One Operand Stack Machine (OOSM16_3sz) ISA

James Brakefield ©2023

Comments:

An older version, ASTOC11_16, has much the same ISA. However, very few spare op-codes exist. In order to generate op-code space I have added bits to the stack registers that encode data type. Herein the data types are: unsigned, signed, float and Posit. The additional register bits allow for two more types which could be "address" and "undefined". Alas, the new ISA also fills up the op-code space and some pruning will be necessary.

I have added one byte op-codes for the "traditional" Forth stack operators. This is redundant as the two byte op-codes form a superset. Pruning needs will determine the fate of the one byte op-codes.

Full documentation of the instruction set and encoding is not yet available. In order to achieve an initial implementation in VHDL much simplification will occur and many possible features, such as binary operators accepting all combinations of operand types, will generate exceptions.

Preface:

OOSM16_3sz 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 three or 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, herein 8, 16, 24 and 32-bit instructions. Increased instruction sizes allow for additional operand fields and additional operation codes.

Immediate and displacement values needed by an instruction have a byte length field of two or three bits within the instruction and the value follows the instruction (for small values the value is coded within the instruction).

The family currently includes a RISC machine (ROC32_8), a stack/accumulator machine (OOSM16_3sz) 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, the PDP-11, the VAX and the x86 (alt_430, alt-11, alt_vax and alt_x86) ISAs which rework the existing ISAs into formats of greater regularity[1][2][3]. Surprisingly 24-bit instruction formats dominate with escapes to longer instruction formats.

[1] The alt_430 and alt_11 designs repurpose the op-code space for the decimal and floating-point instructions as an escape mechanism to longer (24-bit) instructions that allow for more op-codes and more data sizes. A larger register file for alt-11 made possible by reducing the number of address modes.

[2] The alt_vax design limits instructions to a single memory reference location and limits the addressing modes.

[3] The alt_x86 design reorganizes the addressing modes and their encoding and supports a 16 entry register file.

The commonalities among the families are: little endian bit layout in registers and memory, provisions for variable size immediate fields such that all byte lengths are available within any instruction needing an immediate value or relative or absolute address, byte addressing of instructions with the ability to choose instruction and immediate sizes to maintain half-word or full word alignment, and with the majority of operand and result accesses being satisfied by the register file and therefore allowing single cycle execution of simple instructions.

Additionally the register file includes a register for indexed memory access, a register for ALU residue[4] and possibly the top-of-data-stack register, the program counter. To limit the number of register write ports to one, these registers may exist as discrete registers multiplexed into the register file read paths.

Introduction:

OOSM family 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: OOSM 48_12, OOSM 32_8 OOSM 64_8. Herein OOSM16_3sz will be described. The majority of instructions are 16-bits and three data sizes are supported. To conserve op-code space the register file uses four additional bits to encode data type and additional bits for floating point numbers. The data types are: unsigned binary, signed two's complement, address, floating-point and posit[5]. The four additional bits are not saved in general purpose memory but are saved in the stack/register spill/refill area.

OOSM has four instruction sizes of 8, 16, 24 and 32 bits with any immediate value following the instruction. An immediate can be of any size from zero to eight bytes. An immediate of -4 thru 7 is encoded within the instruction. A signed immediate of twelve bits takes an additional byte. In general, instructions are byte aligned, however by choosing a suitable combination of instruction and immediate lengths 16-bit or 32-bit alignment can be maintained.

OOSM uses a register file for data and return stacks, frame area and a global/thread area. With current FPGA technology stacks can be from 32 to 256 registers using LUT RAM, or at additional delay, in Block RAM. For embedded applications this may be sufficient and allows a simple single cycle design with one instruction executed per clock. Using Block RAM instead of LUT RAM for the stacks/register file with a lookaside mechanism should be doable within a high throughput, pipelined design[6].

