IC-UNICAMP

MC 602

Circuitos Lógicos e Organização de Computadores

IC/Unicamp

Prof Mario Côrtes

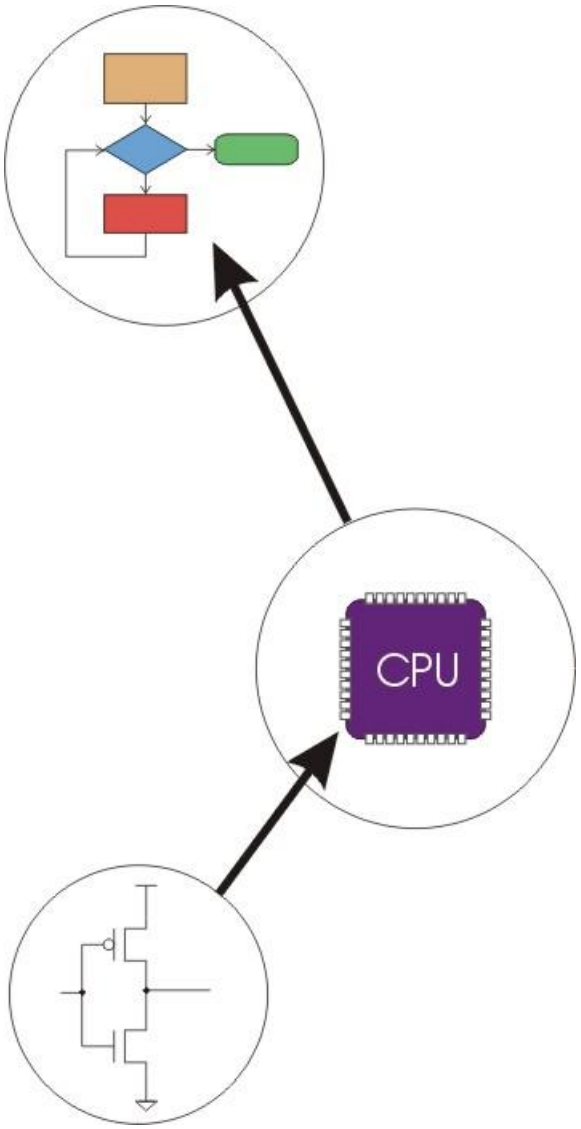
Capítulo MC10

Organização Básica de Processadores

Tópicos

- Níveis de abstração
- Máquina de programa armazenado / Von Neumann / Harvard
- Unidades de um processador: via de dados, controle, banco de processadores, memória
- Visão geral de funcionamento
- ISA: instruction set architecture
- Modos de endereçamento
- Visão introdutória: uma máquina Load / Store
  - ISA
  - Via de dados e controle

Níveis de abstração



Problemas

Algoritmos

Linguagem de alto nível

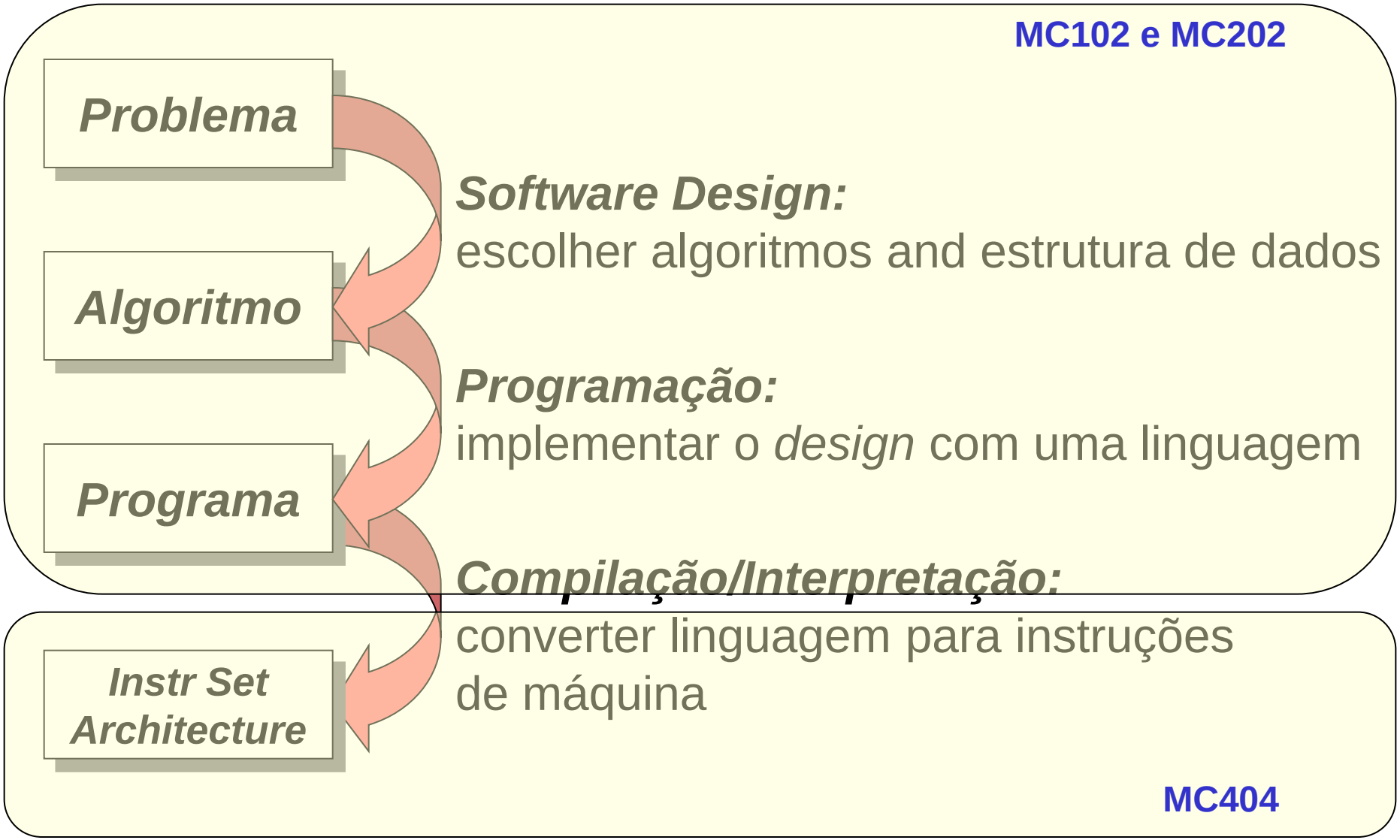
Instruction Set Architecture (ISA)

Microarchitecture

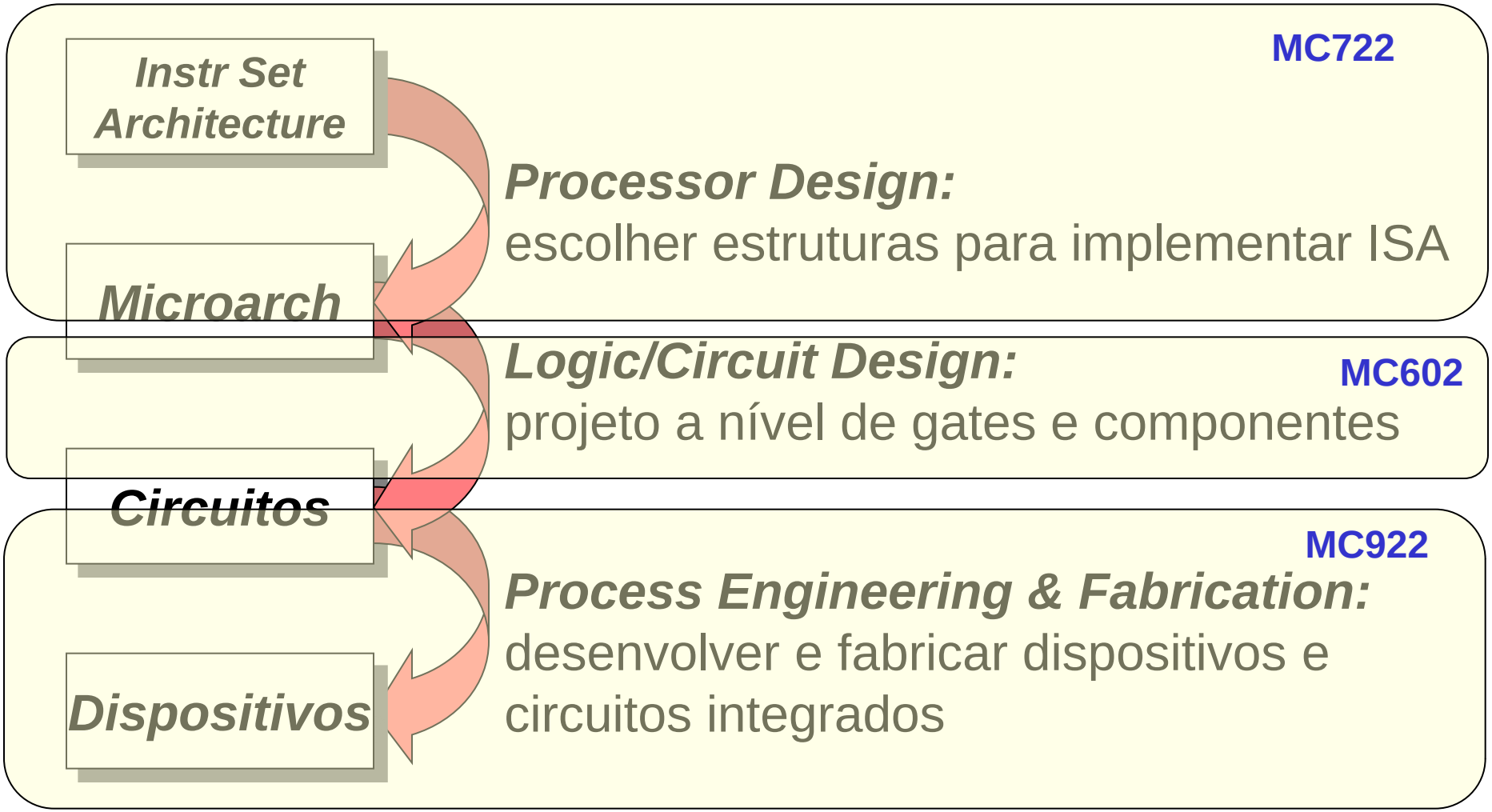
Circuitos

Dispositivos

Níveis de abstração



Níveis de abstração . . . .



