Interne Datenspeicherung

INTEGER $\rightarrow$ Festkommazahlen

- Speicherung als Dualzahl, Zahlbereich ergibt sich aus der Anzahl n verwendeter Bits,
  ohne Vorzeichen: $0 \ldots 2^n - 1$, mit Vorzeichen: $-2^{n-1} \ldots 2^{n-1} - 1$.

kind=	Bits	max. ohne Vz.	max. mit Vz.
1	8	255	127
2	16	65 535	32 767
4	32	4 294 967 295	2 147 483 647
8	64	18 446 744 073 709 551 615	9 223 372 036 854 775 807

- Für negative Zahlen wird die Komplementdarstellung benutzt:
  Wenn $0 < a < 2^{n-1}$, so wird $(-a)$ gespeichert als $(2^n - a)$
  2^n würde das $(n+1)$-te Bit benötigen, für $(2^n - a)$ reichen wieder n Bit,
  denn: $2^n > (2^n - a) > 2^{n-1}$.
- Folglich ist bei negativen Zahlen das höchste Bit (2^{n-1}) gesetzt.
- FK-Rechnungen erfolgen dann vorzeichenlos, modulo 2^n.

Beispiel zur Komplementdarstellung

- Umwandlung zwischen positiven und negativen Zahlen:

$$2^n - a = [\underbrace{(2^n - 1)}_{11\ldots1} - a] + 1,$$

also zuerst alle Bits „umdrehen“ ($0 \leftrightarrow 1$), dann $+1$.
Umkehrung geht genauso, denn: $2^n - (2^n - a) = a$.

- Bsp. $n = 8$, Umwandlung von 6 in (-6) und umgekehrt:

$a = 6 \rightarrow$	00000110	(wird von 11111111 subtrahiert)
$0 \leftrightarrow 1$	11111001	
$+1$	11111010	$\leftarrow (-6) = (-a)$

und wieder zurück:

$0 \leftrightarrow 1$	00000101	
$+1$	00000110	$\leftarrow 6 = a$

Interne Datenspeicherung - Festkommazahlen

Beispiel zur Komplementdarstellung

Bsp. vorzeichenlose Addition: $4 + (-6)$

	00000100	(4)
	11111010	(−6)
Summe:	11111110	(−2)

Festkommaüberlauf wird erkannt, wenn der Übertrag in das Vorzeichenbit (2^{n-1}) **nicht** mit dem Übertrag aus dem Vorzeichenbit übereinstimmt,

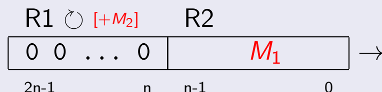
z. B. $n = 4$ (Zahlbereich $-8, \ldots, 7$)

x	0101	$(= 5)$	1111	$(= -1)$
y	0100	$(= 4)$	1110	$(= -2)$
Ü-Bits	0100-		1110-	
x+y	1001	$(= -7)$	1101	$(= -3)$
		FK-Überlauf!		kein FK-Überlauf

Interne Datenspeicherung - Festkommazahlen

Prinzipieller Ablauf ganzzahliger Multiplikation und Division mit Hilfe doppelt langer Rechenregister $[R1|R2]$.

FK-Multiplikation $M_1 * M_2$



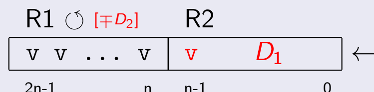
Start: M_1 in $R2$; 0 in $R1$.

Loop: (n-mal)

- Wenn Bit_0 gesetzt, addiere M_2 zu $R1$.
- $[R1|R2]$ 1 Bit nach rechts verschieben

Ergebnis: Produkt korrekt mit $2n$ Bit als vorzeichenlose Zahl.
(Kein Überlaufmerkmal)

FK-Division D_1/D_2



Start: D_1 in $R2$; $R1$ mit v auffüllen.

Loop: (n-mal)

- $[R1|R2]$ 1 Bit nach links; Bit_0 $:= 0$
- Subtrahiere Divisor D_2 von $R1$
- Bei Vz.-Wechsel in $R1$: $R1 := R1 + D_2$,
  sonst: setze Bit_0 $:= 1$

Ergebnis: Quotient in $R2$,
Divisionsrest in $R1$

Interne Datenspeicherung - Gleitkommazahlen

`REAL` / `COMPLEX` $\rightarrow$ Gleitkommazahlen

Halblogarithmische Darstellung durch [Vorzeichen, Exponent, Mantisse]
Basis für Exponenten wird nicht gespeichert (ist im Prozessor festgelegt).

