# Übung zu Betriebssysteme

Ein- und Ausgabe

25. Oktober 2022

Phillip Raffeck, Dustin Nguyen & Bernhard Heinloth

Lehrstuhl für Informatik 4 Friedrich-Alexander-Universität Erlangen-Nürnberg





**Color Graphics Adapter** 



Quelle: Wikipedia

# Der CGA Bildschirm ...



1. Byte: ASCII Zeichen

1. Byte: ASCII Zeichen

1. Byte: ASCII Zeichen

2. Byte: Darstellungsattribut

1. Byte: ASCII Zeichen

2. Byte: Darstellungsattribut



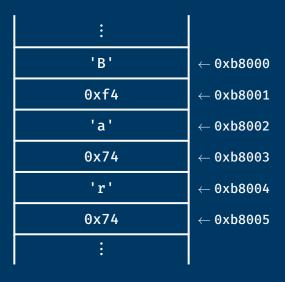
1. Byte: ASCII Zeichen

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

# ...eingeblendet im Arbeitsspeicher ab 0xb8000



Bar

ar

# Rekapitulation: Bitschubsenoperationen

# Rekapitulation: Bitschubsenoperationen

# Rekapitulation: Bit<del>schubsen</del>operationen

```
% Konjunktion (bitweises UND) 3 & 5 = 1
| Disjunktion (bitweises ODER) 3 | 5 = 7
| exklusives ODER 3 ^ 5 = 6
| Einerkomplement (bitweise Negation) ~ 5 = 250
| verschieben nach links 12 << 2 = 48
| verschieben nach rechts 12 >> 2 = 3
x |= (1 << n) Setze das nte Bit in x</p>
```

 $x \&= \sim (1 << n)$  Lösche das nte Bit in x

## **Ansprechen von einzelnen Bits**

■ Bitmasken und logische Bitoperationen

## Ansprechen von einzelnen Bits

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

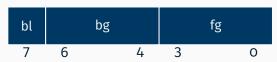
```
struct cga_attrib {
    unsigned char fg : 4,
                  bl : 1;
struct cga_attrib a;
a.fg = 4; // rot
a.bg = 7; // hellgrau
a.bl = 1; // blinken
```

## Ansprechen von einzelnen Bits

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

```
struct cga_attrib {
    unsigned char fg : 4,
struct cga_attrib a;
a.fg = 4; // rot
a.bg = 7; // hellgrau
a.bl = 1; // blinken
```

#### Layout mit GCC auf x86:



```
struct Test {
    char foo[3];
    int bar;
    bool baz;
};
cout << sizeof(Test);</pre>
```

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

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

```
struct Test {
    char foo[3]; // 3 Byte
    int bar; // 4 Byte
    bool baz; // 1 Byte
} __attribute__((packed));

cout << sizeof(Test); // "8"</pre>
```

```
struct Test {
    char foo[3]; // 3 Byte
    int bar; // 4 Byte
    bool baz; // 1 Byte
} __attribute__((packed));

static_assert(sizeof(Test) == 8, "Test kaputt");
```

Schreibmarke (Cursor)

Schreibmarke (Cursor)\_



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

■ CGA hat 18 Steuerregister mit je 8 Bit

0	Horizontal Total	] v
1	Horizontal Displayed	
2	H. Sync Position	
3	H. Sync Width	
4	Vertical Total	
5	V. Total Adjust	
6	Vertical Displayed	
7	V. Sync Position	
8	Interlace Mode	
9	Max Scan Line Address	∏ v
LO	Cursor Start	∏ v
L1	Cursor End	∏ v
L2	Start Address (high)	∏ v
L3	Start Address (low)	V
L4	Cursor (high)	r
L5	Cursor (low)	r
١6	Light Pen (high)	r
L7	Light Pen (low)	r

