			M	IPS64 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)					
				GPR[31] ← PC+4* (no delay branch slot)									
				PC ← PC + sign extend(offset<<2)									
				*GPR[31] ←address of the next instruction									
				or reprint address of the first instruction									
				Internally, offset (which is 26-bit signed offset) is left shifted twice and									
Control				then added to the address of the instruction following the branch. Thus,									
Transfer				the offset can also be viewed as the distance of instruction to the label.									
* Instruction	BALC	Branch and link, compact ("call")	BALC offset	The next instruction after the branch is viewed as offset 0.	111010			offset					
				$PC \leftarrow PC + sign_extend(offset << 2)$									
				Internally, offset (which is 26-bit signed offset) is left shifted twice and									
Control				then added to the address of the instruction following the branch. Thus,									
Transfer				the offset can also be viewed as the distance of instruction to the label.									
* Instruction	BC	Unconditional branch, compact	BC offset	The next instruction after the branch is viewed as offset 0. PC ← PC6328 (instr_index << 2)	110010			offset					
				FC ← FC0328 (IIISU_IIIdex ≪ 2)									
Control				Internally, instr_label (which is 26-bit) is left shifted twice and then									
Transfer				replaces the lower 28 bits of the PC. Thus, instr index can be viewed									
^ Instruction	T	Unconditional Jump	J target	as target address div 4	000010		i	nstruction index					
Instruction		Onconditional value	, ungo	as anget address dry	000010			instruction mater					
				GPR[31] ← PC+8* (with branch delay slot)									
				$PC \leftarrow PC6328 \mid \mid (instr_index << 2)$									
				*GPR[31] ←address of the next instruction after the branch delay slot									
Control				Internally, instr_label (which is 26-bit) is left shifted twice and then									
Transfer			***	replaces the lower 28 bits of the PC. Thus, instr_index can be viewed	000011								
^ Instruction	JAL	Jump and link ("call")	JAL target	as target address div 4	000011		1	nstruction index					
I-Type (6-5-21)													
0 Category	Instruction	Meaning	Format	Operation	OpCode (6)	rs(5)		variable(21)					
Control						-(-/							
Transfer		Compact branch if GPR rs is equal		if GPR[rs] = 0 then									
* Instruction	BEQZC	to zero	BEQZC rs, offset	$PC \leftarrow (PC + sign extend(offset << 2))$	110110	rs != r0		offset					
Control													
Transfer		Compact branch if GPR rs is not		if GPR[rs] != 0 then									
* Instruction	BNEZC	equal to zero	BNEZC rs, offset	$PC \leftarrow (PC + sign extend(offset << 2))$	111110	rs != r0	1	offset					
I-Type (6-5-5-16)													
0 Category	Instruction	Meaning	Format	Operation	OpCode (6)	rs(5)	rt(5)	variable(16)					
Arithmetic					(0)	(-)	(-/						
Instruction	DADDIU	Double-word add w/immediate	DADDIU rt, rs, immediate	$GPR[rt] \leftarrow GPR[rs] + immediate$	011001	rs	rt	immediate					
Logical													

I-Type (6-5-5-16)		he :		lo :	0.01.	(5)		:11.00
0 Category	Instruction	Meaning	Format	Operation	OpCode (6)	rs(5)	rt(5)	variable(16)
Arithmetic				anne i anne i i i				
Instruction	DADDIU	Double-word add w/immediate	DADDIU rt, rs, immediate	$GPR[rt] \leftarrow GPR[rs] + immediate$	011001	rs	rt	immediate
Logical				CDDI 1 CDDI 10 ' F.	001100		l .	
