# 3.3 Analyse und Entwurf von Schaltwerken

## Grundlegende Realisierung von Automaten

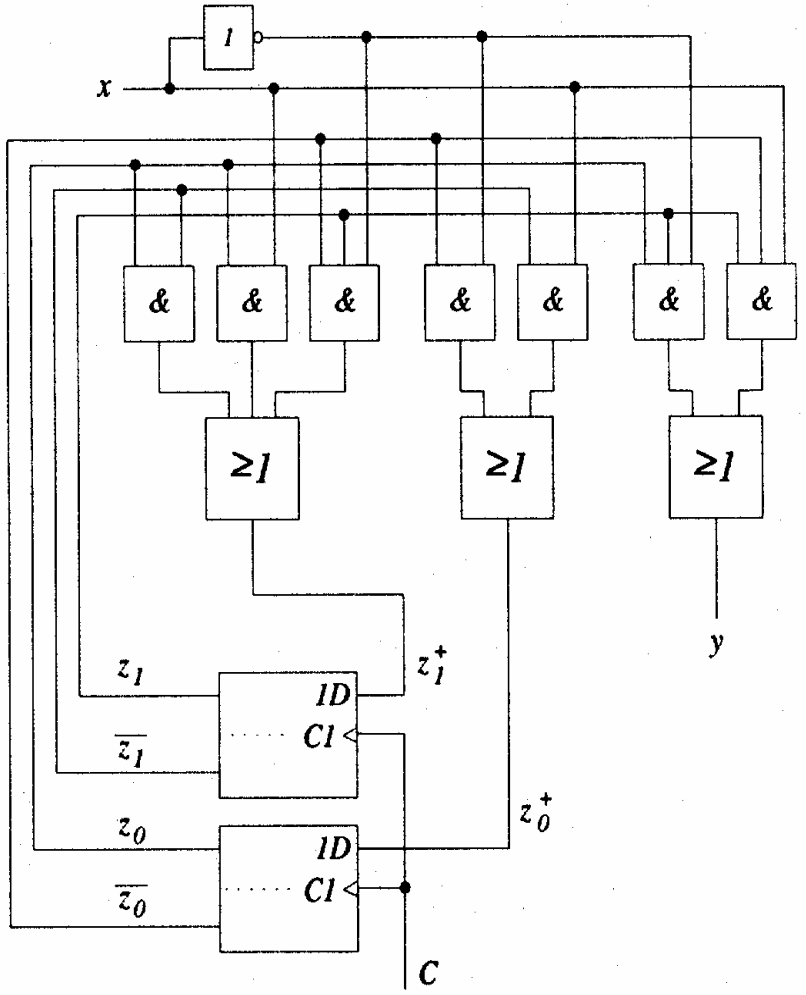
- **Asynchrone Realisierung**
  - ⇨ **Zustandsspeicher durch Rückkopplung**
  - ⇨ **es gibt keinen zentralen Takt**
  - ⇨ **die Zustandsspeicher (Flipflops) können zu jedem Zeitpunkt ihren Wert ändern**
    - **Laufzeiten beachten!**
- **Synchrone Realisierung**
  - ⇨ **Rückkopplung nur durch flanken- oder pegelgetriggerte Flipflops**
  - ⇨ **die Taktleitungen aller Flipflops sind miteinander verbunden (oder hängen nach einem festen Zeitschema voneinander ab)**
- **Obwohl asynchrone Realisierungen auch eine gewisse praktische Bedeutung besitzen, werden hier nur synchrone Realisierungen betrachtet**

# 3.3.1 Analyse von Schaltwerken

- **Ein Schaltwerk zu analysieren bedeutet, sein Schaltverhalten durch**
  - ⇨ **eine Zustandstabelle**
  - ⇨ **dessen Schaltfunktion oder**
  - ⇨ **einen Zustandsgraph zu beschreiben**
- **Prinzipielles Vorgehen:**
  - ⇨ **von einem gegebenen Schaltplan werden zunächst die Ausgabe und Übergangsfunktion abgeleitet**
  - ⇨ **ein Anfangszustand wird angenommen**
  - ⇨ **mit den Werten der Eingangsvariablen werden die Folgezustände abgeleitet**
  - ⇨ **auf diese Weise entstehen die Ablauftabellen**
  - ⇨ **aus den Ablauftabellen kann der Automatengraph abgeleitet werden**

# Beispiel: Ausgangspunkt - der Schaltplan

- **Grundlegende Charakterisierungen**
  - ⇨ **synchrones Schaltwerk**
  - ⇨ **Eingang *x* und Ausgang *y* bestehen je aus einer Variablen**
  - ⇨ **das Schaltwerk enthält 2 D-Flipflops**
  - ⇨ **es kann maximal 4 Zustände besitzen**
  - ⇨ **Das Schaltwerk ist ein Mealy-Automat**



# Die Schaltfunktion

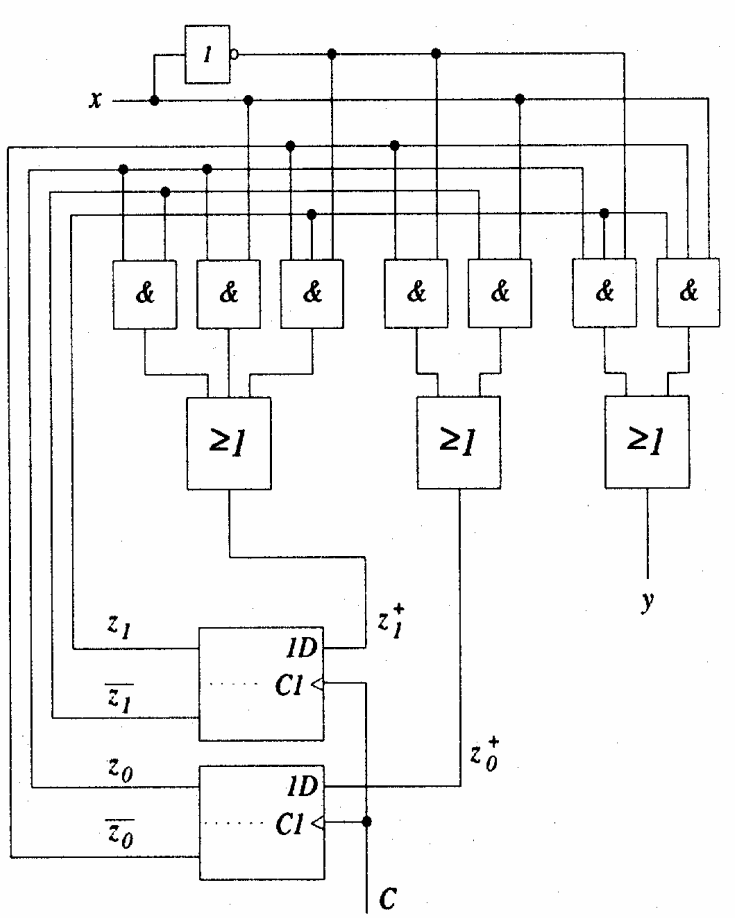
- **Aus dem Schaltplan läßt sich ablesen:**
  - ⇨ **für die Übergangsfunktion**

$$z_0^+ = (\bar{z}_0 \wedge \bar{x}) \vee (\bar{z}_1 \wedge x)$$

$$z_1^+ = (z_0 \wedge \bar{z}_1) \vee (z_0 \wedge x) \vee (\bar{z}_0 \wedge z_1 \wedge \bar{x})$$

  - ⇨ **für die Ausgabefunktion**

$$y = (z_0 \wedge z_1 \wedge \bar{x}) \vee (\bar{z}_0 \wedge z_1 \wedge x)$$

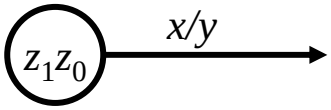


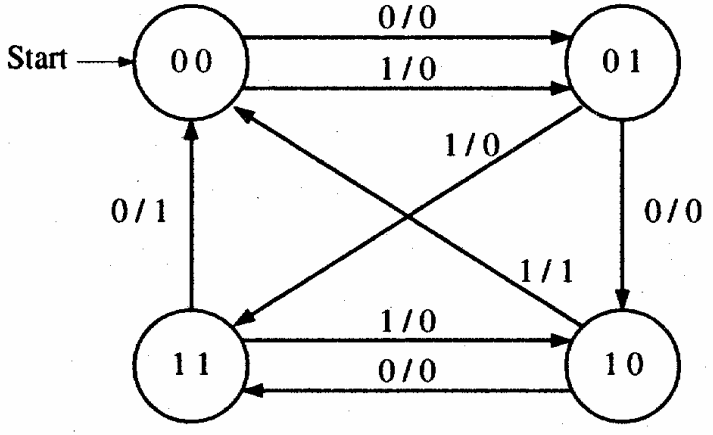
# Die Ablauftabelle und der Automatengraph

- **Aufstellen der Ablauftabelle über die Auswertung der Funktionen für $z_0$, $z_1$ und $y$**
  - ⇨ **alle Belegungen der Eingangsvariablen**
  - ⇨ **alle Belegungen der Zustandsvariablen**

| $z_1$ | $z_0$ | $x$ | $z_1^+$ | $z_0^+$ | $y$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

- **Aufstellen des Automatengraphen über die Auswertung der Ablauftabelle**
  - ⇨ **Beschriftung der Zustände und Übergänge nicht vergessen!**



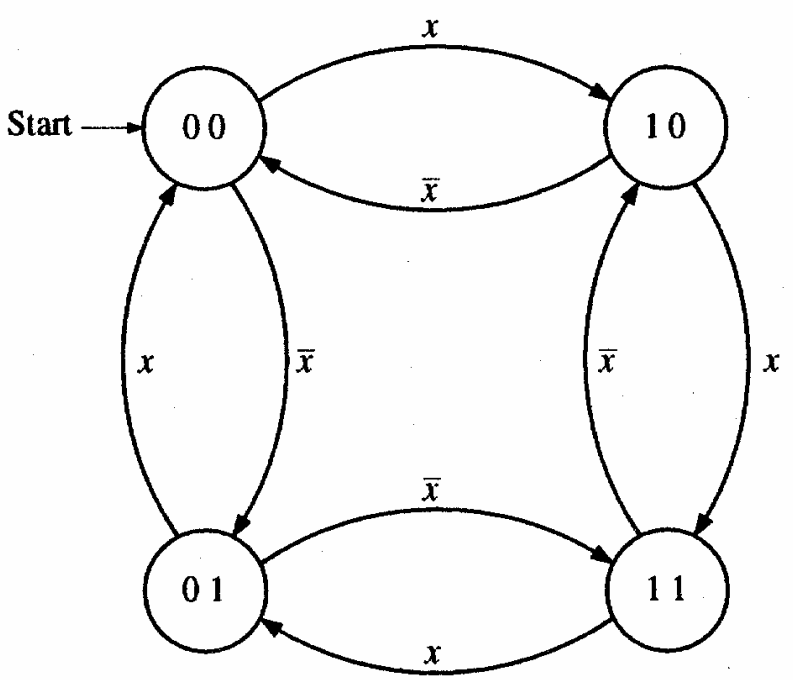



# 3.3.2 Entwurf von Schaltwerken

- **Prinzipielles Vorgehen:**
  - ⇨ **festlegen der Zustandsmenge**
    - **daraus ergibt sich die Anzahl der erforderlichen Speicherglieder**
  - ⇨ **festlegen des Anfangszustands**
  - ⇨ **Definition der Ein- und Ausgangsvariablen**
  - ⇨ **Darstellung der zeitlichen Zustandsfolge in Form eines Zustandsgraphen**
  - ⇨ **aufstellen der Ablauftabelle**
  - ⇨ **Herleitung der Übergangs- und Ausgabefunktionen**
  - ⇨ **Darstellung der Übergangs- und Ausgabefunktionen in einem KV-Diagramm und Minimierung**
  - ⇨ **Darstellung des Schaltwerks in einem Schaltplan**

