

# Princípios de Projeto em Arquitetura

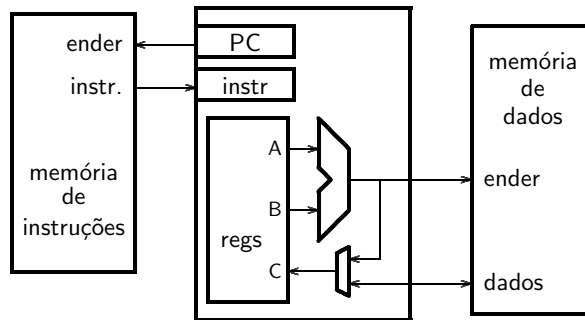
**Princípio 1:** simplicidade favorece regularidade

**Princípio 2:** menor é mais rápido (quase sempre)

**Princípio 3:** um bom projeto demanda compromissos

**Princípio 4:** o caso comum deve ser o mais rápido

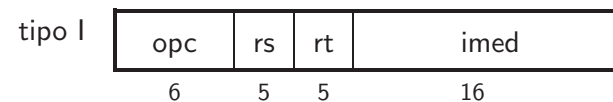
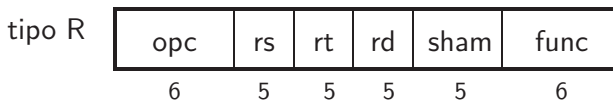
## Modelo do processador



Processador:

- |                                                 |                                       |
|-------------------------------------------------|---------------------------------------|
| 1) busca na memória a instrução apontada por PC | busca                                 |
| 2) decodifica instrução                         | decodificação                         |
| 3) executa operação                             | execução: $A + B$                     |
| 4) acesso à memória                             | memória: $mem[A + desl]$              |
| 5) armazena resultado da operação               | resultado: $regs[c] \leftarrow \dots$ |

## Formato das instruções do MIPS



- opc      operação da instrução (*opcode*)
- rs rt rd   nome dos registradores fonte e destino
- sham     *shift amount*
- func     seleciona variante da operação em **opc**
- imed     deslocamento no endereço, ou imediato/constante
- ender    destino da instrução *jump*

## Projeto do processador com ciclo longo

- Um ciclo de relógio por instrução
- cada recurso do circuito de dados usado só uma vez por instrução
- recursos que são usados mais de uma vez devem ser replicados
- cinco passos de projeto:
  1. análise do conjunto de instruções c.r.a fluxo de dados
  2. seleção de componentes
  3. circuito de dados
  4. análise do circuito de dados c.r.a fluxo de controle
  5. circuito de controle

## Princípios de Projeto – de novo, outra vez

- **Princípio 1:** simplicidade favorece regularidade
- **Princípio 3:** um bom projeto demanda compromissos
- **Princípio 4:** o caso comum deve ser o mais rápido
  
- Quais são os **casos comuns?**
- Quais são os **compromissos?**

## Projeto de CPU em 5 passos

1. Analise o conj de intruções  $\implies$  requisitos de projeto
  - \* semântica das instruções como transferências de registradores
  - \* circuito deve conter armazenadores para registradores do Cdl
  - \* circuito de dados deve permitir todas as transferências
2. selecione componentes e estabeleça a metodologia de sincronização
3. “monte” o circuito de dados que satisfaça aos requisitos de (1)
4. analise as instruções para determinar os pontos de controle que afetam as transferências de registradores
5. construa a lógica de controle

## Primeiro passo: subconjunto do Cdl do MIPS

busca            op : rs : rt : rd : sham : func  $\Leftarrow$  M[PC]  
 busca            op : rs : rt : imed  $\Leftarrow$  M[PC]

INSTRUÇÃO	DESCRIÇÃO
addu rd,rs,rt	$R[rd] \Leftarrow R[rs] + R[rt];$ PC $\Leftarrow$ PC+4
subu rd,rs,rt	$R[rd] \Leftarrow R[rs] - R[rt];$ PC $\Leftarrow$ PC+4
ori rt,rs,imed16	$R[rt] \Leftarrow R[rs] \vee zExt(im16);$ PC $\Leftarrow$ PC+4
lw rt,de16(rs)	$R[rt] \Leftarrow M[ R[rs] + sExt(de16) ];$ PC $\Leftarrow$ PC+4
sw rt,de16(rs)	$M[ R[rs] + sExt(de16) ] \Leftarrow R[rt];$ PC $\Leftarrow$ PC+4
beq rs,rt,de16	if (R[rs] $\equiv$ R[rt]) PC $\Leftarrow$ PC+4+{sExt(de16),00} else PC $\Leftarrow$ PC+4

R[] é o banco de registradores

### 1º Passo: semântica dos modos de endereçamento

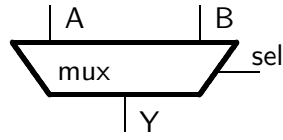
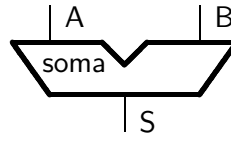
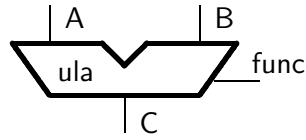
modo	ender. efetivo	exemplo
a registrador	R	add r4,r3,r2
imediatos	imed	add r4,r5,8
deslocamento	M[R+imed]	add r4,100(r1)
indireto a registrador	[R]	jr r4
absoluto	ender	j ender

