

MICROPROCESSADORES II

(EMA864315)

NIOS II - ASSEMBLY

1º SEMESTRE / 2019

Alexandro Baldassin


MATERIAL DIDÁTICO

◆ Harris & Harris

- 6.1 – Introduction
- 6.2 – Assembly Language
- 6.3 – Machine Language
- 6.4 – Programming (exceto 6.4.6)
- 6.6 – Lights, Camera, Action
- 6.7.1 e 6.7.3

◆ Patterson & Hennessy (4a edição)

- 2.1 – Introduction
- 2.2 – Operations of the Computer Hardware
- 2.3 – Operands of the Computer Hardware
- 2.5 – Representing Instructions in the Computer
- 2.7 – Instructions for Making Decisions
- 2.10 – MIPS addressing for 32-bit immediates and addresses
- 2.12 – Translating and Starting a Program



Notem que os livros
usam o processador
MIPS

INSTRUCTION SET ARCHITECTURE (ISA)

◆ ISA

- Parte do processador visível ao programador/compilador

◆ Projeto do ISA

- número de registradores e tipos de dados
- repertório de instruções
- modos de endereçamento
- formato das instruções

◆ Compromisso entre hardware simples e facilidade de programação

FILOSOFIA DE PROJETO

- ◆ **CISC (*Complex Instruction Set Computing*)**
 - mover complexidade do software para hardware

- ◆ **RISC (*Reduced Instruction Set Computing*)**
 - mover complexidade do hardware para software

FILOSOFIA DE PROJETO

◆ **CISC** (*Complex Instruction Set Computing*)

- mover complexidade do software para hardware
- menor tamanho de código, mas ciclos por instrução maior

◆ **RISC** (*Reduced Instruction Set Computing*)

- mover complexidade do hardware para software
- maior tamanho de código, mas ciclos por instrução menor

FILOSOFIA DE PROJETO

◆ **CISC (*Complex Instruction Set Computing*)**

- mover complexidade do software para hardware
- menor tamanho de código, mas ciclos por instrução maior
- operações realizadas diretamente em memória

◆ **RISC (*Reduced Instruction Set Computing*)**

- mover complexidade do hardware para software
- maior tamanho de código, mas ciclos por instrução menor
- arquitetura load-store

PROCESSADOR NIOS II

- ◆ **O Nios II é um processador RISC de 32 bits**
 - Similar ao MIPS
- ◆ **Consequências de ser RISC**
 - arquitetura load-store
 - memória só é acessada através de instruções load e store
 - banco de registradores homogêneo
 - 32 registradores de 32 bits
- ◆ **Nios II é um tipo de *soft processor***
 - termo usado para designar um processador que é implementado usando primitivas lógicas de uma FPGA

SOFT PROCESSOR

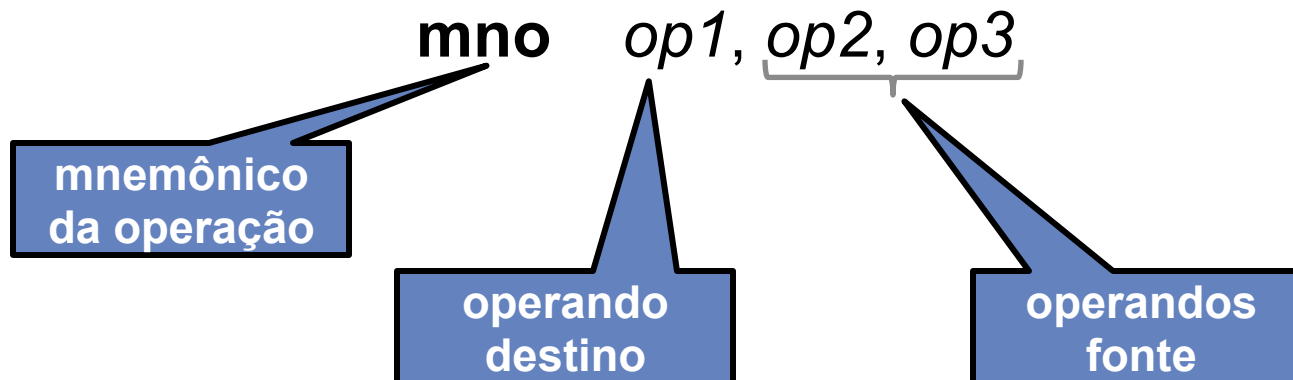
- ◆ **Pelo fato de ser um modelo HDL, o Nios II pode ser configurado de formas diferentes antes de ser sintetizado**
- ◆ **A Altera (fabricante) fornece três configurações básicas**
 - Nios II/f – “fast version”
 - Nios II/s – “standard version”
 - Nios II/e – “economy version”
- ◆ **A configuração básica que começaremos usando utiliza a versão econômica do processador**

NIOS II – LINGUAGEM DE MONTAGEM

◆ Linguagem de Montagem

- versão textual do ISA (pode variar de montador para montador)

◆ Sintaxe para instruções com 3 operandos:



NIOS II – OPERANDOS

- ◆ **Operandos são divididos em 3 tipos principais**
- ◆ **Registrador**
- ◆ **Posição de memória**
- ◆ **Constante (imediato)**

NIOS II – REGISTRADORES

◆ Registrador

- espaço de armazenamento interno extremamente rápido

◆ Nios II

- operações só podem ser executadas sobre registradores
- instruções específicas para transferir dados entre registradores e memória (*load-store instructions*)
- 32 registradores de 32 bits cada
- um dos registradores é somente de leitura (zero)
- uma palavra (*word*) tem 32 bits

NIOS II – REGISTRADORES

- ◆ Registradores podem ser referenciados pelo número ou nome

Register	Name	Function
r0	zero	0x00000000
r1	at	Assembler Temporary
r2		
r3		
.	.	.
.	.	.
.	.	.
r23		
r24	et	Exception Temporary (1)
r25	bt	Breakpoint Temporary (2)
r26	gp	Global Pointer
r27	sp	Stack Pointer
r28	fp	Frame Pointer
r29	ea	Exception Return Address (1)
r30	ba	Breakpoint Return Address (2)
r31	ra	Return Address

(1) The register is not available in User mode
(2) The register is used exclusively by the JTAG Debug module

NIOS II – INSTRUÇÕES ARITMÉTICAS

◆ Adição

add r1, r2, r3

a = b + c

alto nível

NIOS II – INSTRUÇÕES ARITMÉTICAS

◆ Adição

add r1, r2, r3

a = b + c

alto nível

◆ Subtração

sub r1, r2, r3

a = b - c

NIOS II – IMEDIATOS

- ◆ Um dos operandos pode ser especificado como imediato
- ◆ Imediatos são codificados na própria instrução
- ◆ Exemplo: instrução de soma com imediato

addi r1, r2, 10

a = b + 10

complemento
de 2 (16 bits)

NIOS II – IMEDIATOS

- ◆ **Por que não há instrução de subtração com imediato?**

$$a = b - 10$$

NIOS II – IMEDIATOS

- ◆ Por que não há instrução de subtração com imediato?

a = b - 10

addi r1, r2, -10

NIOS II – IMEDIATOS

- ◆ Por que não há instrução de subtração com imediato?

a = b - 10

addi r1, r2, -10

- ◆ Como fazer atribuição?

a = b

NIOS II – IMEDIATOS

- ◆ Por que não há instrução de subtração com imediato?

a = b - 10

addi r1, r2, -10

- ◆ Como fazer atribuição?

a = b

add r1, r2, zero

IMEDIATOS DE 32 BITS

- ◆ **Imediatos representam valores de 16 bits**
 - [-32768, +32767]
- ◆ **Como tratar valores de 32 bits?**

`a = 0xABABCD`

IMEDIATOS DE 32 BITS

◆ Imediatos representam valores de 16 bits

- [-32768, +32767]

◆ Como tratar valores de 32 bits?

$a = 0xABABCD$

◆ Instrução **orhi**

- operação bit a bit (bitwise) OR com a parte alta do imediato
- altera somente os 16 bits mais significativos do registrador
- 16 bits menos significativos ficam zerados

IMEDIATOS DE 32 BITS

a = 0xABABCDCD

```
orhi r1, zero, 0xABAB  
ori  r1, r1, 0xCDCD
```

IMEDIATOS DE 32 BITS

a = 0xABABCDCD

```
orhi r1, zero, 0xABAB  
ori  r1, r1, 0xCDCD
```

◆ Equivalente ao seguinte código?

```
orhi r1, zero, 0xABAB  
addi r1, r1, 0xCDCD
```


IMEDIATOS DE 32 BITS

a = 0xABABCDCD

```
orhi r1, zero, 0xABAB  
ori  r1, r1, 0xCDCD
```

◆ Equivalente ao seguinte código?

```
orhi r1, zero, 0xABAB  
addi r1, r1, 0xCDCD
```

NÃO!!! Por quê?

OUTROS TIPOS DE INSTRUÇÕES

- ◆ **Durante a disciplina veremos vários outros tipos de instruções**
- ◆ **Instruções lógicas**
 - and, or, xor
- ◆ **Instruções de comparação**
 - cmplt, cmpeq, cmpne
- ◆ **Instruções de deslocamento**
 - srl, sll

ALÉM DE REGISTRADORES

- ◆ E se um programa necessitar de mais de 31 variáveis vivas ao mesmo tempo?

