

Introdução à Arquitetura de Computadores

Aula26

μArquitetura MIPS Multicycle: II

Unidade de Controlo MC

Controlador Principal (FSM)

- Entradas e Saídas
 - Máquina de Estados
 - *Fetch e Decode*
 - Execução de Instruções do tipo *lw* e *sw*
Cálculo do endereço de memória
 - Execução de Instruções do tipo-R
 - Execução da Instrução *beq*

Mais Instruções

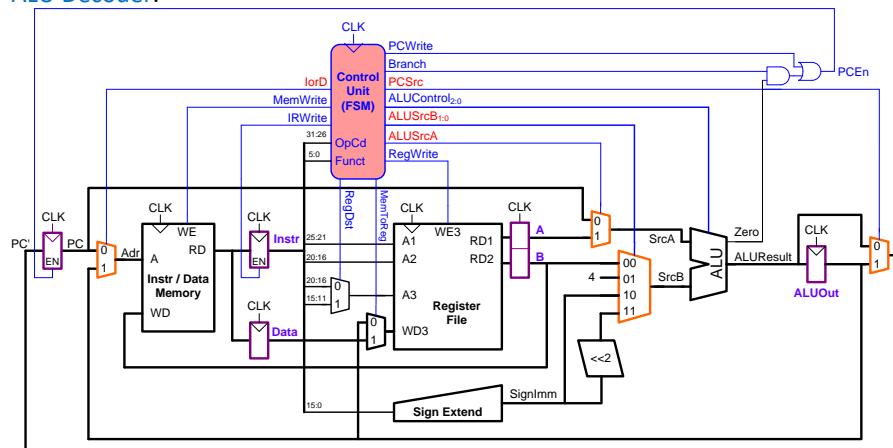
addi e j

A. Nunes da Cruz / DETI - UA

Junho / 2021

Datapath MultiCycle - Unidade de Controlo MC

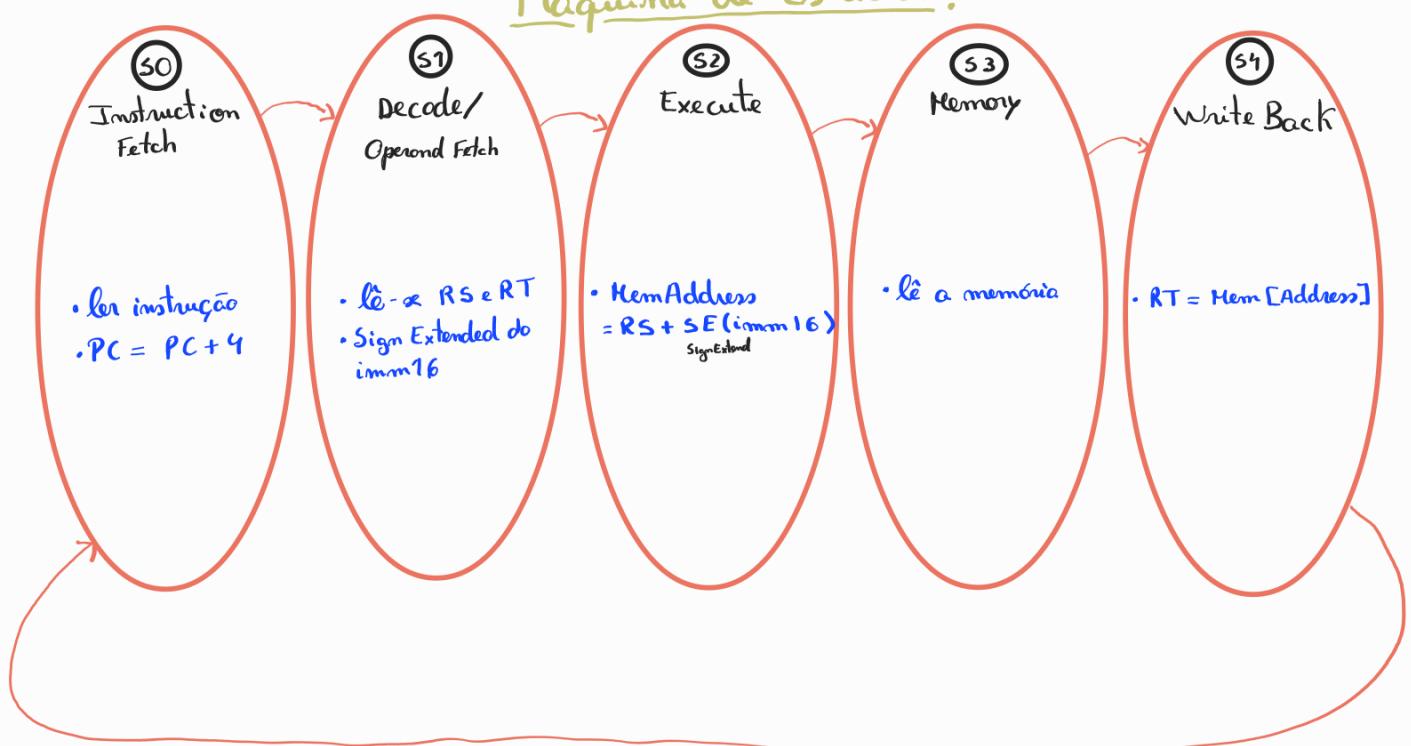
A Unidade de Controlo MC é constituída por um *Controlador Principal* e por um *ALU Decoder*.



O *Controlador Principal* é uma máquina síncrona (*FSM*) do tipo Moore responsável pela geração dos sinais de controlo do *datapath multicycle*.

! → Apenas para a instrução LW

Maquina de Estados!



Tipo R:

Opcode	RS	RT	RD	Shamt	Funct
--------	----	----	----	-------	-------

Tipo I:

Opcode	RS	RT	imm16		
--------	----	----	-------	--	--

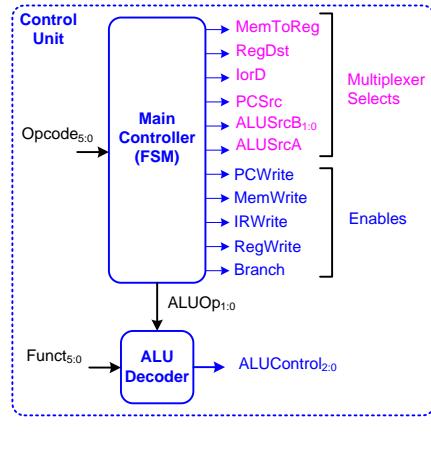
Instruction Fetch
Instruction Decode/Operand Fetch
Execute
Memory
Write Back

Apenas o lw → utilizados os estados

Unidade de Controlo MC - Principal + ALU Decoder

1. O Controlador Principal

É uma máquina síncrona.



2. O ALU Decoder

É igual ao do Single-cycle.

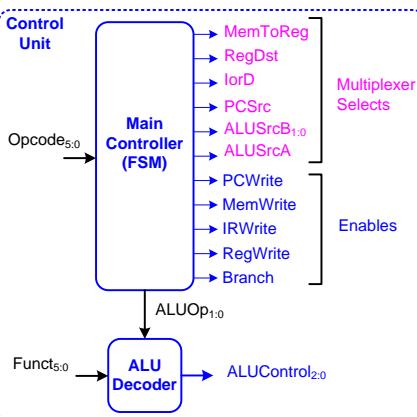
$ALUOp_{1:0}$	$Funct_{5:0}$	$ALUControl_{2:0}$
00	X	010 (Add)
01	X	110 (Subtract)
10	100000 (add)	010 (Add)
10	100010 (sub)	110 (Subtract)
10	100100 (and)	000 (And)
10	100101 (or)	001 (Or)
10	100110 (xor)	100 (Xor)
10	100111 (nor)	101 (Nor)
10	101010 (slt)	111 (Slt)

Se quisermos implementar um novo sinal, o 11 ou 10

Tipo R, decidido com o $Funct$

Controlador FSM (1) - Tipos de Sinais

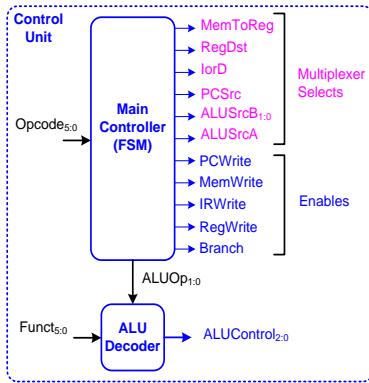
- O Controlador Principal gera 3 tipos de sinais:



- Seleção de Muxes
- Enable
- $ALUOp_{1:0}$

- O ALU Decoder gera: $ALUControl_{2:0}$.

Controlador FSM (2) - Estado e Sinais de Controlo



Tipos de sinais dentro de cada estado:

1. Seleção de multiplexers: exibem o respetivo valor binário.
2. Enable: só estão indicados quando activos.
3. ALUOp: o valor não aparece explicitado no datapath, sendo substituído pelo ALUControl (gerado pelo ALU Decoder).

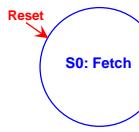
Exemplo:



Sobre os diagramas temporais seguintes:

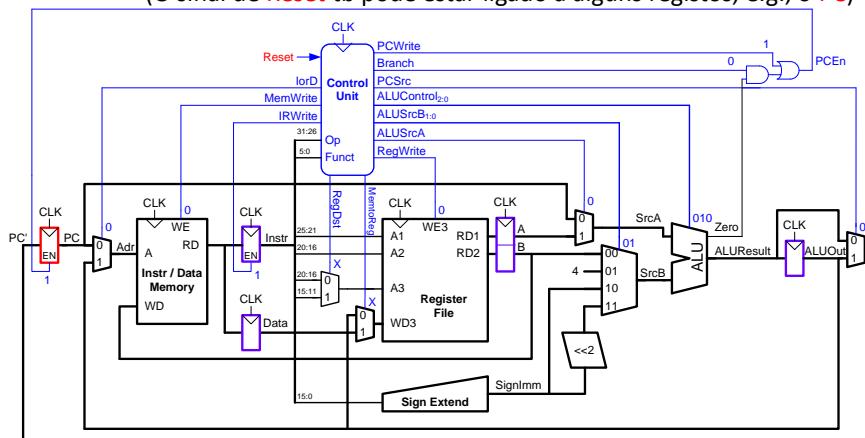
1. Podem ser ignorados numa primeira leitura.
2. Complementam a explicação sobre o funcionamento da FSM da Unidade de Controlo.

Controlador FSM (3) - Reset e Fetch: S0



O primeiro passo da execução consiste na leitura da instrução cujo endereço está contido no registo PC.

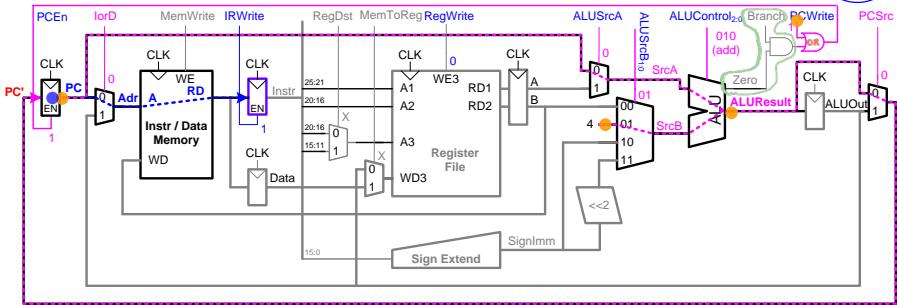
A FSM entra no estado de *Fetch* após o sinal de *Reset*.
(O sinal de *Reset* tb pode estar ligado a alguns registos, e.g., o PC)



Controlador FSM (4) - Fetch e IncPC (1): S0

S0-1: **Fetch**: Lê a instrução da Memória com $Adr = PC$ ($IorD = 0$).

- A instrução (RD) é escrita no Registo de Instrução activando IRWrite.



- Busca a instrução na mem
- Guarda no registo a instrução

S0-2: **IncPC**: Em paralelo com o **Fetch**, é calculado $PC' = PC + 4$:

$ALUSrcA = 0 \Rightarrow SrcA = PC$; $ALUSrcB = 01 \Rightarrow SrcB = 4$;

$ALUOp = 00 \Rightarrow ALUControl = 010$ (soma).

- PC será atualizado com PC' , fazendo:

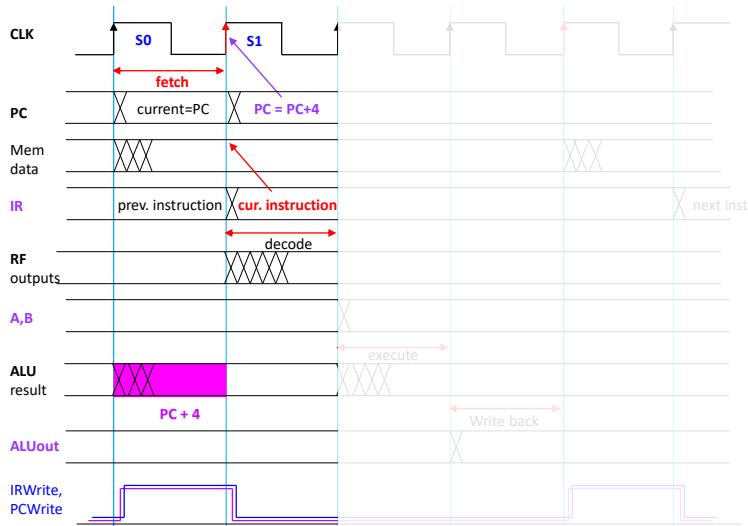
$PCSrc = 0 \Rightarrow PC' = ALUResult$;

$PCWrite$ activo $\Rightarrow PCEn = 1$.

S0: Fetch + IncPC

- Precisamos do $PC + 4$

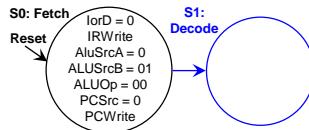
Controlador FSM (5) - Fetch e IncPC (2) - Timing



O **fetch** da instrução atual e o cálculo do **PC-seguinte** são feitos em paralelo.

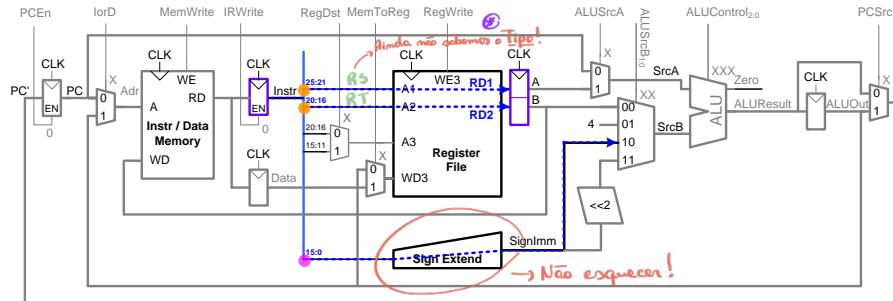
Controlador FSM (6) - Decode: S1

• Unidade de controlo recebe o Opcode!



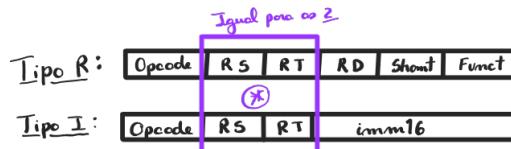
S1-1: O Banco de Registos lê sempre os dois operandos, especificados pelos campos **rs** e **rt** da instrução, e coloca-os nos registos **A** e **B**, respetivamente.

S1-2: Em paralelo, o valor imediato é *sign-extended*.

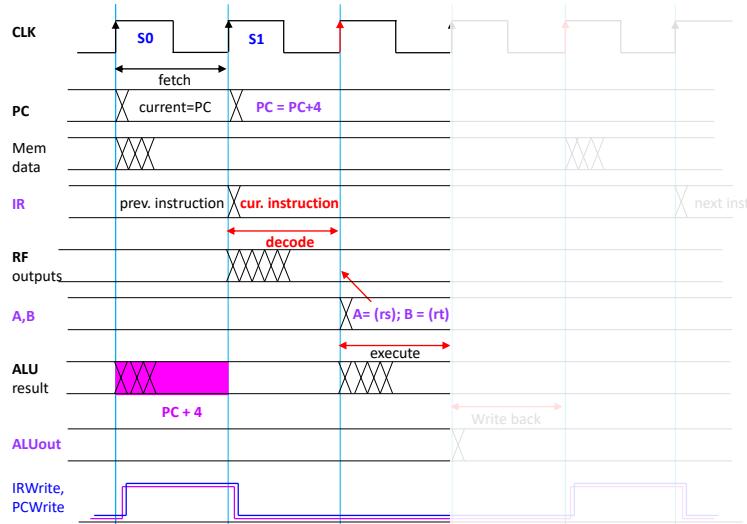


A fase de descodificação (UC) usa o **opcode** da instrução para decidir o que fazer a seguir.

