MIPS64 Instruction Set and Format (v1.2)

Note: This list is not comprehensive.

***Introduced in MIPS64 (Release 6) ^Deprecated in MIPS64 (Release 6)**

J-Type (6-26)

	Category	Instruction	Meaning	Format	Operation	OpCode (6)	Variable (26)
*	Control Transfer Instruction	BALC	Branch and link, compact ("call")	BALC offset	GPR[31] ← PC+4* (no delay branch slot) PC ← PC + sign_extend(offset<<2) *GPR[31] ←address of the next instruction Internally, offset (which is 26-bit signed offset) is left shifted twice and then added to the address of the instruction following the branch. Thus, the offset can also be viewed as the distance of instruction to the label. The next instruction after the branch is viewed as offset 0.	111010	offset
*	Control Transfer Instruction	BC	Unconditional branch, compact	BC offset	PC ← PC + sign_extend(offset<<2) Internally, offset (which is 26-bit signed offset) is left shifted twice and then added to the address of the instruction following the branch. Thus, the offset can also be viewed as the distance of instruction to the label. The next instruction after the branch is viewed as offset 0.	110010	offset
^	Control Transfer Instruction	J	Unconditional Jump	J target	PC ← PC63..28 | | (instr_index << 2) Internally, instr_label (which is 26-bit) is left shifted twice and then replaces the lower 28 bits of the PC. Thus, instr_index can be viewed as target address div 4	000010	instruction index
^	Control Transfer Instruction	JAL	Jump and link ("call")	JAL target	GPR[31] ← PC+8* (with branch delay slot) PC ← PC63..28 | | (instr_index << 2) *GPR[31] ←address of the next instruction after the branch delay slot Internally, instr_label (which is 26-bit) is left shifted twice and then replaces the lower 28 bits of the PC. Thus, instr_index can be viewed as target address div 4	000011	instruction index

I-Type (6-5-21)

0	Category	Instruction	Meaning	Format	Operation	OpCode (6)	rs(5)	variable(21)
*	Control Transfer Instruction	BEQZC	Compact branch if GPR rs is equal to zero	BEQZC rs, offset	if GPR[rs] = 0 then PC ← (PC+ sign_extend(offset<<2))	110110	rs != r0	offset
*	Control Transfer Instruction	BNEZC	Compact branch if GPR rs is not equal to zero	BNEZC rs, offset	if GPR[rs] != 0 then PC ← (PC+ sign_extend(offset<<2))	111110	rs != r0	offset

I-Type (6-5-5-16)

0	Category	Instruction	Meaning	Format	Operation	OpCode (6)	rs(5)	rt(5)	variable(16)
	Arithmetic Instruction	DADDIU	Double-word add w/immediate	DADDIU rt, rs, immediate	GPR[rt] ← GPR[rs] + immediate	011001	rs	rt	immediate
	Logical Instruction	ANDI	Logical AND with immediate	ANDI rt, rs, immediate	GPR[rt] ← GPR[rs] & immediate	001100	rs	rt	immediate
	Logical Instruction	ORI	Logical OR with immediate	ORI rt, rs, immediate	GPR[rt] ← GPR[rs] | immediate	001101	rs	rt	immediate
	Logical Instruction	XORI	Logical XOR with immediate	XORI rt, rs, immediate	GPR[rt] ← GPR[rs] XOR immediate	001110	rs	rt	immediate
	Set-on Condition Instruction	SLTI	Set if less than immediate (Signed)	SLTI rt, rs, immediate	GPR[rt] ← (GPR[rs] < immediate)	001010	rs	rt	immediate
	Set-on Condition Instruction	SLTIU	Set if less than immediate (Unsigned)	SLTIU rt, rs, immediate	GPR[rt] ← (GPR[rs] < immediate)	001011	rs	rt	immediate
*	Arithmetic Instruction	AUI	Add Upper Immediate	AUI rt, rs immediate	GPR[rt] ← sign_extend.32(GPR[rs] + sign_extend(immediate << 16))	001111	rs != r0	rt	immediate
*	Arithmetic Instruction	DAUI	Doubleword Add Upper Immediate	DAUI rt, rs,immediate	GPR[rt] ← GPR[rs] + sign_extend(immediate << 16)	011101	rs != r0	rt	immediate
*	Arithmetic Instruction	DAHI	Doubleword Add Higher Immediate	DAHI rs, rs, immediate	GPR[rs] ← GPR[rs] + sign_extend(immediate << 32)	000001	rs	00110	immediate