ALÉM DE REGISTRADORES

- ◆ **E se um programa necessitar de mais de 31 variáveis vivas ao mesmo tempo?**
 - usar espaço em memória

ALÉM DE REGISTRADORES

- ◆ **E se um programa necessitar de mais de 31 variáveis vivas ao mesmo tempo?**
 - usar espaço em memória

- ◆ **Registradores x Memória**
 - registradores são rápidos, mas poucos
 - memória é grande, mas lenta

ALÉM DE REGISTRADORES

- ◆ **E se um programa necessitar de mais de 31 variáveis vivas ao mesmo tempo?**
 - usar espaço em memória
- ◆ **Registradores x Memória**
 - registradores são rápidos, mas poucos
 - memória é grande, mas lenta
- ◆ **Desafio**
 - quais variáveis preservar em registradores, e quais em memória (*spilling*)?

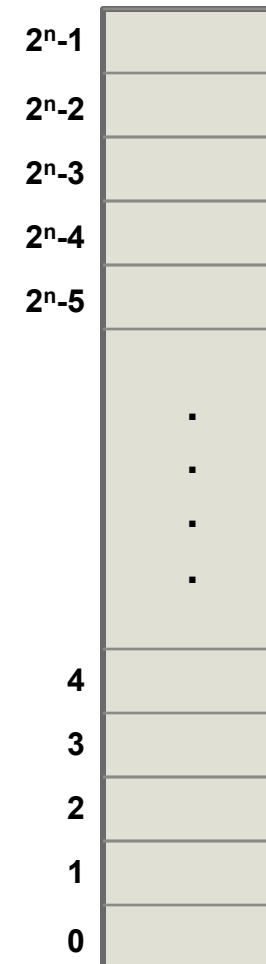
ESTRUTURA DA MEMÓRIA

- ◆ **Memória vista como um vetor unidimensional**



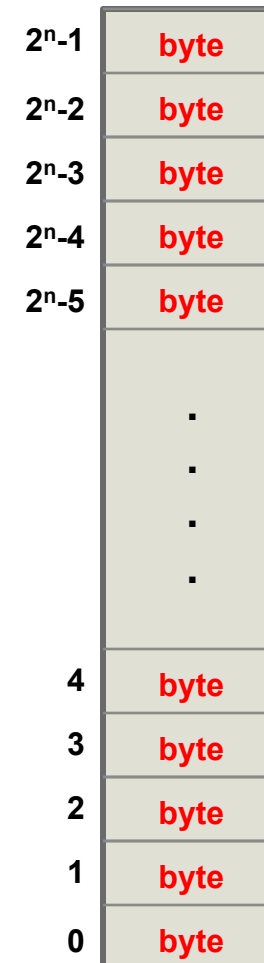
ESTRUTURA DA MEMÓRIA

- ◆ **Memória vista como um vetor unidimensional**
- ◆ **Endereço de memória é um índice do vetor**
 - n bits – 2^n endereços



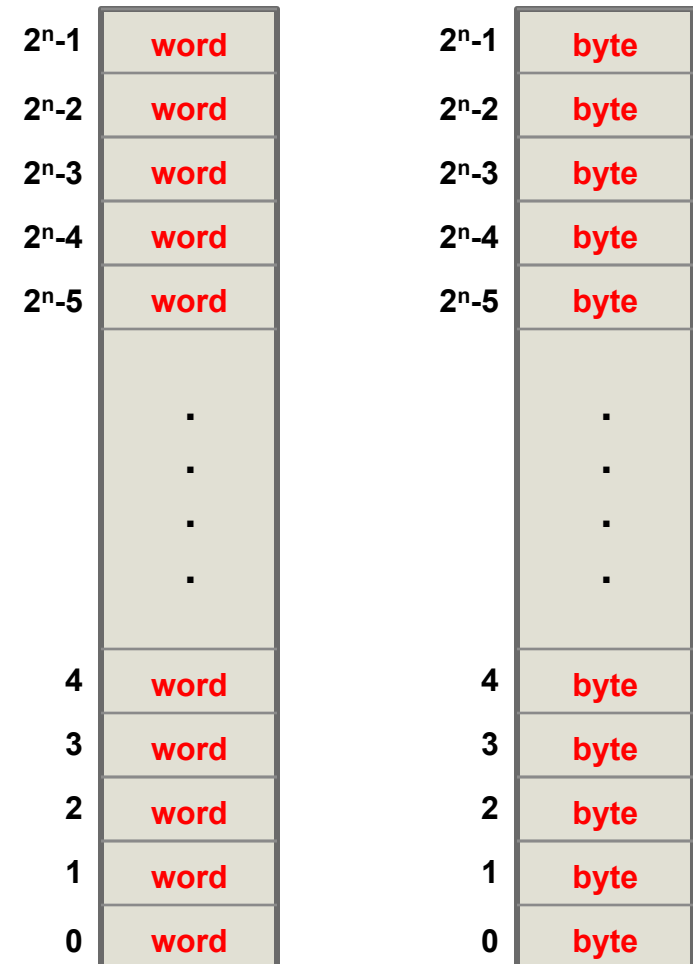
ESTRUTURA DA MEMÓRIA

- ◆ **Memória vista como um vetor unidimensional**
- ◆ **Endereço de memória é um índice do vetor**
 - n bits – 2^n endereços
- ◆ **Granularidade determina forma de endereçamento**
 - endereçado a byte



ESTRUTURA DA MEMÓRIA

- ◆ **Memória vista como um vetor unidimensional**
- ◆ **Endereço de memória é um índice do vetor**
 - n bits – 2^n endereços
- ◆ **Granularidade determina forma de endereçamento**
 - endereçado a byte
 - endereçado a palavra (*word*)

