



Capítulo 4

O Processador

Introdução

- Fatores de desempenho da CPU
 - Contagem de instrução
 - Determinado pelo ISA e compilador
 - CPI e tempo de ciclo
 - Determinado pelo hardware da CPU
- Nós examinaremos duas implementações do MIPS
 - Uma versão simplificada
 - Uma versão de *pipeline* mais realista
- Subconjunto simples, mostra a maioria dos aspectos
 - Referência de memória: `lw`, `sw`
 - Aritmética/lógica: `add`, `sub`, `and`, `or`, `slt`
 - Transferência de controle: `beq`, `j`



®

Transferência de controle: `beq`, `j`

Execução da Instrução (1)

- Para cada instrução, os dois primeiros passos são idênticos:
 - Envia o contador de programa (PC) para a memória que contém o código
 - busca a instrução a partir daquela memória
 - Lê um ou dois registradores, usando os campos da instrução para selecionar os registradores para ler
 - Para a instrução de carregar a palavra, nós precisamos ler somente um registrador
 - A maioria das outras instruções exigem que se leia dois registradores
- Após estas etapas, as ações para completar a instrução depende da classe de instrução

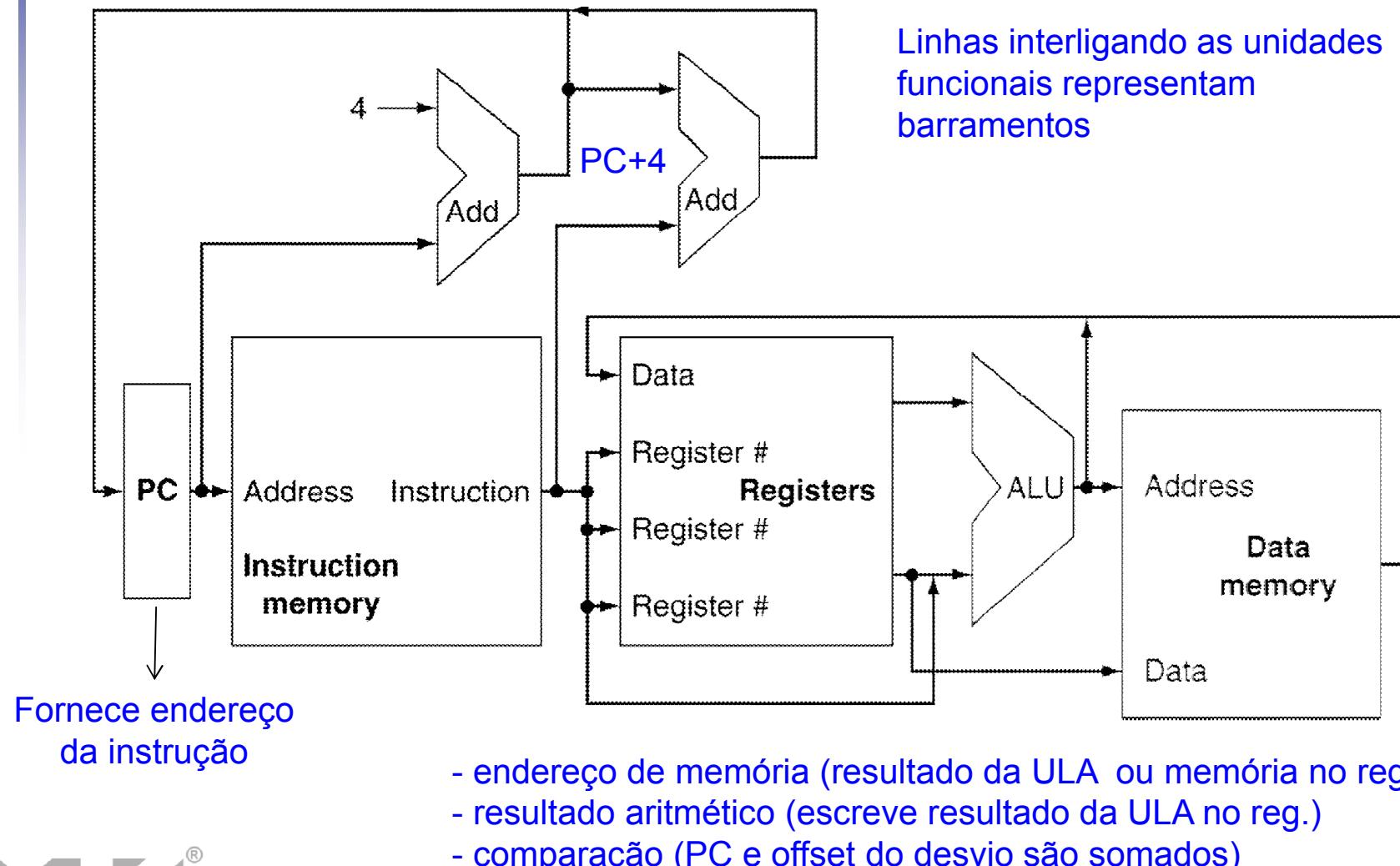


Execução da Instrução (2)

- Dependendo da classe de instrução
 - Usa a unidade lógica aritmética (ULA) para calcular
 - Endereço da memória para instruções *load/store*
 - Resultado aritmético para a execução da operação
 - Comparações para o endereço alvo de desvio
 - Instrução de referência de memória precisa ter acesso a memória para ler/escrever dados
 - Instruções de lógica aritmética deve escrever dados a partir da ULA ou memória no registrador
 - Instruções de desvio precisam alterar o próximo endereço de instrução baseado na comparação
 - $PC \leftarrow \text{endereço alvo ou } PC + 4$

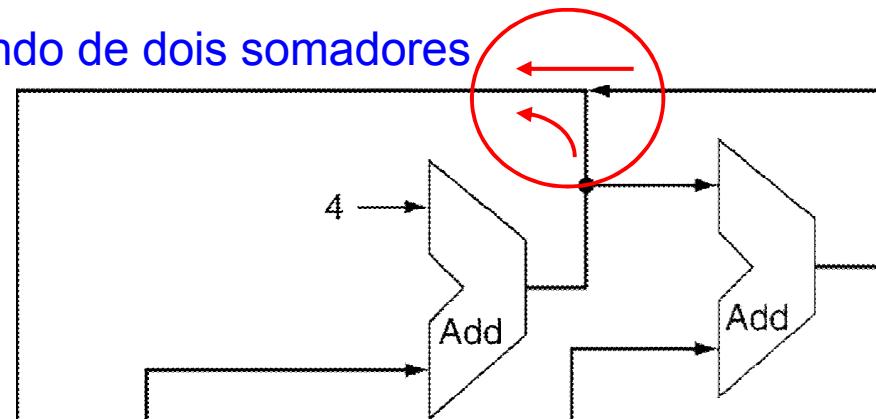


Visão Geral da CPU



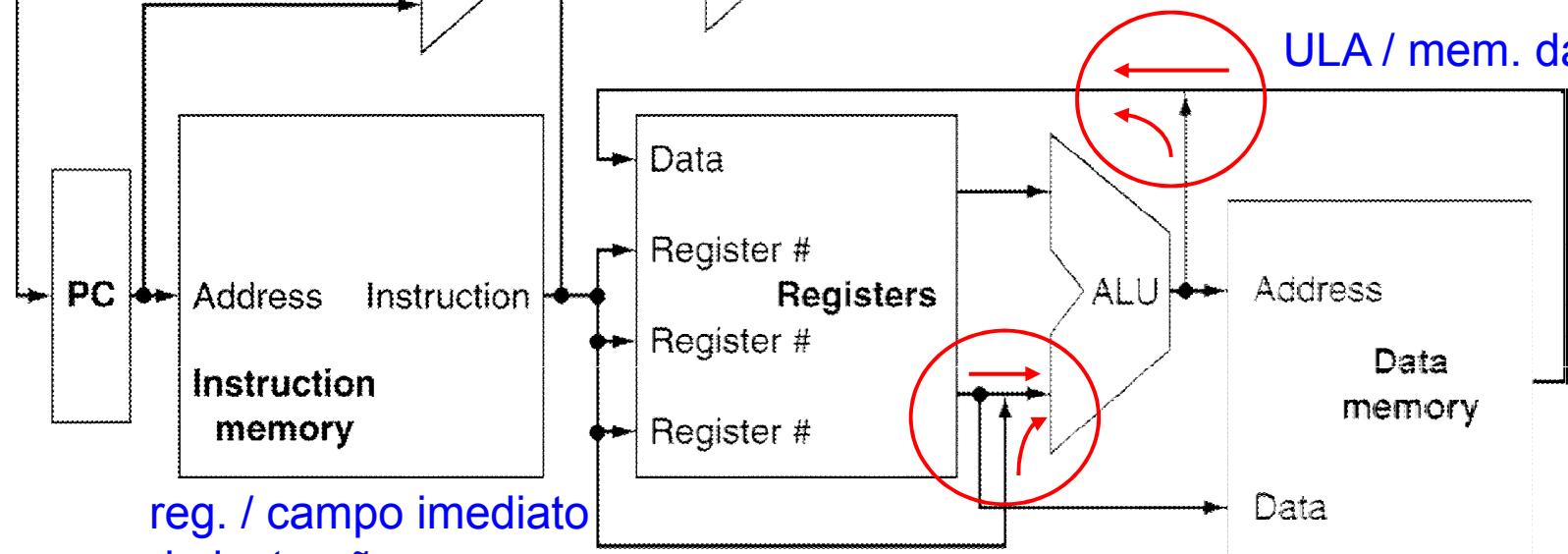
Multiplexadores

Vindo de dois somadores



- Não pode simplesmente juntar os fios
 - Use multiplexadores

ULA / mem. dados



reg. / campo imediato da instrução

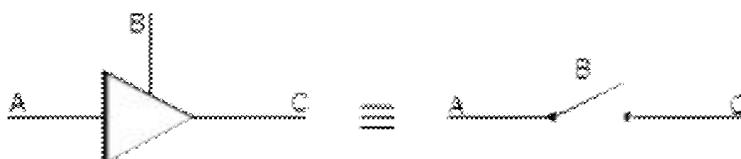
Deve-se acrescentar um elemento lógico que escolhe entre as várias fontes e dirige uma dessas fontes para o seu destino



