

BASIC/DEBUG

BASIC/DEBUG ist eine Tiny-BASIC-Implementierung für den Z8. Entstanden ist sie vermutlich 1979/1980.

In nur 2 KByte ROM des [Z8671](#) steckt ein vollwertiges BASIC mit folgenden Eigenschaften:

- Editor
- 15 BASIC-Kommandos
- 16-Bit Ganzzahlarithmetik
- Punkt-vor-Strich-Rechnung
- Hexadezimalzahlen
- Direktzugriff auf Register und Speicher

Das BASIC ist im [Zilog-Handbuch](#)

[z8671_basic_debug.pdf](#)

ausführlich beschrieben.

Das Basic orientiert sich sehr stark an das TINY-BASIC von Tom Pittman (<http://www.ittybittycomputers.com/ittybitty/tinybasic/>). Literatur und Programme zu den dort beschriebenen BASICs für 1802, 6502, 6800-Controller passen auch zum BASIC/DEBUG des Z8671.

11/2021: Tom Pittman hat mir bestätigt, dass er dieses BASIC Ende der 70er im Auftrag von Zilog programmiert hat.

Downloads

- [z8671-basic_debug_src.zip](#)

meine reassemblierten und kommentierten Quellen:

- Z8 assembler source
- IL source + IL Makros
- IL Decompiler (perl)

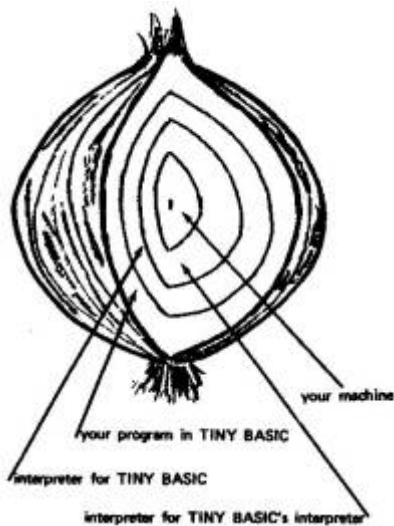
- [tb_pittman.zip](#)

Quellen der Tiny-BASIC-Versionen von T.Pittman (für RCA1802, 6502, 6800) sowie die aufbereiteten IL zwecks Vergleich

zur Anpassung an andere Systeme und andere Adressen sowie Erweiterungen s. [BASIC/DEBUG für ES4.0](#)

Aufbau

2021 habe ich den ROM vollständig und umfassend reassembliert, analysiert und dokumentiert:



BASIC/DEBUG ist in einer eigenen Codesprache geschrieben: TBIL (Tiny Basic Interpreter Language) [Tiny_BASIC#Implementation_in_a_virtual_machine](#).

Das BASIC selbst ist nur 432 Byte groß. In Z8-Code ist der Interpreter für diese Codesprache enthalten, sowie Initialisierung und In/Out-Code. Diese Art der Implementierung ist extrem speichersparend. Nur so konnten die umfangreichen Fähigkeiten des BASIC/DEBUG in nur 2048 Byte untergebracht werden.

Das Zwiebelmodell (s. Bild) beschreibt sehr schön den Ablauf: Das BASIC-Anwenderprogramm wird vom BASIC-Interpreter (in TBIL) interpretiert. Dieser wiederum wird vom TBIL-Interpreter (in Z8-Code) interpretiert, und das ganze läuft auf einem Z8-Prozessor und kommuniziert mit der Außenwelt.

BASIC/DEBUG ist vielfältig konfigurierbar und bietet u.a. folgende Optionen

- beliebige RAM-Unterstützung oder reine ROM-Version
- mehrere BASIC-Programme im Speicher
- Autostart eines Programms nach RESET
- Editierzeichen für DEL, Zeilenende-Zeichenfolge frei vergebbar
- eigene I/O-Treiber anstelle Nutzung der Z8-SIO

