Logeleien (Teil 1)

In vielen Basic-Programmen tauchen sie auf, die logischen Operatoren wie NOT, AND und OR. Bei sinnvollem Einsatz kann man eine Menge Basic-Speicherplatz sparen. Aber auch Ausdrücke wie IF A THEN ... bringen nicht nur den Anfänger manchmal ins Schleudern. Was steckt eigentlich im Endefekt dahinter?

Das hätte sich der »Vater der Logik«, Aristoteles, vor 2200 Jahren nicht träumen lassen können, was aus seinen Gedankengängen einmal werden würde. Man kann aber mit einiger Berechtigung behaupten, daß es ohne seine Werke und die von Leibniz (vor etwa 300 Jahren hat dieser die Logik modernisiert) keine Computer gäbe. Die Arbeiten der Begründer der mathematischen Logik, Boole (1815-64) und Frege (1848-1925), basieren auf dem jahrtausende alten Fundament. Wie für den Computer geschaffen — aber eigentlich ist es umgekehrt: Die logische Maschine ist ein Produkt der mathematischen Logik — ist die sogenannte zweiwertige Logik. Hier geht es um »Wahr« und »Falsch«, ein drittes gibt es nicht (tertium non datur, wie die alten Philosophen zu sagen pflegten). Die Verwandtschaft liegt auf der Hand: Die elektronische Maschine kennt die Zustände Strom an/Strom aus oder Ladung vorhanden/nicht vorhanden oder eben einfach 1 und 0, die Ziffern der Binärarithmetik.

Dazu noch eine technische Bemerkung: Sollten Sie mit dem System der binären Zahlen noch nicht vertraut sein, dann lesen Sie den beigefügten Kasten, in dem die wichtigsten Begriffe erklärt sind.

1. Wahrheitswerte und Aussagen

Was sind Aussagen? Am besten sehen wir uns einfach einige Beispiele an:

— Rosen sind rot
— Die Zahl ist größer als 100.

Das sind einfache Aussagen. Eine zusammengesetzte Aussage ist beispielsweise:

Rosen sind rot **UND** Veilchen sind blau. Hier wurden zwei Aussagen UND-verknüpft. Keine Aussage ist:

— Wohin gehst Du?

Ich hoffe, daß auf diese Weise ohne tiefschürfende philosophische Erörterungen (denn die gibt es natürlich) das Wesen von Aussagen klar geworden ist.

Aussagen können wahr sein. Sie enthalten dann den Wahrheitswert »Wahr«. Sie können aber auch statt dessen falsch sein, was ihnen den Wahrheitswert »Falsch« verleiht. Der Wahrheitswert von zusammengesetzten Aussagen hängt zum einen von den Wahrheitswerten der Einzelaussagen ab und zum anderen von der Art ihrer Verknüpfung. Wir werden im folgenden für die Wahrheitswerte die Buchstaben W (für Wahr) und F (für Falsch) verwenden.

In der binären Arithmetik entspricht die 1 dem W und die 0 dem F. Keine Angst, falls Ihnen das noch unverständlich erscheint, wir werden noch allerlei Logeleien mit Binärzahlen treiben. Dann wird Ihnen das ganz geläufig sein. Was uns momentan interessiert, ist, wie der Computer Wahrheitswerte angibt. Dazu machen wir einen kleinen Test mit dem folgenden Programm:

```
10  INPUTA,B
20  C=(A=B)
30  PRINTA,B,C
```

In Zeile 20 ist eine Aussage enthalten: A=B. Wenn Sie A und B so eingegeben haben, daß beide gleich sind, dann ist diese Aussage wahr (W), und in C finden Sie den Wahrheitswert, den der Computer für W verwendet. Probieren Sie dies mal aus: Sie finden —1! Wenn aber A und B verschiedene Zahlen waren, ist die Aussage A=B falsch, und in C steht 0, was die Computerform von F ist. Weshalb —1 als Wahrheitswert des Commodore 64 für W? Das liegt daran, daß unser Computer für den Wahrheitswert ein ganzes Byte reserviert. Das schreibt er im Falle W voller Einsen:

1111 1111

Nach den Regeln von vorzeichenbehafteten Binärzahlen ist das aber die Erscheinungsform von —1.

2. Das merkwürdige IF A THEN ...

Bei IF..THEN-Verzweigungen wird vom Computer der gesamte Ausdruck zwischen IF und THEN auf seinen Wahrheitswert untersucht (das kann dann eine Aussage sein, aber auch etwas anderes, zum Beispiel eine Variable). Dabei nimmt der C 64 alles für wahr an, was bei dieser Untersuchung ungleich Null ist. Wenn also in einer Verzweigung der Form:

IF A THEN ...

