

Betriebssysteme (BS)

IA-32

das Programmiermodell der Intel-Architektur

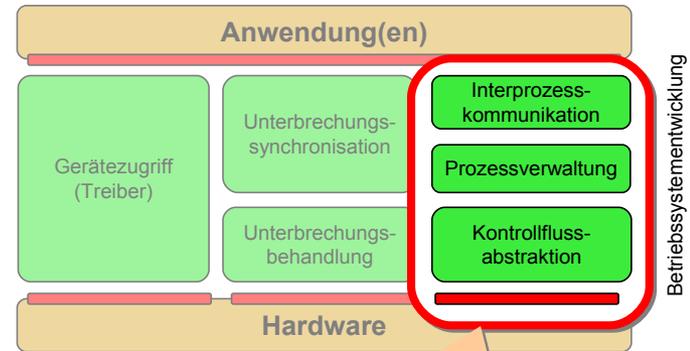
Daniel Lohmann

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme



1

Überblick: Vorlesungen



BS © 2007, 2008 Daniel Lohmann, Olaf Spinczyk

2

Agenda

- Historie
- Basisprogrammiermodell
- Speicherverwaltung und Adressierung
- Schutz
- "Tasks"
- Zusammenfassung



BS © 2007, 2008 Daniel Lohmann, Olaf Spinczyk

3

Historie der Intel x86-Prozessoren

- **8086** (1978) – der Urvater des PC Prozessors
- **80286** (1982) – Einführung des *Protected Mode*
 - segmentbasierter Speicherschutz
- **80386** (1985) – erster IA-32 Prozessor
 - seitenbasierter virtueller Speicher
- **80486** (1989) – integrierte FPU, RISC Ansätze
- **Pentium** (1993) – superskalar, 64-Bit Datenbus
 - SMM, MMX, APIC, Dualprozessor-fähig
- **Pentium II** (1997) – Pentium Pro + MMX
 - RISC-artige Mikroinstruktionen
- **Pentium III** (1999) – SSE
- **Pentium 4** (2000) – *Netburst* Architektur
 - SSE2, Hyperthreading, Vanderpool, EM64T
- **Core** (2005) – Multicore, weniger Strom
- **Core 2** (2006) – Dual Core, Quad Core, 64 Bit



BS © 2007, 2008 Daniel Lohmann, Olaf Spinczyk

4

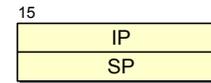
8086: Programmiermodell

- 16-Bit Architektur, little-endian
- 20-Bit Adressbus, d.h. maximal 1 MiB Hauptspeicher
- wenige Register
 - (jedenfalls aus heutiger Sicht)
- 123 Befehle
 - kein orthogonaler Befehlssatz
- Befehls­längen von 1 bis 4 Byte
- segmentierter Speicher
- noch immer aktuell**
 - obwohl von 1978 noch heute von jeder IA-32 CPU unterstützt
 - *Real Mode, Virtual 8086 Mode*

Aufwärtskompatibilität wird bei Intel groß geschrieben

8086: Registersatz

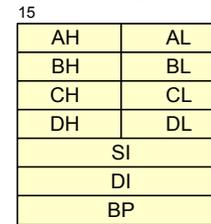
Befehls- und Stapelzeiger



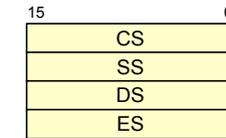
Flag Register



Vielzweckregister



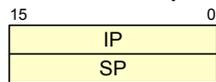
Segmentregister



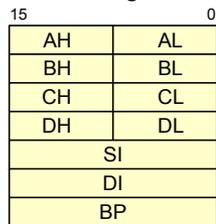
Code
Stack
Data
Extra

8086: Registersatz

Befehls- und Stapelzeiger



Vielzweckregister



AX: Accumulator Register

- arithmetisch-logische Operationen
- I/O
- kürzester Maschinencode

BX: Base Address Register

CX: Count Register

- für LOOP Befehl
- für *String* Operationen mit REP
- für Bit *Shift* und *Rotate*

DX: Data Register

- DX:AX sind 32 Bit für MUL/DIV
- Portnummer für IN und OUT

SI, DI: Index Register

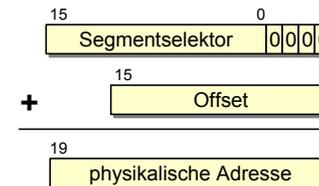
- für Array-Zugriffe (Displacement)

BP: Base Pointer

Jedes „Vielzweckregister“ erfüllt seinen speziellen Zweck

8086: Segmentierter Speicher

- logische Adressen bestehen beim 8086 aus
 - Segmentsektor (i.d.R. der Inhalt eines Segmentregisters)
 - Offset (i.d.R. aus einem Vielzweckregister oder dem Befehl)
- Berechnung der physikalischen Adresse:



die 16 Bit Konkurrenz konnte i.d.R. nur 64KB adressieren

8086: Segmentierter Speicher

- logische Adressen bestehen beim 8086 aus
 - Segmentselektor (i.d.R. der Inhalt eines Segmentregisters)
 - Offset (i.d.R. aus einem Vielseitregister oder dem Befehl)
- Be

„640K ought to be enough for anybody“

angeblich ein Zitat von Bill Gates, 1981

physikalische Adresse

die 16 Bit Konkurrenz konnte i.d.R. nur 64KB adressieren



8086: Speichermodelle

- Programme können Adressen unterschiedlich bilden. Das Ergebnis waren unterschiedliche Speichermodelle:
- Tiny**
 - Code-, Daten- und Stacksegment sind identisch: 64K insgesamt
- Small**
 - Trennung des Codes von Daten und Stack: 64K + 64K
- Medium**
 - 32(20) Bit Zeiger für Code, Daten- und Stapelseg. aber fest (64K)
- Compact**
 - Festes Code Segment (64K), 32(20) Bit Zeiger für Daten und Stack
- Large**
 - „far“ Zeiger für alles: 1MB komplett nutzbar
- Huge**
 - wie „Large“, aber mit normalisierten Zeigern



8086: Fazit

- Urvater der PC-Prozessoren
 - bildete den Kern der ersten PCs
 - noch heute sind IA32-Prozessoren kompatibel
- Segmentregister brachten Vorteile
 - trotz 16-Bit-Architektur 1 MB Speicher
 - Trennung von logischen Modulen im Hauptspeicher
- Programm- und Übersetzerentwicklung ist aber vergleichsweise schwierig
 - verschiedene Speichermodelle
 - nicht orthogonaler Befehlssatz



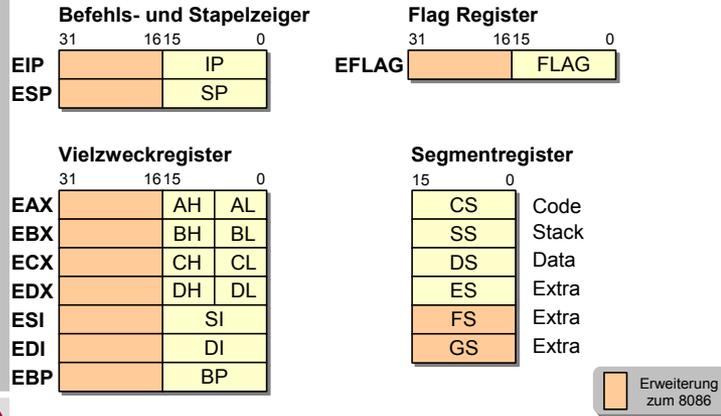
IA-32 – die 32 Bit Intel Architektur

- die erste IA-32 CPU war der **80386**
 - wobei der Begriff „IA-32“ erst sehr viel später eingeführt wurde
- 32 Bit Technologie: Register, Daten- und Adressbus
 - ab Pentium Pro: 64 Bit Daten und 36 Bit Adressbus
- zusätzliche Register
- komplexe Schutz- und Multitaskingunterstützung
 - Protected Mode*
 - ursprünglich schon mit dem 80286 (16-Bit) eingeführt
- Kompatibilität
 - mit älteren Betriebssystemen durch den *Real Mode*
 - mit älteren Anwendungen durch den *Virtual 8086 Mode*
- segmentbasiertes Programmiermodell
- seitenbasierte MMU

