Kapitel 4 Rechenwerke

Ein Rechenwerk besteht aus

- Rechnenden Schaltnetzen oder -werken
- Registern
- internen Bussen.

Man unterscheidet Festkomma- und Gleitkommarechenwerke.

4.1. Festkommarechenwerke

Der Inhalt der Register wird hier als Bitvektoren oder als integer-Zahlen interpretiert.

4.1.1. Operationen auf Ganzzahlen

Die 32 Bit eines Wortes (hier beispielhaft benutzt) werden interpretiert als integer-Zahl im 2-Komplement. Dabei ist $-z = \neg z + 1$.

Der Zahlenbereich ist $-2^{31} \leq z \leq (2^{31} - 1)$

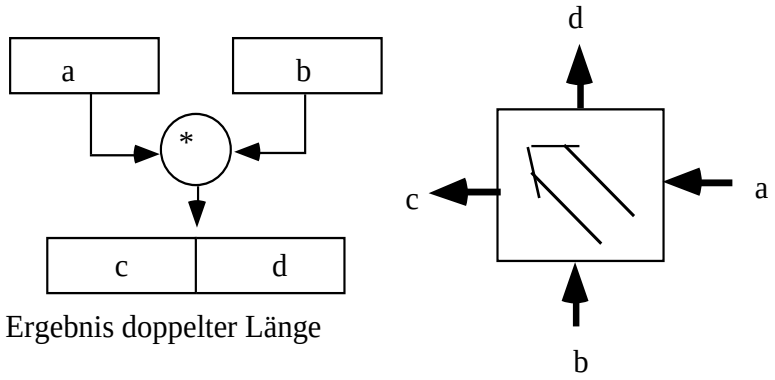
d. h. rd. $-2 \cdot 10^{-9} \leq z \leq +2 \cdot 10^{9}$.

Übliche Operationen in einer arithmetisch-logischen Einheit (ALU) sind **Addition** und **Subtraktion**, realisiert durch Addiernetze (Paralleladdierer) mit Übertragsverrechnung (s. Rechnerorganisation).

Die Subtraktion wird zurückgeführt entweder

- auf 2-Komplementbildung und Addition im Addiernetz
- oder auf Umschalten innerhalb des Addiernetzes auf Subtrahieren.

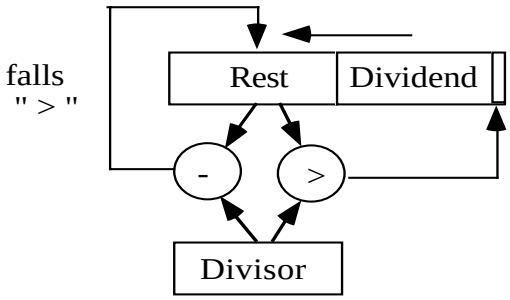
Die **Multiplikation** wird ebenfalls auf Parallelmultiplizierern realisiert. Ein Multiplizierer für zwei n-Bit Zahlen braucht $n * n$ Zellen aus einem einfachen Schaltnetz mit einem Volladdierer.



Ein $32 * 32$ - Bit Parallelmultiplizierer ist mit 1024 Zellen zu realisieren. Durch den regelmäßigen Aufbau problemlos integrierbar.

Laufzeit $n *$ (Laufzeit durch eine Zelle) < 200 ns.

Die **Division** ist schlecht mit Schaltnetzen zu realisieren.
Man greift auf spezielle Schaltwerke zurück (s. Rechnerorganisation).



1. lade Dividend
2. schiebe
3. vergleichen, ggf. subtrahieren
4. n-mal wiederholen
5. Ergebnis ist Dividend

Relationaloperatoren wie die Überprüfung auf $=, \neq, <, >, \leq, \geq$ sind als Maschinenbefehle verfügbar (s. DLX-Beispiel), tatsächlich realisiert durch direkte Schaltnetze in der ALU meist nur die Überprüfung auf:
- negatives Ergebnis (Vorzeichenbit des Ergebnis)
- Null (eigenes Schaltnetz)
- Überlauf (ein Bit im Addierer) .

4.1.2. Operationen auf Bitfolgen

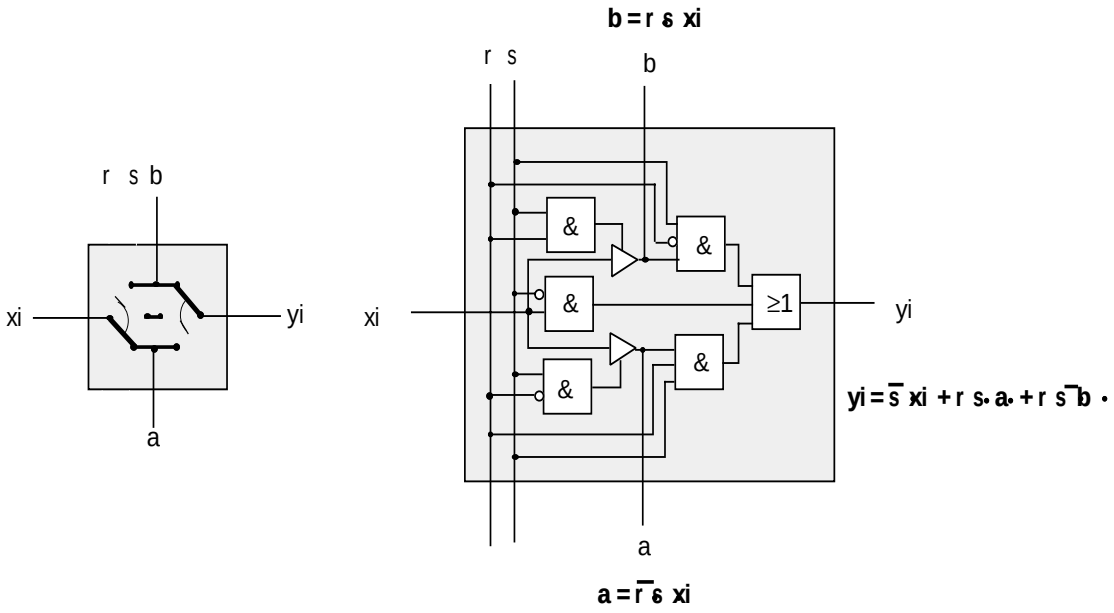
Es werden die Registerinhalte als Array of Boolean behandelt und bitweise verknüpft.

- Laden einer Konstanten
- Laden der leeren Folge (0)
- Negieren (Komplementbildung)
- Dyadische Operationen (AND, OR, NAND, NOR, XOR, ÄQUIV)
- Schiebeoperationen.

