

# Übung zu Betriebssystembau

## Ein- und Ausgabe

---

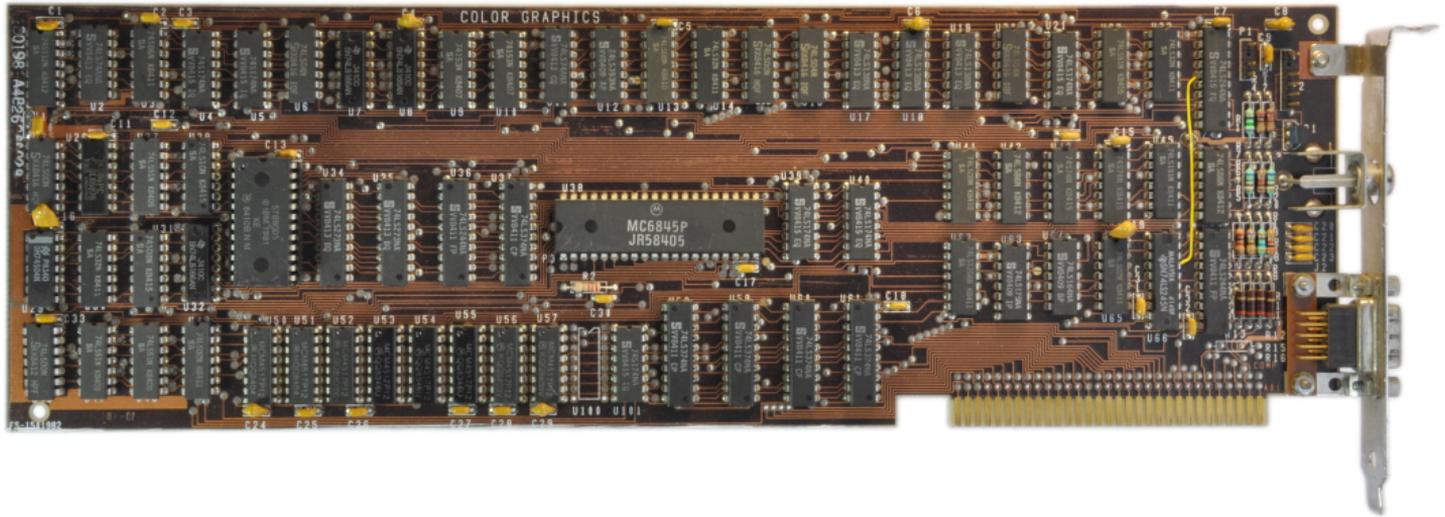
09. Oktober 2023

Peter Ulbrich & Alexander Lochmann  
(Mit Material vom Lehrstuhl 4 der FAU)

Arbeitsgruppe Systemsoftware  
Technische Universität Dortmund

# **Color Graphics Adapter**

---



Quelle: Wikipedia

# Der CGA Bildschirm ...



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

1. Byte: **ASCII Zeichen**

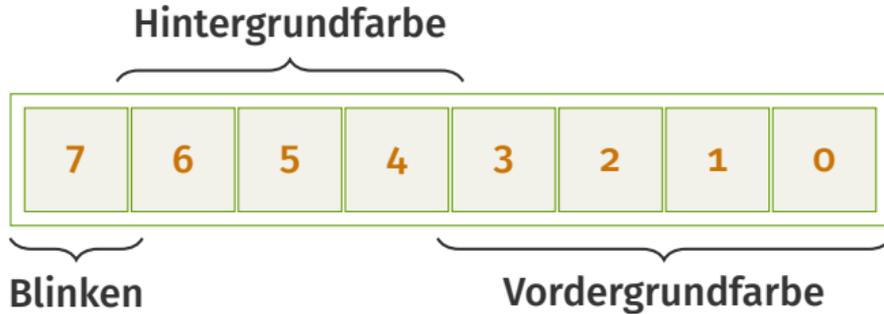


## ... zwei Bytes pro Zeichen ...

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

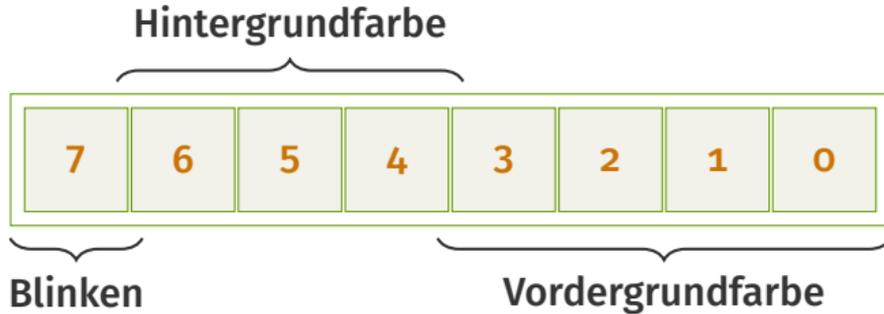
# ... zwei Bytes pro Zeichen ...

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



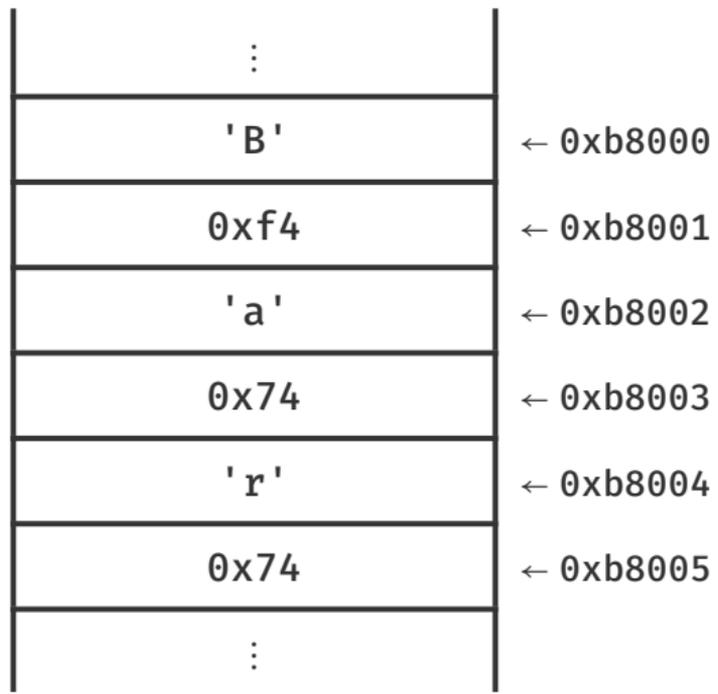
# ... zwei Bytes pro Zeichen ...

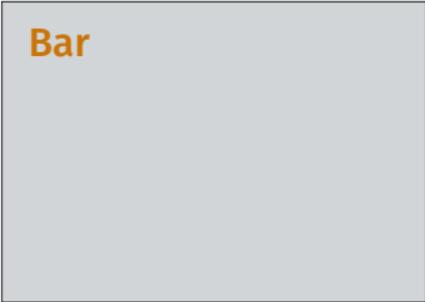
- 1. Byte: **ASCII Zeichen**
- 2. Byte: **Darstellungsattribut**



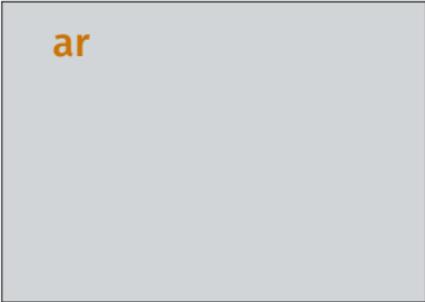
0	schwarz	4	rot	8	dunkelgrau	12	hellrot
1	blau	5	magenta	9	hellblau	13	hellmagenta
2	grün	6	braun	10	hellgrün	14	gelb
3	cyan	7	hellgrau	11	hellcyan	15	weiß

...eingebildet im Arbeitsspeicher ab 0xb8000





**Bar**



ar

# Rekapitulation: Bitshiftoperationen

# Rekapitulation: Bitshiftoperationen

$\&$	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

$\&$	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	$12 \gg 2 = 3$
$x  = (1 \ll n)$	Setze das $n$ te Bit in $x$	
$x \&= \sim(1 \ll n)$	Lösche das $n$ te Bit in $x$	

