



# Übung zu Betriebssystembau

Ein- und Ausgabe

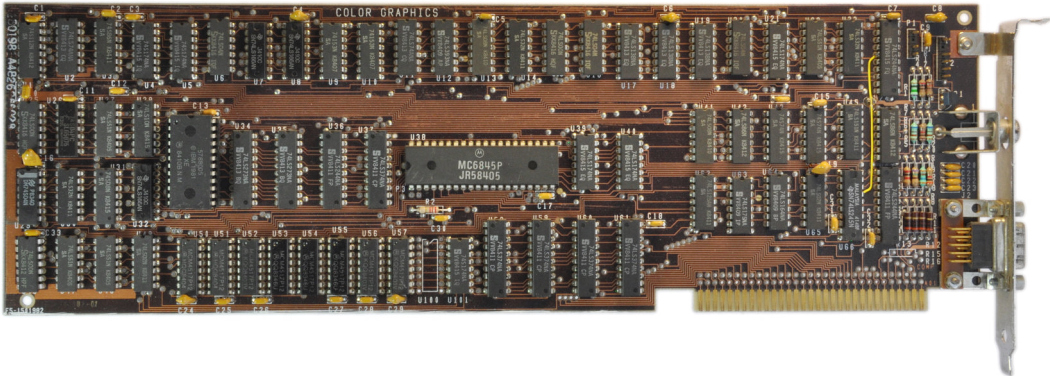
21. Oktober 2025

---

**Alexander Krause**

Arbeitsgruppe Systemsoftware  
Technische Universität Dortmund

(Mit Material vom Lehrstuhl 4 der FAU)



Quelle: Wikipedia

## Der CGA Bildschirm ...

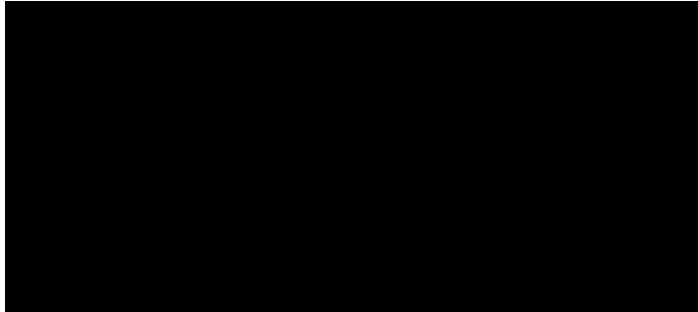


... zwei Bytes pro Zeichen ...

1. Byte: **ASCII Zeichen**

... zwei Bytes pro Zeichen ...

1. Byte: ASCII Zeichen

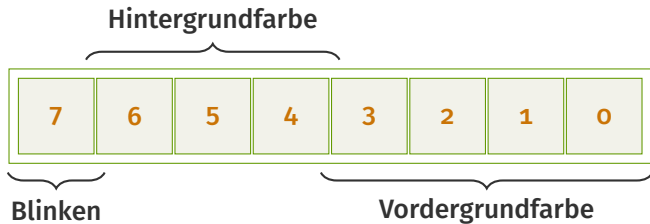


## ... zwei Bytes pro Zeichen ...

1. Byte: **ASCII Zeichen**
2. Byte: **Darstellungsattribut**

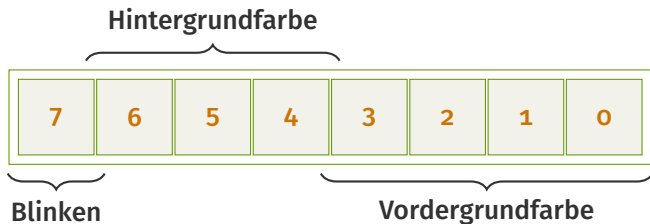
## ... zwei Bytes pro Zeichen ...

1. Byte: **ASCII Zeichen**
2. Byte: **Darstellungsattribut**



## ... zwei Bytes pro Zeichen ...

1. Byte: **ASCII Zeichen**
2. Byte: **Darstellungsattribut**



|   |         |   |          |    |            |    |             |
|---|---------|---|----------|----|------------|----|-------------|
| 0 | schwarz | 4 | rot      | 8  | dunkelgrau | 12 | hellrot     |
| 1 | blau    | 5 | magenta  | 9  | hellblau   | 13 | hellmagenta |
| 2 | grün    | 6 | braun    | 10 | hellgrün   | 14 | gelb        |
| 3 | cyan    | 7 | hellgrau | 11 | hellcyan   | 15 | weiß        |

**...eingebildet im Arbeitsspeicher ab 0xb8000**

|      |           |
|------|-----------|
| ⋮    |           |
| 'B'  | ← 0xb8000 |
| 0xf4 | ← 0xb8001 |
| 'a'  | ← 0xb8002 |
| 0x74 | ← 0xb8003 |
| 'r'  | ← 0xb8004 |
| 0x74 | ← 0xb8005 |
| ⋮    |           |

**Bar**

ar



|          |                                     |                  |
|----------|-------------------------------------|------------------|
| $\&$     | Konjunktion (bitweises UND)         | $3 \& 5 = 1$     |
| $ $      | Disjunktion (bitweises ODER)        | $3   5 = 7$      |
| $\wedge$ | exklusives ODER                     | $3 \wedge 5 = 6$ |
| $\sim$   | Einerkomplement (bitweise Negation) | $\sim 5 = 250$   |
| $\ll$    | verschieben nach links              | $12 \ll 2 = 48$  |
| $\gg$    | verschieben nach rechts             | $12 \gg 2 = 3$   |

# Rekapitulation: Bitshiftoperationen

|                                       |                                                    |                                 |
|---------------------------------------|----------------------------------------------------|---------------------------------|
| <code>&amp;</code>                    | Konjunktion (bitweises UND)                        | <code>3 &amp; 5 = 1</code>      |
| <code> </code>                        | Disjunktion (bitweises ODER)                       | <code>3   5 = 7</code>          |
| <code>^</code>                        | exklusives ODER                                    | <code>3 ^ 5 = 6</code>          |
| <code>~</code>                        | Einerkomplement (bitweise Negation)                | <code>~ 5 = 250</code>          |
| <code>&lt;&lt;</code>                 | verschieben nach links                             | <code>12 &lt;&lt; 2 = 48</code> |
| <code>&gt;&gt;</code>                 | verschieben nach rechts                            | <code>12 &gt;&gt; 2 = 3</code>  |
| <code>x  = (1 &lt;&lt; n)</code>      | Setze das <code>n</code> te Bit in <code>x</code>  |                                 |
| <code>x &amp;= ~(1 &lt;&lt; n)</code> | Lösche das <code>n</code> te Bit in <code>x</code> |                                 |



