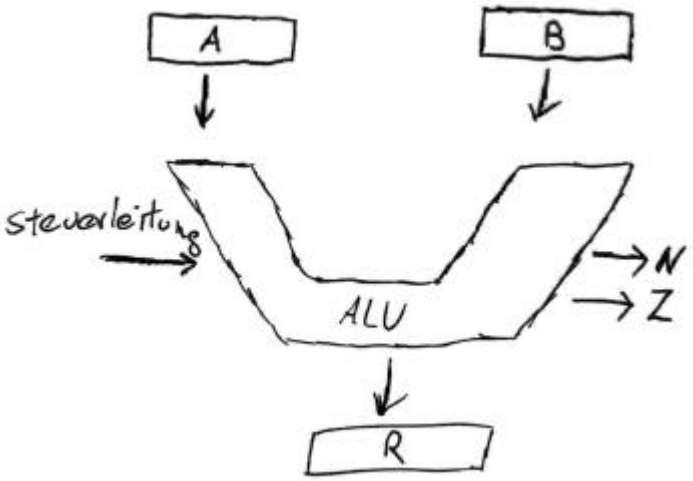
Vortrag von René Grohmann und Mirwais Turjalei, 22.11.2000

Computer Arithmetik

Computer Arithmetik Allgemein

Die ALU: Die Alu ist die Einheit im Computer, die dazu bestimmt ist arithmetische und logische Operationen durchzuführen. Sie arbeitet eng mit so gut wie allen anderen Komponenten des Computers zusammen, unter anderem mit der Steuereinheit, dem Speicher, den Registern und den Input/Output (IO) Komponenten. In diesem Vortrag beziehen wir uns ausschließlich auf die arithmentischen Aufgaben der ALU und nicht auf die logischen. Im Mittelpunkt stehen also Addition, Subtraktion, Multiplikation und Division. Eine einfache ALU besteht aus 2 Eingangs-Registern, einem Ergebnis-Register, Steuerleitungen und so genannten Flags. Bei den Registern handelt es sich jeweils um sogenannte Schieberegister, welche den Zugriff auf einzelne Bits ermöglichen. Die Flags sind einzelne Bits die zum Beispiel angeben, ob ein Ergebnis positiv oder negativ ist, oder ob ein Ergebnis null ist usw. Nun sehen wir uns einmal den groben Aufbau einer einfachen Alu an.

A & B	: Eingangs-Register
R	: Ergebnis-Register
Steuerleitungen	: Bringen die Befehle über die auszuführende Operation in die ALU
N, Z	: Beispiel Flags, N ist 1 wenn das Ergebnis negativ ist und Z wenn das Ergebnis 0 ist.



Integer Arithmetik

Integer Repräsentation im Rechner: Es gibt einige verschiedene Möglichenkeiten Ganzzahlen (Integers) im Rechner darzustellen. Die einfachste Methode ist ausschließlich mit positiven Integers zu arbeiten, dann würden sich mit 8 bit die Zahlen
von 0 bis 255 darstellen lassen. Wie man schnell feststellt ist es sehr unbefriedigend nur mit positiven Zahlen arbeiten zu können, daher gibt es noch ein paar andere Methoden. Zum einen ist da die Sign-Magnitude Methode, bei der das höchstwertige Bit als Vorzeichenbit benutzt wird, ist es eins, ist die Zahl negativ, sonst positiv. Es würden sich so mit 8 Bit die Zahlen von –127 bis +127 darstellen lassen. Das Problem bei dieser Darstellungsform ist nur, das es zum einen 2 Darstellungen für die Null gibt ($1000_2 = 0000_2 = 0_{10}$) und zum anderen die Addition und die Subtraktion bereits sehr kompliziert wären. Dann gibt es noch die 1's Complement (Einer-Komplement) Methode, bei der aus einer positiven Zahl eine negative wird, wenn man die Zahl bitweise invertiert. Es würden sich so ebenfalls mit 8 Bit die Zahlen von –127 bis +127 darstellen lassen, nur das die Addition und Subtraktion schon wesentlich einfacher wären. Allerdings gäbe es weiterhin das Problem mit den 2 Darstellungen für die 0 ($0000_2 = 1111_2 = 0_{10}$). Die wohl beste und somit auch verbreiteste Darstellungsform ist wohl die 2's Complement (Zweier-Komplement) Methode, bei ihr wird aus einer positiven eine negative Zahl, in dem man sie zuerst bitweise invertiert und dann zu der entstandenen Zahl eine 1 addiert. Diese Abbildung funktioniert in beide Richtungen, das heisst man kann auch ohne Probleme aus einer negativen Zahl wieder die zugehörige positive Zahl berechnen. Der Vorteil bei dieses Methode ist das es nur eine Darstellung für die 0 gibt (0000_2) und das die Addition und Subtraktion sehr einfach zu realisieren sind. Es gibt nur eine kleine Unregelmäßigkeit bei dieser Methode die man beachten sollte, da man nur eine Darstellung für die Null hat, bleibt eine Zahl mehr übrig, welches die nächste negative Zahl ist, bei 8 Bit zum Beispiel –128 (10000000_2). Bildet man aus dieser das Zweier-Komplement, entsteht wieder die selbe Zahl. Es lassen sich also im Zweier-Komplement mit 8 Bit die Zahlen von –128 bis +127 darstellen.

