ROC36_24, 42_24 & 48_24[1]

Using the same ISA for different word sizes

A Register Oriented CPU for 24, 32, 36, 42, 48 or 64-bit word size

James Brakefield

Preface

A concise description of the digital processor families presented herein:

The register size sets the address range and the standard word size. A general purpose register file of 32 registers is provided. The processor is, for now, Harvard architecture with a 24 and 32-bit instructions within an 8-bit byte addressable instruction memory. The data memory is based on the register size with the byte size being the smallest sub-multiple of the word size (36: 9-bit bytes, 42: 10-bit bytes[2], 48: 12-bit bytes). The external memory is 8-bit byte addressable and memory transfers between external memory and internal/cache memory is in 256-bit aligned chunks. Seven 36-bit words fit into 256-bits with 1.6% waste, six 42-bit words also with 1.6% waste and five 48-bit words with 6% waste[3][4]. Data transactions with external memory require a divide by 7, 6 or 5 to select the correct 256-bit external memory word.

The basic ISA supports up to four data sizes and four data types. There is a register file of 32 registers, one of which is the "residue register". It serves to hold the carry and overflow from add and subtract instructions. It also gets the upper half of a multiply and the remainder of a divide. For floating point it gets the round-off error.

The basic ISA allows for variable length immediate values encoded as needed within or following each instruction. These immediate values may be small constants, memory offsets, floating point values and memory addresses. By using a longer than necessary length immediate values, 16 or 32-bit instruction alignment can be maintained.

The basic ISA supports three operand instructions; such as a three operand integer add or subtract. This allows use of the residue register instead of add with carry instructions.

---

[1] For ROCnn_mm nn is the word size and mm is the typical instruction size. TROCnn_mm is the name for the variant that maintains a data type code for each register in the register file.

[2] 21 is not evenly divisible by two, therefore 10-bit loads and stores do not affect the 21st bit??

[3] It can be argued that 40 and 48-bit word sizes would accomplish the same goals without departing from an 8-bit byte size, and would give a seamless interface to external memory. Indexed loads and stores would require multiply by five or three plus scaling by a power of two. Such is faster than divide by small odd primes. And there would be no memory wastage. Due to the support of four or more data sizes in the ISA, such data sizes could be supported for 32 or 64-bit word sizes? The issue of unaligned loads and stores remains and is present in all designs herein.

[4] The Burroughs computers used a 48-bit memory word each with three tag bits. 51 divides 256 with one unused bit.

The basic ISA provides for data types of: unsigned integer, signed integer, floating-point and Posit.

Philosophy

The digital processor is the epiphany of the perfect machine. If well designed and implemented, which is normal, they run error free, have perfect repeatability and are not damaged by faulty programs. Calculations can be run with sufficient accuracy to limit digital noise to that in the source data or that due to the algorithms. The wear-out mechanisms are well understood and can be managed to give decades of error free operation. In contrast to mechanisms, the programmable computer can be configured via software to subsume the great variety of mechanical devices. It is capable of modeling in real-time the behavior of most mechanisms from rocket engines to everyday manufacturing.

Progress in integrated circuit technology allows operation of digital processors at billions of operations per second; many orders of magnitude faster than limited accuracy analog operations and still more orders of magnitude faster than mechanical mechanisms.

Prologue

DRAM memory chips are oriented for 8-bit bytes and binary addressing. Any viable computer architecture is constrained to work with these chips individually or in the form of memory SIMM modules. ASIC and FPGA internal memories are not limited to 8-bit bytes.

The design space for digital processors is vast. For today's technology binary addressing and register file based CPUs are strongly dominate. Also dominate are two's complement integers and IEEE-754 floating point numbers.

The Business Case

The use of the "right' bit size for data improves memory capacity over, say, 64-bit data. This improves data cache performance and reduces gate count, allowing move cores per chip, lower power per core and possibly faster operation.

There are applications with well proven software that run on legacy computers that do not use 8-bit bytes and/or power-of-two word sizes. E.g. legacy support for 36 and 48-bit word size computers.

The use of 24-bit RISC instructions provides improved code density over 32-bit RISC ISAs. 32-bit instructions are provided for three operand instructions and more options. Better code density also improves instruction cache performance.

The design herein targets the embedded market where code density, MIPS, chip size and power matters. Code density is 33% better than 32-bit RISC. Compute power and chip area have a 33% to 75% advantage. At the expense of chip size and power, subsets of the design supporting 32-bit instruction alignment are compatible with high performance architectures.

