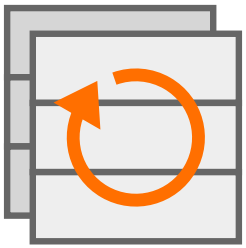


Operating System Group

TUHH

Technische Universität Hamburg

BSB – Übung 1: Ein-und Ausgabe

Yannick Loeck

2022-04-13

Einleitung

TUHH
Technische Universität Hamburg

- Wer bin ich?
  - Studium in Hannover
  - WiMi seit Nov 2021
- Wo bin ich?
  - Hörsaalübung BSB
- Was erzähle ich?
  - Relevante Konzepte für die Implementierung
- Wann erzähle ich das?
  - Zeitplan: StudIP und auf der Website (osg.tuhh.de)

Yannick Loeck

Einleitung

TUHH
Technische Universität Hamburg

- Wer bin ich?
  - Studium in Hannover
  - WiMi seit Nov 2021
- Wo bin ich?
  - Hörsaalübung BSB
- Was erzähle ich?
  - Relevante Konzepte für die Implementierung
- Wann erzähle ich das?
  - Zeitplan: StudIP und auf der Website (osg.tuhh.de)

Yannick Loeck

Probleme?

Rechnerübung (CIP)
StudIP-Forum
Mattermost: Module:BSB (im StudIP verlinkt)
Mail: yannick.loeck@tuhh.de

TUHH
Technische Universität Hamburg

BSB - HÜ1

Ihr baut selber ein Betriebssystem: StuBS

Worum geht es in der Übung heute:

- Kurze Einführung in die Entwicklung
- Aus- und Eingabe

Aufgabe 0

Aufgabe 0

TUHH
Technische Universität Hamburg

- “Optional”, keine Abgabe
- Wird aber in Aufgabe 1 sowieso gebraucht
- Noch auf Linux
- Vorlage im Gruppenrepo (`https://collaborating.tuhh.de/e-exk4/teaching/ss22/bsb/group<n>`)
- Aufgabe: Zeichenausgabe selber bauen (kein `printf`/`cout`)
- Zahlen in verschiedene Bases (binär, oktal, dezimal, hex) umrechnen

Operatorüberladung in C++

TUHH
Technische Universität Hamburg

```
std::cout << "Hello World!" << std::endl;
```

- `cout` ist ein `std::ostream`-Objekt
- Dafür ist die Überladung vom shift-left-Operator `<<` definiert

Operatorüberladung in C++


```
class Foo {
public:
  int id;
  std::string name;
};

bool operator==(const Foo& a, const Foo& b) {
  return a.id == b.id;
}
```

- Operatoren wie `+`, `==`, `<<`, `...` für eigene Klassen definieren

Operatorüberladung in C++


```
class Foo {
  int id;
  std::string name;
public:
  bool operator==(const Foo& other) {
    return this->id == other.id;
  }
};
```

- Operatoren wie `+`, `==`, `<<`, `...` für eigene Klassen definieren
- Innerhalb von Klasse: Impliziter `this`-Pointer

Aufgabe 0 - Operatorüberladung

TUHH
Technische Universität Hamburg

```
class OutputStream {
  OutputStream& operator << (char c);
}

OutputStream kout;

(kout << 'a') << 'b';
```

- Unsere Operatoren sind linksassoziativ

Aufgabe 0 - Operatorüberladung


```
class OutputStream {
  OutputStream& operator << (char c);
}

OutputStream kout;

 kout << 'a'  << 'b';
```

- Unsere Operatoren sind linksassoziativ

Aufgabe 0 - Funktionssignaturen

Technische Universität Hamburg

```c
int do_something(int x, int y) {
    return x+y;
}

// Möglichkeit 1
typedef int (*fun_t)(int x, int y);
fun_t fun = do_something;

// Möglichkeit 2
int (*fun)(int x, int y) = do_something;

fun(4,5); // 9
```

- Signatur ist der Typ der Funktion

Aufgabe 0 - Funktionssignaturen

TUHH
Technische Universität Hamburg

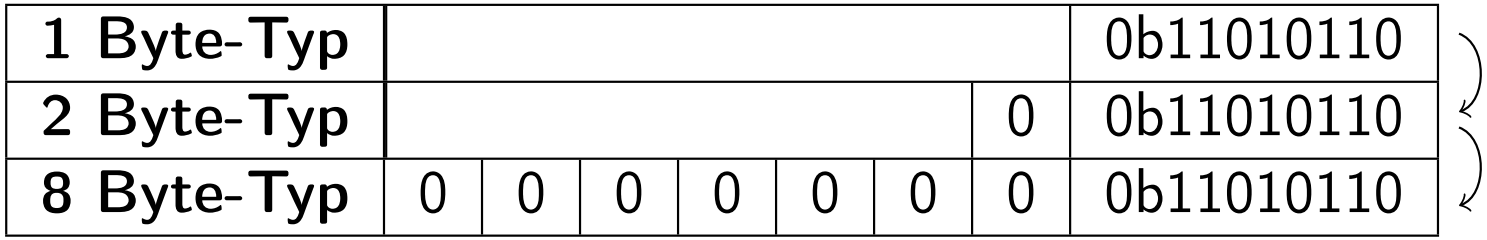
```
OutputStream& operator<<(OutputStream& (*f) (OutputStream&));
```

- Operator hat Funktionssignatur als Parameter:
  `OutputStream& (*f) (OutputStream&)`
- Die Funktion kann dann als `f(<parameter>)` aufgerufen werden

- Benutzt für Stream-Manipulatoren wie `endl`, `hex`

```
kout << "Decimal: " << 400 << " Hex: " << hex << 400 << endl;
```

Aufgabe 0 - Hinweise



1 Byte-Typ								0b11010110
2 Byte-Typ							0	0b11010110
8 Byte-Typ	0	0	0	0	0	0	0	0b11010110

- Casten zu größerem Integer-Typ geht problemlos
- Ausgabe von negativen/signed Zahlen:

```
if (num < 0)
    *this << '-' << (-1 * static_cast<unsigned>(num));
```

Aufgabe 0 - Hinweise

1 Byte-Typ								0b11010110
2 Byte-Typ							0	0b11010110
8 Byte-Typ	0	0	0	0	0	0	0	0b11010110

- Casten zu größerem Integer-Typ geht problemlos
- Ausgabe von negativen/signed Zahlen:

```
if (num < 0)
    *this << '-' << (-1 * static_cast<unsigned>(num));
```

