Einführung in die technische Informatik

Technische Realisierung von Rechnern

Benutzerprogrammschicht	– Hochsprache, evtl. mehrere Schichten: OS, Programm	Java
↓	Compiler / Interpreter	
Von-Neumann-Schicht	– cml (= conventional machine level)	Assembler
↓	Mikroprogrammierung	
Mikroprogrammschicht	– rtl (= register transfer level) Def durch Mikroinstruktionssatz	Mikroprog.
↓	Räumliche (anstatt zeitliche) Darstellung	
Gatterschicht	– Boolsche Funktionen	VHD
↓	Schaltungen	
Bauelementeschicht	– Idealisierte Bauelemente	Schaltplan
↓		
Physikalische Schicht	– Reale Bauelemente	

Maschinenzyklus in Takten (jeweils x Taktzyklen)

1. Befehl adressieren
2. Lesezyklus im Speicher
3. Befehl einlesen
4. Befehl dekodieren und ausführen

Maschinenbefehlszyklus: 1 – 5 Maschinenzyklen

Maschinenbefehlszykluszeit: Ausführungszeit des Befehls

Taktzykluszeit: 1 / Arbeitsfrequenz

10 **Maschinenzustände** (jew. 1 Taktzyklus lang) Abhängig von Befehl

Mit Befehl holen: T1, T2, T3, T4 (Twait, T5, T6)

Ohne Befehl holen: T1, T2, T3

Externe Signale: Twait, Treset, Thalt, Thold

Adressierung des 8085:

- Immediate (Operand im Befehl; 2. (+3.) Byte)
- Implizit (Fest gecoded; im Op-Code)
- Registerdirekt (Register mit Operand angegeben)
- Direkt (Speicheradresse des Operanden im Befehl angegeben; 2. (+3.) Byte)
- Registerindirekt (Register mit Speicheradresse im Befehl angegeben)

Aufbau 8086:

Memory	←	Bus Interface Unit	←	Execution Unit
		Datentransfer		ALU (Arithmetic
		Prefetching		Logic Unit), Register

Daten werden nur über BIU übertragen

Mikroarchitektur

(= Register Transfer Level)

- Maschinenbefehl interpretiert durch Mikroprogramm
- Mikroprogramm besteht aus Mikrobefehlen im Mikroprogrammspeicher
- Mikroinstruktion besteht aus Mikrooperationen, die gleichzeitig (im selben Takt) ausgeführt werden
- Mikrooperation ist Steuerinformation für Registertransfer

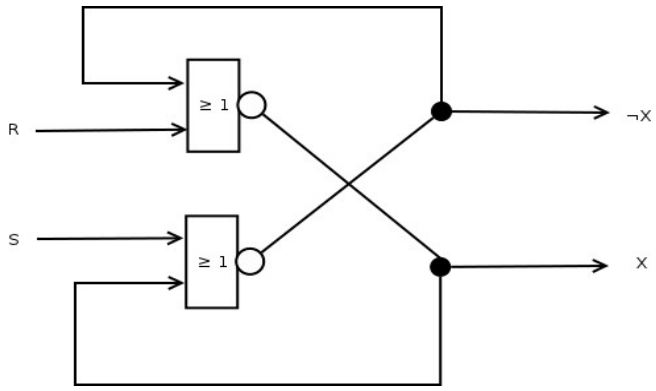
Bestandteile: Schaltnetze (Funktionen, Pfade), Schaltwerke (Speicher)

Bsp: Multiplexer (MUX), Wähle 1 aus N, wenn Output Enabled (oe) gesetzt

Speicherfunktionen (Rückführbar auf 1-Bit Speicher), spezielle Schaltwerke (=sequential networks)

R-S-Flip-Flop:

R	S	X	¬X
0	0	Unverändert	
0	1	1	0
1	0	0	1
1	1	Undefiniert	



Mikroinstruktionsformat

Op-Code Adressteil

Steuercode (+ Immediates)

Horizontal (Unverschlüsselt) s e_1 e_2 s

↓ ↓ ↓ ↓

Vertikal (Stark verschlüsselt) Code Code

Qasihorizontal (optimal verschlüsselt) ↓ ↓

Decoder Decoder

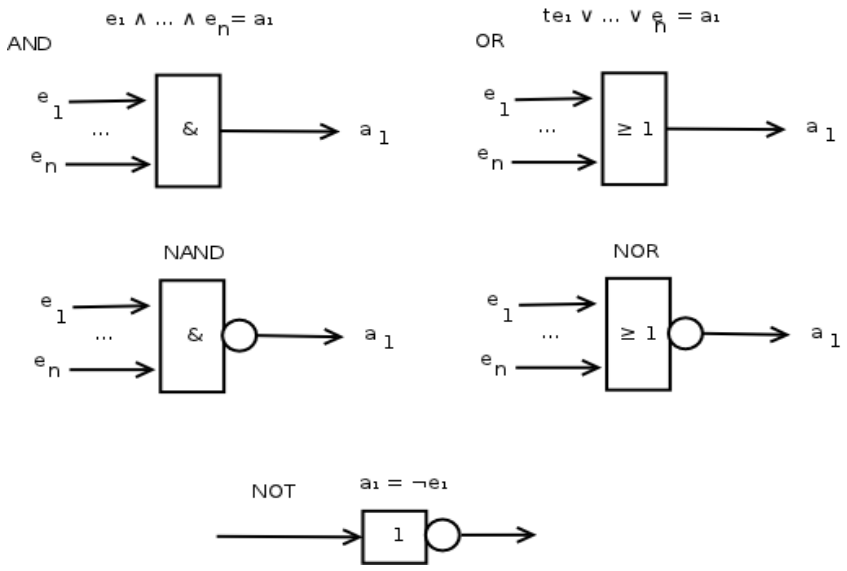
↓ ↓ ↓ ↓

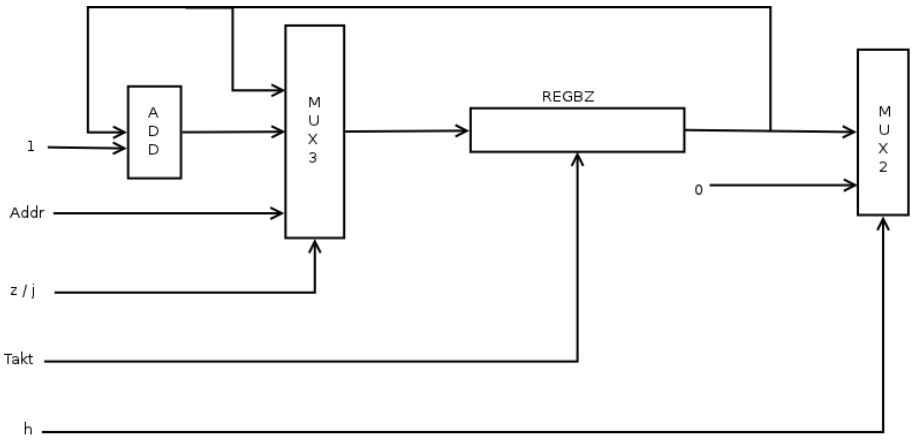
s e_1 e_2 s

Gatterschicht

Realisierung elementarer Boolscher Funktionen

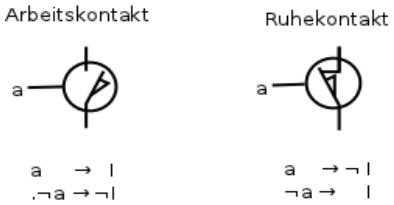
+ Flip-Flops, Befehlszähler, usw.



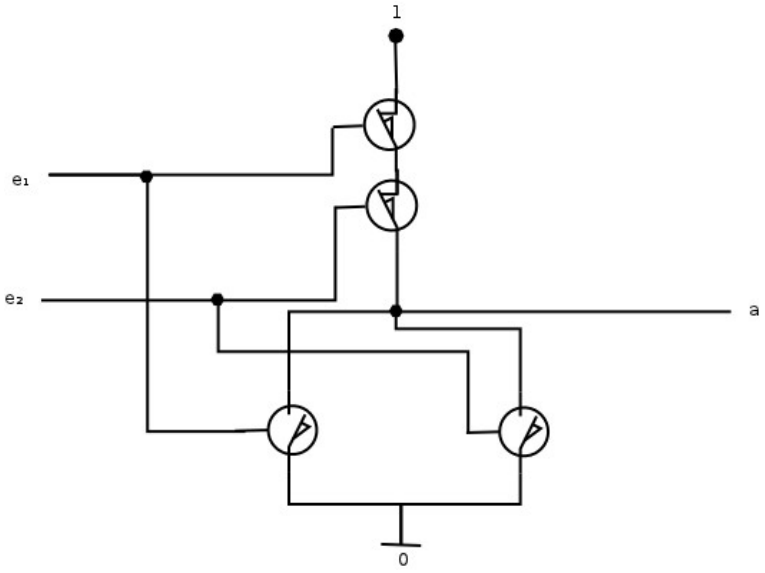


Bauelementeschicht

Realisierung der Gatter (z.B. durch Feldtransistor)



Bsp: a <= e_1 NOR e_2



Physikalische Schicht

Halbleiter Silizum: Es kann nur Strom fließen, wenn ein Steuerstrom anliegt.

Integrierte Schaltkreise ca. 1cm² mit 10^{10} Transistoren

Beispiele für Mikrocontroller

Mikrocontroller vs. Mikroprozessor

- System on a Chip → Mikrocontroller ist Mikroprozessor + Peripherie / Speicher

8051

Indirekte Adressierung: Akkumulator

Befehlszyklus: 1-2 Maschinenzyklen

Maschinenzyklus: 6 Taktzyklen

1. Befehl laden in IR
2. Befehl dekodieren, Befehlszähler ++
3. Operanden vorbereiten
4. tmp1 = Op1; tmp2 = Akkumulator
5. Alu Operation ausführen
6. Ergebnis → Hauptbus

→ Billig, echtzeitfähig, 8-Bit, RISC

→ Mehrere Registersätze → schnelle Interrupt Behandlung

ARM Cortex

RISC Prozessoren für eingebettete Systeme

- in SOC integriert
- Mobilfunk
- Stromsparend
- Konkurrenz zu Intels Atom

→ wenige, einfache Befehle; Wenige Transistoren (stromsparend), schnelle Verarbeitung

- Festes Befehlsformat 4 Byte (32-Bit); Thumb Befehlssatz 2 Byte
- Load / Store (auch multiple → mehrere Register)
- Arithmetische + Logische Operationen + Shift
- Adressierung: Register, Direkt
- Bedingte Ausführung
- 37 Register; 15 für Benutzer

<opcode>	{<cond>}	<Ziel>,	<opt. Quelle>,	<shifter_operand>
ADD	EQ	R2,	R4, R5	

→ wenn Zero Flag gesetzt: R2 = R4 + R5

Bedingte Instruktionsausführung spart Sprünge, Code kompakter, Performance

Stromsparen:

- Clock Gating: Kein Takt an Flip-Flops → Müssen nicht umschalten
- Sprungvorhersage
- Physiche Caches (Speichert Adressen) → Weniger Cache Flushes
- Caches
- Kurze Schleifen ohne Cache
- verschiedene Spannungsbereiche
- verschiedene Power Modes für verschiedene Bereiche → kein Taktsignal für ausgeschaltete Bereiche

Schaltnetze

Bestehend aus Zeichen (keine kontinuierlichen Funktionen): genau, sicher, gut lesbar

Binär:

- sichere / billige Übertragung & Speicherung
- Einfache Arithmetik
- elegant formulierbare Steuerung → Mikromaschine

Einsmenge:	Eingangstupel einer Operation, die 1 ergeben; Bsp OR: { (1,0); (0,1); (1,1) }
Nullmenge:	Analog Einsmenge; Bsp OR: { (0,0) }
Minterm:	Vollkonjunktion; Bsp: 0, 0, 0 → ¬a ∧ ¬b ∧ ¬c
Maxterm:	Volldisjunktion; Bsp: 0, 0, 0 → a ∨ b ∨ c
Funktionseinheit:	Gebilde mit Aufgabe / Wirkung
Schaltnetz:	Funktionseinheit, Bündel von m Schaltfunktionen; D = { 0, 1 }; Sonderfall: m = 1 → Gatter → Datenflusspläne / Logische Pläne (Logik entspricht binärer Bedeutung)

Schaltwerk: Funktionseinheit mit Speicher (z.B. Flip Flop); mindestens ein Ausgang ist an einen Eingang rückgekoppelt → Ausgang hängt nicht mehr nur von der Eingabe ab.

Schaltfunktion: n-stellig → n² Definitionspunkte; Bsp n = 1 → 0 = ?, 1 = ?

n^{2^n} n-stellige Schaltfunktionen; Bsp n = 1 → 0 | 0 0 1 1

Beschreibbar durch: Funktionstafel (Wertetabelle) 1 | 0 1 0 1

Arithmetische Ausdrücke

Karnaugh-Veitch-Diagramm

Durch ∞ verschiedene Ausdrücke definierbar: A, A ∨ A, A ∧ A, KNF, DNF

Funktional: Ergebnisse reproduzierbar, bei selben Eingaben → Algorithmen zur Erstellung von Schaltnetzen

Minimierung:

- weniger Schaltglieder
- weniger Eingänge / Leitungen
- weniger Stufen
- kürzere Leitungen
- billigere Schaltglieder

		b	
a	1	1	
a	0	1	c
	0	1	c
	0	1	

Karnaugh-Veitch-Diagramm (entspricht benachbarten Termen der Algebra)

→ Vereinfachung von Schaltfunktionen Bsp: (abc ∨ ab¬c ∨ ¬abc ∨ ¬ab¬c) → b

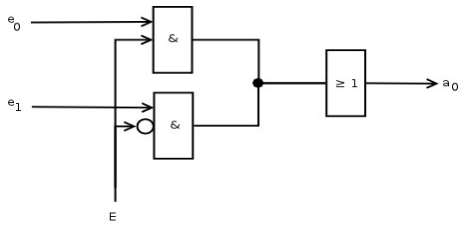
Allgemein:

- Finde Primimplikant zu Minterm (Nicht weiter vereinfachbar; xy ∨ x¬y = x nicht möglich)
- Finde kleinste Überdeckungsmenge aus Primimplikanten
- → DMF (= Disjunktive Minimal Form)

Decoder: Genau ein Ausgang = 1 für genau ein gültiges Eingangswort → $|A| \leq 2^{|E|}$

Encoder: 1 von k Bits gesetzt → 1 Ausgangswort → k < 2|A|

MUX / DEMUX: Multiplexer → Lässt genau einen Ausgang durch. Demultiplexer legt einen Eingang an genau einem Ausgang an.



ROM: n-stelliges wird auf m-stelliges Wort abgebildet

Decoder (Adressentschlüssler) → Encoder (Speicherzellen)

Funktionsbündel: gemeinsame Schaltglieder → oft bessere Schaltnetze möglich, als durch DMF findbar

NAND/NOR:

- funktional vollständig
- kein Minimalisierungsverfahren DMF → NAND / KMF → NOR günstig (aber nicht unbedingt minimal)

  Wandlung: DNF → NAND Auch mehrstufig möglich

[ε | ¬] ∧ ∨ [ε | ¬] → [ε | ¬] NAND NAND [ε | ¬]

Analog für KNF → NOR

Hazards:

- Spezifikationswiedriger Wert des Ausgangssignals bei Umstellung / durch Zeitunterschiede im Schaltnetz
- → Einführung eines Taktes (hinreichend groß), nach dem alle Änderungen abgeschlossen sind
- → Hazards kosten Zeit, Aber in Schaltnetzen nicht „bösartig“

Schaltwerke und endliche Automaten

Schaltwerk: s.o.

Mealy Automat: E, A, Q endlich (Ausgabe bei Zustandsänderung)

M(	E,	A ,	Q,	λ,	δ,	q)
Eingangsalphabet		Ausgangsalphabet	Zustandsraum	E x Q → A Ausgabefktn.	E x Q → Q Zustandsfktn.	Startzustand

Moore Automat: (Ausgabe bei Zustand)

N(	E,	A,	Q,	λ,	δ,	q)
	s.o.	s.o.	s.o.	Q → A	s.o.	s.o.

Endlicher Automat:

- Zur Definiton & Analyse formaler Sprachen
- Modell für Rechner & Werke
- im Allgemeinen: Zustandsraum zu groß für sinnvolle Schlüsse
- λ und δ sind Schaltnetze
- δ mit Verzögerung τ (durch Netz)
- $a_t = \lambda(e_t, q_t)$
- $q_t = \delta(e_{t-\tau}, q_{t-\tau})$
- Darstellung des Automaten: Automatentafel / Automatengraph
- Es gibt zu jedem M einen Äquivalenten N ($\lambda_M = \lambda_N$)

Prozess: Handlung, die zeitlich zergliederbar ist

Parallel: Zwei Handlungen, die zu mindestens einem Zeitpunkt begonnen, aber nicht beendet sind

Asynchrone Prozesse: Keine Verbindung zwischen 2 Prozessen

Synchrone Prozesse: Prozess 1 beginnt erst mit Schritt i, wenn Prozess 2 Schritt j begonnen/beendet hat

Funktional: Reproduzierbare eindeutige A bei gleichen E

Synchronisation durch Takt

Synchrones Schaltwerk: Rückkoppelung ist durch Takt zeitgesteuert → Aufwändiger + langsamer

Schaltwerk stabil, wenn sich die internen Zustände nicht ändern;

RS-FlipFlop: $q_1^t\ q_2^t \rightarrow q_1^{t+\tau}\ q_2^{t+\tau}$

1 0 stabil unter 0 1, 0 0

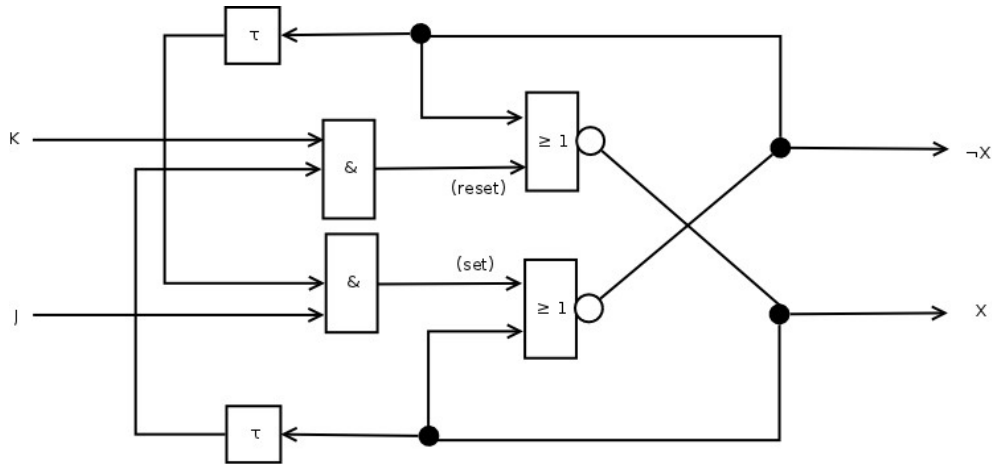
0 1 stabil unter 1 0, 0 0

0 0 stabil unter 1 1

Eingabe 1 1 verboten, wegen möglicher Race-Condition

q = 00; e = 11 → a: 00/10/01 möglich

JK-FlipFlop: (Jump-Kill)



Races: Äbhängigkeit des Folgezustandes von Zeitbedingungen bei Rückkoppelung

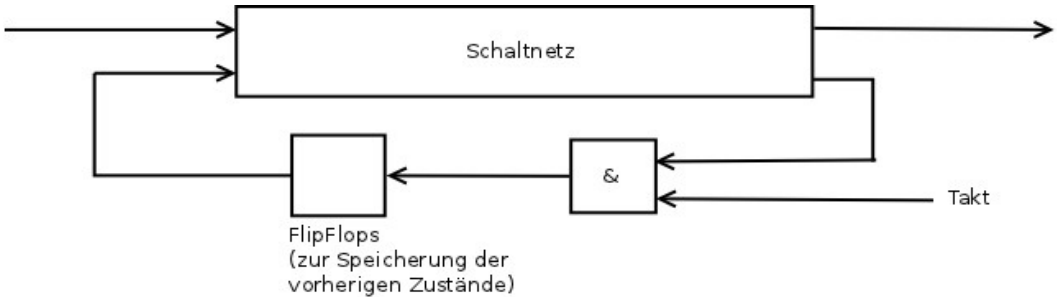
Fehler möglich, wenn: Q → Hazard;

q_1, q_2 nicht gleichzeitig ankommen

Setzen von 10 zu 01 ist nahezu unmöglich. Normalerweise: 10 → 00 → 01 oder 10 → 11 → 01

Critical Race: Schaltwerk irrt dauerhaft

Taktung:

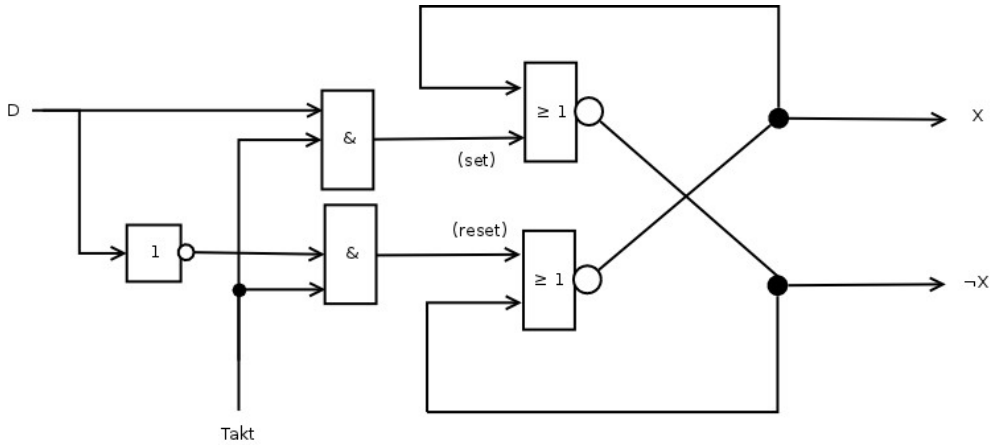


Gatterlaufzeit ~0,05ns → Taktperiode ~0,2 – 1ns (= 1 – 5 Ghz)

