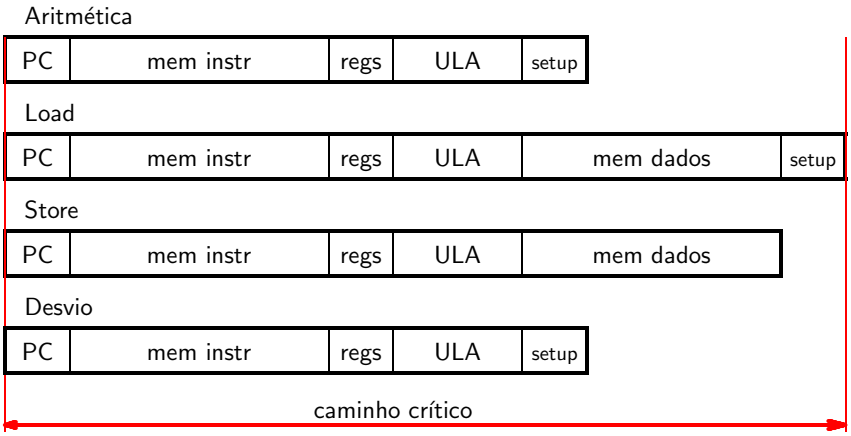
Problema com processador de ciclo longo

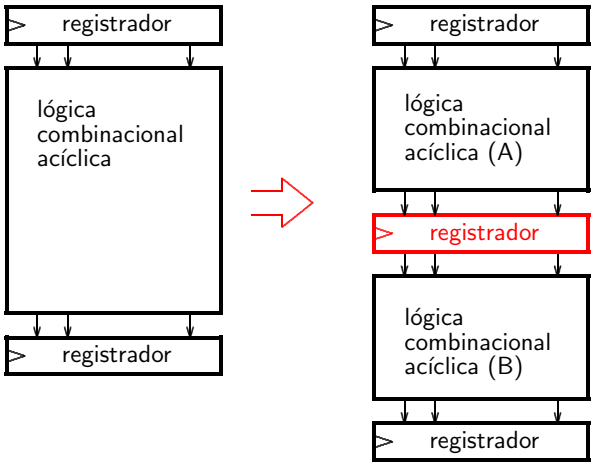


ciclo longo:

todas instruções demoram tanto quanto a mais lenta

memória real é mais lenta que a idealizada

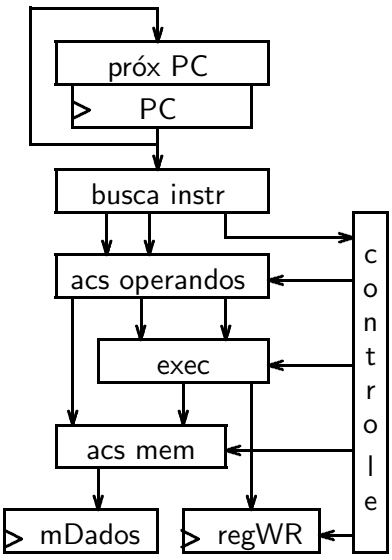
Redução no ciclo de relógio



"Corta" grafo de dependência e insere registrador

faz mesmo trabalho em dois ciclos curtos ao invés de um longo

Redução no ciclo de relógio



lógica de próxPC:
PC $\leftarrow$ desv? PC+desl : PC+4

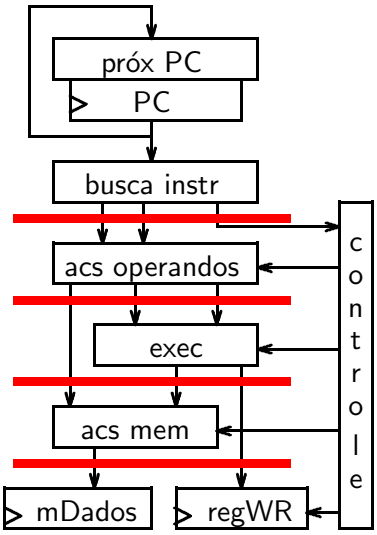
busca de instrução:
instrReg $\leftarrow$ mem[PC]

acesso aos operandos/Regs:
A $\leftarrow$ R[rs], B $\leftarrow$ R[rt]

execução:
res $\leftarrow$ A + B

acesso à memória:
M[res] $\leftarrow$ R[rt]
R[rt] $\leftarrow$ M[res]

