Motivação

Avaliação de desempenho

- Consumidor → mede para comprar
- arquiteto → mede para projetar
- ferramentas disponíveis aos dois:

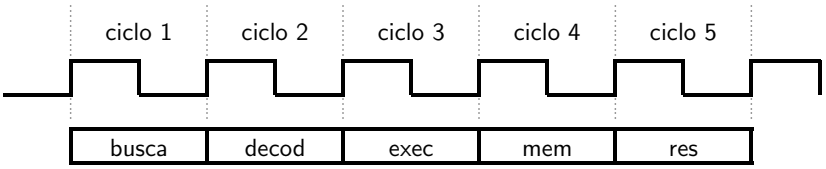
equação do desempenho:

$$\frac{\text{segs}}{\text{prog}} = \frac{\text{instr}}{\text{prog}} \times \frac{\text{ciclos}}{\text{instr}} \times \frac{\text{segs}}{\text{ciclo}}$$

Lei de Amdahl → balanceamento:

$$\text{ganho}_{\text{total}} = \frac{1}{1 - (\text{fração}_{\text{afetado}} \;/\; \text{ganho}_{\text{afetado}})}$$

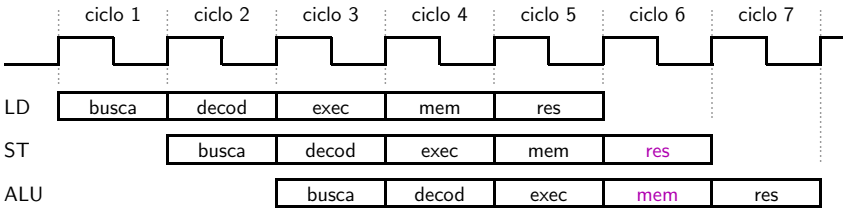
Instrução LOAD em cinco estados



busca	busca instrução e incrementa PC
decod	decodifica instrução e acessa/busca registradores
exec	executa ALU; calcula endereço efetivo
mem	lê/escreve dados na/da memória
res	escreve resultado no bloco de registradores

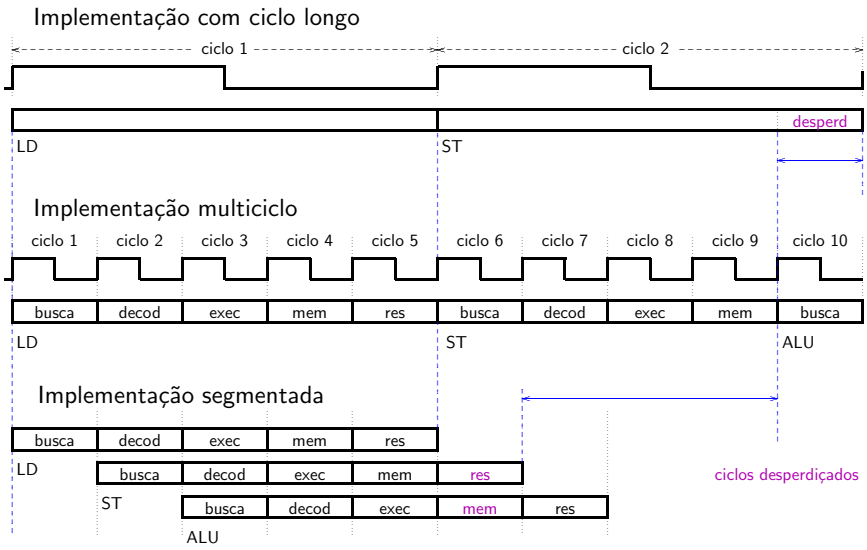
Processador MIPS Segmentado

- Inicia a próxima instrução enquanto executa a corrente
  - melhora vazão ou produção [instr/seg] ou [instr/ciclo]
    mais trabalho completado por unidade de tempo
  - latência da instrução não diminui [tempo] do início ao final

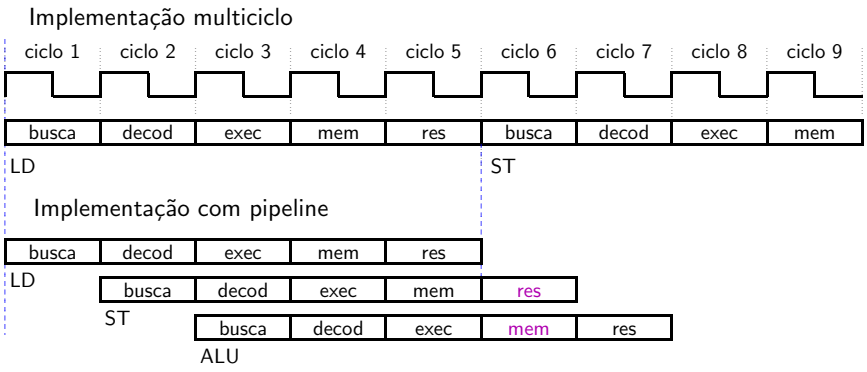


- ciclo do processador é limitado pelo componente mais lento
- para algumas instruções, estágios podem ser desperdiçados

