

Lemps12: Programmierung in C

Die grundlegenden Syntaxelemente von C sind hier aufgelistet und sollen dem Einsteiger helfen, sich rasch mit C zurecht zu finden. Diese Zusammenfassung ist als Arbeitsunterlage und Nachschlagwerk gedacht. Neben den Standard C-Elementen und Funktionen sind hier auch die im Lemps12 Bios vorhandenen Funktionen beschrieben. Die Angaben gelten für Standard C und den ICC12 C-Compiler von Imagecraft.

Für die HC12 Programmierung gibt es im Internet einen sehr guten Lehrgang von **Jonathan W. Valvano**.

Inhalt	
Inhalt	1
Literaturhinweise	1
C Lehrgänge im Internet	1
Anwendung des ICC12 Compilers	2
Allgemeines zu C	2
Darstellung von Zahlen und Zeichen	2
Escape Sequenzen	2
Typen	3
Typen Umwandlung (Casts)	3
Pointer (Zeiger)	3
Einwert Operatoren	4
Arithmetische Operatoren	4
Logische Operatoren	4
Vergleichsoperatoren	4
Zuordnungsoperatoren	5
Reservierte Ausdrücke	5
Bindungsprioritäten	6
Formatierung von printf, scanf	6
Programmaufbau und Strukturen	7
ICC12 V5.0 Funktionen	11
math.h	11
stdio.h	11
stdlib.h	12
ctype.h	12
string.h	13
HC12 Registernamen (hc12.h)	14
Lemps12 Lbios Funktionen (lbios.h)	14
ASCII Tabelle	17

Dez-, Hex- und Dualcode	17
Formatierung der Lbios Funktion Write	18
HC12 Interruptbezeichnungen	18

Literaturhinweise

Programmieren in C
Kernighan/Ritchie
ISBN 3-446-15497-3

C Programmieren von Anfang an
Helmut Erlenkötter
ISBN 3-499-60074-9

C++ für Dummies
Nimir Clement Shammass
ISBN 3-8266-2775-X

ICC12 Benutzer Handbuch
Image Craft
MCT Lange&Thamm Microcomputertechnik D

Introduction to Microcontrollers
G.Jack Lipovski
ISBN 0-12-451831-1

Essentila Guide to Turbo C
Naba Barkakati
ISBN 0-672-22675-8

Turbo C
Alexander Janson
ISBN 3-7723-6202-8

C Lehrgänge im Internet

<http://www.ece.utexas.edu/~valvano/embed/toc1.htm>

<http://www.graylab.ac.uk/doc/tutorials/C/>

<http://www.cm.cf.ac.uk/Dave/C/CE.html>

<http://www.eskimo.com/~scs/c/class/c/class.html>

<http://www.andromeda.com/people/ddyer/topten.html>

Anwendung des ICC12 Compilers

C-Programm in einem Texteditor (Lemps12 Ide) **schreiben**. Vorlage Menue Datei verwenden. File-Extension .C z.B. *MyProg.C*

Datei- und Verzeichnisnamen müssen den Regeln von DOS entsprechen! Nicht länger als 8 Zeichen, keine Umlaute und keine Zwischenräume!!

File **compilieren und linken**. Der Linker erstellt ein File mit der Extension .S19 z.B. *MyProg.S19*. Diese Operationen werden mit der Taste *lcc12* in der L12 Ide ausgeführt.

Das File *MyProg.S19* in das Lemps12 **laden**. (Download .S19)

Das Programm im Lemps12 **starten** (mit dem Token *goto 2000*)

Allgemeines zu C

Die Programmiersprache C ist Case-Sensitiv. d.h. es wird immer zwischen **Gross- und Kleinschreibung unterschieden!**

- C ist formatfrei. Innerhalb von Programmweisungen werden Zeilenumbrüche und Zwischenräume ignoriert.
- C besteht nur aus Funktionen. Die Funktion **main** muss in jedem Programm enthalten sein und wird nach dem Programmstart zuerst ausgeführt.
- Jeder Funktionscode (Block) beginnt mit { und endet mit }
- Jede C-Anweisung muss mit einem Semikolon ; abgeschlossen werden
- Kommentare beginnen mit /* und enden mit */. Solche Kommentare können sich über mehrere Zeilen erstrecken. In C++ leiten die beiden Zeichen // einen Kommentar bis zum Ende der Zeile ein (der ICC12 Compiler kann so eingestellt werden, dass er diese Kommentarangabe auch erkennt).

Vergleichen Sie auch den Abschnitt **Aufbau eines C-Programms**.

Darstellung von Zahlen und Zeichen

Bezeichnung	Beispiele
Dezimal	100, 55, -12, 102203
Oktal	01234, 022, 0745 (vorgestellte 0 bedeutet Oktal)
Hexadezimal	0xFF, 0xAFFE, 0x22 (vorgestellt 0x bedeutet Hex)
Boolean (in C kein spezieller Zahlentyp)	Zahl = 0 bedeutet FALSE Zahl != 0 bedeutet TRUE (!= bedeutet ungleich)
Ascii Zeichen (Charakter)	'A', '5', 'b', '&', '=', '\n', '\x20'
Zeichenketten (Stings)	"hallo", "guten Tag", "23.Januar\n"

Escape Sequenzen

Escape Sequenzen können als Charkter oder in Strings wie ein ASCII-Zeichen eingesetzt werden. So bewirkt z.B. ein \n einen Zeilenvorschub. Escape Sequenzen werden hauptsächlich zusammen mit Strings eingesetzt.

Vergleichen Sie auch die Tabelle mit allen ASCII Zeichen.

Sequ	Name	Hex	Dez
\n	newline, linefeed	0A	10
\t	Tab (Tabulator)	09	9
\b	backspace	08	8
\f	formfeed	0C	12
\a	Bell (alarm)	07	7
\r	return	0D	13
\v	vertical tab	0B	12
\0	null	00	0
\"	ASCII quote	22	34
\\	ASCII back slash	5C	92
'	ASCII single quote	27	39
\?	Fragezeichen	5F	95
\x40	ASCII Zeichen 40h (@) einfügen, usw.		

Typen

Bezeichnung	Bereich	Bytes
char	-128..+127	1
unsigned char	0..255	1
int short	-32768..+32767	2
unsigned int unsigned short	0..65535	2
long	-2147483648.. +2147483647	4
float double	1.17549E-38 3.40282E+38	4

Zum oben beschriebenen Typ kann noch eine weitere Definition hinzukommen:

Bez.	Bedeutung
volatile	Kann den Inhalt auch unabhängig vom Programm ändern (z.B. Port)
const	Fester Wert, kann den Wert zur Laufzeit nicht ändern
unsigned	Nur positive Zahlen inkl. 0
signed	Bereicherstreckt sich über positive und negative Zahlen
auto	Automatisch, auf dem Stack untergebracht
extern	In einem anderen Programmfile definiert
static	Bleibt während der ganzen Programmablaufzeit bestehen

Hier noch einige Beispiele wie Variablen deklariert werden können:

```
/* Deklaration von Variablen */
unsigned char MyVar;
char MyArray[20];
unsigned int i,n,k;

/*Deklarationen mit Anfangswert*/
char Ziffer = 'A';
unsigned int Faktor = 0xAFFE;
char MyString[]="hallo";

/* Konstanten*/
const int ANZWD = 7;
const short DAT[] = {10,9,-922};
const char *WDAYS[] =
    {"MO","DI","MI","DO","FR"};
```

Typen Umwandlung (Casts)

Der C-Compiler wandelt selber Daten von einem Typ in den anderen um, wenn dies erforderlich ist.