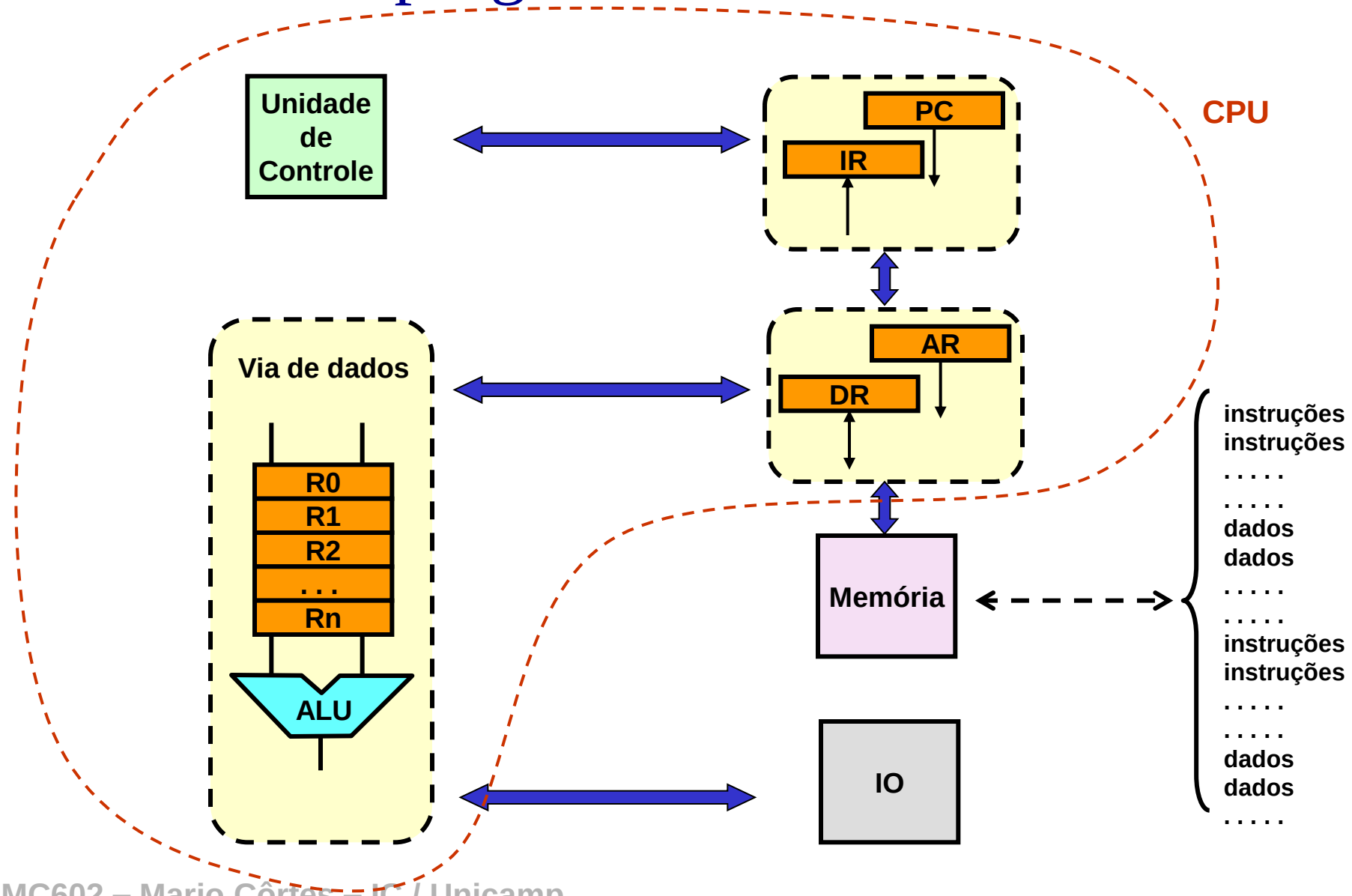
IC-UNICAMP

Objetivos deste capítulo

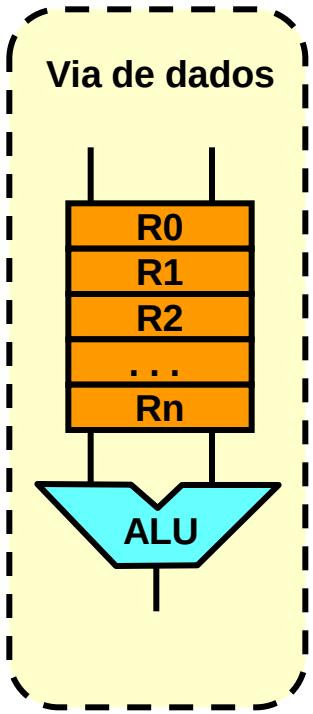
- Dado o conhecimento do projeto de estruturas: lógica combinacional, lógica sequencial, memórias, FSM
- Introdução:
  - como combinar essas estruturas para construir um processador

IC-UNICAMP

Von Neumann: computador de programa armazenado

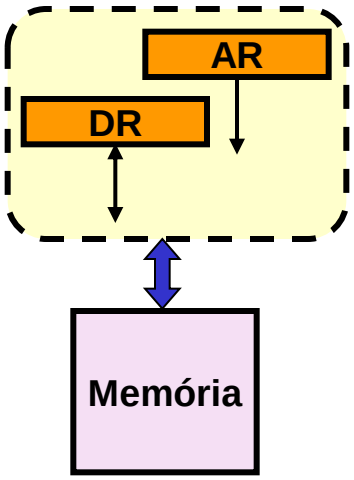


Via de dados



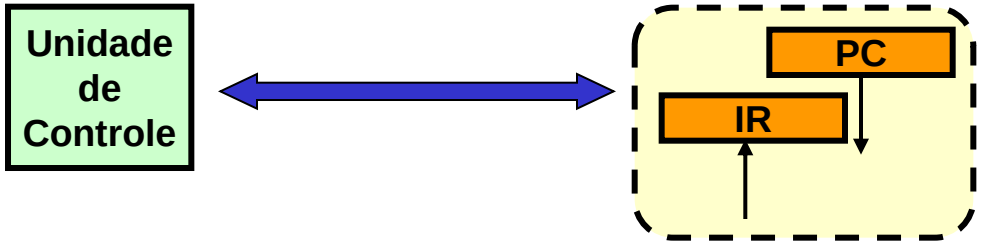
- Unidade responsável pela manipulação dos dados
- Banco de registradores
  - trocam dados com a memória: load (Mem→Reg) e store (Reg→Mem)
  - operandos da ALU
  - armazenam resultados da ALU
- ALU
  - operações lógicas e aritméticas
- Barramentos
  - tipicamente 2 para os operandos
  - potencialmente 3: 2 operandos + 1 resultado
  - define latência (nº de ciclos) para realizar operações

Memória



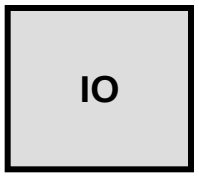
- Na arquitetura de Von Neumann: contém dados e o programa armazenado
- AR: Address Register
- DR: Data Register
- Sinais de controle mais importantes:
  - RD e WR
- Na leitura / escrita de dados:
  - interação com a via de dados através de DR
- Na leitura de instrução:
  - interação com a unidade de controle (para decodificação da instrução) através de PC (Program Counter) e IR (Instruction Register)

Unidade de controle



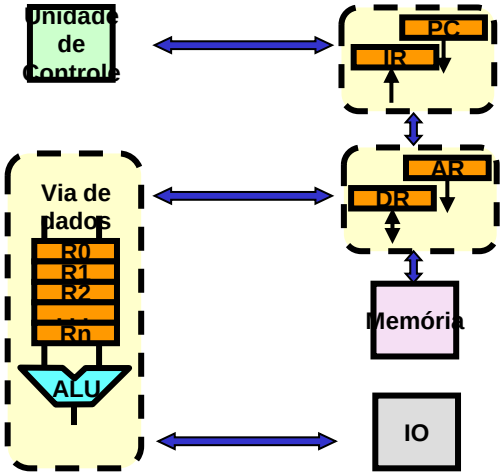
- PC (Program Counter): contem o endereço da instrução sendo executada (lida mais recentemente)
- IR (Instruction Register): contém a instrução sendo executada
  - IR e PC podem ser enviados diretamente para a memória ou PC→AR e DR→IR
- Unidade de controle: normalmente uma FSM
  - decodifica IR e gera sinais de controle para as unidades
  - define valor de PC: incrementa ou salta
  - exemplos de sinais de controle: carga e endereço de registradores, RD / WR da memória, função da ALU, controle de fluxo de execução

Entrada e saída: IO



- Mostrado apenas genericamente
- Interface da CPU com periféricos
  - de entrada: discos, drives de USB, teclado, mouse, audio in
  - de saída: impressora, monitor, audio out
- Normalmente feita via barramento de I/O onde controladores dos dispositivos são conectados
- A ser visto em MC722