### 1º passo (cont): requisitos do Cdl

- Memória
  - \* uma para instruções + uma para dados
- registradores (32 de 32bits)
  - \* ler RS
  - \* ler RT
  - \* escrever RT ou RD
- expansor do sinal (sExt) e para zero (zExt) para imediatos
- +, -,  $\vee$  reg OP reg, reg OP imediato estendido
- PC
- soma 4 ou imediato estendido ao PC

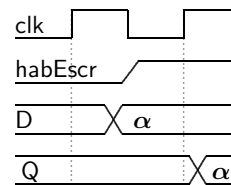
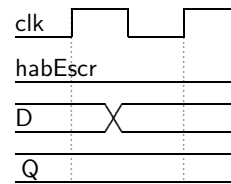
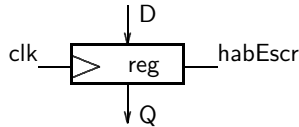
## Segundo passo: componentes do processador

- **Elementos combinacionais**
  - \* unidade de lógica e aritmética
  - \* somador
  - \* seletor, multiplexador
- Elementos de estado
- Metodologia de sincronização



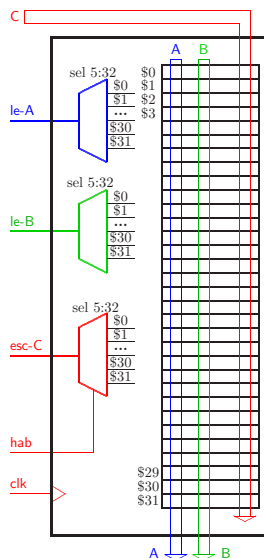
## 2º passo: componentes do processador

- Elementos combinacionais
- **Elementos de estado**
  - \* registrador
    - \* similar ao FF-D
    - \* com N-bits de entrada/saída
    - \* habilitação de escrita explícita
      - habEscr=0 → Q não muda
      - habEscr=1 → Q torna-se D na borda
  - \* contador de programa (PC)
  - \* banco de registradores
  - \* memória de instruções
  - \* memória de dados
- Metodologia de sincronização
  - \* ciclo longo

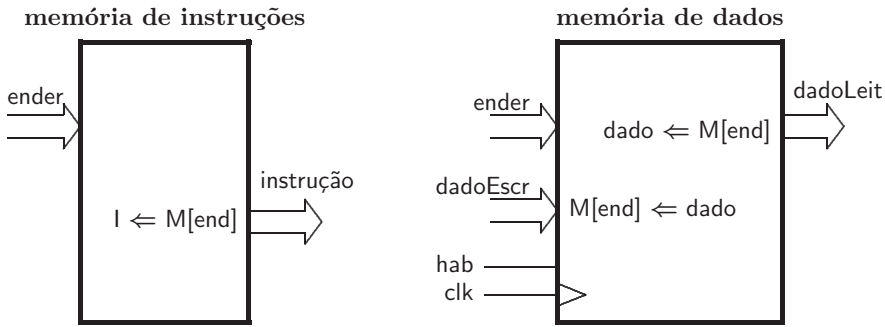


## 2º passo: banco de registradores

- 32 registradores de 32 bits
- dois barramentos de saída: A e B
- um barramento de entrada: C
- endereçamento
  - \* le-A seleciona reg para saída A
  - \* le-B seleciona reg para saída B
  - \* esc-C selec destino do valor em C
- atualiza na borda somente se hab=1
- leitura é “combinacional”:
  - le-A,B válido → valor estável na saída após tempo de acesso



## 2º passo: memória



Memória idealizada:

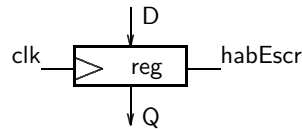
leitura “combinacional”:

endereço estável → saída estável após tempo de acesso

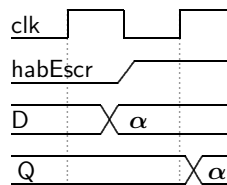
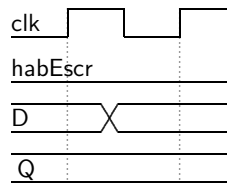
escrita síncrona:

atualiza posição endereçada na borda do relógio se hab=1

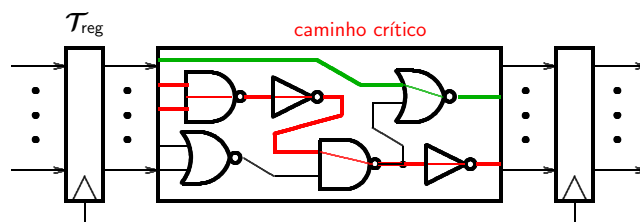
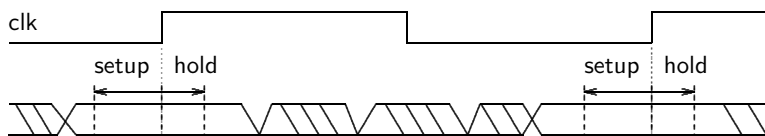
## 2º passo: metodologia de sincronização



