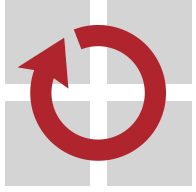
Übung zu Betriebssysteme

Ein- und Ausgabe

26. & 29. Oktober 2018

Andreas Ziegler
Bernhard Heinloth

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

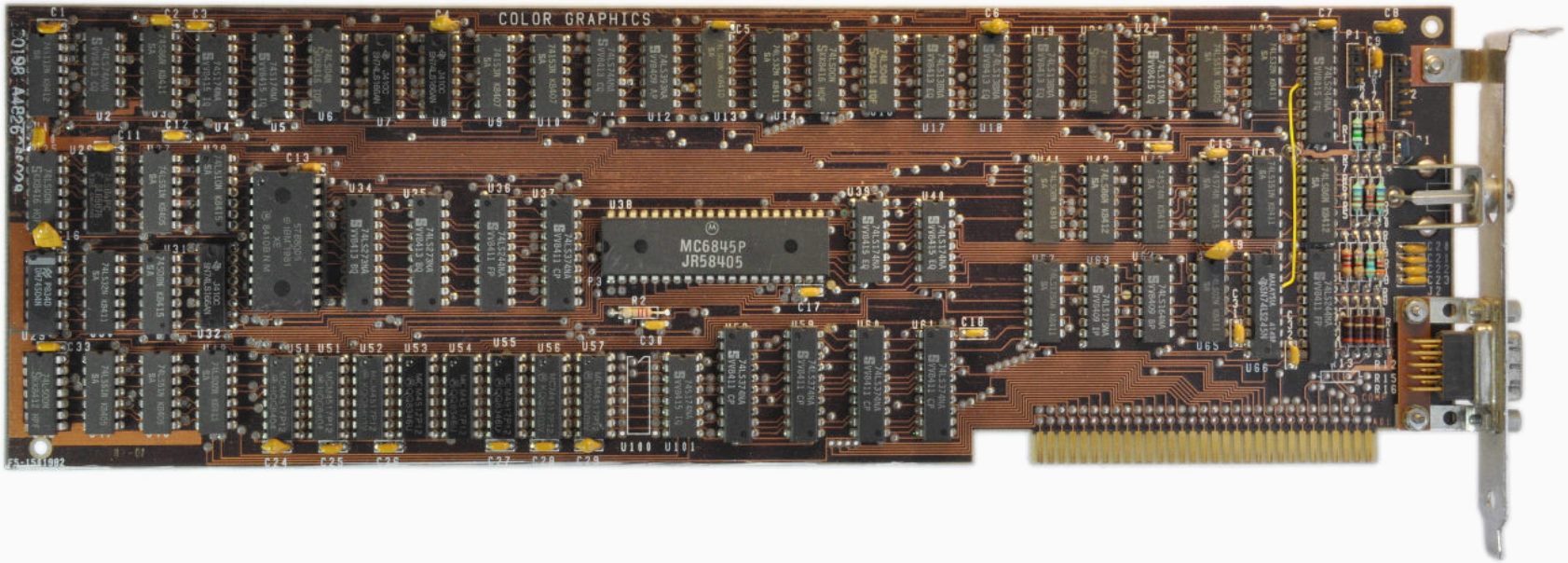


Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

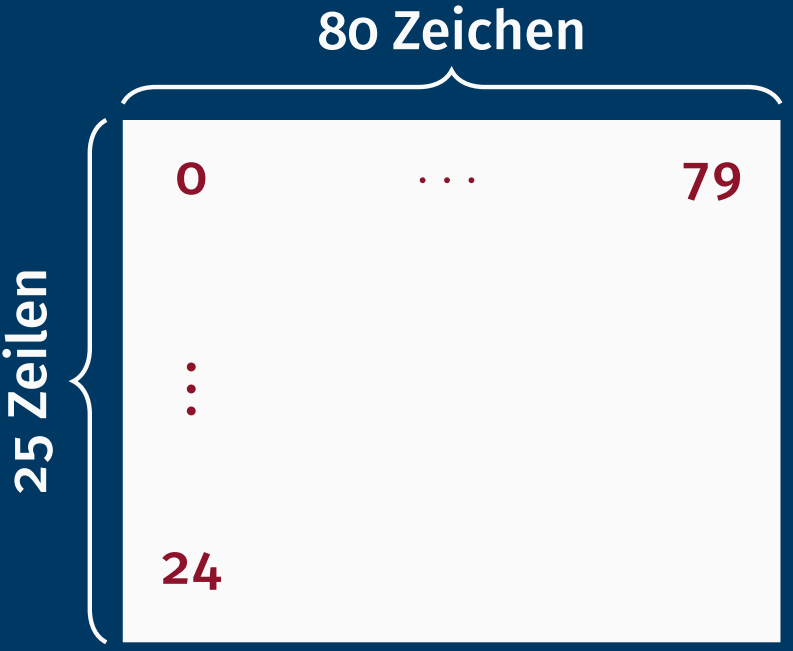
TECHNISCHE FAKULTÄT

Color Graphics Adapter



Quelle: Wikipedia

Der CGA Bildschirm ...



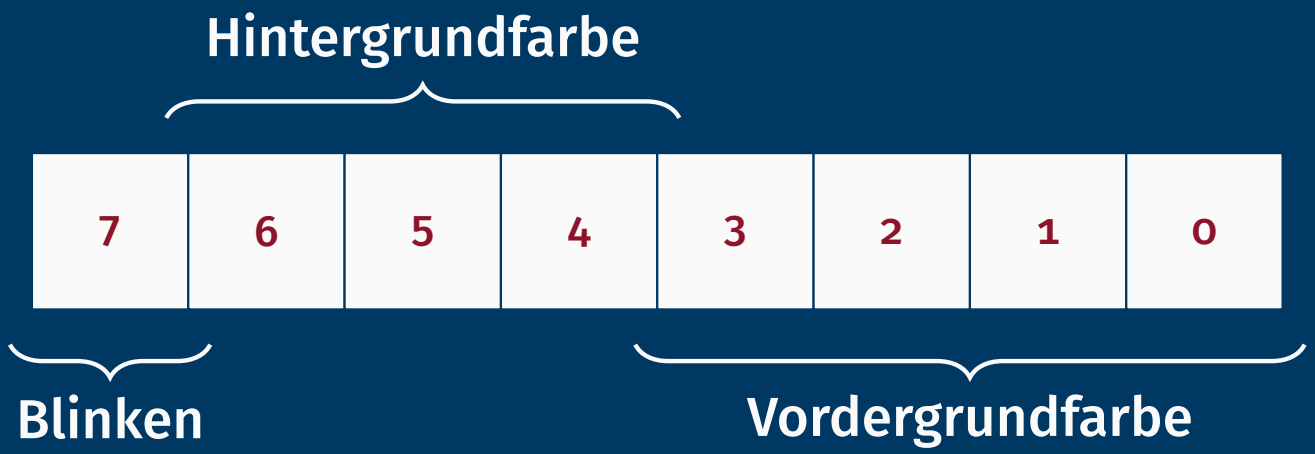
... zwei Bytes pro Zeichen ...

1. Byte: ASCII Zeichen
2. Byte: Darstellungsattribut

… zwei Bytes pro Zeichen …

1. Byte: ASCII Zeichen

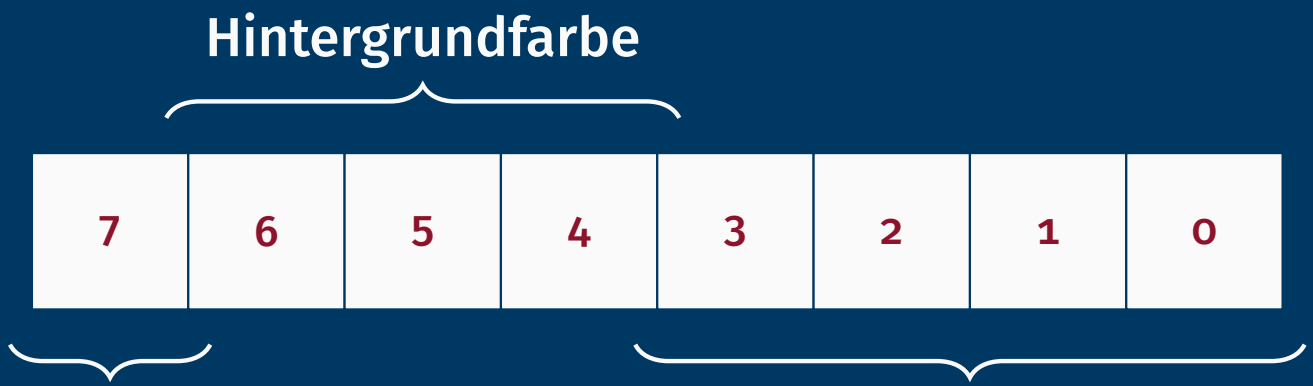
2. Byte: Darstellungsattribut



… zwei Bytes pro Zeichen …

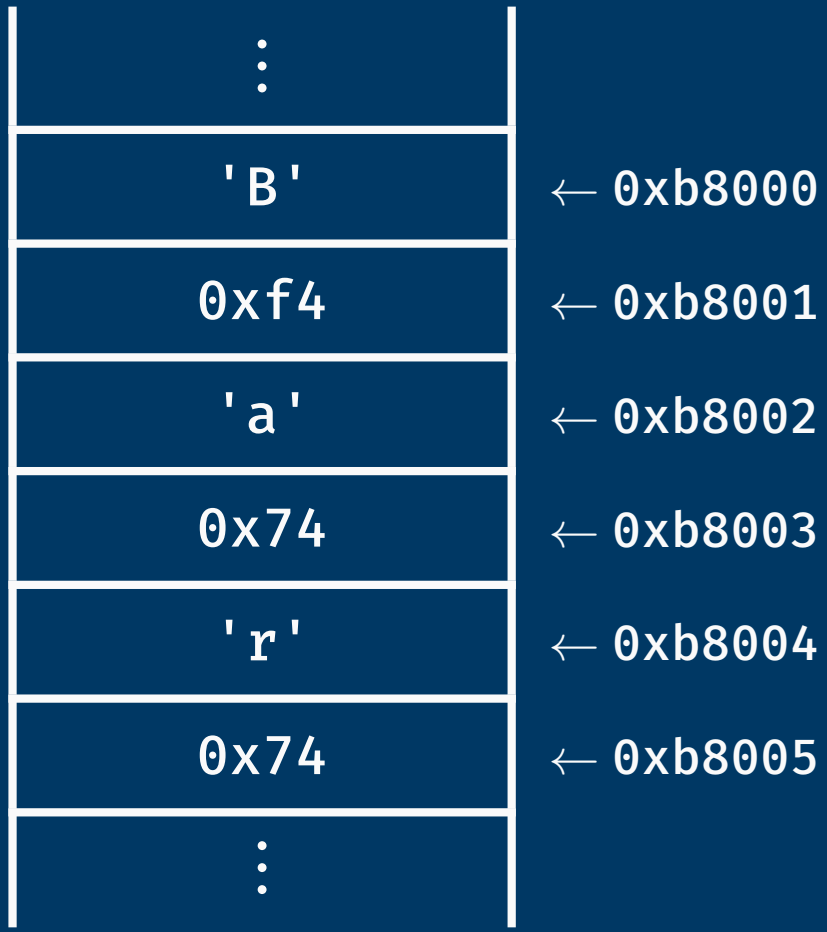
1. Byte: ASCII Zeichen

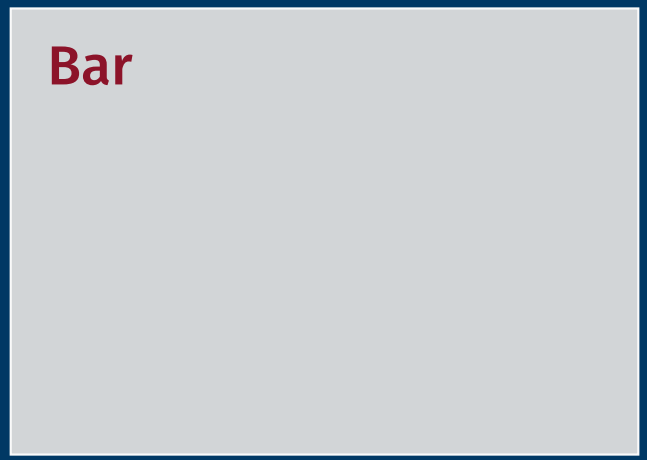
2. Byte: Darstellungsattribut



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

...eingeblendet im Arbeitsspeicher ab `0xb8000`




Bar

Rekapitulation: Bit~~schubsen~~operationen

`&`	Konjunktion (logisches UND)
`|`	Disjunktion (logisches ODER)
`^`	exklusives ODER
`~`	Einerkomplement (logische Negation)
`<<`	verschieben nach links (Multiplikation mit 2)
`>>`	verschieben nach rechts (Division durch 2)

