

**Vorkenntnisse A: Dualarithmetik
Speicherung und Verarbeitung von Daten im Computer**

**Vorkenntnisse B: Logikschaltungen
Aufbau der Computerhardware**

Die folgenden Texte sind entstanden, um die Voraussetzungen für den Einsatz meines Modellrechners 1_AMOR zur Einführung in den Aufbau des von-Neumann-Computers und zur Einführung der Maschinensprache (Assembler) zu schaffen. Insofern sind sie als Anhang zum 1_AMOR-Handbuch konzipiert, prinzipiell aber auch ohne den Modellrechner nutzbar. Neben der hier vorliegenden an Pascal orientierten Fassung gibt es auch eine Fassung, wo für Hochsprachenbezüge Java verwendet wird.

Der Modellrechner 1_AMOR samt seinem Handbuch, das einen Unterrichtsgang zur Einführung in die maschinennahe Programmierung beschreibt, wird von mir als Shareware vertrieben („Prüf-vor-Kauf“-Software) und kann z.B. von meiner Internet-Seite (s.u.) herunter geladen werden.

Der Modellrechner sowie die hier vorgestellten Texte unterliegen dem Urheberrecht. Als Autor behalte ich mir alle Rechte vor!

Bis auf weiteres erlaube ich allerdings kostenlos die Verwendung dieses Textes zum Selbststudium. Außerdem dürfen Lehrerinnen und Lehrer diesen Text oder Teile davon in ihrem (schulgeldfreien!) Unterricht verwenden; Lehramtsanwärtern ist die Verwendung z.B. für Examensarbeiten ebenfalls gratis erlaubt, sofern diese nicht auch mit Gewinnstreben verkauft werden – ob als Buch, in honorierten Zeitschriftenartikeln, im Internet o.ä.

Die Verwendung in Publikationen oder zu kommerziellen Zwecken, z.B. auch im Unterricht von Privatschulen oder bei Veranstaltungen von Anbietern, die Teilnehmerbeiträge erheben, usw., ist hingegen nur mit meiner vorher einzuholenden ausdrücklichen schriftlichen Genehmigung möglich.

Anschrift des Verfassers:

R. Krell
Lotte-Wicke-Weg 12, 40627 Düsseldorf

Homepage: <http://www.r-krell.de>

e-Mail: mail@r-krell.de

Über Rückmeldungen und Kommentare (schriftl. oder per e-mail) freue ich mich!

Speicherung und Verarbeitung von Daten im Computer

Der Computer kennt nur die Ziffern (Bits) 0 und 1; traditionell wird eine Kette von 8 Bits als ein Byte bezeichnet. Alle Daten müssen zunächst in die Binärform umgewandelt werden und werden in dieser Form vom Computer verarbeitet.

Im Folgenden soll untersucht werden, wie dies für die in der Sprache Pascal üblichen Datentypen INTEGER, REAL, BOOLEAN und CHAR (sowie STRING) geschieht.

1. Der Datentyp INTEGER (Ganzzahl)

1.1. Darstellung

0000 0000 0000 0000 =	0
0000 0000 0000 0001 =	1
0000 0000 0000 0010 =	2
0000 0000 0000 0011 =	3
....	
0111 1111 1111 1110 =	32766
0111 1111 1111 1111 =	32767
1000 0000 0000 0000 =	-32768
1000 0000 0000 0001 =	-32767
1000 0000 0000 0010 =	-32766
....	
1111 1111 1111 1101 =	-3
1111 1111 1111 1110 =	-2
1111 1111 1111 1111 =	-1
(1) 0000 0000 0000 0000 =	0

Normalerweise werden in Personalcomputern für eine Ganzzahl 2 Byte Speicherplatz reserviert. Damit sind die Zahlen 0 .. 65535 dual darstellbar, die entweder nur als positive Zahlen oder -- wie üblich -- als "positive" Zahlen von 0 .. 32767 und als negative Zahlen von -32768 .. -1 (Zweierkomplement) aufgefasst werden.

Probiere

```
FOR i:=1 TO 10 DO
    WriteLn (32760+i); !
```

a) Nichtnegative Zahlen:

a1) Da jede Stelle der Dualzahl einer Zweierpotenz entspricht, lässt sich eine Dualzahl leicht in eine Dezimalzahl verwandeln: $0110\ 1001 = 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 64 + 32 + 8 + 1 = 105$.

a2) Das Verfahren lässt sich auch in der umgekehrten Richtung durchführen, um zu einer Dezimalzahl ≥ 0 die

dazugehörige Dualzahl zu finden: man prüft einfach, welche Zweierpotenzen in der Dezimalzahl stecken!

Nichtnegative Dezimalzahl (0..32767) in Dualzahl umwandeln:

```
VAR Dualzahl : ARRAY [0..15] OF 0..1;
...
ReadLn (Zahl);
Dualzahl [15] := 0; {vorderste, linke Stelle der Dualzahl}
Zweierpotenz := 16384 {=214}
FOR Stelle := 14 DOWNT0 0 DO BEGIN
    IF Zahl >= Zweierpotenz {=2Stelle}
    THEN BEGIN
        Dualzahl [Stelle] := 1;
        Zahl := Zahl - Zweierpotenz;
    END
    ELSE Dualzahl [Stelle] := 0;
    Zweierpotenz := Zweierpotenz DIV 2;
END; {for}
```

a3) Eine andere Möglichkeit, eine Dezimal- in eine Dualzahl zu verwandeln, bietet das Divisionsrest-Verfahren, das hier beispielhaft an der Umwandlung von 105 dargestellt werden soll:

$$\begin{array}{rcll}
 105 : 2 & = & 52 & \text{Rest } 1, \text{ -----+} \\
 52 : 2 & = & 26 & \text{Rest } 0, \text{ -----+} \\
 \\
 26 : 2 & = & 13 & \text{Rest } 0, \text{ -----+} \\
 13 : 2 & = & 6 & \text{Rest } 1, \text{ -----+} \\
 6 : 2 & = & 3 & \text{Rest } 0, \text{ ----+} \\
 3 : 2 & = & 1 & \text{Rest } 1, \text{ --+} \\
 1 : 2 & = & 0 & \text{Rest } 1, \text{ -}
 \end{array}$$

110 1001 dual (für 105 dezimal)

b) Negative Zahlen: Zunächst kann die Dualdarstellung der entsprechenden positiven Zahl wie in Abschnitt a) bestimmt werden, dann wird in dieser Dualzahl jede 0 durch eine 1 ersetzt und umgekehrt (d.h. alle Bits werden invertiert -> Einerkomplement) und schließlich wird noch die Zahl 1 hinzu addiert (-> Zweierkomplement).

Beispiel: Dualdarstellung von -5. Dualzahl zu +5 ist 0000 0101. Umkehren ergibt 1111 1010 und plus 1 macht daraus schließlich die endgültige Darstellung von -5, nämlich 1111 1011. (Nach der gleichen Methode wird aus -5 auch wieder +5: 1111 1011 --> 0000 0100 + 1 --> 0000 0101.)

An der vordersten Stelle (Bit ganz links) lässt sich leicht erkennen, ob die Dualzahl eine negative (1) oder eine nicht-negative Zahl (0) repräsentiert.

1.2. Addition und Subtraktion

Die Addition von Dualzahlen (egal, ob diese positive oder negative Dezimalwerte repräsentieren!) geschieht wie beim schriftlichen Addieren ziffernweise, beginnend mit der letzten Stelle: 0+0 gibt 0, 1+0 oder 0+1 gibt eine 1, 1+1 gibt 0 und einen Übertrag von 1, usw. Die Überträge über die vorhandene Stellenzahl hinaus gehen verloren!

Da die Subtraktion die Addition von negativen Zahlen ist, reicht das Additionsverfahren (und ein dafür zuständiges Addierwerk im Computer) für beide Operationen!

Beispiele für die Addition bzw. Subtraktion ganzer Zahlen:

$$\begin{array}{rcl}
 \begin{array}{r}
 11 \\
 + 5 \\
 \hline
 \end{array} & \begin{array}{r}
 0000 \ 1011 \\
 + 0000 \ 0101 \\
 \hline
 \end{array} & \\
 =16 & = 0001 \ 0000 &
 \end{array}
 \qquad
 \begin{array}{rcl}
 \begin{array}{r}
 11 \\
 - 5 \\
 \hline
 \end{array} & \begin{array}{r}
 0000 \ 1011 \\
 + 1111 \ 1011 \\
 \hline
 \end{array} & \\
 = 6 & = (1) \ 0000 \ 0110 &
 \end{array}$$

Und natürlich funktioniert das auch, wenn das Ergebnis der Addition bzw. Subtraktion negativ ist!

