de Instruções da Arquitetura

Visão Simplificada de Processamento de Instrução

- CPU faz continuamente 3 ações:



Implementação Monociclo de CPU Simples



Memórias diferentes para instruções e dados

Centro de Informática

Análise de Implementação Monociclo

- Vantagem:

  Simplicidade de implementação

  - Datapath e Controle

- Desvantagens:

  Custo de hardware

  - Duplicação de componentes devido a restrição de um ciclo para o processamento de instrução

  Desempenho

  - Ciclo de clock longo para comportar instrução mais lenta
  - Instruções que demandam menos tempo deixam CPU ociosa
  - Implementação pouco eficiente

Como Melhorar?

- **Implementação Multiciclo!**

  Dividir processamento de instrução em diferentes estágios

  Cada estágio é executado em um ciclo de clock

  - Ciclo pode ser menor

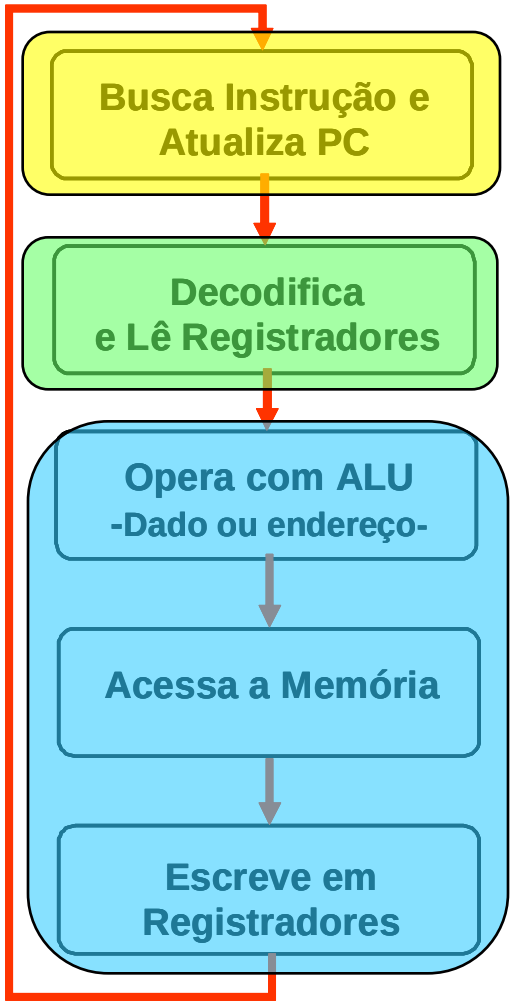
  Diferentes instruções requerem quantidade de estágios diferentes

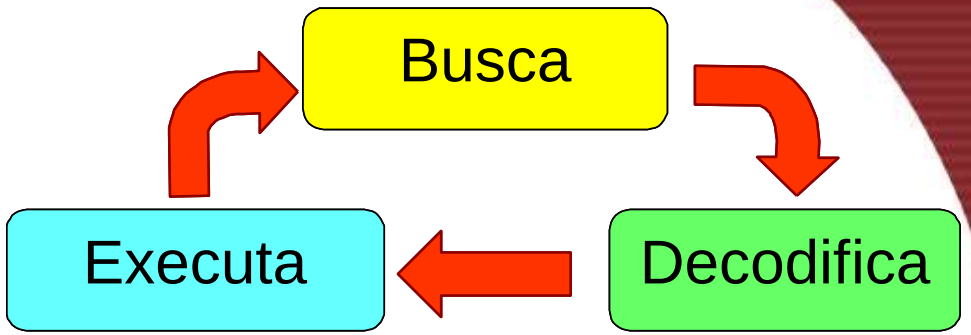
  - Algumas instruções podem ser processadas em menos tempo
  - Tempo médio de processamento pode melhorar

  Mesmo componente de HW pode ser utilizado em estágios diferentes

Uso mais eficiente de HW!

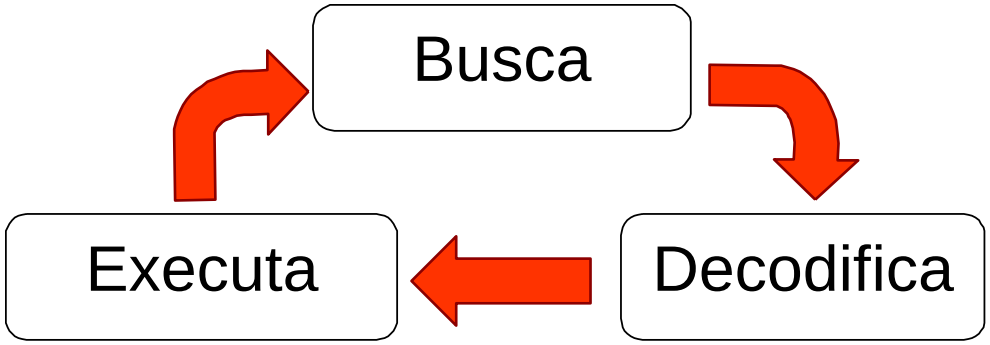
Quebrando Processamento de Instrução em Estágios

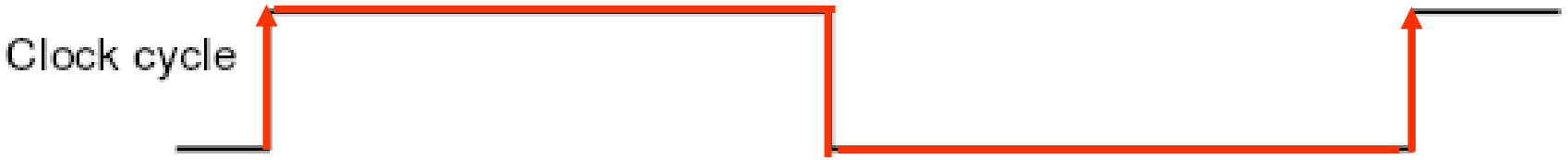




- Dois primeiros estágios sempre executados
- Dependendo da instrução, alguns dos outros estágios podem não ser executados

Implementação Monociclo





Busca – Decodifica - Executa

Implementação Multiciclo

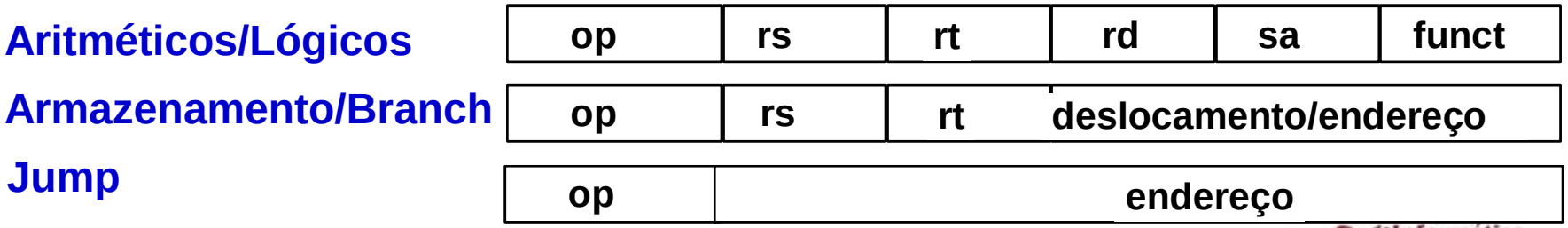
Busca - Decodifica - ALU – Memória – Escrita Reg.

Instruções

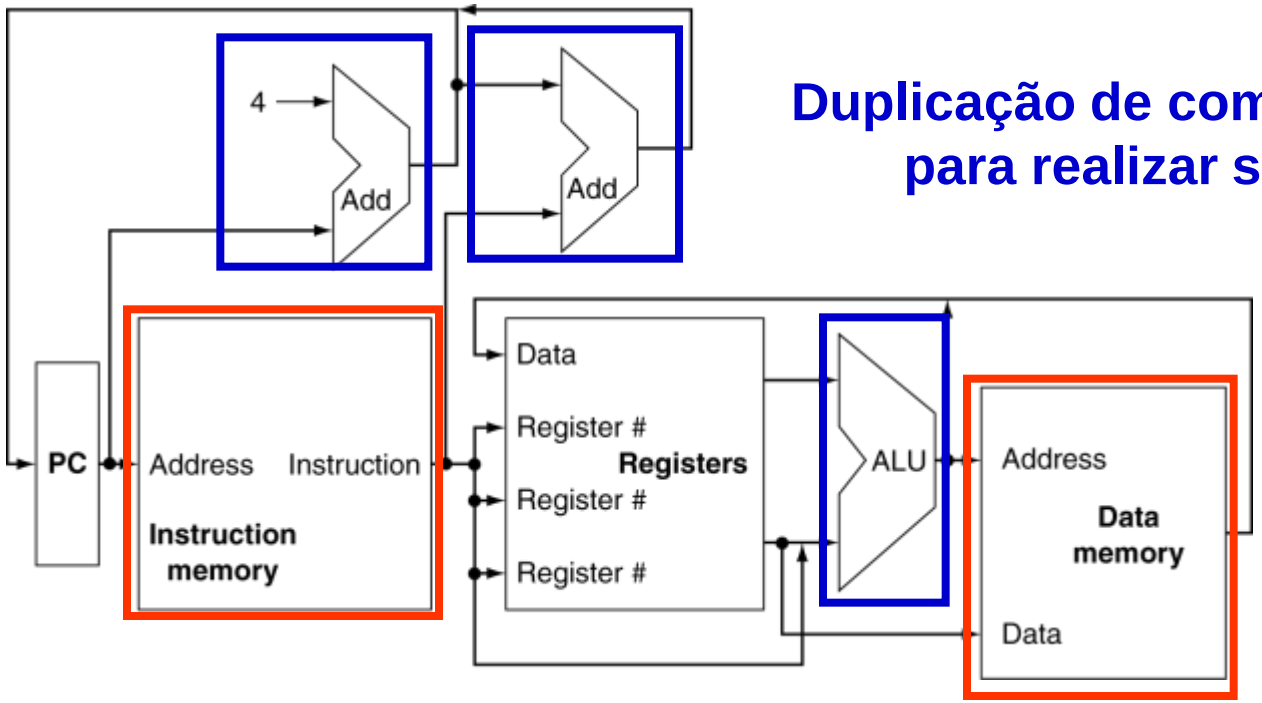
Repertório

Instrução	Descrição
`ADD rd, rs, rt`	rd <- rs + rt
`SUB rd, rs, rt`	rd <- rs - rt
`AND rd, rs, rt`	rd <- rs and rt (bit a bit)
`OR  rd, rs, rt`	rd <- rs or rt (bit a bit)
`SLT rd, rs, rt`	Se rs < rt, rd <- 1, senão rd <- 0
`LW  rt, desl(rs)`	Carrega palavra de mem. em registrador rt
`SW  rt, desl(rs)`	Armazena conteúdo de registrador rt em mem.
`BEQ rs, rt,end`	Desvio para end, se rs == rt
`J end`	Desvio para end

Formato



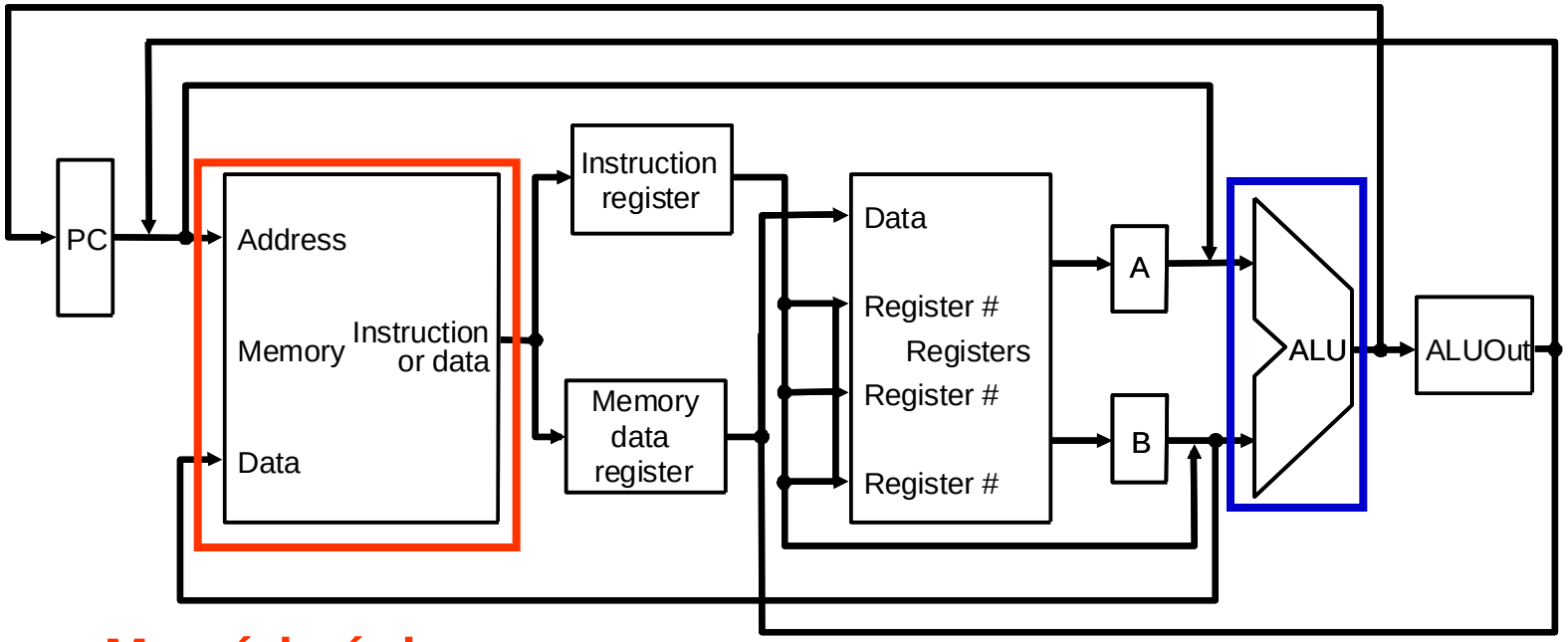
Visão Abstrata de Implementação Monociclo



