

Changing the way Computers Compute

Hillgang
24 April 2008

Mike Cowlshaw
IBM Fellow

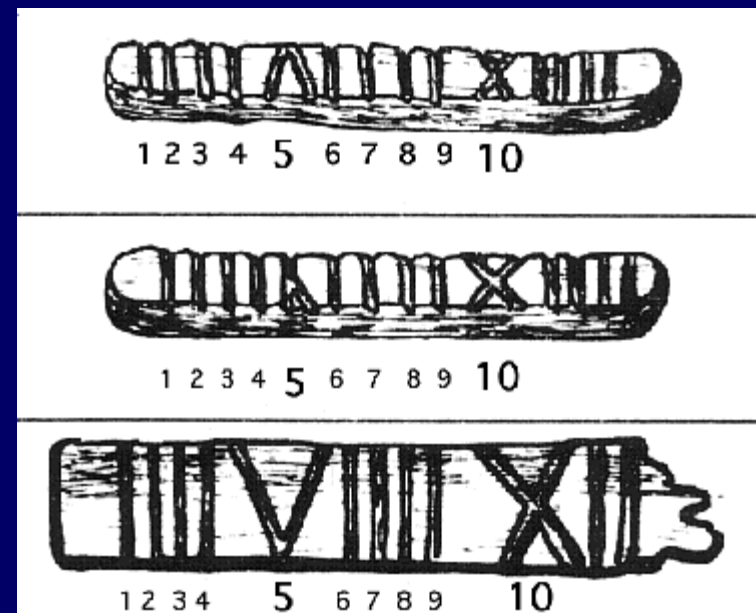


Overview

- Why decimal floating-point (DFP)?
- DFP in System z and software
- Support in standards, *etc.*
- Using DFP in C and Java
- Questions?

Origins of decimal arithmetic

- Decimal (base 10) arithmetic has been used for thousands of years
- Algorism (Indo-Arabic place value system) in use since 800 AD
- Calculators and many computers were decimal ...



IBM 650 (in Böblingen)