1.3. Multiplikation und ganzzahlige Division

Auch die Multiplikation zweier (Dual-)Zahlen lässt sich auf die Addition zurückführen. Wieder dient das schriftliche Verfahren bei Dezimalzahlen als Vorbild:

Beispiel:

$$\begin{array}{r} 17 * 203 \\ \hline 34 \\ + 0 \\ + 51 \\ \hline = 3451 \end{array}$$

Genauso z.B. $10 * 5 = 1010 * 0101$

$$\begin{array}{r} 1010 * 0101 \\ \hline 0 \\ + 1010 \\ + 0 \\ + 1010 \\ \hline = 110010 = 50 \end{array}$$

Hier ist -- außer dem Verschieben der Zahlen nach rechts -- nur die Addition nötig. Da als Dualziffern nur 0 und 1 auftreten, ist (anders als beim normalen schriftlichen Multiplizieren) auch nicht die Kenntnis des Kleinen Einmaleins notwendig! Bei negativen Faktoren kann es hingegen Probleme geben, die aber vermeidbar sind, wenn man nur die Beträge miteinander multipliziert und dem Produkt nachher das richtige Vorzeichen gibt. Dabei kann das Ergebnisvorzeichen leicht durch einstellige, übertragslose Addition der vordersten Bits der beiden Faktoren bestimmt werden: $0+1 = 1+0 = 1$ (entspricht $+\bullet- = -\bullet+ = -$) und $0+0 = 1+1 = 0$ (entspricht $+\bullet+ = -\bullet- = +$).

Das Dividieren von zwei ganzen Zahlen (das natürlich nicht immer ohne Rest gelingt) ist genauso einfach, z.B. $100 : 8 =$

$$\begin{array}{r} 01100100 : 01000 = 1100 \text{ Rest } 0100 \text{ (dezimal 12 Rest 4)} \\ + 11000 \\ \hline (1)001001 \\ + 11000 \\ \hline (1)000010 \\ + 0 \\ \hline 100 \\ + 0 \\ \hline 100 \end{array}$$

Hier zeigt die erste 1 im Ergebnis, dass 01000 (=8) einmal in 01100 steckt. Statt dann 01000 von 01100 abzuziehen, wird das 2er-Komplement 11000 (= -8) addiert. Die zweite 1 zeigt, dass 01000 in 1001 steckt; die Nullen folgen, weil 1000 weder in 10 noch in 100 steckt. Diese letzte 100 (=4) bleibt als Rest übrig.

In Pascal liefert $100 \text{ DIV } 8$ das ganzzahlige Ergebnis 12, der Rest 4 kann mit $100 \text{ MOD } 8$ erfragt werden. Bei der Dualoperation ist wieder nur Verschieben sowie die Addition (hier des Zweierkomplements des Divisors) nötig, so dass sämtliche Grundrechenarten mit Ganzzahlen mit Hilfe des Computer-Addierwerks erledigt werden können.

2. Der Datentyp REAL (Gleitkommazahl)

2.1. Darstellung

Die Umwandlung von Dezimalbrüchen in eine entsprechende Dualkommazahl gelingt ebenfalls mit dem in 1.1. a1) vorgestellten Algorithmus, sofern dort auch gebrochene Zweierpotenzen und negative Stellen (als Nachkommastellen) zugelassen werden. Die Anzahl der erlaubten Komma Stellen wird von der zur Verfügung gestellten Speicherplatz bestimmt (in Turbo-Pascal sind das insges. 6 Bytes für jede REAL-Zahl, wovon aber 1 Byte für die Verwaltung der Position des Kommas benötigt wird. Damit sind 11 Dezimalstellen möglich). Ungünstig auf die Rechengenau-

igkeit wirkt sich aber in jedem Fall aus, dass viele endliche Dezimalbrüche keine endliche Dualbruchdarstellung haben, also selbst bei vielen Stellen nie genau dargestellt werden können.

Beispiel: Dualdarstellung des Dezimalbruchs 0.1 (1 Zehntel). Wendet man (z.B. mit FOR Stelle := 1 DOWNT0 -10) den in 1.1.a)(2) angegebenen Algorithmus an, so erhält man folgende Wertebelegungstabelle

Zahl:	0.1	0.1	0.1	0.1	0.1	0.1	0.0375
Stelle:	1	0	-1	-2	-3	-4	-5
2^{Stelle} :	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
Zweierpotenz:	2	1	0.5	0.25	0.125	0.0625	0.03125
Dualzahl[Stelle]:	0	0	0	0	0	1	1

Zieht man auch jetzt wieder 2^{-5} , also 0.03125, von der verbliebenen Zahl 0.0375 ab, so erhält man 0.00625, das aber 2^{-6} ($= 1/64 = 0.015625$) und 2^{-7} ($= 1/128 = 0.0078125$) nicht enthält, aber wieder 2^{-8} sowie 2^{-9} usw. Tatsächlich ist die Dualdarstellung von 0.1 der periodische Dualbruch 0.0 0011 0011 0011 0011..., der im Computer irgendwo abgeschnitten und damit ungenau werden muss. (Die Periode sieht man leichter bei Ausführung der Dual-Division 01 : 01010)

Die verheerende Auswirkung kann man an einem kurzen Pascalprogramm studieren, das nie aufhört:

```

VAR Zahl : REAL;
BEGIN Zahl := 0;
  REPEAT
    Zahl := Zahl + 0.1
  UNTIL Zahl = 1;
END.
```

Abhilfe ist nur möglich, wenn die Dezimalbrüche nicht in Dualbrüche verwandelt werden, sondern auch im Rechner die Dezimalziffern einzeln abgespeichert werden (wobei natürlich jede Dezimalziffer für sich binär durch eine 4-Bit-Folge dargestellt werden muss, was zum Namen BCD-{=binär-codierte-Dezimalziffern}-Darstellung geführt hat), und durch geeignete Algorithmen das Dezimalrechnen imitiert wird. Eigentlich reicht sogar wieder die normale Dualzahl-Addition, wenn man dabei von rechts beginnend zu Ergebnisziffern, die einen Übertrag zur nächsten 4-Bitgruppe verursachen, sowie zu ungültigen Ziffern ($> 1001=9$) die Dualdarstellung 0110 der 6 unter Berücksichtigung des Übertrags hinzuaddiert. BCD-Codierung und -Rechnung ist z.B. für das Bankwesen vorgeschrieben, damit auch 10-Cent- und Centbeträge korrekt behandelt werden!

2.2. Grundrechenarten

Addition und Subtraktion können praktisch wie bei Ganzzahlen ausgeführt werden, wenn man das Komma nur vorher geeignet verschiebt:

Beisp.: dezimal (E=10er-Expon.) dual (D=2er-Exponent)

0.1250 E+0	12500. E-5	0.10 D-2	100. D-5
+ 0.9375 E-1	+ 9375. E-5	0.11 D-3	+ 11. D-5
	-----		-----
	= 21875. E-5		= 111. D-5
	(= 0.21875 E+0)	(= 0.111 D-2	= 0.111 • 2 ⁻²)

Und die Multiplikation wird -- genau wie beim schriftlichen Rechnen mit Dezimalzahlen -- ohnehin mit Ganzzahlen durchgeführt und erst anschließend wird das Komma an die richtige Stelle gesetzt. Bei der Division schließlich wird das Komma gesetzt, wenn beim Dividenten Stellen nach dem Komma "heruntergeholt" werden müssen -- z.B. eine weitere Null wie beim Beispiel 100:8 aus 1.3. (was zum Dualergebnis 100.1 führt). Durch Erweitern von Divident und Divisor muss natürlich immer zuvor der Divisor ganzzahlig gemacht werden!

2.3. Höhere Rechenarten

Für REAL-Zahlen stehen unter Pascal auch viele mathematische Funktionen zur Verfügung -- wie z.B. SQRT (Wurzel) oder SIN (Sinus).

