

Introdução à Arquitetura de Computadores

Aula 23

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

! Começar a res postimine na página 7 e depois voltar aqui!

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
ori (extra)

Exercício 7.4

Repita o Exercício 7.3 para as seguintes instruções:

- (a) jal
- (b) lh
- (c) jr → R-Type
Escrita jazera:
 $PC = RS$ ou $PC = RT$
- (d) srl
- ori (extra) (ver dicas ao final)

a) sll → Instrução do tipo -R → Tem que se aplicar a linha 1

Mnemônica da instrução: shift amount

sll RD, RT, shamt

Registro RT foi shiftado!

shift amount
da tabela
da página 5

(ver à frente)

E suposto no datapath?

A ALU é que vai fazer o shift!

Hipótesa? Quase!

→ Falta colocar o shamt à entrada da ALU!



Bits 10 a 6 na ALU

lui → ? Tipo -I Load Upper immediate ?

lui RT, imm16

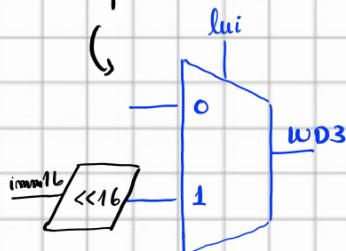


? Constante de 16 → Sign Extend

Termos que colocam a constante shiftada à esquerda de 16 bits no valor que vai ser menor (WD3)

Como suposta? → A resposta não é única!

Uma hipótese seria esta:



≠ Da solução proposta nos slides!

Simple!

e) slti (Ver em casa)

d) ori

? Or immediate e add immediate são quase iguais!

addi RT, RS, imm16 → A constante é sign-extended!

ORI RT, RS, imm16 → zero-extended!

Datapath para supor esta instrução:

→ ? É preciso que a constante seja de 16 bits → Termo que passa por um zero extended!

Exercícios SC (4) - Figura 7.11 - Datapath

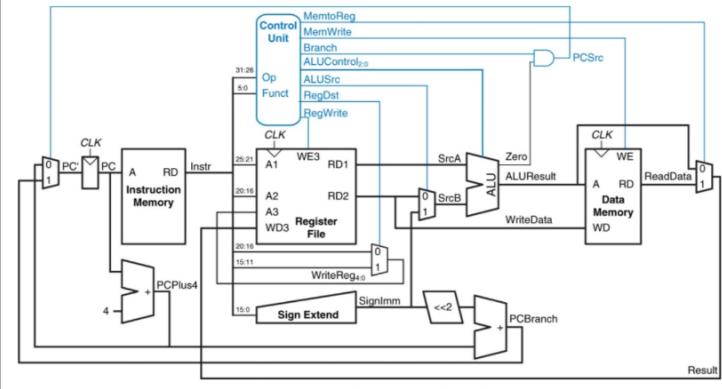


Figura 7.11 Processador MIPS Single-cycle completo

(Ver em baixo)

© A. Nunes da Cruz

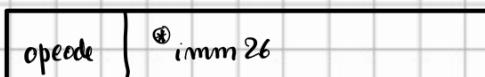
IAC - MIPS - Single-cycle - III - Exercícios

4/23

e) jal target
jump
• O que faz?

Tem como destino um target!

Vimos antes (ver pág. 7 destes slides):
j target



$$PC = JTA = (PC + 4) \oplus (imm_{26} \ll 2)$$

↑ shiftam 2 casas à esquerda
Concatenação

Para isso temos que: (Ver aula 22)

Parte do Jump

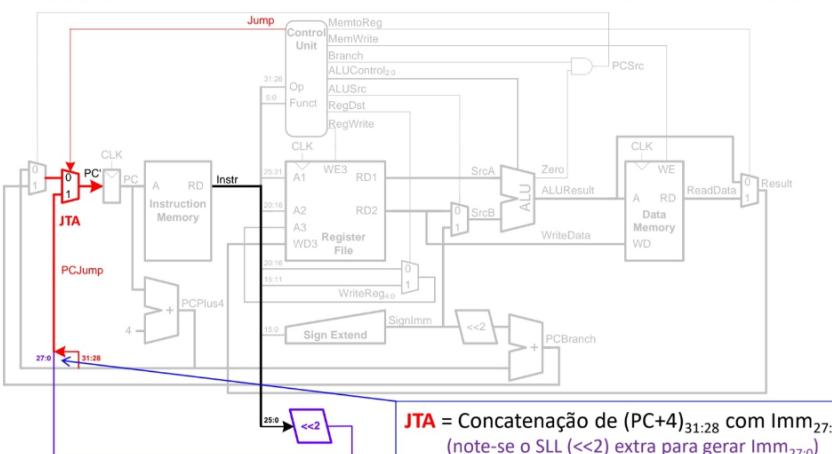
! se fosse "diretamente" tinha que ter aqui os 32 bits!

