Implementação RISC-V
Uniciclo – Caminho de Dados

Introdução

- O desempenho (tempo de execução) de uma máquina pode ser determinado por três fatores:
  - número de instruções executadas (I)
  - número de ciclos por instrução (CPI)
  - período do *clock* (T)
- Para um determinado algoritmo, o compilador e a ISA determinam o número de instruções a serem executadas.
- A implementação do processador determina o período de *clock* e o CPI.

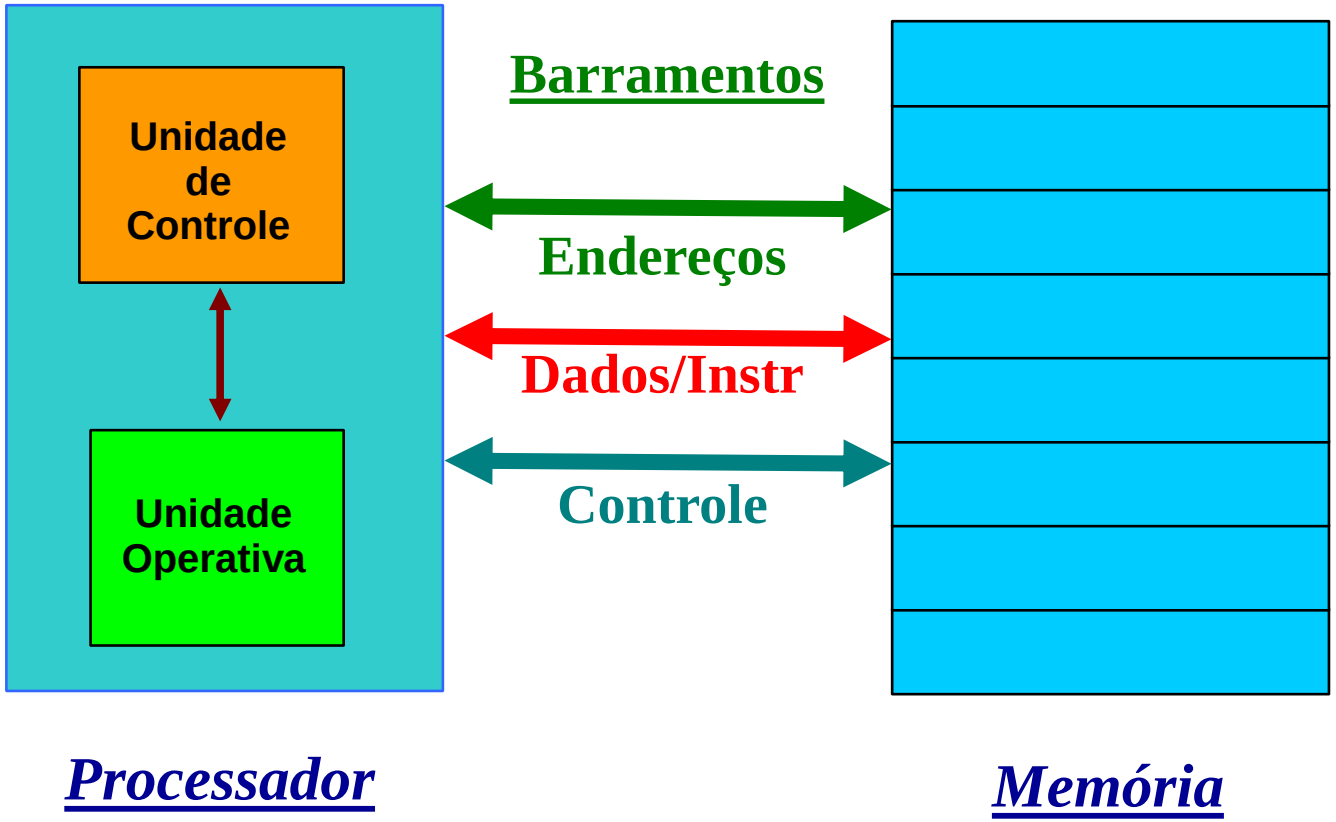
Introdução

- Vamos fazer a implementação de apenas um pequeno subconjunto das instruções do RISC-V

- O inter-relacionamento entre ISA e a implementação é exemplificado em três projetos alternativos:
  - Processador uniciclo
  - Processador multiciclo
  - Processador pipeline

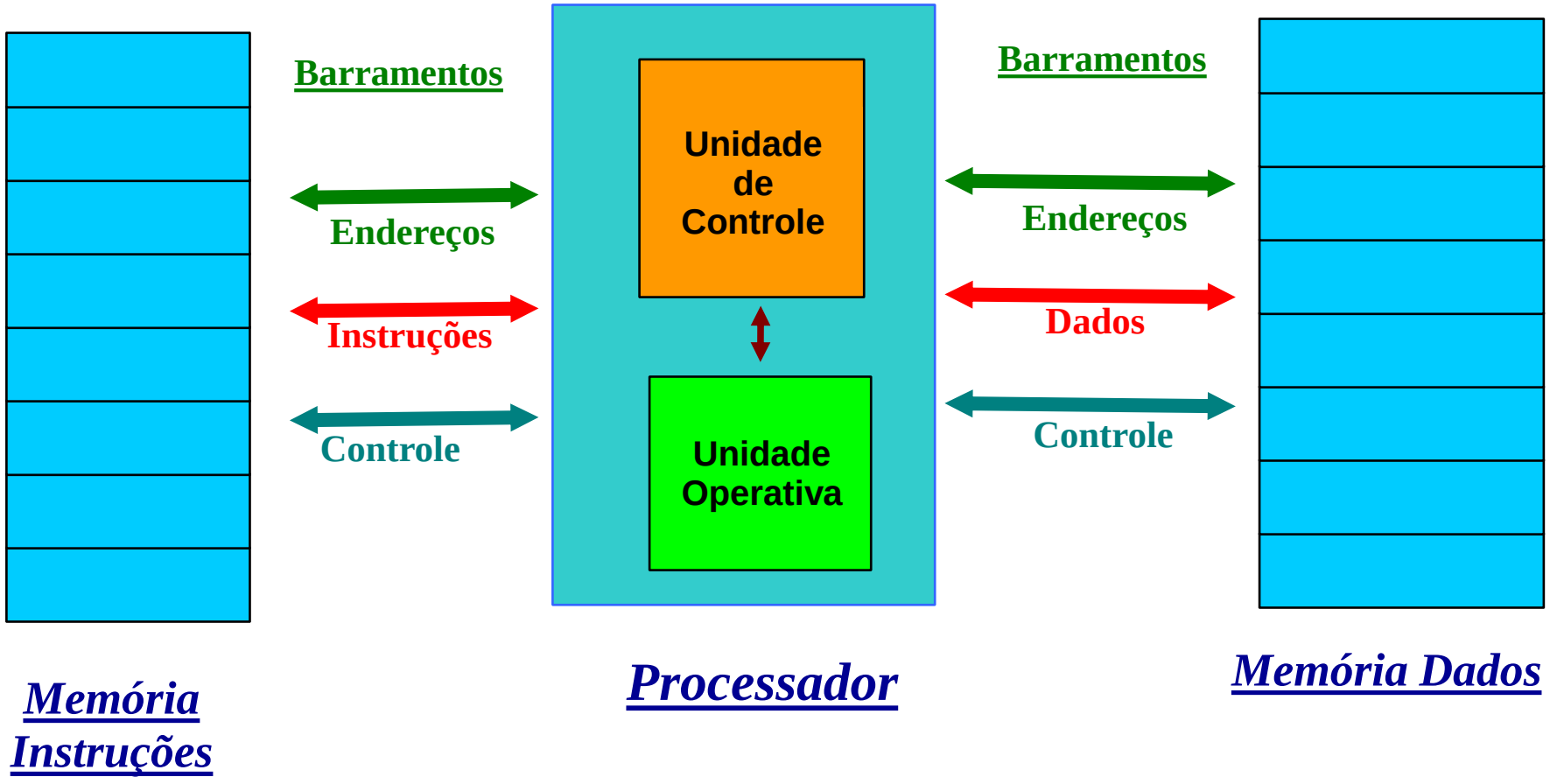
Arquitetura Von Neumann

- Estrutura básica



Arquitetura Harvard

- Estrutura básica



Unidade Operativa

- Também chamada “Parte Operativa”, “Caminho de Dados” ou, em inglês, *“Datapath”*
- É construída a partir dos seguintes componentes:
  - elementos de armazenamento (registradores, flip-flops)
  - operadores lógico-aritméticos (somadores, ULA)
  - recursos de interconexão (barramentos, mux)

Unidade Operativa RISC-V

- Será projetada para implementar o seguinte subconjunto de instruções do RISC-V:
  - Instruções de acesso à memória: lw e sw
  - Instruções lógicas e aritméticas: add, sub, and, or e slt
  - Instruções de desvios: beq e jal

Conjunto incompleto, mas demonstra os princípios básicos de projeto.