- Bitmasken und logischen Bitoperationen



- Bitmasken und logischen Bitoperationen
- Bitfelder in C/C++

```
1 struct cga_attrib {  
2     unsigned char fg : 4,  
3                 bg : 3,  
4                 bl : 1;  
5 };  
6  
7 struct cga_attrib a;  
8 a.fg = 4; // rot  
9 a.bg = 7; // hellgrau  
10 a.bl = 1; // blinken
```

# Ansprechen von einzelnen Bits

- Bitmasken und logischen Bitoperationen
- Bitfelder in C/C++

```
1 struct cga_attrib {  
2     unsigned char fg : 4,  
3                 bg : 3,  
4                 bl : 1;  
5 };  
6  
7 struct cga_attrib a;  
8 a.fg = 4; // rot  
9 a.bg = 7; // hellgrau  
10 a.bl = 1; // blinken
```

Layout mit GCC auf x86:



```
1 struct Test {  
2     char foo[3];  
3     int bar;  
4     bool baz;  
5 };  
6  
7 cout << sizeof(Test);
```



```
1 struct Test {  
2     char foo[3]; // 3 Byte  
3     int bar;     // 4 Byte  
4     bool baz;    // 1 Byte  
5 };  
6  
7 cout << sizeof(Test);
```



```
1 struct Test {  
2     char foo[3]; // 3 Byte  
3     int bar;     // 4 Byte  
4     bool baz;    // 1 Byte  
5 };  
6  
7 cout << sizeof(Test); // "12"
```

```
1 struct Test {  
2     char foo[3]; // 3 Byte  
3     int bar;     // 4 Byte  
4     bool baz;    // 1 Byte  
5 } __attribute__((packed));  
6  
7 cout << sizeof(Test); // "8"
```



```
1 struct Test {  
2     char foo[3]; // 3 Byte  
3     int bar;     // 4 Byte  
4     bool baz;    // 1 Byte  
5 } __attribute__((packed));  
6  
7 static_assert(sizeof(Test) == 8, "Test kaputt");
```



**Entweder Cursorposition in Software merken...**

**... oder Cursorposition aus Hardware auslesen**

## ... oder Cursorposition aus Hardware auslesen

- CGA hat 18 Steuerregister mit je 8 Bit

|    |                       |    |
|----|-----------------------|----|
| 0  | Horizontal Total      | w  |
| 1  | Horizontal Displayed  | w  |
| 2  | H. Sync Position      | w  |
| 3  | H. Sync Width         | w  |
| 4  | Vertical Total        | w  |
| 5  | V. Total Adjust       | w  |
| 6  | Vertical Displayed    | w  |
| 7  | V. Sync Position      | w  |
| 8  | Interlace Mode        | w  |
| 9  | Max Scan Line Address | w  |
| 10 | Cursor Start          | w  |
| 11 | Cursor End            | w  |
| 12 | Start Address (high)  | w  |
| 13 | Start Address (low)   | w  |
| 14 | Cursor (high)         | rw |
| 15 | Cursor (low)          | rw |
| 16 | Light Pen (high)      | r  |
| 17 | Light Pen (low)       | r  |

## ... oder Cursorposition aus Hardware auslesen

- CGA hat 18 Steuerregister mit je 8 Bit
- Position in Register 14 (high) und 15 (low)

|    |                       |    |
|----|-----------------------|----|
| 0  | Horizontal Total      | w  |
| 1  | Horizontal Displayed  | w  |
| 2  | H. Sync Position      | w  |
| 3  | H. Sync Width         | w  |
| 4  | Vertical Total        | w  |
| 5  | V. Total Adjust       | w  |
| 6  | Vertical Displayed    | w  |
| 7  | V. Sync Position      | w  |
| 8  | Interlace Mode        | w  |
| 9  | Max Scan Line Address | w  |
| 10 | Cursor Start          | w  |
| 11 | Cursor End            | w  |
| 12 | Start Address (high)  | w  |
| 13 | Start Address (low)   | w  |
| 14 | Cursor (high)         | rw |
| 15 | Cursor (low)          | rw |
| 16 | Light Pen (high)      | r  |
| 17 | Light Pen (low)       | r  |

## ... oder Cursorposition aus Hardware auslesen

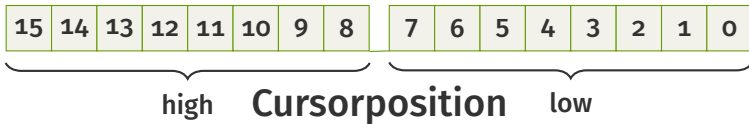
- CGA hat 18 Steuerregister mit je 8 Bit
- Position in Register 14 (high) und 15 (low)
- Nur indirekter Zugriff über Index- (0x3d4) und Datenregister (0x3d5)

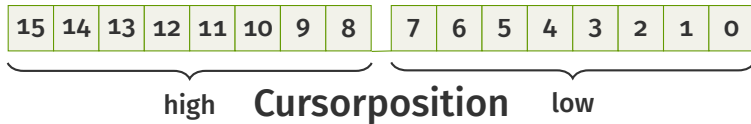
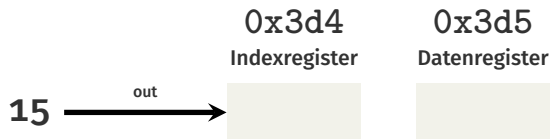
|    |                       |    |
|----|-----------------------|----|
| 0  | Horizontal Total      | w  |
| 1  | Horizontal Displayed  | w  |
| 2  | H. Sync Position      | w  |
| 3  | H. Sync Width         | w  |
| 4  | Vertical Total        | w  |
| 5  | V. Total Adjust       | w  |
| 6  | Vertical Displayed    | w  |
| 7  | V. Sync Position      | w  |
| 8  | Interlace Mode        | w  |
| 9  | Max Scan Line Address | w  |
| 10 | Cursor Start          | w  |
| 11 | Cursor End            | w  |
| 12 | Start Address (high)  | w  |
| 13 | Start Address (low)   | w  |
| 14 | Cursor (high)         | rw |
| 15 | Cursor (low)          | rw |
| 16 | Light Pen (high)      | r  |
| 17 | Light Pen (low)       | r  |

