



# Arquitectura dos Computadores

## Aula 7 – Representação de Instruções

---

Professor: Dr. Christophe Soares

# Programa Guardado em Memória

**Computadores têm por base 2 ideias:**

- 1) instruções são representadas como números**
- 2) programas inteiros podem ser guardados em memória para leitura e escrita como os dados**

**Simplifica o SW/HW dos sistemas de computação**

**Tecnologia de memória usada para os dados também é usada para os programas**

Consequência (1/2): “tudo tem um endereço”

**Tudo tem um endereço de memória:**

Instruções

Dados

**Os saltos (condicionais e incondicionais) usam estes endereços**

**“*Program Counter*” (PC)** → guarda o endereço da instrução a ser executada (apontador para a memória)

Consequência (2/2): “compatibilidade binária”

## Programas distribuídos em binário

- executam um conjunto de instruções específico
- versões diferentes para **ARM, MIPS, M1, INTEL , etc...**

## Conjunto de instruções evolui

Escolha da Intel 8086 (1981) do IBM PC, é a principal razão para que os PCs ainda usem hoje o conjunto de instruções 80x86

# Instruções são Números (1/2)

Dados têm 32 bits

Registo têm 32 bits

*lw* e o *sw* acedem à memória com 32 bits

Como representar as instruções?

Computador entende “0” e “1”, portanto “add \$t0,\$0,\$0” não tem significado!

MIPS pretende simplicidade: os dados estão em “words”, logo as instruções serão “words”

## Instruções são Números (2/2)

**Uma instrução tem 32 bits (word), e será dividida em “campos”**

**Um “campo” indica uma característica da instrução**

**Em MIPS, temos 3 formatos para as instruções:**

- i. Formato R de “*register*”
- ii. Formato I de “*immediate*”
- iii. Formato J de “*jump*”

# Tipo das instruções

## Formato I

usado para instruções com constantes, *lw* e *sw* (deslocamento é uma constante), e saltos condicionais (*beq* e *bne*)

## Formato J

usado para o *j* e o *jal*

## Formato R

usado pelas restantes instruções



FORMATO R

## Instruções com o formato R (1/5)

**“campos” definido:  $6 + 5 + 5 + 5 + 5 + 6 = 32$  (bits)**



Designação dos “campos”:



**Importante:** cada campo é visto como um inteiro sem sinal de 5 ou 6 bits

**Consequência:** 5-bit representa um número de 0-31, e 6-bit representa um número de 0-63

Importan  
sinal de 5

Consequê  
6-bit repr

opcode

Inst

“campos

6

Designa

MIPS Reference Data Card (“Green Card”)

2. Fold bottom side (columns 3 and 4) together

## MIPS Reference Data

### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R [rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2) 25 <sub>hex</sub>
Load Linked	ll	I R[rt]=M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I R[rt]=(imm, 16'b0)	f <sub>hex</sub>
Load Word	lw	I R[rt]=M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor	R R[rd] = ~ (R[rs] & R[rt])	0 / 27 <sub>hex</sub>
Or	or	R R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>
Set Less Than Imm.	siti	I R[rt] = (R[rs] < SignExtImm)? 1 : 0 (2)	a <sub>hex</sub>
Set Less Than Imm. Unsigned	sltiu	I R[rt] = (R[rs] < SignExtImm)? 1 : 0 (2,6)	b <sub>hex</sub>
Set Less Than Unsigned	sltu	R R[rd] = (R[rs] < R[rt]) ? 1 : 0 (6) 0 / 2b <sub>hex</sub>	
Shift Left Logical	sll	R R[rd] = R[rt] << shamt	0 / 00 <sub>hex</sub>
Shift Right Logical	srl	R R[rd] = R[rt] >> shamt	0 / 02 <sub>hex</sub>
Store Byte	sb	I M[R[rs]+SignExtImm](7:0)=R[rt](7:0)	(2) 28 <sub>hex</sub>
Store Conditional	sc	I M[R[rs]+SignExtImm]=R[rt]; R[rt]=(atomic)? 1 : 0 (2,7)	38 <sub>hex</sub>
Store Halfword	sh	I M[R[rs]+SignExtImm](15:0)=R[rt](15:0)	(2) 29 <sub>hex</sub>
Store Word	sw	I M[R[rs]+SignExtImm]=R[rt]	(2) 2b <sub>hex</sub>
Subtract	sub	R R[rd] = R[rs] - R[rt]	(1) 0 / 22 <sub>hex</sub>
Subtract Unsigned	subu	R R[rd] = R[rs] - R[rt]	0 / 23 <sub>hex</sub>

### BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct	0
	31	26 25	21 20	16 15	11 10	6 5	0
I	opcode	rs	rt	immediate			0
	31	26 25	21 20	16 15			0
J	opcode	address					0
	31	26 25					0

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 4th ed.



### ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FMT / FT / FUNCT (Hex)
Branch On FP True	bclt	FI if(FPcond)PC=PC+4+BranchAddr	(4) 11/8/-/-
Branch On FP False	bclf	FI if(!FPcond)PC=PC+4+BranchAddr	11/8/0/-/-
Divide	div	R Lo=R[rs]/R[rt]; Hi=R[rs]%-R[rt]	0/-/-/-1a
Divide Unsigned	divu	R Lo=R[rs]/R[rt]; Hi=R[rs]%-R[rt]	(6) 0/-/-/-1b
FP Add Single	add.s	FR F[fd]=F[fs]+F[ft]	11/10/-/-0
FP Add	add.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}	11/11/-/-0
FP Compare Single	c.x.s*	FR FPcond = (F[fs] op F[ft]) ? 1 : 0	11/10/-/-y
FP Compare	c.x.d*	FR FPcond = ({F[fs],F[fs+1]} op {F[ft],F[ft+1]}) ? 1 : 0	11/11/-/-y
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)			
FP Divide Single	div.s	FR F[fd]=F[fs] / F[ft]	11/10/-/-3
FP Divide	div.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}	11/11/-/-3
FP Multiply Single	mul.s	FR F[fd]=F[fs] * F[ft]	11/10/-/-2
FP Multiply	mul.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}	11/11/-/-2
FP Subtract Single	sub.s	FR F[fd]=F[fs] - F[ft]	11/10/-/-1
FP Subtract	sub.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}	11/11/-/-1
Load FP Single	lwcl	I F[rt]=M[R[rs]+SignExtImm]	(2) 31/-/-/-
Load FP	ldcl	I F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2) 35/-/-/-
Move From Hi	mfhi	R R[rd] = Hi	0 / -/-/10
Move From Lo	mflo	R R[rd] = Lo	0 / -/-/12
Move From Control	mfco	R R[rd] = CR[rs]	10 / 0/-/0
Multiply	mult	R {Hi,Lo} = R[rs] * R[rt]	0 / -/-/18
Multiply Unsigned	multu	R {Hi,Lo} = R[rs] * R[rt]	(6) 0 / -/-/19
Shift Right Arith.	sra	R R[rd] = R[rt] >> shamt	0 / -/-/3
Store FP Single	swcl	I M[R[rs]+SignExtImm] = F[rt]	(2) 39/-/-/-
Store FP	sdcl	I M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2) 3d/-/-/-

### FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fint	ft	fs	fd	funct	0
	31	26 25	21 20	16 15	11 10	6 5	0
FI	opcode	fint	ft	immediate			0
	31	26 25	21 20	16 15			0

### PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]≤R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]≥R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

### REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

32 (bits)

6

funct

o sem

0-31, e

Inst

“campos

6

Designa

opcode

Importan  
sinal de 5

Consequê  
6-bit repr

MIPS Reference Data Card (“Green Card”)

#### BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct	0
31	26 25	21 20	16 15	11 10	6 5		0
I	opcode	rs	rt	immediate			0
31	26 25	21 20	16 15				0
J	opcode	address					0
31	26 25						0

Copyright 2009 by Elsevier, Inc. All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 4th ed.

## MIPS Reference Data

### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)				
Add	add	R[Rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>				
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>				
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>				
Add Unsigned	addu	R R[Rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>				
And	and	R R[Rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>				
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>				
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>				
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>				
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>				
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>				
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>				
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2) 24 <sub>hex</sub>				
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2) 25 <sub>hex</sub>				
Load Linked	ll	I R[rt]=M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>				
Load Upper Imm.	lui	I R[rt]=(imm, 16'b0)	f <sub>hex</sub>				
Load Word	lw	I R[rt]=M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>				
Nor	nor	R R[Rd] = ~ (R[rs] & R[rt])	0 / 27 <sub>hex</sub>				
Or	or	R R[Rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>				
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>				
Set Less Than	slt	R R[Rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>				
Set Less Than Imm.	siti	I R[rt] = (R[rs] < SignExtImm)? 1 : 0 (2)	a <sub>hex</sub>				
Set Less Than Imm. Unsigned	sltiu	I R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2,6) b <sub>hex</sub>				
Set Less Than Unsigned	sltu	R R[Rd] = (R[rs] < R[rt]) ? 1 : 0	(6) 0 / 2b <sub>hex</sub>				
Shift Left Logical	sll	R R[Rd] = R[rt] << shamt	0 / 00 <sub>hex</sub>				
Shift Right Logical	srl	R R[Rd] = R[rt] >> shamt	0 / 02 <sub>hex</sub>				
Store Byte	sb	I M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2) 28 <sub>hex</sub>				
Store Conditional	sc	I M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic)? 1 : 0	(2,7) 38 <sub>hex</sub>				
Store Halfword	sh	I M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2) 29 <sub>hex</sub>				
Store Word	sw	I M[R[rs]+SignExtImm] = R[rt]	(2) 2b <sub>hex</sub>				
Subtract	sub	R R[Rd] = R[rs] - R[rt]	(1) 0 / 22 <sub>hex</sub>				
Subtract Unsigned	subu	R R[Rd] = R[rs] - R[rt]	0 / 23 <sub>hex</sub>				
(1) May cause overflow exception							
(2) SignExtImm = { 16{immediate[15]}, immediate }							
(3) ZeroExtImm = { 16{'b'0}, immediate }							
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }							
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }							
(6) Operands considered unsigned numbers (vs. 2's comp.)							
(7) Atomic test&set pair: R[rt]=1 if pair atomic, 0 if not atomic							



### ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FMT / FT / FUNCT (Hex)
Branch On FP True	bclt	FI if(FPcond)PC=PC+4+BranchAddr	(4) 11/8/-/-
Branch On FP False	bclf	FI if(!FPcond)PC=PC+4+BranchAddr	(4) 11/8/0/-/-
Divide	div	R Lo=R[rs]/R[rt]; Hi=R[rs]%-R[rt]	0/-/-/-1a
Divide Unsigned	divu	R Lo=R[rs]/R[rt]; Hi=R[rs]%-R[rt]	(6) 0/-/-/-1b
FP Add Single	adds	FR F[fd]=F[fs]+F[ft]	11/10/-/-0
FP Add	add.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}	11/11/-/-0
FP Compare Single	c.x.s*	FR FPcond = (F[fs] op F[ft]) ? 1 : 0	11/10/-/-y
FP Compare	c.x.d*	FR FPcond = ({F[fs],F[fs+1]} op {F[ft],F[ft+1]}) ? 1 : 0	11/11/-/-y
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)			
FP Divide Single	div.s	FR F[fd]=F[fs] / F[ft]	11/10/-/-3
FP Divide	div.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}	11/11/-/-3
FP Multiply Single	mul.s	FR F[fd]=F[fs] * F[ft]	11/10/-/-2
FP Multiply	mul.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}	11/11/-/-2
FP Subtract Single	sub.s	FR F[fd]=F[fs] - F[ft]	11/10/-/-1
FP Subtract	sub.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}	11/11/-/-1
Load FP Single	lwcl	I F[rt]=M[R[rs]+SignExtImm]	(2) 31/-/-/-
Load FP	ldcl	I F[rt]=M[R[rs]+SignExtImm]; F[r+1]=M[R[rs]+SignExtImm+4]	(2) 35/-/-/-
Move From Hi	mfhi	R R[Rd] = Hi	0 / -/-/10
Move From Lo	mflo	R R[Rd] = Lo	0 / -/-/12
Move From Control	mfc0	R R[Rd] = CR[rs]	10 / 0/-/0
Multiply	mult	R {Hi,Lo} = R[rs] * R[rt]	0 / -/-/18
Multiply Unsigned	multu	R {Hi,Lo} = R[rs] * R[rt]	(6) 0 / -/-/19
Shift Right Arith.	sra	R R[Rd] = R[rt] >> shamt	0 / -/-/3
Store FP Single	swcl	I M[R[rs]+SignExtImm] = F[rt]	(2) 39/-/-/-
Store FP	sdcl	I M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2) 3d/-/-/-

### FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fint	ft	fs	fd	funct	0
31	26 25	21 20	16 15	11 10	6 5		0
FI	opcode	fint	ft	immediate			0
31	26 25	21 20	16 15				0

### PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]≤R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]≥R[rt]) PC = Label
Load Immediate	li	R[Rd] = immediate
Move	move	R[Rd] = R[rs]

### REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	Yes
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

32 (bits)

6

funct

‘o sem

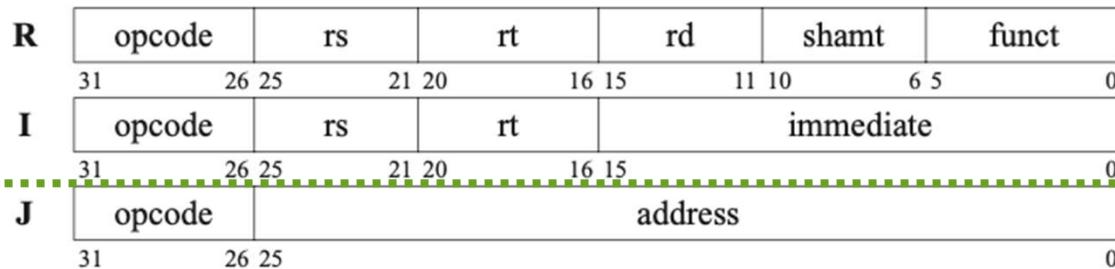
0-31, e

# MIPS Reference Data Card (“Green Card”)

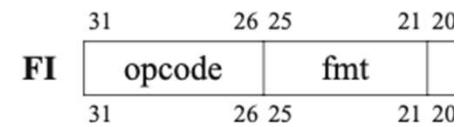
## 1. Pull along

	Unsigned	: 1 . 0	(L,U)
Set Less Than Unsigned	sltu	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6) 0 / 2b <sub>hex</sub>
Shift Left Logical	sll	R R[rd] = R[rt] << shamt	0 / 00 <sub>hex</sub>
Shift Right Logical	srl	R R[rd] = R[rt] >> shamt	0 / 02 <sub>hex</sub>
Store Byte	sb	I M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2) 28 <sub>hex</sub>
Store Conditional	sc	I M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7) 38 <sub>hex</sub>
Store Halfword	sh	I M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2) 29 <sub>hex</sub>
Store Word	sw	I M[R[rs]+SignExtImm] = R[rt]	(2) 2b <sub>hex</sub>
Subtract	sub	R R[rd] = R[rs] - R[rt]	(1) 0 / 22 <sub>hex</sub>
Subtract Unsigned	subu	R R[rd] = R[rs] - R[rt]	0 / 23 <sub>hex</sub>
		(1) May cause overflow exception	
		(2) SignExtImm = { 16{immediate[15]}, immediate }	
		(3) ZeroExtImm = { 16{1b'0}, immediate }	
		(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }	
		(5) JumpAddr = { PC+4[31:28], address, 2'b0 }	
		(6) Operands considered unsigned numbers (vs. 2's comp.)	
		(7) Atomic test&set pair: R[rt] = 1 if pair atomic, 0 if not atomic	

### BASIC INSTRUCTION FORMATS



Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 4th



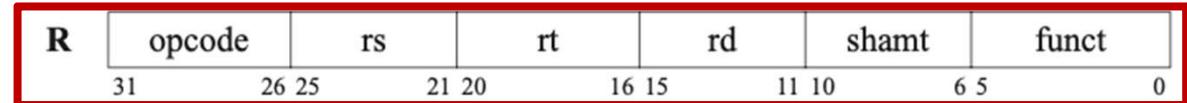
### PSEUDOINSTRUCTION SET

NAME	NAME
Branch Less Than	
Branch Greater Than	
Branch Less Than or Equal	
Branch Greater Than or Equal	
Load Immediate	
Move	