- Elementos combinacionais
- Elementos de estado
- **Metodologia de sincronização**
  - \* ciclo longo
  - \* parâmetros de temporização:
    - setup, hold, skew

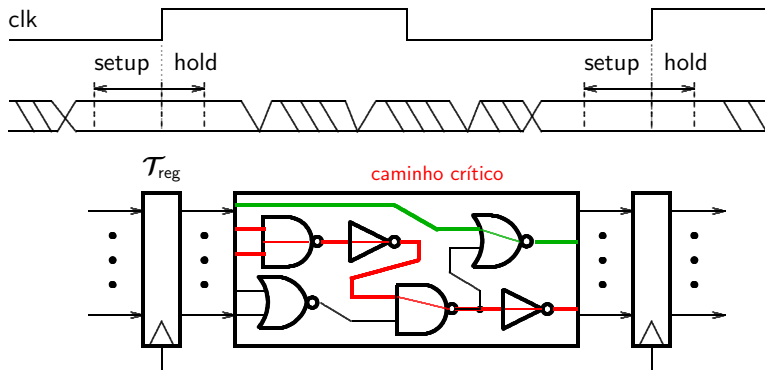


## 2º passo: metodologia de sincronização



- setup – entradas dos FFs estáveis **antes** da borda
- hold – entradas dos FFs estáveis **depois** da borda
- skew – diferença de tempo entre as bordas do relógio nos vários pontos do circuito (veloc dos bits ≈ 20cm/ns)

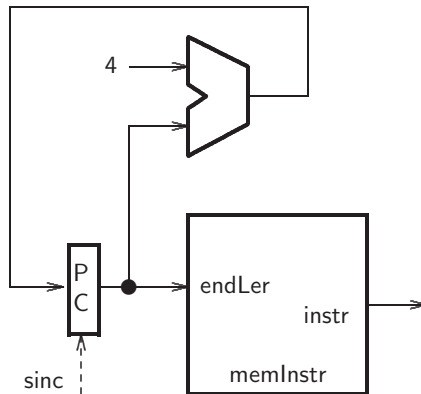
## 2º passo: metodologia de sincronização



- período  $\geq [ T_{reg} + \text{caminhoMaisLongo} + \text{setup} + \text{skew} ]$
- $[ T_{reg} + \text{caminhoMaisCurto} - \text{skew} ] > \text{hold}$

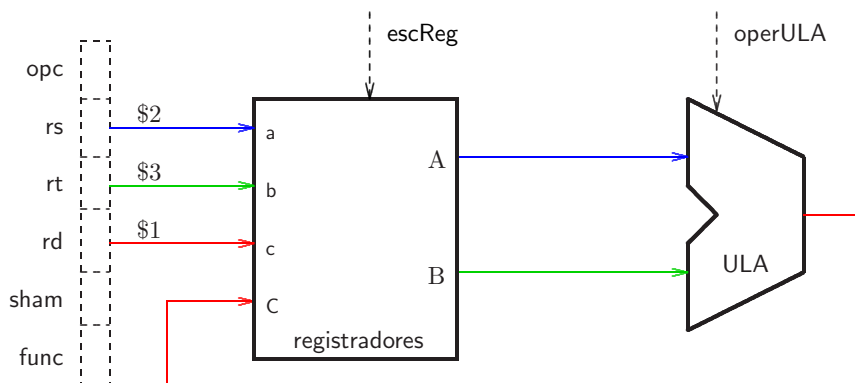
## Terceiro passo – busca de instruções

PC muda a cada pulso do relógio do processador  
 conteúdo de PC indexa memória de instruções  
 após tempo de acesso à memória, nova instrução disponível



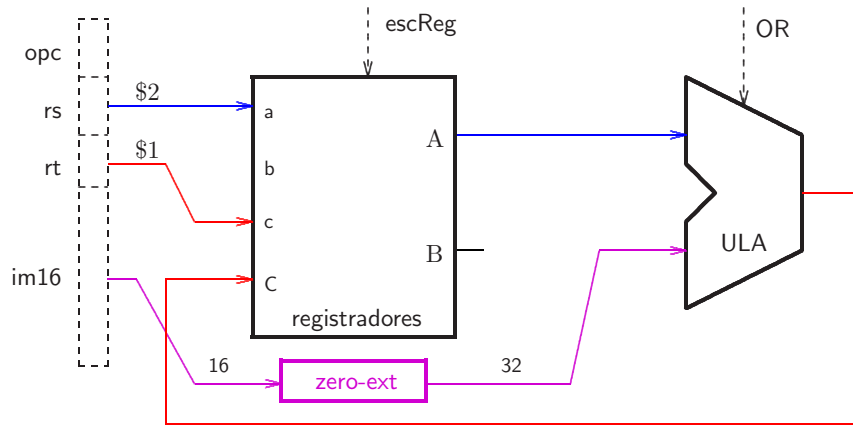
## 3º passo – operações lógicas e aritméticas