Redução no ciclo de relógio

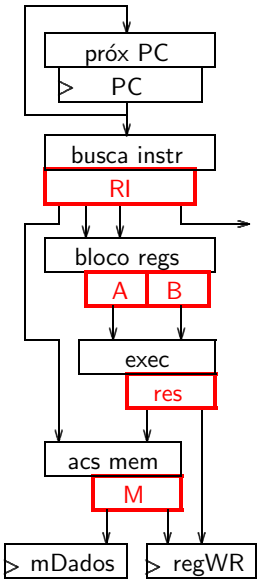


Adicionar registradores entre os menores passos

registradores inseridos para balancear duração do ciclo de relógio
→ atrasos similares entre os regs

registradores mantém informação necessária para uso posterior

Circuito de dados multiciclo



No quarto passo de projeto, as transferências entre registradores (TR) lógicas devem ser mapeadas em TR físicas

addu rd,rs,rt
# R[rd] ⇐ R[rs] + R[rt] ; PC ⇐ PC+4

busca: RI ⇐ mem[PC] ; PC ⇐ PC+4

decod: A ⇐ regs[rs] ; B ⇐ regs[rt]

exec: res ⇐ A + B

resul: regs[rt] ⇐ res

Projeto do processador multiciclo

- Vários ciclos de relógio por instrução
- recursos do circuito de dados podem ser usados mais de uma vez
- ∄ replicação de recursos
- mesmos cinco passos de projeto:
  - ∗ análise do conjunto de instruções c.r.a fluxo de dados
  - ∗ seleção de componentes e metodologia de sincronização
  - ∗ circuito de dados
  - ∗ análise do circuito de dados c.r.a fluxo de controle
  - ∗ circuito de controle

Projeto de CPU em 5 passos

1. Analise o conj de intruções $\Longrightarrow$ requisitos de projeto
   - semântica das instruções como transferências de registradores
   - circuito deve conter armazenadores para registradores do CdI
   - circuito deve conter registradores para valores intermediários
   - circuito de dados deve permitir todas as transferências
2. selecione componentes e estabeleça a metodologia de sincronização
   um ciclo curto por fase de execução
3. "monte" o circuito de dados que satisfaça aos requisitos
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 CdI do MIPS

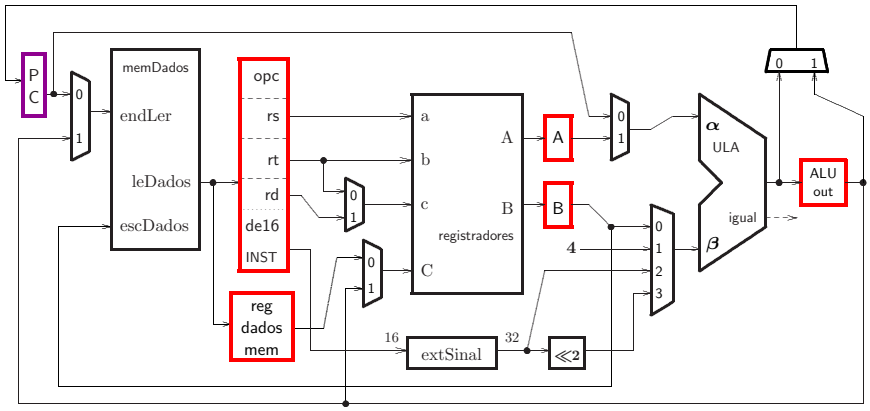
busca	op : rs : rt : rd : sham : fun $\Leftarrow$ M[PC]	
	op : rs : rt : imed $\Leftarrow$ M[PC]	
INSTRUÇÃO	DESCRIÇÃO	
addu rd,rs,rt	R[rd] $\Leftarrow$ R[rs]+R[rt];	PC$\Leftarrow$ PC+4
subu rd,rs,rt	R[rd] $\Leftarrow$ R[rs]−R[rt];	PC$\Leftarrow$ PC+4
ori rt,rs,im16	R[rt]$\Leftarrow$ R[rs] $\vee$ zExt(im16);	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	if (R[rs] $\equiv$ R[rt]) PC $\Leftarrow$ PC+4+{sExt(de16),00}	
	else PC $\Leftarrow$ PC+4	

quais as diferenças c.r. ao projeto com ciclo longo?

