Processor HC680 fictive

documentation of simulation

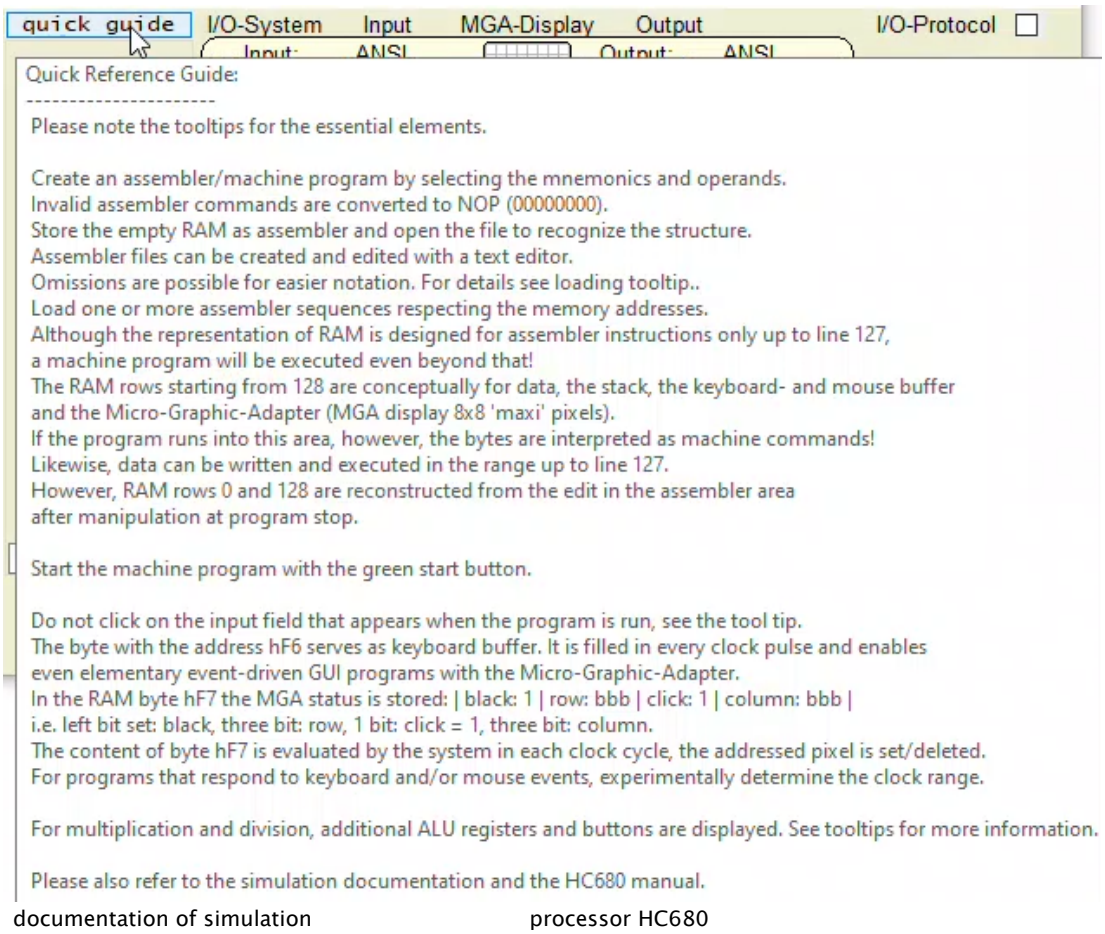
The simulation includes the structure and functionality of the processor and important assemblies of the system. Thereby both the execution of the machine program and its creation by the assembler are realized in a simulation window. Assembler programs can be stored and loaded.