Mit Casts zwingen Sie den Compiler an einer bestimmten Stelle eine solche Umwandlung durchzuführen. Dabei wird der Typ in Klammern dem Ausdruck vorgestellt. Beispiele:

```
int MyI1=3,MyI2=5;
float Res;

Res=MyI1/MyI2; //Res=0
Res=(float)MyI1/MyI2; //Res=0.6
Res=MyI1/(float)MyI2; //Res=0.6
Res=(float)(MyI1/MyI2); //Res=0
Res=3/5; //Res=0
```

Pointer (Zeiger)

Für jede Variable wird im Speicher des Microcontroller Systems ein Platz mit einer Adresse reserviert. Diese Adresse wird in C als Pointer (Zeiger) bezeichnet. In einem Microcontrollersystem (Lemps12) wird meistens mit 2 Byte grossen Adressen gearbeitet. Die Adresse eines Ausdrucks kann mit dem &-Operator ermittelt werden

```
unsigned int Res, Zahl;

Res=PORTJ //Res=Inhalt PortJ
Res=&PORTJ //Res=Adresse PortJ
```

In diesem Fall ist jedoch Res eine Variable und kein Pointer! Ein eigentlicher Pointer muss mit dem *-Operator als solcher definiert werden:

```
int *Ptr; /* Deklariert einen
Zeiger Ptr der auf
einen int zeigt */
Ptr=&Zahl; /* Nun zeigt Ptr auf
die int Var Zahl */
*Ptr=0x55; /* Inhalt der Adr. Ptr
und damit die Var Zahl
hat den Wert 0x55 */
```

Einem Pointer kann nicht direkt ein Zahlenwert zugeordnet werden. Pointer können jedoch inkrementiert und dekrementiert werden!

Syntax	Bedeutung
&MyVar	Liefert die Adresse von MyVar
int *MyPtr	Deklariert den Pointer MyPtr
MyVar=*MyPtr	Übergibt Inhalt von MyPtr

Einwert Operatoren

Diese Operatoren wirken auf einen Wert und ergeben ein Resultat. In den untenstehenden Beispielen werden folgende Variablen verwendet:

```
int Wert /* -32768..+32767 */
int *Zeiger /* Zeiger auf Speicher*/
int Bool /* 0=false, sonst=true*/
```

Op	Funktion	Darst.	Beispiel, Resultat
~	Komplement	~Wert	~0xAA = 0x55
!	Logisches Komplement	!Bool	!5=0; !100=0; !0=1; !1=0;
&	Adresse von	&Wert	Gibt die Adresse der Variablen Wert an
-	negativ	-Wert	-(-100)=100
+	positiv	+Wert	+100=100
++	Pre-increment	++Wert	Wert vor Operation incrementieren
++	Post-increment	Wert++	Wert nach Operation incrementieren
--	Pre-decrement	--Wert	Wert vor Operation decrementieren
--	Post-decrement	Wert--	Wert nach Operation decrementieren
*	Referenz	*Zeiger	Inhalt auf welchen der Zeiger verweist

Arithmetische Operatoren

Bei der Durchführung von arithmetischen Operationen ist unbedingt darauf zu achten, dass die Zahlenbereiche nicht überschritten werden (dies gilt auch für Zwischenergebnisse!). Auch Operationen zwischen 'signed' und 'unsigned' Variablen können zu Berechnungsfehlern führen. Es ist von Vorteil, für arithmetische Operationen vorzeichen-behaftete Variablen zu verwenden.

Beachten Sie, dass der Divisionsoperator für den Typ **int** und **float** derselbe ist. Bei der Integerdivision werden die Stellen hinter dem Komma abgeschnitten (keine Rundung). Mit der Eingabe einer

Operation	Res.	Bemerkung
15/2	7	Integerdivision
15.0/2	7.5	Floatdivision
(float)15/2	7.5	Floatdivision (Cast Op.)

Kommastelle oder dem Cast-Operator kann eine Floatdivision erreicht werden.

Op.	Funktion	Beispiel
+	Addition	20+0x10=36
-	Subtraktion	40-60=-20
*	Multiplikation	10*25=250
/	Division	159/10=15
%	Modulo (Rest)	159/10=9
<<	Links schieben	24<<2=96
>>	Rechts schieben	24>>2=6

Logische Operatoren

Mit den **bitweise** wirkenden logischen Operatoren können mit einer Maske innerhalb eines Bytes oder Wortes einzelne Bits gesetzt (OR), gelöscht (AND) oder invertiert (EXOR) werden.

Op.	Funktion	Zahlenbeispiel
&	Bitweise AND	0xAA & 0x0F ergibt 0x0A
	Bitweise OR	0xAA 0x0F ergibt 0xAF
^	Bitweise EXOR	0xAA ^ 0x0F ergibt 0xA5
~	Bitweise NOT	!0xAA ergibt 0x55

Mit den folgenden logischen Operatoren werden boolesche Ausdrücke in Vergleichen zusammen verknüpft. Ein Wert (int, char) wird als FALSE angenommen wenn er gleich 0 ist. Ist ein Wert ungleich 0 wird er als TRUE angenommen.

Op.	Funktion	Beispiel
&&	AND	25 && 0 = 0 (FALSE) 25 && 3 = 1 (TRUE)
	OR	25 0 = 1 (TRUE) 0 0 = 0 (FALSE)
!	NOT	!25 = 0; !0 = 1

Vergleichsoperatoren

Vergleichsoperatoren werden in Entscheiden zum Verzweigen und in Schleifen als Abbruchkriterien eingesetzt. Oft müssen mehrere Vergleiche zusammen ein bestimmtes Resultat ergeben. In solchen Fällen können einzelne Vergleiche mit AND (&&), OR (||) und NOT (!) kombiniert werden. Das Resultat ist immer 1 (TRUE) oder 0 (FALSE).

Op.	Funktion	Beispiel
==	gleich	100 == 4 (0, FALSE) 12 == 12 (1, TRUE)
!=	ungleich	100 != 4 (1, TRUE) 12 != 12 (0, FALSE)
<	kleiner	100 < 4 (0, FALSE) 12 < 12 (0, FALSE)
<=	kleiner gleich	100 <= 4 (0, FALSE) 12 <= 12 (1, TRUE)
>	grösser	100 > 4 (1, TRUE) 12 > 12 (0, FALSE)
>=	grösser gleich	100 >= 4 (1, TRUE) 12 >= 12 (1, TRUE)

Zuordnungsoperatoren

Zuordnungsoperatoren weisen einer Variablen eine Konstante oder den Inhalt einer anderen Variablen zu.

Operation	Ergebnis
A=5	Zahl 5 wird der Var A zugewiesen
A=B=C=4	A,B,C werden auf 4 gesetzt
A=B=C	A, B sind gleich wie C
A+=B	A=A+B
A-=B	A=A-B
A*=B	A=A*B
A/=B	A=A/B
A<<=B	A=A<<B
A>>=B	A=A>>B
A&=B	A=A&B
A =B	A=A B
A^=B	A=A^B

```

/* Beispiele für Zuordnungen: */
DDRJ=0xFF /* PORTJ Ausgang */
PORTJ|=0x01 /* PJ0 auf 1 */
PORTJ&=~0x80 /* PJ7 auf 0 */
PORTJ^=0x0F /* PJ0..3 invers */
PORTJ=PORTD=0 /* PJ, PD auf 0 */
PORTJ<<=4 /* PJ 4 * 1s1 */
    
```

Reservierte Ausdrücke

Die hier aufgeführten Ausdrücke haben in C eine spezielle Funktion und dürfen nicht für Namen (Variablen, Funktionen) eingesetzt werden.

