

## Introdução à Arquitetura de Computadores

Aula 23

### μArquitetura Single-cycle - III - Exercícios

#### Instruções adicionais

- **sll** - **s**hift **l**eft **l**ogical
- **lui** - **l**oad **u**pper **i**mmediate
- **slti** - **s**et on **l**ess **t**han **i**mmediate
- **jal** - **j**ump **a**nd **l**ink
- **jr** - **j**ump **r**egister
- **ori** - **o**r **i**mmediate

A. Nunes da Cruz / DETI - UA

Junho / 2021

Slides JP.

#### Exercícios SC (1) - Enunciado

Consulte o **Apêndice B** para a definição das instruções (**Tabelas B.1 e B.2**).  
 Faça uma cópia da **Figura 7.11 (Datapath)** para desenhar as modificações.  
 Assinale os **novos** sinais de controlo.  
 Faça uma cópia da **Tabela 7.3 (Main Decoder)** e da **Tabela 7.2 (ALU Decoder)**  
 para anotar as modificações. Descreva quaisquer outras alterações relevantes.

##### Exercício 7.3

Modifique o CPU Single-cycle para adicionar suporte para **uma** das seguintes instruções:

- (a) **sll**
- (b) **lui**
- (c) **slti**
- (d) **blez**

##### Exercício 7.4

Repita o Exercício 7.3 para as seguintes instruções:

- (a) **jal**
- (b) **lh**
- (c) **jr**
- (d) **srl**
- ori** (extra)

Perguntas do teste →

## Exercícios SC (2) - ApdxB - B.1- Operation Codes - tipo-I

Opcode	Name	Description	Opcode	Name	Description
000000 (0)	R-type	all R-type instructions	011100 (28)	mul rd, rs, rt (func = 2)	multiply (32-bit result)
000001 (1)	bltz rs, label / bgez rs, label	branch less than zero/branch greater than or equal to zero	100000 (32)	lb rt, imm(rs)	load byte
000010 (2)	j label	jump	100001 (33)	lh rt, imm(rs)	load halfword
000011 (3)	jal label	jump and link	100011 (35)	lw rt, imm(rs)	load word
000100 (4)	beq rs, rt, label	branch if equal	100100 (36)	lbu rt, imm(rs)	load byte unsigned
000101 (5)	bne rs, rt, label	branch if not equal	100101 (37)	lhu rt, imm(rs)	load halfword unsigned
000110 (6)	blez rs, label	branch if less than or equal to zero	101000 (40)	sb rt, imm(rs)	store byte
000111 (7)	bgtz rs, label	branch if greater than zero	101001 (41)	sh rt, imm(rs)	store halfword
001000 (8)	addi rt, rs, imm	add immediate	101011 (43)	sw rt, imm(rs)	store word
001001 (9)	addiu rt, rs, imm	add immediate unsigned	110001 (49)	lwc1 ft, imm(rs)	load word to FP coprocessor 1
001010 (10)	slti rt, rs, imm	set less than immediate	111001 (56)	swc1 ft, imm(rs)	store word to FP coprocessor 1
001011 (11)	sltiu rt, rs, imm	set less than immediate unsigned			
001100 (12)	andi rt, rs, imm	and immediate			
001101 (13)	ori rt, rs, imm	or immediate			
001110 (14)	xori rt, rs, imm	xor immediate			
001111 (15)	lui rt, imm	load upper immediate			
010000 (16)	mfc0 rt, rd / mtc0 rt, rd	move from/to coprocessor 0			
010001 (17)	F-type	fop = 16/17: F-type instructions			
010001 (17)	bc1f label/ bc1t label	fop = 8: branch if fpcnd is FALSE/TRUE			

