

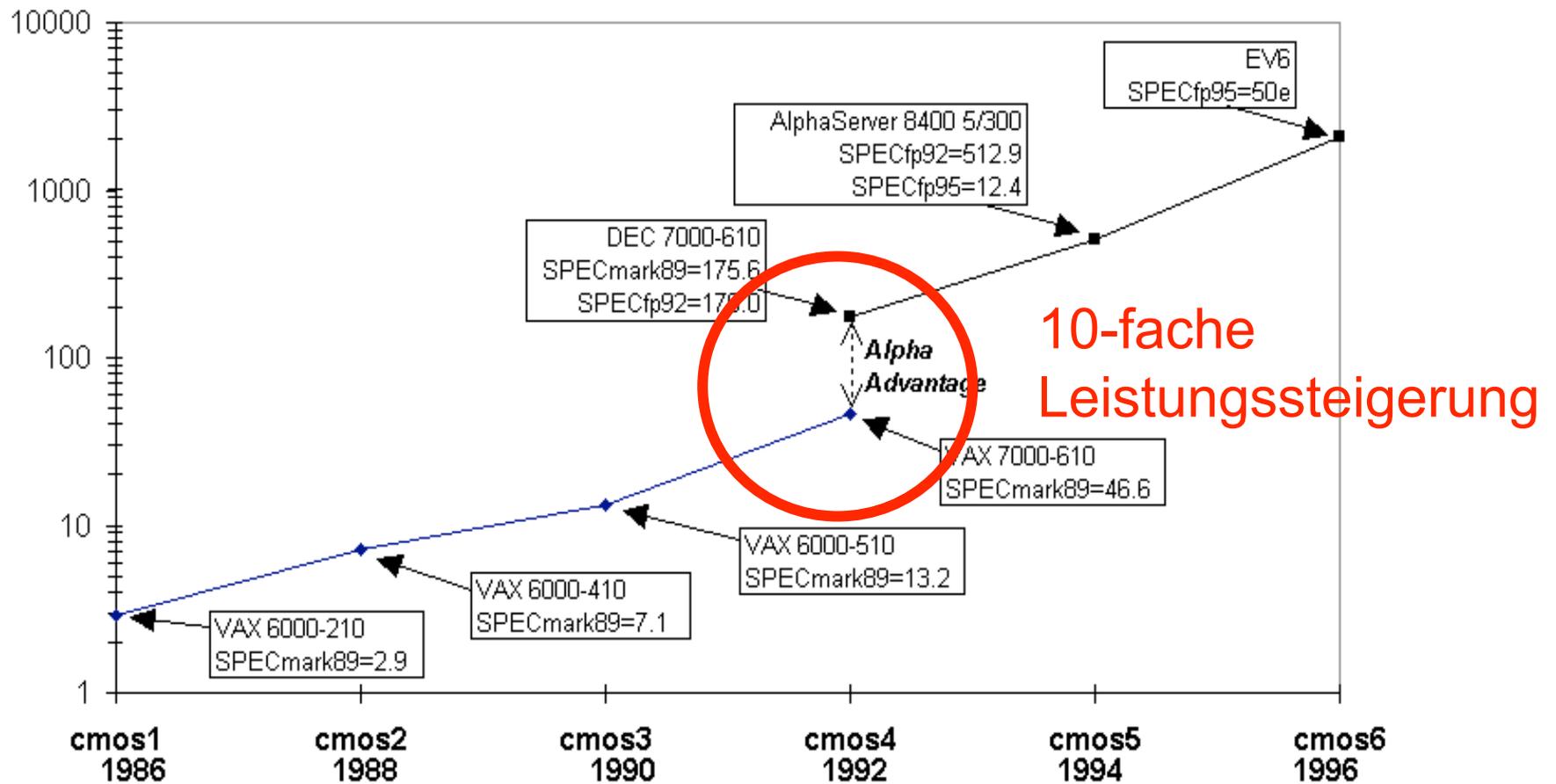
# RISC:

## Reduced Instruction Set Computer



# The CMOS Generations: Speedup through Miniaturization

Figure 2: **Relative Performance by CMOS Generation (Log Scale)**



## Was ist ein Reduced Instruction Set Computer (RISC\*) ?

\* Der Begriff RISC wurde von Carlo Sequin (UCB) geprägt

### Motivationen für die RISC-Entwicklung:

IBM 801:	Single-Cycle Instruktionen
Berkeley RISC:	Begrenzte Chipfläche
Stanford:	Überlappende Ausführung mit einfacher Kontrolle

---

**Gemeinsamer Ausgangspunkt für alle diese Forschungsprojekte war eine Analyse der Nutzung von Maschinenbefehlen und Adressierungsarten durch den Compiler. Eine Grundannahme für RISCs ist, daß die auf ihnen ablaufenden Anwendungsprogramme in einer Hochsprache (HLL) programmiert werden. Die Compiler Technologie war und ist eine der Schlüsselemente für die RISC-Technologie. Der Begriff RISC stellt tatsächlich eine "Entwurfs-Philosophie" dar, in der das Ziel höchste Leistung ist, die im Zusammenspiel von Hardware und einem optimierenden Compiler erreicht wird.**

**Im Gegensatz zu CISC-Architekturen, die versuchten, eine leistungsfähigen Assembler zu realisieren, bzw. die "Semantische Lücke" zwischen einer Hochsprache und der Architektur durch einen an die Hochsprache angepaßten Instruktionssatz zu schließen, so daß der Compiler nur noch einen minimalen Übersetzungsaufwand hat, wird in der RISC Technologie der Instruktionssatz in Hinblick auf Einfachheit und Regularität entworfen, so daß die Verwendung der Instruktionen durch den Compiler einfach und überschaubar ist.**



**Voraussetzungen: Fortschritte in der Speichertechnologie, Compilerbau**

# Was ist ein Reduced Instruction Set Computer (RISC) ?

## Anfänge der RISC - Technologie:

---

1979

IBM 801 Minicomputer

G.Mar Radin

G. Mar Radin:  
The 801 Minicomputer  
*Proceedings of the International Symp.  
on Architectural Support for Programming  
Languages and Operating Systems,*  
Palo Alto, CA, 1982



IBM RT PC, POWER, Power PC

---

1981

Berkeley RISC

Carlo Sequin , David A. Patterson

D. A. Patterson:  
Reduced Instruction Set Computers  
*Communications of the ACM,*  
28(1), 1985



SPARC

---

1982

Stanford RISC

John Hennessy

John Hennessy:  
VLSI Processor Architecture  
*IEEE Transactions on Computers,*  
C-33(11), 1984