Der nachfolgende Algorithmus benutzt nur die Grundrechenarten, um in wenigen Schleifen-Durchläufen die Wurzel aus z zu berechnen:

```

FUNCTION Sqrt (z: REAL): REAL;           {Wurzelfunktion}
VAR t1, t2 : REAL;
BEGIN  {"Heron-Verfahren"}
    t2 := 7;                             {beliebiger Startwert > 0}
    REPEAT t1 := t2;
           t2 := t1/2 + z/(2*t1);
    UNTIL  ABS(t2-t1) < 0.000001;         {Genauigkeit}
    Sqrt  := t2;                           {Näherungswert für Wurzel aus z}
END;
```

Andere Funktionen können leicht mit Hilfe ihrer Taylorschen Potenzreihe bestimmt werden, wie z.B. der Sinus einer Zahl x (x im Bogenmaß):

$$\sin(x) := x/1 - x*x*x/(1*2*3) + x*x*x*x*x/(1*2*3*4*5) - x*x*x*x*x*x*x/(1*2*3*4*5*6*7) + - \dots,$$

wofür ebenfalls nur die Grundrechenarten benötigt werden.

Abschließend sei festgehalten, dass sich sämtliche Rechnungen mit Zahlen auf die Grundrechenarten, damit auf die entsprechenden Ganzzahloperationen und somit letztlich aufs Addieren sowie das Verschieben und -- zusätzlich für die Komplementbildung -- aufs ziffernweise Invertieren (1 wird 0, 0 wird 1) zurückführen lassen. Für alles Rechnen braucht ein Computer also lediglich ein Addierwerk, ein Schieberegister und einen Inverter!

3. Der Datentyp BOOLEAN (Wahrheitswert)

3.1. Darstellung

Da es nur zwei verschiedene Wahrheitswerte, false und true, gibt, wäre zur Codierung nur 1 Bit mit den beiden Möglichkeiten 0 und 1 erforderlich. Weil der Speicherplatz des Computers aber i.a. byteweise organisiert ist, wird meist trotzdem ein ganzes Byte zur Darstellung genommen. In Turbo-Pascal ist

```
false (falsch) = 0    = 0000 0000
true  (wahr)   = 1    = 0000 0001    bzw. alles ≠ 0,
```

wodurch auch die Reihenfolge false < true festgelegt ist (in Basic wird oft anders codiert, z.B. ist in Microsoft-Basic wahr durch -1 = 1111 1111 und falsch durch 0 = 0000 0000 repräsentiert).

3.2. Logische Verknüpfungen

In Pascal stehen die 2-stelligen Verknüpfungen AND und OR sowie die 1-stellige Operation NOT zur Verfügung. Für true=1 und false=0 bräuchten diese Operationen nur aufs letzte Bit zu wirken. Bei NOT wird das Bit mit dem Inverter umgewandelt; auch für AND und OR stehen im Computer hardwaremäßig spezielle Schaltungen ("Gatter") bereit. (Zweistellig: es werden zwei Wahrheitswerte a und b verknüpft wie bei x := a AND b. Einstellig: der Operator bezieht sich nur auf einen Wahrheitswert wie bei x := NOT a)

4. Der Datentyp CHAR (Schriftzeichen)

Die Schriftzeichen sind in einer Tabelle abgelegt und durchnummeriert. Im Rechner wird an Stelle des Schriftzeichens seine Nummer (dezimal 0..255, binär 0000 0000 .. 1111 1111) in einem Byte Speicherplatz abgelegt. In Personalcomputern wird oft die ASCII- (oder ANSI-)Tabelle verwendet. Die Nummern 0..31 sind dort für nicht ausdrückbare Zeichen (z.B. für die Return- und die Kursortasten) reserviert; Sonder- und Schriftzeichen sind mit 32..127 nummeriert. Der Buchstabe 'A' ist z.B. das Zeichen Nr. 65 und wird entsprechend als 0100 0001 im Computer dargestellt, während 'a' das Zeichen Nr. 97 ist und damit als 0110 0001 dargestellt wird. Die Zeichen Nr. 128 ..255 sind in der ASCII-Tabelle nicht verbindlich genormt und werden von verschiedenen Computerherstellern mit unterschiedlichen Zeichen und Funktionen belegt. Hier unterscheiden sich auch ASCII- und ANSI-Tabelle, weswegen die Umlaute von Dos-Texten (ASCII) in von Windows-Programmen (ANSI) meist falsch dargestellt werden und umgekehrt. Ein kurzes Programm gibt Auskunft über die aktuelle Belegung:

```
PROGRAM Ascii;
VAR Nr : Byte;  (* = 0..255 *)
BEGIN
  FOR Nr := 32 TO 255 DO
    Write (Nr:4, '=', CHR(Nr) );
  END.
```

Da es keine Rechnungen mit Schriftzeichen gibt, müssen dafür auch keine Vorkehrungen getroffen werden. Die Reihenfolge der Buchstaben wird durch ihre Nummern festgelegt (die man in Pascal mit der Funktion ORD(Buchstabe) erfragen kann).

5. Der zusammengesetzte Datentyp STRING (Zeichenkette)

Ergänzend sei hier noch auf eine Besonderheit des Pascal-Datentyps "string" hingewiesen, der im Wesentlichen mehrere Schriftzeichen hintereinander speichert. Ist z.B. `VAR Wort: string[10]` deklariert und wird im Programm `Wort:='Beispiel'` zugewiesen, so sind die Komponenten `Wort[1]='B'`, `Wort[2]='e'`, `Wort[3]='i'..` und `Wort[8]='l'`. Die Komponenten `Wort[9]` und `Wort[10]` haben zufällige Inhalte. Außer den ASCII-Kodes dieser Buchstaben wird aber zusätzlich in `Wort[0]` praktisch das Zeichen Nr. 8 der ASCII-Tabelle (bzw. die Bit-Kombination `0000 1000 = 8`) gespeichert, wodurch festgehalten wird, dass die Variable `Wort` nur bis zur 8. Stelle mit sinnvollen Werten gefüllt worden ist. Im Computerspeicher befinden sich einfach hintereinander (ohne die hier zur besseren Lesbarkeit eingefügten Leerstellen): `0000 1000 0100 0010 0110 0101 0110 1001 0111 0011 0111 0000 0110 1001 0110 0101 0110 1100` `????` `????` `????` `????`. Die Pascalfunktion `Length` "berechnet" beim Aufruf von `Length(Wort)` nur `Length := ord(Wort[0])`, d.h. interpretiert das 0-te Byte als Zahl. Jetzt wird sicher verständlich, warum Pascal-Zeichenketten nur bis zur maximalen Länge von 255 Zeichen erlaubt sind!

6. Weitere Datentypen, Allgemeines, Programmsteuerung

Alle weiteren (zusammengesetzten und benutzerdefinierten) Datentypen lassen sich auf die vier oben behandelten Grundtypen zurückführen und werden auch im Computer entsprechend dargestellt und ggf. verknüpft.

Da alle unterschiedlichen Datentypen im Computer aber immer in Binärform vorliegen, muss sich die Speicherplatzverwaltung merken, ob z.B. `0100 0001` jetzt die Zahl 65 oder das Zeichen 'A' oder einen bestimmten Teil einer REAL-Zahl oder gar ein bestimmtes Pascal-Schlüsselwort im Programmtext darstellt. Denn auch das Programm selbst wird natürlich binär codiert, der Pascal-Programmtext im Editor wird zeichenweise nach der ASCII-Tabelle abgespeichert (In Basic-Umgebungen werden die Basic-Schlüsselworte und -befehle oft intern sofort in 2-Byte-Nummern übersetzt und so entsprechend Platz sparender gesichert). Vom Compiler wird das Quelltext-Programm in Maschinensprache übersetzt -- natürlich gleichfalls nur eine Folge von Nullen und Einsen. Die Maschinensprache besteht dabei aus den Nummern, die den einzelnen Maschinenbefehlen zugeordnet sind, sowie den "Adressen" (=Speichernummern) der Variablen. Allerdings kann ein einzelner Pascalbefehl den Aufruf mehrerer Maschinenbefehle erfordern. Der Compiler übersetzt jede Pascalanweisung in alle dafür nötigen Maschinenbefehle; die Befehlsnummern werden in einer Tabelle nachgesehen, die i.a. von einer Diskette in den Computerspeicher geladen wurde, während die Adress-Tabelle entsprechend der jeweiligen Variablendeklaration für jedes Programm eigens erstellt wird (Näheres über die Maschinensprache später!)

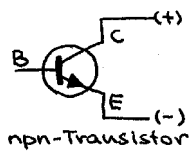
Nachdem nun grundsätzlich klar ist, wie ein Computer ausschließlich mit 0 und 1 arbeiten kann, und auch komplizierte Anwendungen nur einige Gatter (für logische Verknüpfungen) bzw. i.w. ein Addierwerk (zum Rechnen) erfordern, soll im folgenden Anhang B die dafür benötigte Hardware, d.h. der physikalisch-technische Aufbau der Gatter, des Rechenwerks und des Speichers besprochen werden, um so den Computer ganz in Funktion und Aufbau zu durchschauen und ihn dadurch zu entmystifizieren.

Aufgaben und Übungen zur Computerarithmetik

1. Dualdarstellung
 - a) In Fernschreibern stehen nur die 26 Großbuchstaben 'A' bei 'Z' und die Ziffern 0 bis 9 zur Verfügung. Satz- und Sonderzeichen außer dem Leerzeichen gibt's nicht („ANKOMME MORGEN 15 UHR 10 STOPP“). Wie viele Bit sind zur Dualcodierung nötig?
 - b) Gib dezimal den Ganzzahlbereich an, der sich mit Dualzahlen der Länge
 - b1) 3 Bit
 - b2) 13 Bit
 - b3) n Bitdarstellen läßt, wenn (1) nur positive oder (2) neg. und positive Zahlen codiert werden sollen.
 - c) Ist das Morsealphabet ein binärer Code?
2. Fünfer-System: Im Land der Einhändigen hat sich das Fünfersystem durchgesetzt.
 - a) Welche Ziffern gibt es (im Fünfersystem ist die Basis=5) ?
 - b) Übersetze ins Dezimalsystem $(4301)_5$, $(2331042)_5$ bzw. ins Fünfersystem 378 u. 47 !
 - c) Schreibe ein Pascal-Programm für eine der Übersetzungsrichtungen in b). Verwende dabei die fertige PROCEDURE Potenz (Basis, Exponent: integer): integer;
 - d) Welcher positive Zahlbereich ist mit einer 4-stelligen Fünferzahl abzudecken?
 - e) Welcher negative und positive Zahlbereich ist mit 3-stelligen Fünferzahlen abzudecken? Wie würden vermutlich -1 und -2 im Fünfersystem lauten? Warum?
 - f) Stelle dir die Multiplikation zweier (positiver) Fünfer-Zahlen vor. Welche Operationen sind zusätzlich zur Verschiebung und Addition von Fünferzahlen nötig?
3. Rechnen mit Dualzahlen
 - a) 5-Bit-Dualzahlen repräsentieren Dezimalzahlen von -16 bis +15. Führe die angegebenen Rechnungen mit genau 5 Bit aus und notiere zusätzlich, ob das Ergebnis richtig ist (also nicht etwa durch einen Übertrag in die vorderste Stelle das falsche Vorzeichen erhält). Verwende ggf. das Zweierkomplement und übersetze nur die 5-Bit-Ergebnisse in Dezimalzahlen.
 - a1) $01101 + 01011$
 - a2) $00010 - 00101$
 - a3) $01001 - 10110$
 - a4) $00101 * 01101$
 - a5) $00011 / 01000$ (mit allen Nachkommstellen)
 - b) In Pascal gibt es u.a. die Funktion $\exp(x)$, für die laut Formelsammlung gilt:
$$\exp(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \quad \text{wobei z.B. } 3! = 1 \cdot 2 \cdot 3 \text{ ist.}$$
Berechne (dezimal) $\exp(0,4)$ auf 3 Stellen genau. Woran kann der Computer merken, ob er aufhören darf?
 - c) Die Formel in b) läßt Schwierigkeiten für $x > 1$ befürchten. Welche/warum? Berechne z.B. das 19. Glied der Reihe für $x = 1,7$
 - d) Welche Hardware ist nötig, um einem Dual-Computer sämtliche Rechenarten für alle Zahlentypen zu ermöglichen?
 - e) Verwandle in BCD-Zahlen und berechne $4,28 + 6,56$ sowie $0,234 + 2,87$
 - f) Warum wird bei der BCD-Addition zur Dezimalanpassung „0110“ addiert?

Aufbau der Computer-Hardware

In heutigen Computern werden die beiden Dualziffern (Bits) 0 und 1 durch elektrische Spannungen dargestellt: 1 = +5 Volt (neuerdings auch oft nur ca. +3 Volt) = Verbindung mit dem Pluspol der Spannungsquelle; 0 = 0 Volt = Verbindung mit dem Minuspol der Quelle. Zwischen beiden Zuständen könnte mit einem mechanischen Schalter oder auch mit einem Relais hin- und her geschaltet werden; tatsächlich verwendet man inzwischen Transistoren

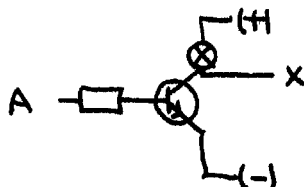


(von den viele auf einem einzigen Chip eines IC [integrated circuit = zusammengefasste Schaltung] untergebracht sind). Nur, wenn in die Basis B eines npn-Transistors Strom fließt (was nur durch Anlegen einer [positiven] Spannung ermöglicht wird), ist die Collector-Emitter-Verbindung C-E leitend. Solange Spannung (d.h. eine logische Eins) an der Basis anliegt, wird praktisch der Schalter in der C-E-Strecke betätigt.

Logische Schaltungen (Gatter)

Im folgenden werden nur Prinzip-Schaltungen gezeigt; in Wirklichkeit werden aus fertigungstechnischen Gründen oder für höhere Geschwindigkeiten auch andere, aufwändigere Schaltungen benutzt.

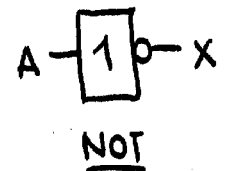
1) Inverter (NOT-Gatter): Wird am Eingang A Spannung (also eine Eins) angelegt, so wird die C-E-Strecke des Transistors leitend. Damit gibt es aber eine gute Verbindung des Aus-



gangs X mit dem Minuspol der Spannungsquelle, so dass die Spannung am Ausgang auf

A	X
0	1
1	0

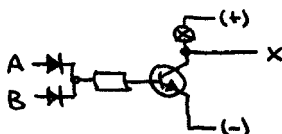
Lampe aus
Lampe hell



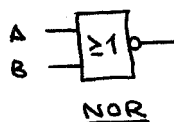
null sinkt, d.h. eine 0 aus dem Ausgang

kommt. Liegt andererseits der Eingang auf 0, so bleibt die C-E-Verbindung unterbrochen und der Ausgang ist über den (kalten) Glühfaden der Lampe nur mit dem Pluspol verbunden: der Ausgang zeigt eine 1 (+5 V). Der Ausgang weist also genau das Gegenteil des Eingangs auf, womit wir bereits einen Inverter, also ein NOT-Gatter (Nicht-Glied) realisiert haben.

2) NOR- und OR-Gatter: Der ursprüngliche Eingang eines Inverters kann durch zusätzliche

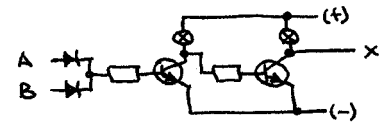


A	B	X
0	0	1
0	1	0
1	0	0
1	1	0



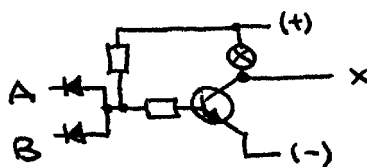
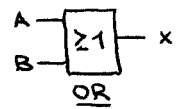
Dioden zu zwei (oder mehr) Eingängen erweitert werden. Nun reicht es, wenn an einem dieser Eingänge eine 1 ankommt (die anderen Eingänge dürfen ruhig mit 0 verbunden sein: ein Abfluss von Strom über einen solchen Eingang ist wegen der Dioden trotzdem nicht möglich!), um den Transistor leitend zu schalten und damit den Ausgang auf 0 zu setzen. Nur wenn alle Eingänge auf 0 sind, erscheint am Ausgang die 1. Damit hat die Schaltung genau die Eigenschaften eines NOR-Gatters (Nicht-Oder-Schaltung), wie der Vergleich von Schalt- und Wahrheitswerttabelle zeigen.