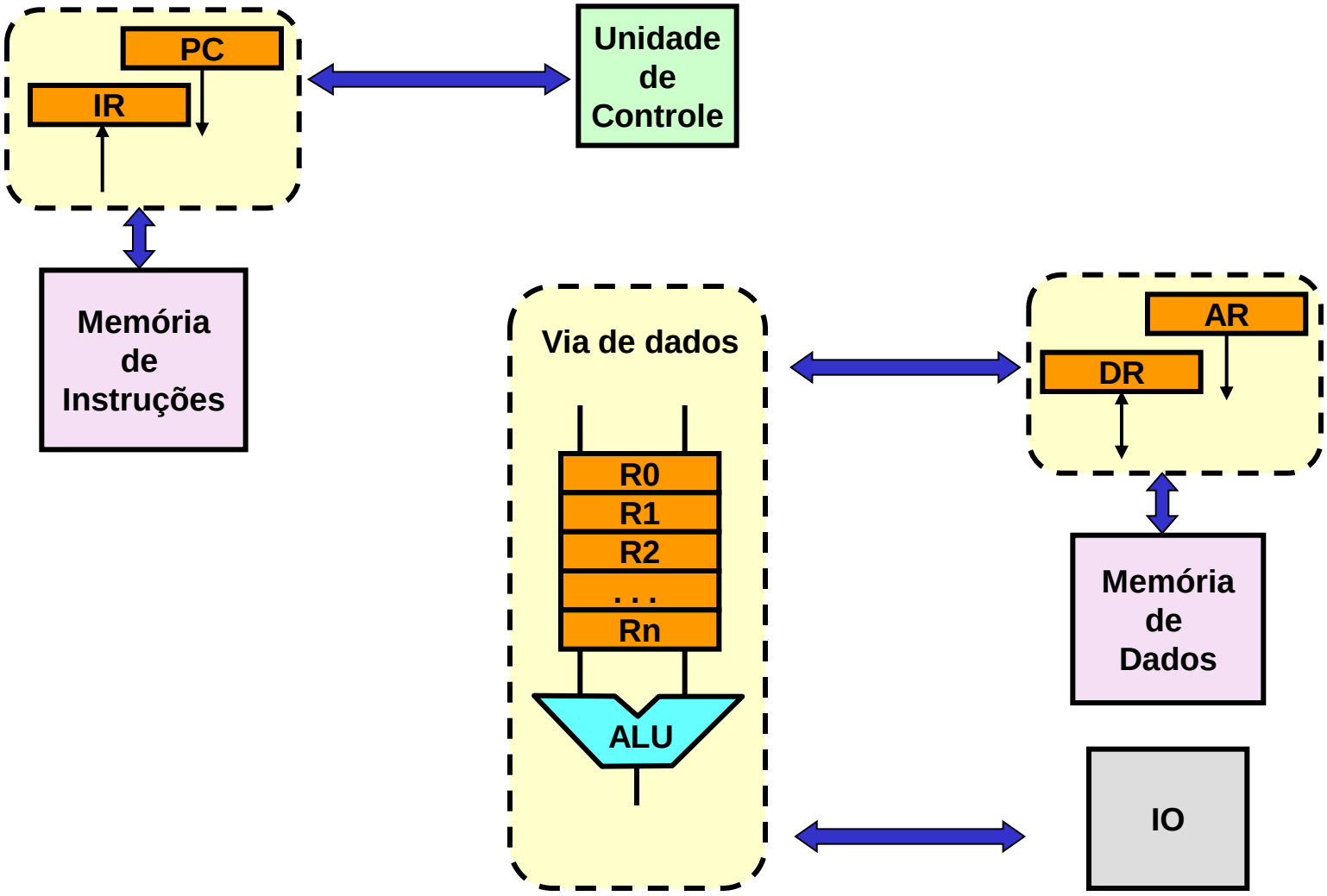
Interconexão



- Na figura, genericamente
- Normalmente há um ou mais barramentos
- Incomum ligação direta, exceto em casos de comunicação dedicada
- Organização de interconexão varia muito

IC-UNICAMP

Arquitetura de Harvard: memórias dedicadas DM e IM



Visão geral do funcionamento

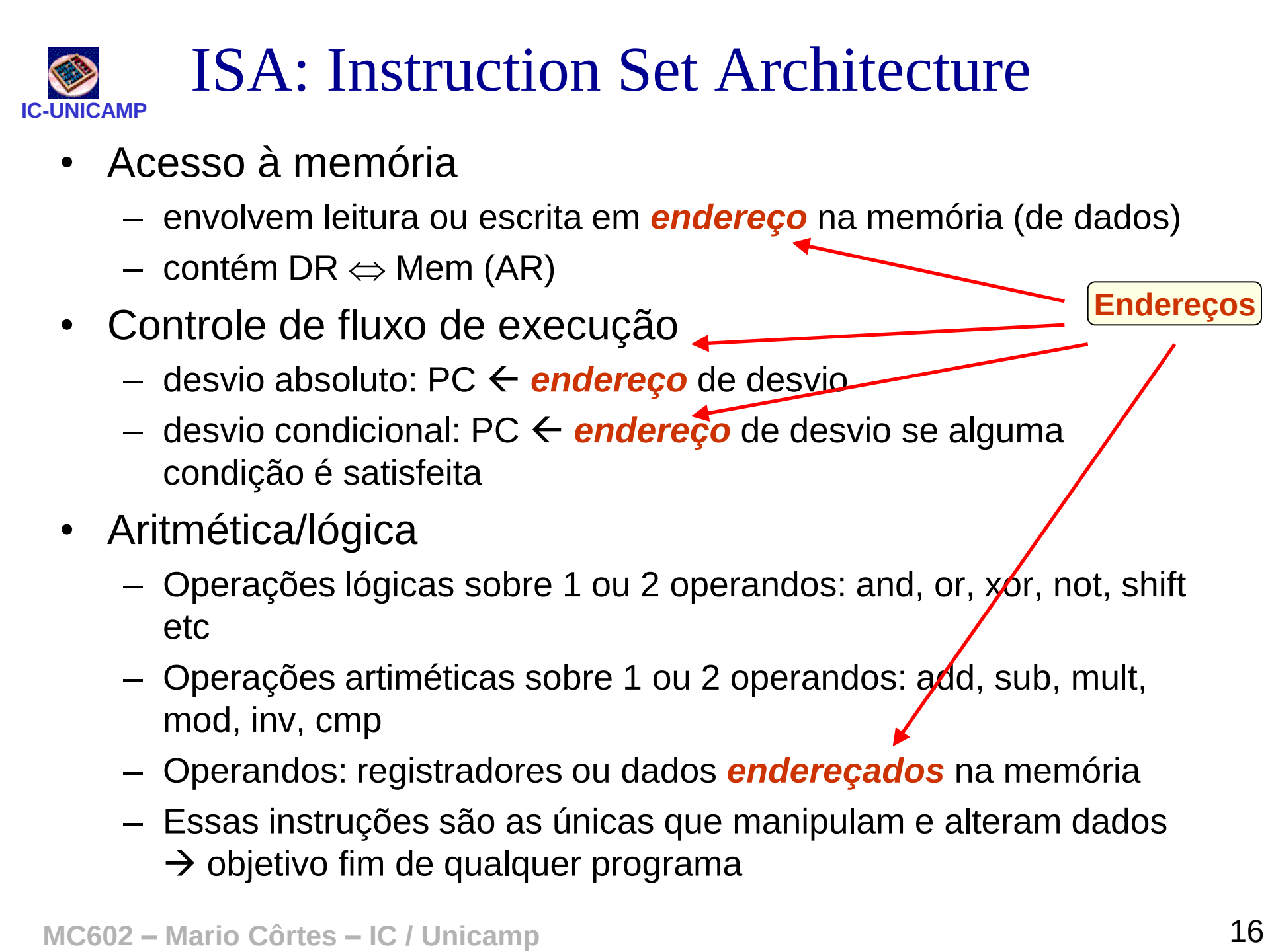
- Programa armazenado na memória (dados+instruções ou instruções)
- Instruções: linguagem de máquina (binário) especificam tipo de instrução (OPcode), endereços, condições e operandos
- Passos
  - Fetch: instrução é lida da memória; IR ←Mem(PC)
    - 1 ou mais ciclos (dependa da velocidade da memória)
  - Decodificação: análise pela unidade de controle
    - 1 ciclo
  - Execução: realização do que está especificado na instrução
    - 1 ou mais ciclos
- Resultado da execução de um programa
  - conteúdo alterado da memória
  - interações com dispositivos de I/O

ISA: Instruction Set Architecture

- Acesso à memória
  - envolvem leitura ou escrita em endereço na memória (de dados)
  - contém DR ⇔ Mem (AR)
- Controle de fluxo de execução
  - desvio absoluto: PC ← endereço de desvio
  - desvio condicional: PC ← endereço de desvio se alguma condição é satisfeita
- Aritmética/lógica
  - Operações lógicas sobre 1 ou 2 operandos: and, or, xor, not, shift etc
  - Operações artiméticas sobre 1 ou 2 operandos: add, sub, mult, mod, inv, cmp
  - Operandos: registradores ou dados endereçados na memória
  - Essas instruções são as únicas que manipulam e alteram dados → objetivo fim de qualquer programa

