**TROC16\_4234\_ISA\_250522**.docx

**Tagged Register Oriented Computer**

James Brakefield

Instruction set architecture for 16-bit Register Oriented Computer supporting 16, 24 and 32-bit instructions on thirty-two 16-bit registers. It is a 16-bit rendition of the full TROC ISA with half-word aligned instructions and data. There are several sections:

1. 16-bit instructions using full encoding of immediate values, that is small values within the five bit register field, 16-bit values using an additional half-word. A different encoding is used for 24 and 32-bit instructions.
2. 24-bit instructions supporting the majority of the instruction set.
3. 32-bit instructions providing three operand instructions and three additional op-code bits.
4. 8-bit instructions from the 32-bit instruction code space for Trap, Breakpoint, Word align and Cache line align.
5. Remaining encoding space reserved for 40 and 48-bit instructions.
6. Slots for all possible instructions for ease of exploration.
7. TROC16 normally has a minimum number of 24 & 32-bit instructions.

The 32-register 16-bit register file is augmented with two “tag” bits for data type and two exponent “tag” bits additional on each register. The data types are: unsigned, signed, floating-point and a fourth data type which could be a second floating-point format or fixed point logarithm or something entirely different. The program counter (PC) and a “residue” register are part of the register file (registers 31 and 28) but implemented as distinct registers. The residue register captures the carry from unsigned add/subtract, the signed carry/overflow from signed add/subtract, the upper half of multiply and the remainder from divide.

Floating-point arithmetic and conversion between memory and register formats not yet formally defined. Expectation is that several float formats will be supported. The other “floating-point” type may instead be fixed-point logarithm or pointers for variable length arithmetic.

**The instruction formats are:**

16-bit instruction specifies N or S

sssss ddddd xxxxx1

nnnnn ddddd xxxxx1 -15…+15

nnnnnnnn nnnnnnnn 10000 ddddd xxxxx1 -32768…32767

24-bit instruction has an immediate enable bit, limited to one additional immediate byte

rrrrr 0 xx sssss ddddd xxxx10

0nnnn 1 xx sssss ddddd xxxx10 -8…+7

1nnnn 1 xx sssss ddddd xxxx10 -1024…+1023

32-bit instruction has three additional op-code bits, no additional immediate byte

ttttt xxx rrrrr 0 xx sssss ddddd xxxx00

nnnnn xxx rrrrr 1 xx sssss ddddd xxxx00 -16…+15

The **X** are the op-code bits. The least significant instruction bits are arranged so that an all zero instruction is a trap, e.g. the processor has encountered an invalid instruction.

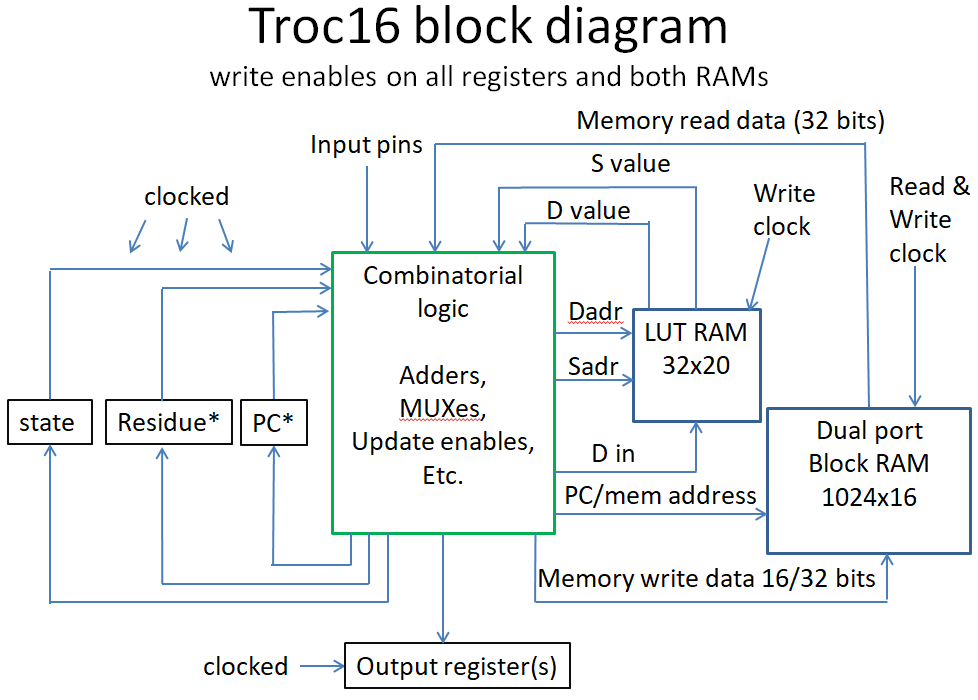
**D**, **S**, **R** and **T** are five bit register selectors.

**N** is a five bit immediate value with multiple interpretations.

For Troc16 all instructions use half-word addressing and half-word displacements/absolute addresses. The 24 and 32-bit Troc16 instructions are limited to 32-bits including any immediate values.

For 24-bit instructions, N indicates a value from -8 to +7 or a 12-bit signed value via an additional instruction byte.

For 32-bit instructions, N indicates a value from -16 to +15.



Using two port register file

Troc24 supports byte addressing and alignments

**16-bit instruction table for Troc16**

**LSBs: xxx001 xxx011 xxx101 xxx111**

**MSBs *n* *N* *n* *N* S *n* *N***

000xx1 IN LD SHFT EXTRCT ADD ADDI ibid

001xx1 OUT ST FUNCT INSRT SUB SUBI ibid

010xx1 BSR ibid AOB ibid MUL MULI ibid

011xx1 MOV S->D SOB ibid DIV DIVI ibid

100xx1 BRM ibid LDI ibid AND ANDI ibid

101xx1 BRP ibid LDIS ibid OR ORI ibid

110xx1 BRZ ibid LDIF ibid XOR XORI ibid

111xx1 BRNZ ibid LDIF2 ibid CMP CMPI ibid

**24-bit instruction table for Troc16**

"10 000110" p3**LD16U** Load half-word unsigned

"10 010110" p3**LD16S** Load half-word signed

"10 100110" p3**LD16F** Load half-word float and convert to register format

"10 110110" p3**LD16F2** Load half-word float2 and convert to register format

"10 001110" p3**LD32U** Load word including tag bits, no conversion

"01 000110" p3**ST16** Store half-word converting register format to memory format

"01 001110" p3**ST32** Store half-word and its tag bits without conversion

**8-bit Instruction table for Troc16**

“00000000” p1**TRAP** Illegal instruction trap, save current PC in internal save register

“01000000” p1**BKPT** Debug trap, save current PC in internal save register

**8-bit Instruction descriptions:**

Op-code R? Mnemonic Precise description (R? column indicates residue update)

“00000000” N p1**TRAP** Illegal instruction trap, save current PC in internal save register

Branch to illegal instruction handler Undefined instructions used different handler

“01000000” N p1**BKPT** Debug trap, save current PC in internal save register

Branch to debug handler location

“10000000” N p1**ALGNW** Skip to next word aligned instruction.

Word size is implementation defined or set by configuration bit.

“11000000” N p1**ALGNC** Skip to next cache line or DRAM line aligned instruction

**16-bit Instruction descriptions:**

If the destination register is residue, the normal residue result is not recorded.

16-bit instructions use -15 to +15 or 16-bit N interpretation.

For Troc16 all instructions use half-word addressing and half-word displacements.

Op-code R? Mnemonic Precise description (R? column indicates residue update)

“000001” N p2**IN** Store value of input port N into D if N is in the range from -15 to +15.

Negative N signifies internal ports, positive N external ports. D’s tag bits set to unsigned.

“000001” N p2**LD** When *n* is outside the range from -15 to +15 load D from absolute

memory location N. D’s tag bits unchanged, for floats memory value is converted to

register format

“001001” N p2**OUT** Place D into output port N if N is in the range from -15 to +15.

Negative N signifies internal ports, positive N external ports. D’s tag bits set to unsigned.