Ein "richtiges" Oder (OR-Gatter) erhält man durch Invertieren des NOR-Ausgangs, d.h. in dem man an den Ausgang des NOR-Gatters einfach einen oben beschriebenen Inverter anhängt.

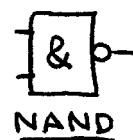


3) NAND- bzw. AND-Gatter: Gegenüber dem NOR-Gatter sind hier die Anforderungen vertauscht. - Beim NAND-Gatter soll nämlich genau dann die 0 am Ausgang abgegriffen werden können, wenn alle Eingänge auf 1 sind. Nur dann darf die C-E-Strecke des Transistors leiten. Andernfalls muss also jeder 0-Eingang dafür sorgen, dass an der Basis keine Spannung anliegt und evtl. Strom zum Minuspol der Quelle abfließt. Das erzwingt einmal, dass die Eingangsdiode anders herum eingebaut werden müssen, und zum anderen, dass jetzt durch einen Widerstand die Basis B mit Strom versorgt wird, weil ein

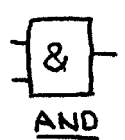
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0



A	B	X
0	0	1
0	1	1
1	0	1
1	1	0



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1



Stromfluss durch die Dioden in Sperrrichtung ja nicht mehr möglich ist. Das Bild zeigt das resultierende NAND-Gatter; wieder wird durch Nachschalten eines Inverters daraus ein AND-Gatter!

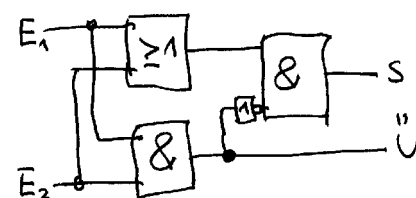
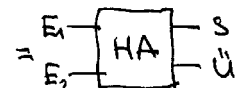
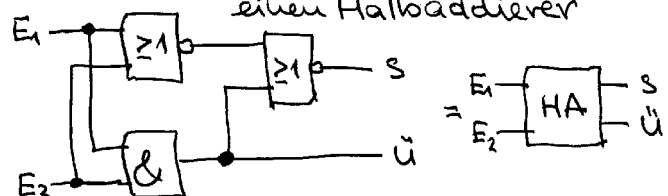
Die oben beschriebenen Schaltungen ermöglichen bereits sämtliche Operationen, die im Computer für den Typ boolean nötig sind. Für das Rechnen mit Zahlen (egal welchen Typs) wurde darüber hinaus nur noch ein Addierwerk gebraucht. Dieses Addierwerk soll im folgenden Abschnitt aus Logikgattern zusammengebaut werden.

Schaltnetze

Intuitiv können verschiedene Gatter zusammengefügt werden: so könnte z.B. ein XOR (=Entweder-Oder) bzw. ein Halbaddierer wie nebenstehend gebaut und aus zwei Halbaddierern und einem OR ein Volladdierer zusammengesetzt werden. Acht parallele Volladdierer bilden das Paralleladdierwerk zum Addieren zweier 8-Bit-Dualzahlen.

Für die Konstruktion beliebiger Schaltungen soll statt des intuitiven Verfahrens jedoch ein systematisches Vorgehen ermöglicht werden:

Zwei von vielen Möglichkeiten für einen Halbaddierer



E ₁	E ₂	S	Ü
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

= XOR = AND

Disjunktive Normalform (DNF): Um kompliziertere Schaltnetze aufzubauen, werden zunächst die Anforderungen an das Bauteil in einer Wahrheitstabelle festgelegt. Eine Möglichkeit, eine Funktionsvorschrift für die zugrunde liegende Schaltfunktion aufzustellen,

geht von folgender Überlegung aus: Findet man zu jeder 1 in der Ausgangs-(=Ergebnis-)spalte der Wertetabelle einen Term, der genau für die in dieser Zeile angegebene Eingangsbelegung den Funktionswert 1 erzeugt, so braucht man diese sog. Minterme nur noch durch eine Oder-Verknüpfung (Disjunktion) zu verbinden, um die gesamte Funktionsvorschrift zu erhalten. Die einzelnen Minterme lassen sich ihrerseits einfach aus den mit Und verbundenen Werten der Eingänge (wenn dort eine 1 anliegt) bzw. der negierten Eingänge bilden (falls der Eingangswert 0 ist). Auf diese Weise findet man die Funktionsvorschrift in der disjunktiven Normalform (DNF), die mit den drei Logikoperationen Oder, Und und Nicht auskommt, für die bereits Hardware-Bausteine vorgestellt wurden.

a	b	c	S
0	0	0	0
0	0	1	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

a	b	c	Ü
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Volladdierer: Sollen zwei mehrstellige Dualzahlen addiert werden, so braucht man für jede Stelle ein Bauteil mit drei Eingangs- und zwei Ausgangsleitungen. An zwei Eingängen liegen die beiden Ziffern der Summanden an, am dritten Eingang kommt ein eventueller Übertrag von früheren, niederwertigen Stellen hinzu. An einem Ausgang (S) erscheint die richtige Stelle des Ergebnisses, am anderen Ausgang (Ü) kommt ein

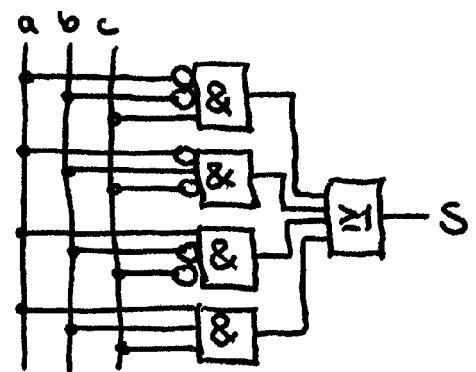
Übertrag für die nächsthöhere Stelle heraus. Die abgebildeten Wahrheitswerttafeln beschreiben das geforderte Verhalten des Volladdierers für beide Ausgänge.

Für diese beiden Ausgänge lässt sich dann nach dem oben beschriebenen Verfahren der Funktionsterm in DNF beschreiben:

$$S(a,b,c) = (\bar{a}\bar{b}\wedge c) \vee (\bar{a}\wedge b\wedge \bar{c}) \vee (a\wedge \bar{b}\wedge \bar{c}) \vee (a\wedge b\wedge c)$$

$$\bar{U}(a,b,c) = (\bar{a}\wedge b\wedge c) \vee (a\wedge \bar{b}\wedge c) \vee (a\wedge b\wedge \bar{c}) \vee (a\wedge b\wedge c)$$

(wobei wie in der Mathematik üblich \wedge = 'und' und \vee = 'oder' bedeutet. In vielen Büchern wird leider das \wedge -Zeichen ganz weggelassen und das \vee -Zeichen durch ein '+' ersetzt). Für den Summenausgang S wird die DNF wie angegeben in Hardware übertragen. Bemerkenswert ist, dass die Realisierung der DNF immer nur zu höchstens dreistufigen Schaltnetzen führt (jedes Signal muss höchstens einen Inverter, ein Und-Glied und ein Oder-Gatter passieren; alle Inverter und Und-Gatter arbeiten parallel). Entsprechend sind solche Schaltungen recht schnell. Für den Übertrags-Ausgang Ü ist natürlich eine analoge Umsetzung möglich; allerdings können bei Ü noch Vereinfachungen durchgeführt werden.



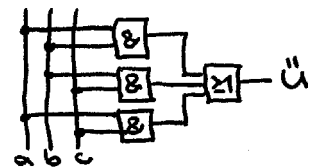
Vereinfachung von Schaltnetzen nach Karnaugh und Veitch: In der DNF werden manchmal zwei Minterme aufgenommen, die sich nur in einer einzigen Eingangsbelegung unterscheiden. In solchen Fällen ist es aber gar nicht nötig, beide Minterme zu notieren (und durch zwei entsprechende Und-Gatter zu realisieren). Vielmehr hat die Änderung des Wertes an diesem Eingang offenbar keine Auswirkung auf das Ergebnis. Deshalb reicht ein Und-Term, in dem der entsprechende Eingangswert gar nicht mehr vorkommt; in der Hardware kann also ein Und-Gatter und die Beschaltung dieses Eingangs eingespart werden. Um solche Einsparungsmöglichkeiten leichter zu finden, haben Karnaugh und Veitch eine graphische Methode vorgeschlagen. Statt in der üblichen Wahrheitswerttafel werden die Funktionswerte in einem tabellenförmigen Diagramm angeordnet, dessen Zeilen und Spalten mit den verschiedenen Eingangswert(kombination)en so beschriftet sind, dass sich benachbarte Spalten bzw. Zeilen immer nur um genau ein Bit in den Eingangswerten unterscheiden. Haben benachbarte Zellen den gleichen Funktionswert 1, so ist das unterschiedliche Eingangsbit für diesen Wert irrelevant. Aber nicht nur Paare benachbarter Einsen lassen sich so zusammenfassen, auch Vierer-, Achter-, Sechzehner-,... -gruppen sind möglich, wenn mehrere Eingänge für einen Funktionswert 1 unerheblich sind. Dabei muss noch beachtet werden, dass auch gegenüberliegende Ränder des Diagramms als benachbart gelten. Außerdem kann eine Zelle mit einer 1 durchaus in mehreren Gruppen enthalten sein.