# Beispiel: ein umschaltbarer Zähler

- **Es soll ein zweistelliger Gray-Code-Zähler entworfen werden, der sowohl vorwärts als auch rückwärts zählen kann**
- **Die Umschaltung der Zählrichtung erfolgt über die Eingangsvariable** $x$
  - ⇨ **für** $x$=0 **ist die Zählfolge**
    `00 - 01 - 11 - 10`
  - ⇨ **für** $x$=1 **ist die Zählfolge**
    `00 - 10 - 11 - 01`
- **Die Ausgangsvariablen sind identisch mit den Zustandsvariablen, da der Zählerstand angezeigt werden soll**
  - ⇨ **Moore-Automat**




**Automatengraph**

# Ablauftabelle und die Übergangsfunktionen

- **Die Ablauftabelle kann direkt aus dem Automatengraph abgeleitet werden**
  - ⇨ **die linke Seite enthält alle Wertekombinationen, die $z_0$, $z_1$ und $x$ einnehmen können**
  - ⇨ **die rechte Seite enthält die Werte der Folgezustände**

| $z_1$ | $z_0$ | $x$ | $z_1^+$ | $z_0^+$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |

- **Aus der Ablauftabelle können die KV-Diagramme für $z_0$ und $z_1$ aufgestellt werden**

$z_1^+$ — $x$ —

| | | | |
|---|---|---|---|
| $0_0$ | $1_1$ | $1_5$ | $0_4$ |
| $1_2$ | $0_3$ | $0_7$ | $1_6$ |

$z_0$ — $z_1$ —

$z_0^+$ — $x$ —

| | | | |
|---|---|---|---|
| $1_0$ | $0_1$ | $1_5$ | $0_4$ |
| $1_2$ | $0_3$ | $1_7$ | $0_6$ |

$z_0$ — $z_1$ —

- **Aus den KV-Diagrammen lassen sich die minimierten Übergangsfunktionen ablesen**

$$z_1^+ = (\bar{z}_0 \wedge x) \vee (z_0 \wedge \bar{x})$$

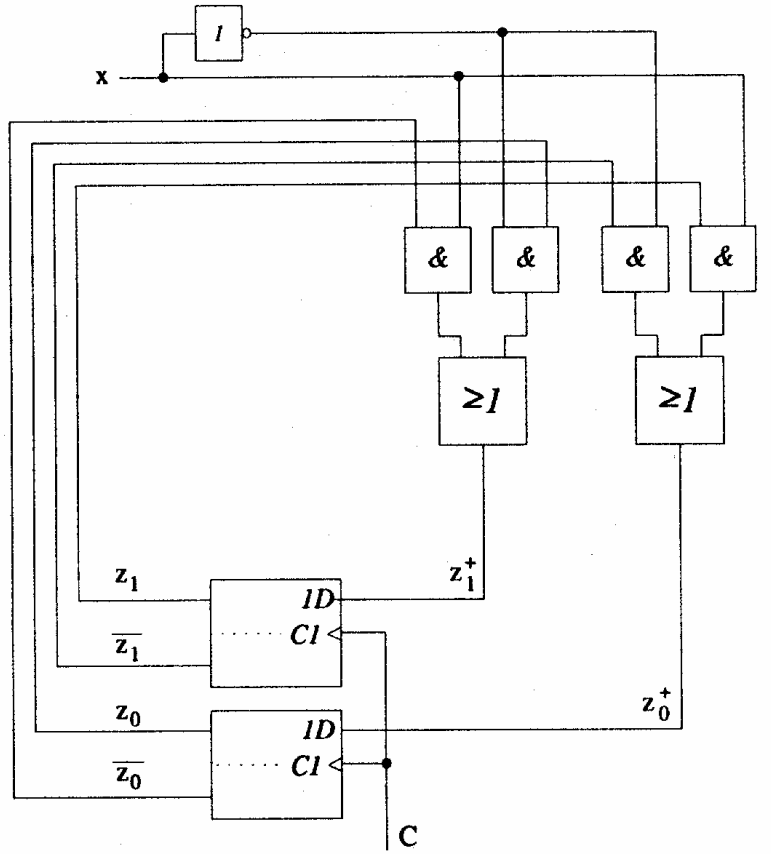
$$z_0^+ = (\bar{z}_1 \wedge \bar{x}) \vee (z_1 \wedge x)$$

# Das Schaltwerk

- **Die minimierten Übergangsfunktionen können schließlich in einem Schaltplan gezeichnet werden**

$$z_0^+ = (\overline{z}_0 \wedge x) \vee (z_0 \wedge \overline{x})$$

$$z_1^+ = (\overline{z}_1 \wedge \overline{x}) \vee (z_1 \wedge x)$$

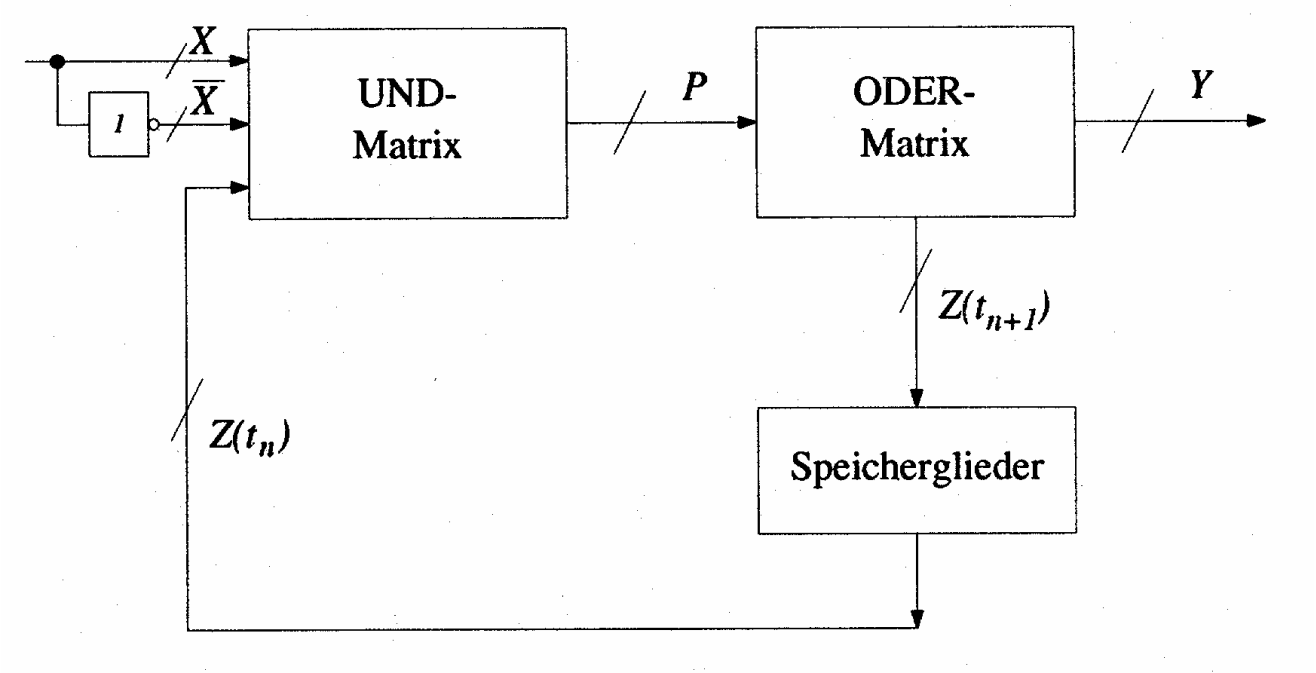


# 3.4 Technische Realisierung von Schaltwerken

- **Realisierung mit diskreten Bauelementen**
  - ⇨ **Verknüpfungsglieder**
  - ⇨ **Speicherglieder**
- **Die Bauelemente werden entsprechend der Aufgabenstellung durch eine feste Verdrahtung miteinander verbunden**
- **Solche Schaltwerksrealisierungen können nur eine feste Aufgabe erfüllen**
  - ⇨ **das Schaltwerk ist nicht flexibel**
  - ⇨ **bei einem Fehler in der Verdrahtung kann keine Korrektur vorgenommen werden**
- **Die Bauelemente stehen als integrierte Schaltkreise zur Verfügung**

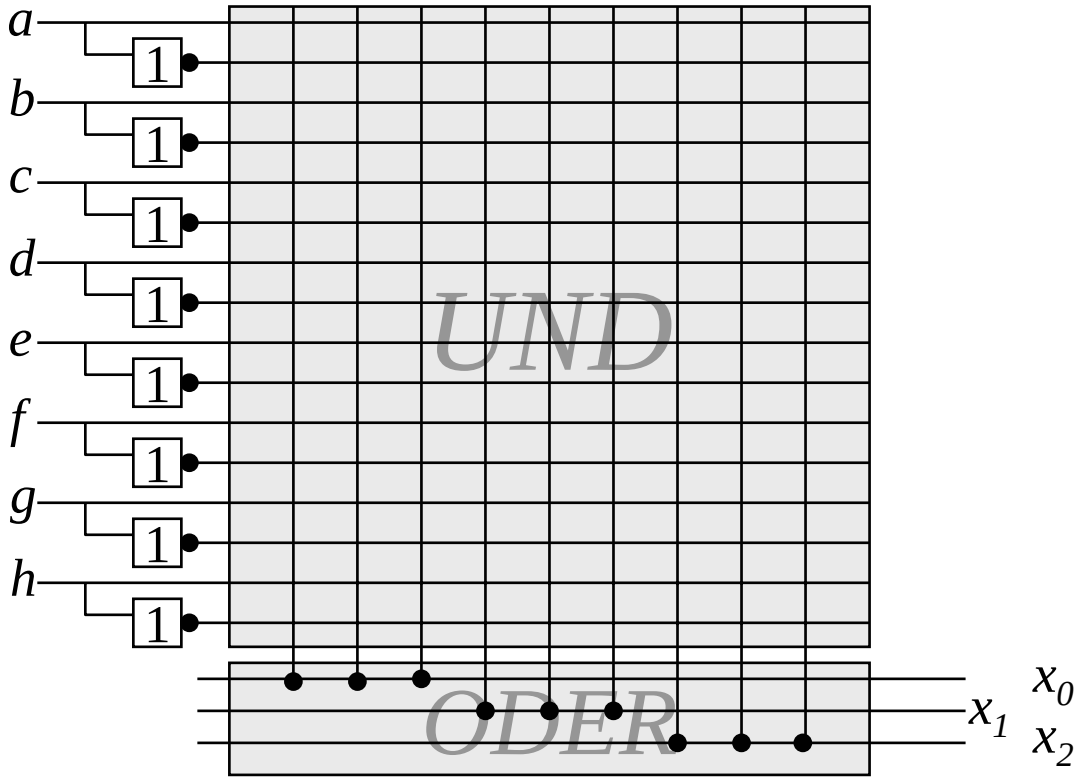
# Realisierung mit einem PLA

- **Programmable Logic Array**
  - ⇨ **technische Realisierung der DMF**
  - ⇨ **UND- und ODER-Matrix sind frei programmierbar**