*	Arithmetic Instruction	DATI	Doubleword Add Top Immediate	DATI rs, rs, immediate	GPR[rs] ← GPR[rs] + sign_extend(immediate << 48)	000001	rs	11110	immediate
	Load-Store Instruction	LB	Load byte (signed)	LB rt, offset(base)	GPR[rt] ← sign_extend(memory[GPR[base] + offset])	100000	base	rt	offset
	Load-Store Instruction	LBU	Load byte (unsigned)	LBU rt, offset(base)	GPR[rt] ← zero_extend(memory[GPR[base] + offset])	100100	base	rt	offset
	Load-Store Instruction	LD	Load double-word	LD rt, offset(base)	GPR[rt] ← memory[GPR[base] + offset]	110111	base	rt	offset
	Load-Store Instruction	L.D	Load double-precision	L.D ft, offset(base)	FPR[ft] ← memory[GPR[base] + offset]	110101	base	ft	offset
	Load-Store Instruction	LDC1	Load double-precision	LDC1 ft, offset(base)	FPR[ft] ← memory[GPR[base] + offset]	110101	base	ft	offset
	Load-Store Instruction	LH	Load half-word (signed)	LH rt, offset(base)	GPR[rt] ← sign_extend(memory[GPR[base] + offset])	100001	base	rt	offset
	Load-Store Instruction	LHU	Load half-word (unsigned)	LHU rt, offset(base)	GPR[rt] ← zero_extend(memory[GPR[base] + offset])	100101	base	rt	offset
	Load-Store Instruction	LW	Load word (signed)	LW rt, offset(base)	GPR[rt] ← sign_extend(memory[GPR[base] + offset])	100011	base	rt	offset
	Load-Store Instruction	L.S	Load single-precision	L.S ft, offset(base)	FPR[ft] ← memory[GPR[base] + offset]	100011	base	rt	offset
	Load-Store Instruction	LWC1	Load single-precision	LWC1 ft, offset(base)	FPR[ft] ← memory[GPR[base] + offset]	110001	base	ft	offset
	Load-Store Instruction	LWU	Load word (unsigned)	LWU rt, offset(base)	GPR[rt] ← zero_extend(memory[GPR[base] + offset])	100111	base	rt	offset
	Load-Store Instruction	LUI	Load upper immediate	LUI rt, immediate	GPR[rt] ← sign_extend(immediate || 0^{16})	001111	00000	rt	immediate
	Load-Store Instruction	SB	Store byte	SB rt, offset(base)	memory[GPR[base] + offset] ← GPR[rt]	101000	base	rt	offset
	Load-Store Instruction	SD	Store double-word	SD rt, offset(base)	memory[GPR[base] + offset] ← GPR[rt]	111111	base	rt	offset
	Load-Store Instruction	S.D	Store double-precision	S.D ft, offset(base)	memory[GPR[base] + offset] ← FPR[ft]	111101	base	ft	offset
	Load-Store Instruction	SDC1	Store double-precision	SDC1 ft, offset(base)	memory[GPR[base] + offset] ← FPR[ft]	111101	base	ft	offset
	Load-Store Instruction	SH	Store half-word	SH rt, offset(base)	memory[GPR[base] + offset] ← GPR[rt]	101001	base	rt	offset
	Load-Store Instruction	SW	Store word	SW rt, offset(base)	memory[GPR[base] + offset] ← GPR[rt]	101011	base	rt	offset

	Load-Store Instruction	S.S	Store single-precision	S.S ft, offset(base)	memory[GPR[base] + offset] ← FPR[ft]	111001	base	ft	offset
	Load-Store Instruction	SWC1	Store single-precision	SWC1 ft, offset(base)	memory[GPR[base] + offset] ← FPR[ft]	111001	base	ft	offset
	Control Transfer Instruction	B	Unconditional Branch	B offset	PC ← PC + sign_extend(offset<<2) Internally, offset (which is 26-bit signed offset) is left shifted twice and then added to the address of the instruction following the branch. Thus, the offset can also be viewed as the distance of instruction to the label. The next instruction after the branch is viewed as offset 0.	000100	00000	00000	offset
	Control Transfer Instruction	BAL	Branch and Link	BAL offset	GPR[31] ← PC+8* (with delay branch slot) PC ← PC + sign_extend(offset<<2) *GPR[31] ←address of the next instruction after the branch delay slot Internally, offset (which is 26-bit signed offset) is left shifted twice and then added to the address of the instruction following the branch. Thus, the offset can also be viewed as the distance of instruction to the label. The next instruction after the branch is viewed as offset 0.	000001	00000	10001	offset
