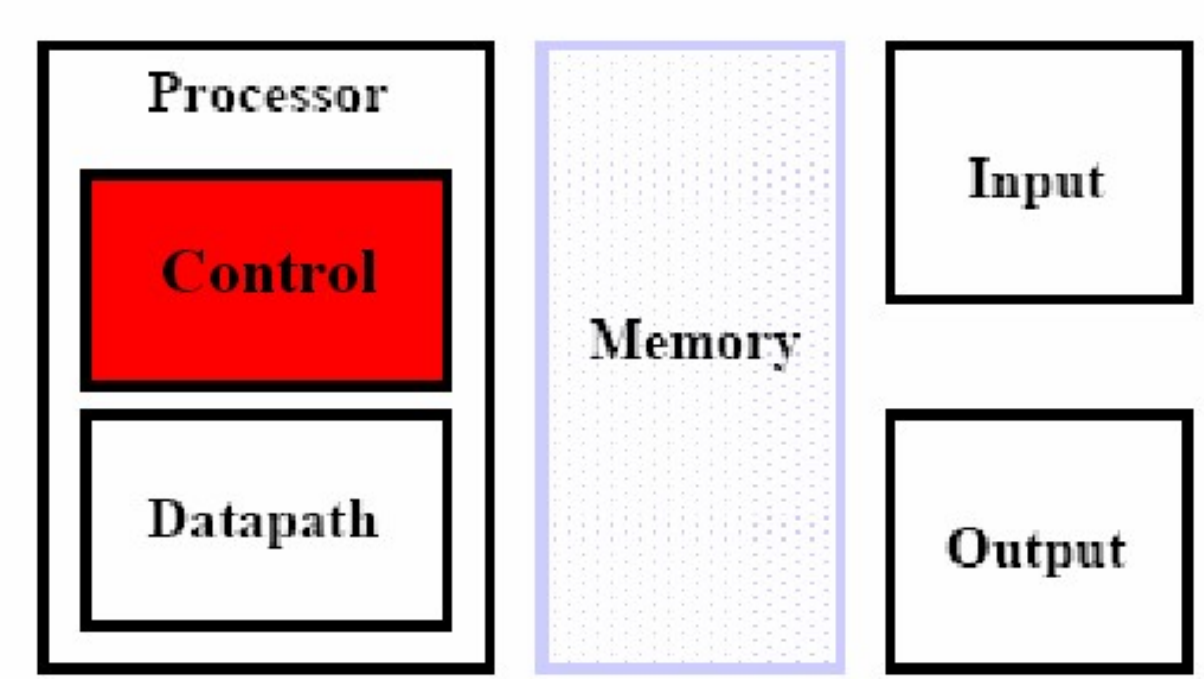
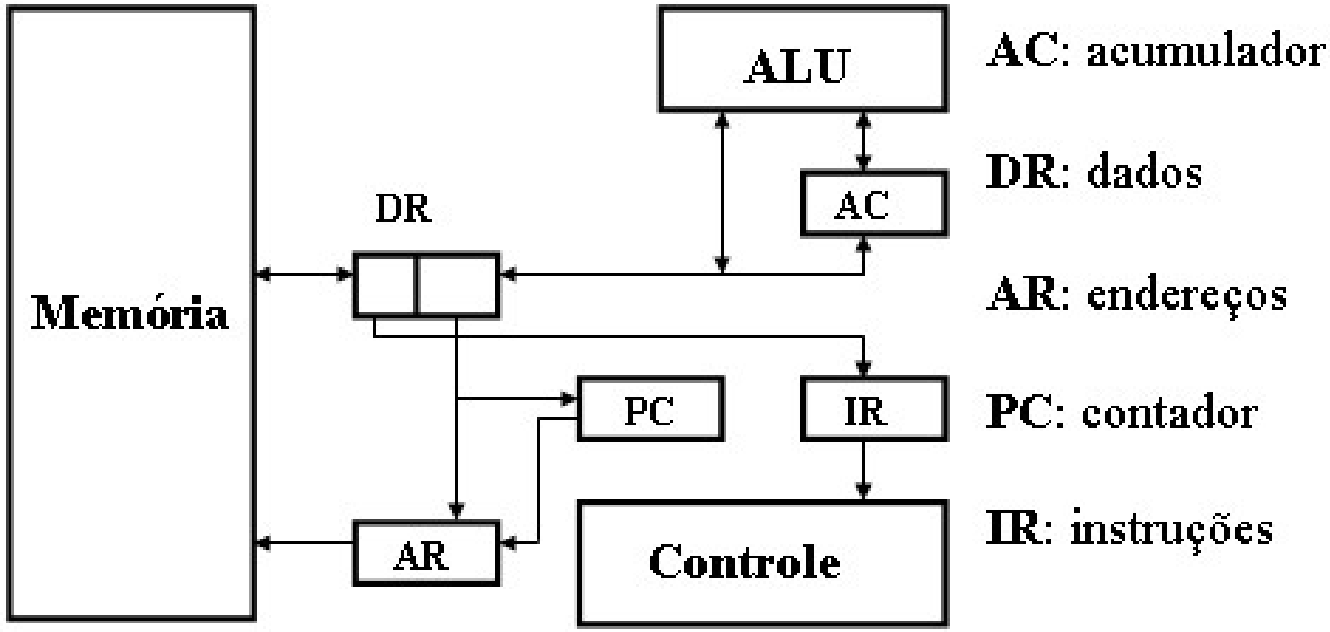
Controle do Pipeline

CES-25 – Arquiteturas para Alto Desmpenho
Prof. Paulo André Castro
pauloac@ita.br
Sala 110 – Prédio da Computação
www.comp.ita.br/~pauloac
IEC - ITA

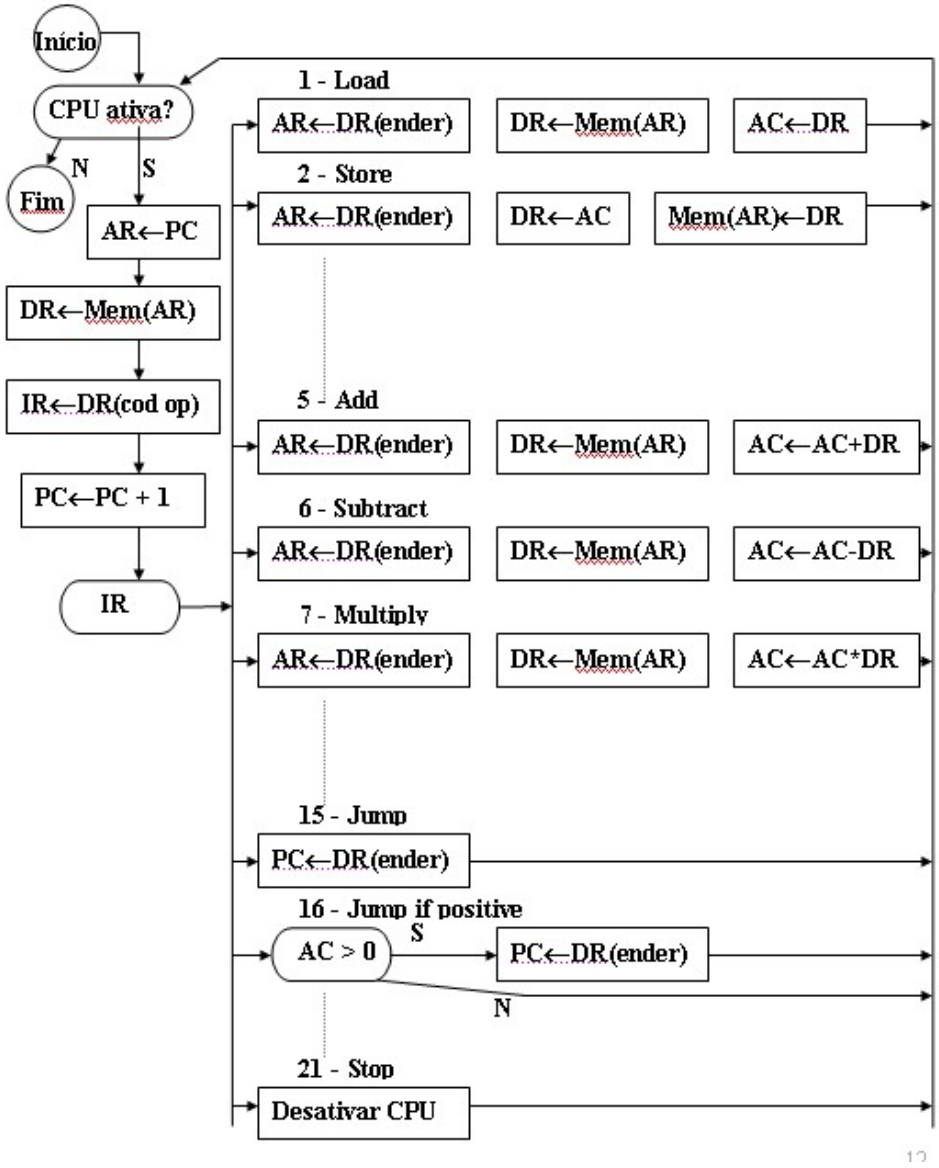
Controle do processador: um dos cinco componentes clássicos de um Computador



Exemplo simples de Controle do Processador: Acumulador



Microprograma da CPU com Acumulador



MIPS - Principais Instruções

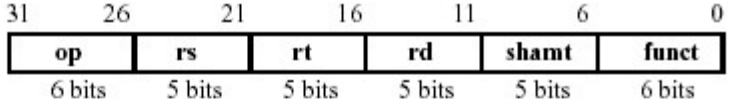
- **Adição**
  - **add R1,R2,R3; R1 = R2 + R3**
- **Subtração**
  - **sub R1,R2,R3; R1 = R2 – R3**
- **Adição de constante (add immediate)**
  - **addi R1,R2,100; R1 = R2 + 100**
- **Multiplicação (resultado em 64 bits)**
  - **mult R2,R3; Hi, Lo = R2 x R3**
- **Divisão (resultado em 64 bits)**
  - **div R2,R3; Lo = R2 ÷ R3, Hi = R2 mod R3**
  - **Lo = quotient, Hi = remainder**

MIPS - Principais Instruções - 2

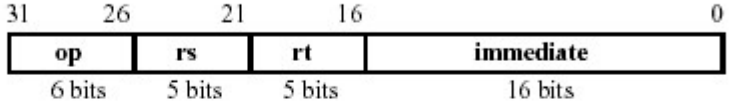
- **Alterar memória (word)**
  - **SW R3, 500(R4) Mem[R4 + 500] =R3**
- **Ler memória (word)**
  - **LW R1, 30(R2) R1 = Mem[R2 + 30]**
- **Desvio Condicional**
  - **beq R1,R2,100 if (R1 == R2) go to PC+4+400**
- **Desvio incondicional (constante)**
  - **jump j 2500; go to 10000**
- **Desvio incondicional (registrador)**
  - **jr R31; go to R31**