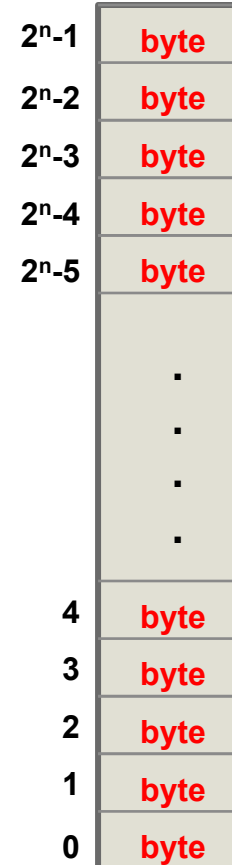


ESTRUTURA DA MEMÓRIA

◆ Alinhamento

- Objetos são armazenados em endereços múltiplos de seu tamanho

Alinhamento de palavras
em memória endereçada a
bytes

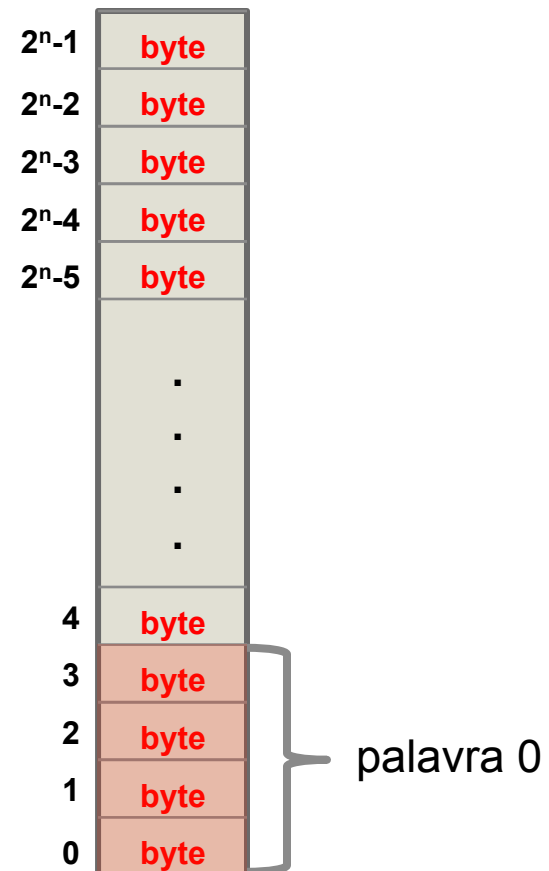


ESTRUTURA DA MEMÓRIA

◆ Alinhamento

- Objetos são armazenados em endereços múltiplos de seu tamanho

Alinhamento de palavras
em memória endereçada a
bytes

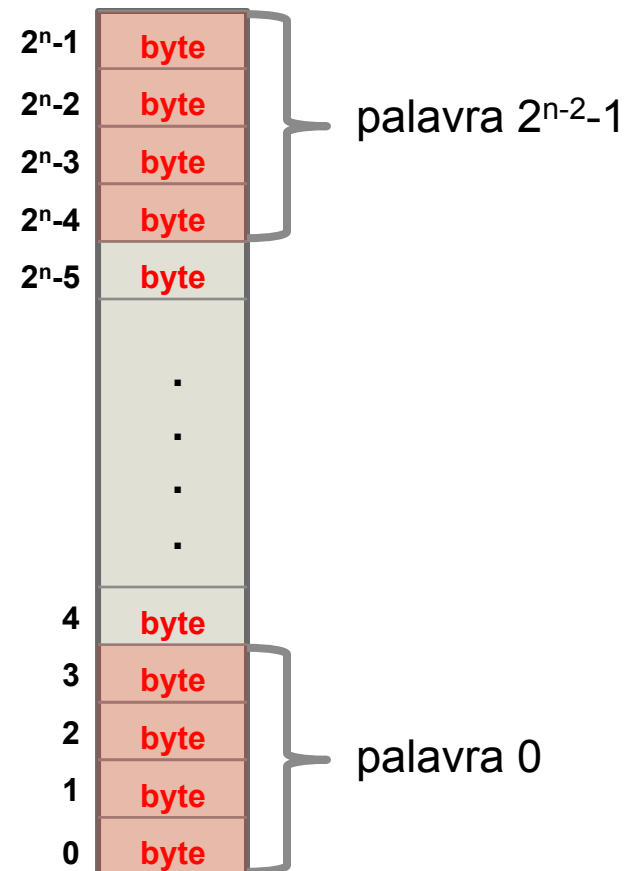


ESTRUTURA DA MEMÓRIA

◆ Alinhamento

- Objetos são armazenados em endereços múltiplos de seu tamanho

Alinhamento de palavras em memória endereçada a bytes

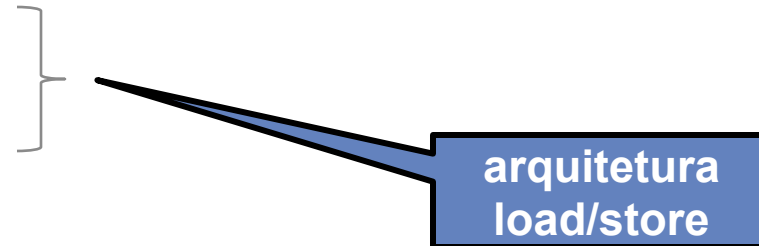


NIOS II – MEMÓRIA

- ◆ **Endereçada a bytes**

- ◆ **Instruções para transferência de dados**

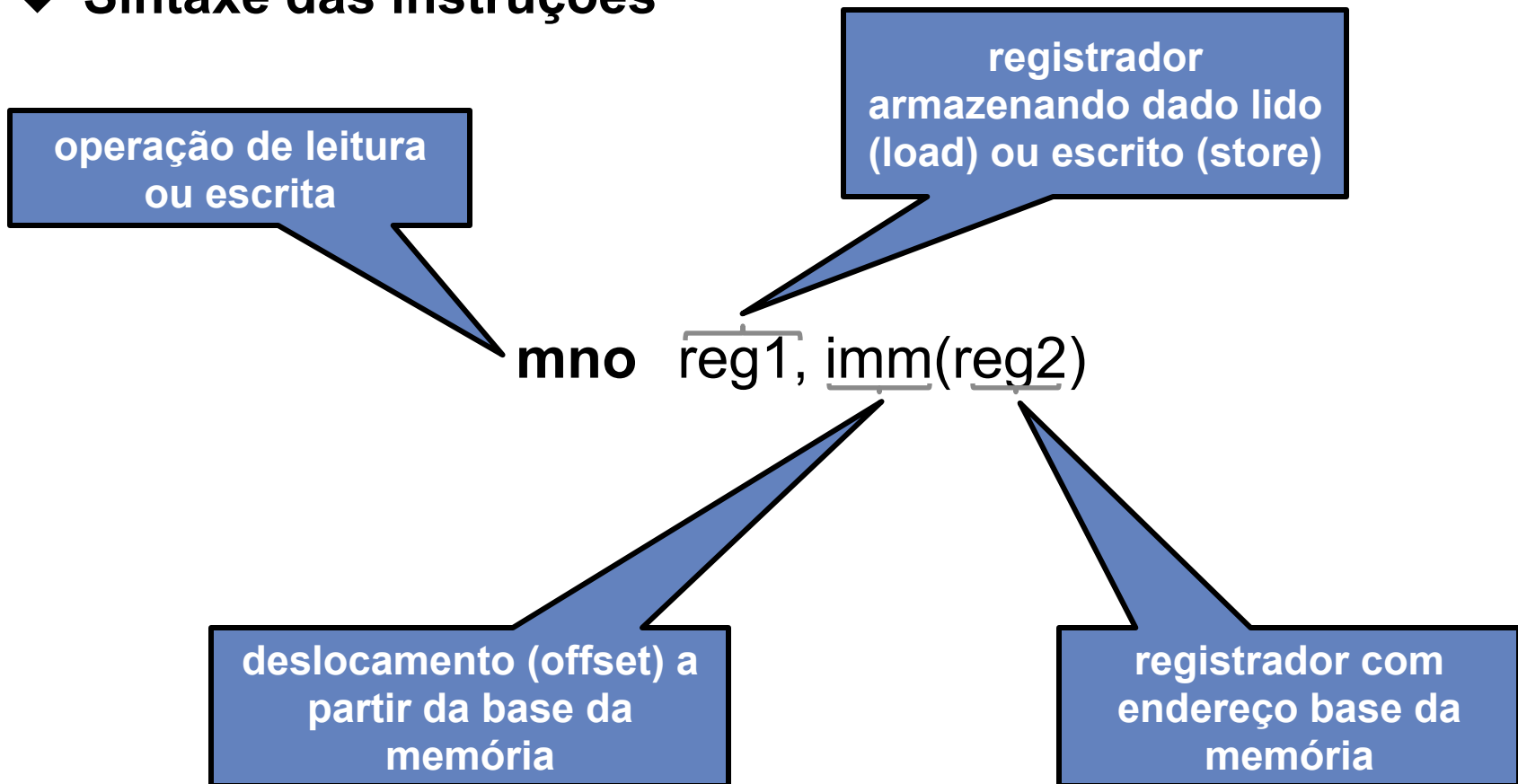
- memória → registrador
- registrador → memória



- ◆ **Toda operação em memória deve ser feita através das instruções de leitura e escrita!!!**

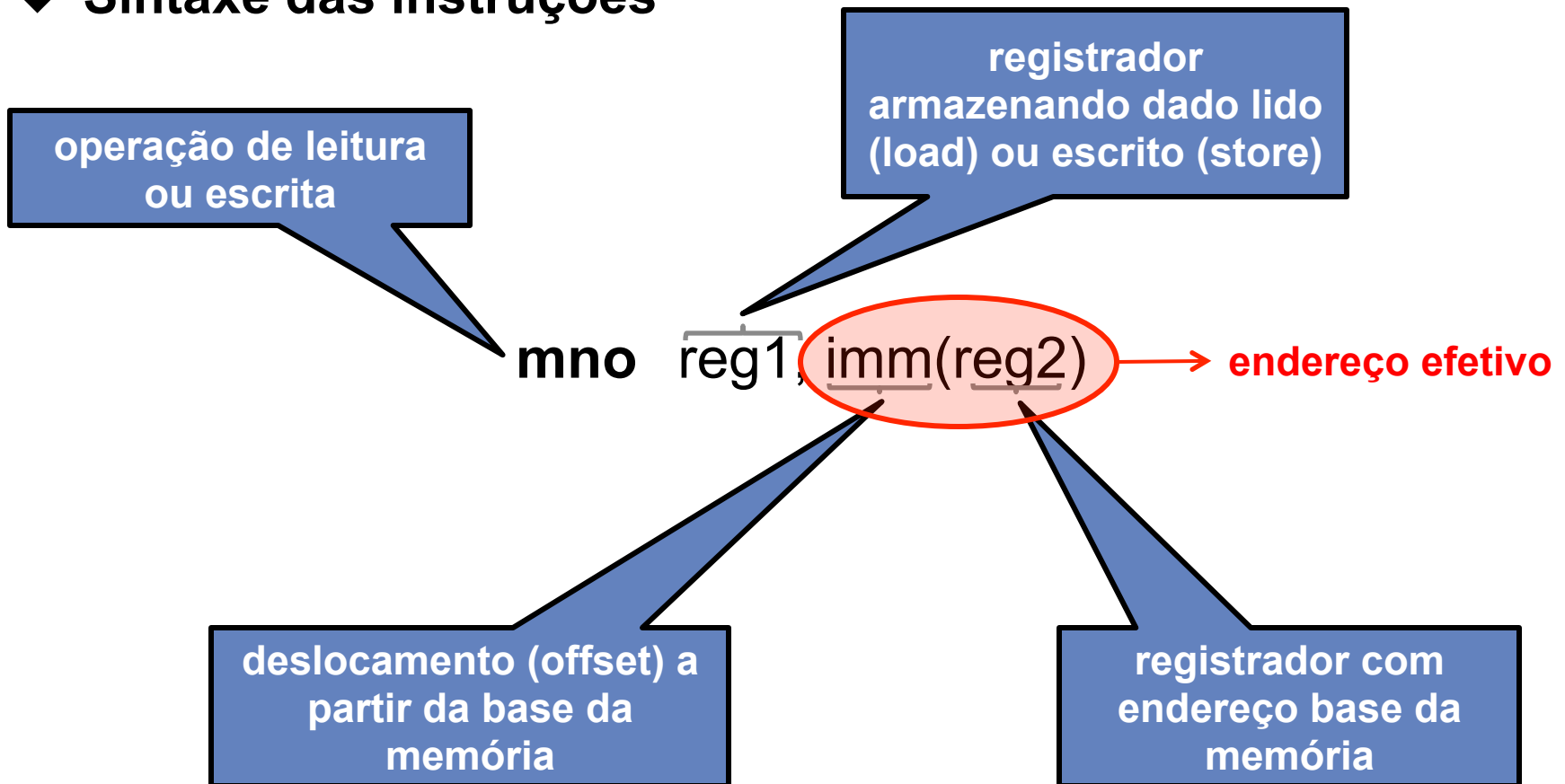
NIOS II – TRANSFERÊNCIA DE DADOS

◆ Sintaxe das instruções



NIOS II – TRANSFERÊNCIA DE DADOS

◆ Sintaxe das instruções



NIOS II – LEITURA DE MEMÓRIA

◆ Leitura de palavras: ldw

ldw r1, 12(r2)