	Control Transfer Instruction	BEQ	Branch if equal	BEQ rs, rt, offset	If GPR[rs]=GPR[rt] then PC ← PC + sign_extend(offset<<2) Internally, offset (which is 16-bit signed offset) is left shifted twice and then added to the address of the instruction following the branch. 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.	000100	rs	rt	offset
*	Control Transfer Instruction	BEQC	Branch if equal Compact	BEQC rs,rt, offset	If GPR[rs]=GPR[rt] then PC ← PC + sign_extend(offset<<2) Internally, offset (which is 16-bit signed offset) is left shifted twice and then added to the address of the instruction following the branch. Thus, the offset can also be viewed as the distance of instruction to the label. The next instruction after the branch is viewed as offset 0.	001000	rs < rt	rt/rs != r0	offset
*	Control Transfer Instruction	BNEC	Branch if not equal Compact	BNEC rs,rt, offset	If GPR[rs]<>GPR[rt] then PC ← PC + sign_extend(offset<<2) Internally, offset (which is 16-bit signed offset) is left shifted twice and then added to the address of the instruction following the branch. Thus, the offset can also be viewed as the distance of instruction to the label. The next instruction after the branch is viewed as offset 0.	011000	rs < rt	rt/rs != r0	offset
	Control Transfer Instruction	BNE	Branch if not equal	BNE rs, rt, offset	If GPR[rs] <> GPR[rt] then PC ← PC + sign_extend(offset<<2) Internally, offset (which is 16-bit signed offset) is left shifted twice and then added to the address of the instruction following the branch. 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.	000101	rs	rt	offset
	Control Transfer Instruction	BGEZ	Branch if greater than or equal to zero	BGEZ rs, offset	if GPR[rs] >= 0 then PC ← (PC+ sign_extend(offset<<2))	000001	rs	00001	offset
	Control Transfer Instruction	BGTZ	Branch if greater than zero	BGTZ rs, offset	if GPR[rs] > 0 then PC ← (PC+ sign_extend(offset<<2))	000111	rs	00000	offset
	Control Transfer Instruction	BLEZ	Branch if less than or equal to zero	BLEZ rs, offset	if GPR[rs] <= 0 then PC ← (PC+ sign_extend(offset<<2))	000110	rs	00000	offset
	Control Transfer Instruction	BLTZ	Branch if less than zero	BLTZ rs, offset	if GPR[rs] < 0 then PC ← (PC+ sign_extend(offset<<2))	000001	rs	00000	offset
*	Control Transfer Instruction	BLTZC	Branch if less than zero compact	BLTZC rt, offset	if GPR[rt] < 0 then PC ← (PC+ sign_extend(offset<<2))	010111	rs = rt	rt != r0	offset
*	Control Transfer Instruction	BGEZC	Branch if greater than or equal to zero compact	BGEZC rt, offset	if GPR[rt] >= 0 then PC ← (PC+ sign_extend(offset<<2))	010110	rs = rt	rt != r0	offset
*	Control Transfer Instruction	BLEZC	Branch if less than or equal to zero compact	BLEZC rt, offset	if GPR[rt] <= 0 then PC ← (PC+ sign_extend(offset<<2))	010110	00000	rt != r0	offset
*	Control Transfer Instruction	BGTZC	Branch if greater than zero compact	BGTZC rt, offset	if GPR[rt] > 0 then PC ← (PC+ sign_extend(offset<<2))	010111	00000	rt != r0	offset
*	Control Transfer Instruction	BLTC	Compact branch if GPR rs is less than GPR rt	BLTC rs,rt, offset	if GPR[rs] < GPR[rt] then PC ← (PC+ sign_extend(offset<<2))	010111	rs != rt	rt != r0	offset
*	Control Transfer Instruction	BGTC	Compact branch if GPR rt is greater than GPR rs (alias for BLTC)	BGTC rt,rs, offset	if GPR[rt] > GPR[rs] then PC ← (PC+ sign_extend(offset<<2))	010111	rt != rs	rs != r0	offset
*	Control Transfer Instruction	BGEC	Compact branch if GPR rs is greater than or equal to GPR rt	BGEC rs,rt, offset	if GPR[rs] >= GPR[rt] then PC ← (PC+ sign_extend(offset<<2))	010110	rs != rt	rt != r0	offset
