

Prozessor HC680 fiktiv

Dokumentation der Simulation



Die Simulation umfasst die Struktur und Funktionalität des Prozessors und wichtiger Baugruppen des Systems. Dabei werden in einem Simulationsfenster sowohl der Ablauf des Maschinenprogramms als auch dessen Erstellung mittels Assembler realisiert. Assemblerprogramme können gespeichert und geladen werden.

Daten des Systems

Prozessor:	HC 680 <small>fiktiv</small>
Verarbeitungsbreite:	8 Bit Daten und Adressen
Befehlslänge:	1 Byte
Adressraum:	256 Byte
Takt:	Einzelschritt, 0,1Hz bis 255Hz, ungebremst
Register:	4 allgemeine Register mit Schattenregister, Startadresse (ST) Befehlsregister (IR), Befehlszähler (IC), Statusregister (SR), Stapelzeiger (SP)
Flags:	Negativ (N), Null (Z), Überlauf (V), Übertrag (C) und I/O-Bits
Rechenwerk:	Arithmetik-Logik-Einheit (ALU), 8+1 Bit Vorzeichenerweiterung bei Addition, Multiplikation nach Booth, Division mit Rest, ALU bitweise darstellbar
Zahlenraum:	dezimal von -128 bis 127
Grafik:	8x8 Maxipixel-Display, 8 Byte Shared Memory
Ein- Ausgabe:	nach I/O-Bits: Tastaturzeichen, dezimal, hexadezimal, – Ausgabe auch binär
Dateien:	nach I/O-Bits: Tastaturzeichen, dezimal, hexadezimal, binär
Breakpoints:	3 – beim Programmlauf änderbar
Debug:	beliebig abschnittsweise in wählbare Datei
Assembler:	40 Mnemonics, 57 Befehle durch Adressierungsarten – siehe Manual

Die Bedienungshilfe wird über umfangreiche, Tooltips realisiert (abschaltbar).

Kurzanleitung | I/O-System | Input | MGA-Display | Output | I/O-Protokoll ☒

Kurzanleitung:

Beachten Sie bitte die Tooltips zu den wesentlichen Elementen.

Erstellen Sie ein Assembler/Maschinenprogramm durch Auswahl der Mnemonics und Operanden.

Ungültige Assemblerbefehle werden in NOP (00000000) umgewandelt.

Speichern Sie den leeren RAM als Assembler und öffnen Sie die Datei, um die Struktur zu erkennen.

Assemblerdateien können mit einem Texteditor erstellt und bearbeitet werden.

Dabei sind zur einfacheren Notation Auslassungen möglich. Details siehe Tooltip Laden.

Laden Sie ein oder mehrere Assemblersequenzen unter Beachtung der Speicheradressen.

Obwohl die Darstellung des RAM nur bis Zeile 127 für Assemblerbefehle konzipiert ist,

wird ein Maschinenprogramm auch darüberhinaus ausgeführt!

Die RAM-Zeilen ab 128 sind konzeptionell für Daten, den Stack, den Tastatur- und Mauspuffer

und den Micro-Graphic-Adapter (MGA-Display 8x8 'Maxi'-pixel) vorgesehen.

Läuft das Programm in diesen Bereich, werden die Bytes jedoch als Maschinenbefehle interpretiert!

Ebenso können Daten in den Bereich bis Zeile 127 geschrieben und ausgeführt werden.

Die RAM-Zeilen 0 und 128 werden jedoch nach Manipulation bei Programmstop aus dem Edit

im Assemblerbereich rekonstruiert.

Starten Sie das Maschinenprogramm mit dem grünen Startknopf.

Das beim Programmlauf erscheinende Eingabefeld ist nicht anzuklicken, Tooltipp dazu beachten.

Als Tastaturpuffer dient das Byte mit der Adresse hF6. Es wird in jedem Takt gefüllt und ermöglicht

mit dem Micro-Graphic-Adapter sogar elementare ereignisgesteuerte GUI-Programme.