ISA: Instruction Set Architecture

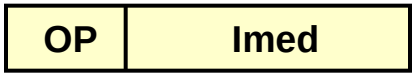
- Acesso à memória
  - envolvem leitura ou escrita em ***endereço*** na memória (de dados)
  - contém DR ⇔ Mem (AR)
- Controle de fluxo de execução
  - desvio absoluto: PC ← ***endereço*** de desvio
  - desvio condicional: PC ← ***endereço*** de desvio se alguma condição é satisfeita
- Aritmética/lógica
  - Operações lógicas sobre 1 ou 2 operandos: and, or, xor, not, shift etc
  - Operações artiméticas sobre 1 ou 2 operandos: add, sub, mult, mod, inv, cmp
  - Operandos: registradores ou dados ***endereçados*** na memória
  - Essas instruções são as únicas que manipulam e alteram dados → objetivo fim de qualquer programa



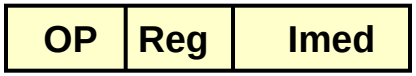
Endereçamento de dados (D) e instruções (I)

IC-UNICAMP

- imediato (D I):
  - na instrução



- indexado (D):
  - registrador + imediato



- direto (D):
  - na instrução

- indireto: armazenado
  - na memória (D)

  - ou em registrador (I)

- relativo (I):
  - imediato + PC

- Em geral: imediato pode ser + ou -

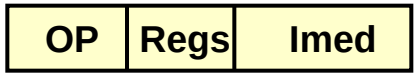
Instruções de acesso à memória

- Leitura ou escrita
  - DR ⇔ Mem (AR)

- Endereçamento
  - direto: AR ⇔ IR_{End}

    - ex: ld R1 End # R1 ← Mem(IR_{End})

  - indexado: AR ⇔ Reg + IR_{End}



    - ex: ld_ind R1, R2, End # R1 ← Mem(R2 + IR_{End})
    - útil para acesso a vetores (loops)

Instruções de controle de fluxo

- Controle de fluxo de execução
  - desvio absoluto: PC ← endereço de desvio
  - desvio condicional: PC ← endereço de desvio se alguma condição é satisfeita
- Endereçamento
  - direto (ou imediato): PC ← IR_{End}
  - relativo: PC ← IR_{Imed} + PC
  - indireto reg: PC ← Reg
- Condições de desvio
  - resultado de operação da ULA: C, V, Z, N
  - registrador de status: guarda situação de execução passada
  - teste de outros registradores

Instruções aritméticas e lógicas

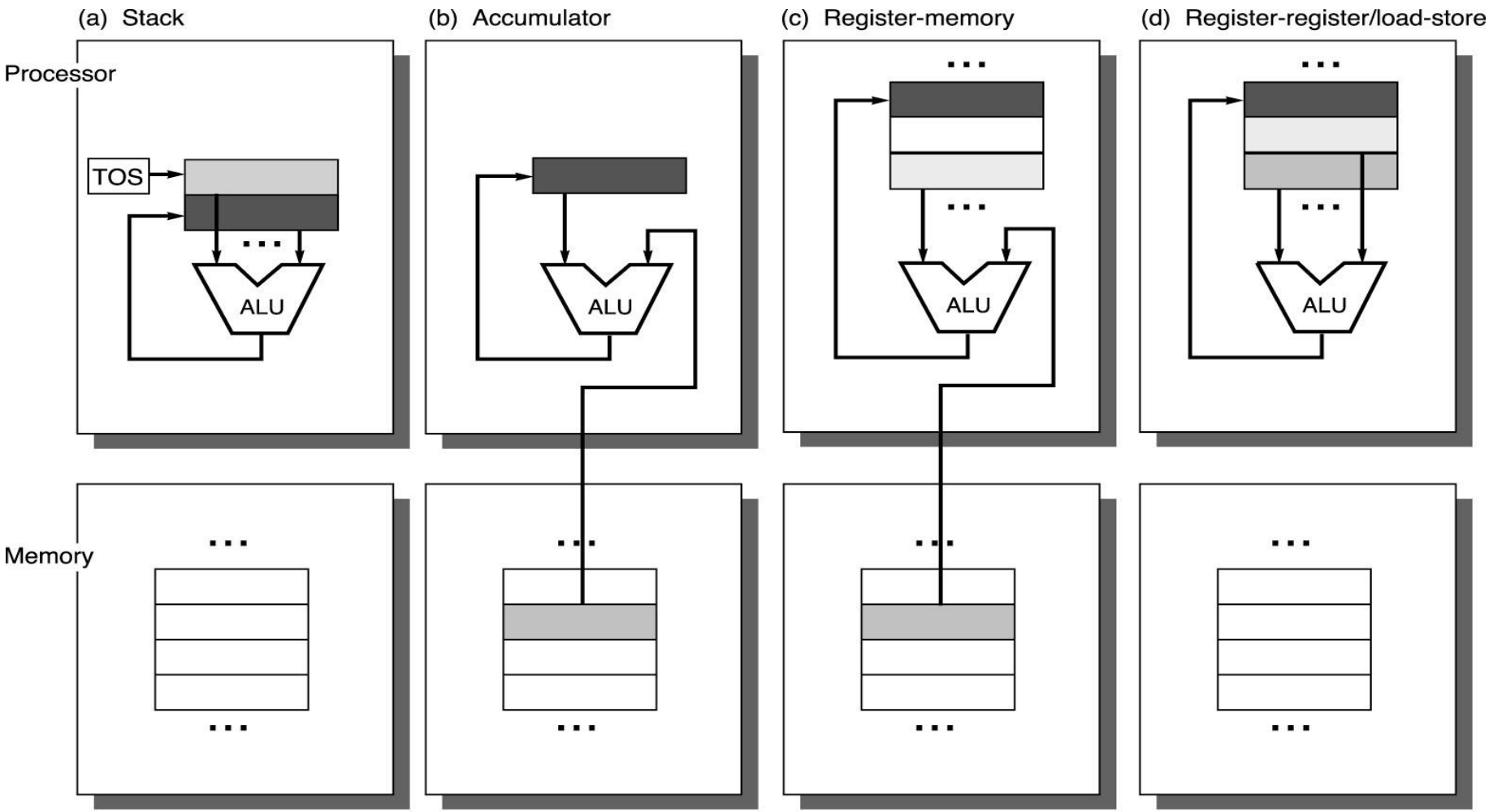
- Dois operandos: res ← Oper(op1, op2)
- Um operando: res ← Oper(op1)
- Operandos: registradores ou dados endereçados na memória
- Máquinas de 3 endereços:
  - Mem(end1) ← Mem(end2) Oper Mem(end3)
  - precisa 3 acessos à memória
- Máquinas de 2 endereços:
  - Mem(end1) ← Reg Oper Mem(end2)
  - Reg de apoio = acumulador = Acc
- Máquinas de 1 endereço
  - Acc ← Acc Oper Mem(end1)

IC-UNICAMP

Instruções aritméticas e lógicas (cont)

- Máquinas load / store
  - R1 ← Ri Oper Rj
  - ULA não opera diremente sobre dados da memória
  - típico dos computadores RISC (todos atuais)
  - implica na necessidade de banco de registradores
  - é o que será visto em MC602 e MC722

Tipos de Máquinas (operandos da ULA)



IC-UNICAMP

Exemplo de uma máquina ld/st: MIPS

- Instruções e dados: palavras tem 32 bits
- Endereçamento à memória byte a byte.

- 3 formatos de instruções

R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	16 bit address		
J	op	26 bit address				

Instruções MIPS: tipo R

- R-type: *Register-type*
  - 3 operandos registradores:
    - `rs`, `rt`: source registers
    - `rd`: destination register
  - Outros campos:
    - `op`: código da *operação ou opcode*
    - `funct`: função

      opcode + função definem operação
    - `shamt`: a quantidade de *shift para instruções de deslocamento*

R-Type

Instruções MIPS: exemplos tipo R

Assembly Code	op	rs	rt	rd	shamt	funct	
	Field Values						
`add $s0, $s1, $s2`	0	17	18	16	0	32	`$s0 ← $s1 + $s2`
`sub $t0, $t3, $t5`	0	11	13	8	0	34	`$t0 ← $t3 - $t5`
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Machine Code

op	rs	rt	rd	shamt	funct	
000000	10001	10010	10000	00000	100000	(0x02328020)
000000	01011	01101	01000	00000	100010	(0x016D4022)
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Nota: a ordem dos registradores no código assembly:

```
add rd, rs, rt
```

Instruções MIPS: tipo I

- I-Type: Immediate-Type
  - 3 operands:
    - `rs`, `rt`: register operands
    - `imm`: 16-bit em complemento de dois immediate
  - Outros campos:
    - `op`: opcode

I-Type

Instruções MIPS: exemplos tipo I

Assembly Code	op	rs	rt	imm	
`addi $s0, $s1, 5`	8	17	16	5	`$s0 ← $s1 + 5`
`addi $t0, $s3, -12`	8	19	8	-12	`$t0 ← $s3 + (-12)`
`lw $t2, 32($0)`	35	0	10	32	`$t2 ← Mem($s0 + 32)`
`sw $s1, 4($t1)`	43	9	17	4	`Mem($t1 + 4) ← $s1`
	6 bits	5 bits	5 bits	16 bits	

Field Values

Nota: a ordem dos registradores no código assembly:

```
addi rt, rs, imm
lw   rt, imm(rs)
sw   rt, imm(rs)
```

