Chapter Five

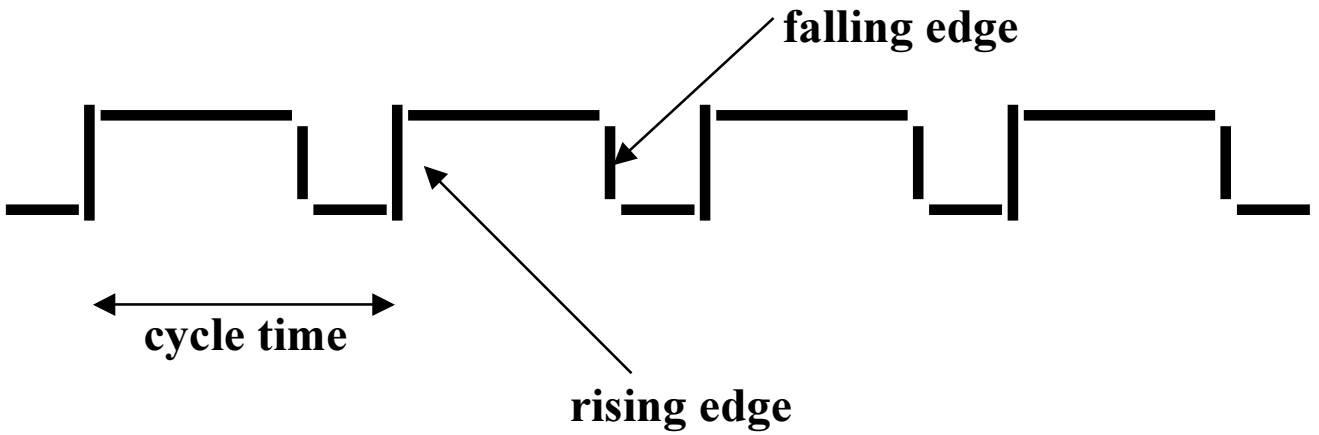
The Processor: Datapath and Control

The Processor: Datapath & Control

- We're ready to look at an implementation of the MIPS
- Simplified to contain only:
  - memory-reference instructions: `lw, sw`
  - arithmetic-logical instructions: `add, sub, and, or, slt`
  - control flow instructions: `beq, j`
- Não implementadas: mult, div, jal, fp
- Generic Implementation:
  - use the program counter (PC) to supply instruction address
  - get the instruction from memory
  - read registers
  - use the instruction to decide exactly what to do
- All instructions use the ALU after reading the registers:
  - memory-reference (sw lw), arithmetic, control flow

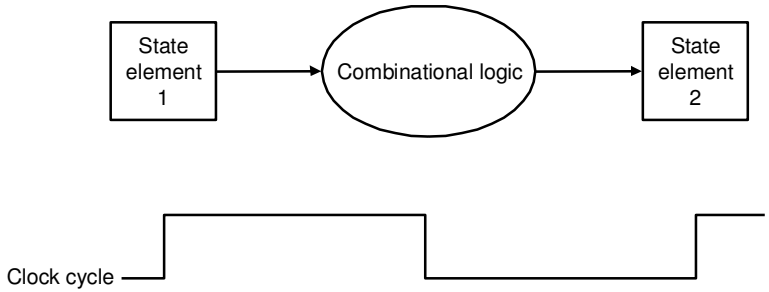
State Elements

- Unclocked vs. Clocked
- Clocks used in synchronous logic
  - when should an element that contains state be updated?



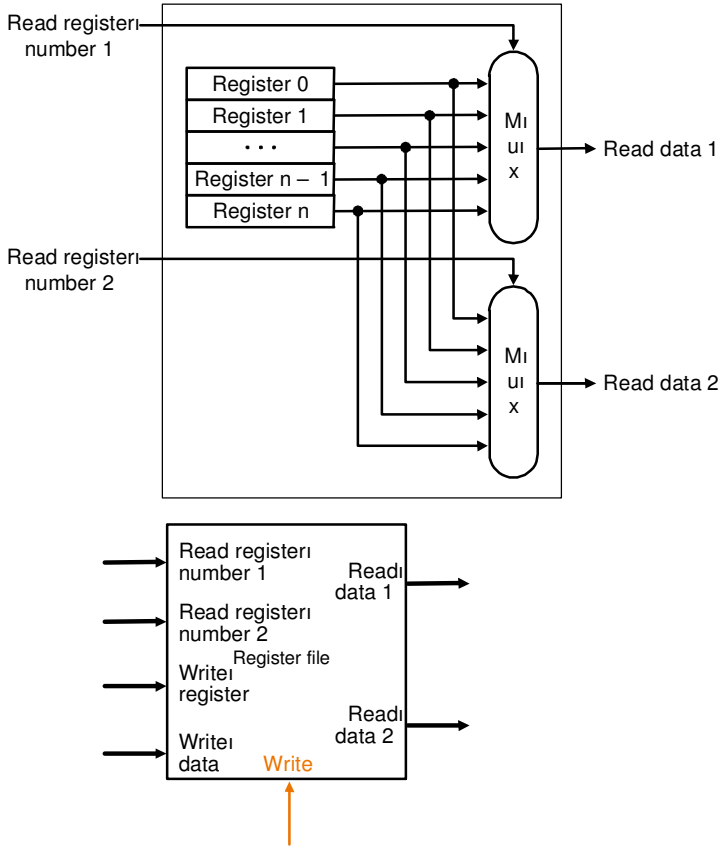
Our Implementation

- **An edge triggered methodology**
- **Typical execution:**
  - **read contents of some state elements,**
  - **send values through some combinational logic**
  - **write results to one or more state elements**

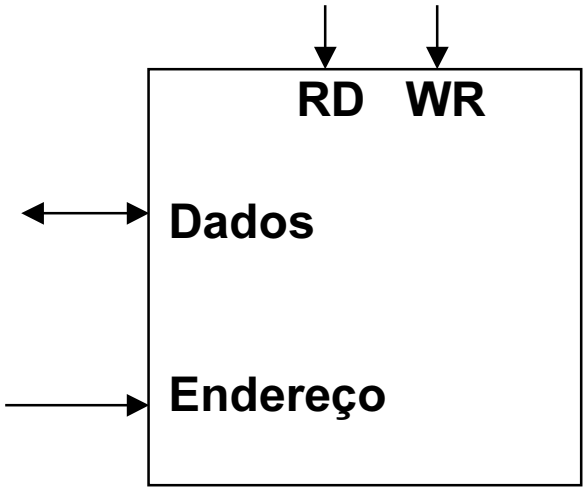


Register File (e Memória)