Ausdruck	Bedeutung
asm	Assembler Code einfügen
auto	Definiert Variable auf Stack
break	Bricht Programmablauf innerhalb einer Struktur ab
case	Pfad in einer Mehrfachverzweigung
char	8 Bit Integer Deklaration
const	Konstantendeklaration (ROM)
continue	Gehe zum Loop-Anfang
default	Mehrfachverzweigung: Pfad wenn keine andere Bedingungen zutrifft
do	do..while Schlaufe
double	Deklaration Var mit Gleitkomma
else	Verzweigung alternativer Pfad
extern	In einem anderen File definiert
float	Deklaration Var mit Gleitkomma
for	Schlaufe mit def. Anzahl Durchläufe
goto	Spring zu einem bestimmten Label
if	Einfache Verzweigung
int	16Bit Integer (wie short beim HC12)
long	32Bit Integer
register	Spezifiziert eine locale Var
return	Parameterrückgabe in Funktion
short	16Bit Integer
signed	Var.-Deklaration vorzeichenbehaftet
sizeof	Gibt Grösse eines Objekts zurück
static	Var.-Deklaration immer im Speicher
struct	Definition einer Datenstruktur
switch	Mehrfachverzweigung
typedef	Definition von neuen Datentypen
unsigned	Nicht vorzeichenbehaftet >=0
void	Kein Parameter wird übergeben
volatile	Kann unabhängig vom Prog. ändern
while	While Schlaufe

Bindungs-Prioritäten

In der Mathematik bindet die Multiplikation Multiplikation stärker als die Addition (Punkt- vor Strichrechnung). In C gibt es verschiedene Stufen von Operationsprioritäten, die in der folgenden Tabelle in abnehmender Bindungsstärke (von oben unten) dargestellt sind.

Op. Gruppe	Operatoren	Reihenfolge
Expression	() [] . ->	links nach rechts
Unary	- + ~ ! * & ++ --	rechts nach links
Multipl.	* / %	links nach rechts
Additive	+ -	links nach rechts
Shift	<< >>	links nach rechts
Relational (inequality)	< <= > >=	links nach rechts
Relational (equality)	== !=	links nach rechts
Bitwise And	&	links nach rechts
Bitwise Xor	^	links nach rechts
Bitwise Or		links nach rechts
Logical And	&&	links nach rechts
Logical Or		links nach rechts
Conditional	? :	rechts nach links
Assignment	= *= /= %= += -= <<= >>= &= = ^=	rechts nach links

Formatierung von printf, scanf

Printf und scanf sind die wichtigsten I/O Funktionen in C.

printf gibt formatierten Text und Zahlenwerte an das 'Standard output file' (Bildschirm) aus.

Beispiel:

```
printf("PJ=%x,PD=%x\n",PORTJ,PORTD);
```

Ausgabe am Bildschirm:
PJ=004F,PD=003B

%x Platzhalter an dessen Stelle der aktuelle Portzustand im angegebenen Format eingesetzt wird. \n bewirkt eine neue Zeile nach der Ausgabe

scanf liest Charakter vom 'Stream' (meistens Tastatur) ein, und weist die Daten entsprechend den Formatierungszeichen den C Variablen zu.

Beispiel:

```
scanf ("%f",&Var_U);
```

Auch scanf braucht einen Formatstring mit dem Format der Zahl die eingelesen werden soll. Der eingelesene Wert wird dann der Variablen übergeben deren Adresse nach dem Formatstring angegeben wird.

Die Kombination %f ist ein Platzhalter für eine einzusetzende Float-Zahl. Die wichtigsten Formate für solche Platzhalter sind in der folgenden Tabelle aufgeführt.

Format	Typ	Bemerkung
%c	char	Einzelner Charakter
%d	int	Integer dezimal
%e	float	Signed Wert im ENG Format
%f	float	Signed Wert in exponential Darstellung
%i	int	Integer (signed)
%s	char string	Zeichenkette
%u	int	Unsigned integer Darstellung
%x	int	Hexadezimale Ausgabe

Fließkomma-Formatierung:

Unmittelbar nach dem % Zeichen kann mit dem Zeichen # die Fließkomma-Ausgabe noch formatiert werden:

```
##[W].[P]f
```

[W] : Eine Zahl die angibt, in wie vielen Stellen die Ausgabe im Minimum erfolgen soll (optional).

[P] : Eine Zahl die angibt, wie viele Stellen im Maximum nach dem Dezimalpunkt ausgegeben werden sollen (optional).

Beispiel:

```
Var_U = 230.56;
printf ("U=%#8.1f v",Var_U);
```

Ausgabe am Bildschirm:
U= 230.6 v

Programmaufbau und Strukturen

```

/* C - Beispielprogramm (Cstruk.c)
=====

Beim Drücken der Taste T7 wird das PortJ mit einem
dreimaligen Beep incrementiert. Hardware: L12 mit
PortJ auf LEDs und PortD auf Taster verbunden.
*/

/* Einbinden von bestehenden Modulen */
#include <stdio.h> /* C Standard I/O-Funktionen */
#include <hc12.h> /* Registerdefinitionen HC12 */
#include <lbios.h> /* Interface zu LBIOS */

/* Definitionen: */
/* 'Led' steht stellvertretend für 'PORTJ': */
#define Led *(unsigned char volatile*)(0x0028)
#define Tasten *(unsigned char volatile*)(0x0005)

/* Hier globale Variablen für Funktionen und
Hauptprogramm definieren (w.m. keine!).*/

/* Function ohne Parameterübergabe */
void MyInitFunc(void){
    DDRJ=0xFF; /* PORTJ Ausgang */
    DDRD=0x00; /* PORTD Eingang */
    Led=0;
}

/* Function mit Input-Parameter */
void nBeep(unsigned char n){
    unsigned char i; /* nur in nBeep bekannt */
    for(i=1;i<n;i++){
        Sound(1000); /* 1000Hz Ton */
        DelayXms(50); /* 200ms warten */
        NoSound(); /* Ton aus */
        DelayXms(50); /* 300ms warten */
    }
}

/* Function mit Output-Parameter */
char Taste7On(void){
    char T7On;
    T7On=0;
    if ((Tastens&0x80)==0x80){
        DelayXms(20); /* Taste entprellen */
        if ((Tastens&0x80)==0x80) T7On=1;
    }
    return T7On; /*T7On Rückgabewert */
}

/* Hauptfunktion (wird beim Start ausgeführt) */
void main(void){
    MyInitFunc(); /* L12 initialisieren */
    while (1){ /* endlose Schleife */
        if (Taste7On()){
            Led+=1; /* LEDs incrementieren */
            nBeep(3); /* 3mal beepen */
        }
    }
}
    
```

Kommentar: über mehrere Zeilen innerhalb der Zeichen /*...*/
 /* dies ist Kommentar */
 C++ auch nach // bis zum Zeilenende:
 // Dies ist C++ Kommentar

Include: Bestehende Standardfunktionen je nach Anwendung einbinden. Auch die HC12 Registerdefinitionen werden so eingebunden.

