

4. Rechnerarchitektur: Vom Transistor zum Programm

4.1 Transistoren, Gates

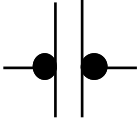
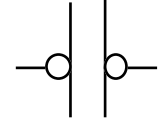
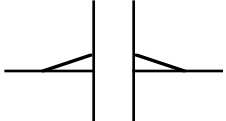
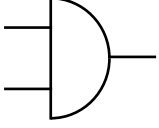
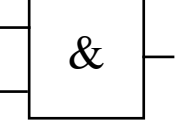
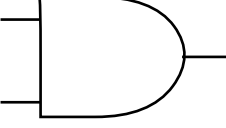
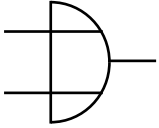
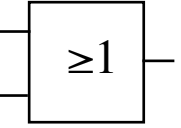
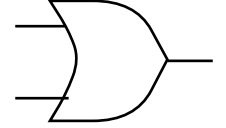
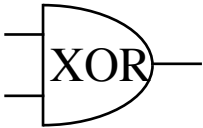
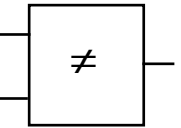
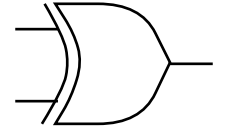
- Schaltfunktionen
 - 1..n Eingänge, 1..m Ausgänge
 - 2^n Argumentkombinationen
 - 2^{2^n} mögliche Funktionen

V_1	V_2	V_3	...	V_n	f_1	f_2	...	$f_{\max}$
1	1	1		1	0	0		1
0	1	1		1	0	0		1
0	0	1		1	0	0		1
0	0	0		0	0	1		1

- 2-stellige Schaltfunktion

		a=1, b=1	a=1, b=0	a=0, b=1	a=0, b=0
f_0		0	0	0	0
f_1		0	0	0	1
...		...	...	...	...
f_8	AND	1	0	0	0
f_{14}	OR	1	1	1	0
f_{15}		1	1	1	1

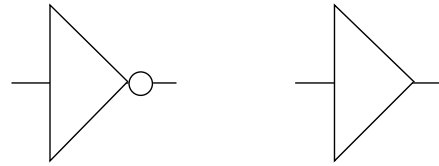
- Symbolische Darstellung von Schaltkreisen
 - Schaltplan
 - DIN 40700, IEEE/ANSI Y32.14, IEC

Funktion	Operator	graphisches Symbol		
		DIN	IEEE	IEC
Negation	$\neg, \bar{}$			
Und	$\cdot, \wedge, *$			
Oder	$+, \vee$			
Exklusiv Oder	$\oplus, \neq$			

- Allgemeine Schaltfunktionen:

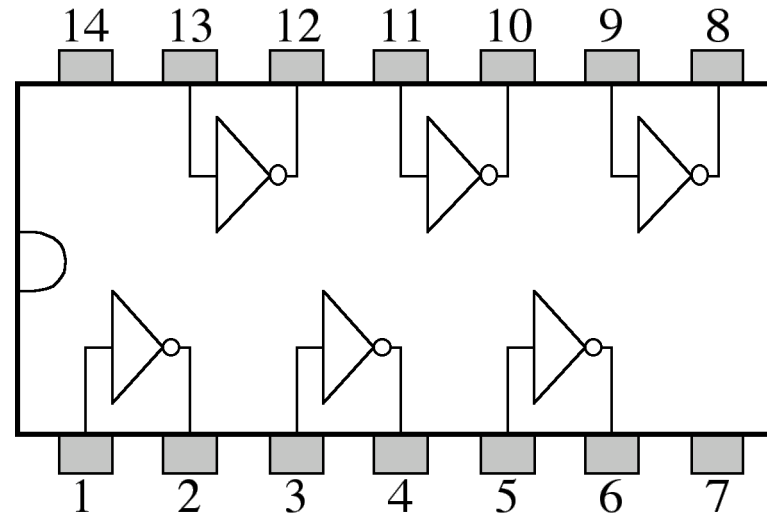


- Inverter, Verstärker:

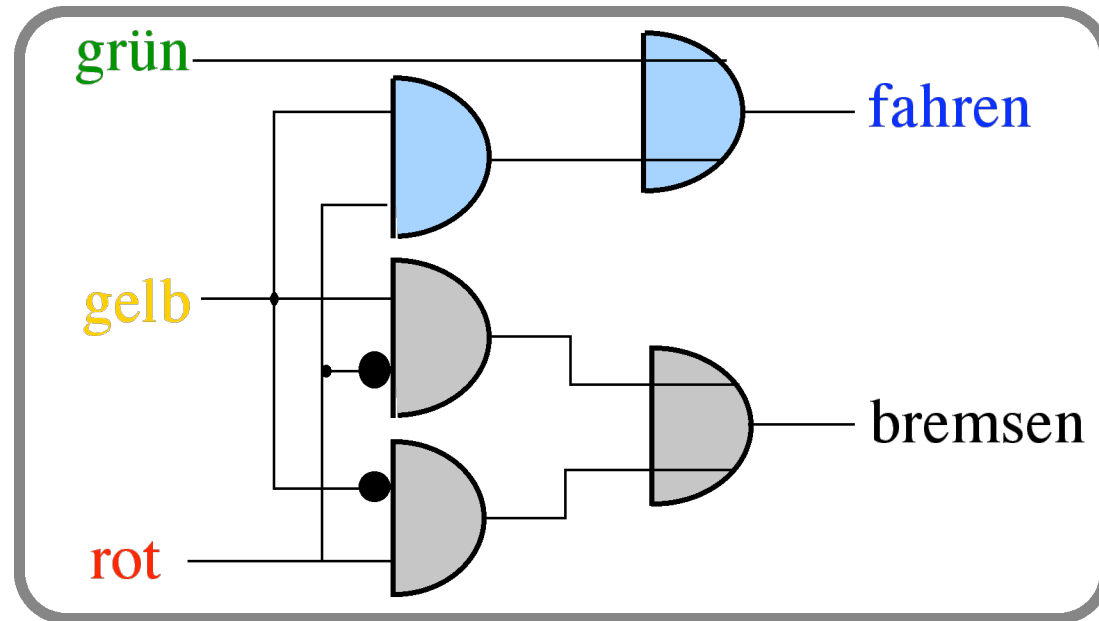


- Hex Inverter 7404 als Chip:

- Stift #7: 0 Volt, Erde, GND
- Stift #14: z.B. +5 Volt:



- Ampel-Reaktion



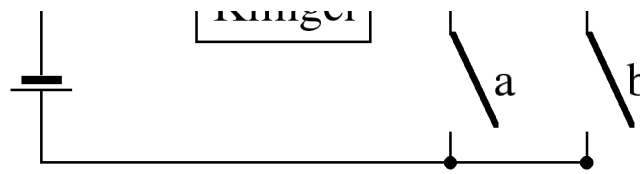
- Schaltalgebra

- Schaltfunktionen
- Rechenregeln
- Normalform
- Minimierung

- ODER (OR) Schaltung

- $a \cup b$

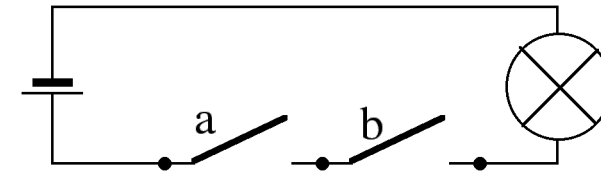
- Disjunktion, logische Summe, Vereinigung



- UND (AND) Schaltung

- $a \cap b$

- Konjunktion, logisches Produkt, Durchschnitt

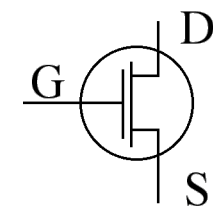


- Feldeffekttransistor (FET)

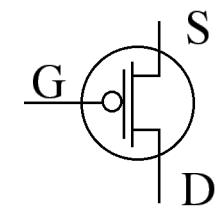
- Gate-Source-Spannung erzeugt Feld

- Feld kontrolliert Stromfluss im Drain-Source-Kanal

- U_{GS} steigt $\rightarrow I_{DS}$ steigt exponentiell

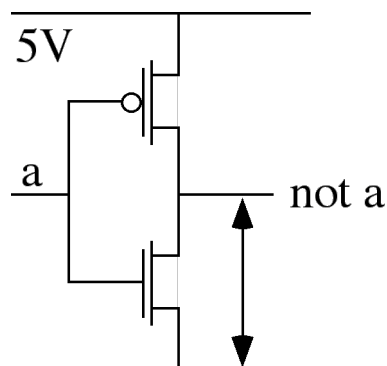


N-Kanal

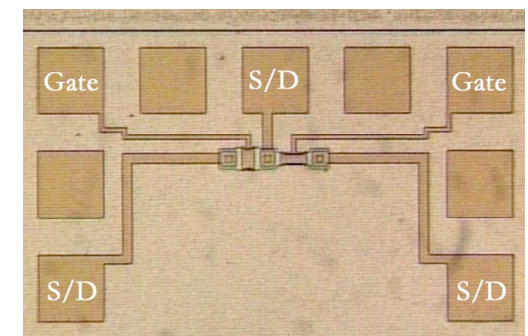
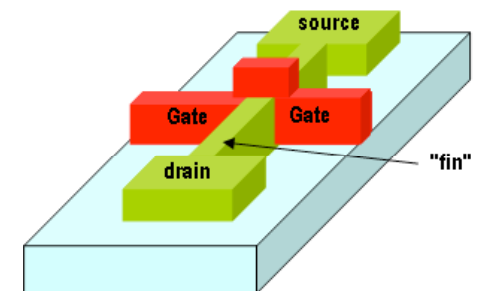
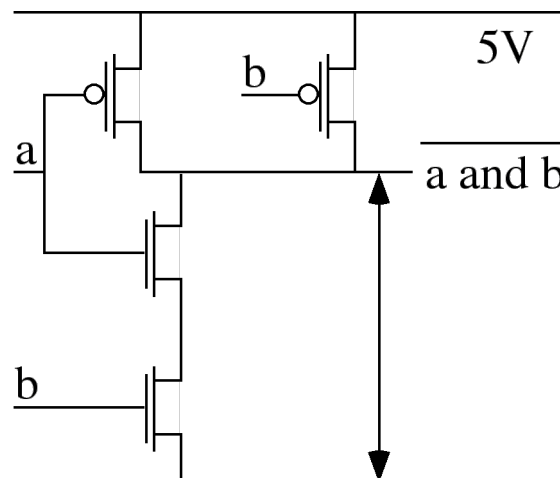


P-Kanal

- Negation (NOT)



- NAND



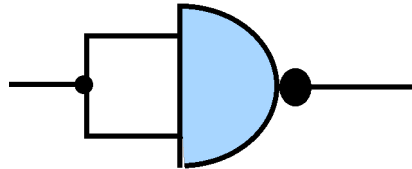
- Boolesche Algebra
- Kombination von NICHT mit UND / ODER

- $\overline{A \wedge B} = \overline{A} \vee \overline{B}$
- $A \vee B = \overline{\overline{A} \wedge \overline{B}}$
- $A \wedge B = \overline{\overline{A} \vee \overline{B}}$

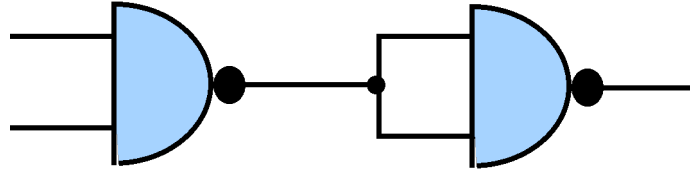
- NAND und NOR einfacher zu bauen

- CMOS: complementary Metal-Oxide-Silicon
- 4 Transistoren in CMOS-Technik

Inverter aus 1 NAND

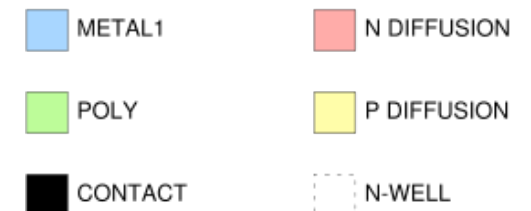
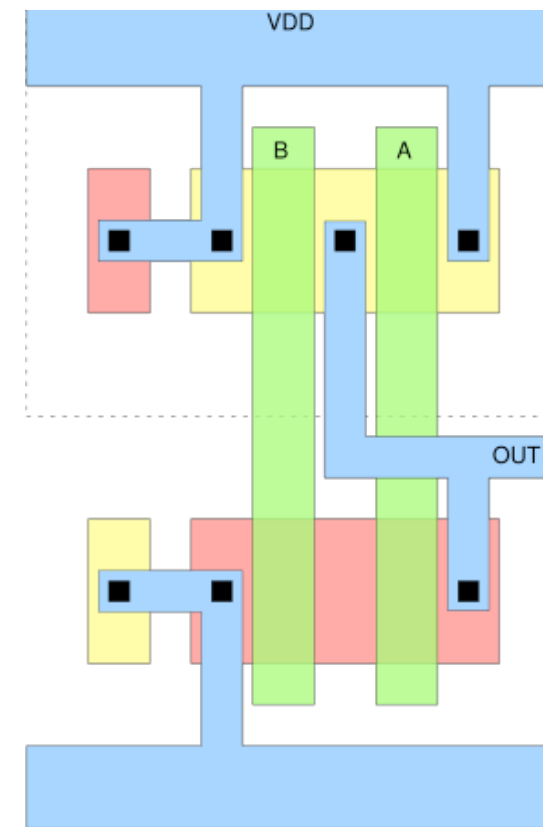
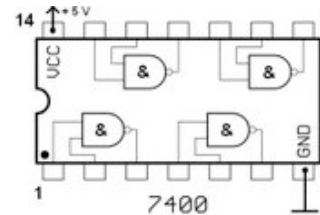


UND aus 2 NANDs



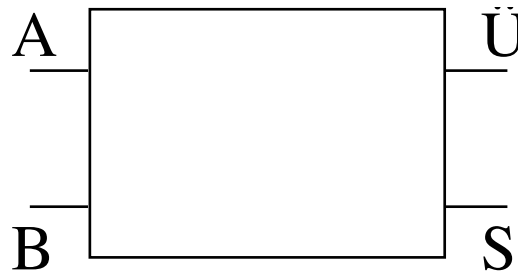
- 74xx, 74HCTxx, ...

- Einfache Logik-Chips
- 7404: 6 Inverter
- 7400: 4 NAND-Gatter
- 7402: 4 NOR-Gatter
- 7420: 2 NAND-Gatter mit je 4 Eingängen



4.2 Rechnen und Speichern

- Addition von zwei einstelligen Binärzahlen
 - Eingangsvariablen
 - Summenausgang
 - Übertrag zur nächsten Stufe

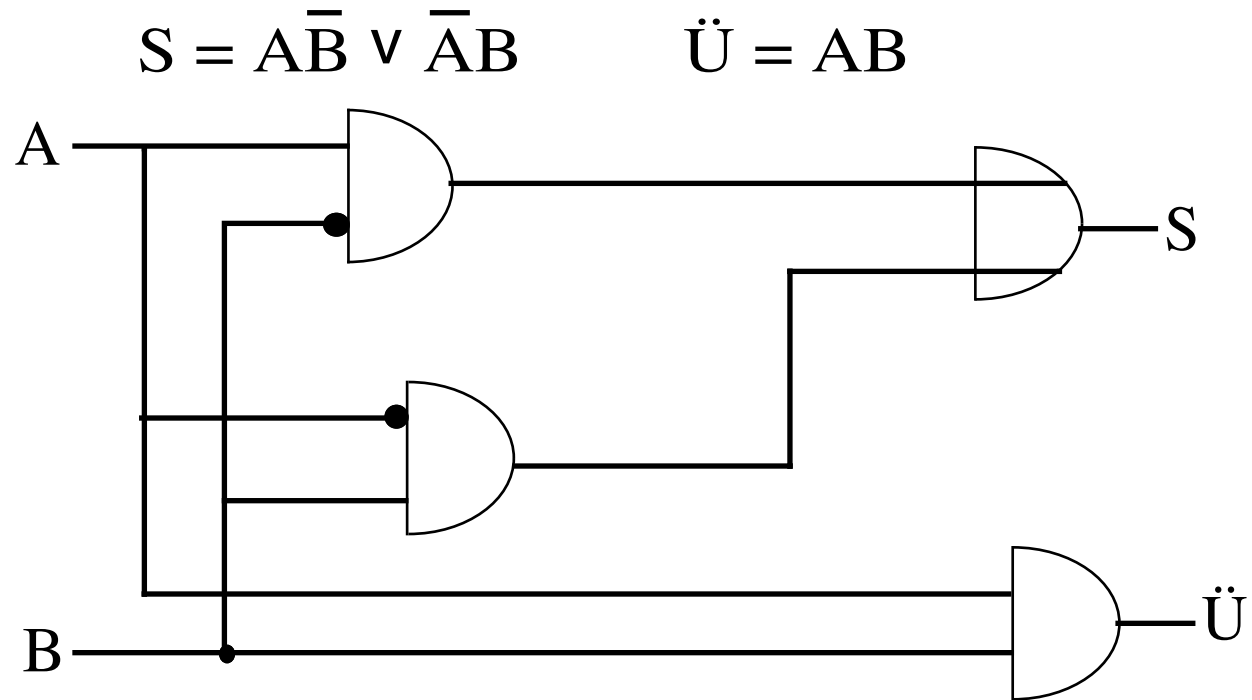


- Wahrheitstafel

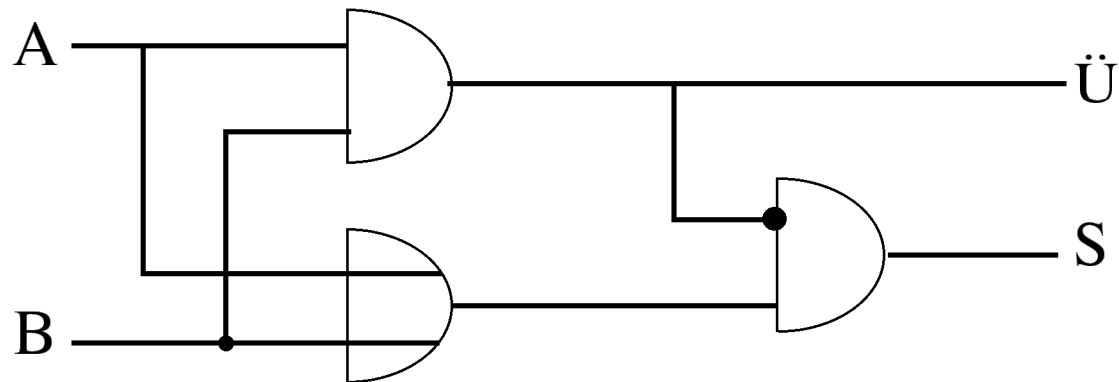
A	B	S	Ü
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Addierer kann aus sogenannten Halbaddieren zusammengesetzt werden
- Halbaddierer berücksichtigt nicht den Übertrag der früheren Stufe

- Halbaddierer

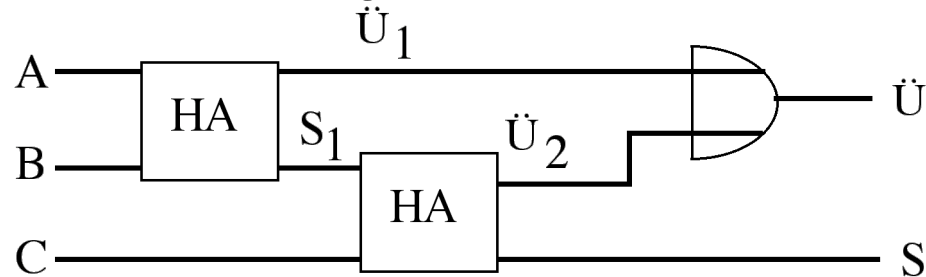


- Vereinfacht



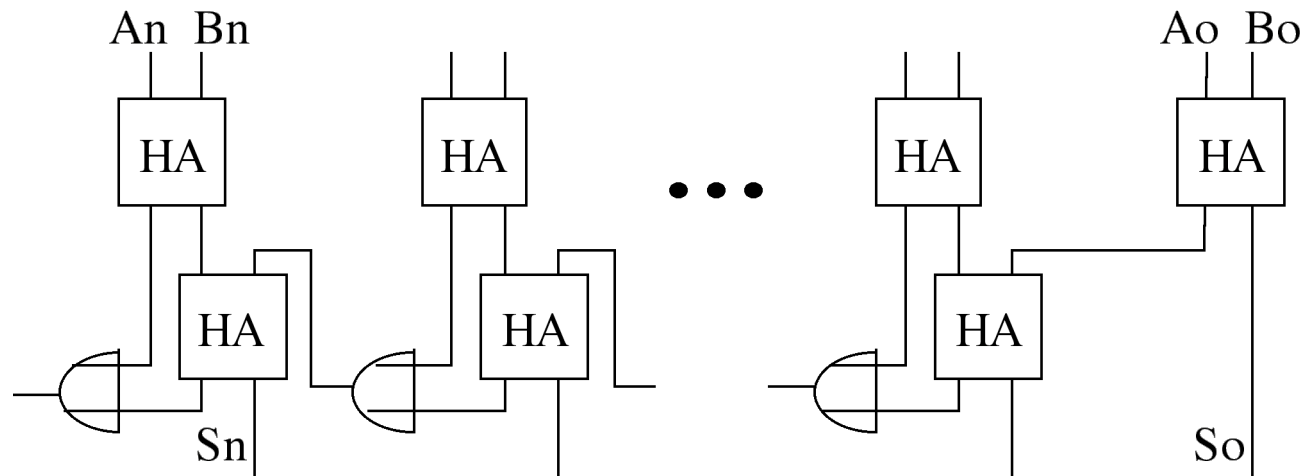
- Volladdierer

- für eine Binärstelle
- berücksichtigt auch den Übertrag einer früheren Stufe



- Paralleladdierer

- für die Addition eines Binärwortes
- die Summen der jeweiligen Binärstellen parallel bilden
- Übertrag durch die Stufen fortpflanzen lassen (Delay!)



