

# **Arquitetura de Computadores II**

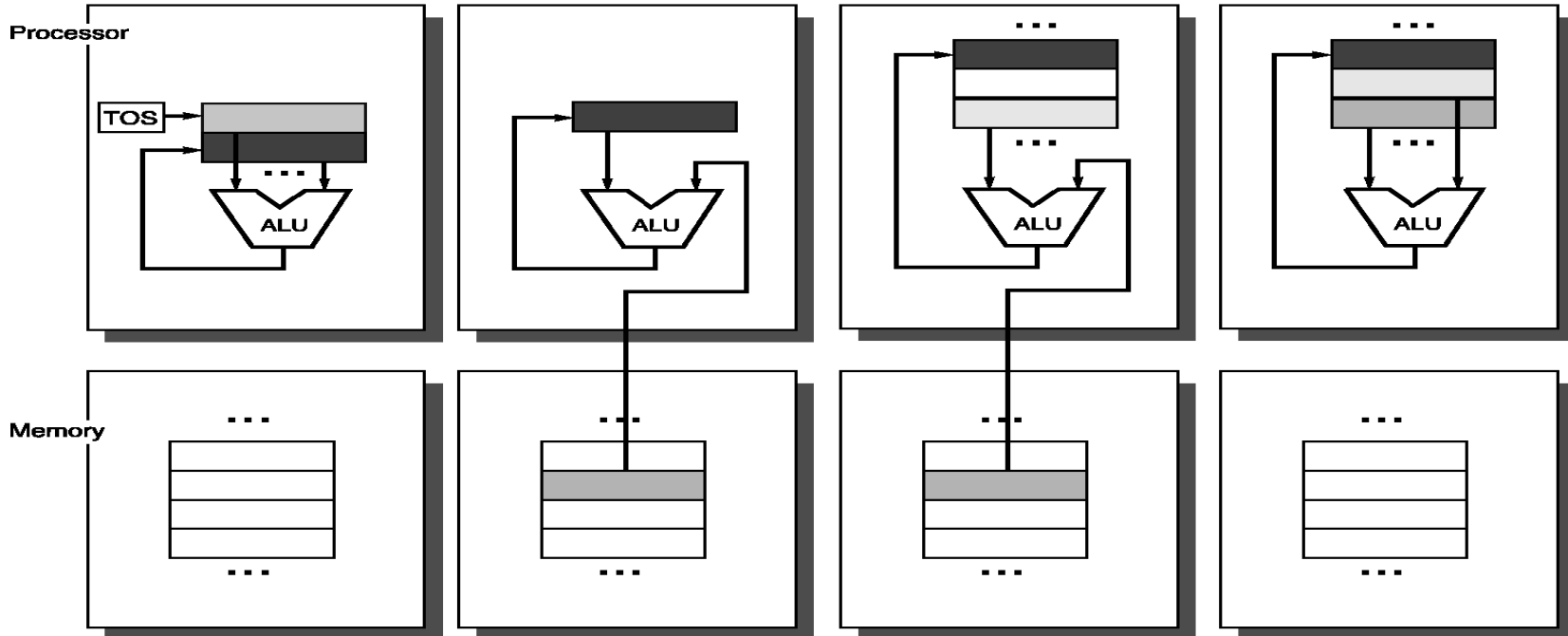
## **MIPS64**

**Prof. Gabriel P. Silva**

# Tipos de Arquitetura

**C := A+B:**

Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1, A	Load R1, A
Push B	Add B	Add R1, B	Load R2, B
Add	Store C	Store C, R1	Add R3, R1, R2
Pop C			Store C, R3



# MIPS64

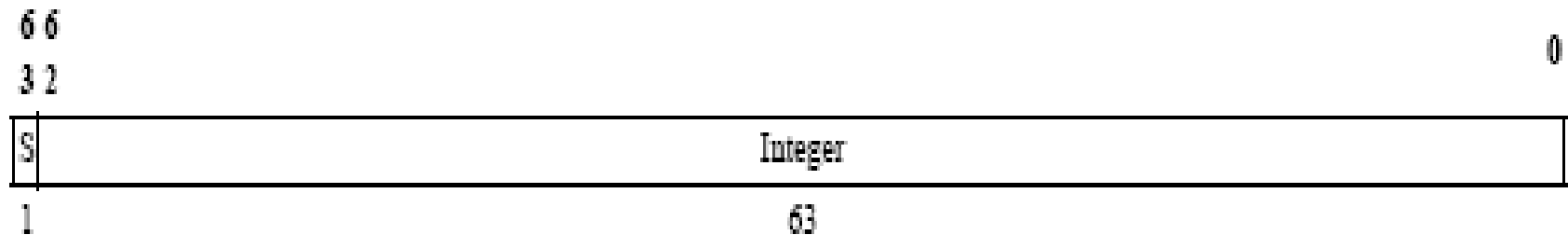
- **Arquitetura do tipo Load/Store**
- **32 registradores de uso geral de 64 bits.**
- **32 registradores de ponto flutuante de 64 bits.**
- **Tipos de dados:**
  - **Inteiros: Byte (8 bits), Half-word (16 bits), Word (32 bits) e Long-word (64 bits)**
  - **Ponto Flutuante: padrão IEEE 754 de precisão simples (32 bits) ou dupla (64 bits).**
  - **Modos de Endereçamento: Deslocamento, Imediato e Registrador Indireto.**
  - **As instruções são um superconjunto do MIPS32.**
  - **Tamanho fixo das instruções (32 bits).**
  - **Ortogonalidade e simplicidade.**

# Operandos Inteiros

Figure 5-4 Word Fixed Point Format (W)



Figure 5-5 Longword Fixed Point Format (L)



# Operandos em Ponto Flutuante

Figure 5-1 Single-Precisions Floating Point Format (S)

