O Processador: Via de Dados e Controle

©1998 Morgan Kaufmann Publishers

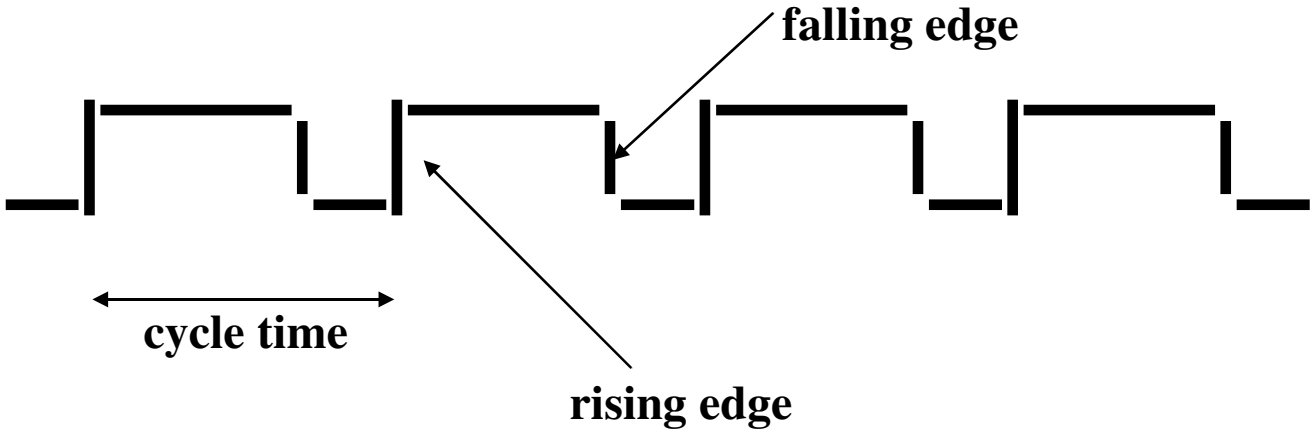
Via de Dados e Controle

- **Implementação da arquitetura MIPS**
- **Visão simplificada de uma arquitetura monociclo**
  - **Instruções de memória:** `lw, sw`
  - **Instruções lógicas-aritméticas:** `add, sub, and, or, slt`
  - **Instruções de desvio:** `beq, j`
- **Não implementadas: mult, div, jal, fp**
- **Implementação genérica:**
  - **O contador de programa (PC) fornece o endereço da instrução**
  - **Obtém instrução da memória**
  - **Ler registradores**
  - **Usar o opcode da instrução para decidir exatamente o que deve ser feito**
- **Todas as instruções utilizam a ULA após a leitura dos registradores**
  -

©1998 Morgan Kaufmann Publishers

Elementos de estado

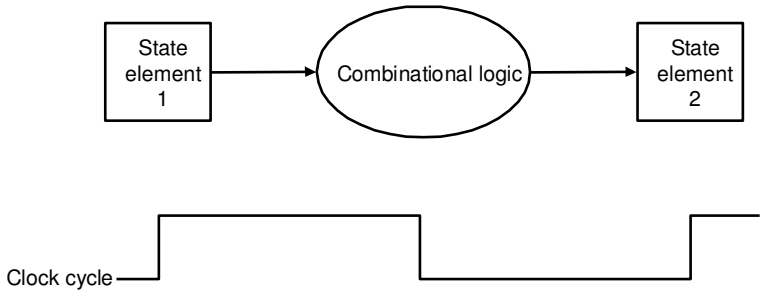
- **Clock é usado em lógica síncrona**
  - **Quando um elemento de estado deve ser atualizado?**



©1998 Morgan Kaufmann Publishers

Implementação da lógica baseada em clock

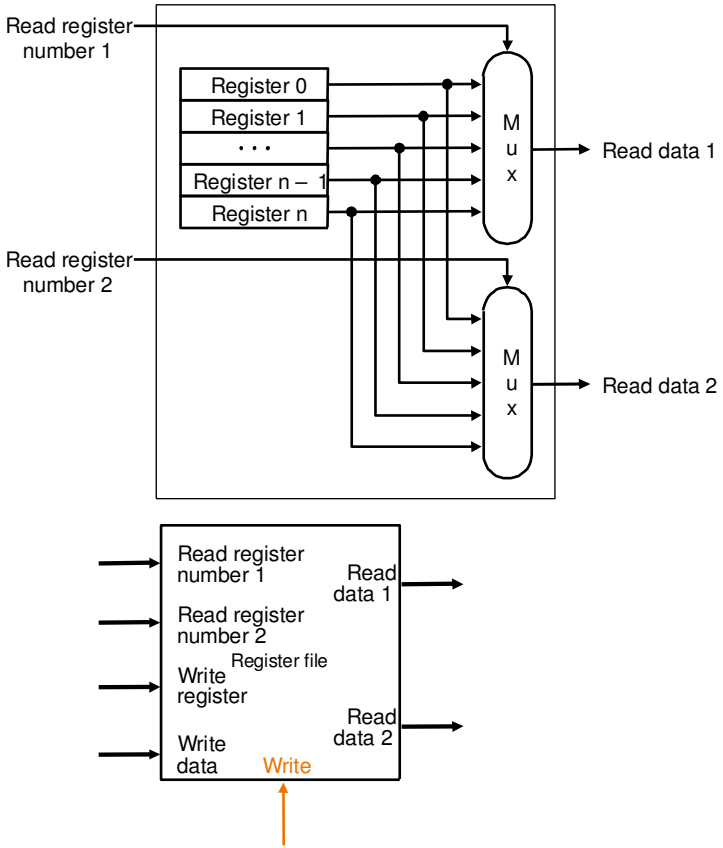
- **Metodologia de disparo na borda**
- **Execução típica:**
  - **Ler conteúdo de elementos de estado**
  - **Enviar valores através de alguma lógica combinacional**
  - **Escrever resultados para um ou mais elementos de estado**



©1998 Morgan Kaufmann Publishers

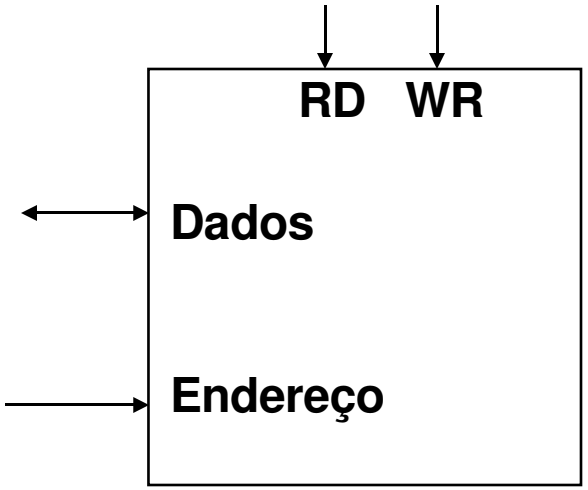
Banco de registradores (e Memória)

Banco de registradores



Memória:

