MIPS64 Instruction Set (ver 3.3)

Legend:
* **double-word (64-bit)** = $b_{63}\ b_{62}\ b_{61}\ b_{60}\ ...\ b_3\ b_2\ b_1 b_0$
* **##** = concatenate
* **$\leftarrow_n$** = n-bit of data to be transferred
* **0^n** = replicate 0 n-times (zero-extend)
* **$[Mem[offset+ RS]_0]^n$** = replicate sign bit n-times (sign-extension)
* **RD, RS, RT** = General Purpose Register (Integer), RD = usually destination, RS=usually source, RT=usually another source
* **FD, FS, FT, FR** = Floating-point Register, FD=destination, FS=source, FT=another source, FR=another source
* **hi/lo** = integer multiply/divide result register
* **c0** = CPU status register / **c1** = FPU status register

***Note: the list is not comprehensive.**

- **Load-Store Instructions:**

Mnemonic	Meaning	Format	Operation
LB	Load byte (signed)	LB RD, offset(RS)	RD $\leftarrow_{64}$ Mem[offset+ RS]$_7$]^{56} ## Mem[offset+ RS]
LBU	Load byte (unsigned)	LBU RD, offset(RS)	RD $\leftarrow_{64}$ 0^{56} ## Mem[offset+ RS]
LH	Load half-word (signed)	LH RD, offset(RS)	RD $\leftarrow_{64}$ [Mem[offset+ RS]$_{15}$]^{48} ## Mem[offset+ RS]
LHU	Load half-word (unsigned)	LHU RD, offset(RS)	RD $\leftarrow_{64}$ 0^{48} ## Mem[offset+ RS]
LW	Load word (signed)	LW RD, offset(RS)	RD $\leftarrow_{64}$ [Mem[offset+RS]$_{31}$]^{32} ## Mem[offset+Regs[RS]]
LWU	Load word (unsigned)	LWU RD, offset(RS)	RD $\leftarrow_{64}$ 0^{32} ## Mem[offset+RS]
LD	Load double-word	LD RD, offset(RS)	RD $\leftarrow_{64}$ Mem[offset+RS]
SB	Store byte	SB RT, offset(RS)	Mem[offset+RS] $\leftarrow_8$ RT$_{7..0}$
SH	Store half-word	SH RT, offset(RS)	Mem[offset+RS] $\leftarrow_{16}$ RT$_{15..0}$
SW	Store word	SW RT, offset(RS)	Mem[offset+ RS] $\leftarrow_{32}$ RT$_{31..0}$
SD	Store double-word	SD RT, offset(RS)	Mem[offset+ RS] $\leftarrow_{64}$ RT$_{63..0}$
L.S	Load single-precision	L.S FD, offset(RS)	FD$\leftarrow_{64}$ 0^{32} ##Mem[offset+RS]
L.D	Load double-precision	L.D FD, offset(RS)	FD $\leftarrow_{64}$ Mem[offset+RS]
LUI	Load upper immediate	LUI RD, imm	RD $\leftarrow$ (imm$_{31}$)^{32} ## imm ## 0^{16}
S.S	Store single-precision	S.S FT, offset(RS)	Mem[offset+RS] $\leftarrow_{32}$ FT$_{31.0}$
S.D	Store double-precision	S.D FT, offset(RS)	Mem[offset+RS] $\leftarrow_{64}$ FT$_{63..0}$

- **Arithmetic Instructions**

Mnemonic	Meaning	Format	Operation
DADDU	Double-word addition	DADDU RD, RS, RT	RD $\leftarrow$ RS + RT
DADDIU	Double-word add w/immediate (signed constant)	DADDIU RD, RS, imm	RD $\leftarrow$ RS + imm
DSUBU	Double-word subtraction	DSUBU RD, RS, RT	RD $\leftarrow$ RS – RT
DMULT	Double-word multiplication (signed)	DMULT RS,RT	hi/lo $\leftarrow$ RS*RT
DMULTU	Double-word multiplication (unsigned)	DMULTU RS,RT	hi/lo $\leftarrow$ RS*RT
DDIV	Double-word division (signed)	DDIV RS, RT	lo = RS div RT; hi = RS mod RT
DDIVU	Double-word division (unsigned)	DDIVU RS, RT	lo = RS div RT; hi = RS mod RT
MADD	Multiply-Accumulate	MADD RS, RT	hilo = hilo + (RS*RT)