*	Control Transfer Instruction	BLEC	Compact branch if GPR rt is less than or equal to GPR rs (alias for BGEC)	BLEC rt,rs, offset	if GPR[rt] <= GPR[rs] then PC ← (PC+ sign_extend(offset<<2))	010110	rt != rs	rs != r0	offset
*	Control Transfer Instruction	BLTUC	Compact branch if GPR rs is less than GPR rt, unsigned	BLTUC rs,rt, offset	if unsigned(GPR[rs]) < unsigned(GPR[rt]) then PC ← (PC+ sign_extend(offset<<2))	000111	rs != rt	rt != r0	offset
*	Control Transfer Instruction	BGTUC	Compact branch if GPR rt is greater than GPR rs, unsigned (alias for BLTUC)	BGTUC rt,rs, offset	if unsigned(GPR[rt]) > unsigned(GPR[rs]) then PC ← (PC+ sign_extend(offset<<2))	000111	rt != rs	rs != r0	offset
*	Control Transfer Instruction	BGEUC	Compact branch if GPR rs is greater than or equal to GPR rt, unsigned	BGEUC rs,rt, offset	if unsigned(GPR[rs]) >= unsigned(GPR[rt]) then PC ← (PC+ sign_extend(offset<<2))	000110	rs != rt	rt != r0	offset
*	Control Transfer Instruction	BLEUC	Compact branch if GPR rt is less than or equal to GPR rt, unsigned (alias for BGEUC)	BLEUC rt,rs, offset	if unsigned(GPR[rt]) <= unsigned(GPR[rs]) then PC ← (PC+ sign_extend(offset<<2))	000110	rt != rs	rs != r0	offset
*	Control Transfer Instruction	BC1EQZ	Branch if Coprocessor 1 (FPU) Register Bit 0 is Equal to Zero	BC1EQZ ft, offset	if FPR[ft] & 1 = 0 PC ← (PC + sign_extend(offset << 2))	010001	01001	ft	offset
*	Control Transfer Instruction	BC1NEZ	Branch if Coprocessor 1 (FPR) Register Bit 0 is Not Equal to Zero	BC1NEZ ft, offset	if FPR[ft] & 1 != 0 then PC ← (PC+ sign_extend(offset<< 2)	010001	01101	ft	offset

R-Type (6-5-5-5-5-6)											
0	Category	Instruction	Meaning	Format	Operation	OpCode (6)	rs(5)	rt(5)	rd(5)	sa(5)	func(6)
	Arithmetic Instruction	DADDU	Double-word addition	DADDU rd, rs, rt	GPR[rd] ← GPR[rs] + GPR[rt]	000000	rs	rt	rd	00000	101101
*	Arithmetic Instruction	DDIV	Double-word divide (signed)	DDIV rd,rs,rt	GPR[rd] ← GPR[rs] div GPR[rt]	000000	rs	rt	rd	00010	011110
*	Arithmetic Instruction	DMOD	Modulo signed integer (with result to GPR)	DMOD rd,rs,rt	GPR[rd] ← GPR[rs] mod GPR[rt]	000000	rs	rt	rd	00011	011110
*	Arithmetic Instruction	DDIVU	Double-word divide (unsigned)	DDIVU rd,rs,rt	GPR[rd] ← GPR[rs] div GPR[rt]	000000	rs	rt	rd	00010	011111
*	Arithmetic Instruction	DMODU	Modulo unsigned integer (with result to GPR)	DMODU rd,rs,rt	GPR[rd] ← GPR[rs] mod GPR[rt]	000000	rs	rt	rd	00011	011111
	Arithmetic Instruction	DSUBU	Double-word subtraction	DSUBU rd, rs, rt	GPR[rd] ← GPR[rs] - GPR[rt]	000000	rs	rt	rd	00000	101111
*	Arithmetic Instruction	DMUL	Multiply signed integer (with loworder result to GPR)	DMUL rd,rs,rt	GPR[rd] ← lo_dword(GPR[rs] * GPR[rt])	000000	rs	rt	rd	00010	011100
*	Arithmetic Instruction	DMUH	Multiply signed integer (with highorder result to GPR)	DMUH rd,rs,rt	GPR[rd] ← hi_dword(GPR[rs] * GPR[rt])	000000	rs	rt	rd	00011	011100
*	Arithmetic Instruction	DMULU	Multiply unsigned integer (with loworder result to GPR)	DMULU rd,rs,rt	GPR[rd] ← lo_dword(GPR[rs] * GPR[rt])	000000	rs	rt	rd	00010	011101
*	Arithmetic Instruction	DMUHU	Multiply unsigned integer (with highorder result to GPR)	DMUHU rd,rs,rt	GPR[rd] ← hi_dword(GPR[rs] * GPR[rt])	000000	rs	rt	rd	00011	011101
	Arithmetic Instruction	NOP	No Operation	NOP	NOP	000000	00000	00000	00000	00000	000000
*	Bit Swap Instruction	BITSWAP	Swaps (reverses) bits in each byte - 32 bit	BITSWAP rd,rt	for every byte in opcode, reverse the bits in each byte	011111	00000	rt	rd	00000	100000
*	Bit Swap Instruction	DBITSWAP	Swaps (reverses) bits in each byte - 64 bit	DBITSWAP rd,rt	for every byte in opcode, reverse the bits in each byte	011111	00000	rt	rd	00000	100100
