



Universidade Federal de Pelotas
Centro de Desenvolvimento Tecnológico
Bacharelado em Ciência da Computação
Engenharia de Computação

Arquitetura e Organização de Computadores I

Prática

Aula 1

Apresentação, Lógica Bit-a-Bit, Shift Lógico

Prof. Guilherme Corrêa
gcorrea@inf.ufpel.edu.br

Prof. Bruno Zatt

Apresentação

► Bibliografia e Software

- PATTERSON, David A.; HENESSY, John L.
Organização e Projeto de Computadores: A Interface Hardware/Software, 3ª ed. Rio de Janeiro: Elsevier, 2005.
- **MARS 4.5** (*MIPS Assembler and Runtime Simulator*)
<http://courses.missouristate.edu/kenvollmar/mars/>

► Atendimento ao Aluno

- Combinar horário por e-mail:
gcorrea@inf.ufpel.edu.br
- Lab 7 (Sistemas Digitais e Embarcados / GACI)

Apresentação

► Avaliação

- Nota da prática vale **40%** na nota final de AOC-1
- Prova Prática 1 (**10%**)
 - **02/10**
- Prova Prática 2 (**10%**)
 - **27/11**
- Trabalhos Práticos (**10%**)
 - **10 trabalhos para entrega em cada aula**
- Trabalho Prático Final (**10%**)
 - **de 13/11 a 04/12**

Apresentação

► Entrega dos Trabalhos Práticos

- **DEZ** trabalhos práticos ao longo do semestre, com peso de **10%** na nota final da disciplina;
- Enviar pelo **AVA** até a data/hora-limite informada nas aulas;
- Não serão aceitos envios após a data/hora-limite.

Introdução

► Assembly

- Linguagem da máquina
 - Instruções
 - Conjunto de instruções
- Diversas, mas parecidas entre si
- Simplicidade
- Por que aprender Assembly?
 - Compreender a **organização** dos computadores
 - Compreender o **funcionamento** dos processadores
 - Compreender como as linguagens são **processadas** em baixo nível

Introdução

► Assembly

- Exemplo:

C/C++

```
a = b & c;
```

**Assembly
(simbólico)**

```
load b  
load c  
and a, b, c  
store a
```

Assembly

```
10100010 00101011  
10000100 00100010
```

```
10100011 00101011  
10011000 00100110
```

```
10110010 00101011  
10100100 00100011
```

```
10100001 00101011  
10000000 00100000
```

Introdução

► Assembly

- **Vantagens**
 - Desempenho
 - Sistemas embarcados
 - Tempo de execução previsível
 - Sistemas com tempo crítico
- **Desvantagens**
 - Programas para uma máquina específica
 - Programas longos
 - Pouca legibilidade (ou seja, comentem os seus códigos!!!)

Introdução

► MIPS

- Protótipo da Stanford University (California), 1983
- John Hennessy
- Arquitetura de processador **RISC** (Reduced Instruction Set Computer)
- Originou o MIPS comercial
- Utilizado em **diversas aplicações** atualmente
 - Computadores da Silicon Graphics
 - Roteadores
 - Videogames (Nintendo 64, PlayStation)
- Veremos um **sub-conjunto** de instruções na disciplina (Prática)
- Paralelamente, veremos a organização do MIPS (Teórica)