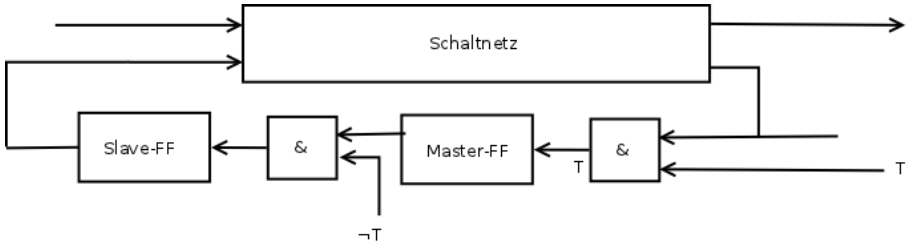
Häufig: Taktflankensteuerung

Takt auch für globale Synchronisation (Bsp. Befehlszähler & Rechenwerk)

D-FlipFlop: (Delay) $x^{t+\tau} = d^t$



Master-Slave-FlipFlop:



Rechner sind Schaltwerke:

- Deterministisch
- Großer Speicherraum (binär)
- Großer Zustandsraum ~2^{10*11} → endliche Maschine
- Charakteristische Begrenzung (kann z.B. π nicht beliebig genau berechnen)
- Zyklisches Verhalten (endliche Zustände, aber extrem viele → Leistungsfähig)

Speicherhierachie

Speichergröße ↔ Speichergeschwindigkeit → Speicherhierachie

Prozessorgeschwindigkeit vs. Speichergeschwindigkeit

Caches

Häufig benötigte Daten in kleinem, schnellen Speicher

- Nur Kopie in Cache
- Inhalt kann wechseln
- Cache wird transparent verwaltet → Zugriff erfolgt egal, ob sich Daten im Cache befinden oder nicht

Funktionsweise:

- Zeitliche Lokalität von Speicherzugriffen
- Räumliche Lokalität von Speicherzugriffen

Aufbau:

- Zeilen ~32-128 Bytes; Laden großer Datenblöcke ist relativ billig; sinnvoll wgn. Räumlicher Lokalität
- Adresstag: Link zum Hauptspeicher
- Statusbits z.B. valid

Prozessor	→	Adressbus	→	Hauptspeicher
		Cache	↓	
Prozessor	←	Datenbus	←	Hauptspeicher

Entwurfsalternativen:

- Platzierungs- / Ersetzungsstrategie: Wohin schreiben, was überschreiben?
- Aktualisierungsstrategie: Was passiert bei Cache-Hit/Cache-Miss
- Fully Associatvie / Direct Mapped / Set-Associative Cache

Cachearten:

- Direct Mapped Cache (Direkt abbildender Cache)
  - Cache Zeile aus (unteren Bits der) Adresse
  - Tag-Vergleich → Tag = Adresse
  - Problem: Speicherblöcke, die auf selbe Cache Zeilen abgebildet werden, können nicht gleichzeitig im Cache stehen
- Fully Associative Cache
  - Freie Anordnung der Speicherblöcke im Cache
  - Flexibler, aber aufwändiger
- Set-Associative Cache
  - Direct Mapped Caches, die Vollassoziativ angesprochen werden können
  - Mischform → Vorteile der Vollassoziativen Caches, aber nicht so teuer

Ersetzungsstrategie:

- Unbenutzte Zeile vorhanden → benutzen
- Sonst:
  - Zufällig
  - Least Recently Used → teuer
  - First In First Out

Aktualisierungsstrategie:

- Write Through
  - Bei schreiben → Hauptspeicher aktualisieren
  - Bei Write Hit: Auch Änderung im Cache → Konsistenz, aber kein Gewinn bei Schrbeizugriffen
  - Bei Write Miss: In Cache Laden und schreiben / Keine Cache Aktion
- Write Back, Copy Back
  - Bei schreiben → Nur Cache aktualisieren
  - Hauptspeicher wird erst bei Verdrängung aktualisiert
  - Dirty-Bit: Statusbit, das Veränderung zeigt

- Weniger Datenverkehr, aber Inkonsistenz

Klassen von Cache Misses

- Compulsory (cold) Miss: Erster Zugriff auf die Adresse
- Capacity Miss: Cache zu klein (auch in voll assoz. Cache)
- Conflict Miss: Nur in Mengenass. oder Direkt abbildendem Cache; Verdrängung bei Set-Gleichheit

Schleifenvertauschung:

```
A = int[ ][ ]
for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
                A[i][j] = 1;
        }
}
```