! Para suportar o jump temos que colocar um sinal novo (na tabela):
↓
jump!

P: Quais são as modificações a fazer, tanto ao datapath como ao Main Decoder, para suportar a instrução j?

Mais Instruções (7) - j (2) - Datapath

R-1: Alteração ao Datapath: adicionamos hardware para calcular o valor do PC'. Isto pode ser conseguido através de mais um **multiplexer**, controlado por um novo sinal **Jump** (proveniente do **Main Decoder**), a cuja entrada_1 ligamos **PCJump** (**JTA**).



- ! Os writers nunca podem ser "Don't care"! → RegWrite = 0, MemWrite = 0
! Não pode causar alterações nos registos mem e nem escrever nada na memória!
! Na operação jump → A ALU não está a fazer nada de útil → ALUSrc = X
Branch = X;
! Todo o resto pode ser!

Parte do "and link"

- Registo de destino para a ser 1 de 3!
- Write Data: Se a instrução for jal → WD3 vai ser PCPLUS4

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	multiply (32-bit result)
000001 (1)	bltz rs, label / (rt = 0/1) 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)	mfco rt, rd / (rs = 0/4) mtco 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 fpcond 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 funct1			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)	sr1 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)	sriv 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)	mfhi rd	move from hi			
010001 (17)	mtfi 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)	multur 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

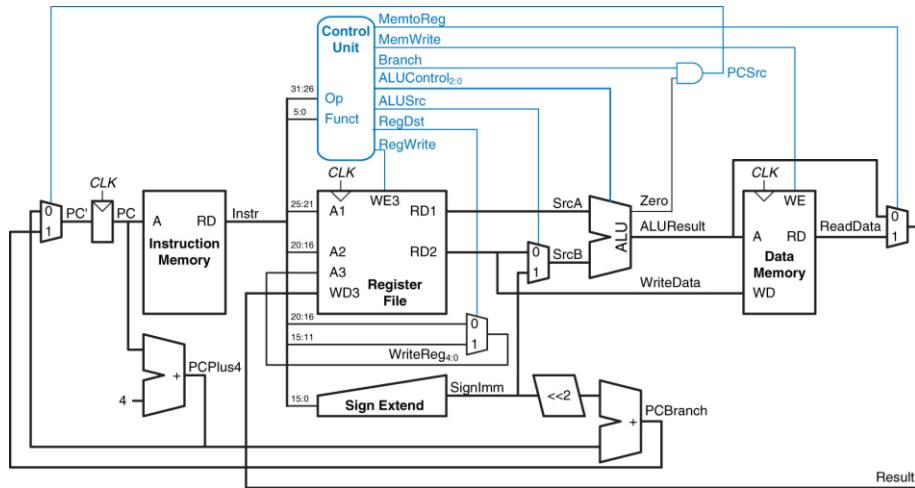


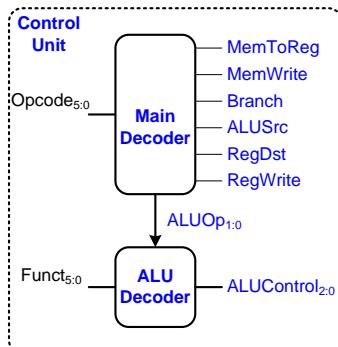
Figura 7.11 Processador MIPS Single-cycle completo (Ver em baixo)

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

4/23

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*

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

5/23

Atenção: Existem instruções não suportadas apesar de se chamar completo
Ex: jump não é suportado

Professor fez no quadro:

00 = ADD
01 = SUB
10 = R-Type

Instruções	jump	RegWrite	Reg Dst	ALUOp	Branch	MemWrite	MemToReg	ALUOp
R-Type	0	1	1	0	0	0	0	10
lw	0	1	0	1	0	0	1	00 - ADD
sw	0	0	X	1	0	1	X	00
beq	0	0	X	0	1	0	X	01 - SUB
addi	0	1	0	1	0	0	0	00
	jump	1	0	X	X	0	X	X

R-Type:

add RD, RS, RT

20 16 15 11

opcode	RS	RT	RD	SHAMT	FUNCT
000 000					

$$RD = RS + RT$$

Registo Destino \rightarrow RegWrite: Tem que ter valor 1 para escrever o valor RD

O resultado da ALU vai ser escrito como WD (unità de dados) no banco de registos

$$\text{RegWrite} = 1!$$

MemUnit \rightarrow R-Type Não existe escrita na memória \Rightarrow Fica a 0!