Segundo Passo – seleção de componentes

- Memória
  - uma porta pode ser usada para instruções e para dados
- registradores (32 de 32bits)
  - ler RS
  - ler RT
  - escrever RT ou RD — leitura e escrita em ciclos distintos
- expansor do sinal/zero (para imediato)
- PC
- ULA efetua TODAS operações lógicas/aritméticas
  - adiciona 4 ao PC
  - operações das instruções ADDU, SUBU, ORI
  - cálculo de endereço efetivo em LD, ST, BEQ
- menos componentes **E** controle mais complexo

2º Passo – circuito de dados



registradores em vermelho são temporários
que acumulam resultados parciais
estes registradores são invisíveis ao programador

2º Passo – circuito de dados

Ao final de cada ciclo do relógio, dados são armazenados em
registradores **invisíveis**, para uso nos próximos ciclos
o conteúdo destes é usado numa **mesma** instrução

Dados são transmitidos de instrução para instrução através dos **registradores visíveis** ($0..$31, PC)

INST registrador de instrução – mantém instrução após busca
regDadosMem registrador de dados da memória – mantém dado lido da memória ($\neq$ instrução)
A,B mantém operandos lidos do bloco de registradores
ALUout mantém valor computado pela ALU (≥ 1 / instrução)
todos exceto INST são atualizados na borda do relógio

2º Passo – metodologia de sincronização

- A cada ciclo de relógio pode ocorrer uma (e só uma) dentre
  acesso à memória
  operação de ULA
  acesso ao bloco de registradores
- limita o período do ciclo à latência da operação mais demorada
  que é o acesso à memória:
  200 ps – acesso à memória
  100 ps – operação de ULA
  50 ps – acesso ao bloco de registradores
- os registradores invisíveis são atualizados na borda final do ciclo;
  escrever/ler o bloco de registradores é demorado porque
  bloco de regs é uma estrutura complexa

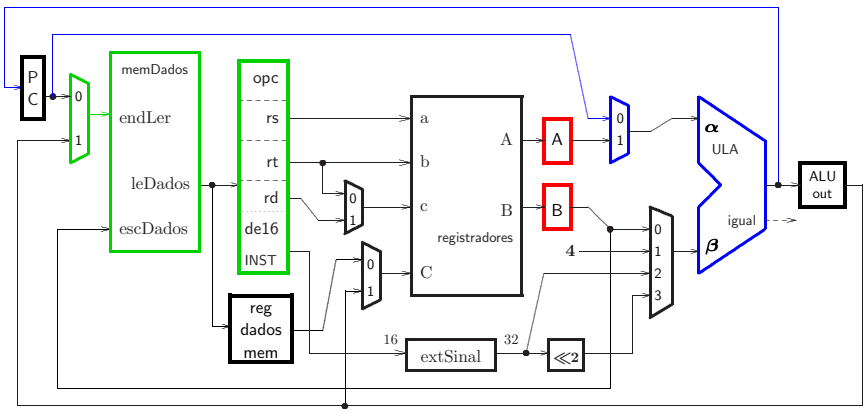
Terceiro passo – busca

Busca instrução & incrementa contador de programa

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

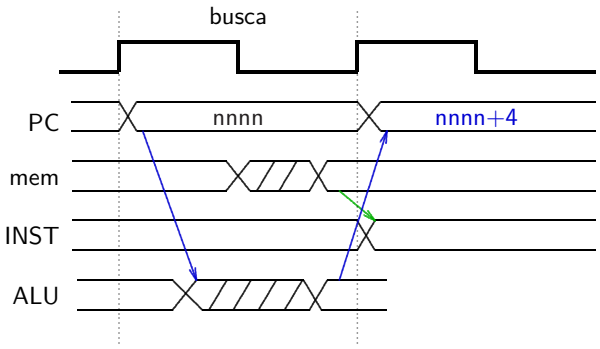
- incremento do PC:
  - entrada A da ULA recebe PC
  - entrada B da ULA recebe 4
  - ULA efetua soma
  - saída da ULA é carregada no PC e em ALUout
- carga da nova instrução:
  - registrador de instrução INST recebe saída da memória

