## Technische Informatik in der gymnasialen Oberstufe

## 1. Auszug aus dem Hessischen Lehrplan

### 13.2 Wahlthema Technische Informatik Std.: GK 24 LK 43

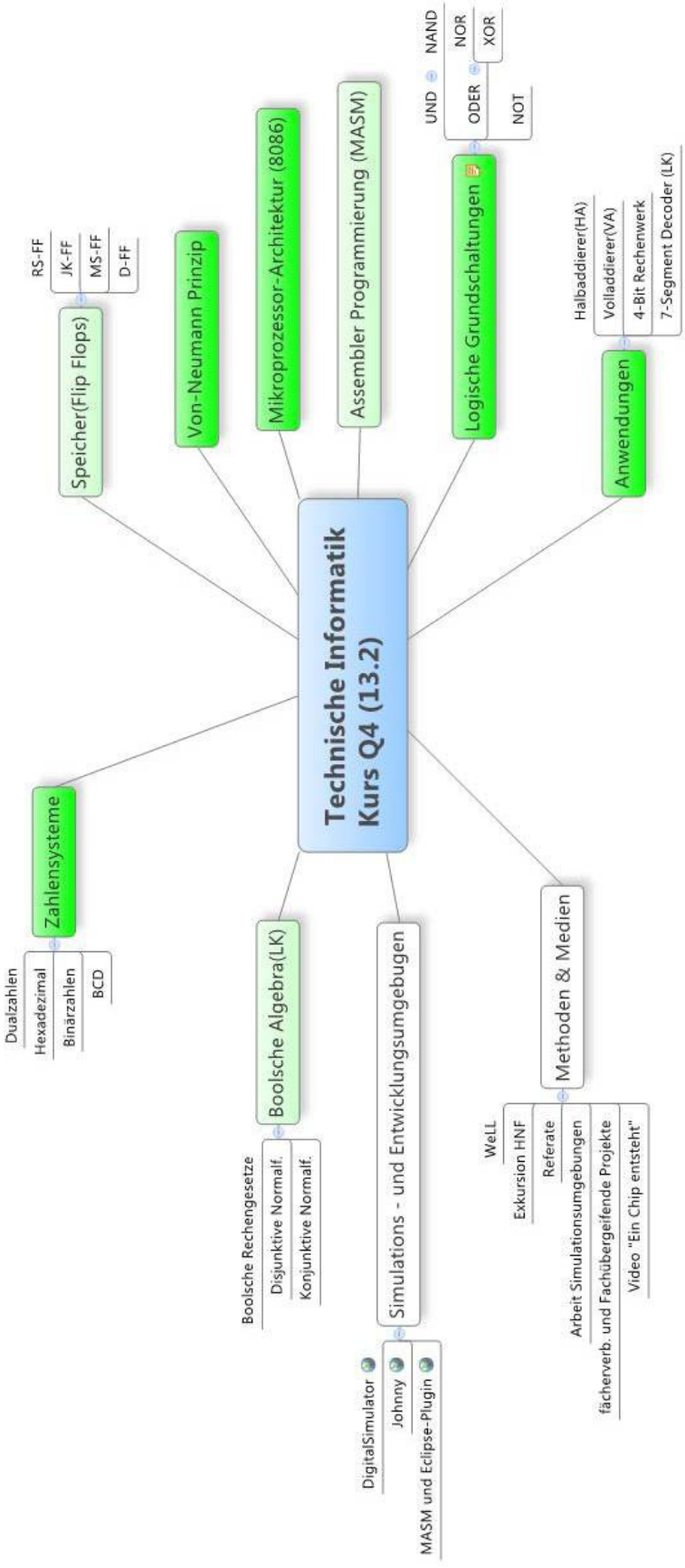
**Begründung:**
Grundlagen aus dem Themenbereich der Technischen Informatik leisten einen Beitrag für das Verständnis der Funktionsprinzipien von Informatiksystemen als symbolverarbeitende, universelle Maschinen. ***Im Vordergrund steht die exemplarische Auseinandersetzung mit den Ideen und den Prinzipien der Computersysteme***, die sie als technisches Produkt ermöglichen. ***Dabei soll ausgehend von den historischen Entwicklungslinien und ihrer Realisierung auf unterschiedlichen technischen Ebenen abstrahiert werden.*** Chancen, Risiken und Folgen bei der Entwicklung zur Informationsgesellschaft und der sich damit verändernden Lebens- und Arbeitsformen lassen sich hierbei aufzeigen.

In der Regel wird man mit den logischen Grundschaltungen beginnen und mit Volladdieren in Form von Blockschaltbildern ein einfaches Addier- und Subtrahierwerk mit Komplementbildung realisieren. Die Schaltnetze können gut mit Simulations-Programmen aufgebaut und analysiert werden. Nach der Behandlung von einfachen Speicher-Bausteinen wird man das „Von Neumann Prinzip“ als Abstraktion und Grundlage moderner Mikroprozessoren einführen. Anhand des Blockschaltbildes und von Simulationen eines Mikroprozessors (z. B. 8086) werden dessen Grundprinzipien erarbeitet. Der Kurs soll mit der Analyse einiger beispielhafter Assemblerprogramme abgerundet werden
.

**Unterrichtsinhalte/Aufgaben:**

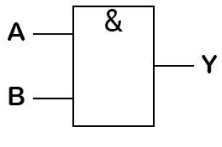
| | |
|---|---|
| Logische Grundschaltungen | AND, OR, NOT, NAND, NOR, EXOR, Arbeitstabelle , Symbol |
| Binärcodierung | Binärcode, BCD-Code, HexCode<br>Subtraktion durch Komplementbildung |
| Einfache Rechenwerke | Halbaddierer, Volladdierer<br>4-Bit Addier- und Subtrahierer |
| Speicherelemente | RS-Flip-Flop, D-Flip-Flop, JK-Master-Slave-Flip-Flop<br>Anwendungen, DEA |
| Von Neumann Prinzip | Funktionseinheiten, Speichermodell, Befehlsabarbeitung |
| Mikroprozessor- und Mikrocomputerarchitektur | Bussysteme, ALU, Register, Flag-Register<br>Steuerwerk, Mikroprogramm<br>BIOS, ROM, RAM |
| Assemblerprogrammierung | Entwicklungswerkzeuge, Beispiele<br>Adressierungsarten<br>Codeschablonen für Schleifen und Fallunterscheidungen<br>Interrupts |
| Technische Realisierung von Grundschaltungen | Relais-, Transistorschaltungen |
| Vereinfachung von Schaltnetzen mit Boolescher Algebra | Gesetze der Booleschen Algebra<br>konjunktive und disjunktive Normalform |

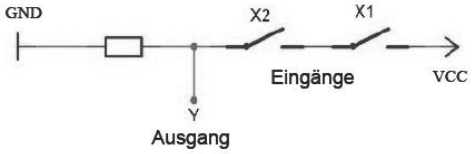
## 2. Advance Organizer

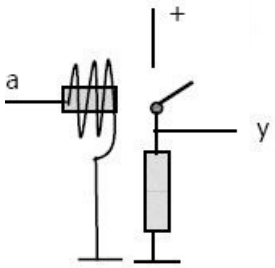


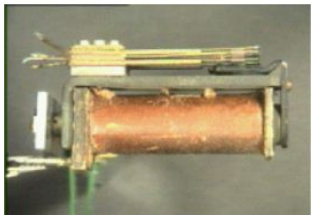
## 3. Logische Grundschaltungen

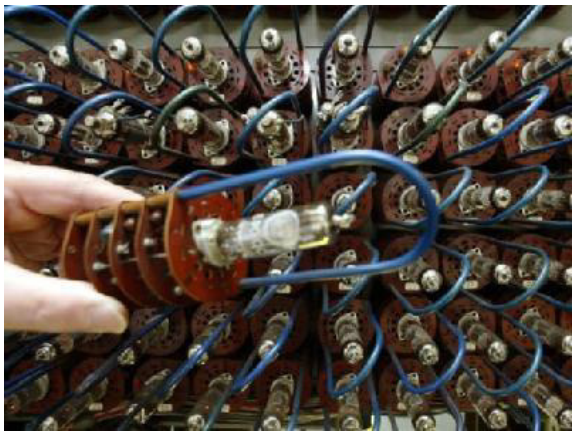
### 3.1 Technische Realisierung eines UND-Gliedes

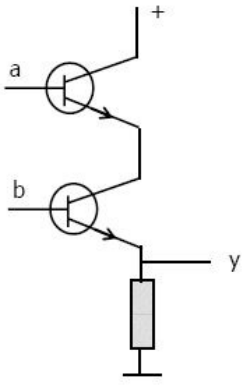


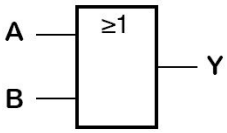



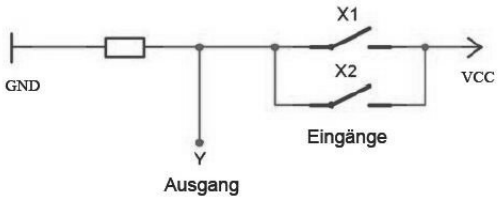



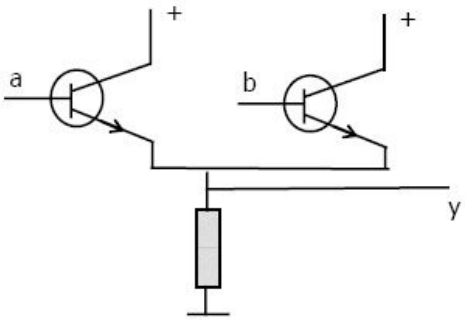




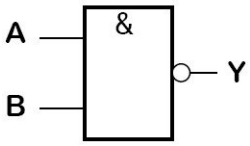
### 3. 2 Technische Realisierung eines ODER-Gliedes

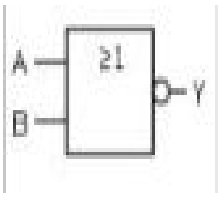


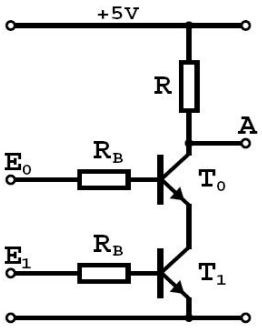



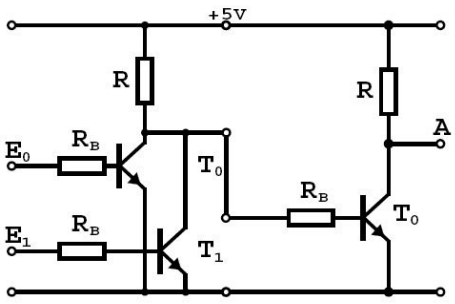




### 3. 3 Technische Realisierung eines NAND- und NOR-Gliedes



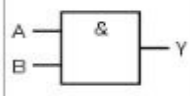
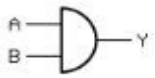
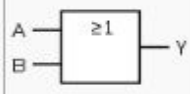
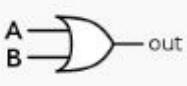
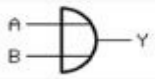
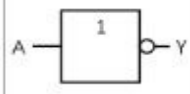
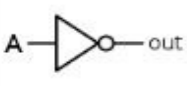
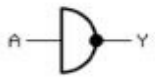
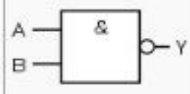
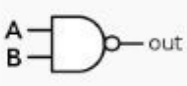
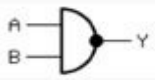
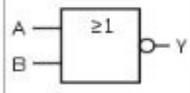
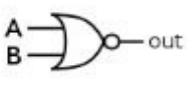
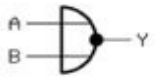
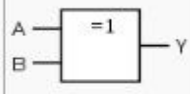
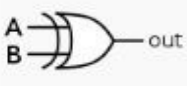
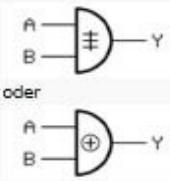
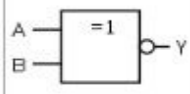
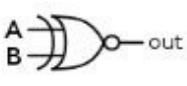
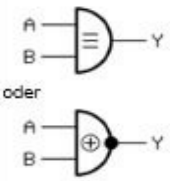