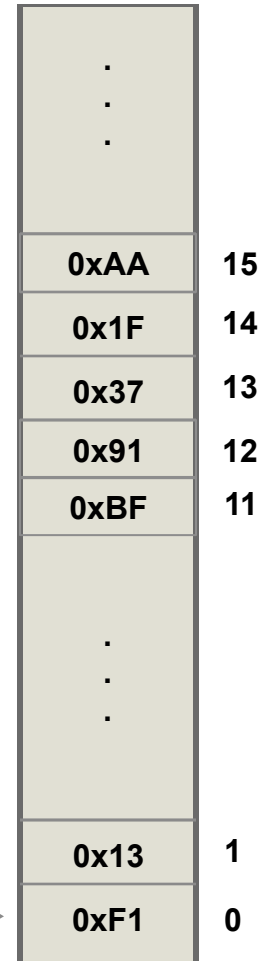


NIOS II – LEITURA DE MEMÓRIA

◆ Leitura de palavras: ldw

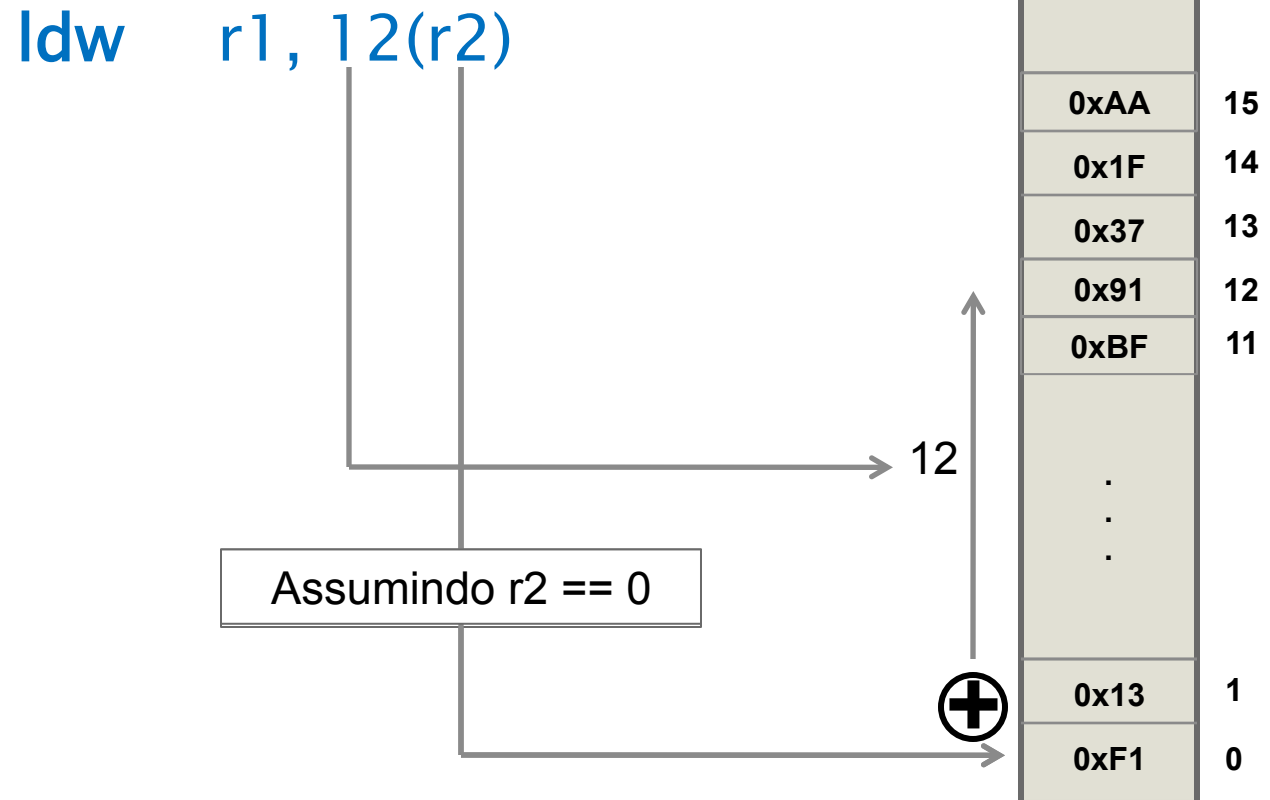
ldw r1, 12(r2)

Assumindo r2 == 0



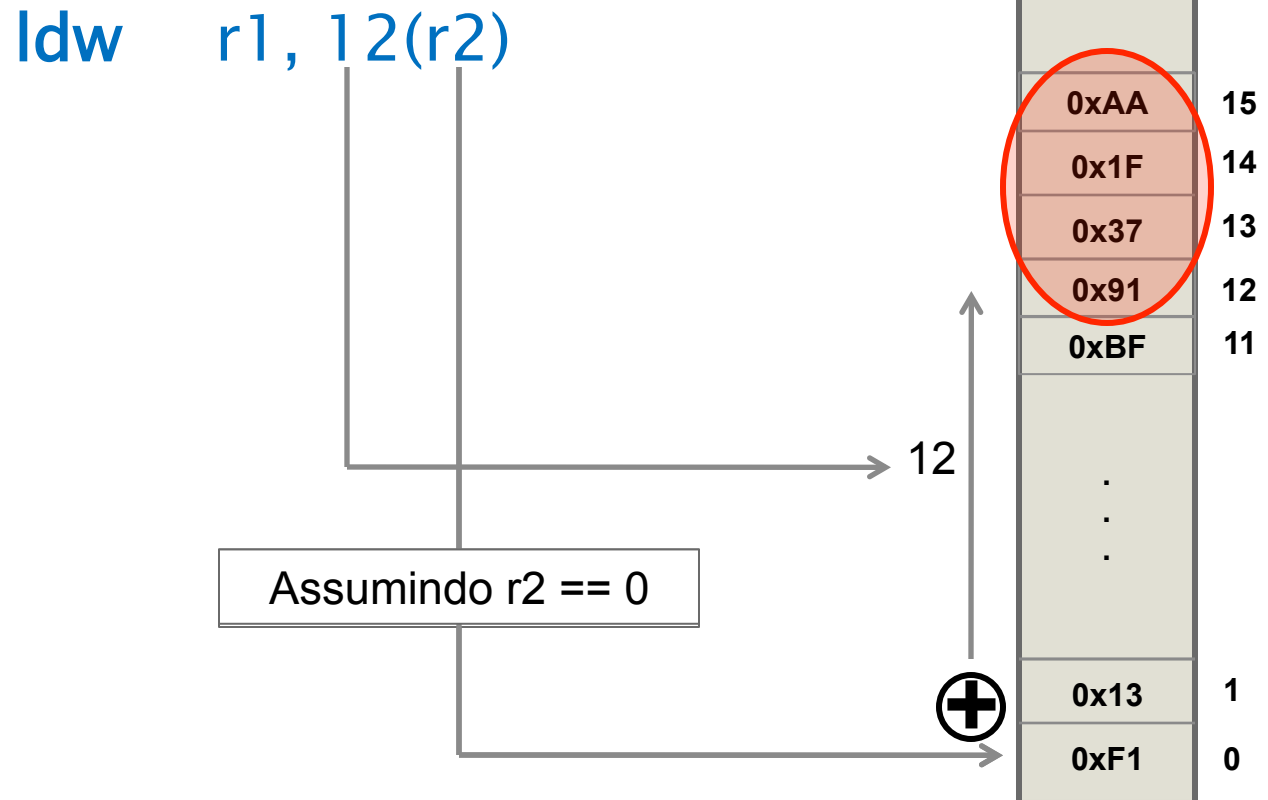
NIOS II – LEITURA DE MEMÓRIA

◆ Leitura de palavras: Idw



NIOS II - LEITURA DE MEMÓRIA

◆ Leitura de palavras: Idw



NIOS II – LEITURA DE MEMÓRIA

- ◆ **Lembre-se: endereço efetivo para `ldw` deve usar alinhamento de palavra**
- ◆ **Para acessar bytes, use `ldb` (lê byte com extensão de sinal) ou `ldbu` (lê byte sem extensão de sinal)**
- ◆ **Quais instruções são válidas (assuma `r1` alinhado)?**
 - a) `ldw r2, 3(r1)`
 - b) `ldw r2, -8(r1)`
 - c) `ldb r2, 3(r1)`
 - d) `ldbu r2, 0(zero)`

