



# MC 602

## Circuitos Lógicos e Organização de Computadores

IC/Unicamp

Prof Mario Côrtes

### Capítulo MC10

Conceitos: Via de Dados e Processadores

*m1ps: meu primeiro processador simples*

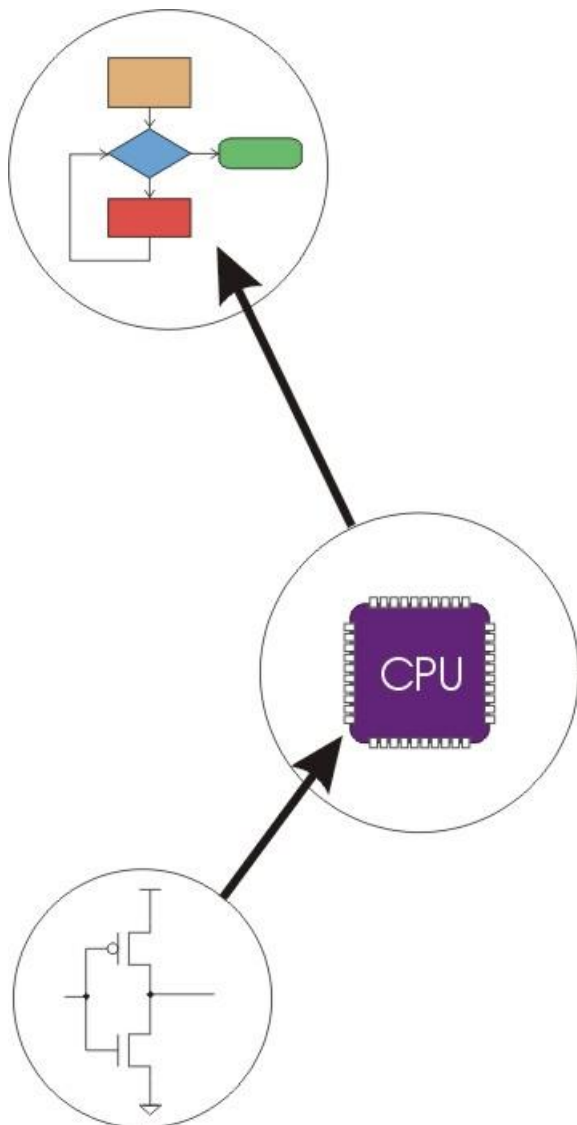
*(minúsculo MIPS)*



# Tópicos

- Níveis de abstração
- Máquina de programa armazenado / Von Neumann / Harvard
- m1ps
  - Specs
  - Diagrama de blocos: via de dados, controle, banco de processadores, memória
  - Conjunto de instruções (ISA / instruction set architecture)
  - Visão geral de funcionamento
- 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

