



Computertechnik

In der Prüfung "Computertechnik und Programmieren" ist nur die unten angehängte Formelsammlung zulässig!

1. Zahlen und Zeichen

1.1. Daten als Bits und Bytes

1.1.1. Binärdarstellung

- 1 - High Pegel - logisch wahr

- 0 - Low Pegel - logisch falsch

1.1.2. Hexadezimaldarstellung

Zusammenfassen von 4 Bits zu 0, 1, ..., E, D, F

0: 0000	1: 0001	8: 1000	9: 1001
2: 0010	3: 0021	A: 1010	B: 1011
4: 0100	5: 0101	C: 1100	D: 1101
6: 0110	7: 0111	E: 1110	F: 1111

1.1.3. Bits/Bytes (MMIX)

- 1 Byte = 8 Bits = 1 Byte
- 2 Byte = 16 Bits = 1 Wyde
- 4 Byte = 32 Bits = 1 Tetra
- 8 Byte = 64 Bits = 1 Octa

1.1.4. ASCII-Tabelle

0x0...	0x1...	0x2...	0x3...	0x4...	0x5...	0x6...	0x7...
...0 NUL	DLE	SP	0	@	P	'	p
...1 SOH	DC1	!	1	A	Q	a	q
...2 STX	DC2	"	2	B	R	b	r
...3 ETX	DC3	#	3	C	S	c	s
...4 EOT	DC4	\$	4	D	T	d	t
...5 ENQ	NAK	%	5	E	U	e	u
...6 ACK	SYN	&	6	F	V	f	v
...7 BEL	ETB	.	7	G	W	g	w
...8 BS	CAN	(8	H	X	h	x
...9 HT	EM)	9	I	Y	i	y
..A NL	SUB	*	:	J	Z	j	z
..B VT	ESC	+	;	K	[k	{
..C NP	FS	,	<	L	\	l	
..D CR	GS	-	=	M]	m	}
..E SO	RS	.	>	N	^	n	~
..F SI	US	/	?	O	_	o	DEL

© Michael Zwick

1.2. Zahlen

1.2.1. Vorzeichenlose Festkommazahlen

Definition: $v = (a_{n-1} \cdot b^{n-1} + \dots + a_1 \cdot b^1 + a_0 \cdot b^0) \cdot b^r$

b : Basis (meist 2); n : Stellenzahl; $a \in 0 \dots (b-1)$; r : Position des Kommas

1.2.2. Einer Komplement

Vorzeichen wird durch MSB abgelesen

Umwandlung durch invertieren aller Bits

1.2.3. Zweier Komplement

Vorzeichen wird durch MSB abgelesen

Umwandlung durch invertieren aller Bits und binärer Addition des Werts 1 (von positiv zu negativ)

1.2.4. Gleitkommazahlen nach IEEE 754

Bestehend aus: Vorzeichen (s), Exponentteil (e), Mantisse (f)

Vorgehen (Bsp. 17,625):

1. Zahl vor dem Komma (1,7) und nach dem Komma (0,625) trennen in binär umwandeln
2. Zusammen in binär schreiben (10001,101)
3. Das Komma durch Anpassung des Exponenten $e - K$, verschieben, um das Format $1, f$ zu erhalten ($1,0001101 \cdot 2^4 \rightarrow e - K = 4$)

Definition (normalisiert): $v = (-1)^s \cdot 1, f \cdot 2^{e-K}$

32 Bit:

- $K = 127$
- $s = 1$ Bit
- $e = 8$ Bit
- $f = 23$ Bit

64 Bit:

- $K = 1023$
- $s = 1$ Bit
- $e = 11$ Bit
- $f = 52$ Bit

Spezialfälle (de-normalisiert) $v = (-1)^s \cdot 0, f \cdot 2^{e-K}$:

- 0: $e = 0; f = 0$
- $+\infty$: $e = 1 \dots 1; f = 0; s = 0$
- $-\infty$: $e = 1 \dots 1; f = 0; s = 1$
- Not a Number: $e = 1 \dots 1; f > 0$

2. Arithmetische Schaltungen

2.1. Schaltungselemente

x	y	Inverter	AND	OR	XOR	NAND	NOR
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

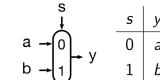
Ein Treiber besteht aus zwei Invertern in Serie geschalten.

2.2. Fan-In/Fan-Out

Fan-In: max. Eingänge in Gatter

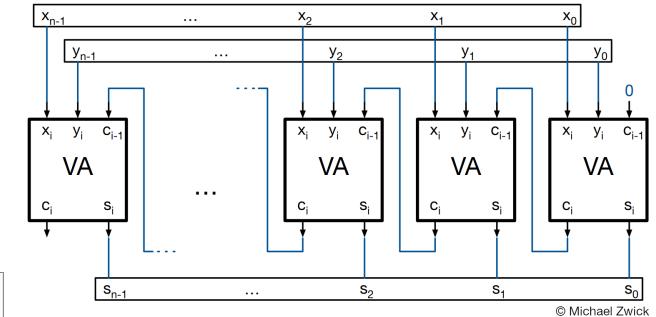
Fan-Out: max. Ausgänge aus Gatter

2.3. Multiplexer



2.4. Ripple-Carry-Addierer

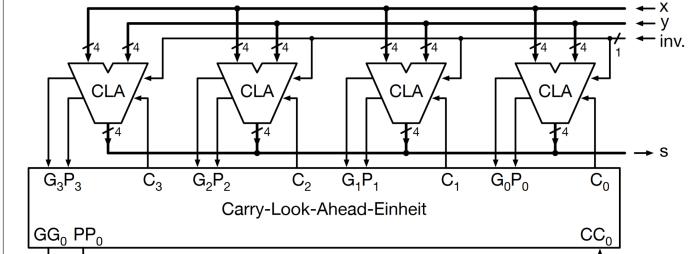
Berechnung der Summe aus zwei Summanden



© Michael Zwick

2.5. Carry-Look-Ahead

Schnellere Berechnung der Summe aus zwei Summanden durch zusätzliche Logik zur Bestimmung des Übertrags

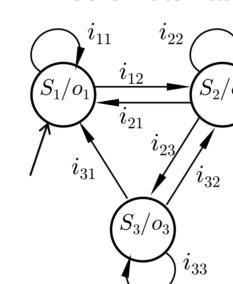


Überlauf (c_i) eines Volladdierers:

$$c_i = g_i \vee (p_i \wedge g_{i-1}) \vee (p_i \wedge p_{i-1} \wedge g_{i-2}) \vee \dots$$

