Register Oriented Computer Architecture (ROC) Description

James Brakefield ©2020

Preface:

ROC is part of a family of computer architectures. There are commonalities within the family: Little Endian, A register file, typically of 16 or 32 registers, and a main memory. The register bit length defines the largest data size and the maximum addressable main memory. Each family member supports four data sizes. The smallest addressable memory unit is called a byte and can be eight, nine or twelve bits in size. Distinct instruction sets exist for each member and each instruction set provides instructions of several sizes.

Immediate and displacement values needed by an instruction have a byte length field of three bits within a five bit register field and the immediate value follows the instruction (for the small values of -8 to 15, the value is coded within the instruction).

The family currently includes a RISC machine (ROC32_8), a stack/accumulator machine (STAOC32_8) and the multi-denominational ALT-A machine (combining stack, RISC and x86 style ISAs within a single architecture). Closely related are alternative variations on the MSP430[1], the VAX and the x86 (alt_430, alt_vax and alt_x86) ISAs which use register files of 16 registers.

Introduction:

ROC has several variants that could be implemented. Byte size could be 12-bits leading to possible data sizes of 12, 24, 36 and 48 bits. More commonly byte size is 8-bits and the four data sizes are 8, 16, 24 and 32 bits; or 8, 16, 32 and 64 bits. The nomenclatures for these variations are: ROC48_12, ROC32_8 or ROC64_8. Herein ROC32_8 will be described.

ROC64_8 and ROC32_8 have four instruction sizes of 24, 32, 40 and 48 bits with any immediate value following the instruction. An immediate can be of any size from one to eight bytes. In general, instructions are byte aligned, however by choosing a suitable combination of instruction and immediate lengths, two or four byte alignment can be maintained.

ROC has a 32 entry register file. Register zero is used to store the return address on subroutine entry. When used as an index register in load/store instructions it reads as zero. The program counter (PC) is available in register 31, register 28 is the default location for the condition code register, and by convention, register 29 is the frame pointer and 30 is the stack pointer. Typically subroutine arguments are passed in the low numbered registers and results returned in their place.

The ROC ISA has four instruction sizes and each larger size supports additional features with the register fields retaining their locations in larger sizes. The first two bits of the instruction indicate the instruction length. The remaining six bits of the first byte are the op-code.

[1] MSP430 is a trademark of Texas Instruments

The three byte long instructions take two operands from the register file and return one result. There is one bit used to indicate the second operand is an immediate. When second operand register field is an immediate, the five bit register field indicates a value from -8 to 15 or that a one to eight byte immediate follows the instruction. The three byte long load and store instructions are either base plus data size scaled index or base plus immediate displacement.

The four byte long instructions add a predicate generate bit that updates a condition code register and add a five bit condition select field to cause the instruction to conditionally execute. The remaining two bits of the fourth instruction byte are used to individually negate or invert (binary complement) the two operands. When the second operand is an immediate the second negate-invert bit becomes the operand swap enable.

Four byte load and store instructions depart from the above: add an additional register field to the three byte load and store instructions. Add a two bit index register scale factor field and a condition code update enable bit. The four byte store instructions replace the condition code update enable with a store immediate enable, so the store immediate instructions can have two independent immediate specifications: the address displacement and the value to be stored.