## 3.4 Übersicht über die logischen Grundschaltungen

Im Unterricht sollten ausschließlich die Symbole nach IEC 60617-12 verwendet werden.

| Name | Funktion | Symbol in Schaltplan | | | Wahrheits-tabelle |
|---|---|---|---|---|---|
| | | IEC 60617-12 | US ANSI 91-1984 | DIN 40700 (vor 1976) | |
| UND-Gatter (AND) | $Y = A \wedge B$<br>$Y = A \cdot B$<br>$Y = AB$ |  |  |  | A B Y<br>0 0 0<br>0 1 0<br>1 0 0<br>1 1 1 |
| ODER-Gatter (OR) | $Y = A \vee B$<br>$Y = A + B$ |  |  |  | A B Y<br>0 0 0<br>0 1 1<br>1 0 1<br>1 1 1 |
| NICHT-Gatter (NOT) | $Y = \overline{A}$<br>$Y = \neg A$ |  |  |  | A Y<br>0 1<br>1 0 |
| NAND-Gatter (NICHT UND) (NOT AND) | $Y = \overline{A \wedge B}$<br>$Y = A \barwedge B$<br>$Y = \overline{AB}$ |  |  |  | A B Y<br>0 0 1<br>0 1 1<br>1 0 1<br>1 1 0 |
| NOR-Gatter (NICHT ODER) (NOT OR) | $Y = \overline{A \vee B}$<br>$Y = A \barvee B$<br>$Y = \overline{A + B}$ |  |  |  | A B Y<br>0 0 1<br>0 1 0<br>1 0 0<br>1 1 0 |
| XOR-Gatter (Exklusiv-ODER, Antivalenz) (EXCLUSIVE OR) | $Y = A \veebar B$<br>$Y = A \oplus B$ |  |  |  | A B Y<br>0 0 0<br>0 1 1<br>1 0 1<br>1 1 0 |
| XNOR-Gatter (Nicht-Exklusiv-ODER, Äquivalenz) (EXCLUSIVE NOT OR) | $Y = \overline{A \veebar B}$<br>$Y = A \barveebar B$<br>$Y = \overline{A \oplus B}$ |  |  |  | A B Y<br>0 0 1<br>0 1 0<br>1 0 0<br>1 1 1 |

Quelle: http://de.wikipedia.org/wiki/Logikgatter

## 4. Gesetze der Boolschen Algebra

**Kommutative Gesetze**

1a) $XY = YX$ 1b) $X+Y = Y+X$

**Assoziative Gesetze**

2a) $X(YZ) = (XY)Z$ 2b) $X+(Y+Z) = (X+Y)+Z$

**Distributive Gesetze**

3a) $X(Y+Z) = XY+XZ$ 3b) $X+YZ = (X+Y)(X+Z)$

**Idempotenz Gesetze**

4a) $XX = X$ 4b) $X+X = X$

**Absorptionsgesetze**

5a) $X(X+Y) = X$ 5b) $X + XY = X$

**Komplementgesetze**

6a) $X\overline{X} = 0$ 6b) $X + \overline{X} = 1$

**Gesetz des doppelten Komplements**

7) $\overline{\overline{X}} = X$

**De Morgansche Gesetze**

8a) $\overline{XY} = \overline{X} + \overline{Y}$ 8b) $\overline{X + Y} = \overline{X}\,\overline{Y}$

**Operationen mit 0 und 1**

9a) $0X = 0$ 9b) $1+X = 1$

10a) $1X = X$ 10b) $0+X = X$

11a) $\overline{0} = 1$ 11b) $\overline{1} = 0$

Die Gesetze können durch Schaltungen, Wertetabellen und Venn-Diagramme bewiesen werden.

## 5. Zahlensysteme

**Binäres System (Duales System)**
Ziffern: 0, 1
Stellenwert: 2^n, ..., 2^2, 2^1, 2^0

**Dezimales System**
Ziffern: 0..9
Stellenwert: 10^n, ..., 10^2, 10^1, 10^0

**Hexadezimales System**
Ziffern: 0..9, A..F (= 10..15)
Stellenwert: 16^n, ..., 16^2, 16^1, 16^0

**Umwandlung vom binären bzw. hexadezimalen ins dezimale System**
Von der letzten Stellen angefangen, werden die Ziffern mit 2^0, 2^1, 2^2, usw. bzw. mit 16^0, 16^1, 16^2, usw. multipliziert und danach aufaddiert:

```
10110          → 22          1A5D9                  → 107993
0 * 2^0 (1)    →  0          9       * 16^0 (1)     →      9
1 * 2^1 (2)    →  2          D (13)  * 16^1 (16)    →    208
1 * 2^2 (4)    →  4          5       * 16^2 (256)   →   1280
0 * 2^3 (8)    →  0          A (10)  * 16^3 (4096)  →  40960
1 * 2^4 (16)   → 16          1       * 16^4 (65536) →  65536
                 22                                    107993
```

**Umwandlung vom dezimalen ins binäre bzw. hexadezimale System**
Über den MOD- und den DIV-Befehl werden aus der dezimalen Zahl die einzelnen Stellen der binären bzw. der hexadezimalen Zahl ermittelt. Diese Stellen müssen dann rückwärts (rekursiv) aneinander gehängt werden.

```
202         → 10110          107993             →  1A5D9
22  MOD 2   → 0              107993     MOD 16  →      9
(22 DIV 2   → 11)            (107993    DIV 16  →  6749)
11  MOD 2   → 1              6749       MOD 16  →     D(13)
(11 DIV 2   → 5)             (6749      DIV 16  →  421)
5   MOD 2   → 1              421        MOD 16  →    5
(5  DIV 2   → 2)             (421       DIV 16  →   26)
2   MOD 2   → 0              26         MOD 16  →   A(10)
(2  DIV 2   → 1)             (26        DIV 16  →  1)
1   MOD 2   → 1              1          MOD 16  →  1
              10110                               1A5D9
```

**Umwandlung vom hexadezimalem ins binäre System und umgekehrt**
Wenn eine hexadezimale Zahl ins binäre System umgewandelt wird, dann entstehen aus einer Ziffer mit Hilfe des MOD- und des DIV-Befehls vier binäre Ziffern. (Dabei ist darauf zu achten, dass aus einer hexadezimalen Ziffer immer vier binäre Ziffern werden!) Im umgekehrten Fall entsteht aus vier binären Ziffern über Multiplikation und Addition eine hexadezimale Ziffer.

```
A4             → 1010 0100          1101 0101      →  D5
4       MOD 2  → 0                  1 * 2^0 (1)    →   1
2       MOD 2  → 0                  0 * 2^1 (2)    →   0
1       MOD 2  → 1                  1 * 2^2 (4)    →   4
!!0     MOD 2  → 0 !!!              0 * 2^3 (8)    →   0
(4             → 0100)                                 5
10(A)   MOD 2  → 0                  1 * 2^0 (1)    →   1
5       MOD 2  → 1                  0 * 2^1 (2)    →   0
2       MOD 2  → 0                  1 * 2^2 (4)    →   4
1       MOD 2  → 1                  1 * 2^3 (8)    →   8
(A             → 1010)                                D(13)
A4             → 1010 0100          1101 0101      →  D5
```

## 5.1 Subtraktion

Methode : *Komplementäraddition*
Man kann die Subtraktion einfach auf eine Addition zurückführen, indem man das Zweierkomplement einer Zahl addiert und den Übertrag weglässt.
Man bildet das Zweierkomplement einer Binärzahl indem man zum Einerkomplement 1 addiert. Das Einerkomplement wird durch Invertierung gebildet, d.h. jede 1 wird zu einer 0 und jede 0 wird zu einer 1.

*Bsp.1:* 26D => 11010b
-11D => - 1011b

15D => 1111b

1. Schritt 11010b
-01011b ausfüllen der leeren Stellen

2. Schritt 11010b
+10100b Einerkomplement -> invertieren
+ 1b Zweierkomplement -> 1 addieren

3. Schritt 11010b addieren
+10101b

**1**01111b

Der Übertrag entfällt.

*Bsp.1(im Dezimalsystem):*

9D => 9D
- 2D => + 8D Zehnerkomplement von 2

7D => **1**7D

Der Übertrag entfällt.

*Bsp.2:* 3D => 11b
-8D => - 1000b

-5D => 101b

1. Schritt 0011b
-1000b ausfüllen der leeren Stellen

2. Schritt 1000b
+0111b Einerkomplement -> invertieren
+ 1b Zweierkomplement -> 1 addieren

3. Schritt 0011b addieren
+ 1000b

**0**1011b

**Falls kein Übertrag entsteht muss noch einmal das Zweierkomplement gebildet werden.**

1011
0100 Einerkomplement -> invertieren
+ 1 Zweierkomplement - > 1 addieren
=> *101* ***(Endergebnis)***

*Bsp.2 im Dezimalsystem:*

```
   2D    =>    2D
 - 9D    =>  + 1D   Zehnerkomplement
_____          _____
 - 7D    =>   03D
```

Kein Übertrag, d.h. es muss noch einmal das Zehnerkomplement gebildet werden.

```
  03
=> 07 (Das entsprechende negative Ergebnis)
```

*Anmerkung:*
Falls ein Übertrag auftritt, ist das entsprechende Ergebnis positiv.

Falls kein Übertrag auftritt, ist das entsprechende Ergebnis in Komplementdarstellung – d.h. es ist negativ - und muss entsprechend umgeformt werden.

**Verallgemeinerung der Komplementbildung:**

Die Differenz zweier Zahlen $a$ und $b$ ist definiert durch $D=a-b$.
Man kann sie aber auch auf einem anderen Weg berechnen: $D=a+b^*-R$ mit $b^*=R-b$, d.h. $b^*$ ist das Komplement zu $b$.

- $R=B^N$ ist das $B$-Komplement ($B$ : Basis, $N$ : Stellenzahl). Für das Binärsystem ist $B=2$ und damit das Zweierkomplement.
- $R=B^N-1$ ist das $B$-1-Komplement. Im Dualen ist es das Einerkomplement.

*Beispiel:*
Auszurechnen ist $D=a-b$ = 723-256 = 467 mit Hilfe des 1000er Komplements ( $R=B^N = 10^3=1000$).

