Organização MIPS

Revisão

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

FIGURE 3.19 MIPS instruction formats in Chapter 3. Highlighted portions show instruction formats introduced in this section.

MIPS operands

Name	Example	Comments
32 registers	$s0–$s7, $t0–$t9, $zero, $a0–$a3, $v0–$v1, $gp, $fp, $sp, $ra, $at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zero always equals 0. Register $at is reserved for the assembler to handle large constants.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; data in registers
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3	Three operands; data in registers
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	Used to add constants
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory
	load byte	lb $s1,100($s2)	$s1 = Memory[$s2 + 100]	Byte from memory to register
	store byte	sb $s1,100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	lui $s1,100	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set less than immediate	slti $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare less than constant
Uncondi-tional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

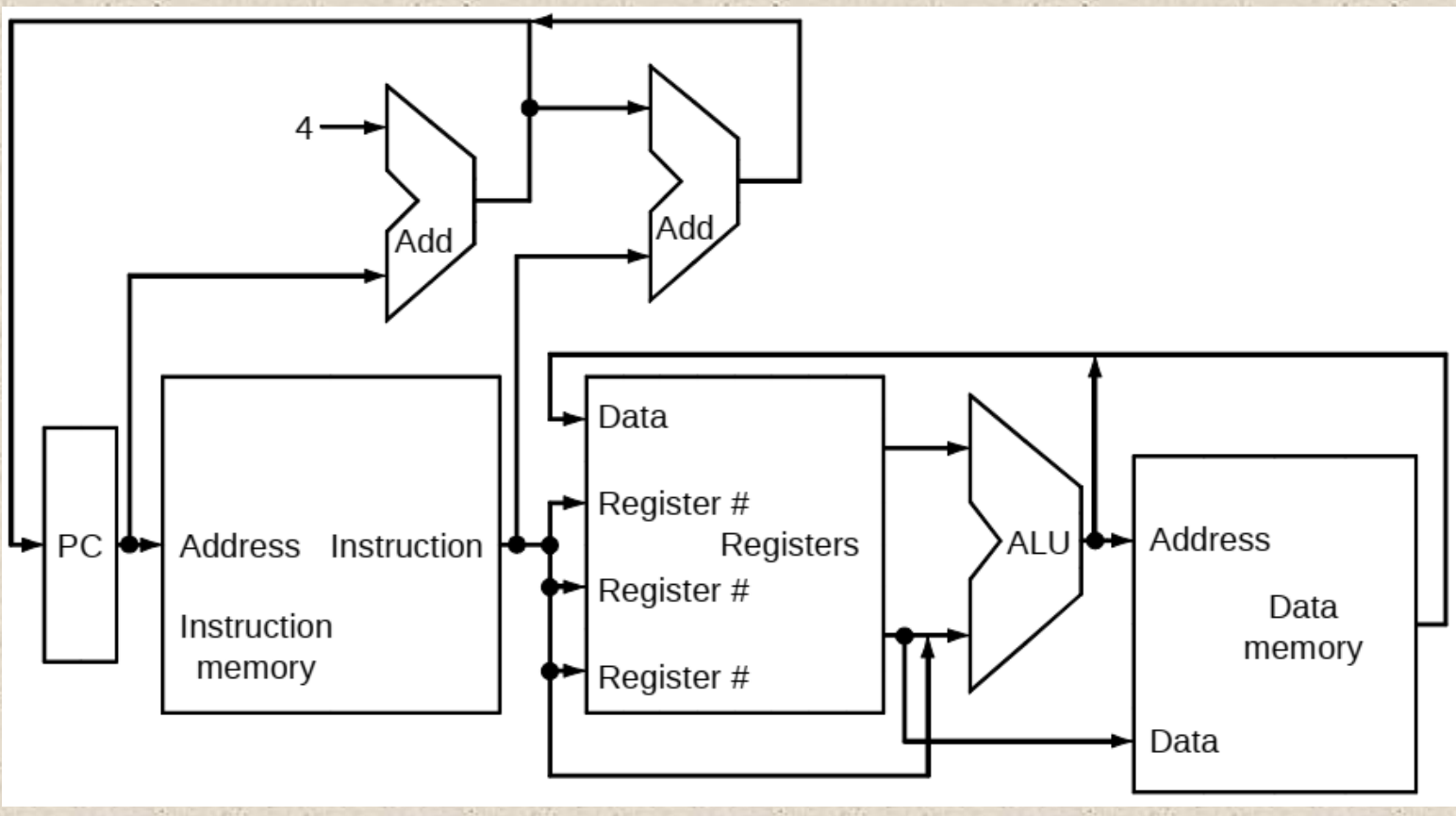
FIGURE 3.20 MIPS assembly language revealed in Chapter 3. Highlighted portions show portions from sections 3.7 and 3.8.

O processador: *datapath* e controle

- Podemos agora conhecer a implementação do MIPS;
- Simplificadamente, ela conterá apenas:
  - Instruções de referência à memória: lw, sw;
  - Instruções lógicas e aritméticas: add, sub, and, or, slt;
  - Instruções de controle de fluxo: beq, j.
- Implementação geral:
  - Use o *program counter* (PC) para oferecer endereços de instruções;
  - Recupere a instrução da memória;
  - Leia registradores;
  - Use a instrução para decidir exatamente o que fazer.
- Todas as instruções usam a ULA depois de ler os registradores
  - Por que? Referência à memória? Aritmética? Controle de fluxo?