“001001” N p2**ST** When *n* is outside the range from -15 to +15 store D into absolute

memory location N. For floats D’s value is converted to memory format

“010001” N p2**BSR** Branch to PC + N, save address of next instruction in D

“011001“ N p2**MOV** Move register S to D, tag bits moved as well

“100001” N p2**BRZ**  Branch to PC + N if D is zero

“101001” N p2**BRNZ** Branch to PC + N if D is non-zero

“110001” N p2**BRP**  Branch to PC + N if D is positive (MSB is zero)

“111001” N p2**BRM**  Branch to PC + N if D is negative (MSB is one)

“000101” Y p2**ADD** Add S to D, if types differ is currently unimplemented trap

Residue register captures the carry, for signed ADD, carry can be either zero, +1 or -1

For floating-point ADD residue register captures round-off error

Floating-point ADD makes use of and may modify the exponent tag bits

“001101” Y p2**SUB** Subtact D from S, if types differ is currently unimplemented trap

Residue register captures the carry, for signed SUB, carry can be either zero, +1 or -1

For floating-point SUB residue register captures round-off error

Floating-point SUB makes use of and may modify the exponent tag bits

“010101” Y p2**MUL** Multiply D by S and store in D, if types differ is unimplemented trap

Residue register captures upper half of product for signed or unsigned multiply

For floating-point MUL residue register captures round-off error

Floating-point MUL makes use of and may modify the exponent tag bits

“011101” Y p2**DIV** Divide D by S and store in D, if types differ is unimplemented trap

Residue register captures remainder for signed or unsigned divide

For floating-point DIV residue register captures a floating-point remainder

Floating-point DIV makes use of and may modify the exponent tag bits

“100101” N p2**AND** Logical AND between S and D to D irrespective of data types

“101101” N p2**OR** Logical OR between S and D to D irrespective of data types

“110101” N p2**XOR** Logical exclusive or between S and D to D irrespective of data types

“111101” ~Y p2**CMP** S subtracted from D and placed in residue register

“000111” Y p2**ADDI** Add I to D, I converted to register format of D before add

Residue register captures the carry, for signed ADDI, carry can be either zero, +1 or -1

For floating-point ADDI residue register captures round-off error

Floating-point ADDI makes use of and may modify the exponent tag bits

“001111” Y p2**SUBI** Subtract D from I, I converted to register format of D before SUB

Residue register captures the carry, for signed SUBII, carry can be either zero, +1 or -1

For floating-point SUBI residue register captures round-off error

Floating-point SUBI makes use of and may modify the exponent tag bits

“010111” Y p2**MULI** Multiply D by I and store in D, if types differ is unimplemented trap

Residue register captures upper half of product for signed or unsigned multiply

For floating-point MULI residue register captures round-off error

Floating-point MULI makes use of and may modify the exponent tag bits

“011111” Y p2**DIVI** Divide D by I and store in D, if types differ is unimplemented trap

Residue register captures remainder for signed or unsigned divide

For floating-point DIVI residue register captures a floating-point remainder

Floating-point DIVI makes use of and may modify the exponent tag bits

“100111” N p2**ANDI** Logical AND between I and D to D irrespective of data types

“101111” N p2**ORI** Logical OR between I and D to D irrespective of data types

“110111” N p2**XORI** Logical exclusive or between I and D to D irrespective of data types

“111111” ~Y p2**CMPI** D subtracted from I and placed in residue register

“100011” N p2**LDI** load unsigned immediate, set tag bits to unsigned

“101011” N p2**LDIS** load signed data immediate, set tag bits to signed

“110011” N p2**LDIF** load float data immediate, convert N to register format

“111011” N p2**LDIF2** load float2 data immediate, convert N to register format

“010011” N p2**SOBN** Decrement D and branch to PC + N if D not zero

“011011” N p2**AOBN** Increment D and branch to PC + N if D not zero

“000011” Y p2**SHFT**  If N is in the range from -15 to +15 shift or adjust exponent by signed N.

For signed or unsigned D shift left or right per sign of N. For floating-point add signed N

to the exponent.

“000011” ~Y p2**EXTRCT** **I**f N is not in the range from -15 to +15 extract field from D and place in

Residue per N (uu000wwwww0mmmmm). Operation is irrespective of tag bits. U is data

type of the result. W is the width of the field extracted. If W is “00000” then width is

16. M is the starting bit of the field. For unsigned, bit field zero extended to register

width. For signed data type, bit field sign extended to register width.