# Realisierung mit einem PAL

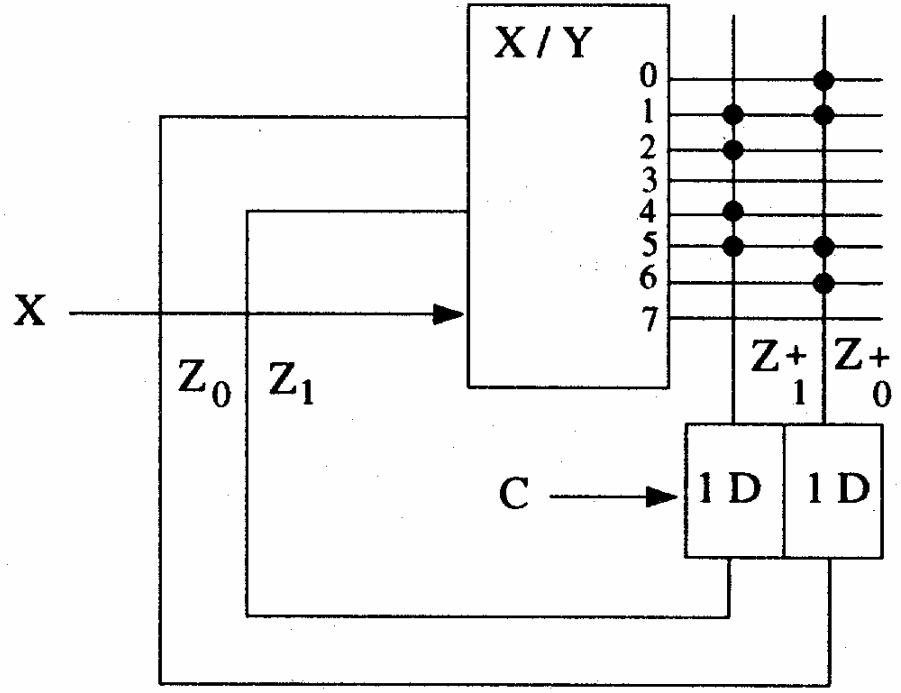
- **Programmable Array Logic**
  - ⇨ **die ODER-Matrix ist vorgegeben**
  - ⇨ **es steht eine feste Anzahl von Implikanten pro Ausgang zur Verfügung**
  - ⇨ **die UND-Matrix ist programmierbar**



# Realisierung mit einem ROM

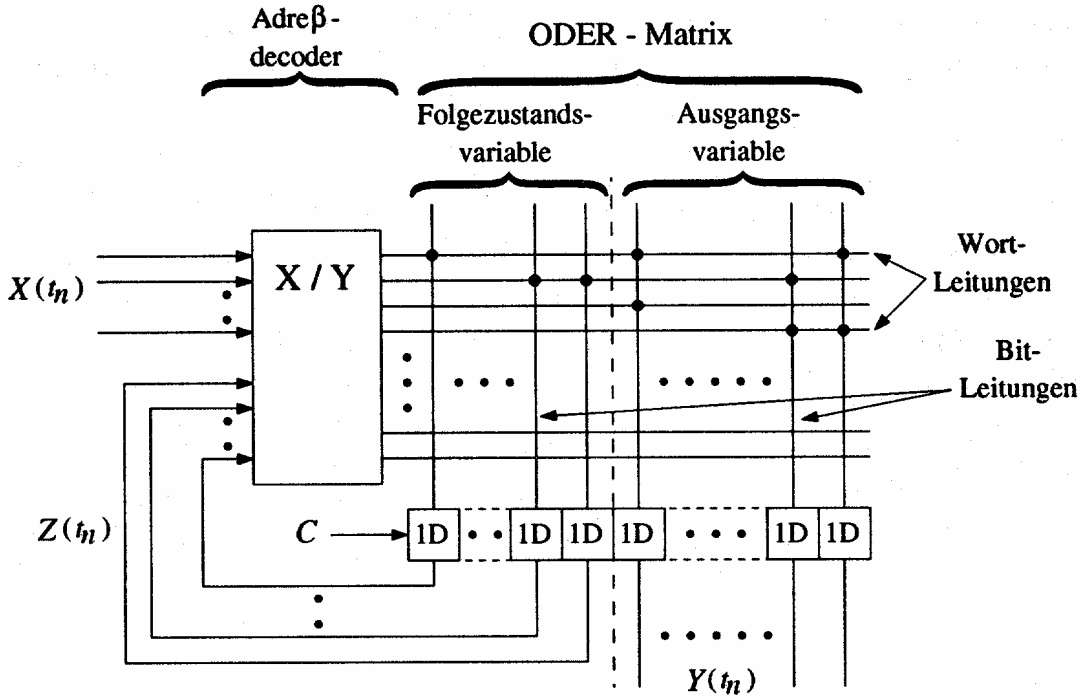
- Technische Realisierung durch ein PROM, EPROM, EEPROM
- Die UND-Matrix ist durch den Adressdekodierer vorgegeben
  - ⇨ alle Minterme sind implementiert
  - ⇨ direkte Implementierung der Funktionstabelle

| $X$ | $Z_1$ | $Z_0$ | $Z_1^+$ | $Z_0^+$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |



# Realisierung mit einem ROM

- **Auch die Ausgabefunktion kann mit einem ROM realisiert werden**
  - ⇨ **Wortorientierung des ROMs wird ausgenutzt**
  - ⇨ **Mikroprogramm**
  - ⇨ **mögliche Implementierung des Steuerwerkes in Mikroprozessoren**

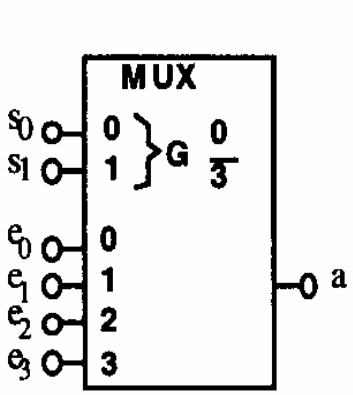


# 4. Spezielle Schaltnetze und Schaltwerke

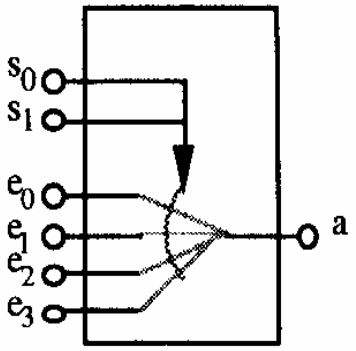
- **Für die Implementierung komplexer Schaltungen werden häufig immer wieder kehrende Bausteintypen verwendet**
- **Typische Schaltnetze sind**
  - ⇨ **Multiplexer/Demultiplexer**
  - ⇨ **Vergleicher**
  - ⇨ **Addierer**
  - ⇨ **Multiplizierer**
- **Typische Schaltwerke sind**
  - ⇨ **Register**
  - ⇨ **Schieberegister**
  - ⇨ **Zähler**

# Multiplexer

- Mehrere Eingänge, ein Ausgang
- über n Steuerleitungen können $2^n$ Eingänge ausgewählt und an den Ausgang durchgeschaltet werden



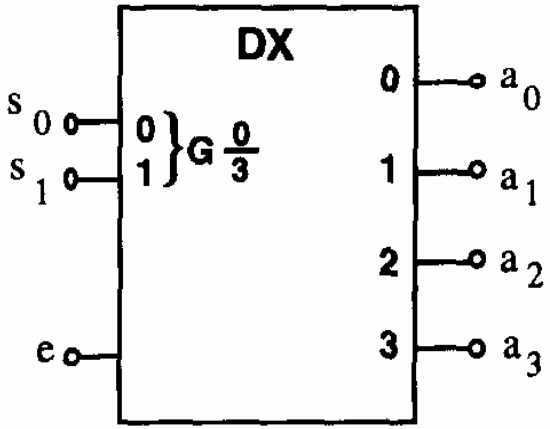

| Steuerleitungen $s_1$ | $s_0$ | Ausgang a |
|---|---|---|
| 0 | 0 | $e_0$ |
| 0 | 1 | $e_1$ |
| 1 | 0 | $e_2$ |
| 1 | 1 | $e_3$ |




**Schaltbild und logisches Verhalten eines 1-aus-4-Multiplexers**

# Demultiplexer

- **Ein Eingang wird auf einen aus $2^n$ Ausgängen durchgeschaltet**

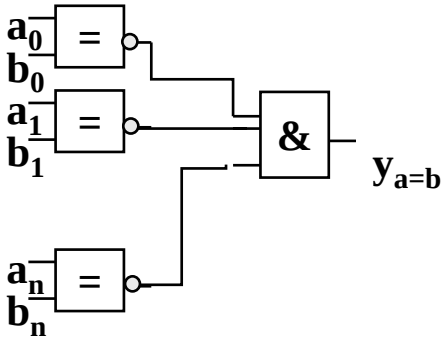



| $s_1$ | $s_0$ | $a_0$ | $a_1$ | $a_2$ | $a_3$ |
|---|---|---|---|---|---|
| 0 | 0 | e | 0 | 0 | 0 |
| 0 | 1 | 0 | e | 0 | 0 |
| 1 | 0 | 0 | 0 | e | 0 |
| 1 | 1 | 0 | 0 | 0 | e |

**Schaltbild und logisches Verhalten eines 1-auf-4-Demultiplexers**

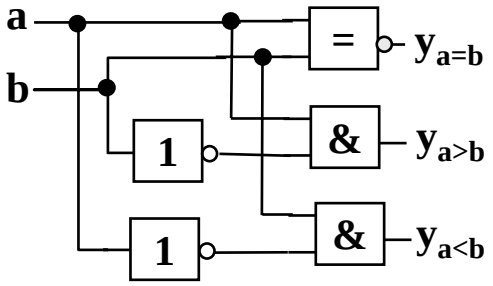
# Vergleicher (Komparatoren)

- **Vergleich zweier Zahlen**
  - ⇨ **A=B, A<B, A>B**
- **Gleichheit bedeutet, dass alle Bits übereinstimmen**



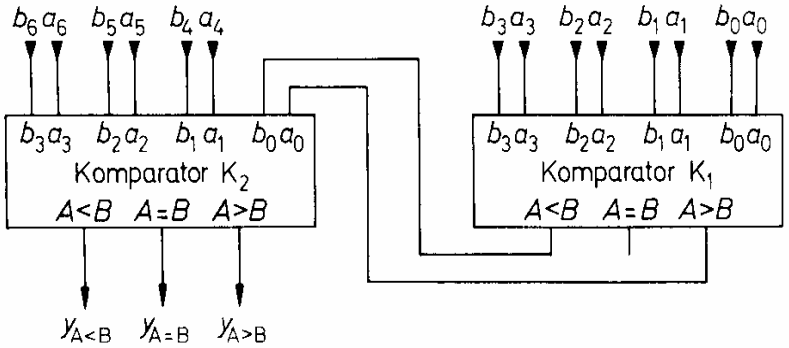

- **1-Bit Komparator mit Größenvergleich**

| $a$ | $b$ | $y_{a>b}$ | $y_{a=b}$ | $y_{a<b}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |

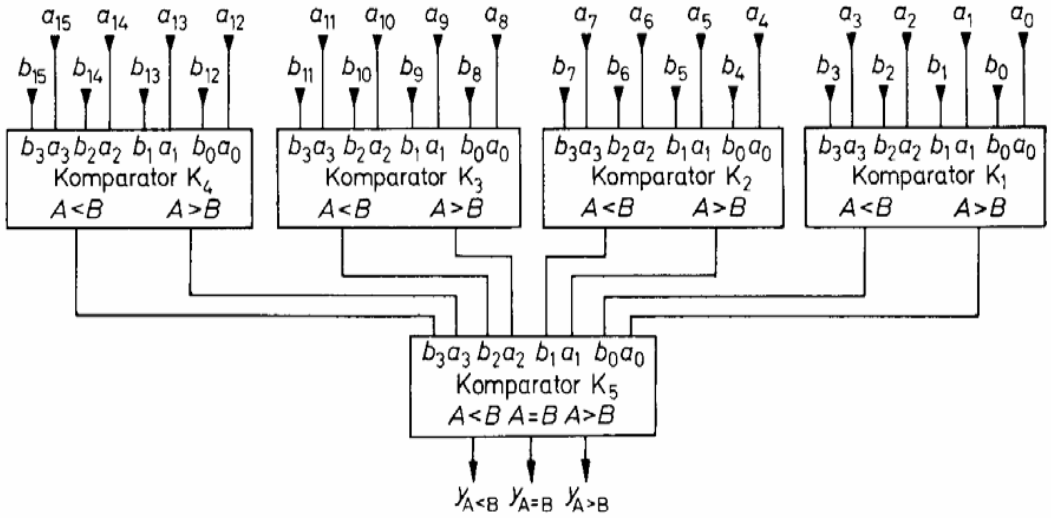


# Komparatoren

- **Serielle Erweiterung**



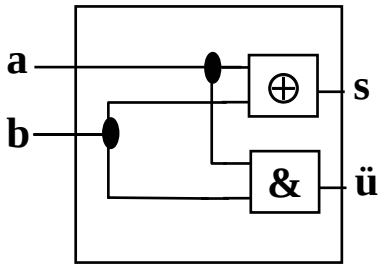

- **Parallele Erweiterung**



# Addierer

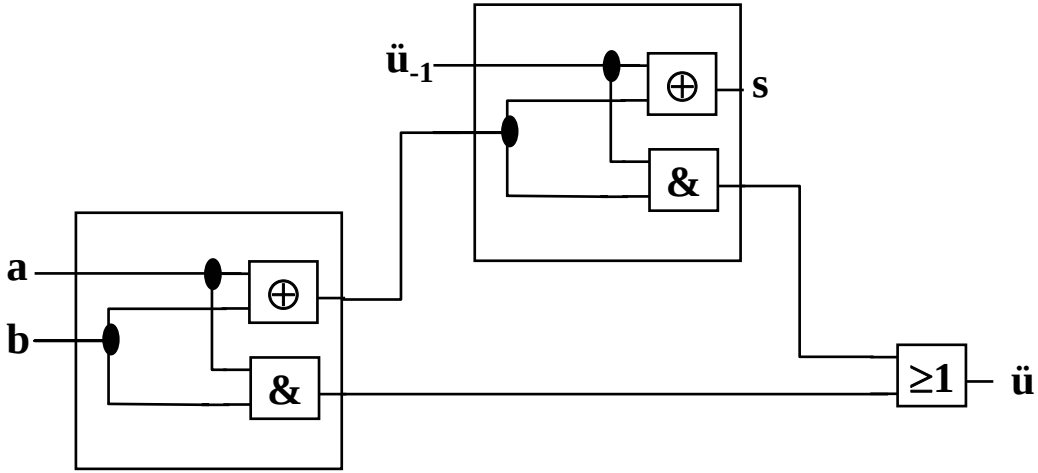
## Halbaddierer

| a | b | s | ü |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |



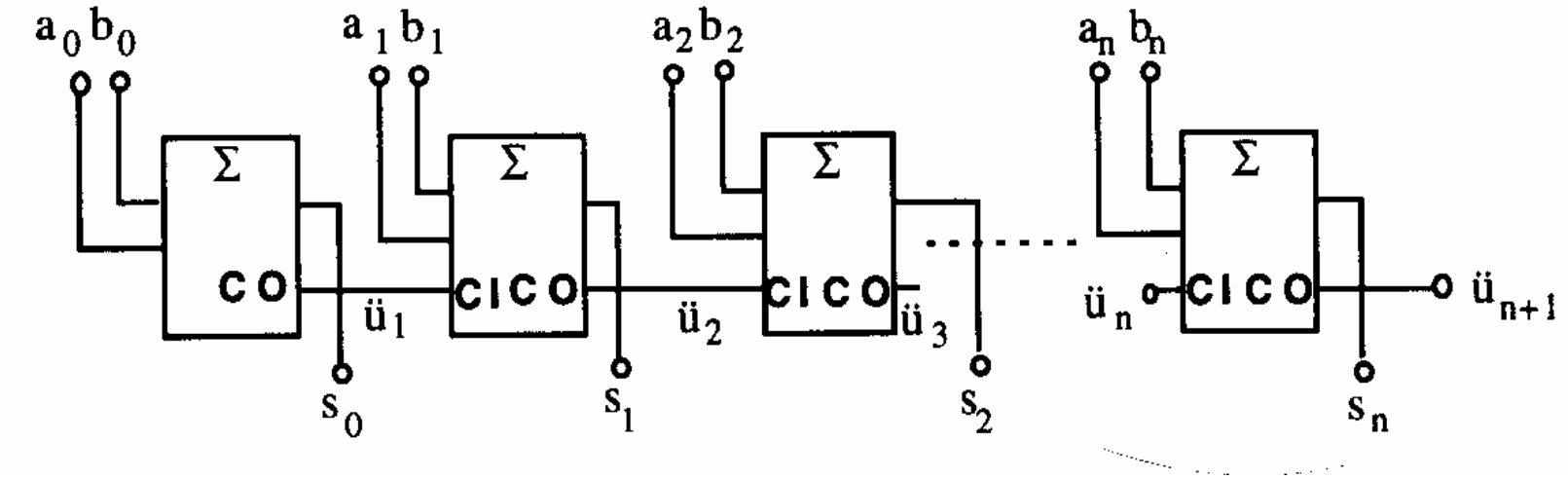

## Volladdierer

| a | b | $ü_{-1}$ | s | ü |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |



# Addition mit seriellem Übertrag

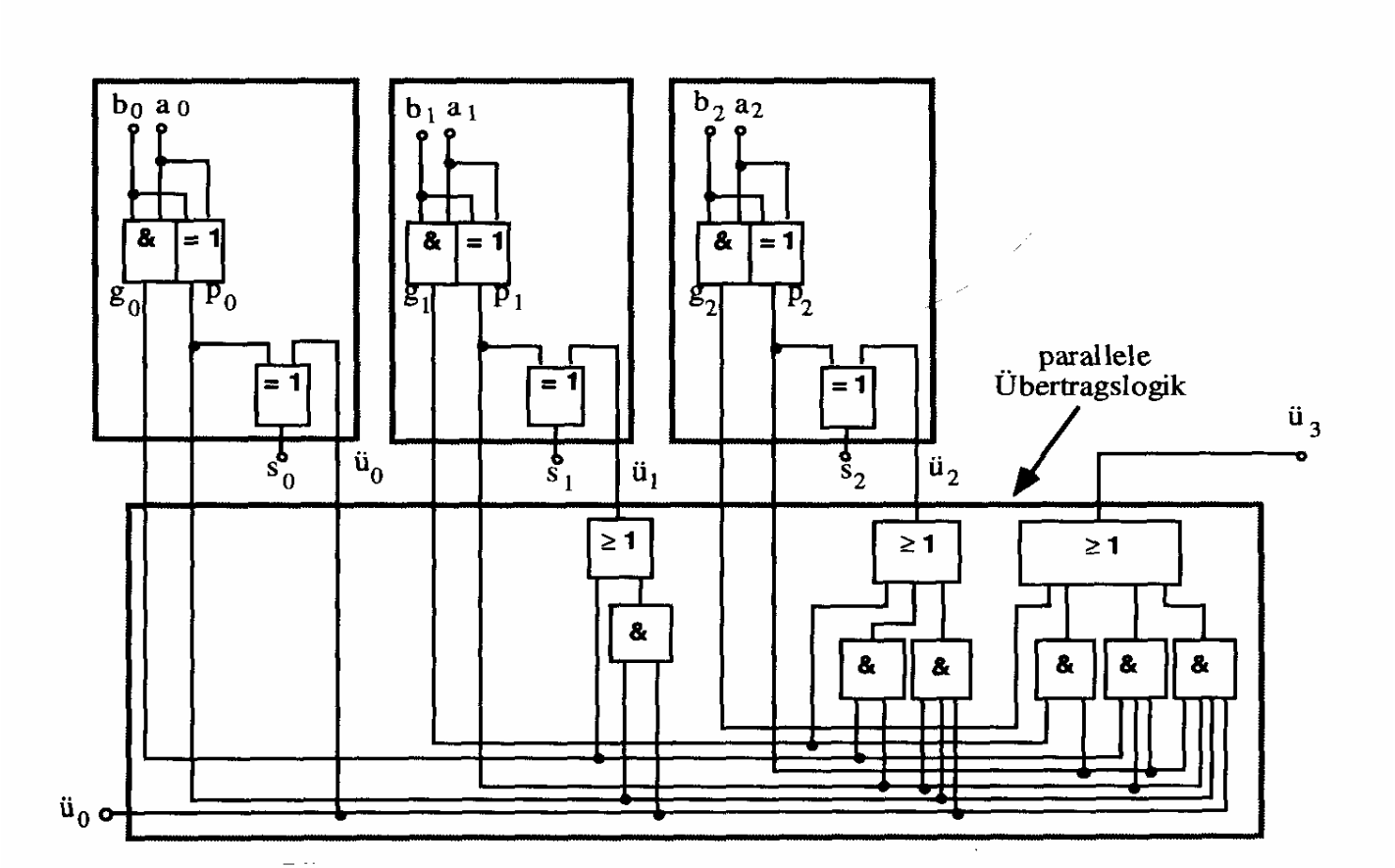
- Der Übertrag des Volladdierers $ü_i$ wird mit $c_{i+1}$ verbunden