Register File:
built out D-FF

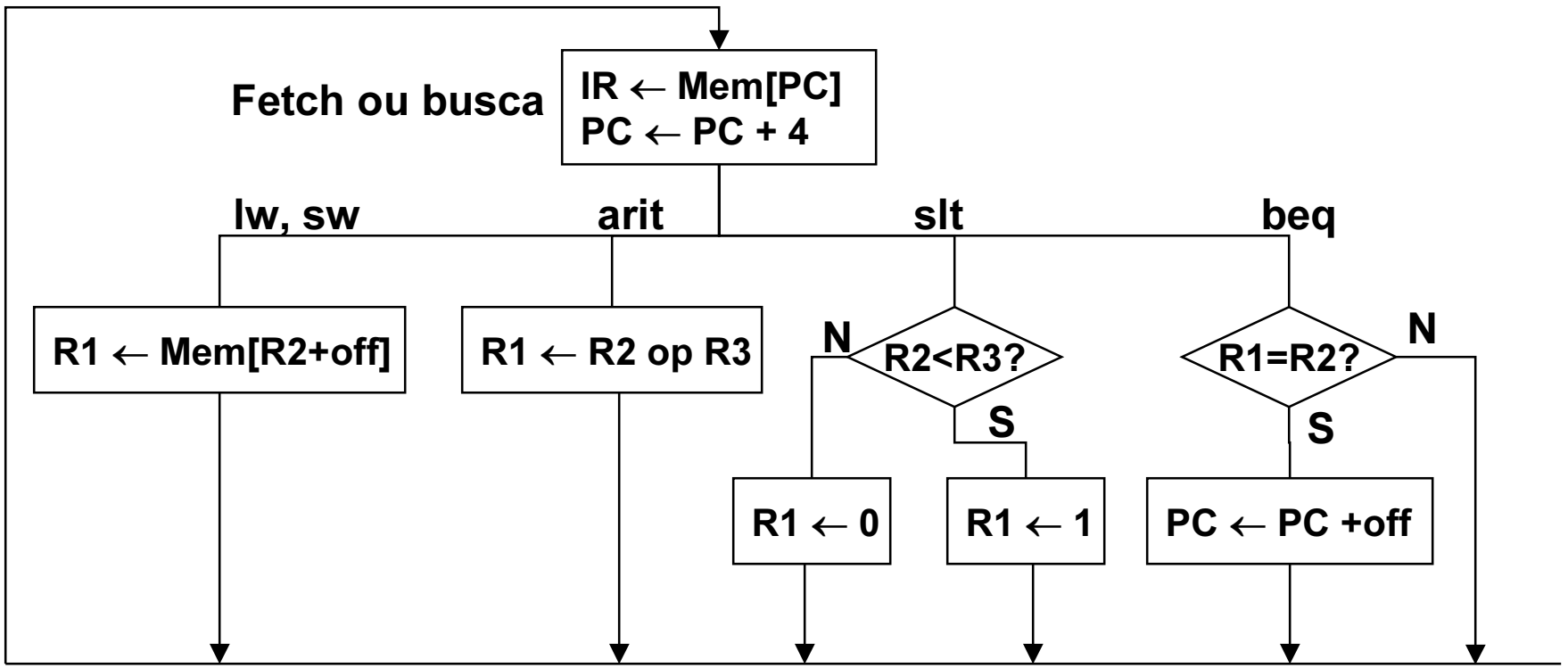


Memória:
• acesso Mem[addr]



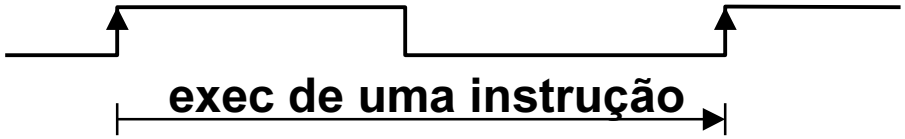
Uma visão simplificada

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`

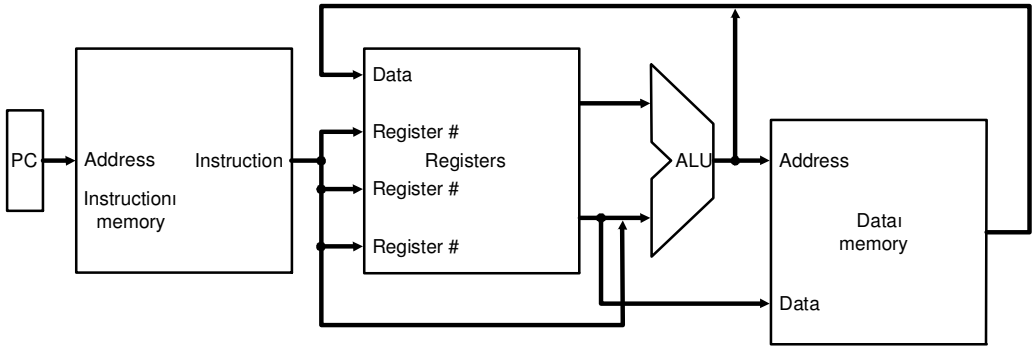


More Implementation Details

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



- **Abstract / Simplified View:**

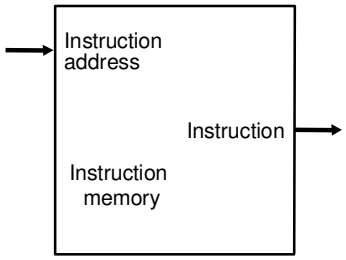


Two types of functional units:

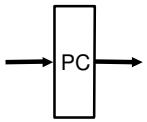
- **elements that operate on data values (combinational)**
- **elements that contain state (sequential)**

Simple Implementation

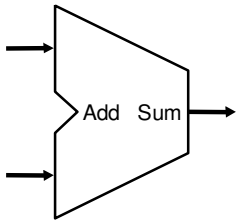
- **Include the functional units we need for each instruction**