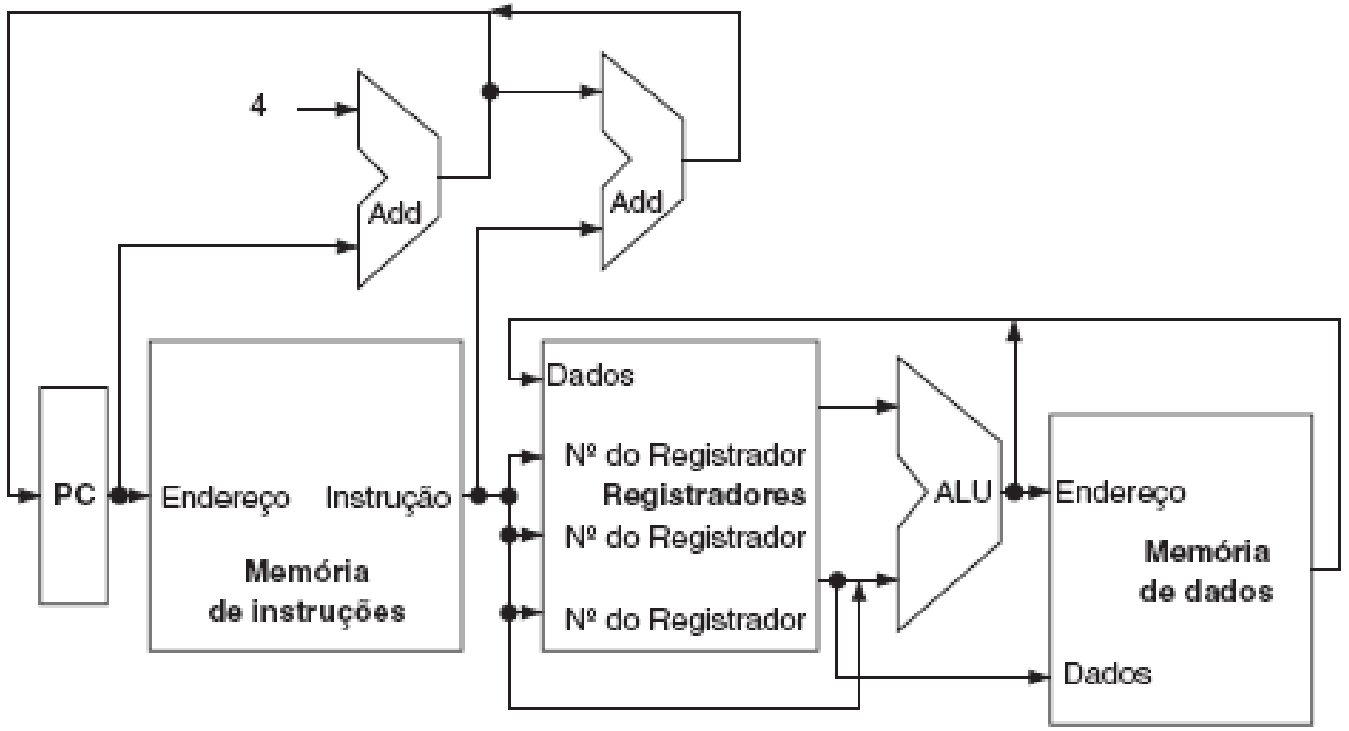
Obs. No laboratório, além destas, estão implementadas várias outras

Sinopse da Implementação

- Implementação genérica:
  - use o contador de programa (PC) para fornecer endereço da instrução
  - obtenha a instrução da memória
  - leia os registradores
  - use a instrução para decidir exatamente o que fazer

- Todas as instruções usam a ALU após lerem os registradores

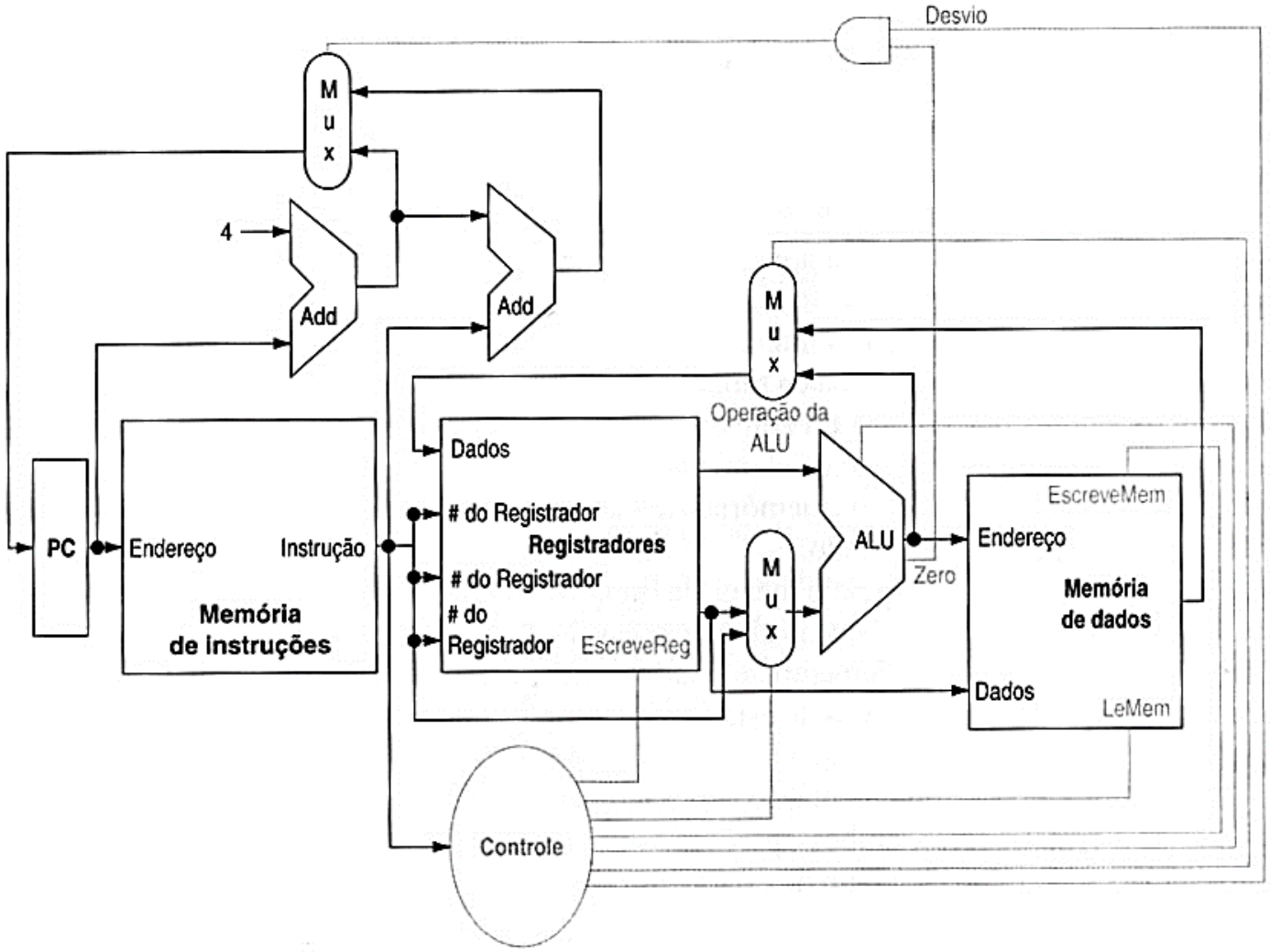
Visão de alto nível da implementação



Obs.: - Em vários pontos temos dados vindo de duas origens diferentes
- As unidades precisam ser controladas

Este esquemático é do processador MIPS (PC+4+Imm), mas os conceitos são os mesmos

