MIPS Encoding Reference

Mnemonic	Meaning	Type	Opcode		Funct	
add	Add	R	00	000000	32	100000
addu	Add Unsigned	R	00	000000	33	100001
and	Bitwise AND	R	00	000000	36	100100
div	Divide	R	00	000000	26	011010
divu	Unsigned Divide	R	00	000000	27	011011
jr	Jump to Address in Register	R	00	000000	08	001000
mfhi	Move from HI Register	R	00	000000	16	010000
mflo	Move from LO Register	R	00	000000	18	010010
mfc0	Move from Coprocessor 0	R	16	010000	NA	NA
mult	Multiply	R	00	000000	24	011000
multu	Unsigned Multiply	R	00	000000	25	011001
nor	Bitwise NOR (NOT-OR)	R	00	000000	39	100111
xor	Bitwise XOR (Exclusive-OR)	R	00	000000	38	100110
or	Bitwise OR	R	00	000000	37	100101
slt	Set to 1 if Less Than	R	00	000000	42	101010
sltu	Set to 1 if Less Than Unsigned	R	00	000000	43	101011
sll	Logical Shift Left	R	00	000000	00	000000
srl	Logical Shift Right (0-extended)	R	00	000000	02	000010
sra	Arithmetic Shift Right (sign-extended)	R	00	000000	03	000011
sub	Subtract	R	00	000000	34	100010
subu	Unsigned Subtract	R	00	000000	35	100011
j	Jump to Address	J	02	000010	NA	
jal	Jump and Link	J	03	000011	NA	
addi	Add Immediate	I	08	001000	NA	
addiu	Add Unsigned Immediate	I	09	001001	NA	
andi	Bitwise AND Immediate	I	12	001100	NA	
beq	Branch if Equal	I	04	000100	NA	
bne	Branch if Not Equal	I	05	000101	NA	
lbu	Load Byte Unsigned	I	36	100100	NA	
lhu	Load Halfword Unsigned	I	37	100101	NA	
lui	Load Upper Immediate	I	15	001111	NA	
lw	Load Word	I	35	100011	NA	
ori	Bitwise OR Immediate	I	13	001101	NA	
sb	Store Byte	I	40	101000	NA	
sh	Store Halfword	I	41	101001	NA	
slti	Set to 1 if Less Than Immediate	I	10	001010	NA	
sltiu	Set to 1 if Less Than Unsigned Immediate	I	11	001011	NA	
sw	Store Word	I	43	101011	NA	

coding format. This format has fields for specifying of up to three registers and a shift amount. For instructions that do not use all of these fields, the unused fields are coded with all 0 bits. All R-type instructions use a 000000 opcode. The operation is specified by the function field.

opcode (6)	rs (5)	rt (5)	rd (5)	sa (5)	function (6)