### REGISTER NAME, NUMBER, USE

NAME	NUMBER	
\$zero	0	The Constant 0
\$at	1	Assembly Address
\$v0-\$v1	2-3	Values and Examples
\$a0-\$a3	4-7	Arguments
\$t0-\$t7	8-15	Temporary
\$s0-\$s7	16-23	Saved Temporary
\$t8-\$t9	24-25	Temporary
\$k0-\$k1	26-27	Reserved
\$gp	28	Global Pointer
\$sp	29	Stack Pointer
\$fp	30	Frame Pointer
\$ra	31	Return Address

## Instruções com o formato R (2/5)



### O que representam os campos?

- opcode: quando é 0 (i.e., 000000) indica que o tipo de instrução é R
- funct: sabendo que é R, este número especifica qual será a instrução

## Instruções com o formato R (3/5)

R	opcode	rs	rt	rd	shamt	funct	
31	26 25	21 20	16 15	11 10	6 5	0	

### Restantes campos:

- rs (“Source Register”): especifica o registo contendo o primeiro operando
- rt (“Target Register”): indica o registo contendo o segundo operando
- rd (“Destination Register”): indica o registo que vai receber o resultado da computação

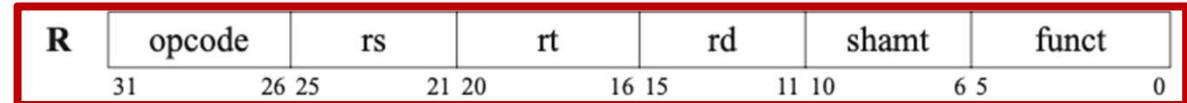
## Instruções com o formato R (4/5)

R	opcode	rs	rt	rd	shamt	funct	
31	26 25	21 20	16 15	11 10	6 5	0	

**Notas acerca dos campos *rs*, *rt*, *rd*:**

- tem exatamente 5 bits o que quer dizer que pode especificar um valor de 0-31
- identifica um dos 32 registos pelo número

# Instruções com o formato R (5/5)



## Campo final:

- shamt: contém o valor do deslocamento
  - desloca uma palavra de 32 bits
  - mais do que 31 é inútil portanto este campo tem apenas 5 bits (0-31)
  - está sempre a zero, exceto nas instruções de deslocamento

# Exemplo do Formato R (1/3)

**Instrução MIPS:**

**add \$8 , \$9 , \$10**

# Instruções add

MIPS Reference Data Card (“Green Card”)

1. Pull along perforation to separate card    2. Fold bottom side (columns 3 and 4) together

## MIPS Reference Data

### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT
Add	add	R R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	add	R R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I if(R[rs]≠R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I R[rt]=(24'b0,M[R[rs]+SignExtImm](7:0))	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I R[rt]=(16'b0,M[R[rs]+SignExtImm](15:0))	(2) 25 <sub>hex</sub>
Load Linked	ll	I R[rt]=M[R[rs]-SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I R[rt]={imm, 16'b0}	f <sub>hex</sub>
Load Word	lw	I R[rt]=M[R[rs]-SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor	R R[rd] = ~ (R[rs]   R[rt])	0 / 27 <sub>hex</sub>
Or	or	R R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori	I R[rt]=R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>
Set Less Than Imm.	slti	I R[rt] = (R[rs] < SignExtImm) ? 1 : 0 (2)	a <sub>hex</sub>
Set Less Than Imm. Unsigned	sltiu	I R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6) b <sub>hex</sub>
Set Less Than Unsigned	sltu	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6) 0 / 2b <sub>hex</sub>
Shift Left Logical	sll	R R[rd] = R[rt] << shampt	0 / 00 <sub>hex</sub>
Shift Right Logical	srl	R R[rd] = R[rt] >> shampt	0 / 02 <sub>hex</sub>
Store Byte	sb	I M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2) 28 <sub>hex</sub>
Store Conditional	sc	I M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7) 38 <sub>hex</sub>
Store Halfword	sh	I M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2) 29 <sub>hex</sub>
Store Word	sw	I M[R[rs]+SignExtImm] = R[rt]	(2) 2b <sub>hex</sub>
Subtract	sub	R R[rd] = R[rs] - R[rt]	(1) 0 / 22 <sub>hex</sub>
Subtract Unsigned	subu	R R[rd] = R[rs] - R[rt]	0 / 23 <sub>hex</sub>

- (1) May cause overflow exception
- (2) SignExtImm = { 16{immediate[15]}, immediate }
- (3) ZeroExtImm = { 16{1b'0}, immediate }
- (4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
- (5) JumpAddr = { PC+4[31:28], address, 2'b0 }
- (6) Operands considered unsigned numbers (vs. 2's comp.)
- (7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

### BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct	
	31 26 25	21 20	16 15	11 10	6 5	0	
I	opcode	rs	rt			immediate	
	31 26 25	21 20	16 15			0	
J	opcode				address		
	31 26 25					0	

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 4th ed.

### ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FMT / FT / FUNCT (Hex)
Branch On FP True	bclt	FI if(FPcond)PC=PC+4+BranchAddr	(4) 11/8/1--
Branch On FP False	bclf	FI if(!FPcond)PC=PC+4+BranchAddr	(4) 11/8/0--
Divide	div	R Lo=R[rs]/R[rt]; Hi=R[rs]%-R[rt]	0/-/-/1a
Divide Unsigned	divu	R Lo=R[rs]/R[rt]; Hi=R[rs]%-R[rt]	(6) 0/-/-/1b
FP Add Single	add.s	FR F[fd] = F[fs] + F[ft]	11/10/-/0
FP Add	add.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}	11/11/-/0
Double			
FP Compare Single	c.x.s*	FR FPcond = (F[fs] op F[ft]) ? 1 : 0	11/10/-/y
FP Compare	c.x.d*	FR FPcond = ((F[fs],F[fs+1]) op {F[ft],F[ft+1]}) ? 1 : 0	11/11/-/y
Double			
*	(x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)		
FP Divide Single	div.s	FR F[Rd] = F[fs] / F[ft]	11/10/-/3
FP Divide	div.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}	11/11/-/3
Double			
FP Multiply Single	mul.s	FR F[fd] = F[fs] * F[ft]	11/10/-/2
FP Multiply	mul.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} *	11/11/-/2
Double			
FP Subtract Single	sub.s	FR F[fd] = F[fs] - F[ft]	11/10/-/1
FP Subtract	sub.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}	11/11/-/1
Double			
Load FP Single	lwcl	I F[rt]=M[R[rs]+SignExtImm]	(2) 31/-/-/-
Load FP	ldcl	I F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2) 35/-/-/-
Double			
Move From Hi	mfhi	R R[rd]=Hi	0 / -/-/10
Move From Lo	mflo	R R[rd]=Lo	0 / -/-/12
Move From Control	mfc0	R R[rd]=CR[rs]	10 / 0/-/0
Multiply	mult	R {Hi,Lo} = R[rs] * R[rt]	0 / -/-/18
Multiply Unsigned	multu	R {Hi,Lo} = R[rs] * R[rt]	(6) 0 / -/-/19
Shift Right Arith.	sra	R R[rd] = R[rt] >> shampt	0 / -/-/3
Store FP Single	swcl	I M[R[rs]+SignExtImm] = F[rt]	(2) 39/-/-/-
Store FP	sdc1	I M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2) 3d/-/-/-
Double			

### FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fint	ft	fs	fd	funct	
	31 26 25	21 20	16 15	11 10	6 5	0	
FI	opcode	fint	ft	immediate			
	31 26 25	21 20	16 15				0

### PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

### REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

# Exercício

## Instruções add

**MIPS Reference Data Card ("Green Card")**

1. Pull along perforation to separate card  
2. Fold bottom side (columns 3 and 4) together

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT		NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FMT / FT / FUNCT																	
			(Hex)	(1) 0 / 20hex				(Hex)	(4) 11/8/1/-																
Add	add	R [rd] = R[rs] + R[rt]	8hex		Branch On FP True	bc1t	FI if(FPcond)PC=PC+4+BranchAddr	11/8/0/-																	
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 9hex		Branch On FP False	bc1f	FI if(!FPcond)PC=PC+4+BranchAddr	11/8/0/-																	
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	0 / 21hex		Divide	div	R Lo=R[rs]/R[rt]; Hi=R[rs] % R[rt]	0/-/-/1a																	
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 24hex		Divide Unsigned	divu	R Lo=R[rs]/R[rt]; Hi=R[rs] % R[rt]	0/-/-/1b																	
And	and	R [rd] = R[rs] & R[rt]	0 / 24hex		FP Add Single	add.s	FR F[fd] = F[fs] + F[ft]	11/10/-/0																	
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	3hex		FP Add	add.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}	11/11/-/0																	
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	4hex		FP Compare Single	c.x.s*	FR FPcond = (F[fs] op F[ft]) ? 1 : 0	11/10/-/y																	
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	5hex		FP Compare	c.x.d*	FR FPcond = ((F[fs],F[fs+1]) op {F[ft],F[ft+1]}) ? 1 : 0	11/11/-/y																	
Jump	j	J PC=JumpAddr	2hex		Double	*	(x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)																		
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	3hex		FP Divide Single	div.s	FR F[Rd] = F[fs] / F[ft]	11/10/-/3																	
Jump Register	jr	R PC=R[rs]	0 / 08hex		FP Divide	div.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}	11/11/-/3																	
Load Byte Unsigned	lbu	I R[rt]=(24'b0,M[R[rs]+SignExtImm](7:0))	24hex		Double	*	(F[fs],F[fs+1]) * (F[ft],F[ft+1])																		
Load Halfword Unsigned	lhu	I R[rt]=(16'b0,M[R[rs]+SignExtImm](15:0))	25hex		FP Multiply Single	mul.s	FR F[fd] = F[fs] * F[ft]	11/10/-/2																	
Load Linked	ll	I R[rt]=M[R[rs]+SignExtImm]	(2,7) 30hex		FP Multiply	mul.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} *	11/11/-/2																	
Load Upper Imm.	lui	I R[rt]={imm, 16'b0}	fhex		Double	{F[ft],F[ft+1]}																			
Load Word	lw	I R[rt]=M[R[rs]+SignExtImm]	23hex		FP Subtract Single	sub.s	FR F[fd] = F[fs] - F[ft]	11/10/-/1																	
Nor	nor	R R[rd] = ~ (R[rs]   R[rt])	0 / 27hex		FP Subtract	sub.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}	11/11/-/1																	
Or	or	R R[rd] = R[rs]   R[rt]	0 / 25hex		Load FP Single	lwcl	I F[rt]=M[R[rs]+SignExtImm]	31/-/-/-																	
Or Immediate	ori	I R[rt]=R[rs]   ZeroExtImm	3hex		Load FP	ldcl	I F[rt]=M[R[rs]+SignExtImm];	35/-/-/-																	
Set Less Than	slt	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a hex		Double	F[rt+1]=M[R[rs]+SignExtImm+4]																			
Set Less Than Imm.	slti	I R[rt] = (R[rs] < SignExtImm) ? 1 : 0 (2)	ahex		Move From Hi	mfhi	R R[rd] = Hi	0 / -/-/10																	
Set Less Than Imm. Unsigned	sltiu	I R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6) bhex		Move From Lo	mflo	R R[rd] = Lo	0 / -/-/12																	
Set Less Than Unsigned	situ	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6) 0 / 2bhex		Move From Control	mfc0	R R[rd] = CR[rs]	10 / 0/-/0																	
Shift Left Logical	sll	R R[rd] << shampt	0 / 00hex		Multiply	mult	R {Hi,Lo} = R[rt] * R[rt]	0 / -/-/18																	
Shift Right Logical	srl	R R[rd] >> shampt	0 / 02hex		Multiply Unsigned	mult	R {Hi,Lo} = R[rt] * R[rt]	(6) 0 / -/-/19																	
Store Byte	sb	I M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	28hex		Shift Right Arith.	sra	R R[rd] = R[rt]>> shampt	0 / -/-/3																	
Store Conditional	sc	I M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7) 38hex		Store FP Single	swcl	I M[R[rs]+SignExtImm] = F[rt]	(2) 39/-/-/-																	
Store Halfword	sh	I M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	29hex		Store FP	swcr	I M[R[rs]+SignExtImm] = F[rt];	(2) 3d/-/-/-																	
Store Word	sw	I M[R[rs]+SignExtImm] = R[rt]	2bhex		Double	M[R[rs]+SignExtImm+4] = F[rt+1]																			
Subtract	sub	R R[rd] = R[rs] - R[rt]	(1) 0 / 22hex																						
Subtract Unsigned	subu	R R[rd] = R[rs] - R[rt]	0 / 23hex																						
(1) May cause overflow exception																									
(2) SignExtImm = { 16{immediate[15]}, immediate }																									
(3) ZeroExtImm = { 16{1b'0}, immediate }																									
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }																									
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }																									
(6) Operands considered unsigned numbers (vs. 2's comp.)																									
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic																									
<b>BASIC INSTRUCTION FORMATS</b>																									
<table border="1"> <tr> <td>R</td> <td>opcode</td> <td>rs</td> <td>rt</td> <td>rd</td> <td>shamt</td> <td>funct</td> <td></td> </tr> <tr> <td>31</td> <td>26 25</td> <td>21 20</td> <td>16 15</td> <td>11 10</td> <td>6 5</td> <td>0</td> <td></td> </tr> </table>										R	opcode	rs	rt	rd	shamt	funct		31	26 25	21 20	16 15	11 10	6 5	0	
R	opcode	rs	rt	rd	shamt	funct																			
31	26 25	21 20	16 15	11 10	6 5	0																			
<table border="1"> <tr> <td>I</td> <td>opcode</td> <td>rs</td> <td>rt</td> <td colspan="4">immediate</td> </tr> <tr> <td>31</td> <td>26 25</td> <td>21 20</td> <td>16 15</td> <td colspan="4">0</td> </tr> </table>										I	opcode	rs	rt	immediate				31	26 25	21 20	16 15	0			
I	opcode	rs	rt	immediate																					
31	26 25	21 20	16 15	0																					
<table border="1"> <tr> <td>J</td> <td>opcode</td> <td colspan="6">address</td> </tr> <tr> <td>31</td> <td>26 25</td> <td colspan="6">0</td> </tr> </table>										J	opcode	address						31	26 25	0					
J	opcode	address																							
31	26 25	0																							
Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 4th ed.																									

### ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FMT / FT / FUNCT
Branch On FP True	bc1t	FI if(FPcond)PC=PC+4+BranchAddr	(4) 11/8/1/-
Branch On FP False	bc1f	FI if(!FPcond)PC=PC+4+BranchAddr	(4) 11/8/0/-
Divide	div	R Lo=R[rs]/R[rt]; Hi=R[rs] % R[rt]	0/-/-/1a
Divide Unsigned	divu	R Lo=R[rs]/R[rt]; Hi=R[rs] % R[rt]	(6) 0/-/-/1b
FP Add Single	add.s	FR F[fd] = F[fs] + F[ft]	11/10/-/0
FP Add	add.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}	11/11/-/0
Double	*		
FP Compare Single	c.x.s*	FR FPcond = (F[fs] op F[ft]) ? 1 : 0	11/10/-/y
FP Compare	c.x.d*	FR FPcond = ((F[fs],F[fs+1]) op {F[ft],F[ft+1]}) ? 1 : 0	11/11/-/y
Double	*		
FP Divide Single	div.s	FR F[Rd] = F[fs] / F[ft]	11/10/-/3
FP Divide	div.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}	11/11/-/3
Double	*		
FP Multiply Single	mul.s	FR F[fd] = F[fs] * F[ft]	11/10/-/2
FP Multiply	mul.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} *	11/11/-/2
Double	*		
FP Subtract Single	sub.s	FR F[fd] = F[fs] - F[ft]	11/10/-/1
FP Subtract	sub.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}	11/11/-/1
Double	*		
Load FP Single	lwcl	I F[rt]=M[R[rs]+SignExtImm]	(2) 31/-/-/-
Load FP	ldcl	I F[rt]=M[R[rs]+SignExtImm];	(2) 35/-/-/-
Double	*		
Move From Hi	mfhi	R R[rd] = Hi	0 / -/-/10
Move From Lo	mflo	R R[rd] = Lo	0 / -/-/12
Move From Control	mfc0	R R[rd] = CR[rs]	10 / 0/-/0
Multiply	mult	R {Hi,Lo} = R[rt] * R[rt]	0 / -/-/18
Multiply Unsigned	mult	R {Hi,Lo} = R[rt] * R[rt]	(6) 0 / -/-/19
Shift Right Arith.	sra	R R[rd] = R[rt]>> shampt	0 / -/-/3
Store FP Single	swcl	I M[R[rs]+SignExtImm] = F[rt]	(2) 39/-/-/-
Store FP	swcr	I M[R[rs]+SignExtImm] = F[rt];	(2) 3d/-/-/-
Double	*		

### FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fint	ft	fs	fd	funct	
31	26 25	21 20	16 15	11 10	6 5	0	

FI	opcode	fint	ft	immediate			
31	26 25	21 20	16 15				0

### PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

### REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

operation to separate card 2. Fold bottom side (columns 3 and 4) together

# MIPS Reference Data



## CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2) 25 <sub>hex</sub>
Load Linked	ll	I R[rt] = M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I R[rt] = {imm, 16'b0}	f <sub>hex</sub>
Load Word	lw	I R[rt] = M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor	R R[rd] = ~ (R[rs]   R[rt])	0 / 27 <sub>hex</sub>
Or	or	R R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>

NAME, MNEMONIC	FOR-MAT
Branch On FP True	bc1t FI if(
Branch On FP False	bc1f FI if(
Divide	div R Lc
Divide Unsigned	divu R Lc
FP Add Single	add.s FR F[
FP Add	add.d FR {F
Double	
FP Compare Single	c.x.s* FR FI
FP Compare Double	c.x.d* FR FI
FP Divide Single	div.s FR F[
FP Divide Double	div.d FR {F
FP Multiply Single	mul.s FR F[
FP Multiply Double	mul.d FR {F
FP Subtract Single	sub.s FR F[
FP Subtract Double	sub.d FR {F
Load FP Single	lwc1 I F[
Load FP Double	ldc1 I F[
Move From Hi	mfhi R R[
Move From Lo	mflo R R[
Move From Control	mfc0 R R[
Multiply	mult R {F
Multiply Unsigned	multu R {F
Shift Right Arith.	sra R R[
Store FP Single	swc1 I M
Store FP Double	sdc1 I M

\* (x is eq, lt, or le) (op is ==,

FP Divide Single

FP Divide Double

FP Multiply Single

FP Multiply Double

FP Subtract Single

FP Subtract Double

Load FP Single

Load FP Double

Move From Hi

Move From Lo

Move From Control

Multiply

Multiply Unsigned

Shift Right Arith.