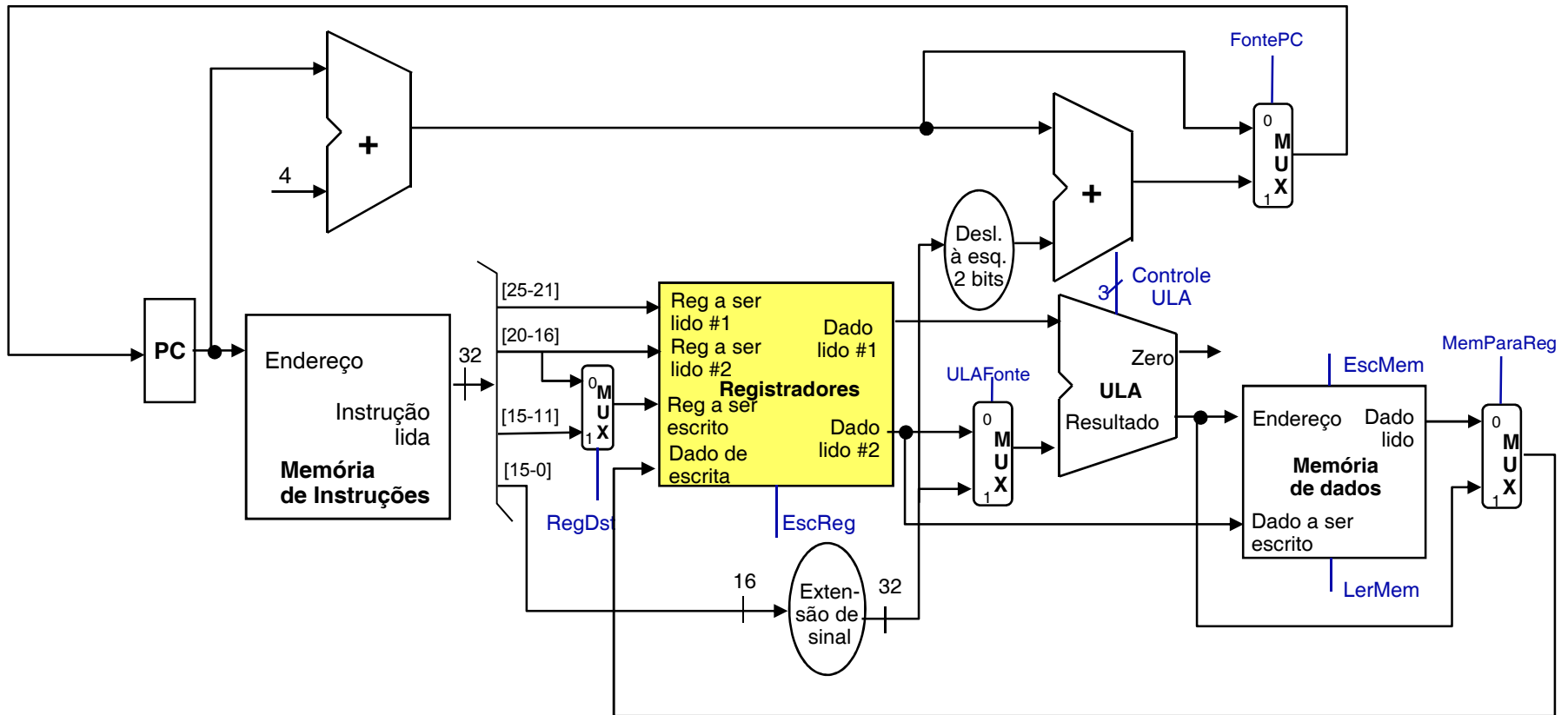
Introdução

► MIPS: estrutura básica

- Banco de registradores (32 registradores)
- Program Counter (PC)
- Memória (dados, instruções)
- Unidade Lógica e Aritmética (ULA)

Introdução

► MIPS (monociclo)



Introdução

► MIPS: Registradores

- Elemento mais alto na hierarquia de memória
- Única memória que o processador **acessa diretamente**
- **32 registradores** de propósito geral de **32 bits**
 - \$0, \$1, ..., \$31
 - operações inteiras
 - endereçamento
- **\$0** tem sempre **valor 0**
- **\$31** é utilizado para **retorno de funções**

Introdução

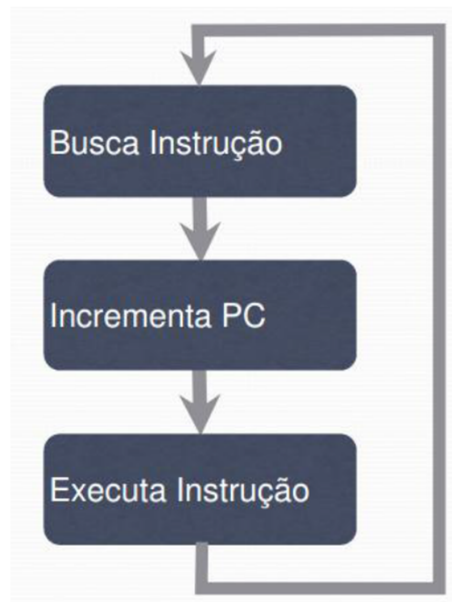
► MIPS: Registradores

Registrador	Nome	Uso (convenção)
\$0	\$zero	Zero
\$1	\$at	<i>Assembler Temporary</i>
\$2, \$3	\$v0, \$v1	Valor de retorno de subrotina
\$4 – \$7	\$a0 – \$a3	Argumentos de subrotina
\$8 – \$15	\$t0 – \$t7	Temporários (locais à função)
\$16 – \$23	\$s0 – \$s7	Salvos (não alterados na função)
\$24, \$25	\$t8, \$t9	Temporários
\$26, \$27	\$k0, \$k1	Kernel (reservado para SO)
\$28	\$gp	<i>Global Pointer</i>
\$29	\$sp	<i>Stack Pointer</i>
\$30	\$fp	<i>Frame Pointer</i>
\$31	\$ra	Endereço de Retorno