Duplicação de componentes para realizar somas

Memórias diferentes para instruções e dados

Visão Abstrata de Implementação Multiciclo



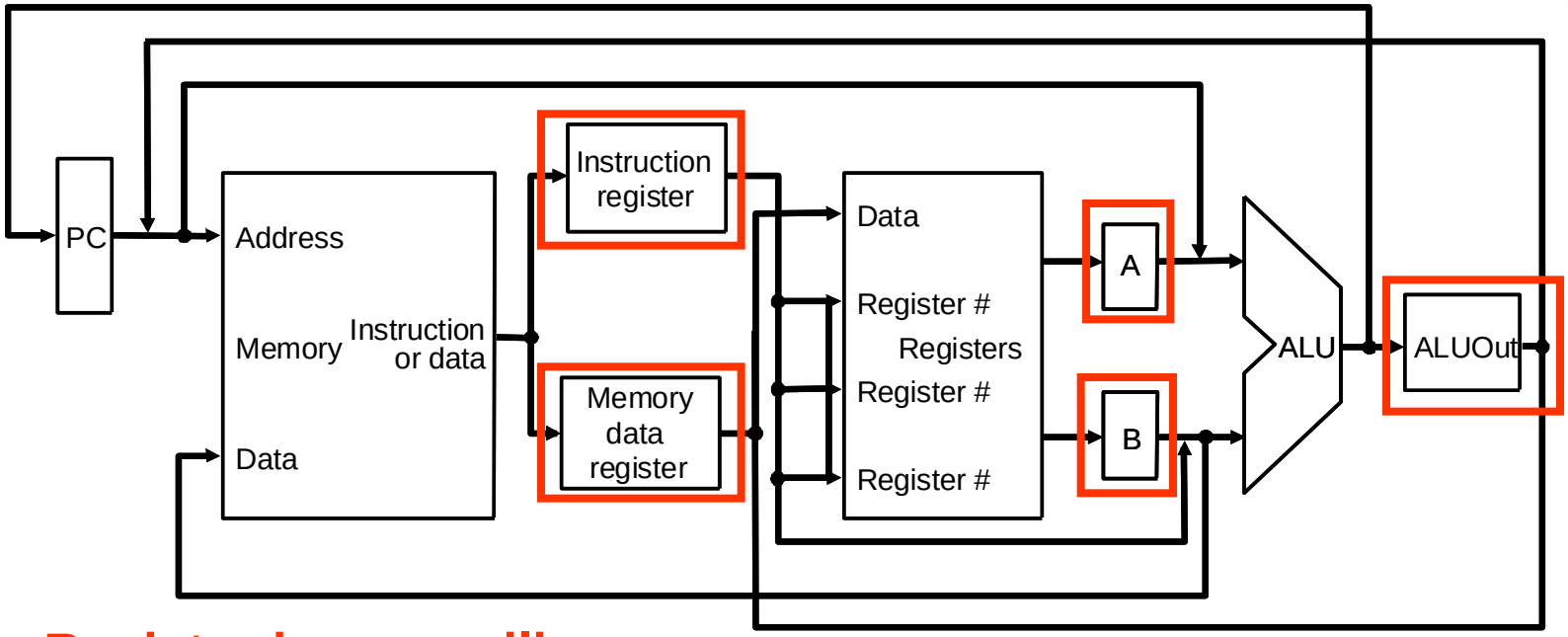
Memória única para instruções e dados

Componente único para realizar operações

- Compartilhamento de recursos

  Utilizados em estágios diferentes (ciclos de clock diferentes)

Mais Registradores…



Registradores auxiliares entre unidades funcionais maiores

- Valores que serão usados na mesma instrução, mas em ciclo de clock diferente devem ser armazenados nestes registradores

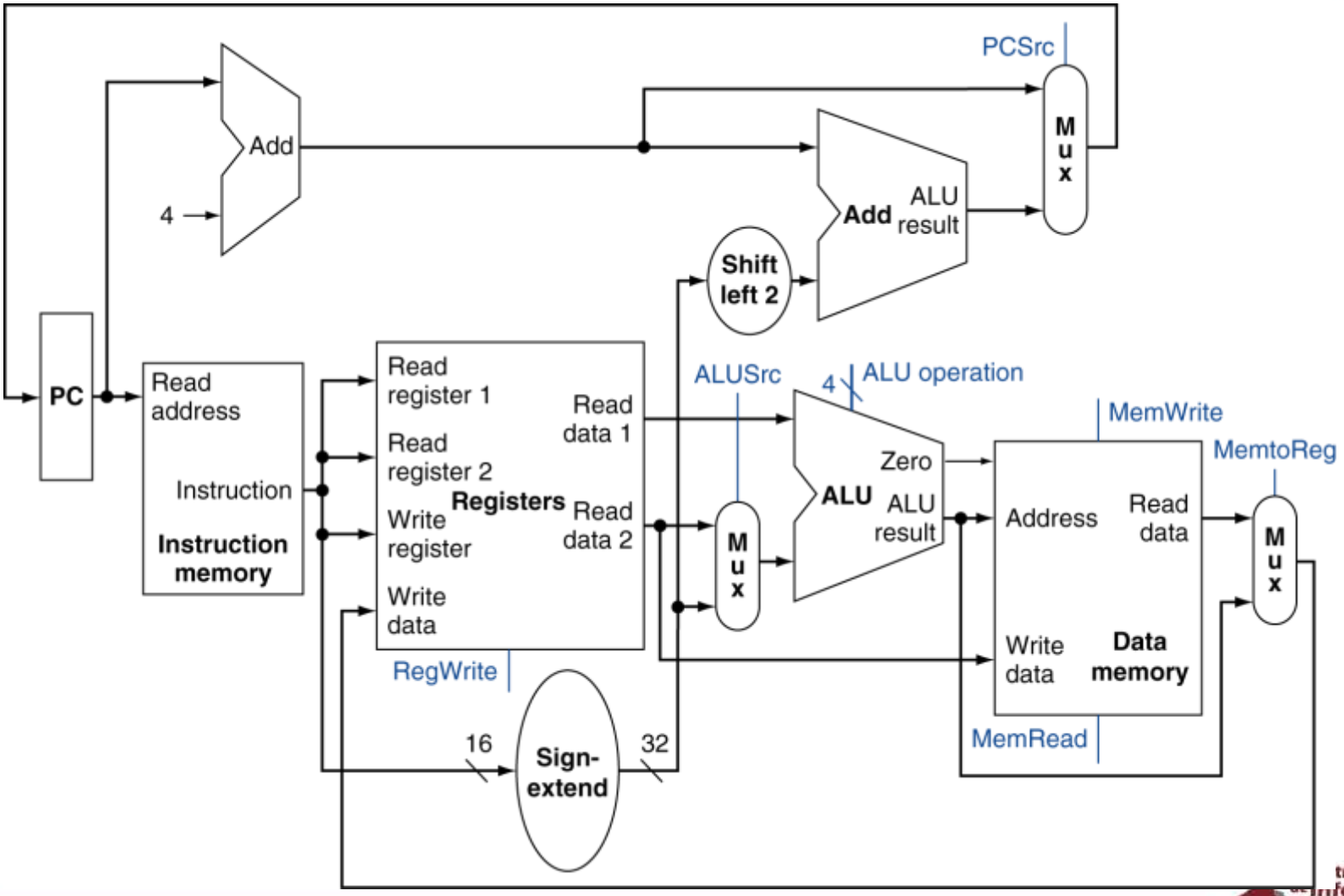
Processamento na Implementação Multiciclo

- Um ciclo de clock pode acomodar uma das operações:
  - Acesso de memória
  - Acesso ao banco de registradores (2 leituras ou 1 escrita)
  - Operação da ALU
- Unidades funcionais só podem ser utilizados uma vez durante um ciclo
  - ALU, memória, banco de registradores
  - Exceção: registradores auxiliares
    - IR – Armazena instrução lida da memória
    - MDR – Armazena dado lido da memória
    - A e B – Armazenam operandos lidos do banco de registradores
    - ALUOut – Armazena saída da ALU

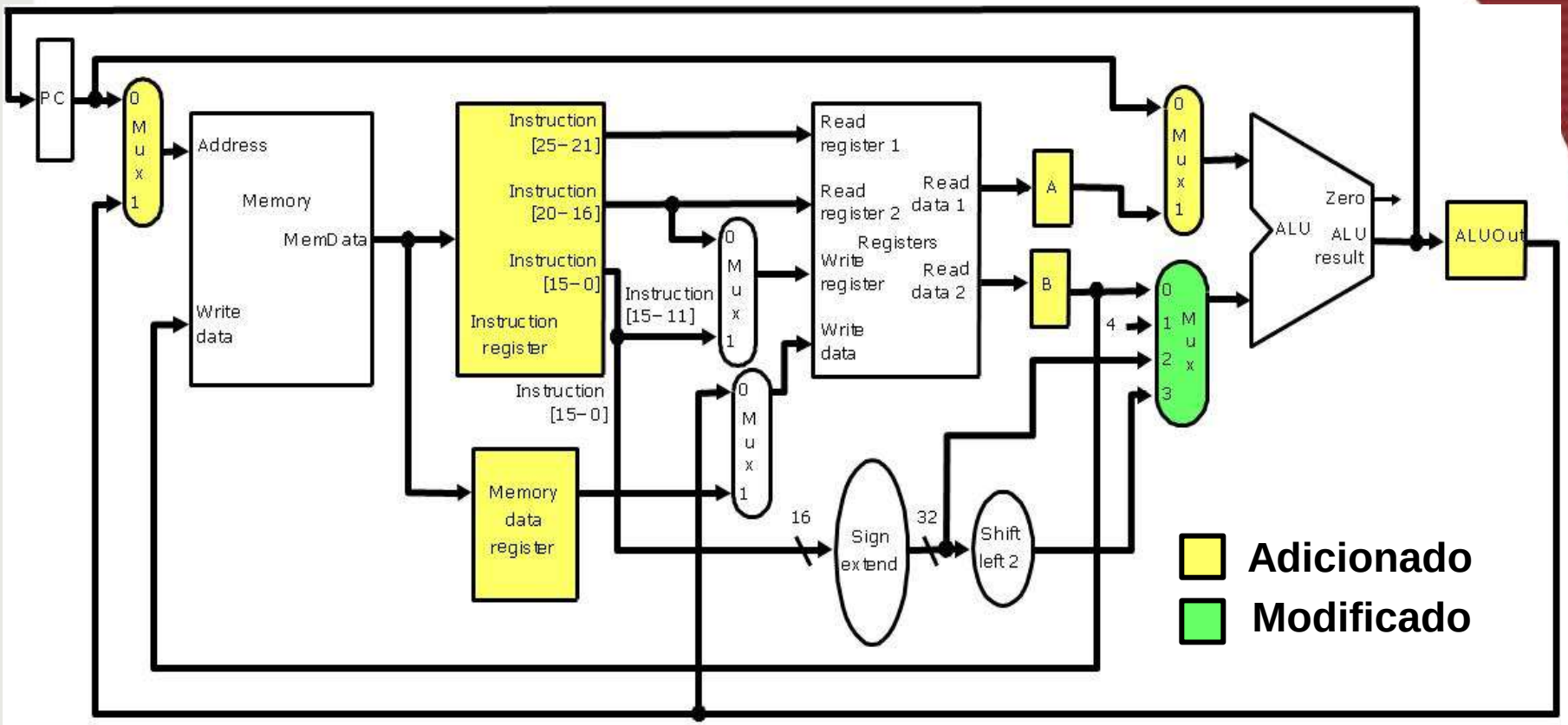
Processamento na Implementação Multiciclo

- Dados de uma instrução necessários para outra instrução são armazenados nas unidades visíveis ao programador
  - Memória
  - Banco de registradores
  - PC
- Dados necessários entre estágios (ciclos) diferentes de uma mesma instrução são armazenados nos registradores auxiliares