# Ansprechen von einzelnen Bits

- Bitmasken und logischen Bitoperationen

# Ansprechen von einzelnen Bits

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

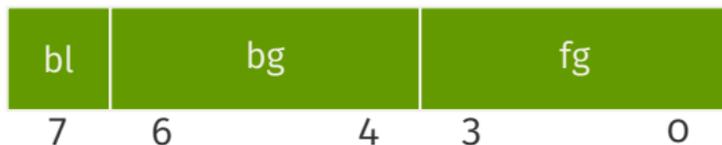
```
01 struct cga_attrib {  
02     unsigned char fg : 4,  
03                 bg : 3,  
04                 bl : 1;  
05 };  
06  
07 struct cga_attrib a;  
08 a.fg = 4; // rot  
09 a.bg = 7; // hellgrau  
10 a.bl = 1; // blinken
```

# Ansprechen von einzelnen Bits

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

```
01 struct cga_attrib {  
02     unsigned char fg : 4,  
03                 bg : 3,  
04                 bl : 1;  
05 };  
06  
07 struct cga_attrib a;  
08 a.fg = 4; // rot  
09 a.bg = 7; // hellgrau  
10 a.bl = 1; // blinken
```

Layout mit GCC auf x86:



## Einschub: Füllen von Strukturen

```
01 struct Test {  
02     char foo[3];  
03     int bar;  
04     bool baz;  
05 };  
06  
07 cout << sizeof(Test);
```

## Einschub: Füllen von Strukturen

```
01 struct Test {  
02     char foo[3]; // 3 Byte  
03     int bar;     // 4 Byte  
04     bool baz;   // 1 Byte  
05 };  
06  
07 cout << sizeof(Test);
```

## Einschub: Füllen von Strukturen

```
01 struct Test {  
02     char foo[3]; // 3 Byte  
03     int bar;     // 4 Byte  
04     bool baz;    // 1 Byte  
05 };  
06  
07 cout << sizeof(Test); // "12"
```

## Einschub: Füllen von Strukturen

```
01 struct Test {  
02     char foo[3]; // 3 Byte  
03     int bar;     // 4 Byte  
04     bool baz;    // 1 Byte  
05 } __attribute__((packed));  
06  
07 cout << sizeof(Test); // "8"
```

## Einschub: Füllen von Strukturen

```
01 struct Test {  
02     char foo[3]; // 3 Byte  
03     int bar;     // 4 Byte  
04     bool baz;    // 1 Byte  
05 } __attribute__((packed));  
06  
07 static_assert(sizeof(Test) == 8, "Test_kaputt");
```