tipo-J: j, jal

tipo-I: slti, ori, lui

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

2/23

## Exercícios SC (3) - ApdxB - B.2- Function Codes - tipo-R

Table B.2 R-type instructions, sorted by funct

Table B.2 R-type instructions, sorted by funct field

Funct	Name	Description	Funct	Name	Description
000000 (0)	sll rd, rt, shamt	shift left logical	100000 (32)	add rd, rs, rt	add
000010 (2)	srl rd, rt, shamt	shift right logical	100001 (33)	addu rd, rs, rt	add unsigned
000011 (3)	sra rd, rt, shamt	shift right arithmetic	100010 (34)	sub rd, rs, rt	subtract
000100 (4)	sllv rd, rt, rs	shift left logical variable	100011 (35)	subu rd, rs, rt	subtract unsigned
000110 (6)	srlv rd, rt, rs	shift right logical variable	100100 (36)	and rd, rs, rt	and
000111 (7)	srav rd, rt, rs	shift right arithmetic variable	100101 (37)	or rd, rs, rt	or
001000 (8)	jr rs	jump register	100110 (38)	xor rd, rs, rt	xor
001001 (9)	jalr rs	jump and link register	100111 (39)	nor rd, rs, rt	nor
001100 (12)	syscall	system call	101010 (42)	slt rd, rs, rt	set less than
001101 (13)	break	break	101011 (43)	sltu rd, rs, rt	set less than unsigned
010000 (16)	mfi rd	move from hi			
010001 (17)	mthi rs	move to hi			
010010 (18)	mflo rd	move from lo			
010011 (19)	mtlo rs	move to lo			
011000 (24)	mult rs, rt	multiply			
011001 (25)	multu rs, rt	multiply unsigned			
011010 (26)	div rs, rt	divide			
011011 (27)	divu rs, rt	divide unsigned			

tipo-R: sll, jr

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

3/23

Exercícios SC (4) - Figura 7.11 - Datapath

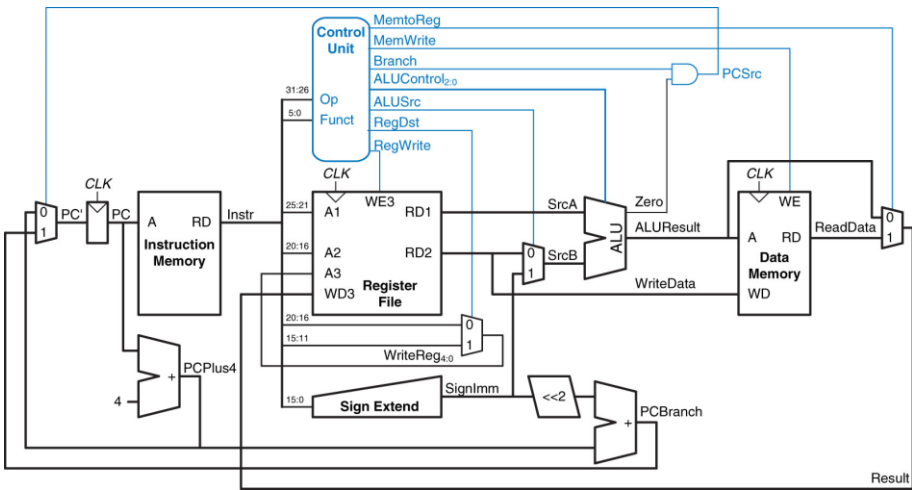
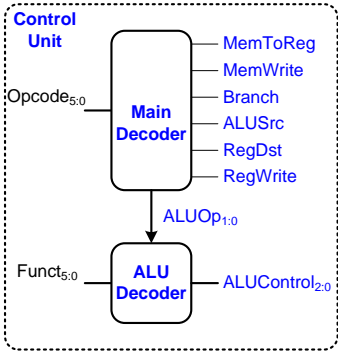


Figura 7.11 Processador MIPS Single-cycle completo

ExercSC (5) - Tabelas de Verdade - ALU + Main Decoders



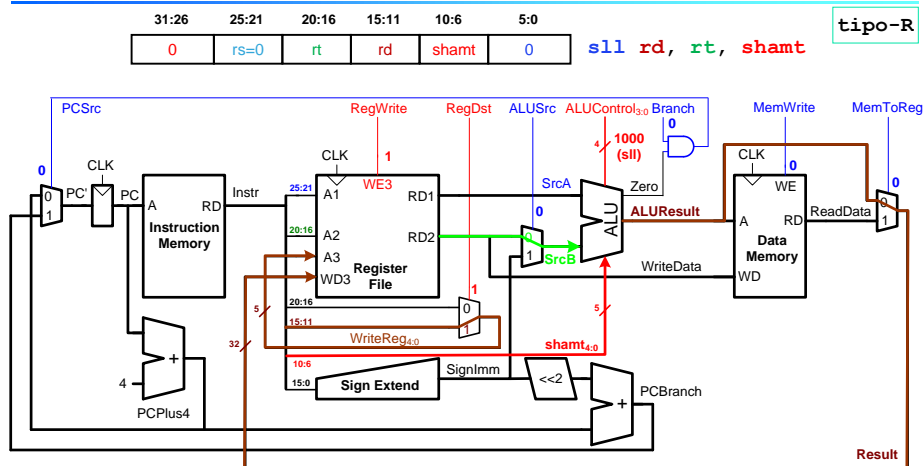
ALUOp <sub>1:0</sub>	Funct <sub>5:0</sub>	ALUControl <sub>2:0</sub>
00	X	010 (Add)
01	X	110 (Subtract)
10	100000 (add)	010 (Add)
10	100010 (sub)	110 (Subtract)
10	100100 (and)	000 (And)
10	100101 (or)	001 (Or)
10	100110 (xor)	100 (Xor)
10	100111 (nor)	101 (Nor)
10	101010 (slt)	111 (Slt)