Não São necessários sinais de controlo nesta fase. Todavia, a FSM deve **aguardar** um ciclo de *clock* para que as operações de leitura (RF) e descodificação se completem.

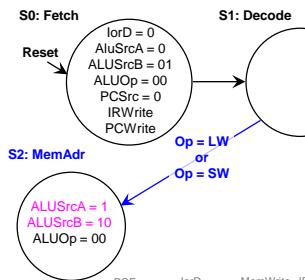


Controlador FSM (7) - Decode (2) - Timing

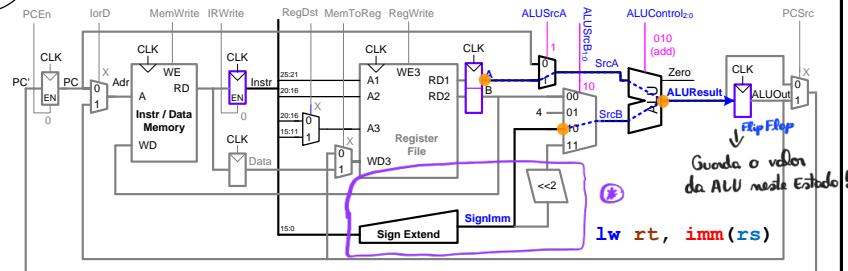


A FSM **aguarda** um ciclo de *clock* para que as operações de leitura (RF) e descodificação se completem.

Control. FSM (8) - lw/sw - Cálculo do Endereço: S2



S2: Se a instrução for **lw** ou **sw**, é calculado o endereço efetivo, adicionando ao Endereço Base (A) o valor imediato depois de *sign-extended* (*SignImm*).
Isto require:
ALUSrcA = 1 para selecionar o registo A
ALUSrcB = 10 para selecionar *SignImm*
ALUOp = 00 para a ALU somar (*ALUControl* = 010).
O endereço é armazenado no registo ALUOut.



Execute

• MemAddress = RS + SE(immm16)

*Após S1 os estados seguintes dependem da instrução. Comecemos com lw/sw.

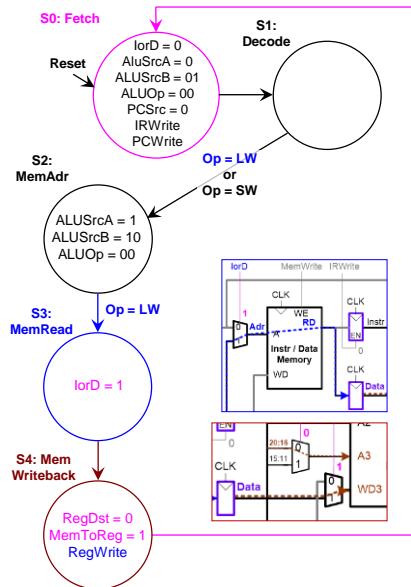
© A. Nunes da Cruz

IAC - MIPS - Multicycle: Unidade de Controlo

10/29

- Mem Address = RS + SignExtend(immm16)
- BTA = (PC+4) + SignExtend(immm16) << 2 → usado no beq.

Controlador FSM (9) - lw (1): S3 + S4



Leitura do valor da memória e escrita no Banco de Registros:

S3: IorD = 1, seleciona o endereço que se encontra em ALUOut. O valor lido da memória é armazenado no registo Data.

S4: O valor em Data é escrito no Banco de Registros.

RegDst = 0, seleciona o registo rt;

MemToReg = 1, seleciona Data;

RegWrite: é ativado para escrever no RF, completando a execução de lw.

S0: Regresso ao estado inicial, para o fetch da instrução seguinte.

lw rt, imm(rs)

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Unidade de Controlo

11/29

2 estados →

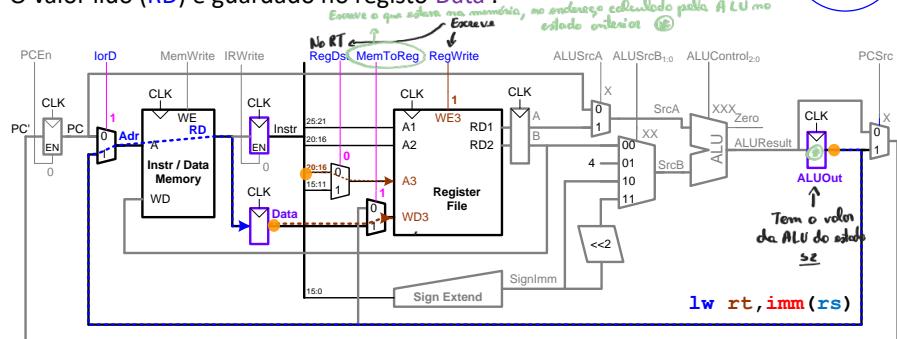
1

Controlador FSM (10) - Iw (2): S3 + S4

Leitura do valor da memória e **escrita** no Banco de Registos (**rt**):

S3: $l0rD = 1$, seleciona o endereço (Adr) que se encontra em ALUOut.

O valor lido (RD) é guardado no registo Data .



S4: O valor em **Data** é escrito no **Banco de Registos**.

RegDst = 0 seleciona o registo **rt**, e **MemToReg** = 1 seleciona **Data**.

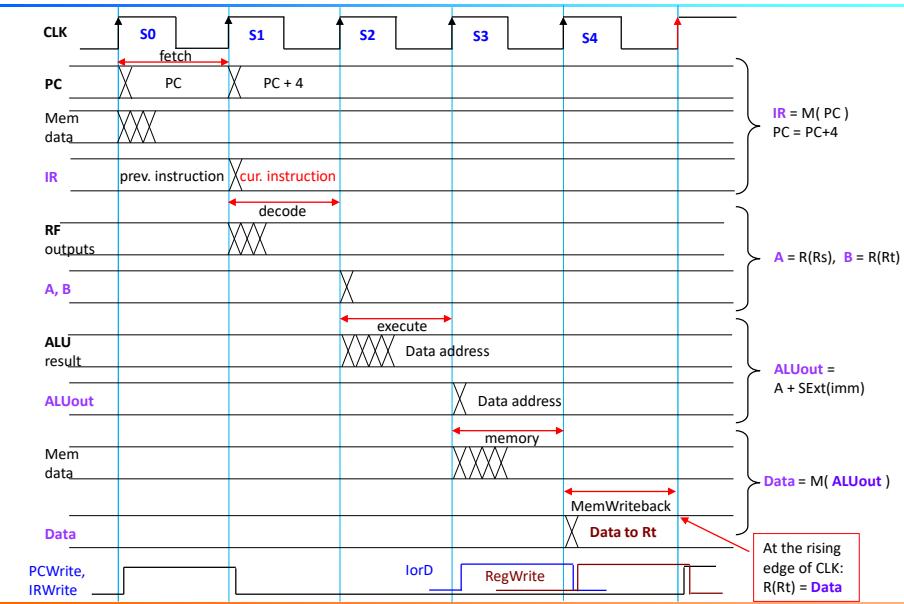
RegWrite é ativado para escrever, **completando** a execução de **Iw**.

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Unidade de Controlo

12/29

Controlador FSM (11) - Iw (3): Timing - 5 ciclos

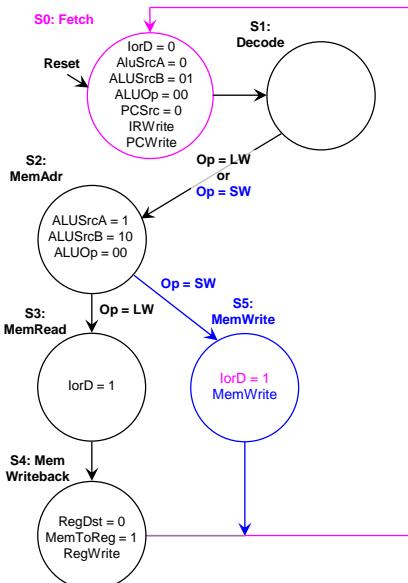


© A. Nunes da Cruz

IAC - MIPS - Multicycle: Unidade de Controlo

13/29

Controlador FSM (12) - sw (1): S5 - 4 ciclos



O valor do registo B é escrito na memória.