Alf	Instruction	Function	Opcd	Funct	Description	Numeric	Instruction	Function	Funct Hex
add	rd, rs, rt	100000	0x00	0x20	Add (with overflow)	sll	rd, rt, sa	000000	0x00
addu	rd, rs, rt	100001	0x00	0x21	Add unsigned (no overflow)	srl	rd, rt, sa	000010	0x02
and	rd, rs, rt	100100	0x00	0x24	Bitwise and	sra	rd, rt, sa	000011	0x03
break		001101	0x00	0x0D	Break (for debugging)	sllv	rd, rt, rs	000100	0x04
div	rs, rt	011010	0x00	0x1A	Divide	srlv	rd, rt, rs	000110	0x06
divu	rs, rt	011011	0x00	0x1B	Divide unsigned	srav	rd, rt, rs	000111	0x07
jalr	rd, rs	001001	0x00	0x09	Jump and link	jr	rs	001000	0x08
jr	rs	001000	0x00	0x08	Jump register	jalr	rd, rs	001001	0x09
mfhi	rd	010000	0x00	0x10	Move from HI	syscall		001100	0x0C
mflo	rd	010010	0x00	0x12	Move from LO	break		001101	0x0D
mthi	rs	010001	0x00	0x11	Move to HI	mfhi	rd	010000	0x10
mtlo	rs	010011	0x00	0x13	Move to LO	mthi	rs	010001	0x11
mult	rs, rt	011000	0x00	0x18	Multiply	mflo	rd	010010	0x12
multu	rs, rt	011001	0x00	0x19	Multiply unsigned	mtlo	rs	010011	0x13
nor	rd, rs, rt	100111	0x00	0x27	Bitwise nor	mult	rs, rt	011000	0x18
or	rd, rs, rt	100101	0x00	0x25	Bitwise or	multu	rs, rt	011001	0x19
sll	rd, rt, sa	000000	0x00	0x00	Shift left logical	div	rs, rt	011010	0x1A
sllv	rd, rt, rs	000100	0x00	0x04	Shift left logical variable	divu	rs, rt	011011	0x1B
slt	rd, rs, rt	101010	0x00	0x2A	Set on less than (signed)	add	rd, rs, rt	100000	0x20
sltu	rd, rs, rt	101011	0x00	0x2B	Set on less than unsigned	addu	rd, rs, rt	100001	0x21
sra	rd, rt, sa	000011	0x00	0x03	Shift right arithmetic	sub	rd, rs, rt	100010	0x22
srav	rd, rt, rs	000111	0x00	0x07	Shift right arithmetic variable	subu	rd, rs, rt	100011	0x23
srl	rd, rt, sa	000010	0x00	0x02	Shift right logical	and	rd, rs, rt.	100100	0x24
srlv	rd, rt, rs	000110	0x00	0x06	Shift right logical variable	or	rd, rs, rt	100101	0x25
sub	rd, rs, rt	100010	0x00	0x22	Subtract	xor	rd, rs, rt	100110	0x26
subu	rd, rs, rt	100011	0x00	0x23	Subtract unsigned	nor	rd, rs, rt	100111	0x27
syscall		001100	0x00	0x0C	System call	slt	rd, rs, rt	101010	0x2A
xor	rd, rs, rt	100110	0x00	0x26	Bitwise exclusive or	sltu	rd, rs, rt	101011	0x2B

I-Type Instructions (All opcodes except 000000, 00001x, and 0100xx)

I-type instructions have a 16-bit immediate field that codes an immediate operand, a branch target offset, or a displacement for a memory operand. For a branch target offset, the immediate field contains the signed difference between the address of the following instruction and the target label, with the two low order bits dropped. The dropped bits are always 0 since instructions are word-aligned.

For the `bgez`, `bgtz`, `blez`, and `bltz` instructions, the rt field is used as an extension of the opcode field.

opcode (6)	rs (5)	rt (5)	immediate (16)

Alf Instruction	Opcode	Notes	Opcd	Description	Numeric Instruction	Opcode	Notes	Opcd
addi rt, rs, immediate	001000		0x08	Add immediate (with overflow)	bltz rs, label	000001	rt=00000	0x01
addiu rt, rs, immediate	001001		0x09	Add immediate unsigned (no overflow)	bgez rs, label	000001	rt=00001	0x01
andi rt, rs, immediate	001100		0x0C	Bitwise and immediate	beq rs, rt, label	000100		0x04
beq rs, rt, label	000100		0x04	Branch on equal	bne rs, rt, label	000101		0x05
bgez rs, label	000001	rt=00001	0x01	Branch on greater than or equal to zero	blez rs, label	000110	rt=00000	0x06
bgtz rs, label	000111	rt=00000	0x07	Branch on greater than zero	bgtz rs, label	000111	rt=00000	0x07
blez rs, label	000110	rt=00000	0x06	Branch on less than or equal to zero	addi rt, rs, immediate	001000		0x08
bltz rs, label	000001	rt=00000	0x01	Branch on less than zero	addiu rt, rs, immediate	001001		0x09
bne rs, rt, label	000101		0x05	Branch on not equal	slti rt, rs, immediate	001010		0x0A
lb rt, immediate(rs)	100000		0x20	Load byte	sltiu rt, rs, immediate	001011		0x0B
lbu rt, immediate(rs)	100100		0x24	Load byte unsigned	andi rt, rs, immediate	001100		0x0C
lh rt, immediate(rs)	100001		0x21	Load halfword	ori rt, rs, immediate	001101		0x0D
lhu rt, immediate(rs)	100101		0x25	Load halfword unsigned	xori rt, rs, immediate	001110		0x0E
lui rt, immediate	001111		0x0F	Load upper immediate	lui rt, immediate	001111		0x0F
lw rt, immediate(rs)	100011		0x23	Load word	lb rt, immediate(rs)	100000		0x20
lwc1 rt, immediate(rs)	110001		0x31	Load word coprocessor 1	lh rt, immediate(rs)	100001		0x21
ori rt, rs, immediate	001101		0x0D	Bitwise or immediate	lw rt, immediate(rs)	100011		0x23
sb rt, immediate(rs)	101000		0x28	Store byte	lbu rt, immediate(rs)	100100		0x24
slti rt, rs, immediate	001010		0x09	Set on less than immediate (signed)	lhu rt, immediate(rs)	100101		0x25
sltiu rt, rs, immediate	001011		0x0B	Set on less than immediate unsigned	sb rt, immediate(rs)	101000		0x28
sh rt, immediate(rs)	101001		0x29	Store halfword	sh rt, immediate(rs)	101001		0x29
sw rt, immediate(rs)	101011		0x2B	Store word	sw rt, immediate(rs)	101011		0x2B
swc1 rt, immediate(rs)	111001		0x39	Store word coprocessor 1	lwc1 rt, immediate(rs)	110001		0x31
xori rt, rs, immediate	001110		0x0E	Bitwise exclusive or immediate	swc1 rt, immediate(rs)	111001		0x39