addu \$1, \$2, \$3 # \$1 ← \$2 + \$3



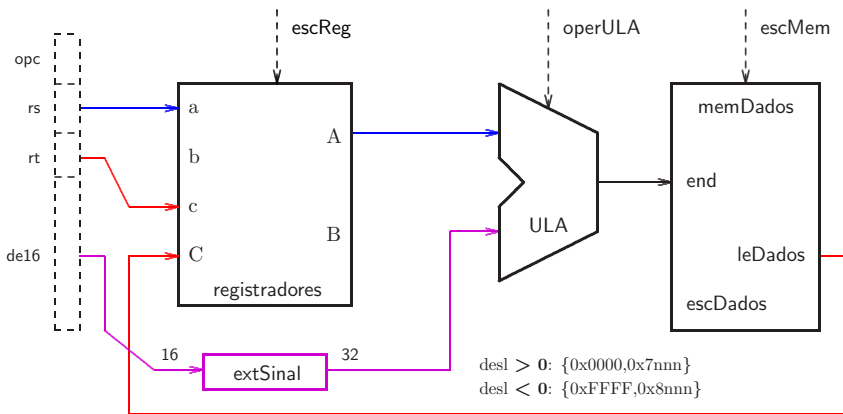
### 3º passo – operação lógica com imediato

```
ori $1, $2, im16 # $1 <- $2 OR zExt(im16)
```



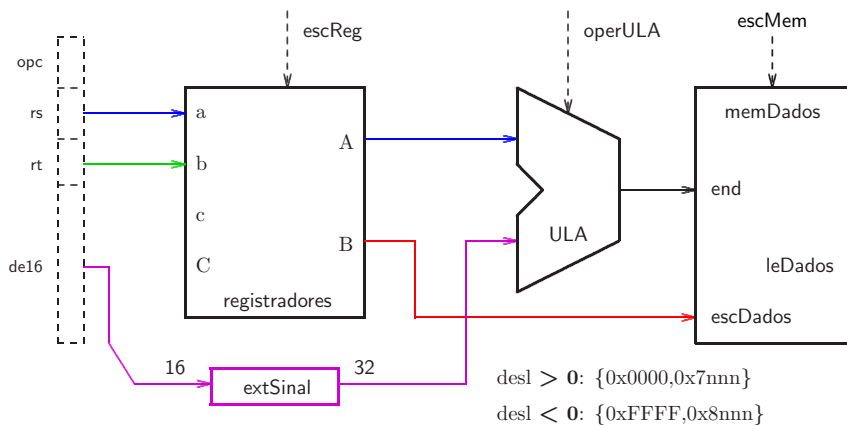
### 3º passo – operação de acesso à memória: LW

```
lw $8, de16($15) # $8 <- M[sExt(de16) + $15]
```



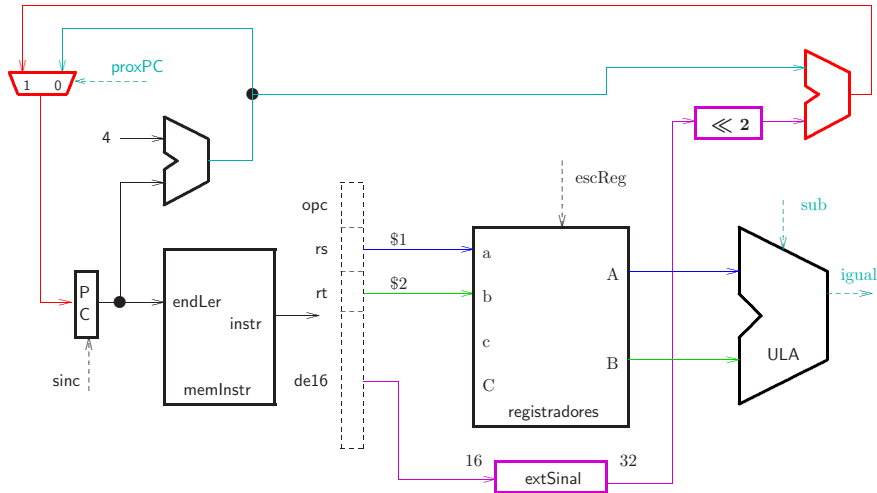
### 3º passo – operação de acesso à memória: SW

```
sw $8, de16($15) # M[sExt(de16) + $15] <- $8
```