Ciclo longo *vs* Multiciclo *vs* Segmentação



Multiciclo *vs* Segmentado – vazão e latência



latência para LW é cinco ciclos nas duas implementações

vazão 1 LW por ciclo para segmentado CPI=5

1 LW em 5 ciclos para multiciclos CPI=1

segmentação melhora a vazão e não a latência das instruções

Implementação segmentada do CdI MIPS

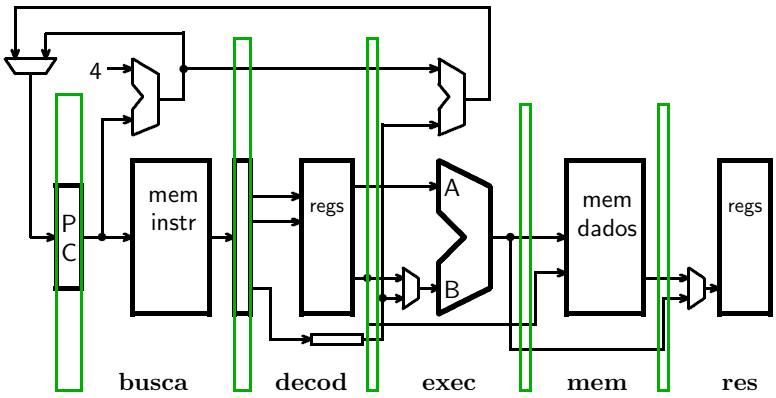
- **O que facilita**
  1) instruções de mesmo tamanho = 32 bits
     decodificação no $2^{\underline{o}}$ estágio
  2) três formatos similares de instruções
     acesso aos registradores no $2^{\underline{o}}$ estágio
  3) acessos à memória somente com LDs e STs
     cálculo do endereço efetivo no $3^{\underline{o}}$ estágio
  4) todos operandos alinhados em memória (ender%4 = 0)
  5) instrução escreve um resultado (máx), nos últimos estágios
- **O que dificulta**
  - ⋆ **riscos estruturais:** conflitos no uso de recursos
  - ⋆ **riscos de controle:** o que acontece com desvios?
  - ⋆ **riscos de dados:** operandos de uma instrução dependem de resultado/s produzido/s por instrução mais antiga

Implementação segmentada do CdI MIPS

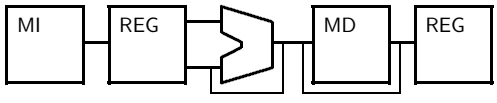
- **O que facilita**
  1) instruções de mesmo tamanho = 32 bits
     decodificação no $2^{\underline{o}}$ estágio
  2) três formatos similares de instruções
     acesso aos registradores no $2^{\underline{o}}$ estágio
  3) acessos à memória somente com LDs e STs
     cálculo do endereço efetivo no $3^{\underline{o}}$ estágio
  4) todos operandos alinhados em memória (ender%4 = 0)
  5) instrução escreve um resultado (máx), nos últimos estágios
- **O que dificulta**
  - ▷ **riscos estruturais:** conflitos no uso de recursos
  - ▷ **riscos de controle:** o que acontece com desvios?
  - ▷ **riscos de dados:** operandos de uma instrução dependem de resultado/s produzido/s por instrução mais antiga

Circuito de dados segmentado

Mudanças e acréscimos ao circuito de dados multiciclo:
registradores entre os estágios para isolar instruções ≠s



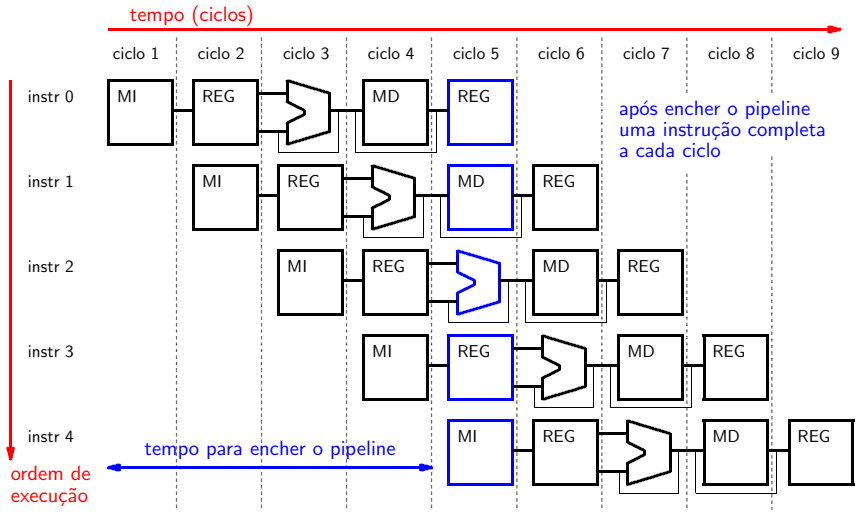
Representação gráfica do processador



Esta representação ajuda a responder questões como:

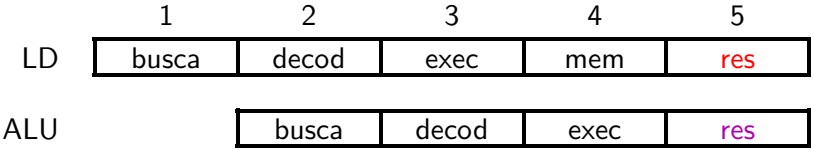
(a) quantos ciclos demora para executar uma instrução?
(b) o que está acontecendo com a ULA no $4^{\underline{o}}$ ciclo?
(c) se existe um risco, por que ocorre, e como pode ser removido?

Segmentação aumenta a vazão



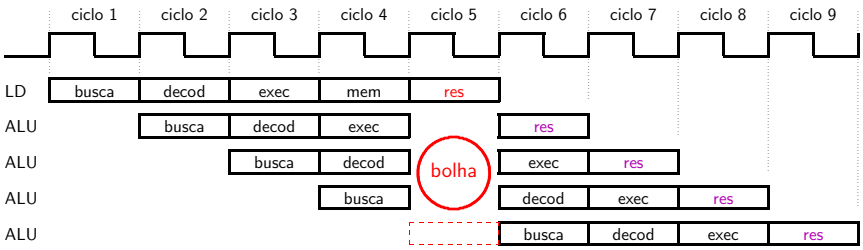
Riscos Estruturais

- Cada unidade funcional pode ser usada uma vez em cada instrução
  → porque há 4 outras instruções executando
- se uma unidade funcional é usada em estágios distintos
  então pode ocorrer **risco estrutural**:
  * LOAD usa porta de escrita dos registradores no 5º estágio
  * instruções de ALU usam a porta de escrita no 4º estágio



- Há duas maneiras de resolver este risco

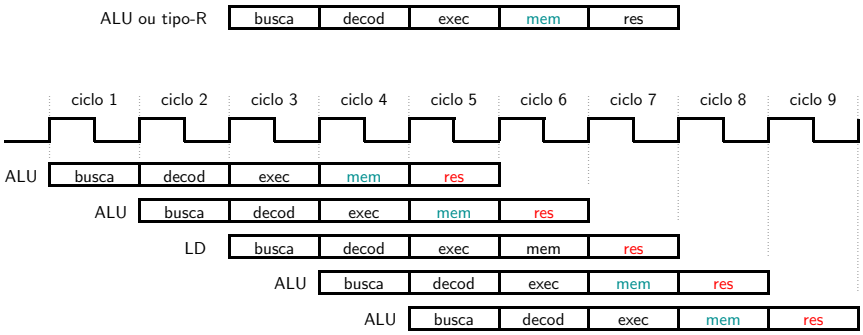
Risco Estr – solução 1: insere uma bolha



- insere uma bolha no segmento final para impedir duas escritas no mesmo ciclo
  → a lógica de controle pode ficar complexa
  → perde a oportunidade de buscar nova instrução
- nenhuma instrução inicia no quinto ciclo
  → mas resolve o risco

Risco Estr – solução 2: atrasa a escrita de um ciclo

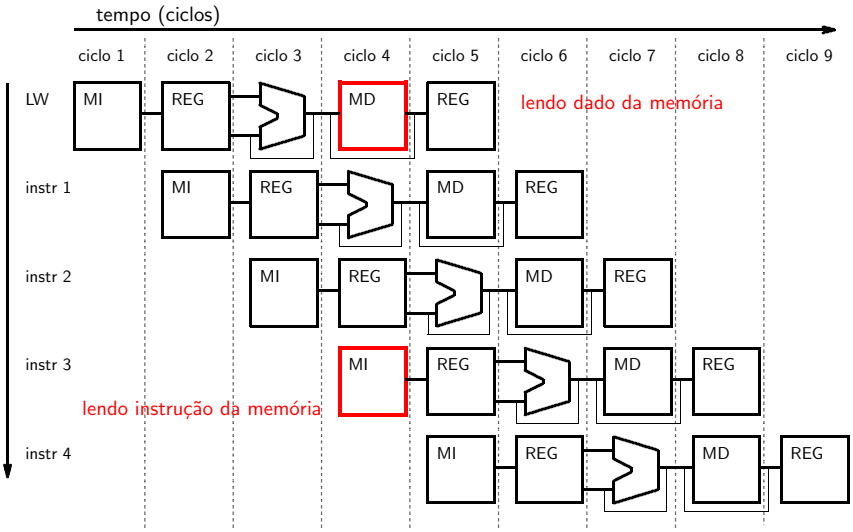
- Atrasa a escrita da instrução do tipo-R (ALU) por um ciclo
  → instruções de ALU tbém usam porta de escrita no 5º estágio
  → estágio de memória é um NOP – nada acontece