Branch \rightarrow Nenhum instruções de salto condicional \Rightarrow Branch = 0!

Load word

lw RT, imm(RS) \Rightarrow Tipo I

opcode	RS	RT	imm 16
000 000			

ALU \rightarrow vai calcular o endereço da memória

Operando B \rightarrow vai ser o imm16 depois de sign extended \rightarrow ALU SRE = 1

\Rightarrow N é um salto \rightarrow Branch = 0; ALU vai somar (p/obter o endereço que vai ser RS + imm16) \Rightarrow ALUOp = 00 - ADD

MemToReg \rightarrow Tem que ter valor 1

RegWrite \rightarrow Load word faz RT igual a imm(RS) \rightarrow Escrita no Banco de Registros

SW

\Rightarrow N se escreve nos registos \rightarrow RegWrite = 0

Então não importa onde se escreve \Rightarrow RegDst = X

mem o que se escreve \Rightarrow MemToReg = X

\Rightarrow N se salta \rightarrow Branch = 0

\Rightarrow Escreve-se na memória \rightarrow MemUnit = 1

beq RS, RT, imm16

1ª tança \rightarrow BTA = (PE+4) + imm16 \ll 2 \Rightarrow Esta tança já está feita no datapath!

2ª tança \rightarrow Tem que saber se é para saltar ou não!

Salta se: RS for igual a RT

Quem decide se são iguais?

↓
ALU fazendo uma subtração

Se der 0, sair que são iguais

\Rightarrow N a alterna nem um registo nem se escreve na memória \rightarrow RegWrite = 0; MemUnit = 0

Então não importa onde se escreve \Rightarrow RegDst = X

mem o que se escreve \Rightarrow MemToReg = X

Se quisermos suportar norte datapath a instrução addi o que fazemos?

addi RT, RS, imm 16

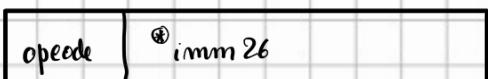
RT toma o valor de RS com a constante

? Vai haver escrita no registo \Rightarrow Tenho que dizer qual é (já não é "X"!)

ALUSrc (recebe o op.B da ALU): Tem que ter valor 1 para o SrcB ser a constante.

Se fosse subi \rightarrow Apenas alterava ALUOp! \rightarrow 01

j target



? Se fosse "diretamente" tinha que ter aqui 32 bits!

$$PC = JTA = (PC + 4); (imm26 \ll 2)$$

Concatenação

shiften 2 casas à esquerda

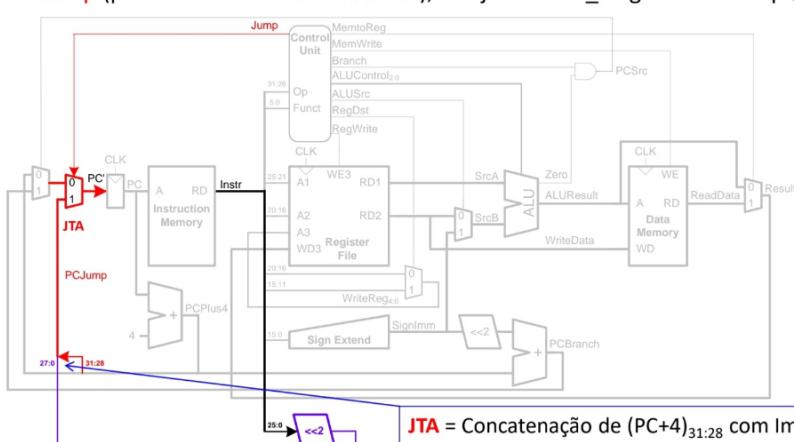
Para isso temos que: (Ver aula 22)

P: Quais são as modificações a fazer, tanto ao datapath como ao Main Decoder, para suportar a instrução j?

Para suportar o jump temos que colocar um sinal novo (na tabela):
jump!

Mais Instruções (7) - j (2) - Datapath

R-1: Alteração ao Datapath: adicionamos hardware para calcular o valor do PC. Isto pode ser conseguido através de mais um multiplexer, controlado por um novo sinal Jump (proveniente do Main Decoder), a cuja entrada_1 ligamos PCJump (JTA).