Hier wird beispielhaft das Karnaugh-Diagramm für den Übertrags-Ausgang \ddot{u} des Volladdierers angegeben; nur noch für jede Gruppe ist ein Und-Term nötig!

\ddot{u}	ab	01	11	10
c	0	0	0	1
1	0	1	1	1

$$\ddot{u} = (a \wedge b) \vee (b \wedge c) \vee (a \wedge c)$$

Der Hardware-Aufwand ist entsprechend geringer als bei der direkten Umsetzung des ursprünglichen DNF-Funktionsterms, wobei die vereinfachte Schaltung weiterhin höch-

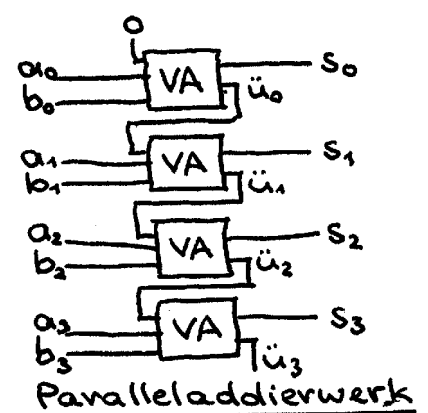


stens dreistufig bleibt.

Die gesamte Hardware des Volladdierers wird zu einem Symbol "VA" (oder auch " Σ ") zusammengefasst.

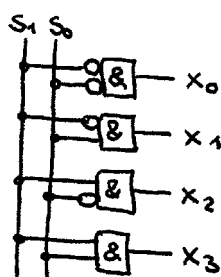
Aufgabe: Normalerweise klingelt es bei Oma Rüstig, wenn jemand unten an der Haustür (a) oder an ihrer Wohnungstür (b) auf den Klingelknopf drückt. Um nicht im Mittagsschlaf gestört zu werden, hat sie sich einen zusätzlichen Schalter (c) einbauen lassen, mit dem sie die Klingel aus- (0) oder einschalten kann (1). Stelle die Wahrheitswerttafel $a \ b \ c \mid$ Klingel auf, notiere die DNF, vereinfache nach Karnaugh/Veitch (falls möglich) und zeichne die Hardware!

Parallel-Addierwerk: Für die Addition z.B. zweier vierstelliger Dualzahlen ist eine Parallelschaltung aus vier Volladdierern nötig (Parallel-Addierwerk), wobei das Ergebnis allerdings erst dann als endgültig betrachtet werden kann, wenn genügend Zeit verstrichen ist, damit alle Überträge bis zur höchstwertigen Stelle durchgelaufen sind. In der Praxis werden daher auch noch andere Addierschaltungen verwendet.



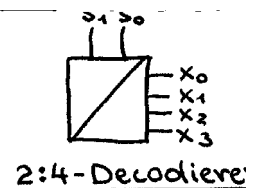
Mit den Logikgattern und dem Addierwerk stehen mithin alle Bauteile zur Verfügung, die bei der Verarbeitung von Daten im Computer überhaupt gebraucht werden. Auch für die Zweierkomplementbildung reichen diese Bauteile: Das Einerkomplement wird durch stellenweises Invertieren der Ursprungszahl mit je einem NOT-Gatter pro Bit bewirkt; mit Hilfe des Addierwerks kann dann die 1 zur letzten Stelle hinzugezählt werden.

Decodierschaltung



Allerdings muss auch gezielt die richtige Verarbeitungseinheit ausgewählt werden können. Willkürlich lassen sich hier die vier Einheiten mit den Nummern 0 bis 3 versehen;

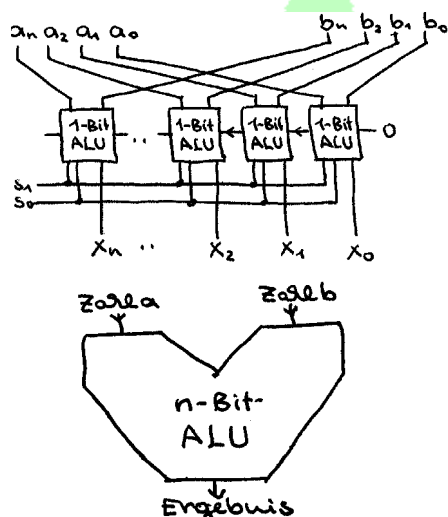
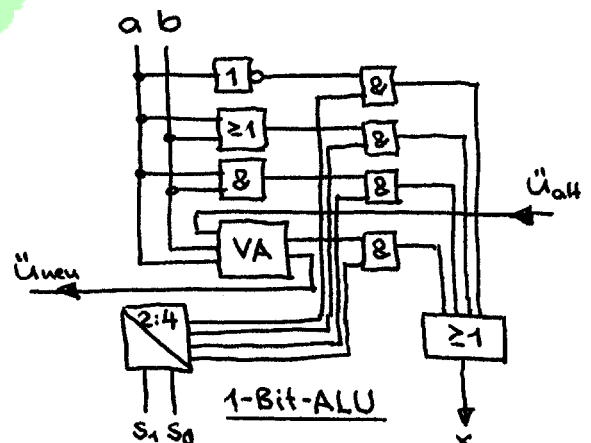
S ₁	S ₀	x ₀	x ₁	x ₂	x ₃
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



in der computergerechten Darstellung sind diese Nummern 2-Bit-Dualzahlen. Jetzt wird noch eine Schaltung benötigt, die bei Anlegen der Dualzahl an zwei Eingängen (wie immer ein Eingang pro Bit) die richtige der vier Ausgangsleitungen aktiviert. Eine solche Schaltung heißt Decodierer und wird nebenstehend gezeigt. Für jeden Ausgang findet sich in der Ergebnisspalte der Wahrheitswerttabelle genau eine 1; die DNF jedes Ausgangs besteht also nur aus einem einzigen Minterm, der durch das entsprechende UND-Gatter vertreten ist.

Aufbau der arithmetisch-logischen Einheit (ALU)