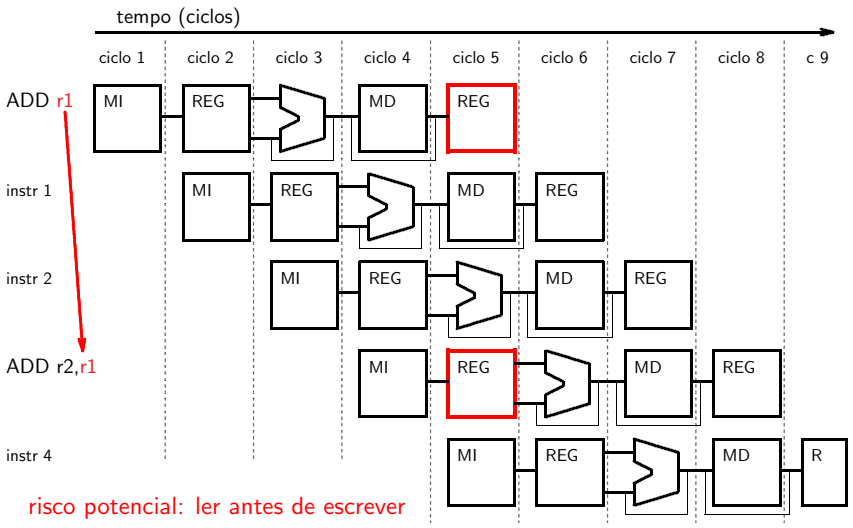
Problemas causados pela segmentação

- **Riscos**
- RISCOS ESTRUTURAIS: tentativa de usar mesmo recurso ao mesmo tempo por duas instruções
- RISCOS DE DADOS: tentativa de usar dado antes de estar disponível
  - ⋆ os operandos da instrução consumidora não estão nos registradores fonte
  - ⋆ LOAD seguido por instr de ULA que usa valor do LOAD como operando
- RISCOS DE CONTROLE: tentativa de tomar decisão antes que a condição seja avaliada
  - ⋆ instruções de desvio
- **Resolução**
- sempre pode resolver riscos com espera *stall*
  - ⋆ controle do processador deve detectar o risco
  - ⋆ fazer algo (ou não fazer nada) para resolver o risco

