Universität Bamberg

GRABS: Grundlagen der Rechnerarchitektur und Betriebssysteme

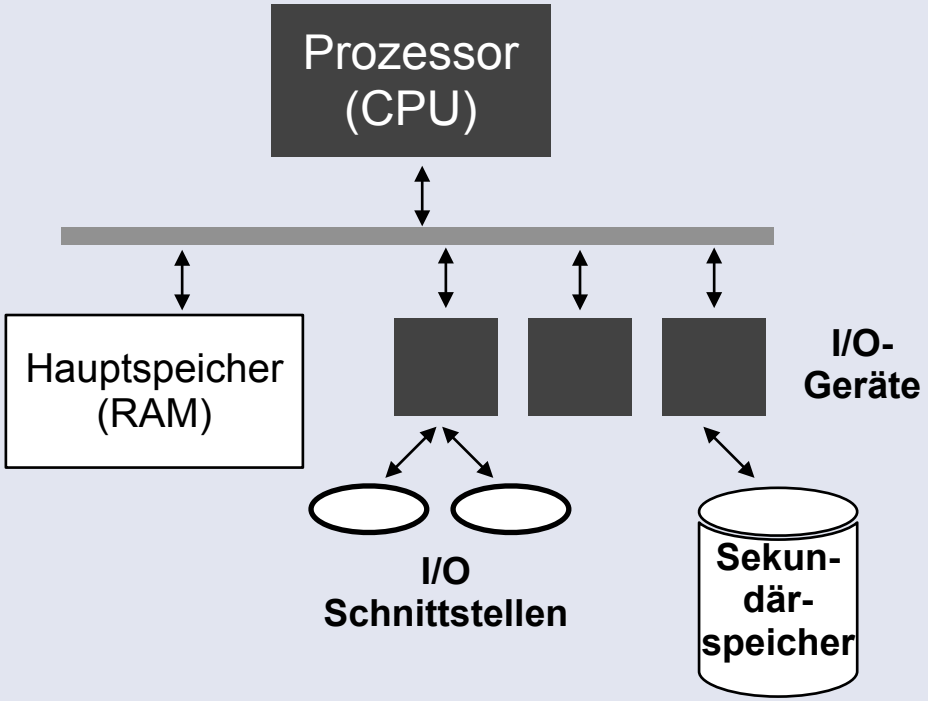
Vorlesung 8: Ein- und Ausgabe

Literatur:
Arpaci-Dusseau [3]
Kap. 36

Michael Engel (michael.engel@uni-bamberg.de)
Lehrstuhl für Praktische Informatik, insbes. Systemnahe Programmierung
https://www.uni-bamberg.de/sysnap

Ein- und Ausgabe (Input/Output, I/O)

- Bisher haben wir betrachtet:
  - Prozessor (CPU)
  - Speicher
- **Jetzt: Ein-Ausgabegeräte** *(I/O devices)*



Schnittstellen für I/O-Geräte

Universität Bamberg

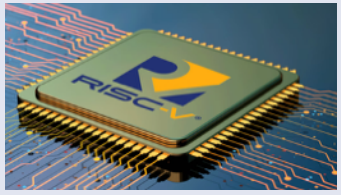
Tastatur-controller

Video-controller

Festplatten-controller

Ethernet-controller

Bus



Hauptspeicher (RAM)

Ein-/Ausgabegeräte sind über einen **Controller** mit dem **Systembus** verbunden
Sie werden über **I/O-Register** in den Controllern programmiert

Beispiel: Tastatur

- Serielle Kommunikation, Zeichenorientiert
  - Tastaturen sind "intelligent" (haben einen eigenen Prozessor)

Steuercodes
z.B. für LEDs

Make und ***Break-Codes***
zeigen gedrückte/
losgelassene Tasten an

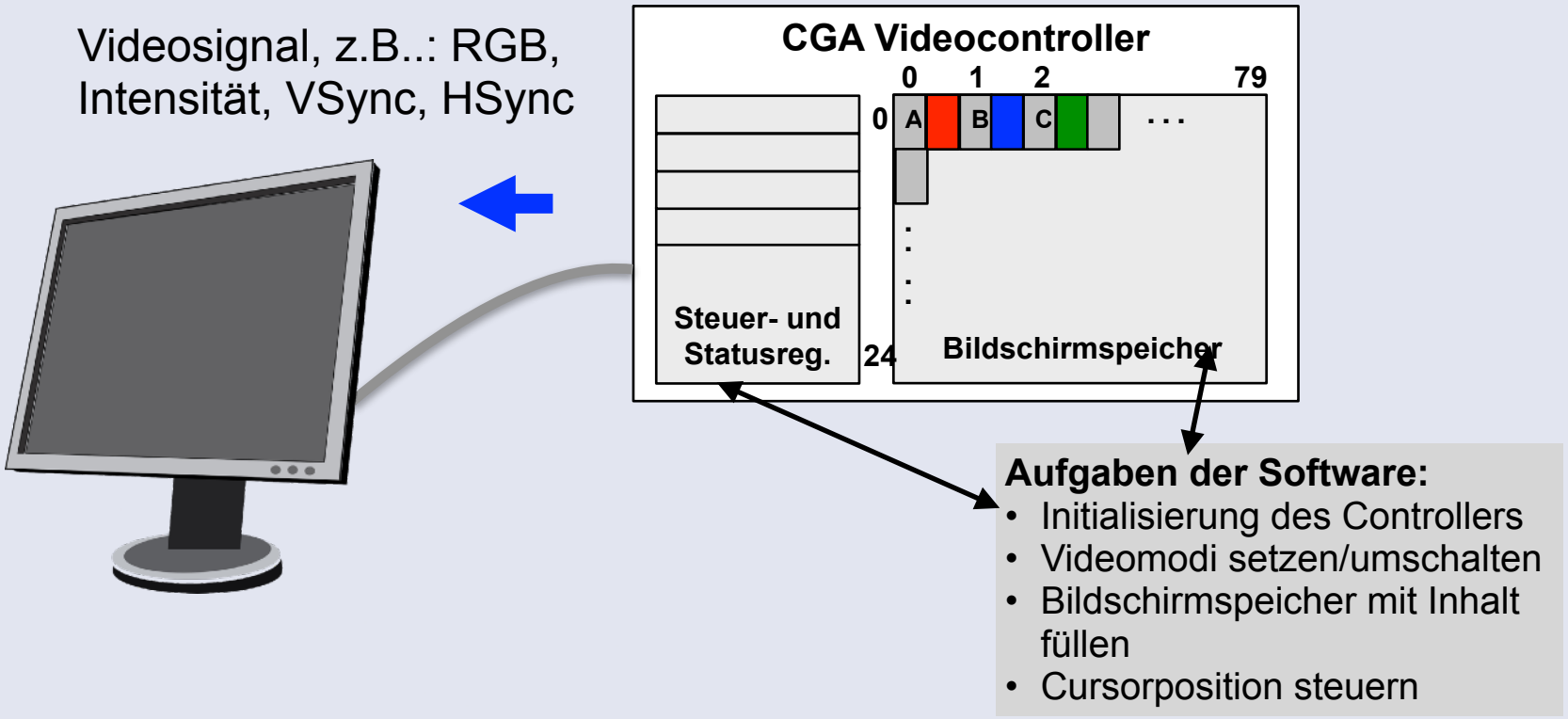
Tastatur-controller

Aufgaben der Software:
- Initialisierung des Controllers
- Zeichen von der Tastatur abholen
- Umsetzen von *make* und *break*-Codes nach ASCII
- Befehle senden (z.B. zum Steuern der LEDs)

Der Controller erzeugt einen ***Interrupt***, sobald ein Zeichen verfügbar ist

Beispiel: Grafikkarte

- Kommunikation über Videosignal
  - analog: VGA, digital: DVI, HDMI, DisplayPort
- Transformiert die Inhalte des ***Bildschirmspeichers*** *(frame buffer)* in ein Bild (z.B. 80x25 Zeichenmatrix oder *bitmap*)



Beispiel: IDE-Festplatte

- Kommunikation über AT-Befehle
  - **Blockweiser wahlfreier Zugriff** auf Datenblöcke

AT-Befehle
- Kalibriere Festplatte
- Lese/schreibe/vergleiche Block
- Formatiere Spur
- Positioniere Schreib-/Lesekopf
- Diagnose
- Plattenparameter konfigurieren

IDE-Festplattencontroller

Steuer- and Statusreg.

