



tappertzhofen

.ccasm

Referenz

**C-Control M Unit 2 Assembler
C-Control Micro Assembler**

**.ccasm Referenz (Deutsch)
Rev 8
Nov. 2007**

www.tappertzhofen.eu/ccasm

.ccasm Referenz

Rev 8 – Nov. 2007

© 2004 – 2008 TAPPERTZHOFEN;

www.tappertzhofen.eu/ccasm

BEFEHLSZEILENARGUMENTE.....	7
ZIELPLATTFORMEN.....	8
ASSEMBLER REFERENZ.....	9
Code Strukturierung.....	9
Unterstützte Adressierungen	10
Systemvariablen	11
Einfaches Assembler Beispiel	14
Assembler Aufruf in BASIC++.....	15
CCASM mit BASIC++ verknüpfen	16
Instruktionen.....	17
ADC.....	17
ADD.....	17
AIS.....	17
AIX.....	18
AND.....	18
ALS.....	18
ASR.....	19
BCC.....	19
BCLR.....	19
BCS.....	19
BEQ.....	20
BHCC.....	20
BHCS.....	20
BHI.....	20
BHS.....	21
BIH.....	21
BIL.....	21

BIT	21
BLO	22
BLS	22
BMC	22
BMI	22
BMS	23
BNE	23
BPL	23
BRA	23
BRCLR	24
BRN	24
BRSET	24
BSET	24
BSR	25
CBEQ	25
CLC	25
CLI	25
CLR	26
CMP	26
COM	26
CPHX	27
CPX	27
DAA	27
DBNZ	28
DEC	28
DIV	28
EOR	29
INC	29
JMP	29
JSR	30
LDA	30
LDHX	30
LDX	31
LSA	31
LSL	31
LSR	32
MOV	32
MUL	32
NEG	33
NOP	33

NSA	33
ORA	34
ROL	34
ROR	34
RSP	34
RTI	35
RTS	35
SBC	35
SEC	35
SEI	36
SSA	36
STA	36
STHX	37
STOP	37
STX	37
SUB	37
SWI	38
TAP	38
TAX	38
TPA	38
TST	39
TSX	39
TXA	39
TXS	39
WAIT	40
Zusammengesetzte Befehle	41
TAH	41
THA	41
THX	41
TXH	41

Befehlszeilenargumente

Um CCAsm aufrufen zu können benötigen Sie die Kenntnis über die erlaubten Befehlszeilenargumente. Der manuelle Aufruf von CCAsm kann zum Beispiel über die MS-DOS Konsole erfolgen.

Aufruf:

ccasm [Dateiname] -[Option]

[Dateiname] Dateiname bzw. vollständiger Pfad und Dateiname der Datei in Anführungszeichen

[Option] Option zur Assemblierung bzw. Codedebugging

 -debug Erzeugt eine Debug Datei

Ohne Angabe der Option generiert der Assembler eine Binärdatei, die mit dem BASIC++ Compiler eingebunden werden kann.

Beispiel:

```
ccasm „test.asm“ -debug
```

```
ccasm „test.asm“
```

Zielplattformen

CCAsm erzeugt Motorola MC68HC08 kompatiblen Maschinencode. Generell sind also alle erzeugten Binärdateien auf allen MC68HC08 kompatiblen Mikrocontrollern lauffähig.

Daneben kann CCAsm auch zum programmieren eigener Assemblerprogramme für die C-Control M Unit 2 und C-Control Micro eingesetzt werden.

Assembler Referenz

Code Strukturierung

Ein typischer CCAsm Source Code gliedert sich in einen (oder mehreren) CODE und einen (oder mehrere) DATEN Bereich(e).

Innerhalb des CODE Bereiches sind nur dem Assembler bekannte und erlaubte Assemblerbefehle zugelassen. Der CODE Bereich wird mit dem Präprozessorschlüsselwort „.CODE“ eingeleitet und schließt bei Bedarf einen vorherigen Datenbereich ab.

Im DATEN Bereich können Werte, Konstanten und Zeichenketten abgelegt werden. Das Präprozessorschlüsselwort „.DATA“ leitet den Datenbereich ein. Auch hier wird bei Bedarf ein schon offener Codebereich geschlossen.

Das Präprozessorschlüsselwort „.END“ ist optional. Nach „.END“ können keine weiteren Daten oder Codes mehr eingefügt werden.

Beispiel:

```
.CODE
    ADC #5
    RTS

.DATA
    T    DB    0

.END
```

Unterstützte Adressierungen

CCAsm unterstützt auf Grund der technischen Rahmenbedingungen nicht alle verfügbaren Adressierungsoptionen des MC68HC.

