Prozessor HC680 fiktiv

Dokumentation der Simulation

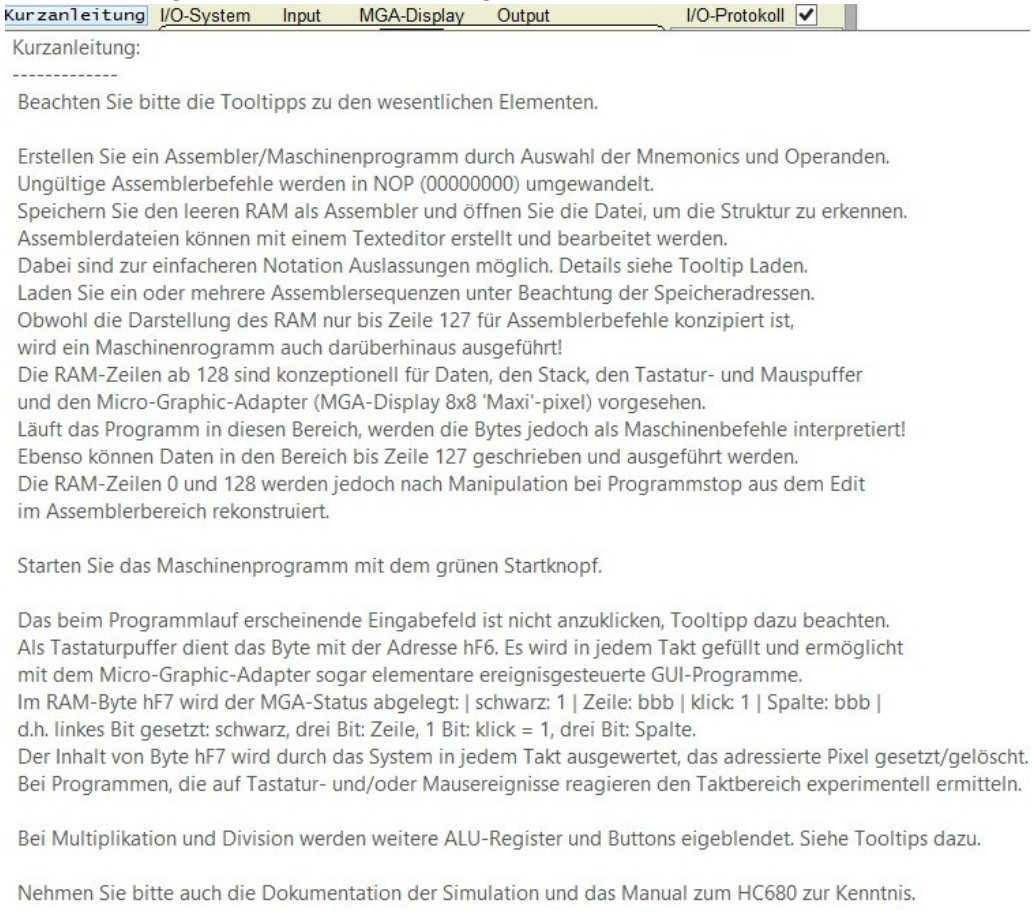
Die Simulation umfasst die Struktur und Funktionalität des Prozessors und wichtiger Baugruppen des Systems. Dabei werden in einem Simulationsfenster sowohl der Ablauf des Maschinenprogramms als auch dessen Erstellung mittels Assembler realisiert. Assemblerprogramme können gespeichert und geladen werden.

Daten des Systems

Prozessor:	HC 680 fiktiv
Verarbeitungsbreite:	8 Bit Daten und Adressen
Befehlslänge:	1Byte
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+1Bit 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

