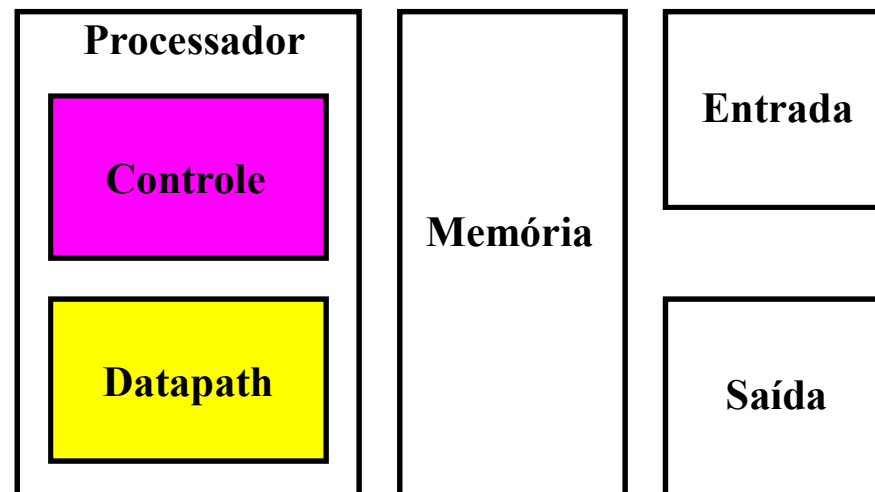


# Pipelining avançado – Parte 1



# ILP

- “Instruction-level parallelism”
  - Paralelismo entre instruções
  - Como aumentá-lo ?
- Solução 1: “**Super-pipelining**”
  - Aumentar número de estágios do pipeline
    - » Mais operações simultâneas
    - » Menor T
- Solução 2: “**Multiple issue**”
  - Disparar mais instruções em cada estágio
    - »  $CPI < 1$

# Panorama do uso de pipeline

Microprocessor	Year	Clock Rate (MHz)	Pipeline Stages	Issue Width	Cores/ Chip
Intel 486	1989	25	5	1	1
Intel Pentium	1993	66	5	2	1
Intel Pentium Pro	1997	200	10	3	1
Intel Pentium Willamette	2001	2000	22	3	1
Intel Pentium Prescott	2004	3600	31	3	1
Intel Core	2006	2930	14	4	2
Sun UltraSPARC III	2003	1950	14	4	1
Sun UltraSPARC T1 (Niagara)	2005	1200	6	1	8

# Emissão múltipla

- **Exemplo (Intel Atom 45nm)**
  - **Processador: 1,6 GHz, emitindo até 2 instruções/ciclo**
  - **Pipeline: 16 estágios**
    - »  $1,6 \text{ G} \times 2 = 3,2$  bilhões de instruções por segundo
    - »  $\text{CPI}_{\text{min}} = 0,5$
    - »  $\text{Até } 2 \times 16 = 32$  instruções em execução simultânea

# Emissão múltipla

- **Processadores contemporâneos**
  - Disparam de 2 a 6 instruções por ciclo
    - » Intel Atom: até 2
    - » Intel Core i5/i7: até 4
    - » ARM A15/A57: até 3
    - » Intel Itanium: até 6
  - Mas há restrições no tipo de instruções simultaneamente executáveis.
- **Alternativas de implementação**
  - Estática e dinâmica
  - Divisão de trabalho entre compilador e HW

# Emissão múltipla

- **Emissão múltipla** **estática**
  - **Decisões tomadas estaticamente**
    - » Em tempo de compilação
  - **Abordagem VLIW**
    - » Exemplo: IA-64 (Itanium and Itanium 2)
    - » DSP: NXP Trimedia, AD Sharc, TI C6000
- **Emissão múltipla** **dinâmica**
  - **Decisões tomadas dinamicamente**
    - » Em tempo de execução
  - **Abordagem Superscalar**
    - » Exemplos: PowerPC, Pentium 4, Core (i5/i7), Atom
    - » ARM Cortex: A15 (ARM v7), A57 (ARM v8)

# Pipeline com emissão múltipla

- **Tarefa 1: Empacotamento de instruções**
  - Definição de quantas e quais instruções podem ser disparadas em um dado ciclo de relógio.
    - » Slot de emissão (“issue slot”)
- **Em CPUs com emissão estática**
  - Realizado pelo compilador
    - » Pelo menos parcialmente
- **Em CPUs com emissão dinâmica**
  - Realizado em tempo de execução pelo HW
    - » Favorecido pelo compilador
    - » Pré-ordenamento de instruções

# Pipeline com emissão múltipla

- **Tarefa 2: Manipulação de hazards**
  - Definição de que ações tomar na presença de hazards de dados e de controle.
- **Em CPUs com emissão estática**
  - Realizado pelo compilador
    - » Escalonamento estático de código
    - » Previsão estática de desvios
- **Em CPUs com emissão dinâmica**
  - Realizado em tempo de execução pelo HW
    - » Escalonamento dinâmico de código
    - » Previsão dinâmica de desvios



# **Emissão múltipla estática**

- **CPU usa compilador para auxiliar em**
  - Empacotamento de instruções
  - Manipulação de hazards
- **Instruções emitidas em um dado ciclo**
  - Pacote de emissão
  - Uma instrução longa com múltiplas operações
    - » VLIW: “Very Long Instruction Word”

