

<b>Strukturierte Programmierung</b> .....	<b>2</b>
<b>Unified Modeling Language 1/3</b> .....	<b>3</b>
<b>UML 2/3</b> .....	<b>4</b>
<b>UML 3/3</b> .....	<b>5</b>
Grundfunktionen (Logik) .....	6
typische Schaltnetze (Blockschaltbilder) .....	7
<b>Programmablaufplan (PAP)</b> .....	<b>10</b>
<b>Abfrageformulierung mit SQL</b> .....	<b>11</b>
1. Projektion und Formatierung .....	11
2. Selektion.....	11
3. Verbund von Tabellen .....	12
4. Aggregatfunktionen und Gruppen .....	13
<b>ER-Diagramm nach Chen</b> .....	<b>14</b>
<b>Netzwerksymbole</b> .....	<b>15</b>
<b>ISO-OSI-7-Schichtenmodell</b> .....	<b>15</b>
Header.....	16
Ethernet II .....	16
TCP –Header .....	16
<b>Assembler</b> .....	<b>17</b>
<b>Die Programmiersprache C (ANSI-C)</b> .....	<b>17</b>
1. Datentypen, Variable, Konstante, Operatoren und Ausdrücke .....	17
<b>2. Aufbau eines C-Programms</b> .....	<b>18</b>
<b>3 Befehle zur Steuerung des Programmflusses</b> .....	<b>19</b>
3.1 If , else.....	19
3.2 switch .....	19
4. Der break-Befehl .....	20

# Strukturierte Programmierung

## Zuweisung

Variable  $\leftarrow$  derAusdruck

Variable := derAusdruck

## Sequenz

Anweisung 1

Anweisung 2

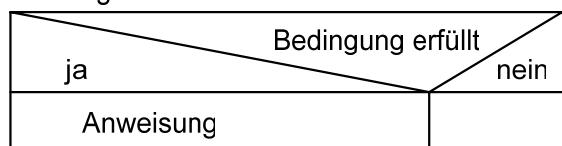
Anweisung 3

Unterprogrammaufruf

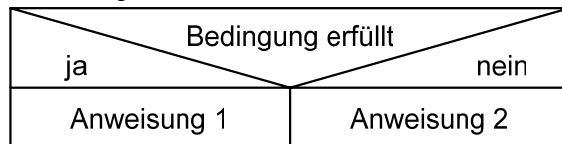
Unterprogramm - Name

## Auswahl

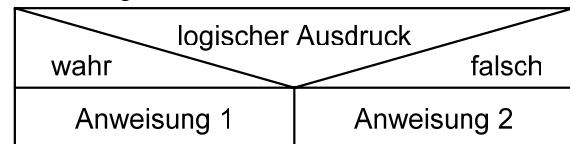
einseitige Auswahl



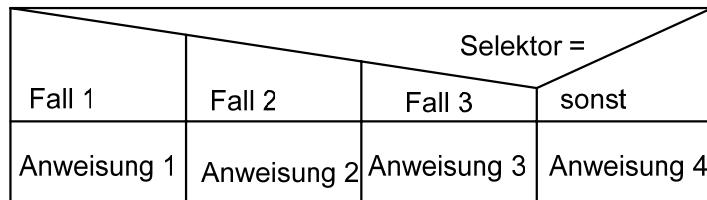
zweiseitige Auswahl



zweiseitige Auswahl



Mehrfachauswahl



## Wiederholung (Iteration)

Schleife mit Eintrittsbedingung

Bedingung

Anweisung

Schleife mit Austrittsbedingung

Anweisung

Bedingung

Schleife mit Abbruchmöglichkeit

Bedingung

...

logischer Ausdruck

falsch

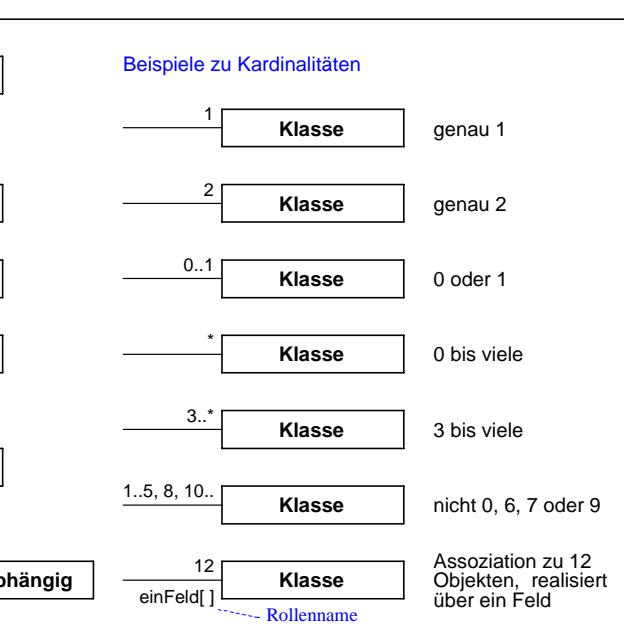
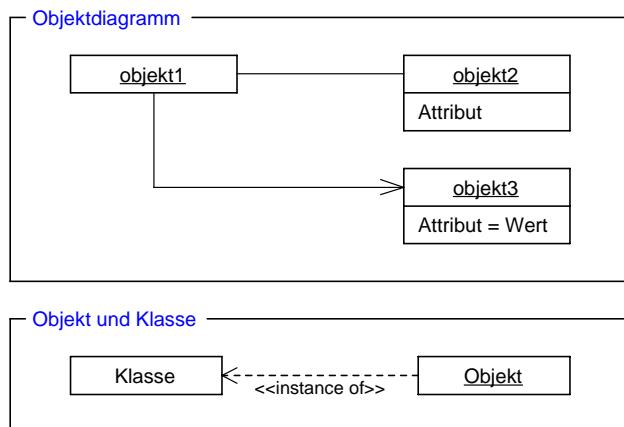
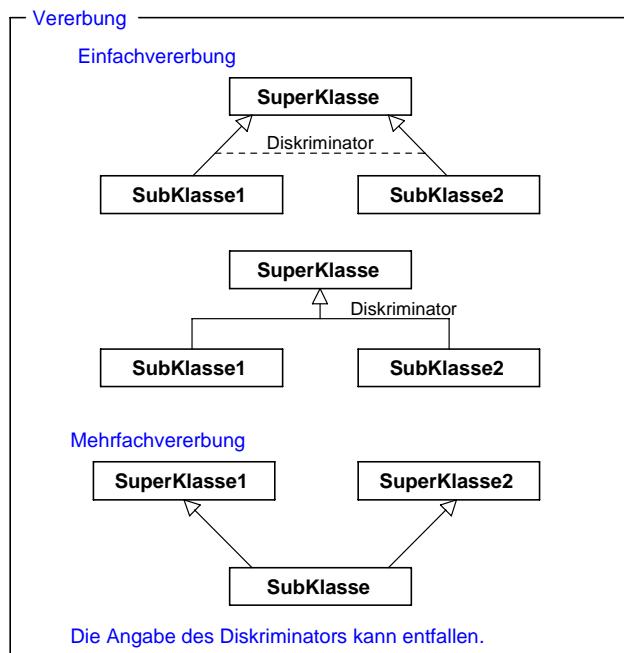
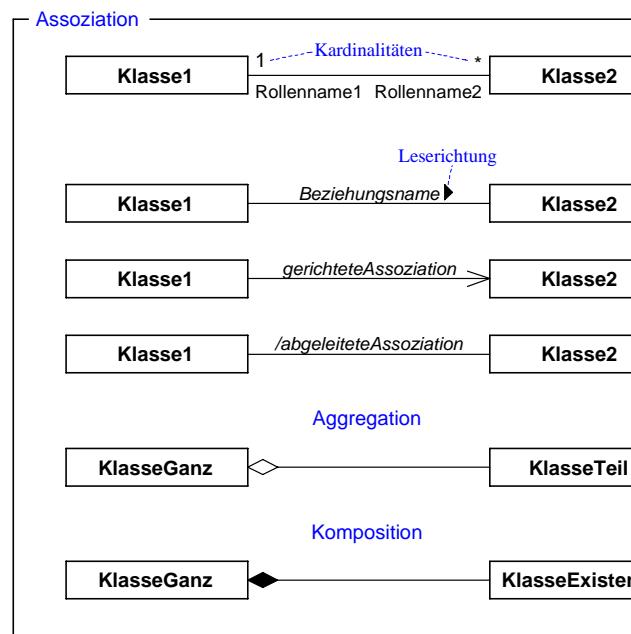
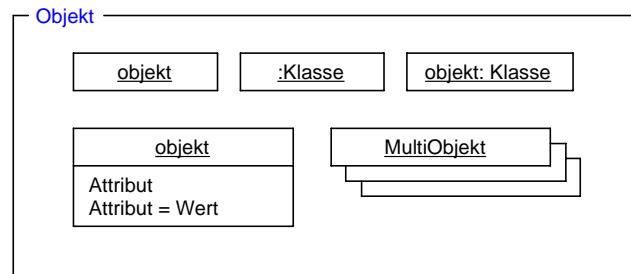
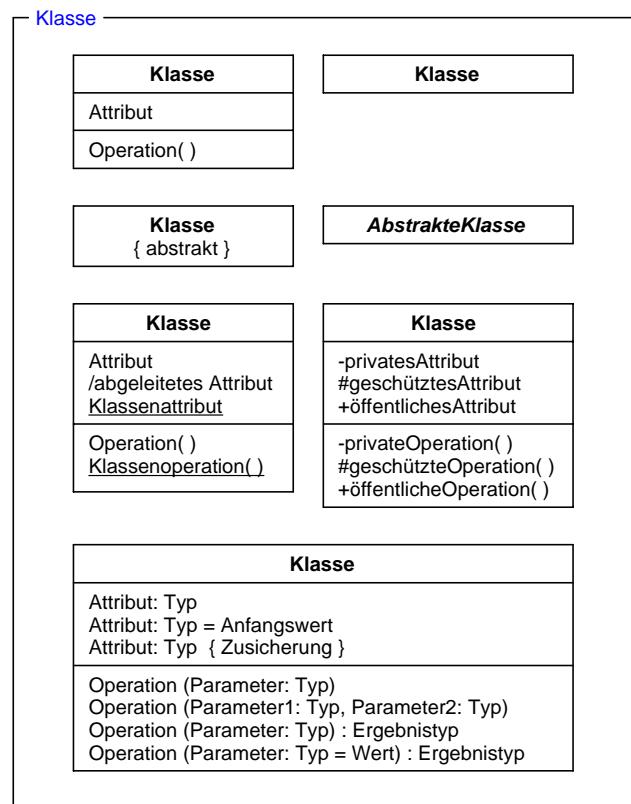
wahr