Exercício

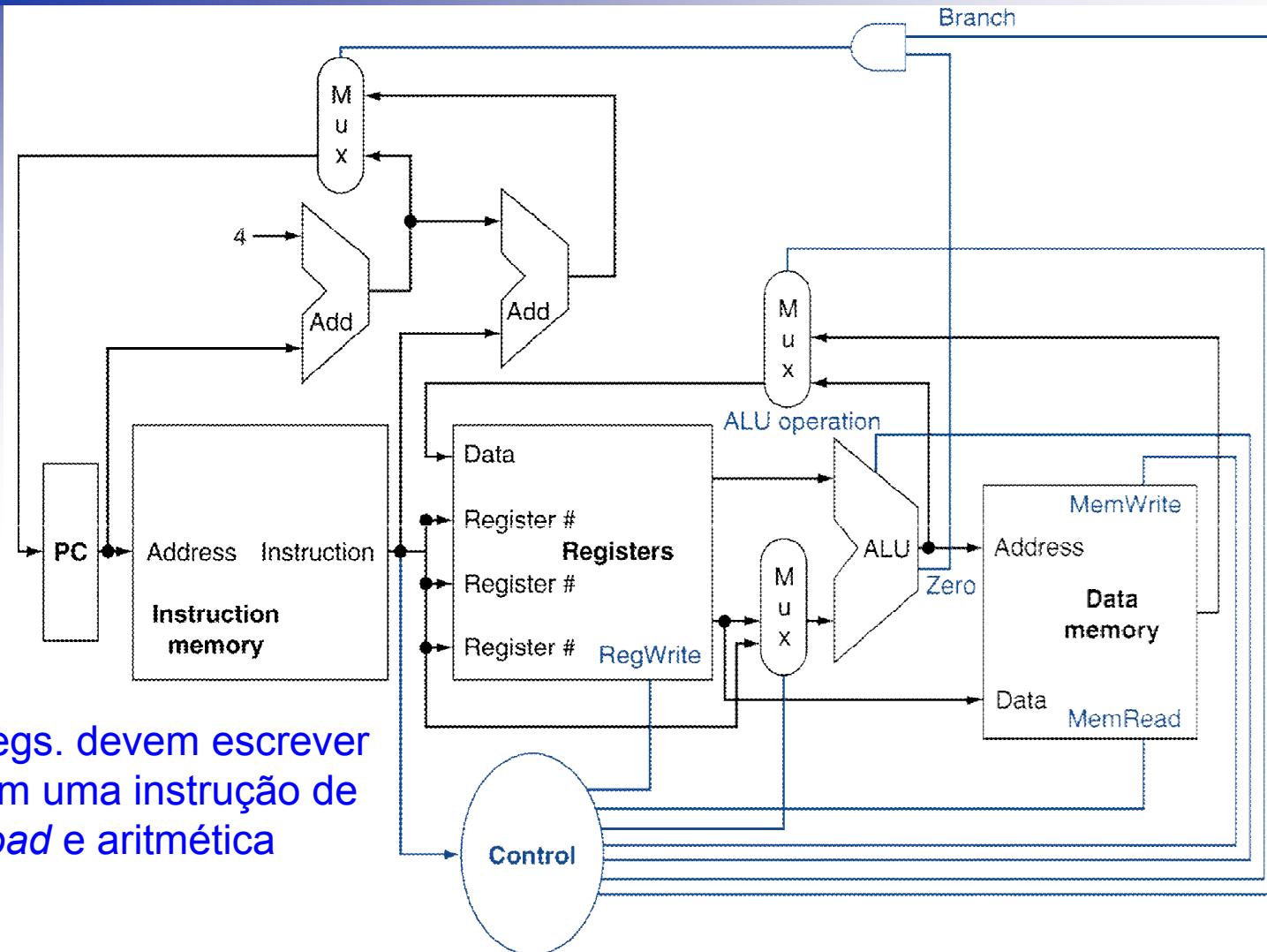
- Considere um pequeno multiplexador com 4 entradas de dados de 1 bit
 - Este bloco contém somente lógica, somente flip-flops, ou ambos?
 - Mostre como este bloco pode ser implementado. Use somente AND, OR, NOT
 - Estenda a letra b) com buffers tri-state

A intenção do estado Z é permitir que diversos circuitos compartilhem o mesmo barramento de dados, sem afetar umas as outras



Entrada	Saída
A 0	B 0
1	Z
0	1
1	1

Controle



regs. devem escrever
em uma instrução de
load e aritmética

Memória de
dados deve
ler em um
load e
escrever
em um
store

estas operações são dirigidas pelas linhas de controle que são
definidas em função de vários campos da instrução



Exercício

- Diferentes instruções utilizam diferentes blocos de hardware na implementação básica do ciclo único

	Instrução	Interpretação
a.	AND Rd, Rs, Rt	$\text{Reg}[\text{Rd}] = \text{Reg}[\text{Rs}] \text{ AND } \text{Reg}[\text{Rt}]$
b.	SW Rt, offs(Rs)	$\text{Mem}[\text{Reg}[\text{Rs}] + \text{offs}] = \text{Reg}[\text{Rt}]$

- Quais são os valores dos sinais de controle gerados pelo controle do slide anterior?
- Quais recursos (blocos) executam uma função útil?



Fundamentos de Lógica

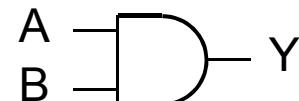
- Os dados são codificados em binário
 - Baixa tensão = 0, Alta tensão = 1
 - Um fio por *bit*
 - Dados de múltiplos bits codificados em barramentos de múltiplos fios
- Elemento combinacional
 - Opera em dados
 - Saída é uma função da entrada
- Elementos de estado (sequencial)
 - Armazena dados



Elementos Combinacionais

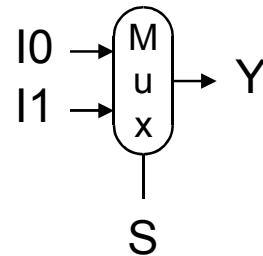
- Porta AND

- $Y = A \& B$



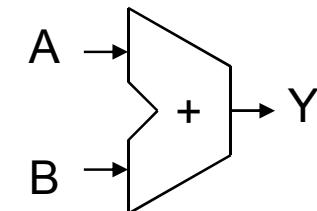
- Multiplexador

- $Y = S ? I_1 : I_0$



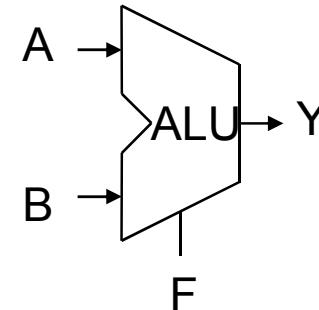
- Somador

- $Y = A + B$



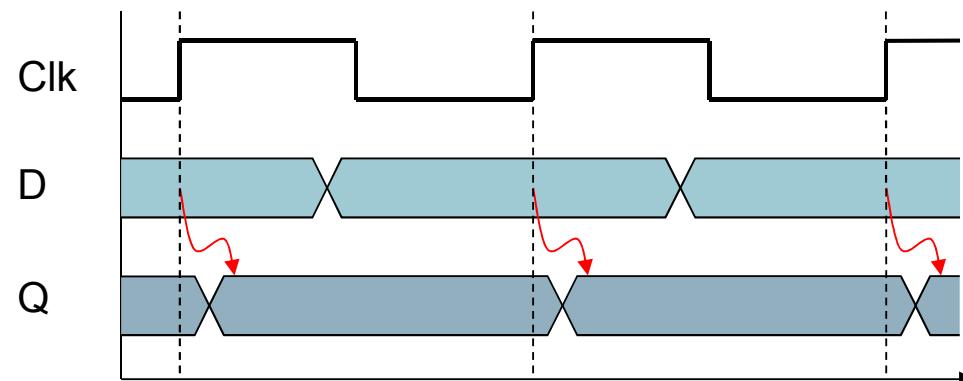
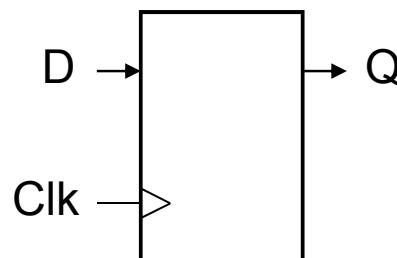
- Unidade lógica/aritmética

- $Y = F(A, B)$



Elementos Sequenciais (1)

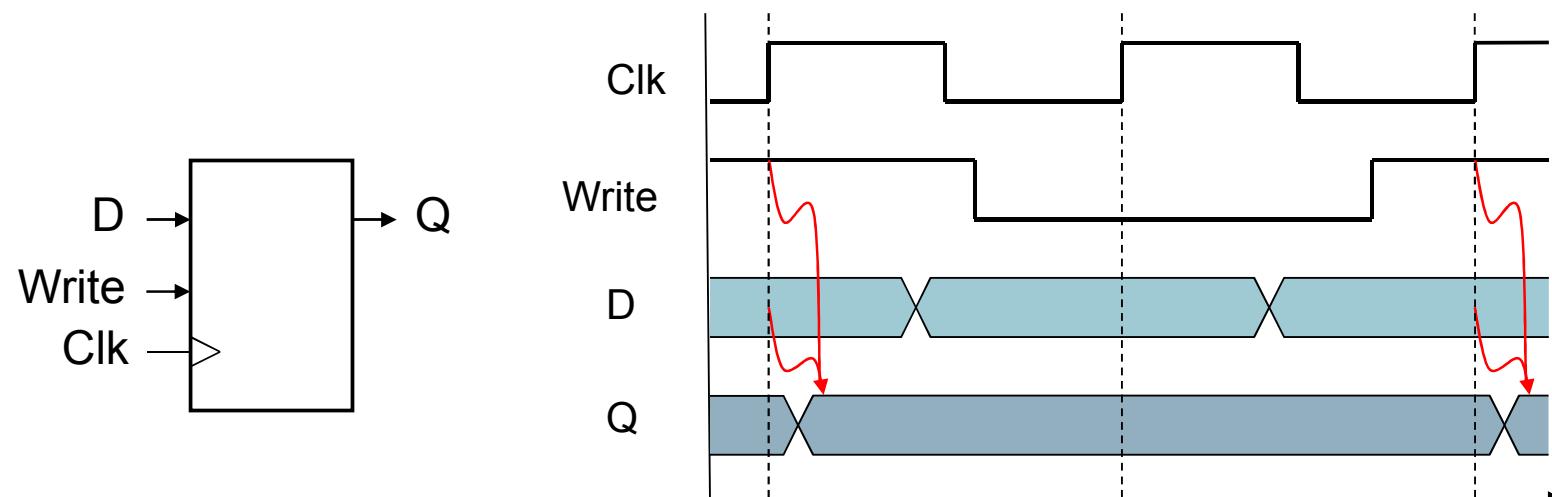
- Registrador: armazena dados num circuito
 - Usa um sinal de relógio para determinar quando atualizar o valor armazenado
 - Disparado pela borda: atualiza quando Clk muda de 0 para 1



