MIPS Reference Sheet

31 26	25 21	20 16	15 11	10 6	5 0	
opcode	rs	rt	rd	shamt	funct	R-type
opcode	rs	rt	immediate			I-type
opcode	target					J-type

R[$x] indicates the register with address x
BMEM indicates a byte aligned access to memory
HMEM indicates a half word aligned access to memory
WMEM indicates a word aligned access to memory

Load and Store Instructions

				Name	Mnemonic	RTL Description
100000	base	dest	signed offset	Load Byte	LB rt,offset(rs)	`R[$rt] = SignEXT(BMEM[(R[$rs]+ SignEXT(imm))[31:0]])`
100001	base	dest	signed offset	Load Halfword	LH rt,offset(rs)	`R[$rt] = SignEXT(HMEM[(R[$rs]+ SignEXT(imm))[31:1]])`
100011	base	dest	signed offset	Load Word	LW rt,offset(rs)	`R[$rt] = WMEM[(R[$rs]+ SignEXT(imm))[31:2]]`
100100	base	dest	signed offset	Load Byte Unsigned	LBU rt,offset(rs)	`R[$rt] = ZeroEXT(BMEM[(R[$rs]+ SignEXT(imm))[31:0]])`
100101	base	dest	signed offset	Load Halfword Unsigned	LHU rt,offset(rs)	`R[$rt] = ZeroEXT(HMEM[(R[$rs]+ SignEXT(imm))[31:1]])`
101000	base	dest	signed offset	Store Byte	SB rt,offset(rs)	`BMEM[(R[$rs]+SignEXT(imm))[31:0]] = R[$rt][7:0]`
101001	base	dest	signed offset	Store Halfword	SH rt,offset(rs)	`HMEM[(R[$rs]+SignEXT(imm))[31:1]] = R[$rt][15:0]`
101011	base	dest	signed offset	Store Word	SW rt,offset(rs)	`WMEM[(R[$rs]+SignEXT(imm))[31:2]] = R[$rt]`

R-Type Computational Instructions

						Name	Mnemonic	RTL Description
000000	000000	src	dest	shamt	000000	Shift Left Logical	SLL rd, rt, shamt	`R[$rd] = R[$rt] << shamt`
000000	000000	src	dest	shamt	000010	Shift Right Logical	SRL rd, rt, shamt	`R[$rd] = R[$rt] >> shamt`
000000	000000	src	dest	shamt	000011	Shift Right Arithmetic	SRA rd, rt, shamt	`R[$rd] = R[$rt] >>> shamt`
000000	src1	src2	dest	00000	100000	Add (with overflow)	ADD rd, rs, rt	`R[$rd] = R[$rs] + R[$rt]`
000000	src1	src2	dest	00000	100001	Add Unsig. (no overflow)	ADDU rd, rs, rt	`R[$rd] = R[$rs] + R[$rt]`
000000	src1	src2	dest	00000	100010	Subtract	SUB rd, rs, rt	`R[$rd] = R[$rs] - R[$rt]`
000000	src1	src2	dest	00000	100011	Subtract Unsigned	SUBU rd, rs, rt	`R[$rd] = R[$rs] - R[$rt]`
000000	src1	src2	dest	00000	100100	And	AND rd, rs, rt	`R[$rd] = R[$rs] & R[$rt]`
000000	src1	src2	dest	00000	100101	Or	OR rd, rs, rt	`R[$rd] = R[$rs] | R[$rt]`
000000	src1	src2	dest	00000	100110	Xor	XOR rd, rs, rt	`R[$rd] = R[$rs] ^ R[$rt]`
000000	src1	src2	dest	00000	100111	Nor	NOR rd, rs, rt	`R[$rd] = ~R[$rs] & ~R[$rt]`
000000	src1	src2	dest	00000	101010	Set Less Than	SLT rd, rs, rt	`R[$rd] = R[$rs] < R[$rt]`
000000	src1	src2	dest	00000	101011	Set Less Than Unsig.	SLTU rd, rs, rt	`R[$rd] = R[$rs] < R[$rt]`

MIPS Reference Sheet

31–26	25–21	20–16	15–11	10–6	5–0	
opcode	rs	rt	rd	shamt	funct	R-type
opcode	rs	rt	immediate			I-type
opcode	target					J-type