S5: IorD = 1 seleciona o endereço guardado em ALUOut (calculado em S2)

MemWrite é ativado para escrever na memória o valor B, completando a execução de sw.

(Menos um ciclo que lw!)

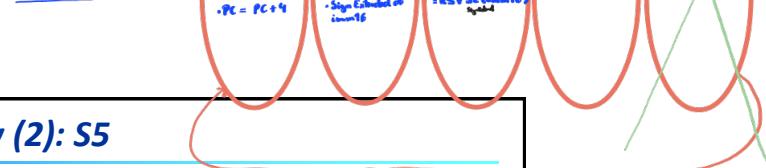
sw rt, imm(rs)

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Unidade de Controlo

14/29

Para o sw: =>



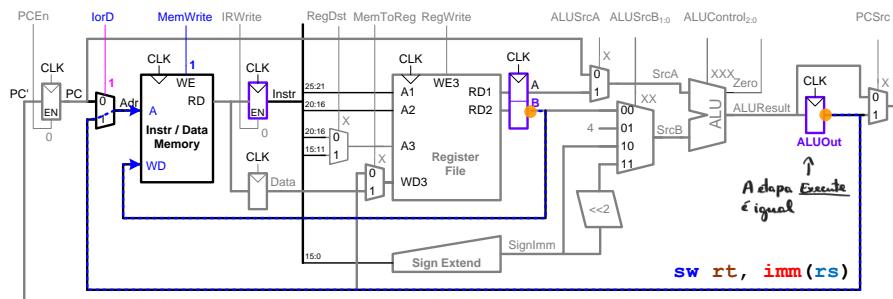
Controlador FSM (13) - sw (2): S5

O valor lido do segundo porto do RF (B) é escrito na memória:

S5: IorD = 1 seleciona o endereço guardado em ALUOut (calc. em S2).

MemWrite é ativado para escrever na memória o valor B.

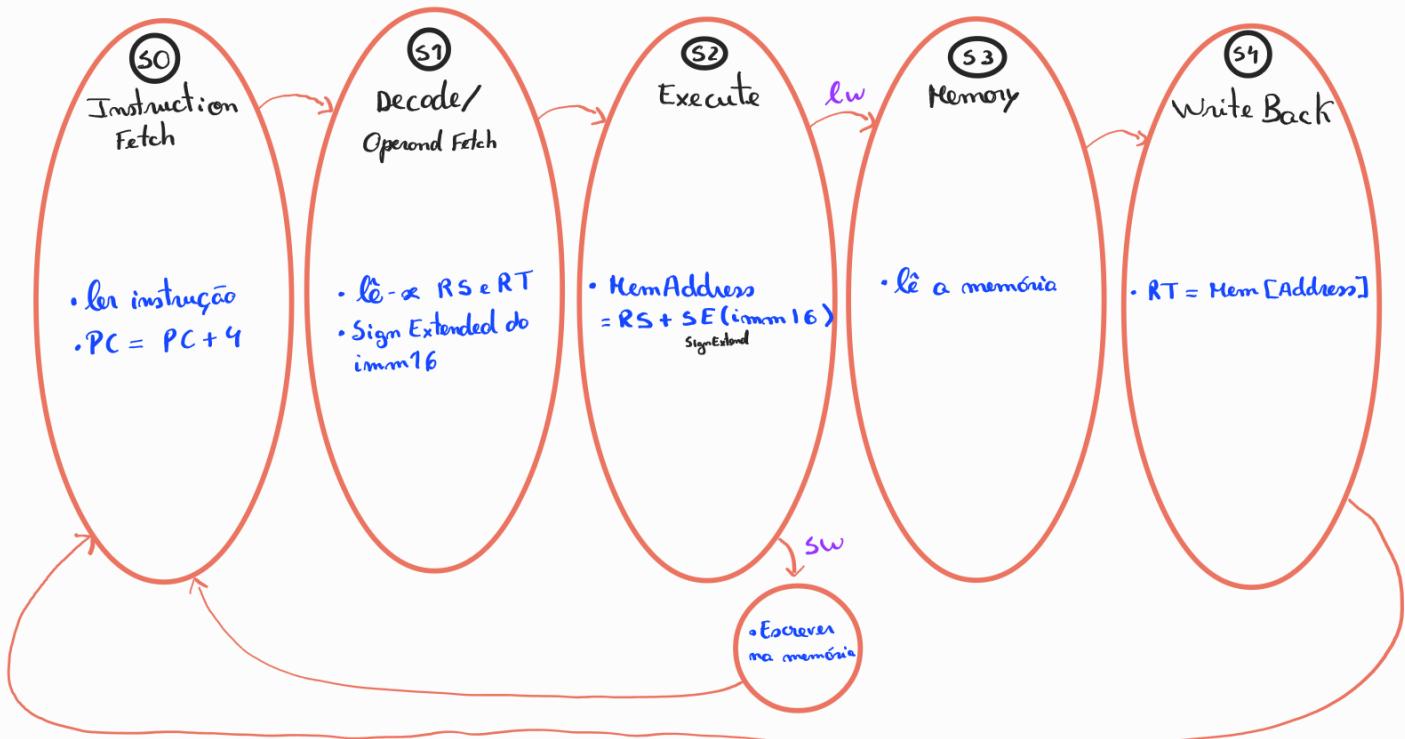
S5
IorD = 1
MemWrite



© A. Nunes da Cruz

IAC - MIPS - Multicycle: Unidade de Controlo

15/29



Caso mais geral:



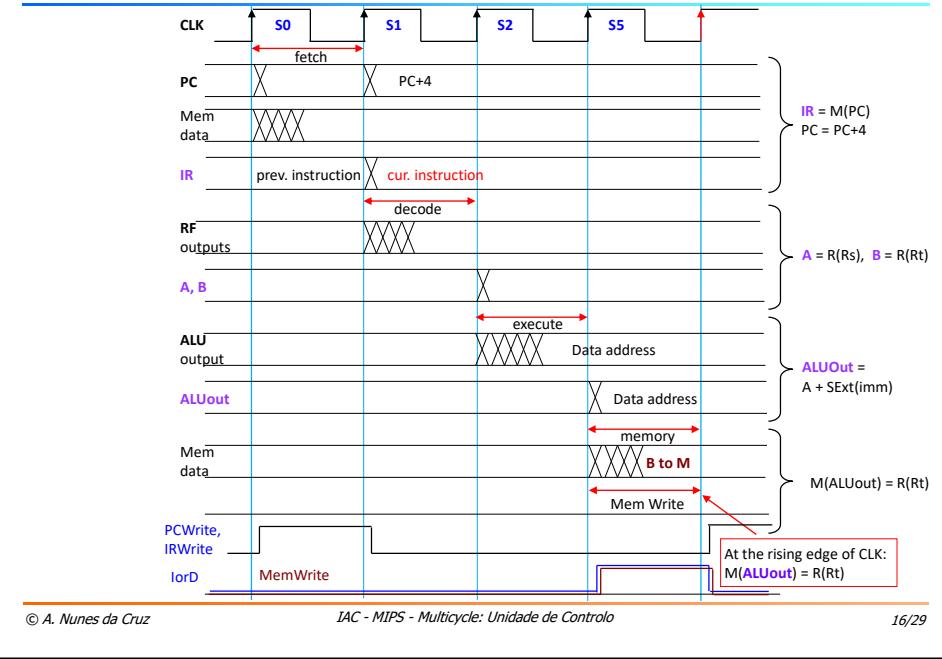
Tipo R:

Opcode	RS	RT	RD	Shamt	Funct
--------	----	----	----	-------	-------

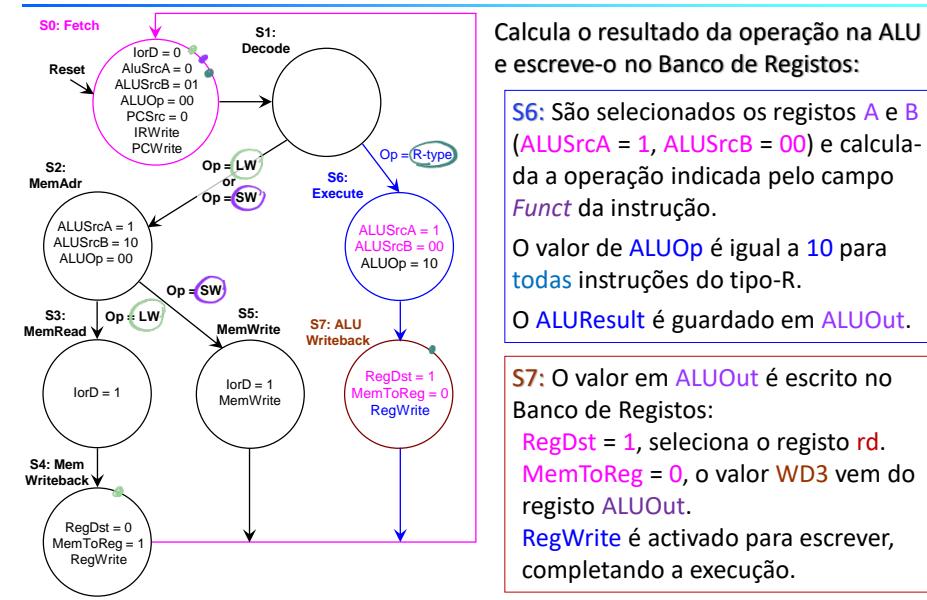
Tipo I:

Opcode	RS	RT	imm16
--------	----	----	-------

Controlador FSM (14) - sw (3): Timing - 4 ciclos



Controlador FSM (15) - tipo-R (1): S6 + S7



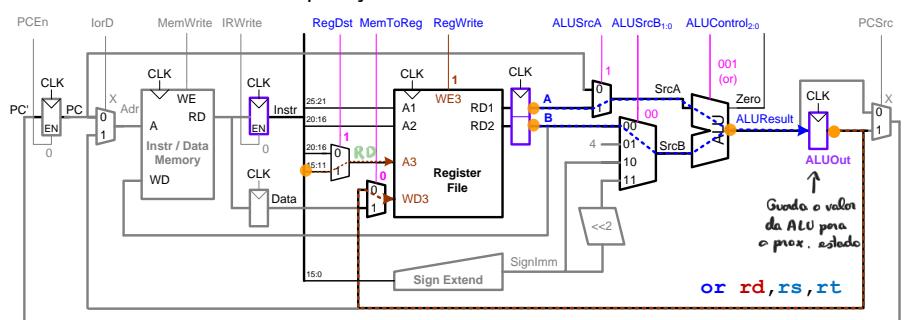
Controlador FSM (16) - tipo-R (2): S6 + S7

Calcula o resultado da operação na ALU e **escreve-o** no RF:

S6: Seleciona os registos A e B ($ALUSrcA = 1$, $ALUSrcB = 00$);

O valor de $ALUOp$ é igual a **10** para todas instruções do tipo-R;

Guarda o resultado da operação em **ALUOut**.



S7: O valor em **ALUOut** é escrito no Banco de Registros:

RegDst = 1 seleciona o registo destino **rd**;

MemToReg = 0 o valor a escrever (**WD3**) vem do registo **ALUOut**;

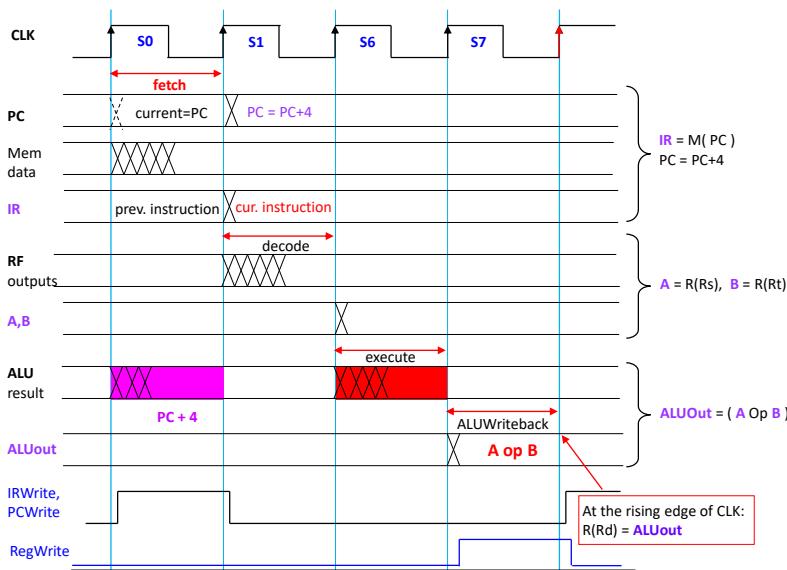
RegWrite é activado para escrever no RF.

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Unidade de Controlo

18/29

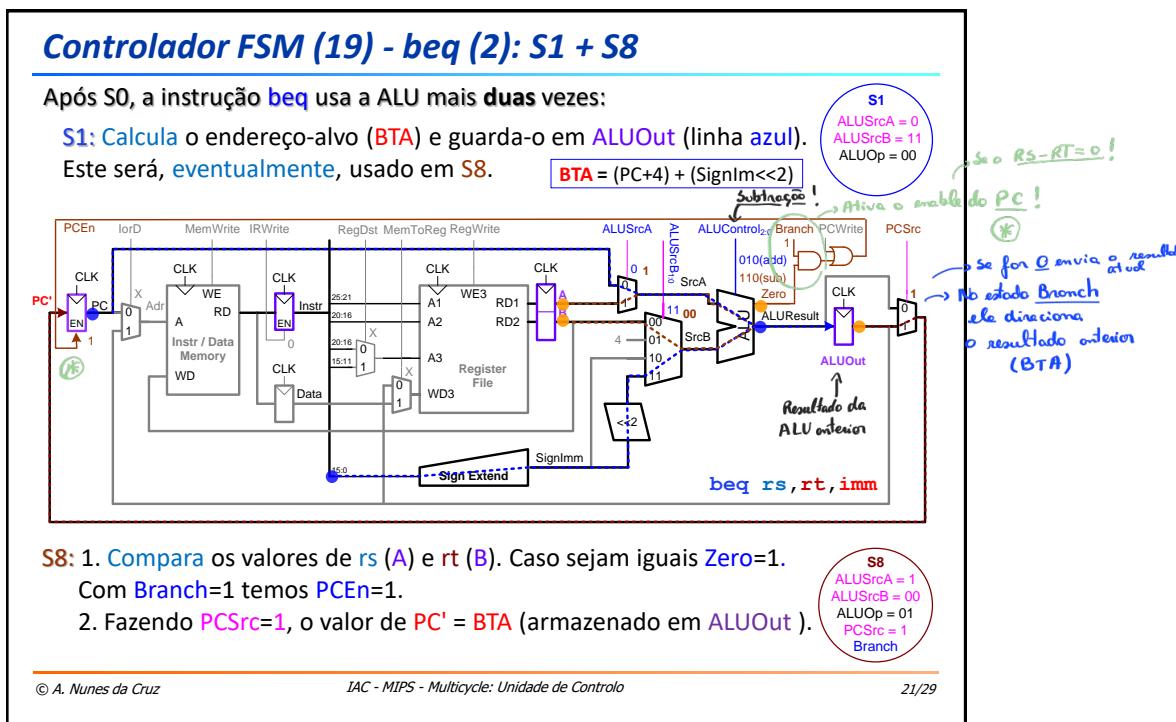
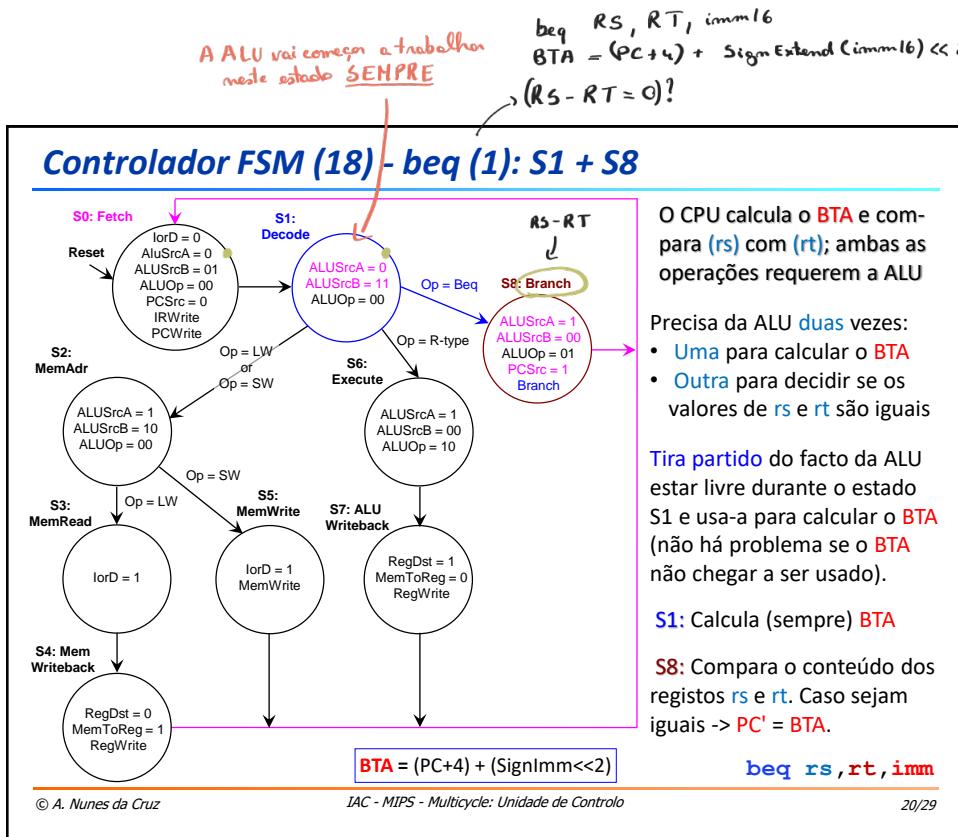
Controlador FSM (17) - tipo-R (3): Timing - 4 ciclos