Define: Zuordnung von Namen zu Speicherplätzen. In hc12.h werden so alle HC12 Register-Adressen den entsprechenden Namen zugeordnet.

Globale Variablen: Sind in allen Funktionen und im Hauptprogramm bekannt. Möglichst vermeiden!

Function: Unterprogramme, die aus anderen Unterprogrammen (inkl. main) aufgerufen werden können. An Funktionen können beliebig viele Werte (Parameter) übergeben werden. Eine Funktion kann einen Wert zurückgeben. Soll die Funktion einen Wert zurückgeben, muss das Statement return auf den Rückgabewert verweisen (vgl. Funktion Taste7On). Die übergebenen Parameter müssen mit dem Typ und ihrem Namen deklariert sein. Werden keine Parameter übergeben, muss an der entsprechenden Stelle der Ausdruck void = leer stehen.

Anweisungen: Jede einzelne Anweisung wird mit einem Semikolon (;) abgetrennt. Auf einer Zeile können mehrere Anweisungen stehen. Innerhalb einer Anweisung werden zusätzliche Leerzeichen, Tabulatorzeichen und Zeilenvorschübe beim Compilieren nicht beachtet. Es empfiehlt sich, diese Zeichen zur Strukturierung und optimalen Lesbarkeit des Programms einzusetzen.

Block {...}: Blöcke werden in C durch geschweifte Klammern markiert. Innerhalb eines Blockes definierte Variablen gelten nur in diesem Bereich.

Function main: Die Funktion main muss in jedem C-Programm enthalten sein. Beim Start des Programms wird diese Funktion zuerst ausgeführt.

Struktur:	Programmierung in Assembler:	Programmierung in C									
<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> Programm Demo Struktur-Block </div>	<p>Programmrahmen: umschließt ein abgeschlossenes Programm oder ein Unterprogramm</p>	<p>Programmrahmen: umschließt das Hauptprogramm 'main' oder eine Funktion</p>									
<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> Anweisung </div>	<p>Anweisungen: movb PORTJ, PORTD clr PORTH inc PORTI</p>	<p>Anweisungen: PORTJ=PORTD; DDRD=0xFF; PORTI=PORTJ+5;</p>									
<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> Unterprogramm beep </div>	<p>Aufruf eines Unterprogramms: jsr beep bsr beep</p>	<p>Aufruf einer Funktion ohne Parameter: beep();</p>									
<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> <table border="1" style="width: 100%; text-align: center;"> <tr> <td>Wenn PortD = \$A3</td> <td>Nein</td> </tr> <tr> <td>Ja</td> <td>PortJ = \$55</td> </tr> <tr> <td>PortJ = \$AA</td> <td>PortJ = \$55</td> </tr> </table> </div>	Wenn PortD = \$A3	Nein	Ja	PortJ = \$55	PortJ = \$AA	PortJ = \$55	<p>Einfache Verzweigung: ldaa PORTD ;Akkua = \$A3 ? Falls cmpa #\$A3 bne Nein Ja movb \$AAA, PORTJ ;PortJ \$AA setzen jmp Ende Nein movb #\$55, PORTJ ;PortJ \$55 setzen Ende</p>	<p>Einfache Verzweigung: Bei nur einem Ausdruck im then oder else Teil, können die geschweiften Klammern weggelassen werden. if (PORTD == 0xA3) {PORTJ = 0xAAA;} else {PORTJ = 0x55;}</p>			
Wenn PortD = \$A3	Nein										
Ja	PortJ = \$55										
PortJ = \$AA	PortJ = \$55										
<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> <table border="1" style="width: 100%; text-align: center;"> <tr> <td rowspan="2">Falls Eing aus</td> <td>U</td> <td>PortJ increm</td> </tr> <tr> <td>D</td> <td>PortJ decrem</td> </tr> <tr> <td>C</td> <td>PortJ = 0</td> </tr> <tr> <td>Sonst</td> <td>PortJ = \$FF</td> </tr> </table> </div>	Falls Eing aus	U	PortJ increm	D	PortJ decrem	C	PortJ = 0	Sonst	PortJ = \$FF	<p>Mehrfachverzweigung: Case ldaa Eing cmpa #'U' beq CaseU cmpa #'D' beq CaseD cmpa #'C' beq CaseC bra Sonst CaseU inc PORTJ bra CEnd CaseD dec PORTJ bra CEnd CaseC clr PORTJ bra CEnd Sonst movb #\$FF, PORTJ bra CEnd CEnd</p>	<p>Mehrfachverzweigung: switch (Eing) { case 'U': ++PORTJ; break; case 'D': --PORTJ; break; case 'C': PORTJ=0; break; default: PORTJ=0xFF; }</p>
Falls Eing aus		U	PortJ increm								
	D	PortJ decrem									
C	PortJ = 0										
Sonst	PortJ = \$FF										

Struktur:	Programmierung in Assembler:	Programmierung in C
<div style="border: 1px solid black; padding: 5px;"> <p>Während PortD <= \$80</p> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> PortD mit 2 multiplizieren </div> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> 500ms warten </div> </div>	Schleife mit Anfangstest (While) While ldaa #\$80 ;Verglwert in A cmpa PORTD ;Vergl A mit PD blo WhiEn ;\$80<PD -> WhiEn lsl PORTD ;PortD*2 jsr Delay500ms ;500ms warten bra While ;Schleufe WhiEn	Schleife mit Anfangstest (While) <pre>while (PORTD<0x80){ PORTD<<1; Delay500ms(); }</pre>
<div style="border: 1px solid black; padding: 5px;"> <p>Rep</p> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> PortD lesen und invertiert an PortJ ausgeben </div> <p>bis PortD = \$FF</p> </div>	Schleife mit Endtest (Repeat..Until) Rep ldaa PORTD ;PortD->A coma ;A invertieren staa PORTJ ;A->PortJ ldaa #\$FF ;VglWert in A cmpa PORTD ;PortD = \$FF? bne Rep ;nein -> Rep RepEn ;ja -> RepEn	Schleife mit Endtest (do...while) <pre>do { PORTJ=~(PORTD); }while(PORTD != 0xFF);</pre>
<div style="border: 1px solid black; padding: 5px;"> <p>Für Var_i von 1 bis 10 (aufwärts)</p> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> Var_i an PortJ ausgeben </div> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> 500ms warten </div> </div>	Schleife für bestimmte Anzahl Durchläufe (For) For movb #1,Var_i ;Zähler auf 1 ForLo ldaa #10 ;EndWert in A cmpa Var_i ;Vgl Zähler,EndW blo ForEn ;EndW erreicht movb Var_i,PORTJ ;Anweisung1 jsr Delay500ms ;Anweisung2 inc Var_i ;Zähler increm. bra ForLo ;Schleufe ForEn	Schleife für bestimmte Anzahl Durchläufe (For) <pre>For (Var_i=1;Var_i<=10;Var_i++){ PORTJ=Var_i; Delay500ms(); }</pre>
<div style="border: 1px solid black; padding: 5px;"> <p>Für Var_i von 10 hinunter bis 1 (abwärts)</p> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> Var_i an PortJ ausgeben </div> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> 500ms warten </div> </div>	Schleife für bestimmte Anzahl Durchläufe (For) For movb #10,Var_i ;Zähler auf 10 ForLo ldaa #1 ;EndWert in A cmpa Var_i ;Vgl Zähler,EndW bhi ForEn ;EndW erreicht movb Var_i,PORTJ ;Anweisung1 jsr Delay500ms ;Anweisung2 dec Var_i ;Zähler decrem. bra ForLo ;Schleufe ForEn	Schleife für bestimmte Anzahl Durchläufe (For) <pre>For (Var_i=10;Var_i>=1;Var_i--){ PORTJ=Var_i; Delay500ms(); }</pre>