...

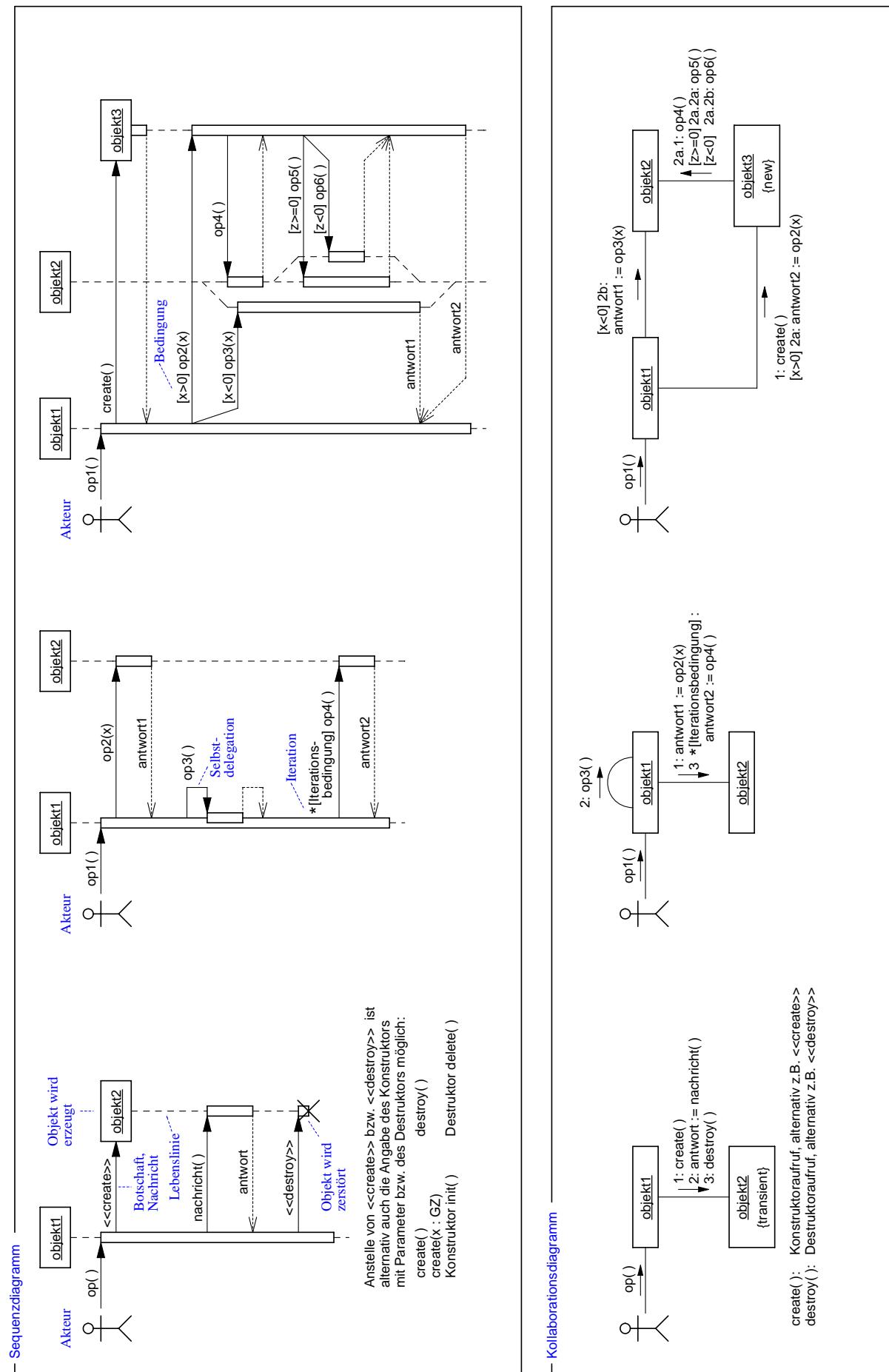
Schleife ohne Bedingungsprüfung (Endlosschleife)

