Introdução à Arquitetura de Computadores

Aula23

µArquitetura Single-cycle - III - Exercícios

Instruções adicionais

- sll - shift left logical
- lui - load upper immediate
- slti - set on less than immediate
- jal - jump and link
- jr - jump register
- ori - or immediate

A. Nunes da Cruz / **DETI - UA** Junho / 2021

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

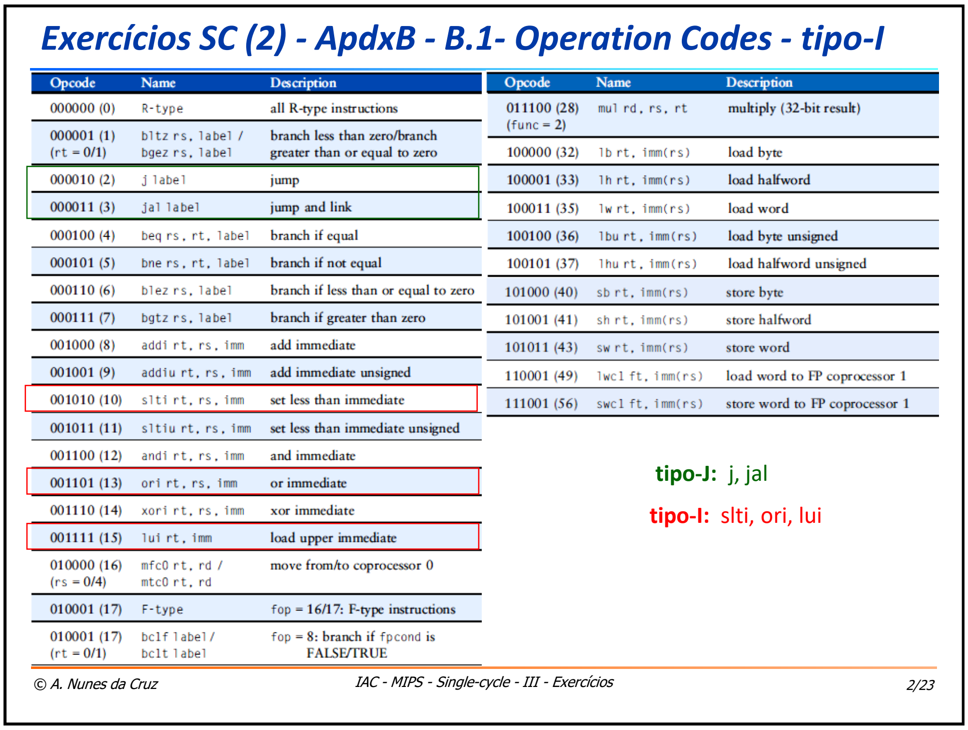
Modifique o CPU Single-cycle para adicionar suporte para uma das seguintes instruções:

(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)