0x3d4  
Indexregister



0x3d5  
Datenregister





0x3d4  
Indexregister

15

0x3d5  
Datenregister

0x55

in



high

Cursorposition

low



0x3d4  
Indexregister

14

0x3d5  
Datenregister

0x0a

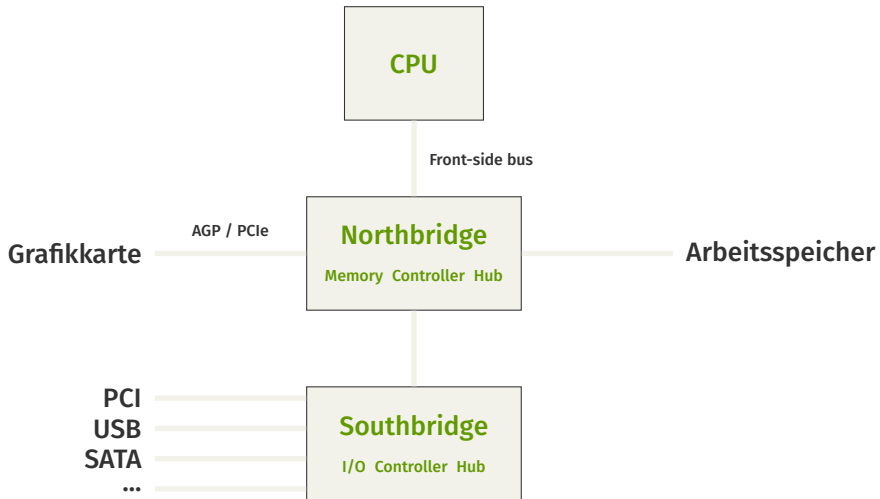
in

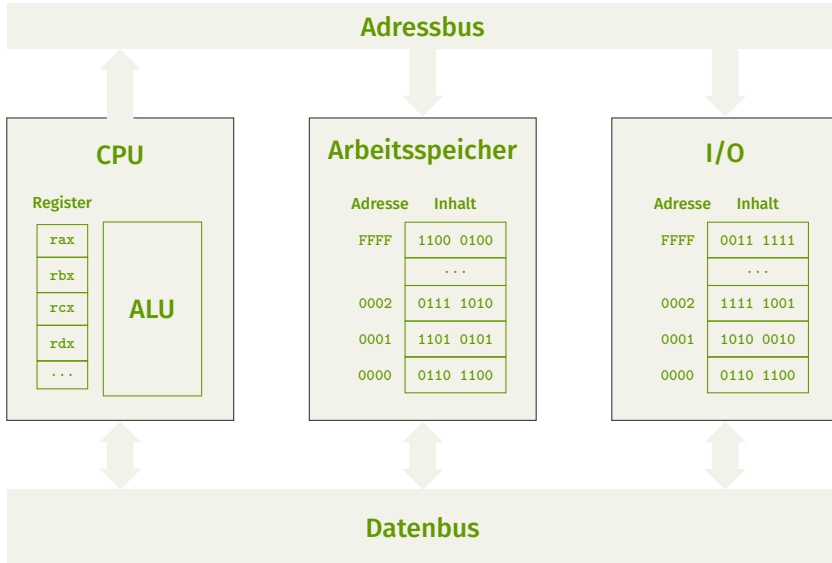


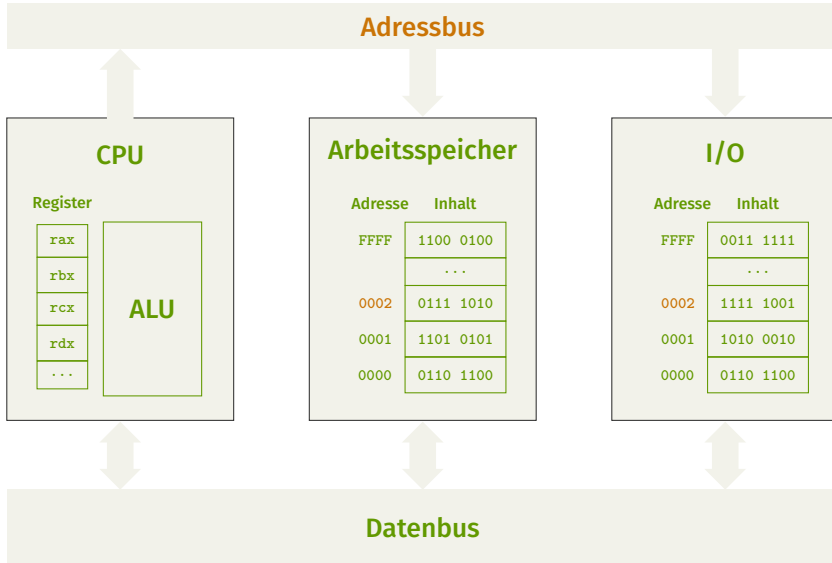
high

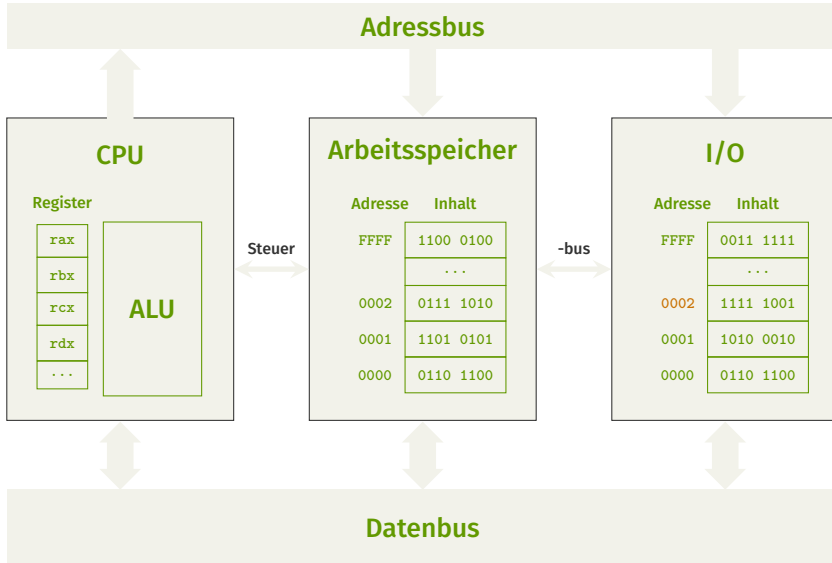
Cursorposition

low



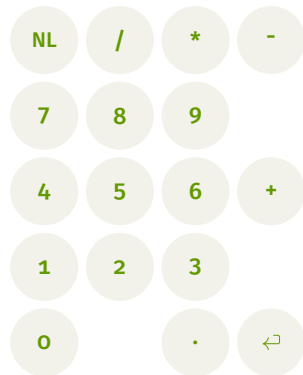


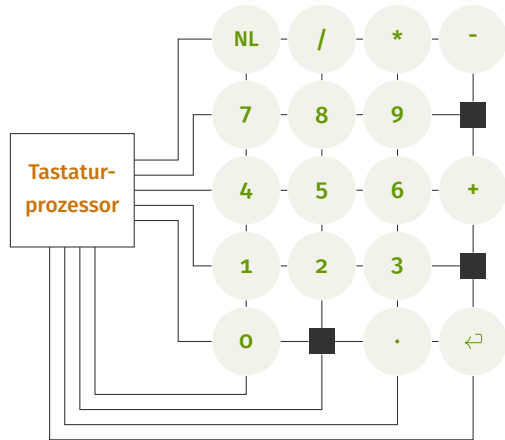


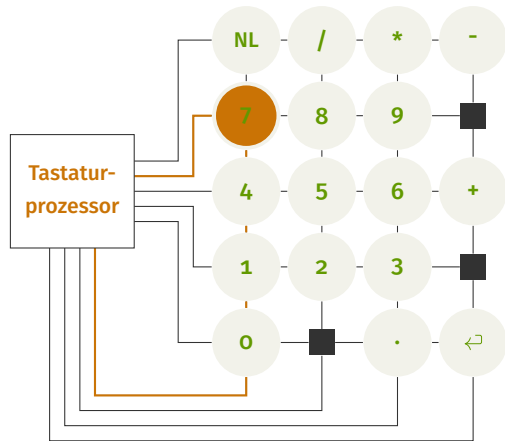


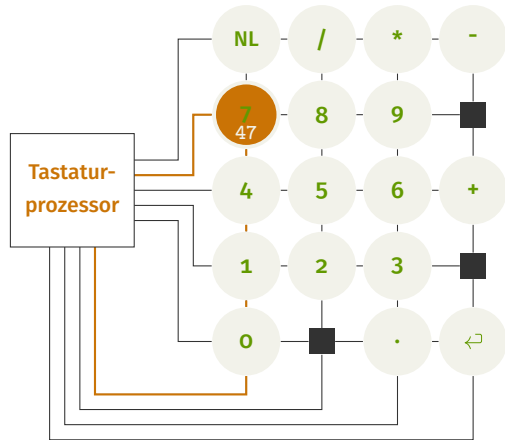


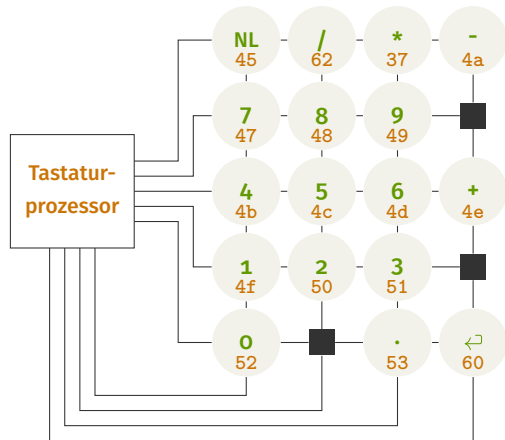
*Quelle: Golem*

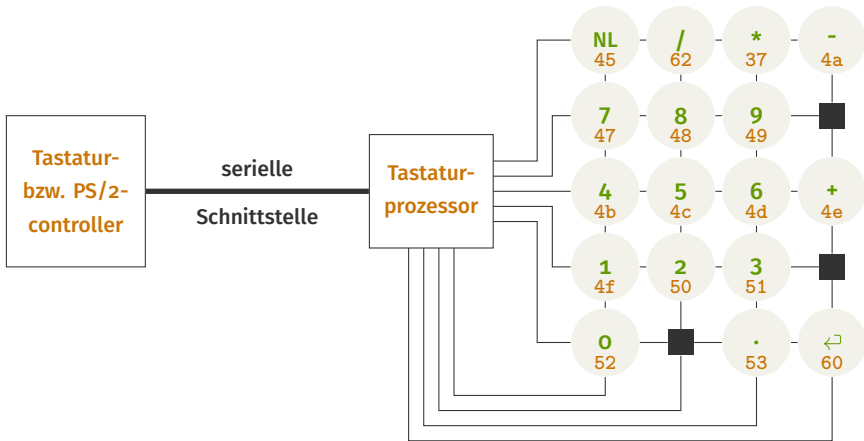


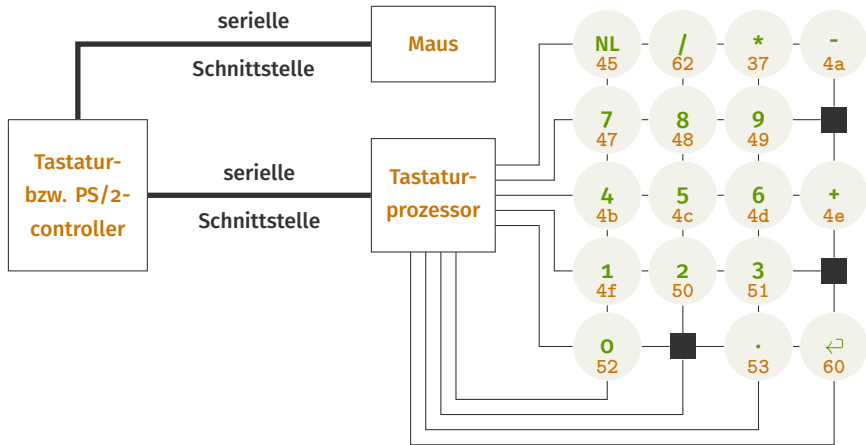












## Kommunikation mit

- Tastatur (primäres PS/2 Gerät)
- Maus (sekundäres PS/2 Gerät)
- PS/2 Controller (Konfiguration)

Kommunikation mit

- Tastatur (primäres PS/2 Gerät)
- Maus (sekundäres PS/2 Gerät)
- PS/2 Controller (Konfiguration)

über I/O Ports des PS/2 Controllers:

0x60 Datenregister

0x64 Kontrollregister

Kommunikation mit

- Tastatur (primäres PS/2 Gerät)
- Maus (sekundäres PS/2 Gerät)
- PS/2 Controller (Konfiguration)

über I/O Ports des PS/2 Controllers:

0x60 Datenregister

**lesen** Ausgabebuffer (des gewählten PS/2 Geräts)

**schreiben** Eingabebuffer (des gewählten PS/2 Geräts)

0x64 Kontrollregister

Kommunikation mit

- Tastatur (primäres PS/2 Gerät)
- Maus (sekundäres PS/2 Gerät)
- PS/2 Controller (Konfiguration)

über I/O Ports des PS/2 Controllers:

0x60 Datenregister

**lesen** Ausgabebuffer (des gewählten PS/2 Geräts)

**schreiben** Eingabebuffer (des gewählten PS/2 Geräts)

0x64 Kontrollregister

**lesen** Statusregister

**schreiben** Kommandoregister (des PS/2 Controllers)

**SCANCODE:** Tastenkennung, 7 Bit

**SCANCODE:** Tastenkennung, 7 Bit

**MAKECODE:** Tastendruck

**BREAKCODE:** Taste loslassen (Scancode + 0x80)

**SCANCODE:** Tastenkennung, 7 Bit

**MAKECODE:** Tastendruck

**BREAKCODE:** Taste loslassen (Scancode + 0x80)

**ASCII:** Darstellung von Zeichen, 8 Bit

**SCANCODE:** Tastenkennung, 7 Bit

**MAKECODE:** Tastendruck

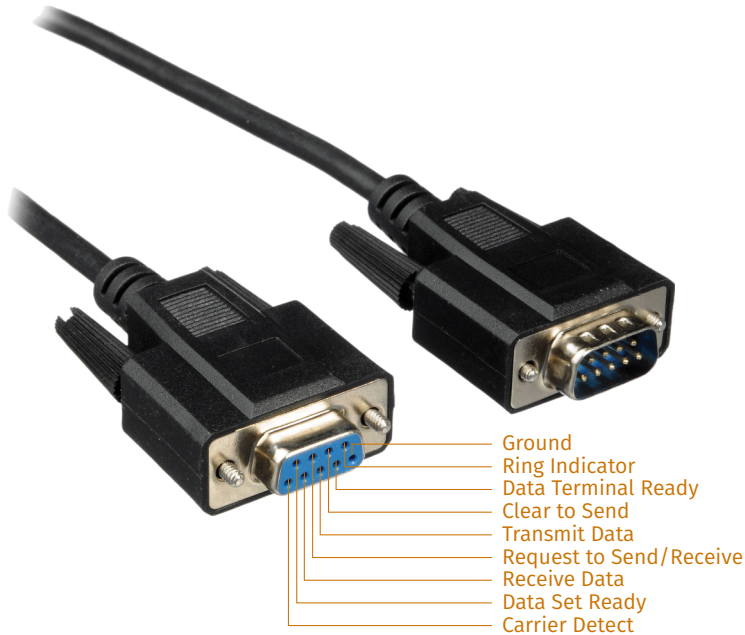
**BREAKCODE:** Taste loslassen (Scancode + 0x80)