# Addierer mit paralleler Übertragslogik

- **Allgemein:**

$$c_i = a_i b_i \vee (a_i \oplus b_i) c_{i-1}$$



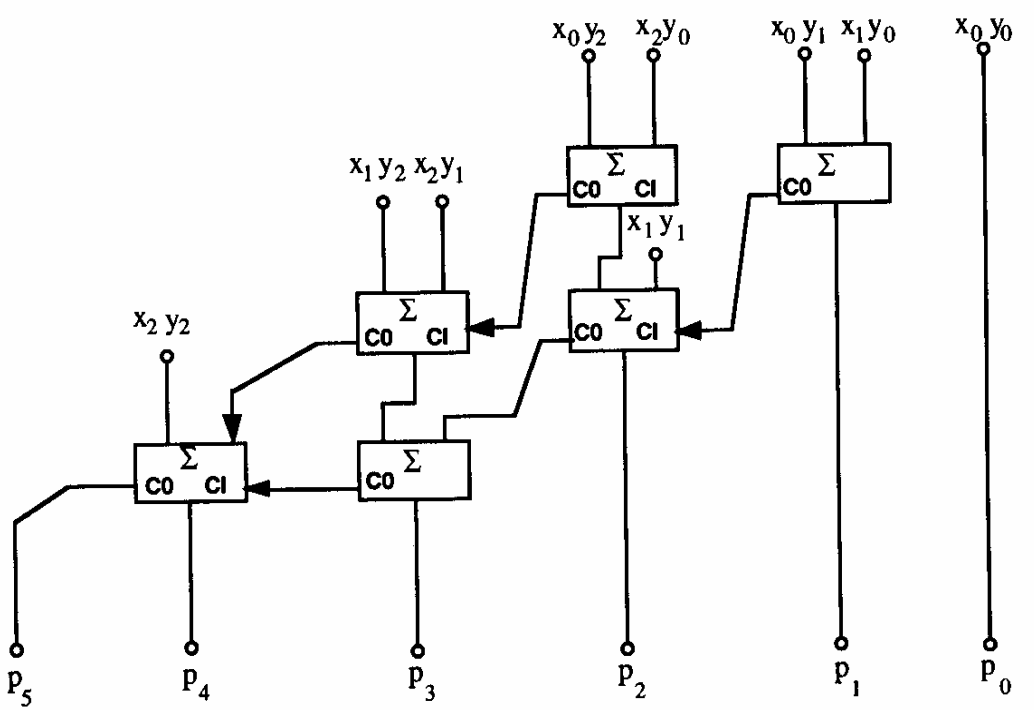
# Multiplizierer

- **Parallele Multiplikation durch Addierwerk**

$$p = x \cdot y = \left( \sum_{i=0}^{n-1} x_i \cdot 2^i \right) \cdot \left( \sum_{j=0}^{n-1} y_j \cdot 2^j \right) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \cdot 2^{i+j} x_i y_j$$

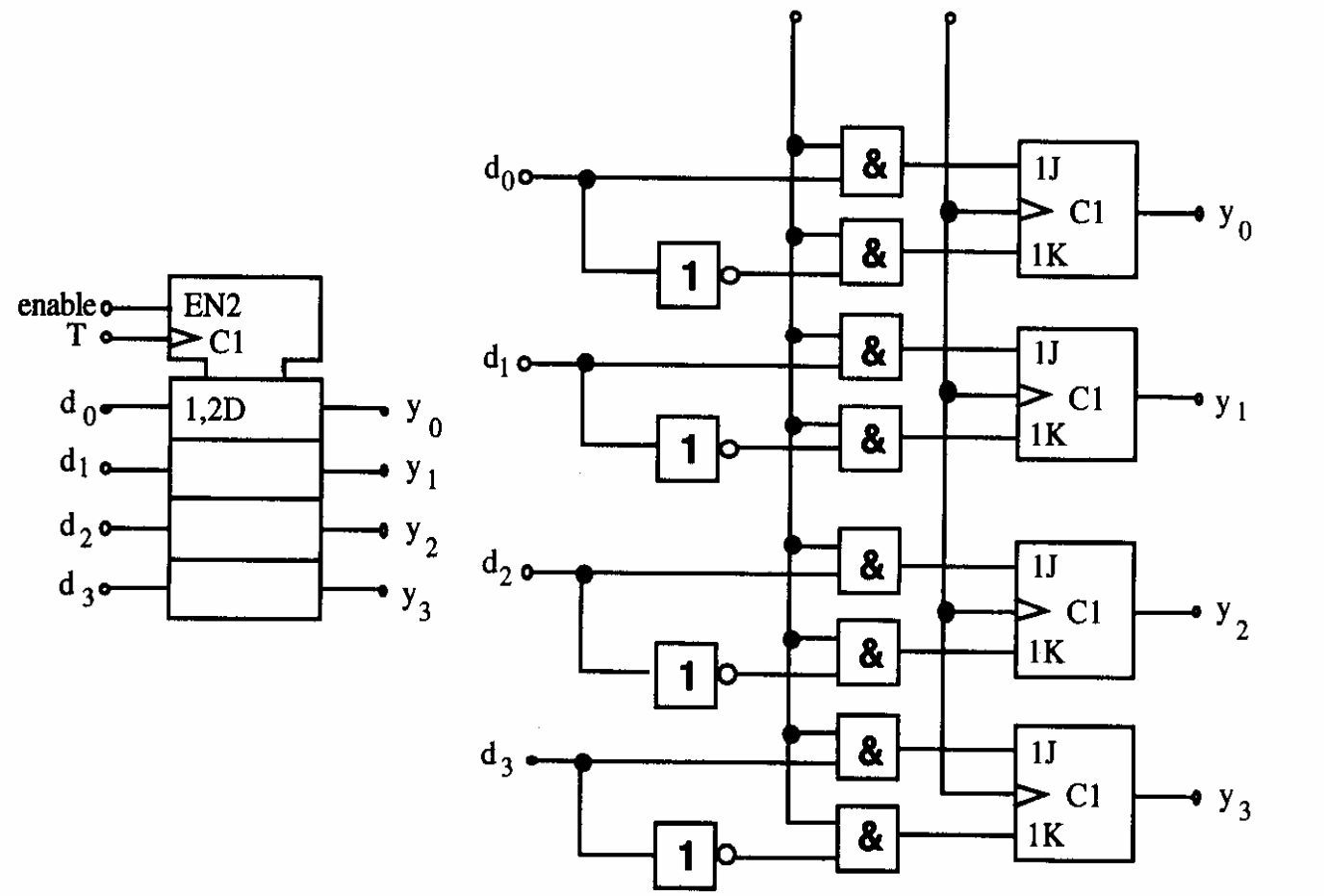
- **für n= 3: ($x_i y_j$ steht für $x_i$ UND $y_i$)**

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | • | 1 | 0 | 1 |
| 1 | 1 | 0 | | | | |
| | 0 | 0 | 0 | | | |
| | | 1 | 1 | 0 | | |
| 1 | 1 | 1 | 1 | 0 | | |



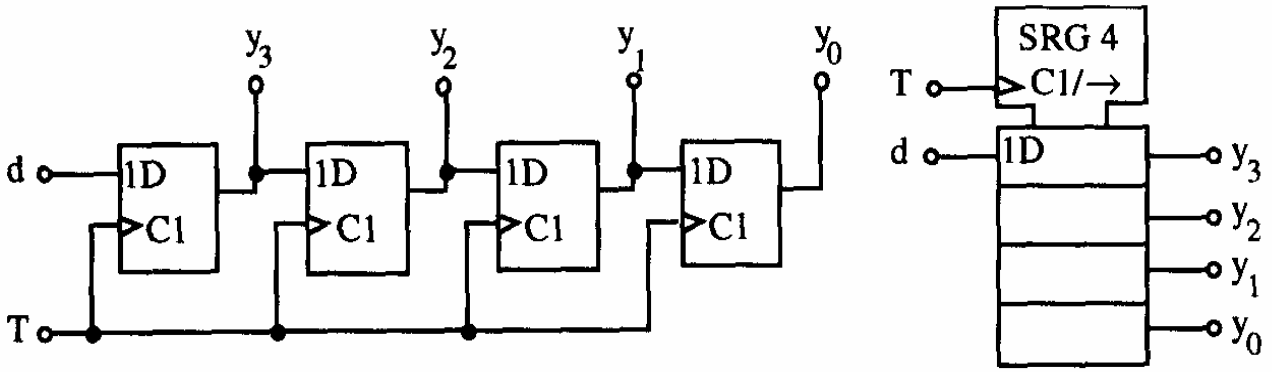
# Register

- **Speicherung einer n-stelligen Zahl durch n Flipflops**



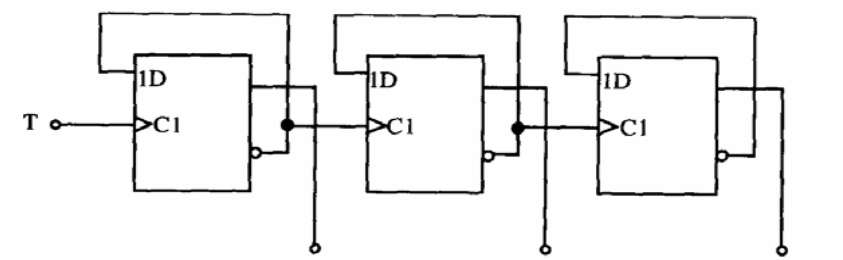
# Schieberegister

- **Kette von Flipflops**
- **Anwendungen:**
  - ⇨ **Serien-Parallel-Wandlung**
  - ⇨ **Parallel-Serien-Wandlung**
  - ⇨ **FIFO oder Stapel-Speicher**
  - ⇨ **Multiplikation mit 2 oder Division durch 2**
  - ⇨ **mit Rückkopplung zur Erzeugung komplexer Signalfolgen (Sequenzer)**