Formato dos Conjunto de Instruções MIPS

- ADD and subtract
  - add rd, rs, rt
  - sub rd, rs, rt

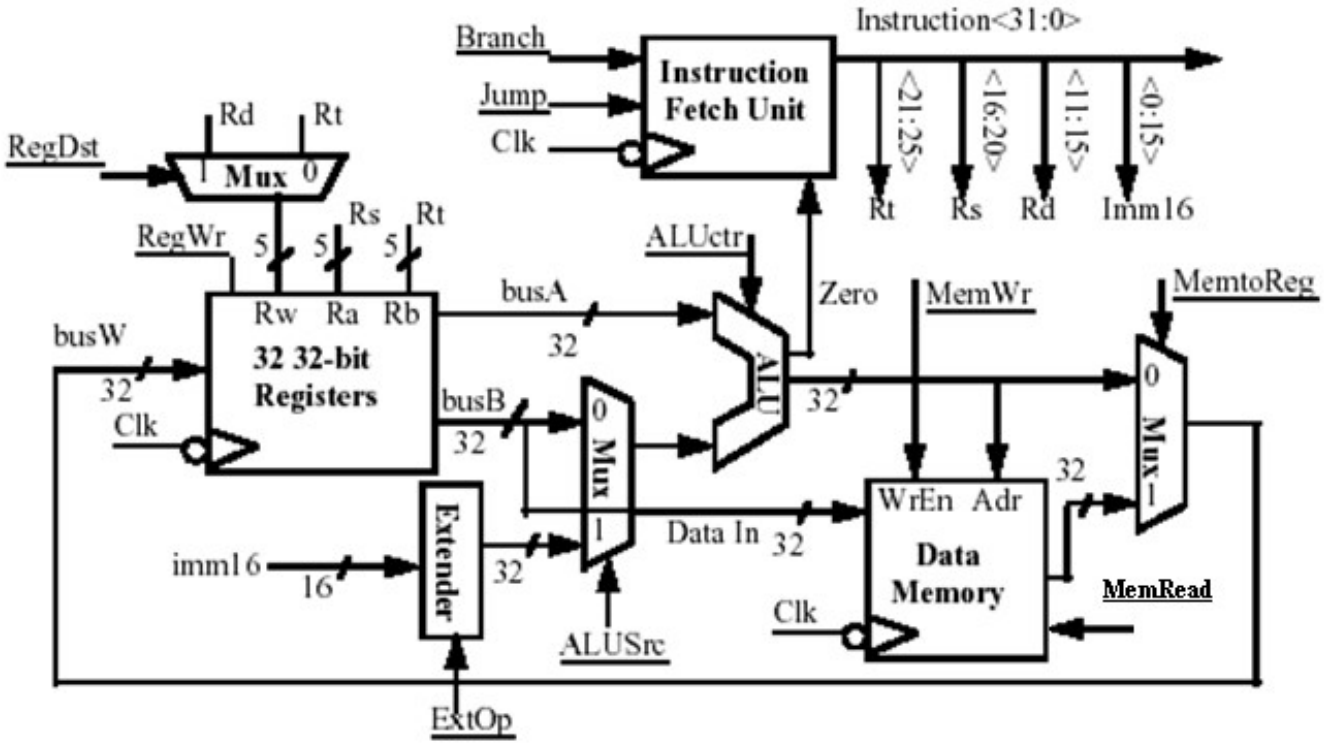


- OR Imm:
  - ori rt, rs, imm16

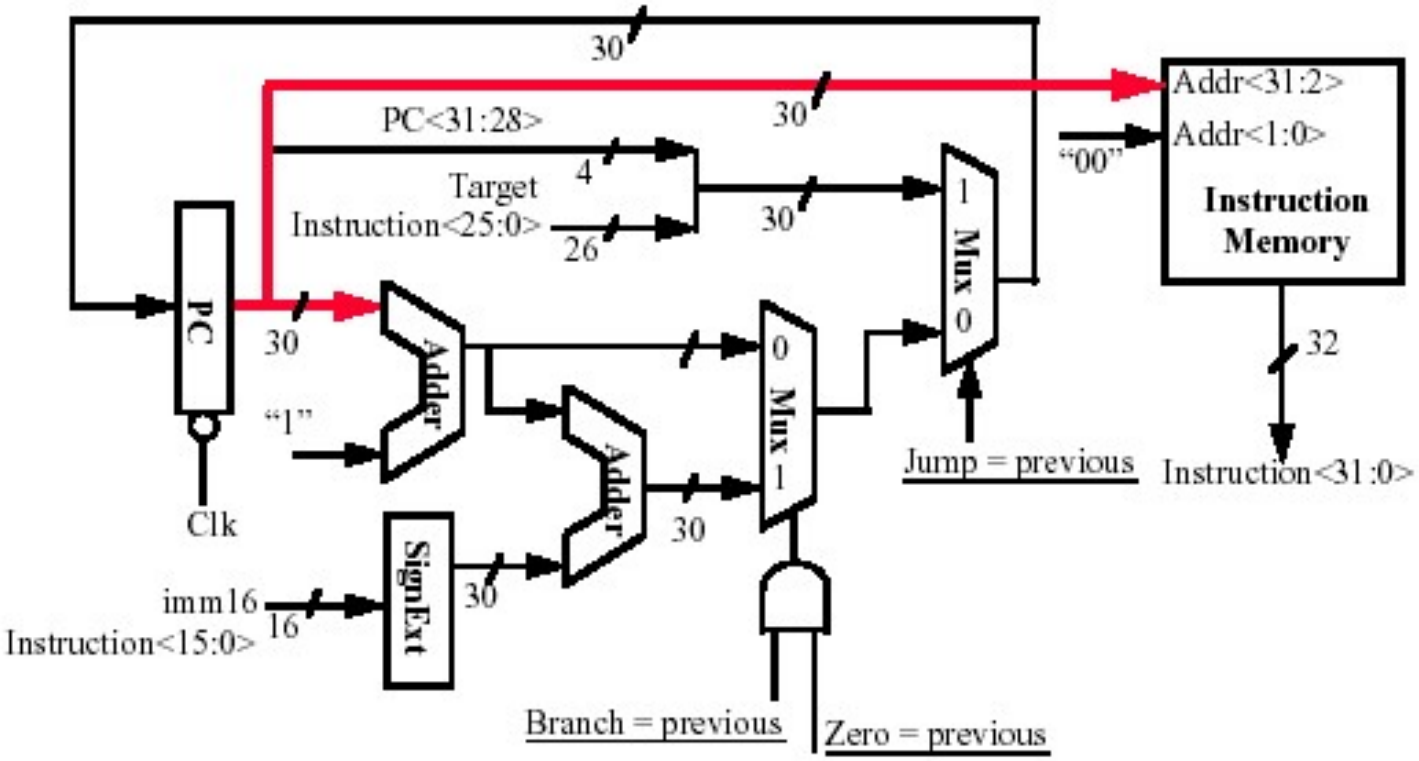


- LOAD and STORE
  - lw rt, rs, imm16
  - sw rt, rs, imm16
- BRANCH:
  - beq rs, rt, imm16
- JUMP:
  - j target

