**TROC16\_16 Documentation[[1]](#footnote-1)**

James Brakefield

TROC16\_16 is a soft core processor with 16+ bit instructions and 16-bit addressing and data. The ISA (Instruction Set Architecture) is a subset of a larger ISA that supports four data types on four data sizes each. TROC16\_16 only supports two data sizes of 8 and 16-bits. It is of the RISC variety with 32 registers of 16-bits each, plus four tag bits, two of which hold the data type and the other two, additional exponent bits. By using tag bits to hold the data type, distinct instructions for each data type are unnecessary, except for memory load instructions.

There are three sizes of instructions: 16, 24 and 32-bits. For TROC16\_16 the 16-bit instructions play a major role. There are 32 of them providing a decent yet minimal set. The 24-bit instructions have three register address fields providing **R *operation* S => D** whereas the 16-bit instructions only provide **S *operation* D => D**. The 32-bit instructions contain a fourth register address field and additional op-code bits: ***operation* (T, R, S) => D**. There are approximately forty-four 24 and 32-bit instructions each with a more complete instruction set.

Most instructions have and immediate form where the left most register field can be used as an immediate value (constant, offset, displacement). The encoding of the immediate value field includes provision for indicating additional immediate bytes such that a full size immediate can be affixed to the base instruction. For offsets and displacements the immediate is interpreted as two’s complement. For constants it is interpreted per the type code of the **D** register. For 16-bit instructions the immediate replaces **S**, for 24-bit instructions it replaces **R** and for 32-bit **T**.

At this time, instructions are half-word aligned and instruction plus immediate limited to 32-bits total. Two 16-bit instructions without immediate values will fit into 32-bits. The 24 and 32-bit instructions have a full set of load and store instructions (twenty four each) that include base register, index register scaled by data size and an offset. For 24-bit one can have either and index register or an immediate offset, not both.

The base set of ALU incudes add, subtract, multiply, divide, compare, AND, OR & XOR. Divide may or may not be implemented as it takes considerable logic and delay. The four data types are nominally unsigned, signed two’s complement, floating point and fixed point logarithm. Several varieties of floating-point are possible. For TROC16\_16 it is eight or 16-bit floats only.

Two of the 32 registers in the register file are “special”. Register 31 is PC (Program Counter) and register 28 is the “Residue” register. The residue register receives the carry and overflow from unsigned and 2’s complement add/subtract, the upper half of a multiply and the remainder of a divide. The use of the PC as a destination register is limited to where it makes sense, otherwise causes a trap.

Instruction and Immediate encodings

“Little Endian” (interpret right to left)

**16-bit op-code specifies N or S**

sssss ddddd xxxxx1

nnnnn ddddd xxxxx1

**24-bit has immediate enable bit**

rrrrr 0 xx sssss ddddd xxxx10

nnnnn 1 xx sssss ddddd xxxx10

**32-bit has three additional op-code bits**

ttttt xxx rrrrr 0 xx sssss ddddd xxxx00

nnnnn xxx rrrrr 1 xx sssss ddddd xxxx00

**16-bit instruction encoding of N**

NNNNN -15 … =15

NNNNNNNNNNNNNNNN 10000 -215 … 215-1

**Suggested 24 and 32-bit instruction encoding of N**

111NN and 10NNN -4 … +7

NNNNNNNNNNNNNNNN 0NNNN 12-bit N -211 … 211-1

NNNN … NNNN 110xx8, 16, 24 or 32-bit N

**Floating point references**

Half-precision floating-point forma, aka float16:

<https://en.wikipedia.org/wiki/Half-precision_floating-point_format>

Logarithmic Fixed-Point Numbers:

<https://ccrma.stanford.edu/~jos/st/Logarithmic_Fixed_Point_Numbers.html>

Also do google search on “*fixed point binary logarithm?”*

16-bit instructions

INN, OUTN input/output to port N or to Mem (N)

BSRN, BRZN, BRNZN, BRPN, BRMN control instructions (PC+N)