Addition und Subtraktion:

Sign-Magnitude Methode: Wie bereits erwähnt ist die Addition bei dieses Methode bereits sehr aufwendig, daher will ich drauf nicht weiter eingehen, sondern nur anhand eines kleinen Beispiels zeigen das es nicht mit einem normalen Addierwerk funktioniert:

```
 0011 (+3)
+1011 (-3)
 1110 (-6)
```

Das 3 + (-3) nicht nicht (-6) sind erkennt wohl jeder, daher brauche ich darauf wohl auch nicht weiter einzugehen.

1's Complement: Hier ist die Addition bereits etwas einfacher, wie man an einem einfachen Beispiel sehen kann, funktioniert dort schon die Benutzung eines normalen Addierwerkes mit einer kleinen Unregelmäßigkeit, auf die ich gleich noch eingehen werde.

```
 0011 (+3)
+1100 (-3)
 1111 ( 0)
```

Dieses Ergebnis ist richtig. Die Unregelmäßigkeit kommt nur Zustande, wenn es bei der Addition einen Übertrag gibt, dann wird allerdings einfach zum entstandenen Ergebnis noch eine 1 Addiert. Dieses nennt man das „end-around carry“.

2’s Complement: Nun zur am einfachsten zu benutzenden Methode, das sogenannte Zweier-Komplement. Hierbei kann die Addition (bzw. auch Subtraktion) mit einem normalen Addierwerk durchgeführt werden. Eventuell entstehende Überträge fallen einfach weg. Man muß nur darauf achten, das wenn das Vorzeichen (höchstwertiges Bit) der Summe anders ist als das beider Summanden, ist es zu einem Überlauf gekommen, das Ergebnis liegt also außerhalb des Wertebereichs und ist somit nicht korreckt. Hier 2 Beispiele

```
 0010 (+2)
+1001 (-7)
 1011 (-5)
```

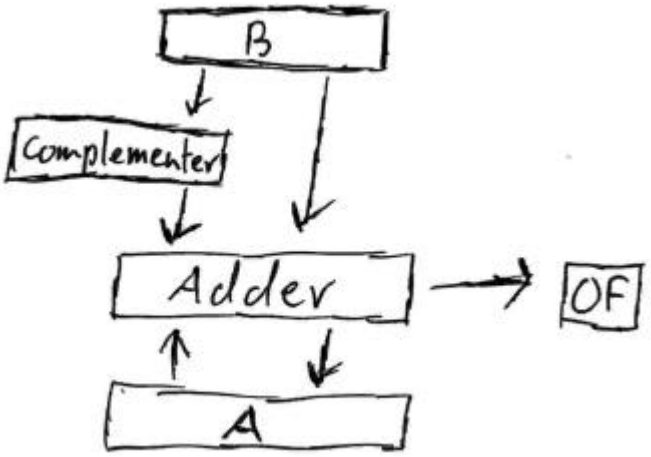
Das ist eindeutig richtig.

```
 0111 (+7)
+0111 (+7)
 1110 (-2)
```

Das Ergebnis wäre nicht richtig, aber da die Vorzeichen der Summanden anders sind als das der Summe, sieht man das es zu einem Überlauf gekommen ist.

Nun will ich noch einmal kurz demonstrieren, wie man sich so eine Addition bzw. Subtraktion nach dem Zweier-Komplement im Rechner vorstellen muß.

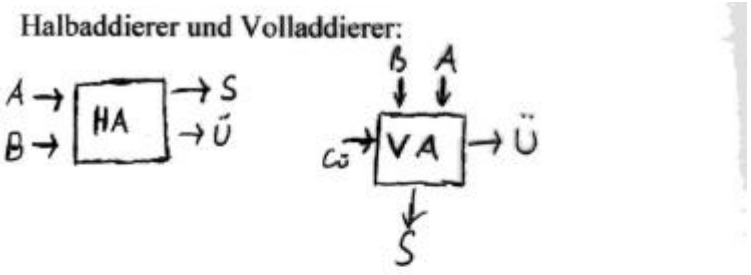
A, B : Sind die Register, das Ergebnis wird nach A zurückgeschrieben
Adder : Ein einfaches Addierwerk
Complementer : Bildet das Komplement
OF : Flag das gesetzt wird, wenn es zu einem Überlauf kommt.