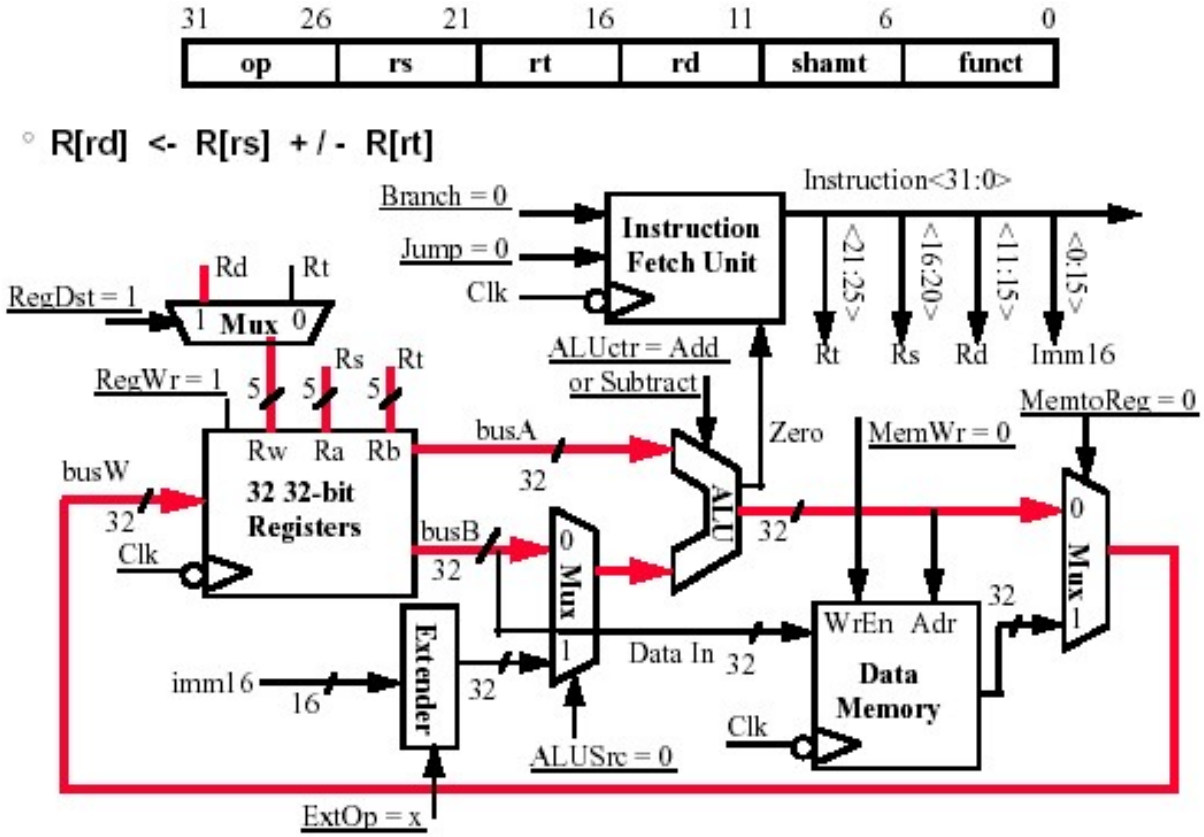
Unidades Funcionais e Controle no MIPS



Carregamento de Instrução

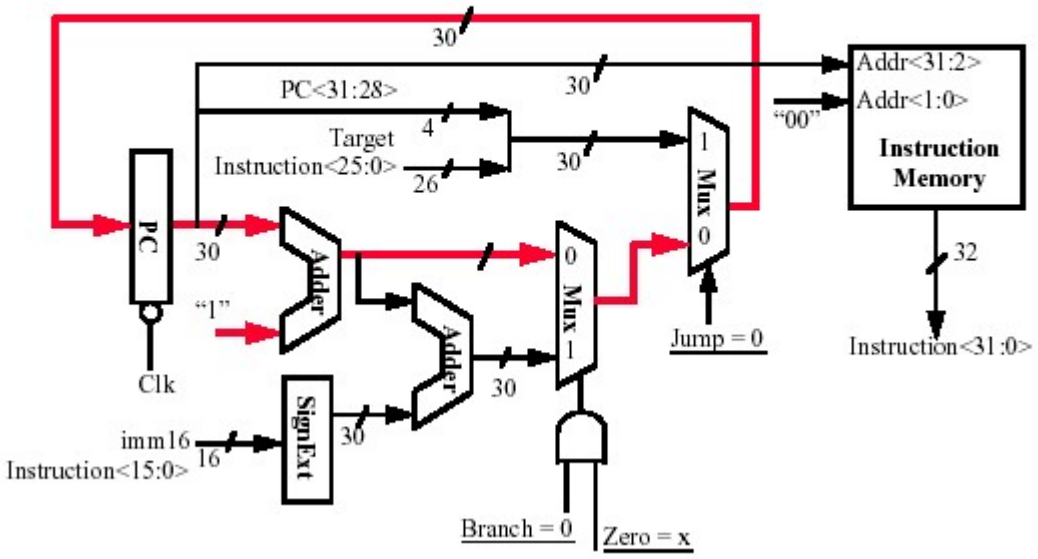


Controlando Operações R-type Adição/Subtração/And/or...

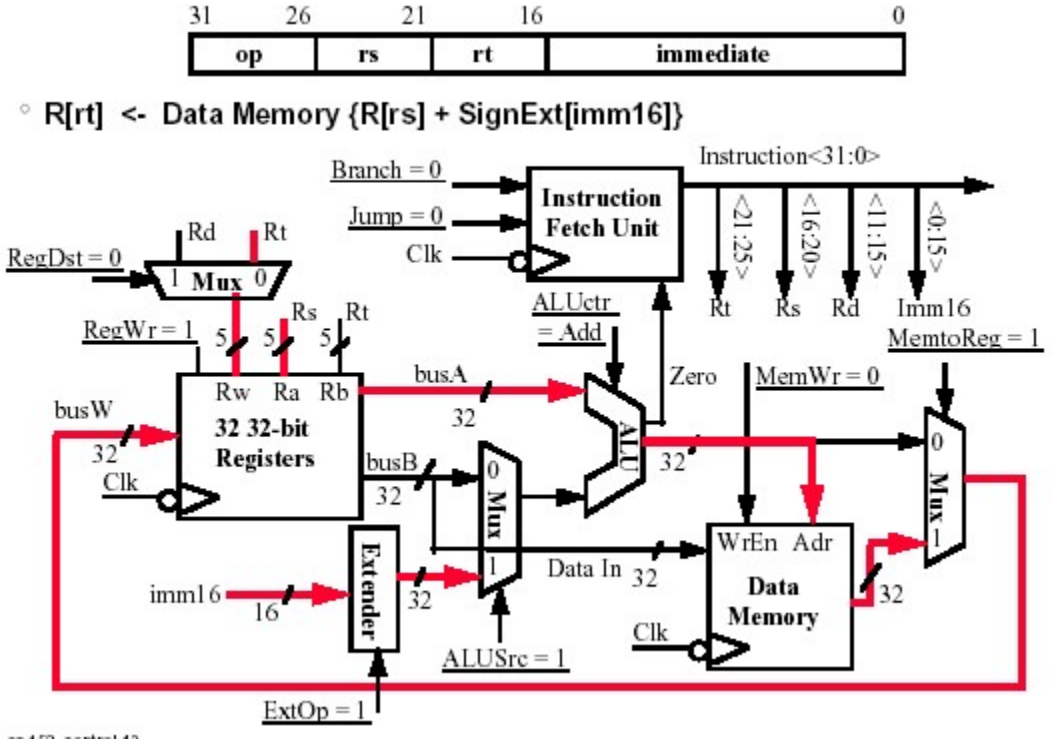


IF no fim de uma instrução

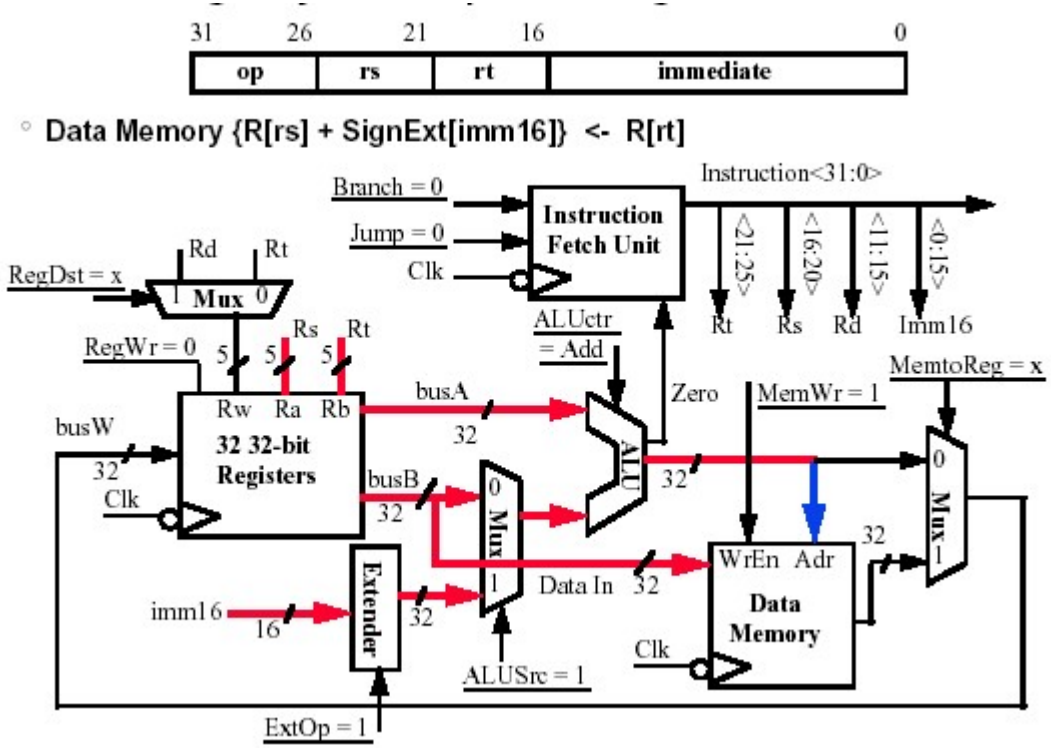
- PC <- PC + 4
  - This is the same for all instructions except: Branch and Jump