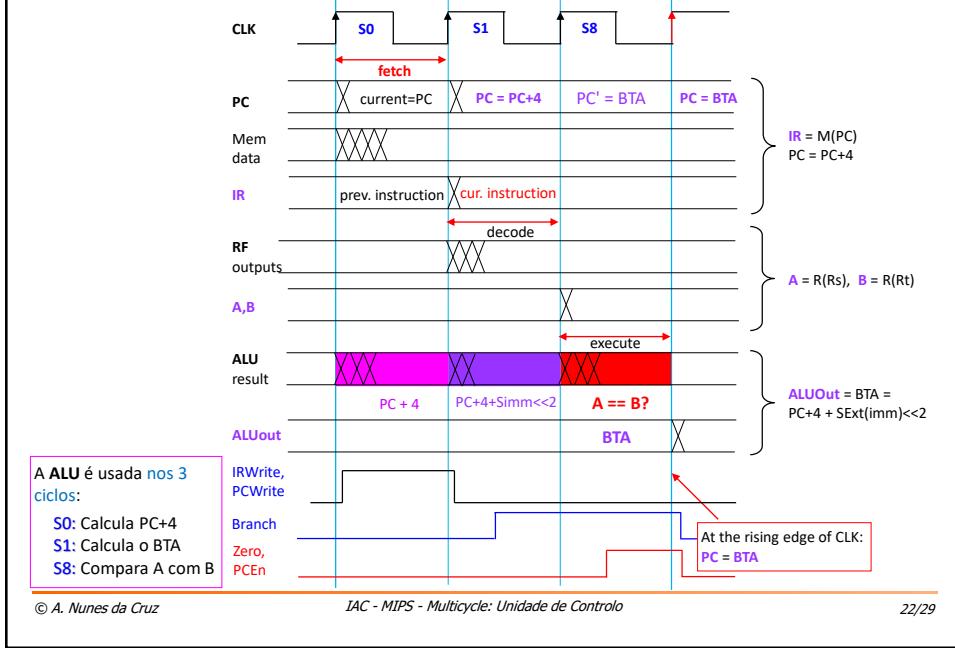
© A. Nunes da Cruz

IAC - MIPS - Multicycle: Unidade de Controlo

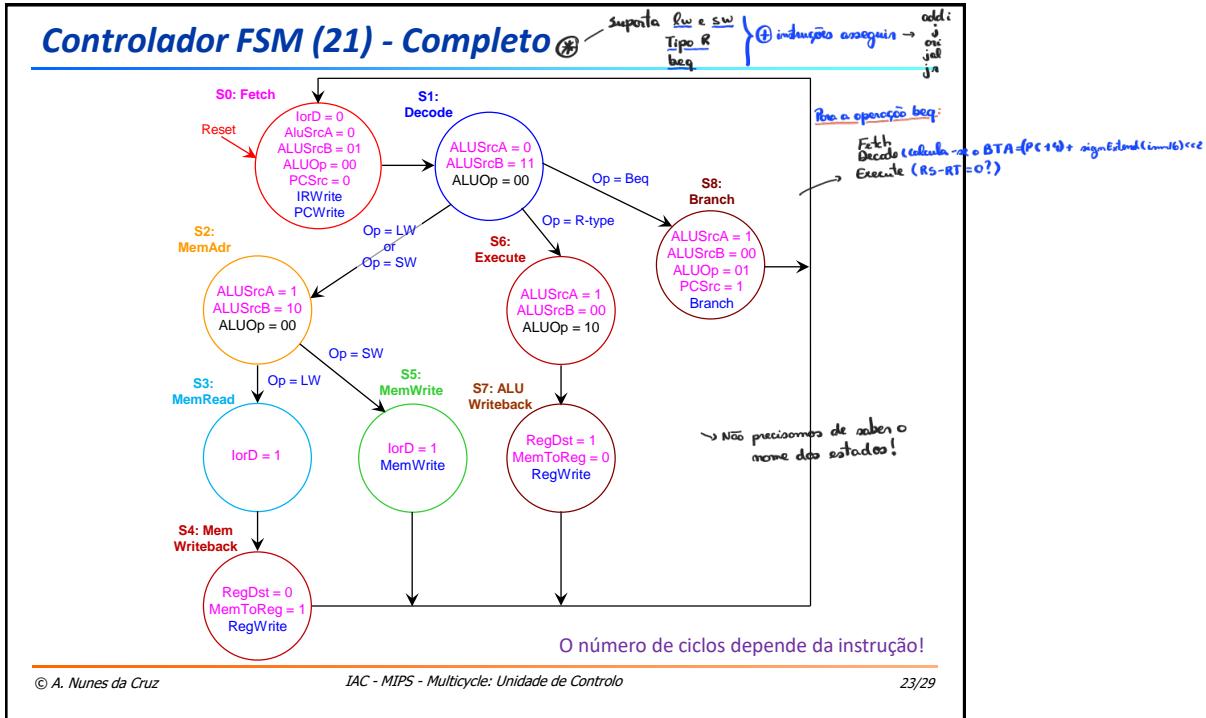
19/29



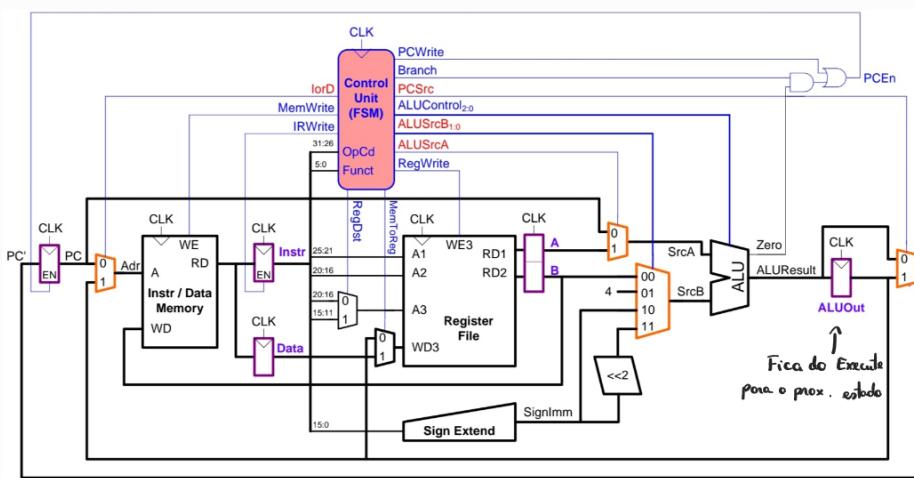
Controlador FSM (20) - beq (3): Timing - 3 ciclos



Controlador FSM (21) - Completo



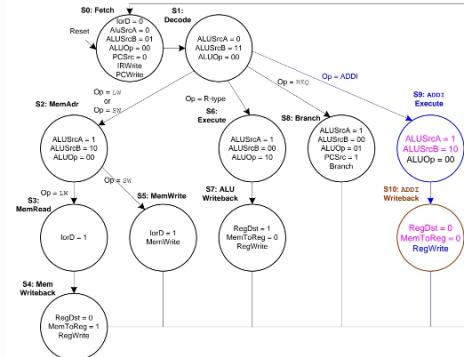
Todos executam → { Instruction Fetch
Instruction Decode / Op. Fetch
Já sabemos qual → { Execute
Memory Write Back
Só são executados todos na instrução Load word (lw)



addi :

addi RT, RS, imm16 \rightarrow RT = RS + SignExtend (imm16)