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

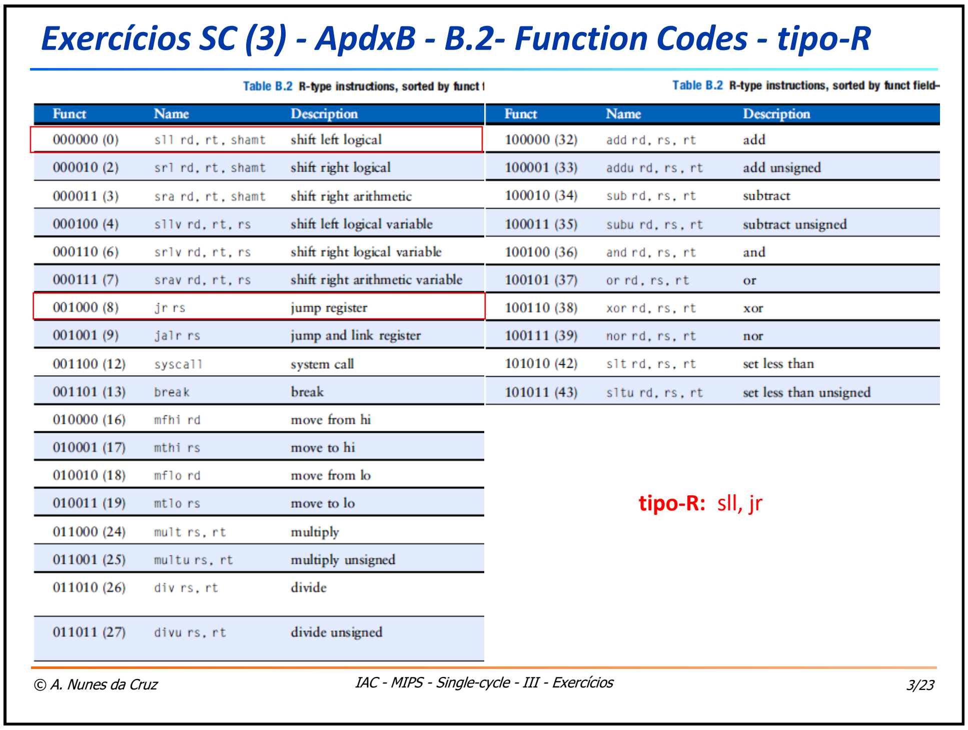
Opcode	Name	Description
000000 (0)	R-type	all R-type instructions
000001 (1) (rt = 0/1)	bltz rs, label / bgez rs, label	branch less than zero/branch greater than or equal to zero
000010 (2)	j label	jump
000011 (3)	jal label	jump and link
000100 (4)	beq rs, rt, label	branch if equal
000101 (5)	bne rs, rt, label	branch if not equal
000110 (6)	blez rs, label	branch if less than or equal to zero
000111 (7)	bgtz rs, label	branch if greater than zero
001000 (8)	addi rt, rs, imm	add immediate
001001 (9)	addiu rt, rs, imm	add immediate unsigned
001010 (10)	slti rt, rs, imm	set less than immediate
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) (rs = 0/4)	mfc0 rt, rd / mtc0 rt, rd	move from/to coprocessor 0
010001 (17)	F-type	fop = 16/17: F-type instructions
010001 (17) (rt = 0/1)	bc1f label / bc1t label	fop = 8: branch if fpcond is FALSE/TRUE

Opcode	Name	Description
011100 (28) (func = 2)	mul rd, rs, rt	multiply (32-bit result)
100000 (32)	lb rt, imm(rs)	load byte
100001 (33)	lh rt, imm(rs)	load halfword
100011 (35)	lw rt, imm(rs)	load word
100100 (36)	lbu rt, imm(rs)	load byte unsigned
100101 (37)	lhu rt, imm(rs)	load halfword unsigned
101000 (40)	sb rt, imm(rs)	store byte
101001 (41)	sh rt, imm(rs)	store halfword
101011 (43)	sw rt, imm(rs)	store word
110001 (49)	lwc1 ft, imm(rs)	load word to FP coprocessor 1
111001 (56)	swc1 ft, imm(rs)	store word to FP coprocessor 1

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 field

Funct	Name	Description
000000 (0)	sll rd, rt, shamt	shift left logical
000010 (2)	srl rd, rt, shamt	shift right logical
000011 (3)	sra rd, rt, shamt	shift right arithmetic
000100 (4)	sllv rd, rt, rs	shift left logical variable
000110 (6)	srlv rd, rt, rs	shift right logical variable
000111 (7)	srav rd, rt, rs	shift right arithmetic variable
001000 (8)	jr rs	jump register
001001 (9)	jalr rs	jump and link register
001100 (12)	syscall	system call
001101 (13)	break	break
010000 (16)	mfhi 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