Unidade de Processamento Monociclo



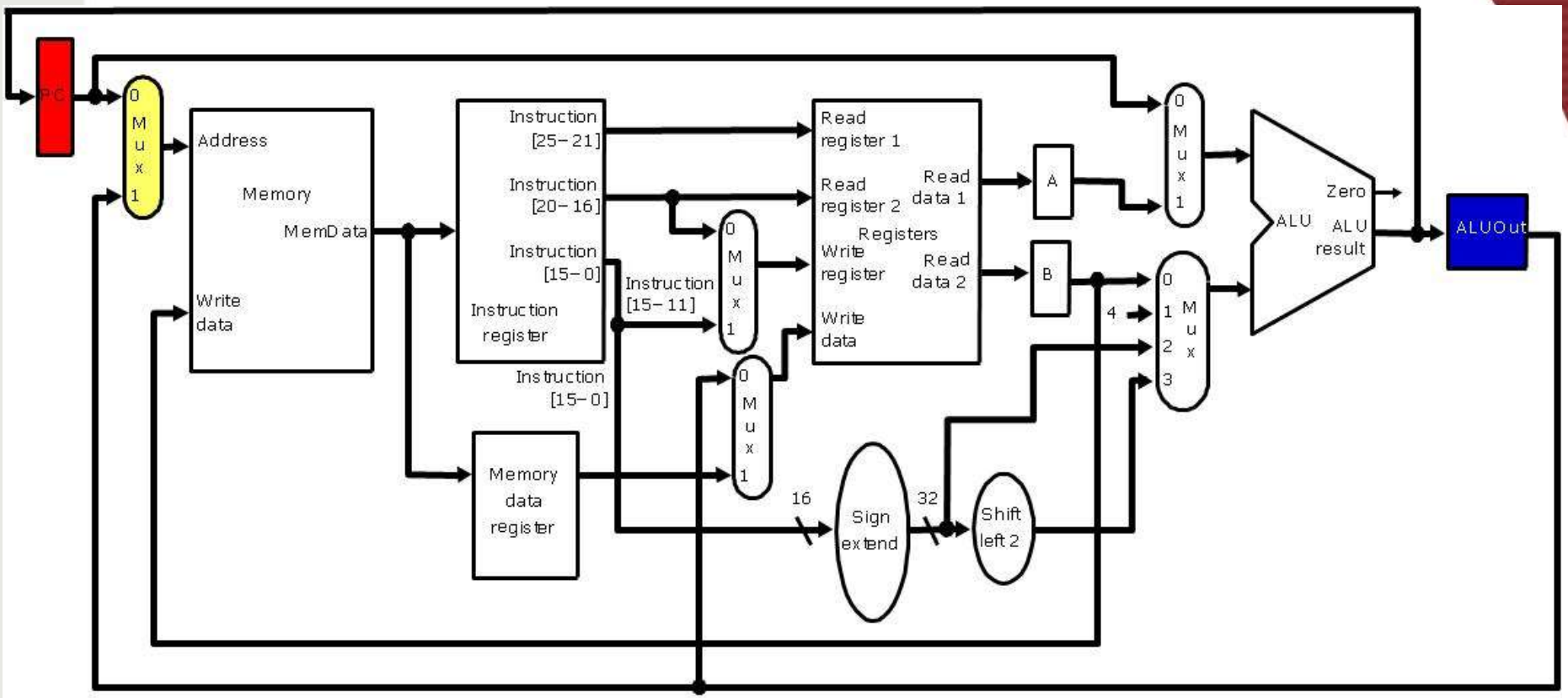
Unidade de Processamento Multiciclo



- Compartilhamento de recursos

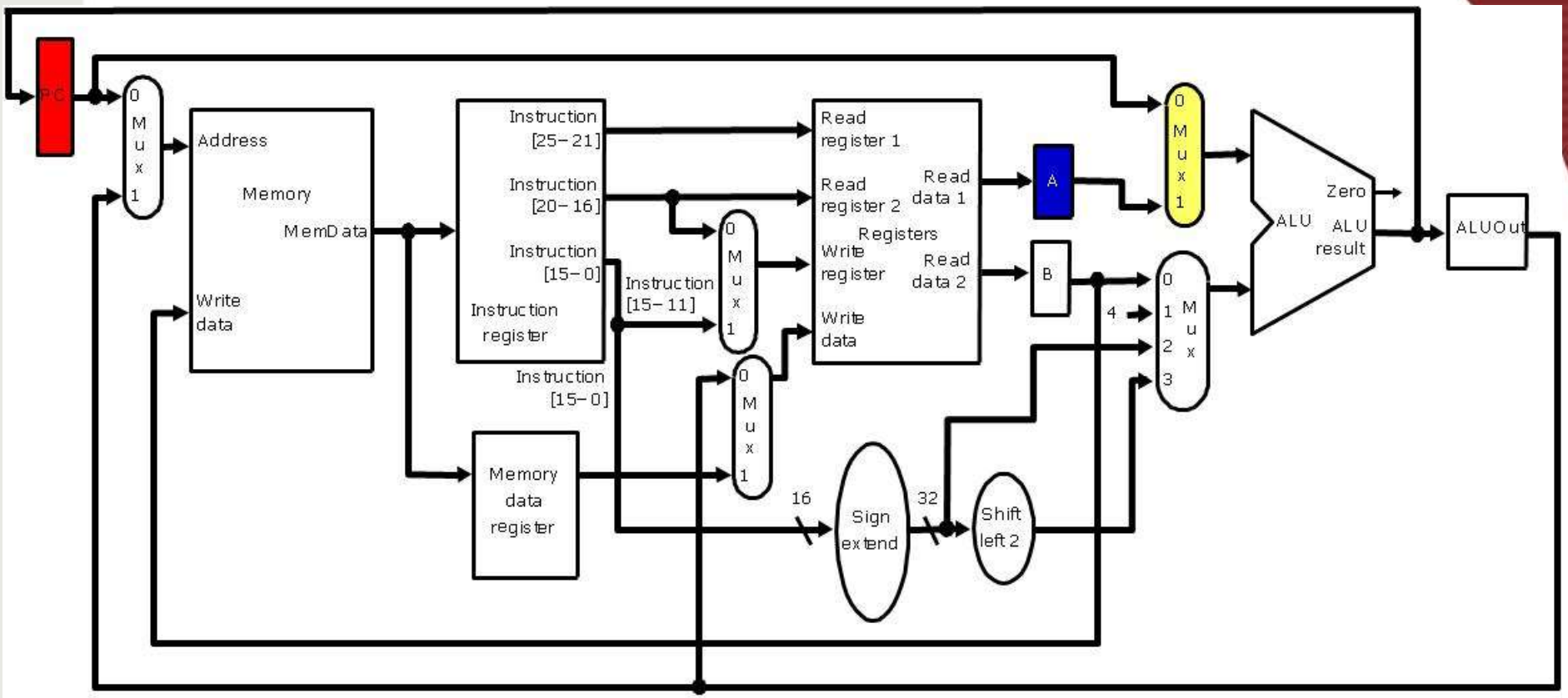
Mais multiplexadores ou multiplexadores expandidos

Entrada de Endereço da Memória



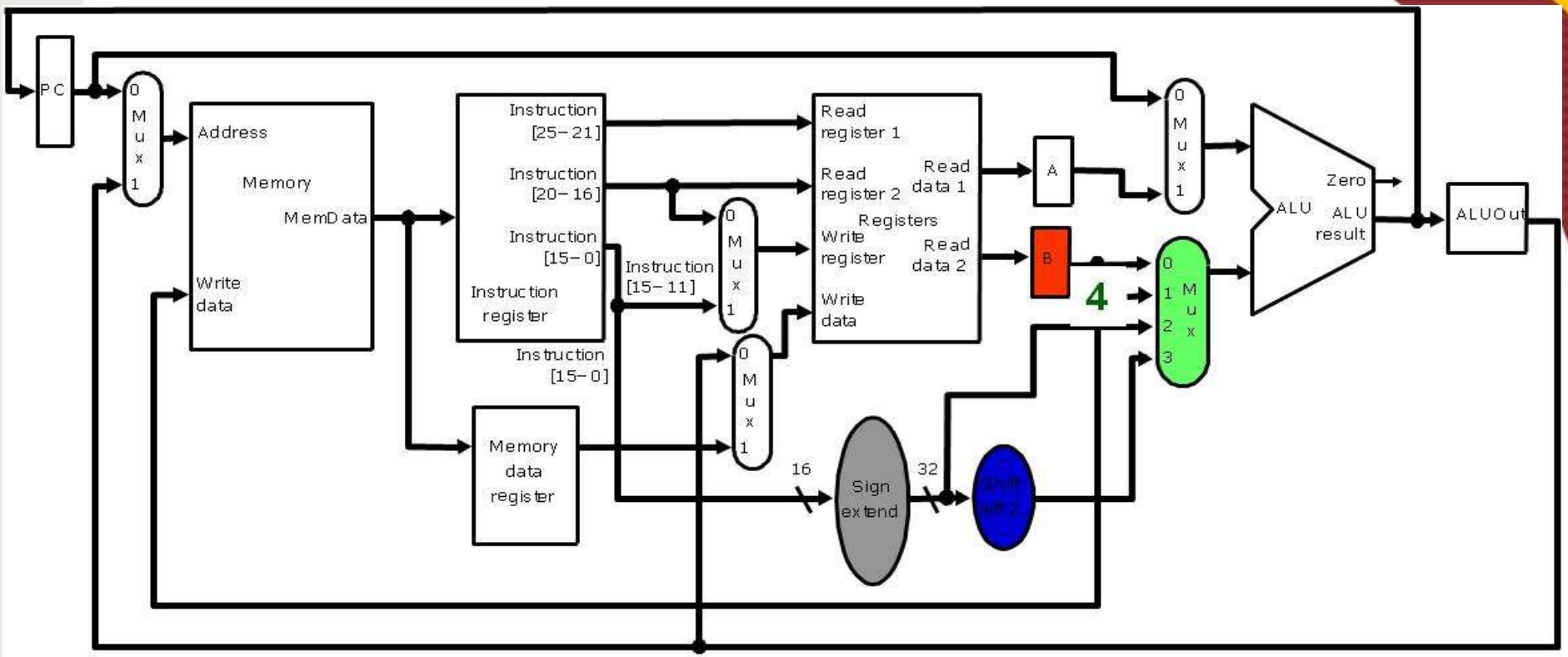
- Multiplexador para escolha entre endereço de instrução (PC) ou endereço de dado (saída da ALU - ALUOut)

Primeiro Operando da ALU



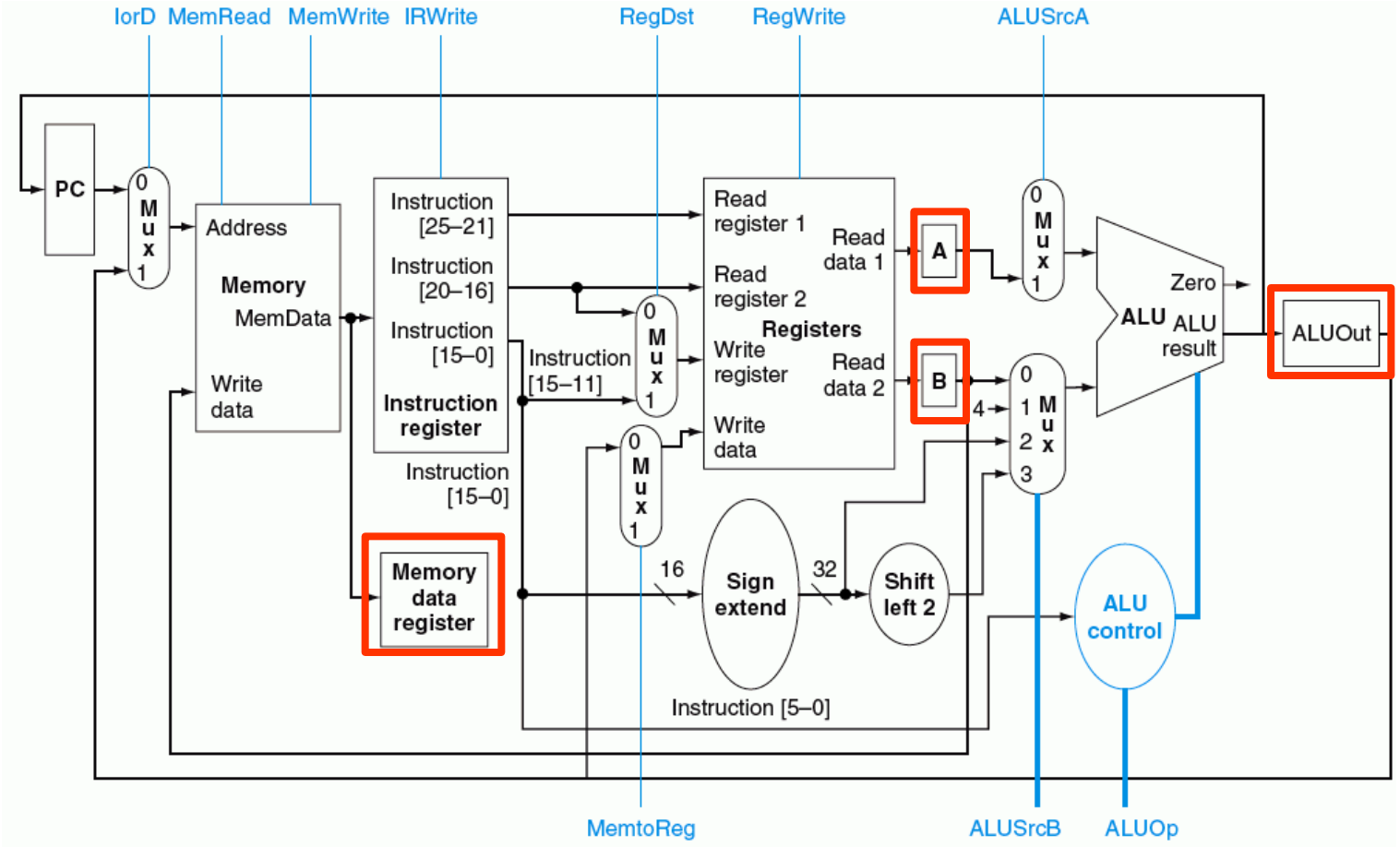
- Multiplexador para escolha entre PC ou operando da saída do banco de registradores (registrador A)

Segundo Operando da ALU



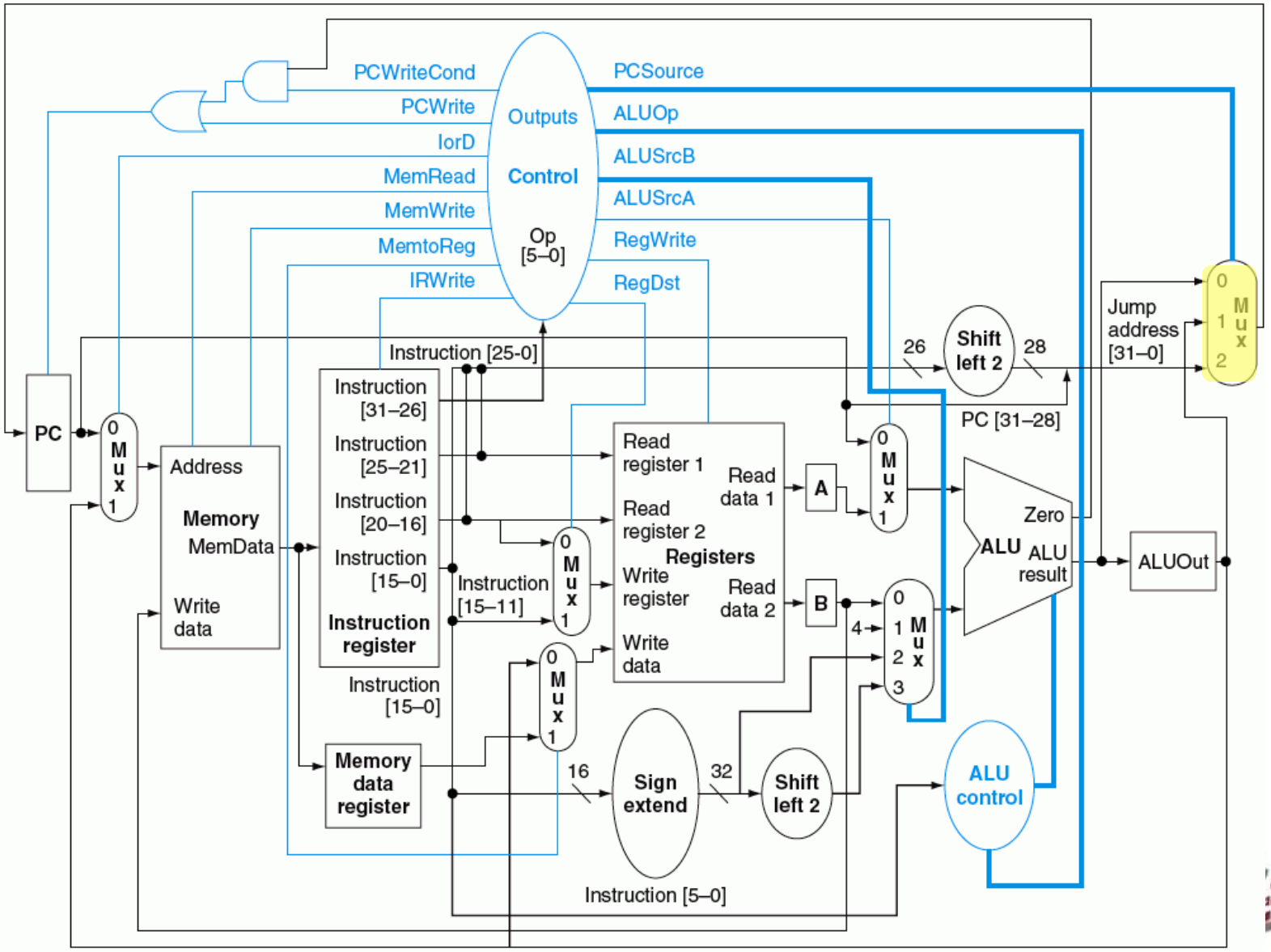
- Multiplexador expandido para escolha entre saída do banco de registradores (B), a constante 4 (para incrementar PC), o sinal estendido e deslocado de 2 bits (para beq) e o sinal simplesmente estendido (para lw/sw)