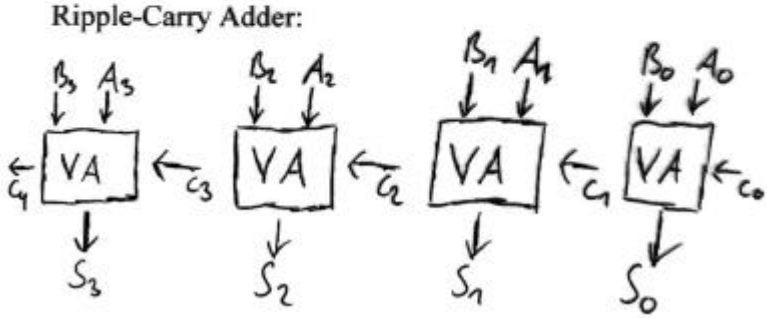


Bei der Addition wird der 2. Summand normal aus Register B vom Addierwerk gelesen, bei der Subtraktion wird erst das Komplement gebildet, womit man dann die Subtraktion auf eine Addition des Komplements zurückgeführt hat.

Addierer: Für diejenigen die noch nicht wissen wie so ein Addierer aufgebaut ist, zeige ich hier noch einmal kurz den Aufbau von Volladdieren, Halbaddieren und einem einfachen Addierwerk.

A,B : Sind immer die Eingangs Bits
S : Das Summen Bit
Ü : Der eventuell entstehende Übertrag der beiden Bits A und B





Der Ripple-Carry Adder ist ein sehr einfacher Addierer, der zwar einen relativ geringen Aufwand an Hardware fordert, aber dafür auch nicht sehr schnell ist. Zu einigen Alternativen komme ich daher später nocheinmal.

Unsigned Integer Multiplikation und Division: Jetzt möchte ich ersteinmal zeigen, wie die Multiplikation und die Division bei Integers ohne Vorzeichen verlaufen. Sie sind schon wesentlich aufwendiger als Addition und Subtraktion**,** was man leicht an den folgenden Algorithmen und Beispielen sehen kann.

Multiplikation

Algorithmus: Initalisiere P mit 0
Wiederhole n mal (n Länge der Faktoren):
(1) Wenn letztes Bit von A ist 1, dann addiere B zu P, sonst addiere 0 zu P
(2) Shift Right mit Carry-Out der Summe ins höchste Bit von P und das niedrigste Bit von P ins höchste Bit von A

Beispiel: (3 * 2)

P	A	B
0000	0011	0010
0010	0011	0010
0001	0001	0010
0011	0001	0010
0001	1000	0010
0001	1000	0010
0000	1100	0010
0000	1100	0010
0000	0110	0010

Nach n Durchläufen steht das Ergebnis jetzt in den Registern A und P, es lautet also: 00000110 (und das enstpricht 6).

Division

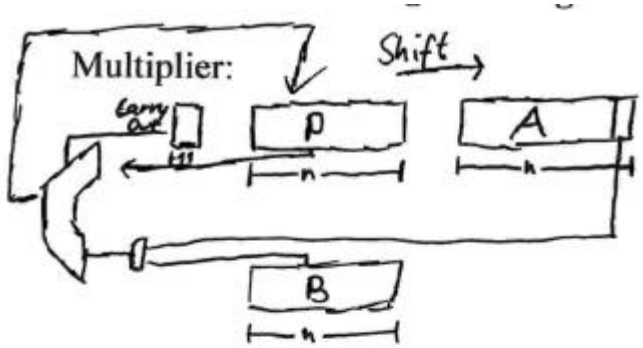
Algorithmus: Initalisiere P mit 0
Wiederhole n mal (n Länge der Faktoren)
(1) Shift Left mit dem höchsten Bit von A ins niedrigste Bit von P
(2) Subtrahiere B von P und schreibe das Ergebnis in P zurück
(3) Wenn das Resultat negativ ist, setze das niedrigste Bit von A gleich 0, sonst gleich 1
(4) Wenn das Resultat negativ ist, hole das alte Ergebnis von P zurück nach P; durch Addition von B