The OOSM ISA has four instruction sizes and each larger size supports additional features. The one byte instructions are the usual stack operators that take two operands from the data stack and return one result. There is room for 64 such instructions. Two byte instructions have a five bit field for a stack

[4] The residue register is my name for a register receiving carry and/or overflow from add or subtract and for receiving the MS half of a multiply or the remainder of a divide or square root or the floating point difference between the floating-point result and the exact result. The mechanism is inspired by Mitch Alsup's My 66000 design "Carry" register.

[5] The 16 codes assigned: one each unsigned, signed, address, reserved, four codes for floats and eight codes for Posit floats. Formats and representations for floats and Posits are non-standard.

[6] Stack spill and restore mechanism is planned to use a FPGA Block RAM such that the instruction accessible registers are located in the register file and spill/restore is handled automatically.

reference, a one bit field for stack pop/push, a one bit field for replace operations (result goes to the stack reference), and a bit field to indicate an immediate instead of a stack reference. The single operand field provides access to 32 registers that is mapped to the data stack, the return stack, the frame area or a global area. Additionally the operand reference includes its size and memory address mode. There is code space for 16 16-bit instructions. The three byte instructions add a second register file reference and provide five op-code bits. The four byte instructions provide eight op-code bits and a third operand field.

The stack reference in the short format (two byte instruction) provides access into the data stack or the frame area (24 locations total) and 4 locations each into return stack and global registers.

The five bit stack reference field when used as an immediate uses the 32 possible values in four ways: A zero to seven positive immediate, a minus one to minus four negative immediate, a twelve bit signed value using an additional following byte, or to indicate that a one to four byte immediate follows the instruction.

OOSM16_3sz Instruction Formats[7]:

Little Endian Instruction Formats (*nnnnn* is a register field[8], *ssmm* is a size and address mode[9]):

Single byte stack instruction generic (31 codes)				`000xxxxx`
Single byte load following immediate (5 types and 3 sizes)				`0010ttss`
Single byte load from memory (5 types and 3 sizes)				`0011ttss`
Single byte store to memory (3 sizes)				`00101111, 00111111, TBD`
Two byte generic (the dominate instruction format)			`nnnnnsmm`	`1xprsxxx`
Two byte memory reference with short offset		`iiiiiiit`	`nnnnnsmm`	`1xprsxxx`
Two byte with immediate			`iiiii111`	`1xpr1xxx`
Two byte with relative branch			`nnnnnnnt`	`11sxx110`
Two byte miscellaneous			`iiiiixxw`	`11xxx111`
Three byte generic		`nnnnnxxx`	`nnnnnsmm`	`01prs0xx`
Three byte with immediate		`nnnnnxxx`	`iiiii111`	`01pr10xx`
Four byte generic	`nnnnnxxx`	`nnnnnxxx`	`nnnnnsmm`	`01prs1xx`
Four byte with immediate	`nnnnnxxx`	`nnnnnxxx`	`iiiii111`	`01pr11xx`

X	op-code bit
P	push/pop TOS (meaning depends on R bit: for R=0 push result, R=1 pop TOS)
R	Replace operation to stack or memory reference if set
N	Five bit stack reference field or immediate bit field

[7] An increase to four data sizes or more addressing modes requires significant rearrangement of the instruction formats or the elimination of one address mode. A decrease to two data sizes is easily handled.

[8] N allows access to the stack tops and to registers near the tops and to a global area and the frame area.

[9] SSMM encodes three data sizes and five address modes: register reference, indexed indirect memory reference, indirect memory reference with variable length displacement, indirect memory reference with full displacement and an address mode TBD. This uses 15 codes, the remaining code signifies a variable length immediate, relative address or absolute address.