Unidade de Processamento com Sinais de Controle

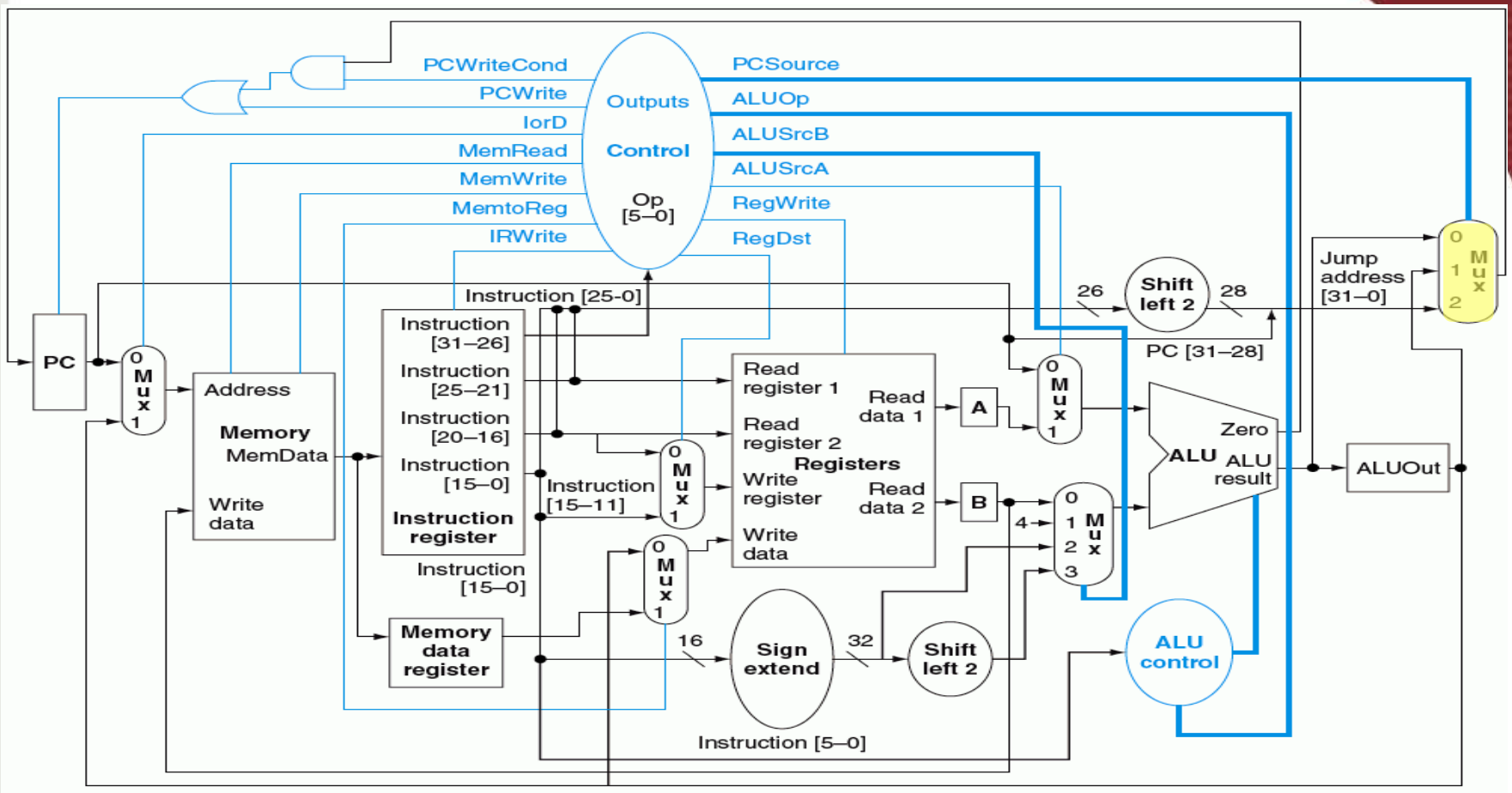


Atualizados em todo ciclo de clock, não precisam de sinal de controle de escrita

Unidade de Processamento com Controle



Atualização do PC



- Multiplexador para escolha entre 3 fontes de atualização: PC + 4, jump, branch

Sinais de 1 bit da Unidade de Controle

Nome	Efeito quando 0	Efeito quando 1
RegDst	Número do registrador destino para escrita vem de rt	Número do registrador destino para escrita vem de rd
RegWrite	Nada	Registrador da entrada “Write register” recebe valor da entrada “Write data”
ALUSrcA	1° operando da ALU vem do PC	1° operando da ALU vem do registrador A
IRWrite	Nada	Saída da memória é escrita em IR
MemRead	Nada	Dado da memória relativo ao endereço especificado é colocado na saída
MemWrite	Nada	Dado da memória do endereço especificado é substituido pelo que tem no “Write data”
MemTo Reg	Valor escrito na entrada “Write data” dos registradores vem da ALUOut	Valor escrito na entrada “Write data” vem de MDR
PCWrite	Nada	Escrita em PC, controlado por PCSource
PCWriteCond	Nada	Escrita em PC se saída Zero de ALU ativo
IorD	PC fornece endereço para a memória	ALUOut fornece endereço para a memória

Sinais de 2 bits da Unidade de Controle

Nome	Valor	Efeito
ALUOp	00	ALU faz soma
	01	ALU faz subtração
	10	Campo funct determina operação
ALUSrcB	00	2° Operando da ALU vem de B
	01	2° Operando da ALU é a constante 4
	10	2° Operando é o sinal estendido, 16 bits menos significantes de IR
	11	2° Operando é o sinal estendido, 16 bits menos significantes de IR, deslocados de 2 bits para a esquerda
PCSource	00	Saída de ALU (PC + 4) é enviado ao PC para escrita
	01	ALUOut (endereço destino do branch) é enviado ao PC para escrita
	10	Endereço destino do jump (26 bits menos significativos do IR deslocados de 2 bits para a esquerda concatenados com PC+4[31:28]) é enviado ao PC para escrita

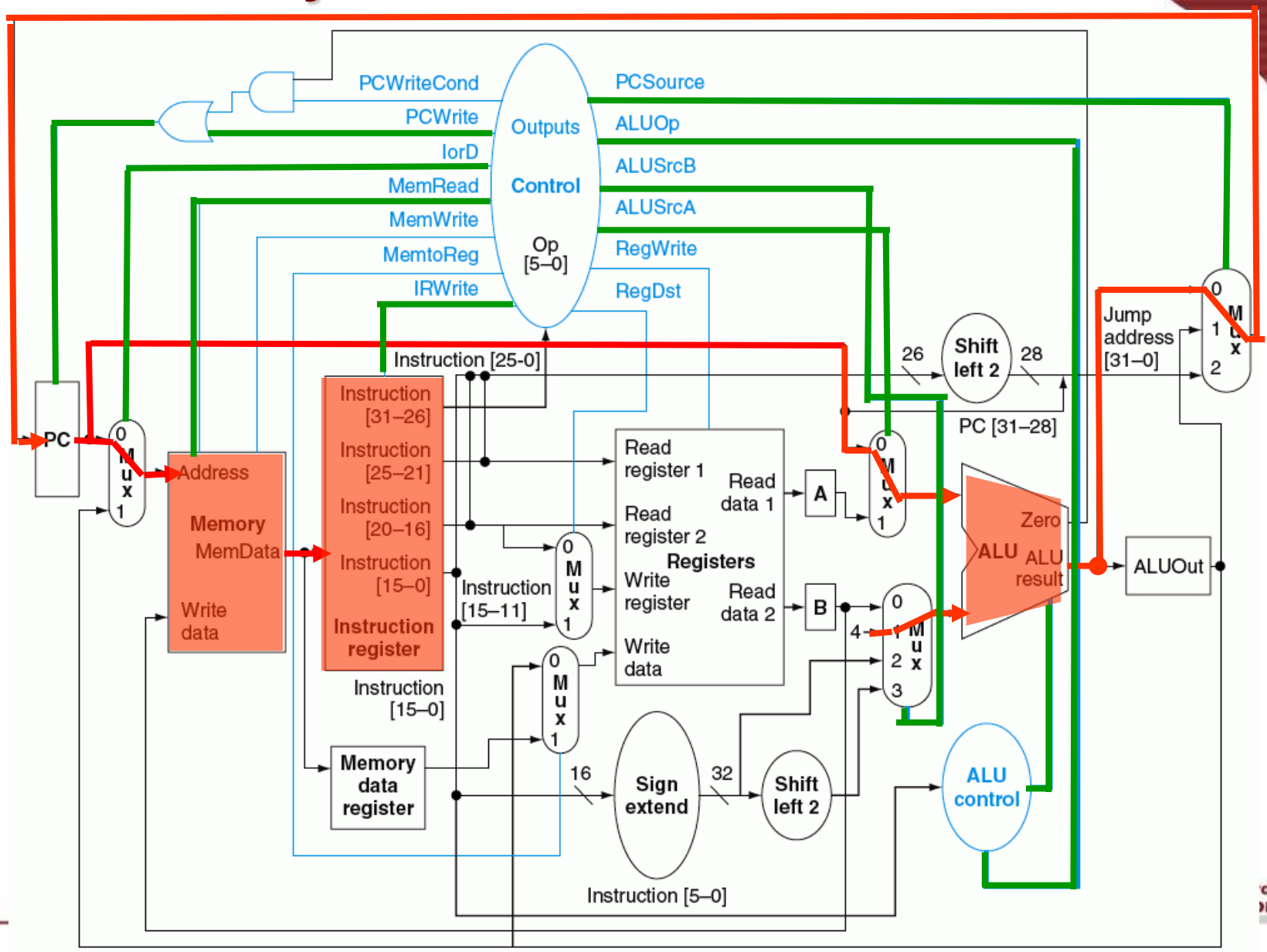
Quebrando Processamento de Instrução em Ciclos de Clock

Busca Instrução e Atualiza PC


```
IR <= Memory[PC];
PC <= PC + 4;
```

- Endereço de memória contido no PC é lido
- Instrução é escrita no IR
- PC é incrementado em 4

Busca Instrução e Atualiza de PC

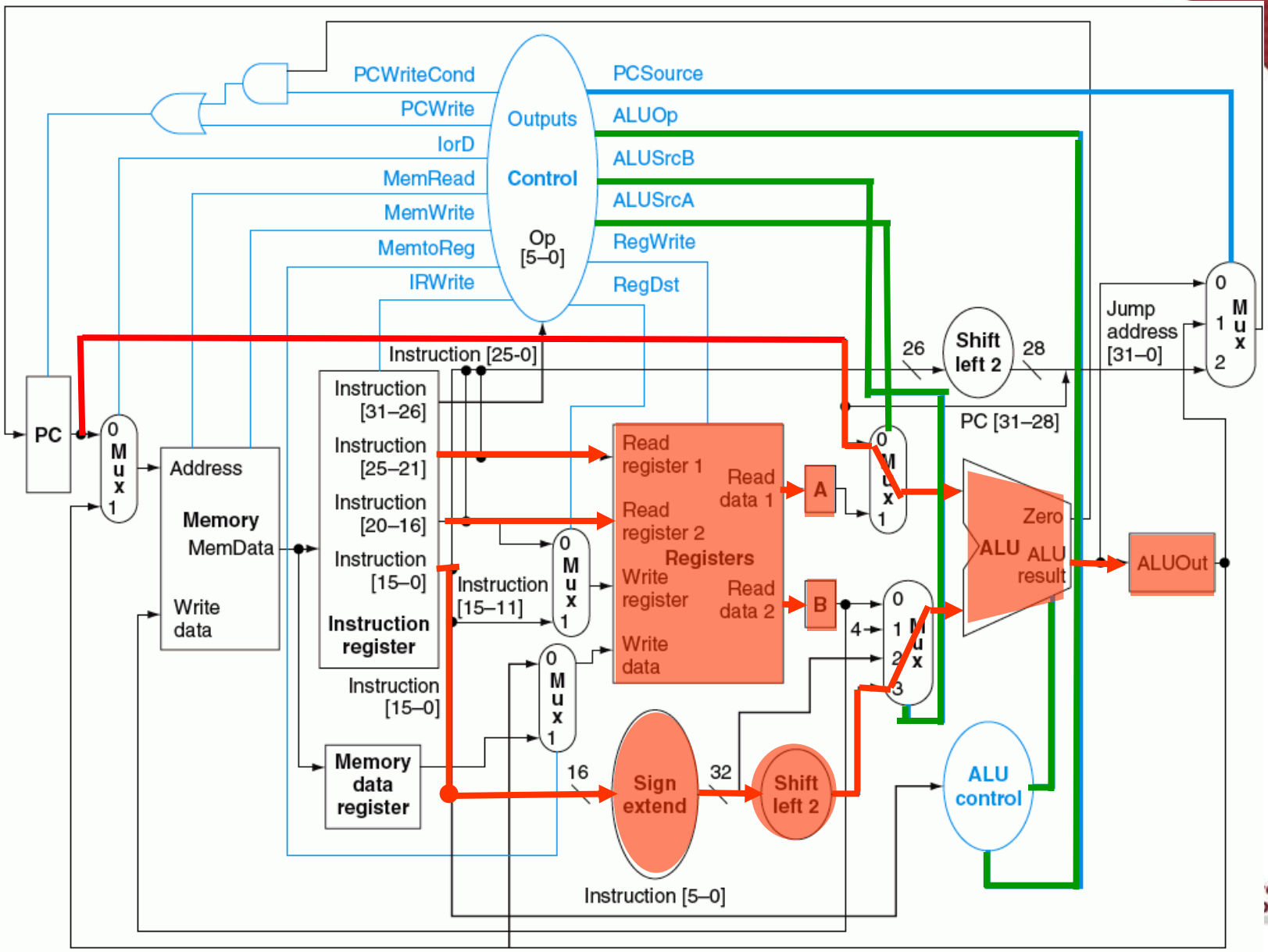


Decodifica e Lê Registradores


```
A <= Reg[IR[25:21]];
B <= Reg[IR[20:16]];
ALUOut <= PC + sign-ext(IR[15:0]) << 2;
```

- Ler registrador rs e escrever em A
- Ler registrador rt colocar e escrever em B
- Calcular endereço destino do branch e escrever em ALUOut
- **Se instrução não for arit/logica ou branch, estas ações não prejudicam funcionamento da CPU**

Decodifica e Lê Instruções



Opera com a ALU


```
//Referência à memória
ALUOut <= A + sign-ext(IR[15:0]);
```