MC102 e MC202

**Problema**

**Algoritmo**

**Programa**

**Instr Set  
Architecture**

**Software Design:**

escolher algoritmos and estrutura de dados

**Programação:**

implementar o *design* com uma linguagem

**Compilação/Interpretação:**

converter linguagem para instruções de máquina

MC404

# Níveis de abstração . . . .

*Instr Set  
Architecture*

**MC722**

***Processor Design:***

escolher estruturas para implementar ISA

*Microarch*

***Logic/Circuit Design:***

projeto a nível de gates e componentes

**MC602**

*Circuitos*

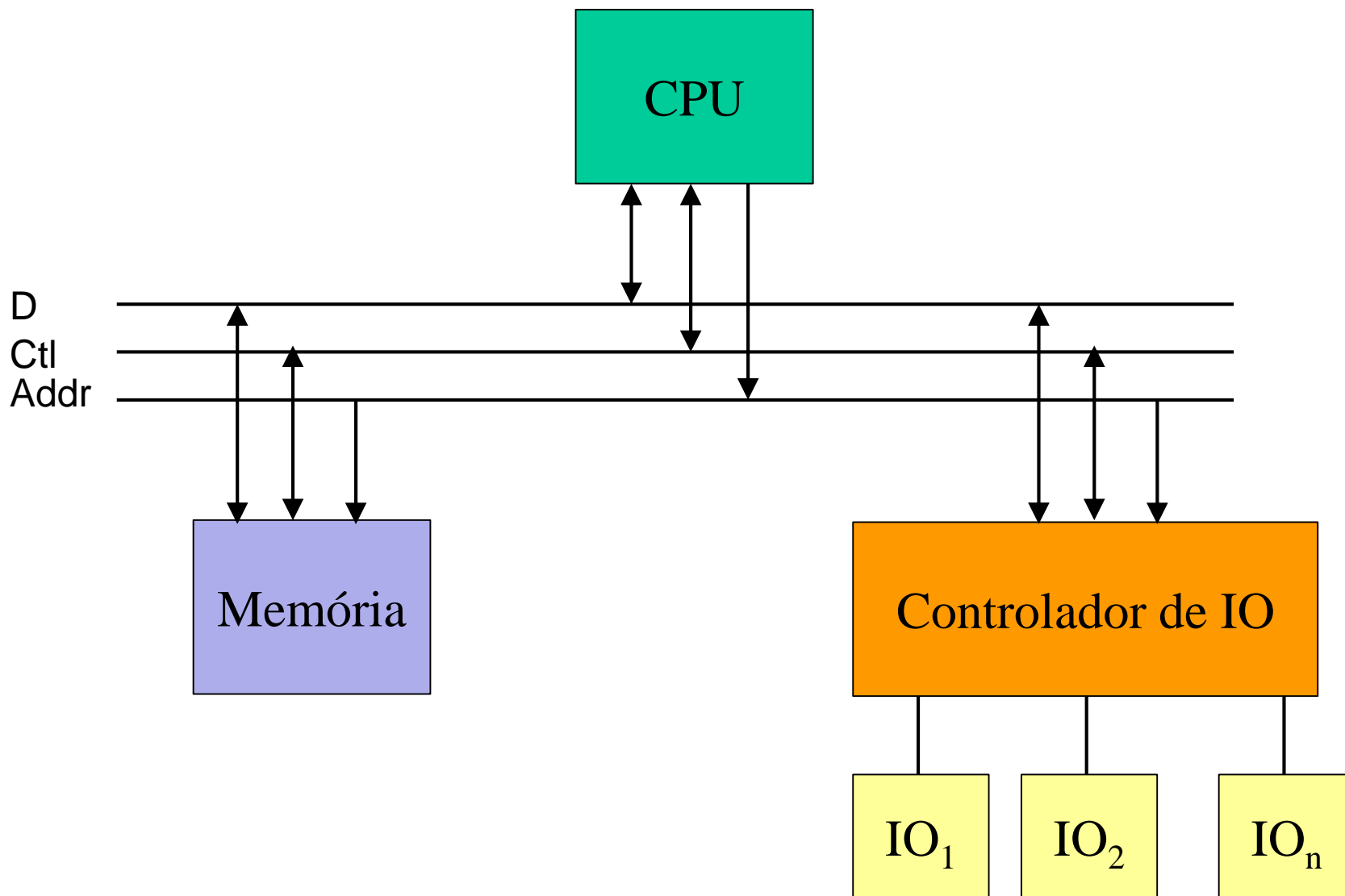
**MC922**

***Process Engineering & Fabrication:***

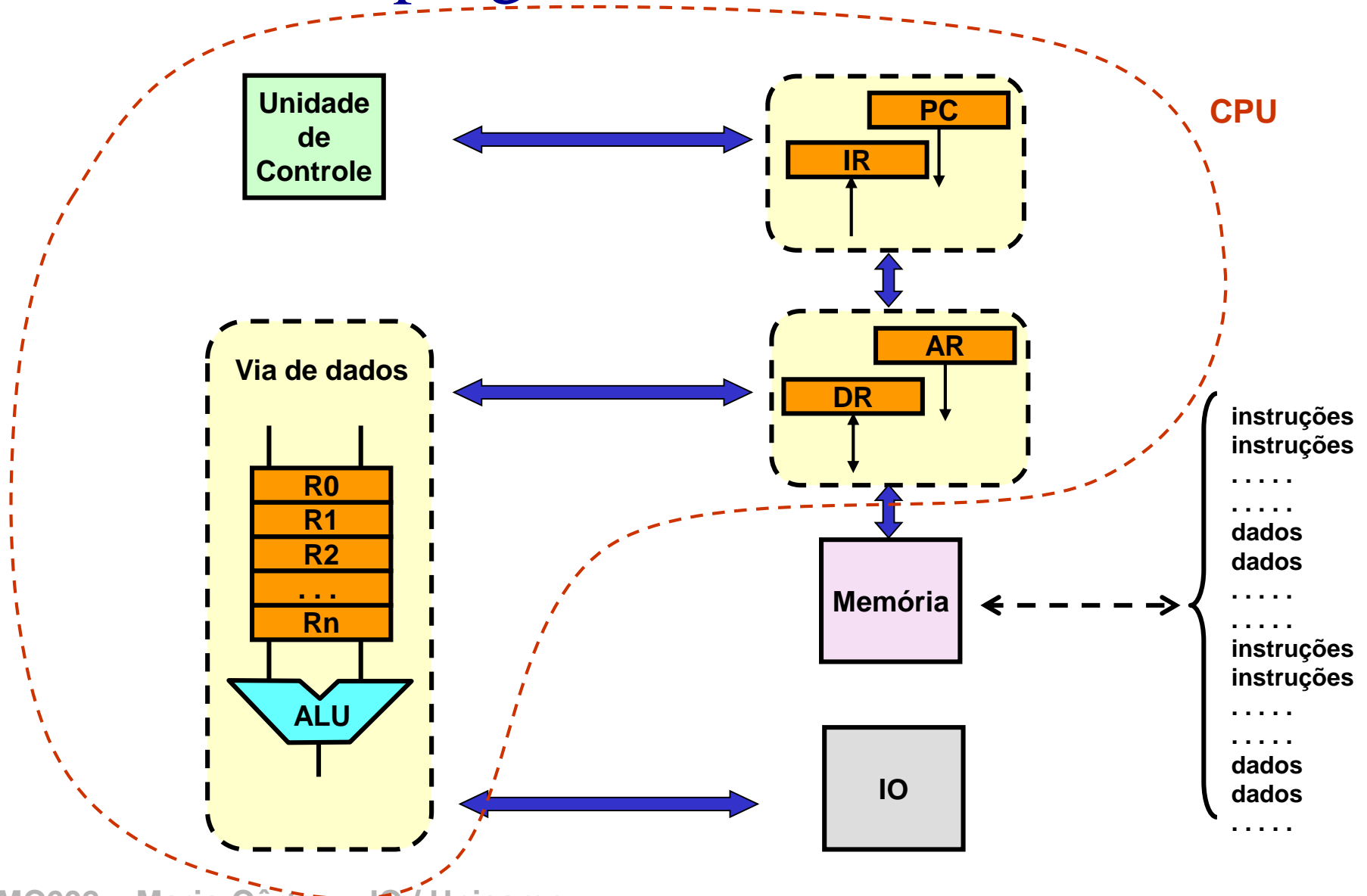
desenvolver e fabricar dispositivos e circuitos integrados

*Dispositivos*

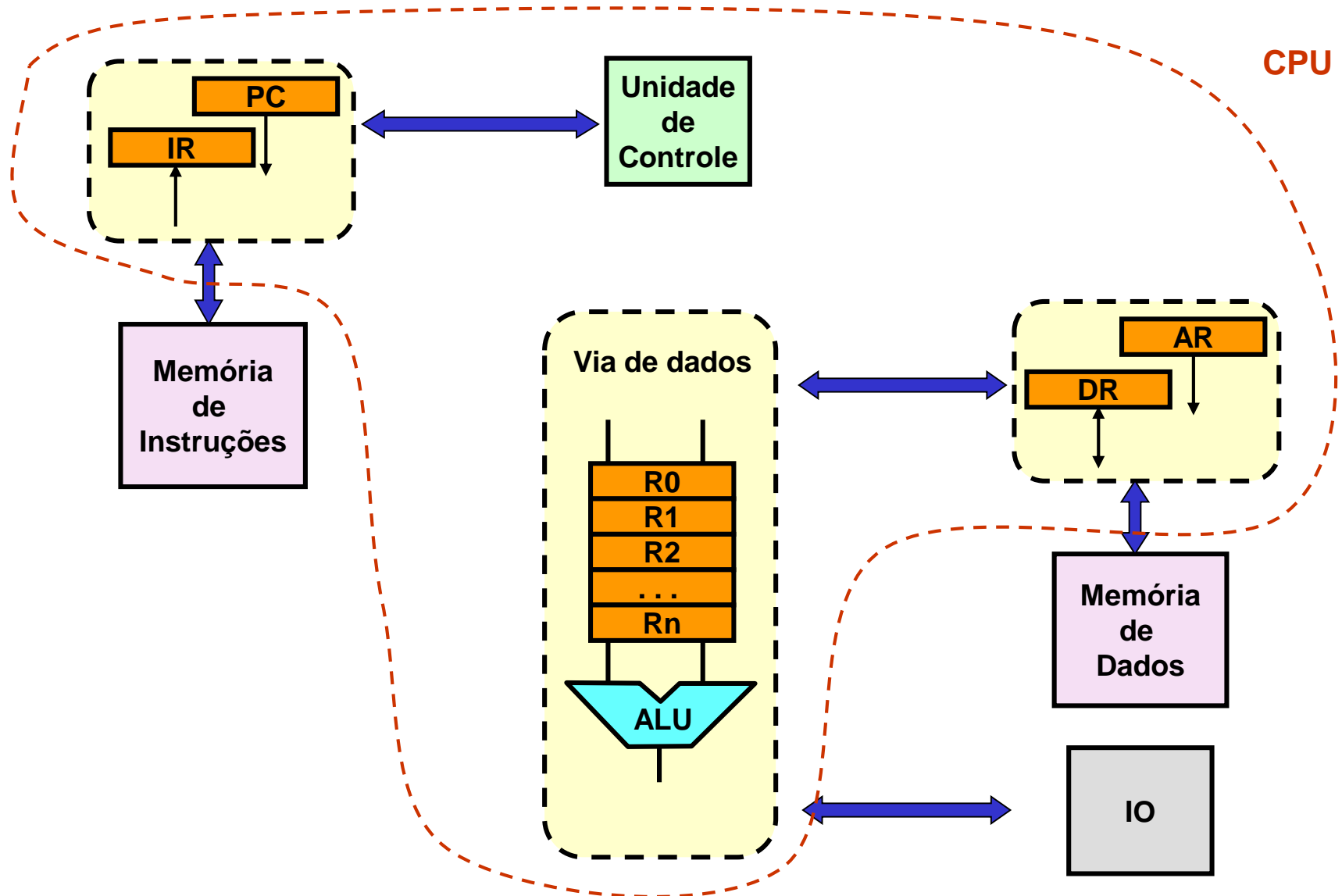
# Sistema típico



# Von Neumann: computador de programa armazenado



# Arquitetura de Harvard: memórias dedicadas DM e IM





# 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
- Apresentar um processador simples
  - m1ps: meu primeiro proc simples (minúsculo MIPS)



# Objetivos do mlps

- Primeira exposição à organização de processadores
- Conceitos principais, sem ser exaustivo
- Simples, mas não mínimo (pouca complexidade)
- Modular, intelegível, intuitivo, apreensível
- (quase) Completo: possível de implementar códigos básicos
- Extensível
- Uso de algumas estruturas iguais ou próximas ao MIPS