Instruction	ANDI	Logical AND with immediate	ANDI rt, rs, immediate	$GPR[rt] \leftarrow GPR[rs]$ & immediate	001100	rs	rt	immediate
Logical	ORI	Iil ODid. idi.ee	ODI at an immediate	CDD[-1] CDD[-1] investigate	001101		l	
Instruction	UKI	Logical OR with immediate	ORI rt, rs, immediate	$GPR[rt] \leftarrow GPR[rs] \mid immediate$	001101	rs	rt	immediate
Logical	XORI	IiI VODid. i dist.	VODI : fire	CDD[-1] CDD[-1] VOD ilists	001110	rs	l.,	
Instruction Set-on	AURI	Logical XOR with immediate	XORI rt, rs, immediate	GPR[rt] ← GPR[rs] XOR immediate	001110	IS	п	immediate
Condition				$GPR[rt] \leftarrow (GPR[rs] < immediate)$				
Instruction	SLTI	Set if less than immediate (Signed)	SI TI rt rs immediate	Of K[tt] (Of K[ts] < ininicalate)	001010	rs	rt	immediate
Set-on	SEII	Set it less than inniculate (Signed)	SETTR, 13, manediate		001010	13		inniculate
Condition		Set if less than immediate		$GPR[rt] \leftarrow (GPR[rs] < immediate)$				
Instruction	SLTIU	(Unsigned)	SLTIU rt, rs, immediate	or reprise (or reprise management)	001011	rs	rt	immediate
Arithmetic	BETTE	(Choighea)	DETTO 11, 10, Immediate		001011	1.0		minediate
* 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		- I - I - I - I - I - I - I - I - I - I						
Instruction	DAUI	Doubleword Add Upper Immediate	DAUI rt. rs.immediate	$GPR[rt] \leftarrow GPR[rs] + sign extend(immediate << 16)$	011101	rs != r0	rt	immediate
Arithmetic		Doubleword Add Higher	7 . 7					
Instruction	DAHI	Immediate	DAHI rs, rs, immediate	$GPR[rs] \leftarrow GPR[rs] + sign extend(immediate << 32)$	000001	rs	00110	immediate
Arithmetic								
* Instruction	DATI	Doubleword Add Top Immediate	DATI rs, rs, immediate	$GPR[rs] \leftarrow GPR[rs] + sign_extend(immediate << 48)$	000001	rs	11110	immediate
Load-Store								
Instruction	LB	Load byte (signed)	LB rt, offset(base)	$GPR[rt] \leftarrow sign_extend(memory[GPR[base] + offset])$	100000	base	rt	offset
Load-Store								
Instruction	LBU	Load byte (unsigned)	LBU rt, offset(base)	$GPR[rt] \leftarrow zero \ extend(memory[GPR[base] + offset])$	100100	base	rt	offset
Load-Store								
Instruction	LD	Load double-word	LD rt, offset(base)	$GPR[rt] \leftarrow memory[GPR[base] + offset]$	110111	base	rt	offset
Load-Store								
Instruction	L.D	Load double-precision	L.D ft, offset(base)	$FPR[ft] \leftarrow memory[GPR[base] + offset]$	110101	base	ft	offset
Load-Store								
Instruction	LDC1	Load double-precision	LDC1 ft, offset(base)	$FPR[ft] \leftarrow memory[GPR[base] + offset]$	110101	base	ft	offset
Load-Store						l.		
Instruction	LH	Load half-word (signed)	LH rt, offset(base)	$GPR[rt] \leftarrow sign \ extend(memory[GPR[base] + offset])$	100001	base	rt	offset
Load-Store		L		contra		L		
Instruction	LHU	Load half-word (unsigned)	LHU rt, offset(base)	$GPR[rt] \leftarrow zero \ extend(memory[GPR[base] + offset])$	100101	base	rt	offset
Load-Store	LW	T 4 4 (-1 4)	I W at a 66 and a cons	CDD[+]	100011		l	- 664
Instruction	LW	Load word (signed)	LW rt, offset(base)	$GPR[rt] \leftarrow sign_extend(memory[GPR[base] + offset])$	100011	base	π	offset
Load-Store	LS	I and simple provision	I S ft offort/boos)	EDD [61] . momor [CDD [boss] official]	100011	haaa		offset
Instruction Load-Store	13	Load single-precision	L.S ft, offset(base)	$FPR[ft] \leftarrow memory[GPR[base] + offset]$	100011	base	11	OHSEL
Instruction	LWC1	Load single-precision	LWC1 ft, offset(base)	$FPR[ft] \leftarrow memory[GPR[base] + offset]$	110001	base	ft	offset
Load-Store	EHCI	Load single-precision	L 11 CT II, Ulisei(base)	T I I I I I I I I I I I I I I I I I I I	110001	vase	11	Offset
Instruction	LWU	Load word (unsigned)	LWU rt, offset(base)	GPR[rt] ← zero_extend(memory[GPR[base] + offset])	100111	base	rt	offset
Load-Store						Juse	<u> </u>	GALLEC
Instruction	LUI	Load upper immediate	LUI rt, immediate	$GPR[rt] \leftarrow sign \ extend(immediate 0^{16})$	001111	00000	rt	immediate
Load-Store			. ,	- [-1 - 0(
Instruction	SB	Store byte	SB rt, offset(base)	$memory[GPR[base] + offset] \leftarrow GPR[rt]$	101000	base	rt	offset
Load-Store		1						
Instruction	SD	Store double-word	SD rt, offset(base)	$memory[GPR[base] + offset] \leftarrow GPR[rt]$	111111	base	rt	offset
Load-Store								
Instruction	S.D	Store double-precision	S.D ft, offset(base)	$memory[GPR[base] + offset] \leftarrow FPR[ft]$	111101	base	ft	offset
Load-Store								
Instruction	SDC1	Store double-precision	SDC1 ft, offset(base)	$memory[GPR[base] + offset] \leftarrow FPR[ft]$	111101	base	ft	offset
Load-Store								
Instruction	SH	Store half-word	SH rt, offset(base)	$memory[GPR[base] + offset] \leftarrow GPR[rt]$	101001	base	rt	offset
Load-Store								
Instruction	SW	Store word	SW rt, offset(base)	$memory[GPR[base] + offset] \leftarrow GPR[rt]$	101011	base	rt	offset

Load-Store Instruction	S.S	Store single-precision	S.S ft, offset(base)	$memory[GPR[base] + offset] \leftarrow FPR[ft]$	111001	base	ft	offset
Load-Store Instruction	SWC1	Store single-precision	SWC1 ft, offset(base)	$memory[GPR[base] + offset] \leftarrow FPR[ft]$	111001	base	ft	offset
		9.7		PC ← PC + sign_extend(offset<<2)				
Control				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,				
Transfer Instruction	D	Unconditional Branch	B offset	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
HISTIUCTION	Б	Unconditional Branch	B offset	The next instruction after the branch is viewed as offset 0.	000100	00000	00000	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				
Control Transfer				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.				