# **Emissão múltipla estática: exemplo**

- **Usando a ISA do MIPS**
  - **Pacote de emissão: 2 instruções**
    - » Primeira: ALU ou desvio
    - » Segunda: load ou store
- **Consequências**
  - **Buscar 64 bits e decodificar par de instruções**
  - **Se uma instrução não pode ser usada**
    - » Substituída por no-op

# Emissão estática: comportamento

ALU/branch	IF	ID	EX	ME	WB
Load/Store	IF	ID	EX	ME	WB

ALU/branch		IF	ID	EX	ME	WB
Load/Store		IF	ID	EX	ME	WB

ALU/branch			IF	ID	EX	ME	WB
Load/Store			IF	ID	EX	ME	WB

ALU/branch				IF	ID	EX	ME	WB
Load/Store				IF	ID	EX	ME	WB

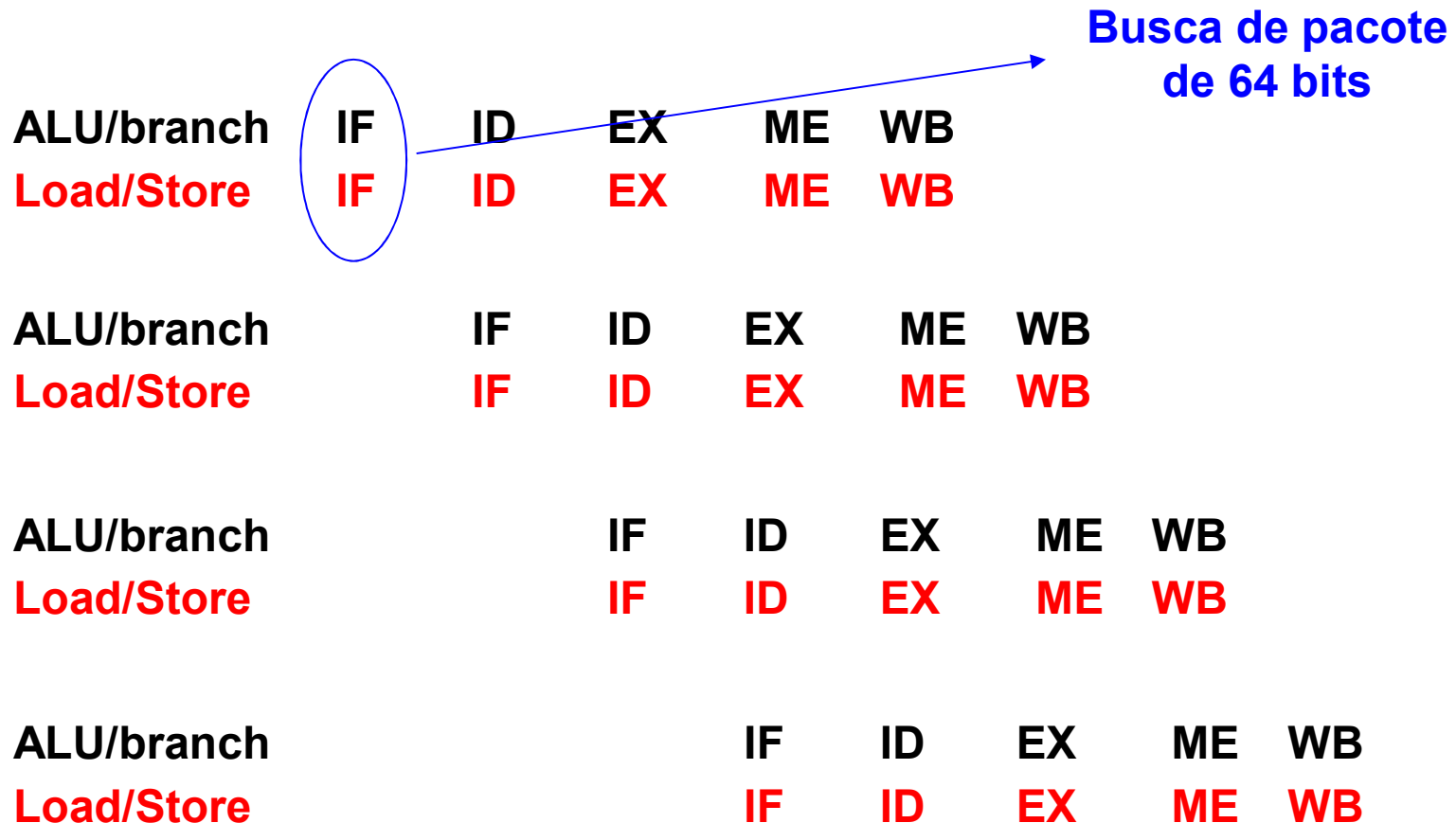
# Emissão estática: abordagem 1

- **Compilador com responsabilidade total**
  - **Remover hazards**
    - » O maior número possível
  - **Inserir no-ops**
    - » Quando hazard não pode ser removido
  - **Escalonar o código**
- **Consequência**
  - **CPU não requer suporte em HW para hazards**
    - » Detecção
    - » Geração de pausas (“stalls”)