OU

```
//Operação Arit/Lógica
ALUOut <= A op B;
```

OU

```
//Branch
if (A == B) PC <= ALUOut;
```

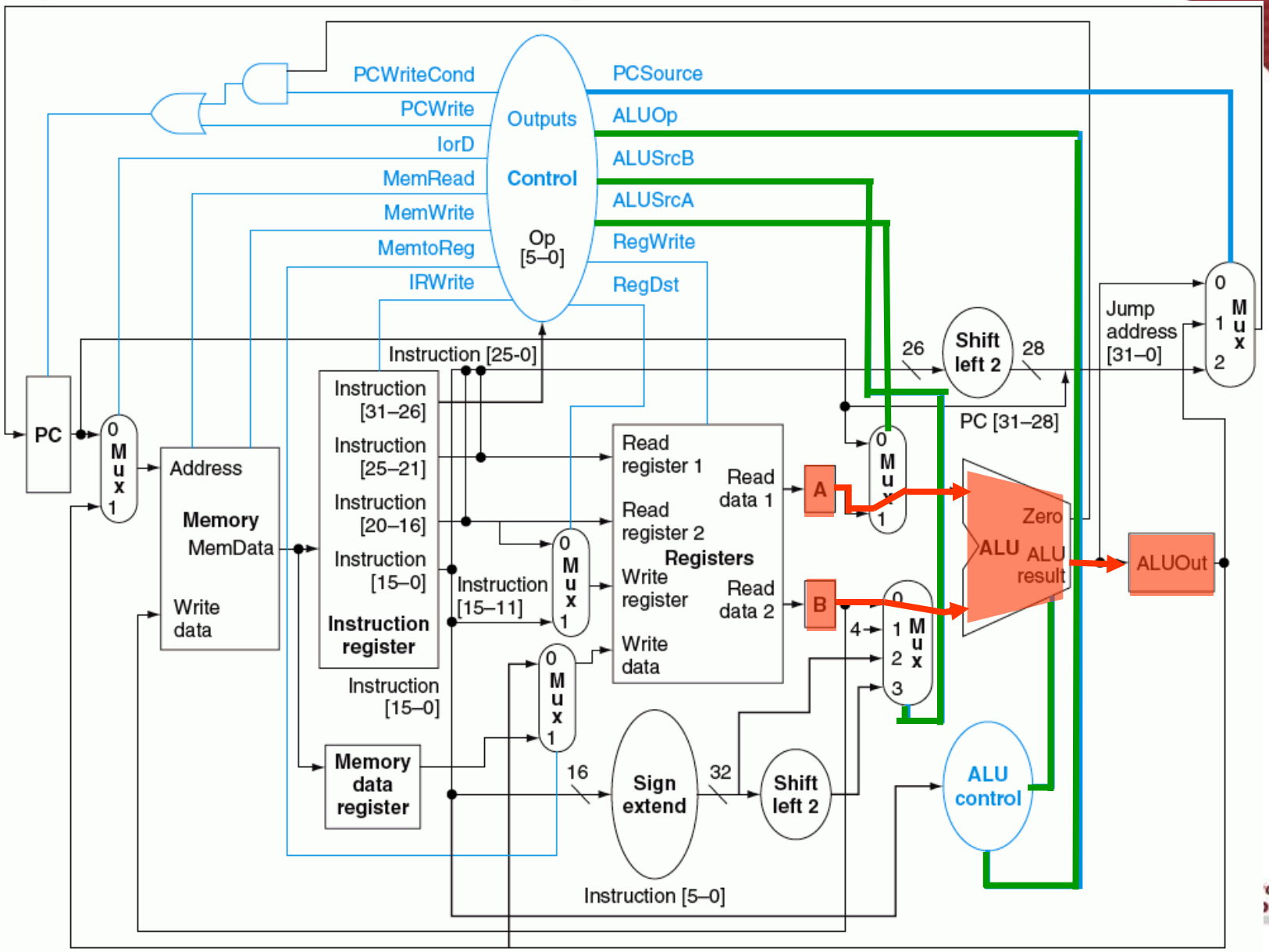
OU

```
//Jump
PC <=conc(PC[31:28],IR[25:0] << 2)
```

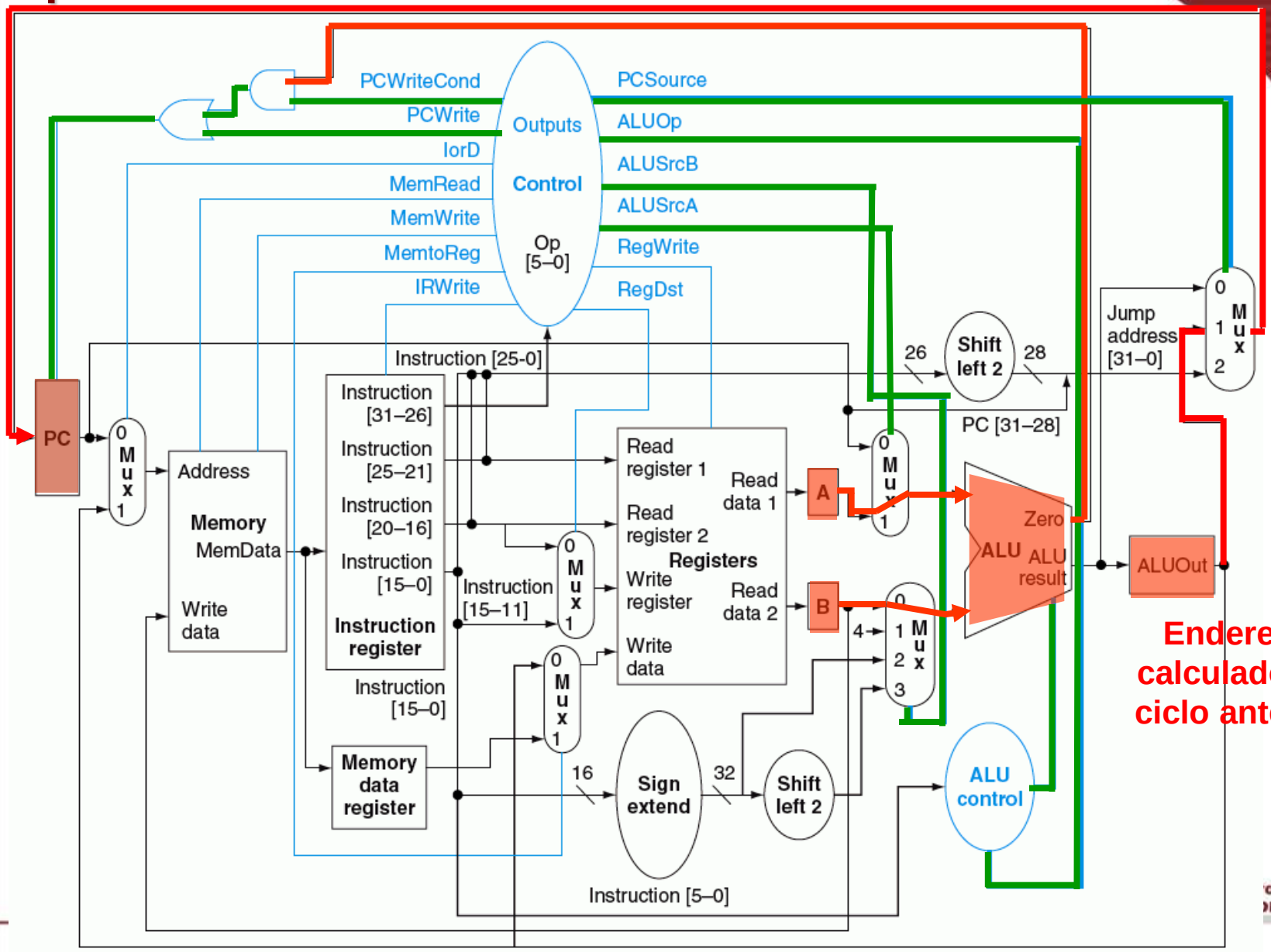
- Lê registradores A e B
- Dependendo da instrução, executa diferentes ações com ALU
- **Branch e Jump completados**

Centro de Informática

Opera com a ALU: Instruções Aritméticas

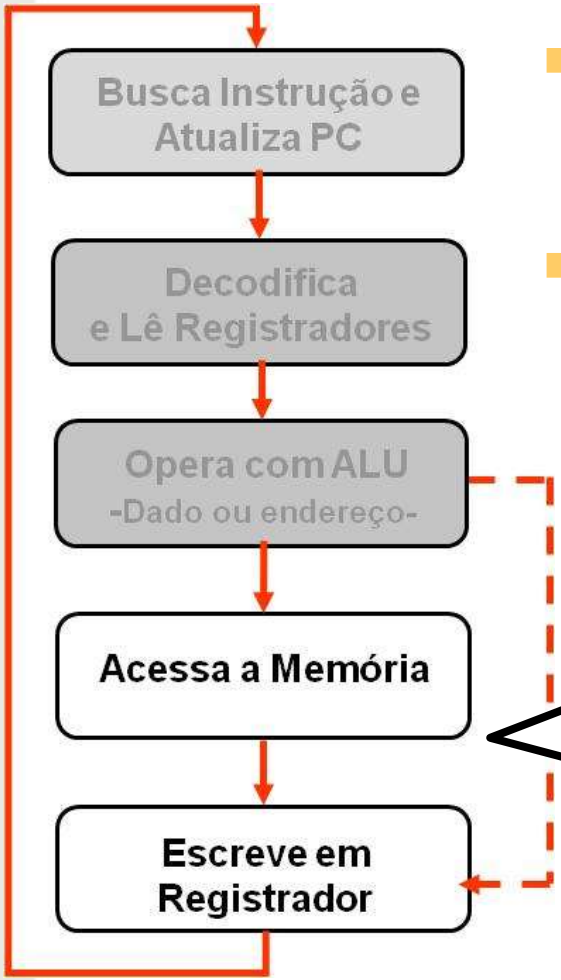


Opera com a ALU: Branch



Endereço calculado no ciclo anterior

Acessa a Memória (ou Escreve em Registrador)



- Instruções arit/lógicas e SW são completadas
  - **Arit/Lógicas escrevem no registrador**
- LW: dado da memória armazenado no endereço ALUOut é carregado em MDR

```
//Referência à memória LW
MDR <= Memory[ALUOut];
```

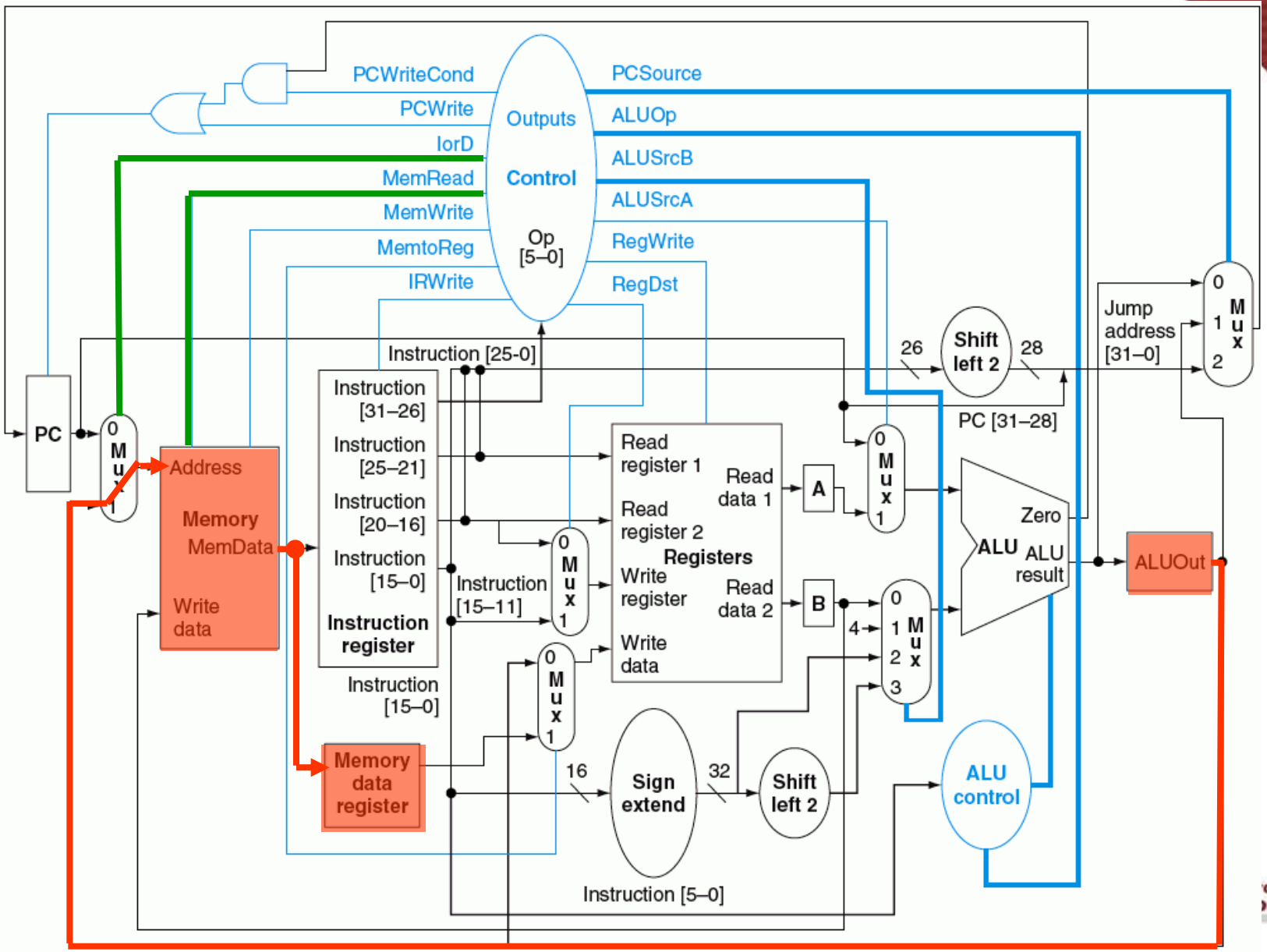
OU

```
//Referência à memória SW
Memory[ALUOut] <= B
```