Literatur

- [z8671_basic_debug.pdf](#)
von Zilog. (der Hochpfeil in der Doku ist das Zeichen „^“)
- Buch „Mikroprozessoren in der Meß- und Regeltechnik. Funktion - Aufbau und Programmierung“ von Gerhard Ledig, Franzis-Verlag 1988

weitere Literatur zu Tom Pittman's Tiny-BASIC:

orig. Scans

- <http://retro.hansotten.nl/uploads/files/tiny%20basic%20manual.pdf>

- <http://retro.hansotten.nl/uploads/files/tiny%20basic%20exp%20kit.pdf>
- <http://retro.hansotten.nl/uploads/files/tiny%20basic%20getting%20the%20most%20of.pdf>

HTML:

- TINY BASIC User Manual
<http://www.ittybittycomputers.com/ittybitty/tinybasic/TBuserMan.htm>

Beispielprogramme:

- TBprogs.zip, Advent.txt, Euphoria.txt, LifeTB.txt
<http://www.ittybittycomputers.com/ittybitty/tinybasic/>
- The First Book of Tiny BASIC Programs
https://www.retrotechnology.com/memship/Son_of_TFBOTBAS.HTM
- Tiny Basic Experimenters Kit
<http://www.ittybittycomputers.com/ittybitty/tinybasic/TBEK.txt>
- originale Scans und Beispiele und weitere Dokumente
<http://retro.hansotten.nl/6502-sbc/kim-1-manuals-and-software/kim-1-software/>

Beispiel

Das folgende Programm zeigt eine Autostart-BASIC-Programm und eigene I/O-Treiber:

Damit Autostart funktioniert, muss das Programm auf Adresse 1020h beginnen. Die Zeilennummer muss < 256 sein (d.h. das erste Byte == 0).

Im Zeile 1 wird Register %FB (Interrupt mask register) gesetzt. Dadurch werden die externen I/O-Treiber aktiviert. Nachfolgende PRINT-Befehle geben dann über diese Treiber aus. Als Beispiel wurde zu Testzwecken ein einfacher Ringpuffer implementiert.

External I/O drivers must conform to the following requirements: They must pass a single ASCII input or output character in R19 with the register pointer set to R16-31. Registers R4-R15 and R22-32 must be preserved. Location %1012 must contain a jump to the character input driver, and %1015 a jump to the character output driver. The input routine must do any necessary echoing, and the parity bit (bit 7) of all input characters must be set to 0. Drivers return to Basic/Debug with a RETURN instruction (hex AF)

```
cpu    z8601
org 1000h
; interrupt routines
jp 0    ; IRQ 0
jp 0    ; IRQ 1
jp 0    ; IRQ 2
jp 0    ; IRQ 3
jp 0    ; IRQ 4
jp 0    ; IRQ 5
; character io
org 1012h
jp CI
```

```

    jp    C0
; Basic-Pgm
    org 1020h
    db    0,5,"@251=0:REM EXTERNAL IO",0
    db    0,10,"LET A=%1234:PRINT HEX(A)",0
    db    0,20,"STOP",0
    db    0ffh,0ffh
; input driver
CI: ret
; output driver
; use ram ringbuffer 8000h..9000h for simple output
C0: srp    #60h
    cp    r0, #90h
    jr    ge, co2
    cp    r0, #80h
    jr    ge, co1
co2:  ld    r0,#80h
    ld    r1,#0
co1:  ld    r3,13h
    ldc   @rr0,r3
    incw  rr0
    srp   #10h
    ret

```

eigene TBIL

Nicht im Handbuch erwähnt ist die Möglichkeit, einen eigenen BASIC-Interpreter (in TBIL) anstelle des im ROM enthaltenen zu nutzen: Die Adresse des Interpreters steht im Constant Block in Byte 0 und 1. Wird ein externer Constant Block genutzt, kann ein eigener/erweiterter BASIC-Interpreter angesprochen werden. Es ist sogar möglich, maximal drei eigene TBIL-Codes (19, 1E, 1F) zu definieren und zu verwenden! Diese drei OP-Codes sind original nicht genutzt. Auf den Adressen 1018h, 101Bh, 101Eh (nur 2 Byte) sind Sprünge zu den zugehörigen Implementierungen zu setzen.

```

    org 1018h
; IL Extension
    jp    xxxx    ; IL_19
    jp    xxxx    ; IL_1E
    jr    xxxx    ; IL_1F    101Eh; nur 2 Byte, sonst memory overlapping mit
Autostart-Programm

; Basic-Pgm
    org 1020h
    db    0,1,"^28=%1100",0        ; SET EXTERNAL CONSTANT BLOCK
    db    ... own syntax by own IL ...
    db    0ffh,0ffh

; constant block
    org 1100h    ; must be xxx0-address
    db    hi(il)    ; 0 00650h