Im RAM-Byte hF7 wird der MGA-Status abgelegt: | schwarz: 1 | Zeile: bbb | klick: 1 | Spalte: bbb |

d.h. linkes Bit gesetzt: schwarz, drei Bit: Zeile, 1 Bit: klick = 1, drei Bit: Spalte.

Der Inhalt von Byte hF7 wird durch das System in jedem Takt ausgewertet, das adressierte Pixel gesetzt/gelöscht.

Bei Programmen, die auf Tastatur- und/oder Mausereignisse reagieren den Taktbereich experimentell ermitteln.

Bei Multiplikation und Division werden weitere ALU-Register und Buttons eingeblendet. Siehe Tooltips dazu.

Nehmen Sie bitte auch die Dokumentation der Simulation und das Manual zum HC680 zur Kenntnis.

[illegible]

VORSICHT!
Den gesamten RAM, die Flag-Spalte und die Kommentare nach Sicherheitsabfrage löschen.

Flags löschen ☒ **CACHE** ☐

ACHTUNG!
Die Spalte der zuletzt gesetzten Flags ohne Sicherheitsabfrage löschen.

Assembler

Mnemonics

Programm | Daten: GUI Rechner

Startadr. von bis | von bis | Speichern Laden

Op. 1 Op. 2 um Ver | schieben

Kommentar zum Befehl

Rücksprung Ereignisschleife

ANSI Kommentar zu Daten

Darstellung Ziffer 1 mit = Zeichen

Liste der Assemblerbefehle

JMP RG IC <- Reg xx xJMP

Operanden

Op. 1

.SR.

.TA.

.+A.

.D0.

.D1.

.A0.

.A1.

.SR.

.SP.

[D0]

[D1]

[A0]

[A1]

0000

0001

0010

0011

0100

0101

0110

0111

1000

1001

1010

1011

1100

1101

1110

1111

00.

01.

10.

11.

Display

Beispiel mit Klickbereichen
(Zifferneingabe hier mit Tastatur)

Stackpointer, Tastaturbyte,

Pixelzeile, MGA-RAM

Adressen	Binär	Hex	Dec	Hex	Dec	Comment
11110011	F3	00000000	0	h00		
11110100	F4	00000000	0	h00		<-SP
11110101	F5	00000000	0	h00		
11110110	F6	00110001	1	h31		->Ta
11110111	F7	00000000	0	h00		>Pix
11111000	F8	00100000	32	h20		row1
11111001	F9	01100111	103	h67		row2
11111010	FA	00100000	32	h20		row3
11111011	FB	00100111	39	h27		row4
11111100	FC	00100000	32	h20		row5
11111101	FD	00000010	2	h02		row6
11111110	FE	11100111	-25	hE7		row7
11111111	FF	00000010	2	h02		row8

Das MGA-Bitmuster ist klar zu erkennen

Mnemonics

Mnemonic | Op. 1 | Op. 2 | um | Ver | schieben

NOP

Assembler Befehls-Mnemonic auswählen oder eintragen

Kommentare

Assemblerbefehle,

Liste mit Erklärung in Kurzform

Navigation im RAM

ANSI

RAM - Adr. zurück

Geben Sie einen Kommentar zum Befehl ein - auch länger als Eingabefeld, wird im Tooltip sichtbar.

Daten	Kommentar zu Daten
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	
41	
42	
43	
44	
45	
46	
47	
48	
49	
50	
51	
52	
53	
54	
55	
56	
57	
58	
59	
60	
61	
62	
63	
64	
65	
66	
67	
68	
69	
70	
71	
72	
73	
74	
75	
76	
77	
78	
79	
80	
81	
82	
83	
84	
85	
86	
87	
88	
89	
90	
91	
92	
93	
94	
95	
96	
97	
98	
99	
100	

Geben Sie einen Kommentar zur Konstanten ein - auch länger als Eingabefeld, wird im Tooltip sichtbar.