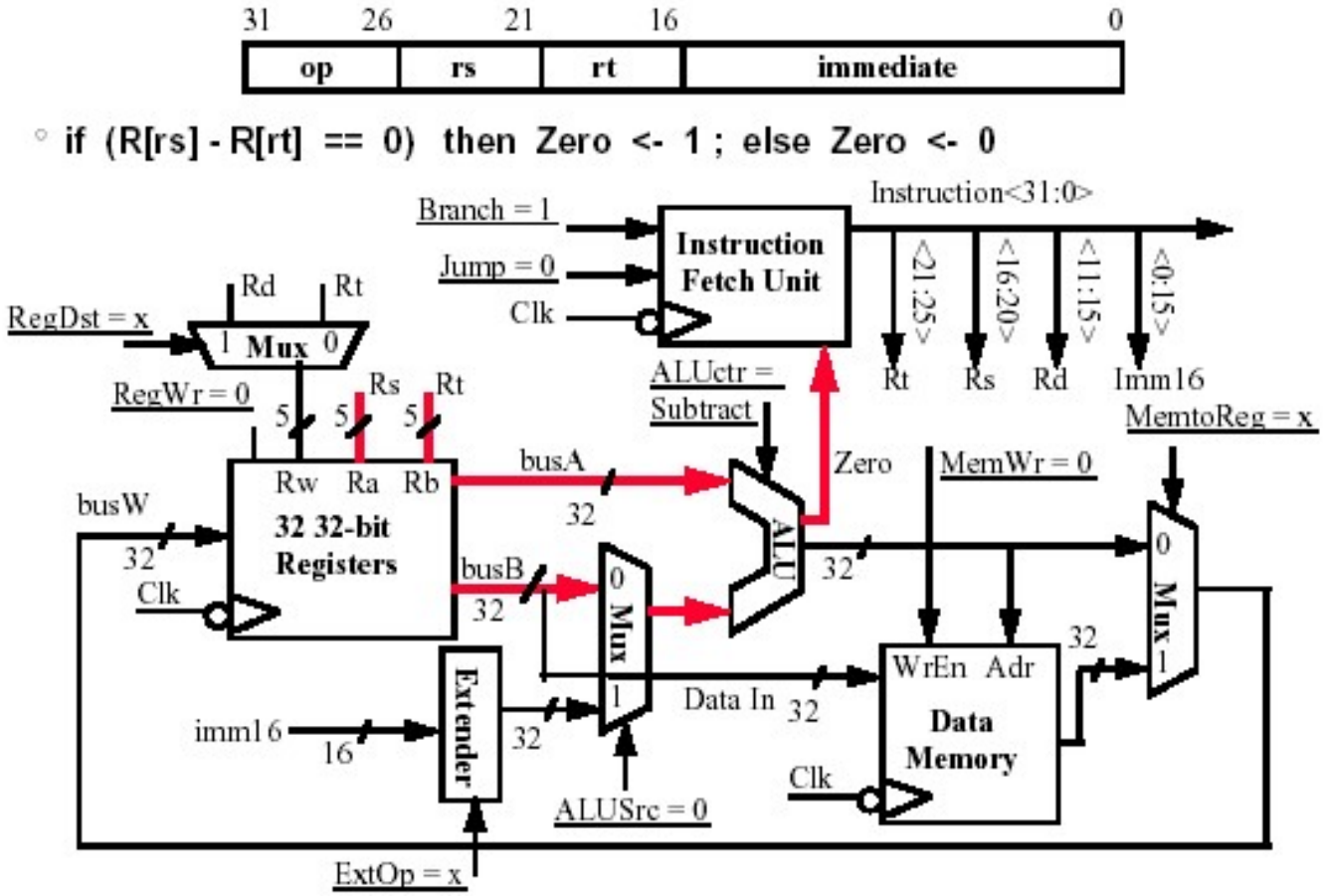
Instrução Load



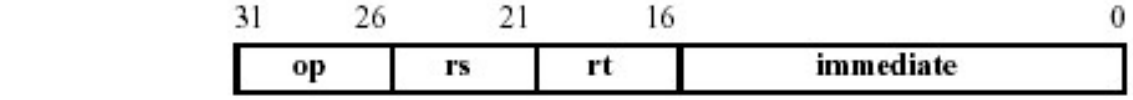
Instrução Store



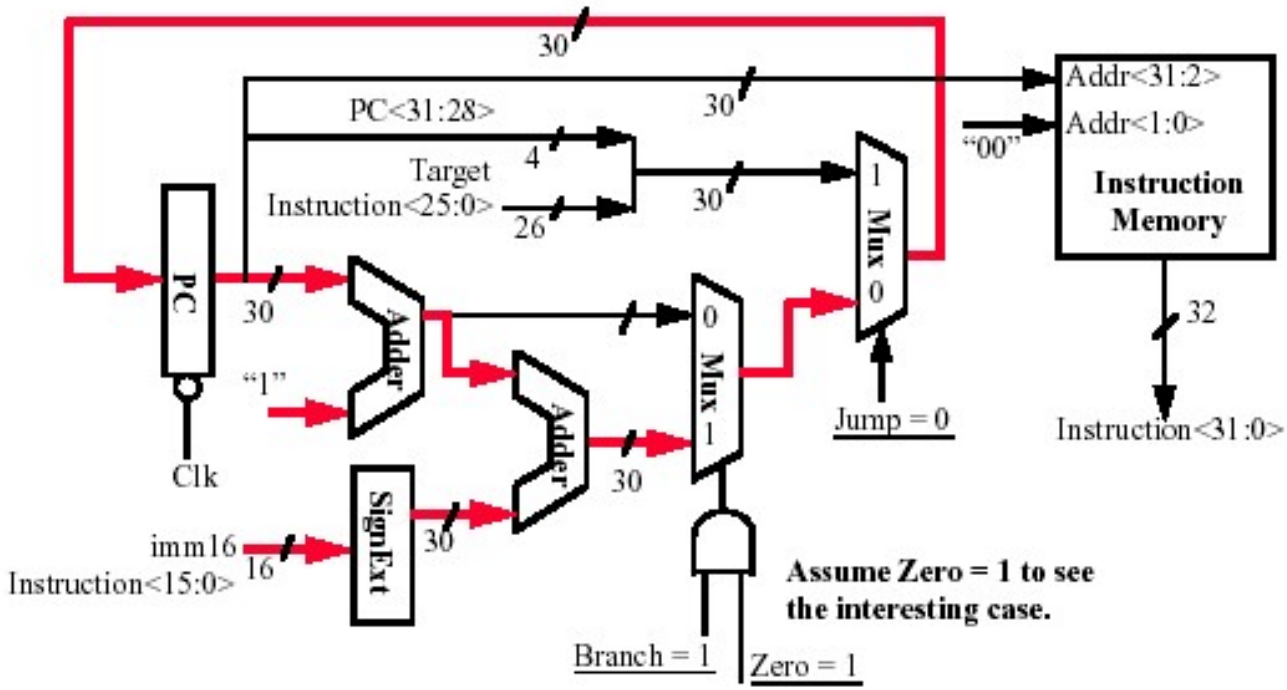
Instrução de Desvio Condicional



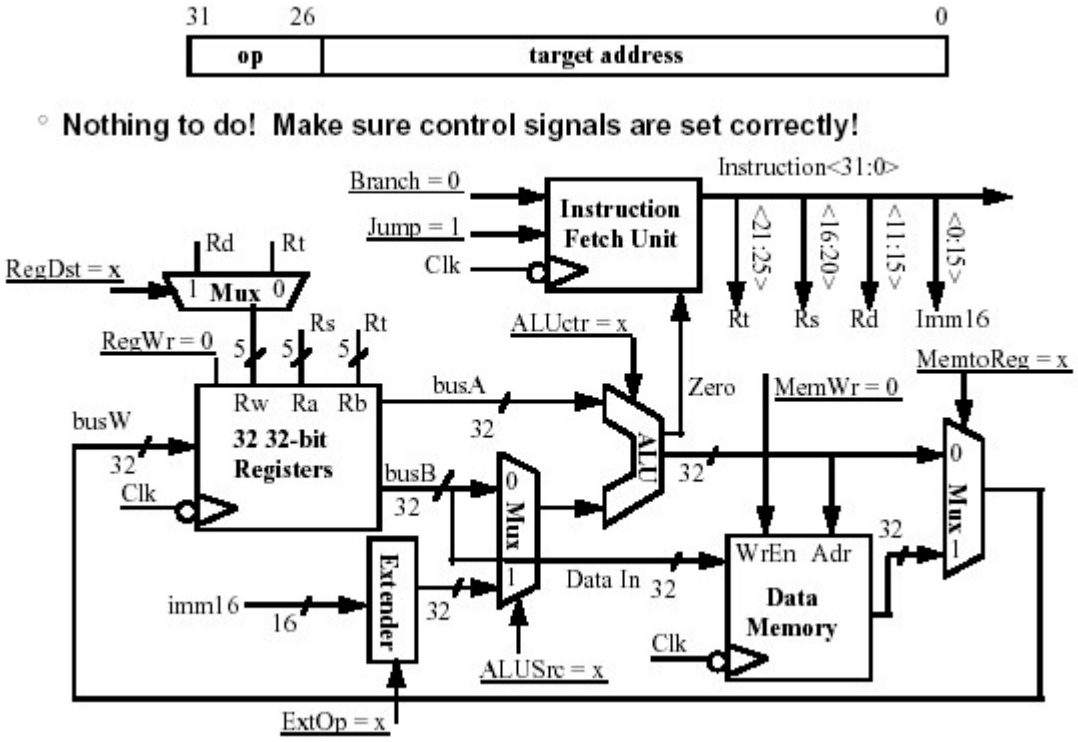
IF ao fim de uma instrução de desvio condicional



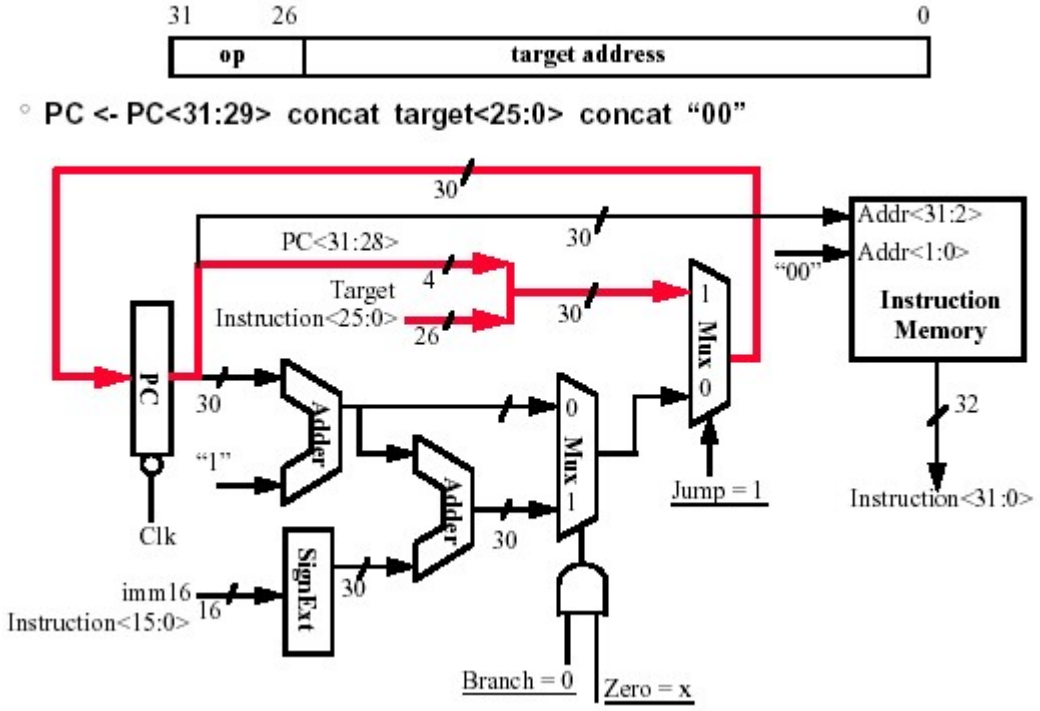
- if (Zero == 1) then PC = PC + 4 + SignExt[imm16]*4 ; else PC = PC + 4



Desvio não condicional

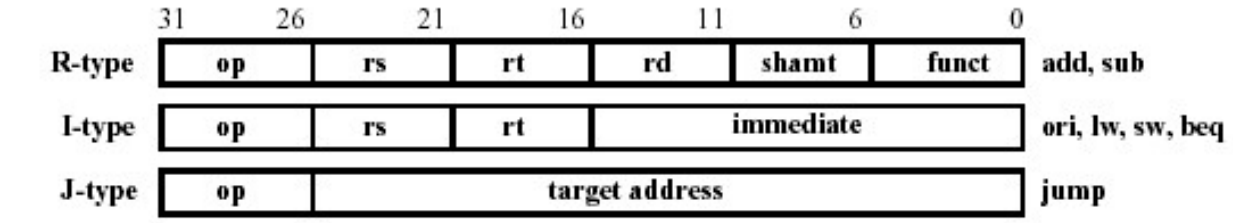


IF ao fim de uma desvio não-condicional



Sumário dos Sinais de Controle

See Appendix A							
func	10 0000	10 0010	We Don't Care :-)				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	add	**sub**	**ori**	**lw**	**sw**	**beq**	**jump**
RegDst	1	0	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
Branch	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract	xxx

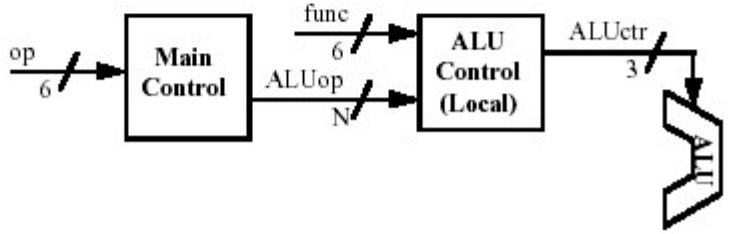


cs 152 control.19

©DAP & SIK 1995

Decodificação Local – Controle da ALU

op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	**ori**	**lw**	**sw**	**beq**	**jump**
RegDst	1	0	0	x	x	x
ALUSrc	0	1	1	1	0	x
MemtoReg	0	0	1	x	x	x
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUop<N:0>	"R-type"	Or	Add	Add	Subtract	xxx