Introdução

► MIPS: Program Counter (PC)

- Aponta para a posição da memória que contém a próxima instrução a ser executada
- No fluxo normal de execução, $PC \leftarrow PC+4$

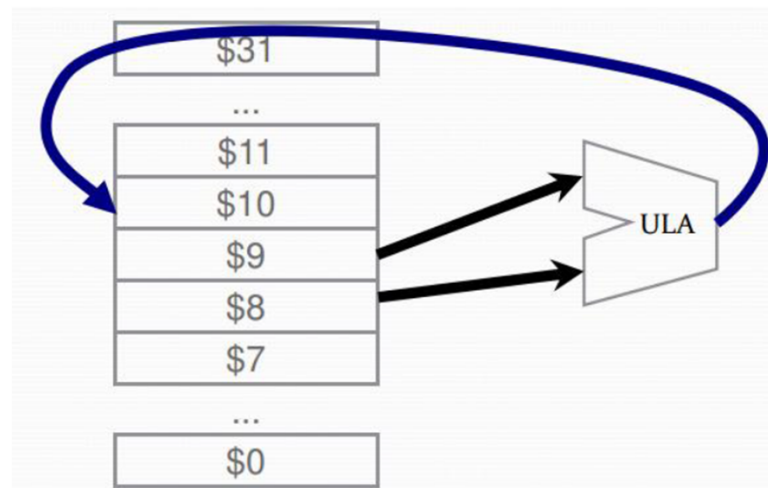


Introdução

► MIPS: Unidade Lógica e Aritmética (ULA)

- Circuito responsável pelas operações lógicas e aritméticas
- Exemplo:

and \$10, \$8, \$9



Introdução

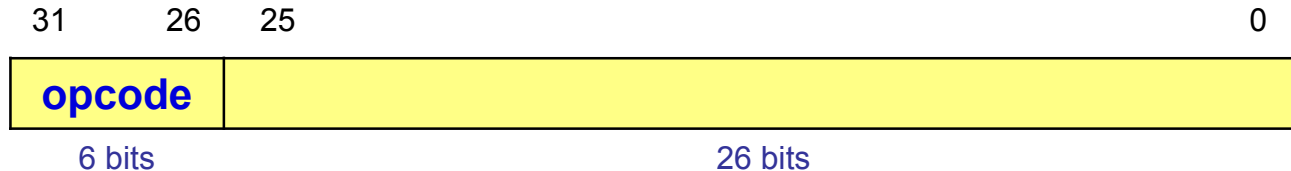
► MIPS: Memória

- Memória de Instruções
- Memória de Dados
- Acessos à memória devem ser alinhados
 - Cada endereço aponta para um byte
 - Dados de 32 bits precisam iniciar em endereços múltiplos de 4
 - Outros tamanhos de dados também são suportados (16 bits, bytes)

Instruções

► MIPS: Instruções

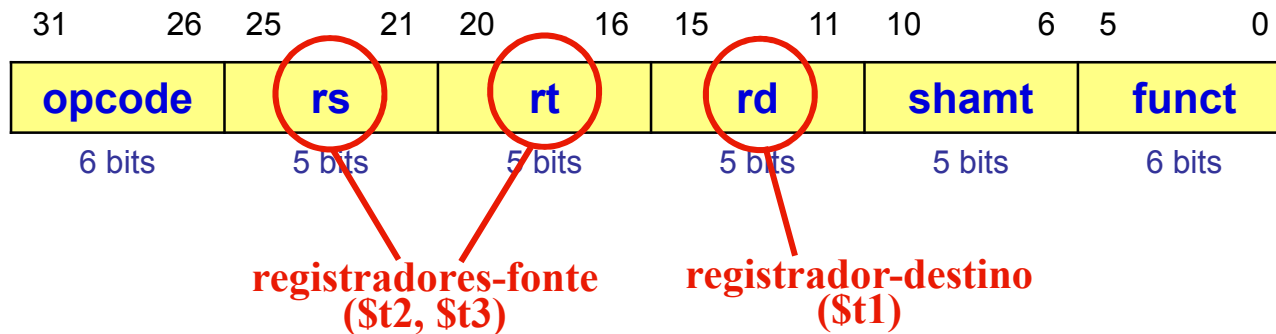
- Todas as instruções têm 32 bits
- Todas têm opcode de 6 bits



Instruções

► Formato R: and, or, xor, nor, etc.

- Opcode = 0
- “funct” define a operação a ser feita pela ALU
- “shamt” (shift amount) é usado em instruções de deslocamento

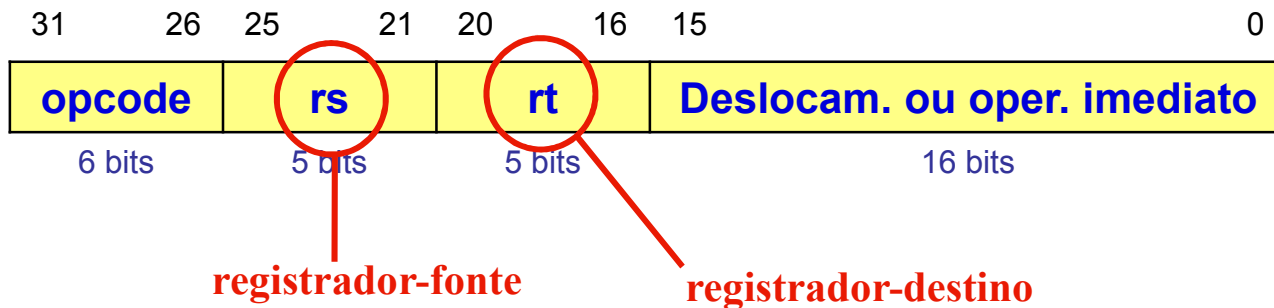


Simbólico (exemplo): or \$t1, \$t2, \$t3 ($\$t1 \leftarrow \$t2 \text{ OR } \$t3$)

Instruções

► Formato I: andi, ori, xori, etc.

- Opcode varia com a operação



Simbólico (exemplo): `ori $t1, $t2, 0x4` ($\$t1 \leftarrow \$t2 \text{ OR } 0x4$)