# Schreibmarke (Cursor)

---

**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

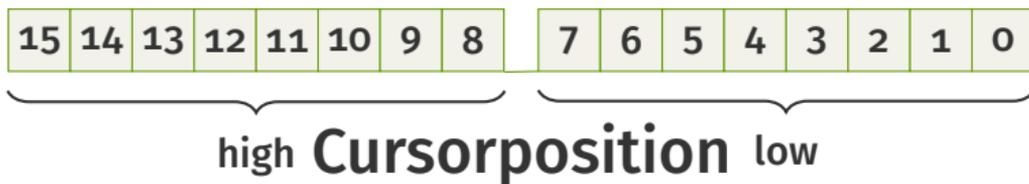


0x3d5

Datenregister



high **Cursorposition** low



0x3d4

Indexregister

15

0x3d5

Datenregister

0x55

in



high **Cursorposition** low



0x3d4

Indexregister

14

0x3d5

Datenregister

0x0a

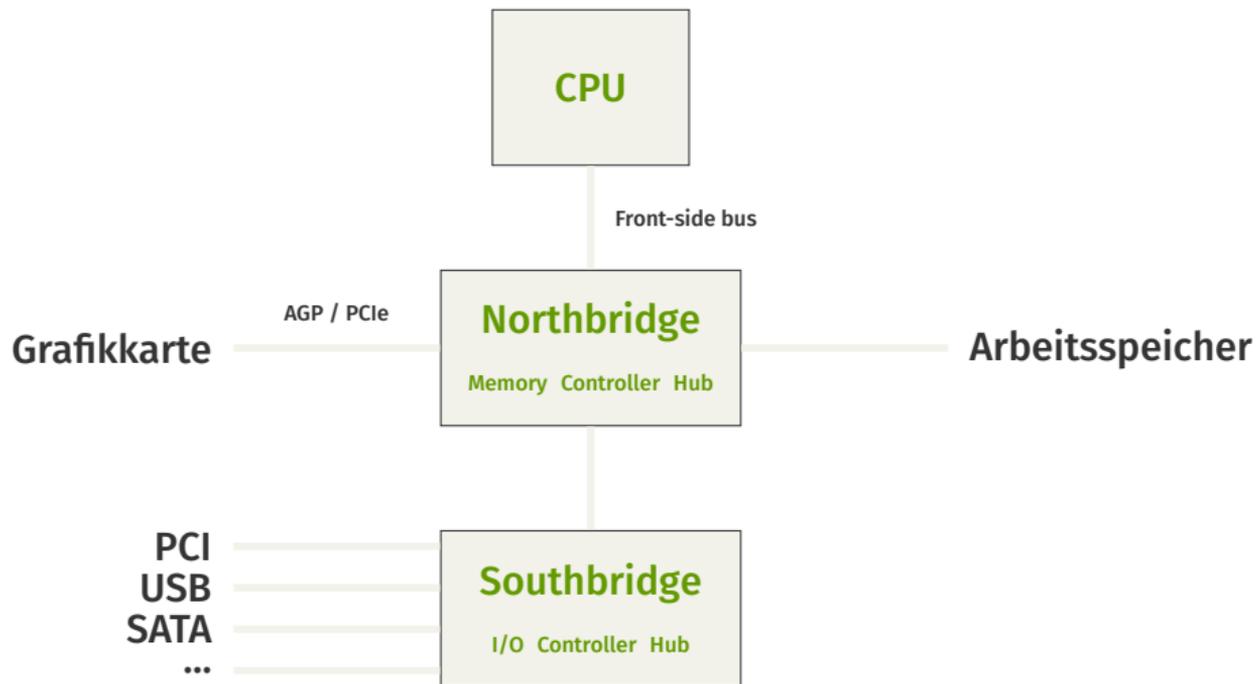
in

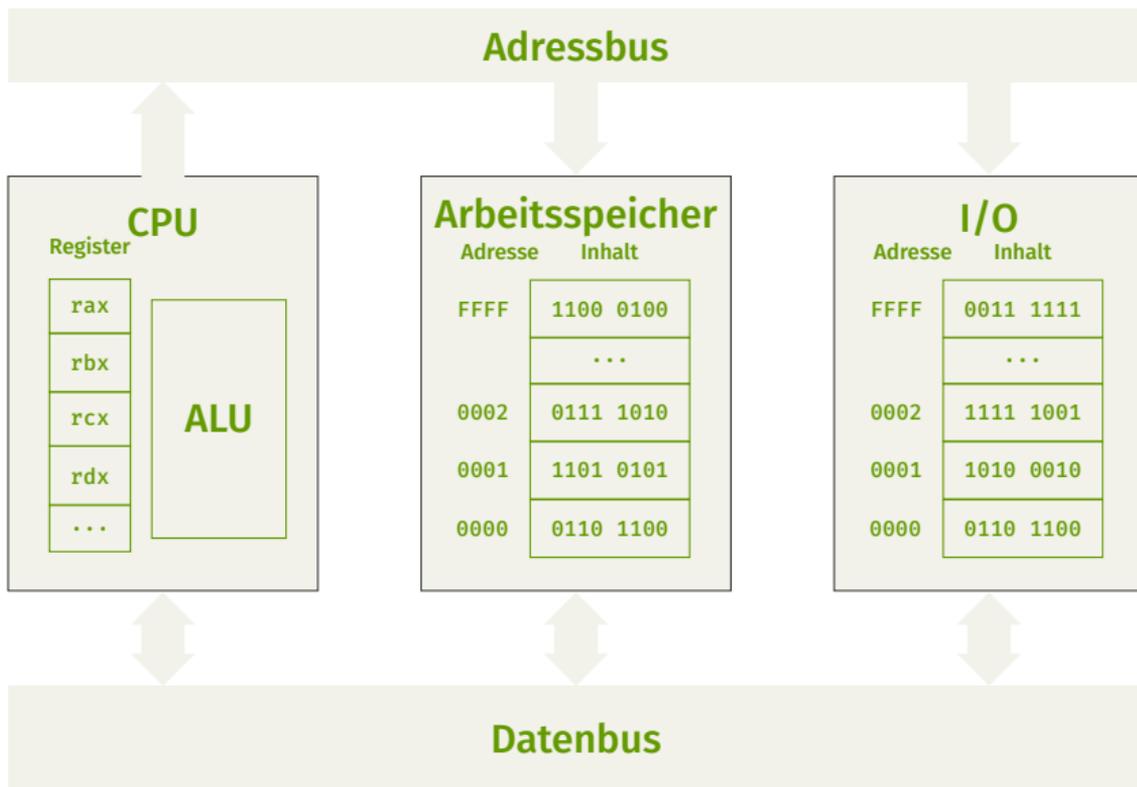


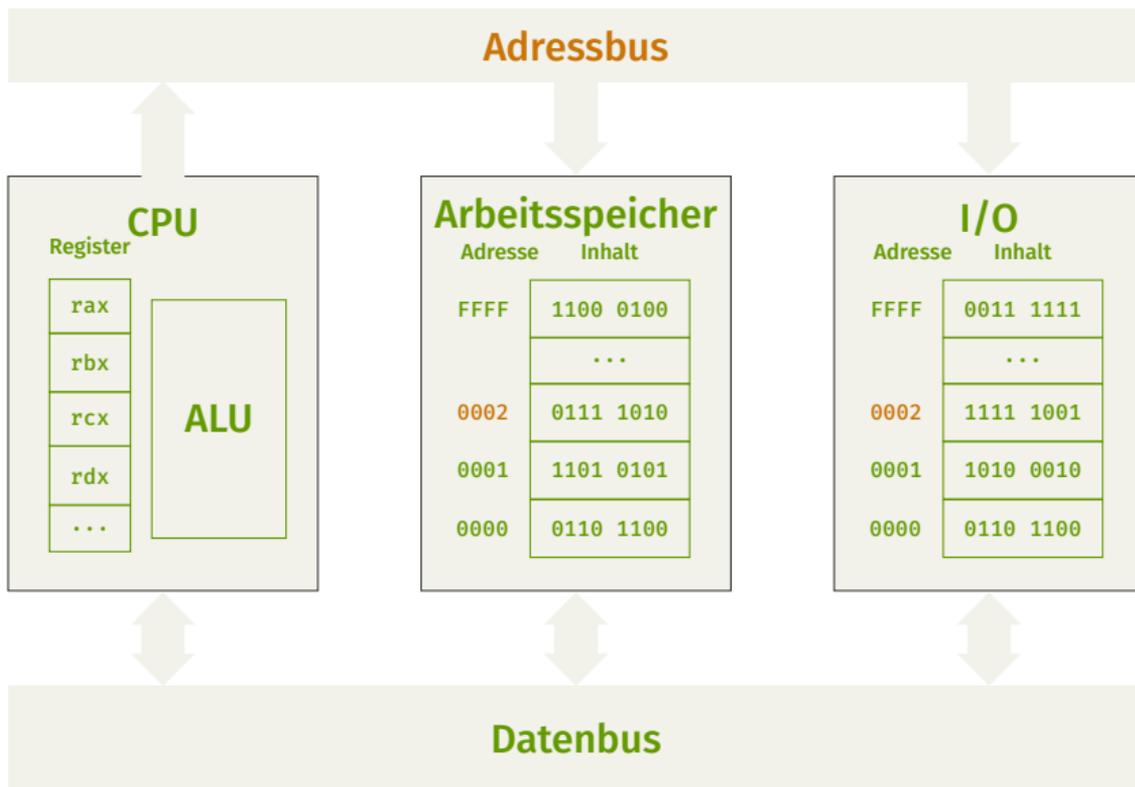
high **Cursorposition** low

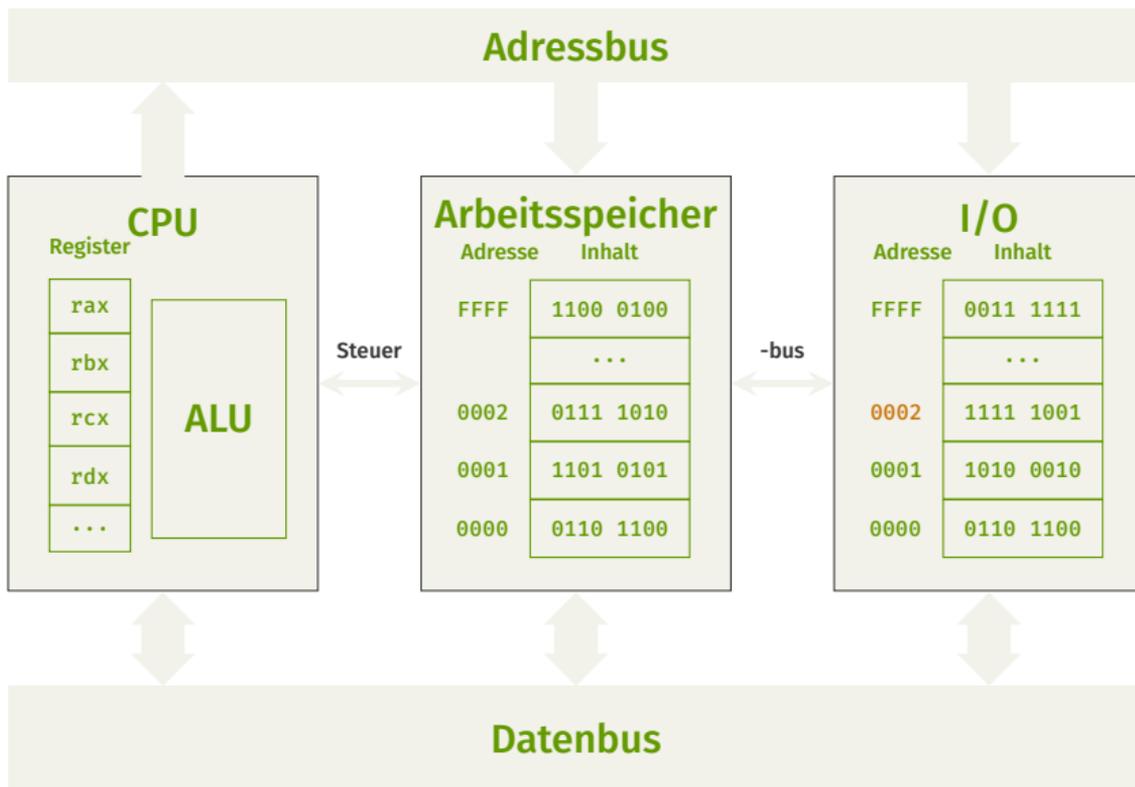
## **Kleiner Exkurs: Zugriff auf Arbeitsspeicher vs. I/O**