-> Carry Lookahead

- CMOS MC14581 4-Bit ALU

- 4 bit parallel
- Steuerung über F0..F3, LA1, LA2

- Logische Funktionen:

A $A \wedge B$ $A \vee B$ $A \neq B$ True False

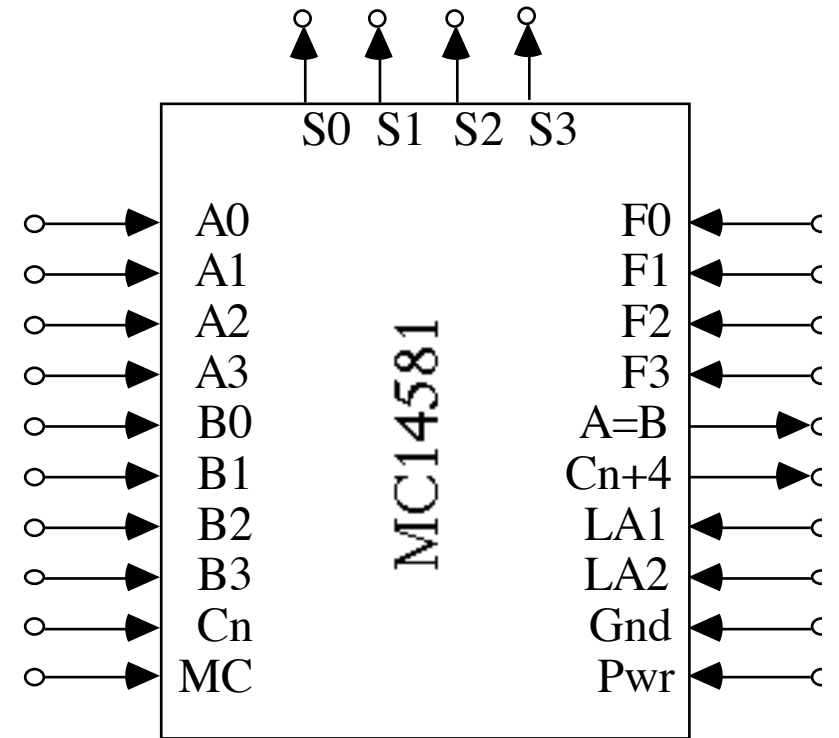
$\overline{A}$ $\overline{A} \wedge B$ $\overline{A} \vee B$ $\overline{A \neq B}$...

- Arithmetische Funktionen:

$A-1$ $A-B-1$ $A+B$ -1

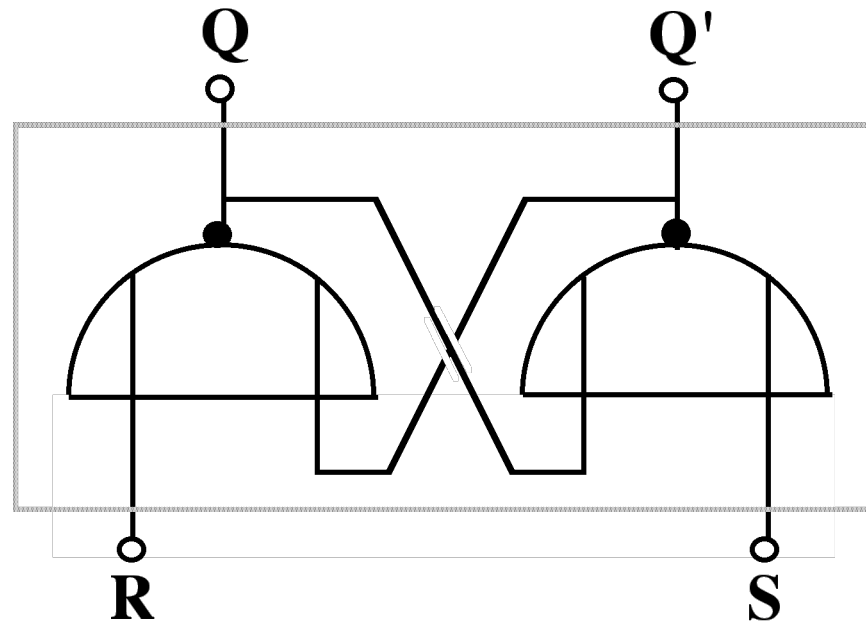
$A \wedge B - 1$ $A \wedge \overline{B} - 1$ $A + (A \vee B)$ $A = B$

...



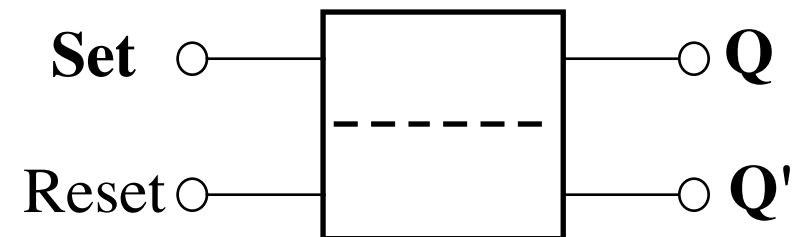
- Einfache Speicherzelle: Flip-Flop

- Speichern einer Binärstelle
- die beiden Hälften halten sich gegenseitig



- sog. RS-Flip-Flop:

- Setzen ("Set"),
- **R**ücksetzen ("Reset")



- Schaltfunktion für RS-Flip-Flop:

- $Q = \overline{R \vee Q'} = \overline{R \vee (\overline{S \vee Q})} = \overline{R} \wedge (S \vee Q)$

- $Q' = \overline{(S \vee Q)} = \overline{S} \wedge (\overline{R \vee Q'})$

- Wahrheitstafel:

R	S	Q_n	Q'_n
0	0	Q_{n-1}	Q'_{n-1}
0	1	1	0
1	0	0	1
1	1	0	0

- undefinierter Folgezustand für **R=1; S=1**

- Zwei Speicherungs-Zustände mit **R=0; S=0**:

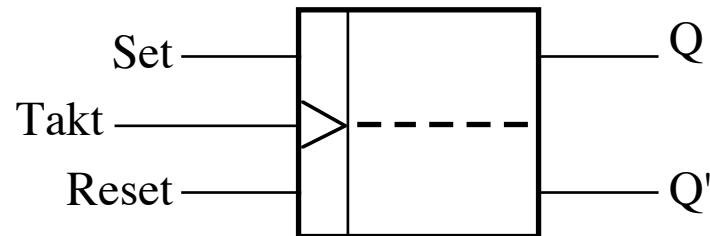
- $Q = 0; Q' = 1$

- $Q = 1; Q' = 0$

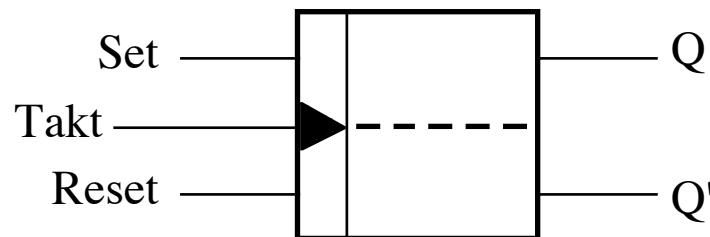
- fast immer $Q \neq Q' !$

- Zwischenzustände während Umschaltung

- Taktsteuerung
 - Eingangswerte stabilisieren lassen
 - dann Übernahmeimpuls
 - Eingangswerte werden nur zum Taktzeitpunkt berücksichtigt:
- Flankengesteuertes Flipflop (abfallend):

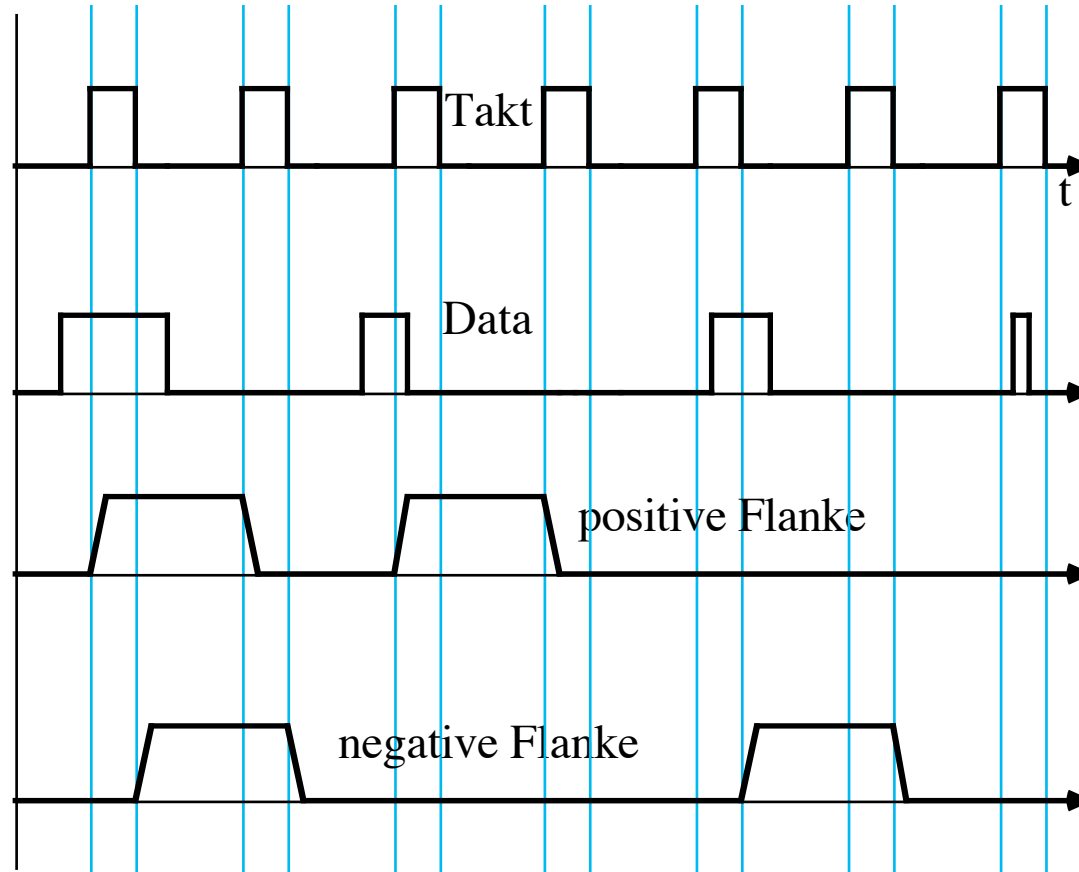


- Flankengesteuertes Flipflop (ansteigend):



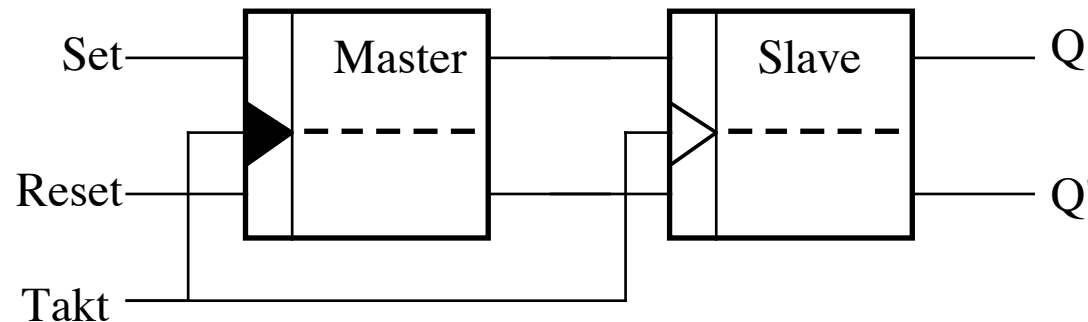
- Signalverlauf

- erst bei entsprechender Taktflanke Eingabe einlesen
- evtl. unterschiedliche Ergebnisse



- Master-Slave Flip-Flop

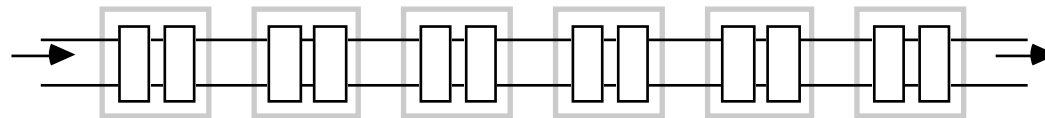
- Zwei FF-Stufen arbeiten phasenverschoben
- Master übernimmt Eingangswerte mit ansteigender Flanke
- Slave gibt mit abfallenden Flanke Werte an Ausgang



- Für Schaltungen mit heiklem Timing

- Registertransfers.
- Pufferregistern mit MS-FFs.

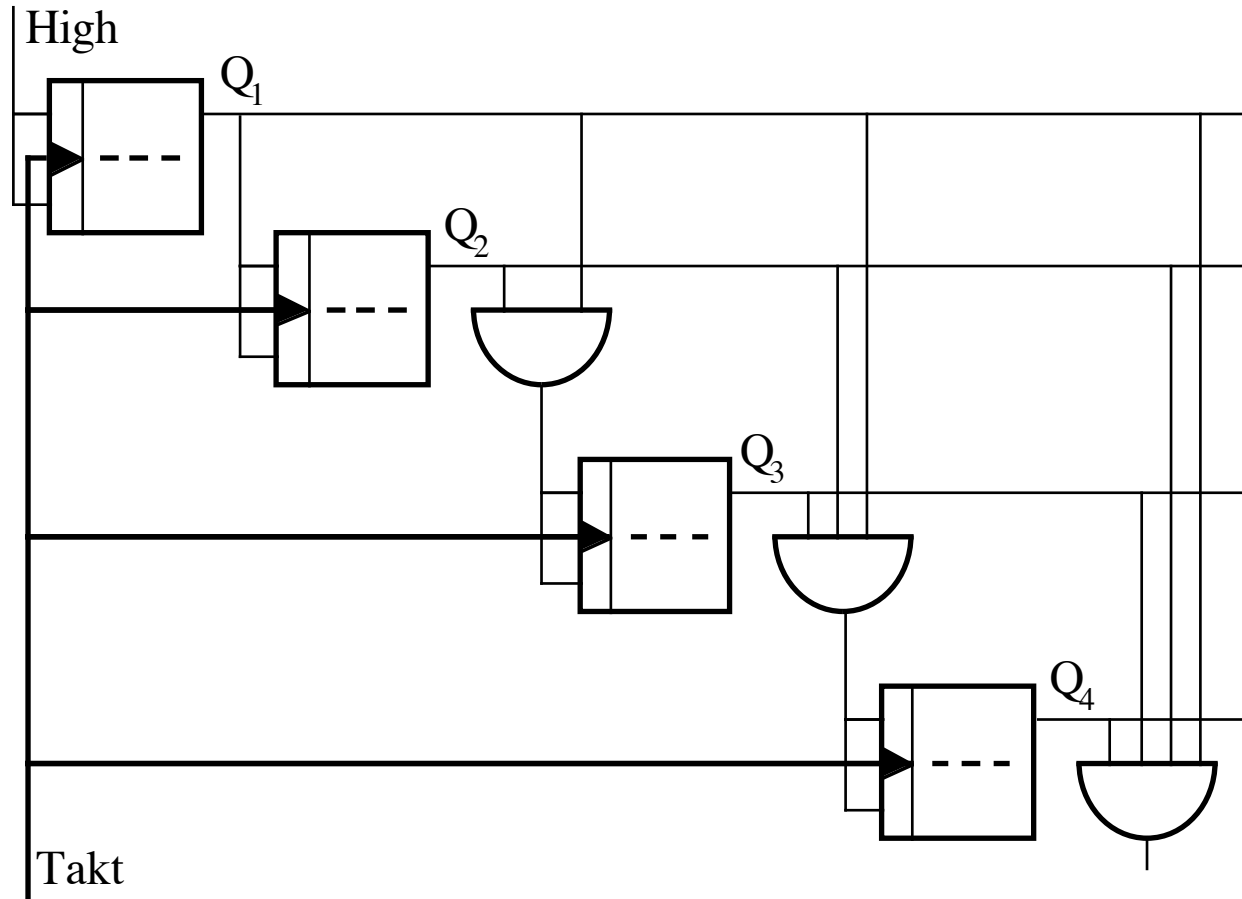
- MS-Schieberegister



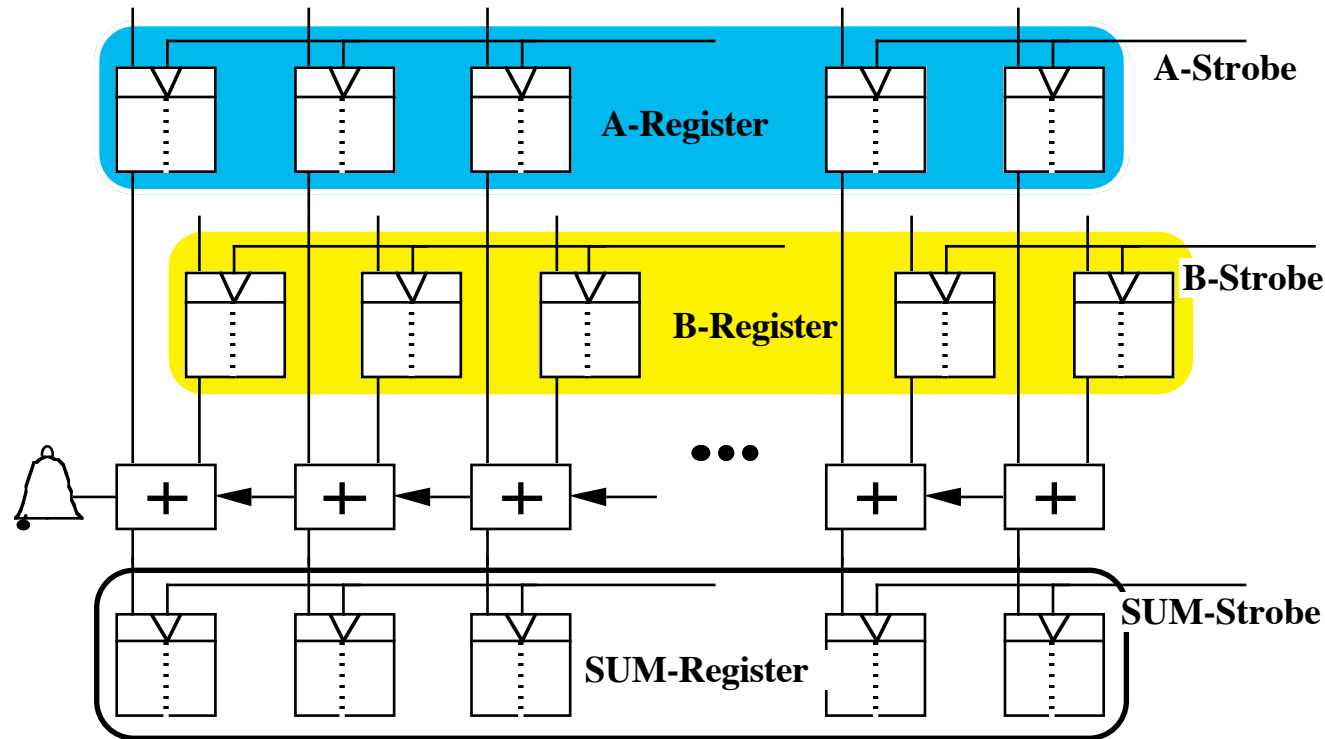
- als digitale Verzögerungsleitung
- für digitale Filter
- als Rechenoperation

- Zähler

- Jede Stufe erhält Takt
- schaltet in Abhängigkeit vom Zustand aller vorherigen Stufen



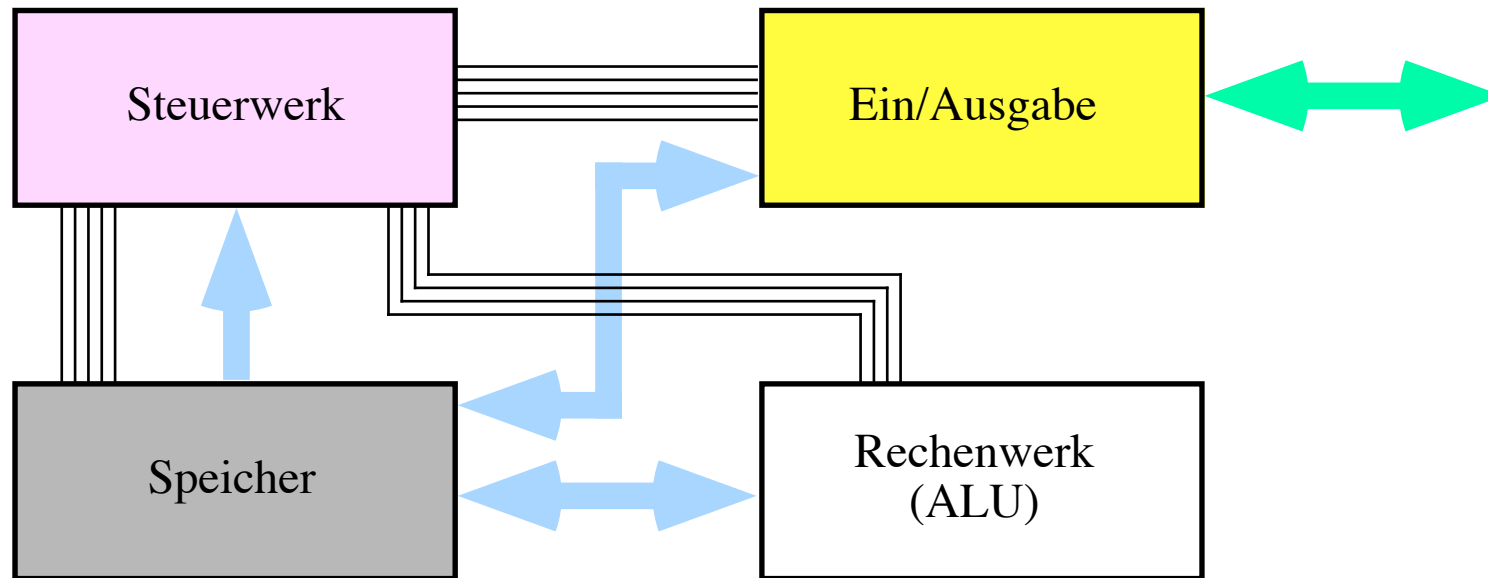
- Register speichert Zahlen
- Register in Verbindung mit Addierschaltung bzw. ALU



- Ablauf
 - A-Strobe zum Füllen des Registers A
 - B-Strobe zum Füllen des Registers B
 - Addition bzw. Übertrag abwarten,
 - Summe abholen mit SUM-Strobe
 - Überlauf?