“001011” N p2**INSRT I**f N is not in the range from -15 to +15 insert field from residue into D

per N (uu000wwwww0mmmmm): M is the residue shifted amount, W is the insert width

“001011” p2**FUNCT** If N is in the range from -15 to +15 do a one operand function f*n*(D) => D

“001011” “11111” p2f**CVTUS** If *n* is “00000” convert D to unsigned

“001011” “11110”” p2f**CVTS** If *n* is “00001” convert D to signed

“001011” “11101” p2f**CVTFLT** If *n* is “00010” convert D to float

“001011” “11100” p2f**CVTFLT2** If *n* is “00011” convert D to float2

“001011” N “00000” p2f**SETUS** If *n* is “00000” set D type to unsigned

“001011” N “00001” p2f**SETS** If *n* is “00001” set D type to signed

“001011” N “00010” p2f**SETFLT** If *n* is “00010” set D type to float

“001011” N “00011” p2f**SETFLT2** If *n* is “00011” set D type to float2

“001011” “00100” p2f**ABS** If *n* is “01000” take the absolute value of D

“001011” “00101” p2f**NABS** If *n* is “01001” take the negative absolute value of D

“001011” Y “00110” p2f**TAG** If *n* is “01001” place D’s tag bits into residue

“001011” N “00111” p2f**ITAG** If *n* is “01001” place residue’s tag bits into D

“001011” “xxxxx” Room for 19 more functions

**24-bit Instruction descriptions:**

For Troc24 with byte addressing and 24-bit instructions the index register R is multiplied by the data byte size and added to the base register S. If R is specified as the PC (register 31) a value of zero is used. It is OK to use the PC as a base register. Loading the PC (i.e. D is register 31) causes a branch.

The 24-bit instructions use the full TROC interpretation of N. If N is specified instead of R then N is added to the base register S without scaling.

For Troc24 the 24-bit instructions that load/store 32-bit data are repurposed for load/store of tagged register values.

The 24-bit instructions split the op-code into two parts: Bits 5 down to 0 contain xxxx01 where the x’s are the LSB op-code bits. Bits 17 down to 16 contain the most significant two bits. When “10” it is a load instruction. When “11” the op-code space is reserved for 40-bit instructions.

Op-code R? Mnemonic Precise description (R? column indicates residue update)

"10 000010" N p3**LD8U** Load byte unsigned, 24-bit instruction

If immediate (bit 18 set), address is register S + N, otherwise address is S + R

"10 000110" N p3**LD16U** Load half-word unsigned, 24-bit instruction

If immediate (bit 18 set), address is register S + N, otherwise address is S + 2 \* R

"10 001010" N p3**LD24U** Load 24-bit word unsigned, 24-bit instruction

If immediate (bit 18 set), address is register S + N, otherwise address is S + 3 \* R

"10 001110" N p3**LD32U** Load word unsigned, 24-bit instruction, includes tag bits

If immediate (bit 18 set), address is register S + N, otherwise address is S + 4 \* R

"10 010010" N p3**LD8S** Load byte signed, 24-bit instruction

If immediate (bit 18 set), address is register S + N, otherwise address is S + R \* ds

"10 010110" N p3**LD16S** Load half-word signed, 24-bit instruction

If immediate (bit 18 set), address is register S + N, otherwise address is S + 2 \* R

"10 011010" N p3**LD24S** Load 24-bit word signed, 24-bit instruction

If immediate (bit 18 set), address is register S + N, otherwise address is S + 3 \* R

"10 011110" N p3**LD32S** Load word signed, 24-bit instruction, includes tag bits

If immediate (bit 18 set), address is register S + N, otherwise address is S + 4 \* R

"10 100010" N p3**LD8F** Load byte unsigned, 24-bit instruction

If immediate, address is register S + N, otherwise address is S + R

"10 100110" N p3**LD16F** Load half-word float, 24-bit instruction

If immediate (bit 18 set), address is register S + N, otherwise address is S + 2 \* R