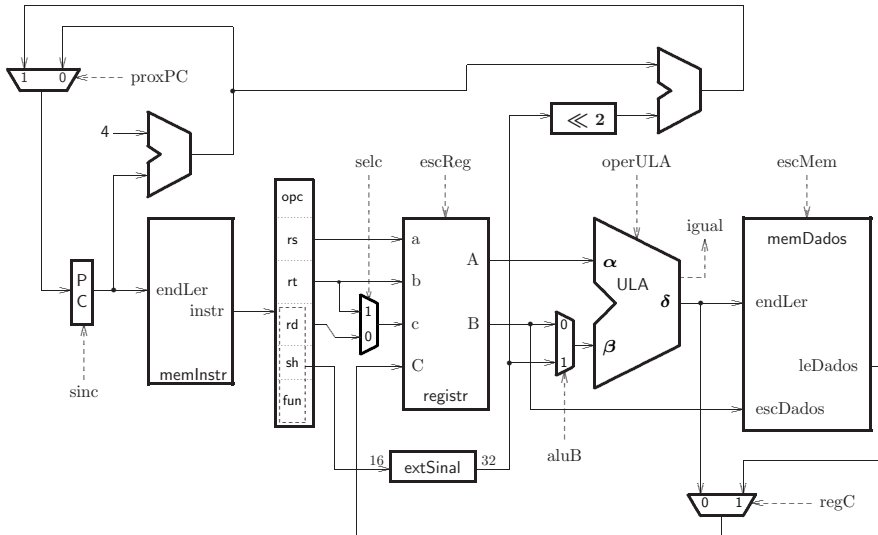


### 3º passo – desvio condicional

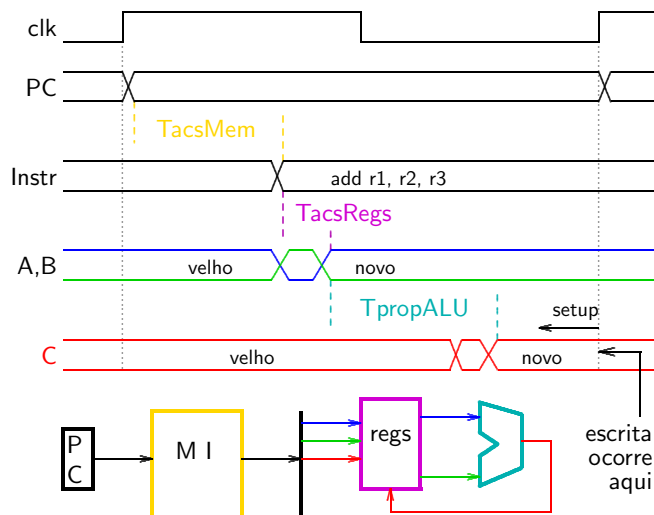
beq \$1, \$2, de16 # if (\$1==\$2) PC <- PC+4 + sExt(de16)<<2



### 3º passo – circuito de dados completo

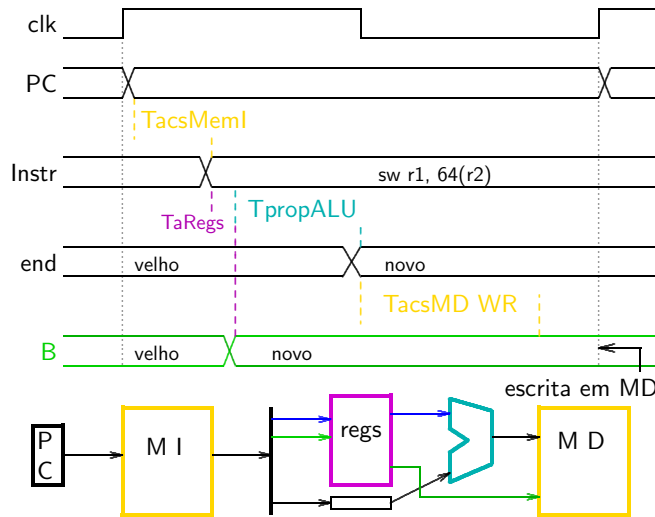


### Quarto passo – fluxo de dados: ADDU

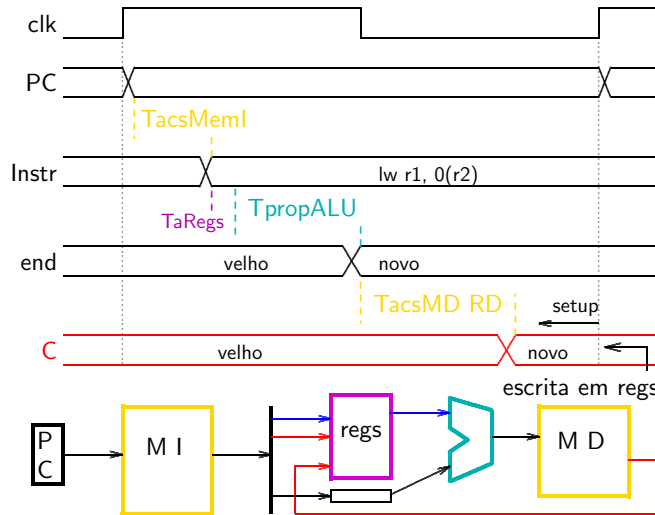




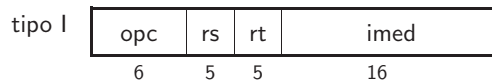
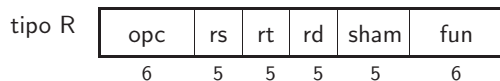
### 4º passo – fluxo de dados: SW