Die n Bits (z.B. $n \in \{32, 64, 80, 128\}$) sind aufgeteilt auf die 3 Bestandteile:

$$n = 1 + e + M$$

$$\underbrace{\boxed{v}}_{1}\underbrace{\boxed{CH}}_{e}\underbrace{\boxed{z_1 \ \dots \ z_i \ \dots \ z_m}}_{M} \quad \Rightarrow \quad z = (-1)^v b^{exp} \sum_{i=1}^{m} z_i b^{-i}$$

1 Bit für Vorzeichen $(+ \rightarrow 0, - \rightarrow 1)$
e Bits für Exponent, gespeichert als $CH = 2^{e-1} - 1 + exp$
M Bits für Mantissenziffern $z_1, \dots, z_m$.

Jede Ziffer z_i besteht aus k Bits, wenn die Basis $b = 2^k$ ist (also $M = k * m$).
GK-Zahlen werden i.a. normalisiert, d. h. $z_1 \neq 0$ (Mantisse ohne führende Nullen).
Bei der Zahl 0.0 sind alle n Bits gleich 0. Mit $v = 1$ gibt es auch „-0.0“.

Interne Datenspeicherung - Gleitkommazahlen

Beispiel: IBM bzw. ESER

(„Großrechentechnik“ der 1960er bis 1980er Jahre)

Basis b = 16, ein Byte für Vorzeichen und Exponent, d. h. $e = 7$, $|\mathrm{exp}| \leq 63$, Mantisse besteht aus 6 bzw. 14 Hexadezimalziffern z_i (3 bzw. 7 Byte).

`REAL*4`	$\Rightarrow$	$M = 24$,	$m = 6$	$\rightarrow$	1.0	=	`40800000`
`REAL*8`	$\Rightarrow$	$M = 56$,	$m = 14$	$\rightarrow$	-1.0	=	`C080000000000000`

Zahlbereich $\approx \quad \pm(16^{-63} \dots 16^{64}) \quad \approx \quad \pm(10^{-76} \dots 10^{77})$

Bemerkung:
Bei diesem Format konnten die ersten 4 Byte einer `REAL*8`-Variable problemlos als `REAL*4`-Wert interpretiert werden (Abschneiden der Mantissenziffern $z_7, \dots, z_{14}$).
Voraussetzung: Exponent und Mantissenziffern müssen in dieser (absteigenden) Reihenfolge gespeichert sein.
(Auch eine Quelle für unentdeckte Fehler: „fast richtige“ Ergebnisse)

Interne Datenspeicherung - Gleitkommazahlen

Beispiel: IEEE (Standard für Intel & Co)

Basis b = 2, unterschiedliche Längen e bei verschiedenen REAL-Formaten:
Beachte:
höchste Mantissenziffer nicht gespeichert; immer 1 bei normalisierter GK-Zahl

$$z = (-1)^v 2^{\exp} \left(I + \sum_{i=1}^{m} z_i 2^{-i} \right), \quad I = 1, \; z_i \in \{0, 1\}.$$

Typ	kind	e	$\lvert\exp\rvert$	M = m	Zahlbereich
short real	4	8	127	23	$\pm(8.4 \cdot 10^{-37} \; \ldots \; 3.4 \cdot 10^{38})$
long real	8	11	1023	52	$\pm(4.2 \cdot 10^{-307} \; \ldots \; 1.7 \cdot 10^{308})$
temp. real	10	15	16383	63+1	$\pm(3.4 \cdot 10^{-4932} \; \ldots \; 1.2 \cdot 10^{4932})$

Nur im Format „Temporary Real“ (kind=10) wird I als z_0 mit gespeichert.

