3.2 Die Objekte der Maschine

Zur Vorlesung Rechenanlagen

SS 2019

3.2.1 Wortbreite einer Maschine

Wir wissen, dass wir alles auf Bitstrings zurückführen müssen, daher müssen die Objekte durch solche kodiert werden.

Es wäre ungeschickt, Speicherplätze mit verschieden langen Bitstrings vorzusehen, da man dann in einen Speicherplatz nur Objekte einer speziellen Sorte unterbringen könnte.

Objekte und Operationen nur auf der Basis einzelner Bits bereitzustellen und alles andere darauf sequentiell durch Befehlsfolgen zu bearbeiten wäre aber auch unsinnig.

Die **Wortbreite** *n* einer Maschine legt die maximale Länge von Objekten als Bitstrings fest, die in einem Register oder Speicherplatz gehalten werden können.

Maschinenworte

Sei *n* die Wortbreite einer Maschine. Dann nennen wir einen Bitstring der Länge *n*

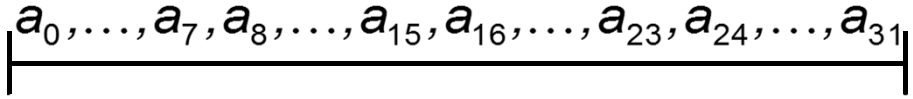
$$a = a_0, \ldots, a_{n-1}$$

ein Maschinenwort. Eine heute übliche Breite ist *n = 64* oder *n = 32*.

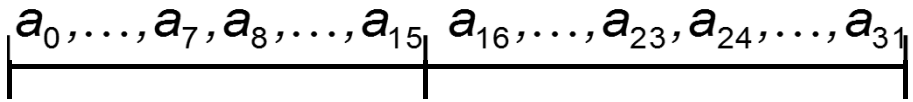
Meist unterteilt man Maschinenworte noch weiter nach 2er Potenzen in Halbworte, Viertelworte, ...

Beispiel: 32 Bit Wortbreite

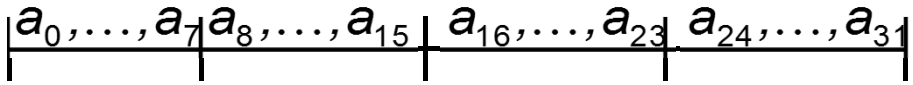
long: 32 - Bit Wort



short: 16 - Bit Wort



byte: 8 - Bit Wort



Maschinenworte ff

Vorsicht: Man spricht oft von 32-, 64-Bit Maschinen. Dies hat nicht notwendig etwas mit unserer Definition der Wortbreite zu tun. Man meint damit oft die maximale Länge, die Adressen von Speicherplätzen als Bitstrings haben können.

Die Objekte unserer Maschine definieren wir nun, indem wir Maschinenworten oder Teilworten davon eine geeignete Interpretation verleihen:

3.2.2 Ganze Zahlen

Die wohl wichtigsten Objekte in Rechnern sind Zahlen. Die einfachste Art und Weise, Bitstrings als Zahlen aufzufassen, ist ihre Interpretation als nichtnegative ganze Zahl:

Definition

Wir nennen $u_n : \mathbb{B}^n \mapsto \mathbb{N}_0$

mit $u_n(a_0,\ldots,a_{n-1}) := 2^{n-1}\sum_{i=0}^{n-1} a_i 2^{-i}$

Darstellung als vorzeichenlose (unsigned) ganze Zahl.

Das Bit a_0 hat das höchste Gewicht 2^{n-1} . Wir nennen es auch das **signifikanteste Bit**.

Wenn klar ist, dass n die Wortbreite ist, schreiben wir auch einfach u statt u_n.

Unsigned Numbers

Der Zahlenbereich einer vorzeichenlosen ganzen Zahl ist

$$u_n(\mathbf{B}^n) = [0 : 2^n - 1]$$

Größere Zahlen kann man nicht direkt darstellen. Will man mit größeren Zahlen arbeiten, dann muss dies explizit durch Programmierung realisiert werden.

Anmerkungen:

Rechnerarithmetiker benutzen oft auch eine gespiegelte Notation der Bitstrings um Zahlen zu definieren. In diesem Fall entspricht das Gewicht eines Bits *i* direkt 2^i.

Den Zahlenbereich erhält man durch

$$u_n(0,\ldots,0) = 0, \text{ sowie } u_n(1,\ldots,1) = 2^{n-1}\sum_{i=0}^{n-1} 1 \cdot 2^{-i} = 2^{n-1}(2 - 2^{-(n-1)}) = 2^n - 1$$

Signed Numbers

Definition

Wir nennen $bv_n : \mathbb{B}^n \mapsto \mathbb{Z}$

mit $$bv_n(a_0,\ldots,a_{n-1}) := (-1)^{a_0} \cdot 2^{n-1} \cdot \sum_{i=1}^{n-1} a_i 2^{-i}$$

Darstellung durch Betrag und Vorzeichen.

Der Zahlenbereich bei dieser Darstellung ist

$$bv_n(\mathbf{B}^n) = [bv_n(1\cdots1) : bv_n(01\cdots1)]$$

$$(-1)^1 \cdot 2^{n-1} \cdot \sum_{i=1}^{n-1} 1 \cdot 2^{-i}$$

$$= -2^{n-1} \cdot (1 - 2^{-(n-1)})$$

$$= \boxed{-(2^{n-1} - 1)}$$

$$(-1)^0 \cdot 2^{n-1} \cdot \sum_{i=1}^{n-1} 1 \cdot 2^{-i}$$

$$= 2^{n-1} \cdot (1 - 2^{-(n-1)})$$

$$= \boxed{2^{n-1} - 1}$$

Signed Numbers ff

Definition

Wir nennen $b_{2,n} : \boldsymbol{B}^n \mapsto \boldsymbol{Z}$

mit $$b_{2,n}(a_0,\ldots,a_{n-1}) := -2^{n-1}a_0 + 2^{n-1} \cdot \sum_{i=1}^{n-1} a_i 2^{-i}$$

Darstellung im **Zweierkomplement**.