Erlaubte Adressierungsoptionen sind IMM (Immediate addressing mode), DIR (Direct addressing mode), REL (Relative addressing mode), IX (Indexed) und IHM (Inherent addressing mode).

Bei Inherenten Befehlen (z.B. ASLA oder ALSX) kann auch der Stammbefehl mit dem Operanden in folgender Schreibweise verwendet werden:

[BEFEHL] [IHM]

Beispiel:

ASL A

Im weiteren Verlauf werden folgende Operatoren für die Adressierungen verwendet:

- # Immediate addressing mode
 - Direct addressing mode (kein Zeichen)
 - Relative addressing mode (kein Zeichen)
 - Inherent addressing mode (kein Zeichen)
- ,
- Indexed (ohne ROM Zugriff nur auf BASIC Ram, mit ROM Zugriff nur auf ROM Speicher)
- & Konstantes Zeichen

Systemvariablen

CCASM verfügt über einige Pseudo-Befehle, die nicht zum Standard Freescale Befehlssatz gehören um Systemvariablen zu lesen und zu setzen.

Diese Parameter (im Folgenden „Variablen“ genannt) können nur mit der LSA und SSA Instruktion verwendet werden.

Beispiel:

```
.code

GetTimer:
    LSA TIMERHIGH
    STA $00C0
    LSA TIMERLOW
    STA $00C1
    RTS

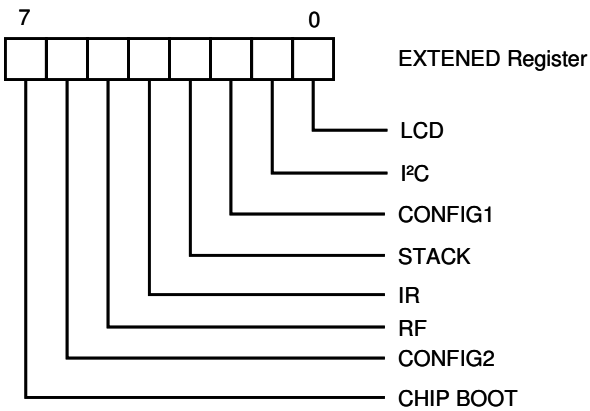
SetTimer:
    LDA $00C0
    SSA TIMERIGH
    LDA $00C1
    SSA TIMERLOW
    RTS
```

Gültige Systemvariablen:

Name	Bedeutung
Second	RTC Sekunde
Minute	RTC Minute
Hour	RTC Stunde
Dow	RTC Wochentag
Day	RTC Tag
Month	RTC Monat
Year	RTC Jahr
Timerhigh	Timer Hi-Byte
Timerlow	Timer Lo-Byte
TTics	/
DCFVerBuf	/
BPHP	Basic Programm Pointer Hi
BPLP	Basic Programm Pointer Lo
BSPointer	/
Freqhigh	Freq Hi-Byte
FreqLow	Freq Lo-Byte
Freq2High	Freq2 Hi-Byte
Freq2Low	Freq2 Lo-Byte
IRQHigh	IRQ Adresse Hi-Byte
IRQLow	IRQ Adresse Lo-Byte
Config	Config Register
Config2	Config 2 Register
StCnt	Stack Counter
Stack_A_Hi	Stack Variablen
Stack_A_Lo	Stack Variablen
Stack_B_Hi	Stack Variablen
Stack_B_Lo	Stack Variablen
Stack_C_Hi	Stack Variablen
Stack_C_Lo	Stack Variablen
Stack_D_Hi	Stack Variablen
Stack_D_Lo	Stack Variablen
Stack_E_Hi	Stack Variablen
Stack_E_Lo	Stack Variablen
Stack_F_Hi	Stack Variablen
Stack_F_Lo	Stack Variablen

Name	Bedeutung
Stack_G_Hi	Stack Variablen
Stack_G_Lo	Stack Variablen
Extended	Extended Objekt Register

Aufbau des Extended Objekt Registers:



Einfaches Assembler Beispiel

Das folgende Beispiel inkrementiert die Bytevariable [1] der C-Control (Aufruf: ccasm test.asm):

```
PORTA      .EQU $0000
PORTB      .EQU $0001
PORTC      .EQU $0002
PORTD      .EQU $0003
PORTE      .EQU $0008
DDRA       .EQU $0004
DDRB       .EQU $0005
DDRC       .EQU $0006
DDRD       .EQU $0007
DDRE       .EQU $000C

USERV1     .EQU $00C0 ;USER VARIABLE 1

.code

ENTRY1:    LDA USERV1
           INCA
           STA USERV1
           RTS

ENTRY2:    LDA PORTB
           EOR #$01
           STA PORTB
           RTS

.end
```