Store FP Single

Store FP Double

Operation to separate card 2. Fold bottom side (columns 3 and 4) together

# MIPS Reference Data



## CORE INSTRUCTION SET

				FOR- MAT	OPCODE / FUNCT (Hex)
	NAME, MNEMONIC		OPERATION (in Verilog)		
Add	add	R	$R[rd] = R[rs] + R[rt]$		(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I	$R[rt] = R[rs] + \text{SignExtImm}$		(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I	$R[rt] = R[rs] + \text{SignExtImm}$		(2) 9 <sub>hex</sub>
Add Unsigned	addu	R	$R[rd] = R[rs] + R[rt]$		0 / 21 <sub>hex</sub>
And	and	R	$R[rd] = R[rs] \& R[rt]$		0 / 24 <sub>hex</sub>
And Immediate	andi	I	$R[rt] = R[rs] \& \text{ZeroExtImm}$		(3) c <sub>hex</sub>
Branch On Equal	beq	I	$\begin{aligned} &\text{if}(R[rs]==R[rt]) \\ &\text{PC}=\text{PC}+4+\text{BranchAddr} \end{aligned}$		(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I	$\begin{aligned} &\text{if}(R[rs]!=R[rt]) \\ &\text{PC}=\text{PC}+4+\text{BranchAddr} \end{aligned}$		(4) 5 <sub>hex</sub>
Jump	j	J	$\text{PC}=\text{JumpAddr}$		(5) 2 <sub>hex</sub>
Jump And Link	jal	J	$R[31]=\text{PC}+8; \text{PC}=\text{JumpAddr}$		(5) 3 <sub>hex</sub>
Jump Register	jr	R	$\text{PC}=R[rs]$		0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I	$R[rt]=\{24'b0, M[R[rs]] + \text{SignExtImm}\}(7:0)$		(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I	$R[rt]=\{16'b0, M[R[rs]] + \text{SignExtImm}\}(15:0)$		(2) 25 <sub>hex</sub>
Load Linked	ll	I	$R[rt] = M[R[rs]] + \text{SignExtImm}$		(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I	$R[rt] = \{\text{imm}, 16'b0\}$		f <sub>hex</sub>
Load Word	lw	I	$R[rt] = M[R[rs]] + \text{SignExtImm}$		(2) 23 <sub>hex</sub>
Nor	nor	R	$R[rd] = \sim(R[rs]   R[rt])$		0 / 27 <sub>hex</sub>
Or	or	R	$R[rd] = R[rs]   R[rt]$		0 / 25 <sub>hex</sub>
Or Immediate	ori	I	$R[rt] = R[rs]   \text{ZeroExtImm}$		(3) d <sub>hex</sub>
Set Less Than	slt	R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$		0 / 2a <sub>hex</sub>

NAME, MNEMONIC	FOR-MAT
Branch On FP True	bc1t FI if(
Branch On FP False	bc1f FI if(
Divide	div R Lc
Divide Unsigned	divu R Lc
FP Add Single	add.s FR F[
FP Add	add.d FR {F
Double	
FP Compare Single	c.x.s* FR FI
FP Compare Double	c.x.d* FR FI
FP Divide Single	div.s FR F[
FP Divide Double	div.d FR {F
FP Multiply Single	mul.s FR F[
FP Multiply Double	mul.d FR {F
FP Subtract Single	sub.s FR F[
FP Subtract Double	sub.d FR {F
Load FP Single	lwcl I F[
Load FP Double	ldcl I F[
Move From Hi	mfhi R R[
Move From Lo	mflo R R[
Move From Control	mfc0 R R[
Multiply	mult R {F
Multiply Unsigned	multu R {F
Shift Right Arith.	sra R R[
Store FP Single	swcl I M
Store FP Double	sdc1 I M

\* (x is eq, lt, or le) (op is ==,

FP Divide Single

FP Divide Double

FP Multiply Single

FP Multiply Double

FP Subtract Single

FP Subtract Double

Load FP Single

Load FP Double

Move From Hi

Move From Lo

Move From Control

Multiply

Multiply Unsigned

Shift Right Arith.

Store FP Single

Store FP Double

>> shamt	0 / 02 <sub>hex</sub>
ExtImm](7:0) = R[rt](7:0)	(2) 28 <sub>hex</sub>
ExtImm] = R[rt]; = (atomic) ? 1 : 0	(2,7) 38 <sub>hex</sub>
ExtImm](15:0) = R[rt](15:0)	(2) 29 <sub>hex</sub>
ExtImm] = R[rt]	(2) 2b <sub>hex</sub>
- R[rt]	(1) 0 / 22 <sub>hex</sub>
- R[rt]	0 / 23 <sub>hex</sub>
ception	
mediate[15]}, immediate }	
'0}, immediate }	
neditate[15]}, immediate, 2'b0 }	
31:28], address, 2'b0 }	
isigned numbers (vs. 2's comp.)	
[rt] = 1 if pair atomic, 0 if not atomic	

rd	shamt	funct	
5	11 10	6 5	0
immediate			
5	0		
address			0

## PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

## REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

ved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.

>> shamt	0 / 02 <sub>hex</sub>
ExtImm](7:0) = R[rt](7:0)	(2) 28 <sub>hex</sub>
ExtImm] = R[rt]; = (atomic) ? 1 : 0	(2,7) 38 <sub>hex</sub>
ExtImm](15:0) = R[rt](15:0)	(2) 29 <sub>hex</sub>
ExtImm] = R[rt]	(2) 2b <sub>hex</sub>
- R[rt]	(1) 0 / 22 <sub>hex</sub>
- R[rt]	0 / 23 <sub>hex</sub>
ception	
mediate[15]}, immediate }	
'0}, immediate }	
nEDIATE[15]}, immediate, 2'b0 }	
31:28], address, 2'b0 }	
isigned numbers (vs. 2's comp.)	
[rt] = 1 if pair atomic, 0 if not atomic	

rd	shamt	funct	
5	11 10	6 5	0
immediate			
5			0
address			
			0

## PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

## REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

ved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.

# Exemplo do Formato R (2/3)

Instrução MIPS:

**add \$8 , \$9 , \$10**

$$R[rd] = R[rs] + R[rt]$$

**opcode** = 0 (ver na tabela de instruções)

**funct** = 32 (ver na tabela de instruções  $20_{\text{hex}} - 2^5 = 32_{10}$ )

**rd** = 8 (destino - \$t0)

**rs** = 9 (primeiro *operando* - \$t1)

**rt** = 10 (segundo *operando* - \$t2)

**shamt** = 0 (não é um deslocamento)

Representação decimal dos campos:

<b>opcode</b>	<b>rs</b>	<b>rt</b>	<b>rd</b>	<b>shamt</b>	<b>funct</b>
0	9	10	8	0	32

# Exemplo do Formato R (3/3)

Instrução MIPS:

**add \$8 , \$9 , \$10**

$$R[rd] = R[rs] + R[rt]$$

Representação decimal dos campos:

0	9	10	8	0	32
---	---	----	---	---	----

Representação binária dos campos:

000000	01001	01010	01000	00000	100000
--------	-------	-------	-------	-------	--------

# Exemplo do Formato R (3/3)

Instrução MIPS:

**add \$8 , \$9 , \$10**

$$R[rd] = R[rs] + R[rt]$$

Representação decimal dos campos:

0	9	10	8	0	32
---	---	----	---	---	----

Representação binária dos campos:

000000	01001	01010	01000	00000	100000
--------	-------	-------	-------	-------	--------

hex

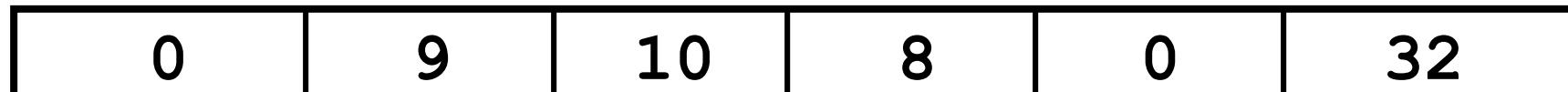
# Exemplo do Formato R (3/3)

Instrução MIPS:

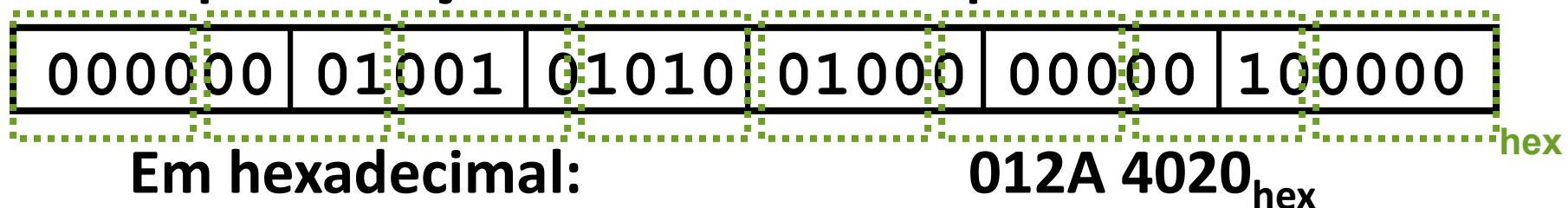
**add \$8 , \$9 , \$10**

$$R[rd] = R[rs] + R[rt]$$

Representação decimal dos campos:



Representação binária dos campos:



# Exemplo do Formato R (3/3)

Instrução MIPS:

**add \$8 , \$9 , \$10**

$$R[rd] = R[rs] + R[rt]$$

Representação decimal dos campos:

0	9	10	8	0	32
---	---	----	---	---	----

Representação binária dos campos:

000000	01001	01010	01000	00000	100000
--------	-------	-------	-------	-------	--------

hex

Em hexadecimal:

**012A 4020<sub>hex</sub>**

Em decimal:

**19,546,144<sub>10</sub>**

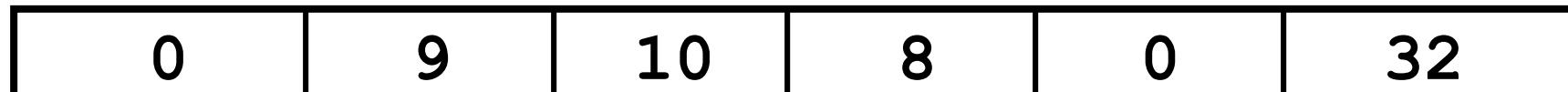
# Exemplo do Formato R (3/3)

Instrução MIPS:

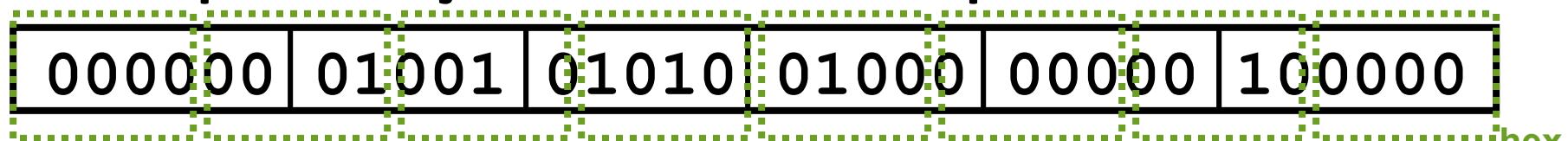
add \$8 , \$9 , \$10

$$R[rd] = R[rs] + R[rt]$$

Representação decimal dos campos:



Representação binária dos campos:



Em hexadecimal:

012A 4020<sub>hex</sub>

Em decimal:

19,546,144<sub>10</sub>

Chama-se Instrução em Linguagem Máquina

# Instruções

add

Repres.

0

Repres.

000000

Em he

Em de

Chama-s

OPCODES, BASE CONVERSION, ASCII SYMBOLS			③		
MIPS (1) MIPS (2)	funct	funct	Binary	Deci-mal	Hexa-ASCII
(31:26)	(5:0)	(5:0)		Char-mal	Char-acter
(1)	sll	add,f	00 0000	0	0 NUL
		sub,f	00 0001	1	1 SOH
j	srl	mul,f	00 0010	2	2 STX
jal	sra	div,f	00 0011	3	3 ETX
beq	silv	sqr,f	00 0100	4	4 EOT
bne		abs,f	00 0101	5	5 ENQ
blez	srly	mov,f	00 0110	6	6 ACK
bgtz	srav	neg,f	00 0111	7	7 BEL
addi	jr		00 1000	8	8 BS
addiu			00 1001	9	9 HT
slti	movz,f		00 1010	10	a LF
sltiu	movn,f		00 1011	11	b VT
andi	syscall	round,w,f	00 1100	12	c FF
ori		trunc,w,f	00 1101	13	d CR
xori		ceil,w,f	00 1110	14	e SO
lui	sync	floor,w,f	00 1111	15	f SI
mphi			01 0000	16	10 DLE
(2)	mtwi		01 0001	17	11 DC1
mflo	movz,f		01 0010	18	12 DC2
mtlo	movn,f		01 0011	19	13 DC3
			01 0100	20	14 DC4
			01 0101	21	15 NAK
			01 0110	22	16 SYN
			01 0111	23	17 ETB
			01 1000	24	18 CAN
			01 1001	25	19 EM
			01 1010	26	1a SUB
			01 1011	27	1b ESC
			01 1100	28	1c FS
			01 1101	29	1d GS
			01 1110	30	1e RS
			01 1111	31	1f US
lb	add	cvt.s,f	10 0000	32	20 Space
lh	addu	cvt.d,f	10 0001	33	21 !
lw	sub		10 0010	34	22 "
sw	subu		10 0011	35	23 #
lbu	and	cvt.w,f	10 0100	36	24 \$
lhu	or		10 0101	37	25 %
lw	xor		10 0110	38	26 &
sw	nor		10 0111	39	27 ,
sb			10 1000	40	28 (
sh			10 1001	41	29 )
swl	slt		10 1010	42	2a *
sw	sltu		10 1011	43	2b +
			10 1100	44	2c ,
			10 1101	45	2d -
			10 1110	46	2e :
			10 1111	47	2f /
swr					100 1111
cache					111 1111
ll	tge	c.f,f	11 0000	48	30 0
lwcl	tgeu	c.unf	11 0001	49	31 1
lwc2	tilt	c.eqaf	11 0010	50	32 2
pref	tltu	c.uegf	11 0011	51	33 3
	tee	c.oltf	11 0100	52	34 4
ldc1	c.ultf		11 0101	53	35 5
ldc2	tne	c.olef	11 0110	54	36 6
		c.ulnf	11 0111	55	37 7
sc	c.sff		11 1000	56	38 8
swcl	c.nglef		11 1001	57	39 9
swc2	c.seqf		11 1010	58	3a :
	c.nglf		11 1011	59	3b ;
	c.lt,f		11 1100	60	3c <
sdc1	c.ngef		11 1101	61	3d =
sdc2	c.lef		11 1110	62	3e >
	c.ngtf		11 1111	63	3f ?