Mais detalhes de implementação

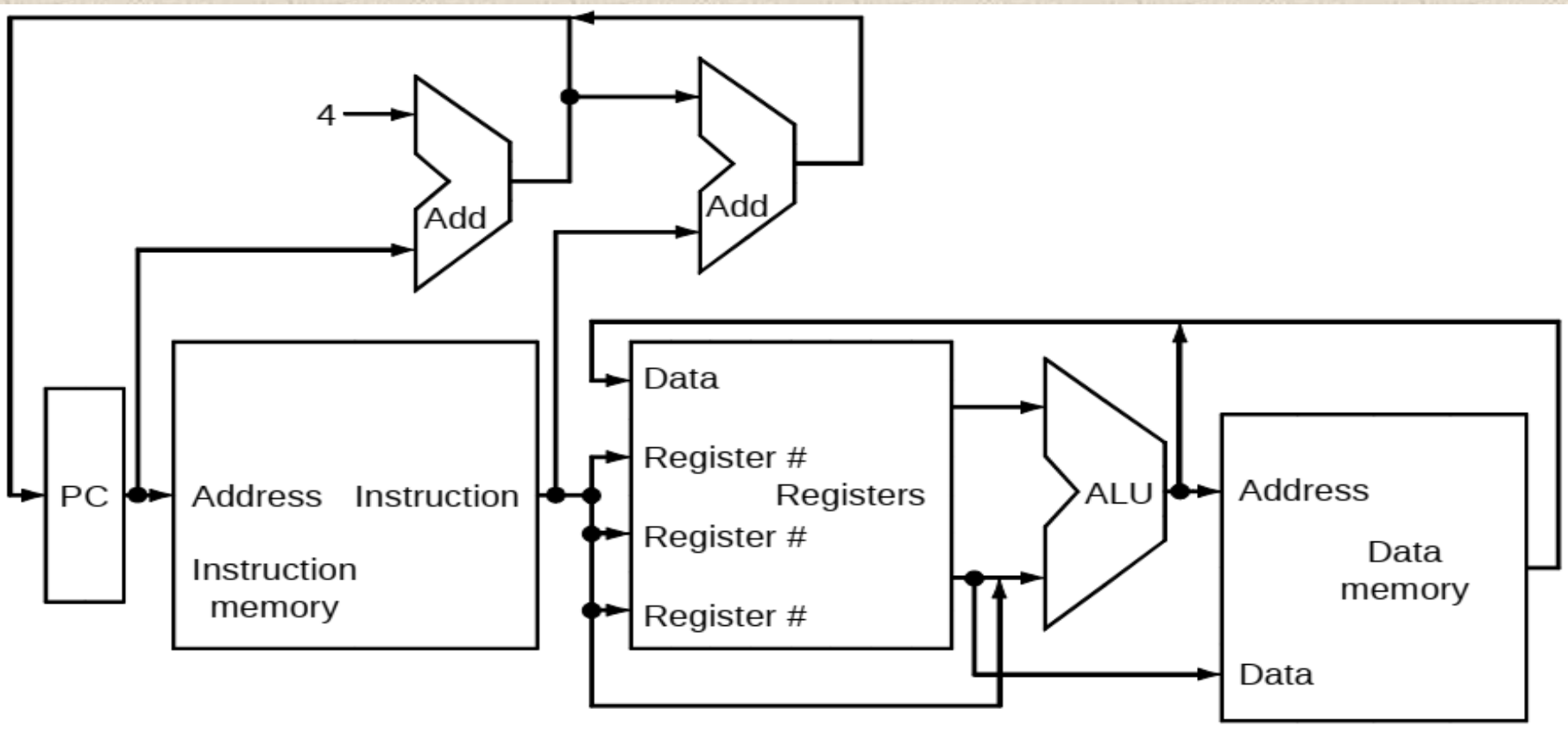
- Visão simplificada/ abstrata:



- Dois tipos de unidades funcionais:
  - Elementos que operam sobre valores de dados (combinacionais): Add e ALU;
  - Elementos que contêm estados (sequenciais): registradores e PC (que é um registrador).

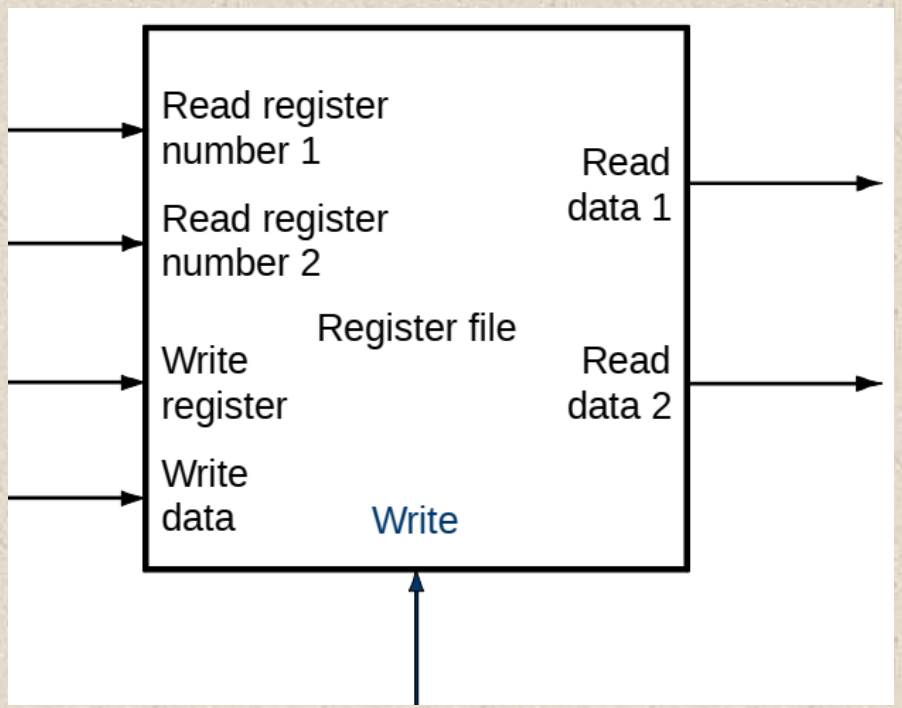
Mais detalhes de implementação

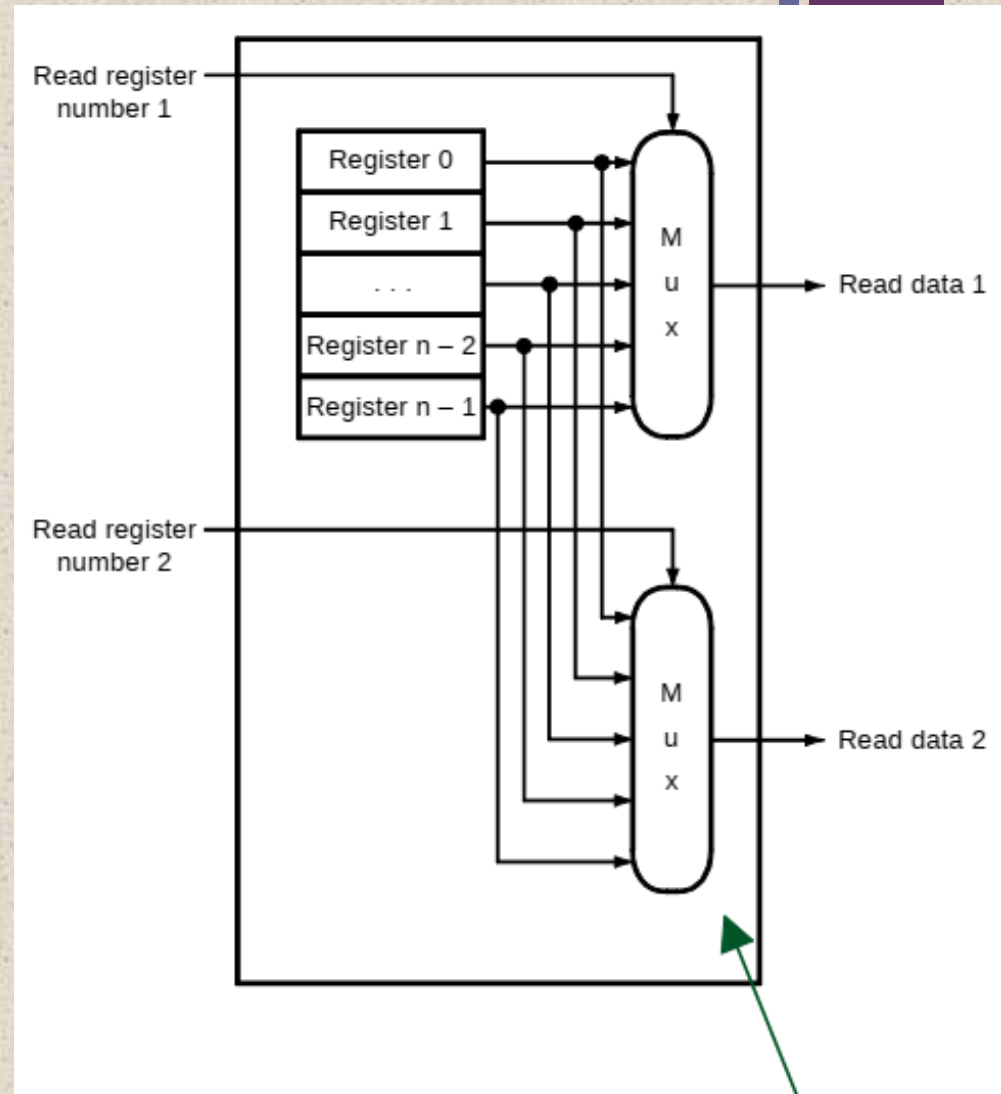
- add $s1, $s2, $s3
- lw $s1, 10($s2)
- beq $s1, $s2, Label (falta sinal de controle na ilustração abaixo)



Leitura de registradores

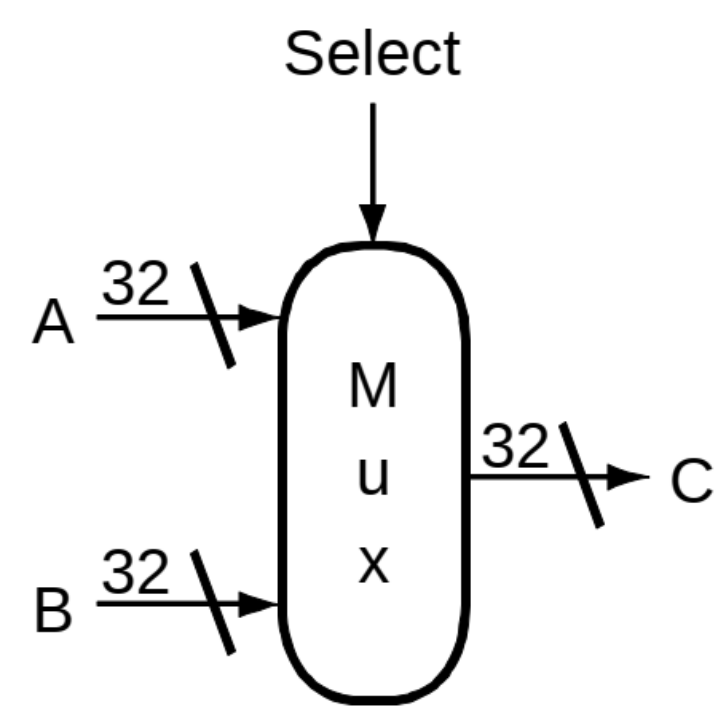
- A construção usando flip-flops D

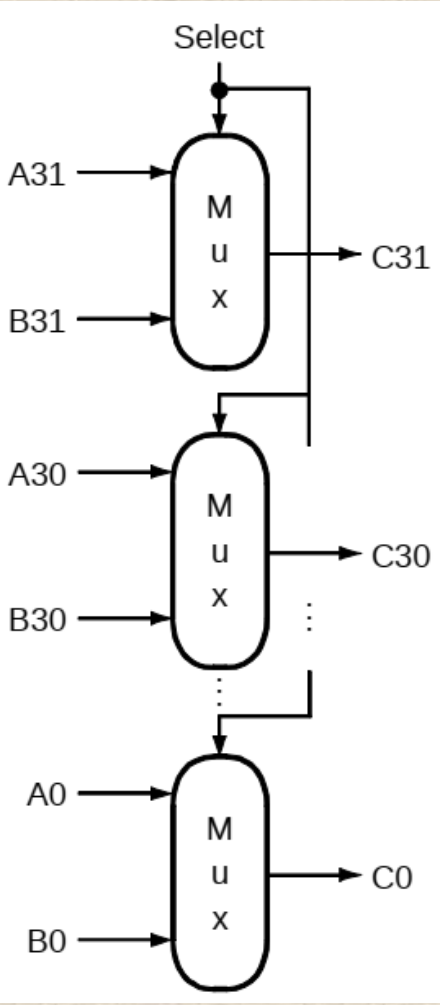




"*Write data*" é o valor a ser armazenado em "*Write register*"

