The IEEE 754-2008 Floating Point Standard and its Pending Revision

Ralph Baker Kearfott

Department of Mathematics
University of Louisiana at Lafayette

Abstract

The IEEE 754 floating point standard, important in science and engineering, is due to expire in 2018 unless it is reviewed, and the P-754 working group has again become active. We review the IEEE 754-2008 floating point standard, explain some issues, and invite input and participation.

UL Applied Math. Seminar, Fall, 2015

Outline

Origins and Early History
(prehistory)

- Early computers used “fixed point” arithmetic, but those computations suffered extreme limitations on size.

Origins and Early History
(prehistory)

- Early computers used “fixed point” arithmetic, but those computations suffered extreme limitations on size.
- Prior to 1977, there were many arithmetic systems based roughly on scientific notation (“floating point:” structured with a sign, mantissa, exponent sign, and exponent). Almost all* had base either 10 or a power of 2, but with varying word lengths (total number of digits used to store a number), varying exponent range, and varying ways of rounding after operations.

Origins and Early History
(prehistory)

- Early computers used “fixed point” arithmetic, but those computations suffered extreme limitations on size.
- Prior to 1977, there were many arithmetic systems based roughly on scientific notation (“floating point:” structured with a sign, mantissa, exponent sign, and exponent). Almost all* had base either 10 or a power of 2, but with varying word lengths (total number of digits used to store a number), varying exponent range, and varying ways of rounding after operations.

  Examples:

Origins and Early History

(prehistory)

- Early computers used “fixed point” arithmetic, but those computations suffered extreme limitations on size.
- Prior to 1977, there were many arithmetic systems based roughly on scientific notation (“floating point:” structured with a sign, mantissa, exponent sign, and exponent). Almost all* had base either 10 or a power of 2, but with varying word lengths (total number of digits used to store a number), varying exponent range, and varying ways of rounding after operations.

  Examples:

  - IBM mainframes had base 16, with a 32-bit word.

Origins and Early History

(prehistory)

- Early computers used “fixed point” arithmetic, but those computations suffered extreme limitations on size.
- Prior to 1977, there were many arithmetic systems based roughly on scientific notation (“floating point:” structured with a sign, mantissa, exponent sign, and exponent). Almost all* had base either 10 or a power of 2, but with varying word lengths (total number of digits used to store a number), varying exponent range, and varying ways of rounding after operations.

Examples:

- IBM mainframes had base 16, with a 32-bit word.
- Univac and Honeywell systems had base 2, with a 36-bit word length.

Origins and Early History
(prehistory)

- Early computers used "fixed point" arithmetic, but those computations suffered extreme limitations on size.
- Prior to 1977, there were many arithmetic systems based roughly on scientific notation ("floating point:" structured with a sign, mantissa, exponent sign, and exponent). Almost all* had base either 10 or a power of 2, but with varying word lengths (total number of digits used to store a number), varying exponent range, and varying ways of rounding after operations.

  Examples:
  - IBM mainframes had base 16, with a 32-bit word.
  - Univac and Honeywell systems had base 2, with a 36-bit word length.

* The proposed Russian "Setun" computer would have used base 3!

Origins and Early History

Reasons for (or against?) a standard

- The same program (written in standard Fortran) would give different results on different machines: *Precision requirements were not portable.*

Origins and Early History

Reasons for (or against?) a standard

- The same program (written in standard Fortran) would give different results on different machines: *Precision requirements were not portable.*
- Results on one machine could not be reproduced on another machine, not even approximately.

Origins and Early History

Reasons for (or against?) a standard

- The same program (written in standard Fortran) would give different results on different machines: *Precision requirements were not portable.*
- Results on one machine could not be reproduced on another machine, not even approximately.
- Even if condition numbers were known, required precision mandated different programs on different machines.

Origins and Early History

Reasons for (or against?) a standard

- The same program (written in standard Fortran) would give different results on different machines: *Precision requirements were not portable.*
- Results on one machine could not be reproduced on another machine, not even approximately.
- Even if condition numbers were known, required precision mandated different programs on different machines.
- Existence of common elementary functions could not be assumed.