- **Special move Instructions:**

Mnemonic	Meaning	Format	Operation
MFHI	Move from hi	MFHI RD	RD ←hi
MFLO	Move from lo	MFLO RD	RD ←lo
MTHI	Move to hi	MTHI RS	hi ← RS
MTLO	Move to lo	MTLO RS	lo ← RS
MFC0	Move from C0 to RD (32-bit)	MFC0 RD, CS	RD ← CS
MTC0	Move to C0 from RS (32-bit)	MTCO RS, CD	CD ← RS
DMFC0	Move from C0 to RD (64-bit)	DMFC0 RD, CS	RD ← CS
DMTC0	Move to C0 from RS (64-bit)	DMTCO RS, CD	CD ← RS
MFC1	Move from FPR to GPR (32-bit)	MFC1 RD, FS	RD ←FS
MTC1	Move to FPR from GPR (32-bit)	MTC1 RS, FD	FD ←RS
DMFC1	Move from FPR to GPR (64-bit)	DMFC1 RD, FS	RD ←FS
DMTC1	Move to FPR from GPR (64-bit)	DMTC1 RS, FD	FD ←RS
MOV.S	Move from one Single Precision FPR to another	MOV.S FD, FS	FD ← FS
MOV.D	Move from one Double Precision FPR to another	MOV.D FD, FS	FD ←FS

- **Logical Instructions**

Mnemonic	Meaning	Format	Operation
AND	Logical AND	AND RD, RS, RT	RD ← RS • RT
OR	Logical OR	OR RD, RS, RT	RD ← RS | RT
XOR	Logical XOR	XOR RD, RS, RT	RD ← RS ⊕ RT
NOR	Logical NOR	NOR RD, RS, RT	RD ← ~ (RS | RT)
ANDI	Logical AND with immediate	ANDI RD, RS, imm	RD ← RS • imm
ORI	Logical OR with immediate	ORI RD, RS, imm	RD ← RS | imm
XORI	Logical XOR with immediate	XORI RD, RS, imm	RD ← RS ⊕ imm

- **Shift Instructions**

Mnemonic	Meaning	Format	Operation
DSLLV	Double-word shift left logical variable	DSLLV RD, RS, RT	RD ← RS << $RT_{5..0}$ (only the lower 6 bits of RT is considered) Shift logical left: as bits shift left, the LSB of the register is filled with zero
DSRLV	Double-word shift right logical variable	DSLRV RD, RS, RT	RD ← RS >> $RT_{5..0}$ (only the lower 6 bits of RT is considered) Shift logical right: as bits shift right, the MSB of the register is filled with zero
DSRAV	Double-word shift right arithmetic variable	DSLRV RD, RS, RT	RD ← RS >> $RT_{5..0}$ (only the lower 6 bits of RT is considered) Shift arithmetic right: as bits shift right, the MSB of the register is filled with sign bit
DSLL	Double-word shift left logical	DSLL RD, RS, imm	RD ← RS << $imm_{4..0}$ (only the low 5 bits of imm is considered) Shift logical left: as bits shift left, the LSB of the register is filled with zero
DSRL	Double-word shift right logical	DSRL RD, RS, imm	RD ← RS >> $imm_{4..0}$ (only the low 5 bits of imm is considered) Shift logical right: as bits shift right, the MSB of the register is filled with zero
DSRA	Double-word shift right arithmetic	DSRA RD, RS, imm	RD ← RS >> $imm_{4..0}$ (only the low 5 bits of imm is considered) Shift arithmetic right: as bits shift right, the MSB of the register is filled with sign bit

- **Control & Transfer Instructions**

J	Unconditional Jump	J Label	PC ← $PC_{63..28}$ ##label ##0^2 Internally, Label (which is 26-bit) is left shifted twice and then replaces the lower 28 bits of the PC. Thus, Label can be viewed as target address div 4
JR	Unconditional Jump	JR RS	PC ← RS
JAL	Jump and link ("call")	JAL Label	R31 ← PC+4 (no delay branch slot) R31 ← PC+8 (with delay branch slot) PC ← $PC_{63..28}$ ##label ##0^2 Internally, Label (which is 26-bit) is left shifted twice and then replaces the lower 28 bits of the PC. Thus, Label can be viewed as target address div 4
JALR	Jump and Link ("call")	JALR RS	R31 ← PC+4 (no delay branch slot) R31 ← PC+8 (with delay branch slot) PC ← RS
BLEZ BLTZ BGEZ BGTZ	Branch if less than or equal to zero Branch if less than zero Branch if greater than or equal to zero Branch if greater than zero	BLEZ RS, Label (RS<=0)* BLTZ RS, Label (RS<0)* BGEZ RS, Label (RS>=0)* BGTZ RS, Label (RS>0)*	If (cond)* then PC ← PC + sign_extend(Label##0^2) Internally, label (which is 16-bit signed offset) is left shifted twice and then added to the PC+4. Thus, the offset can also be viewed as the distance of instruction to a label. The next instruction after the branch is viewed as offset 0
BEQ	Branch if equal	BEQ RS, RT, Label	If (RS=RT) then PC ← PC + sign_extend(Label##0^2) Internally, label (which is 16-bit signed offset) is left shifted twice and then added to the PC+4. Thus, the offset can also be viewed as the distance of instruction to a label. The next instruction after the branch is viewed as offset 0
BNE	Branch if not equal	BNE RS, RT, Label	If (RS<>RT) then PC ← PC + sign_extend(Label##0^2) Internally, label (which is 16-bit signed offset) is left shifted twice and then added to the PC+4. Thus, the offset can also be viewed as the distance of instruction to a label. The next instruction after the branch is viewed as offset 0
MOVZ	Conditional Move if zero	MOVZ RD, RS, RT	If (RT = 0) RD ← RS
MOVN	Conditional Move if Not zero	MOVN RD, RS, RT	If (RT <> 0) RD ← RS

