COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

NORME DE LA CEI

INTERNATIONAL ELECTROTECHNICAL COMMISSION

IEC STANDARD

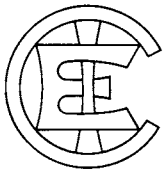
Publication 559

Première édition — First edition

1982

Arithmétique binaire en virgule flottante pour systèmes à microprocesseurs

Binary floating point arithmetic for microprocessor systems



Bureau Central de la Commission Electrotechnique Internationale

3, rue de Varembé

Genève, Suisse

Révision de la présente publication

Le contenu technique des publications de la CEI est constamment revu par la Commission afin d'assurer qu'il reflète bien l'état actuel de la technique.

Les renseignements relatifs à ce travail de révision, à l'établissement des éditions révisées et aux mises à jour peuvent être obtenus auprès des Comités nationaux de la CEI et en consultant les documents ci-dessous:

- **Bulletin de la CEI**
- **Annuaire de la CEI**
- **Catalogue des publications de la CEI**
  Publié annuellement

Terminologie

En ce qui concerne la terminologie générale, le lecteur se reportera à la Publication 50 de la CEI: Vocabulaire Electrotechnique International (V.E.I.), qui est établie sous forme de chapitres séparés traitant chacun d'un sujet défini, l'Index général étant publié séparément. Des détails complets sur le V.E.I. peuvent être obtenus sur demande.

Les termes et définitions figurant dans la présente publication ont été soit repris du V.E.I., soit spécifiquement approuvés aux fins de cette publication.

Symboles graphiques et littéraux

Pour les symboles graphiques, symboles littéraux et signes d'usage général approuvés par la CEI, le lecteur consultera:

— la Publication 27 de la CEI: Symboles littéraux à utiliser en électrotechnique;

— la Publication 117 de la CEI: Symboles graphiques recommandés.

Les symboles et signes contenus dans la présente publication ont été soit repris des Publications 27 ou 117 de la CEI, soit spécifiquement approuvés aux fins de cette publication.

Revision of this publication

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology.

Information on the work of revision, the issue of revised editions and amendment sheets may be obtained from IEC National Committees and from the following IEC sources:

- **IEC Bulletin**
- **IEC Yearbook**
- **Catalogue of IEC Publications**
  Published yearly

Terminology

For general terminology, readers are referred to IEC Publication 50: International Electrotechnical Vocabulary (I.E.V.), which is issued in the form of separate chapters each dealing with a specific field, the General Index being published as a separate booklet. Full details of the I.E.V. will be supplied on request.

The terms and definitions contained in the present publication have either been taken from the I.E.V. or have been specifically approved for the purpose of this publication.

Graphical and letter symbols

For graphical symbols, and letter symbols and signs approved by the IEC for general use, readers are referred to:

— IEC Publication 27: Letter symbols to be used in electrical technology;

— IEC Publication 117: Recommended graphical symbols.

The symbols and signs contained in the present publication have either been taken from IEC Publications 27 or 117, or have been specifically approved for the purpose of this publication.

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

NORME DE LA CEI

INTERNATIONAL ELECTROTECHNICAL COMMISSION

IEC STANDARD

Publication 559

Première édition — First edition

1982

Arithmétique binaire en virgule flottante pour systèmes à microprocesseurs

Binary floating point arithmetic for microprocessor systems

Mots clés: microprocesseurs; opérations arithmétiques binaires; exigences; propriétés; définitions.

Key words: microprocessors; binary arithmetical operations; requirements; properties; definitions.

Bureau Central de la Commission Electrotechnique Internationale

3, rue de Varembé

Genève, Suisse

SOMMAIRE

CONTENTS

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

ARITHMÉTIQUE BINAIRE EN VIRGULE FLOTTANTE POUR SYSTÈMES À MICROPROCESSEURS

PRÉAMBULE

1) Les décisions ou accords officiels de la CEI en ce qui concerne les questions techniques, préparés par des Comités d'Etudes où sont représentés tous les Comités nationaux s'intéressant à ces questions, expriment dans la plus grande mesure possible un accord international sur les sujets examinés.

2) Ces décisions constituent des recommandations internationales et sont agréées comme telles par les Comités nationaux.

3) Dans le but d'encourager l'unification internationale, la CEI exprime le vœu que tous les Comités nationaux adoptent dans leurs règles nationales le texte de la recommandation de la CEI, dans la mesure où les conditions nationales le permettent. Toute divergence entre la recommandation de la CEI et la règle nationale correspondante doit, dans la mesure du possible, être indiquée en termes clairs dans cette dernière.

PRÉFACE

La présente norme a été préparée par le Sous-Comité 47B: Systèmes à microprocesseurs, du Comité d'Etudes N° 47 de la CEI: Dispositifs à semiconducteurs et circuits intégrés.

Un premier projet* fut discuté lors de la réunion tenue à Montreux en 1981. A la suite de cette réunion, un projet, document 47B(Bureau Central)2, fut soumis à l'approbation des Comités nationaux suivant la Règle des Six Mois en novembre 1981.

Les Comités nationaux des pays suivants se sont prononcés explicitement en faveur de la publication:

Afrique du Sud (République d')
Allemagne
Autriche
Belgique
Canada
Egypte
Espagne
Etats-Unis d'Amérique
France
Irlande
Italie
Japon
Pays-Bas
Pologne
Roumanie
Suède
Suisse
Union des Républiques Socialistes Soviétiques
Yougoslavie

* Ce premier projet fut préparé d'après le projet P754, version 8.0, du I.E.E.E.

INTERNATIONAL ELECTROTECHNICAL COMMISSION

BINARY FLOATING POINT ARITHMETIC FOR MICROPROCESSOR SYSTEMS

FOREWORD

1) The formal decisions or agreements of the IEC on technical matters, prepared by Technical Committees on which all the National Committees having a special interest therein are represented, express, as nearly as possible, an international consensus of opinion on the subjects dealt with.

2) They have the form of recommendations for international use and they are accepted by the National Committees in that sense.

3) In order to promote international unification, the IEC expresses the wish that all National Committees should adopt the text of the IEC recommendation for their national rules in so far as national conditions will permit. Any divergence between the IEC recommendation and the corresponding national rules should, as far as possible, be clearly indicated in the latter.

PREFACE

This standard has been prepared by Sub-Committee 47B: Microprocessor Systems, of IEC Technical Committee No. 47: Semiconductor Devices and Integrated Circuits.

A first draft* was discussed at the meeting held in Montreux in 1981. As a result of this meeting, a draft, Document 47B(Central Office)2, was submitted to the National Committees for approval under the Six Months' Rule in November 1981.

The National Committees of the following countries voted explicitly in favour of publication:

Austria	Poland
Belgium	Romania
Canada	South Africa (Republic of)
Egypt	Spain
France	Sweden
Germany	Switzerland
Ireland	Union of Soviet Socialist Republics
Italy	United States of America
Japan	Yugoslavia
Netherlands	

* This first draft was prepared in accordance with draft 8.0 of I.E.E.E. Task P754.

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

ARITHMÉTIQUE BINAIRE EN VIRGULE FLOTTANTE POUR SYSTÈMES A MICROPROCESSEURS

---

1. Domaine d'application

Cette norme est appliquable à l'arithmétique binaire en virgule flottante pour systèmes à microprocesseurs.

Elle précise:

a) les formats des nombres en virgule flottante;

b) les résultats pour l'addition, la soustraction, la multiplication, la division, la racine carrée, le calcul d'un reste et la comparaison;

c) les conversions entre entiers et nombres en virgule flottante;

d) les conversions entre différents formats en virgule flottante;

e) les conversions entre le format de base (voir le paragraphe 4.1) des nombres en virgule flottante et les chaînes décimales;

f) la détection et le traitement des anomalies des nombres en virgule flottante, y compris les non-nombres («NaN»).

Cette norme ne spécifie pas:

a) la représentation des nombres entiers;

b) l'interprétation des signes et champs fractionnaires des non-nombres («NaN»);

c) les conversions de binaire en décimal et inverse avec les formats étendus;

d) les formats des chaînes décimales.

2. Objet

L'objet de cette norme est de définir de quelle façon les nouveaux systèmes doivent manipuler l'arithmétique binaire en virgule flottante. Il est prévu que les réalisations d'un système de calcul en virgule flottante conforme à cette norme puissent se faire entièrement en logiciel, entièrement en matériel ou dans une combinaison quelconque des deux.