(1) opcode(31:26) == 0

(2) opcode(31:26) == 17<sub>10</sub>(11<sub>hex</sub>); if fnt(25:21)==16<sub>10</sub>(10<sub>hex</sub>)f = s (single); if fnt(25:21)==17<sub>10</sub>(11<sub>hex</sub>)f = d (double)

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 4th ed.

## IEEE 754 FLOATING-POINT STANDARD

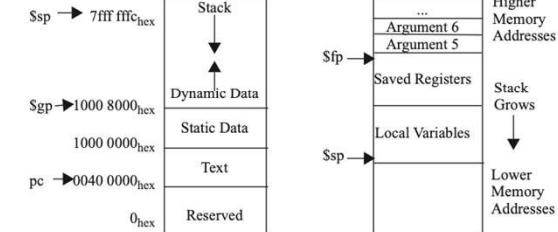
(-1)<sup>S</sup> × (1 + Fraction) × 2<sup>(Exponent - Bias)</sup>  
where Single Precision Bias = 127,  
Double Precision Bias = 1023.

## IEEE Single Precision and Double Precision Formats:

IEEE 754 Symbols		
Exponent	Fraction	Object
0	0	± 0
0	≠ 0	± Denorm
1 to MAX - 1	anything ± Fl. Pt. Num.	
MAX	0	±∞
MAX	≠ 0	NaN

S.P. MAX = 255, D.P. MAX = 2047

## MEMORY ALLOCATION



## DATA ALIGNMENT

Double Word							
Word				Word			
Halfword		Halfword		Halfword		Halfword	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0	1	2	3	4	5	6	7

Value of three least significant bits of byte address (Big Endian)

## EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS

B	D	Interrupt Mask	Exception Code
31	15	8	6
		Pending Interrupt	U M E L I E

BD = Branch Delay, UM = User Mode, EL = Exception Level, IE = Interrupt Enable

## EXCEPTION CODES

Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	System Call Exception	15	FPE	Floating Point Exception

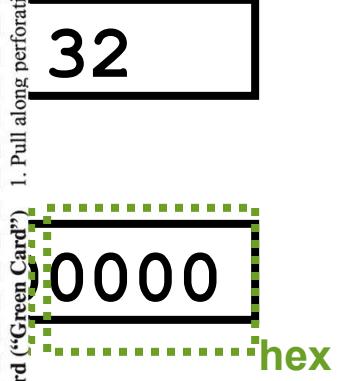
## SIZE PREFIXES (10<sup>x</sup> for Disk, Communication; 2<sup>x</sup> for Memory)

PRE-SIZE	FIX	PRE-SIZE	FIX	PRE-SIZE	FIX
10 <sup>3</sup> , 2 <sup>10</sup>	Kilo-	10 <sup>15</sup> , 2 <sup>50</sup>	Peta-	10 <sup>-3</sup>	milli-
10 <sup>6</sup> , 2 <sup>20</sup>	Mega-	10 <sup>18</sup> , 2 <sup>60</sup>	Exa-	10 <sup>-6</sup>	micro-
10 <sup>9</sup> , 2 <sup>30</sup>	Giga-	10 <sup>21</sup> , 2 <sup>70</sup>	Zetta-	10 <sup>-9</sup>	nano-
10 <sup>12</sup> , 2 <sup>40</sup>	Tera-	10 <sup>24</sup> , 2 <sup>80</sup>	Yotta-	10 <sup>-12</sup>	pico-

The symbol for each prefix is just its first letter, except μ is used for micro.



1. Pull along perforation to separate card  
2. Fold bottom side (columns 3 and 4) together



MIPS Reference Data Card ("Green Card") X

32

Exe  
Instruções  
add  
Representação  
0  
Representação  
000000  
Em hex  
Em decimal

OPCODES, BASE CONVERSION, ASCII SYMBOLS			(3)						
MIPS (1) MIPS (2) MIPS	opcode funct funct	(31:26) (5:0)	Binary	Deci-mal	Hexa-ASCII	Char-acter	Deci-mal	Hexa-ASCII	Char-acter
(1)	sll	add,f	00 0000	0	0	NUL	64	40	@
		sub,f	00 0001	1	1	SOH	65	41	A
j	srl	mul,f	00 0010	2	2	STX	66	42	B
jal	sra	div,f	00 0011	3	3	ETX	67	43	C
beq	silv	sqrt,f	00 0100	4	4	EOT	68	44	D
bne		abs,f	00 0101	5	5	ENQ	69	45	E
blez	srly	mov,f	00 0110	6	6	ACK	70	46	F
bgtz	sra	neg,f	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu			00 1001	9	9	HT	73	49	I
slti	movz,f		00 1010	10	a	LF	74	4a	J
sltiu	movn,f		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w,f	00 1100	12	c	FF	76	4c	L
ori		trunc.w,f	00 1101	13	d	CR	77	4d	M
xori		ceil.w,f	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w,f	00 1111	15	f	SI	79	4f	O
(2)	mthi		01 0000	16	10	DLE	80	50	P
mflo	movz,f		01 0001	17	11	DC1	81	51	Q
mtlo	movn,f		01 0010	18	12	DC2	82	52	R
			01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
	mult		01 1000	24	18	CAN	88	58	X
	multu		01 1001	25	19	EM	89	59	Y
	div		01 1010	26	1a	SUB	90	5a	Z
	divu		01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	J
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	_
lb	add	cvt.s,f	10 0000	32	20	Space	96	60	-
lh	addu	cvt.d,f	10 0001	33	21	!	97	61	a
lw	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w,f	10 0100	36	24	\$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lw	xor		10 0110	38	26	&	102	66	f
lw	nor		10 0111	39	27	,	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l
			10 1101	45	2d	-	109	6d	m
			10 1110	46	2e	.	110	6e	n
			10 1111	47	2f	/	111	6f	o
ll	tge	c,f,f	11 0000	48	30	0	112	70	p
lwcl	tgeu	c.unf	11 0001	49	31	1	113	71	q
lwcl2	tilt	c.eqaf	11 0010	50	32	2	114	72	r
pref	tltu	c.uecf,f	11 0011	51	33	3	115	73	s
teq		c.oldt,f	11 0100	52	34	4	116	74	t
ldcl	c.ultf		11 0101	53	35	5	117	75	u
dc2	tne	c.olef	11 0110	54	36	6	118	76	v
		c.ulcf,f	11 0111	55	37	7	119	77	w
sc	c.sff,f		11 1000	56	38	8	120	78	x
swcl	c.nglef		11 1001	57	39	9	121	79	y
swc2	c.seqf		11 1010	58	3a	:	122	7a	z
		c.nglf,f	11 1011	59	3b	:	123	7b	{
		c.lt,f	11 1100	60	3c	<	124	7c	
sdc1	c.ngef,f		11 1101	61	3d	=	125	7d	}
sdc2	c.lef		11 1110	62	3e	>	126	7e	~
		c.ngtf,f	11 1111	63	3f	?	127	7f	DEL

(1) opcode(31:26) == 0

(2) opcode(31:26) == 1 if fmt(25:21) == 16 and (10) == f == s (single);

if fmt(25:21) == 17 ten (11hex) f = d (double)

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 4th ed.

#### IEEE 754 FLOATING-POINT STANDARD

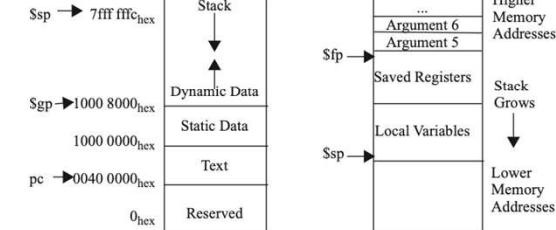
$(-1)^S \times (1 + Fraction) \times 2^{(Exponent - Bias)}$   
where Single Precision Bias = 127,  
Double Precision Bias = 1023.

#### IEEE Single Precision and Double Precision Formats:

IEEE 754 Symbols		
Exponent	Fraction	Object
0	0	$\pm 0$
0	$\neq 0$	$\pm$ Denorm
1 to MAX - 1	anything	$\pm$ Fl. Pt. Num.
MAX	0	$\pm \infty$
MAX	$\neq 0$	NaN

S.P. MAX = 255, D.P. MAX = 2047

#### MEMORY ALLOCATION



#### DATA ALIGNMENT

Double Word							
Word				Word			
Halfword		Halfword		Halfword		Halfword	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0	1	2	3	4	5	6	7

Value of three least significant bits of byte address (Big Endian)

#### EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS

B	D	Interrupt Mask	Exception Code
31	15	8	6
		15	4

BD = Branch Delay, UM = User Mode, EL = Exception Level, IE = Interrupt Enable

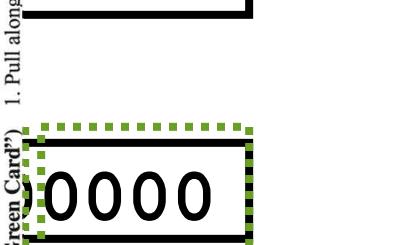
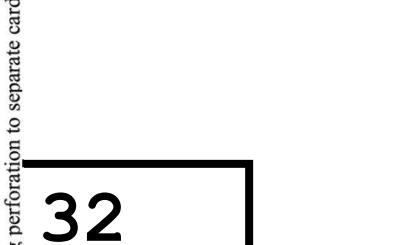
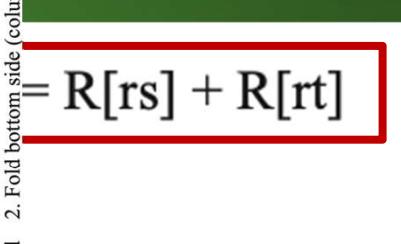
#### EXCEPTION CODES

Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	System Call Exception	15	FPE	Floating Point Exception

#### SIZE PREFIXES (10<sup>x</sup> for Disk, Communication; 2<sup>x</sup> for Memory)

SIZE	PRE-FIX	SIZE	PRE-FIX	SIZE	PRE-FIX
$10^3, 2^{10}$	Kilo-	$10^{15}, 2^{50}$	Peta-	$10^{-3}$	milli-
$10^6, 2^{20}$	Mega-	$10^{18}, 2^{60}$	Exa-	$10^{-6}$	micro-
$10^9, 2^{30}$	Giga-	$10^{21}, 2^{70}$	Zetta-	$10^{-9}$	nano-
$10^{12}, 2^{40}$	Tera-	$10^{24}, 2^{80}$	Yotta-	$10^{-12}$	pico-

The symbol for each prefix is just its first letter, except μ is used for micro.



1. Pull along perforation to separate card  
2. Fold bottom side (columns 3 and 4) together

= R[rs] + R[rt]

32

0000 hex

X

MIPS Reference Data Card ("Green Card")

## OPCODES, BASE CONVERSION, ASCII SYMBOLS

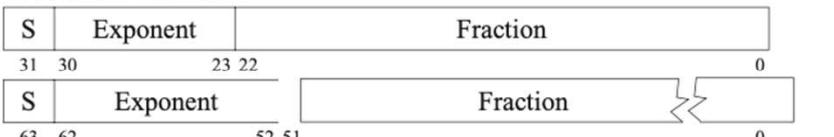
MIPS (1)	MIPS (2)	MIPS (3)	Binary	Deci-mal	Hexa-deci-mal	ASCII Char-acter	Deci-mal	Hexa-deci-mal	ASCII Char-acter
opcode (31:26)	funct (5:0)	funct (5:0)							
(1)	sll	add.f	00 0000	0	0	NUL	64	40	@
		sub.f	00 0001	1	1	SOH	65	41	A
j	srl	mul.f	00 0010	2	2	STX	66	42	B
jal	sra	div.f	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt.f	00 0100	4	4	EOT	68	44	D
bne		abs.f	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov.f	00 0110	6	6	ACK	70	46	F
bgtz	srv	neg.f	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w.f	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w.f	00 1101	13	d	CR	77	4d	M
xori		ceil.w.f	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w.f	00 1111	15	f	SI	79	4f	O
(2)	mfhi		01 0000	16	10	DLE	80	50	P
	mthi		01 0001	17	11	DC1	81	51	Q
mflo	movz.f		01 0010	18	12	DC2	82	52	R
mtlo	movn.f		01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
mult			01 1000	24	18	CAN	88	58	X
multu			01 1001	25	19	EM	89	59	Y
div			01 1010	26	1a	SUB	90	5a	Z
divu			01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	_
lb	add	cvt.s.f	10 0000	32	20	Space	96	60	'
lh	addu	cvt.d.f	10 0001	33	21	!	97	61	a
lw	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w.f	10 0100	36	24	\$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	,	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l

## STANDARD

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

where Single Precision Bias = 127,  
Double Precision Bias = 1023.

## IEEE Single Precision and Double Precision Formats:



## MEMORY ALLOCATION

\$sp → 7fff fffc<sub>hex</sub>

\$gp → 1000 8000<sub>hex</sub>

1000 0000<sub>hex</sub>

pc → 0040 0000<sub>hex</sub>

0<sub>hex</sub>

Stack

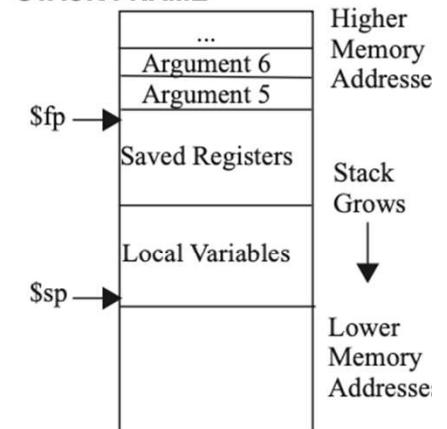
Dynamic Data

Static Data

Text

Reserved

## STACK FRAME

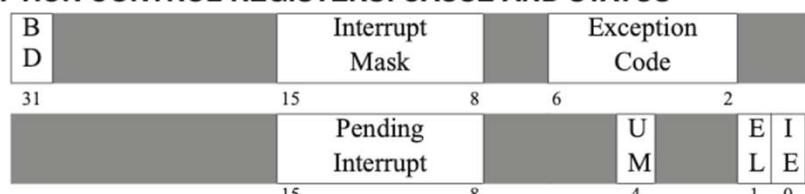


## DATA ALIGNMENT

Double Word							
Word				Word			
Halfword		Halfword		Halfword		Halfword	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0	1	2	3	4	5	6	7

Value of three least significant bits of byte address (Big Endian)

## EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS



BD = Branch Delay, UM = User Mode, EL = Exception Level, IE = Interrupt Enable

## OPCODES, BASE CONVERSION, ASCII SYMBOLS

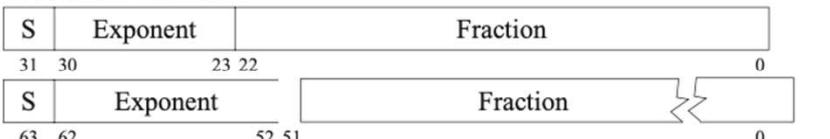
MIPS (1) MIPS (2)	MIPS (31:26)	MIPS (5:0)	Binary	Deci-mal	Hexa-deci-mal	ASCII Char-acter	Deci-mal	Hexa-deci-mal	ASCII Char-acter
(1)	sll	add,f	00 0000	0	0	NUL	64	40	@
		sub,f	00 0001	1	1	SOH	65	41	A
j	srl	mul,f	00 0010	2	2	STX	66	42	B
jal	sra	div,f	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt,f	00 0100	4	4	EOT	68	44	D
bne		abs,f	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov,f	00 0110	6	6	ACK	70	46	F
bgtz	srv	neg,f	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w,f	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w,f	00 1101	13	d	CR	77	4d	M
xori		ceil.w,f	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w,f	00 1111	15	f	SI	79	4f	O
(2)	mfhi		01 0000	16	10	DLE	80	50	P
	mthi		01 0001	17	11	DC1	81	51	Q
mflo	movz,f		01 0010	18	12	DC2	82	52	R
mtlo	movn,f		01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
mult			01 1000	24	18	CAN	88	58	X
multu			01 1001	25	19	EM	89	59	Y
div			01 1010	26	1a	SUB	90	5a	Z
divu			01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	-
lb	add	cvt.s,f	10 0000	32	20	Space	96	60	'
lh	addu	cvt.d,f	10 0001	33	21	!	97	61	a
lw	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w,f	10 0100	36	24	\$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	,	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l

## STANDARD

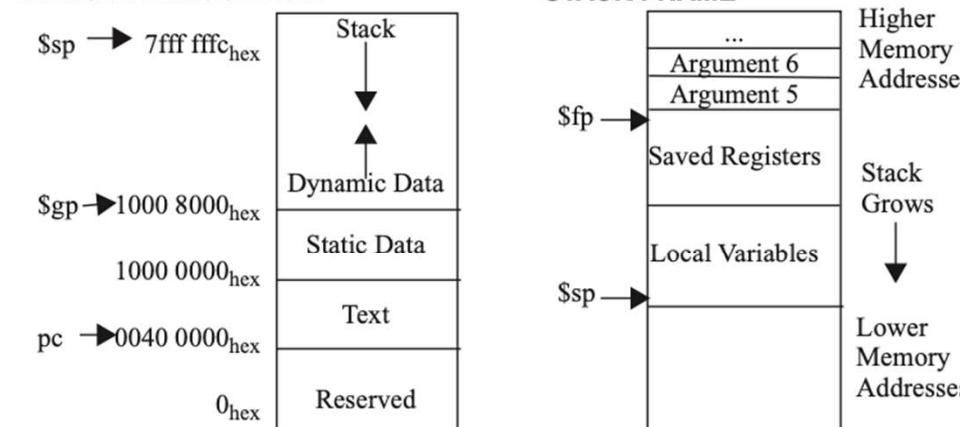
$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

where Single Precision Bias = 127,  
Double Precision Bias = 1023.