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

Funct	Name	Description
100000 (32)	add rd, rs, rt	add
100001 (33)	addu rd, rs, rt	add unsigned
100010 (34)	sub rd, rs, rt	subtract
100011 (35)	subu rd, rs, rt	subtract unsigned
100100 (36)	and rd, rs, rt	and
100101 (37)	or rd, rs, rt	or
100110 (38)	xor rd, rs, rt	xor
100111 (39)	nor rd, rs, rt	nor
101010 (42)	slt rd, rs, rt	set less than
101011 (43)	sltu rd, rs, rt	set less than unsigned

tipo-R: sll, jr

© A. Nunes da Cruz — *IAC - MIPS - Single-cycle - III - Exercícios* — 3/23

Exercícios SC (4) - Figura 7.11 - Datapath

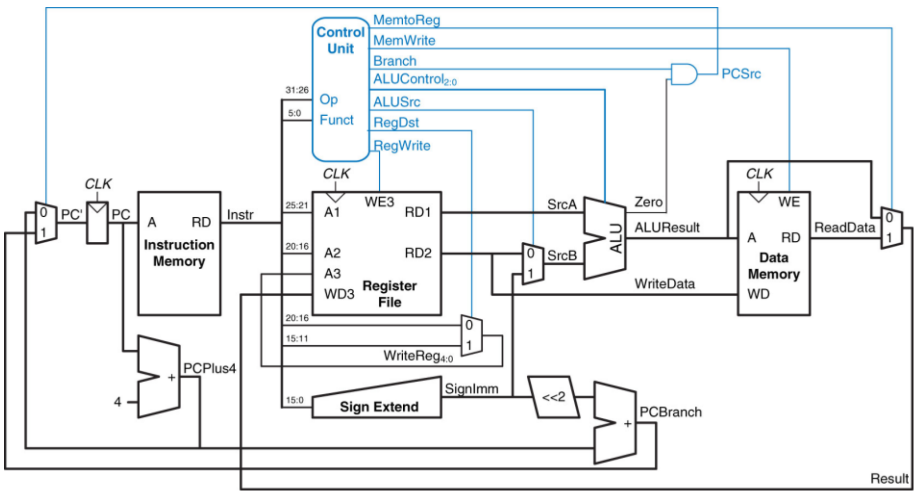
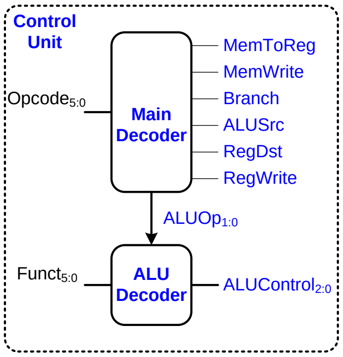


Figura 7.11 Processador MIPS *Single-cycle* completo

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



ALUOp$_{1:0}$	Funct$_{5:0}$	ALUControl$_{2:0}$
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$_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	ALUOp$_{1:0}$
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

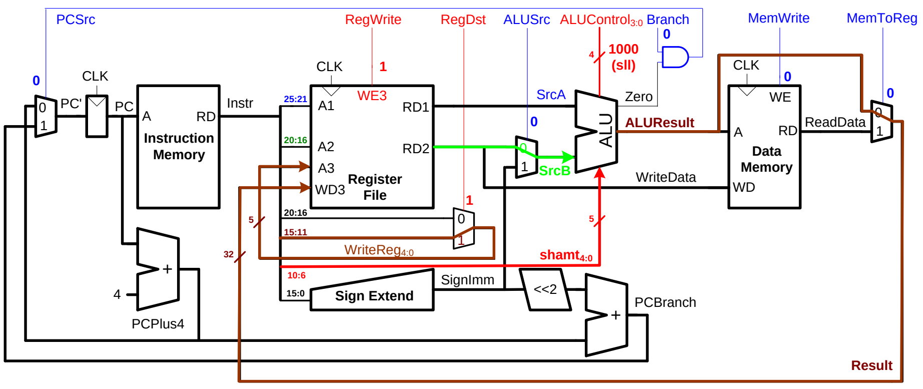
Tabela 7.3 Tabela de verdade *do Main decoder*

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

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`



Datapath single-cycle modificado para sll

Modificações:

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

`(rd)= (rt) << shamt`
`(rs=0,ignorado!)`

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`

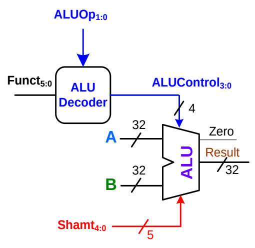
`tipo-R`

$ALUOp_{1:0}$	$Funct_{5:0}$	$ALUControl_{3:0}$
00	X	0010 (Add)
01	X	0110 (Subtract)
10	100000 (add)	0010 (Add)
10	100010 (sub)	0110 (Subtract)
10	100100 (and)	0000 (And)
10	100101 (or)	0001 (Or)
10	100110 (xor)	0100 (Xor)
10	100111 (nor)	0101 (Nor)
10	101010 (slt)	0111 (Slt)
10	000000 (sll)	1000 (Shift Left Logical)

Tabela de verdade do *ALU decoder* para sll

$ALUCtrl_{3:0}$	Função
0000	A & B
0001	A | B
0010	A + B
0011	not used
0100	A ^ B
0101	~(A | B)
0110	A - B
0111	Slt
1000	Sll
1001	Srl

ALU para sll



`SLL:`
`Result = B << 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*).

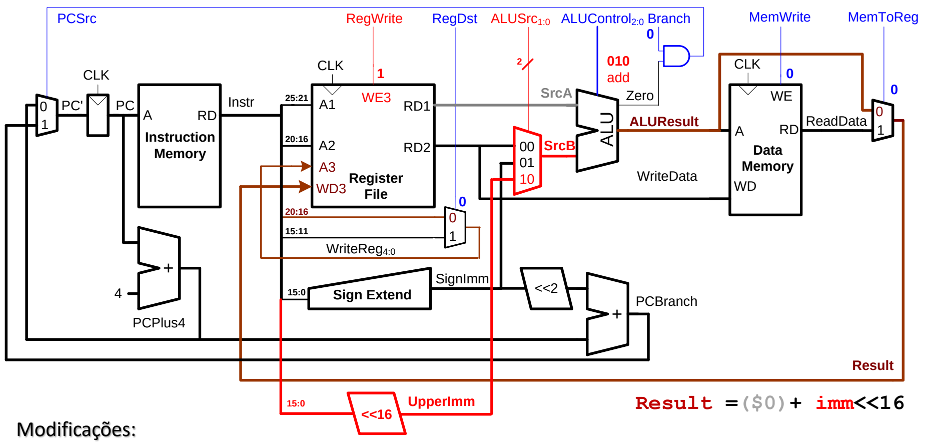
`(rd) = (rt) << shamt`
`(rs=0,ignorado!)`

Problemas SC (3), P7.3b (1) - LUI: Datapath

31:26	25:21	20:16	15:0
15	rs=0	rt	imm

`lui rt,imm($0)`

tipo-I



`Result = ($0) + imm<<16`

Modificações:

1. Imm << 16
2. Mux AluSrc: 3 inputs

Datapath single-cycle modificado para lui

`(rt):= imm<<16`

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

Instruction	$Op_{5:0}$	RegWrite	RegDst	$AluSrc_{1:0}$	Branch	MemWrite	MemToReg	$ALUOp_{1:0}$
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

$AluSrc_{1:0}$ = 10 ➔
ScrB = UpperImm

$ALUOp_{1:0}$ = 00 ➔
$AluCtrl_{2:0}$ = 010 (add)

Tabela de verdade do *ALU decoder* para lui

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

10	rs	rt	imm

slti rt,rs,imm

tipo-I

slti 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.

$ALUOp_{1:0}$	$Funct_{5:0}$	$ALUControl_{2:0}$
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`

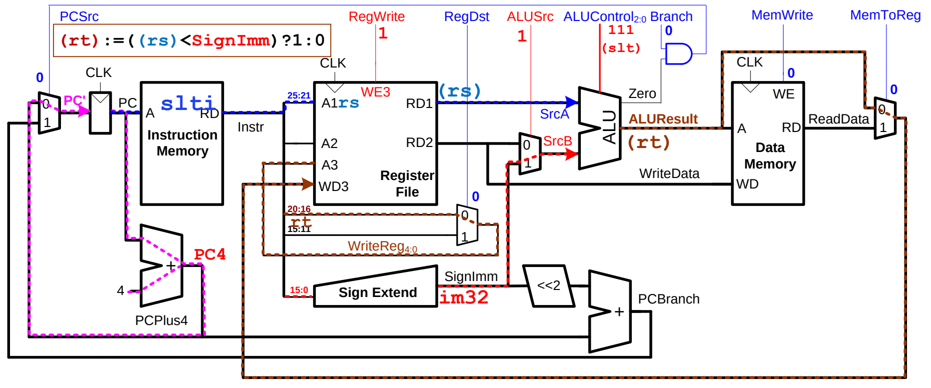
Tabela de verdade do ALU decoder para slti

