Numerische Mathematik I
WS 1993/94

IEEE Gleitkommaarithmetik

C. Cryer

25. Oktober 1996

Zusammenfassung

Die IEEE Arithmetik wurde in der IEEE P754 Spezifikation definiert. Diese Arithmetik ist jetzt auf vielen Rechnern - insbesondere den Sun-Rechnern in Münster - implementiert worden.

Inhaltsverzeichnis

1 Die Darstellung von Zahlen

Die Darstellung von Zahlen in einfacher und doppelter Genauigkeit wird wie folgt festgelegt:

- Einfache Genauigkeit

s	e	f
0	8	31

Die Felder eines 4 Byte Wortes sind:

1. Das Vorzeichen s (1 Bit)
2. Die Charakteristik e (8 Bits)
3. Die Mantisse f (23 Bits)

Der Wert von v ist:

1. Wenn $e = 255$ und $f \neq 0$, dann ist $v = NaN$
2. Wenn $e = 255$ und $f = 0$, dann ist $v = (-1)^s\infty$
3. Wenn $0 < e < 255$, dann ist $v := (-1)^s\, 2^{e-127}(1, f)$
4. Wenn $e = 0$ und $f \neq 0$, dann ist $v := (-1)^s\, 2^{-126}(0, f)$
5. Wenn $e = 0$ und $f = 0$, dann ist $v = (-1)^s 0$.

- Doppelgenauigkeit

s	e	f
0	11	63

Die Felder eines 8-Byte Wortes sind:

1. Das Vorzeichen s (1 Bit)
2. Die Charakteristik e (11 Bits)
3. Die Mantisse f (52 Bits)

Der Wert v ist:

1. $e = 2047$ und $f \neq 0 \Longrightarrow v = NaN$
2. $e = 2047$ und $f = 0 \Longrightarrow v = (-1)^s \infty$
3. $0 < e < 2047 \Longrightarrow v = (-1)^s\, 2^{e-1023}(1, f)$
4. $e = 0$ und $f \neq 0 \Longrightarrow v = (-1)^s\, 2^{-1022}(0, f)$
5. $e = 0$ und $f = 0 \Longrightarrow v = (-1)^s 0\,.$

Sowohl die einfache als auch die doppelte Genauigkeit läßt sich folgendermaßen veranschaulichen:

1. Die Verteilung der 32 bzw. 64 Bits zwischen der Charakteristik e und der Mantisse f entspricht einer Balanz zwischen dem Bedürfnis von größtmöglicher Genauigkeit (f) und größtmöglicher Reichweite (e).

2. Der Wert NaN (Not a Number) werden benutzt, wenn das Ergebnis nicht gültig ist, z. B. wenn man versucht 0.0/0.0 zu berechnen.

3. Die Zahlen $\pm\infty$ werden benutzt, wenn sehr große Zahlen erzeugt werden, z. B. wenn man 1.0/0.0 berechnet. Diese Zahlen werden wie üblich in der Mathematik behandelt. Es gilt z. B. daß $\infty + \infty = \infty$.

4. Zahlen für die $0 < e < 255$ gilt, sind normierte Gleitkommazahlen, wie sonst üblich. Da sie normiert sind, ist die führende Ziffer der Mantisse immer 1 und muß nicht gespeichert werden.

5. Die Zahlen mit $e = 0$ und $f \neq 0$ sind die nichtnormalisierten Gleitkommazahlen, die im Intervall $(0, \omega_+)$ gleichverteilt sind, wo ω_+ die kleinste normierte Gleitkommazahl ist.