---









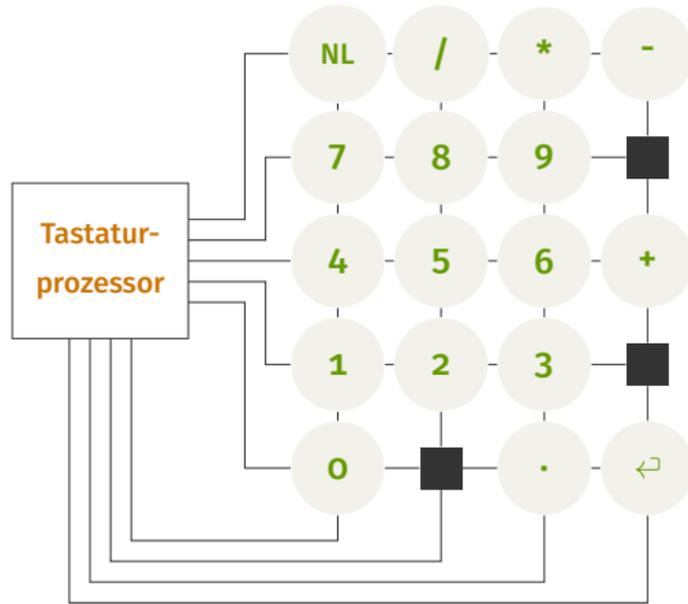
# Tastatur

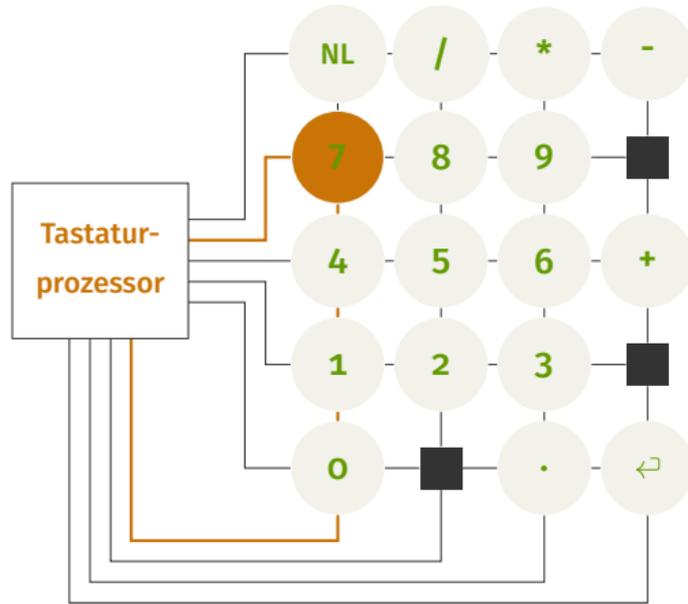
---

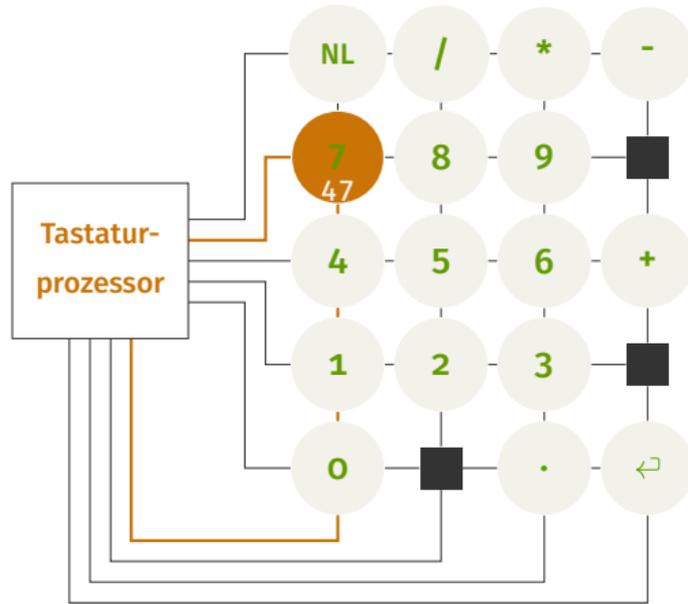


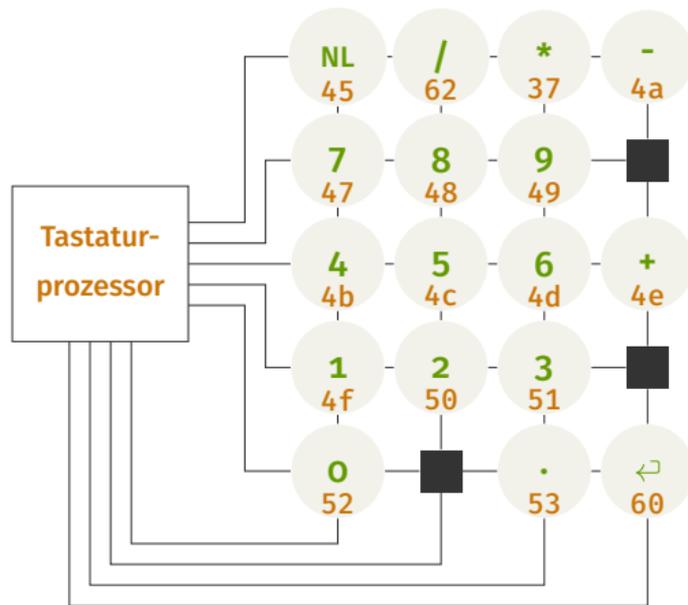
Quelle: Golem

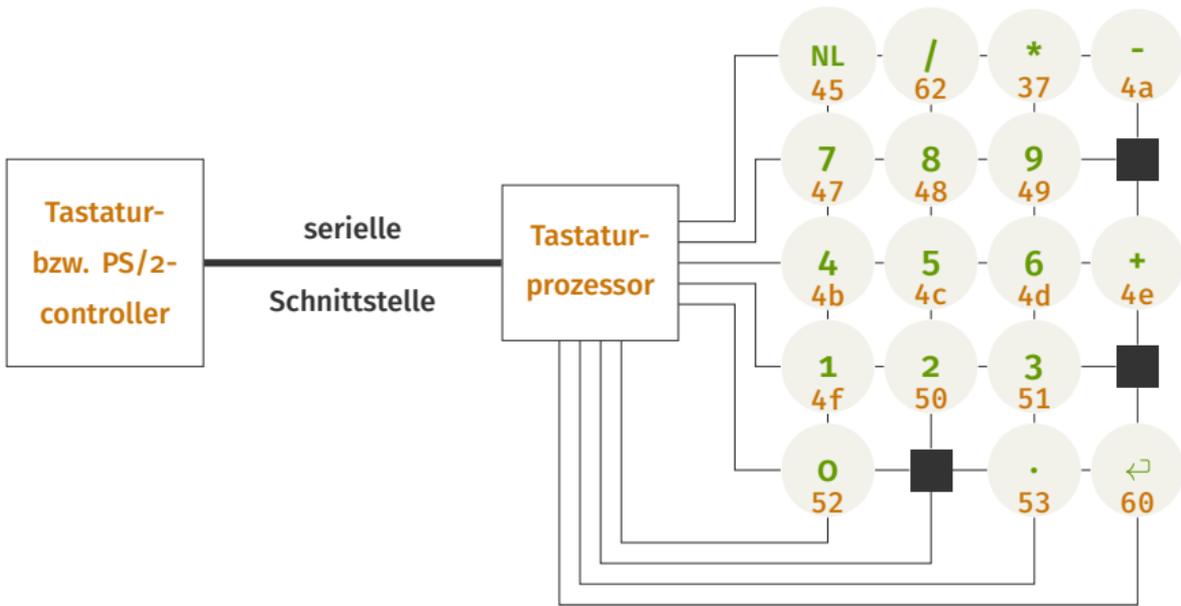


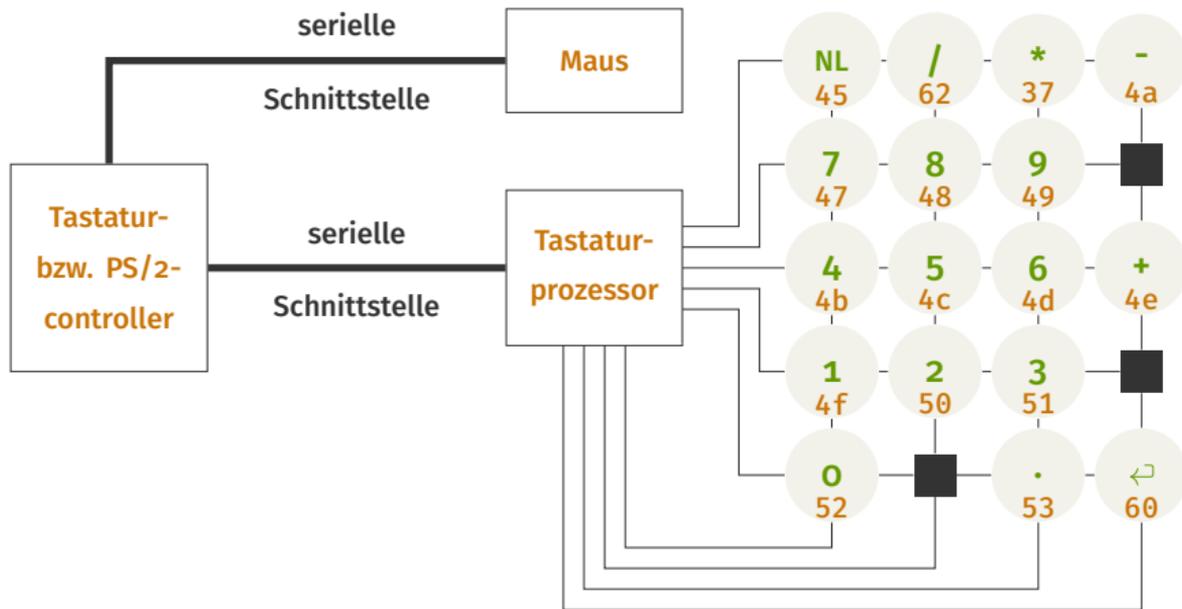












## Kommunikation mit

- 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

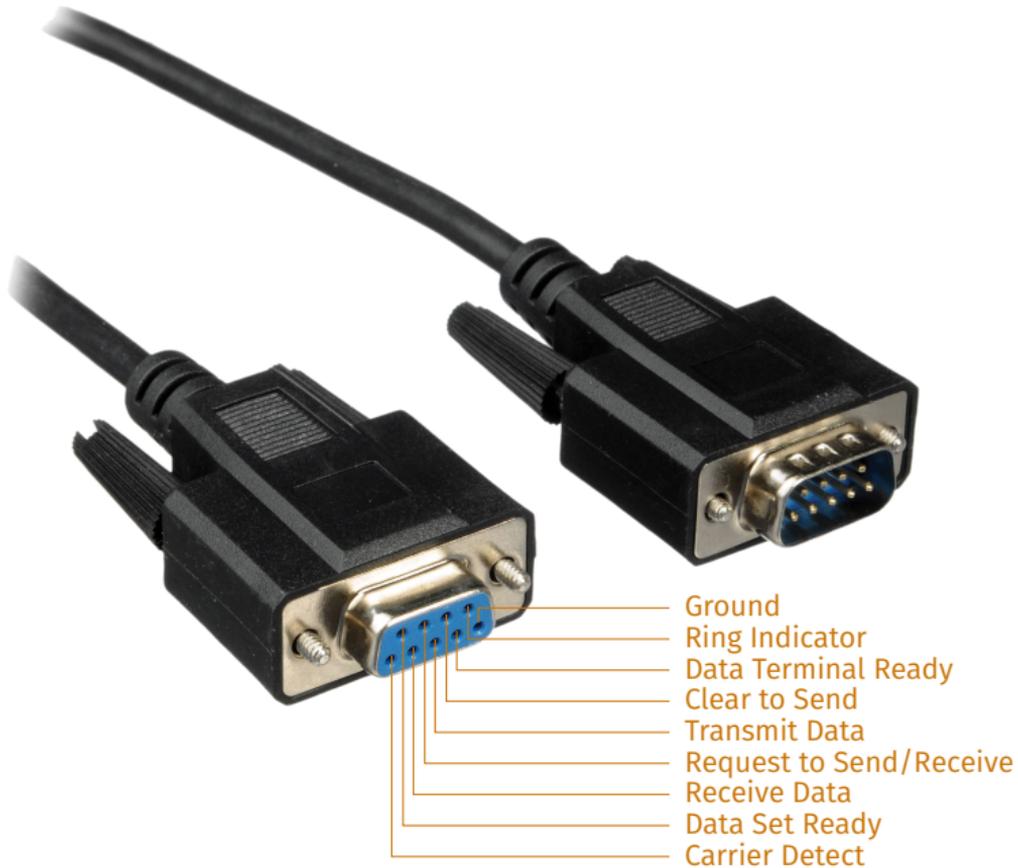
→ Umwandlung mittels (Software)Dekoder

# Serielle Schnittstelle

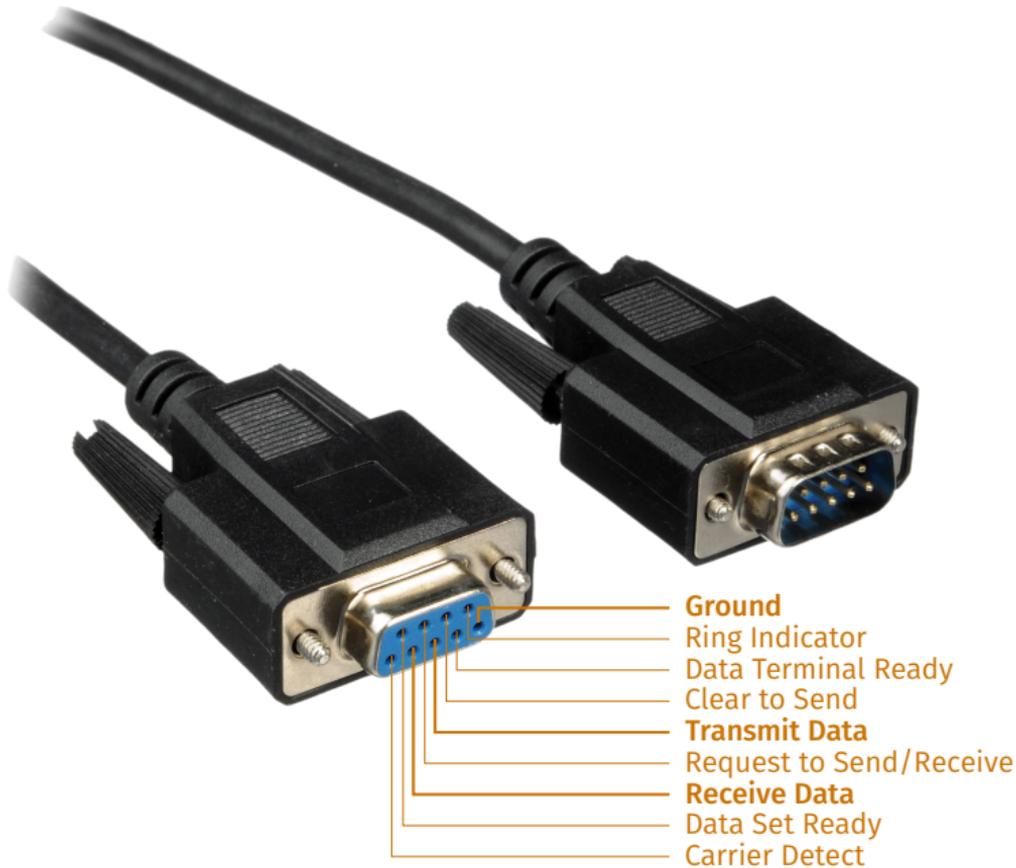
---



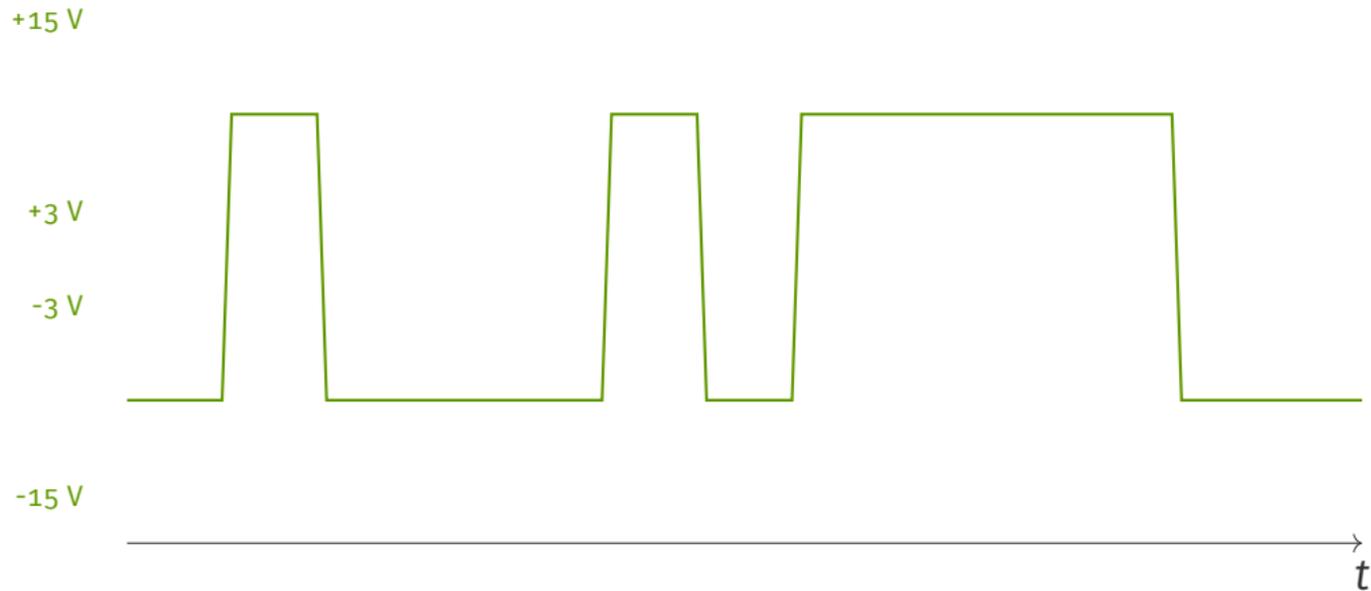
*Quelle: B&H Photo Video*

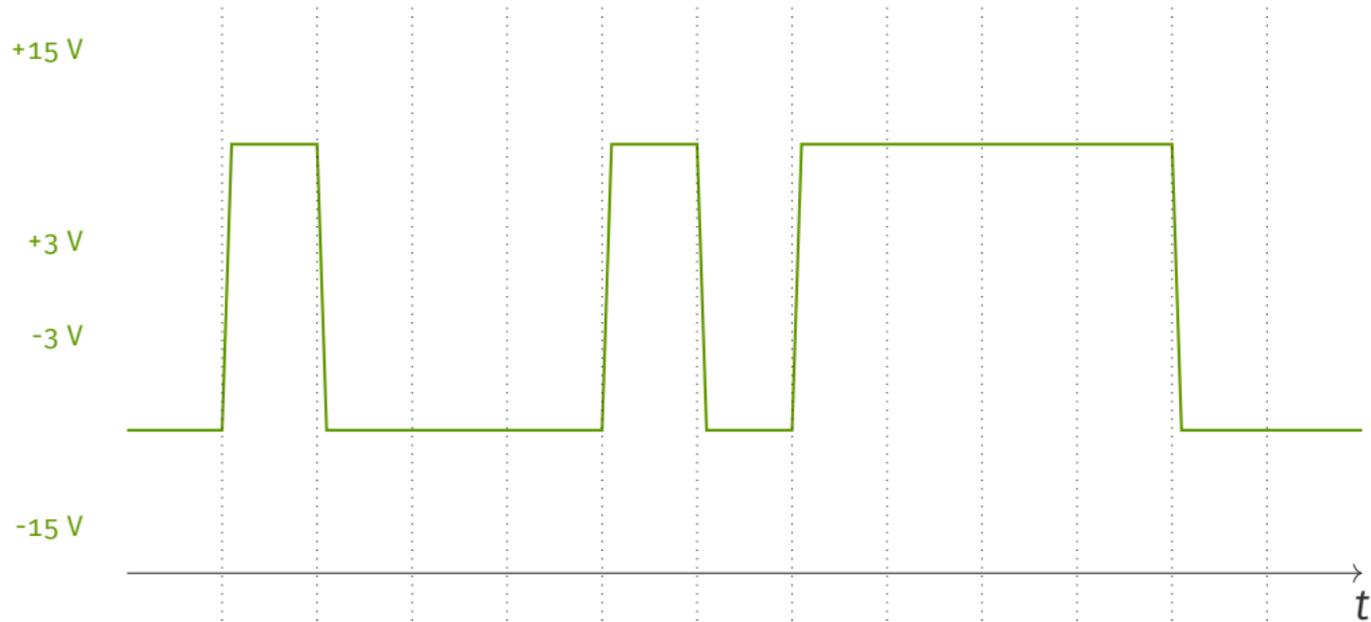