NIOS II – LEITURA DE MEMÓRIA

- ◆ **Lembre-se: endereço efetivo para `ldw` deve usar alinhamento de palavra**
- ◆ **Para acessar bytes, use `ldb` (lê byte com extensão de sinal) ou `ldbu` (lê byte sem extensão de sinal)**
- ◆ **Quais instruções são válidas (assuma `r1` alinhado)?**
 - a) `ldw r2, 3(r1)` ✗
 - b) `ldw r2, -8(r1)` ✓
 - c) `ldb r2, 3(r1)` ✓
 - d) `ldbu r2, 0(zero)` ✓

NIOS II – ESCRITA EM MEMÓRIA

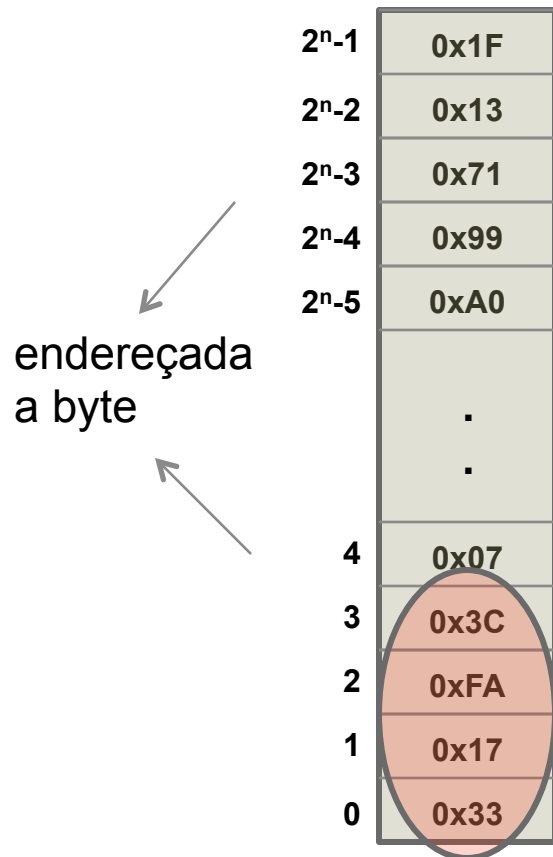
◆ Escrita de palavras: `stw` (para bytes use `stb`)

- similar à operação de leitura

```
stw    r1, 12(r2)
```

ENDIANNESS

- ◆ Quando um objeto múltiplo de byte é endereçado, qual ordem de bytes usar?



ldw r1, 0(zero)

Qual o valor lido?

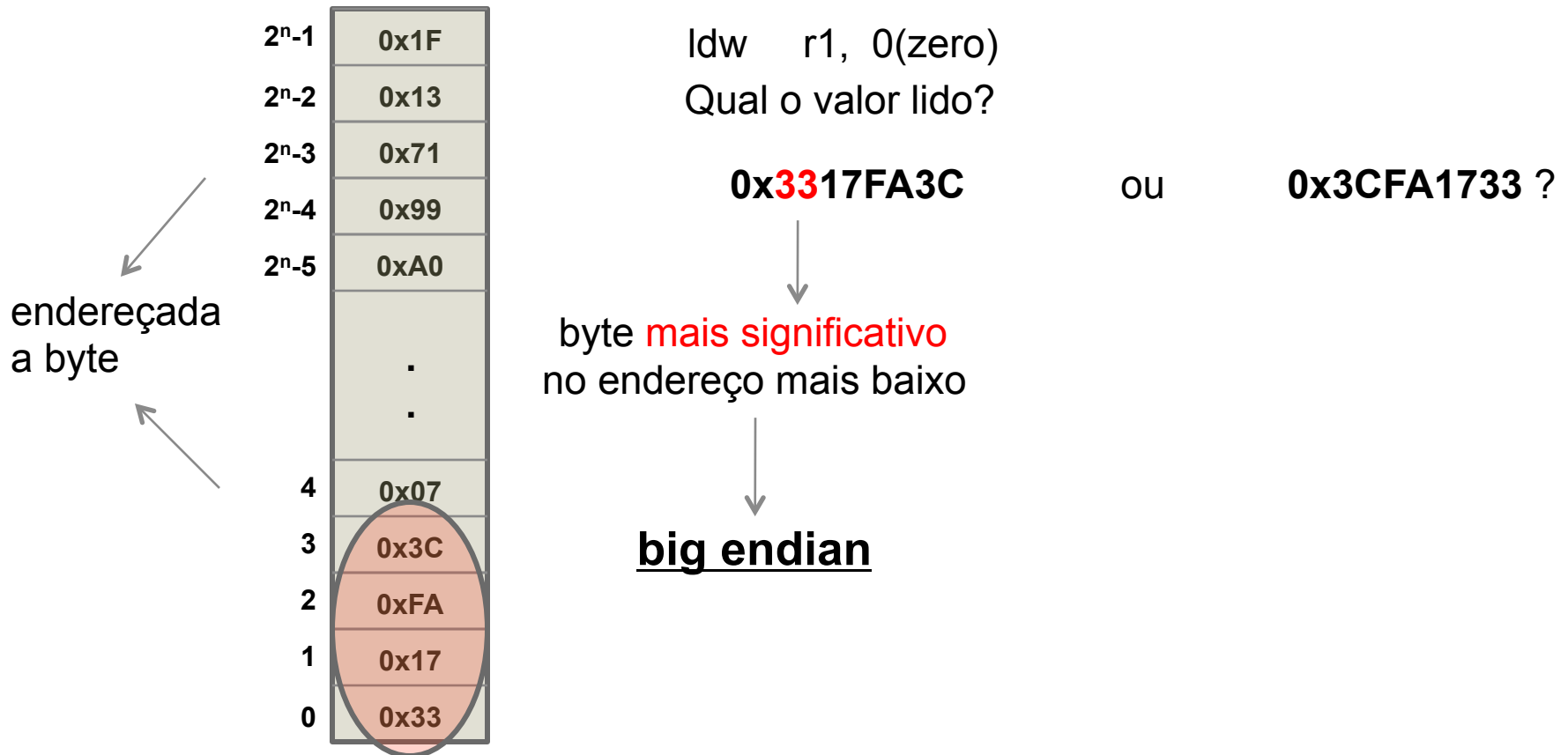
0x3317FA3C

ou

0x3CFA1733 ?

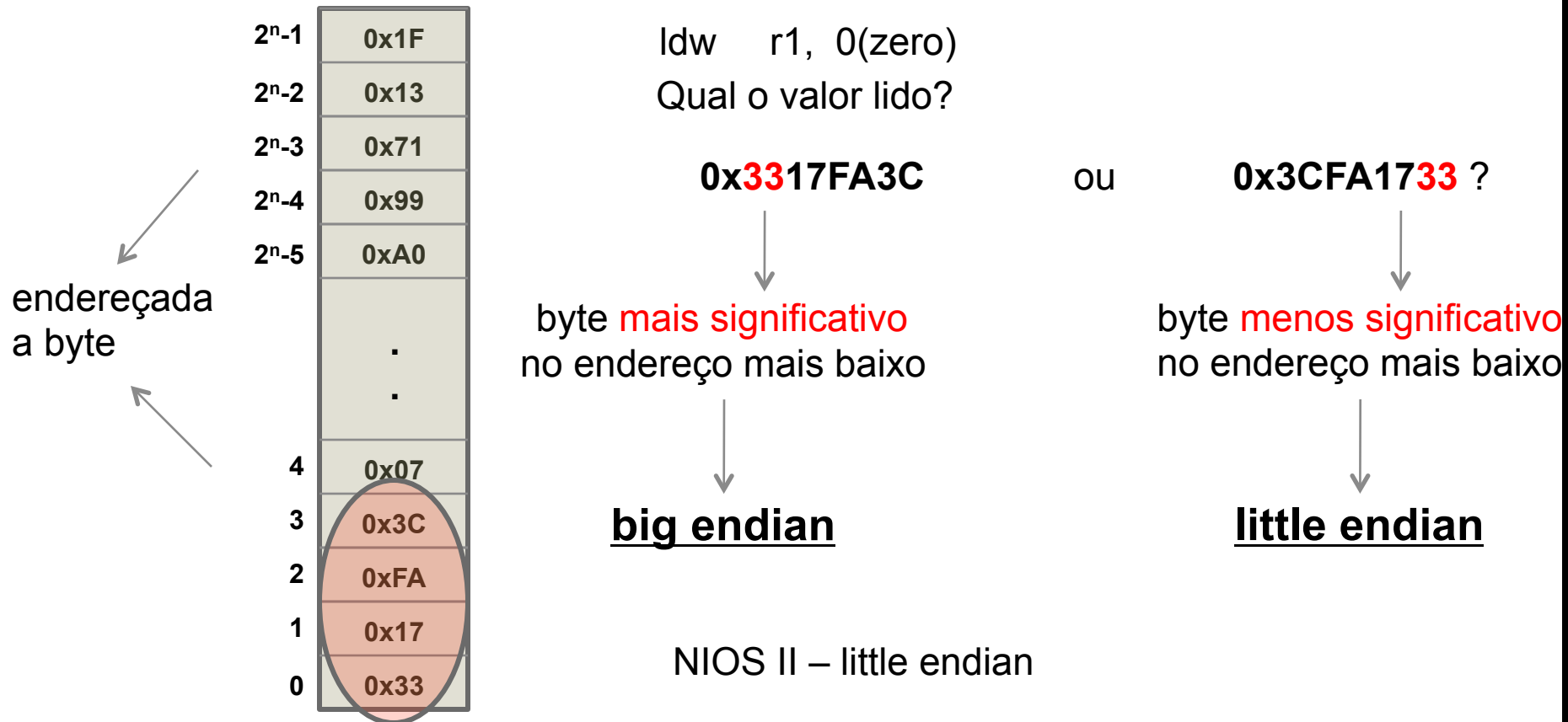
ENDIANNESS

- ◆ Quando um objeto múltiplo de byte é endereçado, qual ordem de bytes usar?