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

0	Horizontal Total	\
1	Horizontal Displayed	٦ ،
2	H. Sync Position	ן ∖
3	H. Sync Width	ן ∖
4	Vertical Total	۱ [
5	V. Total Adjust	۱ [
6	Vertical Displayed	۱ [
7	V. Sync Position	۱ [
8	Interlace Mode	۱ [
9	Max Scan Line Address	ן ∖
10	Cursor Start	۱ [
11	Cursor End	۱ [
12	Start Address (high)	۱ [
13	Start Address (low)	۱ [
14	Cursor (high)	_ r
15	Cursor (low)	_ r
16	Light Pen (high)	_  լ
17	Light Pen (low)	_ լ

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

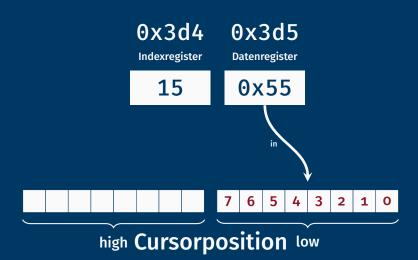




high Cursorposition low

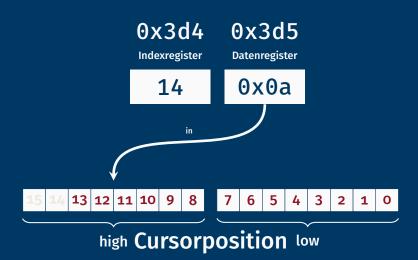




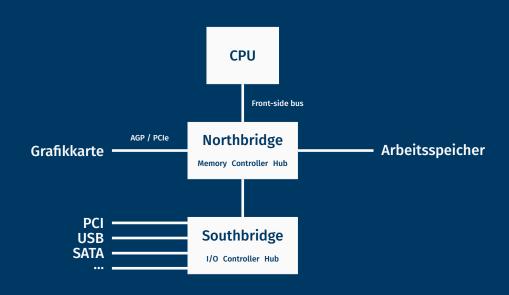








# Kleiner Exkurs: Zugriff auf Arbeitsspeicher vs. I/O







rax	
rbx	
rcx	ALU
rdx	
• • • •	

## Arbeitsspeicher

Adress	e Inhalt
FFFF	1100 0100
	•••
0002	0111 1010
0001	1101 0101
0000	0110 1100

#### 1/0

Adresse Inhalt		
FFFF	0011 1111	
	•••	
0002	1111 1001	
0001	1010 0010	
0000	0110 1100	





#### **Datenbus**





rax	
rbx	
rcx	ALU
rdx	
• • •	

## Arbeitsspeicher

Adresse Inhalt	
FFFF	1100 0100
	•••
0002	0111 1010
0001	1101 0101
0000	0110 1100

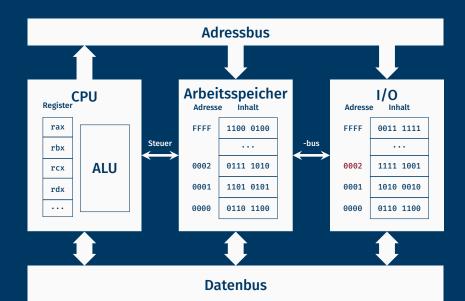
#### I/O

Adresse Inhalt		
FFFF	0011 1111	
	•••	
0002	1111 1001	
0001	1010 0010	
0000	0110 1100	