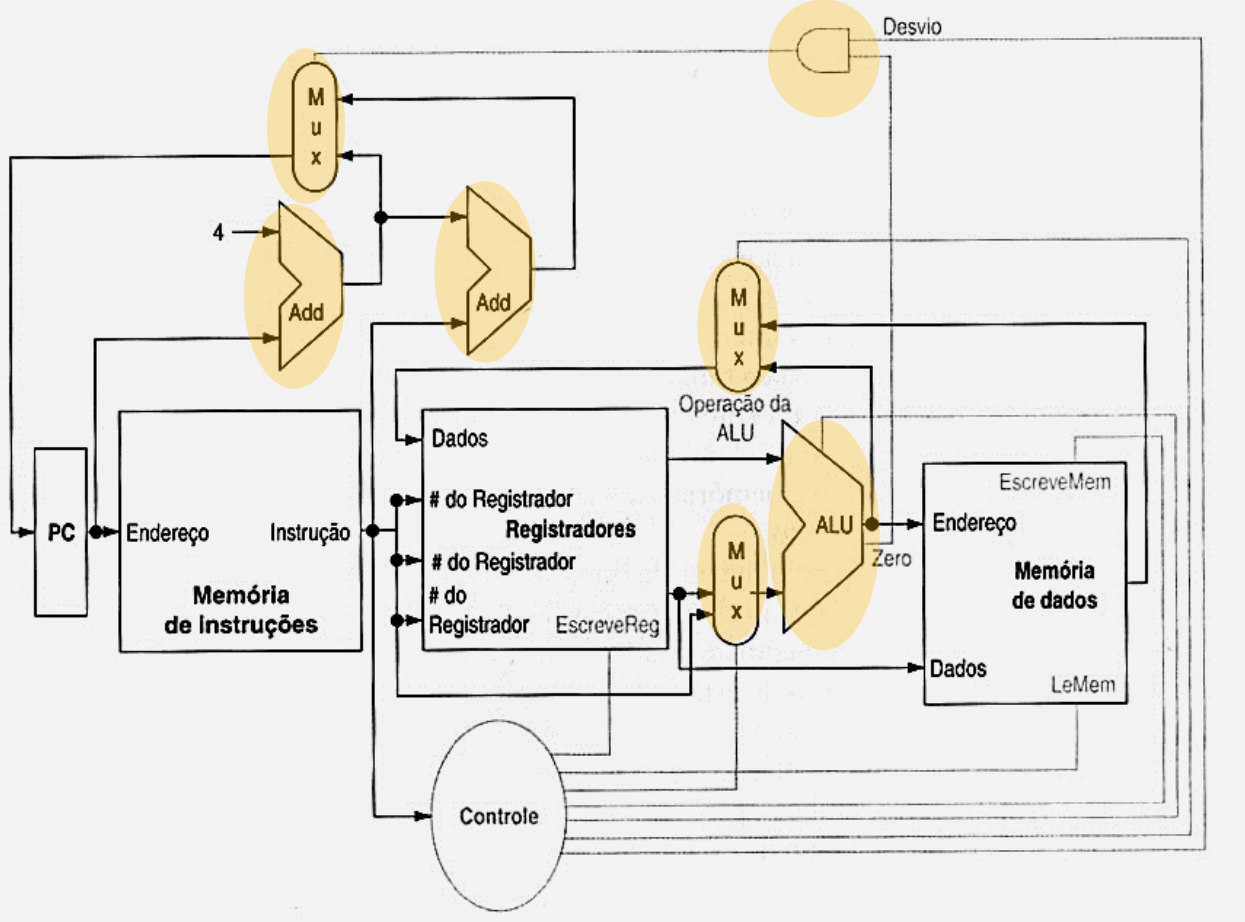
Visão geral da implementação



- Elementos combinacionais e sequenciais

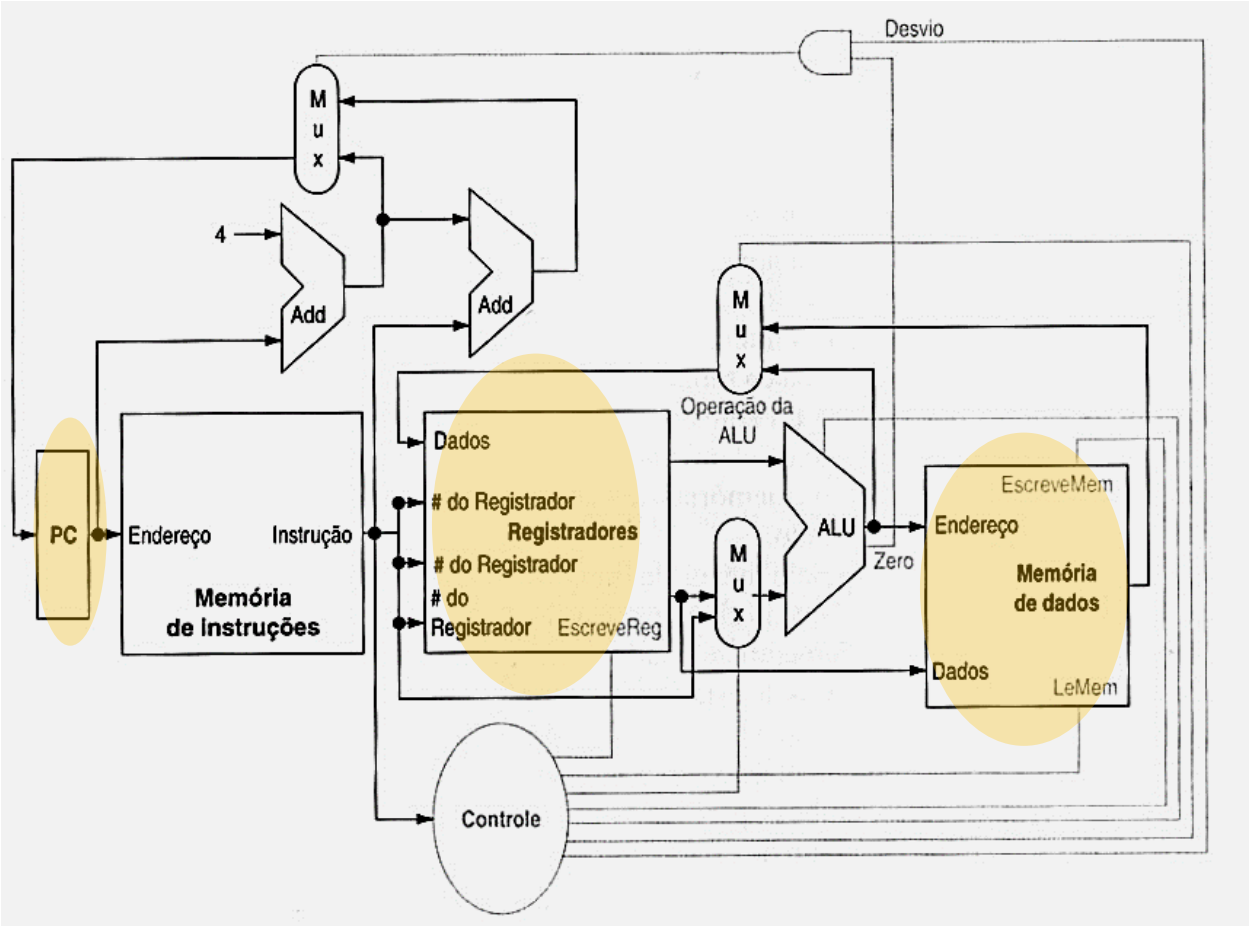
Componentes Combinacionais

- Componentes combinacionais definem o valor de suas saídas apenas em função dos valores presentes nas suas entradas



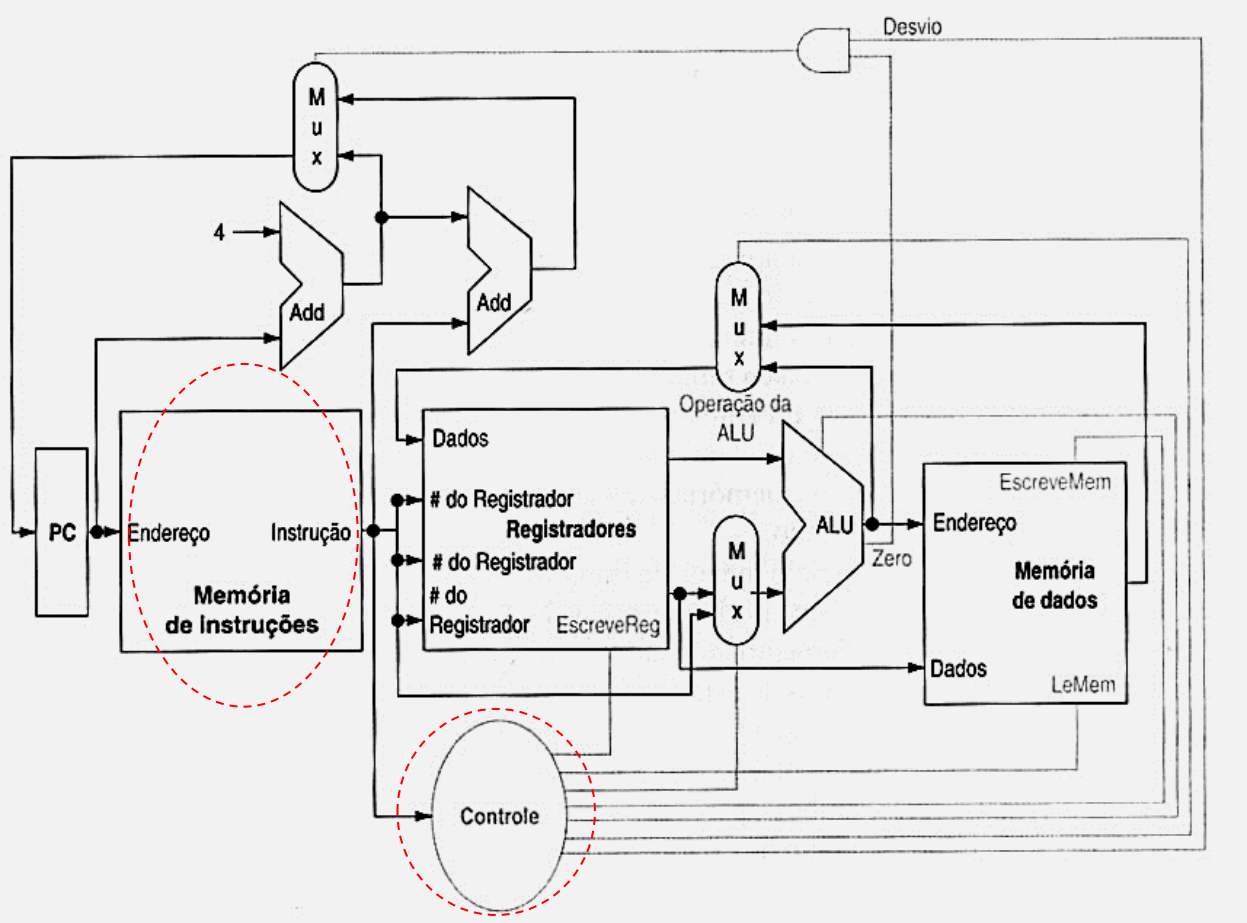
Componentes Sequenciais

- Componentes armazenadores de estado, as saídas dependem das entradas e do tempo.



Componentes ?

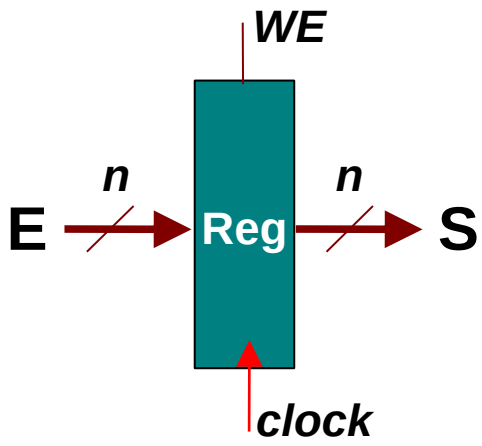
- Memória de Programa (Harvard x Von Neuman)
- Bloco de Controle (Uniciclo x Multiciclo)



Componentes Sequenciais

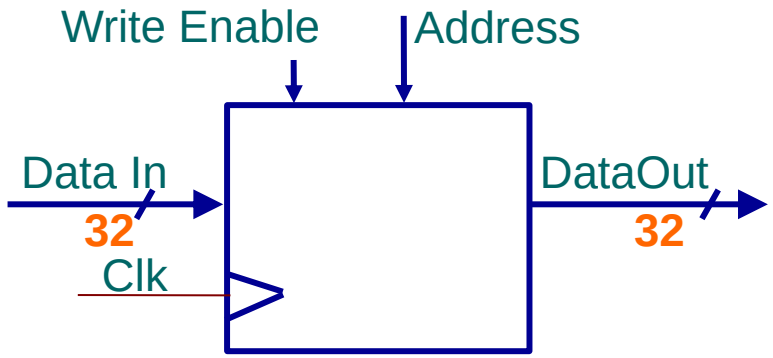
- Componentes sequenciais têm um *estado*, que define seu valor em um dado instante de tempo

- Registrador:
  - **Um conjunto de flip-flops tipo D**
    - **Com *n* bits de entrada e saída**
    - **entrada de habilitação de escrita**
  - **Habilitação de escrita (*Write Enable* - WE):**
    - **0: o dado de saída não muda**
    - **1: o dado de entrada será carregado (saída = entrada) na transição do *clock***



Memória

- Memória *(**One**/Two/Three... Ports)*
  - Um barramento de entrada: *Data In*
  - Um barramento de saída: *Data Out*