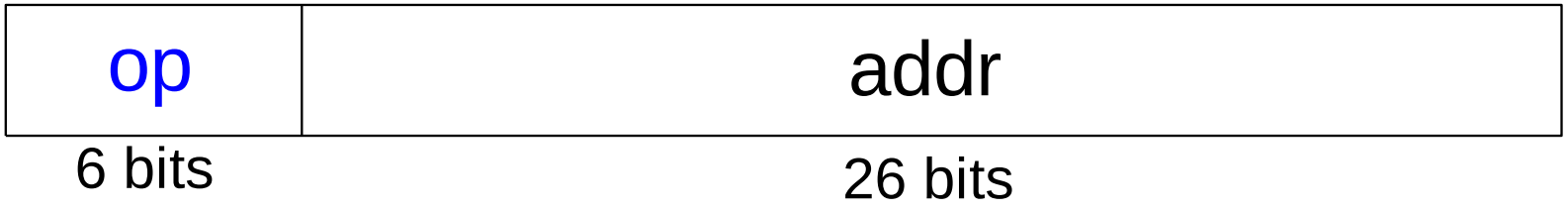
Machine Code

op	rs	rt	imm	
001000	10001	10000	0000 0000 0000 0101	(0x22300005)
001000	10011	01000	1111 1111 1111 0100	(0x2268FFF4)
100011	00000	01010	0000 0000 0010 0000	(0x8C0A0020)
101011	01001	10001	0000 0000 0000 0100	(0xAD310004)
6 bits	5 bits	5 bits	16 bits	

Instruções MIPS: tipo J

- J-Type: Jump-Type
  - 26-bit address operand (`addr`)
  - Usado nas instruções jump (`j`)

J-Type



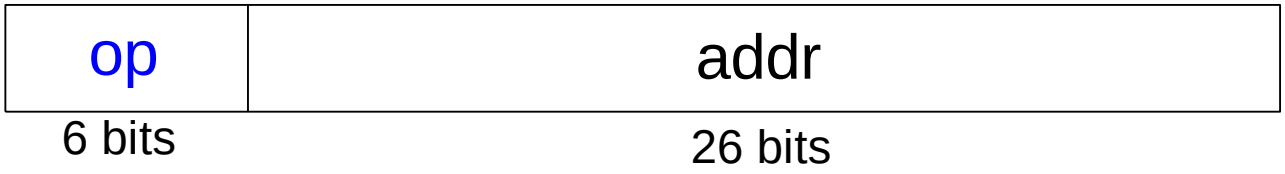
Instruções MIPS

- Formatos das Instruções

R-Type

I-Type

J-Type



IC-UNICAMP

MIPS: resumo

MIPS operands

Name	Example	Comments
32 registers	`$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra, $at`	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zero always equals 0. Register $at is reserved for the assembler to handle large constants.
2^{30} memory words	Memory[0], Memory[4], ..., Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1, $s2, $s3`	`$s1 = $s2 + $s3`	Three operands; data in registers
	subtract	`sub $s1, $s2, $s3`	`$s1 = $s2 - $s3`	Three operands; data in registers
	add immediate	`addi $s1, $s2, 100`	`$s1 = $s2 + 100`	Used to add constants
Data transfer	load word	`lw $s1, 100($s2)`	`$s1` = Memory[`$s2` + 100]	Word from memory to register
	store word	`sw $s1, 100($s2)`	Memory[`$s2` + 100] = $s1	Word from register to memory
	load byte	`lb $s1, 100($s2)`	`$s1` = Memory[`$s2` + 100]	Byte from memory to register
	store byte	`sb $s1, 100($s2)`	Memory[`$s2` + 100] = $s1	Byte from register to memory
	load upper immediate	`lui $s1, 100`	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Conditional branch	branch on equal	`beq $s1, $s2, 25`	if (`$s1 == $s2`) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1, $s2, 25`	if (`$s1 != $s2`) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1, $s2, $s3`	if (`$s2 < $s3`) `$s1` = 1; else `$s1` = 0	Compare less than; for beq, bne
	set less than immediate	`slti $s1, $s2, 100`	if (`$s2 < 100`) `$s1` = 1; else `$s1` = 0	Compare less than constant
Uncondi-tional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to `$ra`	For switch, procedure return
	jump and link	`jal 2500`	`$ra` = PC + 4; go to 10000	For procedure call

IC-UNICAMP

Funcionamento simplificado

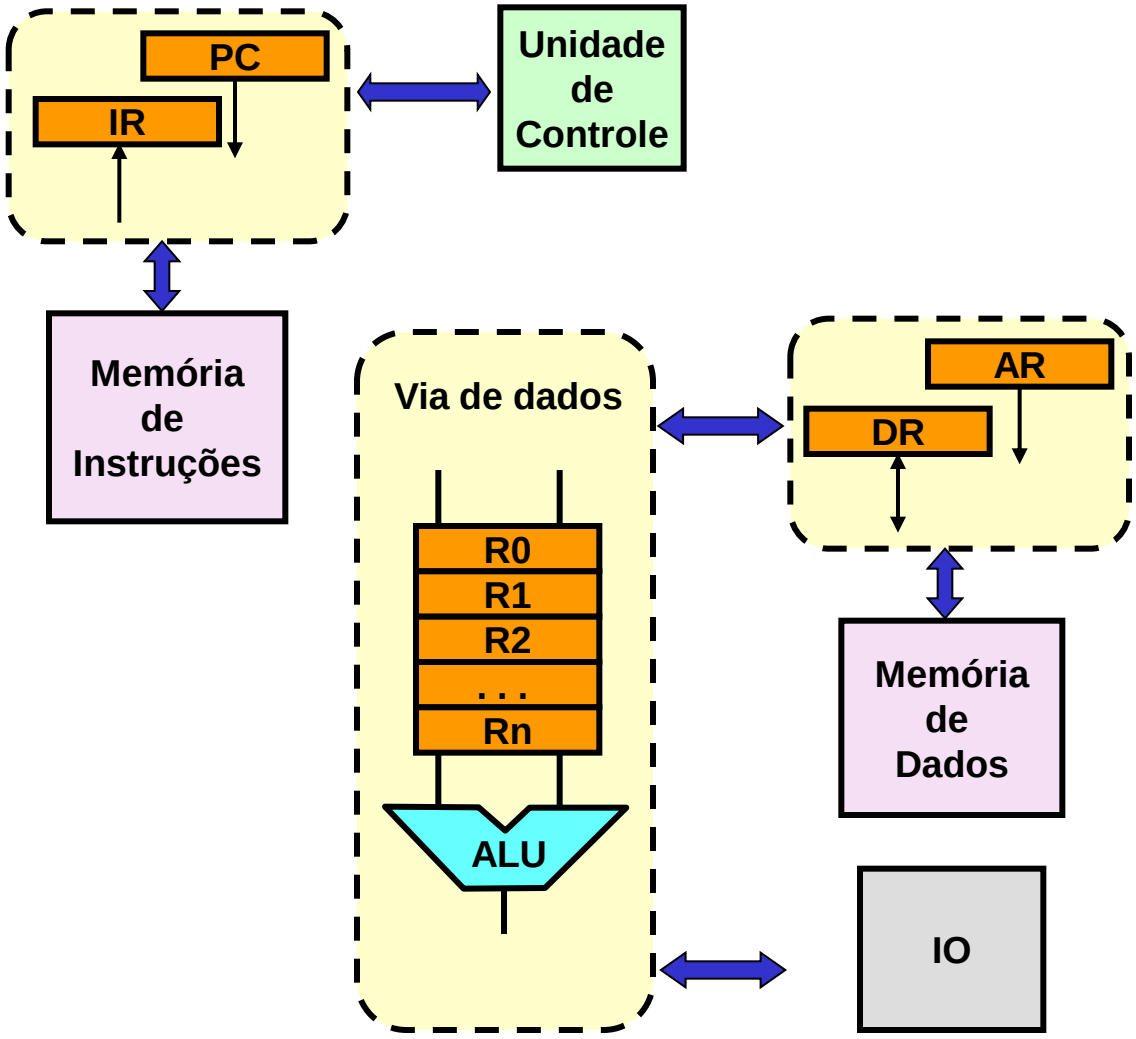
- Funcionamento simplificado para as fases
  - Busca de instrução ou fetch
  - Decodificação
  - Execução

- Sequenciamento das ações governado pela Unidade de Controle (é uma FSM)