Anweisung

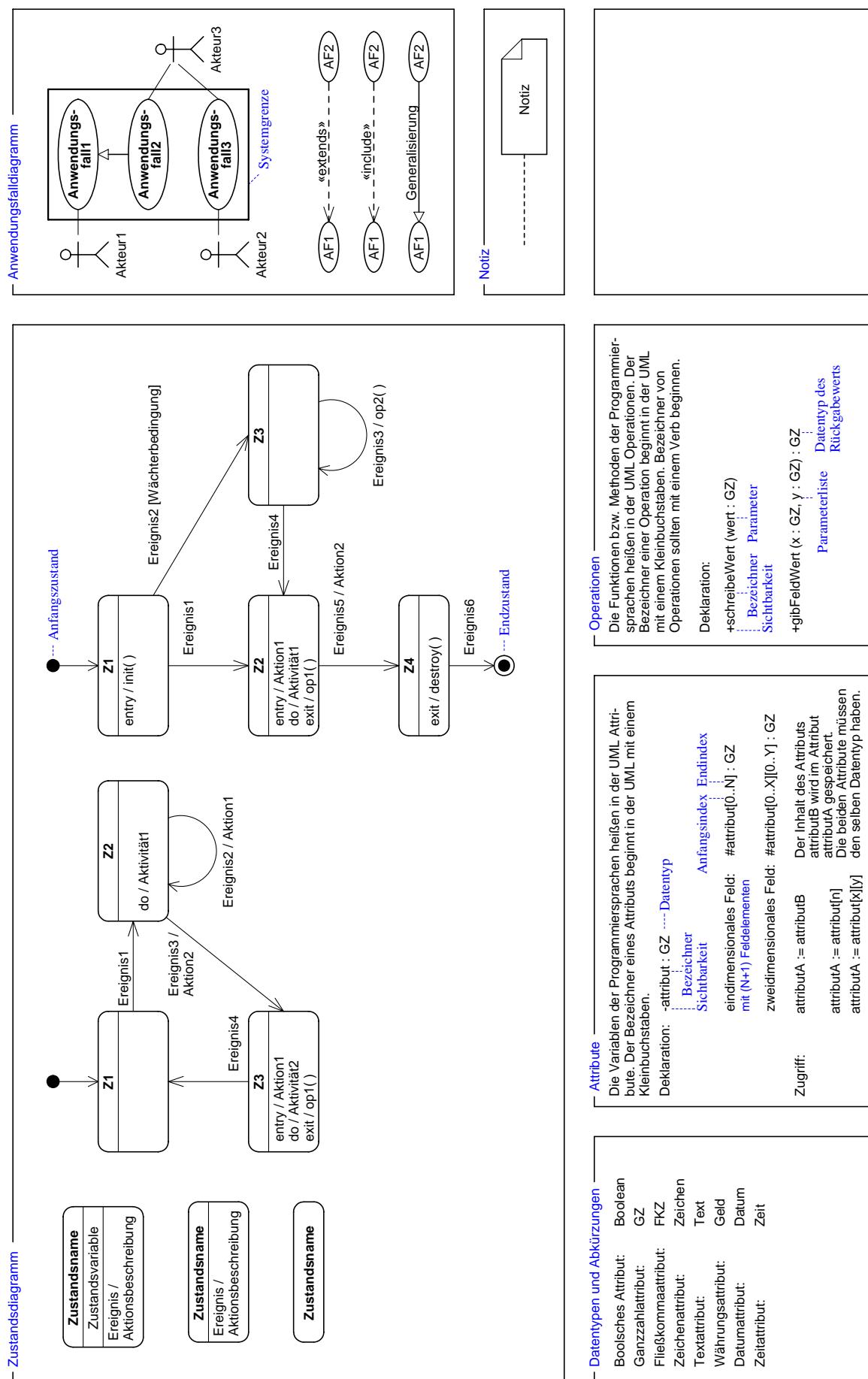
# Unified Modeling Language 1/3



UML 2/3

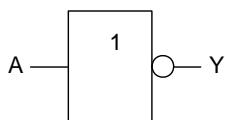


## UML 3/3



## Grundfunktionen (Logik)

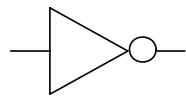
### NOT (Negation)



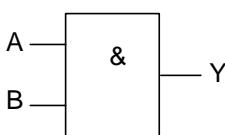
$$\begin{aligned} Y &= !A \\ Y &= \text{NOT } A \\ Y &= /A \\ Y &= \overline{A} \\ Y &= \neg A \end{aligned}$$

A	Y
0	1
1	0

auch zulässig:

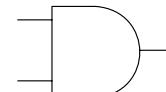


### AND (Konjunktion)

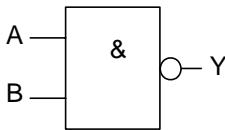


$$\begin{aligned} Y &= A \& B \\ Y &= A \text{ AND } B \\ Y &= A * B \\ Y &= A \wedge B \end{aligned}$$

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

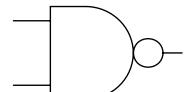


### NAND

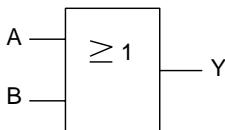


$$\begin{aligned} Y &= !(A \& B) \\ Y &= \text{NOT}(A \text{ AND } B) \\ Y &= /A * B \\ Y &= A \wedge B \end{aligned}$$

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

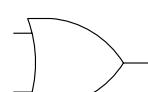


### OR (Disjunktion)

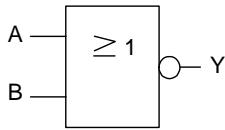


$$\begin{aligned} Y &= A \# B \\ Y &= A \text{ OR } B \\ Y &= A + B \text{ (nicht in ABEL)} \\ Y &= A \vee B \end{aligned}$$

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



### NOR

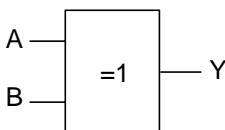


$$\begin{aligned} Y &= !(A \# B) \\ Y &= \text{NOT}(A \text{ AND } B) \\ Y &= /A + B \\ Y &= A \vee B \end{aligned}$$

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

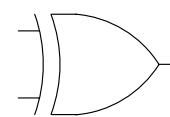


### XOR (Antivalenz)

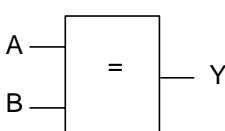


$$\begin{aligned} Y &= A \$ B \\ Y &= A \text{ XOR } B \\ Y &= A/B + /AB \\ Y &= A \oplus B \end{aligned}$$

B	A	Y
0	0	0
0	1	1
1	0	1
1	1	0

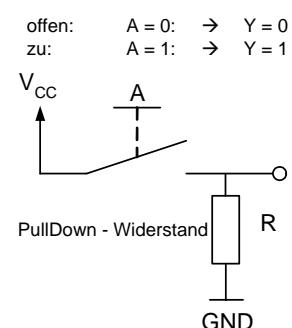
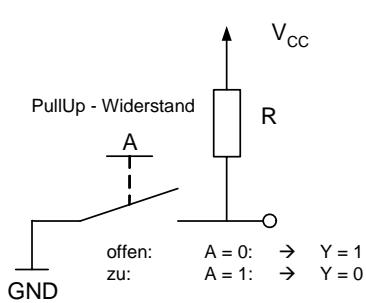
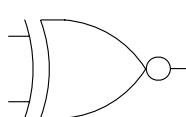


### XNOR (Äquivalenz)

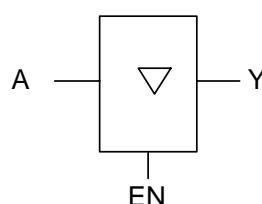


$$\begin{aligned} Y &= A !\$ B \\ Y &= A \text{ XNOR } B \\ Y &= AB + /A/B \\ Y &= \overline{A \oplus B} \end{aligned}$$