A schon irgendeinen Wert angenommen hat, der ungleich Null ist, wird die Bedingung als erfüllt angesehen. Probieren Sie mal mit verschiedenen Eingaben für A:

```
10  INPUT A
20  IF A THEN PRINT "TEST"
30  PRINT A
```

In der uns geläufigeren Form dieser Verzweigung tauchen zwischen IF und THEN Aussagen in mathematischer Form auf, in denen die Vergleichssymbole verwendet werden, zum Beispiel:

IF A > = 100 THEN GOTO ...

3. Rechnen mit Wahrheitswerten

Außerhalb solch einer Verzweigung ist es manchmal gar nicht so einfach festzustellen, ob es sich bei einem Ausdruck um eine Aussage dreht oder vielleicht nur um eine Zuweisung. Es kommt hier sehr auf den Zusammenhang mit dem übrigen Programmtext an. Beispielsweise kann A=B sowohl eine Zuweisung sein (LET A=B) als auch eine Aussage (C=(A=B)). In Programmen, in denen viel mit logischen Operatoren gearbeitet wird, empfiehlt sich deshalb — um mehr Klarheit zu schaffen —, die vom C 64 verstandene, aber so gut wie nie verwendete Basic-Vokabel LET bei Zuweisungen zu verwenden.

Ein noch eklatanteres Beispiel ist enthalten in A=5+2. Das ist offensichtlich keine Aussage, könnte deutlicher aber noch als LET A=5+2 geschrieben werden. In einem anderen Zusammenhang ist es eine Aussage:

C=4+(A=5+2)

Ist hier A=5+2 wahr (wenn A=7), ergibt sich ein C von 3, weil ja der Wahrheitswert der Aussage —1 beträgt. Im anderen Fall ist C=4. Man kann mit Aussagen — besser gesagt, mit ihren Wahrheitswerten — also auch rechnen.

Im folgenden Programmbeispiel sind gleich fünf Wahrheitswerte miteinander rechnerisch verknüpft. Es handelt sich um eine kurze Routine zur Umrechnung von ASCII-Code in Bildschirmcode:

```
10  INPUT"ASCII-WERT";A
20  C=A-161-33*(A<255)-64*(A
<192)-32*(A<160)+32*(A<96)-
64*(A<64)
30  PRINT"ASCII=";A,"POKE-
CODE=";C
```

Beispiel: ASCII=65, Bildschirmcode=1

Aussagen können auch mit Strings gebildet werden. Auch dabei ergeben sich die Wahrheitswerte —1 oder 0. Allerdings lassen sich hier sinnvoll nur die Vergleichsrelationen »=« und »ungleich« einsetzen. Alle anderen »größer als«, »kleiner als« etc. sind mit Vorsicht zu genießen. Zur Ermittlung des Wahrheitswertes werden nämlich von links nach rechts alle Buchstaben beider Strings in Form ihrer ASCII-Werte miteinander verglichen. Die erste auf diese Weise gefundene Abweichung führt dann zum Ergebnis. Ein Beispiel soll das Problem verdeutlichen:

Die Aussage (A$<B$) soll uns einen Wahrheitswert liefern. Wenn nun A$="DZZZ" und B$="EAAA" definiert wurden, ergibt sich —1 als Wahrheitswert, weil der ASCII-Wert von D kleiner ist als der von E. Der Rest der Strings spielt keine Rolle mehr.

4. Die logischen Operatoren in Basic

Drei Operatoren haben wir im Normalfall zur Verfügung für die sogenannte Boolèsche Algebra, wie man manchmal auch das Rechnen mit logischen Operatoren nennt:

NOT, AND, OR

Die Tabelle 1 zeigt, wo in der Prioritätenreihenfolge aller Operatoren diese drei einzuordnen sind. Ebenso wie bei den arithmetischen Operatoren (+,*,↑ etc.) kann auch hier die Reihenfolge durch sinnvolle Anwendung von Klammern verändert werden.

Bemerkung	Operatoren	Beispiel
Höchste Priorität	↑	A↑B
	*, /	A*B, A/B
	+, —	A+B, A-B
	NOT	NOT A
	AND	A AND B
Niedrigste Priorität	OR	A OR B

Tabelle 1. Reihenfolge der Priorität von Operatoren

5. Wahrheitstabellen

Jede Verkoppelung von Aussagen (beziehungsweise ihren Wahrheitswerten) mittels logischer Operatoren ergibt ebenfalls wieder Wahrheitswerte. Wir hatten vorhin schon er-

wähnt, daß das Ergebnis abhängt:
a) von den Wahrheitswerten der zu verknüpfenden Aussagen und
b) von der Art der Verknüpfung.