Aufgabe 1

Aufgabe 1

TUHH
Technische Universität Hamburg

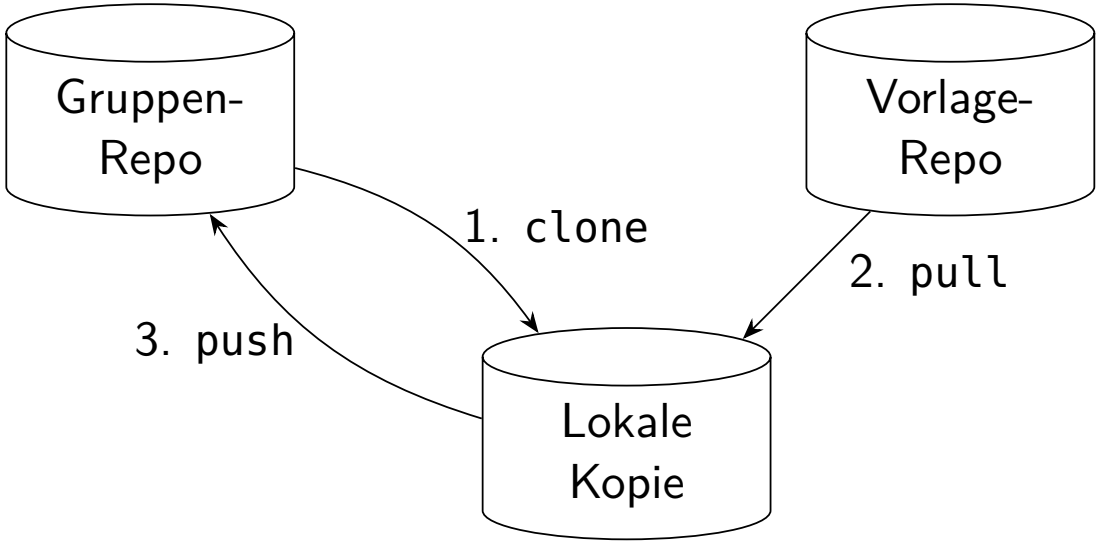
Erste Aufgabe in StuBS

Wir wollen Debugging ermöglichen!

- Ausgabe von Zeichen auf Bildschirm (CGA)
- Ausgabe über serielle Konsole

- Tastatureingabe

Repositories



1. Eigenes Gruppenrepo klonen (enthält aktuell Vorlage zu Aufgabe 0)
2. Vorlage als remote hinzufügen:
   `git remote add vorlage <url>.git`
3. Für jede neue Aufgabe:
   - Vorlage pullen
   - Ins Gruppenrepo pushen

Tip

Kleinere commits (z.B. `git add -i`) $\Rightarrow$ Fehlersuche leichter

StuBS - Erste Schritte

TUHH
Technische Universität Hamburg

- Bauen mit `make all`
  `make all-noopt` für keine Optimierung
- Starten (grafisch) mit `make qemu`
  `make qemu-curses` im Terminal
- Debuggen mit `make qemu-gdb`
  startet Emulator, wartet auf gdb
  `make connect-gdb` startet gdb und verbindet
- Maus im QEMU-Fenster "gefangen"? `Ctrl-Alt-G`
- Beenden mit `(Ctrl-)Alt-2`, `quit` ⏎

Ausgabe - CGA Screen

CGA Screen

- **C**olor **G**raphics **A**dapter (1981), erste Grafikkarte von IBM für den PC
- Grafikmodus: Pixel selbst ansteuern

CGA Screen

TUHH
Technische Universität Hamburg

- **C**olor **G**raphics **A**dapter (1981), erste Grafikkarte von IBM für den PC
- Grafikmodus: Pixel selbst ansteuern
- Textmodus: Zeichen auf 80x25 Matrix ausgeben ⟸

CGA Screen

TUHH
Technische Universität Hamburg

- **C**olor **G**raphics **A**dapter (1981), erste Grafikkarte von IBM für den PC
- Grafikmodus: Pixel selbst ansteuern
- Textmodus: Zeichen auf 80x25 Matrix ausgeben ⟸
- 2000 Zeichen: 2000 Byte + 2000 Byte extra für Attribute

CGA Screen

TUHH
Technische Universität Hamburg

- **C**olor **G**raphics **A**dapter (1981), erste Grafikkarte von IBM für den PC
- Grafikmodus: Pixel selbst ansteuern
- Textmodus: Zeichen auf 80x25 Matrix ausgeben ⟸
- 2000 Zeichen: 2000 Byte + 2000 Byte extra für Attribute
- Darstellungsattribut: 8 bit

7	6	5	4	3	2	1	0
Blinken	Hintergrundfarbe			Vordergrundfarbe			

CGA Screen

- **C**olor **G**raphics **A**dapter (1981), erste Grafikkarte von IBM für den PC
- Grafikmodus: Pixel selbst ansteuern
- Textmodus: Zeichen auf 80x25 Matrix ausgeben ⟸
- 2000 Zeichen: 2000 Byte + 2000 Byte extra für Attribute
- Darstellungsattribut: 8 bit
- ASCII Zeichen: `man ascii`

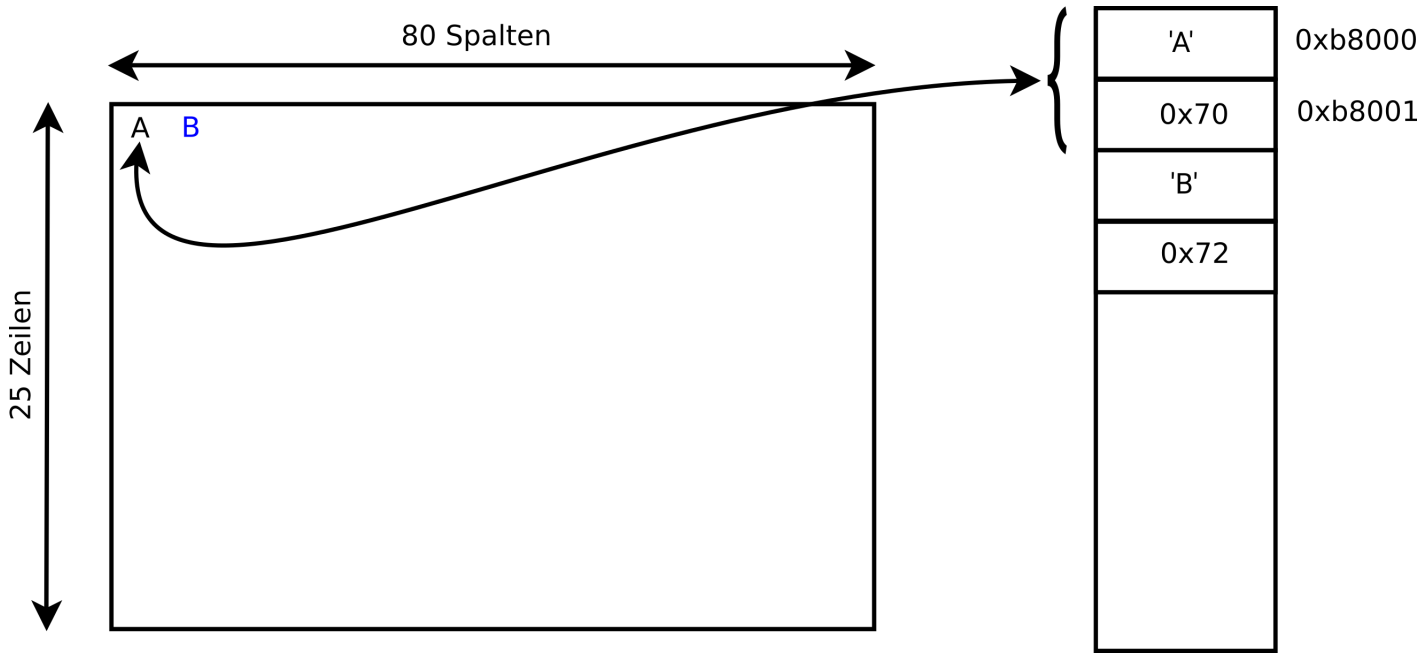
7	6	5	4	3	2	1	0
Blinken	Hintergrundfarbe			Vordergrundfarbe			

