Intel Itanium™ Floating-Point Architecture

Marius Cornea, John Harrison, and Ping Tak Peter Tang
Intel Corporation

Agenda

- Intel® Itanium® Architecture
- Intel® Itanium® Processor Floating-Point Architecture
- Status Fields and Exceptions
- The Floating-Point Multiply-Add
- Exact Arithmetic
- Accurate Remainders
- Accurate Range Reduction
- Comparison and Classification
- Division and Square Root
- Additional Features
- Conclusion

Intel® Itanium® architecture

- One of the major processor architectures present in the market today
  - 2001 – Intel Itanium processor
  - 2003 – Intel Itanium 2 processor – highest SPEC CFP2000 score currently
  - Price/performance ratio and power consumption better with every new implementation
  - Large register sets : 128 floating-point registers
  - Predication
  - Speculation

Intel® Itanium® architecture

- Support for explicit parallelism: static and rotating registers
- Floating-point features aimed at speed and accuracy
- EPIC (Explicitly Parallel Instruction Computing) design philosophy
- Target the most demanding enterprise and high-performance computing applications

Itanium Processor Floating-Point Architecture

- *Floating-point multiply-add* (*fused multiply-add*) allows higher accuracy and performance
- Software and hardware interaction
- Division: the throughput can be as high as one result for every 3.5 clock cycles
- Floating-point formats: 24, 53, 64 bit significands; 8, 11, 15, 17-bit exponents; 1-bit sign
- Register and memory encodings: 0, normalized values, denormalized/unnormalized values, infinity, NaN, NaTVal ('not a value', for speculative operations); redundant representations

Itanium Processor Floating-Point Architecture

- Examples:
  - Using double-extended intermediate precision calculations to compute a double precision function: the double precision input arguments can be freely combined with double-extended intermediate results.
  - Computing functions involving constants with few significant digits: whatever the precision of the computation, the short constants can be stored in single precision

Status Fields and Exceptions

- 64-bit *Floating-Point Status Register* (FPSR)
  - six *trap disable* bits control the five IEEE Standard exceptions and the denormal exception
  - four 13-bit status fields: s0, s1, s2 and s3
  - six flags per status field, that record the occurrence of each of the 6 exceptions
  - Seven control bits per status field: rounding (2 bits), precision (2 bits), traps disable, flush-to-zero (ftz), and widest-range exponent (wre bit, for 17-bit exponents)

Status Fields and Exceptions

- Status field usage determined by software conventions:
  - s0 is the main user status field
  - s1, with wre enabled and all exceptions disabled is used in many standard numerical software kernels such as those for division, square root, and transcendental functions
  - status fields s2 and s3 are commonly used for speculation

The Floating-Point Multiply-Add

- Basic assembly syntax:

  (*qp*) fma.*pc.sf* *f*1 = *f*3, *f*4, *f*2

  which calculates $f1 = f3 \cdot f4 + f2$ with one rounding error
- Addition and multiplication are implemented as special cases of the fma: $x + y = x \cdot 1 + y$ and $x \cdot y = x \cdot y + 0$
- Two variants of the fma exist: the fms (floating-point multiply-subtract) and fnma (floating-point negative multiply-add):

  (*qp*) fms.*pc.sf* *f*1 = *f*3, *f*4, *f*2

  (*qp*) fnma.*pc.sf* *f*1 = *f*3, *f*4, *f*2

  compute $f1 = f3 \cdot f4 - f2$ and $f1 = -f3 \cdot f4 + f2$ respectively

The Floating-Point Multiply-Add

- Example: the vector dot product $x \cdot y$ of two n-dimensional vectors:

  $$p = \sum xi \cdot yi$$

  can be evaluated by a succession of fma operations of the form

  $$p = p + xi \cdot yi$$

  requiring only *n* floating-point operations, whereas with a separate multiplication and addition it would require $2n$ operations, with a longer overall latency

Exact Arithmetic

- Addition - if $|x| \geq |y|$ the exact sum $x + y$ can be obtained as a two-piece expansion $Hi + Lo$:

  $Hi = x + y$

  $tmp = x - Hi$

  $Lo = tmp + y$

  ($Hi + Lo = x + y$ exactly, with Lo a rounding error in $Hi \approx x + y$)

- Multiplication - the exact product $x \cdot y$ can be obtained as a two-piece expansion $Hi + Lo$:

  $Hi = x \cdot y$

  $Lo = x \cdot y - Hi$

  ($Hi + Lo = x \cdot y$ exactly, with Lo a rounding error in $Hi \approx x \cdot y$)