Das kann manchmal schon reichlich schwer zu durchschauen sein, was welche Kombination ergibt. Deshalb schreibt man sich eine Übersicht, die alle möglichen Ausgangs-Wahrheitswerte enthält und die nach der Verknüpfung entstandenen Wahrheitswerte. Solche Tabellen nennt man Wahrheitstabellen. Wir werden diese Tabellen im folgenden bei der Erklärung der logischen Operatoren verwenden. Wie man sie erstellt und liest, wird Ihnen schnell geläufig sein.

Noch eine Bemerkung: Die Anwendung der logischen Operatoren auf Aussagen und auf Zahlen wird bei uns — aus Gründen der besseren Verständlichkeit — getrennt behandelt werden. Das ist eigentlich nicht nötig, denn im Grunde genommen sind ja die Wahrheitswerte von Aussagen auch nur Zahlen.

6. Die Anwendung von NOT auf Aussagen

Der NOT-Operator erzeugt immer das Gegenteil des Wahrheitswertes der Ausgangsaussage. Wenden wir also NOT auf eine Aussage mit dem Wahrheitswert W an, dann ergibt das den Wert F. Dasselbe geschieht im umgekehrten Fall.

Dazu wollen wir uns eine Wahrheitstabelle schreiben (Tabelle 2). In der linken Spalte sind die beiden möglichen Wahrheitswerte einer Aussage A aufgeführt. Die rechte Spalte ergibt sich durch Anwendung von NOT auf die Werte der linken Spalte:
NOT W = F
NOT F = W

A	NOT A
W	F
F	W

Tabelle 2. Wahrheitstabelle: NOT bei Aussagen

Das ist gar nicht so schwer, nicht wahr? An einem Beispiel wollen wir uns das auch noch mal ansehen. Die Zeile:
10 A=(3=5):PRINT NOT A
ergibt den Wert —1. Die Aussage 3=5 ist ja falsch (F oder für den Computer 0) und A deshalb 0. NOT A ergibt dann das Gegenteil, also W oder —1.

Eine Basic-Zeile wie die folgende:
10 A = NOT A
kann man wie einen Flip-Flop-Schalter einsetzen, falls man den Wahrheitswert von A zuvor als 0 oder —1 definiert hat. Jedesmal, wenn das Programm über diesen Ausdruck läuft, kippt der Wert von A.

7. NOT auf Zahlen angewendet

Zahlen oder Variable, auf die der NOT-Operator angewandt wird, verkehren sich in ihr Einerkomplement. Falls Sie mit den Begriffen »Einerkomplement« und »Zweierkomplement« Schwierigkeiten haben, weise ich Sie nochmal auf den hier abgedruckten Kasten mit Erläuterungen hin. Jedes Bit wird also »gekippt«. Aus einer 0 wird 1 und umgekehrt. Die Wahrheitstabelle kann hier ähnlich aufgestellt werden wie für Aussagen, nur tauchen hier anstelle von W und F nun 1 und 0 (also die Bitwerte) auf (siehe Tabelle 3).

A	NOT A
1	0
0	1

Tabelle 3. Wahrheitstabelle: NOT bei Zahlen

Eine Anwendung von NOT wäre beispielsweise das gezielte Kippen von Bytes oder 2-Byte-Werten. Besonders bei Sprite-Programmen ist sowas denkbar. Nehmen wir mal an, wir hätten acht Sprites definiert, von denen vier bis zu einem bestimmten Ereignis sichtbar sein sollen. Die anderen vier sind solange abgeschaltet. Wenn das Ereignis (Tastendruck oder Zusammenstoß ...) eintritt, werden die obersten vier ab- und gleichzeitig die anderen vier angeschaltet. Register 21 (53269) des VIC-II-Chip stellt für jedes Sprite ein Schalt-Bit zur Verfügung (0 steht dort für ein ausgeschaltetes und 1 für ein aktiviertes Sprite). Sollen also zuerst die Sprites 0, 2, 4, 6 an- und 1, 3, 5, 7 abgeschaltet sein, dann enthält Register 21 das Bit-Muster:
0101 0101

Darauf nun eine NOT-Operation angewendet ergäbe genau das erwünschte Kippen:
1010 1010 nach NOT.