	Special Move Instruction	DMFC1	Move double-word from FPR to GPR (64-bit)	DMFC1 rt,fs	GPR[rt] ←FPR[fs]	010001	00001	rt	fs	00000	000000
	Special Move Instruction	DMTC1	Move double-word from GPR to FPR (64-bit)	DMTC1 rt, fs	FPR[fs] ←GPR[rt]	010001	00101	rt	fs	00000	000000
	Special Move Instruction	MFC1	Move word from FPR to GPR (32-bit)	MFC1 rt, fs	GPR[rt] ←FPR[fs]	010001	00000	rt	fs	00000	000000
	Special Move Instruction	MTC1	Move word from GPR to FPR (32-bit)	MTC1 rt, fs	FPR[fs] ←GPR[rt]	010001	00100	rt	fs	00000	000000
*	Special Move Instruction	SELEQZ	Select integer GPR value or zero	SELEQZ rd,rs,rt	if (GPR[rt] = 0) then GPR[rd] ←- GPR[rs] else GPR[rd] ←- 0	000000	rs	rt	rd	00000	110101
*	Special Move Instruction	SELNEZ	Select integer GPR value or zero	SELNEZ rd,rs,rt	if (GPR[rt] <> 0) then GPR[rd] ←- GPR[rs] else GPR[rd] ←- 0	000000	rs	rt	rd	00000	110111
	Logical Instruction	AND	Logical AND	AND rd, rs, rt	GPR[rd] ← GPR[rs] & GPR[rt]	000000	rs	rt	rd	00000	100100
	Logical Instruction	NOR	Logical NOR	NOR rd, rs, rt	GPR[rd] ← ~ (GPR[rs] | GPR[rt])	000000	rs	rt	rd	00000	100111
	Logical Instruction	OR	Logical OR	OR rd, rs, rt	GPR[rd] ← GPR[rs] | GPR[rt]	000000	rs	rt	rd	00000	100101
	Logical Instruction	XOR	Logical XOR	XOR rd, rs, rt	GPR[rd] ← GPR[rs] XOR GPR[rt]	000000	rs	rt	rd	00000	100110
	Shift Instruction	DROTRV	Double-word rotate right variable	DROTRV rd, rt, rs	GPR[rd] ← GPR[rt] <rotate right> GPR[rs5..0]	000000	rs	rt	rd	00001	010110
	Shift Instruction	DSLLV	Double-word shift left logical variable	DSLLV rd, rt, rs	GPR[rd] ← GPR[rt] <<logical GPR[rs5..0]	000000	rs	rt	rd	00000	010100
	Shift Instruction	DSRAV	Double-word shift right arithmetic variable	DSRAV rd, rt, rs	GPR[rd] ← GPR[rt] >>arithmetic GPR[rs5..0]	000000	rs	rt	rd	00000	010111
	Shift Instruction	DSRLV	Double-word shift right logical variable	DSRLV rd, rt, rs	GPR[rd] ← GPR[rt] >>logical GPR[rs5..0]	000000	rs	rt	rd	00000	010110
	Shift Instruction	DROTR	Double-word rotate right	DROTR rd, rt, sa	GPR[rd] ← GPR[rs] <rotate right> sa4..0 (sa range is from 0 to 31)	000000	00001	rt	rd	sa	111010
	Shift Instruction	DSLL	Double-word shift left logical	DSLL rd, rt, sa	GPR[rd] ← GPR[rt] <<logical sa4..0 (sa range is from 0 to 31)	000000	00000	rt	rd	sa	111000
	Shift Instruction	DSRA	Double-word shift right arithmetic	DSRA rd, rt, sa	GPR[rd] ← GPR[rt] >>arithmetic sa4..0 (sa range is from 0 to 31)	000000	00000	rt	rd	sa	111011
	Shift Instruction	DSRL	Double-word shift right logical	DSRL rd, rt, sa	GPR[rd] ← GPR[rt] >>logical sa4..0 (sa range is from 0 to 31)	000000	00000	rt	rd	sa	111010
	Set-on Condition Instruction	SLT	Set if less than (Signed)	SLT rd, rs, rt	GPR[rd] ← (GPR[rs] < GPR[rt])	000000	rs	rt	rd	00000	101010
	Set-on Condition Instruction	SLTU	Set if less than (Unsigned)	SLTU rd, rs, rt	GPR[rd] ← (GPR[rs] < GPR[rt])	000000	rs	rt	rd	00000	101011
	Control Transfer Instruction	JALR	Jump and link ("call")	JALR rs	GPR[31] ← PC+8 (with branch delay slot) PC ← GPR[rs]	000000	rs	00000	11111	hint = 00000	001001
	Control Transfer Instruction	JALR	Jump and link ("call")	JALR rd, rs	GPR[31] ← PC+8 (with branch delay slot) PC ← GPR[rs]	000000	rs	00000	rd!=r0	hint = 00000	001001
	Control Transfer Instruction	JR	Unconditional Jump	JR rs	PC ← GPR[rs] (with branch delay slot)	000000	rs	00000	00000	hint = 00000	001001
	Floating Point Instruction	ABS.S	Floating Point Absolute Value	ABS.S fd, fs	FPR[fd] ← abs(FPR[fs])	010001	10000	00000	fs	fd	000101
	Floating Point Instruction	ABS.D	Floating Point Absolute Value	ABS.D fd, fs	FPR[fd] ← abs(FPR[fs])	010001	10001	00000	fs	fd	000101