- ✓ Instruction Fetch
 - ✓ Instruction Decode / Op. Fetch
 - ✓ Execute \rightarrow Conta na ALU
 - ✗ Memory \rightarrow Não vai à memória
 - ✓ Write Back
- Tempo de aterrar o diagrama de estados



jump :

$$PC = JTA = (PC + 4)_{31:28}$$

- ✓ Instruction Fetch
- ✓ Instruction Decode / Op. Fetch
- ✓ Execute \rightarrow Conta na ALU
- ✗ Memory
- ✗ Write Back

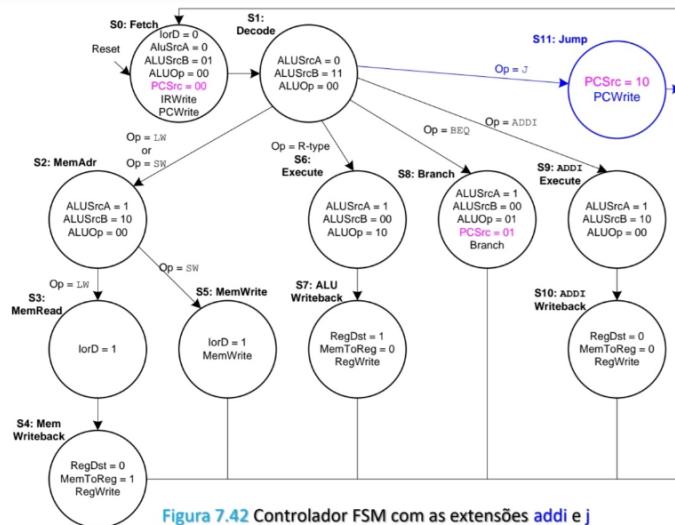
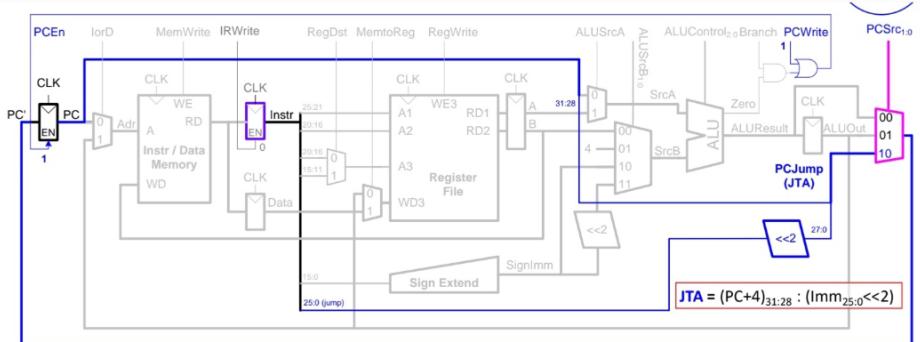
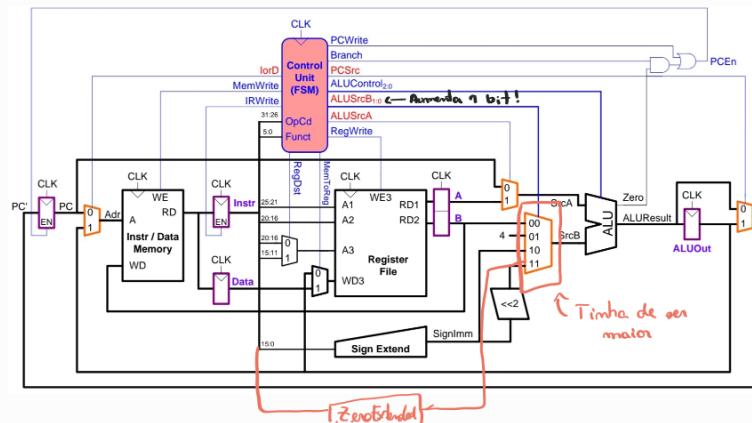


Figura 7.42 Controlador FSM com as extensões addi e j

ori:

$RT = RS \text{ or } \text{ZeroExtend}(\text{imm16})$

↳ Adiciona 1 estado e um mux.

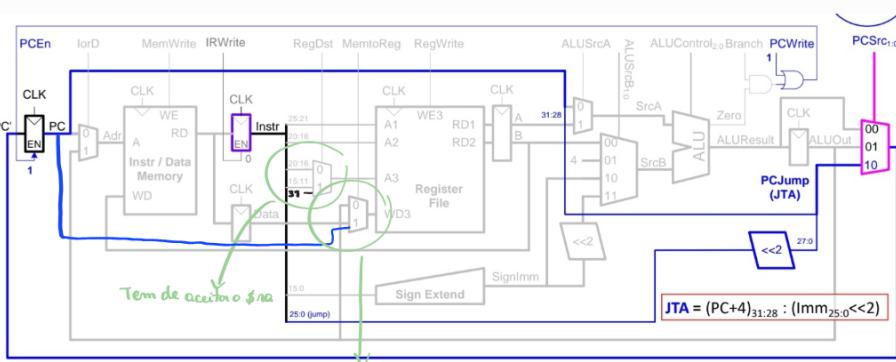


jal:

$$\text{jal target} + \$ra = PC + 4$$

jump link

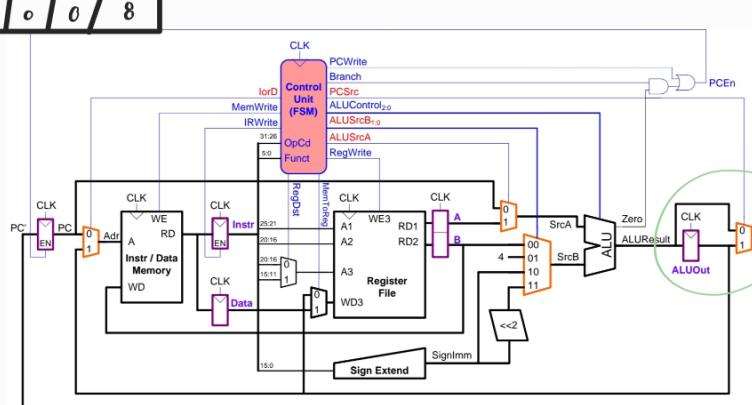
- ✓ Instruction Fetch
- ✓ Instruction Decode / Op. Fetch
- ✗ Execute → conta na ALU
→ Fazemos o link, também
- ✗ Multiplex
- ✗ Write Back



Põe fogo a soma na ALU → Sem adicional "link"

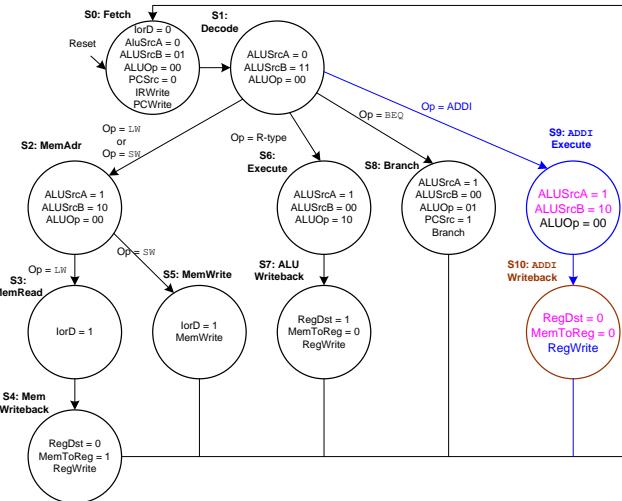
RT RD Short

jr: Tipo R Opcode | RS | 0 | 0 | 0 | 8



Extensão (1) - addi: FSM (1)

P: Como modificar o CPU para suportar addi?



R: O datapath já é capaz de adicionar o conteúdo dum registo com o valor imediato!

➤ Só precisamos de adicionar novos estados à FSM do controlador para addi.

Os estados são semelhantes aos usados pelas instruções do tipo-R.

S9: Ao registo A é somado SignImm e ALUResult é guardado em ALUOut.