	BAL	Branch and Link	BAL offset	The next instruction after the branch is viewed as offset 0.	000001	00000	10001	offset
				If GPR[rs]=GPR[rt] then				
				$PC \leftarrow PC + sign_extend(offset << 2)$				
				Internally, offset (which is 16-bit signed offset) is left shifted twice and				
Control Transfer				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.				
Instruction	BEQ	Branch if equal	BEQ rs, rt, offset	The next instruction after the branch is viewed as offset 0.	000100	rs	rt	offset
				If GPR[rs]=GPR[rt] then				
				$PC \leftarrow PC + sign_extend(offset << 2)$				
				Internally, offset (which is 16-bit signed offset) is left shifted twice and		1		
Control Transfer				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.		1		
Instruction	BEQC	Branch if equal Compact	BEQC rs,rt, offset	The next instruction after the branch is viewed as offset 0.	001000	rs < rt	rt/rs != r0	offset
				If GPR[rs]<>GPR[rt] then		1		
				$PC \leftarrow PC + sign_extend(offset << 2)$		1		
_				Internally, offset (which is 16-bit signed offset) is left shifted twice and				
Control Transfer				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.				
Instruction	BNEC	Branch if not equal Compact	BNEC rs,rt, offset	The next instruction after the branch is viewed as offset 0.	011000	rs < rt	rt/rs != r0	offset
				If GPR[rs] <> GPR[rt] then				
				$PC \leftarrow PC + sign_extend(offset << 2)$				
G1				Internally, offset (which is 16-bit signed offset) is left shifted twice and				
Control Transfer				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.				
Instruction Control	BNE	Branch if not equal	BNE rs, rt, offset	The next instruction after the branch is viewed as offset 0.	000101	rs	rt	offset
Transfer		Branch if greater than or equal to		if $GPR[rs] >= 0$ then				
Instruction Control	BGEZ	zero	BGEZ rs, offset	$PC \leftarrow (PC + sign \ extend(offset << 2))$	000001	rs	00001	offset
Transfer Instruction	BGTZ	Branch if greater than zero	BGTZ rs, offset	if $GPR[rs] > 0$ then $PC \leftarrow (PC + sign extend(offset << 2))$	000111	rs	00000	offset
Control	BUIZ	Branch ii greater than zero	BG1Z1s, onset		000111	15	00000	onset
Transfer Instruction	BLEZ	Branch if less than or equal to zero	BLEZ rs, offset	if $GPR[rs] \le 0$ then $PC \leftarrow (PC + sign_extend(offset \le 2))$	000110	rs	00000	offset
Control Transfer				if GPR[rs] < 0 then				
Instruction	BLTZ	Branch if less than zero	BLTZ rs, offset	$PC \leftarrow (PC + sign_extend(offset << 2))$	000001	rs	00000	offset
Control Transfer				if GPR[rt] < 0 then				
Instruction	BLTZC	Branch if less than zero compact	BLTZC rt, offset	$PC \leftarrow (PC + sign extend(offset << 2))$	010111	rs = rt	rt != r0	offset
Control Transfer		Branch if greater than or equal to		if $GPR[rt] >= 0$ then				
Instruction Control	BGEZC	zero compact	BGEZC rt, offset	PC ← (PC+ sign_extend(offset<<2))	010110	rs = rt	rt != r0	offset
Transfer	D. F.	Branch if less than or equal to zero	DI FIZO . C	if GPR[rt] <= 0 then	010177	00000		ana.