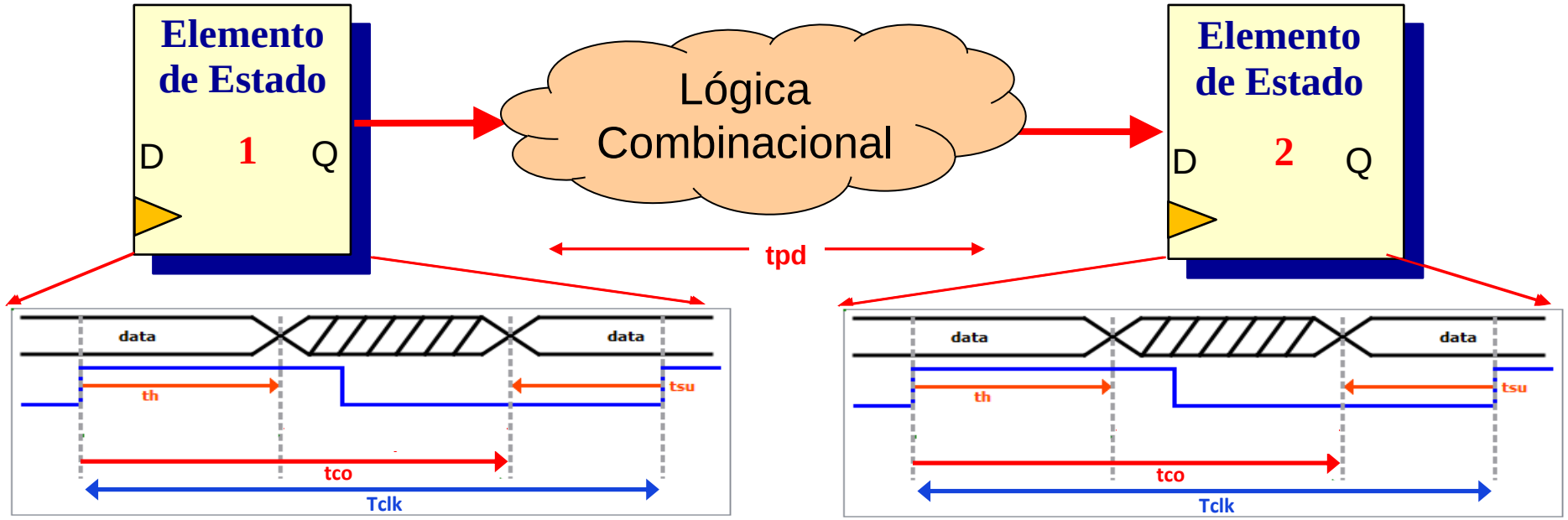
- Uma palavra é selecionada por:
  - Um endereço seleciona a palavra para ser colocada na saída (*Data out*)
  - Write Enable = 1: Permite que a palavra selecionada seja escrita com a informação na entrada (*Data in*)

- Entrada de Clock (*CLK*)
  - Sincroniza os processos de acesso à memória

Estratégia de Temporização

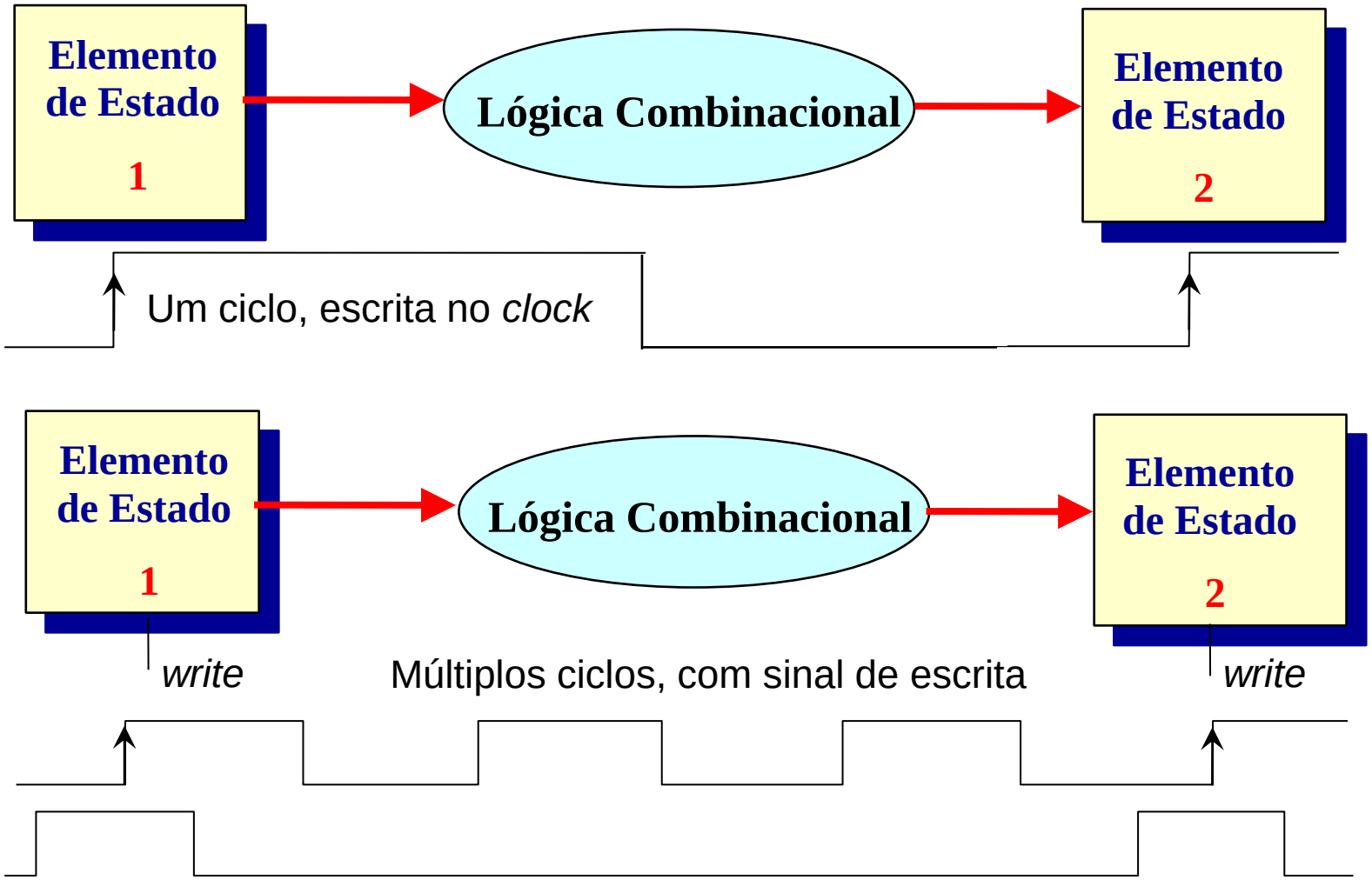
- Uma metodologia de temporização define quando os sinais podem ser lidos e quando eles podem ser escritos
- É necessário evitar situações de conflito, por exemplo, querer ler uma palavra e escrevê-la simultaneamente
- Será adotada uma metodologia de temporização sensível às transições do sinal do *clock*
- Nesta metodologia, qualquer valor armazenado nos elementos de estado só pode ser atualizado durante a transição do sinal de relógio (*clock*)

Revisão: Temporização em circuitos digitais



- **A escrita no elemento de estado 1 deve respeitar os tempos de:**
  - tempo de pré-carga (*setup time,* tsu) do elemento 1
  - tempo de hold (*hold time, th*) do elemento 1
- **O elemento de estado 2 só pode ser escrito depois que os dados em sua entrada estarem estáveis**
  - atraso de propagação da saída dado o clock (*clock to output,* tco) do elemento 1
  - atraso da lógica combinacional (*propagation delay*, tpd=max(tp_{HL},tp_{LH}))
  - tempo de pré-carga (*setup time,* tsu) do elemento de estado 2

Estratégia de Temporização



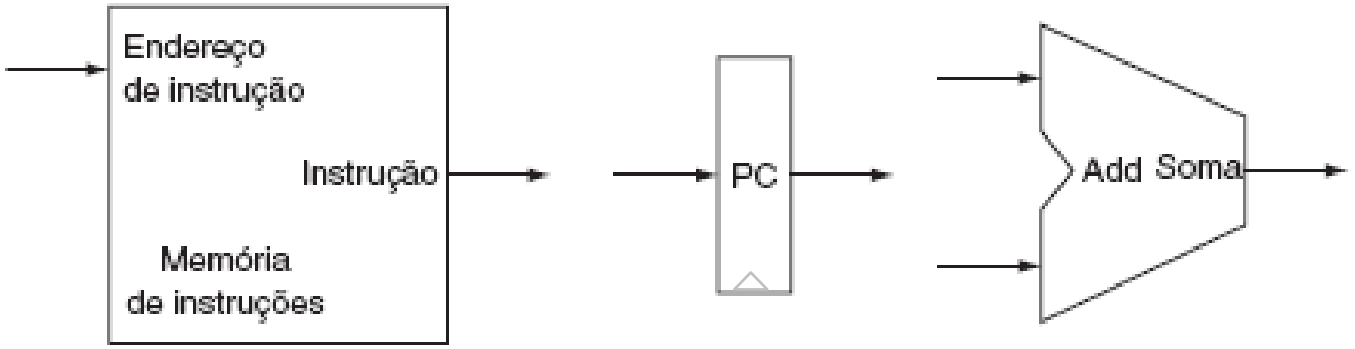
Criando o Caminho de Dados UNICICLO

- Uma maneira de se começar o projeto de um caminho de dados (*data path*) é examinar cada um dos componentes necessários para a execução de cada uma das classes de instruções

- Elementos necessários:
  - um lugar para armazenar as instruções (*memória de instruções*)
  - Um registrador para armazenar o endereço de instrução (*Program Counter – PC*)
  - Um contador para incrementar o PC, para compor o endereço da próxima instrução (soma 4 para endereçar próxima instrução)