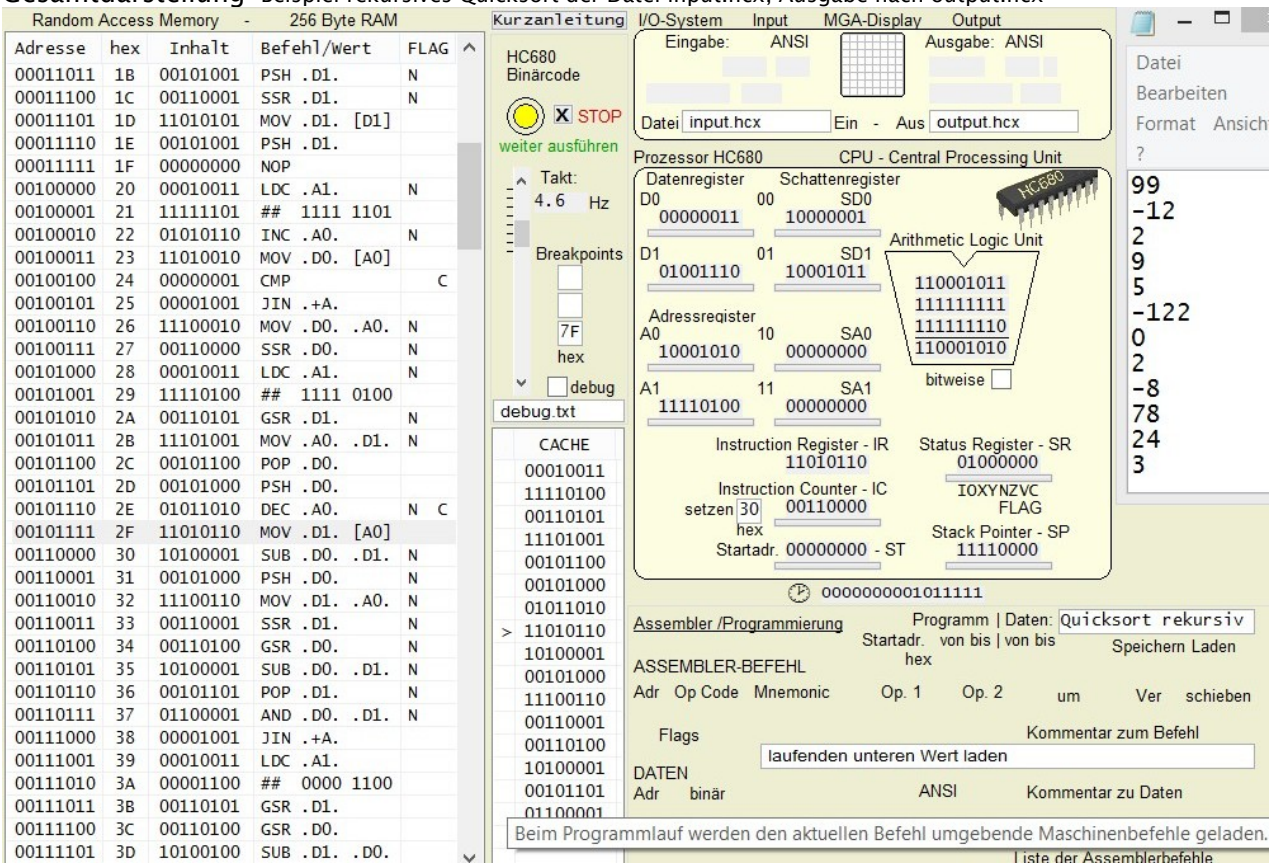
---

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



Gesamtdarstellung

Beispiel rekursives Quicksort der Datei input.hcx, Ausgabe nach output.hcx

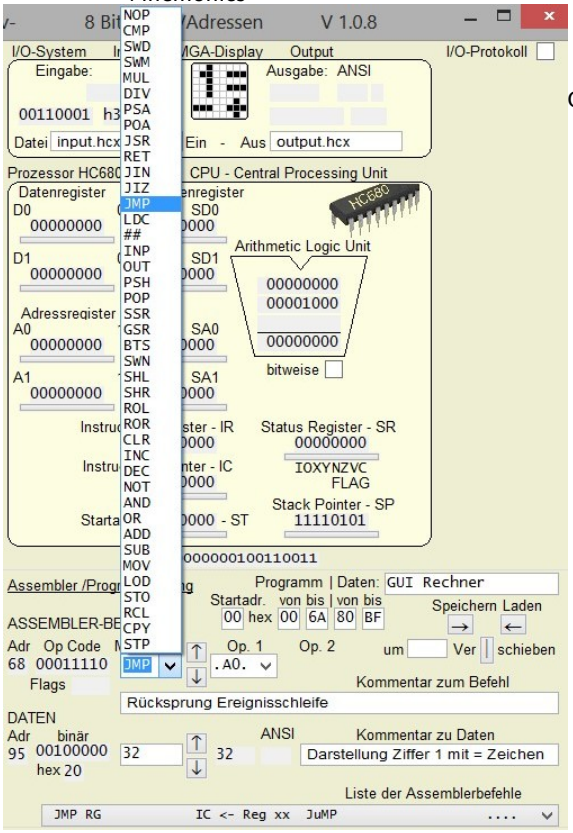


Die Hintergrundfarbe kann in der Datei INI.TXT im Ordner verändert werden. Fehlt die Datei, gilt der Standard.

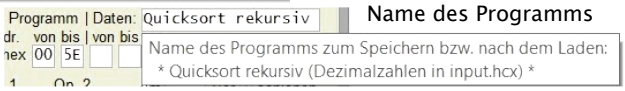
aufräumen

Assembler

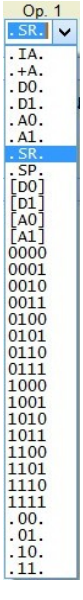
Mnemonics



Name des Programms



Operanden



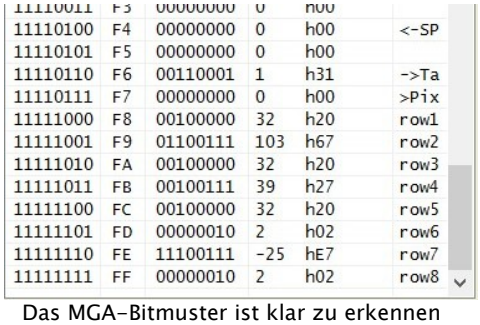
Display

Beispiel mit Klickbereichen

(Zifferneingabe hier mit Tastatur)