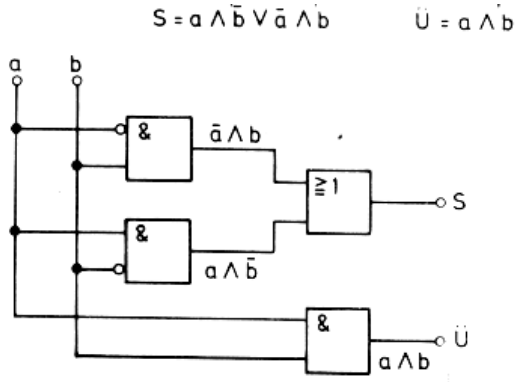
```
  723
+ 744 1000er_Komplment
-------
 1467
```

Der Übertrag auf die Tausender-Stelle wird ignoriert und die restlichen Stellen als Ergebnis gewertet.
Begründung der Korrektheit :

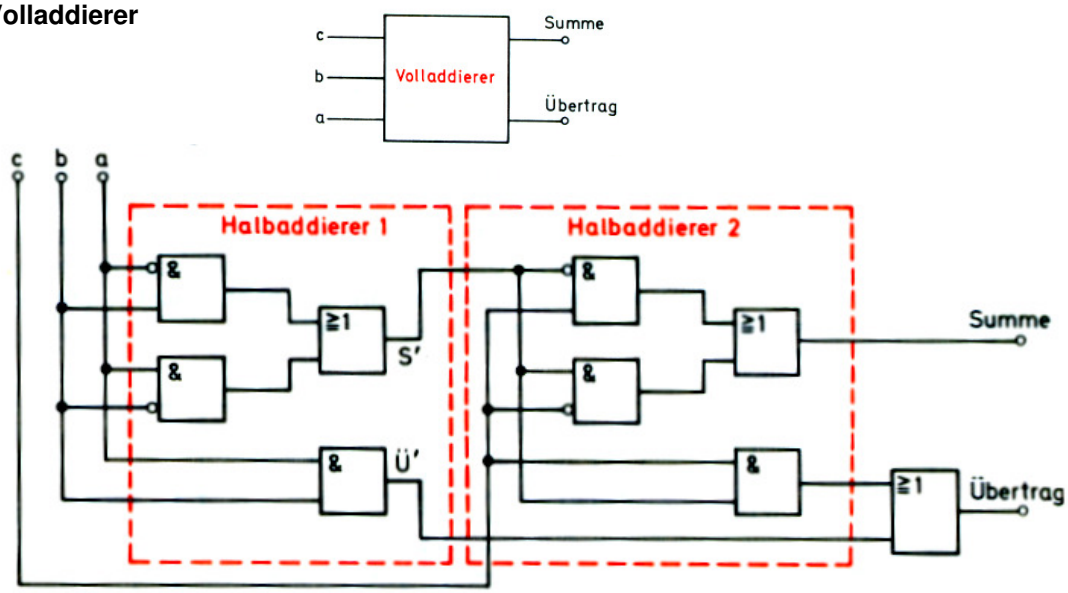
```
      D = a + b* - R
  D + R = a + b*
467 + 1000 = 1467
```

## 6. Einfache Rechenschaltungen

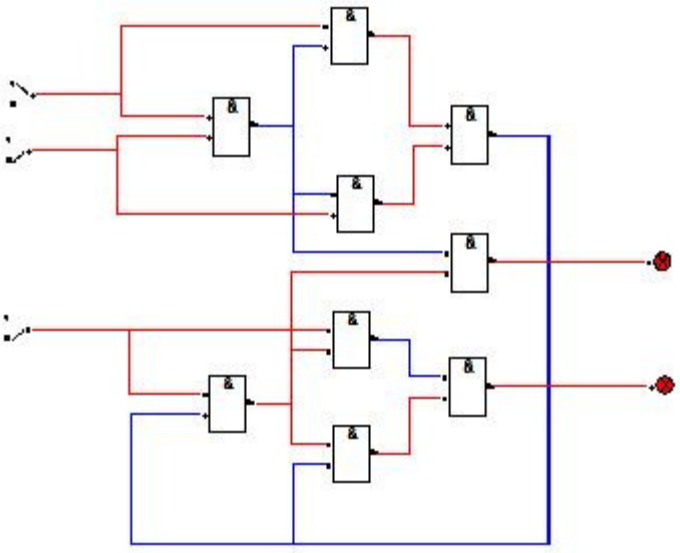
### Halbaddierer



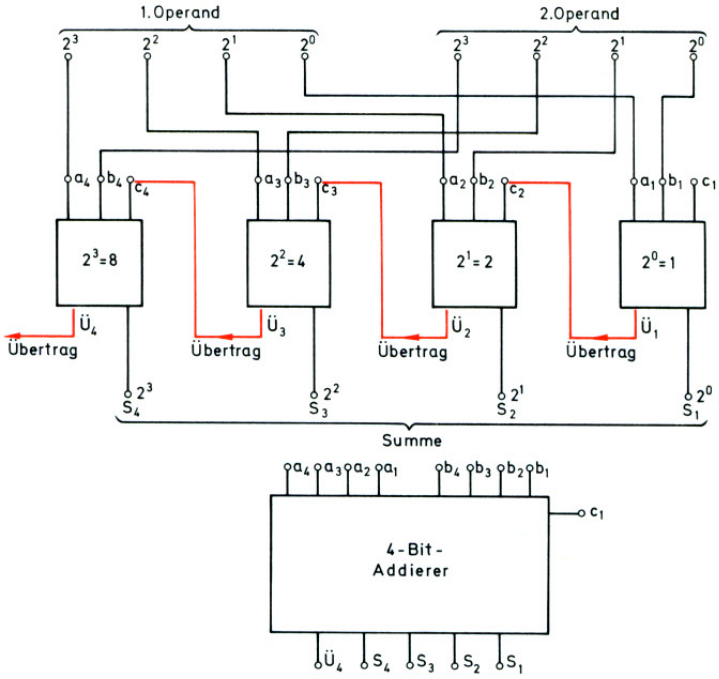

### Volladdierer




### Volladdierer mit nur NAND-Gliedern



### Vierbit-Paralleladdierer




### Vierbit-Additions- Subtraktionsrechenwerk

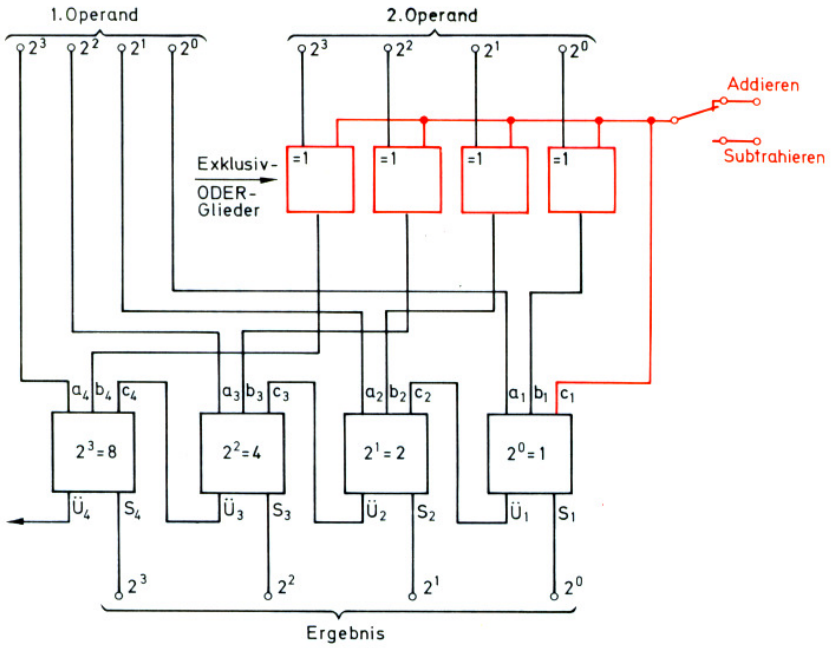
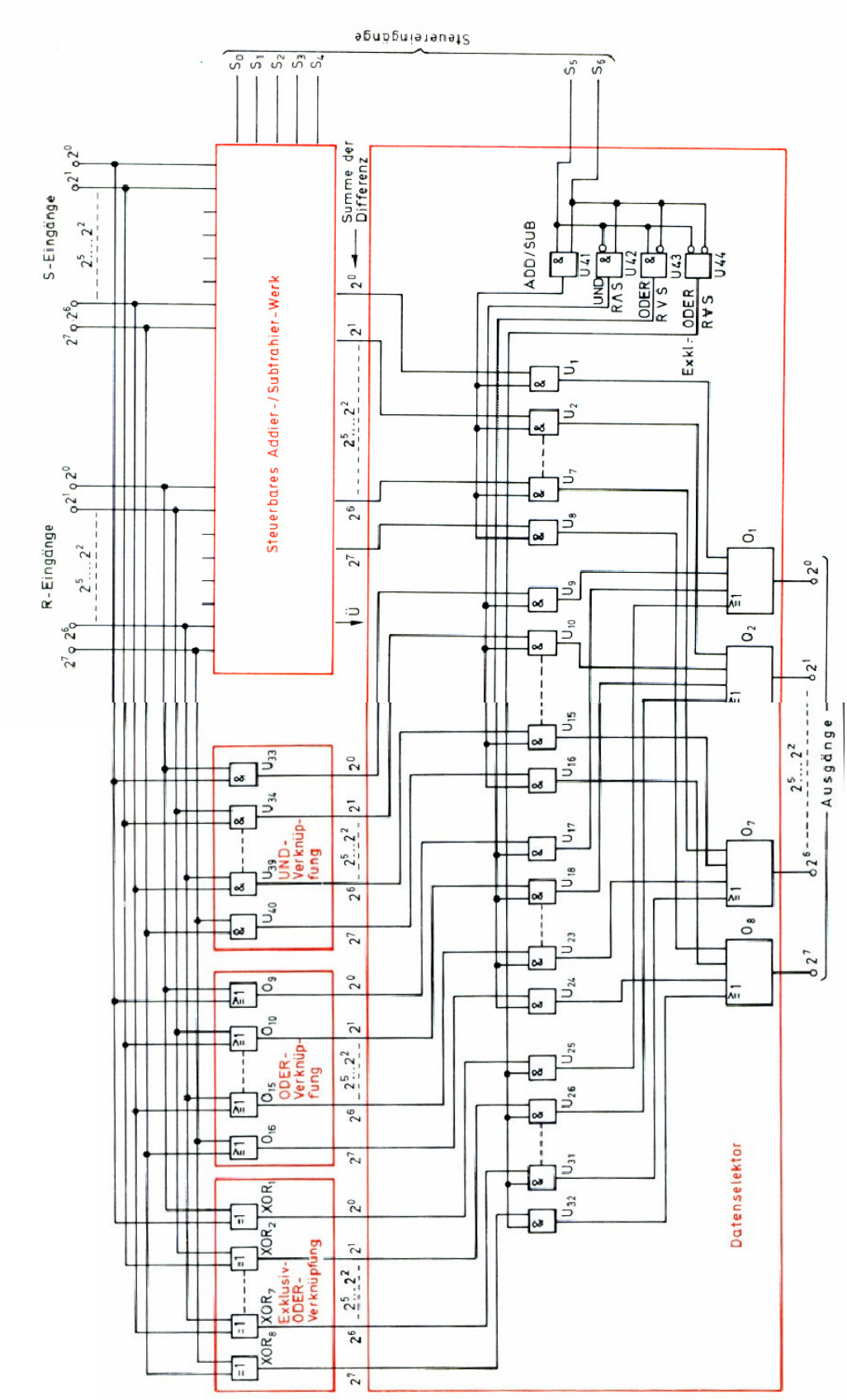



*Bild 7.3*
*4-Bit-Additions-/Subtraktionswerk*

**Beispiel:**

| | Addition | | Subtraktion | |
|---|---|---|---|---|
| | 0101 | | 0101 | |
| Exklusiv-ODER | ⊻ 0000 | Exklusiv-ODER | ⊻ − 1111 | |
| | 0101 | | 1010 | Einer-Komplement |

## Eine einfache ALU (Arithmetic-Logic-Unit)




Quelle: Lothar Starke, Mikroprozessor-Lehre

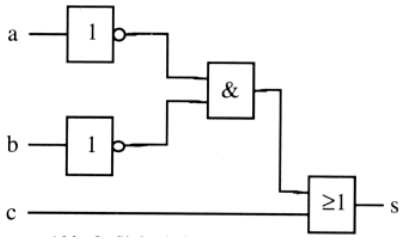
## 7. Übungen u. a. mit dem Digital-Simulator

Der kostenlose Digitalsimulator (http://www.draw2d.org/digitalsimulator/) ist sehr gut für Übungen am PC geeignet. Das Zip-File enthält einen Grundkurs für die Lernenden und Lehrenden.