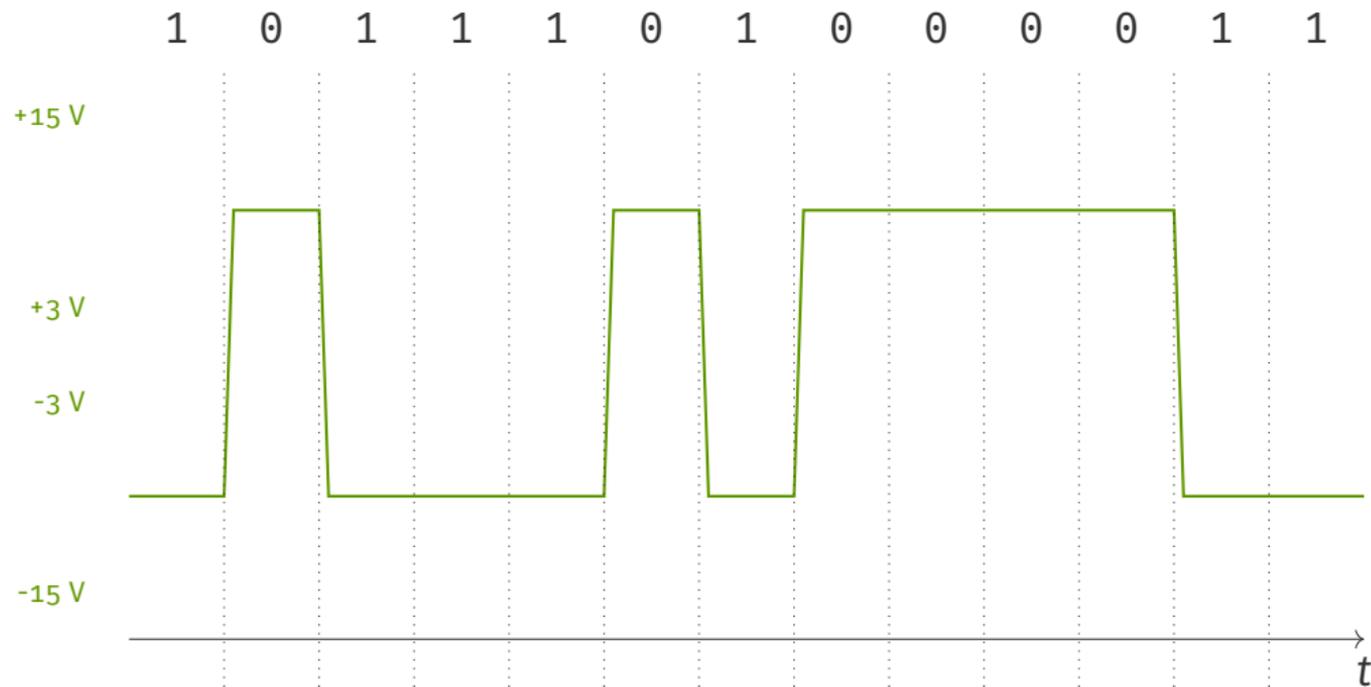
Quelle: B&H Photo Video

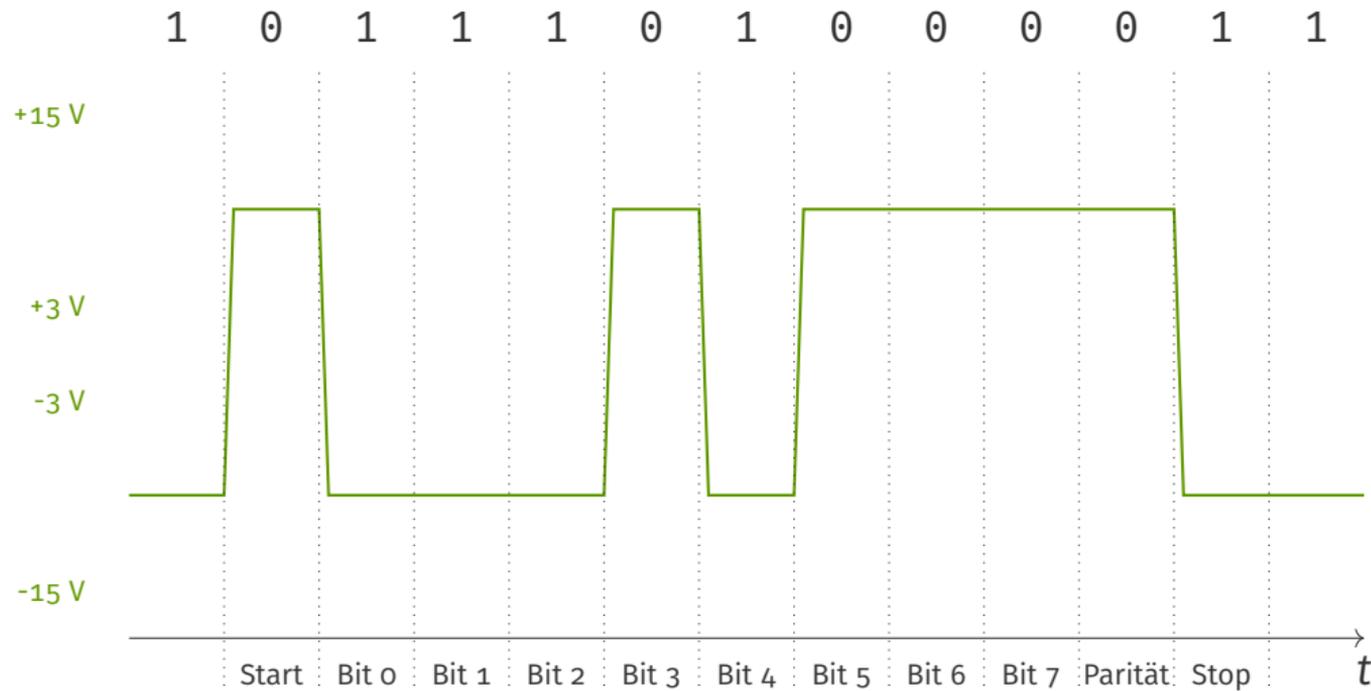


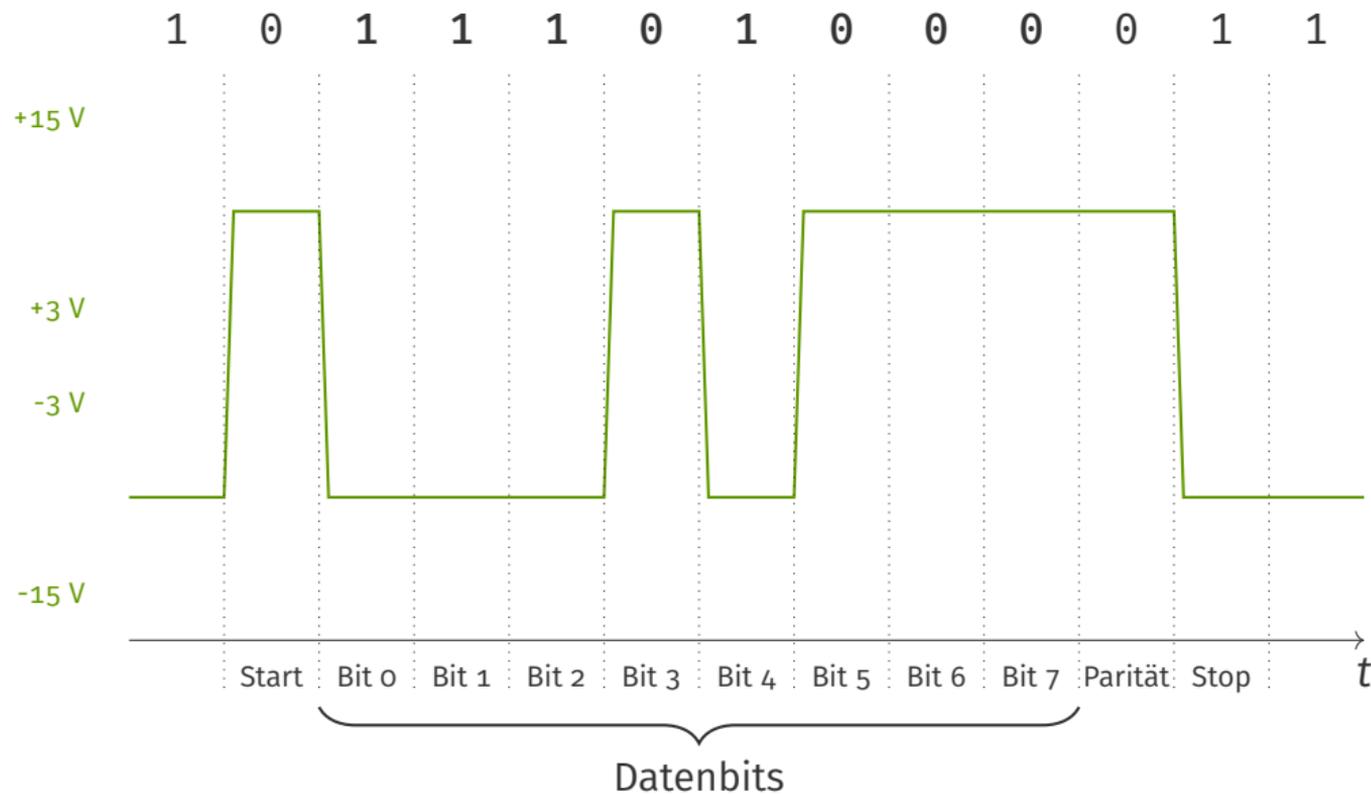
Quelle: B&H Photo Video











- 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

# Serielle Schnittstelle (I/O-Ports)

- Übliche Basisadressen

0x3f8 COM1

0x2f8 COM2

0x3e8 COM3

0x2e8 COM4

# Serielle Schnittstelle (I/O-Ports)

- 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

  - 5 Line-Status

  - 6 Modem-Status

  - 7 Scratch

# Serielle Schnittstelle (I/O-Ports)

- 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

  - 5 Line-Status

