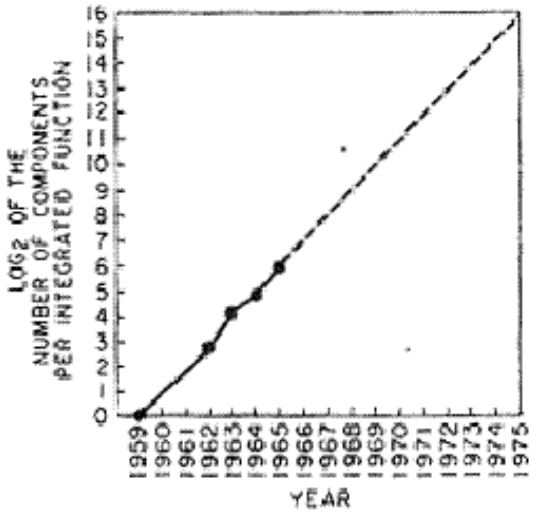
Scheller, besser, komplexer, einfacher,....

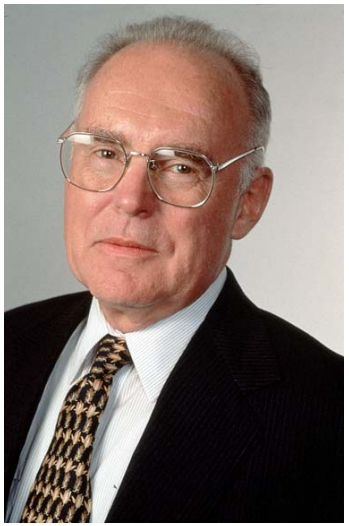
Alternativen beim Entwurf eines Prozessors an Beispielen

Prozessortechnologie



Gordon Moores Gesetz

Cramming more components onto integrated circuits,
Electronics Volume 38, Number 8,
April 19, 1965



Rechnerarchitektur

Moore‘sches Gesetz (von 1965 !!):

Alle 18 Monate verdoppelt sich die Zahl der Transistoren auf einem (Speicher-) Chip.

Was macht man mit den ganzen Transistoren?

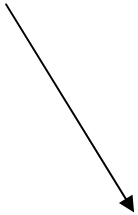
Wie geht es weiter? Längere Worte? Mehr Instruktionen? Kompliziertere Adressierung?

Welche Entscheidungen müssen Designer treffen?

Was sind die relevanten Entwicklungsrichtungen?

Wie organisiert man Chips in denen Signale innerhalb eines Taktes nicht mehr von einem Ende des Chips zum andern kommen?

CISC vs. RISC



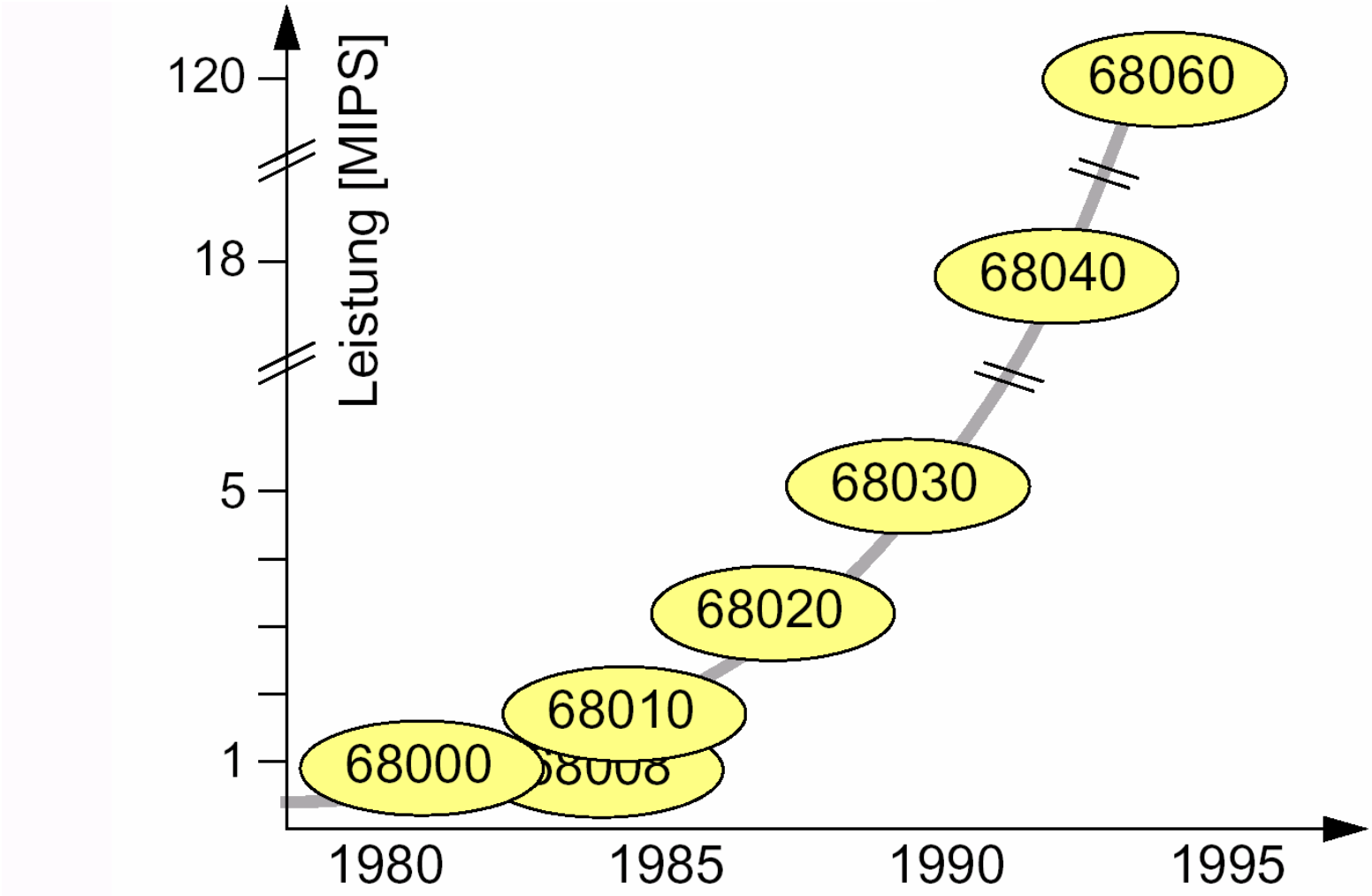
Mikroprogrammierung
Speicher-Register-Architektur
Komplexe Befehle und Adressierungsarten
Mehrzyklen-Befehle

680x0 Familie
80x86 Familie

keine Mikroprogrammierung
keine Speicher-Register-Architektur
keine Komplexe Befehle und Adressierungsarten
keine Mehrzyklen-Befehle

ALPHA, ARM, MIPS, SPARC,

Die 68K-Familie



◆ MIPS = Million Instructions per Second

Was sind die charakteristischen Eigenschaften der 68K-Familie?

- 32-Bit Architektur
- Mehr-Register-Architektur
- Separierung von Code-und Datenadreßraum
- Privilegebenen und Speicherschutz
- Sehr flexibles Bussystem mit Mehrprozessorunterstützung (dyn. Busarbitrierung)
- Co-Prozessor Schnittstelle (ab 68020)

68020 Überblick:

- Architektur - Programmiermodell, Ausführungsmodi
- Adressierungsarten, Befehlssatz

- Bussystem
- Coprozessor-Schnittstelle

68020 Eigenschaften:

“Echter“ 32-Bit-Prozessor (68008 [24/8], 68000 [24/16])

4 Gbyte linearer, nicht segmentierter Adreßraum

32-Bit PC

8 Datenregister

8 Adreßregister (Adreßregister #7 ist der Stackpointer für Unterprogrammaufrufe)

2 Supervisor-Stack Pointer (Master- und Interrupt-SP)

5 Spezial-Kontrollregister

18 Adressierungsarten (9 Basistypen)

7 Datentypen

Flexible Busstruktur

Coprozessor-Schnittstelle

Instruktions-Cache

Zielkonflikte bei der Realisierung eines Registersatzes:

Anzahl der Register:

Zitat: Entweder ein Register (Acc) oder unendlich viele !

Unterschiede in der Bedeutung und im Gebrauch der Register:

Ein einziges Register: Zwischenpeicherung von Operanden zwischen aufeinanderfolgenden Befehlen.
Meherere Register : Speicherung eines Working Sets (z.B. für eine Prozedur).

Unterschiede zum Cache: explizite Verwaltung (ein Cache ist transparent und unterstützt das Modell eines (unendlich) großen Speichers). Warum nicht NUR Cache?

Registerressourcen sind inhärent beschränkt und bedürfen der expliziten Verwaltung durch das Anwenderprogramm.

Problem der Registerallokation und Verwaltung bei sehr vielen Registern:

Probleme beim “Retten“ eines großen Registersatzes bei Unterprogrammsprüngen / Contextwechsel

Asmb-Programmierer kann durch Ausnutzung der Programmsemantik eine sehr effiziente Registernutzung vornehmen, z.B. Bewahrung von Registerinhalten über Prozeduraufrufe hinweg oder sogar über Context-Wechsel.

Compiler: Löschen aller Register bei Unterprogrammsprung.

68020 Register

Datenregister:

- alle Datenoperationen können unterschiedslos auf allen Datenregistern ausgef. werden
- unterstützen Operationen auf allen Datentypen des 68020
- können in einem indizierten Adressierungsmodus den Index enthalten

Adreßregister:

- werden als Basisregister bei der Adressierung verwendet
- werden als 32-Bit Einheiten behandelt und können nur Operationen auf 32-Bit Datentypen ausführen
- arithmetisch logische Operationen auf Adreßreg. modifizieren nicht die Bedingungsflags im CCR
- Adreßregister #7 dient als Stack Pointer in Unterprogrammaufrufen !
  (alle anderen Adreßregister können als SP verwendet werden)
- das Supervisor-Flag (S) und das Master/Interrupt-Flag (M) im Satus Register entscheiden, welcher Stack-Pointer tatsächlich genutzt wird.

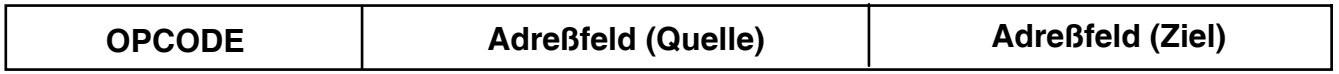
Befehls-formate:

Ein-Adreß-Befehl

Beisp.:
Motorola 6809

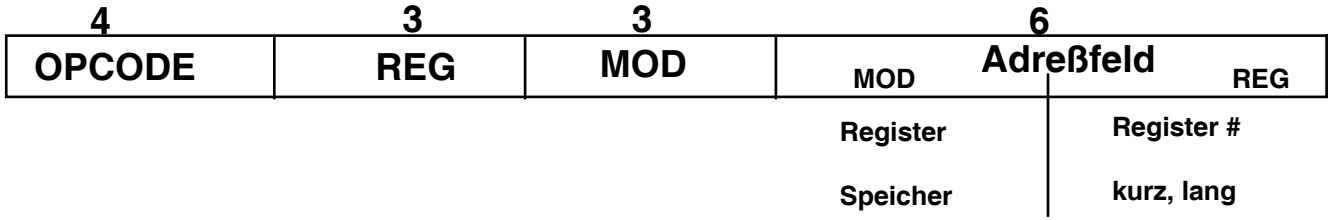
Zwei-Adreß-Befehl

680x0,
Intel 386,486



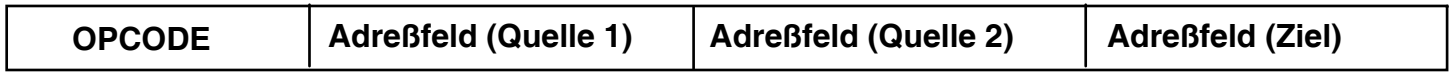
2.ter Quelloperand steht an Zieladresse und wird durch Operation überschrieben

Beisp.:
Motorola
680x0

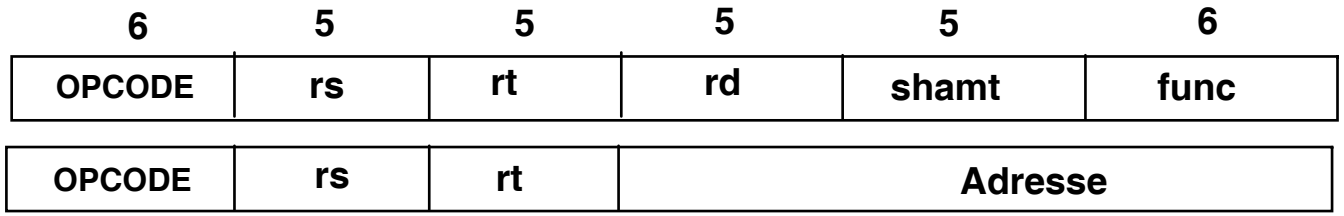


Drei-Adreß-Befehl

MIPS,
Alpha,
SPARC



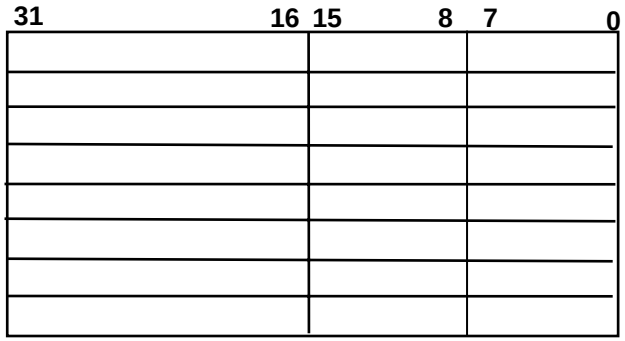
Beisp.
MIPS



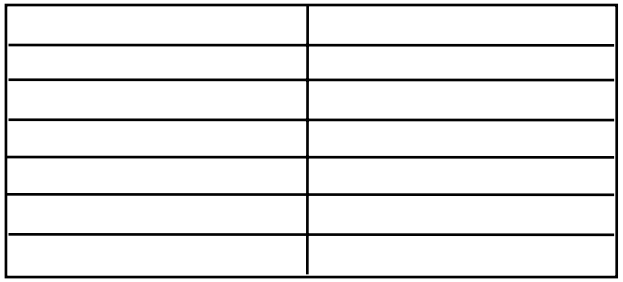
rs: erstes Quellreg.
rt: zweites Quellreg.
rd: Zielreg.
shamt : shift amount
func: funktionale Erw. des OPCODE

