

## Befehlssatz MIPS-R2000

### 0 Notation

<i>RI</i>	$R_t \mid R_s$	wahlweise Register $R_t$ oder Direktoperand <i>imm</i>
<i>imm</i>		16-Bit Direktoperand, Wert:
	$[\textit{symbol}] [\pm \textit{dist}]$	$\textit{symbol} + \textit{dist}$
	$(\textit{dist}) \gg \textit{int}$	$\textit{dist} / 2^{\textit{int}}$
<i>dist</i>	$\textit{int}_t \mid \pm \textit{int}_s$	Distanzangabe $\textit{int}_t + \textit{int}_s$
<i>addr</i>	$[\textit{symbol}] [\pm \textit{dist}] [(R_s)]$	Adressangabe für Speicherstelle $\textit{symbol} + \textit{dist} + R_s$
$C(x)_n$		$n$ Bytes großer Wert an der Speicherstelle $x$
$X_{[b_1 \dots b_n]}$		Bits $b_1$ bis $b_n$ des Wertes $X$
<i>label</i>	<i>symbol</i>	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	$\$zero$	konstante 0
\$1	$\$at$	reserviert für den Assembler
\$2-\$3	$\$v0-\$v1$	Ausdruck- bzw. Funktionsergebnis
\$4-\$7	$\$a0-\$a3$	Funktionsargument
\$8-\$15	$\$t0-\$t7$	frei, wird bei Funktionsaufrufen überschrieben
\$16-\$23	$\$s0-\$s7$	frei, wird bei Funktionsaufrufen gerettet
\$24-\$25	$\$t8-\$t9$	frei, wird bei Funktionsaufrufen überschrieben
\$26-\$27	$\$k0-\$k1$	reserviert für Betriebssystem
\$28	$\$gp$	Zeiger auf globalen Datenbereich
\$29	$\$sp$	Stack-Pointer
\$30	$\$fp$	Zeiger auf lokalen Datenbereich einer Funktion
\$31	$\$ra$	Rücksprungadresse

### 2 Ganzzahlarithmetik

<i>abs</i>	$R_t, R_s$	absolute value	$R_d \leftarrow  R_s $
<i>neg</i>	$R_d, R_s$	negate value	$R_d \leftarrow -R_s$
* <i>add</i>	$R_d, R_s, R_t$	addition	$R_d \leftarrow R_s + R_t$
* <i>addi</i>	$R_d, R_s, \textit{imm}$	addition immediate	$R_d \leftarrow R_s + \textit{imm}$
* <i>sub</i>	$R_d, R_s, R_t$	subtract	$R_d \leftarrow R_s - R_t$
* <i>mult</i>	$R_s, R_t$	multiply	$(hi, lo) \leftarrow R_s \times R_t$
<i>mul</i>	$R_d, R_s, R_t$	multiply	$R_d \leftarrow R_s \times R_t$
† <i>mulou</i>	$R_d, R_s, R_t$	multiply with overflow	$R_d \leftarrow R_s \times R_t$
* <i>div</i>	$R_s, R_t$	divide	$lo \leftarrow R_s / R_t, hi \leftarrow R_s \bmod R_t$
† <i>divu</i>	$R_d, R_s, R_t$	divide	$R_d \leftarrow R_s / R_t$
† <i>rem</i>	$R_d, R_s, R_t$	remainder	$R_d \leftarrow R_s \bmod R_t$

### 3 Ganzzahlvergleiche

<i>seq</i>	$R_d, R_s, R_t$	set equal	$R_d \leftarrow R_s = R_t ? 1 : 0$
<i>sne</i>	$R_d, R_s, R_t$	set not equal	$R_d \leftarrow R_s \neq R_t ? 1 : 0$
<i>sgt</i>	$R_d, R_s, R_t$	set greater than	$R_d \leftarrow R_s > R_t ? 1 : 0$
<i>sge</i>	$R_d, R_s, R_t$	set greater than equal	$R_d \leftarrow R_s \geq R_t ? 1 : 0$
* <i>slt</i>	$R_d, R_s, R_t$	set less than	$R_d \leftarrow R_s < R_t ? 1 : 0$
* <i>slti</i>	$R_d, R_s, \textit{imm}$	set less than immediate	$R_d \leftarrow R_s < \textit{imm} ? 1 : 0$
<i>sle</i>	$R_d, R_s, R_t$	set less than equal	$R_d \leftarrow R_s \leq R_t ? 1 : 0$

