

Betriebssysteme

Ein-/Ausgabe

Prof. Dr.-Ing. Torben Weis
University Duisburg-Essen

Gerätearten

- Blockorientierte Geräte mit Random-Access
 - Beispiel: Festplatte
 - Daten werden blockweise bearbeitet
 - Blöcke können direkt adressiert werden (seeking)
 - Ähnlich wie Speicherblöcke
- Sequentielle Zeichenorientierte Geräte
 - Beispiel: Maus, Terminal, Drucker, Netzwerkschnittstellen
 - Daten werden Zeichen für Zeichen gelesen
 - Daten sind nicht adressierbar
 - Man kann also nicht „zurückspulen“

Gerätearten

- Fragwürdige Klassifizierung
 - Viele Geräte können sowohl das eine als auch das andere sein
- Beispiel Bandlaufwerk
 - Gilt als zeichenorientiert
 - Man könnte die Daten aber in Blöcken zusammenfassen
 - Durch Zurückspulen kann man einzelne Blöcke adressieren
- Beispiel Textausgabe
 - Gilt als zeichenorientiert
 - Grafikspeicher ist oft in den Hauptspeicher eingebunden und lässt sich beliebig adressieren

Gerätearten

- Zeichen- oder Blockorientiert
- Sequentiell oder Random-Access
- Synchron oder Asynchron
 - D.h. die Antwortzeiten sind vorhersehbar oder nicht
- Teilbar oder nicht
 - Beeinflusst die Ressourcenzuteilung
- Geschwindigkeit
- Lesen & Schreiben, nur Lesen, nur Schreiben

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

Zugriff auf Geräte

- Durch das Betriebssystem
 - Spezielle CPU Anweisungen (IN, OUT)
 - Memory-Mapped IO
 - DMA (Direct Memory Access)
- Durch Benutzerprogramme
 - Kein direkter Zugriff auf die Hardware erlaubt
 - Bedeutet zusätzlichen Aufwand
 - Manchmal zuviel: Beispiel 3D-Spiele und Grafikkarte
 - Spezielle BS-Funktionen
 - Dateiabstraktion
 - `/dev` Verzeichnis unter UNIX

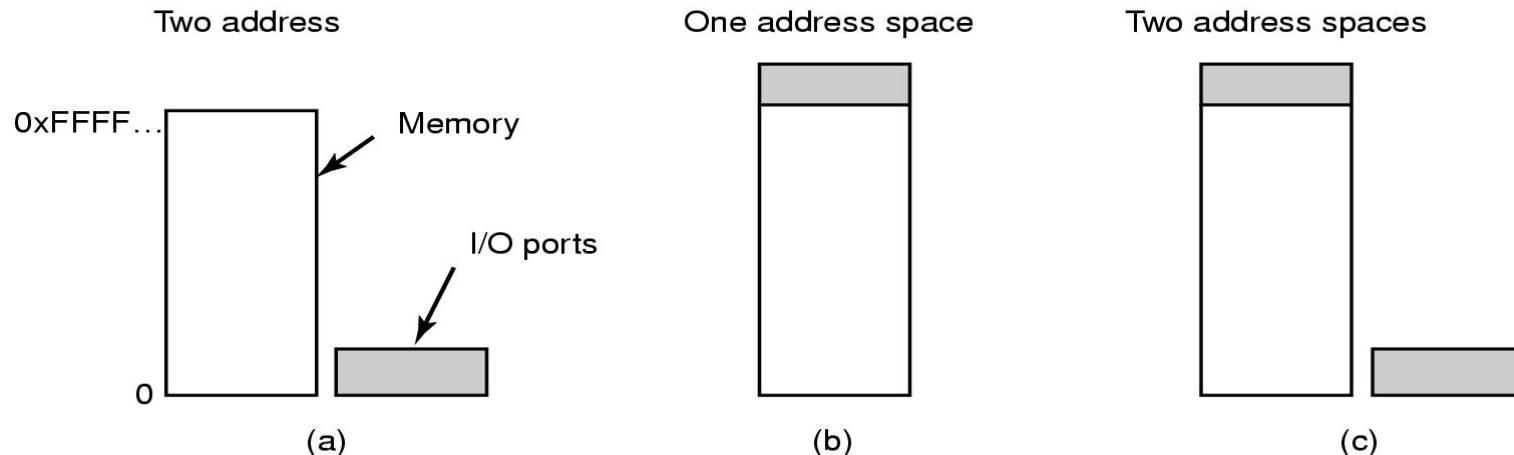
IO-Ports

- Ansprechbar über eine Adresse
 - Beim PC im Bereich 0x0000 bis 0xFFFF
- Ein Port hat normalerweise 4 Register
 - Status Register
 - Read only
 - Fehlercode, Befehl ausgeführt, Daten zum Lesen bereit
 - Control Register
 - Beispiel Serieller Port: full-duplex, half-duplex, parity checking, word length (7 Bit oder 8 Bit)
 - Command Register
 - Die CPU schickt Aufträge an den Controller
 - Data-In Register
 - Die CPU kann hier Daten vom Controller lesen
 - Data-Out Register
 - Die CPU kann Daten an den Controller schicken

IO-Ports im PC

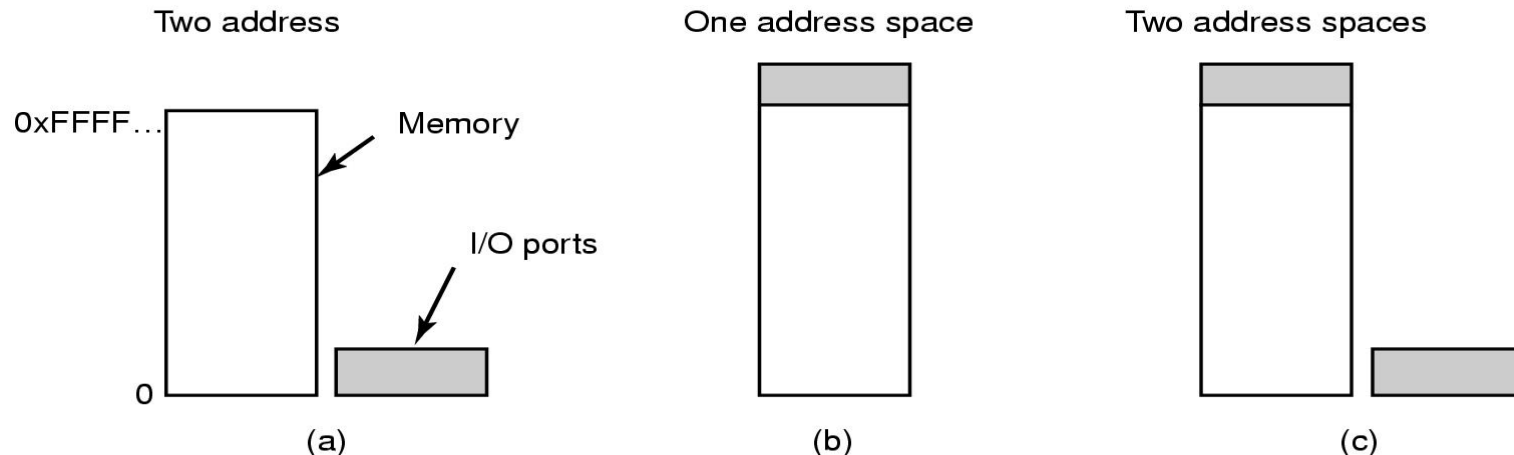
IO Adresse	Gerät
000-00F	DMA Controller
020-021	Interrupt Controller
040-043	Timer
200-20F	Game Controller
2F8-2FF	Serial Port (zweiter)
320-32F	HD Controller
378-37F	Parallel Port
3D0-3DF	Graphics Controller
3F0-3F7	FD Controller
3F8-3FF	Serial Port (erster)

Memory-Mapped versus IN/OUT Instruktionen



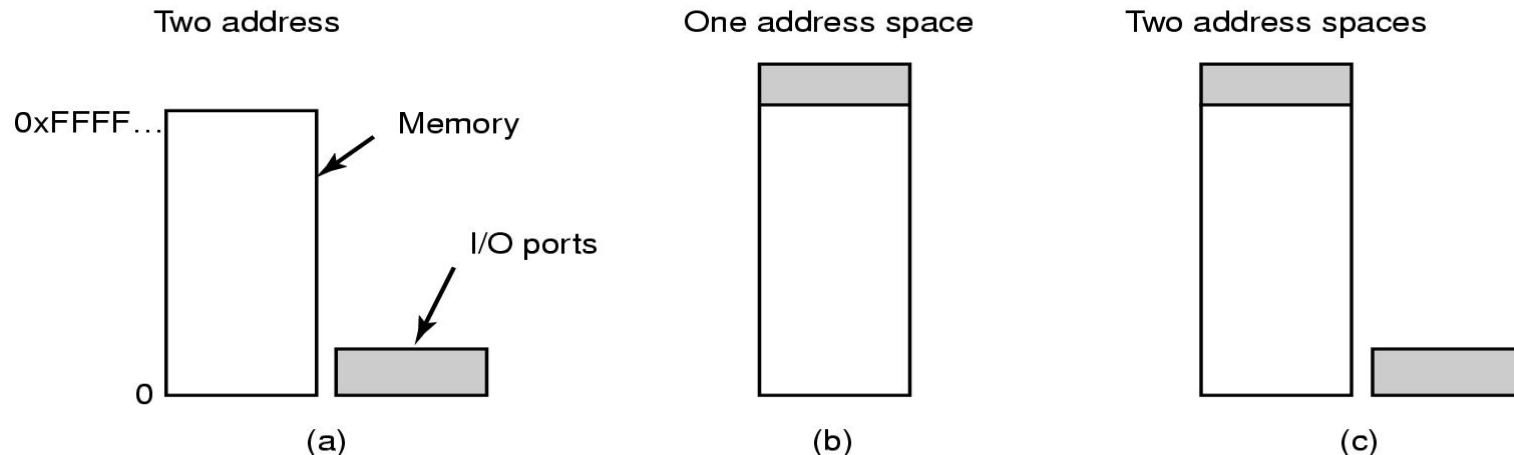
- (a) Getrennte Adressräume
 - `IN R0, 4` // Lade IO-Port 4 in Register R0
 - `MOV R0, 4` // Lade Speicherwort 4 in Register R0
- Probleme
 - Zugriff auf IN/OUT Instruktionen nur dem BS erlaubt
 - CPU muss Zugriff reglementieren

Memory-Mapped versus IN/OUT Instruktionen



- (b) Gemeinsamer Adressraum
 - `MOV R0, 0xFFF4 // Lade IO-Port 4 in Register R0`
- Vorteil
 - Zugriffsschutz durch Seitentabellen
 - Effizienteres Auslesen & Verarbeiten (dazu gleich mehr)
 - Zugriff durch C / C++ direkt möglich
- Probleme
 - Verkompliziert die Busarchitektur (dazu gleich mehr)
 - Caching muss abgeschaltet werden

Memory-Mapped versus IN/OUT Instruktionen



- Hybrider Ansatz mit zwei Adressräumen
 - `MOV R0, 0xFFF4` // Lade IO-Puffer 4 in Register R0
 - `IN R0, 4` // Lade IO-Port 4 in Register R0
- Dieser Ansatz wird von IBM PCs verfolgt
 - Speicherbereich 0 – 64 KB für IO-Ports
 - Speicherbereich von 640 KB bis 1 MB für IO-Puffer

Memory-Mapped versus IN/OUT Instruktionen

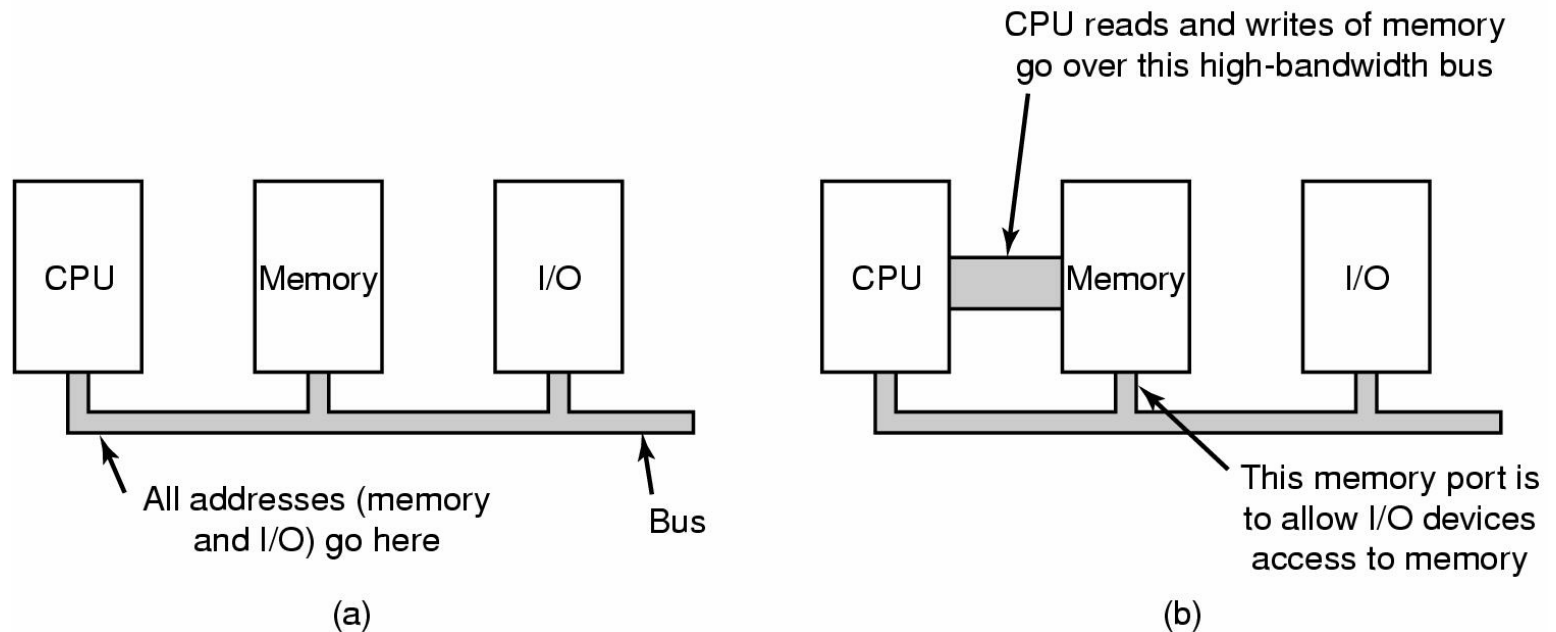
- CPU hat viele spezialisierte Instruktionen für den Speicherzugriff
 - Indirekte Adressierung
 - Lesen und Testen in einem Befehl
- IO-Daten können so effizienter weiterverarbeitet werden
 - IN/OUT müssen IO-Daten erst in ein CPU Register laden

- Beispiel TEST Befehl

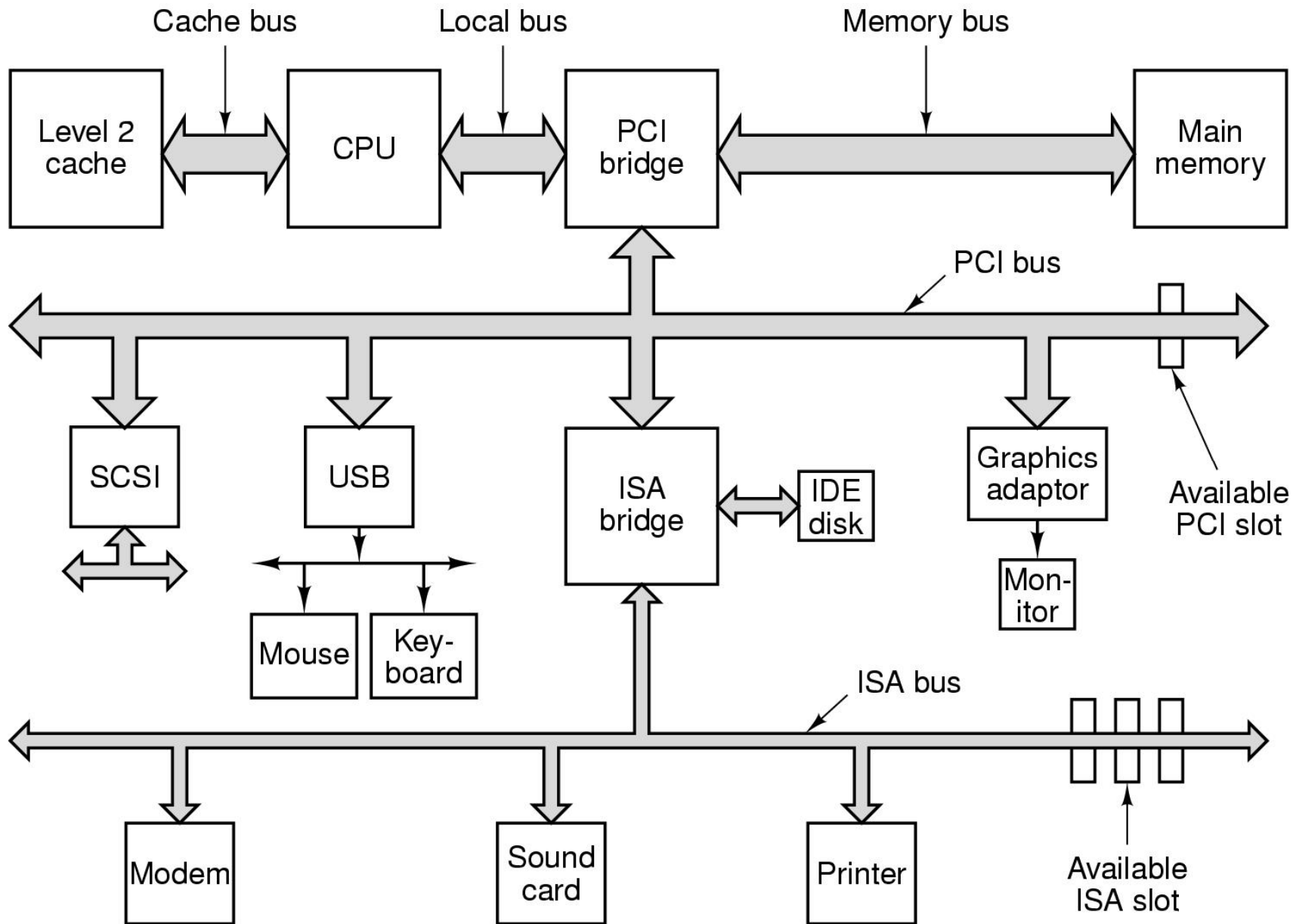
```
LOOP:    TEST 4  
         BEQ READY  
         JMP LOOP
```

```
READY:  ...
```

