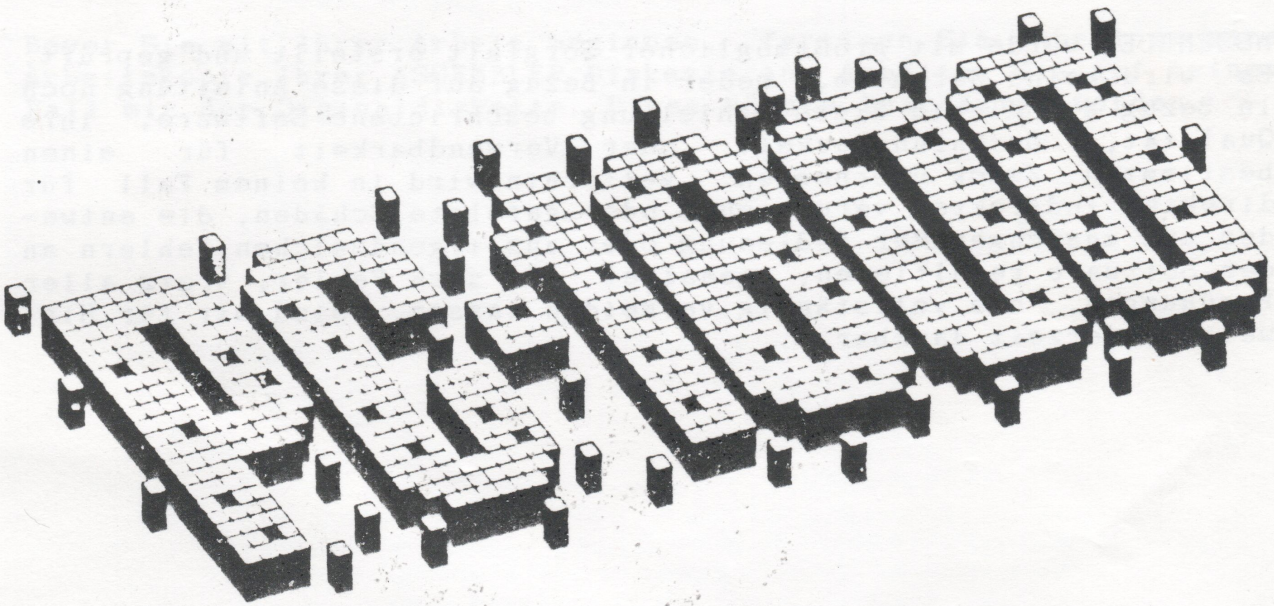
SHARP

ASSEMBLER



Klaus Ditze, Weilerswist

<<< Copyright >>>

<<< Einschränkung der Gewährleistung >>>

ASSEMBLER wurde mit größtmöglicher Sorgfalt erstellt und geprüft. Es wird keine Garantie, weder in Bezug auf diese Anleitung noch in Bezug auf die in dieser Anleitung beschriebene Software, ihre Qualität, Durchführbarkeit oder Verwendbarkeit für einen bestimmten Zweck übernommen. Weiterhin wird in keinem Fall für direkte, indirekte, verursachte oder gefolgte Schäden, die entweder aus unsachgemäßer Bedienung oder aus irgendwelchen Fehlern an der Software resultieren, gehaftet. Da sich Fehler, trotz aller Bemühungen, nie vollständig vermeiden lassen, sind wir für Hinweise jederzeit dankbar.

<<< Funktionsbeschreibung >>>

Mit diesem Assembler können Sie mit dem PC-1600 Quelltexte, die Sie mit einem Textverarbeitungssystem oder in BASIC-Zeilen (mit Apostroph am Zeilenanfang, siehe SAVE* im BASIC-Handbuch) erstellt und gespeichert haben, in Maschinensprache übersetzen.

Für die Erstellung von Maschinenprogrammen auf dem PC-1600 sind Kenntnisse über Z80-Maschinensprache und Systemkenntnisse des PC-1600 unumgänglich. Wir verweisen Sie hier auf die Literaturliste im Anhang.

Fachbegriffe wie MACRO u.ä. sind im Abschnitt <<< QUELLTEXT >>> beschrieben.