Der Zahlenbereich bei dieser Darstellung ist

$$b_{2,n}(\mathbf{B}^n) = [\, b_{2,n}(10\cdots0) : b_{2,n}(01\cdots1)]$$

$$-2^{n-1}\cdot 1 + 2^{n-1} \cdot \sum_{i=1}^{n-1} 0 \cdot 2^{-i}$$

$$= \boxed{-2^{n-1}}$$

$$-2^{n-1}\cdot 0 + 2^{n-1} \cdot \sum_{i=1}^{n-1} 1 \cdot 2^{-i}$$

$$= 2^{n-1}(1-2^{-(n-1)}) = \boxed{2^{n-1}-1}$$

Signed Numbers ff

Definition

Wir nennen $b_{1,n} : \boldsymbol{B}^n \mapsto \boldsymbol{Z}$

mit $b_{1,n}(a_0,\ldots,a_{n-1}) := -(2^{n-1}-1)a_0 + 2^{n-1} \cdot \sum_{i=1}^{n-1} a_i 2^{-i}$

Darstellung im **Einerkomplement**.

Der Zahlenbereich bei dieser Darstellung ist

$$b_{1,n}(\mathbf{B}^n) = [\,b_{1,n}(10\cdots0) : b_{1,n}(01\cdots1)\,]$$

$$-(2^{n-1}-1)\cdot 1 + 2^{n-1} \cdot \sum_{i=1}^{n-1} 0 \cdot 2^{-i} = \boxed{-(2^{n-1}-1)}$$

$$-(2^{n-1}-1)\cdot 0 + 2^{n-1} \cdot \sum_{i=1}^{n-1} 1 \cdot 2^{-i} = 2^{n-1}(1-2^{-(n-1)}) = \boxed{2^{n-1}-1}$$

Zahlendarstellungen -- allgemein

Es gibt eine Fülle von weiteren Zahlendarstellungen (BCD, redundante Zahlendarstellungen, Residuendarstellungen, ...). Im allgemeinen ist eine Zahlendarstellung eine Funktion

$$d_n : \mathbb{B}^n \rightarrow \mathbb{Z} \quad (\mathbb{Q}, \mathbb{R}, \ldots)$$

die Bitstrings als Zahlen interpretiert. Betrachten wir nun Darstellungen ganzer Zahlen, so scheinen folgende Eigenschaften wünschenswert:

1. Der Zahlenbereich $d_n(\mathbb{B}^n)$ sollte ein Intervall sein.
2. $0 \in d_n(\mathbb{B}^n)$
3. Symmetrie: $d_n(\mathbb{B}^n) = [-a:a]$
4. d_n sollte injektiv und total sein.

Zahlendarstellungen ff

Die von uns eingeführten Darstellungen erfüllen alle die Eigenschaft 1 und 2.

Symmetrie gilt nur für *bv* und b_1

Injektiv sind nur *u* und b_2, denn

$$
\begin{aligned}
b_{1,n}(0\cdots0) &= 0 \\
&= -(2^{n-1}-1)+(2^{n-1}-1) \\
&= -(2^{n-1}-1)\cdot 1+2^{n-1}(1-2^{-(n-1)}) \\
&= -(2^{n-1}-1)\cdot 1+2^{n-1}\cdot\sum_{i=1}^{n-1}1\cdot 2^{-i} \\
&= b_{1,n}(1\cdots1)
\end{aligned}
$$

„negative 0“

$$bv_n(0\cdots0) = 0 = bv_n(10\cdots0)$$

3.2.3 Gleitkommazahlen

Reelle Zahlen kann man wegen der Überabzählbarkeit nicht mehr vollständig im Rechner darstellen. Andererseits möchte man mit sehr kleinen (Atomphysik) oder riesig großen (Astronomie) Zahlen rechnen, wobei meist nur eine Näherung der tatsächlichen Zahl genügt.

Bei Handrechnungen kennen wir die Darstellung durch eine **Mantisse** und einen **Exponenten**, der die Größe anpaßt:

$$a \cdot 10^{e} \quad \text{bzw.} \quad a \cdot 10^{-e}$$

Geht man davon aus, dass man damit eine reelle Zahl $a' \cdot 10^{e}$ nur durch wenige Stellen in a annähert mit $a \cdot 10^{e}$, ist dies nicht ungeschickt, weil der Exponent beim relativen Fehler verschwindet:

$$\left| \frac{a \cdot 10^{e} - a' \cdot 10^{e}}{a' \cdot 10^{e}} \right| = \left| \frac{a - a'}{a'} \right|$$

Gleitkommazahlen ff

Der relative Fehler hängt also nur von der Zahl der Stellen ab, die man in *a* kennt, wenn *a* die bestmögliche Näherung für *a‘* mit dieser Zahl von Stellen ist.

Diese Technik macht man sich auch in Rechnern zunutze, indem man ein Raster über die reellen Zahlen legt, das durch Mantissen und Exponenten fester Länge aufgespannt wird. Es gibt einige Freiheitsgrade zur Implementierung von Operationen und zum Runden, sowie zur Festlegung von Mantissen- und Exponentendarstellung und -länge, die wir in dieser Vorlesung nicht erarbeiten wollen.

