MIPS

Programmer's Reference Manual

Ben Roberts

Computer Laboratory
University of Cambridge

July 2007

Sponsored by:

MIPS

1 Introduction

The MIPS is a 32-bit embedded soft core processor with a five stage pipeline and a RISC instruction set. The initial version, Rhino, was designed by Robin Message and David Simner during their internship in the Summer 2006. Tiger is an upgraded version of Rhino, designed by Ben Roberts and Gregory Chadwick. It is designed to be used with Altera's Avalon bus. The 5 stage pipeline has the "standard" RISC structure:
Instruction Fetch → Decode → Execute → Memory Access → Writeback

2 Overview

2.1 Registers

The MIPS processor has 32 32-bit registers. Their names, numbers, uses, and whether the callee must preserve them across a function call are detailed in the table below:

Name	**Number**	**Use**	**Callee must preserve**
$zero	$0	constant 0	N/A
$at	$1	assembly temporary	no
$v0 - $v1	$2 - $3	function returns	no
$a0 - $a3	$4 - $7	function 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	kernel use	no
$gp	$28	global pointer	**yes**
$sp	$29	stack pointer	**yes**
$fp	$30	frame pointer	**yes**
$ra	$31	return address	N/A

The MIPS also has two special-purpose 32-bit registers, HI and LO. These are used to store the results of a division or multiplication. A multiplication of 2 32-bit numbers leaves the most significant 32 bits in HI, and the least significant 32 bits in LO. A division leaves the quotient in LO, and the remainder in HI. The instructions MFHI and MTHI allow you to move values from HI into a register, or move a value from a register into HI respectively. Similar instructions exist for LO.

2.2 Program Flow

There are 10 branch instructions: BEQ, BNE, BLEZ, BGEZ, BLTZ, BGTZ, J, JAL, JR and JALR. These all update the pc. The MIPS makes use of a branch delay slot to remove the need to flush the pipeline when a branch is taken. In other words, the instruction immediately following a branch will *always* be executed regardless of whether the branch is taken or not. If a link operation is specified, the return address is stored in $ra, so jr $ra will return from a function. Note that the return address will point to the instruction after the delay slot, so that no instructions are executed twice. For example, in the code below, 8 will be moved to $4 before the jump to three takes place. Also, 4 will be moved to $1 before the pc points to "two".

```
one:   addi $2, $0, 4   # load constant 4 into $2
       jal three        # jump to "three"
       addi $4, $0, 8   # load constant 8 into $4
two:   addi $4, $0, 6   # load constant 6 into $4
       addi $9, $0, 7   # load constant 7 into $9
       j end            # jump to "end"
       nop              # no-op
three: jr $ra           # jump to address in $ra
       addi $1, 4       # load constant 4 into $1
end:   nop              # no operation
```

2.3 Memory Access

Memory access is via. Altera's Avalon bus. Transfers may only take place when the bus is ready; this means that the processor may have to wait indefinitely.

2.4 Instruction Format

rs	Index of first operand register
rt	Index of second operand register
rd	Index of destination register
shamt	Shift amount, used only in shift operations
imm	16-bit signed immediate
addr	Memory address

R-type instruction

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

I-type instruction

31:26	25:21	20:16	15:0
opcode	rs	rt	imm

J-type instruction

31:26	25:0
opcode	addr

3 Instruction Set Quick Reference

SignImm Sign-extended immediate constant = { { 16 { imm[15] } }, imm }
ZeroImm Zero-extended immediate constant = { { 16 { 1'b0 } }, imm }
[addr] Contents stored at address addr