CGA Screen

- **C**olor **G**raphics **A**dapter (1981), erste Grafikkarte von IBM für den PC
- Grafikmodus: Pixel selbst ansteuern
- Textmodus: Zeichen auf 80x25 Matrix ausgeben ⟸
- 2000 Zeichen: 2000 Byte + 2000 Byte extra für Attribute
- Darstellungsattribut: 8 bit
- ASCII Zeichen: `man ascii`

7	6	5	4	3	2	1	0
Blinken	Hintergrundfarbe			Vordergrundfarbe			

CGA Screen - Speicher



- Grafikkartenspeicher an fester virtueller Speicheradresse `0xb8000`
- In Hauptspeicher eingeblendet: *Memory-Mapped*
- Zeichen und Attribut im Speicher abwechselnd
- Mapping von 2D zu Speicher: zeilenweise

CGA Screen - Benutzung


```
char* CGA = (char*)0xb8000;
CGA[0] = 'A';
CGA[1] = 0x70;
```

- Aufwändig z.B. für Zeichen mitten drin

CGA Screen - Benutzung

TUHH
Technische Universität Hamburg

```
struct CGA_Cell {
    uint8_t character;
    uint8_t attr;
};

struct CGA_Cell* CGA = (struct CGA_Cell*) 0xb8000;
CGA[0] = {'A', 0x70};

CGA[34].attr = 0x90;
```

- Diese Methode ist praktischer

CGA Screen - Attribute

Einschub: Logische vs Bitoperationen

TUHH
Technische Universität Hamburg

- logical
  - not, and, or

```
if (!cond_1)

if (cond_1 && cond_2)

if (cond_3 || cond_4)
```

- bitwise
  - not, and, or, xor
  - shift left/right
  - syntactic sugar: `a &= 0xff;`

```
~0b1101 = 0b0010

a & b     a | b     a ^ b

a << 12 (= a * 4096)
```

```
16 >> 3 = 2
0xdeadbeef >> (4 * 4) = 0xdead
0xdeadbeef >> (4 * 4) & 0xff = 0xad
```

Einschub: Logische vs Bitoperationen

TUHH
Technische Universität Hamburg

- logical
  - not, and, or

```
if (!cond_1)
if (cond_1 && cond_2)
if (cond_3 || cond_4)
```

- bitwise
  - not, and, or, xor
  - shift left/right
  - syntactic sugar: `a &= 0xff;`

```
~0b1101 = 0b0010
a & b     a | b     a ^ b
a << 12 (= a * 4096)
```

```
16 >> 3 = 2
0xdeadbeef >> (4 * 4) = 0xdead
0xdeadbeef >> (4 * 4) & 0xff = 0xad
```
[1]

[1] https://en.cppreference.com/w/c/language/operator_precedence

Bitoperationen - CGA-Attribute

Technische Universität Hamburg

7	6	5	4	3	2	1	0
Blinken	Hintergrundfarbe			Vordergrundfarbe			

```
char make_attribute(char foreground,
                    char background, char blink) {
  foreground &= 0xf; // ffff ffff -> 0000 ffff
  background &= 0x7; // bbbb bbbb -> 0000 0bbb
  blink      &=   1; // BBBB BBBB -> 0000 000B

  background <<=  4; // 0000 0bbb -> 0bbb 0000
  blink      <<=  7; // 0000 000B -> B000 0000

  return foreground | background | blink; // Bbbb ffff
}
```

Bitoperationen - CGA-Attribute

7	6	5	4	3	2	1	0
Blinken	Hintergrundfarbe			Vordergrundfarbe			

```
struct CGA_Attr {
  char foreground : 4;
  char background : 3;
  char blink      : 1;

  CGA_Attr(char fg, char bg, char B)
    : foreground(fg), background(bg), blink(B) {}
} __attribute__((packed));

CGA_Attr attr(7, 0, 0); // Light Grey on Black, No blink
```

- LSB-first
- packed: Kein padding für memory alignment

CGA Screen - Cursor

CGA Screen - Cursor

TUHH
Technische Universität Hamburg

- Für Textausgabe: *Wo sind wir auf dem Bildschirm?*
- Cursor für aktuelle Position, 2 Möglichkeiten
- Software:
  - Manuelles Setzen von Blink-Attribut in `row, col`
  - **funktioniert im QEMU nicht**

CGA Screen - Cursor

TUHH
Technische Universität Hamburg

- Für Textausgabe: *Wo sind wir auf dem Bildschirm?*
- Cursor für aktuelle Position, 2 Möglichkeiten
- Software:
  - Manuelles Setzen von Blink-Attribut in `row, col`
  - **funktioniert im QEMU nicht**
- Hardware:
  - 18 Steuerregister (mit 8 Bit Breite)
  - Cursor-Steuerung über Register 14 und 15
    - Cell Number von 0-1999
    - 14: obere 8 bit
    - 15: untere 8 bit