Stackpointer,Tastaturbyte,

Pixelzeile, MGA-RAM



Das MGA-Bitmuster ist klar zu erkennen

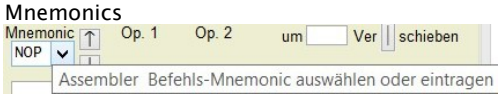
Adresse

Op-Code

Flags

Daten

Mnemonics

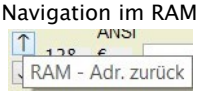


Kommentare

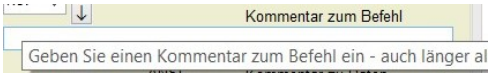
Assemblerbefehle,

Liste mit Erklärung in Kurzform

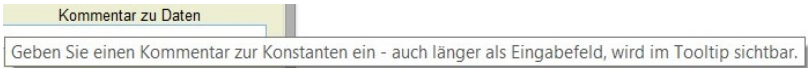
Navigation im RAM



Kommentare Befehl

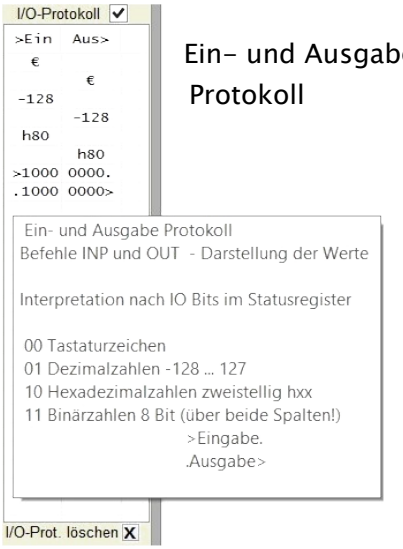


Daten



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          RETturn 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
Mnem OP OP    - Bedeutung Befehl/Adressierung -          FLAG
```

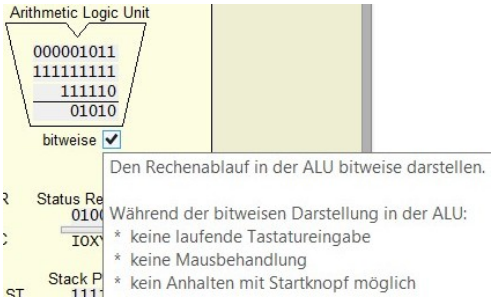


Ein- und Ausgabe Protokoll

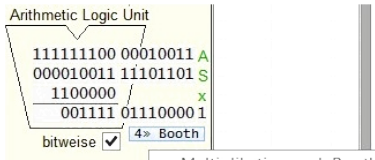
... Sichtbarkeit

... Protokoll löschen

Die 9 Bit ALU bei der Arbeit

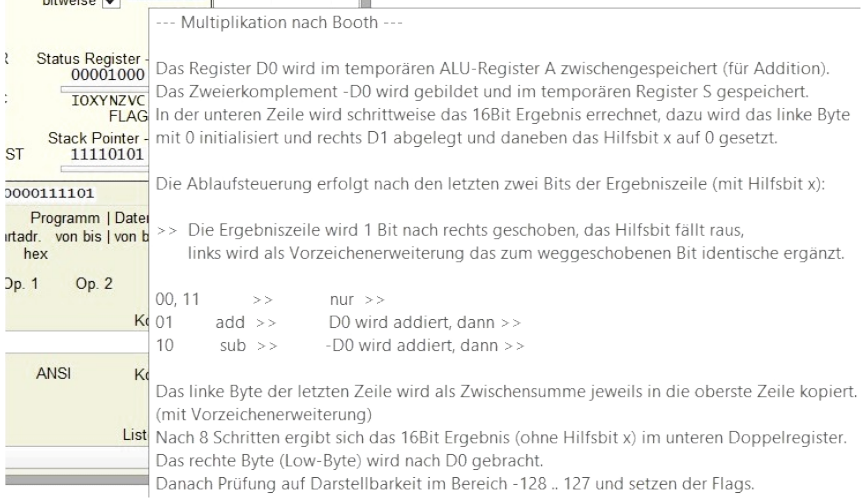


Assemblerbefehl MUL



Die einzelnen Bits bei Taktgeschwindigkeit betrachten.

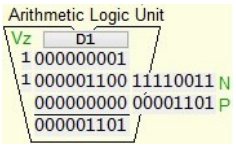