Sumariza uma coisa no processador

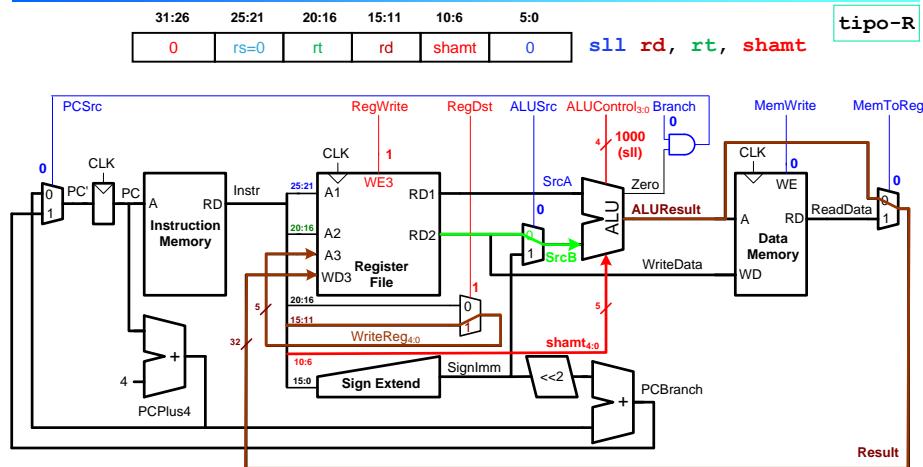
Os Writestrue nunca podem ser "Don't care"! \rightarrow RegWrite = 0, MemWrite = 0

? Não pode causar alterações nos registo nem escrever nada na memória!

? Na operação jump \rightarrow A ALU não está a fazer nada útil \rightarrow ALUSrc = X
Branch = X;

Tudo o resto pode ser!

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



Modificações:

Datapath single-cycle modificado para sll

1. ALU: tem uma entrada extra, $\text{shamt}_{4:0}$; implementa sll.
 2. $\text{ALUControl}_{3:0}$ tem 4 bits

(rd) = (rt) << 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

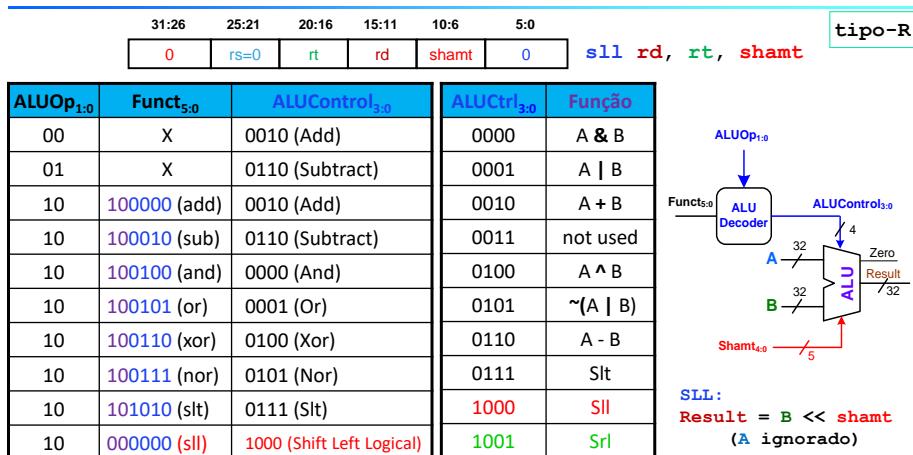


Tabela de verdade do *ALU decoder* para *SLI*

ALU para s/

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)

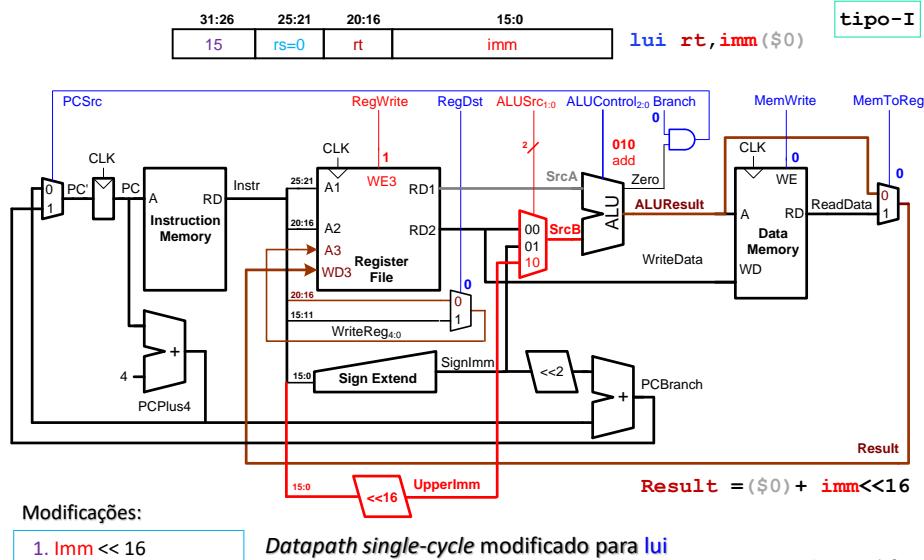
(rd) = (rt) << shamt
(rs=0, ignorado!)