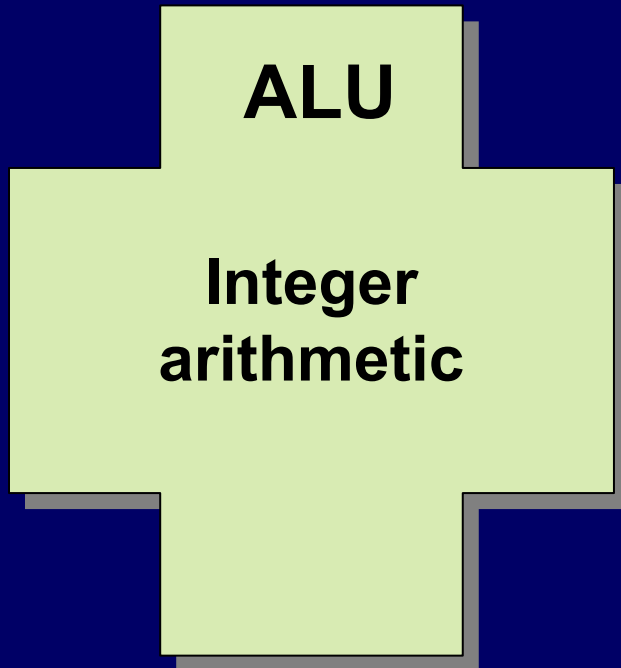


Bi-quinary digit

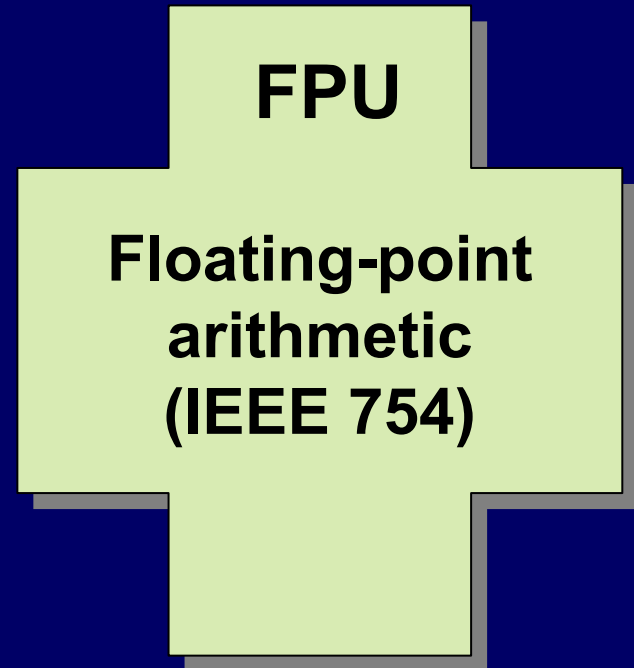
Binary computers

- In the 1950s binary floating-point was shown to be more efficient
 - minimal storage space
 - more reliable (20% fewer components)

Today's computer arithmetic

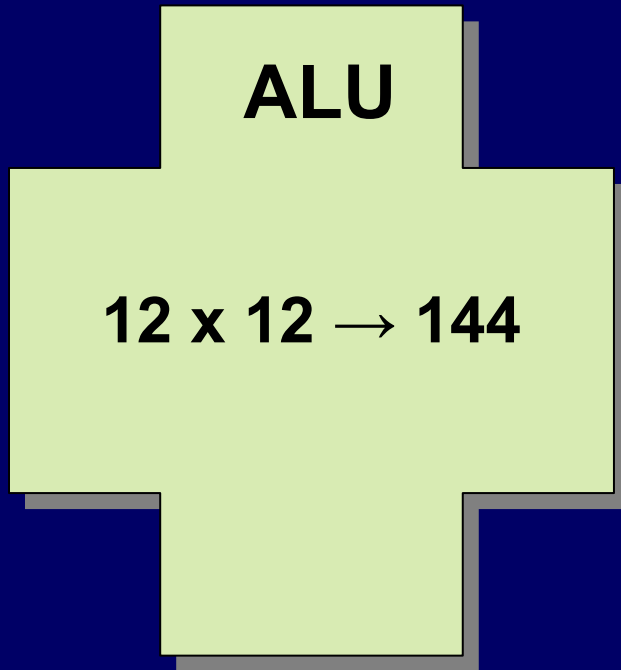


byte, short, int,
long, *etc.*

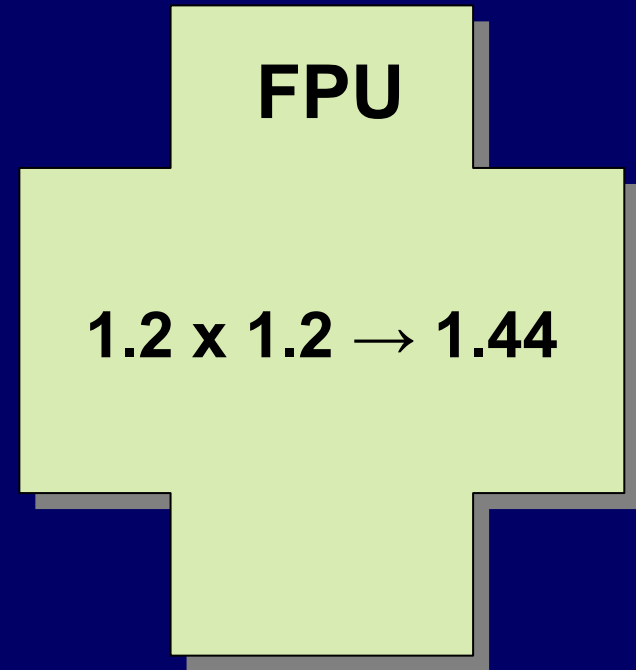


single, float,
double, quad, *etc.*

Today's computer arithmetic



byte, short, int,
long, *etc.*



single, float,
double, quad, *etc.*

$$1.2 \times 1.2 = 1.44 ?$$

- binary fractions *cannot* exactly represent most decimal fractions
 - 0.1 in binary is 0.00011001100110011...
- 1.2 in a 32-bit binary float is actually:
1.2000000476837158203125
- and this squared is:
1.440000057220458984375

Why not just round ?