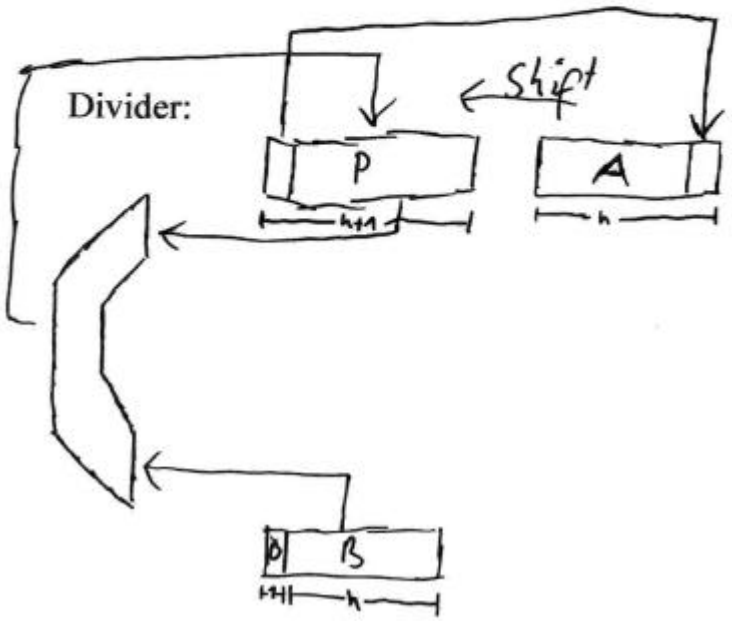
Beispiel: (14 : 3)

P	A	B
0000	1110	0011
0001	110_	
-0011		
-0010	1100	
0001		
0011	100_	
-0011		
0000	1001	
0000		
0001	001_	
-0011		
-0010	0010	
0001		
0010	010_	
-0011		
-0001	0100	
0010		

Das Ergebnis lautet: 0100 Rest 0010 (also 4 Rest 2), das wie man leicht nachrechnen kann völlig richtig ist.

Hier jetzt noch einmal die Unsigned Integer Multiplikation und Division als Hardware Diagramm dargestellt:





Multiplikation & Division bei Signed Integers im Zweier-Komplement: Da die Multiplikation und Division von Integers mit Vorzeichen nocheinmal um ein vielfaches komplizierter sind, zeige ich hier nur einmal ein Beispiel für einen Algorithmus für die Multiplikation, da diese wenigstens noch etwas weniger aufwendig ist als die Division.
Eine einfache Methode wäre zwar Zahlen erstmal in positive unsigned Integers umzuwandeln und dann später die Vorzeichen zu prüfen, nur wäre das sehr rechenintensiv und von daher uneffizient. Ich zeige jetzt erstmal den sogenannten BOOT51:

Booth's Algorithmus: Initalisiere A mit 0, Q_{-1} mit 0,
M mit dem Muliplikant und
Q mit dem Multiplikator
Wiederhole n mal:
(1) Wenn Q_0 gleich 1 und Q_{-1} gleich 0, dann subtrahiere M von A und schreibe das Ergebnis in A zurück
(2) Wenn Q_0 gleich 0 und Q_{-1} gleich 1 dann addiere M zu A und schreibe das Ergebnis in A zurück
(3) Shift Right: A -> Q -> Q_{-1}
$A_{n-1} = A_{n-2}$

Beispiel: (3 * 7)

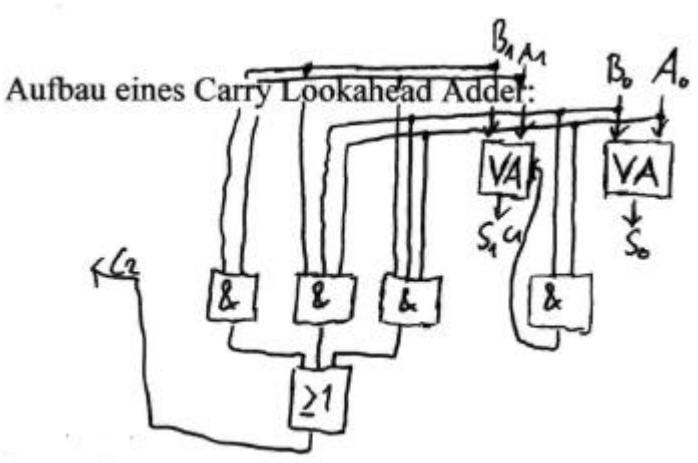
A	Q	Q_{-1}	M
0000	0011	0	0111
1001	0011	0	
1100	1001	1	
1110	0100	1	
0101	0100	1	
0010	1010	0	
0001	0101	0	

Das Ergebnis steht in den Registern A und Q und lautet daher: 00010101 (21), was wiederum richtig ist.

