Betriebssystem-Entwicklung

4. Unterbrechungen

Michael Schöttner

Vorschau

Heinrich Heine
Universität Düsseldorf

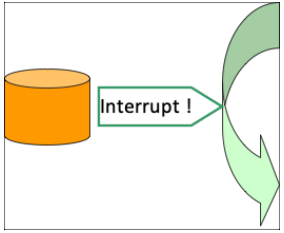
- Grundlagen
- Unterbrechungsbehandlung beim x86_64
- Programmable Interrupt Controller
- Advanced Programmable Interrupt Controller
- Behandlung von Unterbrechungen im Betriebssystem
- Zusammenfassung

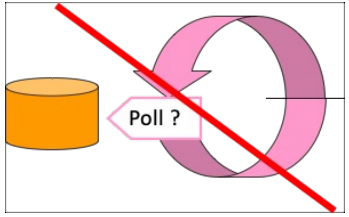
4.1 Grundlagen

- Unterbrechungen (engl. interrupts) wurden eingeführt, damit Ein-/Ausgaben überlappend durchgeführt werden können

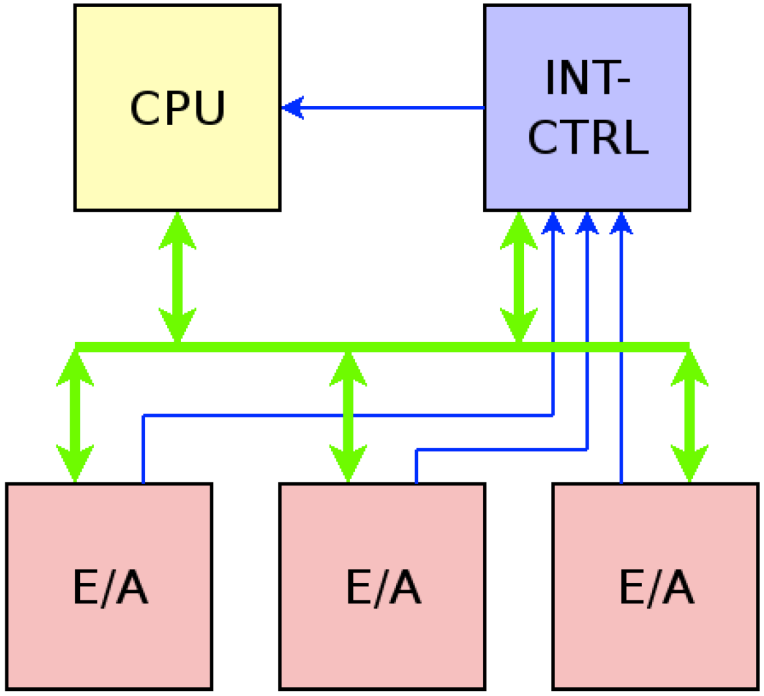
- Statt aktivem Warten auf das Ende eines Ein-/Ausgabeauftrags wird der betroffene Thread blockiert und auf einen anderen Thread umgeschaltet
  → damit kann der Prozessor sinnvoll genutzt werden

- Darüber hinaus sind Unterbrechungen die Basis für präemptives Multithreading
  - Programme Interval Timer (PIT) erzeugt periodisch Interrupts (siehe PC-Lautsprecher)
  - präemptives Multithreading folgt später





Interrupt-Hardware: Single Core



Priorisierung

Heinrich Heine
Universität Düsseldorf

- Ausgangssituation:
  - Mehrere Unterbrechungen können gleichzeitig signalisiert werden. Welche ist wichtiger?
  - Während die CPU eine Unterbrechung behandelt, können weitere signalisiert werden

- Priorisierungsmechanismus
  - in Software: die CPU hat nur einen IRQ (interrupt request) Eingang
  - in Hardware: eine Priorisierungsschaltung ordnet Geräten eine Priorität zu und leitet immer nur die dringendste Unterbrechung (IRQ) zur Behandlung and die CPU weiter
  - Moderne Interrupt-Controller erlauben dynamische Prioritäten

Verlust von IRQs

Heinrich Heine
Universität Düsseldorf

- 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

Heinrich Heine
Universität Düsseldorf

- Problem:
  - die Software sollte möglichst schnell herausfinden können, welches Gerät eine Unterbrechung ausgelöst hat
    - Wir möchten nicht reihum alle Geräte abfragen
- Lösung: Interrupt-Vektor
  - jeder Unterbrechung wird eine Vektor-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 (shared interrupts)
  - der Aufbau der Vektortabelle hängt vom Prozessortyp ab
    - meist enthält sie nur Zeiger auf Funktionen

Zustandssicherung

Heinrich Heine
Universität Düsseldorf

- Problem:
  - nach der Ausführung der Behandlungsroutine muss zum normalen Thread-Kontext zurückgekehrt werden können
  - die Behandlung soll unbemerkt “eingeschoben“ werden

- Lösung: Zustandssicherung
  - Durch die Hardware
    - nur das Notwendigste: z.B. Rücksprungadresse u. Prozessorstatuswort
    - Wiederherstellung durch speziellen Befehl, z.B. `iret`, ...
  - Durch die Software
    - da Unterbrechungen jederzeit auftreten können, muss auch die Behandlungsroutine Zustände (alle Register) sichern und wiederherstellen

Geschachtelte Unterbrechungen

Heinrich Heine
Universität Düsseldorf

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

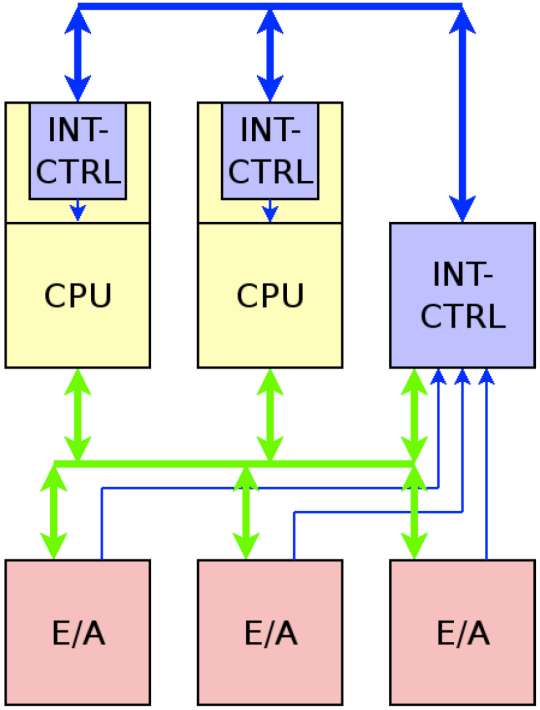
- Lösung:
  - Die CPU erlaubt immer nur Unterbrechungen mit höherer Priorität
  - In der Praxis kaum verwendet, da kompliziert und i.d.R. nicht notwendig

Multicore-Systeme

Heinrich Heine
Universität Düsseldorf

- Probleme:
  - Welcher Core behandelt eine Unterbrechung?
  - Behandlung von Interprozessor-Unterbrechungen
- Lösung: erweiterte Hardware zur Unterbrechungsbehandlung
  - Verteilung von Interrupts an die Cores ist in Software steuerbar
  - feste oder zufällige Zuordnung
  - Zuordnung unter Berücksichtigung der Thread-Priorität auf dem jeweiligen Core

Interrupt-Hardware: Multi Core



Gefahr: „unechte Unterbrechungen“

Heinrich Heine
Universität Düsseldorf

- = spurious interrupts
- Problem: fehlerhaften Unterbrechungsanforderungen, z.B. durch ...
  - Hardwarefehler
  - fehlerhaft programmierte Geräte
- Lösung:
  - Hardware- und Softwarefehler vermeiden
  - Betriebssystem „defensiv“ programmieren
    - mit unechten Unterbrechungen rechnen

Gefahr: „Unterbrechungsstürme“

Heinrich Heine
Universität Düsseldorf

- = interrupts 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
  - Und das verursachende Gerät deaktivieren

Vorschau

Heinrich Heine
Universität Düsseldorf

- Grundlagen
- Unterbrechungsbehandlung beim x86_64
- Programmable Interrupt Controller
- Advanced Programmable Interrupt Controller
- Behandlung von Unterbrechungen im Betriebssystem
- Zusammenfassung

Externe vs. interne Interrupts

Heinrich Heine
Universität Düsseldorf

- Externe- oder Hardware-Interrupts
  - von einem Gerät, z.B. Timer-Interrupt
  - Kommen von außerhalb, aus Sicht der CPU

- Interne- oder Software- Interrupts:
  - Kommen von der CPU selbst
    - Exceptions (siehe nächste Seite)
    - Oder durch die Assemblerinstruktion `INT <nr>`
      - Verwendet für Systemaufrufe (Linux, Windows NT, MacOS, MSDOS)

Exceptions

Heinrich Heine
Universität Düsseldorf

- **Fault** (dt. Störung):
  - kann behoben werden, z.B. Page Fault
  - CPU-Zustand wird gesichert & Adresse der Instruktion, die Fault ausgelöst hat
- **Trap** (~ dt. Falle):
  - ausgelöst durch speziellen Befehl, z.B. INT 3 (Breakpoint)
  - Programm kann fortgeführt werden
- **Abort** (dt. Abbruch):
  - bei schwerem Fehler
  - Auslöser oft nicht genau lokalisierbar
  - führt zum Restart (z.B. Double Fault)

Vectors

- Interrupts und Exceptions werden durch eine Vektornummer identifiziert
- 0 – 31 ist reserviert für Exceptions
- 32 – 255 steht zur freien Verfügung
- Exception-Auszug

Vektor	Bedeutung	Vektor	Bedeutung
0	Division by 0	11	Segment fault
1	Debug	12	Stack overflow
2	NMI	13	General protection fault
3	Break	14	Page fault
4	Overflow	16	-
5	Bounds range exceeded	18	Machine check
6	Illegal instruction	…	. . .
8	Double fault		

Zuordnung von Vektoren zu Handlern

Heinrich Heine
Universität Düsseldorf

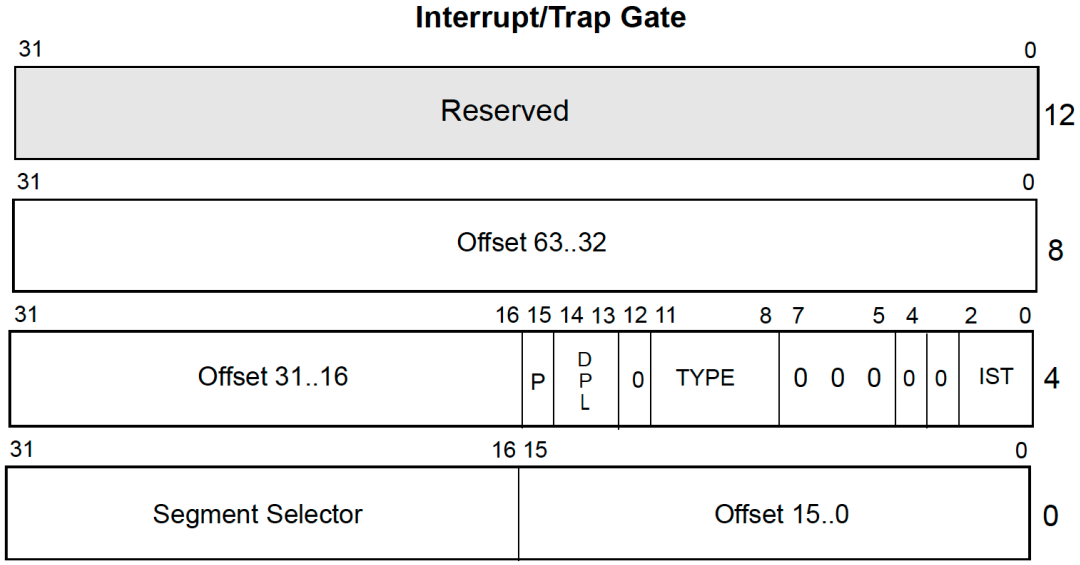
- Interrupt Deskriptor Table (IDT)
- Startadresse der IDT wird im IDTR (1x pro Core vorhanden) gespeichert
- 255 Einträge bilden jeden Vektor auf eine Funktionsadresse ab
  - Diese Funktion ist der Interrupt-Handler / Interrupt Service Routine / ...
  - Pro Eintrag 8 Byte, entweder ein Interrupt- oder ein Trap-gate

- **Interrupt-Gate**: beschreibt einen Interrupt-Handler
  - Wenn der Eintrag verwendet wird, werden die Interrupts auf dem jeweilige Core maskiert (Interrupt Enable Bit im RFLAGS wird gelöscht)

- **Trap-Gate**: beschreibt einen Trap-Handler
  - Arbeitet die das Interrupt-Gate, aber die Interrupts werden nicht maskiert
  - Verwendet für beispielsweise System-Aufrufe via Interrupt

Zuordnung von Vektoren zu Handlern

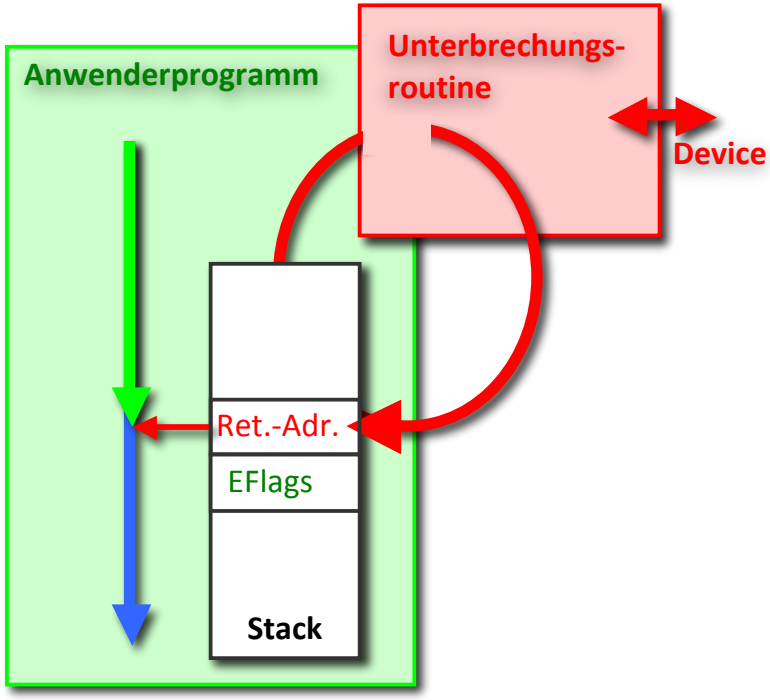
- Interrupt Table Entry Format in Long Mode
- Segment Selector wählt ein Segment in der GDT



DPL	Descriptor Privilege Level
Offset	Offset to procedure entry point
P	Segment Present flag
Selector	Segment Selector for destination code segment
IST	Interrupt Stack Table

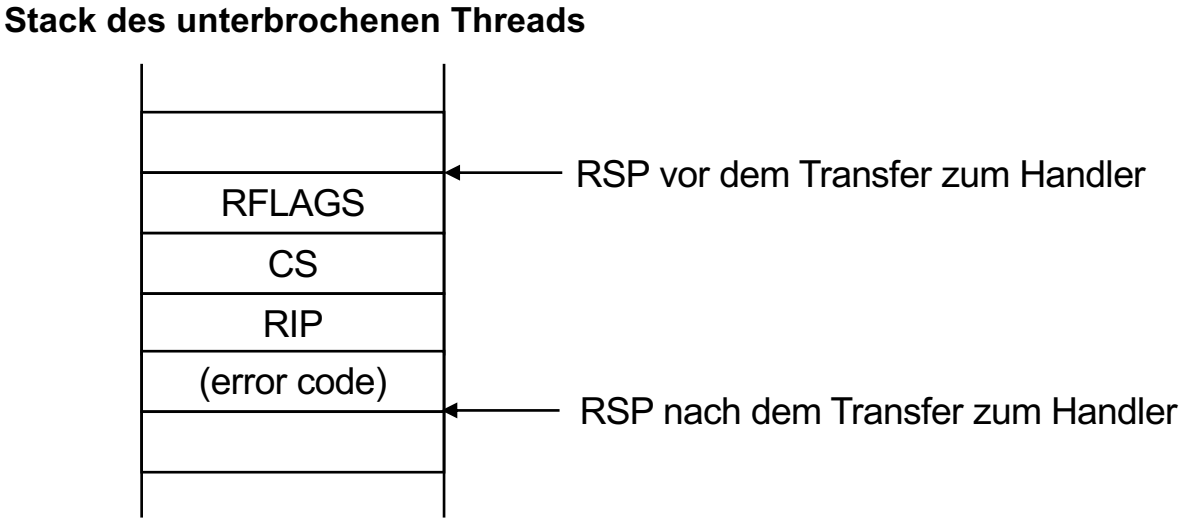
Stackaufbau bei einer Unterbrechung

- Abstrakter Ablauf & Stackaufbau
- Programm wird unterbrochen, nach Abschluss der letzten Instruktion



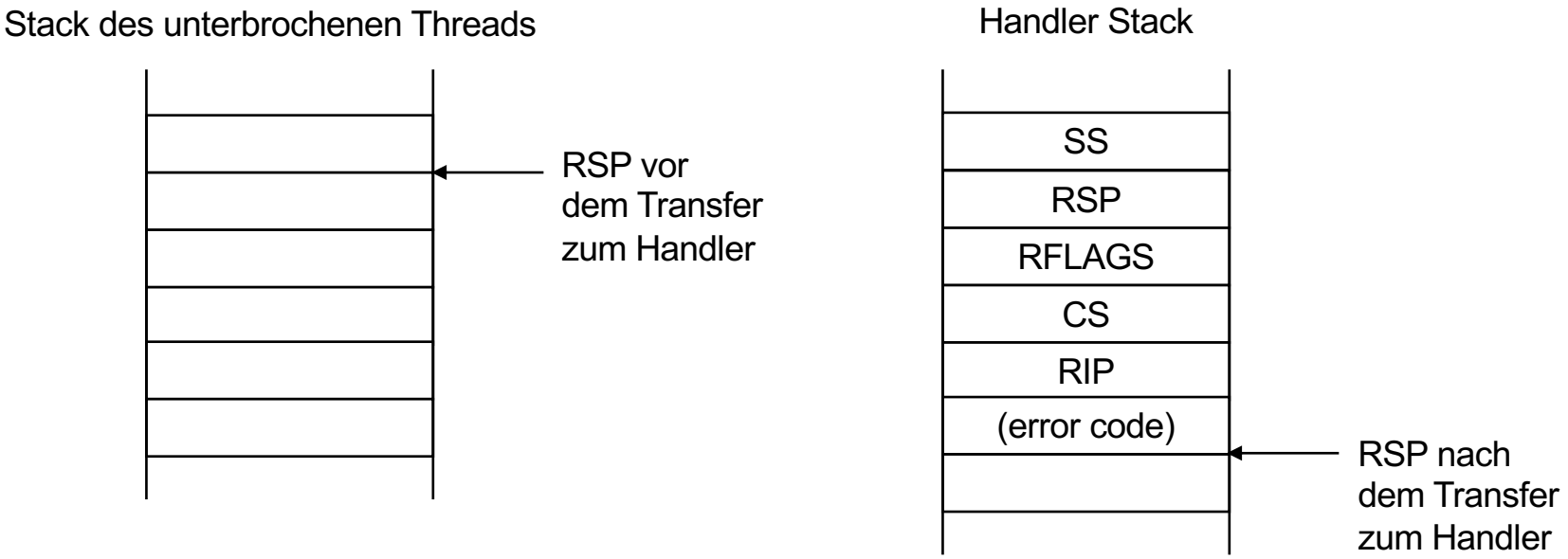
Stackaufbau ohne Ringwechsel

- Falls keine Privilegstufe gewechselt wird, so wird auch der Stack nicht umgeschaltet
- D.h. der Interrupt-Handler verwendet den Stack des unterbrochenen Threads
- Dies entspricht dem Ablauf bei hhuTOS (unser gesamter Code läuft im Ring 0)
- Einen error code gibt es nur bei manchen Exceptions



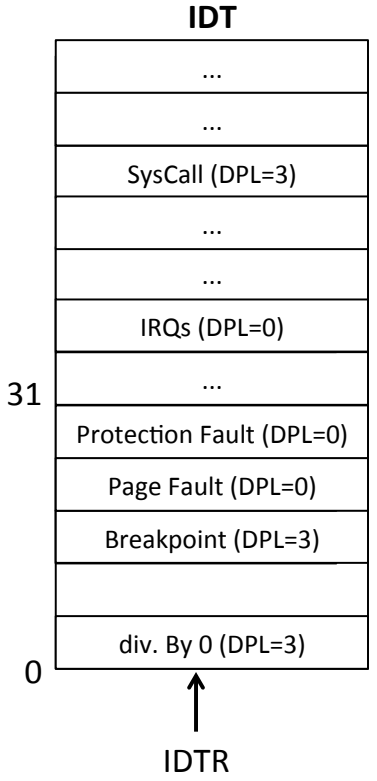
Stackaufbau bei einem Ringwechsel

- Neuer Stack wird aus dem Task State Segment ermittelt (siehe letztes Kapitel)

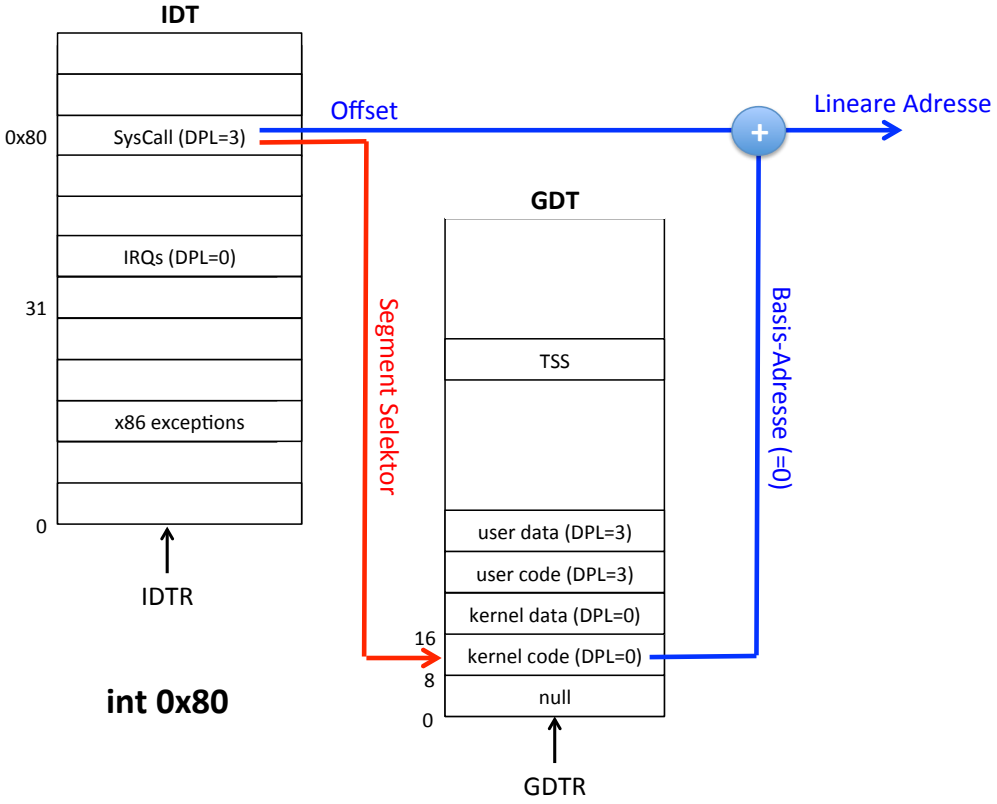


Beispiel einer typischen IDT

- Einträge 0-31 sind durch x86 reserviert für Exceptions
  - Je nach Zweck DPL=0 oder DPL=3
- Einträge für externe Interrupts
  - IRQs haben DPL=0
- System-Aufrufe per Trap-Gate (Windows & Linux)
  - DPL=3
  - Ersetzt in modernen BS durch SYSENTER/SYSEXIT → schneller, vermeiden Interrupt-Overhead
- Da hhuTOS komplett in Ring-0 läuft benötigen wir kein Trap-Gate



Systemaufruf über ein Trap-Gate



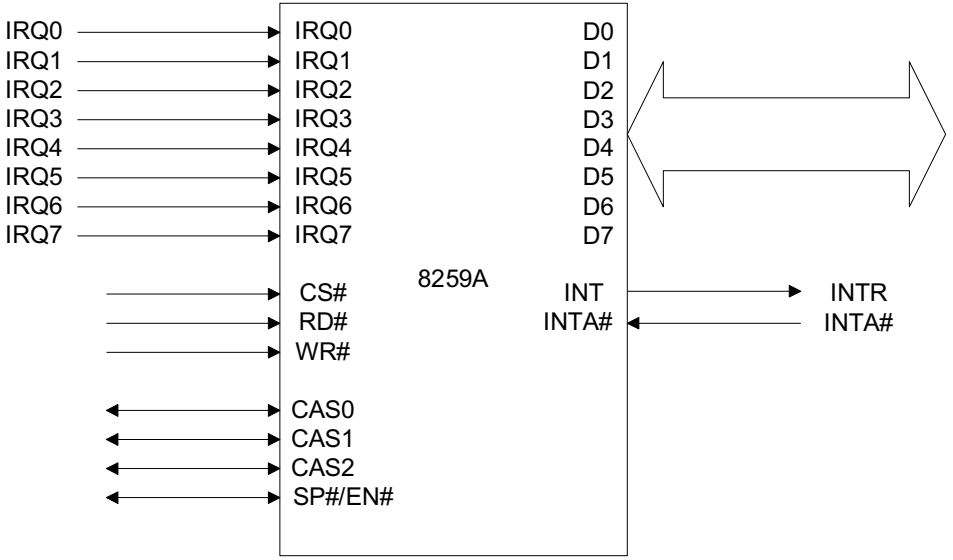
Vorschau

Heinrich Heine
Universität Düsseldorf

- Grundlagen
- Unterbrechungsbehandlung beim x86_64
- Programmable Interrupt Controller
- Advanced Programmable Interrupt Controller
- Behandlung von Unterbrechungen im Betriebssystem
- Zusammenfassung

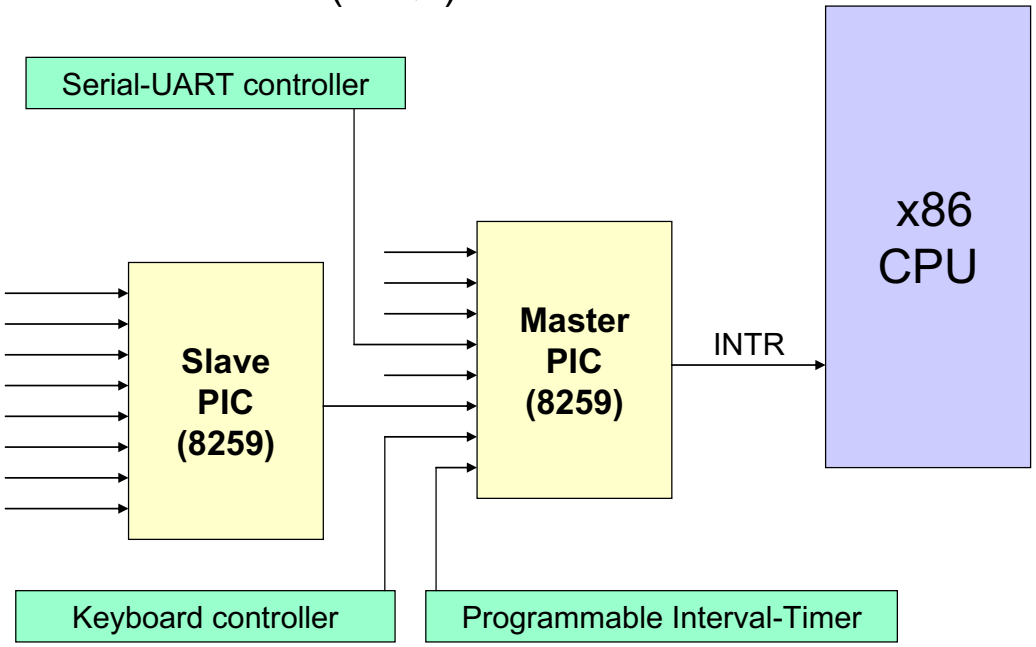
Unterbrechungen bei x86 CPUs

- 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



PIC-Architektur

- Master-Kontroller kann bis zu 8 Slaves steuern (CAS0..2 = Cascade Lines)
- Im PC nur ein Slave vorhanden (IRQ2)



PIC-Architektur

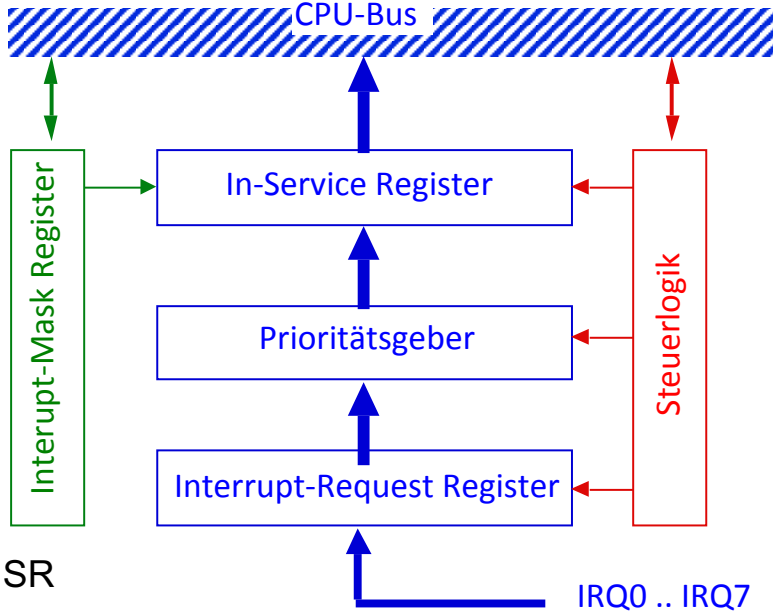
- Typische IRQ Zuordnung in PCs (Auswahl):

IRQ#	Bedeutung
0	Programmable Interval Timer
1	Keyboard Controller
2	Slave PIC
8	Real Time Clock
…	…
12	PS/2 Mouse
…	…
3	COM2
…	...

- Prioritäten
  - Höchste Priorität hat IRQ0
  - Default IRQ-Prioritäten: 0, 1, 8 ...15, 3 ... 7 (bedingt durch Kaskadierung)
  - Interrupts mit höherer Priorität können solche mit niedriger Priorität unterbrechen, sofern das IE-Bit (IE = Interrupt-Enable) im Flag-Register dies zulässt

Register des PIC

- Register haben je 8 Bit (8 IRQs = 8 Leitungen)
- Interrupt Mask Register (IMR)
  - unterdrückt Interrupts (0=enable, 1=masked).
- Interrupt Request Register (IRR):
  - speichert wartende Interrupts
- Prioritätsgeber:
  - Propagiert IRQ aus IRR mit höchster Priorität nach ISR
- In-Service Register (ISR): enthält Interrupt(s) die gerade bearbeitet werden



Ablauf Interruptverarbeitung: PIC <-> CPU

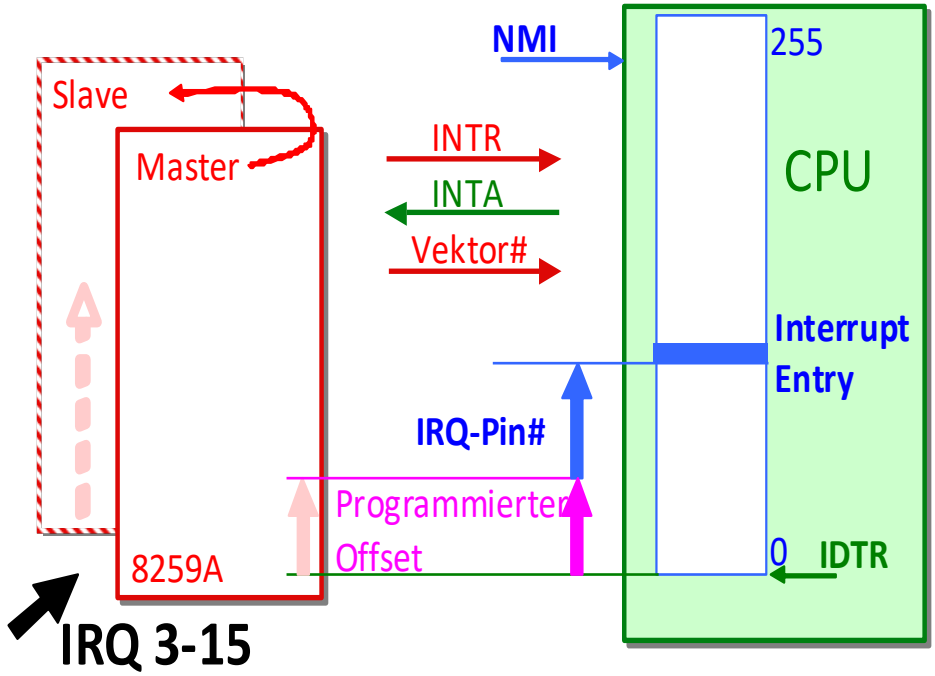
Heinrich Heine
Universität Düsseldorf

- IRQ-Leitung wird aktiviert → entsprechendes Bit in IRR wird gesetzt.
- PIC sendet INTR Signal an CPU (oder falls Slave zunächst an Master)
- CPU antwortet mit INTA Impuls, falls IE-Bit in RFLAGS gesetzt ist.
- Höchstwertiges Bit in IRR wird gelöscht und in ISR Register gesetzt.
- CPU sendet zweiten INTA Impuls
- PIC legt Vektor-Zeiger (8 Bit = 3 Bit IRQ + 5 Bit Offset) auf Datenbus.

- Interrupt Handler schickt am Ende ein EOI an den PIC → löscht ISR-Bit.
- Bem.: falls Interrupt von Slave → zusätzl. EOI an Master für IRQ2!
- Das EOI entfällt in hhuTOS, da wir den PIC im Automatic EOI Modus betreiben!

Ablauf Interruptverarbeitung: PIC <-> CPU

- Interne Unterbrechungen:
  - Feste Vektornummer 0 – 31
- Externe Unterbrechungen:
  - Vektor = IRQ + Offset
  - Offset>31 und IRQ!=Vektor
  - Der Offset muss die IRQs auf Vektor-Nummer > 31 abbilden!



Vorschau

Heinrich Heine
Universität Düsseldorf

- Grundlagen
- Unterbrechungsbehandlung beim x86_64
- Programmable Interrupt Controller
- Advanced Programmable Interrupt Controller
- Behandlung von Unterbrechungen im Betriebssystem
- Zusammenfassung