	Floating Point Instruction	ADD.S	Add single precision	ADD.S fd, fs, ft	FPR[fd] ← FPR[fs] + FPR[ft]	010001	10000	ft	fs	fd	000000
	Floating Point Instruction	ADD.D	Add double precision	ADD.D fd, fs, ft	FPR[fd] ← FPR[fs] + FPR[ft]	010001	10001	ft	fs	fd	000000
*	Floating Point Instruction	CMP.LT.S	Compare Single precision FP values and record the results as either all 0s (false) or all 1s (true) in a floating point register	CMP.LT.S fd, fs, ft	FPR[fd] = FPR[fs] *cond* FPR[ft]	010001	10100	ft	fs	fd	011100
*	Floating Point Instruction	CMP.LE.S	Compare Single precision FP values and record the results as either all 0s (false) or all 1s (true) in a floating point register	CMP.LE.S fd, fs, ft	FPR[fd] = FPR[fs] *cond* FPR[ft]	010001	10100	ft	fs	fd	011110

*	Floating Point Instruction	CMP.EQ.S	Compare Single precision FP values and record the results as either all 0s (false) or all 1s (true) in a floating point register	CMP.EQ.S fd, fs, ft	FPR[fd] = FPR[fs] *cond* FPR[ft]	010001	10100	ft	fs	fd	010010
*	Floating Point Instruction	CMP.LT.D	Compare double precision FP values and record the results as either all 0s (false) or all 1s (true) in a floating point register	CMP.LT.D fd, fs, ft	FPR[fd] = FPR[fs] *cond* FPR[ft]	010001	10101	ft	fs	fd	011100
*	Floating Point Instruction	CMP.LE.D	Compare double precision FP values and record the results as either all 0s (false) or all 1s (true) in a floating point register	CMP.LE.D fd, fs, ft	FPR[fd] = FPR[fs] *cond* FPR[ft]	010001	10101	ft	fs	fd	011110
*	Floating Point Instruction	CMP.EQ.D	Compare double precision FP values and record the results as either all 0s (false) or all 1s (true) in a floating point register	CMP.EQ.D fd, fs, ft	FPR[fd] = FPR[fs] *cond* FPR[ft]	010001	10101	ft	fs	fd	010010
	Floating Point Instruction	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.D.S fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10000	00000	fs	fd	100001
	Floating Point Instruction	CVT.D.W	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.D.W fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10100	00000	fs	fd	100001
	Floating Point Instruction	CVT.D.L	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.D.L fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10101	00000	fs	fd	100001
	Floating Point Instruction	CVT.L.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.L.S fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10000	00000	fs	fd	100101
	Floating Point Instruction	CVT.L.D	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.L.D fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10001	00000	fs	fd	100101
	Floating Point Instruction	CVT.S.D	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.S.D fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10001	00000	fs	fd	100000
	Floating Point Instruction	CVT.S.W	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.S.W fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10100	00000	fs	fd	100000
	Floating Point Instruction	CVT.S.L	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.S.L fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10101	00000	fs	fd	100000
	Floating Point Instruction	CVT.W.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.W.S fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10000	00000	fs	fd	100100
	Floating Point Instruction	CVT.W.D	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.W.D fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10001	00000	fs	fd	100100
	Floating Point Instruction	DIV.S	Divide single precision	DIV.S fd, fs, ft	FPR[fd] ← FPR[fs] / FPR[ft]	010001	10000	ft	fs	fd	000011
	Floating Point Instruction	DIV.D	Divide double precision	DIV.D fd, fs, ft	FPR[fd] ← FPR[fs] / FPR[ft]	010001	10001	ft	fs	fd	000011