### 4 Logische Operationen

<i>not</i>	$R_d, R_s$	bitwise logical negation	$R_d \leftarrow \neg R_s$
* <i>and</i>	$R_d, R_s, R_t$	bitwise AND	$R_d \leftarrow R_s \wedge R_t$
* <i>andi</i>	$R_d, R_s, \textit{imm}$	bitwise AND immediate	$R_d \leftarrow R_s \wedge \textit{imm}$
* <i>or</i>	$R_d, R_s, R_t$	bitwise OR	$R_d \leftarrow R_s \vee R_t$
* <i>ori</i>	$R_d, R_s, \textit{imm}$	bitwise OR immediate	$R_d \leftarrow R_s \vee \textit{imm}$
* <i>xor</i>	$R_d, R_s, R_t$	bitwise XOR	$R_d \leftarrow R_s \oplus R_t$
* <i>xori</i>	$R_d, R_s, \textit{imm}$	bitwise XOR immediate	$R_d \leftarrow R_s \oplus \textit{imm}$
* <i>nor</i>	$R_d, R_s, R_t$	bitwise NOR	$R_d \leftarrow \neg(R_s \vee R_t)$

### 5 Shift-Operationen

* <i>sll</i>	$R_d, R_t, \textit{imm}$	shift left logical	$R_d \leftarrow R_t * 2^{\textit{imm}} \bmod 32$
* <i>sllv</i>	$R_d, R_t, R_0$	shift left logical variable	$R_d \leftarrow R_t * 2^{R_0} \bmod 32$
* <i>srl</i>	$R_d, R_t, \textit{imm}$	shift right logical	$R_d \leftarrow R_t / 2^{\textit{imm}} \bmod 32$
* <i>srlv</i>	$R_d, R_t, R_0$	shift right logical variable	$R_d \leftarrow R_t / 2^{R_0} \bmod 32$
* <i>sra</i>	$R_d, R_t, \textit{imm}$	shift right arithmetic	$R_d \leftarrow R_t / 2^{\textit{imm}} \bmod 32$
* <i>srav</i>	$R_d, R_t, R_0$	shift right arithmetic variable	$R_d \leftarrow R_t / 2^{R_0} \bmod 32$
◊ <i>rol</i>	$R_d, R_t, R_t$	rotate left	$R_d \leftarrow R_t \ll ((31 - R_t) \bmod 32) .. 0, 31 .. ((31 - R_t - 1) \bmod 32)$
◊ <i>ror</i>	$R_d, R_t, R_t$	rotate right	$R_d \leftarrow R_t \ll ((R_t - 1) \bmod 32) .. 0, 31 .. (R_t \bmod 32)$

### 6 Unbedingte Sprünge

<i>b</i>	<i>label</i>	branch instruction	Sprung nach <i>label</i>
* <i>j</i>	<i>label</i>	jump	Sprung nach <i>label</i>
* <i>jr</i>	$R_s$	jump register	Sprung nach $R_s$
* <i>jal</i>	<i>label</i>	jump and link	$\$31 \leftarrow IP + 4$ , Sprung nach <i>label</i>
<i>bal</i>	<i>label</i>	branch and link	$\$31 \leftarrow IP + 4$ , Sprung nach <i>label</i>
* <i>jalr</i>	$R_d, R_s$	jump and link register	$R_d \leftarrow IP + 4$ , Sprung nach $R_s$