Senden Sie unbedingt die beigefügte Kundenkarte an uns ein, damit wir Sie bei Programmänderungen informieren können.
Eine eventuell notwendige Erweiterung oder Fehlerberichtigung dieser Anleitung ist auf der Programmdiskette unter dem Namen 'ASS-NEU' abgespeichert.
Laden Sie dieses Programm im RUN-Modus mit LOAD "X:ASS-NEU",R.
Alles weiter entnehmen Sie dann der Rechneranzeige.

Bevor Sie mit Ihrer Arbeit beginnen, fertigen Sie unbedingt eine Arbeitskopie Ihrer ASSEMBLER Diskette an. Arbeiten Sie auf keinem Fall mit der Orginaldiskette. Diese könnte beschädigt werden.

<<< VOREINSTELLUNGEN >>>

Auf der Programmdiskette befinden drei Assembler-Versionen mit unterschiedlichen Startadressen.
Die Adressen sind im Dateinamen enthalten; bitte suchen Sie sich die für Sie am besten geeignete Version heraus.
Der Assembler liest den Quelltext Zeile für Zeile ein und produziert die lauffähige Objektdatei. Dazu benötigt er zwei Arbeitsbereiche, die nach dem Laden des Assemblers mit BLOAD definiert wwrden müssen:

```
CALL Aufrufadresse;#Bank1,Startadresse1,Endadresse1,#Bank2,
     Startadresse2,Endadresse2
```

Steht der Assembler beispielsweise ab &C0C5 (wenn Sie AS_C0C5 geladen haben), und kann ein RAM-Modul im Slot "S1:" voll für die Assemblierung verwendet werden, so könnte der Aufruf wie folgt aussehen:

```
CALL &C0C5;#0,&80C5,&BFFF,#1,&8000,&BFFF
```

Bei dieser Definition gibt es, außer den im BASIC-Handbuch beschriebenen Fehlermeldungen, noch zwei weitere:
ERROR 19:Wert außerhalb des RAM
ERROR 40:Startadresse gößer Endadresse

Sollten Sie nicht mehr wissen, welchen Bereich Sie reserveirt haben, so können Sie mit

```
CALL Aufrufadresse;V
```

sich diesen anzeigen lassen.
Hier würde der Aufruf CALL &C0C5;V also anzeigen:
#0,&80C5,&BFFF,#1,&8000,&BFFF

Um abschätzen zu können, wie groß diese Arbeitsbereiche zu wählen sind (was vom Quellprogramm abhängig ist), gebe ich hier die interne Organisation der Bereiche an:

Bereich 1: fertiger Maschinencode
definierte Symbole (8 Byte pro Symbol)

Bereich 2: Macrodefinitions-Text (9 Byte + Zeichenzahl bis ENDM)
Macroexpansions-Argumente beim Macroaufruf
(7 Byte + Zeichenzahl der Argumente)
Vorwärtsreferenzen (5 Byte + Zeichenzahl)

Das bedeutet zum Beispiel, daß ein Quelltext mit vielen symbolischen Namen einen großen Bereich 1 benötigt.

<<< START >>>

Beim eigentlichen Start des Assemblers geben Sie die Namen der Quelldatei(en), der Objektdatei und optional den Namen einer Symboldatei an, in der alle Symbole mit ihrem Wert gespeichert werden können:

```
CALL Aufrufadresse;">Symbolfile,Quellfile1,...,QuellfileX,
     Objektfile"
z.B. CALL &C0C5;">X:SYMBOL.T,X:TEST.ASM,S2:TEST.BIN"
```

Alle Dateinamen sind in der üblichen SHARP-Norm, d.h. mit DEVICE:NAME.EXTENSION anzugeben.

Der letzte Dateiname im Befehls-String ist die Objektdatei.