MIPS instruction set - sorted by opcode			
Opcode	Mnemonic	Operands	Function
0000 00	R-type	rs rt rd shamt	various - see next table
0000 01	BLTZ ($rt = 0)	rs rt imm	if ($rs < 0) PC = BTA
	BGEZ ($rt = 1)		if ($rs ≥ 0) PC = BTA
0000 10	J	addr	PC = addr
0000 11	JAL	addr	$ra = PC+4; PC = addr
0001 00	BEQ	rs rt imm	if ($rs == $rt) PC = BTA
0001 01	BNE	rs rt imm	if ($rs != $rt) PC = BTA
0001 10	BLEZ	rs rt imm	if ($rs ≤ 0) PC = BTA
0001 11	BGTZ	rs rt imm	if ($rs > 0) PC = BTA
0010 00	ADDI	rs rt imm	$rt = $rs + SignImm
0010 01	ADDIU	rs rt imm	$rt = $rs + SignImm
0010 10	SLTI	rs rt imm	$rs < SignImm ? $rt = 1 : $rt = 0
0010 11	SLTIU	rs rt imm	$rs < SignImm ? $rt = 1 : $rt = 0
0011 00	ANDI	rs rt imm	$rt = $rs & ZeroImm
0011 01	ORI	rs rt imm	$rt = $rs | ZeroImm
0011 10	XORI	rs rt imm	$rt = $rs ⊕ ZeroImm
0011 11	LUI	rs rt imm	$rt = { imm, { 16 { 1'b0 } } }
1000 00	LB	rs rt imm	$rt = { { 24 { [addr][7] } } , [addr][7:0] }
1000 01	LH	rs rt imm	$rt = { { 16 { [addr][15] } } , [addr][15:0] }
1000 11	LW	rs rt imm	$rt = [addr]
1001 00	LBU	rs rt imm	$rt = { { 24 { 1'b0 } } , [addr][7:0] }
1001 01	LHU	rs rt imm	$rt = { { 16 { 1'b0 } } , [addr][15:0] }
1010 00	SB	rs rt imm	[addr][7:0] = $rt[7:0]
1010 01	SH	rs rt imm	[addr][15:0] = $rt[15:0]
1010 11	SW	rs rt imm	[addr] = $rt
0100 00	MFC0 ($rs = 0)	rd rt	$rt = $rd
	MTC0 ($rs = 4)	rd rt	$rd = $rt
			($rd is in coprocessor 0)

R-type instructions - sorted by funct

all have operands rs, rt, rd and shamt

Funct	Mnemonic	Operation
0000 00	SLL	$rd = $rt << shamt
0000 01	SRL	$rd = $rt >> shamt
0000 11	SRA	$rd = $rt >>> shamt
0001 00	SLLV	$rd = $rt << $rs[4:0]
		assembly: sllv rd rt rs
0001 10	SRLV	$rd = $rt >> $rs[4:0]
		assembly: srlv rd rt rs
0001 11	SRAV	$rd = $rt >>> $rs[4:0]
		assembly: srav rd rt rs
0010 00	JR	PC = $rs
0010 01	JALR	$ra = PC + 4; PC = $rs
0100 00	MFHI	$rd = $hi
0100 01	MTHI	$hi = $rs
0100 10	MFLO	$rd = $lo
0100 11	MTLO	$lo = $rs
0110 00	MULT	{$hi, $lo} = ($rs × $rt)
0110 01	MULTU	{$hi, $lo} = ($rs × $rt)
0110 10	DIV	$lo = $rs / $rt
		$hi = $rs % $rt
0110 11	DIVU	$lo = $rs / $rt
		$hi = $rs % $rt
1000 00	ADD	$rd = $rs + $rt
1000 01	ADDU	$rd = $rs + $rt
1000 10	SUB	$rd = $rs − $rt
1000 11	SUBU	$rd = $rs − $rt
1001 00	AND	$rd = $rs & $rt
1001 01	OR	$rd = $rs | $rt
1001 10	XOR	$rd = $rs ⊕ $rt
1001 11	NOR	$rd = ¬ ($rs | $rt)
1010 10	SLT	$rs < $rt ? $rd = 1 : $rd = 0
1010 11	SLTU	$rs < $rt ? $rd = 1 : $rd = 0

ADD

ADD Register Signed

Type: R-Type

Operation: \$rd = \$rs + \$rt

Assembler Syntax: add rd rs rt

Example: `add $s0, $s1, $s2`

Description: Add the values contained in registers \$rs and \$rt and place the result in \$rd

Instruction Fields:

Source register 1	rs
Source register 2	rt
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
000000	rs	rt	rd	00000	100000

ADDU

ADD Register Unsigned

Type: R-Type

Operation: \$rd = \$rs + \$rt

Assembler Syntax: addu rd rs rt

Example: `addu $s0, $s1, $s2`

Description: Add the values contained in registers \$rs and \$rt and place the result in \$rd

Instruction Fields:

Source register 1	rs
Source register 2	rt
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
000000	rs	rt	rd	00000	100001

ADDI

ADD Immediate Signed

Type: I-Type

Operation: $rt = $rs + {{16{imm[15]}}, imm}

Assembler Syntax: addi rt rs imm

Example: `addi $s0, $s1, 5`

Description: Add the value contained in register $rs to the immediate constant and place the result in $rt

Instruction Fields:

Source register	rs
Immediate constant	imm
Destination register	rt

31:26	25:21	20:16	15:0
001000	rs	rt	imm

ADDIU

ADD Immediate Unsigned

Type: I-Type

Operation: $rd = $rs + {{16{imm[15]}}, imm}

Assembler Syntax: addiu rt rs imm

Example: `addiu $s0, $s1, 5`

Description: Add the value contained in register $rs to the immediate constant and place the result in $rt

Instruction Fields:

Source register	rs
Immediate constant	imm
Destination register	rt

31:26	25:21	20:16	15:0
001001	rs	rt	imm

AND

Bitwise Register AND

Type: R-Type

Operation: \$rd = \$rs & \$rt

Assembler Syntax: and rd rs rt

Example: `and $s0, $s1, $s2`

Description: Bitwise AND the contents of registers \$rs and \$rt and place the result in \$rd

Instruction Fields:

Source register 1	rs
Source register 2	rt
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
`000000`	`rs`	`rt`	`rd`	`00000`	`100100`

ANDI

Bitwise Immediate AND

Type: I-Type

Operation: \$rt = \$rs & {{16{1'b0}}, imm}

Assembler Syntax: andi rd rs rt

Example: `andi $s0, $s1, $s2`

Description: Bitwise AND the contents of register \$rs with the immediate constant zero-extended to 32-bits and place the result in \$rt

Instruction Fields:

Source register	rs
Immediate constant	imm
Destination register	rt

31:26	25:21	20:16	15:0
`001100`	`rs`	`rt`	`imm`

BEQ

Branch if Equal

Type: I-Type

Operation: if ($rs == $rt) PC = PC + 4 + {{14{addr[15]}}, addr, 2’b00}

Assembler Syntax: beq rs rt addr

Example: `beq $s0, $s1, lbl`

Description: If the contents of registers $rs and $rt are equal, then branch to addr; otherwise continue sequential execution. The instruction sequentially after the branch will always be executed due to the branch delay slot.

Instruction Fields:

Source register 1	rs
Source register 2	rt
Destination address	addr

31:26	25:21	20:16	15:0
`000100`	`rs`	`rt`	`addr`

BNE

Branch if Not Equal

Type: I-Type

Operation: if ($rs != $rt) PC = PC + 4 + {{14{addr[15]}}, addr, 2’b00}

Assembler Syntax: bne rs rt addr

Example: `bne $s0, $s1, lbl`

Description: If the contents of registers $rs and $rt are not equal, then branch to addr; otherwise continue sequential execution. The instruction sequentially after the branch will always be executed due to the branch delay slot.

Instruction Fields:

Source register 1	rs
Source register 2	rt
Destination address	addr

31:26	25:21	20:16	15:0
`000100`	`rs`	`rt`	`addr`

BGEZ

Branch if Greater Than or Equal to Zero

Type: I-Type

Operation: if ($rs $\geq$ 0) PC = PC + 4 + {{14{addr[15]}}, addr, 2'b00}

Assembler Syntax: bgez rs addr

Example: `bgez $s0, lbl`

Description: If the contents of register $rs is ≥ 0, then branch to addr; otherwise continue sequential execution. The instruction sequentially after the branch will always be executed due to the branch delay slot.

Instruction Fields:

Source register	rs
Destination address	addr

31:26	25:21	20:16	15:0
`000001`	`rs`	`00001`	`addr`

BGTZ

Branch if Greater Than Zero

Type: I-Type

Operation: if ($rs $>$ 0) PC = PC + 4 + {{14{addr[15]}}, addr, 2'b00}

Assembler Syntax: bgtz rs addr

Example: `bgtz $s0, lbl`

Description: If the contents of register $rs is > 0, then branch to addr; otherwise continue sequential execution. The instruction sequentially after the branch will always be executed due to the branch delay slot.