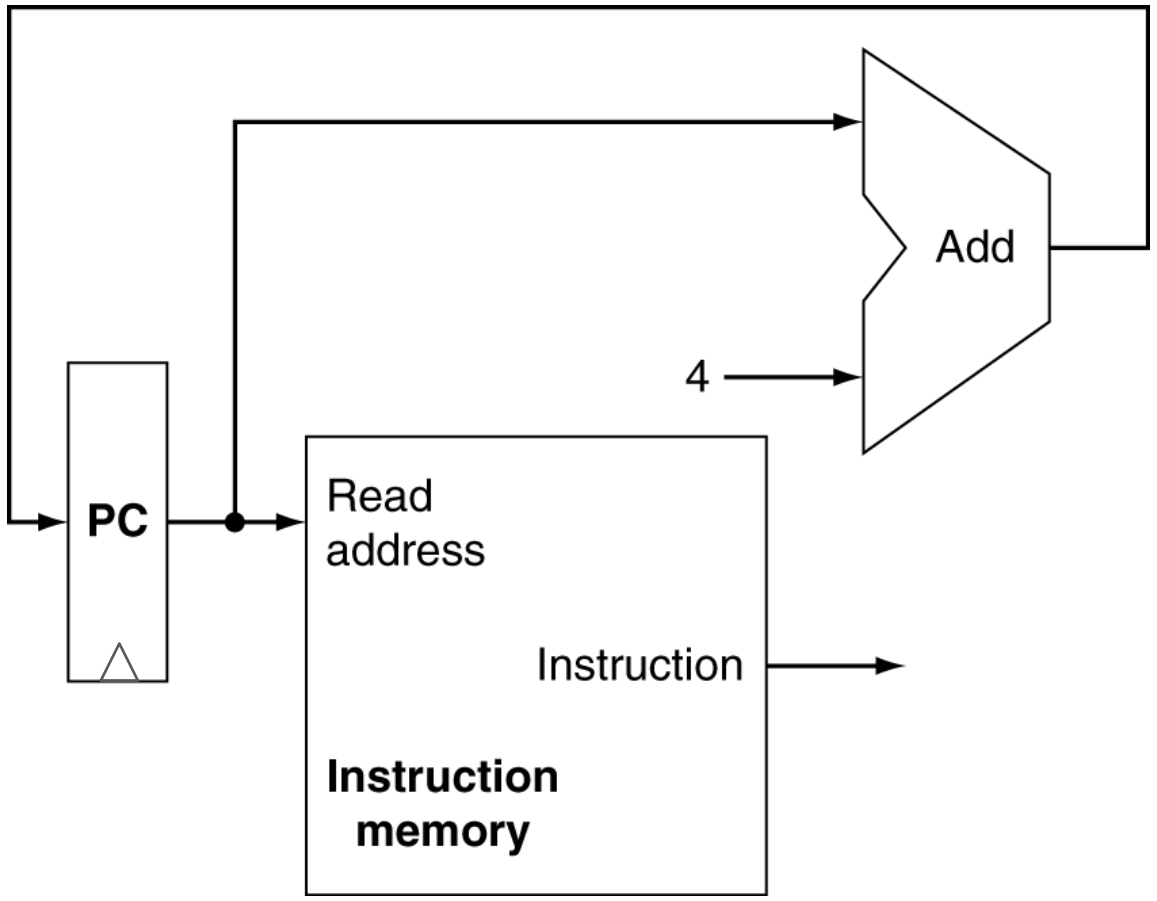
Criando o Caminho de Dados UNICICLO

- Elementos Básicos



a. Memória de instruções b. Contador de programa c. Somador

Etapa de Busca de Instruções



Instruções Lógico-Aritméticas

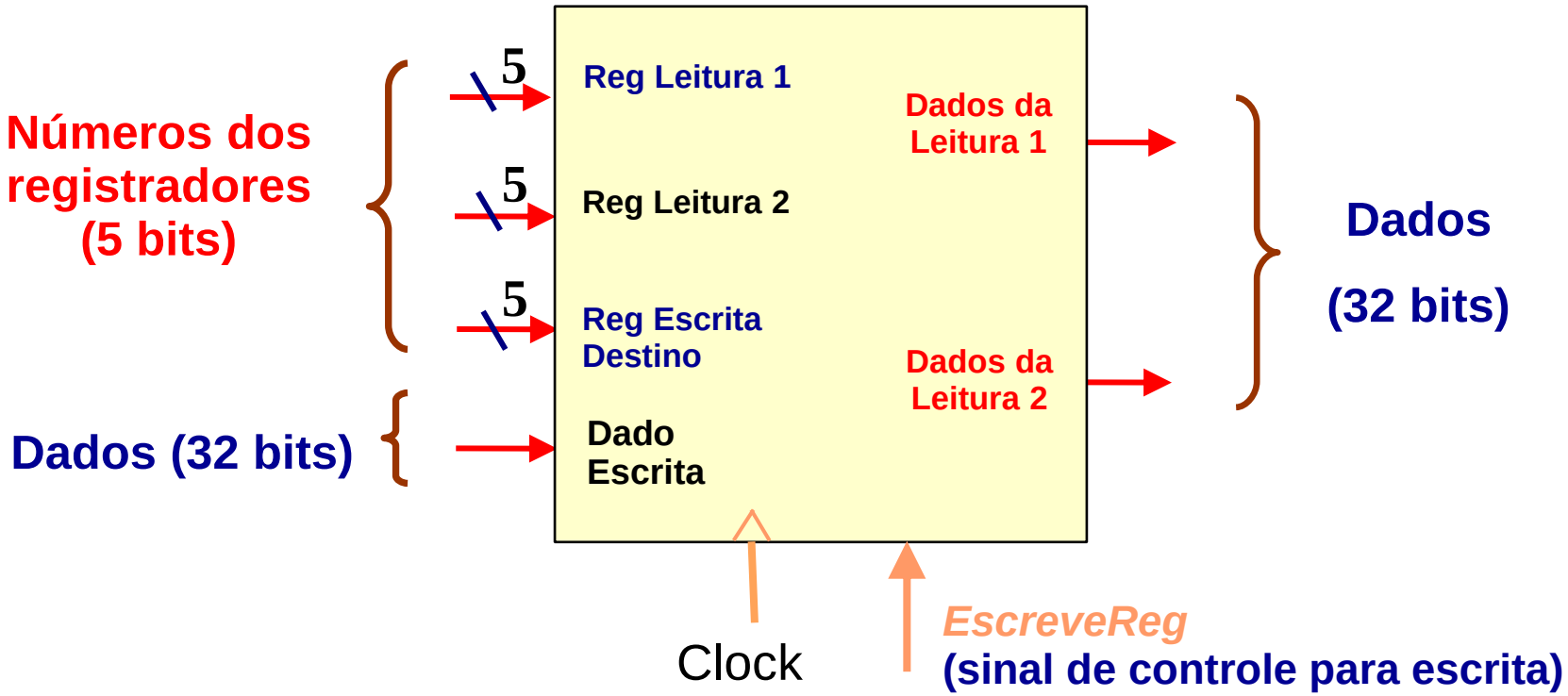
- Formato de uma instrução tipo R:

Campo	**funct7**	**rs2**	**rs1**	**funct3**	**rd**	**opcode**
Tamanho	*7 bits*	*5 bits*	*5 bits*	*3 bits*	*5 bits*	*7 bits*

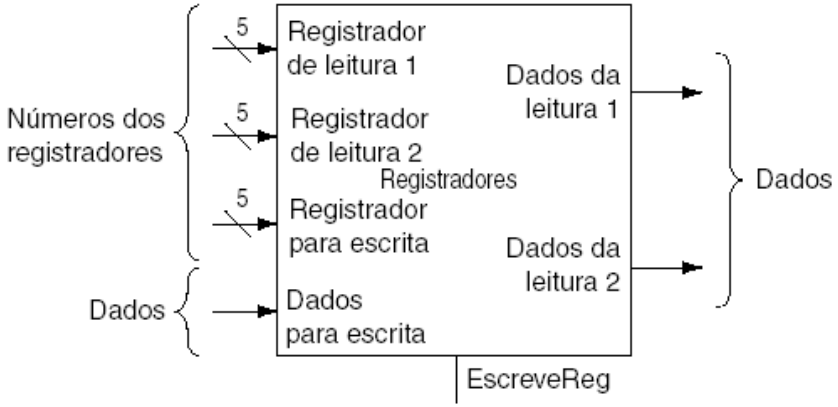
- Semântica:
  - rd ← op(rs1, rs2)
- Estrutura de suporte: banco de registradores

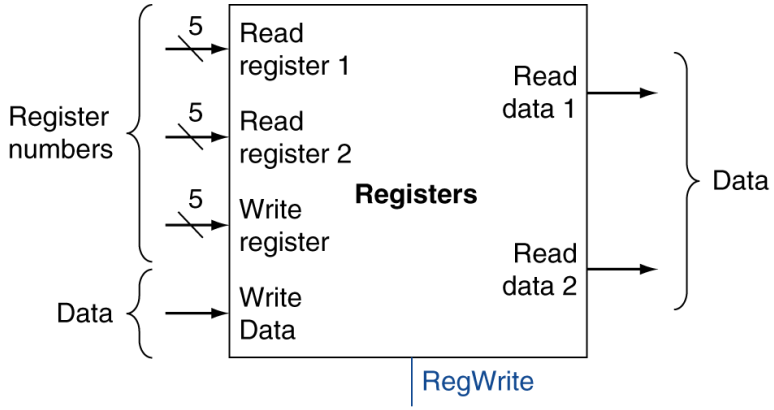
Banco de Registradores (*Register File*)

- Leitura de dois registradores ao mesmo tempo
- Escrita em um terceiro registrador
- Sinal de controle para escrita - leitura não necessita controle



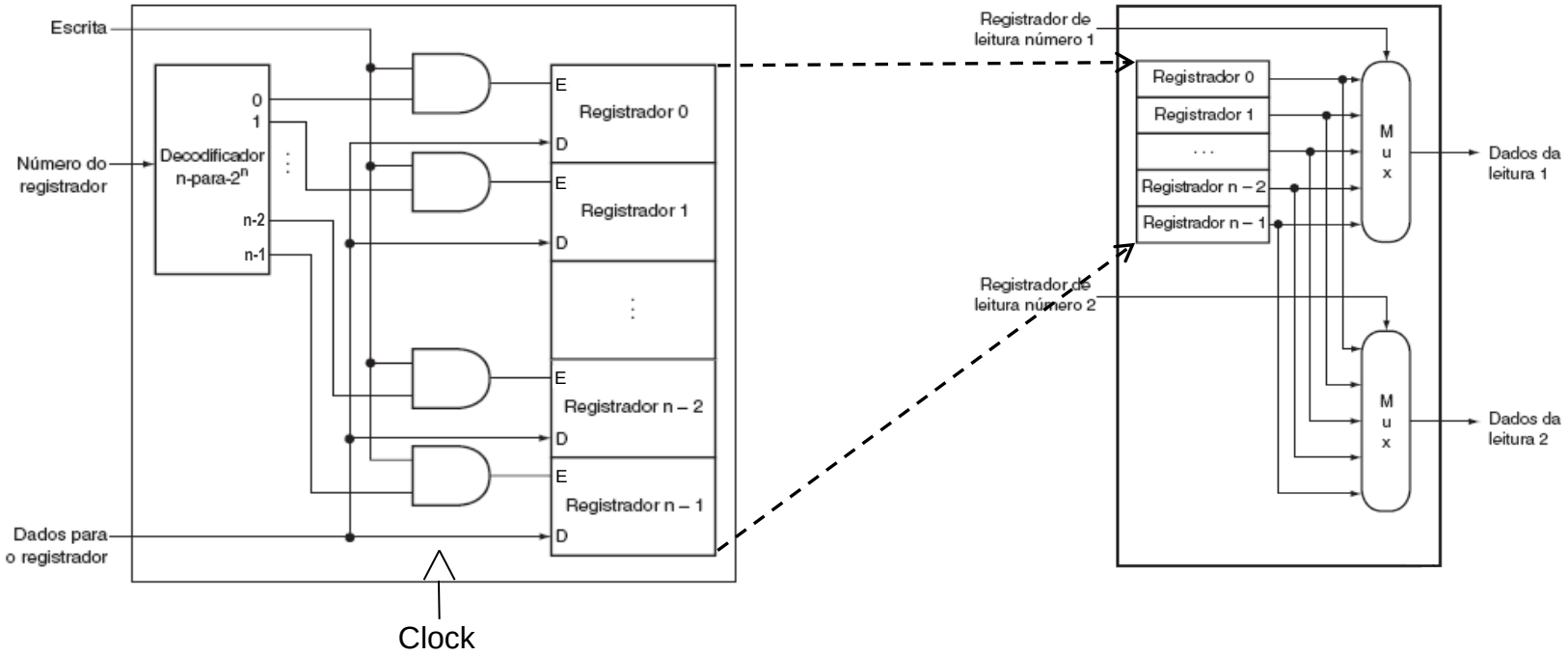
Banco de Registradores (*Register File*)





