1 Entwicklung der Rechensysteme

2 Befehlssatzarchitektur

3 Programmerstellung und Programmiermodelle

4 Ein einfaches Rechnermodell

5 Befehlsabarbeitung

6 Speichertechnologie

7 Ein-/Ausgabe

8 Bussysteme

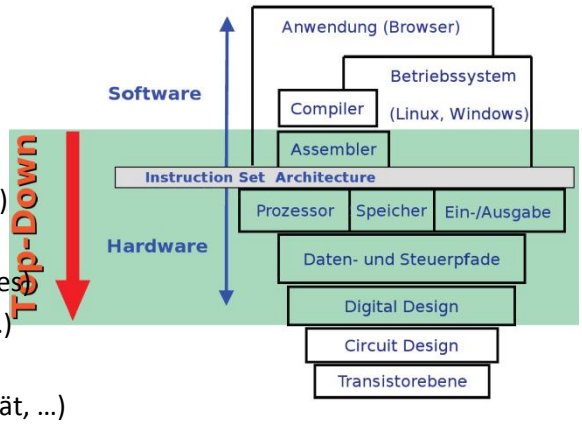
9 Kodierung von Informationen

10 Betriebssystemaspekte I

11 Betriebssystemaspekte II

1 Entwicklung der Rechensysteme

Beschreibungsebenen eines Rechensystems



- Problemstellung (Web-Browser, Datenbank, …)
- Algorithmen (Textdarstellung, Suchen, Sortieren, …)
- Hochsprache (C, C++, Java, Python, …)
- Maschinensprache (Assembler-Mnemonics, Opcodes)
- Funktionsgruppen (ALU, Sprungeinheit, Speicher, …)
- Gatter (UND, ODER, NICHT, …)
- Elektronische Bauteile (Halbleiter, Resistor, Kapazität, …)
- Physikalische Ebene (Chipmasken)

2 Befehlssatzarchitektur

Die CPU aus Programmierersicht	Die CPU aus Compilersicht
- **Ziel: Einfachheit der Programmierung** - Art und Anzahl der Befehle - Art und Anzahl unterstützter Datentypen - Anzahl und Verwendung der Register	- **Ziel: Einfachheit der Programmabbildung** - Orthogonaler, regulärer Befehlssatz - Vielzahl von Registern - Elementare Operationen und Datentypen ausreichend

Datentypen:

- Native: (primitive Typen, Software zur Emulation von nicht-nativen zu nativen Typen)
- Spezifikation von Wertebereichen (Konstanten, Ausdrücke, Variablen, Funktionen)
- Festgelegte Interpretation (erkennbar durch Datentyp)
- z.B. Bit, Nibble (4 Bit), Byte (8 Bit), word (16 Bit), double word (32 Bit), quad word (64 Bit)

Wandlung des Dezimalsystems mit Euklid oder Horner-Schema (übliche)

Abstrahierte Datentypen:

- Bitvektor: Allgemeine Gruppierung von Bits in größere Speicherworte
- Bitfeld: Offset-basierter Zugriff auf eine variable Anzahl konsekutiv im Speicher abgelegter Bits
- Vorzeichenlose Dualzahl
- Vorzeichenbehaftete Dualzahl
- BCD-Ziffern in gepackter Form
- BCD-Ziffern in ungepackter Form
- Gleitkommaformat: einfache und doppelte Genauigkeit

Zahldarstellung:

- Offset-Dualdarstellung
- Darstellung mit Betrag und Vorzeichen
- Einerkomplementarstellung
- Zweierkomplementarstellung

	Einerkomplement	Zweierkomplement
Komplementbildung	Invertieren aller Stellen	Invertieren aller Stellen gefolgt von +1
Null-Darstellung	nicht eindeutig	eindeutig
Zahlenbereich	$[-2^{n-1} – 1, 2^{n-1} -1]$	$[-2^{n-1}, 2^{n-1} -1]$
Bedingter Korrekturschritt bei Addition	ja	nein
Behandlung von vorzeichenlosen und –behafteten Zahlen	unterschiedlich (Korrekturschritt)	einheitlich

Kommadarstellungen: (Folie 23)

- Festkommadarstellung: Feste Länge des ganz und Kommazahlbereichs
- Gleitkommadarstellung: (Zahl wird aufgeteilt in Vorzeichen, Exponent, Mantisse)
  - Genauigkeit variiert mit Größe des Exponenten. Exponent skaliert die Mantisse. Je größer die Zahl, desto grober die Mantisse
  - $X = (-1)^s * a * 2^E = (-1)^s * 2^{(e-B)} * (1 + f)$
  - IEEE 754: Null: e = 0, f = 0; Unendlich: e = 1…1 und f=0…0; No Number: e=1..1, f≠0..0

Ausführungsmodelle (Folie 33)

- Kellermodell: Stack auf dem Operationen ausgeführt werden (Java Bytecode)
- Akkumulator/Speicher-Modell: Ein Register (acc) in dem gerechnet wird. (ältere 8-Bit-Prozessoren) (Z = X+Y: **LDA** X; **ADD** Y; **STA** Z)
- Register/Speicher-Modell: verallgemeinerte Form des Akk/Speicher-Modells (Intelx86) (Z=X+Y: **MOV** X, r1; **ADD** r1, Y; **MOV** r1, A)
- Speicher/Speicher-Modell: alle Operanden stehen im Hauptspeicher (Z=X+Y: **ADD** Z, X, Y)
- Register/Register-Modell: alle Operanden stehen im Register (RISC-Architekturen) (Z=X+Y: **LD** r1, X; **LD** r2, Y; **ADD** r3, r1, r2; **ST** r3, Z)

Einfache Adressierungsarten (Folie 39)

- implicit: Operand steht implizit fest
- immediate: Direkte Angabe einer Konstante
- absolute: Direkte Angabe einer Zugriffsadresse (Speicher)
- register: Direkte Angabe eines Zielregisters
- indicrect adressing: Übergabe eines Zeigers auf eine Struktur im Speicher
- indexed adressing: Angabe von Basisadresse und Index (Tabellen, Felddurchlauf)
  Prädekrement: auto-verminderung vor Auswertung
  Postinkrement: auto-verminderung nach Auswertung
- relative adressing: Bildung der effektiven Adresse relativ zum aktuellen Befehlszählerstand (bedingte Sprünge, typisch kurze Distanz bei Schleifen/Fallunterscheidungen)

3 Programmerstellung und Programmiermodelle

Anforderungen des Compilers:

- Regularität: alle Register für alle Operationen
- Orthogonalität: Unabhängigkeit von Adressierungsarten und Operationen
- Allgemein verwendbare Basisbefehle
- Begrenzung des Suchraums erwünscht
- 10-20% aller Befehle werden von Compilern 80-90% verwendet
- 10-20% aller Befehle sind für 80-90% der Laufzeit verantwortlich

Programmiermodelle:

- Complex Instruction Set Computer (CISC)
  Fokus auf Codedichte und Assemblerprogrammierer (z.B. Z80)
- Reduced Instruction-Set Computer (RISC)
  Einheitliches Befehlsformt, Befehlsgröße, Strikte Trennung von Operationen (z.B. DLX, ARM)

DLX-Prozessor:

- Abarbeitung des Befehls in 5 Stufen
  - Befehlsholphase (Instruction Fetch, IF)
  - Befehlsdekodierphase (Instruction Decode, ID)
  - Ausführungsphase (Execution, EX)
  - Speicherzugriff (Memory Access, MA)
  - Rückschreibphase (Writeback, WB)
- Überlappte Ausführung der Befehle möglich
- Assembler Folie 10

Vergleich früher und heute

Früher: Anwendungserstellung auf Maschinensprache	**Heute: Hochsprachliche Anwendungserstellung**
vergleichsweise einfache Architekturen (kein Cache, kein Piplining, keine Sprungvorhersage) Extreme Ressourcenbegrenzung Hohe Spezialisierung der Software	Anwendungs- und Architekturkomplexität Verbesserung der Compilertechnologie Compiler optimiert hinsichtlich Architektur und Organisation (Cache, Pipeline, Sprungvorhersage)

Begriffsdefinition

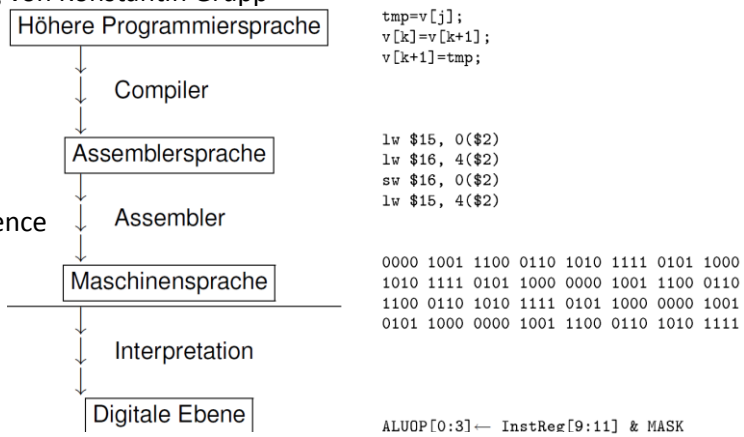
- **Assembler** übersetzt Quellcode in Maschinensprache
- **Objektcode**: Repräsentation eines Maschinenprogramms mit noch unaufgelösten Referenzen
- **Linker** oder **Binder** löst Referenzen auf und verbindet ggf. mehrere Objektcode-Module zu ausführbarem Programm

Vorteile der Hochsprachen Programmierung

- Hochsprache erlaubt einfachere Programmierung mit kürzeren Entwicklungszeiten
- Compiler erzeugen hocheffizienten Code

Zeiger und Vektoren

- Abbildung durch Compiler
  - Direkte Zuweisung falls call by value
  - Indirekte Zweisung falls call by reference



Struktur von Compilern

Ebene	Abhängigkeiten	Funktionen
Front-End	sprachabhängig, maschinenunabhängig	Transformation der Quellsprache in (gemeinsame) Zwischensprache
High-level-Optimierung	leicht sprachabhängig, weitg. maschinenunabh.	Schleifentransformation, Prozedur-Inlining etc.
Code-optimierung	weitg. sprachunabh., leicht maschinenabh.	Globale und lokale Optimierungen, Registerallokation
Code-generierung	sprachunabhängig, maschinenabhängig	Transformation in geeignete Maschinenbefehle, Optimierung

↑ zunehmende Sprachabh. ↑

↓ zunehmende Maschinenabh. ↓

Compileroptimierungen

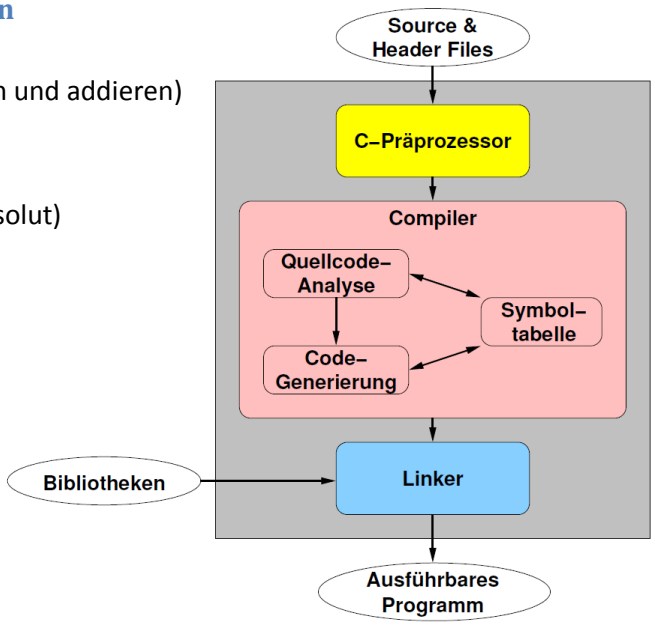
- **Optimierungen auf oberer Ebene**: Vorwiegend Inline-Expansion von Prozeduren und Funktionen
- **Globale Optimierungen**: Eliminierung gemeinsamer Teilausdrücke, Codeverschiebung (Herauslösen von schleifen und verzweigungsunabhängigen Code)
- **Lokale Optimierungen**: Berechnung und Propagierung konstanter Ausdrücke, Umstellung der Berechnungsreihenfolge zur Einsparung von Registern („height reduction“)

Maschinenabhängige Compileroptimierungen

- Vereinfachung von Operationen
  (Multiplikation mit Konstanten → schieben und addieren)
- Pipeline-Scheduling
- Optimierung der Sprungbefehlsadressen
  (Wahl geeigneter Instruktion relativ vs. absolut)

Anforderungen an Compiler-Infrastruktur

- **Unterstützung der Zielplattform**
  - Basisbibliotheken
  - normierte Datentypen
- **Unterstützung der Zielhardware**
  - Optimierung
  - Assembler-Generierung
  - Assemblierung und Binden



4 Ein einfaches Rechnermodell

Aufteilung des Rechners in unabhängige Teilwerke

- Zentraleinheit (Prozessor)
    - Steuerwerk (Interpretation der Programmvorschrift)
        - Statisch: Speicherung in FF, Zustand stabil solange Strom anliegt
        - Dynamisch: Speicherung in Kondensatoren, Mindest-Taktfrequenz benötigt
    - Rechenwerk (Durchführung arithmetisch/logischer Operationen)
- Speicherwerk (Speicherzelle eindeutig durch Adresse identifizierbar)
- Ein-/Ausgabewerk (USB, SCSI)
- Verbindungsstrukturen (Adressleitungen, Datenleitungen, Steuerleitungen, Systembus)

Phasen der Befehlsausführung (Folie 11)

- Holphase: Laden des nächsten Befehls
- Dekodierphase: Dekodierung des Befehls
- Ausführungsphase: Durchführung des Befehls

Befehlsregister (Folie 12)

- Register mit unterschiedlich langen Befehlsformaten (CISC)
- Vorabladen von Befehlen (Opcode Prefetching)

Ablaufsteuerung

- Festverdrahtet, z.B. durch Zustandsautomaten in PLA
- Mikroprogrammiert: Pro Assemblerbefehl ein Mikroprogramm im Festwertspeicher

Rechenwerk mit Akkumulator (Folie 16)

- Vorteile:
    - Höhere Schaltgeschwindigkeiten
    - Enge Kapselung der Alu
- Nachteile:
    - Rechnung nur auf ausgezeichneten Registern
    - Zwischenberechnungen benötigen mehrfaches Ein- und Ausladen

  ➔ Heute kaum mehr verwendet, typisch für alte Architekturen

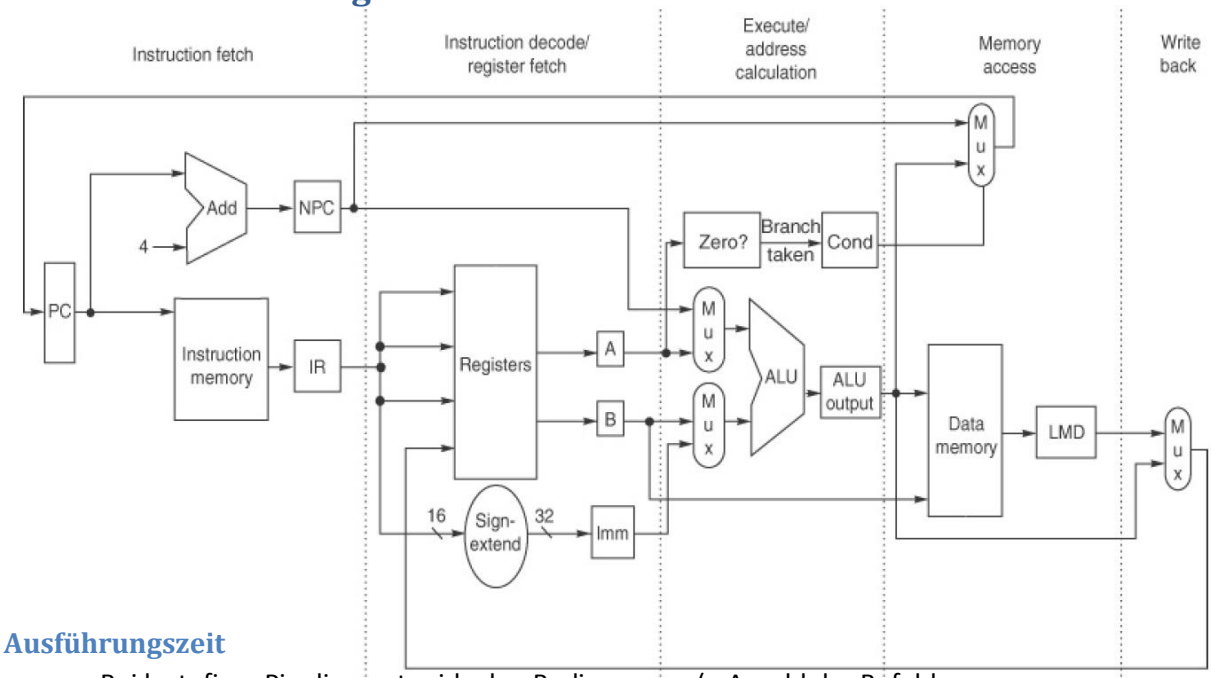
Statusregister (Folie 22)

- Aussage über Rechenergebnis
- Signalisierung von Carry Flag, Zero Flag, Sign Flag, Overflow Flag
- je nach Architektur auch Signalisierung von Decimal Flag, Half Carry, Parity Flag, Even Flag

Registersatz (Folie 24)

- Problemstellung: Schneller Zugriff auf Operanden, Hauptspeicher zu langsam
- Speicherung häufig benutzter Operanden in Registern
- Nutzung von Multiport-Speichern (Gleichzeitiges Lesen und Schreiben)
- Üblich bei fester Nutzungszuordnung: R_0 ist null, R_{15} ist PC, R_{14} ist SP
- Früher wurde Daten- und Adressregister getrennt heute meist universelle Verwendung

5 Befehlsabarbeitung



Ausführungszeit

- Bei k-stufiger Pipeline unter idealen Bedingungen (n Anzahl der Befehle,
  - Programmlaufzeit (optimal): T = n + k – 1 (n Anzahl der Befehle, k Ausführung in Takten)
  - Beschleunigung: S = n * k / T

Verfeinerung von Pipelines

- Ausführungsphase ggf. inhomogen
- Integerverarbeitung (typ. 1 Zyklus, Ausnahme: Division)
- Gleitkommaverarbeitung (Zerlegung der Ausführung in weitere Stufen
- Verwendung mehrerer Rechenwerke (Spezialisierung einzelner Rechenwerke)

Abhängigkeiten und Konflikte (Folie 16)

- **Abhängigkeiten sind Eigenschaften des Programmcodes**
  Benutzung gleicher Ressourcen, Datenfluss
- **Konflikte sind Auswirkungen der Abhängigkeiten** bezogen auf Technologie
  Aufbau von Pipeline und weiterer Mikroarchitektur, Befehlsdistanz
- **Konflikt bewirkt Unterbrechung des taktsynchronen Befehlsdurchlaufs** durch Pipeline
  - Anhalten der Pipeline an blockierender Stelle
  - Leerlaufen der restlichen Pipeline
  - Minderung des Durchsatzes

Strukturkonflikte (structural hazard) (Folie 17)

- Ursache in Ressourcenbeschränkungen
- Summe aller in der Pipeline befindlichen Befehle kann nicht von der Hardware unterstützt werden (z.B. gleichzeitiger Schreibzugriff zweier Befehle auf Registerdatei mit nur einem Eingang)