ERROR 1:Objektdatei fehlt oder anderer Syntaxfehler

<<< ABLAUF >>>

Während der Assemblierung stehen in der ersten Zeile der Anzeige der Quelldateiname und die Nummer der augenblicklich bearbeiteten Zeile. Bei der Expansion eines Macros wird in der zweiten Zeile dessen Name angezeigt, die dritte ist für Fehlermeldungen vorgesehen.
Wenn der Assembler einen Fehler im Quelltext erkennt, dann können Sie die entsprechende Zeile in der vierten Zeile berichtigen (mit ENTER abschließen) oder mit BREAK die Assemblierung abbrechen.
Der Mini-Editor läuft im INSERT-Modus; zum Löschen von Zeichen oder der ganzen Zeile benutzen Sie wie gewohnt die BS-, DEL- bzw. CL-Tasten.
Am besten notieren Sie sich die Zeilennummer, bei der ein Fehler auftrat, damit Sie ihn später auch im Quelltext berichtigen können.
Nach dem Durchlauf werden die Vorwärtsreferenzen gelöst. Tritt hierbei ein Fehler auf, so wird dieser angezeigt. Nach Betätigung mit ENTER wird weitergearbeitet, damit gleich alle fehlerhaften Vorwärtsreferenzen gefunden werden können. Ein Abbruch mit BREAK ist aber auch möglich.
Wenn kein Fehler gefunden wurde werden das Objektfile und gegebenenfalls die Symboldatei abgespeichert. Damit ist die Assemblierung abgeschlossen.

Die Symboldatei enthält in alphabetischer Ordnung die Namen aller definierten Symbole und ihre Werte in hexadezimaler Form. Sie kann mit LOAD* oder mit INPUT# gelesen und beliebig weiterverarbeitet oder mit LLIST* ausgedruckt werden.

<<< QUELLTEXT >>>

Es folgt die Beschreibung der Assembler-Syntax:

ZEILE:

1. KOMMENTAR (beginnt mit einem Apostroph oder Semikolon)

2. MNEMONIC (Z80-Befehl mit der üblichen Syntax;
   Es sind auch die nicht dokumentierten Befehle
   wie SLL A und die Aufteilung der IX/IY-Register in
   eine Low- und High-Hälfte, z.B. LD A,XH ,erlaubt;
   Für Immediates, also Zahlenangaben bei bestimmten
   Befehlen, gelten die Regeln der EXPRESSION, siehe
   unten)

3. DIRECTIVES (Assembleranweisungen)
   Hinter den Mnemonics oder Directives dürfen Kommentare ohne
   besondere Kennzeichnung stehen, da der Assembler das Ende
   einer gültigen Eingabe selbst erkennt.

SYMBOL:
Ein Name, der einen bestimmten Wert darstellt
-Der Name darf aus Buchstaben (Groß- und Kleinschrift), Ziffern
(außer als erstes Zeichen) und den Sonderzeichen _#.' bestehen.
Beginnt er mit einem Punkt, so stellt er ein lokales Symbol dar,
das später einen neuen Wert zugewiesen bekommen darf.
-Bei Symbolnamen mit mehr als 6 Zeichen werden nur die ersten 6
Zeichen beachtet, d.h. SYMBOLX und SYMBOLY sind gleich.
-Nicht erlaubt sind Registernamen u.ä. wie HL.
-Bsp.:
START , c_CR , .XXXXX

CONSTANT:
Ein bereits definiertes Symbol, eine Zahl oder das @-Zeichen
-Zahlen werden als dezimal aufgefaßt. Hexadezimalzahlen beginnen
mit dem &-Zeichen, Binärzahlen beginnen mit dem %-Zeichen.
-Das @-Zeichen stellt den Wert des momentanen Prgrammzählers (PC)
im Code dar und sollte nur bei relativen Sprüngen wie JR @+2
verwendet werden, da sonst nicht immer eindeutig ist, wohin der
imaginäre PC bei der Assemblierung gerade zeigt.
Für den ASCII-Code eines Zeichens darf auch das Zeichen selbst,
in Anführungszeichen gesetzt, verwendet werden.
-Bsp.:
c_CR , 999 , 0FED4 , %0111011 , @ , "a"

VALUE:
Ein Rechenausdruck, bestehend aus Konstanten und Rechenzeichen
-Die Rechenzeichen in der Reihenfolge ihrer Priorität:
unäre Operatoren (mit nachstehendem Argument):
5: + Vorzeichen positiv
   - Vorzeichen negativ (Zweierkomplement)
   ~ NOT-Funktion (Einerkomplement, alle Bits invertiert)
binäre Operatoren (zwischen den Argumenten):
4: < Links-Shift um die gebene Zahl von Bits
   > Rechts-Shift
3: * Multiplikation
   / Division (ganzzahlig)
2: + Addition
   - Subtraktion
1: ^ Logisches AND
   ! Logisches OR
-Klammern dienen zum Durchbrechen der Prioritäts-Reihenfolge
-Wertüberläufe bei der Berechnung bleiben unberücksichtigt, das läßt dem Programmierer volle Freiheit bei der Interpretation von Zahlenwerten
2-Byte-Value:-65536 ... +65535
1-Byte-Value:-256 ... +255
Offset :   :-128 ... +127
-Bsp.:
5*(3+WERT/0200) , -(1<5) ,"P"-(XXXX!7)