data of the system

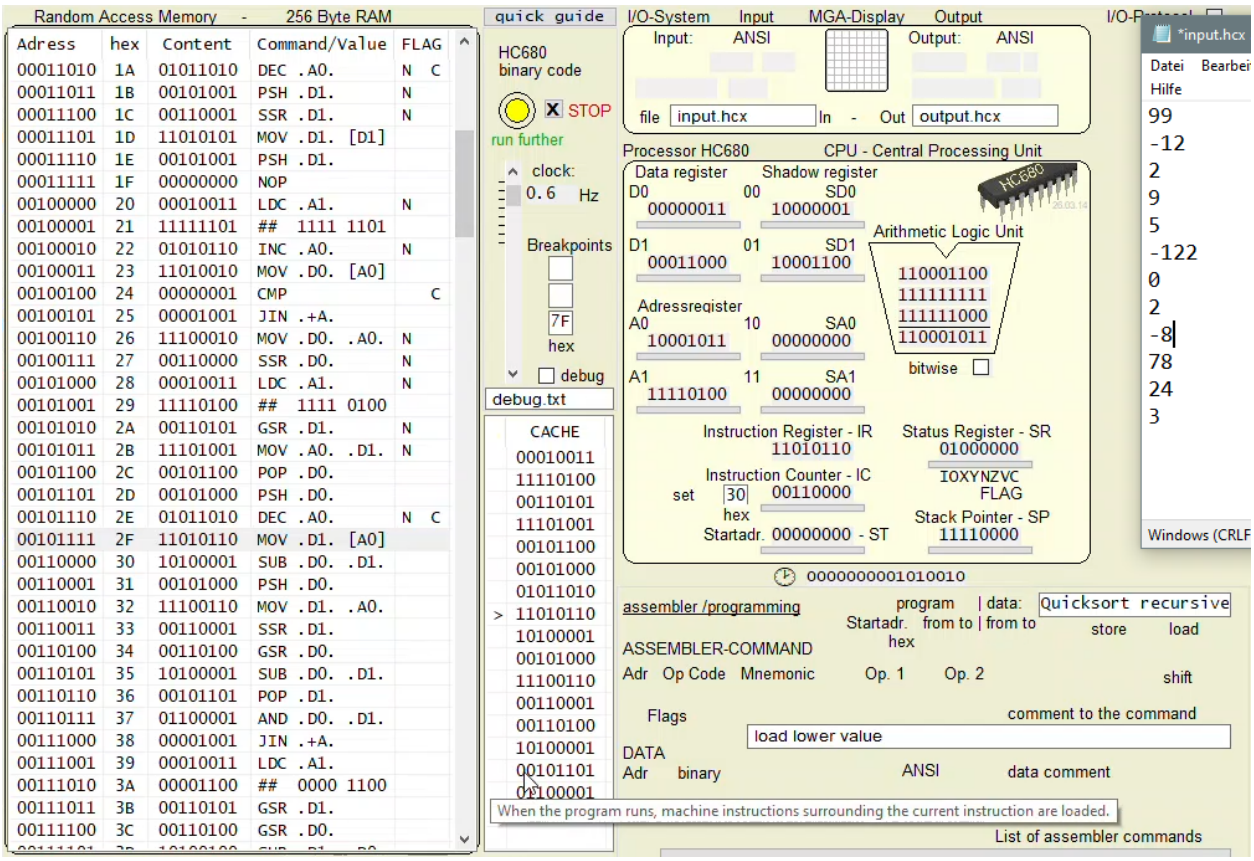
processor:	HC 680 fictive
processing width:	8 bit data and addresses
command length:	1 byte
address range:	256 byte
clock pulse:	Clock: single step, 0.1Hz to 255Hz, unbraked
registers:	4 general registers with shadow register, start address (ST) instruction register (IR), instruction counter (IC), status register (SR), stack pointer (SP)
flags:	Negative (N), Zero (Z), oVerflow (V), Carry (C) and I/O Bits
arithmetic unit:	Arithmetic Logic Unit (ALU), 8+1 bit sign extension for addition, multiplication by Booth, division with remainder, ALU bitwise representable
numberspace:	decimal from -128 to 127
graphic:	8x8 maxi pixel display, 8 byte shared memory
in and output:	by I/O bits: keyboard characters, decimal, hexadecimal, - output also binary
files:	by I/O bits: keyboard characters, decimal, hexadecimal, binary
breakpoints:	3 - changeable during program execution
debug:	any section, into selectable file
assembler:	40 mnemonics, 57 commands by addressing modes - see manual

---

The user assistance is realized via extensive, tooltips (can be switched off).

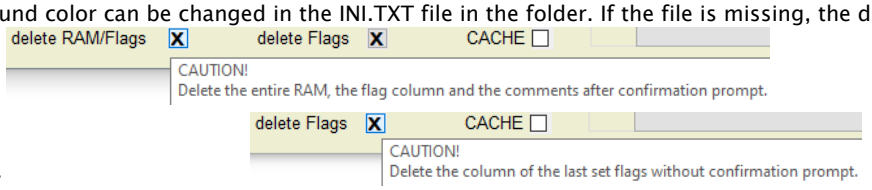


Overall view, example recursive quicksort of the file (german Datei) input.hcx, output to output.hcx.