Assembler Aufruf in BASIC++

```
DEFINE MeineVariable BYTE[1]

LCD.INIT
DO
    SYS FD09h ' Startadresse des Assembler-
              ' Programms
    LCD.CLEAR
    LCD.Print MeineVariable
    PAUSE 20
LOOP
LCD.OFF

SYSCODE "test.bin"
```

CCASM mit BASIC++ verknüpfen

CCASM kann mit BASIC++ verknüpft und die Offsets frei wählbarer Labels exportiert werden. Der Entwickler kann mit dem Pseudo-Befehl PUBLIC ein Label als öffentlich markieren, das dann von CCASM in einer BAS-Datei automatisch exportiert wird. Mit dem .PAGE Befehl kann der Entwickler die Adresse der Page bestimmen (\$FD00 oder \$FC00), in der der Code später in den Controller geladen wird.

```
; TEST.ASM

.page $fd00

.code

Label1:
    rts ; wird nicht exportiert

public Label2:
    rts ; wird exportiert und ist damit in BASIC++
        ; sichtbar

.end

' TEST.BAS

' CCASM Import
CONST Label2 = FD09h

' SYSCODE "test.bin"
```


Instruktionen

ADC

Addition mit Carry

<u>Adressierung</u>	<u>Taktzyklen</u>
ADC #opr	2
ADC opr	3
ADC ,X	2 + 12

ADD

Addition ohne Carry

<u>Adressierung</u>	<u>Taktzyklen</u>
ADD #opr	2
ADD opr	3
ADD ,X	2 + 12

AIS

Direkter vorzeichenbehafteter Wert zu SP addieren

<u>Adressierung</u>	<u>Taktzyklen</u>
AIS #opr	2

AIX

Direkter vorzeichenbehafteter Wert zu H:X addieren

<u>Adressierung</u>	<u>Taktzyklen</u>
AIX #opr	2

AND

Logisches Und

<u>Adressierung</u>	<u>Taktzyklen</u>
AND #opr	2
AND opr	3
AND ,X	2 + 12

ALS

Arithmetischer Links Shift

<u>Adressierung</u>	<u>Taktzyklen</u>
ALS #opr	2
ALSA	1
ALSX	1

ASR

Arithmetischer Rechts Shift

<u>Adressierung</u>	<u>Taktzyklen</u>
ASR #opr	2
ASRA	1
ASRX	1

BCC

Springen, wenn Carry nicht gesetzt

<u>Adressierung</u>	<u>Taktzyklen</u>
BCC rel	3

BCLR

Bit n in Opr löschen

<u>Adressierung</u>	<u>Taktzyklen</u>
BCLR &n, opr	4

BCS

Springen, wenn Carry Flag gesetzt

<u>Adressierung</u>	<u>Taktzyklen</u>
BCS rel	3

BEQ

Springe bei Gleichheit

Adressierung

Taktzyklen

BEQ rel

3

BHCC

Springe, wenn Half-Carry Bit gelöscht

Adressierung

Taktzyklen

BHCC rel

3

BHCS

Springe, wenn Half-Carry Bit gesetzt

Adressierung

Taktzyklen

BHCS rel

3

BHI

Springe, wenn größer

Adressierung

Taktzyklen

ADC rel

3

BHS

Springe, wenn größer oder gleich

<u>Adressierung</u>	<u>Taktzyklen</u>
BHS rel	3

BIH

Springe, wenn IRQ Pin gesetzt

<u>Adressierung</u>	<u>Taktzyklen</u>
BIH rel	3

BIL

Springe, wenn IRQ Pin gelöscht

<u>Adressierung</u>	<u>Taktzyklen</u>
BIL rel	3

BIT

Bit Test

<u>Adressierung</u>	<u>Taktzyklen</u>
BIT #opr	2
BIT opr	3
BIT ,X	2 + 12

BLO

Springe, wenn kleiner

Adressierung

Taktzyklen

BLO rel

3

BLS

Springe, wenn kleiner oder gleich

Adressierung

Taktzyklen

BLS rel

3

BMC

Springe, wenn Interrupt Maske gelöscht

Adressierung

Taktzyklen

BMC rel

3

BMI

