1.EMR:

a.) Aufbau: -Wesentliche Rechnerbaugruppen auf einen chip
- interner Steuer- , Daten- und Adressbus
- E/A-Ports
- Schnittstellen
- Spezialfkt: Timer, A/D-Wandler,

b.) Untere Leistungsklasse:
- Interne Takt- und Reseterzeugung
- RAM: intern, klein, „schnell“, kombiniert mit Prozessorregistern

- Peripherie:_starke Konfigurierbarkeit
- ROM: (verschiedene Varianten)
- „Masken-ROM“
- PROM, OTP (programmierbarer ROM)
- EPROM (löschbar durch UV-Licht)
- EEPROM (elektrisch löschbar), Flash-EEPROM,
- „Bond-Out-Variante“

c.) Höhere Leistungsklasse:
- Datenbreite: 16 bit
- Adressraum: bis 2^{32} bit
- Interner Speicher: „größer“
- Leistungsfähiger Befehlssatz
- Viele E/A-Funktionen
- Starke Konfigurierbarkeit
- Teurer

Watchdog: Schaltung, die aufpasst, ob Prozessor ordentlich arbeitet, sonst Auslösen Hardwarereset

2.DSP:

a.) Merkmale: - allgemeine Eingebettete Systeme für hohen arithmetischen Durchsatz
- getrennte Daten- und Programmspeicher (höhere Rechenleistung)
- Echtzeitverarbeitung
- Kombinierte Befehle:

- Rechnen / Transport
- Add / Mult (MAC-Befehl / Multiplizieren Accumulieren)
- beliebige Kombinationen
- auch 3-Adress-Maschine möglich (3 Register + MAC-Befehl; Anwendung: Mobilfunk, Feststation)

b.) Anwendung:
- Analog In → A/D → DSP → D/A → Analog Out
- typisch:

- Tonsignalverarbeitung
- Bildsignalbearbeitung
- Bildbearbeitung (Bilderkennung)
- Digitale Regler

3.CISC-Architekturen (Complex Instruction Set Computing):

a.) Merkmale: - mächtige Maschinenbefehle
- internes Mikroprogrammwerk zur komplexen Befehlsverarbeitung
- direkte Verarbeitung der Mikrobefehle im Mikroprogrammwerk
- direkte Steuerbits im Bitvektor des Mikrobefehls
- Verzweigungsspezifikation zur Schleifenbildung verwendet (Sprünge)

b.) Bsp. für CISC-Architekturen:
- IBM 360
- IA-16, IA-32 ≡ X86
- Z80
- iAPx 432

c.) Gründe für die Beibehaltung von CISC:

- vorhandene Software
- Intern umwandeln von CISC in RISC durch Mikroprogrammwerk

4.RISC-Architekturen (Reduced Instruction Set Computing):

a.) Merkmale: - „Load-Store-Architektur“

→ Load – Datentransport: Speicher → Register

→ Store – Datentransport: Register → Speicher

→ bessere Optimierung

- einfache Maschinenbefehle von einheitlicher Mächtigkeit
- Architektur viel besser optimierbar
- Taktfrequenz erhöhen, innere Parallelisierung möglich
- höhere Leistungsfähigkeit mit mehr Befehlen
- benötigt mehr Maschinenbefehle
- Maschinenbefehle von einheitlicher Breite
- wenige verschiedene Formate

b.)Bsp. RISC-Architekturen:

- IA-64
- Power PC von Motorola
- Sparc
- Mips

c.)Vergleich CICSC RISC:

CISC	RISC
einfache und komplexe Befehle	Nur einfache Befehle (geringe Mächtigkeit / einheitlich)
Heterogener Befehlssatz (durcheinander, viele Ausnahmen, besondere Befehlsgruppen, unterschiedliche Handhabung	Orthogonaler Befehlssatz (systematisch aufgebaut) wenige Regeln + allgemeingehalten
Verschiedene Taktzahl pro Befehl	Meist 1 Takt pro Befehl (Lesen im externen Speicher braucht mehr)
Viele Befehlscode-Formate mit unterschiedlicher Länge	Wenige Befehlscode-Formate mit einheitlicher Länge
Mikroprogrammwerk	Direktverdrahtung
Vermischung von Verarbeitungsbefehlen und Speicherbefehlen	Trennung von Verarbeitungs- und Speicherbefehlen