4.3 Funktionseinheiten: Speicher, Prozessor, Bus, ...

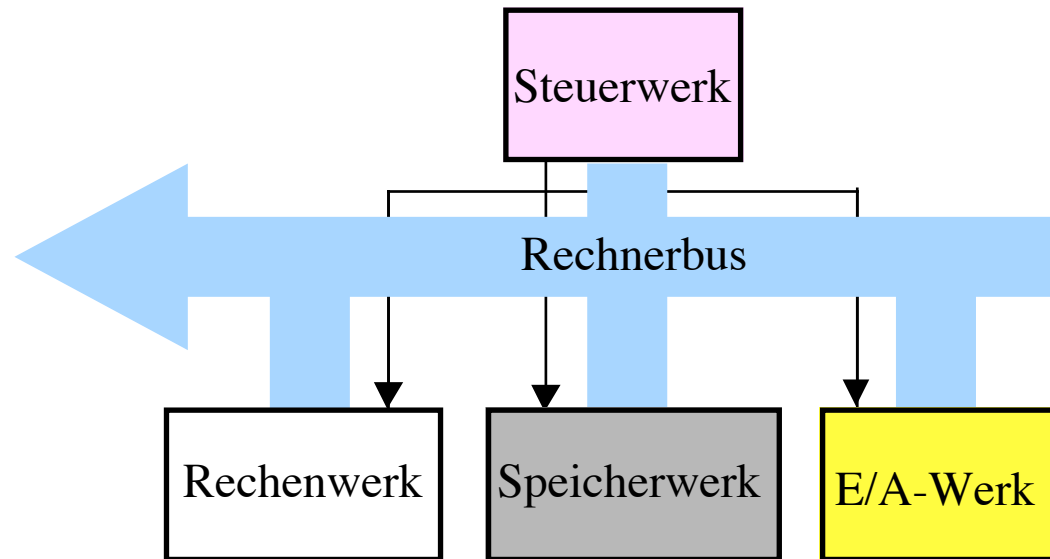
- klassischer Rechner nach J. v. Neumann
 - Rechnerwerk für arithmetische und logische Verknüpfungen
 - Universalspeicher für Daten und Programme



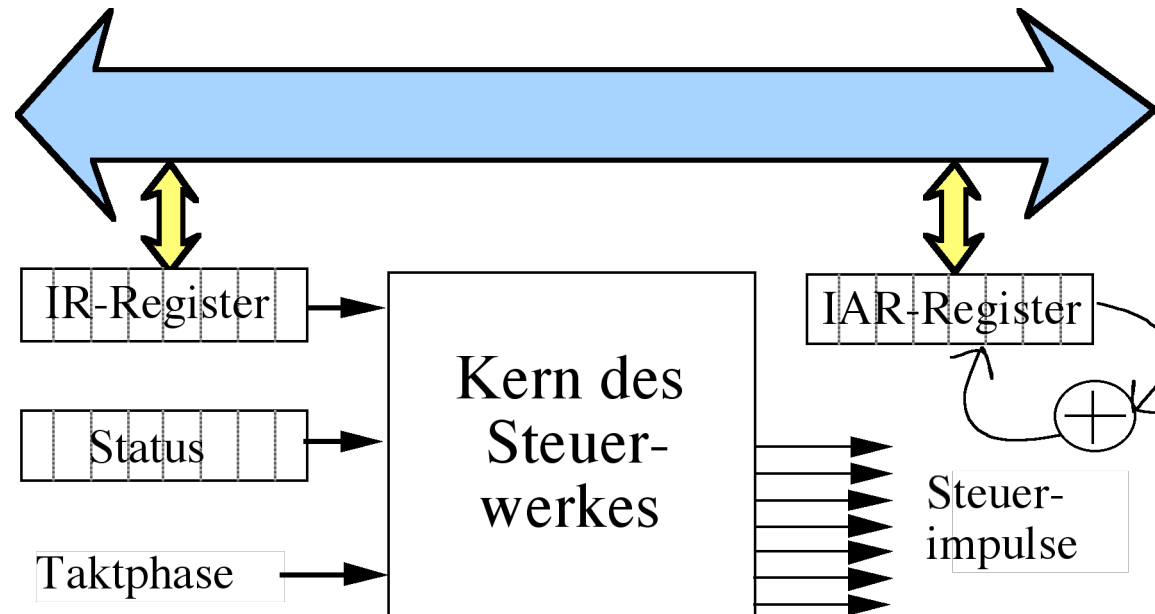
- Steuerwerk
 - sequentieller Ablauf des Programmes
 - Interpretation der Instruktionen => Steuerbefehle

- Busorientierter Rechner

- universell verbunden
- Schwerpunkt Transport und Verteilung
- weniger Verbindungen



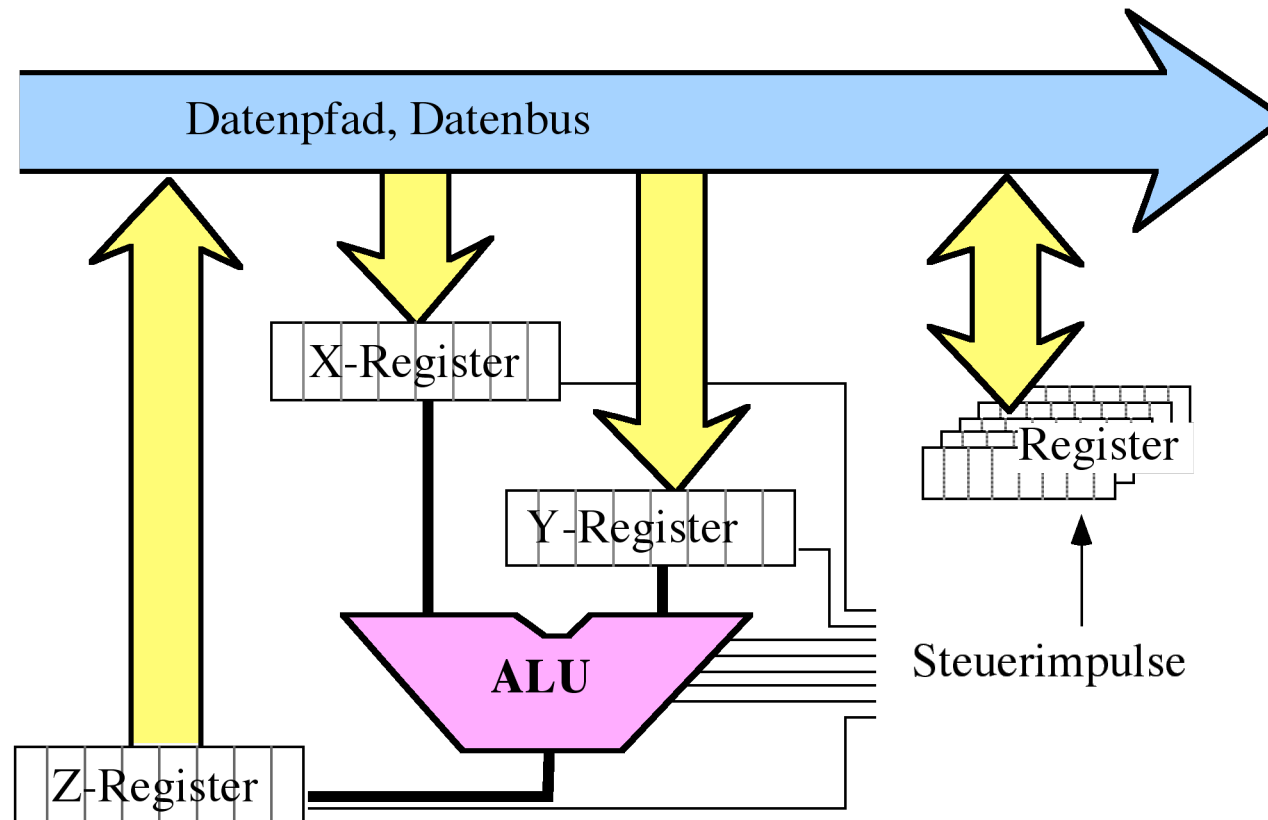
- Steuerwerk erzeugt Signale, die
 - Datentransfer auslösen
 - ALU-Operation auswählen
 - Speicheroperation auslöst



- Befehl wird in Sequenz von Kernzuständen umgesetzt
 - Kernzustand bestimmt Steuersignale
- Register
 - Instruktionsregister (IR)
 - Instruktionsadressregister (IAR) bzw. "Program Counter" (PC)
 - eventuell eigener Addierer für IAR

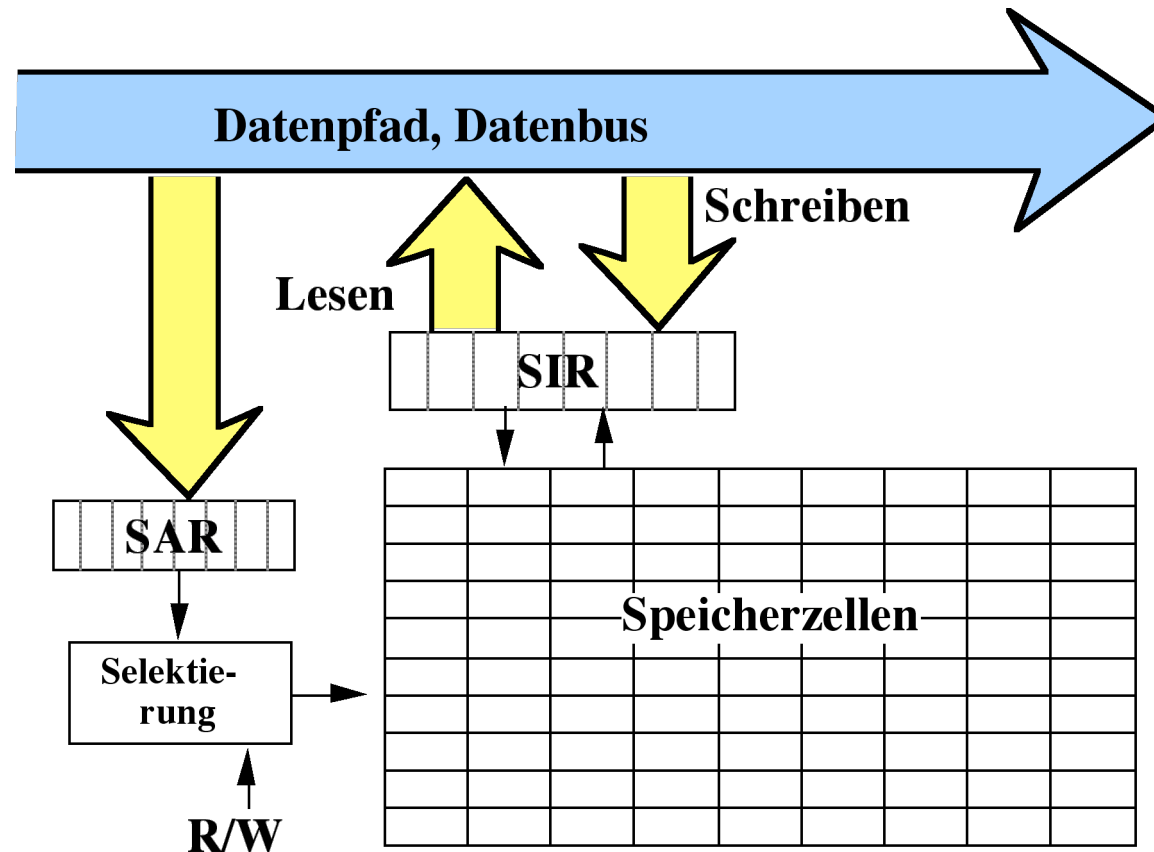
- Rechenwerk

- kombinatorisches Schaltnetz
- Addieren, Multiplizieren, UND, ODER, Vergleich, Shift, ...
- Ausgang der ALU liegt dauernd am Z-Register an
- X-Register und Y-Register liegen dauernd am ALU-Eingang an



- Zwischenresultate in Zusatzregistern
- Steuerleitungen bewirken Datenübernahme = Transport

- Speicher

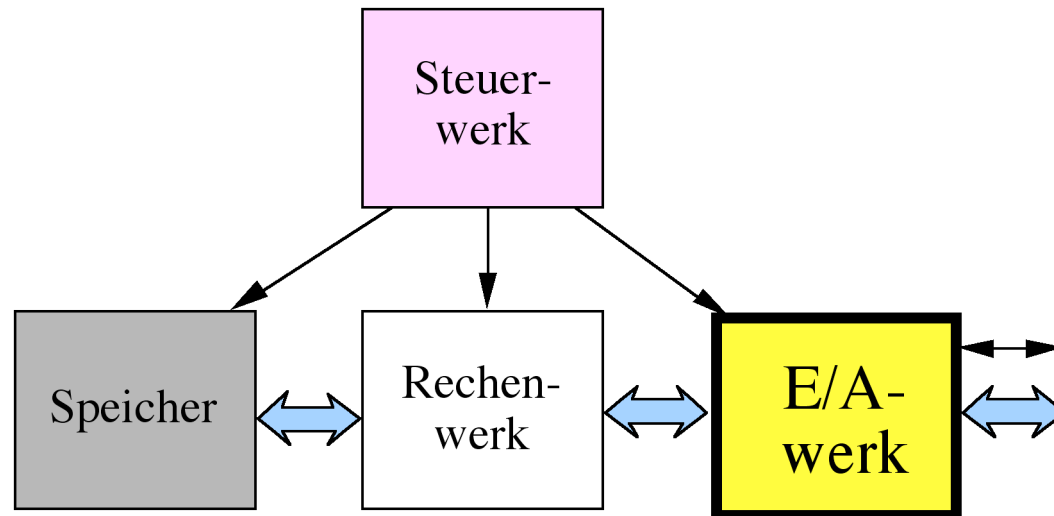


SAR = Speicheradressregister, SIR = Speicherinhaltsregister

- Random Access Memory (RAM)
 - Halbleiterspeicher halten Inhalt nur wenn sie "unter Strom" stehen
 - dynamische Speicher (DRAM) müssen periodisch aufgefrischt werden
- ROM: Festwertspeicher

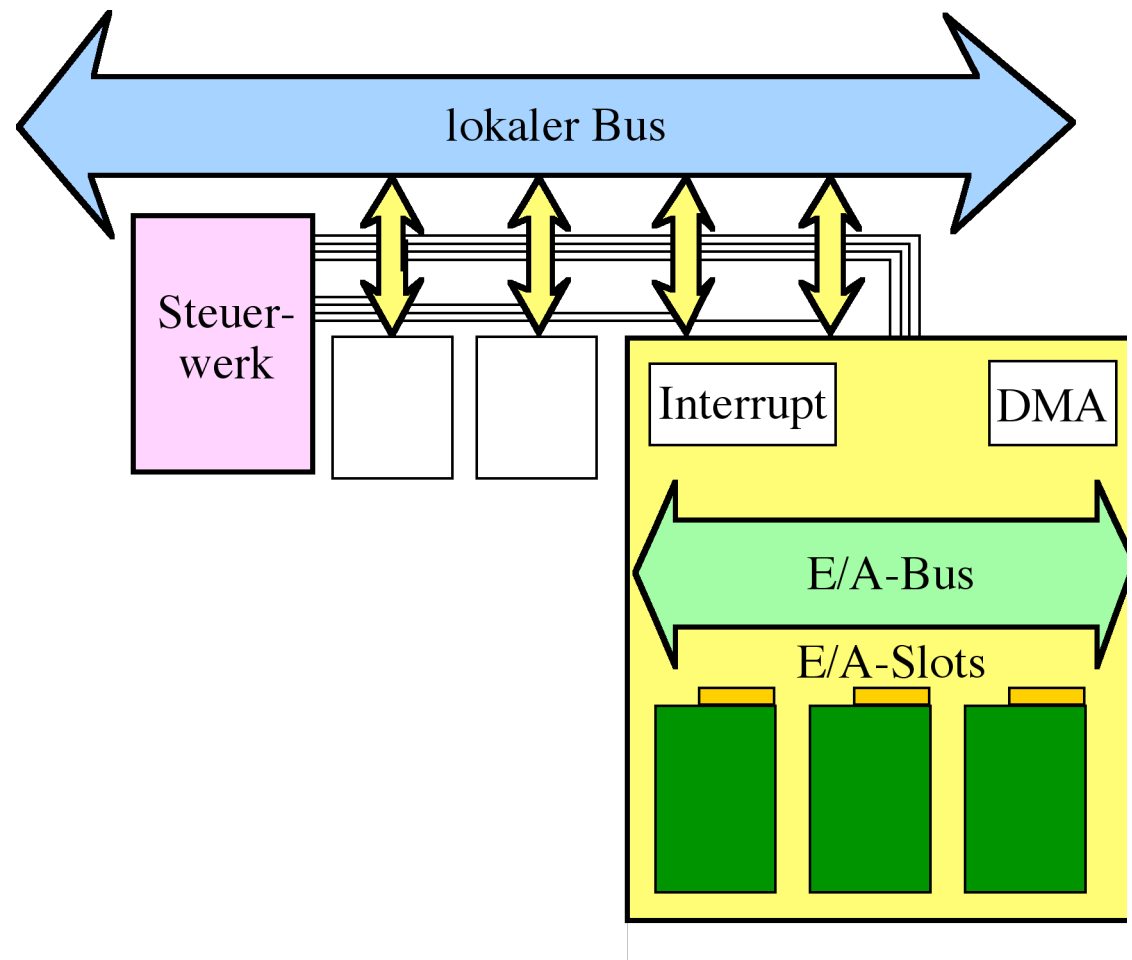
- Speicherhierarchie aus Kostengründen
- Register
 - 10-100 Wörter, < 1 Nanosekunde
 - kein separater Zyklus für die Adresse.
- Cache, Prozessorpufferspeicher
 - 8 KWörter bis 8 MBytes, < 10 Nanosekunden
 - häufig gebrauchte Daten und Codeteile
 - für Programmierer unsichtbar
- Hauptspeicher
 - 256 - 8192 Megabytes, ~ 10 - 50 Nanosekunden
 - evtl. inklusive Adressübersetzung
 - separater Zyklus für die Speicheradresse
- Hintergrundspeicher
 - Festplatte, Netz,
 - 40 - 1000 Gbytes, ~ Millisekunden
 - Zugriff über Betriebssystem
 - blockweise übertragen

- Ein-/Ausgabewerk
 - kontrolliert durch Steuerwerk
 - Transport via Rechenwerk in den Speicher
 - Bedienungsleitungen zum Peripheriegerät
 - Keine Parallelarbeit von Rechenwerk & E/A
 - Missbrauch der Rechenregister
 - Wartezyklen der CPU auf Peripheriegeräte



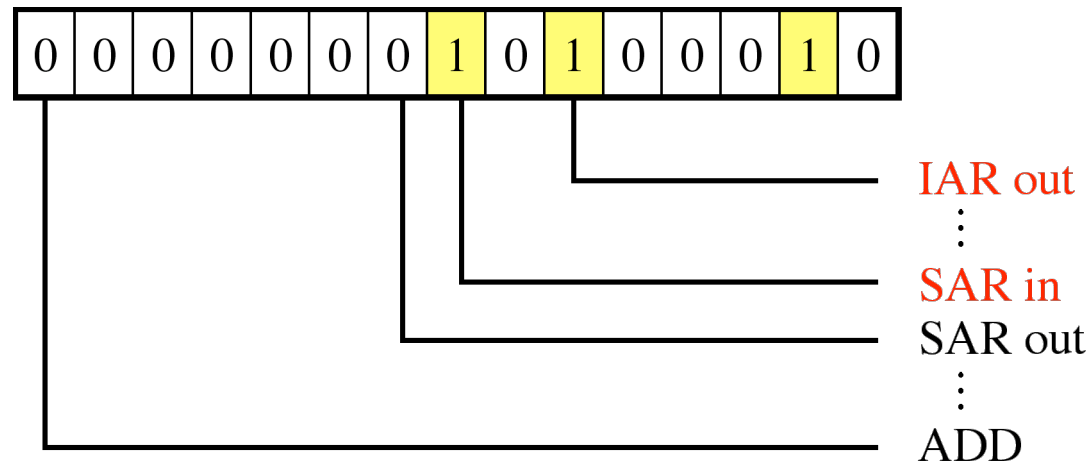
- Pfad in den Speicher als Flaschenhals
 - Resultate von Berechnungen
 - E/A-Übertragungen
 - Instruktionen