- `1.0_4` → 3F800000
- `1.0_8` → 3FF0000000000000
- `1.0_10` → 3FFF8000000000000000

Beispiele, Binärdarstellungen sind zusammengefasst, je 4 Bit als Hexadezimalziffer

Interne Datenspeicherung - Gleitkommazahlen

Beispiel: IEEE (Standard für Intel & Co)

Basis b = 2, unterschiedliche Längen e bei verschiedenen REAL-Formaten:
Beachte:
höchste Mantissenziffer nicht gespeichert; immer 1 bei normalisierter GK-Zahl

$$z = (-1)^v 2^{\exp} \left(l + \sum_{i=1}^{m} z_i 2^{-i} \right), \quad l = 1,\ z_i \in \{0,1\}.$$

Typ	kind	e	$\lvert\exp\rvert$	M = m	Zahlbereich
short real	4	8	127	23	$\pm(8.4 \cdot 10^{-37} \ \ldots\ 3.4 \cdot 10^{38})$
long real	8	11	1023	52	$\pm(4.2 \cdot 10^{-307} \ \ldots\ 1.7 \cdot 10^{308})$
temp. real	10	15	16383	63+1	$\pm(3.4 \cdot 10^{-4932} \ \ldots\ 1.2 \cdot 10^{4932})$

`exp=0, CH=`$2^{e-1}-1$

po... Real“ (kind=10) wird l als ... mit gespeichert.

0|011 1111 1|000 0000 0000 0000 0000 0000

`1.0_4` → 3F800000

0|011 1111 1111 | 0000 0000 ... 0000 0000

`1.0_8` → 3FF0000000000000

... ngen sind zusammengefasst ... Bit

0|011 1111 1111 1111 | 1000 00 ...00 0000

`1.0_10` → 3FFF8000000000000000

... als Hexadezimalziffer.

Interne Datenspeicherung - Gleitkommazahlen

Beispiel: IEEE (Standard für Intel & Co)

Basis b = 2, unterschiedliche Längen e bei verschiedenen REAL-Formaten:
Beachte:
höchste Mantissenziffer nicht gespeichert; immer 1 bei normalisierter GK-Zahl

$$z = (-1)^v 2^{\mathrm{exp}} \left(I + \sum_{i=1}^{m} z_i 2^{-i} \right), \quad I = 1,\ z_i \in \{0, 1\}.$$

Typ	kind	e	$\lvert\mathrm{exp}\rvert$	M = m	Zahlbereich
short real	4	8	127	23	$\pm(8.4 \cdot 10^{-37} \ \ldots\ 3.4 \cdot 10^{38})$
long real	8	11	1023	52	$\pm(4.2 \cdot 10^{-307} \ \ldots\ 1.7 \cdot 10^{308})$
temp. real	10	15	16383	63+1	$\pm(3.4 \cdot 10^{-4932} \ \ldots\ 1.2 \cdot 10^{4932})$

`exp=0`, `CH=`$2^{e-1} - 1$

...porary Real" (kind=10) wird I als z_0 mit gespeichert.

1|011 1111 1|000 0000 0000 0000 0000 0000

`-1.0_4` → BF800000

1|011 1111 1111 | 0000 0000 ... 0000 0000

...ngen

`-1.0_8` → BFF0000000000000

...sind zusammengefasst ... je 4 Bit

1|011 1111 1111 1111 | 1000 00 ...00 0000

`-1.0_10` → BFFF8000000000000000

...als Hexadezimalziffer.

Interne Datenspeicherung - Gleitkommazahlen

Beispiel: IEEE (Standard für Intel & Co)

Basis b = 2, unterschiedliche Längen e bei verschiedenen REAL-Formaten:
Beachte:
höchste Mantissenziffer nicht gespeichert; immer 1 bei normalisierter GK-Zahl

$$z = (-1)^v 2^{\exp} \left(I + \sum_{i=1}^{m} z_i 2^{-i} \right), \quad I = 1,\ z_i \in \{0, 1\}.$$

Typ	kind	e	$\lvert\exp\rvert$	M = m	Zahlbereich
short real	4	8	127	23	$\pm(8.4 \cdot 10^{-37} \ \ldots\ 3.4 \cdot 10^{38})$
long real	8	11	1023	52	$\pm(4.2 \cdot 10^{-307} \ \ldots\ 1.7 \cdot 10^{308})$
temp. real	10	15	16383	63+1	$\pm(3.4 \cdot 10^{-4932} \ \ldots\ 1.2 \cdot 10^{4932})$

exp=0, CH=$2^{e-1} - 1$

…porary Real“ (kind=10) wird I als z_0 mit gespeichert.