Memory-Mapped IO und der Speicherbus



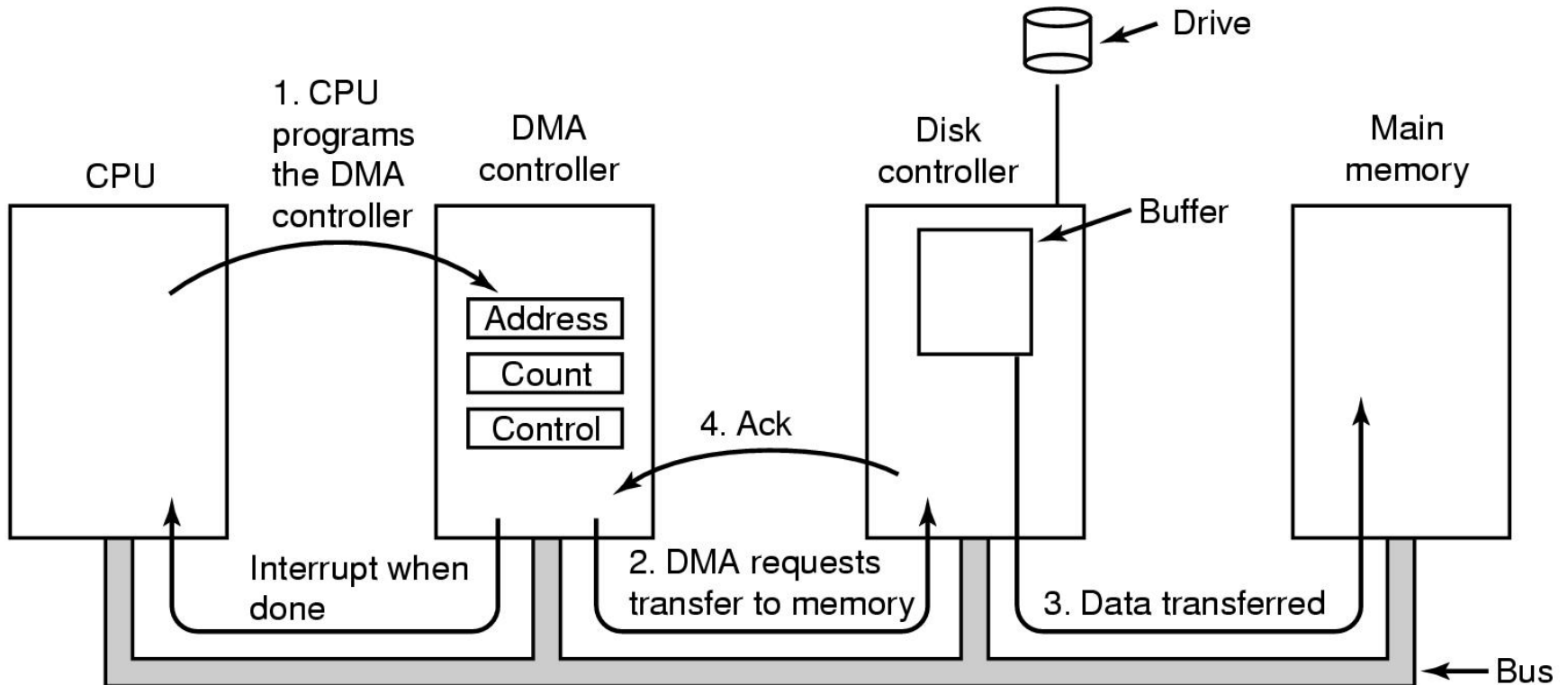
- (a) System mit nur einem Speicherbus
 - IO beantwortet Anfragen im IO Speicherbereich
- (b) System mit sehr schnellem Speicherbus
 - Problem: Wie soll die IO Komponente mitbekommen, dass eine Adresse im IO Speicherbereich gelesen wird?



Direct Memory Access (DMA)

- IO-Ports und Memory-Mapped IO brauchen viel CPU Zeit
 - Die CPU muss jedes Wort einzeln auslesen und in dem Hauptspeicher ablegen (oder andersrum)
 - Probleme z.B. bei Soundwiedergabe in Spielen
- DMA Controller entlasten die CPU
 - Die CPU bestimmt
 - ... welches Gerät
 - ... wieviele Bytes
 - ... wohin speichern soll
 - Der DMA Controller übernimmt den Rest
 - Bei Fertigstellung wird ein Interrupt ausgelöst

Direct Memory Access (DMA)



Bus Mastering

- Der Bus verbindet CPU, Speicher und Peripherie
- Wer kontrolliert den Bus? Der „Bus Master“
- Einfachste Architektur
 - Die CPU ist der einzige Bus Master
 - Jede IO Operation geht über die CPU
- Architektur mit „third-party DMA“
 - Beispiel: ISA Standard in alten IBM PCs
 - DMA Controller kann Bus Master werden
 - Der DMA Controller entzieht der CPU den Bus, überträgt alle Daten und gibt den Bus an die CPU zurück
 - Die CPU kann derweil rechnen, aber nicht auf den Hauptspeicher oder IO zugreifen

Bus Mastering

- Architektur mit „first-party DMA“
 - Beispiel: PCs mit PCI Bus
 - Jedes IO-Gerät hat einen eigenen DMA Controller
 - Jedes IO-Gerät kann Bus Master werden
 - Die CPU „programmiert“ das IO-Gerät, welches dann selbständig Daten mit dem Speicher austauscht
 - Aufwändigere Hardware
 - Jedes IO-Gerät muss Bus Mastering beherrschen d.h. einen Microcontroller mitbringen
 - Heute kein echter Kostenfaktor mehr

Blitter

- BLIT = BLock Image Transfer
- BitBLT = Bit Block Transfer
- Der Blitter ist ein Co-Prozessor
 - Hilft der CPU beim Umkopieren von Daten im Speicher
 - Ähnlich einem DMA Controller
- Anwendungsgebiet
 - Bewegte Grafiken mit großen „Sprites“
 - CPU wird nicht mit Kopieren der Sprites aufgehalten
 - Beispielsweise im Commodore Amiga
- Heute überflüssig
 - CPUs sind schnell genug
 - Moderne Grafikkarten sind noch viel schneller und haben einen eigenen Speicher

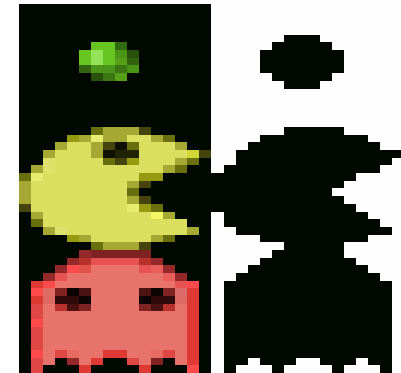
Blitter

- Beispiel PacMan
- Der Hintergrund des Spiels ist oben abgebildet
- Spielfiguren und Punkte (unten) müssen auf das Spielfeld gemalt werden
 - Diesen Job übernimmt der Blitter
- Notiz
 - Farbkodierung 0xRRGGBB
 - Weiß hat Farbwert 0xFFFFFFFF
 - Schwarz hat Farbwert 0x000000



Blitter

- Jede Spielfigur gibt es in „bunt“ und als Maske
- Schritt 1:
 - BitBlit der Maske mit der Operation AND
 - Resultat siehe unten
- Schritt 2:
 - BitBlit der „bunten“ Bitmap mit der Operation OR
- Der Blitter lohnt(e) erst bei größeren Bitmaps als hier gezeigt

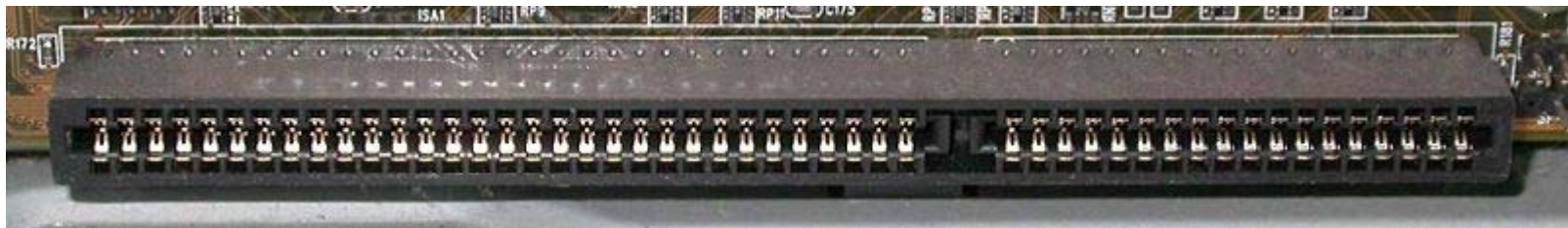


Alte PC Bussysteme

- 8Bit ISA (IBM XT-Architektur)
 - 4,77 MHz
 - 8 MByte/sec (theoretisch)

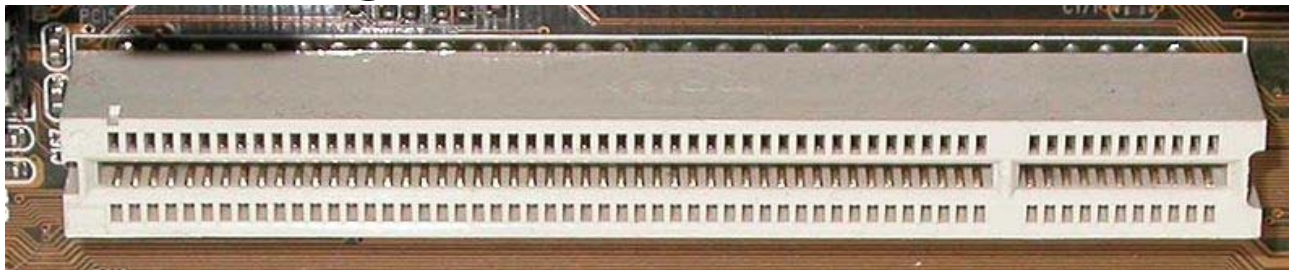


- 16Bit ISA (IBM AT-Architektur)
 - 8,33 MHz
 - 16 MByte/sec (theoretisch)



Neuere PC Bussysteme

- PCI = Peripheral Component Interconnect
 - 33 oder 66 MHz
 - Maximal etwa 500 MByte/sec
 - PCI-ISA Bridge erlaubt Betreiben alter Karten



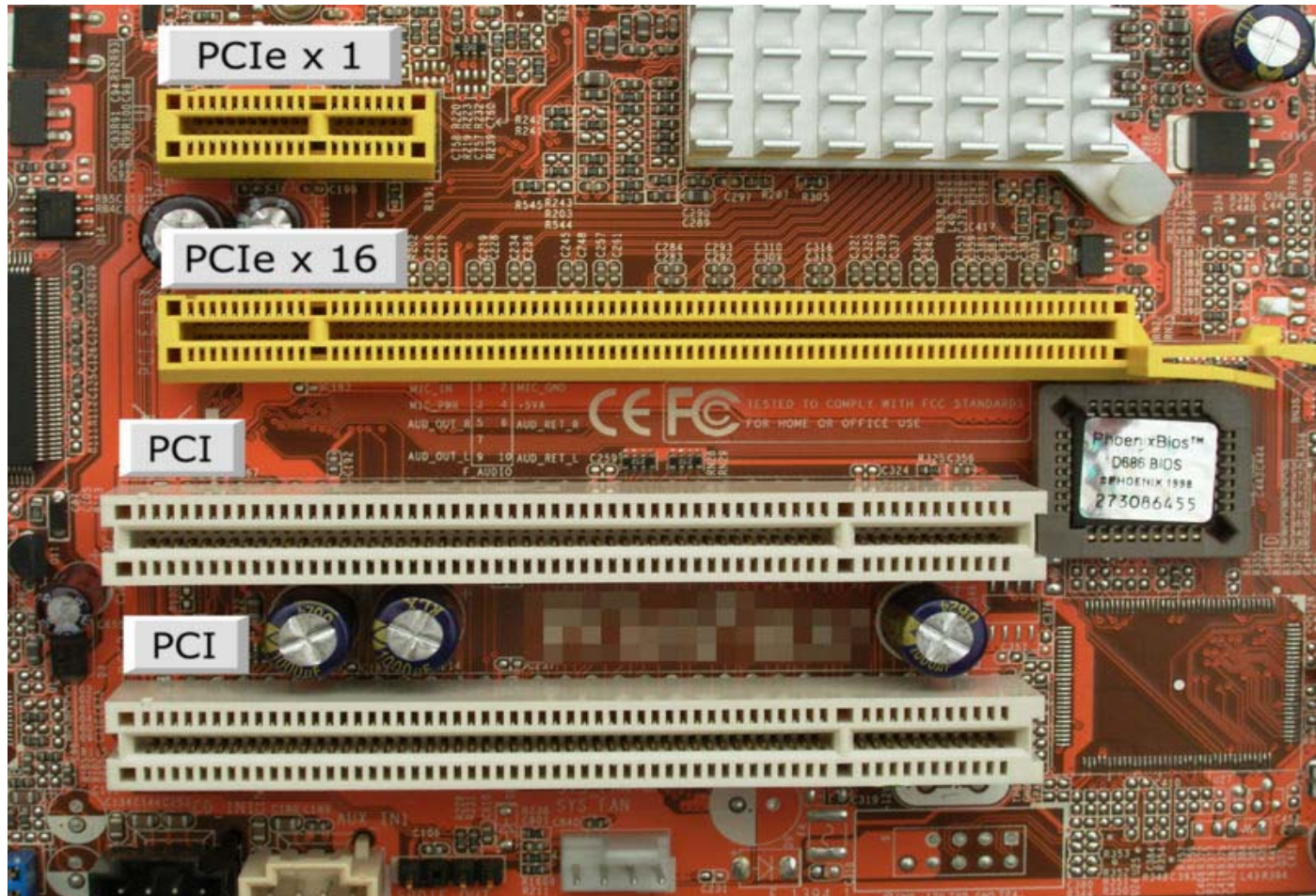
- AGP = Accelerated Graphics Port
 - Eigentlich kein Bus, sondern Punkt-zu-Punkt Verbindung
 - Erlaubt der Grafikkarte schnellen Zugriff auf den Hauptspeicher

Neuere PC Bussysteme



- PCI-Express
 - Löst PCI und AGP ab
 - Mehrere „Lanes“ lassen sich kombinieren
 - Jede Lane schafft 250 MByte/sec bei 1,25 GHz
 - PCIe x1 belegt genau eine Lane
 - PCIe x16 kombiniert 16 Lanes
 - Das langt auch für anspruchsvolle 3D-Karten
 - AGP wird überflüssig

Neuere PC Bussysteme



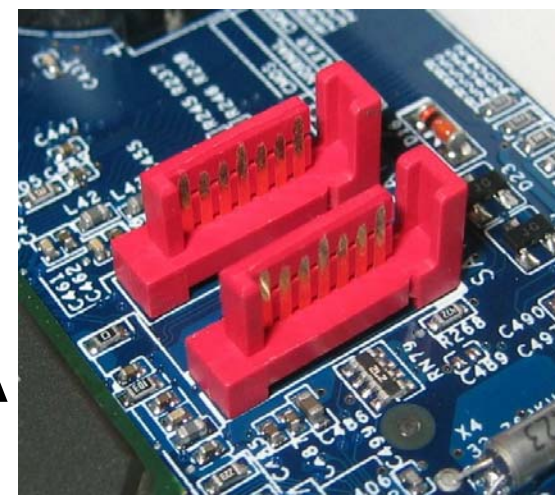
Festplatten am PC



- ATA Standard
 - AT-Attachment (IBM AT)
 - Verbindet Festplatten, CD-ROM, etc. mit Motherboard
 - Verschiedene Modi wurden entwickelt
- Datentransfer
 - PIO (16 MByte/sec)
 - Programmed Input Output
 - CPU trägt die volle Last, keine DMA Unterstützung
 - DMA (16 MByte/sec)
 - UDMA (133 MByte/sec)
 - Ultra Direct Memory Access

Festplatten am PC

- ATA heißt neuerdings Parallel-ATA
- Serial ATA
 - Serielle Übertragung der Bits statt parallel über einzelne Drähte
 - Höhere Transferrate
 - 150 MByte/sec bei SATA I
 - 300 MByte/sec bei SATA II
 - Neue Stecker & Kabel
- Limitierender Faktor ist die Mechanik
 - Platten schaffen < 100 MByte/sec
 - SATA I ist damit noch nicht ausgelastet



IO-Port Polling

- Die CPU will ein Kommando an den Controller schicken
 1. Busy-Bit im Statusregister muss gelöscht sein
 2. Setzen des Command und des Data-Out Registers
 3. Setzen des Command-Ready Bit im Control Register
 4. Controller erkennt Command-Ready Bit und setzt das Busy-Bit im Statusregister
 5. Controller liest Command und Data-Out Register und führt die Operation durch
 6. Controller löscht das Command-Ready und das Busy Bit
 7. Die CPU bemerkt, dass der Controller fertig ist (Busy Bit ist gelöscht) und liest das Data-In Register aus

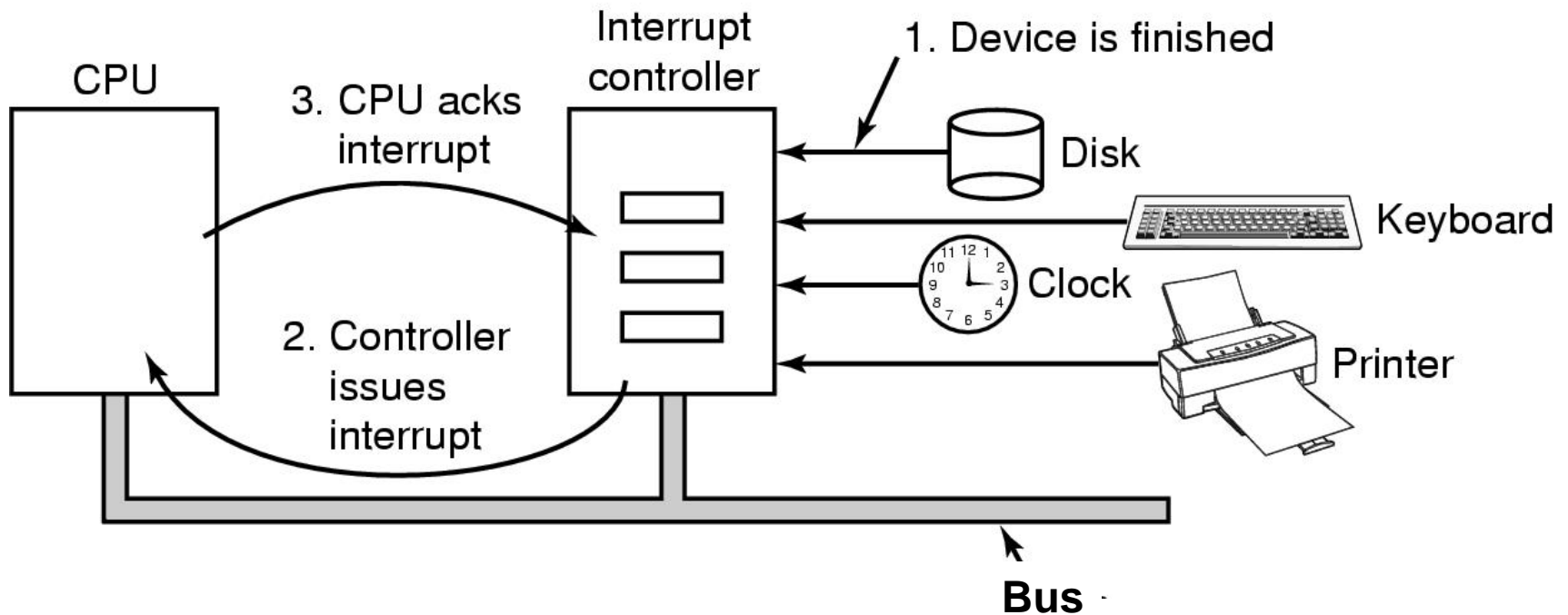
IO-Port Polling

- Das Polling geht sehr schnell
 - 3 Maschinenbefehle
 - Lesen des Registers
 - Logical-AND, um das Busy Bit zu extrahieren
 - Bedingtes Verzweigen
- Die CPU ist mit Warten beschäftigt
 - Sie könnte statt dessen einen anderen Prozess bearbeiten
- Lösung
 - Der Controller erzeugt einen Interrupt wenn er fertig ist
 - Dadurch entfällt das Polling

Interrupt Controller

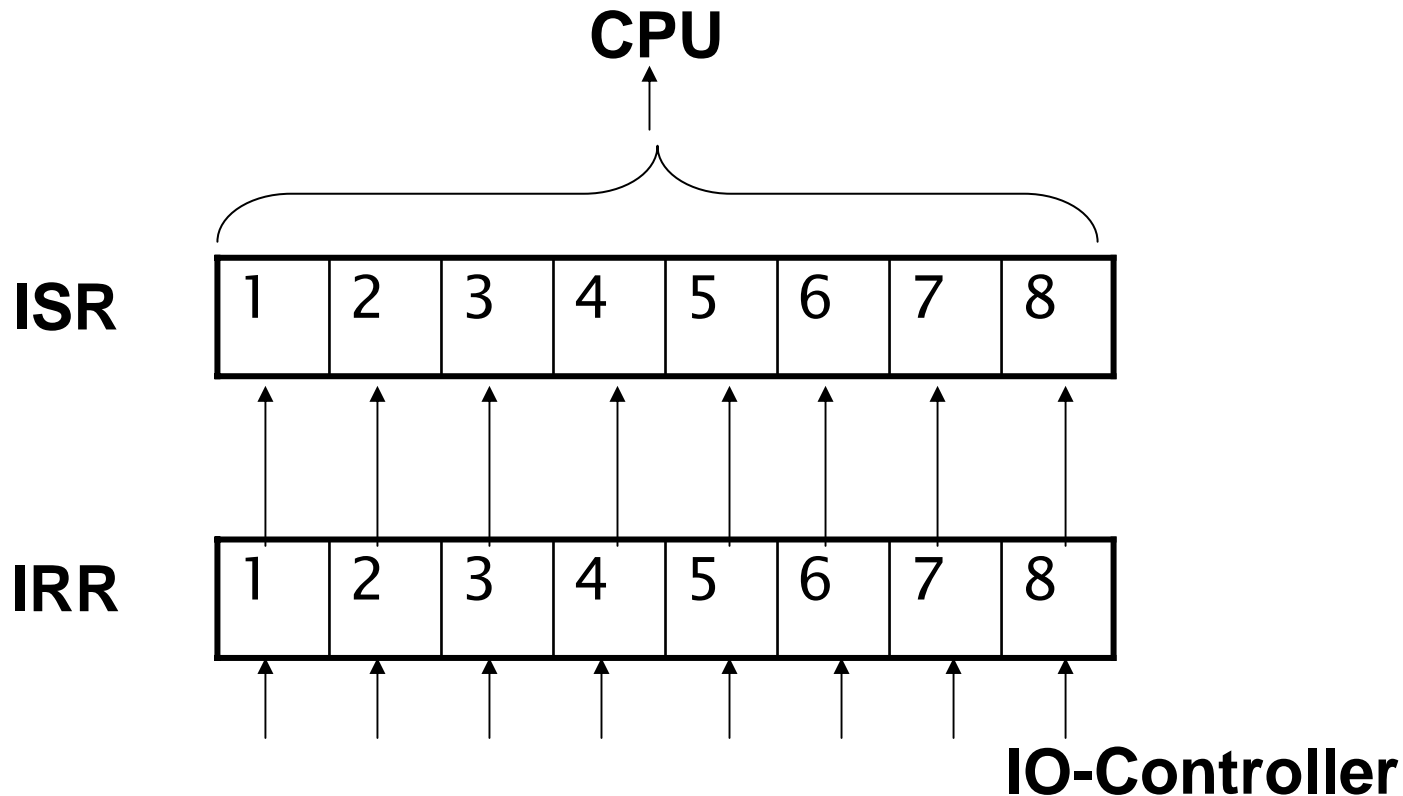
- Jeder PC hat einen PIC
 - Programmable Interrupt Controller
 - Kann 8 verschiedene Interrupts erzeugen
 - Moderne PCs haben einen APIC
- Interrupt Multiplexing
 - Mehrere IO-Controller können gleichzeitig einen Interrupt auslösen wollen
 - Der PIC entscheidet, welches Gerät Vorrang bekommt
 - Der PIC verwaltet eine „Warteschlang“ mit Geräten, die einen Interrupt auslösen wollen
- Kaskadierung ist möglich
 - D.h. ein Interrupt kann einen anderen unterbrechen

Interrupt Controller



Interrupt Controller

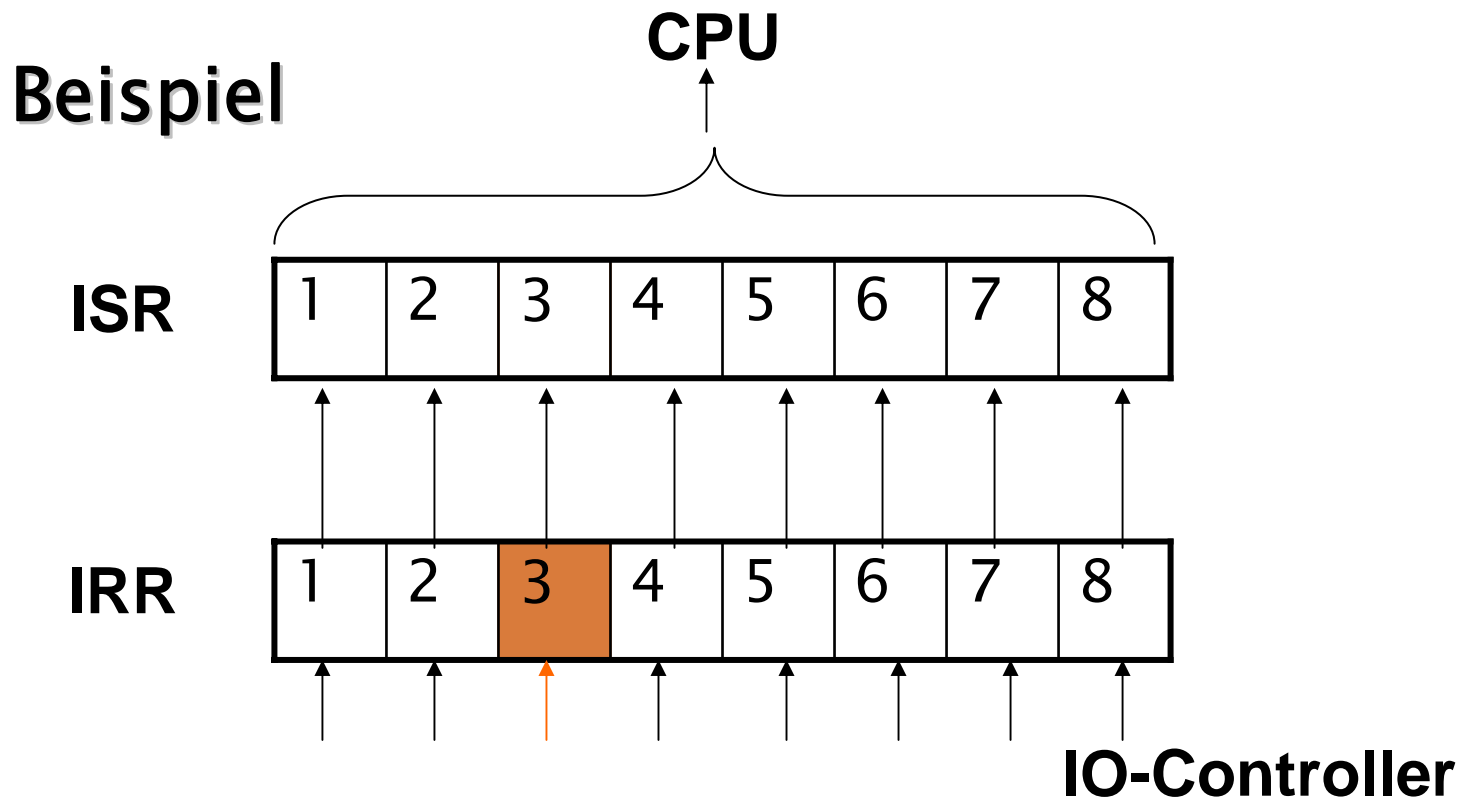
- Register eines PIC:
- Interrupt Request Register (IRR)
 - Ein Bit pro IO-Controller
 - Zeigt an, ob der IO-Controller einen Interrupt auslösen möchte
- Interrupt Service Register (ISR)
 - Ein Bit pro IO-Controller
 - Zeigt an, wessen Interrupt gerade ausgeführt wird, bzw wessen Interrupt in der Ausführung unterbrochen wurde
- Interrupt Mask Register (IMR)
 - Schaltet einzelne Interrupts aus



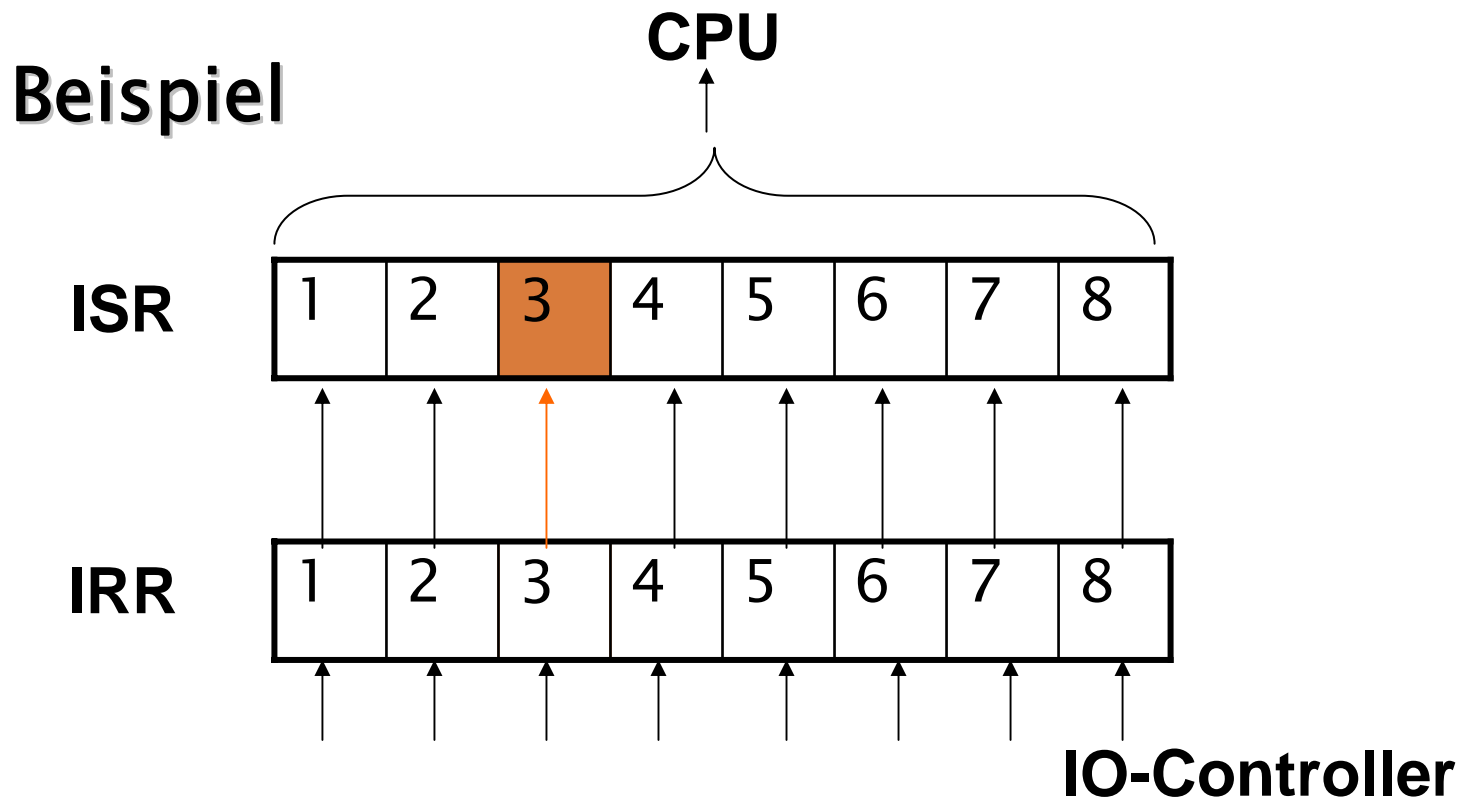
- Je höher die Nummer des Interrupts, desto höher ist seine Priorität