Es läge daher nahe, etwa so zu programmieren:
POKE 53269,NOT PEEK(53269)

Leider ist das nicht möglich. Versucht man es trotzdem, erhält man einen ILLEGAL QUANTITY ERROR. Woran liegt das? Die Ursache dafür liegt wieder in der Eigenart, wie der C 64 Integer-Zahlen interpretiert. Zwar erzeugt die NOT-Operation das Einerkomplement der Ausgangszahl, verstanden wird dieses Ergebnis aber als Zweierkomplementzahl. Daraus folgt, daß man mit der obigen Anweisung versucht, eine negative Zahl in die Speicherstelle zu POKEn, und da streikt unser Computer. Die Binärzahl 1010 1010 wird als —86 verstanden. Was kann man tun, daß der wahre Wert, also 170, in den Speicher gePOKEt wird? Bei 1-Byte-Zahlen ist das relativ einfach: Man wandelt sie durch Abziehen von 256 in eine 2-Byte-Zahl um, also NOT(A-256). In Bild 2 ist zu sehen, was dabei geschieht.

Ein anderer Weg für 1-Byte-Zahlen (also Zahlen, die kleiner als 255 sind) ist NOT A AND 255, was wir aber erst später verstehen werden, wenn wir die AND-Operation kennengelernt haben. Vorsicht muß man walten lassen, wenn auch negative A-Werte auftreten. Negative A können ohne Probleme der NOT-Operation unterzogen werden, weil das Ergebnis in jedem Fall positiv ist. Verwendet man NOT(A-256), kommt es zu Fehlern. Der Weg über NOT A AND 255 ist auch mit negativen 1-Byte-Zahlen möglich. Am sichersten wäre also diese Basic-Zeile:
POKE53269,NOTPEEK(53269) AND255

2-Byte-Zahlen spielen bei POKEs keine Rolle, weshalb wir auf die Umwandlung der negativen Zahl in den vorzeichenfreien Ausdruck hier nicht eingehen werden. Eine andere Sache muß bei der Anwendung von NOT noch bedacht werden: Diese Operation arbeitet mit Integer-Zahlen. Treten nun als Argumente »Komma-Zahlen« auf, dann führt der C 64 zuerst eine INT-Operation aus, die unter Umständen zu Fehlern führen kann: 3999 ergibt dann ein deutlich anderes Ergebnis als 4001.

Die NOT-Operation kann nur auf Zahlen, nicht aber auf Strings angewendet werden. Allerdings erfolgt die computerinterne Speicherung von Buchstaben ja als ASCII-Werte, also auch als Zahlen. Wenn wir daher die ASCII-Codes eines Textes der NOT-Verknüpfung unterziehen, haben wir auch Strings dieser Operation zugänglich gemacht. Genau das tut ein kleines Demo-Programm (Programm Logik-1): Es ver- und entschlüsselt Texte mittels NOT.

In diesem Demo-Programm wird Text Buchstabe für Buchstabe in die ASCII-Werte übertragen und diese dann durch NOT komplementiert und in C$ gespeichert. Der Ausdruck ist dann ein unverständliches Kauderwelsch, das aber durch eine zweite NOT-Operation wieder lesbar wird.

8. Eine kleine Hilfe

Als Programm Logik-2 finden Sie eine kleine Denk- und Rechenhilfe hier abgedruckt. Dieses Programm kann Zahlen zwischen —32767 und +32767 den logischen Verknüpfungen unterziehen und zeigt Ihnen dabei, was sich im Binärformat abspielt und wie die Argumente — falls negative Zahlen auftreten — im vorzeichenfreien Format aussehen. Nach dem RUN bietet Ihnen ein Menü fünf Möglichkeiten an: NOT, AND, OR, EOR und Programmende. NOT kennen Sie ja schon, die anderen logischen Operatoren werden Ihnen in der nächsten Folge geläufig werden. Dort wird Ihnen dieses Programm dann einige Hilfestellungen geben für etwas kompliziertere Sachverhalte. Wenn Sie eine Option angewählt haben, werden zwei Zahleneingaben verlangt (bei NOT nur eine) und dann die entsprechenden Binärzahlen, die vorzeichenfreien Formen und das Ergebnis der Verknüpfung ausgedruckt. Innerhalb der gesteckten Grenzen sind alle Zahleneingabe-Formate möglich. Ein Tastendruck bringt Sie wieder ins Menü zurück.

In der nächsten Folge werden wir uns die Operatoren AND und OR ansehen. Ein in dem WAIT-Befehl versteckter weiterer Operator EOR soll aufgespürt werden, und wir werden wieder einige Anwendungen der Boole'schen Algebra in unserem Computer kennenlernen.