**ASCII:** Darstellung von Zeichen, 8 Bit

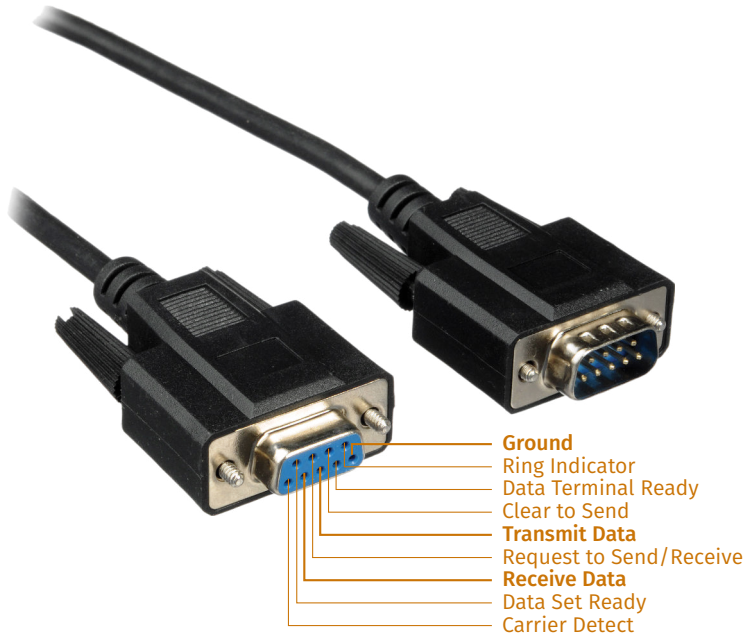
→ Umwandlung mittels (Software)Dekoder



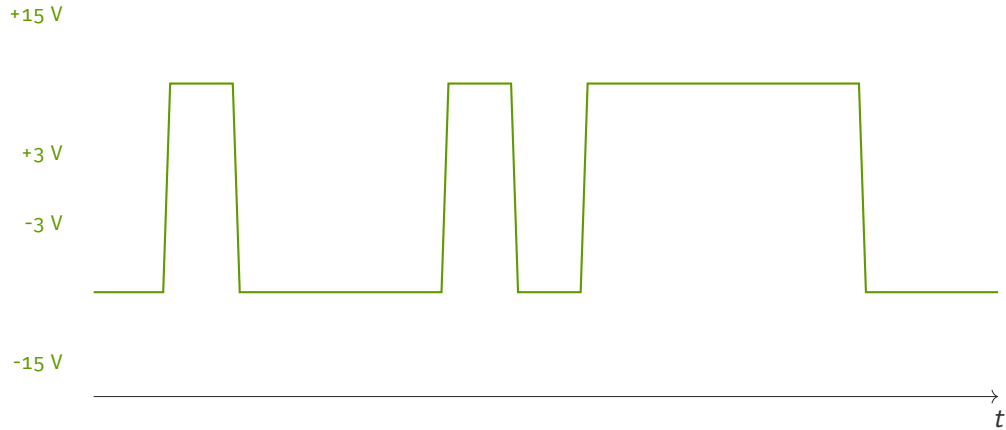
*Quelle: B&H Photo Video*

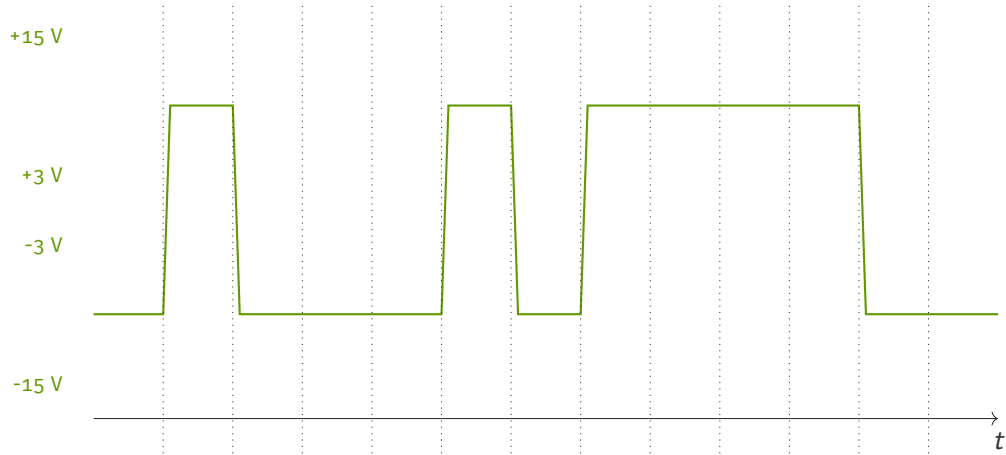


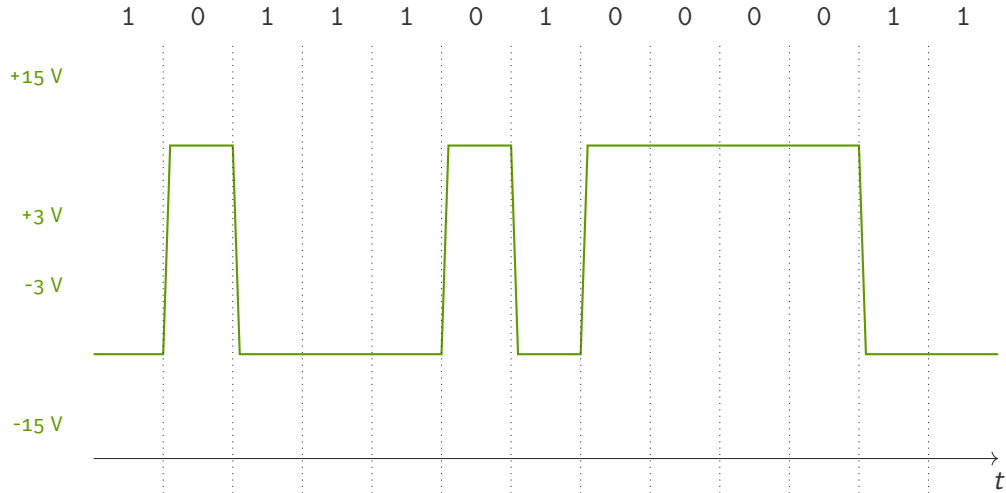
Quelle: B&H Photo Video

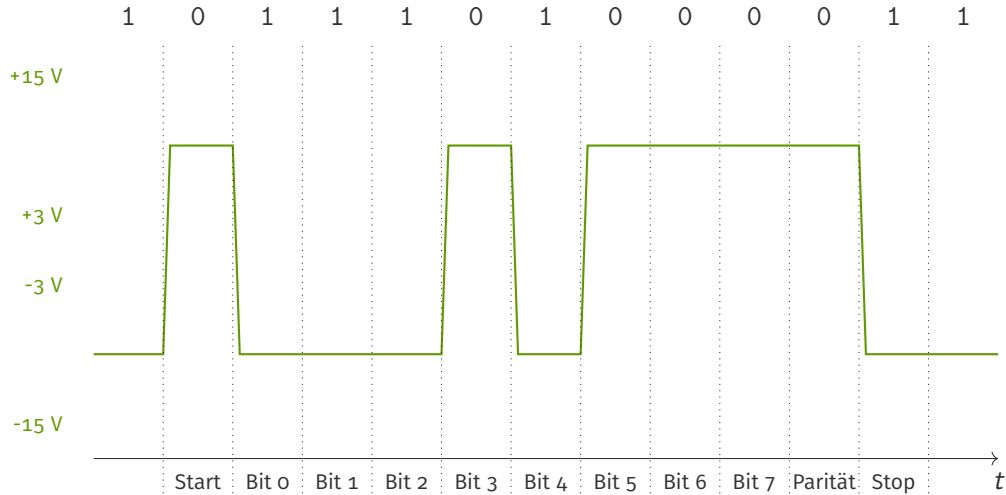


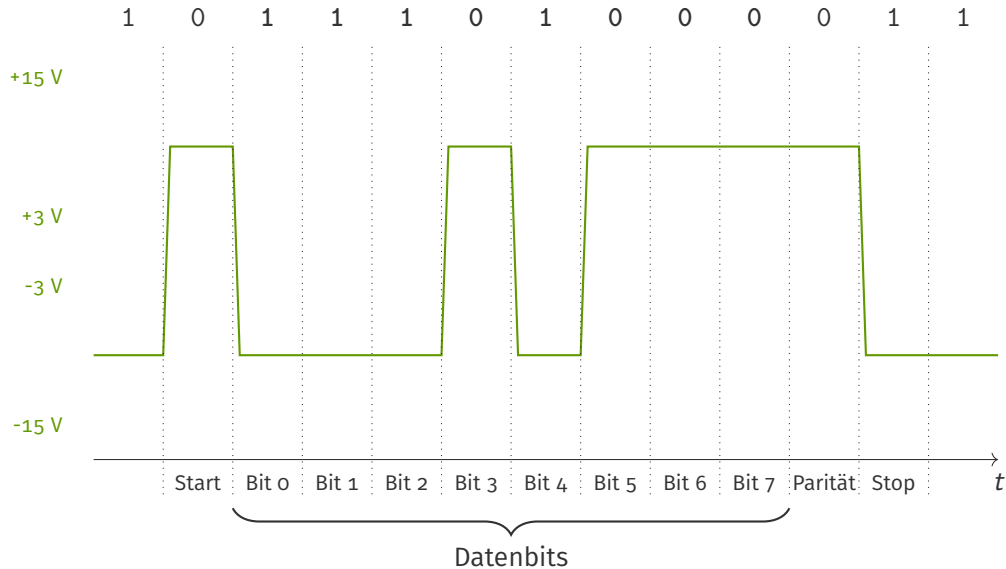
Quelle: B&H Photo Video











- Übertragungsrate (Baud rate) ist Teiler von 115 200 Hz
- Kommunikation 8-N-1 mit 9 600 Baud oft Standardeinstellung
  - 8 Anzahl der Datenbits
  - N kein Paritätsbit
  - 1 Stopbit
- Aktuelle PCs haben derzeit meist maximal eine Hardwareschnittstelle (COM1)
- Controller wird über I/O-Ports programmiert



## ■ Übliche Basisadressen

0x3f8 COM1

0x2f8 COM2

0x3e8 COM3

0x2e8 COM4

- Übliche Basisadressen

  - 0x3f8 COM1

  - 0x2f8 COM2

  - 0x3e8 COM3

  - 0x2e8 COM4

- 12 Register über 8 Offsetadressen

  - 0 Daten (empfangen / senden)

  - 1 Interrupt aktiviert / Teiler niederwertig

  - 2 Interruptregistration / FIFO-Control / Teiler höchstwertig

  - 3 Line-Control

  - 4 Modem-Control

  - ...

- Übliche Basisadressen

  - 0x3f8 COM1

  - 0x2f8 COM2

  - 0x3e8 COM3

  - 0x2e8 COM4

- 12 Register über 8 Offsetadressen

  - 0 Daten (empfangen / senden)

  - 1 Interrupt aktiviert / Teiler niederwertig

  - 2 Interruptregistration / FIFO-Control / Teiler höchstwertig

  - 3 Line-Control

  - 4 Modem-Control

  - ...