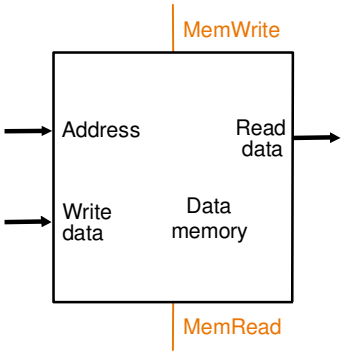
a. Instruction memory



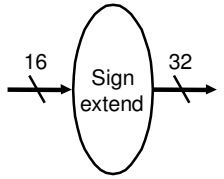
b. Program counter



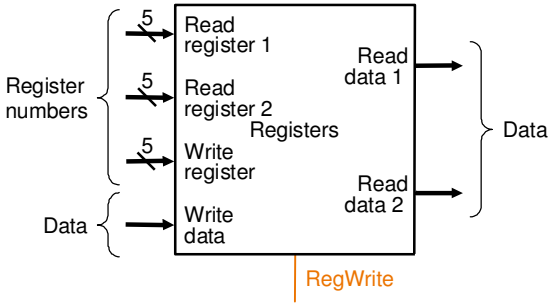
c. Adder



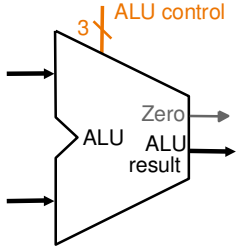
a. Data memory unit



b. Sign-extension unit



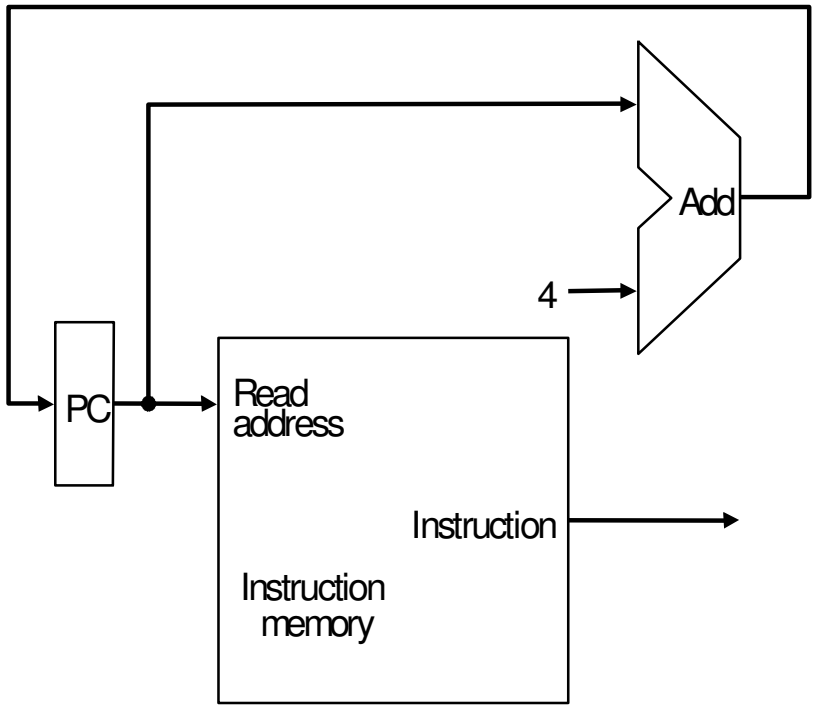
a. Registers



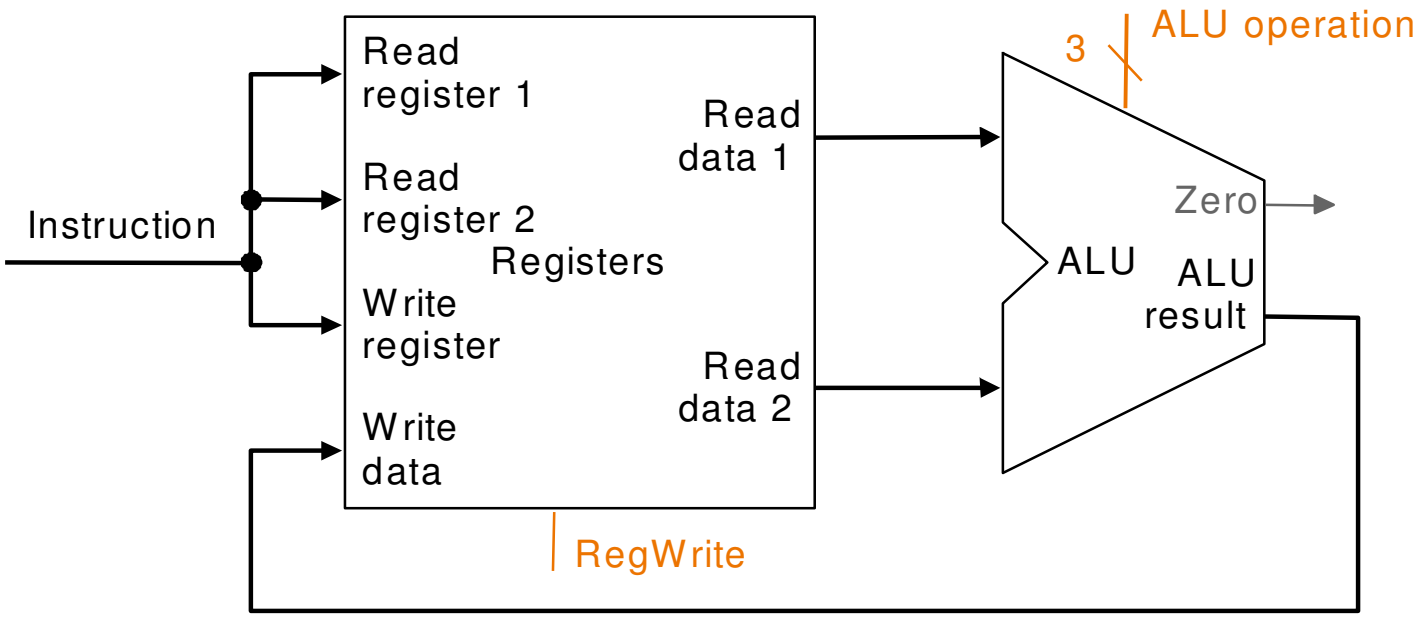
b. ALU

Why do we need this stuff?