Struktur:	Programmierung in Assembler:	Programmierung in C
<div style="border: 1px solid black; padding: 5px;"> <p>Blink (PortJ einmal blinken)</p> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> PortJ als Ausgang setzen </div> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> Blinkmuster 1 an Port J ausgeben </div> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> 500ms warten </div> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> Blinkmuster 2 an Port J ausgeben </div> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> 500ms warten </div> </div>	Unterprogramm ohne Parameterübergabe Blink movb #\$FF,DDRJ ;UP Name movb #\$AA,PORTJ ;PortJ Ausgang jsr Delay500ms ;Blinkmuster 1 movb #\$55,PORTJ ;500ms warten jsr Delay500ms ;Blinkmuster 2 jsr Delay500ms ;500ms warten rts ;zum Aufruf zurück	Funktion ohne Parameterübergabe <pre>void Blink(void){ DDRJ=0xFF; PORTJ=0xAA; Delay500ms(); PORTJ=0x55; Delay500ms(); }</pre>
<div style="border: 1px solid black; padding: 5px;"> <p>Beep (In: Frequenz f)</p> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> Ton mit f einschalten </div> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> 500ms warten </div> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> Ton ausschalten </div> </div>	Unterprogramm Übergabe (f in X) beim Aufruf Beep pshx ;X sichern jsr Sound ;Ton f=X ausgeben jsr Delay500ms ;500ms warten jsr NoSound ;Ton ausschalten pulx ;X wie vor Aufruf rts ;zum Aufruf zurück	Funktion mit Parameter (f) beim Aufruf <pre>void Beep(int Frequ){ Sound(Frequ); Delay500ms(); NoSound(); }</pre>
<div style="border: 1px solid black; padding: 5px;"> <p>Swap (Nibble in Byte vertauschen) In: Byte (N1:N0), Out:Byte (N0:N1)</p> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> Nibble im Byte vertauschen </div> </div>	Unterprogramm Übergabe Byte in A bidirektional Swap pshb ;B sichern (Stack) pshx ;X sichern (Stack) tfr A,B ;A->B kopieren ldx #4 ;Cntn Shift Loop Shift lslb ;MSB B -> Carry rola ;Carry -> LSB A dbne X,Shift ;bis X=0 (4 mal) pulx ;X ursprüngl. Wert pulb ;B ursprüngl. Wert rts ;zum Aufruf zurück	Funktion Übergabe Byte Aufruf und Rückkehr <pre>unsigned char Swap(unsigned char SByte){ unsigned char Help; Help=SByte; Help>>=4; SByte<<=4; SByte=SByte+Help; return SByte;} </pre>
<div style="border: 1px solid black; padding: 5px;"> <p>Swap</p> </div>	Aufruf des Unterprogramms Swap ldaa PORTJ ;Quelle -> AkkuA jsr Swap ;Nibbles tauschen staa PORTD ;AkkuA -> Ziel	Aufruf der Funktion Swap PORTD = Swap(PORTJ);

ICC12 V5.0 Funktionen (im Unterverzeichnis Include)

Damit die hier zusammengefassten Funktionen angewendet werden können, müssen sie mit einer Include-Anweisung im Anwenderprogramm eingebunden werden z.B.:

```
#include <stdio.h> /* C Standard I/O-Funktionen einbinden */
```

<math.h>	ermittelt:
double exp10(double x);	Potenzwert mit der Basis 10 und dem Exponenten x
double exp(double x);	Potenzwert mit der Basis e und dem Exponenten x
double log10(double x);	Zehnerlogarithmus von x
double log(double x);	Natürlicher Logarithmus von x
double pow(double x, double y);	Potenzwert x hoch y
double fabs(double x);	Absolutwert einer Zahl (Vorzeichen ist +)
double fmod(double x, double y);	Rest der Division x/y wird zurückgegeben Rem=fmod(24.95, 5.5); //(Rem ist 2.95)
double sqrt(double x);	Quadratwurzel von x
double sin(double x);	sin(x [rad])
double cos(double x);	cos(x [rad])
double tan(double x);	tan(x [rad])
double asin(double x);	arcsin(x) [rad]
double acos(double x);	arccos(x) [rad]
double atan(double x);	arctan(x) [rad]
double degree_to_radian(double x);	Wert [rad] aus x [deg]
double radian_to_degree(double x);	Wert [deg] aus x [rad]

<stdio.h>	Bedeutung:
int getchar(void);	Einen Charakter vom Standard Inp lesen Ch=getchar();
int putchar(char ch);	Einen Charakter an den Standard Out senden putchar('?');
int puts(const char *s);	Einen String an den Standard Out senden. Rückgabe: letzter ausgegebener Charakter. puts('hallo');
int printf(char *s, ...);	String mit eingesetzten Zahlenwerten an Standard Out senden. Rückgabe: Anzahl ausgegebener Charakter. (vgl. auch Formatierung von printf, scanf)
int fprintf(FILE *fp, char *s, ..);	String mit eingesetzten Zahlenwerten in den Stream fp schreiben. Rückgabe und Formatierung wie printf. (In lcc12 identisch mit sprintf)
int sprintf(char *s1, char *s2, ..);	String s2 mit eingesetzten Zahlenwerten in den String s1 schreiben. Rückgabe und Formatierung wie printf.

<stdlib.h>	Bedeutung:
int abs(int i);	Gibt den Absolutwert von i n=abs(-5); // n ist 5
double atof(const char *s);	Wandelt String s in Fließkommazahl vom Typ double
int atoi(const char *s);	Wandelt String s in einen ganzzahligen Wert vom Typ int um. Im Fehlerfall wird 0 zurückgegeben.
long atol(const char *s);	Wandelt den String s in einen ganzzahligen Wert vom Typ long um. Im Fehlerfall wird 0 zurückgegeben.
void *calloc(size_t n, size_t size);	Gibt einen Zeiger auf einen Speicherbereich zurück der gross genug ist um n Elemente der Länge size aufzunehmen. Gesamter Bereich wird mit 0 initialisiert.
void exit(int i);	Beendet Programm. I wird an das D-Reg übergeben.
void free(void *p);	Gibt den durch p adressierten Speicher wieder frei.
void *malloc(size_t size);	Reserviert Speicher der Länge size auf dem Heap.
int rand(void);	Rückgabe: Pseudo-Zufallszahl zwischen 0 und RAND_MAX zurück
void *realloc(void *p, size_t size);	Weist p die neue Grösse size zu
void srand(unsigned seed);	Initialisiert Startwert seed für die Aufrufe mit rand()
long strtol(const char *s, char **ep, int b);	Wandelt die ersten Zeichen des Strings s in einen long Wert entsprechend der Zahlenbasis b. Ist b=0 wird auch das Hex- Oktal und Dezimalformat berücksichtigt. Falls Pointer ep ungleich 0 zeigt er nach der Ausführung auf das Ende der Konvertierung in s
unsigned long strtoul (const char *s, char **ep, int b);	Wie strtol jedoch ohne Vorzeichen