IC-UNICAMP

1º ciclo: fetch ou busca de instruções

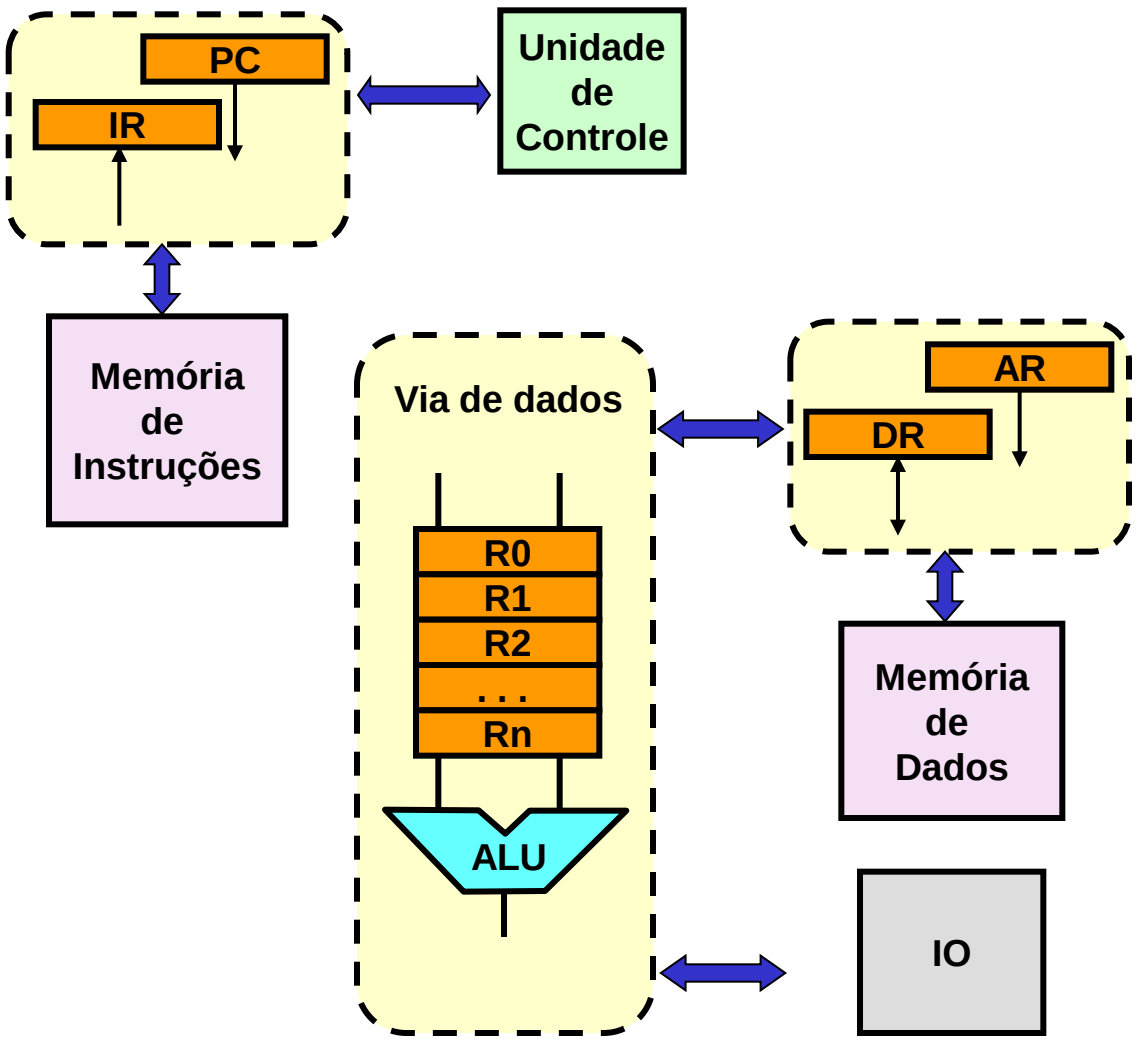
- IR ⇐ IM(PC)
- Incrementa PC



- fase comum p todas as instruções
- PC aponta para próxima instrução

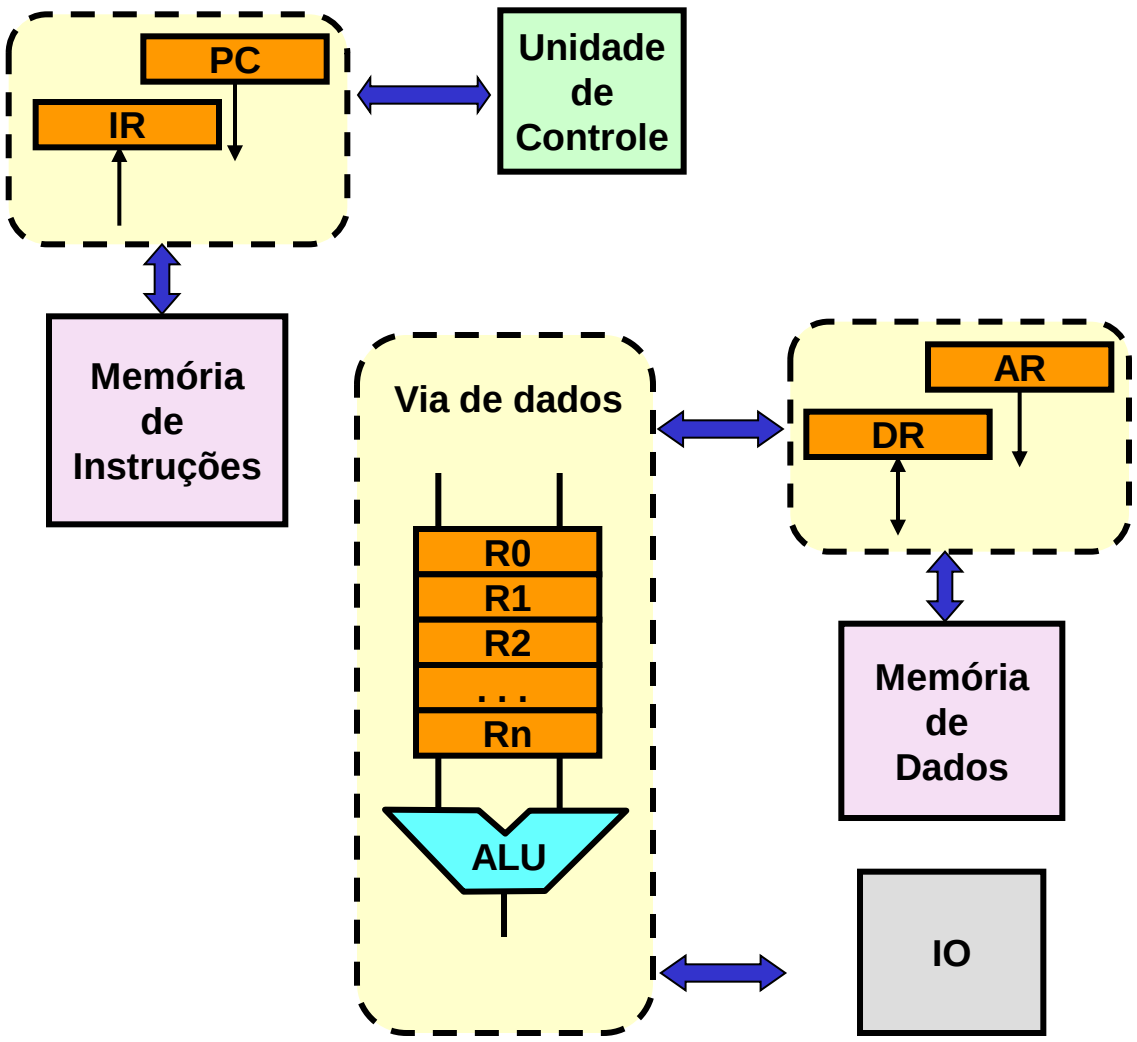
2º ciclo : decodificação

- Unidade de controle analisa IR
- Identifica instrução e gera sinais de controle
- fase comum p todas as instruções



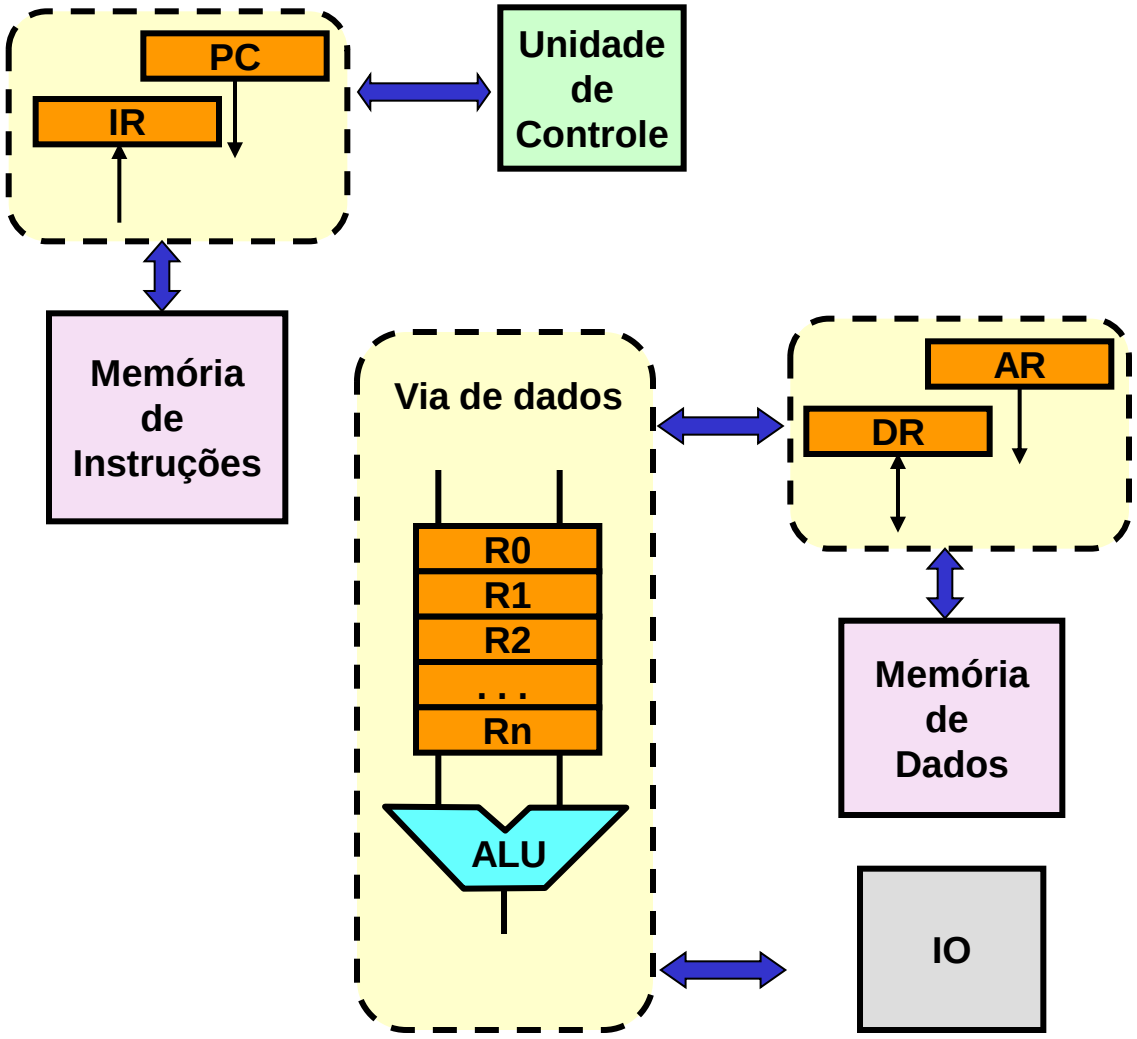
Execução: instruções aritm/lógica

- Ciclo3: controle
  - operação da ALU
  - operandos
- Ciclo4: controle
  - escrever resultado da operação no registrador resultado
- voltar para ciclo 1 (fetch)