MOV move register: S => D

ADD, SUB, MUL, DIV, CMP, AND, OR, XOR basic arithmetic: S op D => D

ADDI, SUBI, MULI, DIVI, CMPI, ANDI, ORI, XORI immediate versions: I op D => D

LDUI, LDSI, LDFI, LDF2I load typed data immediate: I => D

SHFTN shift/adjust exponent: D << N => D

EXTCTN extract field, type-less: field N of D => D, N has start bit & bit width

LDTN, STTN load/store typed-data to Mem (N), uses two half word locations

24 and 32-bit instructions

LDU8, LDU16, LDU24, LDU32,

LD8, LD16, LD24, LD32,

FLD8, FLD16, FLD24, FLD32,

F2LD8, F2LD16, F2LD24, F2LD32,

LEA8, LEA16, LEA24, LEA32,

ST8, ST16, ST24, ST32,

ADD, ADDI, SUB, SUBI,

MUL, MULI, MAC, DIV, DIVI,

AND, ANDI, OR, ORI,

XOR, XORI, CMP, CMPI,

FATAN2PI, FPOW, FPOWI,

INSRT, INSRTI, EXTRCT, EXTRCTI,

ROL, ROLI, SHR, SHRI,

ASR, ASRI, SHL, SHLI,

JMPcc, CALLcc, BRcc, BSRcc,

BBS, BBC, BRccRC, JMPccRC,

MAX, MAXI, MIN, MINI,

MEDIAN, EADD, EADDI,

VECT, LOOP, LOOPI (My 66000 instructions)

TRAP, BKPT, ALIGNW, ALIGNC

Three operand only (32-bit)

MMOV, PCND, PCB1, STM,

LDM, STM, CASE, MUX, MERGE

~32 single operand (24-bit)

LDZCNT, LD1CNT, TRZCNT, TR1CNT

POPCNT, SINPI, COSPI, TANPI,

ASINPI, ACOSPI, ATANPI, …

FPGA Implementation[[2]](#footnote-2):



Detailed Instruction Descriptions

For signed and unsigned operands

INN: If N >= -15 and N <= 15: Port (N) => D otherwise reference memory location N

 N >= 0 are the input ports and N < 0 are internal registers. Sixteen bit data size.

OUTN: If N >= -15 and N <= 15: D => Port (N) otherwise reference memory location N

 N >= 0 are the output ports and N < 0 are internal registers. 16 bit data size.

BSRN: Branch and save relative. Store current PC in register D, set PC to PC + N.

 If D = PC implement as unconditional relative branch

BRZN: Branch relative if D is zero. If D = 0 set PC to PC + N otherwise go to next instruction

BRNZN: Branch relative if D is not zero. If D /= 0 set PC to PC + N otherwise go to next instruction

BRPN: Branch relative if D is positive. If D >= 0 set PC to PC + N otherwise go to next instruction

BRMN: Branch relative if D is negative. If D >= 0 set PC to PC + N otherwise go to next instruction

MOV: Move register and its type code: S => D

ADD: Add S to D: S + D => D: carry (unsigned) or overflow (signed) => Residue register

SUB: Subtract D from S: S - D => D, carry (unsigned) or overflow (signed) => Residue register

MUL: Multiply D and S: S \* D => D, upper half of the product => Residue register

 If D = PC block the write to D

DIV: Divide D by S: D / S => D, remainder => Residue register

 If D = PC block the write to D

CMP: S – D comparison result => Residue register, format TBD

AND: Logical AND of S & D => D, Residue register unchanged, type codes ignored

OR: Logical OR of S & D => D, Residue register unchanged, type codes ignored

XOR Logical exclusive OR of S & D => D, Residue register unchanged, type codes ignored

ADDI: Add I to D: I + D => D: carry (unsigned) or overflow (signed) => Residue register

SUBI: Subtract D from I: I - D => D, carry (unsigned) or overflow (signed) => Residue register

MULI: Multiply D and I: I \* D => D, upper half of the product => Residue register

 If D = PC block the write to D

DIVI: Divide D by I: D / I => D, remainder => Residue register

 If D = PC block the write to D

CMPI: I – D comparison result => Residue register, format TBD

ANDI: Logical AND of I & D => D, Residue register unchanged, type codes ignored

ORI: Logical OR of I & D => D, Residue register unchanged, type codes ignored

XORI: Logical exclusive OR of I & D => D, Residue register unchanged, type codes ignored

LDUI: Immediate value placed in D as unsigned

LDSI: Immediate value placed in D as signed

LDFI: Immediate value placed in D as float type 1, typical format is IEEE-754 Float16

 If D = PC it is an invalid instruction

LDF2I: Immediate value placed in D as float type 2, typical format is fixed point base two logarithm

 If D = PC it is an invalid instruction

SHFTN: Shift D by signed N: D << N => D. Unsigned uses logical shift, Signed uses arithmetic shift.