<u>Instruction Formats:</u>

Three byte with two operands	`00xxxxxx ddddd i rrrrr sssss`
Three byte with one operand	`00xxxxxx ddddd z rrrrr xxxxx`
Three byte with immediate	`00xxxxxx ddddd 1 rrrrr nnnnn`
Three byte load/store with index register	`00xxxxxx ddddd 0 bbbbb jjjjj`
Three byte load/store with displacement	`00xxxxxx ddddd 1 bbbbb nnnnn`
Three byte conditional branch/load imm	`00xxxxxx ccccc n nnnnn nnnnn`
Four byte, two operands with predication	`01xxxxxx ddddd i rrrrr sssss zz u ccccc`
R op S ->D; execute on predicate match	`01xxxxxx ddddd 0 rrrrr sssss zz u ccccc`
Four byte, three operands	`01xxxxxx ddddd i rrrrr sssss zz u ttttt`
Four byte with immediate	`01xxxxxx ddddd 1 rrrrr nnnnn zv u ccccc`
Four byte conditional branch/load imm	`01xxxxxx ccccc n nnnnn nnnnn nn n nnnnn`
4 byte load/store	`01xxxxxx ddddd i bbbbb sssss yy u jjjjj`
4 byte store immediate	`01xxxxxx mmmmm i bbbbb sssss yy 1 jjjjj`
5 byte with 3 source registers	`10xxxxxx ddddd i rrrrr sssss zzuccccc xxxttttt`
Six byte with 3 source registers, predication and 2 destination registers	`11xxxxxx ddddd i rrrrr sssss zzuccccc xxxttttt xxxeeeee`
Six byte with 4 source registers, predication and 2 destination registers	`11xxxxxx ddddd i rrrrr sssss zzuccccc xwwttttt wwweeeee`

<u>Bit field codes:</u>

00, 01, 10 or 11 Instruction length bits

B	Base address register, the condition code register as base register reads as zero
C	Five bit predicate condition code (if condition matches, instruction is executed)
D	Destination or result register, exception: store instructions

E	Second destination or result register, replaces My 66000[2] Carry mechanism
I	Immediate flag bit, enables S to be a short constant (N) or an immediate length specification
ISZ	Part of the immediate bits (N), used to indicate number of following bytes of immediate
J	Index register; register zero, the return address, reads as zero
M	Value/byte-length for the store immediate value
N	Immediate or offset field or byte length of suffixed immediate
T	Third source register, for My 66000 Carry mechanism or three operand instructions
X	op-code bits
U	Update condition code register, register 28
V	Swap operand with immediate
W	Fourth source register
Y	Index register scaling: Data-size * (1, 2, 3 or 4)
Z	Invert/negate operand, one bit per register operand

[2] My 66000, a computer architecture by Mitch Alsup, frequent contributor to comp.arch on usenet.

Instruction Definitions

Load/store instructions:

24 and 32-bit load and store instructions are defined. There are several variations. There are load signed, unsigned and floating-point for four data sizes: 8, 16, 24 and 32-bits. There is also load immediate signed and load effective address. And store immediate which can have two immediate fields, first for the immediate and second for an address offset. Both fields can be of any length from zero(value encoded in the “nnnnn” field) to eight bytes (immediate follows and is considered part of the instruction.

LD8, LD16, LD24, LD32	load two’s complement sign extended
LDU8, LDU16, LDU24	load two’s complement zero extended, LDU32 is same instruction as LD32
FLD8, FLD16, FLD24, FLD32	load floating-point, format is specific to each data size
LEA8, LEA16, LEA24, LEA32	load effective address
LDI	load signed immediate

In all cases the memory value or address value is placed into the D register.

Three byte load/store with index register	`00xxxxxx ddddd 0 bbbbb jjjjj`
Three byte load/store with displacement	`00xxxxxx ddddd 1 bbbbb nnnnn`
Three byte load immediate	`00xxxxxx ddddd n nnnnn nnnnn`
Four byte load with index and offset	`01xxxxxx ddddd i bbbbb sssss yy u jjjjj`
Four byte load immediate	`01xxxxxx ddddd n nnnnn nnnnn nn n nnnnn`

The corresponding stores, which store the truncated or rounded D register contents into memory:

ST8, ST16, ST24, ST32 for store two’s complement truncated register contents into memory
STF8, STF16, STF24, STF32 for store floating-point rounded register contents into memory
STI8, STI16, STI24, STI32 for store signed immediate value into memory
STFI8, STFI16, STFI24, STFI32 for store floating-point immediate value into memory

3 byte store with index register	`00xxxxxx ddddd 0 bbbbb jjjjj`
3 byte store with displacement	`00xxxxxx ddddd 1 bbbbb nnnnn`
4 byte store	`01xxxxxx ddddd i bbbbb sssss yy 0 jjjjj`
4 byte store with imm. offset	`01xxxxxx ddddd 1 bbbbb nnnnn yy 0 jjjjj`
4 byte store immediate	`01xxxxxx mmmmm i bbbbb sssss yy 1 jjjjj`

The four byte store immediate instructions use the same op-code as the four byte store instructions but interpret the “u” bit as a store immediate enable.