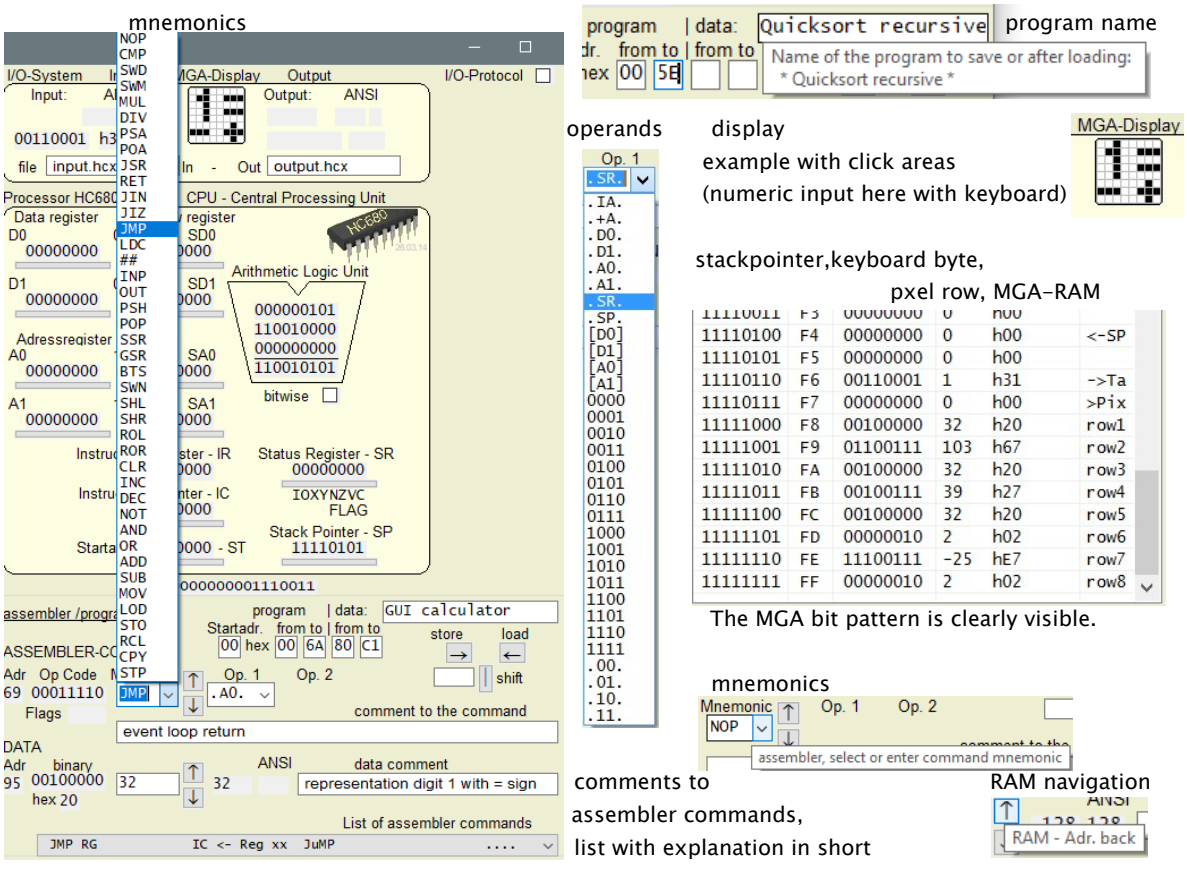


The background color can be changed in the INI.TXT file in the folder. If the file is missing, the default applies.

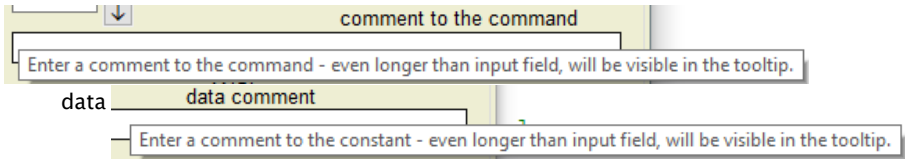
cleanup



assembler

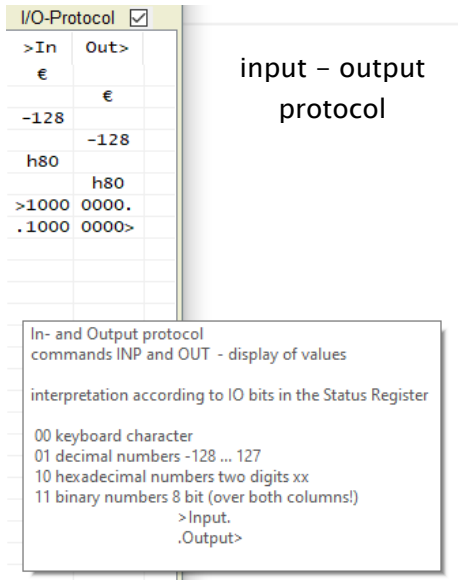


comments command



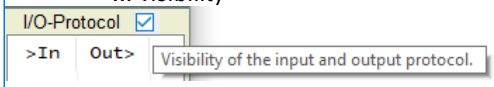
The complete list of assembler commands with explanation in short form.

```
Mnem OP OP    - Meaning Command/Addressing -                    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  (bitwise)                   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 file  <- Adr(h80+A1+ST) D0 byte /D1 .Z..
 RCL (IO)     ReCalL Adr(h80+A1+ST)<- file  /D1 read D0 .ZV.
 CPY          CoPY Adr(A1)<-Adr(A0)  D0 byte via D1             .ZV.
 STP          StoP P. stop, flags from pre-command              vvvv
Mnem OP OP    - Meaning Command/Addressing -                    FLAG
```

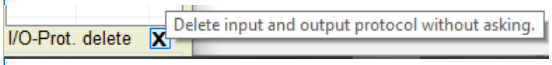


input – output protocol

... visibility

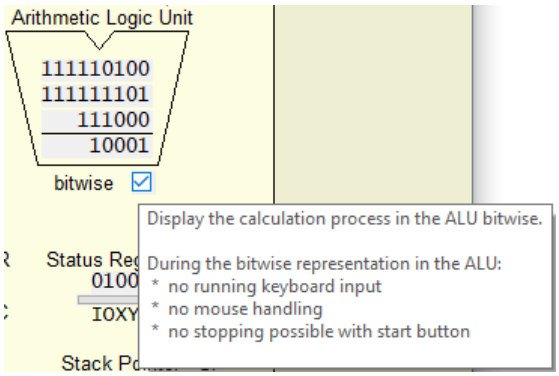


... clear protocol

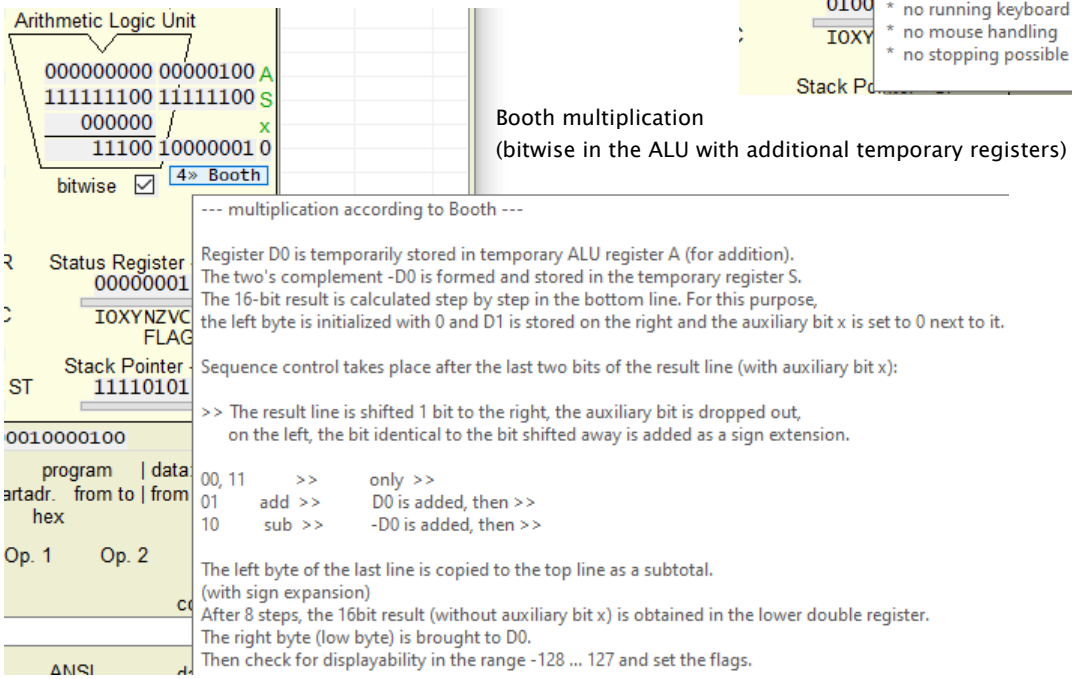


The 9-bit ALU at work.

View the individual bits at clock speed.



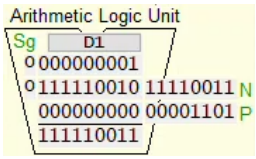
assembler command MUL

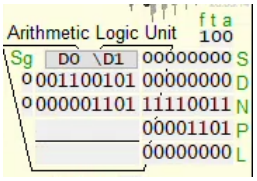


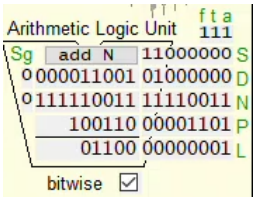
Booth multiplication
(bitwise in the ALU with additional temporary registers)

Division bitwise in the ALU with additional temporary registers.

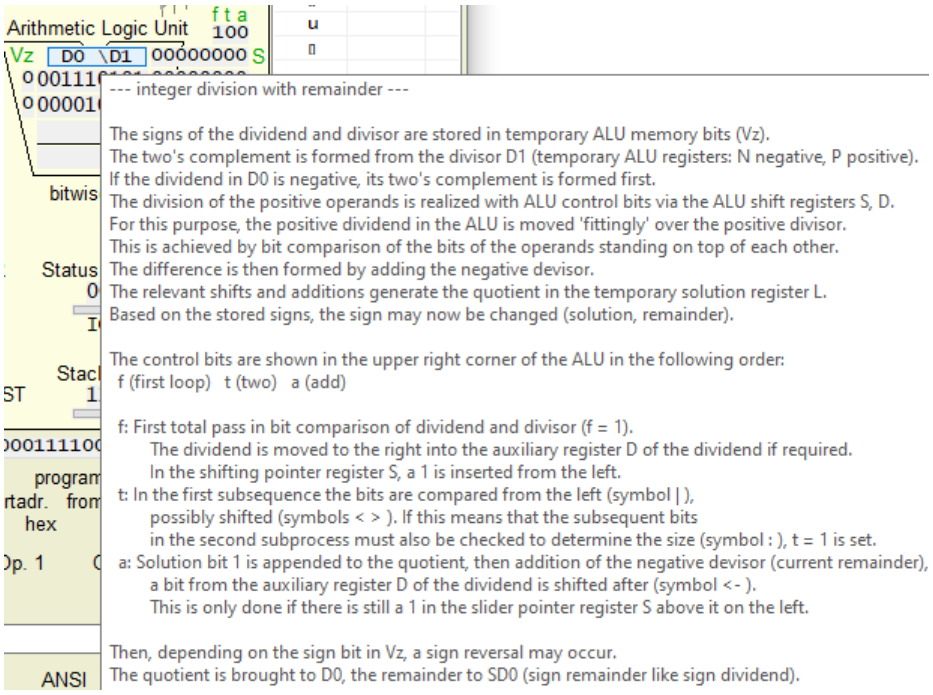
Excerpts:

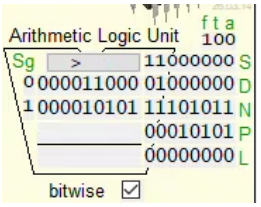




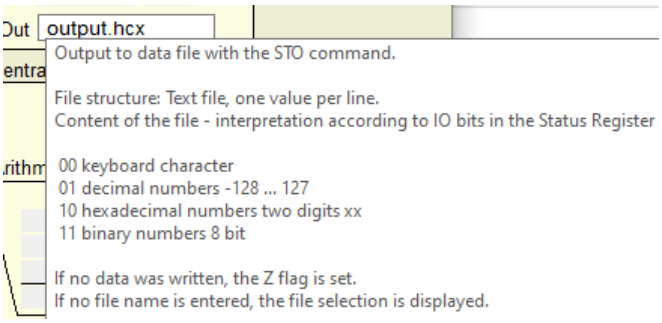


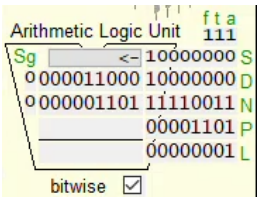
assembler command DIV



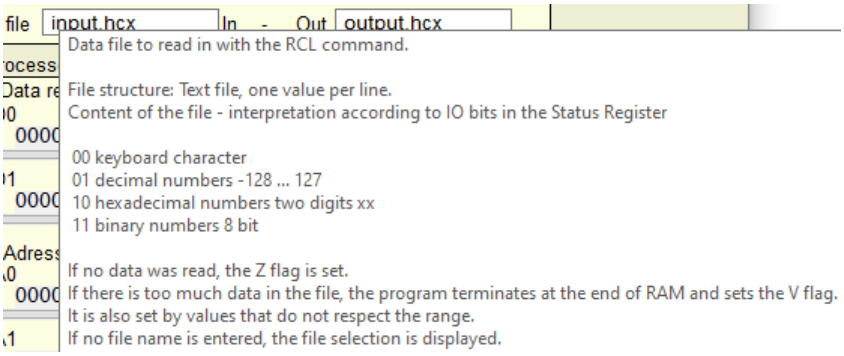


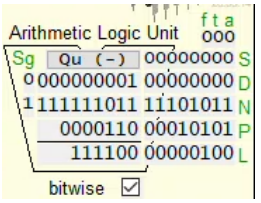
data, store in file



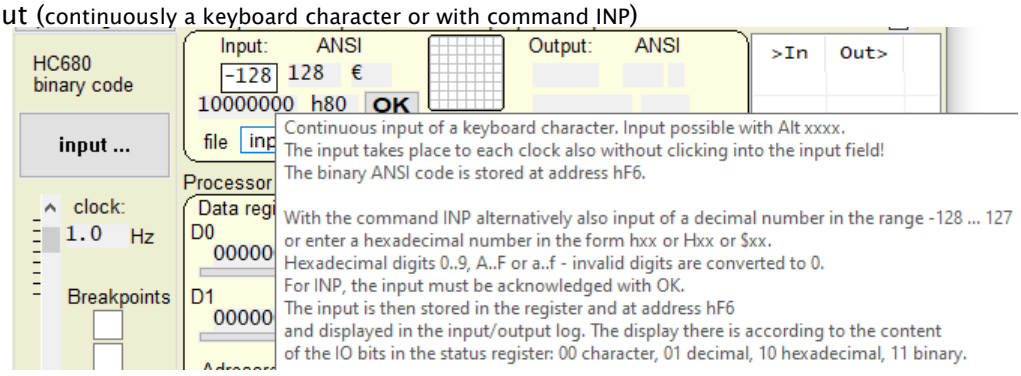


data, read in from file

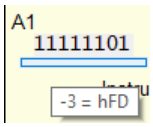




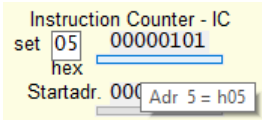
input (continuously a keyboard character or with command INP)

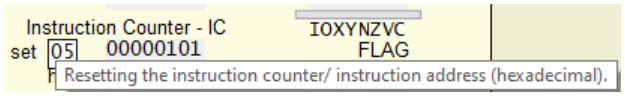


register content (binary, plus decimal, hexadecimal in tooltip)

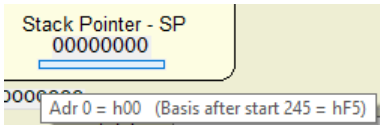


Instruction Counter (at single step –also at breakpoint– manually resettable)