Um Probleme bei Portierungen von Programmen zu mildern legt der **IEEE Standard 754** diese Freiheitsgrade heute weitgehend fest:

Gleitkommazahlen

Allgemein ist eine Gleitkommadarstellung eine Abbildung

$$fp: \boldsymbol{B}^{m+e} \rightarrow \boldsymbol{R}$$

folgender Form:

$$fp(a,b) := d_m(a) \cdot 2^{d_e(b)}$$

wobei d_m, d_e Zahlendarstellungen für die Mantisse und den Exponenten sind. Folgende Freiheitsgrade sind festzulegen:

- m - die Mantissenbreite
- e - die Exponentenbreite
- d_m - die Mantissendarstellung, und
- d_e - die Exponentendarstellung

Dazu betrachten wir zunächst ein paar Zielsetzungen:

Gleitkommazahlen -- Zielsetzungen

Folgende Zielsetzungen sind erstrebenswert. Bei der Festlegung der Parameter und des Formates sollte man darauf Rücksicht nehmen:

- Vergleiche =, >, <, ≤, ≥,=0, >0, <0 sollten mit Integervergleichern möglich sein!
- Möglichst wenig Mehrdeutigkeiten, d.h. *fp* „weitgehend injektiv“
- Rundungsfehler kontrollierbar mit der Mantissenlänge (normalisierte Zahlen)
- Unterstütze Rechnen mit denormalisierten Zahlen
- Balancierter Zahlenbereich, d.h. $\#fp^{-1}([-1,1]) \approx \#fp^{-1}(\mathbb{R} \setminus [-1,1])$
- Unterstütze spezielle Symbole zur Fehlerbehandlung

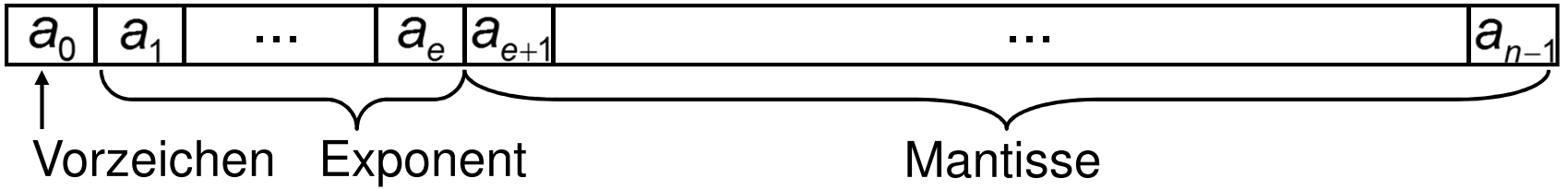
Gleitkommazahlen -- Konsequenzen

Aus diesen Zielsetzungen leitet sich der Standard nahezu zwangsläufig ab:

(1) Integer Vergleich nutzbar

Damit integer Vergleicher direkt nutzbar sind, muß a_0 das Vorzeichenbit sein.

Da ferner grosse Exponenten grosse Zahlen, und kleine Exponenten kleine Zahlen darstellen, muss dem Vorzeichen zuerst der Exponent und dann die Mantisse folgen.



Die Folge *0…0* sollte ferner der kleinste Exponent sein, *1…1* der größte Exponent $\Rightarrow d_e = unsigned - bias$

$$d_e(a_1 \cdots a_e) := \left(2^{e-1} \cdot \sum_{i=0}^{e-1} a_{i+1} \cdot 2^{-i} \right) - bias$$

Gleitkommazahlen -- Konsequenzen ff

(2) Wenig Mehrdeutigkeiten

Man kann natürlich leicht durch Multiplikation mit *2* oder *1/2* Exponent gegen Mantisse ausspielen, wenn man für die Mantisse eine Zahlendarstellung nutzt, bei der neben *x* auch *2x* oder *1/2 x* zulässig ist.

Ausweg: „Normalisierte Zahlen“

Wir wählen für die Mantisse eine Darstellung im Bereich *[1,2)* . Damit sind solche Trade offs ausgeschlossen.

$$d_m(a_{e+1}\cdots a_{e+m-1}) := 1+\sum_{i=1}^{m-1} a_{e+i}\cdot 2^{-i}$$

Man kann also die Mantisse auffassen als eine Binärzahl der Form *1.m…m*, bei der stets die erste Stelle *=1* ist.
(implizites führendes Bit)

Gleitkommazahlen -- Konsequenzen ff

Damit haben wir bisher schon folgende Festlegung getroffen:

Eine **normalisierte Gleitkommazahl** ist ein Bitstring der Form

$$sb_0 \ldots b_{e-1} a_0 \ldots a_{m-2}$$

wobei *z* der benutzte Bias des Exponenten ist:

$$fp(sb_0 \cdots b_{e-1} a_0 \cdots a_{m-2}) := (-1)^s \cdot \frac{(2^{m-1} + u(a))}{2^{m-1}} \cdot 2^{u(b)-z}$$

$$fp(s,b,a) := (-1)^s \cdot (2^{m-1} + u(a)) \cdot 2^{u(b)-(z+m-1)}$$

Gleitkommazahlen -- Konsequenzen ff

(3) Rundungsfehler

Die Menge der normalisierten Gleitkommazahlen, $fp(\mathbf{B}^{m+e})$, ist total geordnet und sei

$$fp(\mathbf{B}^{m+e}) = \{ y_0 < \cdots < y_{2^{m+e}-1} \} \subseteq \mathbf{R}$$