Instruction Control	BLEZC	compact	BLEZC rt, offset	$PC \leftarrow (PC + sign_extend(offset << 2))$	010110	00000	rt != r0	offset
Transfer Instruction	BGTZC	Branch if greater than zero compact	BGT7C rt_offset	if $GPR[rt] > 0$ then $PC \leftarrow (PC + sign extend(offset << 2))$	010111	00000	rt != r0	offset
Control	20120		DOTZETI, OHSEL		010111	00000	10	Offset
Transfer Instruction	BLTC	Compact branch if GPR rs is less than GPR rt	BLTC rs,rt, offset	if $GPR[rs] < GPR[rt]$ then $PC \leftarrow (PC + sign_extend(offset << 2))$	010111	rs!=rt	rt != r0	offset
Control Transfer		Compact branch if GPR rt is greater than GPR rs (alias for		if GPR[rt] > GPR[rs] then				
Instruction	BGTC	BLTC)	BGTC rt,rs, offset	$PC \leftarrow (PC + \text{sign extend(offset} << 2))$	010111	rt != rs	rs != r0	offset
Control Transfer		Compact branch if GPR rs is		if $GPR[rs] >= GPR[rt]$ then				
Instruction Control	BGEC	greater than or equal to GPR rt Compact branch if GPR rt is less	BGEC rs,rt, offset	$PC \leftarrow (PC + sign \ extend(offset << 2))$	010110	rs!=rt	rt != r0	offset
Transfer		than or equal to GPR rs (alias for		$if \ GPR[rt] <= GPR[rs] \ then$		1		
Instruction Control	BLEC	BGEC)	BLEC rt,rs, offset	$PC \leftarrow (PC + sign_extend(offset << 2))$	010110	rt != rs	rs != r0	offset
Transfer	BLTUC	Compact branch if GPR rs is less	DI THE root offers	if unsigned(GPR[rs]) < unsigned(GPR[rt]) then $PC \leftarrow (PC + sign \ extend(offset << 2))$	000111	I	IO	affine
Instruction Control	DLIUC	than GPR rt, unsigned Compact branch if GPR rt is	BLTUC rs,rt, offset		000111	18 != rt	rt != r0	offset
Transfer Instruction	BGTUC	greater than GPR rs, unsigned (alias for BLTUC)	BGTUC rt,rs, offset	if unsigned(GPR[rt]) > unsigned(GPR[rs]) then $PC \leftarrow (PC + sign extend(offset << 2))$	000111	rt != rs	rs != r0	offset
Control	,50100	Compact branch if GPR rs is	= 5100 Miss. Offset		, 500111	:- 15	10	GHSCI
	BGEUC	greater than or equal to GPR rt, unsigned	BGEUC rs,rt, offset	if unsigned(GPR[rs]) $>=$ unsigned(GPR[rt]) then PC \leftarrow (PC+ sign extend(offset $<<$ 2))	000110	rs!= rt	rt != r0	offset
Transfer Instruction		Compact branch if GPR rt is less than or equal to GPR rt, unsigned	., ., .	if unsigned(GPR[n]) <= unsigned(GPR[rs]) then				
Instruction Control			1			Ι.		offset
Instruction Control Transfer Instruction	BLEUC	(alias for BGEUC)	BLEUC rt,rs, offset	$PC \leftarrow (PC + sign extend(offset << 2))$	000110	rt!= rs	rs != r0	Oliset
Instruction Control Transfer Instruction Control	BLEUC		BLEUC rt,rs, offset	PC ← (PC+ sign_extend(offset<<2))	000110	rt != rs	rs != r0	onset
Instruction Control Transfer Instruction	BLEUC BC1EQZ	(alias for BGEUC) Branch if Coprocessor 1 (FPU) Register Bit 0 is Equal to Zero	BLEUC rt,rs, offset BC1EQZ ft, offset	if $FPR[ft]$ & $1 = 0$ PC \leftarrow (PC + sign_extend(offset $<<$ 2))	010001	o1001	ft	offset

R-Type (6-5-5-5-5 0 Category	-6) 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	$GPR[rd] \leftarrow GPR[rs] + GPR[rt]$	000000	re	rt.	rd	00000	101101
Arithmetic						15				
* Instruction Arithmetic	DDIV	Double-word divide (signed) Modulo signed integer (with result	DDIV rd,rs,rt	$GPR[rd] \leftarrow GPR[rs] \text{ div } GPR[rt]$	000000	rs	rt	rd	00010	011110
* Instruction Arithmetic	DMOD	to GPR)	DMOD rd,rs,rt	$GPR[rd] \leftarrow GPR[rs] \mod GPR[rt]$	000000	rs	rt	rd	00011	011110
* Instruction Arithmetic	DDIVU	Double-word divide (unsigned) Modulo unsigned integer (with	DDIVU rd,rs,rt	$GPR[rd] \leftarrow GPR[rs] \text{ div } GPR[rt]$	000000	rs	rt	rd	00010	011111
* Instruction Arithmetic	DMODU	result to GPR)	DMODU rd,rs,rt	$GPR[rd] \leftarrow GPR[rs] \mod GPR[rt]$	000000	rs	rt	rd	00011	011111
Instruction	DSUBU	Double-word subtraction Multiply signed integer (with	DSUBU rd, rs, rt	$GPR[rd] \leftarrow GPR[rs] - GPR[rt]$	000000	rs	rt	rd	00000	101111