FORWARD REFERENCE:
Eine Vorwärtsreferenz ist die Verwendung eines Symbols vor der Wertzuweisung (=Definition)
-Bsp.:
JP ENDE (wobei ENDE: später im Quelltext steht)

EXPRESSION:
Ein Rechenausdruck (Value), der auch Vorwärtsreferenzen enthalten darf
-Ausdrücke mit Vorwärtsreferenzen werden zwischengespeichert und am Ende der Assemblierung, also nach Definition (hoffentlich) aller Symbole, berechnet.

LABEL:
Eine (Sprung-)Marke im Programm
-Bsp.:
ENDE:

MACRO:
Textzeilen, die beim Aufruf durch den vorher definierten Macronamen in den Text eingebunden werden
-Dient zur Erhöhung der Lesbarkeit des Quelltexts, ersetzt häufig im Text vorkommende Befehlsfolgen oder bildet neue Assembler-Befehle, denen auch Argumente übergeben werden können
-Bsp.:

```
MACRO EXSPDE
EX DE,HL
EX (SP),HL
EX DE,HL
ENDM
```

INCLUDE FILE:
Eine Datei, die vor dem eigentlichen Quelltext des zu assemblieren Programms eingelesen wird und die meist nützliche Definitionen für mehrere Programme enthält
Der PC-1600-Assembler ermöglicht dies, weil im Befehls-String beim Aufruf mehrere Quelldateien angegeben werden können
-Bsp.:
Eine Datei mit folgendem Inhalt:

```
EQU DOTSET 0127 Startadresse der PSET-Routine
EQU CRSRXY 0F05F Systemadresse der Cursorposition
MACRO BCALL Bank,Adresse
RST 020
DEFB \1
DEFW \2
ENDM
```

<<< DIRECTIVES >>>

-ORG Value [, Bank]
Das fertige Maschinenprogramm beginnt mit der angegebenen Adresse in der optionalen anzugebenden Bank (wenn nicht angegeben, dann Bank=0).
ORG darf nur einmal, und zwar vor der ersten Anweisung, die einen Maschinencode erzeugt, vorkommen.

-AUTO value [, Bank]
Beim laden des fertigen Maschinenprogramms wird ein Autostart an der angegebenen Adresse in der optional anzugebenden Bank (wenn nicht angegeben, dann Bank=0) durchgeführt.
AUTO darf überall im Programm stehen.

-EQU Symbol Value
Einem Symbol wird ein bestimmter Wert zugewiesen
Nur lokale Symbole (beginnen mit einem Punkt ,z.B. .ABC) dürfen mehrmals mit einem neuen Wert definiert werden.

-DEFB Expression1,Expression2,...
 DEFB "String"
 Im Code werden 1-Byte-Werte der Reihe nach abgelegt.

-DEFW Expression1,Expression2,...
 Im Code werden 2-Byte-Werte, Low vor High, der Reihe nach abgelegt.

-DEFS Value
 Im Code wird ein freier Platz mit der gegebenen Zahl von Bytes reserviert und mit 0 gefüllt.

-Label:
 Dem gegbenen Label-Namen wird der Wert des momentanen Programmzählers übergeben (für relative und absolute Sprungmarken und Adressen im Programm; entspricht EQU Labelname @)

-END
 Ende der Assemblierung, alle folgenden Quelltext-Zeilen werden ignoriert.

Bedingte Assemblierung:
Bei Nichterfüllung der Bedingung werden die entsprechenden Zeilen ignoriert.

-IFZ Value
 Assemblierung nur, wenn der Wert=0 ist
 IFx -Anweisungen dürfen geschachtelt sein.
 Die Bedingung gilt bis ELSE/ENDC.

-IFNZ Value
 Bedingung: Wert<>0

-IFP
 Value Bedingung: Wert positiv (0 ... 32767)

-IFM Value
 Bedingung: Wert negativ (-32768 ... 0)

-IFD Expression
 Bedingung: Alle Symbole im Ausdruck bereits definiert

-IFND Expression
 Begingung: Mindestens ein Symbol im Ausdruck nicht definiert
 z.B.:
 IFND CRSRXY
  EQU CRSRXY 0F05F
 ENDC

-ELSE
Assemblierung im anderen Fall, also Umkehrung der letzten Bedingung

-ENDC
Ende der bedingten Assemblierung

Macroanweisungen:

-MACRO Symbol
Beginn der Definition eines Macros mit dem angegebenen Namen
Macrodefinitionen dürfen, im Gegensatz zu Macroexpansionen (d. h. Aufruf eines bereits definierten Macros), nicht rekursiv sein, d.h. sie dürfen nicht geschachtelt werden.
Im Text der Macrodefinitionen können besondere Zeichenfolgen stehen, die bei der Expansion durch andere Zeichen ersetzt werden:
\n Hier wird, Zeichen für Zeichen, das n. Argument des Macroaufrufs eingesetzt.
\Ø Hier wird eine Dezimalzahl eingesetzt, die die Zahl der Argumente, die beim Aufruf übergeben wurden, angibt.
\@ Erzeugung einer Zahl (Ø ... 255), die angibt, wie oft das Macro bereits aufgerufen wurde
Dies wird oft zur Erzeugung eindeutiger Labels innerhalb eines Macros eingesetzt, z.B.:

```
MACRO LOOP
LOO\@:
DJNZ LOO\@
ENDM
```

-ENDM
Ende der Macrodefinition

-EXIT
Ausbruch aus der Macroexpansion, d.h. die Abarbeitung eines Macros wird, meist durch bedingte Assemblierung, beendet.

-Macroname Argument1,Argument2,...
Abarbeitung eines bereits definierten Macros mit den Argumenten, die an den entsrechenden Stellen im Macrotext eingesetzt werden. Macroexpansionen dürfen rekursiv sein.

<<< FEHLERMELDUNGEN >>>

Speicherfehler, die zum Abbruch der Assemblierung führen:

Code table full	:Bereich 1 zu klein; die Objektdatei wurde zu lang
Symbol table full	:Bereich 1 zu klein; es wurden zu viele Symbole definiert
Macro table full	:Bereich 2 zu klein; es wurden zu viele Macros definiert oder zu viele Expansionen ineinander geschachtelt
Expression table full	:Bereich 2 zu klein; es liegen zu viele Vorwärtsreferenzen vor

Sonstige, meist mit dem Mini-Editor korrigierbare, Fehler:

Symbol not defined	:Nicht erlaubte Vorwärtsreferenz
Syntax error	:Fehlerhaftes Befehlsargument oder Tippfehler
Formula too complex	:Rechenausdruck zu tief geschachtelt
Value out of range	:Wert außerhalb des zulässigen Bereichs
ORG missing/double	:ORG-Anweisung fehlt oder kommt wiederholt vor
No Macro	ENDM/EXITM außerhalb eines Macros
Macro defined in macro	:Macrodefinition innerhalb eines anderen Macros
Reserved Symbol	:Registername als Symbol verwendet
Line too long	:Zeilenlänge von 79 Zeichen beim Einfügen von Macro-Argumenten überschritten

<<< BEISPIELE >>>

Beispiel 1

Folgendes erste kleine Maschinenprogramm schreibt den Text 'HALLO'in die Anzeige.
Das Programm liegt in dem Variablenspeicher des PC-1600 und wird automatisch gestartet.

* * Quelltext * *

```
'Hallo
'
'Demoprogramm für den
'PC-1600 ASSEMBLER
'
'
EQU START &F900 'Das Programm wird in die Standardvariablen geladen
'
EQU CLS &0112
EQU HOME &0109
EQU PRINT &00EB
EQU CR &0D
'
ORG START
AUTO START
'
CALL CLS
CALL HOME
LD DE,TEXT
LD A,CR
CALL PRINT
RET
'
TEXT:
DEFB "**  Hallo  **"
DEFB CR
'
END
```

Bei der Assemblierung wird folgende Symboltybelle erzeugt:

```
0112 CLS
000D CR
0109 HOME
00EB PRINT
F900 START
F90F TEXT
```

Beipiel 2

Diese Programm zeigt Ihnen ein gesetztes PASS-Wort an. Das Programm arbeitet nur auf Rechnern der neuesten PC-1600 Genration. Die entsprechenden Adressen für 'alte' Rechner können dem PC-1600 Systemhadbuch entnommen werden.