Multiplikation nach Booth
(bitweise in der ALU mit zusätzlichen temporären Registern)

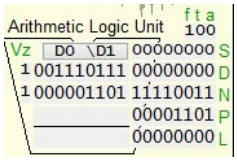


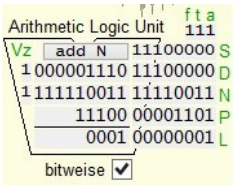
Division bitweise in der ALU mit zusätzlichen temporären Registern.

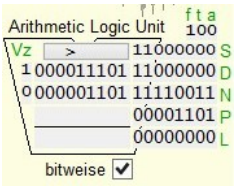
Assemblerbefehl DIV

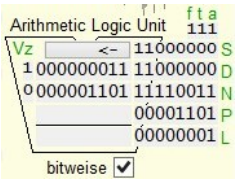
Auszüge:

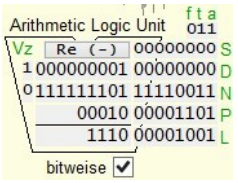




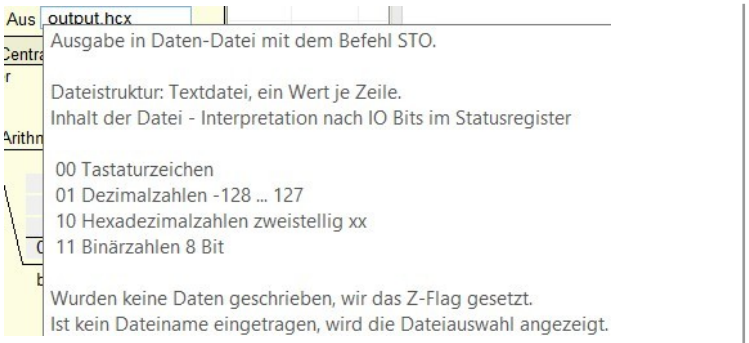




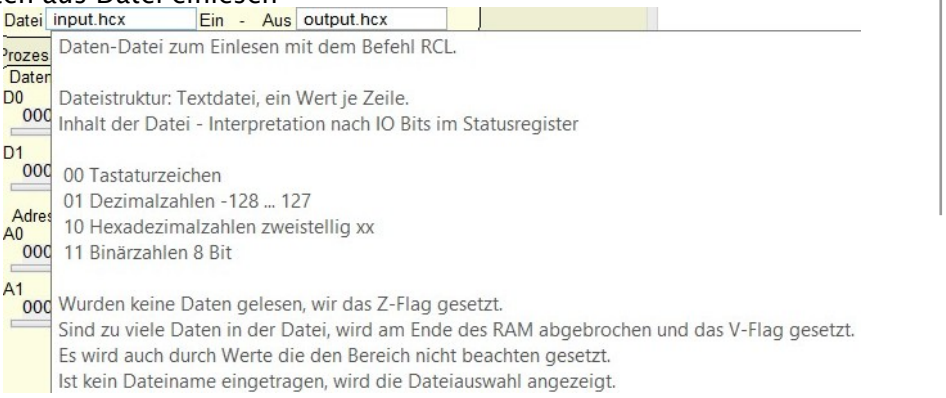




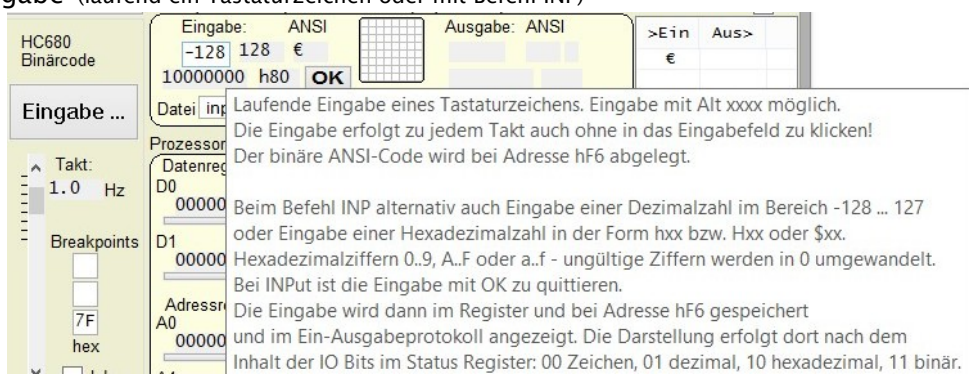
Daten in Datei speichern



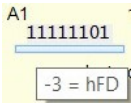
Daten aus Datei einlesen



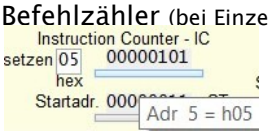
Eingabe (laufend ein Tastaturzeichen oder mit Befehl INP)

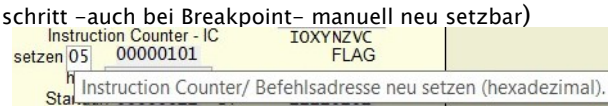


Registerinhalt (binär, dazu dezimal, hexadezimal im Tooltip)

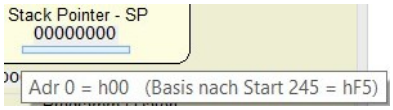


Befehlzähler (bei Einzelschritt –auch bei Breakpoint– manuell neu setzbar)



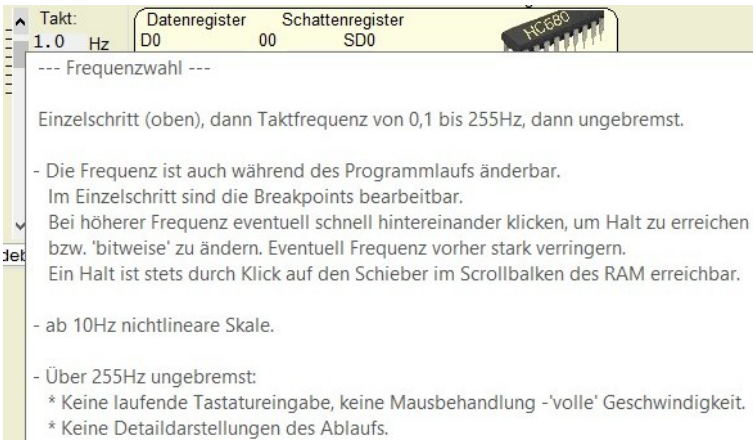


Basisadresse Stackpointer

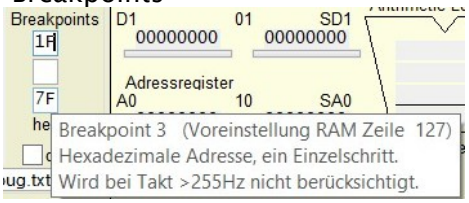


Startadresse

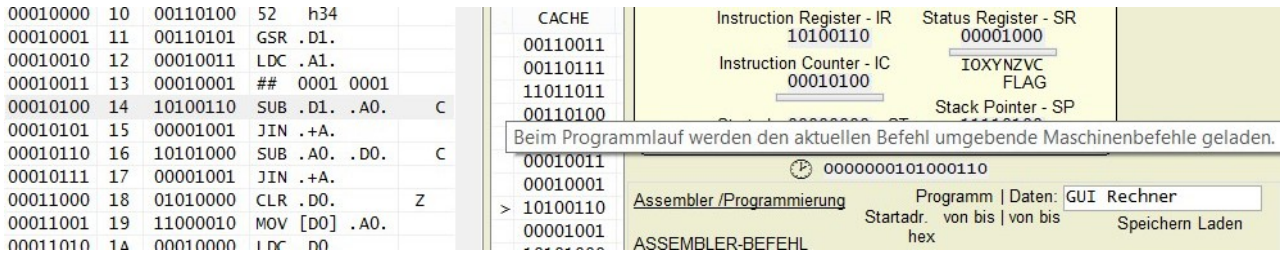
Takt



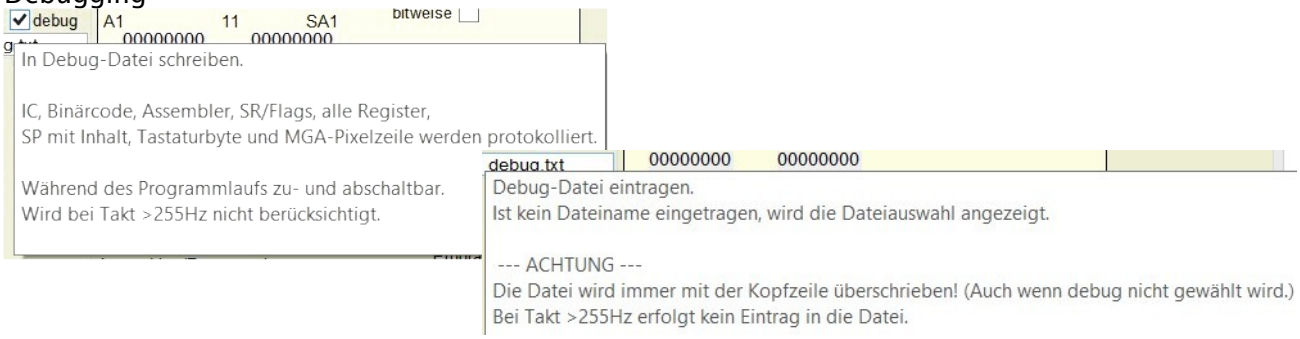
Breakpoints



Cache



Debugging



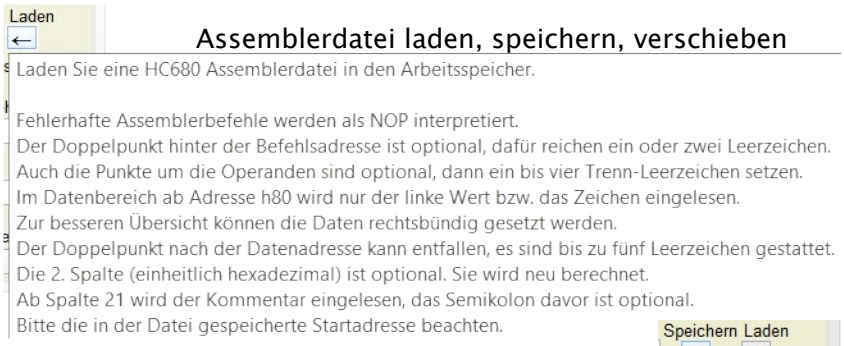
Beispiel in Assembler " Summe bis 50 überschritten" – als Textdatei gespeichert

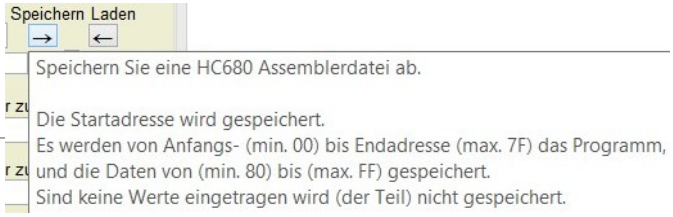
Datei Bearbeiten Format Ansicht ?

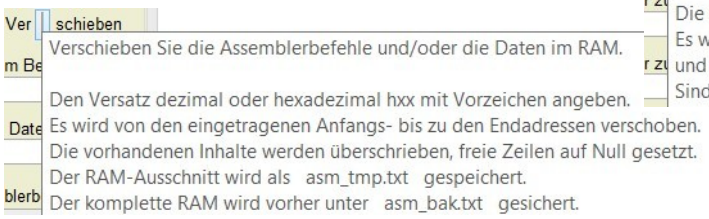
```
HC680 Assembler     ; Summe bis 50 überschritten
00: ST              ; Startadr.
Adr Mnm _Op_ _Op_   ; - Kommentar -
00: MOV .SR. .01.   ;Ausgabe dezimal
01: LOD .D1.        ;Zielsumme von h80 laden
02: INC .A1.        ;Erhöhung des Adress-Indexregisters A1 um die nächste Konstante zu laden
03: LOD .A1.        ;in A1 Sprungdifferenz für IC relativen Sprung laden
04: PSH .D0.        ;Anfangswert 0 zum Kumulieren auf den Stacklegen
05: INP .A0.        ;-EINGABE- Daten in Adressregister - ist möglich!
06: POP .D0.        ;kumulierte Summe vom Stack holen
07: ADD .D0. .A0.   ;Summe mit Eingabewert bilden
08: PSH .D0.        ;Summe wieder auf den Stack legen
09: CMP             ;Summe mit Zielsumme vergleichen (D0-D1 ohne Ergebnis)
0A: JIN .+A.        ;IC-relativer Sprung zur Eingabe, wenn Zielsumme noch nicht erreicht ist
0B: OUT .D0.        ;- AUSGABE-
0C: STP             ;fertig!
80: 50   h32        ;Summe bis 50
81: -5   hFB        ;IC-relative Zieladresse (fünf Befehle zurück)
```

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.

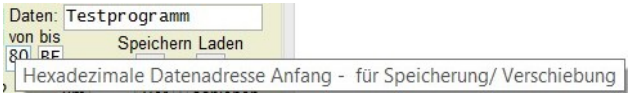
Assemblerdatei laden, speichern, verschieben

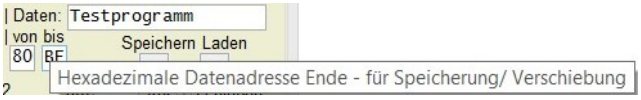






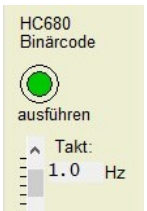
Adressbereiche der Assemblerdatei



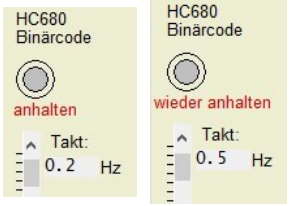


Maschinenprogramm / Binärcode ausführen

klicken –



Zwischenstopps möglich



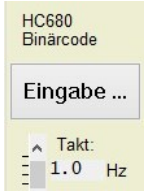
bitweise in der ALU

langsamer geht nicht

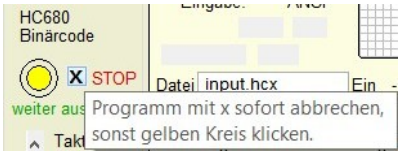
volle Geschwindigkeit

weiter bitte

bei INPut



abbrechen?



fertig!