Betriebssystem-Entwicklung

3. Programmiermodell x86-64

Michael Schöttner

Vorschau

Heinrich Heine
Universität Düsseldorf

- Historie
- x86-Programmiermodell
- Adressübersetzung
- Protection
- Tasks
- Zusammenfassung

2.1 Historie

- 8086 (1978) - der Urvater des PC Prozessors
- 80286 (1982) - segmentbasierter Speicherschutz
- 80386 (1985) - Paging mit Schutz auf Seitenebene
- Pentium (1993) - superskalar, 64-Bit Datenbus, MMX, APIC
- Pentium II (1997) - RISC-artige Mikroinstruktionen
- Pentium III (1999) - SSE
- Pentium 4 (2000) - SSE2, Hyperthreading, Intel 64 / EM64T
- Core (2005) - Dual Core
- Core 2 (2006) - Quad Core, 64 bit (x86_64)
- Has-/Broad-well (2013) - AVX2, TSX, FMA3
- Rocket / Tiger Lake (2021) - DL-Boost, memory encryption
- Alder Lake (2021) - Unterteilung in P- und E-cores (max. 8 + 8)

Advanced Vector Extensions (basiert auf SSE)
Transactional Synchronozation Extensions
Fuse-Multiply-Add Instructions

Deep-Learning (DL) Boost
-> Erweiterung von AVX

P = Performance
E = Efficiency
Hybrid Cores

Vorschau

Heinrich Heine
Universität Düsseldorf

- Historie
- x86-Programmiermodell
- Memory Management
- Protection
- Tasks
- Zusammenfassung

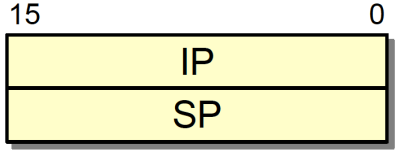
8086

Heinrich Heine
Universität Düsseldorf

- 16-Bit Architektur, little endian
- 20-Bit Adressbus, d.h. maximal 1 MiB Hauptspeicher
- Wenige Register (jedenfalls aus heutiger Sicht)
- 123 Befehle (Befehlslängen von 1 bis 4 Byte)
- Segmentierter Speicher
- Noch immer relevant
  - Bootstrapping von Cores → starten im Real-Mode

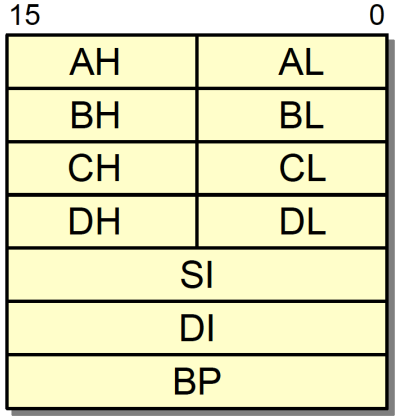
8086: Registersatz

Befehls- und Stapelzeiger

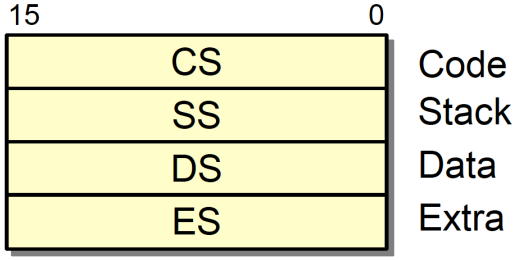


Flag Register

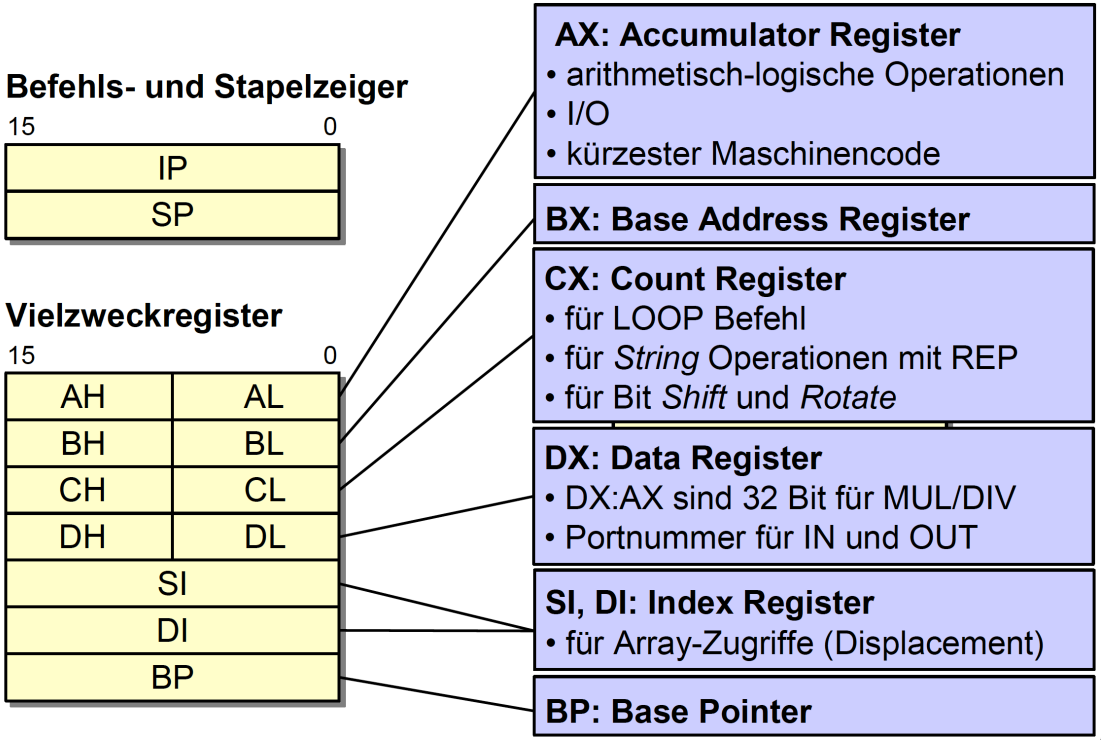
Vielzweckregister



Segmentregister

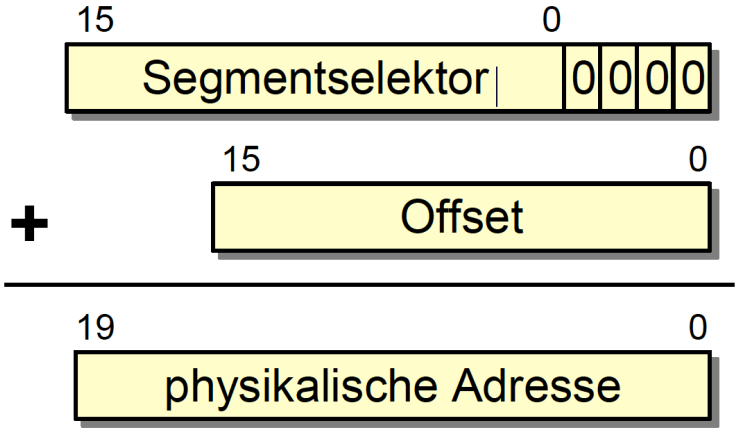


8086: Registersatz



8086: Segmentierter Speicher

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



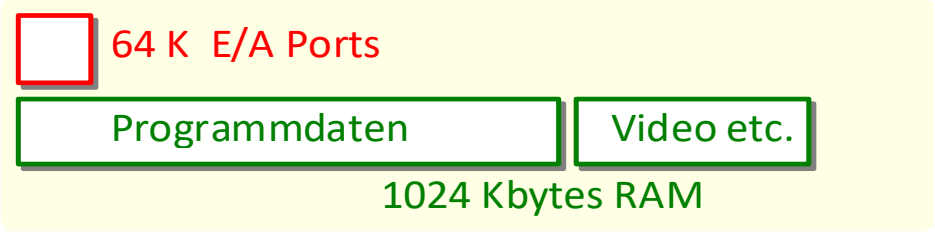
8086: Segmentierter Speicher

8086: Port- vs. Speicheradressen

Heinrich Heine
Universität Düsseldorf

- Port- und Speicher-adressen:
  - Unterscheidung durch CPU-Pin „M/IO“.
  - 16 Adressleitungen für E/A → 16-Bit
  - Heute noch unterstützt (auch im Protected und Long-Mode)

- Zusätzlich: Memory Mapped E/A (z.B. für Videoadapter)
  - Speicherzugriffe gehen nicht ins RAM, sondern zu einem Gerät



IA-32 - Die 32 Bit Intel Architektur

Heinrich Heine
Universität Düsseldorf

- Die erste IA-32 CPU war der Intel 80386
  - Der Begriff „IA-32“ wurde aber erst sehr viel später eingeführt
- 32 Bit Technologie: Register, Daten- und Adressbus
  - Ab Pentium Pro: 64 Bit Daten und 36 Bit Adressbus
- Komplexe Schutz- und Multitasking-Unterstützung
  - 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 (meist aber als Flat-Memory verwendet)
- Seitenbasierte Memory Management Unit (MMU)

x86-64: 64-Bit Prozessor-Architektur

Heinrich Heine
Universität Düsseldorf

- x86-64: ursprünglicher Name für 64-Bit Erweiterung von AMD
  - Wurde manchmal auch abgekürzt als x64
  - Später ersetzt durch Begriff AMD64
- x86-64 hat Intel von AMD lizensiert
- Intel 64 ist Intels Gegenbegriff (für die gleiche Architektur)
  - minimale Unterschiede im Befehlssatz zu AMD64
  - aber komplett anders als die eigene IA-64 (Architektur des Itanium)

x86-64: 64-Bit Prozessor-Architektur

Heinrich Heine
Universität Düsseldorf

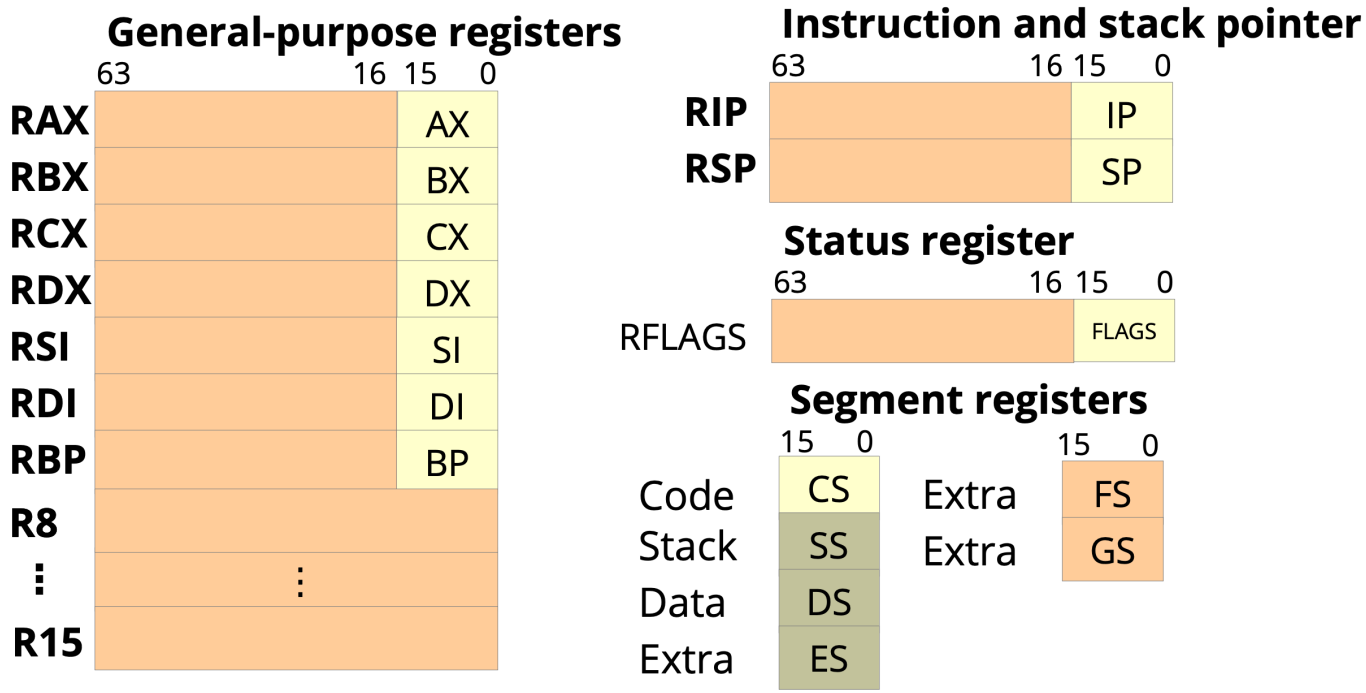
- Es gibt mehr Register und diese haben 64 Bit
- Paging ist zwingend notwendig
- Segmentierung stark eingeschränkt
- Neu u.a.: „no execute“ Schutz auf Seitenebene

x86-64: CPU-Arbeitsmodi

Heinrich Heine
Universität Düsseldorf

- Real mode = 16 Bit Adressierung
- Protected mode = 32 Bit Adressierung, auch IA-32
- Long mode = 64 Bit Adressierung, auch IA-32e
  - e steht für extension
  - Verwendet in den Manuals von Intel
  - Zur Abgrenzung gegenüber dem IA-64
- (compatibility mode)
  - Unterstützung von Protected Mode Anwendungen im Long Mode

x86-64: Registersatz



x86-64: Registersatz (2)

Heinrich Heine Universität Düsseldorf

Memory-management registers

	15 0	63 0	31 0
TR	TSS sel.	TSS Base Address	TSS Limit
LDTR	LDT sel.	LDT Base Address	LDT Limit
IDTR		IDT Base Address	IDT Limit
GDTR		GDT Base Address	GDT Limit

Details follow ...

Control registers

	63 32	31 0
CR8		
CR4		
CR3		
CR2		
CR0		

Debug registers

	63 32	31 0
DR0		
DR1		
DR2		
DR3		
DR6		
DR7		

Model-specific registers (MSRs)

Vorschau

Heinrich Heine
Universität Düsseldorf

- Historie
- x86-Programmiermodell
- Adressübersetzung
- Protection
- Tasks
- Zusammenfassung

x86-64: Adressierung

Heinrich Heine Universität Düsseldorf

- Effektive Adressen (EA) werden nach folgendem Schema gebildet
  - Alle Vielzweckregister können dabei gleichwertig verwendet werden

EA := Base-Reg. + (Index-Reg. * Scale) + Displacement

1/2/4/8

1/2/4 bytes

- Beispiel: `MOV RAX, array[RSI * 4]` (EA)
  - Lesen aus einem Array mit 4 Byte großen Elementen und RSI als Index
- Neu seit x86-64: IP-relative Adressierung

EA := RIP + Displacement

Protected Mode – Segmente

Heinrich Heine
Universität Düsseldorf

- Ein Segment hat eine Startadresse und eine Länge
- Segment-Register speichern Segment-Selektoren = Index in die GDT/LDT
  - LDT wurde von den meisten Betriebssystemen nicht genutzt
- Ein Programm (in Ausführung) besteht aus mehreren Speichersegmenten
  - mindestens CODE, DATEN und STACK
- „Lineare Adresse“ ist Segmentstartadresse + Effektive Adresse
  - Effektive Adresse = Wert in einem Vielzweckregister
  - „Lineare Adresse“ entspricht der physikalischen Adresse, falls die Paging Unit nicht eingeschaltet ist.

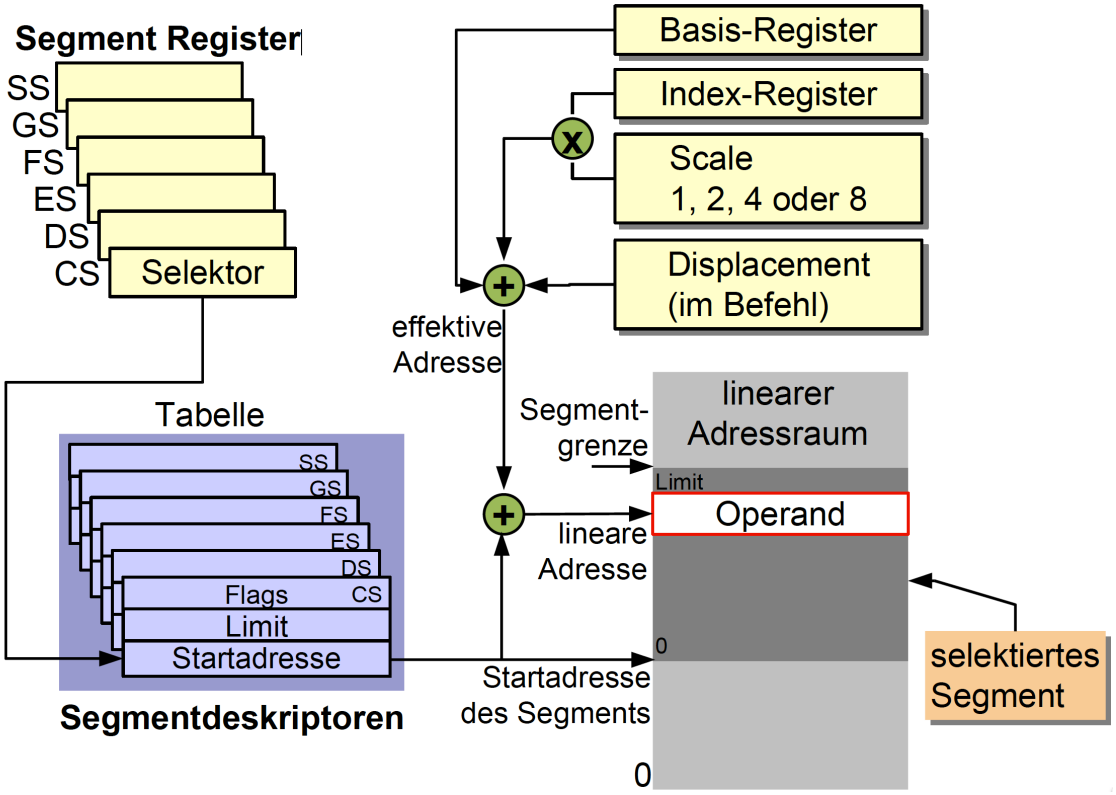
Protected Mode – Segmente (2)

Heinrich Heine
Universität Düsseldorf

- Flat memory:
  - Fast alle moderne 32 Bit Betriebssysteme haben alle Segmente auf Startadresse 0 gesetzt
  - Dadurch beeinflussen Segmente nicht die Adressberechnung
  - Zudem wurden im Prinzip nur vier Einträge in der GDT verwendet (Code+Daten, jeweils für den User- und Kernel-Mode); LDT daher gar nicht genutzt

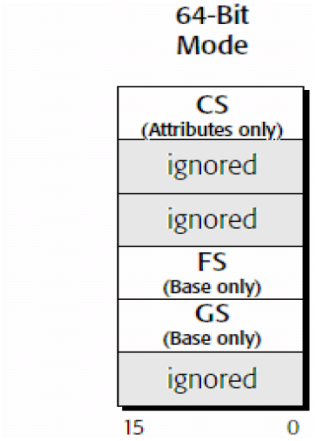
- Segmentierung ist nicht abschaltbar (wegen dem Schutz), jedoch muss das Paging nicht zwingend für den I-A32 Protected Mode aktiviert werden

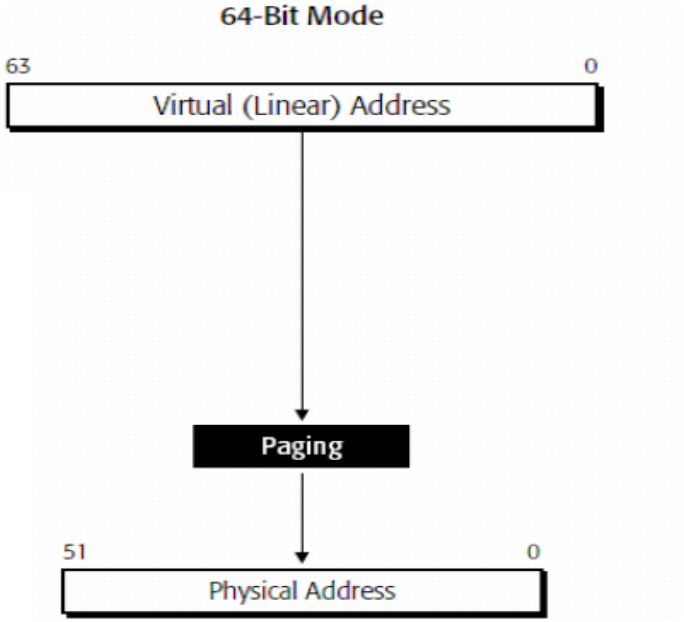
Protected Mode – Segmente (3)



Long Mode: Segmentierung

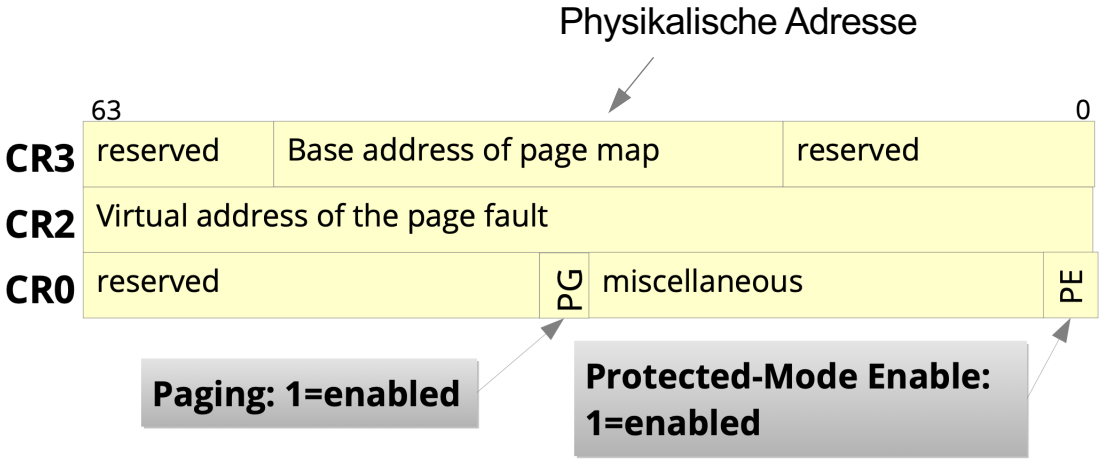
- 16-Bit Selektoren
- Startadressen werden bei fast allen Segmenten ignoriert, außer FS und GS
- Keine Limit-Prüfung; im Wesentlichen ist nur noch die Ring-Nummer in CS relevant = CPL





Long Mode: Paging

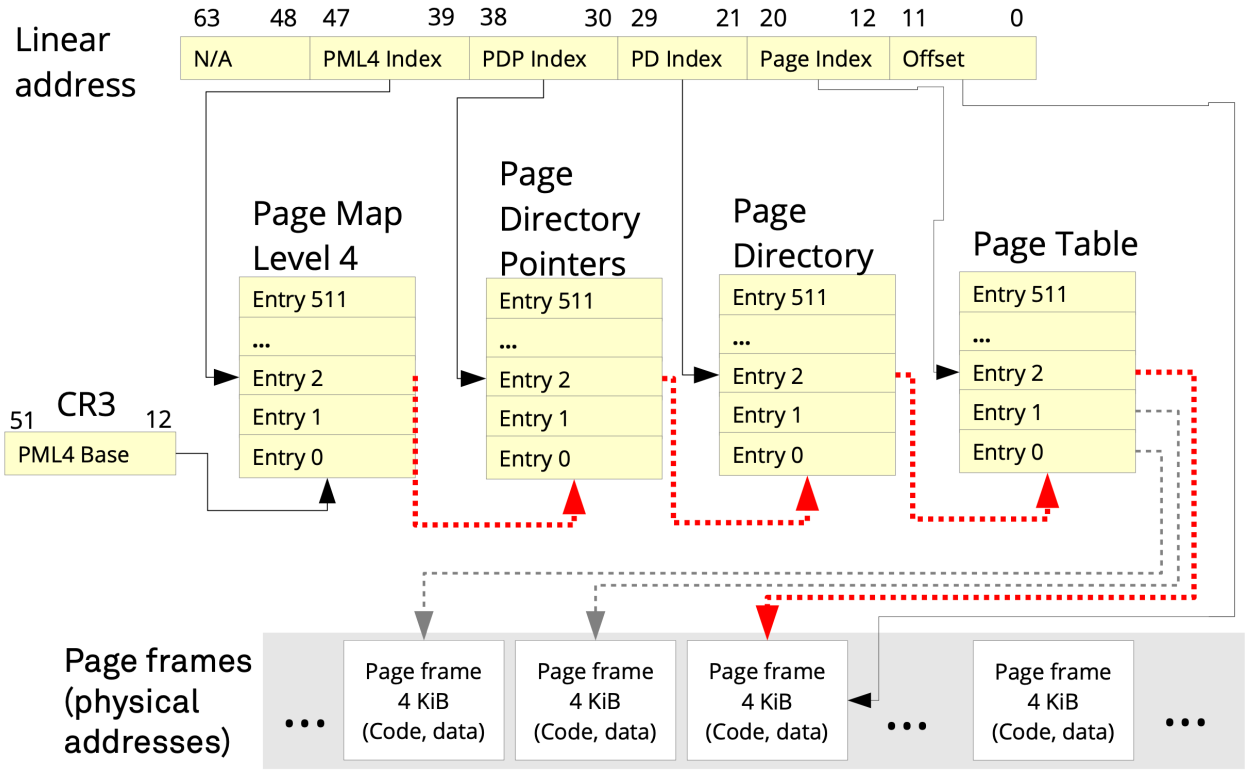
- Paging ist für den Long Mode zwingend erforderlich
- Die wichtigsten Verwaltungsinformationen für das Paging stehen in den CRx Steuerregistern (CR = Control Register):



Long Mode: Page Tables

Heinrich Heine Universität Düsseldorf

- Vier oder fünf-stufiges Paging



Long Mode: TLB

Heinrich Heine
Universität Düsseldorf

- Problem: bei aktivem Paging CPU erheblich langsamer, da bei jedem Speicherzugriff die Seitentabellen mehrfach zugegriffen werden müssten
- Lösung: der Translation Lookaside Buffer (TLB):
  - vollassoziativer Cache
  - Tag: bestehend aus Page-Table-Indizes
  - Daten: Page Frame Adresse
  - Größe beim 80386: 32 Einträge
- Bei normalen Anwendungen erreicht der TLB eine Trefferrate von etwa 98%
- Bemerkungen
  - Schreiben in das CR3 Register invalidiert den TLB
    - Nicht mehr seit Intel Westmere (2010) → TLB Tag hat zusätzl. 12-bit Process-Context ID (PCID); eingeführt werden Meltdown & KPTI
  - Falls Schutzbits oder Access-Bits in einem Page-Table-Eingtrag geändert werden, muss auch der TLB respektive der betroffene TLB-Eintrag gelöscht werden!

Vorschau

Heinrich Heine
Universität Düsseldorf

- Historie
- x86-Programmiermodell
- Adressübersetzung
- Protection
- Tasks
- Zusammenfassung

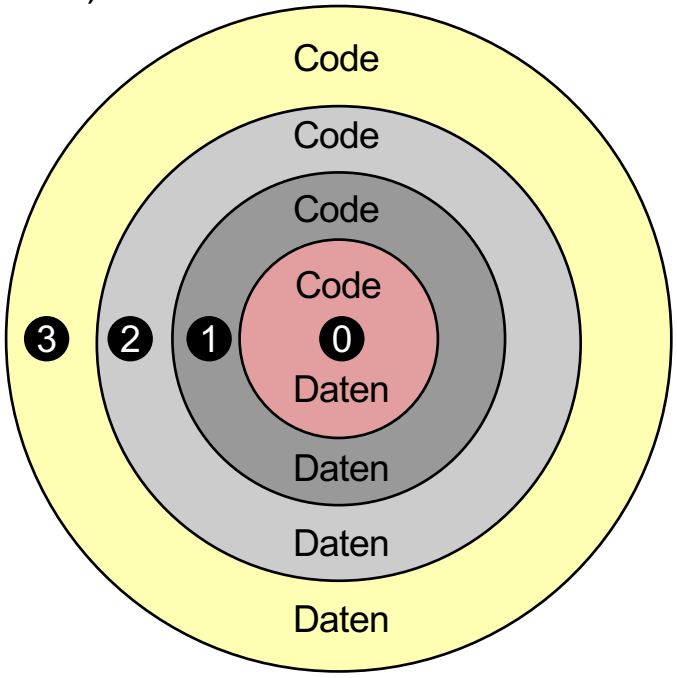
Protection

Heinrich Heine
Universität Düsseldorf

- Ziele:
  - Schutz vor unberechtigten Datenzugriffen oder Funktionsaufrufen
  - Schutz vor Absturz des Kernels respektive anderer Prozesse
- Voraussetzungen: Code und Daten ...
  - Werden hinsichtlich der Vertrauenswürdigkeit kategorisiert
  - Und diese Einteilung wird vom Betriebssystem mithilfe des Prozessors durchgesetzt

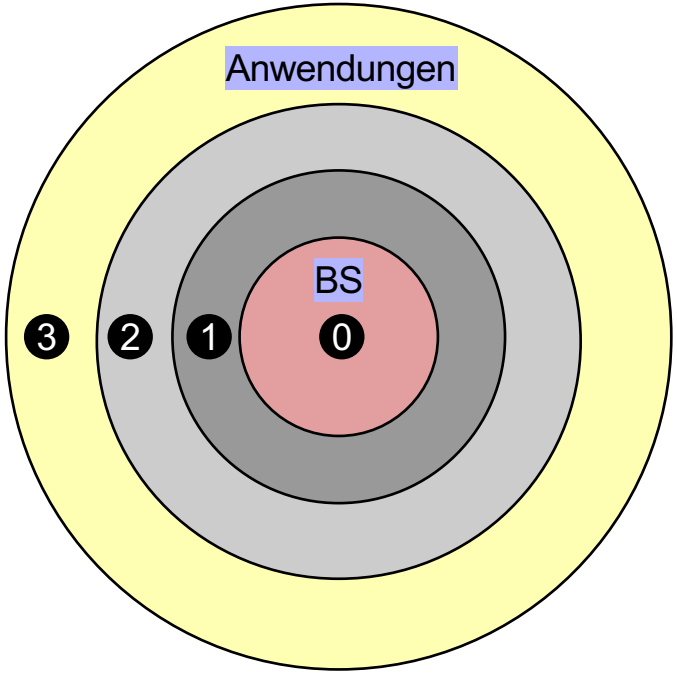
Schutzringe

- Jeder Segmentdeskriptor (Code / Daten) hat 2 Bit, welche anzeigen welcher Privilegstufe das Segment zugeordnet ist
  - Hierdurch sind max. vier Privilegstufen mögl.
  - Genutzt werden in der Praxis nur zwei



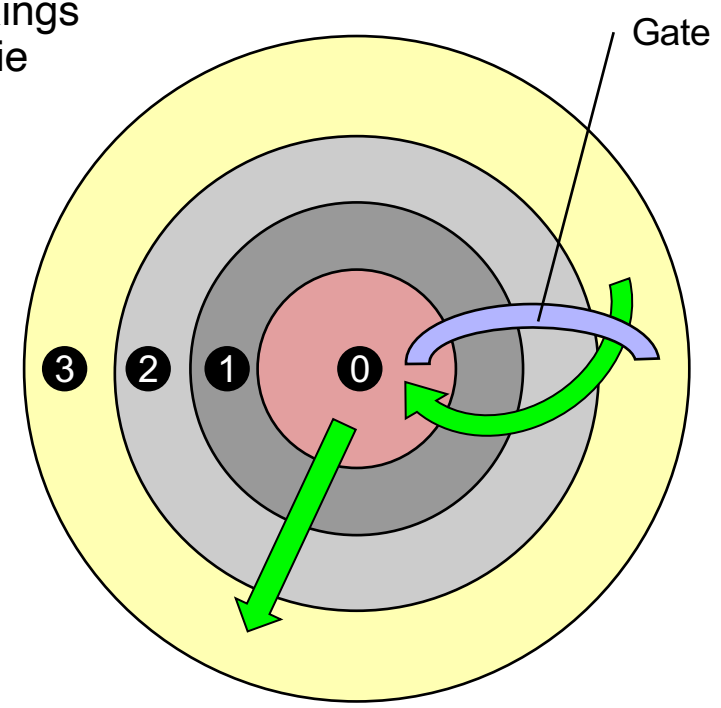
Schutzringe

- Je kleiner die Nummer der Privilegstufe ist, desto höher die Berechtigung
- Stufe 3 ist für Anwendungen
  - = User-Mode
- Stufe 0 ist für das Betriebssystem
  - = Kernel-/Supervisor-Mode



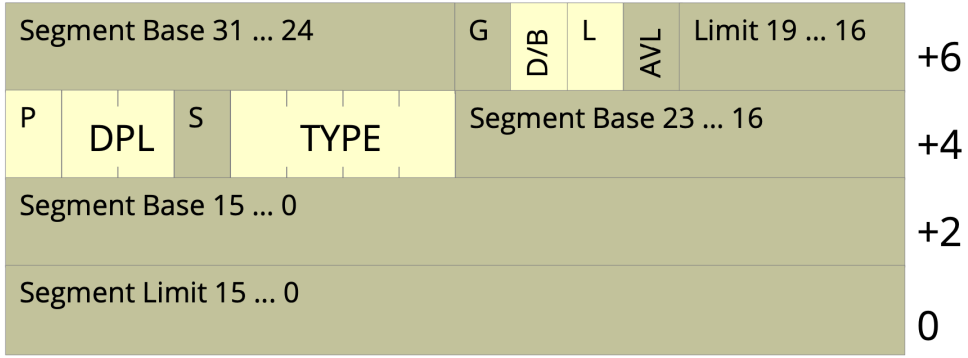
Schutzringe und Gates

- Zugriffe auf Daten eines inneren Rings sind verboten und werden durch die MMU verhindert
  - General Protection Fault
- Code eines inneren Rings kann nur mithilfe eines **Gates** aufgerufen werden
  - Hiermit sind direkte Aufrufe von Ring 3 nach Ring 0 möglich
- Zugriffe / Aufrufe auf äußere Segmente sind erlaubt