Die vollständige Liste der Assemblerbefehle mit Erklärung in Kurzform

Mnem	OP	OP	- Bedeutung Befehl/Adressierung -	FLAG
NOP			No Operation
CMP			CoMPare D0, D1	NZVC
SWD			SWap D0, D1
SWM			SWap Memory Adr(D0), Adr(D1)
MUL			MULTiplication D0 <- D0*D1	NZVC
DIV			DIVision D0 <- D0/D1 Rest SD0	NZVC
PSA			PuSh All Stack <- A,D,SR Reg.
POA			POp All SR,D,A Reg. <- Stack	NZVC
JSR			Jump SubR. S<-IC+1,IC<-IC+A1/IC<-A0+A1+ST
JIN +A			N=1: IC <- IC + A1 Jump If Negative
JIZ +A			Z=1: IC <- IC + A1 Jump If Zero
JMP +A			IC <- IC + A1 JuMP
RET			RETurn subroutine IC <-Stack
LDC Rg			LoAd Constant Reg xx <- next Byte ##	NZ..
## CCCC CCCC			## Constant Byte
JIN IA			N=1: IC <- A0 +A1 +ST Jump If Negative
JIZ IA			Z=1: IC <- A0 +A1 +ST Jump If Zero
JMP IA			IC <- A0 +A1 +ST JuMP
JIN RG			N=1: IC <- Reg xx Jump If Negative
JIZ RG			Z=1: IC <- Reg xx Jump If Zero
JMP RG			IC <- Reg xx JuMP
INP RG			Reg xx <- INPUt	NZ..
OUT RG			OUTPUt <- Reg xx
PSH RG			PuSH Stack <- Reg xx	NZ..
POP RG			POP Reg xx <- Stack	NZ..
SSR RG			Set Shadow Register Reg xx -> Sxx	NZ..
GSR RG			Get Shadow Register Reg xx <- Sxx	NZ..
BTS RG			BitTest Reg xx with Shadow Register	NZ..
SWN RG			SWap Nibble Reg xx	NZ..
SHL RG			SHift Left Reg xx	NZ..C
SHR RG			SHift Right Reg xx	.Z.C
ROL RG			ROtate Left Reg xx	NZ..C
ROR RG			ROtate Right Reg xx	NZ..C
CLR RG			CLear Register Reg xx <- 0	.Z..
INC RG			INCRement Reg xx	NZVC
DEC RG			DECRement Reg xx	NZVC
NOT RG			Reg xx <- NOT Reg xx (bitweise)	NZ..
AND RG RG			Reg yy <- Reg yy AND Reg xx	NZ..
OR RG RG			Reg yy <- Reg yy OR Reg xx	NZ..
ADD RG RG			Reg yy <- Reg yy + Reg xx	NZVC
ADD RG [RG]			Reg yy <- Reg yy + Adr(Reg xx)	NZVC
SUB RG RG			Reg yy <- Reg yy - Reg xx	NZVC
SUB RG [RG]			Reg yy <- Reg yy - Adr(Reg xx)	NZVC
MOV [RG] RG			MOVe Adr(Reg yy) <- Reg xx	NZ..
MOV RG [RG]			MOVe Reg yy <- Adr(Reg xx)	NZ..
MOV RG RG			MOVe Reg yy <- Reg xx	NZ..
MOV RG SR			MOVe Reg yy <- SR	NZ..
MOV RG SP			MOVe Reg yy <- SP	NZ..
MOV SR RG			MOVe SR <- Reg xx	NZ..
MOV SP RG			MOVe SP <- Reg xx	NZ..
MOV SR CC			MOVe SR <- CC.. (2Bit IO)
MOV RG IA			MOVe Reg xx <- Adr(A0+A1+ST)	NZ..
MOV IA RG			MOVe Adr(A0+A1+ST) <- Reg xx	NZ..
LOD RG			LoAd Data Reg xx <- Adr(h80+A1+ST)	NZ..
STO (IO)			STORE Datei <- Adr(h80+A1+ST) D0 Byte /D1	.Z..
RCL (IO)			ReCall Adr(h80+A1+ST) <-Datei /D1 gel.D0	.ZV.
CPY			CoPY Adr(A1)<-Adr(A0) D0 Byte über D1	.ZV.
STP			StOp P. anhalten, Flags vom. Vorbefehl	vvvv