* Instruction	DMUL	loworder result to GPR)	DMUL rd,rs,rt	$GPR[rd] \leftarrow lo_dword(GPR[rs] * GPR[rt])$	000000	rs	rt	rd	00010	011100
* Instruction	DMUH	Multiply signed integer (with highorder result to GPR)	DMUH rd,rs,rt	$GPR[rd] \leftarrow hi \ dword(GPR[rs] * GPR[rt])$	000000	rs	rt	rd	00011	011100
* Arithmetic * Instruction	DMULU	Multiply unsigned integer (with loworder result to GPR)	DMULU rd,rs,rt	$GPR[rd] \leftarrow lo \ dword(GPR[rs] * GPR[rt])$	000000	rs	rt	rd	00010	011101
Arithmetic * Instruction	DMUHU	Multiply unsigned integer (with highorder result to GPR)	DMUHU rd,rs,rt	$GPR[rd] \leftarrow hi \ dword(GPR[rs] * GPR[rt])$	000000	rs	rt	rd	00011	011101
Arithmetic Instruction	NOP	No Operation	NOP	NOP	000000	00000	00000	00000	00000	000000
Bit Swap		Swaps (reverses) bits in each byte -								
* Instruction Bit Swap	BITSWAP	32 bit Swaps (reverses) bits in each byte -	BITSWAP rd,rt	for every byte in opcode, reverse the bits in each byte	011111	00000	rt	rd	00000	100000
* Instruction Special Move	DBITSWAP	64 bit Move double-word from FPR to	DBITSWAP rd,rt	for every byte in opcode, reverse the bits in each byte	011111	00000	rt	rd	00000	100100
Instruction Special Move	DMFC1	GPR (64-bit) Move double-word from GPR to	DMFC1 rt,fs	$GPR[rt] \leftarrow FPR[fs]$	010001	00001	rt	fs	00000	000000
Instruction Special Move	DMTC1	FPR (64-bit) Move word from FPR to GPR (32-	DMTC1 rt, fs	$FPR[fs] \leftarrow GPR[rt]$	010001	00101	rt	fs	00000	000000
Instruction	MFC1	bit)	MFC1 rt, fs	$GPR[\pi] \leftarrow FPR[fs]$	010001	00000	rt	fs	00000	000000
Special Move Instruction	MTC1	Move word from GPR to FPR (32- bit)	MTC1 rt, fs	$FPR[fs] \leftarrow GPR[rt]$	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 $GPR[rd] \leftarrow -GPR[rs]$ else $GPR[rd] \leftarrow -0$	000000	rs	rt	rd	00000	110101
* Special Move * Instruction	SELNEZ	Select integer GPR value or zero	SELNEZ rd,rs,rt	if $(GPR[rt] \Leftrightarrow 0)$ then $GPR[rd] \leftarrow GPR[rs]$ else $GPR[rd] \leftarrow 0$	000000	rs	rt	rd	00000	110111
Logical Instruction	AND	Logical AND	AND rd, rs, rt	$GPR[rd] \leftarrow GPR[rs] & GPR[rt]$	000000	rs	rt	rd	00000	100100
Logical						15				
Instruction Logical	NOR	Logical NOR	NOR rd, rs, rt	$GPR[rd] \leftarrow \sim (GPR[rs] GPR[rt])$	000000	rs	rt	rd	00000	100111
Instruction Logical	OR	Logical OR	OR rd, rs, rt	$GPR[rd] \leftarrow GPR[rs] GPR[rt]$	000000	rs	rt	rd	00000	100101
Instruction Shift	XOR	Logical XOR	XOR rd, rs, rt	GPR[rd] ← GPR[rs] XOR GPR[rt]	000000	rs	rt	rd	00000	100110
Instruction Shift	DROTRV	Double-word rotate right variable Double-word shift left logical	DROTRV rd, rt, rs	$GPR[rd] \leftarrow GPR[rt] < rotate right > GPR[rs50]$	000000	rs	rt	rd	00001	010110
Instruction	DSLLV	variable	DSLLV rd, rt, rs	$GPR[rd] \leftarrow GPR[rt] << logical GPR[rs50]$	000000	rs	rt	rd	00000	010100
Shift Instruction	DSRAV	Double-word shift right arithmetic variable	DSRAV rd, rt, rs	$GPR[rd] \leftarrow GPR[rt] >> arithmetic GPR[rs50]$	000000	rs	rt	rd	00000	010111
Shift Instruction	DSRLV	Double-word shift right logical variable	DSRLV rd, rt, rs	$GPR[rd] \leftarrow GPR[rt] >> logical GPR[rs50]$	000000	rs	rt	rd	00000	010110
Shift Instruction	DROTR	Double-word rotate right	DROTR rd, rt, sa	GPR[rd] ← GPR[rs] <rotate right=""> sa40 (sa range is from 0 to 31)</rotate>	000000	00001	rt	rd	sa	111010
Shift Instruction	DSLL	Double-word shift left logical	DSLL rd, rt, sa	$GPR[rd] \leftarrow GPR[rt] \ll logical sa40$ (sa range is from 0 to 31)	000000	00000	rt	rd	sa	111000
Shift				$GPR[rd] \leftarrow GPR[rt] >> arithmetic sa40$						
Instruction Shift	DSRA		DSRA rd, rt, sa	(sa range is from 0 to 31) GPR[rd] ← GPR[rt] ≫logical sa40	000000	00000	π	rd	sa	111011
Instruction Set-on	DSRL	Double-word shift right logical	DSRL rd, rt, sa	(sa range is from 0 to 31)	000000	00000	rt	rd	sa	111010
