The MAL Instruction Set

Format		Effect	Notes
`la`	R, label	R ← label	• M[i] = contents of the (aligned)
`li`	R, constant	R ← constant	word of memory beginning at location i.
`lw`	R, address	R ← M[address]	• m[i] i= contents of the byte of memory at location i.
`lb`	R, address	$R \leftarrow (m[address]_7)^{24} \Vert m[address]$	• The memory address can take several forms:
`lbu`	R, address	$R \leftarrow 0^{24} \Vert m[address]$	– label – absolute address
`sw`	R, address	R → M[address]	– (R_b) – base address
`sb`	R, address	$[R]_{7..0} \rightarrow m[address]$	– I (R_b) – base displacement
`add`	D, S_1, S_2	$D \leftarrow S_1 + S_2$	• D specifies a general register
`sub`	D, S_1, S_2	$D \leftarrow S_1 - S_2$	where the result is placed.
`mul`	D, S_1, S_2	$D \leftarrow S_1 * S_2$	
`div`	D, S_1, S_2	$D \leftarrow S_1 / S_2$ (integer division)	• S_1 is the contents of a general register.
`rem`	D, S_1, S_2	$D \leftarrow S_1 \% S_2$ (remainder)	
`and`	D, S_1, S_2	$D \leftarrow S_1$ AND S_2	• S_2 can be either the contents of a
`or`	D, S_1, S_2	$D \leftarrow S_1$ OR S_2	general register or a constant.
`xor`	D, S_1, S_2	$D \leftarrow S_1$ XOR S_2	
`nor`	D, S_1, S_2	$D \leftarrow S_1$ NOR S_2	• If S_1 is not present, then S_1 is the same as D.
`not`	D, S_1	$D \leftarrow$ NOT S_1	
`move`	D, S_2	$D \leftarrow S_2$	
`sll`	R_d, R_t, AMT	$R_d \leftarrow [R_t]_{31-AMT..0} \Vert 0^{AMT}$	• AMT may be either a general register or a constant.
`srl`	R_d, R_t, AMT	$R_d \leftarrow 0^{AMT} \Vert [R_t]_{31..AMT}$	• $0 \leq AMT < 32$.
`sra`	R_d, R_t, AMT	$R_d \leftarrow ([R_t]_{31})^{AMT} \Vert [R_t]_{31..AMT}$	
`l.s`	F, address	F ← M[address]	• F specifies a floating point register
`s.s`	F, address	F ← M[address]	where the result is placed.
`li.s`	F, constant	F ← constant	
`mov.s`	F, F_1	$F \leftarrow F_1$	• W specifies a floating point register
`add.s`	F, F_1, F_2	$D \leftarrow F_1 + F_2$	whose content is to be interpreted as
`sub.s`	F, F_1, F_2	$D \leftarrow F_1 - F_2$	a two's compliment integer.
`mul.s`	F, F_1, F_2	$D \leftarrow F_1 * F_2$	
`div.s`	F, F_1, F_2	$D \leftarrow F_1 / F_2$	• F_1, F_2, and G each specify a floating point
`cvt.s.w`	G, W	G ← W	register whose content is to be interpreted
`cvt.w.s`	W, G	W ← G	as a single-precision floating point number.
`mfc0`	R, C	R ← C	• R is a general register.
`mtc0`	R, C	R → C	• F is a floating point register.
`mfc1`	R, F	R ← F	• C is a control register.
`mtc1`	R, F	R → F	
`b`	label	PC ← label	**General Notes**
`beq`	R_s, R_t, label	if ($R_s = R_t$), then PC ← label	(1) R, R_b, R_d, and R_t are the contents of a
`bne`	R_s, R_t, label	if ($R_s \neq R_t$), then PC ← label	general register.
`blt`	R_s, R_t, label	if ($R_s < R_t$), then PC ← label	(2) $\Vert$ (parallel) indicates concatenation of bit fields.
`bgt`	R_s, R_t, label	if ($R_s > R_t$), then PC ← label	(3) Superscripts indicate repetitions of a binary value.
`ble`	R_s, R_t, label	if ($R_s \leq R_t$), then PC ← label	(4) Subscripts indicate bit positions (Little-Endian) of
`bge`	R_s, R_t, label	if ($R_s = R_t$), then PC ← label	sub-field.
`bltz`	R, label	if ($R_s < 0$), then PC ← label	
`bgtz`	R, label	if ($R_s > 0$), then PC ← label	Adapted from:
`blez`	R, label	if ($R_s \leq 0$), then PC ← label	Goodman, James, and Karen Miller.
`bgez`	R, label	if ($R_s \geq 0$), then PC ← label	*A Programmer's View of Computer Architecture.*
`bnez`	R, label	if ($R_s \neq 0$), then PC ← label	New York: Oxford UP, Incorporated, 1993. 390-91.
`beqz`	R, label	if ($R_s = 0$), then PC ← label	Layout by Joe Kohlmann
`j`	address	PC ← address	• Address may be a label or a register.
`jr`	R	PC ← R	
`jal`	address	R_{31} ← PC + 4; PC ← address	
`jalr`	R_d, R_s	R_d ← PC + 4; PC ← R_s	
`getc`	R	$R \leftarrow 0^{24} \Vert input_{7..0}$	• S may be either a general register or a label.
`putc`	R	$R_{7..0} \rightarrow$ output	• If S is a general register, effective address is
`puts`	S	Print string beginning at effective address	contents of S; if S is a label, effective address is S.