Speeding Up Interger Operations: Zum Schluß will ich noch einmal darauf eingehen, wie es möglich ist die Integer Arithmetik effizienter zu machen. Dies ist ein sehr wichtiger Punkt, da wir später im Abschnitt Floating Point Arithmetik noch sehen werden, das auch diese auf die Intergers zurückgeführt wird. Eine besonders wichtige Rolle bei der effizienteren Gestaltung spielt die Addition, da diese immer wieder eine wichtige Rolle spielt, bei allen Operationen. Daher nun einige Möglichkeiten zur effizienteren Gestaltung der Addition:

Eine Möglichkeit die Addition schneller zu machen ist es einen Carry Lookahead Adder anstelle eines Ripple-Carry Adder zu benutzen. Dieser bringt zwar einen größeren Aufwand an Hardware mit sich, ist dafür aber auch wesentlich schneller. Hier der Aufbau:

A, B: Eingangs Bits
S : Summen Bits
C : Übertrag



Es gibt noch andere Arten von Addierwerken, die jedes auf seine Art, Vorteile mit sich bringen. Es geht bei der Auswahl darum, einen schellen aber auch nicht zu teueren (Hardware Aufwand) Weg zu finden. Zwei weitere Arten von Addierwerken wären zum Beispiel:

Carry – Skip Adder : Ist eine Mischung aus dem Ripple-Carry Adder und dem Carry Lookahead Adder. Die Addition wird hierbei in einzelne Blöcke

zerlegt, wobei der Übertrag immer parallel zum jeweiligen Block berechnet wird.

Carry Select Adder : Die Addition wird in 2 Blöcke zerlegt, die parallel berechnet werden. Der 2. Block wird einmal für den Eingangs-Übertrag mit 1 und einmal mit 0 berechnet und am Ende des ersten Blockes wird bestimmt welches Ergebnis man benutzt. Dies hört sich etwas komplizierter an, aber wer interessiert ist, sollte sich geeignete Fachliteratur besorgen, um die genauen Vorgänge zu studieren.

Was lässt sich außerdem verbessern? Als letztes möchte ich noch erwähnen, das sich natürlich nicht nur die Addition verbessern lässt, sondern es auch wesentlich effizientere Algorithmen für Muliplikation, Division usw. gibt, die dann aber auch etwas komplizierter sind. Kein moderner Rechner arbeitet nach den hier beschriebenen Algorithmen, aber sie sind halt gut dafür geeignet, die Problematik zu beschreiben und ein leichtes Grundverständnis zu vermitteln.

Gleitkomma Arithmetik

Bedeutung der Gleitkommadarstellung

Festkommazahlen stoßen bei vielen Anwendungen, vor allem im mathematisch-naturwissenschaftlichen Bereich, an die Grenze ihrer vernünftigen Verwendbarkeit. Hier entsteht oft das Problem, daß große Zahlenbereiche überdeckt werden, jedoch eine Zahlengenauigkeit von einigen Dezimalstellen genügt. Handelsübliche Taschenrechner stellen Dezimalzahlen im Bereich von bis mit einer Genauigkeit von 8 oder 10 Dezimalstellen dar. Die dargestellten Dezimalzahlen setzen sich zusammen aus einer gebrochenen Mantisse und einem ganzzahligen Exponenten zur Basis 10, beide Größen sind multiplikativ verknüpft. Durch die Wahl der Basis 10 für den Exponenten, das ist gleichzeitig die Basis des Zahlensystems, entspricht eine Änderung des Exponenten um 1 einer Verschiebung des Kommas in der Mantisse um eine Stelle, wie das Beispiel zeigt.

Diese als **wissenschaftliche Schreibweise** bekannte Darstellung erlaubt eine höchst flexible Zahlenbehandlung bei recht geringem Berechnungs- und Anzeigeaufwand. Mit beliebig hohem Aufwand ist es natürlich prinzipiell möglich, den oben beschriebenen Wertebereich auch in Festkommadarstellung zu erreichen, in der Praxis wird jedoch das Zahlenformat schnell unhandlich groß, und zudem ist die Genauigkeit auf den niedrigen Stellen unnötig. Es ist deshalb sinnvoll, für die duale Zahlenverarbeitung ein Format ähnlich der wissenschaftlichen Schreibweise zu verwenden, das die Genauigkeit der Zahl und den zu überdeckenden Wertebereich frei wählbar macht. Man benutzt eine Zahlendarstellung, die neben der Ziffernfolge eine Größenordnungsangabe (Maßstabsfaktor) enthält, die sogenannte **Gleitkommadarstellung**. Die oben eingeführte wissenschaftliche Schreibweise ist der Sonderfall der Gleitkommadarstellung für Zahlen zur Basis 10.

Allgemein gilt: Jede Zahl *Z* eines beliebigen Zahlensystems kann in die folgende halblogarithmische Darstellung gebracht werden

M ist dabei die **Mantisse**, eine gebrochene Zahl kleiner Eins. Die ganze Zahl *B* ist die Basis des verwendeten Zahlensystems und *E* ist der **Exponent**.