Clk	D	Q	Comentário
0	X	Q_{ant}	Nenhuma mudança
1	0	0	Reseta
1	1	1	Seta

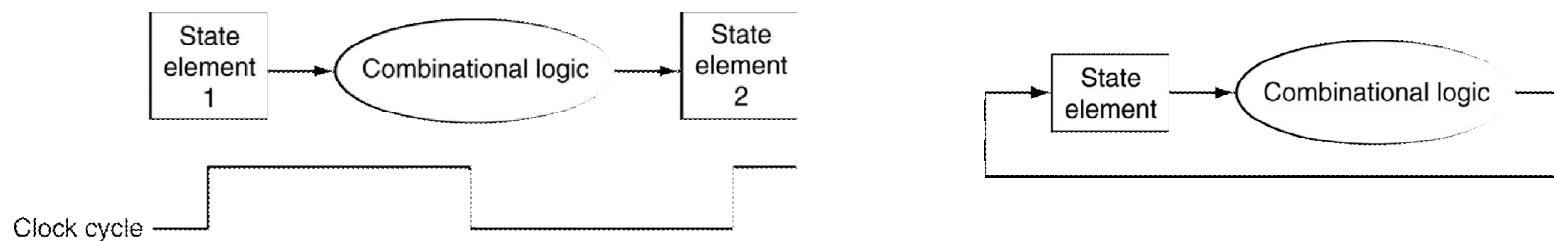
Elementos Sequenciais (2)

- Registrador com controle de escrita
 - Somente atualiza na borda do relógio quando a entrada do controle de escrita é 1
 - Usado quando o valor armazenado é solicitado mais tarde



Metodologia de Relógio

- Lógica combinacional transforma dados durante ciclos de relógio
 - Entre as bordas do relógio
 - Entrada a partir de elementos de estado, saída para elemento de estado
 - Atraso mais longo determina o período de relógio



Uma metodologia de disparo de bordas permite que um elemento de estado possa ser lido e escrito no mesmo ciclo de relógio, sem criar uma corrida que poderia levar a valores de dados indeterminados

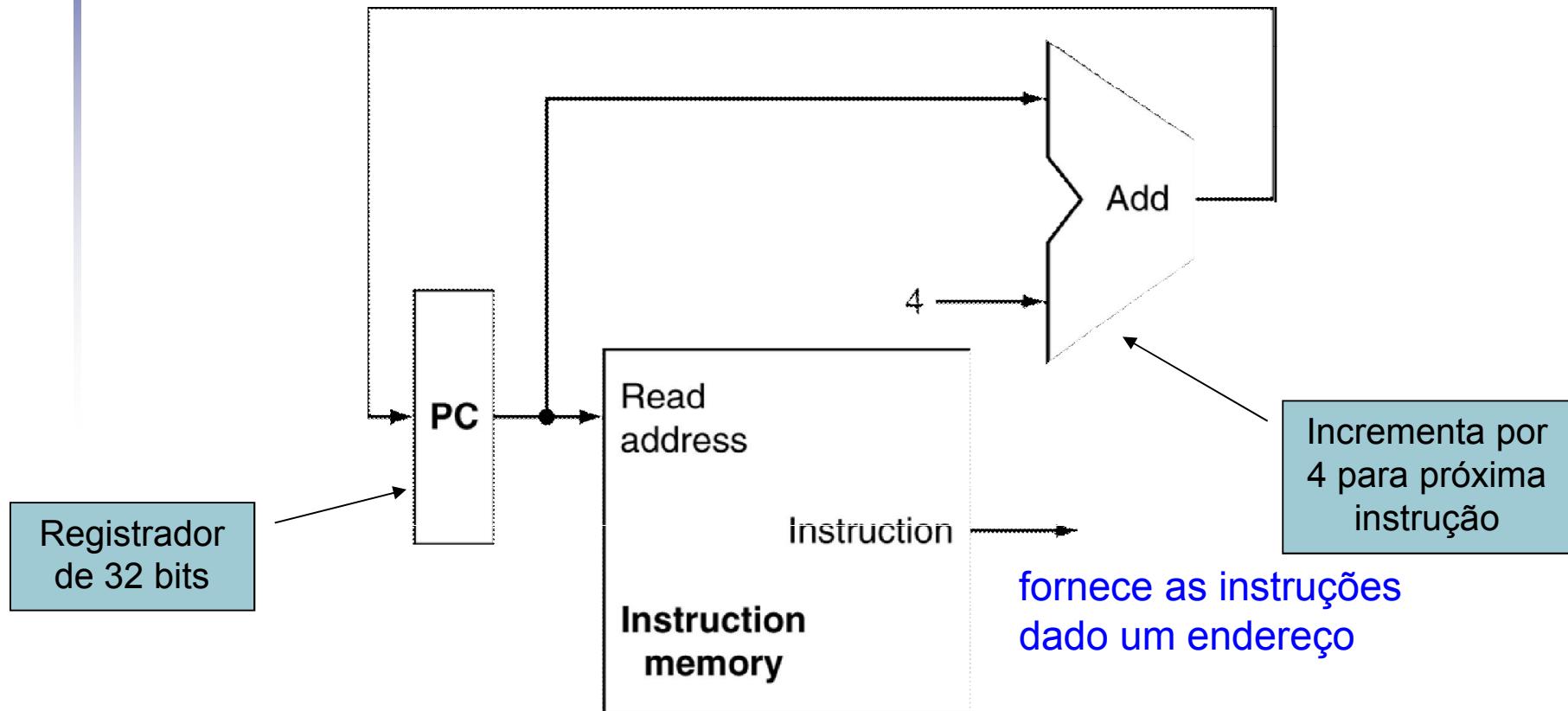


Construindo Caminho de Dados

- Caminho de dados
 - Elementos que processam dados e endereços na CPU
 - Registradores, ULAs, muxes, memórias, ...
- Nós vamos construir um caminho de dados do MIPS de forma incremental
 - Refinando o *design* geral
 - Unidade de memória: armazena as instruções de um programa e fornece as instruções dado um endereço
 - Contador de programa (PC): armazena o endereço da instrução atual
 - Somador: incrementa o endereço de PC para o endereço da próxima instrução

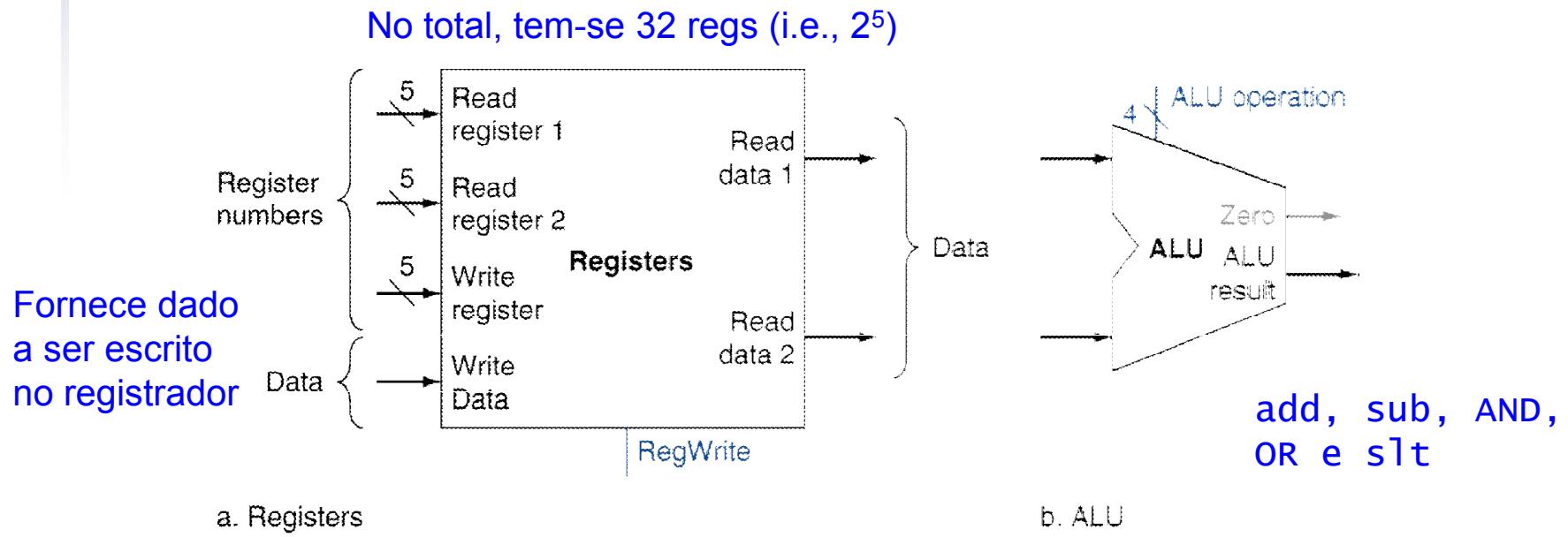


Busca Instrução



Instruções do Format-R

- Lê dois operandos de registrador (*Read Register*)
- Executa operação lógica/aritmética
- Escreve resultado no registrador (*Write Register*)