Ein- und Ausgabe Protokoll

I/O-Protokoll <input checked="" type="checkbox"/>	
>Ein	Aus>
€	€
-128	-128
h80	h80
>1000	0000.
.1000	0000>

Ein- und Ausgabe Protokoll
Befehle INP und OUT - Darstellung der Werte

Interpretation nach IO Bits im Statusregister

```
00 Tastaturzeichen
01 Dezimalzahlen -128 ... 127
10 Hexadezimalzahlen zweistellig hxx
11 Binärzahlen 8 Bit (über beide Spalten!)
    >Eingabe.
    .Ausgabe>
```

I/O-Prot. löschen **X**

... Sichtbarkeit

I/O-Protokoll ☒

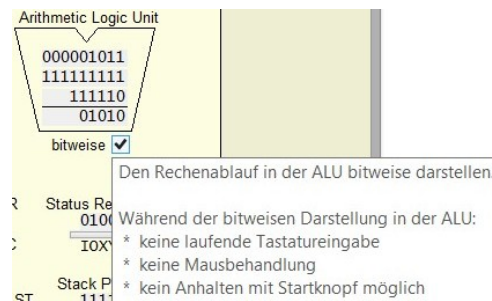
>Ein	Aus>	Sichtbarkeit des Ein- und Ausgabe Protokolls.
------	------	---

... Protokoll löschen

I/O-Prot. löschen X

Ein- und Ausgabe Protokoll ohne Nachfrage löschen.

Die 9 Bit ALU bei der Arbeit



Die einzelnen Bits bei Taktgeschwindigkeit betrachten.

Assemblerbefehl MUL



Multiplikation nach Booth

(bitweise in der ALU mit zusätzlichen temporären Registern)

--- Multiplikation nach Booth ---

Das Register D0 wird im temporären ALU-Register A zwischengespeichert (für Addition).

Das Zweierkomplement -D0 wird gebildet und im temporären Register S gespeichert.

In der unteren Zeile wird schrittweise das 16Bit Ergebnis errechnet, dazu wird das linke Byte mit 0 initialisiert und rechts D1 abgelegt und daneben das Hilfsbit x auf 0 gesetzt.

Die Ablaufsteuerung erfolgt nach den letzten zwei Bits der Ergebniszeile (mit Hilfsbit x):

>> Die Ergebniszeile wird 1 Bit nach rechts geschoben, das Hilfsbit fällt raus, links wird als Vorzeichenerweiterung das zum weggeschobenen Bit identische ergänzt.

```
00, 11    >>    nur >>
01    add >>    D0 wird addiert, dann >>
10    sub >>    -D0 wird addiert, dann >>
```

Das linke Byte der letzten Zeile wird als Zwischensumme jeweils in die oberste Zeile kopiert.
(mit Vorzeichenweiterung)
Nach 8 Schritten ergibt sich das 16Bit Ergebnis (ohne Hilfsbit x) im unteren Doppelregister.
Das rechte Byte (Low-Byte) wird nach D0 gebracht.
Danach Prüfung auf Darstellbarkeit im Bereich -128 .. 127 und setzen der Flags.

Division bitweise in der ALU mit zusätzlichen temporären Registern.