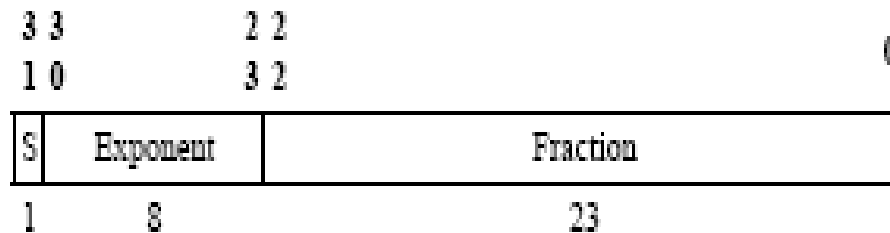
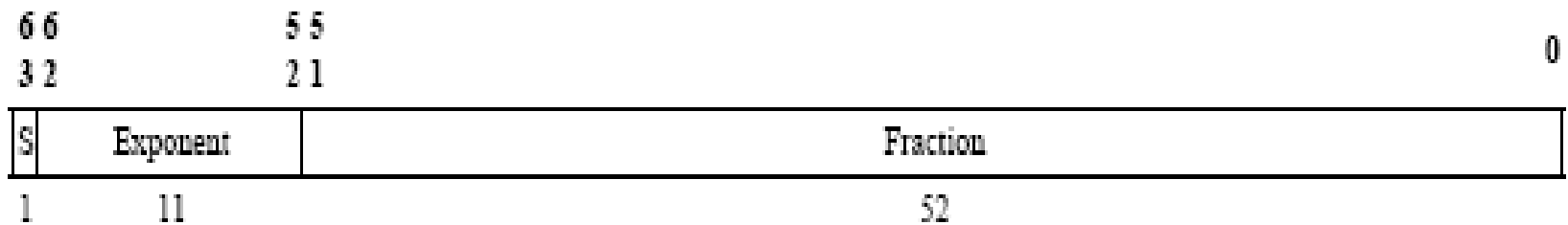


Figure 5-2 Double-Precisions Floating Point Format (D)



# Alinhamento

Aumento dos  
Endereços

The diagram illustrates memory alignment on a 32-bit address bus. A horizontal arrow at the top points left, labeled 'Aumento dos Endereços'. Below it, a table represents memory addresses from 7 to 0. Vertical dashed lines mark the boundaries for alignment to 4, 2, and 1. Three horizontal bars represent words in memory: the first is aligned to the 4 boundary, the second is not aligned to the 2 boundary, and the third is not aligned to the 1 boundary.

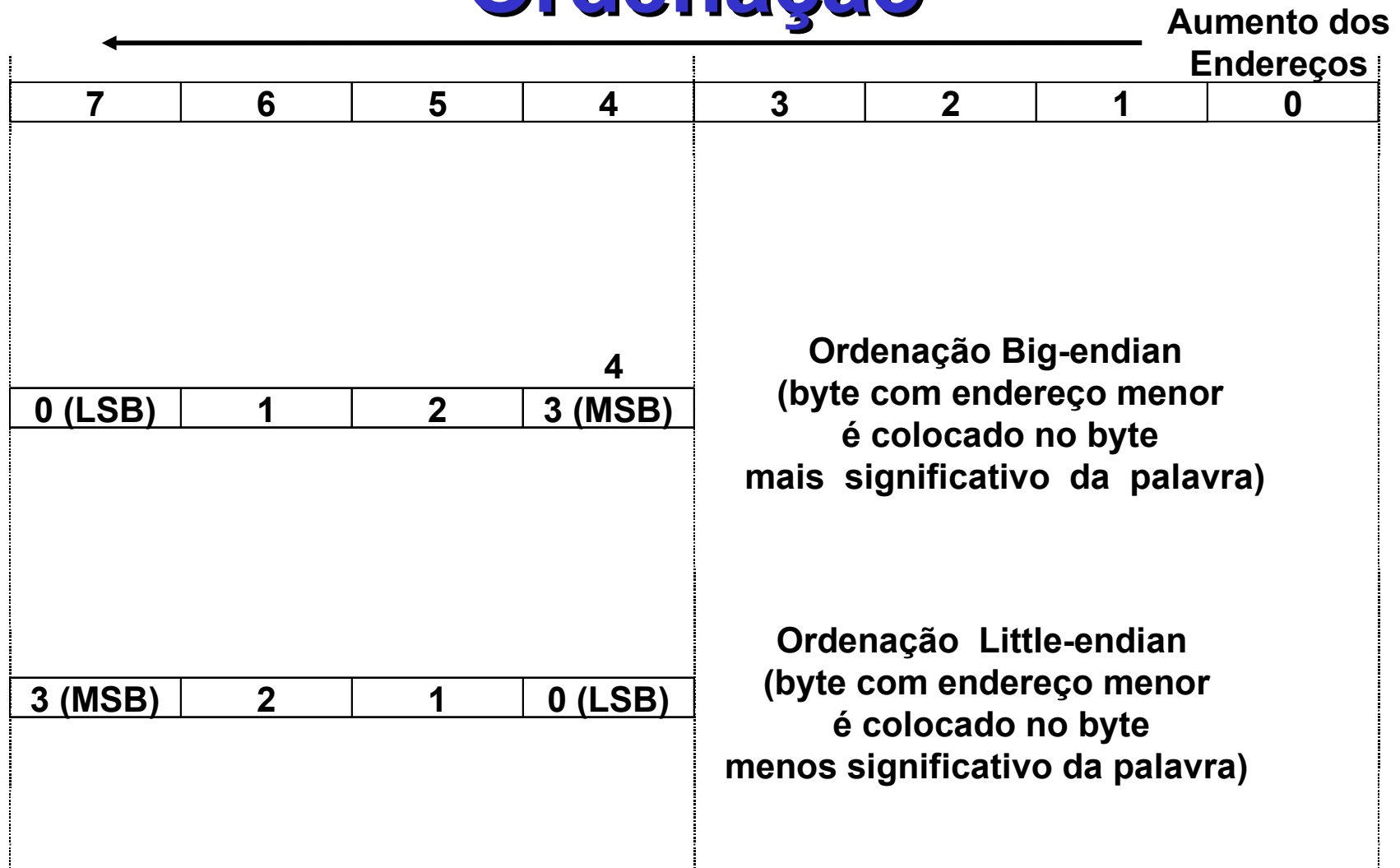
7	6	5	4	3	2	1	0
			4				
					2		
						1	

Palavra alinhada (endereço múltiplo de 4).