Tabela 7.2 Tabela de verdade do ALU decoder (+ xor e nor)

Instruction	OpCode <sub>5:0</sub>	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	ALUOp <sub>1:0</sub>
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01

Tabela 7.3 Tabela de verdade do Main decoder

## Problemas SC (1), P7.3a (1) - SLL: Datapath



Modificações:

Datapath single-cycle modificado para sll

1. ALU: tem uma entrada extra,  $shamt_{4:0}$ ; implementa sll.
2.  $ALUControl_{3:0}$  tem 4 bits

$(rd) = (rt) \ll shamt$   
(rs=0, ignorado!)

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

6/23

## Problemas SC (2), P7.3a (2) - SLL: ALU Decoder + ALU

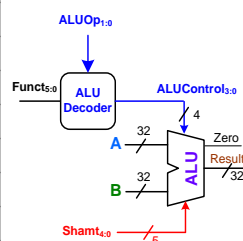
31:26 25:21 20:16 15:11 10:6 5:0

0 rs=0 rt rd shamt 0 sll rd, rt, shamt tipo-R

ALUOp <sub>1:0</sub>	Func <sub>5:0</sub>	ALUControl <sub>3:0</sub>	ALUControl <sub>3:0</sub>	Função
00	X	0010 (Add)	0000	A & B
01	X	0110 (Subtract)	0001	A   B
10	100000 (add)	0010 (Add)	0010	A + B
10	100010 (sub)	0110 (Subtract)	0011	not used
10	100100 (and)	0000 (And)	0100	A ^ B
10	100101 (or)	0001 (Or)	0101	~(A   B)
10	100110 (xor)	0100 (Xor)	0110	A - B
10	100111 (nor)	0101 (Nor)	0111	Slt
10	101010 (slt)	0111 (Slt)	1000	Sll
10	000000 (sll)	1000 (Shift Left Logical)	1001	Srl

Tabela de verdade do ALU decoder para sll

ALU para sll



SLL:  
 $Result = B \ll shamt$   
(A ignorado)

Modificações:

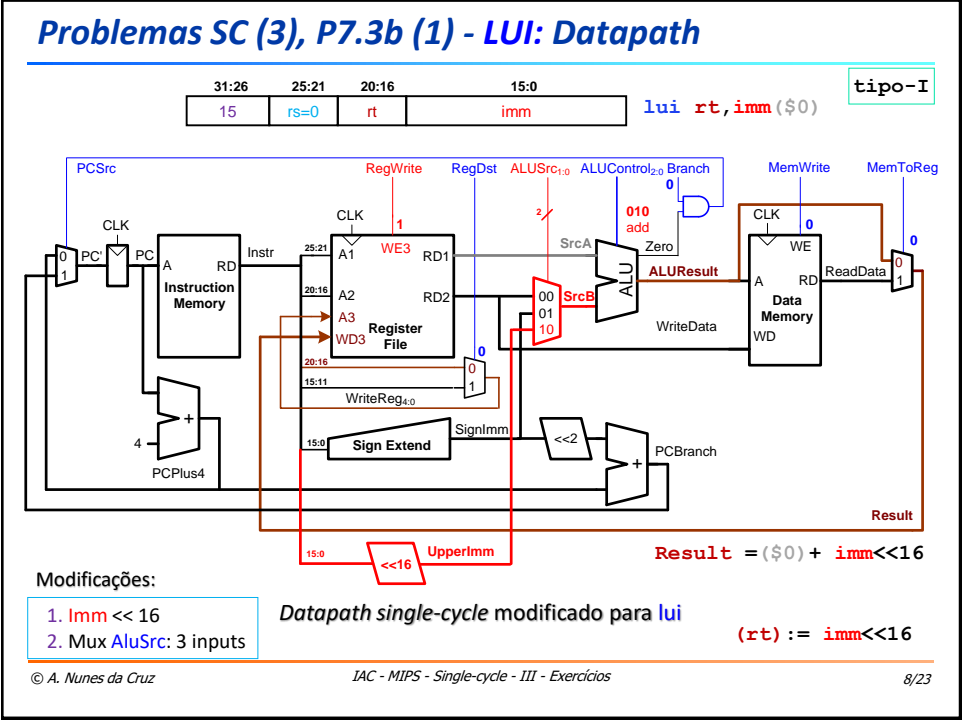
1. ALU: tem uma entrada extra,  $shamt_{4:0}$ ; implementa sll.
2.  $ALUControl_{3:0}$  tem agora 4 bits (+1bit para suportar outros shifts).

