Maschinenprogramme sind architekturgebunden, d.h. sie sind nicht portabel (zu anderen Architekturen)
-aktueller Maschinenbefehl wird im Befehlsregister abgelegt.
- Jeder Maschinenbefehl enthält *Funktionscode*, der vom Leitwerk ausgewertet (dekodiert) wird.

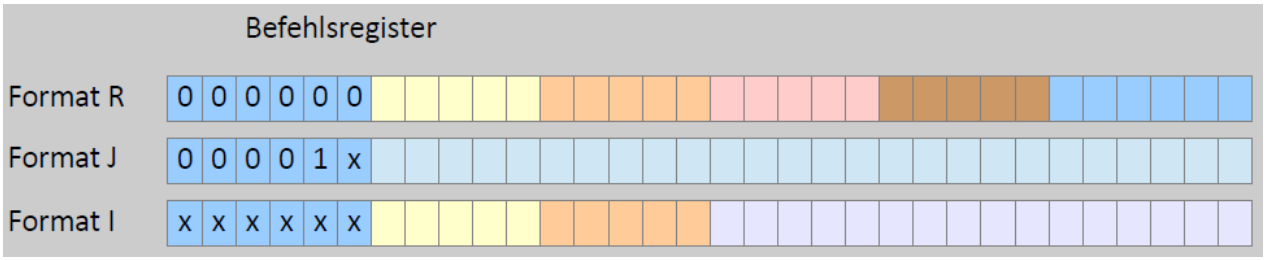
-Maschinenbefehle sind im MARS immer 32bit lang, -> Befehlsregister 32bit

Befehlsdekodierung:

Format R: Register-Operationen

Format J: Jump-Operationen: nur bei absoluten Sprüngen, relative Sprünge sind Immediate!

Format I: Immediate-Operationen: Nur letzten 16bit sind im Befehl kodiert; Aufwertung auf Registerbreite, bei arithmetischen Operationen mit 0 (0 - 7FFF) oder mit 1 (8000 – FFFF); bei logisch immer mit 0



Absolute Sprünge:
Format J: 26bit für Adresse: ersten 4bit werden von Quelladresse übernommen, letzten 2bit müssen 0 sein, da Wortbreite 32bit

Relative Sprünge:
Format I: Sprung zu Offset von aktueller Adresse: Sprungziel immediate; Wert im Zweierkomplement, d.h. -32768 bis +32767bit Sprungweite

Indirekte Adressierung:
Format R: Zieladresse steht in 32bit Register, Sprung an Adresse im Register

Load/Store:
Format I: Angabe eines Offset beim Laden /Speichern

Assembler:
- Kurze Mnemonics anstatt langer Maschinenbefehle; Assembler verwandelt Mnemonics in Maschinenbefehle

Notationen:
Format R: Mnemoic $Rd, $Rs, $Rt Format I: Mnemonic $Rt, $Rs, Wert Format I: Mnemonic, Wert

Jeder Maschinenbefehl in MIPS in einer Zeile darstellbar (Ausnahme: Pseudobefehle)

Marken:
Marke besteht aus lesbarem Text, repräsentiert Position im Speicher; werden für Sprung- und Speicheroperationen genutzt; wird bei Assemblierung vom Assembler in Adresse verwandelt

Angabe von Konstanten im Assemblercode:
- üblicherweise dezimal, oder als Hexwert (0x1234)
- negative Werte durch vorangestelltes `-`, auch bei Hex

Pseudo-Befehle:
Pseudobefehle sind keine Architekturbefehle; werden vom Assembler in (mehrere) Architekturbefehle verwandelt

Darstellung nicht-numerische Werte:
ASCII: American Standard Code for Information Interchange
7bit Code -> 128 Zeichen
Zeichen im Computer als Byte darstellbar
Eingabe als Zeichenkonstanten möglich:
li $t0, 'A'
li $t0, 0x41
li $t0, 65
nicht erlaubt: li $t0, 'ABC'

Nummer	Direkte Bezeichnung	Symbolische Bezeichnung	Bedeutung
0	$0	$zero	immer 0
1	$1	$at	Assembler nutzt dies temporär
2, 3		$v0, $v1	Ergebnisse (values) von Unterprogrammen
4 ... 7		$a0 ... $a3	Aufrufparameter für Unterprogramme
8 ... 15	...	$t0 ... $t7	Temporäre Werte; können vom Uprg geändert werden
16 ... 23		$s0 ... $s7	Gesicherte Werte; zurückzustellen vor Rückkehr
24, 25		$t8, $t9	Weitere temporäre Werte
26, 27		$k0, $k1	Reserviert für spezielle Ereignisse
28		$gp	Globalspeicherzeiger (Pointer)
29	$29	$sp	Stapelspeicherzeiger (Pointer)
30	$30	$s8 / $fp	Frame pointer bzw. weitere s-Variable
31	$31	$ra	Rücksprungadresse für Unterprogramme

Assemblerdirektiven:

.data	Markiert Beginn Datenbereich
.text	Markiert Beginn Codebereich
.byte Wert1,Wert2,..,WertN	Bytes, werden in dieser Reihenfolge im Speicher abgelegt
.word Wert1, Wert2,...,WertN	Wörter, werden in dieser Reihenfolge im Speicher abgelegt
.space n	Reserviert n Byte Speicher
.ascii „Zeichenkette“	Erzeugt Folge von Bytes welche Zeichenkette bilden
.asciiz „Zeichenkette“	Erzeugt Folge von Bytes welche Zeichenkette bilden + 00Byte am Ende (0-terminiert)

Kommentare:
werden in Assembler mit '#' begonnen

Retten von Registerwerten:
- Retten von Werten in anderes Register
- Retten von Werten in Speicher

Syscalls:
-gibt Auftrag an OS
- OS holt „Funktionscode“ aus $v0
- OS holt gegebenenfalls Argumente aus $a0 -$a3
- OS speichert Ergebnis (falls vorhanden) in Register

Befehlssatz inkl. Pseudobefehle:

Maschinenbefehl						Mnemonic	Description
Opcode(6)	**Rs (5)**	**Rt(5)**	**Rd(5)**	**Shamt(5)**	**Function(6)**		
000000	Rs	Rt	Rd	00000	100000	add $Rd, $Rs, $Rt	Addiert Inhalt von Rs und Rt und speichert Erg. in Rd
000000	Rs	Rt	Rd	00000	100001	addu $Rd, $Rs, $Rt	Addiert Inhalt von Rs und Rt und speichert Erg. in Rd(unsigned)
001000	Rs	Rt	Wert (16)			addi $Rt, $Rs, Wert	Addiere Inhalt von Rs mit Wert und speichere Ergebnis in Rt
001001	Rs	Rt	Wert (16)			addiu $Rt, $Rs, Wert	Addiere Inhalt von Rs mit Wert und speichere Ergebnis in Rt(unsign)
000000	Rs	Rt	Rd	00000	100010	sub $Rd, $Rs, $Rt	Subtrahiere Inhalt von Rs und Rt und speichere Erg. in Rd
000000	Rs	Rt	Rd	00000	100011	subu $Rd, $Rs, $Rt	Subtrahiere Inhalt von Rs und Rt und speichere Erg. in Rd(unsign)
000000	Rs	Rt	00000	00000	011010	div $Rs, $Rt	Dividiert Rs durch Rt und speichert Erg. in *lo*, mod. in *hi*
000000	Rs	Rt	00000	00000	011011	divu $Rs, $Rt	Dividiert Rs durch Rt und speichert Erg. in *lo*, mod. in *hi* (unsign)
000000	Rs	Rt	00000	00000	011000	mult $Rs, $Rt	Multipliziert Rs mit Rt und speichert Erg. in *hi* und *lo*
000000	Rs	Rt	00000	00000	011001	multu $Rs, $Rt	Multipliziert Rs mit Rt und speichert Erg. in *hi* und *lo* (unsign)
000000	00000	00000	Rd	00000	010000	mfhi $Rd	Kopiert Wert aus *hi* nach Rd
000000	00000	00000	Rd	00000	010010	mflo $Rd	Kopiert Wert aus *lo* nach Rd
000000	Rs	Rt	Rd	00000	100100	and $Rd, $Rs, $Rt	Bitweise UND von Rs und Rt, wird nach Rd gespeichert
000000	Rs	Rt	Rd	00000	100101	or $Rd, $Rs, $Rt	Bitweise ODER von Rs und Rt, wird nach Rd gespeichert
000000	Rs	Rt	Rd	00000	100110	xor $Rd, $Rs, $Rt	Bitweise XOR von Rs und Rt, wird nach Rd gespeichert
000000	Rs	Rt	Rd	00000	100111	nor $Rd, $Rs, $Rt	Bitweise NOR von Rs und Rt, wird nach Rd gespeichert
						not $Rd, $Rs	Schreibt Einerkomplement von Rs nach Rd
001100	Rs	Rt	Wert(16)			andi $Rt, $Rs, Wert	Bitweise UND von Rs und Wert(wird mit 0x0000 aufgew.) -> Rd
001101	Rs	Rt	Wert(16)			ori $Rt, $Rs, Wert	Bitweise ODER von Rs und Wert, wird in Rd gespeichert
001110	Rs	Rt	Wert(16)			xori $Rt, $Rs, Wert	Bitweise XOR von Rs und Wert, wird in Rd gespeichert
000000	00000	Rt	Rd	Shamt	000000	sll $Rd, $Rt, shamt	Schiebe Inhalt Rt um shamt stellen nach links und speichere in Rd; 0er ergänzen
000000	00000	Rt	Rd	Shamt	000010	srl $Rd, $Rt, shamt	Schiebe Inhalt Rt um shamt stellen nach rechts und speichere in Rd; 0er ergänzen
000000	00000	Rt	Rd	Shamt	000011	sra $Rd, $Rt, shamt	Schiebe Inhalt Rt um shamt stellen nach rechts und speichere in Rd; je nach Vorzeichen werden 0er oder 1er ergänzt
000000	Rs	Rt	Rd	00000	101010	slt $Rd, $Rs, $Rt	Setzt Rd = 1 wenn Rs echt-kleiner Rt, sonst Rd = 0
000000	Rs	Rt	Rd	00000	101011	sltu $Rd, $Rs, $Rt	Setzt Rd = 1 wenn Rs echt-kleiner Rt, sonst Rd = 0 (Vergl. unsign)
001010	Rs	Rt	Wert(16)			slti $Rt, $Rs, Wert	Setzr Rt = 1 wenn Rs echt-kleiner Wert, sonst Rt = 0
001011	Rs	Rt	Wert(16)			sltiu $Rt, $Rs, Wert	Setzr Rt = 1 wenn Rs echt-kleiner Wert, sonst Rt = 0 (Vergl. unsign)