# Specs de implementação

- Dados e instruções de 32 bits
- Endereço de dados e instruções: palavras
- Banco de registradores = MIPS
- ALU: quase igual à do MIPS
- Registrador de status/condição (Z, C, N, V)
- Desvio condicional (status) e incondicional com endereço imediato completo (simplicidade)
- Formatos de instrução iguais ao MIPS

# m1ps Instruction Set Architecture (ISA)

## Instruções Formato R, lógicas e aritméticas (001 0XX)

	Obs
add Rd, Rs1, Rs2	$Rd \leftarrow Rs1 + Rs2$
sub Rd, Rs1, Rs3	$Rd \leftarrow Rs1 - Rs2$
and Rd, Rs1, Rs4	$Rd \leftarrow Rs1 \text{ And } Rs2$
or Rd, Rs1, Rs5	$Rd \leftarrow Rs1 \text{ Or } Rs2$

## Instruções Formato R, transf de dados

	Obs
lw Rd, Rs1	$Rd \leftarrow DM(Rs1)$
sw Rs1, Rs2	$DM(Rs1) \leftarrow Rs2$
in Rd	$Rd \leftarrow \text{IO data in}$
out Rs1	$\text{IO data out} \leftarrow Rs1$

## Instruções Formato J, desvio

	Obs
J addr	$Pc \leftarrow \text{addr}$
BrZ	" if Z=1
BrN	" if N=1
BrV	" if V=1
BrC	" if C=1
BrnZ	" if Z=0
BrnN	" if N=0
BrnV	" if V=0

PC: Program counter  
points to current (or next)  
instruction

PC contents: address of  
current (next) instruction in IM  
(instruction memory)

# Exemplo: soma dos elementos de um vetor

Código em C

```
sum=0;
for (i=0; i < num_elements; i++) {
    sum = sum + a[i];
}
```

Código em Assembly do m1ps

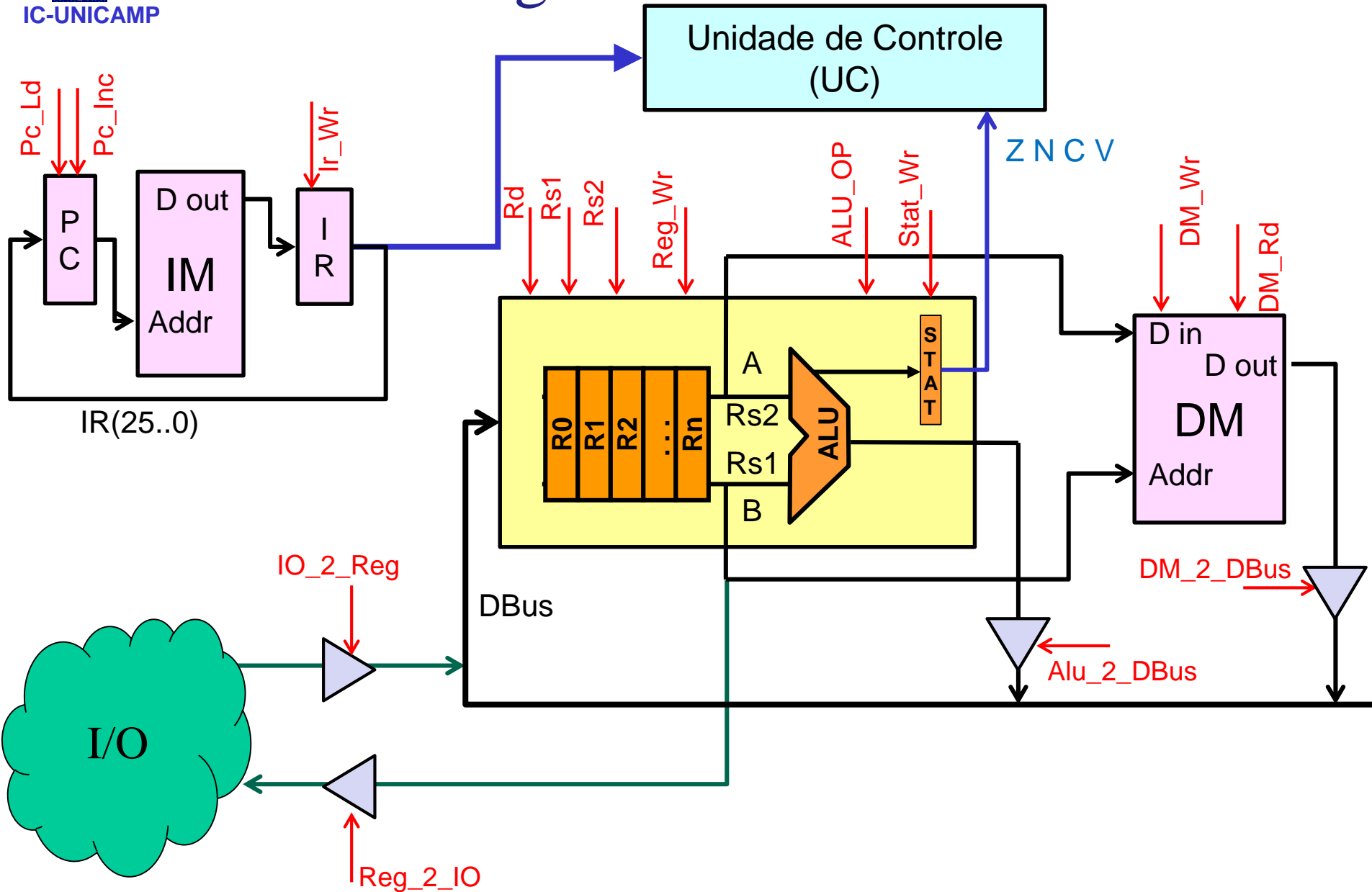
```
# R1 has the vector's base memory address
# R2 has the number of elements
# R4 holds the sum
# R30 is a temporary memory data register

    add R4, R0, R0    # clean R4 variable sum

loop:
    lw  R30, R1      # tmp = vector data pointed by R1
    add R4, R30, R4  # sum = tmp + sum
    addi R1, R1, 1   # base address incr
    addi R2, R2, -1  # nLeft decr
    brnz loop        # goto loop if (nLeft != 0)

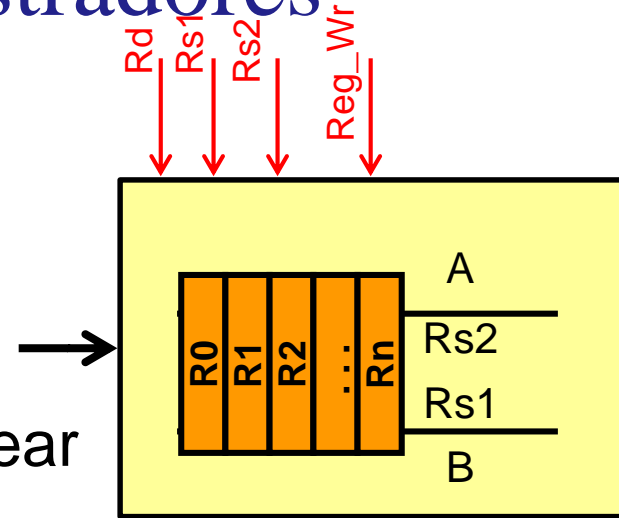
end:
```