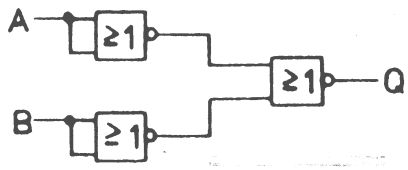
**Aufgabe: 1**
Geben Sie die Arbeitstabelle eines Exklusiv ODER-Gliedes an.
Skizzieren Sie dazu passend die Schaltung aus UND-, ODER und NICHT-Gliedern.

**Aufgabe: 2**
Gegeben ist ein Schaltnetz, das durch die Funktion $Q = A + \overline{A}B$ beschrieben wird.
Wie kann das Schaltnetz vereinfacht werden? Beweisen Sie Ihre Antwort mit Hilfe der booleschen Algebra.

**Aufgabe: 3**
Geben Sie die Arbeitstabelle der abgebildeten Schaltung an.

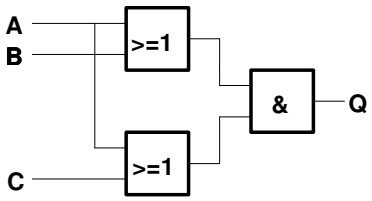



**Aufgabe: 4**
Wie kann die nebenstehende Schaltung vereinfacht werden? Begründung!




**Aufgabe: 5**
a) Geben Sie die Arbeitstabelle der abgebildeten Schaltung an.

b) Geben Sie die dazugehörige schaltalgebraische Gleichung für Q in disjunktiver oder konjunktiver NF an und vereinfachen Sie unter Angabe der Regeln. Skizzieren Sie die vereinfachte Schaltung.




**Aufgabe: 6**
Eine Boolesche Funktion Q = f(A, B, C, D) hat den Funktionswert 1, wenn mehr als eine Variable den Wert 1 annimmt.
a) Geben Sie die vollständige Wertetabelle an.

b) Geben Sie die Funktionsgleichung in einer Normalform an.

c) Skizzieren Sie eine möglichst einfache Schaltung.

**Aufgabe: 7**
Pseudotetraden sind diejenigen Zustände einer Dezimalziffer (BCD), welche nicht genutzt werden. Eine Dezimalziffer benötigt zehn Zustände, wobei diese mit vier Bits (einem halben Byte) codiert wird und somit sechzehn Zustände erlaubt, die sechs übrigen Zustände sind die Pseudotetraden.

a) Entwickeln Sie eine Schaltung, die in einem 4-bit-Wort die Pseudotetraden erkennt.

b) Zeichnen Sie ein Zustandsdiagramm für eines endlichen Automaten zur Erkennung von Pseudotetraden.

**Aufgabe 8:**
Entwickeln Sie arbeitsteilig eine Schaltung mit UND-, ODER und NICHT- Gliedern für einen BCD-7-Segment-Kodierer.
Jede Arbeitsgruppe entwickelt eine Schaltung zur Ansteuerung eines Segmentes der Siebensegmentanzeige.
Fügen Sie Ihre Schaltungen entsprechend zusammen und vereinfachen Sie ;-).

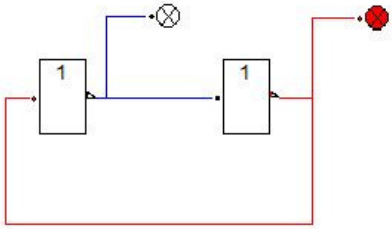
**Aufgabe 9:**

a) Wie kann mit Hilfe der Komplementbildung die Subtraktion 25-13 auf eine Addition zurückgeführt werden?
Führen Sie eine entsprechende Rechnung im Binärcode durch und erläutern Sie die einzelnen Schritte. Wie ist der Fall 13-25 zu behandeln?

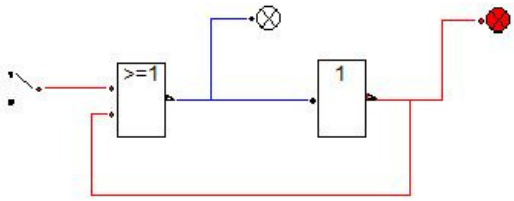
b) Skizzieren Sie die Schaltung eines 2bit-Rechenwerkes (der Einfachheit halber), mit der wahlweise eine Addition oder Subtraktion durchgeführt werden kann.
Die dazu notwendigen Addierer und Exklusiv ODER-Glieder können durch ein entsprechendes Symbol dargestellt werden.

## 8. Speicherelemente (Flip´-Flops)

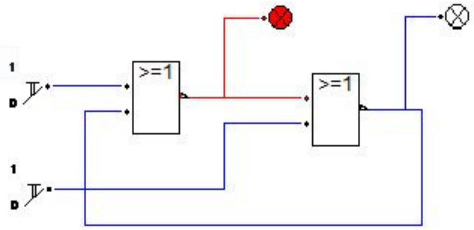
### 8.1 Statischer Speicher durch Rückkopplung



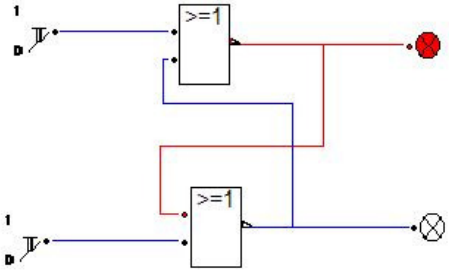

Zwei in Reihe geschaltete NICHT-Glieder. Der Ausgang des letzten Gliedes wird mit dem Eingang des ersten Gliedes verbunden (Rückkopplung).
Die Schaltung hat zwei mögliche Zustände.
Der sich einstellende Zustand ist unbestimmt.




Ersetzt man ein Nicht-Glied durch ein NOR-Glied, so kann man der Schaltung den dargestellten Zustand aufzwingen.




Ersetzt man das andere Nicht-Glied ebenfalls durch ein NOR-Glied, so kann man den entgegengesetzten Zustand erzeugen.




Durch Umstrukturierung erhält man die übliche Darstellung der Ersatzschaltung einer NOR-Latch (RS-Flip-Flop).

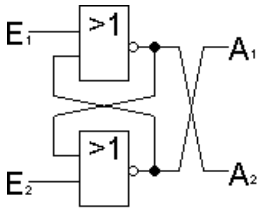
## 8.2 Flip-Flop-Typen

### RS-Flip-Flop

Das RS-Flip-Flop ist ein bistabiles Element und ist der Grundbaustein für alle Flip-Flops in der Digitaltechnik.

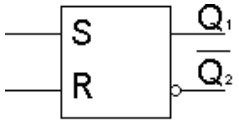
### RS-Flip-Flop aus NOR-Verknüpfungen

Ein Flip-Flop wird aus zwei NOR-Vernüpfungen zusammengeschaltet.
Bei diesem Flip-Flop dürfen die Ausgangspegel $A_1$ und $A_2$ keine gleichen Pegel führen, auch wenn es technisch möglich wäre.




### Schaltzeichen

Im Schaltzeichen werden die Eingänge mit S(setzen) und R(rücksetzen) bezeichnet. $Q_2$ ist zu $Q_1$ negiert.



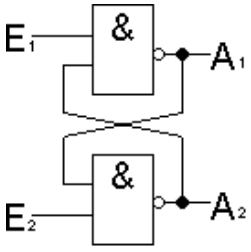

### Arbeitsstabelle

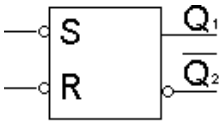
| S | R | $Q_1$ | $Q_2$ | Zustand |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | Setzen |
| 0 | 0 | x | x | Speichern |
| 0 | 1 | 0 | 1 | Rücksetzen |
| 1 | 1 | x | x | Unbestimmt |

| | |
|---|---|
| Setzen: | Bei H-Pegel am S-Eingang wird der Ausgang $Q_1$ auf H-Pegel gesetzt. |
| Speichern: | Führt der S-Eingang L-Pegel so bleibt der Ausgang $Q_1$ unverändert. |
| Rücksetzen: | Wird der R-Eingang mit H-Pegel beschaltet, wird der Ausgang $Q_1$ auf L-Pegel gesetzt. |
| Unbestimmt: | Werden beide Eingänge auf H-Pegel gesetzt, führen die Ausgänge zufällige Pegel. |

### RS-Flip-Flop aus NAND-Verknüpfungen

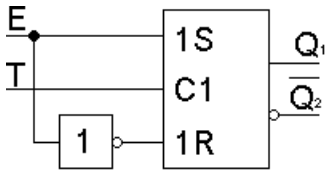
Werden die NOR-Verknüpfungen durch NAND-Verknüpfungen ersetzt, so erhält man ein RS-Flip-Flop mit negierten Eingängen.
Dieses wird durch L-Pegel am S-Eingang gesetzt und am R-Eingang rückgesetzt.
Der Speicherzustand wird durch H-Pegel an beiden Eingängen hergestellt.
Der unbestimmte Zustand wird durch L-Pegel an beiden Eingängen bewirkt.



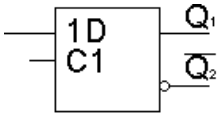



### D-Flip-Flop

Das D-Flip-Flop besteht aus einem RS-FF bei dem der Rücksetzeingang zum Setzeingang negiert ist. Dadurch wird verhindert, daß der unbestimmte Zustand eintritt.




Ein solches Element stellt das Grundelement für statische Schreib-Lese-Speicher dar.
Der einzige Eingang wird als Daten-Eingang bezeichnet. Die Speicherung wird nur mit dem Takteingang gesteuert.
Das D-FF gibt es als taktzustandsgesteuertes und taktflankengesteuertes Flip-Flop.




### Arbeitstabelle

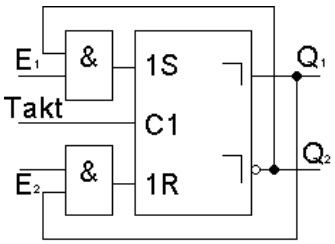
| E | T | $Q_1$ | Funktion |
|---|---|---|---|
| 0 | 0 | n | Speichern |
| 0 | 1 | 0 | Rücksetzen |
| 1 | 0 | n | Speichern |
| 1 | 1 | 1 | Setzen |

Immer wenn am Takteingang eine Null anliegt, wird egal welchen Pegel der Dateneingang hat, der vorhergehende Pegel am Ausgang gespeichert.
Liegt amt Takteingang ein High-Pegel, und ein Low-Pegel am Dateneingang, so wird das Flip-Flop zurückgesetzt.
Liegt am Takteingang ein High-Pegel, und ein High-Pegel am Dateneingang, so wird das Flip-Flop gesetzt.

