**TROC16\_ISA**.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. Herein only the 16-bit instructions are detailed. 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, PC word alignment and Cache or DRAM line PC alignment: WIP revision needed
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 16-bit FUNCT instructions contain register indirect load instructions for each data type and a single store instruction. At this time no 24 or 32-bit instructions are supported.

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 logically 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.

**TROC16 instruction formats are:**

16-bit instruction specifies N or S, D and the op-code

sssss ddddd xxxxx1

nnnnn ddddd xxxxx1 n: -15…+15

nnnnnnnn nnnnnnnn 10000 ddddd xxxxx1 -32768…32767

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.

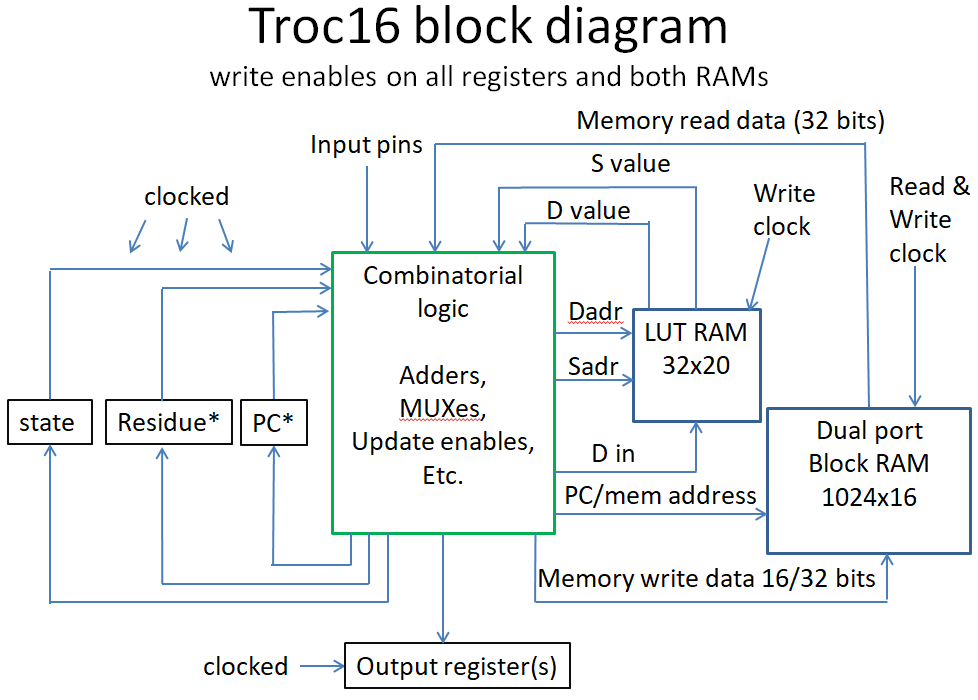
**D** and **S** are five bit register selectors.

**N** is either a five bit signed immediate value (n) or the 16-bit value (N) from the next memory location.

**I** is the **N** when used with ALU instructions, can always use either n or N.

For branch instructions n is used as a relative address and N is used as an absolute address.

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



Using two port register file

\*Residue and \*PC multiplexed onto LUT RAM outputs

**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 LD M(S)->D MUL MULI ibid

011xx1 MOV S->D ST D->M(S) DIV DIVI ibid

100xx1 BRZ ibid LDIU ibid AND ANDI ibid

101xx1 BRNZ ibid LDIS ibid OR ORI ibid

110xx1 BRP ibid LDIF ibid XOR XORI ibid

111xx1 BRM ibid LDIF2 ibid CMP CMPI ibid

**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

“10000000” p1**ALGNW** Full word align, e.g. add one to next PC and clear LSB

“11000000” p1**ALGNC** 256 bitCache line align, e.g. add 15 to next PC and clear four LSBs

**Troc16 FUNCT operations for Troc16**

00 p2SETU

01 p2SETS

02 p2SETF

03 p2SETF2

04 p2ABS

05 p2NABS

06 p2fTAG

07 p2fITAG

0C p2fLDU

0D p2fLDS

0E p2fLDF

0F p2fLDF2

11 p2fST

1C p2CVTU

1D p2CVTS

1E p2CVTF

1F p2CVTF2

(14) Unused

codes

**8-bit Instruction descriptions:** not quite right WIP

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 use 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**LDT** When *n* is outside the range from -15 to +15 load D from absolute

memory location N. Load the tag bits from location N+1. No format conversion needed.

“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.

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

memory location N and store the tag bits into location N+1.

“010001” N p2**BSR** Branch to PC + n or to 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 or to N if D is zero

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

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

“111001” N p2**BRM**  Branch to PC + n or to 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, I assumes the type code of D.

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, I assumes the type code of D.

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 I to register format

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

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

“010011” N p2**LD**  Load D from memory location S using D’s current type

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

“011011” N p2**ST**  Store D to Memory location S

“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 (uu0000wwww00mmmm). Operation is irrespective of tag bits. U is data

type of the result? W is the width of the field extracted. If W is “0000” 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 (uu0000wwww00mmmm): M is the residue shift amount, W is the insert width. If W is “0000” then width is 16.

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

“001011” N “00000” p2f**SETU** 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**SETF**  If *n* is “00010” set D type to float

“001011” N “00011” p2f**SETF2** If *n* is “00011” set D type to float2

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

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

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

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

“001011” “01100” p2f**LDU**  If *n* is “01100” Load residue with unsigned from M(D)

“001011” “01101” p2f**LDS**  If *n* is “01101” Load residue with signed from M(D)

“001011” “01110” p2f**LDF**  If *n* is “01110” Load residue with float from M(D)

“001011” “01111” p2f**LDF2** If *n* is “01111” Load residue with float2 from M(D)

“001011” ~Y “10001” p2f**ST**  If *n* is “10001” convert and store Residue to M(D)

“001011” ~Y “11100” p2f**CVTU** If *n* is “11100” convert D to unsigned

“001011” ~Y “11101” p2f**CVTS** If *n* is “11101” convert D to signed

“001011” ~Y “11110” p2f**CVTF** If *n* is “11110” convert D to float

“001011” ~Y “11111” p2f**CVTF2** If *n* is “11111” convert D to float2

“001011” “xxxxx” Room for 14 more functions

**PC in the D register field:**

The PC is made part of the register file to eliminate several distinct instructions: MOV can be used to jump to an address in a register. LD can be used to jump to an address in memory. ADD, ADDI, SUB, SUBI can be used for unconditional relative branches. LD instruction supports implementation of dispatch tables. An IN instruction to the PC can be used to return from an interrupt. There are many instructions where it does not make sense to have the PC as the D register. In these cases an illegal instruction trap is taken.

**More on interrupts:**

There are several kinds of interrupts, each with an associated branch location:

Reset: PC set to zero.

TRAP: PC set to x20 and set the save register to current PC.

BRKPT: PC set to x40 and set the save register to current PC.

Illegal: PC set to x60 and set the save register to current PC.

Undefined instruction: PC set to x80 and set the save register to current PC.

The trap locations are spaced out to allow space for one or more registers to be saved and restored and to provide space for a branch to the interrupt handler.

**More on internal registers:**

It is expected that one or more additional internal registers will be needed for configuration (such as: rounding modes, type code options and interrupt options). In particular there is need for floating point status bits (such as: overflow, underflow, inexact, NaN operand, interrupt status, etc.). A total of fifteen internal registers are possible and could be implemented as a register file that could also serve for ISA variant declaration.

**Relation to larger word size ISAs:**

It is relatively straight forward to use TROC16 as a subset of 24 and 32 bit size instructions when the word size is greater than 16. In such cases the addressing is to the byte size of the larger word size. And branch displacements are always relative. It can be anticipated that byte sizes could be 8, 9, 12 or 21 bits along with word sizes of 24, 32, 36, 42, 48 or 64 bits. A Harvard implementation could have 16, 24 and 32 bit instructions along with 16, 18, 21, 24, 32, 36, 42, 48 or 64 bit words.

The 16-bit instructions can be used along with 24 and 32-bit instructions on larger than 16-bit word sizes. The 16-bit IN, OUT, SHFT and FUNC instructions retain their behavior with shifts limited to 15-bits, e.g. “n” can be from -15 to +15. The INSRT and EXTRCT instruction N values have enough bits for up to 64-bit words. The 24 and 32-bit instructions always use variable length immediate values of any necessary length providing means to have full capability IN, OUT, SHFT and FUNC operations part of the 24-bit instruction set. The 16-bit instruction relative branches become byte displacements with the N displacement being +/- 327687 bytes and the n displacement being +/- 15 bytes.

The larger word sizes use additional tag bits to handle floating point needs of additional mantissa bits.

The intent of the TROC architecture is to support four data types, four data sizes and the same ISA across all of these options. Additionally data type tag bits simplify and unify the number of necessary instructions thereby eliminating the need for additional op-code bits.

The residue register performs the important role of supporting multiple word data sizes and a uniform approach to ALU result status.

Currently there are 36 16-bit instructions with the FUNC instruction supporting 31 single operand functions, a total of 66 instructions. The 24 and 32-bit instructions support 48 instructions each and both provide a FUNC with even greater capabilities. The 32-bit instruction set includes a 24-bit subset that provides 32 of the 24-bit instructions with three additional “option” bits one of which is typically a residue register update enable.

**Troc16 instruction selection process:**

The 24 and 32-bit instruction sizes support ~44 distinct instructions each. Among them is the single operand group with up to 32 op-codes. 20 of the 44 instructions are the load and store to/from memory. Sixteen unused 24 and 32-bit instruction codes are reserved for 40 and 48-bit instructions.

The selection process is for “useful” instructions that may or may not be implemented. The FUNC operations provide a raft of elementary functions that can be difficult to implement. For some of the three operand operations that have limited usage, they have reserved slots for those applications where they can be justified. The slots for 40 and 48-bit instructions are there for multi-media and vector operations where additional register fields are necessary.

This leaves 32 slots for 16-bit instructions. The register select fields are kept at five bits each. Troc16 is weak on load/store instructions but otherwise has a complete ISA, the initial thinking being that 24 and 32-bit load/store instructions could be supported if needed for indexed or base plus offset load/store.

The necessary instructions are the eight ALU instructions: ADD, SUB, MUL, DIV, AND, OR, XOR and CMP along with their immediate value versions. Add to this the basic conditional branches: BRZ, BRNZ, BRP and BRM. There are four load immediate instructions, one for each data type. ADDI and LDI to the PC provide the unconditional branches. There are two IO instructions IN and OUT. As it is not anticipated to have a large number in IO locations, the 16-bit N values are repurposed for absolute address load and store of a register value and its type code. And there are two other standard instructions SHFT and FUNCT. These are overloaded based on the value of “n” to provide EXTRCT and INSERT. Of the four remaining slots LD and ST are used for memory load/store. MOV is used to move a register to another location. BSR is used for subroutine calls.

At this time the SHFT, EXTRCT, INSRT and FUNC takes 77% more LUTs as without them. This is a considerable penalty for supporting a set of instructions that can be implemented in a few other instructions (mostly MULI, ANDI and ORI). AOB and SOB are currently not included in order to support the LD and ST instructions (ADDI/SUBI and BRNZ are replacements).

The single operand FUNCT group is used for type conversion and coercion as well as for several ad-hoc instructions.

By virtue of the type codes, this limited ISA also supports unsigned, signed, floating point and a TBD data type.

It was decided to use two five bit register fields in preference to three register fields on a smaller set of registers as using a smaller register file would have fundamentally changed the character of the ISA and made software compatibility with 24 and 32-bit instructions difficult.