Moppel

Assembler – Disassembler

Dokumentation

Stand 10.10.85

Assembler-Menü (Aufruf per "a" oder "A", gefolgt von "Return"):

O: Pass Zero
Wie Pass 1, aber ohne eine evtl. vorhandene Symboltabelle zu löschen, d.h. ein "Anhängen" an vorhandene Quellprogramme ist möglich.

1: Pass One
Legt vom gespeicherten Quellprogramm (das im Bereich 9000h...EFFFh steht) eine Tabelle der verwendeten symbolischen Namen an (Symboltabelle TABSYM im Bereich von 8000h...8FFFh).

2: Pass Two
Setzt das Quellprogramm in den 8085-Maschinencode um und überträgt ihn in den Objektbuffer OBJBUF (F000...FFFF); im Objekt-Pointer (2F5Eh/2F5Fh) steht die Endadresse für dieses Maschinenprogramm.

3: Pass Three
Gibt ein paralleles Listing aus von Zeilennummer Z-Nr, Speicheradresse Adr (des späteren Zielbereichs), Maschinencode und Quellprogramm einschließlich Kommentar; Format:

```
0815 2830  31 FF 2E  START  LXI SP,2EFFH    *Stack-Pointer laden
Z-Nr  Adr  Ma-Code   Quellprogramm          ab"*": Kommentar
```

D: Disassembler
Rückübersetzen eines beliebigen Speicherbereichs ins mnemotechnische Format der Assembler-Abkürzungen; Rücksprung zum Assembler: Tasten **CTL+C** gleichzeitig drücken.

E: Editor
Zurück zum Editor und Weitermachen mit dem Erstellen bzw. Ändern des Quellprogramms.

H: HEX-Mode
Ausgabe der Speicherinhalte im hexadezimalen Format (Basis 16).

I: In/Out on/off
Möglichkeit, das Quellprogramm nicht zwischen RAM und Assembler, sondern zwischen externen Peripheriegeräten und Assembler zu transferieren (separate Treiber-Routinen sind nicht enthalten!).

M: Monitor
Rücksprung ins Monitor-Grund-Menü (Anweisungs-Ebene).

O: OCT-Mode
Ausgabe der Speicherinhalte im oktalen Format (Basis 8).

P: Printer on/off
Umschalten des Drucker-Flipflops (in Zelle 37AFh) zum Parallelbetrieb eines Druckers (4800 Baud, über das Serielle Interface).

S: Symbol Table
Ausgabe der verwendeten Symbole mit zugehöriger Speicheradresse.

CTL+C: HELP-Funktion (Ausgabe der möglichen Ass-Anweisungen).

Fehlercodes: A00: Anweisung falsch (entstammt nicht dem Menü)
A01: Zusammengesetztes Label fehlerhaft
A02: Symboltabelle voll
A03: Symbol ist mehrfach definiert worden
A04: Label unvollständig (z.B. Zahl ohne angeh. "H")
A05: Zwei Label in einer Zeile (z.B. fehl.Trennzeichen
A06: Mnemo fehlerhaft (z.B. fehlende Registerangabe)
A07: Symbolname falsch (beginnt z.B. nicht mit ASCII-Z)

Die Arbeit mit dem Assembler

Beim Erstellen von Mikrocomputer-Programmen besteht ein grundsätzliches Problem, dessen Lösung Aufgabe eines Assemblers ist: Der Computer, der das Programm später einmal ausführen soll, versteht ausschließlich Bitmuster in Form logischer Pegel, die sogenannte Maschinensprache also, die Sie, meist in hexadezimaler Notation, sicherlich zur Genüge kennen. In dieser Form zu programmieren ist eine Sträflingsarbeit, weil kein Mensch die Maschinencodes zu den einzelnen Befehlen im Kopf hat und weil vor allem bei Sprungbefehlen (sowie Unterprogramm-Aufrufen) umständliche Adreßberechnungen erforderlich sind, die beim Einfügen oder Löschen eines einzigen Bytes wieder hinfällig werden.