Instruction Fields:

Source register	rs
Destination address	addr

31:26	25:21	20:16	15:0
`000111`	`rs`	`xxxxx`	`addr`

BLEZ

Branch if Less Than or Equal to Zero

Type: I-Type

Operation: if ($rs \leq 0) PC = PC + 4 + {{14{addr[15]}}, addr, 2'b00}

Assembler Syntax: blez rs addr

Example: `blez $s0, lbl`

Description: If the contents of register $rs is ≤ 0, then branch to addr; otherwise continue sequential execution. The instruction sequentially after the branch will always be executed due to the branch delay slot.

Instruction Fields:

Source register rs
Destination address addr

31:26	25:21	20:16	15:0
000110	rs	xxxxx	addr

BLTZ

Branch if Less Than Zero

Type: I-Type

Operation: if ($rs < 0) PC = PC + 4 + {{14{addr[15]}}, addr, 2'b00}

Assembler Syntax: bltz rs addr

Example: `bltz $s0, lbl`

Description: If the contents of register $rs is < 0, then branch to addr; otherwise continue sequential execution. The instruction sequentially after the branch will always be executed due to the branch delay slot.

Instruction Fields:

Source register rs
Destination address addr

31:26	25:21	20:16	15:0
000001	rs	00000	addr

DIV

Signed Register Divide

Type: R-Type

Operation: \$lo = \$rs / \$rt
\$hi = \$rs % \$rt

Assembler Syntax: div rs rt

Example: `div $s0, $s1`

Description: Compute \$rs ÷ \$rt, store the quotient in \$lo and the remainder in \$hi

Instruction Fields: Source register 1 rs
Source register 2 rt

31:26	25:21	20:16	15:11	10:6	5:0
000000	rs	rt	xxxxx	00000	011010

DIVU

Unsigned Register Divide

Type: R-Type

Operation: \$lo = \$rs / \$rt
\$hi = \$rs % \$rt

Assembler Syntax: divu rs rt

Example: `divu $s0, $s1`

Description: Compute \$rs ÷ \$rt, store the quotient in \$lo and the remainder in \$hi. Treats \$rs and \$rt as unsigned integers

Instruction Fields: Source register 1 rs
Source register 2 rt

31:26	25:21	20:16	15:11	10:6	5:0
000000	rs	rt	xxxxx	00000	011011

J

Unconditional Jump

Type: J-Type

Operation: PC = {(PC + 4)[31:28], addr, 2'b00}

Assembler Syntax: j addr

Example: `j lbl`

Description: Jump relative to the current PC by (addr << 2)

Instruction Fields: Address addr

31:26	25:0
`000010`	`addr`

JAL

Unconditional Jump and Link

Type: J-Type

Operation: \$ra = PC + 4; PC = {(PC + 4)[31:28], addr, 2'b00}

Assembler Syntax: jal addr

Example: `jal lbl`

Description: Store PC + 4 in \$ra, then jump relative to the current PC by (addr << 2)

Instruction Fields: Address addr

31:26	25:0
`000011`	`addr`

JR

Unconditional Register Jump

Type: R-Type

Operation: PC = $rs

Assembler Syntax: jr $rs

Example: `jr $ra`

Description: Jump to address stored in $rs

Instruction Fields: Source register rs

31:26	25:21	20:16	15:11	10:6	5:0
000000	rs	xxxxx	xxxxx	00000	001000

JALR

Unconditional Register Jump and Link

Type: R-Type

Operation: $ra = PC + 4; PC = $rs

Assembler Syntax: jalr $rs

Example: `jalr $s0`

Description: Store PC + 4 in $ra then jump to address stored in $rs

Instruction Fields: Source register rs

31:26	25:21	20:16	15:11	10:6	5:0
000000	rs	xxxxx	xxxxx	00000	001001

LB

Load Byte

Type: I-Type

Operation: $rt = {{24{[Address][7]}}, [Address][7:0]}
Address = $rs + {{16{imm[15]}}, imm}

Assembler Syntax: lb rt imm(rs)

Example: `lb $s0 20($s1)`

Description: Load sign-extended lower byte of memory contents at address $rs offset by imm into $rt

Instruction Fields:

Register containing base address	rs
Offset	imm
Destination register	rt

31:26	25:21	20:16	15:0
`100000`	`rs`	`rt`	`imm`

LBU

Load Byte Unsigned

Type: I-Type

Operation: $rt = {{24{1'b0}}, [Address][7:0]}
Address = $rs + {{16{imm[15]}}, imm}

Assembler Syntax: lbu rt, imm(rs)

Example: `lbu $s0 20($s1)`

Description: Load zero-extended lower byte of memory contents at address $rs offset by imm into $rt

Instruction Fields:

Register containing base address	rs
Offset	imm
Destination register	rt

31:26	25:21	20:16	15:0
`100100`	`rs`	`rt`	`imm`

LH

Load Halfword

Type: I-Type

Operation: $rt = {{16{[Address][15]}}, [Address][15:0]}
Address = $rs + {{16{imm[15]}}, imm}

Assembler Syntax: lh rt imm(rs)

Example: `lh $s0, 20($s1)`

Description: Load sign-extended lower 2 bytes of memory contents at address $rs offset by imm into $rt

Instruction Fields:

Register containing base address	rs
Offset	imm
Destination register	rt

31:26	25:21	20:16	15:0
`100001`	`rs`	`rt`	`imm`

LHU

Load Halfword Unsigned

Type: I-Type

Operation: $rt = {{16{1'b0}}, [Address][15:0]}
Address = $rs + {{16{imm[15]}}, imm}

Assembler Syntax: lhu rt imm(rs)

Example: `lhu $s0, 20($s1)`

Description: Load zero-extended lower 2 bytes of memory contents at address $rs offset by imm into $rt

Instruction Fields:

Register containing base address	rs
Offset	imm
Destination register	rt

31:26	25:21	20:16	15:0
`100101`	`rs`	`rt`	`imm`

LW

Load Word

Type: I-Type

Operation: $rt = [Address]
Address = $rs + {{16{imm[15]}}, imm}

Assembler Syntax: lw rt imm(rs)

Example: `lw $s0, 20($s1)`

Description: Load memory contents at address $rs offset by imm into $rt

Instruction Fields:

Register containing base address	rs
Offset	imm
Destination register	rt

31:26	25:21	20:16	15:0
`100011`	`rs`	`rt`	`imm`

LUI

Load Upper Immediate

Type: I-Type

Operation: $rt = {imm, {16{1'b0}}}

Assembler Syntax: lui rt imm

Example: `lui rt, −9`

Description: Load 16-bit immediate constant into upper 16 bits of register $rt

Instruction Fields:

Immediate constant	imm
Destination register	rt

31:26	25:21	20:16	15:0
`001111`	`xxxxx`	`rt`	`imm`

MFHI

Move From HI

Type: R-Type

Operation: $rd = $hi

Assembler Syntax: mfhi rd

Example: `mfhi $s0`

Description: Copy value in register $hi into register $rd

Instruction Fields: Destination register rd

31:26	25:21	20:16	15:11	10:6	5:0
000000	xxxxx	xxxxx	rd	00000	010000

MTHI

Move To HI

Type: R-Type

Operation: $hi = $rs

Assembler Syntax: mthi rs

Example: `mthi $s2`

Description: Copy value in register $rs into register $hi

Instruction Fields: Source register rs

31:26	25:21	20:16	15:11	10:6	5:0
000000	rs	xxxxx	xxxxx	00000	010001

MFLO

Move From LO

Type: R-Type

Operation: \$rd = \$lo

Assembler Syntax: mflo rd

Example: `mflo $s0`

Description: Copy value in register \$lo into register \$rd

Instruction Fields: Destination register rd

31:26	25:21	20:16	15:11	10:6	5:0
`000000`	`xxxxx`	`xxxxx`	`rd`	`00000`	`010010`

MTLO

Move To LO

Type: R-Type

Operation: \$lo = \$rs

Assembler Syntax: mtlo rs

Example: `mtlo $s2`

Description: Copy value in register \$rs into register \$lo

Instruction Fields: Source register rs

31:26	25:21	20:16	15:11	10:6	5:0
`000000`	`rs`	`xxxxx`	`xxxxx`	`00000`	`010011`

MFC0

Move From Coprocessor 0

Type: R-Type

Operation: $rt = $rd
$rd is in coprocessor 0

Assembler Syntax: mfc0 rt rd

Example: `mfc0 $t0 $status`

Description: Copy value in coprocessor register $rd into register $rt
See the processor internals guide for further information.