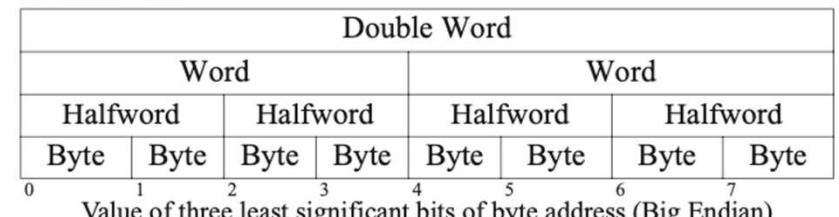
## IEEE Single Precision and Double Precision Formats:



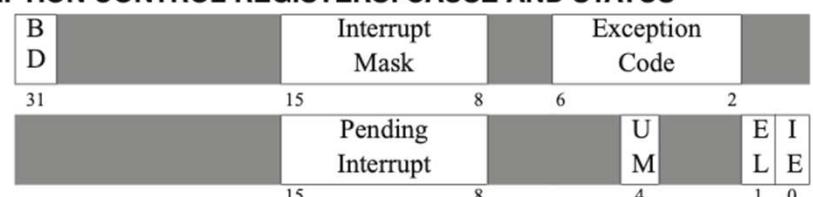
## MEMORY ALLOCATION



## DATA ALIGNMENT



## EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS



BD = Branch Delay, UM = User Mode, EL = Exception Level, IE = Interrupt Enable

## IEEE 754 Symbols

Exponent	Fraction	Object
0	0	± 0
0	≠ 0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Nu
MAX	0	±∞
MAX	≠ 0	NaN

S.P. MAX = 255, D.P. MAX = 2047

## OPCODES, BASE CONVERSION, ASCII SYMBOLS

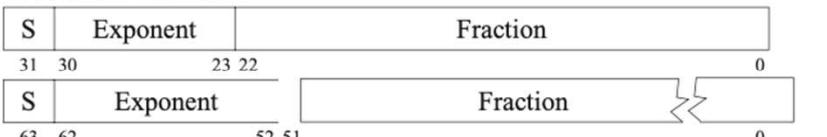
MIPS	(1) MIPS	(2) MIPS	Binary	Deci-	Hexa-	ASCII	Deci-	Hexa-	ASCII
opcode	funct	funct		mal	decim-	Char-	mal	decim-	Char-
(31:26)	(5:0)	(5:0)			al	acter		al	acter
(1)	sll	add.f	00 0000	0	0	NUL	64	40	@
		sub.f	00 0001	1	1	SOH	65	41	A
j	srl	mul.f	00 0010	2	2	STX	66	42	B
jal	sra	div.f	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt.f	00 0100	4	4	EOT	68	44	D
bne		abs.f	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov.f	00 0110	6	6	ACK	70	46	F
bgtz	srv	neg.f	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w.f	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w.f	00 1101	13	d	CR	77	4d	M
xori		ceil.w.f	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w.f	00 1111	15	f	SI	79	4f	O
(2)	mfhi		01 0000	16	10	DLE	80	50	P
	mthi		01 0001	17	11	DC1	81	51	Q
mflo	movz.f		01 0010	18	12	DC2	82	52	R
mtlo	movn.f		01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
mult			01 1000	24	18	CAN	88	58	X
multu			01 1001	25	19	EM	89	59	Y
div			01 1010	26	1a	SUB	90	5a	Z
divu			01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	-
lb	add	cvt.s.f	10 0000	32	20	Space	96	60	'
lh	addu	cvt.d.f	10 0001	33	21	!	97	61	a
lw	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w.f	10 0100	36	24	\$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	,	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l

## STANDARD

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

where Single Precision Bias = 127,  
Double Precision Bias = 1023.

## IEEE Single Precision and Double Precision Formats:

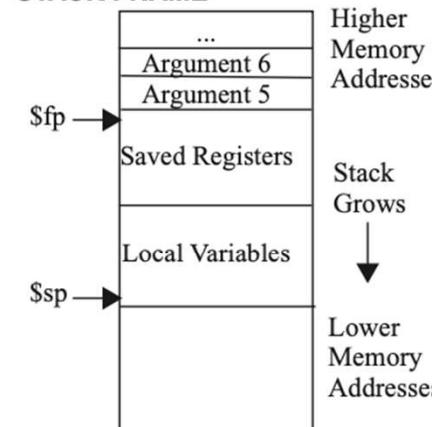


## IEEE 754 Symbols

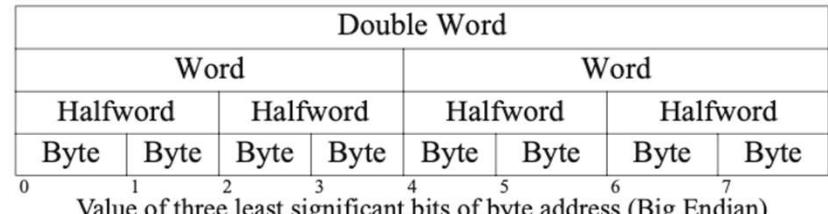
Exponent	Fraction	Object
0	0	$\pm 0$
0	$\neq 0$	$\pm$ Denorm
1 to MAX - 1	anything	$\pm$ Fl. Pt. Nu
MAX	0	$\pm\infty$
MAX	$\neq 0$	NaN

S.P. MAX = 255, D.P. MAX = 2047

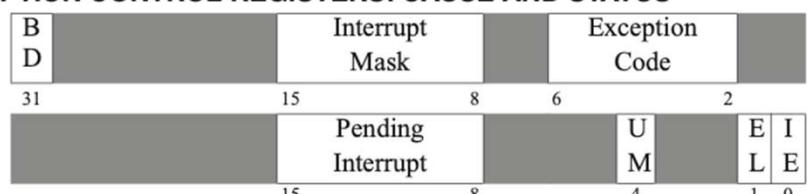
## STACK FRAME



## DATA ALIGNMENT



## EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS



BD = Branch Delay, UM = User Mode, EL = Exception Level, IE = Interrupt Enable

## OPCODES, BASE CONVERSION, ASCII SYMBOLS

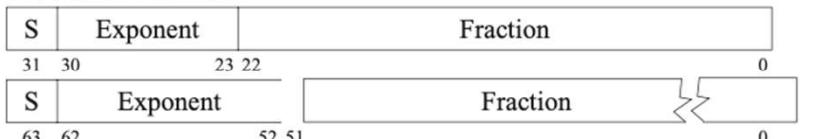
MIPS	(1) MIPS	(2) MIPS	Binary	Deci-	Hexa-	ASCII	Deci-	Hexa-	ASCII
opcode	funct	funct		mal	decim-	Char-	mal	decim-	Char-
(31:26)	(5:0)	(5:0)			al	acter		al	acter
(1)	sll	add.f	00 0000	0	0	NUL	64	40	@
		sub.f	00 0001	1	1	SOH	65	41	A
j	srl	mul.f	00 0010	2	2	STX	66	42	B
jal	sra	div.f	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt.f	00 0100	4	4	EOT	68	44	D
bne		abs.f	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov.f	00 0110	6	6	ACK	70	46	F
bgtz	srav	neg.f	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w.f	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w.f	00 1101	13	d	CR	77	4d	M
xori		ceil.w.f	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w.f	00 1111	15	f	SI	79	4f	O
(2)	mfhi		01 0000	16	10	DLE	80	50	P
	mthi		01 0001	17	11	DC1	81	51	Q
mflo	movz.f		01 0010	18	12	DC2	82	52	R
mtlo	movn.f		01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
mult			01 1000	24	18	CAN	88	58	X
multu			01 1001	25	19	EM	89	59	Y
div			01 1010	26	1a	SUB	90	5a	Z
divu			01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	-
lb	add	cvt.s.f	10 0000	32	20	Space	96	60	'
lh	addu	cvt.d.f	10 0001	33	21	!	97	61	a
lw	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w.f	10 0100	36	24	\$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	,	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l

## STANDARD

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

where Single Precision Bias = 127,  
Double Precision Bias = 1023.

## IEEE Single Precision and Double Precision Formats:

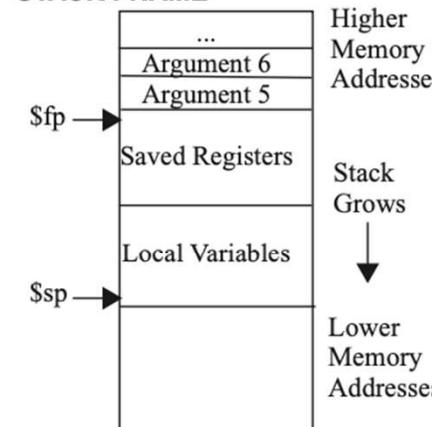


## IEEE 754 Symbols

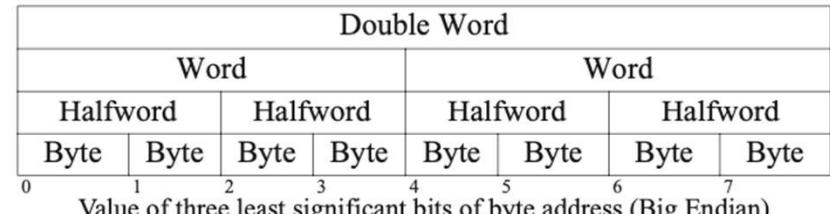
Exponent	Fraction	Object
0	0	$\pm 0$
0	$\neq 0$	$\pm$ Denorm
1 to MAX - 1	anything	$\pm$ Fl. Pt. Nu
MAX	0	$\pm\infty$
MAX	$\neq 0$	NaN

S.P. MAX = 255, D.P. MAX = 2047

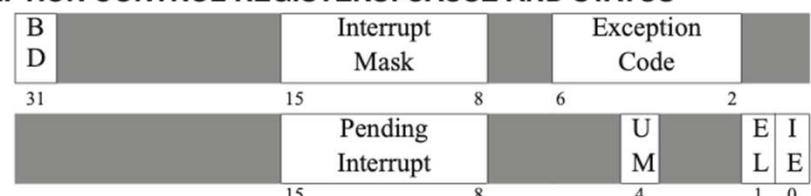
## STACK FRAME



## DATA ALIGNMENT



## EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS



BD = Branch Delay, UM = User Mode, EL = Exception Level, IE = Interrupt Enable



FORMATO I

# Instruções com o Formato I (1/3)

“campos” com:  $6 + 5 + 5 + 16 = 32$  (bits)

6	5	5	16
---	---	---	----

Designação dos campos:

opcode	rs	rt	immediate
--------	----	----	-----------

**Conceito chave:** *opcode* tem a mesma posição mas é  $\neq 0$ ! (cf., formato R é 0)

# Instru

“campo

6

Designa

opcode

Conceito  
mas é ≠

MIPS Reference Data Card (“Green Card”)

1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

## MIPS Reference Data

### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2) 25 <sub>hex</sub>
Load Linked	ll	I R[rt]=M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I R[rt]=(imm, 16'b0)	f <sub>hex</sub>
Load Word	lw	I R[rt]=M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor	R R[rd] = ~ (R[rs]   R[rt])	0 / 27 <sub>hex</sub>
Or	or	R R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>
Set Less Than Imm.	siti	I R[rt] = (R[rs] < SignExtImm) ? 1 : 0 (2)	a <sub>hex</sub>
Set Less Than Imm. Unsigned	sltiu	I R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6) b <sub>hex</sub>
Set Less Than Unsigned	sltu	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6) 0 / 2b <sub>hex</sub>
Shift Left Logical	sll	R R[rd] = R[rt] << shamt	0 / 00 <sub>hex</sub>
Shift Right Logical	srl	R R[rd] = R[rt] >> shamt	0 / 02 <sub>hex</sub>
Store Byte	sb	I M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2) 28 <sub>hex</sub>
Store Conditional	sc	I M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7) 38 <sub>hex</sub>
Store Halfword	sh	I M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2) 29 <sub>hex</sub>
Store Word	sw	I M[R[rs]+SignExtImm] = R[rt]	(2) 2b <sub>hex</sub>
Subtract	sub	R R[rd] = R[rs] - R[rt]	(1) 0 / 22 <sub>hex</sub>
Subtract Unsigned	subu	R R[rd] = R[rs] - R[rt]	0 / 23 <sub>hex</sub>

### BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct	0
	31	26 25	21 20	16 15	11 10	6 5	0
I	opcode	rs	rt	immediate			0
	31	26 25	21 20	16 15			0
J	opcode	address					0
	31	26 25					0

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 4th ed.



### ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FMT / FT / FUNCT (Hex)
Branch On FP True	bclt	F1 if(FPcond)PC=PC+4+BranchAddr	(4) 11/8/-/-
Branch On FP False	bclf	F1 if(!FPcond)PC=PC+4+BranchAddr	(4) 11/8/0/-/-
Divide	div	R Lo=R[rs]/R[rt]; Hi=R[rs] % R[rt]	0/-/-/-1a
Divide Unsigned	divu	R Lo=R[rs]/R[rt]; Hi=R[rs] % R[rt]	(6) 0/-/-/1b
FP Add Single	add.s	FR F[fd]=F[fs] + F[ft]	11/10/-/-0
FP Add	add.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}	11/11/-/-0
FP Compare Single	c.x.s*	FR FPcond = ({F[fs],F[fs+1]} ? 1 : 0)	11/10/-/-y
FP Compare	c.x.d*	FR FPcond = ({F[fs],F[fs+1]} op {F[ft],F[ft+1]}) ? 1 : 0	11/11/-/-y
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)			
FP Divide Single	div.s	FR F[fd]=F[fs] / F[ft]	11/10/-/-3
FP Divide	div.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}	11/11/-/-3
FP Multiply Single	mul.s	FR F[fd]=F[fs] * F[ft]	11/10/-/-2
FP Multiply	mul.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} *	11/11/-/-2
FP Subtract Single	sub.s	FR F[fd]=F[fs] - F[ft]	11/10/-/-1
FP Subtract	sub.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}	11/11/-/-1
Load FP Single	lwcl	I F[rt]=M[R[rs]+SignExtImm]	(2) 31/-/-/-
Load FP	ldcl	I F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2) 35/-/-/-
Move From Hi	mfhi	R R[rd] = Hi	0 / -/-/10
Move From Lo	mflo	R R[rd] = Lo	0 / -/-/12
Move From Control	mfc0	R R[rd] = CR[rs]	10 / 0/-/0
Multiply	mult	R {Hi,Lo} = R[rs] * R[rt]	0 / -/-/18
Multiply Unsigned	multu	R {Hi,Lo} = R[rs] * R[rt]	(6) 0 / -/-/19
Shift Right Arith.	sra	R R[rd] = R[rt] >> shamt	0 / -/-/3
Store FP Single	swcl	I M[R[rs]+SignExtImm] = F[rt]	(2) 39/-/-/-
Store FP	sdcl	I M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2) 3d/-/-/-

### FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fint	ft	fs	fd	funct	0
	31	26 25	21 20	16 15	11 10	6 5	0
FI	opcode	fint	ft	immediate			0
	31	26 25	21 20	16 15			0

### PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]≤R[rt]) PC = Label
Branch Greater Than or Equal	beq	if(R[rs]≥R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

### REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

sição

# Instru

“campo

6

Designa

opcode

Conceito  
mas é ≠

MIPS Reference Data Card (“Green Card”)

1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

## MIPS Reference Data

### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2) 25 <sub>hex</sub>
Load Linked	ll	I R[rt]=M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I R[rt]=(imm, 16'b0)	f <sub>hex</sub>
Load Word	lw	I R[rt]=M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor	R R[rd] = ~ (R[rs]   R[rt])	0 / 27 <sub>hex</sub>
Or	or	R R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>
Set Less Than Imm.	siti	I R[rt] = (R[rs] < SignExtImm) ? 1 : 0 (2)	a <sub>hex</sub>
Set Less Than Imm. Unsigned	sltiu	I R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6) b <sub>hex</sub>
Set Less Than Unsigned	sltu	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6) 0 / 2b <sub>hex</sub>
Shift Left Logical	sll	R R[rd] = R[rt] << shamt	0 / 00 <sub>hex</sub>
Shift Right Logical	srl	R R[rd] = R[rt] >> shamt	0 / 02 <sub>hex</sub>
Store Byte	sb	I M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2) 28 <sub>hex</sub>
Store Conditional	sc	I M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7) 38 <sub>hex</sub>
Store Halfword	sh	I M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2) 29 <sub>hex</sub>
Store Word	sw	I M[R[rs]+SignExtImm] = R[rt]	(2) 2b <sub>hex</sub>
Subtract	sub	R R[rd] = R[rs] - R[rt]	(1) 0 / 22 <sub>hex</sub>
Subtract Unsigned	subu	R R[rd] = R[rs] - R[rt]	0 / 23 <sub>hex</sub>

(1) May cause overflow exception

(2) SignExtImm = { 16{immediate[15]}, immediate }

(3) ZeroExtImm = { 16{b'0}, immediate }

(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }

(5) JumpAddr = { PC+4[31:28], address, 2'b0 }

(6) Operands considered unsigned numbers (vs. 2's comp.)