Ansprechen von einzelnen Bits

- Bitmasken und logischen Bitoperationen

Ansprechen von einzelnen Bits

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

```
struct cga_attrib {
    unsigned char fg : 4,
                  bg : 3,
                  bl : 1;
};

struct cga_attrib a;
a.fg = 4; // rot
a.bg = 7; // hellgrau
a.bl = 1; // blinken
```

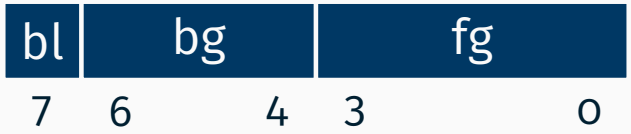
Ansprechen von einzelnen Bits

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

```
struct cga_attrib {
    unsigned char fg : 4,
                  bg : 3,
                  bl : 1;
};

struct cga_attrib a;
a.fg = 4; // rot
a.bg = 7; // hellgrau
a.bl = 1; // blinken
```

Layout mit GCC auf x86:



Einschub: Füllen von Strukturen

```
struct Test {
    char foo[3];
    int bar;
    bool baz;
};

cout << sizeof(Test);
```

Einschub: Füllen von Strukturen

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

cout << sizeof(Test); // "12"
```

Einschub: Füllen von Strukturen

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

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

Einschub: Füllen von Strukturen

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

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

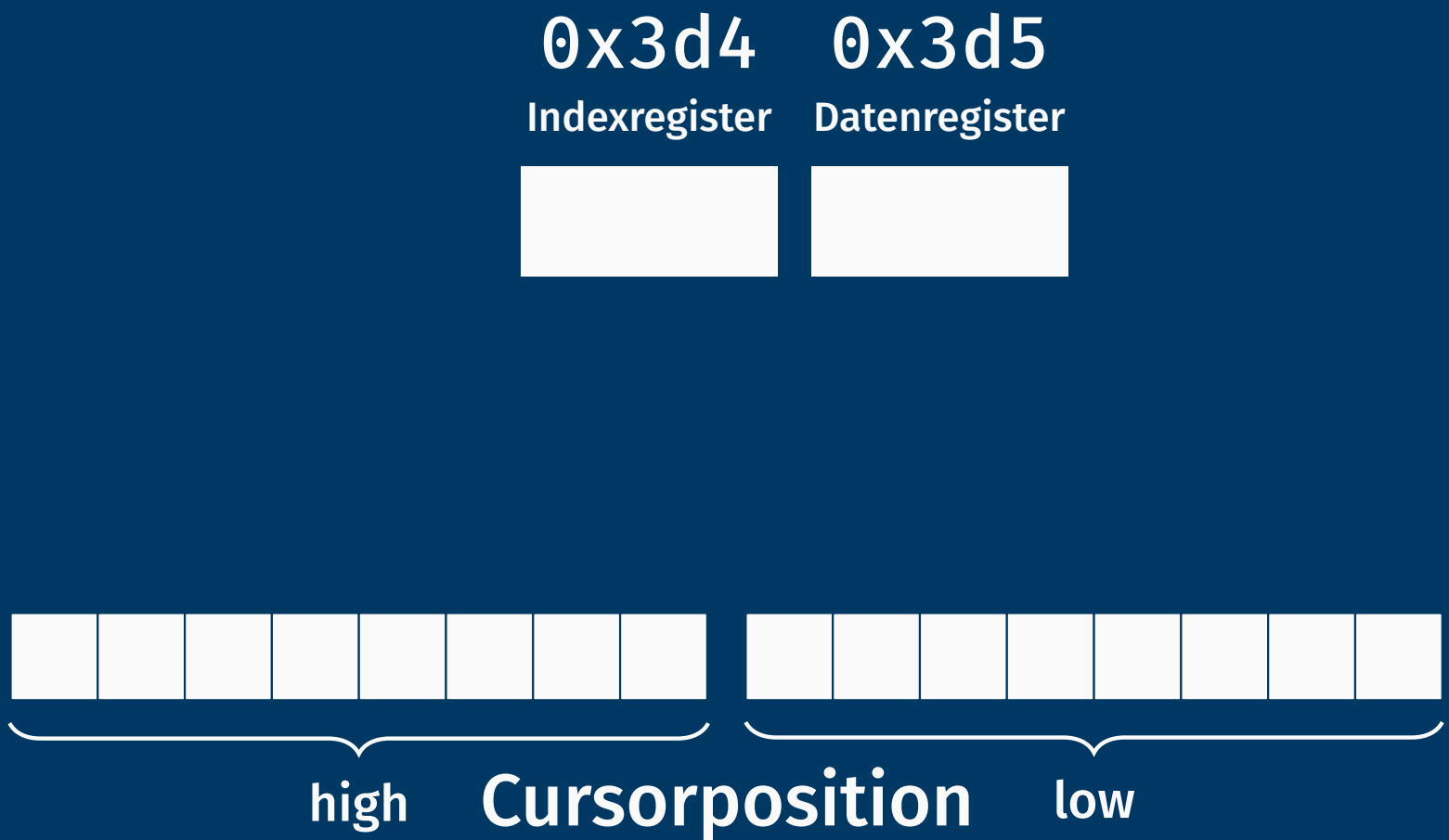
- CGA hat 18 Steuerregister mit je 8 Bit
- Position in Register 14 (high) und 15 (low)

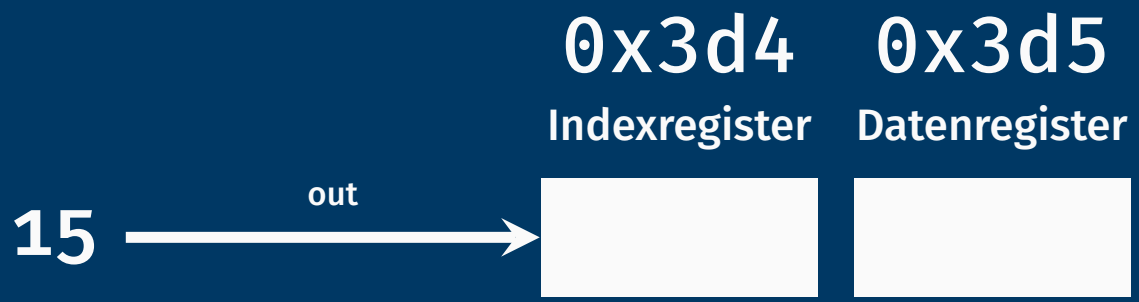
0	Horizontal Total	w
1	Horizontal Displayed	w
2	H. Sync Position	w
3	H. Sync Width	w
4	Vertical Total	w
5	V. Total Adjust	w
6	Vertical Displayed	w
7	V. Sync Position	w
8	Interlace Mode	w
9	Max Scan Line Address	w
10	Cursor Start	w
11	Cursor End	w
12	Start Address (high)	w
13	Start Address (low)	w
14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

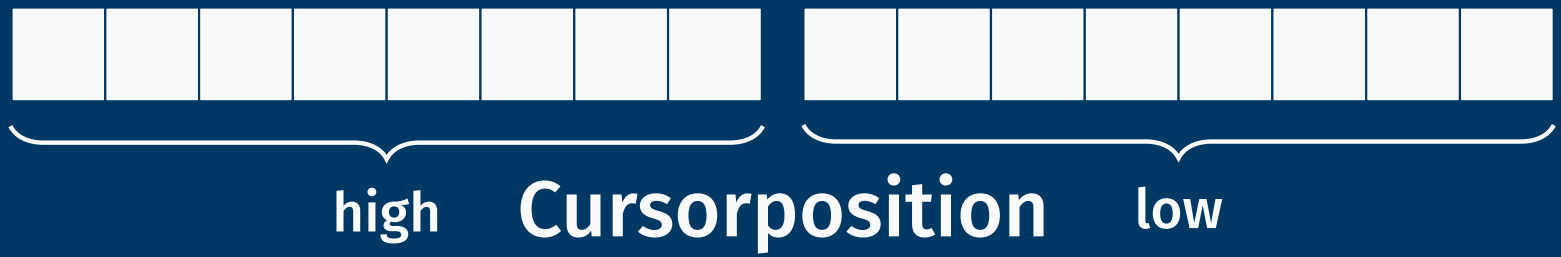
… oder Cursorposition aus Hardware auslesen

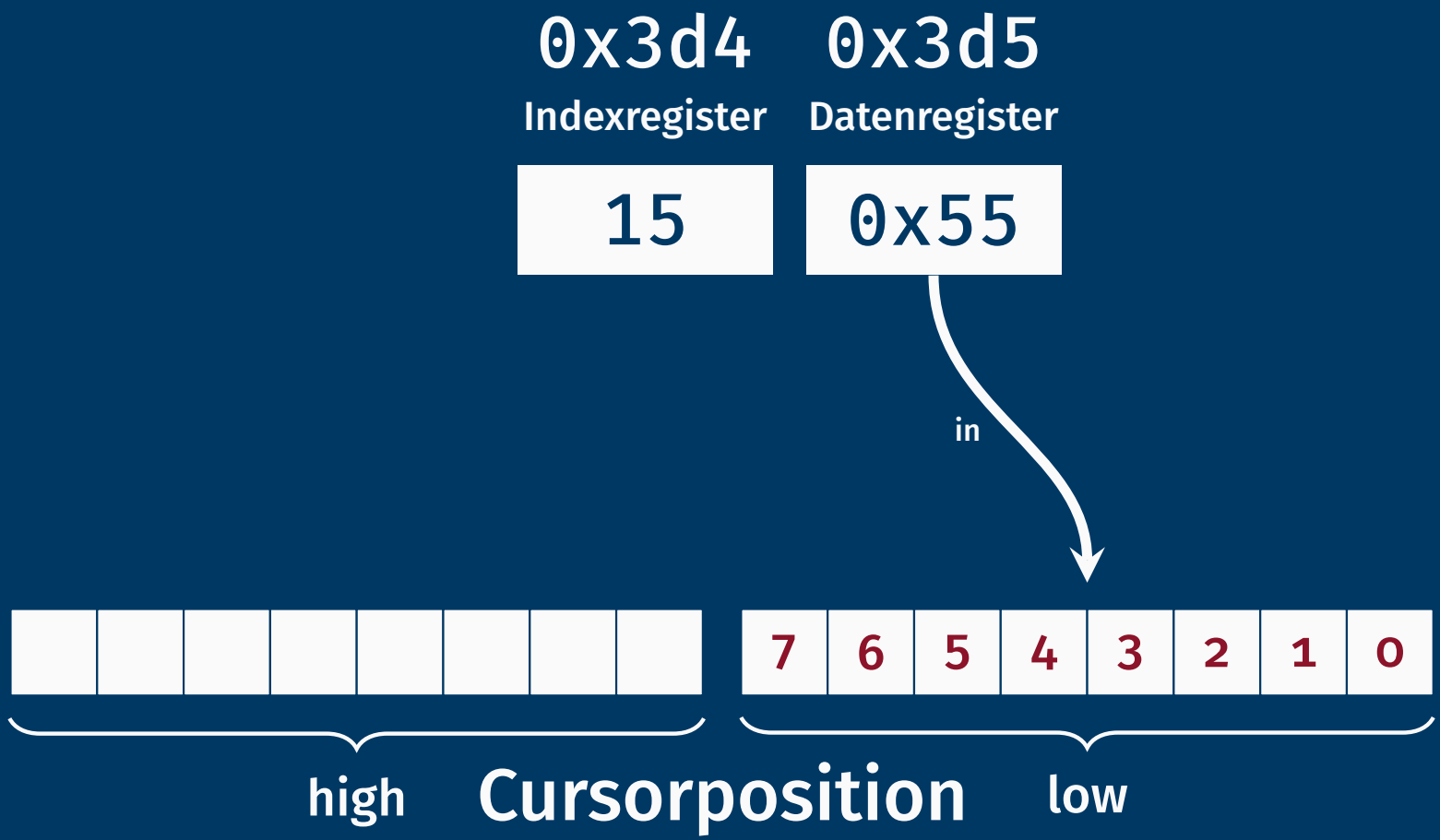
- CGA hat 18 Steuerregister mit je 8 Bit
- Position in Register 14 (high) und 15 (low)
- Nur indirekter Zugriff über Index- (`0x3d4`) und Datenregister (`0x3d5`)

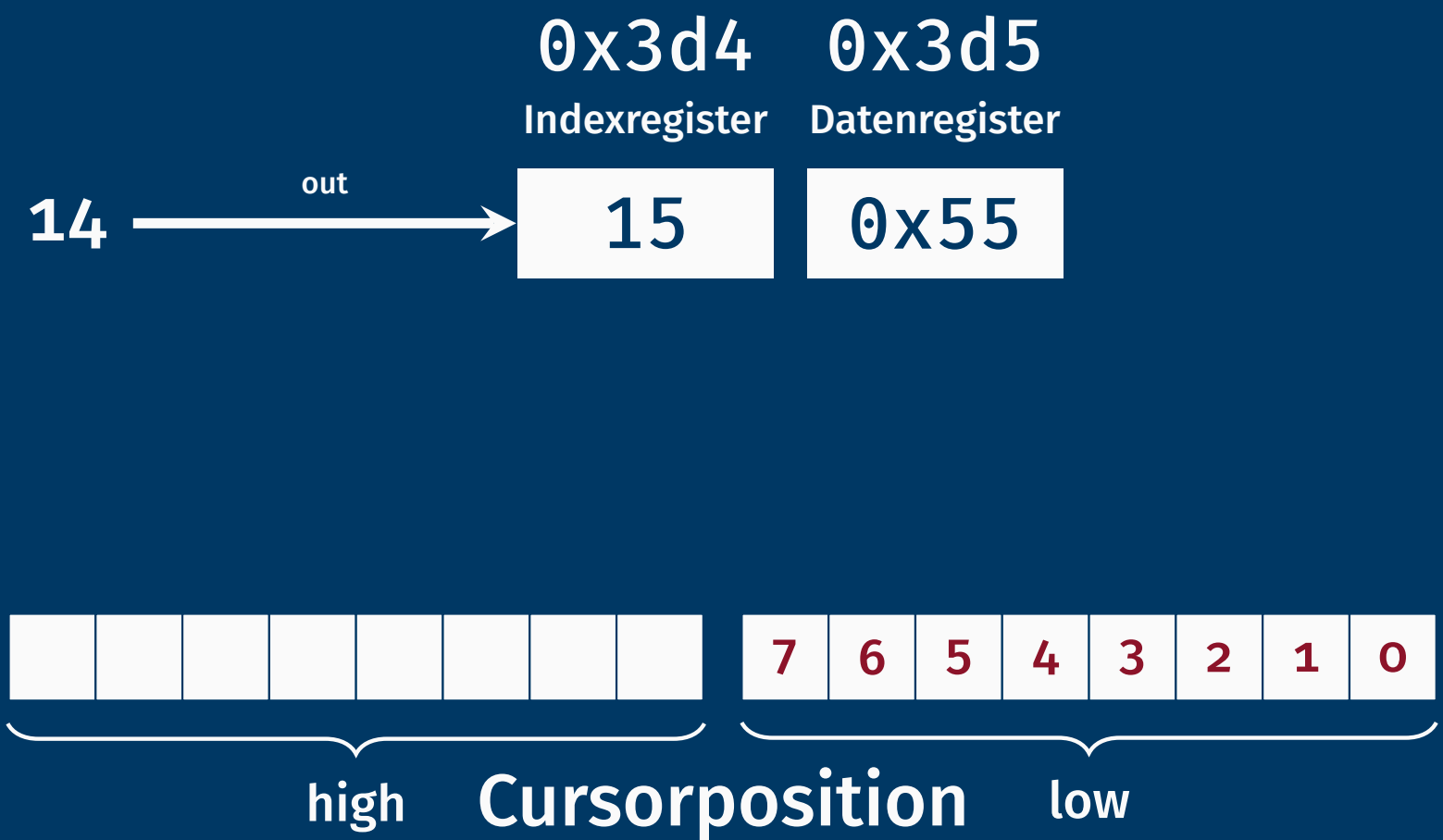
0	Horizontal Total	w
1	Horizontal Displayed	w
2	H. Sync Position	w
3	H. Sync Width	w
4	Vertical Total	w
5	V. Total Adjust	w
6	Vertical Displayed	w
7	V. Sync Position	w
8	Interlace Mode	w
9	Max Scan Line Address	w
10	Cursor Start	w
11	Cursor End	w
12	Start Address (high)	w
13	Start Address (low)	w
14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

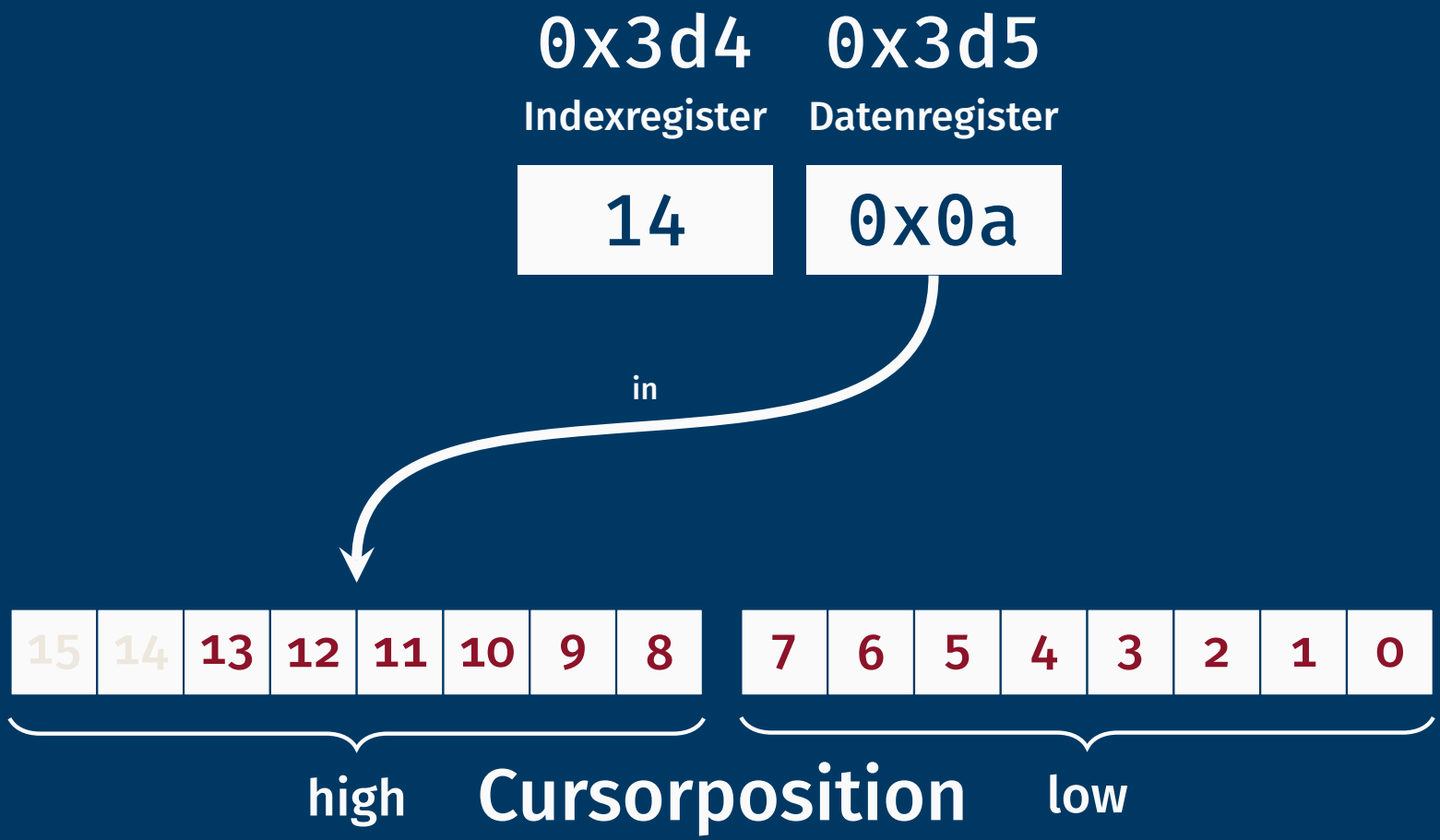

0x3d4
0x3d5
Indexregister
Datenregister
high
Cursorposition
low


0x3d4
0x3d5
Indexregister
Datenregister
15
out

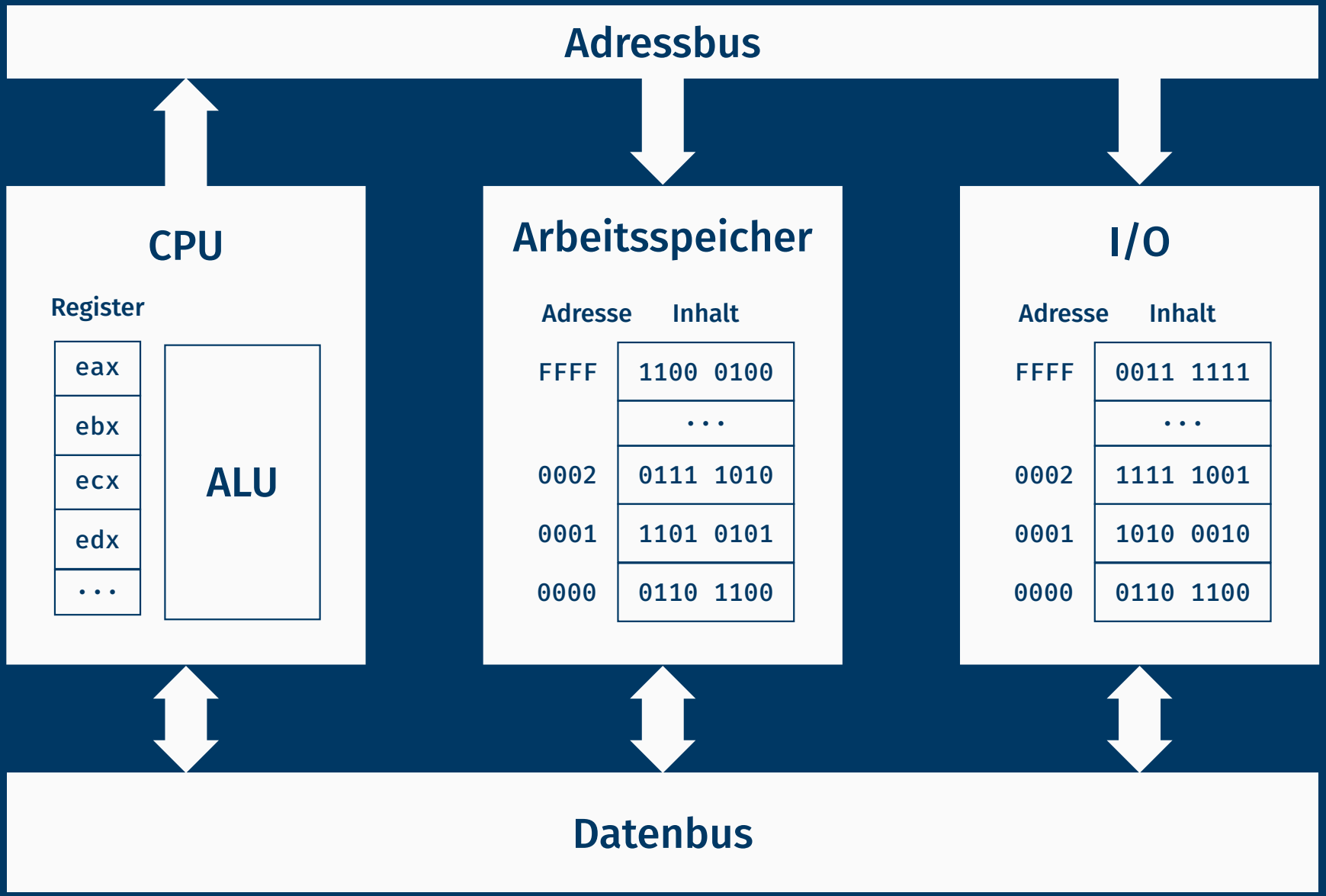

high
Cursorposition
low

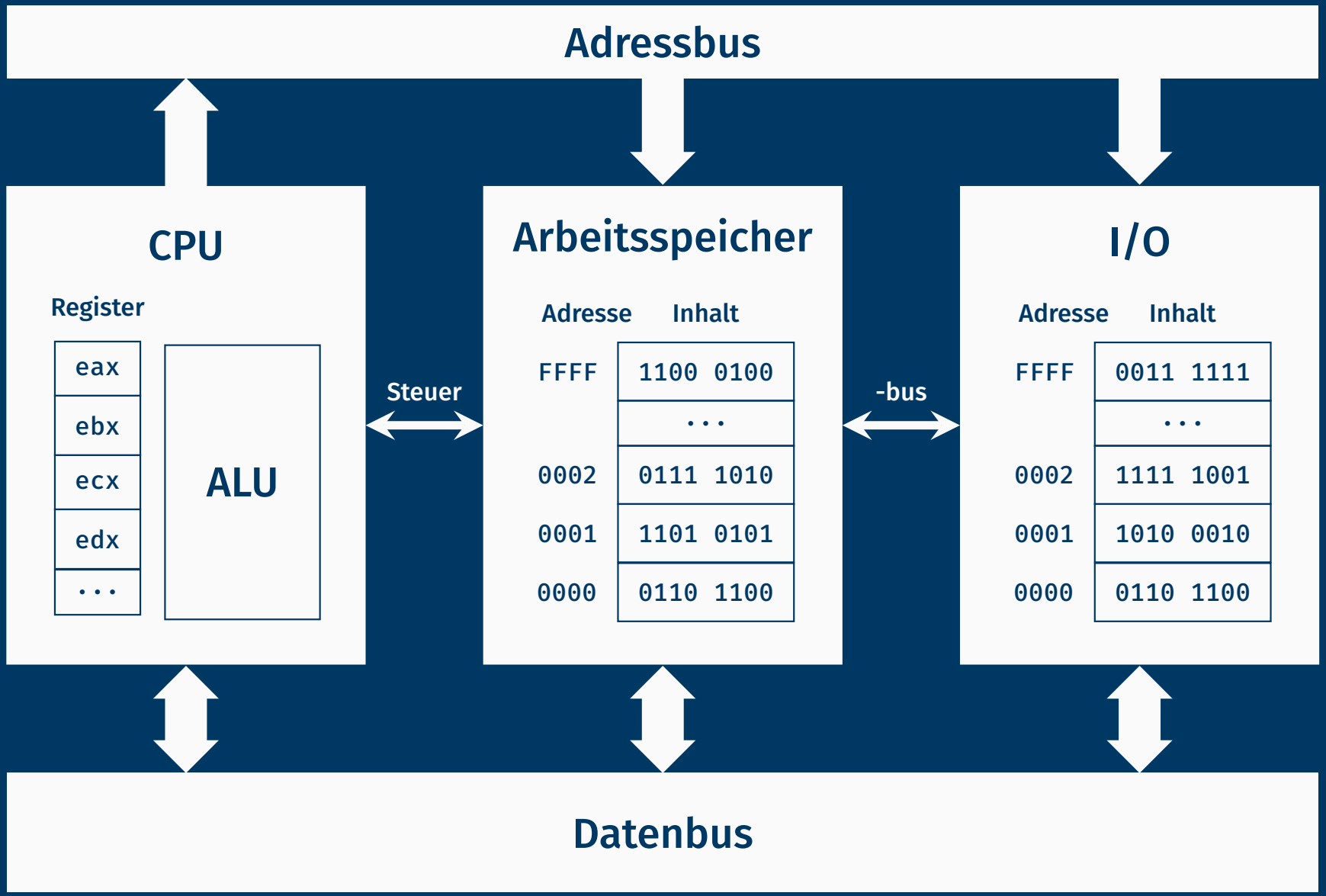

0x3d4
0x3d5
Indexregister
Datenregister
15
0x55
in
7 6 5 4 3 2 1 0
high
Cursorposition
low


0x3d4
0x3d5
Indexregister
Datenregister
14
out
15
0x55
7 6 5 4 3 2 1 0
high
Cursorposition
low


0x3d4
0x3d5
Indexregister
Datenregister
14
0x0a
in
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
high
Cursorposition
low

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

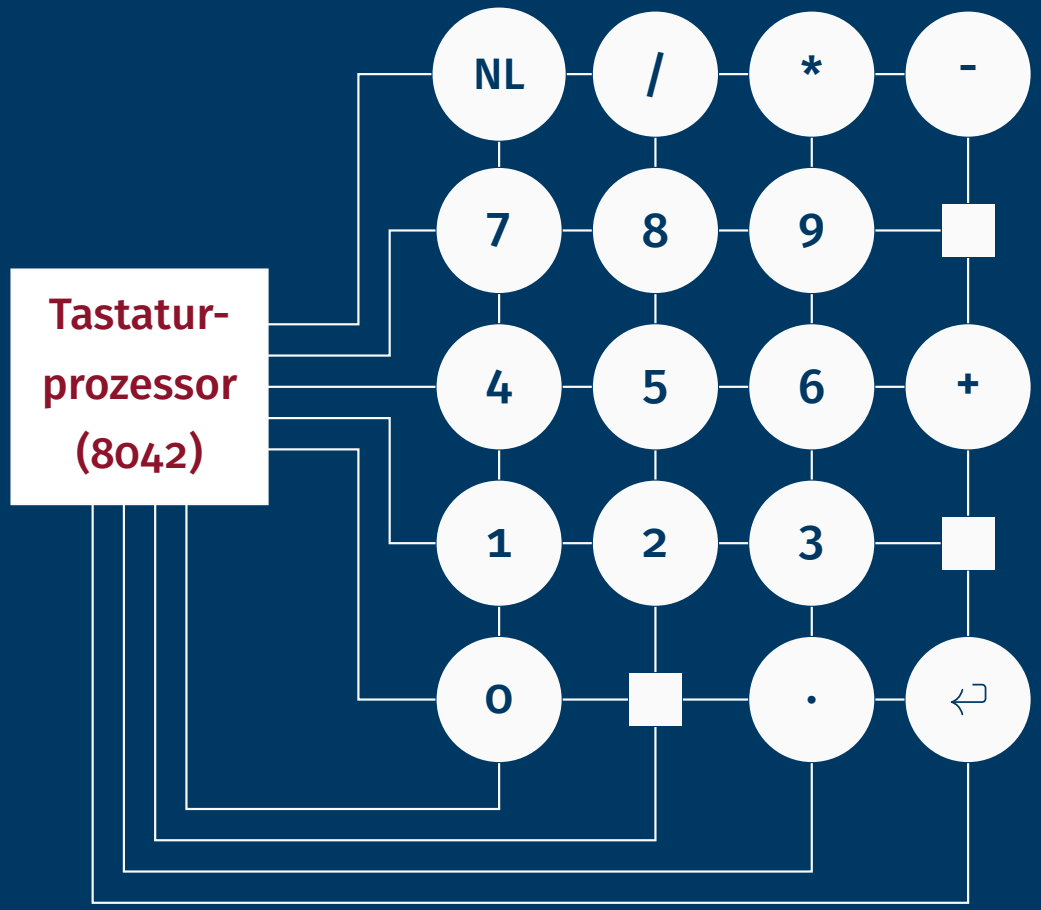

Adressbus
CPU
Register
eax
ebx
ecx
edx
...
ALU
Arbeitsspeicher
Adresse
Inhalt
FFFF
1100 0100
...
0002
0111 1010
0001
1101 0101
0000
0110 1100
I/O
Adresse
Inhalt
FFFF
0011 1111
...
0002
1111 1001
0001
1010 0010
0000
0110 1100
Datenbus

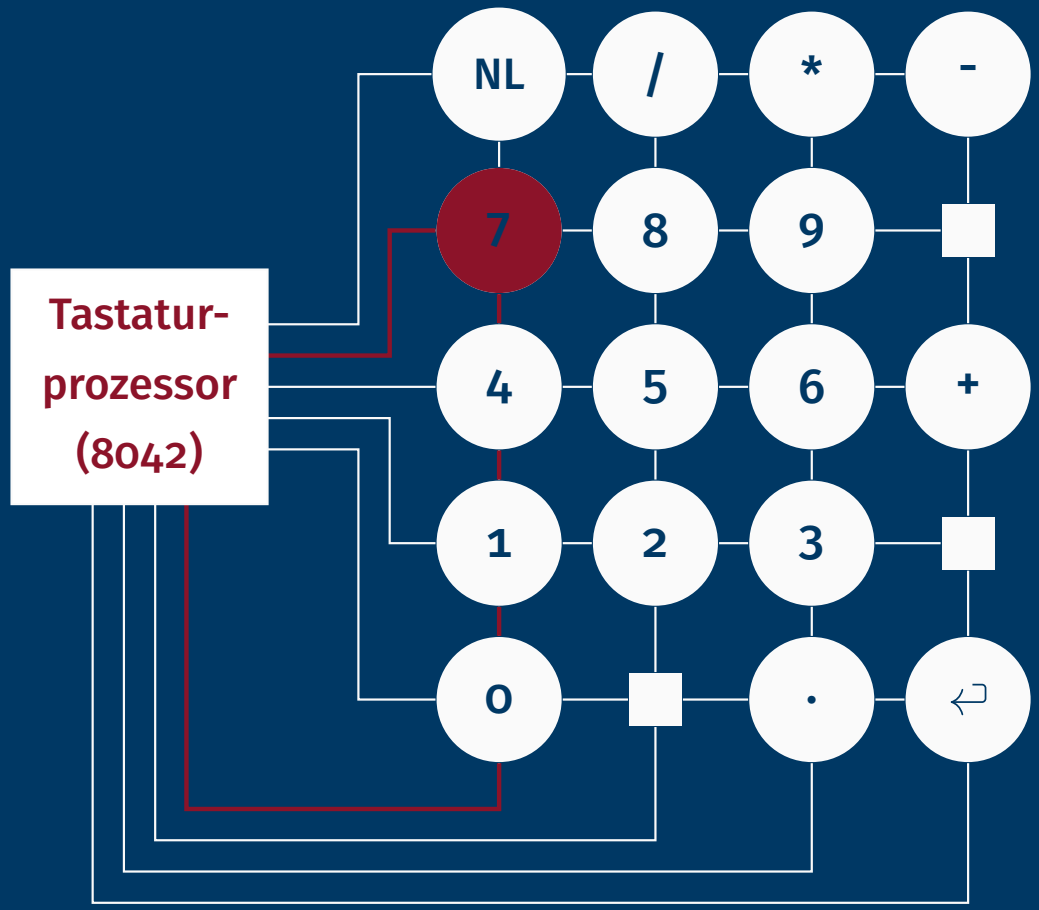

Adressbus
CPU
Register
eax
ebx
ecx
edx
...
ALU
Steuer
Arbeitsspeicher
Adresse
Inhalt
FFFF
1100 0100
...
0002
0111 1010
0001
1101 0101
0000
0110 1100
-bus
I/O
Adresse
Inhalt
FFFF
0011 1111
...
0002
1111 1001
0001
1010 0010
0000
0110 1100
Datenbus

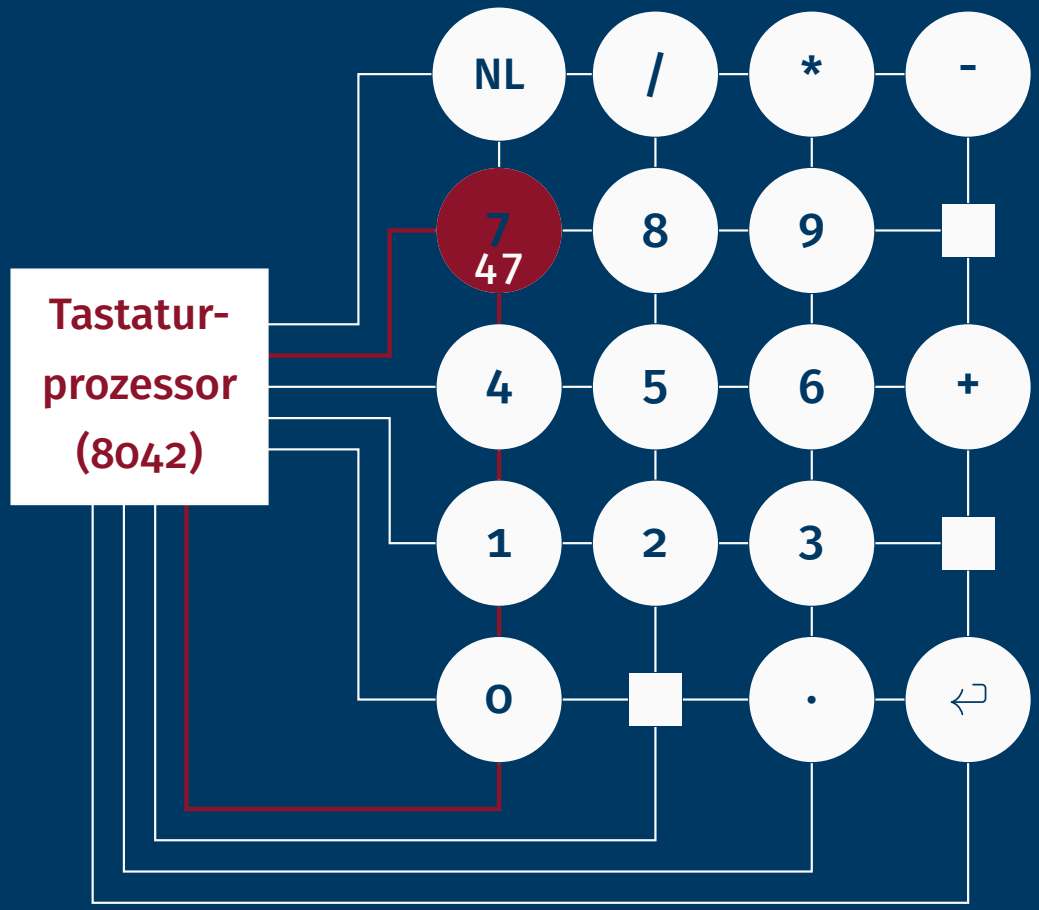
Tastatur

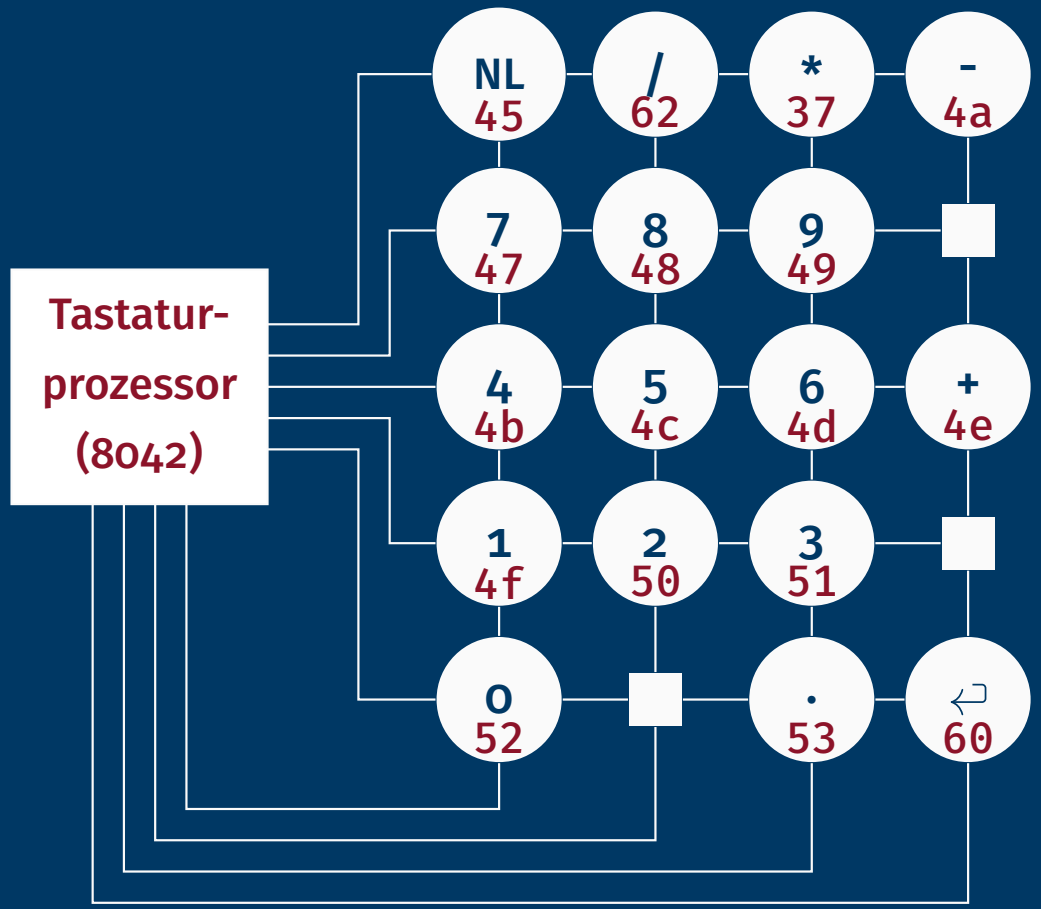
Quelle: Golem

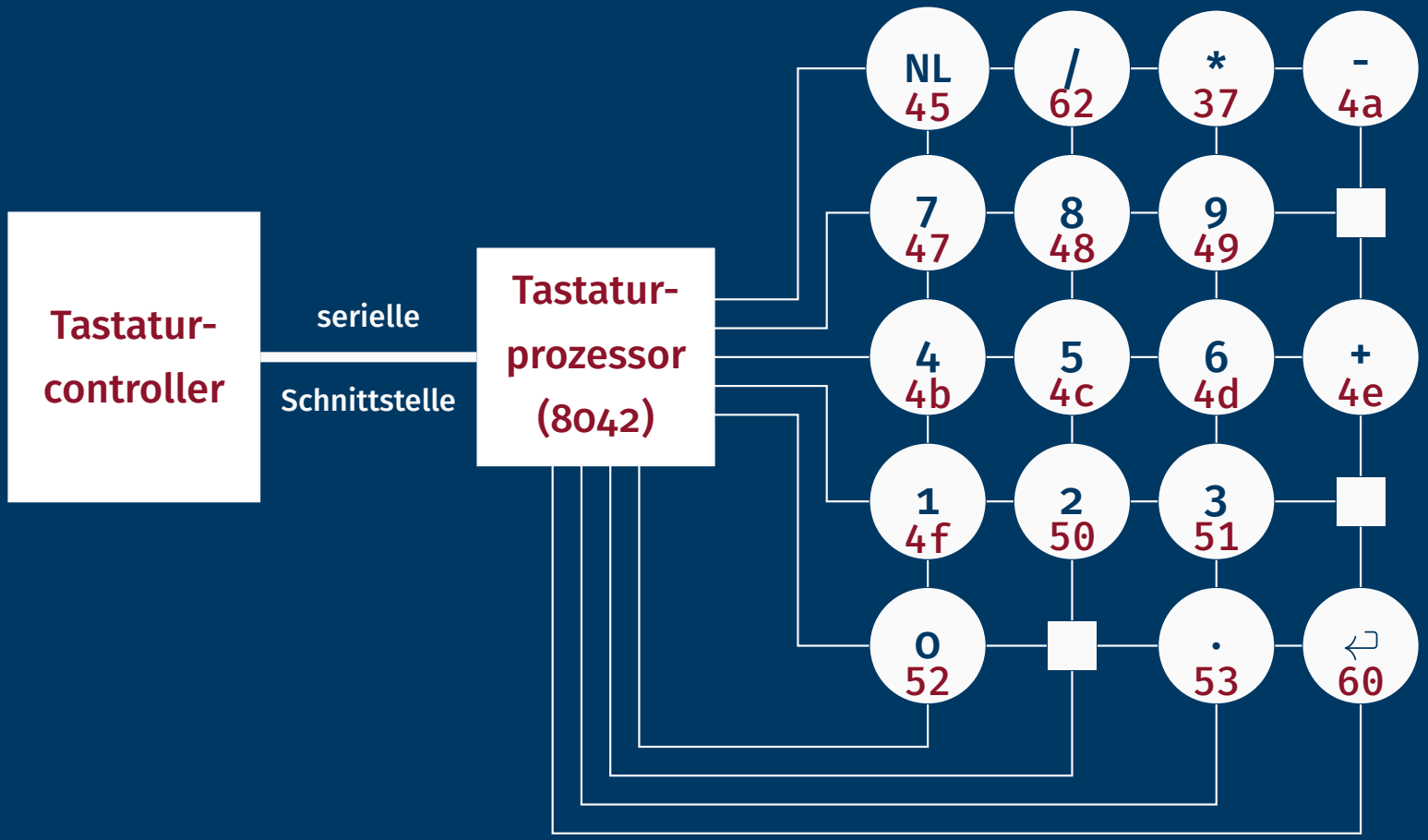

NL
/
*
-
7
8
9
4
5
6
+
1
