Bemerkung 1.2

$$\textit{9er-Komplement}\Big(\sum_{i=0}^{N} y_i \cdot 10^i\Big) = \textit{10er-Komplement}\Big(\sum_{i=0}^{N} y_i \cdot 10^i\Big) - 1$$

Mit Hilfe des 10er Komplements kann also im Beispiel der Subtraktion wie folgt vorgegangen werden:

$$\sum_{i=0}^{N} x_i \cdot 10^i - \sum_{i=0}^{N} y_i \cdot 10^i = \sum_{i=0}^{N} x_i \cdot 10^i + \left(\sum_{i=0}^{N}(9-y_i)\cdot 10^i + 1 \underbrace{- \sum_{i=0}^{N} 9 \cdot 10^i - 1}_{=-10^{N+1}}\right),$$

d.h. man addiert das 9er Komplement von y, korrigiert um $+1$ und *vergißt* einen eventuell auftretenden Überlauf in der $(N+2)$-ten Stelle. Codiert man $-y$ durch das Zehnerkomplement von y, so benötigt die Vorzeichenwechseloperation $(N+1)$ einstellige Subtraktionen und eine $(N+1)$-stellige Korrekturaddition, die Subtraktion $x-y=x+(-y)$ braucht jedoch nur eine $(N+1)$-stellige Addition.

$$\sum_{i=0}^{1} x_i 10^i \in \{0, ..., 99\}$$

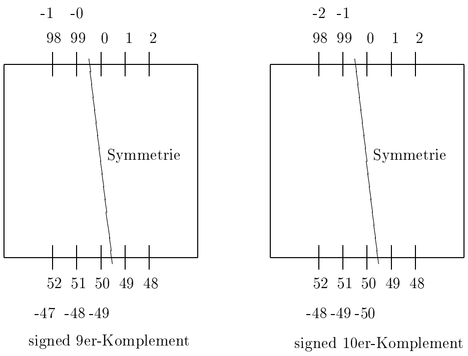


Abbildung 1.23: Darstellung von Zahlen im 9er bzw. 10er Komplement

Es stellt sich die Frage, welche N-stelligen Zahlen nun negativ und welche positiv interpretiert werden müssen.

Nach Abbildung 1.23 ergibt sich für die zweistelligen Zahlen im 9er Komplement ein dargestellter Zahlenbereich von

$$\{-49, -48, \ldots, -0, 0, 1, \ldots, 49\} = \{-(10^N\, div\, 2{+}1), \ldots, -0{+}0, \ldots, 10^N\, div\, 2{-}1\}$$

und im 10er Komplement

$$\{-50, -49, \ldots, -1, 0, 1, \ldots, 49\} = \{-(10^N\, div\, 2), \ldots, -0+0, \ldots, 10^N\, div\, 2 - 1\}$$

Im 10er Komplement ergibt sich also

$$Wert(x_N \ldots x_1 x_0) = \begin{cases} \sum_{i=0}^{N} x_i 10^i & \text{, falls } x_N \in \{0,1,2,3,4\} \\ -10^{N+1} + \sum_{i=0}^{N} x_i 10^i & \text{, falls } x_N \in \{5,6,7,8,9\} \end{cases}$$

Wie ist nun für zwei vorzeichenbehaftete (*signed*) 10er Komplement-Zahlen a, b der Wert $W := Wert(a_N \ldots a_0) + Wert(b_N \ldots b_0)$ zu berechnen?

1. $a_N, b_N \in \{0,1,2,3,4\}$:

$$\begin{aligned} W &= \sum_{i=0}^{N} a_i 10^i + \sum_{i=0}^{N} b_i 10^i \\ &= \sum_{i=0}^{N} (a_i + b_i) 10^i \\ &= \sum_{i=0}^{N} \underbrace{\left(a_i + b_i + \left(\sum_{j=0}^{i-1} (a_j + b_j) 10^j\right) \text{ div } 10^i\right)}_{=:z_i} \cdot 10^i \end{aligned}$$

   Falls $z_N \geq 5$ ergibt sich ein Overflow, d.h. W ist nicht im signed 10er Komplement darstellbar! Die Rechnung leistet besser der folgende iterative Algorithmus:

   `uebertrag`:= 0;
   `i`:= 0;
   Wiederhole
   $z_i := x_i + y_i +$ `uebertrag`;
   `uebertrag`:= z_i div 10;
   $z_i := z_i$ mod 10;
   `i:=i+1`
   bis (`i`$\geq N+1$);
   falls (`uebertrag`$\neq 0$) ...