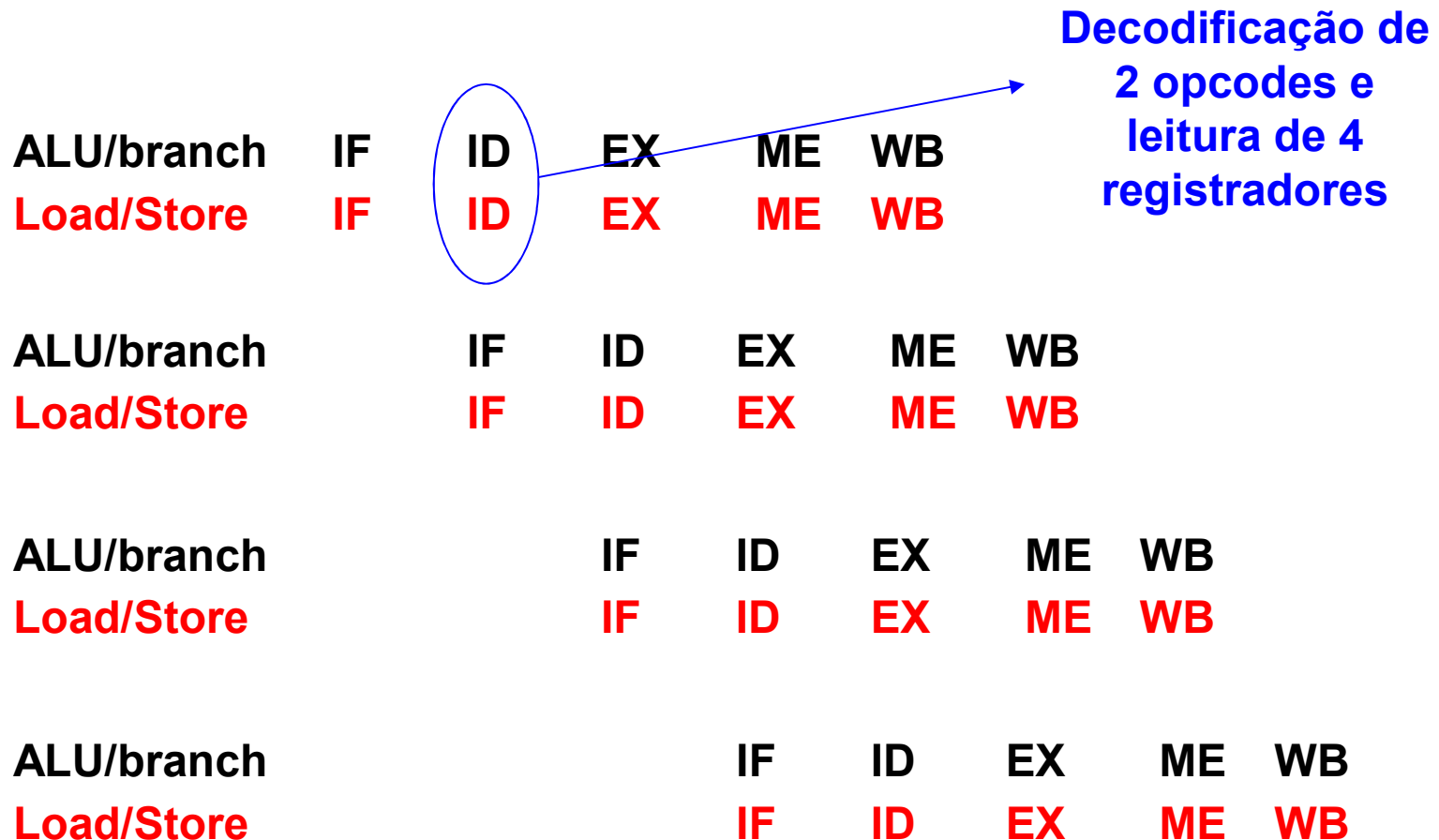
## **Emissão estática: abordagem 2**

- **Compilador só responsável pelo pacote**
  - Garante que par de instruções é independente
  - Favorece a redução de hazards entre pacotes
- **Consequências:**
  - HW detecta hazards entre pacotes
  - HW gera pausas entre pacotes
    - » Em geral todo o pacote contendo a instrução dependente sofre pausa