ENDIANNESS

- ◆ Quando um objeto múltiplo de byte é endereçado, qual ordem de bytes usar?



BYTES X WORDS

	·
	·
	·
5	0x1F
4	0x37
3	0xF7
2	0x8C
1	0x42
0	0x03

instrução	valor registrador (r1)
ldw r1, 3(zero)	
ldb r1, 3(zero)	
ldbu r1, 3(zero)	
ldw r1, 0(zero)	
ldb r1, 5(zero)	

BYTES X WORDS

	·
	·
	·
5	0x1F
4	0x37
3	0xF7
2	0x8C
1	0x42
0	0x03

instrução	valor registrador (r1)
ldw r1, 3(zero)	ND
ldb r1, 3(zero)	0xFFFFFFFF7
ldbu r1, 3(zero)	0x000000F7
ldw r1, 0(zero)	depende do endian, assumo um: little: 0xF78C4203 big: 0x03428CF7
ldb r1, 5(zero)	0x0000001F

ATÉ AGORA ...

◆ Instruções aritméticas

- add, sub, addi

◆ Instruções de memória

- ldw, stw
- ldb, ldbu, stb

◆ Faltando ...

ATÉ AGORA ...

◆ Instruções aritméticas

- add, sub, addi

◆ Instruções de memória

- ldw, stw
- ldb, ldbu, stb

◆ Faltando ...

- Instruções de desvio (controle)

NIOS II – INSTRUÇÕES DE DESVIO

◆ Alteram o fluxo de execução

◆ Desvio condicional

- condição é testada
- desvio pode ser tomado ou não

◆ Desvio incondicional

- desvio sempre tomado

◆ Rótulo

- determina próxima instrução a ser executada caso desvio seja tomado

NIOS II – DESVIO CONDICIONAL

◆ **Condição testada**

- igualdade entre dois registradores

◆ **Duas instruções principais**

beq reg1, reg2, rótulo

salta para rótulo se
reg1 == reg2

bne reg1, reg2, rótulo

salta para rótulo se
reg1 != reg2

NIOS II – DESIGUALDADES

- ◆ **Para desigualdades as seguintes instruções podem ser usadas**
 - blt (branch if LESS THAN)
 - bltu (branch if LESS THAN – unsigned version)
 - bge (branch if GREATER THAN OR EQUAL TO)
 - bgeu (branch if GREATER THAN OR EQUAL TO – unsigned version)

- ◆ **Não há instruções nativas para LESS THAN OR EQUAL TO ou GREATER THAN**
 - Como implementá-las (software)?

NIOS II – DESVIO INCONDICIONAL

◆ Salto sempre tomado

- necessário especificar apenas o rótulo

```
br DESTINO
```

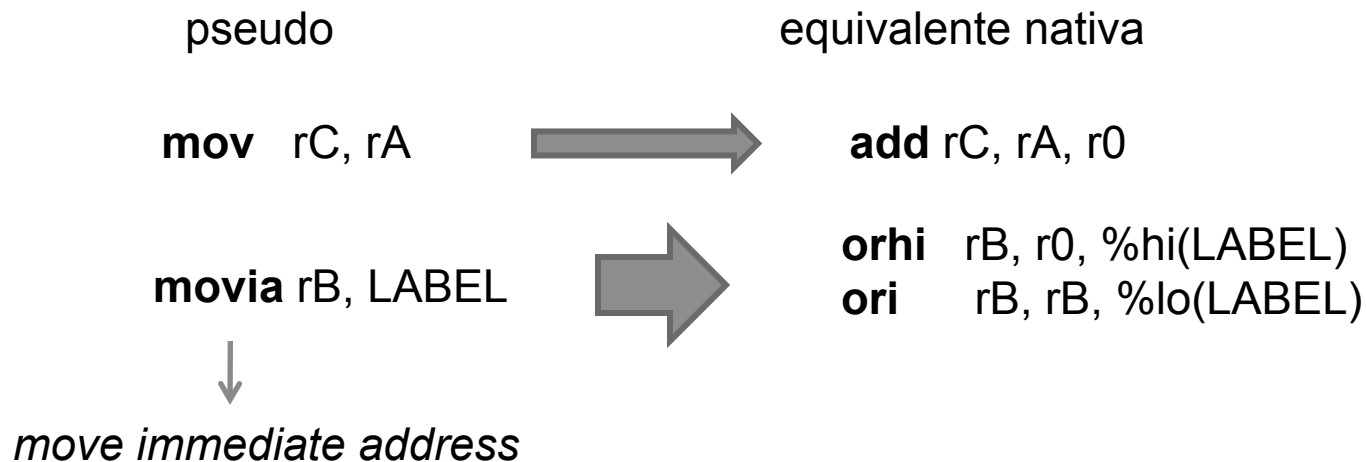
◆ Salto indireto

- execução é desviada para endereço contido no registrador especificado

```
jmp registrador
```

PSEUDO INSTRUÇÕES

- ◆ São instruções disponibilizadas pelo montador, mas que não fazem parte do conjunto de instruções
 - Objetivo de facilitar a vida do programador
- ◆ Montador expande uma pseudo instrução em uma ou mais instruções nativas



PSEUDO INSTRUÇÕES

Pseudo-Instruction	Equivalent Instruction
bgt rA, rB, label	blt rB, rA, label
bgtu rA, rB, label	bltu rB, rA, label
ble rA, rB, label	bge rB, rA, label
bleu rA, rB, label	bgeu rB, rA, label
cmpgt rC, rA, rB	cmplt rC, rB, rA
cmpgti rB, rA, IMMED	cmpgei rB, rA, (IMMED+1)
cmpgtu rC, rA, rB	cmpltu rC, rB, rA
cmpgtui rB, rA, IMMED	cmpgeui rB, rA, (IMMED+1)
cmple rC, rA, rB	cmpge rC, rB, rA
cmplei rB, rA, IMMED	cmplti rB, rA, (IMMED+1)
cmpleu rC, rA, rB	cmpgeu rC, rB, rA
cmpleui rB, rA, IMMED	cmpltui rB, rA, (IMMED+1)
mov rC, rA	add rC, rA, r0
movhi rB, IMMED	orhi rB, r0, IMMED
movi rB, IMMED	addi, rB, r0, IMMED
movia rB, label	orhi rB, r0, %hiadj(label) addi, rB, r0, %lo(label)
movui rB, IMMED	ori rB, r0, IMMED
nop	add r0, r0, r0
subi rB, rA, IMMED	addi rB, rA, (-IMMED)

DIRETIVAS DO MONTADOR

- ◆ **Diretivas são ações executadas pelo montador em tempo de montagem (não execução!)**
- ◆ **Vamos usar nessa disciplina o montador da GNU (gas), no qual diretivas iniciam-se com o ponto (.)**
- ◆ **Exemplos**
 - **.equ** SIMBOLO, VALOR /* define SIMBOLO com o valor VALOR */
 - **.global** SIMBOLO /* torna SIMBOLO visível para fora do arquivo objeto */
 - **.org** ENDERECO /* a partir desse ponto a montagem assume ENDERECO */
 - **.skip** TAMANHO /* emite TAMANHO bytes (zerados) */
 - **.word** VALOR /* uma palavra de 32 bits (VALOR) é emitida */
 - **.end** /* marca o fim do código em linguagem de montagem */

MATERIAL DE REFERÊNCIA

- ◆ **Para diretivas de montagem e pseudo instruções**
 - Introduction to the Altera Nios II Soft Processor
- ◆ **Para descrição das instruções e formatos**
 - Nios II Instruction Set Reference
- ◆ **Para detalhes do processador Nios II**
 - Nios II Processor Reference - Handbook

EXEMPLOS E DICAS

NIOS II – INSTRUÇÕES ARITMÉTICAS

- ◆ Qual a sequência de instruções equivalente à expressão abaixo?

$$a = b + c + d - e$$

NIOS II – INSTRUÇÕES ARITMÉTICAS

- ◆ Qual a sequência de instruções equivalente à expressão abaixo?

$$a = b + c + d - e$$

```
add  r1, r5, r6  # temp0 = b+c
sub  r2, r7, r8  # temp1 = d-e
add  r3, r1, r2  # a = (b+c) + (d-e)
```

NIOS II – LEITURA DE MEMÓRIA

- ◆ Sequência de instruções equivalente à expressão:

vetor de
palavras



$$a = b + A[5]$$

use: $a = r1, b = r2, A = r3$

NIOS II – LEITURA DE MEMÓRIA

- ◆ Sequência de instruções equivalente à expressão:

vetor de
palavras

$$a = b + A[5]$$

use: $a = r1$, $b = r2$, $A = r3$

```
ldw   r4, 20(r3)      # temp = A[5]
add   r1, r2, r4      # a = b + A[5]
```