<ctype.h>	Bedeutung:
int isalnum (int c);	Rückgabe: ungleich 0 falls c eine Ziffer oder ein Buchstabe ist
int isalpha (int c);	Rückgabe: ungleich 0 falls c ein Buchstabe ist
int iscntrl (int c);	Rückgabe: ungleich 0 falls c Steuerzeichen (LF, CR usw.) ist
int isdigit (int c);	Rückgabe: ungleich 0 falls c eine Ziffer ist
int isgraph (int c);	Rückgabe: ungleich 0 falls c eine druckbares Zeichen ist
int islower (int c);	Rückgabe: ungleich 0 falls c ein Kleinbuchstabe ist
int isprint (int c);	Rückgabe: ungleich 0 falls c eine druckbares Zeichen ist
int ispunct (int c);	Rückgabe: ungleich 0 falls c eine druckbares Zeichen ist aber kein Leerzeichen kein Buchstabe und keine Zahl
int isspace (int c);	Rückgabe: ungleich 0 falls c : Leerzeichen, CR, FF, HT, NL oder VT
int isupper (int c);	Rückgabe: ungleich 0 falls c ein Grossbuchstabe ist
int isxdigit (int c);	Rückgabe: ungleich 0 falls c eine hexadezimale Ziffer ist
int tolower (int c);	Gross- in Kleinbuchstaben wandeln. Übrige Zeichen belassen
int toupper (int c);	Klein- in Grossbuchstaben wandeln. Übrige Zeichen belassen

<string.h>

```
void *memchr(void *s, int c, size_t n);
```

Suche in den ersten n Bytes in s nach dem ersten Vorkommen von c. Falls c gefunden wird, wird ein Zeiger auf dieses Zeichen zurückgegeben sonst NULL.

```
int memcmp(void *s1, void *s2, size_t n);
```

Vergleiche die ersten n Bytes von s1 und s2. Rückgabe: Übereinstimmung = 0. Bei Abweichung: Rückgabe > 0 falls erstes abweichendes Byte in s1 > als korrespondierendes Byte in s2. Sonst Rückgabe < 0.

```
void *memcpy(void *s1, void *s2, size_t n);
```

Kopiere n Bytes von s2 nach s1. Die Bereiche dürfen sich überlappen. Rückgabe ist s1

```
void *memset(void *s, int c, size_t n);
```

Schreibe n Mal das Zeichen c in den String s. Rückgabe ist s

```
char *strcat(char *s1, char *s2);
```

Hängt den String s2 an den String s1 an. Rückgabe s1

```
char *strchr(char *s, int c);
```

Suche das erste Vorkommen von c in s. Rückgabe: Zeiger auf erste Übereinstimmung, sonst NULL

```
char *strrchr(char *s, int c);
```

Suche das letzte Vorkommen von c in s. Rückgabe: Zeiger auf letzte Übereinstimmung, sonst NULL

```
int strcmp(char *s1, char *s2);
```

Vergleiche s1 mit s2. Rückgabe: Übereinstimmung = 0; Bei Abweichung: Rückgabe > 0 falls erstes abweichendes Element in s1 > als korrespondierendes Element in s2. Sonst Rückgabe < 0.

```
int strncmp(char *s1, char *s2, size_t n);
```

Wie strcmp() es werden jedoch nur n Zeichen verglichen

```
char *strcpy(char *s1, char *s2);
```

Kopiere s2 nach s1. Rückgabe s1.

```
char *strncpy(char *s1, char *s2, size_t n);
```

Wie strcpy() jedoch werden nur maximal n Zeichen kopiert

```
size_t strcspn(char *s1, char *s2);
```

Suche nach dem ersten Zeichen in s1 das mit einem beliebigen Zeichen in s2 übereinstimmt. Null der Terminierung inbegriffen. Rückgabe: Länge des Teilstrings s1 welcher kein Zeichen von s2 enthält.

```
char *strpbrk(char *s1, char *s2);
```

Wie strcspn() die abschliessende 0 wird jedoch nicht berücksichtigt

```
size_t strspn(char *s1, char *s2);
```

Suche erstes Zeichen in s1 welches nicht in s2 vorkommt. Inkl. 0 Terminierung. Rückgabe: Länge des Teilstrings s1 welcher kein Zeichen von s2 enthält.

```
size_t strlen(char *s);
```

Rückgabe: Länge des Strings s (ohne 0 für die Terminierung)

```
char *strstr(char *s1, char *s2);
```

Suche nach der ersten Übereinstimmung von s2 in s1. Rückgabe: NULL falls s2 nicht in s1 enthalten ist, sonst Zeiger auf das erste Vorkommen von s2 in s1.

HC12 Registernamen (im Unterverzeichnis Include)**<hc12.h>**

In diesem File sind alle Adressen den entsprechenden Registernamen gemäss HC12 Manuals zugeordnet. Wird dieses File eingebunden können mit diesen Namen (mit Grossbuchstaben geschrieben) direkt Operationen durchgeführt werden z.B: PORTJ=0xAA; usw.

Lemps12 Lbios Funktionen

Dient als Interface zu den LBIOS Unterprogrammen. Zudem sind die Adressen der Einsprünge nach Interrupts im RAM-Bereich hier verzeichnet. Für die detaillierte Beschreibung der einzelnen Funktionen vgl. auch *Lemps12 UserManual*, Kapitel *LBIOS Unterprogramme*.

<lbios.h>	Bedeutung:
<pre>char AkkuA, AkkuB; int IndX, IndY;</pre>	Diese Variablen werden global deklariert und von den Lbios Funktionen verwendet
Allgemeine Funktionen:	
<pre>void Delay500ms(void);</pre>	Verzögert den Programmablauf um ca. 500ms
<pre>void Delay1s(void);</pre>	Verzögert den Programmablauf um ca. 1s
<pre>void DelayXms(int Zeit);</pre>	Verzögert den Programmablauf um Zeit in [ms]
<pre>void Sound(int Freq);</pre>	Ton mit Frequenz Freq an Lautsprecher ausgeben. TimerCounter wird neu eingestellt. PPage wird geändert.
<pre>void NoSound(void);</pre>	Ton an Lautsprecher ausschalten. TC wird neu eingestellt. PPage wird geändert.
<pre>void AtdInit(void);</pre>	Initialisiert den AD-Wandler für 8 Kanäle, dauernd
Sci (RS232) Funktionen:	
<pre>void Sci0Init(int BaudRate);</pre>	BaudRate Sci0 einstellen (110..38400) default 19200
<pre>void Sci1Init(int BaudRate);</pre>	BaudRate Sci1 einstellen (110..38400) default 19200
<pre>void StdInOut(int Kanal);</pre>	Standard I/O Sci Kanal definieren: Sci0=0, Sci1=1.
<pre>void PutByte(int SendByte);</pre>	Ein Byte auf das StdInOut ausgeben
<pre>unsigned char GetByte();</pre>	Ein Byte vom StdInOut lesen. Wartet bis ein Zeichen am Sci angekommen ist, falls kein Zeichen im Buffer abholbereit ist.
<pre>unsigned char SciTest();</pre>	Gibt an, wieviele Zeichen im Buffer abholbereit sind
<pre>void Write(char *Fmt, &Daten);</pre>	Ausgabe eines Strings mit eingefügten Variablen auf StdInOut. Aufruf mit Zeiger auf Formatstring Fmt und Adresse des Datenblockes Dat. Formatierung vgl. Anhang
<pre>void WriteLn(char *Fmt, &Daten);</pre>	Wie Write jedoch mit NL und CR abgeschlossen