MIPS



# Meilensteine der RISC Entwicklung

- ✍ 1977: IBM 801 Architektur
  - ✍ 1982: Berkeley RISC Prozessordesign (Patterson et al.)
  - ✍ 1984: Stanford MIPS Prozessordesign (Hennessy et al.)
  - ✍ 1985: Acorn RISC (später *ARM = Advanced RISC Machines*)
  - ✍ 1988: Motorola 88x00
  - ✍ 1989: Intel i860
  - ✍ 1990: IBM RS/6000
  - ✍ 1992: DEC Alpha (erstes 64 Bit Design)
  - ✍ 1994: Motorola/IBM/Apple PowerPC
- ✍ RISC-Prozessoren weichen vom „reinen,, RISC-Konzept ab
- ✍ höhere Anzahl von Befehlen, z.B. 184 bei PowerPC
  - ✍ RISC-CISC-Unterscheidung durch Entwurfsprinzipien gegeben



## Was ist ein Reduced Instruction Set Computer (RISC) ?

D. Tabak: *RISC-Architecture* , John Wiley & Sons, 1987

<b>Anzahl der Instruktionen</b>	$\leq 50$	<b>Restriktionen</b>
<b>Anzahl der Adr. Modi</b>	$\leq 4$	
<b>Anzahl der Befehlsformate</b>	$\leq 4$	
<hr/>		
<b>LOAD/STORE Architektur</b>		<b>Programmiermodell Reg.-Reg. Architektur</b>
<b>Anzahl der Register</b>	$\geq 32$	
<hr/>		
<b>Single Cycle Instruktionen</b>		<b>Implementierungs- Randbedingungen</b>
<b>Festverdrahtete Maschinenbefehle</b>		
<hr/>		
<b>HLL / Optimierende Compiler</b>		<b>Programmierung/ Progr. Umgebung</b>

D. Tabak: min. 5 der Bedingungen müssen erfüllt sein



## Was ist ein Reduced Instruction Set Computer (RISC) ?

**Zentrale Problemstellung: Wie kann man eine Architektur entwerfen, die als Ziel hat, Instruktionen in einem Maschinenzyklus auszuführen.**

**Register/Register Befehle**

**Festverdrahtete Maschinenbefehle**

**viele General-Purpose Register  
LOAD/STORE Architektur**

**weniger Befehle  
einfachere Befehle  
einfachere Adressierungsmodi**

**Eine wesentliche Voraussetzung für Single Cycle Befehlsausführung ist **Pipelining****



## Befehls- Statistik

Befehlsgruppe	Häufigkeit
Datenbewegung	45,3%
Programmfluss (Sprünge, Unterprogrammaufrufe)	28,7%
Arithmetische Befehle	10,8%
Vergleichsbefehle	5,9%
Logische Befehle	3,9%
Schiebebefehle	2,9%
Bitmanipulationen	2,1%
Ein-, Ausgabe, Sonstiges	0,4%

Befehlsstatistik bei populären Programmen (gcc, spice):\*

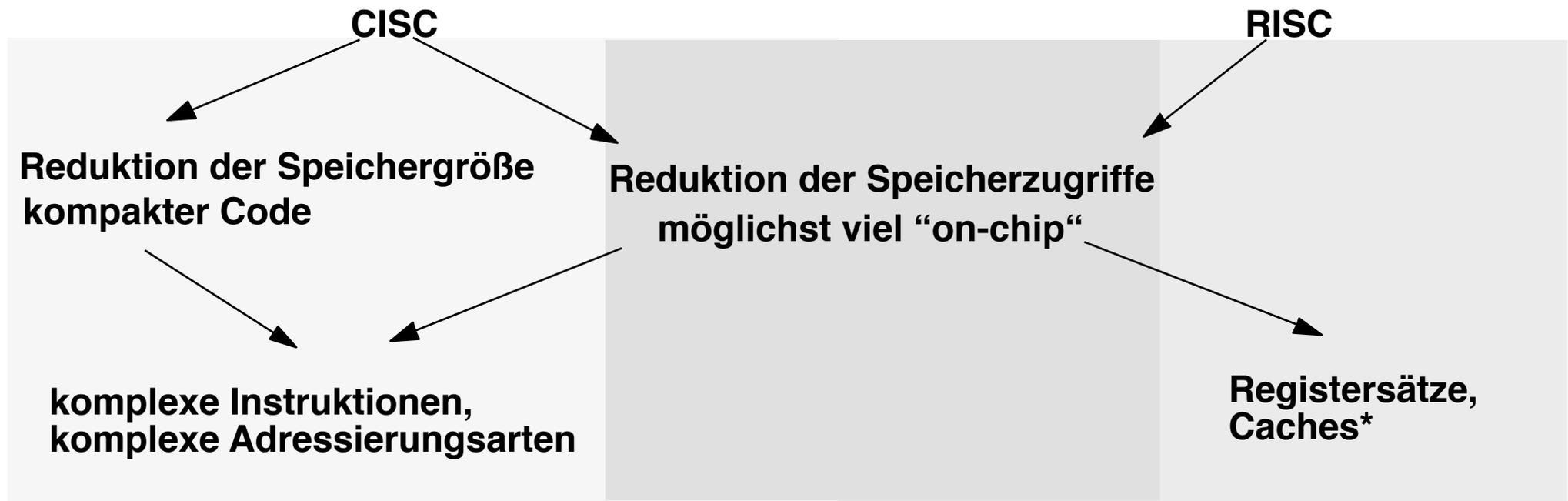
Befehlsgruppe	gcc	spice
Arithmetische Befehle	48%	50%
Datentransfer	33%	41%
Bedingte Sprünge	17%	8%
Unbedingte (long) Sprünge	2%	1%

\*Patterson, Hennessey, 1994



# Was ist ein Reduced Instruction Set Computer (RISC) ?

## Lösungsansätze zur Verringerung der Speicherzugriffe



\* insbesondere lösen Registersätze nicht das Problem, daß eine komplexe Instruktion durch eine Folge von einfacheren Instruktionen ersetzt werden muß, d.h. in der Befehlsholphase mehr Speicherzugriffe notwendig sind. Dieses Problem wird durch ein Instruktions-Cache gelöst.



## Was ist ein Reduced Instruction Set Computer (RISC) ?

### Maße für Ausführungszeiten in einem Rechner:

#### CPU Time

$$CT = TZ \cdot TD$$

**CT:** CPU - Zeit, Ausführungszeit für ein Programm

**TZ:** Gesamtzahl der Taktzyklen in einem Programm

**TD:** Dauer eines Taktzyklus

#### Clock Cycles / Instruction

$$CPI = TZ / IC$$

**CPI:** Zyklen pro Instruktion

**IC:** Anzahl der Befehle in einem Programm

$$CT = IC \cdot TD \cdot CPI$$

**CISC Ansätze versuchen die Gesamtzahl der Instruktionen (IC) zu verringern !**

**RISC Ansätze versuchen die Zyklen/Instruktion (CPI) zu verringern !**



## Was ist ein Reduced Instruction Set Computer (RISC) ?

### Folge der Realisierung einfacher Instruktionen:

#### Cons:

- **Anzahl der Instruktionen wächst**
- **Mehr Speicherplatz für Programme**
- **Größere Speicherbandbreite für den (Befehls-) Speicher**

#### Pros:

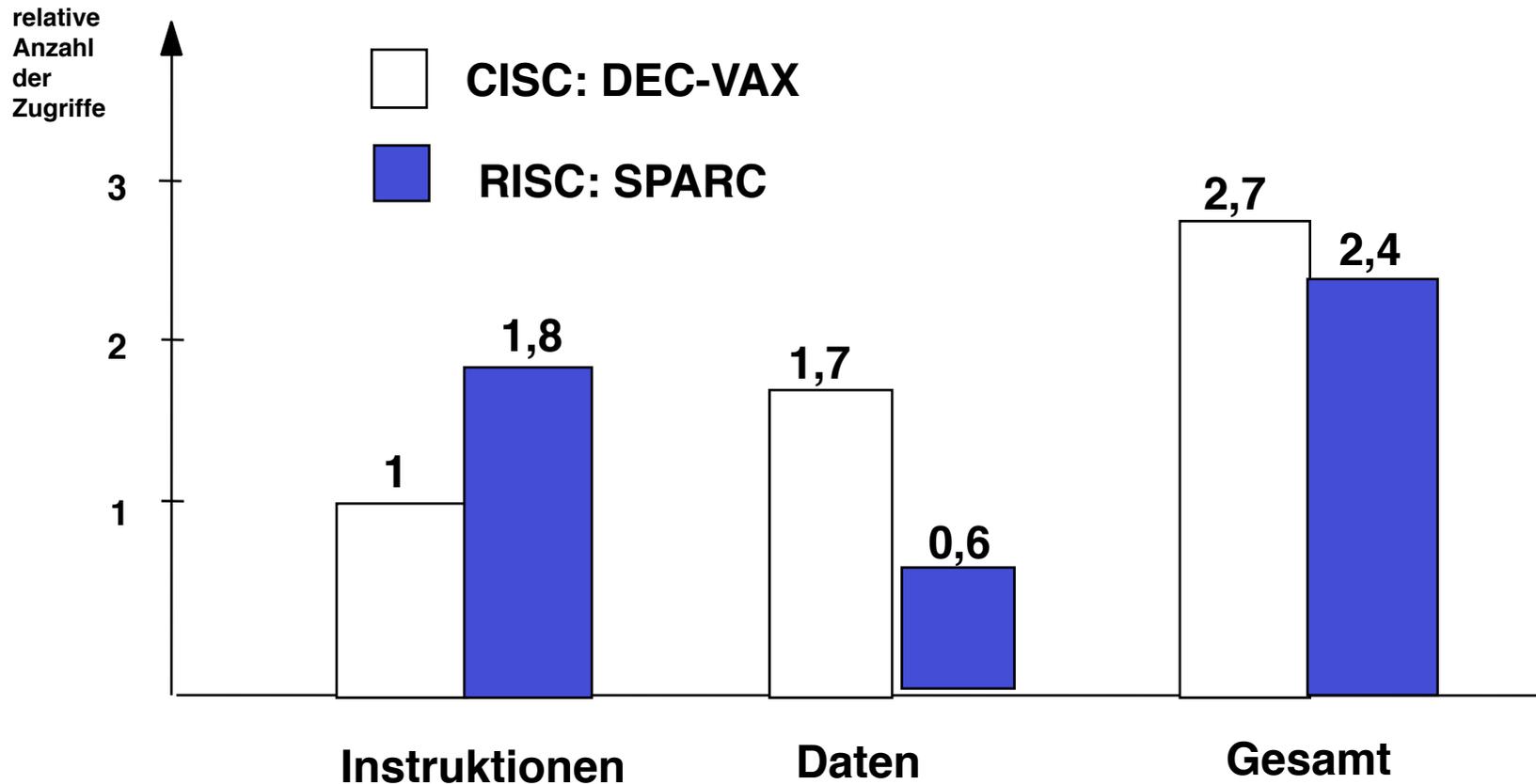
- **Einfachheit**
- **Höhere Taktraten**
- **Ausnutzung von Pipelinetechniken eher möglich**
- **Speicherbandbreite kann durch entsprechende Cachingtechniken erfolgreich erhöht werden.**
- **Höhere Speicherbandbreite ist nur für die Befehle notwendig. Da der Befehlsstrom im wesentlichen sequentiell ist (Lokalität) lassen sich Cachingtechniken besonders erfolgreich einsetzen.**



# Was ist ein Reduced Instruction Set Computer (RISC) ?

Folge der Realisierung einfacher Instruktionen:

## Vergleich der Speichernutzung (Memory Traffic) bei einem CISC und einem RISC



Alle Zugriffe sind auf die Anzahl der Zugriffe in der Befehlsholphase der VAX normiert



## Was ist ein Reduced Instruction Set Computer (RISC) ?

**Folge der Realisierung einfacher Instruktionen:**

**Vergleich 68020 (25 Mhz) und einer frühen Version des SPARC (16Mhz)**

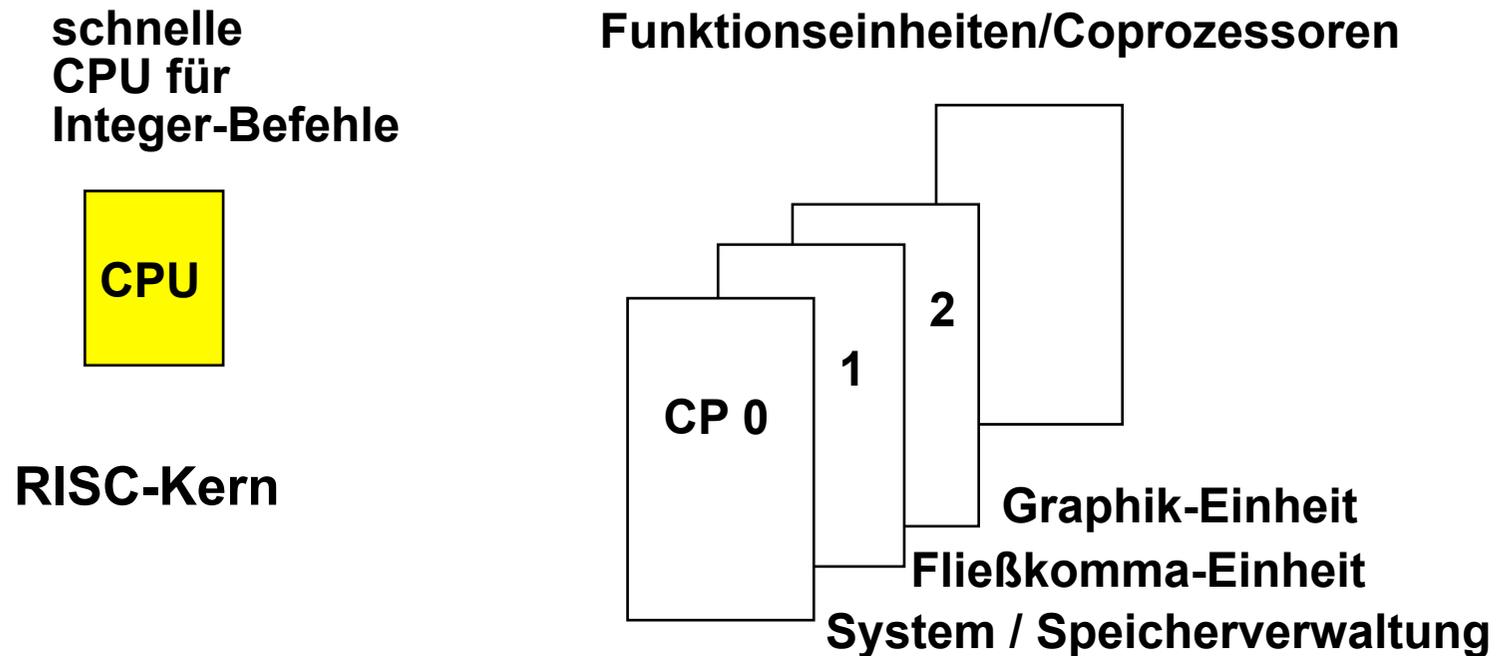
	<b>68020</b>	<b>SPARC</b>	<b>Kommentar</b>
<b>IC (Anz.d.Instr.)</b>	<b>1.0</b>	<b>1,25</b>	<b>25% mehr Instruktionen</b>
<b>TD (Taktdauer)</b>	<b>40 ns</b>	<b>60 ns</b>	<b>50% längerer Takt</b>
<b>CPI</b>	<b>5,0 - 7,0</b>	<b>1,3 - 1,7</b>	<b>!!</b>
<b>CT (CPU-Zeit)</b>	<b>2</b>	<b>1</b>	<b>doppelt so "schnell"</b>

$$CT = IC \cdot TD \cdot CPI$$



# Beispiel für einen RISC: Die MIPS Architektur

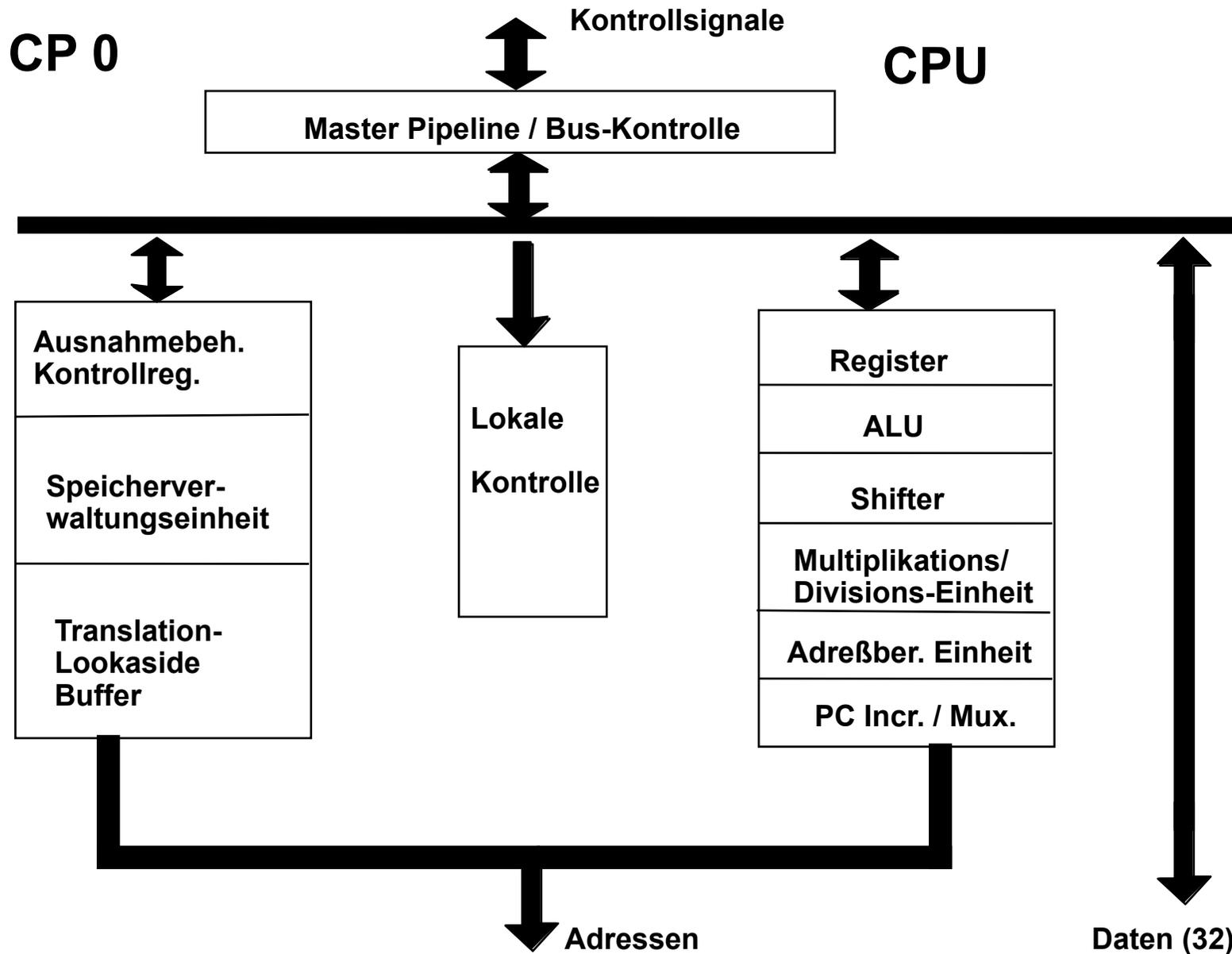
Einfacher RISC-Kern - Komplexere Operationen werden durch Coprozessoren ausgeführt, z.B. Fließkomma Instruktionen, Systemverwaltung.



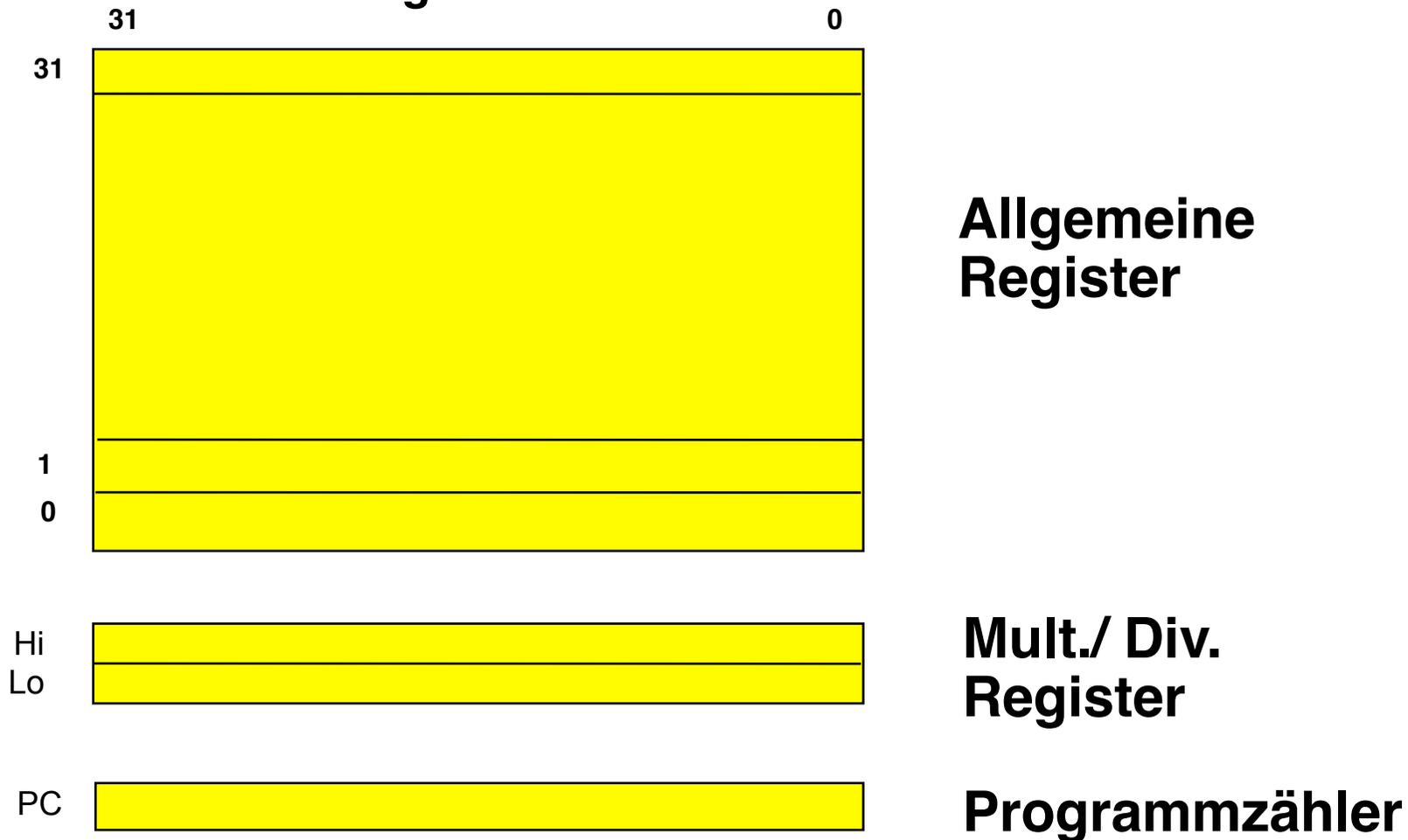
Die RISC-Technologie erfordert funktionale Dekomposition



# MIPS Architektur



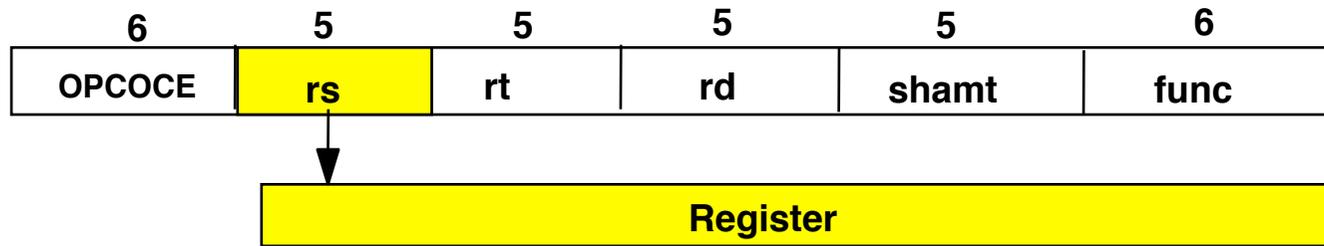
# MIPS Programmiermodell



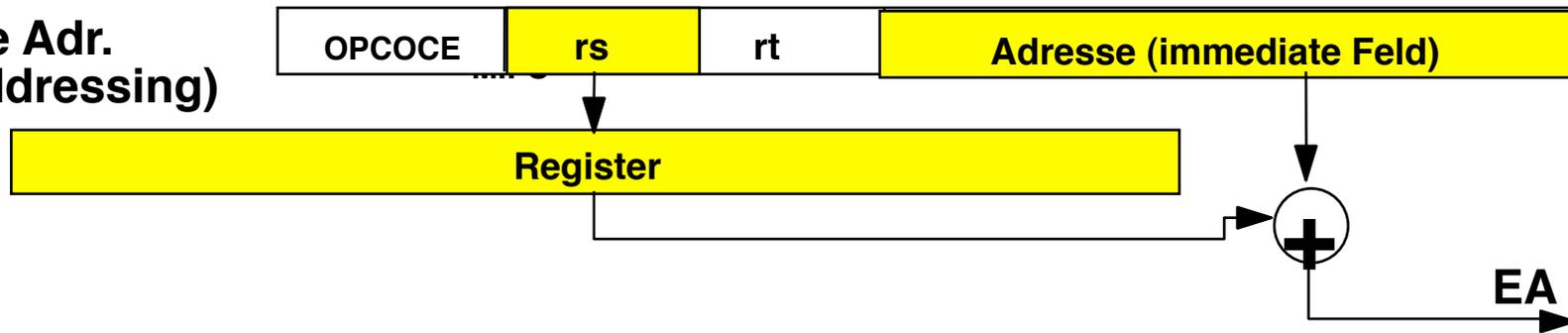
- 32 allgemeine Register
- Register 0 enthält immer den Wert 0
- Register 1 ist für den Assembler reserviert für Pseudoinstruktionen und große Konstanten
- Register Hi und Lo werden für Multiplikation und Division genutzt
- Register 31 dient zur Speicherung der Returnadresse bei Unterprogrammaufrufen (JAL)
- lineares, unsegmentiertes Speichermodell mit  $2^{30}$  32-Bit Worten

# MIPS Adressierungsarten

Register Direct



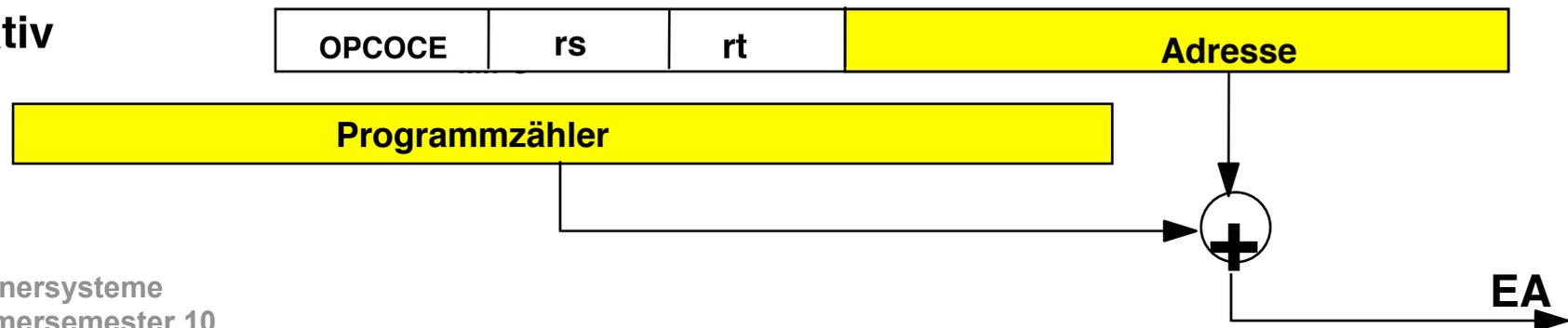
Indizierte Adr.  
(Base Addressing)



Immediate



PC-relativ



# Befehlsformate

	6 Bit	5 Bit	5 Bit	5 Bit	5 Bit	6 Bit	
Typ R	op	rs	rt	rd	shamt	func	Registerbefehle
Typ J	op	target					Sprungbefehle
Typ I	op	rs	rt	immediate			Misc.

- op            6-Bit Operationscode
- rs            5-Bit Selektor für das Quellregister
- rt            5-Bit Selektor für das Quellregister (Typ R), Zielregister oder Sprungbedingung (bei Typ I)
- rd            5-Bit Selektor für das Zielregister
- shamt        5-Bit Shift Amount für Shift von 1 – 31
- immediate   16-Bit immediate-Wert oder ein Sprungversatz oder ein Adressversatz
- funct        6-Bit Feld zur Funktionserweiterung
- target       26-Bit Zieladresse für Sprünge (wird um 2 Bit nach links verschoben(Wort Adresse) und mit den 4 höchstwertigen Bits des PCs konkateniert).



# Der MIPS-Befehlssatz

Patterson, Hennessey, 1994

I-Typ

R-Typ

J-Typ

Befehlsgruppe	Instruktion	Beispiel	Bedeutung	Kommentar
Arithmetic	add	add \$1,\$2,\$3	$\$1 = \$2 + \$3$	3 operands; exception possible
	subtract	sub \$1,\$2,\$3	$\$1 = \$2 - \$3$	3 operands; exception possible
	add immediate	addi \$1,\$2,100	$\$1 = \$2 + 100$	+ constant; exception possible
	add unsigned	addu \$1,\$2,\$3	$\$1 = \$2 + \$3$	3 operands; no exceptions
	subtract unsigned	subu \$1,\$2,\$3	$\$1 = \$2 - \$3$	3 operands; no exceptions
	add imm. unsign.	addiu \$1,\$2,100	$\$1 = \$2 + 100$	+ constant; no exceptions
	Move fr. copr. reg.	mfc0 \$1,\$epc	$\$1 = \$epc$	Used to get exception PC
	multiply	mult \$2,\$3	Hi, Lo = $\$2 \times \$3$	64-bit signed product in Hi, Lo
	multiply unsigned	multu \$2,\$3	Hi, Lo = $\$2 \times \$3$	64-bit unsigned product in Hi, Lo
	divide	div \$2,\$3	Lo = $\$2 \div \$3$ , Hi = $\$2 \bmod \$3$	Lo = quotient, Hi = remainder
	divide unsigned	divu \$2,\$3	Lo = $\$2 \div \$3$ , Hi = $\$2 \bmod \$3$	Unsigned quotient and remainder
	Move from Hi	mfhi \$1	$\$1 = \text{Hi}$	Used to get copy of Hi
	Move from Lo	mflo \$1	$\$1 = \text{Lo}$	Use to get copy of Lo
Logical	and	and \$1,\$2,\$3	$\$1 = \$2 \& \$3$	3 register operands; logical AND
	or	or \$1,\$2,\$3	$\$1 = \$2 \mid \$3$	3 register operands; logical OR
	and immediate	and \$1,\$2,100	$\$1 = \$2 \& 100$	Logical AND register, constant
	or immediate	or \$1,\$2,100	$\$1 = \$2 \mid 100$	Logical OR register, constant
	shift left logical	sll \$1,\$2,10	$\$1 = \$2 \ll 10$	Shift left by constant
shift right logical	srl \$1,\$2,10	$\$1 = \$2 \gg 10$	Shift right by constant	
Data transfer	load word	lw \$1,100(\$2)	$\$1 = \text{Memory}[\$2+100]$	Data from memory to register
	store word	sw \$1,100(\$2)	$\text{Memory}[\$2+100] = \$1$	Data from register to memory
	load upper imm.	lui \$1,100	$\$1 = 100 \times 2^{16}$	Loads constant in upper 16 bits
Conditional branch	branch on equal	beq \$1,\$2,100	if ( $\$1 == \$2$ ) go to PC+4+100	Equal test; PC relative branch
	branch on not eq.	bne \$1,\$2,100	if ( $\$1 \neq \$2$ ) go to PC+4+100	Not equal test; PC relative
	set on less than	slt \$1,\$2,\$3	if ( $\$2 < \$3$ ) $\$1=1$ ; else $\$1=0$	Compare less than; 2's complement
	set less than imm.	slti \$1,\$2,100	if ( $\$2 < 100$ ) $\$1=1$ ; else $\$1=0$	Compare < constant; 2's comp.
	set less than uns.	sltu \$1,\$2,\$3	if ( $\$2 < \$3$ ) $\$1=1$ ; else $\$1=0$	Compare less than; natural number
set l.t. imm. uns.	sltiu \$1,\$2,100	if ( $\$2 < 100$ ) $\$1=1$ ; else $\$1=0$	Compare < constant; natural	
Unconditional jump	jump	j 10000	go to 10000	Jump to target address
	jump register	jr \$31	go to \$31	For switch, procedure return
	jump and link	jal 10000	$\$31 = \text{PC} + 4$ ; go to 10000	For procedure call



## Vollständiger MIPS Maschinenbefehlssatz (R2000)

# Der MIPS-Befehlssatz

Mnemo	Typ	Formatbeispiel						Asm. beispiel
		0	2	3	1	0	32	
add	R	0	2	3	1	0	32	add \$1,\$2,\$3
sub	R	0	2	3	1	0	34	sub \$1,\$2,\$3
addi	I	8	2	1	100			addi \$1,\$2,100
addu	R	0	2	3	1	0	33	addu \$1,\$2,\$3
subu	R	0	2	3	1	0	35	subu \$1,\$2,\$3
addiu	I	9	2	1	100			addiu \$1,\$2,100
mfc0	R	16	0	1	14	0	0	mfc0 \$1,\$epc
mult	R	0	2	3	0	0	24	mult \$2,\$3
multu	R	0	2	3	0	0	25	multu \$2,\$3
div	R	0	2	3	0	0	26	div \$2,\$3
divu	R	0	2	3	0	0	27	divu \$2,\$3
mfhi	R	0	0	0	1	0	16	mfhi \$1
mflo	R	0	0	0	1	0	18	mflo \$1
and	R	0	2	3	1	0	36	and \$1,\$2,\$3
or	R	0	2	3	1	0	37	or \$1,\$2,\$3
andi	I	12	2	1	100			andi \$1,\$2,100
ori	I	13	2	1	100			ori \$1,\$2,100
sll	R	0	0	2	1	10	0	sll \$1,\$2,10
srl	R	0	0	2	1	10	2	srl \$1,\$2,10
lw	I	35	2	1	100			lw \$1,100(\$2)
sw	I	43	2	1	100			sw \$1,100(\$2)
lui	I	15	0	1	100			lui \$1,100
beq	I	4	1	2	100			beq \$1,\$2,100
bne	I	5	1	2	100			bne \$1,\$2,100
slt	R	0	2	3	1	0	42	slt \$1,\$2,\$3
slti	I	10	2	1	100			slti \$1,\$2,100
sltu	R	0	2	3	1	0	43	sltu \$1,\$2,\$3
sltiu	I	11	2	1	100			sltiu \$1,\$2,100
j	J	2	10000					j 10000
jr	R	0	31	0	0	0	8	jr \$1
jal	J	3	10000					jal 10000



## Vollständiger MIPS Maschinenbefehlssatz (R2000)

# Der MIPS- Befehlssatz

Mnemo	Typ	Formatbeispiel						Asm. beispiel
add	R	0	2	3	1	0	32	add \$1,\$2,\$3
sub	R	0	2	3	1	0	34	sub \$1,\$2,\$3
addi	I	8	2	1		100		addi \$1,\$2,100
addu	R	0	2	3	1	0	33	addu \$1,\$2,\$3
subu	R	0	2	3	1	0	35	subu \$1,\$2,\$3
addiu	I	9	2	1		100		addiu \$1,\$2,100
mfc0	R	16	0	1	14	0	0	mfc0 \$1,\$epc
mult	R	0	2	3	0	0	24	mult \$2,\$3
multu	R	0	2	3	0	0	25	multu \$2,\$3
div	R	0	2	3	0	0	26	div \$2,\$3
divu	R	0	2	3	0	0	27	divu \$2,\$3
mfhi	R	0	0	0	1	0	16	mfhi \$1
mflo	R	0	0	0	1	0	18	mflo \$1
and	R	0	2	3	1	0	36	and \$1,\$2,\$3
or	R	0	2	3	1	0	37	or \$1,\$2,\$3
andi	I	12	2	1		100		andi \$1,\$2,100
ori	I	13	2	1		100		ori \$1,\$2,100
sll	R	0	0	2	1	10	0	sll \$1,\$2,10
srl	R	0	0	2	1	10	2	srl \$1,\$2,10
lw	I	35	2	1		100		lw \$1,100(\$2)
sw	I	43	2	1		100		sw \$1,100(\$2)
lui	I	15	0	1		100		lui \$1,100
beq	I	4	1	2		100		beq \$1,\$2,100
bne	I	5	1	2		100		bne \$1,\$2,100
slt	R	0	2	3	1	0	42	slt \$1,\$2,\$3
slti	I	10	2	1		100		slti \$1,\$2,100
sltu	R	0	2	3	1	0	43	sltu \$1,\$2,\$3
sltiu	I	11	2	1		100		sltiu \$1,\$2,100
j	J	2				10000		j 10000
jr	R	0	31	0	0	0	8	jr \$1
jal	J	3				10000		jal 10000



# Der MIPS- Befehlssatz

## Die MIPS Assemblernotation (ohne Fließkommabefehle)

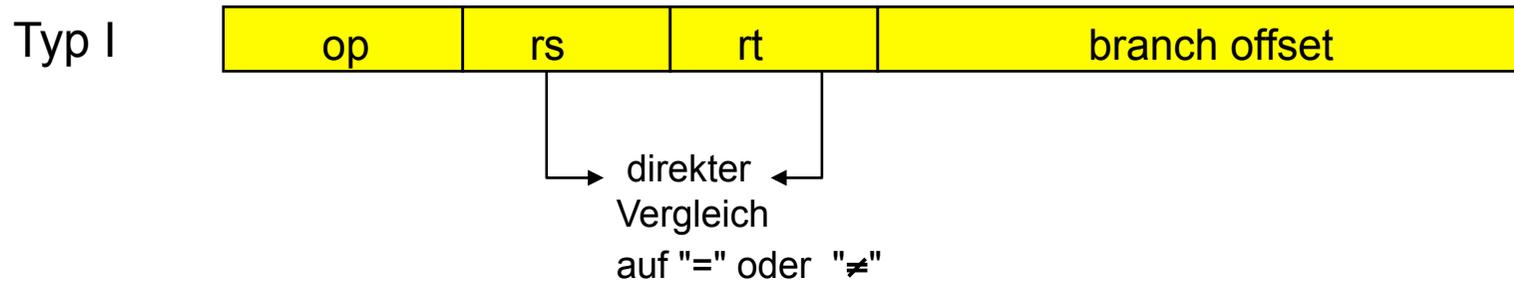
	<b>Load/Store Instructions</b>		<b>Multiply/Divide Instructions</b>
LB	Load Byte	MULT	Multiply
LBU	Load Byte Unsigned	MULTU	Multiply Unsigned
LH	Load Halfword	DIV	Divide
LHU	Load Halfword Unsigned	DIVU	Divide Unsigned
LW	Load Word	MFHI	Move From HI
LWL	Load Word Left	MTHI	Move To HI
LWR	Load Word Right	MFLO	Move From LO
SB	Store Byte	MTLO	Move To LO
SH	Store Halfword		<b>Jump and Branch Instructions</b>
SW	Store Word	J	Jump
SWL	Store Word Left	JAL	Jump And Link
SWR	Store Word Right	JR	Jump to Register
	<b>Arithmetic Instructions (ALU Immediate)</b>	JALR	Jump And Link Register
ADDI	Add Immediate	BEQ	Branch on Equal
ADDIU	Add Immediate Unsigned	BNE	Branch on Not Equal
SLTI	Set on Less Than Immediate	BLEZ	Branch on Less than or Equal to Zero
SLTIU	Set on Less Than Immediate Unsigned	BGTZ	Branch on Greater Than Zero
ANDI	AND Immediate	BLTZ	Branch on Less Than Zero
ORI	OR Immediate	BGEZ	Branch on Greater than or Equal to Zero
XORI	Exclusive OR Immediate	BLTZAL	Branch on Less Than Zero And Link
LUI	Load Upper Immediate	BGEZAL	Branch on Greater than or Equal to Zero And Link
	<b>Arithmetic Instructions (3-operand, R-type)</b>		<b>Coprocessor Instructions</b>
ADD	Add	LWCz	Load Word to Coprocessor
ADDU	Add Unsigned	SWCz	Store Word from Coprocessor
SUB	Subtract	MTCz	Move To Coprocessor
SUBU	Subtract Unsigned	MFCz	Move From Coprocessor
SLT	Set on Less Than	CTCz	Move Control to Coprocessor
SLTU	Set on Less Than Unsigned	CFCz	Move Control From Coprocessor
AND	AND	COPz	Coprocessor Operation
OR	OR	BCzT	Branch on Coprocessor z True
XOR	Exclusive OR	BCzF	Branch on Coprocessor z False
NOR	NOR		<b>Special Instructions</b>
	<b>Shift Instructions</b>	SYSCALL	System Call
SLL	Shift Left Logical	BREAK	Break
SRL	Shift Right Logical		
SRA	Shift Right Arithmetic		
SLLV	Shift Left Logical Variable		
SRLV	Shift Right Logical Variable		
SRAV	Shift Right Arithmetic Variable		



## Der MIPS- Befehlssatz: Bedingte Sprungbefehle

Der Befehlssatz hat nur zwei bedingte Sprünge, die Operanden direkt vergleichen:

- Branch on equal
- Branch on not equal



Einzigster Vergleichsbefehl ist slt: set on less than



slt: if  $rs < rt$  then  $rd \leftarrow 1$  else  $rd \leftarrow 0$



## Der MIPS- Befehlssatz: Bedingte Sprungbefehle

Beispiel: branch on a>b

Da "branch on a>b " nicht direkt ausgeführt werden kann, muß man in 2 Schritten vorgehen:

1. Vergleiche und setze Bedingung (da nur a<b geprüft werden kann, vertausche a und b).
2. Werte Bedingung für den Sprung aus.

slt \$1, \$b, \$a  
bne \$1, \$0, <offset>

Vergleiche  $b < a$  und setze \$1 auf 1 wenn Bedingung zutrifft (d.h.  $b < a$ )  
Wenn  $\$1 \neq \$0$  (das immer 0 ist!) gilt, führe Sprung aus.



# Der MIPS- Befehlssatz

Die MIPS ALU erzeugt keine Condition Codes, d.h. der Status einer ALU-Operation wird nicht gespeichert.

Man kann z.B. nach einer Addition nicht feststellen, ob ein Carry aufgetreten ist, oder ob ein negatives, positives Ergebnis vorliegt.

Erkennung eines Carry:

Dazu führt man folgende Codesequenz aus:

```
addu    $temp, $a, $b          /* addiere (unsigned) $a + $b, Ergebnis nach $temp
sltu    $carry, $temp, $b      /* wenn temp nach einer Addition kleiner ist als b
                                           /* ist ein carry aufgetreten
```

Beachten: die Assemblernotation hat die Registerreihenfolge: rd, rs,rt !



## Der MIPS- Befehlssatz

**Test auf Overflow: MIPS erzeugt bei arithmetischen Operationen mit Vorzeichen eine Ausnahme, wenn ein Overflow auftritt. Im sogenannten "Cause-Register" wird dann angezeigt, welche Bedingung die Ausnahme ausgelöst hat. Soll keine Ausnahmebehandlung angestoßen werden, muß die Addition ohne Vorzeichen (unsigned add: addu) und folgende Codesequenz benutzt werden.**

**Berechne:  $c = a + b$ , springe nach  $m$  bei Überlauf**

Überlauf tritt nur auf, wenn

1. die Vorzeichen beider Summanden gleich sind, sich
2. aber das Vorzeichen der Summe davon unterscheidet

addu	\$c, \$a, \$b	<i>/* berechne die Summe ohne Vorzeichen */</i>
xor	\$v, \$a, \$b	<i>/* wenn a und b ungleiche Vorzeichen haben, kann kein Overflow auftreten !</i>
blzt	\$v, k	<i>/* a und b haben dasselbe Vorzeichen */</i>
xor	\$e, \$c, \$a	<i>/* teste ob die Summe dasselbe Vorz. wie einer der Summanden hat */</i>
blzt	\$e, m	<i>/* Addition erzeugt einen Überlauf ! */</i>
k	.....	<i>/* kein Overflow</i>
	.....	
m	.....	<i>/* Behandlung des Überlaufs</i>



## Zusammenfassung MIPS:

- Trennung von RISC-Core und Spezial-Coprozessoren
- Einfache, orthogonale Architektur
  - großer Registersatz, wenige Spezialregister
  - 1 Befehlslänge, 3 Befehlsformate
  - wenige Adressierungsarten
    - Hauptadressierungsart für Speicherzugriffe: indiziert
    - alle arithmetischen Operationen Reg-Reg
  - explizites Laden und Speichern von Operanden aus dem Hauptspeicher (Load/Store-Architektur)
- Sehr einfache Architektur erfordert die Realisierung vieler "Standard"-Funktionen durch Software, d.h. anheben dieser Funktionen von der MP-Ebene auf SW-Ebene.