DE C PARA ASSEMBLY

- ◆ Transformar o seguinte programa para linguagem de montagem do Nios II:

```
// assuma: endereço base do vetor v no registrador r2 e  
// k em r1 -- v é vetor de palavras (int em C)
```

```
int temp = v[k];  
v[k] = v[k+1];  
v[k+1] = temp;
```

LINGUAGEM DE MONTAGEM

add	r5, r1, r1	<i># r5 = k*2</i>
add	r5, r5, r5	<i># r5 = k*4</i>
add	r5, r2, r5	<i># r5 = v + (k*4)</i>
ldw	r6, 0(r5)	<i># temp = v[k]</i>
ldw	r7, 4(r5)	<i># r7 = v[k+1]</i>
stw	r7, 0(r5)	<i># v[k] = v[k+1]</i>
stw	r6, 4(r5)	<i># v[k+1] = temp</i>

NIOS II – DESVIO CONDICIONAL

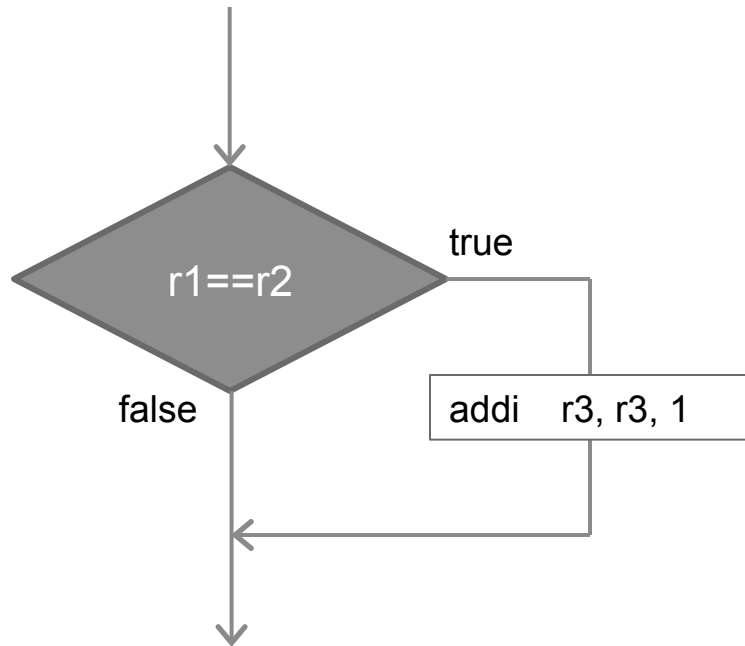
◆ Exemplo

- Se r1 e r2 forem iguais, incrementar r3

NIOS II – DESVIO CONDICIONAL

◆ Exemplo

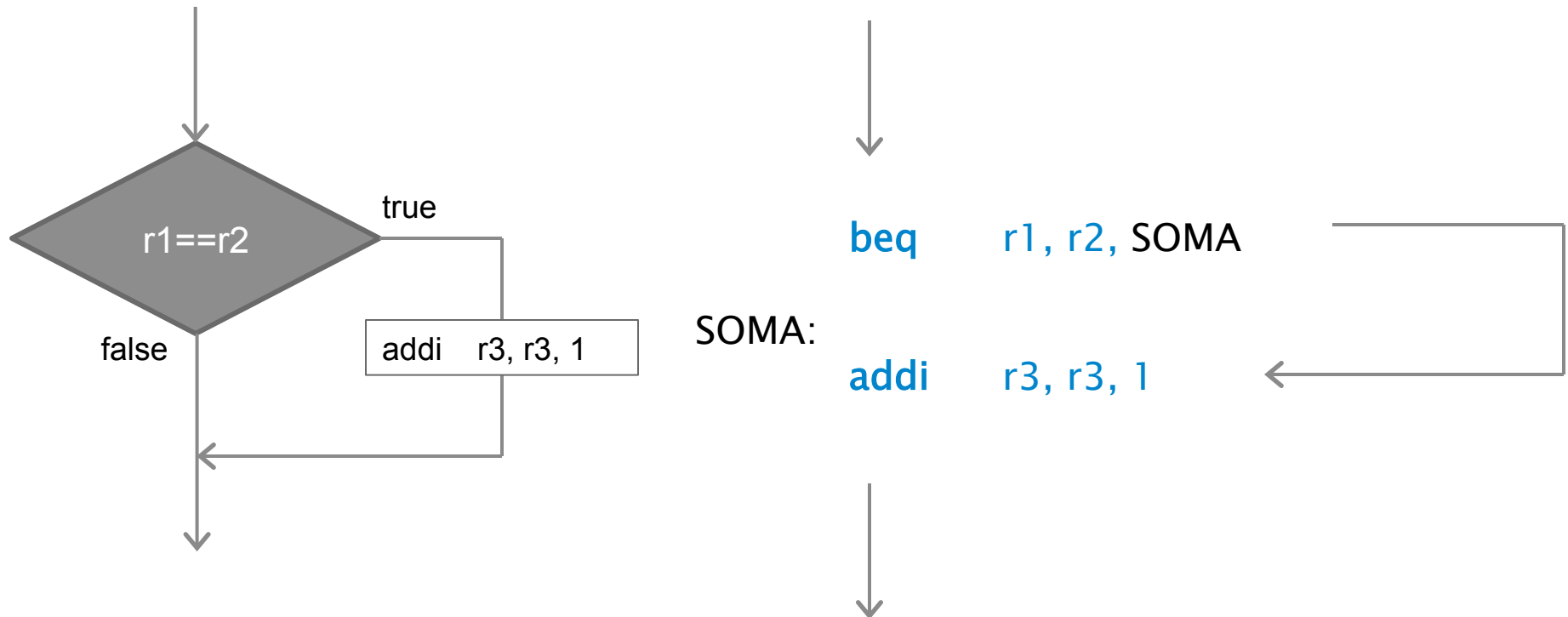
- Se r1 e r2 forem iguais, incrementar r3



NIOS II – DESVIO CONDICIONAL

◆ Exemplo

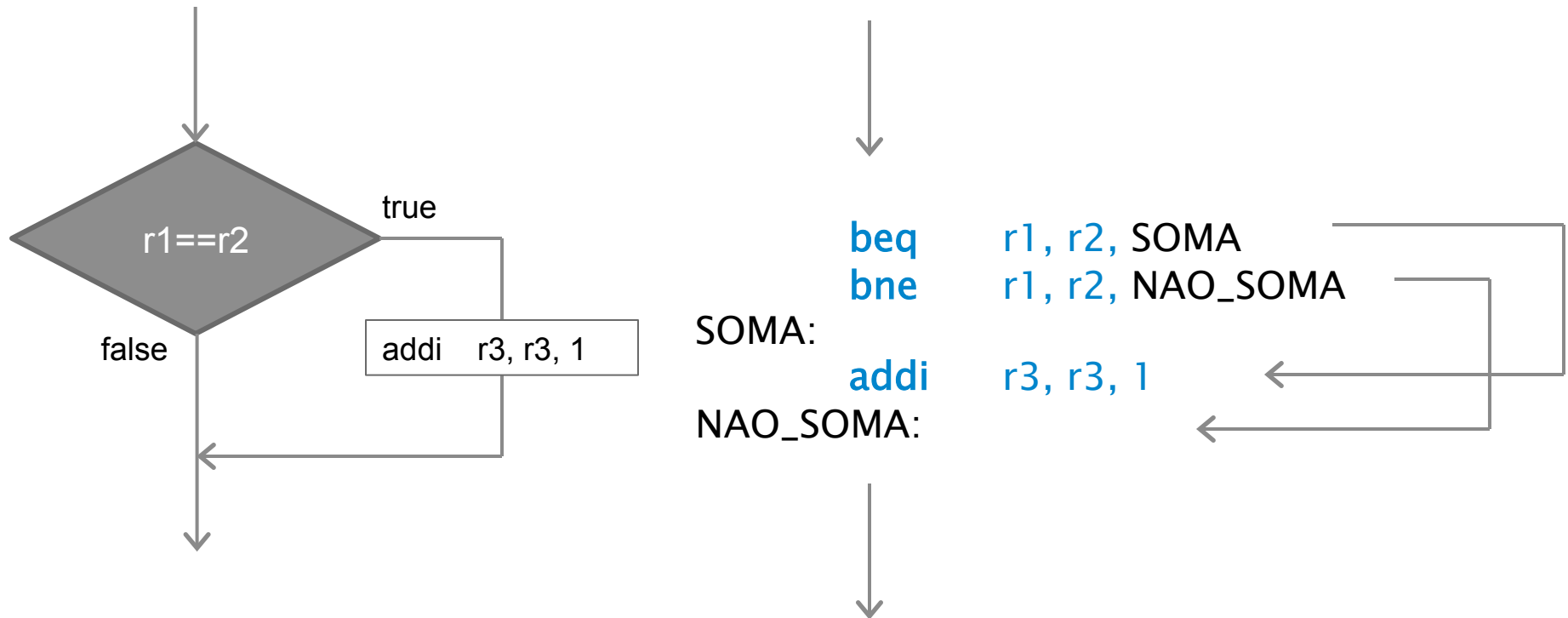
- Se r1 e r2 forem iguais, incrementar r3