Condition Instruction	SLT	Set if less than (Signed)	SLT rd, rs, rt	$GPR[rd] \leftarrow (GPR[rs] < GPR[rt])$	000000	rs	rt	rd	00000	101010
Set-on Condition				$GPR[rd] \leftarrow (GPR[rs] < GPR[rt])$						
Instruction Control	SLTU	Set if less than (Unsigned)	SLTU rd, rs, rt	Or Kini — (Or Kis) > Or Kini)	000000	rs	rt	rd	00000	101011
Transfer				GPR[31] ← PC+8 (with branch delay slot)						
Instruction Control	JALR	Jump and link ("call")	JALR rs	$PC \leftarrow GPR[rs]$	000000	rs	00000	11111	hint = 00000	001001
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	$PC \leftarrow GPR[rs]$ (with branch delay slot)	000000	rs	00000	00000	hint = 00000	001001
Floating Point Instruction	ABS.S	Floating Point Absolute Value	ABS.S fd, fs	$FPR[fd] \leftarrow abs(FPR[fs])$	010001	10000	00000	fs	fd	000101
Floating Point										
Instruction	ABS.D	Floating Point Absolute Value	ABS.D fd, fs	$FPR[fd] \leftarrow abs(FPR[fs])$	010001	10001	00000	fs	fd	000101
Floating Point										
Instruction	ADD.S	Add single precision	ADD.S fd, fs, ft	$FPR[fd] \leftarrow FPR[fs] + FPR[ft]$	010001	10000	ft	fs	fd	000000
Floating Point Instruction	ADD.D	Add double precision	ADD.D fd, fs, ft	$FPR[fd] \leftarrow FPR[fs] + FPR[ft]$	010001	10001	ft	fs	fd	000000
		Compare Single precision FP								
Election Deli		values and record the results as								
* Floating Point * Instruction	CMP.LT.S	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
		Compare Single precision FP								
Floating Point		values and record the results as either all 0s (false) or all 1s (true)								
* Instruction	CMP.LE.S	in a floating point register	CMP.LE.S fd, fs, ft	FPR[fd] = FPR[fs] cond FPR[ft]	010001	10100	ft	fs	fd	011110

				T	I	I				
Floating Point * Instruction	t 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	fe	fd	010010
Floating Point		Compare double precision FP values and record the results as either all 0s (false) or all 1s (true)	CM 1.20.5 Id, 15, It	i i kajaj - i i kajaj toma i i kajaj	010001	10100	1	13	М	010010
* Instruction	CMP.LT.D	in a floating point register Compare double precision FP	CMP.LT.D fd, fs, ft	FPR[fd] = FPR[fs] cond FPR[ft]	010001	10101	ft	fs	fd	011100
* Floating Point * Instruction	t CMP.LE.D	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		Compare double precision FP values and record the results as either all 0s (false) or all 1s (true)								
* Instruction	CMP.EQ.D	in a floating point register Convert from type y to type x; where x & y can be s (SP floating	CMP.EQ.D fd, fs, ft	FPR[fd] = FPR[fs] cond FPR[ft]	010001	10101	ft	fs	fd	010010
Floating Point Instruction	t CVT.D.S	point), d (DP floating point), w (32- bit integer), 1 (64-bit integer)	CVT.D.S fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10000	00000	fs	fd	100001
Floating Point	t 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), 1 (64-bit integer)	CVT.D.W fd, fs	FPR[fd]←convert and round(FPR[fs])	010001	10100	00000	fs	fd	100001
Floating Point		Convert from type y to type x; where x & y can be s (SP floating point), d (DP floating point), w (32-								
Instruction	CVT.D.L	bit integer), 1 (64-bit integer) Convert from type y to type x;	CVT.D.L fd, fs	FPR[fd]←convert and round(FPR[fs])	010001	10101	00000	fs	fd	100001
Floating Point Instruction	CVT.L.S	where x & y can be s (SP floating point), d (DP floating point), w (32- bit integer), 1 (64-bit integer)	CVT.L.S fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10000	00000	fs	fd	100101
Floating Point		Convert from type y to type x; where x & y can be s (SP floating point), d (DP floating point), w (32-								
Instruction	CVT.L.D	Convert from type y to type x;	CVT.L.D fd, fs	FPR[fd]←convert and round(FPR[fs])	010001	10001	00000	fs	fd	100101