OU

```
//Operação Arit/Lógica
Reg[IR[15:11]] <= ALUOut
```

Acessa a Memória: LW

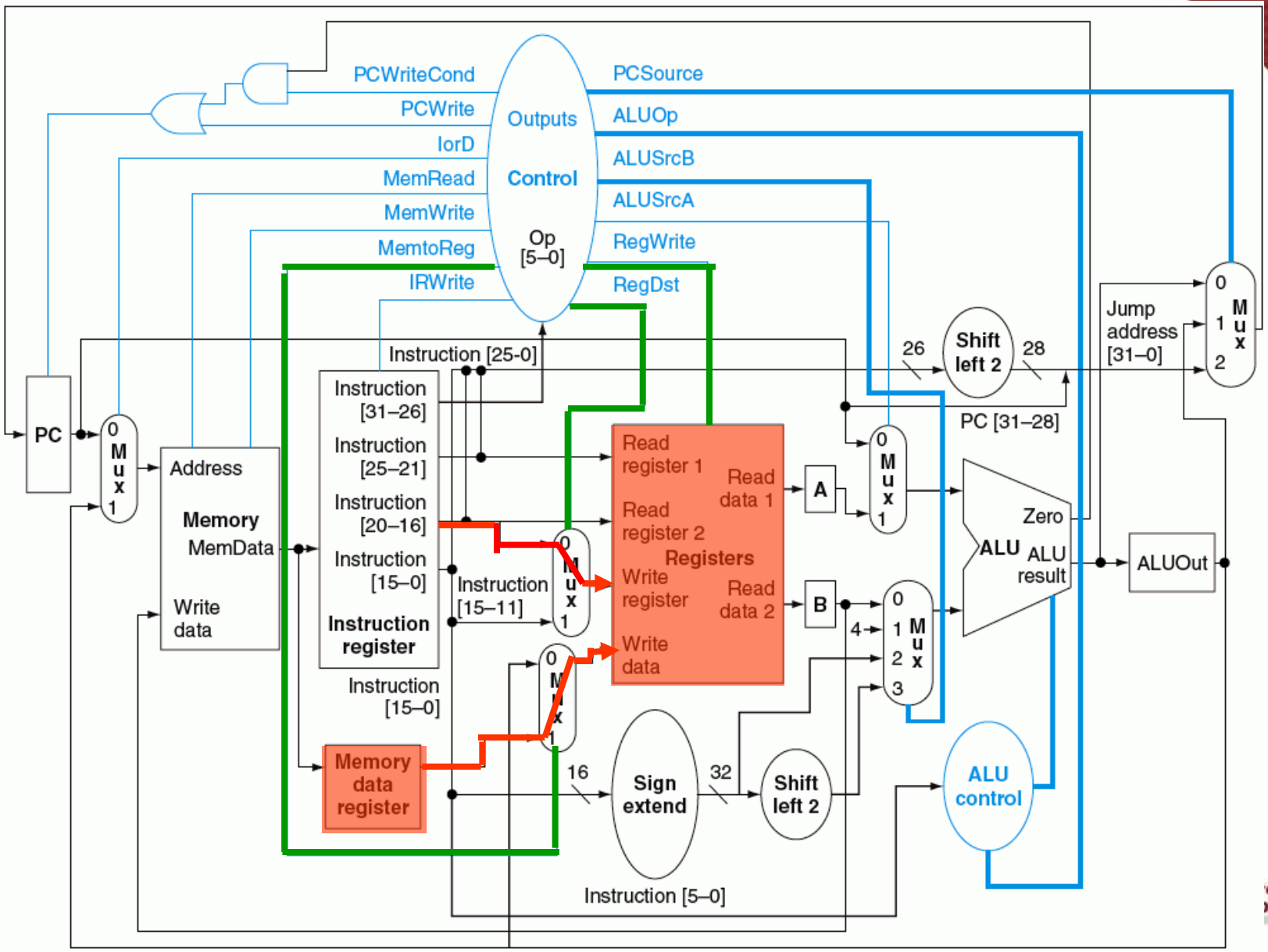


Escreve em Registrador

- Neste ciclo, instrução LW é completada
- Dado do MDR é escrito em um registrador do banco de registradores
- **LW é instrução mais lenta**

```
Reg[IR[20:16]] <= MDR
```

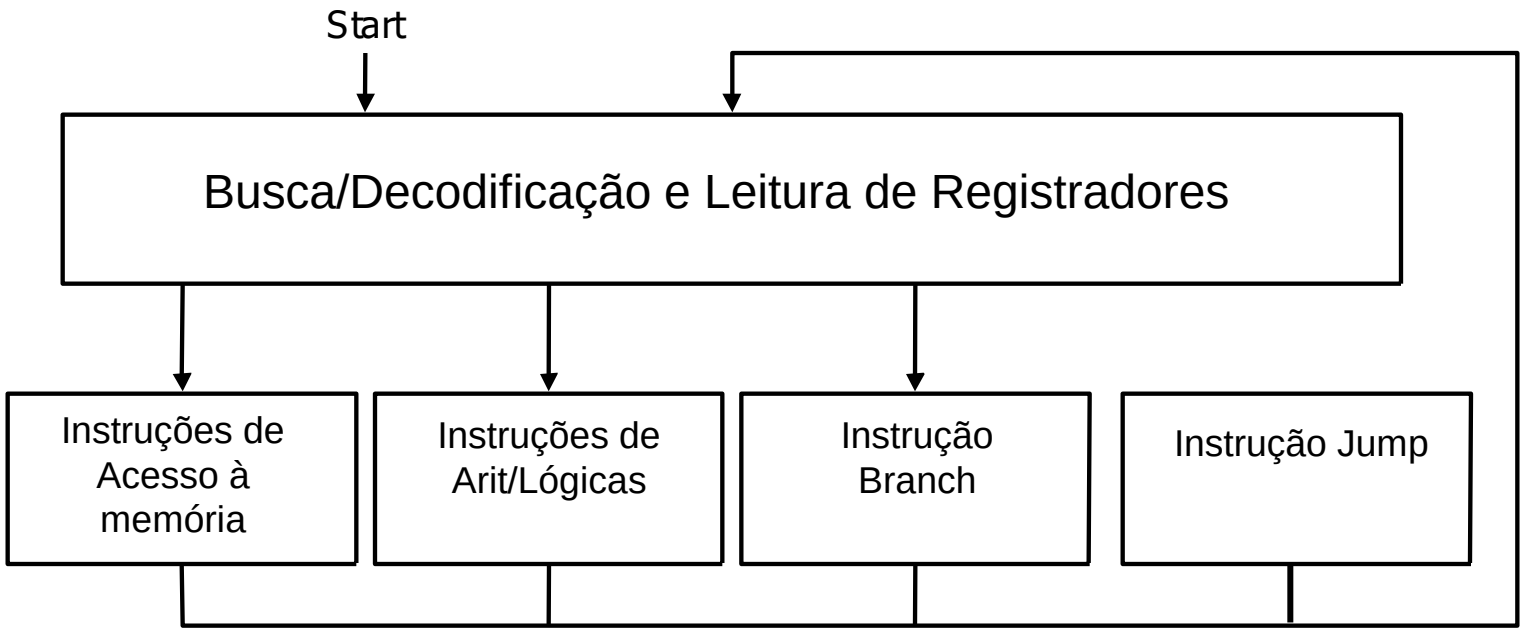
Escreve em Registrador: LW



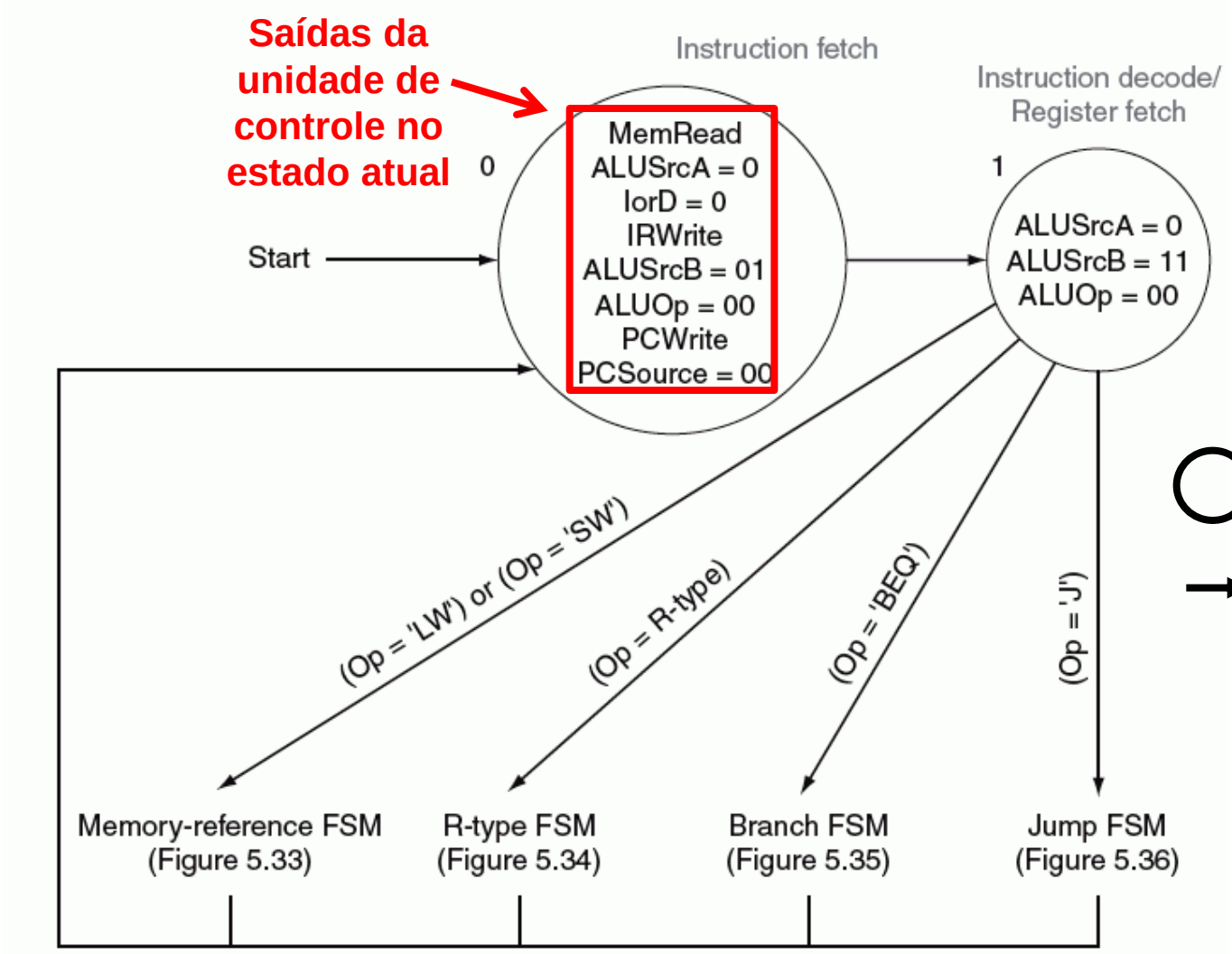
Implementação da Unidade de Controle

- Monociclo
  - Simplicidade de implementação
    - Tabela verdade usada para gerar lógica combinacional
- Multiciclo
  - Complexa
    - Estado do processamento da instrução deve ser levado em conta
    - Pode ser projetada com o auxílio de uma máquina de estados finita (**F**inite **S**tate **M**achine – FSM)

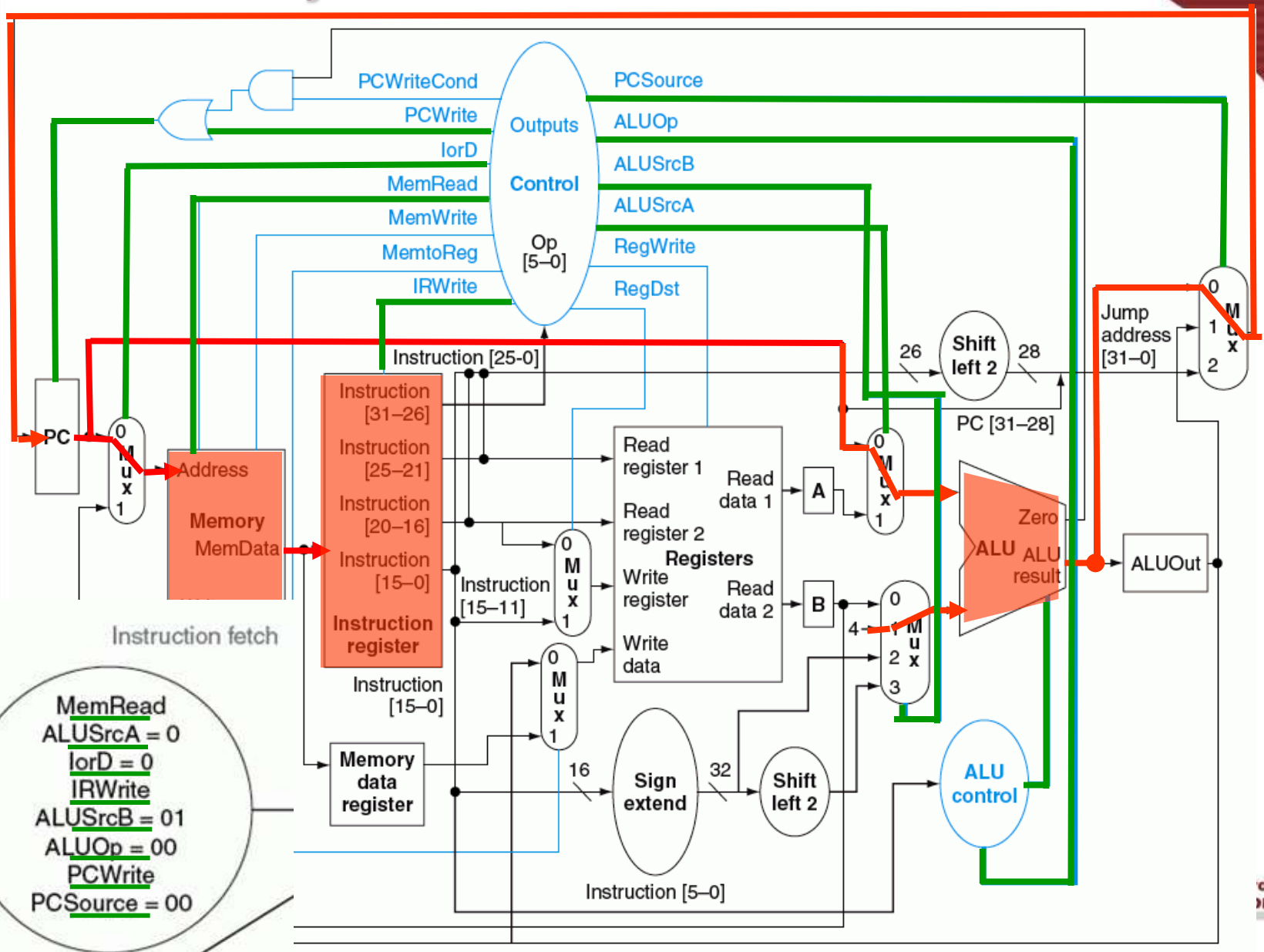
FSM da Unidade de Controle – Visão Abstrata