J-Type Instructions (Opcode 00001x)

The only J-type instructions are the jump instructions `j` and `jal`. These intructions require a 26-bit coded address field to specify the target of the jump. The coded address is formed from the bits at positions 27 to 2 in the binary representation of the address. The bits at positions 1 and 0 are always 0 since instructions are word-aligned.

When a J-type instruction is executed, a full 32-bit jump target address is formed by concatenating the high order four bits of the PC (the address of the instruction following the jump), the 26 bits of the target field, and two 0 bits.

opcode (6)	target (26)

Instruction		Opcode	Target	Opcd	Description
j	label	000010	coded address of label	0x02	Jump
jal	label	000011	coded address of label	0x03	Jump and link

Coprocessor Instructions (Opcode 0100xx)

The only instructions that are described here are the floating point instructions that are common to all processors in the MIPS family. All coprocessor instructions instructions use opcode 0100xx. The last two bits specify the coprocessor number. Thus all floating point instructions use opcode 010001.

The instruction is broken up into fields of the same sizes as in the R-type instruction format. However, the fields are used in different ways.

Most floating point intructions use the format field to specify a numerical coding format: single precision (.s), double precision (.d), or fixed point (.w). The `mfc1` and `mtc1` instructions uses the format field as an extension of the function field; one operand specifies a coprocessor floating point register (fx) and the other, a MIPS general purpose register (rx).

opcode (6)	format (5)	ft (5)	fs (5)	fd (5)	function (6)

Alf Instruction		Opcode	Format	Funct	Opc	Form	Func	Description	Numeric Instruction Function	Funct Hex
add.s	fd, fs, ft	010001	10000	000000	0x11	0x10	0x00	FP add single		
cvt.s.w	fd, fs	010001	10100	100000	0x11	0x14	0x20	Convert to single precision FP from integer		
cvt.w.s	fd, fs	010001	10000	100100	0x11	0x10	0x24	Convert to integer from single precision FP		
div.s	fd, fs, ft	010001	10000	000011	0x11	0x10	0x03	FP divide single		
mfc1	rd, fs	010001	00000	000000	0x11	0x00	0x00	move from coprocessor 1 (FP)		
mov.s	fd, fs	010001	10000	000110	0x11	0x10	0x06	move FP single precision FP		
mtc1	rs, fd	010001	00100	000000	0x11	0x04	0x00	move to coprocessor 1 (FP)		
mul.s	fd, fs, ft	010001	10000	000010	0x11	0x10	0x02	FP multiply single		
sub.s	fd, fs, ft	010001	10000	000001	0x11	0x10	0x01	FP subtract single		

Page URL: http://www.cs.sunysb.edu/~lw/spim/MIPSinstHex.pdf from http://www.d.umn.edu/~gshute/spimsal/talref.html
Page Author: extensions by Larry Wittie from original no-hex instruction lists by Gary Shute
Last Modified: Saturday, 18-Sep-2010 from original of Tuesday, 26-Jun-2007 12:33:40 CDT
Comments to: lw AT ic DOT sunysb DOT edu (Original instruction list author was gshute@d.umn.edu)

Instruction Encodings

Each MIPS instruction is encoded in exactly one word (32 bits). There are three encoding formats.