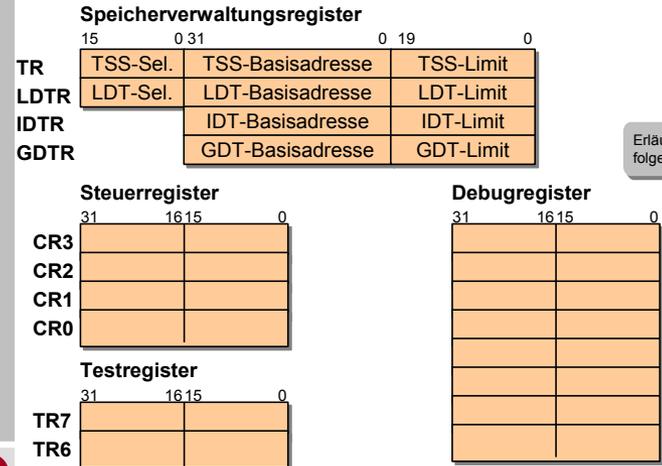


80386: Registersatz (Erweiterungen)

- erweiterte Register heißen aus Kompatibilitätsgründen E...



80386: Registersatz (neue Register)



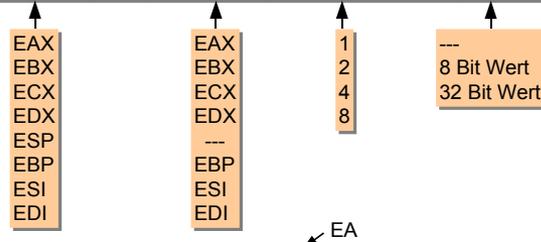
Erläuterungen folgen ...



IA-32: Adressierungsarten

- Effektive Adressen (EA) werden nach einem einfachen Schema gebildet
 - alle Vielseckregister können dabei gleichwertig verwendet werden

$$EA = \text{Basis-Reg.} + (\text{Index-Reg.} * \text{Scale}) + \text{Displacement}$$

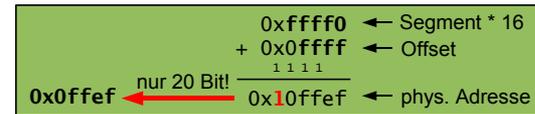


- Beispiel: MOV EAX, Feld[ESI * 4]
- Lesen aus Feld mit 4 Byte großen Elementen und ESI als Index



Das A20-Gate

- ... ist ein Relikt aus der Zeit der 80286 Systeme (IBM AT)
 - beim IBM XT (8086) konnte es bei der Adressberechnung zu einem Überlauf kommen. Im Maximalfall:



- MS-DOS (und andere Systeme) verwenden diesen „Trick“.
- Aus Kompatibilitätsgründen wurde im IBM AT die A20-Leitung über das „A20 Gate“ (Register im Tastaturcontroller) maskiert.
 - A20 muss explizit freigeschaltet werden, um Speicher > 1 MiB zu adressieren
- Ab dem 486 hat Intel das A20-Gate in die CPU integriert!