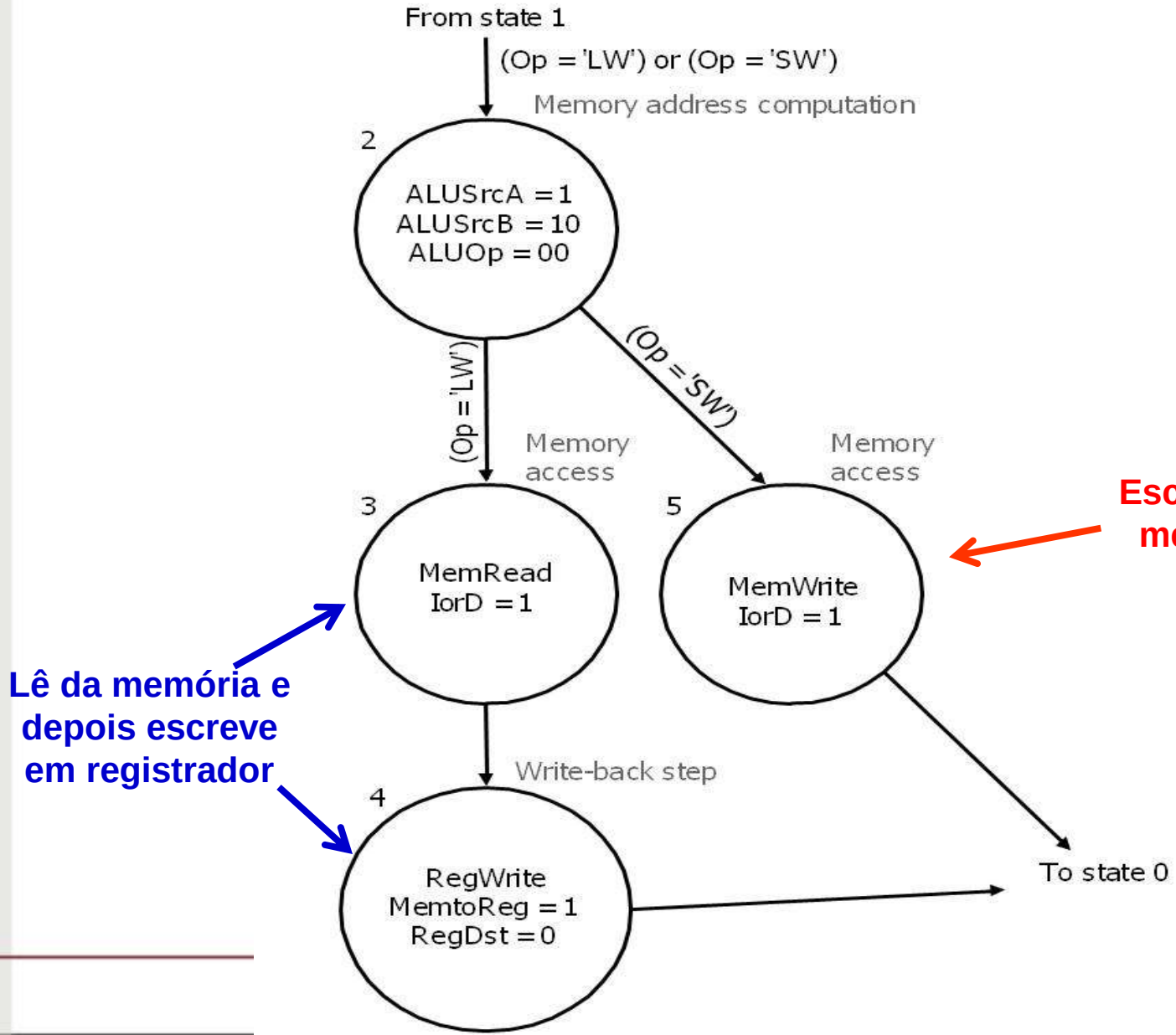
FSM da Busca, Decodificação e Leitura de Registradores



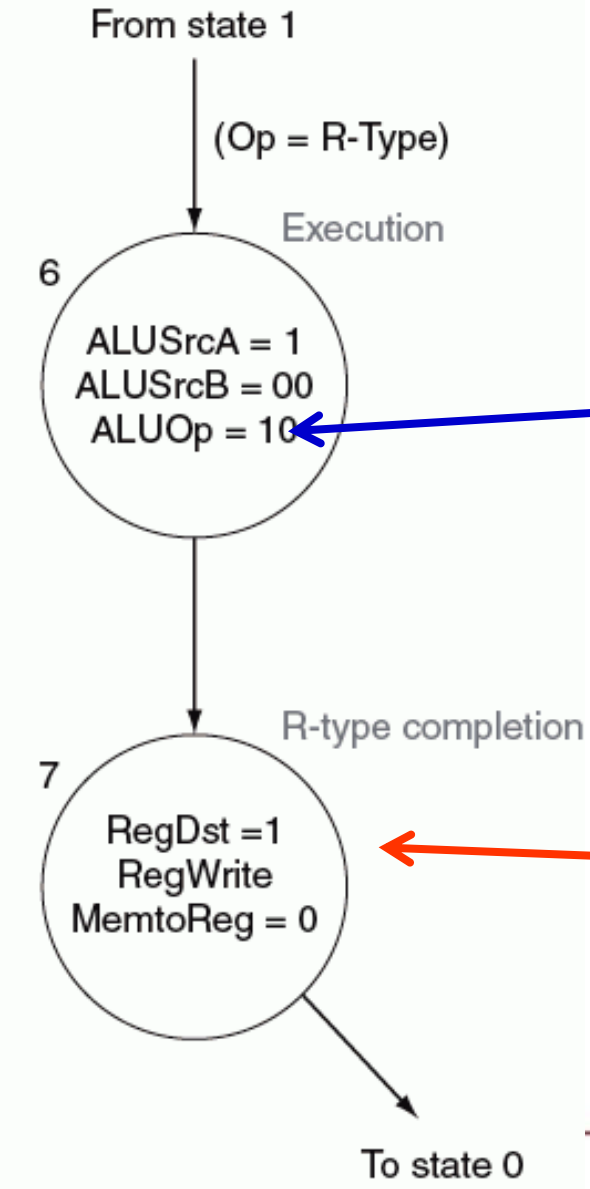
Busca Instrução e Atualiza de PC



Instruções de Acesso a Memória



Instruções de Aritméticas/Lógicas

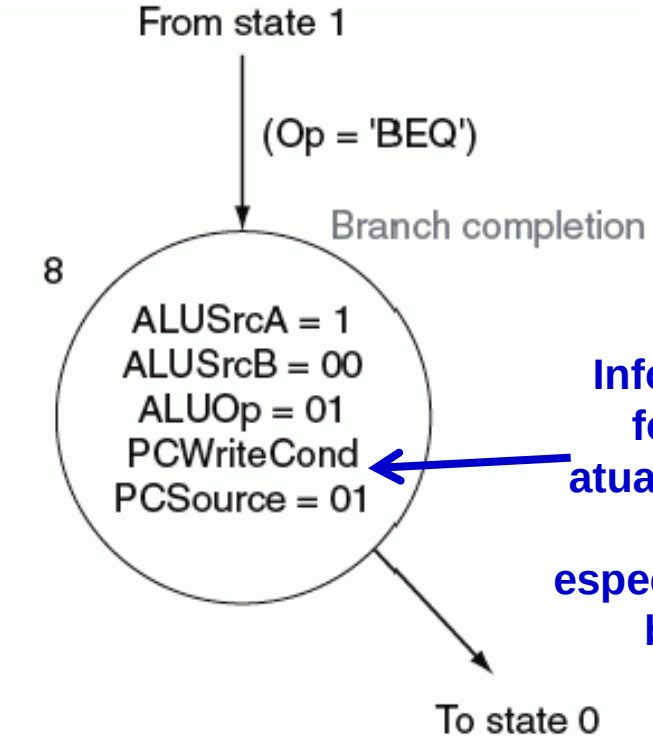


Executa operação especificada no campo funct

Escreve resultado no registrador

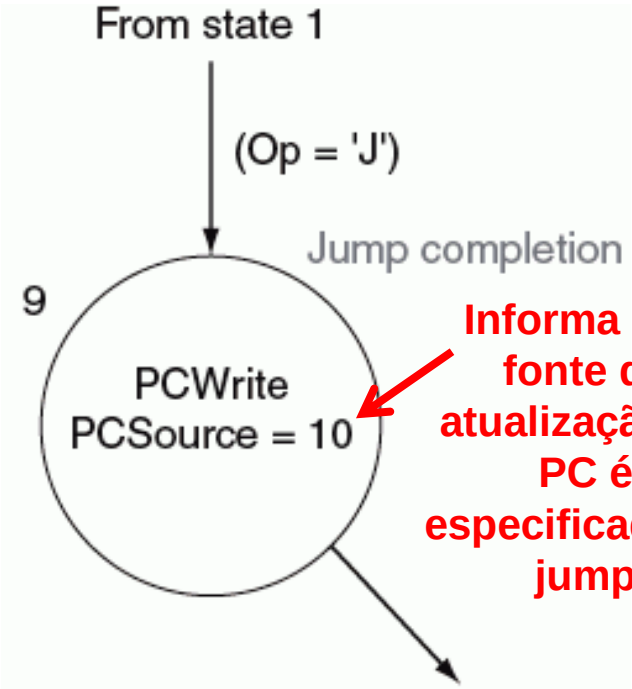
Instruções de Branch e Jump

Branch



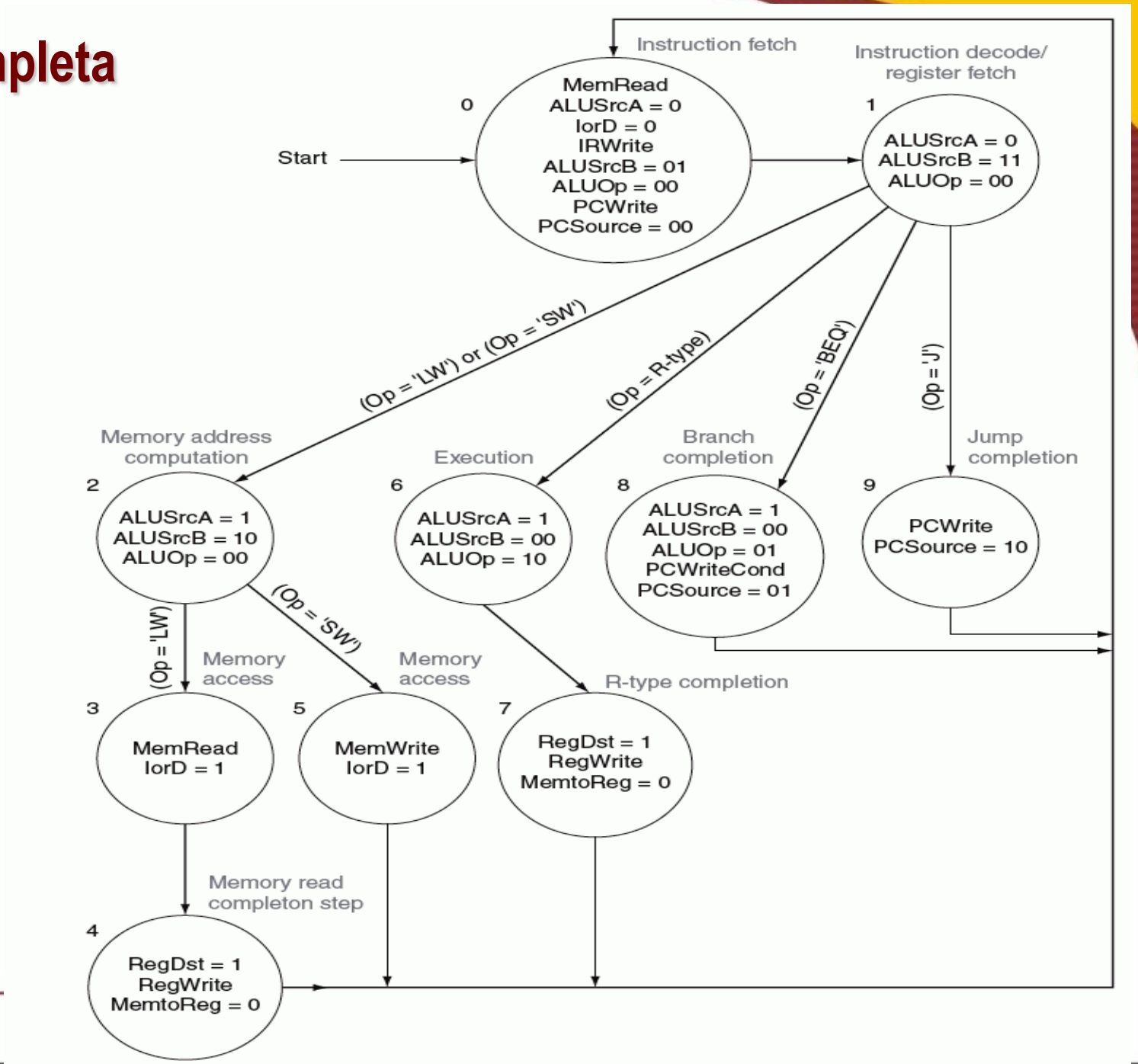
Informa que fonte de atualização de PC é especificado no branch