```

```

db lo(il)      ; 1
db 20h         ; 2  stack reserve
db 8           ; 3  backspace code
db 1Bh         ; 4  cancel code
db 0Dh         ; 5  line separator sequence terminating with 0FEh
db 0Ah         ; 6
db 0FEh       ; end of constant block

```

```

; own IL program:
il: ...

```

Es ist nicht bekannt, ob das für die Entwicklung gedacht war oder ob es erweiterte IL-Programme gibt.

IL Source

Tom Pittman beschreibt in <http://www.ittybittycomputers.com/ittybitty/tinybasic/TBEK.txt> die IL (interpreter Language). Gegenüber dieser Version gibt es ein paar Unterschiede:

Code	Beschreibung	Bemerkungen
0A	CC, stop if cancel code	gibt es nicht in TBEK
0B x	LB, push literal byte to TOS	in TBEK 09
0C x	LN, push addr (00xxh) to TOS	in TBEK 0A. Z8671: Es wird nur das niederwertige Byte genommen. Das höherwertige Byte wird 00 gesetzt. ¹⁾
0D	DS, duplicate stack top	in TBEK 0B
0E	TS, pop Stack	in TBEK 0C
0F	TS, test sign; skip next IL if TOS is positive	in TBEK reserviert
19	reserviert, extern 1018h	in TBEK subtract
1E	reserviert, extern 101Bh	in TBEK reserviert
1F	reserviert, extern 101Eh	in TBEK list program
25	PH, print hex number	in TBEK reserviert
26	HN, convert hex number	in TBEK reserviert
28	SN, put current line number on stack	in TBEK reserviert
29	FL, follow line; goto next line	in TBEK reserviert

Z8671 TINY BASIC

reconstructed Volker Pohl's 10/2021, Germany, Neustadt i.H.

```

; THE IL CONTROL SECTION

```

```

0650  000: 62          X1      BR STRT          ; ENTRY POINT DIRECT
MODE
0651  001: 7A          X2      BR GOTO          ; ENTRY POINT NEXT
STMT MODE
0652  002: 6F          X3      BR XEC          ; ENTRY POINT RUN MODE
0653  003: 24      BA      STRT      PC ":"          ; PROMPT OUTPUT
0655  005: 27          GL          ; GET LINE

