Open CheatSheet: GdI-Edition

Source https://github.com/hubwoop/ocs-gdi-ohm

5. Februar 2017

1 Tabellenwerk

Tabelle 1: Umrechnung zu dezimal bis b^{12}

	b^0	b^1	b^2	b^3	b^4	b^5	b^6	b^7	b^8	b^9	b^{10}	b^{11}	b^{12}
$b = 2$	1	2	4	8	16	32	64	128	256	512	1024	2048	4096
$b = 8$	1	8	64	512	4096	-	-	-	-	-	-	-	-
$b = 16$	1	16	256	4096	65536	-	-	-	-	-	-	-	-

Tabelle 2: Gängige Zahlensysteme: Darstellungen bis Wert 15

dez	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
bin	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111
oct	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Tabelle 3: vielfache von 10_{10} (Nützlich für die Berechnung von Dezimalzahlen im Quellsystem)

Basis	10	2	3	4	5	6	7	8	9
10		1010	101	22	20	14	13	12	11
20			202	110	40	32	26	24	22
30				132	110	50	42	36	33
40					130	104	55	50	44
50						122	101	62	55
60							114	74	66
70								106	77
80									88

2 Rechnen in b-Adischen Systemen

2.1 Subtraktion

Beispiel: $(11)_4 - (2)_4 = (2)_4$
Wie geht das: Man leiht sich eine 1 von der nächsten stelle und hat somit effektiv $(4 + 1 - 2)_{10} = 3_{10} = 3_4$

3 Umrechnen

3.1 $b = 10$ zu $b \neq 10$ (Basis 10 zu beliebige Basis ungleich 10)

1. Rechne mod b notiere Rest als letzte stelle des Ergebnisses
2. Rechne mod b von Ganzahlanteil von Schritt notiere Stelle als vorletzte des Ergebnisses
3. Setze fort bis Ganzahlanteil = 0

3.2 $b \neq 10$ zu $b = 10$ (beliebige Basis ungleich 10 zu Basis 10)

3.2.1 Im Zielsystem

Stellen addieren nach folgender Vorschrift:

$$\sum_{n=0}^{N-1}(a_n * b^n), a_n \in \{0, \ldots, b-1\}, i \in \{0, \ldots, N-1\}$$

wobei N: Anzahl der Stellen, n: Stelle der Zahl und a_n: Wert der Ziffer an Stelle n.

3.2.2 Ein Beispiel: $(124032)_5$

$(124032)_5 = (1*5^5)+(2*5^4)+(4*5^3)+(0*5^2)+(3*5^1)+(2*5^0) = 3125+1250+500+0+15+2 = 4892$

3.3 Im Quellsystem

Rechnen im Quellsystem zur Umwandlung von Zahlen beliebiger b-adischer Systeme zum dezimal System

3.3.1 Der Algorithmus in Worten

1. Teile die umzurechnende Zahl durch die Repräsentation der $(10)_{10}$ im Quellsystem bis das Ergebnis der Subtraktionen (innerhalb des Divisionsalgorithmus) nicht mehr durch "10teilbar sind. Das letzte Ergebnis der Subtraktion ist der Divisionsrest.
2. Wiederhole Schritt 1 für jedes Ergebnis (ohne Rest) bis das Resultat nur noch einen Rest Darstellt. (Das heißt Ganzzahlanteil = 0)
3. Die Reste der Divisionen sind die Quellsystemrepräsentanten der einzelnen Stellen der zu berechnenden Dezimalzahl. Die hochwertigste Stelle entspricht der zuletzt vorgenommenen Division.

3.3.2 Ein Beispiel: $(120102)_3$ zu $(416)_{10}$

Folgende Tabelle hilft bei der Berechnung der Divisionen. Man braucht nur vielfache bis $b-1$ zum berechnen der Divisionen.