Assemblerbefehl DIV

Arithmetic Logic Unit fta

Vz D0 D1 S

000100110 --- ganzzahlige Division mit Rest ---

000000010

bitweise

Status Reg 0000

IOXY

Stack Po 1111

ST

0000001001

Programm

artadr. von bis

hex

Op. 1 Op. 2

ANSI

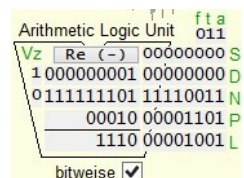
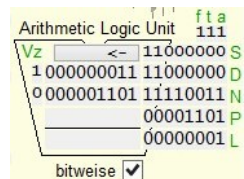
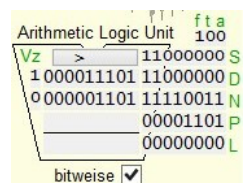
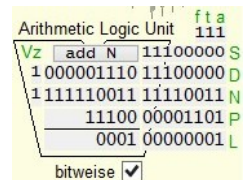
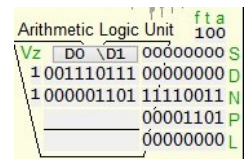
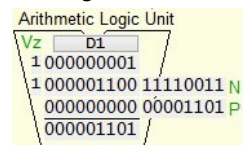
Die Vorzeichen von Dividend und Divisor werden in temporären ALU Speicherbits (Vz) abgelegt.
 Vom Divisor D1 wird das Zweierkomplement gebildet (temporäre ALU-Register: N negativ, P positiv).
 Ist der Dividend in D0 negativ, wird erst sein Zweierkomplement gebildet.
 Die Division der positiven Operanden wird mit ALU-Steuerbits über die ALU-Schieberegister S, D realisiert.
 Dazu wird der positive Dividend in der ALU 'passend' über den positiven Divisor geschoben.
 Erreicht wird das durch Bitvergleich der jeweils übereinander stehenden Bits der Operanden.
 Die Differenzbildung erfolgt dann durch Addition des negativen Divisors.
 Die relevanten Verschiebungen und Additionen erzeugen im temporären Lösungsregister L den Quotienten.
 Anhand der abgelegten Vorzeichen erfolgt nun eventuell die Vorzeichenänderung (Lösung, Rest).

Die Steuerbits werden oben rechts in der ALU in folgender Reihenfolge dargestellt:
 f (first loop) t (two) a (add)

f: Erster Gesamtdurchlauf im Bitvergleich von Dividend und Divisor (f = 1).
 Der Dividend wird bei Bedarf nach rechts in das Hilfsregister D des Dividenten verschoben.
 Im Schiebezeiger Register S wird von links je eine 1 eingeschoben.
 t: Im ersten Teilablauf werden von links die Bits verglichen (Symbol |),
 eventuell wird geschoben (Symbole < >). Ergibt sich daraus, dass zur Größenbestimmung
 auch die Folgebits im zweiten Teilablauf geprüft werden müssen (Symbol :), wird t = 1 gesetzt.
 a: Lösungsbit 1 wird an den Quotienten angehängen, dann Addition des negativen Divisors (aktueller Rest),
 ein Bit aus dem Hilfsregister D des Dividenten wird nachgeschoben (Symbol < -).
 Dies erfolgt nur, wenn im darüberstehenden Schiebezeiger Register S links noch eine 1 steht.

Anschließend erfolgt je nach Vorzeichenbit in Vz eventuell eine Vorzeichenumkehr.
 Der Quotient wird nach D0, der Rest nach SD0 gebracht (Vorzeichen Rest wie Vorzeichen Dividend).

Auszüge:



Daten in Datei speichern

Aus output.hcx

Ausgabe in Daten-Datei mit dem Befehl STO.

Dateistruktur: Textdatei, ein Wert je Zeile.
 Inhalt der Datei - Interpretation nach IO Bits im Statusregister

00 Tastaturzeichen
 01 Dezimalzahlen -128 ... 127
 10 Hexadezimalzahlen zweistellig xx
 11 Binärzahlen 8 Bit

Wurden keine Daten geschrieben, wird das Z-Flag gesetzt.
 Ist kein Dateiname eingetragen, wird die Dateiauswahl angezeigt.

Daten aus Datei einlesen

Datei: input.hcx Ein - Aus output.hcx

Prozes

Daten

D0 000

Datenstruktur: Textdatei, ein Wert je Zeile.
 Inhalt der Datei - Interpretation nach IO Bits im Statusregister

D1 000

00 Tastaturzeichen
 01 Dezimalzahlen -128 ... 127
 10 Hexadezimalzahlen zweistellig xx
 11 Binärzahlen 8 Bit

Adres

A0 000

A1 000

Wurden keine Daten gelesen, wird das Z-Flag gesetzt.
 Sind zu viele Daten in der Datei, wird am Ende des RAM abgebrochen und das V-Flag gesetzt.
 Es wird auch durch Werte die den Bereich nicht beachten gesetzt.
 Ist kein Dateiname eingetragen, wird die Dateiauswahl angezeigt.