Darum ist es für den Anwender wünschenswert, seine Programme im sogenannten Assemblercode zu erstellen; das sind "mnemotechnische" Abkürzungen, die einen Sinnzusammenhang ("Eselsbrücke") herstellen zwischen Bezeichnung und Wirkung eines Befehls, also z.B. **LDA 4711h** für "Lade Akku mit (dem Inhalt der Speicherstelle) 4711h". Adressen müssen hierbei nicht mehr absolut angegeben werden, sondern sie können durch symbolische Namen (die sogenannten Labels") ersetzt werden, was die Programmerstellung ganz erheblich vereinfacht (**LDA LINBUF** wäre statt **LDA 4711h** auch möglich).

Aufgabe eines Assemblers ist es, ein im Assemblercode erstelltes Programm (das sogenannte Quell-Listing oder Quellprogramm) umzusetzen in den passenden Maschinencode (das sogenannte Objektprogramm) und dabei insbesondere die zu den verwendeten symbolischen Namen gehörenden Absolutadressen herauszufinden. Der Assembler selbst kann weder ein Quellenprogramm erstellen noch Änderungen darin vornehmen. Dazu dient ein völlig anderes Programm (z.B. ein Editor), das zwar eng mit dem Assembler zusammenarbeitet, mit ihm selbst aber nichts zu tun hat.

Der MOPPEL-Assembler erwartet, daß sein Quellprogramm ab Adresse 9000h im RAM steht (mit '0Dh'=Return als Zeilenabschluß). Nach erfolgter Umsetzung schreibt er den Maschinencode ans RAM-Ende in den obersten 4-K-Bereich F000h...FFFFh, von wo aus Sie ihn z.B in ein EPROM umkopieren können. Eine Tabelle der im Quellprogramm verwendeten symbolischen Namen mit den bei der Umsetzung ermittelten Absolutadressen legt dieser Assembler am RAM-Anfang ab (Symboltabelle im unteren 4-K-Bereich 8000h...8FFFh).

Folgende Zusatzvereinbarungen gelten für diesen Assembler:
ORG xxyy definiert die Anfangsadresse des späteren Machinenprogramms (das unabhängig davon zunächst im Bereich F000h...FFFFh steht). **END** gibt dem Assembler das Ende des Quellprogramms an. **EQU** kann, muß aber nicht vor Konstanten stehen, die definiert werden (z.B. ist 'ORG 2900H' gleichbedeutend mit 'ORG EQU 2900H'). Labels am Zeilenfang sind mit einem **:** abzuschließen. Das Sternchen "*****" definiert Kommentare (bzw. Kommentarzeilen), um die sich der Assembler beim Umsetzen nicht kümmert. **h (H)** bzw. **o(O)** an einen Term angehängt bedeutet, daß die Eingabe hexadezimal (bzw. oktal) aufzufassen ist. Zahlen müssen unbedingt durch dieses Anhängsel gekennzeichnet werden; bei HEX-Zahlen, die mit A...F beginnen, ist außerdem eine führende "0" (Null!) voranzustellen. Ein in Anführungszeichen **(ASCII-Marker)** stehendes Zeichen wird vom Assembler im zugehörigen ASCII-Code verarbeitet, wenn der Pseudo-Operator **DB** vorangestellt wird (**DB "M"** z.B. führt zum Einfügen des ASCII-Codes '4Fh' für 'M' ins Maschinenprogramm). **DB xx** definiert das Byte 'xx' als Konstante und setzt diese ins Maschinenprogramm ein (anstelle eines Befehls oder einer Adresse). **DW xxyy** definiert das Doppel-Byte xxyy als 16-Bit-Konstante und setzt diese ins Maschinenprogramm ein (erst 'yy', dann 'xx'). **DS nnnn** definiert einen Speicherbereich von 'nnnn' Bytes, der im späteren Maschinenprogramm frei bleiben soll (z.B. für RAM-Zellen, die als Parameter-Buffer dienen).