Eine **Rundung** $\rho: \mathbf{R} \rightarrow fp(\mathbf{B}^{m+e})$ ist eine Abbildung, die jedem $x \in \mathbf{R}$ eine der benachbarten Gleitkommazahlen y_i, y_{i+1} aus $fp(\mathbf{B}^{m+e})$ zuordnet, mit $x \in [y_i, y_{i+1}]$

Der relative Rundungsfehler für x ist damit stets beschränkt durch

$$\left| \frac{x - \rho(x)}{x} \right| \leq \left| \frac{y_{i+1} - y_i}{y_i} \right|$$

Gleitkommazahlen -- Konsequenzen ff

Rundungsfehler ff

Wir nehmen an, dass beide grösser gleich 0 sind, die anderen Fälle sind analog (den Fall =0 erst mal ausgenommen)

Haben y_i, y_{i+1} gleichen Exponenten, d.h. $y_i = fp(0,e,a)$, $y_{i+1}=fp(0,e,b)$ dann ist $u(b) = u(a) + 1$, und damit

$$y_{i+1} - y_i$$
$$= (2^{m-1} + u(b)) \cdot 2^{u(e)-(z+m-1)} - (2^{m-1} + u(a)) \cdot 2^{u(e)-(z+m-1)}$$
$$= 2^{u(e)-(z+m-1)}$$

Also ist dann

$$\left| \frac{y_{i+1} - y_i}{y_i} \right| = \frac{2^{u(e)-(z+m-1)}}{\left(2^{m-1} + u(a)\right) \cdot 2^{u(e)-(z+m-1)}} = \frac{1}{\left(2^{m-1} + u(a)\right)} \leq \frac{1}{2^{m-1}}$$

Gleitkommazahlen -- Konsequenzen ff

Rundungsfehler ff

Haben y_i, y_{i+1} verschiedene Exponenten, dann liegt, da y_i, y_{i+1} benachbart sind folgende Situation vor:

$$y_i = fp(0,e,1\ldots1),\ y_{i+1}=fp(0,e',0\ldots0)$$

mit $u(e') = u(e) + 1$,
und damit

$$y_{i+1} - y_i$$
$$= 2^{m-1} \cdot 2^{u(e)+1-(z+m-1)} - (2^m - 1) \cdot 2^{u(e)-(z+m-1)}$$
$$= 2^{u(e)-(z+m-1)}$$

Also ist ebenfalls wieder

$$\left|\frac{y_{i+1} - y_i}{y_i}\right| = \frac{2^{u(e)-(z+m-1)}}{\left(2^m - 1\right) \cdot 2^{u(e)-(z+m-1)}} = \frac{1}{2^m - 1} \leq \frac{1}{2^{m-1}} \quad \text{-- } m > 0$$

Gleitkommazahlen -- Konsequenzen ff

(4) Balance des Zahlenbereichs

Die Zahlen *1, -1* haben die Darstellung

$$(-1)^s(2^{m-1}+0)\cdot 2^{u(e)-(z+m-1)}$$

Also muß für den Exponenten gelten

$$m\text{-}1 + u(e) - (z+m-1) = 0 \;\Leftrightarrow\; u(e) = z$$

Da sich *u(e)* im Bereich $[0:2^e-1]$ bewegt, ist - will man genauso viele negative wie positive Zahlen haben - *u(e)* in der Mitte zu wählen.
Da wir die Exponenten *0* und 2^e-1 für die Darstellung der *0*, von denormalen Zahlen und speziellen Symbolen reservieren wollen, ist die “Mitte” bei

$$u(e) \in \{2^{e-1}-1,\ 2^{e-1}\}.$$

Wir entscheiden uns für $2^{e-1}-1$ (mehr große Zahlen) und erhalten so für den Bias

$$\boxed{z = 2^{e-1}-1}$$

Gleitkommazahlen -- Konsequenzen ff

(5) Denormale Zahlen und spezielle Symbole

Man möchte im Falle eines Exponentenunterlaufs auch noch mit denormalen Zahlen weiterrechnen können. Da wir die *0* ohnehin nicht als normalisierte Zahl interpretieren können, bietet es sich an, für den Exponenten *0…0* die Interpretation zu ändern, und auf die implizite 1 bei der Mantisse zu verzichten.

Vorsicht: Rundungsfehler können enorm ansteigen!

Andererseits gibt es Operationen, die illegal sind, oder aus dem Zahlenbereich herausführen. (Quadratwurzel mit negativem Argument, Division durch *0* …) Wir möchten dies durch spezielle Symbole anzeigen. Dazu reservieren wir neben *0…0* auch den Exponenten *1…1* für spezielle Symbole.
Damit gilt bei normalisierten Zahlen für die Exponenten folgende Schranke $e_{min} \leq x \leq e_{max}$
mit $e_{min} = 1 - z \qquad e_{max} = 2^e - 2 - z$

IEEE Standard für Symbole

Folgende Tabelle legt die Symbole für Gleitkommazahlen nach dem IEEE Standard 754 fest:

Exponent	Mantisse	Interpretation
0…0	*0…0*	± ***0***
0…0	*≠ 0…0*	**denormalisierte Zahl**
∈ [e_{min},e_{max}]	beliebig	**normalisierte Zahl**
1…1	*0…0*	± ∞
1…1	*≠ 0…0*	**NaN (not a number)**

NaN wird bei illegalen Operationen erzeugt. Bei Verlassen der Zahlenbereichs hingegen ± ∞.

Rundungsarten

Die Menge der Gleitkommazahlen bildet ein endliches Raster auf $\boldsymbol{R}$, das nicht abgeschlossen ist unter den arithmetischen Operationen. Demnach müssen wir, wenn wir Arithmetik auf Gleitkommazahlen betreiben, die erhaltenen Zwischenergebnisse auf das Gleitkommaraster abbilden.