B	A	Y
0	0	1
0	1	0
1	0	0
1	1	1



### Tristate

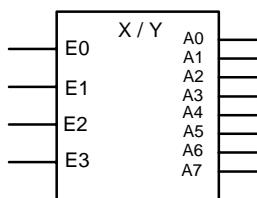


EN	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

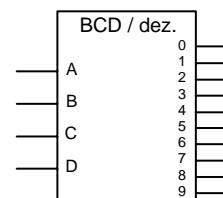
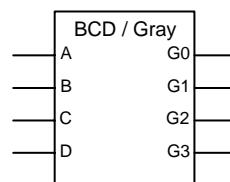
 $Z \hat{=} \text{ hochohmig}$

## typische Schaltnetze (Blockschaltbilder)

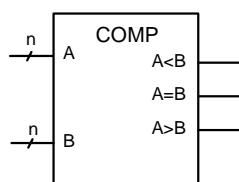
### Codeumsetzer (Umcodierer)



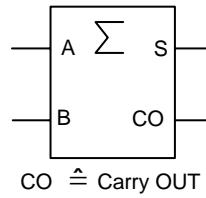
### Beispiele



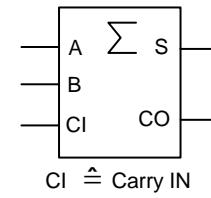
### Komparator



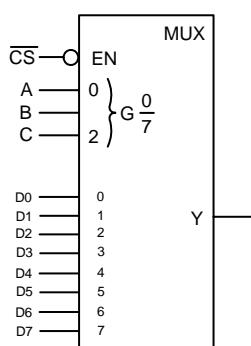
### Halbaddierer



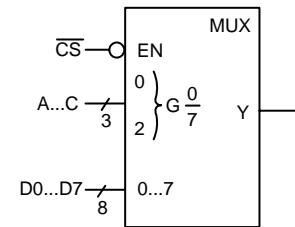
### Volladdierer



### MUX (8 zu 1)

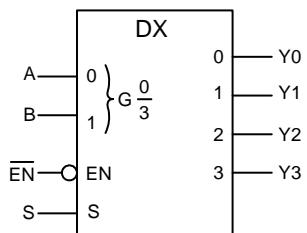


C	B	A	/CS	Y
X	X	X	1	0
0	0	0	0	D0
0	0	1	0	D1
0	1	0	0	D2
0	1	1	0	D3
1	0	0	0	D4
1	0	1	0	D5
1	1	0	0	D6
1	1	1	0	D7



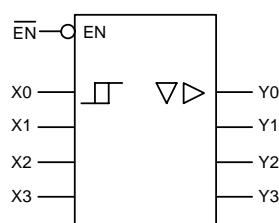
Adress- und Datenleitungen können auch zusammengefasst werden

### DX (2 zu 4) / Decodierer

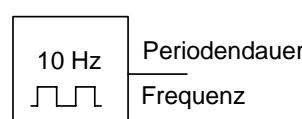


A	B	/EN	Y0	Y1	Y2	Y3
X	X	1	0	0	0	0
0	0	0	S	0	0	0
0	1	0	0	S	0	0
1	0	0	0	0	S	0
1	1	0	0	0	0	S

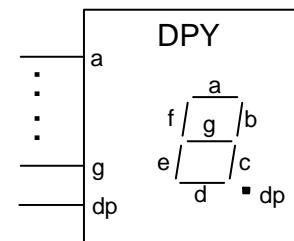
### Bustreiber



### Astabiles Element (allg. Taktgenerator)

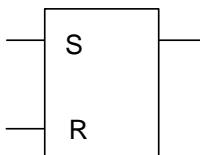


### 7-Segmentanzeige



## FlipFlop

### RS-FlipFlop (statisch)



### Zustandsfolgetabelle

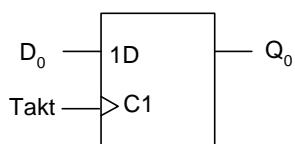
S	R	$Q^n$	$Q^{n+1}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	?	?

$$Q^{n+1} = (Q^n \& !R \# !R \& S)$$

$$Q^{n+1} = S \# !R \& Q^n \quad (\text{Minimalform})$$

in ABEL® steht kein zustandsgesteuertes SR-FF zur Verfügung

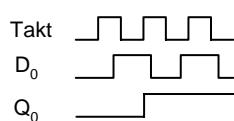
### D-Flip-Flop (dynamisch)



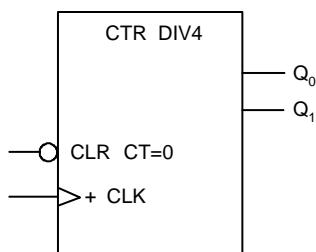
C	D	$Q^{n+1}$
1	X	$Q^n$
0	X	$Q^n$
pos	0	0
pos	1	1
neg	X	$Q^n$

pos  $\hat{=}$  positive Taktflanke  
neg  $\hat{=}$  negative Taktflanke

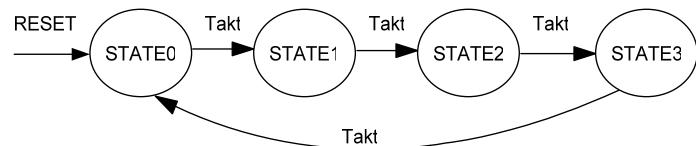
### Impulsdiagramm



### Zähler (Blockschaltbild)



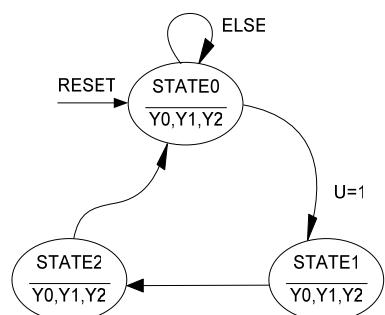
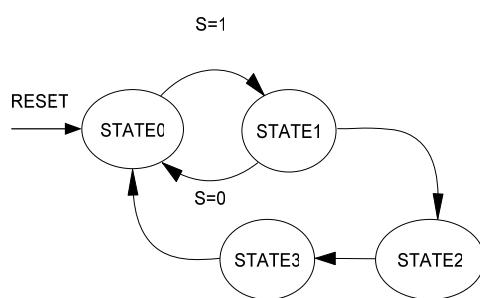
### Zustandsdiagramm



- wenn es sich offensichtlich um ein getaktetes Schaltwerk handelt, kann die Angabe des TAKTES als Übergangskriterium entfallen.
- Zusätzliche Übergangsbedingungen müssen angegeben werden

### Zustandskodierung

Zustand	Codierung $Q_1, Q_0$
STATE0	00
STATE1	01
STATE2	10
STATE3	11

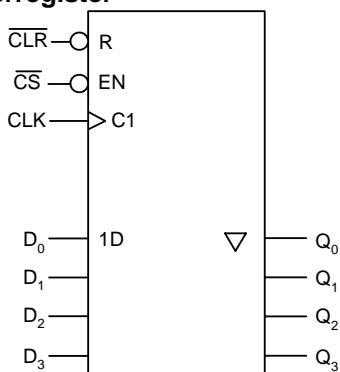


Codierte Zustandsübergangstabelle  
(Codierte Zustandsfolgetabelle)

n	n+1				
	U	Q1	Q0	Q1	Q0
0	0	0	0	0	0
1	0	0	0	0	1
x	0	1	1	1	0
x	1	0	0	0	0

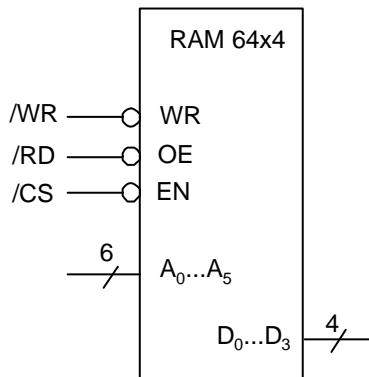
- die Angabe von Ausgabewerten in einem Zustand werden durch einen (Unter-)Strich vom Zustandsnamen getrennt (die „else“ - Angabe kann entfallen).

