

*Machine Language and Representations (Appendix A.9, A.10)***1. MIPS Language****1.1. Organization**

Instructions on MIPS have three different "formats". The 3 formats are R (register) format, I (immediate) format, and J (jump) format.

format	fields:length
R	Op:6 Dest:5 A:5 B:5
I	Op:6 Dest:5 A:5 IMM:16
J	Op:6 Addr:26

As indicated, the Op field of all instruction formats occurs in the same bit positions. This allows the computer hardware to decode instructions enough to decide where within the instruction the fields of that instruction lie. The numbers in these fields of the instruction are used directly by the hardware to address registers, or as data constants.

Storage is divided into a Register File R, containing 32 32-bit registers, and a much larger main memory, Mem. Both R and Mem act like arrays, in that they are referenced by integer indices. R's elements are word-size (32-bits), and are addressed using register numbers which range from 0 to 31. Mem[] however has a peculiar organization: Its elements are byte-size (8-bit) cells, which are usually accessed as words. Instructions are provided which copy data from Mem to R (Load Word (or LW), and Load byte (LB)), and from R to Mem (Store Word (SW) and Store Byte (SB)). The "byte" oriented instructions copy the single byte addressed from or to the destination register; the word oriented instructions require their address to be a multiple of 4. A word-oriented instruction addresses the n-th word of memory at location $4*n$. Such an instruction copies the 4 bytes B0, B1, B2, B3 at locations $4n+0$, $4n+1$, $4n+2$, and $4n+3$, respectively, to or from the instruction's destination register. The bytes are formed into a 32-bit word, arranged: B0 B1 B2 B3, so that B0 is the highest-order byte, and B3 the lowest.

Addresses in main memory are calculated locally by Load and Store instructions, by adding the contents of field IMM of the instruction to the contents of the register specified by the A field of the instruction. This calculation produces an "effective address e()", using the calculation:

$$e() = IMM + R[A]$$

This calculation is needed in both load and store instructions.

IMM is sign-extended before the addition, so the IMM value is treated as a 16-bit 2's complement number. As a detail, IMM is sign-extended whenever it is used in "arithmetic" immediate instructions, like ADDI and SUBI. It is NOT sign-extended when it is used in immediate "logical" operations, like ANDI and ORI.

2. Instructions

In the table below, Dest, A, and B represent register numbers, which are assembled into the instruction in the proper field. Var represents the label of some word in the .data section of the assembly language program. The assembler part of SPIM translates Var into a numeric memory address, then compiles 2 instructions to compute that address into \$at, and possibly another to actually load or store, using the form "0(\$at)" for this final instruction's address. The net effect is to load or store the contents of Mem[&Var] into or from the Dest register. For branch (Bxxx) instructions, if the form using Label is used, "Label" must be the label of some "nearby" instruction. The assembler computes $x = (&Label - PC) \gg 2$, then uses the value of x as the IMM field of the corresponding Bxx Dest,A,IMM. The net effect is to branch to "Label" if the condition tested for holds.

The column of the instruction table headed "#" gives the number of MIPS instructions this "pseudo-instruction" assembles into.

NOTE **: These instructions may print an error message, if the result "overflows" the 32 bit 2's complement representation. Variants addu, addiu, and subu ignore overflows.

The relation between the letters that can replace "cc", and the C comparison operations they represent is given by the table below:

CC	lt	eq	gt	le	ge	ne
condop(CC)	<	==	>	<=	>=	!=

Instruction definitions:

OP	operands	#	Action
add	Dest,A,B		$R[\text{Dest}] = R[A] + R[B]$ **
addi	Dest,A,IMM		$R[\text{Dest}] = R[A] + \text{IMM}$ **
sub	Dest,A,B		$R[\text{Dest}] = R[A] - R[B]$ **
slt	Dest,A,B		$R[\text{Dest}] = (R[A] < R[B])$
slti	Dest,A,IMM		$R[\text{Dest}] = (R[A] < \text{IMM})$
sCC	Dest,A,B	1-2	$R[\text{Dest}] = (R[A] \text{ condop(CC) } R[B])$
sCCi	Dest,A,IMM	1-2	$R[\text{Dest}] = (R[A] \text{ condop(CC) } \text{IMM})$
lw	Dest,IMM(A)		$R[\text{Dest}] = \text{Mem}[\text{IMM} + R[A]]$
lw	Dest,Var	3	$R[\text{Dest}] = \text{Mem}[\&\text{Var}]$
sw	Dest,IMM(A)		$\text{Mem}[\text{IMM} + R[A]] = R[\text{Dest}]$
sw	Dest,Var	3	$\text{Mem}[\&\text{Var}] = R[\text{Dest}]$
la	Dest,Var	2	$R[\text{Dest}] = \&\text{Var}$
beqz	A,IMM		if ($R[A] == 0$) goto $\text{PC} + (\text{IMM} \ll 2)$
beqz	A,Label	1	if ($R[A] == 0$) goto Label
bnez	A,IMM		if ($R[A] != 0$) goto $\text{PC} + (\text{IMM} \ll 2)$
bnez	A,Label	1	if ($R[A] != 0$) goto Label
bCC	A,B,Label	1-3	if ($R[A] \text{ condop(CC) } R[B]$) goto Label
bCC	A,IMM,Label	1-3	if ($R[A] \text{ condop(CC) } \text{IMM}$) goto Label
j	Addr		goto Addr
jr	A		goto $R[A]$
jal	Addr		$R[31] = \text{PC} + 4$; goto Addr
lui	Dest,IMM		$R[\text{Dest}] = \text{IMM} \ll 16$
or	Dest,A,B		$R[\text{Dest}] = R[A] \mid R[B]$
ori	Dest,A,IMM		$R[\text{Dest}] = R[A] \mid \text{IMM}$
and	Dest,A,B		$R[\text{Dest}] = R[A] \& R[B]$
andi	Dest,A,IMM		$R[\text{Dest}] = R[A] \& \text{IMM}$
sll	Dest,A,IMM		$R[\text{Dest}] = R[A] \ll \text{IMM}$
sllv	Dest,A,R[B]		$R[\text{Dest}] = R[A] \ll R[B]$
sra	Dest,A,IMM		$R[\text{Dest}] = R[A] \gg \text{IMM}$ (signed)
srav	Dest,A,R[B]		$R[\text{Dest}] = R[A] \gg R[B]$ (signed)
srl	Dest,A,IMM		$R[\text{Dest}] = R[A] \gg \text{IMM}$
srlv	Dest,A,R[B]		$R[\text{Dest}] = R[A] \gg R[B]$