(Heimo Ponnath/gk)

```
10 REM------------------------------------  <102>
20 REM        LOGIK - 1                     <119>
30 REM------------------------------------  <122>
40 :                                        <098>
50 PRINT"{CLR,7SPACE}GEHEIMCODE MIT NOT"    <071>
60 PRINT:PRINT                              <168>
70 PRINT" GEBEN SIE EIN WORT EIN";          <164>
100 INPUT A$:L=LEN(A$):DIM A(L),B(L),B$(L)
    :K=0                                    <062>
110 C$=""                                   <203>
120 FOR I=1 TO L:A(I)=ASC(MID$(A$,I,1)):B(
    I)=NOT A(I)-257:B$(I)=CHR$(B(I)):NEXT
    I                                       <190>
130 FOR I=1 TO L:C$=C$+B$(I):NEXT I         <136>
140 IF K=0 THEN:PRINT:PRINT"TEXT{11SPACE}=
    ";A$                                    <247>
145 IF K=1 THEN:PRINT:PRINT"GEHEIMCODE{5SP
    ACE}= ";A$                              <130>
150 K=K+1:IF K=1 THEN A$=C$:GOTO 110        <038>
160 PRINT:PRINT"ENTSCHLUESSELT = ";C$       <101>
```

Programm Logik-1. Mit NOT eine Geheimschrift erzeugen

```
1 REM ********************************** <250>
2 REM *                                * <229>
3 REM *     LOGISCHE  OPERATIONEN      * <120>
4 REM *                                * <231>
5 REM * ZUM VERFOLGEN IM BINAERFORMAT * <156>
6 REM *         NOT,AND,OR,EOR         * <184>
7 REM *                                * <234>
8 REM *  HEIMO PONNATH  HAMBURG 1985   * <082>
9 REM *                                * <236>
10 REM********************************** <003>
15 PRINT CHR$(147):GOTO 200 <177>
20 REM ***** UP-CURSOR SETZEN ***** <208>
25 POKE 211,S:POKE 214,Z:SYS 58640:RETURN <088>
30 REM ***** UP-EINGABETEST ******* <162>
35 E=ABS(E):A=(E>=0):B=(E<255) <207>
40 IF A AND B THEN M=8:RETURN <139>
45 A=(E>=255):B=(E<32768) <054>
50 IF A AND B THEN M=16:RETURN <196>
55 M=0:RETURN <046>
60 REM ***** UP-EINGABE 1 ********* <005>
65 S=1:Z=5:GOSUB 25:PRINT CHR$(28)"WELCHE
   ZAHL SOLLS DENN SEIN?" <192>
70 Z=7:GOSUB 25:PRINT"SIE MUSS ZWISCHEN -3
   2767UND +32767 SEIN" <226>
75 Z=9:GOSUB 25:INPUT W:E=W:GOSUB 35:IF M=
   0 THEN POKE 781,9:SYS 59903:GOTO 75 <211>
80 PRINT CHR$(30):RETURN <044>
85 REM ***** UP-EINGABE 2 ********* <031>
90 S=1:Z=5:GOSUB 25:PRINT CHR$(28)"GEBEN S
   IE NUN DIE BEIDEN ZAHLEN EIN." <074>
95 S=1:Z=7:GOSUB 25:PRINT"SIE MUESSEN ZW.
   -32767 U. +32767 SEIN!" <116>
100 Z=9:GOSUB 25:INPUT"ERSTE ZAHL =";W1:E=
    W1:GOSUB 35:M1=M <043>
105 IF M=0 THEN POKE 781,9:SYS 59903:GOTO
    100 <156>
110 Z=11:GOSUB 25:INPUT"ZWEITE ZAHL=";W2:E
    =W2:GOSUB 35 <254>
115 IF M=0 THEN POKE 781,11:SYS 59903:GOTO
     110 <208>
120 M=16+8*((M1=8)AND(M=8)):PRINT CHR$(30)
    :RETURN <195>
125 REM **** UP-UMRECHNUNG DEZ/BIN **** <021>
130 BI$="":DI=DE <041>
135 DI=DI/2:D$="0":IF DI<>INT(DI)THEN D$="
    1" <121>
140 DI=INT(DI):BI$=D$+BI$:IF DI>0 THEN 135 <045>
145 IF LEN(BI$)<M THEN BI$="0"+BI$:GOTO 14
    5 <017>
150 RETURN <036>
155 REM **** UP-UMRECHNUNG BIN/DEZ **** <051>
160 DE=0:FOR I=1 TO M:IF MID$(A$,I,1)="1"T
    HEN DE=DE+2↑(M-I) <168>
165 NEXT I:RETURN <056>
170 REM **** UP-BILDSCHIRMAUSGABE ***** <094>
175 PRINT CHR$(158)TAB(5)DE;TAB(32-M)BI$;C
    HR$(30):RETURN <025>
198 REM <085>
199 REM **** HAUPTPROGRAMM-VARIABLE *** <214>
200 S=0:Z=0:E=0:A=0:B=0:M=0:W=0:W1=0:W2=0:
    M1=0:I=0:K=0:X=0 <142>
205 DI=0:DE=0:C=0:V=0 <178>
210 BI$="":D$="":A$="":B$="":C$="":Z$="---
    ----------":BE$="" <087>
220 DIM DE(3),BI$(3) <206>
299 REM **** HAUPTPROGRAMM-MENUE ****** <237>