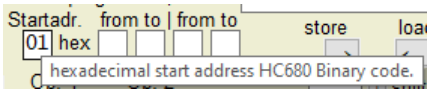




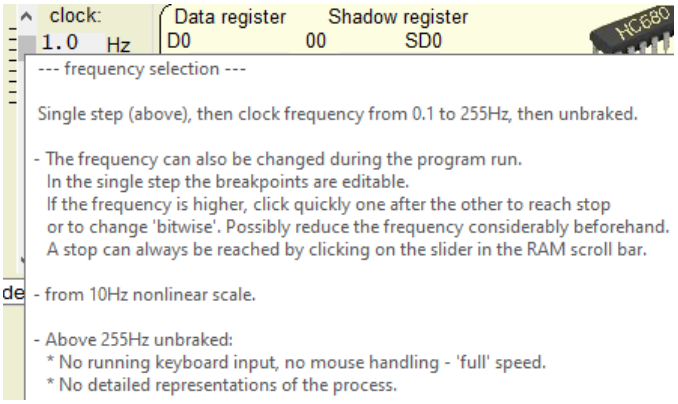
base address Stack Pointer



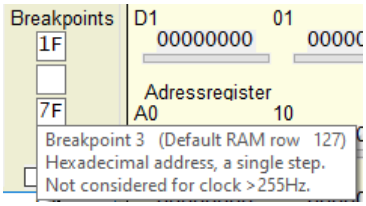
Start address



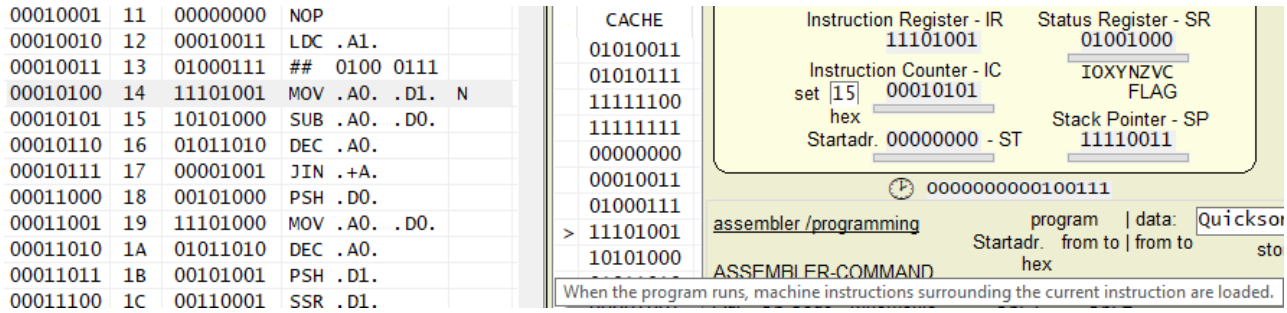
takt



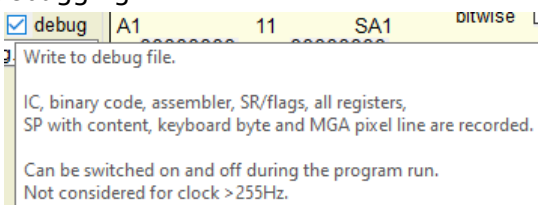
Breakpoints

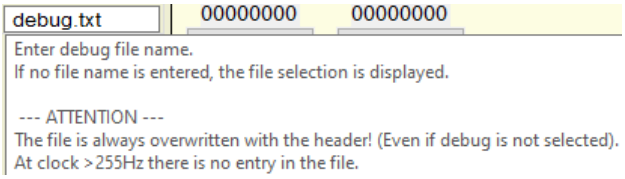


cache



debugging



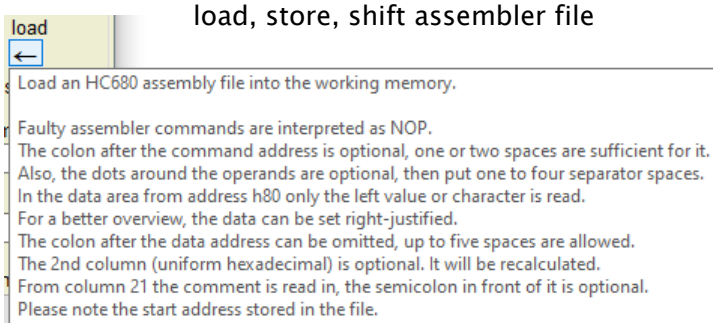


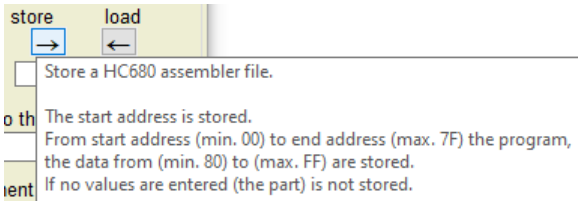
example in assembler "sum exceeded until 50" – stored as a text file