Eingabe (laufend ein Tastaturzeichen oder mit Befehl INP)

HC680 Binärcode

Eingabe: ANSI

Ausgabe: ANSI

>Ein Aus>

-128 128 €

10000000 h80 OK

Eingabe ...

Takt: 1.0 Hz

Breakpoints

7F hex

Datei: inp

Prozessor

Datenreg

D0 00000

D1 00000

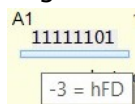
Adres

A0 00000

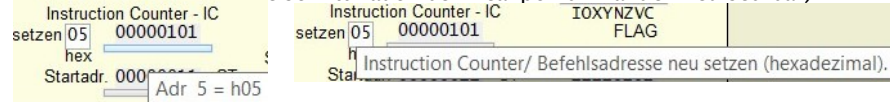
Laufende Eingabe eines Tastaturzeichens. Eingabe mit Alt xxxx möglich.
 Die Eingabe erfolgt zu jedem Takt auch ohne in das Eingabefeld zu klicken!
 Der binäre ANSI-Code wird bei Adresse hf6 abgelegt.

Beim Befehl INP alternativ auch Eingabe einer Dezimalzahl im Bereich -128 ... 127
 oder Eingabe einer Hexadezimalzahl in der Form hxx bzw. Hxx oder \$xx.
 Hexadezimalziffern 0..9, A..F oder a..f - ungültige Ziffern werden in 0 umgewandelt.
 Bei INPut ist die Eingabe mit OK zu quittieren.
 Die Eingabe wird dann im Register und bei Adresse hf6 gespeichert
 und im Ein-Ausgabeprotokoll angezeigt. Die Darstellung erfolgt dort nach dem
 Inhalt der IO Bits im Status Register: 00 Zeichen, 01 dezimal, 10 hexadezimal, 11 binär.

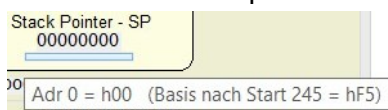
Registerinhalt (binär, dazu dezimal, hexadezimal im Tooltip)



Befehlzähler (bei Einzelschritt – auch bei Breakpoint– manuell neu setzbar)