Escrita:

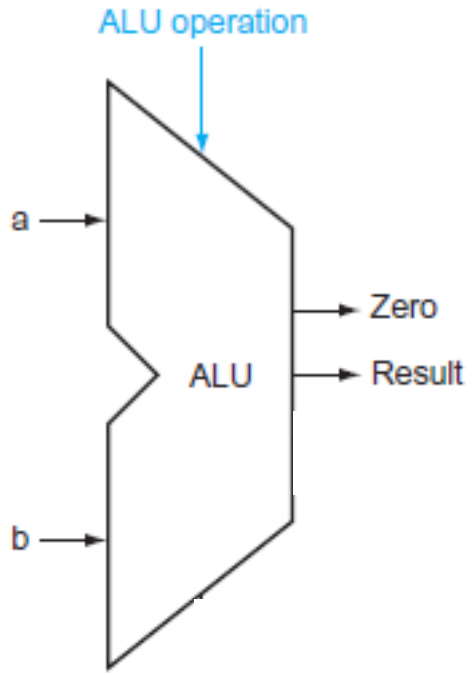
Leitura:



Unidade Lógico-Aritmética (ALU)

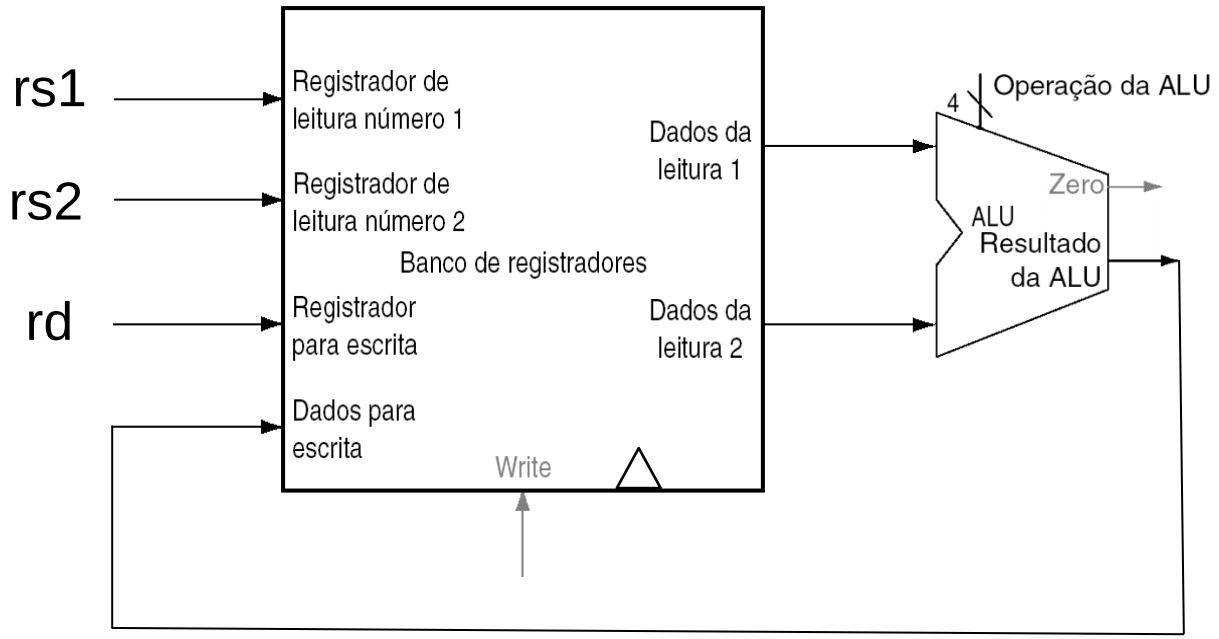
- A ULA foi desenvolvida no capítulo anterior
- 4 bits de controle indicam a operação a ser realizada

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set less than



Unidade Operativa: Instruções Tipo-R

funct7	rs2	rs1	funct3	rd	opcode
7 bits	*5 bits*	*5 bits*	*3 bits*	*5 bits*	*7 bits*



Ex.: add s1,s2,s3

funct7	rs2	rs1	funct3	rd	opcode
0x00	*0x13*	*0x12*	*0x0*	*0x09*	*0x33*

Unidade Operativa: Instruções Tipo-I

- No nosso caso simples: acesso a memória (lw):

Campo	**Imm[11:0]**	**rs1**	**funct3**	**rd**	**opcode**
Tamanho	*12 bits*	*5 bits*	*3 bits*	*5 bits*	*7 bits*

- Ex:
  - lw t0, 32(t1)
  - endereço = t1 + 32

Imm[11:0]	**rs1**	**funct3**	**rd**	**opcode**
0x20	*0x06*	*0x02*	*0x05*	*0x03*

Unidade Operativa: Instruções Tipo-S

- No nosso caso simples: acesso a memória (sw):

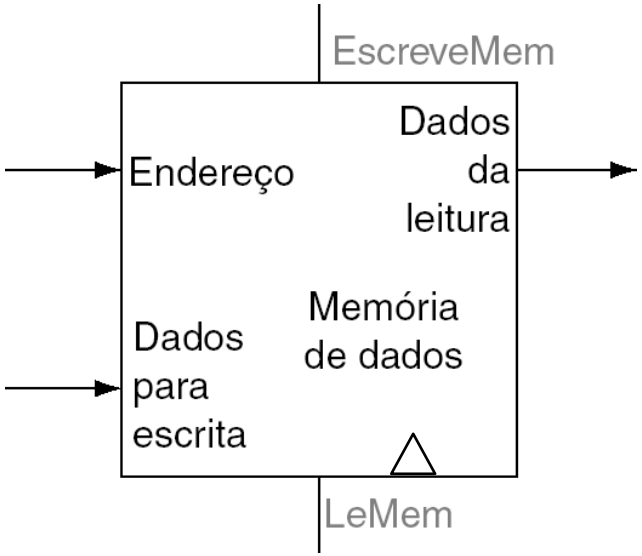
Campo	**Imm[11:5]**	**rs2**	**rs1**	**funct3**	**Imm[4:0]**	**opcode**
Tamanho	*7 bits*	*5 bits*	*5 bits*	*3 bits*	*5 bits*	*7 bits*

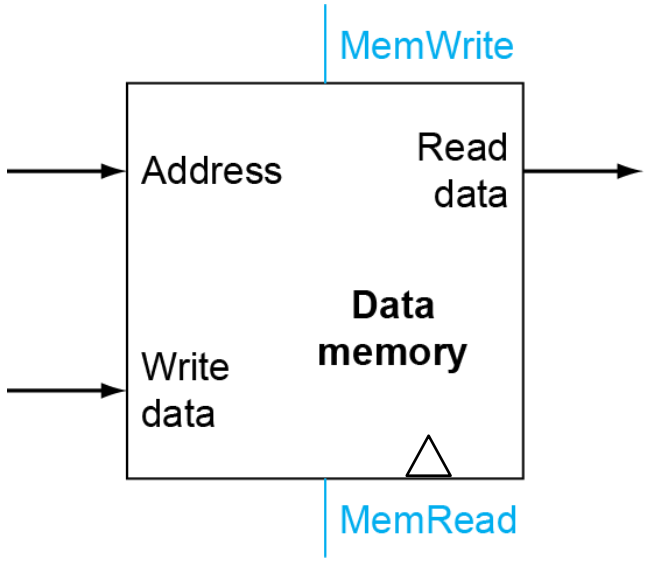
- Ex:
  - sw t0, 32(t1)
  - endereço = t1 + 32

Imm[11:5]	**rs2**	**rs1**	**funct3**	**Imm[4:0]**	**opcode**
0x01	*0x05*	*0x06*	*0x2*	*0x00*	*0x23*

Memória de Dados

- Memória com um barramento de entrada independente do de saída
- Controle de escrita (*write*) e leitura (*read*)
- Barramento de endereços
- Um acesso de cada vez



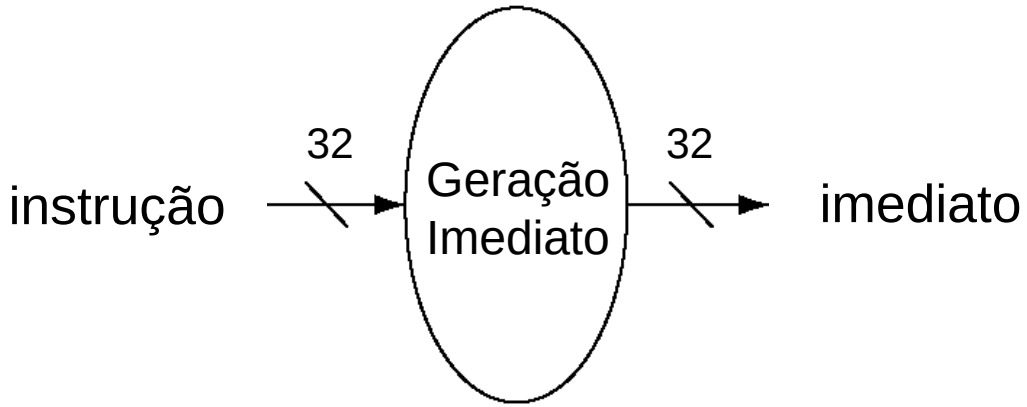


Unidade de geração do Imediato

- A forma de geração do imediato depende da instrução:
  - lw: Imediato = { 20{Instr[31]}, Instr[31:20] }
  - sw: Imediato = { 20{Instr[31]}, Instr[31:25], Instr[11:7] }
  - beq: Imediato = { 20{Instr[31]}, Instr[7], Instr[30:25], Instr[11:8] ,0}
  - jal: Imediato = { 12{Instr[31]}, Instr[19:12], Instr[20], Instr[30:21],0 }

    ps:Conferir!