### 4º passo – fluxo de dados: LW



### 4º passo: subconjunto do Cdl do MIPS



- |                                                                                          |                                        |
|------------------------------------------------------------------------------------------|----------------------------------------|
| op : rs : rt : rd : sham : func $\leftarrow$ M[PC]                                       | op : rs : rt : imed $\leftarrow$ M[PC] |
| addu rd,rs,rt     R[rd] $\leftarrow$ R[rs]+R[rt];                                        | PC $\leftarrow$ PC+4                   |
| subu rd,rs,rt     R[rd] $\leftarrow$ R[rs]-R[rt];                                        | PC $\leftarrow$ PC+4                   |
| ori rt,rs,im16    R[rt] $\leftarrow$ R[rs] $\vee$ zExt(im16);                            | PC $\leftarrow$ PC+4                   |
| lw rt,de16(rs)    R[rt] $\leftarrow$ M[ R[rs] + sExt(de16) ];                            | PC $\leftarrow$ PC+4                   |
| sw rt,de16(rs)    M[ R[rs] + sExt(de16) ] $\leftarrow$ R[rt];                            | PC $\leftarrow$ PC+4                   |
| beq rs,rt,de16    PC $\leftarrow$ ((R[rs] $\equiv$ R[rt]) ? PC+4+{sExt(de16),00} : PC+4) |                                        |

## 4<sup>o</sup> passo – circuito de controle ( $\alpha$ )

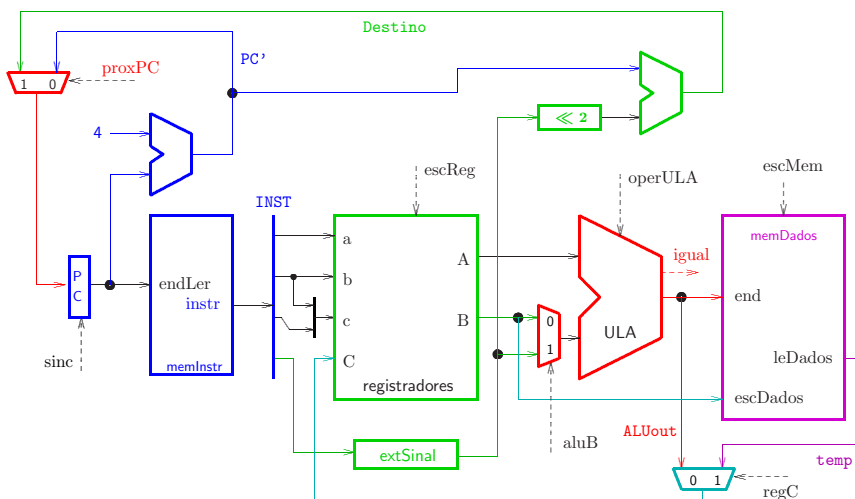
### 1. Busca instrução & incrementa contador de programa

```
INST := Mem[PC];          /* "registrador" de instrução */
PC' ← PC + 4;
```

### 2. Decodificação de instrução & acesso a registradores

```
A ← R[INST(25..21)];     /* entrada da ULA */
B ← R[INST(20..16)];     /* entrada da ULA */
Destino ← PC' + (extSinal(INST(15..0))<<2);
```

## 4<sup>o</sup> passo – circuito de controle ( $\alpha$ )



## 4<sup>o</sup> passo – circuito de controle ( $\beta$ )

### 3. Execução

```
/* execução na ULA */
```

```
ALUout ← A op B;
```

```
/* operação com constante – ori */
```

```
ALUout ← A + extZero(INST(15..0));
```

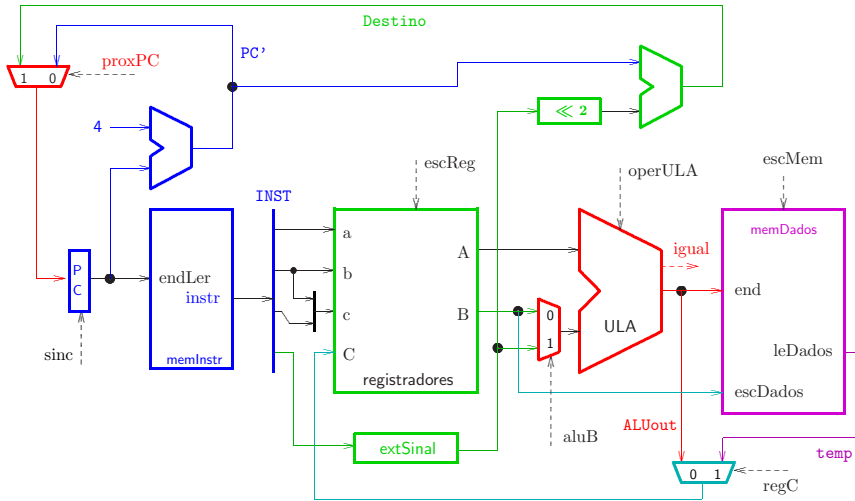
```
/* Cálculo de endereço efetivo – lw ou sw */
```

```
ALUout ← A + extSinal(INST(15..0));
```

```
/* Efetua desvio – beq */
```

```
if (igual) PC := Destino; /* igual=(A==B) */
else PC := PC';
```