Eine **Rundung** $\rho: \boldsymbol{R} \rightarrow fp(\boldsymbol{B}^{m+e})$ ist eine Abbildung, die jedem $x \in \boldsymbol{R}$ eine der benachbarten Gleitkommazahlen y_i, y_{i+1} aus $fp(\boldsymbol{B}^{m+e})$ zuordnet.

Sie muß stets folgende Eigenschaften haben:

- $x \in]y_i, y_{i+1}[$, dann ist $\rho(x) \in \{y_i, y_{i+1}\}$
- $x \in fp(\boldsymbol{B}^{m+e})$, dann ist $\rho(x) = x$

Der IEEE Standard 754 schreibt folgende Rundungsarten vor:

Rundungsarten nach IEEE Standard 754

i. Zur 0 runden (Round to Zero):

Für $x \in]y_i, y_{i+1}[$ ist

$$rz(x) := \begin{cases} y_i & y_i \geq 0 \\ y_{i+1} & y_i < 0 \end{cases}$$

ii. Nach $+\infty$ ($-\infty$) runden (Δ (∇)):

Für $x \in]y_i, y_{i+1}[$ ist $\quad \Delta(x) := y_{i+1}$ ($\nabla(x) := y_i$)

iii. Zur nächsten Zahl runden (Round to Nearest Even):

Diese Rundungsart ist zugleich „default“ Rundungsart.

Für $x \in]y_i, y_{i+1}[$ ist

$$rne(x) := \begin{cases} y_i & x < y_i + \frac{y_{i+1}-y_i}{2} \\ y_{i+1} & x > y_i + \frac{y_{i+1}-y_i}{2} \\ (1-lsb(y_i))y_i + (1-lsb(y_{i+1}))y_{i+1} & x = y_i + \frac{y_{i+1}-y_i}{2} \end{cases}$$

wobei im Falle, dass *x* die Intervallmitte bildet, nach dem niedrigsten signifikanten Bit der Mantisse (*lsb*) entschieden wird. Man rundet zu der Zahl *y* mit *lsb(y) = 0*. (nearest even).

Charakteristika von Gleitkommazahlen

Aus den bisherigen Betrachtungen ergeben sich für konkrete Wortbreiten folgende Charakteristika:

Sei n die Wortbreite
m die Mantissenbreite (inclusive Vorzeichenbit!)
e die Exponentenbreite

Dann gilt $n = m + e$.

Genauigkeit: Wir haben gesehen, dass wir bei normalisierten Zahlen einen relativen Fehler von $\leq 2^{-(m-1)}$ garantieren können.

Bias: der Bias ergab sich direkt aus der Exponentenbreite e als

$$bias = 2^{e-1} - 1$$

e_{min}, e_{max}: Es war $e_{min} = -bias + 1 = -2^{e-1} + 2$

$$e_{max} = 2^e - 2 - bias = 2^e - 2 - 2^{e-1} + 1 = 2^{e-1} - 1 = bias$$

Man muss nun nur noch, bei fester Wortbreite m gegen e aushandeln. Der IEEE Standard macht dazu folgende Vorschläge:

IEEE Standard für Mantisse, Exponent

Folgende Tabelle legt die Mantissen, Exponenten trade offs für Gleitkommazahlen nach dem IEEE Standard 754 fest:

Format	single	single extd.	double	double extd.
Mantisse	*24*	*≥ 32*	*53*	*≥ 64*
$e_{max} = bias$	*127*	*≥ 1023*	*1023*	*≥ 16383*
e_{min}	*-126*	*≤ -1022*	*-1022*	*≤ -16382*
Wortbreite	*32*	*≥ 43*	*64*	*≥ 79*

Spezielle Symbole und Fehlerbehandlung

Ziele:

Ein Gleitkomma-Rechenwerk sollte

- Fehlerbedingungen spezifizieren
- ggf. mit speziellen Symbolen weiterrechnen

Beispiele:

Bei einer Division einer Zahl $\neq \pm \infty$ durch *0* sollte der Wert $\pm \infty$ zurückgegeben werden.

Hingegen bei der Quadratwurzel aus einer negativen Zahl das Symbol *NaN*.

Es macht durchaus Sinn, falls der Benutzer keine Traps gelegt hat, mit $\pm \infty$ Arithmetik zu betreiben:

$$\infty + \infty = \infty \; ; \; \infty * \infty = \infty \; ; \; -\infty + -\infty = -\infty \; ; \; 1 / \infty = 0 \; ; \ldots$$

aber: $\infty - \infty =$ *NaN*; $\infty / \infty =$ *NaN*; …

Alle Operatoren und Funktionen (sin,cos,…) sollten strikt auf *NaN* sein, d.h. ist ein Argument *NaN* liefern sie *NaN* als Ergebnis.