### 7 Bedingte Sprünge nach Vergleich

* <i>bc1f</i>	<i>label</i>	branch coprocessor 1 false	Sprung nach <i>label</i> , wenn $CF_1 = 0$
* <i>bc1t</i>	<i>label</i>	branch coprocessor 1 true	Sprung nach <i>label</i> , wenn $CF_1 = 1$
* <i>bne</i>	$R_s, R_t, \textit{label}$	branch on not equal	Sprung nach <i>label</i> , wenn $R_s \neq R_t$
* <i>bne</i>	$R_s, R_t, \textit{label}$	branch on not equal	Sprung nach <i>label</i> , wenn $R_s \neq R_t$
<i>bgt</i>	$R_s, R_t, \textit{label}$	branch on greater than	Sprung nach <i>label</i> , wenn $R_s > R_t$
<i>bge</i>	$R_s, R_t, \textit{label}$	branch on greater than equal	Sprung nach <i>label</i> , wenn $R_s \geq R_t$
<i>blt</i>	$R_s, R_t, \textit{label}$	branch on less than	Sprung nach <i>label</i> , wenn $R_s < R_t$
<i>ble</i>	$R_s, R_t, \textit{label}$	branch on less than equal	Sprung nach <i>label</i> , wenn $R_s \leq R_t$

## 8 Bedingte Sprünge nach Vergleich mit 0

beqz	$R_s, label$	branch on equal zero	Sprung nach <i>label</i> , wenn $R_s = 0$
bneqz	$R_s, label$	branch on not equal zero	Sprung nach <i>label</i> , wenn $R_s \neq 0$
**bgtz	$R_s, label$	branch on greater than zero	Sprung nach <i>label</i> , wenn $R_s > 0$
**bgez	$R_s, label$	branch on greater than equal zero	Sprung nach <i>label</i> , wenn $R_s \geq 0$
**bgezal	$R_s, label$	branch on greater than equal zero and link	$\$31 \leftarrow IP + 4$ , Sprung nach <i>label</i> , wenn $R_s \geq 0$
**bltz	$R_s, label$	branch on less than zero	Sprung nach <i>label</i> , wenn $R_s < 0$
**bltzal	$R_s, label$	branch on less than and link	$\$31 \leftarrow IP + 4$ , Sprung nach <i>label</i> , wenn $R_s < 0$
**blez	$R_s, label$	branch on less than equal zero	Sprung nach <i>label</i> , 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	$R_d \leftarrow C(addr)_4$
**lw1	$R_d, addr$	load word left	$R_{d[31...(addr \bmod 4)*8]} \leftarrow C(addr)_{addr \bmod 4}$
**lw1r	$R_d, addr$	load word right	$R_{d[31...(addr \bmod 4)*8+7..0]} \leftarrow C(addr)_{addr \bmod 4}$
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	$C(addr)_1 \leftarrow R_{t[7..0]}$
*sh	$R_t, addr$	store halfword	$C(addr)_2 \leftarrow R_{t[15..0]}$
*sw	$R_t, addr$	store word	$C(addr)_4 \leftarrow R_t$
*sw1	$R_t, addr$	store word left	$C(addr)_{addr \bmod 4} \leftarrow R_{t[31...(addr \bmod 4)*8]}$
*swr	$R_t, addr$	store word right	$C(addr)_{addr \bmod 4} \leftarrow R_{t[(addr \bmod 4)*8+7..0]}$
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	$C(addr)_2 \leftarrow R_{t[15..0]}$
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  F_s $
*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	$CF_1 \leftarrow (F_s = NaN) \vee (F_s = NaN)$
*c.eq.s	*c.eq.d	$F_s, F_t$	compare equal	$CF_1 \leftarrow F_s = F_t$
*fc.seq.s	*fc.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	$CF_1 \leftarrow (F_s = F_t) \vee (F_s = NaN) \vee (F_s = NaN)$
*fc.lt.s	*fc.lt.d	$F_s, F_t$	compare less than	$CF_1 \leftarrow F_s < F_t$
*c.lt.s	*c.lt.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	$CF_1 \leftarrow (F_s < F_t) \vee (F_s = NaN) \vee (F_s = NaN)$
*fc.le.s	*fc.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	$CF_1 \leftarrow F_s \leq F_t$
*c.ule.s	*c.ule.d	$F_s, F_t$	compare unordered less than equal	$CF_1 \leftarrow (F_s \leq F_t) \vee (F_s = NaN) \vee (F_s = NaN)$

## 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