300 POKE 53280,0:POKE 53281,0:PRINT CHR$(1
    47)CHR$(30) <084>
305 Z=3:S=2:GOSUB 25:PRINT CHR$(18)" DIE L
    OGISCHEN BEFEHLE IN BINAERFORM " <125>
310 Z=7:GOSUB 25:PRINT TAB(10)"NOT"TAB(25)
    "1" <133>
315 Z=9:GOSUB 25:PRINT TAB(10)"AND"TAB(25)
    "2" <111>
320 Z=11:GOSUB 25:PRINT TAB(10)"OR"TAB(25)
    "3" <108>
325 Z=13:GOSUB 25:PRINT TAB(10)"EOR"TAB(25
    )"4" <185>
330 Z=15:GOSUB 25:PRINT TAB(10)"PROGRAMMEN
    DE"TAB(25)"5" <092>
335 POKE 646,10:Z=20:GOSUB 25:PRINT"BITTE
    WAEHLEN SIE EINEN MENUEPUNKT..."CHR$(3
    0) <222>
340 GET B$:IF B$<"1"OR B$>"5"THEN 340 <045>
345 ON VAL(B$)GOSUB 400,500,600,700,800 <146>
350 Z=23:S=1:GOSUB 25:PRINT CHR$(3)"WEITER
    DURCH TASTENDRUCK..." <232>
355 POKE 198,0:WAIT 198,1:GOTO 300 <026>
399 REM **** OPTION - NOT ********** <140>
400 PRINT CHR$(147):Z=2:S=1:GOSUB 25:PRINT
    CHR$(18)" OPTION{2SPACE}NOT{6SPACE}"C
    HR$(146) <009>
405 BI$="":BE$="":D$="":W=0:DE=0:K=0:GOSUB
    65:IF W<0 THEN 450 <158>
410 DE=W:GOSUB 130 <136>
412 Z=15:S=0:GOSUB 25:GOSUB 175:PRINT TAB(
    15)Z$+" NOT":BE$="" <077>
415 FOR I=1 TO M:D$=MID$(BI$,I,1):IF D$="1
    "THEN D$="0":GOTO 420 <016>
417 D$="1" <049>
420 BE$=BE$+D$:NEXT I:IF K=1 THEN RETURN <236>
425 BI$=BE$:A$=BE$:GOSUB 160:Z=17:S=0:GOSU
    B 25:GOSUB 175:BE$="":RETURN <098>
440 REM **** UP-NEGATIVE BINAERZAHL *** <115>
450 W=NOT W:DE=W:GOSUB 130:K=1:GOSUB 415:A
    $=BE$:GOSUB 160:K=0:IF X=1 THEN RETURN <177>
455 BI$=BE$:BE$="":GOTO 412 <009>
499 REM **** OPTION - AND ********** <210>
500 PRINT CHR$(147):Z=2:S=1:GOSUB 25:PRINT
    CHR$(18)" OPTION{2SPACE}AND{6SPACE}"C
    HR$(146) <079>
502 REM **** UP-EINGABENVERARBEITUNG ** <027>
505 GOSUB 90:IF W1<0 THEN W=W1:X=1:GOSUB 4
    50:X=0:BI$(1)=BE$:DE(1)=DE:BE$="":DE=0 <105>
510 IF W2<0 THEN W=W2:X=1:GOSUB 450:X=0:BI
    $(2)=BE$:DE(2)=DE:BE$="":DE=0 <066>
515 IF W1>=0 THEN DE=W1:GOSUB 130:DE(1)=DE
    :BI$(1)=BI$:DE=0:BI$="" <064>
520 IF W2>=0 THEN DE=W2:GOSUB 130:DE(2)=DE
    :BI$(2)=BI$:DE=0:BI$="" <073>
525 Z=15:S=0:GOSUB 25:DE=DE(1):BI$=BI$(1):
    GOSUB 175 <206>
530 Z=16:GOSUB 25:DE=DE(2):BI$=BI$(2):GOSU
    B 175:IF V=1 THEN RETURN <154>
532 PRINT TAB(15)Z$+" AND":BI$(3)="" <131>
535 FOR I=1 TO M:A=VAL(MID$(BI$(1),I,1)):B
    =VAL(MID$(BI$(2),I,1)):C=A AND B <220>
540 BI$(3)=BI$(3)+RIGHT$(STR$(C),1):NEXT I
    :A$=BI$(3):GOSUB 160:BI$=BI$(3) <100>
545 Z=18:GOSUB 25:GOSUB 175:RETURN <242>
599 REM **** OPTION - OR ************ <047>
600 PRINT CHR$(147):Z=2:S=1:GOSUB 25:PRINT
    CHR$(18)" OPTION{3SPACE}OR{6SPACE}"CH
    R$(146) <130>
605 V=1:GOSUB 505:V=0 <107>
610 PRINT TAB(15)Z$+" OR":BI$(3)="" <159>
615 FOR I=1 TO M:A=VAL(MID$(BI$(1),I,1)):B
    =VAL(MID$(BI$(2),I,1)):C=A OR B <045>
620 BI$(3)=BI$(3)+RIGHT$(STR$(C),1):NEXT I
    :A$=BI$(3):GOSUB 160:BI$=BI$(3) <180>
625 Z=18:GOSUB 25:GOSUB 175:RETURN <066>
699 REM **** OPTION - EOR ********** <174>
700 PRINT CHR$(147):Z=2:S=1:GOSUB 25:PRINT
    CHR$(18)" OPTION{2SPACE}EOR{6SPACE}"C
    HR$(146) <043>
705 V=1:GOSUB 505:V=0 <207>
710 PRINT TAB(15)Z$+" EOR":BI$(3)="" <072>
715 FOR I=1 TO M:A=VAL(MID$(BI$(1),I,1)):B
    =VAL(MID$(BI$(2),I,1)) <047>
720 IF A+B=1 THEN C=1:GOTO 730 <151>
725 C=0 <252>
730 BI$(3)=BI$(3)+RIGHT$(STR$(C),1):NEXT I
    :A$=BI$(3):GOSUB 160:BI$=BI$(3) <034>
735 Z=18:GOSUB 25:GOSUB 175:RETURN <176>
799 REM **** OPTION-PROGRAMMENDE **** <136>
800 PRINT CHR$(147):S=8:Z=12:GOSUB 25:PRIN
    T"DAS WAR'S...TSCHUESS !" <014>
805 Z=22:S=0:GOSUB 25:END <239>
```