Accurate Remainders

- If a floating-point number q is approximately equal to the quotient a / b of two floating-point numbers, the remainder $r = a - b \cdot q$ can be calculated exactly with one fnma operation, if q is within 1 ulp (unit-in-the-last-place) of a / b
- Useful in software implementations of the floating-point division, square root, and remainder; also for integer division and remainder computations, implemented based on floating-point operations

Accurate Range Reduction

- Many algorithms for mathematical functions (e.g. *sin*) begin with an initial range reduction phase, subtracting an integer multiple of a constant such as $\pi / 2$
- With the fma this can be done in a single instruction $x - N \cdot P$
- Typically:

$$y = Q \cdot x$$

$$N = rint\,(y)$$

$$r = x - N \cdot P$$

where *rint* (*y*) denotes the rounding of *y* to an integer, and $Q \approx 1 / P$

Comparison and Classification

- Syntax:

(*qp*) fcmp.*frel.fctype p*1, *p*2 = *f*2, *f*3

where the *frel* completer determines the relation that is tested for.

- Mnemonics for *frel*: eq for $f2 = f3$, lt for $f2 < f3$, le for $f2 \leq f3$, gt for $f2 > f3$, ge for $f2 \geq f3$,and unord for $f2 \; ? \; f3$.
- There is no signed/unsigned distinction but there is a new possibility, ($f2 \; ? \; f3$): two values may be *unordered*, since a NaN (Not a Number) compares false with any floating-point value, even with itself
- *fctype* is the comparison type – normal, or unconditional

Division and Square Root

- Implemented in software, based on the reciprocal approximation and reciprocal square root approximation instructions
- Given two floating-point numbers *a* and *b*, the *floating-point reciprocal approximation* instruction, frcpa, normally returns an approximation of 1/*b* good to about 8 bits

  (*qp*) frcpa.*sf f*1, *p*2 = *f*2, *f*3
- Given a floating-point number *a*, the *floating-point reciprocal square root approximation* instruction normally returns an approximation of 1/√a good to about 8 bits:

  (*qp*) frsqrta.*sf f*1, *p*2 = *f*3

Additional Features

- Transferring values between floating-point and integer registers by means of the getf and setf instructions
- Foating-point merging with fmerge, useful in combining fields of multiple floating-point numbers
- Floating-point to integer and integer to floating-point conversion using the fcvt instructions
- Integer multiplication and division - implemented using the floating-point unit
- Floating-point maximum and minimum, using the fmax, famax, fmin and famin instructions

Conclusion

- The Itanium floating-point architecture was designed with high performance, accuracy, and flexibility characteristics which make it ideal for technical computing
- All floating-point data types are mapped internally to an 82-bit format, with 64 bits of accuracy and a 17-bit exponent - calculations are more accurate and do not underflow or overflow as often as on other processors
- Highest current SPEC CFP2000 score for a single processor system: 1431, for an Itanium 2 system at 1GHz - the Hewlett-Packard HP Server RX2600

References

- [1] Intel(R) Itanium(TM) Architecture Software Developer's Manual, Revision 2.0, Vol 1-4, Intel Corporation, December 2001
- [2] John Hennessy, David Patterson, "Computer Architecture - A Quantitative Approach", Morgan Kauffman Publishers, Inc., third edition, 2002
- [3] Peter Markstein, "IA-64 and Elementary Functions: Speed and Precision", Hewlett-Packard/Prentice-Hall 2000
- [4] Marius Cornea, John Harrison, Ping Tak Peter Tang, "Scientific Computing on Itanium-based Systems", Intel Press 2002
- [5] John Crawford, Jerry Huck, "Motivations and Design Approach for the IA-64 64-Bit Instruction Set Architecture", Oct. 1997, San Jose, http://www.intel.com/pressroom/archive/speeches/mpf1097c.htm
- [6] ANSI/IEEE Standard 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, IEEE, New York, 1985
- [7] O. Moller, "Quasi double-precision in floating-point addition", BIT journal, Vol. 5, 1965, pages 37-50
- [8] T. J. Dekker, "A Floating-Point Technique for Extending the Available Precision", Numerical Mathematics journal, Vol. 18, 1971, pages 224-242
- [9] "Divide, Square Root, and Remainder Algorithms for the Itanium Architecture", Intel Corporation, Nov. 2000, http://www.intel.com/software/products/opensource/libraries/numnote2.htm , http://developer.intel.com/software/products/opensource/libraries/numdown2.htm