(7) Atomic test&set pair: R[rt]=1 if pair atomic, 0 if not atomic

### BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct	0
31	26 25	21 20	16 15	11 10	6 5		0
I	opcode	rs	rt	immediate			0
31	26 25	21 20	16 15				0
J	opcode	address					0
31	26 25						0

Copyright 2009 by Elsevier, Inc. All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 4th ed.



### ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FMT / FT / FUNCT (Hex)
Branch On FP True	bclt	FI if(FPcond)PC=PC+4+BranchAddr	(4) 11/8/-/-
Branch On FP False	bclf	FI if(!FPcond)PC=PC+4+BranchAddr	(4) 11/8/0/-/-
Divide	div	R Lo=R[rs]/R[rt]; Hi=R[rs]%-R[rt]	0/-/-/-1a
Divide Unsigned	divu	R Lo=R[rs]/R[rt]; Hi=R[rs]%-R[rt]	(6) 0/-/-/-1b
FP Add Single	adds	FR F[fd]=F[fs]+F[ft]	11/10/-/-0
FP Add	add.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}	11/11/-/-0
FP Compare Single	c.x.s*	FR FPcond = (F[fs] op F[ft]) ? 1 : 0	11/10/-/-y
FP Compare	c.x.d*	FR FPcond = ({F[fs],F[fs+1]} op {F[ft],F[ft+1]}) ? 1 : 0	11/11/-/-y
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)			
FP Divide Single	div.s	FR F[fd]=F[fs]/F[ft]	11/10/-/-3
FP Divide	div.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}	11/11/-/-3
FP Multiply Single	mul.s	FR F[fd]=F[fs] * F[ft]	11/10/-/-2
FP Multiply	mul.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} *	11/11/-/-2
FP Subtract Single	sub.s	FR F[fd]=F[fs] - F[ft]	11/10/-/-1
FP Subtract	sub.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}	11/11/-/-1
Load FP Single	lwcl	I F[rt]=M[R[rs]+SignExtImm]	(2) 31/-/-/-/-
Load FP	ldcl	I F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2) 35/-/-/-/-
Move From Hi	mfhi	R R[rd] = Hi	0 / -/-/10
Move From Lo	mflo	R R[rd] = Lo	0 / -/-/12
Move From Control	mcfc0	R R[rd] = CR[rs]	10 / 0/-/0
Multiply	mult	R {Hi,Lo} = R[rs] * R[rt]	0 / -/-/18
Multiply Unsigned	multu	R {Hi,Lo} = R[rs] * R[rt]	(6) 0 / -/-/19
Shift Right Arith.	sra	R R[rd] = R[rt] >> shamt	0 / -/-/3
Store FP Single	swcl	I M[R[rs]+SignExtImm] = F[rt]	(2) 39/-/-/-/-
Store FP	sdcl	I M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2) 3d/-/-/-/-

### FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fint	ft	fs	fd	funct	0
31	26 25	21 20	16 15	11 10	6 5		0
FI	opcode	fint	ft	immediate			0
31	26 25	21 20	16 15				0

### PSEUDOINSTRUCTION SET

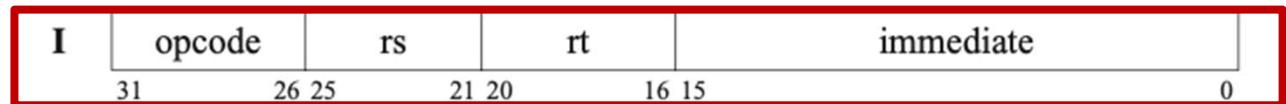
NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]≤R[rt]) PC = Label
Branch Greater Than or Equal	bege	if(R[rs]≥R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

### REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

sição

# Instruções com o Formato I (2/3)



## Qual o significado destes campos?

- **opcode:** especifica a instrução
- **rs:** indica o **único** registo a operar (se existir)
- **rt:** indica o registo que recebe o resultado

# Instruções com o Formato I (3/3)



## O campo “*Immediate*”:

Para as instruções *addi*, *slti*, *sltiu*, a constante de 16 bit sofre **extensão do sinal** para 32 bits (inteiro com sinal)

Caso o número não seja representável em 16 bits?  
(solução mais tarde...)

# Exemplo do Formato I (1/2)

**Instrução MIPS:**

**addi \$21 , \$22 , 50**

**opcode** = 8 **(ver na tabela)**

**rs** = 22 **(primeiro operando - \$s6)**

**rt** = 21 **(registo de destino - \$s5)**

**immediate** = 50 **(em decimal, por defeito)**

Exem  
Instrução  
addi  
opcod  
rs  
rt  
immed

MIPS Reference Data Card (“Green Card”)

1. Pull along perforation to separate card  
2. Fold bottom side (columns 3 and 4) together

## MIPS Reference Data

### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT
Add	add	R[Rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	add	R[Rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R[Rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I R[rt]=(24'b0,M[R[rs]+SignExtImm](7:0))	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I R[rt]=(16'b0,M[R[rs]+SignExtImm](15:0))	(2) 25 <sub>hex</sub>
Load Linked	ll	I R[rt]=M[R[rs]:SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I R[rt]={imm, 16'b0}	f <sub>hex</sub>
Load Word	lw	I R[Rd] = M[R[rs]:SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor	R[Rd] = ~{R[rs]   R[rt]}	0 / 27 <sub>hex</sub>
Or	or	R[Rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R[Rd] = {R[rs] < R[rt]} ? 1 : 0	0 / 2a <sub>hex</sub>
Set Less Than Imm.	slti	I R[rt] = {R[rs] < SignExtImm} ? 1 : 0 (2)	a <sub>hex</sub>
Set Less Than Imm. Unsigned	sltiu	I R[rt] = {R[rs] < SignExtImm} ? 1 : 0	(2,6) b <sub>hex</sub>
Set Less Than Unsigned	situ	R[Rd] = {R[rs] < R[rt]} ? 1 : 0	(6) 0 / 2b <sub>hex</sub>
Shift Left Logical	sll	R[Rd] = R[rt] << shampt	0 / 00 <sub>hex</sub>
Shift Right Logical	srl	R[Rd] = R[rt] >> shampt	0 / 02 <sub>hex</sub>
Store Byte	sb	I M[R[rs]:SignExtImm](7:0) = R[rt](7:0)	(2) 28 <sub>hex</sub>
Store Conditional	sc	I M[R[rs]:SignExtImm] = R[rt]; R[rt] = {atomic} ? 1 : 0	(2,7) 38 <sub>hex</sub>
Store Halfword	sh	I M[R[rs]:SignExtImm](15:0) = R[rt](15:0)	(2) 29 <sub>hex</sub>
Store Word	sw	I M[R[rs]:SignExtImm] = R[rt]	(2) 2b <sub>hex</sub>
Subtract	sub	R[Rd] = R[rs] - R[rt]	(1) 0 / 22 <sub>hex</sub>
Subtract Unsigned	subu	R[Rd] = R[rs] - R[rt]	0 / 23 <sub>hex</sub>

- (1) May cause overflow exception
- (2) SignExtImm = { 16{immediate[15]}, immediate }
- (3) ZeroExtImm = { 16{1b'0}, immediate }
- (4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
- (5) JumpAddr = { PC+4[31:28], address, 2'b0 }
- (6) Operands considered unsigned numbers (vs. 2's comp.)
- (7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

### BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct	
	31	26 25	21 20	16 15	11 10	6 5	0
I	opcode	rs	rt			immediate	
	31	26 25	21 20	16 15			0
J	opcode				address		
	31	26 25					0

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 4th ed.



### ARITHMETIC CORE INSTRUCTION SET

① NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FMT / FT / FUNCT (Hex)
Branch On FP True	bc1t	FI if(FPcond)PC=PC+4+BranchAddr	(4) 11/8/1--
Branch On FP False	bc1f	FI if(!FPcond)PC=PC+4+BranchAddr	(4) 11/8/0--
Divide	div	R Lo=R[rs]/R[rt]; Hi=R[rs]%-R[rt]	0/-/-/1a
Divide Unsigned	divu	R Lo=R[rs]/R[rt]; Hi=R[rs]%-R[rt]	(6) 0/-/-/1b
FP Add Single	add.s	FR F[fd] = F[fs] + F[ft]	11/10/-/0
FP Add	add.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}	11/11/-/0
Double			
FP Compare Single	c.x.s*	FR FPcond = {F[fs] op F[ft]} ? 1 : 0	11/10/-/y
FP Compare	c.x.d*	FR FPcond = ({F[fs],F[fs+1]} op {F[ft],F[ft+1]}) ? 1 : 0	11/11/-/y
Double			

\* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)

FP Divide Single	div.s	FR F[Rd] = F[fs] / F[ft]	11/10/-/3
FP Divide	div.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}	11/11/-/3
Double			
FP Multiply Single	mul.s	FR F[fd] = F[fs] * F[ft]	11/10/-/2
FP Multiply	mul.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} *	11/11/-/2
Double			
FP Subtract Single	sub.s	FR F[Rd] = F[fs] - F[ft]	11/10/-/1
FP Subtract	sub.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}	11/11/-/1
Double			
Load FP Single	lwcl	I F[Rd]=M[R[rs]+SignExtImm]	(2) 31/-/-/-
Load FP	ldcl	I F[Rd]=M[R[rs]+SignExtImm]; F[Rt+1]=M[R[rs]+SignExtImm+4]	(2) 35/-/-/-
Double			
Move From Hi	mfhi	R R[Rd]=Hi	0 / -/-/10
Move From Lo	mflo	R R[Rd]=Lo	0 / -/-/12
Move From Control	mfc0	R R[Rd]=CR[rs]	10 / 0/-/0
Multiply	mult	R {Hi,Lo} = R[rs] * R[rt]	0 / -/-/18
Multiply Unsigned	mult	R {Hi,Lo} = R[rs] * R[rt]	(6) 0 / -/-/19
Shift Right Arith.	sra	R R[Rd] = R[rt]>> shampt	0 / -/-/3
Store FP Single	swcl	I M[R[rs]+SignExtImm] = F[rt]	(2) 39/-/-/-
Store FP	sdcl	I M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2) 3d/-/-/-
Double			

### FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fint	ft	fs	fd	funct	
	31	26 25	21 20	16 15	11 10	6 5	0
FI	opcode	fint	ft		immediate		
	31	26 25	21 20	16 15			0

### PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[Rd] = immediate
Move	move	R[Rd] = R[rs]

### REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

O - \$s6)  
C - \$s5)  
jefeito)



Exem  
Instrução  
addi  
opcod  
rs  
rt  
immed

MIPS Reference Data							
CORE INSTRUCTION SET				ARITHMETIC CORE INSTRUCTION SET			
NAME, MNEMONIC FOR-MAT OPERATION (in Verilog)				OPCODE / FUNCT (Hex)			
Add add R R[rd] = R[rs] + R[rt]				(1) 0 / 20hex			
Add Immediate addi I R[rt] = R[rs] + SignExtImm				(1,2) 8hex			
Add Imm. Unsigned addiu I R[rt] = R[rs] + SignExtImm				(2) 9hex			
Add Unsigned addi R R[rd] = R[rs] + R[rt]				0 / 21hex			
And and R R[rd] = R[rs] & R[rt]				0 / 24hex			
And Immediate andi I R[rt] = R[rs] & ZeroExtImm				(3) chex			
Branch On Equal beq I if(R[rs]==R[rt]) PC=PC+4+BranchAddr				(4) 4hex			
Branch On Not Equal bne I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr				(4) 5hex			
Jump j J PC=JumpAddr				(5) 2hex			
Jump And Link jal J R[31]=PC+8;PC=JumpAddr				(5) 3hex			
Jump Register jr R PC=R[rs]				0 / 08hex			
Load Byte Unsigned lbu I R[rt]=(24'b0,M[R[rs]+SignExtImm](7:0))				(2) 24hex			
Load Halfword Unsigned lhu I R[rt]=(16'b0,M[R[rs]+SignExtImm](15:0))				(2) 25hex			
Load Linked ll I R[rt]=M[R[rs]+SignExtImm]				(2,7) 30hex			
Load Upper Imm. lui I R[rt]={imm, 16'b0}				fhex			
Load Word lw I R[rt]=M[R[rs]+SignExtImm]				(2) 23hex			
Nor nor R R[rd] = ~ (R[rs] & R[rt])				0 / 27hex			
Or or R R[rd] = R[rs]   R[rt]				0 / 25hex			
Or Immediate ori I R[rt] = R[rs]   ZeroExtImm				(3) dhex			
Set Less Than slt R R[rd] = (R[rs] < R[rt]) ? 1 : 0				0 / 2a hex			
Set Less Than Imm. slti I R[rt] = (R[rs] < SignExtImm) ? 1 : 0 (2)				ahex			
Set Less Than Imm. Unsigned sltiu I R[rt] = (R[rs] < SignExtImm) ? 1 : 0				(2,6) bhex			
Set Less Than Unsigned. situ R R[rd] = (R[rs] < R[rt]) ? 1 : 0				(6) 0 / 2bhex			
Shift Left Logical sll R R[rd] << shampt				0 / 00hex			
Shift Right Logical srl R R[rd] >> shampt				0 / 02hex			
Store Byte sb I M[R[rs]+SignExtImm](7:0) = R[rt](7:0)				(2) 28hex			
Store Conditional sc I M[R[rs]+SignExtImm] = R[rt]; R[rt]=(atomic) ? 1 : 0				(2,7) 38hex			
Store Halfword sh I M[R[rs]+SignExtImm](15:0) = R[rt](15:0)				(2) 29hex			
Store Word sw I M[R[rs]+SignExtImm] = R[rt]				(2) 2bhex			
Subtract sub R R[rd] = R[rs] - R[rt]				(1) 0 / 22hex			
Subtract Unsigned subu R R[rd] = R[rs] - R[rt]				0 / 23hex			
(1) May cause overflow exception							
(2) SignExtImm = { 16{immediate[15]}, immediate }							
(3) ZeroExtImm = { 16{1b'0}, immediate }							
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }							
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }							
(6) Operands considered unsigned numbers (vs. 2's comp.)							
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic							
BASIC INSTRUCTION FORMATS							
R	opcode	rs	rt	rd	shamt	funct	
I	31 26 25	21 20	16 15	11 10	6 5	0	
J	31 26 25	21 20	16 15		immediate		
						0	
1. Pull along perforation to separate card							
2. Fold bottom side (columns 3 and 4) together							
MIPS Reference Data Card ("Green Card")				FLOATING-POINT INSTRUCTION FORMATS			
FR	opcode	fint	ft	fs	fd	funct	
FI	31 26 25	21 20	16 15	11 10	6 5	0	
	opcode	fint	ft		immediate		
	31 26 25	21 20	16 15			0	
PSEUDOINSTRUCTION SET							
NAME	MNEMONIC	OPERATION					
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label					
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label					
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label					
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label					
Load Immediate	li	R[rd] = immediate					
Move	move	R[rd] = R[rs]					
REGISTER NAME, NUMBER, USE, CALL CONVENTION							
NAME	NUMBER	USE	PRESERVED ACROSS A CALL?				
\$zero	0	The Constant Value 0	N.A.				
Sat	1	Assembler Temporary	No				
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No				
\$a0-\$a3	4-7	Arguments	No				
\$t0-\$t7	8-15	Temporaries	No				
\$s0-\$s7	16-23	Saved Temporaries	Yes				
\$t8-\$t9	24-25	Temporaries	No				
\$k0-\$k1	26-27	Reserved for OS Kernel	No				
\$gp	28	Global Pointer	Yes				
\$sp	29	Stack Pointer	Yes				
\$fp	30	Frame Pointer	Yes				
\$ra	31	Return Address	Yes				

Copyright 2009 by Elsevier, Inc. All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.