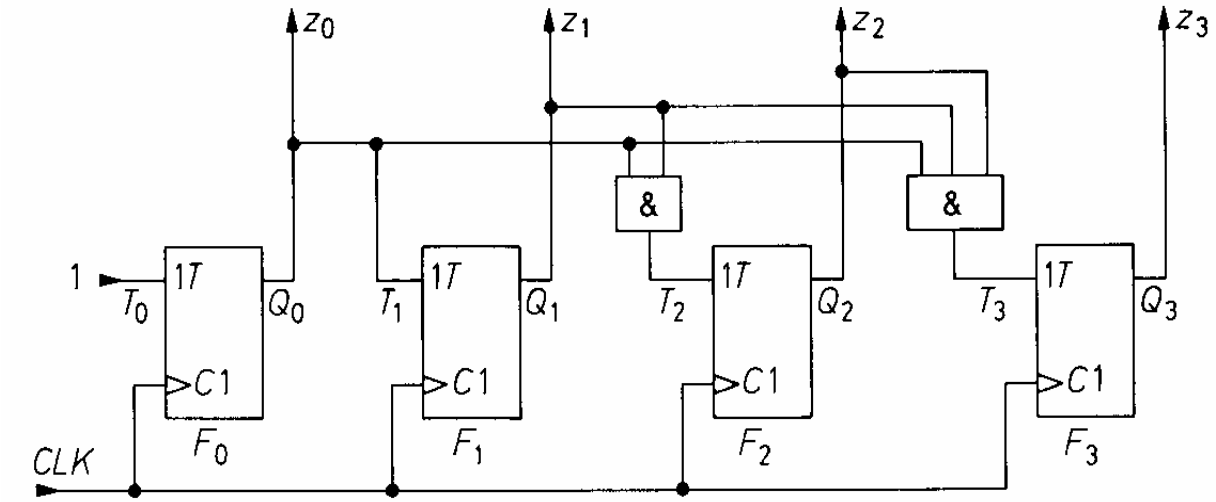


# Zähler

- Einfacher Dualzähler durch Rückkopplung
- Asynchroner Ripple Carry Zähler

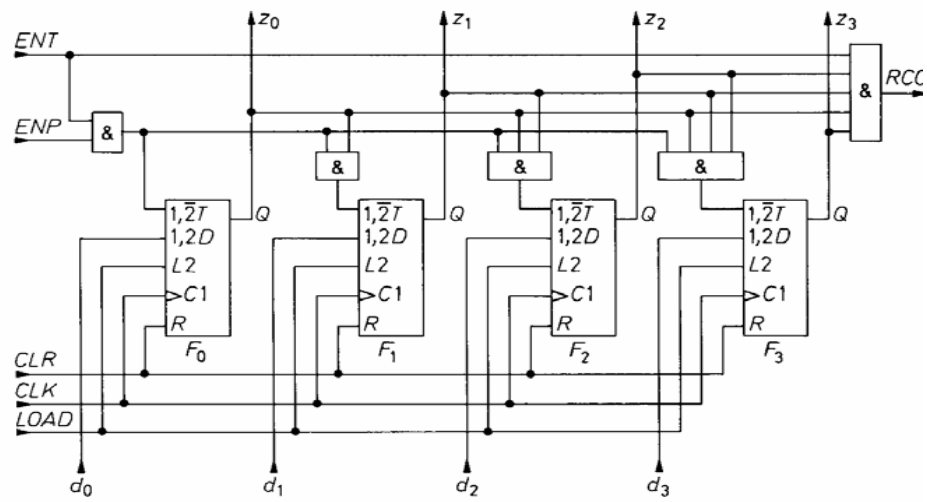



- Synchroner Dualzähler durch Carry-Look-Ahead-Logik

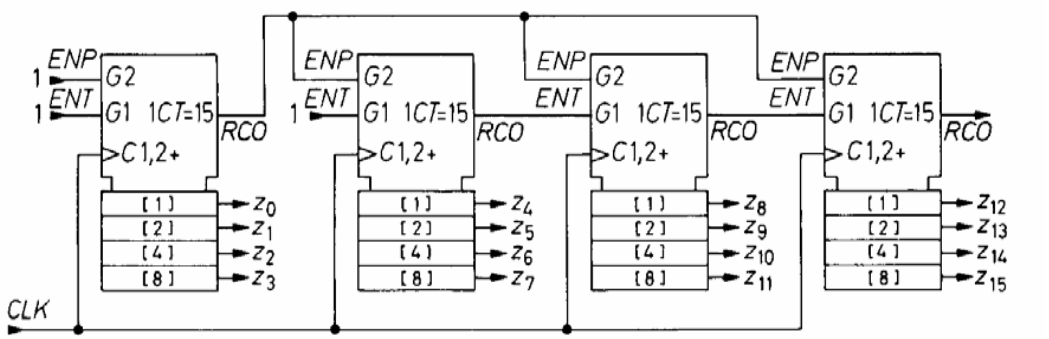


# Zähler

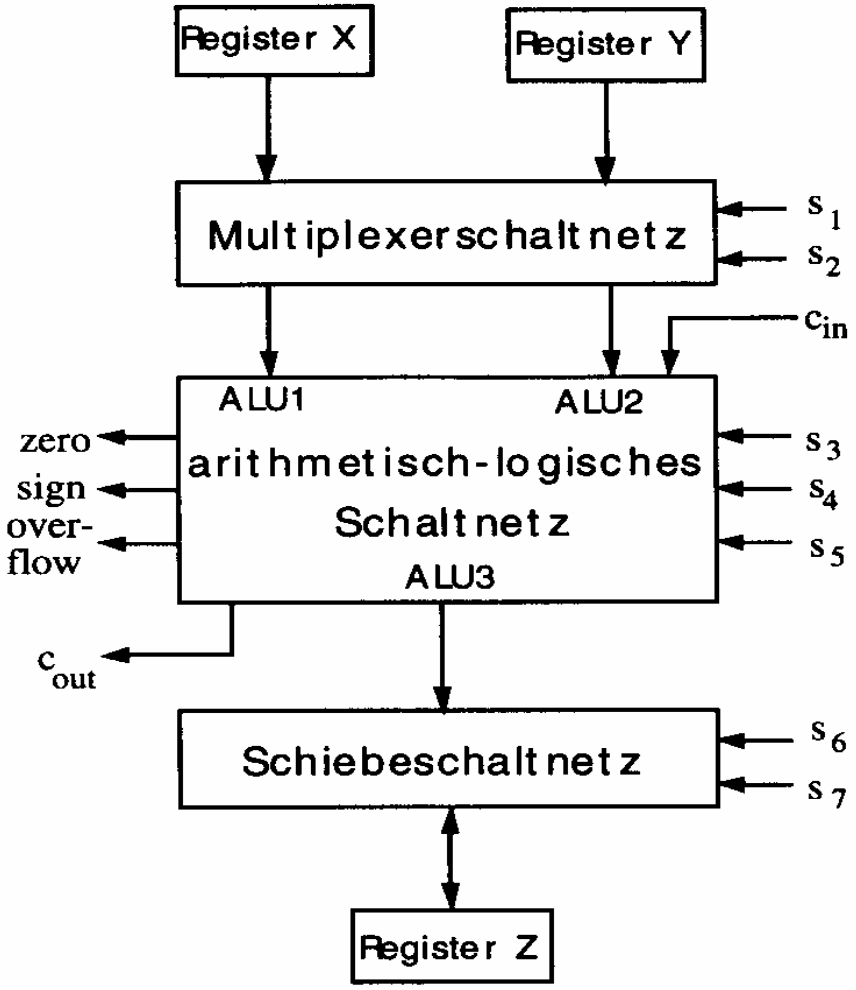
- **Praktische Ausführung eines Zählers**




- **Kaskadierung eines Zählers**



# Aufbau einer ALU




| $s_1$ | $s_2$ | ALU1 | ALU2 |
|---|---|---|---|
| 0 | 0 | X | Y |
| 0 | 1 | X | 0 |
| 1 | 0 | Y | 0 |
| 1 | 1 | Y | X |

| $s_3$ | $s_4$ | $s_5$ | ALU3 |
|---|---|---|---|
| 0 | 0 | 0 | ALU1+ALU2+$c_{in}$ |
| 0 | 0 | 1 | ALU1-ALU2-$\overline{c_{in}}$ |
| 0 | 1 | 0 | ALU2-ALU1-$\overline{c_{in}}$ |
| 0 | 1 | 1 | ALU1 $\vee$ ALU2 |
| 1 | 0 | 0 | ALU1 $\wedge$ ALU2 |
| 1 | 0 | 1 | $\overline{\text{ALU1}}$ $\wedge$ ALU2 |
| 1 | 1 | 0 | ALU1 $\not\leftrightarrow$ ALU2 |
| 1 | 1 | 1 | ALU1 $\leftrightarrow$ ALU2 |

| $s_6$ | $s_7$ | Z |
|---|---|---|
| 0 | 0 | ALU3 |
| 0 | 1 | ALU3 / 2 |
| 1 | 0 | ALU3 * 2 |
| 1 | 1 | Z speichern |

# Bauelemente eines Rechnersystems

- Multiplexer und Demultiplexer zur Steuerung des Datenflusses
- Zähler für die Programmsteuerung
- ALU
  - ⇨ Register
  - ⇨ Addierer
  - ⇨ Multiplizierer
  - ⇨ Schieberegister
- Speicherzellen
  - ⇨ RAM
  - ⇨ ROM

# 5 Rechnerarithmetik

- **Die Rechnerarithmetik behandelt**
  - ⇨ **die Darstellung von Zahlen**
  - ⇨ **Verfahren zur Berechnung der vier Grundrechenarten**
  - ⇨ **Schaltungen, die diese Verfahren implementieren**

# 5.1 Formale Grundlagen

- **Menschen rechnen und denken im Dezimalsystem**
- **Die meisten Rechner verwenden das Dualsystem**
  - ⇨ **man benötigt Verfahren der Konvertierung, die sich algorithmisch umsetzen lassen**

## 7.1.1 Zahlensysteme

- **Stellenwertsysteme**
  - ⇨ **jeder Position $i$ der Ziffernreihe ist ein Stellenwert zugeordnet welcher der Potenz $b^i$ der Basis $b$ eines Zahlensystems entspricht** $z_n z_{n-1} \ldots z_1 z_0 . z_{-1} z_{-2} z_{-m}$
  - ⇨ **der Wert $X_b$ ergibt sich aus der Summe der Werte aller Einzelstellen**

$$X_b = z_n b^n + z_{n-1} b^{n-1} + \ldots + z_1 b + z_0 + z_{-1} b^{-1} + z_{-2} b^{-2} + z_{-m} b^{-m} = \sum_{i=-m}^{n} z_i b^i$$

# Die wichtigsten Zahlensysteme

| b | Zahlensystem | Ziffern |
|---|---|---|
| 2 | Dualsystem | 0, 1 |
| 8 | Oktalsystem | 0, 1, 2, 3, 4, 5, 6, 7 |
| 10 | Dezimalsystem | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 |
| 16 | Hexadezimalsystem | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F |

- Dualsystem kann direkt auf 2-wertige Logik umgewandelt werden
- Oktal- und Hexadezimalsystem sind Kurzschreibweisen der Zahlen im Dualsystem
  - ⇨ sie lassen sich leicht in Zahlen des Dualsystems umwandeln

# Umwandlung vom Dezimalsystem in ein Zahlensystem zur Basis $b$

- **Euklidischer Algorithmus**
  - ⇨ **die einzelnen Ziffern werden sukzessive berechnet**

$$Z = z_n 10^n + z_{n-1}10^{n-1} + \ldots + z_1 10 + z_0 + z_{-1}10^{-1} + z_{-2}10^{-2} + z_{-m}10^{-m}$$

$$= y_p b^p + y_{p-1}b^{p-1} + \ldots + y_1 b + y_0 + y_{-1}b^{-1} + y_{-2}b^{-2} + y_{-q}b^{-q}$$

  - ⇨ **Algorithmus**
    1. **Berechne P gemäß der Ungleichung** $b^{p-1} \le Z < b^p$
    2. **Ermittle** $y_p$ **und den Rest** $R_p$ **durch Division von** $Z$ **durch** $b^p$

       $y_p = Z \; div \; b^p;$ $\quad R_p = Z \; mod \; b^p;$ $\quad y_p = \{0, 1, \ldots, b\text{-}1\}$
    3. **Wiederhole 2. für** $i = p\text{-}1$ **und ersetze dabei nach jedem Schritt** $Z$ **durch** $R_i$**, bis** $R_i = 0$ **oder bis** $b_i$ **klein genug ist**

# Beispiel

- **Umwandlung von** $15741{,}233_{10}$ **ins Hexadezimalsystem**

| | | |
|---|---|---|
| **1. Schritt** | $16^3 \leq 15741{,}233_{10} < 16^4$ | **höchste Potenz** $16^3$ |
| **2. Schritt** | $15741{,}233_{10} : 16^3 = 3$ | **Rest** 3453,233 |
| **3. Schritt** | $3453{,}233 : 16^2 = D$ | **Rest** 125,233 |
| **4. Schritt** | $125{,}233 : 16 = 7$ | **Rest** 13,233 |
| **5. Schritt** | $13{,}233 : 1 = D$ | **Rest** 0,233 |
| **6. Schritt** | $0{,}233 : 16^{-1} = 3$ | **Rest** 0,0455 |
| **7. Schritt** | $0{,}0455 : 16^{-2} = B$ | **Rest** 0,00253 |
| **8. Schritt** | $0{,}00253 : 16^{-3} = A$ | **Rest** 0,000088593 |
| **9. Schritt** | $0{,}000088593 : 16^{-4} = 5$ | **Rest** 0,000012299 |