68020 Programmiermodell

Erweiterungen zum 68000 Programmiermodell



Datenregister



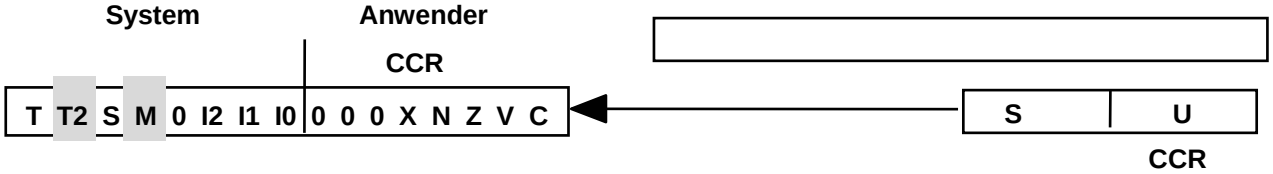
Adreßregister

User Stack Pointer

Supervisor Stack Pointer

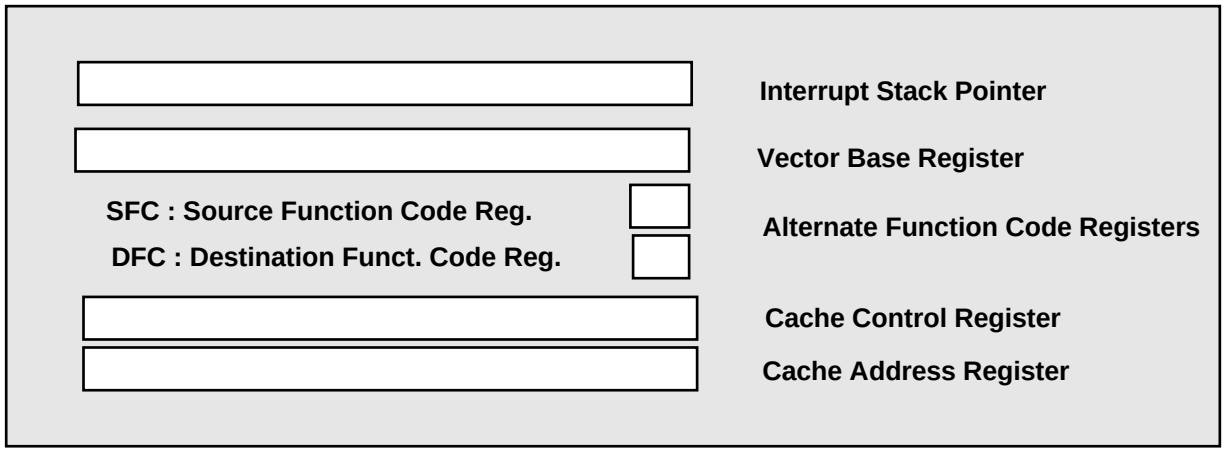
Programmzähler

Status Register

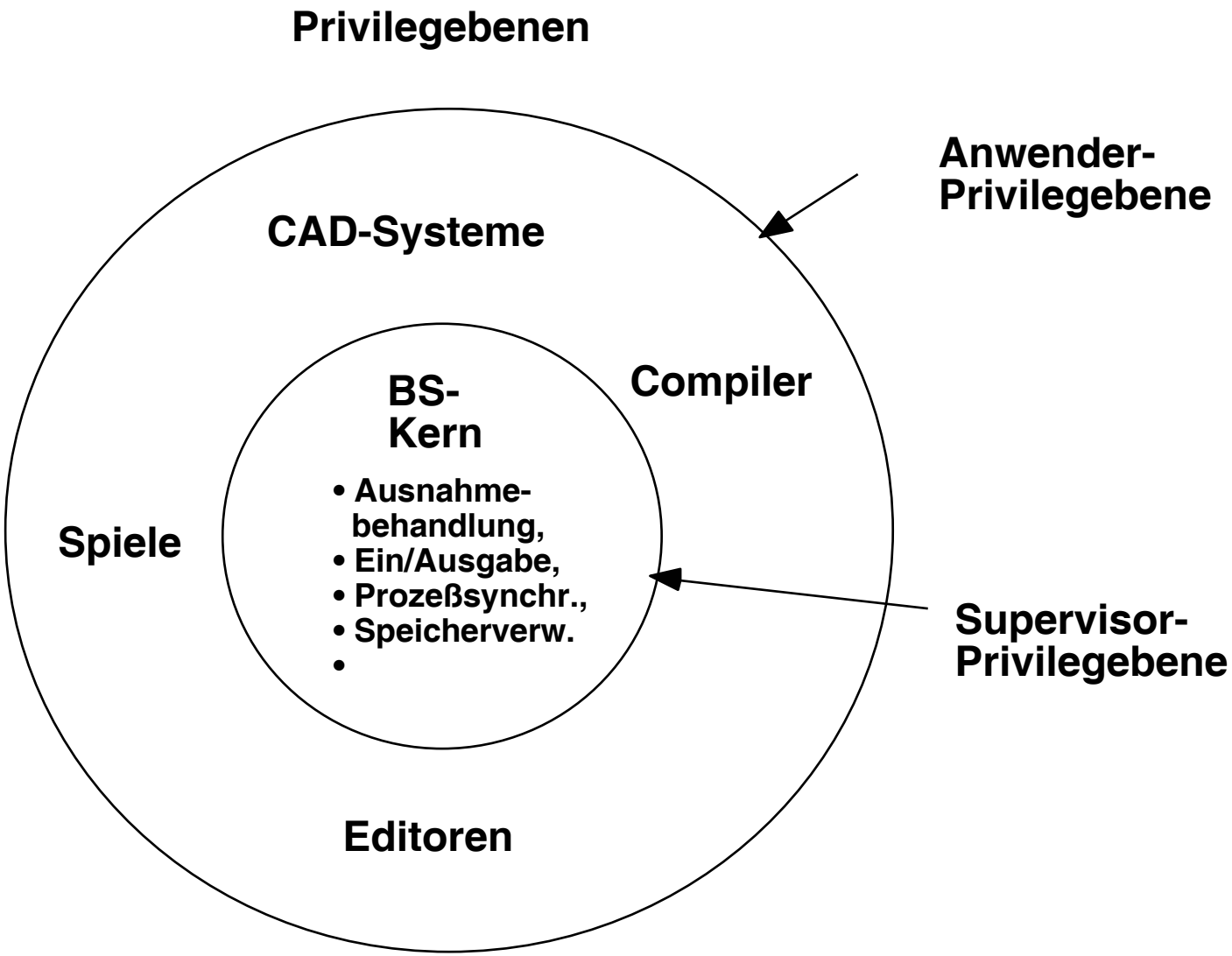


S: Systembyte
U: User Byte

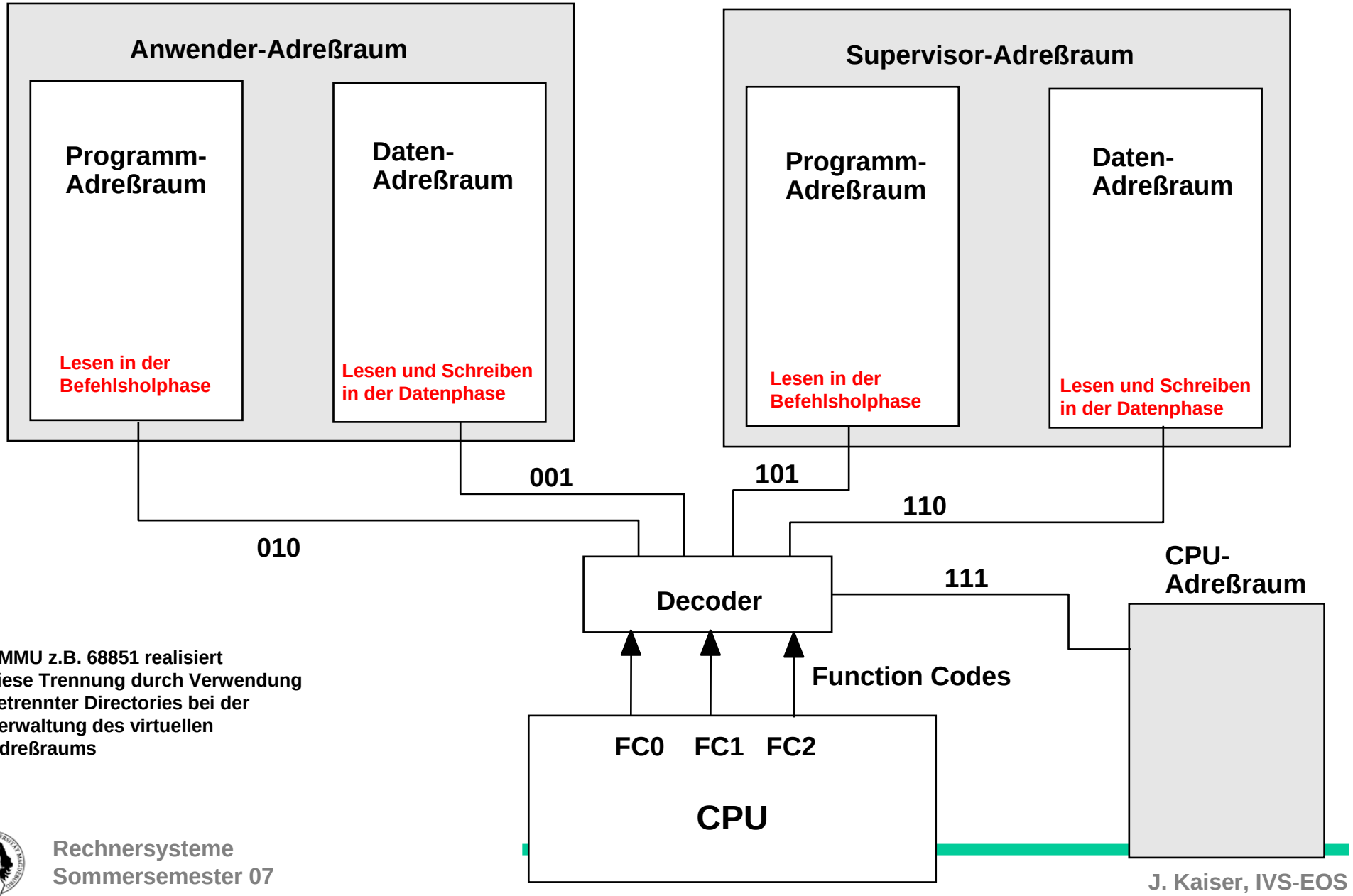
0: nicht belegt
T: Tracemodus
T2: Tracemodus
S: Supervisor-Status
M: Master / Interrupt Status
I2: Interruptmaske Bit 2
I1: Interruptmaske Bit 1
I0 : Interruptmaske Bit 0
X: Erweiterung
N: Negativ
Z: Zero
V: Overflow
C: Carry



68020 Erweiterungen


Privilegebenen
Anwender-
Privilegebene
CAD-Systeme
Compiler
BS-
Kern
• Ausnahme-
behandlung,
• Ein/Ausgabe,
• Prozeßsynchr.,
• Speicherverw.
•
Spiele
Supervisor-
Privilegebene
Editoren

Trennung der Adreßräume durch die Funktionscodes



PMMU z.B. 68851 realisiert diese Trennung durch Verwendung getrennter Directories bei der Verwaltung des virtuellen Adreßraums

Unterscheidung von:

Anwenderadreßraum für Programmcode
Anwenderadreßraum für Daten
Supervisoradreßraum für Programmcode
Supervisoradreßraum für Daten

Function Code Belegung:

FC0	FC1	FC2	Art des R/W-Zyklus
0	0	0	nicht definiert (reserviert)
0	0	1	User Data Space
0	1	0	User Program Space
0	1	1	nicht definiert (reserviert)
1	0	0	nicht definiert (reserviert)
1	0	1	Supervisor Data Space
1	1	0	Supervisor Program Space
1	1	1	CPU Space

Im CPU Space bearbeitet der Prozessor:

1. **Interrupts**
2. **Traps**
3. **Breakpoints**
4. **Kommunikation mit einem Coprozessor**
5. **Moduloperationen**

Prozessor Zustände:

- Normaler Zustand
- Ausnahmezustand (exception processing state)
- Halt Zustand

Privilegebenen:

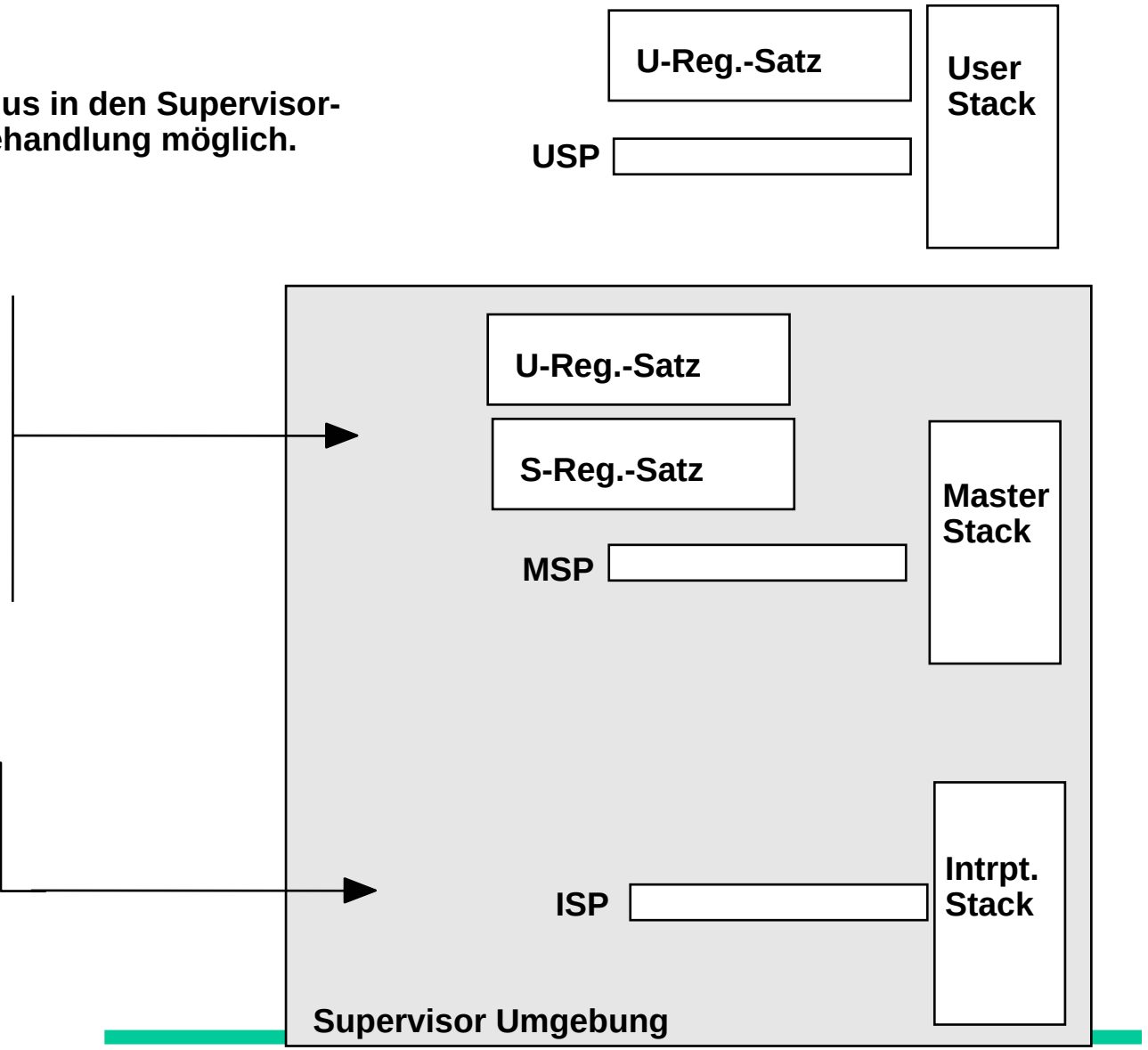
- Supervisor Ebene
  - S-Bit im Status Register ist gesetzt
  - Function Codes signalisieren den Supervisor Adreßraum
  - M-Bit im Satus Register unterscheidet zwischen “Master State“ und “Interrupt State“. Interrupt State entspricht dem 68000/68008/68010 Supervisor State.

- Benutzer Ebene
  mit externer Hardwareunterstützung kann der 68020 bis zu 256 Privilegebenen innerhalb der Benutzerebene realsieren.)

Übergang vom Anwender in den Supervisor Zustand

Der Übergang vom Anwendermodus in den Supervisor-modus ist nur durch Ausnahmebehandlung möglich. Ausnahmen sind:

- **explizite Software Traps:**
  z.B. TRAP, CHK,
- **Ausnahme, die bei der Befehlsabarbeitung auftreten:**
  z.B. Illegale Instruktion,
  Division durch 0
  Adressierungsfehler
  Privilegverletztung
- **Interrupts**



Bussystem

Designentscheidungen:

synchron - asynchron

Daten und Adressen auf einem Bus (multiplexed) - getrennte Busse (non-multiplexed)

Burst-Modus - ein Buszyklus/ein Datum

Arbitrierter Bus (mehrere Bus-Master) - einzelner fester Bus-Master

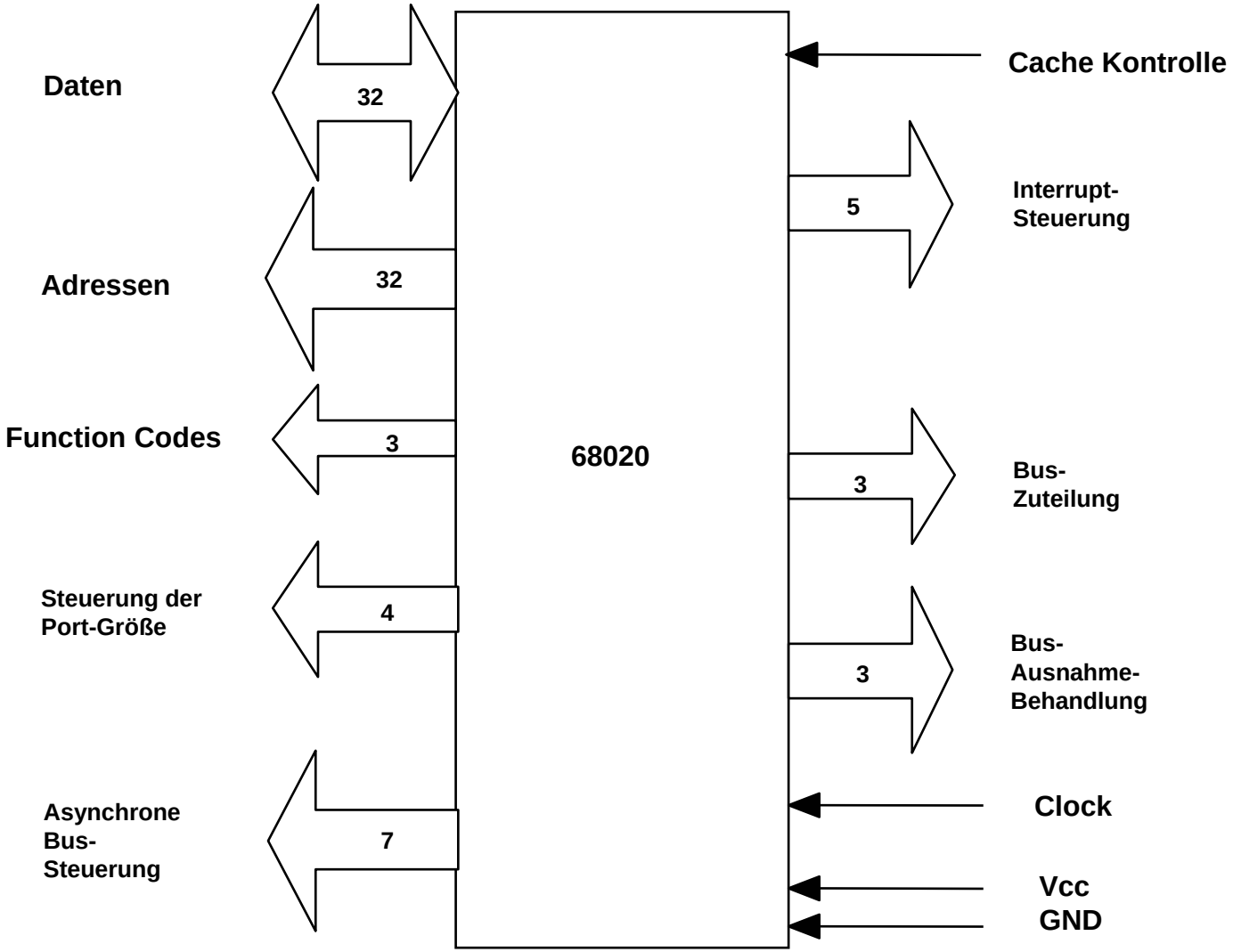
dynamische (automatische) Busanpassung - explizite (programmierte) Busanpassung

automatische Fehlerbehandlung - programmierte Fehlerbehandlung

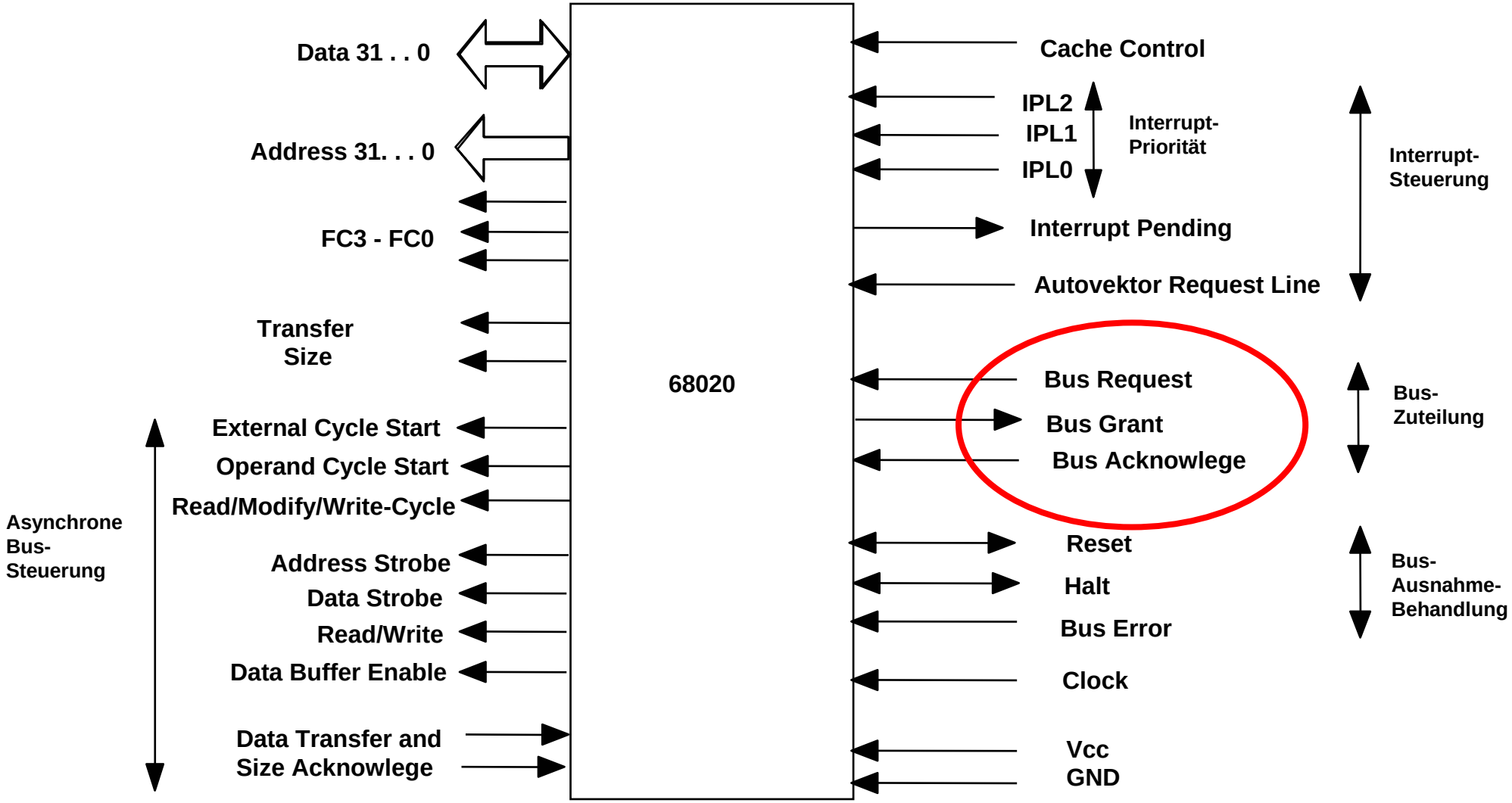
68020 Bussystem

- Parallele 32-Bit-Busse für Daten und Adressen
- getrennte Busse, kein Busmultiplexing
- Asynchrones Busprotokoll
- Master/Slave Konfiguration
- Bus Arbitration, mehrere Bus-Master
- Dynamische Busanpassung
- Unterstützung von nicht auf Wortgrenzen ausgerichteten Daten
- Flexible Behandlung von Busfehlern
- Automatische Wiederholung von Buszyklen (Retry)

68020 Schnittstellensignale



68020 Schnittstellensignale (Detail)



68020 Busarbitrierungs-Zyklus

Prozessor | **Anforderndes Gerät**

Anfordern des Busses:

1. Aktivieren von“ Bus Request“

Bestätigung des Busarbitrierungszyklus

1. Aktivieren von“ Bus Grant“

Bestätigung der Übernahme als Bus Master:

1. Externe Arbitrierungslogik bestimmt den nächsten Bus Master
2. Designierter Bus Master wartet bis der laufende Buszyklus beendet ist
3. Designierter Bus Master aktiviert das Signal “Bus Grant Acknowlege“, um neuer Bus Master zu werden
4. Neuer Bus Master deaktiviert “Bus Request“

Beendigung des Arbitrierungszyklus:

1. Deaktiviere “Bus Grant“ und warte auf die Deaktivierung von “Bus Grant Acknowlege“

Operation als Bus Master:

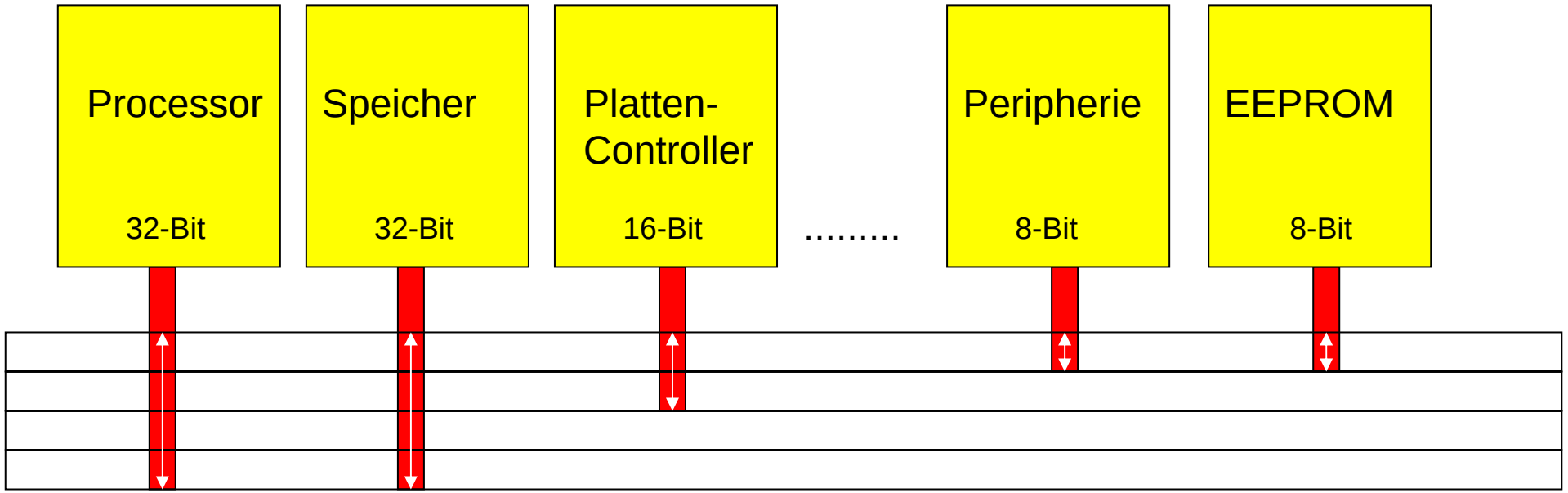
1. Durchführung der Datentransfers (Lesen und/oder Schreiben) ohne jegliche Kontrolle durch den Prozessor.

Freigabe des Bus Masters:

1. Deaktivieren von “Bus Grant Acknowlege“

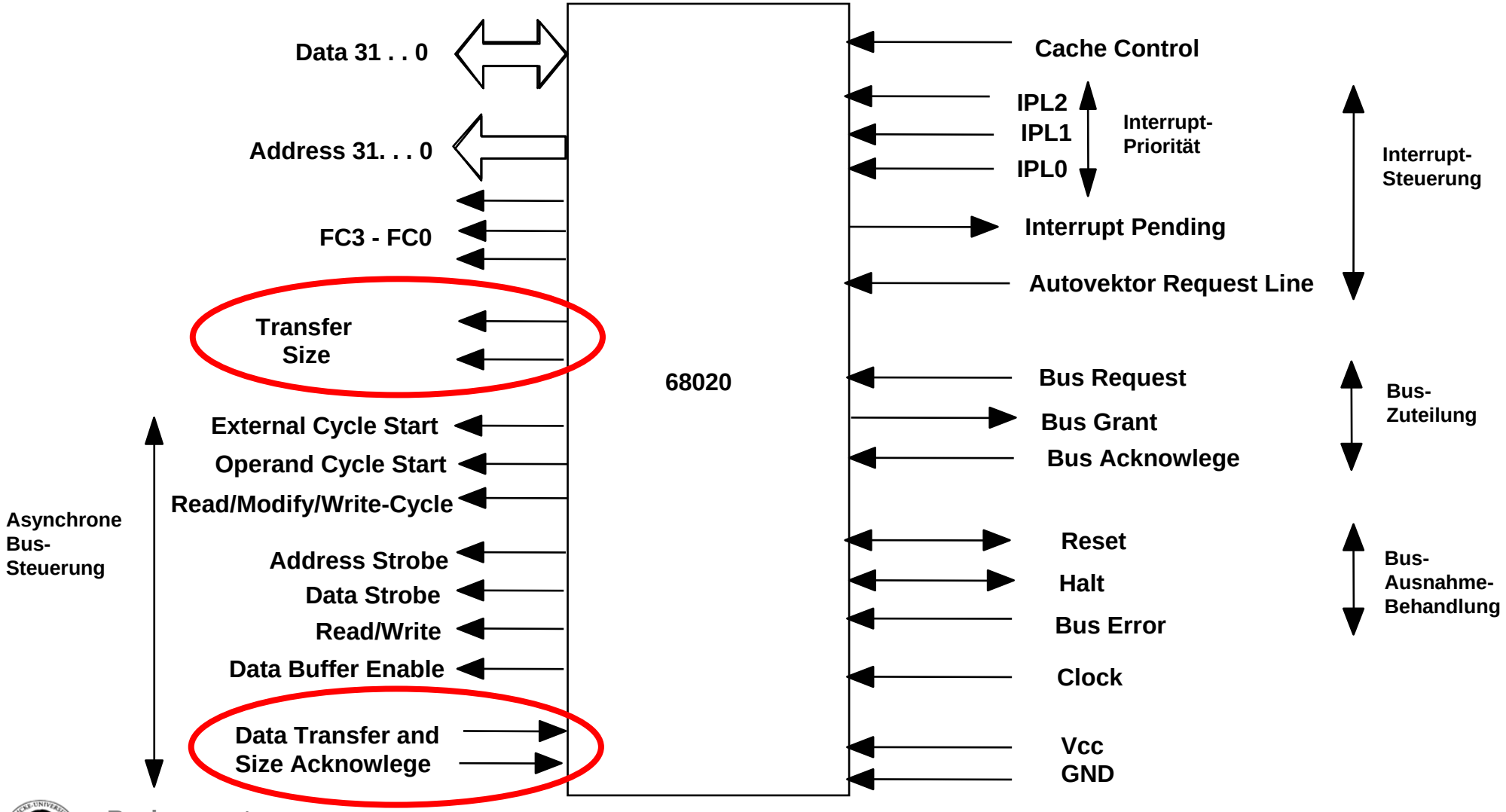
Neues Arbitrierungszyklus oder Fortführen der Prozessoroperationen

Dynamische (automatische) Busanpassung

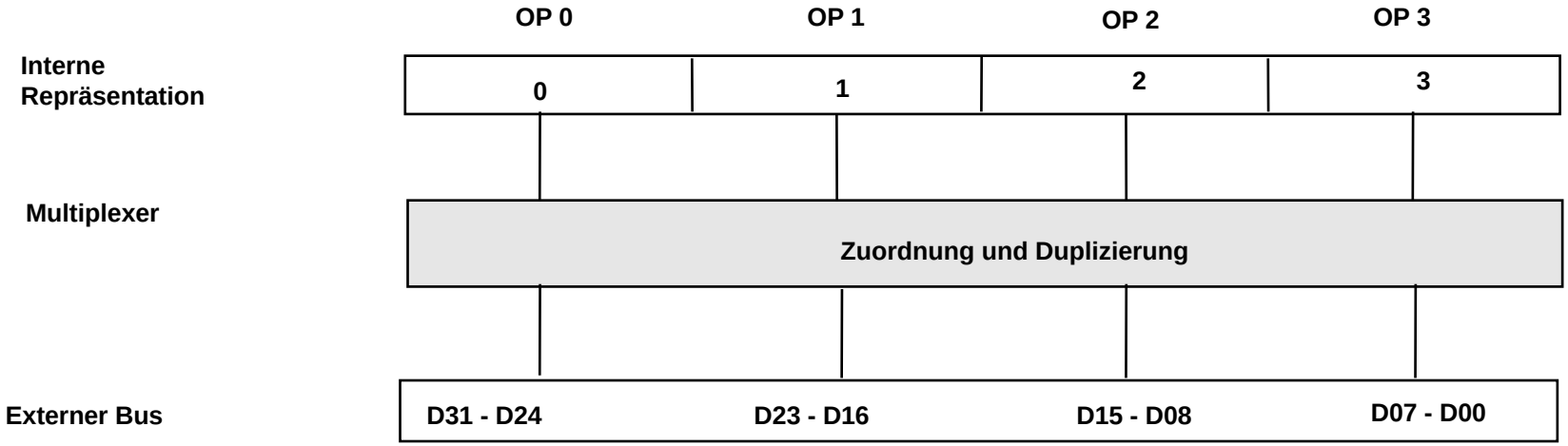


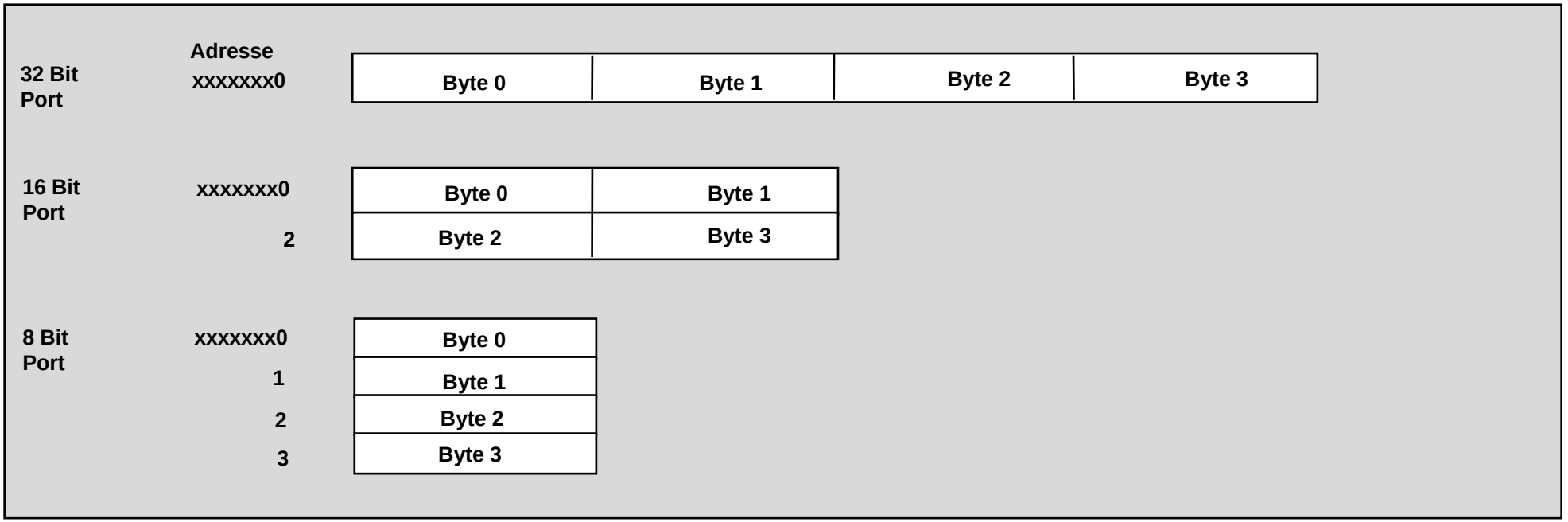
Datenbus 32-Bit

68020 Steuersignale (Detail)



Dynamische Busanpassung





Dynamische Busanpassung

68020 signalisiert über die Ausgänge SIZ0 und SIZ1 die Port-Größe für den durchzu-führenden Datentransfer

Codierung der “Size“-Ausgänge

SIZ1	SIZ0	übertragene Größe
0	0	Langwort (32 Bit)
0	1	Byte
1	0	Wort (16 Bit)
1	1	3 - Byte

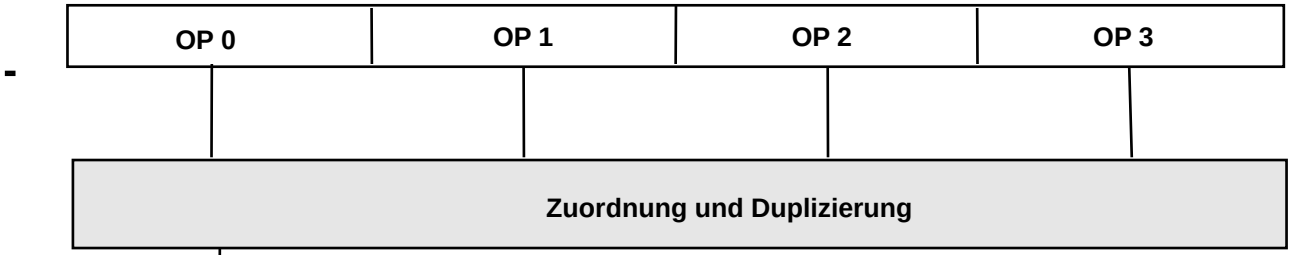
Peripheres Gerät signalisiert über die Eingänge DSACK0 un DSACK1 die erforderliche Port-Größe

Codierung der DSACK- Eingänge

DSACK1	DSACK0	Resultat
1	1	Einfügen von Wartezyklen
1	0	vollst. Zyklus - DBPS : 8 Bit
0	1	vollst. Zyklus - DBPS : 16 Bit
0	0	vollst. Zyklus - DBPS : 32 Bit

Dynamische Busanpassung

Beispiel - Transfer eines Langwortes über einen 8-Bit-Bus:



Externer Bus — D31 - D24

Adresse :	A1 A0	
	xx....0 0	OP 0
	xx....0 1	OP 1
	xx....1 0	OP 2
	xx....1 1	OP 3

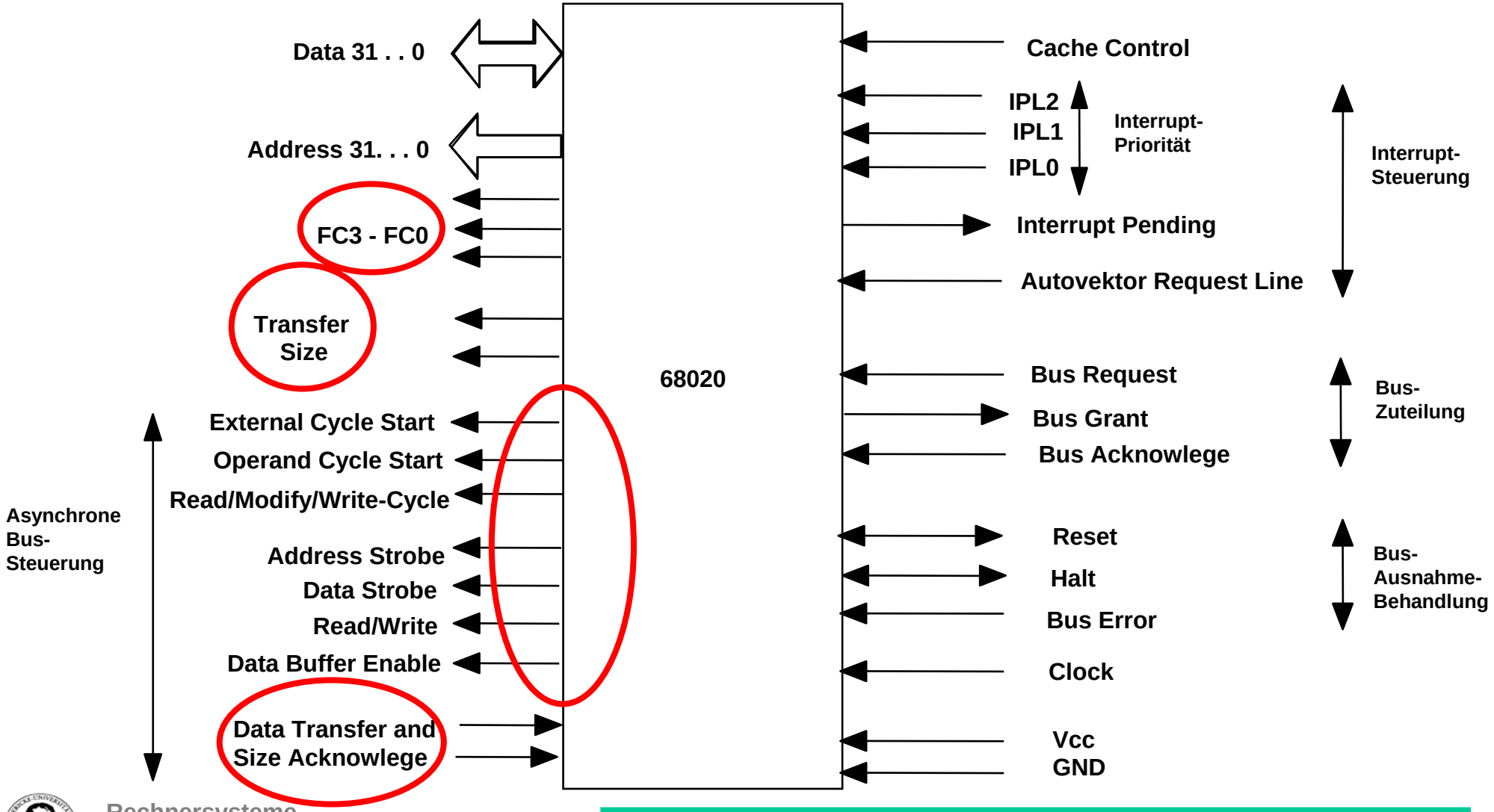
Byte-orientierter Speicher oder Peripherie

SIZ1	SIZ0	
0	0	(32)
0	1	(8)
1	0	(16)
1	1	(24)

DSACK1	DSACK0	
1	1	Wartezyk.
0	1	(8)
1	0	(16)
0	0	(32)

68020				Periph. Kontrolle		
SIZ1	**SIZ0**	**A1**	**A0**	**DSACK1**	**DSACK0**	
0	0	0	0	0	1	**1.Zyklus : OP0 wird übertragen**
1	1	0	1	0	1	**2.Zyklus : OP1 wird übertragen**
1	0	1	0	0	1	**3.Zyklus : OP2 wird übertragen**
0	1	1	1	0	1	**4.Zyklus : OP3 wird übertragen**

68020 Steuersignale für Lesezyklus



68020 Lese-Zyklus

Bus Master

Slave

Adressierungsphase:

1. R/W ← Read
2. Funktionscode anlegen
3. Adresse auf A31-A0 legen
4. SIZ1, SIZ0 anlegen
5. Anzeigen, daß ein Buszyklus gestartet wird .
6. Adreßstrobe aktivieren. Diese Signal bestätigt, daß ein Buszyklus durchgeführt wird.
7. Datenstrobe aktivieren
8. Data Buffer Enable aktivieren

Datenübergabe:

1. Adresse dekodieren
2. Daten auf Datenbus legen
3. DSACK0 und DSACK1 aktivieren

Datenübernahme:

1. Daten in Datenpuffer speichern
2. Deaktivieren des Datenstrobes
3. Deaktivieren des Adreßstrobes
4. Deaktivieren des Signals “Data Buffer Enable“

Buszyklus beenden:

1. Daten vom Bus nehmen
2. DSACK deaktivieren

Nächsten Buszyklus starten

68020 Befehlssatz:

Daten Transfer (Data Movement)

Arithmetische Operationen (Integer) Division und Multiplikation (32/32 und 64/32)

BCD Operationen (ABCD, SBCB: „echte“ BCD-Addition und Subtraktion)

Logische Operationen

Shift und Rotate Operationen (Angabe von Shift Count)

Befehle zur Programmkontrolle

Einzelbit Befehle

Bitfeld Befehle

Komplexe Vergleichsbefehle (Überprüfung von oberen und unteren Schranken)

Befehle zur Systemkontrolle

Befehle zur Multiprozessorkommunikation

Co-prozessor Befehle

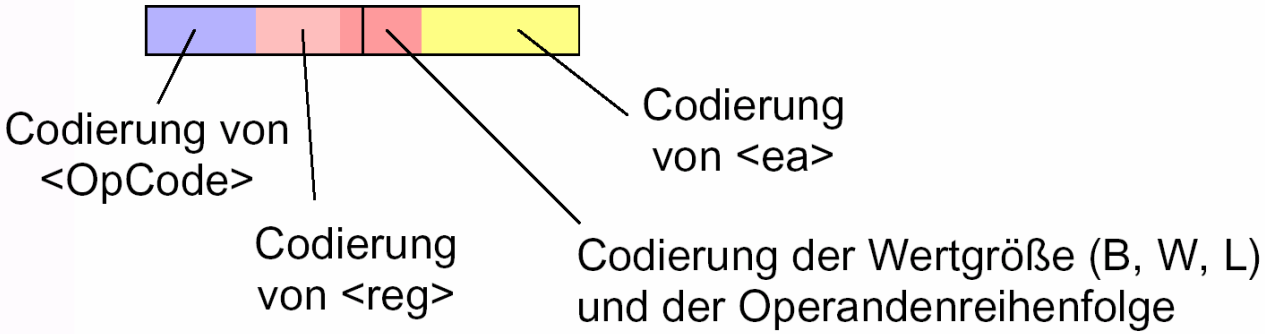
Datentypen:

- Bit
- Bit Feld
- Byte
- BCD (packed (2digits/byte), unpacked (1digit/byte))
- Word
- Long
- Quad

68020 Befehlformat:

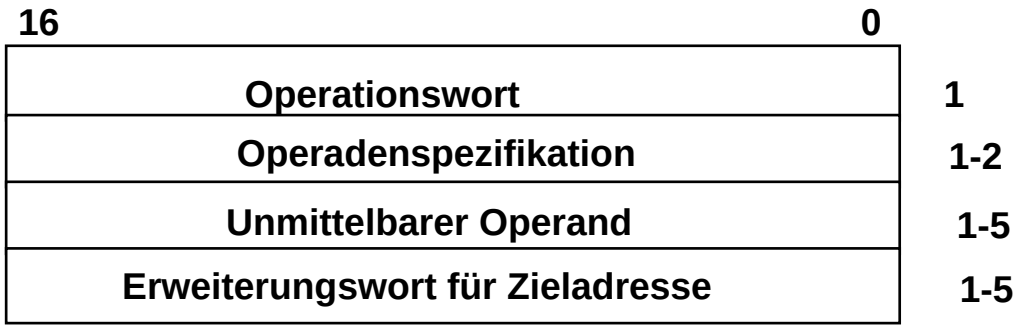
Genereller Aufbau eines 68020 Befehls:

- Codierung des Regelfalls
  - 4 Bit für Codierung des OpCodes
  - 6 Bit für Codierung der <ea>
  - 3 Bit für Codierung des Registers
  - 3 Bit für Codierung der Wertgröße und der Reihenfolge der Operanden
    - <ea>, <reg> oder <reg>, <ea>



68020 Befehlformat:

Genereller Aufbau eines 68020 Befehls:

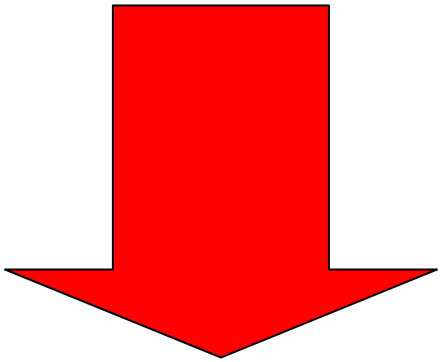


min. Befehlslänge: 2 Bytes

max. Befehlslänge: 22 Bytes

immer noch nicht genug Befehle?

Speicherverwaltung?
Floating Point ?
Grafik ?



Coprozessoren

Coprozessoren: Funktionale Partitionierung in Hardware-Spezialeinheiten

Ziel: Realisierung von Spezialfunktionen wie: Fließkomma-Arithmetik, Vektoroperationen, Speicherverwaltung, Graphikfunktionen, HLL-Interpretation, etc.

Alternativen?: Software-Lösung, Vertikale Verlagerung in die Mikroprogrammebene

Vorteile von Coprozessoren gegenüber vertikaler Verlagerung:

- **Durch funktionale Partitionierung wird die Komplexität des Entwurfs vermindert. Die Komplexität eines umfangreichen Mikroprogramms stellt schon bei konventionellen Prozessoren ein größeres Problem dar.**
- **Durch einen problemangepaßten Coprozessor können spezielle Aufgaben effizienter gelöst werden als durch einen mikroprogrammierbaren Universalprozessor.**
- **Coprozessor und CPU können (im Prinzip) nebenläufig arbeiten.**
- **Höhere Flexibilität, da durch der Partitionierung eine Isolation der Spezial-Funktionen vom Instruktionssatz der CPU erreicht wird. Dadurch können die Spezialfunktionen leichter und ohne Nebenwirkungen auf die CPU geändert werden.**
- **Der Instruktionssatz einer Standard-CPU wird durch einen Coprozessor so erweitert, daß eine volle Kompatibilität mit Standardsoftware erhalten bleibt. Spezielle Coprozessorbefehle können meist leicht emuliert werden.**

Kriterien zur Klassifizierung von Coprozessoren:

Ebene der Coprozessor-Funktionen:

- Instruktions-Coprozessoren (Beisp. FPU, Grafik)
- Funktions-Coprozessoren (Beisp. Grafik,Kommunikation, Scheduling)
- Programm-Coprozessoren (eigene Programmsteuerung, z.B. spez. Sprachproz.)

Steuerung des Coprozessors (Enge Kopplung vs. Lose Kopplung)

- Ein Instruktionsstrom für CPU und Coprozessor (Transparente Erweiterung des Instruktionssatzes der CPU)
- Getrennte Instruktionsströme für CPU und Coprozessor (Autonome Coprozessoren, z.B. Gafik)

Anbindung an die CPU

- Eigene Befehlsdekodierung (Instruction Tracker)
- Befehlsdekodierung wird von der CPU durchgeführt

- synchrones Protokoll mit der CPU
- asynchrones Protokoll mit der CPU

- CPU führt alle Speicherzugriffe durch
- Coprozessor kann selbst Speicherzugriffe durchführen.

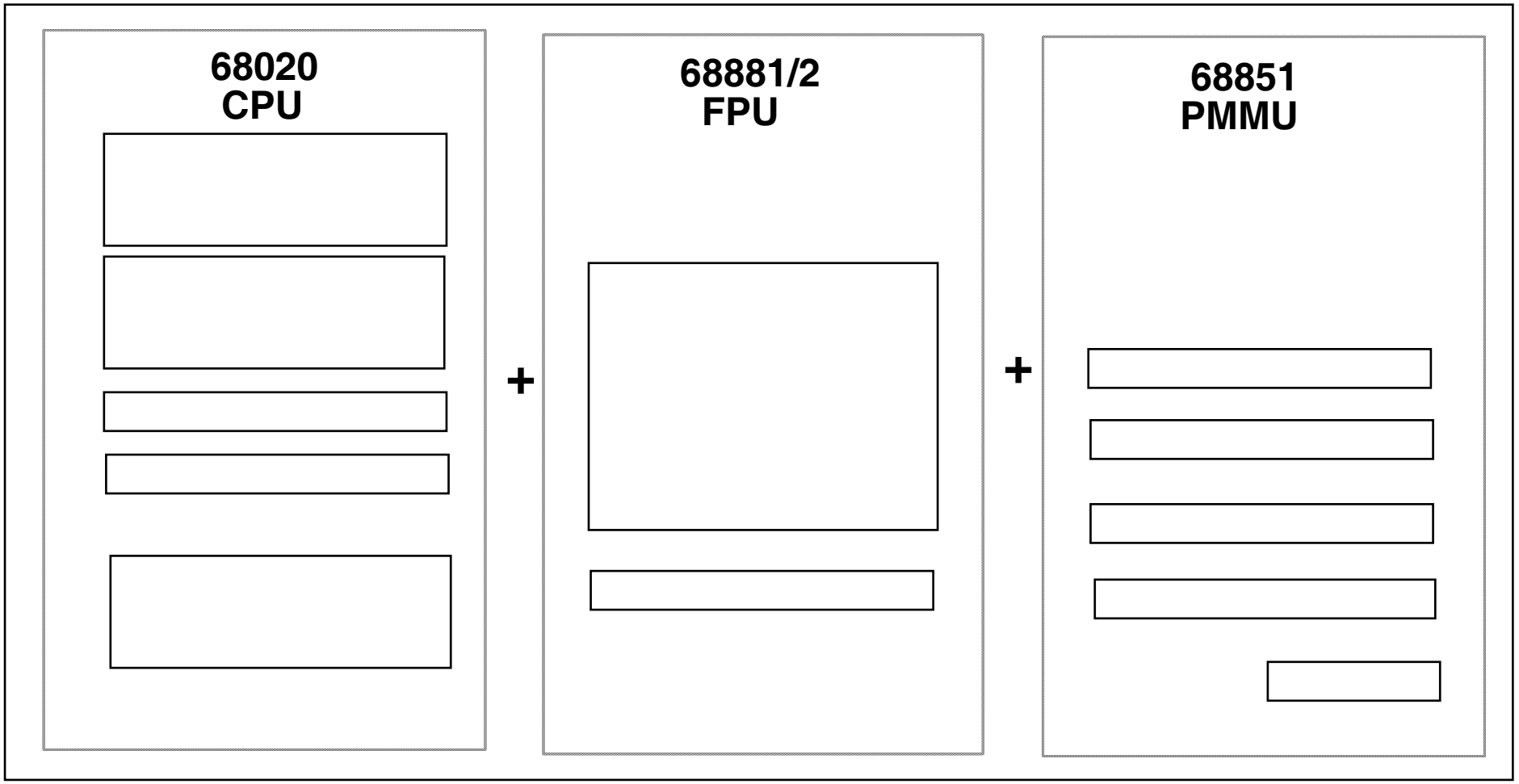
Parallelität : CPU und Coprozessoraktivität können nebenläufig arbeiten

Beispiel: 68881/68882 Floating Point Coprozessor

Befehlssatz:

- 46 Befehle
  - kombinierbar mit drei Datentypen durch Suffixes `.S`, `.D` und `.X`
- Registertransferbefehl
  - z.B. `FMOVE.D 4(A5,D1.W), FP0`
- Grundrechenarten
  - z.B. `FADD.S $123456, FP2`
- trigonometrische Funktionen u.ä.
  - z.B. `FSIN.D (A7)+, FP3`
- bedingte Sprünge
  - z.B. `FBEQ L3`

Programmiermodell: Transparente Funktionserweiterung



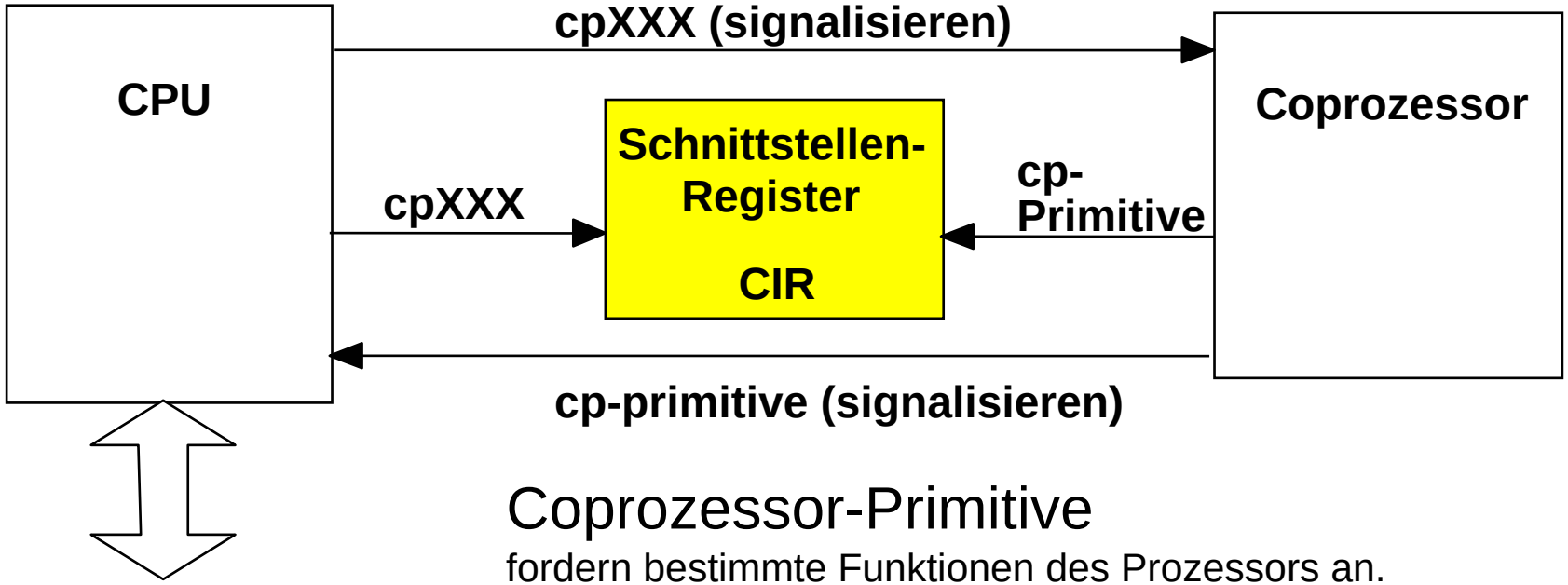
CPU-Instruktionssatz + FPU-Instruktionssatz + PMMU-Instr.satz
. . . . . fadd, jmul, fdiv, . . . pflush, pload, . . .

Instruktionsstrom

.
.
cpXXX
.
.

Coprozessor-Instruktionen

im Instruktionsstrom des Prozessors



Coprozessor-Primitive

fordern bestimmte Funktionen des Prozessors an.

Speicher, Peripherie

Kooperation mit dem Prozessor

Kommunikation über den Systembus
- Keine zusätzlichen Steuerleitungen
- Nutzung der Funktionscodes (CPU Adreßraum)

3-Bit Coprozessor-ID
- maximal 8 Coprozessoren

Schnittstellenregister müssen in jedem Coprozessor vorhanden sein. Sie können von der CPU im CPU-Adreßraum beschrieben und gelesen werden.

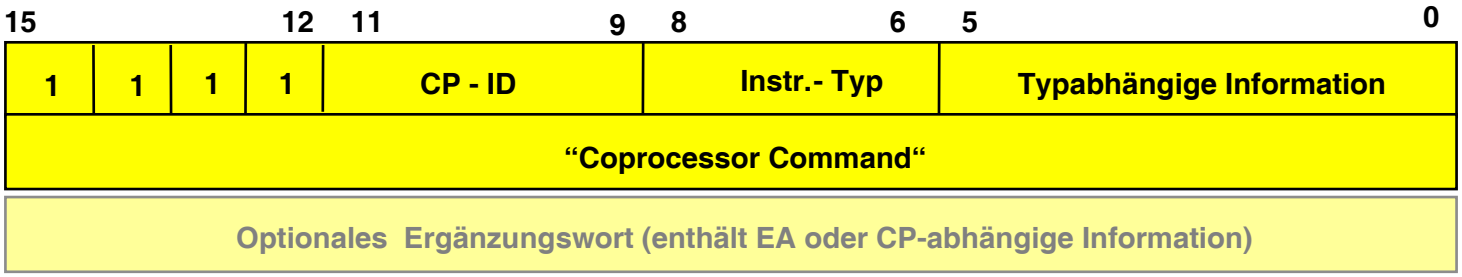
Unterstützung durch 68K Architektur (ab 68020)

Berechnung der effektiven Adresse

Durchführen der Speicherzugriffe

Emulation von Coprozessoren

Coprozessor Instruktion



CP-Operations-Wort

CP-Kommando-Wort

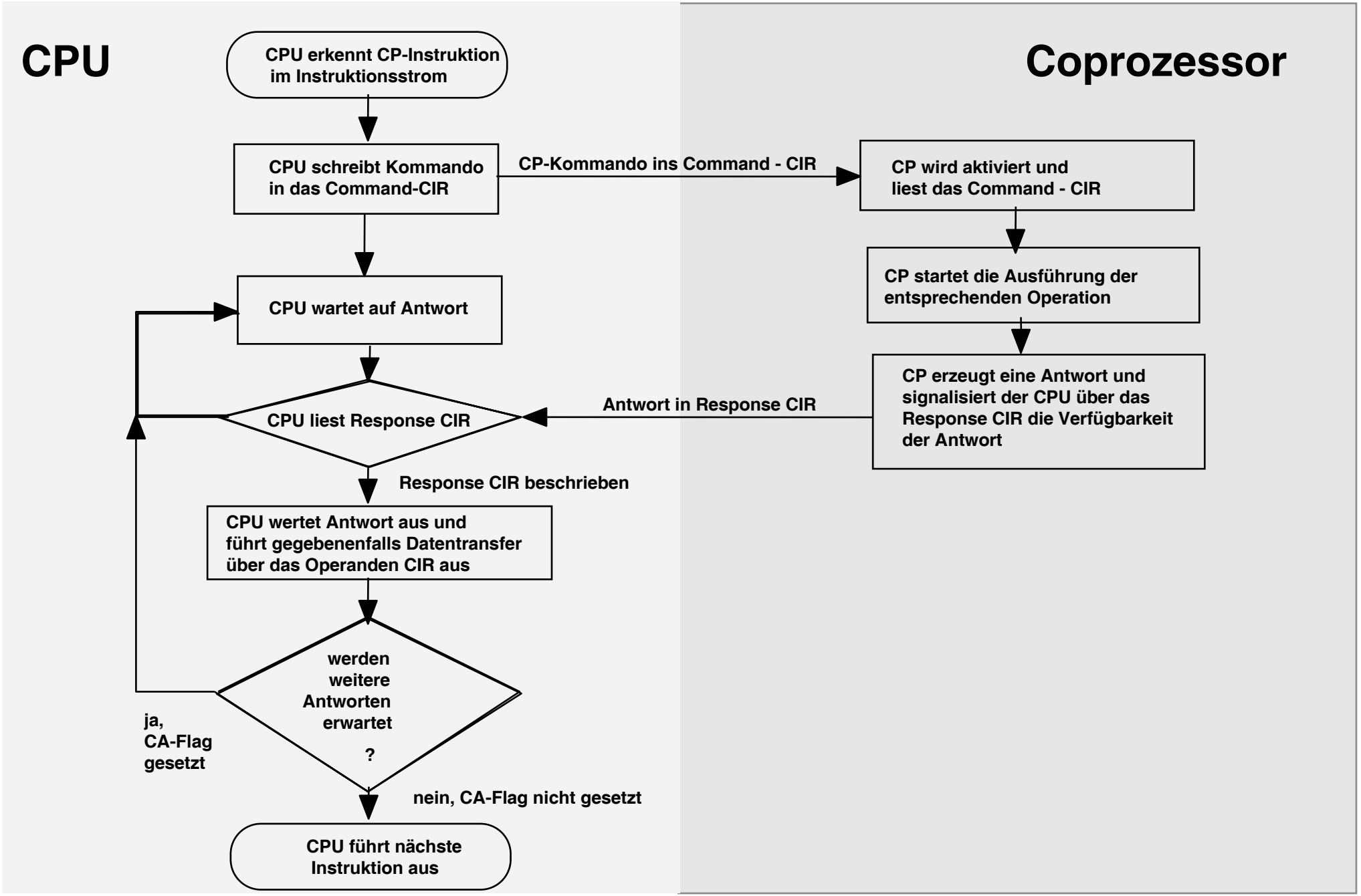
Bit 15 - 12 : F-Line Code. Dieser Code repräsentiert eine Coprozessor-Instruktion

Bit 11 - 9 : Croprozessor Indentifizierung. 8 Coprozessoren können selektiert werden.

Bit 8 - 6 : Typ der Coprozessor Instruktion

Bit 5 - 0 : Information abhängig vom Instruktionstyp

Allgemeines Protokoll zur Ausführung einer Coprozessor-Instruktion



Coprozessoren

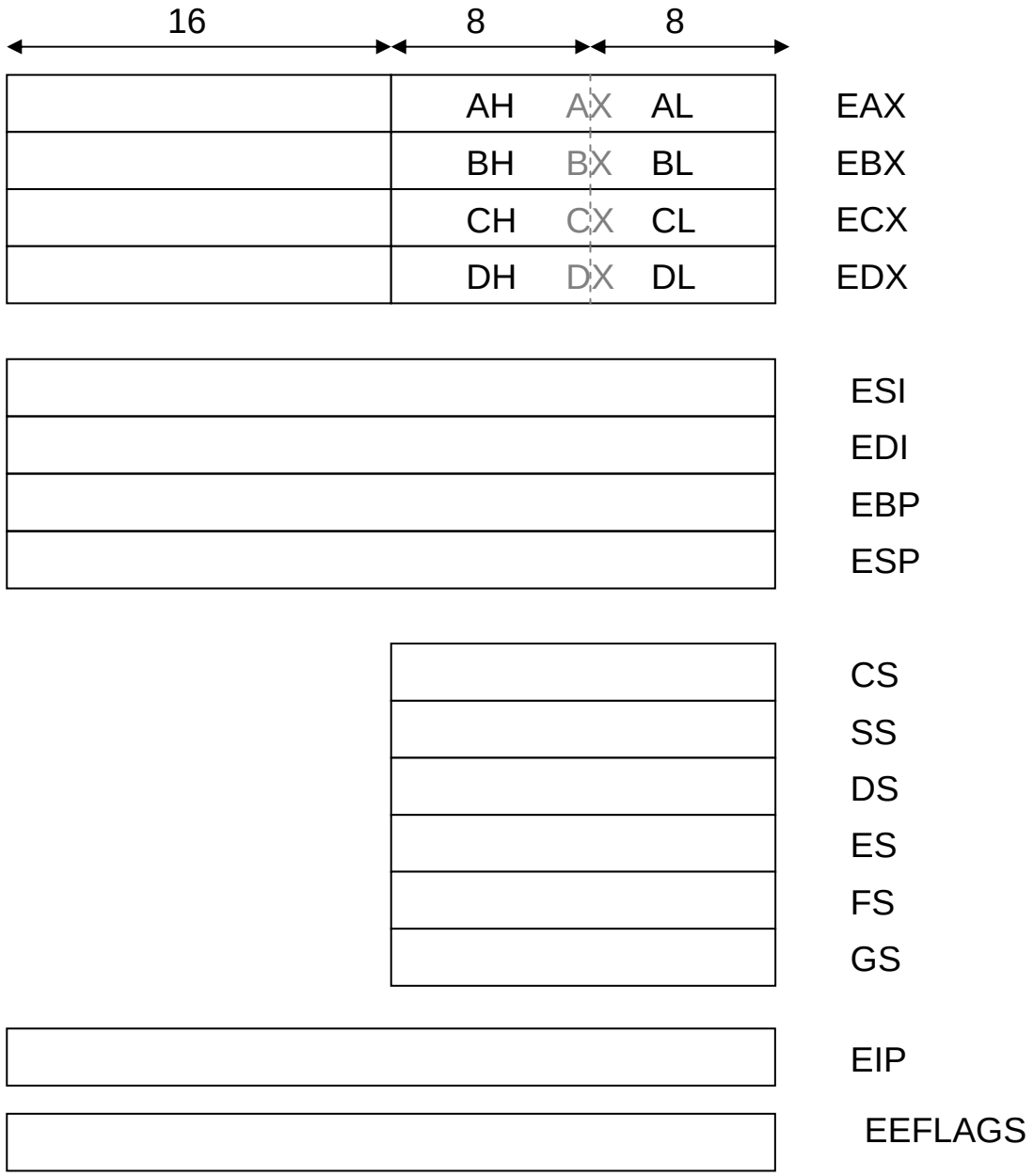
Zusammenfassung:

- Transparente Erweiterung des Maschinenbefehls-Satzes
- Unterschiedliche Grade der Kopplung und des Parallelismus
- Spezialeinheiten generell leistungsfähiger als Mikroprogrammierung

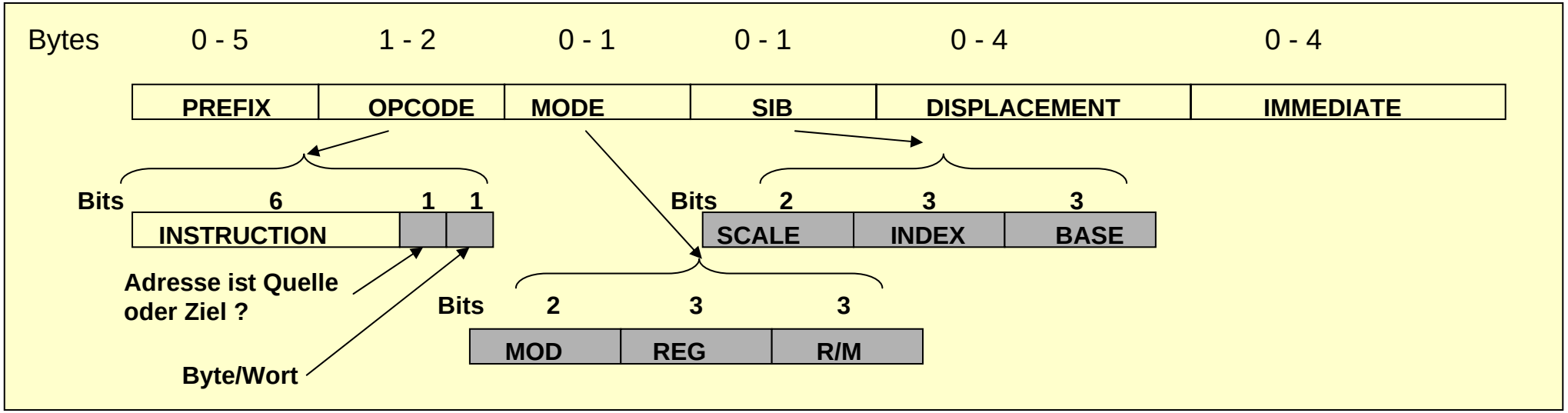
Beispiele von Rechnerarchitekturen

- Pentium
- Sparc
- IJVM

(Haupt) Registersatz des Pentium 4



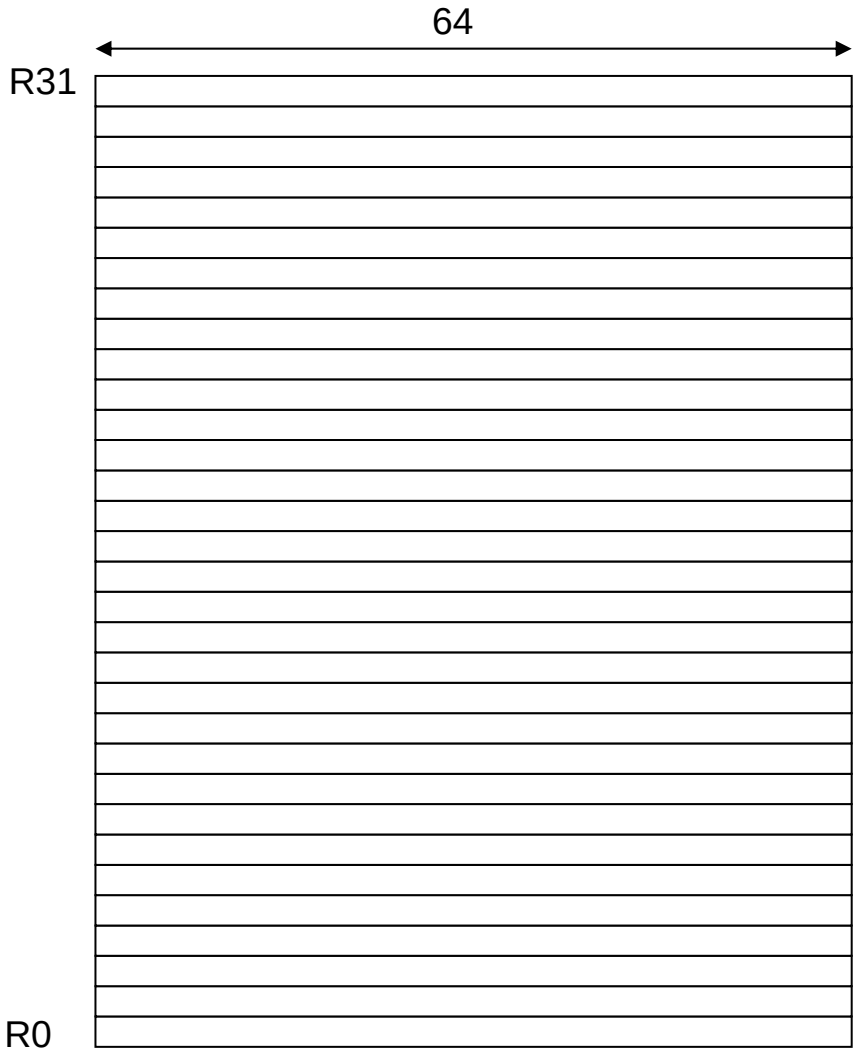
Adressierungsformen



Pentium 4: 32-Bit-ADDR.-MODI

R/M \ MOD	00	01	10	11
000	M[EAX]	M[EAX + o8]	M[EAX + o32]	EAX oder AL
001	M[ECX]	M[ECX + o8]	M[ECX + o32]	ECX oder CL
010	M[EDX]	M[EDX + o8]	M[EDX + o32]	EDX oder DL
011	M[EBX]	M[EBX + o8]	M[EBX + o32]	EBX oder BL
100	SIB	SIB mit o8	SIB mit o32	ESP oder AH
101	direkt	M[EBP + o8]	M[EBP + o32]	EBP oder CH
110	M[ESI]	M[ESI + o8]	M[ESI + o32]	ESI oder DH
111	M[EDI]	M[EDI + o8]	M[EDI + o32]	EDI oder BH

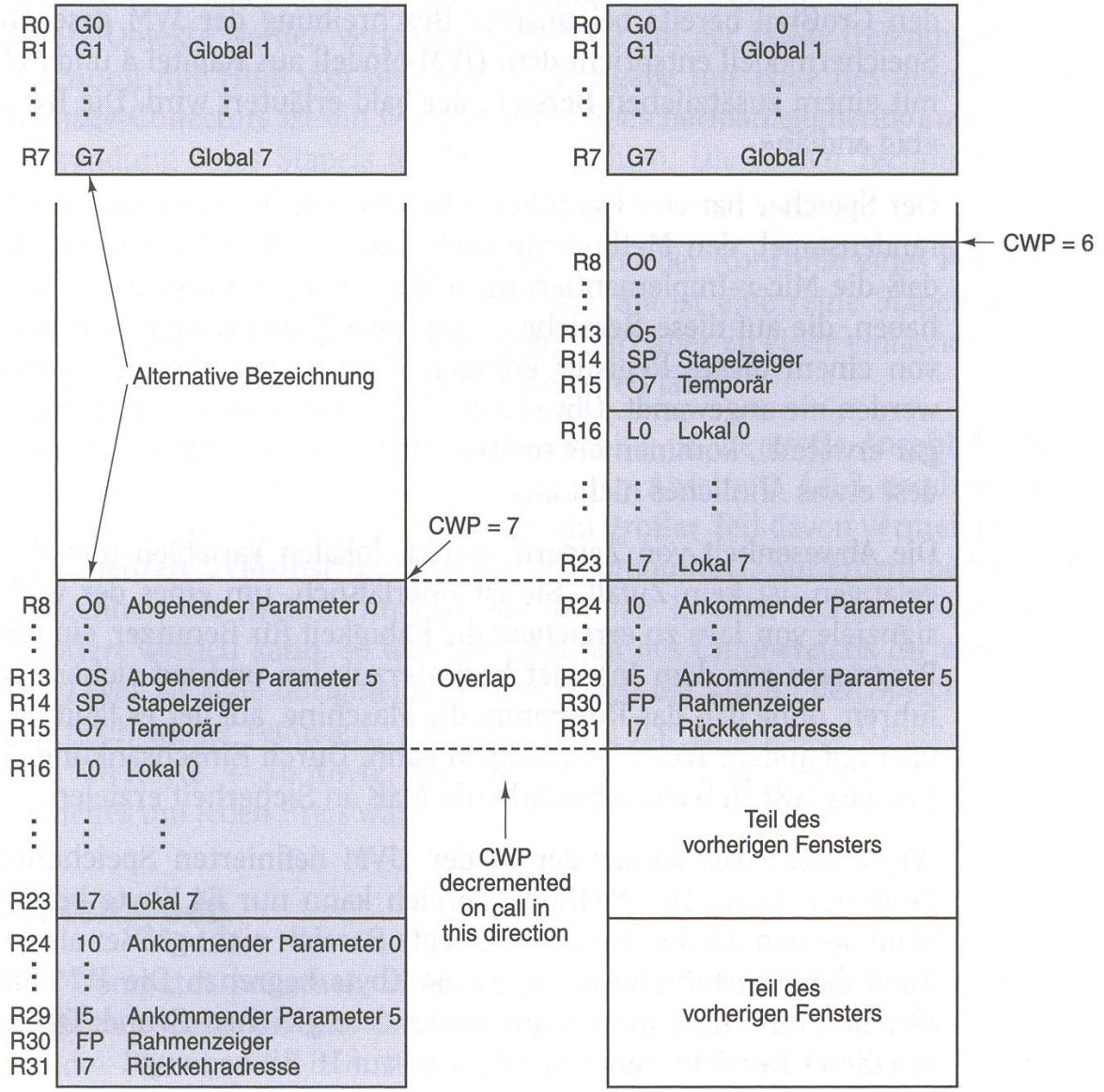
Register der Ultra-Sparc III



Compilernutzung:

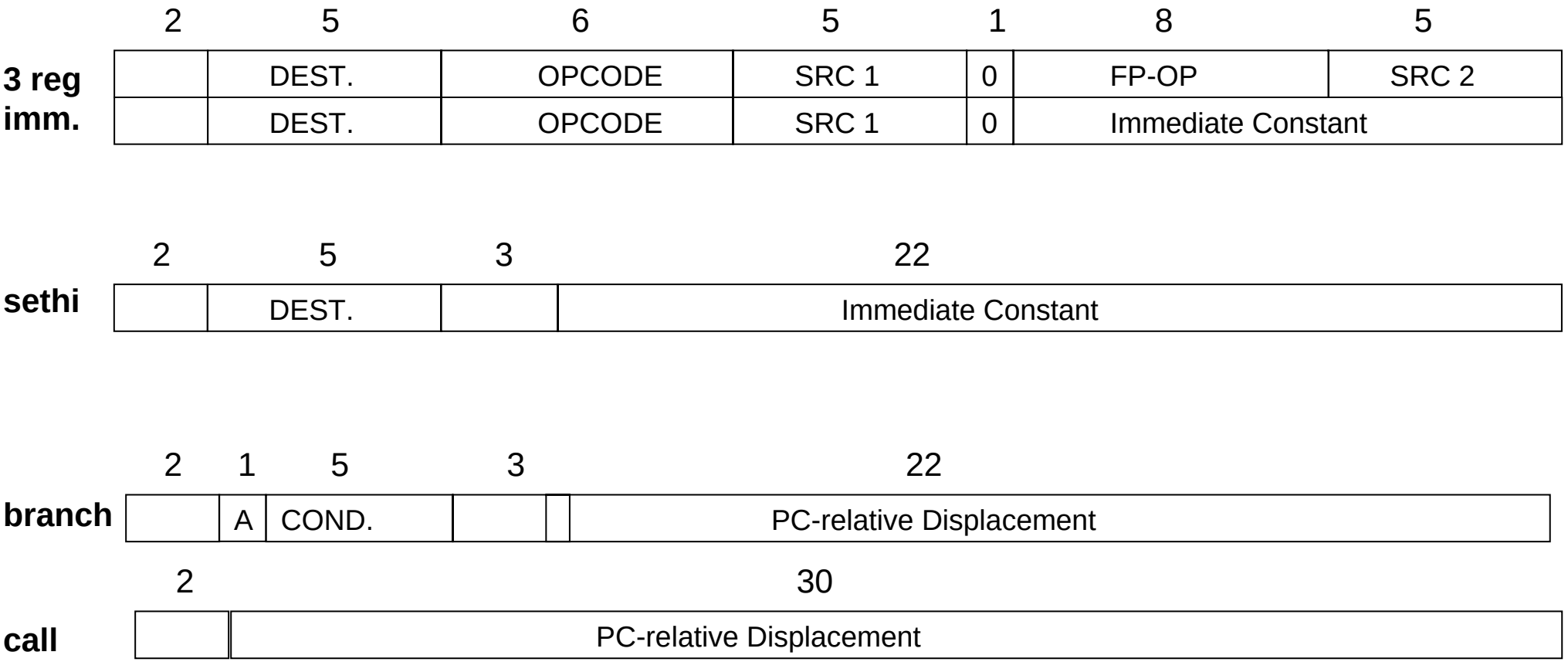
Register	Alt. Bez.	Funktion
R0	G0	Konstante "0"
R1-R7	G1-G7	Globale Variablen
R8-13	00-05	Params. f. aufruf. Prozedur
R14	SP	Stackpointer
R15	07	Temp. Register
R16-23	L0-L7	Lokale Var. f. aktuelle Proz.
R24-29	I0-I5	Eingabe-Parameter
R30	FP	Framepointer
R31	I7	Return Adresse

SPARC
Register Windows



A. Tanenbaum, Computerarchitektur, Pearson Studium, 2001

SPARC Befehlsformate und Adressierungsarten



Addressierungsformen:
Arithm. /logische Befehle: unmittelbar, Register,
Speicherbefehle (Load/Store): Register indirekt, indiziert

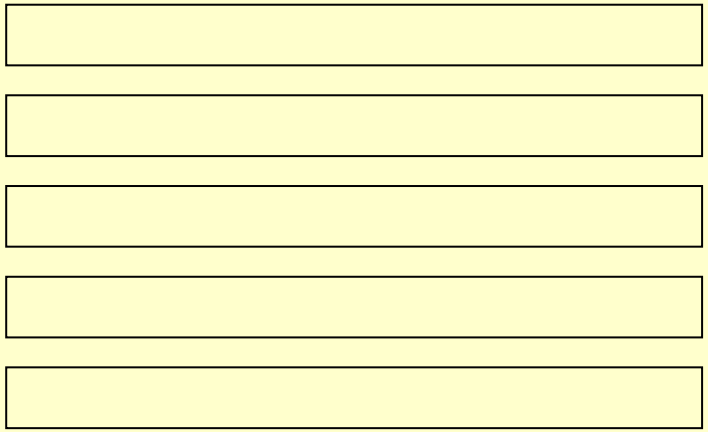
Die Java Virtual Machine (JVM)

- **Stack-orientierter Assemblerbefehlssatz**
- **Relative Adressierung**
- **Komplexer "Invoke"-Befehl**

IJVM: eingeschränkte Architektur,
A. Tanenbaum: Computerarchitektur, Pearson Studium, 2001

Die I-"Java Virtual Machine"

Programmiermodell der IJVM



SP (Stack Pointer)

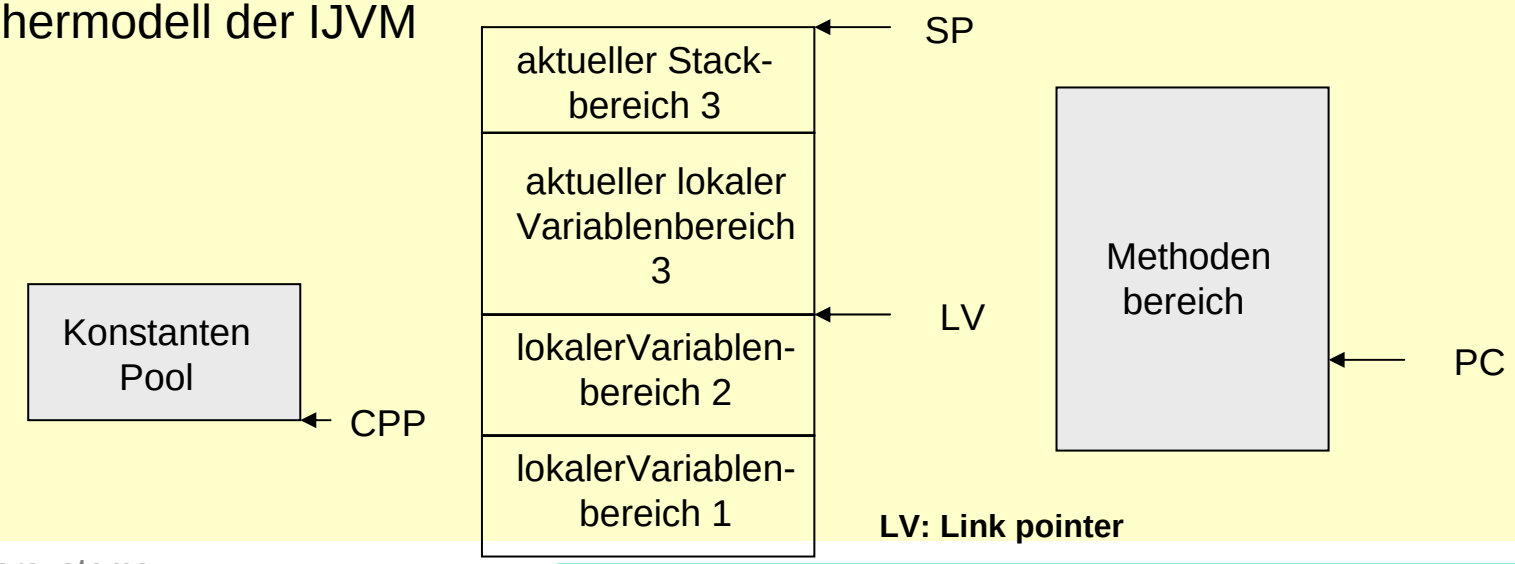
FP (Frame Pointer)

PC (Program Conter)

TOS (Top of Stack)

CPP (Constants Pool Pointer)

Speichermodell der IJVM



JVM Befehlssatz

Hex	Mnemonik	Bedeutung
0x10	BIPUSH *byte*	Schiebe Byte auf den Stapel
0x59	DUP	Kopiere oberstes Stapelwort und schiebe es auf den Stapel zurück
OxA7	GOTO *offset*	Unbedingter Sprung
0x60	IADD	Wirf zwei Wörter vom Stapel; schiebe ihre Summe
Ox7E	IAND	Wirf zwei Wörter vom Stapel; schiebe boolesches AND
0x99	IFEG *offset*	Wirf Wort vom Stapel und verzweige, falls es Null ist
Ox9B	IFLT *offset*	Wirf Wort vom Stapel und verzweige, falls es kleiner als Null ist
Ox9F	IF_ICMPEO *offset*	Wirf zwei Wörter vom Stapel; verzweige, falls gleich
0x84	I INC *varnum const*	Addiere eine Konstante zu einer lokalen Variablen
0x15	ILOAD *varnum*	Schiebe lokale Variable auf Stapel
OxB6	INVOKEVIRTUAL *disp*	Rufe eine Methode auf
0x80	IOR	Wirf zwei Wörter vom Stapel; schiebe boolesches OR
OxAC	IRETURN	Gib Integer-Wert von Methode zurück
0x36	ISTORE *varnum*	Schiebe Wort vom Stapel und speichere es in lokaler Variable
0x64	ISUB	Schiebe zwei Wörter vom Stapel; schiebe ihre Differenz
0x13	LDC_W *index*	Schiebe Konstante vom Konstantenpool auf den Stapel
0x00	NOP	Tue nichts
0x57	POP	Lösche oberstes Stapelwort
Ox5F	SWAP	Vertausche die beiden obersten Stapelwörter
OxC4	WIDE	Präfixinstruktion; nächste Instruktion hat einen 16-Bit-Index

Programmierbeispiel

Java Fragm.		JVM Assembler		Kommentar	HEX Code
i = j + k;	1		ILOAD j	// i = j + k	0x15 0x02
if (i == 3)	2		ILOAD k		0x15 0x03
k = 0;	3		IADD		0x60
else	4		ISTORE i		0x36 0x01
j = j - 1;	5		ILOAD i	// if (i < 3)	0x15 0x01
	6		BIPUSH 3		0x10 0x03
	7		IF_ICMPEO L1		0x9F 0x00 0xOD
	8		ILOAD j	//- 1	0x15 0x02
	9		BIPUSH 1		0x10 0x01
	10		ISUB		0x64
	11		ISTORE j		0x36 0x02
	12		GOTO L2		0xA7 0x00 0x07
	13	L1:	BIPUSH 0	// k = 0	0x10 0x00
	14		ISTORE k		0x36 0x03
	15	L2:			

Zusammenfassung

Die Erhöhung der Komplexität von Rechnern durch technischen Fortschritt kann ausgenutzt werden durch:
- mehr Register
- größerer Mikroprogrammspeicher
- mehr und komplexere Befehle
- mehr und komplexere Adressierungsarten
- mehrere unabhängige Funktionseinheiten
- zusätzliche Eigenschaften zur Unterstützung des Betriebssystems

Zwischen Befehlssatz, Registersatz und Adressierungsmöglichkeiten besteht eine komplexe Beziehung.

Eigenschaften der 68K Architektur:
- mehrere allgemeine Register
- orthogonale Architektur
- Unterstützung von Zugriffsschutz
- leistungsfähiges, asynchrones Bussystem
- Coprozessorschnittstelle