Floating Point Instruction	t CVT.S.D	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		Convert from type y to type x; where x & y can be s (SP floating point), d (DP floating point), w (32-	CYMC W.C. C	CDD(C)	010001	10100			c.	100000
Instruction	CVT.S.W	bit integer), I (64-bit integer) Convert from type y to type x; where x & y can be s (SP floating	CVT.S.W fd, fs	FPR[fd]—convert and round(FPR[fs])	010001	10100	00000	IS	<u>id</u>	100000
Floating Point Instruction	CVT.S.L	point), d (DP floating point), w (32- bit integer), l (64-bit integer) Convert from type y to type x;	CVT.S.L fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10101	00000	fs	fd	100000
Floating Point	t CVT.W.S	where x & y can be s (SP floating point), d (DP floating point), w (32- bit integer), 1 (64-bit integer)	CVT.W.S fd, fs	FPR[fd]←convert and round(FPR[fs])	010001	10000	00000	fs	fd	100100
Floating Point	t	Convert from type y to type x; where x & y can be s (SP floating point), d (DP floating point), w (32-								
Instruction Floating Point	CVT.W.D	bit integer), 1 (64-bit integer)	CVT.W.D fd, fs	FPR[fd]←convert_and_round(FPR[fs])	010001	10001	00000	fs	fd	100100
Instruction Floating Point	DIV.S	Divide single precision	DIV.S fd, fs, ft	$FPR[fd] \leftarrow FPR[fs] / FPR[ft]$	010001	10000	ft	fs	fd	000011
Floating Point * Instruction	DIV.D t MAX.S	Divide double precision Scalar Floating-Point Maximum	DIV.D fd, fs, ft MAX.S fd,fs,ft	$\begin{aligned} & FPR[fd] \leftarrow FPR[fs] / FPR[ft] \\ & FPR[fd] \leftarrow maxNum(FPR[fs], FPR[ft]) \end{aligned}$	010001	10001	ft	fs	fd	011110
Floating Point * Instruction		Scalar Floating-Point Maximum Scalar Floating-Point Maximum	MAX.D fd,fs,ft	$FPR[fd]\leftarrow maxNum(FPR[fs],FPR[ft])$ $FPR[fd]\leftarrow maxNum(FPR[fs],FPR[ft])$	010001	10000	ft	fs	fd	011110
Floating Point * Instruction		Scalar Floating-Point argument with Maximum Absolute Value	MAXA.S fd,fs,ft	$FPR[fd] \leftarrow maxNumMag(FPR[fs], FPR[ft])$	010001	10000	ft	fs	fd	011111
* Floating Point * Instruction	t MAXA.D	Scalar Floating-Point argument with Maximum Absolute Value	MAXA.D fd,fs,ft	$FPR[fd] \leftarrow maxNumMag(FPR[fs], FPR[ft])$	010001	10001	ft	fs	fd	011111
* 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
* 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
* Instruction	MINA.S	Scalar Floating-Point argument with Minimum Absolute Value Scalar Floating-Point argument	MINA.S fd,fs,ft	FPR[fd]←minNumMag(FPR[fs],FPR[ft])	010001	10000	ft	fs	fd	011101
* Instruction Floating Point	MINA.D	with Minimum Absolute Value Move from one Single Precision	MINA.D fd,fs,ft	$FPR[fd] \leftarrow minNumMag(FPR[fs],FPR[ft])$	010001	10001	ft	fs	fd	011101
Instruction	MOV.S	FPR to another	MOV.S fd, fs	$FPR[fd] \leftarrow FPR[fs]$	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

1	1	1					1		1	т
Floating Point Instruction	MUL.S	Multiply single precision	MUL.S fd, fs, ft	$FPR[fd] \leftarrow FPR[fs] * FPR[ft]$	010001	10000	ft	fs	fd	000010
Floating Point Instruction	MUL.D	Multiply double precision	MUL.D fd, fs, ft	$FPR[fd] \leftarrow FPR[fs] * FPR[ft]$	010001	10001	ft	fs	fd	000010
Floating Point Instruction	NEG.S	Floating Point Negate	NEG.S fd, fs	$FPR[fd] \leftarrow -FPR[fs]$	010001	10000	00000	fs	fd	000111
Floating Point Instruction	NEG.D	Floating Point Negate	NEG.D fd, fs	$FPR[fd] \leftarrow -FPR[fs]$	010001	10001	00000	fs	fd	000111
Floating Point Instruction	RECIP.S	Reciprocal Approximation	RECIP.S fd, fs	FPR[fd] ← 1.0 / FPR[fs]	010001		00000	60	64	010101
Floating Point								15	iu	
Instruction Floating Point	RECIP.D	Reciprocal Approximation Reciprocal Square Root	RECIP.D fd, fs	$FPR[fd] \leftarrow 1.0 / FPR[fs]$	010001	10001	00000	fs	fd	010101
Instruction Floating Point	RSQRT.S	Approximation Reciprocal Square Root	RSQRT.S fd, fs	$FPR[fd] \leftarrow 1.0 / sqrt(FPR[fs])$	010001	10000	00000	fs	fd	010110