CGA Screen - Cursor

TUHH
Technische Universität Hamburg

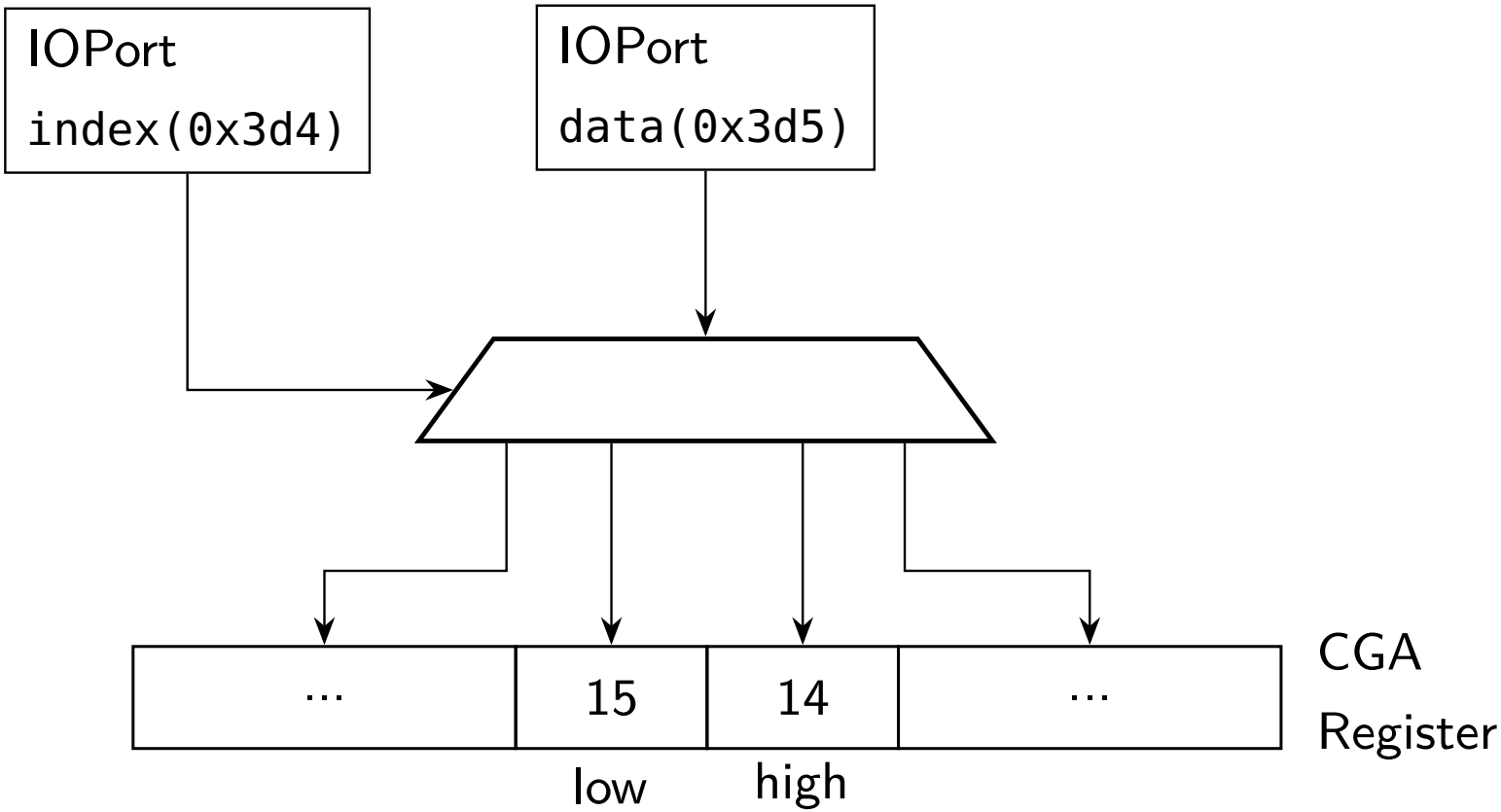
- Für Textausgabe: *Wo sind wir auf dem Bildschirm?*
- Cursor für aktuelle Position, 2 Möglichkeiten
- Software:
  - Manuelles Setzen von Blink-Attribut in `row, col`
  - **funktioniert im QEMU nicht**
- Hardware:
  - 18 Steuerregister (mit 8 Bit Breite)
  - Cursor-Steuerung über Register 14 und 15
    - Cell Number von 0-1999
    - 14: obere 8 bit
    - 15: untere 8 bit

I/O Ports

TUHH
Technische Universität Hamburg

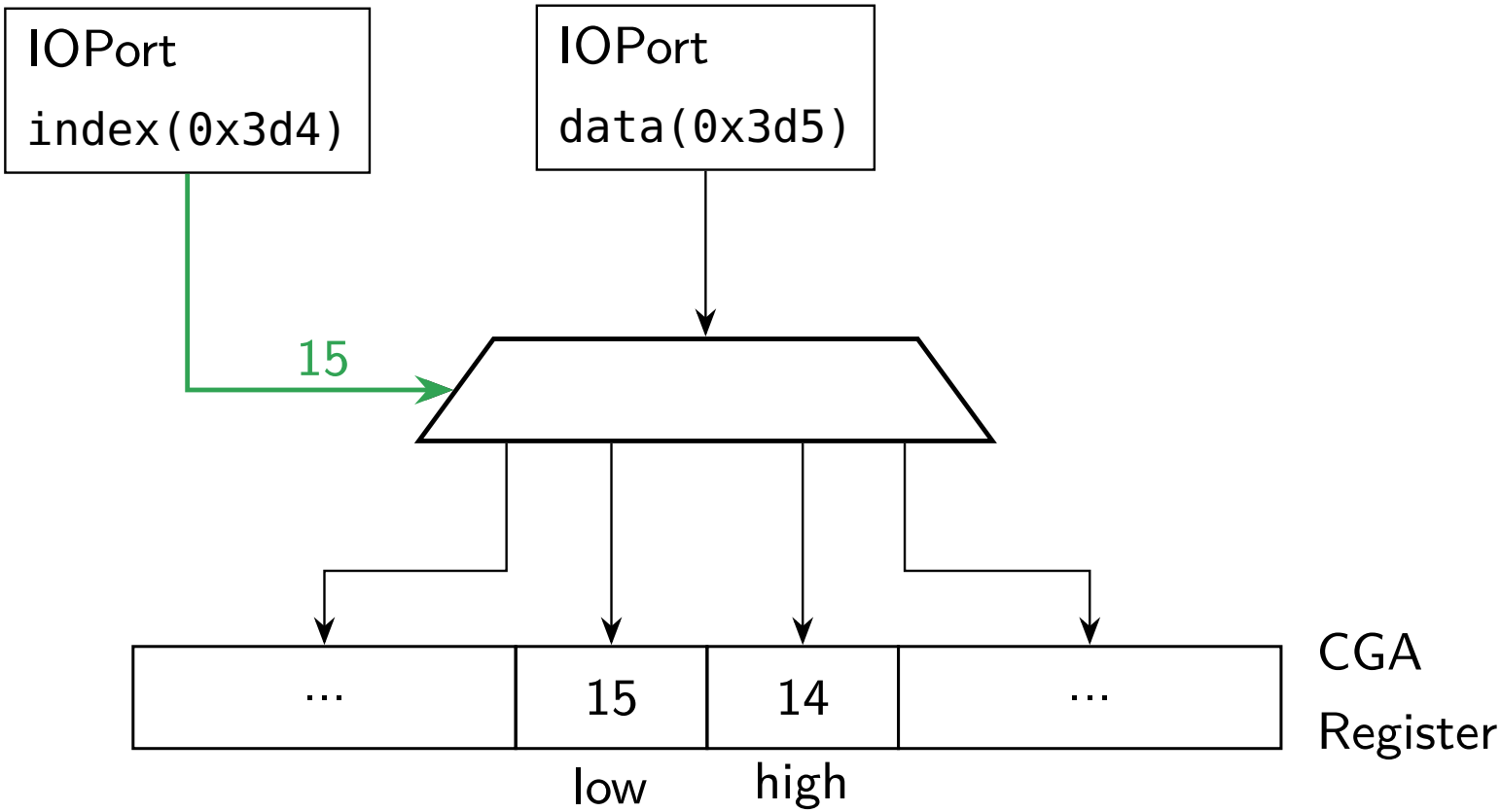
- x86 CPU: mehrere Busse mit eigenem Adressraum
- 2^{32} (oder 2^{64}) für RAM
  - Zugriff z.B. über `mov ..., (%eax)`
- 2^{16} für I/O Ports von Geräten
  - Zugriff über `inb 0x60` / `outb %ax, 0x64`
- Memory-Mapped I/O vs. Port-based I/O
- In StuBS: `machine/ioport.h`

I/O Ports für Hardware-Cursor



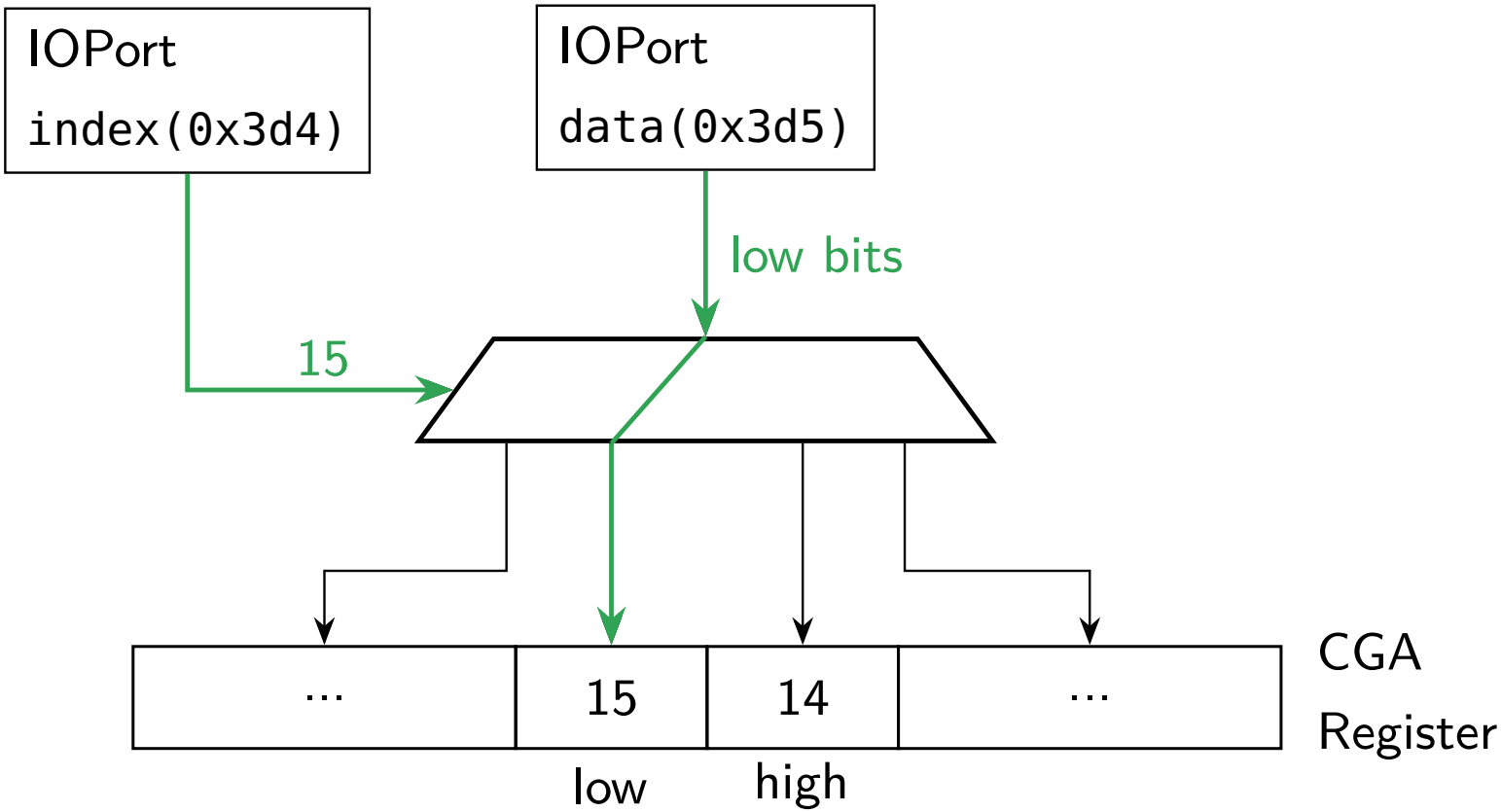
- Erst `index` setzen, dann `data` schreiben
- Empfehlung: Funktion wie `writeCGAReg(int reg, int data)`

I/O Ports für Hardware-Cursor



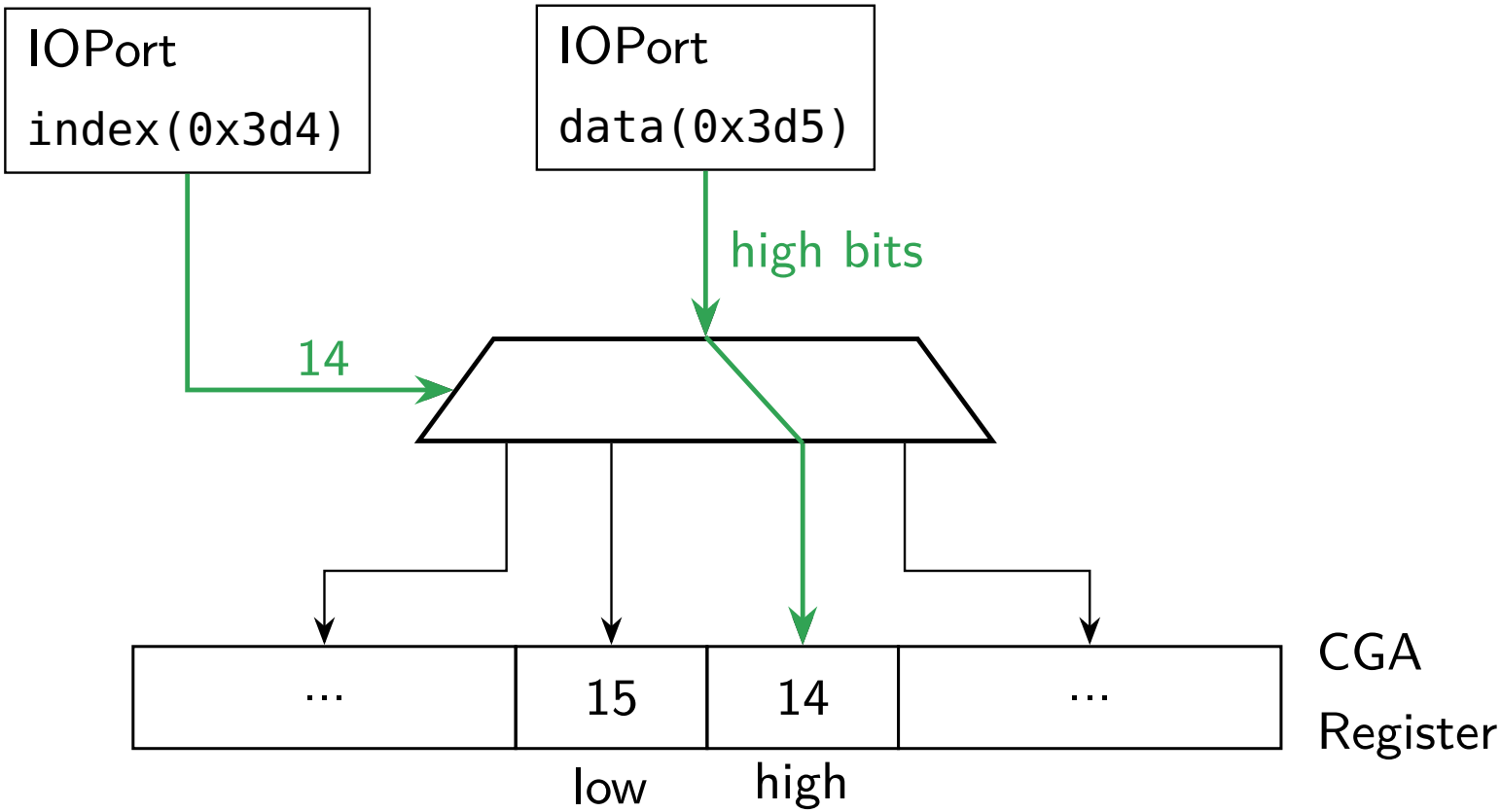
- Erst `index` setzen, dann `data` schreiben
- Empfehlung: Funktion wie `writeCGAReg(int reg, int data)`