  - 6 Modem-Status

  - 7 Scratch

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

# Terminal

---



# 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[Am Attribute

\e[3Cm Vordergrundfarbe

\e[4Cm 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[Am Attribute
  - \e[3Cm Vordergrundfarbe
  - \e[4Cm Hintergrundfarbe
  - \e[Y;XH Cursorposition setzen

# ANSI-Escape-Sequenzen

- SteuerCodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle
  - \e[Am Attribute
  - \e[3Cm Vordergrundfarbe
  - \e[4Cm 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[Am Attribute
  - \e[3Cm Vordergrundfarbe
  - \e[4Cm Hintergrundfarbe
  - \e[Y;XH Cursorposition setzen
  - \e[6n Cursorposition lesen
  - \e[Y;XR 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**C**m Vordergrundfarbe
  - \e[4**C**m Hintergrundfarbe
  - \e[**Y**;**X**H Cursorposition setzen
  - \e[6**n** Cursorposition lesen
  - \e[**Y**;**X**R Antwort
  - \e[c Reset
- Testen auf der Kommandozeile

```
01 heinloth:~$ 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**C**m Vordergrundfarbe
  - \e[4**C**m Hintergrundfarbe
  - \e[**Y**;**X**H Cursorposition setzen
  - \e[6**n** Cursorposition lesen
  - \e[**Y**;**X**R Antwort
  - \e**c** Reset
- Testen auf der Kommandozeile