Steuerkonflikte (control hazard)

- Ursache in Verzweigung im Steuerfluss
- Bereits „falsche“, dem Sprungbefehl nachfolgende Befehle in Pipeline geladen

Datenkonflikte (data hazard)

- Ergeben sich Datenabhängigkeiten zwischen Befehlen im Programm
- Operanden noch nicht verfügbar (Anforderung vor Rückschreiben)

Echte Datenabhängigkeit (true/flow dependence)

- Ein Befehl j ist datenabhängig von einem Befehl i, wenn eine der folgenden Bedingungen gilt:
  - Es besteht eine direkte Abhängigkeit
  - Es besteht eine Datenabhängigkeitskette (j ist datenabhängig von einem Befehl k, der wiederum datenabhängig von i ist)

Namensabhängigkeit (Folie 20)

- Verwendung desselben Operanden (Register, Speicherzelle), jedoch ohne Datenfluss
- **Gegenabhängigkeit** (anti-dependence)
  i liest Operanden, der von j anschießend überschrieben wird (ADD R_2, $\mathbf{R_3}$, R_4; XOR $\mathbf{R_3}$, R_5, R_6)
- **Ausgabeabhängigkeit** (output dependence)
  i und j schreiben auf den selben Operanden (ADD R_2, R_3, R_4; XOR R_2, R_5, R_6)
- Auflösung durch Umbenennung

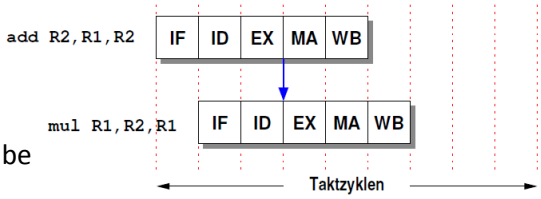
Arten von Datenkonflikten

- **Lese-nach-Schreib-Konflikt (RAW)**
  j liest Quelloperand bevor Aktualisierung durch i (Lesen eines zu alten Wertes)
- **Schreib-nach-Lese-Konflikt (WAR)**
  i liest Quelloperand nach Aktualisierung durch j (Lesen eines zu neuen Wertes)
- **Schreib-nach-Schreib-Konflikt (WAW)**
  i schreibt Zieloperand nach Aktualisierung durch j (Überschreiben eines aktuellen Wertes durch älteren)

Auflösen von Datenkonflikten

- **Statische Verfahren:**
  - Software-Lösung (Erkennen und Beheben durch Compiler)
  - Einfügen von Leeroperationen (NOP)
  - Umordnen der Befehle des Programms (Codeoptimierung)
  - Instruction Scheduling und Pipeline Scheduling
- **Dynamische Verfahren**
  - Hardware-Lösung, benötigt Konflikterkennungslogik
  - Unterschiedliche Lösungstechniken (Leerlauf der Pipeline, Result Forwarding)

Forwarding (Folie 25)



- Rückführung von Zwischenergebnissen
- Einfachster Fall: **Result Forwarding**
  - Rückführung des ALU-Ergebnisses zur Eingabe
  - kein Warten auf Rückschreiben
  - erfordert zusätzliche Datenpfade!
- Forwarding versagt bei externen Einflüssen → **Forwarding mit Interlock**
  - Anhalten bis Resultat bereitsteht
  - Forwarding zur Vermeidung von weiteren Wartezyklen

Zugriffszeiten und Kapazitäten

Hierarchie-ebene	1	2	3	4
Typ	Register	Cache	Hauptspeicher	Hintergrundsp.
Typ. Größe	<1kB	<16MB	<16GB	>1TB
Technologie	Multiport-CMOS-Speicher	CMOS-SRAM	CMOS-DRAM	Magnetisch, SSD
Zugriffszeit	<0.5ns	<10ns	<100ns	>20ms
Bandbreite	>20GB/s	5-10GB/s	1-5GB/s	≈100MB/s
Nutzungs-steuerung	Compiler	Hardware	OS	OS/Anwender

6 Speichertechnologie

Einsatz

- Statisches RAM (SRAM): Register, Cache-Speicher, Schneller Hauptspeicher
- Dynamisches RAM (DRAM): Hauptspeicher
- FlashRom: BIOS-Innhalte, Hintergrundspeicher
- Magnetische, optische und magneto-optische Speicher: Hintergrund- und Archivspeicher

Statisches RAM

- Besteht aus Flipflops und muss deswegen stehts unter Spannung stehen

Dynamisches RAM

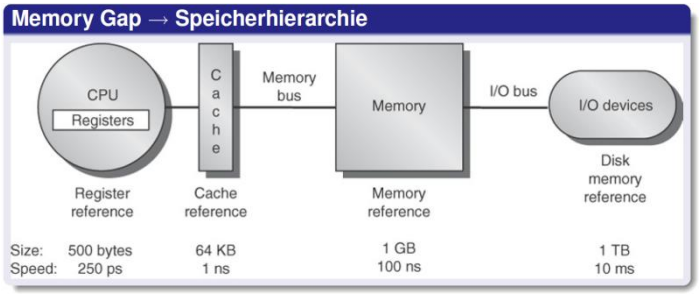
- Gespeichert wird in Kondensatoren und muss regelmäßig refresht werden

Asynchrone Technologien

- **Adressierung erfolgt mehrstufig**
  - Angabe einer Zeilenadresse und Quittierung mit RAS#
  - Angabe einer Spaltenadresse und Quittierung mit CAS#
  - Nach Verzögerung Ausgabe des Datums
- **Sequentielles Lesen → Seitenwechsel selten**
  - Fast Page Mode (FPM): Halten von RAS# und nur Update der Spaltenadresse
  - Überlappung der Zugriffe durch Halten der zuvor gelesenen Daten während neuem Zugriff (Extended Data Out, EDO) liefert weitere 5% Beschleunigung
  - Dedizierte Burst-Unterstützung durch internen Zähler (Burst EDO) erspart externes Adressupdate und liefert bis zu 50% Geschwindigkeitsvorteil

Memory Wall (Folie 22)

- Speicherzugriffe bremsen CPU aus (CPU: 4GHz = 0,25ns, Speicher: 30-50ns)
- Speicher entweder schnell und klein, z.B. Register, oder groß und langsam
- Speicherhierarchie puffert langsamen Zugriff durch Ausschnittsspeicherung in schnellerer Hierarchieebene



Speicher

	Art	Größe	Zugriffszeit	Zugriffsart
Register	SRAM	mehrere 10-100 Reg.	ein Taktzyklus (TZ)	Einzeln
Cache	SRAM	kb	wenige TZ	Wort oder Byte
Hauptspeicher	DRAM	GB	wenige 100 TZ	Blockweise
Hintergrund-Sp	Solid-State-Speicher	TB	mehrere 10ms	Seitenweise
Archivspeicher	Magnetische, optische Speicher	PB	mehrere 100ms	besondere Archiv-formen (tar, seriell)

Cache-Speicher

- Pufferspeicher zwischen Hauptspeicher und Prozessor
- Auszugsweise Spiegelung des Hauptspeichers in schnellem Cache-Speicher
- **Eigenschaften**
  - Blocklänge bestimmt Länge der Cache-Zeile
  - Cache-Zeile enthält Folge von Speicherwörtern
  - Statusanzeige:
    - Valid-Bit: Cache-Zeile enthält gültige Kopien
    - Dirty-Bit: Cache-Zeile enthält veränderte Kopie
- **Funktionsweise**: Cache-Steuerung überprüft Speicherzugriffe des Prozessors
  Steht zur Speicheradresse gehörender Inhalt als Kopie im Cache und ist Inhalt gültig?
  - Treffer bzw. Hit: Beide Bedingungen müssen erfüllt sein → Trefferzugriffszeit
  - Fehlzugriff bzw. Miss → Transfer von Inhalten notwendig → Fehlzugriffszeit
- **Ursachen für Cache-Misses**
  - Compullsory Misses: Erstmaliger Zugriff auf einen Block
  - Capacity Misses: Mehr Blöcke als vorhanden benötigt
  - Conflict Misses: Verdrängung durch Abbildungsvorschrift

Klassifikationskritierien

- Platzierungsstrategie: Wohin kann ein Block platziert werden?
  - Direct-mapped: Jeder Block an nur exakt einer Position
  - Fully associative: Jeder Block an beliebiger Stelle
  - set-associative: Mischform
    Aufteilung der Sätze wie beim direct-map
    Satz selbst aber voll Assoziativ
- Findstrategie: Wie wird ein Block gefunden?
- Ersetzungsstrategie: Welcher Block wird bei Fehlzugriff ersetzt?
  - Least frequently used
  - Least recently used
    → bedingt Zusatzaufwand → Rentabilität
- Rückschreibstrategie: Was passiert bei Blockveränderung?
  - **Write-through-Strategie**
    Schreiben des Datums in den Cache verändert auch Hauptspeicherinhalt
    Einfach zu implementieren, kein Zusatzaufwand bei Miss
    konsistenter Hauptspeicher, wichtig bei Shared-Memory-Multiprocessing
    No write-allocate: Bei Write-miss keine Einlagerung sondern direkte Modifikation
    → häufiger Speicherzugriff (Bandbreitenproblem)
  - **Write-back-Strategie**
    Zurückschreiben bei Verdrängung, dazu ist ein Dirty-Bit erforderlich
    Inkonsistenzproblem
    Write-allocate: Bei write-miss Einlagerung des Blocks und normale Behandlung
    → wenige Speicherzugriffe, schneller bei mehrfachem Ändern des selben Blocks

→ keine Einheitliche Strategie möglich → Abwägung erforderlich

7 Ein-/Ausgabe

Peripheriemerkmale

- Art und Aufgabe der Schnittstelle
  - Parallele Schnittstelle: gleichzeitiges Setzen/Löschen von Signalen (PATA, SCSI)
  - Serielle Schnittstelle: Kommunikation über wenige Leitungen (SATA, USB)
- Auflösung von Unterbrechungen
  - Schnelle und flexible Reaktion auf Ereignisse
  - Priorisierung von Interrupts
- Datentransfer (Abhängig vom gewählten Inhalt)
  - Multicast/Broadcast (vgl. Internetradio)
  - Unicast (vgl. Youtube)
  - Erzeugung eines dedizierten Pfads von Sender zu Empfänger (Telefon)
    reine Weiterleitung der Knoten, Aufbau erfordert Zeit
  - Paketvermittlung (Internet)
    Variable Leitwege vom Sender zum Empfänger, nicht 100%ig verlässlich
    gemeinschaftliche Ressourcen werden gut ausgenützt
- Zeitmessung und zeitgesteuerte Auslösung
- Erfassung und Erzeugung von Kenngrößen
  - Wandlung analoger Signale in digitale (A/D-Wandler)
    - Abtastrate (Häufigkeit der Wandlung, Hz)
    - Quantisierung (Genauigkeit der Wandlung, Bit)
    - Qualität der Wandlung (Genauigkeit)
  - Wandlung digitaler in elektrische Signale (D/A-Wandler)

Direct Memory Access (DMA)

- Autonomer Datentransport zwischen Speicher und Peripherie ohne CPU-Einsatz
- Entlastung der CPU bei großen Datentransfers und häufigen Unterbrechungen

Übertragungstechniken/Protokolle

- Fast/Gigabit Ethernet
- Firewire (Ziel: Hochgeschwindigkeits-Peripheriebus zum Anschluss von Speichmedien)
- Universal Serial Bus (USB)
  - Kommunikation über unterschiedliche Pakettypen
  - Bulk-Daten: Sichere Übertragung, jedoch keine reservierte Bandbreite
  - Isochrone Daten: Reservierte Bandbreite, jedoch unsicher

8 Bussysteme

Busstruktur

- Gemeinsamer Bus vs. getrennte Bussysteme
- Parallele vs. sequentielle Übertragung
- Eigenständiges Busprotokoll oder Ableitung aus Prozessorbus
- Priorisierung
- Signalisierung

Einkartensystem

- Gesamte Hardware ist auf einer Platine untergebracht
- Verbindung der Systembausteine über Bus
- Feste Systemstruktur (minimale, an Aufgabenstellung angepasste Hardware)

Mehrkartensystem

- Mikroprozessorsysteme für allgemeine und spezielle Anwendungen
- Aufbau als modulares Mehrkartensystem
- Änderung der Systemstruktur möglich (Hinzufügen und Tausch von Karten in Slots)

Industrielle Mehrkartensysteme

- Aufteilung der Funktionalität
- Universelle Schnittstellendefinitionen (Aufteilung in Prozessor- und Peripheriebus)

Mehrbussysteme

- Prozessor-/Speicherbus: Schnelle Übertragung zwischen angeschlossenen Einheiten, auf Taktrate des Prozessor abgestimmt, Kopplung mit Systembus
- Zentraler (globaler) Systembus: Global zugängliche Speicher- und E/A-Einheiten, Ergänzung durch weitere (lokale) Busse (Peripherie, Nachrichten, CPUs, DMA-Controller)
- Buskopplung: Erfordert dedizierten Controller, Kopplung der Adress- und Datenleitungen Ermöglicht Anschluss von anderen Speicherkomponenten
- Peripheriebus (Zusammenfassung von peripheren Geräten)
- Nachrichtenbus (Austausch von Nachrichten zwischen eigenständigen Rechnerkarten)