The use of Harvard architecture allows the exact same machine code on all word sizes.

Introduction

This is research computer architecture with many goals venturing into several novel territories.

A) Same ISA across all word sizes
    a. For a Harvard architecture, the exact same encoding
B) 24-bit instructions for better code density
    a. Three full five bit register fields, an immediate flag and an 8-bit op-code
C) Variable length immediate values for better code density
D) Registers have a type field set on loads and on operation results
    a. Basic operations of Add, Subtract, Multiply, Divide, Compare, And, Or, Exclusive-or
    b. Operation type determined by operand types or load instruction
E) Four data sizes supported: “byte”, half, three-quarters and full
F) Four data types supported: unsigned, signed integer, modified float and modified Posit
G) 32-bt instructions for three operand operations
H) Zero extension used for lengthening sub-word size data
    a. Due to unique formats for signed integers and floating-point numbers
I) Floating-point supports unbiased round to odd
J) Floating-point underflow values maintain normalization both in memory and in registers
    a. Besides additional bits for the type code, each register has bits for floating-point needs
K) Floating point NaN code space used instead for exact floating-point values
L) Residue register values vary with operation data type
    a. Generic mechanism for extended precision or extended length computations
M) Byte size is a function of word size sub-multiples
N) External memory (DRAM) is 8-bit byte oriented.
    a. Interaction with external memory takes place via a 256-bit shift register
    b. External memory address space is distinct from internal word size memory space

ISA overview

The 24-bit instruction contains an 8-bit operation code which also indicates the instruction size, three five bit register fields and an immediate bit. If the immediate bit is set the corresponding register field is used either as a small numeric value or as the byte size of a following immediate value.

The 32-bit instruction adds another five bit register field and three option bits. Many instructions offer both 24-bit and 32-bit versions.

The irregular instructions such as relative branches, load immediate, IO port read/write, etc. vary as to the placement of fields. As a general rule the register fields do not move but in some cases are omitted.

Provision is for a full set of operation codes: arithmetic, logic, shifts, load and store, single operand transcendental, conditional branches, extract, insert, bit tests. Looping and predication mechanisms are based on Mitch Alsup’s My 66000 ISA.

Micro-architecture overview

At this time only a basic implementation is planned. Single cycle operation is used where possible. A 32 by 32-bit register file with two or three read ports and one write port form the core[5]. For an FPGA implementation LUTRAM is used for the register file and main memory consists of separate data and instruction block RAMs[6].

Registers and memory are "little endian".

The industry standard performance enhancing mechanisms of word alignment, pipelining, multiple issue, out-of-order, virtual memory and caches are all possible. Interrupt and exception mechanisms TBD.

Access to external memory requires the internal data memory address be divided by 7, 6 or 5 to get the external address. The remainder is used to control the operation of a 256-bit shift register which assembles or disassembles the 256-bit external memory word. This is a multi-cycle operation. For an FPGA implementation a ripple LUT based divide circuit is used operating with a ~10 LUT delay time. For a FPGA having an on-chip ARM processor, the ARM infrastructure is used to access the DRAM within the ARM virtual memory space.

Plans are to allow conditional implementation of instructions so LUT utilization and timing effects can be measured in detail.

Background

This project has a very long time in gestation. Sometime in high school I mastered binary arithmetic, Boolean algebra and the basic components of arithmetic: full adder, carry look ahead, multiplexors, multiplication using a full adder tree and the barrel shifter.

First semester of college (1962) I started using Fortran on an IBM 1620. Soon I was also into assembly language. The university had both the 1620 and a CDC 1604. After obtaining Programmer Reference Manuals (PRM) I was soon considering improvements. The 1620 used a flag bit on each digit. One of the non-BCD digits could serve this purpose and eliminate the flag bit. So one could use the non-BCD digits for sign, floating-point indicator, record terminator and instruction identifiers. Today I have instead settled on radix-100 and radix-128 with a flag bit per digit pair. Still to be settled is as to whether general purpose or special purpose register file is better. Never went very far with 1604 "improvements" other than additional arithmetic operations.

Over the following years I was exposed to a number of computer architectures. Currently have ideas on "upgrades" to TI MSP430, PDP11/34, x86-64, DEC VAX and stack machines.

---

[5] The Residue register requires a second write port. Implementation is as a distinct register updated as required. On register reads the Residue register is multiplexed as needed into the read port results.

[6] At this time it is planned to use the dual-port feature of block RAMs to support unaligned loads, stores and instruction fetch with single cycle performance.