Wenn ein D-Flip-Flop RS-Eingänge hat, so lässt es sich über diese Eingänge taktunabhängig steuern.

### JK-Flip-Flop

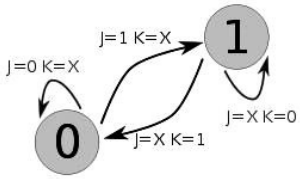
Ein JK-Flip-Flop wechselt bei Anlegen eines Taktimpuls seinen Ausgangszustand, wenn an beiden Eingängen H-Pegel anliegen. Dieses Verhalten wird als Toggeln(kippen) bezeichnet.

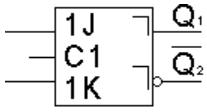



Bei diesem Flip-Flop ist der unbestimmte Zustand ausgeschlossen.
Das JK-FF gibt es als taktflankengesteuertes und taktzustandsgesteuertes Flip-Flop.

### Arbeitsstabelle

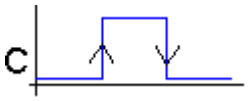
| $E_2$ | $E_1$ | T | $Q_1$ | $Q_2$ | Funktion |
|---|---|---|---|---|---|
| 0 | 1 | 0 | n | n | Speichern |
| 0 | 0 | 0 | n | n | Speichern |
| 1 | 0 | 0 | n | n | Speichern |
| 1 | 1 | 0 | n | n | Speichern |
| 0 | 1 | 1 | 1 | 0 | Setzen |
| 0 | 0 | 1 | n | n | Speichern |
| 1 | 0 | 1 | 0 | 1 | Rücksetzen |
| 1 | 1 | 1 | X | X | Wechseln(Toggln) |



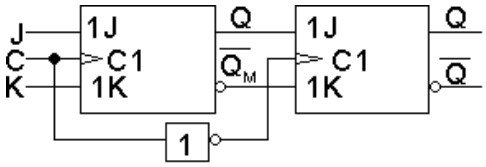

Liegt kein High-Pegel am Takteingang, so wird der an den Ausgängen anstehende Pegel gespeichert.
Liegt am Setzeingang(J) und am Takteingang(C) ein High-Pegel, so wird das Flip-Flop gesetzt.
Liegt am Rücksetzeingang(K) und am Takteingang ein High-Pegel, so wird das Flip-Flop zurückgesetzt.
Liegt an beiden Steuereingängen ein High-Pegel, so wird der gespeicherte Wert gewechselt, d.h. aus High wird Low, aus Low wird High.

## JK-Master-Slave-Flip-Flop



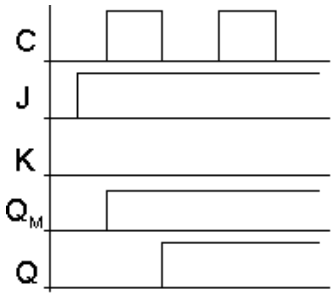

**Funktion:**
**Positive Taktflanke:** Einlesen der am Eingang anstehenden Daten.
**Negative Taktflanke:** Ausgabe(verzögert) der Daten.

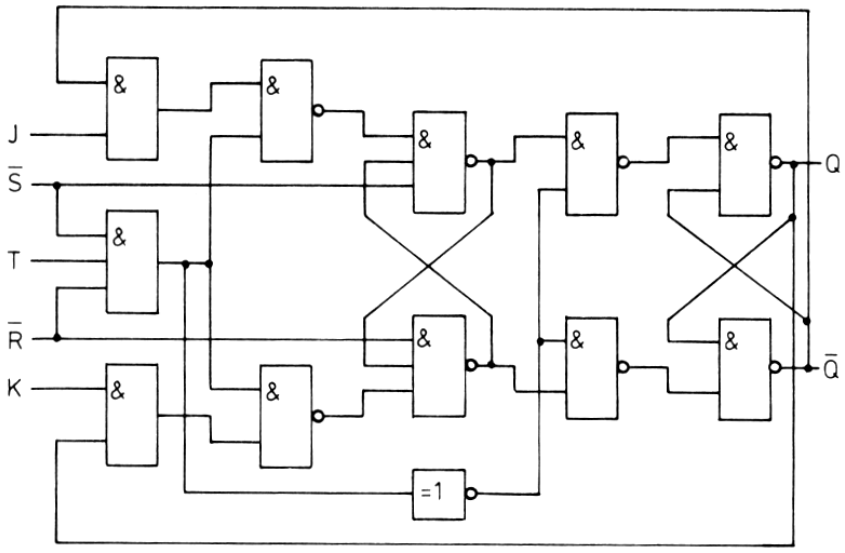
**Schaltungsprinzip:** Beispiel an einem JK-MS-FF



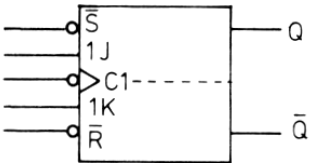

**Impulsdiagramm:**
Mit der positiven Taktflanke wird der Flip-Flop-Zustand eingelesen.

Mit der negativen Taktflanke wird der Zustand an den Ausgang weitergegeben.



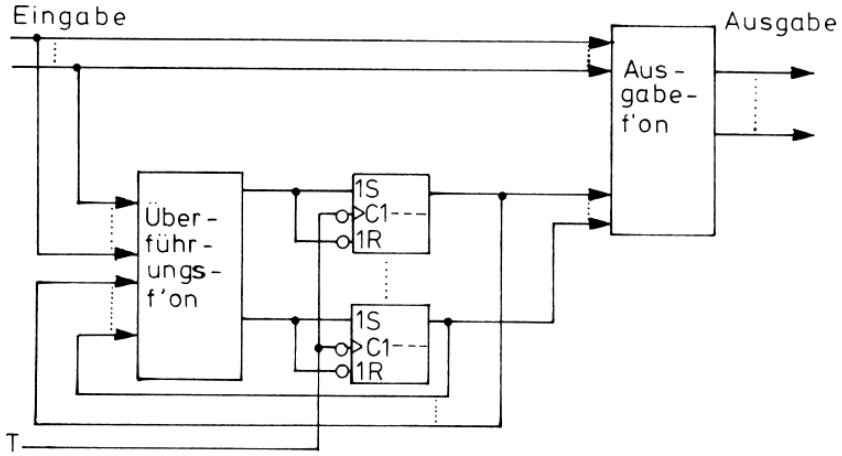




Schaltzeichen:

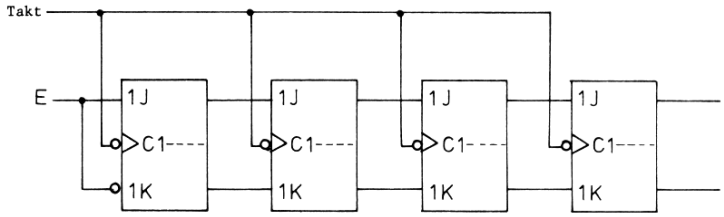


## 8.3 Flip-Flop-Anwendungen

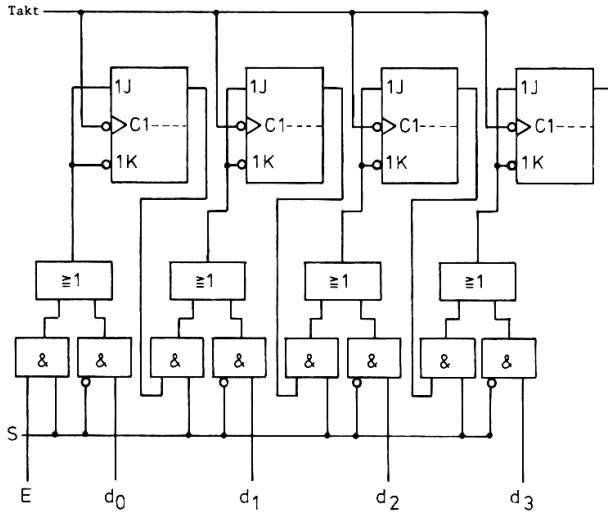
### Digitale Schaltung eines DEA



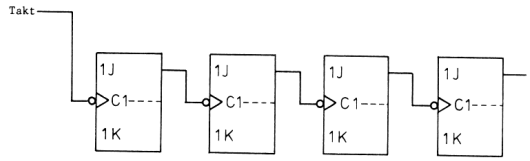

### 4-Bit Schieberegister



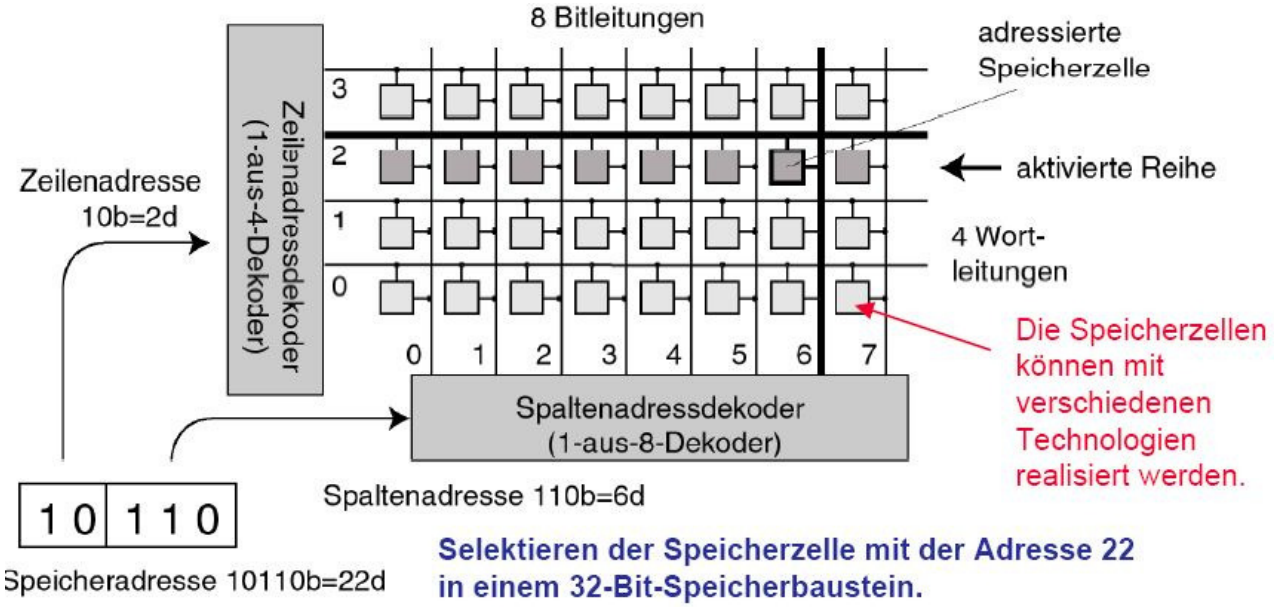

### Parallel ladendes 4-Bit-Schiebregsiter




### Asynchroner 4-Bit-Dualzähler



## 8.4 Speichertypen





SRAM
Statischer Speicher

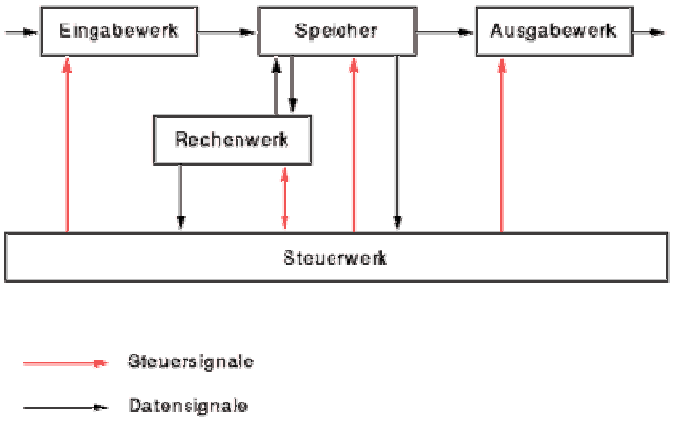
DRAM
Dynamischer Speicher

## 9. Von-Neumann-Architektur (1945 in *First Draft of a Report on the EDVAC.*)

1. Der Rechner besteht aus fünf Funktionseinheiten: dem Steuerwerk, dem Rechenwerk, dem Speicher, dem Eingabewerk und dem Ausgabewerk.
2. Die Struktur des von-Neumann-Rechners ist unabhängig von den zu bearbeitenden Problemen. Zur Lösung eines Problems muss von außen das Programm eingegeben und im Speicher abgelegt werden. Ohne dieses Programm ist die Maschine nicht arbeitsfähig.
3. Programme, Daten, Zwischen- und Endergebnisse werden in demselben Speicher abgelegt.
4. Der Speicher ist in gleichgroße Zellen unterteilt, die fortlaufend durchnumeriert sind. Über die Nummer (Adresse) einer Speicherzelle kann deren Inhalt abgerufen oder verändert werden.
5. Aufeinanderfolgende Befehle eines Programms werden in aufeinanderfolgenden Speicherzellen abgelegt. Das Ansprechen des nächsten Befehls geschieht vom Steuerwerk aus durch Erhöhen der Befehlsadresse um Eins.
6. Durch Sprungbefehle kann von der Bearbeitung der Befehle in der gespeicherten Reihenfolge abgewichen werden.
7. Es gibt zumindest
   - arithmetische Befehle wie Addieren, Multiplizieren usw.;
   - logische Befehle wie Vergleiche, logisches nicht, und, oder usw.;
   - Transportbefehle, z.B. vom Speicher zum Rechenwerk und für die Ein-/ Ausgabe;
   - bedingte Sprünge.
     Weitere Befehle wie Schieben, Unterbrechen, Warten usw. kommen hinzu.
8. Alle Daten (Befehle, Adressen usw.) werden binär codiert. Geeignete Schaltwerke im Steuerwerk und an anderen Stellen sorgen für die richtige Entschlüsselung (Decodierung).

Die Architektur des von-Neumann-Rechners fällt in die Klasse der sogenannten SISD (single instruction stream, single data)-Architekturen. Diese Architekturen sind gekennzeichnet durch

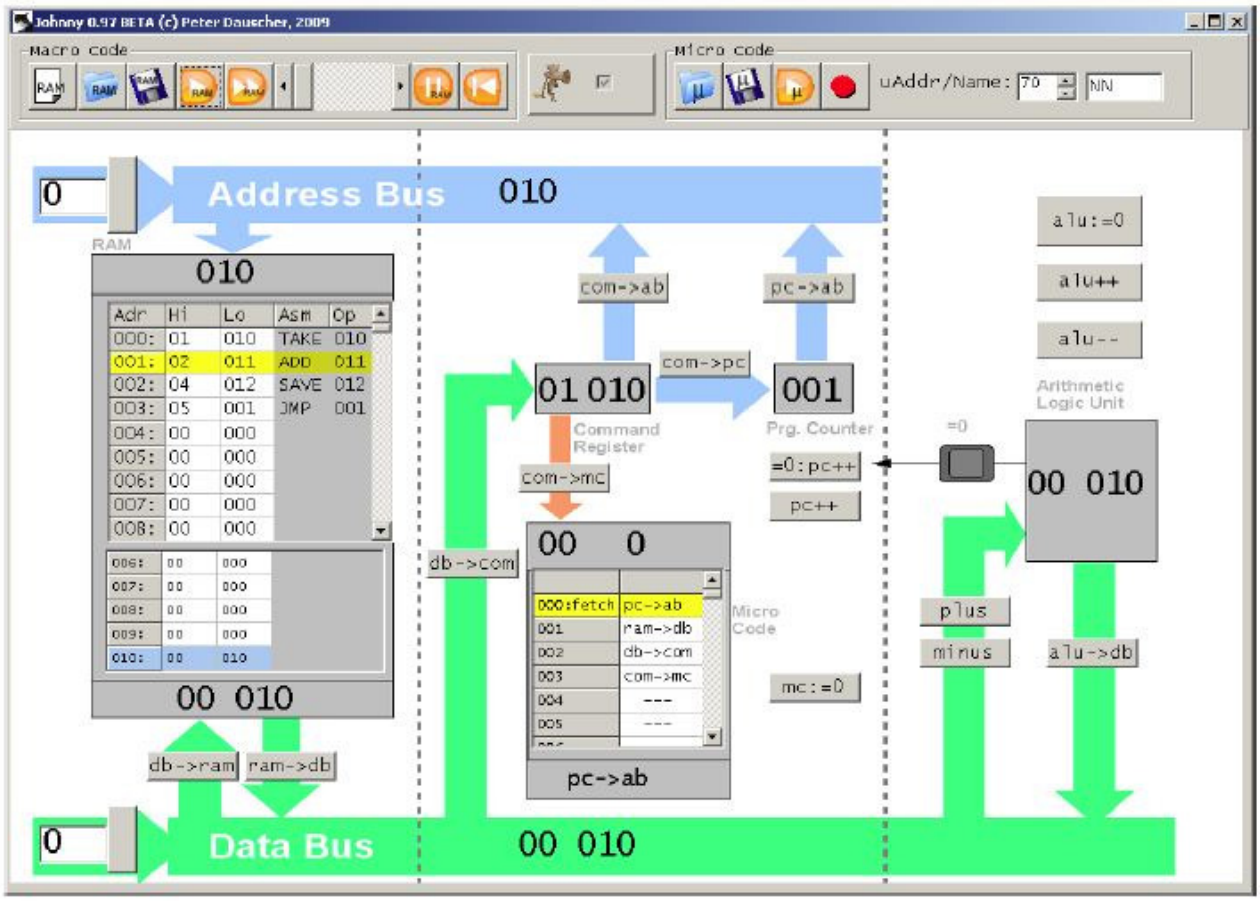
- *einen* Prozessor (*Ein-Prozessor-System*) bestehend aus Steuer- und Rechenwerk und
- die Erzeugung *einer* Befehls- und *einer* Operandenfolge mit streng sequentieller Abarbeitung.



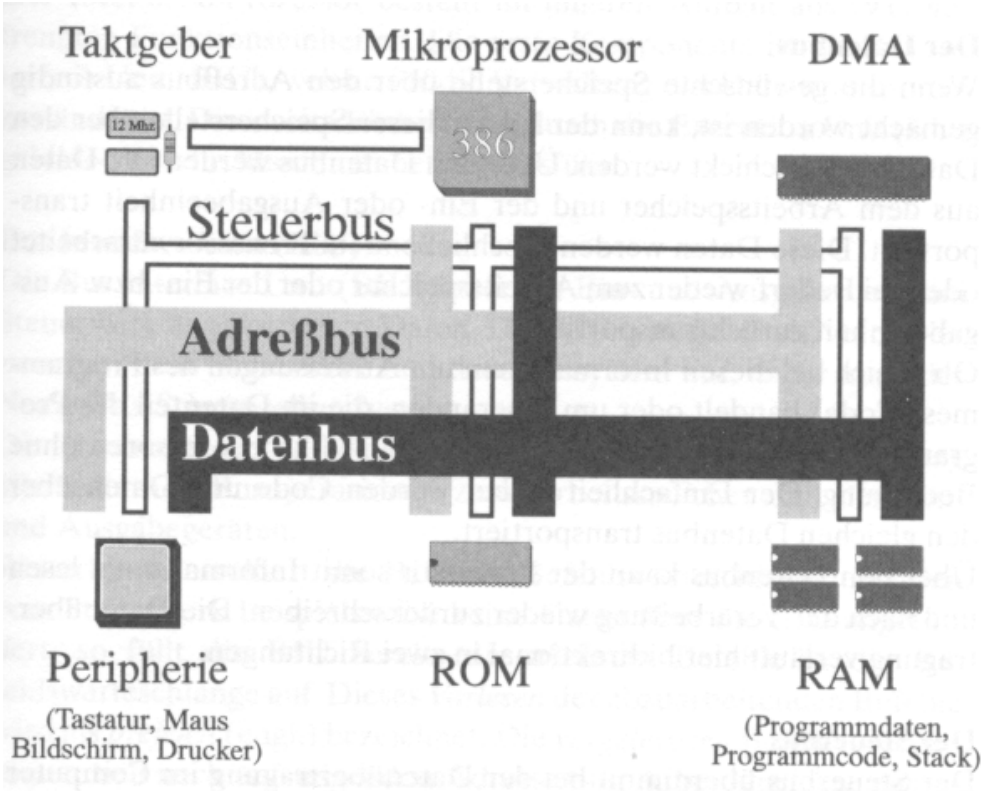

**Von-Neumann-Rechnermit seinen 5 Hauptkomponenten**

Die Architektur ist auch noch der Standard. Nicht-Von-Neumann-Rechner sind Gegenstand der aktuellen Forschung.

Die prinzipielle Arbeitsweise eines Von-Neumann-Rechners sollte dem kostenlose Simulationsprogramm Johnny http://www.heise.de/software/download/johnny/72728 vertieft werden. Eine didaktisch gute Reduktion ist die Eingabe von Dezimalzahlen. Die beiliegende Dokumentation erläutert die ersten Schritte.
Mit einigen Übungen am PC kann man die Schüler gut auf die optionale exemplarische Assembler-Programmierung vorbereiten.



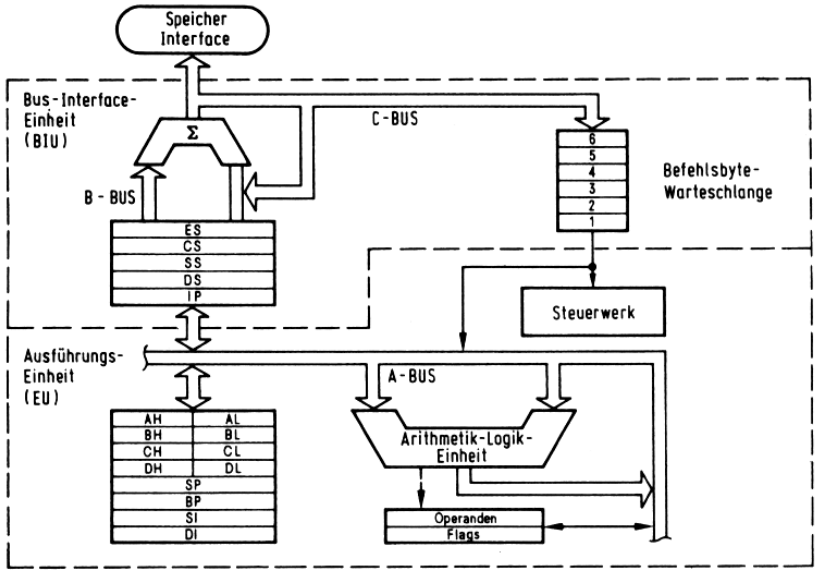
## 10. Blockschaltbild eines Mikrocomputers




Quelle: Backer, Reiner: Programmiersprache ASSEMBLER, Eine strukturierte Einf., rororo, 6. Aufl.

