MIPS Reference Sheet

TA: Kevin Liston

There are a few special notations outlined here for reference.

Notation	**Meaning**	**Example**
{X, Y}	Concatenate the bits of X and Y together.	{10, 11, 011} = 1011011
X × Y	Repeat bit X exactly Y times.	{1, 0 × 3} = 1000
(X)[B:A]	Slice bits A through B (inclusive) out of X.	(1100110101)[4:0] = 10101
SignExt_{Nb}(X)	Sign-extend X from N bits to 32 bits.	SignExt_{4b}(1001) = {1 × 28, 1001}
Mem_{NB}(X)	Refers to the N-byte quantity in memory at byte address X.	
R[N]	Refers to the general-purpose register number N.	

Instruction Formats

There are 3 main instruction formats in MIPS. The fields in each type are laid out in such a way that the same fields are always in the same place for each type.

Type	**31 26**	**25 21**	**20 16**	**15 11**	**10 06**	**05 00**
R-Type	opcode	$rs	$rt	$rd	shamt	funct
I-Type	opcode	$rs	$rt	imm		
J-Type	opcode	address				

R-Type Instructions

These instructions are identified by an opcode of 0, and are differentiated by their funct values. Except for the first 3 shift instructions, these operations only use registers. Note that in addition to arithmetic operations, these instructions also include jumps and the system call instruction.

	Instruction	**RTL**	**Notes**
00	sll $rd, $rt, shamt	R[$rd] ← R[$rt] << shamt	
02	srl $rd, $rt, shamt	R[$rd] ← R[$rt] >> shamt	Unsigned right shift
03	sra $rd, $rt, shamt	R[$rd] ← R[$rt] >> shamt	Signed right shift
04	sllv $rd, $rt, $rs	R[$rd] ← R[$rt] << R[$rs]	

	Instruction	RTL	Notes
06	srlv $rd, $rt, $rs	R[$rd] ← R[$rt] >> R[$rs]	Unsigned right shift
07	srav $rd, $rt, $rs	R[$rd] ← R[$rt] >> R[$rs]	Signed right shift
08	jr $rs	PC ← R[$rs]	R[$rs] must be a multiple of 4
09	jalr $rd, $rs	tmp ← R[$rs] R[$rd] ← PC + 8 PC ← tmp	R[$rs] must be a multiple of 4; Undefined if $rs = $rd
09	jalr $rs	(special form of “jalr $rd, $rs” where $rd = 31, implicitly)	
12	syscall	System Call	
16	mfhi $rd	R[$rd] ← HI	
17	mthi $rs	HI ← R[$rs]	
18	mflo $rd	R[$rd] ← LO	
19	mtlo $rs	LO ← R[$rs]	
24	mult $rs, $rt	{HI, LO} ← R[$rs] * R[$rt]	Signed multiplication
25	multu $rs, $rt	{HI, LO} ← R[$rs] * R[$rt]	Unsigned multiplication
26	div $rs, $rt	LO ← R[$rs] / R[$rt] HI ← R[$rs] % R[$rt]	Signed division
27	divu $rs, $rt	LO ← R[$rs] / R[$rt] HI ← R[$rs] % R[$rt]	Unsigned division
32	add $rd, $rs, $rt	R[$rd] ← R[$rs] + R[$rt]	Exception on signed overflow
33	addu $rd, $rs, $rt	R[$rd] ← R[$rs] + R[$rt]	
34	sub $rd, $rs, $rt	R[$rd] ← R[$rs] - R[$rt]	Exception on signed overflow
35	subu $rd, $rs, $rt	R[$rd] ← R[$rs] - R[$rt]	
36	and $rd, $rs, $rt	R[$rd] ← R[$rs] & R[$rt]	
37	or $rd, $rs, $rt	R[$rd] ← R[$rs] | R[$rt]	
38	xor $rd, $rs, $rt	R[$rd] ← R[$rs] ^ R[$rt]	
39	nor $rd, $rs, $rt	R[$rd] ← !(R[$rs] | R[$rt])	
42	slt $rd, $rs, $rt	R[$rd] ← R[$rs] < R[$rt]	Signed comparison
43	sltu $rd, $rs, $rt	R[$rd] ← R[$rs] < R[$rt]	Unsigned comparison

J-Type Instructions

These instructions are identified and differentiated by their opcode numbers (2 and 3). Jump instructions use pseudo-absolute addressing, in which the upper 4 bits of the computed address are taken relatively from the program counter.