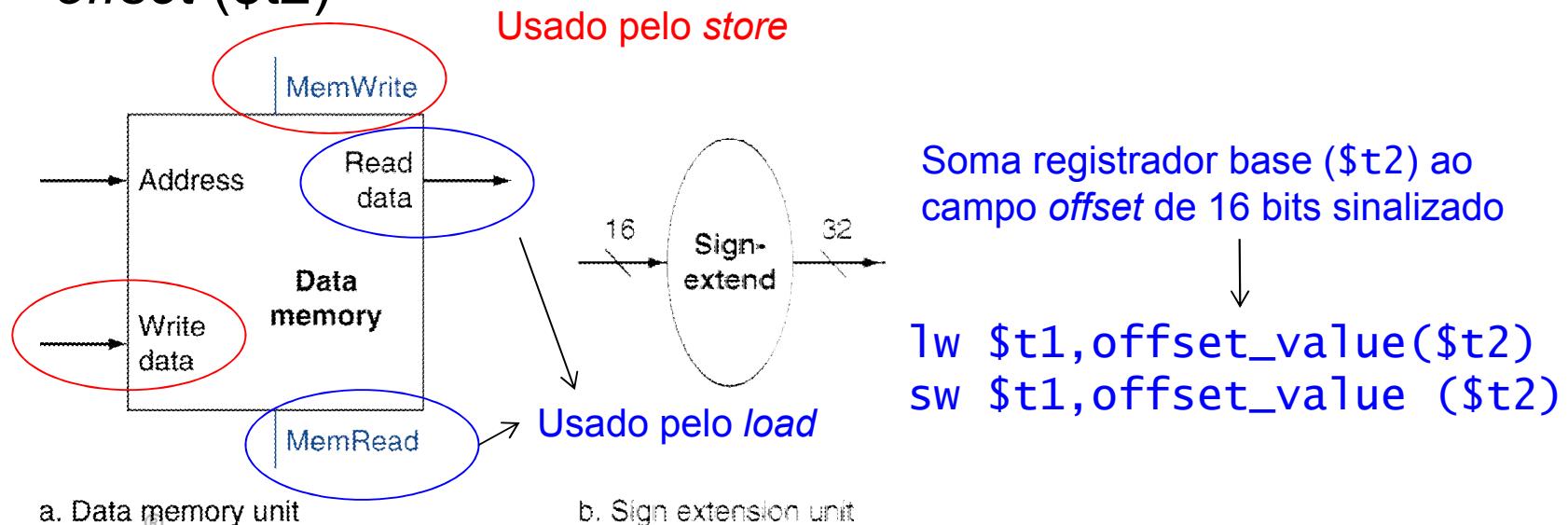


O barramento da entrada e saída de dados são de 32 bits



Instruções de Load/Store

- Lê operandos do registrador
- Calcula endereço usando um deslocamento de 16 bits
 - Usa ULA, mas com deslocamento de sinal estendido
- *Load*: Lê memória *offset* (\$t2) e atualiza registrador (\$t1)
- *Store*: Escreve valor do registrador (\$t1) para memória *offset* (\$t2)

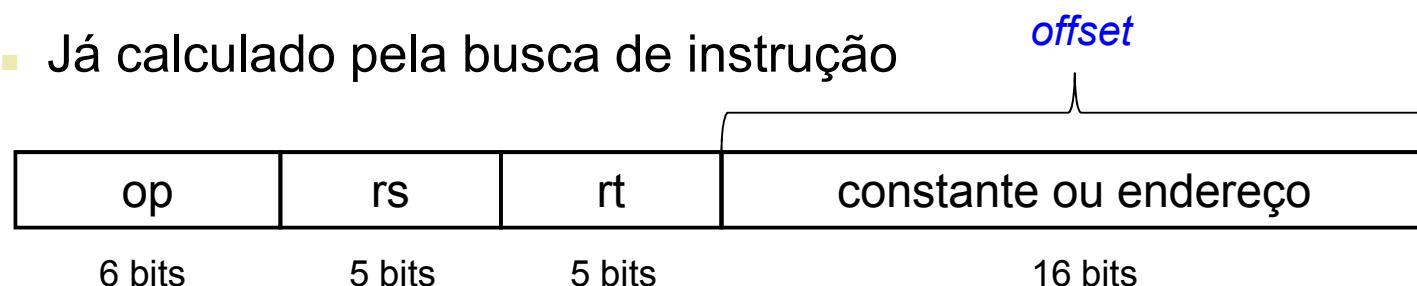


Instruções de Desvio (1)

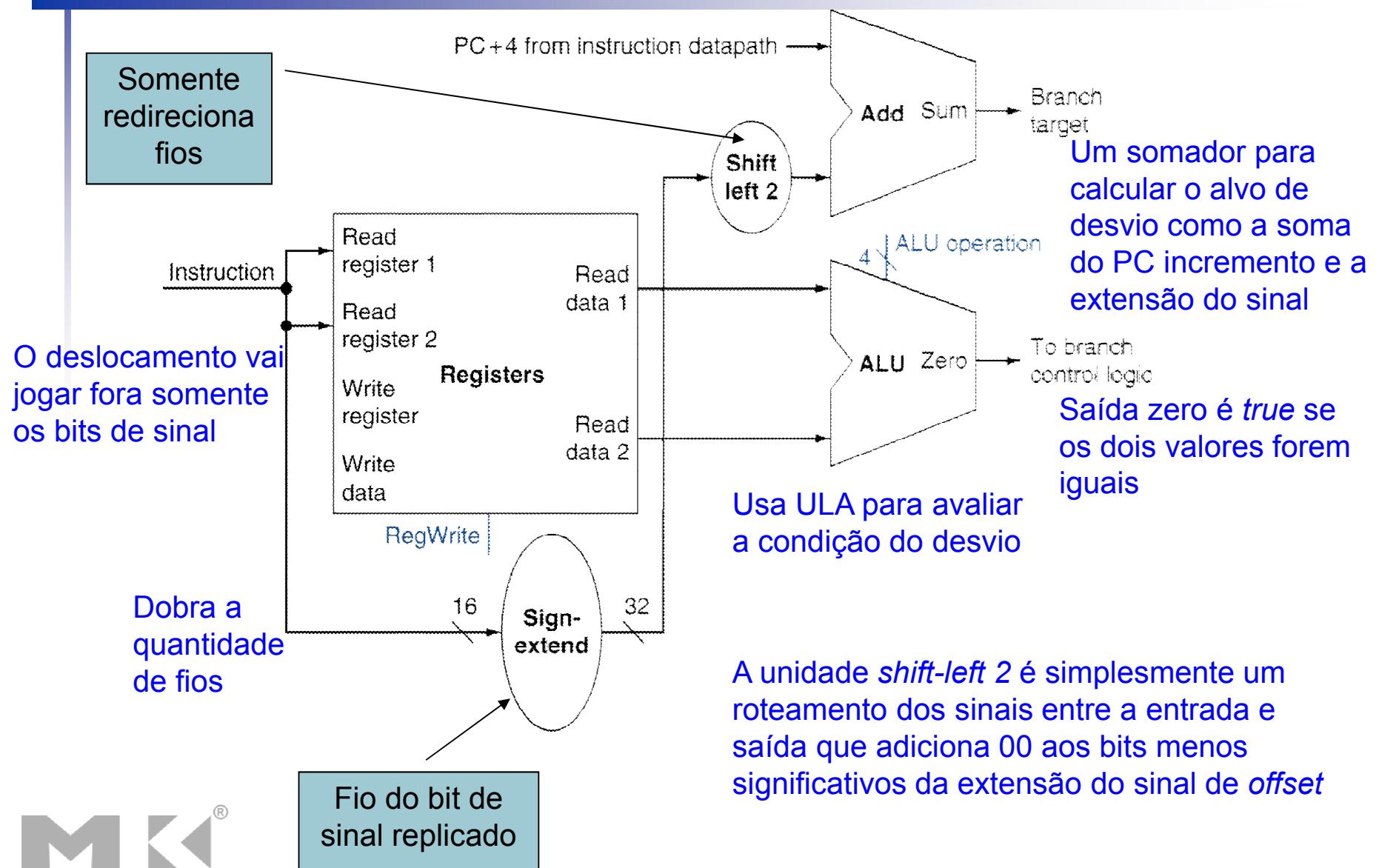
- Lê operandos do registrador
- Compara operandos
 - Usa ULA, subtrai e checa saída Zero
- Calcula endereço alvo
 - Deslocamento de extensão do sinal
 - Desloca 2 casas para esquerda (aumenta o intervalo efetivo do *offset* por um fator de 4)
 - Adiciona ao PC + 4
 - Já calculado pela busca de instrução

beq \$t1,\$t2,offset

offset de 16
bits sinalizado



Instruções de Desvio (2)