## 11. Architektur des Intel - Mikroprozessors 8086

Der 1978 entwickelte 8086 war die Grundlage für die Entwicklung moderner Prozessoren, die weitgehend zum 8086 abwärtskompatibel sind.

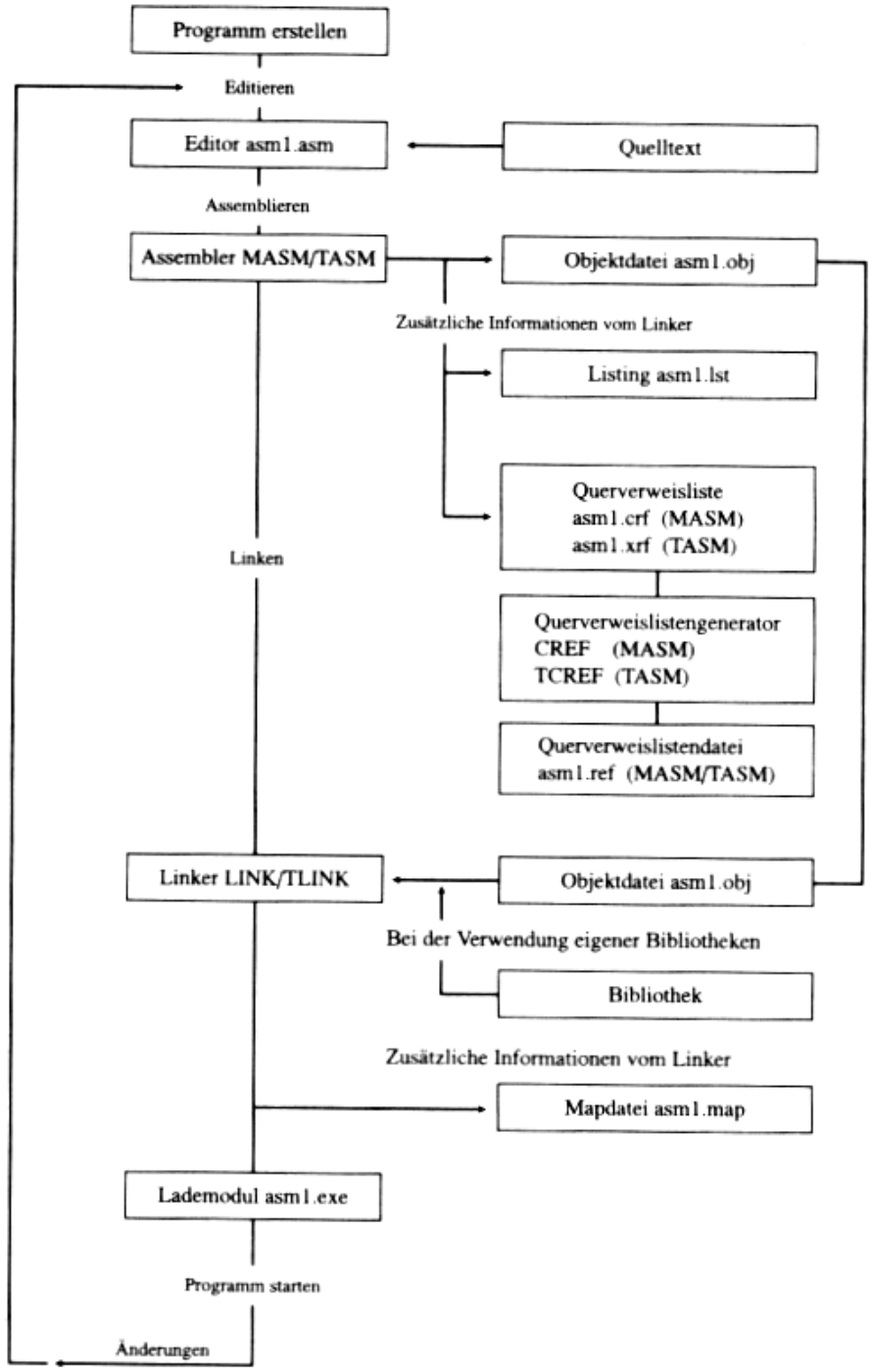



Quelle: Backer, Reiner: Programmiersprache ASSEMBLER, Eine strukturierte Einf., rororo, 6. Aufl.

# 12. Exemplarische Assembler Programmierung

Für weitergehende Studien wird das Tutorial von André Müller (http://andremueller.gmxhome.de/) und das Buch von Rainer Backer empfohlen (vgl. Literatur und Quellenverzeichnis).

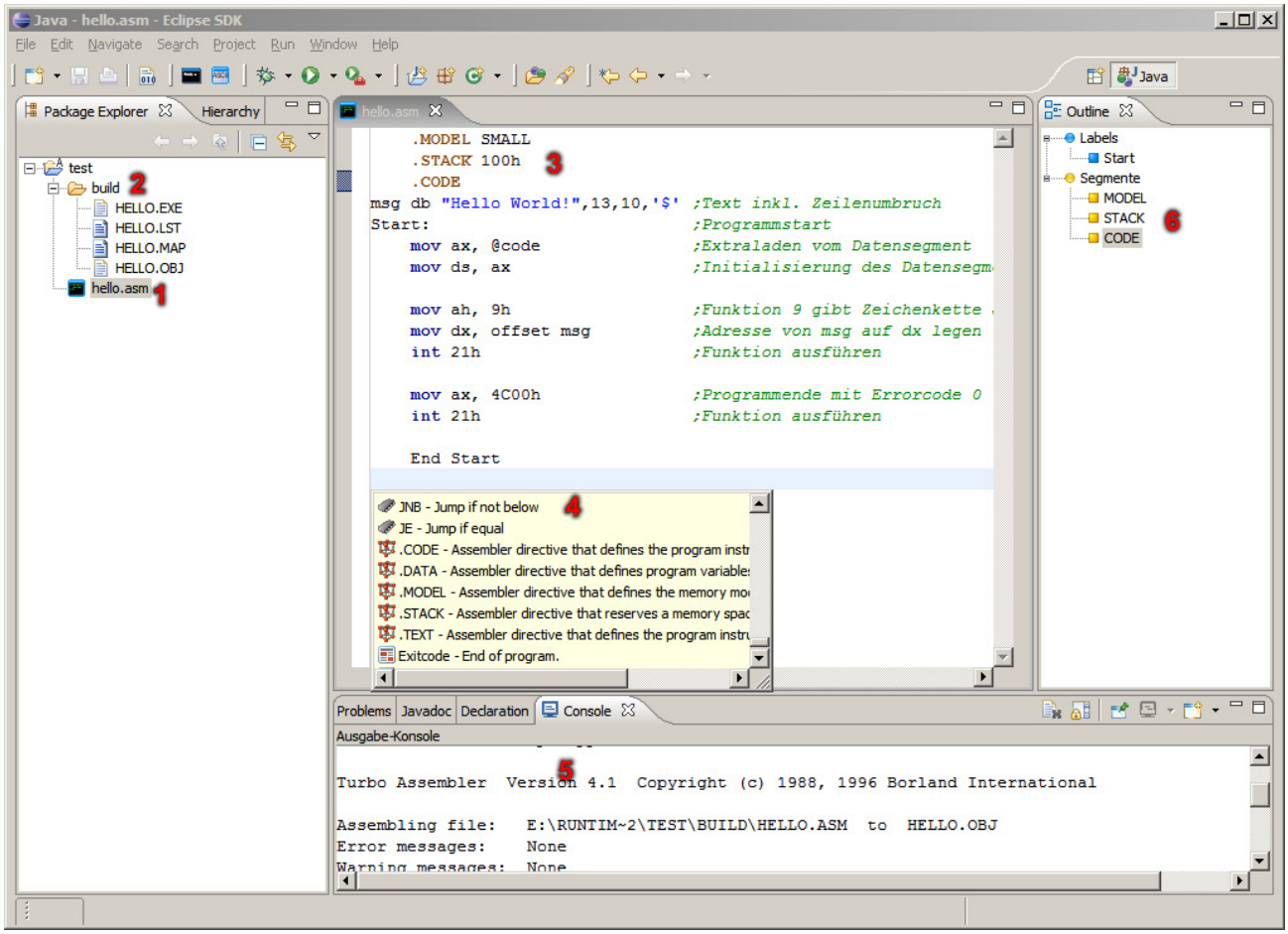
## 12.1 Ablauf der Programmendentwicklung




Quelle: Backer, Reiner: Programmiersprache ASSEMBLER, Eine strukturierte Einf., rororo, 6. Aufl.

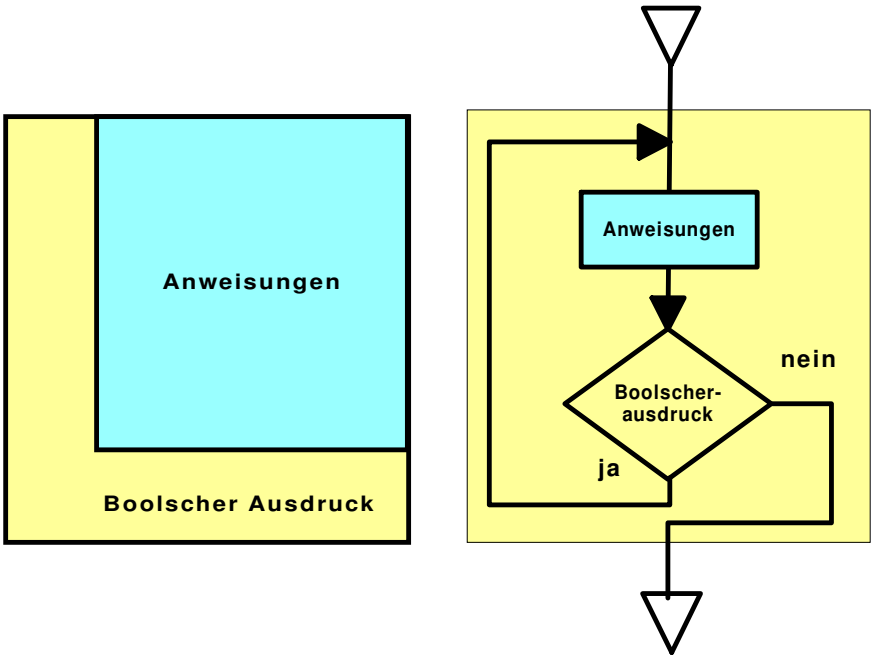
## 12.2 Programmentwicklung mit der universellen IDE-Eclipse

Auch hier zeigt sich, dass die Eclipse-IDE im Informatikunterricht universell einsetzbar ist. Die Lernenden kennen diese aus dem Unterricht im Zusammenhang mit der Webseitenentwicklung und den Programmiersprachen JAVA, Python und Prolog. Für Eclipse wird dazu das kostenlose PlugIn von http://asmplugin.sourceforge.net/ empfohlen. Damit wird der Entwicklungsprozess automatisiert und das Programmieren in der Eclipse typischen Weise unterstützt. Das Plugin enthält eine gute Online für die Konfiguration.