S	Immediate 2’s complement sign bit
SSMM	Four bit code for 3 data sizes and 5 address modes, 1111 indicates an immediate
T	Comma code for longer relative address (additional bytes follow until t=1)
W	I field specifies register rather than an immediate value
U	Update condition-code register
V	swap operands enable, where useful, additional op-code bit otherwise
C	Five predicate condition bits (if condition matches instruction performed otherwise a NOP)
I	Immediate field

	`00iii`	immediate of 0 thru 7
	`011ii`	immediate of -4 thru -1
`iiiiiiii`	`1iiii`	signed immediate of -2048 to 2047
	`010nn`	signed immediate of one to four bytes follows

The two byte OOSM16_3sz Instructions (16 max + P&R bits)

Five LoaD instructions: one for each data type, data size is encoded within instruction
One STore instruction
Five ALU instructions: ADD, SUBtract, MULtiply, DIVide, COMpare
Three Boolean instructions: AND, OR, XOR
Relative addressing: BRanch, Conditional branch (BRZ, BRNZ) and CALL instructions[10]
Miscellaneous instructions: op-code space: 16 codes and N field:[11]
Shift instructions, Input and output port read/write instructions,
Loop setup and iterate, frame area save/restore/adjust, subroutine return
All but last two instruction codes have the P (push/pop) and R (replace operand) bits.

One byte OOSM16_3sz Instructions (64 max)

Single byte load/store instructions (33 codes):
Load immediate (15)
Load from memory address (15)
Store to memory address (3)
Single byte instruction generic (31 codes, load/store/literal already covered):
    The classic Forth op-codes including DUP and SWAP[12]

Three byte OOSM16_3sz Instructions (32 max + P&R bits)

Normally encodes three operand instructions (MAX, MIN, MEDIAN, 3ADD, MULTADD, INSERT, MUX, SELECT, CASE) and single operand instructions such as transcendental functions and type conversions

Four byte OOSM16_3sz Instructions (256 max + P&R bits)

[10] A comma code would provide variable length relative address capability?
[11] Could use one bit to select between N values or register value?
[12] See https://users.ece.cmu.edu/~koopman/stack_computers/sec3_2.html section 3.2 (14 stack machine primitives) and Appendix B (67 Forth primitives, includes 10 double precision and 7 literal related operations) of *Stack Computers: the new wave* Philip J. Koopman, Jr. 1989.

Full instruction set with additional op-code bits for operand sign changes, floating point rounding modes, operand swap, TBD

Floating-point:

In a departure from IEEE 754 floating-point the memory representation of denorms uses a trailing zeros code to indicate the degree of denormalization. The same format also provides for gradual overflow in a manner similar to denorms AKA gradual underflow. Rational is that decoding the trailing zeros code is simpler than shifting the mantissa AKA fraction. The register format includes an additional exponent bit so that the underflow/overflow values have a complete binary exponent. And an additional bit is used to indicate exact/inexact. Inexact values are rounded to unbiased nearest odd. Underflow and overflow values are considered inexact. The rules for operations using inexact zero are TBD.

Posit floating-point[13]:

In a departure from the Posit standard, left to right, the bit regime bits follow the fraction. There are two exponent bits. Thus there are, left to right, fraction and exponent sign bits, two exponent bits, the fraction with implied leading one bit, a one bit, the trailing zero bits and the inexact bit.

For the register representation the trailing zeros count is prefixed to the exponent bits. To support the maximum possible fraction accuracy three more bits are required than register size.

2's complement representation and exponent representation (signed, offset binary etc) are TBD. There is one code for zero and one code for "not a number". The default rounding mode is unbiased round-to-odd[14]. Eight and 16-bit posits are always considered inexact. 24/32-bit posits contain an inexact bit. The rules for operations using inexact zero are TBD.

Exceptions and interrupts:

Exception and interrupt enables are via an IO port register.
Exception and interrupt cause is written to an IO port register and the return address likewise.

[13] https://posithub.org/docs/posit_standard-2.pdf Standard for Posit™ Arithmetic (2022).
[14] https://www.lri.fr/~melquion/doc/05-imacs17_1-article.pdf Web search on: "floating point round to odd".