$(rd) = (rt) \ll shamt$   
(rs=0, ignorado!)

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

7/23



**Problemas SC (4), P7.3b (2) - LUI: Main Decoder**

Instruction	Op <sub>5:0</sub>	RegWrite	RegDst	AluSrc <sub>1:0</sub>	Branch	MemWrite	MemToReg	ALUOp <sub>1:0</sub>
R-type	000000	1	1	00	0	0	0	10
lw	100011	1	0	01	0	0	1	00
sw	101011	0	X	01	0	1	X	00
beq	000100	0	X	00	1	0	X	01
lui	001111	1	0	10	0	0	0	00

AluSrc<sub>1:0</sub> = 10 → SrcB = UpperImm

ALUOp<sub>1:0</sub> = 00 → AluCtrl<sub>2:0</sub> = 010 (add)

Tabela de verdade do ALU decoder para lui

© A. Nunes da Cruz IAC - MIPS - Single-cycle - III - Exercícios 9/23

### Problemas SC (5), P7.3c (1) - STLI: ALU + Main Decoders

10	rs	rt	imm	slt <sub>i</sub> rt,rs,imm	tipo-I
----	----	----	-----	----------------------------	--------

slt<sub>i</sub> é uma instrução do tipo-I, por isso precisamos de usar um novo código ALUOp = 11. No caso de addi (vide aula anterior) não era necessário porque o código ALUOp = 00 já existia.

ALUOp <sub>1:0</sub>	Funct <sub>5:0</sub>	ALUControl <sub>2:0</sub>
00	X	010 (Add)
01	X	110 (Subtract)
10	100000 (add)	010 (Add)
10	100010 (sub)	110 (Subtract)
10	100100 (and)	000 (And)
10	100101 (or)	001 (Or)
10	101010 (slt)	111 (SlT)
11	X	111 (SlT)

O datapath não precisa de ser modificado! Só a Unidade de Controlo tem de ser adaptada.

$$(rt) := ((rs) < SignImm) ? 1 : 0$$

Tabela de verdade do ALU decoder para slt<sub>i</sub>

Instruction	OpCode <sub>5:0</sub>	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	ALUOp <sub>1:0</sub>
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01
slt <sub>i</sub>	001010	1	0	1	0	0	0	11

Tabela de verdade do Main decoder para slt<sub>i</sub>

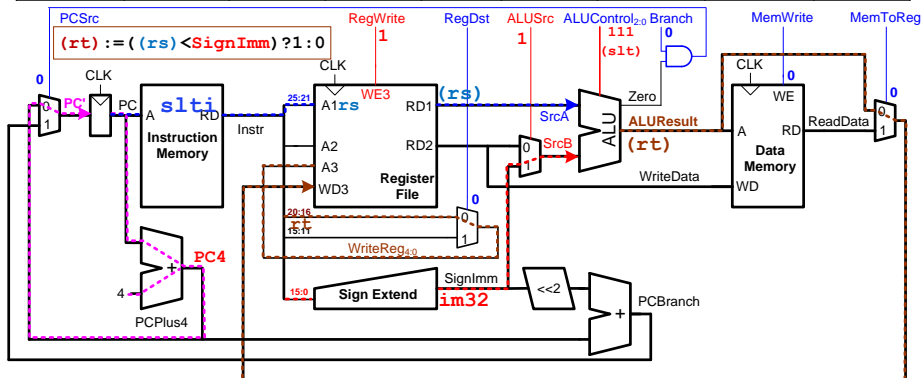
© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