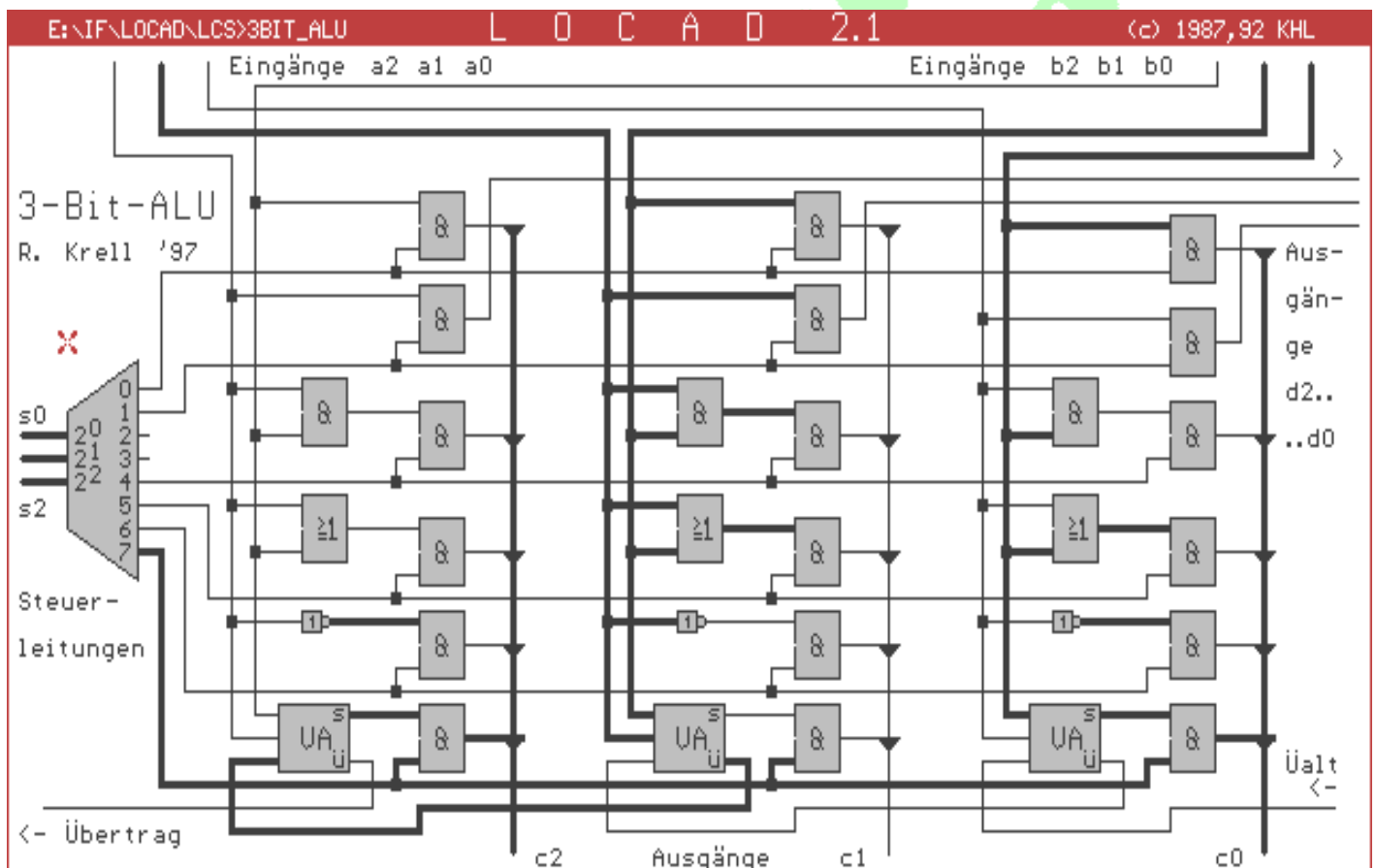
Für jede Stelle der zu verarbeitenden Zahlen werden die vier Verarbeitungseinheiten NOT, OR, AND und Addierer sowie ein Decodierer zusammengepackt. Alle vier Verarbeitungseinheiten verarbeiten die Eingangssignale gleichzeitig. Von den Und-Gattern in den Ausgangsleitungen erhält aber nur eines vom Decodierer die zusätzliche 1, die nötig ist, um das Ergebnis der so ausgewählten Verarbeitungseinheit endgültig zum



Ausgang

des gesamten Bauteils gelangen zu lassen. 8, 16, 32 oder eine der Bus-/Zahlenbreite des Computers entsprechende Anzahl solcher Gesamtbauteile ergeben, parallelgeschaltet, die arithmetisch-logische Einheit der Maschine, die -- zusammen mit mindestens einem Speicher-Register für Ergebnisse -- das Rechenwerk des Computers bildet, das alle Verarbeitungen durchführt.

Die nachfolgende Abbildung zeigt eine mit dem Simulationsprogramm LOCAD aufgebaute arithmetisch-logische Einheit, die jeweils zwei 3-Bit-Dualzahlen verarbeiten kann. Außer den logischen Befehlen 4) AND, 5) OR und 6) NOT und dem arithmetischen Befehl 7) ADD beherrscht diese ALU noch die beiden Datentransportbefehle 0) LAD und 1) TRF. Die Befehlsnummern 0) bis 7) müssen -- als Dualzahlen 000 bis 111 -- an die Steuerleitungen links angelegt werden, um über den Decodierer (dessen Aussehen im Simulationsprogramm etwas von der üblichen Notation abweicht) den gewünschten Befehl auszuwählen, wobei allerdings den Nummern 2) und 3) keine Funktion zugeordnet wurde. Grundsätzlich ist die Nummerierung der Befehle willkürlich; hier folgt sie der Nummerierung des 1_AMOR-Modellrechners. Im dargestellten Fall werden gerade die beiden Dualzahlen $a=010$ und $b=011$ zu $c=101$ addiert, weil durch $s=111$ die Addition (ADD) ausgewählt wurde. Ein alter Übertrag, der bereits bei der niederwertigsten Stelle zu berücksichtigen wäre, wurde rechts unten nicht eingegeben.

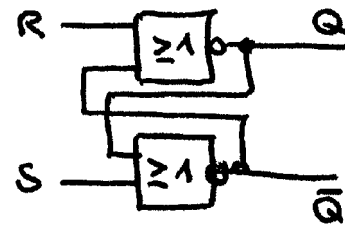


Üblicherweise arbeitet die ALU mit Registern zusammen, d.h. mit Speichern, die die erzeugten Zwischenergebnisse aufnehmen können bzw. benötigte Operanden liefern. Hier würden die Eingänge a vom Akkumulator und die Eingänge b vom Speicherregister beliefert, während die Ausgänge c zum Akkumulator und die Ausgänge d zum Speicherregister führen (vgl. Architektur des Modellrechners 1_AMOR, s.u. Seite 8).

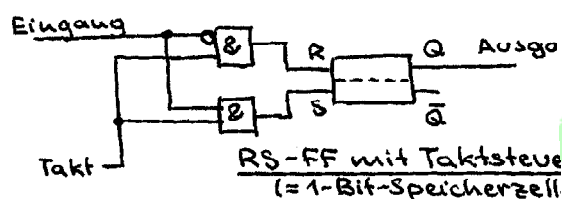
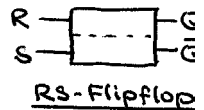
Der Aufbau von Speicherzellen wird im nächsten Kapitel bei den "Schaltwerken" erläutert.

Schaltwerke

Flipflop: Wurden bisher bei den Schaltnetzen die Eingänge immer unabhängig von den Ausgängen beschaltet, so werden bei Schaltwerken Ausgangssignale auf die Eingangsleitungen zurückgeführt. Diese Rückkopplung erlaubt es der nachfolgenden Schaltung aus zwei kreuzgekoppelten NOR-Gattern, auch dann jeweils einen von zwei stabilen Zuständen zu behalten, wenn das ursprünglich auslösende Eingangssignal nicht mehr anliegt: die Rückkopplung übernimmt dessen Aufgabe. Das entstandene RS-Flipflop wäre damit bereits als 1-Bit-Speicherbaustein geeignet. Eine zusätzliche Beschaltung der Eingänge erreicht, dass das Flipflop immer nur