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

7/23

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



Modificações:

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

Datapath single-cycle modificado para lui

(rt) := imm<<16

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

8/23

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

$$\text{AluSrc}_{1:0} = 10 \rightarrow \\ \text{ScrB} = \text{UpperImm}$$

$$\begin{aligned} \text{ALUOp}_{1:0} &= 00 \rightarrow \\ \text{AluCtrl}_{2:0} &= 010 (\text{add}) \end{aligned}$$

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	slti rt, rs, imm			tipo-I
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)					

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.

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 ←

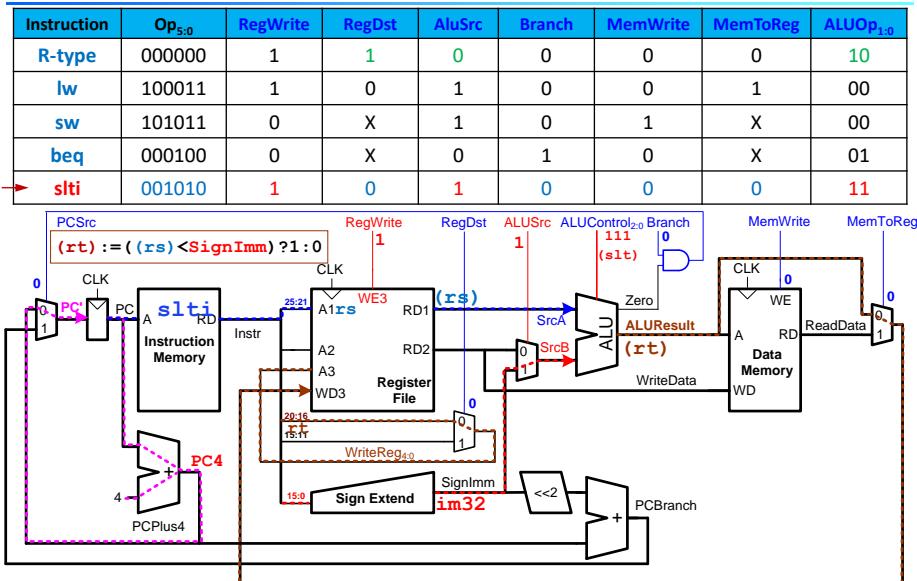
Tabela de verdade do Main decoder para slti