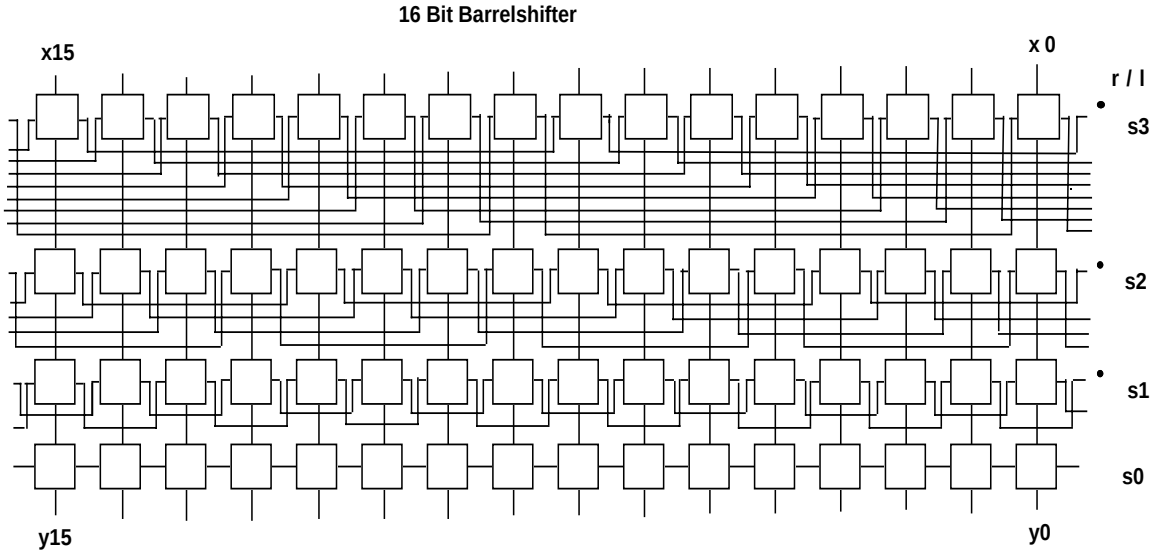
Die Schiebeoperationen werden entweder mit einem **Schiebegatter** realisiert (s. Rechnerorganisation), mit dem man um 1 Bit rechts oder links schieben kann, oder mit einem **Barrelshifter**, mit dem man ein n-Bit Wort um 0 - (n - 1) Bit nach rechts oder links verschieben kann und mit dem sich noch steuern läßt, was nachgeschoben werden soll:

- beim Schieben nach rechts das Vorzeichenbit oder Null oder zyklisch die rechts herausfallenden Bits wieder hereingebracht werden sollen;
- beim Schieben nach links Nullen oder zyklisch die vorne herausfallenden Bits (meist wird das zyklische Schieben nicht realisiert).
  Realisiert wird ein Barrelshifter durch Zellen, die das Durchschalten eines Bits in drei Richtungen erlauben: rechts, links oder geradeaus, gesteuert durch zwei Bits.

Si	r	
0	0/1	Schalte direkt durch nach rechts/links
1	0/1	Schiebe um 2^i Bit nach rechts/links



Diese Zellen werden zu einem Barrelshifter zusammengeschaltet, wobei die Schwierigkeit die vielen Verbindungsleitungen sind, die auf dem Chip viel Platz fortnehmen.



Die Schaltung zum Nachschieben von Null, Eins oder den zyklisch herausfallenden Werten ist nicht mit angegeben. Der Übersichtlichkeit halber ist hier ein 16-Bit-Barrelshifter gezeigt. Der Barrelshifter ist nach beiden Seiten hin verlängerbar. Das kann benutzt werden, um ein Wort als Ganzes um ±n Bit zu verschieben.

4.2. Gleitkommarechenwerk

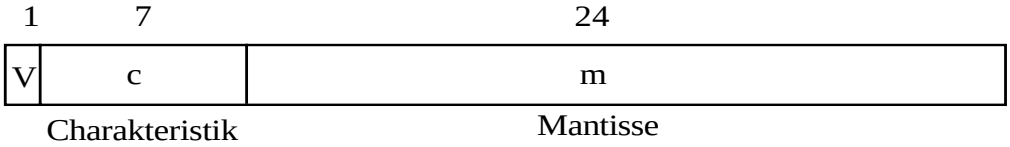
Es wird als Schaltwerke realisiert werden, da schon die elementaren Operationen mehrere Schritte benötigen.

Häufig ist es als eigene floating point unit (FPU) realisiert.

Die Zahlendarstellung im Speicher kann verschiedene Formate haben und Zahlen als Bitstrings verschiedener Länge darstellen.

4.2.1. IBM-Format

Die Wortlänge sind 32 Bit mit einer Aufteilung wie folgt



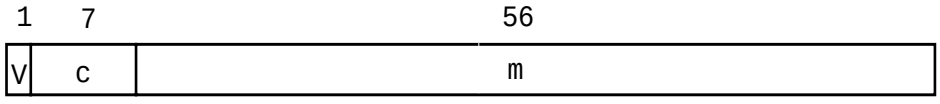
7 Bit sind neben dem Vorzeichen V als **Charakteristik** c ausgewiesen mit der halblogarithmischen Darstellung einer Zahl

$$\mathbf{z = \pm m \cdot b^e} \quad \text{mit} \quad \mathbf{c = e + e_0} \; ; \; \mathbf{e_0 = 64} \; .$$

Die Basis b wird als 16 festgelegt und die Mantisse üblicherweise **normiert**: $1 > m \geq 1/b$.

Bei einer Mantissenlänge von 24 Bit = 6 x 4 Bit ist die Maschinengenauigkeit
$1/2^{24} = 16^{-6+1} \approx 0.9 \cdot 10^{-6}$, d. h. 6 Dezimalen.

Da der maximale Exponent e = 63 ist, ergibt sich ein Zahlenumfang von $16^{\pm 63} \approx 10^{\pm 75}$. Um die Genauigkeit zu erhöhen, benutzt man doppelt lange GK-Zahlen (double precision) mit einer Mantissenlänge von 56 Bit. Damit erreicht man Genauigkeiten von 15 Dezimalen.