Origins and Early History

Reasons for (or against?) a standard

- The same program (written in standard Fortran) would give different results on different machines: *Precision requirements were not portable.*
- Results on one machine could not be reproduced on another machine, not even approximately.
- Even if condition numbers were known, required precision mandated different programs on different machines.
- Existence of common elementary functions could not be assumed.
- However, different accuracies on different machines could sometimes be exploited to identify ill-conditioning.

Origins and Early History

(committee formation)

1977: Intel starts design of microcomputer processor, and is persuaded to standardize the floating point operations; other vendors set up a standardization effort (the IEEE 754 working group) in response, to avoid unfair advantage from Intel.

Origins and Early History

(committee formation)

1977: Intel starts design of microcomputer processor, and is persuaded to standardize the floating point operations; other vendors set up a standardization effort (the IEEE 754 working group) in response, to avoid unfair advantage from Intel.

Nov., 1977: William Kahan*, also an Intel consultant, supplied the 754 WG with a draft proposal.

Origins and Early History

(committee formation)

1977: Intel starts design of microcomputer processor, and is persuaded to standardize the floating point operations; other vendors set up a standardization effort (the IEEE 754 working group) in response, to avoid unfair advantage from Intel.

Nov., 1977: William Kahan*, also an Intel consultant, supplied the 754 WG with a draft proposal.

∗ Bill (Velvel) Kahan, the "Father of Floating Point," has been a highly outspoken advocate of reliable floating point arithmetic, did early work in interval arithmetic, and has supervised prominent graduate students at U.C. Berkeley.

Origins and Early History

(Implementation and revision)

1980: Intel introduces the 8087 coprocessor, an optional add-on to PC's with circuitry based on a draft of the standard.

Origins and Early History

(Implementation and revision)

1980: Intel introduces the 8087 coprocessor, an optional add-on to PC's with circuitry based on a draft of the standard.

- 754 support was provided in software (much slower) if the 8087 was absent.

Origins and Early History

(Implementation and revision)

1980: Intel introduces the 8087 coprocessor, an optional add-on to PC's with circuitry based on a draft of the standard.

- 754 support was provided in software (much slower) if the 8087 was absent.
- Current Intel chips have 754-support built-in.

Origins and Early History

(Implementation and revision)

1980: Intel introduces the 8087 coprocessor, an optional add-on to PC's with circuitry based on a draft of the standard.

- 754 support was provided in software (much slower) if the 8087 was absent.
- Current Intel chips have 754-support built-in.

1985: IEEE 754-1985 becomes an official standard; see `https://en.wikipedia.org/wiki/IEEE_754-1985`

Origins and Early History

(Implementation and revision)

1980: Intel introduces the 8087 coprocessor, an optional add-on to PC's with circuitry based on a draft of the standard.

- 754 support was provided in software (much slower) if the 8087 was absent.
- Current Intel chips have 754-support built-in.

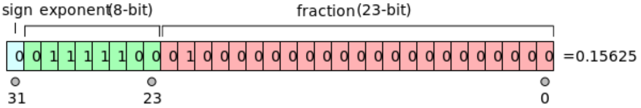
1985: IEEE 754-1985 becomes an official standard; see `https://en.wikipedia.org/wiki/IEEE_754-1985`

2008: A revision is published, and IEEE 754-2008 becomes the official standard.

Main Features in both 1985 and 2008

Classic binary

- Single Precision is based on a 32-bit word (viewed as 4 8-bit bytes), with a 23-bit fraction:



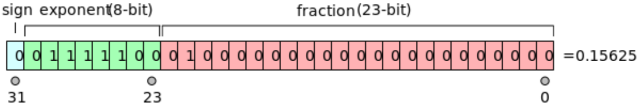
(roughly 7 decimal digits and decimal exponent range ± 38)

(figures from Wikipedia: "IEEE 754 Single Floating Point Format" by Codekaizen)

Main Features in both 1985 and 2008

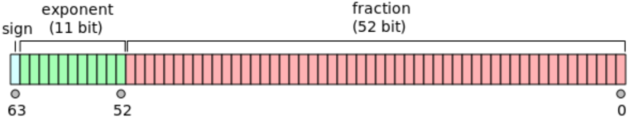
Classic binary

- Single Precision is based on a 32-bit word (viewed as 4 8-bit bytes), with a 23-bit fraction:



(roughly 7 decimal digits and decimal exponent range ± 38)

- Double Precision is based on a 64-bit word (viewed as 8 8-bit bytes), with a 52-bit fraction:



(roughly 16 decimal digits and decimal exponent range ± 308)

(figures from Wikipedia: “IEEE 754 Single Floating Point Format” by Codekaizen)

Main Features in both 1985 and 2008

Rounding Modes

- Has four rounding modes: round to nearest, round up, round down, and round towards 0.