 If D = PC it is an invalid instruction

EXTCTN: Extract field, field N of D => D, N has 12-bit start bit & bit width format: --wwww --ssss

 Type codes ignored. Unsigned and 2’s complement sign extension.

A W of zero maps to a width of 16-bits. Bit zero is the least significant bit (LSB).

 If D = PC it is an invalid instruction

LDTN: Load typed data from Mem (N) to D, uses two half word memory locations

STTN: Store typed data from D to Mem (N), uses two half word memory locations

Selected 24-bit instructions, ds: data size in bytes

Float and Float2 undergo reformatting between memory and register

Unsigned and signed undergo zero or sign extension on loads

For TROC16\_16 immediate R aka N limited to 12-bits

LDU8, LDU16: Load 8 or 16 bit unsigned data from memory location (S + R\*ds) or (B + N) => D

 If R = PC use zero in its place

LDS8, LDS16: Load 8 or 16 bit signed data from memory location (S + R\*ds) or (B + N) => D

 If R = PC use zero in its place

FLD8, FLD16: Load 8 or 16 bit float data from memory location (S + R\*ds) or (B + N) => D

 If D = PC it is an invalid instruction

F2LD8, F2LD16: Load 8 or 16 bit float2 data from memory location (S + R\*ds) or (B + N) => D

 If D = PC it is an invalid instruction

LEA8, LEA16: Place effective memory address (S + R\*ds) or (B + N) into D

 If R = PC use zero in its place

ST8, ST16 Store 8 or 16 bit data from D into Memory location (S + R\*ds) or (B + N)

Selected 32-bit instructions, y: 1 to 4 scaling

For TROC16\_16 immediate T aka N limited to 5-bits

LDU8, LDU16: Load 8 or 16 bit unsigned data from memory location (S + R\*ds\*y+ T) => D

 If R = PC use zero in its place

LDS8, LDS16: Load 8 or 16 bit signed data from memory location (S + R\*ds\*y+ T) => D

 If R = PC use zero in its place

FLD8, FLD16: Load 8 or 16 bit float data from memory location (S + R\*ds\*y+ T) => D

 If D = PC it is an invalid instruction

F2LD8, F2LD16: Load 8 or 16 bit float2 data from memory location (S + R\*ds\*y+ T) => D

 If D = PC it is an invalid instruction

LEA8, LEA16: Place effective memory address (S + R\*ds\*y+ T) into D

 If R = PC use zero in its place

ST8, ST16 Store 8 or 16 bit data from D into memory location (S + R\*ds\*y+ T)

 If R = PC use zero in its place

Use of the PC as a general purpose register

Cases where use of PC as general purpose register makes sense:

ADD, ADDI Unconditional relative branch

MOV Jump to address, such as return from subroutine

LEA Indexed jump, better handled by a CASE instruction with index limit

LDUI, LDSI Absolute address jump

LDU16, LDS16 Table lookup jump

As an index register: replace by zero, e.g. a useful way to disable use of index register

Cases where use of PC as general purpose register does not make sense:

BSR Is a conflict of two separate writes to PC

INN Not likely to be useful

SHFTN, LDFI, LDF2, LDF, LDF2I

As an offset: could be useful if base register is not a base address

Any instruction that generates a float tag on PC

In general, prohibit these uses and take an interrupt

Mixed data type operations:

There is no requirement that operations on mixed data types be implemented

If not implemented, a suitable instruction trap is taken

Exception and interrupt mechanisms

There is a need for:

A return address register for interrupt routines

A configuration register which enables optional features and rounding modes

An interrupt enable register

An interrupt pending register

An interrupt state, e.g. the current interrupt cause number, zero in non-interrupt mode

A dispatch table (probably dedicated memory area) and numbering of interrupt causes

Cache, virtual memory and protection mechanisms TBD

Unimplemented instructions and invalid operands

 To lead to exceptions as determined by configuration register

 For the time being for testing purposes these cause a halt (freeze of the PC)

Preferred implementation is a small register file with configuration always visible

 And interrupt enables usually visible (8-bits wide with three read ports?)

 INN/OUTN instructions can read/write any of the internal registers

1. “A Digital Processor of RISC Variety Suitable for Architecture Exploration” presentation slide deck is available at web pages: <https://events.vtools.ieee.org/m/467923> and <https://github.com/jimbrake/TROC16_16> [↑](#footnote-ref-1)
2. VHDL source code at: <https://github.com/jimbrake/TROC16_16> [↑](#footnote-ref-2)