Instruction Fields:

Destination register	rt
Coprocessor source register	rd

31:26	25:21	20:16	15:11	10:6	5:0
010000	00000	rt	rd	00000	000000

MTC0

Move To Coprocessor 0

Type: R-Type

Operation: $rd = $rt
$rd is in coprocessor 0

Assembler Syntax: mtc0 rt rd

Example: `mtc0 $t0 $status`

Description: Copy value in register $rt into coprocessor register $rd
See the processor internals guide for further information.

Instruction Fields:

Source register	rt
Coprocessor destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
010000	00100	rt	rd	00000	000000

MULT

Signed Register Multiply

Type: R-Type

Operation: {$hi, $lo} = ($rs × $rt)

Assembler Syntax: mult rs rt

Example: `mult $s0, $s1`

Description: Multiply the contents in register $rs by the contents in $rt, store the upper 4 bytes in $hi, and the lower 4 bytes in $lo

Instruction Fields: Source register 1 rs
Source register 2 rt

31:26	25:21	20:16	15:11	10:6	5:0
`000000`	`rs`	`rt`	`xxxxx`	`00000`	`011000`

MULTU

Unsigned Register Multiply

Type: R-Type

Operation: {$hi, $lo} = ($rs × $rt)

Assembler Syntax: multu rs rt

Example: `multu $s0, $s1`

Description: Multiply the contents in register $rs by the contents in $rt, store the upper 4 bytes in $hi, and the lower 4 bytes in $lo. Treats register contents as unsigned numbers.

Instruction Fields: Source register 1 rs
Source register 2 rt

31:26	25:21	20:16	15:11	10:6	5:0
`000000`	`rs`	`rt`	`xxxxx`	`00000`	`011001`

MUL

Signed Register Multiply without HI or LO

Type: R-Type

Operation: $\$rd = (\$rs \times \$rt)[31:0]$

Assembler Syntax: mul rd rs rt

Example: `mul $s0, $s1, $s2`

Description: Multiply the contents in register $rs by the contents in $rt, store the lower 4 bytes in $rd. The upper 4 bytes are discarded.

Instruction Fields:

Source register 1	rs
Source register 2	rt
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
011100	rs	rt	rd	00000	000010

NOP

No-operation

Type: R-Type

Operation: None

Assembler Syntax: nop

Example: `nop`

Description: Does nothing. Equivalent to `or $s0 $s0 $s0`

Instruction Fields: none

31:26	25:21	20:16	15:11	10:6	5:0
000000	00000	00000	00000	00000	000000

NOR

Bitwise Register NOR

Type: R-Type

Operation: $rd = ($rs | $rt)

Assembler Syntax: nor rd rs rt

Example: `nor $s0, $s1, $s2`

Description: Performs the bitwise OR of the contents of registers $rs and $rt, negates the answer and stores this in $rd

Instruction Fields:

Source register 1 rs
Source register 2 rt
Destination register rd

31:26	25:21	20:16	15:11	10:6	5:0
000000	rs	rt	rd	00000	100111

OR

Bitwise Register OR

Type: R-Type

Operation: \$rd = \$rs | \$rt

Assembler Syntax: or rd rs rt

Example: `or $s0, $s1, $s2`

Description: Performs the bitwise OR of the contents of registers \$rs and \$rt, and stores the answer in \$rd

Instruction Fields:

Source register 1	rs
Source register 2	rt
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
`000000`	`rs`	`rt`	`rd`	`00000`	`100101`

ORI

Bitwise Immediate OR

Type: I-Type

Operation: \$rd = \$rt | {{16{1'b0}},imm}

Assembler Syntax: ori rt rs imm

Example: `ori $s0, $s1, -5`

Description: Performs the bitwise OR of the content of the register \$rs with the zero-extended immediate constant and stores the answer in \$rt

Instruction Fields:

Source register	rs
Immediate constant	imm
Destination register	rt

31:26	25:21	20:16	15:0
`001101`	`rs`	`rt`	`imm`

SB

Store Byte

Type: I-Type

Operation: [Address][7:0] = $rt[7:0]
Address = $rs + {{16{imm[15]}}, imm}

Assembler Syntax: sb rt imm(rs)

Example: `sb $s0, 20($s1)`

Description: Stores the lower 8 bits in $rt into the lower 8 bits of the memory location at $rs offset by the sign-extended immediate constant.