Segment-Deskriptoren

- Stehen in der Global Descriptor Table (GDT) und erlauben Schutz von Code
  - (Bei IA32 zusätzl. Schutz über das Limit (=Ende des Segments))
  - Pro Eintrag 8 Byte (Ausnahme: Gate bei 64 Bit hat 16 Byte)



TYPE
Code/Data
Read/Write
Non-/Conforming

System-Segmente (Call-Gate, Task State Segment)

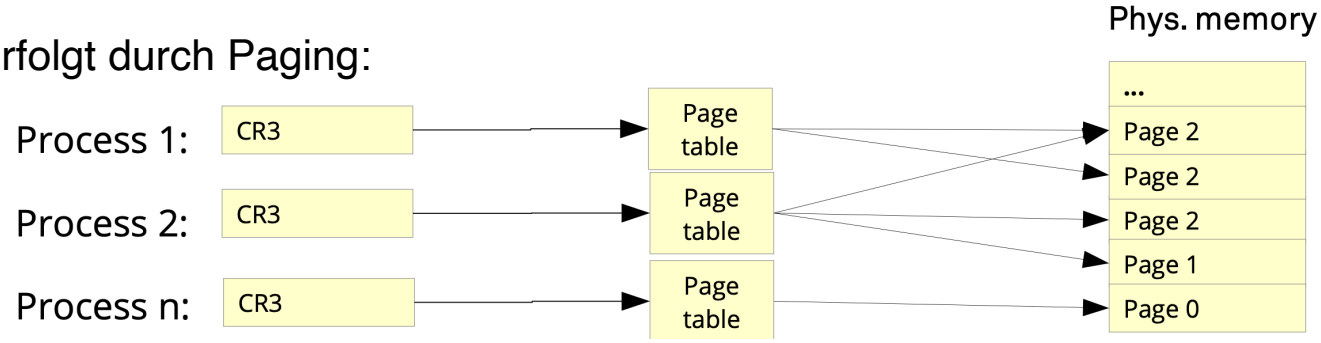
P	- Present Bit
DPL	- Descriptor Privilege Level
S	- System Segment

G	- Granularity
D/B	- 16/32 Bit Seg.
L	- Long Mode aktiv

Flat Memory Model

- Die meisten 32 Bit PC Betriebssysteme haben die Segmentierung nicht genutzt
  - 32 Bit effektive Adresse = lineare Adresse
- Bei 64 Bit wird Segment-Basisadresse durch den Prozessor nicht mehr beachtet

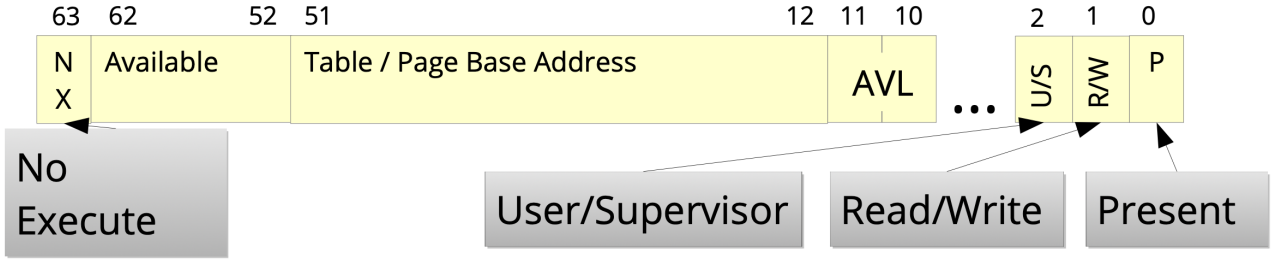
- Schutz erfolgt durch Paging:



- Jeder Prozess sieht nur seinen eigenen virtuellen Adressraum
  - Weitere Zugriffskontrolle auf Seitenebene (siehe nächste Seite)

Seitentabellen-Eintrag

- Einstellungen sind auf allen Ebenen der Page-Table-Hierarchie möglich
- Schutzbits:
  - R/W = Read/Write → Schutz vor Schreibzugriffen
  - NX = NoeXecute → keinen Code in Daten ausführen
  - U/S = User/Supervisor → Zugriff für alle oder nur im Ring 0



  - Bem.: Available = AVL = für das Betriebssystem frei verwendbare Bits

x86_64: Zusammenfassung - Schutz

Heinrich Heine
Universität Düsseldorf

- In modernen 64 Bit Betriebssystemen reichen folgende Einträge in der GDT:
  - ein Daten-Segmentdeskriptor
  - zwei Code-Segmentdeskriptoren (einer mit DPL = 0 und der andere mit DPL = 3)

- Schutz wird durch die Seitentabellen des Pagings (zwingend bei 64 Bit) durchgesetzt
  - Sowohl durch die Bits (NX, U/S, R/W)
  - Als auch die Seitentabellen:
    - Welche Seiten sind genutzt respektive präsent oder ausgelagert

Vorschau

Heinrich Heine
Universität Düsseldorf

- Historie
- x86-Programmiermodell
- Adressübersetzung
- Protection
- Tasks
- Zusammenfassung

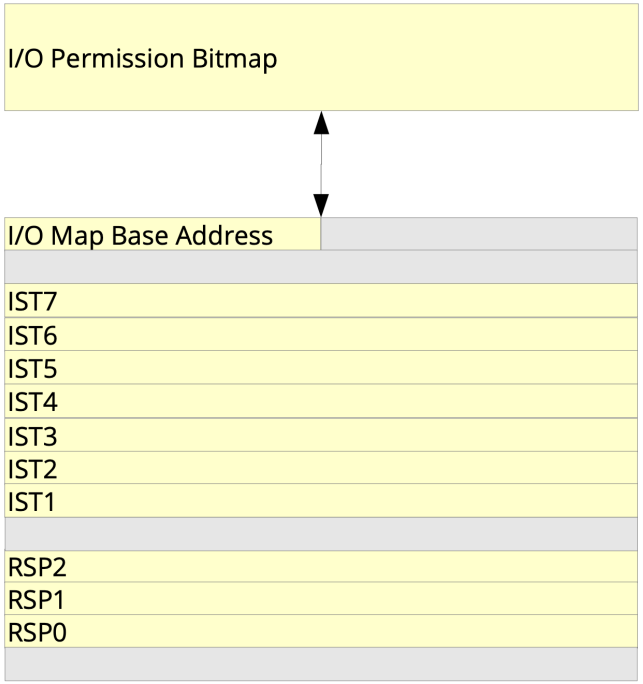
IA-32: HW-Unterstützung für Multitasking

Heinrich Heine
Universität Düsseldorf

- Der Zustand einer Task wird in einem Task State Segment (TSS) gesichert
  - Speichert alle Register
  - Stack-Pointer für jeden Ring
  - CR3-Register
  - I/O-Permission-Bitmap (siehe später)
- Aktuelles TSS steht im Task Register (TR) des jeweiligen Kerns
  - Pro Core gibt es ein TR
- Task-Wechsel mit HW-Unterstützung über ein Task-Gate
  - Wurde von den meisten Betriebssystemen nicht genutzt und stattdessen in Software realisiert

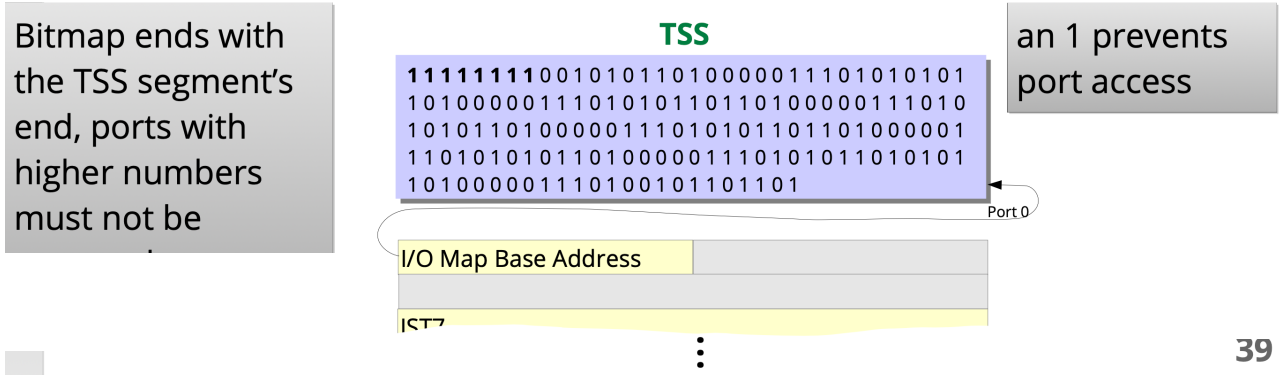
Long Mode: Multitasking

- Keine HW-Unterstützung für Task-Wechsel
- TSS vorhanden, speichert aber nur noch
  - Stack-Pointer pro Ring
  - IST = Interrupt Stack Table
  - I/O-Permission-Bitmap (nächste Seite)
- IST erlaubt separate Stacks für einen Interrupt-Handler
  - IST-Nummer kann in einem Eintrag in der Interrupt-Descriptor Table (IDT) angegeben werden



Long Mode: I/O-Permission-Bitmap

- Zugriffsschutz für Port-Zugriffe (nicht mehr so wichtig heute)
  - I/O Privilege Level (IOPL) der aktuellen Task steht in zwei Bits im RFLAGS-Register
  - Portzugriff ist erlaubt, falls CPL ≤ IOPL
  - Andernfalls wird die I/O-Permission-Bitmap verwendet
    - Je ein Bit pro Port; falls Bit gesetzt ist, so ist der Zugriff nicht erlaubt
- Zugriffsschutz für Memory-Mapped I/O über die Seitentabelle



Vorschau

Heinrich Heine
Universität Düsseldorf

- Historie
- x86-Programmiermodell
- Adressübersetzung
- Protection
- Tasks
- Zusammenfassung

Zusammenfassung

Heinrich Heine
Universität Düsseldorf

- x86-64 Architektur ist sehr komplex
  - Virtueller Speicher mit vier oder fünf-stufigem Paging
  - Seitenbasierter Speicherschutz
  - Port-Zugriffsschutz pro Task
  - Kann 16-Bit Code im IA-32 ausführen oder 32-Bit Code im Long Mode
- Selten genutzte Eigenschaften wurden entfernt
  - Segmentierung
  - Task-Umschaltung in Hardware
- Aber immer noch rückwärtskompatibel
  - Prozessor respektive Cores startet/n immer noch im Real Mode

Was gibt es noch?

Heinrich Heine
Universität Düsseldorf

- Virtualisierung
  - Virtual 8086 Mode
    - 16 Bit Anwendungen
  - Intel-VT, AMD-V
    - HW-Unterstützung für VMware, VirtualBox, etc.
    - Der Hypervisor läuft in Ring -1
- AVX: Advanced Vector Extensions
- TSX: Transactional Synchronozation Extensions
- Hybride Cores