Main Features in both 1985 and 2008

Rounding Modes

- Has four rounding modes: round to nearest, round up, round down, and round towards 0.
- The correctly rounded concept: The stored result is the nearest floating point number to the mathematically exact result, according to the selected rounding scheme.

Main Features in both 1985 and 2008

Rounding Modes

- Has four rounding modes: round to nearest, round up, round down, and round towards 0.
- The correctly rounded concept: The stored result is the nearest floating point number to the mathematically exact result, according to the selected rounding scheme.
- Requires $+$, $-$, $\times$, $\div$, and $\sqrt{\cdot}$ be correctly rounded, as well as binary to decimal, decimal to binary, binary to integer, and integer to binary conversions.

Main Features in both 1985 and 2008

Rounding Modes

- Has four rounding modes: round to nearest, round up, round down, and round towards 0.
- The correctly rounded concept: The stored result is the nearest floating point number to the mathematically exact result, according to the selected rounding scheme.
- Requires $+$, $-$, $\times$, $\div$, and $\sqrt{\cdot}$ be correctly rounded, as well as binary to decimal, decimal to binary, binary to integer, and integer to binary conversions.
- When used astutely, with or without interval arithmetic, the rounding modes* can provide mathematically rigorous lower and upper bounds on exact solutions.

Main Features in both 1985 and 2008

Extended Formats

- The standard allows for (but does not mandate) floating point formats that are wider (with more digits in the mantissa) than the standard ones, as an aid to achieving correct rounding.

Main Features in both 1985 and 2008

Extended Formats

- The standard allows for (but does not mandate) floating point formats that are wider (with more digits in the mantissa) than the standard ones, as an aid to achieving correct rounding.
- For example, the Intel line of chips (80x87, “Pentium”, “Core. . . ””) have 80-bit registers, with 3 extra bits.

Main Features in both 1985 and 2008

Extended Formats

- The standard allows for (but does not mandate) floating point formats that are wider (with more digits in the mantissa) than the standard ones, as an aid to achieving correct rounding.
- For example, the Intel line of chips (80x87, “Pentium”, “Core. . . ””) have 80-bit registers, with 3 extra bits.
- In contrast, the Motorola chips that were used in Sun workstations did not have an extended format, but achieved correct rounding in other ways.

Main Features in both 1985 and 2008

Additional features

- Has $+\infty$ and $-\infty$ (generated e.g. through overflow, etc.), treated as numbers in expressions.

Main Features in both 1985 and 2008

Additional features

- Has $+\infty$ and $-\infty$ (generated e.g. through overflow, etc.), treated as numbers in expressions.
- Has the (sometimes infamous) `NaN` (Not-a-Number), generated through operation exceptions (e.g. `sqrt(-5.0)`) and propagated, allowing for non-stop arithmetic (and cryptic printouts full of `NaN`s).

Main Features in both 1985 and 2008

Additional features

- Has $+\infty$ and $-\infty$ (generated e.g. through overflow, etc.), treated as numbers in expressions.
- Has the (sometimes infamous) `NaN` (Not-a-Number), generated through operation exceptions (e.g. `sqrt(-5.0)`) and propagated, allowing for non-stop arithmetic (and cryptic printouts full of `NaN`s).
- Has "gradual underflow" (use of non-normalized numbers) to fill in the bothersome gap between the smallest normalized floating point number and 0.

Main Features in both 1985 and 2008

Additional features

- Has $+\infty$ and $-\infty$ (generated e.g. through overflow, etc.), treated as numbers in expressions.
- Has the (sometimes infamous) `NaN` (Not-a-Number), generated through operation exceptions (e.g. `sqrt(-5.0)`) and propagated, allowing for non-stop arithmetic (and cryptic printouts full of `NaN`s).
- Has “gradual underflow” (use of non-normalized numbers) to fill in the bothersome gap between the smallest normalized floating point number and 0.
- Requires logical and comparison operators ($<$, $>$, $\leq$, `.NOT.`, etc.)