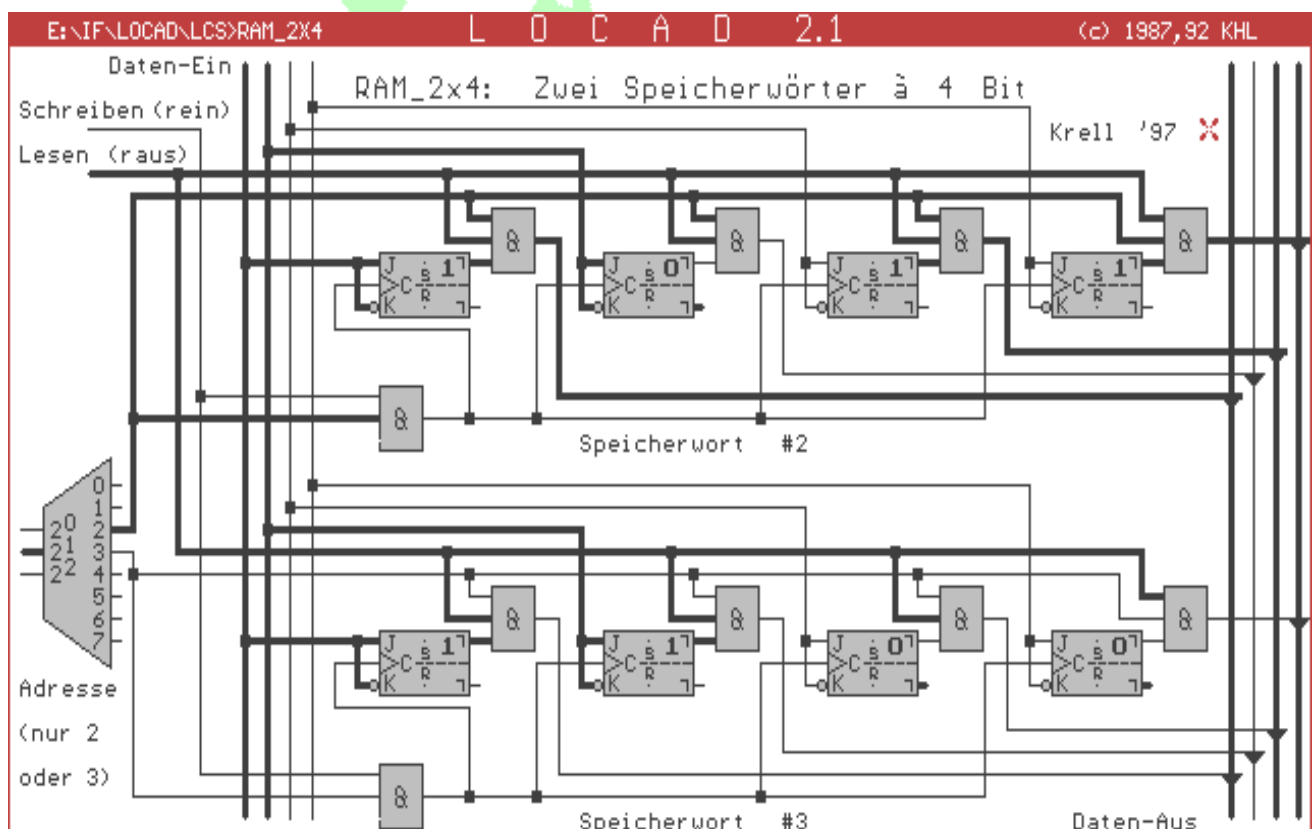
R	S	
0	0	Inhalt halten
0	1	Setzen: Q:=1
1	0	Rücksetzen: Q:=0
1	1	(unerlaubt)



beim Eintreffen eines "Takt"-Signals (z.B. aus dem Adress-Decodierer) die Information einer Eingangsleitung übernimmt und diese beliebig lange -- bis zum nächsten Taktsignal -- behält. Am Ausgang Q kann jederzeit der Speicherinhalt abgefragt werden, ohne diesen zu zerstören. Eine solche Speicherzelle ist auch unter dem Namen Data-Latch bzw. D-Auffang-Flipflop

bekannt und links abgebildet.

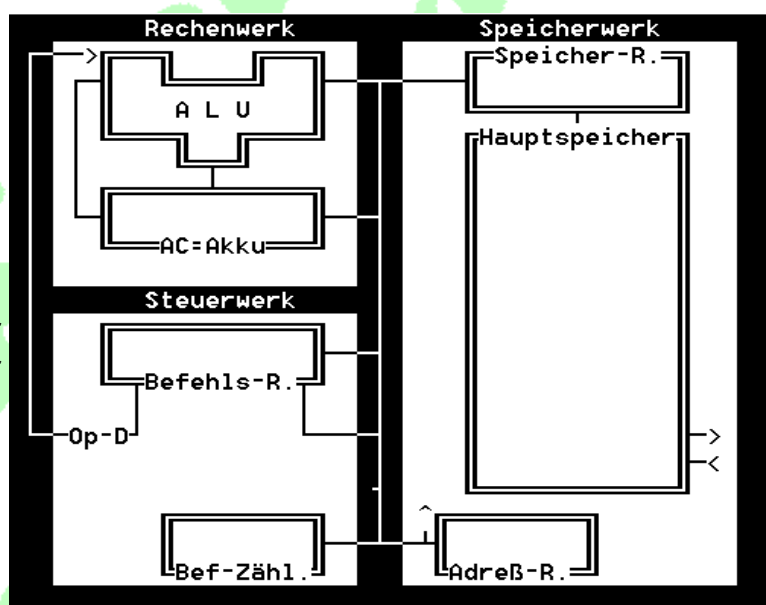
Speicher: Beim Computerspeicher sind üblicherweise acht 1-Bit-Speicherzellen parallel angeordnet (und werden durch den Adress-Decodierer gleichzeitig an- bzw. ausgewählt und durch acht parallele Leitungen gleichzeitig mit Daten versorgt bzw. abgefragt. Der Bildschirmausdruck einer mit dem Simulationsprogramm LOCAD erzeugten Modellschaltung zeigt zwei Blöcke von jeweils nur vier [statt acht] parallelen 1-Bit-Speicherzellen, die gemeinsam



adressiert und zum Lesen oder Schreiben aktiviert werden können. Hat der Hauptspeicher eines Personalcomputers eine Kapazität von 640 kByte = 640 x 1024 Byte = 640 x 1024 x 8 Bit = 5.242.880 Bit, so müssen über 5,2 Millionen solcher Speicherbausteine vorhanden sein. Dank heutiger Miniaturisierung und Fertigungstechnologie gelingt die Unterbringung zum Glück auf wenigen Chips, so dass inzwischen 16 oder mehr MByte (1 MByte = 1024x1024 Bytes \approx 1,05 Mio. Bytes) Speicherkapazität üblich und erschwinglich sind (16 MB = 134 Mrd. 1-Bit-Speicherzellen). Für den Hauptspeicher werden allerdings i.a. nicht unbedingt die hier beschriebenen Flipflops, sondern auch pro Bit ein Kondensator eingesetzt, der aufgeladen eine "1" und ungeladen eine "0" repräsentiert. Die geladenen Kondensatoren müssen regelmäßig und rechtzeitig nachgeladen werden ("refresh"), bevor ihre Spannung unter den für "1" festgelegten Mindestwert sinkt. Millionen von Kondensatoren in Halbleiterbauweise finden auf einem Speicherchip Platz.

Selbst für den recht bescheidenen Hauptspeicher des Modellrechners 1_AMOR wären -- noch ohne alle Register -- immerhin 640 Flipflops bzw. Speichereinheiten nötig, von denen allerdings jeweils 10 gemeinsam adressiert und an 10 parallele Datenleitungen angeschlossen würden.

*Architektur (Aufbau) des
Modellrechners 1_AMOR
Der Hauptspeicher besteht aus
64 Zellen à 10 Bit (10-Bit-
Speicher-
worten), nummeriert von 0 bis 63.
Z. 62: Bildschirmspeicher
Z. 63: Tastaturpuffer*



dual-dez--mnem.----Beschreibung des Maschinenbefehls-----

0000	0)	LAD n	Lade (aus dem Speicher in den Akku) AC := Mem[n]
0001	1)	TRF n	Transferiere (Speichere) Mem[n] := AC; AC := 0
0010	2)	PUSH n	Kellere ein Mem[KZgr] := AC; KZgr := KZgr-1
0011	3)	POP n	Kellere aus KZgr := KZgr+1; AC := Mem[KZgr]
0100	4)	AND n	Verknüpfe bitweise mit "Und" AC := AC AND Mem[n]
0101	5)	OR n	Verknüpfe bitweise mit "Oder" AC := AC OR Mem[n]
0110	6)	NOT n	Invertiere bitweise AC := NOT(AC) { = -1-AC }
0111	7)	ADD n	Addiere AC := AC + Mem[n]
1000	8)	SUB n	Subtrahiere AC := AC - Mem[n]
1001	9)	SHL n	Schiebe links um Mem[n] Stellen AC:=AC*(2^Mem[n])
1010	10)	SPR n	Springe (unbedingt) GOTO n
1011	11)	SP< n	Springe bei negativem AC-Inhalt IF AC<0 THEN GOTO n
1100	12)	SP= n	Springe bei null IF AC=0 THEN GOTO n
1101	13)	CALL n	Starte Unterprogramm ab n (wie Basic: GOSUB n)
1110	14)	RET n	Kehre zurück vom Unterprogramm (wie Basic: RETURN)
1111	15)	STOP n	Beende die Ausführung des Maschinenprogramms HALT

wobei: AC Inhalt des Akkumulators, Mem[n] Inhalt der n-ten Speicher-
stelle, KZgr Kellerzeiger, n Adresse (0..63), Inhalt -512 .. 511