I/O Ports für Hardware-Cursor



- Erst `index` setzen, dann `data` schreiben
- Empfehlung: Funktion wie `writeCGAReg(int reg, int data)`

I/O Ports für Hardware-Cursor



- Erst `index` setzen, dann `data` schreiben
- Empfehlung: Funktion wie `writeCGAReg(int reg, int data)`

Eingabe - Tastatur

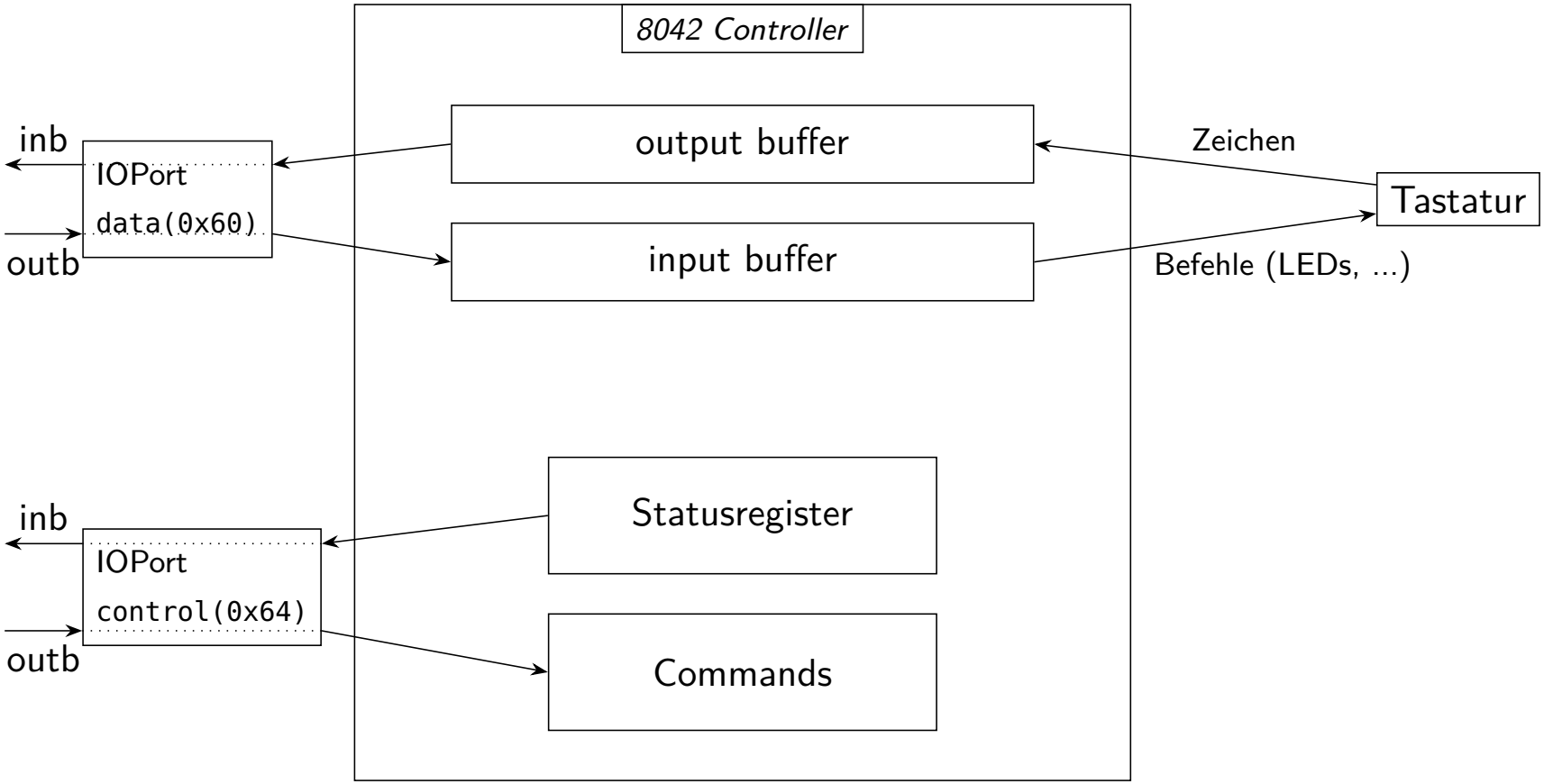
Tastatureingabe

TUHH
Technische Universität Hamburg

- Wir wollen auch Eingabe können
- Zeichen als ASCII kodiert [2]
- Von der Tastatur kommen aber Scan Codes
- Woher weiß die Tastatur welche Taste gedrückt ist?
  - Matrix: Reihen und Spalten von Tasten verbunden
  - Dioden dazwischen