O - \$s6)  
C - \$s5)  
efeito)

operation to separate card 2. Fold bottom side (columns 3 and 4) together

# MIPS Reference Data



## CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2) 25 <sub>hex</sub>
Load Linked	ll	I R[rt] = M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I R[rt] = {imm, 16'b0}	f <sub>hex</sub>
Load Word	lw	I R[rt] = M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor	R R[rd] = ~ (R[rs]   R[rt])	0 / 27 <sub>hex</sub>
Or	or	R R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>

NAME, MNEMONIC	FOR-MAT
Branch On FP True	bc1t FI if(
Branch On FP False	bc1f FI if(
Divide	div R Lc
Divide Unsigned	divu R Lc
FP Add Single	add.s FR F[
FP Add	add.d FR {F
Double	
FP Compare Single	c.x.s* FR FI
FP Compare Double	c.x.d* FR FI
FP Divide Single	div.s FR F[
FP Divide Double	div.d FR {F
FP Multiply Single	mul.s FR F[
FP Multiply Double	mul.d FR {F
FP Subtract Single	sub.s FR F[
FP Subtract Double	sub.d FR {F
Load FP Single	lwcl I F[
Load FP Double	ldcl I F[
Move From Hi	mfhi R R[
Move From Lo	mflo R R[
Move From Control	mfc0 R R[
Multiply	mult R {F
Multiply Unsigned	multu R {F
Shift Right Arith.	sra R R[
Store FP Single	swcl I M
Store FP Double	sdc1 I M

\* (x is eq, lt, or le) (op is ==,

FP Divide Single

FP Divide Double

FP Multiply Single

FP Multiply Double

FP Subtract Single

FP Subtract Double

Load FP Single

Load FP Double

Move From Hi

Move From Lo

Move From Control

Multiply

Multiply Unsigned

Shift Right Arith.

Store FP Single

Store FP Double

Operation to separate card 2. Fold bottom side (columns 3 and 4) together

# MIPS Reference Data



## CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2) 25 <sub>hex</sub>
Load Linked	ll	I R[rt] = M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I R[rt] = {imm, 16'b0}	f <sub>hex</sub>
Load Word	lw	I R[rt] = M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor	R R[rd] = ~ (R[rs]   R[rt])	0 / 27 <sub>hex</sub>
Or	or	R R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>

NAME, MNEMONIC	FOR-MAT
Branch On FP True	bc1t FI if(
Branch On FP False	bc1f FI if(
Divide	div R Lc
Divide Unsigned	divu R Lc
FP Add Single	add.s FR F[
FP Add Double	add.d FR {F
FP Compare Single	c.x.s* FR FI
FP Compare Double	c.x.d* FR FI
FP Divide Single	div.s FR F[
FP Divide Double	div.d FR {F
FP Multiply Single	mul.s FR F[
FP Multiply Double	mul.d FR {F
FP Subtract Single	sub.s FR F[
FP Subtract Double	sub.d FR {F
Load FP Single	lwcl I F[
Load FP Double	ldcl I F[
Move From Hi	mfhi R R[
Move From Lo	mflo R R[
Move From Control	mfc0 R R[
Multiply	mult R {F
Multiply Unsigned	multu R {F
Shift Right Arith.	sra R R[
Store FP Single	swcl I M
Store FP Double	sdc1 I M

\* (x is eq, lt, or le) (op is ==,

FP Divide Single

FP Divide Double

FP Multiply Single

FP Multiply Double

FP Subtract Single

FP Subtract Double

Load FP Single

Load FP Double

Move From Hi

Move From Lo

Move From Control

Multiply

Multiply Unsigned

Shift Right Arith.

Store FP Single

Store FP Double

## Exemplo do Formato I (2/2)

Instrução MIPS:

**addi \$21 , \$22 , 50**

Representação em decimal:

$$R[rt] = R[rs] + \text{SignExtImm}$$

opcode	rs	rt	immediate
8	22	21	50

Representação em binário:

001000	10110	10101	0000000000110010
--------	-------	-------	------------------

## Exemplo do Formato I (2/2)

Instrução MIPS:

**addi \$21 , \$22 , 50**

Representação em decimal:

$$R[rt] = R[rs] + \text{SignExtImm}$$

opcode	rs	rt	immediate
8	22	21	50

Representação em binário:

001000	10110	10101	0000000000110010
hex			

## Exemplo do Formato I (2/2)

Instrução MIPS:

**addi \$21 , \$22 , 50**

Representação em decimal:

$$R[rt] = R[rs] + \text{SignExtImm}$$

opcode	rs	rt	immediate
8	22	21	50

Representação em binário:

001000	10110	10101	0000000000110010
hexadecimal: 22D5 0032 <sub>hex</sub>			hex

## Exemplo do Formato I (2/2)

Instrução MIPS:

**addi \$21, \$22, 50**

Representação em decimal:

$$R[rt] = R[rs] + \text{SignExtImm}$$

opcode	rs	rt	immediate
8	22	21	50

Representação em binário:

001000	10110	10101	0000000000110010
--------	-------	-------	------------------

hex

hexadecimal:

22D5 0032<sub>hex</sub>

decimal:

584,384,562<sub>10</sub>

# Exercício 1

Que instrução tem como código máquina  $23_{\text{hex}}$  ou  $35_{\text{dec}}$ ?

1. add \$0, \$0, \$0

opcode	rs	rt	rd	shamt	funct
--------	----	----	----	-------	-------

2. subu \$s0,\$s0,\$s0

opcode	rs	rt	rd	shamt	funct
--------	----	----	----	-------	-------

3. lw \$0, 0(\$0)

opcode	rs	rt	immediate		
--------	----	----	-----------	--	--

4. addi \$0, \$0, 35

opcode	rs	rt	immediate		
--------	----	----	-----------	--	--

5. subu \$0, \$0, \$0

opcode	rs	rt	rd	shamt	funct
--------	----	----	----	-------	-------

6. Errado! Instruções não são números

Nomes e números dos registos:

0: \$0, ..., 8: \$t0, 9:\$t1, ...,15: \$t7, 16: \$s0, 17: \$s1, ..., 23: \$s7

add: opcode = 0, funct =  $20_{\text{hex}} / 32_{\text{dec}}$

subu: opcode = 0, funct =  $23_{\text{hex}} / 35_{\text{dec}}$

lw: opcode =  $23_{\text{hex}} / 35_{\text{dec}}$

addi: opcode =  $8_{\text{hex}} / 8_{\text{dec}}$

Formato R

Formato I

# Exercício 1

Que instrução tem como código máquina  $23_{\text{hex}}$  ou  $35_{\text{dec}}$ ?

1. add \$0, \$0, \$0

opcode	rs	rt	rd	shamt	funct
--------	----	----	----	-------	-------

2. subu \$s0,\$s0,\$s0

opcode	rs	rt	rd	shamt	funct
--------	----	----	----	-------	-------

3. lw \$0, 0(\$0)

opcode	rs	rt	immediate		
--------	----	----	-----------	--	--

4. addi \$0, \$0, 35

opcode	rs	rt	immediate		
--------	----	----	-----------	--	--

5. subu \$0, \$0, \$0

opcode	rs	rt	rd	shamt	funct
--------	----	----	----	-------	-------

6. Errado! Instruções não são números

Nomes e números dos registos:

0: \$0, ..., 8: \$t0, 9:\$t1, ...,15: \$t7, 16: \$s0, 17: \$s1, ..., 23: \$s7

add: opcode = 0, funct =  $20_{\text{hex}} / 32_{\text{dec}}$

subu: opcode = 0, funct =  $23_{\text{hex}} / 35_{\text{dec}}$

lw: opcode =  $23_{\text{hex}} / 35_{\text{dec}}$

addi: opcode =  $8_{\text{hex}} / 8_{\text{dec}}$

Formato R

Formato I

## Saltos: PC como endereço base (1/5)

opcode	rs	rt	immediate
--------	----	----	-----------

*opcode* especifica *beq* ou *bne*

*rs* e *rt* são os registos a comparar

“*immediate*” indica para onde ir (i.e., label)?

*Immediate* só tem 16 bits, logo não pode indicar o endereço para onde saltar!

PC (*Program Counter*) tem o endereço da instrução que está a ser executada (um apontador)

## Saltos: PC como endereço base (2/5)

**Onde usamos geralmente os saltos condicionais?**

*if-else, while, for*

**ciclos (algumas instruções)**

**Salto condicional geralmente altera pouco o PC  
→ soma ou subtrai um pequeno valor ao PC**

## Saltos: PC como endereço base (3/5)

**Solução para codificar saltos numa instrução de 32 bits: Endereçamento Relativo ao PC**

**“Immediate” com 16 bit representa um inteiro com sinal em complemento para 2 que vai ser *somado* ao PC se fizermos o salto**

**Assim podemos saltar  $\pm 2^{15}$  a partir do PC, o que deve ser suficiente para qualquer ciclo**

**Será que podemos otimizar ainda mais?**

## Saltos: PC como endereço base (4/5)

**Alinhamento a palavra (instruções têm 32 bits)**

**Endereço é múltiplo de 4**

**Portanto termina sempre em 00<sub>bin</sub>**

**o número de bytes a somar ao PC é sempre um múltiplo de 4**

**vamos indicar o “*immediate*” em palavras**

**Sendo assim podemos saltar  $\pm 2^{15}$  palavras desde o PC (ou  $\pm 2^{17}$ )**

# Saltos: PC como endereço base (5/5)

**Cálculo do salto:**

**Se não tivermos um salto:**

$$\text{PC} = \text{PC} + 4 \text{ (endereço da próxima instrução)}$$

**Se fizermos um salto:**

$$\text{PC} = (\text{PC} + 4) + (\text{immediate} * 4)$$

“*immediate*” indica o número de instruções a saltar  
(positivo ou negativo)!

# Exemplo de um salto condicional (1/3)

## Código MIPS:

```
Loop:    beq    $9,$0,End  
          addi   $8,$8,10  
          addi   $9,$9,-1  
          j      Loop
```

End:

***beq* tem o formato I:**

**opcode = 4 (ver tabela)**

**rs = 9 (primeiro operando - \$t1)**

**rt = 0 (segundo operando - \$zero)**

**Immediate = ???**

# Código M

Lo

Er

*beq tem o*

opcode

rs

rt

Immedia

MIPS Reference Data Card ("Green Card")

2. Fold bottom side (columns 3 and 4) together

1. Pull along perforation to separate card

## MIPS Reference Data

### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-	MAT	OPERATION (in Verilog)	OPCODE / FUNCT
Add	add	R	R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R	R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I	if(R[rs]==R[rt])	4 <sub>hex</sub>
Branch On Not Equal	bne	I	if(R[rs]!=R[rt])	5 <sub>hex</sub>
Jump	j	J	PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J	PC=JalAddr; PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R	PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2) 25 <sub>hex</sub>
Load Linked	ll	I	R[rt]=M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I	R[rt]={imm, 16'b0}	f <sub>hex</sub>
Load Word	lw	I	R[rt]=M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor	R	R[rd] = ~ (R[rs]   R[rt])	0 / 27 <sub>hex</sub>
Or	or	R	R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori	I	R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R	R[rd]=(R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>
Set Less Than Imm.	slti	I	R[rt]=(R[rs] < SignExtImm) ? 1 : 0	(2) a <sub>hex</sub>
Set Less Than Imm. Unsigned	sltiu	I	R[rt]=(R[rs] < SignExtImm) ? 1 : 0	(2,6) b <sub>hex</sub>
Set Less Than Unsigned	sltiu	I	R[rt]=(R[rs] < SignExtImm) ? 1 : 0	(2,6) b <sub>hex</sub>
Set Less Than Unsig.	situ	R	R[rd]=(R[rs] < R[rt]) ? 1 : 0	(6) 0 / 2b <sub>hex</sub>
Shift Left Logical	sll	R	R[rd]=R[rt] << sham	0 / 00 <sub>hex</sub>
Shift Right Logical	srl	R	R[rd]=R[rt] >> sham	0 / 02 <sub>hex</sub>
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0)=R[rt](7:0)	(2) 28 <sub>hex</sub>
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt]=(atomic) ? 1 : 0	(2,7) 38 <sub>hex</sub>
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0)=R[rt](15:0)	(2) 29 <sub>hex</sub>
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2) 2b <sub>hex</sub>
Subtract	sub	R	R[rd]=R[rs]-R[rt]	(1) 0 / 22 <sub>hex</sub>
Subtract Unsigned	subu	R	R[rd]=R[rs]-R[rt]	0 / 23 <sub>hex</sub>

- (1) May cause overflow exception
- (2) SignExtImm = { 16{immediate[15]}, immediate }
- (3) ZeroExtImm = { 16{1b'0}, immediate }
- (4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
- (5) JumpAddr = { PC+4[31:28], address, 2'b0 }
- (6) Operands considered unsigned numbers (vs. 2's comp.)
- (7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

### BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
31	26 25	21 20	16 15	11 10	6 5	0
I	opcode	rs	rt		immediate	
31	26 25	21 20	16 15		0	
J	opcode			address		
31	26 25				0	

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.



### ARITHMETIC CORE INSTRUCTION SET

① NAME, MNEMONIC	FOR-	MAT	OPERATION	OPCODE / FMT / FUNCT (Hex)
Branch On FP True	bclt	FI	if(FPcond)PC=PC+4+BranchAddr	(4) 11/8/1--
Branch On FP False	bclf	FI	if(!FPcond)PC=PC+4+BranchAddr	11/8/0--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	0/-/-/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	0/-/-/1b
FP Add Single	add.s	FR	F[fd] = F[fs] + F[ft]	11/10/-/0
FP Add	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}	11/11/-/0
FP Compare Single	c.xs*	FR	FPcond = (F[fs] op F[ft]) ? 1 : 0	11/10/-/y
FP Compare	c.xd*	FR	FPcond = ({F[fs],F[fs+1]} op {F[ft],F[ft+1]}) ? 1 : 0	11/11/-/y
Double			*	
			(x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)	
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[rt]	11/10/-/3
FP Divide	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}	11/11/-/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]	11/10/-/2
FP Multiply	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} *	11/11/-/2
FP Subtract Single	sub.s	FR	F[fd] = F[fs] - F[ft]	11/10/-/1
FP Subtract	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}	11/11/-/1
Load FP Single	lwci	I	F[rt]=M[R[rs]+SignExtImm]	(2) 31/-/-/-
Load FP	ldci	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2) 35/-/-/-
Move From Hi	mfhi	R	R[rd]=Hi	0/-/-/10
Move From Lo	mflo	R	R[rd]=Lo	0/-/-/12
Move From Control	mfc0	R	R[rd]=CR[rs]	10/0/-/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]	0/-/-/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6) 0/-/-/19
Shift Right Arith.	sra	R	R[rd]=R[rt] >> sham	0/-/-/3
Store FP Single	swci	I	M[R[rs]+SignExtImm] = F[rt]	(2) 39/-/-/-
Store FP	sdci	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2) 3d/-/-/-

### FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
31	26 25	21 20	16 15	11 10	6 5	0

FI	opcode	fmt	ft	immediate
31	26 25	21 20	16 15	0

### PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

### REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

# Código M

Lo

Er

*beq tem o*

*opcode*

*rs*

*rt*

*Immedia*

MIPS Reference Data Card ("Green Card")

1. Pull along perforation to separate card.

2. Fold bottom side (columns 3 and 4) together.

## MIPS Reference Data

### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR- FORMAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R[Rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	addu	R[Rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R[Rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R[PC]=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2) 25 <sub>hex</sub>
Load Linked	ll	I R[rt]=M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I R[rt]={imm, 16'b0}	f <sub>hex</sub>
Load Word	lw	I R[rt]=M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor	R[Rd] = ~ (R[rs]   R[rt])	0 / 27 <sub>hex</sub>
Or	or	R[Rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R[Rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>
Set Less Than Imm.	slti	I R[rt] = (R[rs] < SignExtImm) ? 1 : 0 (2)	a <sub>hex</sub>
Set Less Than Imm. Unsigned	sltiu	I R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6) b <sub>hex</sub>
Set Less Than Unsigned	sltu	R[Rd] = (R[rs] < R[rt]) ? 1 : 0	(6) 0 / 2b <sub>hex</sub>
Shift Left Logical	sll	R[Rd] = R[rt] << sham	0 / 00 <sub>hex</sub>
Shift Right Logical	srl	R[Rd] = R[rt] >> sham	0 / 02 <sub>hex</sub>
Store Byte	sb	I M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2) 28 <sub>hex</sub>
Store Conditional	sc	I M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7) 38 <sub>hex</sub>
Store Halfword	sh	I M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2) 29 <sub>hex</sub>
Store Word	sw	I M[R[rs]+SignExtImm] = R[rt]	(2) 2b <sub>hex</sub>
Subtract	sub	R[Rd] = R[rs] - R[rt]	(1) 0 / 22 <sub>hex</sub>
Subtract Unsigned	subu	R[Rd] = R[rs] - R[rt]	0 / 23 <sub>hex</sub>

(1) May cause overflow exception

(2) SignExtImm = { 16{immediate[15]}, immediate }

(3) ZeroExtImm = { 16{1b'0} }, immediate }

(4) Branch Addr = { PC+4(immediate[15]), immediate[2'b0] }

(5) JumpAddr = { PC+4(31:28), address, 2'b0 }

(6) Operands considered unsigned numbers (vs. 2's comp.)