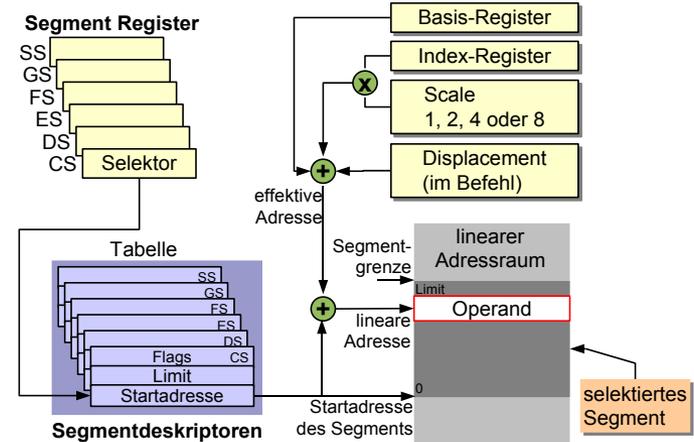


IA-32: Protected Mode – Segmente

- ein Programm (in Ausführung) besteht aus mehreren Speichersegmenten
 - mindestens CODE, DATEN und STACK
 - Segmentselektoren beschreiben (indirekt) Adresse und Länge
- „Lineare Adresse“ ist Segmentstartadresse + EA
 - Segmente dürfen sich im linearen Adressraum überlappen, z.B. dürfen die Segmentstartadressen bei 0 liegen. Dadurch wird ein „flacher“ Adressraum nachgebildet.
 - „Lineare Adresse“ entspricht der physikalischen Adresse, falls die Paging Unit nicht eingeschaltet ist.

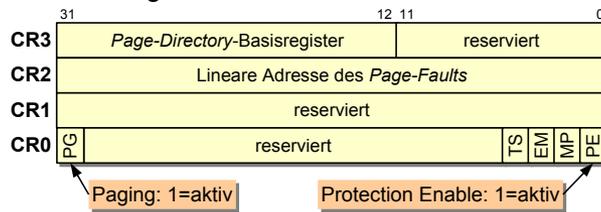


IA-32: Protected Mode – Segmente

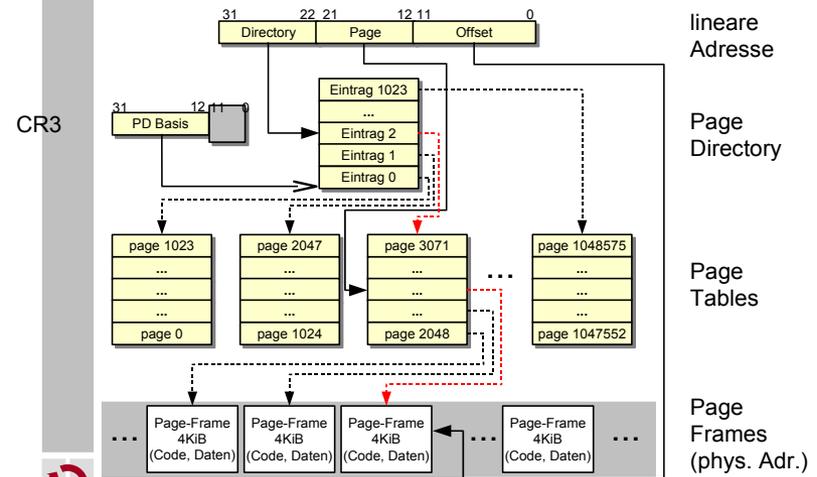


IA-32: Seitenbasierte MMU (1)

- Ein- und Auslagerung von Speicher (zwecks virtuellem Speicher) ist bei Segmentierung aufwändig. Daher bieten viele andere CPUs lediglich eine seitenbasierte Speicherverwaltung.
- ab dem 80386 kann eine **Paging Unit (PU)** optional hinzugeschaltet werden.
- die wichtigsten Verwaltungsinformationen stehen in den CRx Steuerregistern:



IA-32: Seitenbasierte MMU (2)



IA-32: TLB

- Problem: bei aktiver *Paging Unit* wäre eine IA-32 CPU erheblich langsamer, wenn bei jedem Speicherzugriff das *Page Directory* und die *Page Table* gelesen werden müssten
- Lösung: der **Translation Lookaside Buffer** (TLB):
 - vollassoziativer Cache
 - Tag: 20 Bit Wert aus *Page Directory* und *Page Table* Index
 - Daten: *Page Frame* Adresse
 - Größe beim 80386: 32 Einträge
 - bei normalen Anwendungen erreicht der TLB eine Trefferrate von etwa 98%
 - Schreiben in das CR3 Register invalidiert den TLB