**Fehler**

**Ergebnis:** $15741{,}233_{10} = 3D7D{,}3BA5_{16}$

# Umwandlung vom Dezimalsystem in eine Zahl zur Basis $b$

- **Horner-Schema**
  - **Eine ganze Zahl $X_b$ kann auch in der folgenden Form dargestellt werden:**

$$X_b = ((...(((y_n b + y_{n-1})b + y_{n-2})b + y_{n-3})b...)b + y_1)b + y_0$$

- **Die gegebene Dezimalzahl wird sukzessive durch die Basis $b$ dividiert**
  - **Die jeweiligen ganzzahligen Reste ergeben die Ziffern der Zahl $X_b$**
  - **Reihenfolge: niederwertige zur höchstwertige Stelle**
- **Beispiel: Umwandlung von $15741_{10}$ ins Hexadezimalsystem**

| | | |
|---|---|---|
| $15741_{10} : 16 = 983$ | **Rest** 13 | $(D_{16})$ |
| $983_{10} : 16 = 61$ | **Rest** 7 | $(7_{16})$ |
| $61_{10} : 16 = 3$ | **Rest** 13 | $(D_{16})$ |
| $3_{10} : 16 = 0$ | **Rest** 3 | $(3_{16})$ |

**Ergebnis:** $15741_{10} = 3D7D_{16}$

# Umwandlung des Nachkommateils

- **Der Nachkommateil einer Zahl $X_b$ kann in der folgenden Form dargestellt werden**

$$Y_b = ((...((y_{-m}b^{-1} + y_{-m+1})b^{-1} + y_{-m+2})b^{-1} + ... + y_{-2})b^{-1} + y_{-1})b^{-1}$$

- **sukzessive Multiplikation des Nachkommateils der Dezimalzahl mit der Basis $b$ des Zielsystems ergibt nacheinander die $y_{-i}$**
- **Beispiel: Umwandlung von $0{,}233_{10}$ ins Hexadezimalsystem**

| | | |
|---|---|---|
| 0,233 * 16 | = 3,728 | $z_{-1} = 3$ |
| 0,728 * 16 | = 11,648 | $z_{-2} = B$ |
| 0,648 * 16 | = 10,368 | $z_{-3} = A$ |
| 0,368 * 16 | = 5,888 | $z_{-4} = 5$ |

**Ergebnis:** $0{,}233_{10} = 0{,}3BA5_{16}$

# Umwandlung einer Zahl zur Basis *b* ins Dezimalsystem

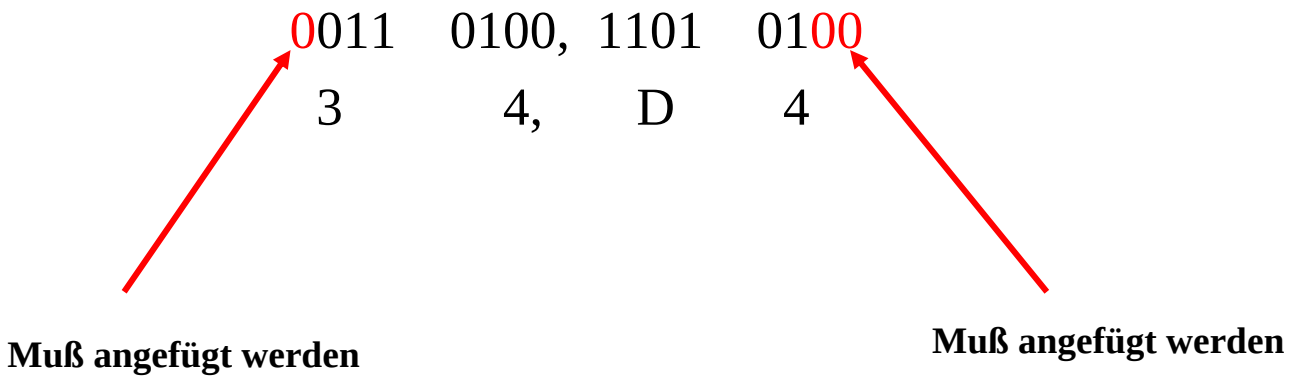
- **Werte der einzelnen Stellen werden mit deren Wertigkeit multipliziert und aufsummiert**
- **Beispiel: Umwandlung von** 101101,1101 **ins Dezimalsystem**

101101,1101

| | | |
|---|---|---|
| $1 * 2^{-4}$ | = | 0,0625 |
| $0 * 2^{-3}$ | = | 0 |
| $1 * 2^{-2}$ | = | 0,25 |
| $1 * 2^{-1}$ | = | 0,5 |
| $1 * 2^{-0}$ | = | 1 |
| $0 * 2^{1}$ | = | 0 |
| $1 * 2^{2}$ | = | 4 |
| $1 * 2^{3}$ | = | 8 |
| $0 * 2^{4}$ | = | 0 |
| $1 * 2^{5}$ | = | 32 |
| | | $45{,}8125_{10}$ |

# Weitere Umwandlungen

- **Umwandlung zwischen zwei beliebigen Zahlensystemen**
  - ⇨ **zwei Schritte: Umwandlung ins Dezimalsystem und danach vom Dezimalsystem ins Zielsystem**
- **Spezialfall: Eine Basis eine Potenz der anderen Basis**
  - ⇨ **Umwandlung erfolgt durch Zusammenfassen der Stellen**
  - ⇨ **Beispiel: Umwandlung von** $0110100{,}110101_2$ **ins Hexadezimalsystem**



# 5.1.2 Kodierung zur Zahlen- und Zeichendarstellung

- **Die Dezimalzahlen können auch ziffernweise in eine Binärdarstellung überführt werden**
  - ⇨ **um die 10 Ziffern 0 bis 9 darstellen zu können, benötigt man 4 Bit**
  - ⇨ **eine solche 4er-Gruppe wird Tetrade genannt**
    - **Pseudotetraden: 6 der 16 Kodierungen stellen keine gültigen Ziffern dar**
- **BCD**
  - ⇨ **Binary Coded Decimals**
  - ⇨ **man verwendet das Dualäquivalent der ersten 10 Dualzahlen**
  - ⇨ **Beispiel:**

$$8127_{10} = 1000\ 0001\ 0010\ 0111_{BCD} = 1111110111111_2$$

  - ⇨ **Nachteile der BCD-Kodierung**
    - **höherer Platzbedarf**
    - **aufwändige Implementierung der Rechenoperationen**

# Grey-Kodierung

- **Einschrittige Kodierung**
  - ⇨ **bei benachbarten Zahlen ändert sich nur ein Binärzeichen**
- **Vorteil**
  - ⇨ **keine Hazards bei der Analog/Digitalwandlung und bei Abtastern**
- **Nachteil**
  - ⇨ **keine Stellenwertigkeit**
  - ⇨ **aufwändige Rechenoperationen**

| Dezimalzahl | Grey-Kodierung |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0011 |
| 3 | 0010 |
| 4 | 0110 |
| 5 | 0111 |
| 6 | 0101 |
| 7 | 0100 |
| 8 | 1100 |
| 9 | 1101 |
| 10 | 1111 |
| 11 | 1110 |
| 12 | 1010 |
| 13 | 1011 |
| 14 | 1001 |
| 15 | 1000 |

# Kodierung von Zeichen

- American Standard Code for Information Interchange (ASCII)
  - ⇨ 7 Bit-Kodierung für 128 Zeichen
  - ⇨ 2*26 Zeichen, 10 Ziffern und
    32 Kommunikationssteuerzeichen
- Umlaute und Sonderzeichen sind nicht enthalten
  - ⇨ 8-Bit Erweiterungen unterschiedlicher Computerhersteller
  - ⇨ Andere Verwendung des 8. Bits: Paritätsprüfung

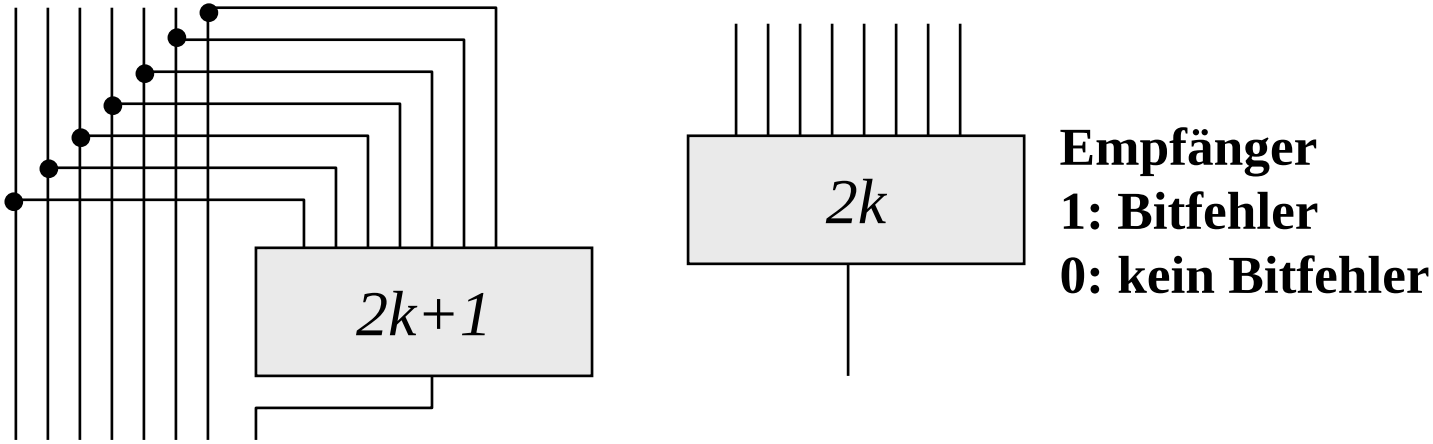
# ASCII-Tabelle

| | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| 0000 | NUL | DLE | SPACE | 0 | @ | P | ' | p |
| 0001 | SOH | DC 1 | ! | 1 | A | Q | a | q |
| 0010 | STX | DC 2 | " | 2 | B | R | b | r |
| 0011 | ETX | DC 3 | # | 3 | C | S | c | s |
| 0100 | EOT | DC 4 | $ | 4 | D | T | d | t |
| 0101 | ENQ | NAK | % | 5 | E | U | e | u |
| 0110 | ACK | SYN | & | 6 | F | V | f | v |
| 0111 | BEL | ETB | ´ | 7 | G | W | g | w |
| 1000 | BS | CAN | ( | 8 | H | X | h | x |
| 1001 | HT | EM | ) | 9 | I | Y | i | y |
| 1010 | LF | SUB | * | : | J | Z | j | z |
| 1011 | VT | ESC | + | ; | K | [ | k | { |
| 1100 | FF | FS | , | < | L | \ | l | \| |
| 1101 | CR | GS | - | = | M | ] | m | } |
| 1110 | SO | RS | . | > | N | ^ | n | ~ |
| 1111 | SI | US | / | ? | O | _ | o | DEL |

Die höchstwertigen Bits der Kodierung eines Zeichens sind in der Kopfzeile abzulesen, die niederwertigen Bits in der ersten Spalte (Beispiel: A $\rightarrow 100\,0001_2$).