Es werden A[i][j] bis A[i][j+x] in den Cache geladen. Beim nächsten Durchlauf wird aber A[i+1][j] benötigt. → Vertauschung der Schleifen, um in den darauf folgenden Iterationen Cache Hits zu bekommen

Hardware Prefetching:

- Reduzierung von Misses
- Laden mehrerer Speicherblöcke; zweiter Block in Stream Buffer

Software Prefetching:

- Reduzierung von Misses
- Prefetch Instruktionen (von Compiler), nicht blockierend, keine Ausnahmen
- Unregelmäßige Zugriffe

Halbleiterspeicher

Statisches RAM (SRAM): FlipFlops (schnell, teuer, aufwändig → Caches)

- Kein Refresh nötig
- Zugriffszeit ~10-30ns

Dynamisches RAM (DRAM): Ladung in Kondensatoren (Leckströme → Auffrischung nötig, billig, langsamer, hohe Speicherdichte → Hauptspeicher)

- Lesen ist zerstörend → Rückschreiben
- Zykluszeit > Zugriffszeit
- SDRAM → Über Taktsignal synchronisiert; Anbindung über Northbridge Datenbreite: 64 Bit

(E)EPROM: Lading in isoliertem Gate (Einbringung durch hohe Spannung; Löschen durch UV-Licht oder elektrisch → Informationen bleiben auch ohne Betriebsspannung erhalten)

PROM (Programable Read Only Memory): zerstörbare Sicherungen

ROM: elektrische Verbindungen bei der Chip-Herstellung

FLASH Speicher

Speicherung von Daten auf dem Floating Gate eines Halbleitertransistors

Schreiben: Bei hoher Spannung zwischen Control und Source → Tunneln von Ladungen zum Floating Gate

Lesen: Ladung im Floating Gate unterbindet Strom zwischen Source und Drain → 0

Löschen: Hohe negative Spannung; Nur für Blöcke möglich; Oxidschicht leidet

NAND Flash

- Pages (512 / 2048 Bytes) werden ganz beschrieben; 200-300µs
- Löschen in größeren Blöcken; 1-2ms
- Lesen; 25µs für 4KB
- 10.000 – 1.000.000 Schreibzyklen → Wear leveling (Verteilung der Schreibzugriffe), Reservezellen

Ein-/Ausgabe

- Zeichenorientiert: Byteweise; i.Allg. Dialoggeräte: Tastatur, Bildschirm, Drucker
- Blockorientiert: Datenblöcke; i.Allg: Netzwerk, Externspeicher: Festplatte, Magnetband
- Anbindung meist über Controller (eigene Mikroprozessoren) zur Steuerung → Entlastung der CPU, Zeitkritische Aufgaben

Direct Memory Access

DMA Controller → Blockorientierte Geräte haben Speicherzugriff, ohne CPU zu belasten

Hintergrundspeicher

Erweiterung des Hauptspeichers / Archivierung → Festplatten, DVD, etc.

Magnetische Speicher:

0, 1 sind Richtung der Magnetisierung

Signal bei Wechsel der Magnetisierungsrichtung; lange Folge identischer Bits → spezielle Codierung (künstlicher Wechsel) + Prüfbits

Festplatten:

- Konzentrische Spuren; ~1-2µm → 5000-10000 Spuren pro cm
- Zwei Köpfe pro Platte
- Größere Spurlänge mit Zylinderdurchmesser → Äußere Zonen mit mehr Sektoren (gleich groß)

Optische Speicher:

Spur: Spirale, Daten durch Pits (¼ Wellenlänge Vertiefungen) dargestellt; Abtastung durch Laser

Übergänge sind dunkler (1); Kein Übergang 0

Erzeugung der Pits: CD-ROM – Presse; CD-R – Farbstoff, der Absorbtionsverhalten ändert; CD-RW – Kristaline Schicht, die Reflexionsverhalten ändert

RAID:

RAID 0: Performance, RAID 1: Redundanz

Virtueller Speicher:

Adressraum, der vom Betriebssystem einem Prozess zur Verfügung gestellt wird

- Logische (= virtuelle) Adressen → Umsetzung in physische Adressen durch Memory Management Unit (MMU) Nach Vorgaben des Betriebssystems (Segmentierung, Paging)
- Umsetzung bei jedem Speicherzugriff nötig

- → Unterstützung von Merhprozess- Mehrbenutzerfähigkeit im BS durch den Prozessor

Betriebssystem:

- Laden von Programmen (Speicher allokieren)
- evtl. Speicher auslagern
- Speicherschutz → Trennung der Daten verschiedener Prozesse
- Umrechnungstabellen (für Segmente und Seiten) durch Betriebssystem
- BS kann beliebigen physischen Speicher verwenden und wenn nötig auslagern.