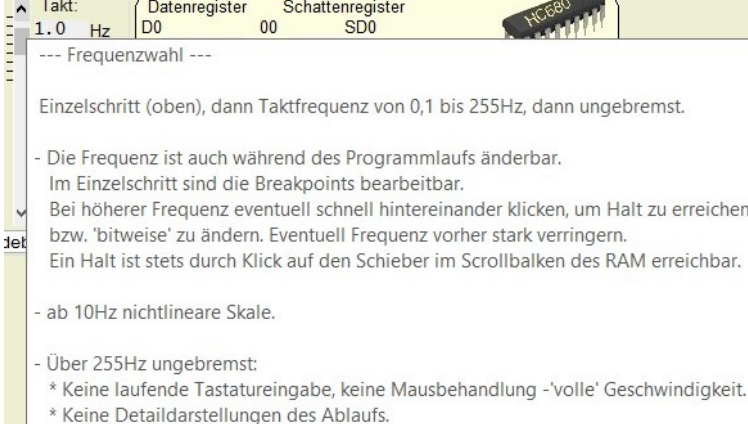
Basisadresse Stackpointer



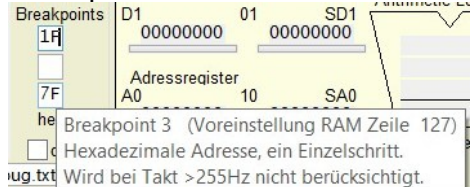
Startadresse



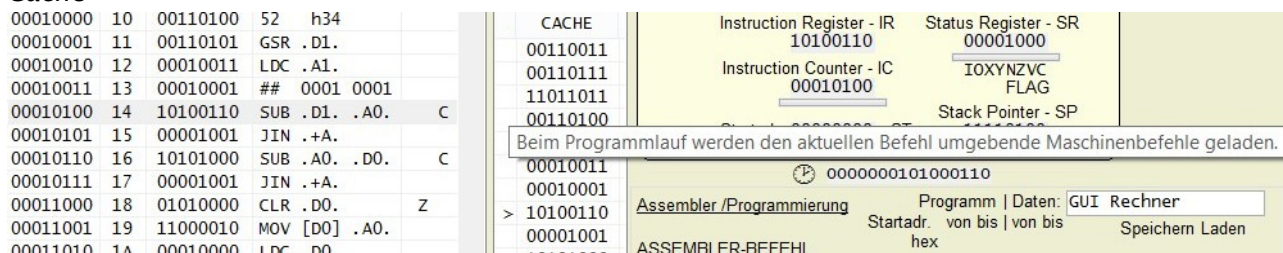
Takt



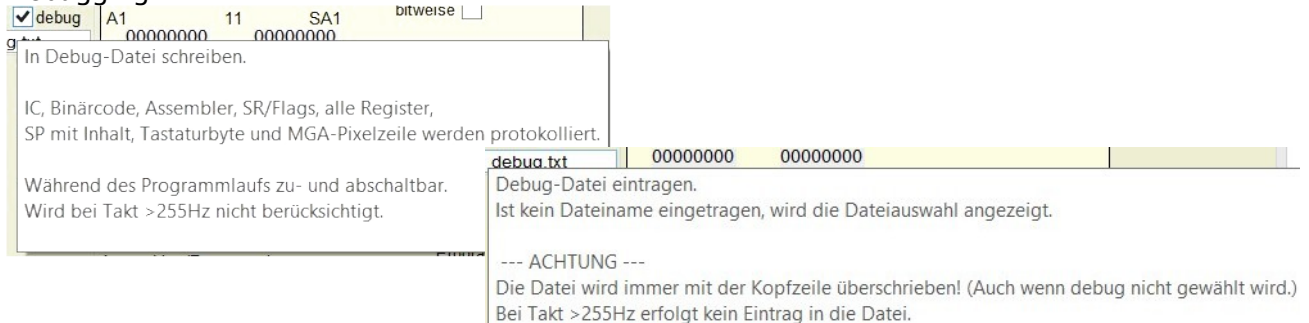
Breakpoints



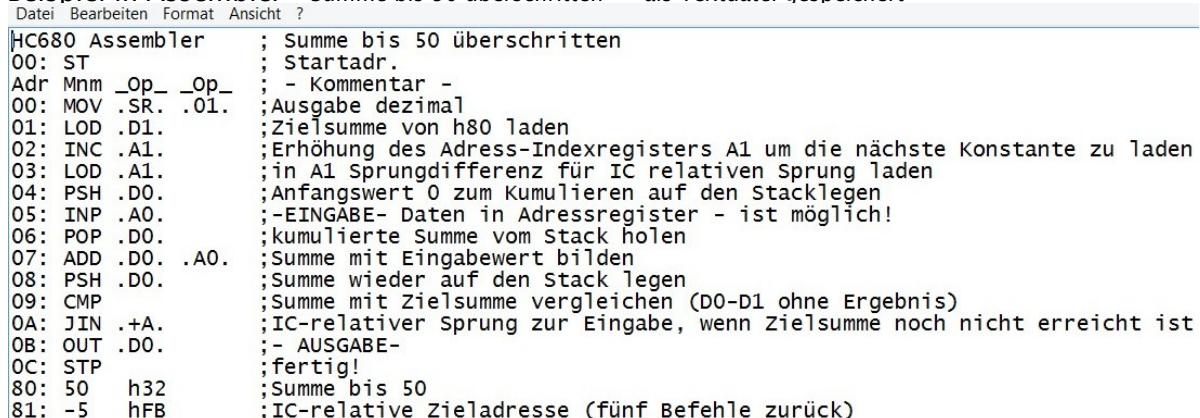
Cache



Debugging



Beispiel in Assembler "Summe bis 50 überschritten" – als Textdatei gespeichert



Bei Erstellung mit einem Texteditor können der Doppelpunkt, die Punkte vor und nach den Operanden und das Semikolon entfallen. Nach Mnm und zwischen den Operanden sind bis zu 4 Leerzeichen zulässig, vor Mnm zwei. Vor den Daten sind bis zu fünf Leerzeichen zur Trennung gestattet, damit kann rechtsbündig notiert werden.