Register Encoding (R-Type)

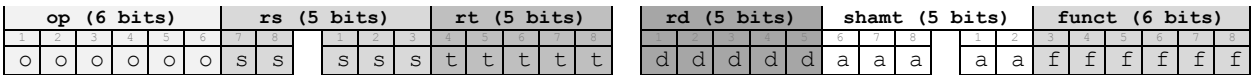
The prototypical R-type instruction is:

```
add $rd, $rs, $rt
```

The semantics of the instruction are:

```
R[d] = R[s] + R[t]
```

This encoding is used for instructions which do not require any immediate data. These instructions receive all their operands in registers. Additionally, certain of the bit shift instructions use this encoding; their operands are two registers and a 5-bit shift amount.



Field	Width	Description
o	6	Instruction opcode. This is 000000 for instructions using this encoding.
s	5	First source register, in the range 0-31.
t	5	Second source register, in the range 0-31.
d	5	Destination register, in the range 0-31.
a	5	Shift amount, for shift instructions.
f	6	Function. Determines which operation is to be performed. Values for this field are documented in the tables at the bottom of this page.

Immediate Encoding (I-Type)

The prototypical I-type instruction looks like:

```
add $rt, $rs, immed
```

The semantics of the addi instruction are;

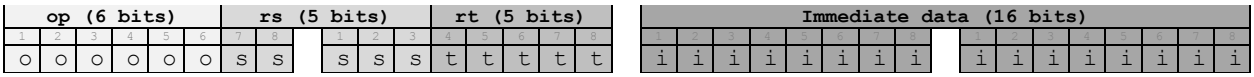
$$R[t] = R[s] + (IR_{15})^{16}\ IR_{15-0}$$

where IR refers to the instruction register, the register where the current instruction is stored. (IR15)16 means that bit B15 of the instruction register (which is the sign bit of the immediate value) is repeated 16 times. This is then followed by IR15-0, which is the 16 bits of the immediate value.

Basically, the semantics says to sign-extend the immediate value to 32 bits, add it (using signed addition) to register R[s], and store the result in register $rt.

This encoding is used for instructions which require a 16-bit immediate operand. These instructions typically receive one operand in a register, another as an immediate value coded into the instruction itself, and place their results in a register. This encoding is also used for load, store, branch, and other instructions so the use of the fields is different in some cases.

Note that the "first" and "second" registers are not always in this order in the assembly language; see "Instruction Syntax" for details.



Field	Width	Description
o	6	Instruction opcode. Determines which operation is to be performed. Values for this field are documented in the tables at the bottom of this page.
s	5	First register, in the range 0-31.
t	5	Second register, in the range 0-31.
i	16	Immediate data. These 16 bits of immediate data are interpreted differently for different instructions. 2's-complement encoding is used to represent a number between -2^{15} and $2^{15}-1$.

Jump Encoding (J-Type)

The prototypical I-type instruction looks like:

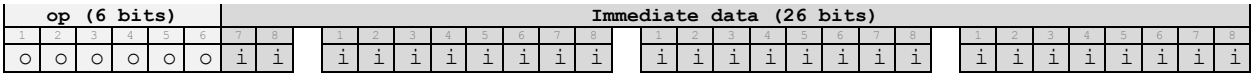
```
j target
```

The semantics of the **j** instruction (**j** means jump) are:

$$PC <- PC_{31-28}\ IR_{25-0}\ 00$$

where PC is the program counter, which stores the current address of the instruction being executed. You update the PC by using the upper 4 bits of the program counter, followed by the 26 bits of the target (which is the lower 26 bits of the instruction register), followed by two 0's, which creates a 32 bit address. The jump instruction will be explained in more detail in a future set of notes.

This encoding is used for jump instructions, which require a 26-bit immediate offset. It is also used for the trap instruction.



Field	Width	Description
o	6	Instruction opcode. Determines which operation is to be performed. Values for this field are documented in the tables at the bottom of this page.
i	26	Immediate data. These 26 bits of immediate data are interpreted differently for different instructions. 2's-complement encoding is used to represent a number between -2^{25} and $2^{25}-1$.

Instruction Syntax

This is a table of all the different types of instruction as they appear in the assembly listing. Note that each syntax is associated with exactly one encoding which is used to encode all instructions which use that syntax.