- **Set-on-condition Instructions**

Mnemonic	Meaning	Format	Operation
SLT	Set if less than (Signed)	SLT RD, RS, RT	RD ← RS < RT ?1 :0
SLTU	Set if less than (Unsigned)	SLTU RD, RS, RT	RD ← RS < RT ?1 :0
SLTI	Set if less than immediate (Signed)	SLTI RD, RS, imm	RD ← RS < imm ?1 :0
SLTIU	Set if less than immediate (Unsigned)	SLTIU RD, RS, imm	RD ← RS < imm ?1 :0

- **Floating Point Instructions**

Mnemonic	Meaning	Format	Operation
ADD.D	Add double precision	ADD.D FD, FS, FT	FD ← FS + FT
ADD.S	Add single precision	ADD.S FD, FS, FT	FD ← FS + FT
ADD.PS	Add pair of single precision	ADD.PS FD, FS, FT	$FD_{0..31} \leftarrow FS_{0..31} + FT_{0..31}$ $FD_{32..63} \leftarrow FS_{32..63} + FT_{32..63}$
SUB.D	Subtract double precision	SUB.D FD, FS, FT	FD ← FS – FT
SUB.S	Subtract single precision	SUB.S FD, FS, FT	FD ← FS – FT
SUB.PS	Subtract pair of single precision	SUB.PS FD, FS, FT	$FD_{0..31} \leftarrow FS_{0..31} - FT_{0..31}$ $FD_{32..63} \leftarrow FS_{32..63} - FT_{32..63}$
MUL.D	Multiply double precision	MUL.D FD, FS, FT	FD ← FS * FT
MUL.S	Multiply single precision	MUL.S FD, FS, FT	FD ← FS * FT
MUL.PS	Multiply pair of single precision	MUL.PS FD, FS, FT	$FD_{0..31} \leftarrow FS_{0..31} * FT_{0..31}$ $FD_{32..63} \leftarrow FS_{32..63} * FT_{32..63}$
DIV.D	Divide double precision	DIV.D FD, FS, FT	FD ← FS / FT
DIV.S	Divide single precision	DIV.S FD, FS, FT	FD ← FS / FT
DIV.PS	Divide pair of single precision	DIV.PS FD, FS, FT	$FD_{0..31} \leftarrow FS_{0..31} / FT_{0..31}$ $FD_{32..63} \leftarrow FS_{32..63} / FT_{32..63}$
MADD.D	Multiply-ADD double precision	MADD.D FD, FR, FS, FT	FD ← FR + (FS * FT)
MADD.S	Multiply-ADD single precision	MADD.S FD, FR, FS, FT	FD ← FR + (FS * FT)
MADD.PS	Multiply-ADD pair of single precision	MADD.PS FD, FR, FS, FT	$FD_{0..31} \leftarrow FR_{0..31} + (FS_{0..31} * FT_{0..31})$ $FD_{32..63} \leftarrow FS_{32..63} + (FS_{32..63} * FT_{32..63})$
CVT.d.s	Convert from type y to type x; where x & y can be **s** (SP floating point), **d** (DP floating point), **w** (32-bit integer), **l** (64-bit integer)	CVT.x.y FD, FS	FD ←FS
C._.S	Set floating point status register (FCC) if condition is _; where "_" can be LT, LE, GT, GE, EQ, NE Default is 0 (0-7). For Single Precision comparison	C._.S FS, FT C._.S cc, FS, FT	FCC(cc) = FS < FT ? 1 : 0
C._.D	Set floating point status register (FCC) if condition is _; where "_" can be LT, LE, GT, GE, EQ, NE Default is 0 (0-7). For Double Precision comparison	C._.D FS, FT C._.D cc, FS, FT	FCC(CC) = FS < FT ? 1 : 0

Mnemonic	Meaning	Format	Operation
BC1T	Branch if FP status register is 1	BC1T Label BC1T cc, Label	If FCC(cc) = 1 then PC ← PC+Label Internally, label (which is 16-bit signed offset) is left shifted twice and then added to the PC+4. Thus, the offset can also be viewed as the distance of instruction to a label. The next instruction after the branch is viewed as offset 0
BC1F	Branch if FP status register is 0	BC1F Label	If FCC(cc) = 0 then PC ← PC+Label Internally, label (which is 16-bit signed offset) is left shifted twice and then added to the PC+4. Thus, the offset can also be viewed as the distance of instruction to a label. The next instruction after the branch is viewed as offset 0

- **Special Instructions**

Mnemonic	Meaning	Format	Operation
TRAP	Transfer to operating at a vectored address	TRAP	
ERET	Return to user code from an exception; restore user mode	ERET	
NOP	No Operation	NOP	