ROC32_8 operator list

James C Brakefield © 2020

register zero used to hold return address
data sizes: byte(8), half(16), word(32), long(64)
grey rows are instructions for ROC64_8, e.g use a 64-bit ALU

immediate field is value of -8..15 or length of postfixed immediate in bytes
last four registers are condition code, frame pointer, stack pointer & PC
CCR as base register & register zero as index register read as zero

format name	instruction layout	usage	address arithmetic
fmt24drsi	00xxxxxx ddddd i rrrrr sssss	basic 24-bit instruction	
fmt24drs	00xxxxxx ddddd 0 rrrrr sssss	R op S ->D	
fmt24drn	00xxxxxx ddddd 1 rrrrr nnnnn	R op immediate ->D	
fmt24dbj	00xxxxxx ddddd 0 bbbbb jjjjj	load/store base + scaled index	B + J * ds
fmt24dbn	00xxxxxx ddddd 1 bbbbb nnnnn	load/store base + offset	B + N
fmt24dr	00xxxxxx ddddd z rrrrr xxxxx	op R -> D; single operand instructions	
fmt24dn	00xxxxxx ddddd nnnnnnnnnnn	eleven bit signed offset or immediate	PC + N

format name	instruction layout	usage	address arithmetic
fmt32drsc	01xxxxxx ddddd i rrrrr sssss zz u ccccc	basic 32-bit instruction	
fmt32drsc	01xxxxxx ddddd 0 rrrrr sssss zz u ccccc	R op S ->D; execute on predicate match	
fmt32drnc	01xxxxxx ddddd 1 rrrrr nnnnn z v u ccccc	R op imm ->D; execute on predicate match	
fmt32dbjs	01xxxxxx ddddd 0 bbbbb sssss yy u jjjjj	load from base + index * scaling + offset	B + J * Y + S
fmt32dbjn	01xxxxxx ddddd 1 bbbbb nnnnn yy u jjjjj	load from base + index * scaling + offset	B + J * Y + N
fmt32dbjs	01xxxxxx ddddd 0 bbbbb sssss yy 0 jjjjj	store to base + index * scaling + offset	B + J * Y + S
fmt32dbjn	01xxxxxx ddddd 1 bbbbb nnnnn yy 0 jjjjj	store to base + index * scaling + offset	B + J * Y + N
fmt32mbjs	01xxxxxx mmmmm 0 bbbbb sssss yy 1 jjjjj	store immediate to base + index * scaling + offset	B + J * Y + S
fmt32mbjn	01xxxxxx mmmmm 1 bbbbb nnnnn yy 1 jjjjj	store immediate to base + index * scaling + offset	B + J * Y + N
fmt32dr	01xxxxxx ddddd z rrrrr xxxxxxx u ccccc	op R -> D; single operand instructions	
fmt32dn	01xxxxxx ddddd nnnnnnnnnnnnnnnnnn	nineteen bit signed offset or immediate	PC + N

24-bit formats	neumonic	short description (72 of 64 allocated)	comments
fmt24dbj, fmt24dbn	LDU8	load unsigned byte from memory	
fmt24dbj, fmt24dbn	LDU16	load unsigned half from memory	
fmt24dbj, fmt24dbn	LDU32	load unsigned word from memory	
fmt24dbj, fmt24dbn	LD64	load unsigned long from memory	
fmt24dbj, fmt24dbn	LDI8	load signed byte from memory	
fmt24dbj, fmt24dbn	LDI16	load signed half from memory	
fmt24dbj, fmt24dbn	LDI32	load signed word from memory	
fmt24dbj, fmt24dbn	ST8	store byte to memory	32-bit version supports store immediate
fmt24dbj, fmt24dbn	ST16	store half to memory	32-bit version supports store immediate
fmt24dbj, fmt24dbn	ST32	store word to memory	32-bit version supports store immediate
fmt24dbj, fmt24dbn	ST64	store long to memory	32-bit version supports store immediate
fmt24drs, fmt24drn	ADD, ADDI	R + S -> D	contents of register R + contents of register S to register D
fmt24drs, fmt24drn	SUB, SUBI	R - S -> D	
fmt24drs, fmt24drn	ADC, ADCI	R + S + carry -> D	
fmt24drs, fmt24drn	SBC, SBCI	R - S + carry -> D	

