

# MIPS Reference Data

①



## ARITHMETIC CORE INSTRUCTION SET

② OPCODE

### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add R	R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi I	R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu I	R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	addu R	R[rd] = R[rs] + R[rt]	(0) 21 <sub>hex</sub>
And	and R	R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi I	R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j J	PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal J	R[31]=PC+8; PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr R	PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu I	R[rt] = {24'b0, M[R[rs] +SignExtImm](7:0)}	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu I	R[rt] = {16'b0, M[R[rs] +SignExtImm](15:0)}	(2) 25 <sub>hex</sub>
Load Linked	ll I	R[rt] = M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui I	R[rt] = {imm, 16'b0}	f <sub>hex</sub>
Load Word	lw I	R[rt] = M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor R	R[rd] = ~ (R[rs]   R[rt])	0 / 27 <sub>hex</sub>
Or	or R	R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori I	R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>
Set Less Than Imm.	slti I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	a <sub>hex</sub>
Set Less Than Imm. Unsigned	sltiu I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6) b <sub>hex</sub>
Set Less Than Unsig.	sltu R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6) 0 / 2b <sub>hex</sub>
Shift Left Logical	sll R	R[rd] = R[rt] << shamt	0 / 00 <sub>hex</sub>
Shift Right Logical	srl R	R[rd] = R[rt] >> shamt	0 / 02 <sub>hex</sub>
Store Byte	sb I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2) 28 <sub>hex</sub>
Store Conditional	sc I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7) 38 <sub>hex</sub>
Store Halfword	sh I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2) 29 <sub>hex</sub>
Store Word	sw I	M[R[rs]+SignExtImm] = R[rt]	(2) 2b <sub>hex</sub>
Subtract	sub R	R[rd] = R[rs] - R[rt]	(1) 0 / 22 <sub>hex</sub>
Subtract Unsigned	subu R	R[rd] = R[rs] - R[rt]	0 / 23 <sub>hex</sub>

- (1) May cause overflow exception
- (2) SignExtImm = { 16{immediate[15]}, immediate }
- (3) ZeroExtImm = { 16{1b'0'}, immediate }
- (4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
- (5) JumpAddr = { PC+4[31:28], address, 2'b0 }
- (6) Operands considered unsigned numbers (vs. 2's comp.)
- (7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

### BASIC INSTRUCTION FORMATS

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

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FUNCT (Hex)
Branch On FP True	bclt FI	if(FPcond)PC=PC+4+BranchAddr	(4) 11/8/1/--
Branch On FP False	bclfi FI	if(!FPcond)PC=PC+4+BranchAddr	(4) 11/8/0/--
Divide	div R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	0/--/--/1a
Divide Unsigned	divu R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6) 0/--/--/1b
FP Add Single	add.s FR	F[fd] = F[fs] + F[ft]	11/10/--/0
FP Add Double	add.d FR	{F[fd], F[fd+1]} = {F[fs], F[fs+1]} + {F[ft], F[ft+1]}	11/11/--/0
FP Compare Single	c.x.s* FR	FPcond = (F[fs] op F[ft]) ? 1 : 0	11/10/--/y
FP Compare Double	c.x.d* FR	FPcond = ({F[fs], F[fs+1]} op {F[ft], F[ft+1]}) ? 1 : 0	11/11/--/y
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)			
FP Divide Single	div.s FR	F[fd] = F[fs] / F[ft]	11/10/--/3
FP Divide Double	div.d FR	{F[fd], F[fd+1]} = {F[fs], F[fs+1]} / {F[ft], F[ft+1]}	11/11/--/3
FP Multiply Single	mul.s FR	F[fd] = F[fs] * F[ft]	11/10/--/2
FP Multiply Double	mul.d FR	{F[fd], F[fd+1]} = {F[fs], F[fs+1]} * {F[ft], F[ft+1]}	11/11/--/2
FP Subtract Single	sub.s FR	F[fd] = F[fs] - F[ft]	11/10/--/1
FP Subtract Double	sub.d FR	{F[fd], F[fd+1]} = {F[fs], F[fs+1]} - {F[ft], F[ft+1]}	11/11/--/1
Load FP Single	lwc1 I	F[rt] = M[R[rs]+SignExtImm]	(2) 31/--/--/0
Load FP Double	ldc1 I	F[rt] = M[R[rs]+SignExtImm]; F[rt+1] = M[R[rs]+SignExtImm+4]	(2) 35/--/--/0
Move From Hi	mfhi R	R[rd] = Hi	0/--/--/10
Move From Lo	mfl0 R	R[rd] = Lo	0/--/--/12
Move From Control	mfc0 R	R[rd] = CR[rs]	10/0/--/0
Multiply	mult R	{Hi, Lo} = R[rs] * R[rt]	0/--/--/18
Multiply Unsigned	multu R	{Hi, Lo} = R[rs] * R[rt]	(6) 0/--/--/19
Shift Right Arith.	sra R	R[rd] = R[rt] >>> shamt	0/--/--/3
Store FP Single	swc1 I	M[R[rs]+SignExtImm] = F[rt]	(2) 39/--/--/0
Store FP Double	sdc1 I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2) 3d/--/--/0

### FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31	26 25	21 20	16 15	11 10	6 5
	0					
FI	opcode	fmt	ft	immediate		
	31	26 25	21 20	16 15		
	0					

### PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	b1e	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

### REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVEDACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes