

# Programmieren

Für den Z9001 stehen diverse Programmiersprachen bereit:

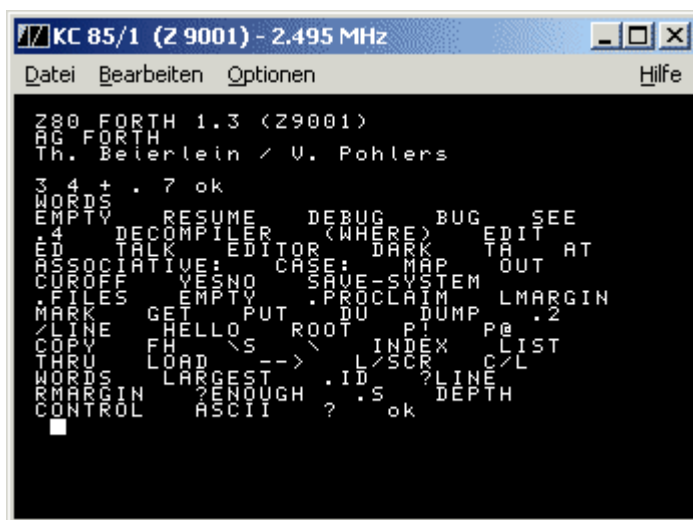
- BASIC
- FORTH
- PASCAL
- Assembler

## BASIC

Der Einsteiger sollte mit BASIC beginnen. Mit einem BASIC-ROM-Modul oder mit dem eingebauten BASIC des KC 87 2.x kann man gleich nach dem Einschalten mit der Arbeit beginnen. Das BASIC wird ausführlich im **Programmierhandbuch** und im **Anhang zum Programmierhandbuch** beschrieben (s. [Handbücher](#) und Downloads der Handbücher bei <http://www.sax.de/~zander/>).

## FORTH

Es gibt verschiedene FORTH-Versionen. Bekannt ist das f.i.g.-FORTH **FORTH** und das [FG FORTH83 d. DDR F83](#). Beide sind im Mega-Flash-Modul enthalten. F83 ist deutlich umfangreicher und schneller.



Beim [KC-Club Treffen 2012](#) gab es eine kleine Einführung ins FG-FORTH FORTH83 [z9001\\_f83.pdf](#).

## PASCAL

Für den KC gibt es das KC-PASCAL, eine Variante des bekannten Hisoft-PASCAL, sowie als 32K-Modul ein Turbo-Pascal-ähnliches Pascal von der TH Leipzig (s. [weitere Module](#), das Handbuch findet man bei <http://www.sax.de/~zander/>).

# Assembler

Zur vollständigen und systemnahen Programmierung eignet sich Assembler. robotron bietet mit **EDAS** und **IDAS** gleich zwei Assembler an. Mit **ZSID** und **R80** stehen außerdem Debugger und Reassembler im Mega-Flash-Modul zur Verfügung.

Allerdings ist das Programmieren in Assembler um einiges komplexer und schwerer als in den „höheren“ Programmiersprachen.

Als Basis sollten unbedingt die Beschreibung des Betriebssystems incl. Betriebssystemlisting **robotron Betriebssystem KC 85/1 (Z9001)** studiert werden. Das Handbuch findet man bei <http://www.sax.de/~zander/>.

## Systemfunktionen

Zur systemunabhängigen Programmierung werden vom Betriebssystem 33 Systemrufe bereitgestellt. Diese werden analog CP/M über CALL 0005 aufgerufen. Die Auswahl des gewünschten Systemrufes erfolgt über das C-Register, dessen Inhalt den Systemruf adressiert. Verschiedene Systemrufe erwarten Eingabeparameter bzw. liefern Parameter zurück.

Eingabeparameter:

- Bytewerte im E -Register
- Wortwerte im DE-Register

Ausgabeparameter:

- Bytewerte im A -Register
- Wortwerte im BC-Register

Wichtige Systemrufe:

Rufnr.	Name	Funktion
01	CONSI	Eingabe eines Zeichens von CONST
02	CONSO	Ausgabe eines Zeichens zu CONST
09	PRNST	Ausgabe einer Zeichenkette zu CONST
10	RCONB	Eingabe einer Zeichenkette von CONST
11	CSTS	Abfrage Status CONST
17	GETCU	Abfrage logische und pyhsische Cursoradresse
18	SETCU	Setzen logische Cursoradresse

## Der OS-Rahmen

Damit eigene Programme vom OS aus gestartet werden können, wird ein spezieller Code benötigt, der sogenannte OS-Rahmen. Damit erscheinen Programme als transiente Kommandos im OS und können über den Kommandoname aufgerufen werden. Außerdem können Parameter übergeben werden (s. z.B. Code von OS-SAVE).

Ein OS-Rahmen muss auf einer integralen 100H-Grenze (300h ... 0BF00h) beginnen. Es können beliebig viele Kommandos in einem OS-Rahmen angegeben werden.

```

ORG    xx00h

JP     AUSF          ;Sprung zur Kommandoausführung1
DB     'NAME'        ',0' ;Kommandoname1 (im OS-Mode einzugeben)
                        ;8 Zeichen, ggf. mit Leerzeichen auffüllen, Null-Byte
JP     AUSF2         ;Sprung zur Kommandoausführung2
DB     'NAME2'       ',0' ;Kommandoname2 (im OS-Mode einzugeben)
...
DB     0             ;Kennzeichen OS-Rahmen Ende

AUSF:  ...

```

## Kommandozeile

Die eingegebene Kommandozeile wird im Puffer CONBU abgelegt, am Ende wird ein Nullbyte angefügt. Mit GVAL werden die einzelnen Parameter nacheinander geholt, d.h. nach INTLN kopiert und in CONBU wird der Parameter durch Leerzeichen ersetzt. Nach Start des Programms ist der Programmname bereits nicht mehr in CONBU lesbar<sup>1)</sup>.

## erneuter Aufruf

Ein Warmstart kann erkannt werden, indem beim Einsprung HL auf den Einsprungswert verglichen wird (AUSF, AUSF2,...). Ist er gleich, wurde das Programm nicht von Kassette geladen (oder vom Megamodul), sondern ist bereits im Speicher abgelegt und wurde von dort gestartet (mittels Routine CPROM).

## Programmende

Bei Programmende wird im Fehlerfall ein gesetztes Carry-Bit und in Register A ein Fehlercode <> 0 erwartet. Das OS gibt dann eine entsprechende Fehlermeldung aus (s. Doku OS, 2.2.3. Fehlerbehandlung, Tabelle der Fehlercodes). Deswegen sollte ein Programm stets mit

```

XOR A ; Cy=0, A=0
RET

```

enden, wenn keine Fehlermeldung erfolgen soll.

## Beispiele

Folgendes Programm gibt den Text „Hallo User!“ auf den Bildschirm aus. Das Programm wird mit dem Kommando TEST gestartet.

```

cpu    z80
org    300h

```

Beispiel:  
*;Löschen Bildschirm in Hintergrundfarbe blau*

```

;Ausgabe einer Kopfzeile in der Farbe rot
; Start im OS mit TEST

    jp    main
    db    "TEST"    ",0"    ; 8 Zeichen; Ende der Zeichenkette
    db    0          ; Ende des Headers

main:    ld    de, text
        ld    c,9
        call  5
        jp    0

;Zeichenkettendefinition
TEXT:    DB 15H          ;Farbsteuercode Hintergrund
        DB 4            ;Farbe BLAU
        DB 0CH          ;Code für CLEAR SCREEN
        DB 14H          ;Farbsteuercode Vordergrund
        DB 1            ;Farbe ROT
        DB "Hallo User!"
        DA 0A0DH        ;CRLF
        DB 0            ;Ende der Zeichenkette

end

```

## Beispiel2: Tastaturabfrage

```

    cpu    z80
    org    300h

;Ausgabe Taste hexadezimal
; Start im OS mit TEST

    jp    main
    db    "TEST"    ",0"    ; 8 Zeichen; Ende der Zeichenkette
    db    0          ; Ende des Headers

main:    ld    c,11        ; CSTS
        call  5
        push  af
        call  out_a
        pop   af

;    jr    main            ; variante A: der Tastcode bleibt erhalten

    or     a
    jr     z,main          ; keine Taste gedrückt
    ld     c,1            ; CONSI
    call   5              ; sonst Taste aus Puffer holen

    jr     main

```

```

; Ausgabe A hexadecimal ASCII 2 Stellen
out_a:    push    af
          and     0F0h
          rlca
          rlca
          rlca
          rlca
          call    out_a1
          pop     af
          and     0Fh
out_a1:   add     a, 30h          ; '0'
          cp      3Ah           ; '9'+1
          jr      c, out_a2
          add     a, 7
; Zeichenausgabe A
out_a2:   ld      e, a
          ld      c, 2          ; CONSO
          call    5
          ret
          end

```

### Beispiel 3: Testprogramm für Parameterübergabe TESTPARA

Es werden Parametertyp, Parameterwert, eventuelle Fehler sowie der Inhalt des Buffers CONBU nach jedem Holen des nächsten Parameters mit GVAL angezeigt.

```

; Testprogramm für GVAL-Funktion (Parameterübergabe an Programme)

cpu      z80

CONBU:    EQU     80H          ; CCP ZEICHENKETTENPUFFER
INTLN:    equ     0100h        ; interner Zeichenkettenpuffer
OCRLF:    EQU     0F2FEH
OUTA:     EQU     0F305H
OSPAC:    EQU     0F310H
GVAL      equ     0F1EAh

org       300h

; -----
; -----
; Kommando-Rahmen
; -----
; -----

jp        para
db        "TESTPARA", 0
db        0

; -----

```

```
-----  
; TESTPARA  
;-----  
-----  
  
para:      ex    af, af'      ;'  
          jr     c, ende      ; keine weiteren Parameter  
  
next_param:  
          call   anz_conbu     ; Anzeige CONBU  
  
; nächsten Parameter holen  
          call   gval  
  
; GVAL  
; Funktion: Löschen internen Puffer (INTLN).  
;           Übernahme Parameter aus CONBU nach INTLN  
;           Test auf Parameterart  
;           Konvertieren Parameter, wenn dieser ein Wert ist  
; Return  
;   Parameter: Z  1 Parameter war Dezimalzahl  
;               0 Parameter war keine Zahl  
;   CY  0 kein Fehler  
;       1 Fehler im Parameter  
;   A   Konvertierte Dezimalzahl, wenn Z = 1 und CY = 0  
;   C   den Parameter begrenzendes Trennzeichen  
;   B   Länge des Parameters  
;   HL  Adresse des nächsten Zeichens in CONBU  
;   CY' 0 weitere Parameter in CONBU (ist in Doku falsch!)  
;       1 keine weiteren Parameter (ist in Doku falsch!)  
;   A'  den Parameter begrenzendes Trennzeichen  
;   INTLN  Länge des Parameters  
;   INTLN+1. . . übernommener Parameter  
;   CONBU übernommener Parameter und Trennzeichen gelöscht mit  
;           Leerzeichen  
  
          jr     z,zahl_parameter  
  
          call   prnst  
          db     "Text ", 0  
          jr     para1  
zahl_parameter:  
          push   af  
          call   prnst  
          db     "Zahl ", 0  
          pop    af  
  
          jr     nc, para1      ; Fehler in Zahl?  
          call   prnst  
          db     "mit Fehler! ",0
```

```

para1:    call    prnst
          db      "Laenge=", 0
          ld      a, (INTLN)          ; Länges des Parameters
          add     a, '0'
          call    OUTA
          call    OSPAC

          ld      a, (INTLN)
          or      A
          jr      z, para2            ; bei Länge 0 nicht anzeigen

          ld      de, INTLN+1
          ld      c, 9
          call    5                    ; Anzeige Text

para2     call    OCRLF
          ex      af, af'              ; '
          jr      c, ende              ; wenn kein Parameter folgt
          jr      next_param

;
;
ende:     call    anz_conbu
          call    prnst
          db      "-- kein weiterer Parameter --"
          db      0dh, 0ah, 0

;
          or      a
          ret

;
;-----
;---
;Ausgabe String, bis 0
;-----
;---
;
prnst:    EX      (SP), HL              ;Adresse hinter CALL
PRS1:     LD      A, (HL)
          INC     HL
          or      A                    ;Ende (A=0=?
          JR      Z, PRS2              ;ja
          CALL    OUTA
          JR      PRS1                 ;nein
PRS2:     EX      (SP), HL              ;neue Returnadresse
          RET

;-----
;---
; Anzeige CONBU
;-----

```

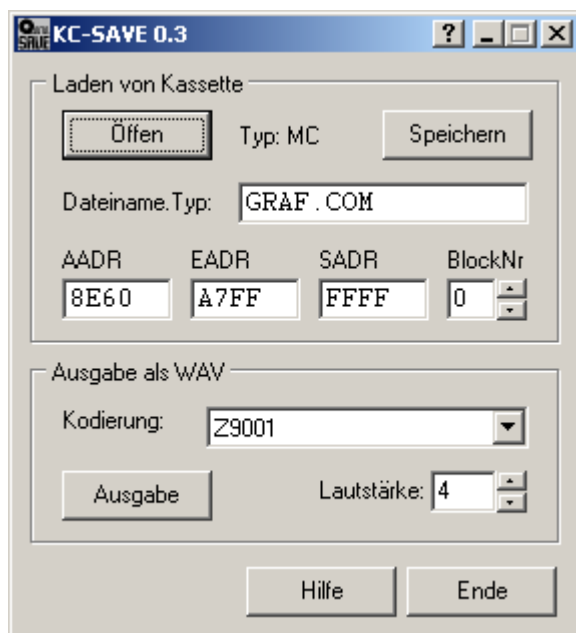
```
---
anz_conbu:  call    prnst
            db      "CONBU >", 0

            ld      de, CONBU+2
anz         ld      c, 9
            call    5
            ld      a, '<'
            call    outa
            call    OCRLF
            ret

;-----
end
```

## Programmerstellung am PC

Bei großen Programmen ist es leichter diese am PC zu schreiben und zu assemblieren. Ich nutze dafür den [arnold-assembler](#). Kleine in Perl geschriebene Hilfstools unterstützen den Prozess und erzeugen z.B. gleich tap-Dateien, die im Emulator geladen werden können oder mit KCSAVE [kcsave.rar](#) als Audiosignal am realen KC geladen werden können.



1)

steht aber noch in INTLN



From:

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

Permanent link:

<https://hc-ddr.hucki.net/wiki/doku.php/z9001/programmieren?rev=1740314483>

Last update: **2025/02/23 12:41**