10/23

### Problemas SC (6), P7.3c (2) - STLI: Datapath

Instruction	Op <sub>5:0</sub>	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	ALUOp <sub>1:0</sub>
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01
slt <sub>i</sub>	001010	1	0	1	0	0	0	11

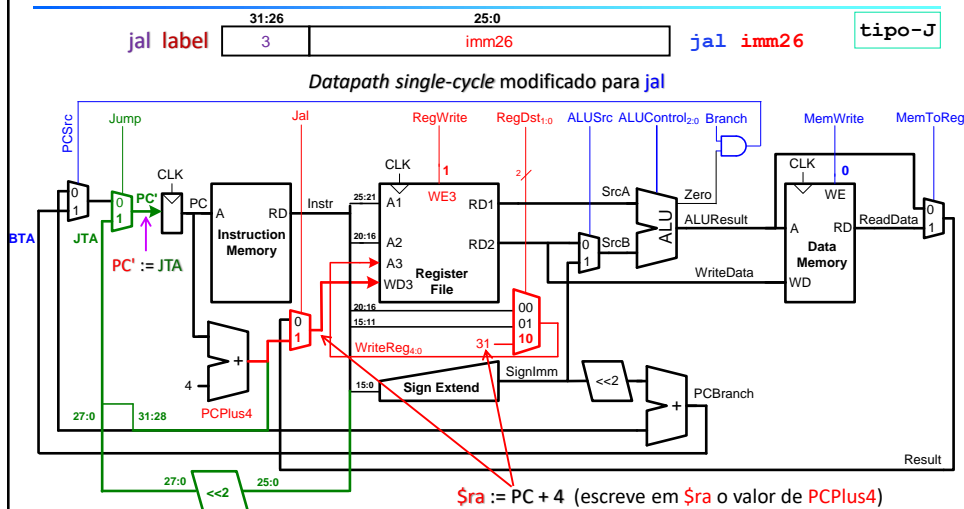


© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

11/23

### Problemas SC (7), P7.4a (1) - JAL: Datapath



© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

12/23

### Problemas SC (8), P7.4a (2) - JAL: Main Decoder

Instruction	Op <sub>5:0</sub>	RegWrite	RegDst <sub>1:0</sub>	AluSrc	Branch	MemWrite	MemToReg	ALUOp <sub>1:0</sub>	Jump	Jal
R-type	000000	1	01	0	0	0	0	10	0	0
lw	100011	1	00	1	0	0	1	00	0	0
sw	101011	0	XX	1	0	1	X	00	0	0
beq	000100	0	XX	0	1	0	X	01	0	0
addi	001000	1	00	1	0	0	0	00	0	0
j	000010	0	XX	X	X	0	X	XX	1	0
jal	000011	1	10	X	X	0	X	XX	1	1

↑  
 $\$ra$ ↑  
JTA↑  
PC4Tabela de Verdade do *Main decoder* para jalModificações *Main decoder*:

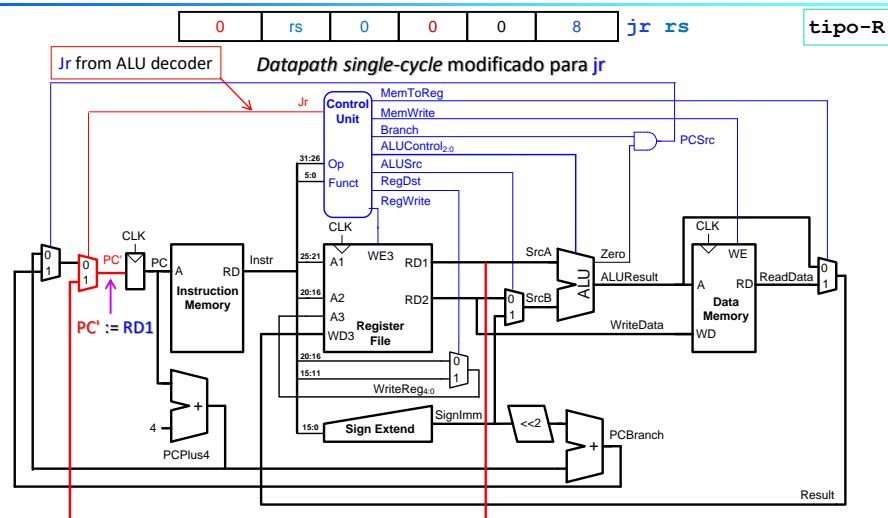
1. +1 linha para OpCode=jal
2. +1 saída = Jal
3. Jump: Esta saída tb está activa.
4. RegDst: +1 bit e valor=10 para  $\$ra$  (31)

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

13/23

## Problemas SC (9), P7.4c (1) - JR: Datapath



Modificações Datapath:

1. Mux Jr: RD1 -> PC'; 2. Ligar RD1 à entrada\_1 do Mux Jr.

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

14/23

## Problemas SC (10), P7.4c (2) - JR: ALU Decoder

Diagrama da instrução **jr rs** (tipo-R) com o sinal **Jr** gerado pelo ALU decoder.