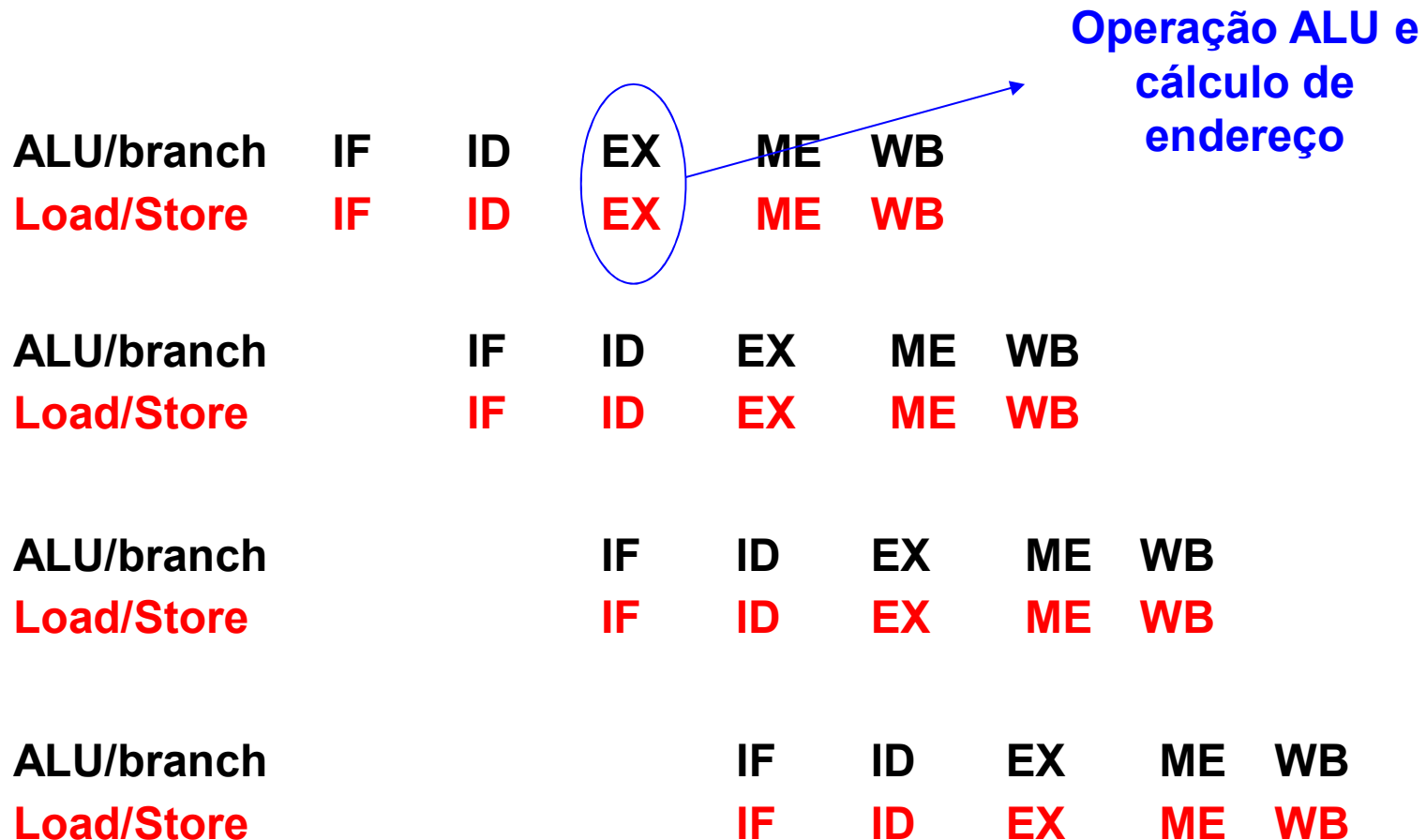
# Emissão estática: comportamento



# Emissão estática: comportamento

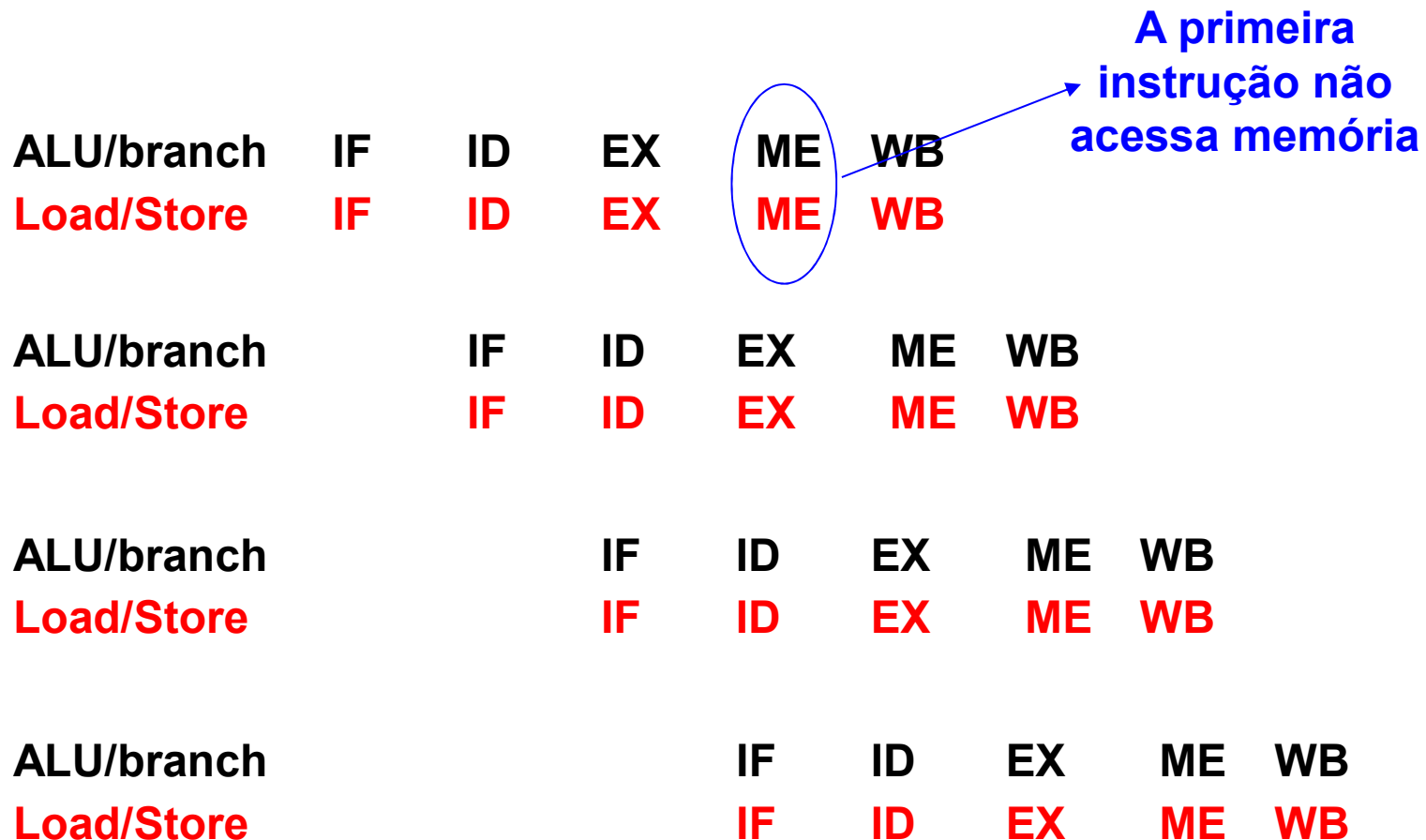


# Emissão estática: comportamento

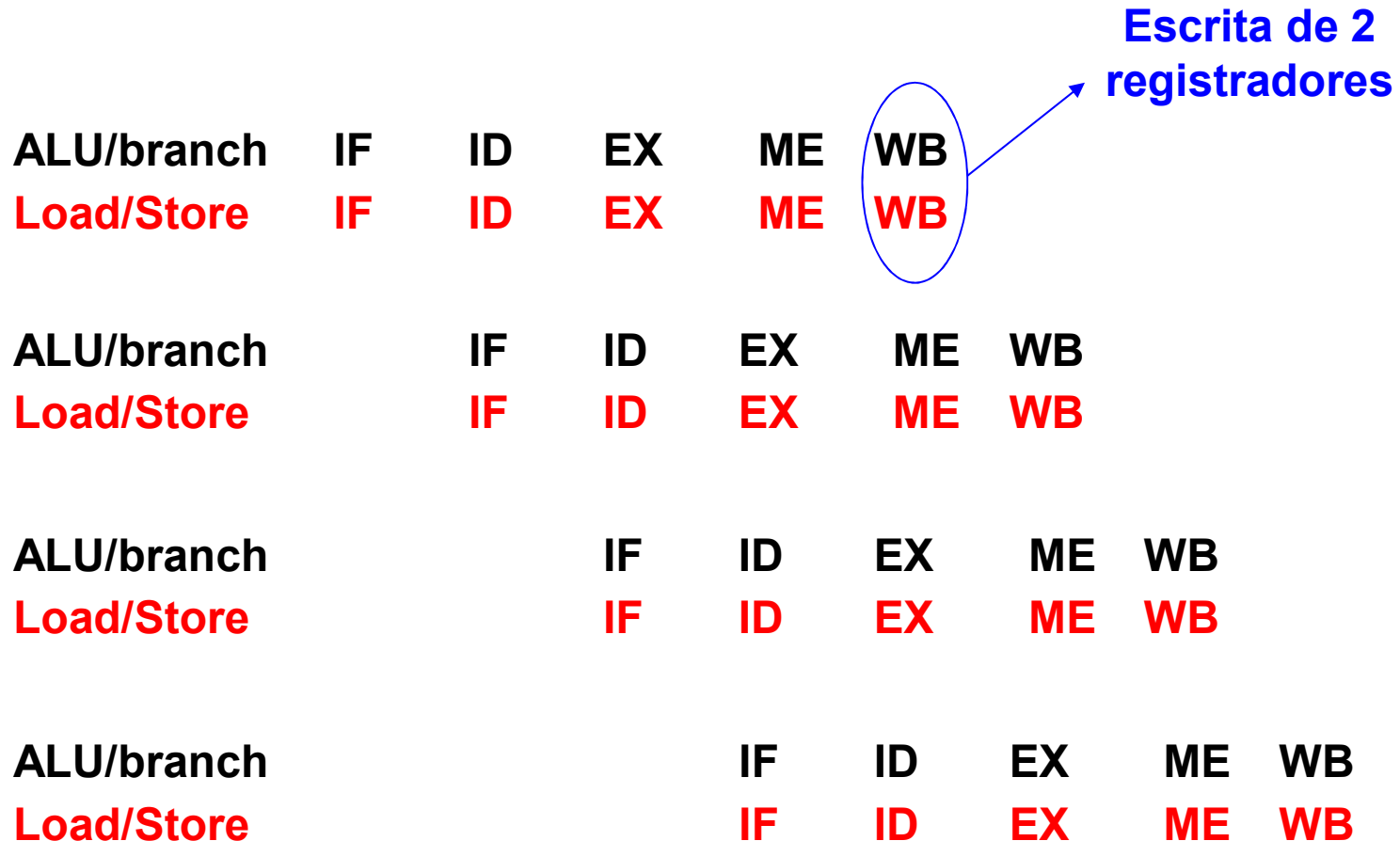




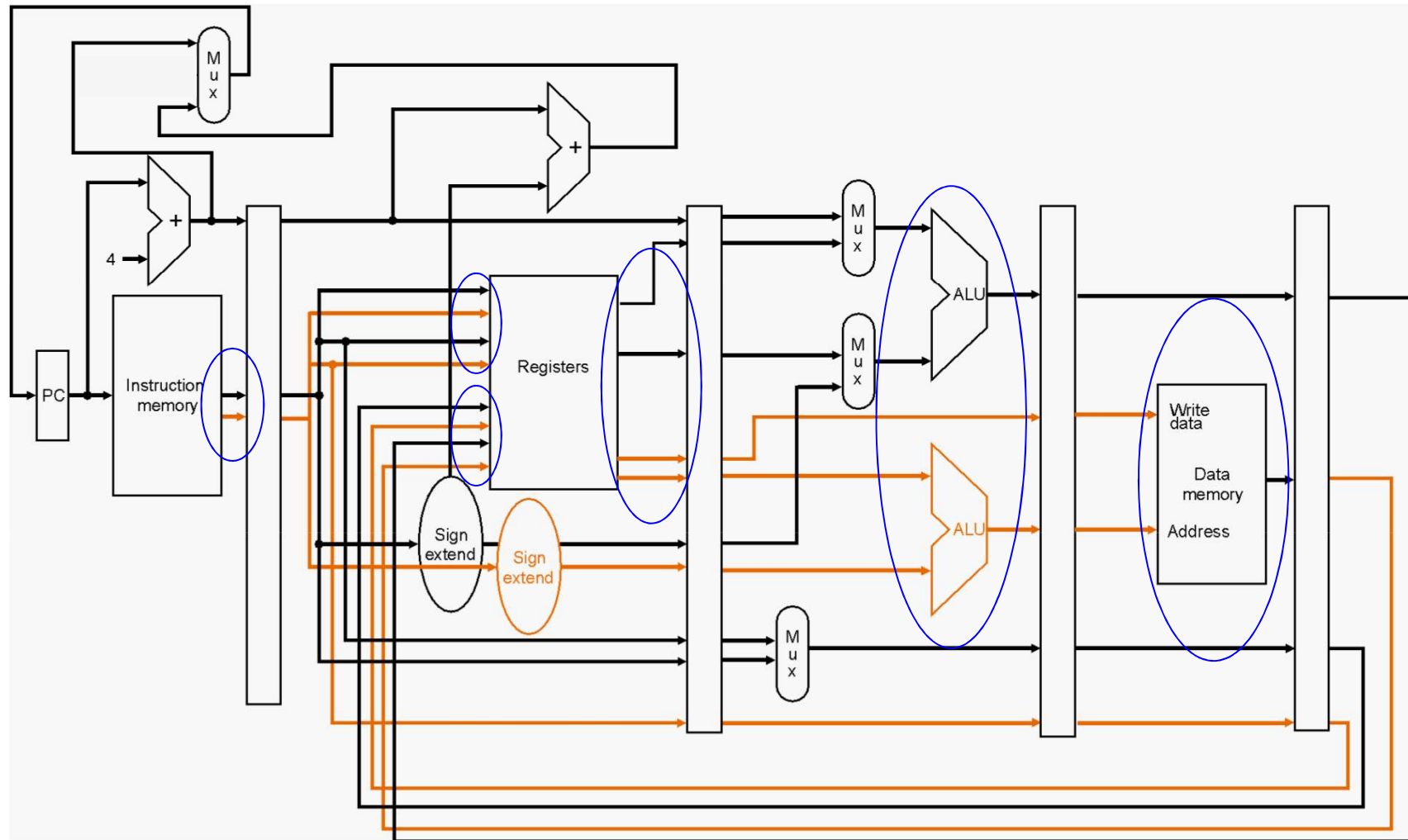
# Emissão estática: comportamento



# Emissão estática: comportamento

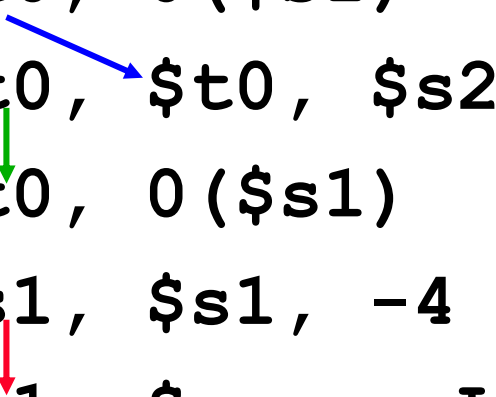


# Emissão estática: estrutura



## Emissão simples: Escalonamento

```
Loop:    lw    $t0, 0($s1)
          addu  $t0, $t0, $s2
          sw    $t0, 0($s1)
          addi  $s1, $s1, -4
          bne   $s1, $zero, Loop
```



(Hipóteses: teste resolvido em ID; todos os atalhos fisicamente realizáveis)

**Onde estão as dependências de dados ?**

**Quais dependências causam hazards ?**

# Onde estão os hazards?

lw \$t0, 0(\$s1)	IF	ID	EX	ME	WB		
------------------	----	----	----	----	----	--	--

# Onde estão os hazards?

lw <b>\$t0</b> , 0(\$s1)	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>		
addu \$t0, <b>\$t0</b> , \$s2		<b>X</b>	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>

# Onde estão os hazards?

lw <b>\$t0</b> , 0(\$s1)	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>		
addu \$t0, <b>\$t0</b> , \$s2		<b>X</b>	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>
addu <b>\$t0</b> , \$t0, \$s2	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>		