Main Features in both 1985 and 2008

Additional features

- Has $+\infty$ and $-\infty$ (generated e.g. through overflow, etc.), treated as numbers in expressions.
- Has the (sometimes infamous) `NaN` (Not-a-Number), generated through operation exceptions (e.g. `sqrt(-5.0)`) and propagated, allowing for non-stop arithmetic (and cryptic printouts full of `NaN`s).
- Has "gradual underflow" (use of non-normalized numbers) to fill in the bothersome gap between the smallest normalized floating point number and 0.
- Requires logical and comparison operators ($<$, $>$, $\leq$, `.NOT.`, etc.)
- Specifies operations involving ∞ and `NaN`.

Main Features in both 1985 and 2008

Five types of operation exceptions

Invalid operation: Operations on a `NaN`, $0 \times \infty$, etc.

Main Features in both 1985 and 2008

Five types of operation exceptions

Invalid operation: Operations on a `NaN`, $0 \times \infty$, etc.

Division by zero: (when the dividend is non-zero)

Main Features in both 1985 and 2008

Five types of operation exceptions

Invalid operation: Operations on a NaN, $0 \times \infty$, etc.

Division by zero: (when the dividend is non-zero)

Overflow: Result larger than the largest representable number.

Main Features in both 1985 and 2008

Five types of operation exceptions

Invalid operation: Operations on a NaN, $0 \times \infty$, etc.

Division by zero: (when the dividend is non-zero)

Overflow: Result larger than the largest representable number.

Underflow: Result non-zero but with absolute value smaller than the smallest representable positive number.

Main Features in both 1985 and 2008

Five types of operation exceptions

Invalid operation: Operations on a NaN, $0 \times \infty$, etc.

Division by zero: (when the dividend is non-zero)

Overflow: Result larger than the largest representable number.

Underflow: Result non-zero but with absolute value smaller than the smallest representable positive number.

Inexact: Result is not exactly representable (more often true than not).

Main Features in both 1985 and 2008

Five types of operation exceptions

Invalid operation: Operations on a NaN, $0 \times \infty$, etc.

Division by zero: (when the dividend is non-zero)

Overflow: Result larger than the largest representable number.

Underflow: Result non-zero but with absolute value smaller than the smallest representable positive number.

Inexact: Result is not exactly representable (more often true than not).

- (Controversial) The exceptions are logged with static flags.

Main Features in both 1985 and 2008

Five types of operation exceptions

Invalid operation: Operations on a NaN, $0 \times \infty$, etc.

Division by zero: (when the dividend is non-zero)

Overflow: Result larger than the largest representable number.

Underflow: Result non-zero but with absolute value smaller than the smallest representable positive number.

Inexact: Result is not exactly representable (more often true than not).

- (Controversial) The exceptions are logged with static flags.
- The default is to set the flag and continue execution; once set, a flag remains set.

New Features in IEEE 754-2008

- Quadruple-precision (128 bit) binary arithmetic.

New Features in IEEE 754-2008

- Quadruple-precision (128 bit) binary arithmetic.
- Two* decimal formats, encoded in 64 bits and 128 bits.

New Features in IEEE 754-2008

- Quadruple-precision (128 bit) binary arithmetic.
- Two* decimal formats, encoded in 64 bits and 128 bits.

*The two formats are the result of competing requirements between two different manufacturers.

New Features in IEEE 754-2008

- Quadruple-precision (128 bit) binary arithmetic.
- Two* decimal formats, encoded in 64 bits and 128 bits.
- Which formats are present are language- or implementation-defined.

New Features in IEEE 754-2008

- Quadruple-precision (128 bit) binary arithmetic.
- Two* decimal formats, encoded in 64 bits and 128 bits.
- Which formats are present are language- or implementation-defined.
- *Interchange formats* are defined for transferring binary and decimal data between different implementations.

New Features in IEEE 754-2008

- Quadruple-precision (128 bit) binary arithmetic.
- Two* decimal formats, encoded in 64 bits and 128 bits.
- Which formats are present are language- or implementation-defined.
- *Interchange formats* are defined for transferring binary and decimal data between different implementations.
- There is a large *informative* section describing four levels (mathematical reals, floating point numbers, representations, and bit encodings).