Risco Estrutural – somente uma memória

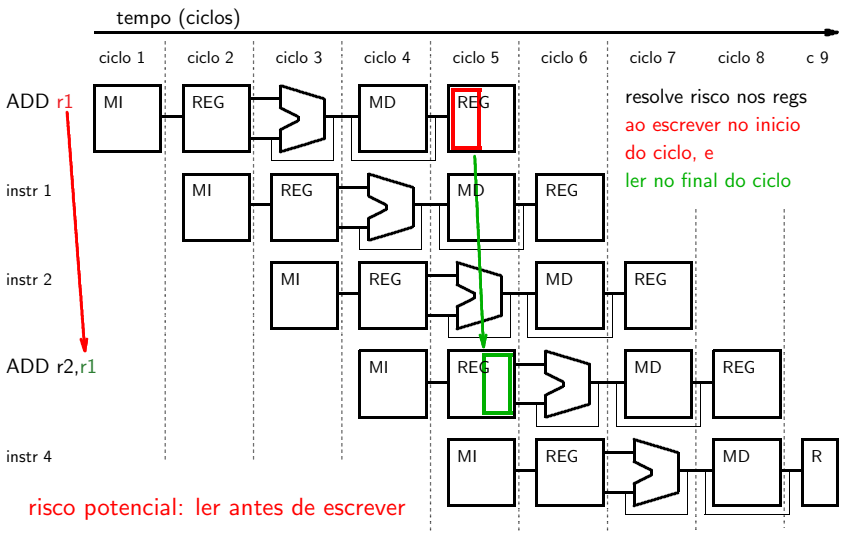


Risco de Dados – acesso aos registradores



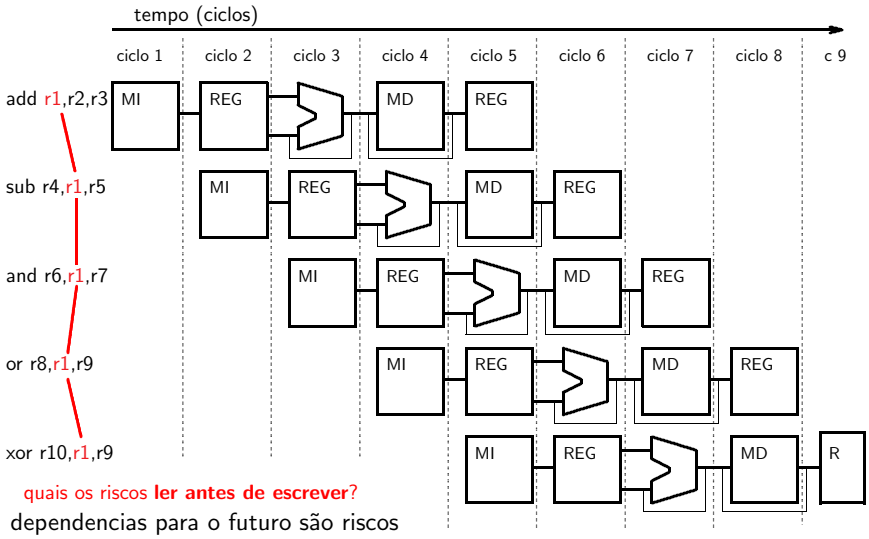
risco potencial: ler antes de escrever

Risco de Dados – acesso aos registradores



risco potencial: ler antes de escrever

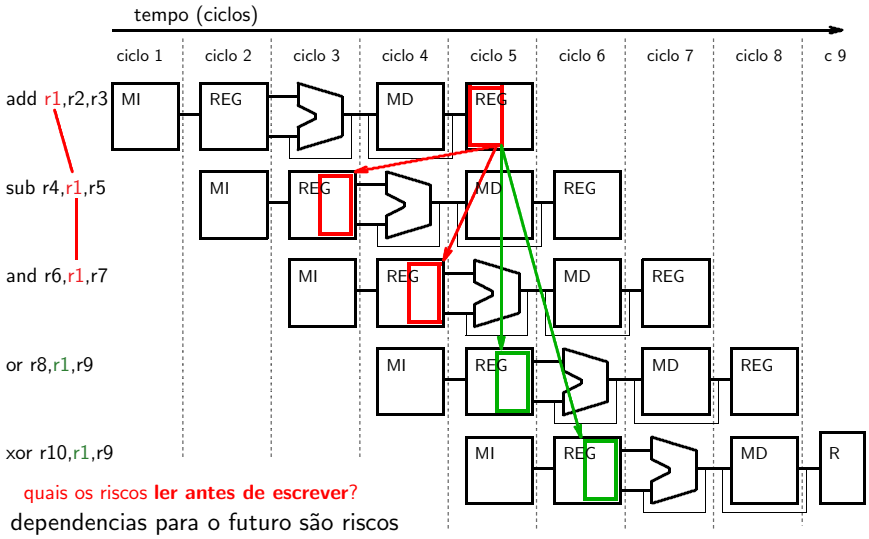
Risco de Dados – uso dos registradores



quais os riscos **ler antes de escrever**?

dependencias para o futuro são riscos

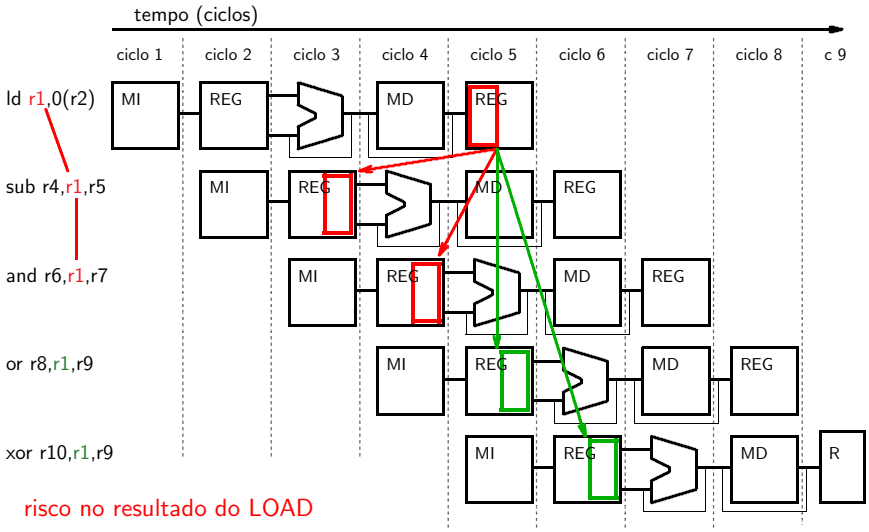
Risco de Dados – uso dos registradores



quais os riscos **ler antes de escrever**?

