Rechnerarchitektur I (RAI)

Befehlssatz

Prof. Dr. Akash Kumar

Chair for Processor Design

Inhalt

- Begriffserklärung
- Klassifikationen von Befehlssatz-Architekturen
- Operandenspeicherung innerhalb der CPU
- Explizit im Befehl adressierte Operanden
- Operandenspeicherung, Adressierung
- Operationen des Befehlssatzes
- Typ und Länge der Operanden
- Beispiele zu Befehlssatz-Architekturen

Inhalt

- JVM (STACK)
- R2000 (MIPS)
- ALPHA AXP
- Intel Pentium II
- UltraSPARC II
- AT91 ARM
- AMD Athlon

Begriffserklärung

Algorithmus: Die Informationsverarbeitung in einem Rechner erfordert die schrittweise Umsetzung eines definierten Algorithmus. Der Algorithmus wird als geordnete Folge von Anweisungen, Befehlen dargestellt.

Befehl: Ein Befehl (instruction) ist eine eindeutig spezifizierte Arbeitsanweisung an den Prozessor (CPU). Er ordnet eine Operation an, die in der Regel an spezifizierten Daten (Operanden) vorzunehmen ist und ein Ergebnis (Resultat) liefert. (maschinenlesbar → Maschinenbefehl)

Befehlssatz: Der Menge aller in einem Prozessor implementierten Befehle bildet den Befehlssatz. Die Architektur eines Rechners wird wesentlich durch den Befehlssatz des verwendeten Prozessors bestimmt (ISC-Instruction Set Computer → wesentliches Architekturmerkmal).
Die Menge aller Maschinenbefehle definiert die Maschinensprache.

Begriffserklärung

Befehlskomponenten: Die Komponenten eines Befehlssatzes sind die Operation, der Datentyp und die Operanden, Adressierung des Befehls.

Maschinensprache: Durch die Menge der im Prozessor realisierbaren Maschinenbefehle ist eine hardwareabhängige Programmiersprache für den Rechner, die Maschinensprache gegeben → Maschinenprogramme. Rechner einer Prozessorfamilie realisieren eine weitestgehend ähnliche Maschinensprache (Binärkompatibilität).

Hauptmerkmale des Befehlssatzes

Befehlsvorrat: Die Menge aller verfügbaren Maschinenbefehle bildet den Befehlsvorrat. Sie werden in der Befehlsliste geordnet zusammengefaßt.

Befehlsformat: Die innere Struktur der Maschinenbefehle, dargestellt durch Binärworte, wird durch das Befehlsformat bestimmt. Entsprechend dem Befehlsformat werden die einzelnen Komponenten des Befehls binär codiert im Befehlswort zusammengefaßt.

Begriffserklärung

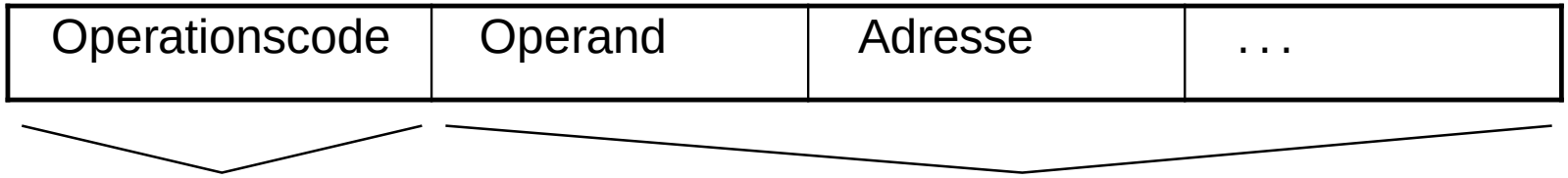
Orthogonalität: Ein Befehlssatz heißt orthogonal, wenn eine möglichst kleine Anzahl von grundlegenden Befehlen existiert, die beliebig miteinander kombinierbar sind und sich in ihrer Funktionalität nicht oder nur wenig überschneiden. Die Komponenten eines Befehlssatzes (Operation, Datentyp, Adressierung) sollten orthogonal zueinander sein (voneinander unabhängig). Jede Operation sollte jede relevante Adressierungsart bzw. jeden relevanten Datentyp zulassen.

Symmetrie: Ein Befehlssatz heißt symmetrisch, wenn jeder Befehl mit jedem relevanten Datentyp ausgeführt werden kann, sowie jede zulässige Adressierungsart benutzt werden kann.

Regularität: Ein Befehlssatz heißt regulär, wenn er nach konsistent anwendbaren Regeln strukturiert ist. Regularität und Orthogonalität stehen in enger Wechselbeziehung.

Befehlsstruktur

Die Befehlskomponenten (Operation, Datentyp und Operanden, Adressierung) werden im Befehlswort strukturiert zusammengefaßt und binär codiert. Die Befehlswortlänge ist allgemein nicht für jeden Befehl des Befehlssatzes einheitlich (byteweise abgestuft).



Die Positionen von Opcode und Adressteil sind nicht fest (der Opcode kann am Anfang oder Ende des Befehlswortes stehen aber auch über das Befehlswort verteilt sein).

Der Operandenteil kann je nach Befehlsformat gleichzeitig auch mehrere Operanden und oder Adressen, wie auch andere Informationen enthalten.

CISC/RISC

Je nach Umfang des im Prozessor realisierten Befehlssatzes können zwei Kategorien von Befehlssatz-Architekturen (ISA) unterschieden werden:

1. **CISC** (Complex Instruction Set Computer)
   Befehlsvorrat: 400..500 Befehle/Befehlsformate
   z.B.: DEC VAX, IBM 360, Intel x86

2. **RISC** (Reduced Instruction Set Computer)
   Befehlsvorrat: 40..50 Befehle/Befehlsformate
   z.B.: Sun SPARC, SGI MIPS, DEC ALPHA, HP PARISC, IBM PowerPC

Diese Unterscheidung beruht nicht primär auf Unterschieden in der Hardware-Realisierung, dem eingesetzten Betriebssystem, . . .

Motivation für RISC

90/10 Regel beim Befehlssatz

Bei einem komplexen Befehlssatz (CISC) werden 90% aller Operationen mit nur 10% der Befehle des Befehlssatzes durchgeführt.

R. Chou und M. Horowitz: „The goal of any instruction format should be: (1) simple decode, (2) simple decode, (3) simple decode.“

Albert Einstein: „Keep it simple, as simple as possible, but no simpler.“

CISC

- Befehlsworte und Opcode in komplexen Befehlsformaten mit variablen Längen und vielen komfortablen Adressierungsarten
- leistungsfähige, komplexe Befehle führen zu einer Verkürzung des Maschinenprogramms und damit zur Erhöhung der Codedichte
- Realisierung der Maschinenbefehle durch Ausführen von Mikroprogrammen im Prozessor (der Befehlszyklus wird durch eine Mikroprogramm-Steuerung realisiert)
- Anzahl der benötigten Taktzyklen pro Befehl ist unterschiedlich (mehr als 1 Taktzyklus/Befehl)

RISC

- stark reduzierter Umfang an Befehlsformaten und Adressierungsarten (meist weniger als 4 Befehlsformate und 4 Adressierungsarten)
- einfache wenige Basisbefehle, aus denen komplexe Operationen zusammengestellt werden können
- Load/Store-Architektur, ALU-Befehle realisieren keine Speicherzugriffe, Speicherzugriffe erfolgen nur über Load/Store-Befehle
- Universalregister-Architektur (meist 32 oder mehr Universalregister)
- festverdrahtete Maschinenbefehle und fester Befehlszyklus, keine Mikroprogramm-Steuerung
- Ausführung der meisten Befehle in nur einem Taktzyklus

Klassifikationen von Befehlssatz-Architekturen

- Merkmale der Klassifikation
  - Operandenspeicherung innerhalb der CPU, wo und wie
  - Zahl der explizit im Befehl adressierten Operanden
  - Operandenspeicherung, Adressierung, wie spezifiziert
  - Operationen des Befehlssatzes
  - Typ und Lange der Operanden, wie spezifiziert
- Alle bekannten Rechnerarchitekturen stellen temporären Operandenspeicher innerhalb der CPU bereit (Register, Stack, Akkumulator).

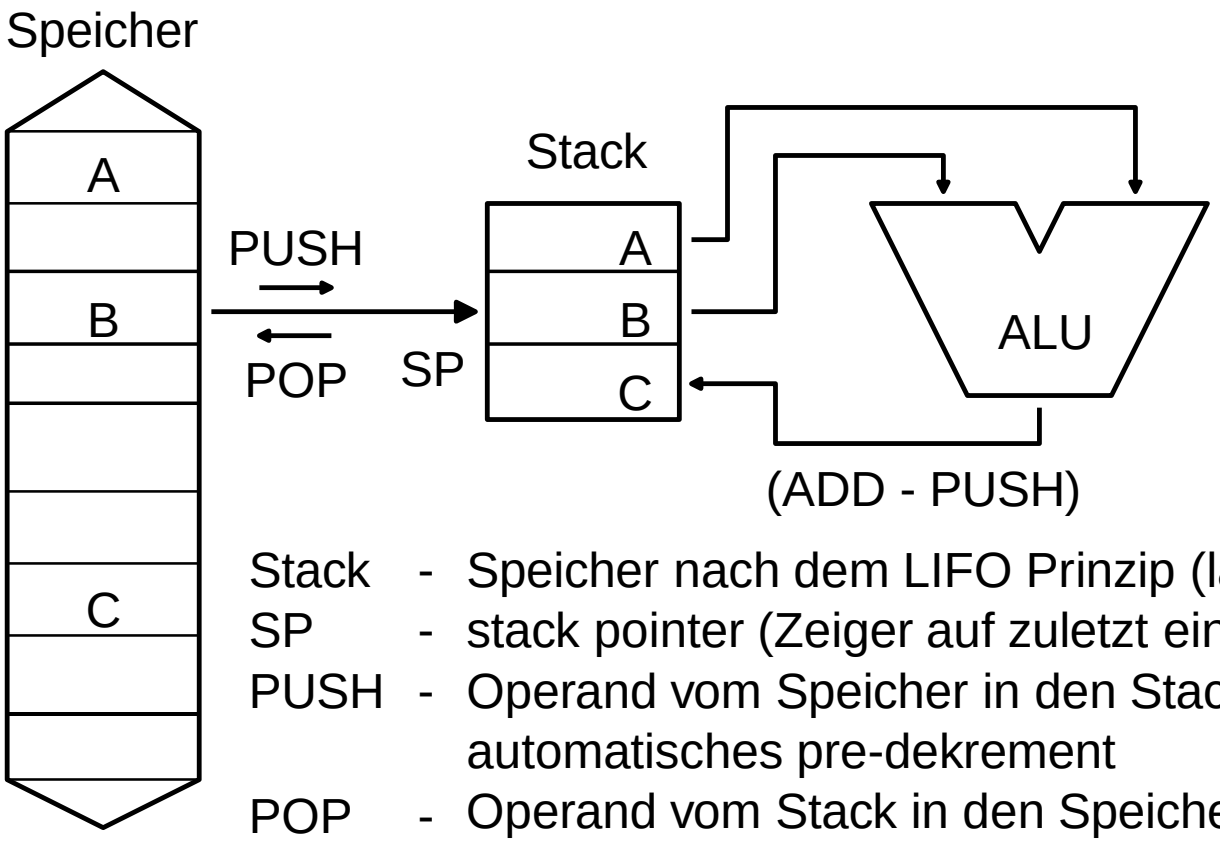
Operandenspeicherung innerhalb der CPU

Hauptvarianten, Alternativen der Operandenspeicherung

temporärer Operandenspeicher	explizite Operanden	Quelle für Operanden	Ziel für Resultate	Zugriff auf Operanden
Stack	0	Stack	Stack	PUSH/POP auf Stack
Akkumulator (ACCU)	1	ACCU/ Speicher	ACCU	LOAD/STORE auf ACCU
Registersatz (Universalregister)	2 oder 3	Register/ Speicher	Register/ Speicher	LOAD/STORE auf Register

Stack-Architektur (stack architecture)

Null-Adress-Maschine



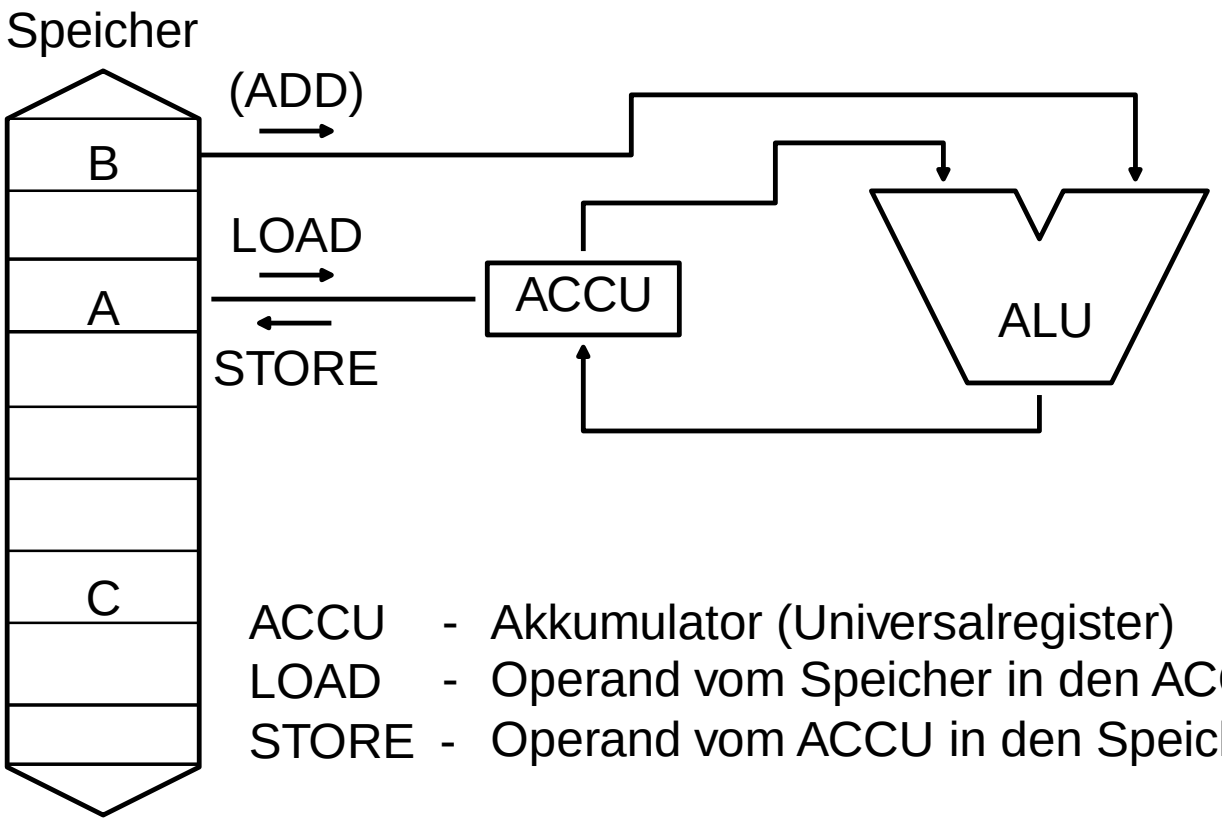
$C := A + B$

PUSH	A
PUSH	B
ADD	
POP	C

Stack - Speicher nach dem LIFO Prinzip (last in - first out)
SP - stack pointer (Zeiger auf zuletzt eingetragene Daten)
PUSH - Operand vom Speicher in den Stack transportieren
automatisches pre-dekrement
POP - Operand vom Stack in den Speicher transportieren
automatisches post-inkrement

Akkumulator-Architektur (accumulator architecture)

Ein-Adress-Maschine



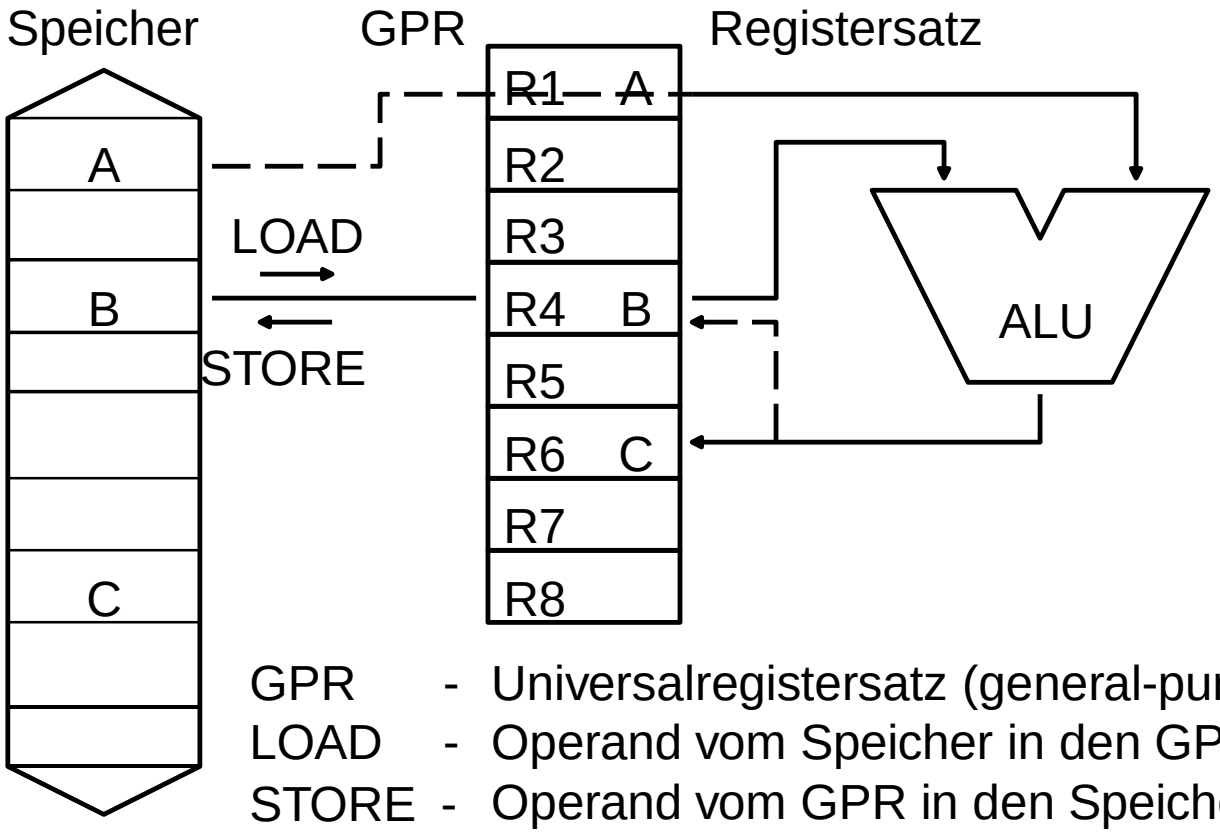
$C := A + B$

LOAD	A
ADD	B
STORE	C

ACCU - Akkumulator (Universalregister)
LOAD - Operand vom Speicher in den ACCU transportieren
STORE - Operand vom ACCU in den Speicher transportieren

Universalregistersatz-Architektur (GPR architecture)

Zwei/Drei-Adress-Maschine



$C := A + B$ (3 Adr)

```
LOAD   R1, A
LOAD   R4, B
ADD    R6, R2, R1
STORE  C, R6
```

$C := A + B$ (2 Adr)

```
LOAD   R4, B
ADD    R4, A
STORE  C, R4
```

GPR - Universalregistersatz (general-purpose register file)
LOAD - Operand vom Speicher in den GPR transportieren
STORE - Operand vom GPR in den Speicher transportieren

Vor- und Nachteile der Architekturtypen

Vor- und Nachteile der Architekturtypen

Typ	Vorteile	Nachteile
Stack	einfachstes Modell, gute Codedichte	kein direkter Zugriff auf Stack, nur relativ zum SP → Engpass
ACCU	kurze Befehle minimale Hardware	ACCU ist einziger temporärer Speicher, höchster Speicherverkehr → Engpass
GRP	allgemeinstes Modell, Zwischenspeicherung der Operanden	alle Operanden explizit adressieren, lange, komplexe Befehlswörter, schlechte Codedichte

Motivation für Universalregister-Architekturen

1. Register erlauben einen schnelleren Zugriff auf die Operanden
2. einfachere Adressierung der Register (kurze Adresslängen)
3. Einbeziehung der Spezialregister in den Universalregistersatz
4. nutzbar als zusätzliche Ebene in der Speicherhierarchie
5. vielfältige Möglichkeiten der Zwischenspeicherung von Operanden
6. für Compiler einfacher und effektiver nutzbar (z.B. Variablenübergabe)

Explizit im Befehl adressierte Operanden

Grundsätzlich werden unterschieden:

monadische Operation: (unäre, einstellige Operation) $B := op\ A$

dyadische Operation: (binäre, zweistellige Operation) $C := A\ op\ B$

Für eine zweistellige Operation (Verknüpfung « op» von zwei Operanden zu einem Resultat) sind mindestens folgende Angaben erforderlich:

- Art der Operation $\rightarrow$ op
- Adresse des 1. Operanden (1. Quelloperand) $\rightarrow$ A
- Adresse des 2. Operanden (2. Quelloperand) $\rightarrow$ B
- Adresse für das Resultat (Zieladresse) $\rightarrow$ C

Explizit im Befehl adressierte Operanden

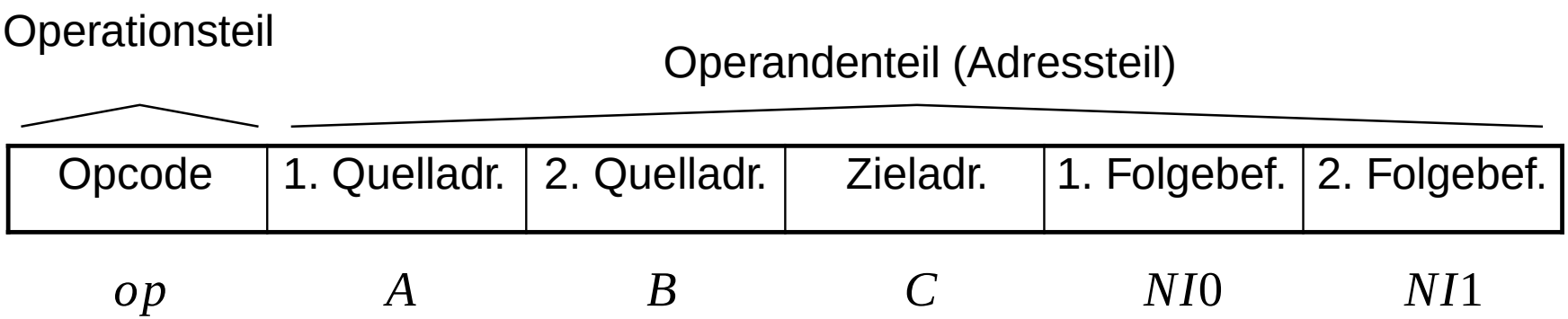
Folgende zusätzliche Angaben sind bei vollständiger Beschreibung eines Befehls mit Programmverzweigungen noch denkbar:

- Adresse des 1. Folgebefehls ohne Verzweigung $\rightarrow$ $NI0$
- Adresse des 2. Folgebefehls bei Verzweigung $\rightarrow$ $NI1$

Operanden und Resultate (Adressen und Befehle) stehen entweder in prozessorinternen temporären Speichern, z.B. Registern oder im Hauptspeicher. Beide werden durch Adressen angesprochen.

Die Codierung aller Angaben in einem Befehlswort führt zu einen 5-Adress-Befehl. Der Operandenteil umfaßt dabei die 1. und 2. Quelladresse der Operanden, die Zieladresse für das Resultat und die beiden Adressen der möglichen Folgebefehle.

5-Adress-Befehlsformat



Beispiel

Opcode	:	8 Bit	$\rightarrow$	256 verschiedene Befehle codierbar
Adressraum	:	32 Bit	$\rightarrow$	4 GByte Speicher adressierbar
Befehlswortlänge	:	168 Bit	$\rightarrow$	6 32-Bit-Worte pro Befehl

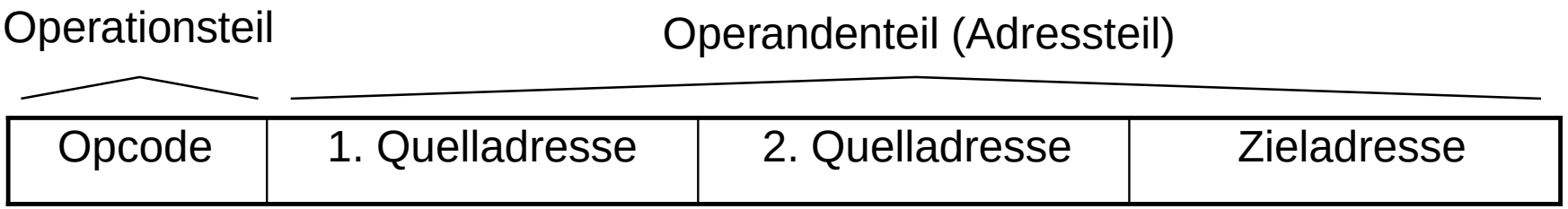
Übliche Befehlswortlängen z.B. 32 . . . 64 Bit ($\rightarrow$ 168 Bit sehr uneffektiv)

$\rightarrow$ Reduzierung des Operanden-/Adressteils (Erhöhung der Codedichte)

Massnahmen zur Reduzierung des Operandenteils

- Befehlszähler: enthält die Adresse für den unmittelbaren Folgebefehl
- Verzweigungs-/Sprungbefehle: enthalten Adressen der Folgebefehle
- implizite Adressierung: Quell oder Zieladressen implizit im Operationsteil
- überdeckte Adressierung: gleichzeitige Nutzung als Quell-/Zieladresse
- Direktoperanden: Operanden werden direkt im Operandenteil codiert
- Registeradressen: gesonderter Adressraum, wesentlich kürzere Adressen
- mehrstufige Adressierung: Umrechnung der Quell- und Zieladressen

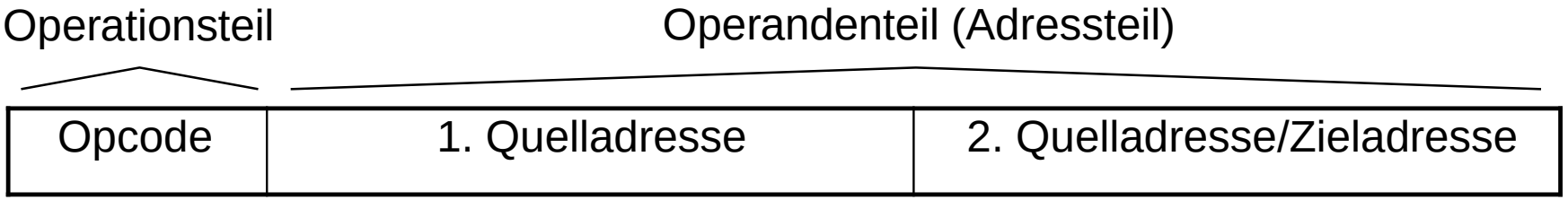
3-Adress-Befehlsformat



op A B C

$C := A\ op\ B$ alle Quell- und Zieladressen explizit adressiert

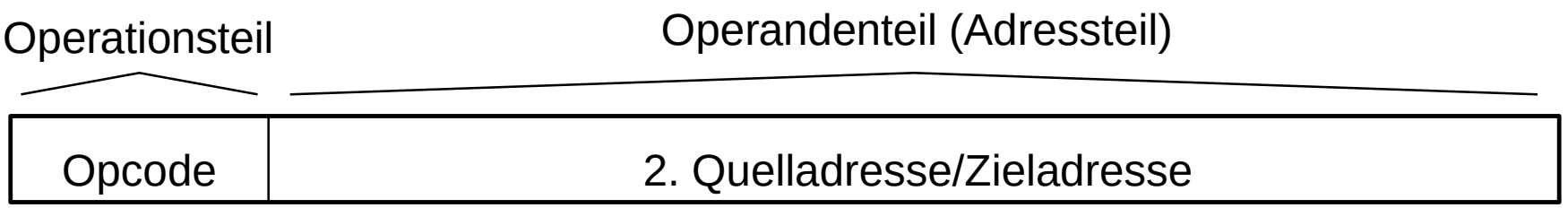
2-Adress-Befehlsformat



op A C

$C := C\ op\ A$ C - überdeckt adressiert

1-Adress-Befehlsformat



op

$ACCU := ACCU\ op\ C$

C

ACCU - implizit, C-überdeckt adressiert

0-Adress-Befehlsformat

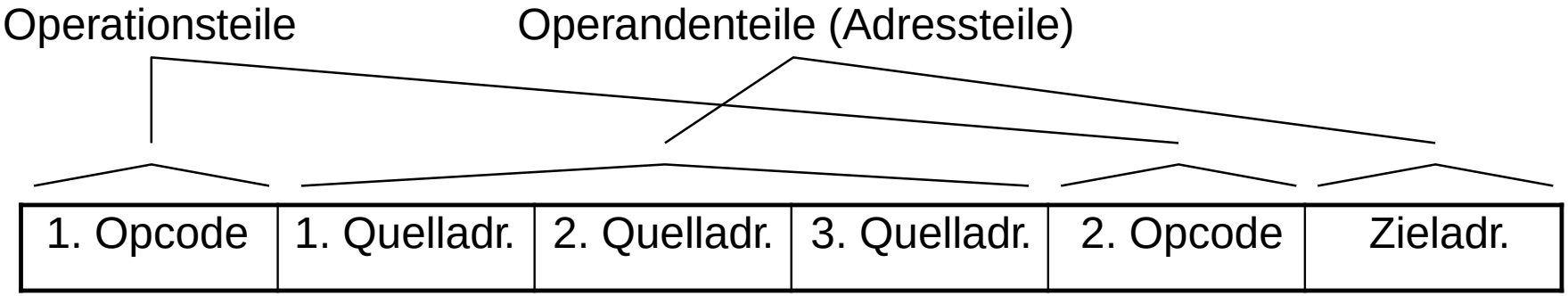
Opcode

kein Operandenteil (Adressteil)

op

alle Quell- und Zieladresssen implizit adressiert

VLIW-Befehlsformat



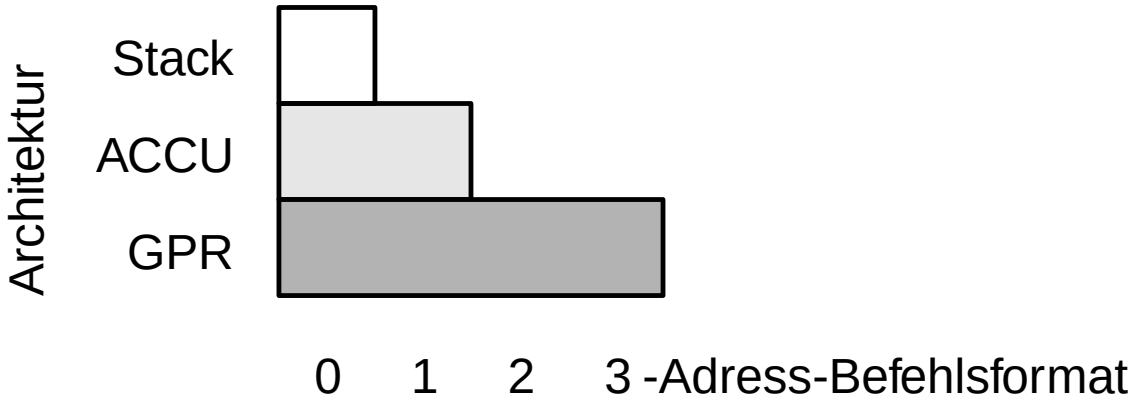
$$D := D + ((A\ op1\ B)\ op2\ (A\ op1\ C))$$

Aus mehreren Einzelbefehlen zusammengesetzter komplexer Befehl

kein festes Befehlsformat, an Operation und Architektur angepasst

speziell für die Nutzung von Parallelität, (z.B. DSP-Architekturen)

Zuordnung Befehlsformat - Architekturtyp



Die GPR-Architektur (3-Adress-Maschine) ist bzgl. der verschiedenen Befehlsformate am universellsten, alle Formate sind prinzipiell möglich.

Bei der Stack-Architektur (0-Adress-Maschine) ist die Universalität nicht direkt sichtbar, Organisation und Funktionalität des Stacks im Zusammenhang mit der ALU sind entscheidend.

Operandenspeicherung, Adressierung

Folgende Betrachtungen nur für Universalregister-Architekturen!

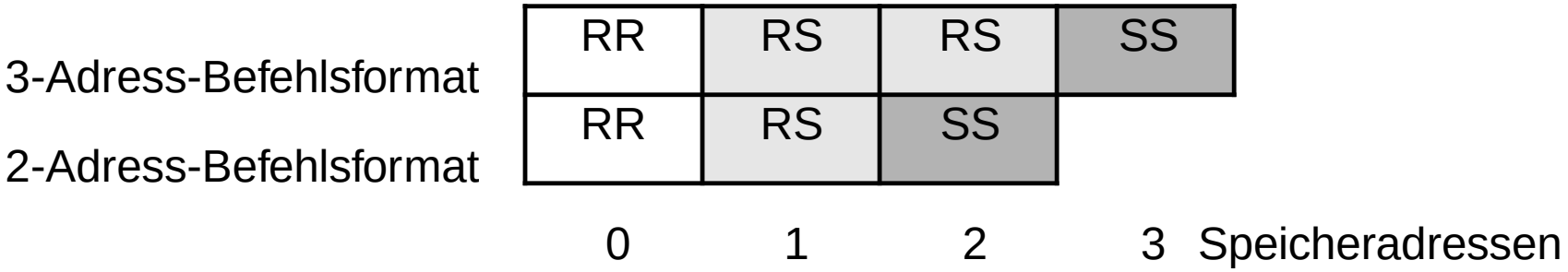
Wesentliche Befehlssatz-Charakteristiken bzgl. der Operanden:

1. Operandenzahl (für typischen ALU-Befehl)
   2-Adress-Befehlsformat (diadisch)
   3-Adress-Befehlsformat (triadisch)

2. Zahl der Speicheradressen (Speicheroperanden)
   0 . . . 3 Adressen für Speicherzugriffe

Die Adressen des Operandenteils, die nicht für einen Speicherzugriff genutzt werden, werden folglich entweder für Registeradressen oder auch für Direktoperanden verwendet.

Kombinationen von Register- und Speicheradressen



Klassifizierung der Kombinationen Register-Speicher (Typen von Universalregister-Architekturen):

RR	Register	-	Register	Load/Store-Architektur
RS	Register	-	Speicher	
SS	Speicher	-	Speicher	

Architekturen, die keinen Speicherzugriff für einen typischen ALU-Befehl erlauben, werden auch Load/Store-Architekturen genannt.

Vor- und Nachteile der einzelnen Typen

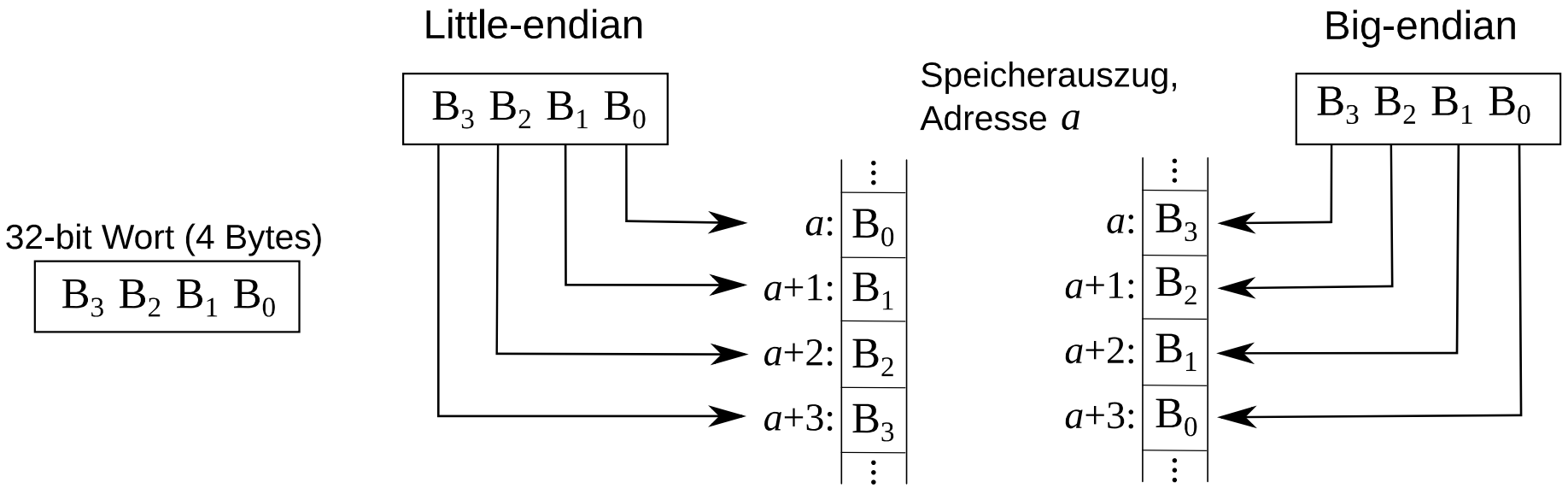
Typ	Vorteile	Nachteile
RR	einfache Befehlscodierung feste Befehlswortlänge einfache Codegenerierung einfache Taktsteuerung	höhere Befehlsanzahl gesonderte Load/Store-Befehle relativ schlechte Codedichte
RS	Datenzugriff ohne gesonderte Load/Store-Befehle gute Codedichte	Operanden nicht äquivalent
SS	kompakteste Form keine Register für Zwischenwerte	große Befehlslängenunterschiede viele Speicherzugriffe

RISC-Befehlssätze → Load/Store-Architekturen (nicht generell)

Speicheradressierung

Interpretation von Speicheradressen

Byte-orientierter Speicherzugriff (kleinste adressierbare Einheit)
→ Probleme bzgl. Byte-Reihenfolge bei Zugriff auf größere Einheiten
(Halbwort-, Wort-, Doppelwort-Zugriff, . . .)



Vereinbarungen zur Bytereihenfolge in einem Wort...

Bezeichnung	Bedeutung	Beispiel
little endian byte ordering	Byteadresse x...x00 liegt bei niedrigster Wertigkeit im Wort	Intel 80x86, DEC VAX
big endian byte ordering	Byteadresse x...x00 liegt bei höchster Wertigkeit im Wort	MIPS, SPARC

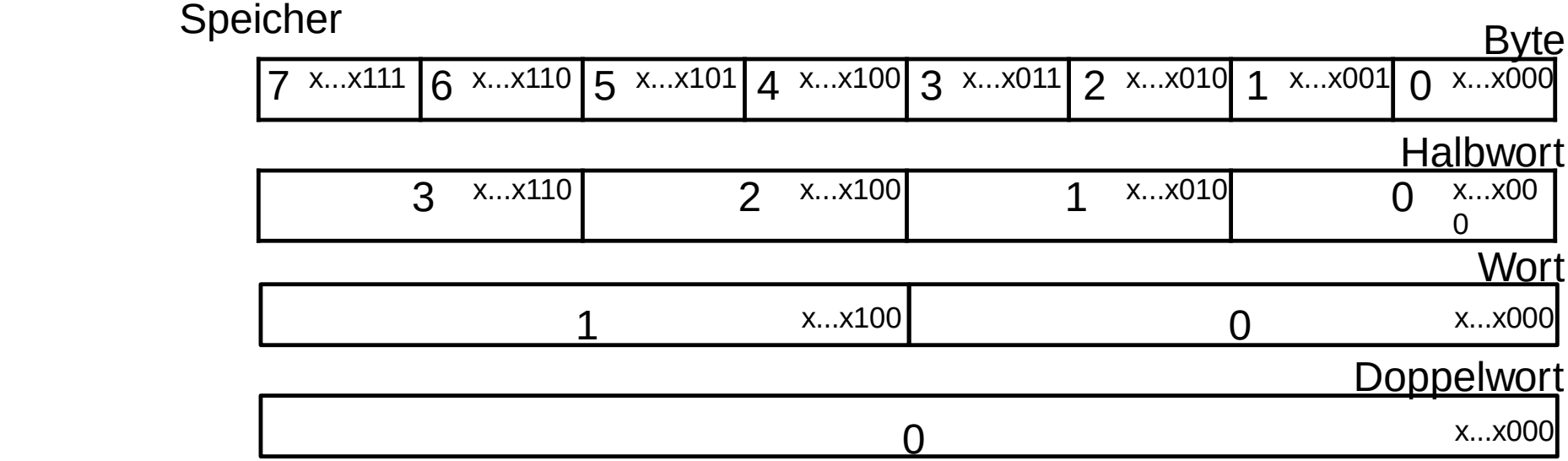
Für die Verarbeitung im Rechner ist die Byte-Reihenfolge meist unbedeutend. Probleme allgemein nur bei Byte-Wort-Umrechnungen.

Beim Datenaustausch zwischen Rechnern unterschiedlicher Byte-Reihenfolge ist diese unbedingt zu beachten (ggf. Konvertierungen).

Speicher-Alignment

Der Zugriff auf Einheiten die größer als ein Byte sind kann ausgerichtet (aligned) oder nicht ausgerichtet (misaligned) erfolgen. Ein Zugriff zu einer Einheit der Länge m Bytes ab der Byte-Adresse A ist ausgerichtet, wenn gilt: A **mod** $m = 0$.

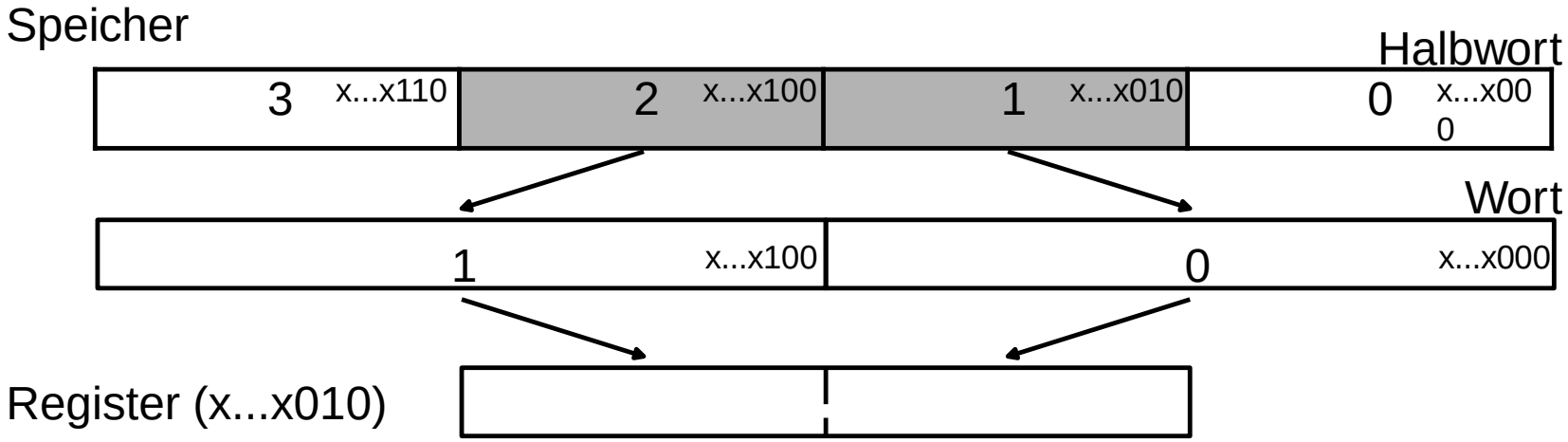
Ausgerichtete Speicherzugriffe bis Doppelwort-Grenze



Alignment-Restriktion

Soll z.B. auf ein Wort auf einer nicht ausgerichteten Halbwordgrenze (x...x010) zugegriffen werden, so müssen bei ausgerichtetem Wortzugriff die beiden benachbarten Worte (x...x000 und x...x100) eingelesen werden und anschließend die darin enthaltenen Halbworte (x...x010 und x...x100) wieder zu einem Wort zusammengesetzt (ausgerichtet) werden.

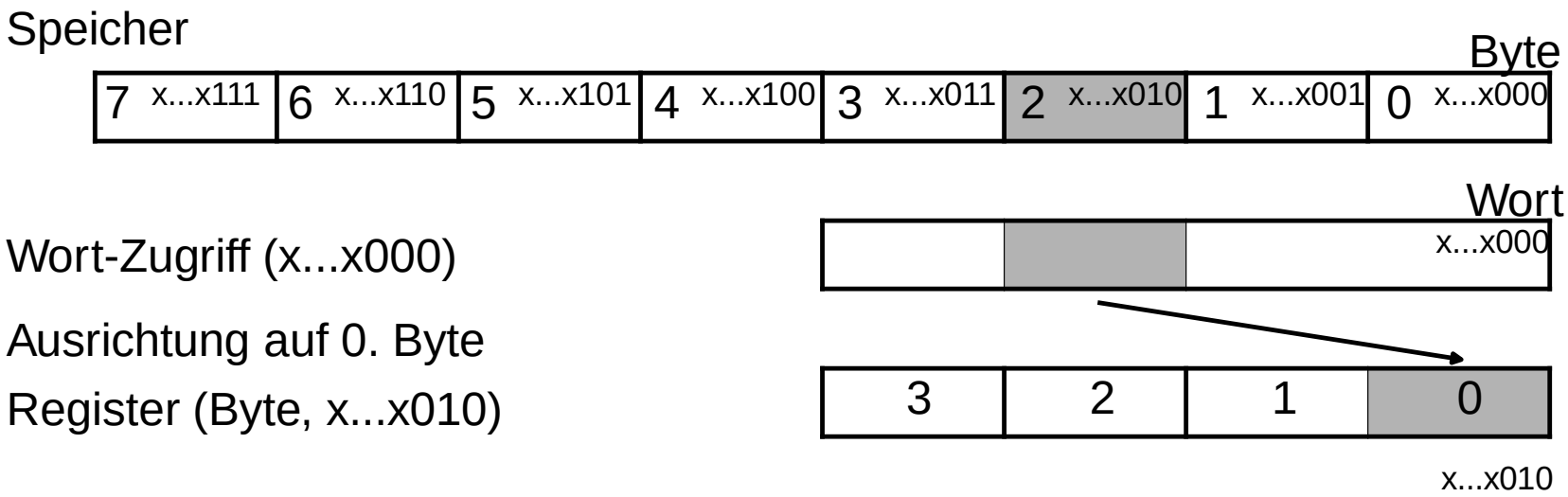
Ausgerichteter Wort-Speicherzugriff auf nichtausgerichtetes Wort



Alignment-Restriktion

Ein nicht ausgerichteter Speicherzugriff erfordert im allgemeinen mehrere ausgerichtete Zugriffe und eine anschließende Ausrichtung (Ausrichtungsnetzwerk). Ausgerichtete Zugriffe sind allgemein schneller. Ausgerichtete Zugriffe erfordern bei kürzeren Einheiten als die Registergröße ebenfalls ein Ausrichtungsnetzwerk (Multiplexer-Netzwerk, Barrel-shifter).

Ausgerichteter Byte-Speicherzugriff bei Wort-Zugriffsbreite



Adressierung

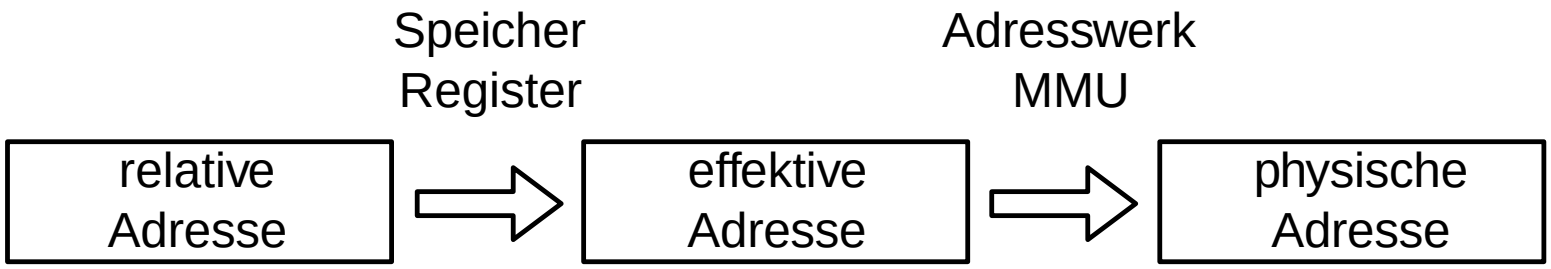
Operanden, Adressen und Befehle können im Hauptspeicher oder im Registerspeicher stehen. Sie werden über ihre Adressen angesprochen, die aus verschiedenen Komponeten zusammengesetzt sein können.

Adressen: Adressen beziehen sich auf Konstante (Direktoperand), Register (Registeradresse) oder den Hauptspeicher (Speicheradresse).

Statische, absolute Adressierung (physische Adressen): Die physischen Adressen werden fest, statisch zur Programmierzeit angegeben. Programme und Daten sind vollständig lageabhängig.

Dynamische, relative Adressierung (effektive Adressen): Die effektive Adresse wird erst zur Laufzeit durch eine Adressrechnung gewonnen. Im Adresswerk werden dann aus den effektiven Adressen die eigentlichen physischen Adressen für die Adressierung gebildet.

Relative Adressierung



Formale Notation der Adressrechnung

Hauptspeicheradresse	(Assembler)	:	A
Registeradresse	(Assembler)	:	RA
Direktoperand	(Assembler)	:	# Operand
Hauptspeicherinhalt von A	(Assembler)	:	(A) oder @(A) auch ((A))
Hexadezimalwert	(Assembler)	:	$ HEX
Hauptspeicherinhalt von A		:	M[A]
Registerinhalt von RA		:	RA

Adressierungsarten

Adressierungsarten sind alle Möglichkeiten eines Prozessors aus relativen Adressen effektive/physische Adressen zu berechnen (zur Laufzeit).

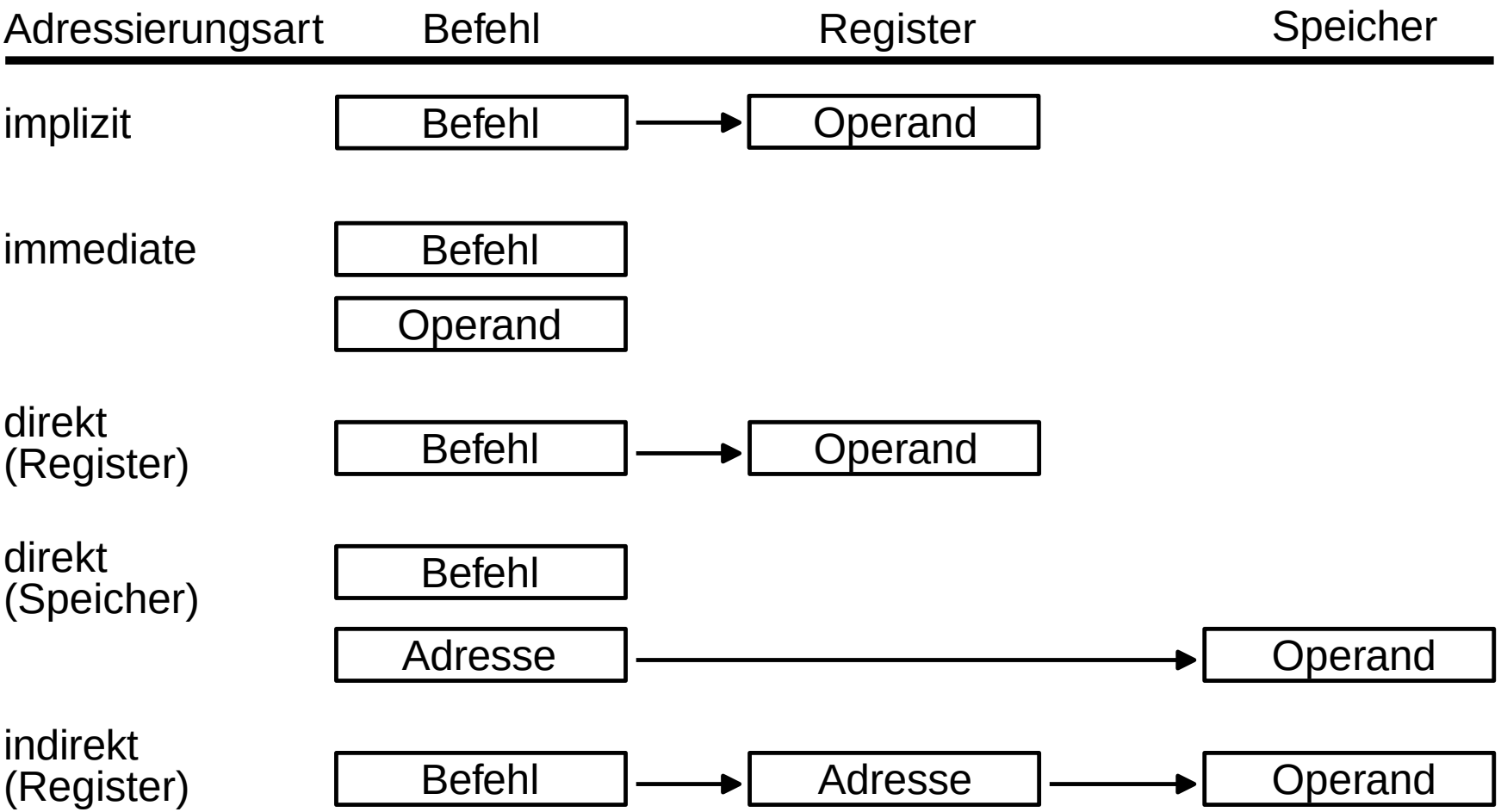
Vorteile des effektiven Einsatzes der Adressierungsarten:

- Einsparung von Hauptspeicherplatz, Rechenzeit und Programmierzeit
- Lageunabhängigkeit der Daten und Programme (relative Adressierung) (Position-Independent Code, PIC)
- Wiederverwendbarkeit von Programmteilen (Unterprogrammtechnik, ...)
- Ermöglichung der wiederholten Befehlsausführung auf verschiedene Daten (z.B. Tabellen, Schleifen), sowie bedingten Operationen (Assembler-Programmierung)

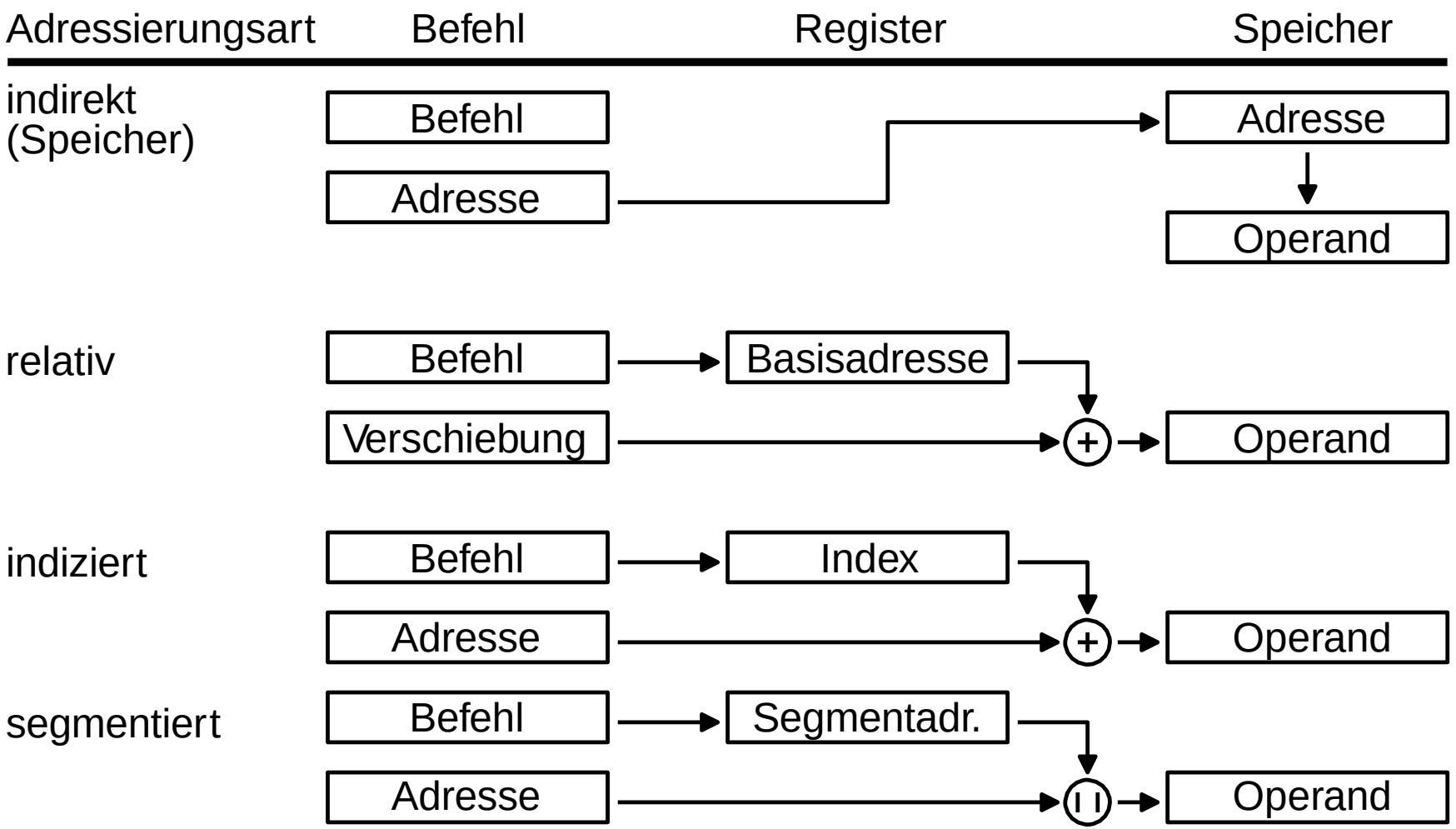
Adressierungsarten

Adress.art	Beschreibung
implizit	Adressen, Operanden sind durch den Opcode selbst festgelegt
immediate	Operand wird im Befehl explizit mitgeführt (Direktoperand)
direkt	Adresse wird im Befehl explizit mitgeführt (Direktadresse)
indirekt	Befehl enthält die Adresse des Speicherplatzes in dem sich die eigentliche Adresse befindet (Adresse von Adresse)
relativ (based)	Befehl enthält einen Offset (Verschiebung) mit dem die Adresse relativ zu einer Basisadresse (Basisregister) gebildet wird
indiziert	Befehl enthält eine Basisadresse, die durch Addition eines Index (Indexregister) modifiziert wird
segmentiert	Adresse wird an den Inhalt eines Segmentregisters angehängt (concatenate, Seitenadressierung)
virtuell	Umsetzung einer virtuellen Adresse in eine physische

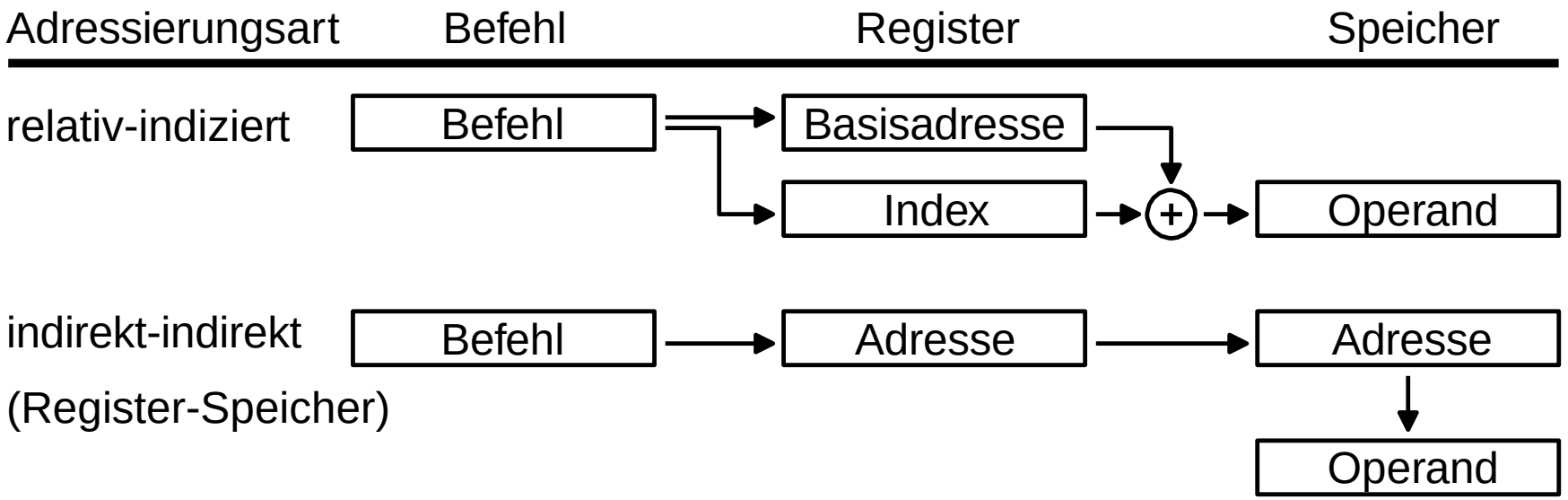
Übersicht Befehl – Register - Speicher



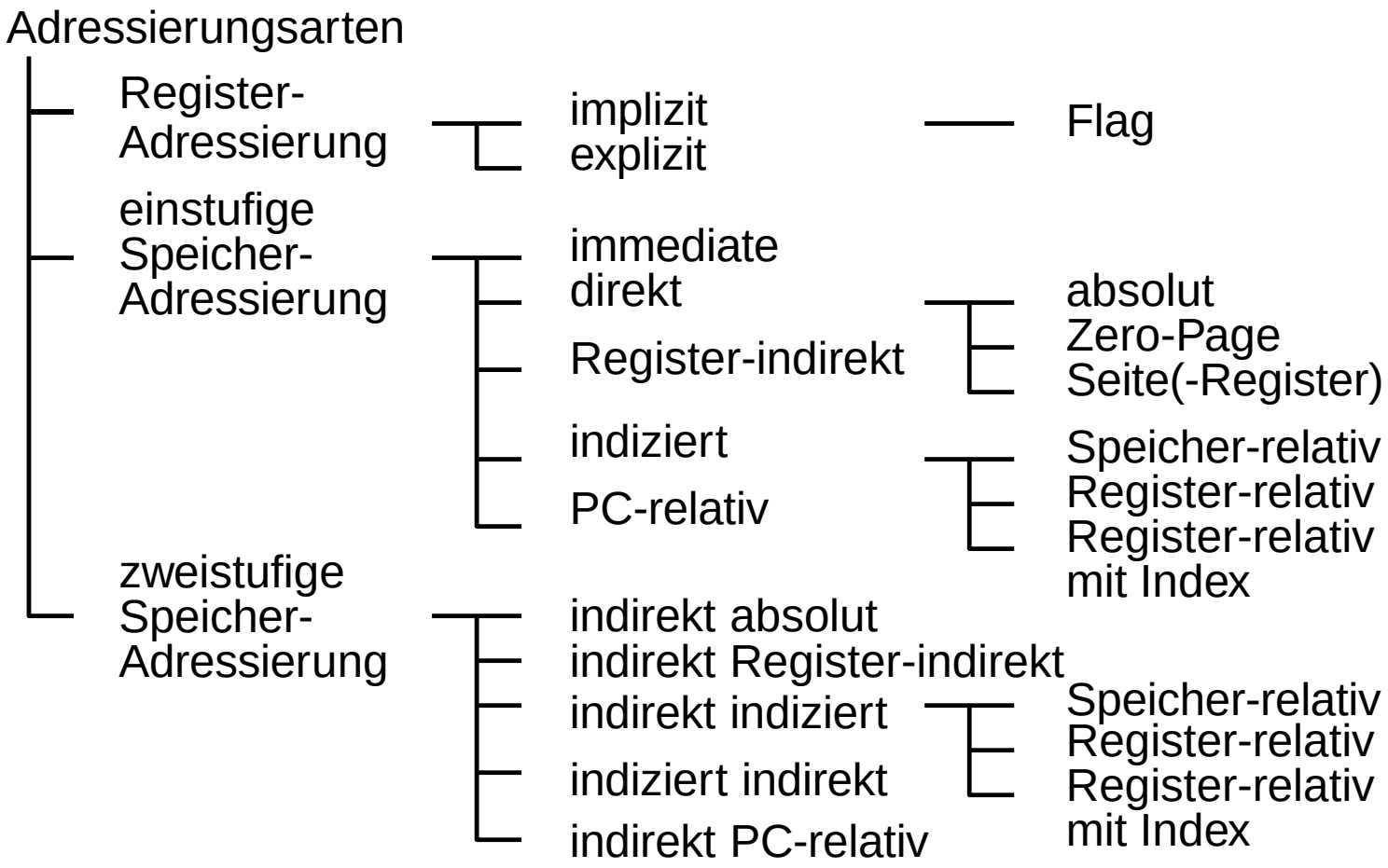
Übersicht Befehl – Register - Speicher



Übersicht Befehl – Register - Speicher



Gebräuchliche Adressierungsarten



Beispiele

Adressierungsart	Befehl (Assembler)	Ergebnis (Zuweisung)
immediate	ADD R2, #7	$R2 := R2 + 7$
direkt	ADD R2, R3	$R2 := R2 + R3$
indirekt (Register)	ADD R2, (R3)	$R2 := R2 + M[R3]$
indirekt (Speicher)	ADD R2, @(R3)	$R2 := R2 + M[M[R3]]$
relativ (Displacement)	ADD R2, 10(R3)	$R2 := R2 + M[10 + R3]$
relativ-indiziert	ADD R2, (R3+R4)	$R2 := R2 + M[R3 + R4]$

Operationen des Befehlsatzes

Bezüglich der Funktionalität sind vier Klassen von Befehlen unterscheidbar:

Operationstyp	Beispiel
Datenübertragung	Register-Register Register-Speicher Register-E/A
Datenmanipulation	arithmetische Operationen logische Operationen Schiebe-/Rotationsoperationen
Verzweigungen (bedingt, unbedingt)	Sprung- und Verzweigungsoperationen Prozedurrufe und -rückkehr Traps
Systemsteuerung	Betriebssystemrufe Speicherverwaltung Interrupts, Traps

Datenübertragung (Beispiele)

Befehl	Bedeutung	
LD	Laden eines Registers	load
ST	Speichern des Inhalts eines Registers	store
MOVE	Übertragung (beliebige Richtung)	move
EXC	Vertauschen der Inhalte	exchange
TFR	Übertragung eines Registers in ein anderes	transfer
PUSH	Ablegen eines oder mehrerer Register in den Stack	push
POP	Laden eines oder mehrerer Register aus dem Stack	pop
READ	Lesen des Prozessor-Statusregisters	read
WRITE	Schreiben des Prozessor-Statusregisters	write
IN	Laden eines Registers aus einem Peripheriebaustein	input
OUT	Übertragen eines Registers in einen Peripheriebaustein	output

Datenmanipulation, arithmetisch- (Beispiele)

Befehl	Bedeutung	
ABS	Absolutbetrag	absolute
ADD	Addition	addition
SUB	Subtraktion	substract
MUL	Multiplikation	multiply
DIV	Division	devide
COM	Einerkomplement	complement
NEG	Vorzeichenwechsel, Zweierkomlement	negate
CLR	Löschen	clear
CMP	Operandenvergleich	compare
DAA	Dual-Dezimal-Umwandlung	decimal adjust
DEC	Dekrement	decrement
INC	Inkrement	increment

Datenmanipulation, Boolesch- (Beispiele)

Befehl	Bedeutung	
AND	AND-Verknüpfung	and
OR	OR-Verknüpfung	or
EOR	XOR-Verknüpfung	exclusive or
NOT	NOT-Verknüpfung	not

Datenmanipulation, Flag- und Bit- (Beispiele)

Befehl	Bedeutung	
SEF	Setzen eines Bedingungs-Flag	flag set
CLF	Löschen eines Bedingungs-Flag	flag clear
TST	Prüfen eines Flags oder Bits	test
BSET	Setzen eines Bit	bit set
BCLR	Rücksetzen eines Bit	bit clear
BCHG	Invertieren eines Bit	bit change
BFCLR	Rücksetzen der Bits eines Bitfeldes	clear bits
BFSET	Setzen der Bits eines Bitfeldes	set bits
BFFFO	Finden der ersten 1 im Bitfeld	find first one
BFEXT	Lesen eines Bitfeldes	extract bits
BFINS	Einfügen eines Bitfeldes	insert bits

Datenmanipulation, String- oder Block- (Beispiele)

Befehl	Bedeutung	
MOVS	Transferieren eines Blocks	move string
INS	Einlesen eines Blocks von der Peripherie	input string
OUTS	Ausgabe eines Blocks an die Peripherie	ooutput string
CMPS	Vergleich zweier Blöcke	compare string
COPS	Kopieren eines Blockes	copy string
SCAS	Suchen eines Zeichens in einem Block	scan string

Datenmanipulation, Schiebe- und Rotation- (Beispiele)

Befehl	Bedeutung	
SHF	Verschieben eines Registerinhlates	shift
ASL	arithmetische Links-Verschiebung	arith. shift left
ASR	arithmetische Rechts-Verschiebung	arith. shift right
LSL	logische Links-Verschiebung	shift left
LSR	logische Rechts-Verschiebung	shift right
ROT	Rotation eines Registerinhaltes	rotate
ROL	Rotation nach links	rotate left
ROR	Rotation nach rechts	rotate right
SWAP	Vertauschen der beiden Hälften eines Registers	swap

Verzweigungen (Beispiele)

Befehl	Bedeutung	
JMP	unbedingter Sprung zu einer Adresse	jump
BCC	Verzweigen falls Bedingung cc erfüllt	branch
BRA	Verzweigen ohne Bedingungsabfrage	branch always
CALL, JSR	Sprung in ein Unterprogramm	jump to subroutine
BSRCC	JSR, wenn Bedingung cc erfüllt	branch to subr.
RTS	Rücksprung aus einem Unterprogramm	return from subr.
TRAP, INT	Sprung in Unterbrechungsroutine	software interrupt
RTI, RTE	Rücksprung aus Unterbrechungsroutine	return from int.

Cc-Bedingungen für Verzweigungen (Beispiele)

cc	Bedingung	Bedeutung
CS	CF=1	branch on carry set
CC	CF=0	branch on carry clear
VS	OF=1	branch on overflow
VC	OF=0	branch on not overflow
EQ	ZF=1	branch on zero/equal
NE	ZF=0	branch on not zero/equal
MI	SF=1	branch on minus
PL	SF=0	branch on plus
PA	PF=1	branch on parity/parity even
NP	PF=0	branch on not parity/parity odd

Cc-Bedingungen für Verzweigungen (Beispiele)

cc	Bedingung	Bedeutung
vorzeichenlose Operanden		
LO	CF=1	branch on lower than
HS	CF=0	branch on higher or same
LS	CF ∨ ZF=1	branch on lower or same
HI	CF ∨ ZF=0	branch on higher than
vorzeichenbehaftete Operanden		
LT	SF ≠ OF=1	branch on less than
GE	SF ≠ OF=0	branch on greater or equal
LE	ZF ∨ (SF ≠ OF)=1	branch on less or equal
GT	ZF ∨ (SF ≠ OF)=0	branch on less greater than

Systemsteuerung (Beispiele)

Befehl	Bedeutung	
NOP	keine Operation	no operation
WAIT	Warten auf spezielles Eingangssignal	wait
SYNC	Wareten auf einen Interrupt	sync
HALT, STOP	Anhalten des Prozessors	stop
RESET	Rücksetzsignal für Pripherie	reset
SVC	Betriebssystem-Aufruf	supervisor call

Typ und Länge der Operanden

Operandentypen (Datentypen)

Der Typ der im Befehl adressierten Operanden wird allgemein im Befehl selbst festgelegt. Die Codierung erfolgt dabei im Opcode zusammen mit der durchzuführenden Operation.

Alternativ wird auch die Abspeicherung von Typkennungen (Tag) zusammen mit den Daten verwendet (Datenflußmaschinen, . . .) verwendet. Sogenannte „Tagged“ Architekturen sind jedoch eher die Ausnahme.

Numerische Daten:

- vorzeichenlose ganze Zahlen (unsigned integer)
- vorzeichenbehaftete ganze Zahlen (signed integer)
- binär codierte Dezimalzahlen (binary coded decimal integer, BCD)
- Gleitkommazahlen (floating point)

Nichtnumerische Daten:

- alphanumerische Zeichen (characters, ASCII)
- Zeichenketten (character strings)
- Boolesche Werte (boolean values)
- Bitfelder (bit map)
- Zeiger, Adressen (pointer)

Operandenlänge (Datenformat)

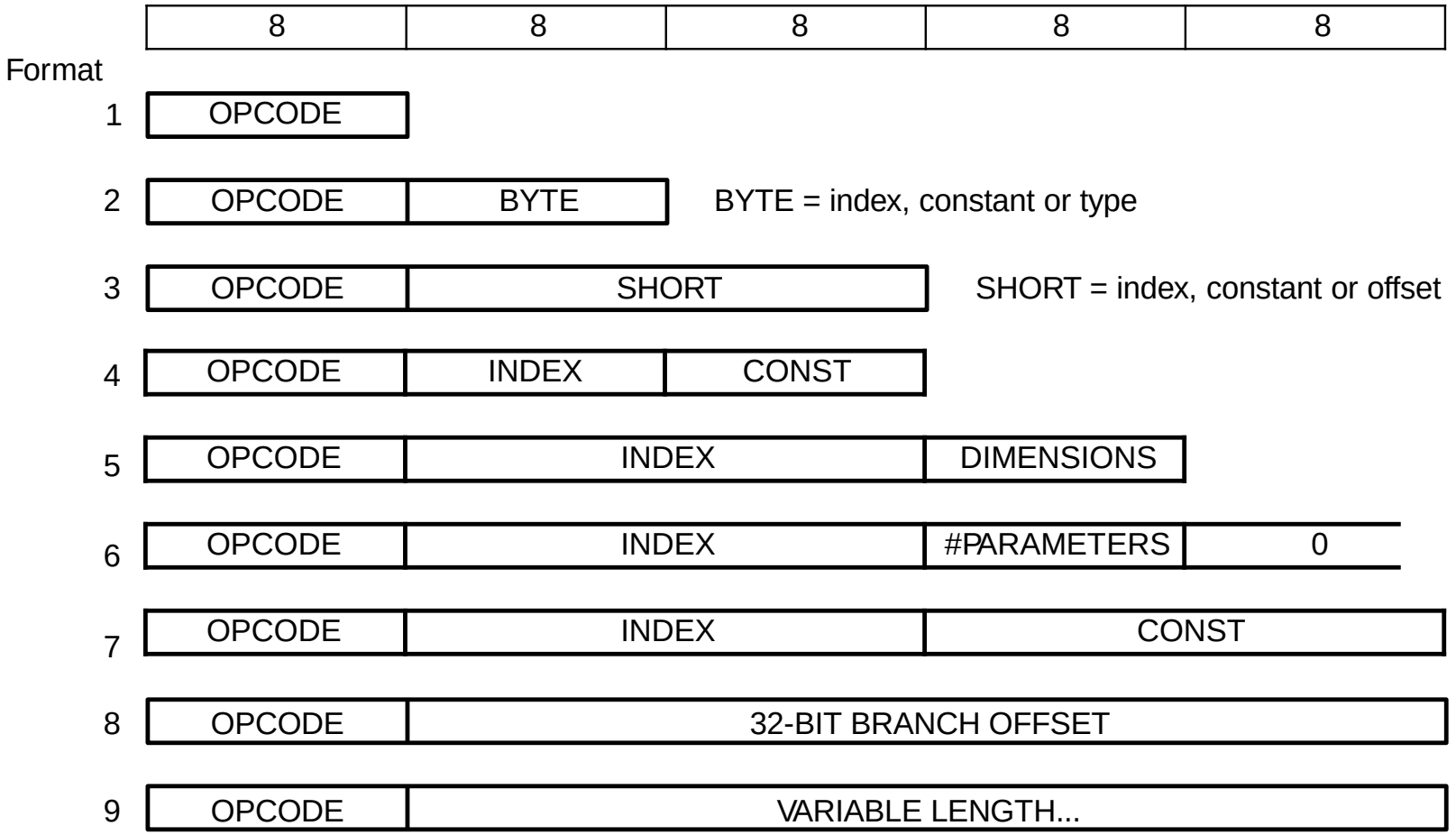
Die Operandenlänge ist allgemein durch den Operandentyp gegeben, wobei pro Typ auch unterschiediche Längen möglich sind (im Befehl codiert).

Zuordnungsbeispiel Operandentyp–Operandenlänge:

Typ	8 Bit	16 Bit	32 Bit	64 Bit	128 Bit
signed integer					
unsigned integer					
BCD					
floating point					
characters					
boolean					
bit map					
pointer					

Beispiele zu Befehlsatz-Architekturen

JVM (STACK) - Instruction Formats



Beispiele zu Befehlsatz-Architekturen

JVM (STACK) - Instruction Set

Load/Store	
typeLOAD ind8	Push local variable onto stack
typeALOAD	Push array element on stack
BALOAD	Push byte from an array on stack
SALOAD	Push short from an array on stack
CALOAD	Push char from an array on stack
AALOAD	Push pointer from an array on stack
typeSTORE ind8	Pop value and store in local var
typeASTORE	Pop value and store in array
BASTORE	Pop byte and store in array
SASTORE	Pop short and store in array
CASTORE	Pop char and store in array
AASTORE	Pop pointer and store in array

Push	
BIPUSH con8	Push a small constant on stack
SIPUSH con16	Push 16-bit constant on stack
LDC ind8	Push constant from const pool
typeCONST_#	Push immediate constant Push
ACONST_NULL	a null pointer on stack

Arithmetic	
typeADD	Add
typeSUB	Subtract

Arithmetic (continued)	
typeMUL	Multiple
typeDIV	Divide
typeREM	Remainder
typeNEG	Negate

Boolean/Shift	
ilAND	Boolean AND
ilOR	Boolean OR
ilXOR	Boolean Exclusive OR
ilSHL	Shift left
ilSHR	Shift right
ilUSHR	Unsigned shift right

Conversion	
x2y	Convert x to y
x2y	Convert x to y
i2c	Convert integer to char
12b	Convert integer to byte

Stack Management	
DUPxx	Six instructions for duping
POP	Pop an int from stk and discard
POP2	Pop two ints from stk and discard
SWAP	Swap top two ints on stack

Beispiele zu Befehlsatz-Architekturen

JVM (STACK) - Instruction Set

Comparison

IF_ICMPrel offset16	Conditional branch
IF_ACMPEQ offset16	Branch if two ptrs equal
IF_ACMPNE offset16	Branch if ptrs unequal
IFrel offset16 IFNULL	Test 1 value and branc
offset16 IFNONNULL	Branch if ptr is null
offset16 LCMP	Branch if ptr is nonnull
FCMPL	Compare two longs
FCMPG	Compare 2 fbats for <
DCMPL	Compare 2 fbats for >
DCMPG	Compare doubles for <
	Compare doubles for >

Transfer of Control

INVOKEVIRTUAL ind16	Method invocation
INVOKESTATIC ind16	Method invocation
INVOKEINTERFACE ...	Method invocation
INVOKESPECIAL ind16	Method invocation
JSR offset16	Invoke fi nally clause
typeRETURN	Return value
ARETURN	Return pointer
RETURN	Return void Return
RET ind8	from fi nally
GOTO offset16	Unconditional branch

Arrays

ANEWARRAY ind16	Create array of ptrs
NEWARRAY atype	Create array of atype
MULTINEWARRAY ind16,d	Create multidim array
ARRAYLENGTH	Get array length

Miscellaneous

IINC ind8,con8	Increment local variabl
WIDE	Wide prefi x
NOP	No operation
GETFIELD ind16	Read fi eld from object
PUTFIELD ind16	Write fi eld to object
GETSTATIC ind16	Get static fi eld from class
NEW ind16	Create a new object
INSTANCEOF offset16	Determine type of obj
CHECKCAST ind16	Check object type
ATHROW	Throw exception Sparse
LOOKUPSWITCH ...	multiway branch Dense
TABLESWITCH ...	multiway branch Enter a
MONITORENTER	monitor
MONITOREXIT	Leave a monitor

ind8/16 = index of local variable
con8/16, d, atype = constant
type, x, y = I, L, F, D offset16 for branch

Beispiele zu Befehlsatz-Architekturen

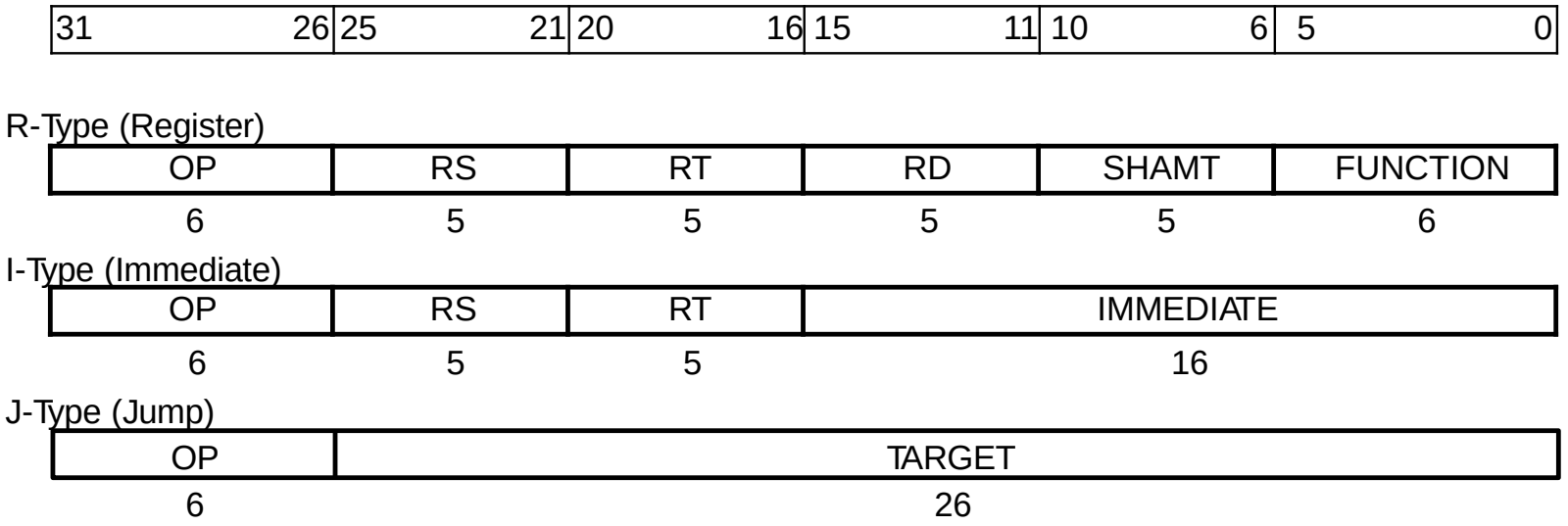
JVM (STACK) - Numeric Data Types / Addressing Modes

Type	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Signed integer	X	X	X	X	
Unsigned integer					
Binary coded decimal integer					
Floating point			X	X	

Addressing Mode	
Immediate	X
Direct	
Register	
Register indirect	
Indexed	X
Based-Indexed	
Stack	X

Beispiele zu Befehlsatz-Architekturen

R2000 (MIPS) - Instruction Formats



OP	- operation code
RS	- source register specifier
RT	- target (source/destination) register or branch condition
RD	- destination register specifier
SHAMT	- shift amount
FUNCTION	- function field
IMMEDIATE	- immediate, branch displacement or adress displacement
TARGET	- jump target adress

Beispiele zu Befehlsatz-Architekturen

R2000 (MIPS) - Instruction Set

Load/Store

LB rt,offset(base)	Load byte
LBU rt,offset(base)	Load byte unsigned
LH rt,offset(base)	Load halfword
LHU rt,offset(base)	Load halfword unsigned
LW rt,offset(base)	Load word
LWL rt,offset(base)	Load word left
LWR rt,offset(base)	Load word right
SB rt,offset(base)	Store byte
SH rt,offset(base)	Store halfword
SW rt,offset(base)	Store word
SWL rt,offset(base)	Store word left
SWR rt,offset(base)	Store word right

Arithmetic (I-Type)

ADDI rt,rs,immediate	Add immediate
ADDIU rt,rs,immediate	Add immediate unsigned
SLTI rt,rs,immediate	Set on less than immediate
SLTIU rt,rs,immediate	Set on less than immediate unsigned
ANDI rt,rs,immediate	AND immediate
ORI rt,rs,immediate	OR immediate
XORI rt,rs,immediate	Exclusive OR immediate
LUI rt,immediate	Load upper immediate

Arithmetic (R-Type)

ADD rd,rs,rt	Add
ADDU rd,rs,rt	Add unsigned
SUB rd,rs,rt	Subtract
SUBU rd,rs,rt	Subtract unsigned
SLT rd,rs,rt	Set on less than
SLTU rd,rs,rt	Set on less than unsigned
AND rd,rs,rt	AND
OR rd,rs,rt	OR
XOR rd,rs,rt	Exclusive OR
NOR rd,rs,rt	OR-NOT

Shift

SLL rd,rt,shamt	Shift left logical
SRL rd,rt,shamt	Shift right logical
SRA rd,rt,shamt	Shift right arithmetic
SLLV rd,rt,rs	Shift left logical variable
SRLV rd,rt,rs	Shift right logical variable
SRAV rd,rt,rs	Shift right arithmetic variable

Special

SYSCALL	System call
BREAK	Breakpoint

Beispiele zu Befehlsatz-Architekturen

R2000 (MIPS) - Instruction Set

Multiply/Divide	
MULT rs,rt	Multiply
MULTU rs,rt	Multiply unsigned
DIV rs,rt	Divide
DIVU rs,rt	Divide unsigned
MFHI rd	Move from HI
MFLO rd	Move from LO
MTHI rd	Move to HI
MTLO rd	Move to LO

Jump/Branch	
J target	Jump
JAL target	Jump and link
JR rs	Jump register
JALR rs,rd	Jump and link register
BEQ rs,rt,offset	Branch on equal
BNE rs,rt,offset	Branch on not equal
BLEZ rs,offset	Branch on ≤ zero
BGTZ rs,offset	Branch on > zero
BLTZ rs,offset	Branch on < zero
BGEZ rs,offset	Branch on ≥ zero
BLTZAL rs,offset	Branch on < zero and link
BGEZAL rs,offset	Branch on ≥ zero and link

Coprocessor	
LWCz rt,offset(base)	Load word to coprocessor
SWCz rt,offset(base)	Store word from coprocessor
MTCz rt,rd	Move to coprocessor
MFCz rt,rd	Move from coprocessor
CTCz rt,rd	Move control to coprocessor
CFCz rt,rd	Move control from coprocessor
COPz cofun	Coprocessor operation
BCzT offset	Branch on coprocessor z true
BCzF offset	Branch on coprocessor z false

System Control Coprocessor (CP0)	
MTC0 rt,rd	Move to CP0
MFC0 rt,rd	Move from CP0
TLBR	Read indexed TLB entry
TLBWI	Write indexed TLB entry
TLBWR	Write random TLB entry
TLBP	Probe TLB for matching entry
RFE	Restore from exception

Beispiele zu Befehlsatz-Architekturen

R2000 (MIPS) - Numeric Data Types / Addressing Modes

Type	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Signed integer	X	X	X	X	
Unsigned integer	X	X			
Binary coded decimal integer					
Floating point			(X)[1]	(X)[1]	

[1] - only with FPU R2010

Addressing Mode	
Immediate	X
Direct	
Register	X
Register indirect	X
Indexed	
Based-Indexed	
Stack	

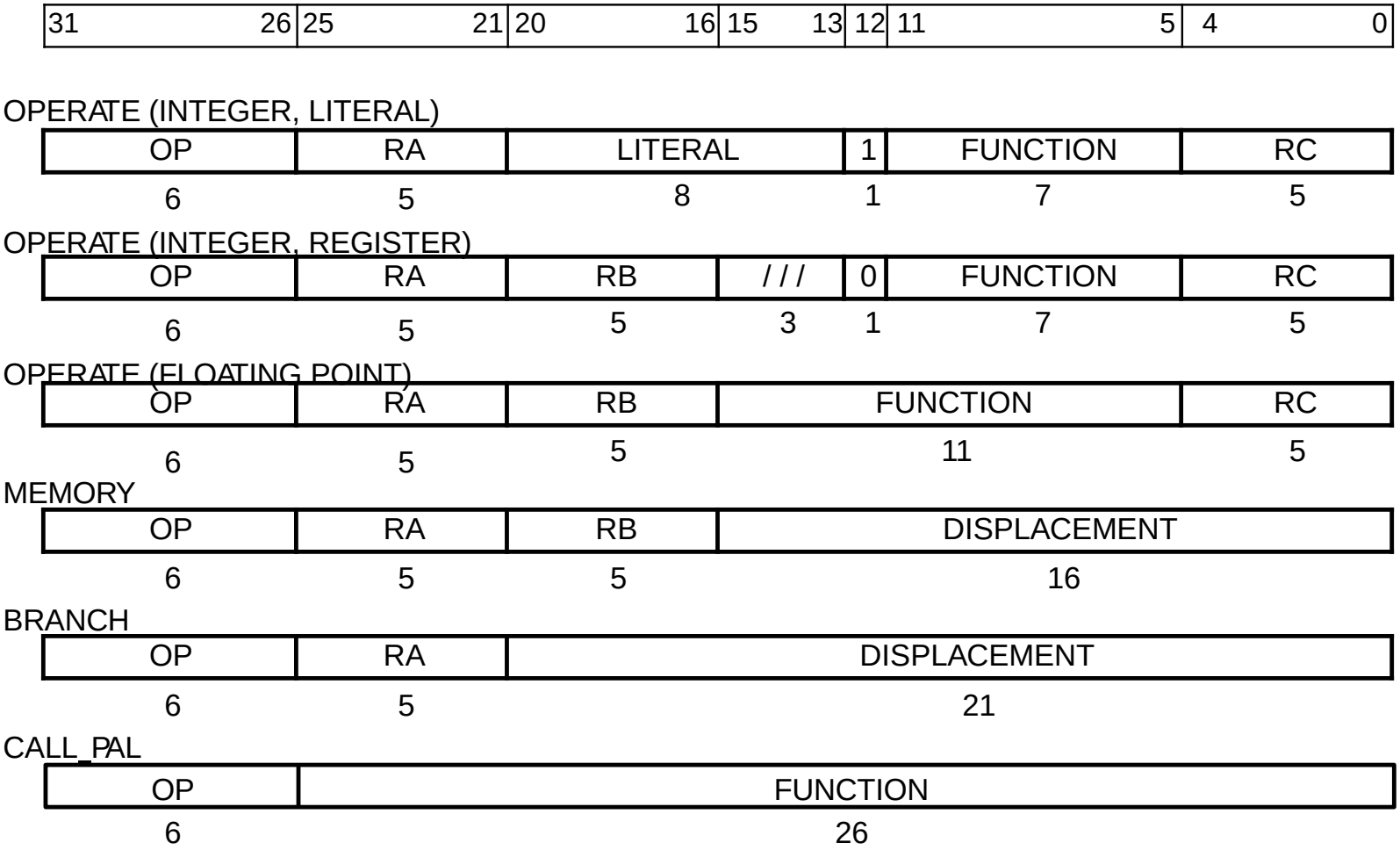
Beispiele zu Befehlsatz-Architekturen

R2000 (MIPS) - Register

Number	Register Name	Usage
0	zero	Constant 0
1	at	Reserved for assembler
2 - 3	v0 - v1	Expression evaluation and results of a function
4 - 7	a0 - a3	Argument
8 - 15	t0 - t7	Temporary
16 - 23	s0 - s7	Saved temporary
24 - 25	t8 - t9	Temporary
26 - 27	k0 - k1	Reserved for OS kernel
28	gp	Pointer to global area
29	sp	Stack pointer
30	fp	Frame pointer
31	ra	Return address

Beispiele zu Befehlsatz-Architekturen

ALPHA AXP - Instruction Formats



Beispiele zu Befehlsatz-Architekturen

ALPHA AXP - Instruction Set

Load/Store, Byte Manipulation	
LDA	Load address
LDAH	Load address high
LDL	Load sign-extended longword
LDQ	Load quadword
LDQ_U	Load unaligned quadword
LDL_L	Load sign-extended longword, locked
LDQ_L	Load quadword, locked
STL_C	Store longword, conditional
STQ_C	Store quadword, conditional
STL	Store longword
STQ	Store quadword
STQ_U	Store unaligned quadword
EXTBL	Extract byte low
EXTWL	Extract word low
EXTLL	Extract longword low
EXTQL	Extract quadword low
EXTWH	Extract word high
EXTLH	Extract longword high
EXTQH	Extract quadword high
INSBL	Insert byte low
INSWL	Insert word low
INSLL	Insert longword low
INSQL	Insert quadword low

Load/Store, Byte Manipulation (continued)	
INSWH	Insert word high
INSLH	Insert longword high
INSQH	Insert quadword high
MSKBL	Mask byte low
MSKWL	Mask word low
MSKLL	Mask longword low
MSKQL	Mask quadword low
MSKWH	Mask word high
MSKLH	Mask longword high
MSKQH	Mask quadword high

Floating Point Load/Store	
LDF	Load F format (VAX single)
LDG	Load G format (VAX double)
LDS	Load S format (IEEE single)
LDT	Load T format (IEEE double)
STF	Store F format (VAX single)
STG	Store G format (VAX double)
STS	Store S format (IEEE single)
STT	Store T format (IEEE double)

Beispiele zu Befehlsatz-Architekturen

ALPHA AXP - Instruction Set

Integer Computation and Conditional Move	
ADDL	Add longword
S4ADDL	Add longword, scale by 4
S8ADDL	Add longword, scale by 8
ADDQ	Add quadword
S4ADDQ	Add quadword, scale by 4 Add
S8ADDQ	quadword, scale by 8
CMPEQ	Compare signed quadword =
CMPLT	Compare signed quadword <
CMPLE	Compare signed quadword ≤
CMPULT	Compare unsigned quadword <
CMPULE	Compare unsigned quadword ≤
MULL	Multiply longword
MULQ	Multiply quadword
UMULH	Multiply quadword high, unsigned
SUBL	Subtract longword
S4SUBL	Subtract longword, scale by 4
S8SUBL	Subtract longword, scale by 8
SUBQ	Subtract quadword
S4SUBQ	Subtract quadword, scale by 4
S8SUBQ	Subtract quadword, scale by 8

Integer Computation and ... (continued)	
AND	AND logical
BIS	OR logical
XOR	Exclusive-OR logical
BIC	AND-NOT logical
ORNOT	OR-NOT logical
EQV	Exclusive-OR-NOT logical
SLL	Shift left, logical
SRL	Shift right, logical
SRA	Shift right, arithmetic
CMOVEQ	Conditional move if reg = 0
CMOVNE	Conditional move if reg /= 0
CMOVLT	Conditional move if reg < 0
CMOVLE	Conditional move if reg ≤ 0
CMOVGT	Conditional move if reg > 0
CMOVGE	Conditional move if reg ≥ 0
CMOVLBC	Conditional move if reg, low bit clear
CMOVLBS	Conditional move if reg, low bit set
CMPBGE	Compare bytes, unsigned
ZAP	Clear selected bytes
ZAPNOT	Clear inselected bytes

Beispiele zu Befehlsatz-Architekturen

ALPHA AXP - Instruction Set

Integer Branch

BEQ	Branch if reg = 0
BNE	Branch if reg /= 0
BLT	Branch if reg < 0
BLE	Branch if reg ≤ 0
BGT	Branch if reg > 0
BGE	Branch if reg ≥ 0
BLBC	Branch if low bit clear
BLBS	Branch if low bit set
BR	Branch
BSR	Branch to subroutine
JMP	Jump
JSR	Jump to subroutine
RET	Return from subroutine
JSR_COROUTINE	Jump to subroutine, return

Floating Point Branch

FBEQ	FP Branch if = 0
FBNE	FP Branch if /= 0
FBLT	FP Branch if < 0
FBLE	FP Branch if ≤ 0
FBGT	FP Branch if > 0
FBGE	FP Branch if ≥ 0

Floating Point Computation and Conditional Move

CPYS	Copy sign
CPYSN	Copy sign, negate
CPYSE	Copy sign and exponent
CVTQL	Convert quadword to longword
CVTLQ	Convert longword to quadword
FCMOVEQ	FP conditional move if reg = 0
FCMOVNE	FP conditional move if reg /= 0
FCMOVLT	FP conditional move if reg < 0
FCMOVLE	FP conditional move if reg ≤ 0
FCMOVGT	FP conditional move if reg > 0
FCMOVGE	FP conditional move if reg ≥ 0
MF_FPCR	Move from FP control register
MT_FPCR	Move to FP control register
ADDF	Add F format (VAX single)
ADDG	Add G format (VAX double)
ADDS	Add S format (IEEE single)
ADDT	Add T format (IEEE double)
CMPGEQ	Compare G format = (VAX double)
CMPGLT	Compare G format < (VAX double)
CMPGLE	Compare G format ≤ (VAX double)
CMPTEQ	Compare T format = (IEEE double)
CMPTLT	Compare T format < (IEEE double)
CMPTLE	Compare T format ≤ (IEEE double)

Beispiele zu Befehlsatz-Architekturen

ALPHA AXP - Instruction Set

FP Computation and Conditional Move (continued)	
CMPTUN	Compare T format unordered (IEEE double)
CVTGQ	Convert G format to quadword (VAX double)
CVTQF	Convert quadword to F format (VAX single)
CVTQG	Convert quadword to G format (VAX double)
CVTDG	Convert D to G format (VAX double/double)
CVTGD	Convert G to D format (VAX double/double)
CVTGF	Convert G to F format (VAX double/single)
CVTTQ	Convert T format to quadword (IEEE double)
CVTQS	Convert quadword to S format (IEEE single)
CVTQT	Convert quadword to T format (IEEE double)
CVTTS	Convert T to S format (IEEE double/single)
CVTST	Convert S to T format (IEEE single/double)
DIVF	Divide F format (VAX single)
DIVG	Divide G format (VAX double)
DIVS	Divide S format (IEEE single)
DIVT	Divide T format (IEEE double)
MULF	Multiply F format (VAX single)
MULG	Multiply G format (VAX double)
MULS	Multiply S format (IEEE single)
MULT	Multiply T format (IEEE double)

FP Computation and ... (continued)	
SUBF	Subtract F format (VAX single)
SUBG	Subtract G format (VAX double)
SUBS	Subtract S format (IEEE single)
SUBT	Subtract T format (IEEE double)

System	
CALL_PAL	Call privileged architecture library
TRAPB	Trap barrier (precise exception)
FETCH	Prefetch (cache) data hint
FETCH_M	Prefetch (cache) data, modify hint
MB	Memory barrier (serialize)
WMB	Memory barrier (serialize) write
RPCC	Read process cycle counter Read
RC	and clear
RS	Read and set
PALRES0	PALcode reserved opcode 0
PALRES1	PALcode reserved opcode 1
PALRES2	PALcode reserved opcode 2
PALRES3	PALcode reserved opcode 3
PALRES4	PALcode reserved opcode 4

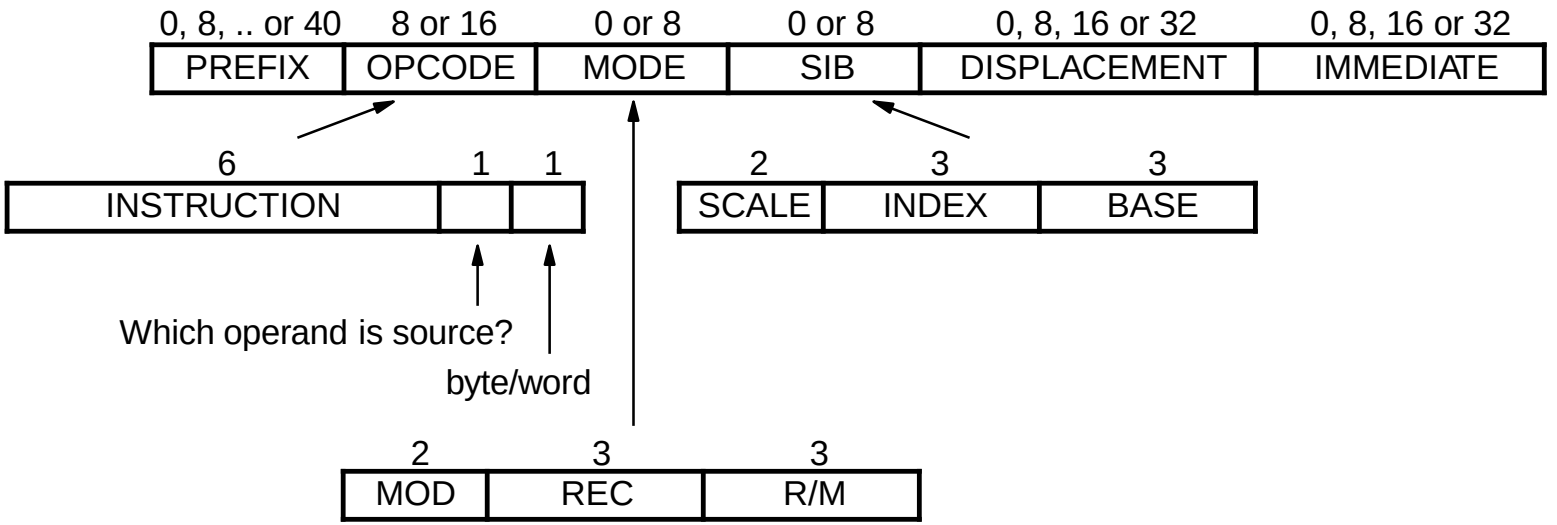
Beispiele zu Befehlsatz-Architekturen

ALPHA AXP - Numeric Data Types

Type	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Signed integer			X	X	
Unsigned integer					
Binary coded decimal integer					
Floating point			X	X	

Beispiele zu Befehlsatz-Architekturen

Intel Pentium II - Instruction Formats



MODE	- specifer for the opcode
SIB	- (scale, index, base) special modes for a further specifi cation
DISPLACEMENT	- specifying a memory address (displacement)
IMMEDIATE	- containing a constant (immediate operand)

Beispiele zu Befehlsatz-Architekturen

Intel Pentium II - Instruction Set

A selection of integer instructions

Move	
MOV dst,src	Move SRC to DST
PUSH src POP	Push SRC onto the stack
dst XCHG	Pop a word from the stack to DST
ds1,ds2 LEA	Exchange DS1 and DS2
dst,src CMOV	Load effective addr of SRC into DST
dst,src	Conditional move

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

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 1s complement

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

Test/Compare	
TST src1,src2	Boolean AND operands, set flags
CMP src1,src2	Set flags based on SRC1 _SRC2

Beispiele zu Befehlsatz-Architekturen

Intel Pentium II - Instruction Set

A selection of integer instructions

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 addr	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

Condition Codes	
STC	Set carry bit in EFLAGS register Clear
CLC	carry bit in EFLAGS register
CMC	Complement carry bit in EFLAGS reg
STD	Set direction bit in EFLAGS register
CLD	Clear direction bit in EFLAGS register
STI	Set interrupt bit in EFLAGS register

Condition Codes (continued)	
CLI	Set interrupt bit in EFLAGS register
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

Beispiele zu Befehlsatz-Architekturen

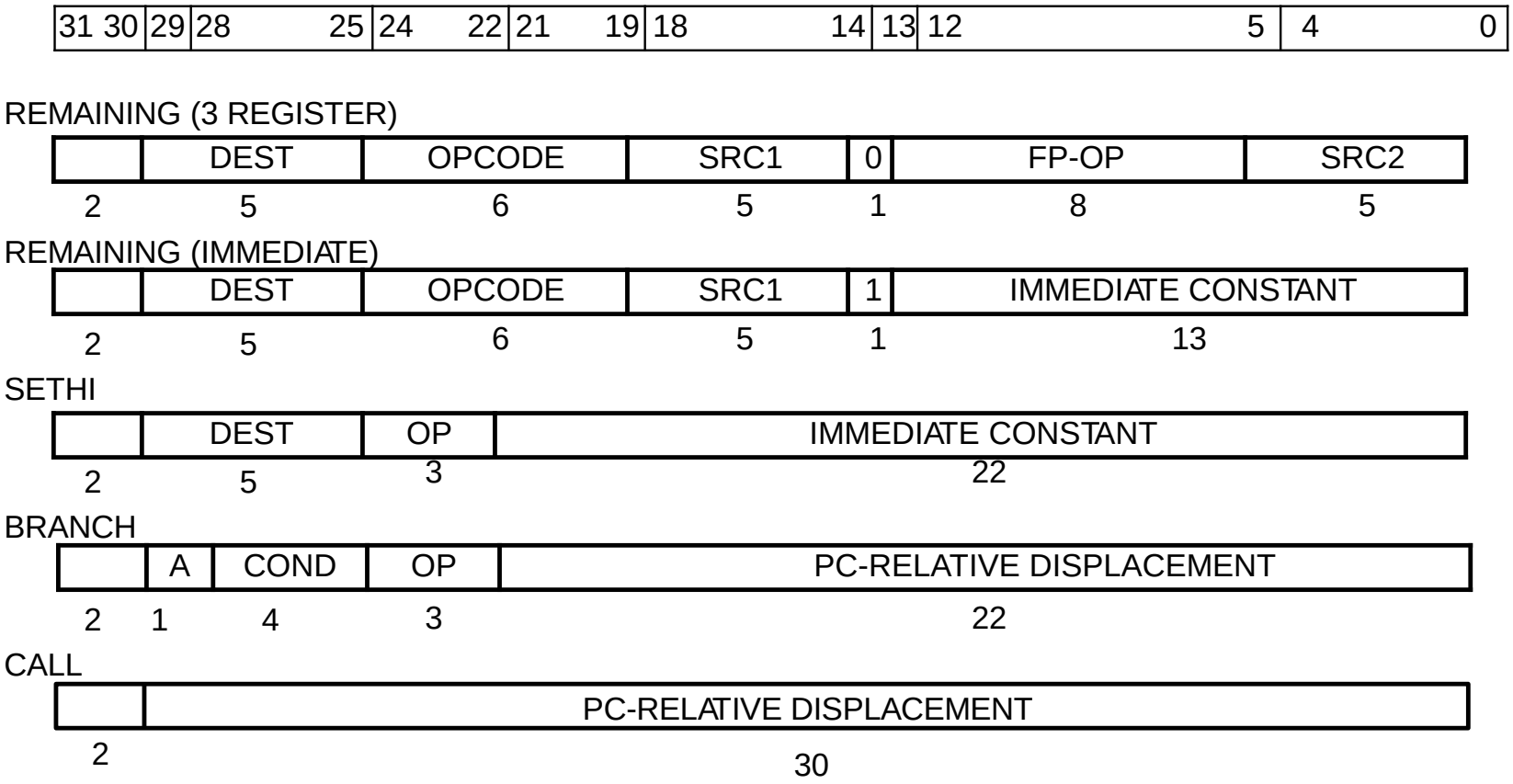
Intel Pentium II - Numeric Data Types / Addressing Modes

Type	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Signed integer	X	X	X		
Unsigned integer	X	X	X		
Binary coded decimal integer	X				
Floating point			X	X	

Addressing Mode	
Immediate	X
Direct	X
Register	X
Register indirect	X
Indexed	X
Based-Indexed	
Stack	

Beispiele zu Befehlsatz-Architekturen

UltraSPARC II - Instruction Formats



Beispiele zu Befehlsatz-Architekturen

UltraSPARC II - Instruction Set

The primary integer instructions

Load/Store	
LDSB addr,dst	Load signed byte
LDUB addr,dst	Load unsigned byte
LDSH addr,dst	Load signed halfword
LDUH addr,dst	Load unsigned halfword
LDSW addr,dst	Load signed word
LDUW addr,dst	Load unsigned word
LDX addr,dst	Load extended (64 bits)
STB src,addr	Store byte
STH src,addr	Store halfword
STW src,addr	Store word
STX src,addr	Store extended

Arithmetic	
ADD r1,s2,dst	Add
ADDCC r1,s2,dst	Add and set icc
ADDC r1,s2,dst	Add with carry
ADDCCC r1,s2,dst	Add with carry and set icc
SUB r1,s2,dst	Subtract
SUBCC r1,s2,dst	Subtract and set icc
SUBC r1,s2,dst	Subtract with carry
SUBCCC r1,s2,dst	Sub with carry and set icc
MULX r1,s2,dst	Multiply
SDIVX r1,s2,dst	Signed divide

Arithmetic (continued)	
UDIVX r1,s2,dst	Unsigned divide
TADCC r1,s2,dst	Tagged add

Shift/Rotate	
SLL r1,s2,dst	Shift left logical (32 bits)
SLLX r1,s2,dst	Shift left logical extended (64 bits)
SRL r1,s2,dst	Shift right logical
SRLX r1,s2,dst	Shift right logical extended
SRA r1,s2,dst	Shift right arithmetic
SRAX r1,s2,dst	Shift right arithmetic extended

Boolean	
AND r1,s2,dst	Boolean AND
ANDCC r1,s2,dst	Boolean AND and set icc
ANDN r1,s2,dst	Boolean AND-NOT
ANDNCC r1,s2,dst	Boolean AND-NOT and set icc
OR r1,s2,dst	Boolean OR
ORCC r1,s2,dst	Boolean OR and set icc
ORN r1,s2,dst	Boolean OR-NOT
ORNCC r1,s2,dst	Boolean OR-NOT and set icc
XOR r1,s2,dst	Boolean Exclusive-OR
XORCC r1,s2,dst	Boolean Exclusive-OR and set icc

Beispiele zu Befehlsatz-Architekturen

UltraSPARC II - Instruction Set

The primary integer instructions

Boolean (continued)

XNOR r1,s2,dst	Boolean Exclusive-OR-NOT
XNORCC r1,s2,dst	Boolean XOR-NOT and set icc

Transfer of Control

BPcc addr	Branch with prediction
BPr scr,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 r1,s2,dst	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

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 r1,dst	Population count
RDCCR v,dst	Read condition code register
WRCCR r1,s2,v	Write condition code register
RDPC v,dst	Read program counter

scr = 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

Beispiele zu Befehlsatz-Architekturen

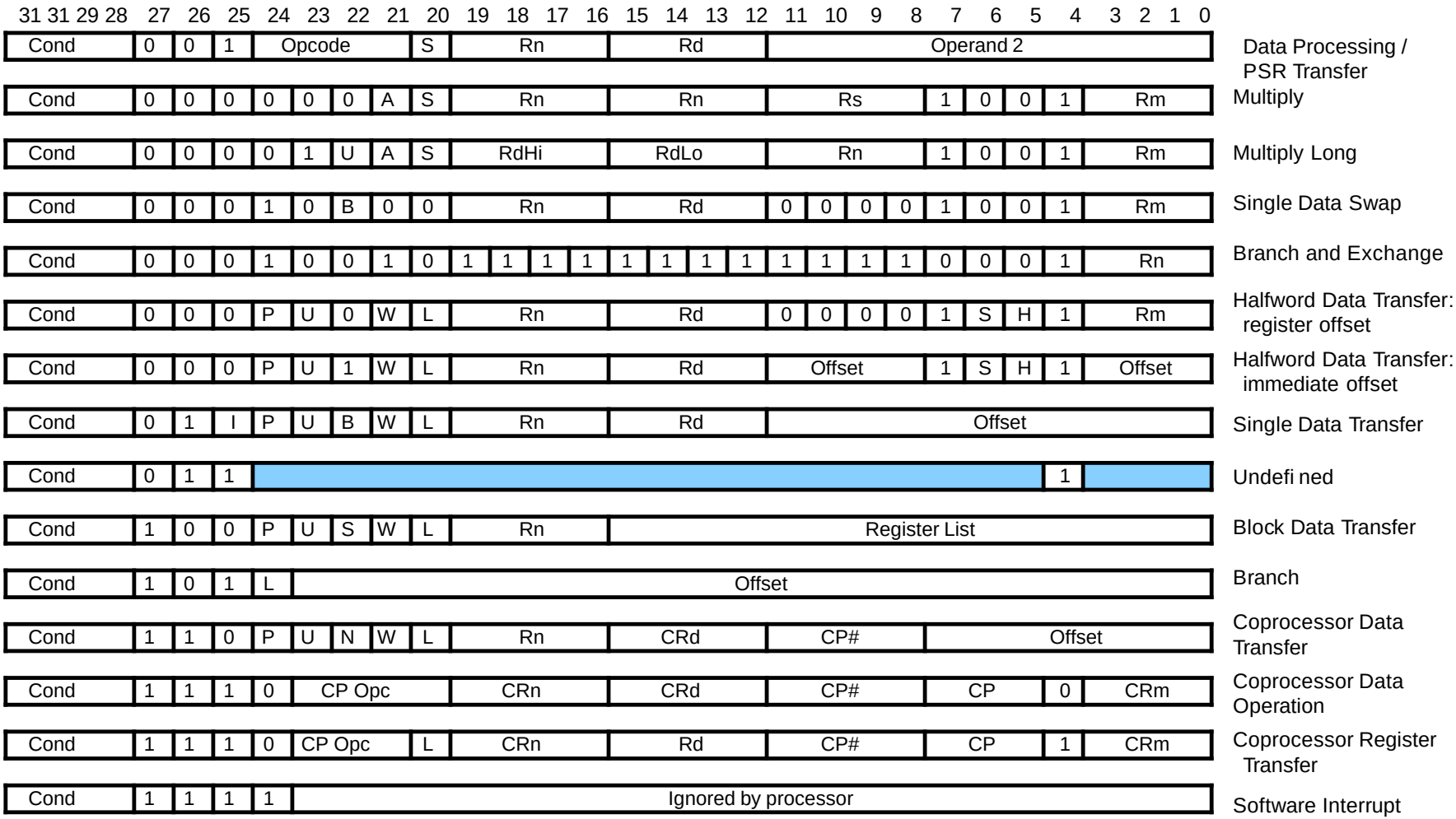
UltraSPARC II - Numeric Data Types / Addressing Modes

Type	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Signed integer	X	X	X	X	
Unsigned integer	X	X	X	X	
Binary coded decimal integer					
Floating point			X	X	X

Addressing Mode	
Immediate	X
Direct	
Register	X
Register indirect	
Indexed	X
Based-Indexed	X
Stack	

Beispiele zu Befehlsatz-Architekturen

AT91 ARM - Instruction Formats



Beispiele zu Befehlsatz-Architekturen

AT91 ARM - Instruction Set

Load/Store/Move	
LDR	Load register from memory
LDM	Load multiple registers (Pop from stack)
LDC	Load coprocessor from memory
STR	Store register to memory
STM	Store multiple (Push into stack)
STC	Store coprocessor register to memory
MOV	Move register or constant
MCR	Move CPU register to coprocessor register
MRC	Move from coprocessor to CPU register
MRS	Move PSR status/flags to register
MSR	Move register to PSR status/flags
MVN	Move negative register

Arithmetic	
ADC	Add with carry
ADD	Add
RSB	Reverse Subtract
RSC	Reverse Subtract with Carry
SBC	Subtract with Carry
SUB	Subtract
MLA	Multiply Accumulate
MUL	Multiply

Boolean	
AND	AND
EOR	Exclusive OR
ORR	OR

Branch	
B	Branch
BL	Branch with Link
BX	Branch and Exchange

Miscellaneous	
CMP	Compare
CMN	Compare Negative
TST	Test bits
TEQ	Test bitwise equality
BIC	Bit Clear
SWP	Swap register with memory
SWI	Software Interrupt
CDP	Coprocessor Data Processing

Beispiele zu Befehlsatz-Architekturen

AMD Athlon (x86) - Instruction Set

Integer Instructions

AAA	CMOVA/CMOVNBE	CWD/CDQ	JNP/JPO
AAD	CMOVAE/CMOVNB/CMOVNC	DAA	JL/JNGE
AAM	CMOVB/CMOVC/CMOVNAE	DAS	JNL/JGE
AAS	CMOVBE/CMOVNA	DEC	JLE/JNG
ADC	CMOVE/CMOVZ	DIV	JNLE/JG
ADD	CMOVG/CMOVNLE	ENTER	JCXZ/JEC
AND	CMOVGE/CMOVNL	IDIV	JMP
ARPL	CMOVL/CMOVNGE	IMUL	LAHF
BOUND	CMOVLE/CMOVNG	IN	LAR
BSF	CMOVNE/CMOVNZ	INC	LDS
BSR	CMOVNO	INVD	LEA
BSWAP	CMOVNP/CMOVPO	INVLPG	LEAVE
BT	CMOVNS	JO	LES
BTC	CMOVO	JNO	LFS
BTR	CMOVP/CMOVPE	JB/JNAE/JC	LGDT
BTS	CMOVS	JNB/JAE/JNC	LGS
CALL	CMP	JZ/JE	LIDT
CBW/CWDE	CMPSB	JNZ/JNE	LLDT
CLC	CMPSW	JBE/JNA	LMSW
CLD	CMPSD	JNBE/JA	LODSB AL
CLI	CMPXCHG	JS	LODSW AX
CLTS	CMPXCHG8B	JNS	LODSD EAX
CMC	CPUID	JP/JPE	LOOP

Beispiele zu Befehlsatz-Architekturen

AMD Athlon (x86) - Instruction Set

Integer Instructions (continued)

LOOPE/LOOPZ	PUSHF/PUSHFD	SETS	SUB
LOOPNE/LOOPNZ	RCL	SETNS	SYSCALL
LSL	RCR	SETP/SETPE	SYSENTER
LSS	RDMSR	SETNP/SETPO	SYSEXIT
LTR	RDPMC	SETL/SETNGE	SYSRET
MOV	RDTSC	SETGE/SETNL	TEST
MOVSB	RET	SETLE/SETNG	VERR
MOVSD	ROL	SETG/SETNLE	VERW
MOVSW	ROR	SGDT	WAIT
MOVSX	SAHF	SIDT	WBINVD
MOVZX	SAR	SHL/SAL	WRMSR
MUL	SBB	SHR	XADD
MULEAX	SCASB	SHLD	XCHG
NEG	SCASW	SHRD	XLAT
NOP	SCASD	SLDT	XOR
NOT	SETO	SMSW	
OR	SETNO	STC	
OUT	SETB/SETC/SETNAE	STD	
POP	SETAE/SETNB/SETNC	STI	
POPA/POPAD	SETE/SETZ	STOSB	
POPF/POPFD	SETNE/SETNZ	STOSW	
PUSH	SETBE/SETNA	STOSD	
PUSHA/PUSHAD	SETA/SETNBE	STR	

Beispiele zu Befehlsatz-Architekturen

AMD Athlon (x86) - Instruction Set

MMX Instructions

EMMS	PMULLW
MOVD	POR
MOVQ	PSLLD
PACKSSDW	PSLLQ
PACKSSWB	PSLLW
PACKUSWB	PSRAW
PADDB	PSRAD
PADDD	PSRLD
PADDSB	PSRLQ
PADDSW	PSRLW
PADDUSB	PSUBB
PADDUSW	PSUBD
PADDW	PSUBSB
PAND	PSUBSW
PANDN	PSUBUSB
PCMPEQB	PSUBUSW
PCMPEQD	PSUBW
PCMPEQW	PUNPCKHBW
PCMPGTB	PUNPCKHDQ
PCMPGTD	PUNPCKHWD
PCMPGTW	PUNPCKLBW
PMADDWD	PUNPCKLDQ
PMULHW	PUNPCKLWD