Bussteuereinheiten

- Datenübertragung zwischen Master und Slave
- Interrupt-Verwaltung (Signalverwaltung)
- Busarbitrierung (Priorisierung der von mehreren Mastern kommenden Anforderungssignale)
- Nutzfunktionen (Strom-, Taktversorgung)

Bussignale

- Leitungsbündel: Adress-, Daten-, Steuer- und Versorgungsbus (ggf. Teilzusammenfassung)
- Signalfluss: Unidirektionale vs. bidirektionale Signale

Busarten

- Split Bus (Getrennte Adress- und Datenleitungen)
- Multiplexed Bus (Zusammenlegung von Adress- und Datenleitungen)
- Einzelbuszyklus: Durchführung einer einzigen Datenübertragung
- Blockzyklus: Kapselung konsekutiver (sequentieller) Datentransfers

Blockübertragung (Burst Transfer)

- Übertragung konsekutiver Datenblöcke, d.h. Daten stehen in aufeinanderfolg. Speicherzellen
- Initialisierung: Signalisierung des Zugriffsbeginns, Angabe einer Startadresse
- Transfer: Hochzählen der Adresse bei nachfolgenden Buszugriffen
- Abschluss Signalisierung des Transferendes

Busteilnehmer

- Master: Initiiert Bus-Transaktion, Generiert Adress-Steuerinfos
- Slave: Passiver Teilnehmer, Kommunikation nur wenn angesprochen, Abschluss der Transaktion
- Sender: Einheit, die gerade Daten auf den Bus gibt
- Arbitrierer: Zentrale Steuerung des Busfreigabe, Essentiell in Multi-Master-System

Datentransportsteuerung (Folie 16)

- **Synchroner Systembus**
  - Übertragung unterliegt festem Zeitraster (orientiert sich am Bustakt)
  - Adressierungsphase → Datentransferphase (Lesen) → Datentransferphase (Schreiben)
- **Semisynchroner Systembus**
  - Gemeinsames Taktsignal
  - Signalisierung des Beginns (Master)
  - Definition fester Zeitpunkte für bestimmte Interaktionen
- **Asynchroner Systembus**
  - Keine Synchronisation durch gemeinsames Taktsignal
  - Signalisierung durch diverse Signalleitungen (ALE, R/W#, CS#)

9 Kodierung von Informationen

Serielle Datenübertragung

- Elektrische oder akustische Definition der 1/0 Symbole
- Festlegung einer festen Symbollänge (ASCII 7-Bit-Code, UTF)
- Festlegung eines Zeitrasters (Symbolrate, Rohdatenrate)

Lauflängenkodierung

- Annahme: Häufige Abfolge identischer Symbole
- Kodierung einer Rekonstruktionsvorschrift (z.B. 12w1B11w für 12 Weiße, 1 Schwarzer, 11 weiße)
- Verlustfreies Kompressionsverfahren, kein Datenverlust durch Kompression, aber ineffizient, falls zu wenige Symbole einander folgen → Verallgemeinerung: Dictionary Coders

Sequenzkodierung

- Ermitteln von Sequenzen in Zeichenfolge
- Ablegen von wiederholten Sequenzen in Wörterbuch
- Ersetzen von wiederholten Sequenzen durch Zeiger auf Wörterbuch

Huffman-Kodierung (Folie 10)

- Berücksichtigung der Häufigkeit von Symbolen
    - häufige Symbole → kurze Repräsentation
    - seltene Symbole → lange Repräsentation
      → Transofrmation des Alphabets (z.B. Morse-Code: e häufig, y selten)
- Aufbau eines Kodierungsbaums unter Berücksichtigung der Wahrscheinlichkeiten des Auftretens der Zeichen (Ableitung des Codes über die Bewertung der Kanten)

Fehlererkennung und –behebung

- **Error Detection Code** (EDC): Fehlererkennung
    - **Parity Checking**: Ermitteln der Anzahl von 1-Bits in einem Byte/Wort
        - Anzahl 1 gerade: even parity
        - Anzahl 1 ungerade: odd parity
          → erkennt Übertragungsfehler
    - **Checksumming**: (Problem: je zwei Fehler an gleicher Position neutralisieren sich)
        - Bilden eines Fingerabdrucks der kompletten Nachricht
        - Übertragen des Fingerabdrucks mit Nachricht
        - Empfänger generiert eigenen Fingerabdruck und vergleicht
- **Error-Correcting Codes** (ECC): Fehlerbehebung
    - Funktionsprinzip: Redundante Kodierung (erlaubt Rückrechnung bei Fehlern)
    - einfaches Beispiel Triplet-Code: Dreifaches Übertragen jedes Bits → ineffizient
    - besser z.B. Hamming-Code

Hamming-Code (7,4) (Folie 18)

- 4 Bit Nutzdaten in 7 Bit Daten → Parität 3 Bit
- Überlappende Paritätsbildung erlaubt Rückrechnung
- Über Matrizen-Rechnung kodieren und dekodieren

Funktionswahrscheinlichkeit

$$\varphi_m^n = \sum_{k=n}^{m} \binom{m}{k} * \varphi(K)^k * (1 - \varphi(K))^{(m-k)}$$

- Jede Komponente K eines Systems hat Funktionswahrscheinlichkeit φ (K)
- Funktionswahrscheinlichkeit eines Seriensystems: $\prod$
- Funktionswahrscheinlichkeit eines Parallelsystems: Σ
- Repräsentation über Zuverlässigkeitsblockdiagramm oder Strukturformel

Beispiel: 2-aus-3-System, n=2, m=3

$$\varphi_3^2 = \sum_{k=2}^{3} \binom{3}{k} * \varphi(K)^k * (1 - \varphi(K))^{(3-k)}$$

Redundanzsysteme

- Erhöht die Systemsicherheit durch Vervielfachung der kritischen Komponenten
- **Hot Standby**: Backup-Komponenten rechnen immer parallel, hoher Energiebedarf, Verschleiß der Backup-Komponenten, schnelle Umschaltzeit
- **Cold Standby**: Backup-Komponenten werden bei Bedarf aktiviert
- **Graceful Degradation** (Kompromiss): System wird auf durchschnittlich benötigte Leistung hin aufgebaut mit Sicherheitszulage
    - Im Normalbetrieb: typischerweise Parallelbetrieb aller Einheiten (gleichmäßig)
    - Im Fehlerfall: Verteilung der verbleibenden Ressourcen → weiterhin funktionsfähig

Alterungseffekte

- Konstante Ausfallrate ist vereinfachtes Modell
- Reale System haben eine variable Ausfallwahrscheinlichkeit über die Zeit
- **Badewannenkurve**:
    - Frühphase: Intitialausfälle, Fertigungsfehler, Bauteildefekte
    - Betriebsphase: Nahezu konstente Ausfallrate
    - Spätphase: Alterungseffekte, Ausfallrate exponentiell ansteigend

10 Betriebssystemaspekte

Virtueller Speicher

- **Hauptspeicher**: Adressraum durch Architektur beschränkt (IA32: 4 GB)
- **Virtueller Speicher**:
  - Kapazitätserweiterung durch Hintergrundspeicher
  - Vorspiegelung eines Hauptspeichers großer Kapazität
  - Großer, einheitlicher Adressraum für die einzelnen Prozesse (Schutzmechanismen)
- **Multitasking**: Ausführung unabhängiger Tasks, Abtrennung der Adressräume zur Absicherung

Virtuelle Speicherverwaltung

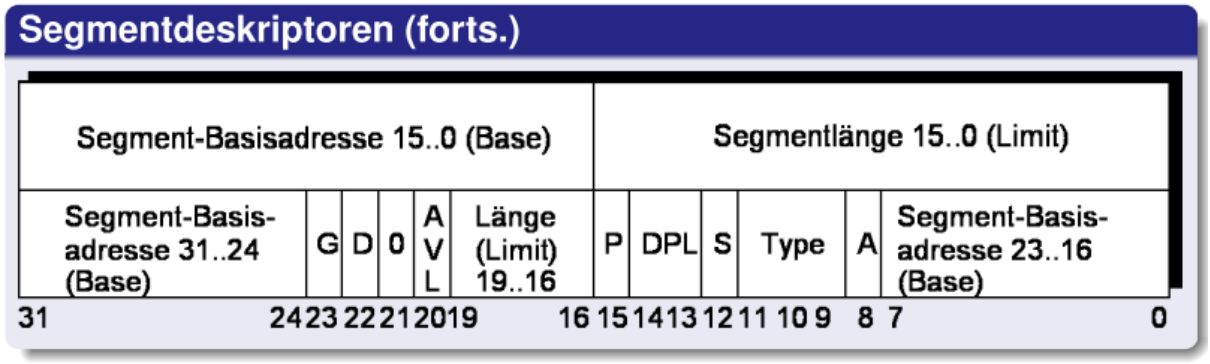
- Einladen von Programmen und Daten in den realen Hauptspeicher
- Vermerken freier Speicherbereiche (Buchführung)
- Auslagern von Speicherinhalten in Hintergrundspeicher
- **Voraussetzungen**:
  - Lageunabhängigkeit der Programme und Daten
  - Adressierungsmethoden
  - Verwendung einer Speicherverwaltungseinheit

Flat-Modell (Linux)

- Ein einziger, linearer Adressraum
- Abbildung aller Segmente auf den gesamten physikalischen Adressraum

Segmentselektor

- Speicherung im Segmentregister
- Index in die Deskriptortabelle
- Verweis auf Segmentdeskriptor
- Segment-Deskriptor Cache: Änderung des Inhalts eines Segmentregisters führt zum automatischen laden des zum Selektor gehörenden Deskriptors in den unsichtbaren Teil des Segmentregisters
- Deskriptortabellen: variable lange Felder im Speicher, Einträge der Tabellen enthalten Deskriptoren der im System verfügbaren Segmente
  - **Globale Deskriptortabelle** (GDT): Existiert einmal pro System
    - Deskriptoren der OS-Segmente (Code, Daten)
    - Deskriptoren der shared-Sgemente (gemeinsam genutzte Datenbereiche)
    - Deskriptoren für die lokalen Deskriptoretabellen
  - **Lokale Deskriptortabelle** (LDT): Existiert individuell pro Task
    - Deskriptoren der taskeigenen Segmente (Code, Daten)
    - Kein Zugriff anderer Tasks auf die lokalen Segmente eines anderen Tasks möglich
- Zugriff auf die Tabellen über privilegierte Befehle (L**G**DT/L**L**DT, S**G**DT/S**L**DT)



Segmentdeskriptoren

- **Basisadresse (Segment Base, 32 Bit)**
- **Granularität (G)**: Skalierung für die Segmentlänge (0: 1 Byte, 1: 4kByte)
- **Standardgröße (D)**: Festlegung der Längen der Operanden und effektiven Adressen (1: 32-Bit-Operanden, 0: 16-Bit-Operanden)
- **Segmentlänge (Limit)**: Überprüfung von Speicherschutzbedingungen
- **Segment Present (P)**: Anzeige, ob Segment im Speicher geladen ist (1)
- **Privilegebene (DPL)**: Privileg-Ebene, bis zu der auf den Deskriptor zugegriffen werden darf
- **Deskriptorformat (S)**: Systemdeskriptor (0) oder Code-/Datendeskriptor (1)
- **Ausführbarkeit (E)**: Codesegment (1), Datensegment (0)
- **Konformität (c)**: Systemdienste, die nicht auf geschützte Bereiche/Einheiten zugreifen
- **Lesezugriff (R)**: Code-Segment ist execute-only (0), auch Lesezugriff möglich (1)
- **Zugriff (A)**: Signalisierung, ob Segmentselektor in Segmentregister geladen wurde

Seitenverwaltung

- Umsetzung einer linearen in eine physikalische Adresse
  - Transparent, d.h. für den Programmierer unsichtbar
- Nur aktuell benötigte Seiten im Hauptspeicher
  - Auslagerung aller anderen Seiten in den Hintergrundspeicher
- Aufteilung des Adressraums in Seiten fester Größe

Translation Lookaside Buffers (TLBs)

- **Adressberechnung ist aufwendig**: ggf. mehrere Hauptspeicherzugriffe, bis Adresse gefunden Seitentabellen erstecken sich ggf. auf mehrere Seiten
- **Adressumsetzung mit Hilfe der TLBs**
  - TLB ist dedizierter Cache-Speicher
  - Speicherung der jüngsten Seitentabelleneinträge
  - Zugriff auf Hauptspeicher nur, wenn kein Treffer in TLB
  - Tag ist Teil der virtuellen Adresse
  - Inhalt entspricht Teil der physikalischen Adresse incl. Verwaltungsinformation
- **Getrennte TLBs** für Befehle und Daten oder unterschiedliche Seitengrößen
- **Privilegierter Zugriff** (erfordert Systemmodus)
- **Invalidierung der Einträge** bei Laden von CR3