Programm Logik-2. Sie sehen, was bei der Verknüpfung von Zahlen mit logischen Operatoren geschieht. Beachten Sie bitte die Eingabehinweise auf Seite 77

Einige Erläuterungen zum Rechnen mit Binärzahlen

a) Dezimales und binäres Zahlensystem

Wir sind es gewohnt, im dezimalen Zahlensystem zu rechnen unter Verwendung der Ziffern 0 bis 9. Die Grundlage unseres Zahlensystems ist die Zahl 10. Reichen die Ziffern 0 bis 9 nicht aus, um eine Zahl (zum Beispiel 9+1) darzustellen, dann führen wir den Stellenwert einer Zahl ein. Hierbei hat jede Stelle in einer Zahl einen Faktor, der sich berechnet aus 10^{Stelle}. Zum Beispiel in der Zahl 132 befindet sich die Ziffer 2 an der nullten Stelle, und sie wird mit dem Faktor 10^0 multipliziert ($10^0=1$). Für unsere Beispielszahl können wir so schreiben:

132 = 1*10^2 + 3*10^1 + 2*10^0 = 1*100 + 3*10 + 2*1

Nach demselben Prinzip wird im Binärsystem verfahren. Hier gibt es allerdings nur die Ziffern 0 und 1. Die Grundlage dieses Systems ist die Zahl 2. Auch hier spielt der Stellenwert die entscheidende Rolle. Der Faktor ist nun aber die entsprechende Zweierpotenz (2^Stelle). Die Binärzahl 110 beispielsweise setzt sich dann so zusammen:

110 = 1*2^2 + 1*2^1 + 0*2^0
= 1*4 + 1*2 + 0*1 = 6

b) Umrechnung Binär- in Dezimalzahl

Damit haben Sie praktisch schon gesehen, wie man Binärzahlen in Dezimalzahlen umrechnen kann: Jeder Stelle entspricht eine Zweierpotenz. Alle Zweierpotenzen, an deren Stelle eine 1 steht, werden zusammengezählt. Dabei gilt: $2^0=1$ (siehe Bild 1).