Sektor-puffer

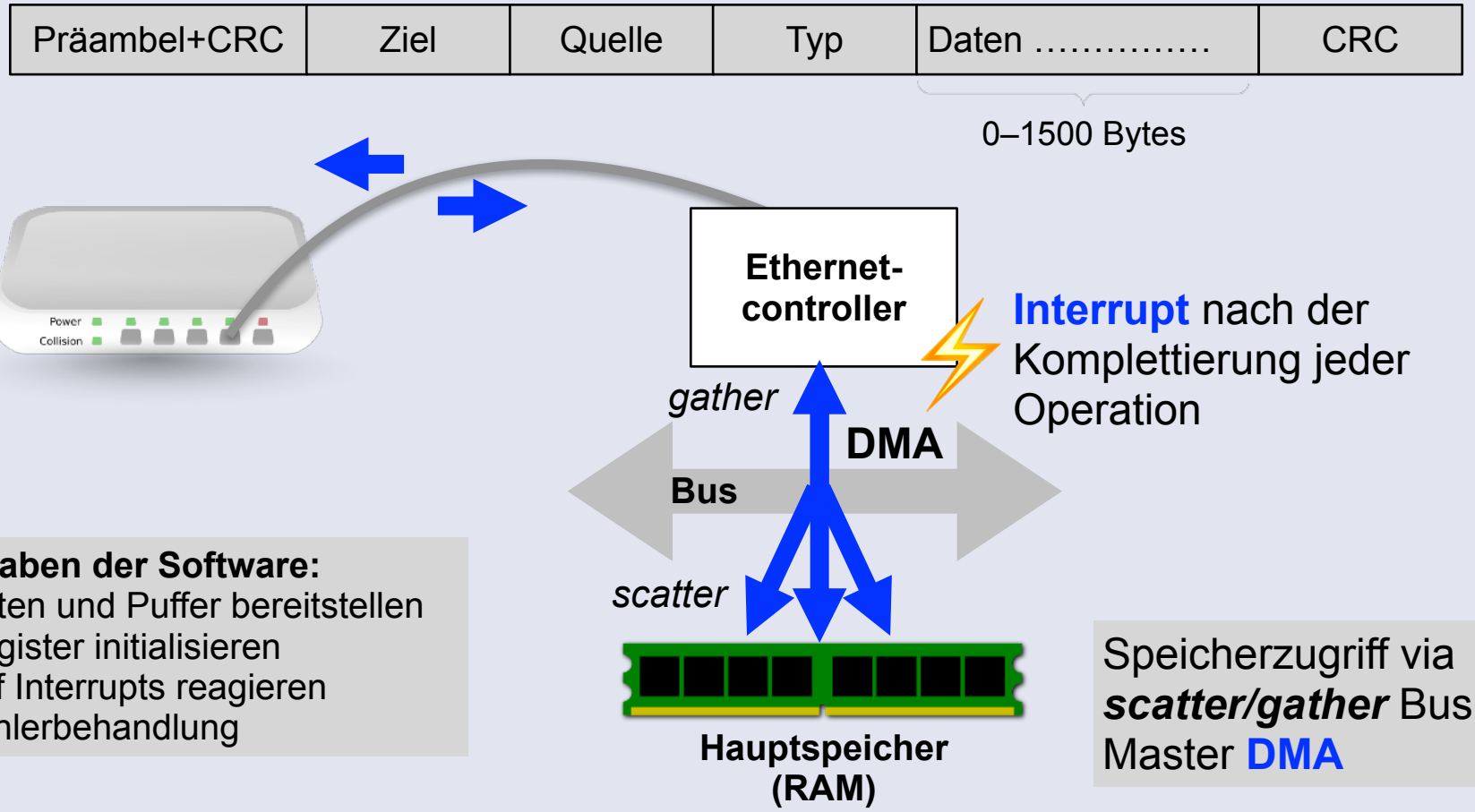
Datenblöcke
(je 512 Bytes)

Aufgaben der Software:
- Schreibe AT-Befehle in Register
- Sektorpuffer verwalten
- Auf Interrupts reagieren
- Fehlerbehandlung

Der Festplattencontroller erzeugt einen Interrupt, sobald der Sektorpuffer gelesen oder geschrieben wurde

Beispiel: Ethernet

- Serielle paketbasierte Buskommunikation
  - Pakete haben variable Größen und enthalten Adressen:



Aufgaben der Software:
- Daten und Puffer bereitstellen
- Register initialisieren
- Auf Interrupts reagieren
- Fehlerbehandlung

Speicherzugriff via ***scatter/gather*** Bus Master **DMA**

Ein- und Ausgabe und das Betriebssystem: Geräteklassen

- Zeichenorientierte Geräte *(character devices)*
  - Tastatur, Drucker, Modem, Maus, ...
  - Meist nur sequentieller Zugriff, selten wahlfrei
- Blockorientierte Geräte *(block devices)*
  - Festplatte, SSD, CD-ROM, DVD, Bandlaufwerke, ...
  - Üblicherweise blockweiser wahlfreier Zugriff
- Andere Geräte passen nicht in dieses Schema, z.B.
  - (GP)GPUs (besonders mit 3D-Beschleunigung)
  - Netzwerkkarten (Protokolle, Adressierung, broadcast/multicast, Paketfilterung, ...)
- Timer (sporadische oder periodische Interrupts)
- ...

Interrupts (Unterbrechungen, IRQs)

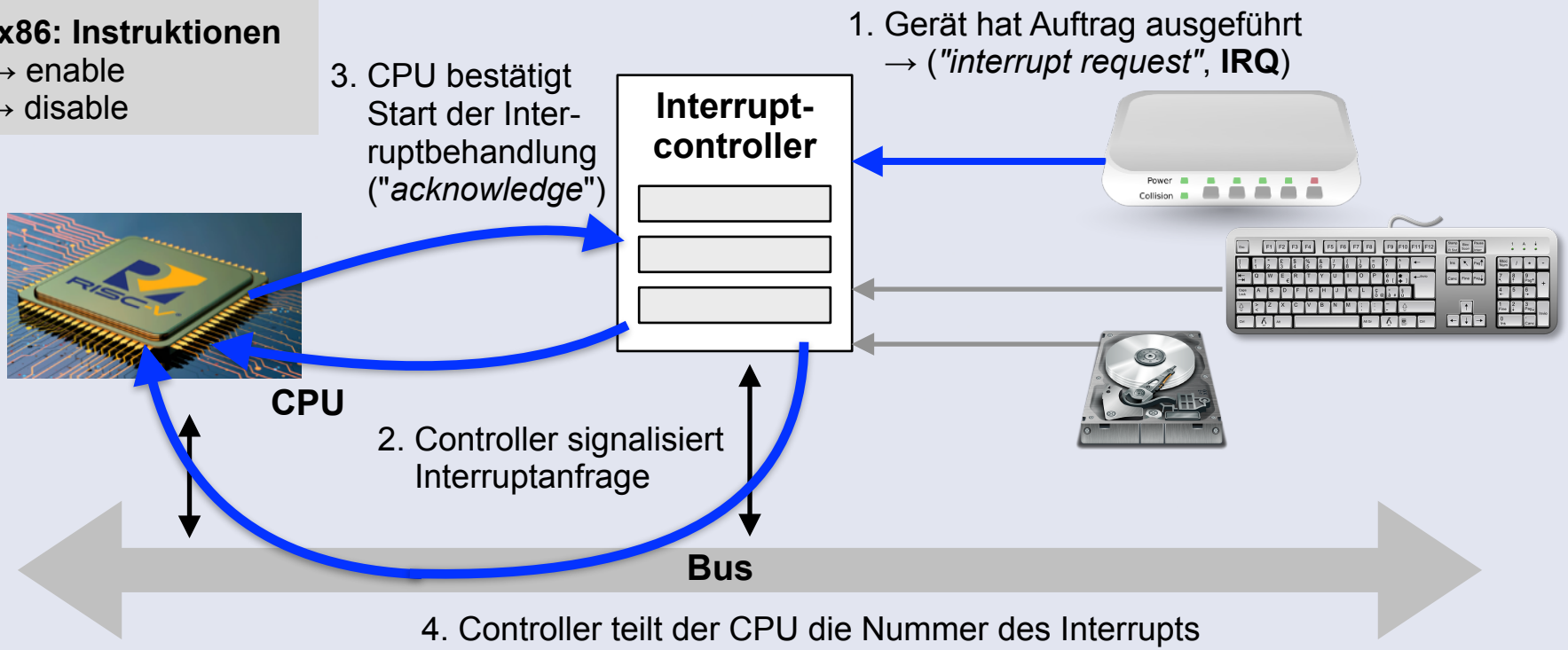
Universität Bamberg

…zeigen der Software an, dass sie reagieren muss

Software kann die Behandlung von IRQs deaktivieren, bei **RISC-V** durch **mie-Register**

Bei x86: Instruktionen
sti → enable
cli → disable

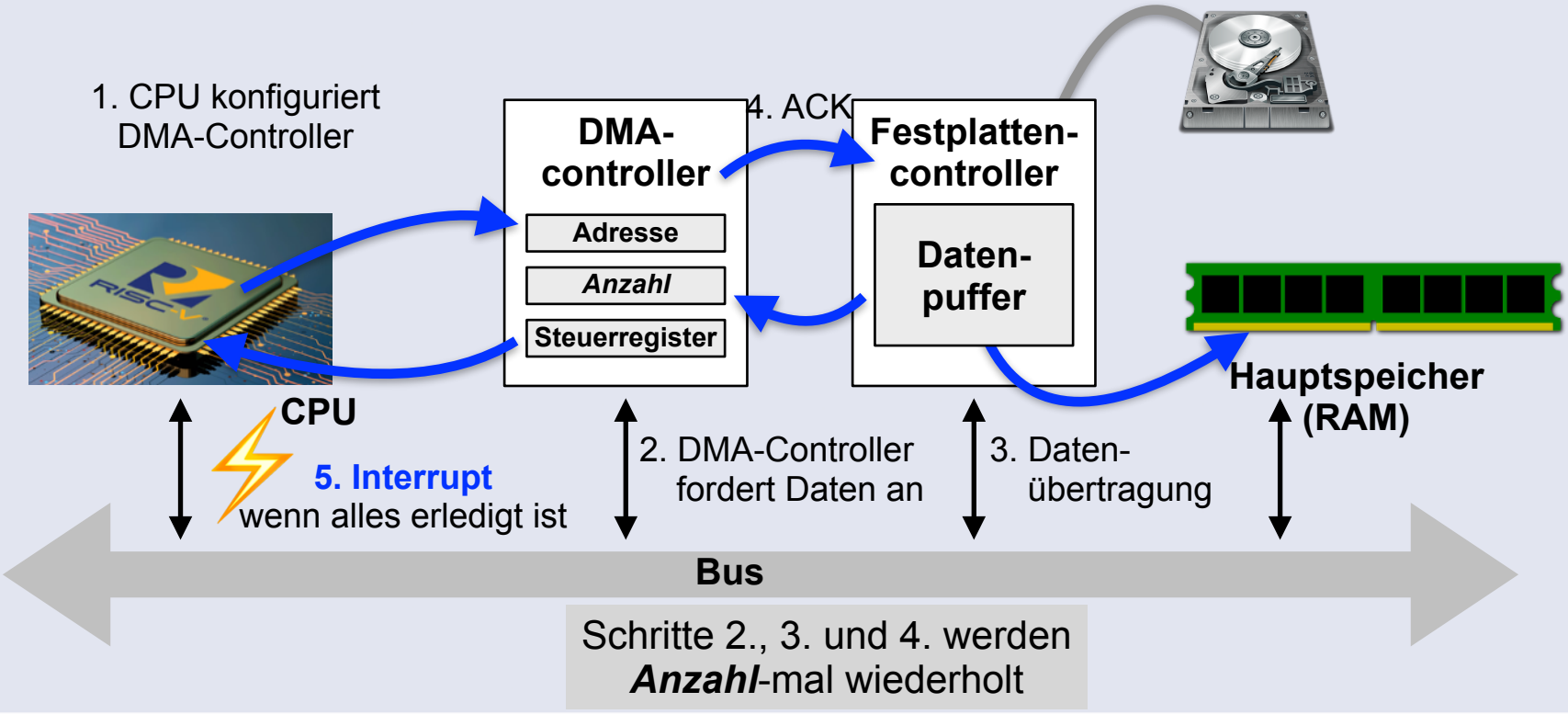
Interruptverarbeitungs-Sequenz auf Hardwareebene



Direkter Speicherzugriff (Direct Memory Access, DMA)

DMA wird bei komplexen Controllern verwendet, um Daten ohne Beteiligung der CPU vom Gerät zum Hauptspeicher (und zurück) zu übertragen

DMA-Transfersequenz



Exceptions, Traps und Interrupts

Was unterscheidet die Begriffe?

Diese Begriffe werden je nach Autor und auch Prozessorhersteller sehr unterschiedlich verwendet! Wir verwenden die RISC-V-Definitionen

- ***Exception***:
  - Ausnahmebedingung, die zur Laufzeit durch die Ausführung einer Instruktion der aktuellen CPU (gewollt oder ungewollt) erzeugt wird
    - Nicht erlaubter Speicherzugriff, Division durch Null, …
- ***Interrupt***:
  - Externes ***asynchrones*** Ereignis, *kann* bei einem RISC-V Prozessorcore zu einem nicht (vom laufenden Programm) erwarteten Kontrolltransfer verursachen
    - Interrupts vom Timer und von externen Geräten
- ***Trap***:
  - Kontrolltransfer an einen *trap handler*, der durch entweder eine Exception oder einen Interrupt ausgelöst wurde – Überbegriff

Interrupts bei RISC-V

Universität Bamberg

Alle Traps führen zu einem Sprung des Prozessors an die Adresse, die im Konfigurationsregister (csr) **mtvec** festgelegt ist

Wie kann der Prozessor die Art des Traps erkennen? **mcause**-Register

- ***Synchrone*** Traps (Exceptions) setzen höchstwertiges Bit auf 0
- ***Asynchrone*** Traps (Interrupts) setzen höchstwertiges Bit auf 1

Asynchrone Traps

mcause MSB = 1		
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2	*Reserved for future standard use*
1	3	Machine software interrupt
1	4	User timer interrupt
1	5	Supervisor timer interrupt
1	6	*Reserved for future standard use*
1	7	Machine timer interrupt
1	8	User external interrupt
1	9	Supervisor external interrupt
1	10	*Reserved for future standard use*
1	11	Machine external interrupt
1	12–15	*Reserved for future standard use*
1	≥16	*Reserved for platform use*

Synchrone Traps

mcause MSB = 0		
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	*Reserved*
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved for future standard use*
0	15	Store/AMO page fault
0	16–23	*Reserved for future standard use*
0	24–31	*Reserved for custom use*
0	32–47	*Reserved for future standard use*
0	48–63	*Reserved for custom use*
0	≥64	*Reserved for future standard use*

Interrupts bei RISC-V

Zwei *memory mapped* I/O-Einheiten unterstützen Interrupts:

- ***Core Local Interrupt Controller*** (CLINT):
  - Einer pro Prozessorkern ("hart" bei RISC-V)
  - Erzeugt Timer-Interrupts
  - Handhabt Software-Interrupts
- ***Platform Level Interrupt Controller*** *(PLIC)*
  - Einer pro Chip (SoC)!
  - Externe Interrupts
- **csr**s zur Interruptsteuerung:
- **MIE**: **I**nterrupt **E**nable/Disable auf globaler Ebene
- M**S**IE, M**E**IE, M**T**IE:
  - Interrupt enable/disable für **S**oftware, **E**xterne, **T**imer IRQs

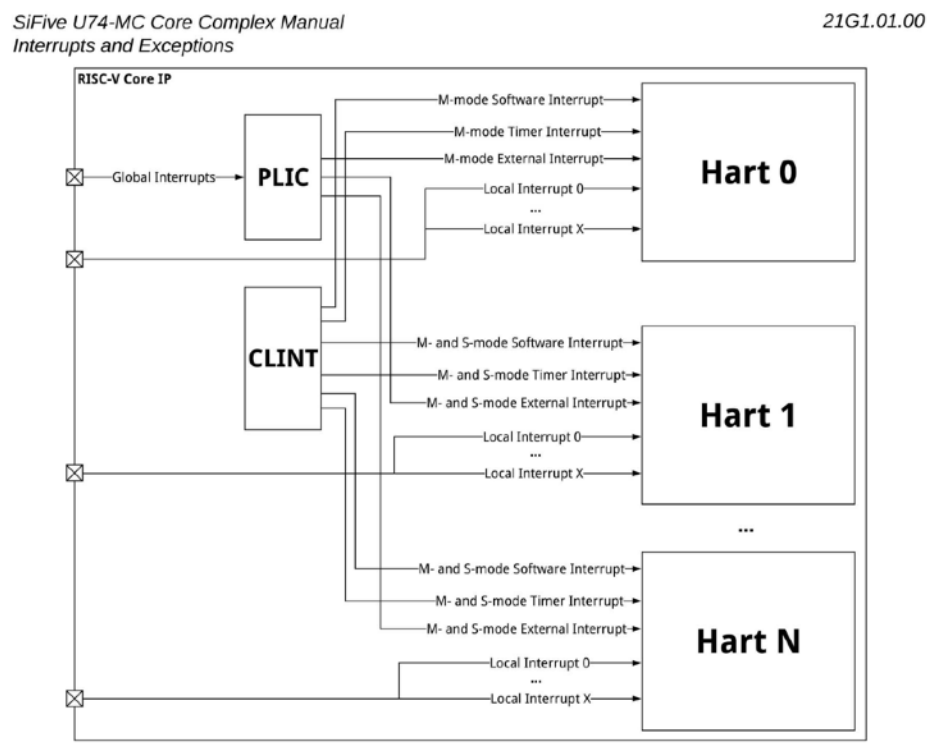
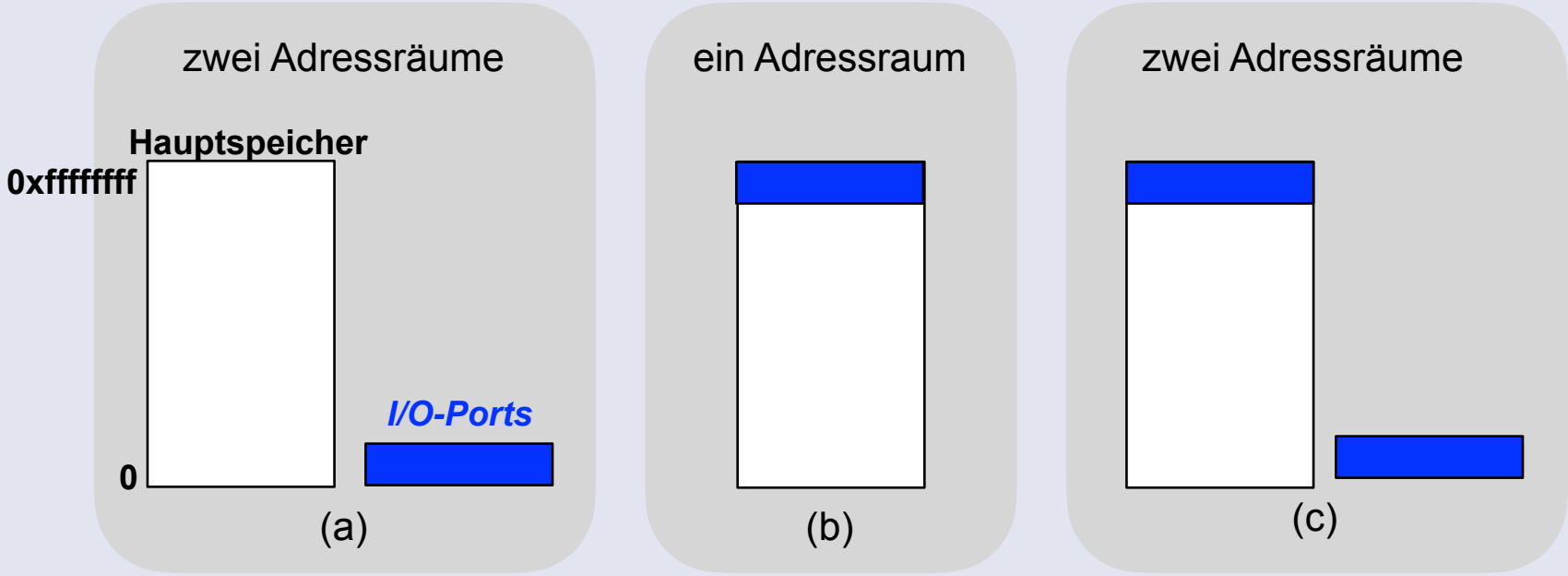


Figure 100: U74-MC Core Complex Interrupt Architecture Block Diagram

I/O-Adressierung

- Zugriff auf *Controller-Register* und *Controllerspeicher* abhängig von der Systemarchitektur



(a) Separater I/O-Adressraum

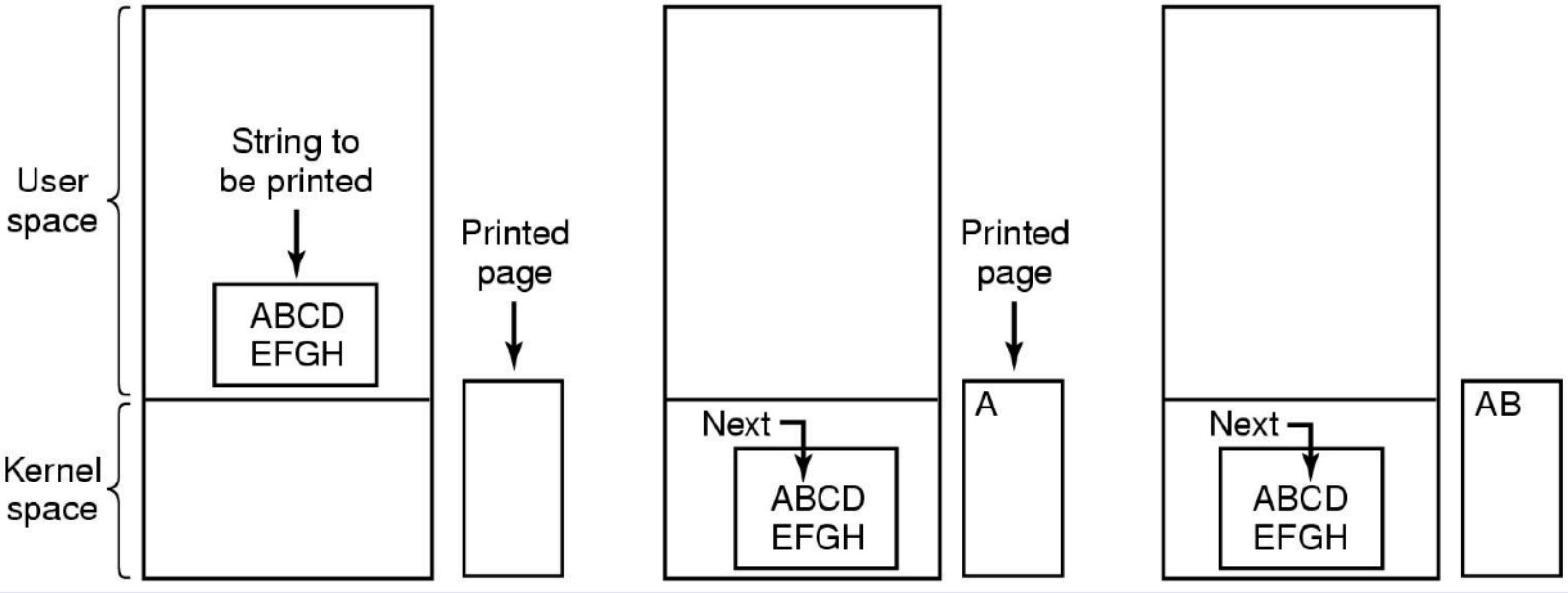
- Zugriff über besondere Maschineninstruktionen – bei x86

(b) Mit Hauptspeicher geteilter Adressraum (***memory mapped I/O***) – bei RISC-V

(c) Hybride Architektur (für Rückwärtskompatibilität bei x86)

Gerätetreiber

- In Abhängigkeit vom Gerät kann I/O durchgeführt werden über
  - ***Polling*** ("programmed I/O"),
  - ***Interrupts*** oder
  - ***DMA*** (Direkter Speicherzugriff)
- Beispiel: Drucken einer Seite Text



Source: Tanenbaum, Modern Operating Systems

Geräteinteraktion: **Polling** (programmierte Ein-/Ausgabe)

...impliziert ***aktives Warten*** auf ein I/O-Gerät

```
/* Kopiere Zeichen in Betriebssystempuffer p */
copy_from_user (buffer, p, count);

/* Schleife über alle Zeichen */
for (i=0; i<count; i++) {

  /* “Aktives” Warten bis Drucker bereit ist */
  while (*printer_status_reg != READY);

  /* Drucke ein Zeichen */
  *printer_data_reg = p[i];
}

return_to_user ();
```

Warum das Semikolon?

Pseudocode einer Betriebssystem-funktion, um Text mit Polling zu drucken

Interrupt-getriebene Ein-/Ausgabe

…impliziert, dass die CPU an einen anderen Prozess zugeteilt werden kann, während auf eine Antwort des Gerätes gewartet wird

```
copy_from_user (buffer, p, count);

/* Aktiviere Drucker-Interrupts */
enable_interrupts ();

/* Warte, bis Drucker bereit ist */
while (*printer_status_reg != READY);

/* Drucke erstes Zeichen */
*printer_data_reg = p[i++];

scheduler ();
return_to_user ();
```

Code, um die I/O-Operation zu starten

```
if (count > 0) {

  *printer_data_reg = p[i];
  count--;
  i++;

} else {

  unblock_user ();

}
acknowledge_interrupt ();
return_from_interrupt ();
```

Interrupthandler

DMA-getriebene Ein-/Ausgabe

...die CPU muss nicht mehr selbst die Daten zwischen I/O-Gerät und Hauptspeicher übertragen

- Weitere Reduktion der CPU-Last

```
copy_from_user (buffer, p, count);
set_up_DMA_controller (p, count);
scheduler ();
return_to_user ();
```

Code, um die I/O-Operation zu starten

```
acknowledge_interrupt ();
unblock_user ();
return_from_interrupt ();
```

Interrupthandler

Diskussion: Interrupts

- Interrupts sind ***die*** Quelle für asynchrones Verhalten im System
  - Können zu ***Wettlaufsituationen*** *(race conditions)* im BS-Kernel führen (→ später im Detail)
- Interruptsynchronisation
  - Einfacher Ansatz:
    "hart" Interrupts abschalten, während *kritischer* Abschnitt ausgeführt wird (MIR, sti/cli, etc.)
    - *Dabei können Interrupts verloren gehen!*
  - Bei modernen Systemen werden Interrupts durch mehrere Softwareebenen verarbeitet
    - Diese minimieren die Zeitdauer, in der Interrupts gesperrt sind
      - UNIX: *top half*, *bottom half*
      - Linux: *tasklets*
      - Windows: *deferred procedures*