Auslösen eines Interrupts

1. Der IO-Controller signalisiert dem PIC, dass er einen Interrupt wünscht
2. Das Bit im IRR wird gesetzt
3. Wenn zZ kein Interrupt ausgeführt wird ($IRS=0$), dann Setzen des Bit im IRS, Löschen des Bit im IRR
4. Der Interrupt wird an die CPU weitergeleitet und die Nummer des Interrupts auf den Bus gelegt
5. Die CPU holt die Interrupt Nummer vom Bus und schaut im Interrupt-Vektor nach
6. Die CPU ist fertig. Das Bit im IRS wird gelöscht
7. Wenn im IRS noch Bits gesetzt sind, dann geht es mit Schritt 4 weiter

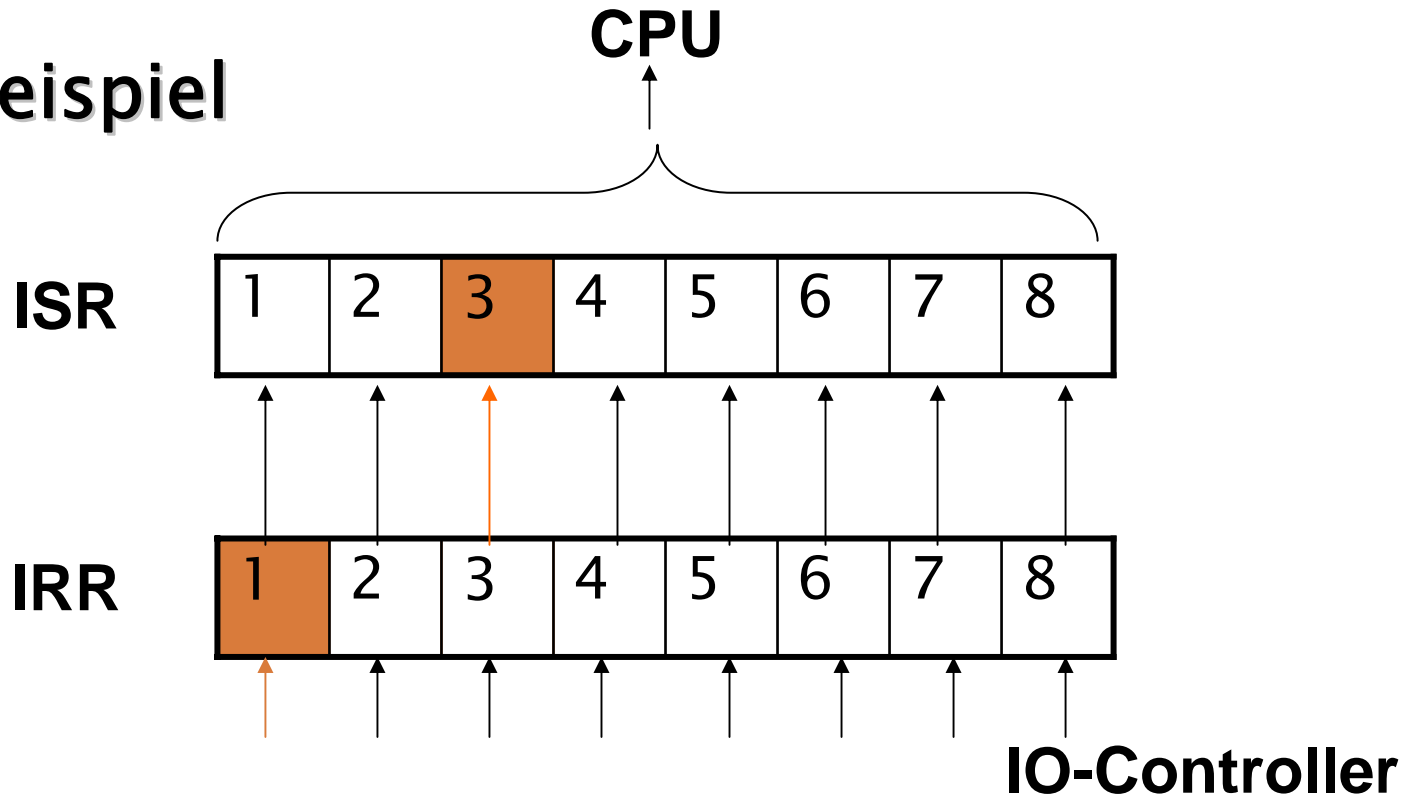


- Gerät an Leitung 3 will einen Interrupt auslösen
- Das Bit im IRR wird gesetzt

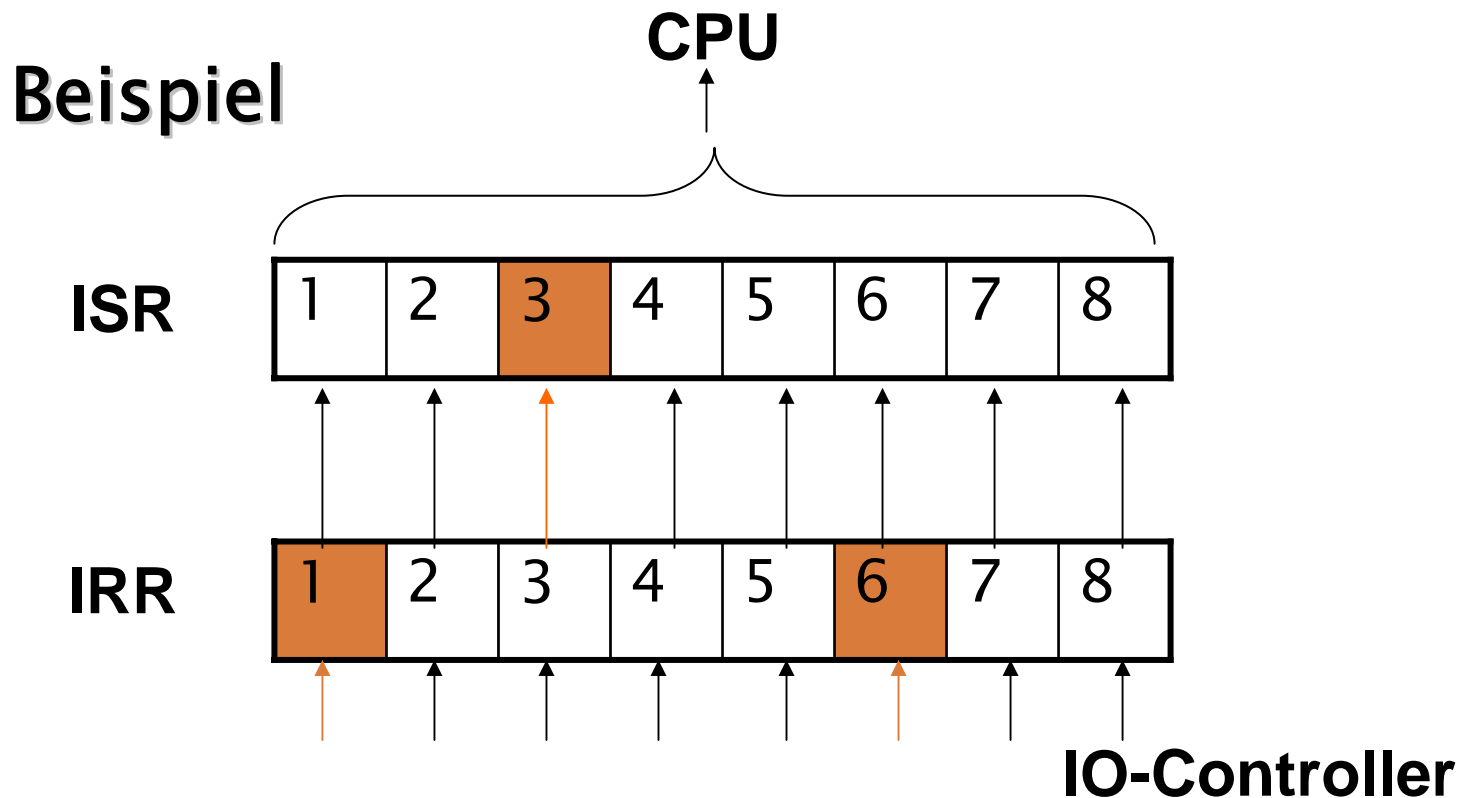


- Kein anderer Interrupt wird z.Z. ausgeführt
- Bit im IRR löschen und im ISR setzen
- Die CPU wird den Interrupt-Handler ausführen