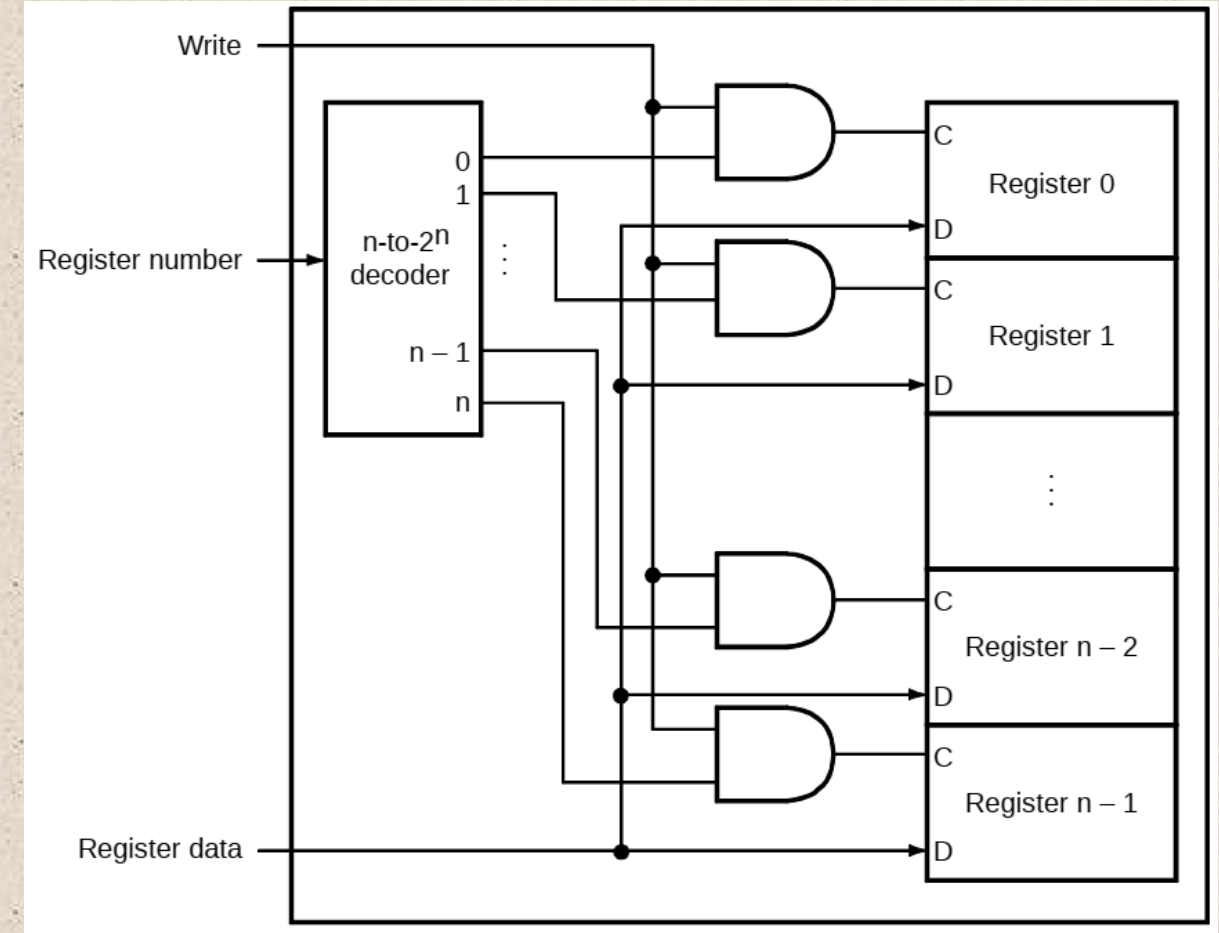
Abstração para leitura de registradores





Escrita em registradores

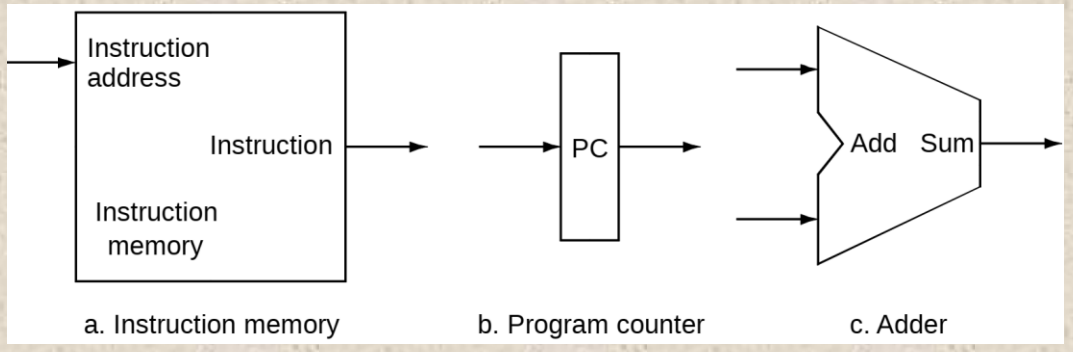
- Note que ainda usamos o clock real para determinar quando escrever