As the PC register is mapped to register file register number 31, the load and load effective address instructions can be used to jump or branch to a new address. Likewise the conditional branch/jump instructions can be interpreted as conditional load of the PC. Rather than have the conditional jump/branch “never” instruction be a NOP it is instead used as a CALL instruction saving the next instruction address in register zero.

3 byte conditional branch relative	`00xxxxxx ccccc n nnnnn nnnnn`
3 byte conditional jump	`00xxxxxx ccccc i bbbbb jjjjj`
4 byte conditional branch relative	`01xxxxxx ccccc n nnnnn nnnnn nn n nnnnn`
4 byte conditional indexed jump	`01xxxxxx ccccc i bbbbb sssss yy x jjjjj`

Two operand instructions:

The 24-bit two operand instructions provide two source registers of which one can be an immediate value. The 32-bit two operand instructions augment the24-bit instructions with a negate/complement bit for each operand, a status register update enable and a five bit predication condition (execute instruction if predicate matches that in the status register, AKA register 28).

Three byte with two operands	`00xxxxxx ddddd i rrrrr sssss`
Three byte with immediate operand	`00xxxxxx ddddd 1 rrrrr nnnnn`
R op S ->D; four byte generic	`01xxxxxx ddddd i rrrrr sssss zz u ccccc`
R op S ->D; four byte with immediate	`01xxxxxx ddddd 1 rrrrr nnnnn zv u ccccc`

The use of register 32, AKA program counter (PC) as a destination register is prohibited (causes an illegal instruction trap) where it does not make sense. Likewise the use of register 31 as a source register is prohibited likewise. The four byte instructions with immediate operand repurposes the second operand negate/complement enable as a swap operand enable.

The following two operand instructions have 24-bit encodings:

ADD, ADI	Two’s complement addition
ADC, ADCI	Two’s complement add with carry
AND, ANDI	Boolean bitwise AND
OR, ORI	Boolean bitwise OR
XOR, XORI	Boolean bitwise exclusive or
CMP, CMPI	Compare, condition code results can be placed in the status register or elsewhere
SUB, SUBI	Two’s complement subtract
SBC, SBCI	Two’s complement subtract with carry
MUL, MULI	Two’s complement multiplication, also serves as unsigned binary multiplication
MULU, MULUI	Unsigned binary multiplication upper half
MULUS, MULSI	Signed binary multiplication upper half
DIVU, DIVUI	Unsigned binary division quotient
DIVS, DIVSI	Signed binary division quotient, remainder is unsigned
FADD, FADI	Floating-point addition
FSUB, FSUBI	Floating-point subtract
FMUL, FMULI	Floating-point multiplication
FDIV, FDIVI	Floating-point division quotient
FCMP, FCMPI	Floating-point compare, condition code results can be placed in the status register

For shift instructions shift amount is modulo 32. Second operand has shift amount or field length/position.

ROL, ROLI	Rotate left
SHR, SHRI	Shift right unsigned (zero fill)
ASR, ASRI	Shift right signed (sign fill)
SHL, SHLI	Shift left, zero fill (serves both signed and unsigned two's complement)

For the insert/extract instructions the second operand is a combined and concatenated field length and field position, both five bits. A field length of 32 is indicated by five zero bits.

INSRT, INSRTI	Insert truncated source operand into destination register field
EXTRCT, EXTRCTI	Extract and zero extend specified field from source
EXTRCTS, EXTRCTSI	Extract and sign extend specified field from source

For the above two operand instructions the 32-bit versions omit instructions covered by operand negate/complement: SUB, SBC. The operand complement enables allows the AND, OR and XOR to provide a complete set of two operand Boolean/bitwise operations.
As negate/complement on an immediate operand is unnecessary, the unused enable is repurposed as an operand swap enable.