(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

### BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
31	26 25	21 20	16 15	11 10	6 5	0
I	opcode	rs	rt		immediate	
31	26 25	21 20	16 15		0	
J	opcode			address		
31	26 25				0	

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 4th ed.

### ARITHMETIC CORE INSTRUCTION SET

② OPCODE / FMT / FT / FUNCT (Hex)	FOR- NAME, MNEMONIC MAT OPERATION (Hex)
Branch On FP True	bclt FI if(FPcond)PC=PC+4+BranchAddr (4) 11/8/1--
Branch On FP False	bclf FI if(!FPcond)PC=PC+4+BranchAddr(4) 11/8/0--
Divide	div R Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt] 0/-/-/1a
Divide Unsigned	divu R Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt] (6) 0/-/-/1b
FP Add Single	add.s FR F[fd] = F[fs] + F[ft] 11/10/-/0
FP Add	add.d FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]} 11/11/-/0
Double	c.xs* FR FPcond = {F[fs] op F[ft]} ? 1 : 0 11/10/-/y
FP Compare Single	c.xd* FR FPcond = ({F[fs],F[fs+1]} op {F[ft],F[ft+1]}) ? 1 : 0 11/11/-/y
FP Compare Double	* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e) 11/10/-/3
FP Divide Single	div.s FR F[fd] = F[fs] / F[ft] 11/10/-/3
FP Divide Double	div.d FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]} 11/11/-/3
FP Multiply Single	mul.s FR F[fd] = F[fs] * F[ft] 11/10/-/2
FP Multiply Double	mul.d FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]} 11/11/-/2
FP Subtract Single	sub.s FR F[fd] = F[fs] - F[ft] 11/10/-/1
FP Subtract Double	sub.d FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]} 11/11/-/1
Load FP Single	lwci I F[rt]=M[R[rs]+SignExtImm] (2) 31/-/-/
Load FP Double	ldci I F[rt]=M[R[rs]+SignExtImm]; F[rt+4]=M[R[rs]+SignExtImm+4] (2) 35/-/-/
Move From Hi	mphi R R[rd]=Hi 0/-/-/10
Move From Lo	mflo R R[rd]=Lo 0/-/-/12
Move From Control	mfc0 R R[rd]=CR[rs] 10/0/-/0
Multiply Unsigned	multu R {Hi,Lo} = R[rs] * R[rt] 0/-/-/18
Multiply	mult R {Hi,Lo} = R[rs] * R[rt] (6) 0/-/-/19
Shift Right Arith.	sra R R[rd] = R[rt] >> sham 0/-/-/3
Store FP Single	swci I M[R[rs]+SignExtImm] = F[rt] (2) 39/-/-/
Store FP Double	sdci I M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1] (2) 3d/-/-/

### FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
31	26 25	21 20	16 15	11 10	6 5	0

FI	opcode	fmt	ft	immediate
31	26 25	21 20	16 15	0

### PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

### REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

operation to separate card 2. Fold bottom side (columns 3 and 4) together

# MIPS Reference Data



## CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2) 25 <sub>hex</sub>
Load Linked	ll	I R[rt] = M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I R[rt] = {imm, 16'b0}	f <sub>hex</sub>
Load Word	lw	I R[rt] = M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor	R R[rd] = ~ (R[rs]   R[rt])	0 / 27 <sub>hex</sub>
Or	or	R R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>

NAME, MNEMONIC	FOR-MAT
Branch On FP True	bc1t FI if(
Branch On FP False	bc1f FI if(
Divide	div R Lc
Divide Unsigned	divu R Lc
FP Add Single	add.s FR F[
FP Add	add.d FR {F
Double	
FP Compare Single	c.x.s* FR FI
FP Compare Double	c.x.d* FR FI
FP Divide Single	div.s FR F[
FP Divide Double	div.d FR {F
FP Multiply Single	mul.s FR F[
FP Multiply Double	mul.d FR {F
FP Subtract Single	sub.s FR F[
FP Subtract Double	sub.d FR {F
Load FP Single	lwcl I F[
Load FP Double	ldcl I F[
Move From Hi	mfhi R R[
Move From Lo	mflo R R[
Move From Control	mfc0 R R[
Multiply	mult R {F
Multiply Unsigned	multu R {F
Shift Right Arith.	sra R R[
Store FP Single	swcl I M
Store FP Double	sdc1 I M

\* (x is eq, lt, or le) (op is ==,

FP Divide Single

FP Divide Double

FP Multiply Single

FP Multiply Double

FP Subtract Single

FP Subtract Double

Load FP Single

Load FP Double

Move From Hi

Move From Lo

Move From Control

Multiply

Multiply Unsigned

Shift Right Arith.

Store FP Single

Store FP Double

Operation to separate card 2. Fold bottom side (columns 3 and 4) together

# MIPS Reference Data



## CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2) 25 <sub>hex</sub>
Load Linked	ll	I R[rt] = M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I R[rt] = {imm, 16'b0}	f <sub>hex</sub>
Load Word	lw	I R[rt] = M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor	R R[rd] = ~ (R[rs]   R[rt])	0 / 27 <sub>hex</sub>
Or	or	R R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>

NAME, MNEMONIC	FOR-MAT
Branch On FP True	bc1t FI if(
Branch On FP False	bc1f FI if(
Divide	div R Lc
Divide Unsigned	divu R Lc
FP Add Single	add.s FR F[
FP Add	add.d FR {F
Double	
FP Compare Single	c.x.s* FR FI
FP Compare Double	c.x.d* FR FI
FP Divide Single	div.s FR F[
FP Divide Double	div.d FR {F
FP Multiply Single	mul.s FR F[
FP Multiply Double	mul.d FR {F
FP Subtract Single	sub.s FR F[
FP Subtract Double	sub.d FR {F
Load FP Single	lwcl I F[
Load FP Double	ldcl I F[
Move From Hi	mfhi R R[
Move From Lo	mflo R R[
Move From Control	mfc0 R R[
Multiply	mult R {F
Multiply Unsigned	multu R {F
Shift Right Arith.	sra R R[
Store FP Single	swcl I M
Store FP Double	sdc1 I M

\* (x is eq, lt, or le) (op is ==,

FP Divide Single

FP Divide Double

FP Multiply Single

FP Multiply Double

FP Subtract Single

FP Subtract Double

Load FP Single

Load FP Double

Move From Hi

Move From Lo

Move From Control

Multiply

Multiply Unsigned

Shift Right Arith.

Store FP Single

Store FP Double

# Exemplo de um salto condicional (2/3)

## Código MIPS:

```
Loop:    beq    $9,$0,End  
          addi   $8,$8,10  
          addi   $9,$9,-1  
          j      Loop
```

End:

```
if(R[rs]==R[rt])  
PC=PC+4+BranchAddr
```

## Campo “*Immediate*” :

Número de **instruções** a adicionar (ou subtrair) ao PC, começando na instrução **depois** do salto

Neste caso, “*immediate*” é 3

# Exemplo de um salto condicional (3/3)

## Código MIPS:

```
Loop:    beq    $9,$0,End  
          addi   $8,$8,10  
          addi   $9,$9,-1  
          j      Loop
```

End:

```
if(R[rs]==R[rt])  
PC=PC+4+BranchAddr
```

opcode	rs	rt	immediate
4	9	0	3

## Representação binária:

000100	01001	00000	0000000000000011
--------	-------	-------	------------------

# Exemplo de um salto condicional (3/3)

## Código MIPS:

```
Loop:    beq    $9,$0,End  
          addi   $8,$8,10  
          addi   $9,$9,-1  
          j      Loop
```

End:

```
if(R[rs]==R[rt])  
PC=PC+4+BranchAddr
```

opcode	rs	rt	immediate
4	9	0	3

## Representação binária:

000100	01001	00000	0000000000000011
hex			

# Exemplo de um salto condicional (3/3)

## Código MIPS:

```
Loop:    beq    $9,$0,End  
          addi   $8,$8,10  
          addi   $9,$9,-1  
          j      Loop
```

End:

```
if(R[rs]==R[rt])  
PC=PC+4+BranchAddr
```

opcode	rs	rt	immediate
4	9	0	3

## Representação binária:

000100	01001	00000	0000000000000011
--------	-------	-------	------------------

hex

hexadecimal:

1120 0003<sub>hex</sub>

## Exercício 2

### Código MIPS:

```
add $t0,$s1,$0  
Loop: addi $t0,$t0,-1  
      beq $t0,$0,Loop  
      (...)
```

## Exercício 2

### Código MIPS:

```
Loop:    add   $t0,$s1,$0
          addi  $t0,$t0,-1
          beq   $t0,$0,Loop
          (...)
```

Formato R

# Exercício 2

## Código MIPS:

```
Loop:    add   $t0,$s1,$0
          addi  $t0,$t0,-1
          beq   $t0,$0,Loop
          (...)
```

Formato R

## Representação decimal:

## Exercício 2

### Código MIPS:

```
Loop:    add   $t0,$s1,$0
          addi  $t0,$t0,-1
          beq   $t0,$0,Loop
          (...)
```

Formato R

### Representação decimal:

0	17	0	8	0	32
---	----	---	---	---	----

## Exercício 2

### Código MIPS:

```
Loop:    add   $t0,$s1,$0  
          addi  $t0,$t0,-1  
          beq   $t0,$0,Loop  
          (...)
```

Formato R

### Representação decimal:

0	17	0	8	0	32
---	----	---	---	---	----

### Representação binária:

## Exercício 2

### Código MIPS:

```
Loop:    add   $t0,$s1,$0
          addi  $t0,$t0,-1
          beq   $t0,$0,Loop
          (...)
```

Formato R

### Representação decimal:

0	17	0	8	0	32
---	----	---	---	---	----

### Representação binária:

000000	10001	00000	01000	00000	100000
--------	-------	-------	-------	-------	--------

## Exercício 2

### Código MIPS:

```
Loop:    add   $t0,$s1,$0
          addi  $t0,$t0,-1
          beq   $t0,$0,Loop
          (...)
```

Formato R

### Representação decimal:

0	17	0	8	0	32
---	----	---	---	---	----

### Representação binária:

000000	10001	00000	01000	00000	100000
hex					

## Exercício 2

### Código MIPS:

```
Loop:    add   $t0,$s1,$0
          addi  $t0,$t0,-1
          beq   $t0,$0,Loop
          (...)
```

Formato R

### Representação decimal:

0	17	0	8	0	32
---	----	---	---	---	----

### Representação binária:

000000	10001	00000	01000	00000	100000	hex
--------	-------	-------	-------	-------	--------	-----

hexadecimal:

0220 4020<sub>hex</sub>

## Exercício 2

### Código MIPS:

```
Loop:    add  $t0,$s1,$0
          addi $t0,$t0,-1
          beq  $t0,$0,Loop
          (...)
```

## Exercício 2

### Código MIPS:

```
Loop:    add  $t0,$s1,$0  
          addi $t0,$t0,-1  
          beq  $t0,$0,Loop  
          (...)
```

Formato I

## Exercício 2

### Código MIPS:

```
Loop:    add  $t0,$s1,$0  
          addi $t0,$t0,-1  
          beq  $t0,$0,Loop  
          (...)
```

Formato I

### Representação decimal:

## Exercício 2

### Código MIPS:

```
Loop:    add  $t0,$s1,$0  
          addi $t0,$t0,-1  
          beq $t0,$0,Loop  
          (...)
```

Formato I

### Representação decimal:

8	8	8		-1
---	---	---	--	----

## Exercício 2

### Código MIPS:

```
Loop:    add  $t0,$s1,$0
          addi $t0,$t0,-1
          beq  $t0,$0,Loop
          (...)
```

Formato I

### Representação decimal:

8	8	8	-1
---	---	---	----

### Representação binária:

## Exercício 2

### Código MIPS:

```
Loop:    add  $t0,$s1,$0  
          addi $t0,$t0,-1  
          beq  $t0,$0,Loop  
          (...)
```

Formato I

### Representação decimal:

8	8	8	-1
---	---	---	----

### Representação binária:

001000	01000	01000	1111111111111111
--------	-------	-------	------------------

# Exercício 2

## Código MIPS:

```
Loop:    add  $t0,$s1,$0
          addi $t0,$t0,-1
          beq  $t0,$0,Loop
          (...)
```

Formato I

## Representação decimal:

8	8	8	-1
---	---	---	----

## Representação binária:

001000	01000	01000	1111111111111111
hex			

## Exercício 2

### Código MIPS:

```
Loop: add $t0,$s1,$0  
       addi $t0,$t0,-1  
       beq $t0,$0,Loop  
       (...)
```

Formato I

### Representação decimal:

8	8	8	-1
---	---	---	----

### Representação binária:

001000	01000	01000	1111111111111111	hex
--------	-------	-------	------------------	-----

hexadecimal:

2108 FFFF<sub>hex</sub>

## Exercício 2

### Código MIPS:

```
          add  $t0,$s1,$0  
Loop:   addi $t0,$t0,-1  
          beq $t0,$0,Loop  
          (...)
```

## Exercício 2

### Código MIPS:

```
Loop:    add  $t0,$s1,$0  
          addi $t0,$t0,-1  
          beq $t0,$0,Loop  
          (...)
```

Formato I

## Exercício 2

### Código MIPS:

```
Loop:    add  $t0,$s1,$0  
          addi $t0,$t0,-1  
          beq $t0,$0,Loop  
          (...)
```

Formato I

### Representação decimal:

## Exercício 2

### Código MIPS:

```
Loop:    add  $t0,$s1,$0  
          addi $t0,$t0,-1  
          beq $t0,$0,Loop  
          (...)
```

Formato I

### Representação decimal:

4	8	0	-2
---	---	---	----

## Exercício 2

### Código MIPS:

```
Loop:    add  $t0,$s1,$0  
          addi $t0,$t0,-1  
          beq $t0,$0,Loop  
          (...)
```

Formato I

### Representação decimal:

4	8	0	-2
---	---	---	----

### Representação binária:

## Exercício 2

### Código MIPS:

```
Loop:    add  $t0,$s1,$0  
          addi $t0,$t0,-1  
          beq $t0,$0,Loop  
          (...)
```

Formato I

### Representação decimal:

4	8	0	-2
---	---	---	----

### Representação binária:

000100	01000	00000	1111111111111110
--------	-------	-------	------------------

## Exercício 2

### Código MIPS:

```
Loop:    add  $t0,$s1,$0  
          addi $t0,$t0,-1  
          beq $t0,$0,Loop  
          (...)
```

Formato I

### Representação decimal:

4	8	0	-2
---	---	---	----

### Representação binária:

000100	01000	00000	1111111111111110
--------	-------	-------	------------------

hex

## Exercício 2

### Código MIPS:

```
Loop:    add  $t0,$s1,$0  
          addi $t0,$t0,-1  
          beq $t0,$0,Loop  
          (...)
```

Formato I

### Representação decimal:

4	8	0	-2
---	---	---	----

### Representação binária:

000100	01000	00000	1111111111111110
--------	-------	-------	------------------

hex

hexadecimal:

1100 FFFE<sub>hex</sub>

# Conclusão...

- Interpretação/escrita de Código Máquina/Nativo
- Existem 3 formatos/tipos de instrução
  - Formato I
  - Formato R
  - Formato J