> **jr** é uma instrução de tipo-R, com a particularidade de ambos **rt** e **rd** serem **zero**.

ALUOp <sub>1:0</sub>	Funct <sub>5:0</sub>	ALUControl <sub>2:0</sub>	Jr
00	X	010 (Add)	0
01	X	110 (Subtract)	0
10	100000 (add)	010 (Add)	0
10	100010 (sub)	110 (Subtract)	0
10	100100 (and)	000 (And)	0
10	100101 (or)	001 (Or)	0
10	100110 (xor)	100 (Xor)	0
10	100111 (nor)	101 (Nor)	0
10	101010 (slt)	111 (Slit)	0
10	001000 (jr)	XXX	1

Tabela de verdade do **ALU decoder** para **jr** ↑

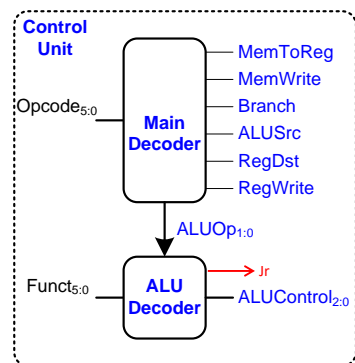
Modificação **ALU decoder**:

1. +1 saída = **Jr** para **Funct<sub>5:0</sub> = 001000**

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

15/23

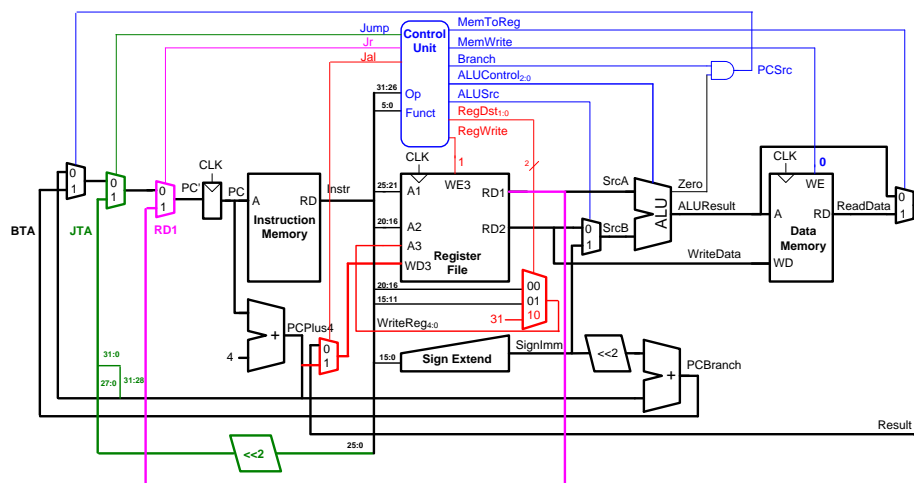


O sinal **Jr** é gerado no **ALU decoder**, porque é aí que o **Funct<sub>5:0</sub>** está ligado.

Tabela de verdade do **Main decoder** para **jr**: sem modificações!



### Problemas SC (11), P7.4a + P7.4c - JAL + JR: Datapath



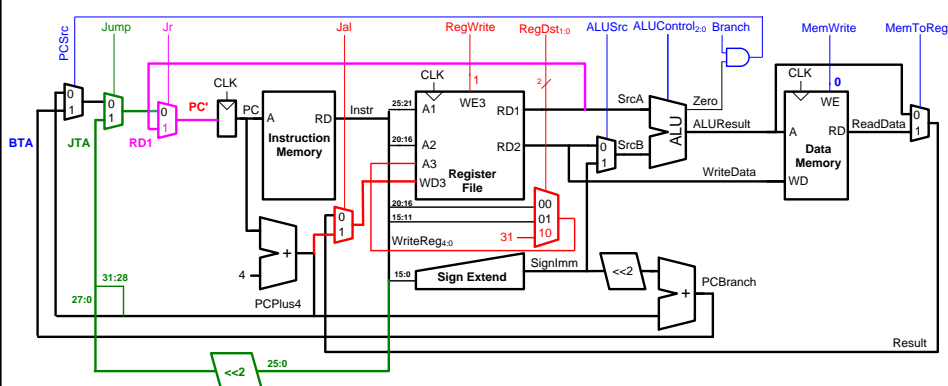
Datapath single-cycle modificado para jal + jr  
(jal precisa de jr)