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

10/23

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

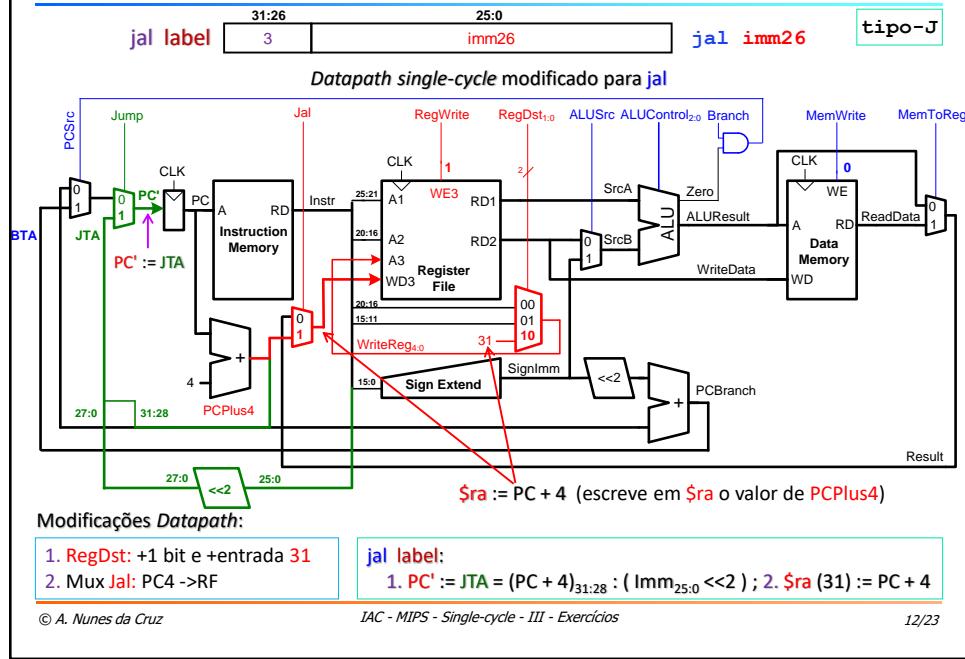


© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

11/23

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



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

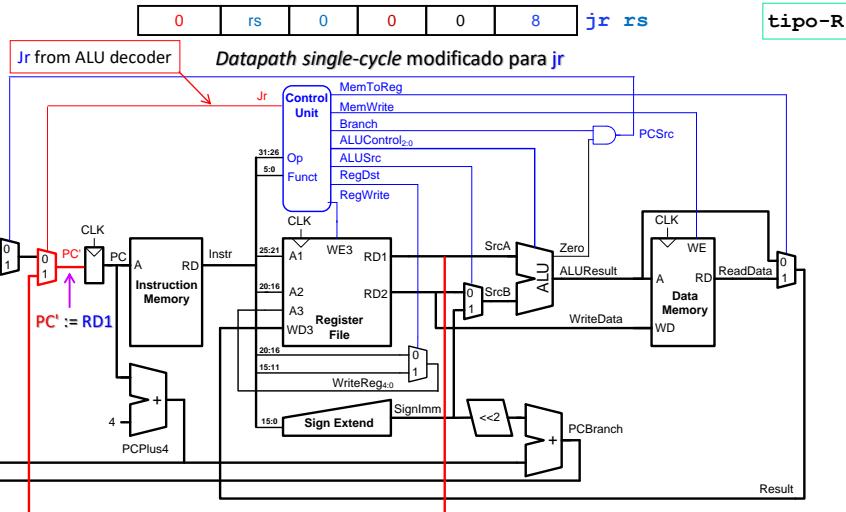
Instruction	Op _{5:0}	RegWrite	RegDst _{1:0}	AluSrc	Branch	MemWrite	MemToReg	ALUOp _{1:0}	Jump	Jal
R-type	000000	1	01	0	0	0	0	10	0	0
lw	100011	1	00	1	0	0	1	00	0	0
sw	101011	0	XX	1	0	1	X	00	0	0
beq	000100	0	XX	0	1	0	X	01	0	0
addi	001000	1	00	1	0	0	0	00	0	0
j	000010	0	XX	X	X	0	X	XX	1	0
jal	000011	1	10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

Tabela de Verdade do *Main decoder* para jai

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



Modificações Datapath:

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

© A. Nunes da Cruz

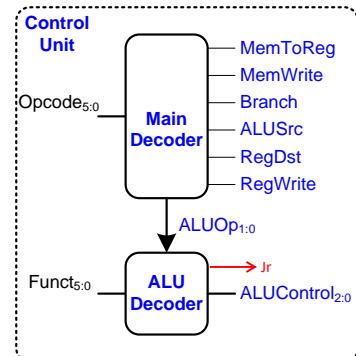
IAC - MIPS - Single-cycle - III - Exercícios

14/23

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



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

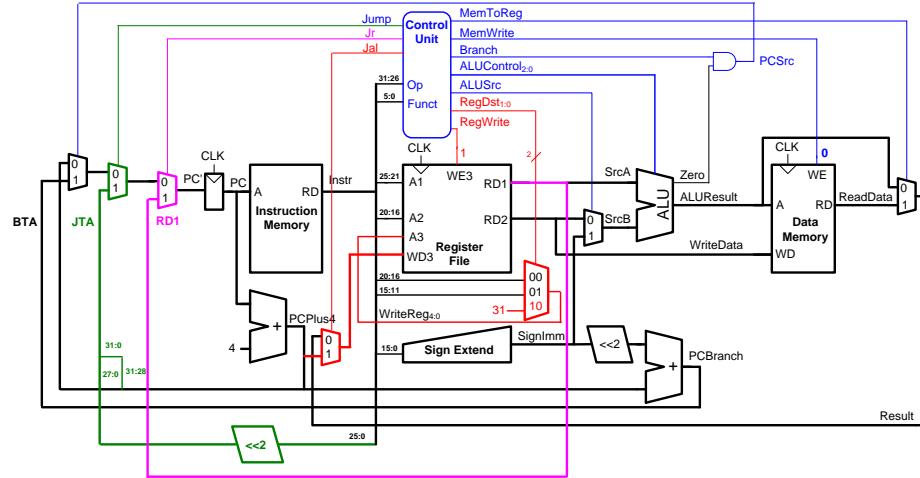
1. +1 saída = Jr para $Funct_{5:0} = 001000$

© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

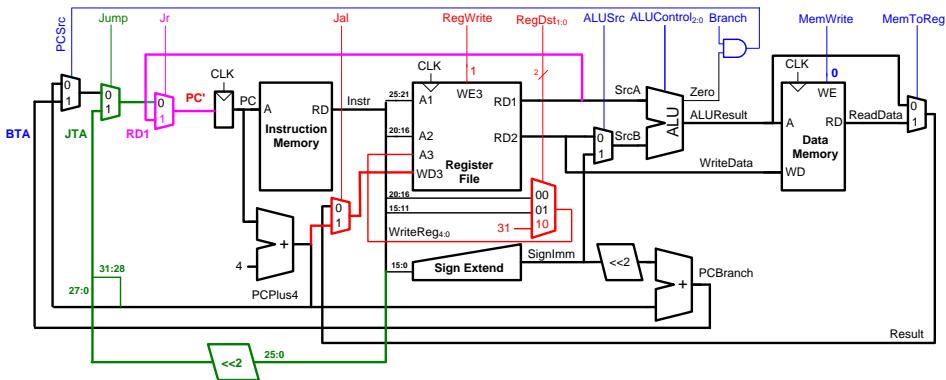
15/23

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
----	----	----	-----	-----------------	--------

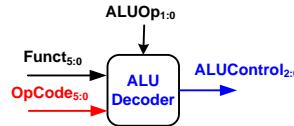
ALUOp _{1:0}	OpCode _{5:0}	Funct _{5:0}	ALUControl _{2:0}
00		X	010 (Add)
01		X	110 (Subtract)
10		100000 (add)	010 (Add)
10		100010 (sub)	110 (Subtract)
10		100100 (and)	000 (And)
10		100101 (or)	001 (Or)
10		100110 (xor)	100 (Xor)
10		101010 (slt)	111 (Slt)
11	001010	X	111 (Slt)
11	001101	X	001 (Or)

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 →

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

Tabela de verdade do ALU decoder para ori



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
----	----	----	-----	-----------------	--------

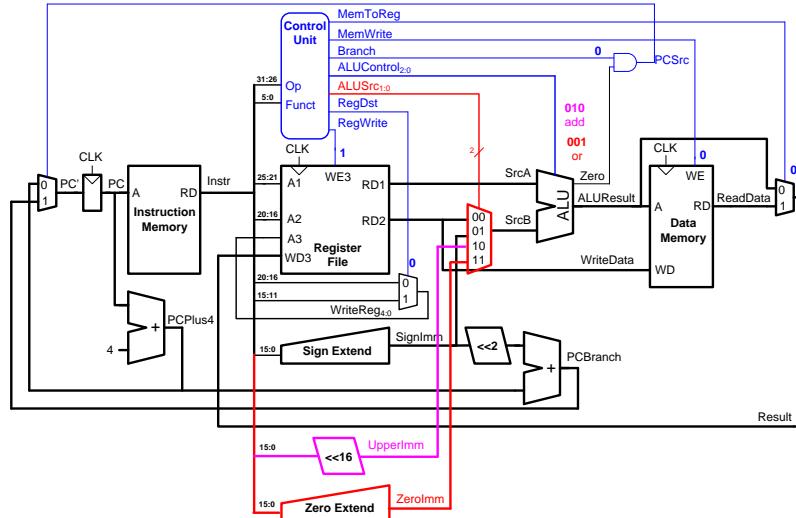
Instruction	Op _{5:0}	RegWrite	RegDst	AluSrc _{1:0}	Branch	MemWrite	MemToReg	ALUOp _{1:0}
R-type	000000	1	1	00	0	0	0	10
lw	100011	1	0	01	0	0	1	00
sw	101011	0	X	01	0	1	X	00
beq	000100	0	X	00	1	0	X	01
lui	001111	1	0	10	0	0	0	00
slti	001010	1	0	01	0	0	0	11
ori	001101	1	0	11	0	0	0	11

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 tínhamos feito para lui).