6. Die positiven IEEE Maschinenzahlen können wie in Abb. 1 veranschaulicht werden.

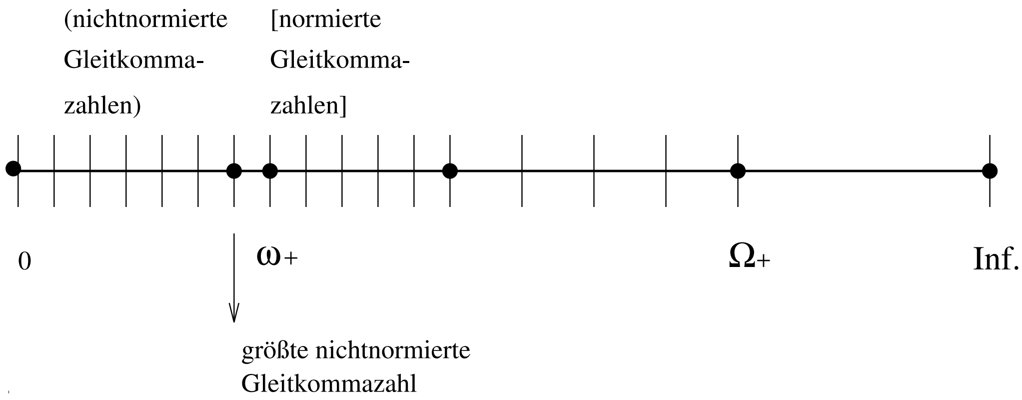

Abbildung 1: Die positiven IEEE Maschinenzahlen

2 Arithmetik

Sei A die Menge der IEEE Maschinenzahlen:

$A = \{0\} \cup \{\text{Normierte Gleitkommazahlen}\} \cup \{\text{Nichtnormierte Gleitkommazahlen}\} \cup \{\pm\infty\}$

Wenn $x, y \in A$ und $\Box$ eine arithmetische Operation $+-$ / oder $\times$ bezeichnet, ist es durchaus möglich, daß $x \Box y \notin A$. Es ist deshalb nötig, Zwischenergebnisse auf A abzubilden.

Definition: *Eine Abbildung rd:* $\mathbb{R} \to A$ *heißt Rundung falls:*

$$|rd(x) - x| \leq \min_{a \in A} |a - x|$$

Im IEEE Standard wird verlangt, daß das gelieferte Ergebnis $gl(x\Box y)$ einer Rechenoperation folgende Bedingungen erfüllt:

$$\left.\begin{array}{r} gl(x\Box y) \equiv x\Box^* y := rd(x\Box y), \quad \text{für} \Box \in \{+,-,/,\times\}, \\ gl(\sqrt{x}) := rd(\sqrt{x}), \end{array}\right\} \tag{1}$$

für $x, y \in A$.

In Fällen, wo diese Bedingungen nicht erfüllt werden können, entstehen Ausnahmen (Exceptions), siehe unter 3.

Die Tatsache, daß es gelungen ist, die Bedingungen 1 als Industriestandard durchzusetzen, ist ein großer Verdienst. Vorher wurden diese Bedingungen oft unnötigerweise verletzt.

3 Ausnahmen (Exceptions)

Folgende Ausnahmen sind möglich:

IEEE Exception	**Beispiel**	**Ergebnis**
Invalid (ungültig)	$\sqrt{-10.0}$ $10.0 >=$ NaN $0/0$ signalling NaN	NaN
Division by zero (Teilung durch Null)	x/0.0 mit $x \neq 0, \pm\infty$, NaN	$+\infty$ falls $x > 0$ $-\infty$, falls $x < 0$
Overflow (Exponentenüberlauf)	$\Omega_+ + \Omega_+$	$\pm\infty$
Underflow (Exponentenunterlauf)	$\omega_+/2.3$	Null oder Nichtnormierte Gleitkommazahl
Inexact (ungenau)	$x = 2.0$ $y = 3.0$ $z = x/y$	Das gerundete Ergebnis

- Es wird oben vorausgesetzt, daß eine normale Rundung benutzt wird.
- Im Standard wird zwischen "quiet" und "signalling" NaN Zahlen unterschieden. Dieser Unterschied ist für die jetzigen Zwecke unwichtig.

4 Normaler Umgang mit IEEE Arithmetik

Am Ende eines Programmes, kann überprüft werden, ob Ausnahmen aufgetreten sind, und zwar wie folgt:

- **C - Programme**

  ieee-retrospective ();

- **FORTRAN und Pascal**

  ieee-retrospective wird automatisch am Ende aufgerufen.

Falls Ausnahmen aufgetreten sind, können die wie folgt aufgespürt werden:

1. Kompiliere mit der -g Option, z. B. cc -g program.c.
2. Benutze dbxtool, z. B. dbxtool a.out.
3. In dbxtool den Befehl "catch fpe" im unteren Fenster eingeben.
4. In dbxtool "RUN" anklicken.

Das Programm wird dann ausgeführt und hält bei der ersten Ausnahme an.

Wichtig: Wenn eine Ausnahme auf diese Weise aufgespürt worden ist, dann ist das Ergebnis undefiniert.