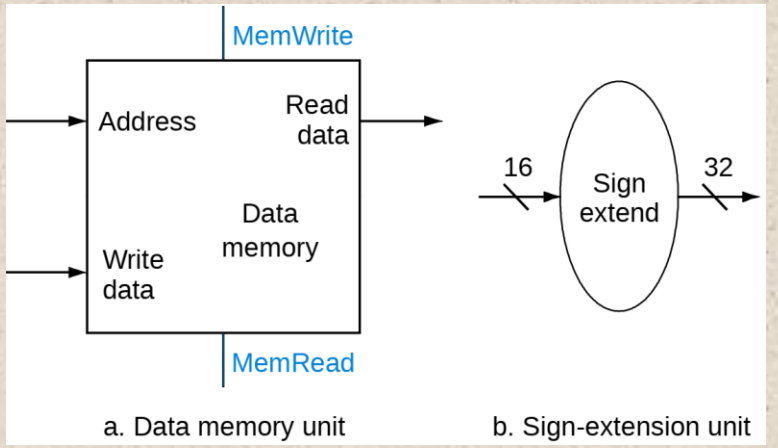


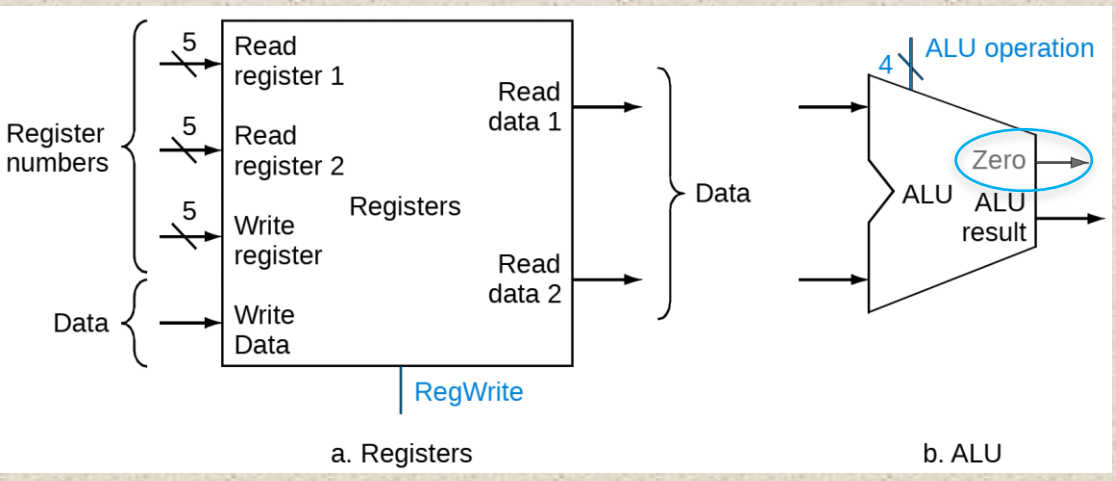
32 bits

Implementação simples

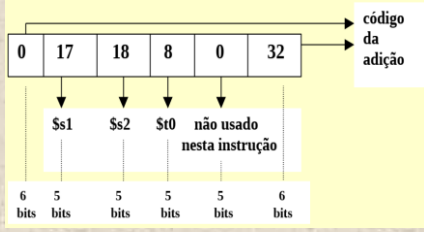
- Inclusão de unidades funcionais que precisamos para cada instrução