Flags für Benutzer-Traps

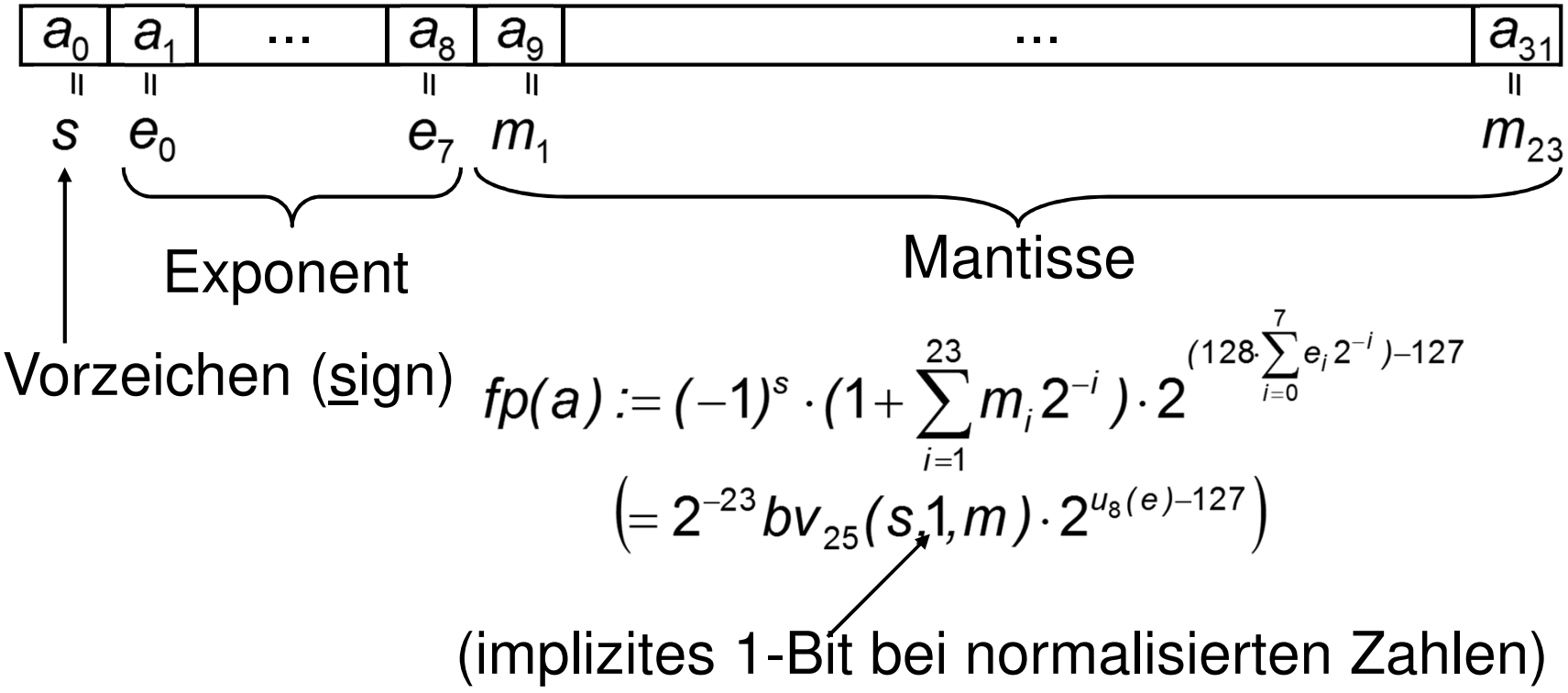
Folgende Flags sollen nach IEEE Standard 754 von einem Gleitkomma-Rechenwerk geliefert werden, damit man ggf. über benutzerspezifische Masken in gewissen Situationen Traps (Ausnahmebehandlungen) auslösen kann.

- underflow -- Ergebnis wurde denormal
- overflow -- Ergebnis $\pm \infty$
- divide by zero
- inexakt -- Ergebnis wurde gerundet
- invalid -- Ergebnis NaN

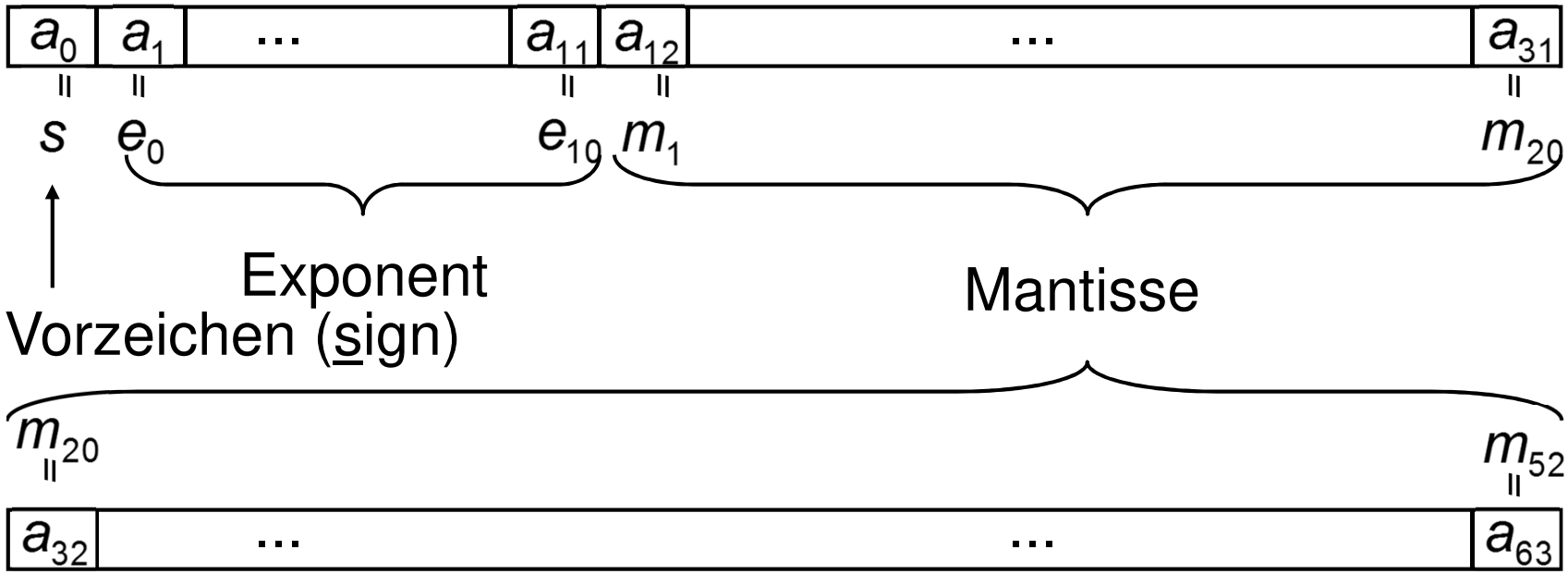
Die Flags sollen bei Vorliegen entsprechender Bedingungen gesetzt werden, und auch bei folgenden Operationen gesetzt bleiben, bis sie explizit zurückgesetzt werden **sticky flags**.