0|011 1111 1|100 0000 0000 0000 0000 0000

1.5_4 → 3FC00000

0|011 1111 1111 | 1000 0000 ... 0000 0000

1.5_8 → 3FF8000000000000

0|011 1111 1111 1111 | 1100 00 ...00 0000

1.5_10 → 3FFFC000000000000000

Interne Datenspeicherung - Gleitkommazahlen

Beispiel: IEEE (Standard für Intel & Co)

Basis b $= 2$, unterschiedliche Längen e bei verschiedenen REAL-Formaten:
Beachte:
höchste Mantissenziffer nicht gespeichert; immer 1 bei normalisierter GK-Zahl

$$z = (-1)^v 2^{\exp} \left(I + \sum_{i=1}^{m} z_i 2^{-i} \right), \quad I = 1,\ z_i \in \{0, 1\}.$$

Typ	kind	e	$\lvert\exp\rvert$	M = m	Zahlbereich
short real	4	8	127	23	$\pm(8.4 \cdot 10^{-37} \ldots 3.4 \cdot 10^{38})$
long real	8	11	1023	52	$\pm(4.2 \cdot 10^{-307} \ldots 1.7 \cdot 10^{308})$
temp. real	10	15	16383	63+1	$\pm(3.4 \cdot 10^{-4932} \ldots 1.2 \cdot 10^{4932})$

exp=0, CH=$2^{e-1} - 1$

…porary Real“ (kind=10) wird I als z mit gespeichert.

1|011 1111 1|100 0000 0000 0000 0000 0000

-1.5_4 → BFC00000

1|011 1111 1111 | 1000 0000 ... 0000 0000

-1.5_8 → BFF8000000000000

…ngen sind zusammengefasst … je 4 Bit

1|011 1111 1111 1111 | 1100 00 ...00 0000

-1.5_10 → BFFFC000000000000000

Beispiel: IEEE (Standard für Intel & Co)

Basis b = 2, unterschiedliche Längen e bei verschiedenen REAL-Formaten:
Beachte:
höchste Mantissenziffer nicht gespeichert; immer 1 bei normalisierter GK-Zahl

$$z = (-1)^v 2^{\mathrm{exp}} \left(I + \sum_{i=1}^{m} z_i 2^{-i} \right), \quad I = 1,\ z_i \in \{0, 1\}.$$

Typ	kind	e	$\lvert\mathrm{exp}\rvert$	M = m	Zahlbereich
short real	4	8	127	23	$\pm(8.4 \cdot 10^{-37} \ \ldots\ 3.4 \cdot 10^{38})$
long real	8	11	1023	52	$\pm(4.2 \cdot 10^{-307} \ \ldots\ 1.7 \cdot 10^{308})$
temp. real	10	15	16383	63+1	$\pm(3.4 \cdot 10^{-4932} \ \ldots\ 1.2 \cdot 10^{4932})$

exp=2, CH=$2^{e-1}+1$

... porary Real“ (kind=10) wird I als z_0 mit gespeichert.

0|100 0000 1|001 0000 0000 0000 0000 0000

0|100 0000 0001 | 0010 0000 ... 0000 0000

0|100 0000 0000 0001 | 1001 00 ...00 0000

... ngen

sind zusammengefasst ... je 4 Bit

als Hexadezimalziffer

4.5_4 → 40900000

4.5_8 → 4012000000000000

4.5_10 → 40019000000000000000

Interne Datenspeicherung - Gleitkommazahlen

Beispiel: IEEE (Standard für Intel & Co)

Basis b = 2, unterschiedliche Längen e bei verschiedenen REAL-Formaten:
Beachte:
höchste Mantissenziffer nicht gespeichert; immer 1 bei normalisierter GK-Zahl

$$z = (-1)^v 2^{\mathrm{exp}} \left(I + \sum_{i=1}^{m} z_i 2^{-i} \right), \quad I = 1,\ z_i \in \{0, 1\}.$$

Typ	kind	e	$\lvert\mathrm{exp}\rvert$	M = m	Zahlbereich
short real	4	8	127	23	$\pm(8.4 \cdot 10^{-37} \ \ldots\ 3.4 \cdot 10^{38})$
long real	8	11	1023	52	$\pm(4.2 \cdot 10^{-307} \ \ldots\ 1.7 \cdot 10^{308})$
temp. real	10	15	16383	63+1	$\pm(3.4 \cdot 10^{-4932} \ \ldots\ 1.2 \cdot 10^{4932})$

exp=2, CH=$2^{e-1}+1$

porary Real“ (kind=10) wird I als z_0 mit gespeichert.