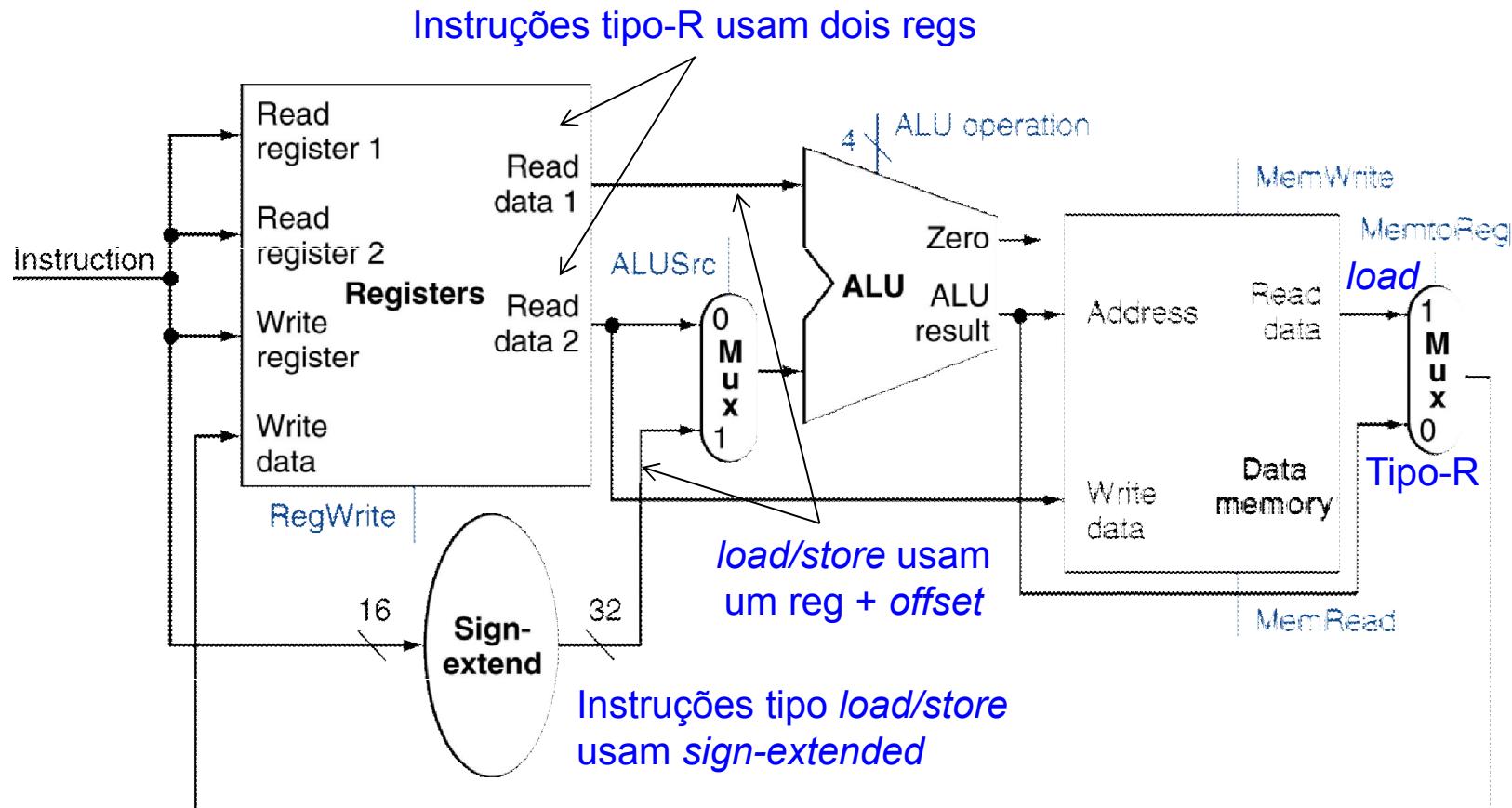


Compondo os Elementos

- O caminho de dados tenta **executar** todas as instruções em um **único ciclo de relógio**
 - Nenhum dos recursos do caminho de dados pode ser usado mais de uma vez por cada instrução
 - de modo que, qualquer elemento necessário mais do que uma vez, deve ser duplicado
 - Por isso, precisamos de uma memória de instruções separada da memória de dados
- **Compartilhar elemento** do caminho de dados entre duas **diferentes classes** de instrução
 - Usar **multiplexadores** e **sinal de controle** para selecionar entre as múltiplas entradas

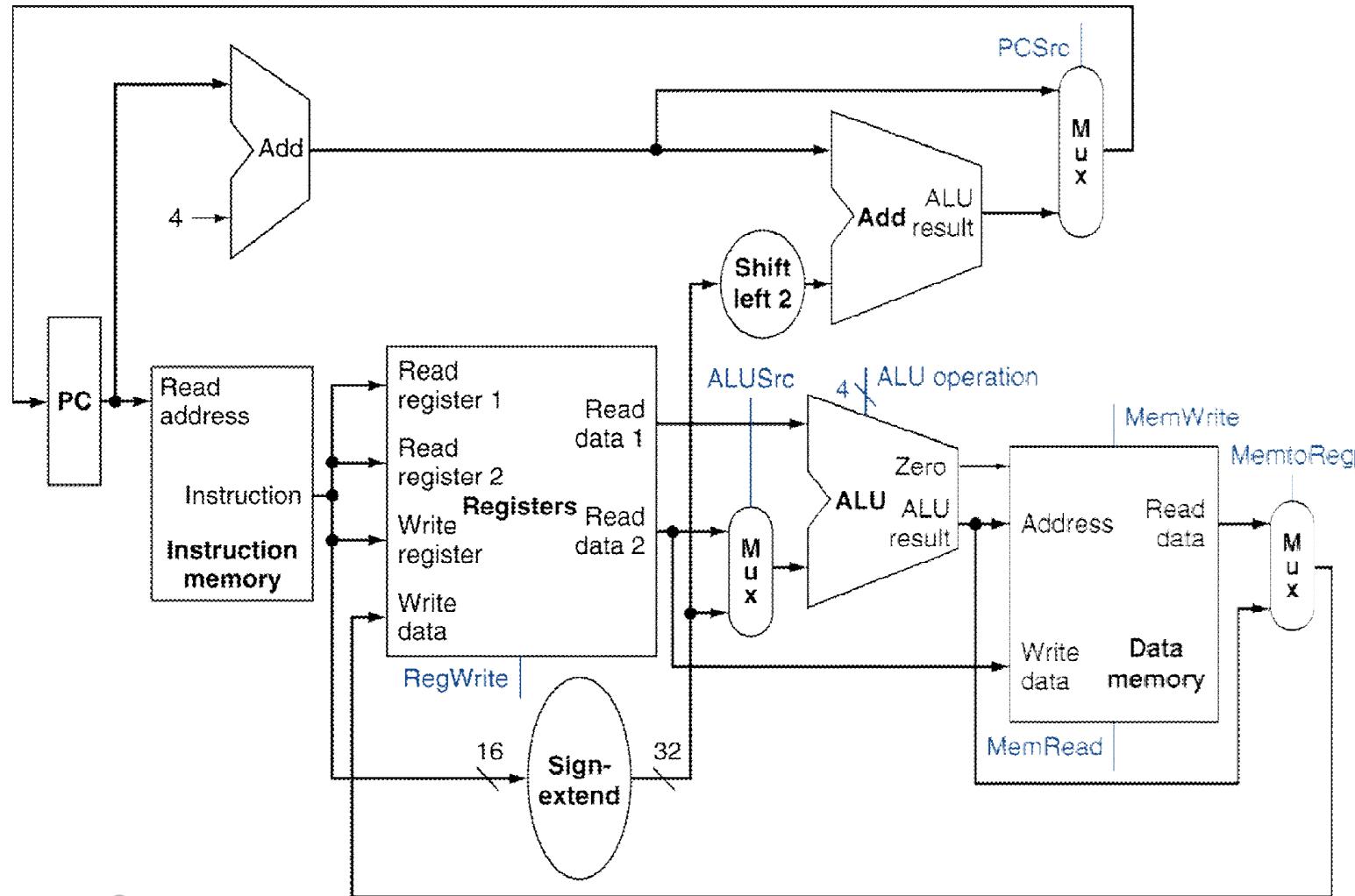


Caminho de Dados Tipo-R/Load/Store



O valor armazenado no registrador de destino vem da ULA (p/ uma instrução do tipo-R) ou da memória (p/ um instrução do tipo load)

O Caminho de Dados Completo



Exercício 1

- Quais das seguintes frases estão corretas para uma instrução *load*? Consulte o slide anterior
 - a) *MemtoReg* deve ser setado para fazer com que o dado da memória seja enviado para o arquivo de registrador
 - b) *MemtoReg* deve ser setado para fazer com que o registrador de destino correto seja enviado para o arquivo de registrador
 - c) Nós não nos importamos sobre a definição de *MemtoReg* para *loads*



Exercício 2

- O caminho de dados de ciclo único, descrito conceitualmente até aqui, deve ter memória de instruções e dados separada, porque
 - a) O formato das instruções e dos dados são diferentes no MIPS, e deste modo diferentes memórias são necessárias
 - b) Tendo memórias separadas é menos dispendioso
 - c) O processador opera em um ciclo e não pode usar uma memória de única porta para dois acessos diferentes dentro desse ciclo



Exercício 3

- Diferentes unidades de execução e blocos de lógica digital têm diferentes latências (tempo necessário para realizar o trabalho).

	I-Mem	Add	Mux	ULA	Regs	D-Mem	Control
a.	200ps	70ps	20ps	90ps	90ps	250ps	40ps
b.	750ps	200ps	50ps	250ps	300ps	500ps	300ps

Qual é o tempo necessário para executar as instruções AND , LOAD e BEQ? Responda esta questão com base no circuito do slide 8



Controle da ULA (1)

- ULA usada para
 - Load/Store: F = adiciona
 - Desvio: F = subtrai
 - Tipo-R: F = depende do campo *funct*

Controle da ULA	Função
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR



Controle da ULA (2)

- Assume ALUOp de 2-bit derivado a partir do *opcode*
 - Lógica combinacional deriva o controle da ULA

opcode	ALUOp	Operação	funct	Função da ULA	Controle da ULA
Iw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111



Controle da ULA (3)

- Tabela da verdade para os 4 bits de controle da ULA
 - As entradas são ALUOp e o campo de código de função