3º passo – busca



incremento do PC
carga da nova instrução

3º passo – busca



incremento do PC
carga da nova instrução

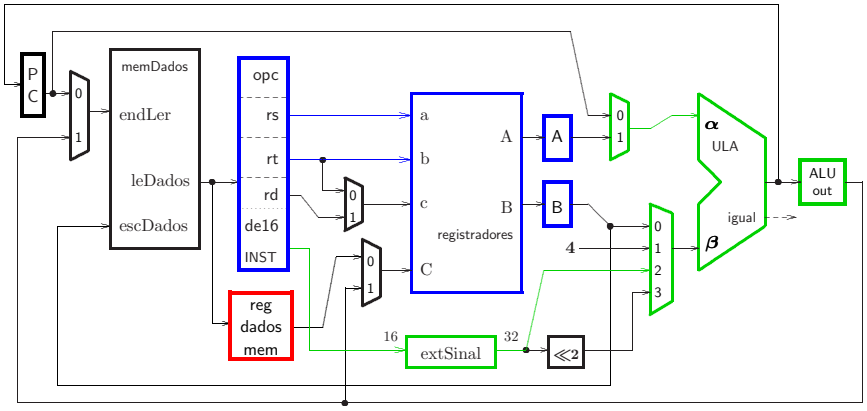
3º passo – decodificação

Decodificação de instrução & acesso a registradores

```
A ⇐ Reg[INST[25..21]];     /* entrada da ULA */
B ⇐ Reg[INST[20..16]];     /* entrada da ULA */
ALUout ⇐ PC + (extSinal(INST[15..0])<<2);
```

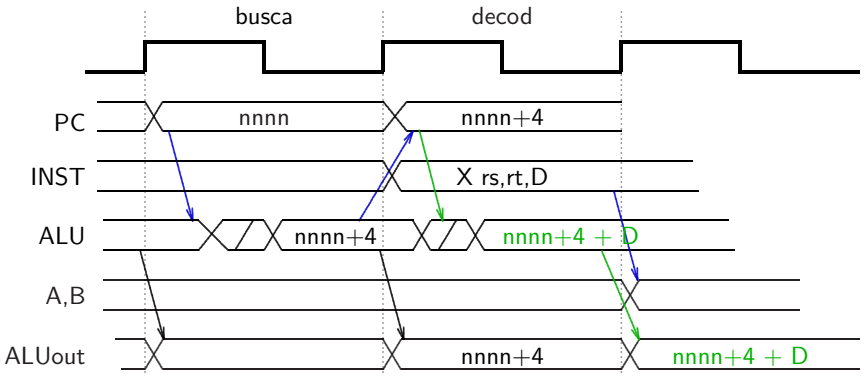
- leitura dos registradores
  - copia saída do bloco de registradores nos temporários A e B
- computa (otimisticamente) endereço destino de desvio
  - se próx intrução for BEQ, destino já estará pronto

3º passo – decodificação



leitura dos registradores
cálculo do endereço de desvio

3º passo – decodificação



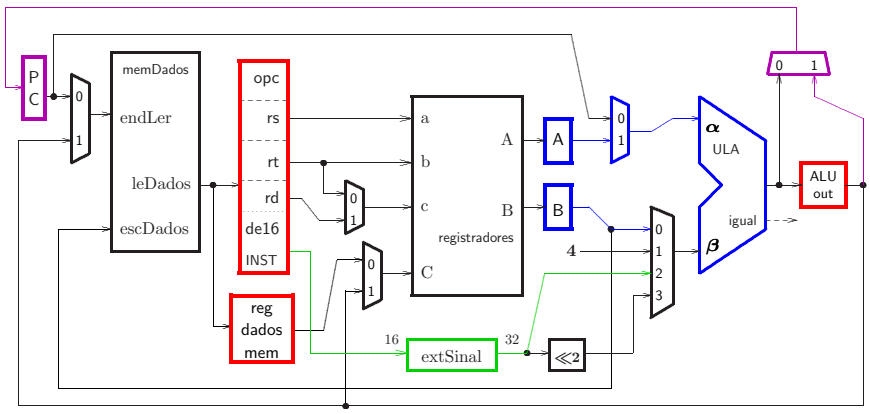
leitura dos registradores
cálculo do endereço de desvio