S10: ALUOut é escrito no RF.

addi rt,rs,imm

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Unidade de Controlo

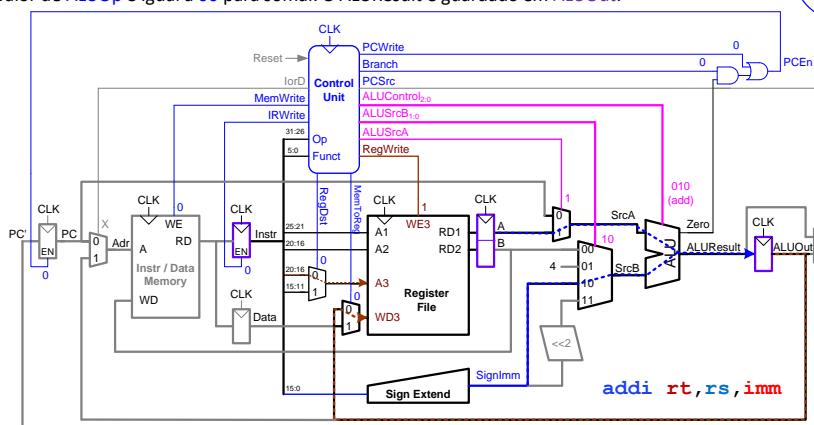
24/29

Extensão (2) - addi: FSM (2) - S9 + S10

O CPU calcula o resultado na ALU e escreve-o no Banco de Registros:

S9: São selecionados o reg. A e SignImm (ALUSrcA = 1, ALUSrcB = 10) e calculada a operação na ALU. O valor de ALUOp é igual a 00 para somar. O ALUResult é guardado em ALUOut.

S9
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00



S10: O valor em ALUOut é escrito no Banco de Registros. RegDst = 0 selecciona o registo destino rt. MemToReg = 0, significa que o valor a escrever em WD3 vem de ALUOut.

RegWrite é activado para escrever no RF, completando a execução da instrução do tipo-I.

S10
RegDst = 0
MemToReg = 0
RegWrite

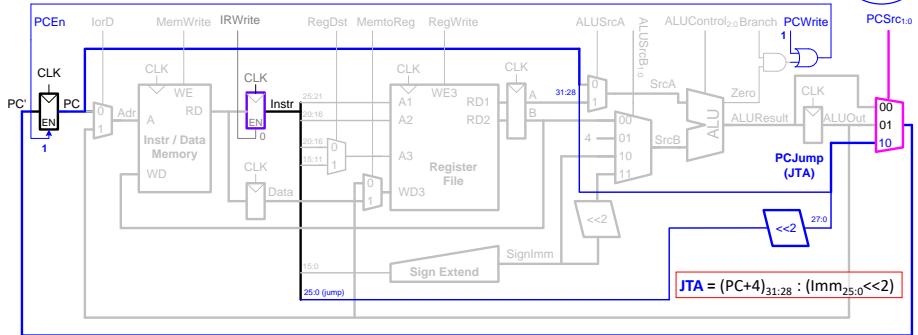
© A. Nunes da Cruz

IAC - MIPS - Multicycle: Unidade de Controlo

25/29

Extensão (3) - j: Datapath

P: Como modificar o CPU para suportar j?



R: 1. Modificamos o datapath para calcular o valor PC' no caso da instrução 'j'.

$$\text{JTA (Jump Target Address)} = (\text{PC} + 4)_{31:28} : \text{Imm}_{25:0} \ll 2 ; \quad (\text{requere mais um } <<2)$$

O multiplexor PCSrc aceita uma terceira entrada JTA, selecionada com PCSrc = 10 .

2. Adicionamos um estado (S11) ao controlador FSM para gerar PCSrc = 10 e PCWrite.

(O PCWrite é ativado para forçar PCEn=1).

Extensão (4) - j: FSM

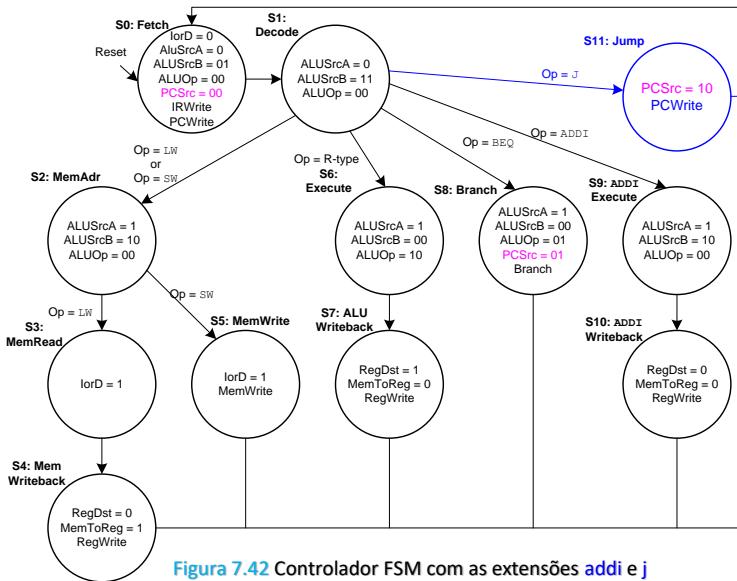
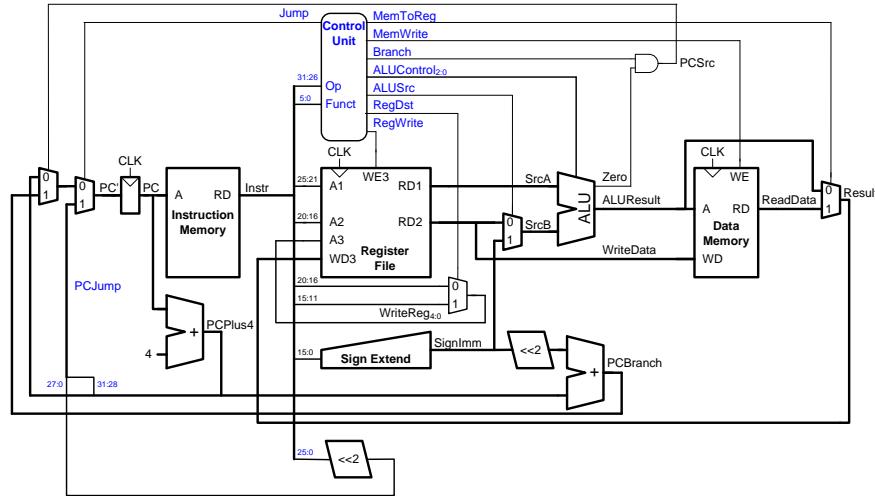


Figura 7.42 Controlador FSM com as extensões addi e j

Revisão (1) - MIPS Single-Cycle

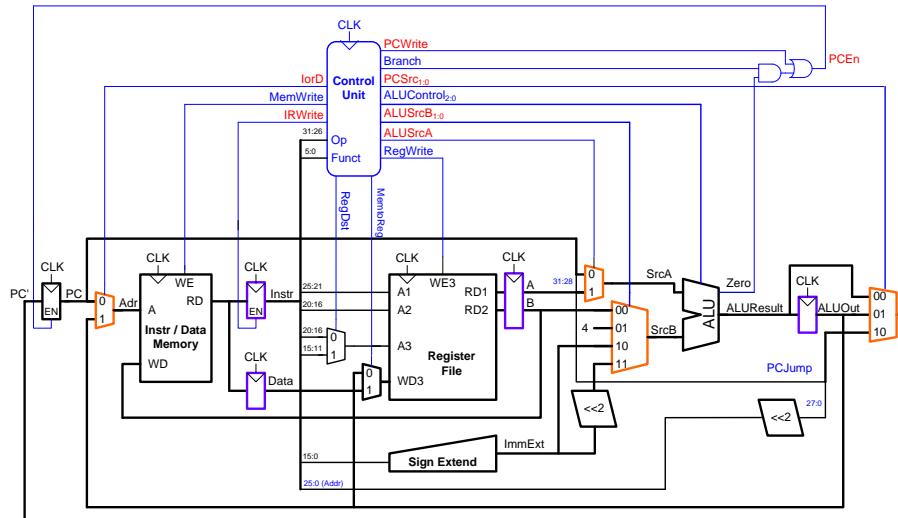


© A. Nunes da Cruz

IAC - MIPS - Multicycle: Unidade de Controlo

28/29

Revisão (2) - MIPS Multicycle



*O ISA só suporta: lw/sw, tipo-R, beq + addi e j.

© A. Nunes da Cruz

IAC - MIPS - Multicycle: Unidade de Controlo

29/29