```

```

0656 006: E1          BE X4          ; BRANCH IF NOT EMPTY
0657 007: 5B          BR STRT          ; TRY AGAIN IF NULL LINE
0658 008: D3          X4          BN GOTO          ; TEST FOR LINE NUMBER
0659 009: 0F          TS              ; SKIP NEXT IF POSITIVE
065a 00A: 60          BR *            ; ERROR STOP
065b 00B: 10          SB              ; SAVE BASIC POINTER
065c 00C: 0D          DS              ; DUPLICATE TOP OF STACK
065d 00D: 16          GO              ; GOTO LINE
065e 00E: 28          SN              ; PUT FOUND LINE NUMBER ON
STACK
065f 00F: 0E          SP              ; STACK POP
0660 010: 2A          IL              ; INSERT LINE INTO PROGRAM
0661 011: 51          BR STRT          ; GO GET NEXT
0662 012: 10          XEC          SB          ; SAVE POINTERS FOR
RUN WITH
0663 013: 11          RB              ; CONCATENATED INPUT
0664 014: 0C          00          LN 00          ; PUT 0000 ON STACK
0666 016: 16          GO              ; GOTO FIRST LINE
0667 017: 28          SN              ; PUT FOUND LINE NUMBER ON
STACK
0668 018: 0F          TS              ; SKIP NEXT IF POSITIVE
0669 019: 60          BR *            ; ERROR STOP
066a 01A: 0E          SP              ; STACK POP
066b 01B: 2C          X5          XQ          ; EXECUTE PROGRAM

; STATEMENT EXECUTOR

066c 01C: 8F          47 CF          GOTO          BC IF "GO"          ;
066f 01F: 9B          54 CF          BC GOSB "T0"          ; GOTO
0672 022: 35          1C          S0          JS EXPR          ; GET LINE NUMBER
0674 024: E0          S1          BE *            ; ERR IF NOT END OF
LINE
0675 025: 10          SB              ; (DO THIS FOR STARTING)
0676 026: 11          RB              ;
0677 027: 0F          TS              ; SKIP NEXT IF POSITIVE
0678 028: 60          BR *            ; ERROR STOP
0679 029: 16          GO              ; GO THERE
067a 02A: 50          BR X5          ;
067b 02B: 60          BR *            ;
067c 02C: 9D          49 C6          IF          BC LET "IF"          ;
067f 02F: 34          E8          JS REL          ;
0681 031: 84          54 48 45 CE          BC S2 "THEN"          ; OPTIONAL
NOISEWORD
0686 036: 1C          S2          CP              ; COMPARE SKIPS NEXT IF
TRUE
0687 037: 1D          NX              ; FALSE.
0688 038: C1          BN S4          ;
0689 039: 4A          S3          BR S1          ;
068a 03A: 41          S4          BR GOTO          ;
068b 03B: 89          53 55 C2          GOSB          BC S5 "SUB"          ; GOSUB
068f 03F: 34          FF          JS EXPR          ; GET DESTINATION.

```

```

0691 041: E0 BE * ; ERR IF NOT END OF LINE
0692 042: 28 SN ; PUT LINE NUMBER ON
STACK
0693 043: 14 14 ;
0694 044: 54 BR S3 ;
0695 045: 80 C0 S5 BC * "@" ; GO@
0697 047: 34 F0 JS ARG3 ; GET 3 ARGUMENTS
0699 049: 7A BR S10 ; EXECUTE MACHINE CODE
069a 04A: 9B 4C 45 D4 LET BC S11 "LET" ;
069e 04E: A6 LET1 BV S8 ; VARIABLE NAME?
069f 04F: 80 BD S7 BC * "=" ;
06a1 051: 34 ED JS EXPR ; GO GET EXPRESSION
06a3 053: 13 SV ; STORE RESULT
06a4 054: 71 BR S11 ;
06a5 055: 84 DE S8 BC S9 "^" ; ADDR REFERENCE?
06a7 057: 35 25 JS FACT ; GET ADDR
06a9 059: 55 BR S7 ;
06aa 05A: 9A C0 S9 BC P0 "@" ; BYTE/REGISTER
REFERENCE?
06ac 05C: 0C 81 LN 81 ; 0081 / =LO(LN81),
HI(LN81) MUST BE 0
06ae 05E: 35 1E JS FACT ; GET DEST
06b0 060: 80 BD BC * "=" ;
06b2 062: 34 DC JS EXPR ; GET SRC
06b4 064: 2E S10 US ; USER SUBROUTINE.
STACK: 0081 (ADDR), DEST, SRC
06b5 065: 0E SP ; STACK POP
06b6 066: E1 S11 BE S12 ; IF STATEMENT END,
06b7 067: 1D NX ; DONE
06b8 068: 83 BA S12 BC S14 ":" ; NEXT STATEMENT
06ba 06A: 3B B0 S13 J GOTO ; GET NEXT
STATEMENT
06bc 06C: 82 A2 S14 BC PRNT "" ; PRINT STRING
(WITHOUT KEYWORD)
06be 06E: 7B BR P8 ;
06bf 06F: 9D 50 52 49 4E D4 PRNT BC P9 "PRINT" ; PRINT
06c5 075: 83 BA P0 BC P1 ":" ; STATEMENT END?
06c7 077: 23 NL ; NEW LINE
06c8 078: 51 BR S13 ; NEXT STATEMENT
06c9 079: EE P1 BE P7 ; END OF LINE?
06ca 07A: 68 BR P5 ; YES
06cb 07B: 83 BB P2 BC P4 ";" ; NO CR
06cd 07D: E7 P3 BE P6 ; END OF LINE?
06ce 07E: 1D NX ; YES, NEXT BASIC STATEMENT
06cf 07F: 83 AC P4 BC P5 ", " ; COMMA SPACING?
06d1 081: 22 PT ; PRINT TAB
06d2 082: 5A BR P3 ;
06d3 083: 23 P5 NL ; NEW LINE
06d4 084: 41 BR S11 ;
06d5 085: 82 BA P6 BC P7 ":" ; PRINT STMT END?
06d7 087: 42 BR S13 ;

```

```

06d8 088: 85 A2 P7 BC P10 "" ; STRING?
06da 08A: 21 P8 PQ ; PRINT QUOTE MARKS
STRING
06db 08B: 60 BR * ; ERROR STOP
06dc 08C: 4E BR P2 ;
06dd 08D: 74 P9 BR RETN ; FORWARD BRANCH
06de 08E: 88 48 45 58 A8 P10 BC P12 "HEX(" ; HEX?
06e3 093: 35 16 JS F12 ; GET EXPR AND )
06e5 095: 25 PH ; PRINT HEX NUMBER
06e6 096: 44 P11 BR P2 ;
06e7 097: 34 A7 P12 JS EXPR ; PRINT EXPR
06e9 099: 17 NE ; NEGATE
06ea 09A: 0F TS ; SKIP NEXT IF POSITIVE
06eb 09B: 63 BR P13 ; NO, WAS POSITIVE NUMBER
06ec 09C: 24 AD PC "-" ; PRINT SIGN
06ee 09E: 61 BR P14 ;
06ef 09F: 17 P13 NE ; NEGATE (ORIG VALUE)
06f0 0A0: 20 P14 PN ; PRINT NUMBER
06f1 0A1: 54 BR P11 ;
06f2 0A2: 8C 52 45 D4 RETN BC STOP "RET" ; RETURN
06f6 0A6: 83 55 52 CE BC S15 "URN" ; OPTIONAL
06fa 0AA: E0 S15 BE * ; ERR IF NOT END OF
LINE
06fb 0AB: 15 RS ; RESTORE SAVED LINE
06fc 0AC: 16 GO ;
06fd 0AD: 1D NX ; NEXT BASIC STATEMENT
06fe 0AE: 60 BR * ; ERROR STOP
06ff 0AF: 86 53 54 4F D0 STOP BC RUN "STOP" ;
0704 0B4: E0 BE S16 ; END OF LINE?
0705 0B5: 2D S16 WS ; YES, STOP
0706 0B6: 85 52 55 CE RUN BC NEW "RUN" ;
070a 0BA: 3B 56 J XEC ; RUN PROGRAM
070c 0BC: 85 4E 45 D7 NEW BC REM "NEW" ;
0710 0C0: E0 BE * ; ERR IF END OF LINE?
0711 0C1: 2B MT ; MARK BASIC PGM EMPTY
0712 0C2: 84 52 45 CD REM BC INPT "REM" ;
0716 0C6: 1D NX ; NEXT BASIC STATEMENT
0717 0C7: 9D 49 CE INPT BC LIST "IN" ;
071a 0CA: 86 50 55 D4 BC S17 "PUT" ; INPUT
071e 0CE: A0 BV * ; GET VARIABLE
071f 0CF: 10 SB ; SWAP POINTERS
0720 0D0: 63 BR I3 ;
0721 0D1: A0 S17 BV * ; IN COMMAND, FIRST
VAR?
0722 0D2: 10 SB ; SWAP POINTERS
0723 0D3: E6 BE I4 ; END OF LINE?
0724 0D4: 24 3F A0 I3 PC "? " ; YES, TYPE
PROMPT
0727 0D7: 27 GL ; READ INPUT LINE
0728 0D8: E1 BE I4 ; DID ANYTHING COME?
0729 0D9: 5A BR I3 ; NO, TRY AGAIN
    
```



```

072a  0DA: 81  AC          I4      BC I5 ", "      ; NEXT PARAMETER
(IN CMD)
072c  0DC: 34  62          I5      JS EXPR          ; READ A NUMBER
072e  0DE: 13          SV              ; STORE INTO VARIABLE
072f  0DF: 11          RB              ; SWAP BACK
0730  0E0: 82  AC          BC I6 ", "      ; ANOTHER?
0732  0E2: 4E          BR S17          ; YES IF COMMA
0733  0E3: 3B  81          I6      J  S11          ; OTHERWISE QUIT

0735  0E5: 8C  4C 49 53 D4  LIST      BC L1 "LIST"      ; LIST?
073a  0EA: 28          SN              ; PUT LINE NUMBER ON STACK
073b  0EB: E8          BE L2          ; END OF LINE?
073c  0EC: 0C  FF          LN FF          ; THEN PUT 00FF ON STACK
073e  0EE: 17          NE              ; NEGATE -> FF01
073f  0EF: 0C  7F          LN 7F          ; PUT 007F ON STACK
0741  0F1: 67          BR L3          ; FF01007F -> 7F0100FF ->
7FFF0001
0742  0F2: 3B  5A          L1      J  LET1          ; ELSE LET STMT
(WITHOUT KEYWORD)
0744  0F4: 34  46          L2      JS ARG2          ; GET TWO NUMBERS
0746  0F6: E0          BE *          ; ERR IF NOT END OF LINE
0747  0F7: 03          SX 3          ; EXCHANGE THE TOP TWO
aAbB > BAba
0748  0F8: 01          SX 1          ; NUMBERS ON THE STACK
> BAab
0749  0F9: 03          L3      SX 3          ;
> bAaB
074a  0FA: 02          SX 2          ;
> bBaA
074b  0FB: 16          GO              ; GO FIRST LINE IN AREA
074c  0FC: 28          SN              ; PUT LINE NUMBER ON STACK
074d  0FD: 0E          SP              ;
074e  0FE: 23          L4      NL          ; OUTPUT NEW LINE
074f  0FF: 0A          0A              ; GOT CANCEL CODE?
0750  100: 74          BR L8          ; THEN CANCEL LIST
0751  101: 0D          DS              ; DUP STACK
0752  102: 0B  03          LB 03          ; 03 (>=)
0754  104: 28          SN              ; PUT LINE NUMBER ON STACK
0755  105: 1C          CP              ; COMPARE. SKIP NEXT IF TRUE
0756  106: 6E          BR L8          ; ELSE LIST END
0757  107: 28          SN              ; PUT LINE NUMBER ON STACK
0758  108: 0F          TS              ; SKIP NEXT IF POSITIVE
0759  109: 6A          BR L7          ; NEGATIVE LINE NUMBER FFFF
-> END
075a  10A: 20          PN              ; OUTPUT LINE NO
075b  10B: 24  A0          PC " "          ; OUTPUT SPACE
075d  10D: 21          L5      PQ          ; OUTPUT LINE TO 00 OR
"
075e  10E: 63          BR L6          ; (PQ SKIPPES BRANCH IF ")
075f  10F: 24  A2          PC ""          ; OUTPUT "
0761  111: 5B          BR L5          ; AND CONTINUE

```

```

0762    112: 29                L6      FL                ; SET NEXT LINE
0763    113: 4A                BR L4      ; LOOP
0764    114: 0E                L7      SP                ; STACK POP
0765    115: 0E                L8      SP                ; RESTORE ORIG LINE
NUMBER
0766    116: 16                GO                ; GO ORIG LINE NUMBER
0767    117: 1D                NX                ;
0768    118: 1D                NX                ; ?? WHY ??

                ; SUBROUTINES

0769    119: 34    25        REL      JS EXPR          ; LEFT EXPRESSION
076b    11B: 8E    BC        BC R2 "<"          ; CONVERT RELATION
OPERATORS
076d    11D: 84    BD        BC R0 "="          ; TO CODE BYTE ON STACK
076f    11F: 0B    06        LB 06          ; <= 6
0771    121: 76                BR R5          ;
0772    122: 84    BE        R0      BC R1 ">"          ;
0774    124: 0B    05        LB 05          ; <> 5
0776    126: 71                BR R5          ;
0777    127: 0B    04        R1      LB 04          ; < 4
0779    129: 6E                BR R5          ;
077a    12A: 89    BE        R2      BC R4 ">"          ;
077c    12C: 84    BD        BC R3 "="          ;
077e    12E: 0B    03        LB 03          ; >= 3
0780    130: 67                BR R5          ;
0781    131: 0B    01        R3      LB 01          ; > 1
0783    133: 64                BR R5          ;
0784    134: 80    BD        R4      BC * "="          ;
0786    136: 0B    02        LB 02          ; = 2
0788    138: 67                R5      BR EXPR        ; RIGHT EXPRESSION

0789    139: 34    01        ARG3     JS ARG2          ; GET 3 ARGUMENTS
078b    13B: 62                BR A1          ; 3RD OPT ARGUMENT
078c    13C: 34    02        ARG2     JS EXPR          ; GET 2 ARGUMENTS
078e    13E: 97    AC        A1      BC E4 ", "          ; OPTIONAL
ARGUMENT
0790    140: 85    AD        EXPR     BC E0 "- "          ; TRY FOR UNARY
MINUS
0792    142: 34    14                JS TERM        ; GET VALUE
0794    144: 17                NE            ; NEGATE IT
0795    145: 64                BR E1          ; LOOK FOR MORE
0796    146: 81    AB        E0      BC E4 "+"          ; IGNORE UNARY
PLUS
0798    148: 34    0E        E4      JS TERM        ; GET VALUE
079a    14A: 85    AB        E1      BC E2 "+"          ; TERMS SEPARATED
BY PLUS
079c    14C: 34    0A                JS TERM        ; GET TERM
079e    14E: 18                E5      AD            ; SUM TERM
079f    14F: 5A                BR E1          ;