```
01 heinloth:~$ echo -e "\ec\e[47m\e[1mF\e[31moo\e[0m"
```

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

# Entwicklungsumgebung

---

- 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:

    - `https://sys-stubs.cs.tu-dortmund.de`

- 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:  
`https://sys-stubs.cs.tu-dortmund.de`

## Abgabe der Aufgaben auf unseren Testrechnern

# Systemvoraussetzung (für zuhause)

## 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 cpp lint
- *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)

# Weitere Makefile Targets

**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)

# Weitere Makefile Targets

- 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

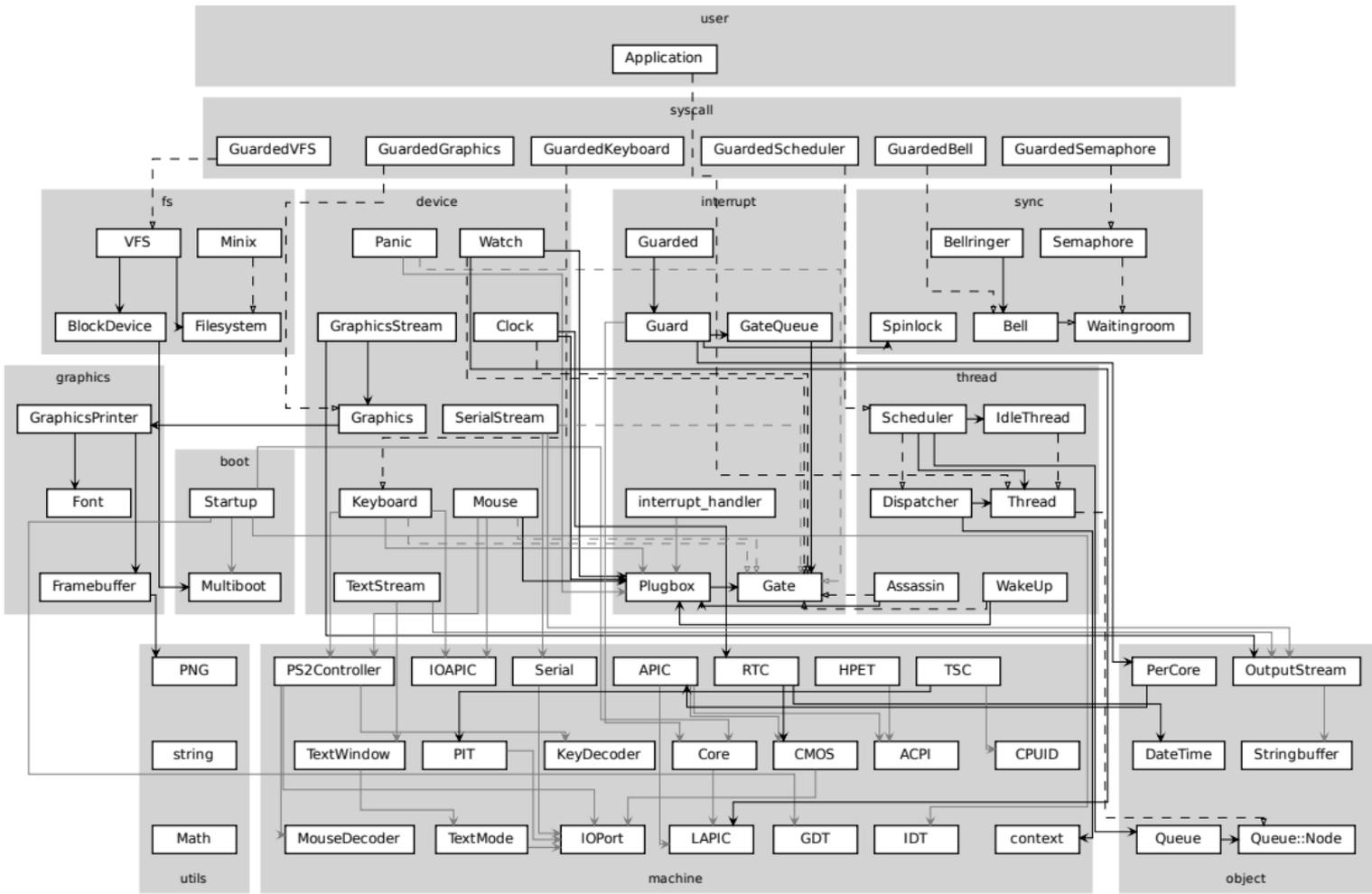
# Aufgabe 1

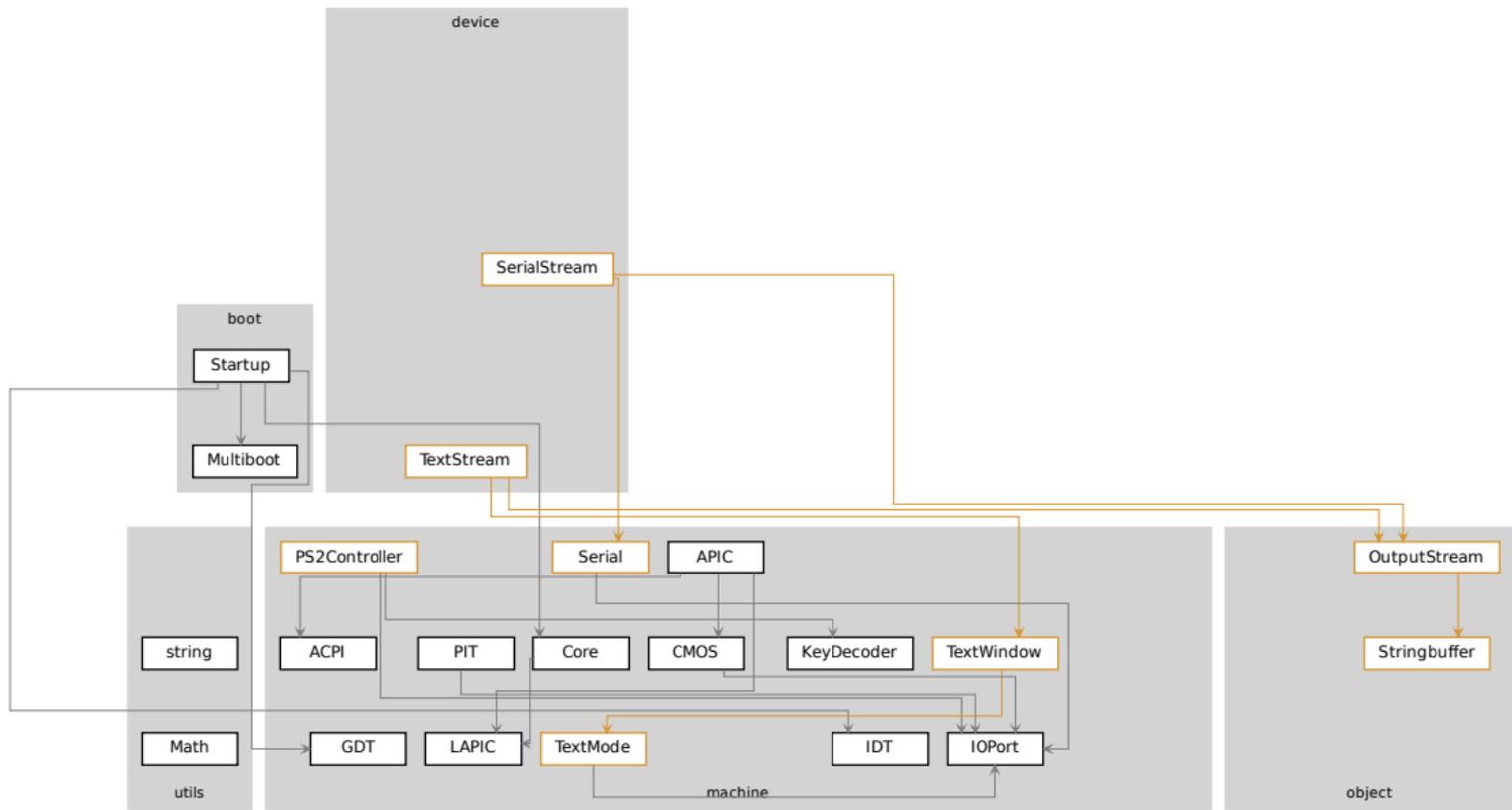
---

- 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 Mittwoch, den 01. 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?**

---

**Zur Erinnerung: Nächsten Dienstag (17.10.)  
findet eine Rechnerübung [mit Vortrag] statt.**