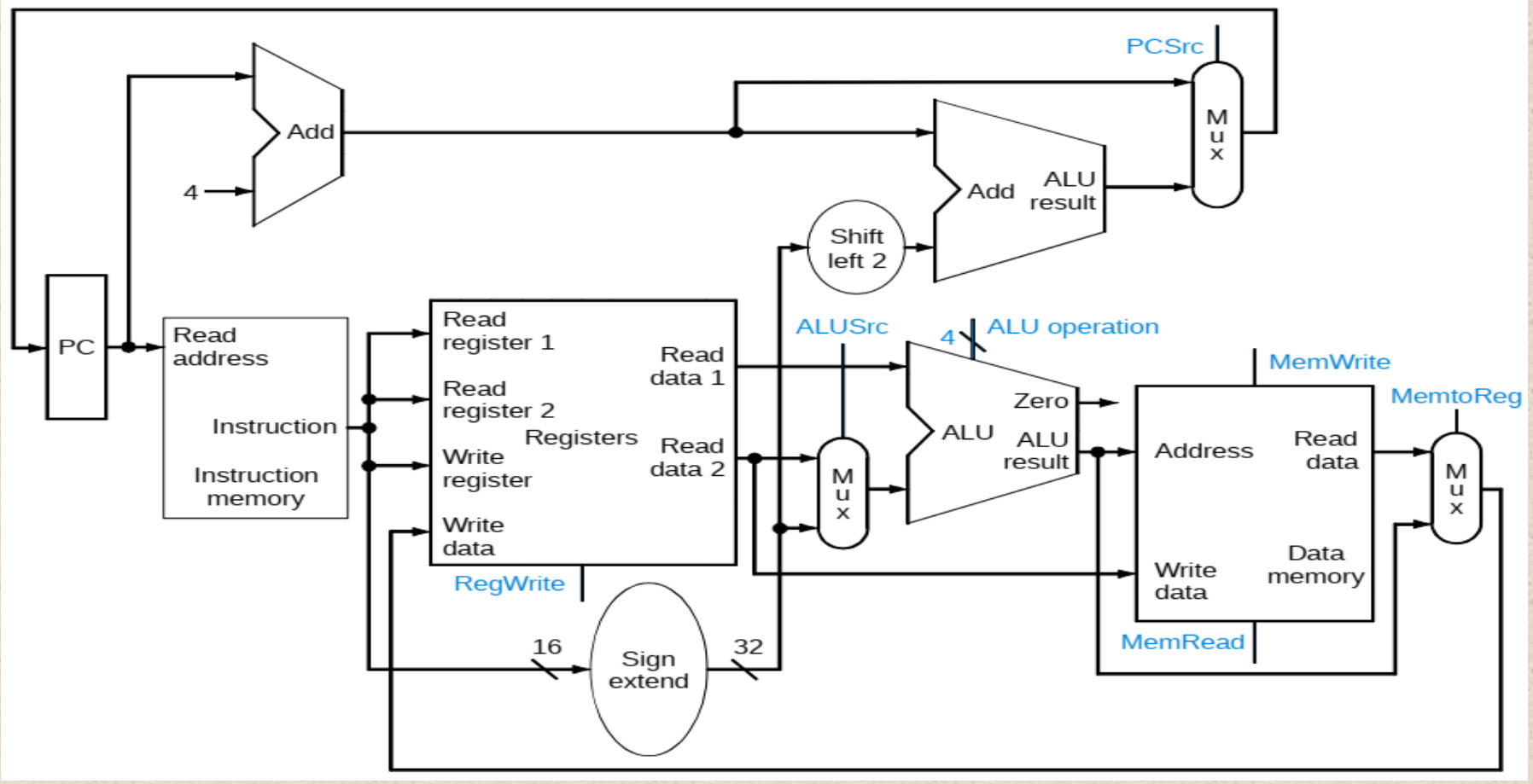


Por que 4 bits?
1. Nem sempre 6 bits são necessários ou
2. Muitas instruções fazem a mesma operação, como add e addi.



Construção do *datapath*

- add $s1, $s2, $s3
- lw $s1, 10($s2)
- beq $s1, $s2, Label



Controle

- Seleção de operações a serem realizadas (ULA, *read/write,* etc);
- Controla o fluxo de dados (entradas de multiplexadores);
- Informação vem dos 32 bits de instrução;

- Exemplo:

  add $8, $17, $18

op	rs	rt	rd	shamt	funct
000000	10001	10010	01000	00000	100000

- Operação da ULA são baseadas no tipo de instrução e código da função.

Controle

- O que a ULA deve fazer com essa instrução?
- Exemplo:
- lw, $1, 100($2)

op	rs	rt	*Offset* de 16bits
35	2	1	100

- Entrada de controle da ULA
  - 0000 AND
  - 0001 OR
  - 0010 *add*
  - 0110 *subtract*
  - 0111 *set-on-less-than*
  - 1100 NOR

Controle

- Deve descrever o hardware para computar a entrada de controle da ULA de 4 bits;
  - Tipo de instrução:
    - 00 = lw, sw
    - 01 = beq,
    - 10 = aritmética
  - Código de função para operações aritméticas

ALUOp
computado a partir do opcode

- Descrição usando uma tabela verdade (que podem ser tornar portas lógicas):

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

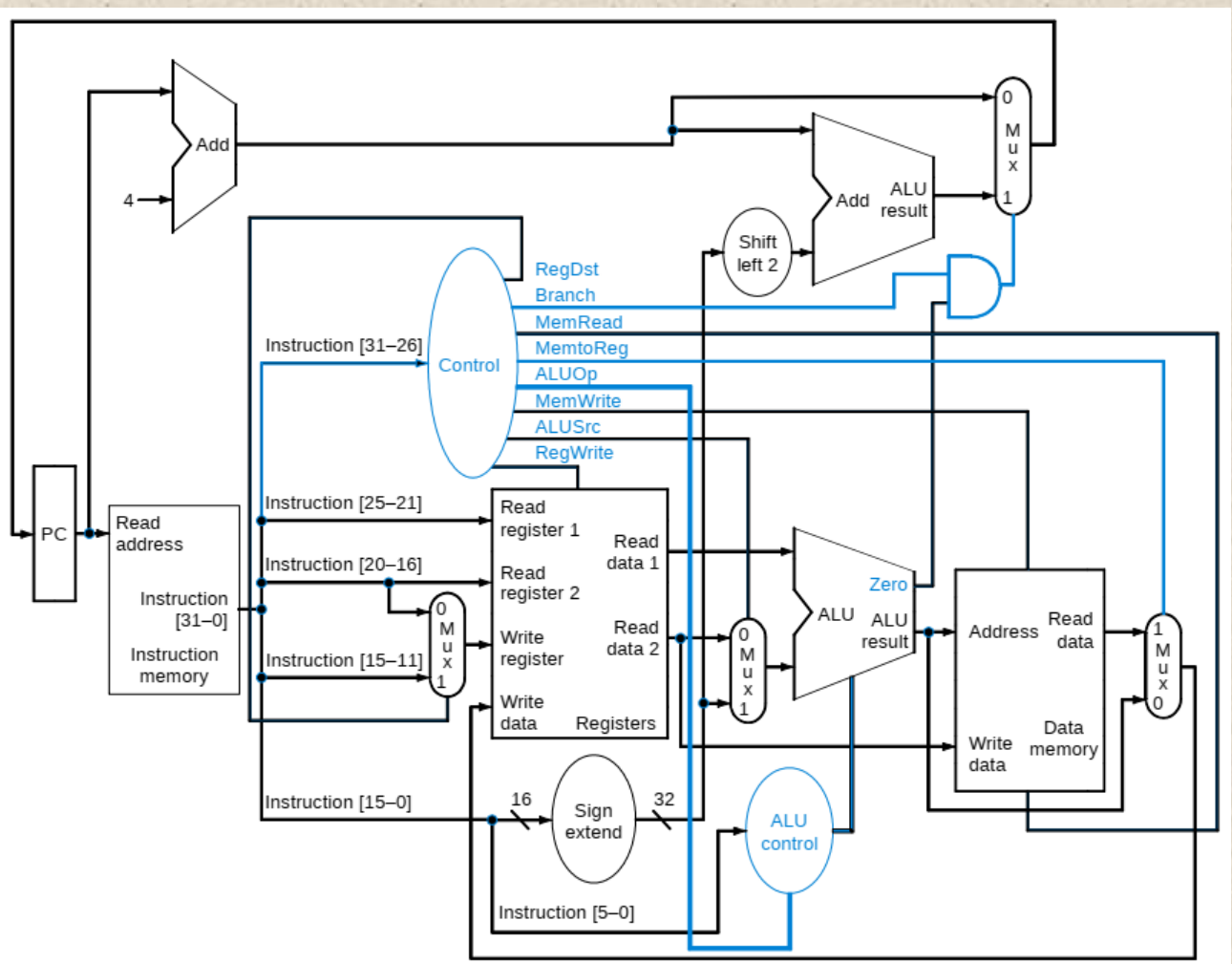
0000 AND
0001 OR
0010 *add*
0110 *subtract*
0111 *set-on-less-than*
1100 NOR