New Features in IEEE 754-2008

- Quadruple-precision (128 bit) binary arithmetic.
- Two* decimal formats, encoded in 64 bits and 128 bits.
- Which formats are present are language- or implementation-defined.
- *Interchange formats* are defined for transferring binary and decimal data between different implementations.
- There is a large *informative* section describing four levels (mathematical reals, floating point numbers, representations, and bit encodings).
- A larger set of *recommended* functions is *specified*.

Issues

Inhibition of parallelization, an example:

- The five exception flags are global and static.

Issues

Inhibition of parallelization, an example:

- The five exception flags are global and static.
- What happens if an exception happens in one thread but not another?

Issues

Inhibition of parallelization, an example:

- The five exception flags are global and static.
- What happens if an exception happens in one thread but not another?
- What if the exceptions were associated with the datum, rather than with the overall computation?

Issues

Inhibition of parallelization, an example:

- The five exception flags are global and static.
- What happens if an exception happens in one thread but not another?
- What if the exceptions were associated with the datum, rather than with the overall computation?
- In such a scheme, the exception would be confined to a particular computational thread.

Issues

Inhibition of parallelization, an example:

- The five exception flags are global and static.
- What happens if an exception happens in one thread but not another?
- What if the exceptions were associated with the datum, rather than with the overall computation?
- In such a scheme, the exception would be confined to a particular computational thread.
- Such a scheme has been worked out for the IEEE 1788-2015 standard for interval arithmetic, and can possibly be adapted to a 754 revision.

Issues

Inhibition of parallelization, an example:

- The five exception flags are global and static.
- What happens if an exception happens in one thread but not another?
- What if the exceptions were associated with the datum, rather than with the overall computation?
- In such a scheme, the exception would be confined to a particular computational thread.
- Such a scheme has been worked out for the IEEE 1788-2015 standard for interval arithmetic, and can possibly be adapted to a 754 revision.
- Such a scheme may aid reproducibility.

Lack of reproducibility

- Numerical results running the same standard-complying program with the same input data may not be the same, even when run more than once on the same machine.

Lack of reproducibility

- Numerical results running the same standard-complying program with the same input data may not be the same, even when run more than once on the same machine.
- This problem is due to partially due to the multi-level nature of memory (main chip memory, processor cache, computation registers).

Lack of reproducibility

- Numerical results running the same standard-complying program with the same input data may not be the same, even when run more than once on the same machine.
- This problem is due to partially due to the multi-level nature of memory (main chip memory, processor cache, computation registers).
- Multiple operating system functions beyond user control are performed concurrently.

Lack of reproducibility

- Numerical results running the same standard-complying program with the same input data may not be the same, even when run more than once on the same machine.
- This problem is due to partially due to the multi-level nature of memory (main chip memory, processor cache, computation registers).
- Multiple operating system functions beyond user control are performed concurrently.
- These system functions may force the user's computation out of registers or cache at some times but not others. (A string of computations done in registers will be more accurate.)

Lack of Reproducibility

Pros and cons

- Reproducibility enables easier debugging of complicated software.

Lack of Reproducibility

Pros and cons

- Reproducibility enables easier debugging of complicated software.
- Reproducibility enables easier porting of software across platforms.

Lack of Reproducibility

Pros and cons

- Reproducibility enables easier debugging of complicated software.
- Reproducibility enables easier porting of software across platforms.

However:

Lack of Reproducibility

Pros and cons

- Reproducibility enables easier debugging of complicated software.
- Reproducibility enables easier porting of software across platforms.

However:

- It may be difficult to achieve reproducibility in concurrent (i.e. parallel) computations without giving up concurrency or without a major performance sacrifice.

Lack of Reproducibility

Pros and cons

- Reproducibility enables easier debugging of complicated software.
- Reproducibility enables easier porting of software across platforms.

However:

- It may be difficult to achieve reproducibility in concurrent (i.e. parallel) computations without giving up concurrency or without a major performance sacrifice.
- Do we want exactly the same results on all systems, even if they are incorrect on all systems?

Lack of Reproducibility

The standard

- By recommending, but not mandating, an extended register format, the standard allows for different register sizes on different machines.

Lack of Reproducibility

The standard

- By recommending, but not mandating, an extended register format, the standard allows for different register sizes on different machines.
- The standard does not recommend order of operations.

Lack of Reproducibility

The standard

- By recommending, but not mandating, an extended register format, the standard allows for different register sizes on different machines.
- The standard does not recommend order of operations.