NIOS II – DESVIO CONDICIONAL

◆ Exemplo

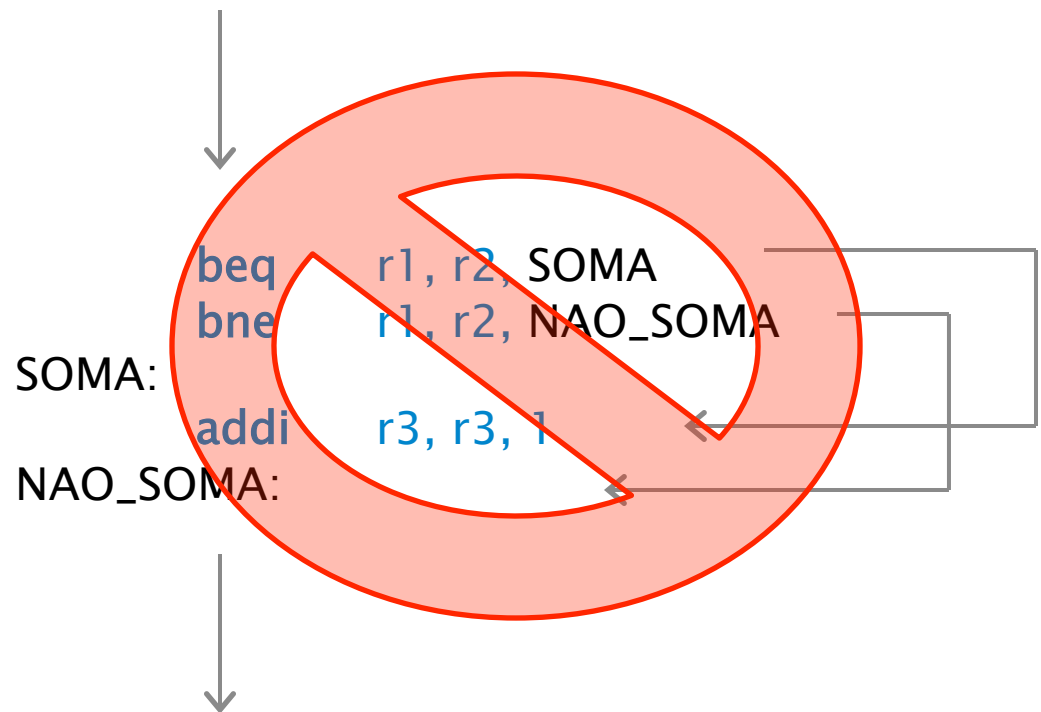
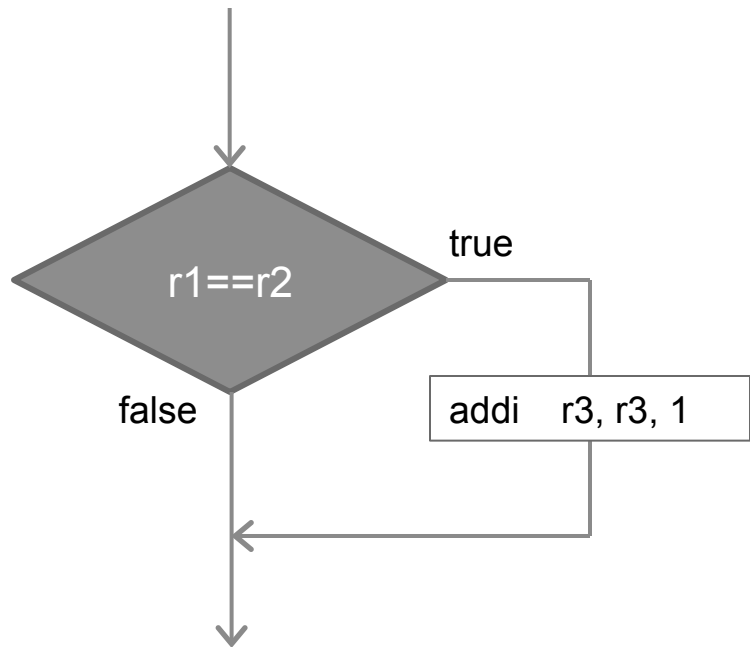
- Se r1 e r2 forem iguais, incrementar r3



NIOS II – DESVIO CONDICIONAL

◆ Exemplo

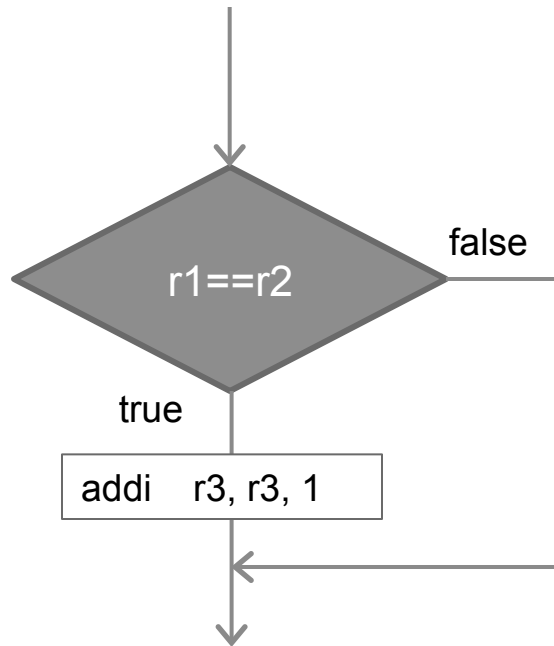
- Se r1 e r2 forem iguais, incrementar r3



NIOS II – DESVIO CONDICIONAL

◆ Exemplo

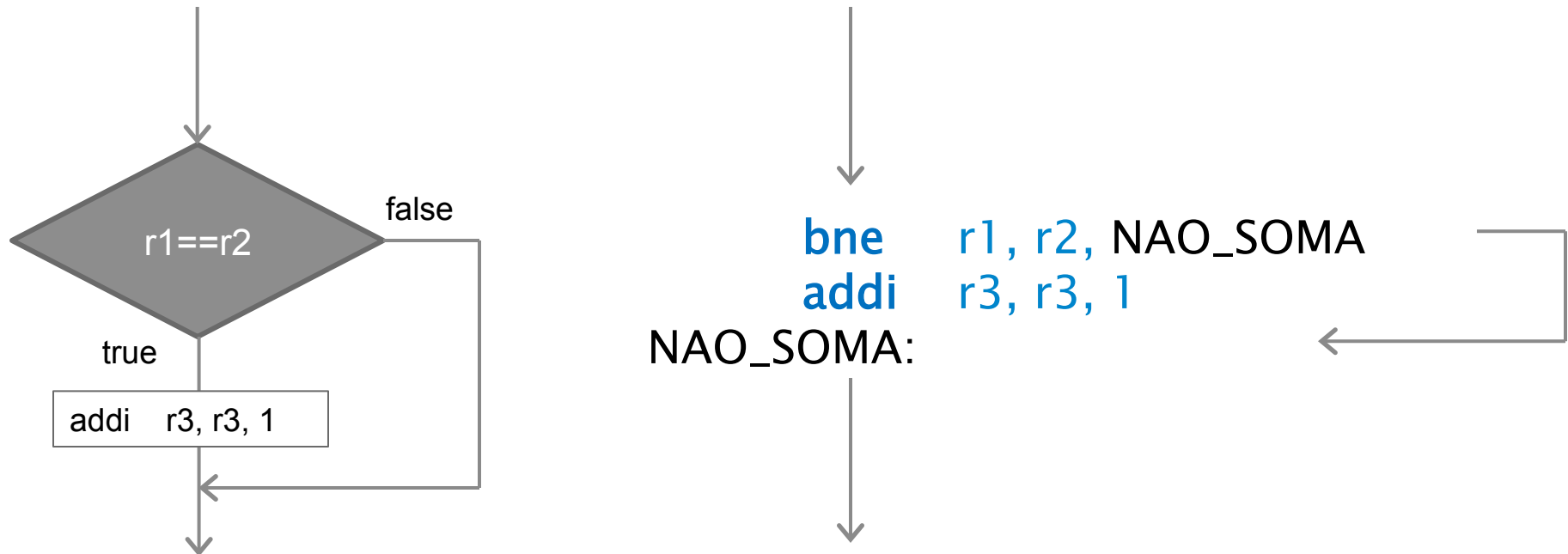
- Se r1 e r2 forem iguais, incrementar r3



NIOS II – DESVIO CONDICIONAL

◆ Exemplo

- Se r1 e r2 forem iguais, incrementar r3



LAÇO FOR

- ◆ Transformar para linguagem de montagem do MIPS:

// assumo: r1 = sum, r2 = i

```
int sum = 0;
```

```
for (int i=1; i<101; i++)
```

```
    sum = sum + 2;
```

LAÇO FOR

```
    addi    r1, zero, 0           # sum = 0
    addi    r2, zero, 1           # i = 1
    addi    r3, zero, 101        # temp= 101
FOR:
    bge     r2, r3, FIM          # se i >= 101, acaba
    addi    r1, r1, 2            # sum = sum + 2
    addi    r2, r2, 1            # i++
    br      FOR
FIM:
```