```

```

07a0 150: 86 AD E2 BC E3 "-" ; TERMS SEPARATED
BY MINUS
07a2 152: 34 04 JS TERM ; GET TERM
07a4 154: 17 NE ; NEGATE
07a5 155: 58 BR E5 ; AND ADD
07a6 156: 0D E4 DS ; DUPLICATE NUMBER ON
STACK
07a7 157: 2F E3 RT ;

07a8 158: 34 24 TERM JS FACT ; LEFT FACTOR
07aa 15A: 85 AA T0 BC T1 "*" ; FACTORS
SEPARATED BY TIMES
07ac 15C: 34 20 JS FACT ; RIGHT FACTOR
07ae 15E: 1A MP ; MULTIPLY
07af 15F: 5A BR T0 ; NEXT FACTOR
07b0 160: 92 AF T1 BC T6 "/" ; FACTORS
SEPARATED BY DIVIDE
07b2 162: 17 NE ; NEGATE DIVIDEND
07b3 163: 0F TS ; SKIP NEXT IF POSITIVE
07b4 164: 67 BR T3 ;
07b5 165: 34 17 JS FACT ; GET DIVISOR
07b7 167: 0F TS ; SKIP NEXT IF POSITIVE (-
/+)
07b8 168: 71 BR T8 ;
07b9 169: 1B T2 DV ; DIVIDE (-/+ , +/-)
07ba 16A: 17 NE ; NEGATE RESULT
07bb 16B: 4E BR T0 ; NEXT FACTOR
07bc 16C: 17 T3 NE ; NEGATE DIVIDEND BACK
07bd 16D: 34 0F T4 JS FACT ; GET DIVISOR
07bf 16F: 0F TS ; SKIP NEXT IF POSITIVE
(+/+)
07c0 170: 65 BR T7 ;
07c1 171: 1B T5 DV ; DIVIDE (+/+ , -/-)
07c2 172: 47 BR T0 ; NEXT FACTOR
07c3 173: 94 DC T6 BC F3 "\" ; UNSIGNED
DIVISION?
07c5 175: 57 BR T4 ;
07c6 176: 17 T7 NE ; NEGATE
07c7 177: 0F TS ; SKIP NEXT IF POSITIVE
07c8 178: 63 BR T9 ;
07c9 179: 4F BR T2 ; (+/-)
07ca 17A: 17 T8 NE ; NEGATE BACK
07cb 17B: 0F TS ; SKIP NEXT IF POSITIVE
07cc 17C: 60 T9 BR * ; ERROR STOP
07cd 17D: 53 BR T5 ; (-/-)

07ce 17E: 8A 55 53 52 A8 FACT BC F4 "USR(" ; USR
FUNCTION?
07d3 183: 33 B4 JS ARG3 ; GET ARGUMENTS
07d5 185: 80 A9 F1 BC * ")" ;
07d7 187: 2E F2 US ; USER SUBROUTINE.

```

```

STACK USF-ADDR, 2 PARAM
07d8 188: 2F          F3      RT          ; RETURN
07d9 189: 89      41 4E 44 A8    F4      BC F5 "AND("      ; AND
FUNCTION?
07de 18E: 0C      3C          LN 3C          ; 003C ADDRESS USF AND
07e0 190: 33      AA          JS ARG2        ; GET ARGUMENTS
07e2 192: 52          BR F1          ;
07e3 193: 87      C0          F5      BC F6 "@"      ; BYTE REFERENCE?
07e5 195: 0C      47          LN 47          ; 0047 ADDRESS USF FETCH
07e7 197: 33      E5          JS FACT        ; GET BYTE
07e9 199: 0D          DS            ; DUPLICATE NUMBER ON STACK
07ea 19A: 4C          BR F2          ;
07eb 19B: 84      DE          F6      BC F7 "^"      ; ADDR REFERENCE?
07ed 19D: 33      DF          JS FACT        ; GET ADDR
07ef 19F: 61          BR F8          ;
07f0 1A0: A2          F7      BV F9          ; VARIABLE?
07f1 1A1: 12          F8      FV            ; FETCH VALUE
07f2 1A2: 2F          RT            ; RETURN
07f3 1A3: 83      A5          F9      BC F10 "%"      ; HEX NUMBER?
07f5 1A5: 26          HN            ; GET HEX NUMBER
07f6 1A6: 2F          RT            ; RETURN
07f7 1A7: C1          F10     BN F11          ; NUMBER?
07f8 1A8: 2F          RT            ; GOT IT.
07f9 1A9: 80      A8          F11     BC * "("      ; OTHERWISE MUST
BE (EXPR)
07fb 1AB: 33      93          F12     JS EXPR          ;
07fd 1AD: 80      A9          BC * ")"      ;
07ff 1AF: 2F          RT            ; RETURN
    
```

Aufgrund der Beschränkungen bei LN (nur Werte 00xxh möglich) muss bei LIST ohne Parameter ein wenig gezaubert werden: Die Werte auf dem Stack werden durch Manipulation in kleinste und größtmögliche Zeilennummer gewandelt. Das entspricht dann dem Aufruf „LIST 1,32767“.

Die Division ist nur als vorzeichenlose 16-Bit-Division implementiert. Deshalb muss in der IL die Fallunterscheidung bzgl. Vorzeichen und Division durch 0 erfolgen.

Um Bytes zu sparen, wurden Unterprogramme verlegt und Codes zusammengeführt. Das erzeugt leider auch schwerer lesbaren Spaghetti-Code, aber nur so passt der gesamte Basic-Interpreter in 2kByte.

¹⁾
Das spart ein paar Bytes, aber schränkt den Code bzgl. Adressen ein. USR-Funktionen müssen im Bereich 00xx liegen

From:
<https://hc-ddr.hucki.net/wiki/> - **Homecomputer DDR**

Permanent link:
https://hc-ddr.hucki.net/wiki/doku.php/elektronik/z8671/basic_debug

Last update: **2022/08/04 06:37**