One operand instructions:

3 byte with one operand	`00xxxxxx ddddd z rrrrr xxxxx`
4 byte with one operand	`01xxxxxx ddddd z rrrrr xxxxx xx u ccccc`

Immediate operand not supported. I bit repurposed as the Z bit indicating operand negation.
The S field is repurposed as five op-code bits allowing 32 one operand operation codes.
The four byte version allows 128 one operand operation codes by repurposing the Z bits.

Three byte one operand instructions (25):

IN	Input from IO port, R is port number
OUT	Output to IO port, R is port number
LDZCNT	Leading zero bits count, zero to 32
LD1CNT	Leading one bits count, zero to 32
TRZCNT	Trailing zero bits count, zero to 32
TR1CNT	Trailing one bits count, zero to 32
POPCNT	Count of one bits, zero to 32
SINPI	Floating-point sine of angle in rotations
COSPI	Floating-point cosine of angle in rotations
TANPI	Floating-point tangent of angle in rotations
ASINPI	Floating-point arcsine giving an angle in rotations
ACOSPI	Floating-point arccosine giving an angle in rotations
ATANPI	Floating-point arctangent giving an angle in rotations
INT	Convert floating-point number to integer
FLT	Convert integer to floating-point
EXPON	Extract exponent from floating-point number
FRACT	Extract fraction and hidden leading one bit from floating-point number, unsigned

SQRT	Integer square root
FSQRT	Floating-point square root
FRSQRT	Floating-point reciprocal square root
FRCP	Floating-point reciprocal
LN2P1	Floating-point log base 2 of one plus the operand
LN2	Floating-point log base 2 of the operand
EXP2M1	Floating-point base 2 exponential of operand, result has 1.0 subtracted
EXP2	Floating-point base 2 exponential of operand

Four byte one operand instructions:

All of the three 3-byte one operand instructions (25X).

Logs and exponentials to base E and base 10 (8X)

Trigonometric functions with angles in radians (6X)

CVTnn	Convert between representations (~48X)
RNDnn	Directed roundings (6X)

Three operand instructions:

By not having the predication byte on 32-bit instructions three operands can be supported. A list of three operand instructions follows. It is TBD which of the three operand instructions will have a 32-bit version. Expectation is to use a 40-bit instruction which has both a third register field and a predication byte. The three unused bits in the 40-bit instruction will be used to provide additional variations on the operation list; typically signed versus unsigned, third operand negate/complement and floating-point rounding mode.

4 byte, three operands `01xxxxxx ddddd i rrrrr sssss zz u ttttt`

5 byte with 3 source registers `10xxxxxx ddddd i rrrrr sssss zz u ccccc xxx ttttt`

INSRT, INSRTI	Insert truncated source operand into third source register field
LUT	Treat two or more consecutive registers as binary lookup tables
CMOV	Use condition code to determine which of two operands is moved to the result
MERG	Boolean merge; use the third operand as a binary mask

The following have floating-point versions:

MEDIAN, MEDIANI	Signed two's complement median
MAX3, MAX3I	Three operand maximum
MIN3, MIN3I	Three operand minimum
ADD3, ADD3I	Three operand addition
LERP, LERPI	Linear interpolation
FMAC, FMACI	Floating-point multiply accumulate

Special instructions (which don't fit elsewhere):

BBS, BBC Branch bit set, branch bit clear (relative or absolute).

3 byte with two operands `00xxxxxx ddddd x rrrrr nnnnn`

D is the register to examine and R is the bit to examine. Remaining register field is always immediate and gives either the relative displacement or the absolute address depending on the I bit.