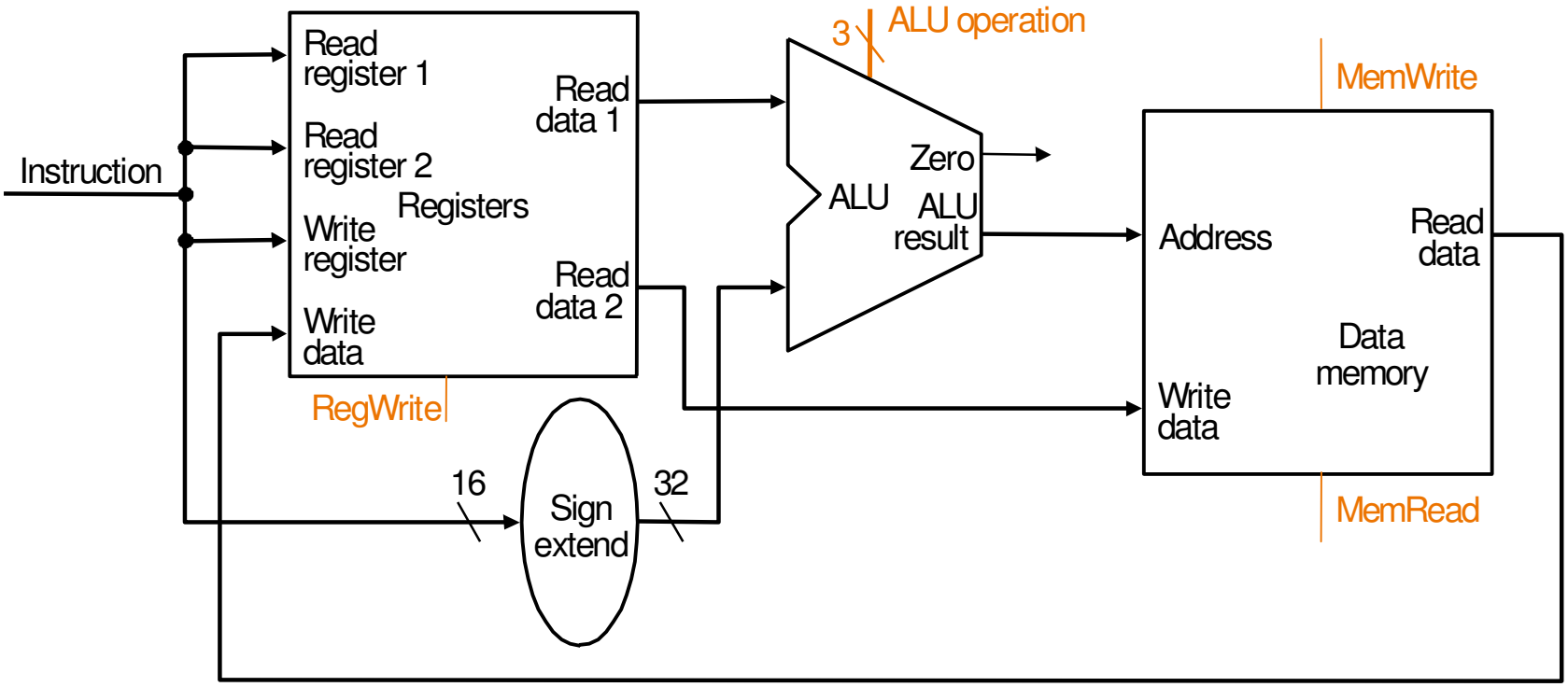
Fetch



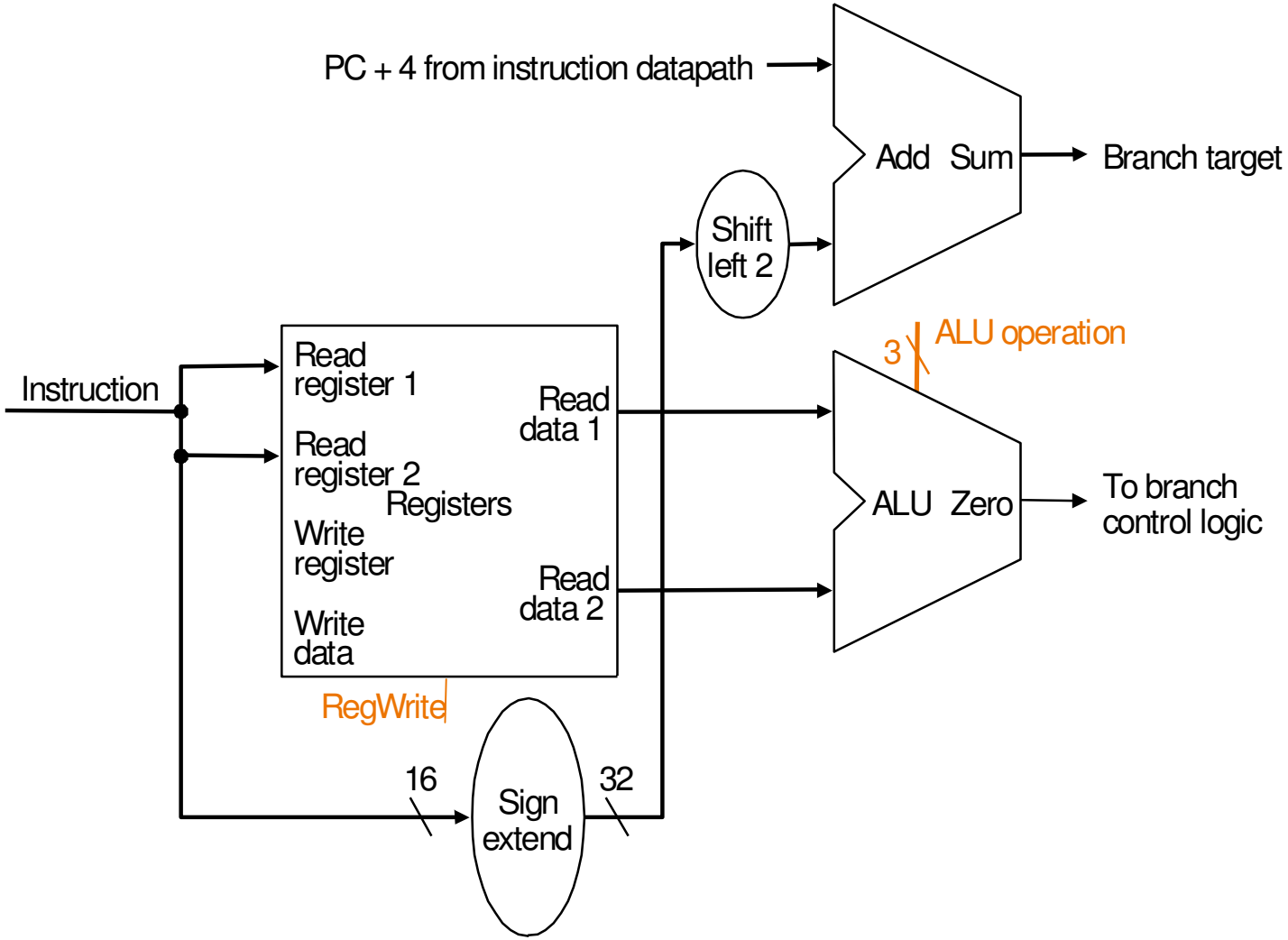
Tipo R



Referência a Memória

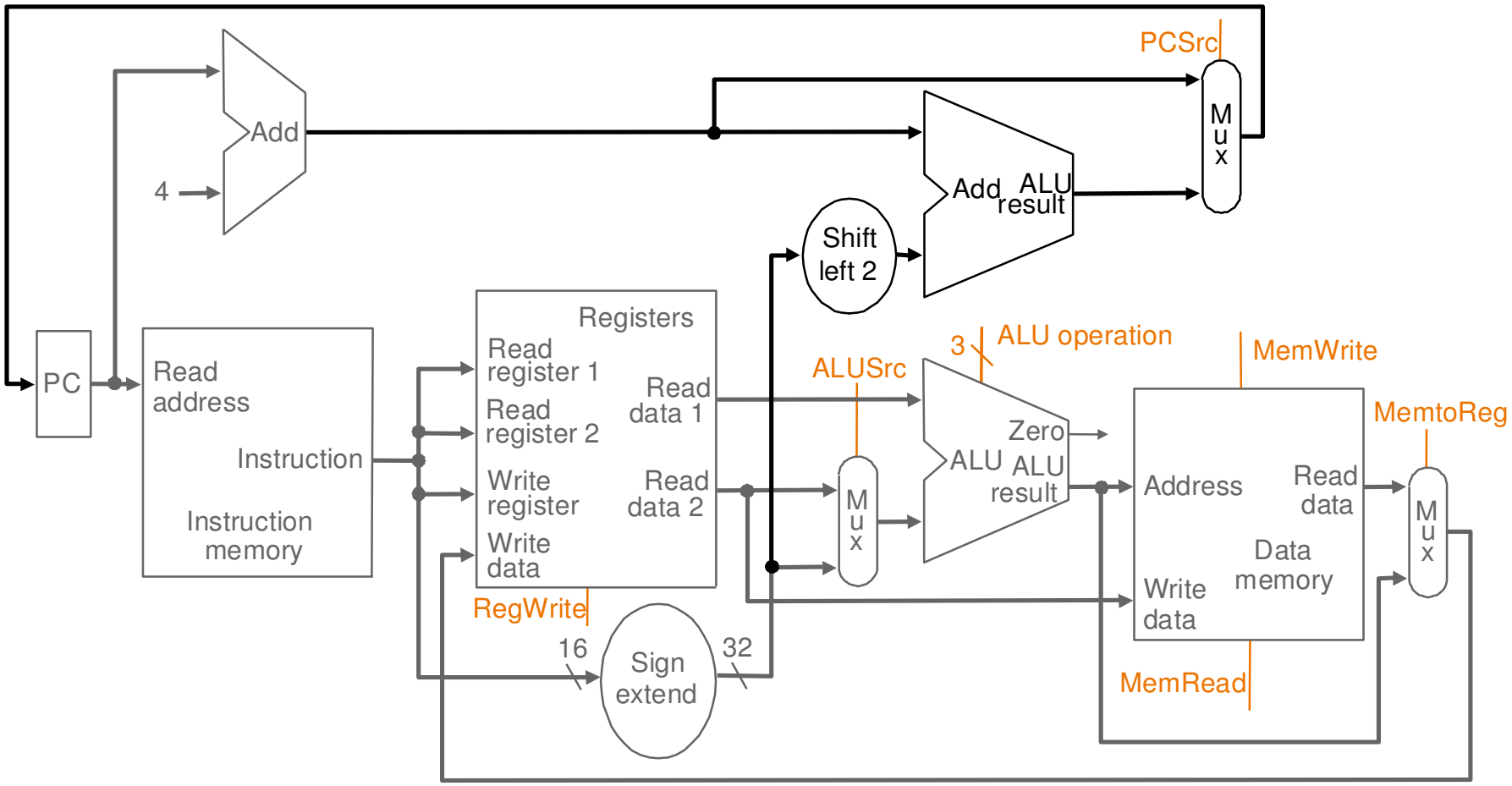


beq



Building the Datapath

- **Use multiplexors to stitch them together**



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)**

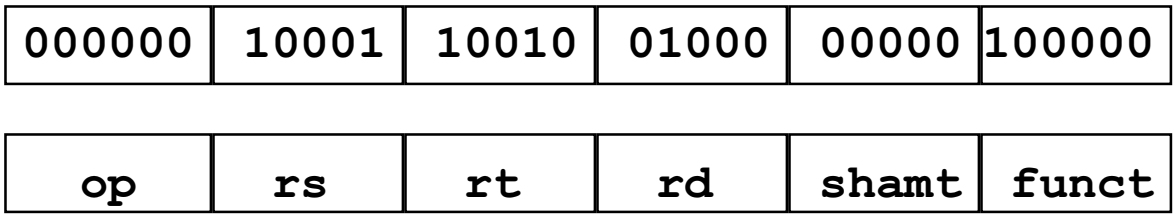
- **MUX 2:1 para escolher o que será carregado no PC**
  - **PC + 4**
  - **PC + SignExt (ShiftLeft (Offset))**

- **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**

Control

- Selecting the operations to perform (ALU, read/write, etc.)
- Controlling the flow of data (multiplexor inputs)
- Information comes from the 32 bits of the instruction
- Example:

add $8, $17, $18 **Instruction Format:**



- ALU's operation based on instruction type and function code

Control

- e.g., what should the ALU do with this instruction
- Example: lw $1, 100($2)