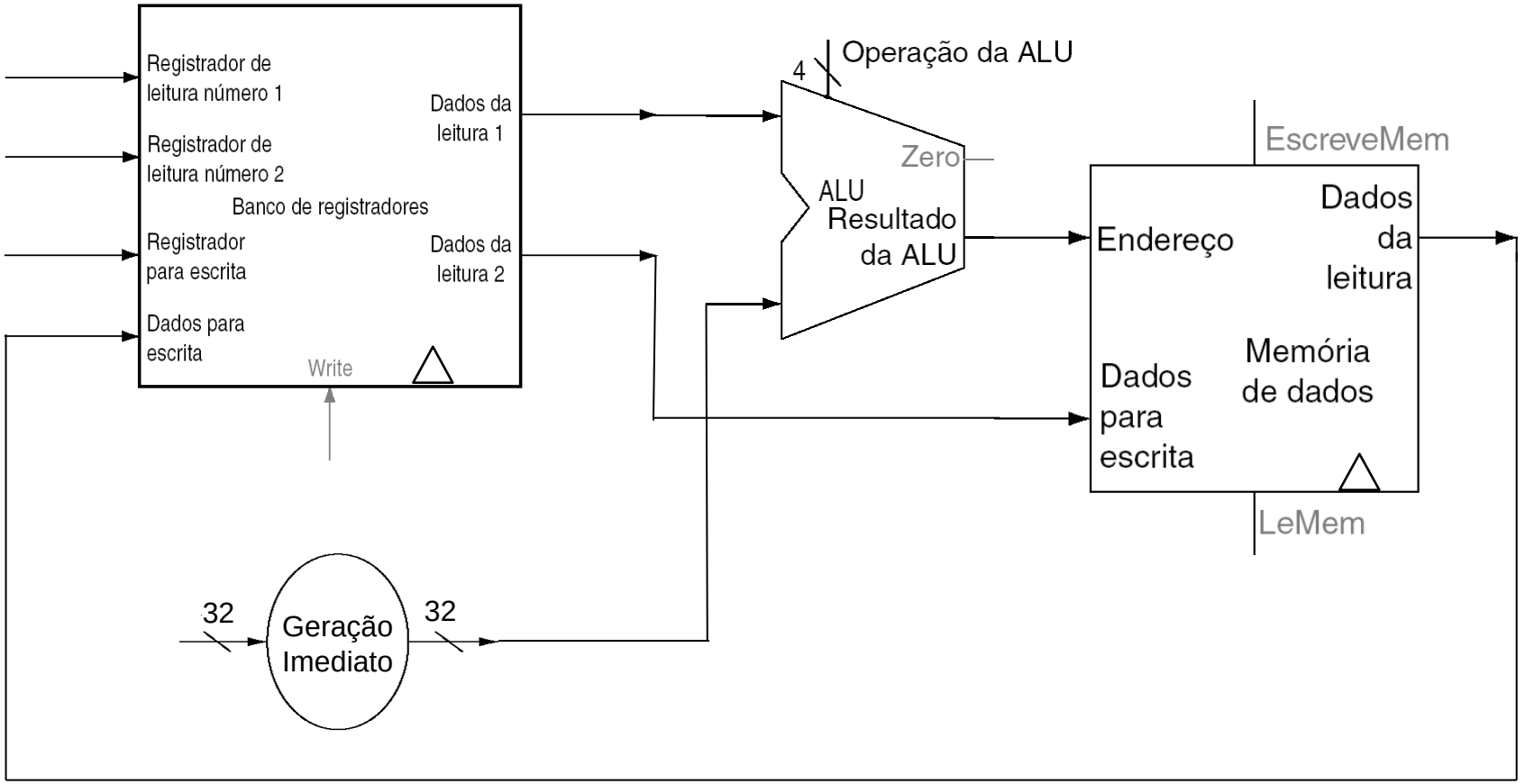
Obs.: se colocar 0 pode tirar o Shift left



Acesso à Memória

- lw lê um dado da memória e escreve em um registrador
  - conexão entre a saída da memória e a entrada do banco de registradores
- sw lê um dado de um registrador e escreve na memória
  - ligação entre saída do banco de registradores e entrada de dados da memória
- endereço calculado pela ULA
  - saída da ULA ligada ao barramento de endereços da memória

Caminho de Dados: Instruções de Acesso à Memória



Unidade Operativa: Instruções Tipo-B

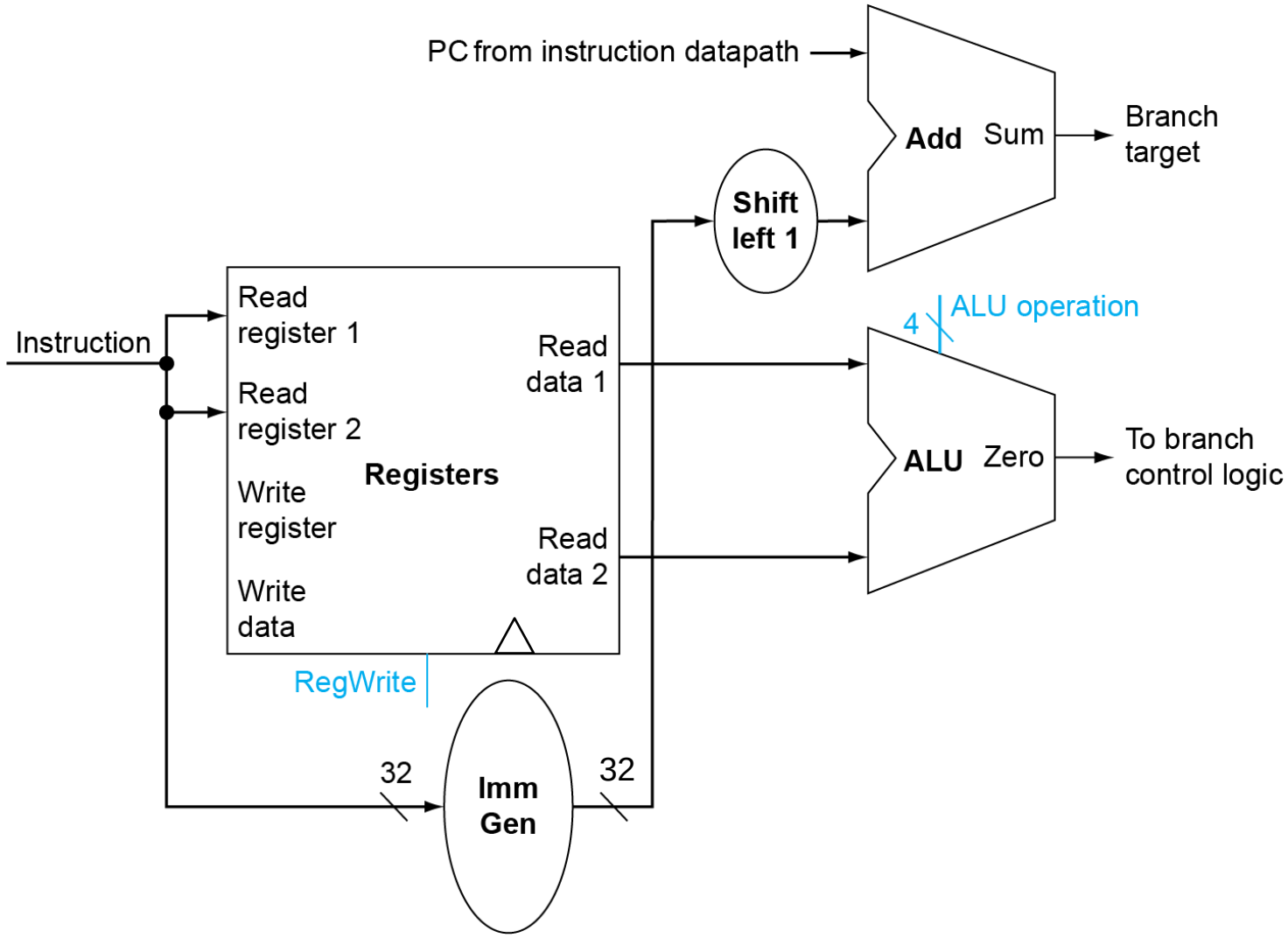
- No nosso caso : Desvio Condicional (beq):

Campo	**Imm[12,10:5]**	**rs2**	**rs1**	**funct3**	**Imm[4:1,11]**	**opcode**
Tamanho	*7 bits*	*5 bits*	*5 bits*	*3 bits*	*5 bits*	*7 bits*