3º passo – execução

Execução

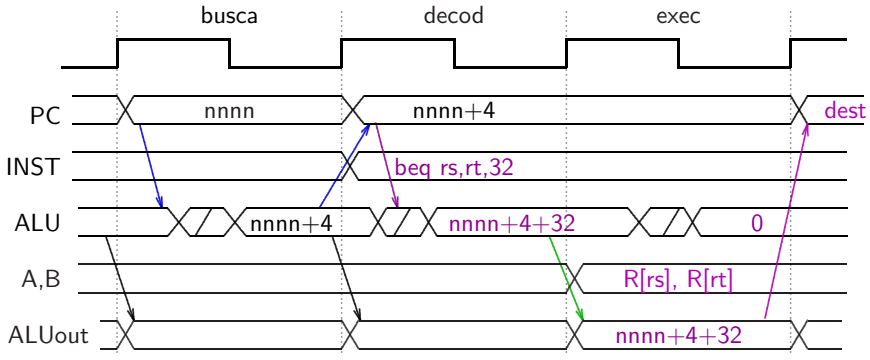
```
/* execução na ULA */
ALUout ⇐ A op B;

/* 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 ⇐ ALUout;   /* igual = (A==B) */
```


3º passo – execução



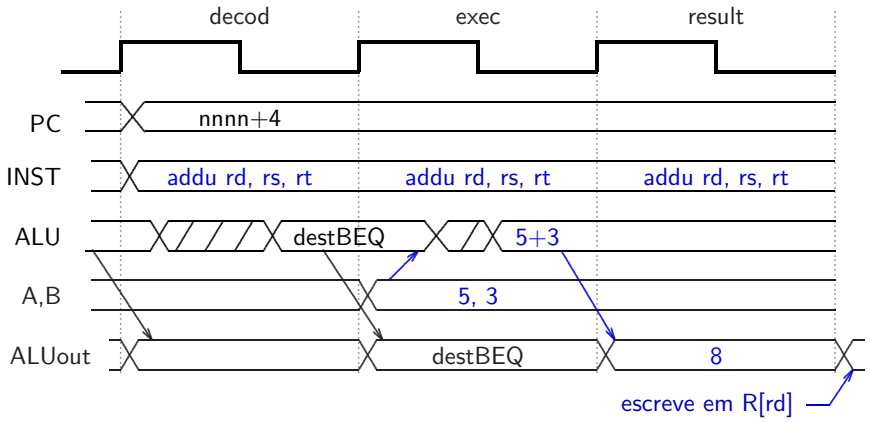
operação com registradores (tipo-R)
cálculo de endereço efetivo ou operação com imediato
carga de endereço de desvio

3º passo – execução, desvio tomado



carga de endereço de desvio

3º passo – execução, ALU



operação com registradores (tipo-R)

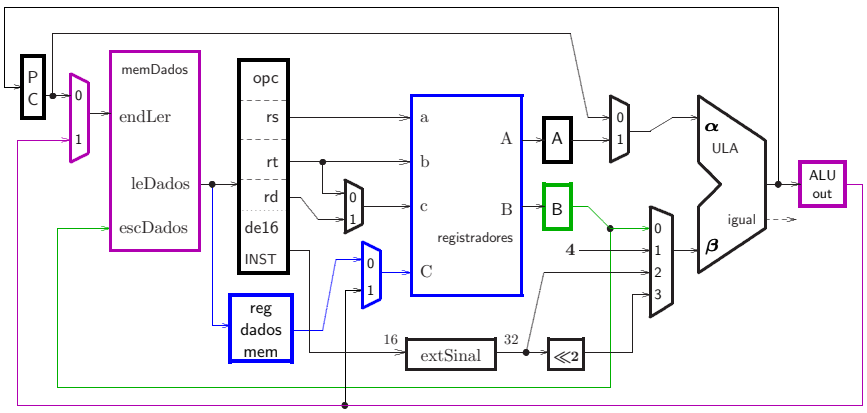
3º passo – acesso à memória

Acesso à memória

```
/* LW: acesso para leitura */
regDadosMem ⇐ Mem[ALUout];
/* SW: acesso para escrita */
Mem[ALUout] ⇐ B;
```