INTERNATIONAL ELECTROTECHNICAL COMMISSION

BINARY FLOATING POINT ARITHMETIC FOR MICROPROCESSOR SYSTEMS

---

1. Scope

This standard applies to binary floating-point arithmetic used in microprocessor systems.

This standard specifies:

a) floating-point number formats;

b) the results for add, subtract, multiply, divide, square root, remainder and compare;

c) conversions between integers and floating-point numbers;

d) conversions between different floating-point formats;

e) conversions between basic format (see Sub-clause 4.1) floating-point numbers and decimal strings;

f) floating-point exceptions and their handling, including non-numbers (NaNs).

This standard does not specify:

a) integer representation;

b) the interpretation of the signs and the fraction field of non-numbers (NaNs);

c) binary - decimal conversions to and from extended formats;

d) formats of decimal strings.

2. Object

The object of this standard is to define ways for new systems to perform binary floating-point arithmetic. It is intended that an implementation of a floating-point system conforming to this standard can be realized entirely in software, entirely in hardware or in any combination of hardware and software.

C'est tout l'environnement réel que le programmeur ou l'utilisateur du système voit qui est conforme ou non à cette norme. Les composants matériels qui nécessitent un support logiciel spécial pour être conformes à la norme ne doivent pas être dits conformes sans mention de ce logiciel spécial.

3. Terminologie

3.1 *Niveau d'exigence*

Dans cette norme, l'usage du verbe:

— «doit» implique que le respect de la spécification est exigé pour être compatible avec la norme;

— «devrait» implique que le respect de la spécification est fortement recommandé, mais n'est pas obligatoire pour être compatible avec la norme.

3.2 *Définitions*

3.2.1 *Utilisateur*

Est considéré comme utilisateur d'un système en virgule flottante toute personne, tout matériel ou programme, non précisés dans cette norme, qui a accès aux opérations effectuées dans l'environnement logiciel spécifié dans cette norme et qui les contrôle.

3.2.2 *Nombre binaire exprimé en virgule flottante*

Chaîne de bits caractérisée par trois éléments: un signe, un exposant signé et une mantisse. Sa valeur numérique, si elle existe, est le produit signé de sa mantisse par deux, à la puissance donnée par l'exposant. Dans cette norme, une chaîne de bits n'est pas toujours distinguée du nombre qu'elle peut représenter.

3.2.3 *Exposant*

Cet élément d'un nombre binaire exprimé en virgule flottante indique généralement la puissance à laquelle deux est élevé pour calculer la valeur du nombre représenté. Occasionnellement, l'exposant est appelé «exposant signé» ou «sans excédent».

3.2.4 *Exposant avec excédent*

Somme de l'exposant et d'une constante (excédent ou biais) choisie de sorte que le champ de l'exposant avec excédent ne soit jamais négatif.

3.2.5 *Mantisse*

Elément d'un nombre binaire exprimé en virgule flottante constitué d'un bit significatif explicite ou implicite à gauche de la virgule et d'un champ fractionnaire à droite de la virgule.

3.2.6 *Partie fractionnaire*

Champ de la mantisse situé à droite de la virgule correspondante.

It is the actual environment which the programmer or user of the system sees that conforms or fails to conform to this standard. Hardware components that require software support to conform shall not be said to conform apart from such software.

3. Terminology

3.1 *Degree of requirements*

In this standard, the auxiliary verb:

— "shall" implies that compliance with a requirement is mandatory for compliance with the standard;

— "should" implies that compliance with a requirement is strongly recommended but is not mandatory for compliance with the standard.

3.2 *Definitions*

3.2.1 *User*

The user of a floating-point system is considered to be any person, hardware, or program, not itself specified by this standard, having access to and controlling those operations of the programming environment specified in this standard.

3.2.2 *Binary floating-point number*

A bit-string characterized by three components, a sign, a signed exponent, and a mantissa. Its numerical value, if any, is the signed product of its mantissa and two raised to the power of its exponent. In this document a bit-string is not always distinguished from a number it may represent.

3.2.3 *Exponent*

That component of a binary floating-point number which normally signifies the power to which two is raised in determining the value of the represented number. Occasionally the exponent is called the signed or unbiased exponent.

3.2.4 *Biased exponent*

The sum of the exponent and a constant (bias) chosen to make the range of the biased exponent non-negative.

3.2.5 *Mantissa*

That component of a binary floating-point number which consists of an explicit or implicit leading bit to the left of its binary point and a fraction field to the right of the binary point.

3.2.6 *Fraction*

That field of the mantissa that lies to the right of its implied binary point.

3.2.7 *Zéro normalisé*

L'exposant a la valeur minimale correspondant au format et la mantisse est zéro. Le zéro normalisé peut avoir soit un signe positif soit un signe négatif. Seuls les formats étendus ont des zéros non normalisés (voir paragraphe 3.2.9).

3.2.8 *Nombre dénormalisé*

L'exposant a la valeur minimale correspondant au format, le bit de poids fort de la mantisse (explicite ou implicite) est nul et le nombre n'est pas un zéro normalisé. Dénormaliser un nombre binaire exprimé en virgule flottante signifie décaler sa mantisse à droite tout en incrémentant son exposant, jusqu'à ce qu'on obtienne un nombre dénormalisé.

3.2.9 *Nombre non normalisé*

Ce type de nombre n'apparaît que pour le format étendu. L'exposant du nombre est supérieur à la valeur minimale correspondant au format et le bit le plus significatif explicite est nul. Si la mantisse est nulle, c'est un zéro non normalisé.

3.2.10 *Normaliser*

Si le nombre est non nul, normaliser revient à décaler sa mantisse à gauche en décrémentant son exposant jusqu'à ce que le bit de poids fort de la mantisse soit un; l'exposant est traité comme si son champ était illimité. Si la mantisse est nulle, le nombre est remplacé par le zéro normalisé. La normalisation d'un nombre ne change pas son signe.

3.2.11 *Non-nombre («NaN»)*

Entité symbolique codée selon le format virgule flottante. Voir l'article 4 et le paragraphe 7.2.

3.2.12 *Indicateur d'état*

Variable qui peut prendre deux états: actif (à 1) ou inactif (à 0). Un programme peut désactiver ou copier un indicateur. Un indicateur d'état actif peut contenir des informations additionnelles qui dépendent du système et qui peuvent être inaccessibles à l'utilisateur. Les opérations définies par cette norme peuvent avoir comme effet secondaire d'activer certains indicateurs: résultat inexact, dépassement de capacité inférieure, dépassement de capacité (supérieure), division par zéro et opération invalide.

3.2.13 *Résultat*

Chaque opération unaire ou binaire mémorise son résultat dans un emplacement de résultat, qui peut être explicitement désigné par l'utilisateur ou fourni implicitement par le système (par exemple résultats intermédiaires dans des sous-expressions ou arguments de procédures). Quelques langages placent les résultats intermédiaires dans des emplacements de résultats qui ne sont pas sous le contrôle de l'utilisateur. Néanmoins cette norme définit le résultat d'une opération en donnant le format de l'emplacement du résultat aussi bien que les valeurs des opérandes.

3.2.14 *Modes*

Variable qu'un programme peut activer, tester, sauver et rétablir afin de commander l'exécution des opérations arithmétiques concernées. Le mode par défaut est le mode

3.2.7 *Normal zero*

The exponent is the minimum value of the format, and the mantissa is zero. Normal zero may have either a positive or a negative sign. Only the extended formats have any unnormalized zeros (see Sub-clause 3.2.9).

3.2.8 *Denormalized Number*

The exponent of the number is the minimum value of the format, the explicit or implicit leading bit of the significand is zero, and the number is not normal zero. To denormalize a binary floating-point number means to shift its mantissa right while incrementing its exponent, until it becomes a denormalized number.

3.2.9 *Unnormalized Number*

This type of number occurs only in the extended format. The exponent of the number is greater than the minimum value of the format, and the explicit leading bit is zero. If the mantissa is zero, this is an unnormalized zero.

3.2.10 *Normalize*

If the number is nonzero, the number is normalized by shifting its mantissa left while decrementing its exponent until the leading bit of the mantissa becomes one; the exponent is regarded as if its range were unlimited. If the mantissa is zero, the number is replaced by a normal zero. Normalizing a number does not change its sign.

3.2.11 *Non-Number (NaN)*

This is a symbolic entity encoded in floating-point format. See Clause 4 and Sub-clause 7.2.

3.2.12 *Status flag*

A variable which may take two states, set and clear. A program may clear or copy a flag. When set, a status flag may contain additional system-dependent information, possibly inaccessible to the user. The operations of this standard may, as a side effect, set some of the following flags: inexact result, underflow, overflow, divide by zero, and invalid operation.