000100	Rs	Rt	Wert(16)		beq $Rs, $Rt, Wert (Offset/Marke)	Wenn Inhalt von Rs und Rt gleich, springe um Offset (Marke: Assembler berechnet Offset)
000101	Rs	Rt	Wert(16)		bne $Rs, $Rt, Wert (Offset/Marke)	Wenn Inhalt von Rs und Rt nicht gleich, springe um Offset (Marke: Assembler berechnet Offset)
					bgez $Rs, Wert (Marke)	Springt zur Marke wenn Rs >= 0
					bgtz $Rs, Wert (Marke)	Springt zur Marke wenn Rs > 0
					blez $Rs, Wert (Marke)	Springt zur Marke wenn RS <= 0
					bltz $Rs, Wert (Marke)	Springt zur Marke wenn RS < 0
					b Marke	Springt um Offset (zur Marke, Assembler berechnet Offset
000010	Wert(26)				j Wert (Adresse/Marke)	Springe zur Adresse (Marke: Assembler setzt Adresse ein)
000011	Wert(26)				jal Wert (Adresse/Marke)	Springe zur Adresse und rette Adresse nächster Instruktion in $ra
000000	Rs	00000	00000	00000 001000	jr $Rs	Springe zu Adresse die in Rs steht
001111	00000	Rt	Wert(16)		lui $Rt, Wert	Speichert Wert in die oberen 16bit von Rt
100011	Rs	Rt	Wert(16)		lw $Rt, Offset($Rs)	Hole Wort der Speicherzellen deren Adresse in Rs steht um 8 reduziert, und lege in Rt
100000	Rs	Rt	Wert(16)		lb $Rt, Offset($Rs)	Hole Byte von Adresse und speichere es in den rechtesten 8bit v. Rt
100100	Rs	Rt	Wert(16)		lbu $Rt, Offset($Rs)	Hole Byte von Adresse und speichere es in den rechtesten 8bit v. Rt
					li $Rd, Wert(32)	Schreibt Wert in Rd
					la $Rd, Marke	Schreibt Adresse der Marke in Rd
					move $Rd, $Rs	Kopiert Inhalt von Rs nach Rd
101011	Rs	Rt	Wert(16)		sw $Rt, Offset($Rs)	Schreibe den Inhalt von Rt an die Speicherstelle Rs + Offset
101000	Rs	Rt	Wert(16)		sb $Rt, Offset($Rs)	Schreibe die rechtesten 8bit von Rt an die Speicherstelle Rs + Offset

ASCII:

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciicharstable.com

Programmbeispiele:

if-else:

```
        bne   $s3, $s4, else   # if
        add   $s0, $s1, $s2    # then
        b     endif            # springe über else-Teil
else:   sub   $s0, $s1, $s2
endif:  # weiter im Programm
```

for-Schleife:

```
        la    $t1, a           # t1 = Adresse des Feldes
        li    $t0, 0           # i = 0
        li    $v0, 0           # summe = 0
weiter: beq   $t0, $a0, ende   # i == n? Wenn ja, Ende
        sll   $t2, $t0, 2      # adr = i · 4
        add   $t2, $t2, $t1    #       + Anfang
        lw    $t3, ($t2)       # Hole den Wert
        add   $v0, $v0, $t3    # Addiere ihn auf summe
        addi  $t0, $t0, 1      # i = i + 1
        b     weiter
ende:   # weiter im Programm
```

while-Schleife:

```
        addi $t0, $zero, 10     #t0 = Laufvariable
loop:   .....                   #Schleifenrumpf
        .....
        addi $t0, $t0, -1       #Vermindere t0
        bne $t0, $zero, loop    #Springe solange t0 != 0 (hier 10 mal)
        ....                    #weiter im Programm
```