1|100 0000 1|001 0000 0000 0000 0000 0000

`-4.5_4` → `C0900000`

1|100 0000 0001 | 0010 0000 . . . 0000 0000

ngen

`-4.5_8` → `C012000000000000`

1|100 0000 0000 0001 | 1001 00 . . . 00 0000

`-4.5_10` → `C0019000000000000000`

Interne Datenspeicherung - Gleitkommazahlen

Beispiel: IEEE (Standard für Intel & Co)

Basis b = 2, unterschiedliche Längen e bei verschiedenen REAL-Formaten:
Beachte:
höchste Mantissenziffer nicht gespeichert; immer 1 bei normalisierter GK-Zahl

$$z = (-1)^v 2^{\exp} \left(I + \sum_{i=1}^{m} z_i 2^{-i} \right), \quad I = 1,\ z_i \in \{0, 1\}.$$

Typ	kind	e	$\lvert\exp\rvert$	M = m	Zahlbereich
short real	4	8	127	23	$\pm(8.4 \cdot 10^{-37} \ldots 3.4 \cdot 10^{38})$
long real	8	11	1023	52	$\pm(4.2 \cdot 10^{-307} \ldots 1.7 \cdot 10^{308})$
temp. real	10	15	16383	63+1	$\pm(3.4 \cdot 10^{-4932} \ldots 1.2 \cdot 10^{4932})$

Nur im Format „Temporary Real“ (kind=10) wird I als z_0 mit gespeichert.

```
-4.5_4  → C0900000
-4.5_8  → C012000000000000
-4.5_10 → C0019000000000000000
```

Beispiele, Binärdarstellungen sind zusammengefasst, je 4 Bit als Hexadezimalziffer

Interne Datenspeicherung - Gleitkommazahlen

Beispiel: IEEE (Standard für Intel & Co)

Basis b $=$ **2**, unterschiedliche Längen e bei verschiedenen REAL-Formaten:
Beachte:
höchste Mantissenziffer nicht gespeichert; immer 1 bei normalisierter GK-Zahl

$$z = (-1)^v 2^{\mathrm{exp}} \left(l + \sum_{i=1}^{m} z_i 2^{-i} \right), \quad l = 1,\ z_i \in \{0,1\}.$$

Typ	kind	e	$\lvert\mathrm{exp}\rvert$	M = m	Zahlbereich
short real	4	8	127	23	$\pm(8.4 \cdot 10^{-37} \ \ldots\ 3.4 \cdot 10^{38})$
long real	8	11	1023	52	$\pm(4.2 \cdot 10^{-307} \ \ldots\ 1.7 \cdot 10^{308})$
temp. real	10	15	16383	63+1	$\pm(3.4 \cdot 10^{-4932} \ \ldots\ 1.2 \cdot 10^{4932})$

Spezielle Codierungen:

inf_4 → 7F800000
inf_8 → 7FF0000000000000
inf_10 → 7FFF80000000000000000

Interne Datenspeicherung - Gleitkommazahlen

Beispiel: IEEE (Standard für Intel & Co)

Basis b = 2, unterschiedliche Längen e bei verschiedenen REAL-Formaten:
Beachte:
höchste Mantissenziffer nicht gespeichert; immer 1 bei normalisierter GK-Zahl

$$z = (-1)^v 2^{\text{exp}} \left(I + \sum_{i=1}^{m} z_i 2^{-i} \right), \quad I = 1,\ z_i \in \{0, 1\}.$$

Typ	kind	e	$\lvert\text{exp}\rvert$	M = m	Zahlbereich
short real	4	8	127	23	$\pm(8.4 \cdot 10^{-37} \ \ldots\ 3.4 \cdot 10^{38})$
long real	8	11	1023	52	$\pm(4.2 \cdot 10^{-307} \ \ldots\ 1.7 \cdot 10^{308})$
temp. real	10	15			$\pm(3.4 \cdot 10^{-4932} \ \ldots\ 1.2 \cdot 10^{4932})$

$\text{exp}=2^{e-1}$, $\text{CH}=2^e-1$

Spezielle Codierungen:

inf_4 → 7F800000

0|111 1111 1|000 0000 0000 0000 0000 0000

inf_8 → 7FF0000000000000

0|111 1111 1111 |0000 0000 ... 0000 0000

inf_10 → 7FFF80000000000000000

0|111 1111 1111 1111 |1000 00 ...00 0000

Interne Datenspeicherung - Gleitkommazahlen

Beispiel: IEEE (Standard für Intel & Co)