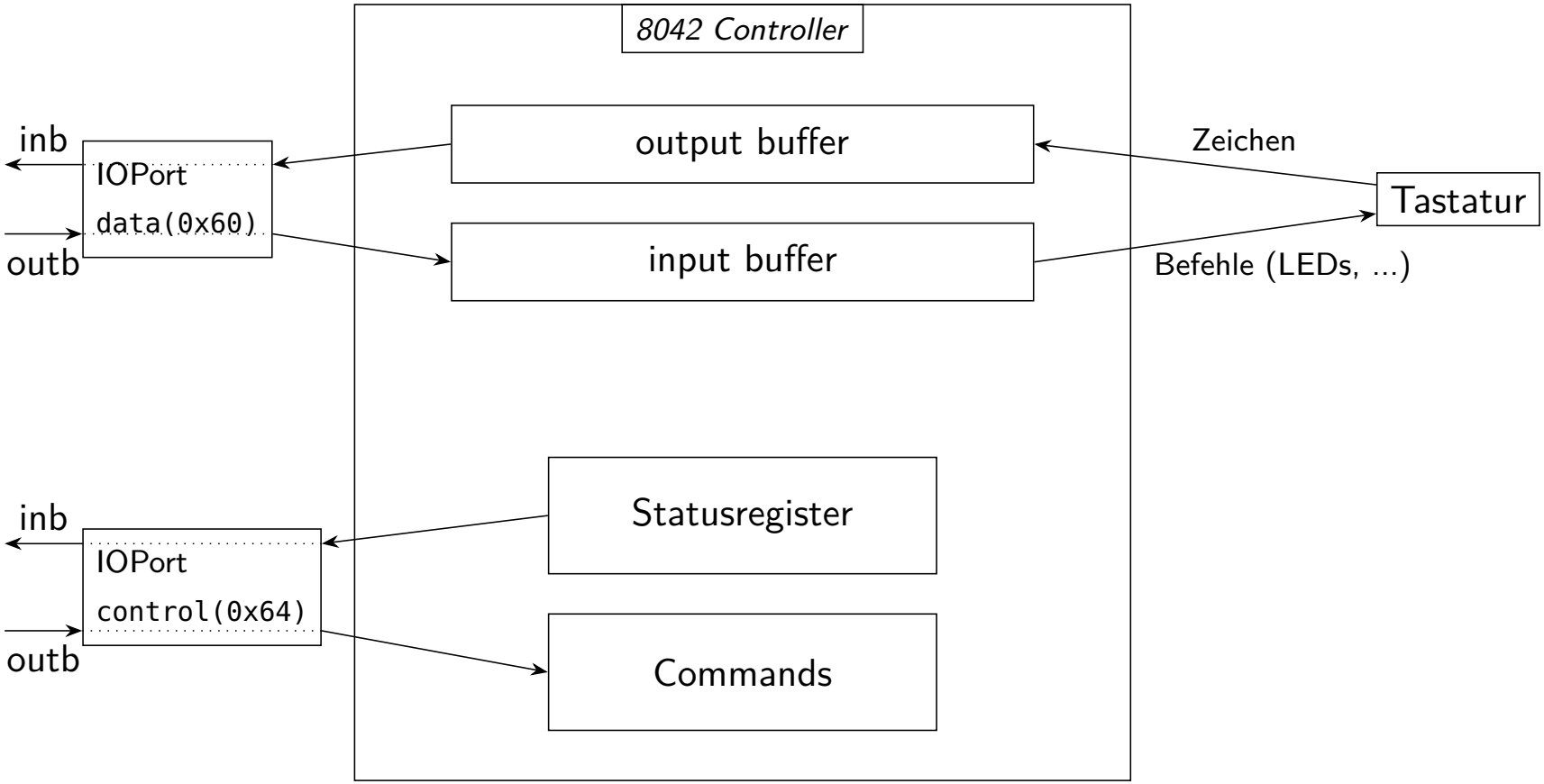
[2] American Standard Code for Information Interchange

Tastaturcontroller



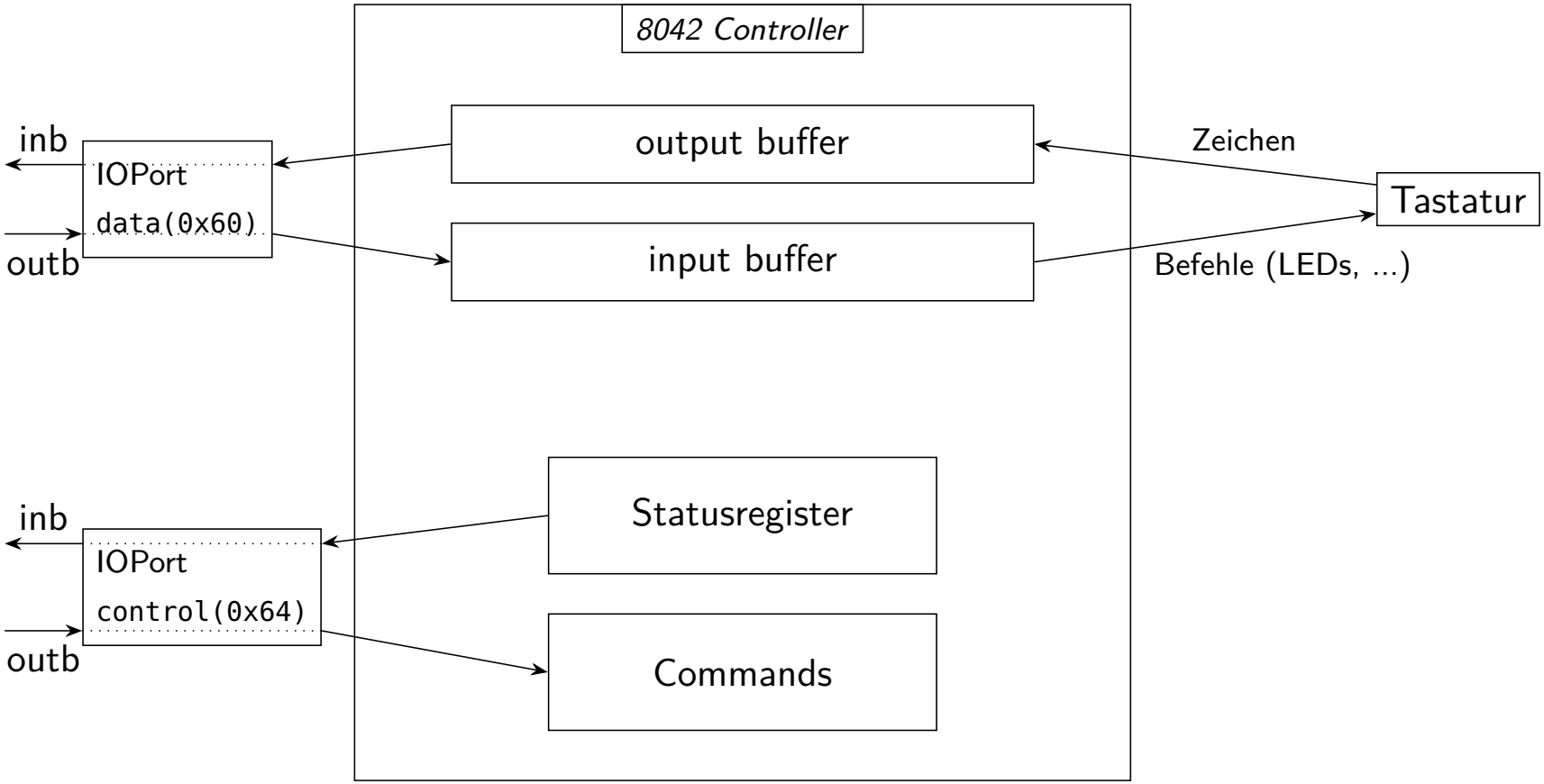
- Intel 8042 Chip
- PS/2 Controller, *alt*
- Hat wieder I/O Ports

Tastaturcontroller



- Data Port `0x60` (RW):
  - Lesen von Daten (Zeichen)
  - Schreiben von Daten an ein PS/2 Gerät
    z.B. `0xed` (set_led) oder `0xf3` (set_speed)

Tastaturcontroller



- Control Port 0x64 (RW):
  - Status Register (R): Flags
  - Command Register (W): Kommandos an den PS/2 Controller

