Befehlssatz MIPS-R2000

0 Notation

RI	$R_t \mid imm$	wahlweise Register R_t oder Direktoperand imm
imm		16-Bit Direktoperand, Wert:
	$[symbol]$ $[\pm dist]$	$symbol + dist$
	$(dist)$ >> int	$dist/2^{int}$
$dist$	int_1 $[\pm int_2]$	Distanzangabe $int_1 + int_2$
$addr$	$[symbol]$ $[\pm dist]$ $[(R_s)]$	Adressangabe für Speicherstelle $symbol + dist + R_s$
$C(x)_n$		n Bytes großer Wert an der Speicherstelle x
$X_{[b_1 \ldots b_n]}$		Bits b_1 bis b_n des Wertes X
$label$	$symbol$	Symbolischer Name des Sprungziels: bei absoluten Sprüngen (`j*`) 26-Bit Instruktionsadresse, bei relativen Sprüngen (`b*`) eine 16-Bit Distanzangabe

⋆	Hardware-Instruktion des Prozessors
∗	*delayed*-Instruktion; Ausführung endet nach dem nächsten Befehl
◇	Vorzeichen wird ignoriert
†	Instruktion kann Exception auslösen
‡	Pseudo-Instruktion löst beim Fehlerfall Exceptions durch `break` aus

1 Registerbelegung

\$0	*\$zero*	constant 0	Konstante 0
\$1	*\$at*	assembler	reserviert für den Assembler
\$2–\$3	*\$v0–\$v1*	value	Ausdruck- bzw. Funktionsergebnis
\$4–\$7	*\$a0–\$a3*	argument	Funktionsargument
\$8–\$15	*\$t0–\$t7*	temporary	frei, wird bei Funktionsaufrufen überschrieben
\$16–\$23	*\$s0–\$s7*	saved temporary	frei, wird bei Funktionsaufrufen gerettet
\$24–\$25	*\$t8–\$t9*	temporary	frei, wird bei Funktionsaufrufen überschrieben
\$26–\$27	*\$k0–\$k1*	kernel	reserviert für Betriebssystem
\$28	*\$gp*	global pointer	Zeiger auf globalen Datenbereich
\$29	*\$sp*	stack pointer	Stack-Pointer
\$30	*\$fp*	frame pointer	Zeiger auf lokalen Datenbereich einer Funktion
\$31	*\$ra*	return address	Rücksprungadresse

2 Ganzzahlarithmetik

`abs`		R_d, R_s	absolute value	$R_d \leftarrow \lvert R_s \rvert$
`neg`	◇ `negu`	R_d, R_s	negate value	$R_d \leftarrow -R_s$
⋆† `add`	⋆◇ `addu`	R_d, R_s, RI	addition	$R_d \leftarrow R_s + RI$
⋆† `addi`	⋆◇ `addiu`	R_d, R_s, imm	addition immediate	$R_d \leftarrow R_s + imm$
⋆† `sub`	⋆◇ `subu`	R_d, R_s, RI	subtract	$R_d \leftarrow R_s - RI$
⋆ `mult`	⋆◇ `multu`	R_s, R_t	multiply	$(hi, lo) \leftarrow R_s \times R_t$
`mul`		R_d, R_s, RI	multiply	$R_d \leftarrow R_s \times RI$
‡ `mulo`	‡ `mulou`	R_d, R_s, RI	multiply with overflow	$R_d \leftarrow R_s \times RI$
⋆ `div`	⋆◇ `divu`	R_s, R_t	divide	$lo \leftarrow R_s/R_t, hi \leftarrow R_s \bmod R_t$
‡ `div`	◇‡ `divu`	R_d, R_s, RI	divide	$R_d \leftarrow R_s/RI$
‡ `rem`	◇‡ `remu`	R_d, R_s, RI	remainder	$R_d \leftarrow R_s \bmod RI$

3 Ganzzahlvergleiche

`seq`		R_d, R_s, RI	set equal	$R_d \leftarrow R_s = RI\ ?\ 1 : 0$
`sne`		R_d, R_s, RI	set not equal	$R_d \leftarrow R_s \neq RI\ ?\ 1 : 0$
`sgt`	◇ `sgtu`	R_d, R_s, RI	set greater than	$R_d \leftarrow R_s > RI\ ?\ 1 : 0$
`sge`	◇ `sgeu`	R_d, R_s, RI	set greater than equal	$R_d \leftarrow R_s \geq RI\ ?\ 1 : 0$
⋆ `slt`	⋆◇ `sltu`	R_d, R_s, RI	set less than	$R_d \leftarrow R_s < RI\ ?\ 1 : 0$
⋆ `slti`	⋆◇ `sltiu`	R_d, R_s, imm	set less than immediate	$R_d \leftarrow R_s < imm\ ?\ 1 : 0$
`sle`	◇ `sleu`	R_d, R_s, RI	set less than equal	$R_d \leftarrow R_s \leq RI\ ?\ 1 : 0$

4 Logische Operationen

`not`	R_d, R_s	bitwise logical negation	$R_d \leftarrow \neg R_s$
⋆ `and`	R_d, R_s, RI	bitwise AND	$R_d \leftarrow R_s \wedge RI$
⋆ `andi`	R_d, R_s, imm	bitwise AND immediate	$R_d \leftarrow R_s \wedge imm$
⋆ `or`	R_d, R_s, RI	bitwise OR	$R_d \leftarrow R_s \vee RI$
⋆ `ori`	R_d, R_s, imm	bitwise OR immediate	$R_d \leftarrow R_s \vee imm$
⋆ `xor`	R_d, R_s, RI	bitwise XOR	$R_d \leftarrow R_s \oplus RI$
⋆ `xori`	R_d, R_s, imm	bitwise XOR immediate	$R_d \leftarrow R_s \oplus imm$
⋆ `nor`	R_d, R_s, RI	bitwise NOR	$R_d \leftarrow \neg(R_s \vee RI)$

5 Shift-Operationen

⋆◇ `sll`	R_d, R_t, imm	shift left logical	$R_d \leftarrow R_t * 2^{imm \bmod 32}$
⋆◇ `sllv`	R_d, R_t, R_v	shift left logical variable	$R_d \leftarrow R_t * 2^{R_v \bmod 32}$
⋆◇ `srl`	R_d, R_t, imm	shift right logical	$R_d \leftarrow R_t/2^{imm \bmod 32}$
⋆◇ `srlv`	R_d, R_t, R_v	shift right logical variable	$R_d \leftarrow R_t/2^{R_v \bmod 32}$
⋆ `sra`	R_d, R_t, imm	shift right arithmetic	$R_d \leftarrow R_t/2^{imm \bmod 32}$
⋆ `srav`	R_d, R_t, R_v	shift right arithmetic variable	$R_d \leftarrow R_t/2^{R_v \bmod 32}$
◇ `rol`	R_d, R_t, RI	rotate left	$R_d \leftarrow R_{t[((31-RI) \bmod 32) \ldots 0, 31 \ldots ((31-RI-1) \bmod 32)]}$
◇ `ror`	R_d, R_t, RI	rotate right	$R_d \leftarrow R_{t[((RI-1) \bmod 32) \ldots 0, 31 \ldots (RI \bmod 32)]}$

6 Unbedingte Sprünge

`b`	$label$	branch instruction	Sprung nach $label$
⋆ `j`	$label$	jump	Sprung nach $label$
⋆ `jr`	R_s	jump register	Sprung nach R_s
⋆ `jal`	$label$	jump and link	$\$31 \leftarrow IP + 4$, Sprung nach $label$
`bal`	$label$	branch and link	$\$31 \leftarrow IP + 4$, Sprung nach $label$
⋆ `jalr`	R_d, R_s	jump and link register	$R_d \leftarrow IP + 4$, Sprung nach R_s

7 Bedingte Sprünge nach Vergleich

⋆∗ `bc1f`		$label$	branch coprocessor 1 false	Sprung nach $label$, wenn $CF_1 = 0$
⋆∗ `bc1t`		$label$	branch coprocessor 1 true	Sprung nach $label$, wenn $CF_1 = 1$
⋆∗ `beq`		$R_s, RI, label$	branch on equal	Sprung nach $label$, wenn $R_s = RI$
⋆∗ `bne`		$R_s, RI, label$	branch on not equal	Sprung nach $label$, wenn $R_s \neq RI$
`bgt`	◇ `bgtu`	$R_s, RI, label$	branch on greater than	Sprung nach $label$, wenn $R_s > RI$
`bge`	◇ `bgeu`	$R_s, RI, label$	branch on greater than equal	Sprung nach $label$, wenn $R_s \geq RI$
`blt`	◇ `bltu`	$R_s, RI, label$	branch on less than	Sprung nach $label$, wenn $R_s < RI$
`ble`	◇ `bleu`	$R_s, RI, label$	branch on less than equal	Sprung nach $label$, wenn $R_s \leq RI$