Springe, wenn minus

Adressierung

Taktzyklen

BMI rel

3

BMS

Springe, wenn Interrupt Maske gesetzt

<u>Adressierung</u>	<u>Taktzyklen</u>
BMS rel	3

BNE

Springe, wenn ungleich

<u>Adressierung</u>	<u>Taktzyklen</u>
BNE rel	3

BPL

Springe, wenn plus

<u>Adressierung</u>	<u>Taktzyklen</u>
BPL rel	3

BRA

Unbedingter Sprung

<u>Adressierung</u>	<u>Taktzyklen</u>
BRA rel	3

BRCLR

Springe, wenn Bit n in Opr gelöscht

<u>Adressierung</u>	<u>Taktzyklen</u>
BRCLR &n, opr, rel	5

BRN

Niemals Springen

<u>Adressierung</u>	<u>Taktzyklen</u>
BRN rel	3

BRSET

Springe, wenn Bit n in Opr gesetzt

<u>Adressierung</u>	<u>Taktzyklen</u>
BRSET &n, opr, rel	5

BSET

Setzte Bit n in Opr

<u>Adressierung</u>	<u>Taktzyklen</u>
BSET &n, opr	4

BSR

Springe in ein Unterprogramm

<u>Adressierung</u>	<u>Taktzyklen</u>
BSR rel	4

CBEQ

Vergleichen und Springe bei Gleichheit

<u>Adressierung</u>	<u>Taktzyklen</u>
CBEQ opr, rel	5
CBEQA #opr, rel	4
CBEQX #opr, rel	4

CLC

Carry Bit löschen

<u>Adressierung</u>	<u>Taktzyklen</u>
CLC	1

CLI

Interrupt Maske löschen

<u>Adressierung</u>	<u>Taktzyklen</u>
CLI	2

CLR

Löschen

Adressierung

Taktzyklen

CLR opr	3
CLRA	1
CLR X	1
CLR H	1
CLR ,X	2 + 12

CMP

Vergleichen

Adressierung

Taktzyklen

CMP #opr	2
CMP opr	3
CMP ,X	2 + 12

COM

Einerkomplement

Adressierung

Taktzyklen

COM opr	4
COM A	1
COM X	1
COM ,X	2 + 12

CPHX

Vergleiche H:X mit Opr

Adressierung

Taktzyklen

CPHX #opr

3

CPHX opr

4

CPX

Vergleiche X mit Opr

Adressierung

Taktzyklen

CPX #opr

2

CPX opr

3

DAA

Dezimaler Abgleich von A

Adressierung

Taktzyklen

DAA

2

DBNZ

Dekrementieren und Springen, wenn nicht Null

<u>Adressierung</u>	<u>Taktzyklen</u>
DBNZ opr, rel	5
DBNZ rel	3
DBNZX rel	3

DEC

Dekrementieren

<u>Adressierung</u>	<u>Taktzyklen</u>
DEC opr	4
DECA	1
DECX	1
DEC ,X	2 + 12

DIV

Division

<u>Adressierung</u>	<u>Taktzyklen</u>
DIV	7

EOR

Exklusives Oder von Opr mit A

<u>Adressierung</u>	<u>Taktzyklen</u>
EOR #opr	2
EOR opr	3
EOR ,X	2 + 12

INC

Inkrementieren

<u>Adressierung</u>	<u>Taktzyklen</u>
INC opr	4
INCA	1
INCX	1
INC ,X	2 + 12

JMP

Springen

<u>Adressierung</u>	<u>Taktzyklen</u>
JMP opr	2

JSR

Springe in Unterprogramm

Adressierung

Taktzyklen

JSR opr

4

LDA

Lade Opr in A

Adressierung

Taktzyklen

LDA #opr

2

LDA opr

3

LDA ,X

2 + 12

LDA ROM ,X

2 + 12

LDHX

Lade Opr in H:X

Adressierung

Taktzyklen

LDHX #opr

3

LDHX opr

4

LDX

Lade Opr in X

Adressierung

Taktzyklen

LDX #opr

2

LDX opr

3

LDX ,X

2 + 12

LSA

Lade Systemvariable nach A

Adressierung

Taktzyklen

LSA Variable

3

LSL

Logischer Links Shift

Adressierung

Taktzyklen

LSL opr

4

LSLA

1

LSLX

1

LSR

Logischer Rechts Shift

<u>Adressierung</u>	<u>Taktzyklen</u>
LSR opr	4
LSRA	1
LSRX	1

MOV

Kopieren (Quelle, Ziel)