- Rounding hides but does not help
 - obscures the slight inaccuracies
 - errors accumulate [double rounding, too]

Decimal	Java float (binary)
9	9
0.9	0.9
0.09	0.0899999996
0.009	0.009

Where it costs real money...

- Add 5% sales tax to a € 0.70 telephone call, rounded to the nearest cent
- 1.05×0.70 using binary double is exactly
0.73499999999999998667732370449812151491641998291015625
(should have been 0.735)
- rounds to € 0.73, instead of € 0.74

Hence...

- Binary floating-point cannot be used for commercial or human-centric applications
 - cannot meet legal and financial requirements
- Decimal data and arithmetic are pervasive
- 55% of numeric data in databases are decimal (and a further 43% are integers, often held as decimal integers)

Why floating-point decimal?

- Traditional integer arithmetic with ‘manual’ scaling is awkward, error-prone, and very hard to maintain
- Floating-point is increasingly necessary
 - interest calculated daily
 - telephone calls priced by the second
 - taxes more finely specified
 - financial analysis, *etc.*

Why decimal hardware?

Software is slow: typical Java™ 1.4 BigDecimal add is 1,708 cycles, hardware is ~ 12 cycles

	software penalty
add	150x – 380x
quantize	60x – 140x
multiply	30x – 120x
divide	180x – 200x

penalty = Java 1.4 BigDecimal cycles ÷ DFPU clock cycles

Effect on real applications

- The 'telco' billing application
1,000,000 calls (two minutes)
read from file, priced, taxed,
and printed



	Java [†] BigDecimal	C, C# packages	Itanium hand-tuned
% execution time in decimal operations	93.2%	72 – 78%	45% *

† Java 1.4
(Sun JVM)

* Intel™ figure

The path to hardware...

- A 2x (maybe more) performance improvement in applications makes hardware support *very* attractive
- Standard formats are essential for language and hardware interfaces
 - IEEE 754 is being revised (since 2001)
 - incorporates IEEE 854 (radix-independent)

IEEE 754 agreed draft ('754r')

- Now has decimal floating-point formats (7, 16, and 34-digit) and arithmetic
 - suitable for mathematical applications, too
- Fixed-point and integer decimal arithmetic are subsets (no normalization)
- Compression maximizes precision and exponent range of formats

IEEE 754r Formats

size (bits)	digits	exponent range
32	7	-95 to +96
64	16	-383 to +384
128	34	-6143 to +6144

(range always larger than same-size binary format)