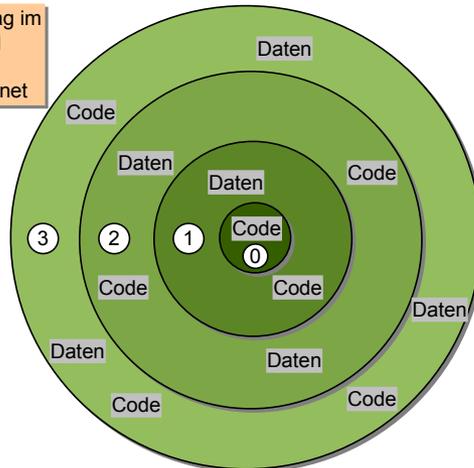
Schutz

- die wichtigste Eigenschaft des IA-32 Protected Mode ist das Schutzkonzept
- **Ziel:** fehlerhaften oder nicht vertrauenswürdigen Code isolieren
 - Schutz vor Systemabstürzen
 - Schutz vor unberechtigten Datenzugriffen
 - keine unberechtigten Operationen, z.B. I/O Port Zugriffe
- Voraussetzungen: Code und Daten ...
 - werden hinsichtlich der Vertrauenswürdigkeit kategorisiert
 - bekommen einen Besitzer (siehe "Multitasking")



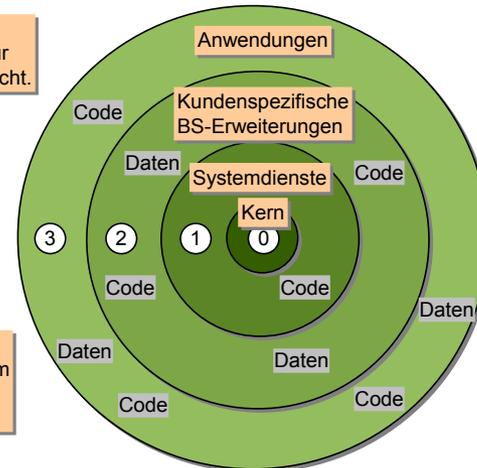
Schutzringe und Gates

Durch einen 2 Bit Eintrag im Segmentdeskriptor wird jedes Segment einer Privileg-Ebene zugeordnet



Schutzringe und Gates

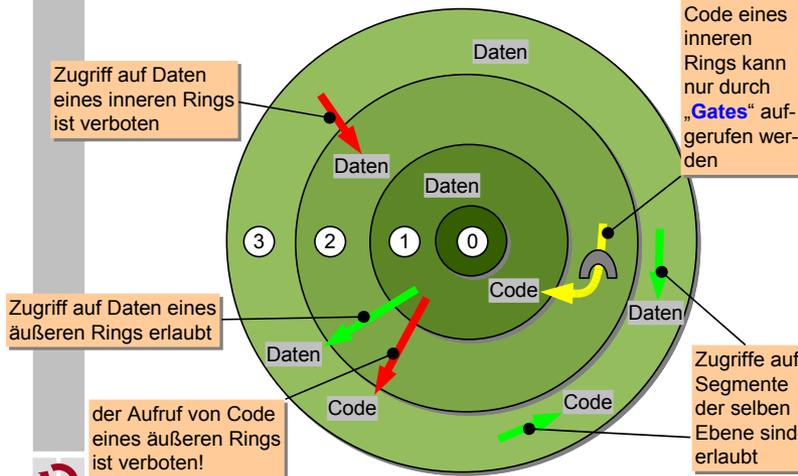
Privileg-Ebene 3 ist die niedrigste und für Anwendungen gedacht.



Privileg-Ebene 0 ist die höchste und dem Betriebssystemkern vorbehalten.