- Details auf [osdev.org](http://osdev.org) und [lowlevel.eu](http://lowlevel.eu)





# ANSI-Escape-Sequenzen

- SteuerCodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')



# ANSI-Escape-Sequenzen

- Steuercodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle

\e[Am Attribute

- 0 keine
- 1 fett
- 2 matt
- 3 kursiv \*
- 4 unterstrichen
- 5 blinkend
- 6 blinkend (schnell) \*
- 7 invertiert
- 8 unsichtbar \*

\* wird nur selten unterstützt



# ANSI-Escape-Sequenzen

- Steuercodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle

\e[A**m** Attribute

\e[3**Cm** Vordergrundfarbe

\e[4**Cm** Hintergrundfarbe

**0** schwarz

**1** rot

**2** grün

**3** gelb

**4** blau

**5** magenta

**6** cyan

**7** weiß

**9** *Standardfarbe*



# ANSI-Escape-Sequenzen

- Steuercodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle
  - \e[A<sub>m</sub> Attribute
  - \e[3C<sub>m</sub> Vordergrundfarbe
  - \e[4C<sub>m</sub> Hintergrundfarbe
  - \e[Y;X<sub>H</sub> Cursorposition setzen



- Steuercodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle
  - \e[A<sub>m</sub> Attribute
  - \e[3C<sub>m</sub> Vordergrundfarbe
  - \e[4C<sub>m</sub> Hintergrundfarbe
  - \e[Y;XH Cursorposition setzen
  - \e[6n Cursorposition lesen
  - \e[Y;XR Antwort



# ANSI-Escape-Sequenzen

- Steuercodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle

\e[A<sub>m</sub> Attribute

\e[3C<sub>m</sub> Vordergrundfarbe

\e[4C<sub>m</sub> Hintergrundfarbe

\e[Y;X<sub>H</sub> Cursorposition setzen

\e[6<sub>n</sub> Cursorposition lesen

\e[Y;X<sub>R</sub> Antwort

\ec Reset



# ANSI-Escape-Sequenzen

- Steuercodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle

\e[A**m** Attribute

\e[3**Cm** Vordergrundfarbe

\e[4**Cm** Hintergrundfarbe

\e[Y;**XH** Cursorposition setzen

\e[6**n** Cursorposition lesen

\e[Y;**XR** Antwort

\e**c** Reset

- Testen auf der Kommandozeile

```
1 stud@bsb:~$ echo -e "\ec\e[47m\e[1mF\e[31moo\e[0m"
```



# ANSI-Escape-Sequenzen

- Steuercodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle

\e[A**m** Attribute

\e[3**Cm** Vordergrundfarbe

\e[4**Cm** Hintergrundfarbe

\e[Y;**XH** Cursorposition setzen

\e[6**n** Cursorposition lesen

\e[Y;**XR** Antwort

\e**c** Reset

- Testen auf der Kommandozeile

```
1 stud@bsb:~$ echo -e "\ec\e[47m\e[1mF\e[31moo\e[0m"
```

Siehe auch <http://www.termSYS.demon.co.uk/vtansi.htm> oder bei Wikipedia unter ANSI Escape Code



- Virtualisiert

- QEMU** Softwareemulation

- KVM** Hardware-Virtualisierung

- Virtualisiert

  - **QEMU** Softwareemulation

  - **KVM** Hardware-Virtualisierung

- Nackte Hardware (*bare-metal*)

  - vier Testrechner

    - Intel® Core®-i5 4590 3,3GHz

    - 8 GB Arbeitsspeicher

    - PS/2 Tastatur + Maus

    - COM1 verbunden mit `vesta.cs.tu-dortmund.de (/dev/ttyBSX)`

  - Boot via Netzwerk (PXE)

  - Zugriff: entfernt mittels VNC – via Web:

    - `sys.cs.tu-dortmund.de/de/lehre/ws25/bsb/rechnerverwaltung/`

- Virtualisiert

  - **QEMU** Softwareemulation

  - **KVM** Hardware-Virtualisierung

- Nackte Hardware (*bare-metal*)

  - vier Testrechner

    - Intel® Core®-i5 4590 3,3GHz

    - 8 GB Arbeitsspeicher

    - PS/2 Tastatur + Maus

    - COM1 verbunden mit `vesta.cs.tu-dortmund.de (/dev/ttyBSX)`

  - Boot via Netzwerk (PXE)

  - Zugriff: entfernt mittels VNC – via Web:

    - `sys.cs.tu-dortmund.de/de/lehre/ws25/bsb/rechnerverwaltung/`

Abgabe der Aufgaben auf unseren Testrechnern

## Minimal:

- Internet
- SSH-Zugang zum Sys-Labor

## Optimal:

- PC mit x64 Architektur
- Unixoides System
  - Referenzsysteme: **Debian 11** und **Ubuntu 20.04**
  - Unter Windows **WSL** oder **VirtualBox** möglich
- Möglichkeit Softwarepakete zu installieren

- Aktueller Übersetzer (für x64)
  - Bevorzugt Gcc ( $\geq 7$ )
  - Alternativ LLVM/CLANG ( $\geq 7$ )
  - Via Docker verfügbar: `inf4/stubs`
- Assemblierer NETWIDE ASSEMBLER (NASM)
- Buildtools (u.a. MAKE, objcopy)
- Emulator QEMU/KVM (für x86\_64 Gast)
- Debugger GDB
- *Optional*: Python für `cpplint`
- *Optional*: GNU GRUB & GNU XORRISO für ISO



# Wichtige Makefile Targets

**make qemu** QEMU **ohne** Hardware-Virtualisierung

**make kvm** QEMU **mit** Hardware-Virtualisierung



# Wichtige Makefile Targets

**make qemu** QEMU **ohne** Hardware-Virtualisierung

**make kvm** QEMU **mit** Hardware-Virtualisierung

**make qemu-gdb** starte in QEMU und verbinde zu integrierten GDB-Stub  
→ Fehlersuche mit gdb

**make kvm-gdb** selbiges mit Hardware-Virtualisierung



# Wichtige Makefile Targets

**make qemu** QEMU **ohne** Hardware-Virtualisierung

**make kvm** QEMU **mit** Hardware-Virtualisierung

**make qemu-gdb** starte in QEMU und verbinde zu integrierten GDB-Stub  
→ Fehlersuche mit gdb

**make kvm-gdb** selbiges mit Hardware-Virtualisierung

**make netboot** für Boot am Test-Rechner ins NFS kopieren

Suffix **-dbg** für Debugtransparente Optimierungen (-Og & Framepointer)

Suffix **-noopt** um Optimierungen auszuschalten (sonst -O3)