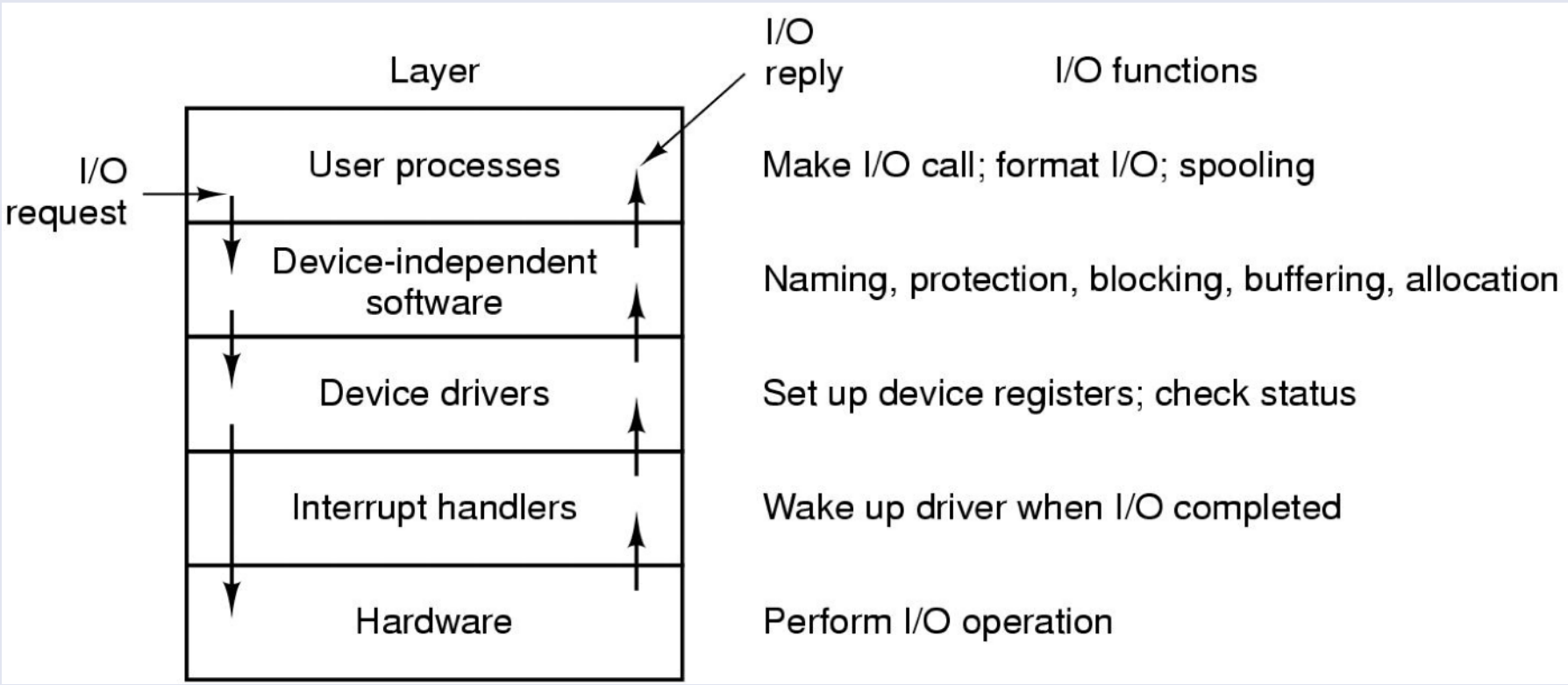
Diskussion: DMA

- **Caches**
  - Moderne Prozessoren nutzen *Datencaches*
    DMA umgeht den Cache!
  - DMA-Zugriffe werden also nicht im Prozessorcache registriert, damit kann der Cache veraltete Werte beinhalten
    - Einige Prozessoren unterstützen nicht-cachebare Adressbereiche für I/O-Operationen
- **Speicherschutz**
  - Moderne Prozessoren nutzen eine MMU *(memory management unit)*, um Prozesse voneinander und vom BS zu isolieren
    DMA umgeht den Speicherschutz! (aber: IOMMU)
  - Fehler bei Konfiguration von DMA-Transfers sind extrem kritisch
  - Anwendungsprozesse dürfen ***nie*** direkten Zugriff auf den DMA-Controller haben!

Geräte und Aufgaben des Betriebssystems

- Bereitstellen von **Geräteabstraktionen**
  - Gleichförmig, einfach, aber flexibel
- Bereitstellung von **grundlegenden I/O-Funktionen** *(I/O primitives)*
  - Synchroner und/oder asynchrone Verwendung von Geräten
- **Pufferung**
  - Wenn Gerät oder nutzendes Programm noch nicht ansprechbar sind
- **Gerätesteuerung**
  - So effizient wie möglich unter Berücksichtigung mechanischer Geräteeigenschaften (z.B. bei Festplatten)
- Verwalten von **Ressourcenzuweisungen**
  - Für Geräte mit gleichzeitigem Zugriff (z.B. Festplatten): welcher Prozess darf wann lesen/schreiben?
  - Für Geräte mit exklusivem Zugriff (z.B. Drucker): zeitbeschränkte Reservierungen
- Verwaltung von **Energiesparmodi**
- Unterstützung von **plug&play**
  - Hinzufügen und Entfernen von Geräten zur Laufzeit

Ebenen des I/O-Systems



Quelle: Tanenbaum, "Modern Operating Systems"

Unix-API für den Gerätezugriff

- Unix-Philosophie: ***alles ist*** *(bzw. sieht aus wie)* ***eine Datei***

- Peripheriegeräte werden über ***Spezialdateien*** angesprochen
  - Geräte werden wie gewöhnliche Dateien über Lese- und Schreiboperationen (*read* und *write*) angesprochen
  - Das Öffnen einer Spezialdatei erzeugt eine Verbindung zu dem zugehörigen Gerät, die durch den Gerätetreiber realisiert wird
  - Dies erlaubt Benutzern direkten Zugriff auf den Gerätetreiber

- **Block-orientierte Spezialdateien** (*block devices*)
  - Festplatten/SSDs, Bandlaufwerke, CD-ROM/DVD/BluRay

- **Zeichen-orientierte Spezialdateien** (*character devices*)
  - Serielle Schnittstellen, Drucker, Audioausgabe usw.

Geräteabstraktionen in Unix

- Geräte werden *eindeutig* durch ein Tupel identifiziert:
  - **Gerätetyp**
    - Block- oder zeichenorientiertes Gerät
  - **major device number**
    - Wählt einen bestimmten Gerätetreiber aus
  - **minor device number**
    - Wählt eines von mehreren von selben Gerätetreiber kontrollierten Geräte aus (z.B. eine von mehreren Festplatten)

Geräteabstraktionen in Unix

- Auszug aus dem Listing des /dev-Verzeichnisses, in dem üblicherweise die Spezialdateien abgelegt sind:

```
brw-rw---- me   disk 3,  0 2008-06-15 14:14 /dev/hda
brw-rw---- me   disk 3, 64 2008-06-15 14:14 /dev/hdb
brw-r----- root disk 8,  0 2008-06-15 14:13 /dev/sda
brw-r----- root disk 8,  1 2008-06-15 14:13 /dev/sda1
crw-rw---- root uucp 4, 64 2006-05-02 08:45 /dev/ttyS0
crw-rw---- root lp   6,  0 2008-06-15 14:13 /dev/lp0
crw-rw-rw- root root 1,  3 2006-05-02 08:45 /dev/null
lrwxrwxrwx root root     3 2008-06-15 14:14 /dev/cdrecorder -> hdb
lrwxrwxrwx root root     3 2008-06-15 14:14 /dev/cdrom -> hda
```

Zugriffs-rechte

Eigentümer und Gruppe

major und minor ID

Modifikations-datum&-zeit

Name der Spezialdatei

`c`: *character device*
`b`: *block device*
`l`: *link*

Datei- und Gerätezugriff in Unix

Ein kurzer Überblick... (Details in den *man pages*...)

- `int open(const char *devname, int flags)`
  - "Öffnet" ein Gerät und gibt einen Dateideskriptor zurück
- `off_t lseek(int fd, off_t offset, int whence)`
  - Positioniert den Lese-/Schreibzeiger (relativ zum Anfang der Datei) – nur für wahlfrei zugreifbare Dateien
- `ssize_t read(int fd, void *buf, size_t count)`
  - Liest bis zu `count` Bytes von Deskriptor `fd` in den Puffer `buf`
- `ssize_t write(int fd, const void *buf, size_t count)`
  - Schreibt `count` Bytes von Puffer `buf` in Datei mit Deskriptor `fd`
- `int close(int fd)`
  - "Schließt" ein Gerät. Danach kann der Dateideskriptor `fd` nicht mehr zum Zugriff auf die Datei verwendet werden

Gerätespezifische Funktionen

- Spezifische Eigenschaften eines Geräts einstellen mit `ioctl`:

```
IOCTL(2)          Linux Programmer's Manual          IOCTL(2)

NAME
       ioctl - control device

SYNOPSIS
       #include <sys/ioctl.h>

       int ioctl(int d, int request, ...);
```

Beispiel: Geschwindigkeit von Ethernet- oder UART-Schnittstellen

- Generischer Aufruf, aber gerätespezifische Semantik:

```
CONFORMING TO
    No single standard. Arguments, returns, and semantics of
    ioctl(2) vary according to the device driver in question
    (the call is used as a catch-all for operations that
    don't cleanly fit the Unix stream I/O model). The ioctl
    function call appeared in Version 7 AT&T Unix.
```

Verwaltung von Unix-Gerätetreibern

- Wie bildet der Kernel den Zugriff auf eine Gerätedatei via open/read/… auf den jeweiligen Gerätetreiber ab?
  - Die **device switch table** enthält **Funktionszeiger** auf die von einem Treiber für sein spezifisches Geräte implementierten Funktionen
  - Frühe Unix-Systeme: **statische Gerätekonfiguration**
    - Kernel musste zum Hinzufügen und Entfernen von Treibern neu compiliert werden
  - Heute: **Kernelmodule**
    - Treiber sind zur Laufzeit lad- und installierbar
    - Mögliches **Sicherheitsproblem** – Module laufen im Kernel-Adressraum und mit Kernel-Privilegien!

