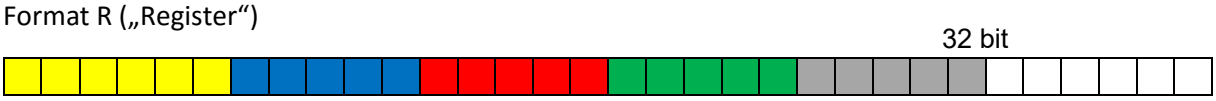
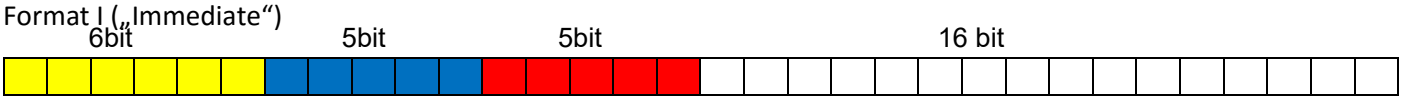
Format R („Register“)



Opcode Rs Rt Rd Shamt Function

Opcode	Befehl	Function	Format	Beschreibung
Rechenoperationen				
0 0 0 0 0 0	add	1 0 0 0 0 0	add Rd, Rs, Rt	Addition mit Beachtung Vorzeichen RD = RS + RT
0 0 0 0 0 0	addu	1 0 0 0 0 1	addu Rd, Rs, Rt	Vorzeichenlose Addition
0 0 0 0 0 0	sub	1 0 0 0 1 0	sub Rd, Rs, Rt	Subtraktion mit Beachtung Vorzeichen
0 0 0 0 0 0	subu	1 0 0 0 1 1	subu Rd, Rs, Rt	Vorzeichenlose Subtraktion
0 0 0 0 0 0	div	0 1 1 0 1 0	div Rs, Rt	Ganzzahlige Div. mit Rest/Beachtung Vorzeichen: hi = Rs mod Rt lo = Rs / Rt
0 0 0 0 0 0	divu	0 1 1 0 1 1	divu Rs, Rt	Vorzeichenlose, ganzzahlige Div. mit Rest
0 0 0 0 0 0	mult	0 1 1 0 0 0	mult Rs, Rt	Multiplikation mit Beachtung Vorzeichen (hi, lo) = Rs * Rt
0 0 0 0 0 0	multu	0 1 1 0 0 1	multu Rs, Rt	Vorzeichenlose Multiplikation
0 0 0 0 0 0	mfhi	0 1 0 0 0 0	mfhi Rd	Move from high, kopiert den Wert von hi in Rd
0 0 0 0 0 0	mflo	0 1 0 0 1 0	mflo Rd	Move from low, kopiert den Wert von lo in Rd
Vergleichsoperationen				
0 0 0 0 0 0	slt	1 0 1 0 1 0	slt Rd, Rs, Rt	„Set on Less Than“ – Wenn Rs < Rt dann Rd = 1, sonst Rd = 0
0 0 0 0 0 0	sltu	1 0 1 0 1 1	sltu Rd, Rs, Rt	Vorzeichenlose Variante
Logische Operationen				
0 0 0 0 0 0	and	1 0 0 1 0 0	and Rd, Rs, Rt	Bitweise UND-Verknüpfung von Rs und Rt
0 0 0 0 0 0	or	1 0 0 1 0 1	or Rd, Rs, Rt	ODER
0 0 0 0 0 0	xor	1 0 0 1 1 0	xor Rd, Rs, Rt	Exklusiv-ODER
0 0 0 0 0 0	nor	1 0 0 1 1 1	nor Rd, Rs, Rt	Nicht-ODER
Schiebeoperationen				
0 0 0 0 0 0	sll	0 0 0 0 0 0	sll Rd, Rt, shamt	„Shift Left Logical“ - Schieben der Binärstellen um n Positionen nach links - n wird im Feld shamt binär dargestellt - von rechts werden Nullen ergänzt
0 0 0 0 0 0	srl	0 0 0 0 1 0	srl Rd, Rt, shamt	„Shift Right Logical“ - Schieben der Binärstellen um n Positionen nach rechts - n wird im Feld shamt binär dargestellt
0 0 0 0 0 0	sra	0 0 0 0 1 1	sra, Rd, Rt, shamt	„Shift Right Arithmetic“ - Schieben der Binärstellen um n Positionen nach rechts - n wird im Feld shamt binär dargestellt
			Skr. S. 55 / Kap. 9	Unterschied: sra berücksichtigt das Vorzeichen
Unbedingte Sprünge				
0 0 0 0 0 0	jr	0 0 1 0 0 0	jr Rs	Kopiere den Inhalt von Rs in den Programmschrittzähler
Pseudo-Befehle				
Pseudo-Befehl			Echter Befehl	Beschreibung
move Rd, Rs			add Rd, Rs, $zero	Kopiert einen Wert von einem Register zu einem anderen
not Rd, Rs			nor Rd, Rs, $zero	Bilden des Einerkomplements

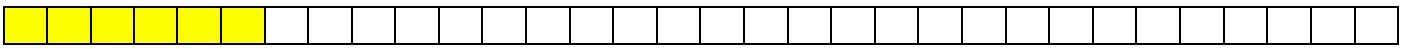
Format I („Immediate“)



Opcode Rs Rt Wert/Versatz

Opcode	Befehl	Format	Beschreibung
Rechenoperationen			
0 0 1 0 0 0	addi	addi Rt, Rs, Wert	Unmittelbare Addition mit Beachtung Vorzeichen
0 0 1 0 0 1	addiu	addiu Rt, Rs, Wert	Vorzeichenlose Addition
Vergleichsoperationen			
0 0 1 0 1 0	slti	slti Rt, Rs, Wert	„Set on Less Than“ – Wenn Rs < Wert dann Rt = 1, sonst Rt = 0
0 0 1 0 1 1	sltiu	sltiu Rt, Rs, Wert	Vorzeichenlose Variante
Logische Operationen			
0 0 1 1 0 0	andi	andi Rt, Rs, Wert	Bitweise UND-Verknüpfung von Rs und Rt
0 0 1 1 0 1	ori	ori Rt, Rs, Wert	ODER
0 0 1 1 1 0	xori	xori Rt, Rs, Wert	Exklusiv-ODER
Beladen von Registern			
Problem: Register haben 32-Bit, alle bisherigen Befehle mit unmittelbaren Argumenten können jedoch nur 16 Bit als Wert anbieten. Lösung: Spezieller lui-Befehl (siehe Skript S. 52 / Kap. 9)			
0 0 1 1 1 1	lui	lui Rt, Wert	„Load upper word immediate“ 1. Schritt: Setzen der oberen 16 Bit
0 0 1 1 1 0	ori	xori Rt, Rs, Wert	2. Schritt: Setzen der unteren 16 Bit
Sprungoperationen letzter Teil: Versatz, wird aber vom Assembler ausgerechnet wenn marke angegeben wird			
0 0 0 1 0 0	beq	beq Rs, Rt, Marke	„Branch if equal“ – Wenn Rs = Rt, dann springe zur Marke
0 0 0 1 0 1	bne	bne Rs, Rt, Marke	„Branch if not equal“ – Wenn Rs != Rt, dann springe zur Marke
			Siehe Skript S.55 / Kap. 9
Speicheroperationen Versatz wird in hex schreibweise 0xZZZZ angegeben			
1 0 0 0 1 1	lw	lw Rt, Versatz (Rs)	„Load Word“ – Hole das Wort von der Adresse im Speicher, die sich aus dem Inhalt von Rs plus dem Versatz ergibt und speichere es in Rt
1 0 0 0 0 0	lb	lb Rt, Versatz (Rs)	„Load Byte“ – Hole das Byte von der (siehe oben) und speichere es in den rechtesten 8 Bit von Rt füllt die höherwertigen bits auf (belässt sie also nicht)
1 0 0 1 0 0	lbu	lbu Rt, Versatz (Rs)	Siehe lb
1 0 1 0 1 1	sw	sw Rt, Versatz (Rs)	„Store Word“ – Schreibe den Inhalt von Rt an die Speicheradresse
1 0 1 0 0 0	sb	sb Rt, Versatz (Rs)	„Store Byte“ – Schreibe die rechtesten 8 Bit von Rt an die berechnete Speicheradresse (siehe seite 61 kapitel 9)
			Siehe Skript S. 59 - 62 / Kap. 9
Pseudo-Befehl		**Echte Befehle**	
li Rd, Wert		lui Rd, Wert1 ori Rd, Rd, Wert2	Belegen eines Registers mit einem 32-Bit-Wert Wert = (Wert1 Wert2) Wert = 32Bit Zahl
la Rd, Marke wichtig		lui Rd, Wert1 ori Rd, Rd, Wert2	Laden der Adresse einer Marke wichtig denn man kennt die Adresse nicht Adresse = (Wert1 Wert2) man programmiert mit marken als verweise
b Marke		beq $zero, $zero, Marke	Unbedingter (relativer) Sprung

Format J („Jump“)



Opcode Zieladressenangabe

Opcode	Befehl	Format	Beschreibung
			Unbedingte Sprünge ziel muss im gleichen bereich sein wie befehl weil die ersten 4 bits ergänzt werden
0 0 0 0 1 0	j	j Marke	„Jump“ – Springe zur Marke
0 0 0 0 1 1	jal	jal Marke	„Jump And Link“ – Speichere die Adresse der nächsten Instruktion in $ra und springe dann zur Marke
			Siehe Skript S. 56 / Kap. 9

Definition Marke:

Für Sprungoperationen benötigen wir eine Möglichkeit, die Position im Code anzugeben.

- Relative Sprünge: Angabe der Wörter nach vorne oder nach hinten
  Wir müssten nachzählen, wie weit das Sprungziel entfernt ist.
- Absolute Sprünge: Angabe der Zieladresse
  Wir müssten zu jeder Programmzeile wissen, auf welcher Adresse sie liegt.

Für beide Zwecke führt der Assembler so genannte **Marken** (Labels) ein:

- Eine Marke ist ein Symbol, das eine Position im Speicher repräsentiert
- Eine Marke wird gesetzt, indem sie im Programmcode vor der betreffenden Codezeile platziert wird und ein Doppelpunkt angefügt wird

Marke: Codezeile oder Marke:

Codezeile

(siehe Skript S. 57 / Kap. 9)

MIPS-Register in Assemblersprache

für uns interessant

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