- Moderne Rechner
 - besonderer, standardisierter E/A-Bus
 - getrennt vom Prozessorbus
- autonomere E/A-Schnittstelle
 - externer Bus,
 - Abläufe parallel zum Rechenwerk
 - Interrupt-Funktion
 - DMA-Kanäle (direct memory access)
 - Display-Kontroller
 - Adapterplatinen
 - VLSI-Chips
- Grafik evtl. Extrabus
 - AGP

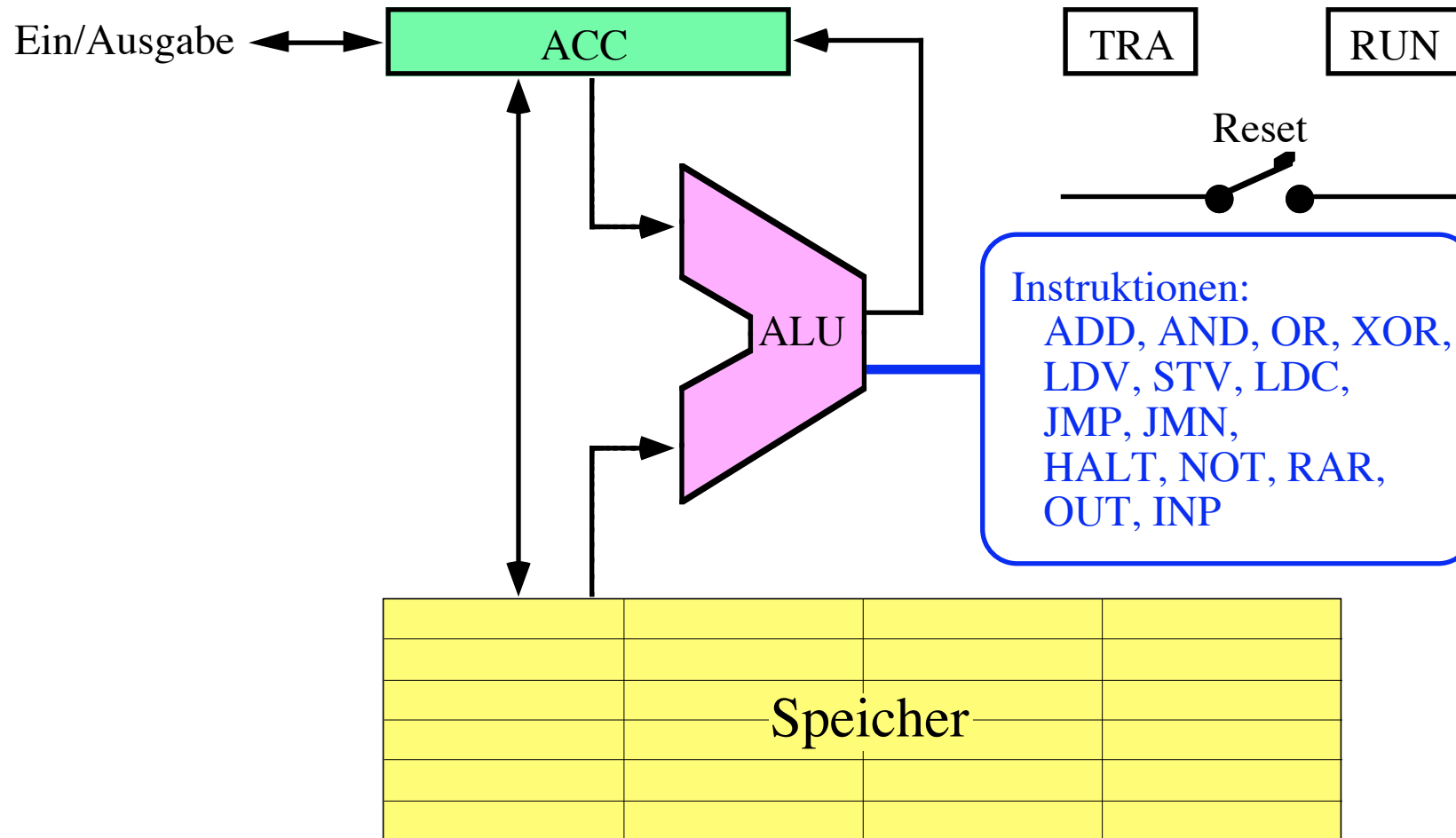


4.4 Maschinenbefehle

- Befehle liegen im Speicher
 - werden geholt wie Daten
 - kommen in Befehlsregister
 - Befehlszähler
- Befehl $\otimes$ Status $\Rightarrow$ Sequenz von Steuerworten
 - Bits des Steuerwortes an Steuerleitungen anlegen
 - Speicher, E/A, ALU, ... reagieren auf Steuerleitungen
 - in jedem Takt ein Steuerwort
 - Befehle können mehrere Steuerworte enthalten
 - Mikrobefehle



4.4.1 Minimalmaschine



- ein Register
- nur wenige, einfache Instruktionen
- TRA: Warten auf E/A
- 32-Bit Architektur

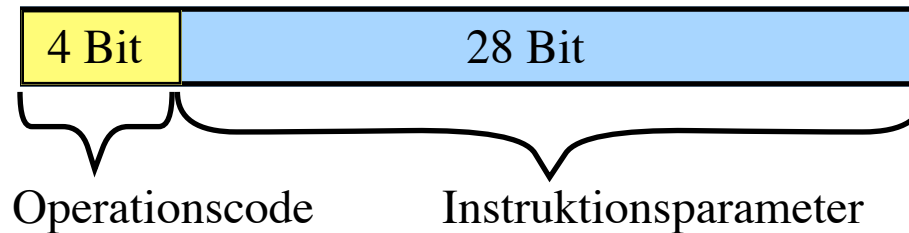
- Befehle zur Datenmanipulation

NOT		Komplementiere den Akkumulator, B-1 Komplement
RAR		Rotiere den Akkumulator um 1 Bit nach rechts
ADD	a	Addiere den Inhalt der Speicherzelle a zum Akkumulator $ACC \leftarrow ACC + \text{Speicher}[a]$
AND	a	$ACC \leftarrow ACC \wedge \text{Speicher}[a]$
OR	a	$ACC \leftarrow ACC \vee \text{Speicher}[a]$
XOR	a	$ACC \leftarrow ACC \oplus \text{Speicher}[a]$
EQL	a	$ACC \neq \text{Speicher}[a]: ACC \leftarrow 0$ $ACC = \text{Speicher}[a]: ACC \leftarrow 11..1$
LDV	a	$ACC \leftarrow \text{Speicher}[a]$
STV	a	$\text{Speicher}[a] \leftarrow ACC$
LDC	c	$ACC \leftarrow c$

- Kontrollfluß

JMP	p	Nächste Instruktion in Speicher[p]
JMN	p	Sprung nach p falls ACC negativ
HALT		$RUN \leftarrow 0$ Instruktionsausführung hält an Später Restart nur aus Speicher[0] möglich

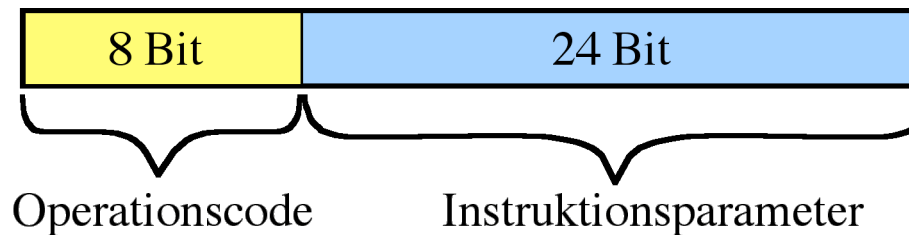
- Basisformat



- 0,1,2,3 ADD, AND, OR, XOR
- 4, 5, 6 LDV, STV, LDC
- 7, 8, 9 JMP, JMN, EQL
- A .. E future use

- Erweitertes Format

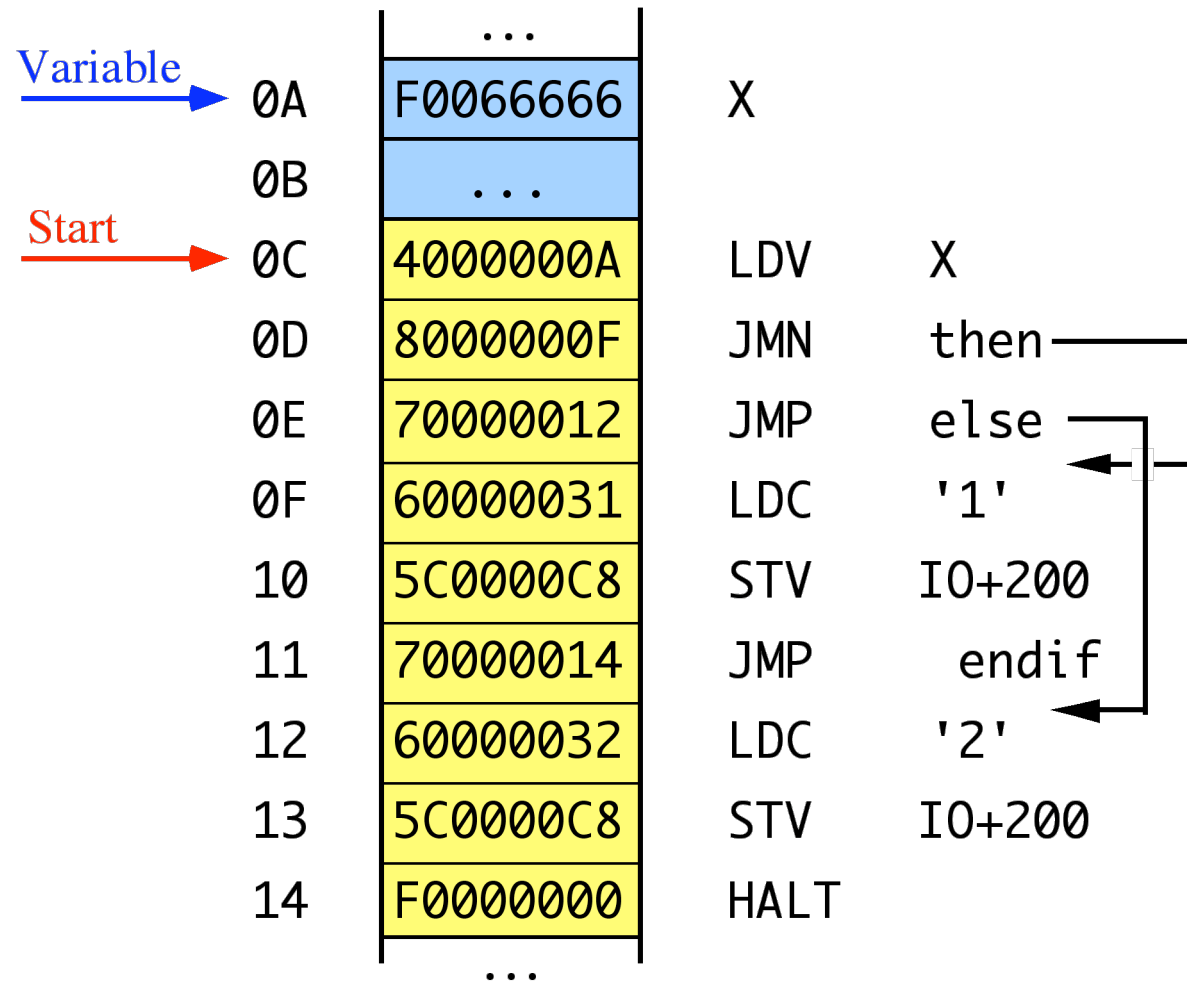
- mehr als 16 Instruktionen möglich,
- 24 Bit Parameter & nicht immer benützt



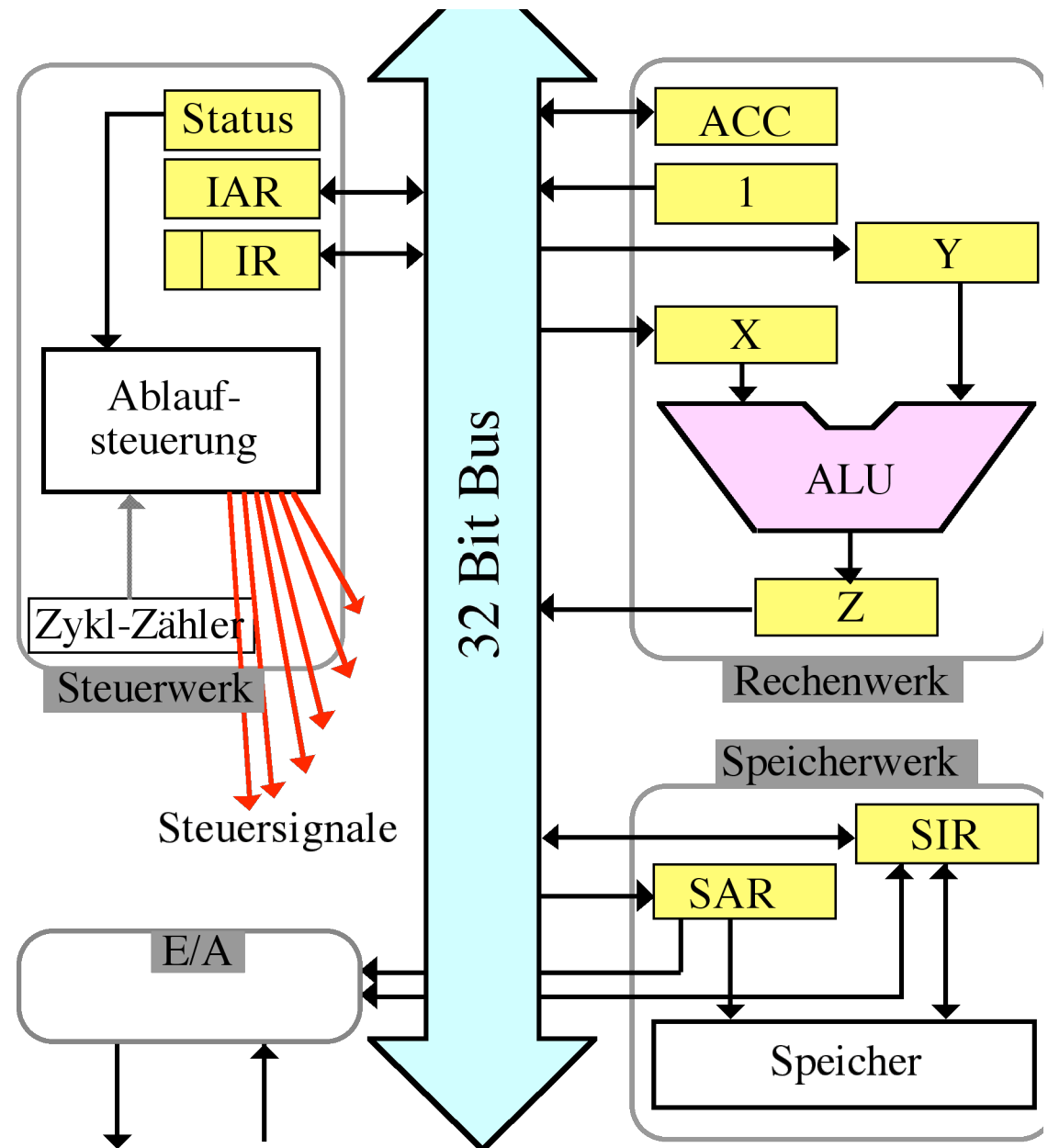
- F0, F1, F2 HALT, NOT, RAR
- F3, F4,... future use

- Assemblerprogrammieren im Schnelldurchgang

```
if (X < 0)    printf("1")
              else printf("2");
```
- Hexadezimaler MiMa-Speicherauszug

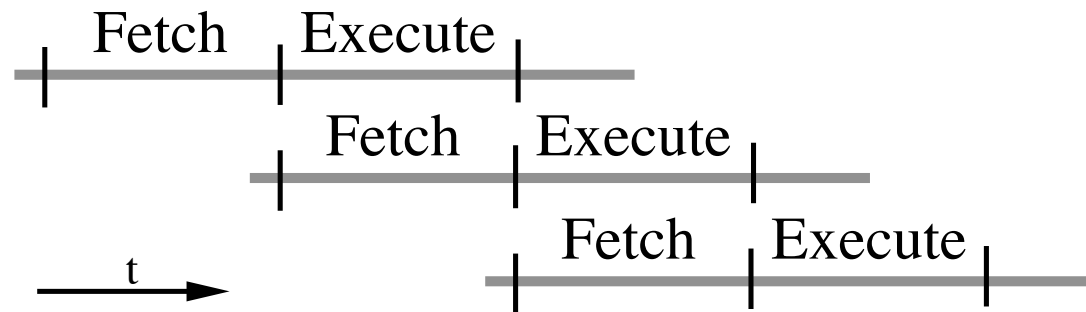


- Speicher
 - statisch
 - 24 Bit/Wort
- 1-Register
 - z.B. Inkrement
- Steuerwerkregister
 - Instruktionen
 - Instruktionsadresse (PC)
 - Status
- ALU
 - Eingang X,Y
 - Ergebnis Z
 - ADD, AND, XOR, OR
- Registertransfer
 - Quell-Register an Bus
 - Zielregister übernimmt
 - Steuerleitungen



4.4.2 Ablauf der Instruktionen

- Zweiteilung der Befehle
 - Fetch-Zyklus holt Befehl
 - Execute Zyklus führt Befehl aus
 - eventuell überlappend



- Unterzyklen im Steuerwerk
 - Mikrozyklus, "Minor Cycle", Taktphase
 - andere Steuerimpulse in jedem Unterzyklus
 - Registertransferebene
- Mima-Instruktionen
 - 12 Unterzyklen
 - 5 Unterzyklen Fetch
 - 7 Unterzyklen Execute

- Ablauf der Lade-Instruktion

LDV a ; laden von Inhalt der Speicherzelle a in ACC

 ; ACC <= Speicher[a]

;Fetch-Zyklus

1. IAR -> (SAR, X), Leseimpuls an Speicher
2. 1 -> Y, ALU auf Addition schalten,
 Warten auf Speicher
3. Warten auf ALU, Warten auf Speicher
4. Z -> IAR, Warten auf Speicher
5. SIR -> IR

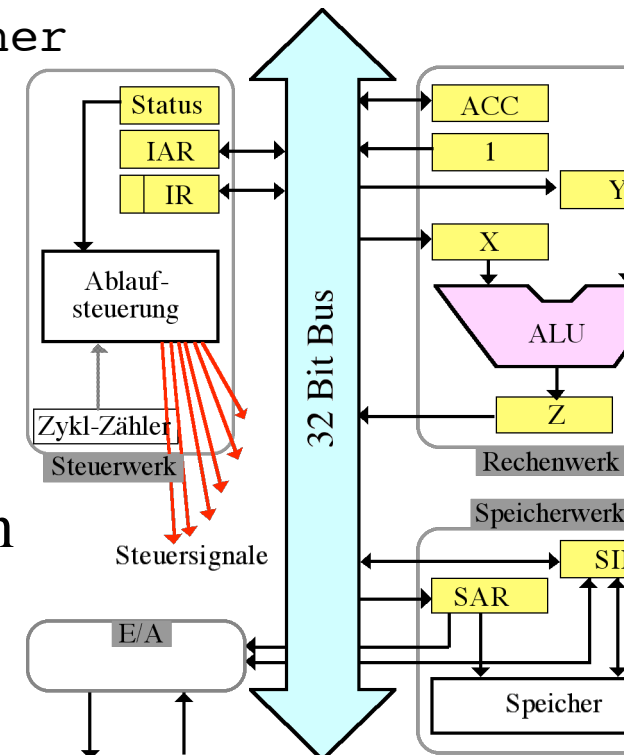
;Fetch-Ende, jetzt Execute-Zyklus

6. IR -> SAR, Leseimpuls an Speicher
- 7., 8., 9. Warten auf Speicher
10. SIR -> ACC
- 11., 12. leere Unterzyklen

;Execute-Ende, nächste Instruktion

- IAR -> (SAR, X)

- IAR auf den Bus legen
- SAR und X mit Steuerimpuls vom Bus lesen lassen



- Ablauf der ADD-Instruktion

ADD a ; addieren von Inhalt der Speicherzelle a zum ACC

;Fetch-Zyklus

1. IAR $\rightarrow$ (SAR, X), Leseimpuls an Speicher
2. 1 $\rightarrow$ Y, ALU auf Addition schalten,
Warten auf Speicher
3. Warten auf ALU, Warten auf Speicher
4. Z $\rightarrow$ IAR, Warten auf Speicher
5. SIR $\rightarrow$ IR

;Fetch-Ende, jetzt Execute-Zyklus

- ```

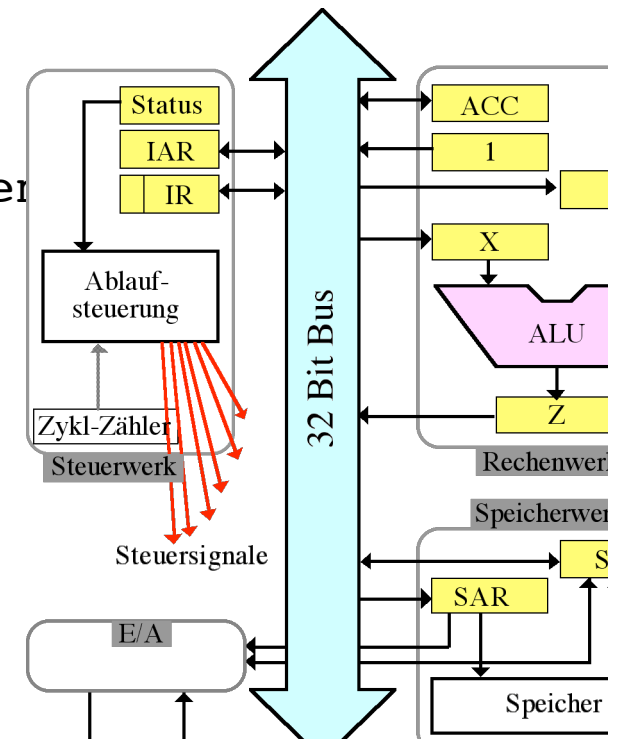
6. IR -> SAR, Leseimpuls an Speicher
7. ACC -> X, Warten auf Speicher
8., 9. Warten auf Speicher
10. SIR -> Y, ALU auf Addition schalten
11. warten auf ALU
12. Z -> ACC

```

```

;Execute-Ende, nächste Instruktion

```



- JMN p - Jump if Negative

- Bedingter Sprung zur Instruktion im Speicherwort[ p ]
- negativ bedeutet, das oberste Bit im Akkumulator ist 1
- X, Y-Register und ALU werden ignoriert
- kein Datenzugriff auf Speicher

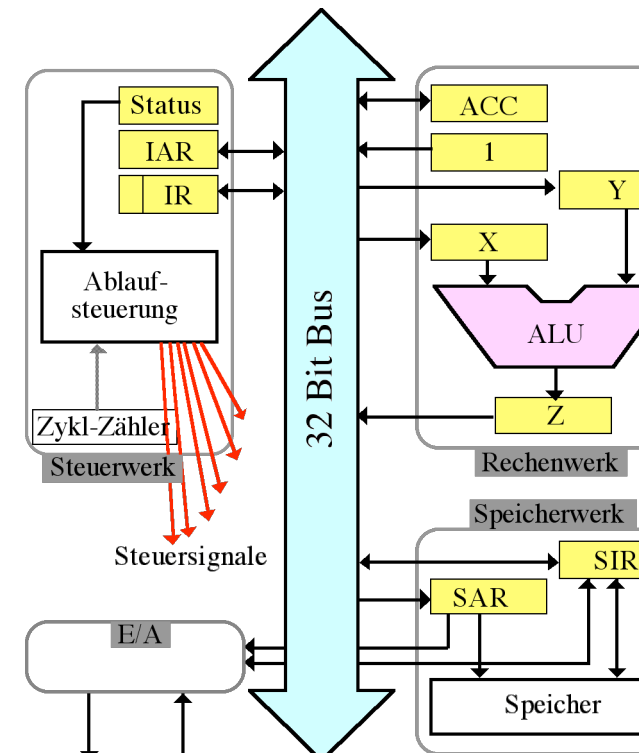
### ;Fetch-Zyklus

1. IAR  $\rightarrow$  (SAR, X), Leseimpuls an Speicher
2. 1  $\rightarrow$  Y, ALU auf Addition schalten,  
Warten auf Speicher
3. Warten auf ALU, Warten auf Speicher
4. Z  $\rightarrow$  IAR, Warten auf Speicher
5. SIR  $\rightarrow$  IR

### ;Fetch-Ende, jetzt Execute-Zyklus

- 6a. falls vorderstes Bit in ACC = 1:  
IR  $\rightarrow$  IAR
  - 6b. falls vorderstes Bit in ACC = 0:  
leerer Unterzyklus
7. ... 12. leere Unterzyklen

### ;Execute-Ende, nächste Instruktion



- Start der Mima
  - Drücken der Reset-Taste
    - 0 -> IAR
    - 0 -> TRA
    - 1 -> RUN
- Maschine beginnt ab Adresse 0 Instruktionen auszuführen
  - evtl. andere, festgelegte Adresse
- Urlader
  - einige Speicherzellen ab Adresse 0 sind Festwertspeicher
  - enthalten einen Urlader ("Boot-Strap Loader")
  - liest Programm von einem bestimmten Gerät in den Speicher
  - beginnt mit dessen Ausführung
- Schaltpult mit Schaltern und Lämpchen zum
  - Modifizieren und Inspizieren von Registern und Speicherinhalten
  - bei älteren Rechnern

- Integration der E/A in den normalen Adressbereich

- memory mapped I/O
- In - LDV, Out - STV
- Geräte langsamer als Speicher
- variable Geschwindigkeit
- => Handshake

- Speicherwerk wertet Adresse aus

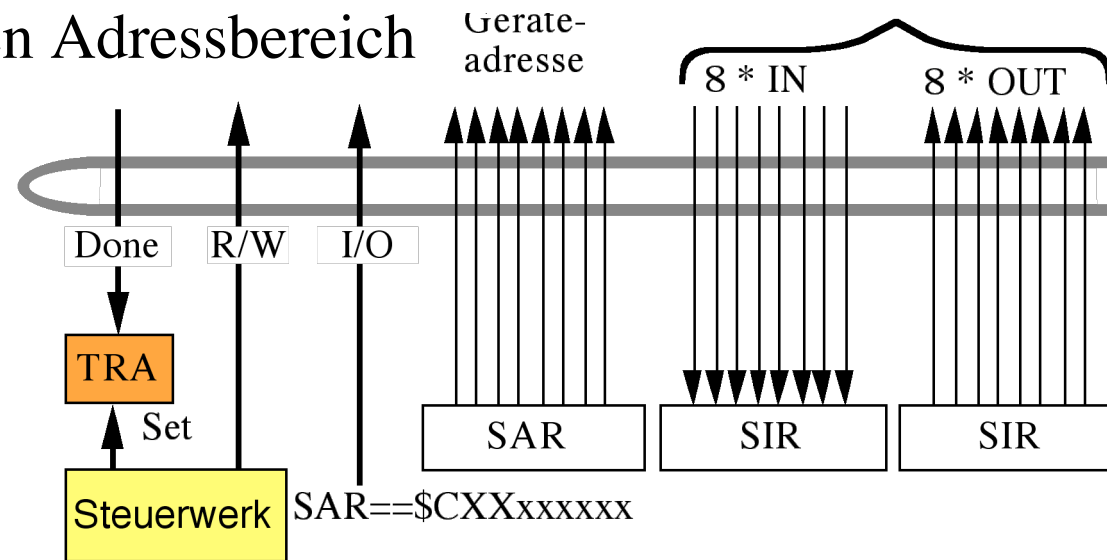
- $\text{Adr} == \$\text{CXXxxxxxx} \Rightarrow \text{E/A}$
- $\text{Adr} \neq \$\text{CXXxxxxxx} \Rightarrow \text{Speicher}$

- TRA-Bit

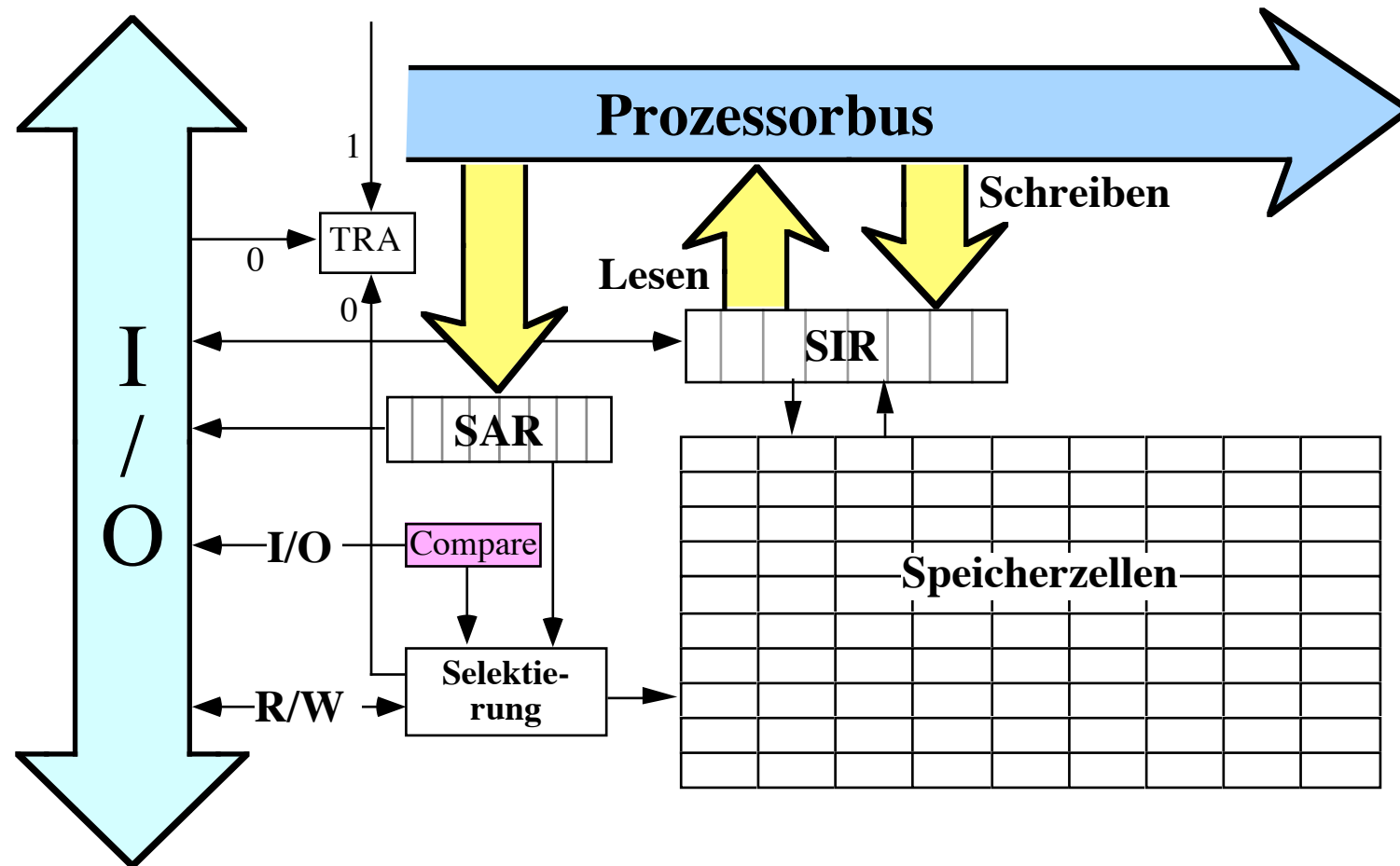
- vom Steuerwerk gesetzt
- von Speichercontroller oder Peripherie zurückgesetzt
- Steuerwerk wartet auf  $\text{TRA}=0$

- Geräte beobachten die Geräteadresse im EA-Werk

- Als Bus ausgeführte E/A-Schnittstelle



- Speicher mit Memory-mapped I/O
  - TRA Bit wird vom Prozessor gesetzt
  - von der Selektionsschaltung zurückgesetzt
  - oder vom Gerätebus zurückgesetzt



SAR = Speicheradressregister, SIR = Speicherinhaltsregister

### 4.4.3 Erzeugung der Steuersignale

- Eingangssignale zum Steuerwerk:
  - Operationscode aus IR-Register (4/8 Bit)
  - Vorzeichenbit im Akkumulator
  - Mikrozyklus 1..12 aus externem Zähler
  - Status-Register = (RUN, TRA)

=>  $8+1+4+2 = 15$  Eingangsleitungen

- Ausgangssignale vom Steuerwerk
  - Rechteckige Steuer-Impulse
  - evtl. statische Steuerung (z.B. für ALU)
  - Takteingang für Flip-Flops
  - Out-Enable für Tristate-Ausgang am Bus

- RUN: 1 Leitung zum Clear-Eingang
- TRA: 1 Leitung zum Preset-Eingang
- ALU: 3 statische Pegel für die unterschiedlichen ALU-Funktionen
- Speicher: 2 Leitungen für Read & Write
- je 1 Input-Enable für Register
  - X, Y, ACC
  - IR, IAR, SIR, SAR
- zusätzlich Output-Enable für Register
  - ACC, 1, Z
  - IAR, SIR, IR (nur Bit [0..27])

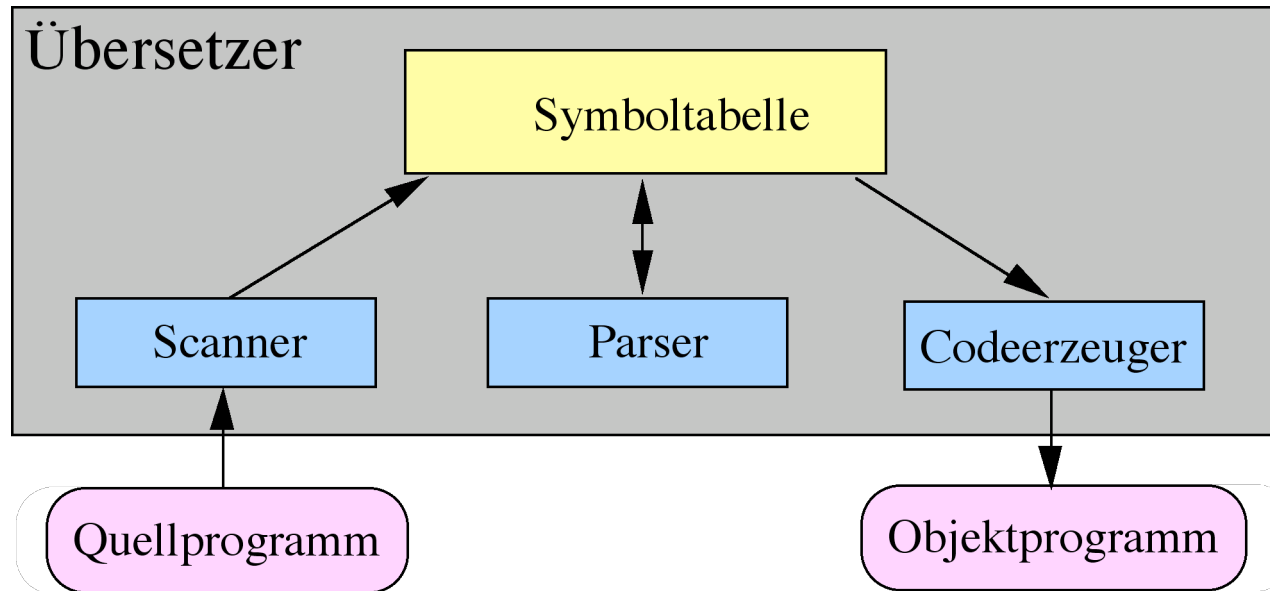
=>  $1+1+3+2+7+6 = 20$  Steuersignale

- Steuerwerk kann realisiert werden:
  - als 20 Schaltfunktionen mit 15 Variablen
  - als ROM mit  $2^{15}$  Wörtern à 20 Bit



## 4.5 Compiler

- Brücke Programmiersprache - Objektcode
  - C, Pascal, Modula, Fortran,
  - IA, 68000, PowerPC, 8051, Z80, DSPs, ...
  - Name => Adresse
  - Statement => Instruktionen
  - Prozeduraufruf => Sprung und Rücksprung



- Einlesen des Programmes (Scanner)
  - findet Symbole
  - Identifier, Konstanten, ...
- Syntaktische Analyse
  - zulässige Symbole werden verarbeitet ("Parsing")
  - für unzulässige Symbole Fehlermeldungen erzeugen
  - über "Look-Ahead" entschieden, welcher Pfad gewählt werden soll
  - bei schwierigen Programmiersprachen sehr weit vorausschauen
  - LL1 Programmiersprachen => maximal 1 Symbol Look-Ahead.
- Erzeugen der Maschinenbefehle (Codegenerierung)
  - syntaktische Prozeduren können auch die Instruktionen erzeugen
- Strategien
  - rekursive decent
  - bottom-up
  - top-down
  - Übersetzung für virtuelle Maschine besonders einfach
  - zeilenweise Übersetzung

- Beispiel: Ausdruck übersetzen

$$a = b - (c - ((d * e) - g/h))$$

|     |      |                        |
|-----|------|------------------------|
| LDV | g    |                        |
| DIV | h    | ; g/h                  |
| STV | hilf | ; optimiert wird nicht |
| LDV | d    |                        |
| MUL | e    | ; (d*e)                |
| SUB | hilf | ; -                    |
| STV | hilf |                        |
| LDV | c    |                        |
| SUB | hilf |                        |
| STV | hilf |                        |
| LDV | b    |                        |
| SUB | hilf |                        |
| STV | a    | ; a= ...               |

## 5. Betriebssystem: Abstrahieren und Koordinieren

- Sammlung häufig gebrachter Softwarekomponenten

- Hardware-Abstraktion

```
printf("%c", char);
```

```
(* anstelle von:
```

```
MOVE char,$A7823A
```

```
oder: *)
```

```
MOV AX,[BP]
```

```
OUT $3F8,AX
```

- essentielle Software

- Zuteilung der Ressourcen

- CPU, Speicher, Bildschirm, ...

- Oft mit Kommandointerpreter verwechselt

- Shell, Command.COM, ...

- Explorer, Finder, ...

- Dateisystem

- ebenfalls nur Teil des Betriebssystems

- Verwaltung von Plattenplatz (Sektoren)

- Zuordnung und Wiederverwendung

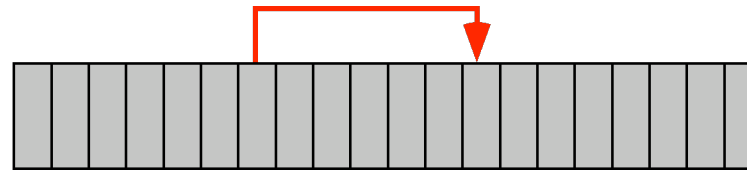
- Abstraktion Sektor - Bytestrom

## 5.1 Verteilung der Ressourcen

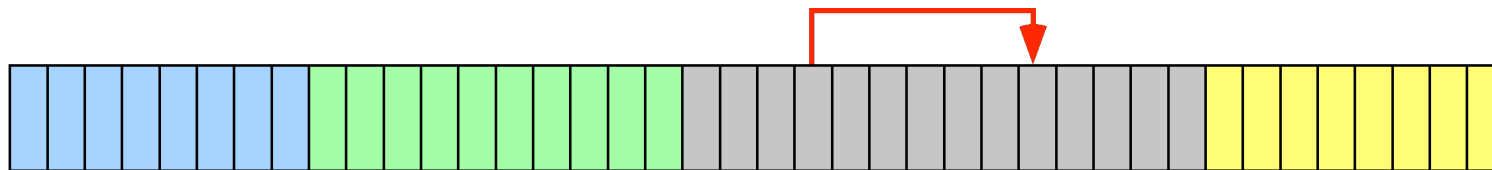
- Prozessor
  - Verteilung der Arbeit auf Prozessoren
  - Unterbrechungen
  - wartende Prozesse
- Speicher (RAM, Festplatte)
- Ein/Ausgabe
  - Bildschirm, Drucker
  - Audio-Ausgabe
- Verteilung des Mangels
  - Bildschirmfläche endlich => Fensterkonzept
  - Tastatur, Maus
  - Audio-Ausgang => Mixer
  - Prozessorzeit => Zeitscheiben, Zeitüberwachung

## 5.1.1 Prozesse und Scheduling

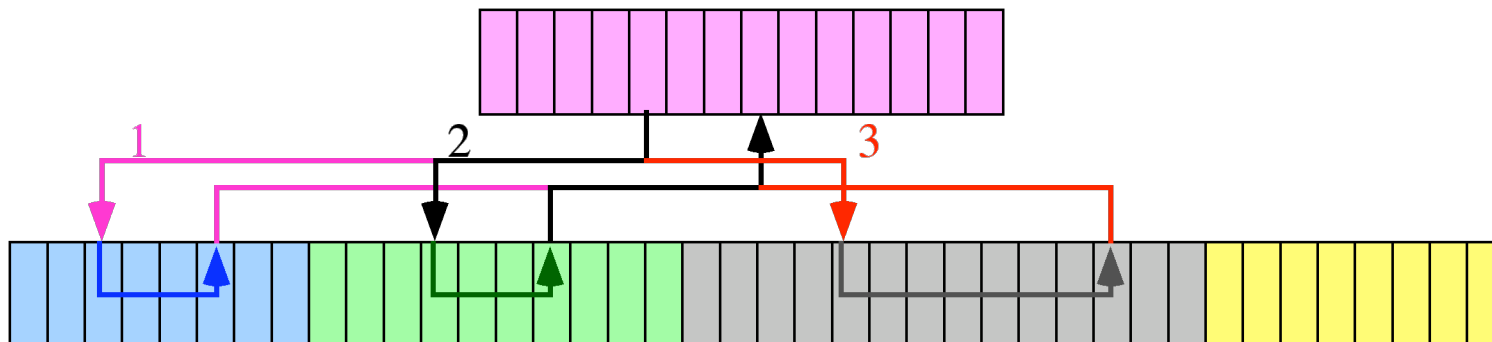
- Instruktionen werden sequentiell ausgeführt
  - in der Regel nicht gleichzeitig
  - ein Befehl nach dem anderen
  - Sprünge verändern nur die Reihenfolge



- Programm = ausführbares Objekt = Prozeß
  - mehrere Prozesse liegen im Speicher



- Benutzerprozesse und ein besonderer Steuerprozeß



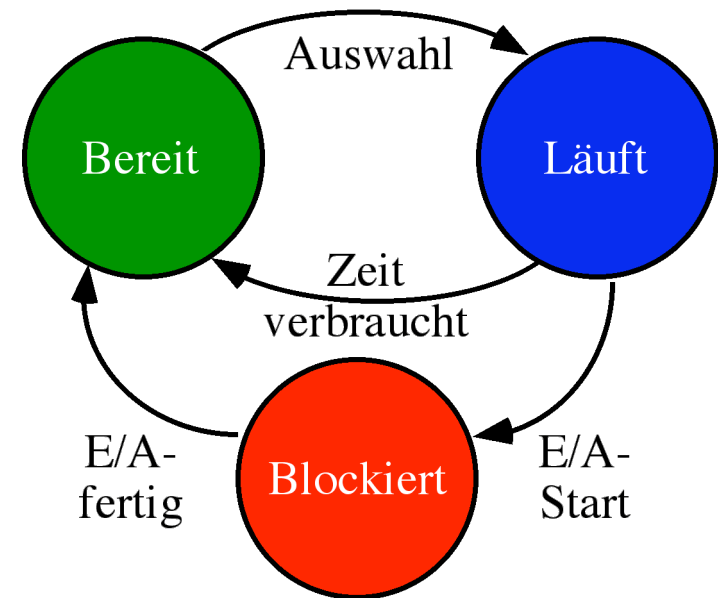
- Besonderer Steuerprozeß
  - gibt Ausführungsrecht befristet ab an Benutzerprozesse
  - verteilt Ausführungsrecht 'gerecht'
  - hat besondere Rechte
  - kennt alle anderen Prozesse
- Rückwechsel vom Benutzerprozeß zum Scheduler
  - Sprung in den Scheduler
  - freiwillig
  - erzwungen nach Fristablauf (=> Unterbrechung)
- Prozeßeigenschaften
  - Wichtigkeit (Priorität)
  - benutzte Ressourcen (Festplatte, Drucker, ...)
  - verbrauchte Zeit
  - zugeordneter Speicher
- Auslagern (Swapping)
  - falls Hauptspeicher zu klein
  - exaktes Prozeßabbild auf Festplatte schreiben

- Prozesse

- selbständige Codeeinheiten
- Object-Code
- Speicher
- Attribute

- Multi-Tasking

- mehrere Prozesse laufen 'verschachtelt'
- "Zeitmultiplex"
- für den Benutzer gleichzeitig
- verschiedene Strategien des Wechsels



- Prozesswechsel (Umschalten zwischen Programmen)

- Anhalten eines Prozesses
- Speichern der Status-Information (PC, Register etc.)
- Laden der Status-Information des neuen Prozesses
- Wiederaufnahme der Ausführung bei der nächsten Instruktion
- z.B. nach Zeit t, wenn gewartet werden muß, ...