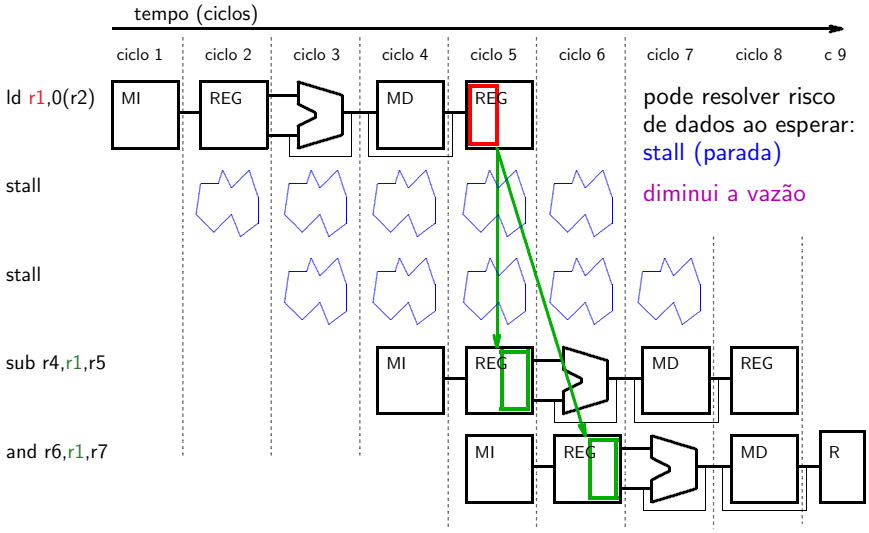
dependencias para o futuro são riscos

Risco de Dados – LOAD pode causar riscos



risco no resultado do LOAD

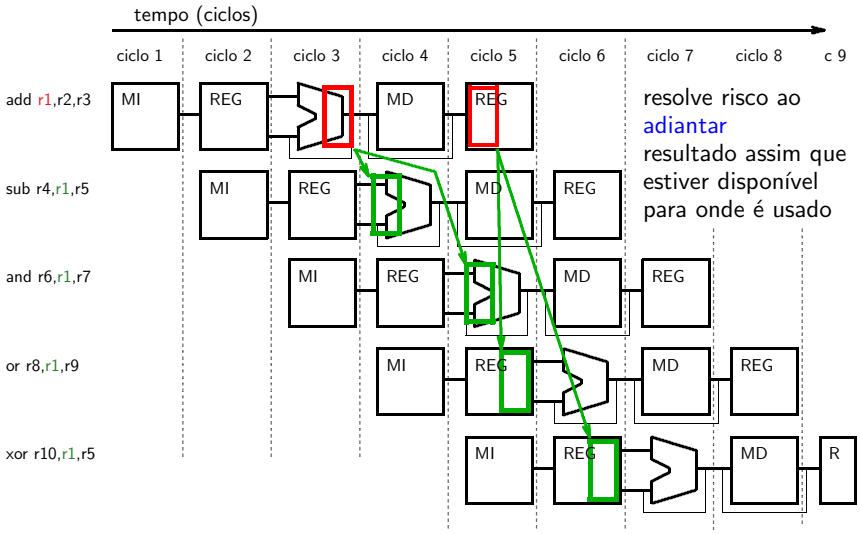
Risco de Dados – uma solução: bloqueio *(stall)*



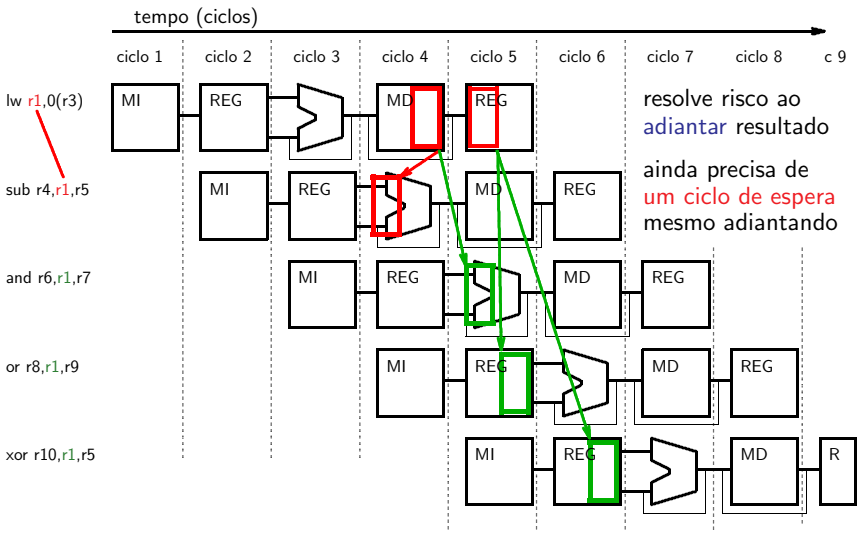
pode resolver risco de dados ao esperar: stall (parada)

