

# Betriebssysteme (BS)

## VL 3 – Unterbrechungen, Hardware

**Volkmar Sieh / Daniel Lohmann**

Lehrstuhl für Informatik 4  
Verteilte Systeme und Betriebssysteme

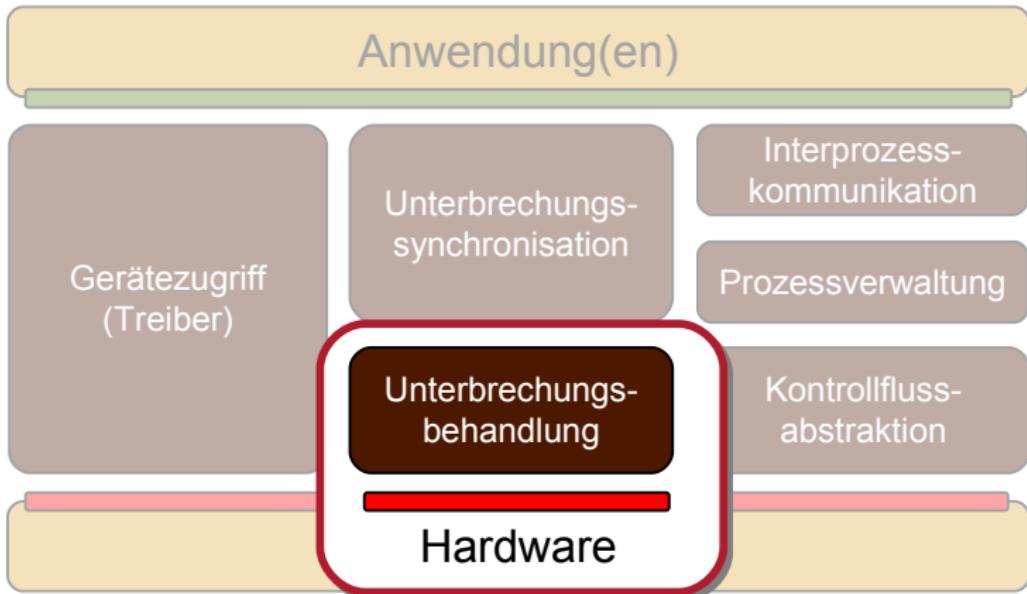
Friedrich-Alexander-Universität  
Erlangen Nürnberg

WS 18 – 8. November 2018

[https://www4.cs.fau.de/Lehre/WS18/V\\_BS](https://www4.cs.fau.de/Lehre/WS18/V_BS)



# Überblick: Einordnung dieser VL



# Agenda

---

Einordnung

Grundlegende Fragestellungen

Hardware-Architekturen

Zusammenfassung und Ausblick



## Einordnung

Grundlegende Fragestellungen

Hardware-Architekturen

Zusammenfassung und Ausblick



Ein Blick zurück in die Historie von Betriebssystemen...

- Überlappte Ein-/Ausgabe:

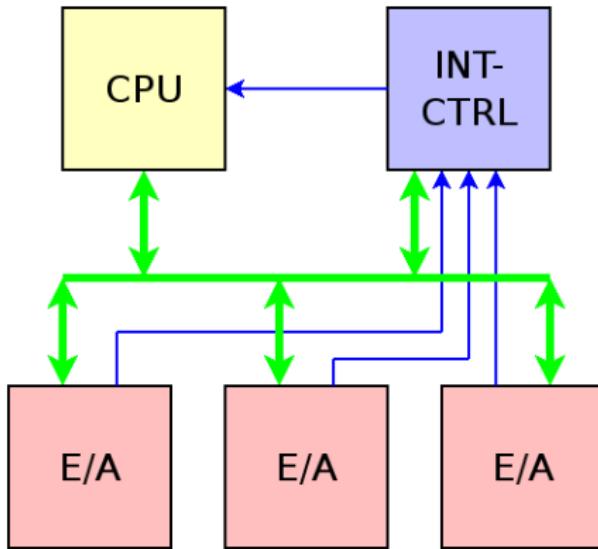
- Eingaben: Verschwendung von anderweitig nutzbaren Prozessorzyklen bei (oft nicht vorhersagbar langem) *aktivem Warten*
- Ausgaben: selbständiges Agieren der E/A-Geräte (z.B. durch *DMA*) entlastet die CPU

- Timesharing Betrieb:

- Zeitgeber-Unterbrechungen geben dem Betriebssystem die Möglichkeit
  - zur *Verdrängung von Prozessen*
  - Aktivitäten zeitgesteuert zu starten



# Interrupt-Hardware



Interrupt-Controller ist im einfachsten Fall ein „Oder“-Gatter...



Einordnung

## Grundlegende Fragestellungen

Priorisierung

Verlust von IRQs

Behandlungsroutine

Zustandssicherung

Multiprozessorsysteme

Gefahren

Hardware-Architekturen

Zusammenfassung und Ausblick



## ■ Problem:

- Mehrere Unterbrechungsanforderungen können gleichzeitig signalisiert werden. Welche ist wichtiger?
- Während die CPU auf die wichtigste Anforderung reagiert, können weitere Anforderungen signalisiert werden.