"10 101010" N p3**LD24F** Load 24-bit word float, 24-bit instruction

If immediate (bit 18 set), address is register S + N, otherwise address is S + 3 \* R

"10 101110" N p3**LD32F** Load word float, 24-bit instruction, includes tag bits

If immediate (bit 18 set), address is register S + N, otherwise address is S + 4 \* R

"10 110010" N p3**LD8F2** Load byte float2, 24-bit instruction

If immediate (bit 18 set), address is register S + N, otherwise address is S + R

"10 110110" N p3**LD16F2** Load half-word float2, 24-bit instruction

If immediate (bit 18 set), address is register S + N, otherwise address is S + 2 \* R

"10 111010" N p3**LD24F2** Load 24-bit word float2, 24-bit instruction

If immediate (bit 18 set), address is register S + N, otherwise address is S + 3 \* R

"10 111110" N p3**LD32F2** Load word float2, 24-bit instruction, includes tag bits

If immediate (bit 18 set), address is register S + N, otherwise address is S + 4 \* R

"01 000010" N p3**ST18** Store byte converting register format to memory format

If immediate (bit 18 set), address is register S + N, otherwise address is S + R

"01 000110" N p3**ST16** Store half-word converting register format to memory format

If immediate (bit 18 set), address is register S + N, otherwise address is S + 2 \* R

"01 001010" N p3**ST24** Store 24-bit word converting register format to memory format

If immediate (bit 18 set), address is register S + N, otherwise address is S + 3 \* R

"01 001110" N p3**ST32** Store word converting register format to memory format

If immediate (bit 18 set), address is register S + N, otherwise address is S + 4 \* R

"01 010010" empty, was LEA8

"01 010110" empty, was LEA16

"01 011010" empty, was LEA24

"01 011110" empty, was LEA32

"01 100010" Y p3**ADD, p3ADDI** R,I + S => D, if types differ is currently unimplemented trap

Residue register captures the carry, for signed ADD, carry can be either zero, +1 or -1

For floating-point ADD residue register captures round-off error

Floating-point ADD makes use of and may modify the exponent tag bits

"01 100110" Y p3**SUB**, p3**SUBI** R,I - S => D, if types differ is currently unimplemented trap

Residue register captures the carry, for signed SUB, carry can be either zero, +1 or -1

For floating-point SUB residue register captures round-off error

Floating-point SUB makes use of and may modify the exponent tag bits

“01 101010” Y p3**MUL, p3MULI** R,I \* S => D, if types differ is unimplemented trap

Residue register captures upper half of product for signed or unsigned multiply

For floating-point MUL residue register captures round-off error

Floating-point MUL makes use of and may modify the exponent tag bits

“01 101110” Y p3**DIV**, p3**DIVI** S / R,I => D, if types differ is unimplemented trap

Residue register captures remainder for signed or unsigned divide

For floating-point DIV residue register captures a floating-point remainder

Floating-point DIV makes use of and may modify the exponent tag bits

"01 110010" Y p3**AND,** p3**ANDI** Logical AND between R,I and S to D irrespective of data types

"01 110110" Y p3**ORI,** p3**ORII** Logical OR between R,I and S to D irrespective of data types

"01 111010" Y p3**XOR,** p3**XORI** Logical exclusive or between R,I and S to D irrespective of data types

"01 111110" N p3**CMP,** p3**CMPI** S subtracted from R,I and placed in register D

"00 0xxx10"

EXTRCT, EXTRCTI

INSRT, INSRTI

SHL, SHLI

MAX, MAXI

MIN, MINI

FUNC

INTP-BLEND-LERP, FINTP

CMOV

"00 1xxx10"

JMPcc, CALLcc

BRcc, BSRcc

BBS0

BBS1

BBC0

BBC1

empty

CASE, CASEI

"11 xxxx10" reserved for 40-bit instructions

**32-bit Instruction descriptions:**

For Troc24 with byte addressing and 32-bit instructions the scaled index register R added to the base register S and to the T register or immediate value. The three additional op-code bits in 32-bit instructions modify the scaling of the index register. The three bits indicate the number of data grouped items (“gs” from one to eight)[[1]](#footnote-1). Thus scaling of R can range from one to eight times the data size. If R is specified as the PC (register 31) a value of zero is used. It is OK to use the PC as a base register. Loading the PC (i.e. D is register 31) causes a branch. The symbol “T,I” represents value of register T or immediate value I depending on if bit 18 is set.

The 32-bit instructions use the full TROC interpretation of N.

For Troc24 the 32-bit instructions that load/store 32-bit data are repurposed for load/store of tagged values.

The 32-bit instructions split the op-code into two parts: Bits 5 down to 0 contain “xxxx00” where the x’s are the LSB op-code bits. Bits 17 down to 16 contain the most significant two bits. When “00” it is a load instruction. When “11” the op-code space is reserved for 48-bit instructions.

Bit 24 of 32-bit instructions is used as residue update enable where appropriate. Bits 25 & 26 used as sign change or one’s complement enables as appropriate on registers R & S.

Op-code R? Mnemonic Precise description (R? column indicates residue update)

"10 000000" N p4**LD8U** Load byte unsigned, 32-bit instruction

Memory address is register S + T,I + R \* gs. If R is PC then zero is used in place of PC.

"10 000100" N **LD16U** Load half-word unsigned, 24-bit instruction

Memory address is register S + T,I + 2 \* R \* gs. If R is PC then zero is used in place of PC.

"10 001000" N **LD24U** Load 24-bit word unsigned, 24-bit instruction

Memory address is register S + T,I + 3 \* R \* gs. If R is PC then zero is used in place of PC.

"10 001100" N **LD32U** Load word unsigned, 24-bit instruction, includes tag bits

If immediate, address is register S + N, otherwise address is S + 4 \* R \* gs

"10 010000" N **LD8S** Load byte signed, 24-bit instruction

If immediate, address is register S + N, otherwise address is S + R \* gs

"10 010100" N **LD16S** Load half-word signed, 24-bit instruction

If immediate, address is register S + N, otherwise address is S + 2 \* R \* gs

"10 011000" N **LD24S** Load 24-bit word signed, 24-bit instruction

If immediate, address is register S + N, otherwise address is S + 3 \* R \* gs

"10 011100" N **LD32S** Load word signed, 24-bit instruction, includes tag bits

If immediate, address is register S + N, otherwise address is S + 4 \* R \* gs

"10 100000" N **LD8F** Load byte unsigned, 24-bit instruction

If immediate, address is register S + N, otherwise address is S + R \* gs

"10 100100" N **LD16F** Load half-word float, 24-bit instruction

If immediate, address is register S + N, otherwise address is S + 2 \* R \* gs

"10 101000" N **LD24F** Load 24-bit word float, 24-bit instruction

If immediate, address is register S + N, otherwise address is S + 3 \* R \* gs

"10 101100" N **LD32F** Load word float, 24-bit instruction, includes tag bits

If immediate, address is register S + N, otherwise address is S + 4 \* R \* gs

"10 110000" N **LD8F2** Load byte float2, 24-bit instruction

If immediate, address is register S + N, otherwise address is S + R \* gs

"10 110100" N **LD16F2** Load half-word float2, 24-bit instruction

If immediate, address is register S + N, otherwise address is S + 2 \* R \* gs

"10 111000" N **LD24F2** Load 24-bit word float2, 24-bit instruction

If immediate, address is register S + N, otherwise address is S + 3 \* R \* gs

"10 111100" N **LD32F2** Load word float2, 24-bit instruction, includes tag bits

If immediate, address is register S + N, otherwise address is S + 4 \* R \* gs

"01 000000" N **ST18** Store byte converting register format to memory format

If immediate, address is register S + N, otherwise address is S + R \* gs

"01 000100" N **ST16** Store half-word converting register format to memory format

If immediate, address is register S + N, otherwise address is S + 2 \* R \* gs

"01 001000" N **ST24** Store 24-bit word converting register format to memory format

If immediate, address is register S + N, otherwise address is S + 3 \* R \* gs

"01 001100" N **ST32** Store word converting register format to memory format

If immediate, address is register S + N, otherwise address is S + 4 \* R \* gs

"01 01xx00" empty slots, was LEA

"01 010000" N **LEA8** Place byte memory address into D, type set to unsigned

If immediate, address is register S + N, otherwise address is S + R \* gs

"01 010100" N **LEA16** Place half-word memory address into D, type set to unsigned

If immediate, address is register S + N, otherwise address is S + 2 \* R \* gs

"01 011000" N **LEA24** Place 24-bit memory address into D, type set to unsigned

If immediate, address is register S + N, otherwise address is S + 3 \* R \* gs

"01 011100" N **LEA32** Place word memory address into D, type set to unsigned

If immediate, address is register S + N, otherwise address is S + 4 \* R \* gs

"01 100000" ~Y p4**ADD, p4ADDI** T,I + R + S => D, if types differ is currently unimplemented trap

Residue register captures the carry, for signed ADD, carry can be either zero, +1 or -1

For floating-point ADD residue register captures round-off error

Floating-point ADD makes use of and may modify the exponent tag bits

Bit 24 is residue update enable, bit 25 is sign change for S and bit 26 is sign change for R

"01 100100" ~Y p4**SUB**, p4**SUBI** T,I - R - S => D, if types differ is currently unimplemented trap

Residue register captures the carry, for signed SUB, carry can be either zero, +1 or -1

For floating-point SUB residue register captures round-off error

Floating-point SUB makes use of and may modify the exponent tag bits

Bit 24 is residue update enable, bit 25 is sign change for S and bit 26 is sign change for R

“01 101000” ~Y p4**MUL, p4MULI** T,I \* S + R => D, if types differ is unimplemented trap

Residue register captures upper half of product for signed or unsigned multiply

For floating-point MUL residue register captures round-off error

Floating-point MUL makes use of and may modify the exponent tag bits

Bit 24 is residue update enable, bit 25 is sign change for S and bit 26 is sign change for R

“01 101100” ~Y p4**DIV**, p4**DIVI** R|S / T,I => D, if types differ is unimplemented trap

For signed and unsigned, R is the remainder from a previous divide (R must be < T,I)

and is concatenated to S with appropriate sign handling.

Residue register captures remainder for signed or unsigned divide

For floating-point DIV residue register captures a floating-point remainder

Floating-point DIV makes use of and may modify the exponent tag bits

Bit 24 is residue update enable, bit 25 is sign change for S and bit 26 is sign change for R

"01 110000" N p4**AND,** p4**ANDI** Logical AND between T,I and R and S to D irrespective of data types

Bit 24 is complement enable for T,I; bit 25 is one’s complement enable for S and bit 26 is

complement enable for R

"01 110100" N p4**ORI,** p4**ORII** Logical OR between T,I and R and S to D irrespective of data types

Bit 24 is complement enable for T,I; bit 25 is one’s complement enable for S and bit 26 is

complement enable for R

"01 111000" N p4**XOR,** p4**XORI** Logical exclusive or between T,I and R and S to D irrespective of data

types. Bit 24 is zero, bit 25 is one’s complement enable for S and bit 26 is complement

enable for R.

"01 111000" N p4**XMERGE,** p4**MERGEI** T,I bit by bit selects bits from either R (if zero bit) or S (if one bit)

to D irrespective of data types. Bit 24 is one, bit 25 is one’s complement enable for S

and bit 26 is complement enable for R.

"01 111100" N p4**CMP,** p4**CMPI** S + R is subtracted from T,I and placed in register D

Floating-point CMP makes use of and may modify the exponent tag bits

"00 0xxx00"

TRAP

INSRT, INSRTI

SHL, SHLI

MAX, MAXI

MIN, MINI

MEDIAN

INTP-BLEND-LERP, FINTP

CMOV

"00 1xxx00"

JMPcc, CALLcc

BRcc, BSRcc

MISC (MMOV, PCND, PCB1, STM, LDM, MUX, TBD, TBD)

P3OPS

EADD, EADDI

VECT

LOOP, LOOPI

CASE, CASEI

"11 xxxx00" reserved for 48-bit instructions

1. An alternative code is a group size of one to four and an additional byte if the third bit is set, R multipliers then range for one (1 \* 1 + 0) to 17 (4 \* 4 + 1, data-size \* 1..4 + 0..1)) [↑](#footnote-ref-1)