5 Behandlung von Ausnahmen (Exceptions) auf Sun-Rechnern

Zwei Routinen ermöglichen die Bearbeitung von Ausnahmen:

- ieee-flags (action, mode, in, out)

  Diese Routine wird für zwei Zwecke eingesetzt:

  1. Um die Art der Rundung zu ändern, z. B. um Abschneiden oder Abrundung (engl.: chopping, truncation) zu benutzen.
  2. Um die Zähler ("flags") für eine Ausnahme anzuschauen oder zu löschen. Z. B. der Aufruf:

     ieee-flags ("clear", "exception", "division", & out)

     bewirkt, daß der Zähler für "division" Ausnahmen gleich Null gesetzt wird.

- ieee-handler (action, exception, handler)

  Diese Routine wird dafür benutzt, um

  1. die Ein/Ausschaltung von Interrupts beim Auftreten von Ausnahmen zu bewirken und
  2. den Namen der Routine anzugeben, zu der nach einem Interrupt gesprungen werden sollte.

Z. B. der Aufruf i=ieee-handler("set", "division", "cwc-trap") bewirkt, daß wenn die "division"-Ausnahme vorkommt, die Routine cwc-trap aufgerufen wird. Nach Beenden von cwc-trap läuft das Hauptprogramm weiter.

Bemerkung *Wenn ein Interrupt vorkommt, ist das gelieferte Ergebnis nicht definiert. Z. B. wenn das "division"-Interrupt ausgeschaltet ist, druckt das Programm*

$$x = 1.0;$$
$$y = 0.0;$$
$$z = x/y;$$

printf("z=1.0/0.0 = % d",z);

die Zeile

z=1.0/0.0 = Inf.

Falls das "Division" Interrupt eingeschaltet ist, ist der Wert von z nicht definiert.

Die Benutzung dieser Routinen wird durch das Programm im Anhang demonstriert. Weitere Dokumentationen befinden sich in XMan, Mosaic, Answerbook usw.

Literatur

[1] Numerical Computation Guide. Sun Microsystems, Oktober 1992.

[2] IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std. 754-1985.