# Diagrama de blocos



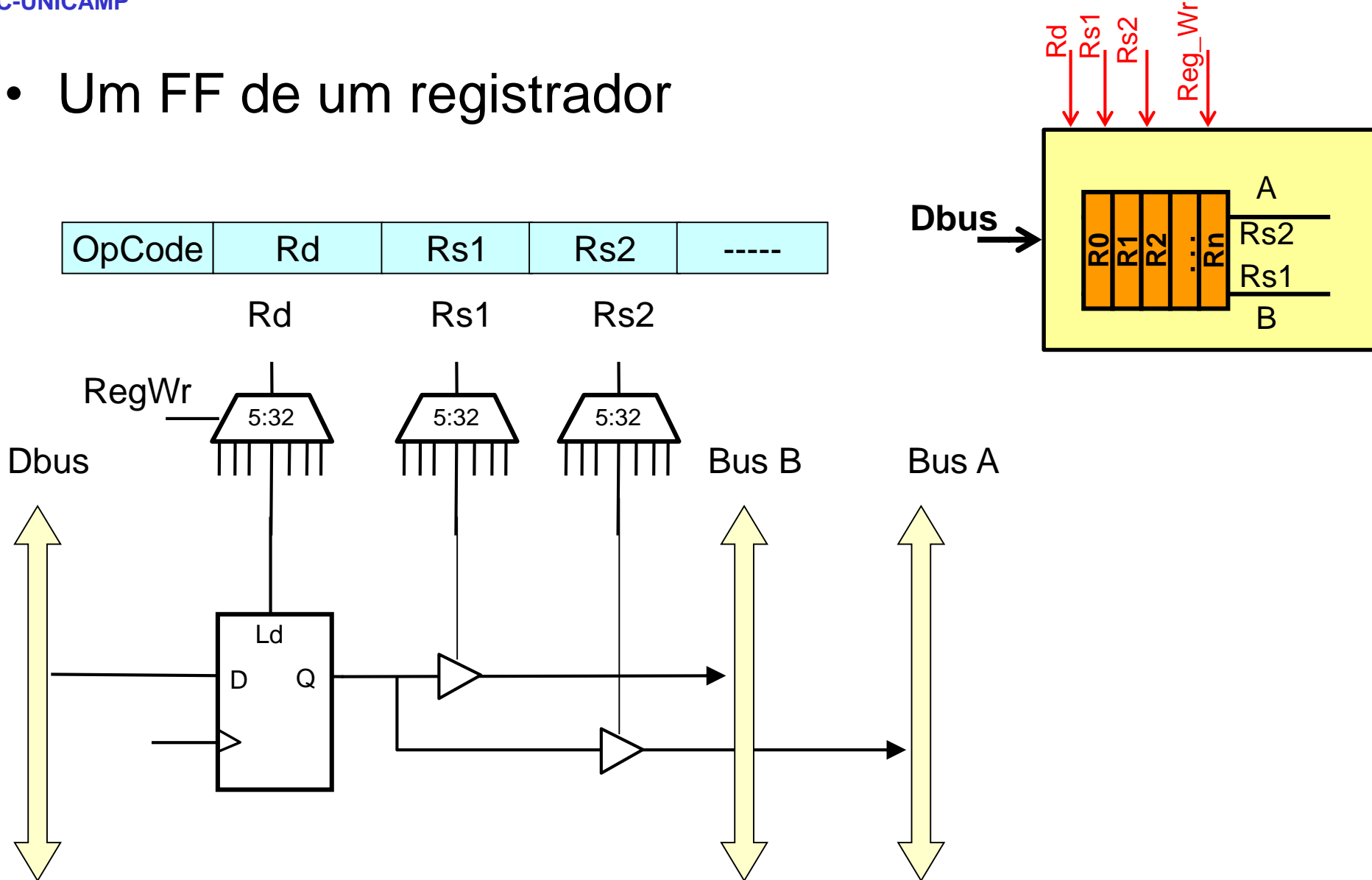
# Módulos: banco de registradores

- Parte da via de dados
- 32 registradores de 32 bits
  - ATENÇÃO: R0 = 0
  - permite pseudo instruções move e clear
- Dados
  - Entrada: Barramento Dbus (32 bits)
  - Saídas: Barramentos A e B (32 bits)
- Controle
  - Rs1 e Rs2 (5bits): selecionam registradores → saídas A e B
  - Rd (5bits): seleciona registrador a ser escrito
  - Reg\_Wr (1b): controle de escrita



# Banco de registradores: implementação

- Um FF de um registrador

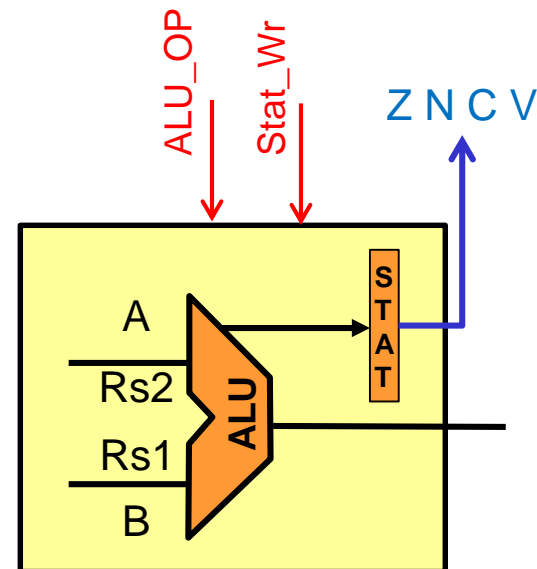






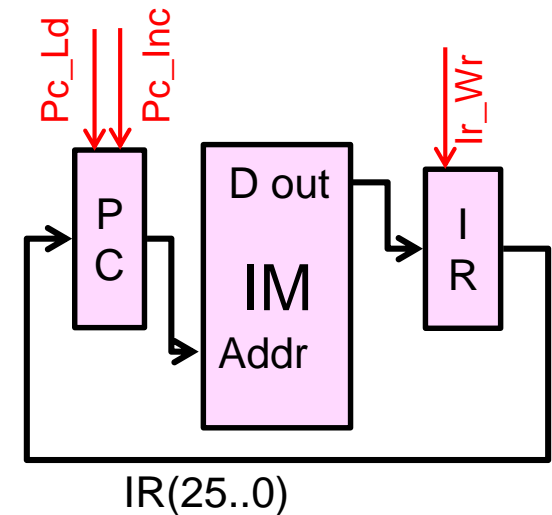
# Módulos: ALU

- Parte da via de dados
- Operações lógicas e aritméticas de operandos de 32 bits: add, sub, and, or
- Dados
  - Entradas: Barramentos A e B (32 bits)
  - Saídas: ALU\_out (32 bits)
- Controle
  - AluOp: define operação da ALU (ver conj de instruções)
  - Z,C,V,N: bits de status da operação



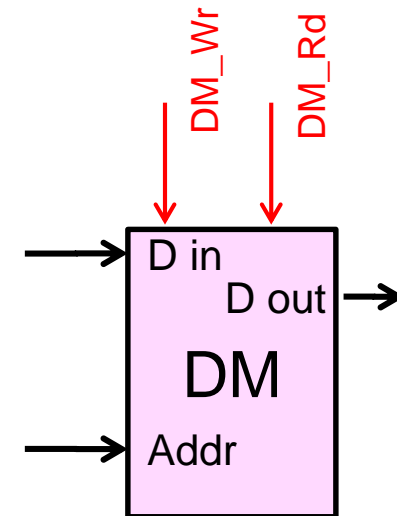
# Módulos: sistema de memória de instruções (IM)

- IM (somente leitura):  $2^{26}$  linhas de 32b de largura
  - Dados: leitura da instrução (32 bits)  $\rightarrow$  IR
  - Endereço: PC (26 bits)
  - Controle: leitura sempre
- PC
  - Dados: incremento ou carga paralela
  - Controle:
    - PC-Inc:  $PC \leftarrow PC + 1$
    - PC-Ld:  $PC \leftarrow \text{Target Address}$ 
      - carga paralela de endereço de desvio (26 bits)
- Observações:
  - cuidado com a temporização
  - limitar tamanho na implementação DE1 ( $< 2^{26}$  linhas)



# Módulos: sistema de memória de dados (DM)

- DM:  $2^{32}$  linhas de 32b de largura
  - sem Regs dedicados para dados e endereço
- Dados:
  - saída Dout (32b) é um dos sinais a acionar o DBus
  - entrada Din (32b): saída A do banco de registradores (definido por Rs2)
- Endereço: vem de Rs1
  - entrada Addr (32b): saída B do banco de registradores (definido por Rs1)
- Controles (1b): DM\_Wr e DM\_Rd
- Observações:
  - cuidado com a temporização
  - limitar tamanho na implementação DE1 ( $< 2^{32}$  linhas )