Instruction Fields:

Base address register	rs
Immediate constant	imm
Source Register	rt

31:26	25:21	20:16	15:0
`101000`	`rs`	`rt`	`imm`

SH

Store Halfword

Type: I-Type

Operation: [Address][15:0] = $rt[15:0]
Address = $rs + {{16{imm[15]}}, imm}

Assembler Syntax: sh rt imm(rs)

Example: `sh $s0, 20($s1)`

Description: Stores the lower 16 bits in $rt into the lower 16 bits of the memory location at $rs offset by the sign-extended immediate constant.

Instruction Fields:

Base address register	rs
Immediate constant	imm
Source Register	rt

31:26	25:21	20:16	15:0
`101001`	`rs`	`rt`	`imm`

SW

Store Word

Type: I-Type

Operation: [Address] = $rt
Address = $rs + {{16{imm[15]}}, imm}

Assembler Syntax: sw rt imm(rs)

Example: `sw $s0, 20($s1)`

Description: Stores the contents of register $rt into the memory location at $rs offset by the sign-extended immediate constant.

Instruction Fields:

Base address register	rs
Immediate constant	imm
Source Register	rt

31:26	25:21	20:16	15:0
101011	rs	rt	imm

SLL

Shift Left Logical

Type: R-Type

Operation: \$rd = \$rt << shamt

Assembler Syntax: sll rd rt shamt

Example: `sll $s0, $s1, 2`

Description: Left shifts the contents of register \$rt by shamt, padding with zeros. The result is stored in \$rd

Instruction Fields:

Source register	rt
Shift amount	shamt
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
`000000`	`xxxxx`	`rt`	`rd`	`shamt`	`000000`

SLLV

Shift Left Logical Variable

Type: R-Type

Operation: \$rd = \$rt << \$rs[4:0]

Assembler Syntax: sllv rd rt rs
Note the change in order to normal R-Type instructions

Example: `sllv $s0, $s1, $s2`

Description: Left shifts the contents of register \$rt by the amount stored in the lower 4 bit of register \$rs, padding with zeros. The result is stored in \$rd

Instruction Fields:

Source register	rt
Shift amount register	rs
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
`000000`	`rs`	`rt`	`rd`	`00000`	`000100`

SRL

Shift Right Logical

Type: R-Type

Operation: \$rd = \$rt >> shamt

Assembler Syntax: srl rd rt shamt

Example: `srl $s0, $s1, 2`

Description: Right shifts the contents of register \$rt by shamt, padding with zeros. The result is stored in \$rd

Instruction Fields:

Source register	rt
Shift amount	shamt
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
`000000`	`xxxxx`	`rt`	`rd`	`shamt`	`000010`

SRLV

Shift Right Logical Variable

Type: R-Type

Operation: \$rd = \$rt >> \$rs[4:0]

Assembler Syntax: srlv rd rt rs
Note the change in order to normal R-Type instructions

Example: `srlv $s0, $s1, $s2`

Description: Right shifts the contents of register \$rt by the amount stored in the lower 4 bit of register \$rs, padding with zeros. The result is stored in \$rd

Instruction Fields:

Source register	rt
Shift amount register	rs
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
`000000`	`rs`	`rt`	`rd`	`00000`	`000100`

SRA

Shift Right Arithmetic

Type: R-Type

Operation: \$rd = \$rt >>> shamt

Assembler Syntax: sra rd rt shamt

Example: `sra $s0, $s1, 2`

Description: Right shifts the contents of register \$rt by shamt, padding with a copy of \$rt[31] i.e. a sign-preserving shift. The result is stored in \$rd

Instruction Fields:

Source register	rt
Shift amount	shamt
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
`000000`	`xxxxx`	`rt`	`rd`	`shamt`	`000011`

SRAV

Shift Right Arithmetic Variable

Type: R-Type

Operation: \$rd = \$rt >>> \$rs[4:0]

Assembler Syntax: srav rd rt rs
Note the change in order to normal R-Type instructions