	Instruction	RTL
02	j address	PC ← {(PC + 4)[31:28], address, 00}
03	jal address	R[31] ← PC + 8 PC ← {(PC + 4)[31:28], address, 00}

I-Type Instructions

These instructions are identified and differentiated by their opcode numbers (any number greater than 3). All of these instructions feature a 16-bit immediate, which is sign-extended to a 32-bit value in every instruction (except for the and, or, and xor instructions which zero-extend and the lui instruction in which it does not matter). Branch instructions also effectively multiply the immediate by 4, to get a byte offset.

	Instruction	RTL	Notes
04	beq $rs, $rt, imm	if(R[$rs] = R[$rt]) PC ← PC + 4 + SignExt_{18b}({imm, 00})	
05	bne $rs, $rt, imm	if(R[$rs] != R[$rt]) PC ← PC + 4 + SignExt_{18b}({imm, 00})	
06	blez $rs, imm	if(R[$rs] <= 0) PC ← PC + 4 + SignExt_{18b}({imm, 00})	Signed comparison
07	bgtz $rs, imm	if(R[$rs] > 0) PC ← PC + 4 + SignExt_{18b}({imm, 00})	Signed comparison
08	addi $rt, $rs, imm	R[$rt] ← R[$rs] + SignExt_{16b}(imm)	Exception on signed overflow
09	addiu $rt, $rs, imm	R[$rt] ← R[$rs] + SignExt_{16b}(imm)	
10	slti $rt, $rs, imm	R[$rt] ← R[$rs] < SignExt_{16b}(imm)	Signed comparison
11	sltiu $rt, $rs, imm	R[$rt] ← R[$rs] < SignExt_{16b}(imm)	Unsigned comparison
12	andi $rt, $rs, imm	R[$rt] ← R[$rs] & {0 × 16, imm}	
13	ori $rt, $rs, imm	R[$rt] ← R[$rs] | {0 × 16, imm}	
14	xori $rt, $rs, imm	R[$rt] ← R[$rs] ^ {0 × 16, imm}	
15	lui $rt, imm	R[$rt] ← {(imm)[15:0], 0 × 16}	
32	lb $rt, imm($rs)	R[$rt] ← SignExt_{8b}(Mem_{1B}(R[$rs] + SignExt_{16b}(imm)))	
33	lh $rt, imm($rs)	R[$rt] ← SignExt_{16b}(Mem_{2B}(R[$rs] + SignExt_{16b}(imm)))	Computed address must be a multiple of 2

	Instruction	RTL	Notes
34	lw $rt, imm($rs)	R[$rt] ← Mem_{4B}(R[$rs] + $SignExt_{16b}$(imm))	Computed address must be a multiple of 4
36	lbu $rt, imm($rs)	R[$rt] ← {0 × 24, Mem_{1B}(R[$rs] + $SignExt_{16b}$(imm))}	
37	lhu $rt, imm($rs)	R[$rt] ← {0 × 16, Mem_{2B}(R[$rs] + $SignExt_{16b}$(imm))}	Computed address must be a multiple of 2
40	sb $rt, imm($rs)	Mem_{1B}(R[$rs] + $SignExt_{16b}$(imm)) ← (R[$rt])[7:0]	
41	sh $rt, imm($rs)	Mem_{2B}(R[$rs] + $SignExt_{16b}$(imm)) ← (R[$rt])[15:0]	Computed address must be a multiple of 2
43	sw $rt, imm($rs)	Mem_{4B}(R[$rs] + $SignExt_{16b}$(imm)) ← R[$rt]	Computed address must be a multiple of 4