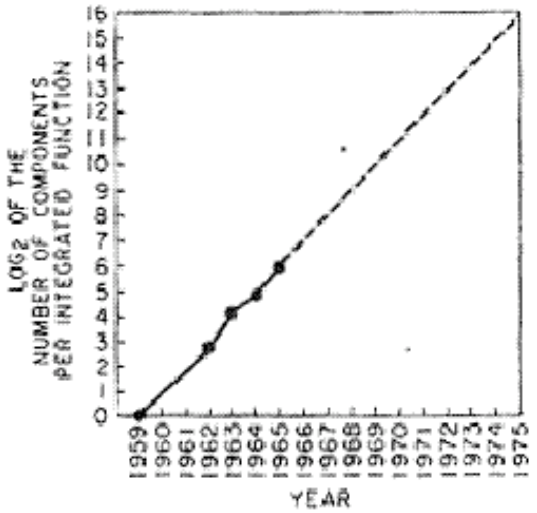
Scheller, besser, komplexer, einfacher,....

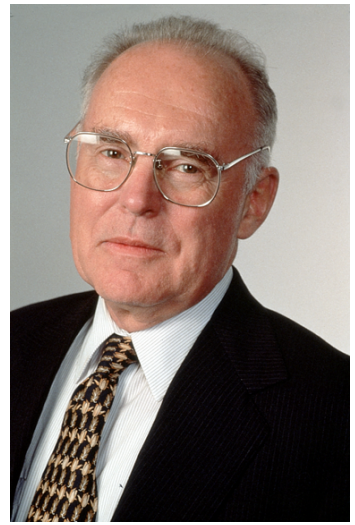
Alternativen beim Entwurf eines Prozessors an Beispielen

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? Mehr Register? Mehr spezielle Funktionseinheiten?

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?

Alternativen in der Rechnerarchitektur

Befehlssatz und Adressierungsarten
- **Weniger ist Mehr?**

Schnittstelle zum Speicher und zur Peripherie
- **asnchrone und synchrone Busse**
- **Wort- oder Byteorientierung**

Registersatz
- **generelle Frage: wie viele?**
- **Auswirkungen auf den Befehlssatz**

Erweiterung des Befehlssatzes
- **Alternativen: Co-Prozessor vs. Mikroprogrammierung**
- **68K Co-Prozessorschnittstelle**

Befehlssatz und Adressierungsarten: Weniger ist Mehr?

CISC vs. RISC

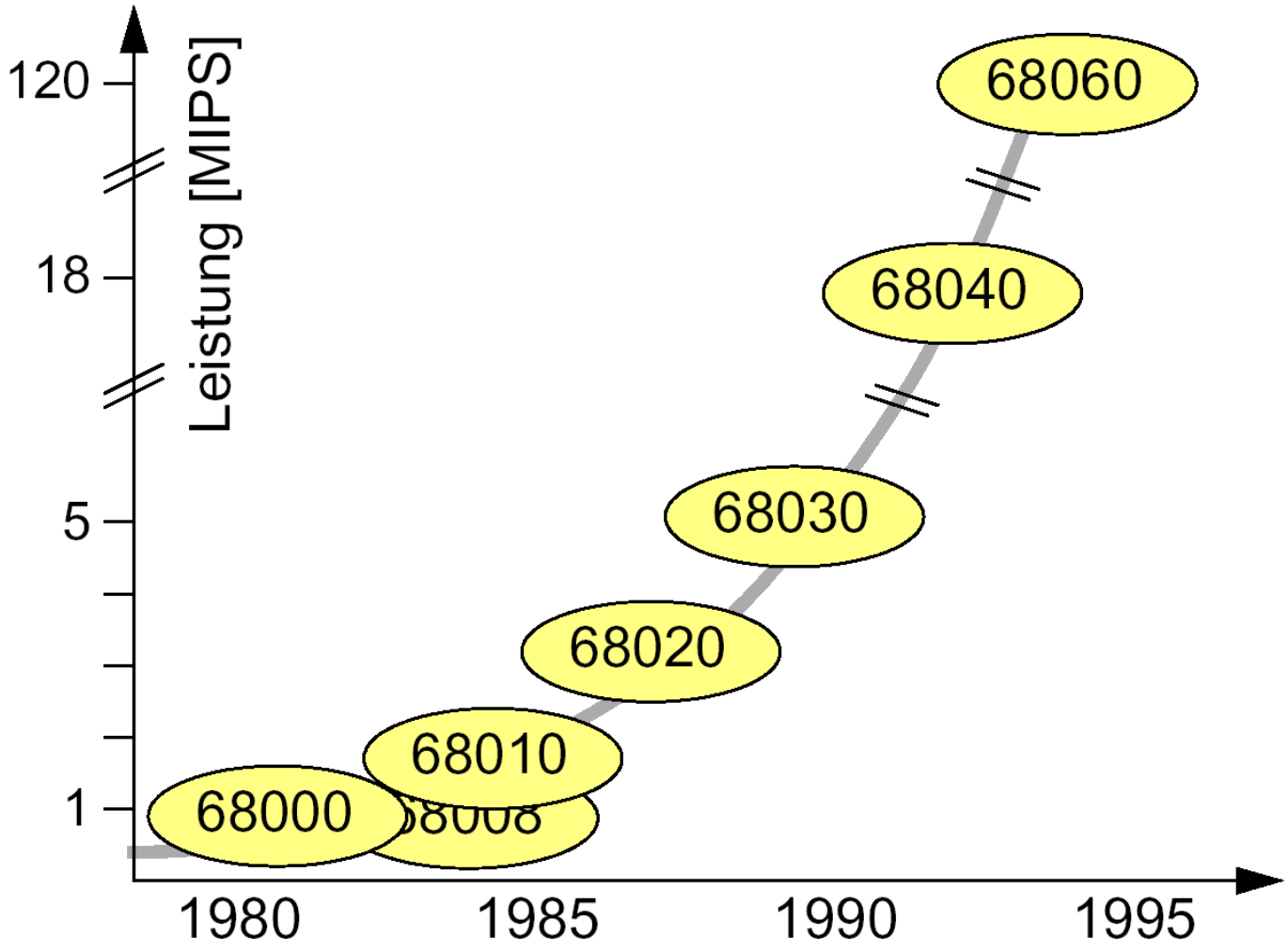
Mikroprogrammierung
Speicher-Register-Architektur
Komplexe Befehle und Adressierungsarten
Mehrzyklen-Befehle

DEC VAX Architektur
680x0 Familie
80x86 Familie

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

ALPHA, ARM, MIPS, SPARC,

Die CISC-Schiene: 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)
- Dynamische Anpassung der Wortlänge auf dem Bus
- Flexibles "Alignement" und Packung ungterschiedlicher Datentypen
- Co-Prozessor Schnittstelle (ab 68020)

68020 Überblick:

- Architektur - Programmiermodell, Ausführungsmodi
- Adressierungsarten, Befehlssatz

- Bussystem
- Coprozessor-Schnittstelle

68020 Eigenschaften

Erster “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:

Warum überhaupt 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.
Mehrere 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

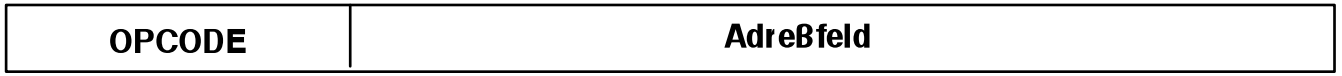
Asmb-Programmierer kann durch Ausnutzung der Programmsemantik eine sehr effiziente Registernutzung vornehmen, z.B. Bewahrung von Registerinhalten über Prozeduraufrufe hinweg oder sogar über Context-Wechsel.

Compiler: Löschen aller Register bei Unterprogrammsprung.

Befehls-formate:

Ein-Adreß-Befehl

Beisp.:
Motorola 6809

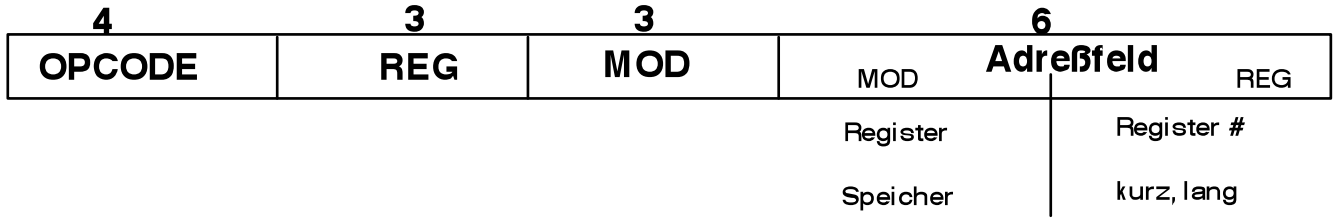


Zwei-Adreß-Befehl

680x0,
Intel 386,486

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

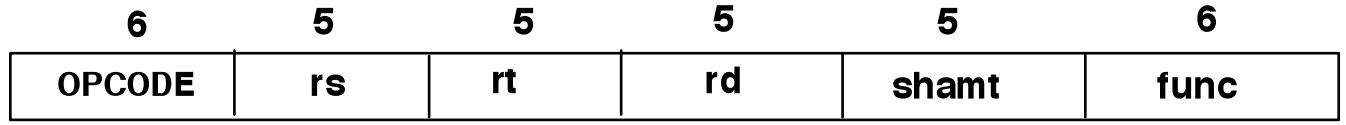
Beisp.:
Motorola
680x0



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 Register

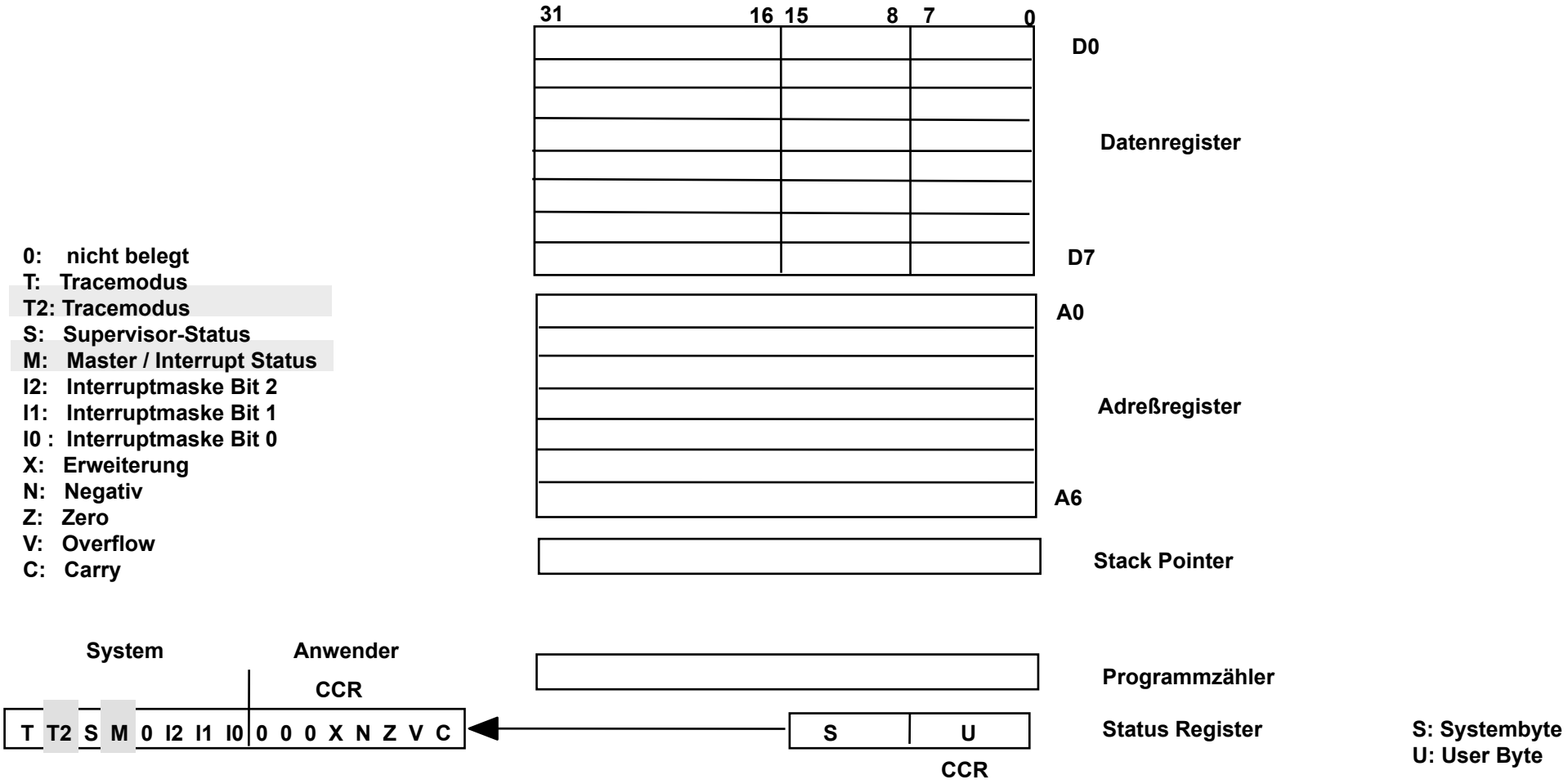
Datenregister:

- **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:

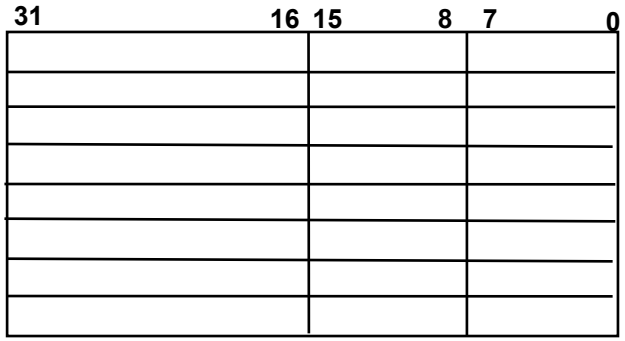
- **werden als Basisregister bei der Adressierung verwendet**
- **werden als 32-Bit Einheiten behandelt und können nur Operationen auf 32-Bit Datentypen ausführen**
- **arithmetisch/logische Operationen auf Adreßreg. modifizieren nicht die Bedingungsflags im CCR**
- **Adreßregister #7 dient als Stack 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.**

68K Programmiermodell

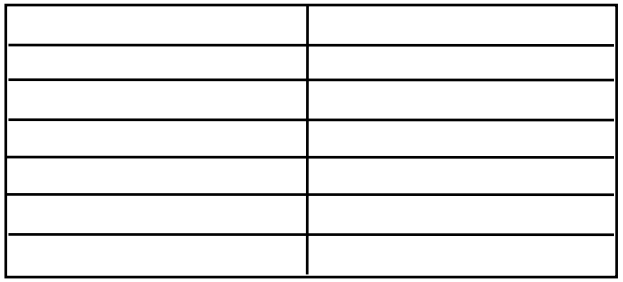


68020 Programmiermodell

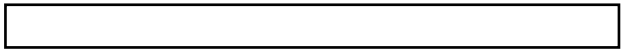
Erweiterungen zum 68000 Programmiermodell



Datenregister

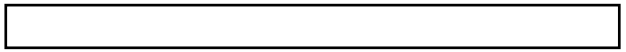


Adreßregister



User Stack Pointer

Supervisor Stack Pointer



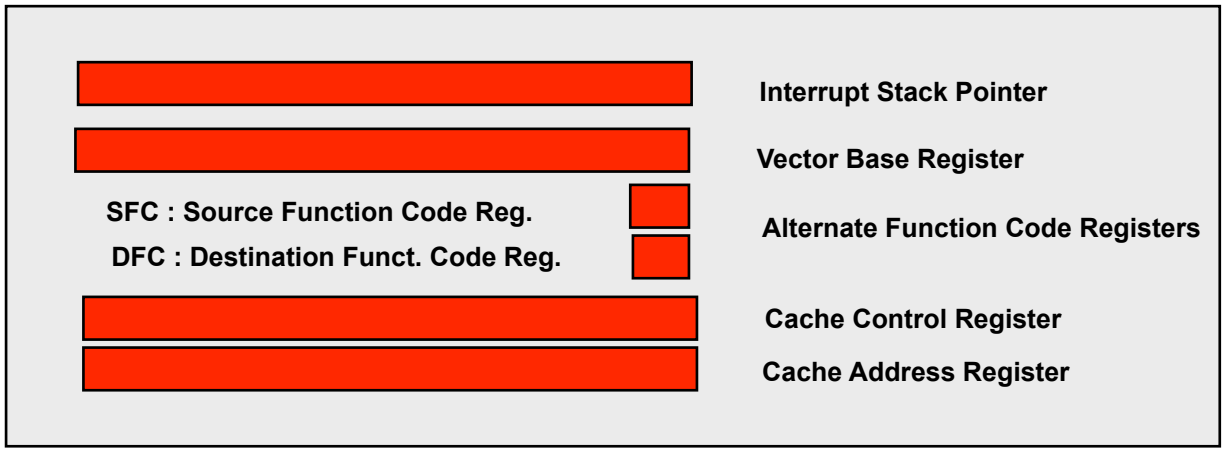
Programmzähler

Status Register

S: Systembyte
U: User Byte

Nur im System-modus verf.

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



Interrupt Stack Pointer

Vector Base Register

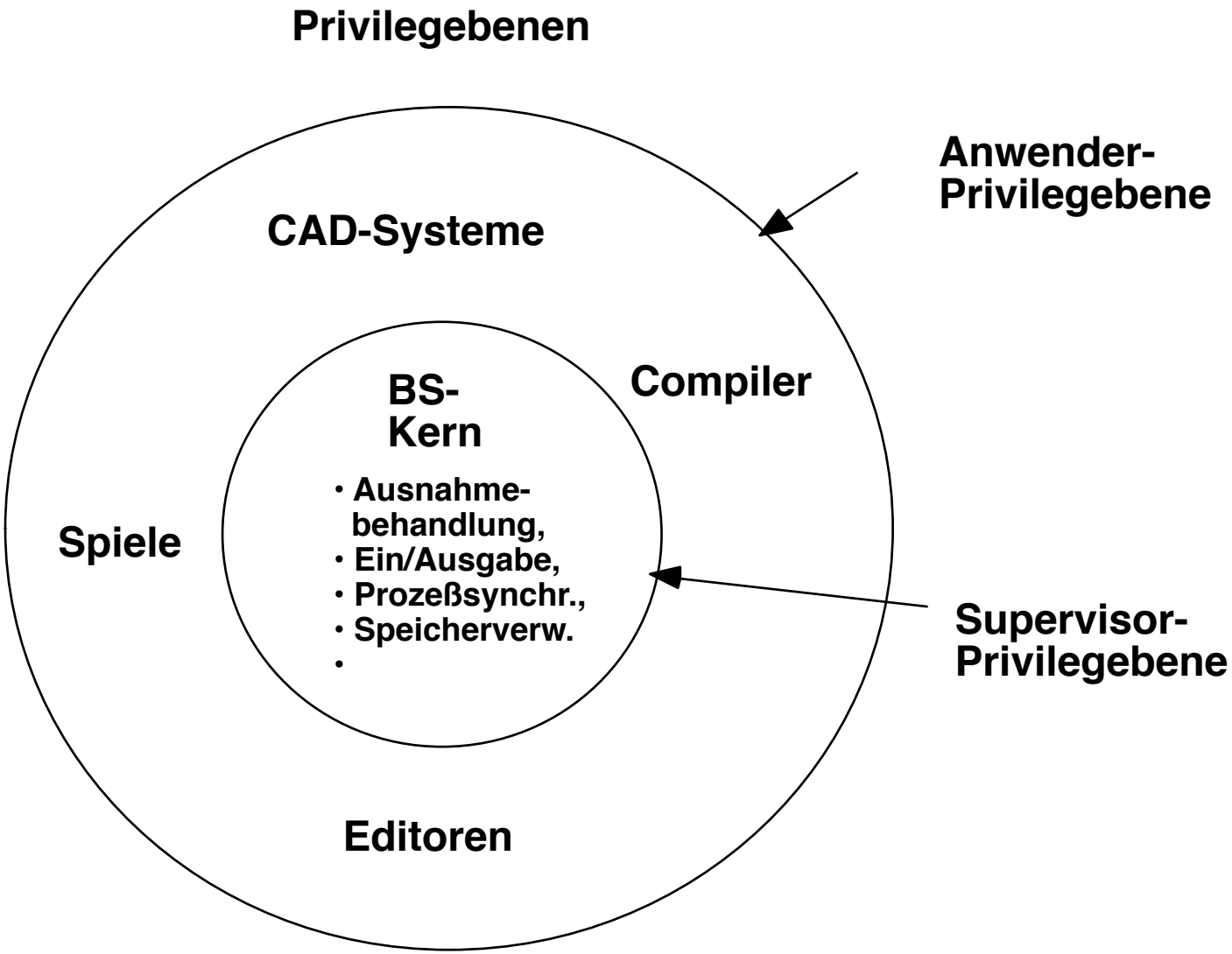
Alternate Function Code Registers

Cache Control Register

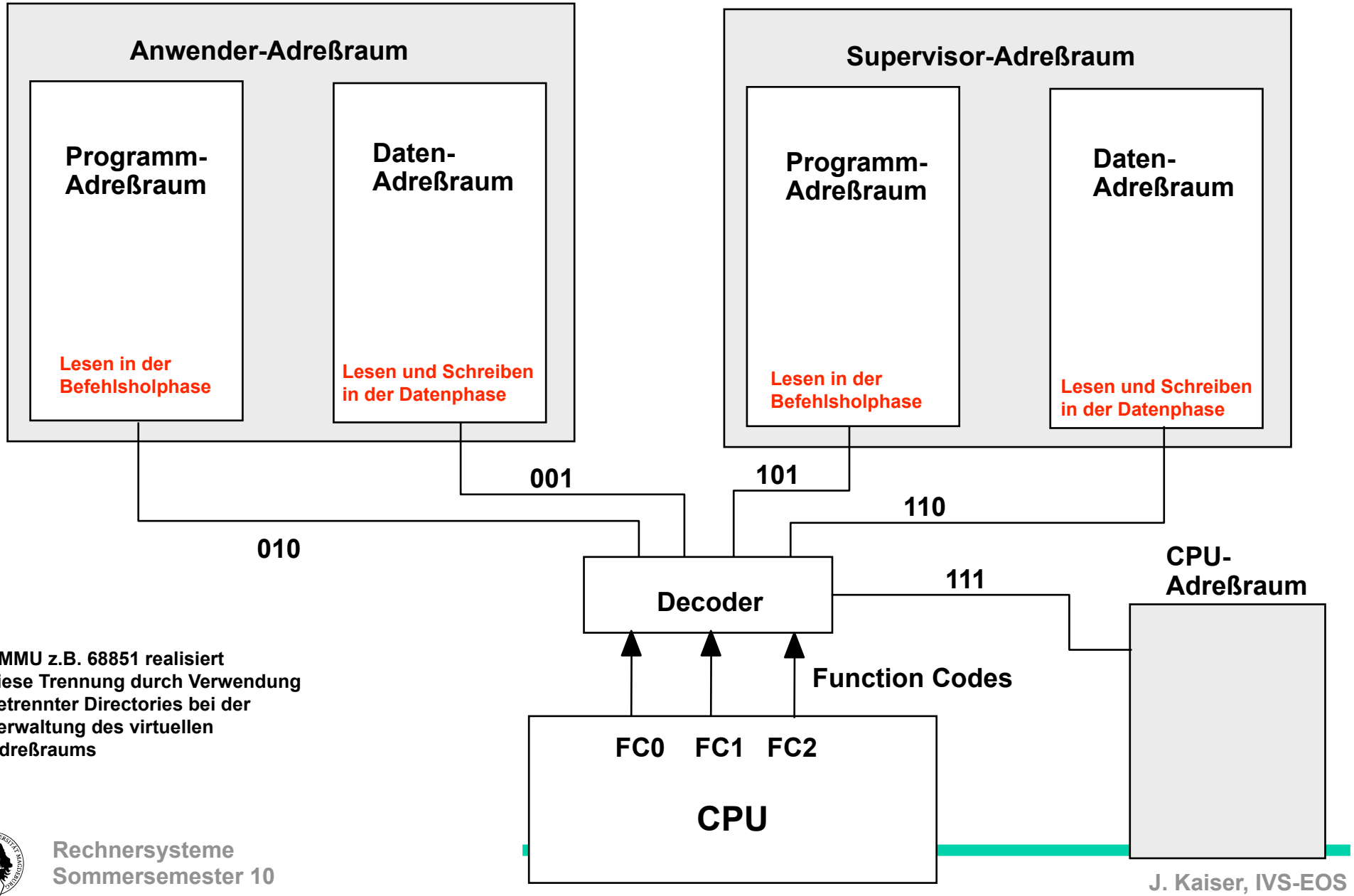
Cache Address Register

68020 Erweiterungen

"Sichtbarkeit" von 68020 Register



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:

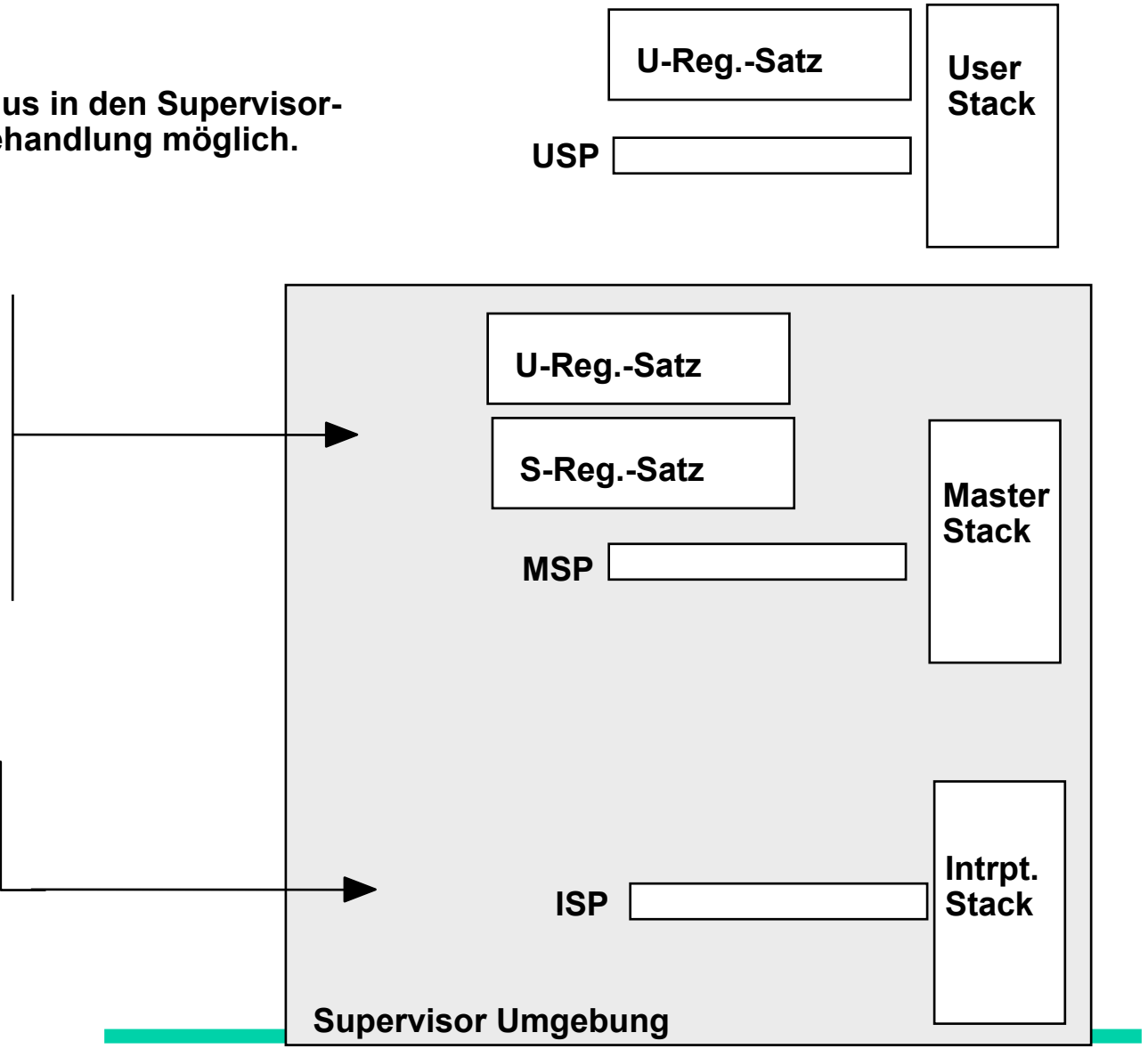
- Supervisor Ebene
  - S-Bit im Status Register ist gesetzt
  - Function Codes signalisieren den Supervisor Adreßraum
  - M-Bit im Satus Register unterscheidet zwischen “Master State“ und “Interrupt State“. Interrupt State entspricht dem 68000/68008/68010 Supervisor State.

- 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)

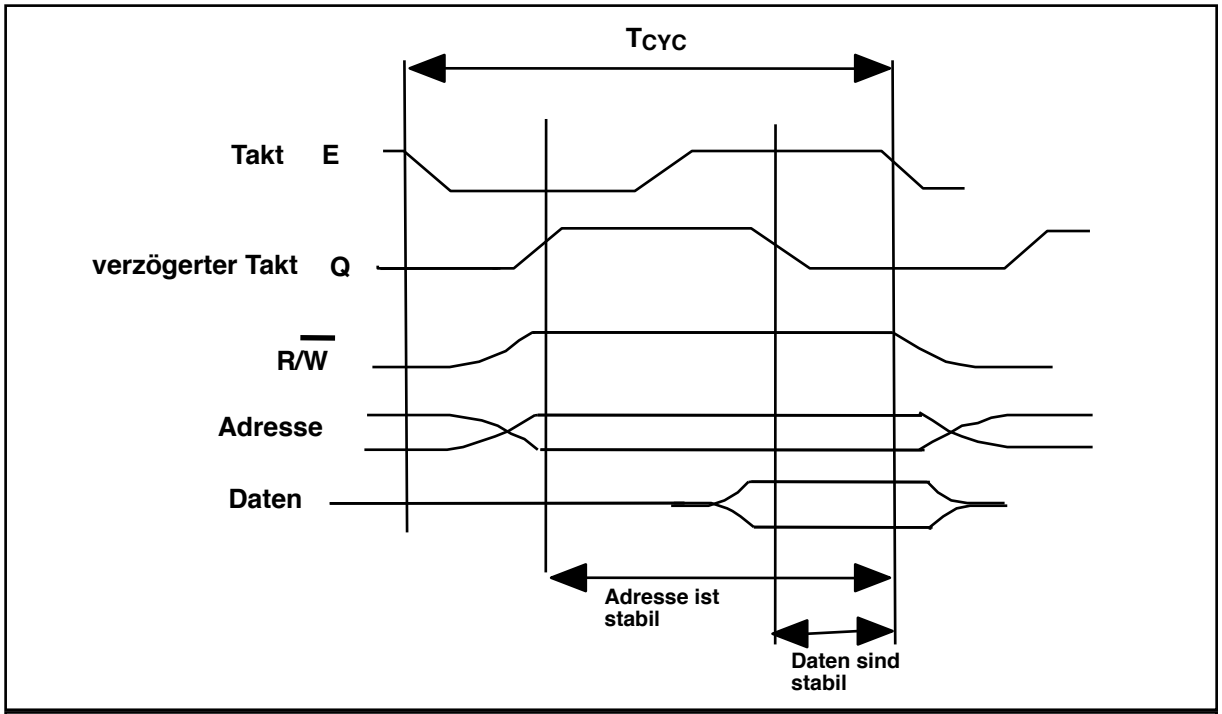
Burst-Modus - ein Buszyklus/ein Datum

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

dynamische (automatische) Busanpassung - explizite (programmierte) Busanpassung

automatische Fehlerbehandlung - programmierte Fehlerbehandlung

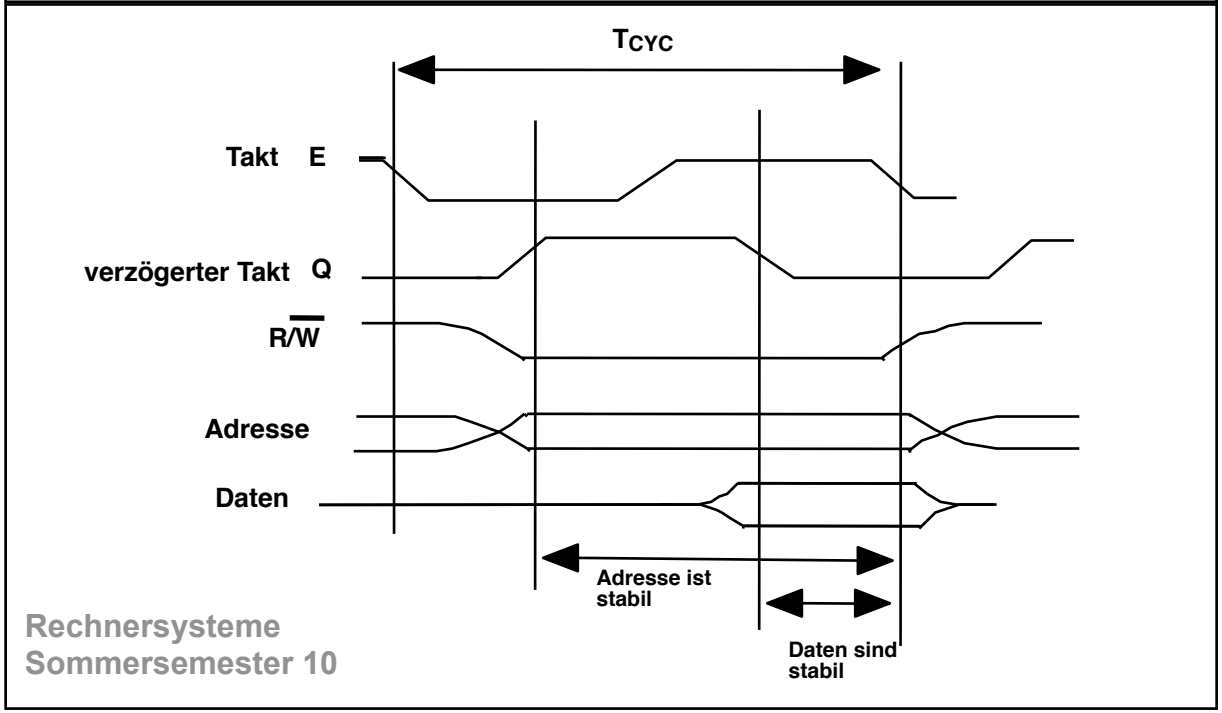
Einfaches, synchrones Busprotokoll des MC 6809



Lese-Zyklus

getrennte Busse,
feste Wortlänge für:

Adressen	**16 Bit**
Daten	**8 Bit**

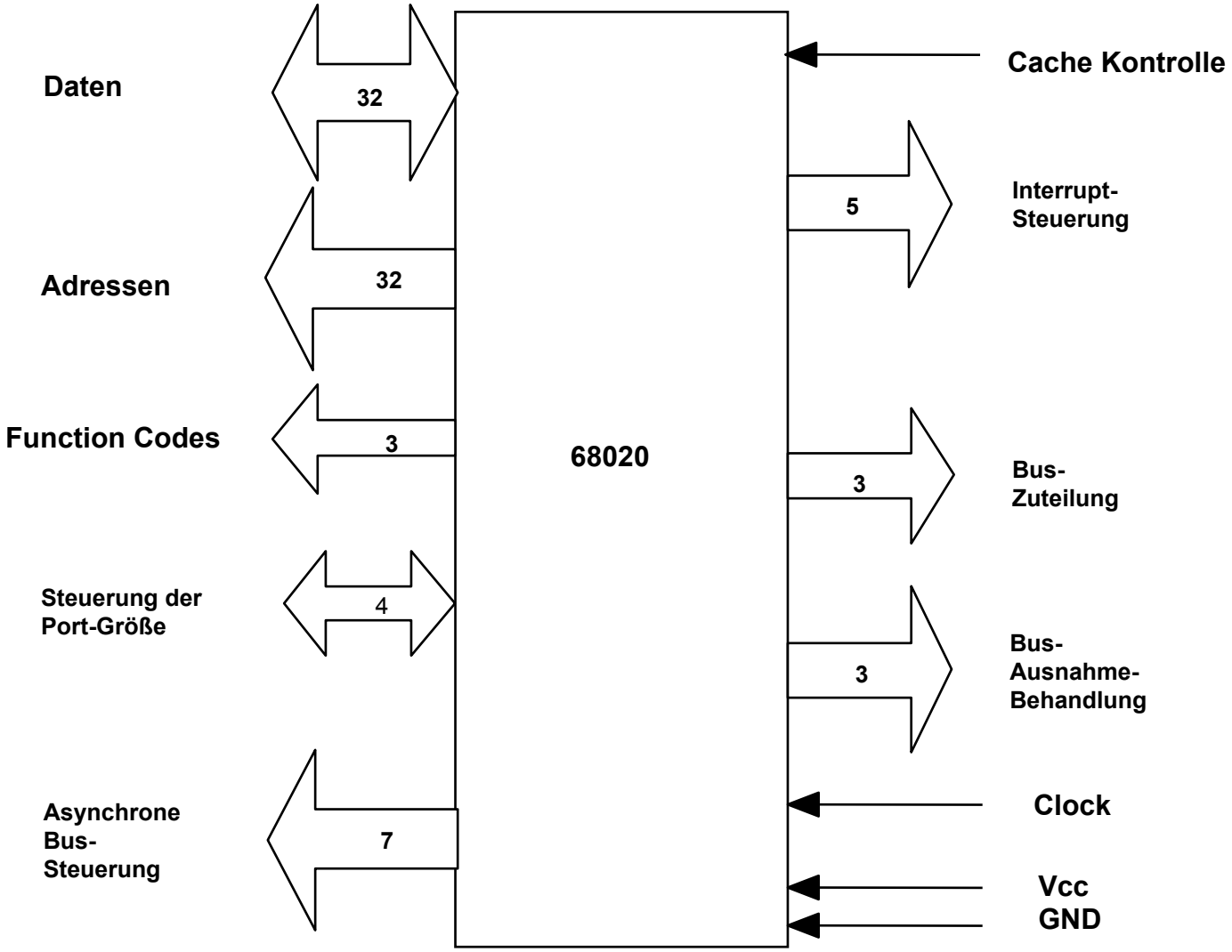


Schreib-Zyklus

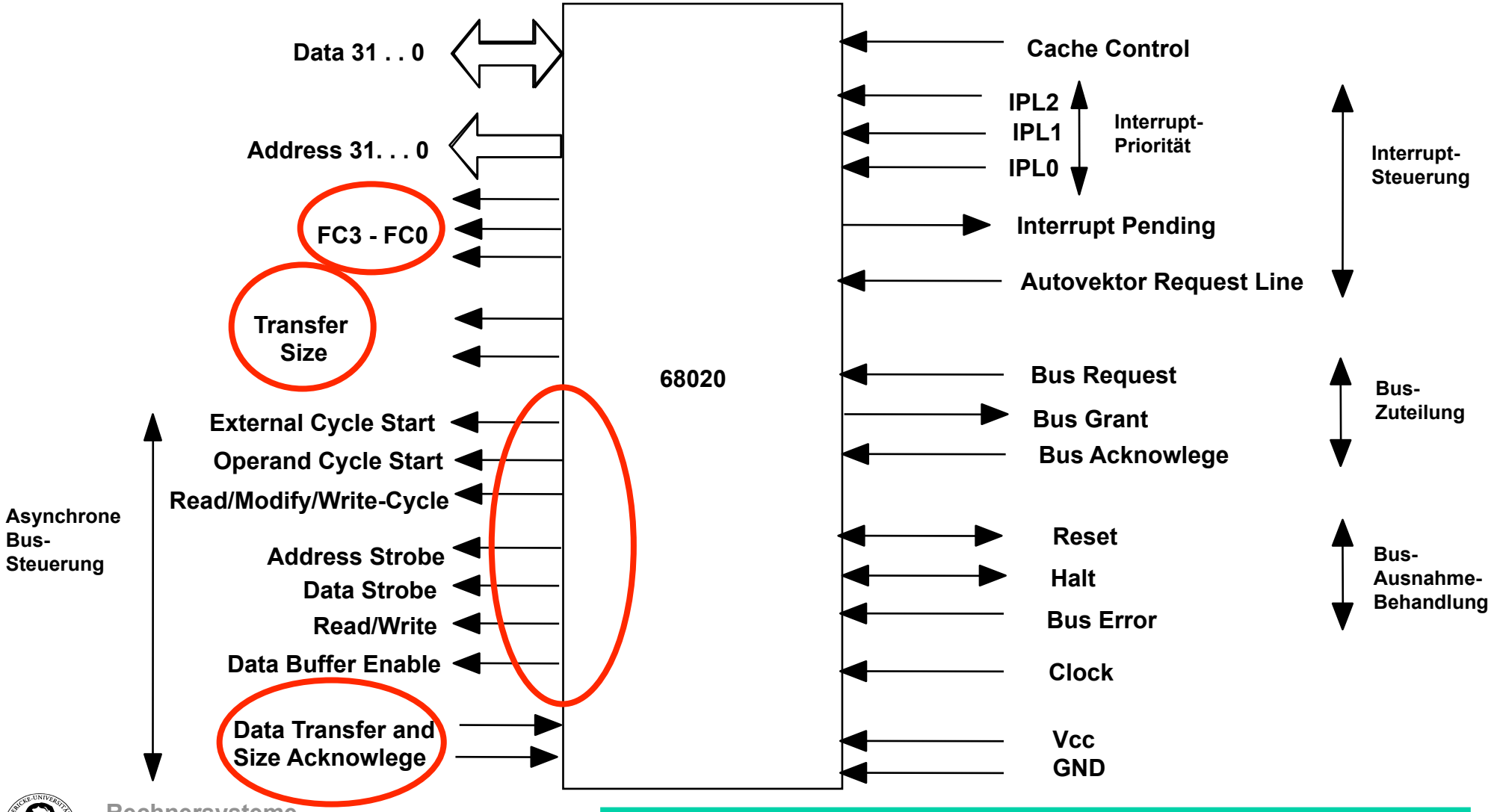
68020 Bussystem

- 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 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:

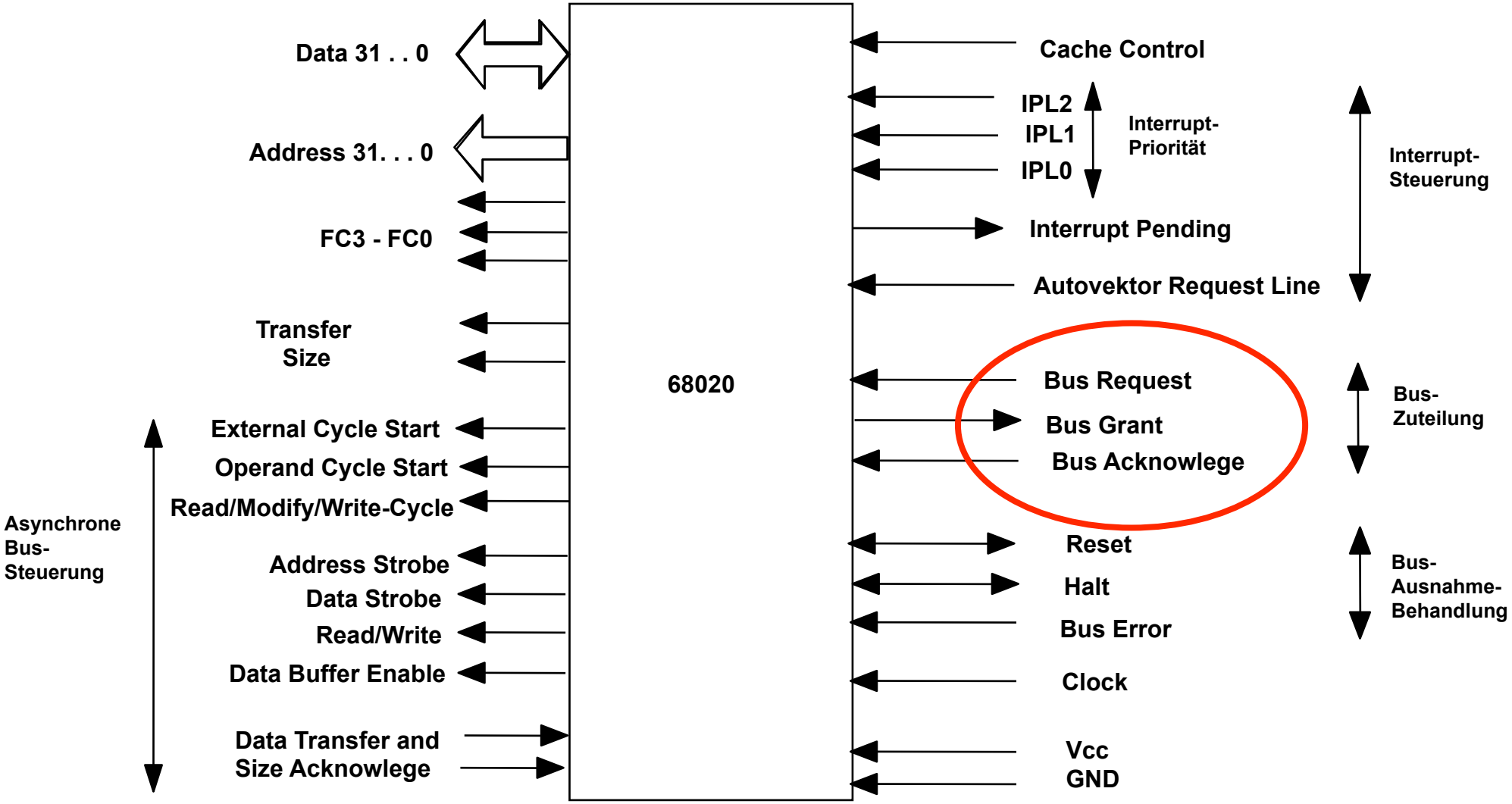
1. Daten in Datenpuffer speichern
2. Deaktivieren des Datenstrobes
3. Deaktivieren des Adreßstrobes
4. Deaktivieren des Signals “Data Buffer Enable“

Buszyklus beenden:

1. Daten vom Bus nehmen
2. DSACK deaktivieren

Nächsten Buszyklus starten

68020 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:

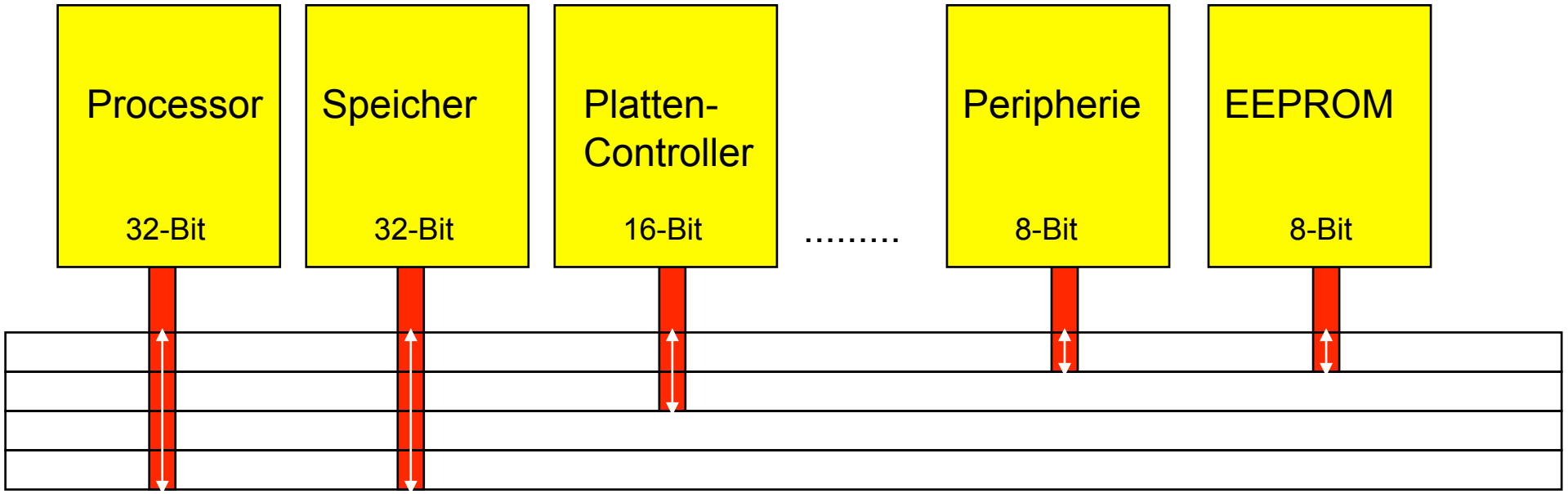
1. Durchführung der Datentransfers (Lesen und/oder Schreiben) ohne jegliche Kontrolle durch den Prozessor.

Freigabe des Bus Masters:

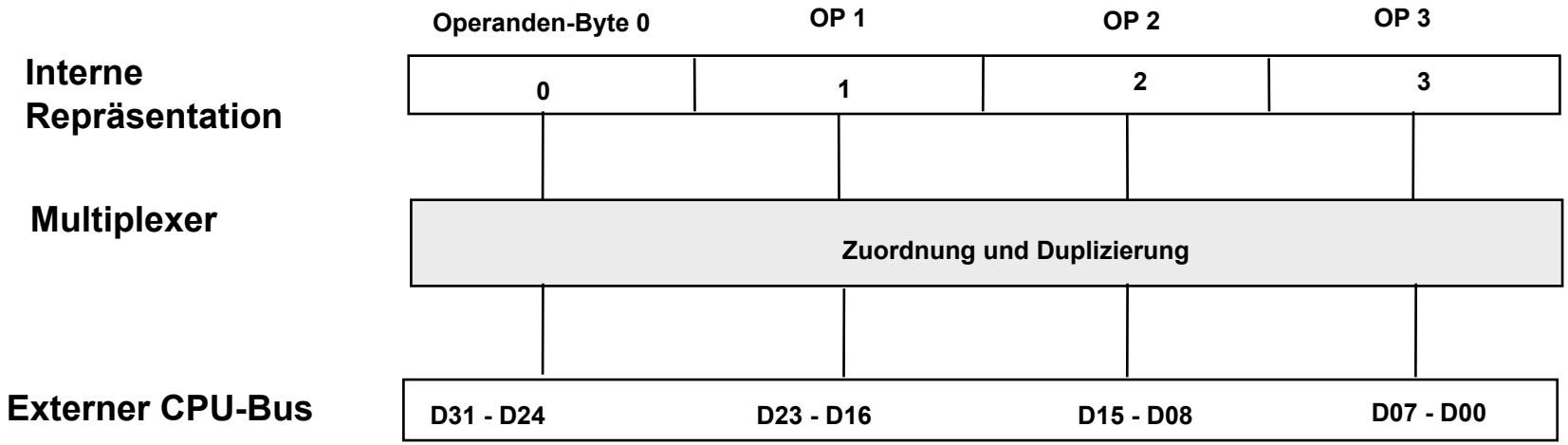
1. Deaktivieren von “Bus Grant Acknowlege“

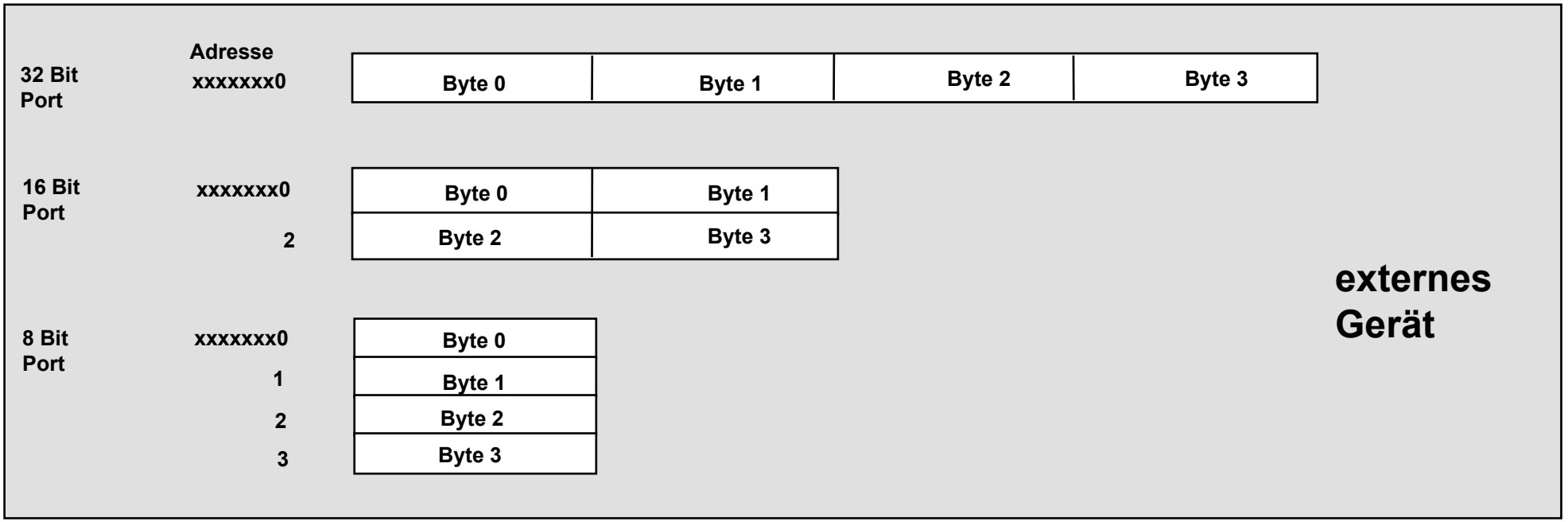
Neues Arbitrierungszyklus oder Fortführen der Prozessoroperationen

Dynamische (automatische) Busanpassung

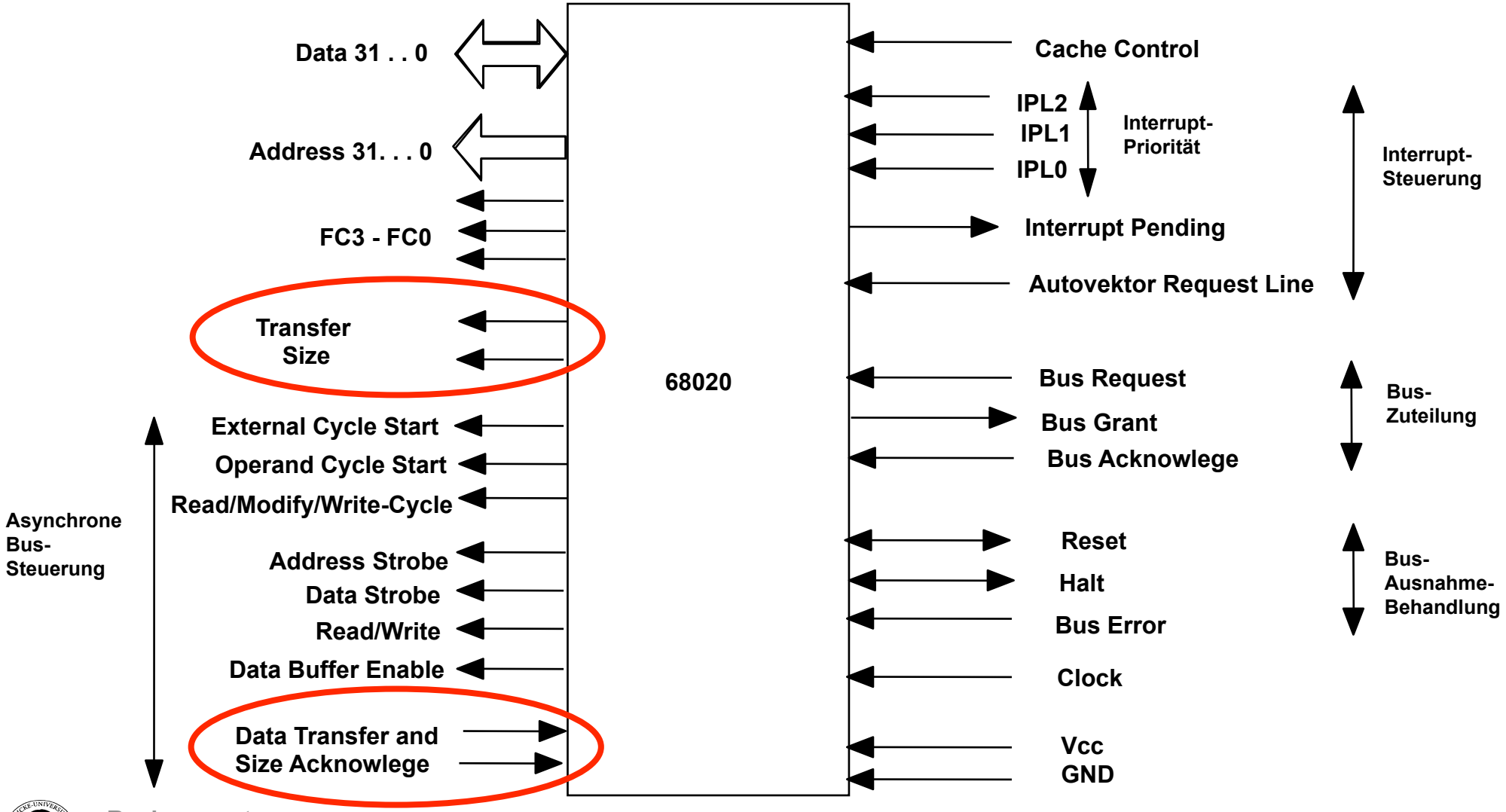


Dynamische Busanpassung





68020 Steuersignale (Detail)



Dynamische Busanpassung

68020 signalisiert über die Ausgänge SIZ0 und SIZ1 die Port-Größe für den durchzuführenden 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

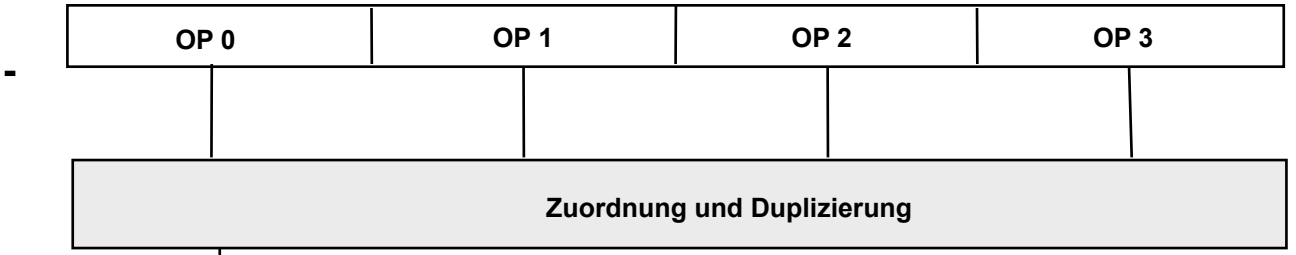
Peripheres Gerät signalisiert über die Eingänge DSACK0 un DSACK1 die erforderliche Port-Größe

Codierung der DSACK- Eingänge

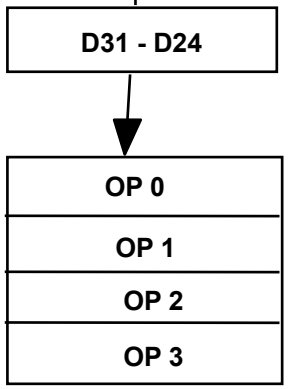
DSACK1	DSACK0	Resultat
1	1	Einfügen von Wartezyklen
1	0	vollst. Zyklus - DBPS : 8 Bit
0	1	vollst. Zyklus - DBPS : 16 Bit
0	0	vollst. Zyklus - DBPS : 32 Bit

Dynamische Busanpassung

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



Externer Bus



A1 A0

Adresse :

xx....0 0
xx....0 1
xx....1 0
xx....1 1

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

SIZ1	SIZ0	A1	A0
0	0	0	0
1	1	0	1
1	0	1	0
0	1	1	1

Periph. Kontrolle

DSACK1	DSACK0
0	1
0	1
0	1
0	1

1.Zyklus : OP0 wird übertragen
2.Zyklus : OP1 wird übertragen
3.Zyklus : OP2 wird übertragen
4.Zyklus : OP3 wird übertragen

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

99 Standardinstruktionen, eingeteilt in

16 OPCODE Klassen:

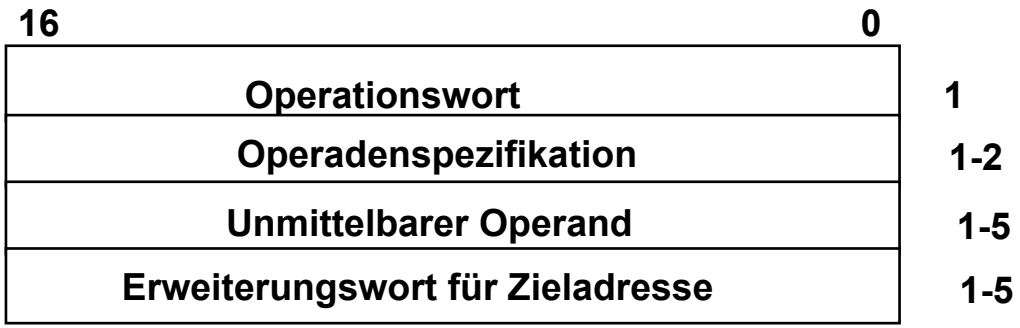
0000	Bit Manipulation/MOVEP/Immediate
0001	Move Byte
0010	Move Long
0011	Move Word
0100	Miscelleneous
0101	ADDQ/SUBQ/Scc/DBcc/TRAPcc
0110	Bcc/BSR/BRA
0111	MOVEQ
1000	OR/DIV/SBCD
1001	SUB/SUBX
1010	reserved
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	Shift/Rotate/Bit Field
1111	Coprocessor Interface

Datentypen

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

68020 Befehlformat:

Genereller Aufbau eines 68020 Befehls:

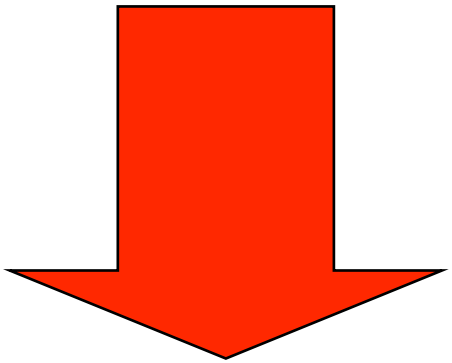


min. Befehlslänge: 2 Bytes

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. Grafik)

Anbindung an die CPU

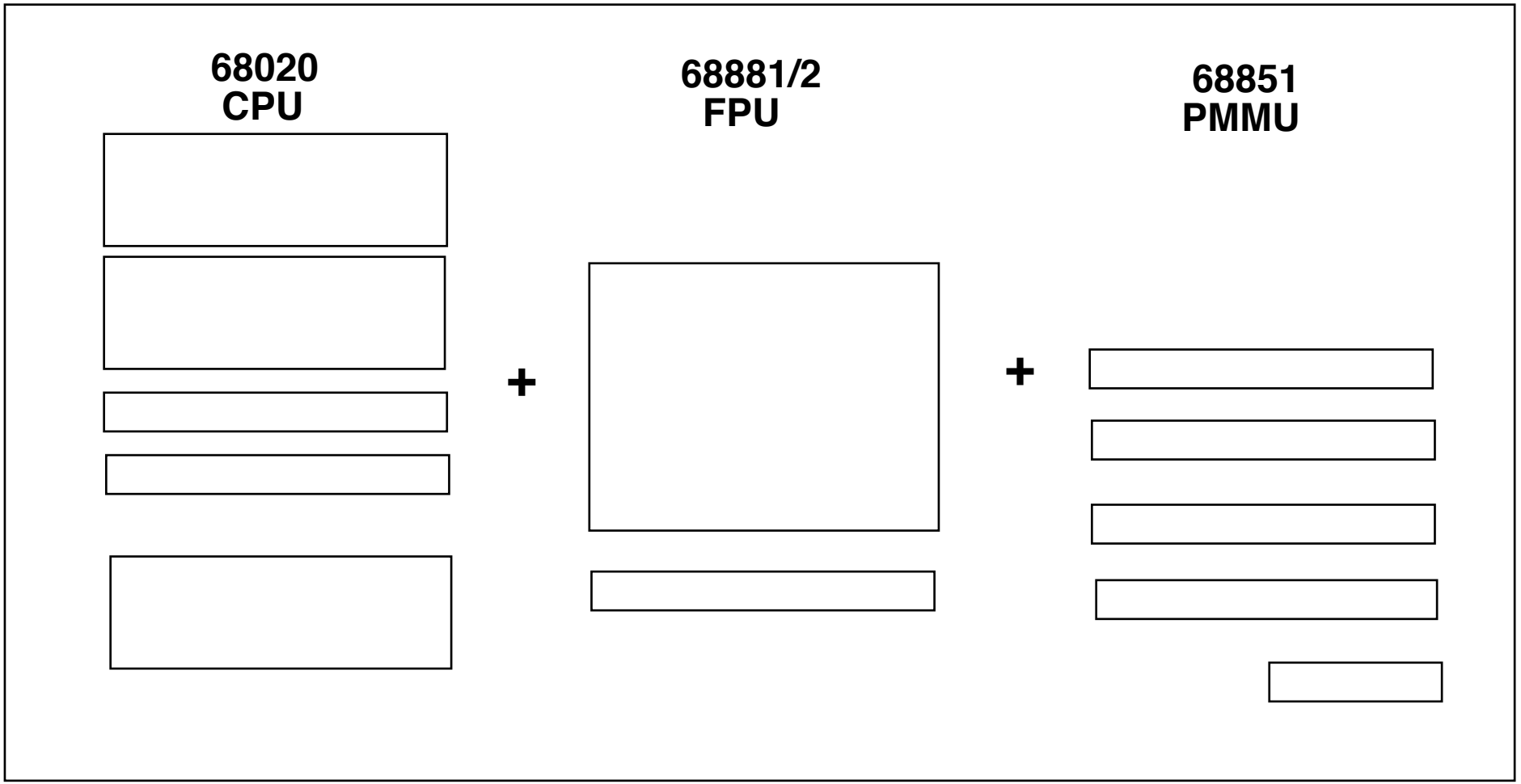
- Eigene Befehlsdekodierung (Instruction Tracker)
- Befehlsdekodierung wird von der CPU durchgeführt

- 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

Programmiermodell: Transparente Funktionserweiterung



CPU-Instruktionssatz + FPU-Instruktionssatz + PMMU-Instr.satz

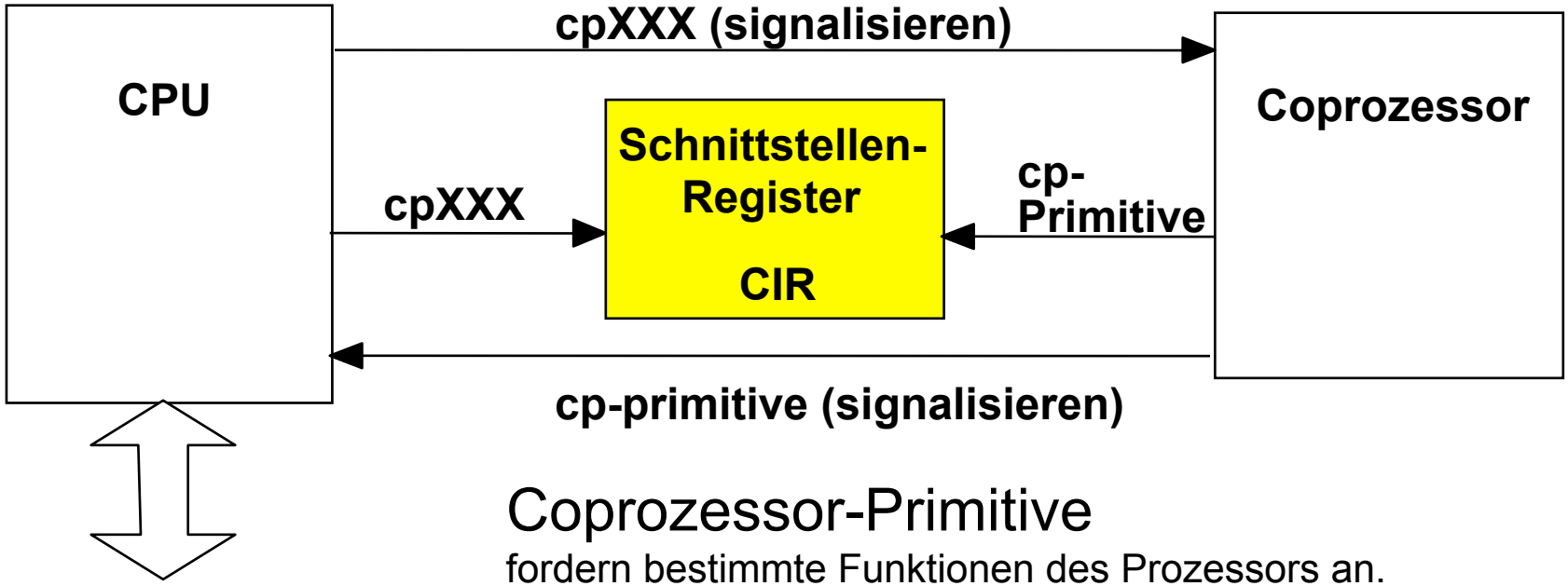
. . . . . fadd, jmul, fdiv, . . . pflush, pload, . . .

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.

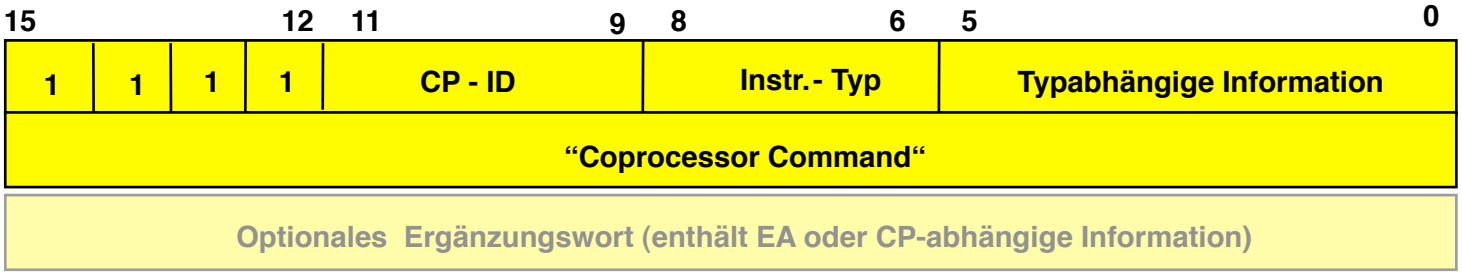
Unterstützung durch 68K Architektur (ab 68020)

Berechnung der effektiven Adresse

Durchführen der Speicherzugriffe

Emulation von Coprozessoren

Coprozessor Instruktion



CP-Operations-Wort

CP-Kommando-Wort

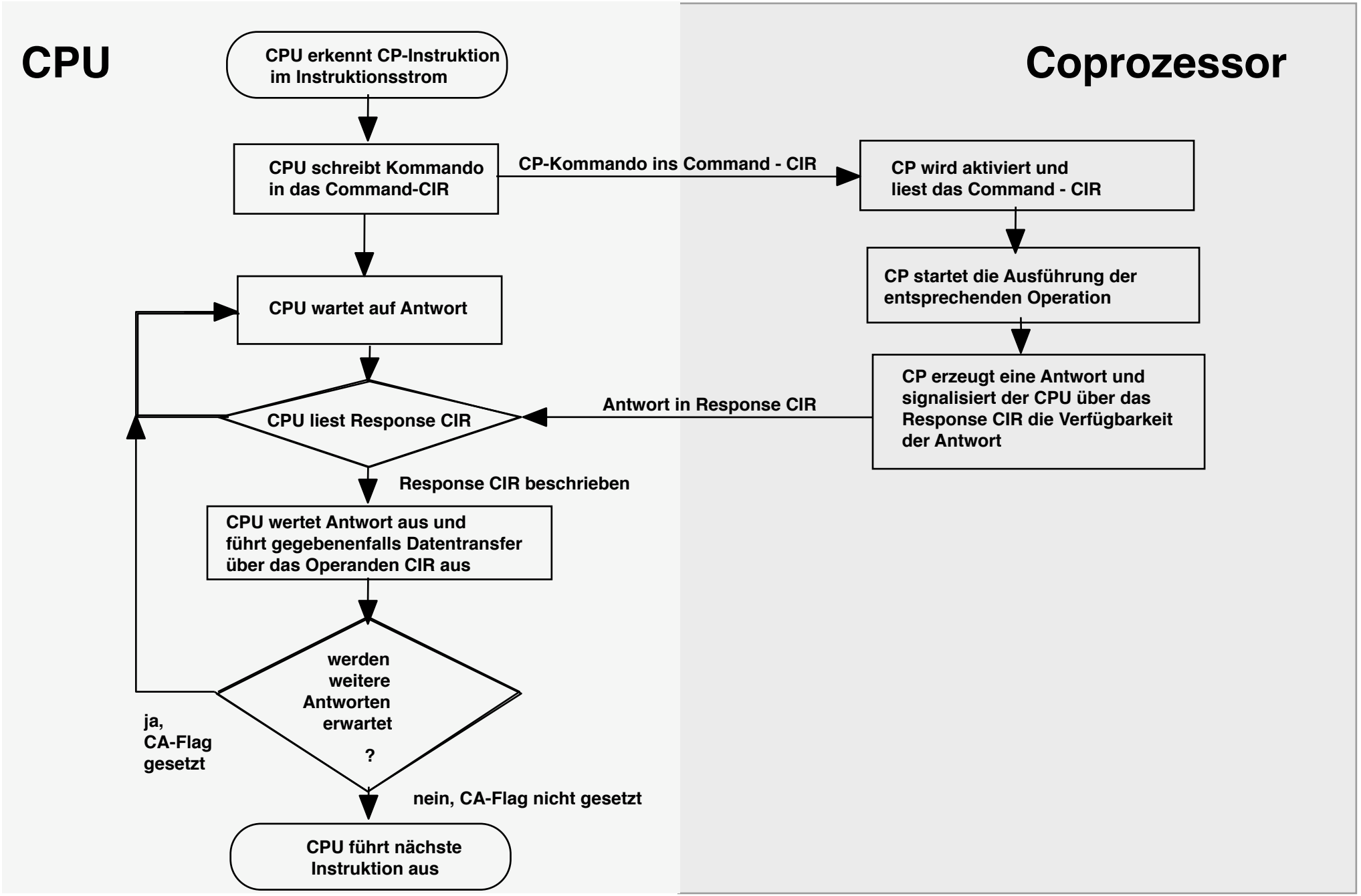
Bit 15 - 12 : F-Line Code. Dieser Code repräsentiert eine Coprozessor-Instruktion

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

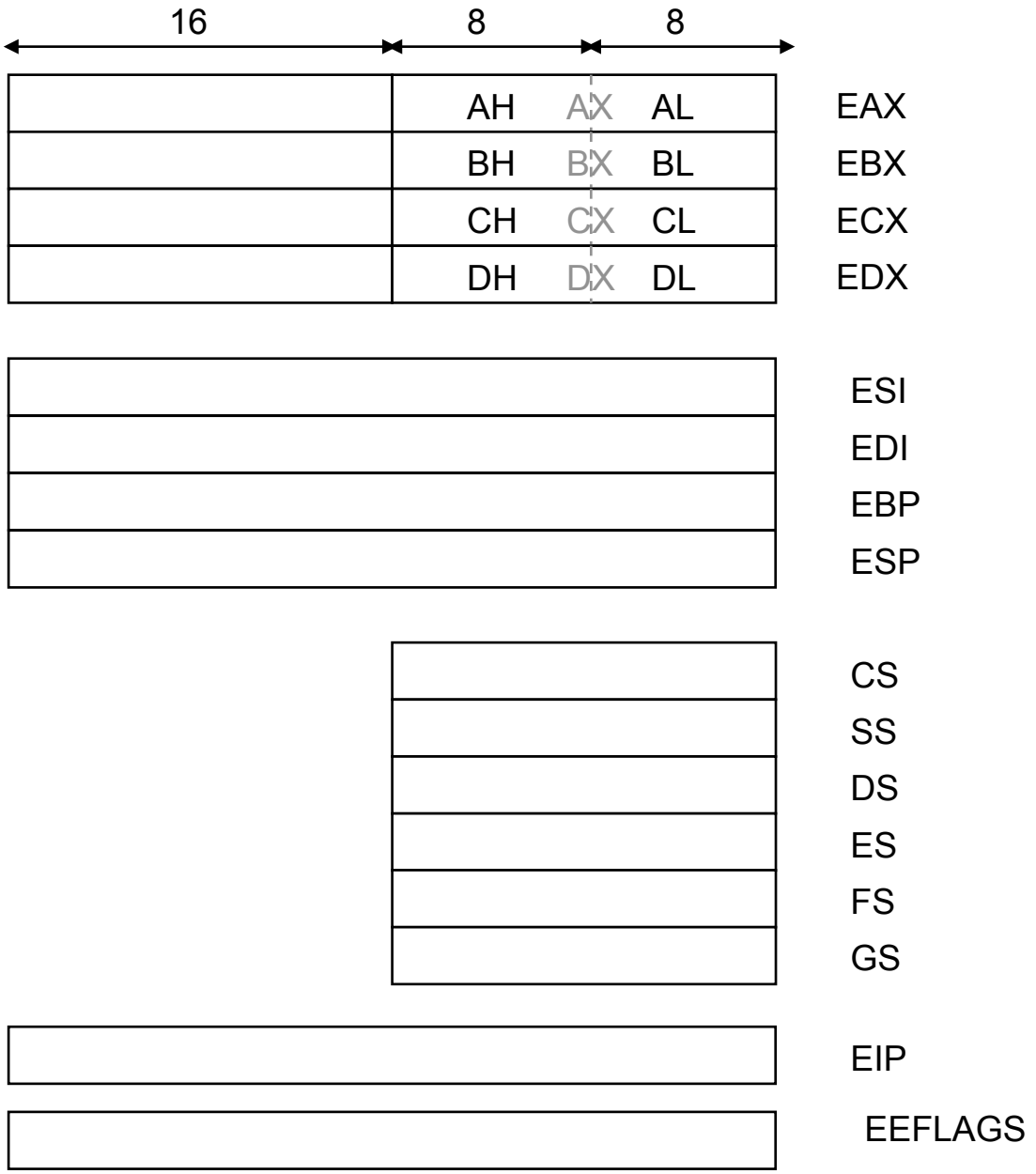
Zusammenfassung:

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

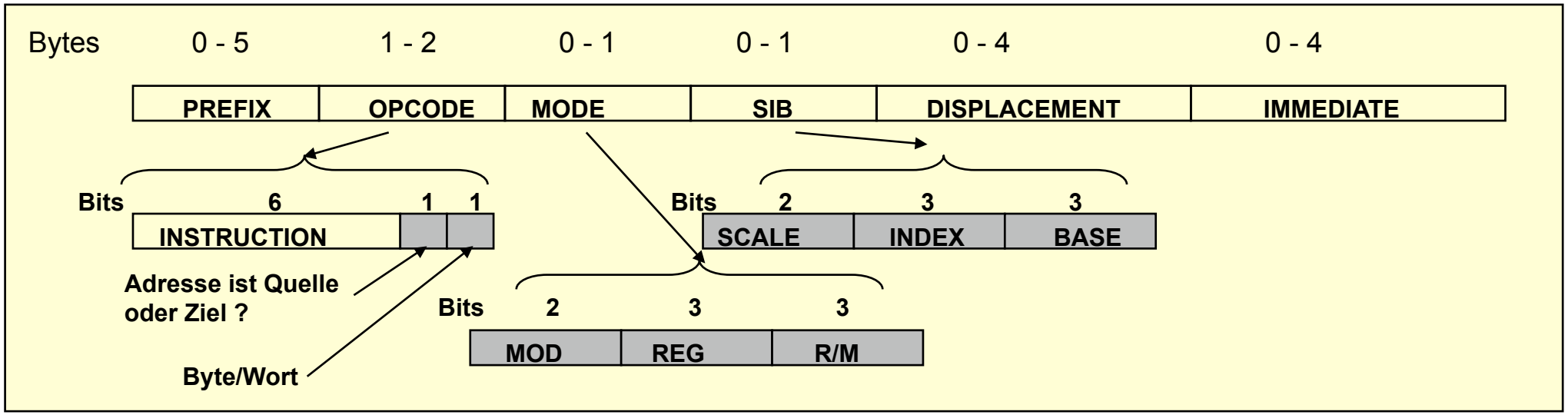
Alternativen für Register- und Befehlssätze

- Pentium
- Sparc
- IJVM
- Single Instruction Computer

(Haupt) Registersatz des Pentium 4



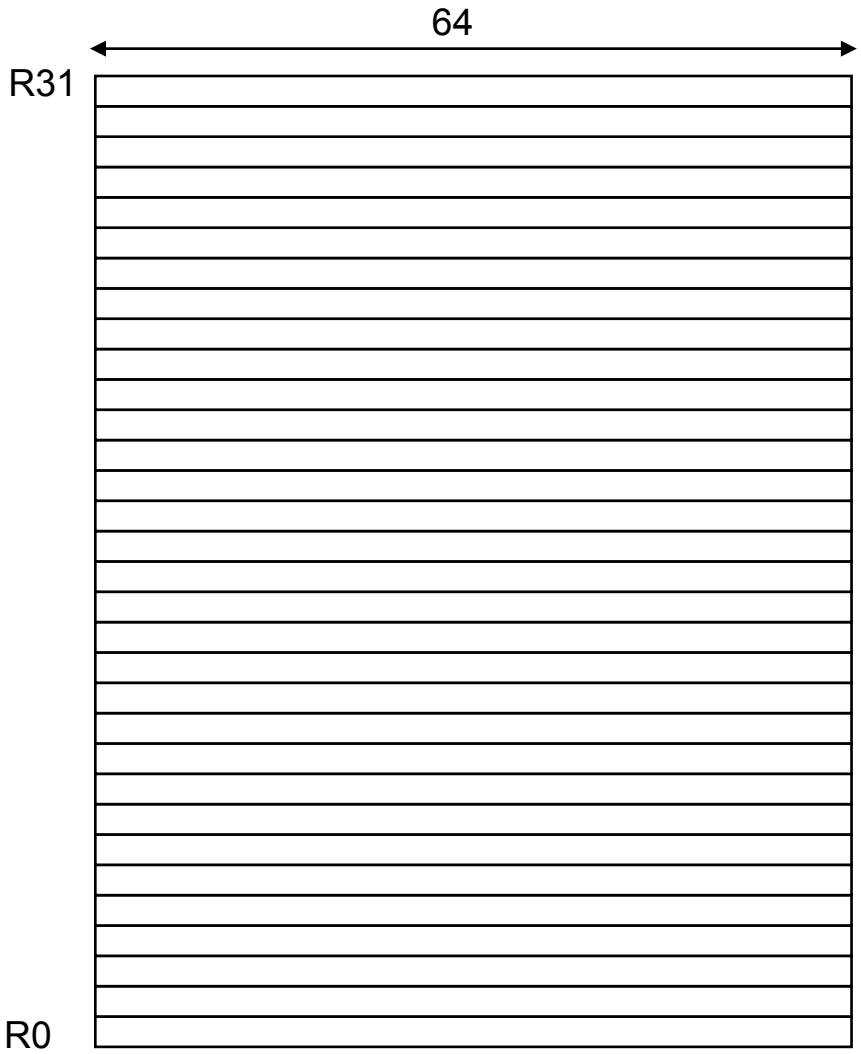
Adressierungsformen



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

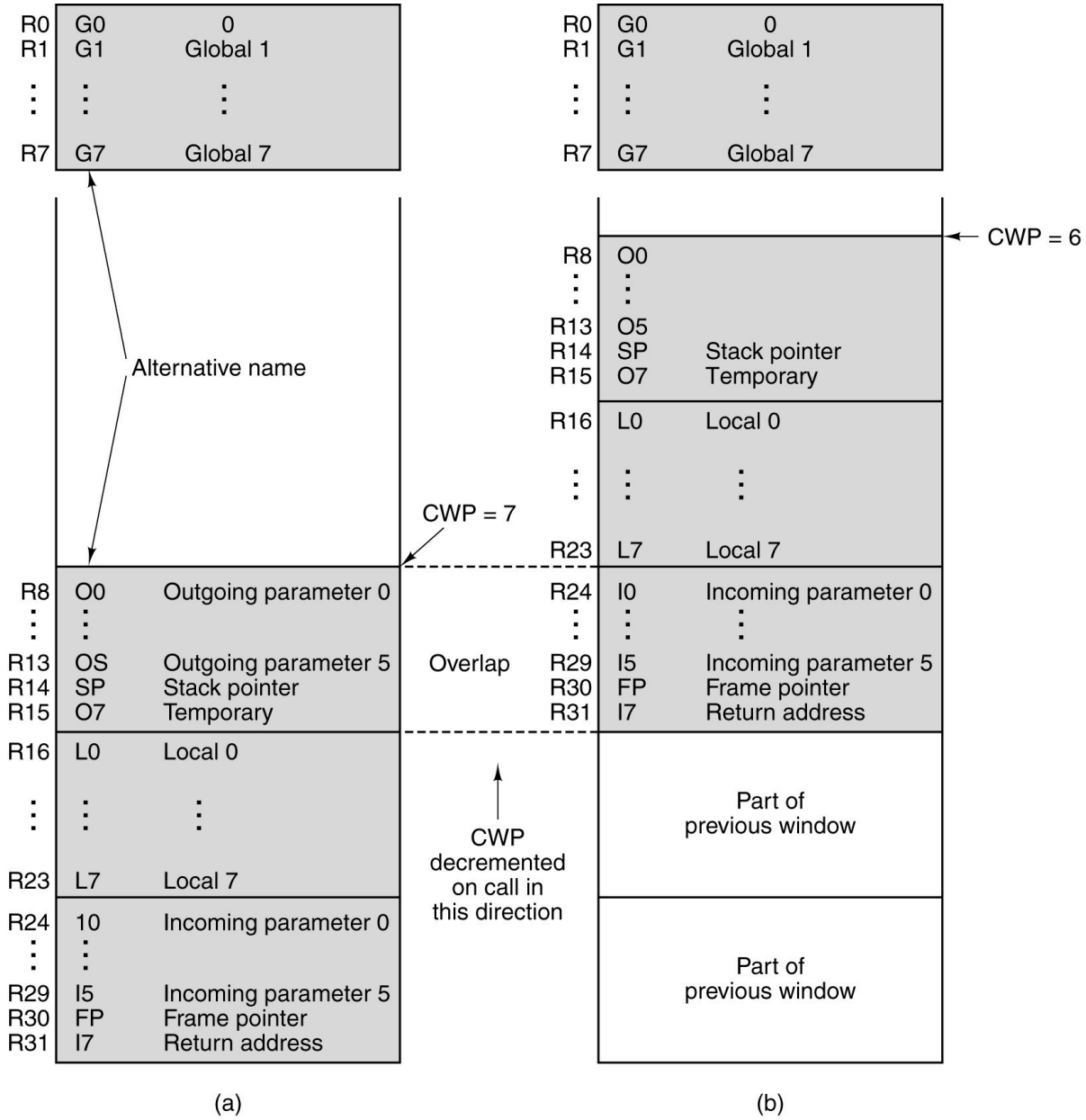
Register der Ultra-Sparc III



Compilernutzung:

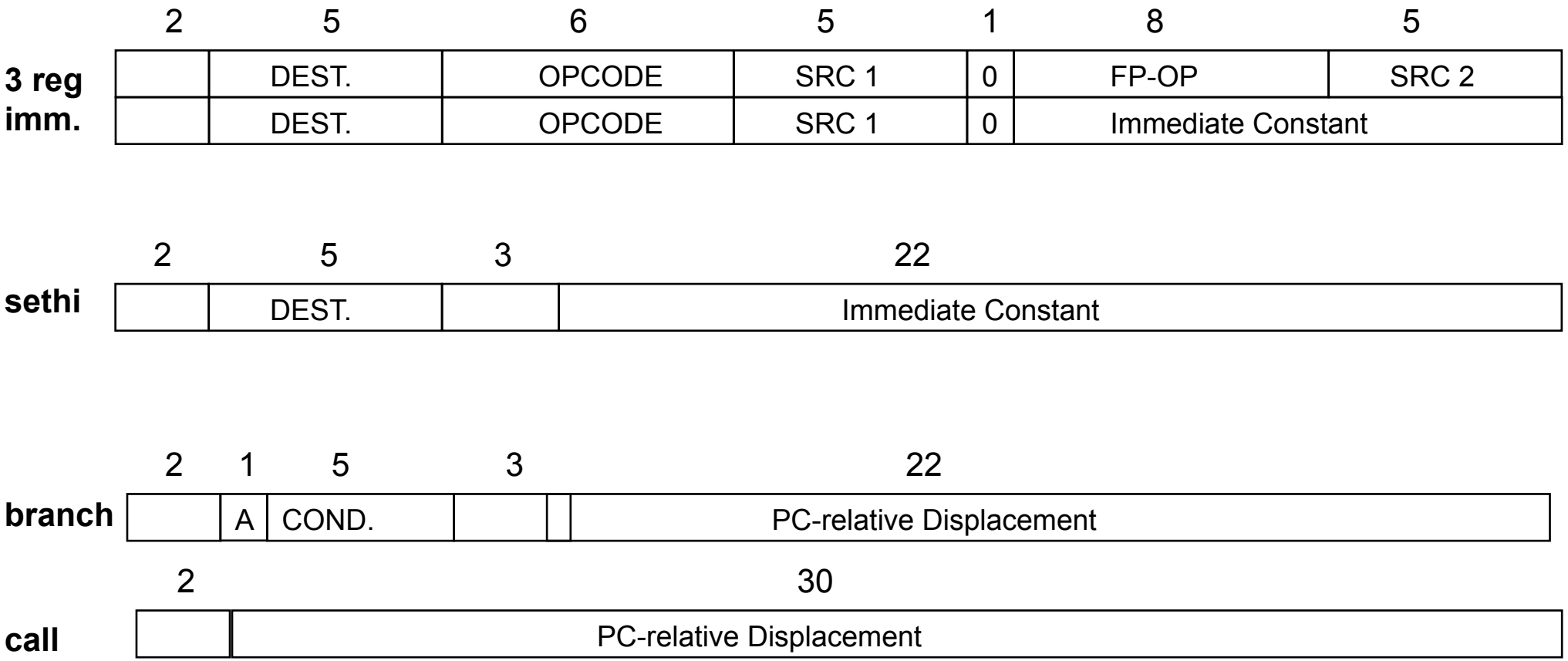
Register	Alt. Bez.	Funktion
R0	G0	Konstante "0"
R1-R7	G1-G7	Globale Variablen
R8-13	00-05	Params. f. aufruf. Prozedur
R14	SP	Stackpointer
R15	07	Temp. Register
R16-23	L0-L7	Lokale Var. f. aktuelle Proz.
R24-29	I0-I5	Eingabe-Parameter
R30	FP	Framepointer
R31	I7	Return Adresse

SPARC
Register Windows



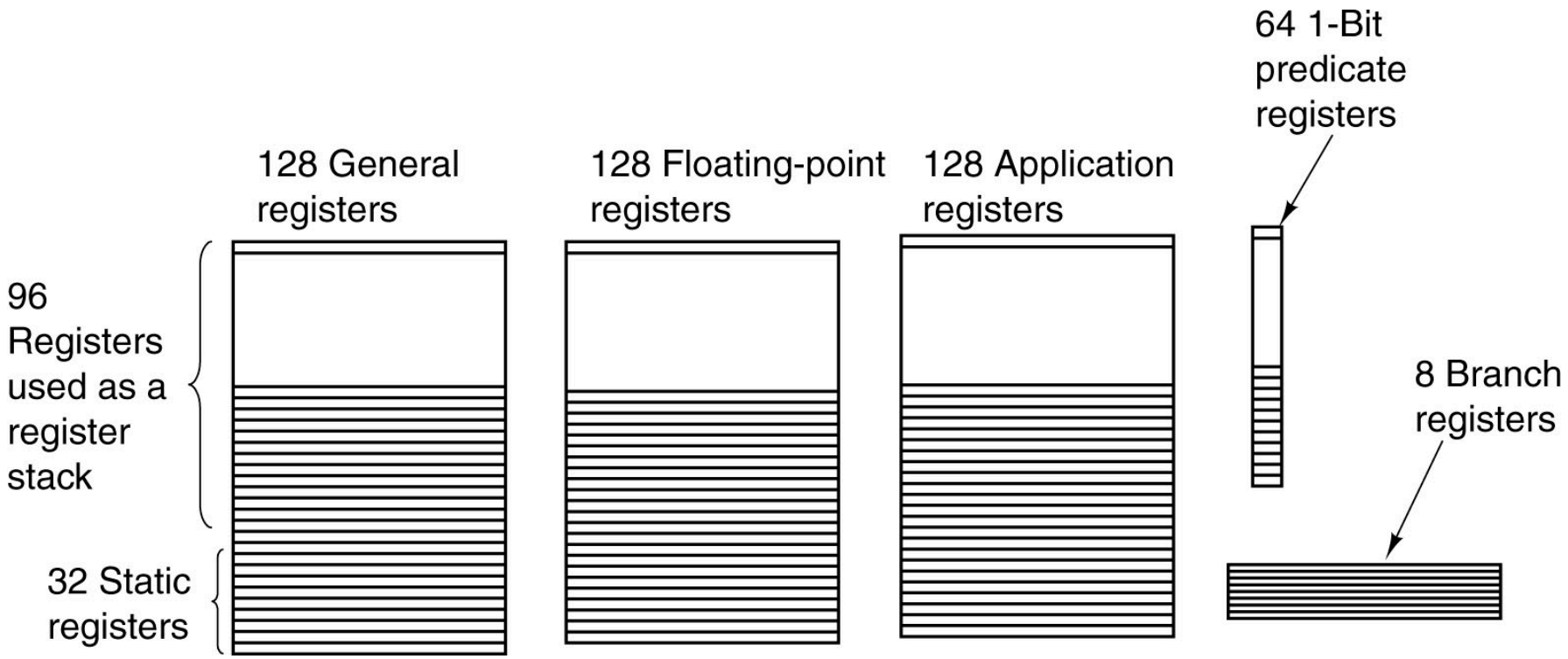
A. Tanenbaum, Computerarchitektur, Pearson Studium, 2001

SPARC Befehlsformate und Adressierungsarten



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

IA-64, Itanium und EPIC



Die Itanium 2 Register

Addressing mode	Pentium 4	UltraSPARC III	MC6809
Accumulator			×
Immediate	×	×	×
Direct	×		×
Register	×	×	×
Register indirect	×	×	×
Indexed	×	×	**x**
Based-indexed		×	**x**
Stack			

fester Offset

Register Offset

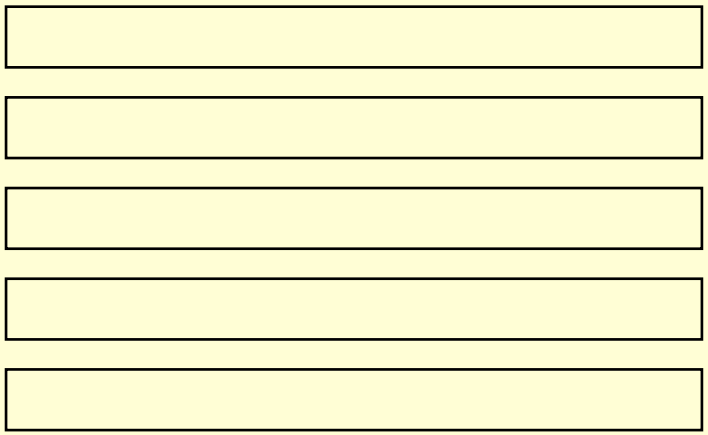
Die Java Virtual Machine (JVM)

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

IJVM: eingeschränkte Architektur, A. Tanenbaum

Die I-"Java Virtual Machine"

Programmiermodell der IJVM



SP (Stack Pointer)

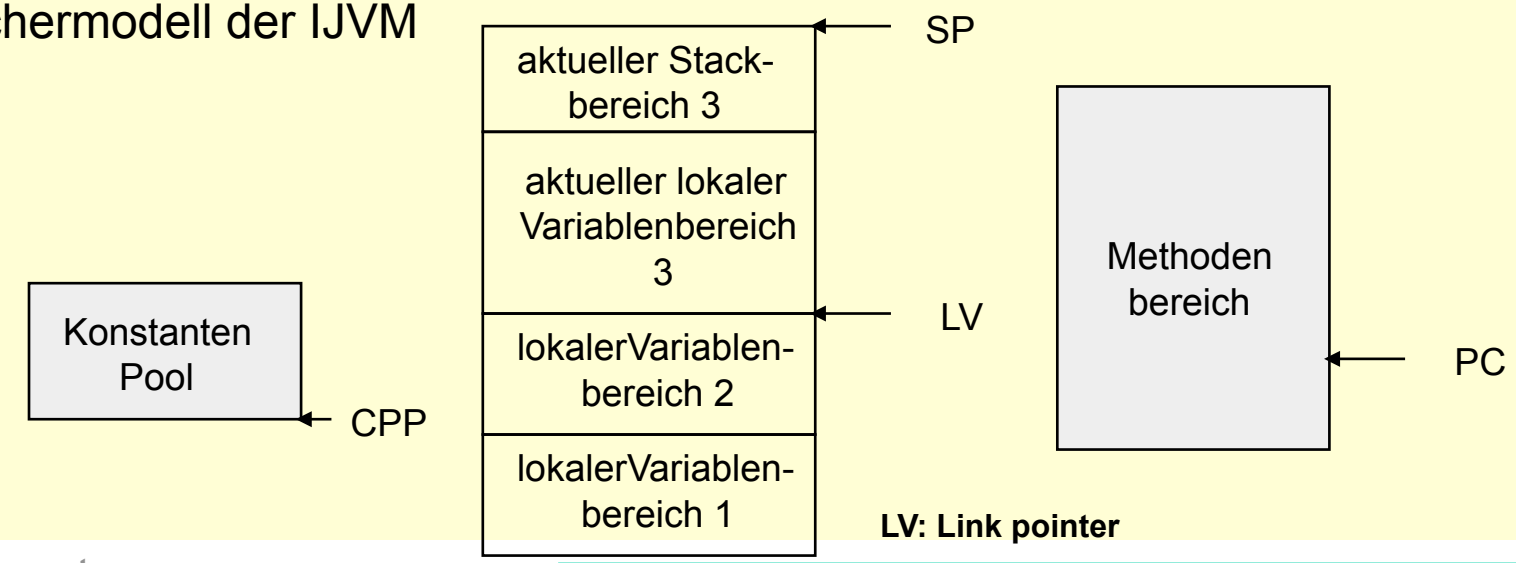
FP (Frame Pointer)

PC (Program Conter)

TOS (Top of Stack)

CPP (Constants Pool Pointer)

Speichermodell der IJVM



IJVM 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

Auswertung arithmetischer Ausdrücke in einer Stack-Architektur

Infix	Reverse Polish notation
A + B × C	A B C × +
A × B + C	A B × C +
A × B + C × D	A B × C D × +
(A + B) / (C – D)	A B + C D – /
A × B / C	A B × C /
((A + B) × C + D)/(E + F + G)	A B + C × D + E F + G + /

Infix Notation und Umgekehrte Polnische (Postfix) Notation

Auswertung arithmetischer Ausdrücke

Step	Remaining string	Instruction	Stack
1	8 2 5 × + 1 3 2 × + 4 – /	BIPUSH 8	8
2	2 5 × + 1 3 2 × + 4 – /	BIPUSH 2	8, 2
3	5 × + 1 3 2 × + 4 – /	BIPUSH 5	8, 2, 5
4	× + 1 3 2 × + 4 – /	IMUL	8, 10
5	+ 1 3 2 × + 4 – /	IADD	18
6	1 3 2 × + 4 – /	BIPUSH 1	18, 1
7	3 2 × + 4 – /	BIPUSH 3	18, 1, 3
8	2 × + 4 – /	BIPUSH 2	18, 1, 3, 2
9	× + 4 – /	IMUL	18, 1, 6
10	+ 4 – /	IADD	18, 7
11	4 – /	BIPUSH 4	18, 7, 4
12	– /	ISUB	18, 3
13	/	IDIV	6

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		Ox9F 0x00 OxOD
	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:			

Minimalistisch: Der Single Instruction Computer

- **Eine Instruktion "***substract and brach if negative" SBN***.**
- **Keine Register, nur Speicher-Speicher Operation.**
- **Absolute Adressierung.**
- **Drei Adressen: <***Operand 1/Ergebnis, Operand 2, (bedingte) Folgeadresse***>.**

Bedeutung der SBN-Instruktion:

M[a] = M[a] - M[b];
if (M[a] < 0) goto c else goto next instruction;

Programmbeispiele:

Lösche Speicherplatz M[a] und gehe zur nächsten Instruktion:

M[a], M[a], +1

Kopiere M[a] nach M[tmp] und gehe zur nächsten Instruktion:

M[tmp], M[tmp], +1	**tmp = 0**
M[tmp], M[a], +1	**tmp = -a**
M[tmp], 0, +1	

Addiere a+b = -(-a-b) und gehe zur nächsten Instruktion

M[a], 0, +1	**berechne -m**
M[a], M[b], +1	**berechne -m - n**
M[a], 0, +1	**berechne -(-a -b)**

Lernziele

- **Zielkonflikte und Alternativen**

  für Registersätze
  für Befehlssätze

- **Geschützte Operations-Modi**
- **Bus-System**
- **Erweiterungsmöglichkeiten durch Co-Prozessoren**

Instruktionssätze

The Pentium 4 Instructions (1)

Moves

MOV DST,SRC	Move SRC to DST
PUSH SRC	Push SRC onto the stack
POP DST	Pop a word from the stack to DST
XCHG DS1,DS2	Exchange DS1 and DS2
LEA DST,SRC	Load effective addr of SRC into DST
CMOVcc DST,SRC	Conditional move

Arithmetic

ADD DST,SRC	Add SRC to DST
SUB DST,SRC	Subtract SRC from DST
MUL SRC	Multiply EAX by SRC (unsigned)
IMUL SRC	Multiply EAX by SRC (signed)
DIV SRC	Divide EDX:EAX by SRC (unsigned)
IDIV SRC	Divide EDX:EAX by SRC (signed)
ADC DST,SRC	Add SRC to DST, then add carry bit
SBB DST,SRC	Subtract SRC & carry from DST
INC DST	Add 1 to DST
DEC DST	Subtract 1 from DST
NEG DST	Negate DST (subtract it from 0)

The Pentium 4 Instructions (2)

Binary coded decimal

DAA	Decimal adjust
DAS	Decimal adjust for subtraction
AAA	ASCII adjust for addition
AAS	ASCII adjust for subtraction
AAM	ASCII adjust for multiplication
AAD	ASCII adjust for division

Boolean

AND DST,SRC	Boolean AND SRC into DST
OR DST,SRC	Boolean OR SRC into DST
XOR DST,SRC	Boolean Exclusive OR SRC to DST
NOT DST	Replace DST with 1's complement

Shift/rotate

SAL/SAR DST,#	Shift DST left/right # bits
SHL/SHR DST,#	Logical shift DST left/right # bits
ROL/ROR DST,#	Rotate DST left/right # bits
RCL/RCR DST,#	Rotate DST through carry # bits

The Pentium 4 Instructions (3)

Test/compare

TEST SRC1,SRC2	Boolean AND operands, set flags
CMP SRC1,SRC2	Set flags based on SRC1 - SRC2

Transfer of control

JMP ADDR	Jump to ADDR
Jxx ADDR	Conditional jumps based on flags
CALL ADDR	Call procedure at ADDR
RET	Return from procedure
IRET	Return from interrupt
LOOPxx	Loop until condition met
INT n	Initiate a software interrupt
INTO	Interrupt if overflow bit is set

Strings

LODS	Load string
STOS	Store string
MOVS	Move string
CMPS	Compare two strings
SCAS	Scan Strings

The Pentium 4 Instructions (4)

Condition codes

STC	Set carry bit in EFLAGS register
CLC	Clear carry bit in EFLAGS register
CMC	Complement carry bit in EFLAGS
STD	Set direction bit in EFLAGS register
CLD	Clear direction bit in EFLAGS reg
STI	Set interrupt bit in EFLAGS register
CLI	Clear interrupt bit in EFLAGS reg
PUSHFD	Push EFLAGS register onto stack
POPFD	Pop EFLAGS register from stack
LAHF	Load AH from EFLAGS register
SAHF	Store AH in EFLAGS register

Miscellaneous

SWAP DST	Change endianness of DST
CWQ	Extend EAX to EDX:EAX for division
CWDE	Extend 16-bit number in AX to EAX
ENTER SIZE,LV	Create stack frame with SIZE bytes
LEAVE	Undo stack frame built by ENTER
NOP	No operation
HLT	Halt
IN AL,PORT	Input a byte from PORT to AL
OUT PORT,AL	Output a byte from AL to PORT
WAIT	Wait for an interrupt

SRC = source
DST = destination

# = shift/rotate count
LV = # locals

The UltraSPARC III Instructions (1)

Loads

LDSB ADDR,DST	Load signed byte (8 bits)
LDUB ADDR,DST	Load unsigned byte (8 bits)
LDSH ADDR,DST	Load signed halfword (16 bits)
LDUH ADDR,DST	Load unsigned halfword (16)
LDSW ADDR,DST	Load signed word (32 bits)
LDUW ADDR,DST	Load unsigned word (32 bits)
LDX ADDR,DST	Load extended (64-bits)

Stores

STB SRC,ADDR	Store byte (8 bits)
STH SRC,ADDR	Store halfword (16 bits)
STW SRC,ADDR	Store word (32 bits)
STX SRC,ADDR	Store extended (64 bits)

The UltraSPARC III Instructions (2)

Arithmetic

ADD R1,S2,DST	Add
ADDCC "	Add and set icc
ADDC "	Add with carry
ADDCCC "	Add with carry and set icc
SUB R1,S2,DST	Subtract
SUBCC "	Subtract and set icc
SUBC "	Subtract with carry
SUBCCC "	Subtract with carry and set icc
MULX R1,S2,DST	Multiply
SDIVX R1,S2,DST	Signed divide
UDIVX R1,S2,DST	Unsigned divide
TADCC R1,S2,DST	Tagged add

The UltraSPARC III Instructions (3)

Shifts/rotates

SLL R1,S2,DST	Shift left logical (32 bits)
SLLX R1,S2,DST	Shift left logical extended (64)
SRL R1,S2,DST	Shift right logical (32 bits)
SRLX R1,S2,DST	Shift right logical extended (64)
SRA R1,S2,DST	Shift right arithmetic (32 bits)
SRAX R1,S2,DST	Shift right arithmetic ext. (64)

Miscellaneous

SETHI CON,DST	Set bits 10 to 31
MOVcc CC,S2,DST	Move on condition
MOVr R1,S2,DST	Move on register
NOP	No operation
POPC S1,DST	Population count
RDCCR V,DST	Read condition code register
WRCCR R1,S2,V	Write condition code register
RDPC V,DST	Read program counter

The UltraSPARC III Instructions (4)

Boolean	
AND R1,S2,DST	Boolean AND
ANDCC "	Boolean AND and set icc
ANDN "	Boolean NAND
ANDNCC "	Boolean NAND and set icc
OR R1,S2,DST	Boolean OR
ORCC "	Boolean OR and set icc
ORN "	Boolean NOR
ORNCC "	Boolean NOR and set icc
XOR R1,S2,DST	Boolean XOR
XORCC "	Boolean XOR and set icc
XNOR "	Boolean EXCLUSIVE NOR
XNORCC "	Boolean EXCL. NOR and set icc

The UltraSPARC III Instructions (5)

Transfer of control

BPcc ADDR	Branch with prediction
BPr SRC,ADDR	Branch on register
CALL ADDR	Call procedure
RETURN ADDR	Return from procedure
JMPL ADDR,DST	Jump and link
SAVE R1,S2,DST	Advance register windows
RESTORE "	Restore register windows
Tcc CC,TRAP#	Trap on condition
PREFETCH FCN	Prefetch data from memory
LDSTUB ADDR,R	Atomic load/store
MEMBAR MASK	Memory barrier

SRC = source register
DST = destination register
R1 = source register
S2 = source: register or immediate
ADDR = memory address

TRAP# = trap number
FCN = function code
MASK = operation type
CON = constant
V = register designator

CC = condition code set
R =destination register
cc = condition
r = LZ,LEZ,Z,NZ,GZ,GEZ

Ladeinstruktionen

typeLOAD IND8	Schiebe lokale Variable auf den Stapel
typeALOAD	Schiebe Array-Element auf den Stapel
BALOAD	Schiebe Byte von einem Array auf Stapel
SALOAD	Schiebe short von einem Array auf Stapel
CALOAD	Schiebe char von einem Array auf Stapel
AALOAD	Schiebe den Zeiger von einem Array auf Stapel

Speicherinstruktionen

typeSTORE IND8	Wirf Wert aus und speichere ihn i. d. lok. Var.
typeASTORE	Wirf Wert aus und speichere ihn im Array
BASTORE	Wirf Byte aus und speichere es im Array
SASTORE	Wirf short aus und speichere es im Array
CASTORE	Wirf char aus und speichere es im Array
AASTORE	Wirf Zeiger aus und speichere ihn im Array

Schiebeinstruktionen

BIPUSH CON8	Schiebe eine kl. Konstante auf den Stapel
SIPUSH CON16	Schiebe 16-Bit-Konstante auf den Stapel
LDC IND8	Schiebe Konstante vom Konstanten-Pool
typeCONST_#	Schiebe unmittelbare Konstante
ACONST_NULL	Schiebe einen Nullzeiger auf den Stapel

Boolesche/Verschieben

iIAND	Boolesches AND
iIOR	Boolesches OR
iIXOR	Boolesches EXCLUSIVE OR
iISHL	Verschiebe nach links
iISHR	Verschiebe nach rechts
iIUSHR	Verschiebe nach rechts vorzeichenlos

JVM Befehle 1

JVM Befehle 2

Arithmetik

typeADD	Addiere
typeSUB	Subtrahiere
typeMUL	Multipliziere
typeDIV	Dividiere
typeREM	Rest
typeNEG	Negiere

Umwandlung

x2y	Konvertiere x in y
i2c	Konvertiere Ganzzahl in Zeichen (char)
i2b	Konvertiere Ganzzahl in Byte

Stapelverwaltung

DUPxx	Sechs Instruktionen zum Duping
POP	Wirf eine Ganzzahl v. Stapel u. verwerfe sie
POP2	Wirf zwei Ganzzahlen v. Stapel u. verwerfe sie
SWAP	Vertausche d. oberen beiden Ganzzahlen a. d. St.

Vergleich

IF_ICMPrel OFFSET16	Bedingter Sprung
IF_ACMPEQ OFFSET16	Verzweige, w. 2 Zeiger gleich sind
IF_ACMPNE OFFSET16	Verzweige, w. 2 Zeiger ungleich sind
IFrel OFFSET16	Prüfe 1 Wert und verzweige
IFNULL OFFSET16	Verzweige, wenn Zeiger Null ist
IFNONNULL OFFSET16	Verzweige, wenn Zeiger n. Null ist
LCMP	Vergleiche zwei longs
FCMPL	Vergleiche 2 floats auf <
FCMPG	Vergleiche 2 floats auf >
DCMPL	Vergleiche doubles auf <
DCMPG	Vergleiche doubles auf >

Steuerungsübergabe

INVOKEVIRTUAL IND16	Methodenaufruf
INVOKESTATIC IND16	Methodenaufruf
INVOKEINTERFACE ...	Methodenaufruf
INVOKESPECIAL IND16	Methodenaufruf
JSR OFFSET16	Rufe letzte Klausel auf
typeRETURN	Gib Wert zurück
ARETURN	Gib Zeiger zurück
RETURN	Gib void zurück
RET IND8	Kehre von letzter zurück
GOTO OFFSET16	Unbedingter Sprung

JVM Befehle 3

Arrays

ANEWARRAY IND16	Erstelle Zeiger-Array
NEWARRAY ATYPE	Erstelle a-type-Array
MULTINEWARRAY IN16,D	Erstelle multidim-Array
ARRAYLENGTH	Hole Array-Länge

Verschiedenes

IINC IND8,CON8	Inkrementiere lokale Variable
WIDE	Wide-Präfix
NOP	Keine Operation
GETFIELD IND16	Lies Feld von Objekt
PUTFIELD IND16	Schreibe Feld in Objekt
GETSTATIC IND16	Hole statisches Feld von Klasse
NEW IND16	Erstelle ein neues Objekt
INSTANCEOF OFFSET16	Ermittle Objekttyp
CHECKCAST IND16	Prüfe Objekttyp
ATHROW	Wirf Ausnahme aus
LOOKUPSWITCH ...	Verdünne mehrwegigen Sprung
TABLESWITCH ...	Verdichte mehrwegigen Sprung
MONITORENTER	Gib eine Überwachung ein
MONITOREXIT	Verlasse eine Überwachung

IND8/16 = Index von lokalen Variablen
CON8/16, D, ATYPE = Konstante

type, x, y = I, L, F, D
OFFSET16 für Sprung