Instruction	$OpCode_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	$ALUOp_{1:0}$
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
slti	001010	1	0	1	0	0	0	11

Tabela de verdade do *Main decoder* para slti

Problemas SC (6), P7.3c (2) - STLI: Datapath slti rt,rs,imm

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	$ALUOp_{1:0}$
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
slti	001010	1	0	1	0	0	0	11

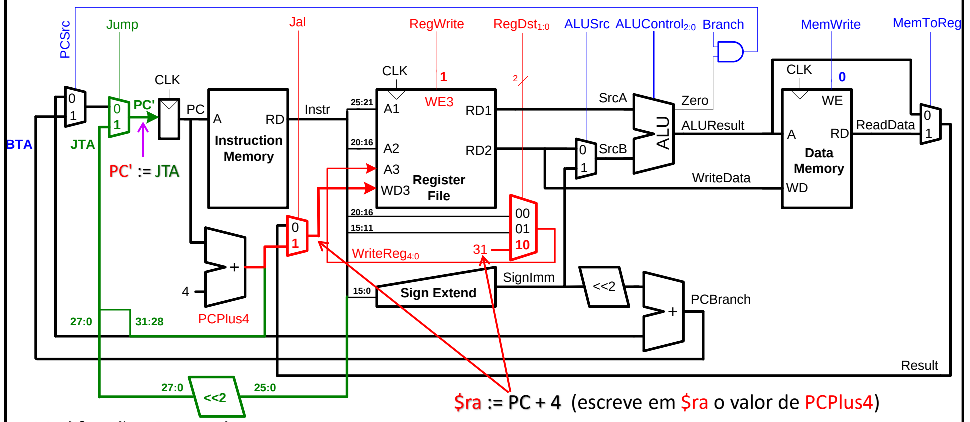


Problemas SC (7), P7.4a (1) - JAL: Datapath

	31:26	25:0	
jal label	3	imm26	jal imm26

tipo-J

Datapath single-cycle modificado para jal



$ra := PC + 4 (escreve em $ra o valor de PCPlus4)

Modificações *Datapath*:

1. RegDst: +1 bit e +entrada 31
2. Mux Jal: PC4 ->RF

jal label:
1. PC' := JTA = $(PC + 4)_{31:28}$: ($Imm_{25:0}$ <<2) ; 2. $ra (31) := PC + 4

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

Instruction	$Op_{5:0}$	RegWrite	$RegDst_{1:0}$	AluSrc	Branch	MemWrite	MemToReg	$ALUOp_{1:0}$	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

$\uparrow$ $ra $\uparrow$ JTA $\uparrow$ PC4

Tabela de Verdade do *Main decoder* para jal

Modificações *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)

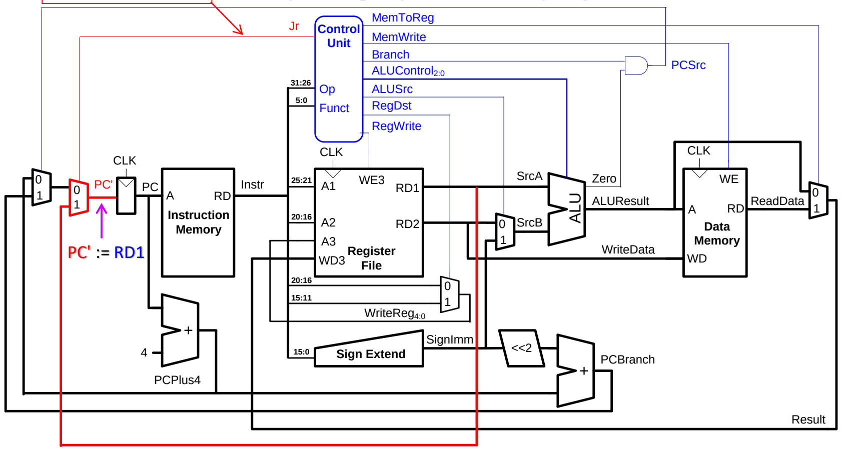
Problemas SC (9), P7.4c (1) - JR: Datapath