3.2.13 *Destination*

Every unary or binary operation delivers its result to a destination, either explicitly designated by the user or implicitly supplied by the system (e.g. intermediate results in subexpresssions or arguments for procedures). Some languages place the results of intermediate calculations in destinations beyond the control of the user. None the less, this standard defines the result of an operation in terms of that destination format as well as the values of the operands.

3.2.14 *Mode*

A mode is a variable which a program may set, sense, save and restore, to control the execution of subsequent arithmetic operations. The default mode is that mode which a

valable tant qu'une instruction contraire explicite n'est pas contenue dans le programme ou ses spécifications. Cette norme comporte les modes suivants:

a) projectif/affine, qui concerne l'interprétation de la valeur infinie notée (voir paragraphe 7.1);

b) sens de l'arrondi, qui concerne le sens des erreurs d'arrondi (voir paragraphe 5.2).

Dans certaines utilisations, on a encore le mode suivant:

c) précision de l'arrondi, pour diminuer la précision des résultats intermédiaires.

Une utilisation particulière peut avoir les modes optionnels suivants:

d) avertissement/normalisation pour le traitement des dépassements de capacité inférieure;

e) pièges autorisés ou non, pour traiter les anomalies.

4. Formats

Cette norme définit quatre formats en virgule flottante, classés selon deux groupes, de base et étendu, chacun admettant deux longueurs: simple et double précision.

4.1 *Formats de base*

4.1.1 *Simple précision*

Le format 32 bits pour un nombre binaire en virgule flottante X est divisé comme indiqué dans la figure 1. Les champs de X sont le signe s (1 bit), l'exposant avec excédent e (8 bits) et la mantisse (j,f) comportant une partie entière implicite j et une partie fractionnaire f (23 bits). La valeur v de X vaut:

a) Si $e = 255$ et $f \neq 0$ alors $v =$ NaN

b) Si $e = 255$ et $f = 0$ alors $v = (-1)^s \infty$

c) Si $0 < e < 255$ alors $v = (-1)^s 2^{e-127} (1,f)$

d) Si $e = 0$ et $f \neq 0$ alors $v = (-1)^s 2^{-126} (0,f)$

e) Si $e = 0$ et $f = 0$ alors $v = (-1)^s 0$ (zéro)

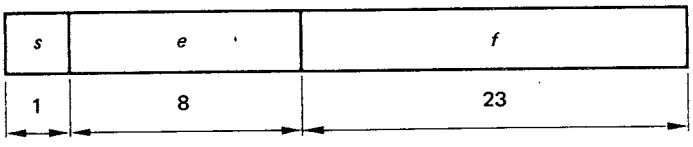


FIGURE 1. — Format de base en simple précision

4.1.2 *Double précision*

Un format 64 bits pour un nombre binaire en virgule flottante X est divisé comme indiqué dans la figure 2. Les champs de X sont le signe s (1 bit), l'exposant avec excédent e

program can assume to be in effect unless an explicitly contrary statement is included either in the program or its specification. This standard includes the following modes:

a) Projective/affine, which concerns the interpretation of infinity, (See Sub-clause 7.1);

b) Rounding direction, which concerns the direction of rounding errors, (See Sub-clause 5.2).

In certain implementations, one also has the following mode:

c) Rounding precision, to shorten the precision of intermediate results.

In particular implementations, one may have the following optional modes:

d) Warning/normalizing, for handling underflowed values, and

e) Traps disabled/enabled, for handling exceptions.

4. Formats

This standard defines four floating-point formats in two groups, basic and extended, each having two widths, single-precision and double-precision.

4.1 *Basic formats*

4.1.1 *Single-Precision*

A 32-bit format for a binary floating-point number X is divided as shown in Figure 1. The component fields of X are the 1-bit sign s, the 8-bit biased exponent e, and the mantissa $(j.f)$ consisting of an implied integer part j and a 23-bit fraction f. The value v of X is as follows:

a) If $e = 255$ and $f \neq 0$, then v = NaN.

b) If $e = 255$ and $f = 0$, then $v = (-1)^s \infty$

c) If $0 < e < 255$, then $v = (-1)^s 2^{e-127} (1.f)$.

d) If $e = 0$ and $f \neq 0$, then $v = (-1)^s 2^{-126} (0.f)$.

e) If $e = 0$ and $f = 0$, then $v = (-1)^s 0$, (zero).

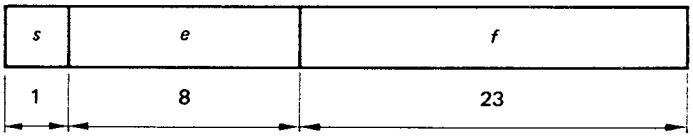


FIGURE 1. — Single-precision format

4.1.2 *Double-Precision*

A 64-bit format for a binary floating-point number X is divided as shown in Figure 2. The component fields of X are the 1-bit sign s, the 11-bit biased exponent e, and the man-

(11 bits) et une mantisse (j,f) comportant une partie entière implicite j et une partie fractionnaire f (52 bits). La valeur v de X est la suivante:

a) Si $e = 2047$ et $f \neq 0$ alors $v =$ NaN

b) Si $e = 2047$ et $f = 0$ alors $v = (-1)^s \infty$

c) Si $0 < e < 2047$ alors $v = (-1)^s 2^{e-1023} (1,f)$

d) Si $e = 0$ et $f \neq 0$ alors $v = (1-1)^s 2^{-1022} (0,f)$

e) Si $e = 0$ et $f = 0$ alors $v = (-1)^s 0$ (zéro)

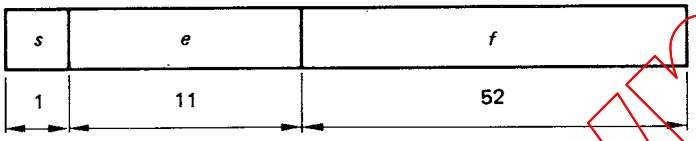


FIGURE 2. — Format de base en double précision

4.2 *Formats étendus*

4.2.1 *Format étendu en simple précision*

Ce format étendu dépend de l'utilisation. Un nombre binaire étendu exprimé en virgule flottante X a quatre composants: un signe s (1 bit), un exposant e pouvant se trouver dans un champ spécifié et combiné avec un excédent dépendant de l'utilisation, une partie entière j (1 bit), et une partie fractionnaire f comportant au minimum 31 bits. L'exposant peut varier entre la valeur minimale $m \leqslant -1023$ et la valeur maximale $M \geqslant +1024$.

La valeur v de X vaut alors:

a) Si $e = M$ et $f \neq 0$ alors $v =$ NaN

b) Si $e = M$ et $f = 0$ alors $v = (-1)^s \infty$

c) Si $m < e < M$ alors $v = (-1)^s 2^e (j,f)$

d) Si $e = m$ et $j = f = 0$ alors $v = (-1)^s 0$ (zéro normalisé)

e) Si $e = m$ et j ou f non nul, alors $v = (-1)^s 2^{e'} (j,f)$

où $e' = m$ ou $m + 1$ au choix de l'utilisateur.

4.2.2 *Format étendu en double précision*

Le format étendu en double précision est le même que celui qui a été décrit au paragraphe 4.2.1, mais avec l'exposant compris entre les limites $m \leqslant -16383$ et $M \geqslant +16384$, et une partie fractionnaire comportant au moins 63 bits.

4.2.3 *Champ de l'exposant*

Pour une utilisation de la norme, il n'est pas nécessaire de stipuler (et l'utilisateur ne devrait pas présumer) que les extensions en simple précision ont un champ plus vaste que les extensions en double précision.

tissa $(j.f)$ consisting of an implied integer part j and a 52-bit fraction f. The value v of X is as follows:

a) If $e = 2047$ and $f \neq 0$, then $v = \text{NaN}$.

b) If $e = 2047$ and $f = 0$, then $v = (-1)^s \infty$

c) If $0 < e < 2047$, then $v = (-1)^s 2^{e-1023} (1.f)$.

d) If $e = 0$ and $f \neq 0$, then $v = (-1)^s 2^{-1022} (0.f)$.

e) If $e = 0$ and $f = 0$, then $v = (-1)^s 0$, (zero).