Schutzringe und Gates



BS © 2007, 2008 Daniel Lohmann, Olaf Spinczyk

25

Segmentdeskriptoren

- weitere Informationen über die Schutzanforderungen der Segmente enthalten die Deskriptoren
 - jede Verletzung führt zum Auslösen einer Ausnahme

ein Segment-Deskriptor

| | | | | | | | | | | |
|------------------------|-----|---|------|------------------------|-----|---|-----|-----------------|----|----|
| Segment Base 31 ... 24 | | | | G | D/B | 0 | AVL | Limit 19 ... 16 | +6 | |
| P | DPL | S | TYPE | Segment Base 23 ... 16 | | | | | | +4 |
| Segment Base 15 ... 0 | | | | | | | | | +2 | |
| Segment Limit 15 ... 0 | | | | | | | | | 0 | |

TYPE – Data:
 ED - Expansion Direction
 W - Writable
 A - Accessed

TYPE – Code:
 C - Conforming
 R - Readable
 A - Accessed

P - Present Bit
 DPL - Descriptor Privilege Level
 S - System Segment

G - Granularity
 D/B - 16/32 Bit Seg.
 AVL - Available for OS

BS © 2007, 2008 Daniel Lohmann, Olaf Spinczyk

26

Beispiel: Das "flache" Speichermodell

- die meisten PC Betriebssysteme nutzen die Segmentierung nicht.
 - 32 Bit *Offset* der logischen Adresse = lineare Adresse
 - trotzdem müssen zwei Segmentdeskriptoren angelegt werden:

```

;
; Descriptor-Tabellen
;
gdt:
    dw    0,0,0,0    ; NULL Deskriptor

    dw    0xFFFF    ; 4Gb - (0x100000*0x1000 = 4Gb)
    dw    0x0000    ; base address=0
    dw    0x9A00    ; code read/exec
    dw    0x00CF    ; granularity=4096,
                    ; 386 (+5th nibble of limit)

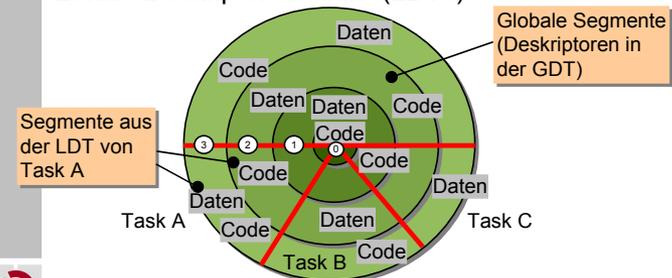
    dw    0xFFFF    ; 4Gb - (0x100000*0x1000 = 4Gb)
    dw    0x0000    ; base address=0
    dw    0x9200    ; data read/write
    dw    0x00CF    ; granularity=4096,
                    ; 386 (+5th nibble of limit)
    
```

BS © 2007, 2008 Daniel Lohmann, Olaf Spinczyk

27

Multitasking

- neben dem Schutz vor unberechtigten "vertikalen" Zugriffen zwischen Segmenten unterschiedlicher Ebenen unterstützt IA-32 auch ein Task-Konzept ("horizontale Trennung")
- die Zuordnung von Segmenten zu Tasks erfolgt über "Lokale Deskriptortabellen" (LDTs)

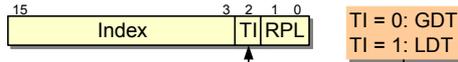


BS © 2007, 2008 Daniel Lohmann, Olaf Spinczyk

28

Lokale Segmentdeskriptortabellen

- ... sind nötig, wenn der Segmentselektor (z.B. aus einem Segmentregister) sich auf die LDT bezieht:



- ... werden mit Hilfe des LDTR gefunden, das bei jedem Taskwechsel ausgetauscht wird:

| Speicherwaltungsregister | | | | |
|--------------------------|----------|------------------|-----------|---|
| | 15 | 0 31 | 0 19 | 0 |
| TR | TSS-Sel. | TSS-Basisadresse | TSS-Limit | |
| LDTR | LDT-Sel. | LDT-Basisadresse | LDT-Limit | |
| IDTR | | IDT-Basisadresse | IDT-Limit | |
| GDTR | | GDT-Basisadresse | GDT-Limit | |



Der Task-Zustand: TSS Segmente

- das Task-Register TR verweist auf eine Datenstruktur, die den kompletten Task-Zustand aufnimmt
- bei einem Task-Wechsel (siehe nächste Seite) wird der komplette Zustand gesichert und der Zustand des Ziel-Tasks geladen