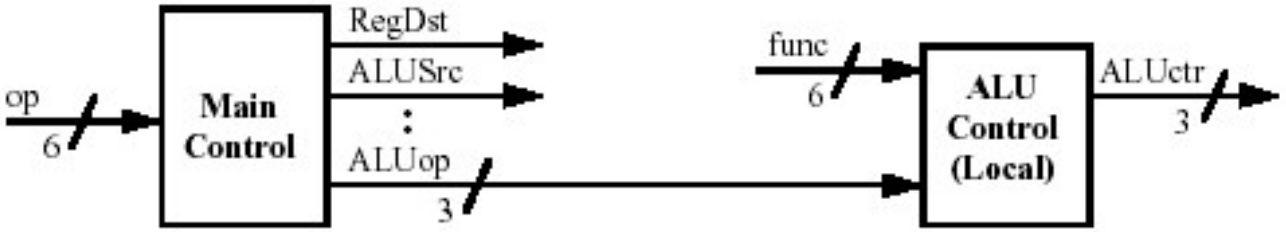
The Truth Table for ALUctr

ALUop (Symbolic)	R-type	ori	lw	sw	beq
ALUop<2:0>	1 00	0 10	0 00	0 00	0 01

funct<3:0>	Instruction Op.
0000	add
0010	subtract
0100	and
0101	or
1010	set-on-less-than

ALUop			func				ALU Operation	ALUctr		
bit<2>	bit<1>	bit<0>	bit<3>	bit<2>	bit<1>	bit<0>		bit<2>	bit<1>	bit<0>
0	0	0	x	x	x	x	Add	0	1	0
0	x	1	x	x	x	x	Subtract	1	1	0
0	1	x	x	x	x	x	Or	0	0	1
1	x	x	0	0	0	0	Add	0	1	0
1	x	x	0	0	1	0	Subtract	1	1	0
1	x	x	0	1	0	0	And	0	0	0
1	x	x	0	1	0	1	Or	0	0	1
1	x	x	1	0	1	0	Set on <	1	1	1

The "Truth Table" for the Main Control



op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	**ori**	**lw**	**sw**	**beq**	**jump**
RegDst	1	0	0	x	x	x
ALUSrc	0	1	1	1	0	x
MemtoReg	0	0	1	x	x	x
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUop (Symbolic)	"R-type"	Or	Add	Add	Subtract	xxx
ALUop <2>	1	0	0	0	0	x
ALUop <1>	0	1	0	0	0	x
ALUop <0>	0	0	0	0	1	x

Como melhorar desempenho ?

•Desempenho é o inverso do tempo...

• **O** tempo de execução de uma instrução é a soma dos tempos gastos pelas unidades funcionais

•O que faz a ULA enquanto se carrega uma nova instrução?

•O que faz a unidade de memória enquanto a ULA realiza uma operação?

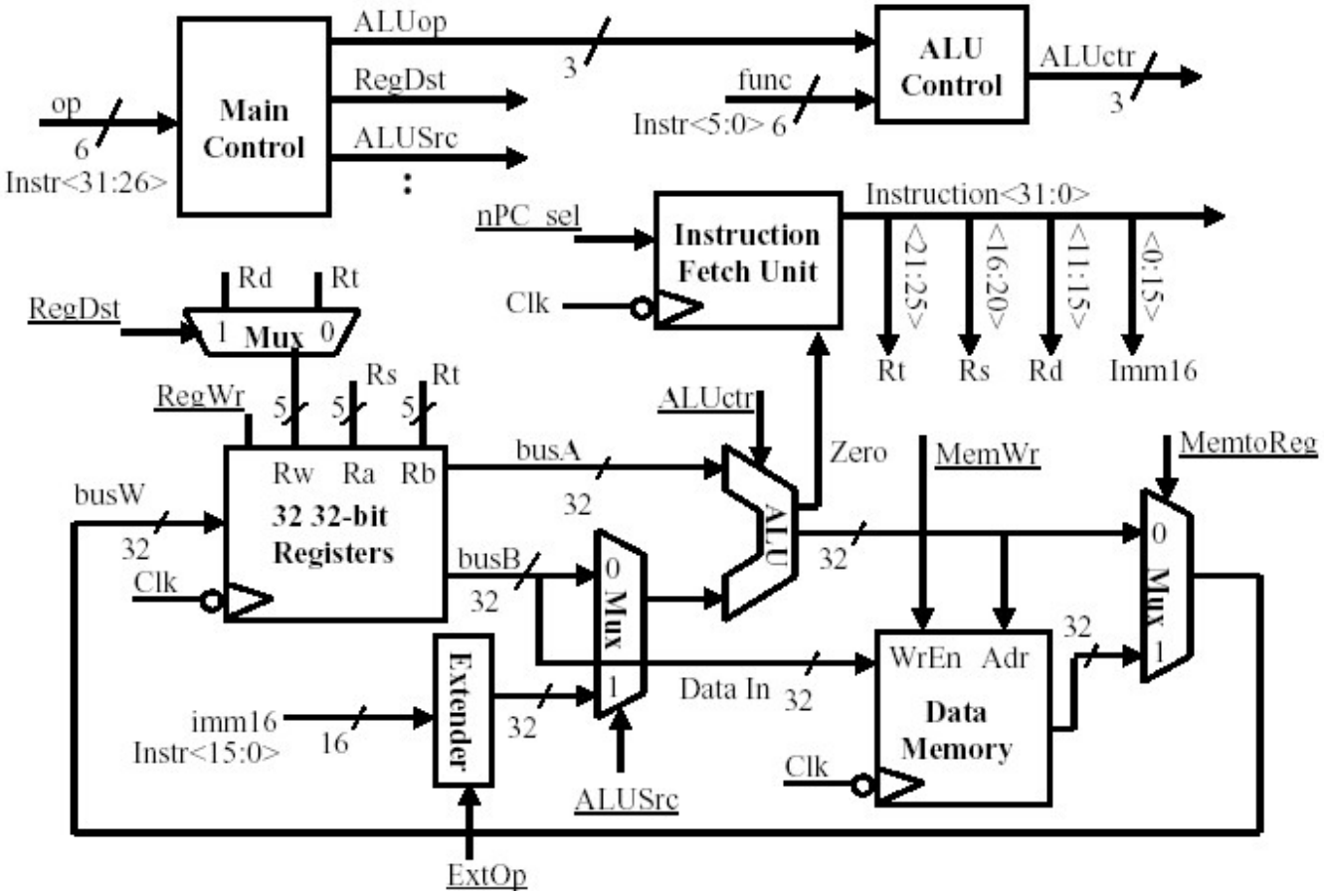
•Idéia: usar o pipeline.....mas é necessário controlar o pipeline...

- O controle será igual?

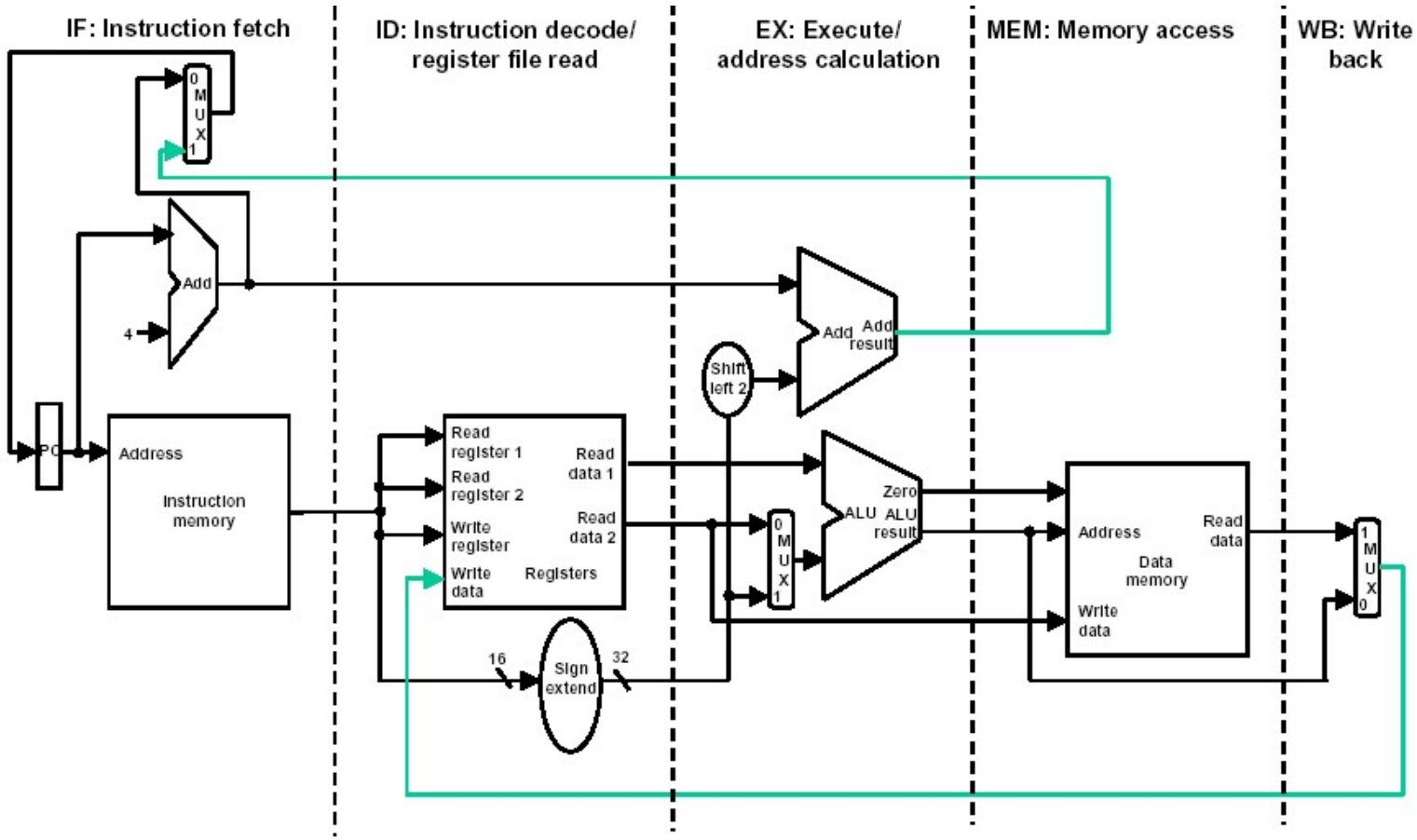
Controle do Processador
Ciclo Único X MIPS com pipeline