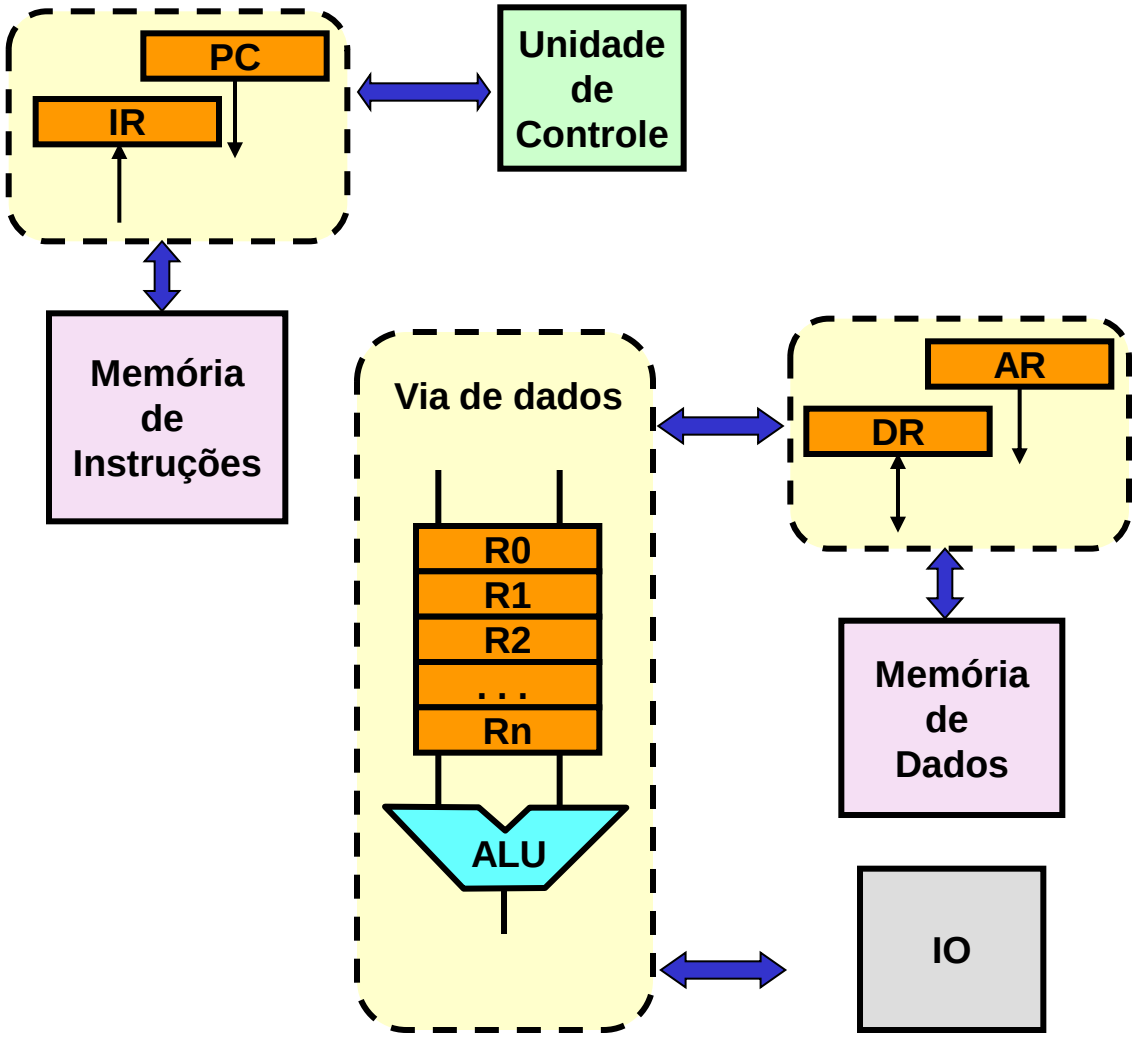
Execução: instruções de acesso à memória

- Ciclo3: controle
  - operação da ALU →soma
  - operandos: registrador e imediato
- Ciclo4: controle
  - enviar resultado (endereço) para AR
  - controle para DM (read write)
- Ciclo5:
  - se load, copiar DR → Reg
  - se store, execução concluída
- Voltar para ciclo 1 (fetch)

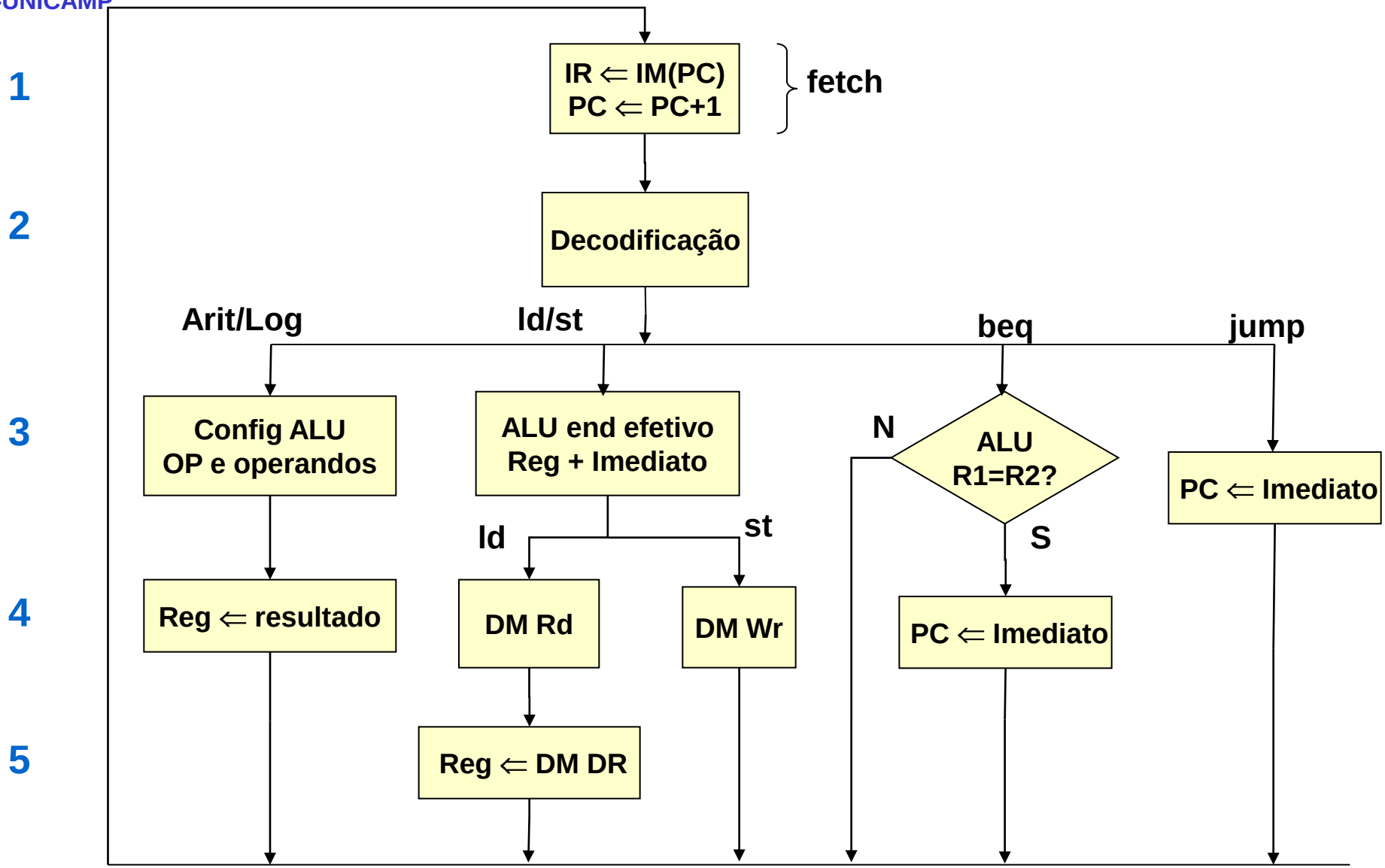


Execução: desvio

- Ciclo3: controle
  - se absoluto: imediato → PC e fim
  - se beq: definir ALU compara R1 e R2
- Ciclo4: controle
  - se R1=R2 então imediato → PC
  - se R1≠ R2, fim
- Voltar para ciclo 1 (fetch)