## 12.3 Codeschablonen für Schleifen

**REPEAT – Schleife,** postchecked-Loop, annehmende Schleife



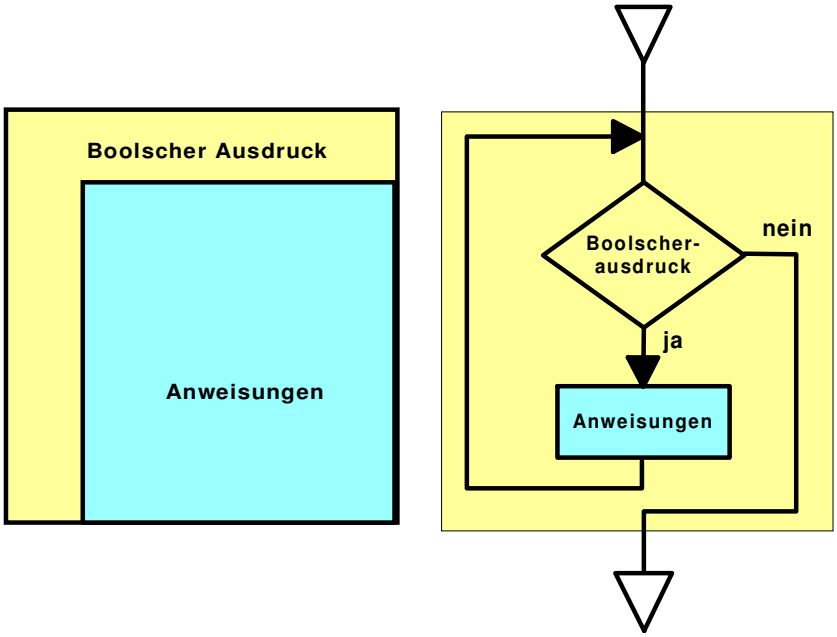

```
_REPEAT:  Anweisung ;erste Anweisung in der Schleife

          Anweisungen

          cmp cx, a        ;Vergleichsanweisung
          jne _REPEAT      ;Bedingter Sprung bei NOT(BA)
          Anweisung        ;erste Anweisung nach der Schleife
```

Ein Beispiel: Der größte, gemeinsame Teiler von A und B

```
PROGRAM GGT;
VAR A      : INTEGER;
    B      : INTEGER;
    Rest   : INTEGER;
    Ergenis: INTEGER;
BEGIN
    A:= 30;
    B:= 105;
    REPEAT
      Rest:= A MOD B;
      A:= B;
      B:= Rest;
    UNTIL Rest = 0;
    Ergebnis:= A;
END.
```

```
.MODEL SMALL      ;ggT von A und B
.DATA
   A      DW  30
   B      DW 105
   Rest   DW   ?
   Erg    DW   ?

.CODE
Anfang:   mov  ax, @data
          mov  ds, ax

          mov  cx, B     ;B in cx-Register
          mov  ax, A     ;A in Akku
_REPEAT:  cwd            ;wandelt ax in ein DWord
dx:ax um
          idiv cx        ;ax:= dxax /cx, Rest in dx
          mov  ax, cx    ;A:= B;
          xchg dx, cx    ;Rest nach cx
          cmp cx, 0
          jne _REPEAT
          mov Erg, ax
ENDE:     mov ah, 4Ch
          int 21h
END       Anfang
```

**WHILE – Schleife,** prechecked-Loop, abweisende Schleife




```
_WHILE:   cmp cx,a          ;Vergleichsanweisung
          jne _ENDWhile     ;Bedingter Sprung bei NOT(BA)
                            ;verneinte Schleifenbedingung

          Anweisungen       ;Anweisungsfolge
          .....             ;innerhalb der Schleife

          jmp _WHILE        ;Unbedingter Sprung zum Anfang
_EndWhile: Anweisung        ;erste Anweisung nach der Schleife
```

**Ein Beispiel: Bestimmung der Länge eines nullterminierten Strings**

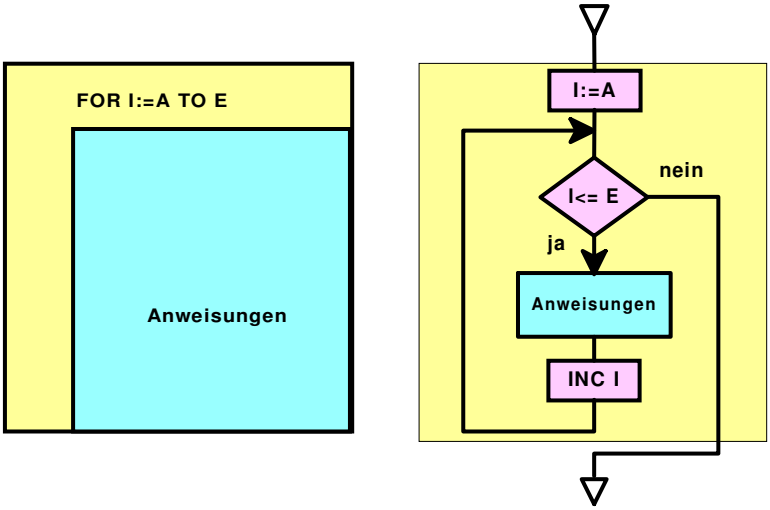
In vielen Programmiersprachen arbeitet man mit nullterminierten Strings. Auch in der Windows-Programmierung wird dieser Datentyp häufig verwendet. Nullterminierte Strings enthalten nicht als erstes Byte ein Längenbyte, sondern beginnen sofort mit der Zeichenfolge. Das Ende wird durch eine Null markiert

```
PROGRAM StringLaenge;
CONST
MyString: ARRAY[0..255] OF CHAR
          = 'Informatik'+ CHR(0);
VAR    i       : INTEGER;
       StrLen  : INTEGER;
BEGIN
  i:=0;
  WHILE ORD (MyString[i]) <> 0 do
        INC (i);
  StrLen:= i;
END.
```

```
.MODEL SMALL
.DATA
   String DB "Informatik",0
   StrLen DW ?

.CODE
Anfang:    mov ax, @data
           mov ds, ax

           mov bx, 0
_While:    cmp String[bx], 0;
           je  _EndWhile
           inc bx
           jmp _While
_EndWhile: mov StrLen, bx

ENDE:      mov ah, 4Ch
           int 21h
END        Anfang
```

BORLAND Pascal stellt mit der UNIT Strings eine Reihe von Funktionen und Prozeduren für die Manipulation nullterminierter Strings zur Verfügung. Schaut man sich den Sourcecode in der Runtime-Libery an, so stellt man fest, dass die Module größtenteils in sehr effizientem Assemblercode realisiert sind. Dabei werden sehr mächtige Assemblerbefehle zur String -und Speichermanipulation verwendet.

**FOR – Schleife,** meistens eine aufwärts- oder abwärtszählende Schleife, prechecked-Loop wird immer dann verwendet, wenn man bereits zum Schleifenbeginn weiß, wie oft die Schleife durchlaufen werden muss.




```
          mov cx, A

_For      cmp cx, E
          ja  _ForEnd

          Anweisungen


          INC
          JMP _For
_ForEnd   Anweisung
```

**Ein Beispiel:** Ganzzahliges arithmetisches Mittel einer Bytefolge mit konstanter Länge.
Anwendung: Messwerterfassung

```
PROGRAM Mittelwert;
CONST
Anzahl= 10;
Feld:   ARRAY[1..Anzahl] OF BYTE
           = (10, 10, 20, 20, 30,
              30, 40, 40, 50, 50);
VAR     i        : INTEGER;
        Summe    : INTEGER;
        Mittelw  : INTEGER;

BEGIN
  Summe:= 0;
  FOR i:=1 TO ANZAHL DO
      Summe:= Summe + Feld[i];
  Mittelw:= Summe DIV Anzahl;
END.
```

```
.MODEL SMALL     Mittelwert

.STACK 100h
.DATA
  Anzahl EQU 10
  Feld   DB 1 DUP (25,25,25,25,25,25,25,25,25,25)
  Mittel DW ?

.CODE
Anfang:  mov ax, @DATA
         mov ds, ax

         mov ax, 0              ;Akku auf 0
         mov bx, 0              ;bx als Feldindex und als
                                ;Schleifenzähler
                                ;Anfangswert ist null
_For:    cmp bx, anzahl-1       ;Endwert ist Anzahl-1
         ja _ForEnd
         add al, Feld[bx]       ;indizierte Adressierung
         adc ah, 0              ;8-bit Addition mit Übertrag
         inc bx
         jmp _For

_ForEnd: mov cx, Anzahl
         cwd
         div cx
         mov  Mittel, ax

Ende:    mov  ax,4C00H
         int  21H
END      Anfang
```

## 12.4 Einfache Beispiele mit Interrupts

```
;*************************************************************
;* Aufgabe 1: Einlesen einer Taste und Anzeige auf Bildschirm *
;*     mit definierter R ckkehr nach MS-DOS      *
;* Prog.Name: PROG1.ASM              *
;*************************************************************

.MODEL SMALL
.CODE
    MOV AH,08H ; liest Tastatur - bringt
    INT 21H    ; Zeichencode nach AL
    MOV DL,AL
    MOV AH,02H ; bringt Zeichen aus Register
    INT 21H    ; DL zur Anzeige
    MOV AH,4CH ; definierte R ckkehr
    INT 21H    ; nach MSDOS
    END
```

```
;*************************************************************
;* Aufgabe 2: Fortl.Zeicheneinlesen von Tastatur und Anzeige*
;*            auf Bildschirm (Verwendung von Fkt.01H )          *
;* Prog.Name: PROG2.ASM                *
;*************************************************************

.MODEL SMALL

.CODE
M1: MOV AH,01H     ; liest Tastatur - bringt
    INT 21H        ; Zeichencode nach AL
    JMP M1         ; und zur Anzeige
    END
```

```
;*************************************************************
;* Aufgabe 3: Einlesen der Tastatur und Anzeige der          *
;*             numerischen Eingaben auf Bildschirm           *
;* Prog.Name: PROG4.ASM                    *
;*************************************************************

.MODEL SMALL
.CODE
M1:     MOV AH,08H ; liest Tastatur - bringt
        INT 21H    ; Zeichencode nach AL
        CMP AL,30H
        JB M1        ; Abfrage, ob kleiner 30H
        CMP AL,39H
        JA M1        ; Abfrage, ob gr”áer 39H
        MOV DL,AL
        MOV AH,02H ; bringt Zeichen aus Register
        INT 21H    ; DL zur Anzeige
        JMP M1
        END
```

Viele weitere Beispiele findet man im Internet.

### Literatur und Quellen


| | |
|---|---|
| Backer, Reiner: | Programmiersprache ASSEMBLER, Eine strukturierte Einf., rororo, 6. Aufl. |
| Starke, Lothar: | Mikroprozessorlehre, Frankfurter Fachverlag, 1979 |
| Leonhard, Erich: | Grundlagen der Digitaltechnik, Carl Hanser, 1976 |
| Gasper et al.: | Technische und theoretische Informatik, bayrischer Schulbuchverl. 1992 |
| Whitesitt, J. E.: | Einf. i. d. Boolesche Algebra, Vieweg, 1972 |
| Ganzhorn, Karl: | Die geschichtliche Entwicklung der Datenverarbeitung, IBM, 1975 |
| HKM: | Lehrplan Informatik, Gymnasialer Bildungsgang, |


**Internet:**


| | |
|---|---|
| Digitalsimulator: | http://www.draw2d.org/digitalsimulator/ |
| Symbole: | http://de.wikipedia.org/wiki/Logikgatter |
| Von-Neumann-Architektur: | http://de.wikipedia.org/wiki/Von-Neumann-Architektur |
| Jonny: | http://www.heise.de/software/download/johnny/72728 |
| Eclipse-plugin: | http://asmplugin.sourceforge.net/ |
| | http://de.wikipedia.org/wiki/Logikgatter |
| Assembler Tutorial: | http://andremueller.gmxhome.de/ |

Zuletzt besucht am 20.08.2011