0	rs	0	0	0	8	jr rs

tipo-R

Jr from ALU decoder

Datapath single-cycle modificado para jr



Modificações *Datapath*:

1. Mux Jr: RD1 ->PC'; 2. Ligar RD1 à entrada_1 do Mux Jr.

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

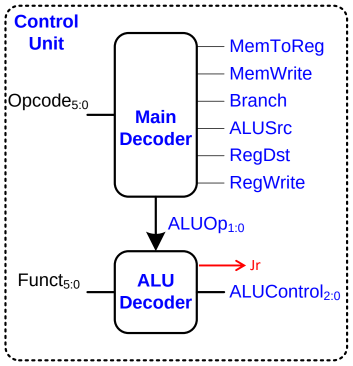
0	rs	0	0	0	8	jr rs

tipo-R

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

$ALUOp_{1:0}$	$Funct_{5:0}$	$ALUControl_{2:0}$	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 (Slt)	0
10	001000 (jr)	XXX	1

Tabela de verdade do *ALU decoder* para jr ↑



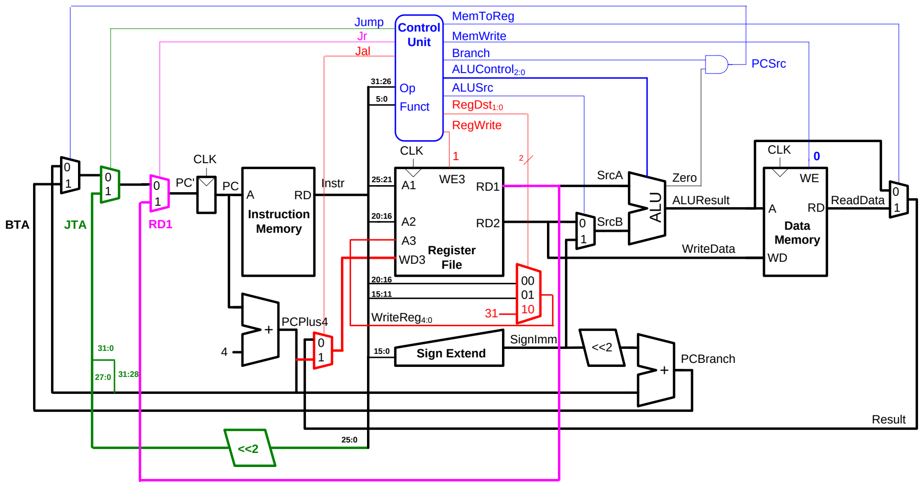
O sinal Jr é gerado no *ALU decoder*, porque é aí que o $Funct_{5:0}$ está ligado.

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

Modificação *ALU decoder*:

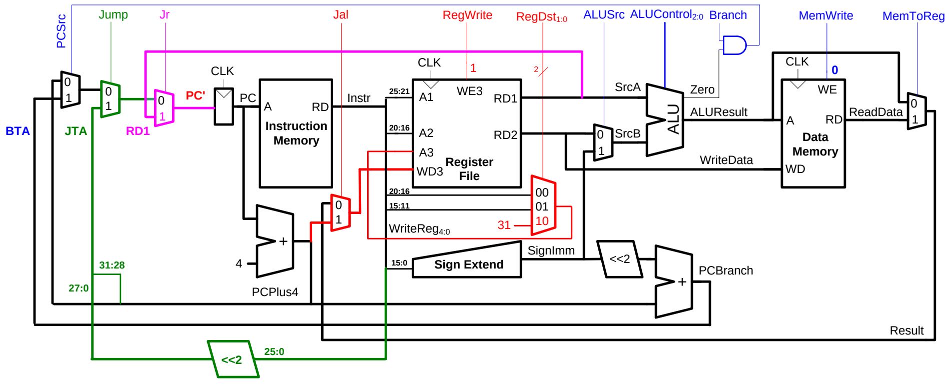
1. +1 saída = Jr para $Funct_{5:0}$ = 001000

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



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

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



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

Problemas SC (13), Extra (1) - ORI: ALU Decoder

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.