Palavra não alinhada (endereço múltiplo de 2).

Palavra não alinhada (endereço múltiplo de 1).

# Ordenação



# Ordenação

When configured in **big-endian** order, byte 0 is the most-significant (left-hand) byte. Figure 2-10 shows this configuration.

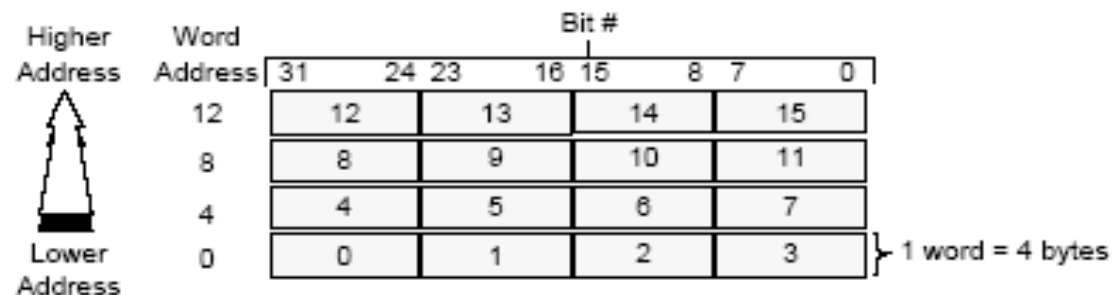


Figure 2-10 Big-Endian Byte Ordering

## 2.9.6.2 Little-Endian Order

When configured in **little-endian** order, byte 0 is always the least-significant (right-hand) byte. Figure 2-11 shows this configuration.