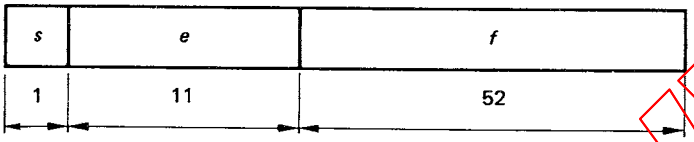


FIGURE 2. — Double-precision format

4.2 *Extended formats*

4.2.1 *Single-precision-extended*

This format is an implementation-dependent format. An extended binary floating-point number X has four components: a 1-bit sign s, an exponent e of specified range combined with an implementation-dependent bias, a 1-bit integer part j, and a fraction f with at least 31 bits. The exponent shall range between a minimum value $m \leqslant -1023$ and a maximum value $M \geqslant +1024$.

The value v of X is as follows:

a) If $e = M$ and $f \neq 0$, then $v = \text{NaN}$

b) If $e = M$ and $f = 0$, then $v = (-1)^s \infty$

c) If $m < e < M$, then $v = (-1)^s 2^e (j.f)$

d) If $e = m$ and $j = f = 0$, then $v = (-1)^s 0$ (normal zero)

e) If $e = m$ and j or f is nonzero, then $v = (-1)^s 2^{e'} (j.f)$

where $e' = m$ or $m + 1$ at the implementor's option.

4.2.2 *Double-precision-extended*

The double-precision-extended format is the same as single-extended described in Subclause 4.2.1, except that the exponent shall range between $m \leqslant -16383$ and $M \geqslant +16384$, and the fraction shall have at least 63 bits.

4.2.3 *Exponent range*

An implementation of this standard is not required to provide (and the user should not assume) that single-precision-extended has greater range than double-precision-extended.

4.3 *Combinaison de formats*

Toutes les utilisations conformes à cette norme doivent supporter le format étendu en simple précision et n'ont pas besoin de supporter d'autres formats étendus. Les utilisations devraient supporter le format étendu correspondant au plus grand format de base supporté.

Note. — Un système supportant le format étendu en double précision ne devrait supporter également le format étendu en simple précision que dans les cas où la compatibilité avec des systèmes plus complexes et plus rapides est importante.

5. Arrondi

A l'exception de la conversion binaire-décimale, toutes les opérations décrites aux articles 6 et 8 doivent être effectuées comme si elles avaient une précision infinie, puis arrondies selon les spécifications de cet article. L'arrondi part d'un nombre considéré comme infiniment précis et, si nécessaire, le modifie pour entrer dans le format du résultat, tout en signalant qu'il est inexact (voir paragraphe 9.5).

5.1 *Mode d'arrondi par défaut*

Une utilisation de cette norme doit fournir un arrondi au plus près, avec arrondi pair en cas d'indécision, comme mode d'arrondi par défaut. Lorsqu'on arrondit au plus près, le résultat doit s'écarter du résultat exact d'au maximum la moitié du digit le moins significatif; effectuer un arrondi pair signifie que si la différence est exactement la moitié, alors le résultat doit avoir un dernier digit pair.

5.2 *Modes d'arrondi orientés*

Une utilisation doit offrir à l'utilisateur le choix entre arrondis orientés positif ou négatif (arrondi vers $+\infty$ ou arrondi vers $-\infty$) et arrondi tronqué (arrondi vers 0) pour toutes les opérations.

Lorsqu'un arrondi est orienté vers $+\infty$, le résultat doit être la valeur du format (y compris $+\infty$) la plus proche et non inférieure au résultat exact, à l'exception des cas décrits au paragraphe 9.3. Dans le cas d'un arrondi orienté vers $-\infty$, le résultat doit être la valeur du format (y compris $-\infty$) la plus proche et non supérieure au résultat exact, à l'exception des cas signalés au paragraphe 9.3. Dans le cas d'un arrondi vers zéro, le résultat doit être la valeur du format la plus proche et non supérieure en valeur absolue au résultat dont l'exactitude tend vers l'infini. Les modes d'arrondi peuvent affecter le signe de sommes nulles (voir paragraphe 7.3).

5.3 *Précision de l'arrondi*

En général, un résultat est arrondi avec la précision correspondant à son format. Cependant certains matériels donnent toujours, à partir d'opérandes en simple précision, des résultats en double précision ou en précision étendue. L'utilisateur d'un tel système, qui peut être un compilateur d'un langage de haut niveau, doit pouvoir spécifier qu'un résultat doit être arrondi en simple précision, même lorsqu'il est mémorisé dans un format en double précision ou étendu avec un exposant ayant un domaine plus étendu (voir note). De manière semblable, un système qui met en format étendu en précision tous les résultats d'opérations en double précision doit permettre à l'utilisateur de spécifier «arrondi en double précision». Remarquez que pour satisfaire aux spécifications données au paragraphe 5.1, le résultat ne peut pas être entaché de plus d'une erreur d'arrondi.

4.3 *Combinations of format*

All implementations conforming to this standard shall support single-precision. Implementations should support the extended format corresponding to the widest basic format supported, and need not support any other extended format.

Note. — A system supporting the double-extended format should also support single-extended only if upward compatibility and speed are important issues.

5. **Rounding**

Except for binary-decimal conversion, all operations specified in Clauses 6 and 8 shall be performed as if correct to infinite precision, then rounded according to the specifications in this section. Rounding takes a number regarded as infinitely precise and, if necessary, modifies it to fit in the format of the destination while signalling that it is inexact (see Sub-clause 9.5).

5.1 *Default rounding mode*

An implementation of this standard shall provide rounding to nearest, with rounding to even in case of a tie, as the default rounding mode. When rounding to nearest, the result shall differ from the infinite precision exact result by at most one-half in the least-significant-digit position; rounding to even means that if the difference is exactly half then the rounded result shall have an even last digit.

5.2 *Directed rounding modes*

An implementation shall provide user-selectable positive- and negative-directed rounding (round toward $+\infty$ and round toward $-\infty$) and truncation (round toward 0) for all operations.

When rounding toward $+\infty$, the result shall be the format's value (possibly $+\infty$) closest to and no less than the infinitely precise result, except as specified in Sub-clause 9.3, analogously, when rounding toward $-\infty$, the result shall be the format's value (possibly $-\infty$) closest to and no greater than the infinitely precise result, except as specified in Sub-clause 9.3. When rounding toward 0, the result shall be the format's value closest to and no greater in magnitude than the infinitely precise result.

The rounding modes may affect the signs of zero sums (see Sub-clause 7.3).

5.3 *Rounding precision*

Normally a result is rounded to the precision of its destination format. However, some hardware will always deliver results from single precision format operands to double or extended precision. On such a system the user, which may be a high-level language compiler, shall be able to specify that a result be rounded instead to single precision, though it is stored in the double or extended format with its wider exponent range (see note). Similarly, a system that delivers all results from double format operands to extended destinations shall permit the user to specify rounding to double precision. Note that to meet the specifications in Sub-clause 5.1, the result cannot suffer more than one rounding error.

Note. — La commande de la précision de l'erreur d'arrondi a pour but de permettre à des systèmes, pour lesquels les résultats sont toujours en double précision ou étendue, de simuler des systèmes avec des résultats en simple ou en double précision. Toutefois, l'utilisation de la commande de précision pour obtenir, à partir des opérandes en double précision (ou étendus), un résultat en simple précision avec un seul arrondi n'est pas considéré comme une procédure normale.

6. Opérations

Toutes les utilisations conformes à cette norme doivent permettre les opérations suivantes: addition, soustraction, multiplication, division, extraction de la racine, reste, conversion des formats en virgule flottante, conversion entre virgule flottante et entiers, conversions binaires-décimales et comparaisons.

Lorsque tous les opérandes sont normalisés, les opérations doivent être exécutées avec une précision infinie avant d'arrondir le résultat, comme décrit à l'article 5. L'article 8 donne les résultats dans le cas où au moins un des opérandes n'est pas normalisé. L'article 7 élargit les spécifications afin de couvrir les cas du zéro signé, de l'∞ et des NaN. L'article 9 énumère les anomalies.

6.1 *Arithmétique*

Une utilisation doit permettre l'addition, la soustraction, la multiplication, la division et le reste d'une division pour tout couple d'opérandes de même format, pour chaque format supporté. Elle devrait également permettre d'effectuer ces opérations pour des opérateurs de différents formats. Le format du résultat (indépendamment de la commande de la précision d'arrondi du paragraphe 5.3) doit être au moins aussi étendu que le format des opérandes. Tous les résultats doivent être arrondis comme indiqué à l'article 5.