*	Floating Point Instruction	MAX.S	Scalar Floating-Point Maximum	MAX.S fd,fs,ft	FPR[fd]←maxNum(FPR[fs],FPR[ft])	010001	10000	ft	fs	fd	011110
*	Floating Point Instruction	MAX.D	Scalar Floating-Point Maximum	MAX.D fd,fs,ft	FPR[fd]←maxNum(FPR[fs],FPR[ft])	010001	10001	ft	fs	fd	011110
*	Floating Point Instruction	MAXA.S	Scalar Floating-Point argument with Maximum Absolute Value	MAXA.S fd,fs,ft	FPR[fd]←maxNumMag(FPR[fs],FPR[ft])	010001	10000	ft	fs	fd	011111
*	Floating Point Instruction	MAXA.D	Scalar Floating-Point argument with Maximum Absolute Value	MAXA.D fd,fs,ft	FPR[fd]←maxNumMag(FPR[fs],FPR[ft])	010001	10001	ft	fs	fd	011111
*	Floating Point Instruction	MIN.S	Scalar Floating-Point Minimum	MIN.S fd,fs,ft	FPR[fd]←minNum(FPR[fs],FPR[ft])	010001	10000	ft	fs	fd	011100
*	Floating Point Instruction	MIN.D	Scalar Floating-Point Minimum	MIN.D fd,fs,ft	FPR[fd]←minNum(FPR[fs],FPR[ft])	010001	10001	ft	fs	fd	011100
*	Floating Point Instruction	MINA.S	Scalar Floating-Point argument with Minimum Absolute Value	MINA.S fd,fs,ft	FPR[fd]←minNumMag(FPR[fs],FPR[ft])	010001	10000	ft	fs	fd	011101
*	Floating Point Instruction	MINA.D	Scalar Floating-Point argument with Minimum Absolute Value	MINA.D fd,fs,ft	FPR[fd]←minNumMag(FPR[fs],FPR[ft])	010001	10001	ft	fs	fd	011101
	Floating Point Instruction	MOV.S	Move from one Single Precision FPR to another	MOV.S fd, fs	FPR[fd] ←FPR[fs]	010001	10000	00000	fs	fd	000110
	Floating Point Instruction	MOV.D	Move from one Double Precision FPR to another	MOV.D fd, fs	FPR[fd] ←FPR[fs]	010001	10001	00000	fs	fd	000110

	Floating Point Instruction	MUL.S	Multiply single precision	MUL.S fd, fs, ft	FPR[fd] ← FPR[fs] * FPR[ft]	010001	10000	ft	fs	fd	000010
	Floating Point Instruction	MUL.D	Multiply double precision	MUL.D fd, fs, ft	FPR[fd] ← FPR[fs] * FPR[ft]	010001	10001	ft	fs	fd	000010
	Floating Point Instruction	NEG.S	Floating Point Negate	NEG.S fd, fs	FPR[fd] ← -FPR[fs]	010001	10000	00000	fs	fd	000111
	Floating Point Instruction	NEG.D	Floating Point Negate	NEG.D fd, fs	FPR[fd] ← -FPR[fs]	010001	10001	00000	fs	fd	000111
	Floating Point Instruction	RECIP.S	Reciprocal Approximation	RECIP.S fd, fs	FPR[fd] ← 1.0 / FPR[fs]	010001	10000	00000	fs	fd	010101
	Floating Point Instruction	RECIP.D	Reciprocal Approximation	RECIP.D fd, fs	FPR[fd] ← 1.0 / FPR[fs]	010001	10001	00000	fs	fd	010101
	Floating Point Instruction	RSQRT.S	Reciprocal Square Root Approximation	RSQRT.S fd, fs	FPR[fd] ← 1.0 / sqrt(FPR[fs])	010001	10000	00000	fs	fd	010110
	Floating Point Instruction	RSQRT.D	Reciprocal Square Root Approximation	RSQRT.D fd, fs	FPR[fd] ← 1.0 / sqrt(FPR[fs])	010001	10001	00000	fs	fd	010110
	Floating Point Instruction	SQRT.S	Floating Point Square Root	SQRT.S fd, fs	FPR[fd] ← SQRT(FPR[fs])	010001	10000	00000	fs	fd	000100
	Floating Point Instruction	SQRT.D	Floating Point Square Root	SQRT.D fd, fs	FPR[fd] ← SQRT(FPR[fs])	010001	10001	00000	fs	fd	000100
*	Floating Point Instruction	SEL.S	Select floating point values with FPR condition	SEL.S fd,fs,ft	FPR[fd] ← FPR[fd].bit0 ? FPR[ft] : FPR[fs]	010001	10000	ft	fs	fd	010000
*	Floating Point Instruction	SEL.D	Select floating point values with FPR condition	SEL.D fd,fs,ft	FPR[fd] ← FPR[fd].bit0 ? FPR[ft] : FPR[fs]	010001	10001	ft	fs	fd	010000
*	Floating Point Instruction	SELEQZ.S	Select floating point value or zero with FPR condition	SELEQZ.S fd,fs,ft	FPR[fd] ← FPR[ft].bit0 ? 0 : FPR[fs]	010001	10000	ft	fs	fd	010100
*	Floating Point Instruction	SELEQZ.D	Select floating point value or zero with FPR condition	SELEQZ.D fd,fs,ft	FPR[fd] ← FPR[ft].bit0 ? 0 : FPR[fs]	010001	10001	ft	fs	fd	010100