- **Problem:**
  - Mehrere Unterbrechungsanforderungen können gleichzeitig signalisiert werden. Welche ist wichtiger?
  - Während die CPU auf die wichtigste Anforderung reagiert, können weitere Anforderungen signalisiert werden.
- **Lösung:** ein **Priorisierungsmechanismus** ...
  - **in Software:** die CPU hat nur einen IRQ (*interrupt request*) Eingang und fragt die Geräte in bestimmter Reihenfolge ab
  - **in Hardware:** eine Priorisierungsschaltung ordnet Geräten eine Priorität zu und leitet immer nur die dringendste Anforderung zur Behandlung weiter
  - **mit festen Prioritäten:** jedem Gerät wird statisch eine Priorität zugeordnet
  - **mit variablen Prioritäten:** Prioritäten sind dynamisch änderbar oder wechseln zum Beispiel zyklisch



## ■ Problem:

- während der Behandlung oder Sperrung von Unterbrechungen, kann die CPU keine neuen Unterbrechungen behandeln
- die Speicherkapazität für Unterbrechungsanforderungen ist endlich.
  - i.d.R. ein Bit pro Unterbrechungseingang



## ■ Problem:

- während der Behandlung oder Sperrung von Unterbrechungen, kann die CPU keine neuen Unterbrechungen behandeln
- die Speicherkapazität für Unterbrechungsanforderungen ist endlich.
  - i.d.R. ein Bit pro Unterbrechungseingang

## ■ Lösung: in Software

- die Unterbrechungsbehandlungsroutine sollte möglichst kurz sein (zeitlich!), um die Wahrscheinlichkeit von Verlusten zu minimieren
- Unterbrechungen sollten nicht unnötig lange gesperrt werden
- jeder Gerätetreiber sollte davon ausgehen, dass **eine** Unterbrechung **mehr als eine** abgeschlossene E/A Operation anzeigen kann



# Zuordnung einer Behandlungsroutine

## ■ Problem:

- die Software soll mit möglichst wenig Aufwand herausfinden können, welches Gerät die Unterbrechung ausgelöst hat
  - eine sequentielle Abfrage der Geräte kostet nicht nur Zeit, sondern verändert die Zustände von E/A Bussen und unbeteiligten Geräten



# Zuordnung einer Behandlungsroutine

## ■ Problem:

- die Software soll mit möglichst wenig Aufwand herausfinden können, welches Gerät die Unterbrechung ausgelöst hat
  - eine sequentielle Abfrage der Geräte kostet nicht nur Zeit, sondern verändert die Zustände von E/A Bussen und unbeteiligten Geräten

## ■ Lösung:

- jeder Unterbrechung wird eine Nummer zugeordnet, die als Index in eine Vektortabelle verwendet wird
  - die Vektornummer hat nicht zwangsläufig etwas mit der Priorität zu tun
  - es kommt in der Praxis leider vor, dass Geräte sich eine Vektornummer teilen müssen (*interrupt sharing*)
- der Aufbau der Vektortabelle variiert je nach Prozessortyp
  - meist enthält sie Zeiger auf Funktionen
  - seltener sind die Einträge selbst bereits Instruktionen



# Zustandssicherung

## ■ Problem:

- nach der Ausführung der Behandlungsroutine soll zum normalen Kontext zurückgekehrt werden können
- die Behandlung soll quasi unbemerkt ablaufen (*transparency*)



- **Problem:**
  - nach der Ausführung der Behandlungsroutine soll zum normalen Kontext zurückgekehrt werden können
  - die Behandlung soll quasi unbemerkt ablaufen (*transparency*)
- **Lösung:**
  - Zustandssicherung durch Hardware
    - nur das Notwendigste: z.B. Rücksprungadresse u. Prozessorstatuswort
    - Wiederherstellung durch speziellen Befehl, z.B. iret, rte, ...
  - Zustandssicherung durch Software
    - da Unterbrechungen jederzeit auftreten können, muss auch die Behandlungsroutine Zustände sichern und wiederherstellen



# Geschachtelte Behandlung

## ■ Problem:

- um auf sehr wichtige Ereignisse schnell reagieren zu können, soll auch eine Unterbrechungsbehandlung unterbrechbar sein
- eine unbegrenzte Schachtelungstiefe muss aber vermieden werden



# Geschachtelte Behandlung

## ■ Problem:

- um auf sehr wichtige Ereignisse schnell reagieren zu können, soll auch eine Unterbrechungsbehandlung unterbrechbar sein
- eine unbegrenzte Schachtelungstiefe muss aber vermieden werden

## ■ Lösung:

- die CPU erlaubt immer nur Unterbrechungen mit höherer Priorität
- die aktuelle Priorität wird im Prozessorstatuswort gespeichert
- die vorherige Priorität wird auf einem Stapel abgelegt



# Interrupt-Ablauf

---

- HW: E/A-Gerät signalisiert Interrupt an Interrupt-Controller
- HW: Interrupt-Controller signalisiert Interrupt an CPU
- HW: CPU sichert einige Register (PC, IE, ...)
- HW: CPU disabled Interrupt-Eingang (IE = 0)
- HW: CPU holt sich von Interrupt-Controller IRQ-Nummer
- HW: CPU lädt Vektor aus Interrupt-Vektor-Tabelle in PC
- SW: Interrupt-Handler sichert weitere Register
- SW: Interrupt-Handler behandelt Interrupt
- SW: Interrupt-Handler restauriert Register
- SW: Interrupt-Handler ruft `iret` auf
- HW: CPU restauriert PC, IE, ...  
Rücksprung in das unterbrochene Haupt-Programm,  
Interrupt-Eingang wieder enabled

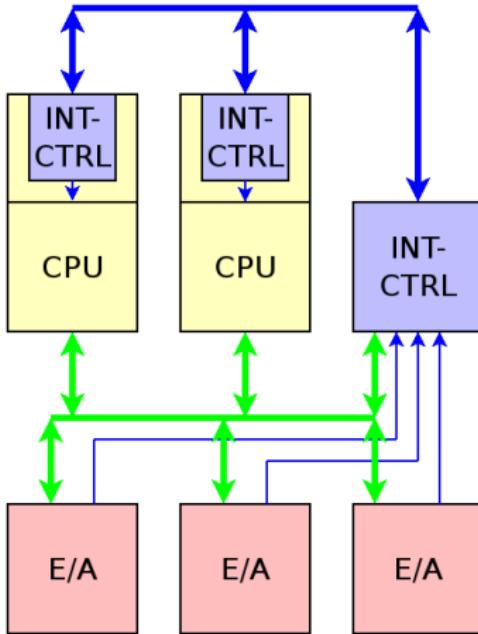


- **Problem:**
  - Unterbrechungen können immer nur von einer CPU behandelt werden. Aber welche?
  - es gibt eine weitere Kategorie von Unterbrechungen: die Interprozessor-Unterbrechungen
- **Lösung:** die Hardware zur Unterbrechungsbehandlung auf Multiprozessorsystemen muss komplexer ausgelegt sein. Es gibt viele Entwurfsvarianten ...
  - feste Zuordnung
  - zufällige Zuordnung
  - programmierbare Zuordnung
  - Zuordnung unter Berücksichtigung der Prozessorlast

... und Kombinationen davon.



# Interrupt-Hardware (SMP)



Interrupt-Controller sind selbständige, konfigurierbare Einheiten...



# Gefahr: „unechte Unterbrechungen“

(„*spurious interrupts*“)

- **Problem:** ein technischer Mechanismus zur Unterbrechungsbehandlung birgt die Gefahr von fehlerhaften Unterbrechungsanforderungen, z.B. durch ...
  - Hardwarefehler
  - fehlerhaft programmierte Geräte
- **Lösung:**
  - Hardware- und Softwarefehler vermeiden 
  - Betriebssystem „defensiv“ programmieren
    - mit unechten Unterbrechungen rechnen



# Gefahr: „Unterbrechungsstürme“

(„*interrupt storms*“)

## ■ Problem:

- hochfrequente Unterbrechungsanforderungen können einen Rechner lahm legen
- es handelt sich entweder um unechte Unterbrechungen oder der Rechner ist mit der E/A Last überfordert
- kann leicht mit Seitenflattern (*thrashing*) verwechselt werden

## ■ Lösung: durch das Betriebssystem

- Unterbrechungsstürme erkennen
- das verursachende Gerät deaktivieren



# Agenda

---

Einordnung

Grundlegende Fragestellungen

**Hardware-Architekturen**

Motorola/Freescale 68k

Intel x86

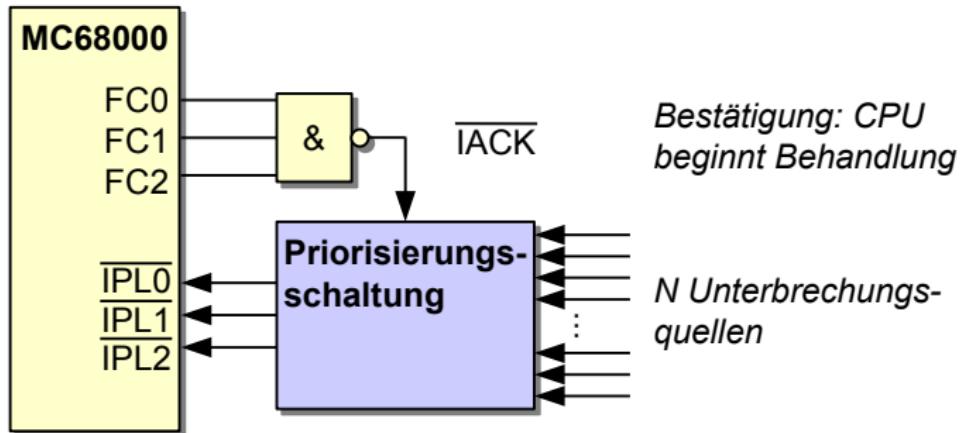
Zusammenfassung und Ausblick



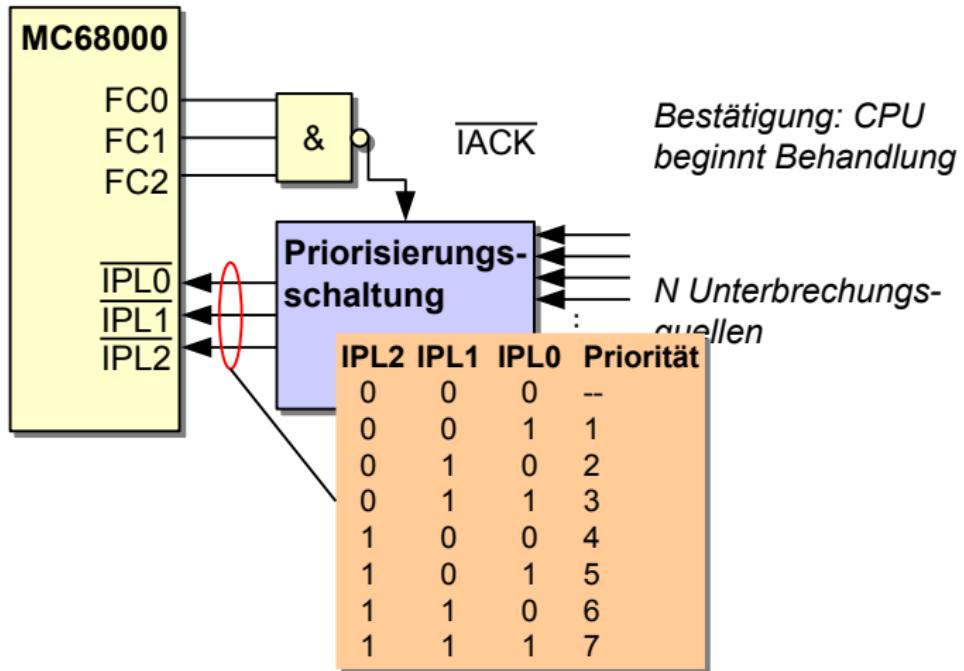
# Unterbrechungen beim MC68000



# Unterbrechungen beim MC68000

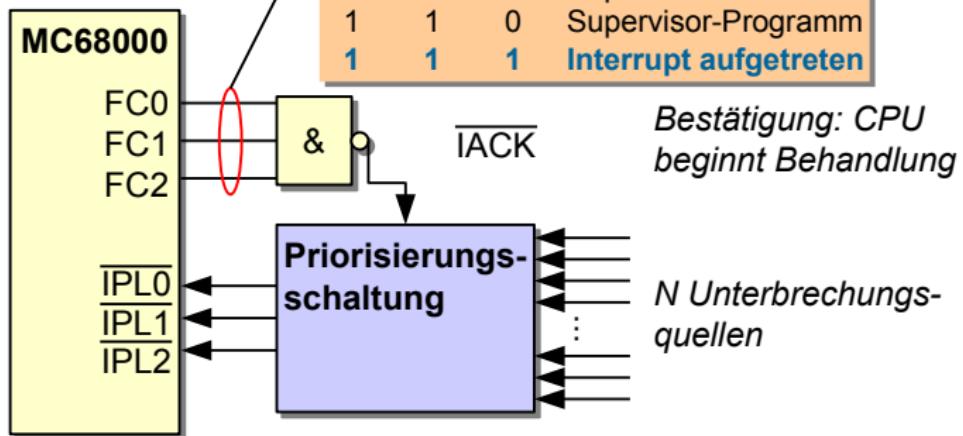


# Unterbrechungen beim MC68000



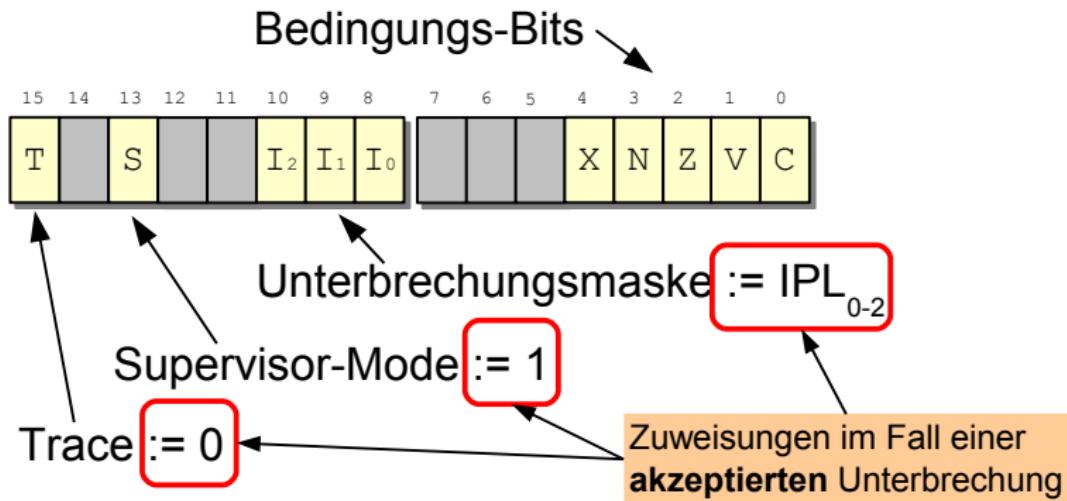
# Unterbrechungen

| FC2      | FC1      | FC0      | Zyklustyp                    |
|----------|----------|----------|------------------------------|
| 0        | 0        | 0        | reserviert                   |
| 0        | 0        | 1        | Anwender-Daten               |
| 0        | 1        | 0        | Anwender-Programm            |
| 0        | 1        | 1        | reserviert                   |
| 1        | 0        | 0        | reserviert                   |
| 1        | 0        | 1        | Supervisor-Daten             |
| 1        | 1        | 0        | Supervisor-Programm          |
| <b>1</b> | <b>1</b> | <b>1</b> | <b>Interrupt aufgetreten</b> |



# Das Statusregister (SR) des MC68000

- enthält u.A. die aktuelle Unterbrechungsmaske
  - bei einer Unterbrechung wird geprüft, ob  $IPL_{0-2} > I_{0-2}$  ist. Wenn nein, wird der Anforderung (noch) nicht stattgegeben.
  - eine Unterbrechung mit  $IPL_{0-2} = 7$  wird aber immer bearbeitet (NMI)

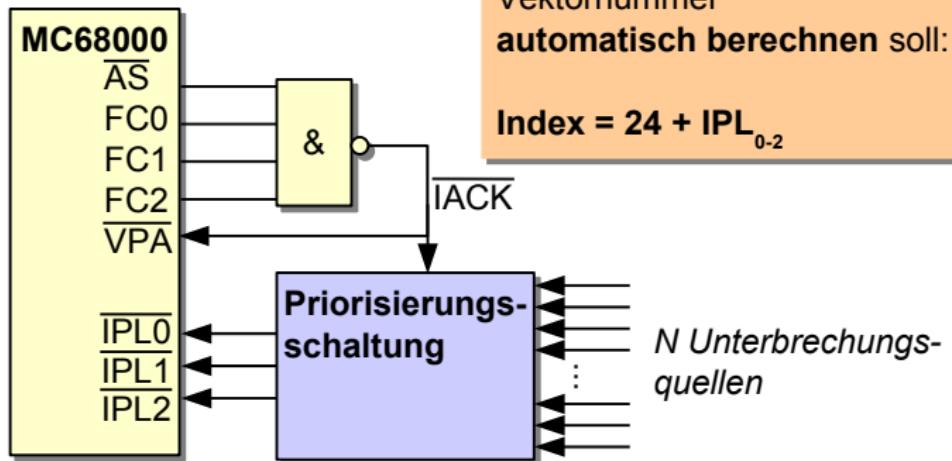


# Vektortabelle des MC68000

| Index  | Adresse | Bedeutung   |
|--------|---------|---|
| 0      | 0x000   | Reset: Supervisor-Stapelzeiger                      |
| 1      | 0x004   | Reset: PC   |
| 2      | 0x008   | Busfehler   |
| 3      | 0x00c   | Adressfehler  |
| 4      | 0x010   | Illegaler Befehl                                    |
| 5      | 0x014   | Division durch Null                                 |
| ...    |         |   |
| 24     | 0x060   | unechte Unterbrechung                               |
| 25     | 0x064   | <b>autovektorielle Unterbrechung, Ebene 1</b>       |
| 26     | 0x068   | <b>autovektorielle Unterbrechung, Ebene 2</b>       |
| ...    |         |   |
| 30     | 0x078   | <b>autovektorielle Unterbrechung, Ebene 6</b>       |
| 31     | 0x07c   | <b>autovektorielle Unterbrechung, Ebene 7 (NMI)</b> |
| 32-47  | 0x080   | TRAP-Befehlsvektoren                                |
| 48-63  | 0x0c0   | reserviert  |
| 64-255 | 0x100   | <b>Anwender-Unterbrechungsvektoren</b>              |



# Autovektorielle Unterbrechungen



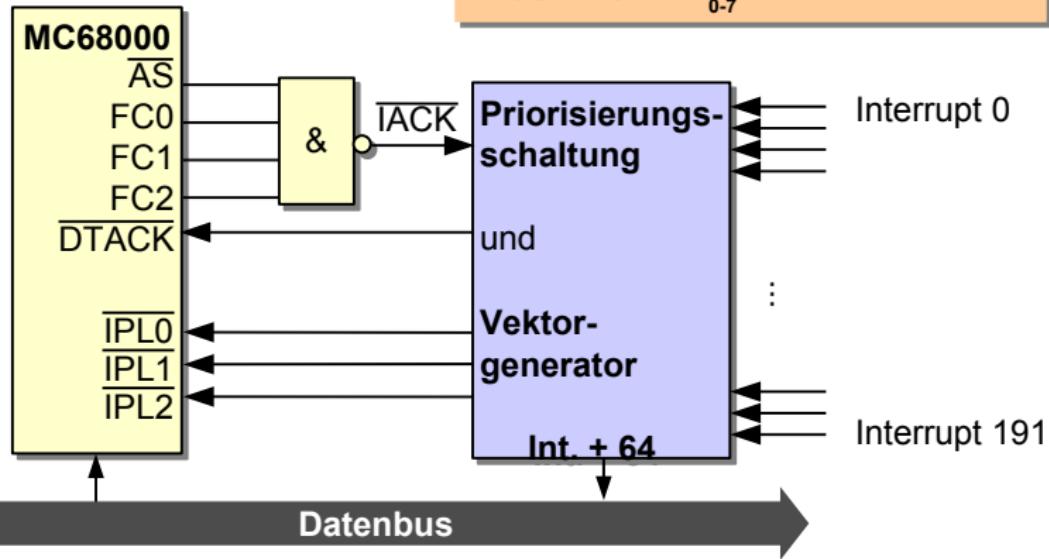
Problem: Es stehen nur 6 Vektoren für Geräte bereit.  
Bei mehr Geräten ist „sharing“ nicht zu vermeiden.



# Nicht-autovektorielle Unterbrechungen

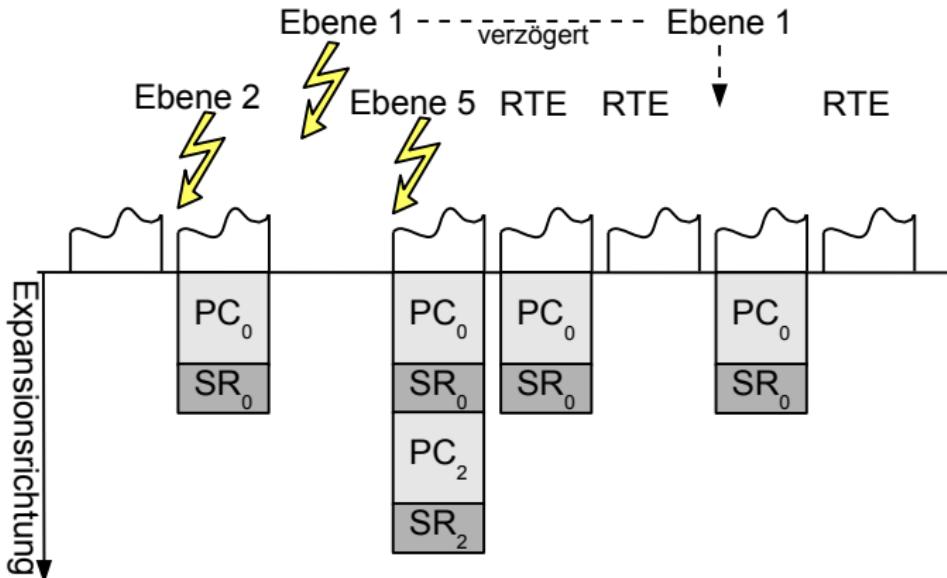
über **DTACK** signalisiert die externe Schaltung, dass die CPU die Vektornummer **über den Datenbus lesen** soll.

$$\text{Index} = 64 + D_{0-7}$$



# Zustandssicherung beim MC68000

- der vorherige SR Inhalt und der PC werden bei einer Unterbrechung auf dem Supervisor-Stapel gesichert
- der RTE Befehl macht den Vorgang rückgängig



# MC68000 - Zusammenfassung

- 6 Prioritätsebenen für Hardware-Unterbrechungen + NMI
  - Unterbrechungsebene 1-6, NMI Ebene 7
  - „Maskierung“ über  $I_{0-2}$  im Statusregister möglich
- nur Unterbrechungen höherer Priorität und der NMI können eine laufende Behandlung unterbrechen
  - Statusregister wird automatisch angepasst
- automatische Zustandssicherung auf dem *Supervisor*-Stapel, geschachtelte Behandlung möglich.
- die Vektornummernerzeugung erfolgt entweder ...
  - autovektoriell: Index = Priorität + 24
  - nicht-autovektoriell (durch externe Hardware): Index = 64 ... 255
- keine Multiprozessorunterstützung



# Unterbrechungen bei x86 CPUs



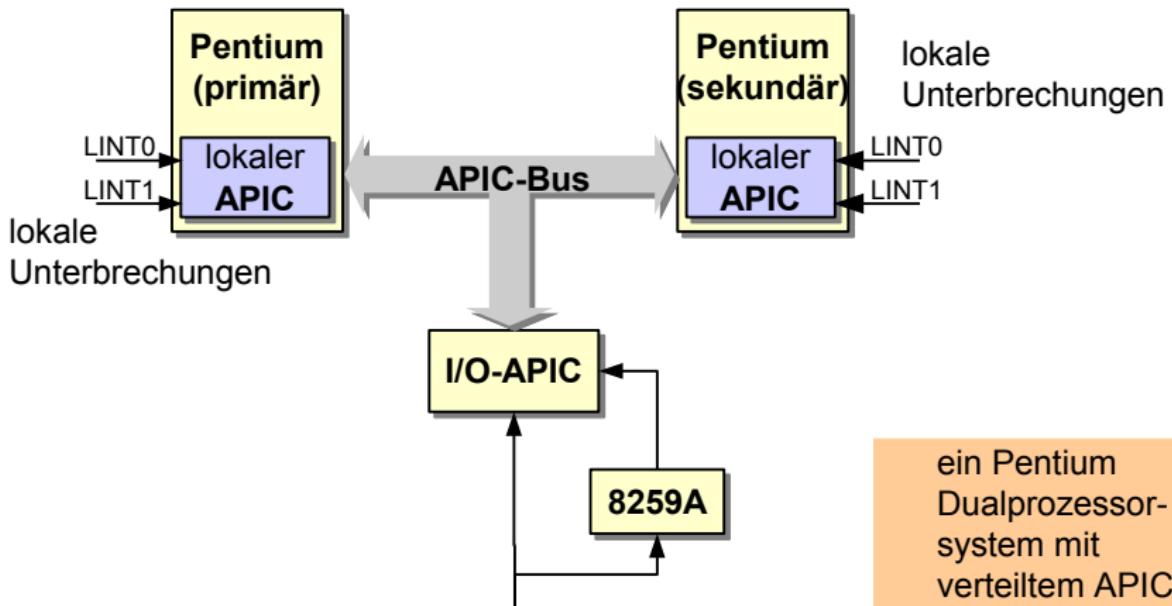
# Unterbrechungen bei x86 CPUs

- bis einschließlich i486 hatten x86 CPUs nur einen IRQ und einen NMI Eingang
- externe Hardware sorgte für die Priorisierung und Vektornummerngenerierung
  - durch einen Chip namens **PIC 8259A**
    - 8 Interrupt-Eingänge
    - 15 Eingänge bei Kaskadierung von zwei PICs
    - keine Multiprozessorunterstützung
- heutige x86 CPUs enthalten den weit leistungsfähigeren „**Advanced Programmable Interrupt Controller**“ (**APIC**)
  - notwendig für **Multiprozessorsysteme**
  - inzwischen aber auch in allen Einprozessorsystemen aktiv
    - natürlich gibt es den PIC 8259A noch immer 😊



# Die APIC Architektur

- ein APIC *Interrupt*-System besteht aus lokalen APICs auf jeder CPU und einem I/O APIC



Unterbrechungsanforderungen



# Der I/O APIC

- heute typischerweise in der *Southbridge* von PC Chipsätzen integriert
- normalerweise 24 *Interrupt*-Eingänge
  - zyklische Abfrage (Round-Robin Priorisierung)
- für jeden Eingang gibt es einen 64 Bit Eintrag in der ***Interrupt Redirection Table***
  - beschreibt das Unterbrechungssignal
  - dient der Generierung der APIC-Bus Nachricht