Instruction Floating Point	RSQRT.D	Approximation	RSQRT.D fd, fs	$FPR[fd] \leftarrow 1.0 / sqrt(FPR[fs])$	010001	10001	00000	fs	fd	010110
Instruction	SQRT.S	Floating Point Square Root	SQRT.S fd, fs	$FPR[fd] \leftarrow SQRT(FPR[fs])$	010001	10000	00000	fs	fd	000100
Floating Point Instruction	SQRT.D	Floating Point Square Root	SQRT.D fd, fs	$FPR[fd] \leftarrow SQRT(FPR[fs])$	010001	10001	00000	fs	fd	000100
* Instruction	SEL.S	Select floating point values with FPR condition	SEL.S fd,fs,ft	$FPR[fd] \leftarrow FPR[fd].bit0 ? FPR[ft] : FPR[fs]$	010001	10000	ft	fs	fd	010000
* Floating Point * Instruction	SEL.D	Select floating point values with FPR condition	SEL.D fd,fs,ft	$FPR[fd] \leftarrow FPR[fd].bit0 ? FPR[ft] : FPR[fs]$	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	$FPR[fd] \leftarrow FPR[ft].bit0?0:FPR[fs]$	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	$FPR[fd] \leftarrow FPR[ft].bit0?0:FPR[fs]$	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	$FPR[fd] \leftarrow FPR[ft].bit0? FPR[fs]: 0$	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	$FPR[fd] \leftarrow FPR[ft].bit0 ? FPR[fs]: 0$	010001	10001	ft	fs	fd	010111
Floating Point Instruction	SUB.S	Subtract single precision	SUB.S fd, fs, ft	$FPR[fd] \leftarrow FPR[fs] - FPR[ft]$	010001	10000	ft	fs	fd	000001
Floating Point Instruction	SUB.D	Subtract double precision	SUB.D fd, fs, ft	$FPR[fd] \leftarrow FPR[fs] - FPR[ft]$	010001	10001	ft	fs	fd	000001
Floating Point		Fixed Point Ceiling Convert to					-	is .	iu .	
Instruction Floating Point	CEIL.L.D	Long Fixed Point Fixed Point Ceiling Convert to	CEIL.L.S fd, fs	FPR[fd] ← convert and round($FPR[fs]$) $FPR[fd]$ ← convert and round($FPR[fs]$)	010001	10000	00000	IS .	e.	001010
Instruction Floating Point		Long Fixed Point Floating Point Ceiling Convert to	CEIL.L.D fd, fs					IS	Id	
Instruction Floating Point	CEIL.W.S	Word Fixed Point Floating Point Ceiling Convert to	CEIL.W.S fd, fs	$FPR[fd] \leftarrow convert_and_round(FPR[fs])$	010001	10000	00000	fs	fd	001110
Instruction Floating Point	CEIL.W.D	Word Fixed Point Floating Point Floor Convert to	CEIL.W.D fd, fs	$FPR[fd] \leftarrow convert_and_round(FPR[fs])$	010001	10001	00000	fs	fd	001110
Instruction	FLOOR.L.S	Long Fixed Point	FLOOR.L.S fd, fs	$FPR[fd] \leftarrow convert \text{ and } round(FPR[fs])$	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	$FPR[fd] \leftarrow convert \text{ and } round(FPR[fs])$	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	$FPR[fd] \leftarrow convert_and_round(FPR[fs])$	010001	10001	00000	fs	fd	001111
* Floating Point * Instruction	RINT.S	Floating-Point Round to Integral	RINT.S fd,fs	$FPR[fd] \leftarrow round_int(FPR[fs])$	010001	10000	00000	fs	fd	011010
Floating Point * Instruction	RINT.D	Floating-Point Round to Integral	RINT.D fd,fs	$FPR[fd] \leftarrow round int(FPR[fs])$	010001	10001	00000	fs	fd	011010
Floating Point Instruction	ROUND.L.S	Floating Point Round to Long Fixed Point	ROUND.L.S fd, fs	$FPR[fd] \leftarrow convert$ 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	$FPR[fd] \leftarrow convert_and_round(FPR[fs])$	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		Floating Point Truncate to Long						4.0		
Instruction Floating Point	TRUNC.L.S	Fixed Point Floating Point Truncate to Long	TRUNC.L.S fd, fs	FPR[fd] ← convert and round(FPR[fs])	010001	10000	00000	fs	fd	001001
Instruction Floating Point	TRUNC.L.D	Floating Point Truncate to Word	TRUNC.L.D fd, fs	FPR[fd] ← convert and round(FPR[fs])	010001	10001	00000	fs	fd	001001
Instruction	TRUNC.W.S	Fixed Point	TRUNC.W.S fd, fs	$FPR[fd] \leftarrow convert_and_round(FPR[fs])$	010001	10000	00000	fs	fd	001101

Г											
	Floating Point		Floating Point Truncate to Word								
	Instruction	TRUNC.W.D	Fixed Point	TRUNC.W.D fd, fs	$FPR[fd] \leftarrow convert \ and \ round(FPR[fs])$	010001	10001	00000	fs	fd	001101