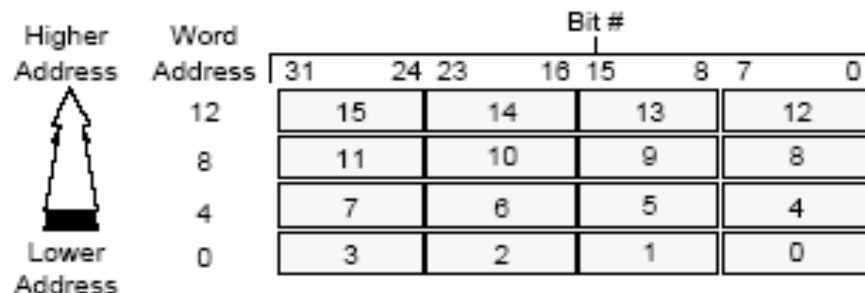


Figure 2-11 Little-Endian Byte Ordering

# Ordenação

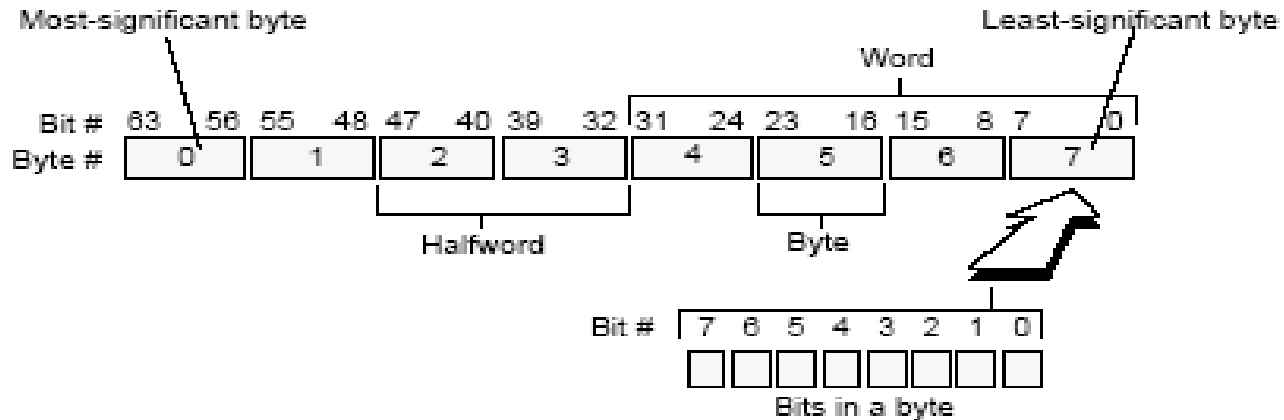


Figure 2-12 Big-Endian Data in Doubleword Format

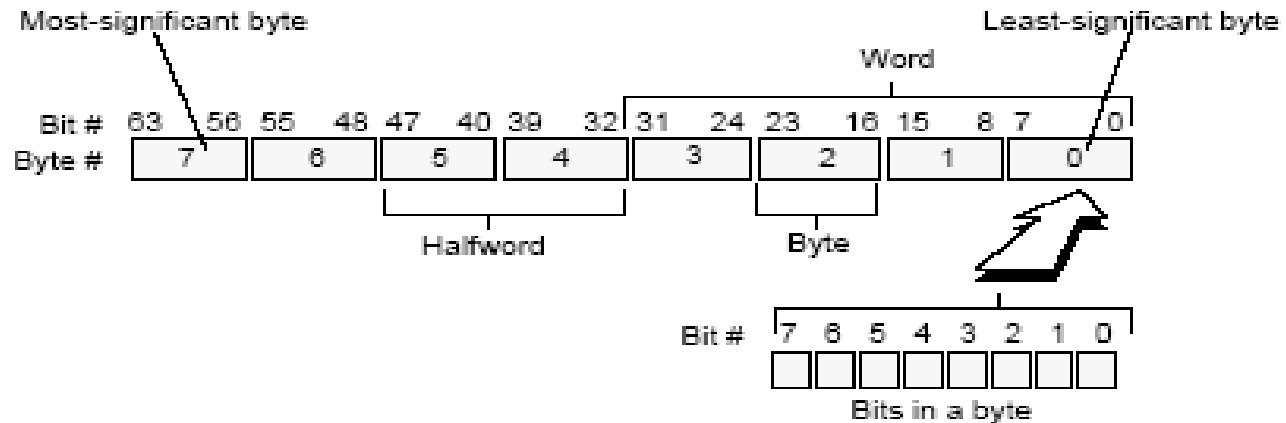


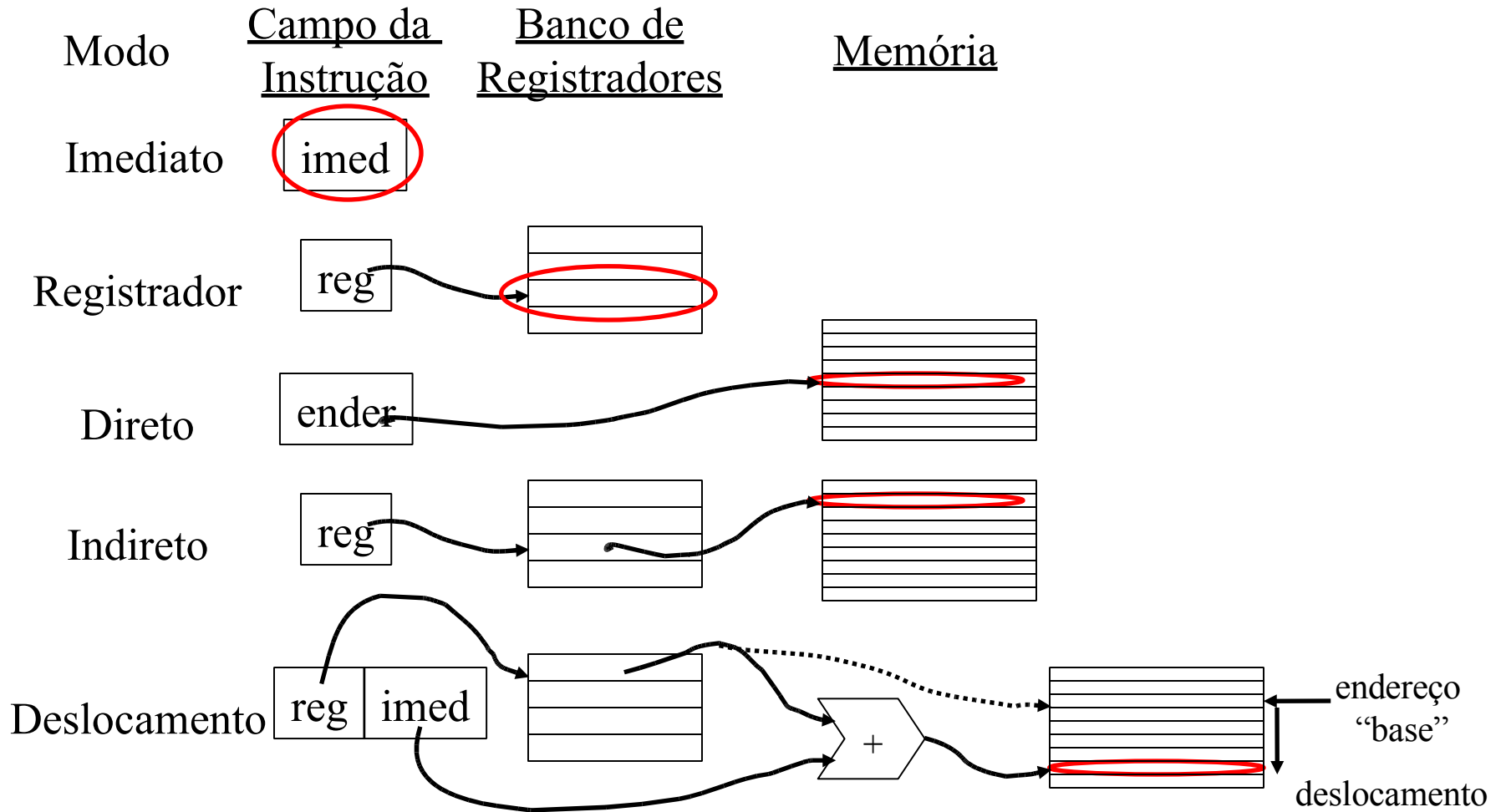
Figure 2-13 Little-Endian Data in Doubleword Format