diminui a vazão

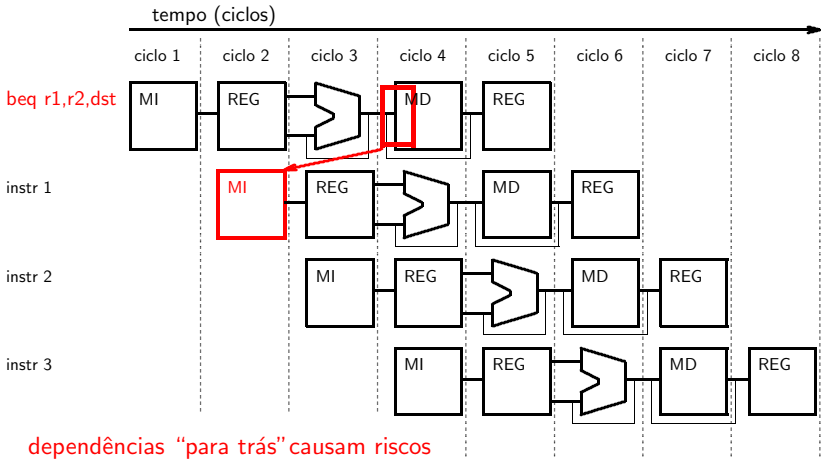
Risco de Dados – outra solução: adiantamento