Beispiel:

432,5 = 0,4325 * 10^3

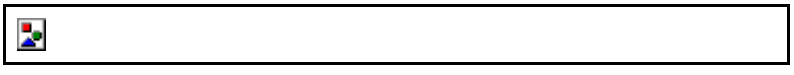
0,0004325 = 0,4325 * 10^-3
(^ : Parameter danach ist exponent)

Die Bezeichnung Gleitkommadarstellung resultiert aus dem variablen Zahlenbereich, den die Mantisse zusammen mit dem **Maßstabsfaktor** □ überstreichen kann. Für die Darstellung von Gleitkommazahlen im Rechner ist die Basis *B* durch das im Rechner verwendete Zahlensystem (Dualsystem) fest vorgegeben. Da die Basis bekannt ist, brauchen rechnerintern nur die Mantisse *M* und der Exponent *E* abgespeichert zu werden. Ein Wort ist für die Zahlendarstellung in zwei Teile zu unterteilen:

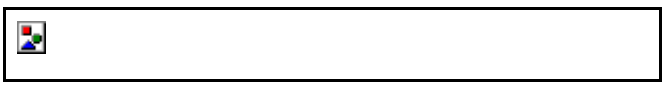
1. Mantissenbereich, der den Zahlenwert der Mantisse *M* enthält
2. Exponentenbereich, der den Zahlenwert des Exponenten *E* angibt

Das bedeutet, daß im Gleitkommaformat weniger Ziffern einer Zahl dargestellt werden können als bei der Festkommadarstellung mit gleicher Wortlänge. Dieser Nachteil wird jedoch in vielen Fällen durch die bessere Ausnutzung der Stellen (keine unnötigen führenden Nullen) wieder ausgeglichen.

Da auch negative Zahlen und Zahlen mit sehr kleinen Beträgen (□) dargestellt werden müssen, sind Mantisse und Exponent vorzeichenbehaftet. Sie können in der bereits bekannten Art als Zweierkomplementzahl dargestellt werden. Um beim Exponenten die Angabe eines Vorzeichens einzusparen, wird im allgemeinen eine sogenannte **Charakteristik** *C* eingeführt. Zum Exponenten *E* wird eine Konstante □ addiert, so daß die Summe aus Exponent und □ immer größer oder gleich Null wird. Diesen so modifizierten Exponenten nennt man die Charakteristik *C*.



Damit erhält man für die Gleitkommazahl *Z*



Beispiel: *Gleitkomma-Darstellung*

Reserviert man bei der Darstellung einer Gleitkomma-Dezimalzahl 2 Stellen für die Charakteristik *C*, so erhält man für *C* den Bereich:

Wird die Konstante gewählt, so ergibt sich für den Exponenten:

Damit ergeben sich folgende Zahlendarstellungen

10,2471 ***** 10^2 = 0,102471 ***** 10^4
Das entspricht einer Mantisse (M) von 0,102471 und einer Charakteristik ($C = E+K_0$) von 54.

0,000572 ***** 10^-1 = 0,572 ***** 10^-4
Das entspricht einer Mantisse von 0,572000 und einer Charakteristik von 46.
(^ : Parameter danach ist exponent)

Für die Darstellung von Gleitkommazahlen muß noch eine Vereinbarung getroffen werden, um die Zahlen auch eindeutig wiedergeben zu können. Eine Zahl läßt sich als Produkt ja auf verschiedene Weise schreiben:

z.B. .

Fordert man, daß die erste Ziffer nach dem Komma von Null verschieden und die Ziffern vor dem Komma identisch Null sein müssen, so ist nur noch die Darstellung möglich. Diese Darstellung führt zu einer optimalen Ausnutzung der für die Mantisse verfügbaren Stellen. Man bezeichnet dies als **normierte Darstellung** der Mantisse. Der Wertebereich der Mantisse in normierter Darstellung ist damit festgelegt. Es gilt:

Abweichend von dieser Normalisierungsfestlegung wird die Zahl 0 durch die Mantisse und beliebigen Exponenten dargestellt. Normalisieren bedeutet im Dualsystem, daß die 1. Stelle nach dem Komma (der Mantisse), die Binärstelle mit der Wertigkeit eine 1 enthält. Der Betrag der Mantisse $|M|$ ist also immer . Bei negativen Zahlen wird die Mantisse im *B*-Komplement dargestellt. Im Dualsystem enthält somit die 1. Binärstelle vor dem Komma bei positiven Zahlen eine 0 und bei negativen Zahlen eine 1. Die Mantisse stellt ein normiertes Festkommawort dar. Eine schematische Gleitkommaformatdarstellung hat die folgende Form:

Mantisse	Exponent
X,XXXX...	**XXXXX**
(m Stellen)	(e Stellen)

Beispiel: *Gleitkomma-Darstellung*

Darstellung im dualen Gleitkommaformat mit , mit .

a) positive Zahl

Damit ist der Exponent E bekannt

Darstellung im gegebenen Format:

b) negative Zahl

wegen *folgt die folgende*
Darstellung im gegebenen Format:

Die Verschiebung des Kommas um eine Stelle nach links oder rechts bedeutet eine Veränderung des Exponenten um 1. **Bei negativen Zahlen wird die Zahl zuerst normalisiert und dann ins 2er-Komplement gewandelt**.