<u>Adressierung</u>	<u>Taktzyklen</u>
MOV opr, opr	5
MOV #opr, opr	4

MUL

Vorzeichenlose Multiplikation

<u>Adressierung</u>	<u>Taktzyklen</u>
MUL	5

NEG

Negieren

Adressierung

Taktzyklen

NEG opr

4

NEGA

1

NEGX

1

NEG ,X

2 + 12

NOP

“Keine Operation” – No operation

Adressierung

Taktzyklen

NOP

1

NSA

Nibble von A vertauschen

Adressierung

Taktzyklen

NSA

3

ORA

Oder mit A und Opr

<u>Adressierung</u>	<u>Taktzyklen</u>
ORA #opr	2
ORA opr	3
ORA ,X	2 + 12

ROL

Links Shift mit Carry

<u>Adressierung</u>	<u>Taktzyklen</u>
ROL opr	4
ROLA	1
ROLX	1

ROR

Rechts Shift mit Carry

<u>Adressierung</u>	<u>Taktzyklen</u>
ROR opr	4
RORA	1
RORX	1

RSP

Stack Zeiger zurücksetzen

<u>Adressierung</u>	<u>Taktzyklen</u>
RSP	1

RTI

Rücksprung aus Interrupt Routine

Adressierung

Taktzyklen

RTI

7

RTS

Rücksprung aus einem Unterprogramm

Adressierung

Taktzyklen

RTS

4

SBC

Subtraktion mit Carry

Adressierung

Taktzyklen

SBC #opr

2

SBC opr

3

SBC ,X

2 + 12

SEC

Carry Bit setzen

Adressierung

Taktzyklen

SEC

1

SEI

Interrupt Maske setzen

Adressierung

Taktzyklen

SEI

2

SSA

Speiche A in Systemvariable

Adressierung

Taktzyklen

SSA Variable

3

STA

Speichere A in Opr

Adressierung

Taktzyklen

STA opr

3

STA ,X

2 + 12

LDA ROM ,X

2 + 12

STHX

Speichere H:X in Opr

Adressierung

Taktzyklen

STHX opr

4

STOP

Stop Modus (Low Power Modus)

Adressierung

Taktzyklen

STOP

1

STX

Speichere X in Opr

Adressierung

Taktzyklen

STX opr

3

STX ,X

2 + 12

SUB

Subtraktion

Adressierung

Taktzyklen

SUB #opr

2

SUB opr

3

SUB ,X

2 + 12

SWI

Software Interrupt

Adressierung

Taktzyklen

SWI

9

TAP

A nach CCR

Adressierung

Taktzyklen

TAP

2

TAX

A nach X

Adressierung

Taktzyklen

TAX

1

TPA

CCR nach A

Adressierung

Taktzyklen

TPA

1

TST

Auf negative oder Null testen

<u>Adressierung</u>	<u>Taktzyklen</u>
TST opr	3
TSTA	1
TSTX	1
TST ,X	2 + 12

TSX

SP nach H:X

<u>Adressierung</u>	<u>Taktzyklen</u>
TSX #opr	2

TXA

X nach A

<u>Adressierung</u>	<u>Taktzyklen</u>
TXA	1

TXS

H:X nach SP

<u>Adressierung</u>	<u>Taktzyklen</u>
TXS	2

WAIT

Wait Modus (Low Power Modus)

Adressierung

Taktzyklen

WAIT

1

Zusammengesetzte Befehle

CCAsm kennt die hier gelisteten zusammengesetzte Befehle.

TAH

A nach H

Adressierung

TAH

THA

H nach A

Adressierung

THA

THX

H nach X

Adressierung

THX

TXH

X nach H

Adressierung

TXH

Kontakt

TAPPERTZHOFEN
Heinsenstraße 32
D-40221 Düsseldorf

Tel. ++49 (0)241 169 3400

Weitere Informationen:

<http://www.tappertzhofen.eu>

Alle Informationen in dieser Dokumentation wurden nach besten Wissen zusammengestellt und aufgearbeitet. Eventuelle Verbesserungen oder Veränderungen jeglicher Art können ohne vorherige Ankündigung vorgenommen werden.

Es wird keinerlei Haftung für eventuelle Schäden übernommen.

Jede Art von nicht erlaubter elektronischer Vervielfältigung der Dokumentation oder Auszüge der Dokumentation können strafrechtlich verfolgt werden.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zur Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürfen

Copyright © 2004 - 2008 Tappertzhofen; Alle Rechte vorbehalten.



tappertzhofen