4. Stelle	3. Stelle	2. Stelle	1. Stelle	0. Stelle
=1*2^4	+1*2^3	+1*2^2	+0*2^1	+1* 2^0 =
= 1*16	+1*8	+1*4	+0*2	+1* 1 =
= 16	+ 8	+ 4	+	1 =
= 29				

Bild 1. Umrechnung einer Binärzahl (11101) in eine Dezimalzahl (29)

c) Umrechnung Dezimal- in Binärzahl

Der umgekehrte Weg läßt sich durch eine Kettendivision der Dezimalzahl durch 2 realisieren. Jeweils die Divisionsreste ergeben eine Stelle der Binärzahl. Beispiel (die Zahl 29):

				Binärzahl
29:	2 =	14	Rest 1	0. Stelle
14:	2 =	7	Rest 0	1. Stelle
7:	2 =	3	Rest 1	2. Stelle
3:	2 =	1	Rest 1	3. Stelle
1:	2 =	0	Rest 1	4. Stelle

Bild 1a. Umrechnung einer Dezimalzahl in eine Binärzahl

Die Kettendivision ist beendet, wenn sich als Ergebnis 0 zeigt. Die Zahl 29 lautet also im Binärformat 11101.

d) Bits und Bytes

Eine Stelle einer solchen Binärzahl nennt man ein »Bit«. Unser Computer ist so gebaut, daß er 8 Bit zu einer Einheit zusammenfaßt. Solch eine Einheit ist ein Byte. Jede Speicherstelle des C 64 faßt genau ein solches Byte. Weil aber die größte in einem Byte darstellbare Zahl binär 11111111 ist — was dezimal 255 entspricht (rechnen Sie mal nach) —, meldet unser Computer einen ILLEGAL QUANTITY ERROR, wenn wir versuchen, eine größere Zahl in eine Speicherstelle zu POKEn.

Der C 64 ist darauf eingerichtet, ganze Zahlen als 2-Byte-Werte (also mit 16 Bit) zu verarbeiten. Die auf diese Weise größte darstellbare Zahl ist 11111111 11111111, was 65535 entspricht. Das ist übrigens genau die Anzahl der Speicherstellen unseres Computers. Allerdings erlaubt der C 64 bei Integerzahlen lediglich Eingaben zwischen 32767 und —32768. Die Ursache dafür ist die Art, wie negative Binärzahlen erzeugt und gespeichert werden.

e) Negative Binärzahlen

Bit 7 (also das ganz links stehende Bit eines Bytes: Die Zählung fängt rechts mit 0 an) einer 8-Bit-Zahl oder Bit 15 einer 16-Bit-Zahl wird als Vorzeichenkennung verwendet. Ist dieses Bit eine 0, dann liegt eine positive, ansonsten eine negative Zahl vor. Die Binärzahl 0111 1111 1111 1111 (32767) ist daher die größte vorzeichenbehaftete 16-Bit-Binärzahl. Wenn also Bit 7 (oder bei der 16-Bit-Zahl Bit 15) eine 1 enthält, handelt es sich um eine negative Zahl. Unser Computer hat aber eine besondere Art, solche negativen Zahlen zu verstehen: Er interpretiert sie als »Zweierkomplementzahlen«. Ein Beispiel, in dem wir die Zahl —29 in dieses Format bringen, soll erläutern, worum es sich dabei handelt:

Schritt 1:

Man berechnet zuerst die Binärzahl der positiven Zahl, also von 29:

0000 0000 0001 1101

Schritt 2:

Davon wird nun das sogenannte Einerkomplement gebildet, wobei jedes Bit in sein Gegenteil verkehrt wird:

1111 1111 1110 0010

Schritt 3:

Aus dem so erhaltenen Einerkomplement bildet man das »Zweierkomplement« durch Addieren einer 1:

```
1111  1111  1110  0010
                    1 +
---------------------
1111  1111  1110  0011
```

Das Ergebnis ist —29 im Zweierkomplement-Format.

Hat man eine Zahl im Zweierkomplementformat vorliegen, dann kann man auf genau demselben Weg ihren Wert feststellen. Das wollen wir uns noch an einer 8-Bit-Zahl ansehen:

Schritt 1:

Es liegt eine Binärzahl vor, die so aussieht:

1111 1110

Schritt 2:

Das Einerkomplement wird gebildet:

0000 0001

Schritt 3:

Das Zweierkomplement wird durch Addieren von 1 erzeugt:

```
0000  0001
          1 +
---------------
0000  0010
```

Schritt 4:

Umrechnen der Binärzahl in die Dezimalzahl ergibt hier den Wert 2. Das Vorzeichen war negativ (Bit 7 ist ja 1 in der ursprünglichen Zahl). Es handelte sich also um die Zahl —2.

256	=0000	0001	0000	0000
Einerkomplement	=1111	1110	1111	1111
+ 1				1
Zweierkomplement	=1111	1111	0000	0000
= —256				
unsere Binärzahl			0101	0101
NOT-Operation:	1111	1111	0101	0101
	0000	0000	1010	1010

↑ Vorzeichenbit = 0 (positiv)

Bild 2. Ablauf der Operation NOT (A—256) bei A=0101 0101