Der wesentliche Vorteil der Gleitkommadarstellung gegenüber der Festkommadarstellung ist wie schon gesagt der viel größere darstellbare Zahlenbereich. Für ein allgemeines Gleitkommaformat mit *m* Stellen Mantisse und *e* Stellen Exponent gelten die folgenden charakteristischen Größen. Dabei wird angenommen, daß die Mantisse im 2er-Komplement dargestellt und normalisiert ist. Für die Konstante *K* wird angenommen, so daß der Zahlenbereich des Exponenten zur Hälfte in positive und negative Zahlen geteilt ist.

1. kleinste positive darstellbare Zahl (normiert)
2. größte positive darstellbare Zahl
3. betragsmäßig kleinste negative darstellbare Zahl
4. betragsmäßig größte negative darstellbare Zahl
   Ihr Betrag ist um größer als die größte positive Zahl
   (wesentliches Merkmal des 2er-Komplements)
5. größte Genauigkeit = Differenz zwischen der kleinsten und zweitkleinsten unnormierten darstellbaren Zahl

Dezimal-Dual-Wandlung im Gleitkommaformat

Die Wandlung von Dezimalzahlen in ein duales Gleitkommaformat ist etwas aufwendiger als bei Festkommazahlen. Sie kann auf verschiedene Arten durchgeführt werden. Im folgenden wird eine Methode gezeigt.

Wandlungsverfahren
Die Wandlung erfolgt durch Umweg über ein Festkommaformat. □für Festkommazahlen beschriebenen Regeln angewendet um die Zahl in ein Festkommaformat zu wandeln. Anschließend wird dieses Wandlungsergebnis durch Wahl eines geeigneten Exponenten in das vorgegebene Gleitkommaformat umgeformt. Dies erfordert eine Verschiebung des Kommas mit einer entsprechenden Anpassung des Exponenten.

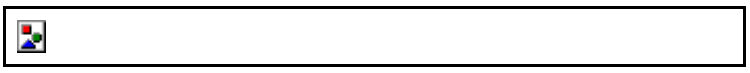
Beispiel: *Wandlungsverfahren*

Gesucht sei zur Zahl □*die duale Gleitkommadarstellung im Format*

mit der Konstanten □.
Es gilt: □
Eine Verschiebung des Kommas um 23 Stellen nach links bedeutet E=23. Damit ergibt sich für die Charakteristik C: □
Die Darstellung im Format ergibt somit: □.
Für den absoluten Wandlungsfehler F in diesem Format gilt:

□

Der relative Wandlungsfehler f ist:



Der Nachteil dieses Verfahrens ist, daß insbesondere bei betragsmäßig großen Exponenten die Übergangsdarstellung im Festkommaformat länglich und damit rechenaufwendig werden kann.

Arithmetik im Gleitkommaformat

Bei den Grundrechenarten in der Gleitkommadarstellung müssen beide Teile des Wortes (Mantisse, Charakteristik) getrennt verarbeitet werden, was die Operationsabläufe etwas aufwendiger macht. Da man sowohl die Mantisse als auch die Charakteristik für die getrennte Verarbeitung als Festkommazahlen betrachten kann, treten keine wesentlich neuen Überlegungen bei der Gleitkommaverarbeitung auf.

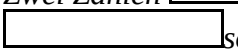
Addition und Subtraktion

Es ist klar, daß nur die Mantissen zweier Zahlen mit gleicher Charakteristik addiert bzw. subtrahiert werden können. Zwei Zahlen im Gleitkommaformat werden *addiert* (voneinander *subtrahiert*), indem die folgenden Operationen ausgeführt werden:

1. Durch Linksverschiebung des Kommas wird der Operand mit der kleineren Charakteristik so umgeformt, daß beide Operanden die gleichen (größeren)

Charakteristiken haben. Der umgeformte Operand ist dann natürlich nicht mehr normalisiert.
2. Die Operation wird durchgeführt, indem die Charakteristik erhalten bleibt und die beiden Mantissen addiert werden (Addition) bzw. das Zweierkomplement des zweiten Operanden addiert wird (Subtraktion).
3. Das Ergebnis wird durch Verschiebung des Kommas, also durch Wahl der geeigneten Charakteristik normalisiert und in das gegebene Format gebracht. Dabei können Abweichungen vom richtigen Ergebnis durch die Einschränkungen des Ergebnisformats auftreten.