2. $a_N, b_N \in \{5,6,7,8,9\}$: der Algorithmus funktioniert ähnlich, ist $z_N \in \{0,1,2,3,4\}$ so ergibt sich ein Underflow!

3. $a_N \in \{0,1,2,3,4\}, b_N \in \{5,6,7,8,9\}$ oder $a_N \in \{5,6,7,8,9\}, b_N \in \{0,1,2,3,4\}$: wieder ähnlich, aber keine Probleme mit eventuellem Over- oder Underflow.

1.2.2.5 Negative Zahlen im Dualsystem

Auf Großrechnern werden negative Zahlen zuweilen im 1er-Komplement dargestellt, meist jedoch ist eine Darstellung im 2er-Komplement üblich:

Bit 0..1	$0, -1$
Byte 00..FF	$-128, .., +127$
	Dabei: oberstes Bit$=1 \Leftrightarrow$ Wert <0
	Eigentlich: $0, .., 127, -128, -127, .., -1$
Halbwort (16 Bit)	$-2^{15} = -32768, .., +32767 = 2^{15}-1$
Wort (32 Bit)	$-2^{31}, .., 2^{32}-1$
Doppelwort (64 Bit)	$-2^{63}, .., 2^{63}-1$
Quadwort (128 Bit)	$-2^{127}, .., 2^{127}-1$

Ein Vorzeichenwechsel wird dabei durch

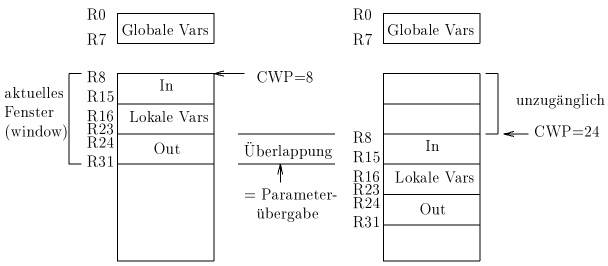
$$x \mapsto \overline{x} + 1$$

realisiert, wobei $\overline{x}$ das bitweise Komplement von x darstellt, also $0 \leftrightarrow 1$. Dies funktioniert allerdings bei der kleinsten negativen Zahl nicht. Bei der Addition tritt ein Überlauf genau dann auf, wenn im obersten Bit ein Übertrag stattfindet oder (xor) aus dem obersten Bit heraus ein Übertrag erfolgt. Leider stellen viele Programmiersprachen (z.B. Pascal) einen solchen Überlauf nicht fest!
Die Multiplikation zweier Zahlen erfolgt analog zum schriftlichen Rechenverfahren, in CISC-CPU's häufig mittels Microprogrammen, sonst mittels Parallel- oder durch Seriell-Addierer. Gleiches gilt für die Division. Spezialfunktionen wie *log* oder *arctan* werden durch abbrechende Taylorreihen, Newtoniteration, Kettenbrüche, Funktionaltheoreme o.ä. realisiert. Dies geschieht meist in Software.

1.2.2.6 Die SPARC-CPU im Inneren

Die Register des SPARC V8-Prozessors (SUPER-SPARC) sind durch verschiedene Merkmale gekennzeichnet:

1. Registerstack



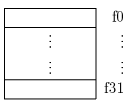
   CWP steht für *current window pointer*. Die SPARCclassic besitzt z. B. acht solcher Fensterbereiche. Auf die Register greift man mit *logischen* Namen zu:

`%g0..%g7`	global vars
`%i0..%i7`	in
`%l0..%l7`	local vars
`%o0..%o7`	out

   mit `%g0`$\equiv$0 (fest), `%o6=%sp` (stack pointer), `%o7` *return adress* für Unterprogramm-Rücksprung.

2. $\boxed{}$ Y für Multiplikation und Division.

3. Gleikommaregister:



   Je 32Bit-Gleitkommaregister für 16 *double precision* oder 8 *quad precision* Gleitkommazahlen.