Don't care terms = X

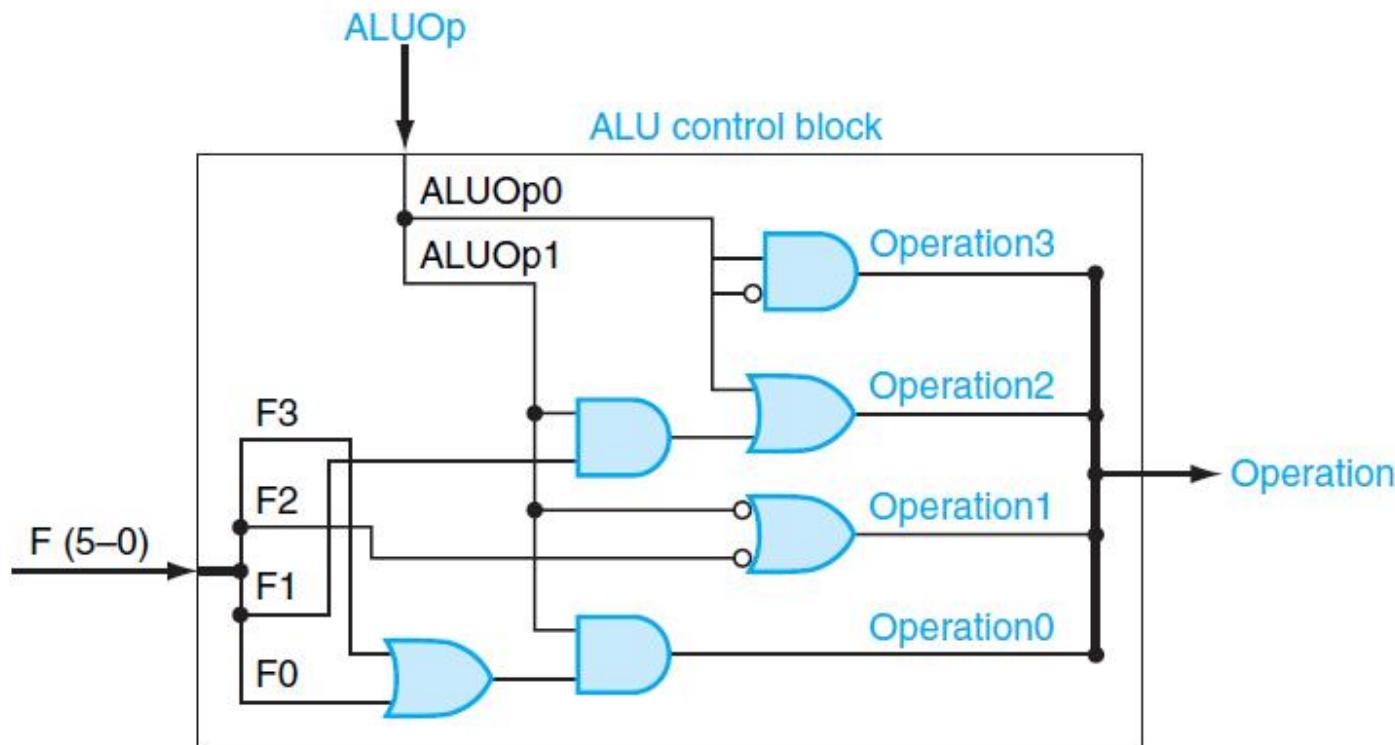
ALUOp		Funct field							Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0		
0	0	X	X	X	X	X	X	0010	
0	1	X	X	X	X	X	X	0110	
1	0	X	X	0	0	0	0	0010	
1	X	X	X	0	0	1	0	0110	
1	0	X	X	0	1	0	0	0000	
1	0	X	X	0	1	0	1	0001	
1	X	X	X	1	0	1	0	0111	

Os dois primeiros campos do campo de função são sempre 10



Controle da ULA (4)

- Gera os 4 bits de controle baseado nos bits da ALUOp e o código de função



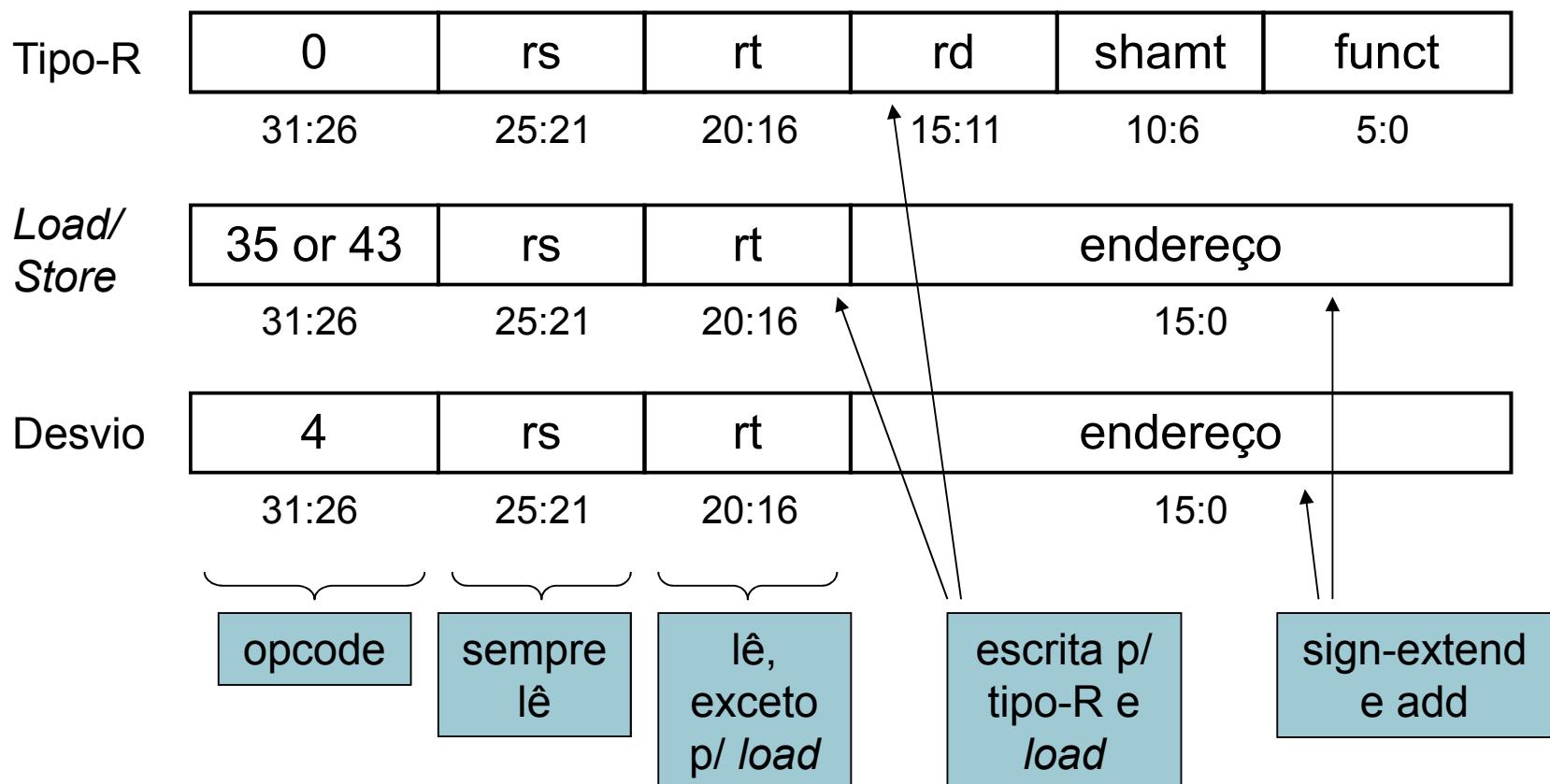
Exercício

- Projete um circuito lógico com três entradas A, B e C, cuja saída será nível alto apenas quando a maioria das entradas for nível alto

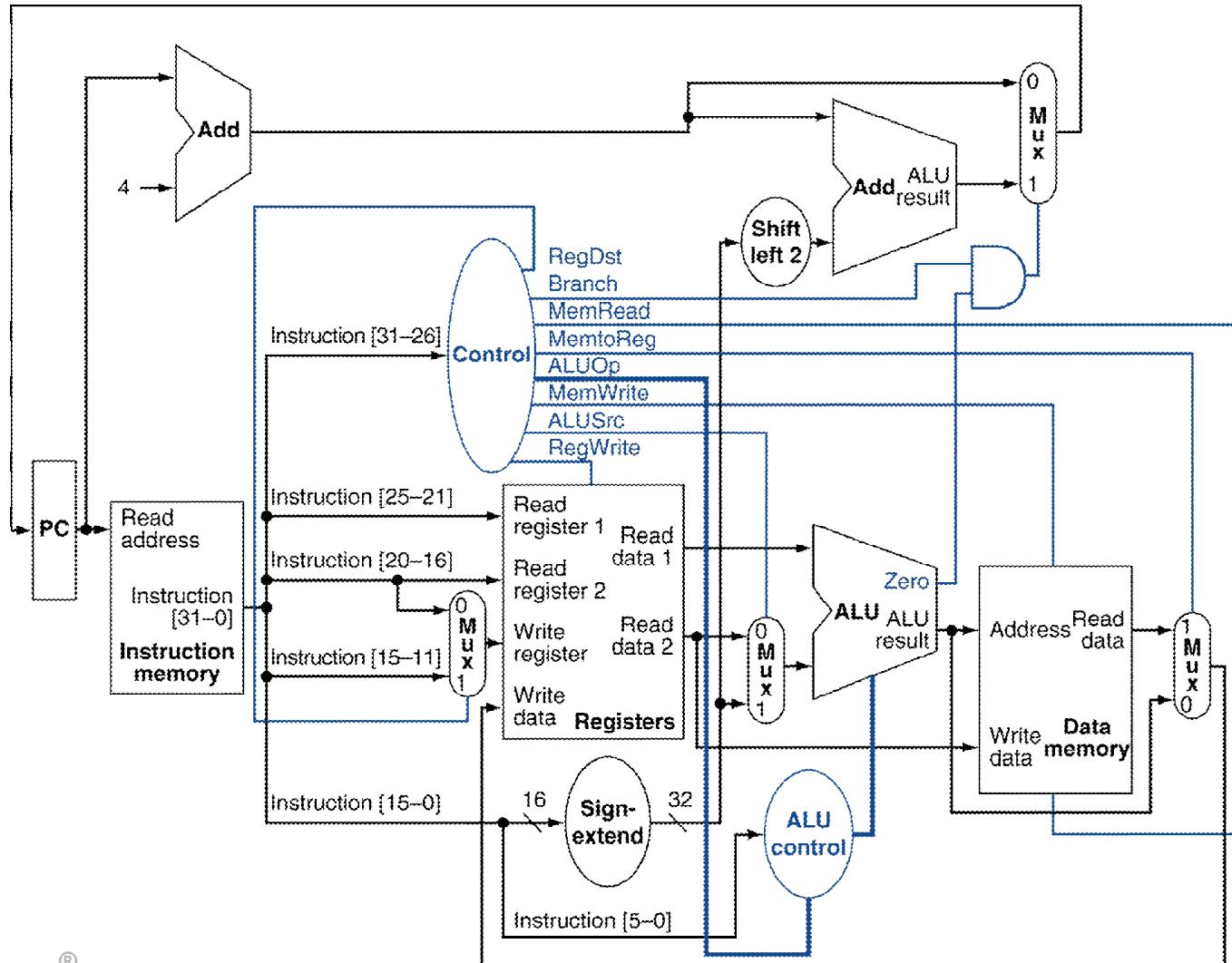


A Unidade de Controle Principal

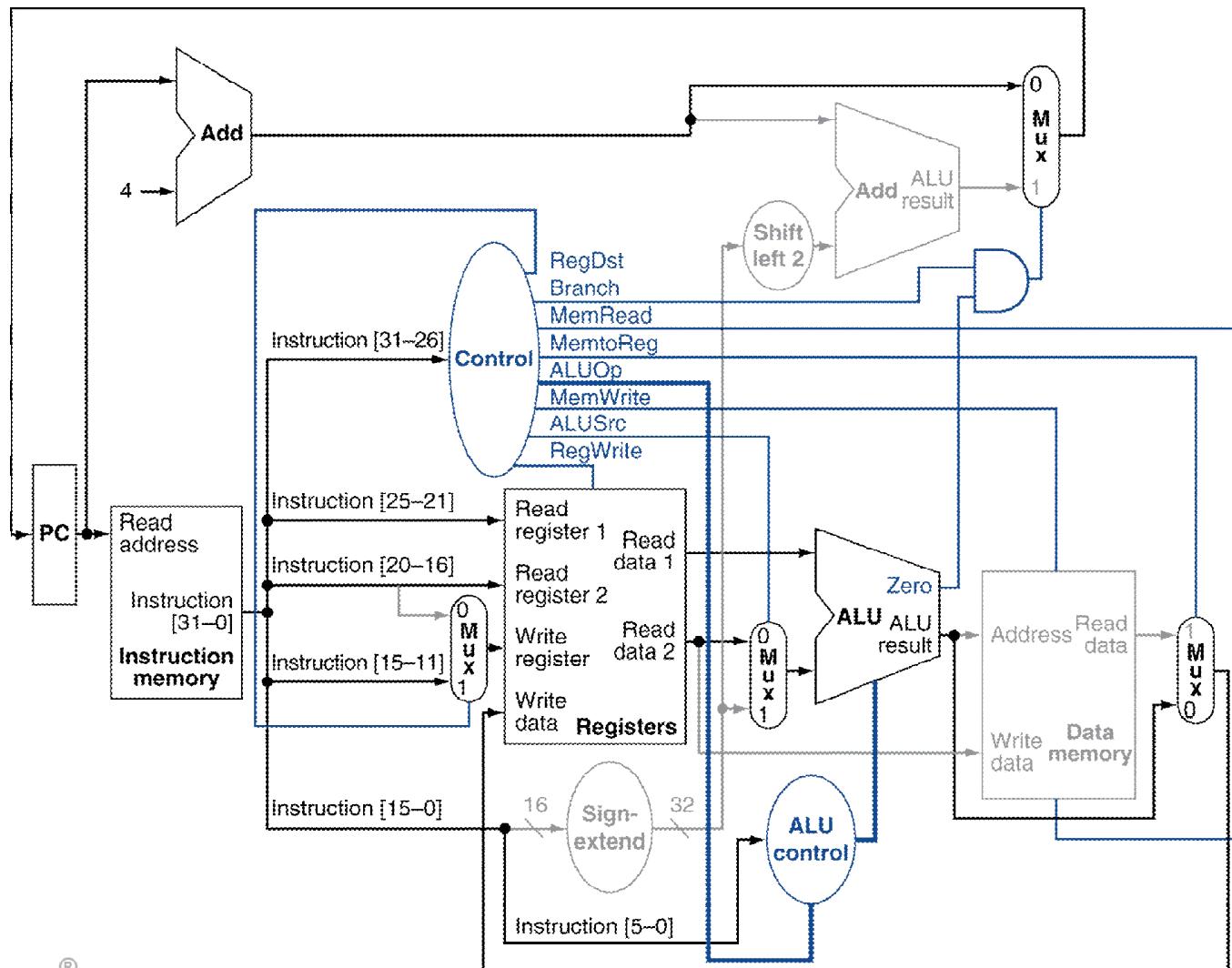
- Sinais de controle derivados a partir da instrução



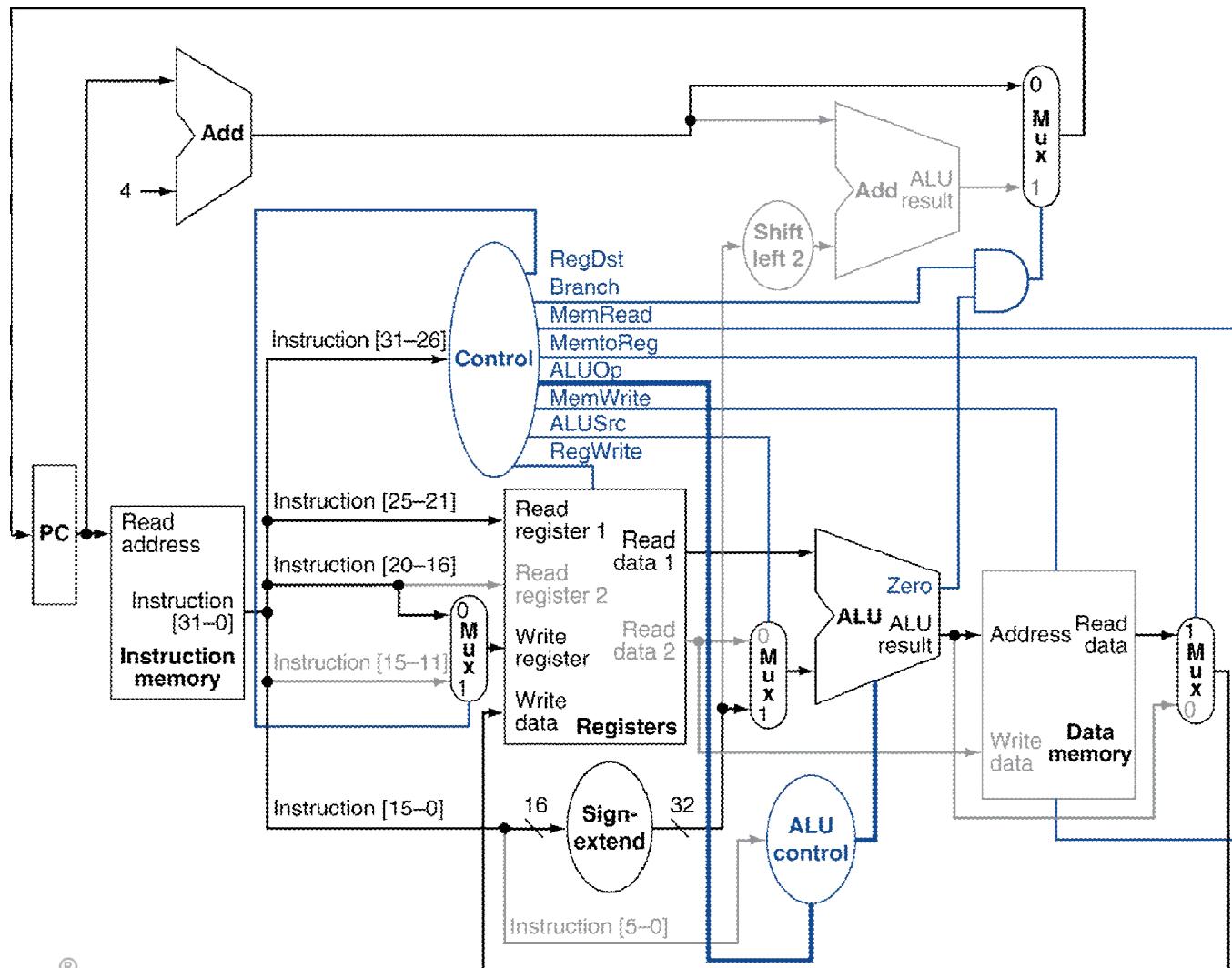
Caminho de Dados com Controle



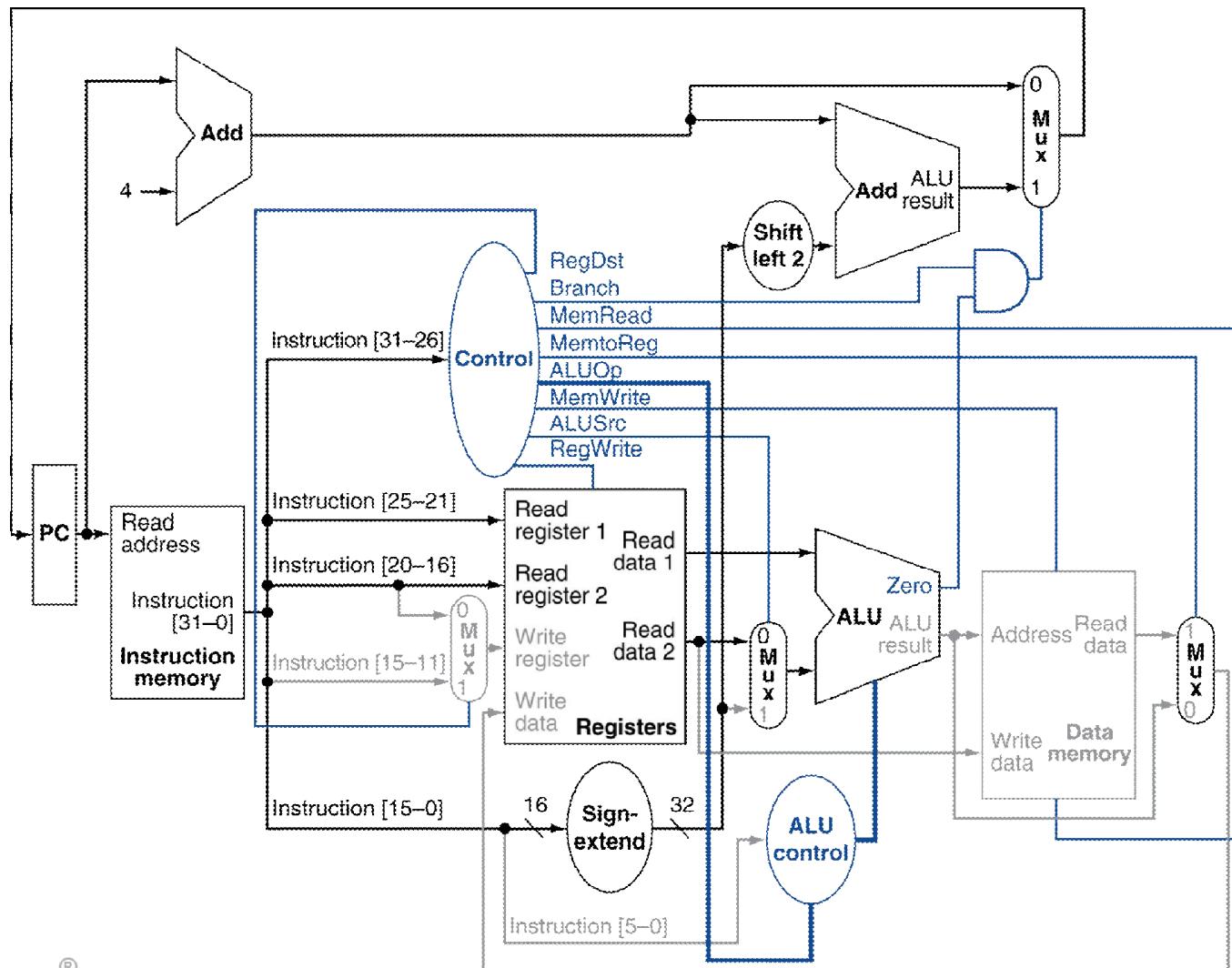
Instruções do Tipo-R



Instruções Load



Instruções *Branch-on-Equal*



Finalizando o Controle (1)

- A função de controle pode ser precisamente definida usando a seguinte tabela:

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

- Instruções do formato-R (add, sub, AND, OR, and slt):
 - Possuem regs fontes rs e rt e reg de destino rd
 - Define como ALUSrc e RegDst são setados
 - Usam RegWrite=1, mas nunca lêem nem escrevem na memória de dados



Finalizando o Controle (2)

- Quando o sinal de controle de *branch* é zero, o PC é substituído de forma incondicional por PC+4
 - Caso contrário, o PC é substituído pelo alvo de desvio se a saída Zero da ULA for também alto
- O campo ALUOp para instruções do tipo-R é setado para 10 para indicar que o controle da ULA deve ser gerado a partir do campo *funct*
- Instruções do formato *load/store* usam ALUSrc e ALUOp para executar o cálculo do endereço
 - *MemRead* e *MemWrite* são setados para executar acesso a memória
 - *RegDst* e *RegWrite* são setados para um *load* para fazer com que o resultado seja armazenado em *rt*



Finalizando o Controle (3)

- A instrução de desvio envia os regs *rs* e *rt* para a ULA (similar ao formato-R)
 - *ALUOp* é setado para uma subtração (Controle da ULA = 01), que é usado para teste de igualdade
 - *MemtoReg* é irrelevante quando o sinal *RegWrite* é zero (uma vez que o registrador não está sendo escrito)
 - *RegDst* é também irrelevante quando *RegWrite* é 0



Finalizando o Controle (4)

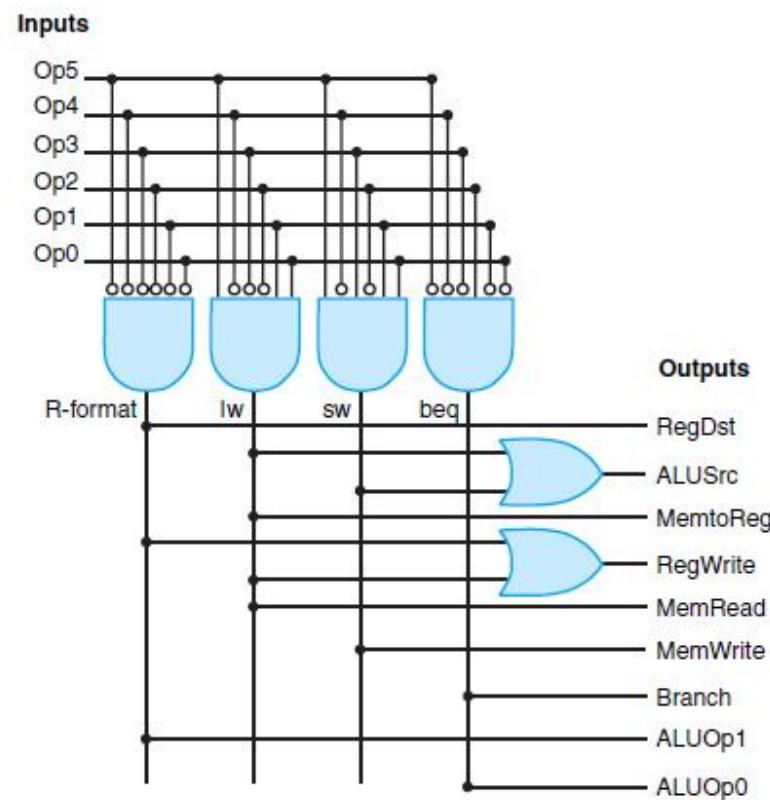
- A função de controle para uma implementação de um único ciclo

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1



Finalizando o Controle (5)

- Implementação estruturada da função de controle



Exercício

- A latência de componentes individuais do caminho de dados afetam o tempo de ciclo de relógio de todo caminho de dados

	I-Mem	Add	Mux	ULA	Regs	D-Mem	Sign-extend	Shift-left-2
a.	200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps
b.	750ps	200ps	50ps	250ps	300ps	500ps	100ps	5ps

- Qual é o tempo de ciclo de relógio se somente as instruções do tipo ULA (add, AND, etc.) são executadas?
- Qual é o tempo de ciclo de relógio se somente instruções LW são suportadas?
- Qual é o tempo de ciclo de relógio se somente instruções ADD, BEQ, LW e SW são suportadas?



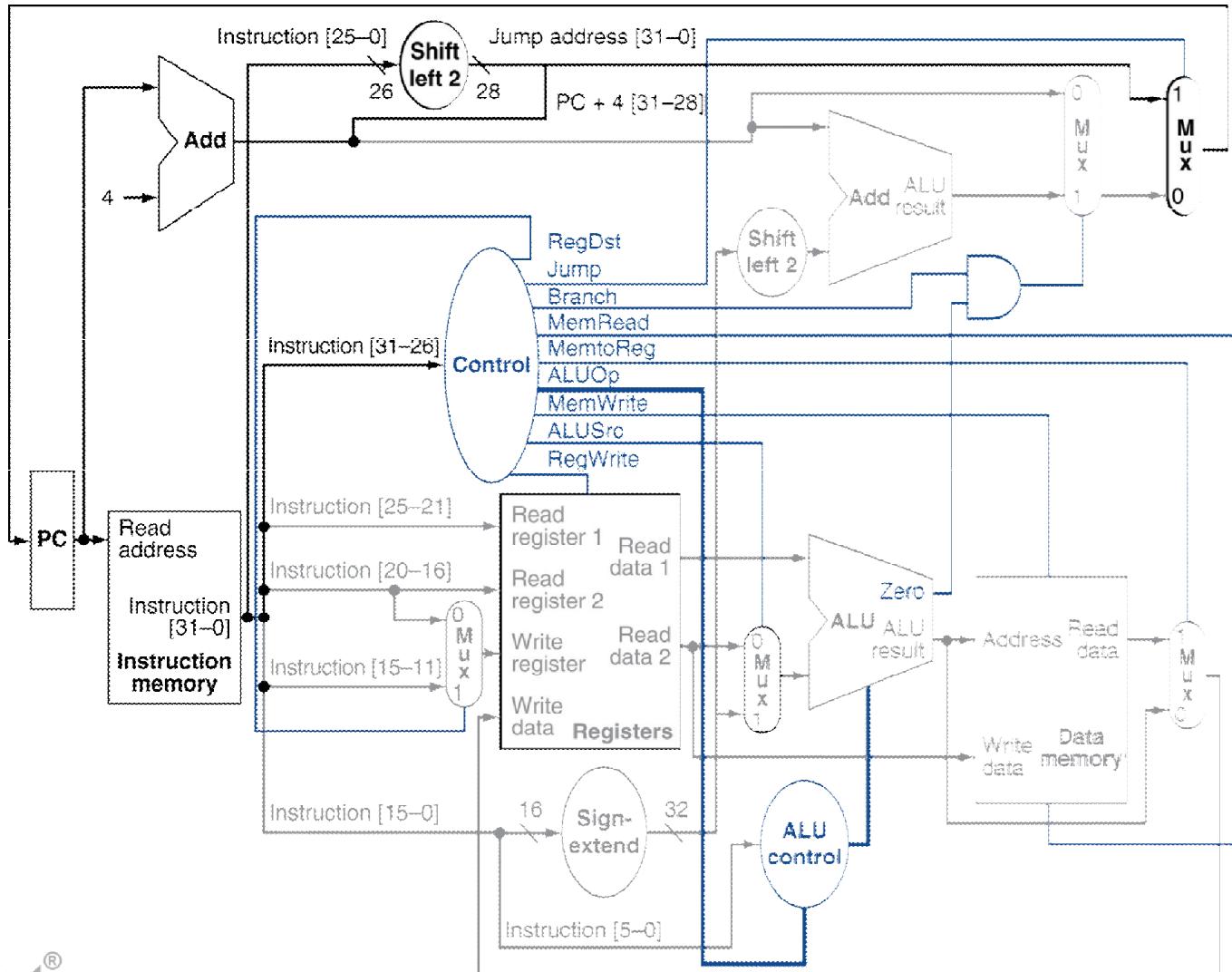
Implementando *Jumps*

Jump	000010	endereço
	31:26	25:0

- *Jump* usa endereço de palavra
- Atualiza PC com concatenação de
 - Os 4 bits superiores do PC atual + 4 (estes são os bits 31:28 da instrução de endereço seguinte)
 - Endereço de *jump* de 26 bits
 - Os bits 00_2
- Precisa de um sinal de controle extra (*Jump*) decodificado a partir do *opcode*



Caminho de Dados com Jumps Adicionados



Aspectos de Desempenho

- Atraso mais longo determina período de relógio
 - Caminho crítico: instrução de *load*
 - Memória de instrução → registrador → ULA → memória de dados → registrador
- Não é viável variar período para diferentes instruções
 - O ciclo de relógio deve ter a mesma duração para cada instrução neste projeto de ciclo único
- Viola princípio de projeto
 - Fazendo o caso comum rápido
- Melhorar o desempenho através do *pipeline*