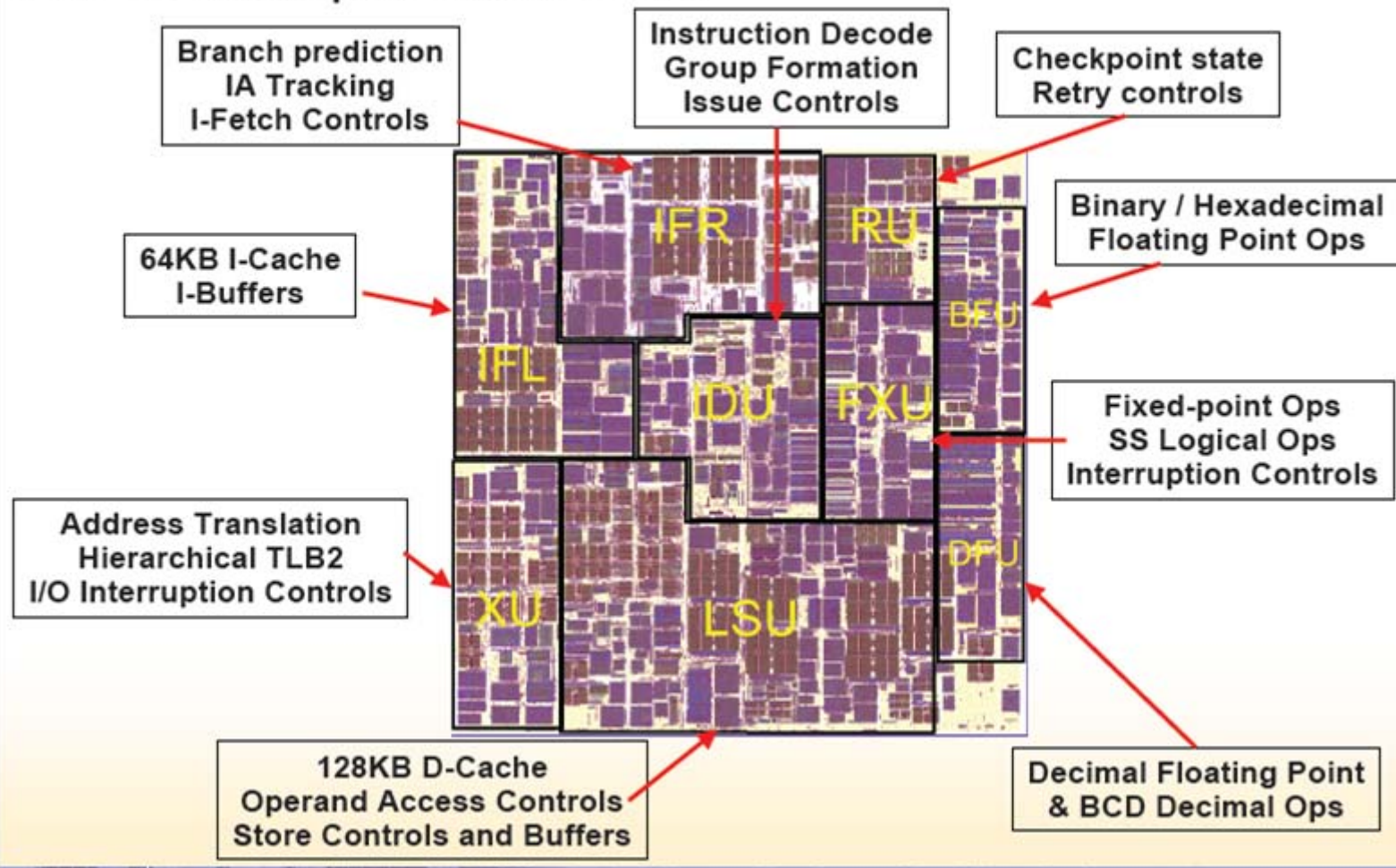
Integer-based floating-point

- Derived from Rexx decimal arithmetic
 - adds Infinities and NaNs
 - rounding modes (HALF_UP, HALF_EVEN, TRUNCATE, *etc.*)
 - improved division: $2.40/2$ gives 1.20
 - fixed precision
 - fixed (smaller) exponent range
 - some subtle differences (edge effects, *etc.*)

IBM Decimal Hardware

- Power6 processor (2007) has a decimal floating-point unit (DFPU) in hardware
 - 54 instructions; registers shared with BFPU
 - first commercial DFPU for 40 years
- System z9 (2007) – emulation with hardware assists
- System z10 (Feb. 2008) hardware DFPU

IBM z6 Microprocessor Core



All statements regarding IBM future directions and intent are subject to change or withdrawal without notice and represent only

IBM Software

- Operating system enhancements
 - AIX, Linux tools, z/OS, z/VM, i5/OS
- Languages
 - Java, Language Environment (LE)
 - XLC for z/OS, AIX, and Linux
 - High Level Assembler, Enterprise PL/I
- DB2 Database (new DECFLOAT datatype)
 - Available for z/OS, AIX, Linux, and Windows

Other standards, *etc.*

- Java 5 BigDecimal
 - IBM JVM will use hardware if available
 - no re-compile necessary
- C# and .Net ECMA and ISO standards
 - arithmetic changed to match IEEE 754r, and now allow use of 745r decimal128
- XML Schema 1.1 draft now has *pDecimal*

Other standards, *etc.* [2]

- ISO C and C++ are jointly adding decimal floating-point as first-class primitive types
- ECMAScript (JavaScript / JScript / ActionScript) is adding the decimal128 type
- Strong support expressed by SAP, Microsoft, SHARE, academia, and many others