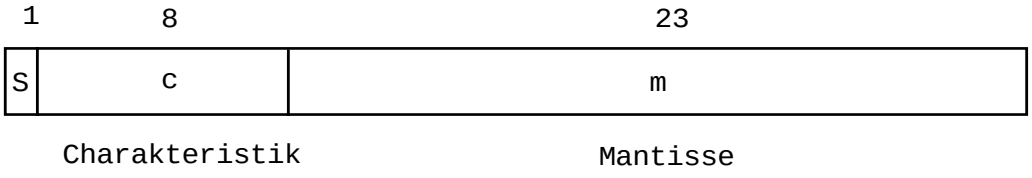


Eine Sonderreglung wird für die Darstellung von Null vereinbar: abweichend von der Normierung der Mantisse gilt wegen $0 = \pm 0 \cdot b^e$ $\quad m = 0 \Rightarrow z = 0$.

Das Bitpattern 0 ... 0 stellt die Zahl Null dar, aber auch jedes Pattern x ... x 0 ... 0 mit m = 0 .

4.2.2. IEEE - 754 Standard

Inzwischen hat sich für die Darstellung von GK-Zahlen der IEEE-Standard durchgesetzt.



Es ist

$$\mathbf{z = (-1)^s * (1.0 + 0.m) \cdot 2^{c\,-127}} \text{ mit } \mathbf{0 \le m \le (1 - 2^{-23})}$$

Die normalisierte Mantisse m hat nur 23 Bit, das 24. Bit, der **Signifikand** ist implizit (hidden Bit).

Null wird dargestellt durch $m = 0$ und $c = 0$
=> das 32-Bit-Wort 0 ... 0 = 0 .

Ungültige Zahlen werden gekennzeichnet mit (NaN - not a number) $m \neq 0$, $c = 255$.

Ein Überlauf entsprechend $z = \pm \infty$ wird signalisiert durch $m = 0$, $c = 255$.

Eine nicht normalisierte Gleitkommazahl (Underflow) ist $m \neq 0$, $c = 0$
d. h. $0 < z < (1 + 0.m) \cdot 2^{-127}$.

Der Wertebereich für $z > 0$ ist

$$1 * 2^{-127} \le z \le (2 - 2^{-23}) * 2^{+128}$$

dezimal gerundet entspricht das

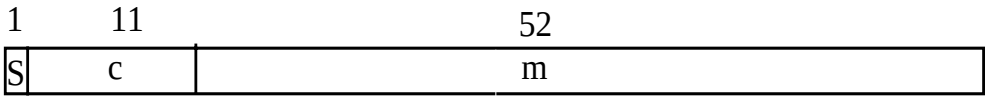
$$1.17 * 10^{-38} \le z \le 3.4 \cdot 10^{+38} .$$

Bei der Genauigkeit der Darstellung der GK-Zahlen muß man unterscheiden:

- absolute Genauigkeit:
  bei $\Delta m = 2^{-23}$ ist bei $z \approx 2^{128}$ (maximale Zahl)
  $\Delta z = 2^{127} (2 - 2^{-23}) - 2^{127} (2 - 2^{-22}) = 2^{104} \approx 2 \cdot 10^{31}$

  bei $\Delta m = 2^{-23}$ und $z = 1.1 * 2^{-126}$ (kleinste positive Zahl) ist
  $\Delta z = 1.2 \cdot 2^{-126} \cdot 2^{-23} = 1.2 \cdot 2^{-149} \approx 1.7 \cdot 10^{-45}$
- relative Genauigkeit:
  Sie ist in beiden Fällen gleich:
  $\Delta m = 2^{-23} \approx 1.19 \cdot 10^{-7}$.

  Für höhere Genauigkeiten stellt man GK-Zahlen mit 64 Bit dar:



Es ist $z = (-1)^s * (1 + 0.m) * 2^{(c\ -1023)}$.

Der Zahlenbereich umfaßt das Intervall

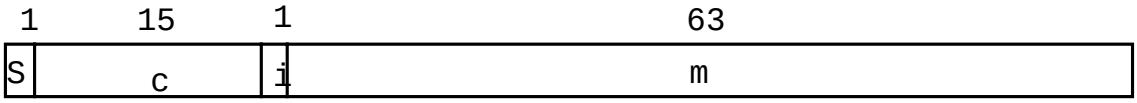
$2.3 * 10^{-308} \leq z \leq 1.7 * 10^{+308}$.

Wieder gibt es Sonderfälle:

c = 0	m = 0	stellt Null dar
c = 0	m ≠ 0	ist eine denormalisierte Zahl
c = 2047	m = 0	stellt $z = \pm \infty$ dar
c = 2047	m ≠ 0	ist keine gültige Zahl (NaN - not a number)

Die relative Genauigkeit ist $2^{-52} \approx 15$ Dezimalen.

Maschinenintern, d. h. in den inneren, dem Programmierer nicht sichtbaren Registern des GK-Rechenwerks, wird nach dem Standard mit 80-Bit-langen Worten gerechnet.



Die Kennung i unterscheidet normalisierte und denormalisierte Zahlen:

i = 1	$z = (-1)^s * (1.m) * 2^{(c\ -16383)}$	(normalisiert)
i = 0	$z = (-1)^s * (0.m) * 2^{(c\ -16383)}$	(denormalisiert)

Null wird dargestellt als i = 0, c = 0, m = 0 .

Die große Charakteristik vermeidet Überläufe bei fortgesetzten Multiplikationen/Divisionen und die lange Mantisse beugt der Gefahr von Auslöschung bei der Subtraktion annähernd gleicher Zahlen vor, kann sie aber nicht beseitigen.

4.2.3. Rechnen mit GK-Zahlen

Multiplikation

Es ist dies eine unkritische Operation:

Sei $z_1 = m_1 \cdot b^{e1}$; $z_2 = m_2 \cdot b^{e2}$

=> $\mathbf{z_1 \cdot z_2 = m_1 \cdot m_2 \cdot b^{(e1 + e2)}}$.

Das Entstehen einer doppelt langen Mantisse wird durch verkürzte Multiplikation unterbunden, bei der das Ergebnis nur wieder die Länge von m_1 oder m_2 hat.

Bei Darstellung in IEEE-Standard muß vor der Multiplikation der Mantissen der Signifikand (führende 1) hinzugefügt werden: $(1.m_1)_2 \cdot (1.m_2)_2 = m_1 + m_2 + m_1 \cdot m_2 + 1$.

Der Überlauf bei der Multiplikation der Mantissen muß durch anschließendes Schieben behandelt werden:

$$1 + m_1 \,.\, m_2 + m_1 + m_2 \overset{!}{<} 2 \,.$$

Division

Bei der Division ist mit $z_1 = m_1 \cdot b^{e1}$ und $z_2 = m_2 \cdot b^{e2}$

$$\mathbf{z_1/z_2 = m_1/m_2 \cdot b^{e1-e2}} \,.$$

Die Division der Mantissen und Subtraktion der Exponenten (d. h. der Charakteristiken) ist unkritisch.