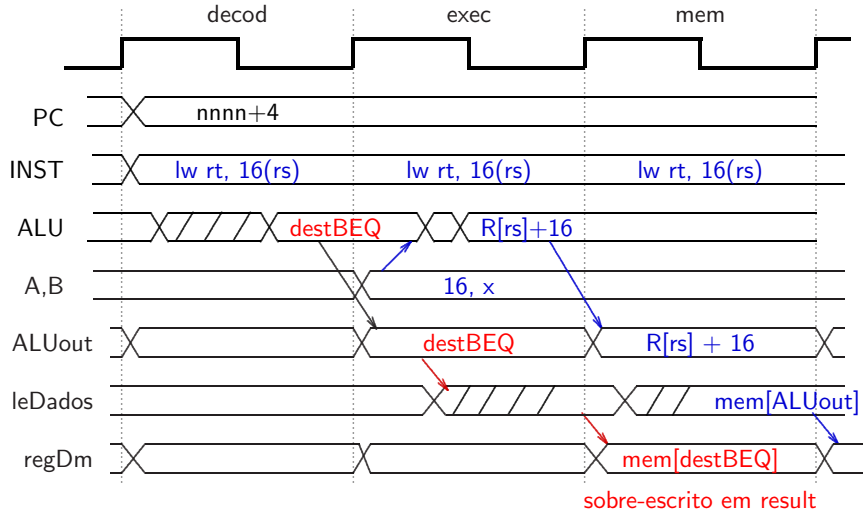
- nos dois casos ALUout contém endereço efetivo computado no estado anterior (execução)
- SW escreve direto na memória (custa um ciclo)
- LW escreve em regDadosMem porque dado chega no fim do ciclo
  → não há tempo para acessar banco de registradores

3º passo – acesso à memória



leitura – LW
escrita – SW
compartilhado em LW e SW

3º passo – acesso à memória, leitura



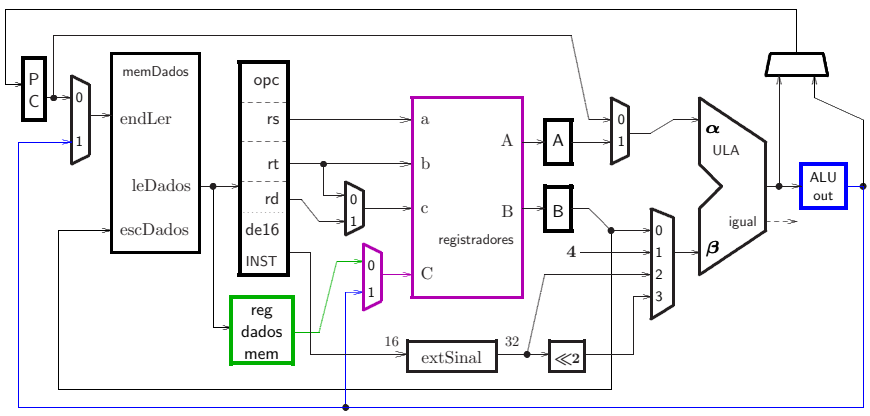
3º passo – resultado

Resultado

```
/* escreve resultado da execução na ALU */
Reg[INST[15..11]] := ALUout;

/* escreve resultado da leitura da memória */
Reg[INST[20..16]] := regDadosMem;
```

- seleciona endereço do registrador destino: $rt ou $rd
- seleciona fonte do valor: regDadosMem ou ALUout

3º passo – resultado



resultado de operação de ULA
armazena valor lido da memória – LW

Avaliação de desempenho

TEMPO DE PROPAGAÇÃO DOS CIRCUITOS		
memória	200ps	ps = pico s = 10^{-12}s
ULA, somador	100ps	
registradores	50ps	**duração mínima do ciclo: 200 ps**

INSTRUÇÃO	UNIDADES FUNCIONAIS OCUPADAS					ciclos
addu rd,rs,rt	mem RD	regs RD	ALU	regs WR		4
subu rd,rs,rt	mem RD	regs RD	ALU	regs WR		4
ori rt,rs,im16	mem RD	regs RD	ALU	regs WR		4
lw rt,de16(rs)	mem RD	regs RD	ALU	mem RD	regs WR	5
sw rt,de16(rs)	mem RD	regs RD	ALU	mem WR		4
beq rs,rt,de16	mem RD	regs RD	ALU			3

CPI = ?