Tabelle 4: Vielfache von $(10)_{10}$ im 3-er System

Basis 10	basis 3
10	101
20	202

$$120102/101 = 1112\ R\,20 \quad (1)$$
$$1112/101 = 11\ R\,1 \quad (2)$$
$$11/101 = 0\ R\,11 \quad (3)$$

Aus (3) $\Rightarrow (11)_3 \mathrel{\widehat{=}} (4)_{10}$
Aus (2) $\Rightarrow (1)_3 \mathrel{\widehat{=}} (1)_{10}$
Aus (1) $\Rightarrow (20)_3 \mathrel{\widehat{=}} (6)_{10}$

$\Rightarrow (416)_{10}$

4 Vorzeichenbehaftete Darstellungen

4.1 Komplementbildung

$(b-1) - Stelle$ bedeutet für bspw. für hex 15−stelle

Beispiel Oktalsystem Komplement von 00354 = 77423 weil $b - 1 = 7$ und somit lautet die Berechnung für jede stelle v.l.n.r: $7 - 0 = 7, 7 - 0 = 7, 7 - 3 = 4, 7 - 5 = 2, 7 - 4 = 3 \rightarrow 77423$

4.2 kleinste / größte darstellbare Zahl mit N Bits

Exzess-N kleinste: -N; Größte: N-1
größte positive: $2^{Anzahl\ der\ bits}$ - 1 - Exzesswert
kleinste negative: -Exzesswert

b-1-Komplement $[-(2^{N-1} - 1) \ldots + (2^{N-1} - 1)]$
doppelte null: 0000 und 1111 sind beides null

b-Komplement $[-2^{N-1} \ldots + 2^{N-1} - 1]$

4.3 Rechnen mit dem einer Komplement

Den einer Rücklauf beachten (carry bit von höchster stelle wird (falls vorhanden) zum Ergebnis addiert (Ergebnis + 1)

4.4 Rechnen mit dem zweier Komplement

nach der Komplementbildung eins addieren.

4.5 Exzesscode

Sinnvollste Darstellung $2^{AnzahlderBits}/2$
Umwandlung zu Exzess ist immer: darzustellende Zahl + Exzesszahl ($2^{AnzahlderBits}/2$) Unwandlung von Exzess ist immer: Codierte Zahl - Exzesszahl

4.6 Welche Zahlen sind im b-Komplement negativ

bei geraden basen ist die erste Ziffer größergleich $b/2$ gilt die zahl als negativ ansonsten ist sie positiv
So sind die vorzeichenlosen Zahlen $1 \ldots (b^N/2) - 1$ im b-Komplement positiv, $(b^N/2) \ldots b^N - 1$ sind negativ, 0 ist null.
Im (b-1)-Komplement gilt, dass die Zahlen $(b^N/2) \ldots b^N - 2$ negativ sind, $b^N - 1$ ist minus null.

bei ungeraden basen Hier wird die Trennlinie bei der Zahl $(aaa...a)_b$ gezogen, wobei $a = (b-1)/2$ ist. Bei$b = 3$ ist es die Zahl $(111...1)_3$.

5 Reele Zahlen

5.1 2-adische Entwicklung (Dezimal zu Binär wandeln)

1. Multipliziere die Dezimalzahl mit 2
2. Wenn Ergebnis < 0 notiere null ansonsten notiere 1 und ziehe 1 vom Ergebnis eins ab
3. wiederhole Schritt 1 mit Ergebnis der bisherigen Schritte
4. Erstes Ergebnis: Erste Stelle nach dem Komma

Trick: Wenn die Zahl welche binär dargestellt werden soll eine Zweierpotenz im Nenner hat, dann stellt der Wert des Exponenten die Anzahl der Nachkommastellen dar. Also bspw. $1/8 = 1/2^3$ dar. Daraus folgt 3 Nachkommastellen (weil der Exponent die drei ist). Die Zahl auf dem Bruch wird dann von rechts eingeschoben also bspw. $2/8 = 2/2^3 =>$ drei Nachkommastellen und $2 = 01$ daraus folgt das Ergebnis: $0.010 = 1/4$

Beispiel Umwandlung Dezimal zu Binär $(0.2)_{10}$
$0.2 * 2 = 0,4$ Notiere 0
$0.4 * 2 = 0,8$ Notiere 0
$0.8 * 2 = 1,6$ Notiere 1 und ziehe eins von $1,6ab$
$0.6 * 2 = 1,2$ Notiere 1 und ziehe eins von $1,2ab$
$0.2 * 2 = 0,4$ Periodische Dualzahl entdeckt
$(0.2)_{10} = 0.\overline{0011}$

5.2 Konvertieren zwischen Binär/Hexadezimal/Dezimal

Folgende Beispiele beschreiben das Schema:

Vorkomma Anteil (siehe Kapitel 2 und 3 - Umrechnen ganzer Zahlen)

$$10110.110010 \tag{1}$$
$$0001 = (1)_{16} \tag{2}$$
$$0110 = (6)_{16} \tag{3}$$
$$\Rightarrow (16)_{16} \tag{4}$$
$$(16)_{16} = (22)_{10} \tag{5}$$

Nachkomma Anteil zu dez Aufsummieren: $N * b^{-1} + N * b^{-2} + ... + N * b^{-i}$ wobei i die letzte Nachkommastelle ist und N der wert der stelle

Shortcut bei Zahlen mit nur einsen nach dem komma Wert von 0.1111 (also vier stellen nach dem Komma): $(1 - 2^{-4}) = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} = \frac{15}{16}$

Nachkommaanteil von bin zu hex :
Vierergrüppchen bilden: $\Rightarrow (10110.110010)_2 = (0001)(0110).(1100)(1000) = (16.C8)_{16}$

5.3 Gleitkommazahlen

normalisiert ist eine Gleitkommazahl wenn die Ziffern nach dem Komma sich nicht mehr weiter nach links verschieben lassen ohne das werte über das Komma "hinausrutschen". Die folgenden Werte sind bereits normalisiert:

$$0.11001 * 2^{-39} \tag{1}$$
$$0.001011 * 8^4 \tag{2}$$

normalisieren Bei positiven Exponenten gilt: wenn die zahlen nach links geschoben werden müssen muss vom Exponenten abgezogen werden, wenn die zahlen nach rechts geschoben werden muss man zum Exponenten addieren.
Bei negativen verhält es sich genau anders herum.
$0.001011 \cdot 2^{12} = 0.1011 \cdot 2^{10}$
Dabei unbedingt auf die Exponentenbasis achten! Im Dualsystem zur Basis zwei normalisieren geht einfach: pro geschobene stelle wird der Exponent um eins erhöht/erniedrigt. Bei Exponentenbasis 8 kann nur um 3 stellen geschoben werden(pro Exponentveränderung um 1), weil $8 = 2^3$!
$0.00011001 \cdot 8^{-12} = 0.11001 \cdot 8^{-13}$

Unterlauf- und Überlaufbereich Unterlauf: Kleinste zulässige(!) Darstellung wählen und Dezimalwert berechnen. Dabei darauf achten: wenn Normalisierung gefordert wird, ist die kleinste zulässige Mantisse nicht 0.000001 sonder 0.1. In IEEE ist bspw. keine Normalisierung gefordert, daher kann man hier wirklich den kleinst möglichen Mantissenwert annehmen.

Floating-Point generell: Abstand zwischen zwei Ziffern in einem Bereich (Auflösung bestimmen)
Wähle die größere Zahl (von den Rändern des zu untersuchenden Bereiches) und multipliziere diese mit dem niedrigsten Mantissenbit $(2^{-laenge\ der\ Mantisse}) \cdot groessere\ rand\ zahl$.

Beispiele für eine Darstellung mit 15-Bit-Mantisse:
Bereich: $2^{-15} \leq z < 2^{-14}$
Abstand: $(0.000000000000001)_2 \cdot 2^{-14} = 2^{-15} * 2^{-14} = 2^{-29}$

Bereich: $2^{21} \leq z < 2^{22}$
Abstand: $(0.000000000000001)_2 \cdot 2^{22} = 2^{-15} * 2^{22} = 2^{7}$

5.3.1 IEEE-Single-Precision Floating Point

Wichtiges zuerst:

1. normalisiert: Exponent: 0 < EXP < 255 Mantisse: jedes Bitmuster
2. denormalisiert: Exponent: NUR NULLEN Mantisse: jedes Bitmuster UNGLEICH null
3. Null: Exponent: 0 Mantisse: 0
4. unendlich: Exponent: NUR EINSEN Mantisse: 0
5. NaN: Exponent: NUR EINSEN Mantisse: jedes Bitmuster ungleich null
6. Denormalisierte Darstellung: Exponent = -126 und 0 vorm Komma (der Mantisse)

Umrechnung IEEE zu Dez von links: Bit 1: Vorzeichen, Bits 2-9 (8bit): Exponent, Bits 10-32 (23bit): Mantisse.
$$z = \pm 1.m_1...m_{23} * 2^{e_1...e_8 - 127} \text{ mit VZ } 1 \stackrel{\wedge}{=} - \text{ und } 0 \stackrel{\wedge}{=} +$$

Umrechnen Dez zu IEEE

1. Normalisieren auf 1,...
2. Potenz zu basis 2 sicherstellen
3. Mantisse zu dual wandeln
4. Potenz = potenz + 127
5. Potenz zu dual wandeln
6. Erstes Ergebnis: Erste Stelle nach dem Komma

IEEE single precision: Abstand zwischen zwei Ziffern in einem Bereich (Auflösung bestimmen)
Wähle die kleinere Zahl (von den Rändern des zu untersuchenden Bereiches) und multipliziere diese mit dem niedrigsten Mantissenbit $(2^{-23}) \cdot kleinere\ rand\ zahl$.

Beispiele für IEEE-Single-Precision:
Bereich: $2^{-15} \leq z < 2^{-14}$
Abstand: $(0.00000000000000000000001)_2 \cdot 2^{-15} = 2^{-23} * 2^{-15} = 2^{-38}$

Bereich: $2^{21} \leq z < 2^{22}$
Abstand: $(0.00000000000000000000001)_2 \cdot 2^{21} = 2^{-23} * 2^{21} = 2^{-2}$

6 Logische Schaltungen

6.1 Symbolik

+ = oder
* = und

6.2 Minterme und Maxterme

Tabelle 5: Minterm/Maxterm Vergleich mit aussicht auf DNF und KNF

x_2	x_1	x_0	a_0	Minterm	Maxterm
1	1	0	**0**		$\overline{x_2} + \overline{x_1} + x_0$
0	1	1	**1**	$\overline{x_2} \cdot x_1 \cdot x_0$	

Minterme: verundung jener elemente die eins ergeben (0 führt zu verneinung)
Maxterme: veroderung jener elemente die 0 ergeben (1 führt zu verneinung)

6.3 DNF und KNF

DNF vernüpft Minterme mit 'oder' $m_1 + m_2$ wobei beispielsweise $m_1 = x_1 \cdot x_2 \cdot x_3$
KNF verknüpft Maxterme mit 'und': $(M_1) \cdot (M_2)$ wobei beispielsweise $M_1 = x_1 + x_2 + x_3$

6.4 Im KV-Diagram

Wenn einsen zusammengefasst werden ergibt sich die DNF
Wenn nullen zusammengefasst werden ergibt sich die KNF
DNF: Variablen am Rand beim zusammenfassen so betrachten wie sie sind
KNF: Variablen am Rand beim zusammenfassen negiert betrachten: Bereiche die NICHT gemeint sind (also ausgeschlossen werden) werden nicht verneint.
Die Maxterm-Methode unterscheidet sich von der Minterm-Methode lediglich in folgenden Punkten:

1. Statt Einsen werden Nullen zu Päckchen zusammengefasst.
2. Ein Päckchen bildet einen Disjunktionsterm (ODER-Verknüpfungen statt eines Konjunktionsterms).
3. Die Disjunktionsterme werden konjunktiv (mit UND) verknüpft.
4. Die Variablen werden zusätzlich einzeln negiert.

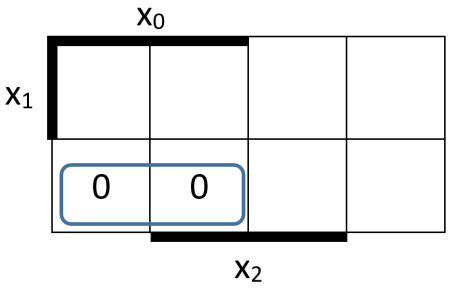


Abbildung 1: KV Diagram KNF lautet: $\overline{x_0} + x_1$

7 Assembler

7.1 Bits zu Assembler wandeln

Dabei ist besonders darauf zu achten, dass die Reihenfolge der Bits nicht der Reihenfolge Befehle in Assemblersprache entspricht

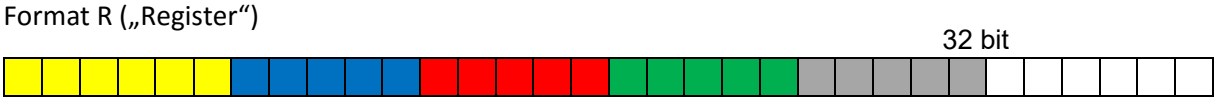
7.2 Coden

Dabei darauf achten, dass oftmals, wenn über Register iteriert wird, auf einen Zähler verzichtet werden kann, da das 'Zielregister' berechnet und als Abbruchbedingung genutzt werden kann.

7.3 ASCII-Tabelle

Dez	Hex	Zeichen	Dez	Hex	Zeichen	Dez	Hex	Zeichen	Dez	Hex	Zeichen
0	0x00	NUL	32	0x20	SP	64	0x40	@	96	0x60	‘
1	0x01	SOH	33	0x21	!	65	0x41	A	97	0x61	a
2	0x02	STX	34	0x22	“	66	0x42	B	98	0x62	b
3	0x03	ETX	35	0x23	#	67	0x43	C	99	0x63	c
4	0x04	EOT	36	0x24	$	68	0x44	D	100	0x64	d
5	0x05	ENQ	37	0x25	%	69	0x45	E	101	0x65	e
6	0x06	ACK	38	0x26	&	70	0x46	F	102	0x66	f
7	0x07	BEL	39	0x27	’	71	0x47	G	103	0x67	g
8	0x08	BS	40	0x28	(	72	0x48	H	104	0x68	h
9	0x09	TAB	41	0x29	)	73	0x49	I	105	0x69	i
10	0x0A	LF	42	0x2A	*	74	0x4A	J	106	0x6A	j
11	0x0B	VT	43	0x2B	+	75	0x4B	K	107	0x6B	k
12	0x0C	FF	44	0x2C	,	76	0x4C	L	108	0x6C	l
13	0x0D	CR	45	0x2D	-	77	0x4D	M	109	0x6D	m
14	0x0E	SO	46	0x2E	.	78	0x4E	N	110	0x6E	n
15	0x0F	SI	47	0x2F	/	79	0x4F	O	111	0x6F	o
16	0x10	DLE	48	0x30	0	80	0x50	P	112	0x70	p
17	0x11	DC1	49	0x31	1	81	0x51	Q	113	0x71	q
18	0x12	DC2	50	0x32	2	82	0x52	R	114	0x72	r
19	0x13	DC3	51	0x33	3	83	0x53	S	115	0x73	s
20	0x14	DC4	52	0x34	4	84	0x54	T	116	0x74	t
21	0x15	NAK	53	0x35	5	85	0x55	U	117	0x75	u
22	0x16	SYN	54	0x36	6	86	0x56	V	118	0x76	v
23	0x17	ETB	55	0x37	7	87	0x57	W	119	0x77	w
24	0x18	CAN	56	0x38	8	88	0x58	X	120	0x78	x
25	0x19	EM	57	0x39	9	89	0x59	Y	121	0x79	y
26	0x1A	SUB	58	0x3A	:	90	0x5A	Z	122	0x7A	z
27	0x1B	ESC	59	0x3B	;	91	0x5B	[	123	0x7B	{
28	0x1C	FS	60	0x3C	«	92	0x5C	\	124	0x7C	
29	0x1D	GS	61	0x3D	=	93	0x5D	]	125	0x7D	}
30	0x1E	RS	62	0x3E	»	94	0x5E	^	126	0x7E	-
31	0x1F	US	63	0x3F	?	95	0x5F	_	127	0x7F	DEL

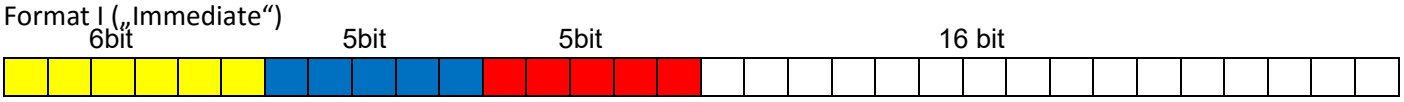
Format R („Register“)



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
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

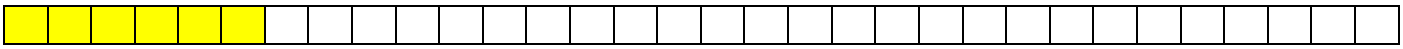
Format I („Immediate“)



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			
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
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 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)
la Rd, Marke		lui Rd, Wert1 ori Rd, Rd, Wert2	Laden der Adresse einer Marke Adresse = (Wert1 Wert2)
b Marke		beq $zero, $zero, Marke	Unbedingter (relativer) Sprung

Format J („Jump“)



Opcode Zieladressenangabe

Opcode	Befehl	Format	Beschreibung
Unbedingte Sprünge			
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