Beispiel: *Addition und Subtraktion im Gleitkommaformat*

Zwei Zahlen  *und*  *im Format*  *sollen addiert bzw. subtrahiert werden.*

Addition 

Ergebnis im Format und normalisiert: 

Subtraktion 

Ergebnis im Format und normalisiert: 

Multiplikation und Division

Zwei Zahlen im normalisierten Gleitkommaformat werden miteinander *multipliziert* (durcheinander *dividiert*), indem die folgenden Operationen ausgeführt werden:

1. Die Mantissen der Operanden werden miteinander multipliziert (durcheinander dividiert). Bei negativen Mantissen ist es sinnvoll, die Beträge zu multiplizieren (zu dividieren) und eine getrennte Vorzeichenbetrachtung durchzuführen.
2. Die Exponenten der Operanden werden addiert (bei Multiplikation) bzw. das Zweierkomplement des zweiten Operanden zum ersten addiert (bei Division). Da die Zahlendarstellung mit Charakteristik statt Exponent verwendet wird, sind entsprechend die Charakteristiken zu addieren (zu subtrahieren). Dabei wird jedoch

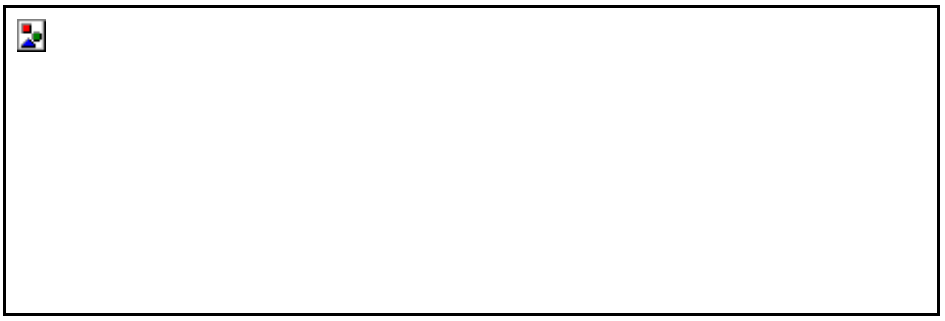
die Konstante zuviel addiert (subtrahiert), was anschließend wieder rückgängig gemacht werden muß.

3. Das Ergebnis wird durch Verschiebung des Kommas, also durch Wahl der geeigneten Charakteristik *C*, normalisiert und in das gegebene Format gebracht. Dabei können Fehler (Abweichungen vom richtigen Ergebnis) durch Einschränkungen des Ergebnisformats auftreten.

Beispiel: *Multiplikation und Division im Gleitkommaformat*

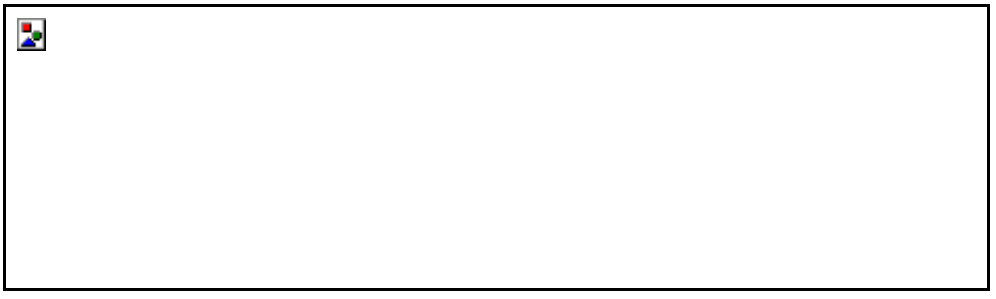
Zwei Zahlen  *und*  *im Format* *sollen multipliziert bzw. durcheinander dividiert werden.*

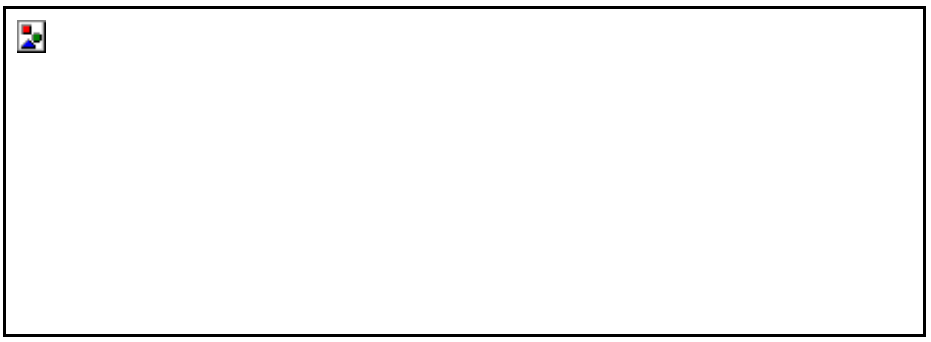
Multiplikation



Ergebnis im Format und normalisiert:

Division





Ergebnis im Format und normalisiert: