Kap.2
Befehlsschnittstelle

Prozessoren,
externe Sicht

- **2.1** elementare Datentypen, Operationen
- **2.2** logische Speicherorganisation
- **2.3** Maschinenbefehlssatz
- **2.4** Klassifikation von Befehlssätzen
- **2.5** Unterbrechungen
- **2.6** Prozesse

Mikroprozessorklassen (1)

Mikroprozessorklassen (1)

- MPs mit komplexem Instruktionssatz (**CISC**=complex instruction set computer)
  - verfügen über Satz von zum Teil komplexen Befehlen
  - Struktur des Befehlssatzes ist fest vorgegeben und nicht veränderbar (Makrobefehle)

Mikroprozessorklassen (1)

- MPs mit komplexem Instruktionssatz (**CISC**=complex instruction set computer)
  - verfügen über Satz von zum Teil komplexen Befehlen
  - Struktur des Befehlssatzes ist fest vorgegeben und nicht veränderbar (Makrobefehle)
- Mikroprogrammierbare MPs
  - Satz von Mikrobefehlen steht zur Verfügung
  - können zur Definition von Makrobefehlen genutzt werden

Mikroprozessorklassen (2)

Mikroprozessorklassen (2)

- MPs mit reduziertem Befehlssatz (**RISC**=reduced instruction set computer)
  - einfache Befehle
  - in einem Verarbeitungsschritt ausführbar

Mikroprozessorklassen (2)

- MPs mit reduziertem Befehlssatz (**RISC**=reduced instruction set computer)
  - einfache Befehle
  - in einem Verarbeitungsschritt ausführbar
- MPs mit niedrigem Befehlssatz (**low-RISC**)
  - bis zu 16 Befehle
  - für dezidierte Problemlösung besonders gut angepaßt

Mikroprozessorklassen (2)

- MPs mit reduziertem Befehlssatz (**RISC**=reduced instruction set computer)
  - einfache Befehle
  - in einem Verarbeitungsschritt ausführbar
- MPs mit niedrigem Befehlssatz (**low-RISC**)
  - bis zu 16 Befehle
  - für dezidierte Problemlösung besonders gut angepaßt
- weitere Klassen

Mikroprozessorklassen (3)

- Im folgenden zunächst:

  CISC, RISC, CISC versus RISC

CISC (1)

CISC (1)

- Erster moderne Rechner (IBM 360, 1965) war CISC-Rechner

CISC (1)

- Erster moderne Rechner (IBM 360, 1965) war CISC-Rechner
- Charakterisierung:

CISC (1)

- Erster moderne Rechner (IBM 360, 1965) war CISC-Rechner
- Charakterisierung:
  - mächtige Befehlssätze (über 100 Befehle)

CISC (1)

- Erster moderne Rechner (IBM 360, 1965) war CISC-Rechner
- Charakterisierung:
  - mächtige Befehlssätze (über 100 Befehle)
  - komplexe Maschinenbefehle

CISC (1)

- Erster moderne Rechner (IBM 360, 1965) war CISC-Rechner
- Charakterisierung:
  - mächtige Befehlssätze (über 100 Befehle)
  - komplexe Maschinenbefehle
  - viele Adressierungsarten

CISC (1)

- Erster moderne Rechner (IBM 360, 1965) war CISC-Rechner
- Charakterisierung:
  - mächtige Befehlssätze (über 100 Befehle)
  - komplexe Maschinenbefehle
  - viele Adressierungsarten
  - mehrere Datentypen

CISC (1)

- Erster moderne Rechner (IBM 360, 1965) war CISC-Rechner
- Charakterisierung:
  - mächtige Befehlssätze (über 100 Befehle)
  - komplexe Maschinenbefehle
  - viele Adressierungsarten
  - mehrere Datentypen
  - mikrokodierter Befehlssatz (Steuerung durch Mikroprogramm)

CISC (2)

CISC (2)

- Orthogonaler Befehlssatz, d.h. jeder Datentyp kann mit vielen Adressierungsarten angesprochen werden

CISC (2)

- Orthogonaler Befehlssatz, d.h. jeder Datentyp kann mit vielen Adressierungsarten angesprochen werden
- wenige Register

CISC (2)

- Orthogonaler Befehlssatz, d.h. jeder Datentyp kann mit vielen Adressierungsarten angesprochen werden
- wenige Register
- Maschinenbefehle haben unterschiedliche Ausführungszeiten

CISC (2)

- Orthogonaler Befehlssatz, d.h. jeder Datentyp kann mit vielen Adressierungsarten angesprochen werden
- wenige Register
- Maschinenbefehle haben unterschiedliche Ausführungszeiten
- Mikroprogramm als Interpretationsebene (Mikroprogrammsteuerwerk kontrolliert Befehlsausführung)

CISC (3)

CISC (3)

- Argumente für CISC:

CISC (3)

- Argumente für CISC:
  - komplexe Befehle schließen **semantische Lücke** zwischen Software und Hardware

CISC (3)

- Argumente für CISC:
  - komplexe Befehle schließen **semantische Lücke** zwischen Software und Hardware
  - Diskrepanz zw. interner Verarbeitungsgeschwindigkeit und Speichergeschwindigkeit reduziert

CISC (3)

- Argumente für CISC:
  - komplexe Befehle schließen **semantische Lücke** zwischen Software und Hardware
  - Diskrepanz zw. interner Verarbeitungsgeschwindigkeit und Speichergeschwindigkeit reduziert
  - Komplexe Befehle reduzieren Speicherplatzbedarf (bezogen auf Hauptspeicher)

CISC (3)

- Argumente für CISC:
  - komplexe Befehle schließen **semantische Lücke** zwischen Software und Hardware
  - Diskrepanz zw. interner Verarbeitungsgeschwindigkeit und Speichergeschwindigkeit reduziert
  - Komplexe Befehle reduzieren Speicherplatzbedarf (bezogen auf Hauptspeicher)
  - komplexe Befehle **vereinfachen Compiler**

CISC (3)

- Argumente für CISC:
  - komplexe Befehle schließen **semantische Lücke** zwischen Software und Hardware
  - Diskrepanz zw. interner Verarbeitungsgeschwindigkeit und Speichergeschwindigkeit reduziert
  - Komplexe Befehle reduzieren Speicherplatzbedarf (bezogen auf Hauptspeicher)
  - komplexe Befehle **vereinfachen Compiler**
  - **Zuverlässigkeit** (mehr Funktionalität unterhalb der Maschinensprache)
    - „Hardware ist sicher, Software enthält Fehler"

CISC (4)

CISC (4)

- Argumente gegen CISC:

CISC (4)

- Argumente gegen CISC:
  - Hardware ist **nicht** zuverlässiger als Software
    - aber schwieriger abzuändern bei Fehlern

CISC (4)

- Argumente gegen CISC:
  - Hardware ist **nicht** zuverlässiger als Software
    - aber schwieriger abzuändern bei Fehlern
  - Cache-Strategien für Programmspeicher sehr gut
    - fast alle Zugriffe auf Programmteile in einem Mikroprogrammzyklus möglich

Beispiele

Beispiele

- **Motorola MC68000 (1979)**

Beispiele

- **Motorola MC68000 (1979)**
  - 16-Bit Prozessor, interne Struktur 32-Bit

Beispiele

- **Motorola MC68000 (1979)**
  - 16-Bit Prozessor, interne Struktur 32-Bit
  - Charakterisierung
    - allgemeiner Registersatz
    - 2 Stacks
    - 2 Betriebszustände
    - privilegierte Befehle
    - Adressraum von 16 MByte

Motorola (2)

Motorola (2)

- für Anwender benutzbare Register
  - 8 32-Bit Datenregister (D0 bis D7)
  - 7 32-Bit Adreßregister (A0 bis A6)
  - 2 32-Bit-Stackpointerregister (A7 und A7‘)
  - 1 16-Bit Statusregister
  - 1 32-Bit Befehlsregister

Motorola (2)

- für Anwender benutzbare Register
  - 8 32-Bit Datenregister (D0 bis D7)
  - 7 32-Bit Adreßregister (A0 bis A6)
  - 2 32-Bit-Stackpointerregister (A7 und A7‘)
  - 1 16-Bit Statusregister
  - 1 32-Bit Befehlsregister
- Datenformate
  - Byte (8 Bit), Wort (16 Bit), Doppelwort (32 Bit)
  - Doppelworttransport erfordert zwei Zugriffe
    - da 16-Bit Datenbus
  - Wörter und Doppelwörter an geraden Adressen

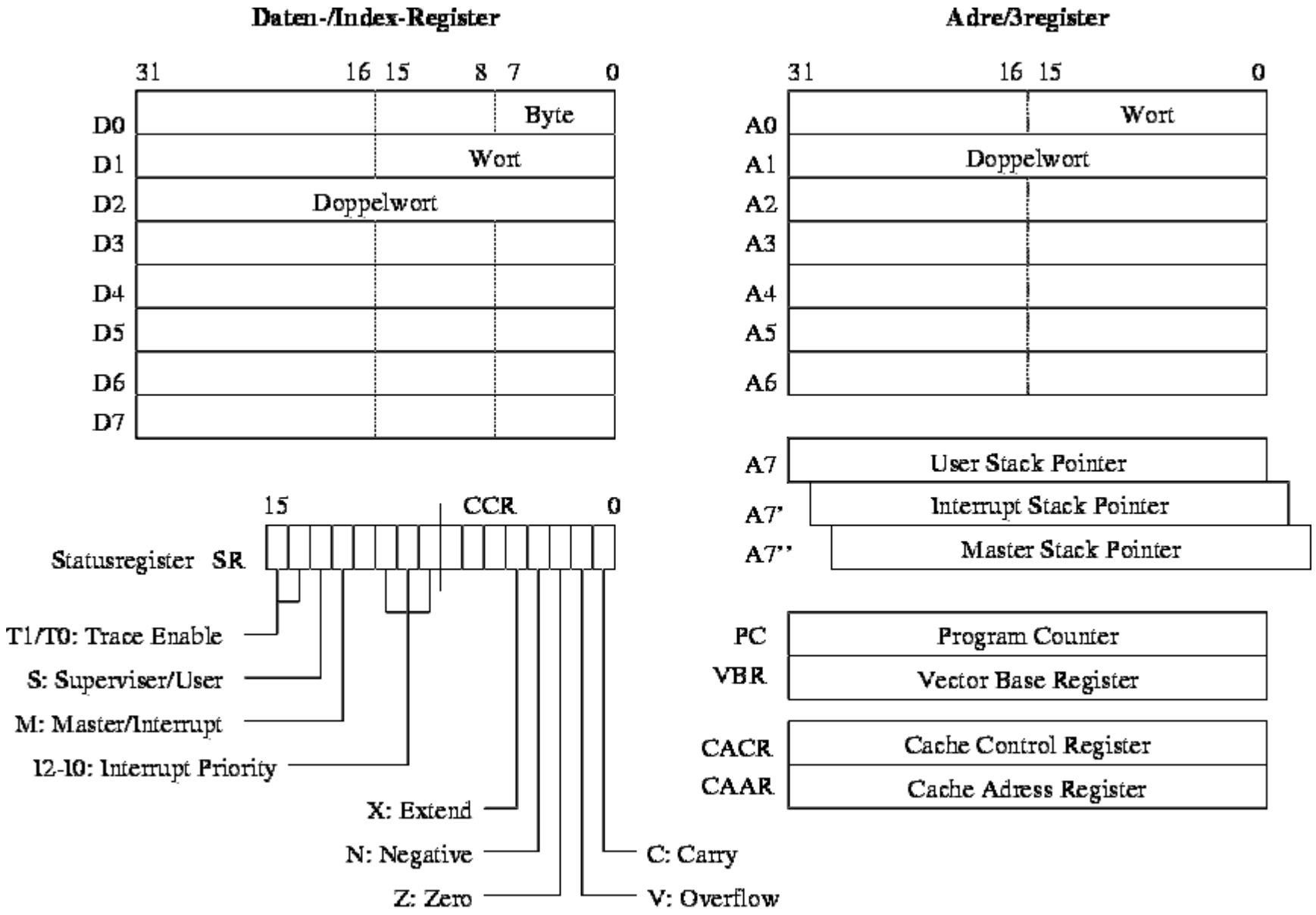


Abbildung 2.10: Registersatz der Motorola 680x0-Prozessoren

Motorola (3)

Motorola (3)

- Addressierungsarten
  - 14 werden unterstützt
  - Addressierungart nicht an Befehlstyp gebunden

Motorola (3)

- Addressierungsarten
  - 14 werden unterstützt
  - Addressierungart nicht an Befehlstyp gebunden
- Befehlsformate und Befehlssatz
  - über 70 Befehle
  - bestehen aus 1 bis 5 16-Bit Wörtern

Motorola (4)

Motorola (4)

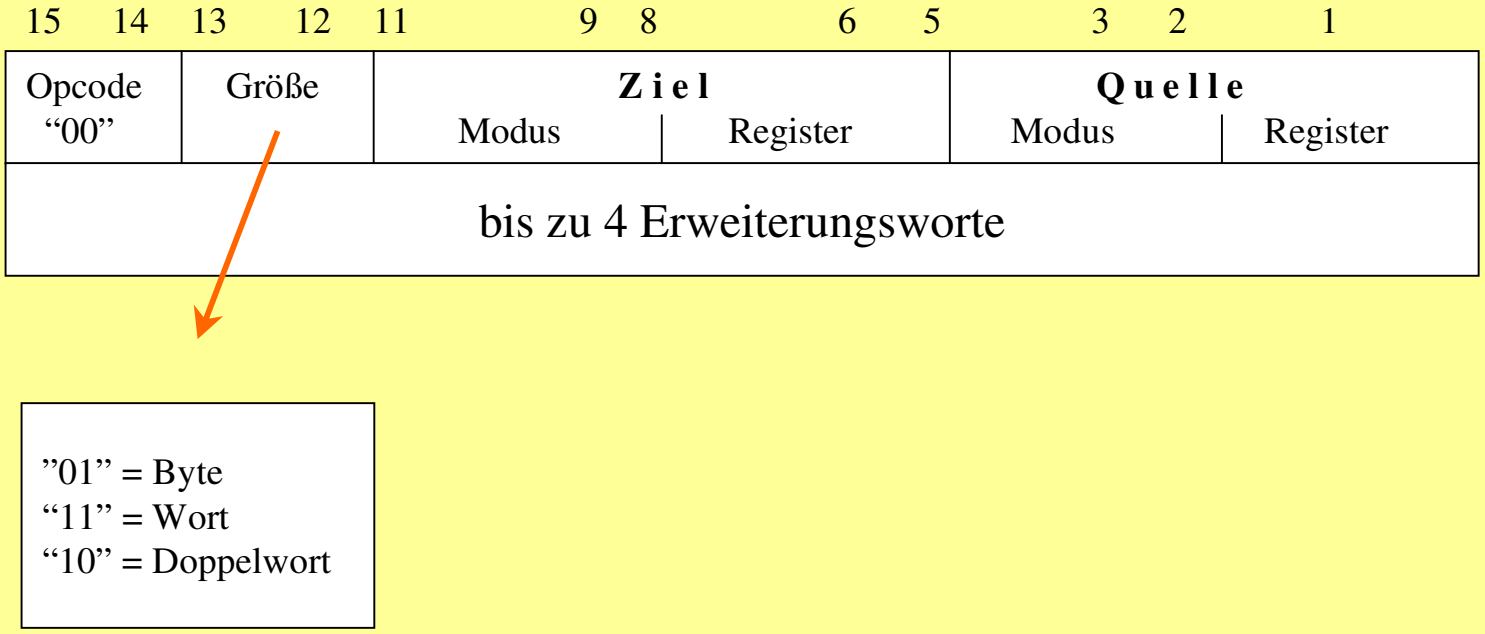
- Format der 68000-Maschinenbefehle am Beispiel des MOVE-Befehls

Motorola (4)

- Format der 68000-Maschinenbefehle am Beispiel des MOVE-Befehls

Motorola (4)

- Format der 68000-Maschinenbefehle am Beispiel des MOVE-Befehls



MOVE-Befehle des 68000

Modus	Register-Feld	Erweit.-worte	Assembler-Notation	Adressierungstyp	effektiver Operand ggf. Seiteneffekt
"000"	n	0	Dn	Register-Adressierung	D[n]
"001"	n	0	An	(Adreß-)Register-Adress.	A[n]
"010"	n	0	(An)	(Adreß-)Register indir.	Speicher[A[n]]
"011"	n	0	(An)+	(Adreß-)Register indir. mit Postincrement;i=1,2,4	Speicher[A[n]]; A[n]:=A[n]+i
"100"	n	0	-(An)	(Adreß-)Register indir. mit Prädecrement	A[n]:=A[n]-i; Speicher[A[n]]
"101"	n	1	d(An)	Relative A. mit 16-Bit Distanz	Speicher[d+A[n]]
"110"	n	1	d(An,Xm)	Register-Relative Adr. mit Index; Xm kann Daten- oder Adreßregister sein	Speicher[d+A[n]+X[m]]
"111"	"000"	1	d	direkte Adressierung	Speicher[d]
"111"	"001"	2	d	direkte Adr. (32 Bit)	Speicher[d]
"111"	"010"	1	d(PC),d(*)	Programmzähler-relativ	Speicher[PC+d+2]
"111"	"011"	1	d(*,Xn)	Programmzähler-relativ mit Index	Speicher[PC+d+2+X[n]]
"111"	"100"	1-2	#zahl	unmittelbare Adressierung	zahl

MOVE-Befehle des 68000

Modus	Register-Feld	Erweit.-worte	Assembler-Notation	Adressierungstyp	effektiver Operand ggf. Seiteneffekt
"000"	n	0	Dn	Register-Adressierung	D[n]
"001"	n	0	An	(Adreß-)Register-Adress.	A[n]
"010"	n	0	(An)	(Adreß-)Register indir.	Speicher[A[n]]
"011"	n	0	(An)+	(Adreß-)Register indir. mit Postincrement;i=1,2,4	Speicher[A[n]]; A[n]:=A[n]+i
"100"	n	0	-(An)	(Adreß-)Register indir. mit Prädecrement	A[n]:=A[n]-i; Speicher[A[n]]
"101"	n	1	d(An)	Relative A. mit 16-Bit Distanz	Speicher[d+A[n]]
"110"	n	1	d(An,Xm)	Register-Relative Adr. mit Index; Xm kann Daten- oder Adreßregister sein	Speicher[d+A[n]+X[m]]
"111"	"000"	1	d	direkte Adressierung	Speicher[d]
"111"	"001"	2	d	direkte Adr. (32 Bit)	Speicher[d]
"111"	"010"	1	d(PC),d(*)	Programmzähler-relativ	Speicher[PC+d+2]
"111"	"011"	1	d(*,Xn)	Programmzähler-relativ mit Index	Speicher[PC+d+2+X[n]]
"111"	"100"	1-2	#zahl	unmittelbare Adressierung	zahl

Intel 8086

Intel 8086

- aufwärtskompartibel zu 8-Bit Prozessor 8080A/8085

Intel 8086

- aufwärtskompartibel zu 8-Bit Prozessor 8080A/8085
- für 8- und 16-Bit Verarbeitung ausgelegt

Intel 8086

- aufwärtskompartibel zu 8-Bit Prozessor 8080A/8085
- für 8- und 16-Bit Verarbeitung ausgelegt
- Charakterisierung
  - segmentierte Speicheradressierung
  - dynamisches Verschieben von Programm-, Daten- und Speicherbereichen
  - Interrupt-System
  - Unterstützung von Mehrprozessorbetrieb
  - Adreßraum von 1 MByte

Intel (2)

Intel (2)

- für Anwender benutzbare Register
  - 4 allgemeine Register
  - 4 Pointer- und Indexregister
  - 4 Segmentregister
  - 1Statusregister
  - 1Befehlszähler

Intel (2)

- für Anwender benutzbare Register
  - 4 allgemeine Register
  - 4 Pointer- und Indexregister
  - 4 Segmentregister
  - 1Statusregister
  - 1Befehlszähler
- im Gegensatz zu MC68000 Verwendungs-möglichkeit der Register eingeschränkt und genau festgelegt

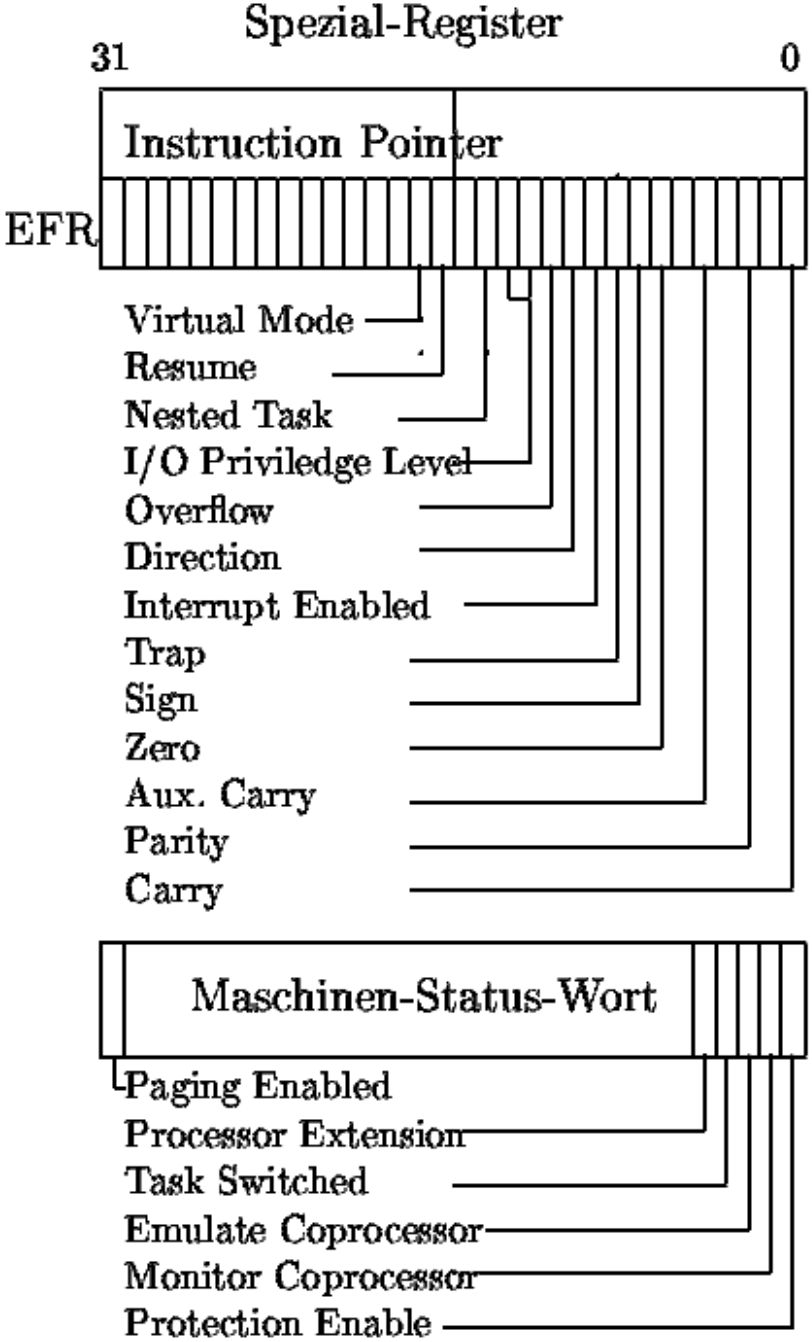
Allgemeine Register

	31	0
EAX	Arithm. Ergeb.	AX (AH | AL)
EDX	und Ein/Ausgabe	DX (DH | DL)
ECX	Zähl-Register	CX (CH | CL)
EBX	Basis-Register	BX (BH | BL)
EBP	Basis-Register	BP
ESI	Index-Register	SI
EDI	Index-Register	DI
ESP	Stack-Pointer	SP

Segmentregister

CS	Code-Segment
SS	Stack-Segment
DS	Daten-Segment
ES	Daten-Segment
FS	Daten-Segment
GS	Daten-Segment

Spezial-Register



Intel (3)

Intel (3)

- Datenformate
  - arithmetische und logische Operationen können Byte- oder wortweise durchgeführt werden
  - weitere Datenformate
    - dezimale Operanden in BCD-Darstellung (gepackte Form) und ASCII-Darstellung (ungepackte Form)
    - Byte- und Wortketten

Intel (3)

- Datenformate
  - arithmetische und logische Operationen können Byte- oder wortweise durchgeführt werden
  - weitere Datenformate
    - dezimale Operanden in BCD-Darstellung (gepackte Form) und ASCII-Darstellung (ungepackte Form)
    - Byte- und Wortketten
- Adressierungsarten
  - 24 werden unterstützt

Intel (3)

- Datenformate
  - arithmetische und logische Operationen können Byte- oder wortweise durchgeführt werden
  - weitere Datenformate
    - dezimale Operanden in BCD-Darstellung (gepackte Form) und ASCII-Darstellung (ungepackte Form)
    - Byte- und Wortketten
- Adressierungsarten
  - 24 werden unterstützt
- Befehlsformate und Befehlssatz
  - 77 Befehle und bestehen aus 1 bis 6 Byte

CISC-Probleme (1)

CISC-Probleme (1)

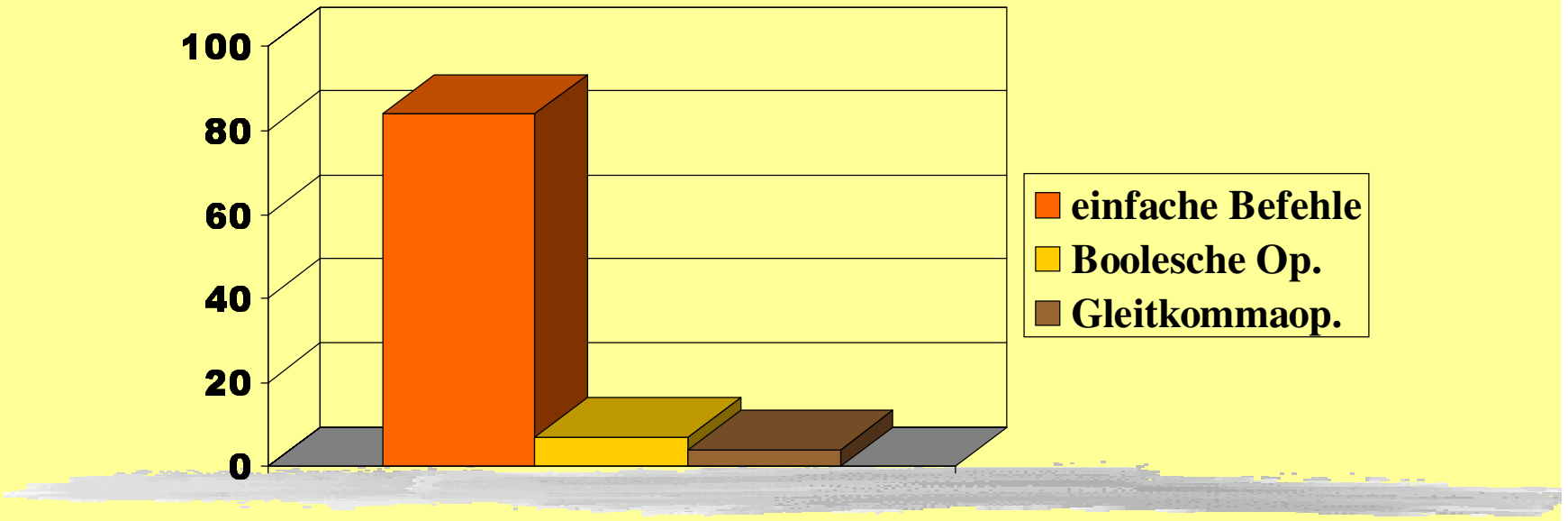
- komplexe Befehle werden (von Compiler) kaum genutzt

CISC-Probleme (1)

- komplexe Befehle werden (von Compiler) kaum genutzt
- Statistik auf DEC VAX 11/780

CISC-Probleme (1)

- komplexe Befehle werden (von Compiler) kaum genutzt
- Statistik auf DEC VAX 11/780



CISC-Probleme (2)

CISC-Probleme (2)

- Mehr als 90% der ausgeführten Befehle sind **einfach**
  - können schnell ausgeführt werden

CISC-Probleme (2)

- Mehr als 90% der ausgeführten Befehle sind **einfach**
  - können schnell ausgeführt werden
- Unterstützung der restlichen 10% durch komplexe Maschinenbefehle **erhöht** idR Ausführungszeit der 90%
  - 10% der auszuführenden Befehle schneller machen zu Lasten der übrigen 90%?

Designprinzip schneller Rechner

Designprinzip schneller Rechner

- Was sind die **Schlüsseloperationen**?

Designprinzip schneller Rechner

- Was sind die **Schlüsseloperationen**?
- **Datenpfadentwurf** für diese Operationen

Designprinzip schneller Rechner

- Was sind die **Schlüsseloperationen**?
- **Datenpfadentwurf** für diese Operationen
- Entwurf der Maschinenoperationen
  - jede Operation in einem Datenpfadzyklus ausfürbar -> kein Mikroprogramm vonnöten

Designprinzip schneller Rechner

- Was sind die **Schlüsseloperationen**?
- **Datenpfadentwurf** für diese Operationen
- Entwurf der Maschinenoperationen
  - jede Operation in einem Datenpfadzyklus ausfürbar -> kein Mikroprogramm vonnöten
- Erweiterung des Befehlssatzes nur dann, wenn Maschine dadurch nicht langsamer wird

RISC (1)

RISC (1)

- Nur **wichtige** Befehle werden realisiert

RISC (1)

- Nur **wichtige** Befehle werden realisiert
- **RISC** kein neuer Prozessortyp
  - idR von-Neumann-Architektur

RISC (1)

- Nur **wichtige** Befehle werden realisiert
- **RISC** kein neuer Prozessortyp
  - idR von-Neumann-Architektur
- **Kleine** Befehlsätze (max. 100 Befehle)

RISC (1)

- Nur **wichtige** Befehle werden realisiert
- **RISC** kein neuer Prozessortyp
  - idR von-Neumann-Architektur
- **Kleine** Befehlsätze (max. 100 Befehle)
- Wenige (notwendige) Adressierungsarten
  - nur zwei Befehle für Speicherzugriff
    LOAD/STORE-Architektur

RISC (1)

- Nur **wichtige** Befehle werden realisiert
- **RISC** kein neuer Prozessortyp
  - idR von-Neumann-Architektur
- **Kleine** Befehlsätze (max. 100 Befehle)
- Wenige (notwendige) Adressierungsarten
  - nur zwei Befehle für Speicherzugriff LOAD/STORE-Architektur
- Verzicht auf Mikrocode
  - festverdrahtetes Steuerwerk

RISC (2)

RISC (2)

- Befehle in **einem** Zyklus verarbeiten

RISC (2)

- Befehle in **einem** Zyklus verarbeiten
- Aufwandsverlagerung von Hardware in die Compiler

RISC (2)

- Befehle in **einem** Zyklus verarbeiten
- Aufwandsverlagerung von Hardware in die Compiler
- **Synergie** zwischen Hardware und Compiler (anstatt interpretierende Mikroprogramme)

RISC (2)

- Befehle in **einem** Zyklus verarbeiten
- Aufwandsverlagerung von Hardware in die Compiler
- **Synergie** zwischen Hardware und Compiler (anstatt interpretierende Mikroprogramme)
- Großer interner Registersatz

RISC (2)

- Befehle in **einem** Zyklus verarbeiten
- Aufwandsverlagerung von Hardware in die Compiler
- **Synergie** zwischen Hardware und Compiler (anstatt interpretierende Mikroprogramme)
- Großer interner Registersatz
- Leistungsfähige Speicherhierarchie

RISC (3)

RISC (3)

- Einsparung des Mikrobefehlssatzes
  - Performance-Gewinn durch Einsparung einer Transformationsebene

RISC (3)

- Einsparung des Mikrobefehlssatzes
  - Performance-Gewinn durch Einsparung einer Transformationsebene
- Schnelles Dekodieren der Befehle
  - durch einfaches Befehlsformat

RISC (3)

- Einsparung des Mikrobefehlssatzes
  - Performance-Gewinn durch Einsparung einer Transformationsebene
- Schnelles Dekodieren der Befehle
  - durch einfaches Befehlsformat
- Schnelle Ausführung
  - Register-orientiert durch LOAD/STORE
  - meisten Instruktionen in einem Zyklus ausführbar

RISC (4)

RISC (4)

- Optimierende Compiler
  - wegen einfacher Hardware

RISC (4)

- Optimierende Compiler
  - wegen einfacher Hardware
- *Kürzere Entwurfszeit*
  - einfaches Design

RISC (4)

- Optimierende Compiler
  - wegen einfacher Hardware
- *Kürzere Entwurfszeit*
  - einfaches Design
- Große Speicher verfügbar
  - mehr speicherverbrauchende Ansätze von Architekturlösungen werden realisiert
  - Optimierung der Zeiteffizienz anstelle der Speichereffizienz

Beispiel:

Beispiel:

- **MIPS RS2000/3000 (1982-89)**

Beispiel:

- **MIPS RS2000/3000 (1982-89)**
  - 32-bit RISC Prozessor

Beispiel:

- **MIPS RS2000/3000 (1982-89)**
  - 32-bit RISC Prozessor
  - 32 weitgehend homogene 32-bit Register (bis auf Register mit Nr. 0 und 1)

Beispiel:

- **MIPS RS2000/3000 (1982-89)**
  - 32-bit RISC Prozessor
  - 32 weitgehend homogene 32-bit Register (bis auf Register mit Nr. 0 und 1)
  - 32 32-bit floating-point-Register (einzeln für single precision, paarweise für double precision)

MIPS (2)

- LOAD/STORE Architektur
- alle Befehle haben 32-bit Format
- arithm. Befehle sind vom Typ RRR (3-Adress-Befehle auf Reg. Beschränkt)
- Assembler hat Pseudo-Befehle (werden durch Sequenzen von Maschinenbefehlen ersetzt)

MIPS(3)

Befehlsformate der Rechner MIPS R2000/3000

	Größe [bit]	6	5	5	5	5	6
Arithmetische Befehle	Format R	op	rs	rt	rd	shamt	funct
Immediate + LOAD/STORE	Format I	op	rs	rt	adr-teil		
Bedingte Sprungbefehle	Format J	op	Sprungadresse				

RISC versus CISC (1)

- Einfache Instruktion-en (in einem Zyklus ausführbar)
- Befehle werden durch Hardware ausgeführt
- Einfache Instruktionsformate

- Zum Teil komplexe Instruktionen (benö-tigen z.T. mehrere Zyklen)
- Einsatz eines Mikroprogramms
- Variable Instruktionsformate

RISC versus CISC (2)

- LOAD/STORE-Architektur
- Nur wenige Instruktionen
- Komplexität in den Compilern

- Viele Befehle dürfen Hauptspeicher adressieren(z.T. sehr aufwendige Adressierungsmodi)
- Viele Instruktionen
- Komplexität in der Hardware

Weitere Mikroprozessorklassen
Klassen von Befehlssätzen

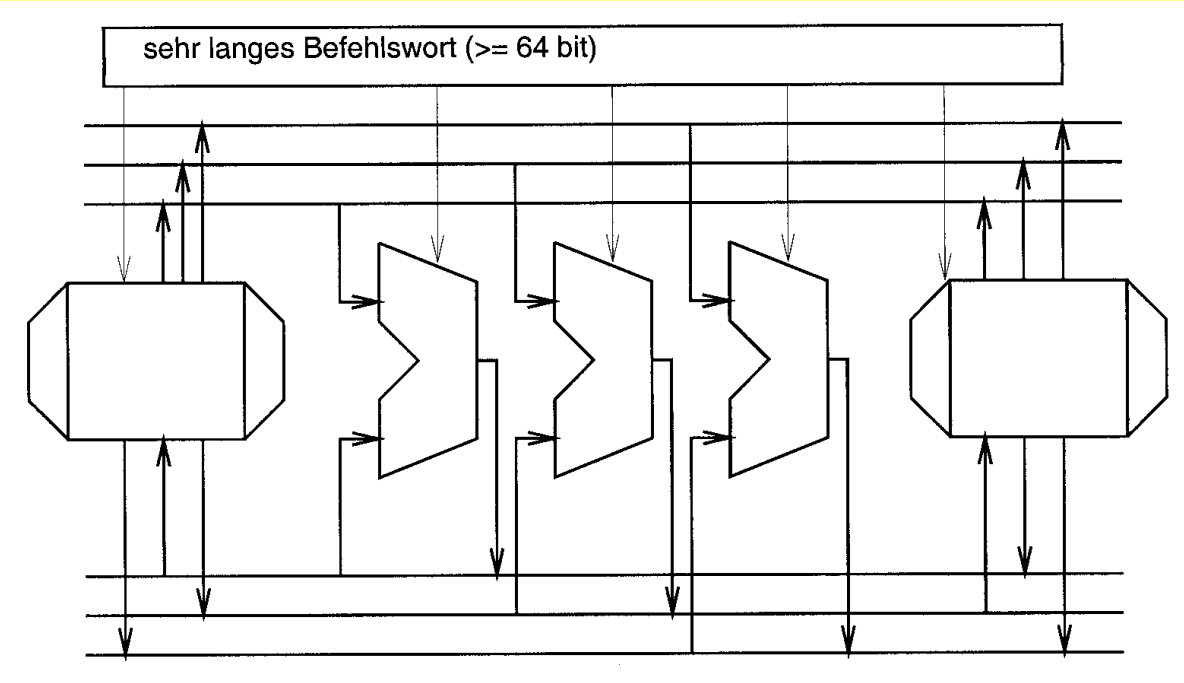
Weitere Mikroprozessorklassen
Klassen von Befehlssätzen

- DSP-Befehlssätze
  - eingesetzt in eingebetteten Systemen
  - angepaßt an die spezifischen Aufgaben

Weitere Mikroprozessorklassen
Klassen von Befehlssätzen

- DSP-Befehlssätze
  - eingesetzt in eingebetteten Systemen
  - angepaßt an die spezifischen Aufgaben
- EPIC- und VLIW- Befehlssätze
  - größere Zahl von funktionellen Einheiten
  - Speicher mit mehreren Ports
  - „breite" Befehle steuern alle Einheiten gleichzeitig

VLIW-Architektur (Beispiel)



Weitere Mikroprozessorklassen
Klassen von Befehlssätzen

Weitere Mikroprozessorklassen
Klassen von Befehlssätzen

- Multimedia-Befehle
  - Erweiterung bei RISC und CISC

Weitere Mikroprozessorklassen
Klassen von Befehlssätzen

- Multimedia-Befehle
  - Erweiterung bei RISC und CISC
- ASIPs

Weitere Mikroprozessorklassen
Klassen von Befehlssätzen

- Multimedia-Befehle
  - Erweiterung bei RISC und CISC
- ASIPs
- Abstrakte Maschinen
  - Java Abstract Machine