- Einplanung ≠ Scheduling



- Non-Preemptive Scheduling

- Prozesse nicht unterbrechbar



- Weniger Prozesswechsel
- Kritische Prozesse können nicht unterbrochen werden

- Preemptive Scheduling

- Prozesse durch andere Prozesse mit höherer Priorität unterbrechbar

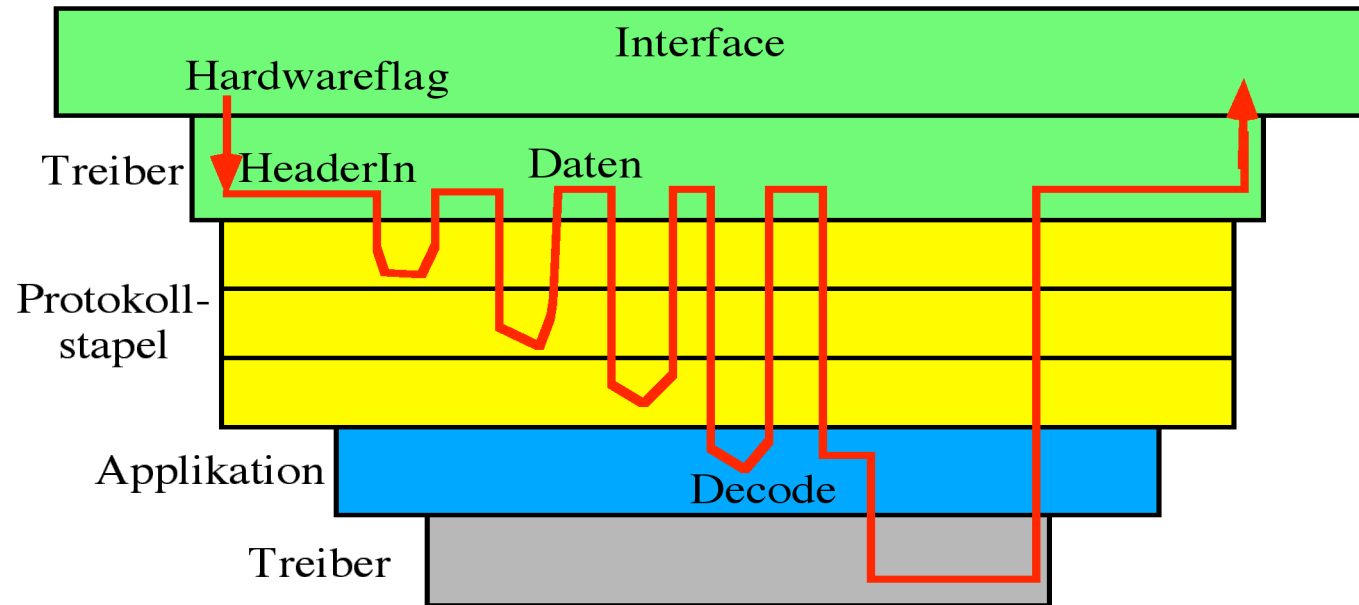


- Oft in Betriebssystemen vorhanden für CPU
- Prozesswechsel häufig und teuer

- Prioritätsfestlegung

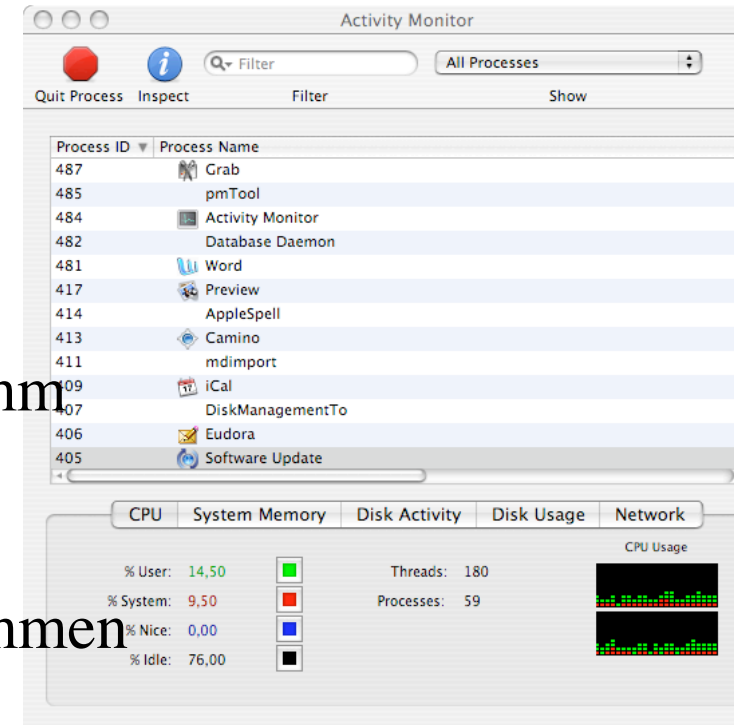
- Wichtigkeit
- nahe 'Abgabe'-Termine
- lange gelaufen => Priorität sinkt

- Unterbrechungsmanagement (Interrupts)
  - Interrupt-Service-Routine im Netzwerktreiber
  - Completion-Routine auf höheren Schichten (Call Back Routine)



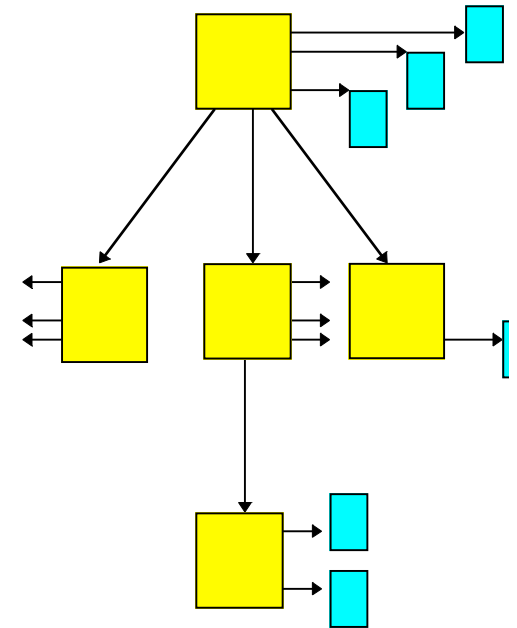
- Interrupt-Latenz durch nicht unterbrechbare Prozesse
- Interruptsperr, z.B. UNIX-Kern

- Steuerung der Prozesse
  - besonderes Steuerprogramme ('Shell')
  - kennt Scheduler
  - Starten eines Programmes => neuer Prozeß
  - Zwangs-Beenden entfernt Prozeß
  - besonders bekannt bei UNIX bzw. DOS
- Grafische Benutzeroberflächen für Steuerprogramm
  - Menubefehl 'Open'
  - Doppelklick als Abkürzung
  - Windows: Alt-Tab wechselt zwischen Programmen
  - Cntl-Alt-Del (Strg-Alt-Entf) zeigt Prozeßliste
- Eingebaute Kommandos
  - Prozessliste anzeigen
  - Geräteverwaltung
  - auch Dateiverwaltung auf der Festplatte

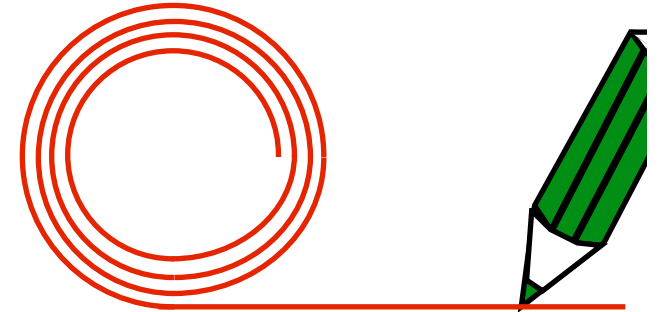


## 5.1.2 Verwaltung des Plattenplatzes

- Zentrale Abstraktion: Datei
  - Menge von Bytes auf der Platte
  - wird vom Betriebssystem zusammengehalten
  - für Programme und Daten
  - Daten: Briefe, Tabellen, Bilder, Datenbank, ...
- Verzeichnisse strukturieren Dateimenge
  - Verzeichnis enthält Dateien
  - Verzeichnis kann andere Verzeichnisse enthalten
  - Wurzelverzeichnis
- Pfad
  - Wurzel/Verzeichnis/Verzeichnis/Verzeichnis/Datei
  - c:\benutzer\froitzheim\daten\vorlesung.doc

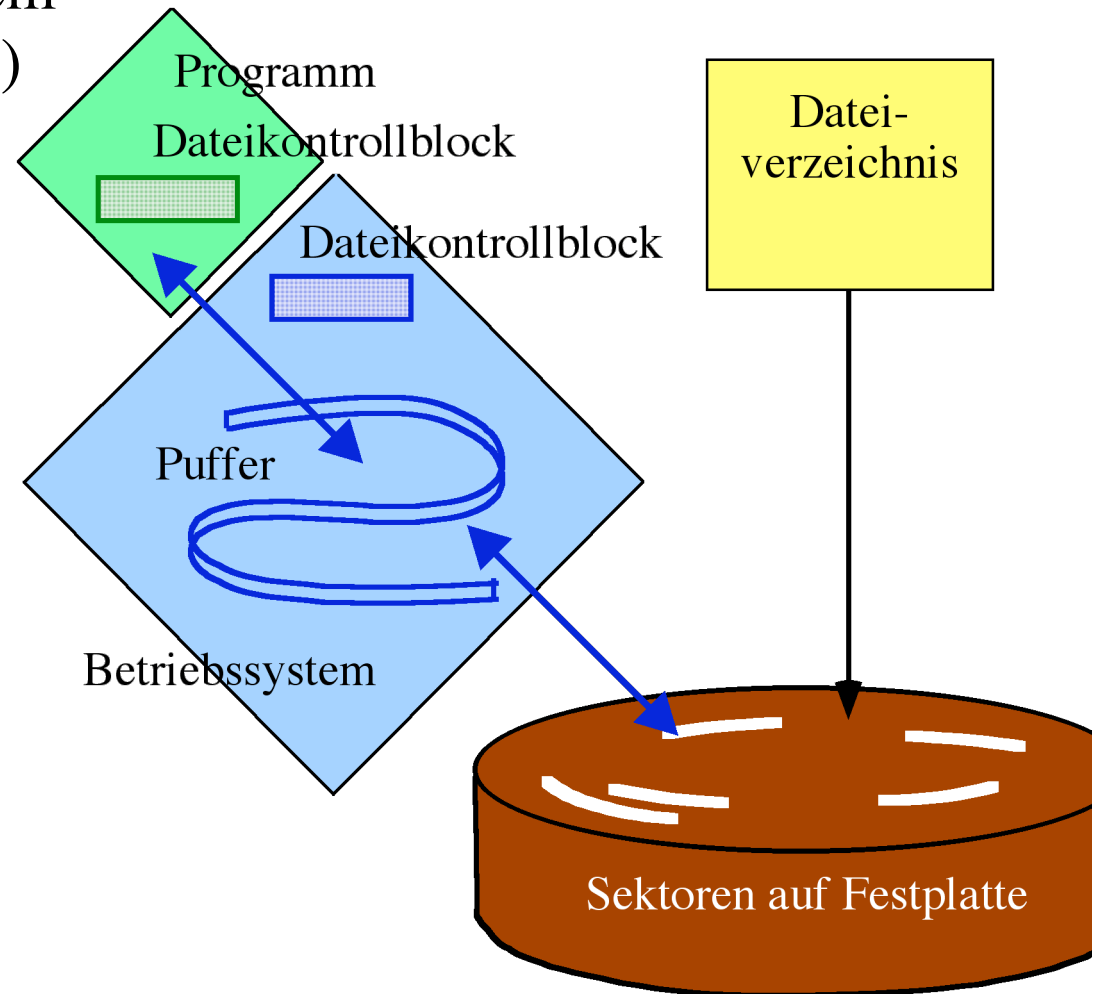


- Historisch gesehen ein Magnetband
  - mit einem Schreib-/Lesekopf
  - Datenblöcke sequentiell aneinanderreihen
  - wahlfreie Zugriffe teuer bzw. langsam
  - Einfügeoperationen sehr teuer
  - beinahe beliebiges Größenwachstum
- Aus der Sicht der Programmiersprache
  - Von Dateimodulen exportierter Typ "File"
  - sequentiell Lesen und Schreiben
  - Öffnen und Schliessen, Zugriffsrechte
- Aus der Sicht des Betriebssystems
  - benanntes Objekt ("WS1996PI.TXT")
  - Dateikontrollblock mit Puffer im RAM
  - Assoziation einer Dateivariablen mit externer Datenregion
  - Abbildung:  
blockweise adressierte Objekte auf Platte  
=> byteweise adressierte Objekte im Hauptspeicher



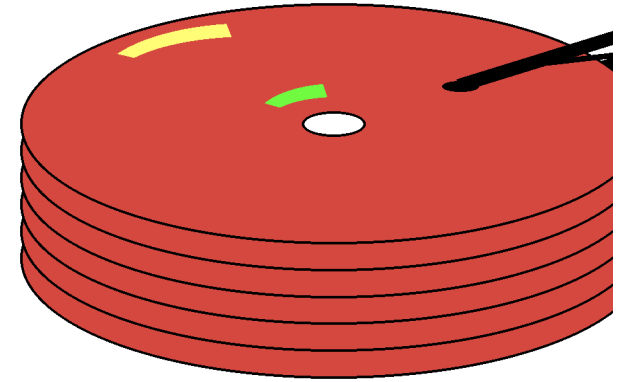
- Implementierung

- Programm mit Zugriffsroutinen
- Treiberrouinen im Betriebssystem
- Dateivariable im Programm (file)
- Dateikontrollblock (User & System)
- Pufferbereich im Hauptspeicher
- Dateiverzeichnis für Festplatte
- Sektorzuordnung auf Festplatte



- Plattenformat

- $n$  Platten auf einer Spindel (z.B. 5)
- $2n$  Oberflächen  $\rightarrow$   $2n$  Köpfe
- viele Spuren pro Platte (z.B. 40.000)
- Zylinder: Menge von Spuren im vertikalen Schnitt
- viele Sektoren pro Spur (z.B. 1000 - 6000)



- Mehrere Festplatten-Partitionen

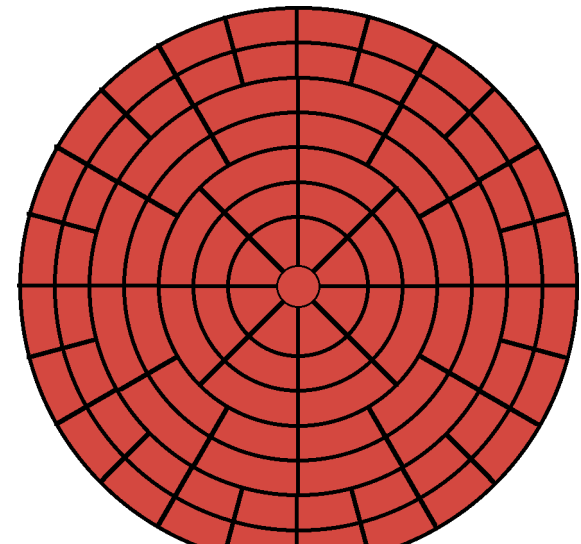
- $m$  Gbytes pro Platte
- $d$  Dateinamen pro Platte
- $s$  Dateistücke pro Platte
- bessere Nutzung für kleine Dateien ...

- Verschiedene Datei- & Betriebssysteme auf verschiedenen Partitionen

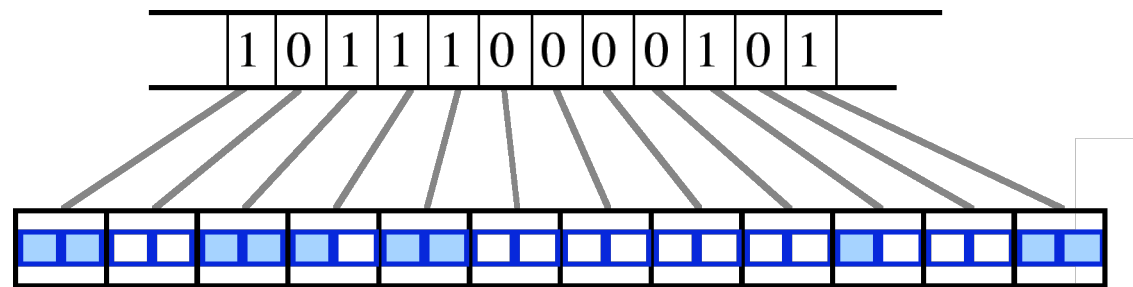
- MacOS, MS-DOS
- Unix, Linux
- Windows (NTFS)

- Umschalten zwischen Betriebssystemen

- Parameter RAM
- Bootmanager



- Plattenspeichervergabe: Wo ist noch Plattenplatz frei?
  - Absuchen der FAT
  - kompakter mit Allokierungs-Bittabelle
  - Bitplätze entsprechen Platz auf der Platte

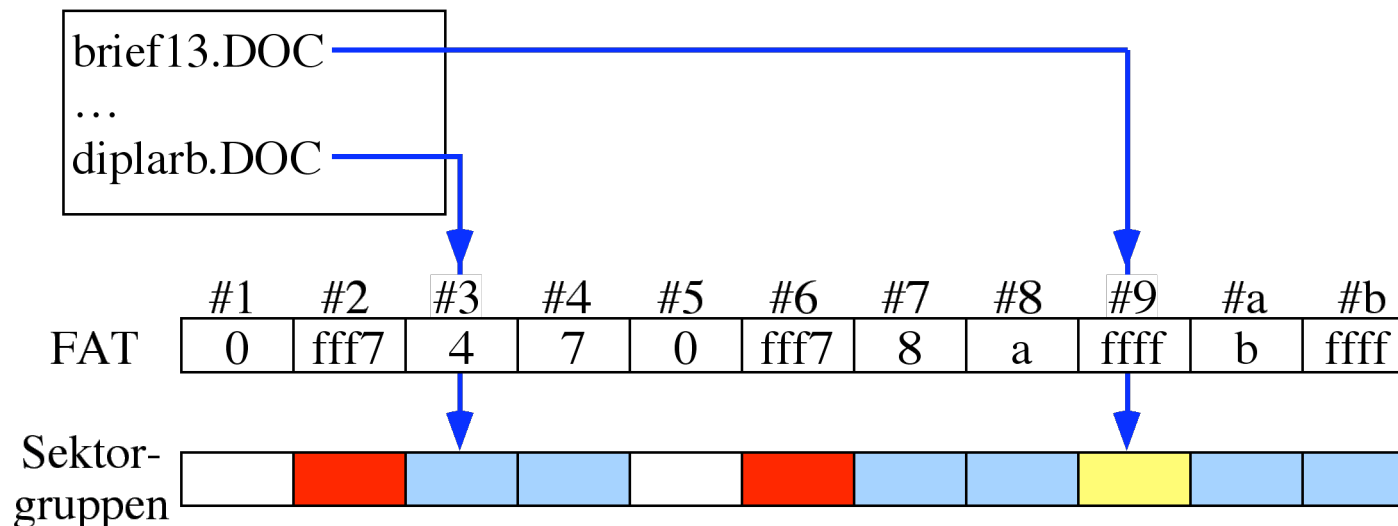


- Enthält ein Bit für jede Belegungseinheit (Allocation Block)
  - teilweise im Hauptspeicher
  - gesetzt, falls der Block belegt ist
  - z.B. 65536 Bit in der Tabelle
  - ab 64 Mbyte mehr als 2 log. Blöcke pro Bit
- Große Platten
  - viele Blöcke pro Eintrag (Cluster)
  - größere Bitmap
- Keine Aussage über Dateizugehörigkeit
- Sehr platzsparend