Im IEEE-Standard ist

$(1.m_1)_2 \geq (1.m_1)_2 / (1.m_2)_2 \geq (0.1\, m_1)_2$

d. h. man hat maximal die Mantisse um 1 nach links zu schieben (mit gleichzeitigem Dekrementieren der Charakteristik).

Addition / Subtraktion

Das Vorgehen erfordert 4 Schritte:

1. Vergleich der Exponenten; sei o.B.d.A. e1>e2
2. Schieben der Mantisse der kleineren Zahl um (e1 - e2) Bit.
3. Addition/Subtraktion der Mantissen
4. Ergebnis wieder normalisieren.

Dabei kann Auslöschung auftreten: bei der Addition von zwei unterschiedlich großen Zahlen geht die kleinere verloren, bei der Subtraktion annähernd gleicher Zahlen täuscht die anschließende Normalisierung eine nicht vorhandene Genauigkeit vor.

0.100 000	$\cdot\ 10^7$	0.123 456	$\cdot\ 10^7$
+ 0.123 456	$\cdot\ 10^3$	- 0.123 444	$\cdot\ 10^7$
0.100 012	$\cdot\ 10^7$	0.000 012	$\cdot\ 10^7$
		= 0.12? ???	$\cdot\ 10^3$

Gegen diese Fehler kann man durch Verwendung interner Schutzstellen (z. B. 63-Bit-Mantisse) versuchen anzugehen, aber immer noch liefert eine Rechnung der Form

$$(1 \cdot 10^{30} + 1) - 1 \cdot 10^{30} \;\rightarrow\; 1 \cdot 10^{30} - 1 \cdot 10^{30} = 0$$

während offensichtlich das Ergebnis 1 ist.

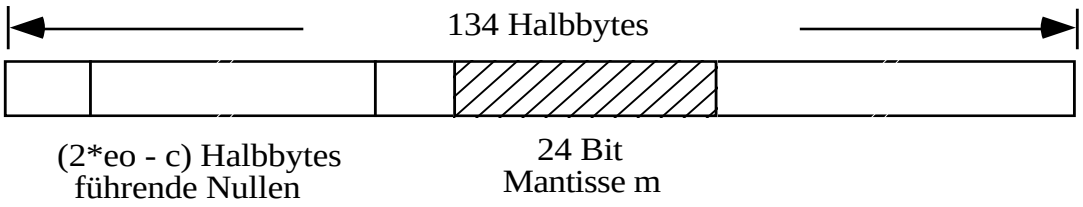
Solche Auslöschungen können insbesondere auftreten bei der Bildung von Skalarprodukten $y = \sum a_i\, b_i$ in schlecht konditionierten Gleichungssystemen, wo dann ggf. das Ergebnis von der Reihenfolge der Teilrechnungen abhängt.

4.2.4. Rechnen mit langem Akkumulator (Acryth-Rechner, IBM)

Es wird der gesamte Zahlenbereich der real-Zahlen intern auf ein langes Register abgebildet und dort als große ganze Zahl behandelt.

Sei der Zahlenbereich 16^{-64} ... 16^{+63} mit einer 24-Bit-Mantisse (IBM-GK-Darstellung), dann reicht ein Akkumulator von 4 x 128 + 24 = 536 Bit. In ihm wird eine real-Zahl z als große ganze Zahl interpretiert mit einem impliziten Faktor 16^{-70}.

$$z = m * 16^{c - eo} \qquad eo = 64$$



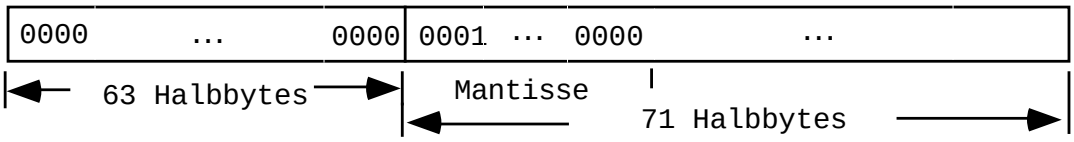
Z. B. $1 = 1/16 \cdot 16^{71} \cdot 16^{-70}$

Mantisse: 0001 0 ... 0

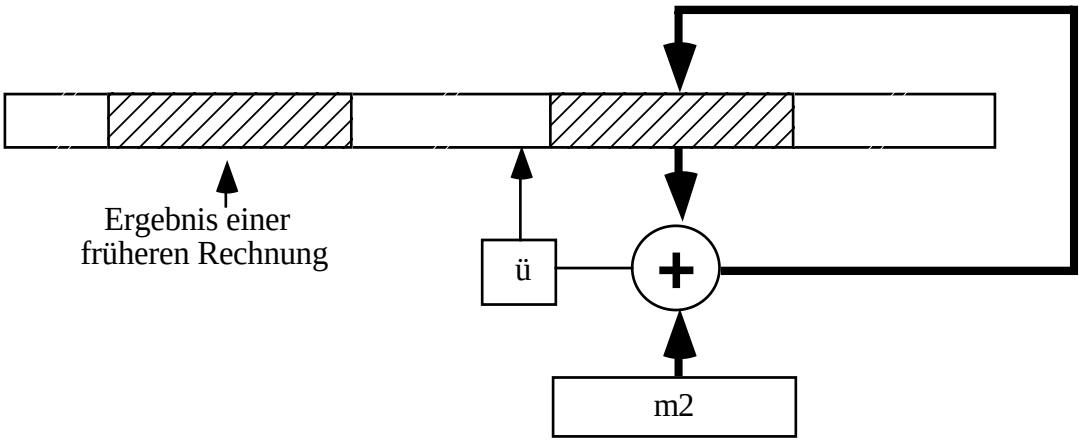
Charakteristik: 65 (= 64 + 1)