Jump

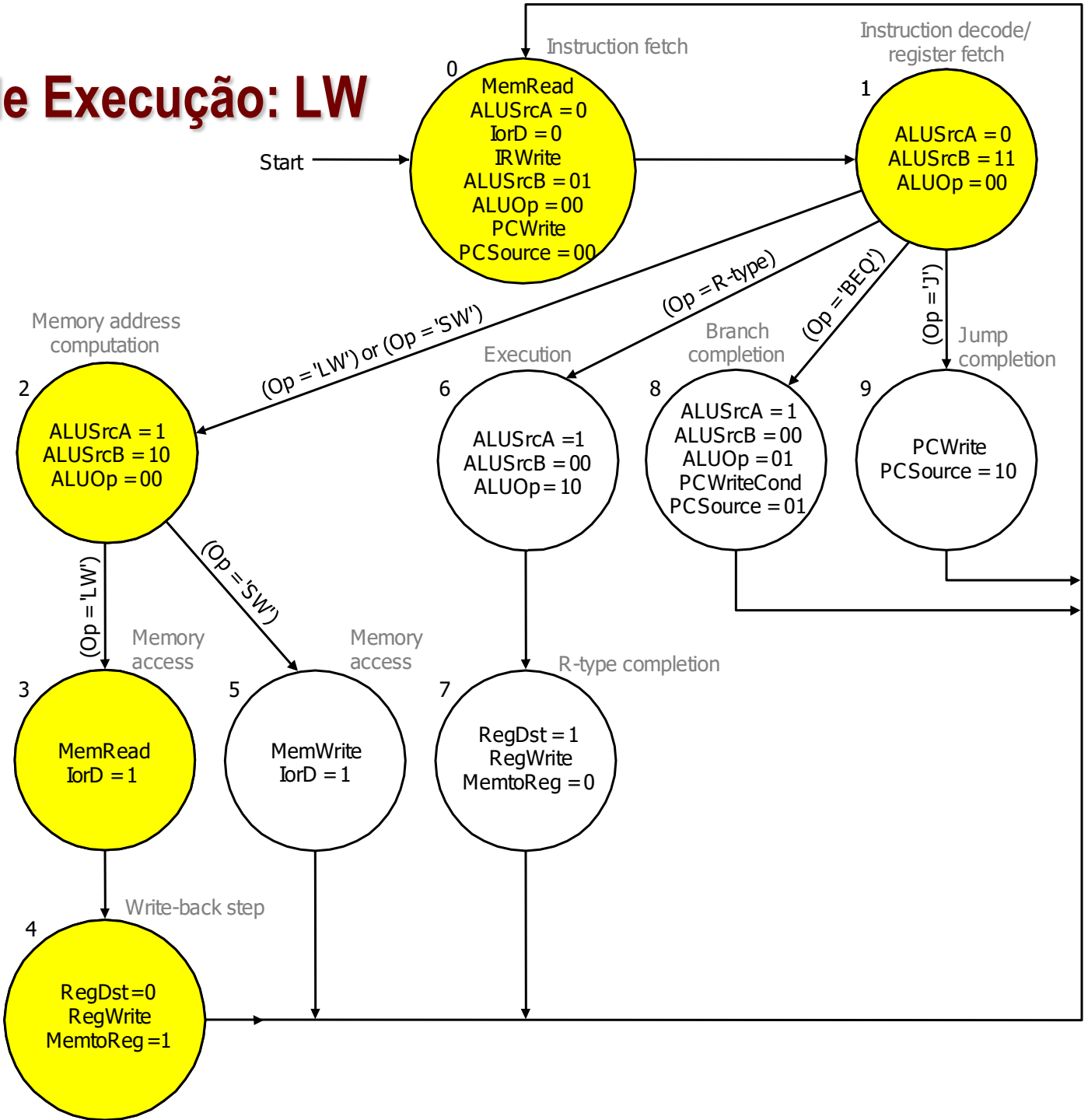


Informa que fonte de atualização de PC é especificado no jump

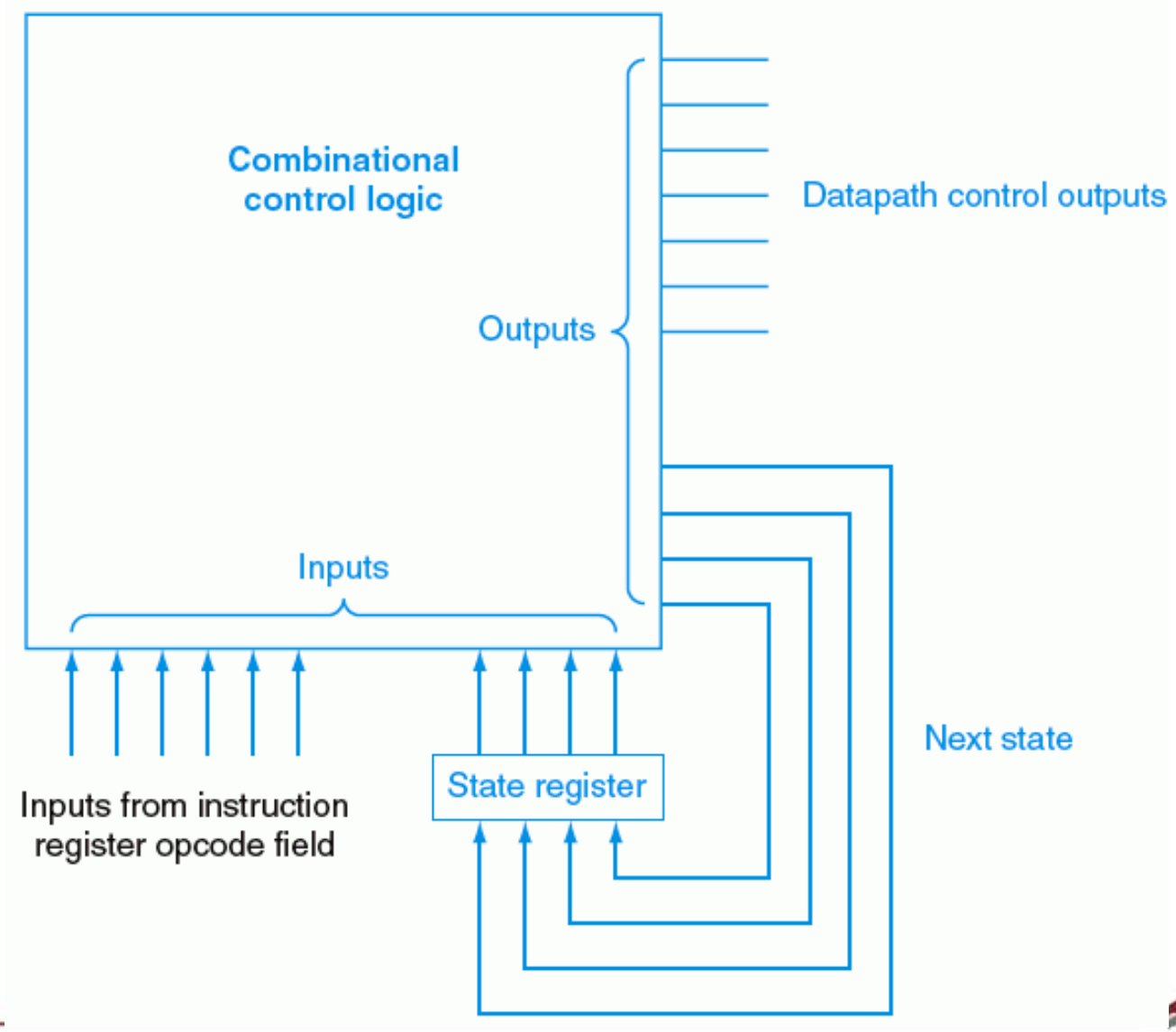
FSM Completa



Exemplo de Execução: LW



Implementação de Unidade de Controle



CPU: Exceções

- Sequência de execução é alterada devido a eventos não esperados:
  - internos:
    - opcode inexistente
    - overflow
    - divisão por zero
  - externos:
    - dispositivo de entrada/saída

Exceções

- Execução do programa é interrompida e uma rotina de tratamento é executada
  - Valor do PC deve ser guardado
  - O endereço da rotina de tratamento deve ser carregado em PC

Perguntas Importantes

- **Onde guardar o endereço do PC?**
  - Registrador
    - MIPS: EPC
  - Pilha
- **Onde o endereço da sua subrotina de tratamento deve ser guardado?**
  - Valor fixo – MIPS
    - 0x8000 0180
    - precisa-se saber a causa da exceção
  - Vetor de endereços na memória
    - Endereço pré-estabelecido informa a causa da exceção

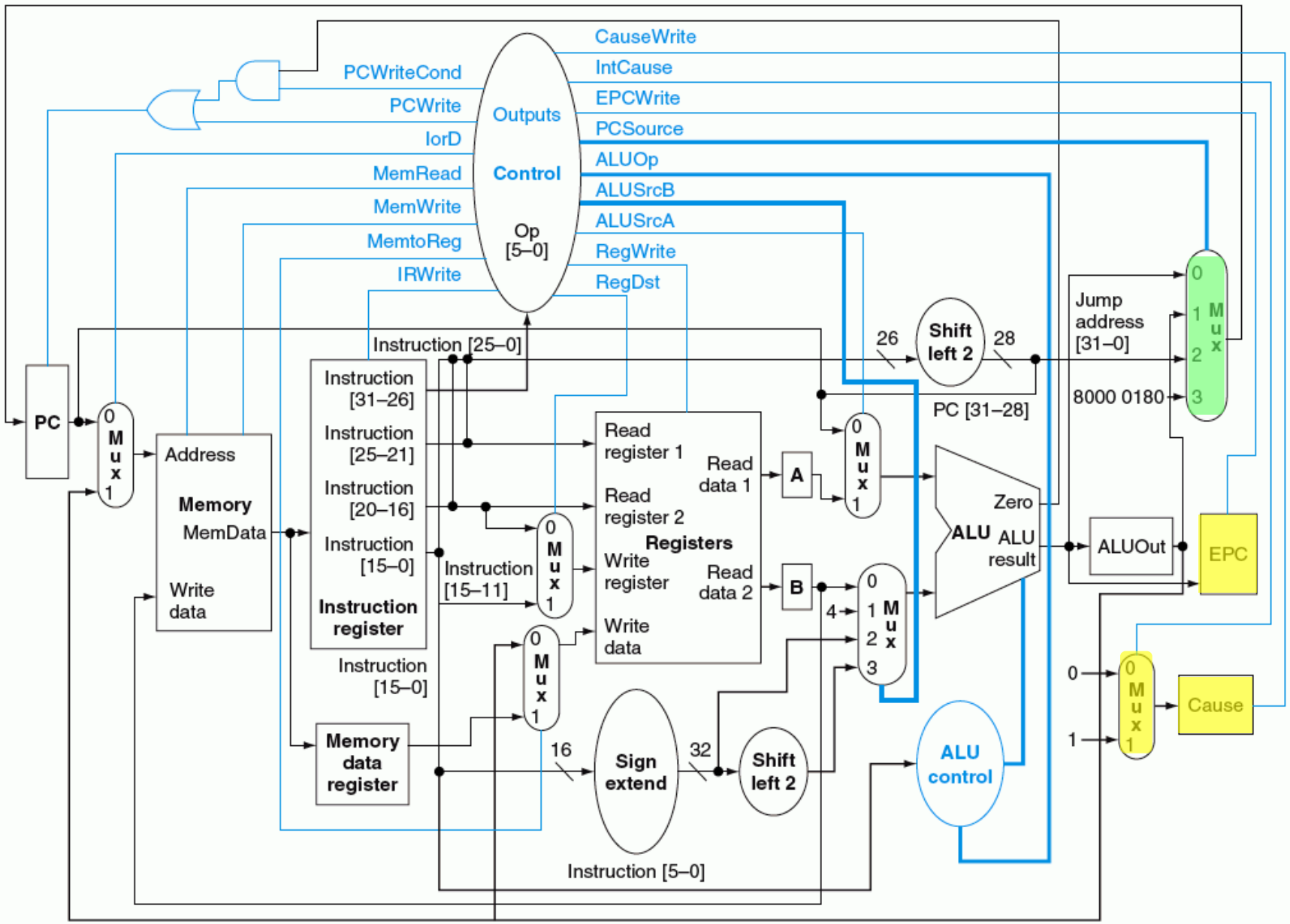
Exceções - MIPS

- Tipos de Exceções:
  - Instrução Indefinida
  - Overflow aritmético
- Registrador EPC
  - guarda endereço da instrução afetada
- Registrador de Causa
  - identifica o tipo de evento que causou a exceção

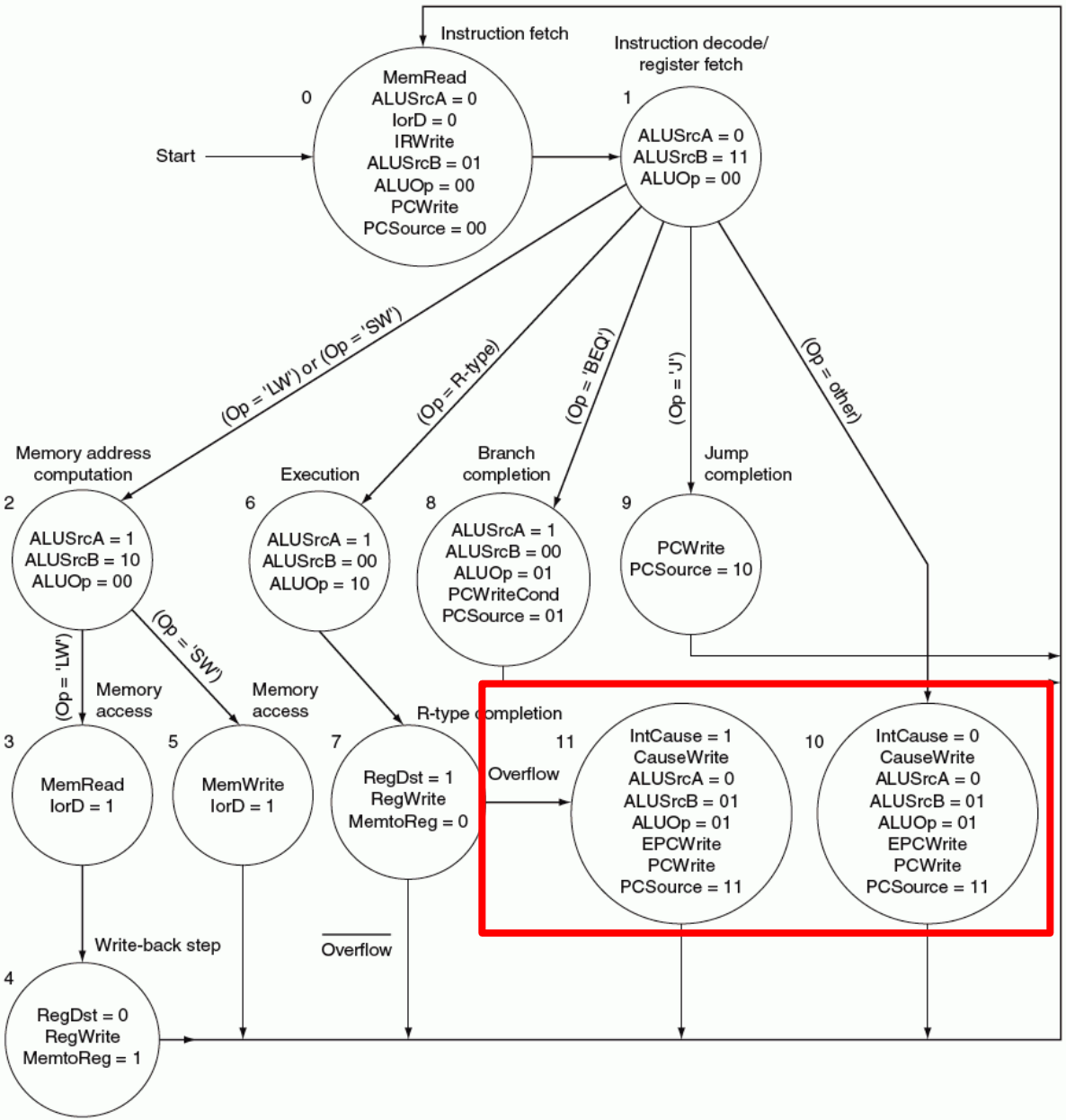
Implementando Exceções

- Assumindo que a CPU trata dois tipos de exceção:
  - Instrução inexistente (código da causa = 0)
  - Overflow (código da causa = 1)
- Precisa-se mudar unidade de processamento para:
  - Armazenar causa
  - Armazenar endereço de instrução afetada (EPC)
  - Escrever no PC endereço da subrotina de tratamento
    - 0x8000 0180
- Precisa-se mudar unidade de controle para:
  - Permitir escrita no registrador de causa e EPC
  - Selecionar o que é escrito dependendo da exceção
  - Permitir a atualização do PC com endereço da rotina de tratamento

CPU Trabalhando com Exceções



Trabalhando com Exceções - Controle



Novos estados decorrentes do tratamento de exceção