- 5 Fases
  - IF, ID/RF, EX, MEM, WB
- Complexidades Adicionais
  - Cada fase executa uma instrução diferente
  - Como desacoplar as fases?
  - Dependências...vamos desconsiderar, por enquanto

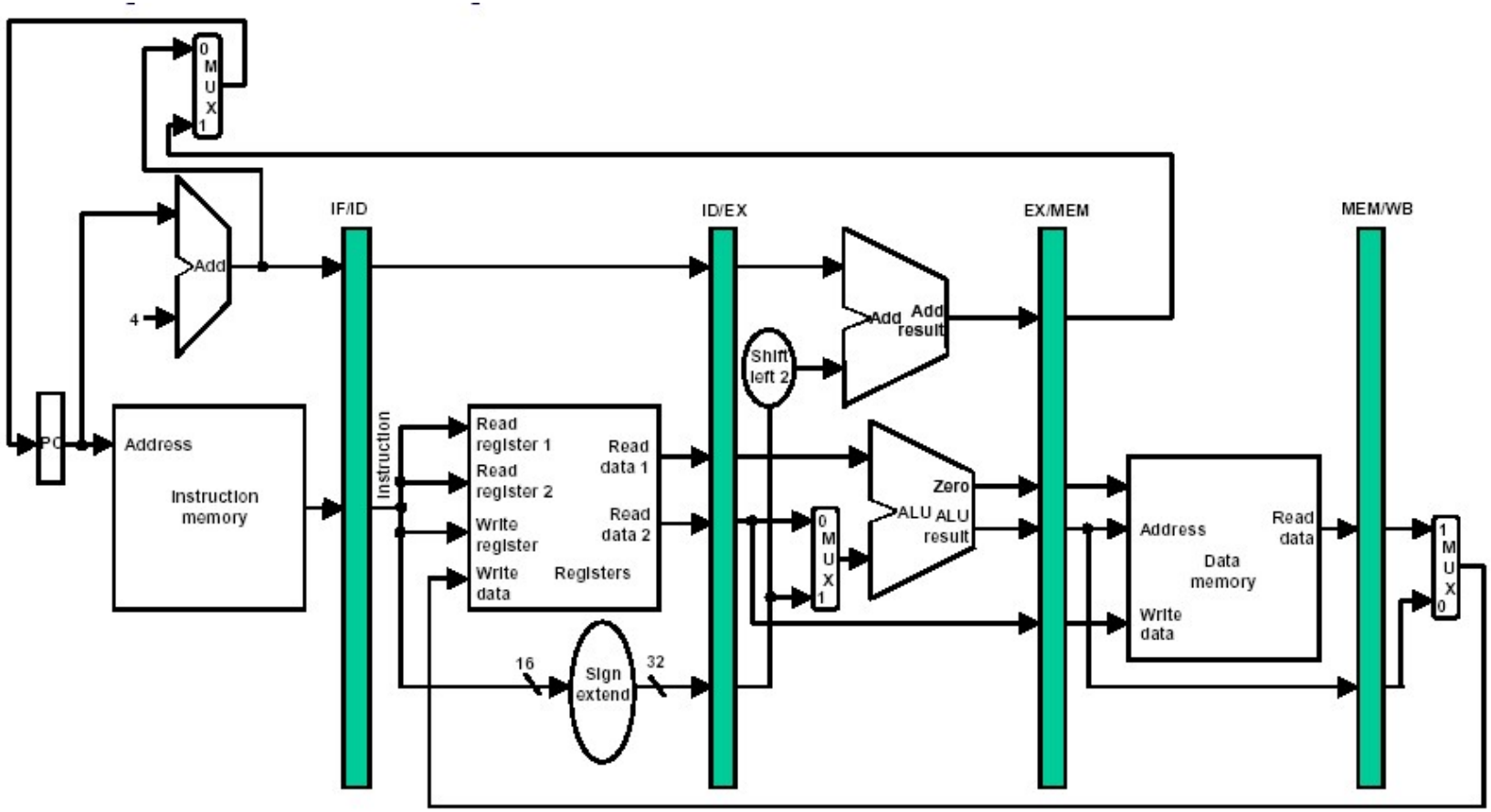
O controle de um Processador de Ciclo Único (Single Cycle)



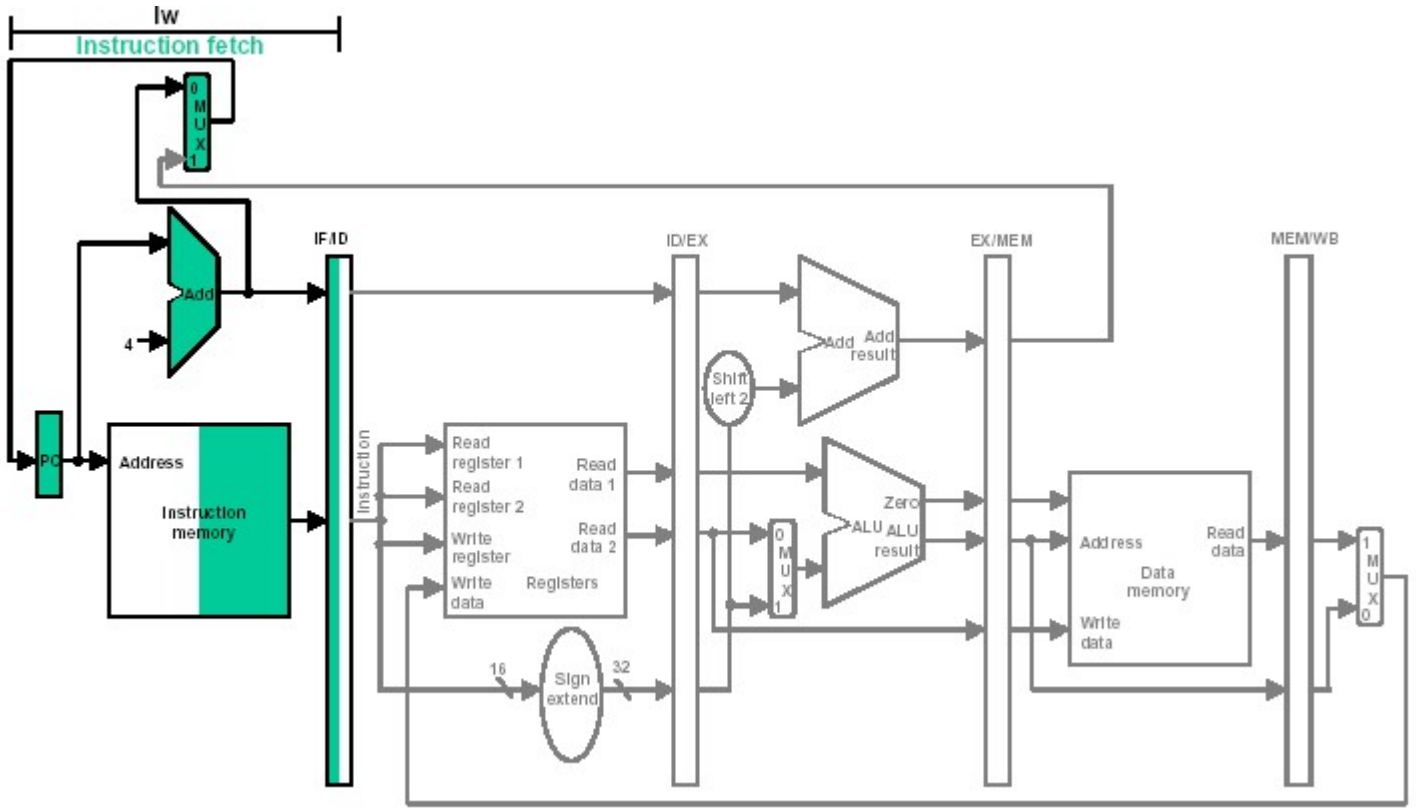
O datapath de um Processador Single Cycle com Fases Definidas