### 4º passo – circuito de controle ( $\beta$ )



### 4º passo – circuito de controle ( $\gamma$ )

#### 4. Acesso à memória

```

/* Efetua acesso à memória */
lw: temp ← Mem[ALUout];
sw: Mem[ALUout] := B;
    
```

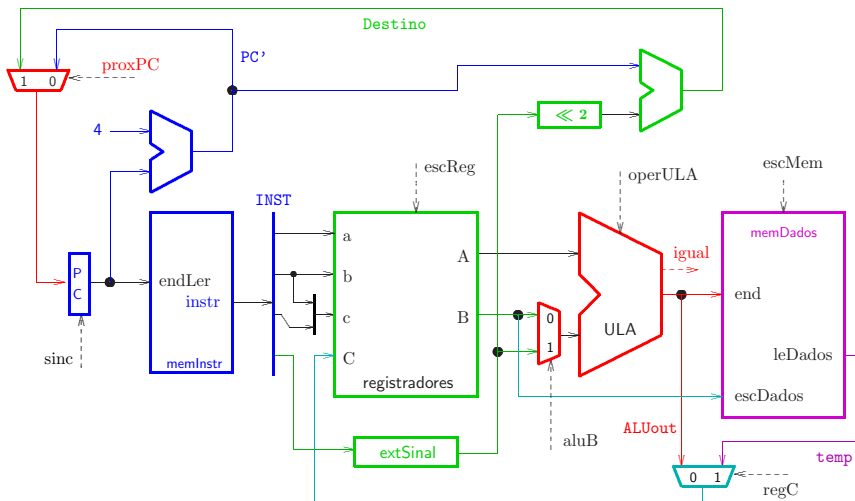
#### 5. Resultado

```

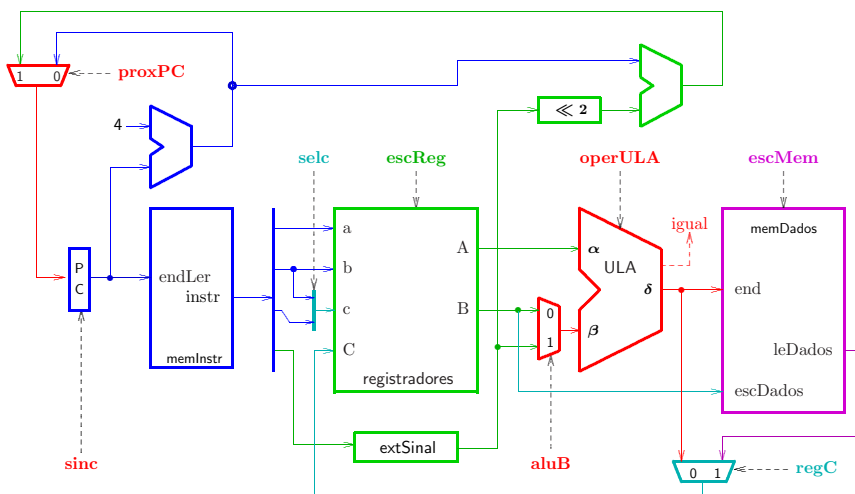
/* escreve resultado da execução na ALU */
R[INST(15..11)] := ALUout;

/* escreve resultado da busca em memória */
R[INST(20..16)] := temp; /* só em lw */
    
```

### 4º passo – circuito de controle ( $\gamma$ )



### 4º passo – sinais de controle



### Quinto passo – implementação do circuito de controle

- com base nos opcodes gera sinais de controle
- **ciclo longo:** todos os sinais ativos durante todo o ciclo
- implementação: tabela? ROM?

SINAL	sinc	proxPC	escReg	aluB	regC	opULA	escMem
busca	1	1/0	0	x	x	x	0
addu rd,rs,rt	0	x	1	0	0	fun	0
subu rd,rs,rt	0	x	1	0	0	fun	0
ori rt,rs,im16	0	x	1	1	0	✓	0
lw rt,de16(rs)	0	x	1	1	1	+	0
sw rt,de16(rs)	0	x	0	1	x	+	1
beq rs,rt,de16	0	igual	0	0	x	–	0

### 5º passo – implementação de operULA

INSTRUÇÃO	opc	fun	operação
addu rd,rs,rt	00	21	soma
subu rd,rs,rt	00	23	subtração
ori rt,rs,im16	0d	x	disjunção
lw rt,de16(rs)	23	x	soma
sw rt,de16(rs)	2b	x	soma
beq rs,rt,de16	04	x	subtração

implementação? tabela, ROM, função

## Avaliação de desempenho

TEMPO DE PROPAGAÇÃO DOS CIRCUITOS		
memória	200ps	ps = pico s = $10^{-12}$ s
ULA, somador	100ps	
registradores	50ps	