add $s1, $s2, $s3

lw $s1, 10($s2)

beq $s1, $s2, Label

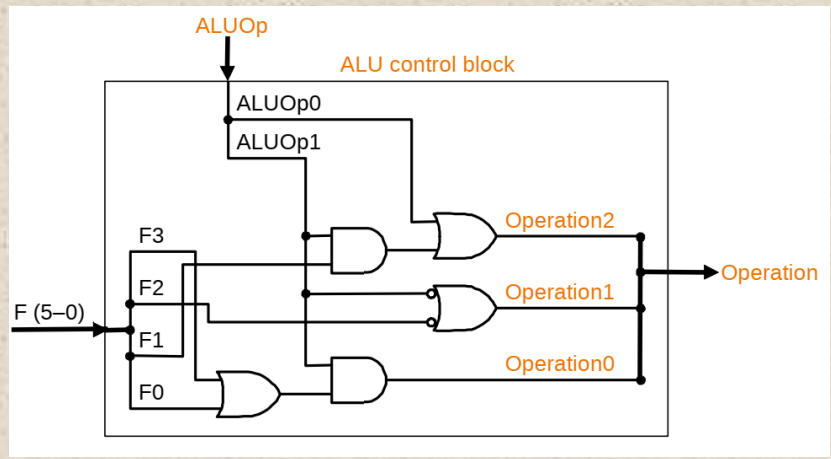
add	$s2	$s3	$s1	0	funct
lw	$s2	$s1	imediato		
beq	$s1	$s2	endereço		