Exploiting DFP in Java

- Java decimal processing is done using the `BigDecimal` class
 - used by JDBC (database interface), *etc.*
 - arbitrary-length decimal arithmetic
- Greatly enhanced in Java 5 to match IEEE 754r arithmetic (except Infinities and NaNs)
- Added new `MathContext` class to set sizes

Java BigDecimal Add

- Cannot simply use '+' (yet):

```
BigDecimal a = new BigDecimal("1.27");  
BigDecimal b = new BigDecimal("1.23");  
BigDecimal c = a.add(b);  
System.out.println(c);
```

→ 2.50

Java MathContext

- Sets a size (and rounding mode) to match hardware size (16 digits):

```
MathContext mc = MathContext.DECIMAL64;
```

```
BigDecimal c = a.divide(b, mc);
```

```
System.out.println(c);
```

→ 1.032520325203252

Java DFP hardware support

- The hardware support in the JIT saves values in hardware format if they fit
- Operations use hardware if possible, and keep results in hardware format if they fit
 - almost always fit if `MathContext.DECIMAL64` is used
- In Java 5 SR3; enabled by command-line option: `-Xbddfp`

Using DFP in C

- Option A: packages to use with existing compilers
 - decNumber (includes decFloats)
- Option B: new compilers which support DFP as native datatypes
 - Gnu (GCC)
 - IBM XLC for z/OS

decNumber C package

- Arbitrary-precision decimal package
 - includes the hardware formats but does not use hardware
 - also converts to / from strings and BCD
- Open source cross-platform ANSI C
 - very free licence
 - matches hardware results exactly
 - available since 2003, look for it at:
<http://www2.hursley.ibm.com/decimal>

decFloats C package

- Fixed-precision decimal package
 - computes directly on the new IEEE 754r decimal formats
 - often 2–3 times faster than decNumber
- Open source cross-platform ANSI C
 - very free licence
 - included in the decNumber package, at:
<http://www2.hursley.ibm.com/decimal>

Gnu C Compiler (GCC)

- Open source C compiler for Linux, *etc.*
- Includes the proposed ISO C decimal types (since GCC 4.2)
 - uses decFloats for decimal arithmetic when no hardware available
 - hardware code generation is in GCC 4.3, shipped in Suse SLES 11 and Red Hat RHEL 6; the option to use the instructions is `-march=z9-ec`

Performance expectations

- Java:
 - JIT approach has overheads, so will not help trivial applications very much (e.g., adding a column of aligned numbers)
 - biggest improvement is using MathContext when rounding (and for divisions, *etc.*)
- C packages are quite fast
- C compilers exploit full hardware speed

Benefits of the new types

- Hardware performance:
 - individual operations can be 150x faster than Java 1.4 and 10x–50x faster than C packages
 - applications may be 2x faster
 - almost all commercial and financial applications will get a performance boost

Benefits of the new types [2]

- Standard data types for decimal bring all the benefits that binary applications enjoy:
 - known, standard, formats, so no conversions (less checking needed at interfaces)
 - faster processing (especially with hardware)
 - well-defined arithmetic, rounding rules, *etc.*
 - safer and easier-to-write applications
 - long-term: single numeric type (no binaries)

Questions?

Google: general decimal arithmetic



Format details

IEEE 754r: common 'shape'



- Sign and combination field fit in first byte
 - combination field (5 bits) combines 2 bits of the exponent (0–2), first digit of the coefficient (0–9), and the two special values
 - allows 'bulk initialization' to zero, NaNs, and \pm Infinity by byte replication

Exponent continuation



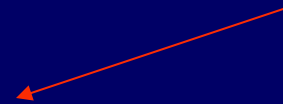
Simple concatenation

Format	exponent bits	bias	normal range
32-bit	2+6	101	-95 to +96
64-bit	2+8	398	-383 to +384
128-bit	2+12	6176	-6143 to +6144

(All ranges larger than binary in same format.)

Coefficient continuation

Sign	Comb. field	Exponent	Coefficient
------	-------------	----------	-------------



- Densely Packed Decimal – 3 digits in each group of 10 bits (6, 15, or 33 in all)
- Derived from Chen-Ho encoding, which uses a Huffman code to allow expansion or compression in 2–3 gate delays