Segmentierung:

- Virtueller Adressraum in logische Blöcke (Segmente) beliebiger Größe unterteilt; Bsp: Code, Daten, Stack
- Adressierung durch Segmentnummer und Offset
- Umsetzung durch Segment-Deskriptortabelle mit Anfangsadresse, Länge, Zugriffsrechten

Paging:

- Aufteilung in Seiten (feste Größe)
- Seitentabelle definiert für jede Seite mit physischer Adresse, Zugriffsrechten
- Typische Seitengröße: 4kB
  - groß → Kleinere Tabellen, effizientes ein-/auslagern, TLB effizienter
  - klein → weniger ungenutzter Speicher, weniger Aufwand zum Start kleiner Prozesse

Translation Lookaside Buffer:

- Cache für Adresstabellen (TLB)
- Seitenadresse als Tag, Datum ist physische Kacheladresse, auch Statusbits gespeichert
- TLB meist voll-/mengenassoziativ
- Typisch: 32-512 Einträge, Verschiedene TLBs für Befehls- / Datenzugriffe

Konzepte Moderner Mikroprozessoren

Pipelining

Zerteilung von Befehlen in Phasen, welche mit den Phasen anderer Befehle gleichzeitig ausgeführt werden.

Vgl. Fließbandfertigung der Industrie

Pipeline: Gesamtheit der Verarbeitungseinheiten

Pipeline-Stufe: Verarbeitungseinheit für eine Phase

Voraussetzungen:

- RISC Architektur mit einheitlichem Befehlsformat und Einzyklus-Maschinenbefehlen
- Befehlszyklus:
  - IF (Instruction Fetch);
  - ID (Instruction Decode)
  - EX (Execute)
  - MA (Memory Access)

    - WB (Write Back)
- Eigene Hardware für jede Stufe

Vorteile:

- Bessere Auslastung der Hardware
- Höherer Durchsatz (In jedem Takt wird eine Operation fertig) → Geschwindigkeitsgewinn

Nachteile:

- evtl. höhere Latenz (Bearbeitungszeit pro Befehl)
- Alle Stufen müssen gleich lang sein
- Komplexe Steuerung
- Pipline-Konflikte (Hazards) möglich
    - Nächste Instruktion kann nicht ausgeführt werden (Ressourcenkonflikt: 2 Stufen brauchen Hauptspeicher, Datenkonflikt: Daten benötigt, die noch nicht berechnet sind, Steuerkonflikt: 2 Stufen verändern Befehlszähler)
    - Ganze Pipeline wird angehalten
- Vermeidung von Konflikten:
    - Dynamisches Sheduling → Umsortierung der Instruktionen durch die Hardware, unabhängige Instruktionen werden vorgezogen
    - Sprungvorhersage → Spekulative Ausführung von Instruktionen bei bedingten Sprüngen

Superskalare Prozessoren

- Mehrere ALUs
- Parallele Bearbeitung skalarer Instruktionen
- Jeweils zwei Befehle befinden sich in der selben Stufe der Pipeline

Multithreading

Mehrerere Threads (Handlungsfäden) eines Prozesses, die sich dessen Ressourcen teilen zur Parallelisierung und Umsetzung nebenläufiger Vorgänge

- Bessere Ausnutzung der Funktionseinheiten → Überlappung von Letenzzeiten
- Instruktionen vollständig unabhängig
- Hardware: Muss Threadwechsel steuern; Mehrere Registersätze incl. Befehlszähler für schnellen Threadwechsel

Scheduler:

- Cycle by cycle: Pro Taktzyklus wird ein Thread ausgeführt
- Block Interleaving: Threads wird ausgeführt, bis ein Befehl mit langer Latenz kommt
- Simultaneous Multithreading (Hyperthreading): Mehrere Befehlspuffer, die den Befehlsstrom eines Befehls liefern; Jeder Befehlsstrom hat einen eigenen Registersatz

→ Threads unterschiedlicher Anwendungen; Durchsatzsteigerung

Multicore

Von-Neumann-Flaschenhals bei Steigerung der Rechengeschwindigkeit

→ Multicore statt schnellere Prozessoren

Mehrere Kerne profitieren von höherer Bandbreite, aber Programme müssen explizit parallelisiert geschrieben sein

Achtung: Cachekohärenz → Verhinderung von Inkonsistenzenz zwischen verschiedenen Caches.