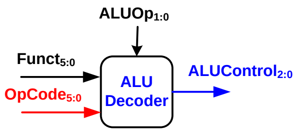
Vamos reutilizar o código ALUOp=11 já usado para slti.

	ALUOp$_{1:0}$	OpCode$_{5:0}$	Funct$_{5:0}$	ALUControl$_{2:0}$
	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)
slti $\longrightarrow$	11	001010	X	111 (Slt)
ori $\longrightarrow$	11	001101	X	001 (Or)

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

Tabela de verdade do ALU decoder para ori $\longleftarrow$

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).



Problemas SC (14), Extra (2) - ORI: Main Decoder

13	rs	rt	imm

`ori rt,rs,imm` tipo-I

Instruction	Op$_{5:0}$	RegWrite	RegDst	AluSrc$_{1:0}$	Branch	MemWrite	MemToReg	ALUOp$_{1:0}$
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

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

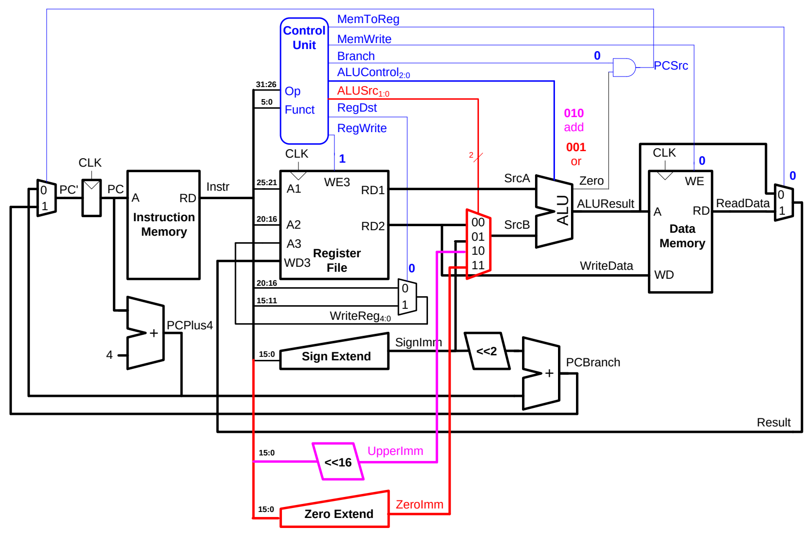
ori é uma instrução lógica do **tipo-I**, por isso o valor imediato (im16) deve ser estendido com zeros, i.e., ignorando o bit de sinal. Assim, temos de adicionar uma nova entrada ao multiplexer AluSrc (como aliás já tinhamos feito para lui).

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

13	rs	rt	imm

`ori rt,rs,imm`

tipo-I



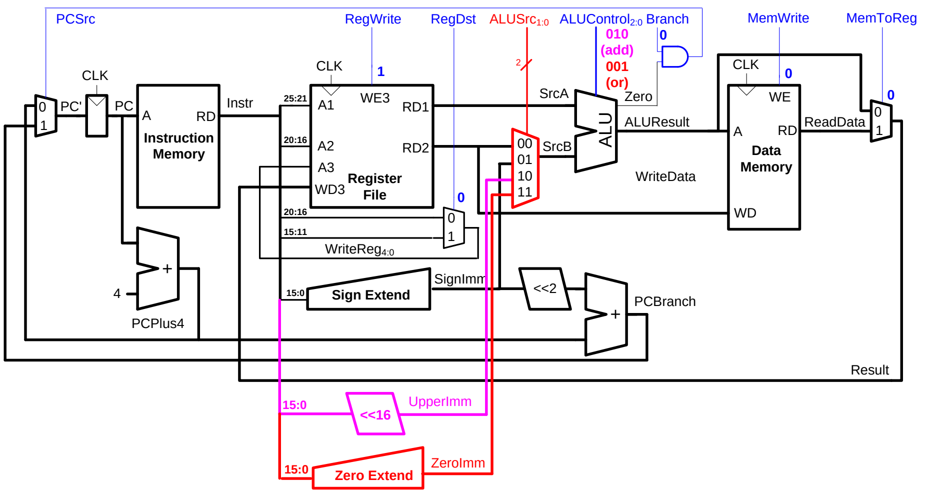
Problemas SC (16), Extra (4) - LUI + ORI: Datapath - v2