8 Bedingte Sprünge nach Vergleich mit 0

`beqz`	R_s, *label*	branch on equal zero	Sprung nach *label*, wenn $R_s = 0$
`bnez`	R_s, *label*	branch on not equal zero	Sprung nach *label*, wenn $R_s \neq 0$
⋆∗`bgtz`	R_s, *label*	branch on greater than zero	Sprung nach *label*, wenn $R_s > 0$
⋆∗`bgez`	R_s, *label*	branch on greater than equal zero	Sprung nach *label*, wenn $R_s \geq 0$
⋆∗`bgezal`	R_s, *label*	branch on greater than equal zero and link	$\$31 \leftarrow IP + 4$, Sprung nach *label*, wenn $R_s \geq 0$
⋆∗`bltz`	R_s, *label*	branch on less than zero	Sprung nach *label*, wenn $R_s < 0$
⋆∗`bltzal`	R_s, *label*	branch on less than and link	$\$31 \leftarrow IP + 4$, Sprung nach *label*, wenn $R_s < 0$
⋆∗`blez`	R_s, *label*	branch on less than equal zero	Sprung nach *label*, wenn $R_s \leq 0$

9 Registertransfer

`move`	R_d, R_s	move register	$R_d \leftarrow R_s$
⋆`mfhi`	R_d	move from hi	$R_d \leftarrow hi$
⋆`mflo`	R_d	move from lo	$R_d \leftarrow lo$
⋆`mthi`	R_d	move to hi	$hi \leftarrow R_d$
⋆`mtlo`	R_d	move to lo	$lo \leftarrow R_d$
⋆`mfc1`	R_d, F_s	move from coprocessor 1	$R_d \leftarrow F_s$
⋆`mtc1`	R_d, F_s	move to coprocessor 1	$F_s \leftarrow R_d$
`mfc1.d`	R_d, F_s	move double from coprocessor 1	$(R_d, R_{d+1}) \leftarrow (F_s, F_{s+1})$
`mov.s`	F_d, F_s	move floating-point single	$F_d \leftarrow F_s$
`mov.d`	F_d, F_s	move floating-point double	$(F_d, F_{d+1}) \leftarrow (F_s, F_{s+1})$

10 Ladebefehle

`la`	R_d, *addr*	load address	$R_d \leftarrow addr$
⋆∗ `lb`	R_d, *addr*	load byte	$R_d \leftarrow C(addr)_1$
⋆∗◇`lbu`	R_d, *addr*	load byte unsigned	$R_d \leftarrow C(addr)_1$
⋆∗ `lh`	R_d, *addr*	load halfword	$R_d \leftarrow C(addr)_2$
⋆∗◇`lhu`	R_d, *addr*	load halfword unsigned	$R_d \leftarrow C(addr)_2$
⋆∗ `lw`	R_d, *addr*	load word	$R_d \leftarrow C(addr)_4$
⋆∗◇`lwl`	R_d, *addr*	load word left	$R_{d[31\ldots(addr \bmod 4)*8]} \leftarrow C(addr)_{addr \bmod 4}$
⋆∗◇`lwr`	R_d, *addr*	load word right	$R_{d[(addr \bmod 4)*8+7\ldots0]} \leftarrow C(addr)_{addr \bmod 4}$
`ld`	R_d, *addr*	load double-word	$(R_d, R_{d+1}) \leftarrow C(addr)_8$
⋆∗ `lwc1`	F_d, *addr*	load word coprocessor 1	$F_d \leftarrow C(addr)_4$
`l.s`	F_d, *addr*	load floating-point single	$F_d \leftarrow C(addr)_4$
`l.d`	F_d, *addr*	load floating-point double	$(F_{d+1}, F_d) \leftarrow C(addr)_8$
`li`	R_d, *imm*	load immediate	$R_d \leftarrow imm$
⋆ `lui`	R_d, *imm*	load upper immediate	$R_d \leftarrow imm * 2^{16}$
`li.s`	F_d, *imm*	load immediate floating-point single	$F_d \leftarrow imm$
`li.d`	F_d, *imm*	load immediate floating-point double	$(F_{d+1}, F_d) \leftarrow imm$
`ulh`	R_d, *addr*	unaligned load halfword	$R_d \leftarrow C(addr)_2$
◇ `ulhu`	R_d, *addr*	unaligned load halfword unsigned	$R_d \leftarrow C(addr)_2$

11 Speicherbefehle

⋆`sb`	R_t, *addr*	store byte	$C(addr)_1 \leftarrow R_{t[7\ldots0]}$
⋆`sh`	R_t, *addr*	store halfword	$C(addr)_2 \leftarrow R_{t[15\ldots0]}$
⋆`sw`	R_t, *addr*	store word	$C(addr)_4 \leftarrow R_t$
⋆`swl`	R_t, *addr*	store word left	$C(addr)_{4-addr \bmod 4} \leftarrow R_{t[31\ldots(addr \bmod 4)*8]}$
⋆`swr`	R_t, *addr*	store word right	$C(addr + 4 - addr \bmod 4)_{addr \bmod 4} \leftarrow R_{t[(addr \bmod 4)*8-1\ldots0]}$
`sd`	R_t, *addr*	store double-word	$C(addr)_8 \leftarrow (R_t, R_{t+1})$
⋆`swc1`	F_d, *addr*	store word coprocessor 1	$C(addr)_4 \leftarrow F_d$
`s.s`	F_d, *addr*	store floating-point single	$C(addr)_4 \leftarrow F_d$
`s.d`	F_d, *addr*	store floating-point double	$C(addr)_8 \leftarrow (F_{d+1}, F_d)$
`ush`	R_t, *addr*	unaligned store halfword	$C(addr)_2 \leftarrow R_{t[15\ldots0]}$
`usw`	R_t, *addr*	unaligned store word	$C(addr)_4 \leftarrow R_t$

12 Gleitkomma-Arithmetik

⋆`abs.s`	⋆`abs.d`	F_d, F_s	absolute value	$F_d \leftarrow \lvert F_s \rvert$
⋆`add.s`	⋆`add.d`	F_d, F_s, F_t	addition	$F_d \leftarrow F_s + F_t$
⋆`sub.s`	⋆`sub.d`	F_d, F_s, F_t	subtract	$F_d \leftarrow F_s - F_t$
⋆`mul.s`	⋆`mul.d`	F_d, F_s, F_t	multiply	$F_d \leftarrow F_s \times F_t$
⋆`div.s`	⋆`div.d`	F_d, F_s, F_t	divide	$F_d \leftarrow F_s / F_t$

13 Gleitkomma-Vergleiche

⋆ `c.un.s`	⋆ `c.un.d`	F_s, F_t	compare unordered	$CF_1 \leftarrow (F_s = \mathrm{NaN}) \vee (F_s = \mathrm{NaN})$
⋆ `c.eq.s`	⋆ `c.eq.d`	F_s, F_t	compare equal	$CF_1 \leftarrow F_s = F_t$
⋆†`c.seq.s`	⋆†`c.seq.d`	F_s, F_t	compare equal	$CF_1 \leftarrow F_s = F_t$
⋆ `c.ueq.s`	⋆ `c.ueq.d`	F_s, F_t	compare unordered equal	$CF_1 \leftarrow (F_s = F_t) \vee (F_s = \mathrm{NaN}) \vee (F_s = \mathrm{NaN})$
⋆†`c.lt.s`	⋆†`c.lt.d`	F_s, F_t	compare less than	$CF_1 \leftarrow F_s < F_t$
⋆ `c.olt.s`	⋆ `c.olt.d`	F_s, F_t	compare ordered less than	$CF_1 \leftarrow F_s < F_t$
⋆ `c.ult.s`	⋆ `c.ult.d`	F_s, F_t	compare unordered less than	$CF_1 \leftarrow (F_s < F_t) \vee (F_s = \mathrm{NaN}) \vee (F_s = \mathrm{NaN})$
⋆†`c.le.s`	⋆†`c.le.d`	F_s, F_t	compare less than equal	$CF_1 \leftarrow F_s \leq F_t$
⋆ `c.ole.s`	⋆ `c.ole.d`	F_s, F_t	compare ordered less than equal	$CF_1 \leftarrow F_s \leq F_t$
⋆ `c.ule.s`	⋆ `c.ule.d`	F_s, F_t	compare unordered less than equal	$CF_1 \leftarrow (F_s \leq F_t) \vee (F_s = \mathrm{NaN}) \vee (F_s = \mathrm{NaN})$

14 Gleitkomma-Konversion

⋆`cvt.d.s`	F_d, F_s	convert single to double	$F_d \leftarrow [D]F_s$
⋆`cvt.d.w`	F_d, F_s	convert integer to double	$F_d \leftarrow [D]F_s$
⋆`cvt.w.d`	F_d, F_s	convert double to integer	$F_d \leftarrow [I]F_s$
⋆`cvt.w.s`	F_d, F_s	convert single to integer	$F_d \leftarrow [I]F_s$
⋆`cvt.s.d`	F_d, F_s	convert double to single	$F_d \leftarrow [S]F_s$
⋆`cvt.s.w`	F_d, F_s	convert integer to single	$F_d \leftarrow [S]F_s$

15 Verschiedenes

⋆`break`	n	break	Exception n auslösen
⋆`syscall`		system call	Betriebssystemaufruf
⋆`rfe`		return from exception	Statusregister restaurieren
`nop`		no operation	leere Anweisung