Tastaturcontroller - Statusregister

TUHH
Technische Universität Hamburg

Bit	Bedeutung
0	Zeichen im Output Buffer? `outb`
1	Input Buffer voll? `inpb`
2	System Flag
3	PS/2 Gerät (0) oder Controller (1) als Ziel vom Input Buffer
4	*Chipsatz-Spezifisch*
5	Wert im Output Buffer von der Maus? `auxb`
6	Time-Out Fehler
7	Paritätsfehler

- Relevant: `outb`, `inpb`, `auxb`

Hinweise

TUHH
Technische Universität Hamburg

- Maus-Support nicht nötig, aber filtern von Maus-Events über `auxb`
- Lesen aus Data Port **entfernt** vorderstes Zeichen aus Output Buffer
- Hier aktives Warten (Polling): bis Zeichen im Output Buffer (`outb`)
- Für Set-LED und Set-Speed in Input Buffer schreiben (über data-Port)
- Auf Set-LED antwortet Tastatur mit `ACK` in Output Buffer
  Kann verloren gehen $\Rightarrow$ `ACK` einfach ignorieren

Ausgabe - Serielle Schnittstelle

Serielle Schnittstelle

TUHH
Technische Universität Hamburg

- Mehr Zeichen ausgeben als über Grafikkarte
- Historisch, Bildschirmterminals
- An COM-Ports angeschlossen
- VT100-Protokoll (1978)
- ANSI Escape-Sequenzen, fangen an mit ^[
- Benutzung:
  1. Konfigurieren
  2. Zeichen ausgeben

Serielle Schnittstelle - Konfiguration

TUHH
Technische Universität Hamburg

- Baudrate einstellen ($\approx$ bits pro Sekunde)
  - $\neq$ Datenbits pro Sekunde: Start und Stop bits
  - gewöhliche Rate: 115200 Baud
- Format einstellen: 8N1
  - 8 bits per Byte (oder 7)
  - 1 stop bit (oder 2)
  - no parity bit (oder odd oder even)

Serielle Schnittstelle - Konfiguration

TUHH
Technische Universität Hamburg

- Baudrate einstellen ($\approx$ bits pro Sekunde)
  - $\neq$ Datenbits pro Sekunde: Start und Stop bits
  - gewöhliche Rate: 115200 Baud
- Format einstellen: 8N1
  - 8 bits per Byte (oder 7)
  - 1 stop bit (oder 2)
  - no parity bit (oder odd oder even)

StuBS - Serial

TUHH
Technische Universität Hamburg

- Zu schreibendes Register: IOPort (COM1 + offset)
- Konfiguration
  - Baudrate als Teiler von 115200 (1 für maximum)
  - Oberstes bit des Line Control Registers setzen (DLAB)
  - Offset 0: Divisor LSB, 1: MSB
- Line Status Register
  - Bit 0: Daten zum empfangen sind da (Data Ready)
  - Bit 6: Bereit Daten zu senden (Transmitter empty)
- Erst mal mit Polling und `bool blocking = 1`

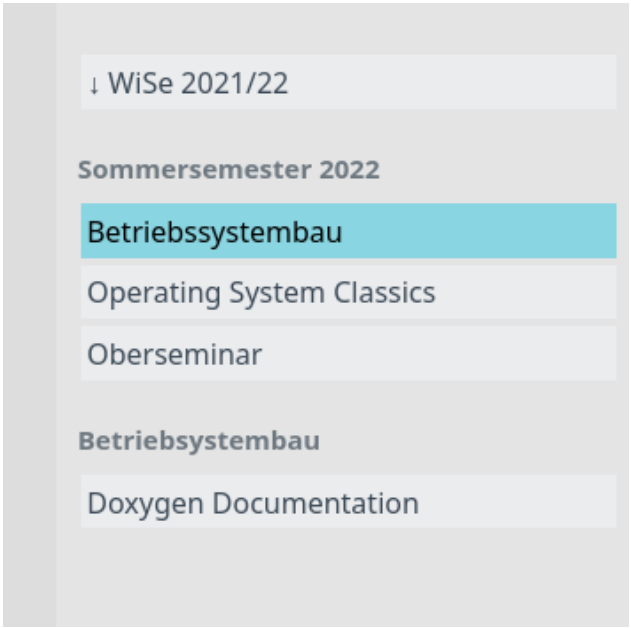
StuBS - SerialStream

TUHH
Technische Universität Hamburg

- Aufbau einer Escape-Sequenz
  - Escape-Zeichen: `\x1b`, `\033`, in gcc/clang auch `\e`
  - Mit `[` wird es zum Control Sequence Inducer
  - Steuerzeichen für Befehl (z.B. `2K` um Zeile zu löschen)
- Cursor-Position
  - Anfragen mit: `\e[6n`
  - Antwort im Format: `\e[6<ROW>;<COLUMN>R`
  - **Achtung:** Zeile und Spalte fangen bei 1 an
- Kompatibilität: Kein `\n` ohne `\r` ausgeben
- Farben/etc.: siehe `doc/serial.html`

Links

- Veranstaltung auf der Website:
  `https://osg.tuhh.de/Lehre/SS22/V_BSB/`
- Dokumentation:
  `https://osg.tuhh.de/Lehre/SS22/V_BSB/doc/`
- Mattermost:
  `https://communicating.tuhh.de/eim/channels/module-bsb`



Zeitplan

TUHH
Technische Universität Hamburg

KW	Dienstag	Di 16:15	Mi 12:15	Mi 13:45	Späteste Abgabe
KW14	05. April	VL1 Einführung	VL2 BS-Entwicklung		
KW15	12. April	VL3 IRQs (Hardware)	Ü1 Ein-/Ausgabe	RÜ	
KW16	19. April	Ü0 C++	RÜ	RÜ	
KW17	26. April	VL4 IRQs (Software)	Ü2 IRQ-Behandlung	RÜ	A1 Abgabe 1
KW18	03. Mai	VL5 Intel IA-32	RÜ	RÜ	
KW19	10. Mai	VL6 IRQs (Synchronisation)	RÜ	RÜ	
KW20	17. Mai	--	Ü3 IRQ-Synchronisation	RÜ	A2 Abgabe 2
KW21	24. Mai	Ferien	RÜ	RÜ	
KW22	31. Mai	VL7 Koroutinen und Fäden	Ü4 Fadenumschaltung	RÜ	A3 Abgabe 3
KW23	07. Juni	--	RÜ	RÜ	
KW24	14. Juni	VL8 Scheduling	Ü5 Zeitscheiben-Scheduling	RÜ	A4 Abgabe 4
KW25	21. Juni	VL9 Architekturen	RÜ	RÜ	
KW26	28. Juni	VL10 Fadensynchronisation	Ü6 Fadensynchronistion	RÜ	A5 Abgabe 5
KW27	05. Juli	VL11 Gerätetreiber	RÜ	RÜ	
KW28	12. Juli	VL12 Ausblick	Ü7 Anwendung (opt)	RÜ	A6 Abgabe 6

- Beachtet die Deadlines!