Advanced Programmable Interrupt Controller

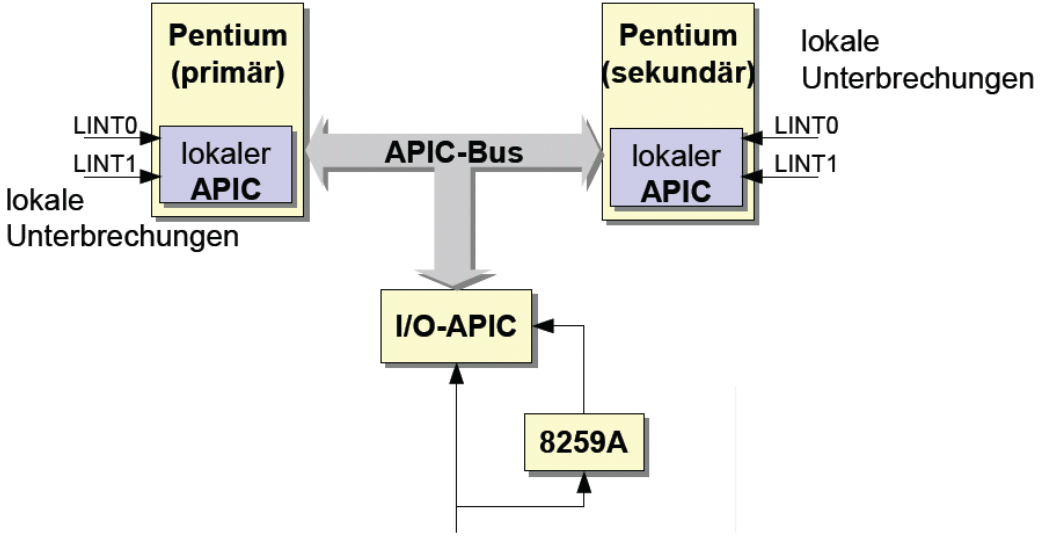
Heinrich Heine
Universität Düsseldorf

- APIC bietet mehr Interrupts als PIC und unterstützt Multikern- und Multiprozessorsysteme
  - Verteilung von Interrupts auf Cores unter Beachtung von Prioritäten der Threads
  - Interrupts zwischen Cores, u.a. für Multicore Bootstrapping
  - Ist Rückwärtskompatibel zu PIC (8259)

- Der APIC ist nach dem Booten aktiviert
  - Arbeitet aber zunächst nur im PIC-kompatiblen Modus
  - Die APIC-Register findet man über sogenannte Model Specific Registers (MSR)

APIC-Architektur

- Ein APIC Interrupt-System besteht aus einem lokalen APIC pro Core sowie i.d.R. einem I/O APIC (können aber auch mehrere sein)



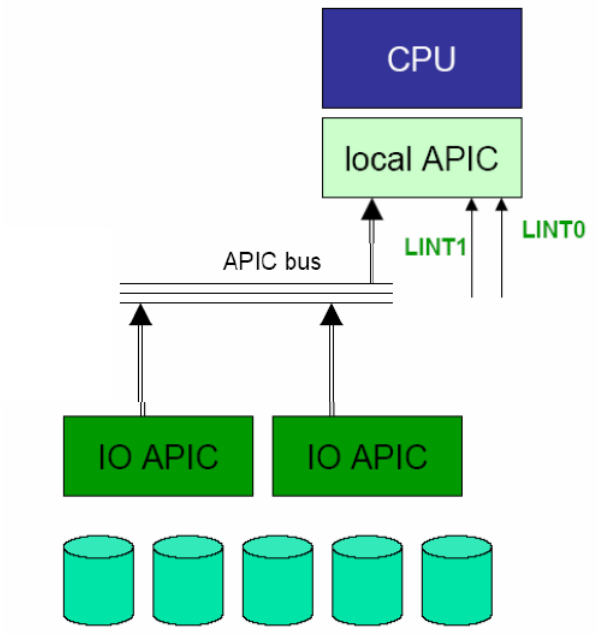
Lokaler APIC (pro CPU/Core)

Heinrich Heine
Universität Düsseldorf

- Empfängt Unterbrechungs-Anforderungen normalerweise vom APIC-Bus
- Kann zusätzlich zwei lokale Unterbrechungen direkt verarbeiten
  - Über CPU-PINs: LINT0 und LINT1 (LINT = Local Interrupt)
  - Falls APIC deaktiviert wird, so hat LINT0 die Rolle des alten INTR und LINT1 übernimmt den NMI (nicht notwendig für HHUos)
- APICs sind über eine ID adressierbar (wird auto. festgelegt, siehe später)
- Enthalten weitere Funktionseinheiten
  - Eingebauten Timer (abhängig von System-Bus Taktrate, kann entsprechend geteilt werden)
  - Interrupt Command-Register
    - um selber APIC-Nachrichten zu verschicken, für Inter-Prozessor-Interrupt (IPI)

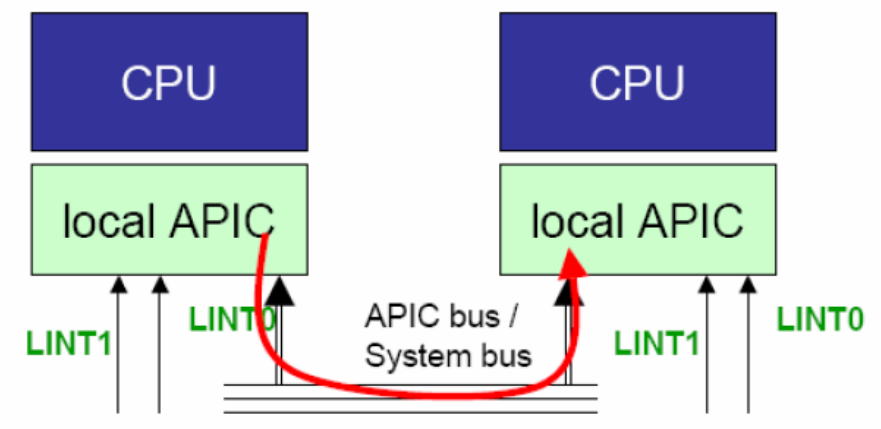
Interrupt-Quellen (1)

- Externer Geräteinterrupt, vermittelt durch einen IO-APIC über den APIC-Bus



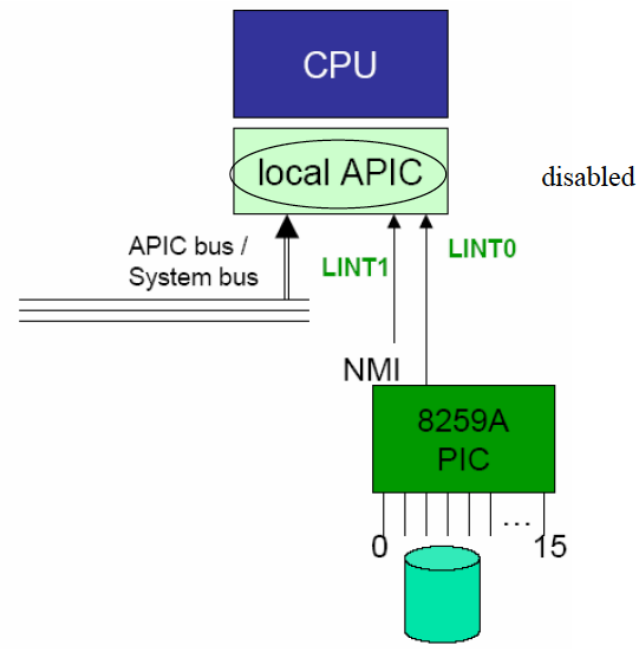
Interrupt-Quellen (2)