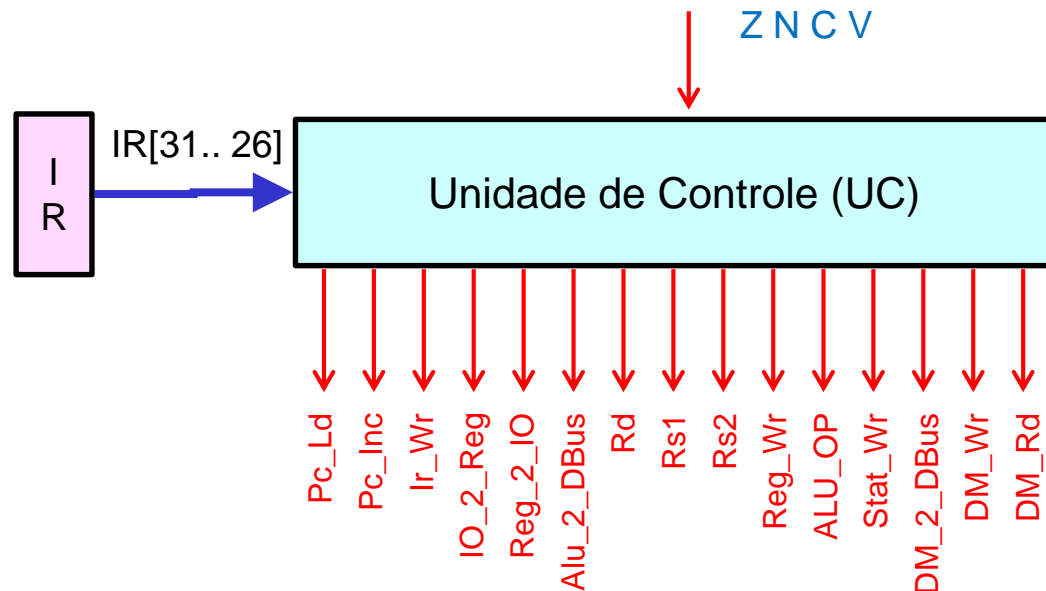
# Módulos: Unidade de controle

- Interfaces

- Entradas: IR e Status
- Saídas: 15 sinais de controle

- Estrutura

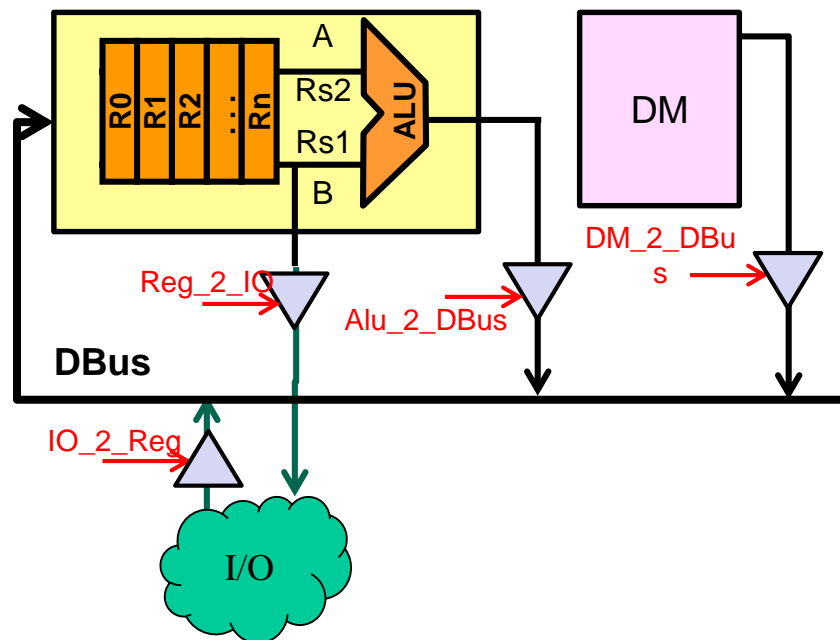
- Principal: FSM
- Apoio:
  - lógica para controle de desvio
  - lógica para controle da ALU





# Módulos: DBus e I/O

- I/O
  - In: I/O → Reg, via DBus
  - Out: Reg → I/O
- DBus
  - 32 bits
  - Acionam o barramento:
    - DM, saída da ALU, I/O in
  - Leem do barramento
    - Banco d registradores





# ISA: código de máquina

## Instruções Formato R, lógicas e aritméticas (001 OXX)

	Op(31..26)	Rd(25..21)	Rs1(20..16)	Rs2(15..11)	Unused(10..0)	Obs
add Rd, Rs1, Rs2	001 000				-	Rd ← Rs1 + Rs2
sub Rd, Rs1, Rs3	001 001				-	Rd ← Rs1 - Rs2
and Rd, Rs1, Rs4	001 010				-	Rd ← Rs1 And Rs2
or Rd, Rs1, Rs5	001 011				-	Rd ← Rs1 Or Rs2
	6	5	5	5	11	

## Instruções Formato R, transf de dados

	Op(31..26)	Rd(25..21)	Rs1(20..16)	Rs2(15..11)	Unused(10..0)	Obs
lw Rd, Rs1	000 111			-	-	Rd ← DM(Rs1)
sw Rs1, Rs2	010 111	-			-	DM(Rs1) ← Rs2
in Rd	100 000		-	-	-	Rd ← IO data in
out Rs1	110 000	-		-	-	IO data out ← Rs1
	6	5	5	5	11	

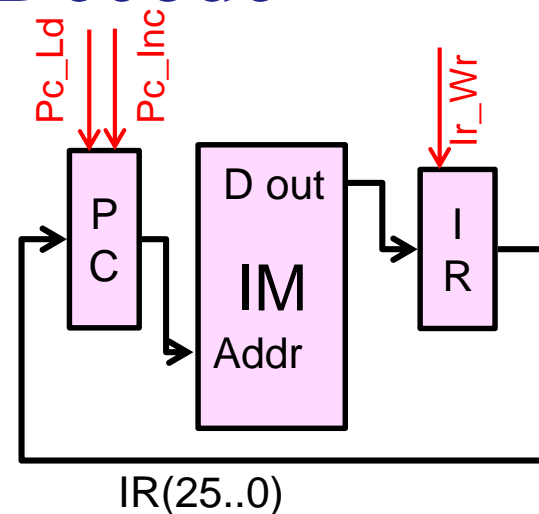
## Instruções Formato J, desvio

	Op(31..26)	Addr(25..0)	Obs
J addr	111 111		Pc ← addr
BrZ	111 000		" if Z=1
BrN	111 001		" if N=1
BrV	111 010		" if V=1
BrC	111 011		" if C=1
BrnZ	111 100		" if Z=0
BrnN	111 101		" if N=0
BrnV	111 110		" if V=0



# Ciclos de execução: Fetch e Decode

- Ciclo 1: Fetch (busca de instrução)
  - IR\_Ld: saída da IM escrita em IR
    - saída de IM mostra continuamente conteúdo da posição apontada por PC
  - PC\_Inc: atualiza PC
    - a ser usado na próxima instrução
    - pode ser sobre-escrito se instrução = desvio

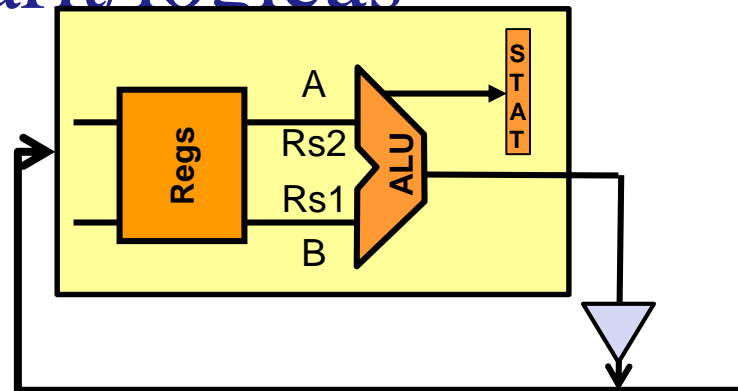


- Ciclo 2: Decodificação
  - Um ciclo para a unidade de controle decodificar a instrução e gerar os sinais de controle