Example: `srav $s0, $s1, $s2`

Description: Right shifts the contents of register \$rt by the amount stored in the lower 4 bit of register \$rs, padding with a copy of\$rt[31] i.e. a sign-preserving shift. The result is stored in \$rd

Instruction Fields:

Source register	rt
Shift amount register	rs
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
`000000`	`rs`	`rt`	`rd`	`00000`	`000111`

SLT — Set Less Than Register

Type: R-Type

Operation: \$rs < \$rt ? \$rd = 1 : \$rd = 0

Assembler Syntax: slt rd rs rt

Example: `slt $s0, $s1, $s2`

Description: If \$rs ¡ \$rt then \$rd is set to 0, otherwise it is set to 1.

Instruction Fields:

Source register 1	rs
Source register 2	rt
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
000000	rs	rt	rd	00000	101010

SLTU — Set Less Than Unsigned Register

Type: R-Type

Operation: \$rs < \$rt ? \$rd = 1 : \$rd = 0

Assembler Syntax: sltu rd rs rt

Example: `sltu $s0, $s1, $s2`

Description: If \$rs ¡ \$rt then \$rd is set to 0, otherwise it is set to 1. Treats operands as unsigned numbers.

Instruction Fields:

Source register 1	rs
Source register 2	rt
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
000000	rs	rt	rd	00000	101011

SLTI

Set Less Than Immediate

Type: I-Type

Operation: $rs < {{16{imm[15]}}, imm} ? $rt = 1 : $rt = 0

Assembler Syntax: slti rt rs imm

Example: `slti $s0, $s1, 9`

Description: If $rs ¡ (sign-extended immediate constant) then $rt is set to 0, otherwise it is set to 1.

Instruction Fields:

Source register	rs
Immediate constant	imm
Destination register	rt

31:26	25:21	20:16	15:0
001010	rs	rt	imm

SLTIU

Set Less Than Unsigned Immediate

Type: I-Type

Operation: $rs < {{16{imm[15]}}, imm} ? $rt = 1 : $rt = 0

Assembler Syntax: sltiu rt rs imm

Example: `sltiu $s0, $s1, 9`

Description: If $rs ¡ (sign-extended immediate constant) then $rd is set to 0, otherwise it is set to 1. Treats operand as unsigned numbers.

Instruction Fields:

Source register	rs
Immediate constant	imm
Destination register	rt

31:26	25:21	20:16	15:0
001011	rs	rt	imm

SUB

SUB Register Signed

Type: R-Type

Operation: \$rd = \$rs − \$rt

Assembler Syntax: sub rd rs rt

Example: `sub $s0, $s1, $s2`

Description: Subtract the value contained in register \$rt from the value contained in register \$rs and place the result in \$rd

Instruction Fields:

Source register 1	rs
Source register 2	rt
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
000000	rs	rt	rd	00000	100010

SUBU

SUB Register Unsigned

Type: R-Type

Operation: \$rd = \$rs − \$rt

Assembler Syntax: subu rd rs rt

Example: `subu $s0, $s1, $s2`

Description: Subtract the value contained in register \$rt from the value contained in register \$rs and place the result in \$rd

Instruction Fields:

Source register 1	rs
Source register 2	rt
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
000000	rs	rt	rd	00000	100011

XOR

Bitwise Register XOR

Type: R-Type

Operation: $\$rd = \$rs \oplus \$rt$

Assembler Syntax: xor rd rs rt

Example: `xor $s0, $s1, $s2`

Description: Performs the bitwise XOR of the contents of registers $rs and $rt, and stores the answer in $rd

Instruction Fields:

Source register 1	rs
Source register 2	rt
Destination register	rd

31:26	25:21	20:16	15:11	10:6	5:0
`000000`	`rs`	`rt`	`rd`	`00000`	`100110`

XORI

Bitwise Immediate XOR

Type: I-Type

Operation: $\$rd = \$rt \oplus \{\{16\{1'b0\}\},imm\}$

Assembler Syntax: xori rt rs imm

Example: `xori $s0, $s1, -5`

Description: Performs the bitwise XOR of the content of the register $rs with the zero-extended immediate constant and stores the answer in $rt

Instruction Fields:

Source register	rs
Immediate constant	imm
Destination register	rt

31:26	25:21	20:16	15:0
`001110`	`rs`	`rt`	`imm`