35	2	1	100

op	rs	rt	16 bit offset

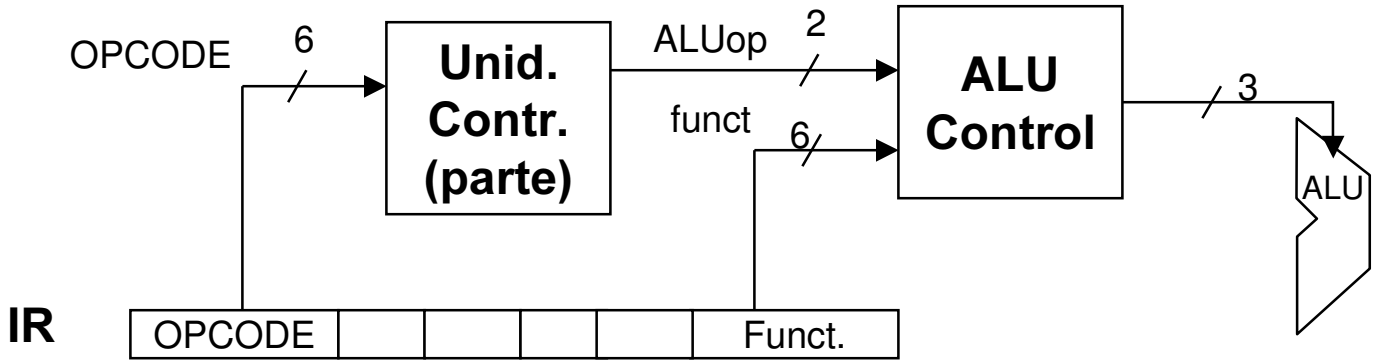
- ALU control input

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

- Why is the code for subtract 110 and not 011?