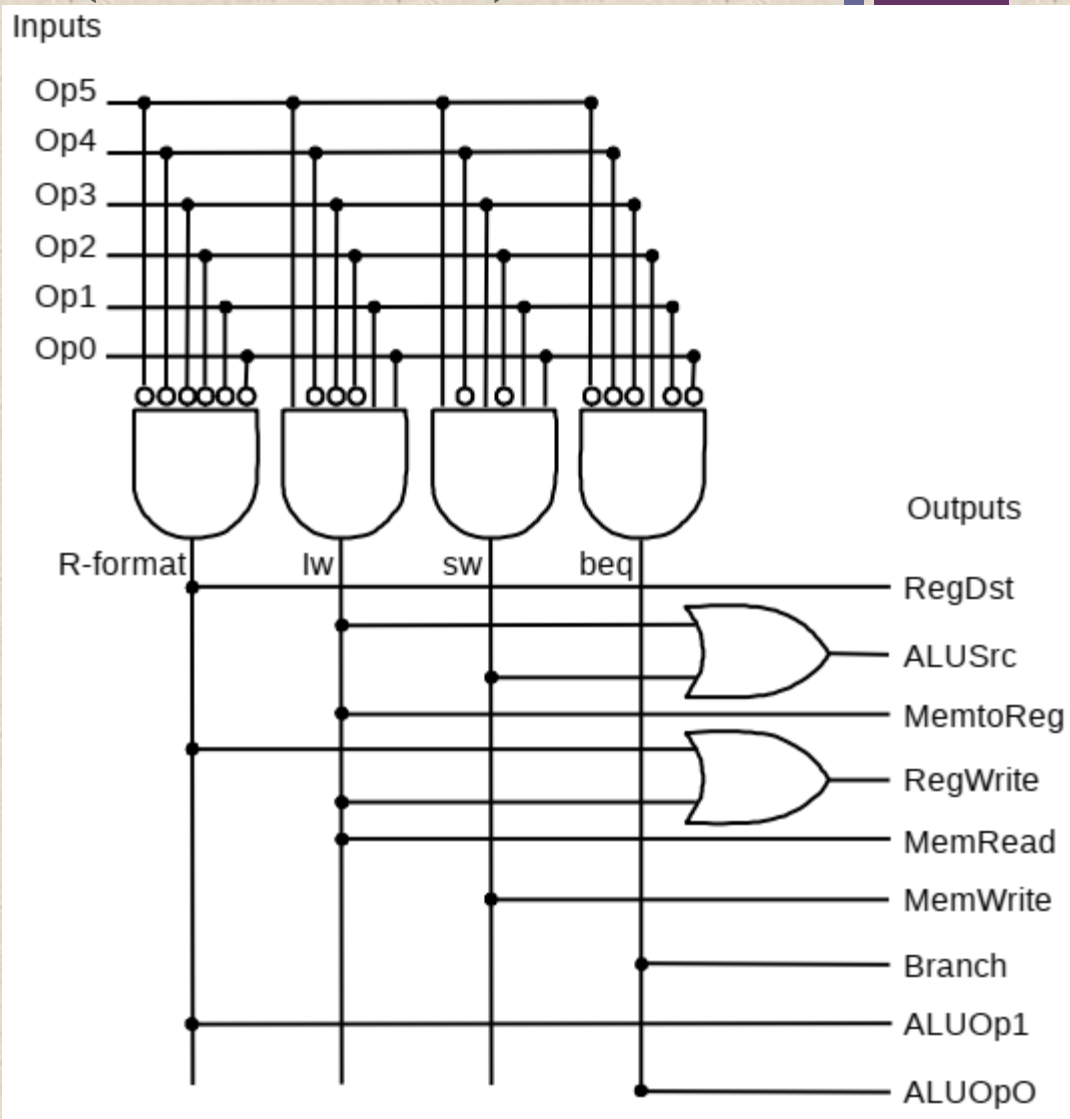


Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Controle

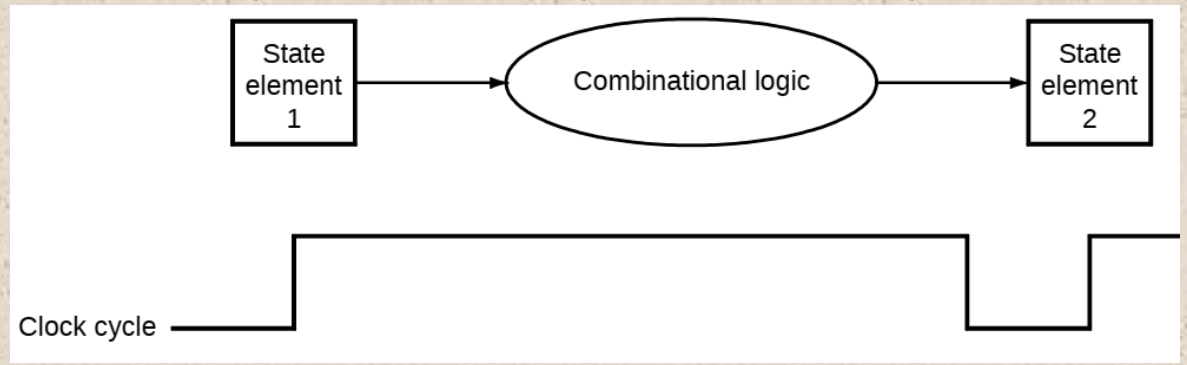
- Lógica combinacional simples (tabelas verdade)





Nossa estrutura de controle simples

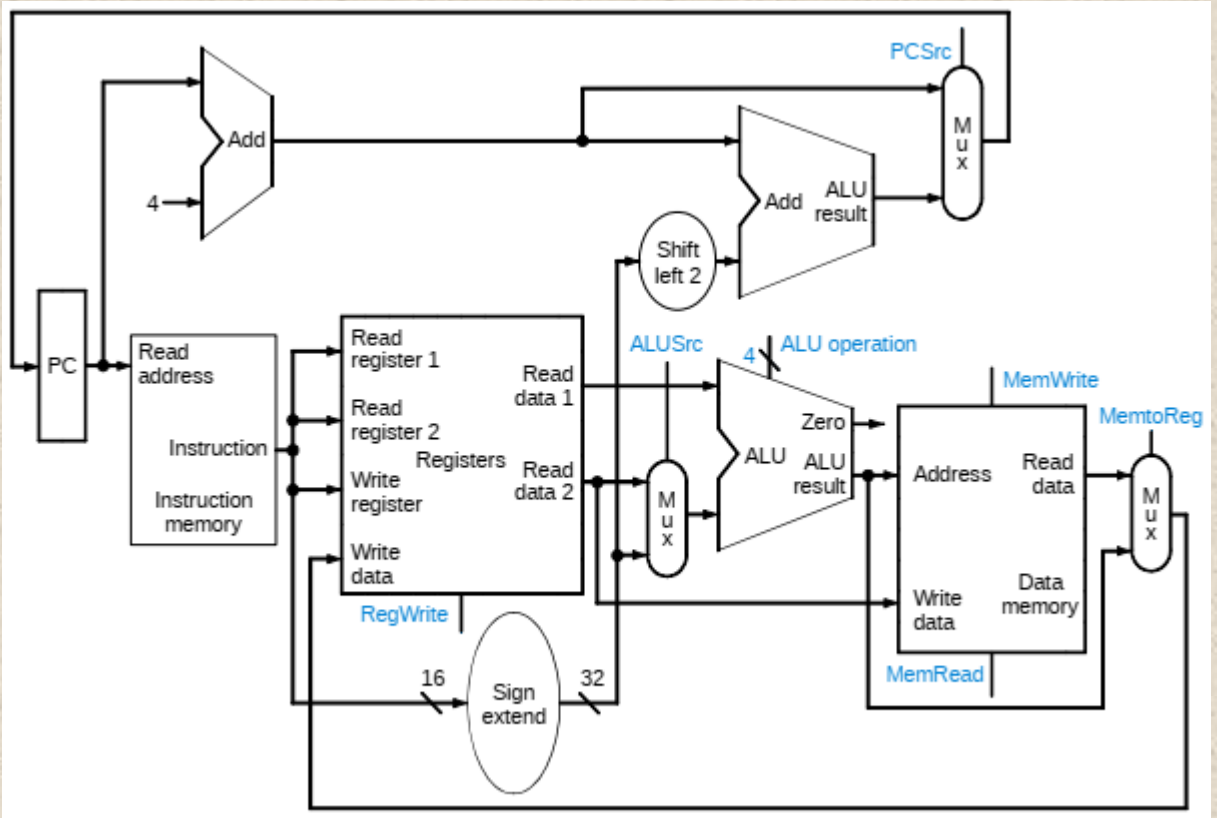
- Todas as operações lógicas são combinacionais;
- Esperamos que tudo esteja sincronizado para que a coisa certa seja feita:
  - ULA pode não produzir a "resposta certa" imediatamente;
  - Usamos sinais de escrita junto com o relógio para determinar quando escrever.
- Os tempos de ciclo são determinados pelo comprimento do caminho mais longo



Implementação de monociclo

- Calcule o tempo de ciclo assumindo atrasos insignificantes, exceto:
  - Memória (200ps),
  - ULA e somadores (100ps),
  - Registrar acesso ao arquivo (50ps).

ps = 10^{-12} do segundo



add = 400ps
lw = 600ps
sw = 550 ps
beq = 350 ps

Portanto, clock de 600 ps

Organização MIPS

Agradeço a Prof. Dr. Fábio A. M. Cappabianco, pelos materiais disponibilizados.