Laden

←

Assemblerdatei laden, speichern, verschieben

Laden Sie eine HC680 Assemblerdatei in den Arbeitsspeicher.

Fehlerhafte Assemblerbefehle werden als NOP interpretiert.
Der Doppelpunkt hinter der Befehlsadresse ist optional, dafür reichen ein oder zwei Leerzeichen.
Auch die Punkte um die Operanden sind optional, dann ein bis vier Trenn-Leerzeichen setzen.
Im Datenbereich ab Adresse h80 wird nur der linke Wert bzw. das Zeichen eingelesen.
Zur besseren Übersicht können die Daten rechtsbündig gesetzt werden.
Der Doppelpunkt nach der Datenadresse kann entfallen, es sind bis zu fünf Leerzeichen gestattet.
Die 2. Spalte (einheitlich hexadezimal) ist optional. Sie wird neu berechnet.
Ab Spalte 21 wird der Kommentar eingelesen, das Semikolon davor ist optional.
Bitte die in der Datei gespeicherte Startadresse beachten.

Speichern Laden

→ ←

Speichern Sie eine HC680 Assemblerdatei ab.

Die Startadresse wird gespeichert.
Es werden von Anfangs- (min. 00) bis Endadresse (max. 7F) das Programm, und die Daten von (min. 80) bis (max. FF) gespeichert.
Sind keine Werte eingetragen wird (der Teil) nicht gespeichert.

Ver schieben

m Be

Verschieben Sie die Assemblerbefehle und/oder die Daten im RAM.

Den Versatz dezimal oder hexadezimal hxx mit Vorzeichen angeben.
Es wird von den eingetragenen Anfangs- bis zu den Endadressen verschoben.
Die vorhandenen Inhalte werden überschrieben, freie Zeilen auf Null gesetzt.
Der RAM-Ausschnitt wird als asm_tmp.txt gespeichert.
Der komplette RAM wird vorher unter asm_bak.txt gesichert.

Date

um -231 Ver schieben

-h17 | Verschiebung von Programm und/oder Daten im Speicher, dezimal oder hexadezimal (-)hxx bzw. Hxx, \$xx

Adressbereiche der Assemblerdatei

Programm | Daten: Testprogramm

dr. von bis | von bis

hex 00 6A 80 FF

Speichern Laden

1 Hexadezimale Anfangsadresse des Binärcode zur Speicherung/ Verschiebung in der Assemblerdatei.

Programm | Daten: Testprogramm

dr. von bis | von bis

hex 00 6A 80 FF

Speichern Laden

1 Hexadezimale Endadresse Programm - für Speicherung/ Verschiebung

Daten: Testprogramm

von bis

80 FF

Speichern Laden

Hexadezimale Datenadresse Anfang - für Speicherung/ Verschiebung

Daten: Testprogramm

von bis

80 FF

Speichern Laden

2 Hexadezimale Datenadresse Ende - für Speicherung/ Verschiebung

Maschinenprogramm / Binärcode ausführen

klicken

–

Zwischenstopps möglich

bitweise in der ALU

langsamer geht nicht

volle Geschwindigkeit

HC680 Binärcode

ausführen

Takt: 1.0 Hz

HC680 Binärcode

anhalten

Takt: 0.2 Hz

HC680 Binärcode

wieder anhalten

Takt: 0.5 Hz

HC680 Binärcode

... Bit ...

Takt: 0.2 Hz

HC680 Binärcode

STOP

Step ...

Takt: Step Hz

weiter bitte

HC680 Binärcode

Takt > 255

Takt: Speed Hz

bei INPUT

abbrechen?

fertig!

HC680 Binärcode

Eingabe ...

Takt: 1.0 Hz

HC680 Binärcode

STOP

Datei: input.hcx

Ein

Programm mit x sofort abbrechen, sonst gelben Kreis klicken.

HC680 Binärcode

gestoppt