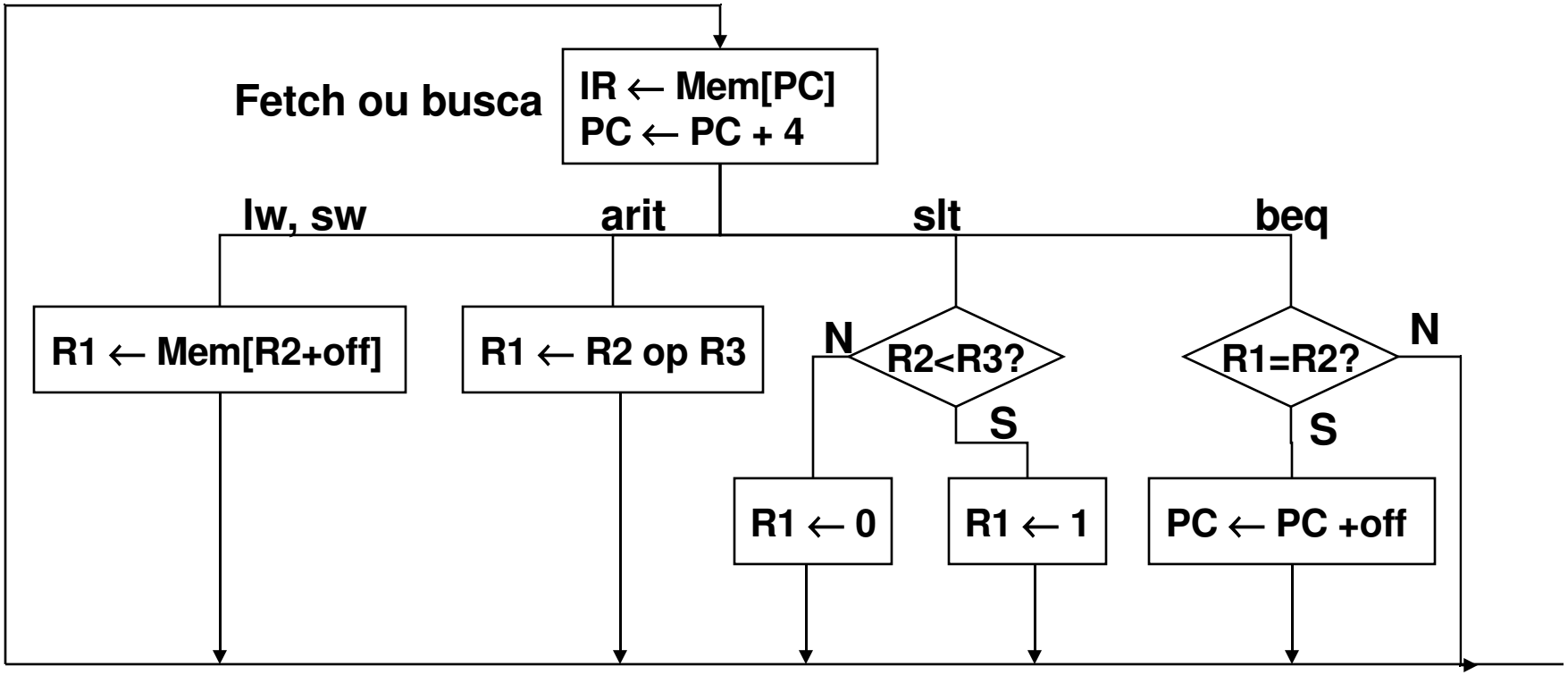
- acesso Mem[addr]



©1998 Morgan Kaufmann Publishers

Uma visão simplificada

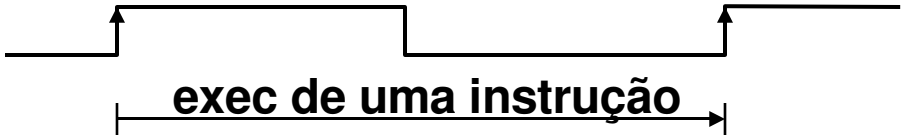
Dados: lw, sw	`lw $t1, 100($t2)`
Arit: add, sub, .. op	`add $t1, $t2, $t3`
Log: slt	`slt $t1, $t2, $t3`
Desvio: beq	`beq $t1, $t2, label`



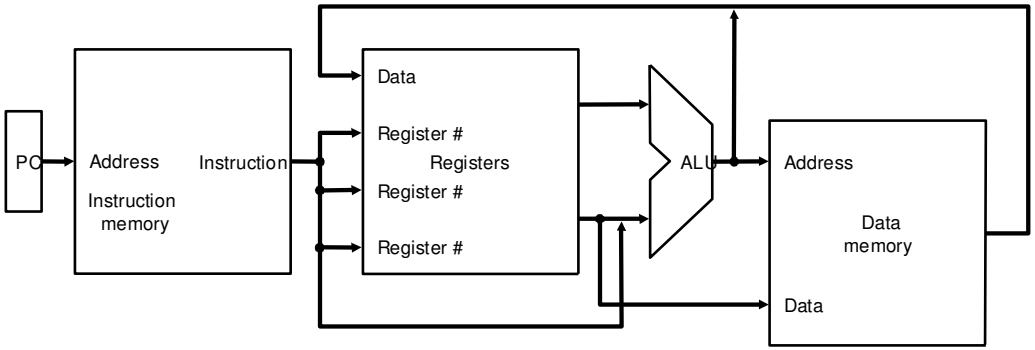
©1998 Morgan Kaufmann Publishers

Mais detalhes de implementação

- **Primeira abordagem: 1 período de clock por instrução**



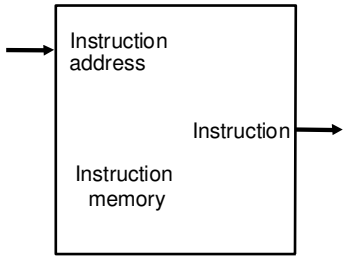
- **Visão simplificada**



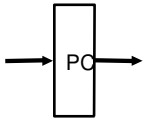
©1998 Morgan Kaufmann Publishers

Simple Implementation

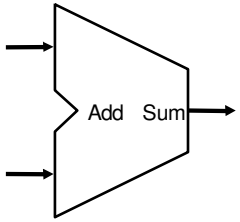
- **Include the functional units we need for each instruction**