Device switch table

Universität Bamberg

conf.h

```
/* Declaration of block device switch. Each entry (row) is the only link between the main
 * unix code and the driver. The initialization of the device switches is in the file conf.c. */
struct bdevsw
{
        int     (*d_open)();
        int     (*d_close)();
        int     (*d_strategy)();
        int     (*d_dump)();
        int     (*d_psize)();
        int     d_flags;
};
struct  bdevsw bdevsw[];

/* Character device switch */
struct cdevsw
{
        int     (*d_open)();
        int     (*d_close)();
        int     (*d_read)();
        int     (*d_write)();
        int     (*d_ioctl)();
        int     (*d_stop)();
        int     (*d_reset)();
        struct tty *d_ttys;
        int     (*d_select)();
        int     (*d_mmap)();
};
struct  cdevsw cdevsw[];
```

conf.c

```
struct bdevsw   bdevsw[] = {
{ hpopen, nulldev, hpstrategy, hpdump, /*0*/
  hpsize,       0 },
{ htopen, htclose, htstrategy, htdump, /*1*/
        0,  B_TAPE },
{ upopen, nulldev, upstrategy, updump, /*2*/
  upsize,       0 },
…
}
```

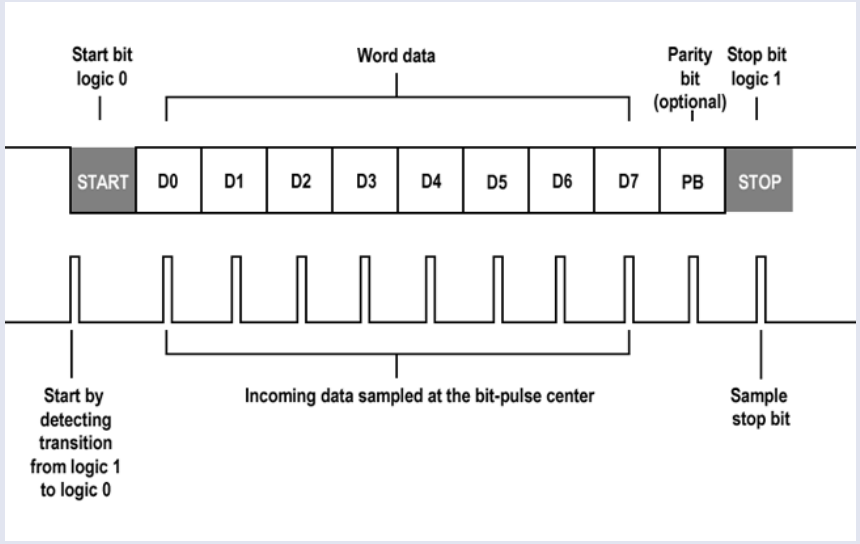
Major device number:

Index in das `devsw`-Array!

Wie können wir Text ausgeben?

- Verschiedene Optionen (Hardware-Schnittstellen)
  - Videoausgabe (Framebuffer+Display z.B. HDMI oder VGA)
  - Kleines LC-Display (über SPI oder I2C angebunden)
  - Serielle Schnittstellen (UART – universal asynchronous receiver/transmitter) – **einfach!**

- UARTs sind ICs, die Zeichen **bitseriell** übertragen
  - Ein Bit nach dem anderen über eine **Dreidraht-Verbindung** (Senden, Empfangen und Masseleitung)

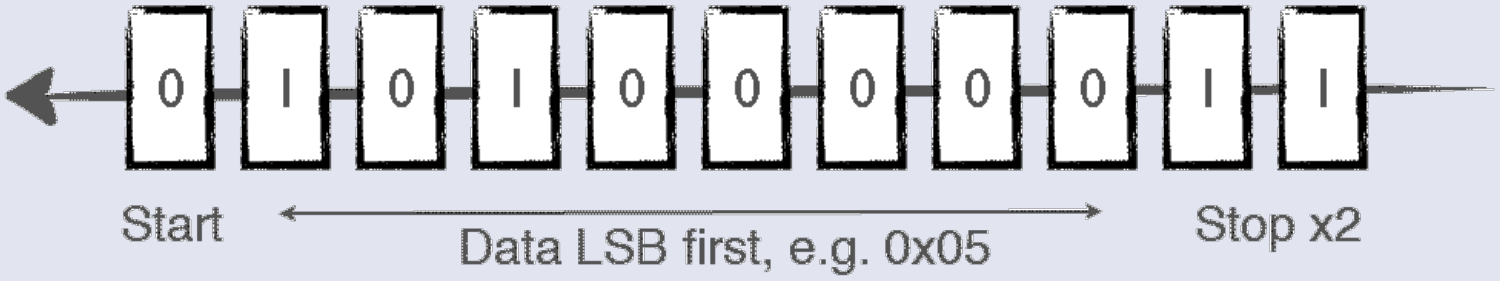


Serielle und parallele Datenübertragung

Serielle Datenübertragung ist eine der meistverwendeten Kommunkationsmethoden in Compuitersystemen

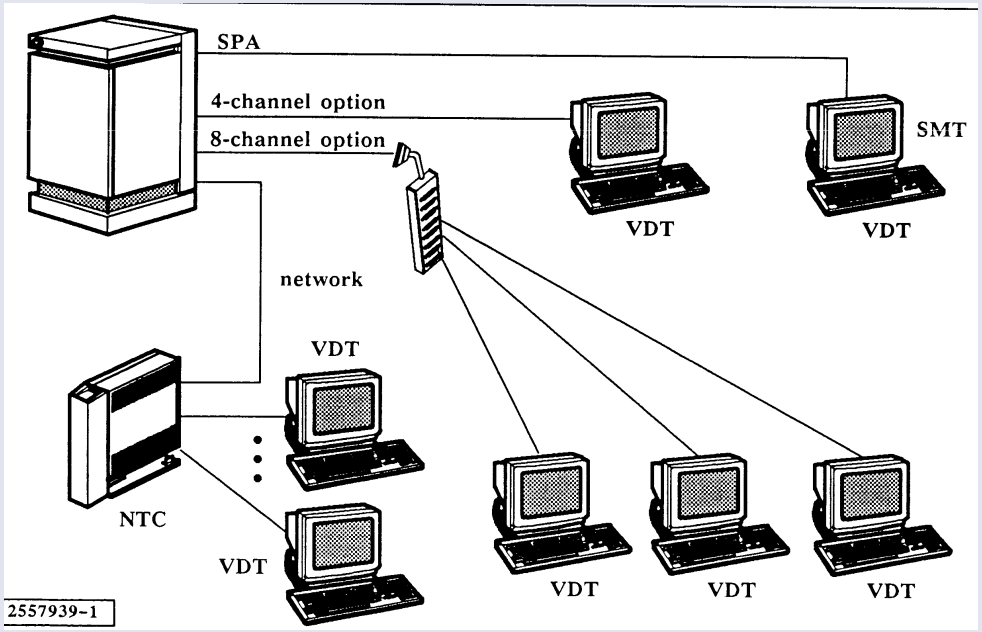
- *Asynchron (selbstgetaktet):* RS232
  - über einen UART – Universal Asynchronous Receiver/Transmitter
- *Synchron (explizit getaktet):* SPI, I2C, I2S, SATA, PCIe, …

- Die Bits eines Bytes (oder Words usw.) werden *eins nach dem anderen* über eine einzelne Leitung (pro Richtung) übertragen
  - Dagegen: *parallele* Schnittstellen, z.B. IDE/ATA, Centronics, SCSI
    - Übertragen 8 oder mehr Bits gleichzeitig über parallele Leitungen