# Onde estão os hazards?

lw <b>\$t0</b> , 0(\$s1)	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>		
addu \$t0, <b>\$t0</b> , \$s2		<b>X</b>	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>
addu <b>\$t0</b> , \$t0, \$s2	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>		
sw <b>\$t0</b> , 0(\$s1)		<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>	



# Onde estão os hazards?

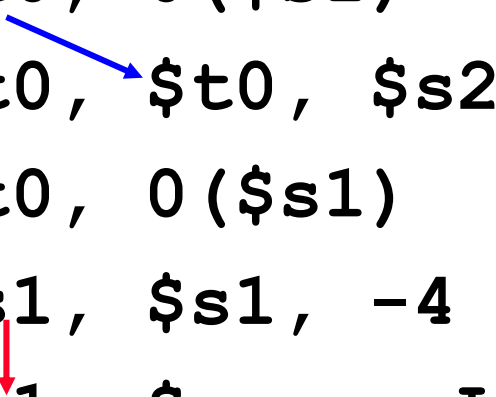
lw <b>\$t0</b> , 0(\$s1)	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>		
addu \$t0, <b>\$t0</b> , \$s2		<b>X</b>	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>
addu <b>\$t0</b> , \$t0, \$s2	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>		
sw <b>\$t0</b> , 0(\$s1)		<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>	
addi <b>\$s1</b> , \$s1, -4	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>		

# Onde estão os hazards?

lw <b>\$t0</b> , 0(\$s1)	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>		
addu \$t0, <b>\$t0</b> , \$s2		<b>X</b>	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>
addu <b>\$t0</b> , \$t0, \$s2	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>		
sw <b>\$t0</b> , 0(\$s1)		<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>	
addi <b>\$s1</b> , \$s1, -4	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>		
bne <b>\$s1</b> , \$zero, Loop		<b>X</b>	<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>

## Emissão simples: Escalonamento

```
Loop:    lw    $t0, 0($s1)
          addu  $t0, $t0, $s2
          sw    $t0, 0($s1)
          addi  $s1, $s1, -4
          bne   $s1, $zero, Loop
```



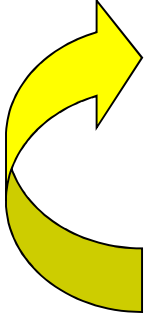
**Emissão simples:  $5 + 2 = 7$  ciclos por iteração**

**$CPI = 7/5 = 1,4$**

**Melhor caso (emissão simples):  $CPI = 1$**

## Emissão simples: Escalonamento

Loop:      lw    \$t0, 0(\$s1)  
             addu \$t0, \$t0, \$s2  
             sw    \$t0, 0(\$s1)  
             addi \$s1, \$s1, -4  
             bne  \$s1, \$zero, Loop



Como eliminar os hazards ?

# Emissão múltipla: o código escalonado

	ALU or branch instruction	Data transfer instruction	Clock Cycle
Loop:		lw \$t0, 0(\$s1)	1
	addi \$s1, \$s1, -4		2
	addu \$t0, \$t0, \$s2		3
	bne \$s1, \$zero, Loop	sw \$t0, 0(\$s1)	4

**Nenhuma dependência causa hazard.**

# Emissão múltipla: o código escalonado

	ALU or branch instruction	Data transfer instruction	Clock Cycle
Loop:		lw \$t0, 0(\$s1)	1
	addi \$s1, \$s1, -4		2
	addu \$t0, \$t0, \$s2		3
	bne \$s1, \$zero, Loop	sw \$t0, 0(\$s1)	4

 no-ops

4

4 ciclos por iteração

4 ciclos para executar 5 instruções → CPI = 0,8

Melhor caso: CPI = 0,5

# Emissão múltipla: “loop unrolling”

```
Loop:  lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -4
      bne     $s1, $zero, Loop
```

[Ex. gcc: **-funroll-loops**  
**-funroll-all-loops**]

**Laço desenrolado 4 vezes**

# Emissão múltipla: “loop unrolling”

```
Loop:  lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -4
      bne     $s1, $zero, Loop
      lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -4
      bne     $s1, $zero, Loop
      lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -4
      bne     $s1, $zero, Loop
      lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -4
      bne     $s1, $zero, Loop
```

Laço desenrolado 4 vezes



# Emissão múltipla: “loop unrolling”

```
Loop:  lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -4
      bne     $s1, $zero, Loop
      lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -4
      bne     $s1, $zero, Loop
      lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -4
      bne     $s1, $zero, Loop
      lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -4
      bne     $s1, $zero, Loop
```

# Emissão múltipla: “loop unrolling”

```
Loop:  lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -4

      lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -4

      lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -4

      lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -4
      bne     $s1, $zero, Loop
```

# Emissão múltipla: “loop unrolling”

```
Loop:  lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)

      lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)

      lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)

      lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)
      addi    $s1, $s1, -16
      bne     $s1, $zero, Loop
```

# Emissão múltipla: “loop unrolling”

```
Loop:  lw    $t0, 0($s1)
      addu  $t0, $t0, $s2
      sw    $t0, 0($s1)
      ↓
      lw    $t0, 0($s1)
      addu  $t0, $t0, $s2
      sw    $t0, 0($s1)
      ↓
      lw    $t0, 0($s1)
      addu  $t0, $t0, $s2
      sw    $t0, 0($s1)
      ↓
      lw    $t0, 0($s1)
      addu  $t0, $t0, $s2
      sw    $t0, 0($s1)
      addi  $s1, $s1, -16
      bne   $s1, $zero, Loop
```