Encoding	Syntax	Template	Comments
Register	ArithLog	f $d, $s, $t	
	DivMult	f $s, $t	
	Shift	f $d, $t, a	
	ShiftV	f $d, $t, $s	
	JumpR	f $s	
	MoveFrom	f $d	
	MoveTo	f $s	
Immediate	ArithLogI	o $t, $s, i	
	LoadI	o $t, immed32	i is high or low 16 bits of immed32
	Branch	o $s, $t, label	i is calculated as (label - (current + 4)) >> 2
	BranchZ	o $s, label	i is calculated as (label - (current + 4)) >> 2
	LoadStore	o $t, i ($s)	
Jump	Jump	o label	i is calculated as (label - (current + 4)) >> 2
	Trap	o i	

Opcode Table

These tables list all of the available operations in MIPS. For each instruction, the 6-bit opcode or function is shown. The syntax column indicates which syntax is used to write the instruction in assembly text files. Note that which syntax is used for an instruction also determines which encoding is to be used. Finally the operation column describes what the operation does in pseudo-Java plus some special notation as follows:

"`MEM [`*a*`]:`*n*" means the *n* bytes of memory starting with address *a*.
The address must always be aligned; that is, *a* must be divisible by *n*, which must be a power of 2.

"`LB (`*x*`)`" means the least significant 8 bits of the 32-bit location *x*.
"`LH (`*x*`)`" means the least significant 16 bits of the 32-bit location *x*.
"`HH (`*x*`)`" means the most significant 16 bits of the 32-bit location *x*.

"`SE (`*x*`)`" means the 32-bit quantity obtained by extending the value *x* on the left with its most significant bit.
"`ZE (`*x*`)`" means the 32-bit quantity obtained by extending the value *x* on the left with 0 bits.

Arithmetic and Logical Instructions

110								
111								

Register Name	Common Name	Description
$0	zero	Always has the value 0. Any writes to this register are ignored.
$1	at	Assembler temporary.
$2-$3	v0-v1	Function result registers. Functions return integer results in v0, and 64-bit integer results in v0 and v1 when using 32-bit registers. In cases where floating-point hardware is not present, or when compiler options enable floating-point emulation, functions return single precision floating-point results in v0 and double precision floating-point results in v0 and v1 when using 32-bit registers. v0 and v1 can be temporary registers. Not preserved across function calls.
$4-$7	a0-a3	Function argument registers that hold the first four words of integer type arguments. Functions use these registers to hold floating-point arguments. When floating-point hardware is not present, or compiler options enable floating-point emulation, functions use a0 to hold the first single precision floating-point argument and a1 to hold the second single precision floating-point argument. Functions use a0-a1 for the first double precision floating-point argument, and a2-a3 to hold the second double precision floating-point argument. Not preserved across function calls.
$8-$15, $24-$25	t0-t9	Temporary registers you can use as you want. Not preserved across function calls.
$16-$23, $30	s0-s8	Saved registers to use freely. Preserved across function calls. These registers must be saved before use by the called function.
$26-$27	k0-k1	Reserved for use by the operating system kernel and for exception return.
$28	gp	Global pointer. Not used in Windows CE and may be used as save register for called functions.

$29	sp	Stack pointer.
$31	ra	Return address register, saved by the calling function. Available for use after saving.
$f0	n/a	Function return register used to return float and double values from function calls.
($f12, $f13) and ($f14, $f15)	n/a	Two pairs of registers used to pass float and double valued parameters to functions. Pairs of registers are parenthesized because they have to pass double values. To pass float values, only $f12 and $f14 are used.

The following list contains additional information on floating-point registers:

- In MIPS ISAs I, II, and in MIPS III and up ISAs running in 32-bit mode, only $f4, $f6, $f8, $f10, $f16, and $f18 temporary registers are available.
  When manipulating these registers with double precision instructions, the high-order 32-bits are in the implied odd register. The odd registers are not directly accessible.
- Permanent registers $f20, $f22, $f24, $f26, $f28, and $f30 are registers where values are preserved across function calls.
- In MIPS architectures III and up running in 64-bit mode, the following registers are also available as temporary registers: $f1, $f3, $f5, $f7, $f9, $f11, $f17, $f19, $f21, $f23, $f25, $f27, $f29, $f31.