- File Allocation Table (FAT)

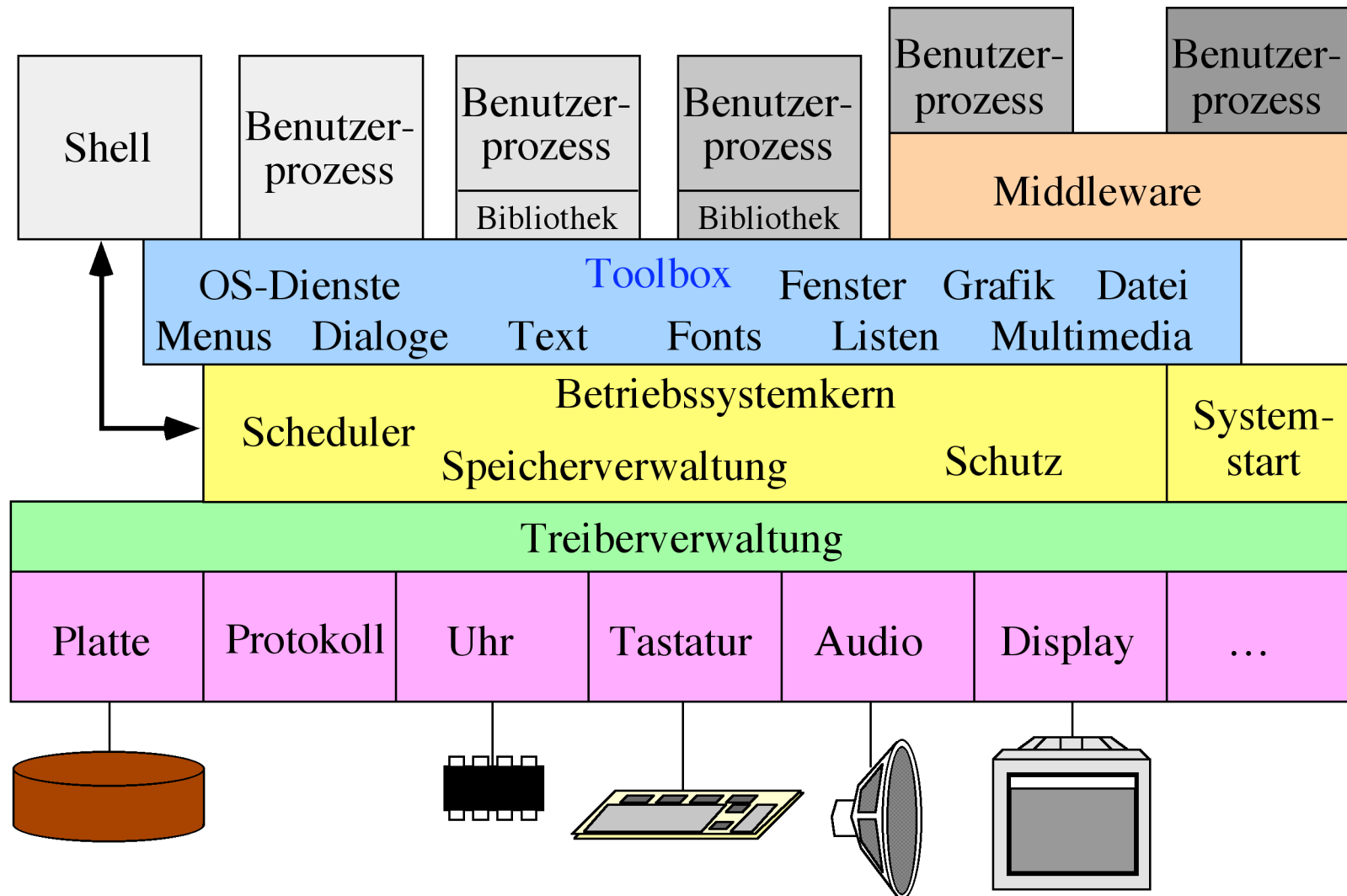
- Dateizuordnungstabelle auf den ersten Spuren einer Platte
- Belegungseinheiten (Blöcke) der Datei über FAT verkettet
- der letzte Block ist mit \$FFFF markiert
- schadhafte Blöcke sind mit \$FFF7 markiert



- Verzeichnis zeigt auf den ersten Block einer Datei
- Andere Dateisysteme
  - Windows NTFS komplexer
  - UNIX-NFS netzwerkfähig
  - Macintosh mit Dateityp und zweiteiligen Dateien
- Virtual File System als Abstraktion

## 5.2 Struktureller Aufbau eines Betriebssystems

- Konzept: Schichtenstruktur
  - auch Seitenpfade
  - Interfaces an der Oberkante jeder Schicht



## 5.3 Toolbox

- Abstrakter Zugriff auf Hardware
  - Präsentation: Bildschirm, Drucker, Audio
  - permanenter Speicher: Festplatte, Diskette, CD/DVD
  - Systemressourcen: Uhr, FPU, Sensoren, ...
  - Kommunikation: V.24, Centronics, SCSI, Ethernet, USB, SATA
  - Software: Dateien, Fonts, Menus, Fenster, ...
- Verbergen von:
  - Speicheradressen
  - Registern
  - Kommandos
- Benutzung fertiger Komponenten
- Austausch der Implementierung
  - Beschleunigung
  - neuer Standard
  - Portabilität (WindowsNT auf IA, PowerPC, Alpha, ...)
  - anderes Netzwerk (Bsp: Ethernet statt Token Ring)

## 5.3.1 Beispiel Grafik in einem Fenster

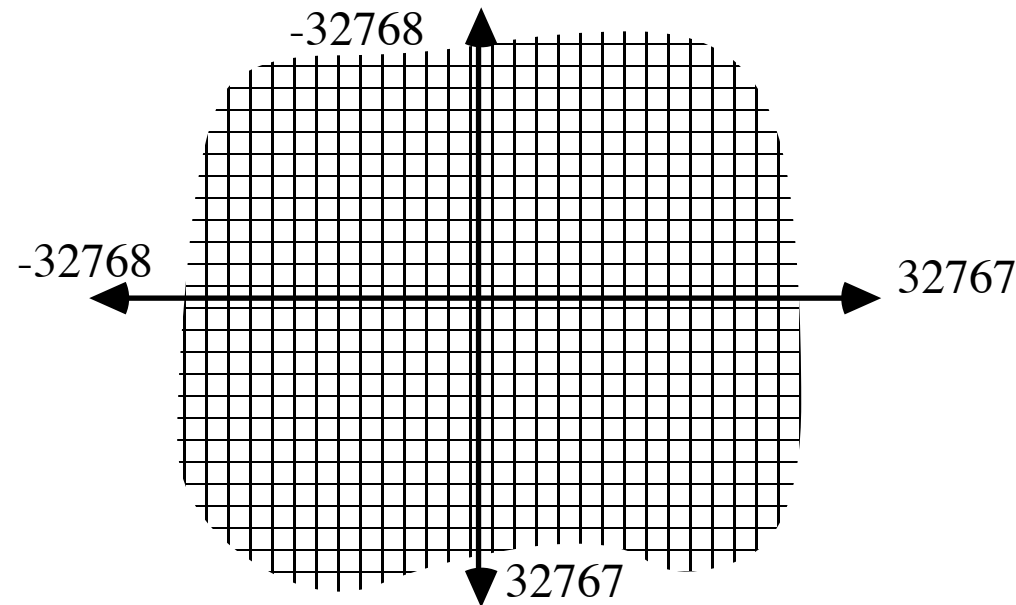
- Rasterbildschirm

- jeder Punkt einzeln ansprechbar
- uneingeschränkt grafikfähig

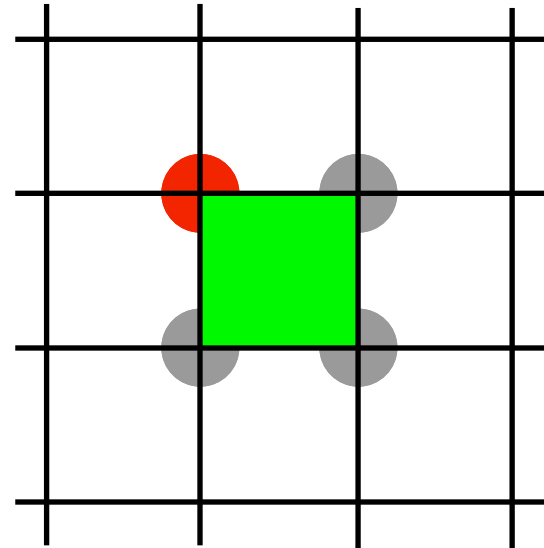
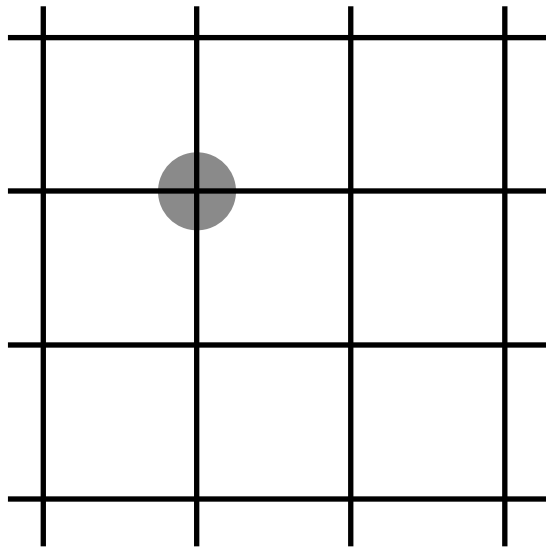


- Punkteanzahl typisch 1024\*768 bis 2560\*1600
- 72, 80 bis 100 Punkte/Zoll (dpi)
- 2560\*1600\*24 bit für 30" Farbmonitor -> 12.188.000 Byte
- Bildänderungsrate (Framerate,  $\geq 25$  Hz) und Bildwiederholrate ( $> 70$  Hz)

- Koordinatensystem
  - Cartesische Koordinaten: x-Achse, y-Achse
  - andere Koordinatensysteme möglich,
  - wenig genutzt für Computergrafik
- Koordinatenebene
  - unendlich dünne Koordinatenlinien,
  - endlicher Koordinatenbereich,
  - keine feste Zuordnung zum Schirm.

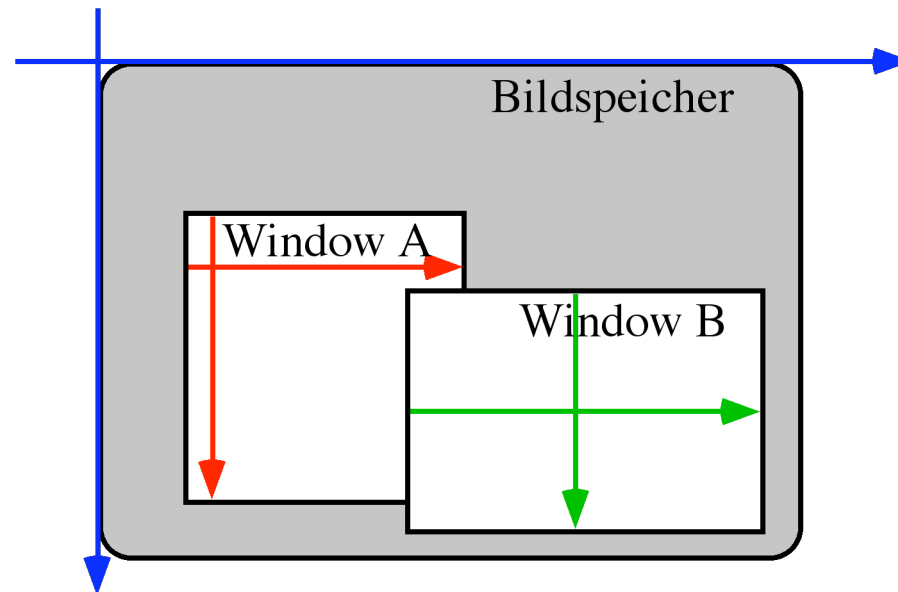


- Koordinaten-Punkt als Kreuzungspunkt von zwei Koordinatenlinien:
  - als geometrische Abstraktion, nicht verwechseln mit Bild-Punkt
  - z.B.  $2^{32}$  Punkte in einer Koordinatenebene
  - unendlich klein



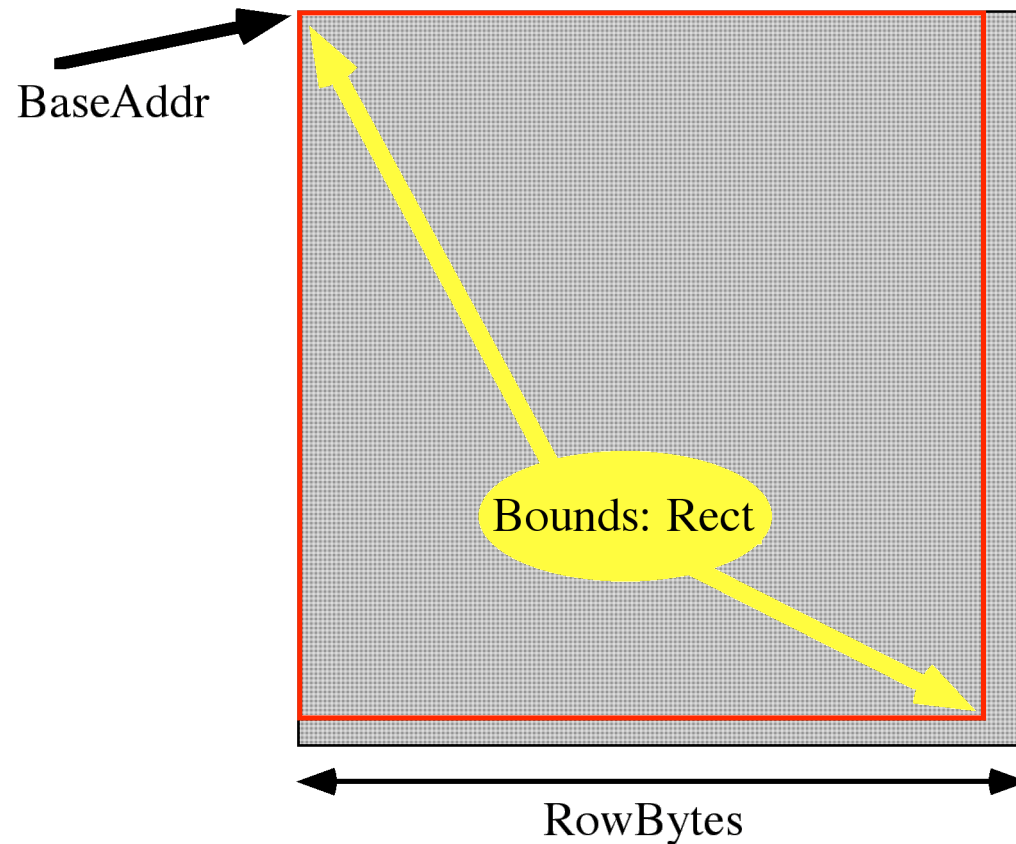
- Bild-Punkt:
  - Als Pixel in einer Bitmap (PixelMap)
  - Begrenzt durch vier Koordinatenpunkte
  - Position rechts unter dem Koordinatenpunkt

- Drawable (Window bzw. Pixmap)
  - Speicherbereich für Bildpunkte (Pixel)
  - insbesondere der Bildschirm
  - Zeichnen verändert Pixel im Drawable
  - globales Koordinatensystem
  - eigene lokale Koordinatensysteme für Zeichenbereiche von Prozessen



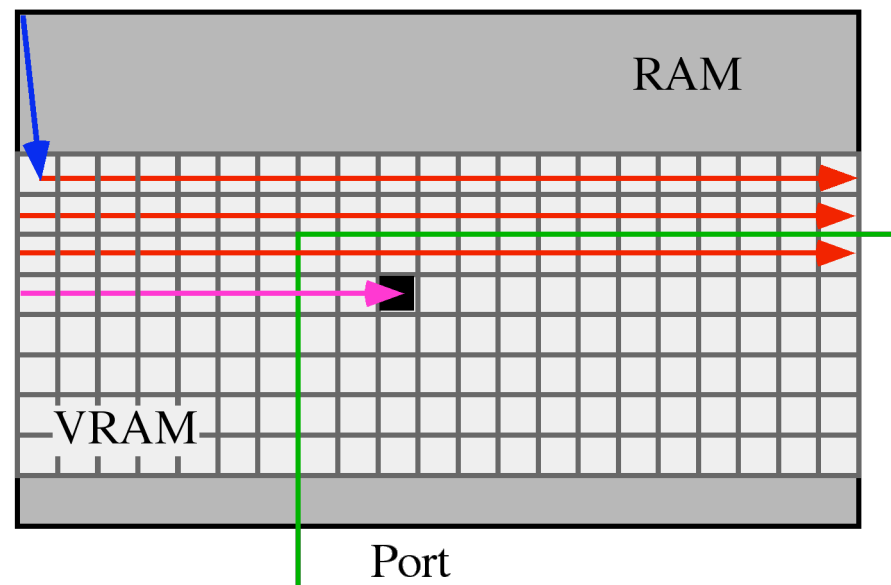
- Drawable -> Window -> Visual
- Framebuffer
- Window-Manager koordiniert überlappende Fenster

- Datenstruktur eines Drawable
  - Basisadresse im (Video-)RAM
  - Zeilenstruktur einer Bildpunkt-Menge
  - Lage der Bildpunkte relativ zum lokalen Koordinatensystem





- Zeichnen eines Pixels mit den Koordinaten (x,y)
  1. (x,y) mit bounds.topleft -> Koordinaten für PixMap: (pixx, pixy)  
`pixx = x + bounds.left; pixy = y + bounds.top;`
  2. Schreibadresse um pixy Zeilen verschieben:  
`VRAMAddr = pixy * rowBytes`
  3. Schreibadresse um pixx Punkte verschieben:  
`VRAMAddr = VRAMAddr + pixx * bit_pro_Pixel / 8`
  4. Basisadresse addieren: `RAMAddr = VRAMAddr + basAddr`
  5. Pixel in Bildwiederholtspeicher schreiben: `*RAMAddr=pixelValue`



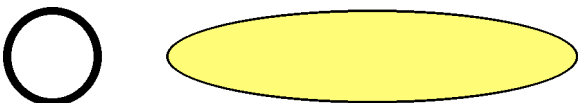
- Geometrische Figuren

- Strecken: 

- Rechtecke: 

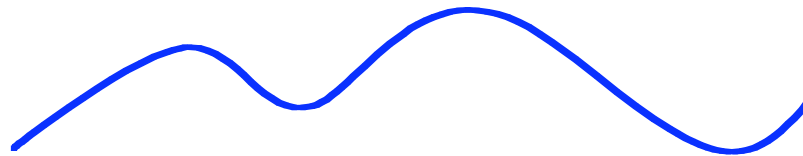
- Rechtecke mit abgerundeten Ecken: 

- Polygone und geglättete Polygone: 

- Kreise und Ellipsen: 