## Ciclo 3 de execução: arit/lógicas

- Configura ALU
  - função: ALU\_OP
  - operandos de entrada: Rs1, Rs2
  - registrador de destino: Rd
- Aciona saída do barramento ALU\_2\_DBus
  - ALU\_2\_DBus
- Ao final do ciclo (borda do próx. clock), escrita
  - No registrador de destino: Reg\_Wr
  - No registrador de status: Stat\_Wr
- Pode ser realizada em um ciclo ou 2
  - caminho crítico: seleciona operandos, envia p ALU, realiza operação (32bits), aciona barramento, escreve em Rd

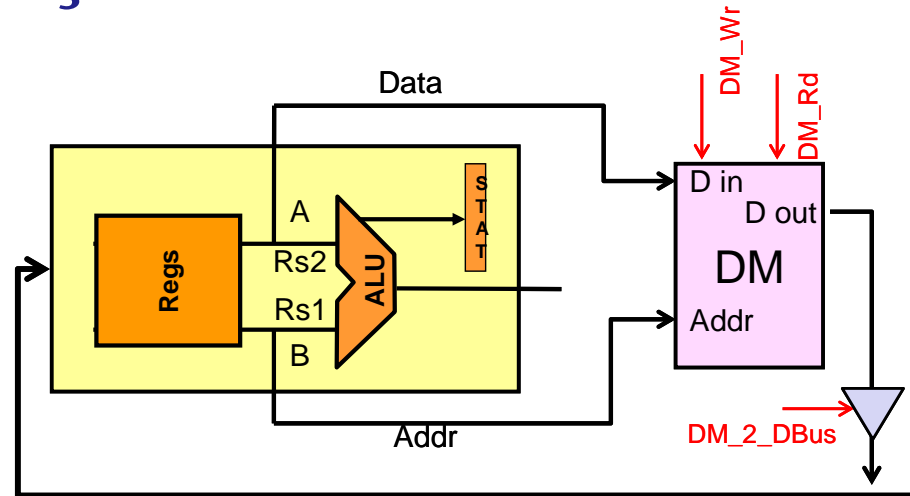







# Ciclo 3 de execução: lw e sw

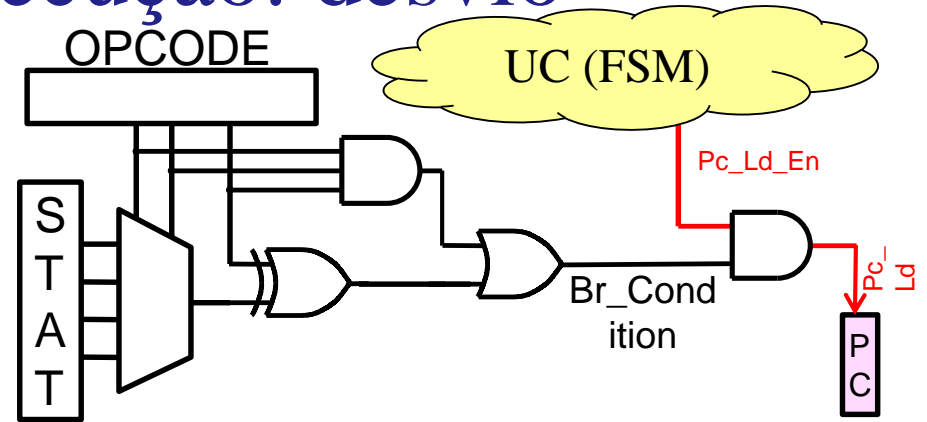
- DM no ciclo 2, definidos
  - Endereço: Rs1
  - Dados para escrita: Rs2
  - Registrador destino: Rd
- SW
  - $DM(Rs1) \leftarrow Rs2$
  - ao final do ciclo: DM\_Wr
- lw
  - $Rd \leftarrow DM(Rs1)$
  - $DM\_2\_DBus = 1$
  - ao final do ciclo: Reg\_Wr
- Pode ser realizada em um ciclo ou 2
  - caminho crítico: seleciona dados e endereço, envia p DM, operação de leitura ou escrita, aciona barramento, escreve em Rd





# Ciclo 3 de execução: desvio

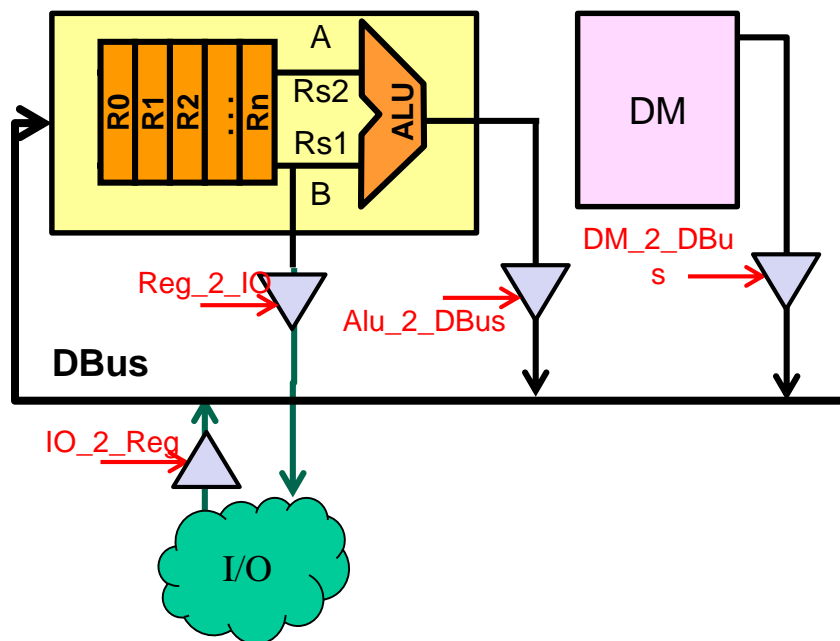
- Uma única ação da UC
  - ver 
  - sinalizar momento da carga de PC → Pc\_Ld\_En
- Desvio condicional
  - opcode comparado com condição em STAT (última operação aritmética)
- Desvio incondicional
  - decodifica OpCode e gera condição, independente de STAT





# Ciclo 3 de execução: I/O

- In Rd
  - $Rd \leftarrow I/O$
  - controles: IO\_2\_Reg
  - ao final do clock: Reg\_Wr
- Out Rs1
  - $I/O \leftarrow Rs1$
  - controles: Reg\_2\_IO
- Pode ser realizada em um ciclo ou 2
  - caminho crítico: seleciona Rd, ativa leitura do barramento, escreve em Rd