Instruções Lógicas Bit-a-Bit

► Instrução or

or \$t1, \$t2, \$t3

or \$9, \$10, \$11

- Coloca no registrador \$t1 o valor do **OR bit-a-bit** entre \$t2 e \$t3

\$t2	...	1	1	0
\$t3	...	1	0	0
<hr/>				
\$t1	...	1	1	0

► Instrução and

and \$t1, \$t2, \$t3

and \$9, \$10, \$11

- Coloca no registrador \$t1 o valor do **AND bit-a-bit** entre \$t2 e \$t3

\$t2	...	1	1	0
\$t3	...	1	0	0
<hr/>				
\$t1	...	1	0	0

Instruções Lógicas Bit-a-Bit

► Instrução xor

xor \$t1, \$t2, \$t3

xor \$9, \$10, \$11

- Coloca no registrador \$t1 o valor do **XOR bit-a-bit** entre \$t2 e \$t3

\$t2	...	1	1	0
\$t3	...	1	0	0
<hr/>				
\$t1	...	0	1	0

► Instrução nor

nor \$t1, \$t2, \$t3

nor \$9, \$10, \$11

- Coloca no registrador \$t1 o valor do **NOR bit-a-bit** entre \$t2 e \$t3

\$t2	...	1	1	0
\$t3	...	1	0	0
<hr/>				
\$t1	...	0	0	1

Instruções Lógicas Bit-a-Bit

► Instrução ori

ori \$t1, \$t2, 0x15

ori \$9, \$10, 0x15

- Coloca no registrador \$t1 o valor do **OR bit-a-bit** entre \$t2 e 0x15

► Instrução andi

andi \$t1, \$t2, 0x15

andi \$9, \$10, 0x15

- Coloca no registrador \$t1 o valor do **AND bit-a-bit** entre \$t2 e 0x15

Instruções Lógicas Bit-a-Bit

► Instrução xori

xori \$t1, \$t2, 0x15

xori \$9, \$10, 0x15

- Coloca no registrador \$t1 o valor do **XOR bit-a-bit** entre \$t2 e 0x15

Instruções Lógicas Bit-a-Bit

► Utilizações Especiais

or \$t1, \$t2, \$zero

- Copia o valor em \$t2 para \$t1

\$t2	...	0	1	1
\$zero	...	0	0	0
<hr/>				
\$t1	...	0	1	1

ori \$t1, \$zero, 0x6

- Coloca no registrador \$t1 o valor 0x6

0x6	...	1	1	0
\$zero	...	0	0	0
<hr/>				
\$t1	...	1	1	0

Instruções Lógicas Bit-a-Bit

► Utilizações Especiais

nor \$t1, \$t2, \$zero

- Cópia para \$t1 o valor de \$t2 negado
isto é, $\$t1 \leftarrow \text{NOT } \$t2$

\$t2	...	0	1	1
\$zero	...	0	0	0
<hr/>				
\$t1	...	1	0	0

Instruções de Shift Lógico

► Instrução sll

sll \$t1, \$t2, 2

sll \$9, \$10, 2

- sll = **shift left logical**
- Coloca no registrador \$t1 o resultado do deslocamento lógico de \$t2 duas casas à **esquerda** (considerando o exemplo acima!)
- Preenche os espaços criados à direita com zeros

sll \$t1, \$t2, 2

\$t2

0	0	0	0	0	0	...	0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	-----	---	---	---	---	---	---	---	---	---

\$t1

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Instruções de Shift Lógico

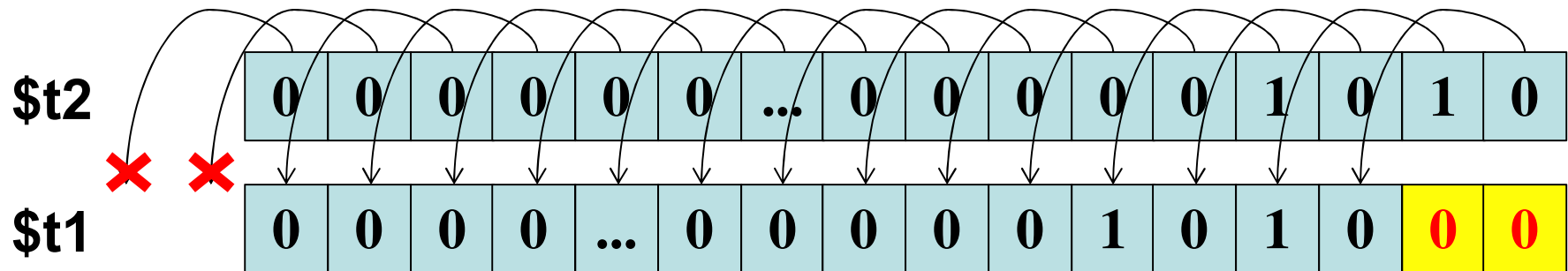
► Instrução sll

sll \$t1, \$t2, 2

sll \$9, \$10, 2

- sll = **shift left logical**
- Coloca no registrador \$t1 o resultado do deslocamento lógico de \$t2 duas casas à **esquerda** (considerando o exemplo acima!)
- Preenche os espaços criados à direita com zeros

sll \$t1, \$t2, 2



Instruções de Shift Lógico

► Instrução sll

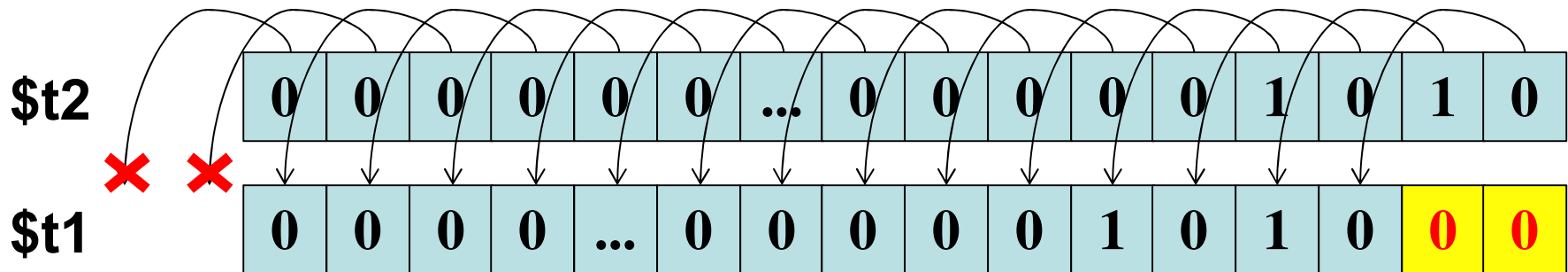
sll \$t1, \$t2, 2

sll \$9, \$10, 2

- sll = **shift left logical**
- Coloca no registrador \$t1 o resultado do deslocamento lógico de \$t2 duas casas à **esquerda** (considerando o exemplo acima!)
- Preenche os espaços criados à direita com zeros