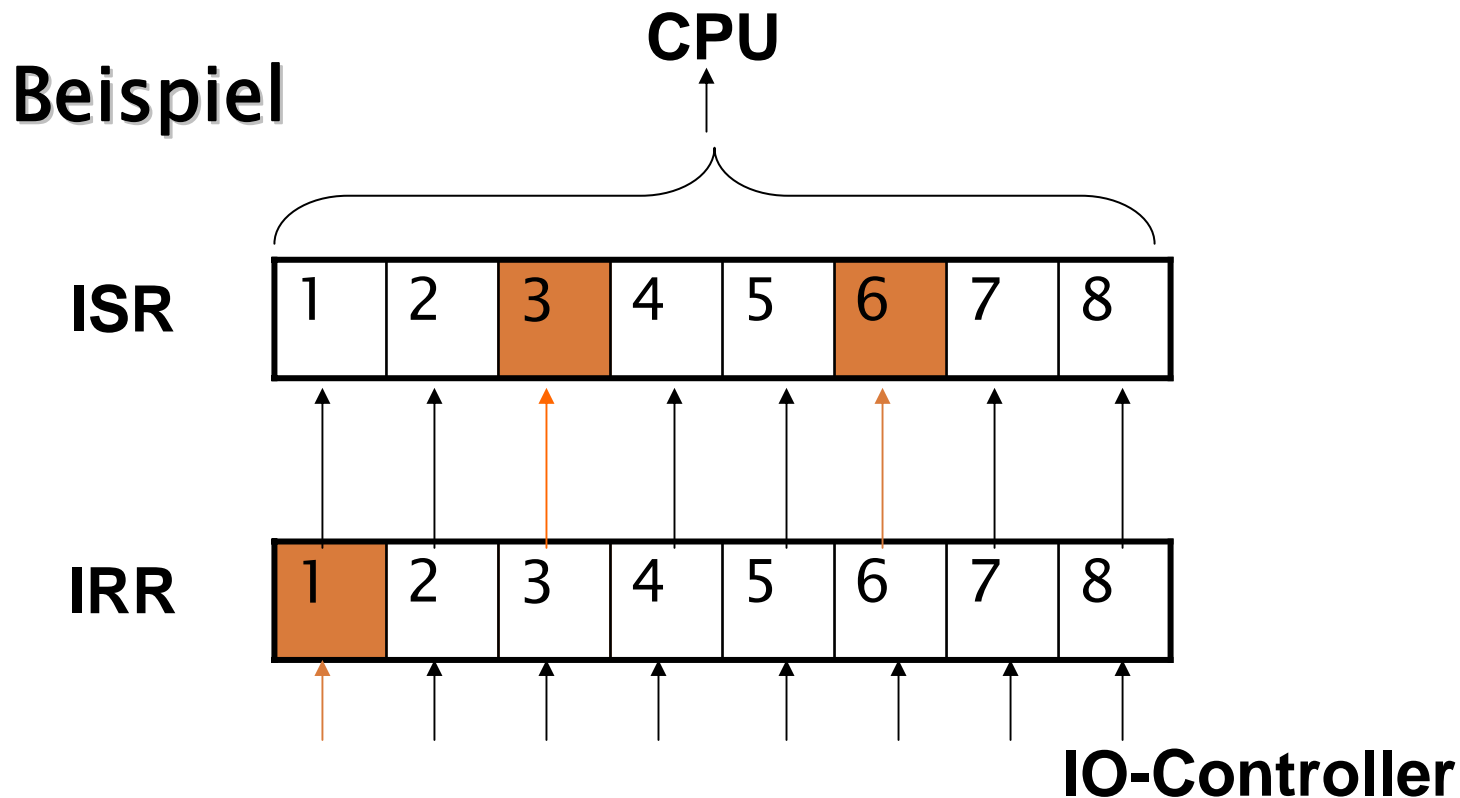
Beispiel



- Gerät 1 will einen Interrupt auslösen
- Seine Priorität langt aber nicht, um 3 zu unterbrechen
- Die Warteschlange ist damit { 1 }

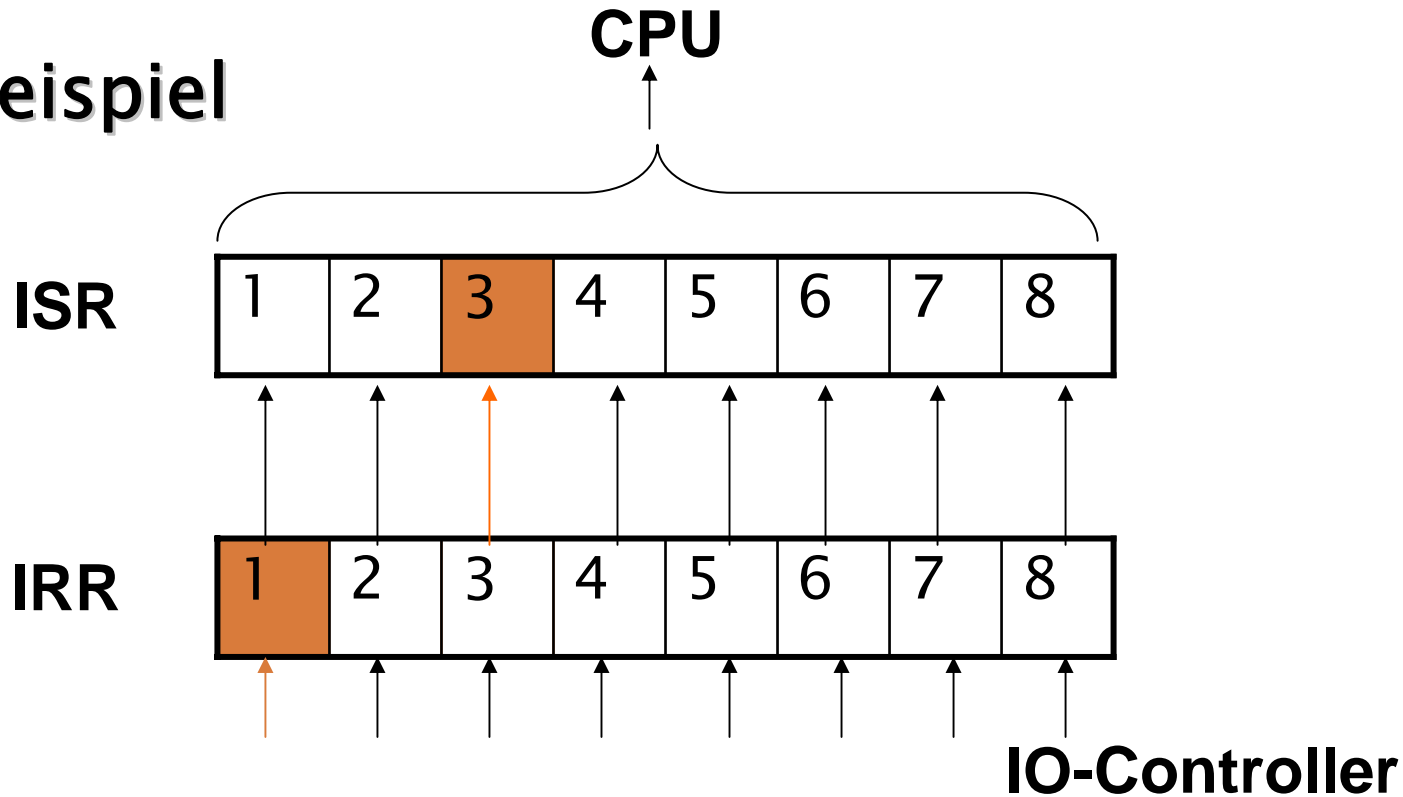


- Gerät 6 möchte einen Interrupt auslösen
- Zuerst wird das Bit im IRR gesetzt



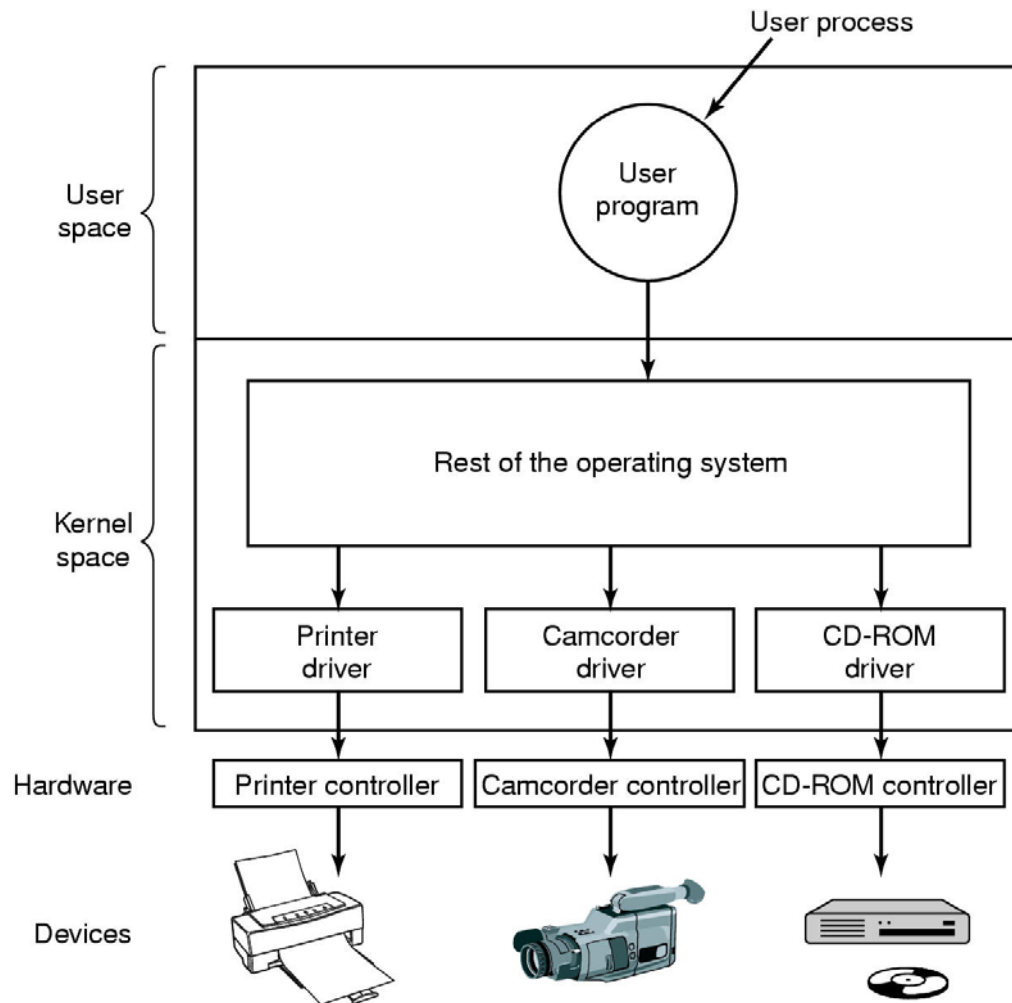
- Gerät 6 darf den Handler von Gerät 3 unterbrechen
- Die Warteschlange ist damit { 3, 1 }

Beispiel

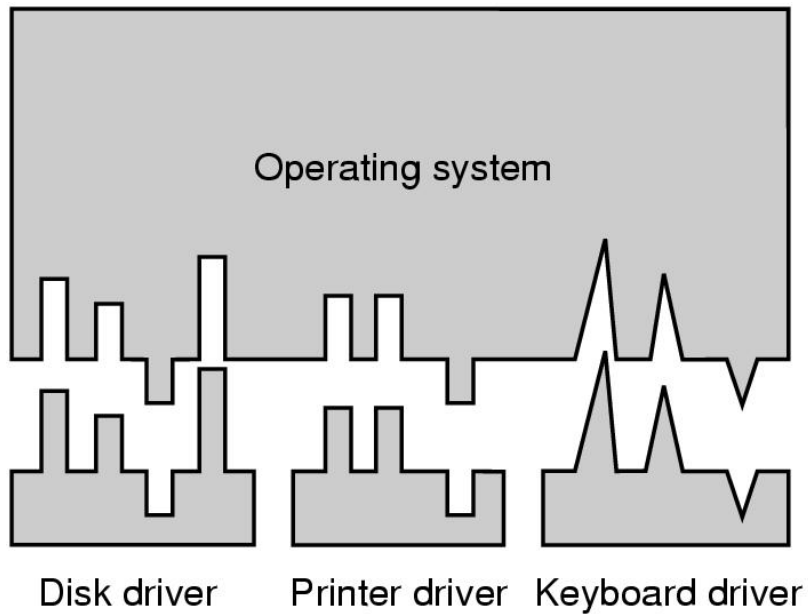


- Der Handler von Gerät 6 ist fertig
- Das Bit im ISR wird gelöscht
- Das nächste Gerät aus der Warteschlange darf einen Interrupt auslösen