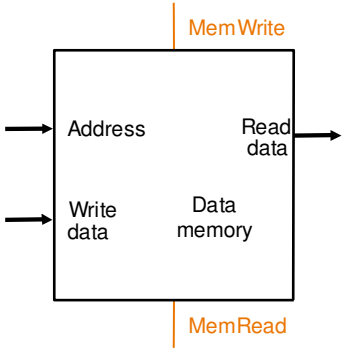
a. Instruction memory



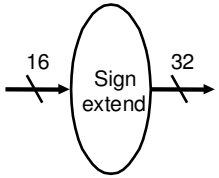
b. Program counter



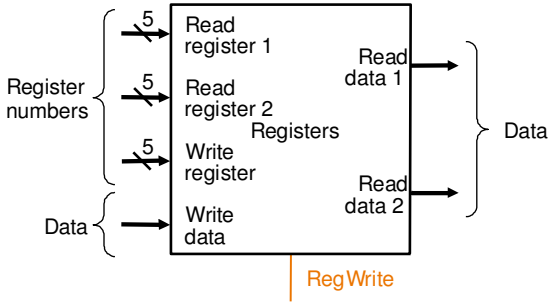
c. Adder



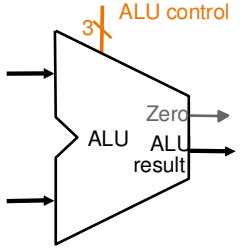
a. Data memory unit



b. Sign-extension unit



a. Registers

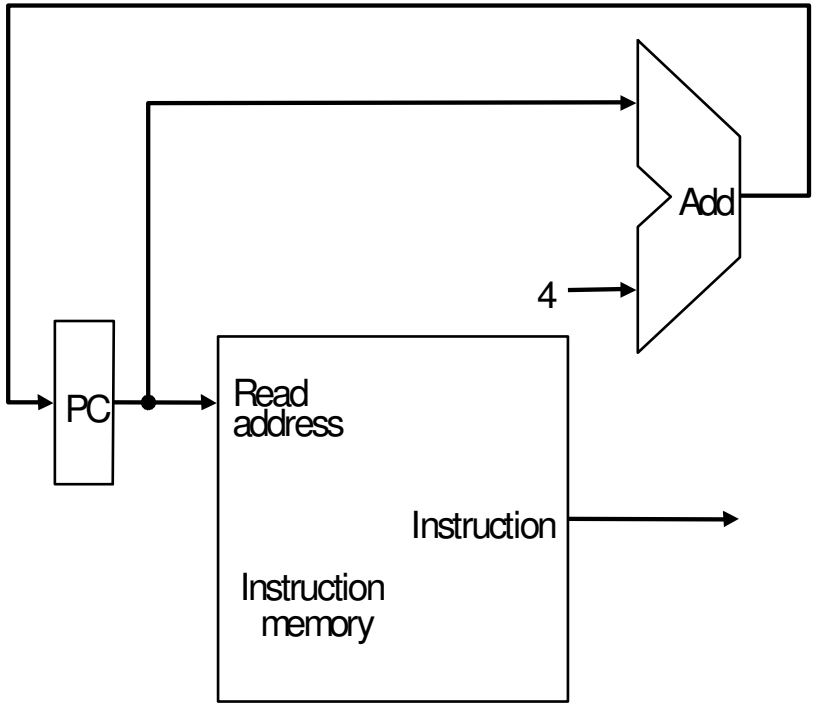


b. ALU

Why do we need this stuff?

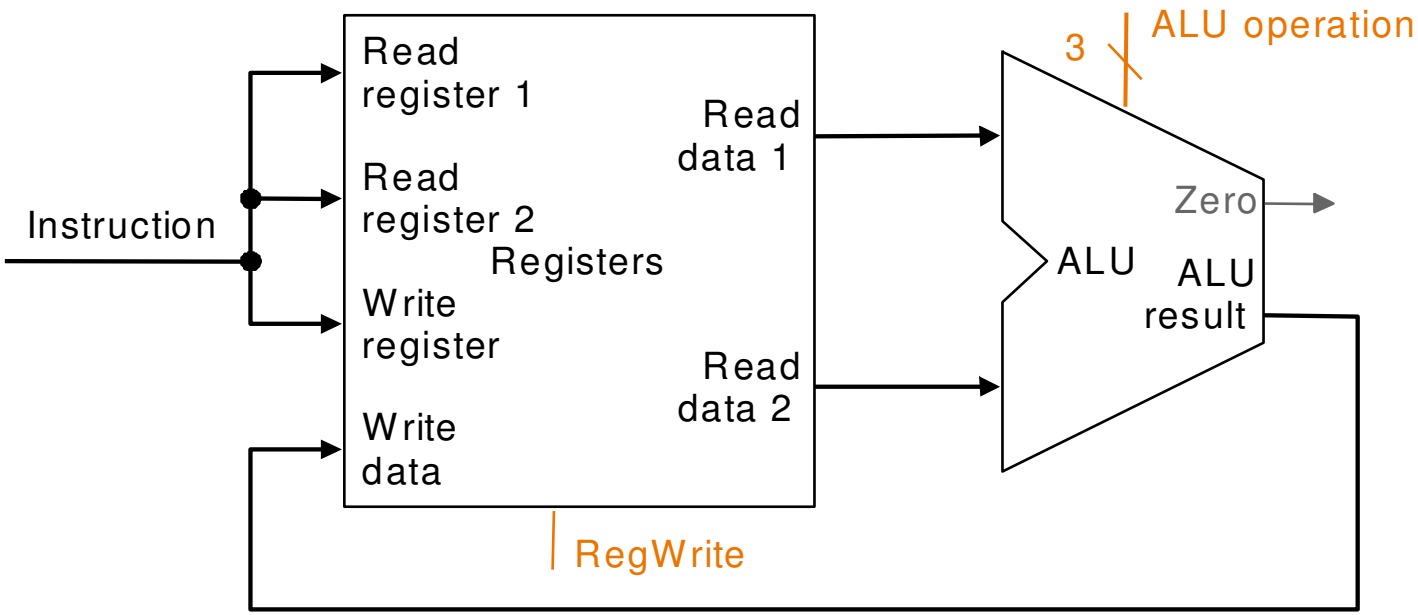
©1998 Morgan Kaufmann Publishers

Busca (Fetch)



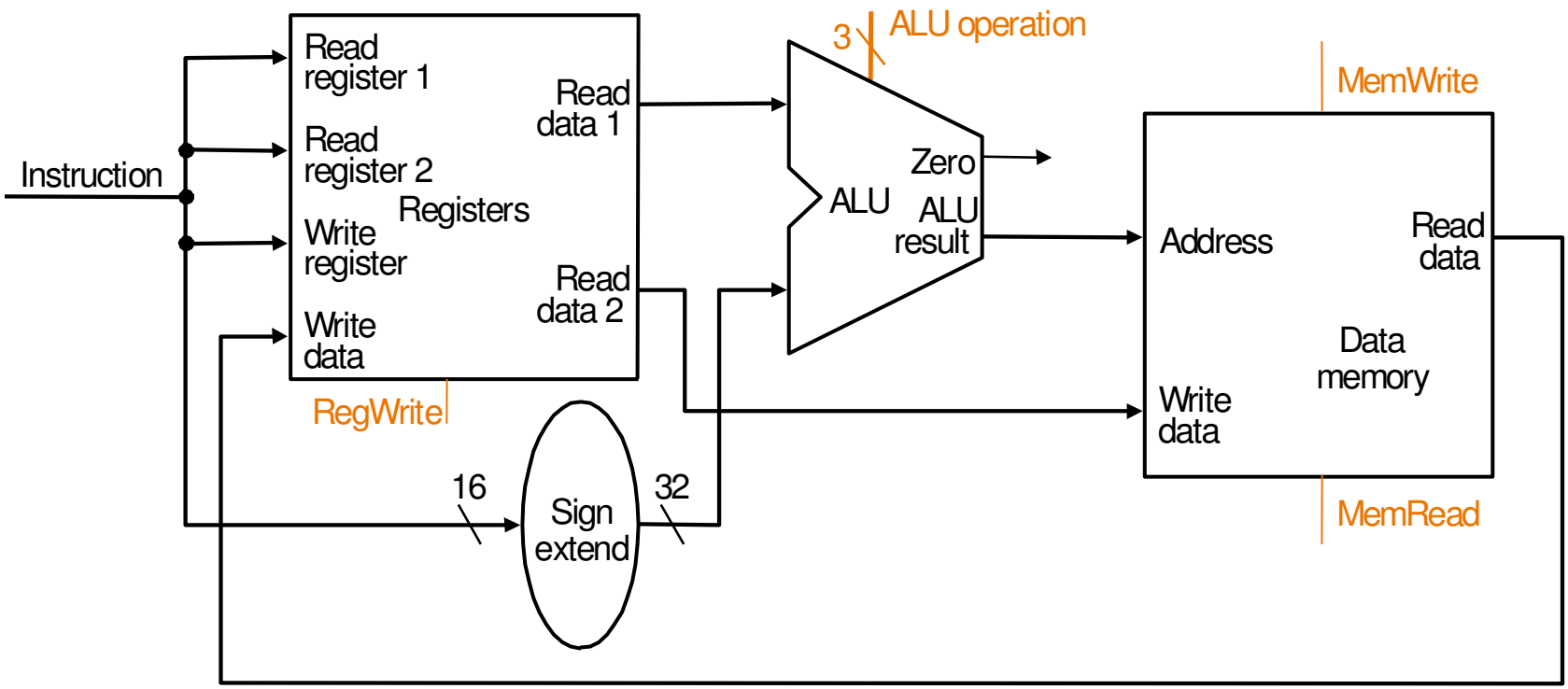
©1998 Morgan Kaufmann Publishers

Tipo R



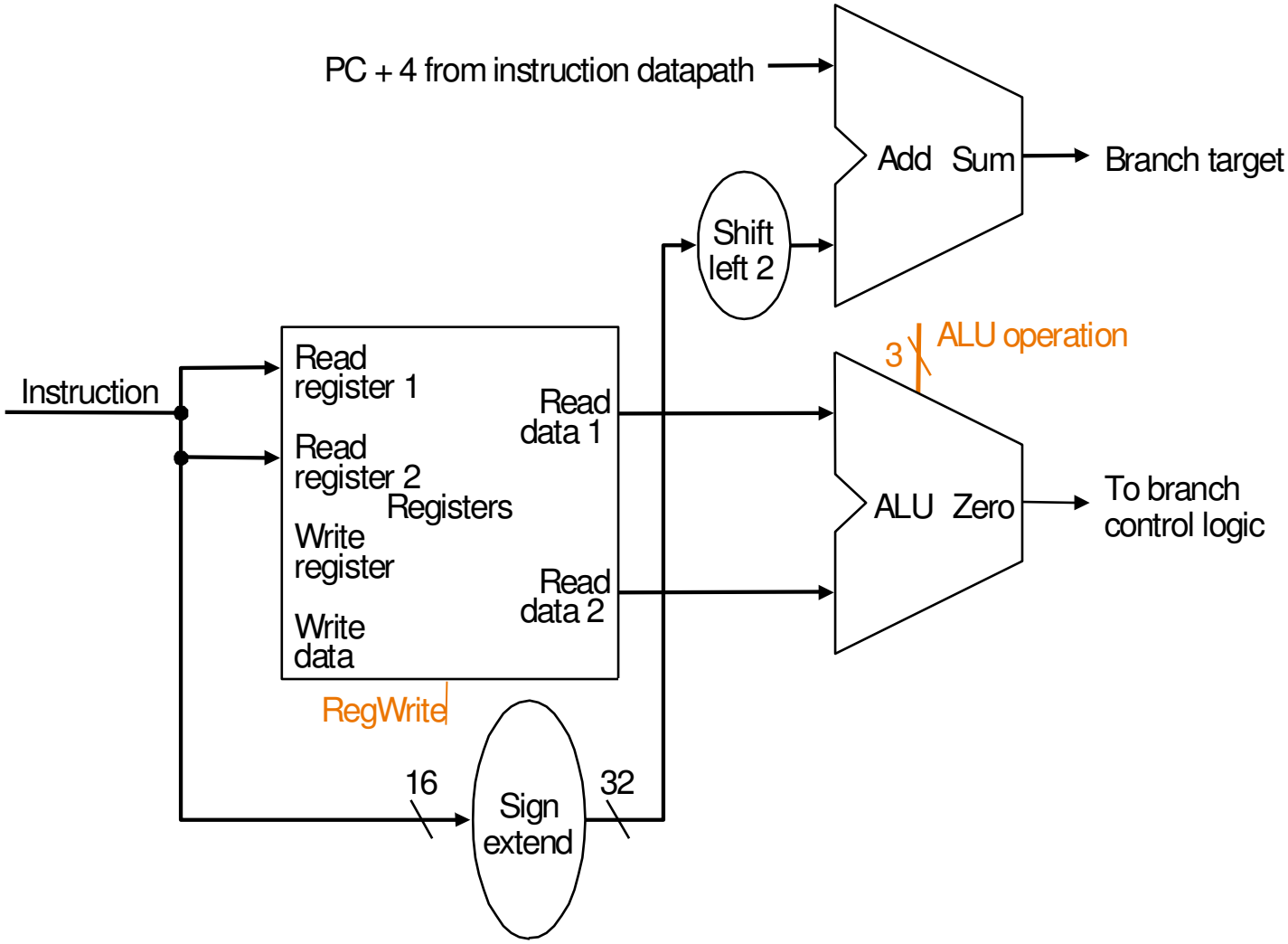
©1998 Morgan Kaufmann Publishers

Referência a Memória



©1998 Morgan Kaufmann Publishers

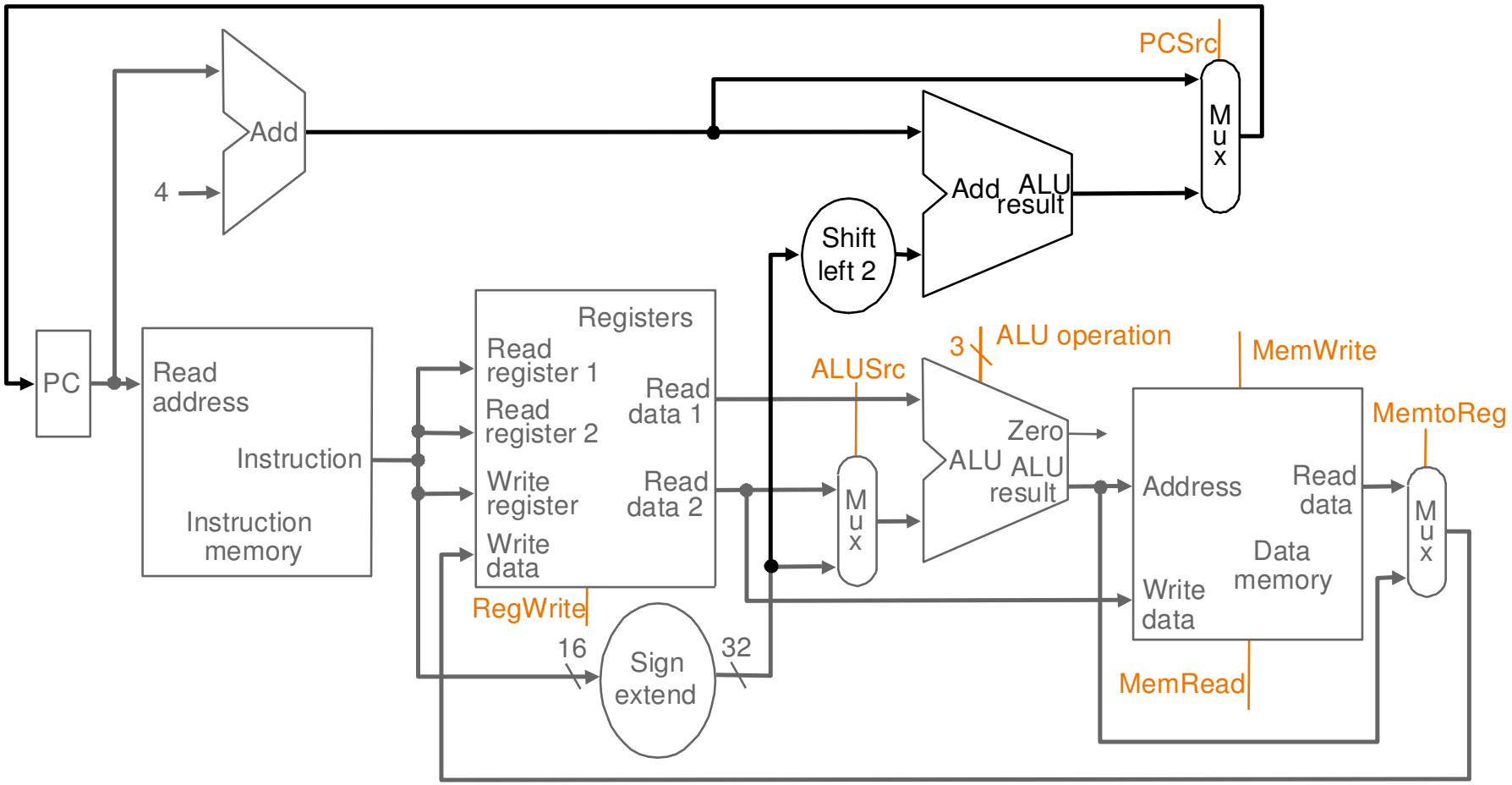
beq



©1998 Morgan Kaufmann Publishers

Construção da via de dados

- **Uso de multiplexadores para seleção do valor dos dados**



©1998 Morgan Kaufmann Publishers

ALUs e Multiplexadores

- **ALU genérica: R1 ← R2 op R3**
- **ALU soma para endereço instrução: PC + 4**
- **ALU soma para endereço de desvio: PC + SignExt (ShiftLeft (Offset))**

- **MUX 2:1 para escolher segundo operando da ALU genérica**
  - **R2**
  - **SignExt (Offset)**

- **MUX 2:1 para escolher o que será carregado no PC**
  - **PC + 4**
  - **PC + SignExt (ShiftLeft (Offset))**

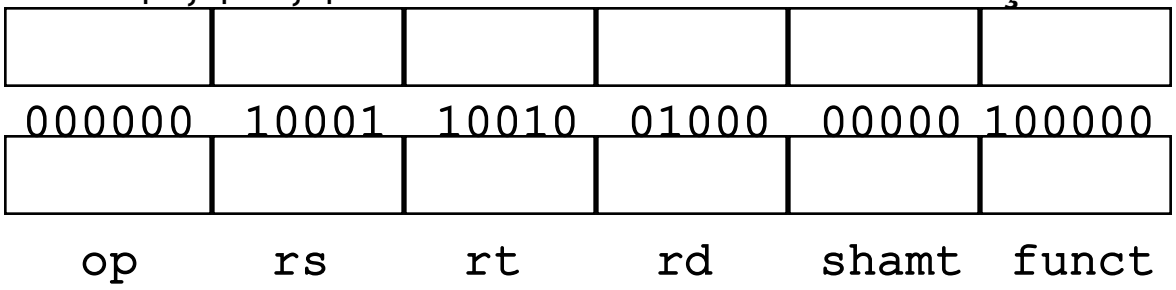
- **MUX 2:1 para escolher qual é a origem do dado a ser escrito no Banco de Registradores**
  - **lido da memória**
  - **resultado da operação da ALU genérica**

©1998 Morgan Kaufmann Publishers

Controle

- **Selecionar operações para uso da ALU, leitura/escrita dos elementos, etc.**
- **Controlar o fluxo de dados (entrada dos multiplexadores)**
- **Todas as informações vêm dos 32 bits da instrução**
- **Exemplo:**

add $8, $17, $18 **Formato da instrução**

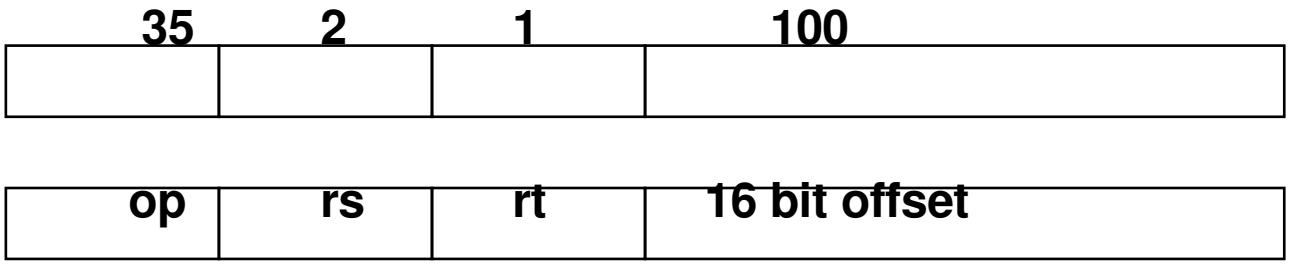


- **Operação da ULA é baseada nos campos opcode e função**

©1998 Morgan Kaufmann Publishers

Controle

- **Exemplo: lw $1, 100($2)**



- **Entrada de controle da ULA**

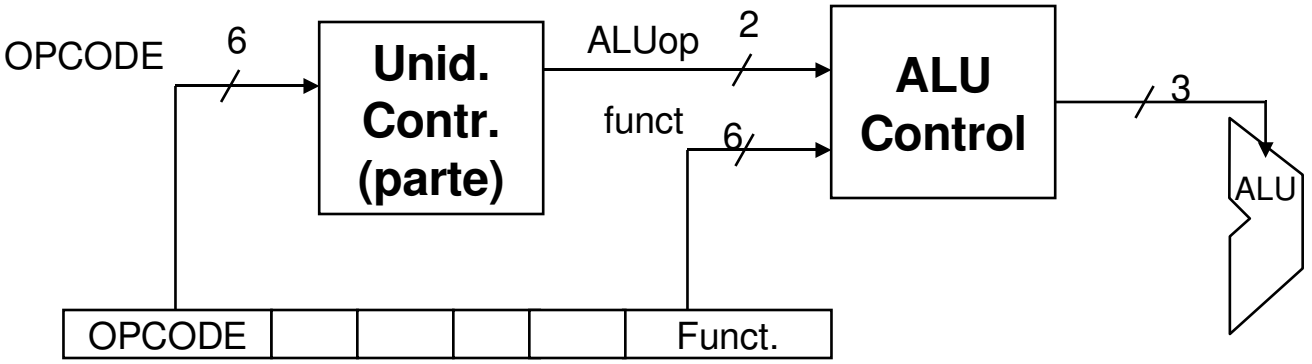
```
000    AND
001    OR
010    add
110    subtract
111    set-on-less-than
```

-

©1998 Morgan Kaufmann Publishers

Controle da ALU

OPCODE	Function	ALUop	Operação
lw ou sw	xxx xxx	00	soma
000 000	tipo da instr.	10	depende da instr.
beq	xxx xxx	01	subtração



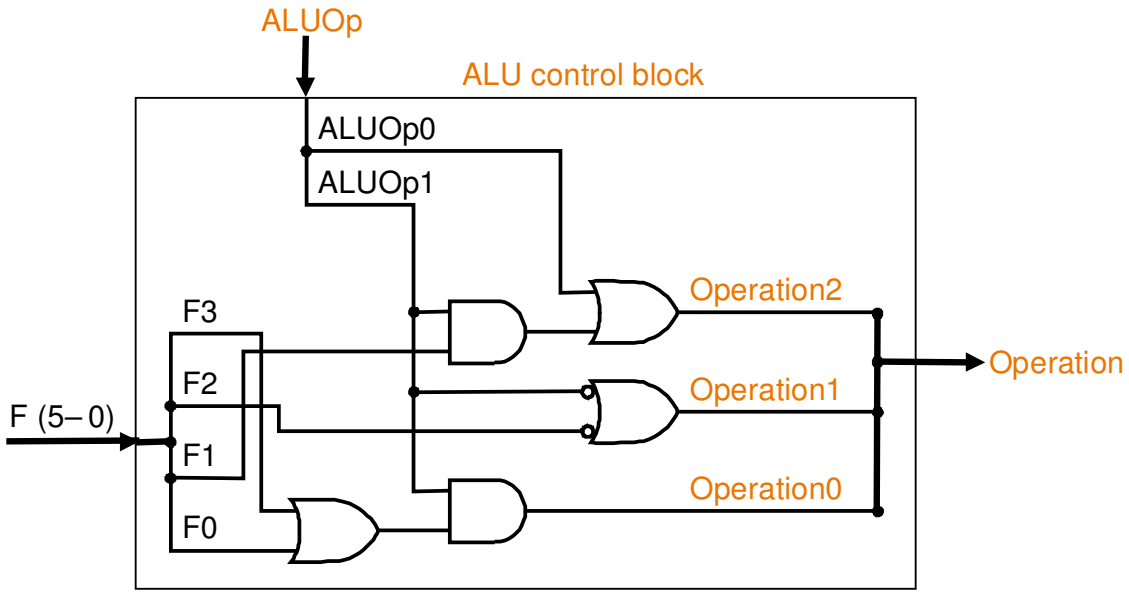
ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

Binv - cod

Inst	Funct.
add	32=20H
sub	34=22H
and	36=24H
or	37=25H
slt	42=2AH

©1998 Morgan Kaufmann Publishers

Lógica de controle da ALU

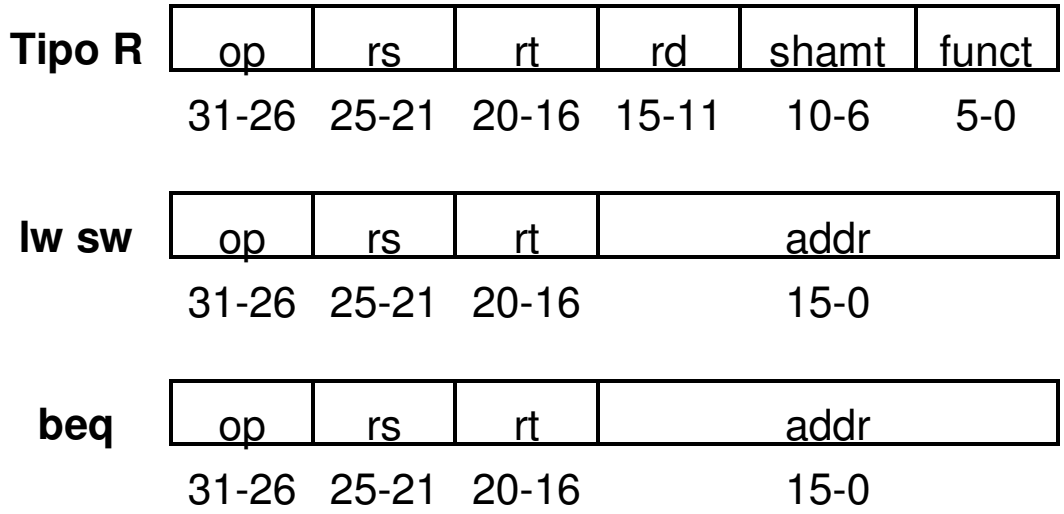


ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

Inst	Funct.
add	32=20H
sub	34=22H
and	36=24H
or	37=25H
slt	42=2AH

©1998 Morgan Kaufmann Publishers

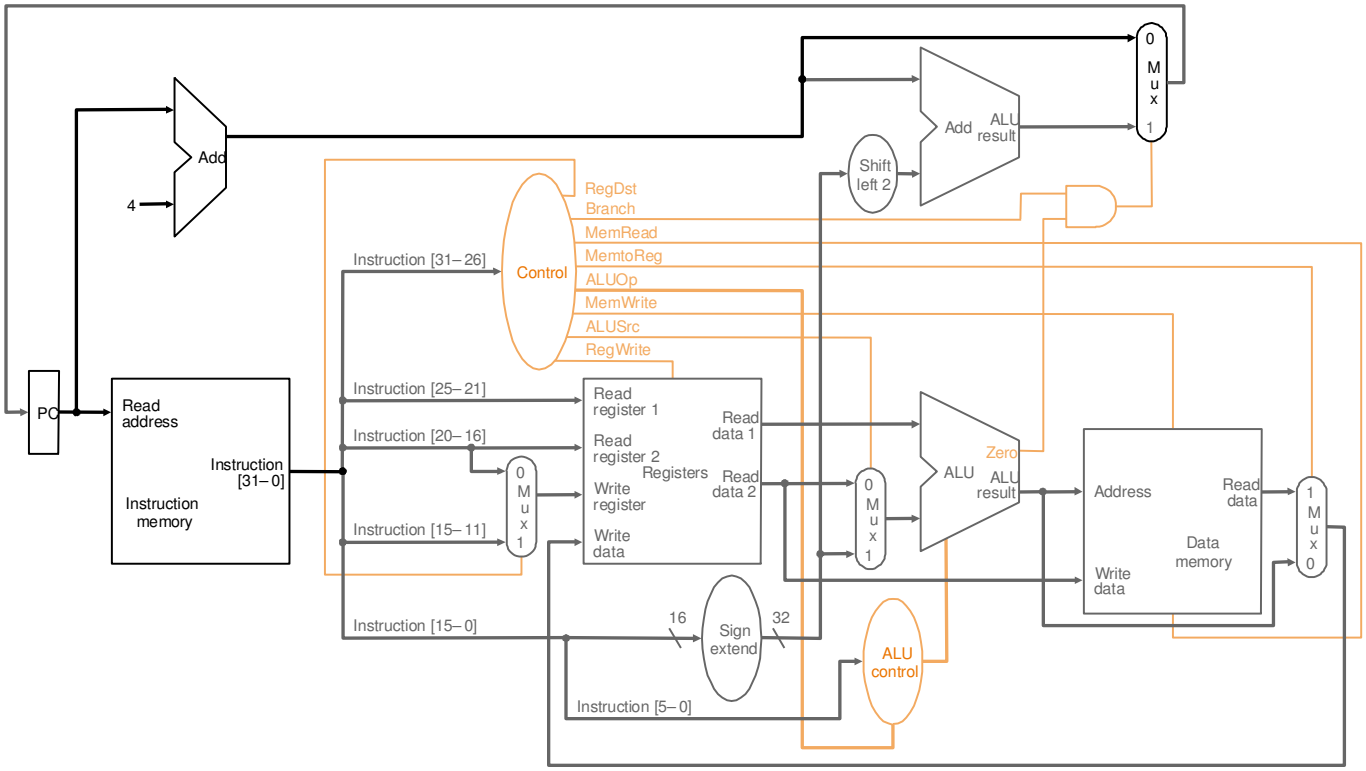
Campos da instrução e MUX adicional



- **2 registradores a serem lidos em todos os tipos:**
  - **campos rs (25-21) e rt (20-16)**
- **offset para beq, lw e sw:**
  - **(15-0)**
- **Registradores de destino (write register) em dois lugares:**
  - **lw e sw: rt (20-16)**
  - **Tipo R: rd (15-11)**
  - **Necessário MUX controlado por RegisterDestination: RegDst**

©1998 Morgan Kaufmann Publishers

Controle



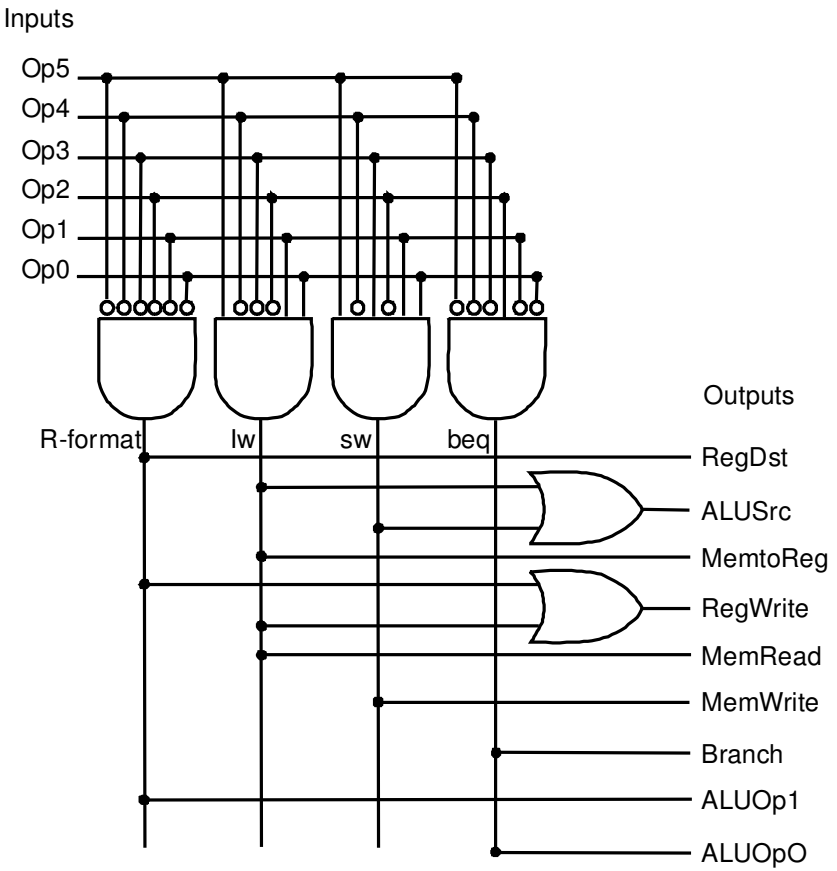
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1

©1998 Morgan Kaufmann Publishers

Unidade de Controle

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
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

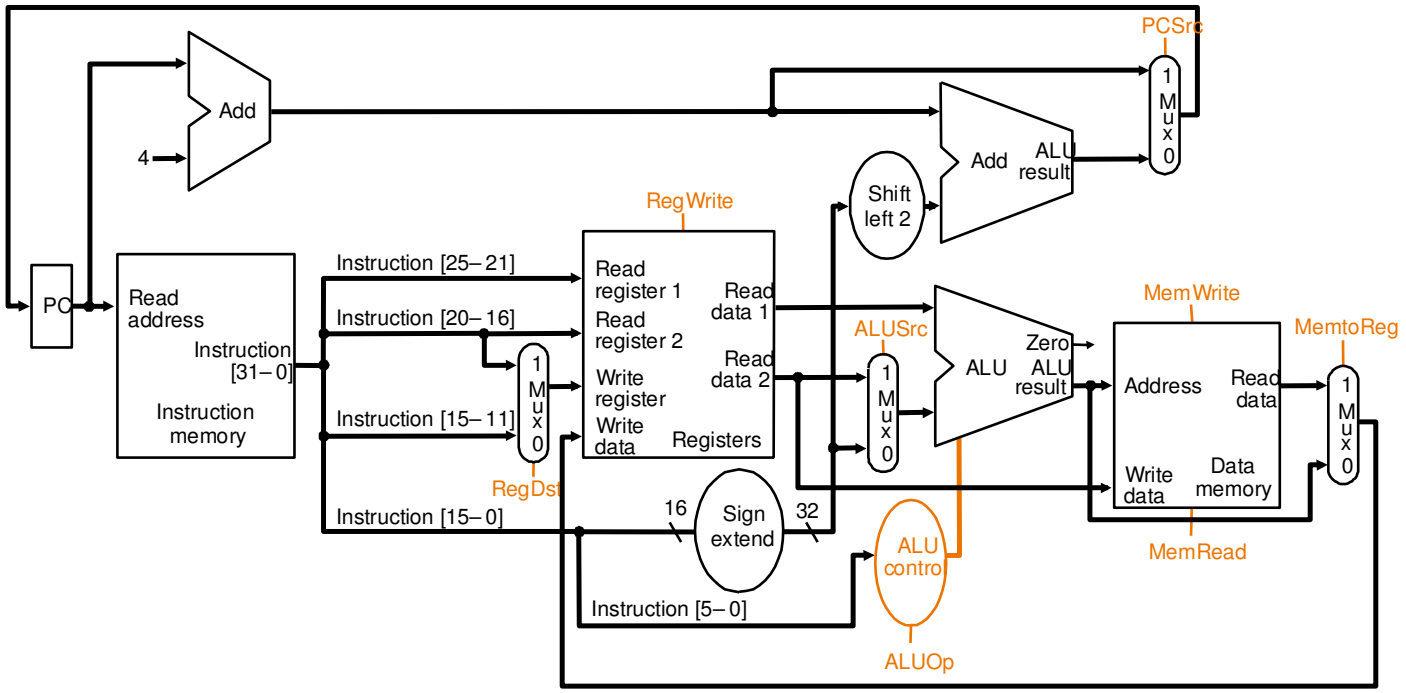
Formato R: 000 000
lw: 100 011
sw: 101 011
beq: 000 100