BRK	24 and 48 bit instructions of which only the first byte matters. When executed causes a breakpoint exception. The first byte is coded as all zeros (24-bit inst.) or all ones (48-bit inst.).
ALGN, ALGN32	48 bit instructions of which only the first byte matters. Causes next instruction to be cache block aligned or 32-bit word aligned.
NOP	One byte instruction, not sure if needed?
TBD	Exception and interrupt handling, virtual memory mapping

Rare instructions (which don't get used often and therefore can be placed in longer instructions):

REMU, REMUI	Unsigned remainder from unsigned divide
REMS, REMSI	Signed remainder from signed divide
FTAN2PI, FTAN2PII	Floating point two argument arc-tangent, result in rotations
FPOW, FPOWI	Raise first argument to the power of the second argument
EXADD, EXADDI	Add integer to floating-point exponent
STM, LDM	Store/Load multiple registers to/from memory
MMOV8	Memory to memory byte move

Micro-Architecture

Whereas the overall architecture can be conveyed in the ISA description, the micro-architecture is the implementation design. Herein the implementation is into a current FPGA family such as the Xilinx series-7. The Altera/Intel Cyclone V is similar in capability. The logic elements are six input lookup tables (6LUT). The families either offer a small block RAM (typically 32 entries of 20 bits) or LUT RAM using grouped LUTs. These RAMs offer asynchronous reads and synchronous writes. There can be multiple read ports and at most one write port. These RAMs find use as register files. Associated with the LUTs are one or two flip-flops. They can be used as latches but such usage is difficult and most usage is as D flip-flops.

Block RAMs offer 9k to 36k bits of memory each. They have two ports each with synchronous read and write as long as each side writes to different locations. Typically there is a ninth bit for each byte. The block RAM data width is configurable from one to 36 bits with each byte having a write enable (available data widths of 1, 2, 4, 8, 9, 16, 18, 32 or 36 bits).

DSP units have an integer accumulator and an integer multiplier. Altera/Intel multipliers are 18 by 18 and signed. Xilinx series-7 multipliers are signed 18 by 25. The resulting product is the sum of the lengths of the inputs (36 or 43 bits). The accumulators are 64 and 48 bits. The accumulator logic supports operand negation/complement and a full set of two input Boolean operations. Both vendor software tools (Quartus and Vivado) support IP (intellectual property) generators for floating point or integer multiply and/or add to 64-bits. The DSP units are ASIC implementations and provide very fast multiplies and adds. They are designed to be (but need not be) pipelined and to operate at maximum clock rates.

Given the above brief description of FPGA resources and a need for a rapid prototype the micro-architecture goes as follows:

For initial experimentation no pipelines are implemented. A single clock is used. The code is written in VHDL. The 32-entry register file is composed of LUT RAM with four read ports and one write port (writes are to the fourth read address). Program and instruction memory is a single block RAM configured as 32-bits by 1024 words. A byte address is generated and that address and that address plus four is applied to the two halves of the block RAM.

In a single clock cycle an instruction is read from block RAM, shifted into alignment and the immediate if there is one and it is encompassed by the 64-bit read, also shifted and aligned. The register address fields are applied to the four LUT RAM address ports. The register contents and the immediate value are forwarded to the operation logic and DSP units. The result is clocked back into the register file at the next clock. This results in an instruction rate of one per clock if the full immediate is available and the instruction is not a load or store. In any of these three other cases additional block RAM memory cycles occur via a simple state machine.

The expected performance will decrease as additional instructions are implemented. Some instructions require considerable combinatorial logic. The reason for the “single cycle” approach is that each instruction can be individually enabled or disabled and the resulting logic delay and LUT utilization determined. For example, divide is slow and resource (LUT) intensive. Once its LUT and delay characteristics are determined a decision can be made to either: implement via subroutine, peripheral function unit, pipeline, multi-cycle timing constraint or micro-code (more elaborate state machine).

The VHDL code consists of RTL for the register file, the instruction/data memory, the register updates and the combinatorial logic. Each instruction is an entry in a large case statement. A separate source file contains the numeric (Boolean) values of each op-code. The different instruction sizes can be distinct case statements or merged, which ever works best. The single operand instructions have their own case statement as a part of the overall case statement.

Once an initial design is coded and working (e.g. no syntax errors or major warnings) some test instruction code is written and the design simulated. After correct simulation the design can be converted to a bit-map file and downloaded to an actual FPGA board. Currently in possession of Altera Cyclone V board, a Xilinx Artix-7 C-mod board and a Xilinx ultra96 board (and other boards as well). The Altera and Xilinx software tool infer LUT-RAM and block RAM. The same code (RTL) can be made to run on any of these boards.