Le reste $r = x$ REM y est défini indépendemment du mode d'arrondi par la relation suivante, où $y \neq 0$,

$$r = x - y \cdot n,$$

n étant l'entier le plus proche de x/y; n est pair si $|n - x/y| = ½$. Notons qu'avec cette définition, le reste est exact. Le résultat doit être normalisé, sauf en cas de dépassement de capacité inférieure.

6.2 *Racine carrée*

L'extraction de la racine carrée doit être possible pour tous les formats supportés, et définie pour tous les opérandes normalisés $\geqslant 0$; $\sqrt{-0} = -0$. Le format du résultat doit être arrondi d'après les spécifications de l'article 5.

6.3 *Conversion du format flottant*

Il doit être possible de convertir des nombres en virgule flottante d'un format supporté à l'autre. Si le résultat de la conversion a une précision moindre, le résultat sera arrondi comme spécifié à l'article 5. Si le résultat de la conversion est de précision supérieure, il doit être exact, bien qu'une anomalie puisse se produire comme spécifié au paragraphe 9.1.2 (résultat invalide).

6.4 *Conversion entre virgule flottante et entier*

Il doit être possible d'arrondir un nombre en virgule flottante à une valeur entière dans le même format en virgule flottante, pour tous les formats supportés. L'arrondi doit être fait

Note. — Rounding precision control is intended to allow systems whose destinations are always double or extended to mimic systems with single and double destinations. However, use of precision control to combine double (or extended) operands to produce a single format result with just one rounding is considered not to be a standard procedure.

6. Operations

All conforming implementations of this standard shall provide add, subtract, mutiply, divide, square root, remainder, floating-point format conversions, conversions between floating-point and integers, binary-decimal conversions, and comparisons.

When all operands are normalized, the operations shall be performed as if to infinite precision before rounding as specified in Clause 5. Clause 8 specifies the results when at least one of the operands is not normalized. Clause 7 augments the specifications to cover signed zero and ∞ and NaN; Clause 9 enumerates exceptions.

6.1 *Arithmetic*

An implementation shall provide add, subtract, multiply, divide, and remainder for any two operands of the same format, for each supported format; it should also provide the operations for operands of differing formats. The destination format (regardless of the rounding precision control of Sub-clause 5.3) shall be at least as wide as the operands' format. All results shall be rounded as specified in Clause 5.

The remainder $r = x$ REM y is defined regardless of the rounding mode by the following relation when $y \neq 0$:

$$r = x - y \cdot n$$

where n is the integer nearest x/y; n is even whenever $|n - x/y| = ½$. Note that with this definition the remainder is exact. The result shall be normalized unless it underflows.

6.2 *Square root*

The square root operation shall be provided in all supported formats and is defined for all normalized operands $\geqslant 0$; $\sqrt{-0} = -0$. The destination format shall be at least as wide as the operand's. The result shall be rounded as specified in Clause 5.

6.3 *Floating-point format conversions*

It shall be possible to convert floating-point numbers between all supported formats. If the conversion is to a less wide precision, the result shall be rounded as specified in Clause 5. If the conversion is to a wider precision, it shall be exact, although an invalid result exception may be raised as specified in Sub-clause 9.1.2.

6.4 *Conversion between floating-point and integer*

It shall be possible to round a floating-point number to an integer value in the same floating-point format, for all supported formats. The rounding shall be as specified in Clause 5,

comme spécifié à l'article 5, en tenant compte du fait qu'avec le mode «arrondi au plus près», si la différence entre le résultat non arrondi et le résultat arrondi est exactement ½, le résultat arrondi est pair.

Il doit être possible de convertir entre tous les formats supportés (en virgule flottante ou entiers). La conversion en nombres entiers doit être effectuée en arrondissant conformément aux indications de l'article 5. La conversion entre formats de nombres entiers en virgule flottante et formats de nombres entiers doit être exacte, sauf pour les anomalies citées au paragraphe 9.1.1.

6.5 *Conversion binaire-décimale*

La conversion entre chaînes décimales dans un format au moins et nombres binaires en virgule flottante dans tous les formats supportés doit être possible pour tous les nombres, dans le domaine spécifié dans le tableau I. Les entiers M et N des tableaux I et II sont tels que les chaînes décimales ont pour valeur $\pm M \cdot 10^{\pm N}$. A l'entrée, les zéros seront ajoutés ou ôtés de M (dans les limites spécifiées au tableau I) afin de minimiser N. Lorsque le résultat est une chaîne décimale, son digit le moins significatif doit être localisé par les spécifications pour l'arrondi.

Les conversions doivent être arrondies correctement, comme spécifié à l'article 5 pour des opérandes situés dans les domaines décrits dans le tableau II. Sinon l'erreur du résultat converti ne doit pas dépasser de plus de 0,47 unité du digit le moins significatif du résultat, l'erreur qui résulterait en appliquant les spécifications de l'article 5, pour autant qu'il n'y ait aucun dépassement de capacité concernant l'exposant.

Les conversions doivent être monotones. En d'autres termes, l'augmentation de la valeur d'un nombre binaire exprimé en virgule flottante ne doit pas diminuer la valeur du résultat converti en chaîne décimale; et l'augmentation de la valeur de la chaîne décimale ne doit pas diminuer sa valeur convertie en nombre binaire flottant.

Lorsqu'on arrondit au plus près, la suite d'une conversion de binaire en décimal et retour au binaire doit redonner la valeur initiale tant que la chaîne décimale a la précision maximale spécifiée dans le tableau I, c'est-à-dire 9 digits en simple précision et 17 en double précision (voir note).

Lors d'un dépassement de capacité dans une conversion décimal-binaire, le résultat est indiqué à l'article 9. Les dépassements de capacité « NaN » et infinis produits lors d'une conversion binaire-décimal devraient être indiqués à l'utilisateur par des chaînes adéquates.

Note. — Les propriétés définies pour les conversions résultent de limites d'erreur qui dépendent du format (simple ou double précision) et du nombre de chiffres décimaux qui interviennent; la valeur 0,47 mentionnée correspond au cas le pire.

TABLEAU I

Domaine de conversion décimale

Format	Décimal-binaire		Binaire-décimal	
	M max	N max	M max	N max
Simple	$10^9 - 1$	99	$10^9 - 1$	54
Double	$10^{19} - 1$	999	$10^{17} - 1$	341

with the understanding that in rounding to nearest mode, if the difference between the unrounded operand and the rounded result is exactly one-half, the rounded result is even.

It shall be possible to convert between all supported floating-point formats and all supported integer formats. Conversion to integer shall be effected by rounding as specified in Clause 5. Conversions between floating-point integers and integer formats shall be exact unless an exception arises as specified in Sub-clause 9.1.1.

6.5 *Binary - decimal conversion*

Conversion between decimal strings in at least one format and binary floating-point numbers in all supported basic formats shall be provided for numbers throughout the ranges specified in Table I. The integers M and N in Tables I and II below are such that the decimal strings have values $\pm M \cdot 10^{\pm N}$. On input, trailing zeros shall be appended to or stripped from M (up to the limits specified in Table I) in order to minimize N. When the destination is a decimal string, its least- significant digit should be located by format specifications for purposes of rounding.

Conversions shall be correctly rounded as specified in Clause 5 for operands lying within the ranges specified in Table II. Otherwise the error in the converted result shall not exceed by more than 0.47 units in the destination's least-significant digit the error that would be incurred by the rounding specifications of Clause 5, provided that exponent over/underflow does not occur.

Conversions shall be monotonic. In other words, increasing the value of binary floating-point number shall not decrease its value when converted to a decimal string, and increasing the value of a decimal string shall not decrease its value when converted to a binary floating-point number.

When rounding to nearest, conversion from binary to decimal and back to binary shall produce the initial value as long as the decimal string is carried to the maximum precision specified in Table I, namely nine digits for single precision and 17 for double precision (see note).

If decimal to binary conversion over/underflows, the response is as specified in Clause 9. Over/underflow and NaNs and infinities encountered during binary to decimal conversion should be indicated to the user by appropriate strings.

Note. — The properties specified for conversions are implied by error bounds that depend on the format (single or double) and the number of decimal digits involved; the 0.47 mentioned is a worst-case bound only.

TABLE I

Decimal conversion ranges

Format	Decimal to binary max M	Decimal to binary max N	Binary to decimal max M	Binary to decimal max N
Single	$10^9 - 1$	99	$10^9 - 1$	54
Double	$10^{19} - 1$	999	$10^{17} - 1$	341

TABLEAU II

Domaine d'arrondi exact pour la conversion décimale

Format	Décimal-binaire		Binaire-décimal	
	M max	*N* max	*M* max	*N* max
Simple	$10^9 - 1$	13	$10^9 - 1$	13
Double	$10^{19} - 1$	27	$10^{17} - 1$	27

6.6 *Comparaison*

Il doit être possible de comparer des nombres en virgule flottante dans tous les formats supportés, y compris les comparaisons entre opérandes ayant différents formats. Les comparaisons sont exactes et ne créent jamais de dépassement de capacité. Il existe quatre relations possibles, qui s'excluent mutuellement: «plus petit que», «égal», «plus grand que » et «non ordonné». Le dernier cas survient quand un des opérateurs au moins est un NaN, ou lorsque l'∞ dans le mode projectif est comparé à n'importe quel nombre différent de ∞; la comparaison de tout NaN avec n'importe quoi, y compris soi-même, doit répondre «non ordonné». La comparaison doit ignorer le signe de l'∞ dans le mode projectif (où $+\infty = -\infty$) et le signe du zéro (ainsi $+0 = -0$).

6.6.1 *Représentation des résultats*

Lorsque le résultat d'une comparaison est donné par des codes, le résultat doit être un codage de l'une des quatre relations données au paragraphe 6.6.

6.6.2 *Prédicats*

Quand le résultat d'une comparaison est donné sous la forme de l'affirmation ou de la négation d'un prédicat, les implications suivantes déterminent la réponse:

a) La relation «plus petit que» affirme les prédicats $<$, $\leqslant$, $\neq$ et nie les prédicats $=$, $\geqslant$, $>$, non ordonné.

b) La relation «égal» affirme les prédicats $=$, $\leqslant$, $\geqslant$ et nie les prédicats $<$, $>$, $\neq$, non ordonné.

c) La relation «plus grand que» affirme les prédicats $>$, $\geqslant$, $\neq$ et nie les prédicats $=$, $\leqslant$, $<$, non ordonné.

d) La relation «non ordonné» affirme les prédicats $\neq$, non ordonné et nie les prédicats $<$, $\leqslant$, $=$, $\geqslant$, $>$.

En complément aux réponses spécifiées ci-dessus, une exception «opération invalide» (voir paragraphe 9.1) se produit dans une comparaison où deux valeurs non ordonnées sont comparées avec $<$, $\leqslant$, $\geqslant$, $>$, ou leurs négations, comme décrit au paragraphe 9.1.1 lettre *h*).

TABLE II

Correctly rounded decimal conversion range

Format	Decimal to binary max M	Decimal to binary max N	Binary to decimal max M	Binary to decimal max N
Single	$10^9 - 1$	13	$10^9 - 1$	13
Double	$10^{19} - 1$	27	$10^{17} - 1$	27

6.6 *Comparison*

It shall be possible to compare floating-point numbers in all supported formats, including comparisons between operands of differing formats. Comparisons are exact and never overflow or underflow. Four mutually exclusive relations are possible: "less than", "equal", "greater than", and "unordered". The last case arises when at least one operand is NaN, or when ∞ in the projective mode is compared to anything other than ∞ every NaN shall compare "unordered" with everything, including itself. Comparisons shall ignore the sign of infinity in the projective mode (where $+\infty = -\infty$), and shall ignore the sign of zero (so $+0 = -0$).

6.6.1 *Representation of results*

When the result of a comparison is reported via condition codes, the result shall be an encoding of one of the four relations listed in Sub-clause 6.6.

6.6.2 *Predicates*

When the result of a comparison is reported as an affirmation or negation of a predicate, the following implications shall determine that response:

a) The relation "less than" affirms the predicates $<$, $\leqslant$, $\neq$, and denies the predicates $=$, $\geqslant$, $>$, unordered.

b) The relation "equal" affirms the predicates $=$, $\leqslant$, $\geqslant$, and denies the predicates $<$, $>$, $\neq$, unordered.

c) The relation "greater than" affirms the predicates $>$, $\geqslant$, $\neq$, and denies the predicates $=$, $\leqslant$, $<$, unordered.

d) The relation "unordered" affirms the predicates $\neq$, "unordered", and denies the predicates $<$, $\leqslant$, $=$, $\geqslant$, $>$.

In addition to the response specified above, an invalid operation exception (see Sub-clause 9.1) shall arise in a comparison just when two values whose relation is "unordered" are compared via a predicate involving $<$, $\leqslant$, $\geqslant$, $>$, or their negations, as specified in Item *h)* of Sub-clause 9.1.1.

7. Infini, NaN et zéro signé

7.1 *Arithmétique de l'infini*

L'arithmétique de l'infini doit être considérée comme la limite de l'arithmétique réelle, avec des opérateurs aussi grands que l'on veut, lorsqu'une telle limite existe. L'arithmétique de l'infini doit être supportée dans les deux modes affine et projectif, au choix de l'utilisateur, le mode projectif étant le mode par défaut. Dans le mode affine, on a $-\infty <$ tout nombre fini $< +\infty$, mais dans le mode projectif, le résultat de la comparaison entre deux infinis est «égal» quel que soit leur signe, et le résultat de la comparaison avec tout autre nombre est «non ordonné». Par conséquent les deux modes se distinguent par les anomalies dans les opérations d'addition, soustraction, racine carrée et comparaison, comme spécifié à l'article 9.

A l'exception du cas « opération invalide » spécifié pour ∞, l'arithmétique de l'∞ est toujours exacte et ne doit pas créer d'anomalies. Les trois anomalies ayant affaire à l'∞ ne se produisent que si:

a) ∞ est créé à partir d'opérateurs finis par dépassement de capacité (paragraphe 9.3), division par 0 (paragraphe 9.2) sans piège ou

b) ∞ est un opérande invalide (paragraphe 9.1).

7.2 *Opérations avec des NaN*

Deux types de NaN, avec ou sans piège, doivent être supportés par toutes les opérations.

Les NaN avec pièges doivent être des opérandes réservés qui provoquent un processus d'anomalie pour «opération invalide» (paragraphe 9.1) ou une autre anomalie dépendant de l'utilisation lors de l'exécution de toute opération citée à l'article 6 (voir note).

Les NaN sans piège doivent obéir aux règles suivantes; ces NaN devraient, par des moyens laissés à la discrétion de l'utilisateur, tenir compte rétroactivement de messages obtenus à la suite de données ou résultats invalides ou non disponibles.

Pour les opérations prévues pour donner des résultats en virgule flottante,

a) toute opération portant sur un NaN qui crée un piège ou «opération invalide» (paragraphe 9.1) devrait, si aucun piège ne survient, mettre l'indicateur « opération invalide » et donner en lieu et place du résultat invalide un NaN sans piège;

b) toute opération portant sur un ou deux « NaN » sans piège ne devrait pas signaler d'anomalie, mais donner comme résultat soit le même NaN (si l'opération se fait sur un seul opérande), soit l'un ou l'autre des «NaN» en entrée, selon des règles dépendant de l'utilisation.

Les opérations non mentionnées dans ce paragraphe, c'est-à-dire celles dont le résultat n'est pas en virgule flottante, sont la comparaison (paragraphe 6.6) et la conversion à un format sans NaN (paragraphes 6.4 et 6.5).

Note. — Ces « NaN » permettent des adjonctions semblables à celles que l'on a avec des opérations arithmétiques (telles que grandeurs infinies complexes affines, des champs très étendus, etc.) qui ne font pas l'objet de cette norme. Toutefois, s'il n'y a pas de piège spécialement prévu et activé pour ces « NaN », alors le processus d'anomalie pour « résultat invalide » est activé, comme prévu dans le paragraphe 9.1.1.

7. Infinity, NaNs, and signed zero

7.1 *Infinity arithmetic*

Infinity arithmetic shall be construed as the limiting case of real arithmetic with operands of arbitrarily large magnitude, when such a limit exists. Infinity arithmetic shall be supported under two user-selectable modes, affine and projective, with projective being the default. In affine mode $-\infty <$ (every finite number) $< +\infty$, but in projective mode infinites compare "equal" regardless of sign and compare "unordered" with everything else. Consequently, the two modes are distinguished by exceptions in add, subtract, square root, and compare, as specified in Clause 9.

Except for the invalid operations specified for ∞, arithmetic upon ∞ is always exact and therefore shall raise no exception. The three exceptions that do pertain to ∞ are raised only when

a) ∞ is created from finite operands by overflow (Sub-clause 9.3) or division by zero (Sub-clause 9.2), with the corresponding trap disabled, or

b) ∞ is an invalid operand (Sub-clause 9.1).

7.2 *Operations with NaNs*

Two different kinds of NaNs, trapping and non-trapping, shall be supported in all operations.

Trapping NaNs shall be reserved operands which precipitate an invalid operation exception (Sub-clause 9.1) or some other implementation-dependent exception for every operation listed in Clause 6 that is performed upon them (see note).

Non-trapping NaNs shall obey the following rules; these NaNs should, by means left to the implementor's discretion, afford retrospective diagnostic information inherited from invalid or unavailable data and results.

For those operations specified to deliver floating-point results,

a) every operation involving a trapping NaN or invalid operation (Sub-clause 9.1) if no trap occurs, shall set the invalid operation flag and deliver in place of its invalid result a non-trapping NaN;

b) every operation involving one or two input NaNs, none of them trapping, shall raise no exception but deliver as a result either the same NaN (if operating upon just one) or one or the other of the input NaNs, according to an implementation-dependent precedence rule.

The operations not covered in this paragraph, namely those which do not deliver a floating-point result, are comparison (Sub-clause 6.6) and conversion to a format that has no NaNs (Sub-clauses 6.4 and 6.5).

Note. — These NaNs afford arithmetic-like enhancements (such as complex affine infinities, extremely wide range, etc.) that are not the subject of this standard. However, if there is no special trap designated and enabled for these NaNs, then the invalid operation exception is raised as specified in Sub-clause 9.1.1.

7.3 *Bit de signe*

Cette norme ne parle pas du signe d'un NaN. Dans les autres cas, le signe du produit ou du quotient s'obtient par un «OU exclusif» appliqué aux signes des opérandes; et le signe de la somme, ou de la différence $x - y$ considérée comme la somme $x + (-y)$, diffère au maximum de l'un des signes des termes. Ces règles s'appliquent même lorsque les résultats ou les opérateurs sont 0 ou ∞.

Lorsque la somme de deux opérandes de signes contraires (ou la différence de deux opérateurs de même signe) est exactement zéro, normalisé ou non (voir article 8), le signe de la somme (ou différence) doit être « + » dans tous les modes d'arrondi, sauf pour l'arrondi vers $-\infty$ pour lequel le signe est « − ». Toutefois le signe de $x + x = x - (-x)$ est le signe de x, même lorsque x est nul.

Une racine carrée valide ne peut avoir un signe négatif que si l'opérande est − 0.

8. Arithmétique non normalisée et dénormalisée

Les règles suivantes s'appliquent par défaut (voir note) dans le cas où l'un des opérandes au moins n'est pas normalisé. Le traitement de l'arrondi et des dépassements de capacité est effectué après les opérations décrites ici, et peut modifier les résultats. Dans les spécifications suivantes expon(x) désigne l'exposant sans excédent de x.

a) Addition ou soustraction ($z := x \pm y$): si l'un au moins des opérandes, ayant l'exposant m (m = max (expon(x), expon(y)) est normalisé, alors z doit être normalisé avant l'arrondi. Sinon, expon(z) = m.

b) Multiplication ($z := x \cdot y$): expon(z) = expon(x) + expon(y), avec les mêmes anomalies que sous lettre *c*).

c) Division ($z := x/y$): expon(z) = expon(x) − expon(y), − 1 lorsque y est normalisé et non nul, excepté lorsque l'un seul de x et y est non normalisé et l'autre est 0 normalisé ou ∞, le quotient z est le même (0 normalisé ou ∞ ou «non valable») que si le quotient non normalisé était remplacé par son équivalent normalisé. Sinon, une anomalie doit être signalée, comme décrit au paragraphe 9.1.

d) Reste ($z := x$ REM y): z doit être calculé comme si x était d'abord normalisé.

e) Racine carrée: opération invalide si l'opérande n'est pas normalisé.

f) Conversion ($z := x$) : expon(z) = expon(x).

g) Partie entière ($z :=$ partie entière (x)): si l'exposant de x est supérieur ou égal au nombre de bits de la partie fractionnaire, alors $z = x$. Sinon z doit être normalisé.

h) La conversion de nombres binaires dénormalisés en virgule flottante selon une représentation décimale $M \cdot 10^N$, où M et N sont entiers, devrait utiliser les zéros significatifs de la représentation de M pour indiquer le degré auquel le nombre est dénormalisé.

i) Comparaison: les comparaisons doivent être faites comme si les deux opérandes avaient été préalablement normalisés.

Note. — Ces règles par défaut sont similaires à celles qui s'appliquent aux nombres normalisés.

7.3 *The sign bit*

This standard says nothing about the sign of a NaN. Otherwise the sign of a product or quotient is the exclusive OR of the operands' signs; and the sign of a sum, or of a difference $x - y$ regarded as a sum $x + (-y)$, differs from at most one of the addends' signs. These rules shall apply even when operands or results are zero or infinite.

When the sum of two operands with opposite signs (or the difference of two operands with like signs) is exactly zero, either normal or unnormalized (see Clause 8), the sign of that sum (or difference) shall be "+" in all rounding modes except round toward $-\infty$, in which mode that sign shall be "−". However, $x + x = x - (-x)$ retains the same sign as x even when x is zero.

A valid square root can have a negative sign only when the operand is −0.

8. Unnormalized and denormalized arithmetic

The default mode of arithmetic (see note), when at least one operand is not normalized, shall obey the following rules. Rounding and over/underflow handling are performed after the operations specified here and may modify the results. In the following specifications expon(x) refers to the unbiased exponent of x.

a) Add or substract ($z := x \pm y$): If at least one of the operands having exponent m, where $m = \max(\text{expon}(x), \text{expon}(y))$, is normalized, then z shall be normalized before rounding. Otherwise expon(z) = m.

b) Multiply ($z := x \cdot y$): expon(z) = expon(x) + expon(y), with the same exceptions as noted in Item *c)* if this clause.

c) Divide ($z := x/y$): expon(z) = expon(x) + expon(y) − 1 when y is normalized and nonzero, except that when only one of x and y is unnormalized and the other is normal 0 or ∞ the result z is the same (normal 0 or ∞ or invalid) as if the unnormalized operand were replaced by its normalized quivalent. Otherwise an exception shall be signalled as specified in Sub-clause 9.1.

d) remainder ($z := x$ REM y): z shall be calculated as if x were first normalized.

e) Square root is an invalid operation if its operand is not normalized.

f) Conversion ($z := x$): expon(z) = expon(x).

g) Integer conversion ($z :=$ Integer Part(x)): If expon(x) $\geqslant$ the number of fraction bits, then z shall be identically x. Otherwise z shall be normalized.

h) Conversion of denormalized binary floating-point numbers to decimal forms representing values $M \cdot 10^N$, where M and N are integers, should use leading zeros in the representation of M to indicate the degree to which the number is denormalized.

i) Compare: comparisons shall be made as if both operands had first been normalized.

Note. — These default rules are analogous to those for normalized numbers.

8.1 *Mode de normalisation*

Un autre mode arithmétique devrait être disponible, qui normaliserait toutes les valeurs dénormalisées avant d'exécuter des opérations sur elles et empêcherait la formation d'autres opérandes dénormalisés. Ceci s'applique à toutes les opérations énumérées dans ce paragraphe. Les opérandes non normalisés ne peuvent pas être affectés par ce mode.

Note. — Dans de nombreux calculs, la perte de précision due à la dénormalisation n'a pas de conséquence, et les résultats invalides (voir paragraphe 9.1.2) qui apparaissent dans le mode d'avertissement sont trop pessimistes. En conséquence, les réalisateurs sont fortement encouragés à supporter le mode normalisé, sauf peut-être dans les processus vectoriels en pipeline avec un format étendu pour l'accumulation des sommes intermédiaires. L'utilisateur d'un format étendu avec son bit significatif explicite permet le calcul des produits et quotients intermédiaires impliquant des nombres dénormalisés dans le processus d'anomalie pour résultat invalide.

9. **Anomalies**

Cinq types d'anomalies doivent être détectés. A chaque anomalie doit être associé un piège sous contrôle de l'utilisateur, comme spécifié à l'article 10. La réponse par défaut à une anomalie doit être la continuation sans piège. Cette norme précise les résultats qui doivent découler à la fois de situations avec ou sans pièges. Dans certains cas le résultat est différent si un piège est permis.

Pour chaque type d'anomalie, l'utilisateur doit fournir un indicateur d'état qui doit être activé à chaque occurrence de l'anomalie correspondante lorsque aucun piège n'intervient. L'indicateur est désactivé seulement à la demande de l'utilisateur. L'utilisateur doit pouvoir tester et modifier les indicateurs d'état individuellement et devrait pouvoir sauver et restaurer les cinq indicateurs en bloc.