2
3
0
.

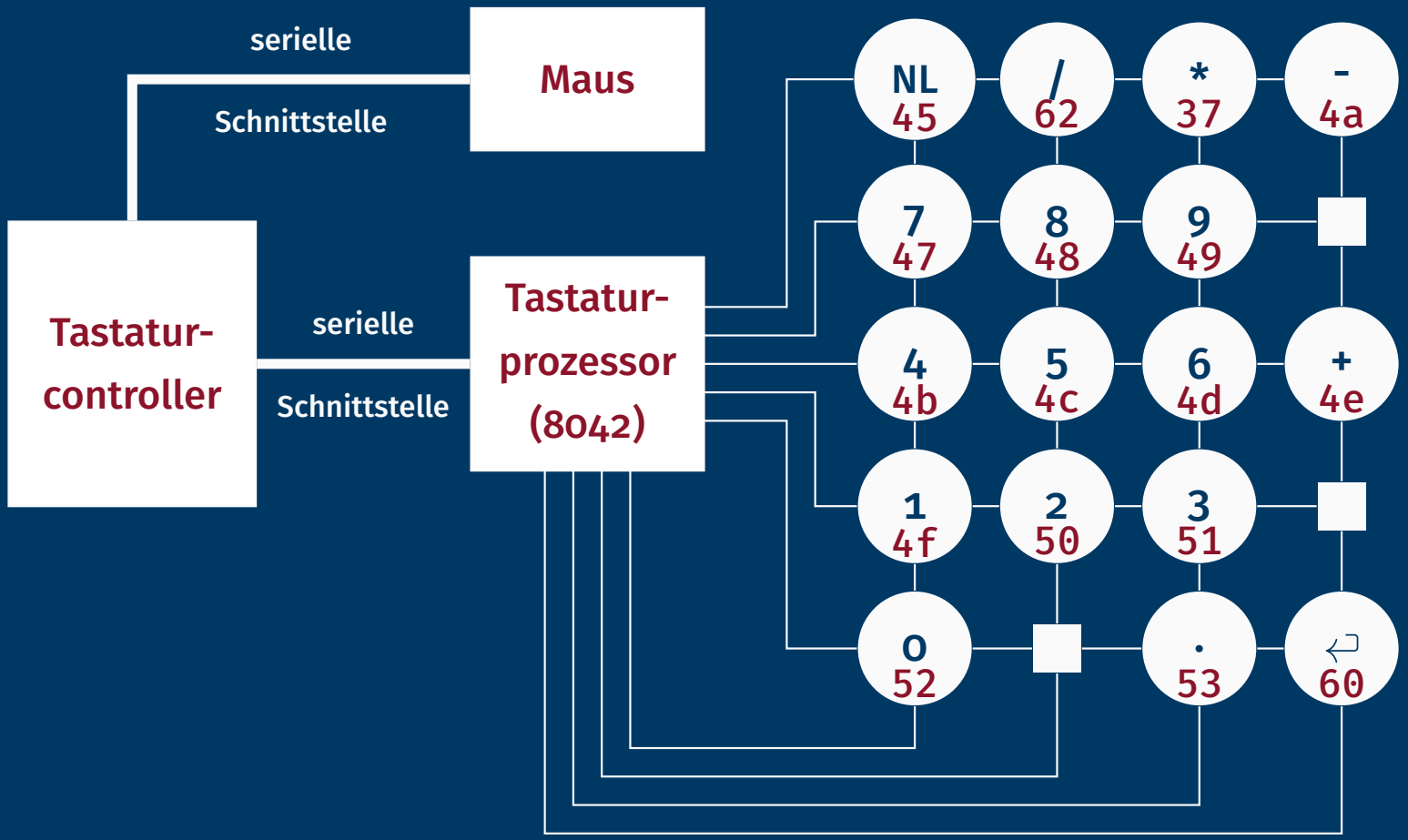

Tastatur-
prozessor
(8042)
NL
/
*
-
7
8
9
4
5
6
+
1
2
3
0
.


NL
/
*
-
7
8
9
Tastatur-
prozessor
(8042)
4
5
6
+
1
2
3
0
.


NL
/
*
-
7
47
8
9
Tastatur-
prozessor
(8042)
4
5
6
+
1
2
3
0
.


Tastatur-
prozessor
(8042)
NL
45
/
62
*
37
-
4a
7
47
8
48
9
49
4
4b
5
4c
6
4d
+
4e
1
4f
2
50
3
51
0
52
.
53
60


Tastatur-
controller
serielle
Schnittstelle
Tastatur-
prozessor
(8042)
NL
45
/
62
*
37
-
4a
7
47
8
48
9
49
4
4b
5
4c
6
4d
+
4e
1
4f
2
50
3
51
0
52
.
53
60


serielle
Schnittstelle
Maus
Tastatur-
controller
serielle
Schnittstelle
Tastatur-
prozessor
(8042)
NL
45
/
62
*
37
-
4a
7
47
8
48
9
49
4
4b
5
4c
6
4d
+
4e
1
4f
2
50
3
51
0
52
.
53
60

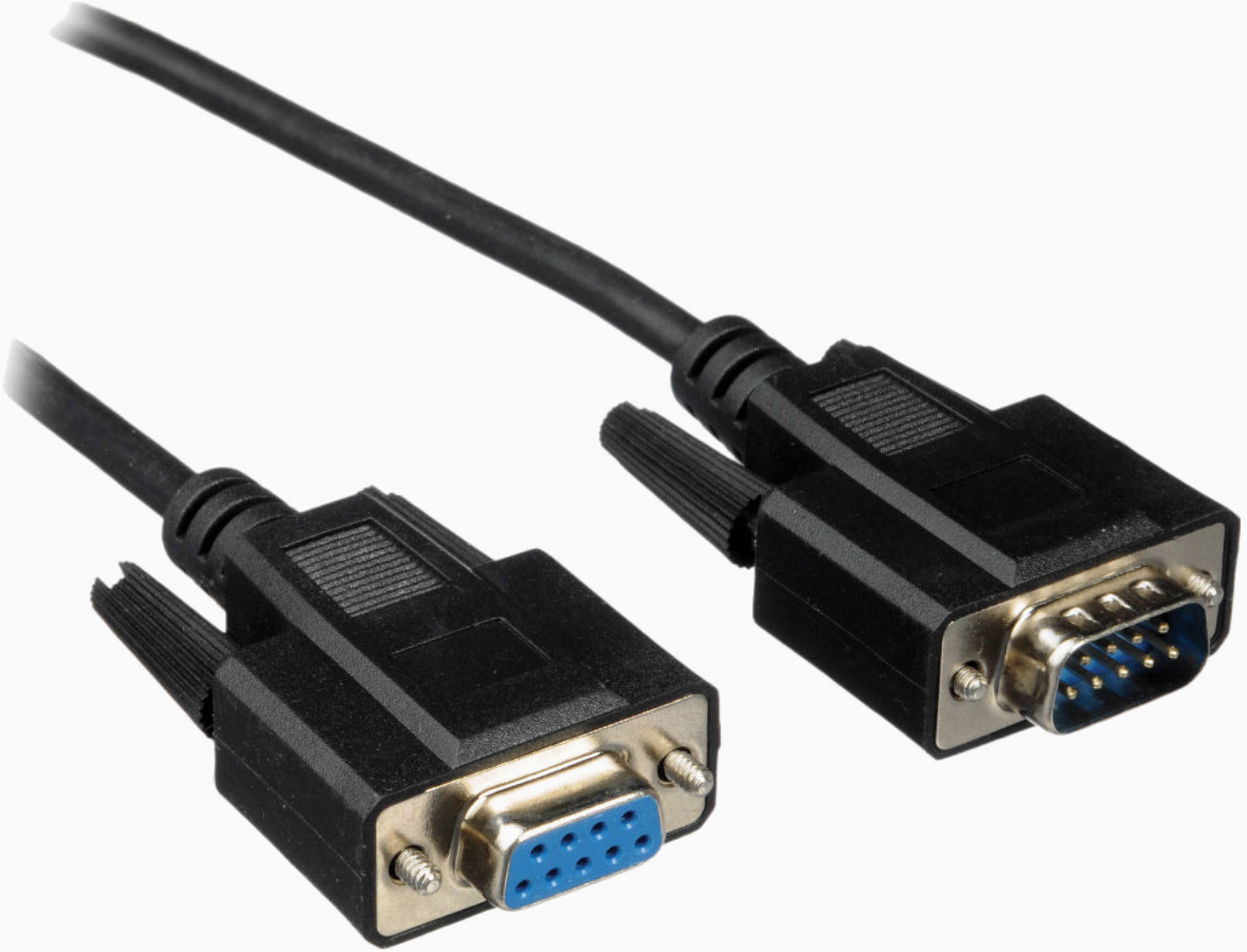
ASCII: Darstellung von Zeichen, 8 Bit

Scancode: Tastenkennung, 7 Bit

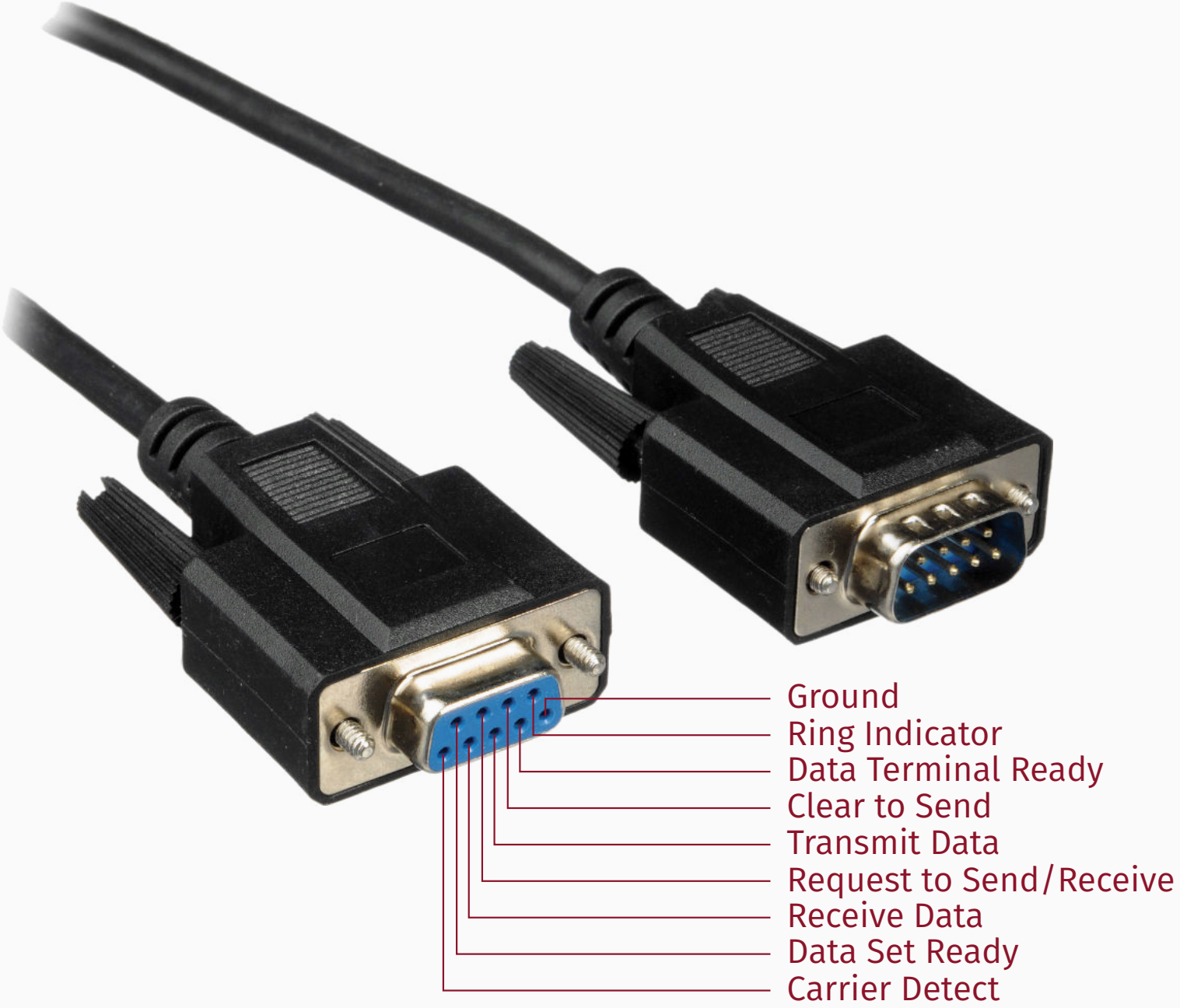
Makecode: Tastendruck (Scancode + `0x80`)

Breakcode: Taste loslassen

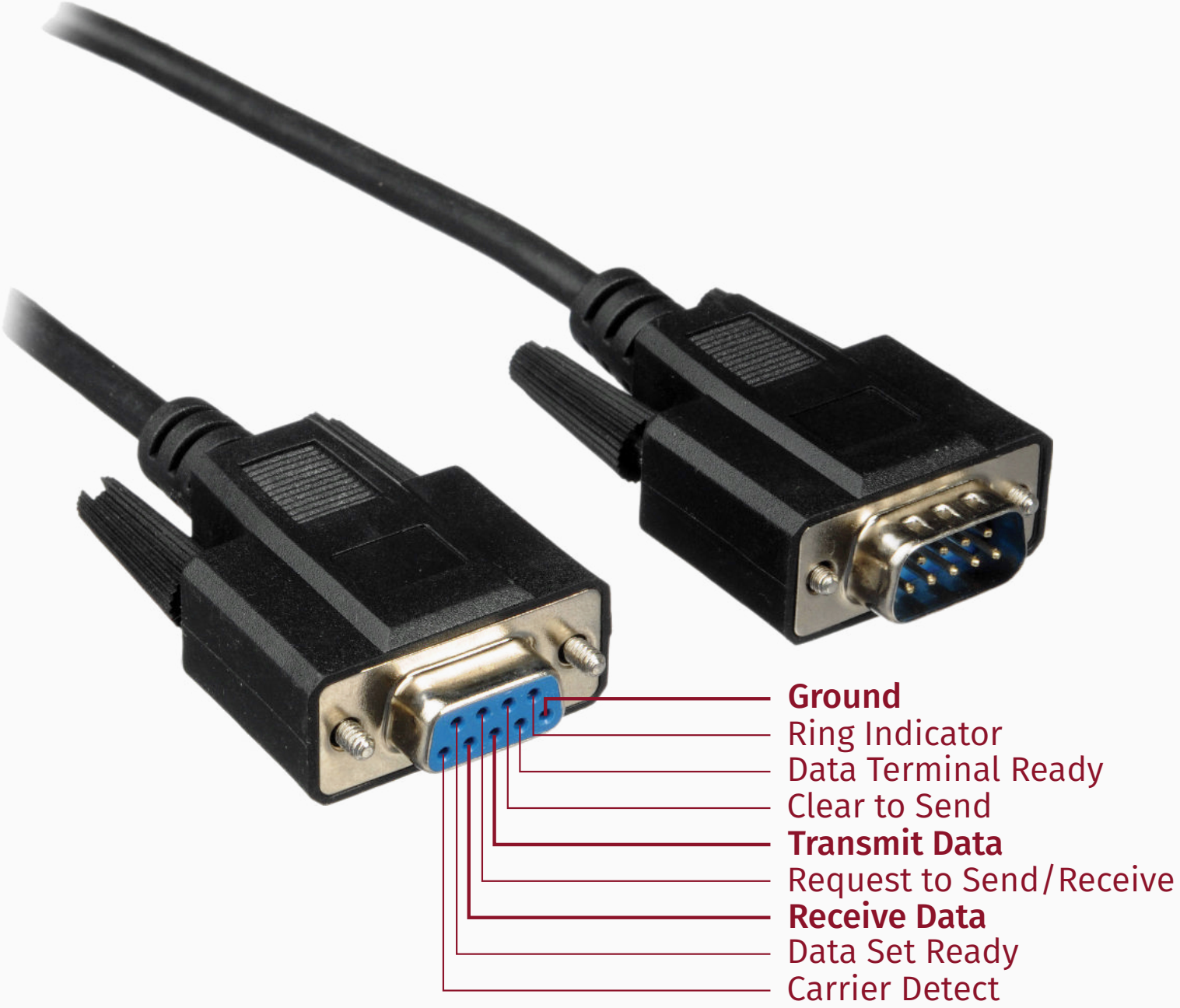
Serielle Schnittstelle



Quelle: B&H Photo Video

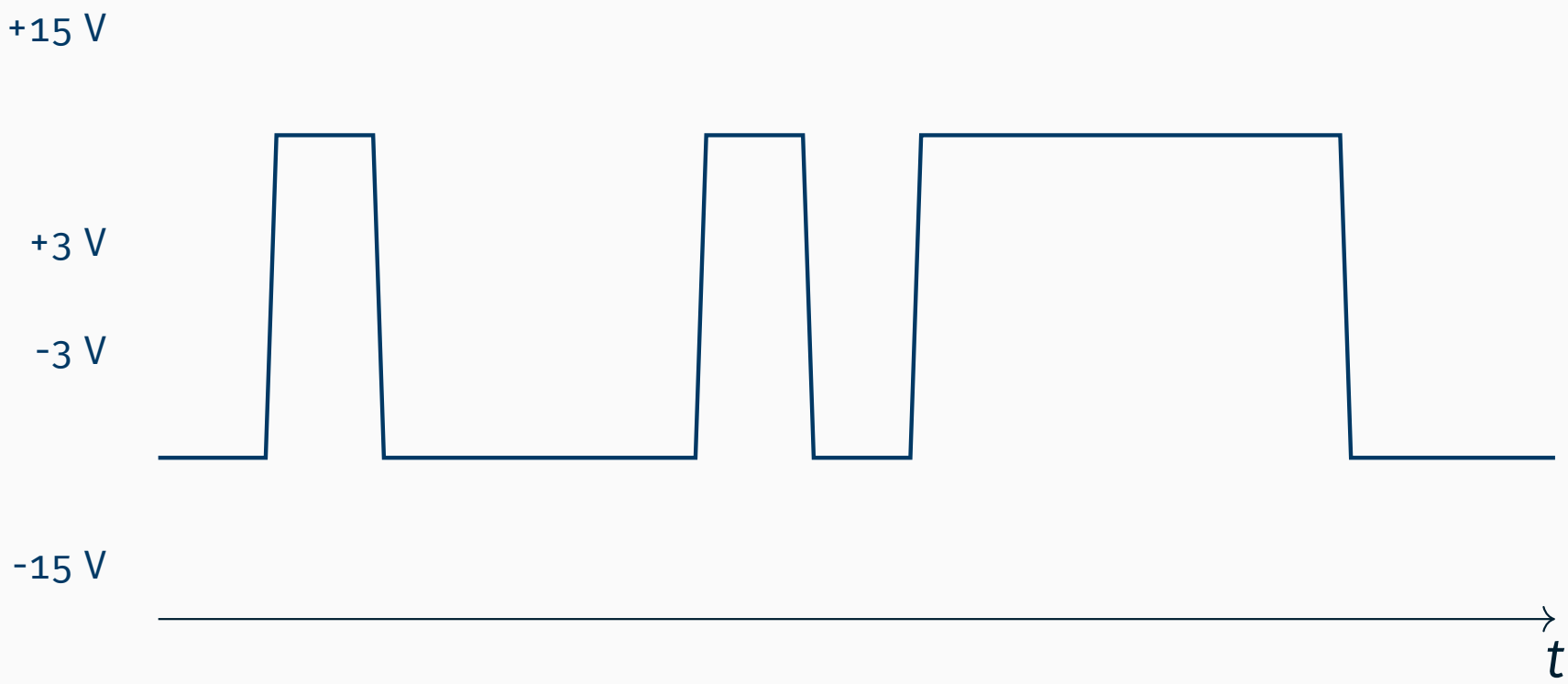


Quelle: B&H Photo Video

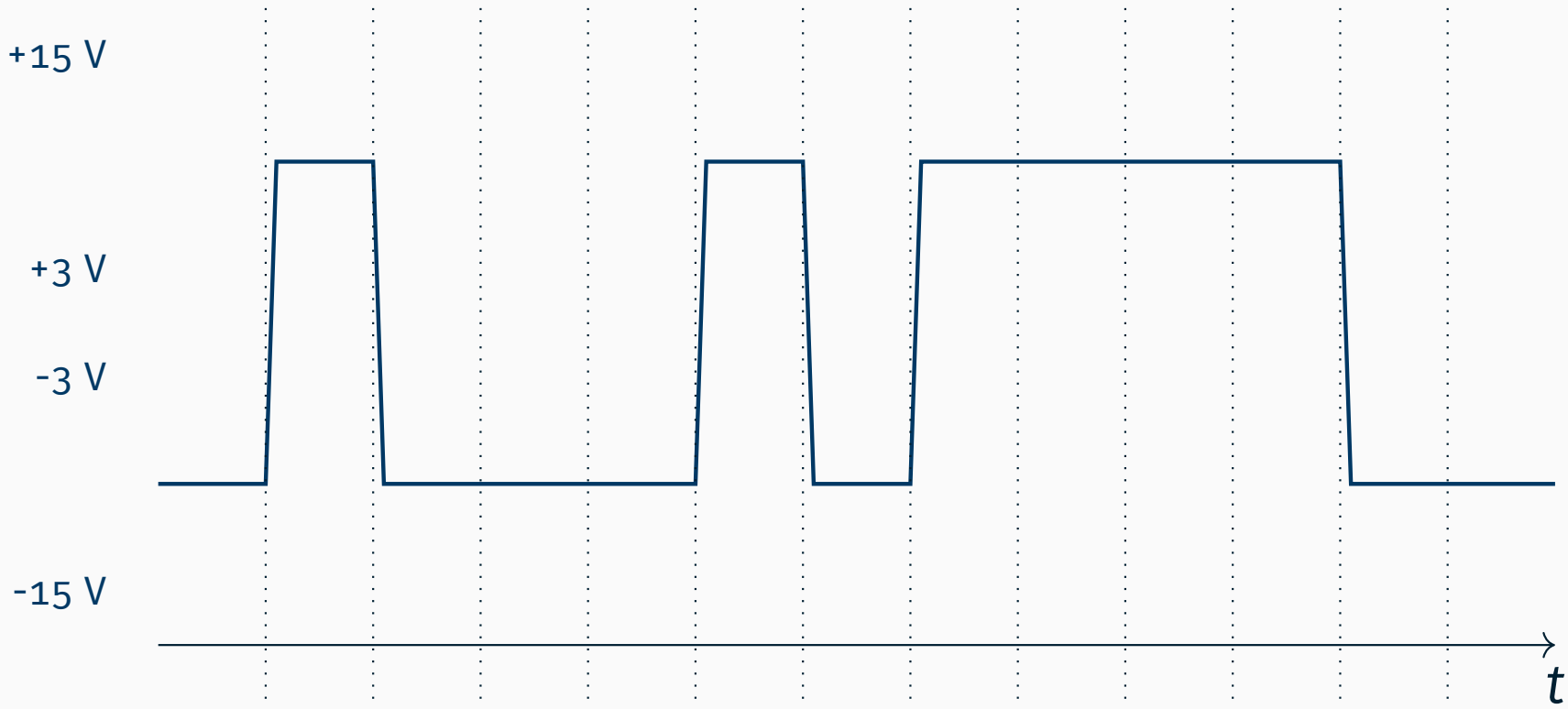


Quelle: B&H Photo Video

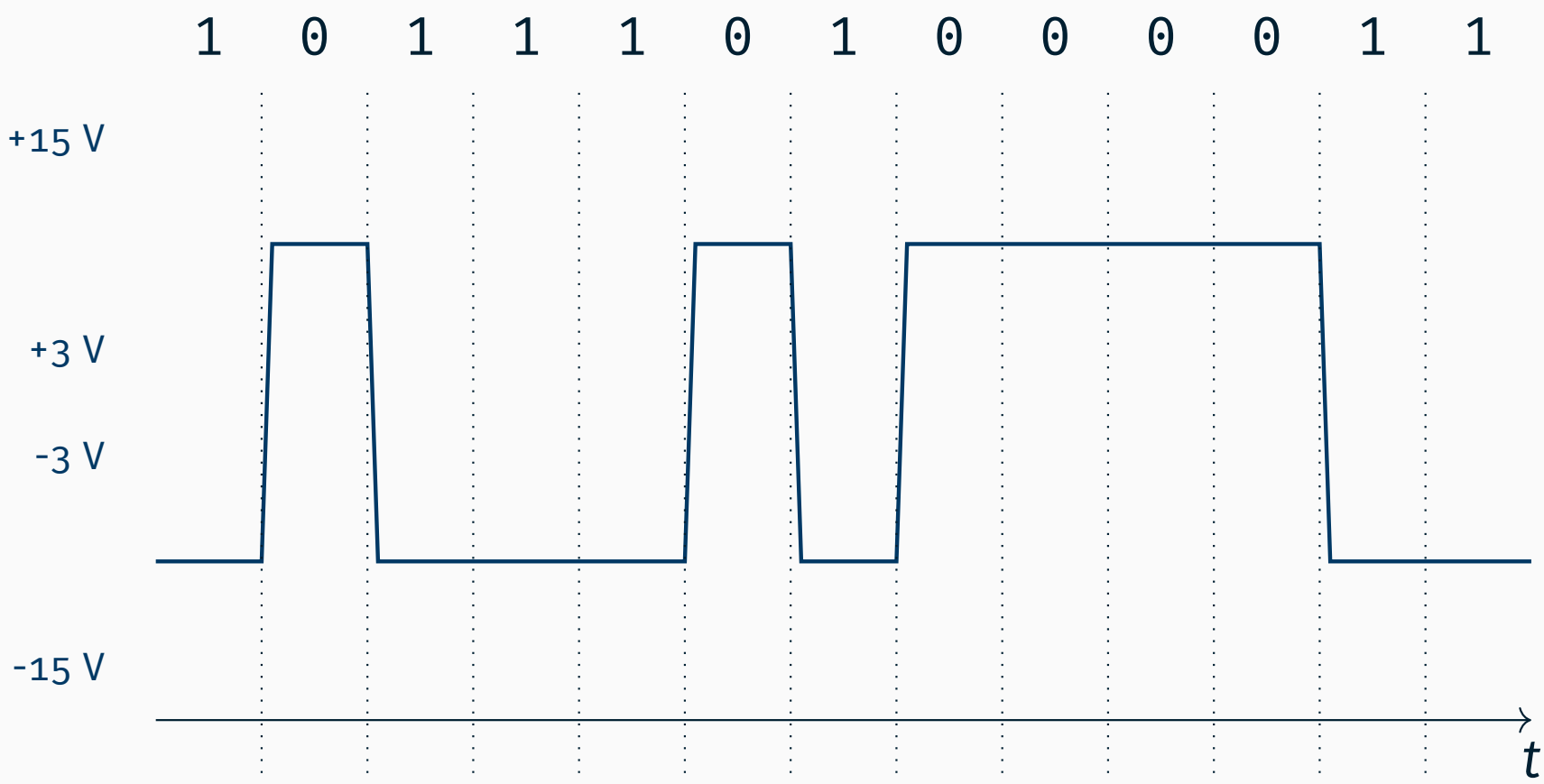
RS232



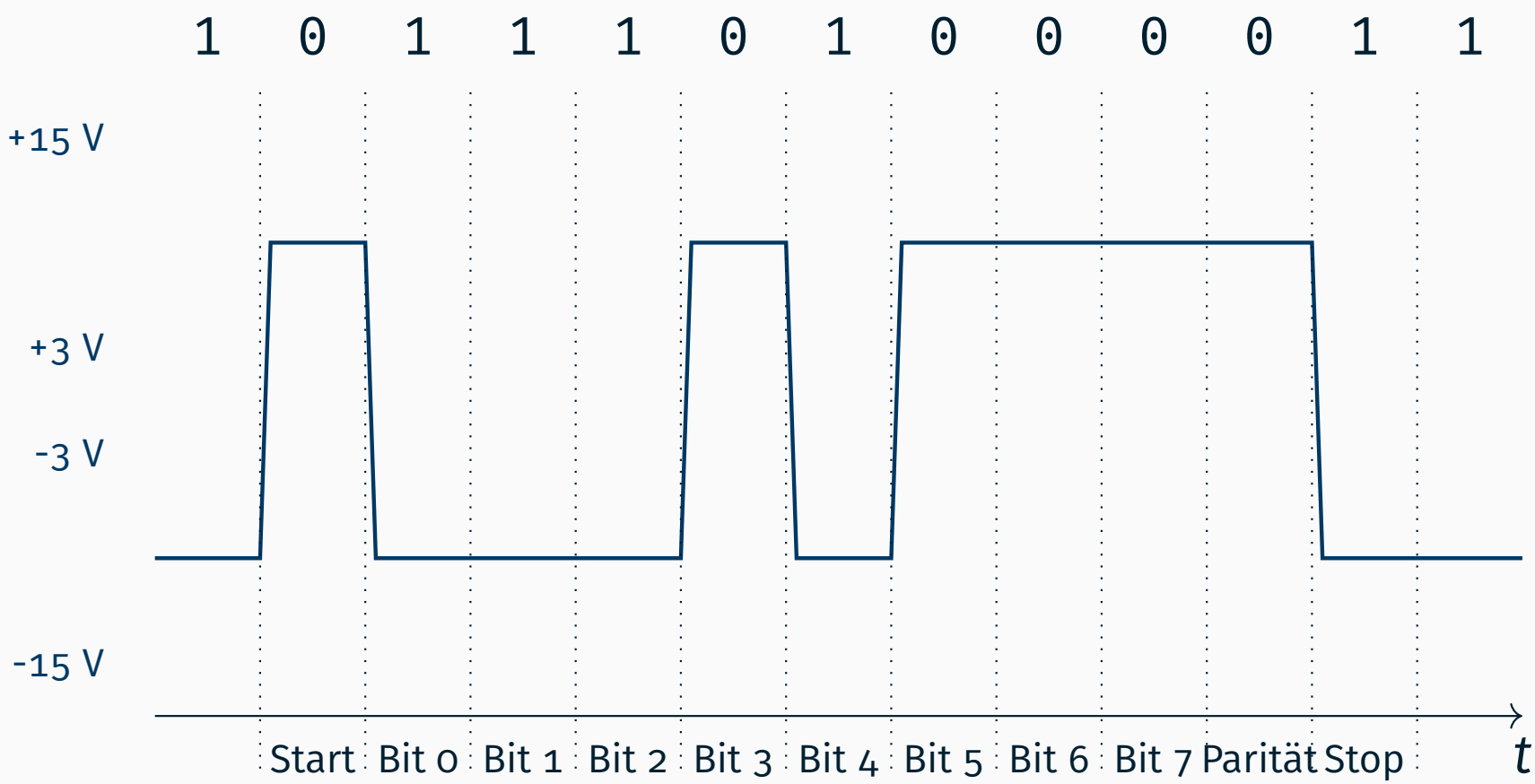
RS232



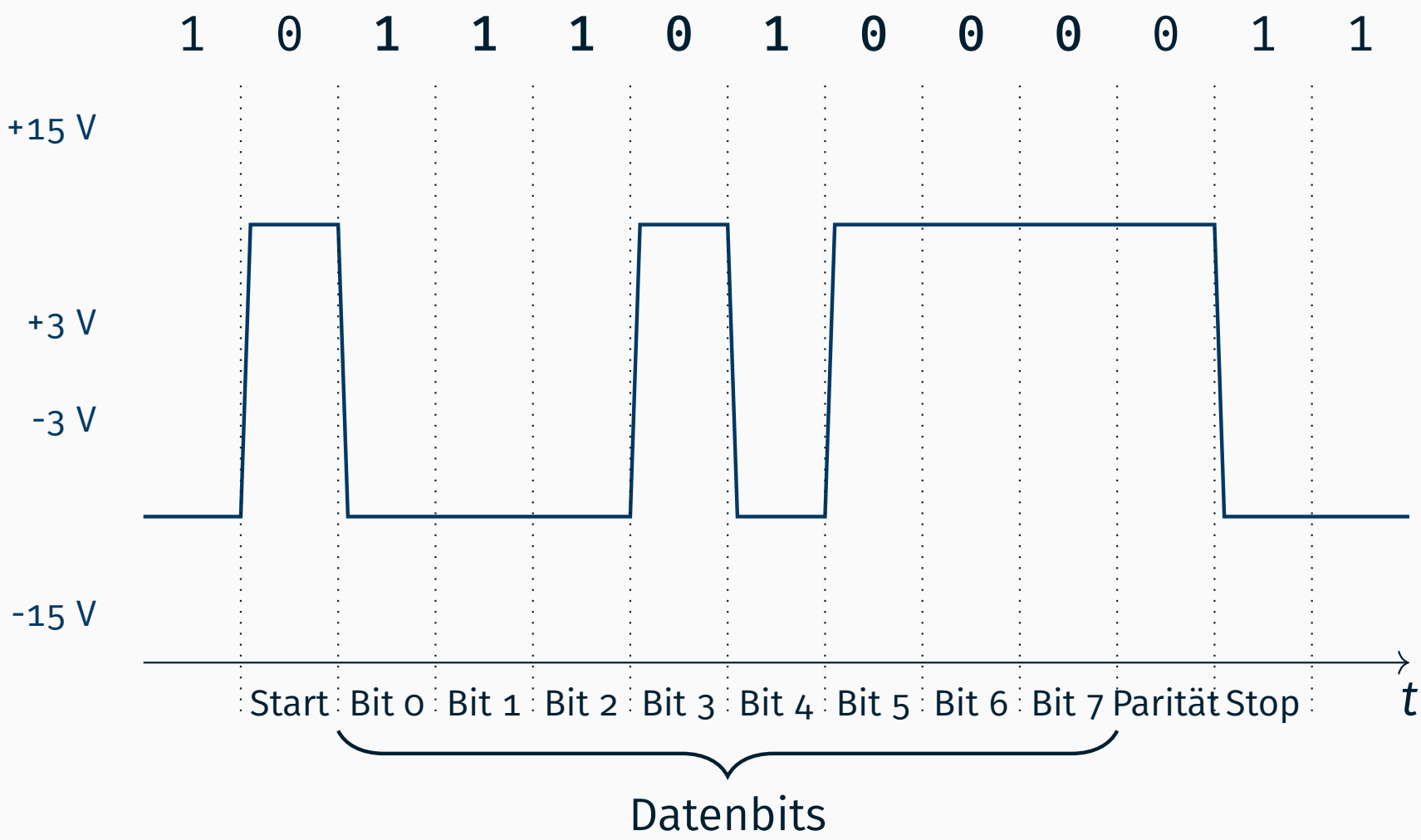
RS232



RS232


1 0 1 1 1 0 1 0 0 0 0 1 1
+15 V
+3 V
-3 V
-15 V
Start
Bit 0
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
Parität
Stop
t

RS232


1 0 1 1 1 0 1 0 0 0 0 1 1
+15 V
+3 V
-3 V
-15 V
Start
Bit 0
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
Parität
Stop
t
Datenbits

Serielle Schnittstelle

- Übertragungsrate (Baud rate) ist Teiler von 115200 Hz
- Kommunikation 8-N-1 mit 9600 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 und lowlevel.eu

Terminal

Quelle: vecchicomputer.com

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[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