INSTRUÇÃO	UNIDADES FUNCIONAIS OCUPADAS						total
addu rd,rs,rt	mem RD	regs RD	ALU	regs WR			400 ps
subu rd,rs,rt	mem RD	regs RD	ALU	regs WR			400 ps
ori rt,rs,im16	mem RD	regs RD	ALU	regs WR			400 ps
lw rt,de16(rs)	mem RD	regs RD	ALU	mem RD	regs WR		600 ps
sw rt,de16(rs)	mem RD	regs RD	ALU	mem WR			550 ps
beq rs,rt,de16	mem RD	regs RD	ALU				350 ps

**duração mínima do ciclo: 600 ps** = 0.600 ns  
**CPI = 1.0**

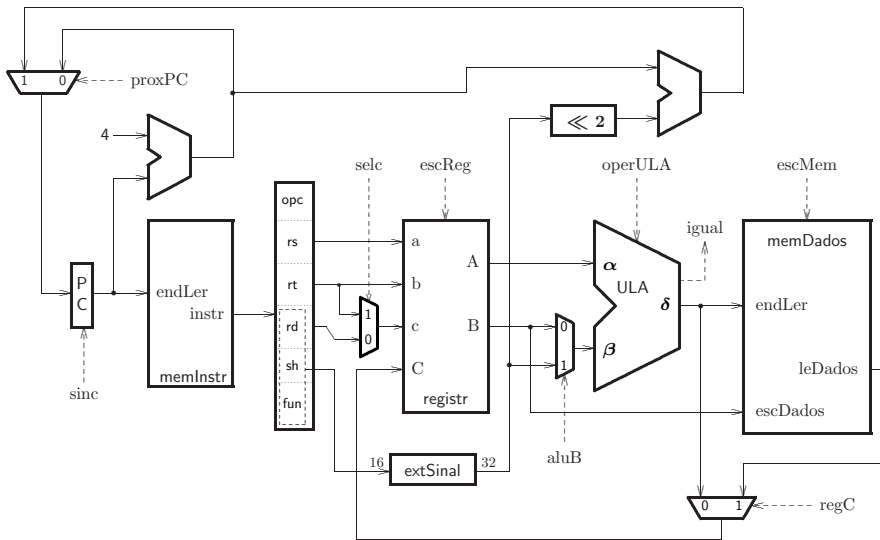
## Resumo

- Cinco passos de projeto:
  1. análise do conjunto de instruções c.r.a fluxo de dados
  2. seleção de componentes e metodologia de sincronização
  3. projeto/construção do circuito de dados
  4. análise do circuito de dados c.r.a fluxo de controle
  5. projeto/construção do circuito de controle
- regularidade do Cdl MIPS facilita/simplifica projeto
- recursos replicados para atender requisitos
- uso do ciclo longo de relógio é pouco eficiente
- **Princípio 1:** simplicidade favorece regularidade
- **Princípio 3:** um bom projeto demanda compromissos
- **Princípio 4:** o caso comum deve ser o mais rápido
- Quais são os **casos comuns?**                      Quais são os **compromissos?**

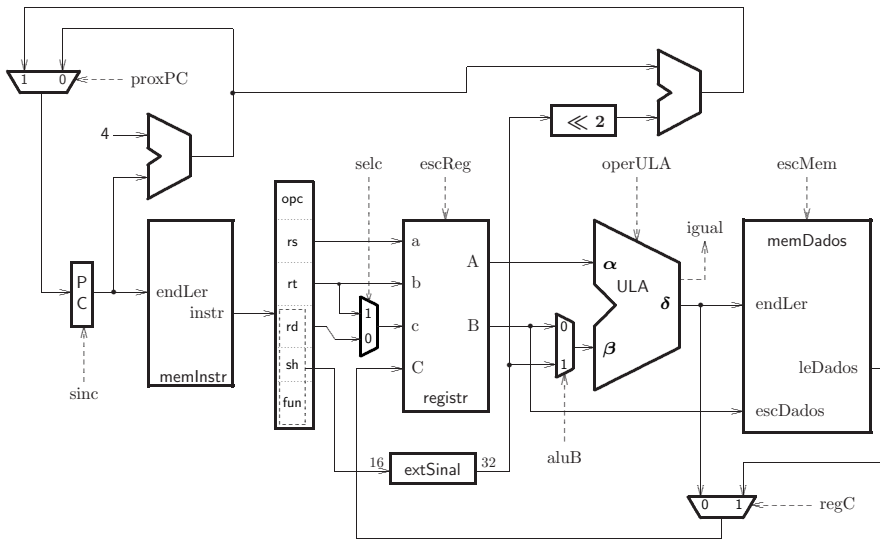
## Exercícios

- 1) Faça o projeto do circuito de dados para a instrução `jump`;
- 2) Repita para a instrução `beq`;
- 3) Repita para a instrução `jr`;
- 4) Repita para a instrução `jal`;
- 5) Para todas as instruções acima, compute o período mínimo do relógio.

### Circuito de dados completo



### Circuito de dados completo



### Circuito de dados completo