Basis b = 2, unterschiedliche Längen e bei verschiedenen REAL-Formaten:
Beachte:
höchste Mantissenziffer nicht gespeichert; immer 1 bei normalisierter GK-Zahl

$$z = (-1)^v 2^{\mathrm{exp}} \left(I + \sum_{i=1}^{m} z_i 2^{-i} \right), \quad I = 1,\ z_i \in \{0,1\}.$$

Typ	kind	e	|exp|	M = m	Zahlbereich
short real	4	8	127	23	$\pm(8.4 \cdot 10^{-37} \ \ldots\ 3.4 \cdot 10^{38})$
long real	8	11	1023	52	$\pm(4.2 \cdot 10^{-307} \ \ldots\ 1.7 \cdot 10^{308})$
temp. real	10	15			$\pm(3.4 \cdot 10^{-4932} \ \ldots\ 1.2 \cdot 10^{4932})$

exp=2^{e-1}, CH=2^e-1

Spezielle Codierungen:

`-inf_4` → `FF800000` — 1|111 1111 1|000 0000 0000 0000 0000 0000

`-inf_8` → `FFF0000000000000` — 1|111 1111 1111 |0000 0000 ... 0000 0000

`-inf_10` → `FFFF8000000000000000` — 1|111 1111 1111 1111 |1000 00 ...00 0000

Interne Datenspeicherung - Gleitkommazahlen

Beispiel: IEEE (Standard für Intel & Co)

Basis b = 2, unterschiedliche Längen e bei verschiedenen REAL-Formaten:
Beachte:
höchste Mantissenziffer nicht gespeichert; immer 1 bei normalisierter GK-Zahl

$$z = (-1)^v 2^{\exp} \left(I + \sum_{i=1}^{m} z_i 2^{-i} \right), \quad I = 1,\ z_i \in \{0, 1\}.$$

Typ	kind	e	$\|\exp\|$	M = m	Zahlbereich
short real	4	8	127	23	$\pm(8.4 \cdot 10^{-37} \ \ldots \ 3.4 \cdot 10^{38})$
long real	8	11	1023	52	$\pm(4.2 \cdot 10^{-307} \ \ldots \ 1.7 \cdot 10^{308})$
temp. real	10	15			$\pm(3.4 \cdot 10^{-4932} \ \ldots \ 1.2 \cdot 10^{4932})$

$\exp = 2^{e-1}$, $CH = 2^e - 1$

Spezielle Codierungen:

nan_4 → FFC00000 — 1|111 1111 1|100 0000 0000 0000 0000 0000

nan_8 → FFF8000000000000 — 1|111 1111 1111 | 1000 0000 ... 0000 0000

nan_10 → FFFFC000000000000000 — 1|111 1111 1111 1111 | 1100 00 ... 00 0000

Interne Datenspeicherung - Dezimalzahlen

BCD - Binary Coded Decimals — z. B. PL/1, COBOL

- Für eine Dezimalziffer $d \in \{0, \ldots, 9\}$ werden 4 Bits verwendet, also pro Byte 2 Ziffern (gepacktes Format)
- Im entpackten Format steht pro Byte eine Dezimalziffer d im niederen Halbbyte, so dass das höhere Halbbyte daraus einen (druckbaren) Zeichencode machen kann, z. B. ASCII: $\boxed{3\ d}$ oder EBCDIC: $\boxed{F\ d}$.
- Prozessoren besitzen „Korrekturbefehle“ (z. B. `DAA`, Decimal Adjust after Addition), die nach (oder vor) jeder Binäroperation mit BCD-Zahlen benutzt werden, damit das Ergebnis wieder als BCD erscheint, z.B.

$$\boxed{17} + \boxed{28} = \boxed{3F} \xrightarrow{\texttt{DAA}} \boxed{45}$$

- IEEE-Standard zum Rechnen mit 10-Byte-BCD-Zahlen:

$v\ 0$	$d_{17} d_{16}$	$\ldots$	$d_1\ d_0$

BCD - Binary Coded Decimals — z. B. PL/1, COBOL

- Für eine Dezimalziffer $d \in \{0, \dots, 9\}$ werden 4 Bits verwendet, also pro Byte 2 Ziffern (gepacktes Format)
- Im entpackten Format steht pro Byte eine Dezimalziffer d im niederen Halbbyte, so dass das höhere Halbbyte daraus einen (druckbaren) Zeichencode machen kann, z. B. ASCII: $\boxed{3\ d}$ oder EBCDIC: $\boxed{F\ d}$.
- Prozessoren besitzen „Korrekturbefehle“ (z. B. `DAA`, Decimal Adjust after Addition), die nach (oder vor) jeder Binäroperation mit BCD-Zahlen benutzt werden, damit das Ergebnis wieder als BCD erscheint, z.B.

$$\boxed{17} + \boxed{28} = \boxed{3F} \xrightarrow{\texttt{DAA}} \boxed{45}$$

- …hen mit 10-Byte-BCD-Zahlen:

v 0	$d_{17}\, d_{16}$	...	$d_1\ d_0$

Vorzeichenbyte:

$\boxed{v000\,0000}$, $v = \begin{cases} 0 & + \\ 1 & - \end{cases}$

Interne Datenspeicherung - Zeichenketten

CHARACTER → Zeichenkette

- gebräuchliche Codierung für Zeichen (mit 1 Byte pro Zeichen): ASCII (American Standard Code of Information Interchange)
- ASCII-Standard nutzt 7 Bit

Steuercodes	00, ..., 1F	(0,...,31)
Textzeichen	20, ..., 7F	(32,...,127)
erweiterter Zeichensatz	80, ..., FF	(128,...,255)
(z. B. IBM-Zeichensatz CP437 mit Pseudografik-/Sonderzeichen)		

- Vorsicht mit Sonderzeichen (incl. Umlaute), wenn interne Codierung nicht mit der außerhalb des Programms am Computer verwendeten übereinstimmt.
- Verwendet man UTF-8-Zeichen im Standardtyp `CHARACTER`, ist die Anzahl der Zeichen ggf. kleiner als die Anzahl der Bytes (= `LEN_TRIM(str)`),

  z. B. `str = char(int(z'C6'))//char(int('8E'))`
  liefert **ein** Zeichen '∃', hat aber die Länge **2**.

Interne Datenspeicherung - Zeichenketten

CHARACTER → Zeichenkette

- gebräuchliche Codierung für Zeichen (mit 1 Byte pro Zeichen): ASCII (American Standard Code of Information Interchange)
- ASCII-Standard nutzt 7 Bit

Steuercodes	00, ..., 1F
Textzeichen	20, ..., 7F
erweiterter Zeichensatz	80, ..., FF

(z. B. IBM-Zeichensatz CP437 mit P

Codierung von Sonderzeichen (Bsp.)

	IBM	latin1	utf-8
Ä	z'8E'	z'C4'	z'C384'
Ö	z'99'	z'D6'	z'C396'
Ü	z'9A'	z'DC'	z'C39C'
ß	z'E1'	z'DF'	z'C39F'

- Vorsicht mit Sonderzeichen (incl. Umlaute), wenn interne Codierung nicht mit der außerhalb des Programms am Computer verwendeten übereinstimmt.
- Verwendet man UTF-8-Zeichen im Standardtyp `CHARACTER`, ist die Anzahl der Zeichen ggf. kleiner als die Anzahl der Bytes (= `LEN_TRIM(str)`),

  z. B. `str = char(int(z'C6'))//char(int('8E'))`
  liefert **ein** Zeichen '∃', hat aber die Länge **2**.

Interne Datenspeicherung - Zeichenketten

CHARACTER → Zeichenkette

- gebräuchliche Codierung für Zeichen (mit 1 Byte pro Zeichen): ASCII (American Standard Code of Information Interchange)
- ASCII-Standard nutzt 7 Bit

Steuercodes	00, ..., 1F	(0,...,31)
Textzeichen	20, ..., 7F	(32,...,127)
erweiterter Zeichensatz	80, ..., FF	(128,...,255)

(z. B. IBM-Zeichensatz CP437 mit Pseudografik-/Sonderzeichen)

- Vorsicht mit Sonderzeichen (incl. Umlaute), wenn interne Codierung nicht mit der außerhalb des Programms am Computer verwendeten übereinstimmt.
- Neben ASCII wird auch UCS-4 (Universal Character Set, ISO_10646) unterstützt (Kind-Selektor entspricht Anzahl Byte pro Zeichen):
  `SELECTED_CHAR_KIND("ascii")` → 1
  `SELECTED_CHAR_KIND("ISO_10646")` → 4