Im Akkumulator ist die Anzahl der führenden Halbbytes

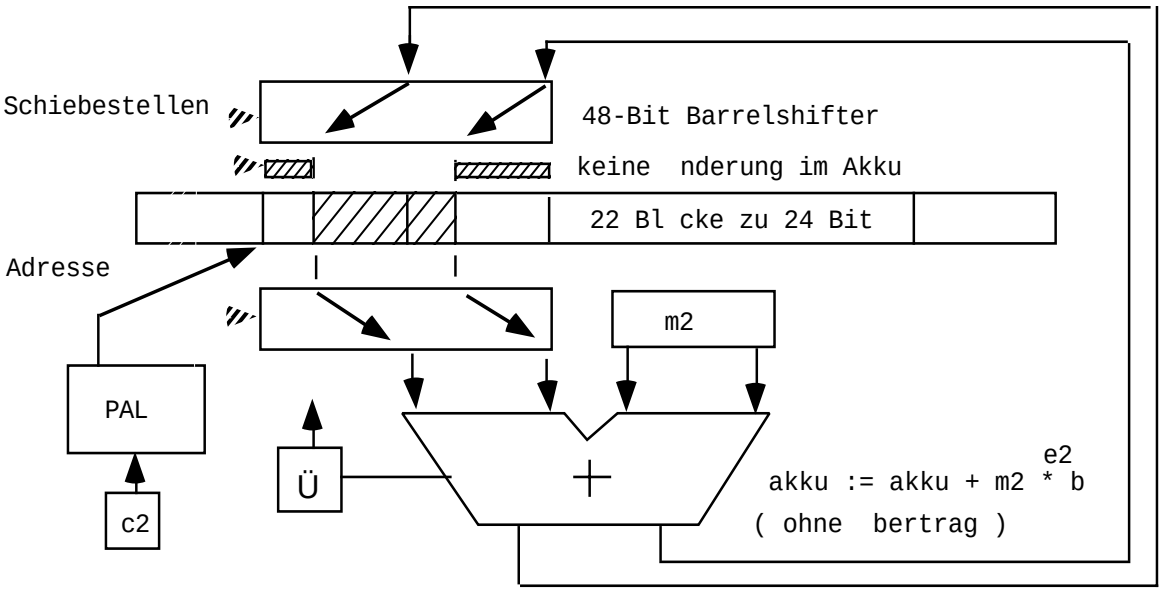
$2 * e_0$ - Charakteristik, hier 63 .



Addition/Subtraktion vom Akkumulator bezieht sich auf 24 Bit, wobei allerdings ein Überlauf durch den Akkumulator durchlaufen kann.



Wo die Addition angesetzt wird, bestimmt sich aus der Charakteristik und wird durch einen entsprechenden Zugriff auf dem Akkumulator geleistet, der als schneller Speicher M(i) zu 22 Worten à 24 Bit organisiert ist. Ausgelesen werden 48 Bit: M(i) ## M(i+1). Die Adresse bestimmt sich zu $(2 * e_0 - 1 - c)$ div 6 und das 48-Bit-Wort durchläuft Barrelshifter, in denen es um $4 * [(2\ e_0 - 1 - c) \bmod 6]$ Bit geschoben wird.



Dann haben auflaufende Rechnungen keine Rundungsfehler, schlecht konditionierte numerische Probleme bleiben handhabbar und eine exakte Fehlereingrenzung wird möglich.

Die o. g. Aufgabe ergibt $1 * 10^{30} + 1 - 1 * 10^{30} = 1$ wie gefordert, da das Zwischenergebnis im Akkumulator nicht angetastet wird.

4.3. Adressrechenwerk

Es gibt in Rechnern Adressrechnungen, die immer wieder vorgenommen werden müssen und durch eine Hardware unterstützt werden können:

- Inkrementieren des Befehlszählers: PC := PC + 1 (2, 4)
  oder eines Stackpointers: SP := SP + 1 (2, 4)

- Dekrementieren eines Stackpointers: SP := SP - 1 (2, 4)

  Die Konstante richtet sich nach der Wortlänge der Maschine und der Art des Zugriffs auf den Speicher, ob er byte- oder wortadressiert ist.

- Adressrechnung im engeren Sinne

  Hier ist von Interesse der Zugriff auf Matrizen von Werten. Ein Pointer in eine Matrix ist fortzuschalten.

  $$ADR := ADR + Q \cdot (IXR) + b$$

  oder ADR := ADR + z ; z := z + a .

  Damit lassen sich Scanpattern in der Matrix erzeugen.

4.4. Registerstrukturen

Neben den unbedingt nötigen Registern für die Befehlsfortschaltung

- Befehlszähler (PC) (Instruction-Pointer)
- Statusregister (SR) (Flagregister)
- Instruktionsregister (IR) (kann bei Programmen in einem ROM fortfallen)

und den Registern für das Ansprechen des Speichers

- Speicheradressregister (MAR) (data pointer, DP)
- Speicherdatenregister (MDR) (data register, DR)

braucht ein Rechner Register im Rechenwerk für das Zwischenspeichern von Ergebnissen.

Es sind dies die Register, die explizit durch Befehle angesprochen werden.

- Man hat mindestens ein Register, den Akkumulator (nur noch in primitiven 4-Bit-Rechnern zu finden).
- Ein Registersatz von 4, 8, 16, 32, ... Registern.

4.4.1. Ein Satz allgemeiner Register

Das sind allgemeine Register, die sowohl für Rechnungen als auch als Adressregister benutzt werden können.

Beispiel: IBM 360/370
16 allgemeine Register von Wortlänge 32 Bit, R0 ... R15;
R0 ist der PC .
Es reichen 4 Bit im Befehl als Adresse eines Registers.

Beispiel: DLX-Maschine, 32 allgemeine Register von Wortlänge.
Es werden 5 Bit gebraucht, um ein Register zu adressieren.

4.4.2. Separate Daten- und Adressregister

Gleiche Registeradressen in Befehlen sprechen verschiedene Register an je nach Art des Befehls:

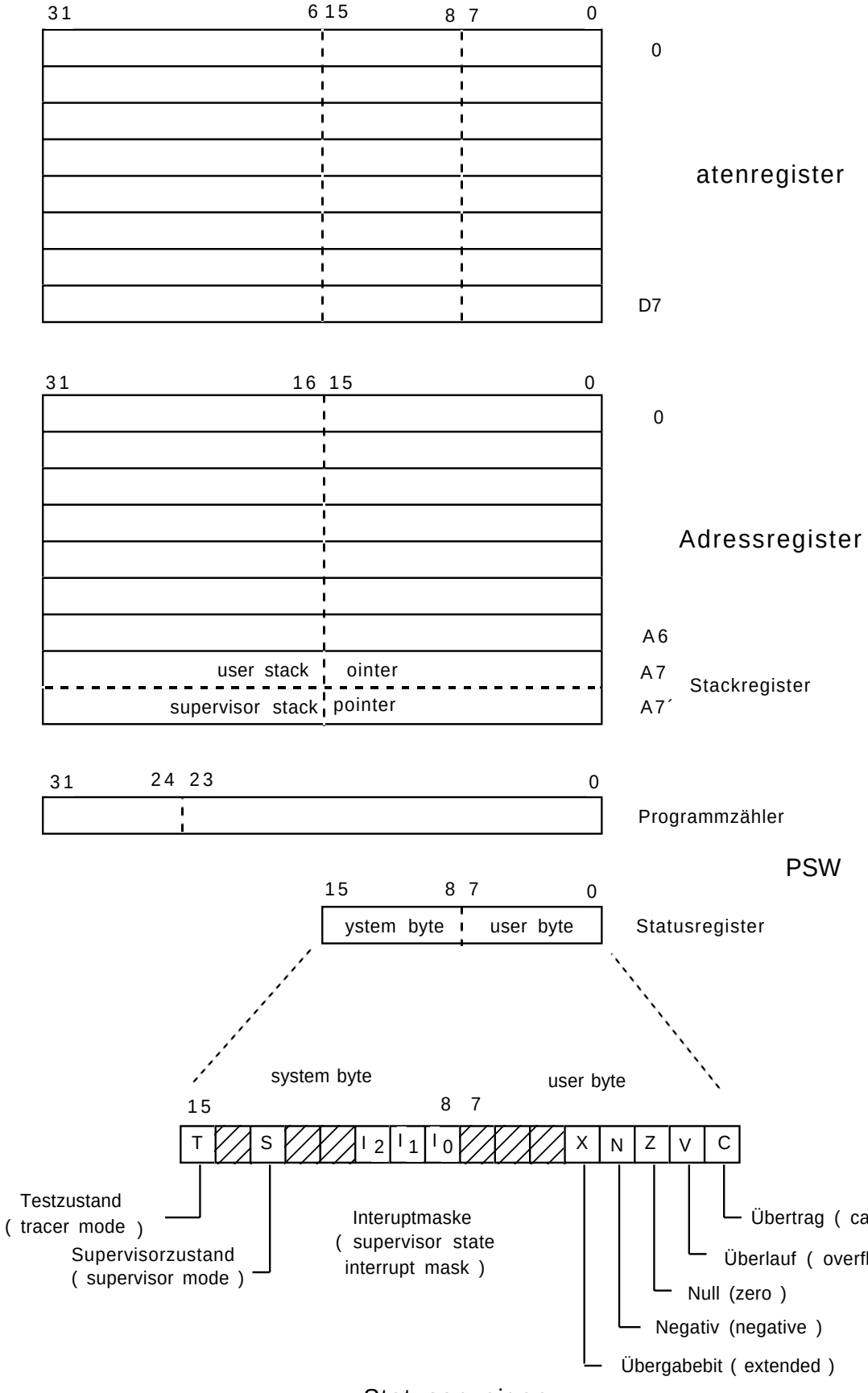
Datenregister in datenmanipulierenden Befehlen,
Adressregister bei Adressierungen des Speichers.

Man braucht keine längeren Registeradressen in Befehlen, dafür ist die Zahl der Register für eine Anwendung begrenzt.

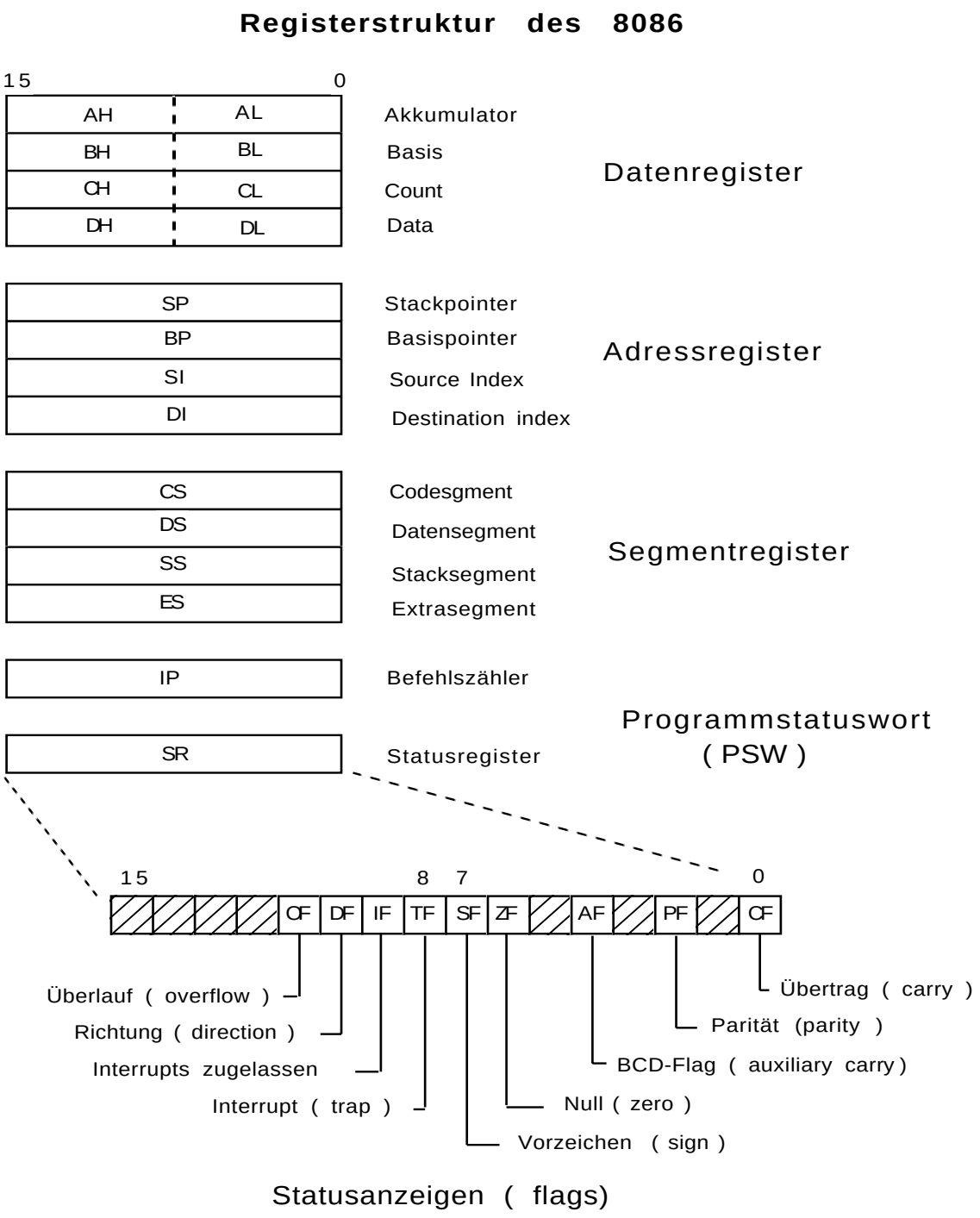
Beispiel: Motorola 68000 - Rechner

Es gibt 8 allgemeine Datenregister D0 ... D7 . Dazu kommen 8 Adressregister A0 ... A7 , von denen A7 (bzw. A7' im Supervisormode) als Stackpointer extra gekennzeichnet ist. Befehlszähler und Statuswort komplettieren den Registersatz.