# Modos de Endereçamento

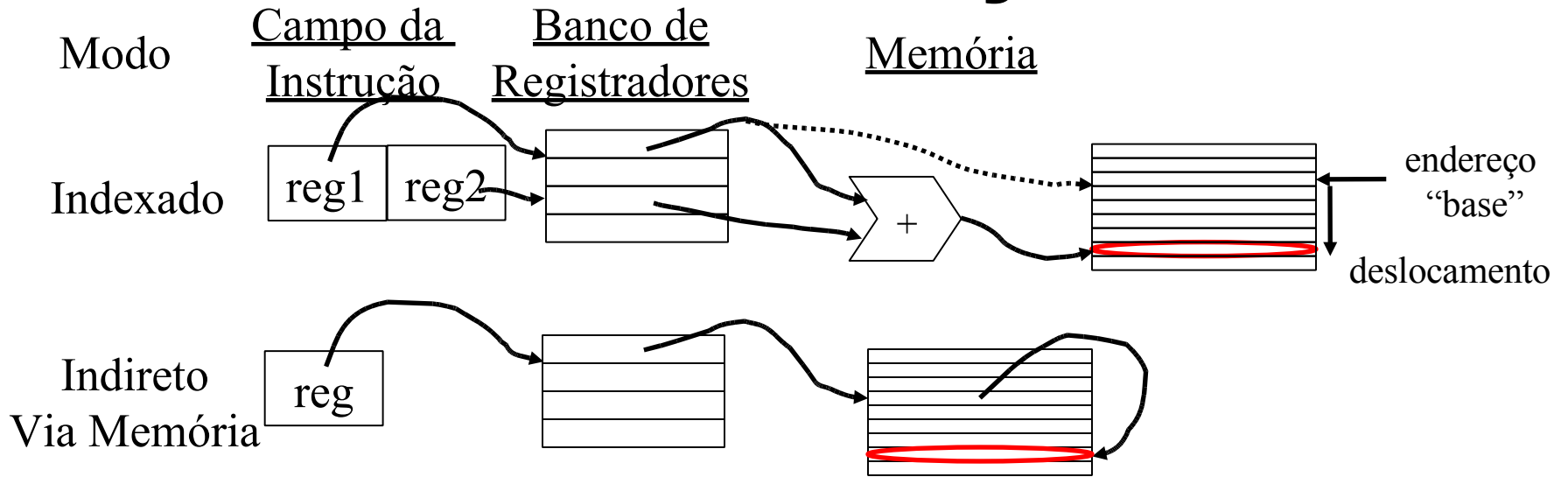
Modo	Exemplo	Significado (RTL)
Imediato	dadd R4, R4, 3	$R[4] \leftarrow R[4]+3$
Registrador	dadd R4, R4, R3	$R[4] \leftarrow R[4]+R[3]$
Direto	dadd R1, (1001)	$R[1] \leftarrow R[1]+M[1001]$
Indireto	dadd R4, (R1)	$R[4] \leftarrow R[4]+M[R[1]]$
Deslocamento	ld R4, 100(R1)	$R[4] \leftarrow MEM[100+R[1]]$
Indexado	dadd R3, (R1+R2)	$R[3] \leftarrow R[3]+M[R[1]+R[2]]$
Indireto Mem.	dadd R1, @(R3)	$R[1] \leftarrow R[1]+M[M[R[3]]]$

- Na sintaxe na coluna em linguagem de montagem os parenteses ( ) indicam acesso à memória.
- Na sintaxe RTL à direita, [ ] denota acesso a um elemento de um vetor, seja Registrador ou Memória.

# Modos de Endereçamento



# Modos de Endereçamento



# Tipos de Instrução

- **Aritméticas e Lógicas:** Aritmética inteira e operações lógicas, cadeias de caracteres.
- **Transferência de Dados:** Instruções de Load/Store.
- **Transferência de Controle:** Desvios, chamadas e retorno de procedimentos.
- **Sistema:** Chamadas ao sistema operacional, gerenciamento de memória virtual, HALT.
- **Ponto Flutuante:** operações de ponto flutuante.
- **Gráficas:** operações em pixel, compressão e descompressão.

# Arquitetura MIPS64

- **Instruções de 32 bits.**
- **32 GPRs de 64 bits, R0-R31.**
  - R0 é apenas a constante 0.
- **32 FPRs 64 bits, F0-F31**
  - Permitem operandos de 32-bits ( $\frac{1}{2}$  não utilizado).
  - Extensões “SIMD” operam com dois operandos em 1 FPR.
- **Poucos registradores especiais:**
  - Floating-point status register
- **Load/store de inteiros de 8, 16, 32 e 64-bits**
  - Todos tem o sinal estendido para caber no registrador de 64 bits.
  - Também operam os registradores de ponto flutuante (32 ou 64 bits)

# Modos de Endereçamento MIPS

- Registrador (apenas operações arit./lógicas)
- Imediato (apenas arit./lógica) & Deslocamento (apenas load/stores)
  - Imediato de 16-bits / deslocamento
  - Registrador indireto: usa o valor 0 como deslocamento
  - Direto (absoluto): usa R0 com base para o deslocamento
  - Memória endereçada a byte, endereços de 64 bits.
- Ordenação big-endian/little-endian escolhida por SW.
- Alinhamento dos dados é necessário.

# Modelo de Programação

- **A seguir são mostrados os conjuntos de registradores inteiros e de ponto flutuante vistos pelo programador.**

Figure 2-6 CPU Registers

63	0
r0 (hardwired to zero)	
r1	
r2	
r3	
r4	
r5	
r6	
r7	
r8	
r9	
r10	
r11	
r12	
r13	
r14	
r15	
r16	
r17	
r18	
r19	
r20	
r21	
r22	
r23	
r24	
r25	
r26	
r27	
r28	
r29	
r30	
r31	

General Purpose Registers

63	0
HI	
LO	

63	0
PC	

Special Purpose Registers

63

0

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19
R20
R21
R22
R23
R24
R25
R26
R27
R28
R29
R30
R31

General Purpose Registers

31

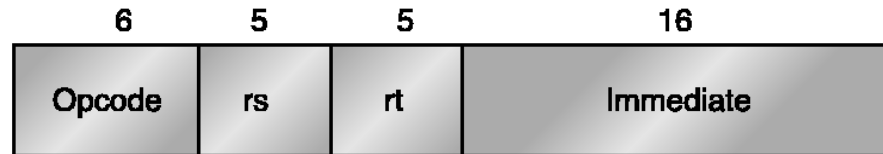
0

FR
FCCR
FEXR
FENR
FCSR

Special Purpose Registers

# Formato das Instruções MIPS

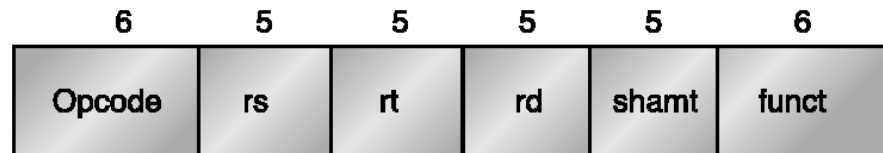
I-type instruction



Encodes: Loads and stores of bytes, half words, words, double words. All immediates ( $rt \leftarrow rs \text{ op immediate}$ )

Conditional branch instructions (rs is register, rd unused)  
Jump register, jump and link register  
(rd = 0, rs = destination, immediate = 0)

R-type instruction



Register-register ALU operations:  $rd \leftarrow rs \text{ funct } rt$   
Function encodes the data path operation: Add, Sub, ...  
Read/write special registers and moves

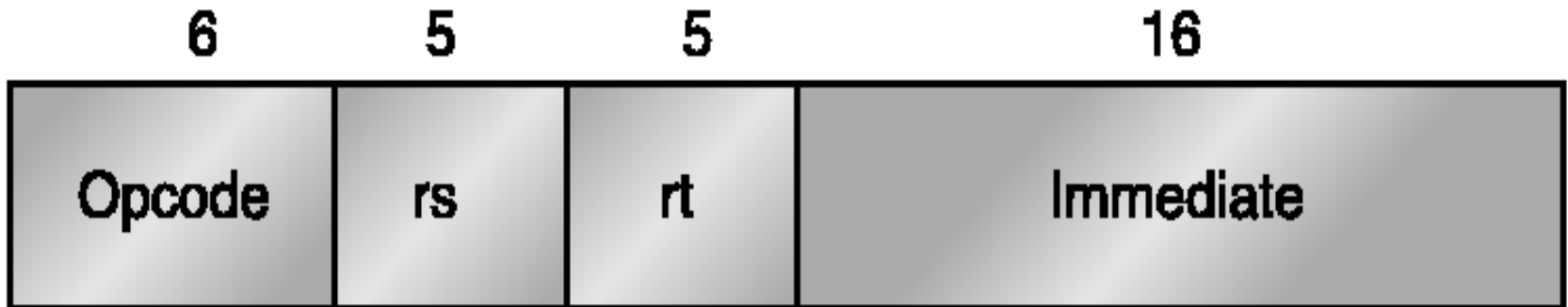
J-type instruction



Jump and jump and link  
Trap and return from exception

# Instruções Tipo I

I-type instruction



**Encodes: Loads and stores of bytes, half words, words, double words. All immediates ( $rt \leftarrow rs \text{ op immediate}$ )**

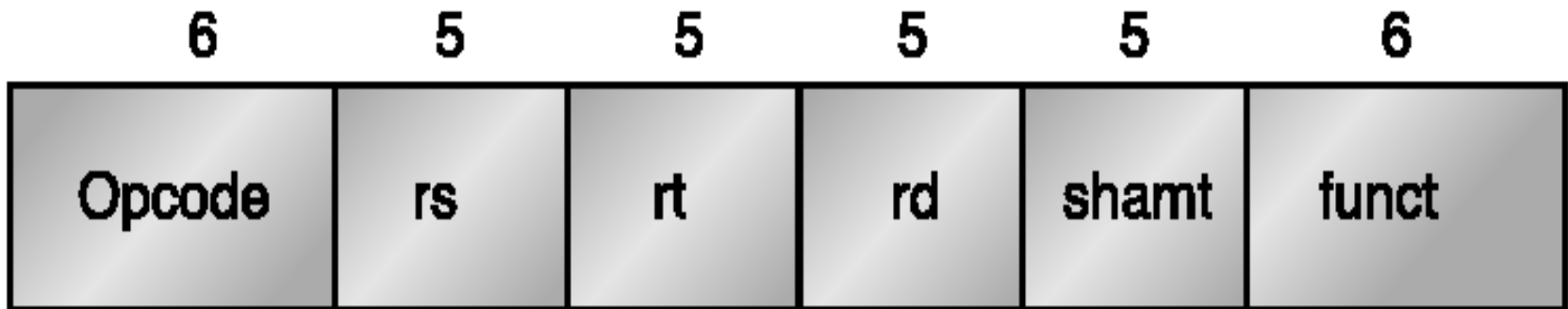
**Conditional branch instructions (rs is register, rd unused)**

**Jump register, jump and link register**

**(rd = 0, rs = destination, immediate = 0)**

# Instruções Tipo R

R-type instruction



Register-register ALU operations:  $rd \leftarrow rs \text{ funct } rt$

Function encodes the data path operation: Add, Sub, . . .

Read/write special registers and moves

# Instruções Tipo J

J-type instruction

6

26



Jump and jump and link

Trap and return from exception

# Transferência de Dados

Mnemonic	Instruction	Defined in MIPS ISA
LB	Load Byte	MIPS32
LBU	Load Byte Unsigned	MIPS32
LD	Load Doubleword	MIPS64
LH	Load Halfword	MIPS32
LHU	Load Halfword Unsigned	MIPS32
LW	Load Word	MIPS32
LWU	Load Word Unsigned	MIPS64
SB	Store Byte	MIPS32
SD	Store Doubleword	MIPS64
SH	Store Halfword	MIPS32
SW	Store Word	MIPS32

# Transferência de Dados

<b>ld R1, A (R3)</b>	<b>→</b>	<b>R1 = MEM (R3 + A)</b>
<b>ld R2, 0 (R4)</b>	<b>→</b>	<b>R2 = MEM (R4)</b>
<b>ld R5, A (R0)</b>	<b>→</b>	<b>R5 = MEM (A) ou R5 = A</b>
<b>sd R16, B (R3)</b>	<b>→</b>	<b>MEM(R3 +B) = R16</b>
<b>sd R12, 0 (R4)</b>	<b>→</b>	<b>MEM (R4) = R12</b>
<b>sd R15, A (R0)</b>	<b>→</b>	<b>MEM (A) = R15 ou A = R15</b>