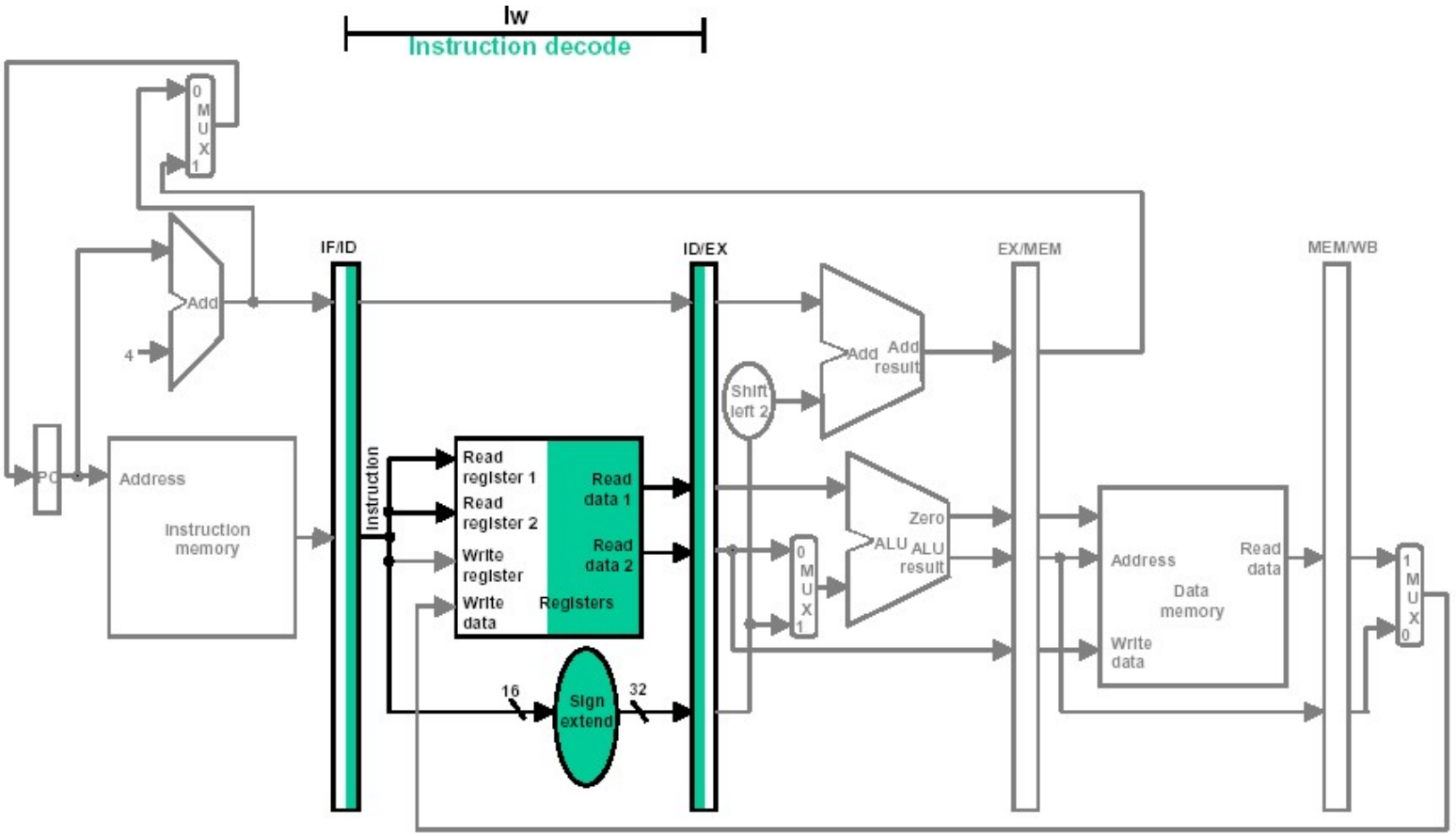
O datapath de um Processador Multicycle



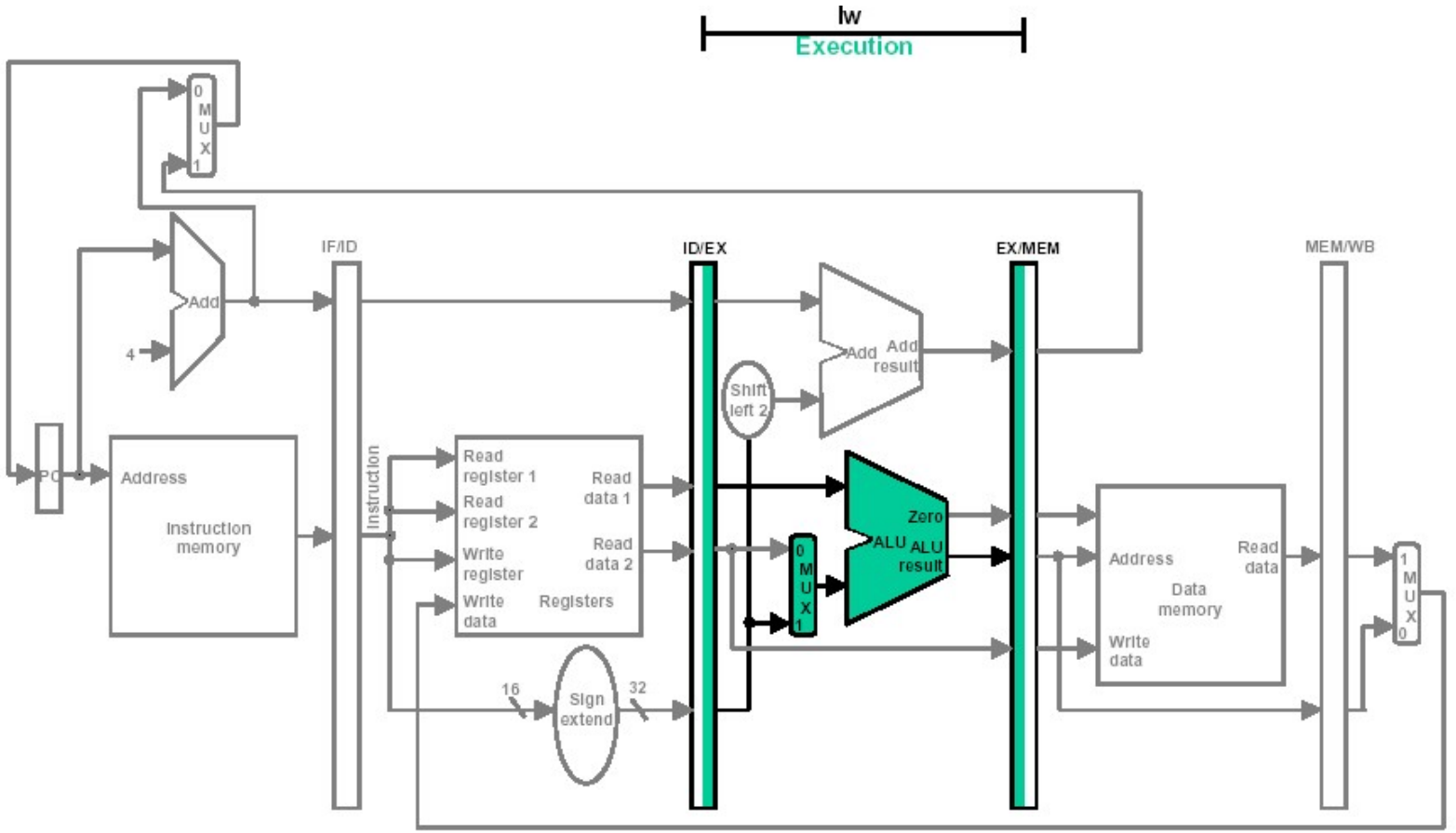
Executando uma Instrução -1



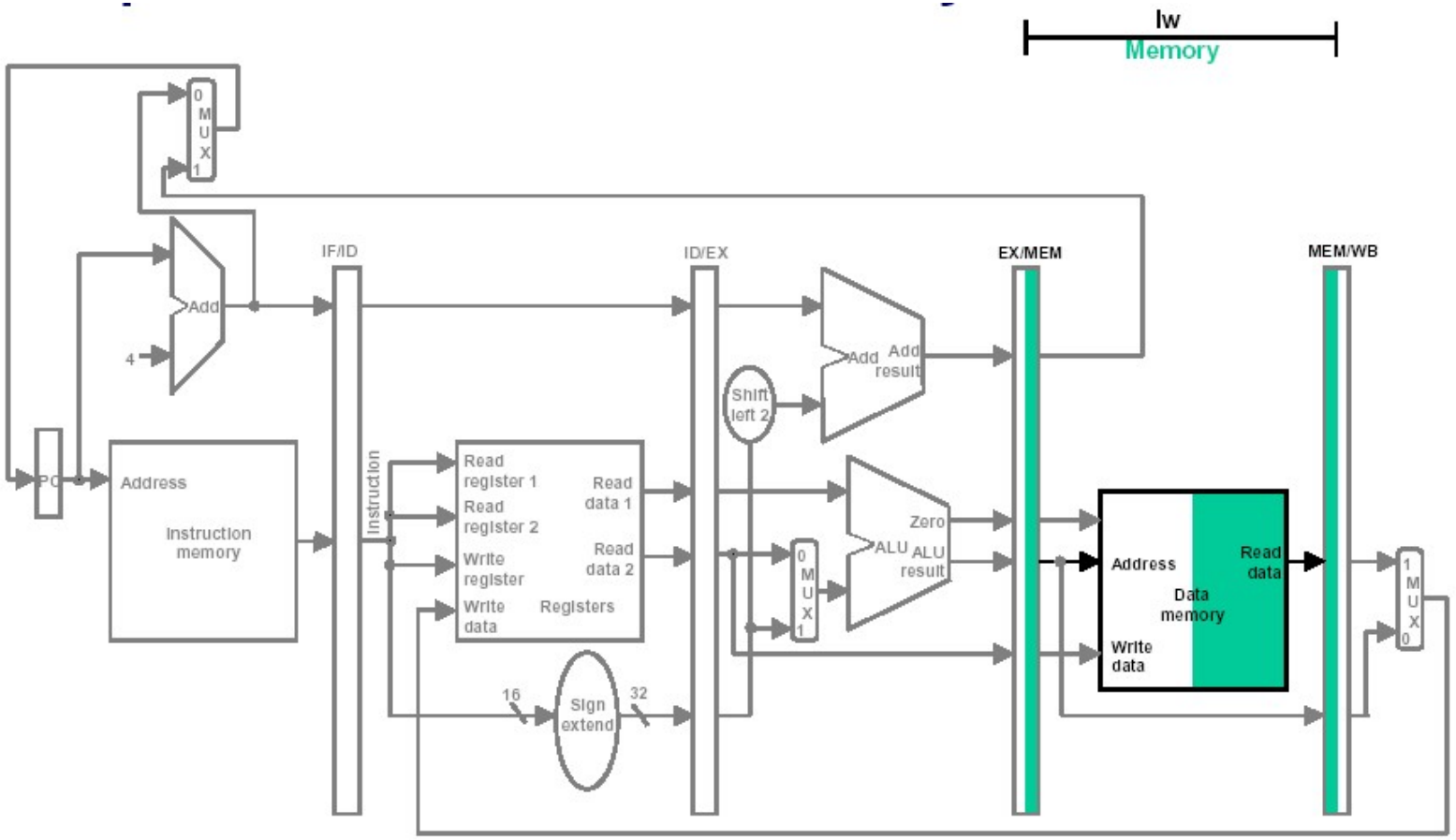
Executando uma Instrução - 2



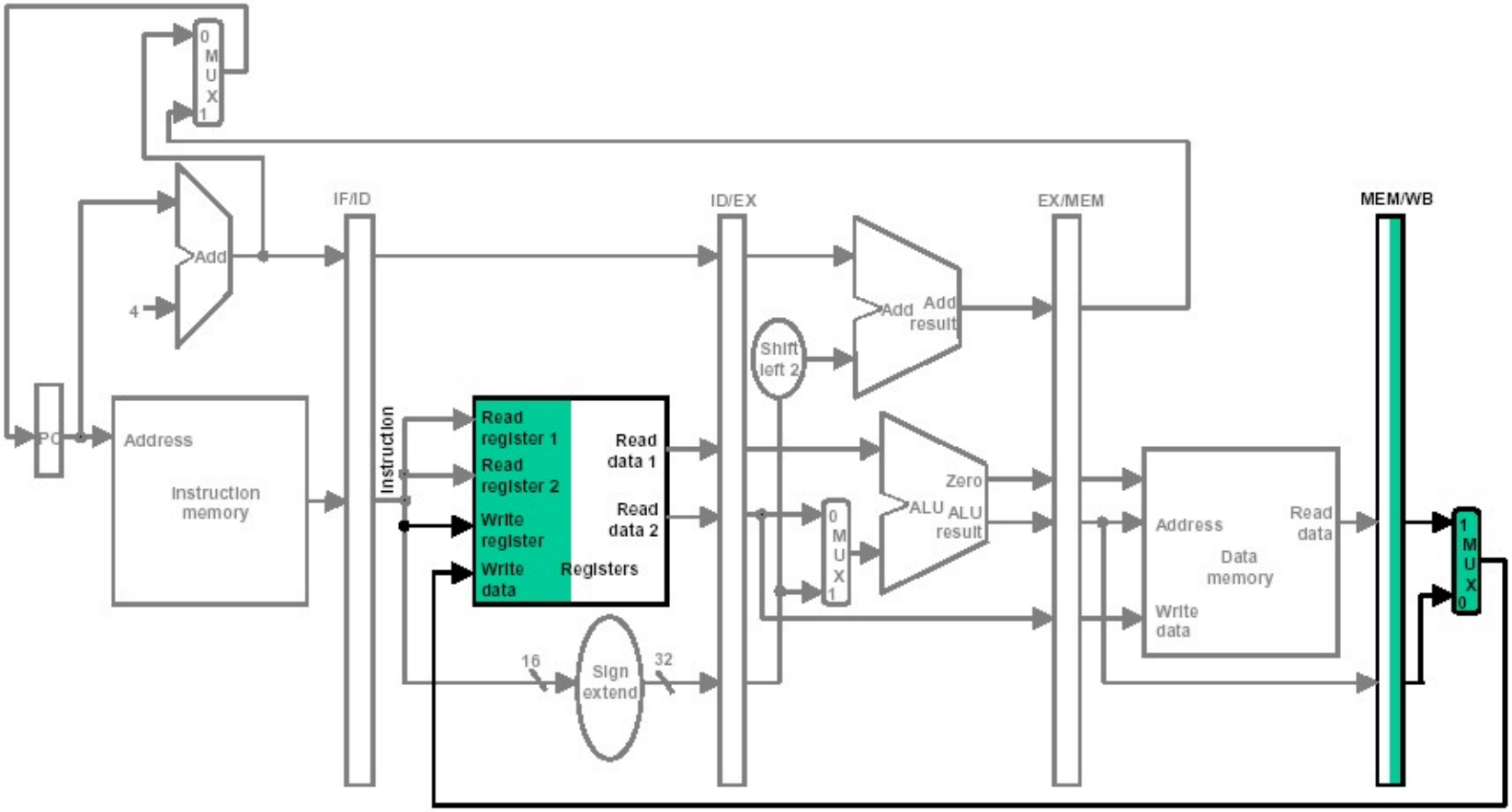
Executando uma Instrução - 3



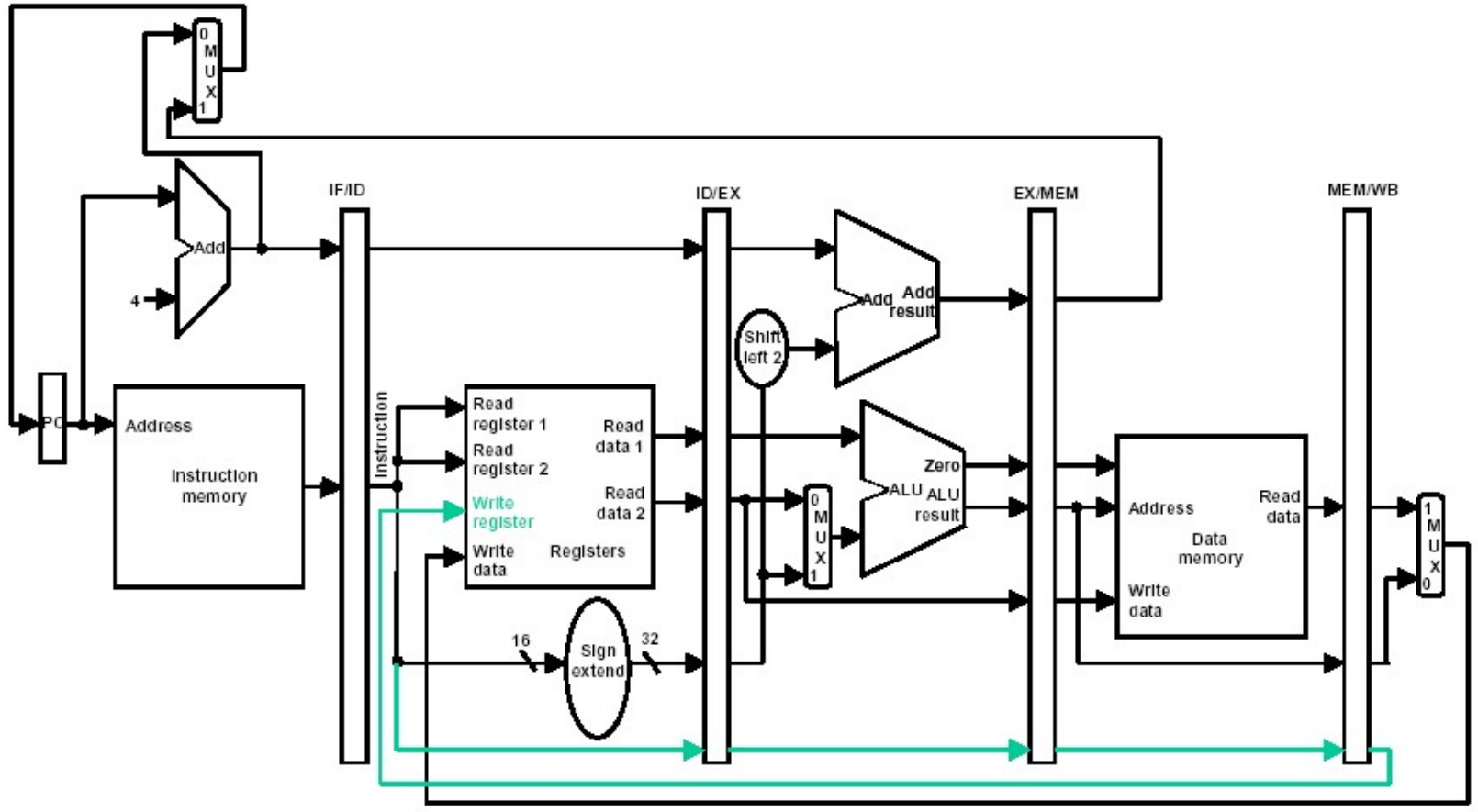
Executando uma Instrução - 4



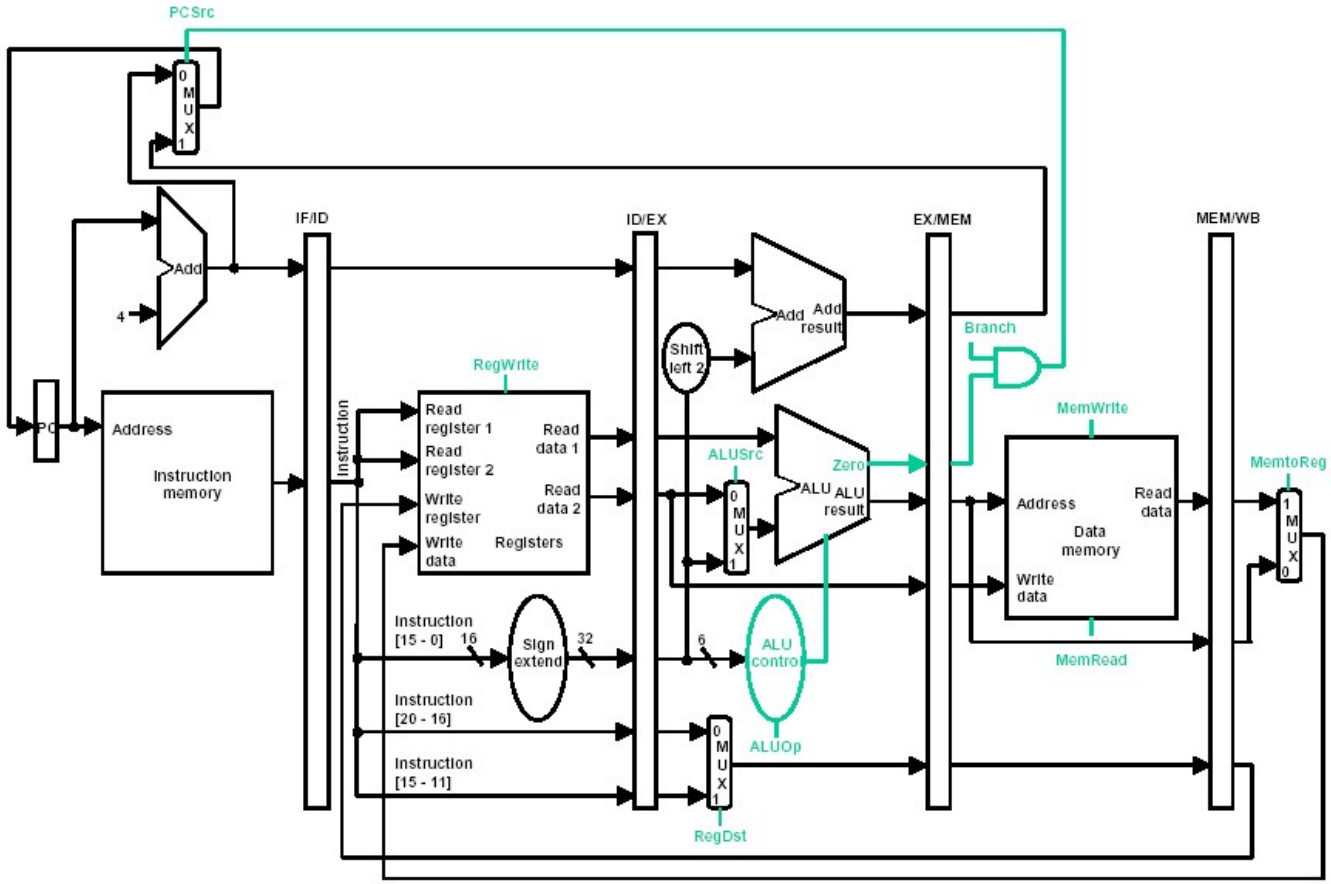
Executando uma Instrução - 5
O que está errado ?



Correção !


M
U
X
0
1
Add
4
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
Add result
Shift left 2
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Zero
ALU
ALU result
Address
Data memory
Read data
Write data
16
Sign extend
32

Controlando um Pipelined Datapath



Valores de controle para os três estágios finais

• Como a maioria dos sinais de controle são usados a partir do estágio EX, podemos cria-los na fase de decodificação(ID)
•Estes controles são então usados no estágio apropriado a medida que a instruções move-se dentro do pipeline.