MSP430: One can use the BCD add instruction code space as an escape to a 24-bit instruction that supports five addressing modes, four data sizes and an address space set by the register length. The other original instructions remain in the 16-bit instructions. However the five address modes limit the register file to fourteen registers.

PDP11: One can use the floating-point instruction code space as an escape to a 24-bit instruction that supports twelve registers and six addressing modes, four data sizes and an address space set by the register length. The other original instructions can remain in the 16-bit instructions. However the six address modes limit the register file addressing to thirteen registers, a significant improvement over eight registers.

VAX: Limit memory access to a single operand/result per instruction. Reduce the number of addressing modes by eliminating double indirect. It would have 16 registers and a 24-bit instruction. Specification of an index register is by sixteen one byte prefix instructions.

X86: Get rid of all the prefix instructions and use a 24 or 32-bit instruction format offering 16 general purpose registers and the familiar x86 addressing modes.

In the last few years I have monitored the comp.arch usenet group and learned much about high performance computer design. However my roots are in the embedded world where deterministic execution speed, instruction density and limited memory size are important factors.

As a result I have pursued instruction density and settled on two main ISAs:

1. RISC with 32-registers, 24-bit instructions and variable length immediate values.
2. Stack/accumulator machine with one stack operand specified, 16-bit instruction and five addressing modes.

The RISC ISA is relatively stable. There are four data sizes and four data types. One of the general purpose registers (residue register) captures carries, overflows, upper half of a multiply and the remainder of a divide. 32-bit instructions support an additional register field and it is used for indexed load/stores and three-operand add/subtract, the three operand multiply-add and a few well used instructions: mux, select, bit field insert, and when the residue value needs to be included.

Driven by the lack of encoding space in the 16-bit stack ISA[7], developed the idea of using a complete set of load instructions for each data size and each data type. This simplifies the operation codes as the type of operation can be determined from the data types of the operands: Add, Subtract, Multiply, Divide, Compare, And, Or & Exclusive-Or. Thus there is a net gain in instruction encoding space.

It was pointed out that the auto-increment and auto-decrement add pressure on register file write port(s) and is a limiting factor for high performance. So I have developed alternate addressing modes

[7] Now named OSMnn_mm (One operand Stack Machine). the first implementation is likely to be OSM18_18

wherein an implied index register addressing mode replaces auto-increment/auto-decrement with addition/subtraction of an implied scaled index register[8].

Another recent development sprung from a challenge: Wide register file: Initially 16 1024-bit registers that can be accessed as 256 64-bit values, 128 128-bit values, 64 256-bit values, 32 512-bit values and 16 1024-bit values. SIMD operations are assumed. For an FPGA implementation a 64 element 256-bit register file makes the most sense.

On comp.arch there are those who would prefer 36 or 48-bit words. 36 divides into 256 with a remainder of 4-bits. And 48 divides into 256 with a remainder of 16-bits. The idea is that a 36-bit word size ISA would support 9-bit "bytes" and 18-bit half-words. Likewise a 48-bit word size ISA would support 12-bit "bytes". Was there another word size possible? Yes, a 42-bit word size.

Besides the legacy support for 36 and 48-bit word size architectures there are some performance advantages to a word size smaller than 64-bits: The arithmetic function units are smaller and memory capacity increases for a given memory. Since ASICs and FPGAs readily support 36, 42 and 48 bit word sizes and the corresponding data cache memory, the remaining problem is interfacing to the 8-bit byte oriented external DRAM memory. So, picking one of these word sizes, the interface to DRAM can be through a 256-bit shift register with a 256-bit interface to DRAM and a 36, 42 or 48 bit interface to internal, typically cache, memory. A divide by 7, 6 or 5 circuit is needed where the quotient gives the DRAM memory address and the remainder gives the word location within the shift register. DRAM is much slower than cache or register memory and the divide operation effects a small additional delay in memory access time.

And finally, why not use the same ISA for each of these word sizes? My current RISC ISA is 8-bit byte oriented and implementation would use a standard instruction cache and a standard interface to DRAM. The data side of each implementation would have 9, 21 and 12-bit bytes in a straight binary address space. Harvard architectures for 36, 42 and 48-bit word size computers that can share the exact same instruction binaries[9]!

An FPGA implementation can now begin.

[8] For many of the inner loops using vector or matrix operands a single index register suffices. For ROC_nn_mm the longer length load/store instructions support a general purpose index register.

[9] Elementary function evaluation will need optimal polynomials and polynomial coefficients on different word sizes for maximum accuracy etc.