Risco de Dados – adiantamento para LOADs

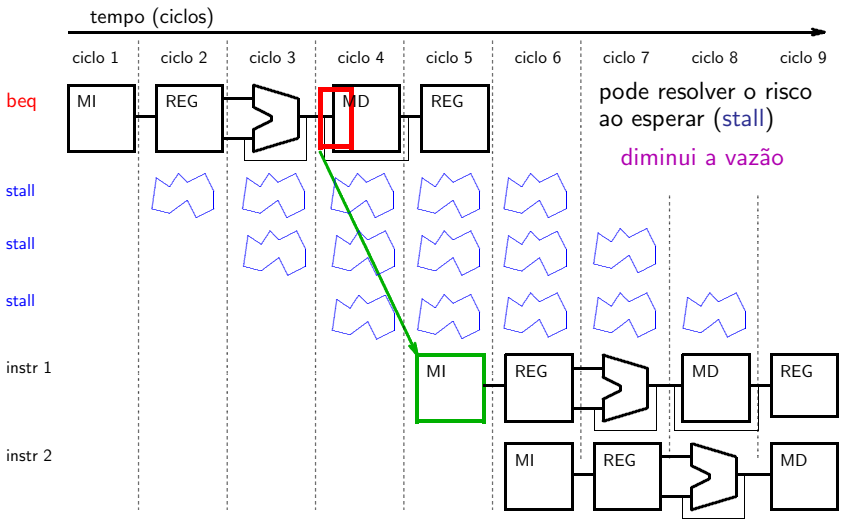


Risco de Controle – causados por desvios



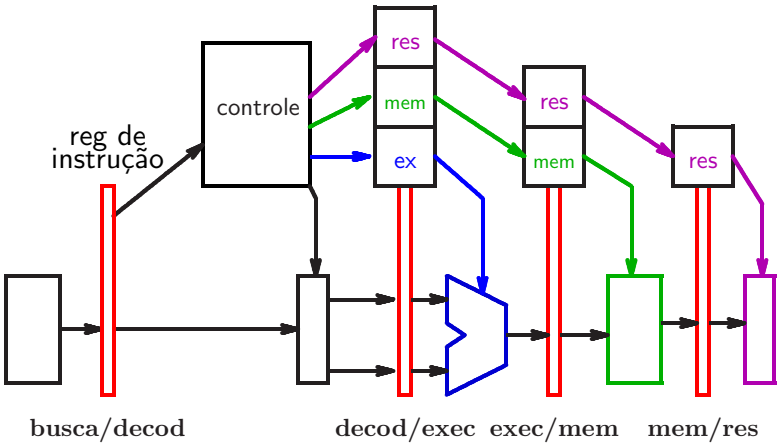
dependências "para trás" causam riscos

Risco de Controle – solução: bloqueio *(stall)*

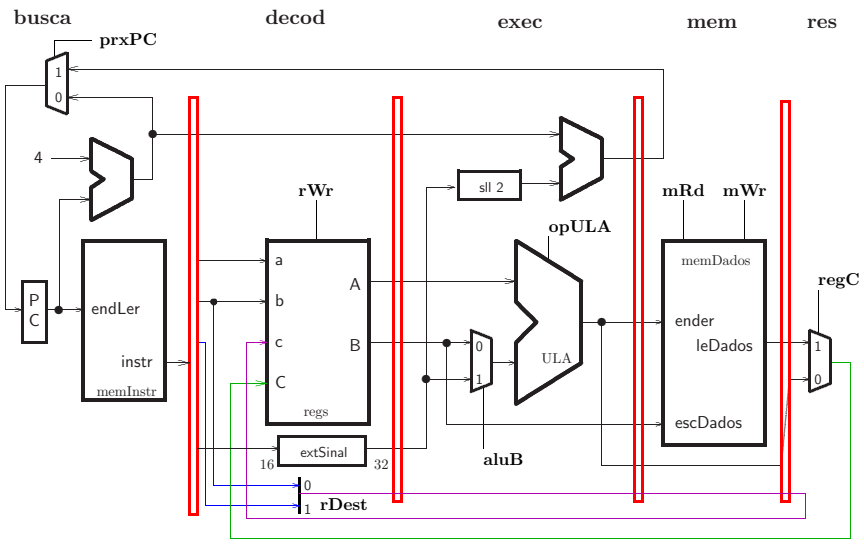


Controle em Processador Segmentado

Todos os sinais de controle são determinados na decodificação e mantidos nos registradores entre os estágios



Controle em Processador Segmentado (cont)



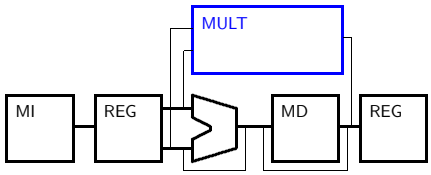
Sinais de controle do processador segmentado

	exec			mem			res	
	rDest	opULA	aluB	prxPC	mRd	mWr	rWr	regC
ALU	1	fun	0	0	0	0	1	0
IMM	0	oper	1	0	0	0	1	0
lw	0	+	1	0	1	0	1	1
sw	x	+	1	0	0	1	0	x
beq	x	−	0	1	0	0	0	x

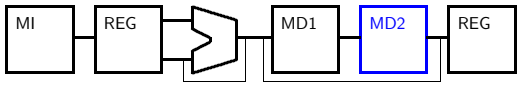
registradores dos segmentos são atualizados a cada ciclo
busca e **decod**: sempre busca instrução e incrementa PC

Outras estruturas são possíveis

- O que fazer com operações lentas (multiplicações)?
  → executa em dois ciclos



- O que fazer se o acesso à memória de dados é duas vezes mais lento que o acesso à memória de instruções?
  → reduza a velocidade do relógio para a metade, ou...
  → o acesso à memória dura dois ciclos e mantém relógio



Projeto do processador segmentado em 5+1 passos

1) Analise o conj de intruções $\Longrightarrow$ requisitos de projeto
   semântica das instruções → transferências de registradores

2) selecione componentes e estabeleça a metodologia de sincronização
   (a) associe recursos com estados → estágios
   (b) deve garantir que não há riscos estruturais: um uso por ciclo

3) projete um circuito de dados que satisfaça aos requisitos
   Lei de Amdahl recomenda dividir o estágio mais longo

Projeto do processador segmentado em 5+1 passos

4) analise as instruções para determinar os pontos de controle
   que afetam as transferências de registradores
   resolva todas as dependências de dados e de controle
   se dependência para trás no desenho, c.r. a registradores
   → risco de dados: adiantamento ou bloqueio para resolver
   se dependência para trás no desenho, c.r. ao PC
   → risco de controle → será visto adiante

5) lógica de controle:
   sinais de controle ativos nos estágios/ciclos adequados

6) invente seqüências de teste que ajudem a descobrir problemas
   sem testar, não vai funcionar

Resumo

- Todos os processadores modernos usam segmentação
- ganho potencial: número de estágios CPI: 3-5→ 1
- segmentação não reduz a latência de uma instrução
  mas aumanta a vazão/produção do programa inteiro
  → várias tarefas em execução simultânea usando recursos distintos
- vazão dos segmentos limitada pelo estágio mais lento
  estágios desbalanceados reduzem ganho
  tempo para encher e para drenar segmentos reduz ganho
- controle deve detectar e resolver **riscos**
  bloqueios afetam vazão negativamente
- próxima aula: controle dos segmentos (e de riscos)