Zusammenfassung: Float

Single precision floating point numbers



Zusammenfassung : double



$$fp(a) := (-1)^s \cdot (1 + \sum_{i=1}^{52} m_i 2^{-i}) \cdot 2^{(1024 \cdot \sum_{i=0}^{11} e_i 2^{-i}) - 1023}$$

$$\left(= 2^{-53} bv_{54}(s,1,m) \cdot 2^{u_{11}(e)-1023} \right)$$

3.2.4 Zeichen

Eine Zeichendarstellung ist meist gegeben durch eine injektive Abbildung $c : A \mapsto \mathbb{B}^k$

die Zeichen in einem Zeichensatz *A* einen Code zuordnet. Diese Abbildung kann man in sog. Codetabellen nachlesen. Sie legt typischerweise die Codes für Buchstaben und Sonderzeichen (EOT,LF,CR,…) fest. Meist ist diese Abbildung nicht surjektiv, d.h. bestimmte Bitvektoren haben keine Bedeutung als Zeichen. Diese Redundanz kann anwendungsspezifisch für weitere Sonderzeichen (Grafik, Gerätesteuerung,…) oder zur Fehlertoleranz (Parity Bit) benutzt werden.

Gebräuchliche Codes sind z.B. **ASCII** oder **EBCDIC**. In unserem Kulturraum genügt $k = 8$ um alle notwendigen Zeichen zu kodieren (1 Byte).

Zeichen

Folgende Eigenschaften für Codes sind sinnvoll (gelten aber nicht immer!)

$$u_k(c(0)) < \ldots < u_k(c(9)) < u_k(c(A)) \ldots < u_k(c(Z)) < u_k(c(a)) \ldots < u_k(c(z))$$

Damit entspricht die Interpretation der Codes als Zahl der gewohnten lexikographischen Ordnung.

Nützlich ist es auch, wenn die Zeichen dicht liegen, weil man dann Konvertierungen leichter implementieren kann, d.h. die Codes bilden Intervalle:

$$u_k(c(\{0,\ldots,9\})) = [u_k(c(0)) : u_k(c(9))]$$

Dann erhält man den Code für die Ziffer *i* z.B. durch

$$c(i) = u_k^{-1}(u_k(c(0)) + i)$$

3.2.5 Der Krieg der Endians

Wir benötigen viel weniger Platz, um Zeichen zu speichern, als um Zahlen zu speichern. Bei Zahlen bietet es sich ferner an, verschiedene Genauigkeiten anzubieten, sowohl bei reellen Zahlen (double, float) als auch bei ganzen Zahlen (Multimedia).

Problem: Unterstütze den Zugriff auf verschieden große Teilworte.

Die kleinsten adressierbaren Teilworte sind sinnvoller-weise Bytes, da sie für Zeichensätze benutzt werden. Andere Teilworte werden dann als Folgen von Bytes betrachtet. Ein *k*-Byte Teilwort mit Adresse *a* besteht dann aus den Bytes mit Adressen *a+0,a+1,…,a+k-1*

In welcher Reihenfolge legt man diese Bytes in einem Teilwort ab?

Der Krieg der Endians ff

Es gibt zwei ebenso einfache wie sinnvolle Antworten darauf:

GrößereTeilworte stellen Zahlen dar, das heißt sie haben die Bits von signifikant (*0*) nach weniger signifikant (*n-1*) geordnet. Man kann nun die Bytes mit kleinerer Adresse auf

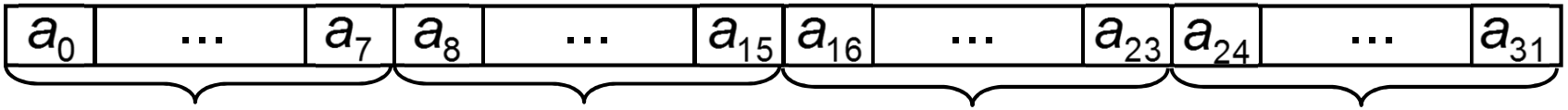
- die weniger signifikanten Positionen des Teilworts **(little endian)**, oder
- die mehr signifikanten Positionen des Teilwortes **(big endian)**

legen.

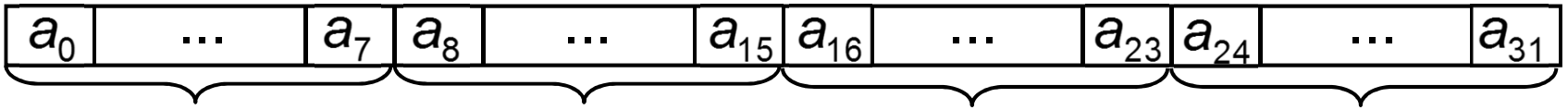
Der Krieg der Endians ff

Beispiel: 32-bit Maschine

Little Endian (I80x86, VAX)



Big Endian (IBM370, Motorola 680x0, Hitachi H8)



Solange man Daten zwischen Maschinen gleichen Typs austauscht, sind beide Prinzipien gleich gut. Probleme treten bei wortorientierten Datensätzen zwischen Maschinen unterschiedlichen Typs auf.