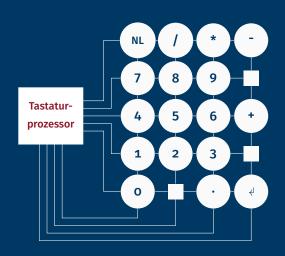
**Datenbus** 

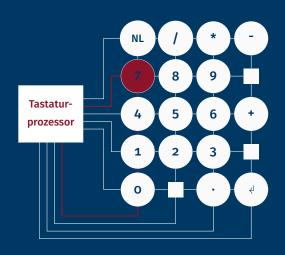


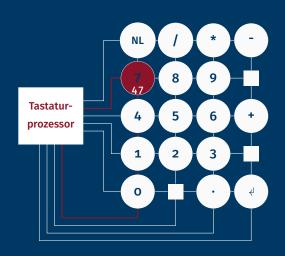
## **Tastatur**

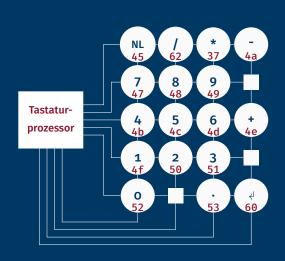


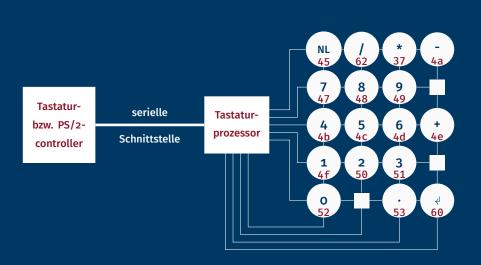


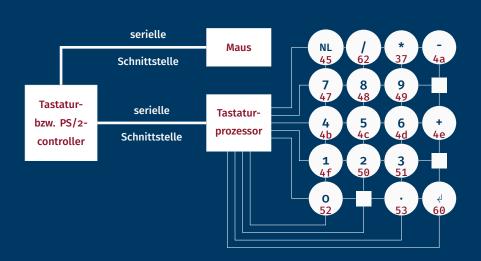












- Tastatur (primäres PS/2 Gerät)
- Maus (sekundäres PS/2 Gerät)
- PS/2 Controller (Konfiguration)

- 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

- 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

- 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)



**MAKECODE:** Tastendruck

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

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

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

**MAKECODE:** Tastendruck

MAKECODE: Tastendruck

BREAKCODE: Taste loslassen (Scancode + 0x80)

ASCII: Darstellung von Zeichen, 8 Bit

→ Umwandlung mittels (Software)Dekoder

Entwicklungsumgebung

## **Testumgebung**

■ Virtualisiert

**QEMU** Softwareemulation **KVM** Hardware-Virtualisierung

## Testumgebung

■ Virtualisiert

**QEMU** Softwareemulation

**KVM** Hardware-Virtualisierung

- Nackte Hardware (bare-metal)
  - vier (identische) Testrechner
    - Intel<sup>®</sup> Core<sup>®</sup> i5 CPU (2 Kerne @ 3.2 GHz / 4 Threads (HT))
    - 8 GB Arbeitsspeicher
    - PS/2 Tastatur + Maus
    - COM1 verbunden mit cip6d0.cip.cs.fau.de (/dev/ttyBSX)
  - Boot via Netzwerk (PXE)
  - Zugriff
    - direkt im WinCIP mittels KVM-Switch
    - entfernt mittels VNC auch via Web: https://i4stubs.cs.fau.de
  - weitere Testrechner (mit 8 Threads) verfügbar

## Testumgebung

■ Virtualisiert

**QEMU** Softwareemulation

**KVM** Hardware-Virtualisierung

- Nackte Hardware (bare-metal)
  - vier (identische) Testrechner
    - Intel<sup>®</sup> Core<sup>®</sup> i5 CPU (2 Kerne @ 3.2 GHz / 4 Threads (HT))
    - 8 GB Arbeitsspeicher
    - PS/2 Tastatur + Maus
    - COM1 verbunden mit cip6d0.cip.cs.fau.de (/dev/ttyBSX)
  - Boot via Netzwerk (PXE)
  - Zugriff
    - direkt im WinCIP mittels KVM-Switch
    - entfernt mittels VNC auch via Web: https://i4stubs.cs.fau.de
  - weitere Testrechner (mit 8 Threads) verfügbar