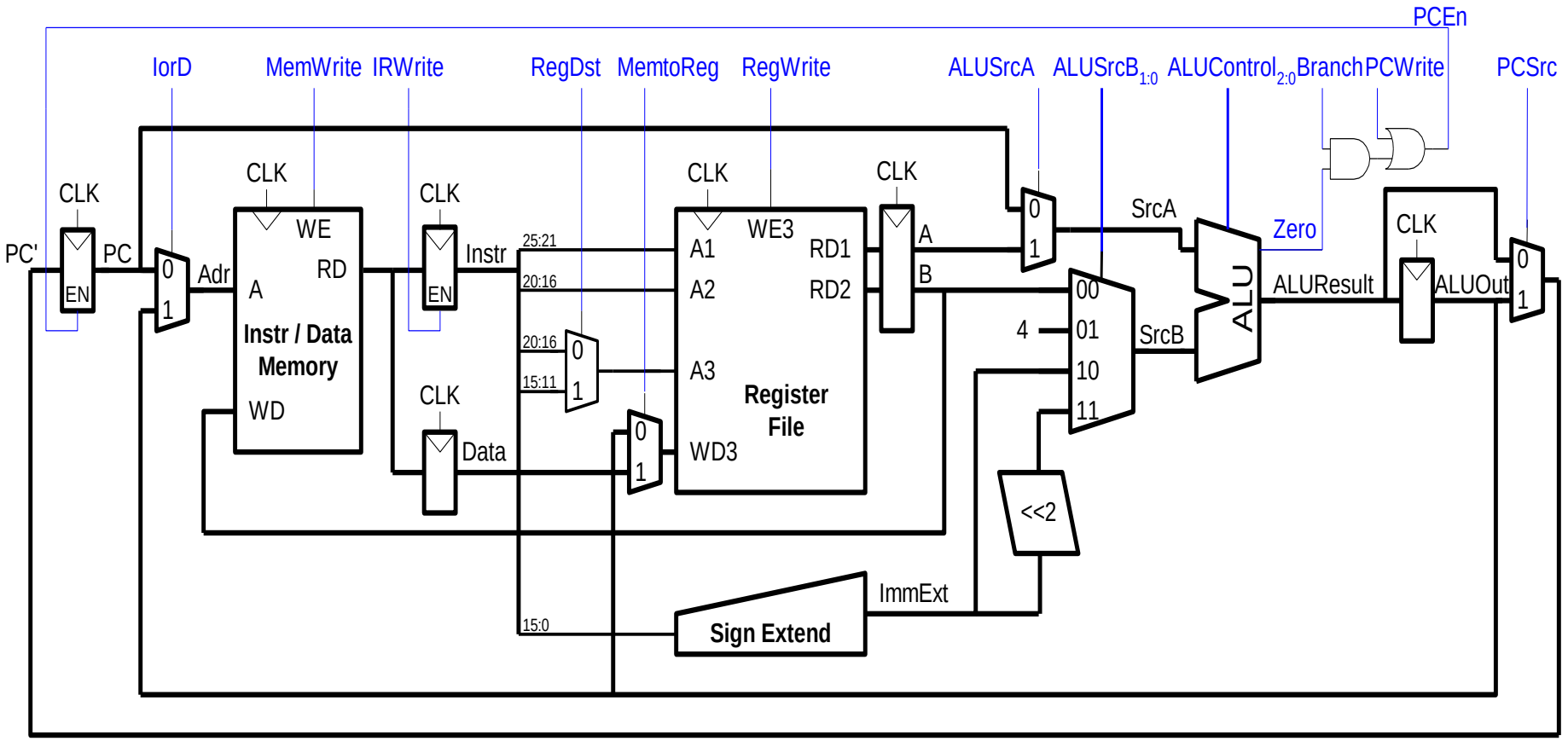


Fluxograma simplificado



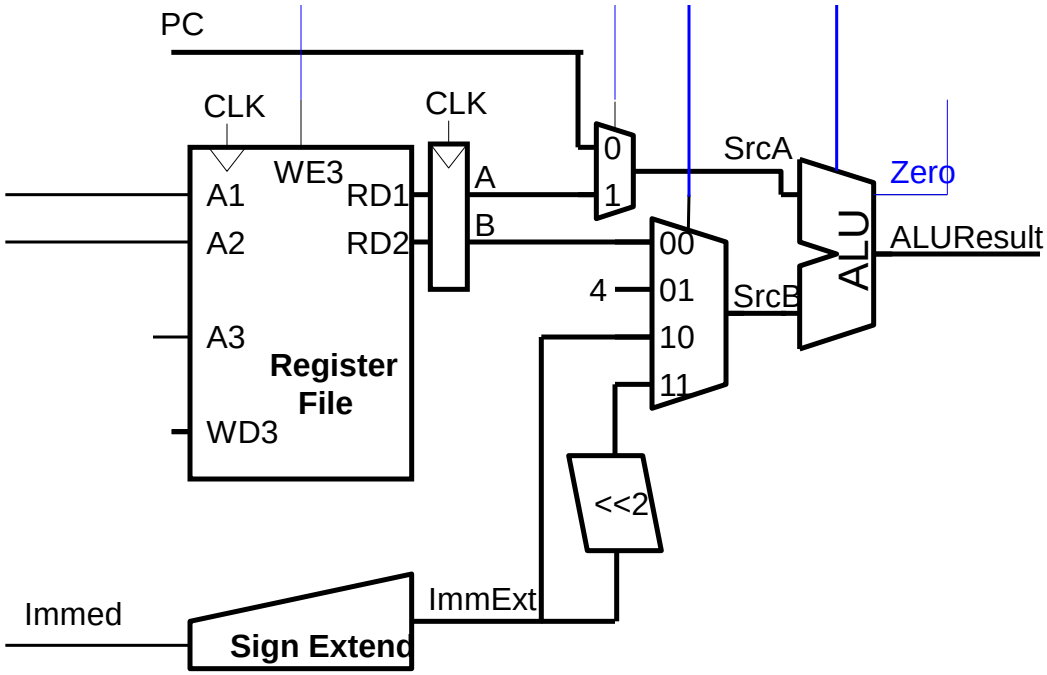
Processador MIPS Multicycle

- Datapath para o Processador Multicycle



Uso múltiplo da ALU

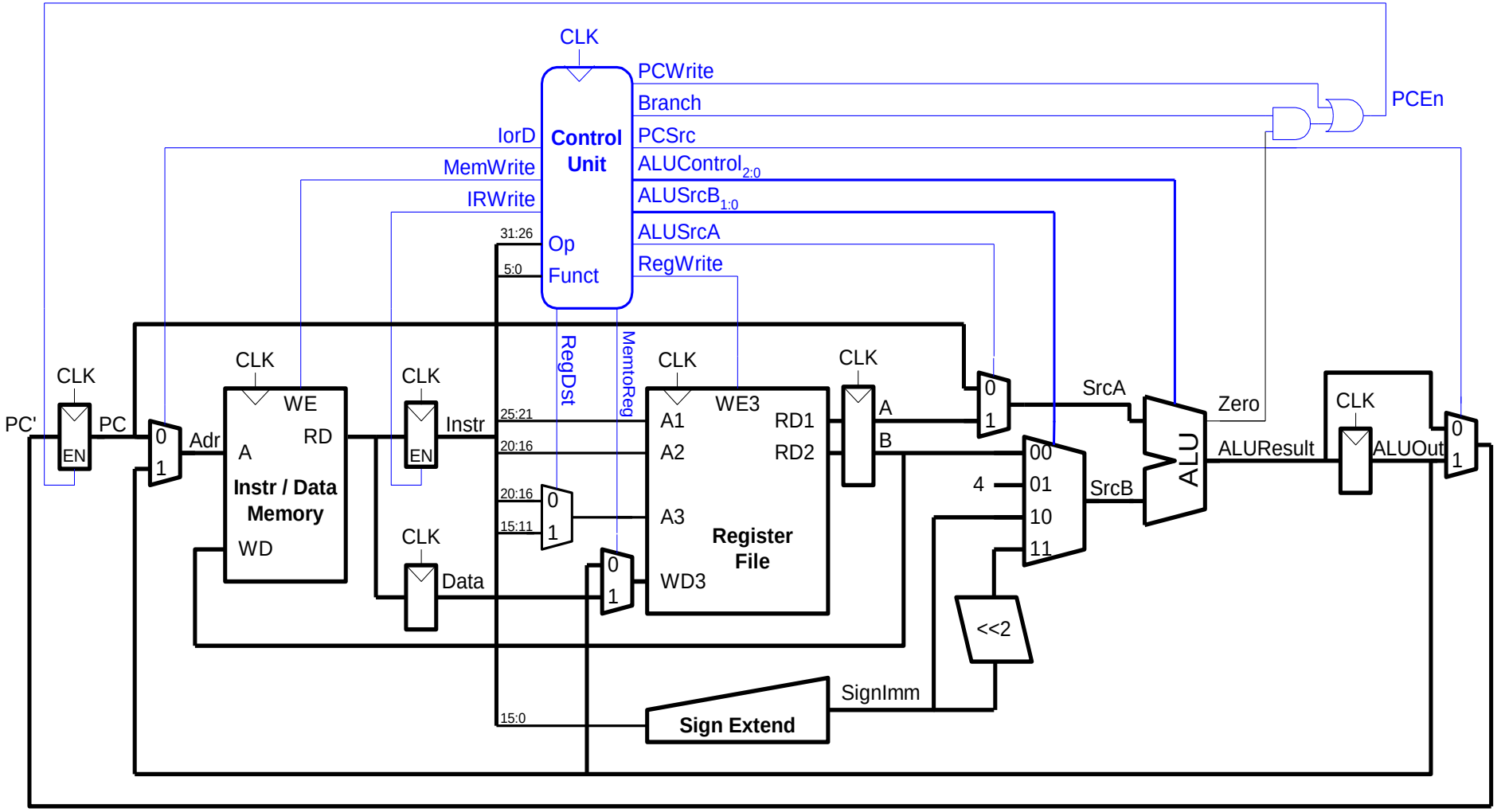
IC-UNICAMP



Mux2:1	Mux4:1	ALU_A	ALU_B	ALU_{Result}	Uso
1	00	A	B	A op B	Arit/log
0	11	PC	Imed>>2	End. desvio	Desv relativo PC
0	01	PC	4	PC←PC+4	Preparar fetch
1	10	A	Imediato	A+Imediato	End p ld e st

Processador MIPS Multicycle

- Controle para o Processador Multicycle



IC-UNICAMP

Conclusão

- Detalhes da implementação serão vistos em MC722