

# Prozessor HC680 fiktiv

## Dokumentation der Simulation



Bilder Windows Variante

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+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 Programmablauf änderbar
Debug:	beliebig abschnittsweise in wählbare Datei
Assembler:	40 Mnemonics, 57 Befehle durch Adressierungsarten - siehe Manual

Die Bedienungshilfe wird über umfangreiche, Tooltips realisiert (Windows Variante 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 Tastatur- und Mauseingabe 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 Programmstopp aus dem Edit im Assemblerbereich rekonstruiert.

Starten Sie das Maschinenprogramm mit dem grünen Startknopf.

Das beim Programmablauf erscheinende Eingabefeld ist nicht anzuklicken, Tooltip 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 Mauseingabe 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.

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

The screenshot shows the CPU simulation interface with several panels:

- Random Access Memory:** A table with columns for Adresse (hex), Inhalt, Befehl/wert, and FLAG. It lists memory addresses from 00011011 to 00111101 and their corresponding instructions and flags.
- HC680 Binärcode:** A section with a 'STOP' button and a 'weiter ausführen' button. It shows the current clock speed at 4.6 Hz and a 'debug.txt' file.
- Prozessor HC680:** A detailed view of the CPU components including:
  - Datenregister (D0-D7):** Values like 00000011, 00000001, 01001110, 10001011.
  - Schattenregister (SD0-SD7):** Values like 00, 10000001, 01, 10001011.
  - Arithmetische Logik Unit (ALU):** Shows the current operation: 110001011 + 111111111 = 111111110.
  - Adressregister (A0-A7):** Values like 10001010, 00000000, 11110100, 00000000.
  - Status Register (SR):** 01000000.
  - Instruction Register (IR):** 11010110.
  - Stack Pointer (SP):** 11110000.
- Assembler/Programmierung:** Shows the current program 'Quicksort rekursiv' and the current instruction being assembled: 'laufenden unteren Wert laden'.

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

Two warning dialog boxes are shown:

- VORSICHT!** Den gesamten RAM, die Flag-Spalte und die Kommentare nach Sicherheitsabfrage löschen.
- ACHTUNG!** Die Spalte der zuletzt gesetzten Flags ohne Sicherheitsabfrage löschen.

aufräumen

Assembler

The screenshot shows the Assembler interface with several panels and annotations:

- Mnemonics:** A list of assembly instructions like NOP, CMP, SWD, MUL, etc.
- Operanden:** A list of operands like .IA, .A, .D, .D1, .A0, .A1, .SR, .SP, etc.
- Stackpointer, Tastaturbyte, Pixelzeile, MGA-RAM:** A table showing memory addresses and their contents, with annotations like 'Das MGA-Bitmuster ist klar zu erkennen'.
- Annotations:**
  - 'Adresse Op-Code Flags Daten' pointing to the assembly table.
  - 'Operanden Display' pointing to the operand list.
  - 'Name des Programms' pointing to the program name field.
  - 'Mnemonics' pointing to the mnemonic list.
  - 'Kommentare Assemblerbefehle, Liste mit Erklärung in Kurzform' pointing to the comment field.
  - 'Navigation im RAM' pointing to the RAM navigation buttons.

Kommentare

Befehl

Kommentar zum Befehl

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

Daten

Kommentar zu Daten

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			RETRurn 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

I/O-Protokoll

```

>Ein Aus>
€
€
-128
-128
h80
h80
>1000 0000.
.1000 0000>

```

### Ein- und Ausgabe Protokoll

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>

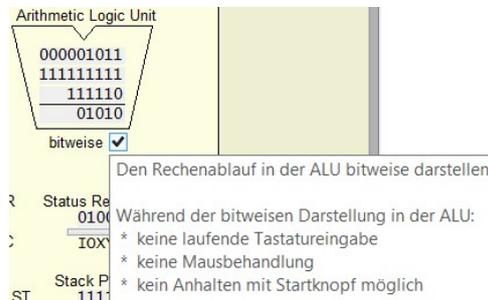
Sichtbarkeit des Ein- und Ausgabe Protokolls.

### ... 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

Arithmetic Logic Unit

```

111111100 00010011 A
000010011 11101101 S
1100000 x
001111 01110000 1
bitweise
4> Booth

```

Status Register: 00001000

IOXNZVC

FLAG

Stack Pointer: 11110101

0000111101

Programm | Datei | von bis | von bis | hex

Op. 1 Op. 2

ANSI

List

### 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 Vorzeichenerweiterung)

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 – nur Windows Variante.

Assemblerbefehl DIV

Arithmetic Logic Unit f t a  
 Vz D0 D1 S  
 000100110  
 000000010  
 --- ganzzahlige Division mit Rest ---

bitweise

Status Reg 0000  
 IOXY

Stack Po 1111  
 ST

000001001  
 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 Dividenden 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 angehängt, dann Addition des negativen Divisors (aktueller Rest),  
 ein Bit aus dem Hilfsregister D des Dividenden 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:

Arithmetic Logic Unit f t a  
 Vz D1  
 1 000000001  
 1 000001100 11110011 N  
 000000000 00001101 P  
 000001101

Arithmetic Logic Unit f t a  
 Vz D0 D1 S  
 1 000000000  
 1 001110111 00000000 D  
 1 000001101 11110011 N  
 00001101 P  
 00000000 L

Arithmetic Logic Unit f t a  
 Vz add N 11100000 S  
 1 000001110 11100000 D  
 1 111110011 11110011 N  
 11100 00001101 P  
 0001 00000001 L

bitweise

Arithmetic Logic Unit f t a  
 Vz > 11100000 S  
 1 000011101 11000000 D  
 000001101 11110011 N  
 00001101 P  
 00000000 L

bitweise

Arithmetic Logic Unit f t a  
 Vz <- 11100000 S  
 1 000000011 11000000 D  
 000001101 11110011 N  
 00001101 P  
 00000001 L

bitweise

Arithmetic Logic Unit f t a  
 Vz Re (-) 00000000 S  
 1 000000001 00000000 D  
 011111101 11110011 N  
 00010 00001101 P  
 1110 00001001 L

bitweise

Daten in Datei speichern

Ausgabe: 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.

web Variante in Zwischenablage

Daten aus Datei einlesen

Datei: input.hcx Ein - Aus output.hcx  
 Daten-Datei zum Einlesen mit dem Befehl RCL.

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 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  
 -128 128 €  
 10000000 h80 OK

Eingabe ...

Datei: inp  
 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.

Prozessor  
 Datenreg  
 D0 000000  
 D1 000000  
 Adresse A0 000000

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.



## Laden **Assemblerdatei laden, speichern, verschieben (Windows Variante: speichern in Zwischenablage)**

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.

### Verschieben

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.

um -231 Verschieben  
-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 | Speicher Laden  
hex 00 6A 80 BE  
1 Hexadezimale Anfangsadresse des Binärcode zur Speicherung/ Verschiebung in der Assemblerdatei.

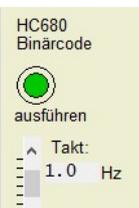
Programm | Daten: Testprogramm  
dr. von bis | von bis | Speicher Laden  
hex 00 6A 80 BE  
1 Hexadezimale Endadresse Programm - für Speicherung/ Verschiebung

Daten: Testprogramm  
von bis | Speicher Laden  
80 BE  
Hexadezimale Datenadresse Anfang - für Speicherung/ Verschiebung

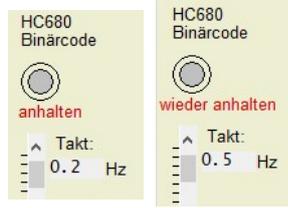
Daten: Testprogramm  
von bis | Speicher Laden  
80 BE  
2 Hexadezimale Datenadresse Ende - für Speicherung/ Verschiebung

## Maschinenprogramm / Binärcode ausführen

klicken



Zwischenstopps möglich



bitweise in der ALU



langsamer geht nicht



volle Geschwindigkeit



bei INPUT



abbrechen?



fertig!