15	rs=0	rt	imm
13	rs	rt	imm

`lui rt,imm($0)`

`ori rt,rs,imm`

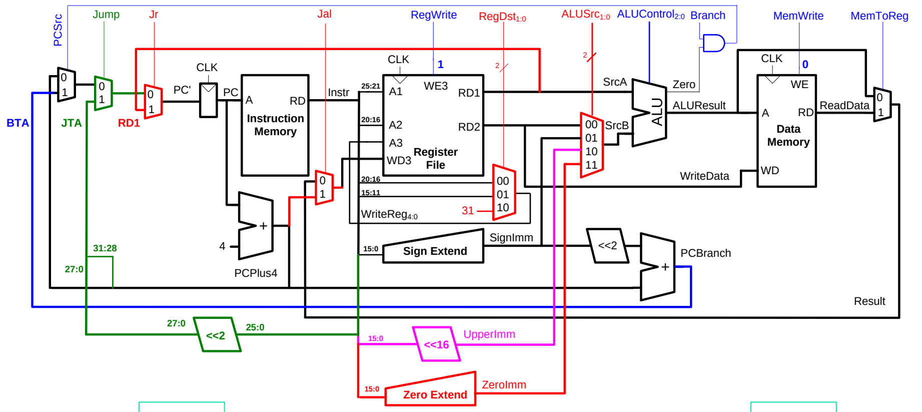
tipo-I



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

31:26	25:21	20:16	15:0	
15	0	rt	imm	lui rt,imm($0)
13	rs	rt	imm	ori rt,rs,imm

tipo-I



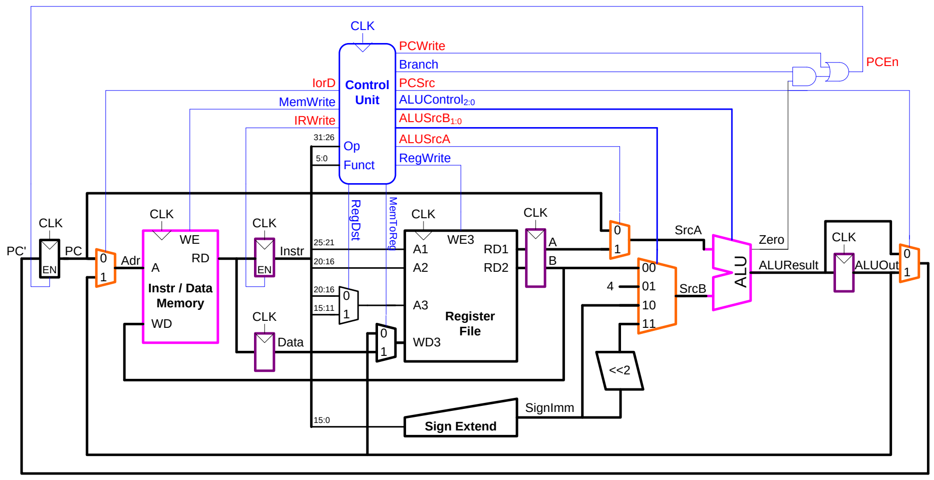
jal imm26 tipo-J

31:26	25:0
3	imm26

tipo-R jr rs

31:26	25:21	20:16	15:11	10:6	5:0
0	rs	0	0	0	8

A seguir: Datapath Multicycle



* Uma única Memória (*von Neumann*) e uma única ALU

* Unidade de Controlo mais complexa: FSM.