* * Quelltext * *

```
'PASSWORT LISTER
'
'Das Programm liegt in den Standardvariablen
'
EQU START &F900
'
EQU ADR1 &A87B
EQU ADR2 &A9A1
'
EQU CHKMOD &7FFF
EQU NEWMOD 5
'
EQU CLS &0112
EQU CRSET &0115
EQU PRTSTR &0106
'
EQU TIMER &01D5
EQU SRINP &13
'
EQU NUL &00
'
MACRO CRSET_ X,Y
LD DE,\1*256+\2
CALL CRSET
ENDM
'
'
ORG START
AUTO START
'
'
'Bank sichern
IN A,(&31)
PUSH AF
'
'Anzeige löschen und Cursor setzen
CALL CLS
CRSET_ 0,0
'
'Neues Modell prüfen
LD A,&00
OUT (&31),A
LD A,(CHKMOD)
```

```
CP NEWMOD
JR Z,OK
'
'Alter PC-1600, Text ausgeben
LD DE,TEXT3
JR PRINT
'
OK:
'
'Auf Passwort prüfen
LD C,SRINP
CALL TIMER
BIT 2,A
JR NZ,P_SET
'
'Kein Passwort, Text ausgeben
LD DE,TEXT1
JR PRINT
'
P_SET: Passwort
'
'Timer-RAM lesen
DI
LD HL,BUFFER
LD BC,&0880
LD A,&60
OUT (&31),A
CALL ADR1
CALL ADR2
LD A,&5F
OUT (&35),A
EI
'
'Passwort anzeigen
LD DE,TEXT2
'
PRINT: Print Text
'
CALL PRTSTR
'
'Programm beenden
POP AF
OUT (&31),A
RET
'
TEXT1:
DEFB "No Password!"
DEFB NUL
'
TEXT2:
DEFB "Password: "
BUFFER:
```

```
DEFS 8
DEFB NUL
'
TEXT3:
DEFB "Old PC-1600!"
DEFB NUL
'
ENDE:
END
```

Bei der Assemblierung wird folgende Symboltabelle erzeugt:

```
A87B ADR1
A9A1 ADR2
F961 BUFFER
7FFF CHKMOD
0112 CLS
0115 CRSET
F977 ENDE
0005 NEWMOD
0000 NUL
F91C OK
F943 PRINT
0106 PRTSTR
F92A P_SET
0013 SRINP
F900 START
F94A TEXT1
F957 TEXT2
F96A TEXT3
01D5 TIMER
```

<<< DISKETTENINHALT >>>

Auf der mitgelieferten Diskette befinden sich folgende Dateien:

- A<S_C0C5	Assembler für Adreßbereich C0C5-D750 Lage: Unterhalb von Basic-Programmen Reservierung: NEW "S0:",&1751 (wenn nur S0: als Memory initialisiert ist)
- AS_D500	Assembler für Adreßbereich D500-EB8B Lage: über Basic, unter Disk-Puffer
- AS_D900	Assembler für Adreßbereich D900-EF8B Lage: bei PC-1600 ohen Floppy oder über Disk-Puffer, unter System RAM Reserveirung: A=&1700 : CALL &2DD,A
- HALLO.ASM	Quelltext für HALLO.BIN
HALLO.SYM	Erzeugtes Symbolfile bei der Assemblierung von HALLO.ASM
HALLO.BIN	Objektfile
- PASSLIST.ASM	Quelltext für PASSLIST.BIN
PASSLIST.SYM	Erzeugtes Symolfile bei der Assemblierung von PASSLIST.BIN
PASSLIST.BIN	Objektfile

<<< Literaturhinweise >>>

- David von Oheimb

  Das Systemhandbuch für den PC-1600

  ISBN 3-925641-08-4

  bei: Klaus Ditze, D-5354 Weilerswist

- Rodnay Zaks:

  Programmierung des Z80

  bei: Sybex, Düsseldorf