**Obs: Todas as operações acima transferem 64 bits.**

# Aritméticas/Lógicas c/ Imediato

Mnemonic	Instruction	Defined in MIPS ISA
ADDI	Add Immediate Word	MIPS32
ADDIU <sup>1</sup>	Add Immediate Unsigned Word	MIPS32
ANDI	And Immediate	MIPS32
DADDI	Doubleword Add Immediate	MIPS64
DADDIU <sup>1</sup>	Doubleword Add Immediate Unsigned	MIPS64
LUI	Load Upper Immediate	MIPS32
ORI	Or Immediate	MIPS32
SLTI	Set on Less Than Immediate	MIPS32
SLTIU	Set on Less Than Immediate Unsigned	MIPS32
XORI	Exclusive Or Immediate	MIPS32

# Aritméticas/Lógicas c/ Imediato

**addi R2, R3, #5      → R2 = R3 + 5 (32 bits)**

**dsubi R6, R8, #10    → R6 = R8 – 10 (64 bits)**

**andi R6, R9, 0xFFFF → R6 = R9 AND 0xFFFF (32 bits)**

**lui R8, R7, 0xAAAA → R8 = (R7 AND 0xAAAA) << 16  
(32 bits)**

**slti R1, R2, #3      → Se R2 < 3 então R1 = 1**

**Senão R1 = 0**

**xori R1, R5, #6      → R1 = R5 xor 6**

# Aritméticas/Lógicas

Mnemonic	Instruction	Defined in MIPS ISA
ADD	Add Word	MIPS32
ADDU <sup>1</sup>	Add Unsigned Word	MIPS32
AND	And	MIPS32
DADD	Doubleword Add	MIPS64
DADDU <sup>1</sup>	Doubleword Add Unsigned	MIPS64
DSUB	Doubleword Subtract	MIPS64
DSUBU <sup>1</sup>	Doubleword Subtract Unsigned	MIPS64
NOR	Nor	MIPS32
OR	Or	MIPS32
SLT	Set on Less Than	MIPS32
SLTU	Set on Less Than Unsigned	MIPS32
SUB	Subtract Word	MIPS32
SUBU <sup>1</sup>	Subtract Unsigned Word	MIPS32
XOR	Exclusive Or	MIPS32

1. The term "unsigned" in the instruction name is a misnomer; this operation is 32-bit modulo arithmetic that does not trap on overflow.



# Deslocamento

<b>Mnemonic</b>	<b>Instruction</b>	<b>Defined in MIPS ISA</b>
DROR	Doubleword Rotate Right	MIPS64 Release 2
DROR32	Doubleword Rotate Right Plus 32	MIPS64 Release 2
DRORV	Doubleword Rotate Right Variable	MIPS64 Release 2
DSLL	Doubleword Shift Left Logical	MIPS64
DSLL32	Doubleword Shift Left Logical + 32	MIPS64
DSLLV	Doubleword Shift Left Logical Variable	MIPS64
DSRA	Doubleword Shift Right Arithmetic	MIPS64
DSRA32	Doubleword Shift Right Arithmetic + 32	MIPS64
DSRAV	Doubleword Shift Right Arithmetic Variable	MIPS64
DSRL	Doubleword Shift Right Logical	MIPS64
DSRL32	Doubleword Shift Right Logical + 32	MIPS64

# Deslocamento

DSRLV	Doubleword Shift Right Logical Variable	MIPS64
ROR	Rotate Word Right	MIPS32 Release 2
RORV	Rotate Word Right Variable	MIPS32 Release 2
SLL	Shift Word Left Logical	MIPS32
SLLV	Shift Word Left Logical Variable	MIPS32
SRA	Shift Word Right Arithmetic	MIPS32
SRAV	Shift Word Right Arithmetic Variable	MIPS32
SRL	Shift Word Right Logical	MIPS32
SRLV	Shift Word Right Logical Variable	MIPS32

# Deslocamento

- dsllv R1, R2, R3 → R1 = R2 << R3**  
(64 bits sem sinal)
- srl R10, R11, R12 → R10 = R11 >> R3**  
(32 bits sem sinal)
- dsra R5, R7 , #20 → R5 = R7 >> 20**  
( 64 bits com sinal)
- sra R6, R7, R8 → R6 = R7 >> R8**  
( 32 bits com sinal)
- dsra32 R5, R7, #20 → R5 = R7 >> (32 + 20)**  
( 64 bits com sinal)

# Multiplicação / Divisão

Mnemonic	Instruction	Defined in MIPS ISA
DDIV	Doubleword Divide	MIPS64
DDIVU	Doubleword Divide Unsigned	MIPS64
DIV	Divide Word	MIPS32
DIVU	Divide Unsigned Word	MIPS32
DMULT	Doubleword Multiply	MIPS64
DMULTU	Doubleword Multiply Unsigned	MIPS64
MADD	Multiply and Add Word	MIPS32
MADDU	Multiply and Add Word Unsigned	MIPS32
MFHI	Move From HI	MIPS32
MFLO	Move From LO	MIPS32

# Multiplicação / Divisão

MSUB	Multiply and Subtract Word	MIPS32
MSUBU	Multiply and Subtract Word Unsigned	MIPS32
MTHI	Move To HI	MIPS32
MTO	Move To LO	MIPS32
MUL	Multiply Word to Register	MIPS32
MULT	Multiply Word	MIPS32
MULTU	Multiply Unsigned Word	MIPS32