Abgabe der Aufgaben auf unseren Testrechnern

## Systemvoraussetzung (für zuhause)

#### **Minimal:**

- Internet
- SSH Zugang in CIP
- oder Webbrowser für remote.cip.cs.fau.de

### **Optimal:**

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

## Entwicklungsumgebung

- Aktueller Übersetzer (für x64)
  - Bevorzugt Gcc (≥ 7)
  - Alternativ LLVM/CLANG (≥ 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)

## 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)

make lint prüft die Konformität des Coding Styles

make help zeige eine Beschreibung der verfügbaren Targets an

### Arbeiten mit QEMU/KVM

- lokale Ausführung mit **GTK** Frontend: **Linke** Ctrl + Alt Tasten
  - Maus mittels Ctrl + Alt + g befreien
  - Vollbild umschalten mit [Ctrl]+[Alt]+[f]
  - Beenden mit Ctrl + Alt + q
- lokale Ausführung mit (altem) **SDL** Frontend: **Rechte** Ctrl | Alt | Tasten
  - Maus mittels [Ctrl]+[Alt] Kombination befreien
  - Vollbild mit Ctrl + Alt + f umschalten
  - Beenden durch wechseln in den Monitor mit Ctrl + Alt + 2 und anschließendem q
- Curses Frontend (z.B. bei SSH): Linke Alt -Taste
  - Beenden durch wechseln in den Monitor mit Alt + 2 und anschließendem q
  - Zurück zur Ausgabe mit Alt + 1

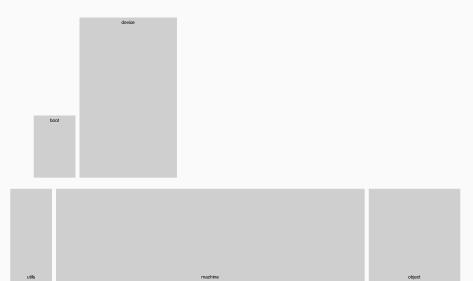
# Aufgabe 1

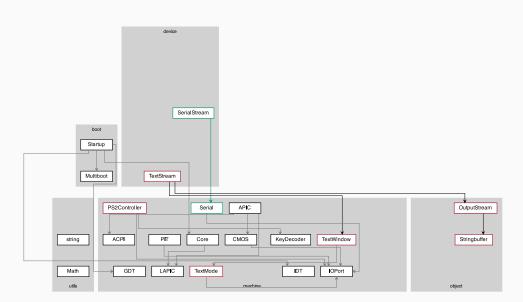
■ Einarbeitung in die Entwicklungsumgebung

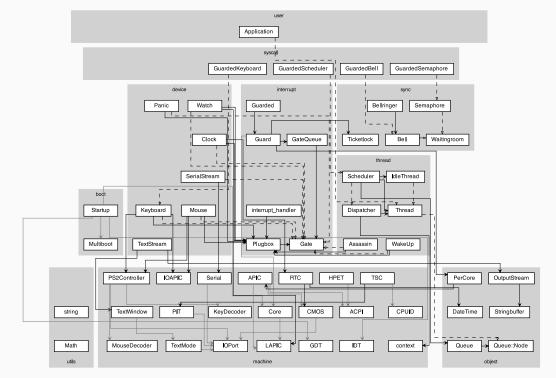
■ C++ Kenntnisse aneignen/auffrischen

Einarbeitung in die Entwicklungsumgebung

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







Abgabe der 1. Aufgabe

bis Freitag, den 11. November

#### Worauf legen wir Wert?

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

### Worauf legen wir Wert?

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

## Worauf legen wir Wert?

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

# Fragen?

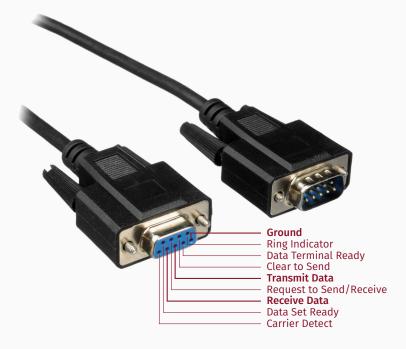
Zur Erinnerung: Übernächsten Dienstag (8. November) ist das Boot-Seminar