### Speicherregister



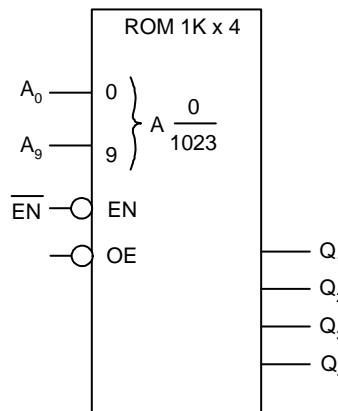
- 4-Bit Speicherregister (4 flankengesteuerte D-Flipflops)
- Parallele Ein- und Ausgabe
- Wenn der Baustein ausgewählt ist ( $EN = 0$ ), werden mit der ansteigenden Flanke des Takt-Signals die an den Eingängen  $D_0 \dots D_3$  anstehenden Daten übernommen.
- Mit einem 0-Signal am  $-CLR$ -Eingang kann das Register gelöscht werden.
- $EN$  ermöglicht, die Ausgänge in Tri-State zu schalten ( $EN=1$ ) oder den Speicherinhalt auszulesen ( $EN=0$ )

### RAM



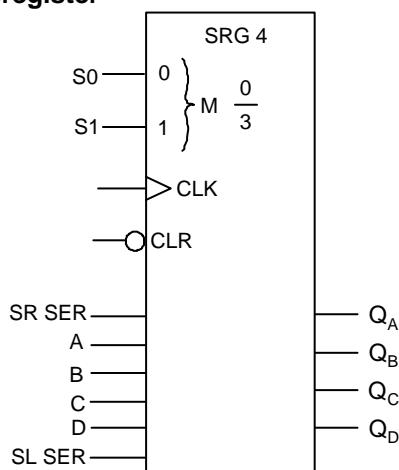
Schreib- Lesespeicher mit 64 x 4 Bit

### ROM



- Read Only Memory
- mit den Adressen 0 ... 1023
- einer Wortbreite von 4 Bit
- und 1 Freigabeeingang

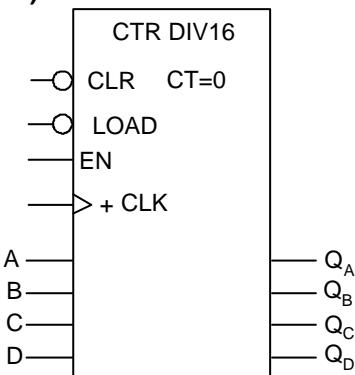
### Schieberegister



- 4-Bit Schieberegister
- Links- Rechtsbetrieb:

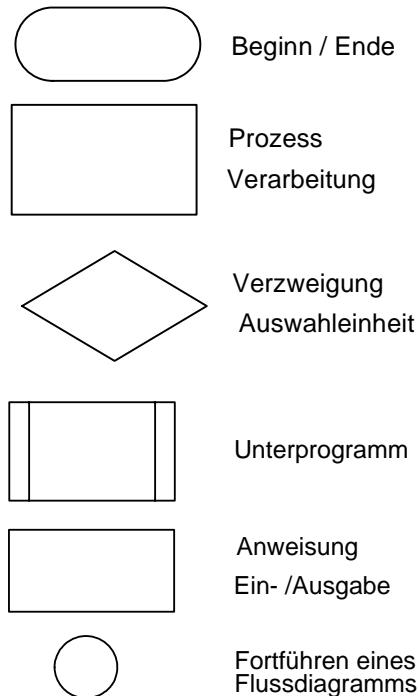
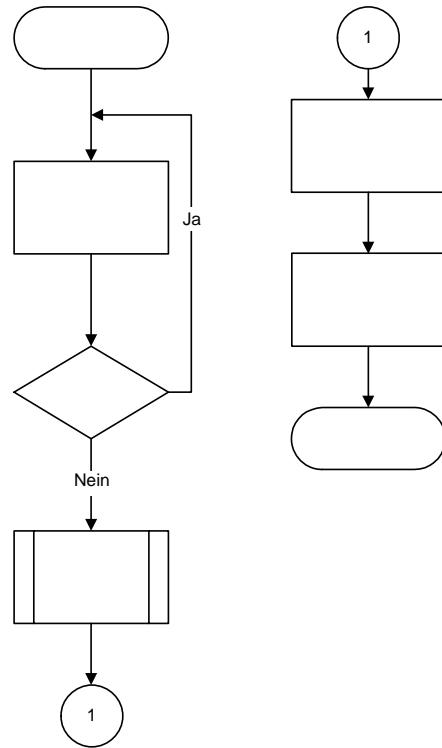
Mode	S1	S0	Funktion
0	0	0	-
1	0	1	rechts
2	1	0	links
3	1	1	parallele Eingabe

- mit serieller Eingabe
- paralleler Ausgabe
- Vorwärtschieben mit der positiven Taktflanke und der Möglichkeit, das gesamte Register zu löschen

**Zähler (4-Bit)**

- CTR  $\hat{=}$  Zähler
- DIV 16  $\hat{=}$  16 verschiedene binäre Zustände
- Vorwärtszähler (+)
- EN = 1 und die positive Taktflanke führen zum nächsten Zählzustand
- Mit /LOAD kann ein Anfangszustand geladen werden

	Ein- oder Ausgang mit Hysterese
	Treiberausgang
	Tri-State-Ausgang
	Negation am Eingang
	Negation am Ausgang
	Dynamischer Eingang aktiv bei positiver Flanke
	... aktiv bei negativer Flanke
	Vorwärtszähler
	Eingang, der Rechtsschieben bewirkt
	MUX / DX Multiplexer / Demultiplexer
	CTR <sub>m</sub> Zähler mit $m$ Bits / Zykluslänge = $2^m$
	CTR DIV <sub>m</sub> Zähler mit Zykluslänge $m$
	SRG <sub>m</sub> Schieberegister mit $m$ Bits

**Programmablaufplan (PAP)****Symbole****Beispiel**

## Abfrageformulierung mit SQL

## 1. Projektion und Formatierung

## Auswahl aller Spalten einer Tabelle

Syntax :      **SELECT**      \*

## Auswahl einer Spalte einer Tabelle

## Auswahl mehrerer Spalten einer Tabelle

Syntax :      **SELECT**      <Spalte1> , <Spalte2> , ...  
                  **FROM**      <Tabelle>

Hinweis: In SQL kann ein Abfrageergebnis mehrere identische Tupel enthalten.

## Auswahl ohne mehrfaches Auftreten desselben Tupels

Syntax :      **SELECT DISTINCT**      **<Spalte>**  
                  **FROM**                **<Tabelle>**

## Formatierte Ausgabe und Berechnungen in einer Selektion

Syntax      **SELECT**      "Die Veranstaltung dauert " <Spalte1>\*8 " Stunden"  
              **FROM**      <Tabelle>

Hinweis: Alle mathematischen Rechenzeichen sind verwendbar.