4. State-Register sind unter anderem PSR *processor state register*, FSR *floating point state register*, PC *program counter*, nPC *next program counter*, WIM *window invalid maske*, TBR *trap base register*, IU-deferred trap queue, FP-deferred trap queue, …

Befehle und Ausführungszeiten bei SPARC(RISC)-ALU's

LDSB	load signed byte sign extended
LDSH	load signed halfword sign extended
LDUB	load unsigned byte zero filled
LDUH	load unsigned halfword zero filled
LD	load word
LDD	load doubleword
LDF	load floating-point
LDDF	load double floating-point
LDFSR	load floating-point state register
STB	store byte
⋮	
LDSTUB	atomic load/store unsigned byte
SWAP	swap register with memory
SETHI	set 22 bit of register
NOP	no operation
AND	
ANDcc	... and modify icc (= integer condition codes) of PSR
OR	
ORN	A or not B
XOR	
XNOR	exclusive NOR = XOR not
SLL	shift left logical
SRL	shift right logical (fill zero)
SRA	shift right arithmetic (fill most significant bit)
ADD	A+B
ADDcc	A+B and modify icc
ADDX	add with carry = A + B + c
⋮	
SUB	subtract
⋮	
MULSss	multiply step (zusammen mit Register Y für 64Bit-Ergebnis)

UMUL	unsigned integer multiply	
SMUL	signed integer multiply	
UDIV	unsigned integer division	
SDIV	signed integer division	
SAVE	register stack	
RESTORE	register stack	
Bicc	branch on integer condition codes, z.B.	
	BA	branch always
	BN	branch never
	BNE	branch not equal
FBfcc	branch on floating point condition codes	
CALL	call and link	
JMPL	jump and link	
RETT	return from Trap	
Ticc	Trap on icc, z.B. TA, TN, ...	
⋮	(read/write state ... registers)	
UNIMP	unimplemented	
FLUSH	flush instruction memory	
FPop	(rounding direction, wahlweise *to nearrest, towards 0, towards* $-\infty$*, towards* $+\infty$)	
	FiTOs	integer to single
	FiTOd	integer to double
	FiTOq	integer to quad
	FsTOi	
	FdTOi	
	FqTOi	
	FsTOd	
	FsToq	
	⋮	
	FMOVs	
	FNEGs	
	FABSs	
	FSQRTs	
	FADDs	
	FADDd	
	FADDq	
	FSUBs	
	FMULs	
	FsMULd	entspricht $s \cdot s \to d$

FdMULq	entspricht $d \cdot d \to q$
FDIVs	
FCMPs	
FCMPEs	compare and exception if unordered
⋮	

Ein Beispiel-PASCAL Programm ...

```
program Tabelle(output);
    var
        i : integer ;
    function f (x : real) : real;
    const
        pi = 3.141592653589793 ;
    var
        arg : real;
    begin
        arg := 2.0 * pi * x + 5.0;
        f   := sin(arg);
    end;
begin
        writeln("f(arg)");
        writeln("——");
        for i := 0 to 20 do
            writeln(i/10.0,f(i/10.0));
end.
```