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

13 rs rt imm **ori rt,rs,imm** tipo-I



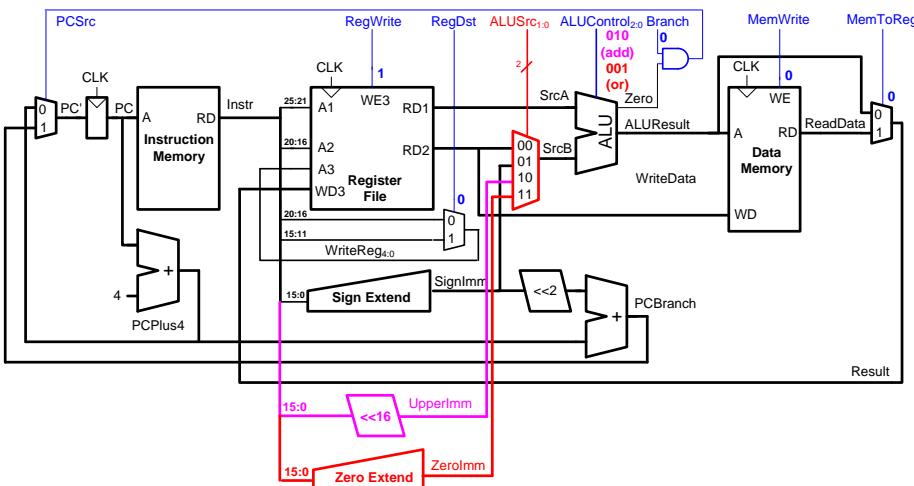
© A. Nunes da Cruz

IAC - MIPS - Single-cycle - III - Exercícios

20/23

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

15 rs=0 rt imm **lui rt,imm(\$0)** tipo-I
13 rs rt imm **ori rt,rs,imm**

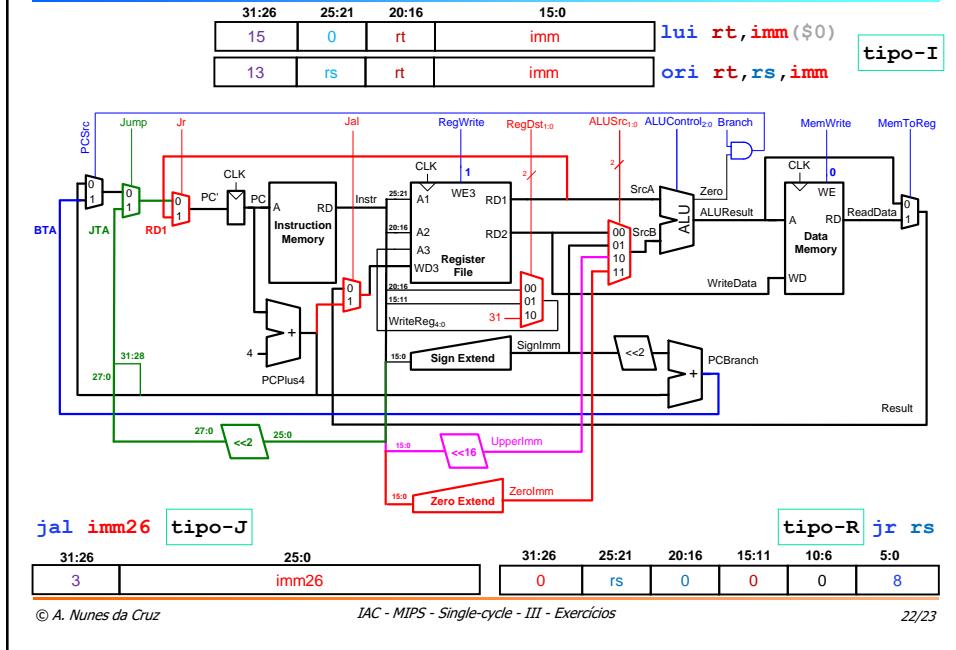


© A. Nunes da Cruz

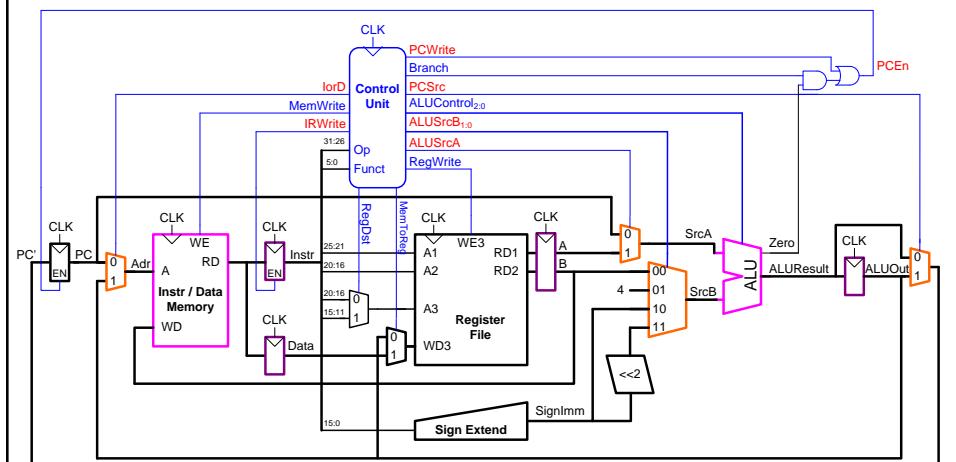
IAC - MIPS - Single-cycle - III - Exercícios

21/23

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



A seguir: Datapath Multicycle



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

* Unidade de Controlo mais complexa: FSM.