Registerstruktur des 68000



Beispiel: Intel 80x 86, hier für den 16-Bit-Rechner 8086 dargestellt.



In den 4 Datenregistern ist noch die Herkunft vom 8-Bit-Rechner 8080 zu sehen.
Der Block von 8 Adressregistern ist aufgeteilt in 4 Adressregister und 4 Segmentregister.
Befehlszähler und Statusregister komplettieren den Registersatz.

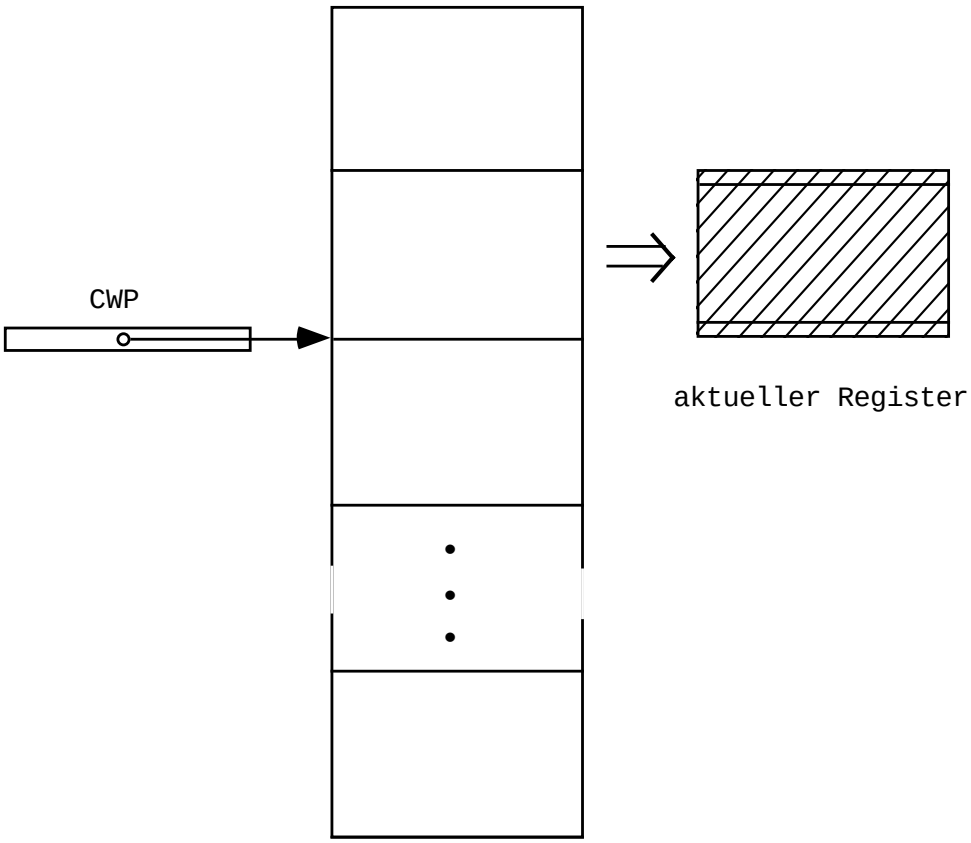
4.4.3. Mehrere Registerbänke

Es gibt mehrere Registerbänke, von denen aktuell eine benutzt wird.

- User- und Supervisorregisterbank
  Benutzer und Supervisor haben eigene Registersätze. Beim Umschalten von User- in den Supervisormode braucht der Inhalt der Register nicht gerettet zu werden.
- Mehrere Registerbänke für Interruptverarbeitung
  Sie sind z. B. dem Supervisor und verschiedenen Interrupts zugewiesen. Diese Lösung hat man eine Weile bei Prozeßrechnern verfolgt. Das Umschalten bei Interrupts (Kontextswitch) geht dann sehr rasch (< 1µs).

4.4.4. Registerband

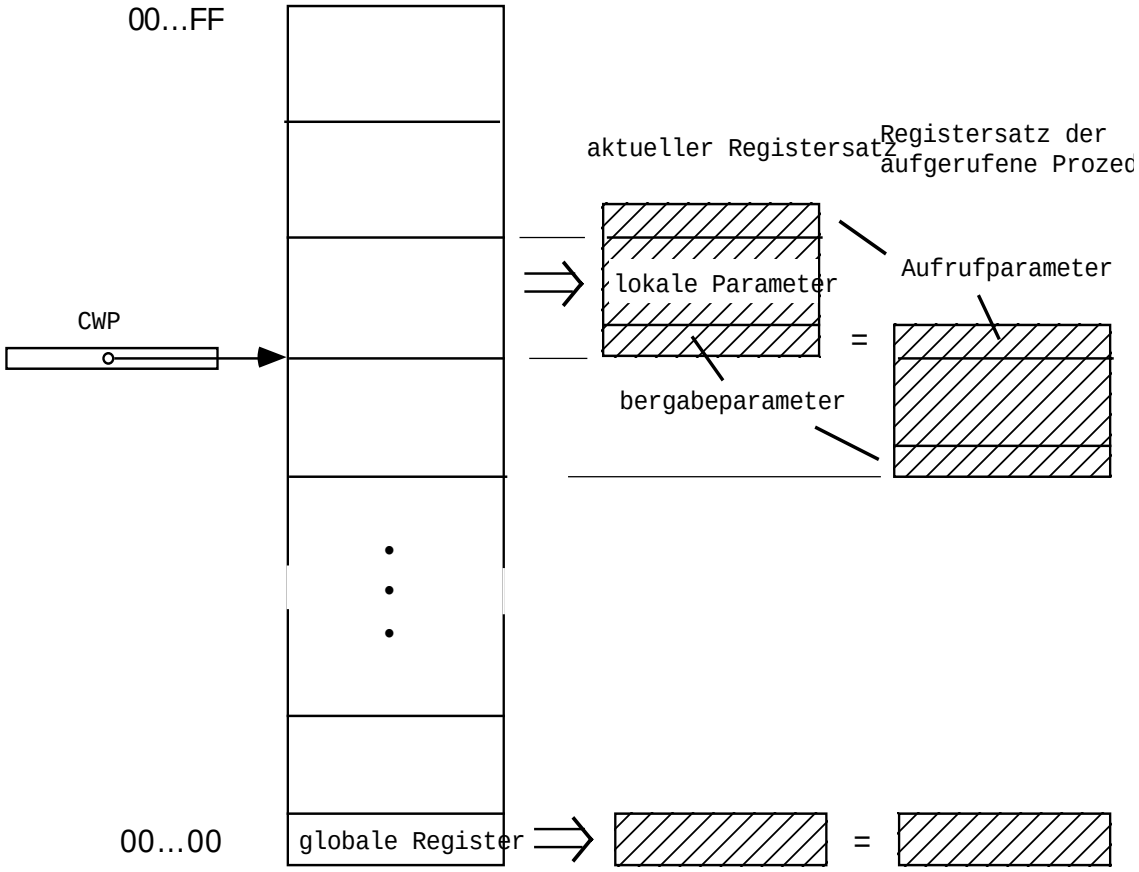
Bei Verwendung vieler Register (z. B. 128 Register zu 4 Byte = 0,5 k Byte Speicher) kann man den aktuell zugreifbaren Registersatz von z. B. 32 Registern so anordnen, daß Prozeduraufrufe vereinfacht werden. Der aktuelle Registersatz wird durch einen Fensterzeiger (current window pointer, CWP) beschrieben. Er enthält die Basisadresse in das Registerband und die Registeradresse im Befehl wird zu dieser Basisadresse addiert.