©1998 Morgan Kaufmann Publishers

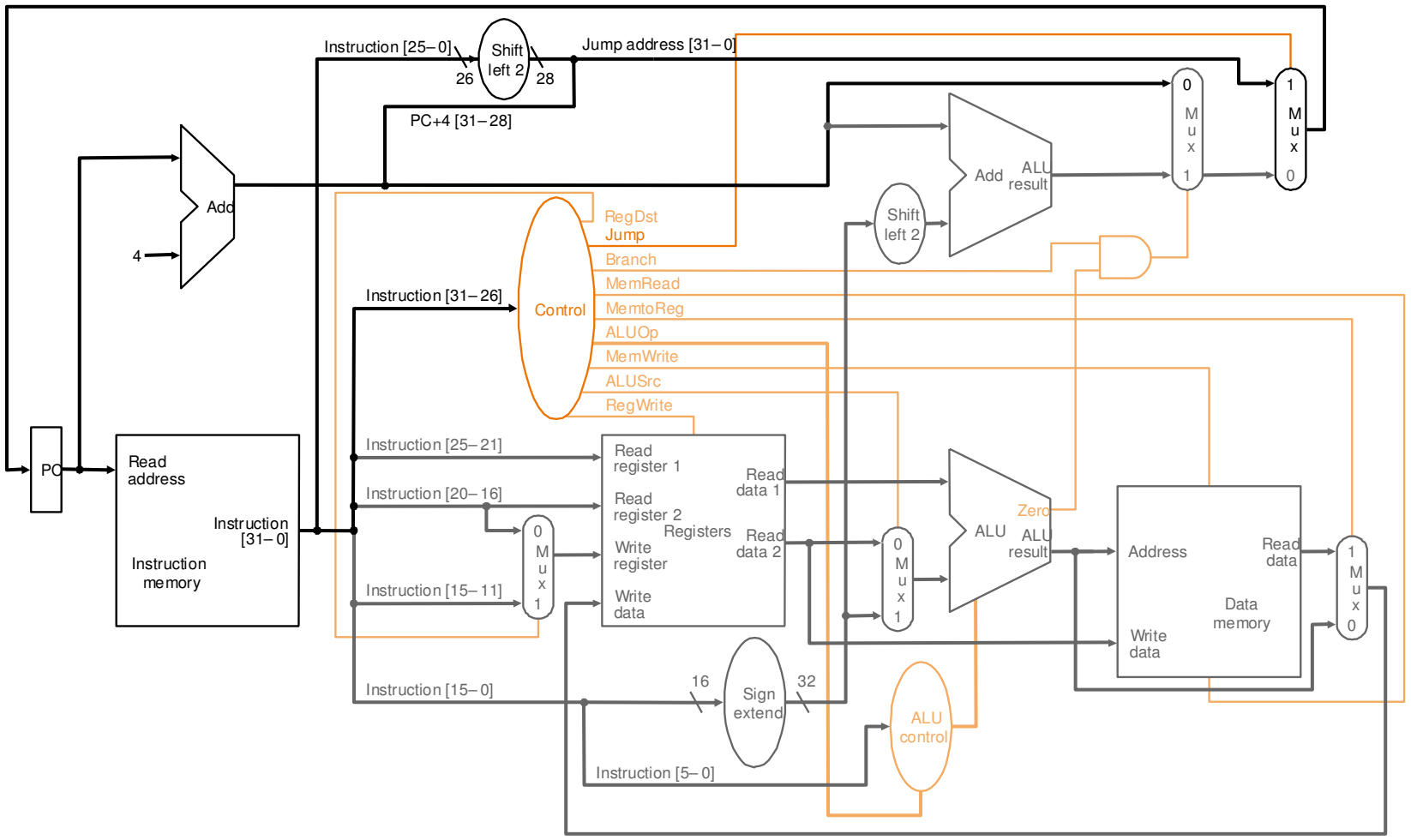
Implementação monocicloc

- **Considerar os seguintes atrasos**
  - **memory (2ns), ALU and adders (2ns), register file access (1ns)**



©1998 Morgan Kaufmann Publishers

Adicionando a instrução jump



2	address
31-26	25-0

©1998 Morgan Kaufmann Publishers

Problemas com a alternativa de ciclo único

- **"Vantagem": CPI = ?**
- **Desempenho?**
  - **supor unidades em uso: Memória (2ns), ALU (2ns), Registr (1ns)**

Classe					
R	Fetch	Registr	ALU	Registr	
lw	Fetch	Registr	ALU	Mem	Registr
sw	Fetch	Registr	ALU	Mem	
beq	Fetch	Registr	ALU		
j	Fetch				

Classe	Mem. Instr.	Reg RD	ALU	Data Mem	Reg WR	total
R	2	1	2	0	1	6
lw	2	1	2	2	1	8
sw	2	1	2	2		7
beq	2	1	2			5
j	2					2

©1998 Morgan Kaufmann Publishers

Problemas com o ciclo único (2)

- **Desempenho:**
  - **TCK = caminho crítico = 8 ns**
  - **velocidade limitada pelo pior caso**
- **Alternativas:**
  - **fazer clock com freqüência variável (muito custoso e complicado)**
  - **"picar" a instrução em pequenas funções e executá-las em vários ciclos de clock, uma (ou mais) por ciclo**
    - em vez de duração variável de ciclo, número variável
- **Vantagens do multiciclo:**
  - **velocidade: ciclo limitado pela "operação" mais lenta e não pela "instrução" mais lenta**
  - **instruções mais simples executam mais rapidamente**
  - **economia de hardware**
    - ter apenas 1: ALU, memória, registrador
    - usar uma vez todos esses, por ciclo

©1998 Morgan Kaufmann Publishers