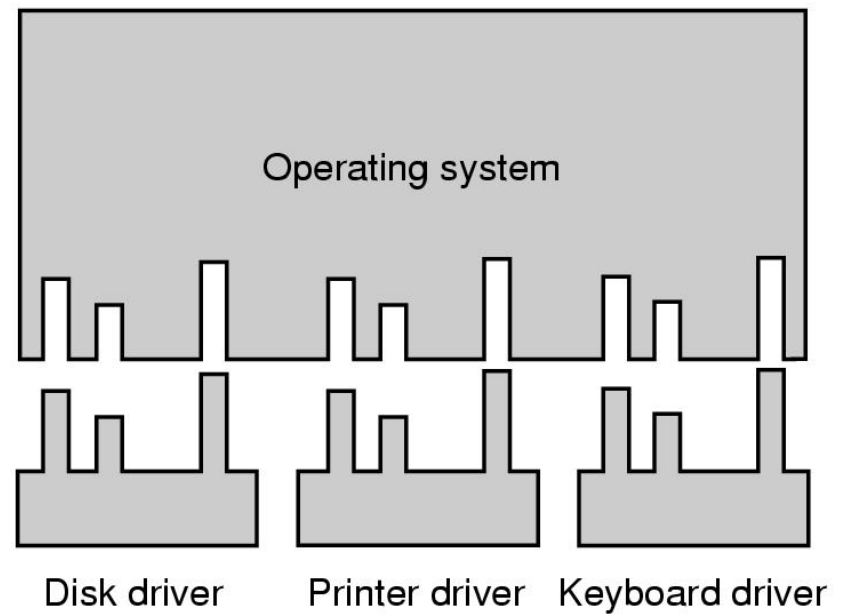
Gerätetreiber



Gerätetreiber Schnittstelle



(a)

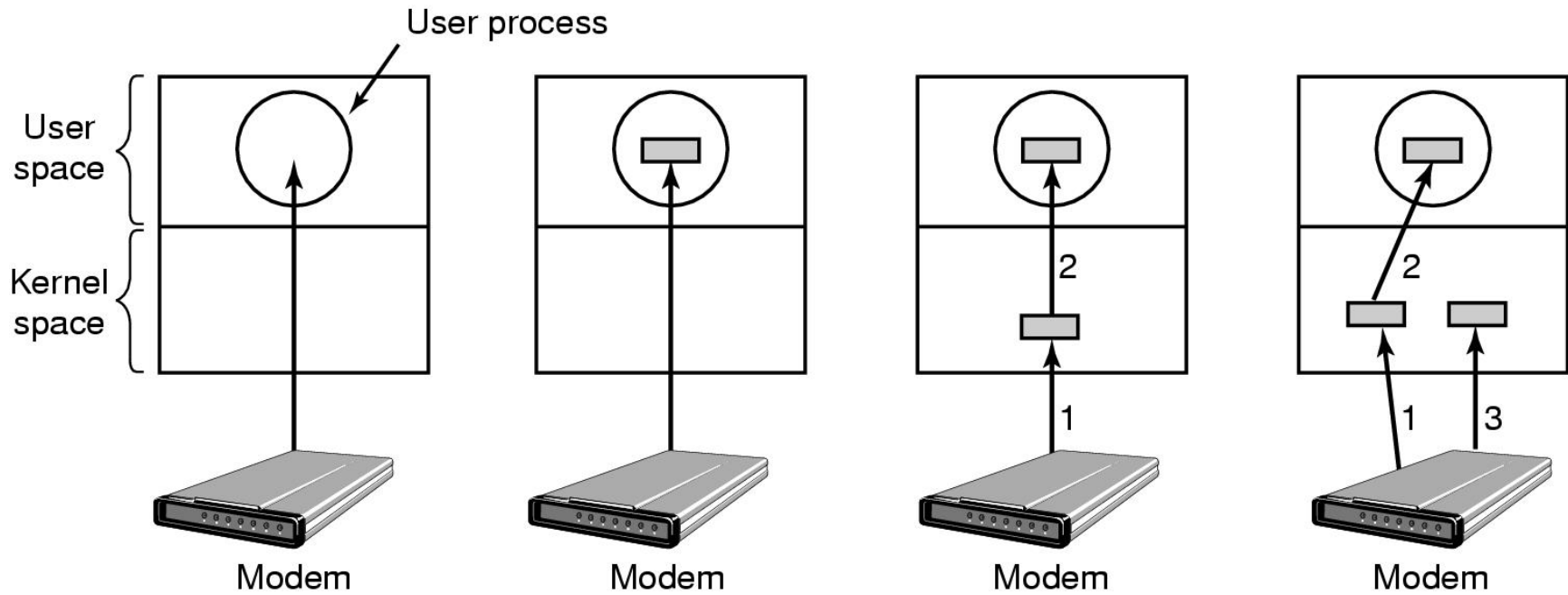


(b)

Geräteunabhängige IO

- Uniforme Schnittstelle für Gerätetreiber
 - Das BS muss nicht a priori alle Gerätetypen kennen
 - Neue Gerätetypen benötigen keine BS Änderung
 - ... leider nur theoretisch
- Pufferung
 - Entkoppelt Gerät und Konsument (d.h. Applikation)
- Fehlerreport
 - Applikation hat einheitliche Schnittstelle
 - UNIX: `errno` Variable
- Anfordern und Freigeben von Geräten
 - Ressourcenverwaltung
- Symbolische Namen für Geräte
 - UNIX: `/dev/mouse`

Geräteunabhängige IO: Pufferung



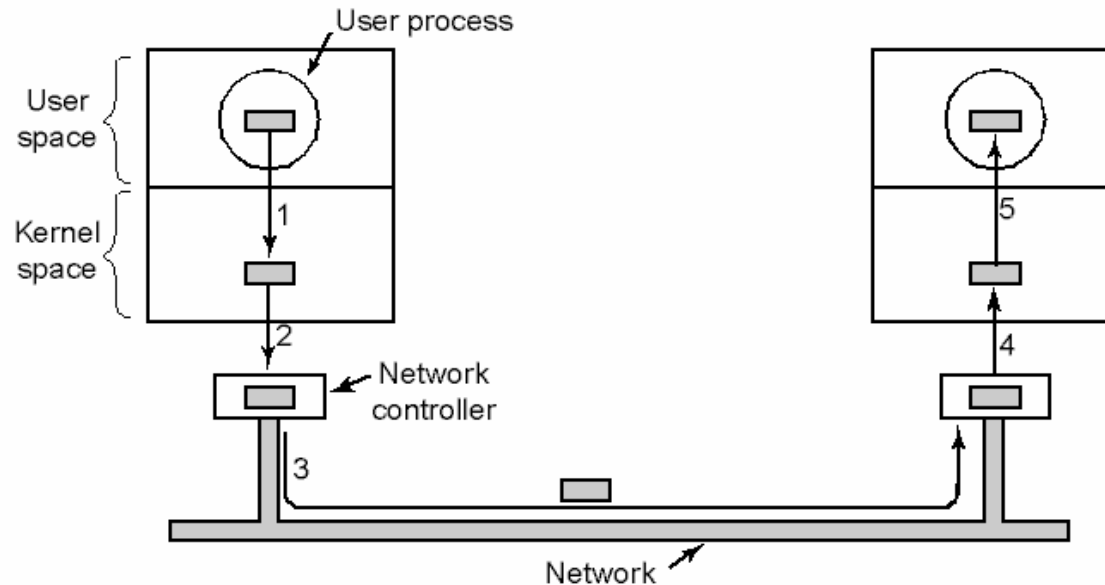
- (a) Ungepufferte Eingabe
- (b) Pufferung im Benutzeradressraum
- (c) Pufferung im Kern und Benutzeradressraum
- (d) Doppelte Pufferung im Kern

Geräteunabhängige IO: Pufferung

- Pufferung auch bei Ausgabe relevant
- Beispiel
 - Prozess allokiert Puffer der Größe 1KB
 - Prozess ruft `write(puffer)` aus
 - Die Funktion kehrt sofort zurück
 - Der Prozess arbeitet weiter
 - Das BS wartet, bis das Gerät bereit ist
 - Das BS schickt Daten aus dem Puffer an das Gerät
- Problem
 - Woher weiß der Prozess, wann das BS fertig ist?
 - Wann gibt der Prozess den Puffer frei?
 - Was passiert, wenn der Prozess den Puffer voreilig überschreibt?
 - Was passiert, wenn der Puffer ausgelagert wird?

Geräteunabhängige IO: Pufferung

- Zuviel Pufferung ist auch nicht gut
 - Die Latenz steigt
- Häufiges Umkopieren kostet Zeit
 - Beispiel: Netzwerkstack



Geräteunabhängige IO: ioctl

- Geräte können wie Dateien behandelt werden
 - open, read, write, seek, close
- Geräte haben aber oft Spezialeigenschaften, die man ansprechen möchte
- Spezielle BS API für jedes Gerät wäre overkill
- Ein Befehl langt: ioctl (IO Control)
 - Er schleust Daten von der Applikation direkt zum Gerätetreiber durch
 - Das BS interessiert sich nicht für die Daten, es reicht sie nur weiter

Geräteunabhängige Blockgröße

- Dateisysteme nehmen einheitliche Blockgröße
 - Unterschiedliche Festplatten haben andere physikalische Blockgrößen
 - Das BS abstrahiert von der physikalischen Blockgröße
- Blockorientierte Geräte wie zeichenorientierte benutzen
 - Eine Datei kann man Byte für Byte auslesen, obwohl sie blockweise gespeichert ist
 - Das BS liest einen Block in den Kern-Puffer und bedient daraus die Anwendung nach Bedarf

IO Layer