Suffix **-opt** für aggressive Optimierungen (-Ofast und LTO)

Suffix **-verbose** aktiviert zusätzliche Ausgaben (DBG\_VERBOSE)



**make iso** erstelle ein bootfähiges ISO-Abbild

**make qemu-iso** boote das Abbild in QEMU

**make usb-dev** bootfähigen USB-Stick auf **dev** (z.B. *sdb* - **aber Vorsicht!**)  
erstellen (als Superuser)

**make iso** erstelle ein bootfähiges ISO-Abbild

**make qemu-iso** boote das Abbild in QEMU

**make usb-dev** bootfähigen USB-Stick auf **dev** (z.B. *sdb* - **aber Vorsicht!**) erstellen (als Superuser)

**make solution-#** Musterlösung zur Aufgabe **#** mit Hardware-Virtualisierung starten (auf Testrechner bereits installiert); funktioniert nur im Sys-Labor

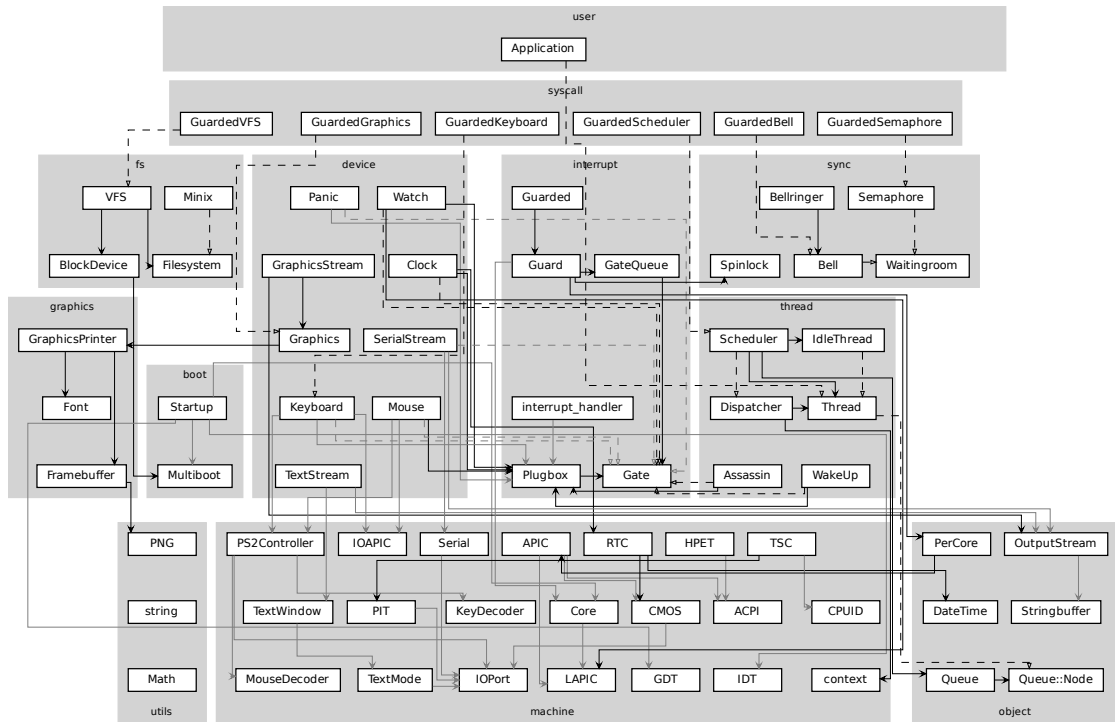
**make lint** prüft die Konformität des Coding Styles

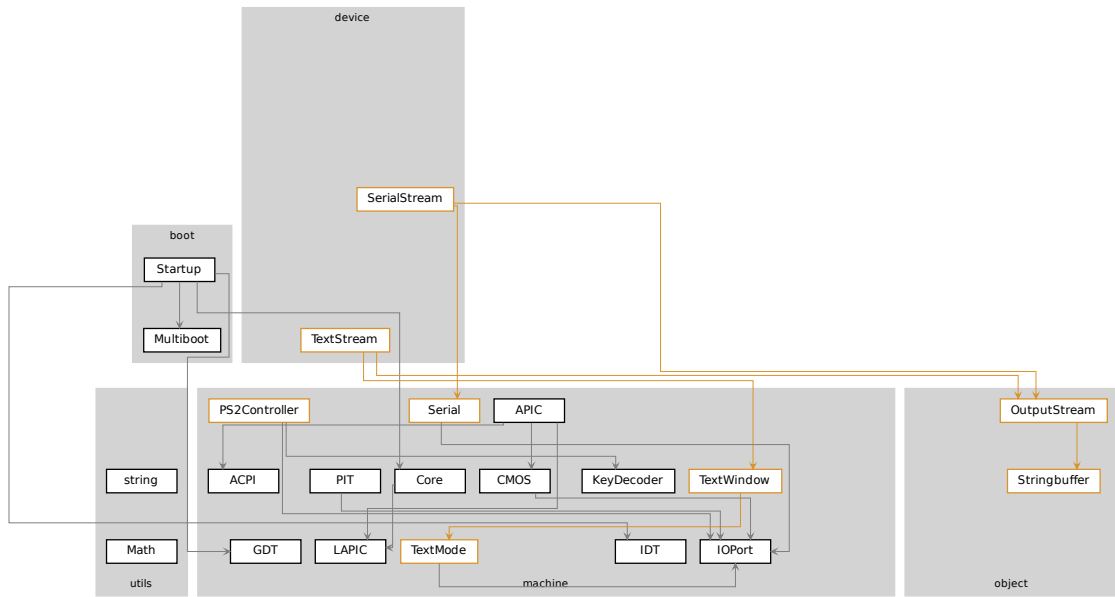
**make help** zeige eine Beschreibung der verfügbaren Targets an

- Einarbeitung in die **Entwicklungsumgebung**

- Einarbeitung in die **Entwicklungsumgebung**
- **C++ Kenntnisse** aneignen/auffrischen

- Einarbeitung in die **Entwicklungsumgebung**
- **C++ Kenntnisse** aneignen/auffrischen
- **Hardwarenahe** Programmierung
  - Ausgabe mittels **CGA Text Mode**
  - Eingabe über die **Tastatur**
  - **Serielle Schnittstelle**





**Abgabe der 1. Aufgabe  
bis Donnerstag, den 12. November**

## **Worauf legen wir Wert?**

- Vollständige Umsetzung der Aufgabenstellung
- Gut getestete Implementierung (auch Randfälle)
- Ordentlicher Quelltext (mit Kommentaren)

**Danke für Eure Aufmerksamkeit!**  
**Gibt es Fragen?**