5.**Pipelining:**

- Durchsatz: Wie oft tritt Befehlsende pro Zeiteinheit auf
- Latenz: Zeit, einen individuellen Befehl zu bearbeiten
- IF -> ID/OF -> EX -> WB (eine Befehlsbearbeitung)
- Prozessorbausteine:
  - IF: Instruction Fetch (holen des Behfels)
  - ID: Instruction Decode(Decodieren des Befehls)
  - OF: Operand Fetch(Operanten holen)
  - EX: Execute(ausführen)
  - WB: Write Back(zurück geben)

a.)Pipeline Stall(Operandenproblem):

- benötigter Operant von Befehl 2 wird in Befehl 1 noch berechnet
- Verzögerung im Befehl zwei nach IF aber kein löschen

Lösung: - bei Programmierung Datenkonflikte ausschliessen

- Bypass:- Register nicht zurück schreiben sonder weiterverwenden bei Datenkonflikt
  - zusätzliche Verbindung des ALU Ergebnis Puffer Registers mit Eingang

b.)Pipeline Flush(Sprungproblem):

- bei jump Befehl
- falscher nachfolge Befehl
- lösche alle Prozessorbausteine nach WB von Befehl 1
- aussetzten der Pipeline

Lösung (normalerweise keine Behandlung):

- vorgezogene Adressberechnung (Ergebnis des Sprungs wird in ID Phase berechnet somit kann nächster Befehl schon gefecht werden)
- Warteplätze (Delay Slots)
  - ADD
  - JMP
  - SMB (Warteplatz)
  - MML (Warteplatz)
- **statische Vorhersage:**
  an jedem Befehl 1 Bit → wird überwiegend ausgeführt → ja
  Befehl am Sprungziel wird geladen
  → wird überwiegend nicht ausgeführt → nein
  Befehl nach Sprungbefehl wird geladen
- **dynamische Vorhersage:**

a.)Branch-Prediction-Buffer:

- Intern im Prozessor Bit (endlich) für Sprungvorhersage
- je nach Prozessor (Automat) wechsel von 0 und 1 (nach n ausgeführten Sprüngen auf 1 n=(1;2))
- man braucht aber mehrere solcher Automaten für mehrere Sprungbefehle
- dynamisch wird den Sprungbefehlen die Automaten zugeordnet
- man braucht aber nur das Bit mehrmals und nicht den ganzen Automaten

b.)Branch-Target-Buffer:

- wie BPB ausserAdresse des Sprungzieles wird sich zusätzlich gemerkt)
- zu dem Zeitpunkt, wo Vorhersage gemacht wird, wird ja auch die Adresse berechnet → gleich Speichern und nicht nocheinmal berechnen

c.)Branch-Target-Cache:

- Speichert noch Befehle nach dem Sprung, weil die auch immer dieselben sind (Folie 3.90)

Superpipeline: mehr Phasen

6. Skalare Prozessoren:

- Prozessoren, bei denen zu einem Zeitpunkt 2 Befehlsausführung beginnen kann

a.)Einteilung: - Super skalare Architekturen

- VLIW-Architekturen (very large instruction word):
  - Befehle können zu grossen Befehl zusammengefasst werden (im Programmcode) Werden somit gleichseitig ausgeführt

Register Renaming –Umbennen von nicht benötigte Register in Benötigte Prozessor intern

b.)Typen von Datenabhängigkeiten:

- RAW – read after write
- WAR – write after read
- WAW – write after write
- RAR – read after read (Tausch von Registermöglichkeiten)

Bsp.: MOV B, A

RAW ┤
MOV C, B ┤ WAW
WAR ┤
MOV B, D

7.Out of Order Architektur:

- Befehl holen → Befehl dekodieren → Befehlspool → Verteiler → Ausführungseinheiten

8.Speicher:

a.)Addresspipelining:

- Durchsatz Verbesserung
- Während Daten von Adresse 1 geliefert werden kann die Adresse 2 schon auf Adressbus gelegt werden

Adresse: -Ad1-Ad2-Ad3-….. -Ad1-------Ad2------Ad3-....
Daten: ------Da1-Da2-Da3-..... -------Da1-------Da2------
mit Adresspipeline ohne Adresspipeline

b.)Interleaving:

- Verhinderung eines doppelten Zugiffs hintereinander auf die selbe Bank
- sehr geringe wahrscheinlichkeit das es doch passiert --> nur Zeitverlust (Lücke) für diesen Fall
- Adressen werden nicht nacheinander auf eine Bank geschrieben Sonder im Abwechselnt auf die nächste
- Grund: Doppelt zugriff auf eine Bank kann zu Zeitverlust führen

c.)Burstmode (Blockmodus):

- Vorrausschauendes einlesen von Befehlen/Speicherzellen
- RAS = Row Adress Select
- CAS = Column Adress Select

d.)Speicherfamilie mit Adresspipelineing, Burst Mode und Interleaving:

- Dynamisch (nur 1 Transistor pro Speicherzelle) :
- SDRAM: synchron/dynamisch --------> SDR (single(Dateneinheiten pro Takt) data rate); DDR (double .....); QDR (quad .....)
- RDRAM: RAMBUS
- Statisch (4 Transistoren pro Speicherzelle):
- SBSRAM: synchron burst static RAM

e.)Speicherhirarchie:

- tendenziel: grosser Speicher ---> langsam
  kleiner Speicher ---> schnell
- statischer RAM schneller als dynamischer bei gleicher Halbleitertechnologie
- deshalb cache klein

f.)Lokalität von Speicherzugriffen:

- temporale Lokalität: mehrmaliger Zugriff auf die selbe Adresse in begrenzter Zeit
- spatiale (örtliche) Lokalität: Zugriff auf benachbarte Adressen in begrenzter Zeit

==> statistische Aussage

- Cache entscheidet aufgrund der Aufrufhäufigkeit eines Befehls
- sogar komplette (kleinere) Schleifen im cache möglich
- Prozessor weiß nichts vom Cache

g.)Kenngrößen vom Cache:

- Treffer/Hit: Nc
- kein Treffer/Miss

- Trefferrate/Hitrate: T = Nc/Gesamtzahl der Zugriffe
- durchschnittliche Zugriffszeit: $t_r=(1-T)t_a+Tt_c$; $t_c<=t_r<=t_a$; c = cache ; a = Arbeitssp.

h.)Cache-Architekturen:

- direct mapped cache: direkt abbildender cache
- n-way-set-cache: Mehrweg cache
- fully association cache: Voll-Assoziativer-Cache

Schaltungsaufwand/Effektivität nimmt von oben nach unten zu

- cache – Tag: speichert die Zuordnungsinformation zwischen cache und Arbeitsspeicher
- cache – Datenbereich: ausgewählte Daten aus dem Speicher

i.)Strategien:

1. Ladestrategie:

- on demand, d.h. nur im Moment des Bedarfs
- pre demand, vor dem Bedarf mittels Wahrscheinlichkeit

===> Laden immer nur wenn ein MISS erfolgt

2.Ersetzungsstrategie:

- bei direct mapping cache nicht notwendig
- bei n-way-cache n-fache Entscheidung
- bei fully-ass.-cache volle Freiheit der zu ersetzenden Zelle

===>LRU(least recently use): die Zelle die am längsten keinen Hit mehr hatte ersetzten

===>pseudo LRU: nur für kurz zurückliegende Ereignisse Überblick bewahren

3.Schreibstategie:

a.) write trough: CPU==========Arbeitsspeicher
└==cache

CPU muss auf Arbeitsspeicher warten
--> kein Vorteil durch cache

b.)write back: - Phase 1 cache immer aktueller als RAM
- Phase 2 Controller aktualisiert RAM (CPU unabhängig)

----> Dirty Bit um bei Ersetzungsstrategie Zellen auszuschliessen deren Wert noch nicht zum Hauptspeicher zurückgeschrieben wurde (Vermeidung von Verdrängung)

c.)snooping:

cache<-----Controller<==========╗
Arbeitspeicher<╝====fremder Master
└---wait------┘

(Controller: beobachten der Zugriffe fremder Master auf den eigenen cache)

Konsistenz: Übereinstimmung cache Speicher

Koharenz: Übereinstimmung zwischen verschiedenen caches mittels snooping und MESI-Protokol