... und was daraus in SPARC-Maschinenbefehlen wird (pc -s):

```
        .section  ".data"
        .align 4
        .common i,4,4
        .common PROGFRAME,4,4
        .align 4
.L15:
        .ascii    "%21.14e"
        .byte 0
        .section  ".text"
        .proc 010
        .global f
```



```
        .align 4
f:
        !#PROLOGUE# 0
        sethi   %hi(.LF2),%g1
        add     %g1,%lo(.LF2),%g1
        save    %sp,%g1,%sp
        !#PROLOGUE# 1
        st      %i0,[%fp+0x44]
        st      %i1,[%fp+0x48]
.L3:
.L4:
        .stabs "SC3.0.1 07 Nov 1994 Pascal 3.0.3 patch 101912-01 NN",0x30,0,0xd,0x0
        .ident  "SC3.0.1 07 Nov 1994 Pascal 3.0.3 patch 101912-01"
        .stabs  "Tabelle.p",0x30,0,0x1,0x1
        .file   "Tabelle.p"
        .xstabs ".stab.index", "V=2.0",0x3c,0,0,0x33b3734e
!       arg := 2.0 * pi * x + 5.0;
        sethi   %hi(.L2000000),%o0
        ldd     [%o0+%lo(.L2000000)],%f0
        ld2     [%fp+0x44],%f2
        fmuld   %f0,%f2,%f0
        sethi   %hi(.L2000001),%o1
        ldd [%o1+%lo(.L2000001)],%f4
        faddd   %f0,%f4,%f6
        std     %f6,[%fp+-0x20]
!       f := sin(arg);
        ldd     [%fp+-0x20],%o0
        call    __sin,2
        nop
        std     %f0,[%fp+-0x18]
.L6:
.L7:
        ldd     [%fp+-0x18],%f0
        b       .LE2
        nop
.LE2:
        ret
        restore
        .optim "-O Q R S"
        .LF2 = -128
        .LP2 = 96
        .LST2 = 96
```

```
               .LT2 = 96
               .type f,#function
               .size f,.-f
               .section   ".rodata"
               .align 8
.L2000000:     .word      0x401921fb,0x54442d18
               .align 8
.L2000001:     .word      0x40140000,0x0
               .section   ".text"
               .proc      022
               .global    program

               .align 4
program:
               !#PROLOGUE# 0
               sethi      %hi(.LF1),%g1
               add        %g1,%lo(.LF1),%g1
               save       %sp,%g1,%sp
               !#PROLOGUE# 1
.L8:
.L9:
               .xstabs ".stab.index", "Tabelle",0x2a,0,0,0
               call       __PC0__tb_init,0
               nop
               sethi      %hi(PROGFRAME),%o2
               st         %fp,[%o2+%lo(PROGFRAME)]
!              writeln("f(arg)");
               set        output,%o0
               call       __PC0__UNIT,1
               nop
               st         %o0,[%fp+-0x8]
               ld         [%fp+-0x8],%o3
               ld         [%o3+0xc],%o1
               mov        0xa,%o0
               call       fputc,2
               nop
!              writeln("——");
               set        output,%o0
               call       __PC0__UNIT,1
               nop
               st         %o0,[%fp+-0x8]
               ld         [%fp+-0x8],%o4
```



```
               ld         [%o4+0xc],%o1
               mov        0xa,%o0
               call       fputc,2
               nop
!              for i := 0 to 20 do
               st         %g0,[%fp+-0x10]
.L12:
               ld         [%fp+-0x10],%o5
               sethi      %hi(i),%o7
               st         %o5,[%o7+%lo(i)]
!              writeln(i/10.0, f(i/10.0));
               set        output,%o0
               call       __PC0__UNIT,1
               nop
               st         %o0,[%fp+-0x8]
               ld         [%fp+-0x8],%l0
               ld         [%l0+0xc],%o0
               sethi      %hi(.L2000002),%l1
               ldd        [%l1+%lo(.L2000002)],%f8
               ld         [%fp+-0x10],%f10
               fitod      %f10,%f12
               fdivd      %f12,%f8,%f12
               st         %f12,[%sp+.LP1]
               ld         [%sp+.LP1],%o2
               st         %f13,[%sp+.LP1]
               ld         [%sp+.LP1],%o3
               set        .L15,%o1
               call       fprintf,4
               nop
               sethi      %hi(.L2000002),%l2
               ldd        [%l2+%lo(.L2000002)],%f14
               ld         [%fp+-0x10],%f16
               fitod      %f16,%f18
               fdivd      %f18,%f14,%f18
               st         %f18,[%sp+.LP1]
               ld         [%sp+.LP1],%o0
               st         %f19,[%sp+.LP1]
               ld         [%sp+.LP1],%o1
               call       f,2
               nop
               std        %f0,[%sp+.LP1+0x8]
               ld2        [%sp+.LP1+0x8],%o2
```



```
               ld         [%fp+-0x8],%l3
               ld         [%l3+0xc],%o0
               set        .L15,%o1
               call       fprintf,4
               nop
               ld         [%fp+-0x8],%l4
               ld         [%l4+0xc],%o1
               mov        0xa,%o0
               call       fputc,2
               nop
.L13:
               ld         [%fp+-0x10],%l5
               add        %l5,0x1,%l5
               st         %l5,[%fp+-0x10]
               ld         [%fp+-0x10],%l6
               cmp        %l6,0x14
               bg         .L16
               nop
               b          .L12
               nop
.L16:
.L14:
.L11:
.L17:
               b          .LE1
               nop
.LE1:
               ret
               restore
               .optim "-O Q R S"
               .LF1 = -128
               .LP1 = 96
               .LST1 = 112
               .LT1 = 112
               .type      program,#function
               .size      program,.-program
               .section   ".rodata"
               .align 8
.L2000002:     .word      0x40240000,0x0
               .section   ".data"
               .align 4
```