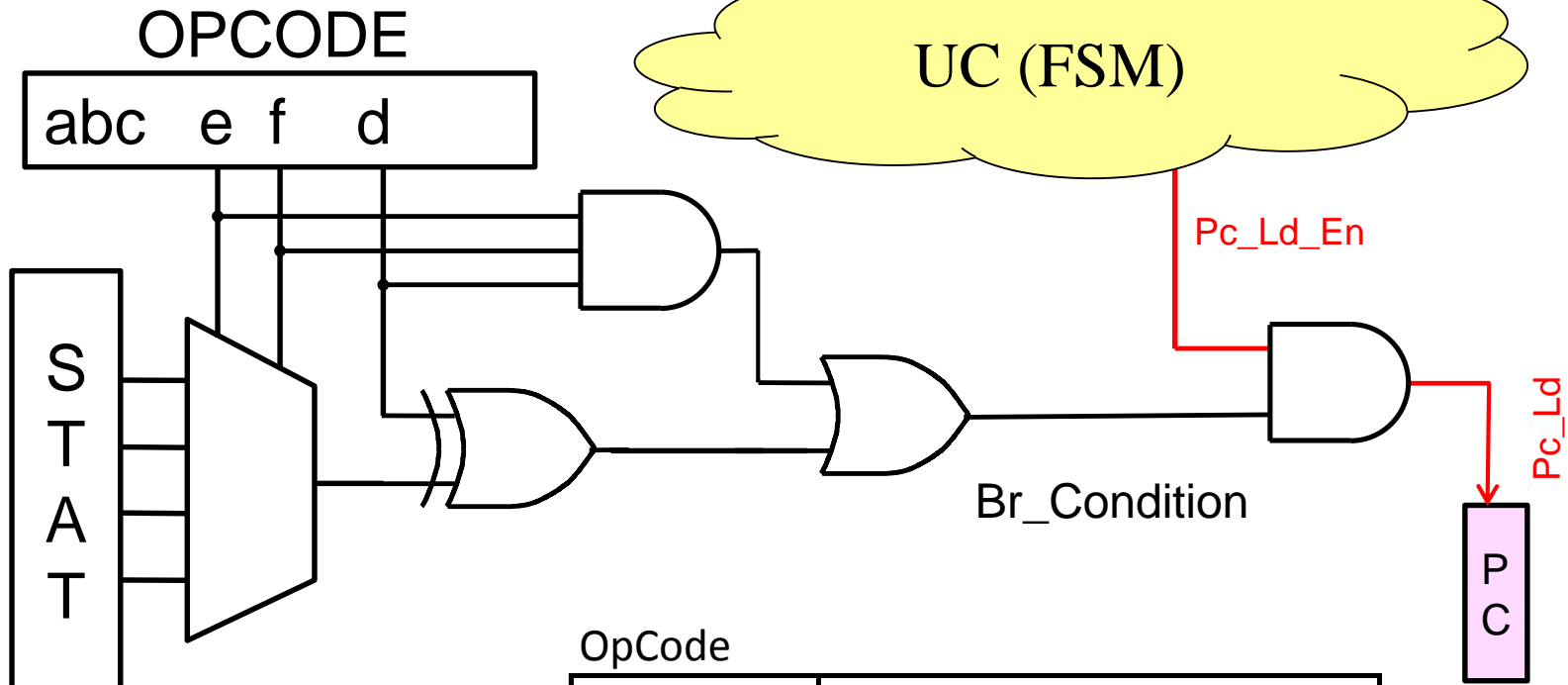




# Controle de Desvio: Possibilidades

- Um estado para cada instrução de desvio
  - Teste é específico para o estado
  - Haverá tantos estados específicos quanto instruções de desvio
  - Complica a máquina de estados desnecessariamente
- Alternativa
  - Um único estado na FSM de controle sinaliza instrução de desvio
  - Hardware especializado para controlar desvios (Target Address Controller – TrgtAdrCtl)
  - Situado entre a FSM, PC e Stat
  - Menos HW e mais flexibilidade

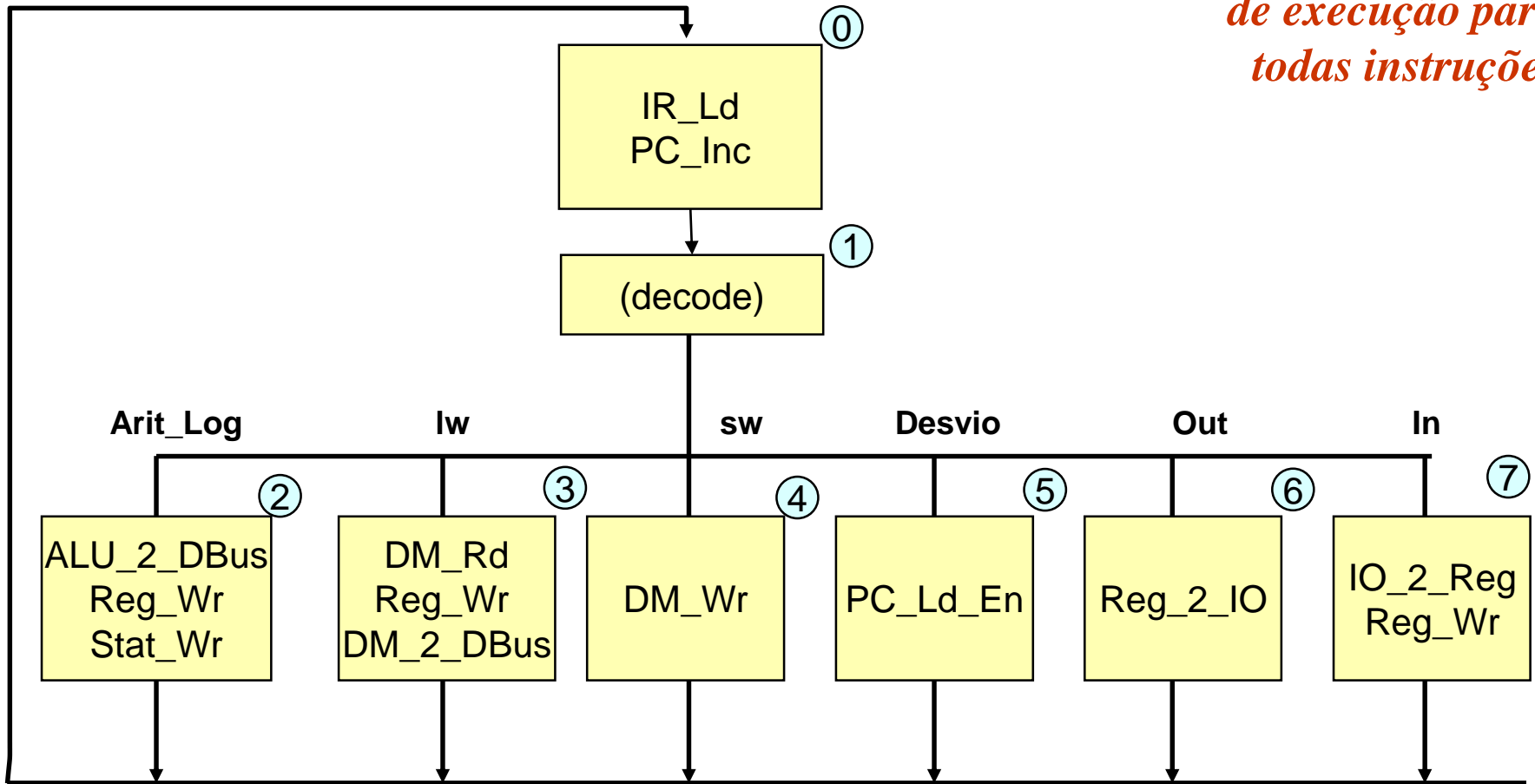
# Controle de Desvio: Target Address Ctl



	OpCode	
J addr	111 111	Pc ← addr
BrZ	111 000	Pc ← addr if Z=1
BrN	111 001	Pc ← addr if N=1
BrV	111 010	Pc ← addr if V=1
BrC	111 011	Pc ← addr if C=1
BrnZ	111 100	Pc ← addr if Z=0
BrnN	111 101	Pc ← addr if N=0
BrnV	111 110	Pc ← addr if V=0