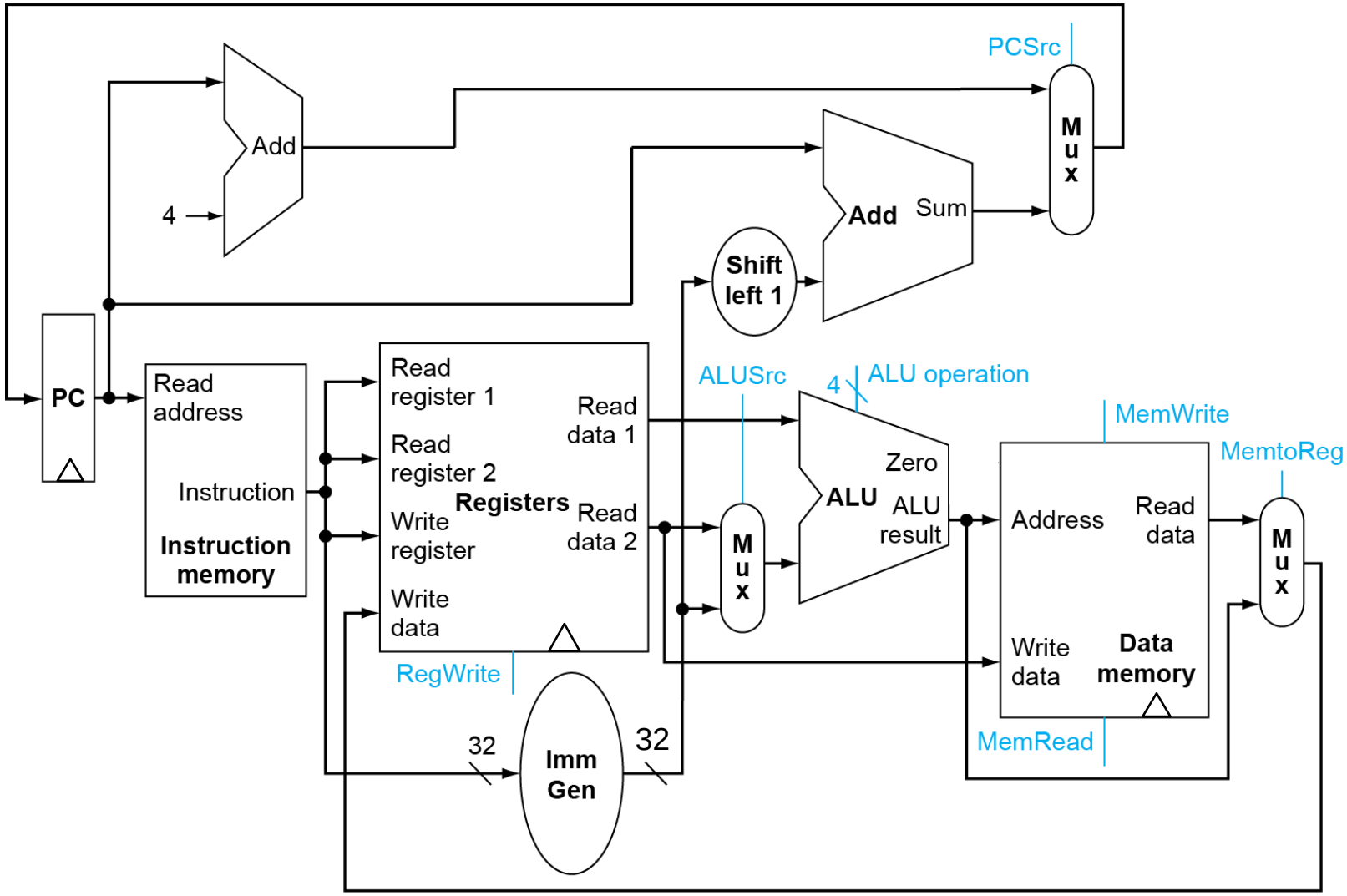
- Ex:
  - beq t0,t1, -12
  - Endereço de destino = PC + deslocamento em half-words ou PC + 4

Imm[12,10:5]	**rs2**	**rs1**	**funct3**	**Imm[4:1,11]**	**opcode**
0xFF	*0x06*	*0x05*	*0x0*	*0x15*	*0x63*

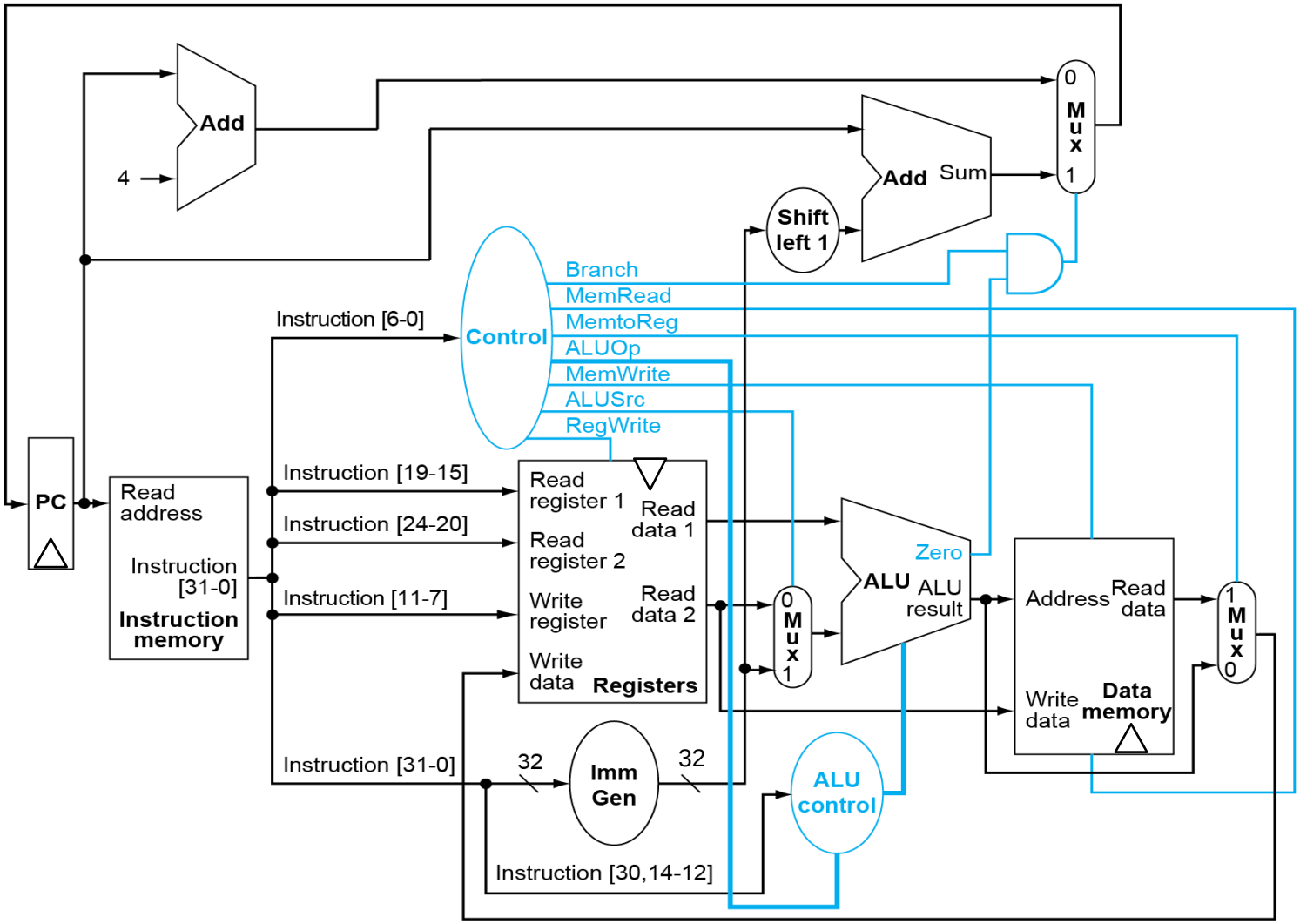
Caminho de Dados Instrução de Desvio Condicional



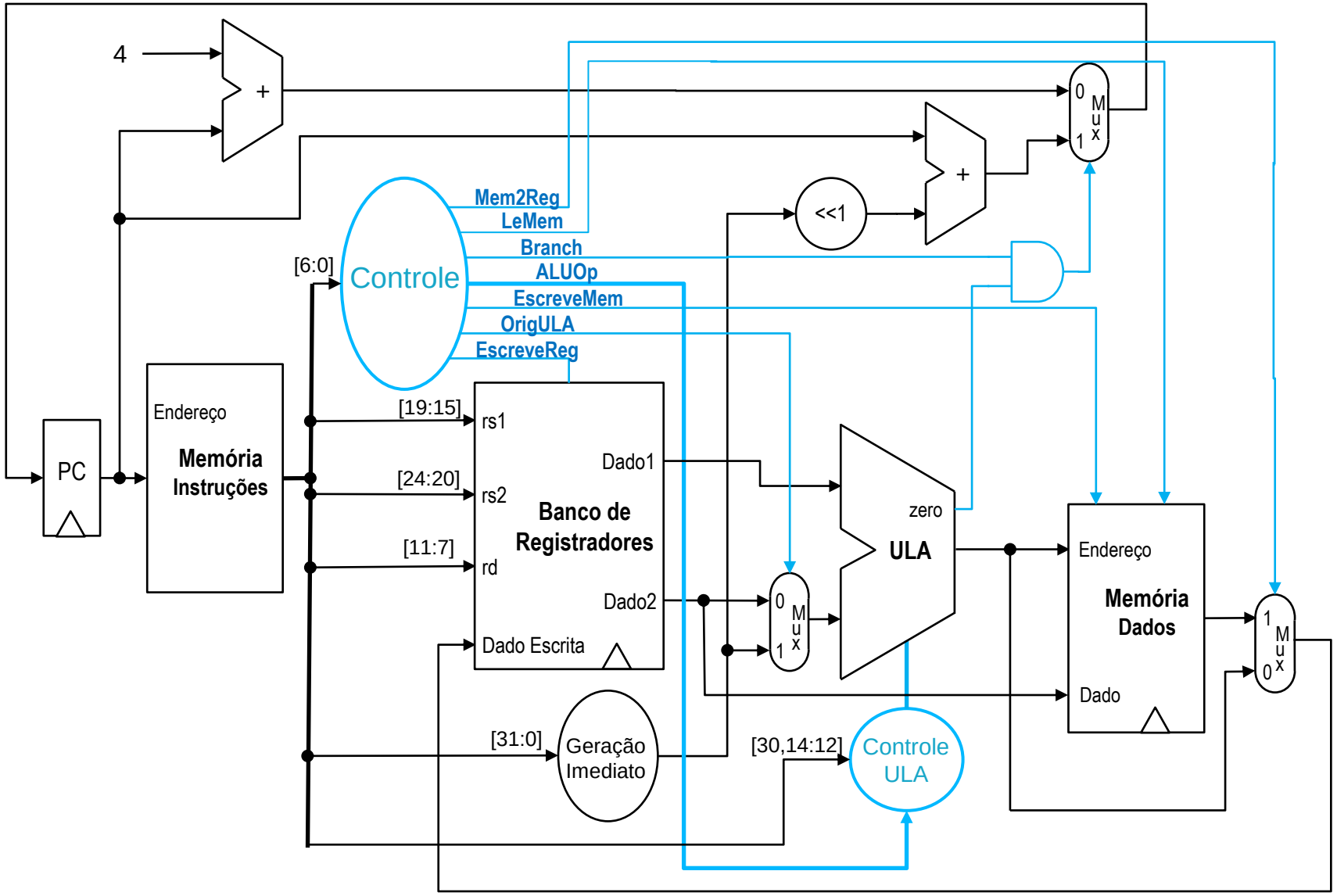
Caminho de Dados UNICICLO



Caminho de Dados UNICICLO



Caminho de Dados UNICICLO



Caminho de Dados UNICICLO - Laboratório