## Umbenennen von Spalten

Syntax : **SELECT** <Spalte> AS <neuer Spaltenname>  
**FROM** <Tabelle>

## Sortierung

Syntax :      **SELECT**      **<Spalte>**  
                 **FROM**      **<Tabelle>**  
                 **ORDER BY**    **<Spalte> {DESC | ASC}**

## 2. Selektion

Syntax : **SELECT** <Spalte>  
**FROM** <Tabelle>  
**WHERE** <Bedingung>  
WHERE Klausel definiert die auszuwählenden Zeilen, Vergleichsoperatoren sind = , <> , > , < , >= , <=

## Selektion mit mehreren Bedingungen

## Selektion mit dem Operator IN

Syntax	SELECT	<Spalte>
	FROM	<Tabelle>
	WHERE	<Spalte> (NOT)IN ('Wert1', 'Wert2',.....)

## Selektion mit dem Operator BETWEEN

Syntax	SELECT FROM WHERE	<Spalte> <Tabelle> Spalte BETWEEN 'Wert1' AND 'Wert2'
--------	-------------------------	---

## Selektion mit dem Operator LIKE

Der LIKE-Operator ermöglicht den Vergleich eines Strings mit einem Muster. Muster werden aus beliebigen Zeichen eines Strings und den beiden Sonderzeichen '?' und '\*' gebildet. '?' steht für genau ein beliebiges Zeichen, während '\*' für eine beliebig große (evtl. leere) Kette von beliebigen Zeichen steht. Achtung: In ANSI-SQL: '?' = '\_' und '\*' = '%'.

```
SELECT      <Spalte>
FROM        <Tabelle>
WHERE       <Spalte> LIKE '?????3*' -
```

## Selektion und NULL-Werte

NULL wird i.a. interpretiert als ein Platzhalter für die Aussage "Information/Attribut ist nicht vorhanden oder nicht bekannt oder nicht anwendbar".

Syntax:      **SELECT**      **<Spalte>**  
                 **FROM**      **<Tabelle>**  
                 **WHERE**      **<Spalte> IS [NOT] NULL**

### 3. Verbund von Tabellen

## Einfacher Equijoin mit zwei Tabellen (Natural Join)

Syntax :      **SELECT**      **<Spalte1> , <Spalte2>, ...**  
                  **FROM**      **<Tabelle1> , <Tabelle2>**  
                  **WHERE**      **<Join-Bedingung>**

Hinweis : Wenn die Tabellen, die miteinander zu verbinden sind, Spalten mit gleichem Spaltennamen aufweisen, dann muß jeweils spezifiziert werden, welche Spalte welcher Tabelle gemeint ist.

Beispiel : Zur Verkürzung des Anfragetextes können für die Tabellen in der FROM-Komponente auch Alias-Namen vergeben werden.

Beispiel :      **SELECT**                      <Spalte1> , <Spalte2>, ...  
                    **FROM**                              <Tabelle1> T1 , <Tabelle2> T2  
                    **WHERE**                              T1.ID = T2.ID

Hinweis : Die Alias-Namen können bereits in der SELECT-Komponente verwendet werden, auch wenn sie erst in der FROM-Komponente definiert werden.

## Einfacher Equijoin über $n > 2$ Tabellen

```
SELECT      <Spalte1> , <Spalte2>, ...
FROM        <Tabelle1> T1 , <Tabelle2> T2,<Tabelle3> T3
WHERE       T1.ID = T2.ID AND
           T2.ID = T3.ID
```

## INNER Join mit zwei Tabellen

## **Vereinigung mit UNION**

Die Datensätze von Tabellen, die identische Spalten enthalten, können durch UNION [ALL] zusammengefasst werden. Spaltentypen und –größe müssen übereinstimmen. Mehrfache Datensätze werden automatisch entfernt („DISTINCT“ ist Standardeinstellung). Dies kann umgangen werden mit UNION ALL. Beispiel :      Gewünscht wird eine Tabelle mit den Ortsnummern Ort\_Nr. der Orte, an denen

das Modul 4.1 Voraussetzung ist oder deren Ort Freiburg oder Ulm ist.

```

SELECT      <Spalte>
FROM        <Tabelle1>
WHERE       .....
UNION
SELECT      <Spalte>
FROM        <Tabelle2>
WHERE       .....

```

## 4. Aggregatfunktionen und Gruppen

Hinweis: NULL-Werte werden vor der Auswertung einer Aggregatfunktion eliminiert.

### Zählfunktion

Syntax :	SELECT	COUNT ([DISTINCT] <Spaltenliste * >)
	FROM	<Tabelle>
Anm.: COUNT(DISTINCT) funktioniert nicht mit JET SQL		

### Arithmetische Funktionen

Syntax :	SELECT	SUM ({numerische Spalte   Arithmetischer Ausdruck mit numerischen Spalten})
	FROM	<Tabelle>
Syntax :	SELECT	AVG ({numerische Spalte   Arithmetischer Ausdruck mit numerischen Spalten })
	FROM	<Tabelle>

### Min-/Max-Funktionen

Syntax :	SELECT	MAX ({numerische Spalte   Arithmetischer Ausdruck mit numerischen Spalten})
	FROM	<Tabelle>
Syntax :	SELECT	MIN ({numerische Spalte   Arithmetischer Ausdruck mit numerischen Spalten})
	FROM	<Tabelle>

## Gruppenbildung in SQL-Anfragen

In den vorangegangenen Beispielen wurden die Aggregatfunktionen immer auf eine ganze Tabelle angewandt. Daher bestand das Abfrageergebnis immer nur aus einem Tupel. In SQL ist es aber auch möglich, eine Tabelle zu gruppieren, d.h. die Tupel einer Tabelle in Gruppen einzuteilen, und dann die Aggregatfunktionen jeweils auf die Gruppen anzuwenden.

Syntax : 

SELECT	<Spalte> , <Aggregatfunktion (<Spalte>)...>
FROM	<Tabelle>
GROUP BY	<Spalte>

Hinweis : Die in der GROUP BY-Komponente spezifizierten Spalten müssen auch in der SELECT-Komponente spezifiziert sein, da Basis für die Gruppierung die "Zwischen-Ergebnis"-Tabelle ist, die durch Select ... From spezifiziert wurde. Andererseits müssen alle Spalten der Selektionsliste, die nicht durch eine Aggregatfunktion gebunden sind, in der group by-Komponente aufgeführt werden. Daraus ergibt sich eine gewisse Redundanz im Abfragecode.  
Die Reihenfolge der Spaltenspezifikation in der GROUP BY-Komponente hat keinen Einfluß auf das Resultat der Abfrage.  
Einschränkungen sind nicht mit WHERE möglich, sondern mit HAVING!