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

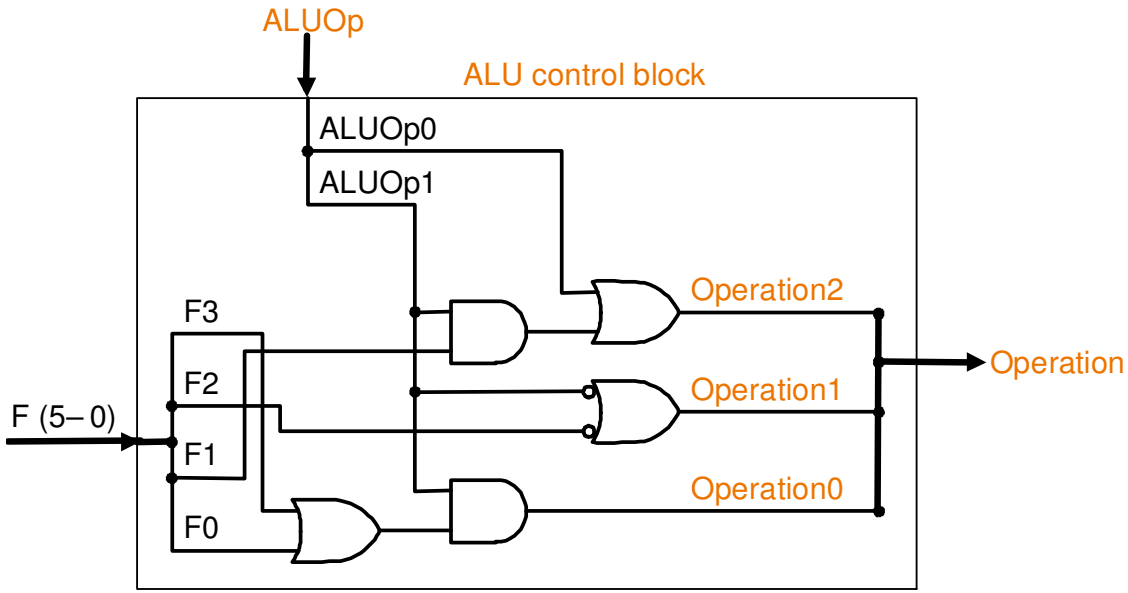


ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

Binv - cod

Inst	Funct.
add	$32=20_H$
sub	$34=22_H$
and	$36=24_H$
or	$37=25_H$
slt	$42=2A_H$

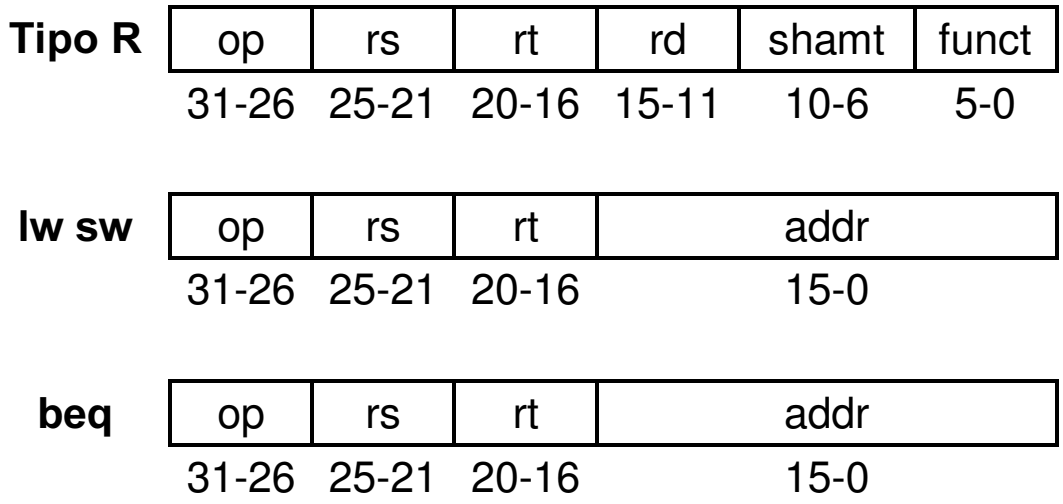
Lógica de controle da ALU



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

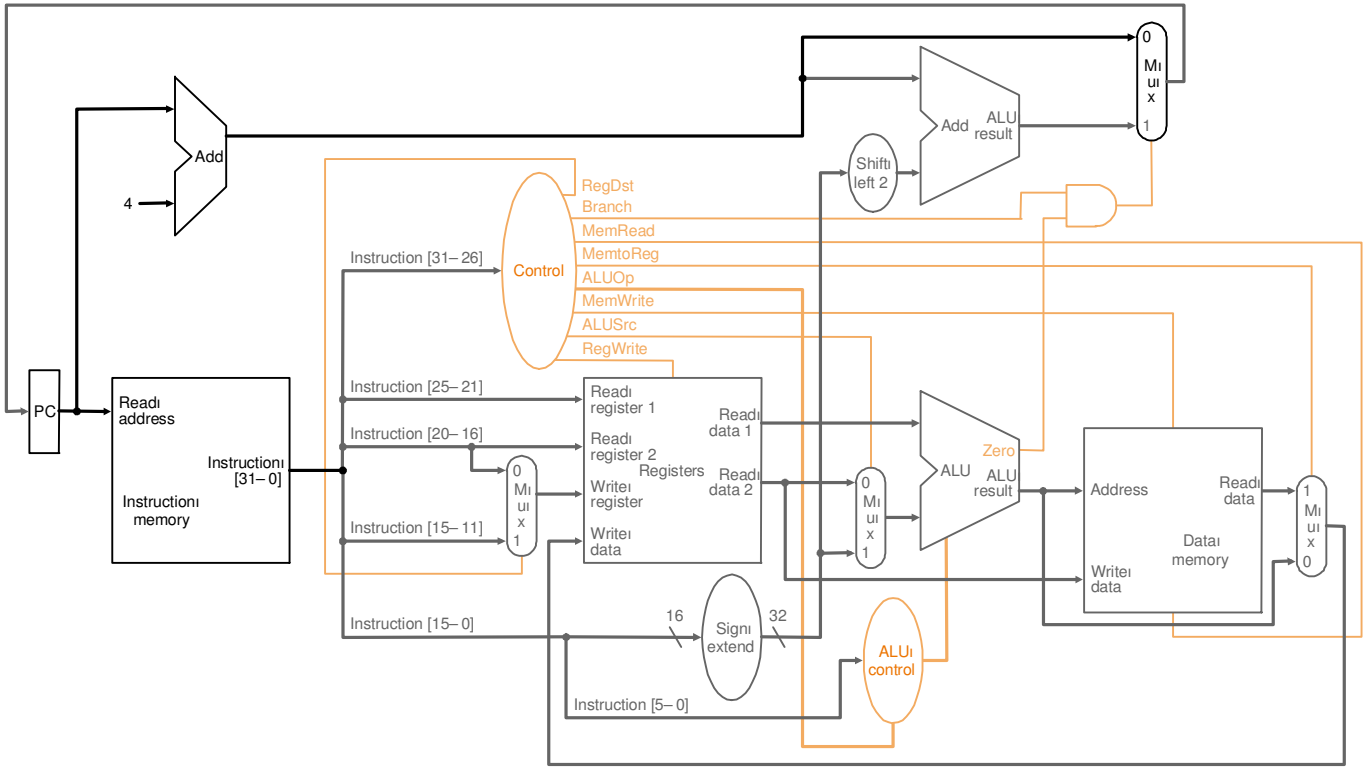
Inst	Funct.
add	$32=20_H$
sub	$34=22_H$
and	$36=24_H$
or	$37=25_H$
slt	$42=2A_H$

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**

Control

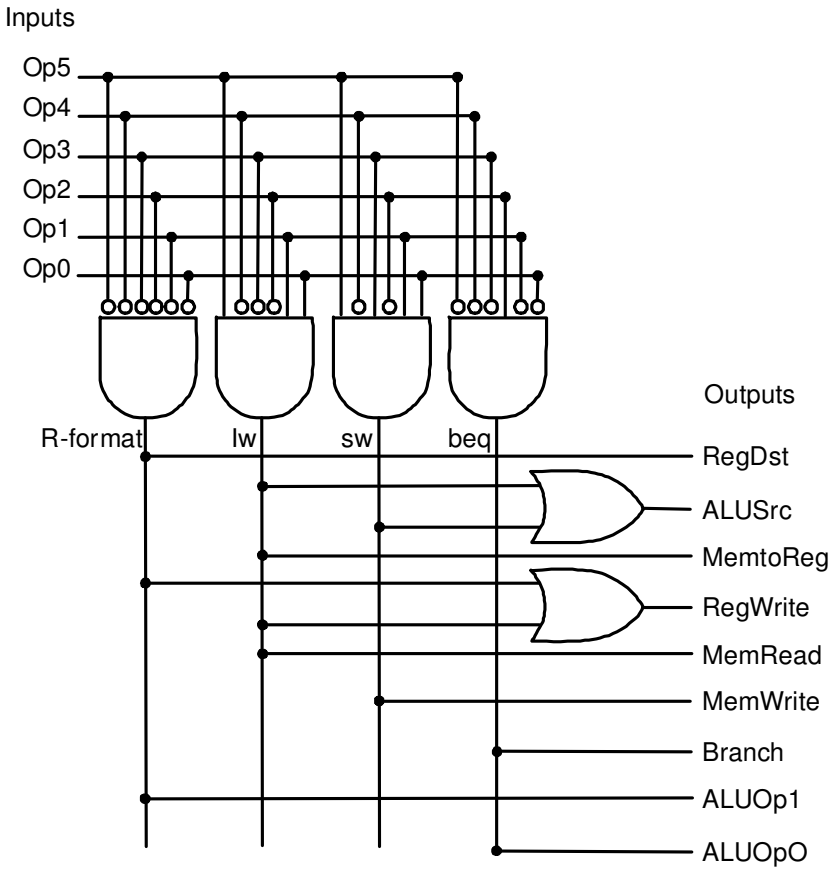


Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1

Unidade de Controle

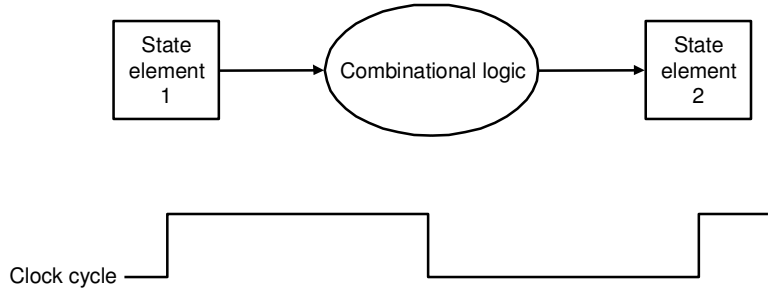
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1