Alignment

Bei byteorientierten Speichern beziehen sich die Adressen von Speicherinhalten stets auf Bytes. Beim Zugriff auf größere Teilworte (Halbworte (16b), Worte (32b), Doppelworte (64b)) hat man nun zwei Möglichkeiten:

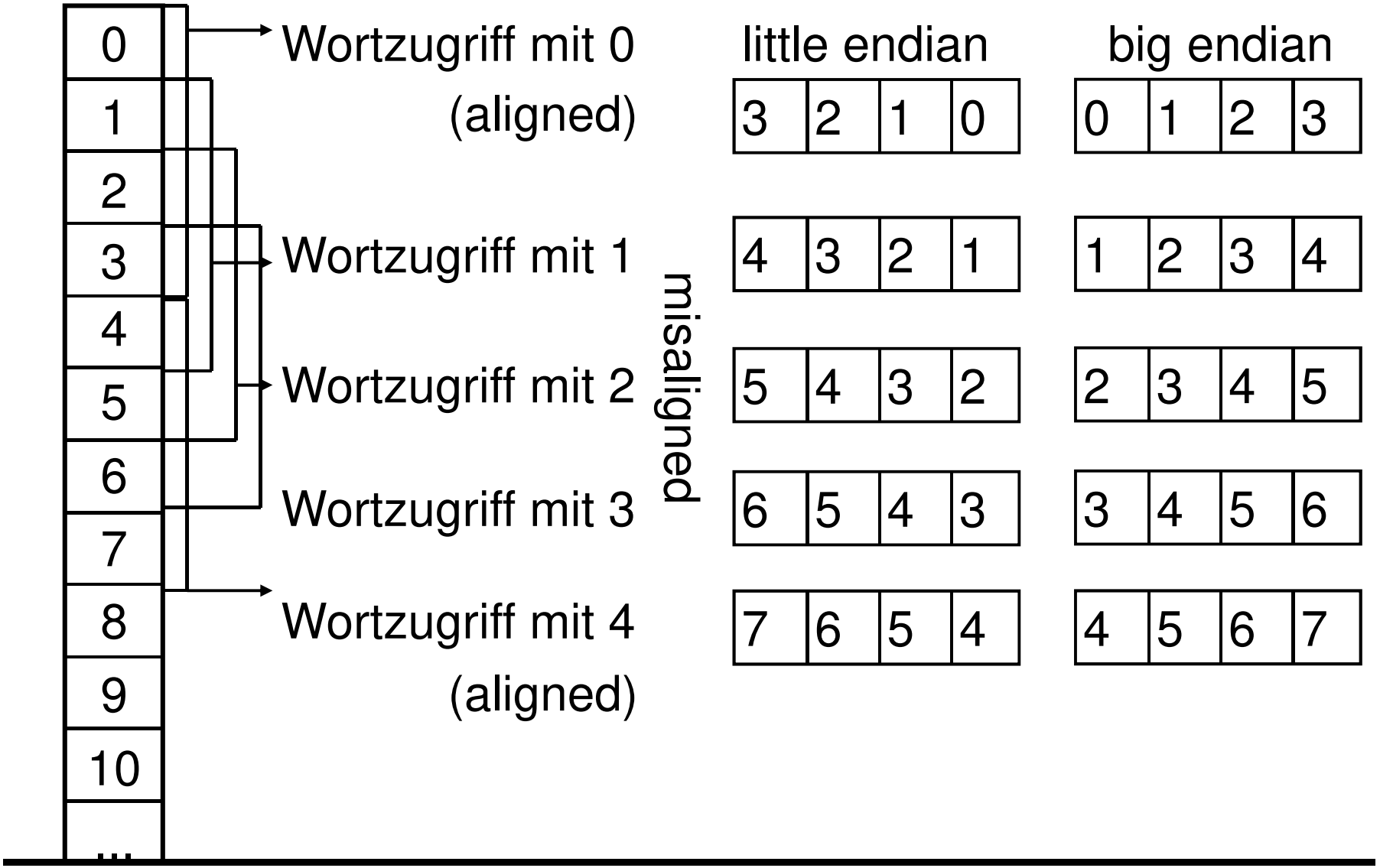
- Aligned access:

  Teilwortaddressen müssen stets ein Vielfaches der Länge in Bytes sein, d.h. *a…a0* (Halbworten) oder *a…a00* (Worte) oder *a…a000* (Doppelworte), oder

- Misaligned access:Teilworte dürfen auf beliebigen Positionen beginnen.

Misalignment ist schwieriger zu realisieren und kostet Zugriffszeit. (Erklärung folgt in 3.3)

Alignment -- Veranschaulichung



Ausblick

Wir wissen schon etwas mehr darüber, wie die Objekte unserer Maschine nun aussehen müssen:

- **Adressen**:

  Adressen stellen wir durch Worte als unsigned Zahlen dar.

- **Befehle**:

  Befehle müssen auch in Maschinenworten oder Vielfachen davon kodiert werden. Ferner kennen wir nun mehrere Sorten von Objekten: signed und unsigned Numbers, einfach und doppelt genaue Gleitkommazahlen, Bytes, Halbworte, …

  Zu all diesen Objekten benötigen wir entsprechende Verknüpfungsbefehle wie Arithmetik auf den entsprechenden Zahlen (signed, unsigned, float, double), oder Logikverknüpfungen, Schiebebefehle etc. auf Maschinenworten….