## Auswahl von Gruppen

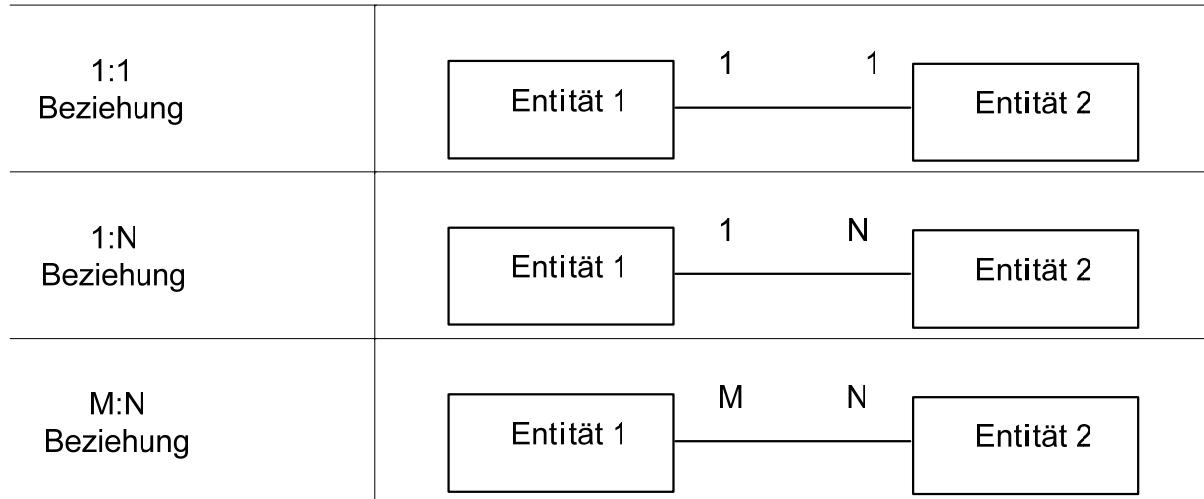
Syntax : 

SELECT	<Spalte> , <Aggregatfunktion ...>
--------	-----------------------------------

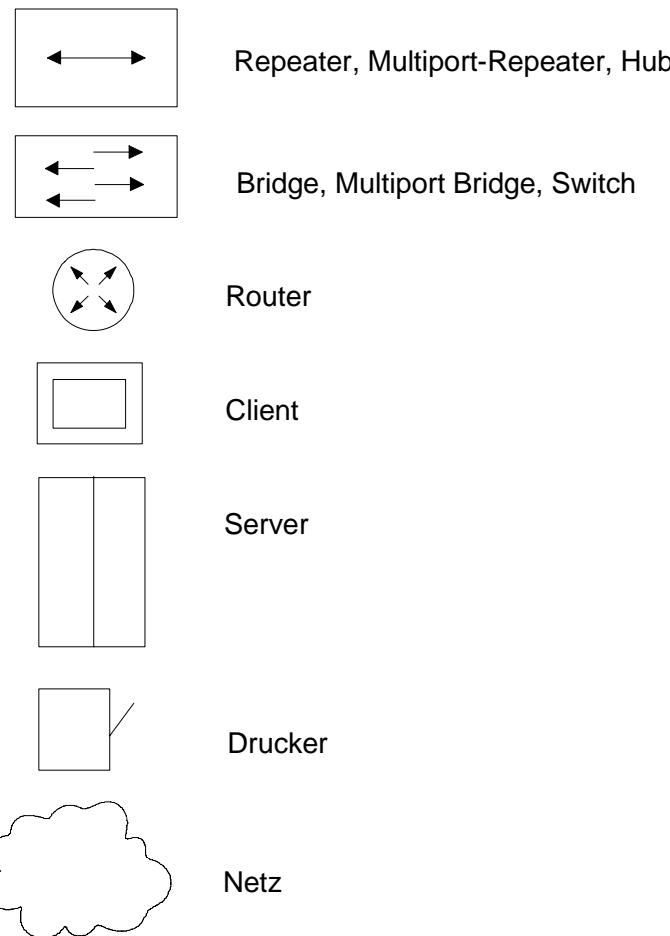
FROM <Tabelle>  
GROUP BY <Spalte>  
HAVING <Bedingung>

## ER-Diagramm nach Chen

ER - Diagramm nach Chen



## Netzwerksymbole



## ISO-OSI-7-Schichtenmodell

Application Layer	7	Anwendungs-Schicht
Presentation Layer	6	Darstellungs-Schicht
Session Layer	5	Sitzungs-Schicht
Transport Layer	4	Transport-Schicht
Network Layer	3	Netzwerk-Schicht
Data Link Layer	2	Verbindungs-Schicht
Physical Layer	1	Physikalische-Schicht

## Header

### Ethernet II

Präambel	Zieladresse	Absenderadresse	Typ	Daten	Link Trailer
8	6	6	2	46...1500	4

### IP-Header

Bit	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Byte 0		Version					IHL	
Byte 1				TOS				
Byte 2					Paketlänge			
Byte 3								
Byte 4				Identifikation				
Byte 5								
Byte 6		Flags			Fragmentabstand			
Byte 7				Fragmentabstand				
Byte 8				Time To Live (TTL)				
Byte 9					Protokoll			
Byte 10						Kopf-Prüfsumme		
Byte 11								
Byte 12								
Byte 13				IP-Sendeadresse				
Byte 14								
Byte 15								
Byte 16								
Byte 17				IP-Empfängeradresse				
Byte 18								
Byte 19								
Byte 20 ...				Optionen (mit evtl. Füllzeichen)				

### TCP –Header

Bit	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Byte 0				Source Port				
Byte 1								
Byte 2				Destination Port				
Byte 3								
Byte 4					Sequenznummer			
Byte 5								
Byte 6								
Byte 7								
Byte 8								
Byte 9				Quittungsfeld (Piggyback, Acknowledgement Number)				
Byte 10								
Byte 11								
Byte 12			Header-Länge			reserviert		
Byte 13		reserviert	URG	ACK	PSH	RST	SYN	FIN
Byte 14								
Byte 15				Fenster Größe				
Byte 16								
Byte 17				Prüfsumme				
Byte 18					Urgent Zeiger			
Byte 19								
Byte 20 ...				Optionen (evtl. mit Füllzeichen)				

## Assembler

- Assemblerdirektiven und Maschinenbefehle sind in einem **lokalen Teil** der Formelsammlung zu finden.
- Der lokale Teil wird von den Schulen entsprechend dem eingesetzten Mikrocontroller zur Verfügung gestellt.
- Für den MC 8051 wurde ein solcher gemeinsamer Anhang (lokaler Teil) entwickelt und steht allen Schulen zur Verfügung.

## Die Programmiersprache C (ANSI-C)

### 1. Datentypen, Variable, Konstante, Operatoren und Ausdrücke

#### Datentypen

Datentyp	Größe	Wertebereich
bit	1 Bit	0 oder 1
signed char	1 Byte	-128 bis +127
unsigned char	1 Byte	0 bis 255
signed int	2 Byte	-32768 bis +32767
unsigned int	2 Byte	0 bis 65535
signed long	4 Byte	-2147483648 bis +2147483647
unsigned long	4 Byte	0 bis 4294967295
float	4 Byte	$\pm 1,176 \cdot 10^{-38}$ bis $\pm 3,40 \cdot 10^{38}$
pointer	1-3 Byte	Adresse einer Variablen
FILE		Dateizeiger

#### Operatoren und ihre Priorität

Mathematische Operatoren		Priorität	Verhältnis und logische Operatoren	
++ --	Inkrement dekrement	Höchste	!	NOT
-	monadisches Minus (Vorzeichen)		> >= < =<	Größer größer gleich kleiner kleiner gleich
*	Mal		==	Gleich
/	Div		!=	ungleich
%	mod		&&	AND
+	Plus			OR
-	minus	niedrigste		
Bitweise Operatoren		Beispiele		
&	UND	$X = 10:$		
	ODER	$Y = ++X \rightarrow Y = 11 ; Y = X++ \rightarrow Y = 10$		
^	EXOR	$Y = --X \rightarrow Y = 9 ; Y = X-- \rightarrow Y = 10$		
~	Einerkomplement	$Y = Y >> 1$ schiebe Y um 1 nach <b>rechts</b>		
<<	schieben nach <b>links</b> ;			
>>	.... - nach <b>rechts</b>			