**Anti-dependências**  
**(dependências de nome)**

# Emissão múltipla: “loop unrolling”

```
Loop:  lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)

      lw      $t1, 0($s1)
      addu    $t1, $t1, $s2
      sw      $t1, 0($s1)

      lw      $t2, 0($s1)
      addu    $t2, $t2, $s2
      sw      $t2, 0($s1)

      lw      $t3, 0($s1)
      addu    $t3, $t3, $s2
      sw      $t3, 0($s1)
      addi    $s1, $s1, -16
      bne     $s1, $zero, Loop
```

Renomeação de registradores  
 (“register renaming”)


# Emissão múltipla: “loop unrolling”

```
Loop:  lw      $t0, 0($s1)
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)

      lw      $t1, 0($s1)
      addu    $t1, $t1, $s2
      sw      $t1, 0($s1)

      lw      $t2, 0($s1)
      addu    $t2, $t2, $s2
      sw      $t2, 0($s1)

      lw      $t3, 0($s1)
      addu    $t3, $t3, $s2
      sw      $t3, 0($s1)
      addi    $s1, $s1, -16
      bne     $s1, $zero, Loop
```



# Emissão múltipla: “loop unrolling”

```
Loop:  lw      $t0, 0($s1)
      addi    $s1, $s1, -16
      addu    $t0, $t0, $s2
      sw      $t0, 0($s1)

      lw      $t1, 0($s1)
      addu    $t1, $t1, $s2
      sw      $t1, 0($s1)

      lw      $t2, 0($s1)
      addu    $t2, $t2, $s2
      sw      $t2, 0($s1)

      lw      $t3, 0($s1)
      addu    $t3, $t3, $s2
      sw      $t3, 0($s1)

      bne     $s1, $zero, Loop
```

Compensação dos  
deslocamentos

(“offset compensation”)

# Emissão múltipla: “loop unrolling”

```
Loop:  lw      $t0, 0($s1)
      addi    $s1, $s1, -16
      addu    $t0, $t0, $s2
      sw      $t0, 16($s1)

      lw      $t1, 12($s1)
      addu    $t1, $t1, $s2
      sw      $t1, 12($s1)

      lw      $t2, 8($s1)
      addu    $t2, $t2, $s2
      sw      $t2, 8($s1)

      lw      $t3, 4($s1)
      addu    $t3, $t3, $s2
      sw      $t3, 4($s1)

      bne     $s1, $zero, Loop
```

Compensação dos  
deslocamentos

(“offset compensation”)



# Emissão múltipla: o código escalonado

	ALU or branch instruction	Data transfer instruction	Clock Cycle
Loop:	addi \$s1, \$s1, -16	lw \$t0, 0(\$s1)	1
		lw \$t1, 12(\$s1)	2
	addu \$t0, \$t0, \$s2	lw \$t2, 8(\$s1)	3
	addu \$t1, \$t1, \$s2	lw \$t3, 4(\$s1)	4
	addu \$t2, \$t2, \$s2	sw \$t0, 16(\$s1)	5
	addu \$t3, \$t3, \$s2	sw \$t1, 12(\$s1)	6
		sw \$t2, 8(\$s1)	7
	bne \$s1, \$zero, Loop	sw \$t3, 4(\$s1)	8

Nenhuma dependência causa hazard.

# Emissão múltipla: o código escalonado

	ALU or branch instruction	Data transfer instruction	Clock Cycle
Loop:	<code>addi \$s1, \$s1, -16</code>	<code>lw \$t0, 0(\$s1)</code>	1
		<code>lw \$t1, 12(\$s1)</code>	2
	<code>addu \$t0, \$t0, \$s2</code>	<code>lw \$t2, 8(\$s1)</code>	3
	<code>addu \$t1, \$t1, \$s2</code>	<code>lw \$t3, 4(\$s1)</code>	4
	<code>addu \$t2, \$t2, \$s2</code>	<code>sw \$t0, 16(\$s1)</code>	5
	<code>addu \$t3, \$t3, \$s2</code>	<code>sw \$t1, 12(\$s1)</code>	6
		<code>sw \$t2, 8(\$s1)</code>	7
	<code>bne \$s1, \$zero, Loop</code>	<code>sw \$t3, 4(\$s1)</code>	8

 no-ops

8 ciclos para 4 iterações = 2 ciclos por iteração

8 ciclos para executar 14 instruções → CPI = 0,57

**Melhor caso: CPI = 0,5**

## **Conclusão: emissão simples**

- **Impacto do escalonamento**
  - Código original não-escalonado:  $CPI = 1,4$
  - Código escalonado:  $CPI = 1$
- **Fontes de melhoria**
  - Eliminação de hazards
- **Custo da melhoria**
  - Nenhum custo adicional

# Conclusão: emissão múltipla (dupla)

- **Impacto do escalonamento**
  - Código original escalonado: CPI = 0,8
  - Código desenrolado 4x e escalonado: CPI = 0,57
- **Fontes de melhoria**
  - Eliminação de hazards
  - Redução das instruções de controle do laço
  - Exploração de ILP
    - » Expor paralelismo no programa
    - » Para acomodá-lo no HW com emissão múltipla
- **Custo da melhoria**
  - Uso de 4x mais registradores
  - Aumento do tamanho do código ( $14/5 \approx 3$  vezes)