- Polygone  $\neq$  Rechtecke

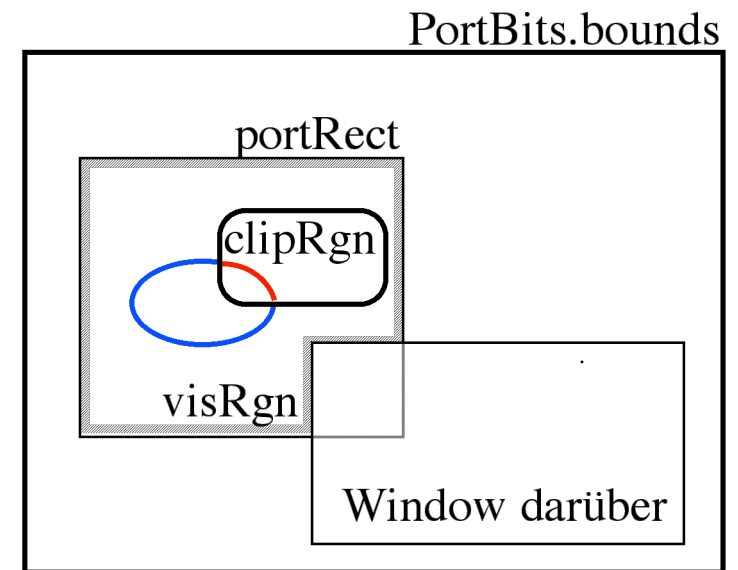
- Kurven



- Begrenzte Anzahl Primitive vs. Semantikerhaltung

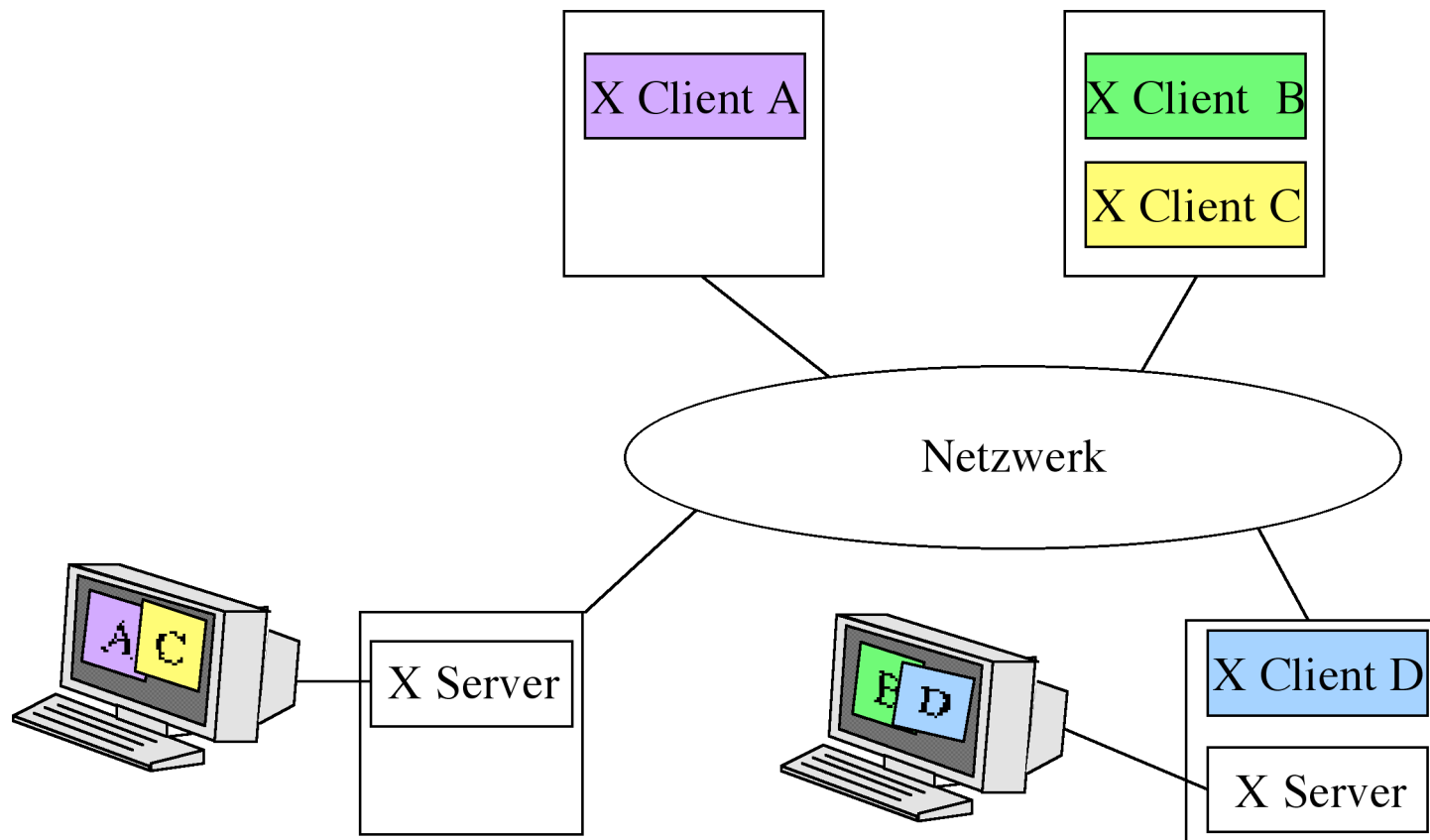
- > effiziente Implementierung

- Grafikkontext GC
  - meist einem Fenster zugeordnet
- Enthält Zustand des Zeichenzieles:
  - Stiftdicke, -farbe, -muster, und -position,
  - Füllfarbe und -muster,
  - Textattribute (Font, Typeface, Größe, ...)
- Clipping
  - Zeichenoperationen nur im nicht überdeckten Bereich eines Fensters
  - Operationen außerhalb dieses Bereiches werden 'weggeschnitten'
  - VisRgn zeigt die nicht abgedeckten Teile
  - ClipRgn zeigt logischen Zeichnungsbereich

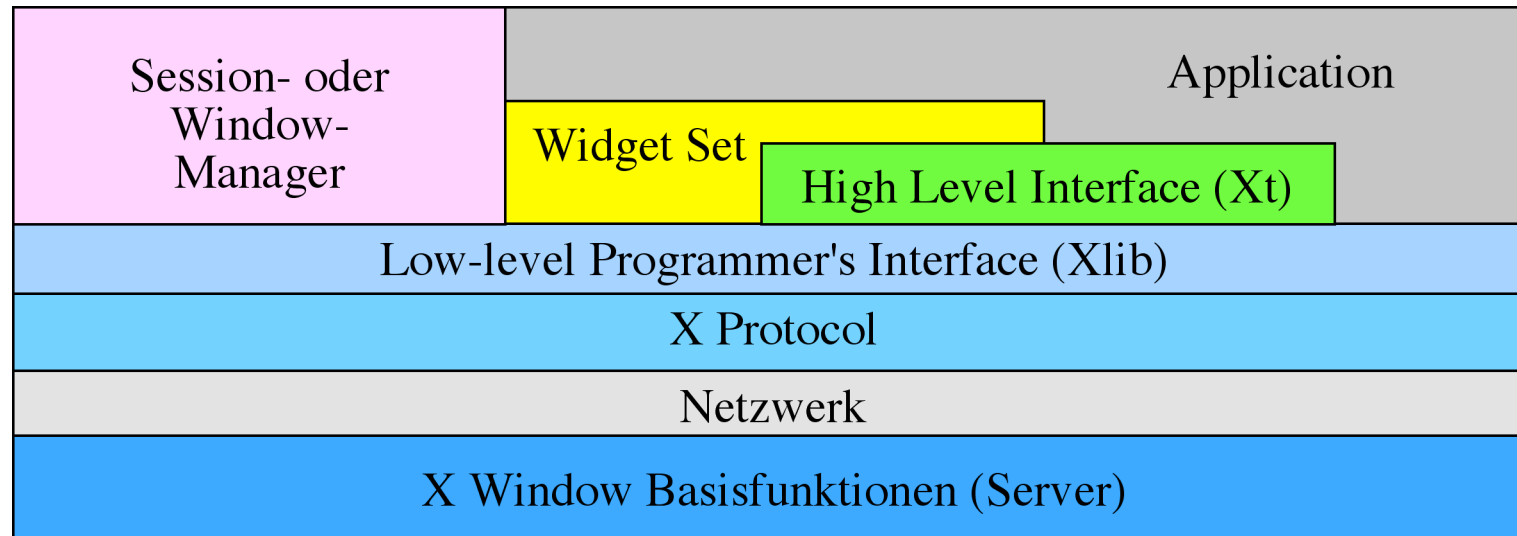


### 5.3.1.1 X-Window System

- Clients und Server (Vorsicht, die Bezeichnungen sind 'verdreht')
  - X-Client ist die Applikation
  - X-Server ist das 'Terminal', schreibt in Bildwiederholtspeicher
    - passive\_open auf Port 6000+display



- X Architektur



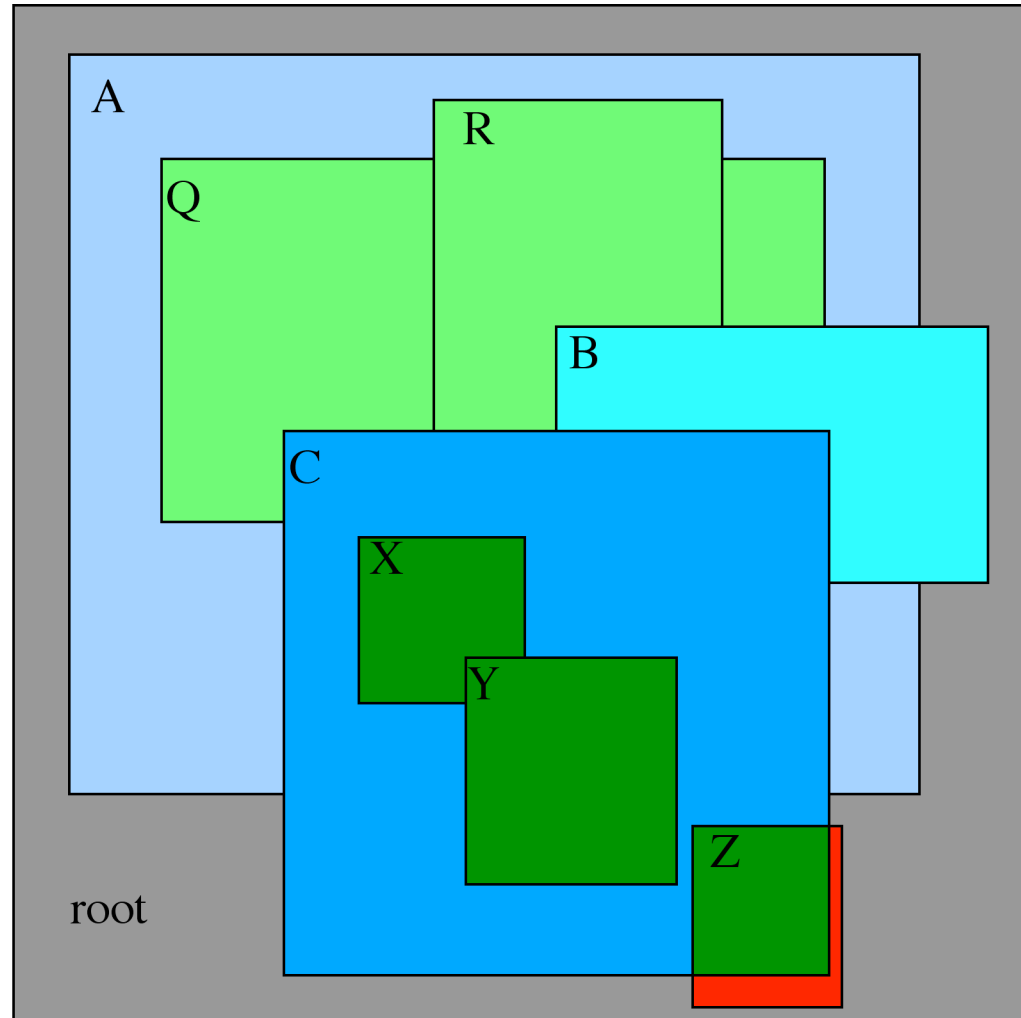
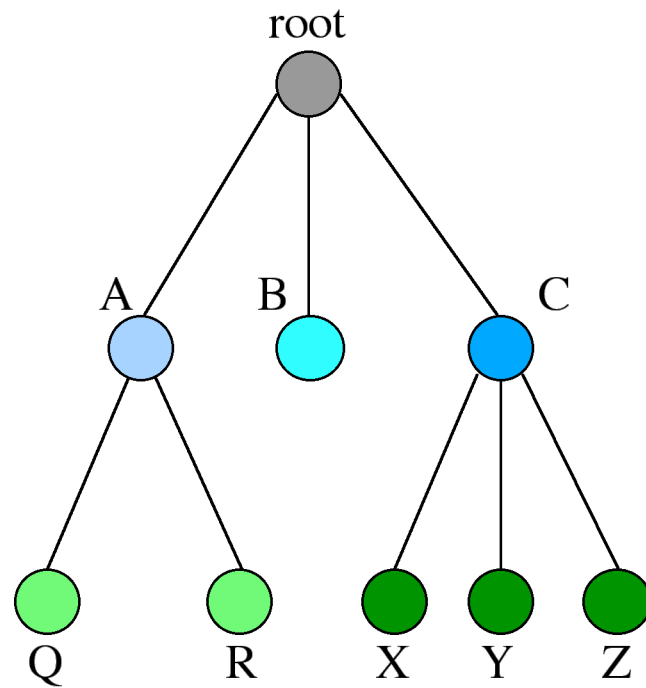
- Mechanisms

- X Protocol : X-Befehle, Kodierungsvorschrift
- Xlib : Application Programmers IF für X-Protocol
- Xt Intrinsics : Rahmen für Widgets

- Policy (look and feel)

- Window Mngr. : Fensteranordnung, Clipping, Fensterdekoration, UI für Fenstermanagement
- Widgets : Buttons, Menus, Scrollbars, Dialogboxen, ...

- Fensterhierarchie



- Stacking Order bestimmt das Abdecken durch 'Geschwister'

- Referenz z.B. <http://tronche.com/gui/x/>
- Display
  - ein Server kontrolliert ein Display
  - zum Display gehören Screen(s), Tastatur und Zeiger (Maus)
  - Name eines Displays: <hostname>:<DispNr>.<ScreenNr>
  - Beispiel: `xclock -display kuckuck.informatik.tu-freiberg.de:0.0`
- Drawable
  - Window, Pixmap
  - Ort für Ausführung von Zeichenoperationen
- Grafikkontext (GC)
  - XGCValues-Datenstruktur:  
function, foreground, background, line\_width, line\_style,  
font, fill\_style, ...  
`myGC = XCreateGC(display, drawable, mask, &values);`  
...  
`XChangeGC(display, myGC, mask, &values);`  
Nicht alle Felder besetzt: `mask = GCForeground | GCLineStyle;`
  - Spezialfunktionen zur Veränderung einzelner Felder:  
`XSetFont(display, myGC, emphasisFont);`

- X Funktionen (Auswahl)

- Verwaltungsarbeit:

Display öffnen: `myDisp = XOpenDisplay(theName);`

Window erzeugen:

`myWin = XCreateWindow(myDisp, parent, x, y, width, hgt, ...);`

Window sichtbar machen: `XMapRaised(myDisp, myWin);`

Weitere Funktionen um Windows und Displays zu schließen etc.

- Grafische Objekte

`XDrawPoint(myDisp, myWin, myGC, x, y);`

`XDrawLine(myDisp, myWin, myGC, x1, y1, x2, y2);`

`XFillRectangle(myDisp, myWin, myGC, x, y, width, height);`

`XDrawArc(myDisp, myWin, myGC, x, y, width, height,  
angle1, angle2);`

Randbedingungen in myGC

- Text

Text anzeigen: `XDrawString(myDisp, myWin, myGC, x, y, str, len);`

Font in den Server laden: `myFnt=XLoadFont(myDisp, fontName);`

`/* Do not forget XSetFont() */`

Eigenschaften des Fonts: `theFntDesc=XQueryFont(myDisp, myFnt);`



- Events

- Ereignisse am Server werden dem Klienten gemeldet
- Zustandsveränderungen
- Taste, Maus, Expose
- beziehen sich auf ein Window
- Warteschlange
- Beispiele:

`ButtonPress, ButtonRelease, KeyPress,  
MotionNotify, EnterNotify, LeaveNotify,  
Expose, ...`

- Datenstruktur XEvent

- gemeinsamer Teil für alle Events: XAnyEvent-Record  
type, display, window, ...

`event.xany.window`

- eventspezifischer Teil; Beispiel XKeyEvent-Record

`time, x, y, state, keycode, ...`

`theChar = XLookupString(event.xkey.keycode);`

- Anzahl wartende Expose-Events: `event.xexpose.count`

- Verarbeitung von Events

- Klienten verlangen Events:

`XSelectInput(display, window, Evt1Mask | Evt2Mask | ... | EvtNMask) ;`

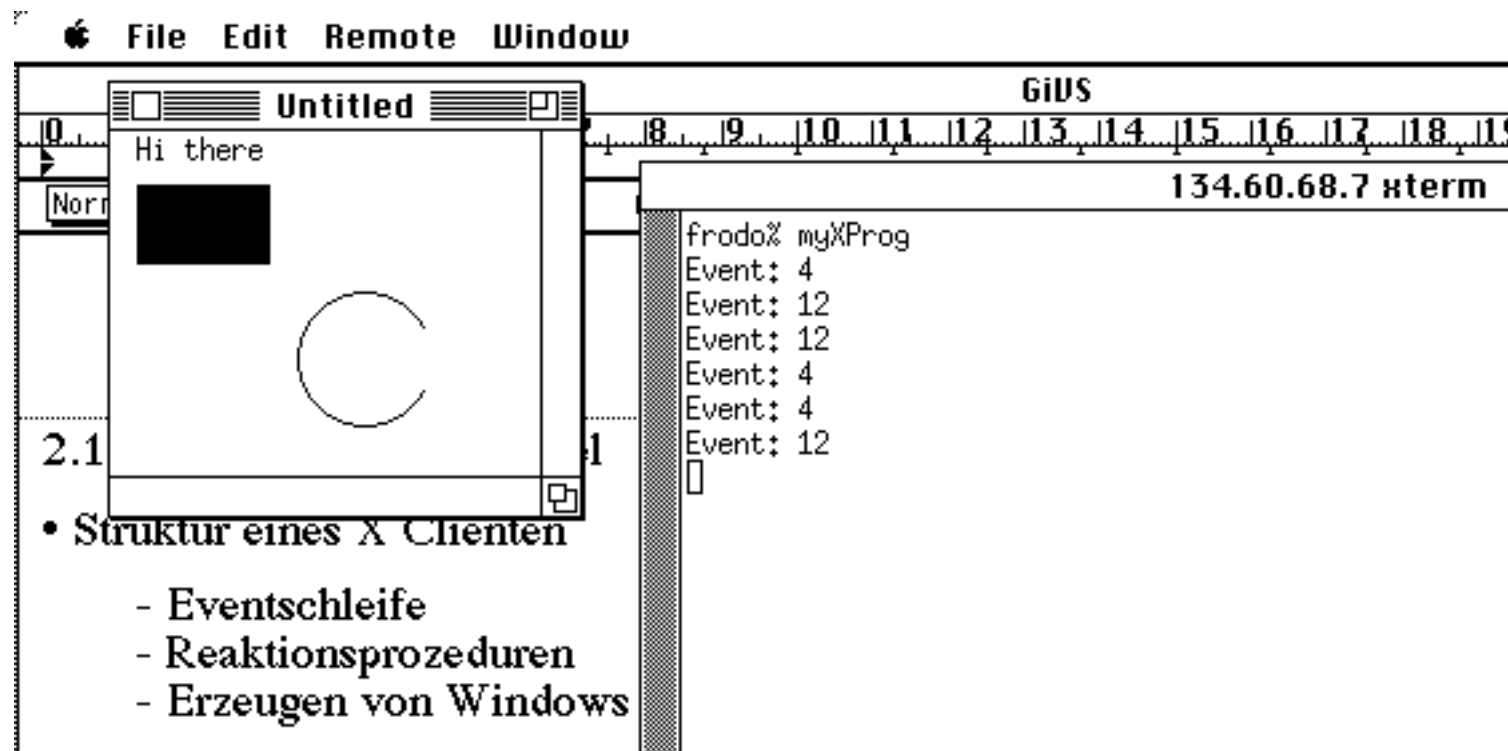
- Maskenkonstanten:

`KeyPressMask, ButtonPressMask,  
PointerMotionMask, KeyPressMask,  
KeyReleaseMask, ExposureMask,  
EnterWindowMask, LeaveWindowmask, ...`

- switch Statement

## 5.3.1.2 Programmierbeispiel X-Window

- Struktur eines X Clienten
  - Eventschleife
  - Reaktionsprozeduren
  - Erzeugen von Windows etc.



- Hauptschleife fragt Events ab und reagiert entsprechend

```
void Loop4Events ()
{
 XEvent theEvent;
 quit = False;
 XSelectInput(myDisp,myWin,(ButtonPressMask|KeyPressMask|
 ExposureMask));

 while (quit == False)
 {
 XNextEvent(myDisp, &theEvent);
 if (theEvent.xany.window==myWin)
 switch (theEvent.type)
 {
 case ButtonPress : ReDraw(); break;
 case KeyPress : quit = True; break;
 case Expose : ReDraw(); break;
 }
 }
}
```

- Reaktion auf ExposeEvent, Zeichnen des gesamten Fensters

```
void ReDraw ()
{
 XSetForeground(myDisp, myGC, WhitePixel(myDisp, myScreen));
 XFillRectangle(myDisp, myWin, myGC,0,0,150,120);
 XSetForeground(myDisp, myGC, BlackPixel(myDisp, myScreen));
 XDrawString(myDisp,myWin, myGC,10,10,"Hi there",8);
 XFillRectangle(myDisp, myWin, myGC,10,20,50,30);
 XDrawArc(myDisp, myWin, myGC,70,60,50,50,30*64,300*64);
}
```

- Hilfsfunktion zum Erzeugen eines einfachen Fensters

```
Window MakeWindow(char *name, int x, int y, int w, int h)
{
 Window aWin;
 unsigned long black, white;
 XGCValues gcVals;

 aWin = NULL;
 myDisp = XOpenDisplay(name);
 myScreen = DefaultScreen(myDisp);
 myParent = RootWindow(myDisp, myScreen);
 black = BlackPixel(myDisp, myScreen);
 white = WhitePixel(myDisp, myScreen);

 aWin =
 XCreateSimpleWindow(myDisp, myParent, x, y, w, h, 0, black, white);
 if (aWin != NULL)
 {
 XMapRaised(myDisp, aWin);
 gcVals.foreground = black;
 gcVals.background = white;
 myGC = XCreateGC(myDisp, aWin,
 (GCForeground | GCBackground), &gcVals);
 }
 return(aWin);
}
```

- Programmrahmen

```
#include <X11/Xlib.h>
```

```
#include <stdio.h>
```

```
Display *myDisp;
```

```
Window myWin, myParent;
```

```
int myScreen;
```

```
GC myGC;
```

```
Bool quit;
```

```
Window MakeWindow(name, x, y, w, h)
```

```
...
```

```
void ReDraw ()
```

```
...
```

```
void Loop4Events ()
```

```
...
```

```
main()
```

```
{
```

```
 myWin = MakeWindow("134.60.70.20:0.0", 100, 50, 150, 120);
```

```
 if (myWin != NULL) Loop4Events();
```

```
}
```