# Multiplicação / Divisão

<b>ddiv</b>	<b>R4, R6</b>	<b>→</b>	<b>LO, HI = R4 / R6 (64 bits)</b>
<b>div</b>	<b>R6, R7</b>	<b>→</b>	<b>LO, HI = R6 / R7 (32 bits)</b>
<b>dmult</b>	<b>R13, R15</b>	<b>→</b>	<b>LO, HI = R13 * R15 (64 bits)</b>
<b>mfhi</b>	<b>R5</b>	<b>→</b>	<b>R5 = HI</b>
<b>mtlo</b>	<b>R7</b>	<b>→</b>	<b>LO = R7</b>
<b>mult</b>	<b>R8, R9</b>	<b>→</b>	<b>LO, HI = R8 * R9 ( 32 bits)</b>

# Desvio Incondicional

<b>Mnemonic</b>	<b>Instruction</b>	<b>Location to Which Jump Is Made</b>	<b>Defined in MIPS ISA</b>
J	Jump	256 Megabyte Region	MIPS32
JAL	Jump and Link	256 Megabyte Region	MIPS32
JALR	Jump and Link Register	Absolute Address	MIPS32
JALR.HB	Jump and Link Register with Hazard Barrier	Absolute Address	MIPS32 Release 2
JALX	Jump and Link Exchange	Absolute Address	MIPS16e
JR	Jump Register	Absolute Address	MIPS32
JR.HB	Jump Register with Hazard Barrier	Absolute Address	MIPS32 Release 2

# Desvio Incondicional

**j LABEL**

**→ PC = LABEL**

**jal ROTINA**

**→ R31 = PC+4; PC = ROTINA**

**( chamada de procedimento)**

**jalr R20**

**→ R31 = PC+4; PC = R20**

**jr R31**

**→ PC = R31**

**(retorno de procedimento)**

# Desvio Condicional

Table 4-14 PC-Relative Conditional Branch Instructions Comparing Two Registers

Mnemonic	Instruction	Defined in MIPS ISA
BEQ	Branch on Equal	MIPS32
BNE	Branch on Not Equal	MIPS32

Table 4-15 PC-Relative Conditional Branch Instructions Comparing With Zero

Mnemonic	Instruction	Defined in MIPS ISA
BGEZ	Branch on Greater Than or Equal to Zero	MIPS32
BGEZAL	Branch on Greater Than or Equal to Zero and Link	MIPS32
BGTZ	Branch on Greater Than Zero	MIPS32
BLEZ	Branch on Less Than or Equal to Zero	MIPS32
BLTZ	Branch on Less Than Zero	MIPS32
BLTZAL	Branch on Less Than Zero and Link	MIPS32

# Desvio Condicional

**beq R8, R9, LABEL → se  $R8 = R9$  então  $PC = LABEL$**

**senão  $PC = PC + 4$**

**bgez R10, LABEL → se  $R10 \geq 0$  então  $PC = LABEL$**

**senão  $PC = PC + 4$**

# Sistema

**Table 4-18 System Call and Breakpoint Instructions**

<b>Mnemonic</b>	<b>Instruction</b>	<b>Defined in MIPS ISA</b>
BREAK	Breakpoint	MIPS32
SYSCALL	System Call	MIPS32

**Table 4-19 Trap-on-Condition Instructions Comparing Two Registers**

<b>Mnemonic</b>	<b>Instruction</b>	<b>Defined in MIPS ISA</b>
TEQ	Trap if Equal	MIPS32
TGE	Trap if Greater Than or Equal	MIPS32
TGEU	Trap if Greater Than or Equal Unsigned	MIPS32
TLT	Trap if Less Than	MIPS32
TLTU	Trap if Less Than Unsigned	MIPS32H
TNE	Trap if Not Equal	MIPS32

# Sistema

**break**

**syscall #5**

**teq R12, R13, #10**

**teqi R9, #5, #4**

**tge R9, R11, #5**

# Ponto Flutuante

Mnemonic	Instruction	Defined in MIPS ISA
<code>ABS.fmt</code>	Floating Point Absolute Value	MIPS32
<code>ABS.fmt (PS)</code>	Floating Point Absolute Value (Paired Single)	MIPS64 MIPS32 Release 2
<code>ADD.fmt</code>	Floating Point Add	MIPS32
<code>ADD.fmt (PS)</code>	Floating Point Add (Paired Single)	MIPS64 MIPS32 Release 2
<code>C.cond.fmt</code>	Floating Point Compare	MIPS32
<code>C.cond.fmt (PS)</code>	Floating Point Compare (Paired Single)	MIPS64 MIPS32 Release 2
<code>DIV.fmt</code>	Floating Point Divide	MIPS32
<code>MUL.fmt</code>	Floating Point Multiply	MIPS32
<code>MUL.fmt (PS)</code>	Floating Point Multiply (Paired Single)	MIPS64 MIPS32 Release 2
<code>NEG.fmt</code>	Floating Point Negate	MIPS32
<code>NEG.fmt (PS)</code>	Floating Point Negate (Paired Single)	MIPS64 MIPS32 Release 2
<code>SQRT.fmt</code>	Floating Point Square Root	MIPS32
<code>SUB.fmt</code>	Floating Point Subtract	MIPS32
<code>SUB.fmt (PS)</code>	Floating Point Subtract (Paired Single)	MIPS64 MIPS32 Release 2

# Ponto Flutuante

<b>add.s f0, f1, f2</b>	<b>→</b>	<b>f0 = f1 + f2 (32 bits)</b>
<b>add.d f0, f1, f2</b>	<b>→</b>	<b>f0 = f1 + f2 (64 bits)</b>
<b>mul.d f3, f4, f5</b>	<b>→</b>	<b>f3 = f4 * f5 ( 64 bits)</b>
<b>div.s f9, f11, f12</b>	<b>→</b>	<b>f9 = f11 / f12 (32 bits)</b>

## + MIPS

- Figuras 2.27-2.31 no livro texto, apêndice C online @ [www.mkp.com](http://www.mkp.com)
- Patterson & Hennessy, *Computer Organization & Design: The Hardware/Software Interface*, apêndice A-10 tem uma descrição da arquitetura MIPS.