CPI do processador multiciclo

Ciclos por Instrução = $\sum_{i=1}^{5}$ freqInstr$_i$ × númEstados$_i$

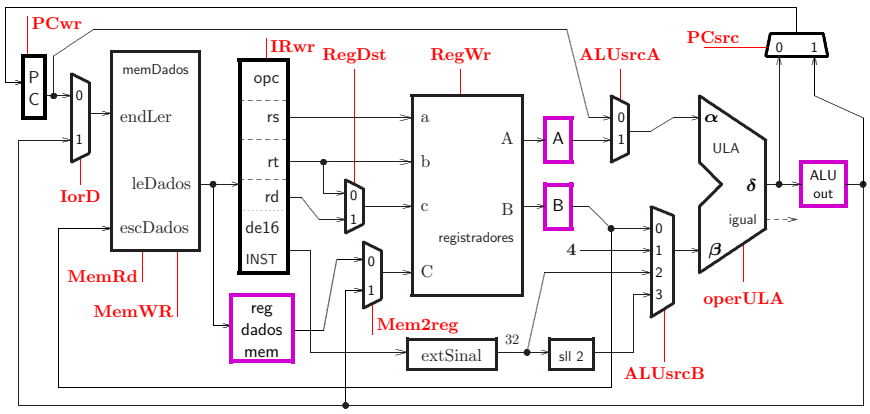
Freqüência de execução de instruções

- lw ≈ 20–30%
- sw ≈ 5–10%
- beq ≈ 20%
- alu ≈ 30%
- ori ≈ 5%

número de ciclos por instrução: [lw: 5, sw: 4, beq: 3, alu: 4, ori: 4]

$$\begin{aligned} CPI &= 0.30 * 5 + 0.10 * 4 + 0.20 * 3 + 0.30 * 4 + 0.05 * 4 \\ &= 3.9 \text{ ciclos por instrução} \\ &\leq 5 \text{ ciclos por instrução} \end{aligned}$$

Quarto Passo – Pontos de Controle

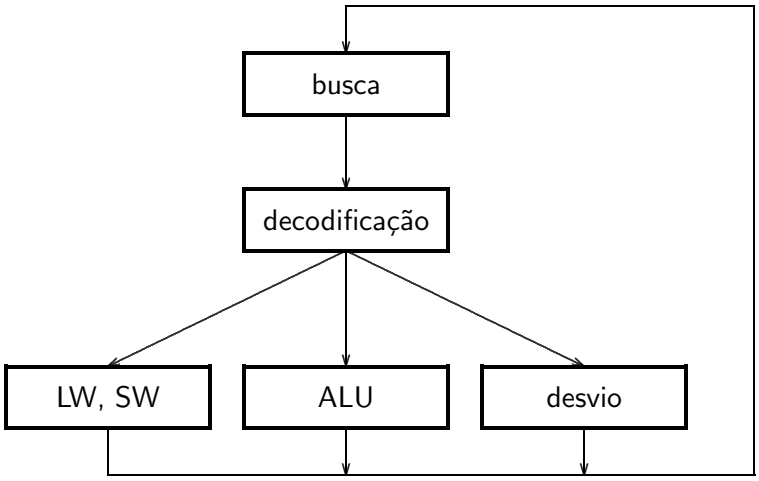


Quinto Passo – Implementação do Controle

Implementação:

- máquina de estados
- microprograma

5º Passo – Controle por Máquina de Estados



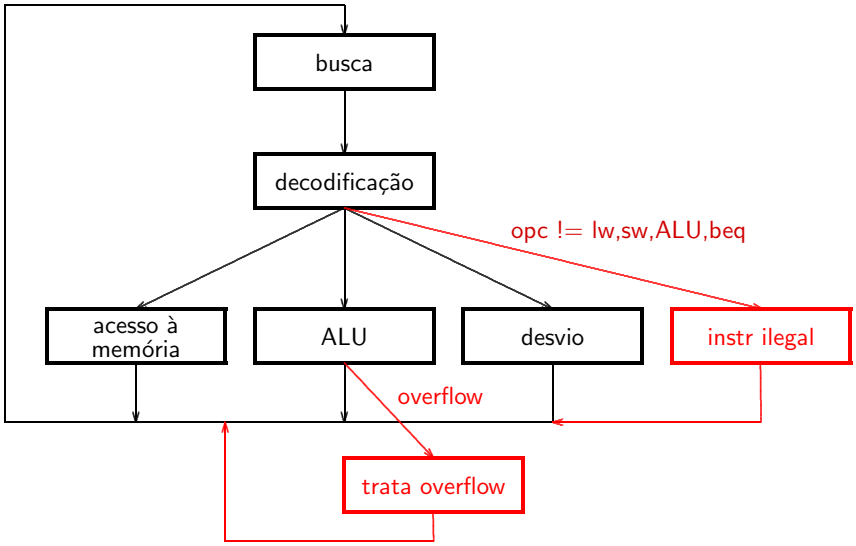
Exercícios

1. Desenhe diagramas de tempo detalhados para todos os estados de todas as instruções; ok, não precisa repetir busca e decodificação
2. Desenhe os diagramas de estado com todos os sinais de controle ativos em cada estado. Estes diagramas são consistentes com os diagramas de tempo?
3. Suponha que o relógio tem ciclo de 100 ps ao invés de 200 ps — os acessos à memória duram dois ciclos ao invés de um. Qual o efeito disso no CPI e no desempenho do processador?

Controle – Excessões

- Excessões – eventos internos ao processador
  - ∗ overflow
  - ∗ instrução indefinida (opcode inválido)
  - ∗ condição detectada **durante** execução da instrução
- Interrupções – eventos externos ao processador
  - ∗ periféricos necessitam atenção
  - ∗ condição detectada **entre** a execução de duas instruções
- podem ser benignas (falta de página, interrupção)
  trata evento e continua execução
- ou malignas (opcode inválido, overflow)
  talvez aborta programa

Controle – Excessões



Controle – Excessões

- Tratamento
  - ∗ executa chamada de função assíncrona — tratador
    para tratar evento causador da condição interrupt handler
  - ∗ salta para rotina de tratamento através de um
    vetor de endereços de tratadores de excessões
    - ▷ vetor contém um ponto de entrada para cada rotina/evento
    - ▷ tratador salva PC da instrução causadora (EPC)
      e registrador de status/causa (Cause)
    - ▷ se possível, retorna para ponto onde execução foi interrompida
    - ▷ senão, aborta programa

Resumo

- Cinco passos de projeto:
  - ⋆ análise do conjunto de instruções c.r.a fluxo de dados
  - ⋆ seleção de componentes e metodologia de sincronização
  - ⋆ projeto/construção do circuito de dados
  - ⋆ análise do circuito de dados c.r.a fluxo de controle
  - ⋆ projeto/construção do circuito de controle
- regularidade do CdI MIPS facilita/simplifica projeto
- recursos não são replicados → controle mais complexo
- uso de núm variável de ciclos por instr é mais eficiente
- Excessões são eventos internos ao processador
  - ⋆ detectadas durante a execução
  - ⋆ tratamento através do vetor de interrupções
  - ⋆ benignas ou malignas

Processador ciclo longo × multiciclo

Considerando mesmo programa que no slide 29,
qual o desempenho relativo dos dois processadores?

tempo de CPU = núm de instr × CPI × ciclo de relógio

T_{cl} = 1.0 × 600ps = $\mathbf{600 \cdot 10^{-12}}$ ciclo longo

T_m = 3.9 × 200ps = $\mathbf{780 \cdot 10^{-12}}$ multiciclo

T_m / T_{cl} = (NI × CPI_m × Rel_m) / (NI × CPI_m × Rel_{cl})

processador ciclo longo é 780/600 = 1.3 vezes mais rápido

▷ grande diferença entre velocidade do
dispositivo mais + lento (mem) e mais rápido (regs)
▷ multiciclo é pessimista porque todos os ciclos duram 200ps
▷ monociclo é pessimista porque considera pior tempo 600ps

Circuito de dados completo