A Beispiel der Benutzung der Routinen ieee-flags und ieee-handler

```
/*
Program um IEEE Arithmetik zu demonstrieren
Colin Cryer 15. November 1993

Basiert auf dem Program Seite 131-133
Numerical Computation Guide

*/



#include<math.h>
#include<signal.h>
#include <stdio.h>
#include <stdlib.h>


/*
 Das folgende Routine trap_all_fp_exc
 gibt an, welcher Fehler entstanden ist
 und wo dies passiert ist.

 trap_all_fp_exc wird eingeschaltet, wenn ein Fehler ensteht
 UND diese Art von Fehler getrappt werden sollte
*/

static void
trap_all_fp_exc( sig,  code, scp, addr)
int sig, code;
struct sigcontext *scp;
char *addr;
{
```

```
char *label="undefined";
switch (code){
case FPE_FLTINEX_TRAP:
  label="INEXACT";
  break;
case FPE_FLTDIV_TRAP:
  label="DIVISION";
  break;
case FPE_FLTUND_TRAP:
  label="UNDERFLOW";
  break;
case FPE_FLTOPERR_TRAP:
  label="INVALID";
  break;
case FPE_FLTOVF_TRAP:
  label="OVERFLOW";
  break;

/*
Bemerkung: Dieser Fall fuehrt zur Fehlermeldung:
"trapping.c",  undefined symbol: FPE_FLTNAN_TRAP
was ich nicht ganz verstehe da ALLE traps in der Datei
/usr/include/sys/signal.h
vorhanden sind

case FPE_FLTNAN_TRAP:
  label="notanumber";
  break;
*/

}

  printf("signal %d, sigfpe code %d:CRYER %s : EXCEPTION at hex address %x \n",
         sig,code, label, addr);
}
```

```
main()
{
double x,y,z;
int i,k;
char *out, *in;

for (k=0;k<=3;k++){

   if(k==0){
  /* case '0': disable trapping on all exceptions */
    printf("\n\nCASE 0: DISABLE ALL TRAPS  \n\n");
    if(ieee_handler("clear","all",NULL) !=0) {
      printf("Trap handler for all not cleared \n");
    }
  }


  if (k==1){
 /* case '1': Trap common exceptions */
    printf("\n\nCASE 1: TRAP COMMON EXCEPTIONS \n\n");
         if(ieee_handler("set","common",trap_all_fp_exc) !=0) {
  printf("IEEE trapping common not supported here \n");
}
  }


  if(k==2){
  /*case '2': Trap all exceptions */
    printf("\n\nCASE 2: TRAP ALL EXCEPTIONS \n\n");
    if(ieee_handler("set","all",trap_all_fp_exc) !=0) {
      printf("IEEE trapping all not supported here \n");
    }
  }

 if(k==3){
  /*case '3': Trap underflow and inexact exceptions */
    printf("\n\nCASE 3: TRAP UNDERFLOW AND INEXACT EXCEPTIONS \n\n");
```

```
    if(ieee_handler("set","underflow",trap_all_fp_exc) !=0) {
      printf("Trap handler for underflow not set \n");
    }
if(ieee_handler("set","inexact",trap_all_fp_exc) !=0) {
      printf("Trap handler for inexact not set \n");
    }
if(ieee_handler("set","overflow",SIGFPE_IGNORE) !=0) {
      printf("Trap handler for overflow not ignored \n");
    }
if(ieee_handler("set","invalid",SIGFPE_IGNORE) !=0) {
      printf("Trap handler for invalid not ignored \n");
    }
if(ieee_handler("set","division",SIGFPE_IGNORE) !=0) {
      printf("Trap handler for division not ignored \n");
    }
  }


/*raise invalid*/
i=ieee_flags("clear","exception","invalid",&out);
x=signaling_nan(0);
printf("signaling nan = %20e\n",x);
y=2.5;
z=x*y;
printf("2.5 * NaN = %20e\n\n",z);

/*add infinity*/
i=ieee_flags("clear","exception","invalid",&out);
x=infinity(0);
printf("infinity = %20e\n",x);
z=x+x;
printf("infinity+ infinity = %20e\n\n",z);

/*subtract infinity*/
i=ieee_flags("clear","exception","invalid",&out);
x=infinity(0);
z=x-x;
```

```
printf("infinity- infinity = %20e\n\n",z);

/*raise division */
i=ieee_flags("clear","exception","division",&out);
x=1.0;
y=0.0;
z=x/y;
printf("1.0/0.0 = %20e\n\n",z);

/*raise overflow */
i=ieee_flags("clear","exception","overflow",&out);
x=-max_normal();
printf("-max normal = %20e\n",x);
y=-1.0e294;
z=x+y;
printf("-maximum normal -something = %20e\n\n",z);

/*raise overflow */
i=ieee_flags("clear","exception","overflow",&out);
x=max_normal();
printf("max normal = %20e\n",x);
y=1.0e294;
z=x+y;
printf("maximum normal +something = %20e\n\n",z);

/*raise underflow */
i=ieee_flags("clear","exception","underflow",&out);
x=min_normal();
printf("minimum normal = %20e\n",x);
y=4.0001;
z=x/y;
printf("min normal /4.0001 = %20e\n\n",z);

/*raise underflow */
i=ieee_flags("clear","exception","underflow",&out);
x=min_normal();
printf("minimum normal = %20e\n",x);
```

```
y=4.0;
z=x/y;
printf("min normal /4.0 = %20e\n\n",z);

/*raise underflow */
i=ieee_flags("clear","exception","underflow",&out);
x=min_normal();
printf("minimum normal = %20e\n",x);
y=3.9999;
z=x/y;
printf("min normal /3.9999 = %20e\n\n",z);


/*raise underflow */
i=ieee_flags("clear","exception","underflow",&out);
x=min_subnormal();
printf("minimum subnormal = %20e\n",x);
z=x*x;
printf("min subnormal * min subnormal = %20e\n\n",z);

/*compare NaN*/
i=ieee_flags("clear","exception","invalid",&out);
x=signaling_nan(0);
if (x>1.0) printf("NaN is greater than 1\n\n");
   else printf("NaN not comparable to 1\n\n");



/*raise inexact*/
i=ieee_flags("clear","exception","inexact",&out);
x=2.0;
y=3.0;
z=x/y;
printf("2.0/3.0 = %20e\n\n",z);
```

```
/*Find out if any outstanding exceptions */

ieee_retrospective_();

}

/* exit gracefully */
exit(0);
}
```