*	Floating Point Instruction	SELNEZ.S	Select floating point value or zero with FPR condition	SELNEZ.S fd,fs,ft	FPR[fd] ← FPR[ft].bit0 ? FPR[fs]: 0	010001	10000	ft	fs	fd	010111
*	Floating Point Instruction	SELNEZ.D	Select floating point value or zero with FPR condition	SELNEZ.D fd,fs,ft	FPR[fd] ← FPR[ft].bit0 ? FPR[fs]: 0	010001	10001	ft	fs	fd	010111
	Floating Point Instruction	SUB.S	Subtract single precision	SUB.S fd, fs, ft	FPR[fd] ← FPR[fs] - FPR[ft]	010001	10000	ft	fs	fd	000001
	Floating Point Instruction	SUB.D	Subtract double precision	SUB.D fd, fs, ft	FPR[fd] ← FPR[fs] - FPR[ft]	010001	10001	ft	fs	fd	000001
	Floating Point Instruction	CEIL.L.S	Fixed Point Ceiling Convert to Long Fixed Point	CEIL.L.S fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10000	00000	fs	fd	001010
	Floating Point Instruction	CEIL.L.D	Fixed Point Ceiling Convert to Long Fixed Point	CEIL.L.D fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10001	00000	fs	fd	001010
	Floating Point Instruction	CEIL.W.S	Floating Point Ceiling Convert to Word Fixed Point	CEIL.W.S fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10000	00000	fs	fd	001110
	Floating Point Instruction	CEIL.W.D	Floating Point Ceiling Convert to Word Fixed Point	CEIL.W.D fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10001	00000	fs	fd	001110
	Floating Point Instruction	FLOOR.L.S	Floating Point Floor Convert to Long Fixed Point	FLOOR.L.S fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10000	00000	fs	fd	001011
	Floating Point Instruction	FLOOR.L.D	Floating Point Floor Convert to Long Fixed Point	FLOOR.L.D fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10001	00000	fs	fd	001011
	Floating Point Instruction	FLOOR.W.S	Floating Point Floor Convert to Word Fixed Point	FLOOR.W.S fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10000	00000	fs	fd	001111
	Floating Point Instruction	FLOOR.W.D	Floating Point Floor Convert to Word Fixed Point	FLOOR.W.D fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10001	00000	fs	fd	001111
*	Floating Point Instruction	RINT.S	Floating-Point Round to Integral	RINT.S fd,fs	FPR[fd] ← round_int(FPR[fs])	010001	10000	00000	fs	fd	011010
*	Floating Point Instruction	RINT.D	Floating-Point Round to Integral	RINT.D fd,fs	FPR[fd] ← round_int(FPR[fs])	010001	10001	00000	fs	fd	011010
	Floating Point Instruction	ROUND.L.S	Floating Point Round to Long Fixed Point	ROUND.L.S fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10000	00000	fs	fd	001000
	Floating Point Instruction	ROUND.L.D	Floating Point Round to Long Fixed Point	ROUND.L.D fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10001	00000	fs	fd	001000
	Floating Point Instruction	ROUND.W.S	Floating Point Round to Word Fixed Point	ROUND.W.S fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10000	00000	fs	fd	001100
	Floating Point Instruction	ROUND.W.D	Floating Point Round to Word Fixed Point	ROUND.W.D fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10001	00000	fs	fd	001100
	Floating Point Instruction	TRUNC.L.S	Floating Point Truncate to Long Fixed Point	TRUNC.L.S fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10000	00000	fs	fd	001001
	Floating Point Instruction	TRUNC.L.D	Floating Point Truncate to Long Fixed Point	TRUNC.L.D fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10001	00000	fs	fd	001001
	Floating Point Instruction	TRUNC.W.S	Floating Point Truncate to Word Fixed Point	TRUNC.W.S fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10000	00000	fs	fd	001101

	Floating Point Instruction	TRUNC.W.D	Floating Point Truncate to Word Fixed Point	TRUNC.W.D fd, fs	FPR[fd] ← convert_and_round(FPR[fs])	010001	10001	00000	fs	fd	001101