Port-LCD Unterstützung:	
<code>void lcdInit (&PORTx, &DDRx);</code>	LCD am PORTx und DDRx initialisieren
<code>void lcdClr();</code>	Löscht den LCD
<code>void lcdCursorSet (int CPos);</code>	Setzt den LCD Cursor an die Position CPos. Für den 4 Zeilen LCD LM044L (oder kompatibel) gilt: Zeile 1: 0x80..0x93; Zeile 2: 0xCO..0xD3; Zeile 3: 0x94..0xA7; Zeile 4: 0xD4..0xE7
<code>void lcdCursor();</code>	Cursor ist sichtbar
<code>void lcdCursorOff();</code>	Cursor ist unsichtbar
<code>void lcdWrite (char *Fmt, &Dat);</code>	Ausgabe Text und Daten an der Cursorposition des LCD. Aufruf mit Zeiger auf Formatstring Fmt und Adresse des Datenblockes Dat. (vgl. Write-Steuereichen im Anhang)
<code>void lcdDat (int c);</code>	Das Daten-Byte c an den LCD senden
<code>void lcdCtrl (int c);</code>	Das Control-Byte c an den LCD senden
<code>void lcdBackon();</code>	Hintergrundbeleuchtung ein (bedingt spez. LCD)
<code>void lcdBackoff();</code>	Hintergrundbeleuchtung aus (bedingt spez. LCD)
Funktionen für die RTC:	
<code>void TimeGet (char *Time);</code>	Übergibt die Zeit im BCD Format (3 Byte: hh,mm,ss) an den Buffer Time. Dieser Buffer muss vom Anwender zuvor deklariert werden
<code>void DateGet (char *Date);</code>	Übergibt das Datum im BCD Format (3 Byte: jj,mm,tt) an den Buffer Date. Dieser Buffer muss vom Anwender zuvor deklariert werden
<code>unsigned char DayGet();</code>	Gibt den Wochentag (1=Mo 7=So) zurück
I2C Send- und Empfangsprogramme:	
<code>void I2cStart();</code>	Gibt die Startbedingung aus
<code>void I2cStop();</code>	Gibt die Stoppbedingung aus
<code>void I2cSend (char Dat);</code>	Sendet ein Byte ohne Start/Stopp
<code>unsigned char I2cRead (int A);</code>	Liest ein Byte (A = 0 gibt AKN, A != 0 gibt kein AKN)
<code>void I2cStrout (int Adr, char s);</code>	Sendet den nullterminierten String s an die I2C Deviceadresse Adr (mit Start und Stopp)
<code>void I2cDatout (int Adr, char n, char *sSend);</code>	Sendet n Bytes ab der Speicheradresse sSend an die I2C Deviceadresse Adr
<code>void I2cDatIn (int Adr, char n, char *sEmpf);</code>	Sendet an die Deviceadresse Adr ein Read-Command, liest anschließend n Zeichen vom I2C Bus und speichert diese Zeichen im Empfangsbuffer ab der Adresse sEmpf
Spezielle Funktionen:	
<code>void SetIntHand (int Bez, *Fkt);</code>	Bewirkt, dass beim Auftreten des Interrupts Bez (vgl. Anhang) die Interrupt-Handelfunktion Fkt ausgeführt wird.
<code>void TerMinal ();</code>	Bewirkt einen SWI. Das Anwenderprogramm wird verlassen und der Terminal-Modus aktiviert

RTC Timer mit IRQ	Vorsicht funktioniert nur richtig, falls die RTC läuft!!
<code>void Timer1Init();</code>	Ausgang RTC für 1s Intervall schalten und IRQ Interrupt aktivieren
<code>void Timer250usInit();</code>	Ausgang RTC für 1s Intervall schalten und IRQ Interrupt aktivieren
<code>void TimerDestroy();</code>	Ausgang RTC ausschalten und IRQ Interrupt sperren
<code>void TimerStart();</code>	Timer Zählvariable auf 0 setzen und den Timer starten
<code>void TimerStop();</code>	Timer anhalten
<code>unsigned int TimerGetx();</code>	Timer Zählvariable auslesen. Gibt die Anzahl Intervalle (1s oder 250us) zurück, die seit dem letzten Aufruf von TimerStart() abgelaufen sind.

ASCII (American Standard Code for Information Interchange)

Char	Hex	Dez	Char	Hex	Dez	Char	Hex	Dez	Char	Hex	Dez
NUL	0	0	Space	20	32	@	40	64	`	60	96
SOH	1	1	!	21	33	A	41	65	a	61	97
STX	2	2	"	22	34	B	42	66	b	62	98
ETX	3	3	#	23	35	C	43	67	c	63	99
EOT	4	4	\$	24	36	D	44	68	d	64	100
ENQ	5	5	%	25	37	E	45	69	e	65	101
ACK	6	6	&	26	38	F	46	70	f	66	102
Beep	7	7	'	27	39	G	47	71	g	67	103
Back sp	8	8	(28	40	H	48	72	h	68	104
Tab	9	9)	29	41	I	49	73	i	69	105
Linefeed	A	10	*	2A	42	J	4A	74	j	6A	106
VT	B	11	+	2B	43	K	4B	75	k	6B	107
Formfeed	C	12	,	2C	44	L	4C	76	l	6C	108
Return	D	13	-	2D	45	M	4D	77	m	6D	109
SO	E	14	.	2E	46	N	4E	78	n	6E	110
SI	F	15	/	2F	47	O	4F	79	o	6F	111
DLE	10	16	0	30	48	P	50	80	p	70	112
DC1	11	17	1	31	49	Q	51	81	q	71	113
DC2	12	18	2	32	50	R	52	82	r	72	114
DC3	13	19	3	33	51	S	53	83	s	73	115
DC4	14	20	4	34	52	T	54	84	t	74	116
NAK	15	21	5	35	53	U	55	85	u	75	117
SYN	16	22	6	36	54	V	56	86	v	76	118
ETB	17	23	7	37	55	W	57	87	w	77	119
CAN	18	24	8	38	56	X	58	88	x	78	120
EM	19	25	9	39	57	Y	59	89	y	79	121
SUB	1A	26	:	3A	58	Z	5A	90	z	7A	122
Escape	1B	27	;	3B	59	[5B	91	{	7B	123
FS	1C	28	<	3C	60	\	5C	92	!	7C	124
GS	1D	29	=	3D	61]	5D	93	}	7D	125
RS	1E	30	>	3E	62	^	5E	94	~	7E	126
US	1F	31	?	3F	63	-	5F	95	DEL	7F	127

Dez-, Hex-, und Dualcode

DEZ	10 ⁰	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HEX	16 ⁰	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DUAL	2 ⁰	A	0	1	0	1	0	1	0	1	0	1	0	1	0	1	
	2 ¹	B	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
	2 ²	C	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
	2 ³	D	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1

Formatierung der Lbios Funktion Write (für Bildschirm- und LCD- Ausgabe)

Vorsicht: Bedingt durch die Kompatibilität zum Lemps12 Assembler sind diese Formatierungsanweisungen nicht gleich aufgebaut wie für printf():

- Es wird ein Slash (/) und nicht ein BackSlash (\) verwendet
- Es kann nur eine Adresse für die Daten angegeben werden. Der Anwender muss selber dafür sorgen, dass die weiteren Daten in der richtigen Reihenfolge im Speicher vorhanden sind.
- Es können keine Floats ausgegeben werden. Fließkommadarstellungen können jedoch zuerst mit der C-Funktion sprintf() in einen String abgelegt werden. Dieser String lässt sich dann in der Write Funktion mit %s eingefügen.

P. H.	Beschreibung	P. H.	Beschreibung
%bd	DezByte, Shortint +/-127	%bu	DezByte, unsigned, 0..255
%bx	HexByte	%bn	DezByte, unsigned, mit vorgestellten Nullen
%d	DezWort, Integer +/-32767	%%	% ausgeben
%x	HexWort 0000..FFFF	/n	Zeilenvorschub+Wagenrücklauf (#10 + #13)
%bi	%bi BinaerByte 00000000..11111111	/r	Wagenrücklauf (#10)
%i	BinaerWort	/v	Zeilenvorschub (#13)
%c	ASCII Charakter	/t	Tabulator = 8 Leerzeichen
%s	String (Zeiger auf nullterminierten String)	/00c	00 bis 99 mal Zeichen c ausgeben
%u	DezWort, unsigned, 0..65535	//	/ ausgeben
%n	DezWort, unsigned, mit vorgestellten Nullen		

HC12 Interruptbezeichnungen

```

#define KWIEHI    _INT_BASE+(0)    /* $FFCE,$FFCF Key Wakeup H */
#define KWIEJI    _INT_BASE+(3)    /* $FFD0,$FFD1 Key Wakeup J */
#define ADIEI     _INT_BASE+(6)    /* $FFD2,$FFD3 AD Converter */
#define SC1I1     _INT_BASE+(9)    /* $FFD4,$FFD5 Serial 1 */
#define SC1O1     _INT_BASE+(12)   /* $FFD6,$FFD7 Serial 0 */
#define SPI0EI    _INT_BASE+(15)   /* $FFD8,$FFD9 SPI */
#define PA1I1     _INT_BASE+(18)   /* $FFDA,$FFDB Pulse Accu input */
#define PA0V1I1   _INT_BASE+(21)   /* $FFDC,$FFDD Pulse Accu overf */
#define TO1I1     _INT_BASE+(24)   /* $FFDE,$FFDF Timer overflow */
#define TC7I1     _INT_BASE+(27)   /* $FFE0,$FFE1 Timer Ch7 */
#define TC6I1     _INT_BASE+(30)   /* $FFE2,$FFE3 Timer Ch6 */
#define TC5I1     _INT_BASE+(33)   /* $FFE4,$FFE5 Timer Ch5 */
#define TC4I1     _INT_BASE+(36)   /* $FFE6,$FFE7 Timer Ch4 */
#define TC3I1     _INT_BASE+(39)   /* $FFE8,$FFE9 Timer Ch3 */
#define TC2I1     _INT_BASE+(42)   /* $FFEA,$FFEB Timer Ch2 */
#define TC1I1     _INT_BASE+(45)   /* $FFEC,$FFED Timer Ch1 */
#define TC0I1     _INT_BASE+(48)   /* $FFEE,$FFF0 Timer Ch0 */
#define RT1EI1    _INT_BASE+(51)   /* $FFF0,$FFF1 Real Time Int */
#define IRQENI    _INT_BASE+(54)   /* $FFF2,$FFF3 IRQ */
#define XIRQI     _INT_BASE+(57)   /* $FFF4,$FFF5 XIRQ */
#define SW1I1     _INT_BASE+(60)   /* $FFF6,$FFF7 Software Int */
#define IN1TRI1   _INT_BASE+(63)   /* $FFF8,$FFF9 Instr Trap */
#define COP1AI1   _INT_BASE+(66)   /* $FFFA,$FFFB COP Watch Dog */
#define COP1LOI1 _INT_BASE+(69)   /* $FFFC,$FFFD COP Clock */
#define RE1SE1I1 _INT_BASE+(72)   /* $FFFE,$FFFF Reset */
    
```

Wichtig Funktionen für die Programmierung von Embedded Systems:

Myfunc.c <i>(Oft verwendete Funktionen die Sie selber schreiben sollten)</i>	
<code>char BitNrSet(char BitNr, char Vari);</code>	Das Bit BitNr (0..7) in Vari auf 1 setzen übrige Bits unbeeinflusst. Rückgabe: Vari geändert
<code>char BitNrClr(char BitNr, char Vari);</code>	Das Bit BitNr (0..7) in Vari auf 0 setzen übrige Bits unbeeinflusst. Rückgabe: Vari geändert
<code>char MusterSet(char BitMuster, char Vari);</code>	BitMuster (Bits mit 1 bezeichnet) in Vari auf 1 setzen übrige Bits unbeeinflusst. Rückgabe: Vari geändert
<code>char MusterClr(char BitMuster, char Vari);</code>	BitMuster (Bits mit 1 bezeichnet) in Vari auf 1 setzen übrige Bits unbeeinflusst. Rückgabe: Vari geändert
<code>char Qs1Char(char Vari);</code>	Anz. gesetzte Bits (1) in Vari bestimmen (Quersumme aller 1, max. 8). Rückgabe: Quersumme 1
<code>char Qs0Char(char Vari);</code>	Anz. zurückgesetzte Bits (0) in Vari bestimmen (Quersumme aller 0, max. 8). Rückgabe: Quersumme 0
<code>char Bar8N(char Anz);</code>	8 Bit Balken mit Anz (0..8) 1 darstellen. Beisp: Bar8N(5) gibt 0x1F zurück
<code>char Bar8Prop(int InWert, int Max);</code>	8 Bit Balken für InWert proportional zu Max darstellen. Beisp: Bar8Prop(40,100) gibt 0x0F zurück
<code>int BinDez(int Bin);</code>	Binär Bin (0..15) als Dezimalwert darstellen. Beisp: Dez.6: Bit6 = 1, übrige Bits = 0 (BinDez-Decoder)
<code>char GetLoNib(char InByte);</code>	Gibt das LS Nibble von InByte zurück (MS Nibble = 0)
<code>char GetHiNib(char InByte);</code>	Gibt das MS Nibble von InByte zurück (LS Nibble = 0)
<code>char SetHiNib(char Vari, char LoNib);</code>	Setzt MS Nibble in Vari auf den Wert von LoNib. LS Nibble in Vari unverändert. Rückgabe: Vari geändert
<code>char SetLoNib(char Vari, char LoNib);</code>	Setzt LS Nibble in Vari auf den Wert von LoNib. MS Nibble in Vari unverändert. Rückgabe: Vari geändert
<code>char SwapNib(char InByte);</code>	Vertauscht die Nibbles in einem Byte. Rückgabe InByte mit vertauschten Nibbles
<code>int GetHiByte(int InByte);</code>	Gibt nur das MSByte von InByte zurück
<code>int GetLoByte(int InByte);</code>	Gibt nur das LSByte von InByte zurück
<code>char StelleToCode(char Stelle);</code>	Universeller 4 Bit Coder. Gibt für Stelle (0..15) den entsprechenden Code zurück
<code>char CodeToStelle(char Code);</code>	Universeller 4 Bit Decoder. Gibt für einen bestimmten Code (0..15) die Stelle zurück