```
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[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

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

Siehe auch http://www.termsys.demon.co.uk/vtansi.htm

Entwicklungsumgebung

Testumgebung

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

Testumgebung

- Virtualisiert

  QEMU Softwareemulation

  KVM Hardware-Virtualisierung
- Nackte Hardware (*bare-metal*)
  - vier (identische) Testrechner
    - Intel© Core® i5 CPU (2 Kerne @ 3.2 GHz / 4 Threads (HT))
    - 8 GB Arbeitsspeicher
    - PS/2 Tastatur + Maus
    - COM1 verbunden mit `faui04a` (`/dev/ttyBSX`)
  - Boot via Netzwerk (PXE)
  - Zugriff
    - direkt im Huber-CIP mittels KVM-Switch
    - entfernt mittels VNC – auch via Web: `https://i4stubs.cs.fau.de`
  - *privater PC (theoretisch) auch möglich*

Testumgebung

- Virtualisiert

  QEMU Softwareemulation

  KVM Hardware-Virtualisierung
- Nackte Hardware (*bare-metal*)
  - vier (identische) Testrechner
    - Intel© Core® i5 CPU (2 Kerne @ 3.2 GHz / 4 Threads (HT))
    - 8 GB Arbeitsspeicher
    - PS/2 Tastatur + Maus
    - COM1 verbunden mit `faui04a` (`/dev/ttyBSX`)
  - Boot via Netzwerk (PXE)
  - Zugriff
    - direkt im Huber-CIP mittels KVM-Switch
    - entfernt mittels VNC – auch via Web: `https://i4stubs.cs.fau.de`
  - *privater PC (theoretisch) auch möglich*

Abgabe der Aufgaben auf unseren Testrechnern

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 $\rightarrow$ Fehlersuche mit gdb.

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 $\rightarrow$ Fehlersuche mit gdb.
make netboot	für Boot am Test-Rechner ins NFS kopieren

Suffix **-noopt** um Optimierungen auszuschalten (sonst -O3)
Suffix **-verbose** aktiviert zusätzliche Ausgaben (DBG_VERBOSE)

Weitere Makefile Targets

make help	Zeige eine Beschreibung der verfügbaren Targets an
make iso	Erstelle ein bootfähiges ISO-Abbild
make usb-*dev*	bootfähigen USB-Stick auf ***dev*** (z.B. *sdb* - aber Vorsicht!) erstellen (als Superuser)

Weitere Makefile Targets

make help	Zeige eine Beschreibung der verfügbaren Targets an
make iso	Erstelle ein bootfähiges ISO-Abbild
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)

Fragen?