# Paritätsprüfung

- **Problem:**
  - ⇨ **Erkennung von Übertragungsfehlern**
- **Prinzip:**
  - ⇨ **die 7-Bit Kodierung wird beim Sender so auf 8 Bit ergänzt, dass stets eine gerade (ungerade) Anzahl von Einsen ergänzt**
    - **gerade (ungerade) Parität**
  - ⇨ **beim Empfänger wird diese Eigenschaft überprüft**
    - **falls bei der Übertragung ein Bitfehler auftritt, wird dieser erkannt**



# Beispiel: Paritätsprüfung

- **Gerade Parität (even parity)**
  - ⇨ **Paritätsbit ‚0' wenn gerade, ‚1' wenn ungerade**
  - ⇨ **Codewort 0110110 hat 4 ‚1', also gerade →00110110**
  - ⇨ **Codewort 0111110 hat 5 ‚1', also gerade →10110110**

- **Ungerade Parität (odd parity)**
  - ⇨ **Paritätsbit ‚0' wenn ungerade, ‚1' wenn gerade**
  - ⇨ **Codewort 0110110 hat 4 ‚1', also gerade →10110110**
  - ⇨ **Codewort 0111110 hat 5 ‚1', also gerade →00110110**

# 5.1.3 Darstellung negativer Zahlen

- **Für die Darstellung von Zahlen in Rechnern werden vier verschiedene Formate benutzt**
  - ⇨ **Darstellung mit Betrag und Vorzeichen**
  - ⇨ **Stellenkomplement (Einerkomplement)**
  - ⇨ **Zweierkomplement**
  - ⇨ **Offset-Dual-Darstellung (Charakteristik)**

# Darstellung mit Betrag und Vorzeichen

- Die erste Stelle der Zahl wird als Vorzeichen benutzt
  - ⇨ 0: Die Zahl ist positiv
  - ⇨ 1: Die Zahl ist negativ
- Beispiel:
  - ⇨ `0001 0011 = + 19`
  - ⇨ `1001 0011 = - 19`
- Nachteile dieser Darstellung
  - ⇨ bei Addition und Subtraktion müssen die Vorzeichen getrennt betrachtet werden
  - ⇨ es gibt 2 Repräsentanten der Zahl 0
    - positives und negatives Vorzeichen

# Einerkomplement

- **Jede Ziffer der Binärzahl wird negiert**
  - ⇨ **negative Zahlen werden ebenfalls durch eine 1 an der 1. Stelle gekennzeichnet**
- **Vorteil:**
  - ⇨ **die 1. Stelle muss bei Addition und Subtraktion nicht gesondert betrachtet werden**
- **Beispiel:**

| | | | | |
|---|---|---|---|---|
| | 2 | | 0010 | |
| + | -3 | + | 1100 | (Komplement: 0011) |
| = | -1 | = | 1110 | (Komplement: 0001) |

- **Nachteil:**
  - ⇨ **es gibt 2 Repräsentanten der Zahl 0:**
    - **0000 und 1111**

# Zweierkomplement

- **Addiert man zum Einerkomplement noch 1 hinzu, dann fallen die beiden Darstellungen der Zahl 0 durch den Überlauf wieder aufeinander**
  - **Die Zahl 0** `0000`
  - **Einerkomplement** `1111`
  - **Zweierkomplement** `1111 + 0001 = 0000`
- **Vorteile**
  - **das 1. Bit enthält das Vorzeichen**
  - **direkte Umwandlung der Zahl $Z$ über die Stellenwertigkeit**

- **Beispiel** $Z = -z_n \cdot 2^n + z_{n-1} \cdot 2^{n-1} + \ldots + z_1 \cdot 2 + z_0$
  - **Die Zahl** 54 = $00110110_2$
  - **mit Vorzeichenbit** $-54_{10}$ = $10110110_2$
  - **Einerkomplement** = $11001001_2$
  - **Zweierkomplement** = $11001010_2$

# Addition im Zweierkomplement

- **Beispiel:**

```
    73        01001001
   -54        11001010
  ------------------------
 =  19      (1)00010011
```

- **Beispiel:**

```
    37        00100101
   -54        11001010
  ------------------------
 = -17        11101111     (00010001)
```

# Charakteristik

- **Hauptsächlich in der Darstellung von Exponenten für Gleitkommazahlen**
  - ⇨ **der gesamte Zahlenbereich wird durch die Addition einer Konstanten so nach oben verschoben, dass die kleinste Zahl die Darstellung 0...0 erhält**

- **Übersicht der Zahlendarstellungen**

| Dez. | Betrag mit Vorz. | Einerkomp. | Zweierkomp. | Charakteristik |
|---|---|---|---|---|
| -4 | --- | --- | 100 | 000 |
| -3 | 111 | 100 | 101 | 001 |
| -2 | 110 | 101 | 110 | 010 |
| -1 | 101 | 110 | 111 | 011 |
| 0 | 100 oder 000 | 000 oder 111 | 000 | 100 |
| 1 | 001 | 001 | 001 | 101 |
| 2 | 010 | 010 | 010 | 110 |
| 3 | 011 | 011 | 011 | 111 |

# 5.1.4 Fest- und Gleitkommazahlen

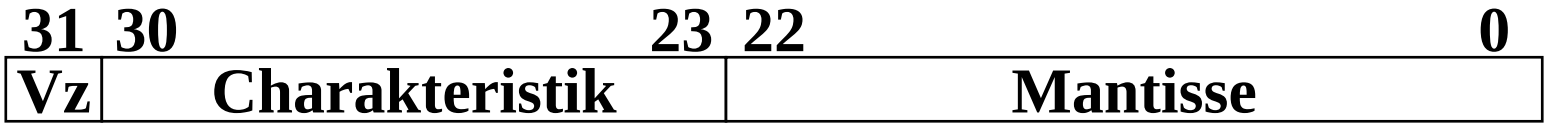
- Darstellung von Zahlen mit einem Komma
- Festkommadarstellung
  - ⇨ Festlegung der Stelle in einem Datenwort

| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | , | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

  - ⇨ wird heute hardwareseitig nicht mehr eingesetzt
- Gleitkommadarstellung
  - ⇨ Angabe der Stelle des Kommas in der Zahlendarstellung

$$Z = \pm \text{Mantisse} \cdot b^{\text{Exponent}}, b \in \{2,16\}$$

  - ⇨ negative Zahlen werden meist in Betrag und Vorzeichen dargestellt (kein Zweierkomplement)
  - ⇨ sowohl für die Mantisse als auch für die Charakteristik wird eine feste Anzahl von Speicherstellen vorgesehen



# Normalisierte Gleitkommadarstellung

- **Eine Gleitkommazahl heißt normalisiert, wenn die folgende Beziehung gilt:**

$$1 \le \text{Mantisse} < b \quad \text{oder} \quad \frac{1}{b} \le \text{Mantisse} < 1 \qquad \frac{1}{2} \le \text{Mantisse} < 1$$

  - ⇨ **bei allen Zahlen außer der 0 ist die erste Stelle hinter dem Komma immer 1**
  - ⇨ **legt man für die Zahl 0 ein festes Bitmuster fest, kann man die erste 1 nach dem Komma weglassen**
- **Beispiel: Die Zahl** $7135_{10}$
  - ⇨ **Festkommazahl**

    `0 000 0000 0000 0000 0001 1011 1101 1111`$_2$
  - ⇨ **Gleitkommadarstellung, normiert**

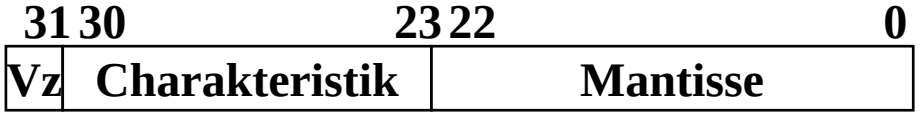
| | | |
|---|---|---|
| `0` | `100 0110 1` | `110 1111 0111 1100 0000 0000` |

  - ⇨ **Gleitkommadarstellung, normiert, implizite erste 1**

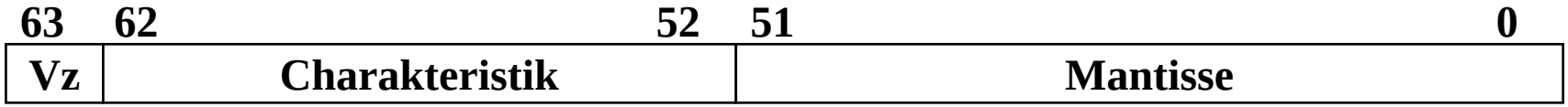
| | | |
|---|---|---|
| `0` | `100 0110 1` | `101 1110 1111 1000 0000 0000` |

# IEEE Gleitkommadarstellung

- Auch bei gleicher Wortbreite lassen sich unterschiedliche Gleitkommaformate definieren
  - ⇨ Normung durch IEEE
  - ⇨ einfache Genauigkeit (32 Bit)




  - ⇨ doppelte Genauigkeit (64 Bit)




- Eigenschaften
  - ⇨ Basis b ist gleich 2
  - ⇨ das erste Bit wird implizit zu 1 angenommen, wenn die Charakteristik nicht nur Nullen enthält
  - ⇨ Es wird so normalisiert, dass das erste Bit vor dem Komma steht

# IEEE Gleitkommadarstellung

- **Zusammenfassung des 32-bit IEEE-Formats:**

| Charakteristik | Zahlenwert |
|---|---|
| 0 | $(-1)^{Vz}$ 0,Mantisse*$2^{-126}$ |
| 1 | $(-1)^{Vz}$ 1,Mantisse*$2^{-126}$ |
| ... | $(-1)^{Vz}$ 1,Mantisse*$2^{\text{Charakteristik-127}}$ |
| 254 | $(-1)^{Vz}$ 1,Mantisse*$2^{127}$ |
| 255 | Mantisse = 0: overflow, $(-1)^{Vz}\infty$ |
| 255 | Mantisse $\neq$ 0: NaN (not a number) |

- **Um Rundungsfehler zu vermeiden, wird intern mit 80 Bit gerechnet**

# 5.2 Addition und Subtraktion

- **Addition erfolgt Hilfe von Volladdierern wie im letzten Abschnitt beschrieben**
  - ⇨ **Ripple-Carry oder Carry-Look-Ahead Addierer**
- **Für die Subtraktion können ebenfalls Volladdierer verwendet werden**
  - ⇨ `X -Y = X + (-Y)`
  - ⇨ **Zweierkomplement berechnet sich über die Negation aller Bits mit einer 1 am ersten Übertrag des Addierers**
- **Bei Gleitkommazahlen müssen Mantisse und Exponent separat betrachtet werden**
  - ⇨ **Angleichen der Exponenten: Bilde die Differenz der Exponenten und verschiebe die Mantisse, die zum kleineren Exponenten gehört um die entsprechende Anzahl nach rechts**
  - ⇨ **Addition der Mantissen**
  - ⇨ **Normalisierung**

# 5.3 Multiplikation und Division

- **Prinzip der Multiplikation: Schieben und Addieren**
- **Multiplikation von Zahlen im Zweierkomplement:**
  - ⇨ **die Zahlen werden in eine Form mit Betrag und Vorzeichen konvertiert**
  - ⇨ **die Beträge werden Multipliziert (kaskadiertes Addierwerk)**
  - ⇨ **das neue Vorzeichen wird berechnet (Exklusiv-ODER-Verknüpfung)**
- **Prinzip der Division: Schieben und Subtrahieren**
  - ⇨ **zwei Sonderfälle:**
    - **Division durch 0 muss eine Ausnahme auslösen**
    - **Die Division muss abgebrochen werden, wenn die vorgegebene Bitzahl des Ergebnisregisters ausgeschöpft ist**