# Der I/O APIC

## Aufbau (Bits) eines Eintrags in der *Interrupt Redirection Table*

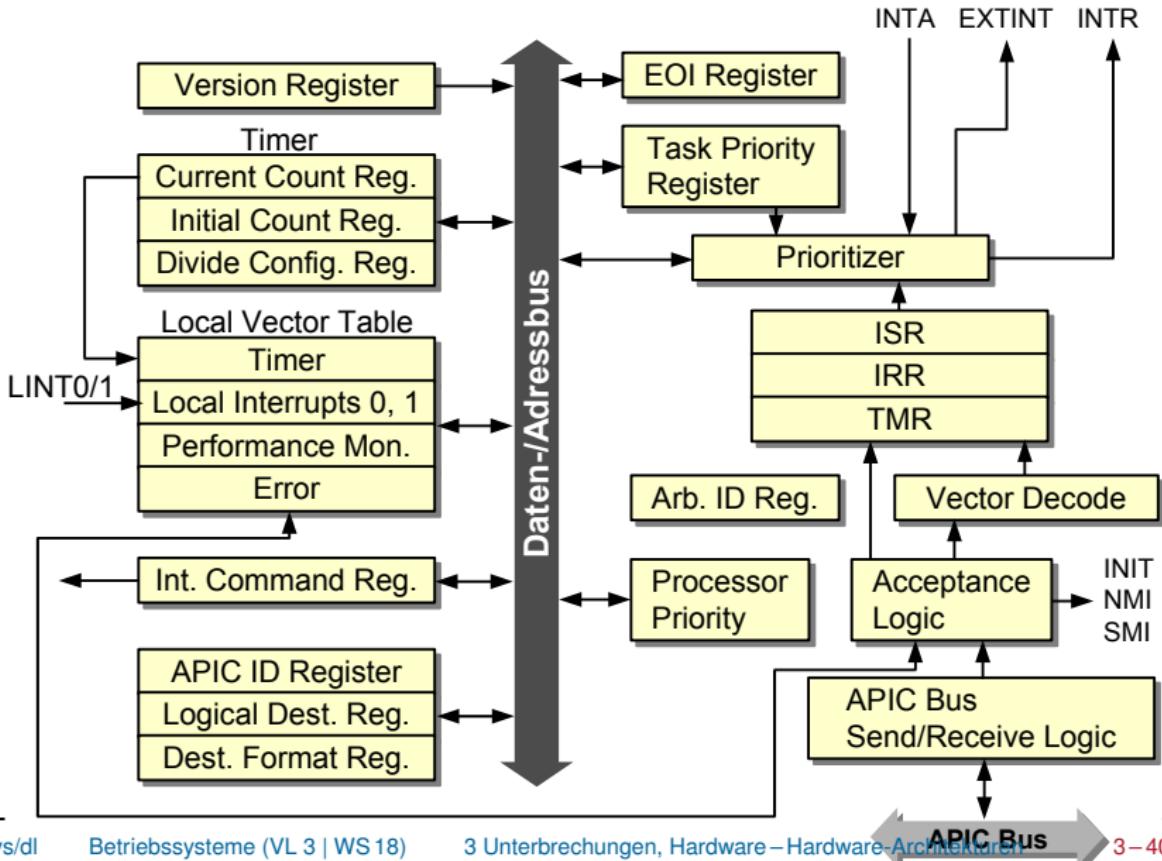
|       |  |  |
|-------|--|--|
| 63:56 | <b>Destination Field</b><br>je nach Bit 11:  | – R/W. 8 Bit Zieladresse.<br>APIC ID der CPU ( <i>Physical Mode</i> ) oder<br>CPU Gruppe ( <i>Logical Mode</i> )   |
| 55:17 | <reserviert>   |  |
| 16    | <b>Interrupt-Mask</b>  | – R/W. Unterbrechungssperre.   |
| 15    | <b>Trigger Mode</b>  | – R/W. <i>Edge-</i> oder <i>Level-Triggered</i>  |
| 14    | <b>Remote IRR</b>  | – RO. Art der erhaltenen Bestätigung   |
| 13    | <b>Interrupt Pin Polarity</b>  | – R/W. Signalpolarität   |
| 12    | <b>Delivery Status</b>   | – RO. Interrupt-Nachricht unterwegs?   |
| 11    | <b>Destination Mode</b>  | – R/W. <i>Logical Mode</i> oder <i>Physical Mode</i>   |
| 10:8  | <b>Delivery Mode</b><br><br>000 – <i>Fixed</i> :<br>001 – <i>Lowest Priority</i> :<br>010 – <i>SMI</i> :<br>100 – <i>NMI</i> :<br>101 – <i>INIT</i> :<br>111 – <i>ExtINT</i> : | – R/W. Wirkung bei Ziel-APIC<br><br>Signal an alle Zielprozessoren ausliefern<br>Liefern an CPU mit aktuell niedrigster Prio.<br><i>System Management Interrupt</i><br><i>Non-Maskable Interrupt</i><br>Ziel-CPUs initialisieren (Reset)<br>Antwort an PIC 8259A |
| 7:0   | <b>Interrupt Vector</b>  | – R/W. <b>8 Bit Vektornummer (16 – 254)</b>  |

# Local APICs

- empfangen Unterbrechungsanforderungen vom APIC Bus
- führen die Auswahl und Priorisierung durch
- können zwei lokale Unterbrechungen direkt verarbeiten
- enthalten weitere Funktionseinheiten
  - Eingebauten *Timer, Performance Counter*
  - *Command*-Register
    - um selber APIC-Nachrichten zu verschicken
    - insbesondere Inter-Prozessor-Interrupt (IPI)
- programmierbar über 32 Bit Register ab 0xfee00000
  - memory mapped (ohne externe Buszyklen)
  - jede CPU programmiert „ihren“ *Local APIC*



# Local APICs - Register



# APIC Architektur - Zusammenfassung

- flexible Verteilung an CPUs im x86 Multiprozessorsystem
  - fest, Gruppen, an die CPU mit der geringsten Priorität
  - Liegen mehrere IRQs an, so wird nach Vektornummer priorisiert
- Vektornummer 16-254 können frei zugeordnet werden
  - sollte (an sich) reichen, um „sharing“ zu vermeiden
- Local APIC erwartet explizites EOI
  - dafür muss die Software sorgen
- Mit APIC unterstützt x86 prinzipiell auch Prioritätsebenen
  - Systemsoftware muss jedoch entsprechend agieren  
(Unterbrechungen freigeben, evtl. *Task-Priority-Register* verwenden)



# Agenda

---

Einordnung

Grundlegende Fragestellungen

Hardware-Architekturen

Zusammenfassung und Ausblick



# Zusammenfassung und Ausblick

- Unterbrechungsbehandlungshardware befasst sich mit ...
  - Priorisierung
  - Zuordnung und Ausführung einer Behandlungsroutine
  - Zustandssicherung und geschachtelter Ausführung
- moderne Unterbrechungsbehandlungshardware kann ...
  - Unterbrechungsvektoren frei zuordnen
  - „sharing“ von Unterbrechungsvektoren vermeiden
  - Unterbrechungen im Multiprozessorsystem flexibel zuordnen
- das Betriebssystem muss ...
  - Probleme wie „*spurious interrupts*“ und „*interrupt storms*“ einkalkulieren.
  - das eingetretene Ereignis aus der Behandlungsroutine an die höheren Ebenen und letztendlich zum Anwendungsprozess weiterleiten.

