2. Prozessorarchitektur

RA1: Prozessorgrundstruktur mit Erweiterungen

- Steuerwerk
  Ablaufsteuerung (AST)
  Befehlsregister und
  Befehls-Dekoder
  Befehlsaderssregister (Befehlszähler)
  Prozessor-Statusregister
- Rechenwerk
  Arithmetik-Logik-Einheit (ALU)
  Datenregister (Operandenregister)

Befehlsausführung in 5 Phasen (F3)

1. Befehl lesen (Instruction Fetch, IF)
2. Befehl dekodieren (Instruction Decode, ID)
3. Operand lesen (Operand Fetch, OF, nur aus Register)
4. Befehl Ausführen (Execute, EX)
5. Operand Rückschreiben (Write Back, nur in Register)

2.1. RISC-Prozessoren

➔ hier RISC-Struktur

RISC: (Reduced Instruction Set Computer)

Ziel: Steuerwerk mit möglichst kleiner Logik -> schnellere Ausführung der Phasen.

Grundidee: nur einige, relativ einfache Befehle, komplexe Funktionen werden durch sequentielle Folgen dieser Befehle realisiert.

Ergebnisse:

- einfache und damit schnellere Logik

- gleichartigerAblauf für alle Befehe, logisch, zeitlich
- weniger verfügbare Maschinenbefehle, Compiler löst das, zugänglich für Optimierung
- 2 Gruppen von Befehlen:
  a) verarbeitende Befehle, ohne Operandenspeicherzugriff -> möglichst viele Operandenregister
  b) Operandentransportbefehle von und zum Speicher (Load, Store) ohne Verarbeitung, anderen Phasenablauf als verarbeitende Befehle

RISC ist sehr günstig für weiter Strukturen der Mikroparallelität.

2.2 CISC_prozessoren
CISC: Complex Instruction Set Computer
Grundidee:sehr viele Maschinenbefehle, von sehr einfach bis sehr komplex.

Ziele (u.a.) Befehle auf dem Niveau einer höheren Programmiersprache
Beispiel: Pico-Java-Prozessor (SUN)
➔ direkte Ausführung von Java-Bytecode
➔ ausgestorben (am Markt nicht erfolgreich)

Typische Realierung der sehr komplexen Ablaufsteuerung:
durch Mikroprogrammwerk (F4):
- jeder CISC-Befehl wird in eine seuentielle Folge von Mikrobefelen umgesetzt
- Mikrobefehle werden aus Mikroprogrammspeicher gelesen, Dekoder wählt Mikrobefehlsfolge aus, die aktuell in Ausführung befindlichen Mikrobefehle steuern die ALU, Register und Buss

- Mikrostaus aus Rechenwerk kann Verzeigungen und auch Schleifen im Mikroprogramm steuern
- Mikrobefehlszähler zeigt auf den gerade aktiven Mikrobefehl

CISC trennt nicht streng zwischen ausführenden Befehlen und Speichertransport und erzeugt dabei auch variable Operandenadressierung:
0-Operandenadressierung: alle E- und A-Operanden auf festen Plätzen (z.B. an der Spitze desw Stacks)
1-Operandenadressierung: Ein Operand und Ergebnis im festen Akkumulator, 2. Operand adressierbar
2-Operandenadressierung: ein adressierbares Register 1. Operand und Ergebnis, 2. Operand adressierbar
3-Operandenadressierung: 1., 2. Opearand und Ergebnis adressierbar.

Bei CISC können Operanden und Ergebnisse auch im Speicher liegen.

Gegenüberstellung

RISC	CISC
einfache Befehle	einfache bis komplexe Befehle
mehrere Maschinenbefehle für komplexe Funktionen	einen Befehl für komplexe Funktionen
größerer Programmcode	kleienerer Programmcode
Optimierung auf Maschinencode	nur auf Niveau der komplexen Befehle, nicht auf Mikrobefehls-Niveau optimierbar
einfache Ablaufsteuerung und Logik	Mirosequenzer, komplexe Logik:

- weniger Chipfläche
- geringerer Leistungsbedarf
regelmäßige logische und zeitliche Ausführung
-> günstig für Mikroparallelität

- mehr Chipfläche
- höherer Leistungsbedarf
unregelmäßige logische und zeitliche Ausfüphrung
-> ungünstig für Mikroparalellität

RISC eigentlich in fast allen Punkten die bessere Lösung
CISC-Befehlssätze insbesondere noch verbreitet aufgrund der Kompatibilität zu früheren Prozessoren,
z. B. moderne intel-Prozessoren zu Vorgängeren der i86-Familie

2.3. Befehlspipeline

Konventionelle Prozessoren ohne Pipeline arbeiten alle Phasen eines Befehls vollständig ab, bevor sie den nächsten Befehl beginnen
(F6 oben)

- ➔ bei einem Takt pro Phase: Befehl braucht 5 Takte und alle 5 Takte wird ein Befehl fertig

Die 5 Phasen benutzen weitgehen unabhängige Harwareteile im Prozessor, in jeder Phase werden die anderen Teile weitgehend nicht genutzt.

- ➔ verschachteltes Ausführen der Phasen möglich, d.h. nach IF des Befehls i kommt ID von Befehl i, parallel kann schon IF von Befehl i+1 ausgeführt werden usw. (F6 mitte)
- ➔ bei einem Takt pro Phase: Befehl braucht 5 Takte, aber ab dem 5. Takt wird in jedem Takt ein Befhl fertig, das bedeutet eine Leistungssteigerung um den Faktor 5

(F6 mitte)

Superpipeline: einzelen Phasen werden in Teilphasen zelegt (zb in je 2, F6 unten)

- ➔ je Takt wird ein Befehl fertig, Befehl hat 10 Take, aufgrund der Zerlegung der Phasen wird aber ein Takt mit 0,5 Taktperiode möglich
- ➔ Verdopplung der Leistung