11 Betriebssystemaspekte II

Schutzmechanismen

- Trennung von Anwender- und Systemschicht (User vs. Supervisor)
- Isolation der Anwendertasks (Separierung der Speicherbereiche, Zugriff/Überschreiben)
- Isolation von Code, Daten und Stack (Zugriff, Ausführbarkeit)

Segment-Deskriptoren

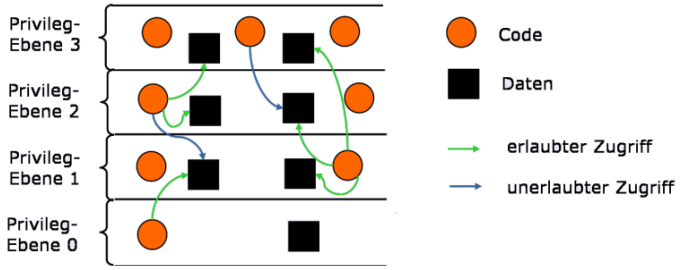
- Unterscheidung zwischen Code- und Datensegmenten
- Durchführung einer Typprüfung
- **Code-Segment**: Ausführbarer Programmcode und lesbare Konstanten (kein Schreibzugriff)
- **Daten-Segment**: Keine Ausführung! Read-only bzw. Read/Write
- **Stack-Segment**: Read/Write, Festlegung auf Zugriffsart erfolgt im Typ-Feld des Deskriptors

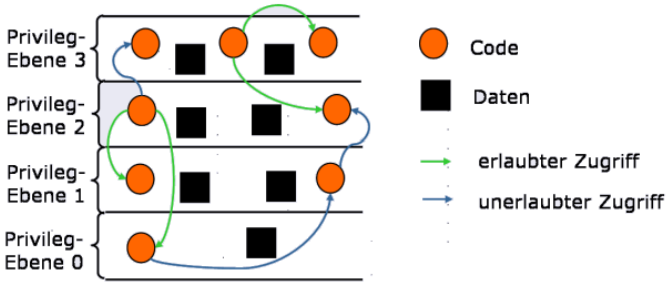
Überprüfungen

- Prozessor prüft ob geladenes Segment ausführbar und als Code verwendet wird. (entspr. mit Daten)
- Segmentgrenzenüberprüfung: Stacküberlauf, Fehlerhafte Zeiger, Böswilliger Zugriff

Privileg-Stufen

- Zuordnen von Privileg-Ebenen für Daten- und Codesegmente
- Individuelle Privilegien pro Datensegment
- Ausführung eines Prozessors auf bestimmter Privileg-Ebene
- Hierarchisches Schutzmodell: Schutz der Systemsoftware gegenüber unerlaubten Zugriffen durch die Anwenderprogramme, Programmunterbrechung bei unerlaubtem Zugriff





- **REGELN**: Ein Prozess darf nur auf **Daten zugreifen, die höchstens** genauso vertrauenswürdig sind wie er selbst und ein Prozess darf nur **Code benutzen, der mindestens** genauso vertrauenswürdig ist wie er selbst.
- Stufe 0: Betriebssystemkern; Stufe 1: Betriebssystemdienste (Treiber etc.)
  Stufe 2: Betriebssystemdienste; Stufe 4: Anwendungsprogramme (niedrigste Stufe)
- Durch die Stufen gibt es Einschränkungen bei Kontrollflussübergängen (JMP, CALL, RET)

Schutzmechanismen

- Gate-Deskriptoren: Regeln den Zugriff auf Segmente verschiedener Privilegebenen
- Unterprogrammaufrufe: Call-Gates, definieren Einsprung in eine Prozedur
  Fehlerbehandlung → Trap-Gates
  Unterbrechungsbehandlung → Interrupt-Gates
  Prozesswechsel → Task-Gates
- **Seitenverwaltung**: Einschränkungen des Adressierbaren Bereichs
  Systemmodus: CPL=0,1,2, alle Seiten zugreifbar
  Benutzermodus: CPL=3, nur Seiten des Benutzers zugreifbar

Multitasking

- Quasi-parallele Ausführung mehrerer Benutzerprogramme durch periodisches Durchschalten
- Task: in Ausführung befindliches Programm, auf Ausführung wartendes Programm
- Taskwechsel: Unterbrechen eines Tasks und Fortführen eines anderen, erfordert vollständige Isolation der Tasks

Ein-/Ausgabe

- Memory-mapped: Gemeinsamer Adressraum mit Hauptspeicher, Adressierung der Register der Schnittstellenbausteine wie Speicherzellen, Verwendung der herkömmlichen Speicherzugriffsbefehle, Schutzmechanismen der Segment- & Seitenverwaltung
- Zugriffskontrolle bei isolierter Ein-/Ausgabe:
    - Ausführen von Ein-/Ausgabebefehlen auf höher privilegierten Ebenen (0 oder 1)
    - Anwendungsprogramme führen dies mit Hilfe des OS durch
    - I/O Permission Bitmap: Kontrollierter Zugriff auf I/O-Ports, I/O-Adressen eingeschränkt, Generierung von Exceptions

Programmunterbrechung - Ablauf

- **Unterbrechung** der Abarbeitung eines laufenden Programms durch besondere Situation z.B. Exceptions, Traps, Prozessorinterne und –externe Ursachen
- **Reaktion auf eingetretene Situation**: Unterbrechungbehandlung
- **Rücksprung**
- Priorisierung von Unterbrechungsanforderungen
    - Reset
    - Nicht maskierbarer Interrupt
    - Maskierbarer Interrupt
    - Interne und externe Priorisierung

Exceptions (Alarme)

- Auslösung synchron zur Prozessorverarbeitung, Prozessorinterne Ursache
- Unterschreidung zwischen
    - **Faults**: Unterbrechung und Behandlung finden *vor* der verursachenden Instruktion statt, z.B. Page Fault (Seite liegt nicht im Hauptspeicher)
    - **Traps**: Unterbrechung und Behandlung finden *nach* der verursachenden Instruktion statt, Beispiel: Breakpoint
    - **Aborts**: Signalisierung eines schwerwiegenden Fehlers, keine Fortsetzung möglich, ebenso keine genaue Bestimmung und Behebung möglich (Division durch 0)

Interrupts

- Auftreten typisch asynchron zur Prozessorverarbeitung, Prozessorexterne Ursache
- Unterscheidung zwischen
    - Maksierbar: Spezieller Interrupteingang (NMI) Kann nicht vom Programmierer verhindert werden, Blockeirung weiterer NMIs während der Behandlung
    - Nichtmaskierbar: Gemeinsamer Eingang oder Gruppe von Eingängen, stattgabe, wenn Interrupt-Enable-Flag im Steuerregister gesetzt ist