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

16/23

### Problemas SC (12), P7.4a + P7.4c - JAL + JR: Datapath - v2



Datapath single-cycle modificado para jal + jr  
(jal precisa de jr)

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

17/23

13

rs

rt

imm

ori rt,rs,imm

tipo-I

ori é uma instrução do tipo-I. Precisamos de criar um novo código ALUOp? Não. Vamos reutilizar o código ALUOp=11 já usado para slti.

slti →

ori →

ALUOp <sub>1:0</sub>	OpCode <sub>5:0</sub>	Funct <sub>5:0</sub>	ALUControl <sub>2:0</sub>
00		X	010 (Add)
01		X	110 (Subtract)
10		100000 (add)	010 (Add)
10		100010 (sub)	110 (Subtract)
10		100100 (and)	000 (And)
10		100101 (or)	001 (Or)
10		100110 (xor)	100 (Xor)
10		101010 (slt)	111 (Slt)
11	001010	X	111 (Slt)
11	001101	X	001 (Or)

O ALU Decoder precisa agora também do OpCode para poder gerar o ALUControl qdo o ALUOp = 11 (por este passar a ser partilhado pelas instruções imediatas).

ALUOp<sub>1:0</sub>

Funct<sub>5:0</sub>

OpCode<sub>5:0</sub>

ALU Decoder

ALUControl<sub>2:0</sub>

O datapath precisa de ser modificado! O Main Decoder tb tem de ser adaptado.

Tabela de verdade do ALU decoder para ori

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

18/23

13

rs

rt

imm

ori rt,rs,imm

tipo-I

Tabela de verdade do Main decoder para tb suportar slti e ori.

ori é uma instrução lógica do tipo-I, por isso o valor imediato (im16) deve ser estendido com zeros, i.e., ignorando o bit de sinal. Assim, temos de adicionar uma nova entrada ao multiplexer AluSrc (como aliás já tínhamos feito para lui ).

Instruction	Op <sub>5:0</sub>	RegWrite	RegDst	AluSrc <sub>1:0</sub>	Branch	MemWrite	MemToReg	ALUOp <sub>1:0</sub>
R-type	000000	1	1	00	0	0	0	10
lw	100011	1	0	01	0	0	1	00
sw	101011	0	X	01	0	1	X	00
beq	000100	0	X	00	1	0	X	01
lui	001111	1	0	10	0	0	0	00
slti	001010	1	0	01	0	0	0	11
ori	001101	1	0	11	0	0	0	11

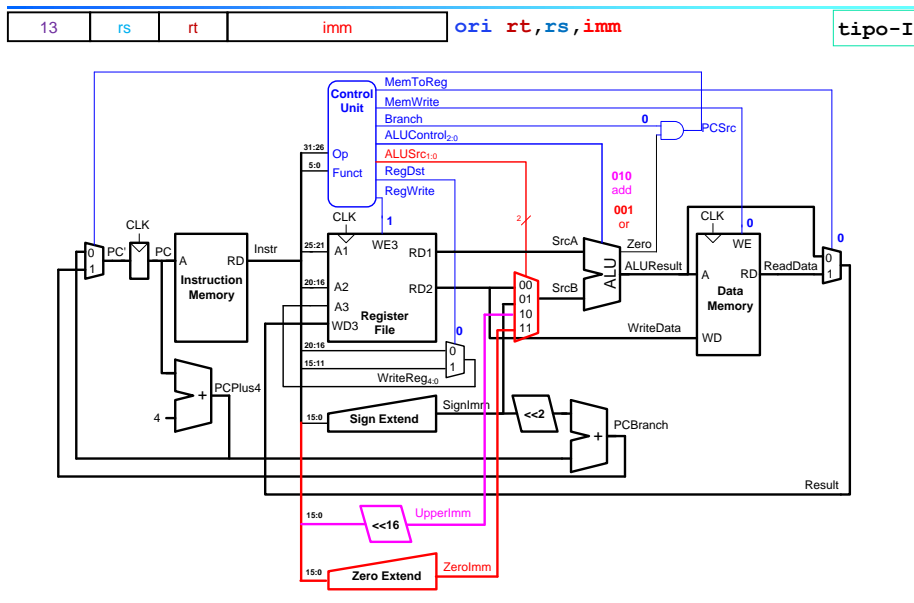
© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