Serielle Schnittstelle

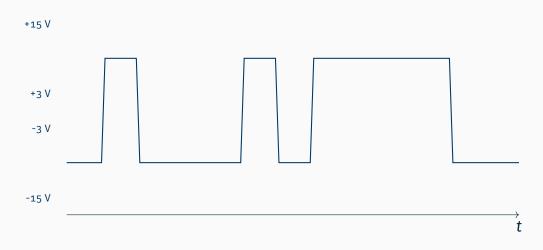


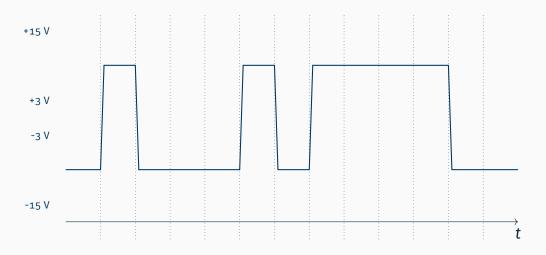


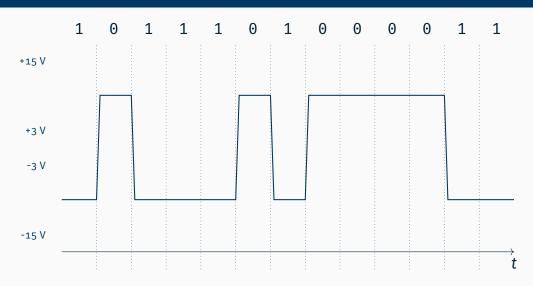
Quelle: B&H Photo Video

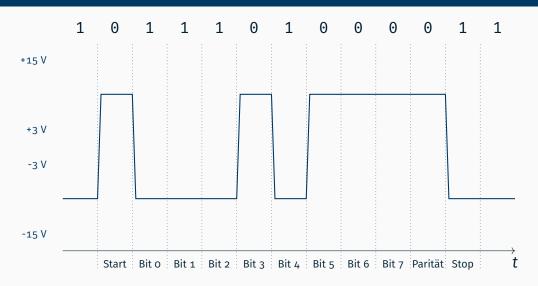


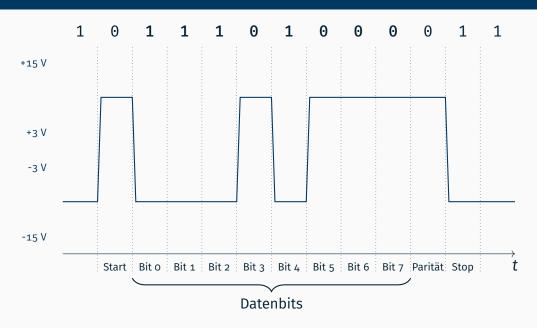
Quelle: B&H Photo Video











#### Serielle Schnittstelle

- 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
  - O 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 und lowlevel.eu

# Terminal



Quelle: vecchicomputer.com

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

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

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

<sup>\*</sup> wird nur selten unterstützt

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

```
\e[Am Attribute
\e[3Cm Vordergrundfarbe
\e[4Cm Hintergrundfarbe
```

- o schwarz
- 1 rot
- 2 grün
- 3 gelb
- 4 blau
- 5 magenta
- 6 cyan
- 7 weiß
- 9 Standardfarbe

- 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
```

- 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
```

- 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
```

- 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
```

■ Testen auf der Kommandozeile

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

- 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
```

■ Testen auf der Kommandozeile

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

```
Siehe auch http://www.termsys.demon.co.uk/vtansi.htm
https://en.wikipedia.org/wiki/ANSI_escape_code
```