- alles in Hardware!

| I/O Map Base Address | T |
|----------------------|------------------|
| | LDT Segment Sel. |
| | GS |
| | FS |
| | DS |
| | SS |
| | CS |
| | ES |
| | EDI |
| | ESI |
| | EBP |
| | ESP |
| | EBX |
| | EDX |
| | ECX |
| | EAX |
| | EFLAGS |
| | EIP |
| | CR3 (PDBR) |
| | SS2 |
| | ESP2 |
| | SS1 |
| | ESP1 |
| | SS0 |
| | ESP0 |
| | Prev. Task Link |



Task-Wechsel

- für einen Task-Wechsel benötigt man entweder ...
 - ein Task-Gate in der GDT, einer LDT oder der IDT (Task-Wechsel bei Unterbrechungen!)
 - oder einfach nur einen TSS Deskriptor in der GDT
- ausgelöst werden kann ein Wechsel durch ...
 - eine JMP Instruktion
 - eine CALL Instruktion
 - eine Unterbrechung
 - eine IRET Instruktion
- Nested Tasks
 - bei Unterbrechungen und CALLs wird das NT Flag im EFLAGS Register und der "Prev. Task Link" im TSS gesetzt.
 - Wenn dies der Fall ist, springt IRET zum vorherigen Task zurück.



Ein-/Ausgaben im Protected Mode

- nicht jeder beliebige Task darf Ein-/Ausgabe durchführen!
- Zugriffe auf Geräte im Speicher (memory-mapped I/O) können über Speicherschutz abgefangen werden
- Zugriffe auf I/O Ports werden eingeschränkt:
 - die I/O Privilege Level Bits im EFLAGS Register erlauben Ein- und Ausgaben auf bestimmten Schutzringen
 - auf den anderen Ebenen regelt die I/O Permission Bitmap für jeden Task und Port der Zugriff:

Ports mit größeren Nummern dürfen nicht angesprochen werden

den Abschluss bildet immer ein Byte mit 0xff

TSS

| | |
|----------|----------------------------------|
| Port 151 | 11111111001010110100000111010101 |
| | 01101000001110101011011010000011 |
| | 10101010110100000111010101101101 |
| | 00000111010101011010000011101010 |
| | 11010101101000001110100101101101 |

Port 0

| I/O Map Base Address | T |
|----------------------|------------------|
| | LDT Segment Sel. |
| | GS |

eine '1' verhindert den Portzugriff



IA-32: Was gibt es sonst noch?

- Physical Address Extension (PAE)
 - ab Pentium Pro: 36-Bit Adressen (physikalisch)
 - erweiterte *Page Table* Einträge
 - weitere *Page Directory* Ebene
- System Management Mode (SMM)
 - gibt dem BIOS Kontrolle über das System
 - das Betriebssystem merkt davon nichts!
- Virtualisierung der CPU
 - der Virtual 8086 Mode
 - 16 Bit Anwendungen oder Betriebssysteme laufen als IA-32 Task in einer geschützten Umgebung
 - ganz neu: "Vanderpool Technology"
 - Hardwareunterstützung für virtuelle Maschinenlösungen wie VmWare, VirtualPC oder Xen
 - erlaubt die Ausführung von Ebene 0 Protected Mode Code in einer VM



Zusammenfassung

- die IA-32 Architektur ist ausgesprochen komplex
 - segmentbasierter und seitenbasierter Speicherschutz
 - Hardwareunterstützung für Multitasking
 - Task-Aktivierung bei Unterbrechungen
 - Schutz von I/O Ports pro Task
 - ...
- viele dieser Features werden von heutigen Betriebssystemen nicht genutzt
 - typisch ist der flache Adressraum mit aktiver Paging Unit
 - "schwergewichtige Prozesse" sind nicht für jede Anwendung nötig
- bemerkenswert ist auch die konsequente Kompatibilität mit älteren Prozessorversionen
 - Stichwort "PIC" und "A20 Gate"! 😊