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



Our Simple Control Structure

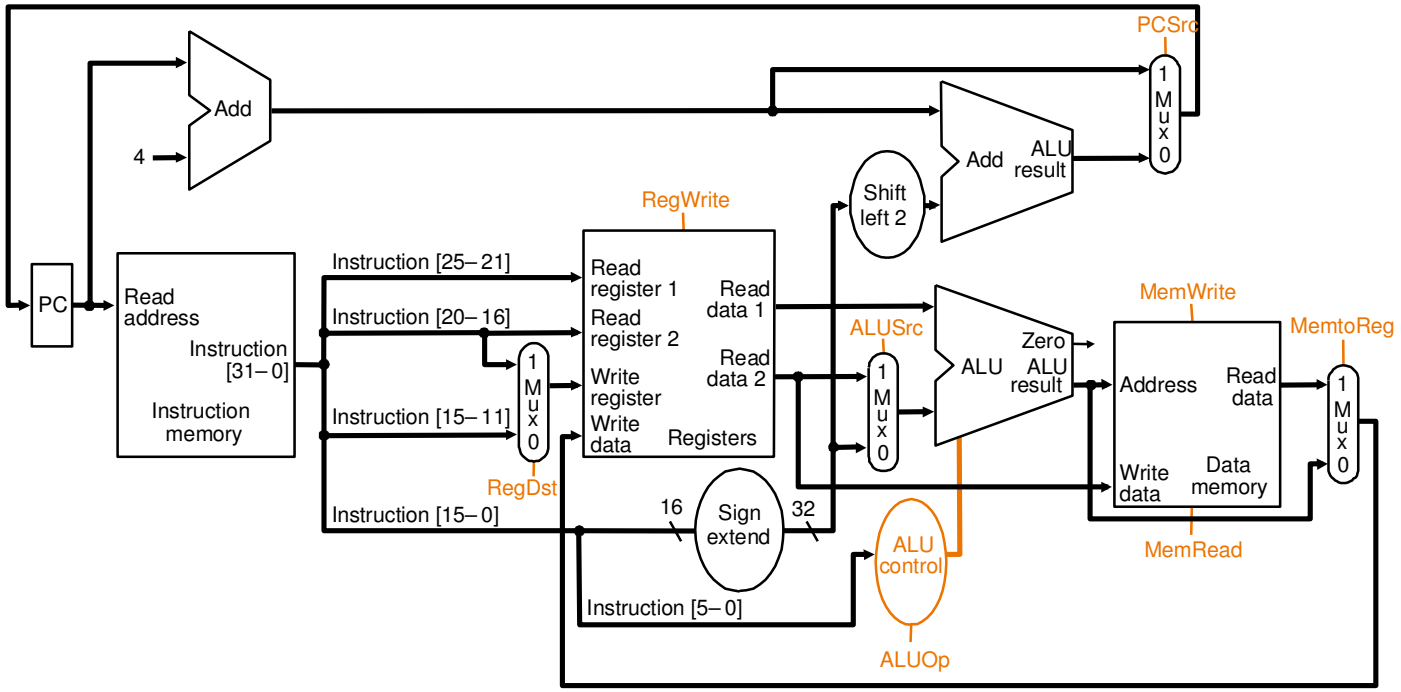
- All of the logic is combinational
- We wait for everything to settle down, and the right thing to be done
  - ALU might not produce “right answer” right away
  - we use write signals along with clock to determine when to write
- Cycle time determined by length of the longest path



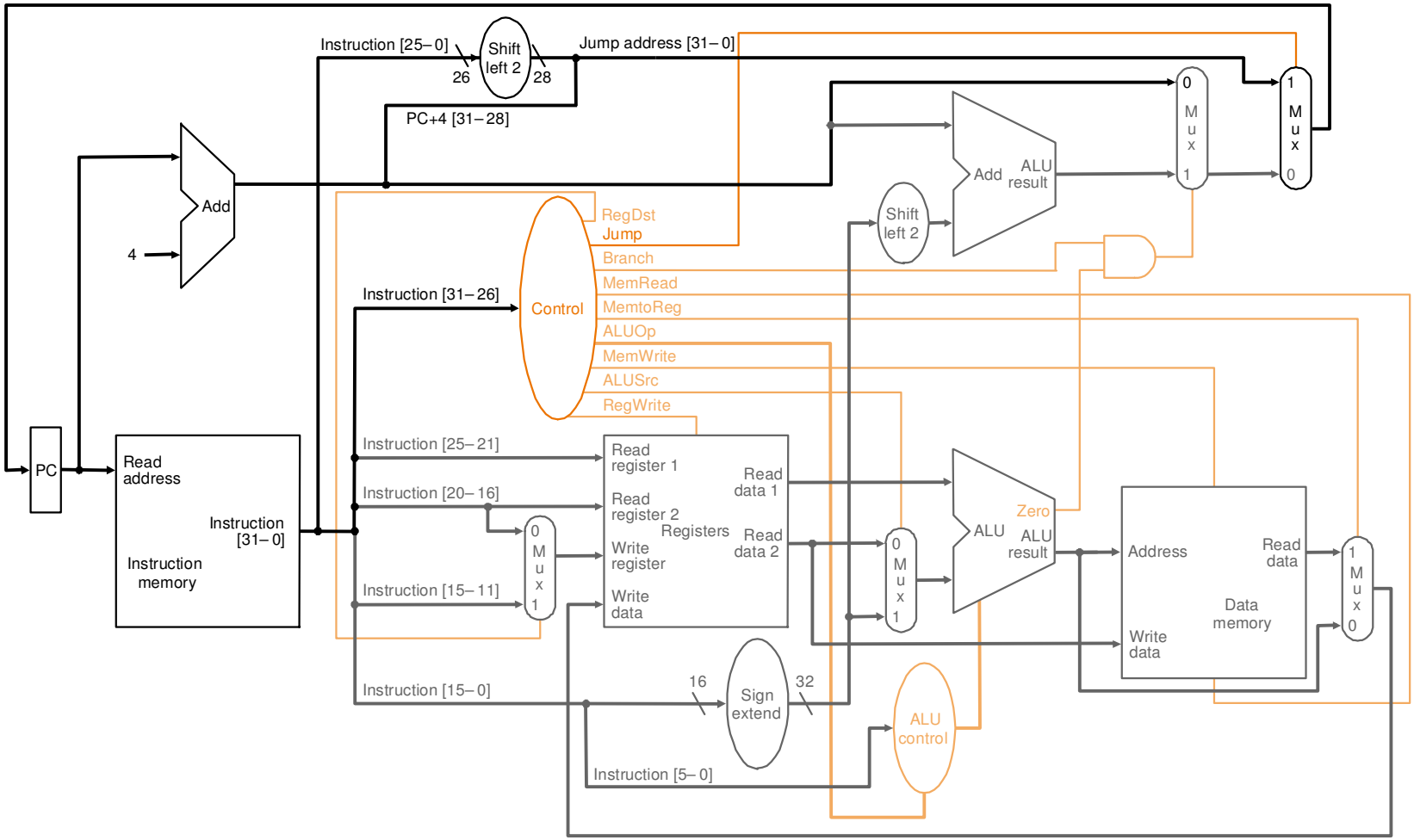
We are ignoring some details like setup and hold times

Single Cycle Implementation

- **Calculate cycle time assuming negligible delays except:**
  - **memory (2ns), ALU and adders (2ns), register file access (1ns)**



Adicionando a instrução jump



2	address
31-26	25-0

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

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