## 2. Aufbau eines C-Programms

C bietet zwei Möglichkeiten zur Kommentareingabe :

- a) **//** Kommentar für eine Zeile
- b) **/\*** Kommentar für einen Block von einer oder mehreren Zeilen **\*/**

**{** Anfang eines zusammengehörigen Befehlsblocks (begin)  
**}** Ende eines zusammengehörigen Befehlsblocks (end)

// **INCLUDE** Files: → Compileranweisung über zusätzliche Quellcodes mit Funktionen und Deklarationen

```
#include <reg51xx.h>      // Registerdeklaration zum 5051xx
#include <math.h>          // Einbinden mathematischer Funktionen
```

// **Konstantendeklaration** → Ablage im Programmspeicher → Wert im Programm nicht änderbar

```
#define ANZAHL 10      // ANZAHL entspricht 10
#define TRUE 1          // TRUE entspricht 1
```

// Deklaration globaler **Variablen** → Ablage im Datenspeicher → Wert im Programm änderbar

```
int i = 8, j = 3, k;      /* Zählvariablen i , j und k mit den Anfangswerten 8 für i und 3 für j */
char TASTE;               // 1 Byte große Variable von 0-255
signed long 4BYTE;        // 4 Byte große Variable von 0 – 232
char code *text = "Hallo"; /* Textstring mit 5 Bytes (+ 0 als Stringende) im Programmspeicher*/
char bdata schieb;         // 1 Byte-Variable im bitadressierbaren Speicherbereich
```

// Deklaration von **Funktionen**

```
Typ Funktion_1(Typ Parameter1, Typ Parameter2 )
{
    // Begin von Funktion_1
    // lokale Variablen Deklaration;
    // Befehlsfolge;
}
```

```
Typ Funktion_2( )
{
    // Begin von Funktion_2
    // lokale Variablen Deklaration;
    // Befehlsfolge;
}
```

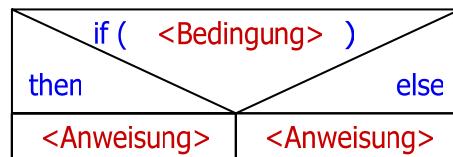
// Hauptprogramm - auch Hauptfunktion main()

```
main()
{
    // Begin des Hauptprogramms
    // lokale Variablen Deklaration;
    // Befehlsfolgen;
}
```

### 3 Befehle zur Steuerung des Programmflusses

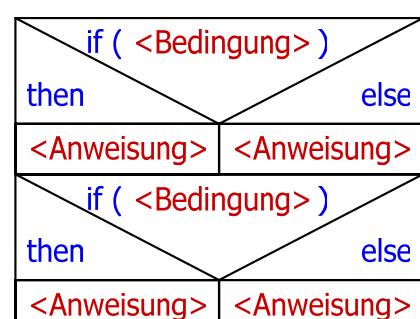
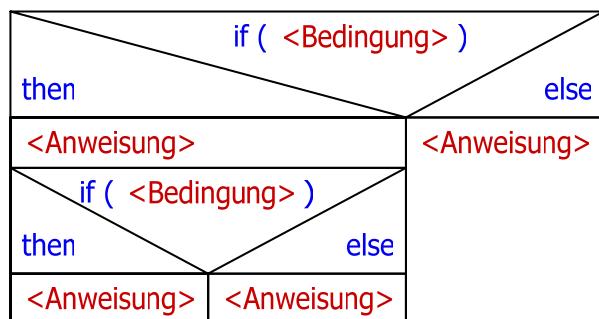
#### 3.1 If , else

```
if (Bedingung)
{
  Befehlsfolge für wahre Bedingung ;
}
else
{
  Befehlsfolge für falsche Bedingung;
}
```



Anmerkungen:

- Die Befehlsfolgen selbst können wiederum If-Anweisungen sein oder verschachtelte if-Anweisung
- Die else- Anweisung ist nicht zwingend notwendig



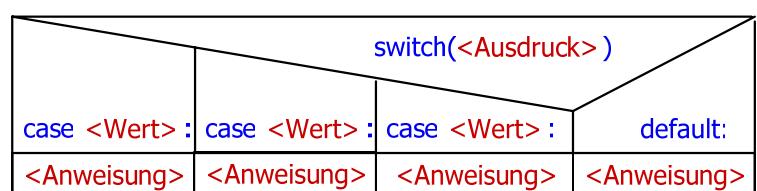
#### 3.2 switch

```
switch (Ausdruck)
{
  case Konstante 1:
    Befehlsfolge 1;
    break;

  case Konstante 2:
    Befehlsfolge 2;
    break;

  case Konstante X:
    Befehlsfolge X;
    break;

  default: {
    Befehlsfolge;
  }
}
```



Anmerkungen:

- Die Befehlsfolgen selbst können wiederum If-Anweisungen oder switch-Anweisungen sein.
- Die default-Anweisung ist nicht zwingend notwendig.
- Der break-Befehl bricht die switch-Anweisung ab.

## 3.2 Schleifen

### 3.2.1 Die for-Schleife

```
for ( Initialisierung ; Bedingung ; Veränderung )
{
  Befehlsfolge;
}
```



#### Anmerkungen:

- geeignet für Schleifen, bei denen die Anzahl der Durchläufe bekannt ist.
- Der Körper der Schleife kann auch leer sein (for ( ; ; )) jedoch die Semikolon müssen bleiben.
- Die Initialisierung legt die Startwerte der Variablen fest. Es können dabei auch mehrere Variablen mit Komma getrennt initialisiert werden.
- Ist die Bedingung erfüllt (TRUE) wird die Befehlsfolge bearbeitet
- Nach der Befehlsfolge bestimmt die Veränderung, wie die Variablen verändert werden.

**Beispiel:**

```
for ( i = 0, j = 8 , z = 0 ; i < j ; i++ , j - = 2 )
{
  z = i + j;
} // z = 6
```

### 3.2.2 Die while-Schleife (Kopfgesteuert)

```
while ( Bedingung )
{
  Befehlsfolgen;
}
```



#### Anmerkungen:

- Ist die Bedingung nicht erfüllt, also FALSE, dann wird die Befehlsfolge nicht bearbeitet
- Die Befehlsfolgen werden solange wiederholt, solange die Bedingung erfüllt bzw. WAHR ist
- **Endlosschleife mit while(1)** oder while(TRUE)
- Controller-Programme werden normalerweise mit einem Hardwarereset abgebrochen. Daher fangen die Programme für den µC meistens mit **while(1) {** an

### 3.2.3 Die do-while-Schleife (Fußgesteuert)

```
do {
  Befehlsfolgen;
} while ( Bedingung );
```



#### Anmerkungen:

- Die Befehlsfolge wird mindestens einmal bearbeitet, auch wenn die Bedingung nicht erfüllt ist.
- Die Befehlsfolgen werden solange wiederholt, wie die Bedingung erfüllt bzw. WAHR ist
- Endlosschleife mit while(1) oder while(TRUE)

## 4. Der break-Befehl

In einer Schleife beendet der break-Befehl diese, die Programmsteuerung geht direkt an die auf die Schleife folgenden Befehle über.