R[$x] indicates the register with address x
BMEM indicates a byte aligned access to memory
HMEM indicates a half word aligned access to memory
WMEM indicates a word aligned access to memory

I-Type Computational Instructions

				Name	Mnemonic	RTL Description
001001	src	dest	signed immediate	Add Imm. Unsigned	ADDIU rt, rs, signed-imm.	`R[$rt] = R[$rs] + SignEXT(imm)`
001010	src	dest	signed immediate	Set Less Than Imm.	SLTI rt, rs, signed-imm.	`R[$rt] = R[$rs] < SignEXT(imm)`
001011	src	dest	signed immediate	Set Less Than Imm. Unsig.	SLTIU rt, rs, signed-imm.	`R[$rt] = R[$rs] < SignEXT(imm)`
001100	src	dest	zero-ext. immediate	And Immediate	ANDI rt, rs, zero-ext-imm.	`R[$rt] = R[$rs] & ZeroEXT(imm)`
001101	src	dest	zero-ext. immediate	Or Immediate	ORI rt, rs, zero-ext-imm.	`R[$rt] = R[$rs] | ZeroEXT(imm)`
001110	src	dest	zero-ext. immediate	Xor Immediate	XORI rt, rs, zero-ext-imm.	`R[$rt] = R[$rs] ^ ZeroEXT(imm)`
001111	00000	dest	zero-ext. immediate	Load Upper Imm.	LUI rt, zero-ext-imm.	`R[$rt] = {imm, 0x0000}`

Jump and Branch Instructions

						Name	Mnemonic	RTL Description
000010	target					Jump	J target	`PC = {PC[31:28], target, 00}`
000011	target					Jump and Link	JAL target	`R[31] = PC + 8;` `PC = {PC[31:28], target, 00}`
000000	src	00000	00000	00000	001000	Jump Register	JR rs	`PC = R[$rs]`
000000	src	00000	dest	00000	001001	Jump and Link Register	JALR rd, rs	`R[$rd] = PC + 8;` `PC = R[$rs]`
000100	src1	src2	signed offset			Branch On Equal	BEQ rs, rt, offset	`PC = PC + 4 + (R[$rs] == R[$rt] ? SignEXT(imm) << 2 : 0)`
000101	src1	src2	signed offset			Branch On Not Equal	BNE rs, rt, offset	`PC = PC + 4 + (R[$rs] != R[$rt] ? SignEXT(imm) << 2 : 0)`
000110	src	00000	signed offset			Branch On Less Than or Equal to Zero	BLEZ rs, offset	`PC = PC + 4 + (R[$rs] <= 0 ? SignEXT(imm) << 2 : 0)`
000111	src	00000	signed offset			Branch on Greater than Zero	BGTZ rs, offset	`PC = PC + 4 + (R[$rs] > 0 ? SignEXT(imm) << 2 : 0)`
000001	src	00000	signed offset			Branch on Less Than Zero	BLTZ rs, offset	`PC = PC + 4 + (R[$rs] < 0 ? SignEXT(imm) << 2 : 0)`
000001	src	00001	signed offset			Branch on Greater than or Equal to Zero	BGEZ rs, offset	`PC = PC + 4 + (R[$rs] >= 0 ? SignEXT(imm) << 2 : 0)`

Pseudoinstruction Set

These are simple assembly language instructions that do not have a direct machine language equivalent. During assembly, the assembler translates each psedudoinstruction into one or more machine language instructions.

Name	Mnemonic	RTL Description
Branch Less Than	BLT rs, rt, label	`if(R[$rs]<R[$rt]) PC = Label`
Branch Greater Than	BGT rs, rt, label	`if(R[$rs]>R[$rt]) PC = Label`
Branch Less Than or Equal	BLE rs, rt, label	`if(R[$rs]<=R[$rt]) PC = Label`
Branch Greater Than or Equal	BGE rs, rt, label	`if(R[$rs]>=R[$rt]) PC = Label`
Load Immediate	li rd, immediate	`R[$rd] = immediate`
Move	move rd, rs	`R[$rd] = R[$rs]`