Atenção!
Deslocamento
máximo de 31
posições!

sll \$t1, \$t2, 2



Instruções de Shift Lógico

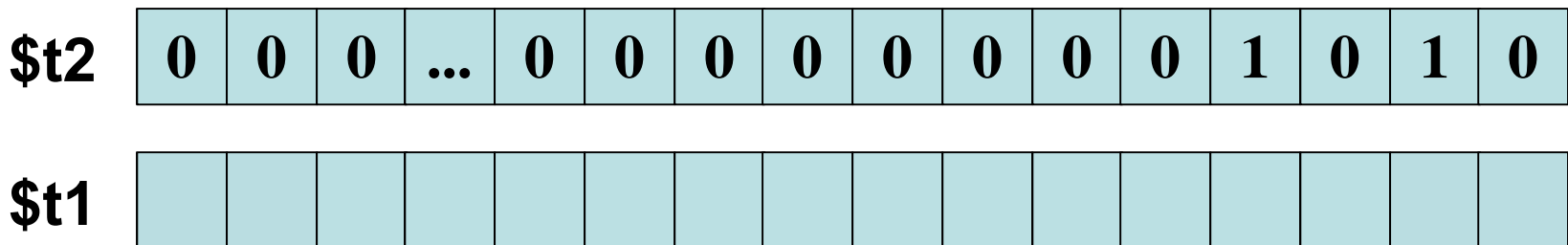
► Instrução srl

srl \$t1, \$t2, 2

srl \$9, \$10, 2

- srl = **shift right logical**
- Coloca no registrador \$t1 o resultado do deslocamento lógico de \$t2 duas casas à **direita** (considerando o exemplo acima!)
- Preenche os espaços criados à esquerda com zeros

srl \$t1, \$t2, 2



Instruções de Shift Lógico

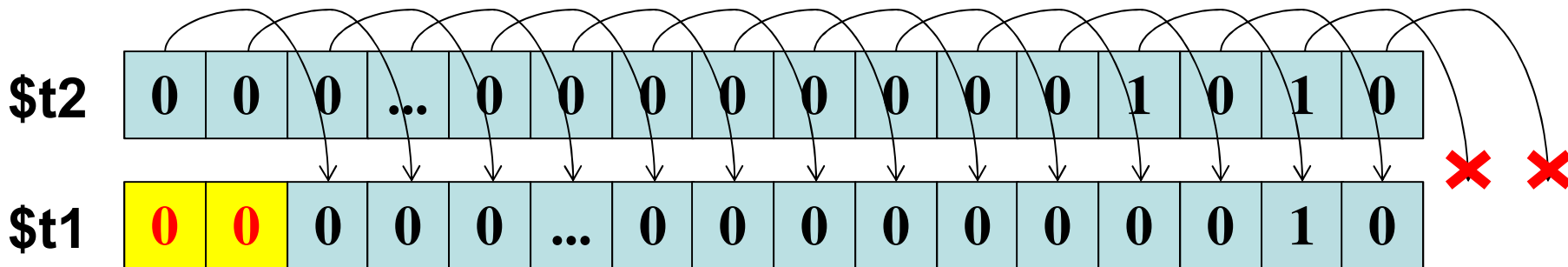
► Instrução srl

srl \$t1, \$t2, 2

srl \$9, \$10, 2

- srl = **shift right logical**
- Coloca no registrador \$t1 o resultado do deslocamento lógico de \$t2 duas casas à **direita** (considerando o exemplo acima!)
- Preenche os espaços criados à esquerda com zeros

srl \$t1, \$t2, 2



Instruções de Shift Lógico

► Instrução srl

srl \$t1, \$t2, 2

srl \$9, \$10, 2

- srl = **shift right logical**
- Coloca no registrador \$t1 o resultado do deslocamento lógico de \$t2 duas casas à **direita** (considerando o exemplo acima!)
- Preenche os espaços criados à esquerda com zeros

Atenção!
Deslocamento
máximo de 31
posições!

srl \$t1, \$t2, 2