2.6. Zustandsautomat

Moore Automat



Input: Eingangssignale in den Automaten

Output: Takteletriten/Clock-Signale (0 oder 1) und andere Signale (0, 1, X)

2.7. ROM-Steuerung

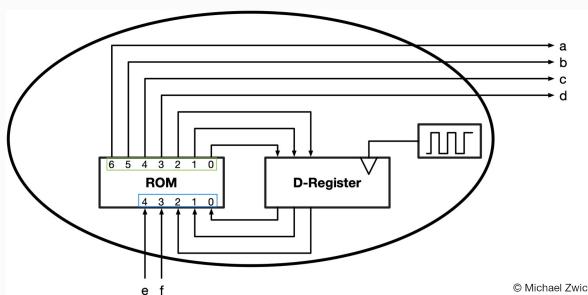


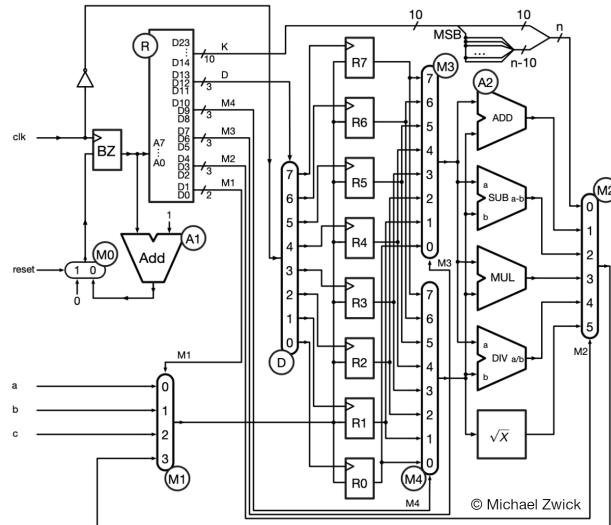
Tabelle mit **Eingabe** (gegeben) und **Ausgabe** (auszufüllen)

Vorgehen: Einträge der gegebenen Tabelle in Binärzahlen umwandeln. Bits 0, 1, 2 der Eingabe sind der derzeitige Zustand. Bits 3, 4, ... der Eingabe sind die Eingangssignale für den nächsten Zustand. Bits 0, 1, 2 der Ausgabe sind der Folgezustand und Bits 3, 4, ... die Ausgangssignale des aktuellen Zustands.

Ansatz:

e f	Zst.	a b c d	Folgezst.
(4 3)	(2 1 0)	(6 5 4 3)	(2 1 0)
...
...

2.8. Universalrechner



Auch hier gilt: Klammer vor Wurzel vor Punkt vor Strich!

- K: Konstante
- D: Ziel-Register
- M1: Ergebnis-Auswahl
- M2: Rechenoperation/Konstante-Laden
- M3: Operand 1 (bei \sqrt{x} jedoch M4 verwenden)
- M4: Operand 2

3. MMIX-Architektur

3.1. Big-Endian und Little-Endian

Big-Endian: höherwertigeres Byte wird adressiert (von links nach rechts)

Little-Endian: niedrigerwertigeres Byte wird adressiert (von rechts nach links)

3.2. Übersetzungstabelle

0x..0	0x..1	0x..2	0x..3	0x..4	0x..5	0x..6	0x..7	
TRAP 5V	FCMP V	FUN V	FEQL V	FADD 4V	FIX 4V	FSUB 4V	FIXU 4V	0x0..
FLOT[I] 4V	FCMPF 4V	FUNE V	FEQLE 4V	SFLOT[I] 4V	SFLOTU[I] 4V			0x0..
FMUL 4V	FCMPF 4V	MULU[I] 10V	MULU[I] 10V	FDIV 40V	FSQRT 40V	FREM 4V	FINIT 4V	0x1..
MULU[I] 10V		ADDU[I] V	SUBU[I] V	DIVU[I] 60V		DIVU[I] 60V		0x1..
ADDU[I] V		SADDU[I] V	SRU[I] V			16ADDU[I] V		0x2..
SADDU[I] V		SLU[I] V	SRU[I] V					0x3..
SLU[I] V		BZ[B] V+π	BP[B] V+π					0x4..
BZ[B] V+π		BNZ[B] V+π	BNP[B] V+π					0x4..
BNZ[B] V+π		PBN[B] 3V-π	PBZ[B] 3V-π	PBP[B] 3V-π	PBOD[B] 3V-π	PBOD[B] 3V-π	PBEB[B] 3V-π	0x5..
PBN[B] 3V-π		PBNZ[B] 3V-π	PBNZ[B] 3V-π	PBNP[B] 3V-π	PBEB[B] 3V-π			0x5..
PBNZ[B] 3V-π		CSN[I] V	CSZ[I] V	CSP[I] V	CSOD[I] V			0x6..
CSP[I] V		CSNN[I] V	CSNZ[I] V	CSNP[I] V	CSEV[I] V			0x6..
CSNN[I] V		ZSN[I] V	ZSZ[I] V	ZSP[I] V	ZSDO[I] V			0x7..
ZSN[I] V		ZSNN[I] V	ZSNZ[I] V	ZSNP[I] V	ZSEV[I] V			0x7..
ZSNN[I] V		LDBU[I] μ+v	LDBU[I] μ+v	LDWU[I] μ+v	LDWU[I] μ+v			0x8..
LDBU[I] μ+v		LDTU[I] μ+v	LDTU[I] μ+v	LDO[I] μ+v	LDOU[I] μ+v			0x8..
LDTU[I] μ+v		LDFTU[I] μ+v	CSWAP[B] 2μ+2V	LDUNC[I] μ+v	LDUNC[B] μ+v			0x9..
LDFTU[I] μ+v		PRELD[I] V	PRREGO[I] V	GO[I] 3V				0x9..
PRELD[I] V		STB[I] μ+v	STBU[I] μ+v	STWU[I] μ+v	STWU[I] μ+v			0xA..
STB[I] μ+v		STTU[I] μ+v	STTU[I] μ+v	STOU[I] μ+v	STOU[I] μ+v			0xA..
STTU[I] μ+v		STSF[I] μ+v	STHT[I] μ+v	STCO[I] μ+v	STUNC[I] μ+v			0xB..
STSF[I] μ+v		SYNC[D] I V	PREST[I] V	SYNCID[I] V	PUSHGO[I] 3V			0xB..
SYNC[D] I V		OR[I] V	ORN[I] V	NOR[I] V	XOR[I] V			0xC..
OR[I] V		AND[I] V	ANDN[I] V	NAND[I] V	NXOR[I] V			0xC..
AND[I] V		BDIF[I] V	WDIF[I] V	TDDIF[I] V	ODIF[I] V			0xD..
BDIF[I] V		MUX[I] V	SADD[I] V	MOR[I] V	MXOR[I] V			0xD..
WDIF[I] V		SETH[V] SETHMH[V]	SETML[V] SETL[V]	INCH[V] INCHM[V]	INCL[V] INCL[V]			0xE..
SETH[V] SETHMH[V]		ORMH[V]	ORL[V]	ANDNMH[V]	ANDNML[V]	ANDNL[V]		0xE..
ORMH[V]		JMP[B] V	PUSHJ[B] V	GETA[B] V	PUT[I] V			0xF..
PUSHJ[B] V		POP 3V	RESUME 5V	[UN]SAVE 20μ+V	SYNC V SWM V	GET V TRIP 5V		0xF..
[UN]SAVE 20μ+V		0x..8	0x..9	0x..A	0x..B	0x..C	0x..D	0x..E
0x..B								0x..F

3.4.2. Laden in Register

Beim Laden wird das Register so aufgefüllt, dass das LSB der geladenen Bitfolge auch das LSB des Registers ist.

ACHTUNG: Das MSB der vordersten Hex-Zahl entscheidet bei signed Operationen ob der Rest links davon mit 0 ... 0 oder F ... F aufgefüllt wird.

3.5. Pipelining

3.5.1. Konflikte

- **Datenkonflikt:** Zugriff auf ein Register das noch gespeichert werden muss
- **Strukturkonflikt:** Zugriff von zwei Befehlen auf die selbe Hardware-Resource
- **Steuerungskonflikt:** Bei Sprungbefehlen werden ggf. darauffolgende Befehle fälschlicherweise geladen, wenn noch nicht klar ist ob gesprungt werden soll oder nicht

3.5.2. Datenpfad

Vorgehen: von rechts nach links die jeweiligen Inhalte der Pipelining-Register eintragen. Dabei ggf. Änderungen von Werten berücksichtigen.

3.6. Cache

3.6.1. Allgemeines

Hit-Rate + Miss-Rate = 100%

Mittlere Zugriffszeit: (Hit-Time) · (Hit-Rate) + (Miss-Time) · (Miss-Rate)

Arbeitspeicher-Adresse: Schlüssel + Rahmen-Nr + Byte-Auswahl

Cache-Inhalt: (Rahmen-Nr + Byte-Auswahl)

Tag-RAM: (Schlüssel + Rahmen-Nr)

Merke: 2^{10} Byte = 1kB und 2^{20} Byte = 1MB

3.6.2. Direkt-Abgebildet

Es existiert pro Rahmen-Nr. nur **ein Rahmen** in dem eine Byte-Folge gespeichert werden kann

Problem: Man kann nicht zwei verschiedene Byte-Folgen mit unterschiedl. Schlüsseln und selber Rahmen-Nr. speichern

3.6.3. Voll-Assoziativ

Eine Bytefolge kann in jedem Rahmen gespeichert werden

Problem: Schlüsselvergleich dauert lange

3.6.4. Set-Assoziativ

Es existieren pro Rahmen-Nr. mehrere Rahmen, in denen eine Byte-Folge gespeichert werden kann (Set)

Vorteil: man kann zwei verschiedene Byte-Folgen mit selber Rahmen-Nr. speichern und ist trotzdem noch relativ schnell

4. Mikrocontroller-Programmierung

4.1. Das Board

Anschluss: 2x USB für Programmierung und Spannungsversorgung

AVR-Programmieradapter

Steckplatine mit V_{dd} und GND Anschläßen

4.2. Übersetzungsbefehle

- Übersetzen: make
- Übersetzen und in den Controller laden: make program
- Übersetzen und Assembler-Code zeigen: make show
- Übersetzte Dateien löschen: make clean
- Taktfrequenz u.a. einstellen: make fuses

4.3. Bitweise Operationen (Bsp: PORTD)

- Bit Nr. 0 setzen: $\text{PORTD} |= 1$ (bitweise ODER)
- Bit Nr. 0 löschen: $\text{PORTD} &= \sim 1$ (bitweise UND)
- Bit Nr. 0 ändern: $\text{PORTD} ^= 1$ (bitweise XOR)

4.4. Wortbreitenunterschiede

Typ	herkömmlicher PC	8-Bit Mikrocontroller
char	1 Byte	1 Byte
short	2 Bytes	2 Bytes
int	4 Bytes	2 Bytes
float	4 Bytes	4 Bytes
double	8 Bytes	4 Bytes
long	8 Bytes	4 Bytes
long long	8+ Bytes	8 Bytes

4.5. Interrupts

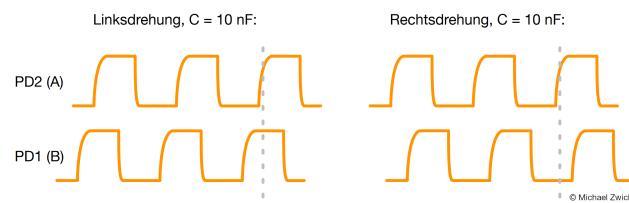
Beim Auftreten eines Interrupts wird aus dem derzeit ausgeführten Code in die Interrupt-Service-Routine (ISR) gesprungen. Dieses Verhalten kann genutzt werden, um z.B. beim Drücken eines Tasters oder nach einer bestimmten Zeit eine Funktion aufzurufen. Dabei muss nicht ständig überprüft werden, ob ein Interrupt auftritt (das springen in die ISR geschieht automatisch).

4.5.1. Interrupt mit Taster

1. Gewünschten Pin als Eingang konfigurieren
2. Interrupts global freischalten
3. Spez. Interrupt im EIMSK-Register freischalten
4. Im EICRA konfigurieren, welcher Pegelzustand des spez. Interrupt berücksichtigt wird (z.B. steigende Flanke)
5. ISR-Funktion schreiben:

```
ISR(INTn_vect) {
    ...
}
```

4.5.2. Drehgeber



2 Pins werden benötigt (ein Interrupt und ein normaler Eingang)

Pin A: Feststellen, dass gedreht wurde (Interrupt).

Pin B: Feststellen, in welche Richtung gedreht wurde (Eingang auslesen).

Zuerst wird in die ISR vom Interrupt (A) gesprungen. Dort wird dann folgendes geprüft:

- hat B High-Pegel? - Dann war es eine Linksdrehung
- hat B Low-Pegel? - Dann war es eine Rechtsdrehung

4.5.3. Timer-Interrupts

Der Zähler TCNT_n wird mit der Frequenz $f = \frac{\text{clk}}{\text{Prescaler}}$ um 1 erhöht.

Überlauf-Interrupt, wenn $\text{TCNT}_n \dots$

- von $2^8 - 1$ auf $2^8 = 256$ erhöht wird (bei $n \in \{0, 2\}$)
- von $2^{16} - 1$ auf $2^{16} = 65536$ erhöht wird (bei $n \in \{1, 3\}$)

1 MMIX-Befehle

Laden aus Speicher

Befehl	Operanden	Name/Aktion	Definition
LDB	\$X, \$Y, \$Z	Load byte	$\$X \leftarrow s'_{64}(s(M_1[u($Y) + u($Z)]))$
	\$X, \$Y, Z	Load byte immediate	$\$X \leftarrow s'_{64}(s(M_1[u($Y) + u(Z)]))$
LDBU	\$X, \$Y, \$Z	Load byte unsigned	$\$X \leftarrow u'_{64}(u(M_1[u($Y) + u($Z)]))$
	\$X, \$Y, Z	Load byte uns. immed.	$\$X \leftarrow u'_{64}(u(M_1[u($Y) + u(Z)]))$
LDW	\$X, \$Y, \$Z	Load wyde	$\$X \leftarrow s'_{64}(s(M_2[u($Y) + u($Z)]))$
	\$X, \$Y, Z	Load wyde immediate	$\$X \leftarrow s'_{64}(s(M_2[u($Y) + u(Z)]))$
LDWU	\$X, \$Y, \$Z	Load wyde unsigned	$\$X \leftarrow u'_{64}(u(M_2[u($Y) + u($Z)]))$
	\$X, \$Y, Z	Load wyde uns. immed.	$\$X \leftarrow u'_{64}(u(M_2[u($Y) + u(Z)]))$
LDT	\$X, \$Y, \$Z	Load tetra	$\$X \leftarrow s'_{64}(s(M_4[u($Y) + u($Z)]))$
	\$X, \$Y, Z	Load tetra immediate	$\$X \leftarrow s'_{64}(s(M_4[u($Y) + u(Z)]))$
LDTU	\$X, \$Y, \$Z	Load tetra unsigned	$\$X \leftarrow u'_{64}(u(M_4[u($Y) + u($Z)]))$
	\$X, \$Y, Z	Load tetra uns. immed.	$\$X \leftarrow u'_{64}(u(M_4[u($Y) + u(Z)]))$
LDO	\$X, \$Y, \$Z	Load octa	$\$X \leftarrow M_8[u($Y) + u($Z)]$
	\$X, \$Y, Z	Load octa immediate	$\$X \leftarrow M_8[u($Y) + u(Z)]$
LDOU	\$X, \$Y, \$Z	Load octa unsigned	$\$X \leftarrow M_8[u($Y) + u($Z)]$
	\$X, \$Y, Z	Load octa uns. immed.	$\$X \leftarrow M_8[u($Y) + u(Z)]$

Schieben

Befehl	Operanden	Name/Aktion	Definition
SL	\$X, \$Y, \$Z	Shift left; with overflow	$\$X \leftarrow u'_{64}(u($Y \ll u($Z)))$ $u($Y) \cdot 2^{u($Z)} \geq 2^{64} \Rightarrow$ $rA \leftarrow rA u'_{64}(2^5)$
	\$X, \$Y, Z	Shift left immediate; with ovf.	$\$X \leftarrow u'_{64}(u($Y \ll u(Z)))$ $u($Y) \cdot 2^{u(Z)} \geq 2^{64} \Rightarrow$ $rA \leftarrow rA u'_{64}(2^5)$
SLU	\$X, \$Y, \$Z	Shift left uns., no overflow	$\$X \leftarrow u'_{64}(u($Y \ll u($Z)))$
	\$X, \$Y, Z	Shift left uns. immed.; no ovf.	$\$X \leftarrow u'_{64}(u($Y \ll u(Z)))$
SR	\$X, \$Y, \$Z	Shift right; fill with sign	$\$X \leftarrow \$Y \gg^s u($Z)$
	\$X, \$Y, Z	Shift right imm.; fill with sign	$\$X \leftarrow \$Y \gg^s u(Z)$
SRU	\$X, \$Y, \$Z	Shift right unsigned; fill with 0	$\$X \leftarrow \$Y \gg^u u($Z)$
	\$X, \$Y, Z	Shift right uns. imm.; fill w. 0	$\$X \leftarrow \$Y \gg^u u(Z)$

Schreiben in Speicher

Befehl	Operanden	Name/Aktion	Definition
STB	\$X, \$Y, \$Z	Store byte; with overflow	$M_1[u($Y) + u($Z)] \leftarrow (\$X)_{7...0}$ $s($X) \geq 2^7 \Rightarrow rA \leftarrow rA u'_{64}(2^6)$ $s($X) < -2^7 \Rightarrow rA \leftarrow rA u'_{64}(2^6)$
	\$X, \$Y, Z	Store byte immed.; ovf.	$M_1[u($Y) + u(Z)] \leftarrow (\$X)_{7...0}$ $s($X) \geq 2^7 \Rightarrow rA \leftarrow rA u'_{64}(2^6)$ $s($X) < -2^7 \Rightarrow rA \leftarrow rA u'_{64}(2^6)$
STBU	\$X, \$Y, \$Z	Store byte unsigned	$M_1[u($Y) + u(Z)] \leftarrow (\$X)_{7...0}$
	\$X, \$Y, Z	Store byte uns. imm.	$M_1[u($Y) + u(Z)] \leftarrow (\$X)_{7...0}$
STW	\$X, \$Y, \$Z	Store wyde; with overflow	$M_2[u($Y) + u($Z)] \leftarrow (\$X)_{15...0}$ $s($X) \geq 2^{15} \Rightarrow rA \leftarrow rA u'_{64}(2^6)$ $s($X) < -2^{15} \Rightarrow rA \leftarrow rA u'_{64}(2^6)$
	\$X, \$Y, Z	Store wyde immed.; ovf.	$M_2[u($Y) + u(Z)] \leftarrow (\$X)_{15...0}$ $s($X) \geq 2^{15} \Rightarrow rA \leftarrow rA u'_{64}(2^6)$ $s($X) < -2^{15} \Rightarrow rA \leftarrow rA u'_{64}(2^6)$
STWU	\$X, \$Y, \$Z	Store wyde unsigned	$M_2[u($Y) + u(Z)] \leftarrow (\$X)_{15...0}$
	\$X, \$Y, Z	Store wyde uns. imm.	$M_2[u($Y) + u(Z)] \leftarrow (\$X)_{15...0}$
STT	\$X, \$Y, \$Z	Store tetra; with overflow	$M_4[u($Y) + u($Z)] \leftarrow (\$X)_{31...0}$ $s($X) \geq 2^{31} \Rightarrow rA \leftarrow rA u'_{64}(2^6)$ $s($X) < -2^{31} \Rightarrow rA \leftarrow rA u'_{64}(2^6)$
	\$X, \$Y, Z	Store tetra immed.; ovf.	$M_4[u($Y) + u(Z)] \leftarrow (\$X)_{31...0}$ $s($X) \geq 2^{31} \Rightarrow rA \leftarrow rA u'_{64}(2^6)$ $s($X) < -2^{31} \Rightarrow rA \leftarrow rA u'_{64}(2^6)$
STTU	\$X, \$Y, \$Z	Store byte unsigned	$M_4[u($Y) + u(Z)] \leftarrow (\$X)_{31...0}$
	\$X, \$Y, Z	Store byte uns. imm.	$M_4[u($Y) + u(Z)] \leftarrow (\$X)_{31...0}$
STO	\$X, \$Y, \$Z	Store octa	$M_8[u($Y) + u($Z)] \leftarrow \$X$
	\$X, \$Y, Z	Store octa immediate	$M_8[u($Y) + u(Z)] \leftarrow \X
STOU	\$X, \$Y, \$Z	Store octa unsigned	$M_8[u($Y) + u($Z)] \leftarrow \$X$
	\$X, \$Y, Z	Store octa uns. imm.	$M_8[u($Y) + u(Z)] \leftarrow \X
Marken auflösen/Adressen laden			
LDA	\$X, \$Y, \$Z	Get address (absolute)	$\$X \leftarrow u'_{64}(u($Y) + u($Z))$
	\$X, \$Y, Z	Get address immed. (absolute)	$\$X \leftarrow u'_{64}(u($Y) + u(Z))$
GETA	\$X, YZ	Get address (relativ)	$\$X \leftarrow u'_{64}(u(@) + 4 \cdot s(YZ))$

Arithmetische Befehle (Festkomma)

Befehl	Operanden	Name/Aktion	Definition
ADD	\$X, \$Y, \$Z	Add; signed, with overflow	$\$X \leftarrow s'_{64}(s(\$Y) + s(\$Z))$ $(s(\$Y) + s(\$Z) < -2^{63}) \quad $ $(s(\$Y) + s(\$Z) \geq 2^{63}) \Rightarrow$ $rA \leftarrow rA u'_{64}(2^5)$
	\$X, \$Y, Z	Add immediate; signed, with overflow	$\$X \leftarrow s'_{64}(s(\$Y) + u(Z))$ $(s(\$Y) + s(Z) < -2^{63}) \quad $ $(s(\$Y) + s(Z) \geq 2^{63}) \Rightarrow$ $rA \leftarrow rA u'_{64}(2^5)$
ADDU	\$X, \$Y, \$Z	Add unsigned; no overflow	$\$X \leftarrow u'_{64}(u(\$Y) + u(\$Z))$
	\$X, \$Y, Z	Add unsigned; no overflow	$\$X \leftarrow u'_{64}(u(\$Y) + u(Z))$
SUB	\$X, \$Y, \$Z	Subtract; signed, with overflow	$\$X \leftarrow s'_{64}(s(\$Y) - s(\$Z))$ $(s(\$Y) - s(\$Z) < -2^{63}) \quad $ $(s(\$Y) - s(\$Z) \geq 2^{63}) \Rightarrow$ $rA \leftarrow rA u'_{64}(2^5)$
	\$X, \$Y, Z	Subtract immediate; signed, with overflow	$\$X \leftarrow s'_{64}(s(\$Y) - u(Z))$ $(s(\$Y) - s(Z) < -2^{63}) \quad $ $(s(\$Y) - s(Z) \geq 2^{63}) \Rightarrow$ $rA \leftarrow rA u'_{64}(2^5)$
SUBU	\$X, \$Y, \$Z	Subtract unsigned; no overflow	$\$X \leftarrow u'_{64}(u(\$Y) - u(\$Z))$
	\$X, \$Y, Z	Subtract unsigned immed.; no ovf.	$\$X \leftarrow u'_{64}(u(\$Y) - u(Z))$
NEG	\$X, Y, \$Z	Negate; signed, with overflow	$\$X \leftarrow s'_{64}(u(Y) - s(\$Z))$ $u(Y) - s(\$Z) \geq 2^{63} \Rightarrow$ $rA \leftarrow rA u'_{64}(2^5)$
	\$X, Y, Z	Negate immediate; signed, with overflow	$\$X \leftarrow s'_{64}(u(Y) - s(Z))$
NEGU	\$X, Y, \$Z	Negate unsigned; no overflow	$\$X \leftarrow s'_{64}(u(Y) - s(\$Z))$
	\$X, Y, Z	Negate unsigned immed.; no overflow	$\$X \leftarrow s'_{64}(u(Y) - u(Z))$
MUL	\$X, \$Y, \$Z	Multiply; signed, with overflow	$\$X \leftarrow s'_{64}(s(\$Y) \cdot s(\$Z))$
	\$X, \$Y, Z	Multiply immediate; signed, with ovf.	$\$X \leftarrow s'_{64}(s(\$Y) \cdot u(Z))$
MULU	\$X, \$Y, \$Z	Multiply unsigned; $rH \leftarrow (\$Y \cdot \$Z)_{127\dots64}$	$rH\$X \leftarrow u'_{128}(u(\$Y) \cdot u(\$Z))$
	\$X, \$Y, Z	Multiply uns. imm.; $rH \leftarrow (\$Y \cdot Z)_{127\dots64}$	$rH\$X \leftarrow u'_{128}(u(\$Y) \cdot u(Z))$
DIV	\$X, \$Y, \$Z	Divide; signed, with overflow (case $\$Z \neq 0$)	$\$X \leftarrow s'_{64}(s(\$Y)/s(\$Z))$ $rR \leftarrow s'_{64}(s(\$Y) \% s(\$Z))$
	\$X, \$Y, Z	Divide immediate; signed, with ovf. (case $\$Z = 0$)	$\$X \leftarrow u'_{64}(0)$ $rR \leftarrow \$Y$

\$X, \$Y, \$Z	Divide unsigned; no overflow; (case $u(\$Z) > u(rD)$)	$\$X \leftarrow u'_{128}(\lfloor u(rD\$Y)/u(\$Z) \rfloor)$ $rR \leftarrow u'_{128}(u(rD\$Y) \% u(\$Z))$
DIVU	no overflow (case $u(\$Z) \leq u(rD)$)	$\$X \leftarrow rD$ $rR \leftarrow \$Y$
\$X, \$Y, Z	Divide unsigned immediate; no overflow; (case $u(Z) > u(rD)$)	$\$X \leftarrow u'_{128}(\lfloor u(rD\$Y)/u(Z) \rfloor)$ $rR \leftarrow u'_{128}(u(rD\$Y) \% u(Z))$
	Divide unsigned immediate; no overflow (case $u(Z) \leq u(rD)$)	$\$X \leftarrow rD$ $rR \leftarrow \$Y$

Arithmetische Befehle (Gleitkomma)

Befehl	Operanden	Name/Aktion	Definition
FADD	\$X, \$Y, \$Z	Floating point add	$\$X \leftarrow f'_{64}(f_{64}(\$Y) + f_{64}(\$Z))$
FSUB	\$X, \$Y, \$Z	Floating point subtract	$\$X \leftarrow f'_{64}(f_{64}(\$Y) - f_{64}(\$Z))$
FMUL	\$X, \$Y, \$Z	Floating point multiplication	$\$X \leftarrow f'_{64}(f_{64}(\$Y) \cdot f_{64}(\$Z))$
FDIV	\$X, \$Y, \$Z	Floating point divide	$\$X \leftarrow f'_{64}(f_{64}(\$Y)/f_{64}(\$Z))$
FSQRT	\$X, \$Z	Square root	$\$X \leftarrow f'_{64}(\sqrt{f_{64}(\$Z)})$

Umwandlung Festkomma <-> Gleitkomma

FLOT	\$X, \$Z	Convert fixed to floating	$\$X \leftarrow f'_{64}(s(\$Z))$
	\$X, Z	Conv. fixed to float. imm.	$\$X \leftarrow f'_{64}(u(Z))$
FLOTU	\$X, \$Z	Conv. uns. fixed to floating	$\$X \leftarrow f'_{64}(u(\$Z))$
	\$X, Z	Conv. uns. fixed to float. imm.	$\$X \leftarrow f'_{64}(u(Z))$
FIX	\$X, \$Z	Convert floating to fixed with overflow	$\$X \leftarrow s'_{64}(r(f_{64}(\$Z)))$ $f_{64}(\$Z) < -2^{63} \Rightarrow rA \leftarrow rA u'_{64}(2^5)$ $f_{64}(\$Z) > 2^{63} - 1 \Rightarrow rA \leftarrow rA u'_{64}(2^5)$
FIXU	\$X, \$Z	Convert floating to fixed without overflow	$\$X \leftarrow s'_{64}(r(f_{64}(\$Z)))$

Spezialregister

GET	\$X, Z	Get value of special purpose register	$\$X \leftarrow r[Z]$
PUT	X, \$Z	Put value to special purpose register	$r[X] \leftarrow \$Z$
	X, Z	Put immed. value to spec. purp. reg.	$r[X] \leftarrow u'_{64}(u(Z))$

Logische Operationen

Befehl	Operanden	Name/Aktion	Definition
AND	\$X,\$Y,\$Z	Bitwise AND	$\$X \leftarrow \$Y \& \$Z$
	\$X,\$Y,Z	Bitwise AND immediate	$\$X \leftarrow \$Y \& u'_{64}(u(Z))$
ANDN	\$X,\$Y,\$Z	Bitwise AND NOT	$\$X \leftarrow \$Y \& \sim \$Z$
	\$X,\$Y,Z	Bitwise AND NOT immed.	$\$X \leftarrow \$Y \& \sim u'_{64}(u(Z))$
ANDNL	\$X,YZ	Bitwise AND NOT low wyde	$\$X \leftarrow \$X \& \sim u'_{64}(u(YZ))$
ANDNML	\$X,YZ	Bw. AND NOT med. l. wd.	$\$X \leftarrow \$X \& \sim (u'_{64}(u(YZ)) \ll 16)$
ANDNMH	\$X,YZ	Bw. AND NOT med. h. wd.	$\$X \leftarrow \$X \& \sim (u'_{64}(u(YZ)) \ll 32)$
ANDNH	\$X,YZ	Bw. AND NOT high wyde	$\$X \leftarrow \$X \& \sim (u'_{64}(u(YZ)) \ll 48)$
NAND	\$X,\$Y,\$Z	Bitwise NOT AND	$\$X \leftarrow \sim (\$Y \& \$Z)$
	\$X,\$Y,Z	Bitwise NOT AND immed.	$\$X \leftarrow \sim (\$Y \& u'_{64}(u(Z)))$
OR	\$X,\$Y,\$Z	Bitwise OR	$\$X \leftarrow \$Y \mid \$Z$
	\$X,\$Y,Z	Bitwise OR immediate	$\$X \leftarrow \$Y \mid u'_{64}(u(Z))$
ORL	\$X,YZ	Bitwise OR low wyde	$\$X \leftarrow \$X \mid u'_{64}(u(YZ))$
ORML	\$X,YZ	Bitwise OR med. low wyde	$\$X \leftarrow \$X \mid (u'_{64}(u(YZ)) \ll 16)$
ORMH	\$X,YZ	Bitwise OR med. high wyde	$\$X \leftarrow \$X \mid (u'_{64}(u(YZ)) \ll 32)$
ORH	\$X,YZ	Bitwise OR high wyde	$\$X \leftarrow \$X \mid (u'_{64}(u(YZ)) \ll 48)$
ORN	\$X,\$Y,\$Z	Bitwise OR NOT	$\$X \leftarrow \$Y \mid \sim \$Z$
	\$X,\$Y,Z	Bitwise OR NOT immediate	$\$X \leftarrow \$Y \mid \sim u'_{64}(u(Z))$
NOR	\$X,\$Y,\$Z	Bitwise NOT OR	$\$X \leftarrow \sim (\$Y \mid \$Z)$
	\$X,\$Y,Z	Bitwise NOT OR immediate	$\$X \leftarrow \sim (\$Y \mid u'_{64}(u(Z)))$
XOR	\$X,\$Y,\$Z	Bitwise XOR	$\$X \leftarrow \$Y \otimes \$Z$
	\$X,\$Y,Z	Bitwise XOR immediate	$\$X \leftarrow \$Y \otimes u'_{64}(u(Z))$
NXOR	\$X,\$Y,\$Z	Bitwise NOT XOR	$\$X \leftarrow \sim (\$Y \otimes \$Z)$
	\$X,\$Y,Z	Bitwise NOT XOR immediate	$\$X \leftarrow \sim (\$Y \otimes u'_{64}(u(Z)))$

Funktionsaufruf

Befehl	Operanden	Name/Aktion	Definition
GO	\$X,\$Y,\$Z	Go to location	$\$X \leftarrow u'_{64}(u(\circ) + 4);$ $\circ \leftarrow u'_{64}(u(Y) + u(Z))$
	\$X,\$Y,Z	Go to location immediate	$\$X \leftarrow u'_{64}(u(\circ) + 4);$ $\circ \leftarrow u'_{64}(u(Y) + u(Z))$

Verzweigungen

Befehl	Operanden	Name/Aktion	Definition
JMP	XYZ	Jump	$\circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(XYZ))$
BZ	\$X,YZ	Branch if zero	$s(\$X) = 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
PBZ	\$X,YZ	Probable br. if zero	$s(\$X) = 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
BNZ	\$X,YZ	Branch if nonzero	$s(\$X) \neq 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
PBNZ	\$X,YZ	Prob. br. if nonzero	$s(\$X) \neq 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
BN	\$X,YZ	Branch if negative	$s(\$X) < 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
PBN	\$X,YZ	Prob. br. if negative	$s(\$X) < 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
BNN	\$X,YZ	Branch if nonneg.	$s(\$X) \geq 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
PBNN	\$X,YZ	Prob. br. if nonneg.	$s(\$X) \geq 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
BP	\$X,YZ	Branch if positive	$s(\$X) > 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
PBP	\$X,YZ	Prob. br. if positive	$s(\$X) > 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
BNP	\$X,YZ	Branch if nonpositive	$s(\$X) \leq 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
PBNP	\$X,YZ	Prob. br. if nonpos.	$s(\$X) \leq 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
BEV	\$X,YZ	Branch if even	$s(\$X) \% 2 = 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
PBEV	\$X,YZ	Prob. branch if even	$s(\$X) \% 2 = 0 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
BOD	\$X,YZ	Branch if odd	$s(\$X) \% 2 = 1 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$
PBOD	\$X,YZ	Prob. branch if odd	$s(\$X) \% 2 = 1 \Rightarrow \circ \leftarrow u'_{64}(u(\circ) + 4 \cdot s(YZ))$

Direktoperanden

SETL	\$X,YZ	Set to low wyde	$\$X \leftarrow u'_{64}(u(YZ))$
SETML	\$X,YZ	Set to med. low wyde	$\$X \leftarrow u'_{64}(u(YZ \ll 16))$
SEMH	\$X,YZ	Set to med. high wyde	$\$X \leftarrow u'_{64}(u(YZ \ll 32))$
SETH	\$X,YZ	Set to high wyde	$\$X \leftarrow u'_{64}(u((YZ \ll 48)))$
INCL	\$X,YZ	Increase by low wyde	$\$X \leftarrow u'_{64}(u(\$X) + u(YZ))$
INCML	\$X,YZ	Increase by med. low wyde	$\$X \leftarrow u'_{64}(u(\$X) + u(YZ \ll 16))$
INCMH	\$X,YZ	Increase by med. high wyde	$\$X \leftarrow u'_{64}(u(\$X) + u(YZ \ll 32))$
INCH	\$X,YZ	Increase by high wyde	$\$X \leftarrow u'_{64}(u(\$X) + u(YZ \ll 48))$

2 Mikrocontroller-Programmierung

Assembler und Loader-Befehle:

IS - Register umbenennen

GREG - Globales Register anlegen

LOC - Einfügemarken setzen

BYTE - 8 Bit im Speicher anlegen

WYDE - 16 Bit im Speicher anlegen

TETRA - 32 Bit im Speicher anlegen

OCTA - 64 Bit im Speicher anlegen

@ - the place where we are at;

Bei Befehlen: aktueller Wert des Befehlszählers

Bei Loader-Befehlen: aktuelle Position der Einfügemarken

2.1 Ein- und Ausgabe über Ports

- **DDRn** ist das sog. *Data Direction Register* für Port n . Die einzelnen Bits dieses Registers legen für den entsprechenden Pin i des Ports n fest, ob dieser als Eingang (Bit i in DDRn ist gelöscht, d.h. 0) oder als Ausgang (Bit i in DDRn ist gesetzt, d.h. 1) dienen soll.

Bit-Nr. i:	7	6	5	4	3	2	1	0
DDRn7	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

- Ein gesetztes Bit i in Register DDRn konfiguriert Pin i des Ports n als Ausgang. Am entsprechenden Prozessor-Pin liegt dann der mit Bit i in Register PORTn ($n \in \{A, B, C, D\}$) spezifizierte Pegel (0 oder 1) an.
- Ein gelöschtes Bit in DDRn konfiguriert den entsprechenden Pin als Eingang.
 - Ist das entsprechende Bit im PORTn-Register gelöscht, so ist der Eingang hochohmig (tristate).
 - Ist das entsprechende Bit im PORTn-Register gesetzt, so wird der Pin über einen Widerstand (5 - 50 k) auf High-Pegel gezogen.
- **PORTn** ist das sog. *Port Register* für Port n .

Bit-Nr. (i):	7	6	5	4	3	2	1	0
PORTn7	0	0	0	0	0	0	0	0
Wert nach Reset:	0	0	0	0	0	0	0	0

- Im Fall $DDRni = 1$ (Bit i des Ports n ist als *Ausgang* konfiguriert) wird über PORTni der Ausgangspegel (1 = High = 5 V; 0 = Low = 0 V) des entsprechenden Ausgangs-Pin des Prozessors festgelegt.

- Im Fall DDR_n = 0 (Bit i des Ports n ist als *Eingang* konfiguriert) wird über PORT_ni für Bit i des Ports n festgelegt, ob der Pull-Up-Widerstand aktiviert wird (PORT_ni = 1) oder nicht (PORT_ni = 0).
 - Die **Aktivierung des Pull-Up-Widerstands** sorgt dafür, dass der entsprechende Pin über einen hohen Widerstand auf Vcc gezogen wird.
 - Ist der **Pull-Up-Widerstand nicht aktiviert**, liegt der Pin auf unbestimmtem Potential und muss durch externe Beschaltung (z.B. externer Pull-Up/Down-Widerstand) in einen sinnvollen Zustand gebracht werden.
- **PINn** ist das sog. *Pin-Register* für Port n. Aus dem Pin-Register lassen sich über die Bits PIN_ni die Pegel (High/Low) auslesen, die am entsprechenden Pin anliegen.

Bit-Nr. (i):	7	6	5	4	3	2	1	0
Wert	PINn7	PINn6	PINn5	PINn4	PINn3	PINn2	PINn1	PINn0
nach Reset:	0	0	0	0	0	0	0	0

2.2 Interrupts allgemein

- **SREG** ist das Status-Register des Mikrocontrollers, über das Eigenschaften des Ergebnisses arithmetischer Operationen angezeigt wird: C = Carry, Z = Zero, N = Negative, V = Overflow, S = Sign, H = Half-Carry (BCD (Binary Coded Decimal)-Arithmetik). Das T-Bit wird als Operand für Bit-Operationen verwendet. Über das I-Bit können Interrupts global freigeschaltet werden.

Bit-Nr. (i):	7	6	5	4	3	2	1	0
Wert	I	T	H	S	V	N	Z	C
nach Reset:	0	0	0	0	0	0	0	0

1.3 Externe Interrupts

- **EIMSK** ist das sog. *External Interrupt Mask Register*, über das die externen Interrupts freigeschaltet werden können. Ein gesetztes Bit an Position i schaltet den externen Interrupt Nr. i frei. Es können auch mehrere externe Interrupts gleichzeitig freigeschaltet werden.

Bit-Nr. (i):	7	6	5	4	3	2	1	0
Wert	-	-	-	-	-	INT2	INT1	INT0
nach Reset:	0	0	0	0	0	0	0	0

- **EICRA** ist das sog. *External Interrupt Control Register A*. Es legt für jeden externen Interrupt n ∈ {0, 1, 2} über die zwei Bits ISCn1 bzw. ISCn0 fest, ob
 - der Logikpegel 0 (Low) den Interrupt auslöst (00),
 - eine Pegel-Änderung den Interrupt auslöst (01),
 - ein fallender Pegel den Interrupt auslöst (10), oder
 - ein steigender Pegel den Interrupt auslöst (11)

Bit-Nr. (i):	7	6	5	4	3	2	1	0
Wert	-	-	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00
nach Reset:	0	0	0	0	0	0	0	0

- **EIFR** ist das sog. *External Interrupt Flag Register*. Es zeigt für den externen Interrupts INT_n über das Bit INTF_n an, ob das entsprechende Interrupt-Ereignis (z.B. steigende/fallende Flanke) des entsprechenden externen Interrupts vorliegt (High-Pegel) oder nicht (Low-Pegel). Die ver-UND-ung dieser Bits mit dem I-Bit aus Register SREG und den entsprechenden Bits INT_n, n ∈ {2, 1, 0} im EIMSK-Register führt zu einem Sprung an die entsprechende Position der Interrupt-Vektor-Tabelle.

Bit-Nr. (i):	7	6	5	4	3	2	1	0
Wert	-	-	-	-	-	INTF2	INTF1	INTF0
nach Reset:	0	0	0	0	0	0	0	0

1.4 Timer-Interrupts

- **TIMSK_n** mit $n \in \{0, 1, 2, 3\}$ ist das Timer/Counter Interrupt Mask Register für Timer n. Hier kann über Bit 0 der Interrupt für den Timer-Überlauf freigeschaltet werden.

- ICIE: Nur bei Timer 1 und 3 verfügbar; wird bei uns nicht verwendet
- OCIEB: Bei uns nicht verwendet
- OCIEA: Bei uns nicht verwendet
- TOIE: Schaltet den Timer-Überlauf-Interrupt für Timer n frei.

Bit-Nr. (j):	7	6	5	4	3	2	1	0
Wert nach Reset:	-	-	ICIE	-	-	OCIEB	OCIEA	TOIE

- **TCCR_{nB}** mit $n \in \{0, 1, 2, 3\}$ ist das Timer Counter Control Register B für Timer n. Die einzelnen Bits der TCCR_{nB}-Register haben bei den beiden 8 Bit Timern teilweise andere Funktionen als bei den beiden 16 Bit Timern:

TCCR_{nB} bei 8 Bit Timern ($n \in \{0, 2\}$):

Bit-Nr. (j):	7	6	5	4	3	2	1	0
Wert nach Reset:	FOC0A	FOC0B	-	-	WG _{Mn2}	CS _{n2}	CS _{n1}	CS _{n0}

TCCR_{nB} bei 16 Bit Timern ($n \in \{1, 3\}$):

Bit-Nr. (j):	7	6	5	4	3	2	1	0
Wert nach Reset:	ICNC _n	ICES _n	-	WG _{Mn3}	WG _{Mn2}	CS _{n2}	CS _{n1}	CS _{n0}

In beiden Fällen kann jedoch die Taktquelle (inkl. Prescaler) über die Bits 2, 1 und 0 eingestellt werden. Für die angegebenen Werte von [CS_{n2}, CS_{n1}, CS_{n0}] ergeben sich folgende Einstellungen:

CS _{n2/1/0}	TCCR0B	TCCR1/3B	TCCR2B
000	Timer gestoppt	Timer gestoppt	Timer gestoppt
001	clk	clk	clk
010	clk/8	clk/8	clk/8
011	clk/64	clk/64	clk/32
100	clk/256	clk/256	clk/64
101	clk/1024	clk/1024	clk/128
110	Fallende Flanke an ext. Pin T0	Fallende Flanke an ext. Pin T0	clk/256
111	Steigende Flanke an ext. Pin T0	Steigende Flanke an ext. Pin T0	clk/1024

- **TCNT_n** mit $n \in \{0, 1, 2, 3\}$ ist das Timer Counter Register für Timer n. Register TCNT_n enthält den Zähl-Wert des Timers n, der beim Auftreten des ausgewählten Takt-Ereignisses verändert/erhöht wird. 16 Bit breite Register ($n \in \{1, 3\}$) setzen sich aus zwei 8 Bit breiten Registern TCNT_{nH} und TCNT_{nL} zusammen.