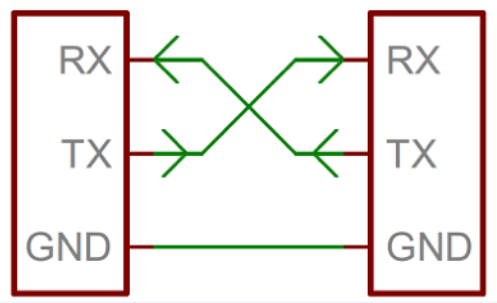
UARTs in Computern

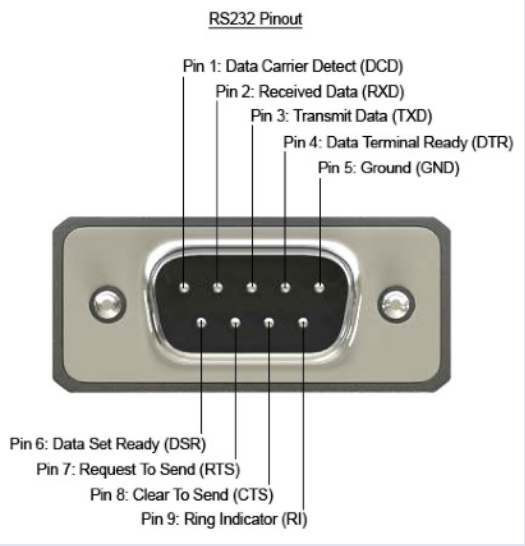
- Asynchrone Datenübertragung wurde oft zur Anbindung von Terminals an Mehrbenutzer-Computersysteme genutzt
  - Tastaturdaten des Terminals werden über dessen TX-Ausgang an den TX-Eingang des Computers gesendet (an eine bestimmte RS232-Schnittstelle von vielen)
  - Anzeigedaten werden über den TX-Ausgang des Computers an den RX-Eingang des Terminals gesendet (an der selben RS232-Schnittstelle)
- Meist ist eines der Terminals als *Konsole* konfiguriert – das vom Systemadministrator genutzte Terminal
- Terminalfenster (z.B.. xterm auf Linux/X11 oder Terminal.app bei macOS) und Textbildschirme beiLinux/Unix verhalten sich wie Terminals



RS232 und UARTs

- Asynchrone Datenübertragung mit RS232 verwendet drei Leitungen:
  - Empfangen (*receive*, RX), Übertragen (*transmit*, TX), Masse (*ground*, GND)
  - Verbindung von TX und RX (*cross-over*) über Kreuz
  - Daten, die auf einer Seite über TX gesendet werden, kommen auf der anderen Seite an RX an
  - Üblicher Steckverbinder
    - 9-pin "D-Sub"
    - optionale Leitungen werden für Zusatz-funktionen verwendet (Handshake, Verbindungserkennung usw.)
- UARTs – Universal Asynchronous Receiver/Transmitter verbinden den Prozessor eines Computers mit einer RS232-Schnittstelle
  - Üblicher UART-Chip: NS16550





Der 16550-UART

- Der 16550-UART (im IBM PC seit 1981 als 8250 bekannt) ist einer der Standard-UART-"Chips"
  - Heute meist kein eigenes IC mehr, sondern integriert in einen größeren Chip, ein sog. *system-on-chip* (SoC)
- Das *Datenblatt* des UART enthält eine Liste der *Register* des Chips und eine Beschreibung der Funktionen
- Register sind oft als aufeinanderfolgende Bytes zugreifbar, Start an der UART-Basisadresse
- Register mit DLAB=1 sind nur verfügbar, wenn LCR Bit 7 = 1 gesetzt

Offset	DLAB	R/W	Function
0	0	R	Receive buffer reg. RBR
0	**0**	**W**	**Transmit hold reg. THR**
0	1	R/W	Divisor latch (LSB)
1	0	R/W	Interrupt enable reg. IER
1	1	R/W	Divisor latch (MSB)
2	X	R	Interrupt identific. reg. IIR
2	X	W	FIFO control reg. FCR
3	X	R/W	Line control reg. LCR
4	X	R/W	Modem control reg. MCR
5	**X**	**R/W**	**Line status reg. LSR**
6	X	R/W	Modem status reg. MSR
7	X	R/W	Scratch register

16550 Line Status Register (LSR)

- Das LSR gibt den Status des UART wieder:
  - Neues Zeichen kann versendet werden
  - Neues Zeichen wurde empfangen
  - Diverse Fehlerzustände

Bit	Name	Bedeutung
0	Data Ready	Zeichen in Receive Holding Register verfügbar
1	Overrun Error	Neues Zeichen angekommen, vorheriges nicht abgeholt
2	Parity Error	Paritätsfehler entdeckt
3	Framing Error	Fehlendes/falsches Stoppbit
4	Break Interrupt	Datenleitung permanent auf 0 gehalten
5	THR Empty	Neues Zeichen kann in THR geschrieben werden
6	Transmitter Empty	Vorheriges Zeichen wurde vollständig versendet
7	FIFO data Error	Problem mit Daten im Sende- oder Empfangspuffer

Abbildung des UART in Software

Wir definieren eine Struktur, die die Register beschreibt:

```
typedef char uint8_t; // oder auch mit #include <stdint.h>

struct uart {
   uint8_t THR; // transmit hold register (offset 0)
   uint8_t IER;
   uint8_t IIR;
   uint8_t LCR;
   uint8_t MCR;
   uint8_t LSR; // line status register (offset 5)
};

volatile struct uart* uart0 = (volatile struct uart*)0x10000000;
```

- Die **uart**-Struct bildet die UART-Registers auf Speicheradressen ab
- Die Startadresse der struct wird **initialized** mit `0x1000_0000`;
  - Das erste Element (THR) ist bei relativer Adresse `0` (`0x1000_0000`)…und das LSR bei relative Adresse 5 (`0x1000_0005`)

Polling der UART-Datenübertragung

Universität Bamberg

```
struct uart {
    …  // wie vorhin
};

volatile struct uart* uart0 = (volatile struct uart*)0x10000000;

void putachar(char c) {
    while ((uart0->LSR & (1<<5)) == 0)
        ;               // nichts tun - warten bis Bit 5 im LSR = 1
    uart0->THR = c;     // dann das Zeichen in THR schreiben
}

void printstring(char *s) {
    while (*s) {        // solange das aktuelle Zeichen != Null ist
        putachar(*s); // …gib das Zeichen aus
        s++;            // …und fahre mit dem nächsten Zeichen fort
    }
}
```

Wir werden weich…

Betriebssysteme stehen im Rest des Semesters im Mittelpunkt!

- Aufgaben des Betriebssystems, **Programme** vs. **Prozesse**
  - Prozesse und das Betriebssystem
  - Vom Programm zum Prozess
    - Programmrepräsentation, Prozess vs. Programm, Zustand
- **Ressourcen**zugriff und **-multiplexing**
  - I/O-Schichtensystem, Unix-Geräteabstraktionen, Zugriffsrechte
  - Pufferung
- **Nebenläufigkeit** und **Parallelität**
  - Kritische Abschnitte, Koordination/Synchronisation, Konflikte
- **Virtueller Speicher**
  - Funktionsweise (HW/SW), Nutzung, Isolation

Referenzen

[1] Frank Slomka, Michael Glaß
Grundlagen der Rechnerarchitektur
Von der Schaltung zum Prozessor

[2] David Harris, Sarah Harris
Digital Design and Computer Architecture, RISC-V Edition

[3] Remzi Arpaci-Dusseau, Andrea Arpaci-Dusseau
Operating Systems: Three Easy Pieces
https://pages.cs.wisc.edu/~remzi/OSTEP/