# Possível fluxo de controle

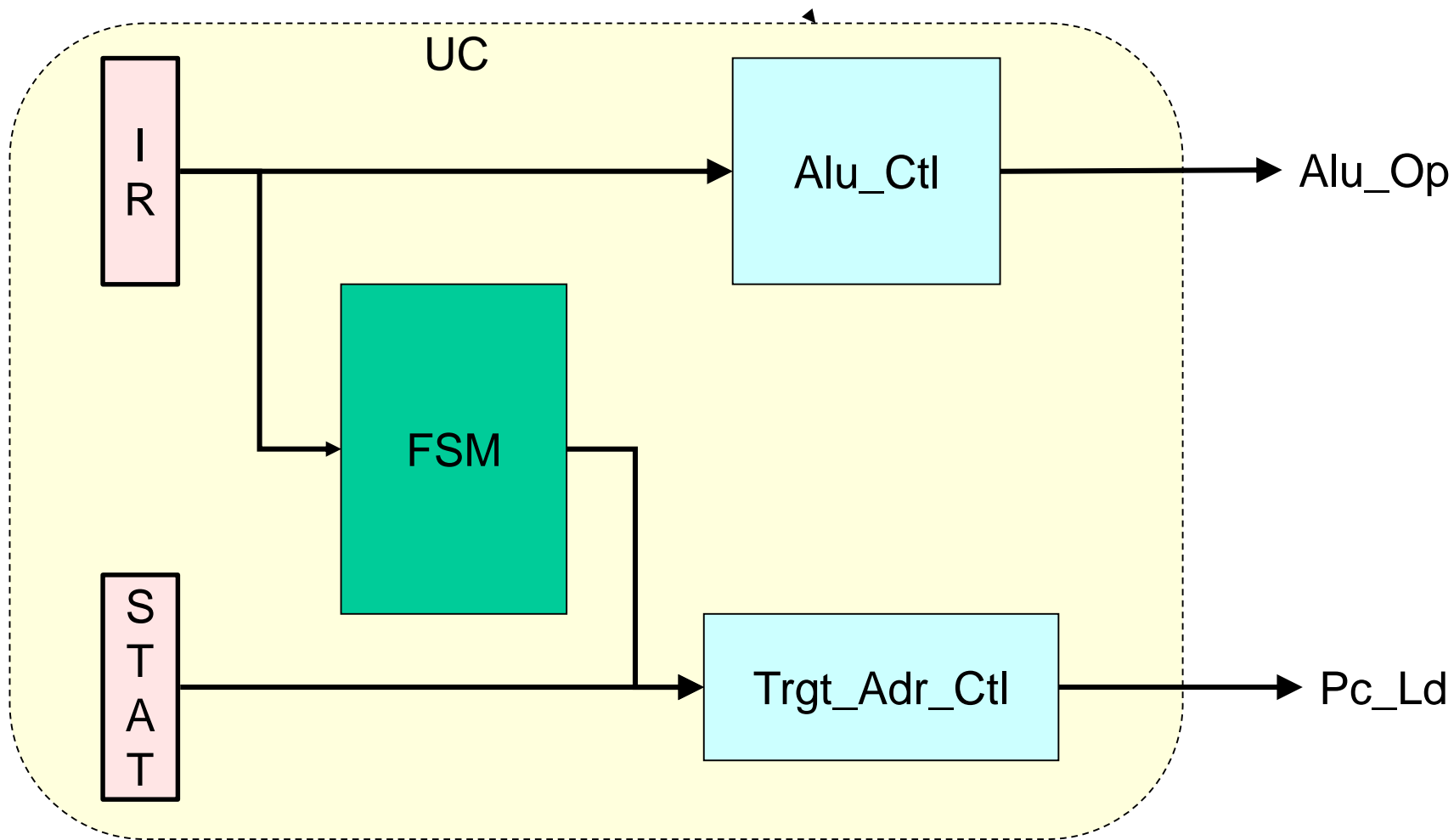
*Assumindo 1 ciclo de execução para todas instruções*





# Detalhes da UC

- FSM + controladores especializados: ALU e PC



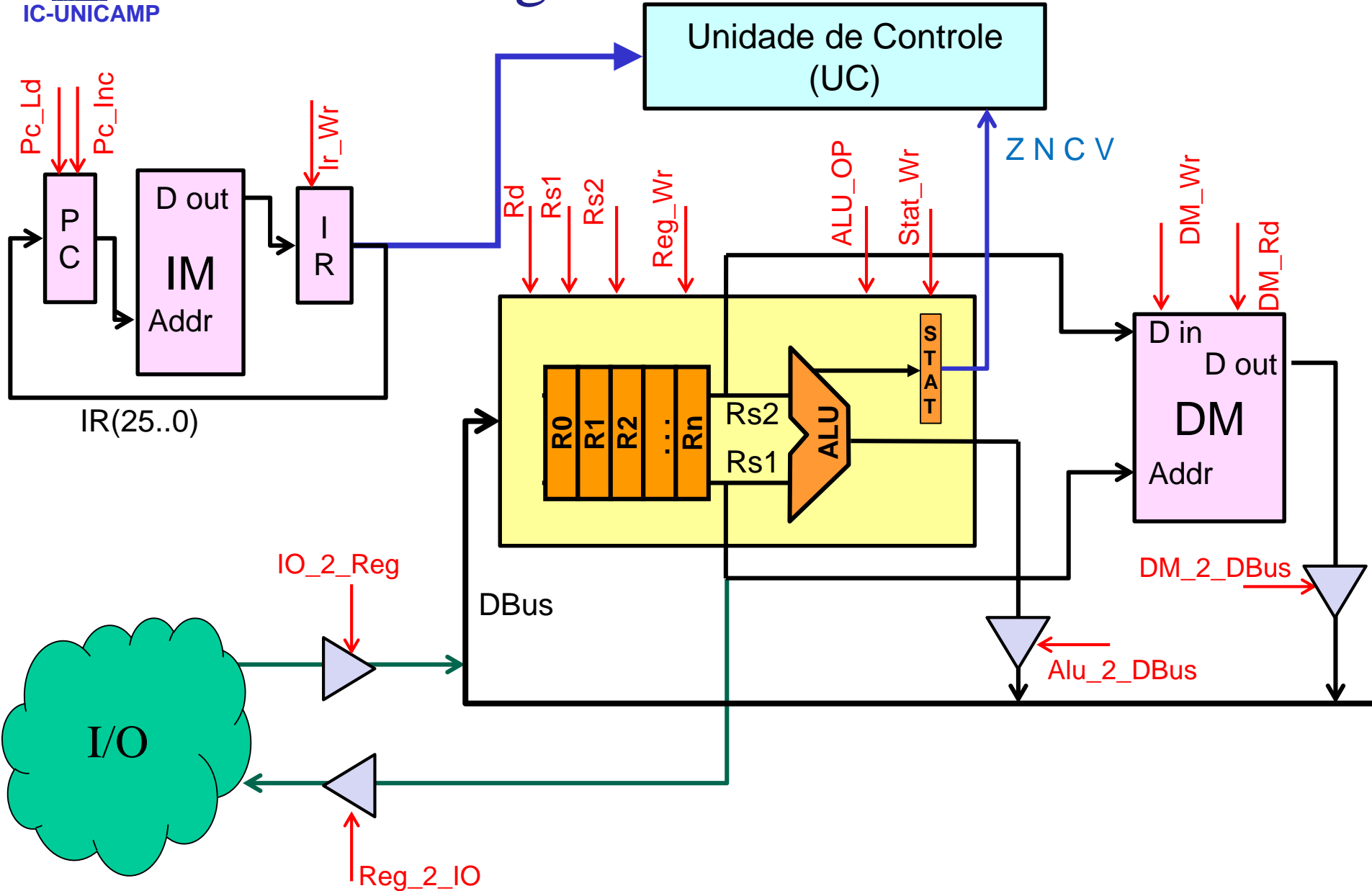


# Convenções de timing

- Controles:
  - síncronos com clock (mudança na borda de subida +  $\Delta$ )
- Escrita em registradores:
  - na próxima borda de subida (sensível à borda)
- Escrita na Data Memory
  - depende da implementação da memória



# Diagrama de blocos





# Conclusão

- Processador usa módulos básicos vistos em circuitos lógicos
  - banco de registradores
  - barramentos
  - ULA
  - unidade de controle (FSM)
  - memórias
- Detalhes de implementação: MC722
- Detalhes de linguagens de montagem: MC404