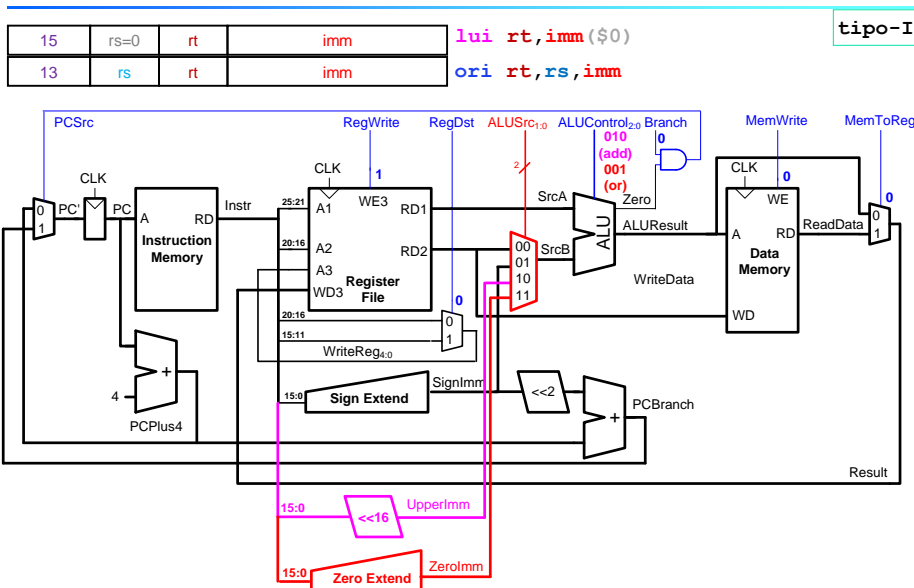
19/23

10

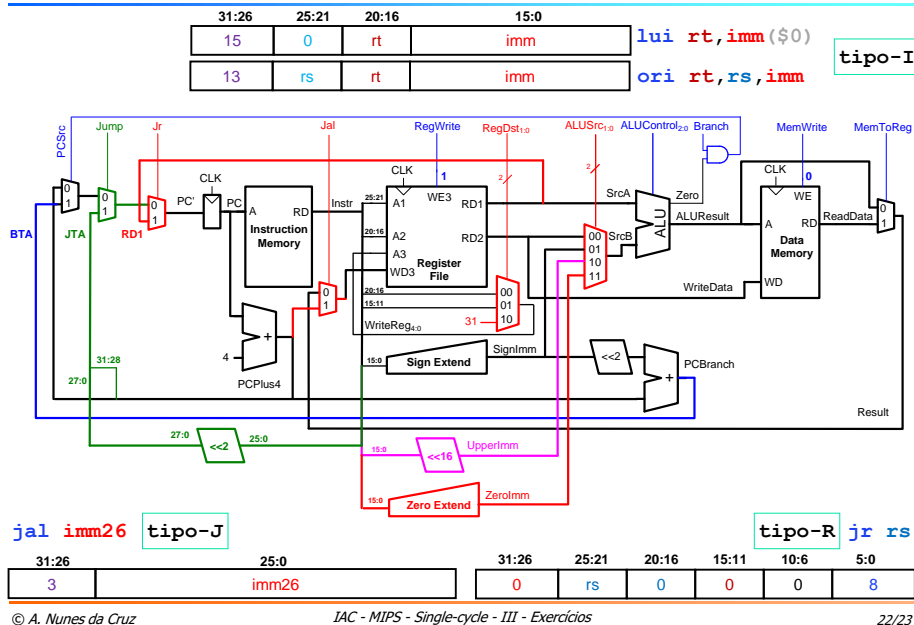
### Problemas SC (15), Extra (3) - LUI + ORI: Datapath



### Problemas SC (16), Extra (4) - LUI + ORI: Datapath - v2



## Problemas SC (17) - LUI + ORI & JAL + JR



## A seguir: Datapath Multicycle