9.1 *Opération invalide*

Il y a deux types d'anomalies concernant «l'opération invalide». L'une, nommée «opérande invalide» se produit lorsqu'un opérande n'est pas valable pour l'opération à effectuer. L'autre, appelée «résultat invalide», se produit lorsque le résultat n'est pas valable pour la destination. Le résultat qui doit être rendu, dans les cas où l'une de ces deux anomalies apparaît en l'absence de piège, doit être un NaN sans piège (voir le paragraphe 7.2).

9.1.1 *Opérande invalide*

«Opération invalide» apparaît dans les cas suivants:

a) si l'un des opérandes est un NaN avec piège (paragraphe 7.2) et qu'aucun autre piège (dépendant de l'utilisateur) n'est prévu ou autorisé;

b) addition ou soustraction $\infty \pm \infty$ en mode projectif et soustraction en valeur absolue de quantités infinies comme $(+\infty) + (-\infty)$ en mode affine;

c) multiplication $0 \cdot \infty$;

d) division 0/0, ∞/∞, ou diviseur non normalisé et dividende fini et différent du zéro normalisé;

e) reste *x* REM *y* avec y égal à zéro ou non normalisé, ou *x* infini;

f) racine carrée si l'opérande est négatif, ∞ dans le mode projectif ou non normalisé;

8.1 *Normalizing mode*

Another mode of arithmetic should be provided which normalizes all denormalized values before performing arithmetic with them, and hence precludes the creation of new unnormalized operands (see note). This applies to all operations listed in this clause. Unnormalized operands shall not be affected by this mode.

Note. — In many computations the loss of significance due to denormalization is not consequential, and the invalid results (see Sub-clause 9.1.2) that arise in the warning mode are too pessimistic. Thus implementors are strongly encouraged to support the normalizing mode, except perhaps in pipelined array processors with an extended format for accumulation of intermediate sums. Use of the extended format, with its explicit leading significant bit, permits the calculation of intermediate products and quotients involving denormalized numbers without invalid result exceptions.

9. **Exceptions**

There are five types of exceptions that shall be detected. A trap under user control should be associated with each exception as specified in Clause 10. The default response to an exception shall be to proceed without a trap. This standard specifies results to be delivered in both trapping and non-trapping situations. In some cases the result is different if a trap is enabled.

For each type of exception the implementation shall provide a status flag which shall be set on any occurrence of the corresponding exception when no trap occurs. It shall be reset only at the user's request. The user shall be able to test and to alter the status flags individually, and should further be able to save and restore all five at one time.

9.1 *Invalid operation*

There are two kinds of invalid operation exception. One, called invalid operand, arises if an operand is invalid for the operation to be performed. The other, called invalid result, arises if the result is invalid for the destination. The result to be delivered, when either kind of invalid operation exception occurs without a trap, shall be a non-trapping NaN (see Sub-clause 7.2).

9.1.1 *Invalid operand*

Invalid operation shall be signalled in the following cases:

a) if any operand is a trapping NaN (see Sub-clause 7.2) and no other (implementation-dependent) trap is designated and enabled;

b) addition or subtraction $\infty \pm \infty$ in projective mode and magnitude subtraction of infinities like $(+\infty) + (-\infty)$ in affine mode;

c) multiplication $0 \cdot \infty$;

d) division 0/0, ∞/∞, or the divisor is not normalized and the dividend is finite and not normal zero;

e) remainder x REM y, where y is zero or not normalized, or x is infinite;

f) square root if the operand is less than zero, ∞ in the projective mode, or not normalized;

g) conversion d'un nombre binaire exprimé en virgule flottante dans un format entier ou décimal lorsqu'il y a dépassement de capacité, infini, ou NaN empêchant une représentation fidèle dans ce format, et que ce fait ne peut pas être signalé autrement;

h) comparaison avec les prédicats $<$, $\leqslant$, $\geqslant$, $>$ ou leurs négations, quand la relation est: «non ordonné».

Un résultat binaire en virgule flottante résultant d'une anomalie sans piège doit être un NaN sans piège.

9.1.2 *Résultat invalide*

Dans n'importe quelle opération, lorsque le résultat destiné à un format en simple ou double précision est non normalisé mais pas dénormalisé, le message «opération invalide» doit apparaître (voir note). Lorsqu'un «résultat invalide» coïncide avec un «dépassement de capacité», ou «inexact» ou «dépassement de capacité inférieure» avec piège, le message «résultat invalide» aura la priorité; mais des dépassements de capacité inférieure sans piège ne peuvent pas être des résultats invalides, car ils sont dénormalisés.

Note. — Ceci ne peut apparaître que dans certains cas avec les opérations suivantes:

a) Lorsqu'un nombre étendu non normalisé est converti dans un format de base.

b) Lorsque des opérations sur des nombres en simple précision dénormalisés ont des formats de résultat en double précision, excepté dans le mode qui normalise.

c) Lorsqu'un nombre dénormalisé est multiplié ou divisé et que le format du résultat est en simple ou double précision, excepté dans le mode qui normalise.

9.2 *Division par zéro*

Si le diviseur est le zéro normalisé et le dividende un nombre fini non nul, alors le message «division par zéro» doit être signalé. Le résultat par défaut sera l'∞ correctement signé (voir le paragraphe 7.3).

9.3 *Dépassement de capacité*

Si un résultat arrondi est fini et valable, mais que son exposant est trop grand pour être représenté dans le format flottant utilisé, le message «dépassement» doit apparaître, sauf dans les cas où le mode d'arrondi est vers $+\infty$ ou vers $-\infty$ et qu'il n'y a pas de piège pour les dépassements de capacité. Les dépassements de capacité doivent être arrondis ainsi:

a) L'arrondi vers $-\infty$ considère les dépassements de capacité normalisés positifs et les remplace par le nombre le plus positif correspondant au format, et considère les dépassements de capacité non normalisés positifs et les remplace par l'exposant le plus grand correspondant au format, sans changer la mantisse.

b) L'arrondi vers $+\infty$ considère les dépassements de capacité normalisés négatifs et les remplace par le nombre le plus négatif correspondant au format, et considère les dépassements de capacité non normalisés négatifs et les remplace par l'exposant le plus grand correspondant au format, sans changer la mantisse.

Dans ces deux cas une erreur «opération invalide» peut survenir, mais uniquement dans le cas de conversions du format étendu dans le format de base. Dans les autres cas de dépassement de capacité sans piège, on doit obtenir ∞ avec le signe approprié.

g) conversion of a binary floating-point number to an integer or decimal format when overflow, infinity, or NaN precludes a faithful representation in that format and this cannot otherwise be signalled; and

h) comparison via the predicates $<$, $\leqslant$, $\geqslant$, $>$, or their negations, when the relation is "unordered".

A binary floating-point result to be delivered, when an invalid operation exception arises without a trap, shall be a non-trapping NaN.

9.1.2 *Invalid result*

In any operation, when the result destined for a single or double format would be unnormalized but not denormalized, invalid operation shall be signalled (see note). When an invalid result exception coincides with an overflow or inexact or trapped underflow exception, the invalid result shall take precedence; but untrapped underflows cannot be invalid results because they are denormalized.

Note. — This can happen only in certain cases of the following operations:

a) When an unnormalized extended is converted to a basic format.

b) Except in the normalizing mode, when operations upon denormalized single format operands have double destinations.

c) Except in the normalizing mode, when a denormalized number is magnified by multiplication or division and the destination's format is single or double.

9.2 *Division by zero*

If the divisor is normal zero and the dividend is a finite nonzero number, then division by zero exception shall be signalled. The default result shall be a correctly signed ∞ (see Subclause 7.3).

9.3 *Overflow*

If a rounded result is finite and not an invalid result but its exponent is too large to represent in the target floating-point format, then overflow shall be signalled, unless the rounding mode is round toward $+\infty$ or round toward $-\infty$ and there is no trap on overflow. In the latter case overflows shall be rounded thus:

a) Round toward $-\infty$ carries normalized positive overflows to the format's largest number, and unnormalized positive overflows to the format's largest number's exponent without changing the mantissa.

b) Round toward $+\infty$ carries normalized negative overflows to the format's most negative number, and unnormalized negative overflows to the format's largest number's exponent without changing the mantissa.

In these two cases an invalid result may arise, but only in conversions from extended to basic formats. All other cases of overflow without a trap shall yield ∞ with the appropriate sign.