- IPI – Inter-Processor Interrupt
  - erzeugt durch einen anderen lokalen APIC → für Multi-Core / Multi-Processor Systeme
  - werden über den APIC Bus als Nachrichten verschickt
  - Z.B. Koordinierung Startup & Shutdown, cache & TLB flush wegen Prozesswechsel, etc.



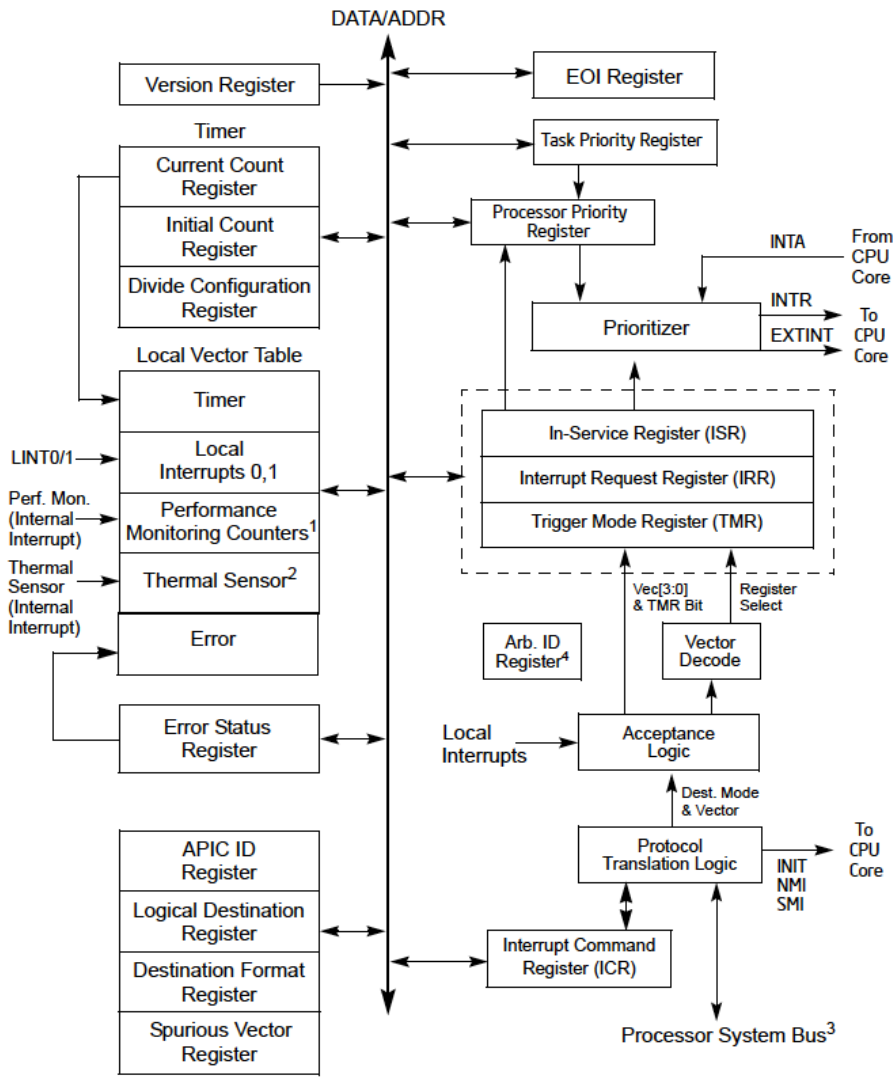
Interrupt-Quellen (3)

- Lokaler Geräteinterrupt:
  - An einem CPU-Pin: LINT0, LINT1 (nicht über den APIC Bus)
  - Beispiele:
    - Performance Monitoring Interrupt (MSR)
    - Temperatur Sensor Interrupt
    - 8259A (falls vorhanden)



Registersatz für Local APIC

- Task Priority Register (TPR)
  - Speichert Thread-Priorität
  - Verwaltet durch OS
- Processor Priority Register
  - verwaltet durch CPU
  - ergibt sich aus TPR & ISR
- EOI Register
  - End-Of-Interrupt
  - Vor IRET schreiben
- Interrupt Command Register
  - Für Inter-Processor-Interrupts



Auslieferung eines Interrupts an einen APIC

Heinrich Heine
Universität Düsseldorf

- Statische Interrupt-Auslieferung bei flat-model Adressierung
  - Keine Arbitrierung, Auslieferung an adressierte CPU, notfalls warten.

- Dynamische Interrupt-Auslieferung:
  - Nachdem die Gruppe der adressierten CPUs/APICs bestimmt ist, wird eine „Lowest Priority Arbitration“ durchgeführt.
  - Ziel: Prozessor mit derzeit am wenigsten wichtigen Aufgabe soll den Interrupt behandeln
    - Berücksichtigt Processor Priority Register und Task-Priority-Register

I/O-APIC

Heinrich Heine
Universität Düsseldorf

- Heute typischerweise in der Southbridge von PC-Chipsätzen integriert
- 24 Interrupt-Eingänge
  - zyklische Abfrage durch Chip
- Für jeden Interrupt-Eingang gibt es einen 64 Bit Eintrag in der Interrupt Redirection Table
  - beschreibt das Unterbrechungssignal
  - legt fest, welcher Interrupt an welchen Prozessor-Kern geschickt wird und mit welcher Vektor-Nummer (dient der Generierung der APIC-Bus Nachricht)
  - Die IRT hat 24 Einträge

Eintrag in der Interrupt Redirection Table

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