Eine Verfeinerung teilt den Registersatz auf in globale Register und ein Fenster. Bei einem Prozeduraufruf wird das Fenster überlappend weitergeschaltet.

Im überlappenden Teil stehen dann die Übergabeparameter an die aufgerufene Prozedur;

entsprechend enthält der momentan sichtbare Satz von Registern die vom vorherigen Aufruf übergebenen Parameter und lokale Parameter dazwischen.



Der Prozeduraufruf benötigt dann keinen Transport von Parametern z. B. über einen Stack und ist dadurch schnell.

Das Schema ist gut für Sprachen vom Typ "C". In Sprachen vom Typ "Pascal" sind auch die Variablen der in der Aufrufkette darüberliegenden Prozeduren unten sichtbar; will man teure Umlagerungen vermeiden, kann man das Registerband in den Speicherraum einspiegeln: Die Register können auch als Speicherplätze angesprochen werden und sind so im Prinzip auf dem Umweg über Speicherzugriffe nach wie vor greifbar.

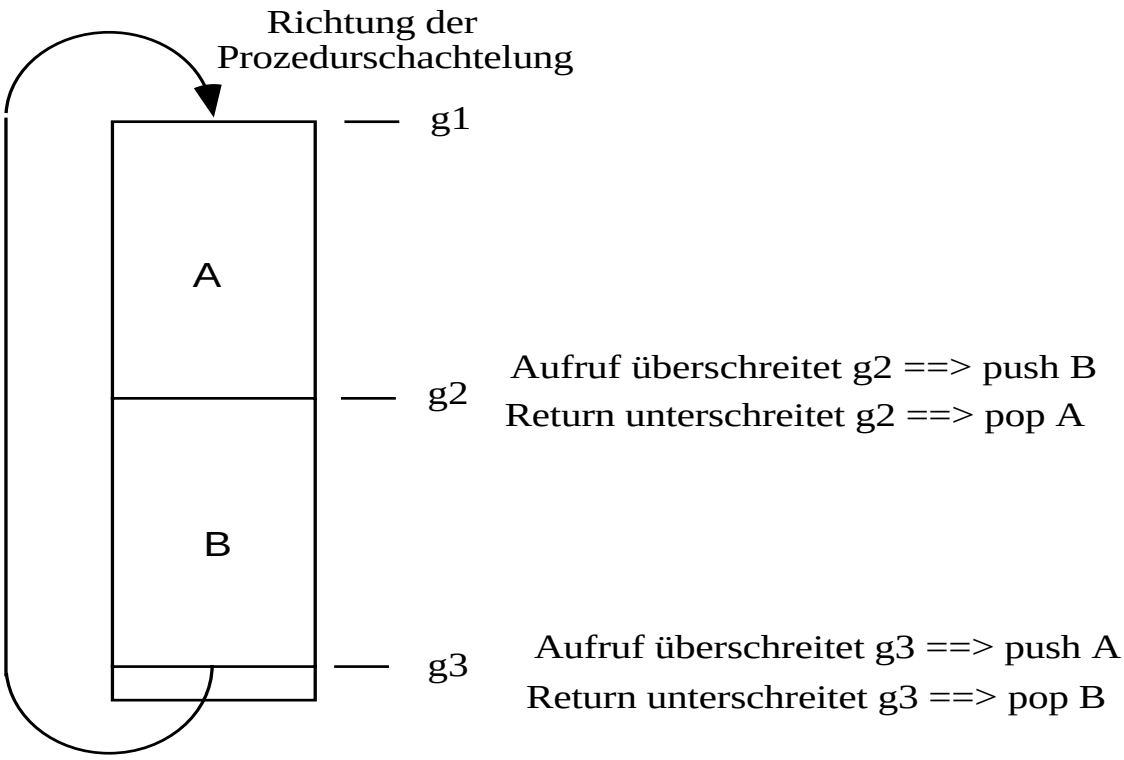
Bei tief geschachtelten Prozeduren kann das Registerband nicht ausreichen. Dann wird man mit einer Wechselpuffertechnik eine Hälfte auf den Stack retten und kann in der

anderen weiterarbeiten. Die Prozedurschachtelung kann sich nach jeder Seite hin um das halbe Registerband ausdehnen. Damit können Rekursionen abgefangen werden. Gibt man dem Registerband eine Länge von

$$2n * (2 * \#\text{ Aufrufparameter} + \#\text{ lokale Parameter}) + \#\text{ globale Parameter}$$

dann liegen die Grenzen g2 und g3 für die Überprüfung bei

$$k * n * (2 * \#\text{ Übergabeparameter} + \#\text{ lokale Parameter}) , \qquad k = 1, 2$$


Richtung der
Prozedurschachtelung
g1
A
g2
Aufruf überschreitet g2 ==> push B
Return unterschreitet g2 ==> pop A
B
g3
Aufruf überschreitet g3 ==> push A
Return unterschreitet g3 ==> pop B