  Note: The Java programming language, meant for web applications, attempts to achieve complete reproducibility, at the expense of maximum performance.

Issues

Incomplete Implementation

Example: Binary to decimal conversion is typically done when "printing" values in a format specified within a programming language.

Issues

Incomplete Implementation

Example: Binary to decimal conversion is typically done when “printing” values in a format specified within a programming language.

- Although chip hardware implements the basic operations, the programming language standard does not require correct rounding upon conversion, and often does not supply it.

Issues

Incomplete Implementation

Example: Binary to decimal conversion is typically done when "printing" values in a format specified within a programming language.

- Although chip hardware implements the basic operations, the programming language standard does not require correct rounding upon conversion, and often does not supply it.
- The values users see are sometimes significantly less accurate than the actual internal binary representations.

Incomplete Implementation

Recommended functions

- The standard specifies a list of recommended functions, largely coinciding with many common programming language function (`SIN`, `EXP`, `LOG`, etc.).

Incomplete Implementation

Recommended functions

- The standard specifies a list of recommended functions, largely coinciding with many common programming language function (`SIN`, `EXP`, `LOG`, etc.).
- If these are present, a system is standard-conforming if the values of these functions are correctly rounded within specified ranges.

Incomplete Implementation

Recommended functions

- The standard specifies a list of recommended functions, largely coinciding with many common programming language function (`SIN`, `EXP`, `LOG`, etc.).
- If these are present, a system is standard-conforming if the values of these functions are correctly rounded within specified ranges.
- Programming language implementations often do not have all of the recommended functions.

Incomplete Implementation

Recommended functions

- The standard specifies a list of recommended functions, largely coinciding with many common programming language function (`SIN`, `EXP`, `LOG`, etc.).
- If these are present, a system is standard-conforming if the values of these functions are correctly rounded within specified ranges.
- Programming language implementations often do not have all of the recommended functions.
- Programming language implementations of IEEE 754-2008 standard functions may not conform to the standard.

Lack of User Access to Features

- Intel, AMD, etc. chips widely implement basic IEEE 754 arithmetic.

Lack of User Access to Features

- Intel, AMD, etc. chips widely implement basic IEEE 754 arithmetic.
- However, programming languages need not use or give access to this.

Lack of User Access to Features

- Intel, AMD, etc. chips widely implement basic IEEE 754 arithmetic.
- However, programming languages need not use or give access to this.
- For example, Fortran and C (or C++) until recently did not have syntax to specify or change the rounding mode.

Lack of User Access to Features

- Intel, AMD, etc. chips widely implement basic IEEE 754 arithmetic.
- However, programming languages need not use or give access to this.
- For example, Fortran and C (or C++) until recently did not have syntax to specify or change the rounding mode.
- Matlab generally uses IEEE 754 double precision for computations, but has not provided documentation to routines to set the rounding mode.

Abandonment of the Standard

- A combination of speed and low power consumption is often the priority (such as in smart phones or graphics processors).

Abandonment of the Standard

- A combination of speed and low power consumption is often the priority (such as in smart phones or graphics processors).
- Designers sometimes judge compliance with IEEE 754-2008 arithmetic to be too complicated to allow fast computation without using more power.

Abandonment of the Standard

- A combination of speed and low power consumption is often the priority (such as in smart phones or graphics processors).
- Designers sometimes judge compliance with IEEE 754-2008 arithmetic to be too complicated to allow fast computation without using more power.
- The fastest supercomputers, consisting of many tiny units such as graphics processors, are presently constrained by power consumption, and have opted to forego standard compliance.

An Additional Comment

- IEEE 754 *does* specify *interchange format* at the bit level, but the *internal representation* of IEEE numbers differs from machine to machine (example: *big endian* versus *small endian*).

An Additional Comment

- IEEE 754 *does* specify *interchange format* at the bit level, but the *internal representation* of IEEE numbers differs from machine to machine (example: *big endian* versus *small endian*).
- Direct transfer of binary data, without interchange functions, is not possible.

An Additional Comment

- IEEE 754 *does* specify *interchange format* at the bit level, but the *internal representation* of IEEE numbers differs from machine to machine (example: *big endian* versus *small endian*).
- Direct transfer of binary data, without interchange functions, is not possible.
- This can be a good thing. (It allows innovation in design.)

Current Status

- The IEEE Standards Association has just authorized the P-754 working group to review and revise the document.

Current Status

- The IEEE Standards Association has just authorized the P-754 working group to review and revise the document.
- The working group's term ends December, 2018.

Current Status

- The IEEE Standards Association has just authorized the P-754 working group to review and revise the document.
- The working group's term ends December, 2018.
- An organizational meeting was held September 22 on the Berkeley campus, with David Hough presiding.

Current Status

- The IEEE Standards Association has just authorized the P-754 working group to review and revise the document.
- The working group's term ends December, 2018.
- An organizational meeting was held September 22 on the Berkeley campus, with David Hough presiding.
- There will be a combination of in-person, teleconference, and email conduct of business.

Prospects

- $ Billions are invested in systems implementing the current standard, making radical changes to it more difficult.

Prospects

- $ Billions are invested in systems implementing the current standard, making radical changes to it more difficult.
- Sentiment has been expressed to mainly

Prospects

- $ Billions are invested in systems implementing the current standard, making radical changes to it more difficult.
- Sentiment has been expressed to mainly
  - correct errors;

Prospects

- $ Billions are invested in systems implementing the current standard, making radical changes to it more difficult.
- Sentiment has been expressed to mainly
  - correct errors;
  - clarify ambiguities.

Prospects

- $ Billions are invested in systems implementing the current standard, making radical changes to it more difficult.
- Sentiment has been expressed to mainly
  - correct errors;
  - clarify ambiguities.
- Nonetheless, wide participation and discussion is important.

How to Participate

(the working group)

- There is an IEEE-SA sponsored mailing list, open to all, to contribute to discussion.

How to Participate

(the working group)

- There is an IEEE-SA sponsored mailing list, open to all, to contribute to discussion.
- Persons may register with the IEEE-SA through a web-based system (MyProject) to join the working group.

How to Participate

(the working group)

- There is an IEEE-SA sponsored mailing list, open to all, to contribute to discussion.
- Persons may register with the IEEE-SA through a web-based system (MyProject) to join the working group.
- Voting privileges are maintained within the working group by active participation.

How to Participate

(the working group)

- There is an IEEE-SA sponsored mailing list, open to all, to contribute to discussion.
- Persons may register with the IEEE-SA through a web-based system (MyProject) to join the working group.
- Voting privileges are maintained within the working group by active participation.
- The working group is responsible for formulating the revision.

How to Participate

(Sponsor Ballot)

- When the P-754 working group reaches consensus on the document, it is submitted for Sponsor Ballot.

How to Participate

(Sponsor Ballot)

- When the P-754 working group reaches consensus on the document, it is submitted for Sponsor Ballot.
- Working group members and others are invited to become members of the Sponsor Ballot Group.

How to Participate
(Sponsor Ballot)

- When the P-754 working group reaches consensus on the document, it is submitted for Sponsor Ballot.
- Working group members and others are invited to become members of the Sponsor Ballot Group.
- Sponsor Ballot members may vote if they either become members of the IEEE-SA (IEEE Standards Association) or pay a per-ballot fee.

How to Participate

(Sponsor Ballot)

- When the P-754 working group reaches consensus on the document, it is submitted for Sponsor Ballot.
- Working group members and others are invited to become members of the Sponsor Ballot Group.
- Sponsor Ballot members may vote if they either become members of the IEEE-SA (IEEE Standards Association) or pay a per-ballot fee.
- When the Sponsor Ballot Group reaches consensus, the document is submitted for procedural review.

How to Participate
(Sponsor Ballot)

- When the P-754 working group reaches consensus on the document, it is submitted for Sponsor Ballot.
- Working group members and others are invited to become members of the Sponsor Ballot Group.
- Sponsor Ballot members may vote if they either become members of the IEEE-SA (IEEE Standards Association) or pay a per-ballot fee.
- When the Sponsor Ballot Group reaches consensus, the document is submitted for procedural review.
- When the document passes procedural review it becomes a revised standard.

Additional Resources

- The Microprocessor Standardization Committee web site is at
  `http://grouper.ieee.org/groups/msc/`
- As chair of the Microprocessor Standardization Committee (the oversight committee for the P-754 working group), you may ask me (at `rbk@louisiana.edu` or in person) about the organization, parliamentary procedures, whom to contact, etc.