fmt24drs, fmt24drn	MUL, MULI	R * S -> D	
fmt24drs, fmt24drn	MULU, MULUI	unsigned multiply, upper half result	
fmt24drs, fmt24drn	MULUS, MULUSI	signed multiply, upper half result	
fmt24drs, fmt24drn	DIVU, DIVUI	R / S -> D, unsigned	
fmt24drs, fmt24drn	DIVS, DIVSI	signed integer divide	
fmt24drs, fmt24drn	REMU, REMUI	unsigned integer divide remainder	
fmt24drs, fmt24drn	REMS, REMSI	signed integer divide remainder	
fmt24drs, fmt24drn	AND, ANDI	R and S -> D	
fmt24drs, fmt24drn	OR, ORI	R or S -> D	
fmt24drs, fmt24drn	XOR, XORI	R xor S -> D	
fmt24drs, fmt24drn	CMP, CMPI	R : S -> D, result to D register	My 66000 CCR format plus even/odd bits
fmt24drs, fmt24drn	FADD, FADDI	floating-point add	
fmt24drs, fmt24drn	FSUB, FSUBI	floating-point subtract	
fmt24drs, fmt24drn	FMUL, FMULI	floating-point multiply	
	FMA, FMAI	floating-point multiply and add	three operands
fmt24drs, fmt24drn	FDIV, FDIVI	floating-point divide	
fmt24drs, fmt24drn	FCMP, FCMPI	floating-point compare, result to D register	My 66000 CCR format
fmt24drs, fmt24drn	FATAN2PI	arc-tangent in rotations for two operands	
fmt24drs, fmt24drn	FPOW, FPOWI	R raised to the S power	
fmt24drs, fmt24drn	INSRT, INSRTI	bit field insert	starting bit and length concatenated into S or N as a 12-bit contiguous field
fmt24drs, fmt24drn	EXTRCT, EXTRCTI	unsigned extract bit field	starting bit and length concatenated into S or N as a 12-bit contiguous field
fmt24drs, fmt24drn	EXTRCTS, EXTRCTSI	sign extended bit field extract	starting bit and length concatenated into S or N as a 12-bit contiguous field
fmt24drs, fmt24drn	ROL, ROLI	rotate left	
fmt24drs, fmt24drn	SHR, SHRI	shift right	
fmt24drs, fmt24drn	ASR, ASRI	arithmetic shift right	
fmt24drs, fmt24drn	SHL, SHLI	shift left	
fmt24dbj, fmt24dbn	JMPcc, CALLcc	conditional jump/call	D is the condition, JMP-never mapped to CALL
fmt24dn	BRcc, BSRcc	conditional relative branch/call with 11-bit signed offs	D is the condition, BR-never mapped to CALL
fmt24drs, fmt24drn	BBS, BBC	relative branch on bit set/clear	N: branch delta, R: source, D: bit number; requires 2/4 op-codes on 64-bit machine
fmt24drs, fmt24drn	MAX, MAXI	signed maximum	
fmt24drs, fmt24drn	MIN, MINI	signed minimum	
fmt24drs, fmt24drn	EADD, EADDI	add to exponent	
fmt24dn	VECT	identify loop vector registers	
fmt24drs, fmt24drn	LOOP, LOOPI	vector loop instruction	
fmt24dn	LDI	load immediate	
fmt24dr		single operand instructions	operand can be negated/inverted
	Probably useful instructions		
	MMOV	memory to memory byte string move	My 66000 instruction
	PCND	create predicate shadow on condition	My 66000 instruction
	PCB1	create predicate shadow on bit	My 66000 instruction
	STM	multi-register save to memory	My 66000 instruction
	LDM	multi-register load from memory	My 66000 instruction
fmt24dbj, fmt24dbn	FLD8, FLD16, FLD32, FLD64	floating-point load	could support 2nd 16-bit float format instead of the 8-bit float

fmt24dbj, fmt24dbn	FST8, FST16, FST32, FST64	floating-point store	could support 2nd 16-bit float format instead of the 8-bit float
fmt24drs, fmt24drn	FMAX, FMAXI	floating-point maximum	My 66000 instruction
fmt24drs, fmt24drn	FMIN, FMINI	floating-point minimum	My 66000 instruction
	LEA8, LEA16, LEA32, LEA64	load effective address	

Single Operand Instructions (92 allocated of 32 in fmt24, or of 128 in fmt32), operand negate/invert supported

fmt24dr, fmt32dr	IN	read input port	R is the port number
	OUT	write to output port	R is the port number
	MOV, MOVI	move register	macro op
	NOT	Boolean complement	macro op
	NEG	Negate	macro op
	INC, DEC	Increment, Decrement	macro op
	ABS, FABS, NABS, FNABS	Absolute value	macro op
	LDZCNT, LD1CNT, TRZCNT, TR1CNT	Leading and trailing zero/ones count	ranges from zero to register size
	POPCNT	Count of one/zero bits	ranges from zero to register size
	SINPI, COSPI, TANPI	Trig functions in half rotations	
	ASINPI, ACOSPI, ATANPI	Trig functions in half rotations	
	INT, FLT	Integer from/to float	
	ANDCCR	clear bits in CCR	macro op
	ORCCR	set bits in CCR	macro op
	CVT (48)	to & from: single, double, unsigned, signed	up to 6 rounding modes
	EXPON	extract exponent	
	FRACT	extract fraction	
	SQRT	square root	
	FSQRT	floating-point square root	
	FRSQRT	floating-point reciprical square root	
	FRCP	reciprical	macro op
	LN2P1	log to base 2 plus one	
	LN2	log to base 2	
	LNP1	natural log plus one	
	LN	natural log	
	LOGP1	base 10 logarithm plus one	
	LOG	base 10 logarithm	
	EXP2M1	base 2 exponentation minus one	
	EXP2	base 2 exponentation	
	EXPM1	base E exponentation minus one	
	EXP	base E exponentation	
	EXP10M1	base 10 exponentation minus one	
	EXP10	base 10 exponentation	
	SIN	trig functions in radians	
	COS	trig functions in radians	
	TAN	trig functions in radians	
	ASIN	trig functions in radians	
	ACOS	trig functions in radians	

ATAN	trig functions in radians
RND	round using current mode
RNDTEV	round to nearest, ties to even
RNDTOD	inexact round to nearest odd
RNDTZ	round toward zero
RNDFZ	round away from zero
CEIL	ceiling
FLOOR	floor

Triple Operand Instructions, typically 32 or 40-bit instructions, see also RTF64 by Robert Finch

MEDIAN	
MAX3	
MIN3	
ADD3	
INTP-BLEND-LERP	linear interpolation
FMAC	multiply-add
LUT	Boolean predicate over multiple bits (5 to 8), R & D are start and ending register numbers which sets the size of the LUT
CMOV	Select one of two operands
MERG	Select bits from two operands