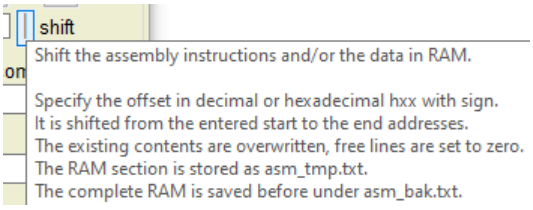
```
HC680 assembler     ; sum exceeded until 50
00: ST              ; Startadr.
Adr Mnm _Op_ _Op_   ; - comment -
00: MOV .SR. .01.   ;output decimal
01: LOD .D1.        ;load target sum from h80
02: INC .A1.        ;incrementing the address index register A1 to load the next constant
03: LOD .A1.        ;load jump difference for IC relative jump in A1
04: PSH .D0.        ;put initial value 0 on the stack for cumulating
05: INP .A0.        ;-INPUT- data into address register - is possible!
06: POP .D0.        ;get cumulated sum from stack
07: ADD .D0. .A0.   ;sum with input value
08: PSH .D0.        ;put sum back on stack
09: CMP             ;compare sum with target sum (D0-D1 without result)
0A: JIN .+A.        ;IC-relative jump to input, if target sum is not yet reached
0B: OUT .D0.        ;-OUTPUT-
0C: STP             ;done!
80: 50   h32        ;sum up to 50
81: -5   hFB        ;IC-relative destination address (five commands back)
```

When creating with a text editor, the colon, the dots before and after the operands and the semicolon can be omitted.
After Mnm and between the operands up to 4 spaces are allowed, before Mnm two.
Before the data up to five blanks are allowed for separation, thus right-justified notation is possible.

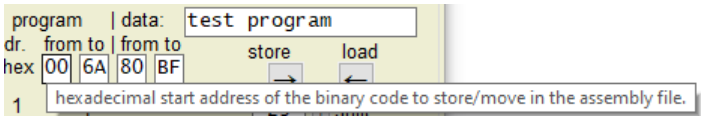
load, store, shift assembler file

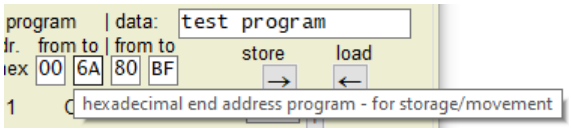


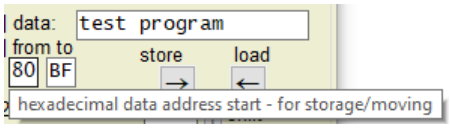


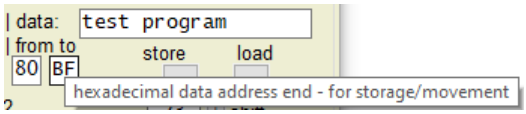


address ranges of the assembler file



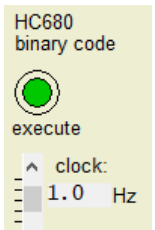




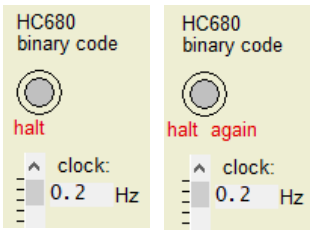


execute machine program / binary code

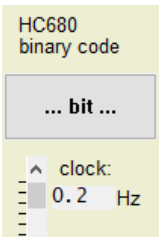
click –



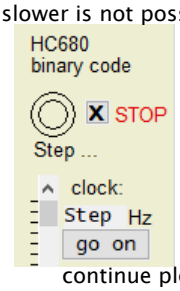
Intermediate stops possible



bitwise in the ALU

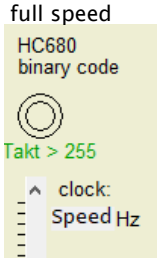


slower is not possible

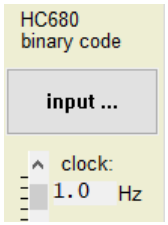


continue please

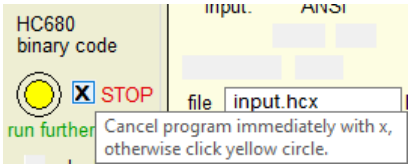
full speed



at INPut



break off?



done!