1. Ein Steckbrief vom Assembler

Beim Erstellen von Mikrocomputer-Programmen besteht ein grundsätzliches Problem, dessen Lösung **Aufgabe eines Assemblers** ist: Der Computer, der ein Programm später einmal ausführen soll, versteht ausschließlich Bitmuster in Form logischer Pegel ("Nullen" und "Einsen", repräsentiert durch Spannungspegel von 0 V bzw. +5 V). Das ist die sogenannte Maschinensprache, die Sie, meist in hexadezimaler Schreibweise, sicherlich zur Genüge kennen. In dieser Form zu programmieren ist eine Sträflingsarbeit, weil kein Mensch die Maschinencodes zu den einzelnen Befehlen im Kopf hat, und weil vor allem bei Sprungbefehlen (sowie Unterprogramm-Aufrufen) umständliche Adreßberechnungen erforderlich sind, die beim Einfügen oder Löschen eines einzigen Bytes wieder hinfällig werden.

Darum ist es für den Programmierer von unschätzbarem Vorteil, wenn er seine **Programme** im sogenannten **Assemblercode** erstellen kann; das sind mnemotechnische Abkürzungen, die einen **Sinnzusammenhang** (Eselsbrücke) herstellen zwischen Bezeichnung und Wirkung eines Befehls. Ein Beispiel: In einem Programm soll zum Inhalt des Akkumulators der Inhalt des Registers B addiert werden; dazu wird der Befehl 'ADD B' verwendet, der dies (im 8085-Befehlssatz) bewirkt: Bei seiner Ausführung addiert die Zentraleinheit zum Inhalt des Akkus den Inhalt des B-Registers. In Maschinensprache lautet dieser Befehl '80h' (hexadezimal abgekürzt), wohinter sich das Bitmuster '1000 0000b' verbirgt. Und Sie werden zugeben, daß es beim Programmieren einfacher ist, 'ADD B' zu formulieren als umständlich die '80h' aus einer Tabelle zu suchen und einzusetzen.

Aber ein Assembler soll noch viel mehr, er soll nämlich **symbolische Bezeichnungen** (sogenannte **Label**, gesprochen "Lejbel") **verwalten**; darunter ist folgendes zu verstehen: Angenommen, man benutzt eine Speicherzelle im RAM als Zähler für ausgedruckte Zeilen, dann tauft man diese RAM-Zelle 'LINCNT' (als Abkürzung für 'Line Counter'; jede andere, sogar deutsche Bezeichnung wäre auch möglich!) und spricht sie bei der Programmerstellung nur mit diesem symbolischen Namen an. Und an einer einzigen Stelle im Assemblerprogramm definiert man, welche Adresse man für 'LINCNT' vorsieht: So bedeutet beispielsweise 'LINCNT EQU 28CFH', daß der Assembler bei jedem Auftauchen von 'LINCNT' im Maschinenprogramm die hexadezimale Adresse 28FCh einsetzen muß.

Diese **symbolischen Namen** kann man nun nicht nur für RAM- (oder ROM-)Zellen verwenden, sondern auch **für Sprungziele** innerhalb des Programms. Wenn man beispielsweise in einer Schleife einen Register-Inhalt bis auf Null herunterzählen will, kann man den Schleifenanfang mit 'LOOP' bezeichnen (engl. 'Schleife', gesprochen 'Luup'; dieses stereotype Label finden Sie in 99% aller Assembler-Programme wieder, weil Programmierer einfallslose Leute sind). Am Schleifenende (nach dem Herunterzählen des Registerinhalts) steht der Befehl 'JNZ LOOP' (vom engl. 'Jump On No Zero To LOOP'; Springe nach 'LOOP', wenn das Herunterzählen **nicht** zum Ergebnis **Null** geführt hat); und in diesem Fall ist 'LOOP' der symbolische Name für die Zieladresse im bedingten Sprungbefehl 'JNZ'. Eine der vornehmsten Aufgaben des Assemblers ist es, die zu 'LOOP' gehörende Absolutadresse zu ermitteln und beim Erstellen des Maschinenprogramms einzusetzen (Näheres s.u.).

1.1 Kurz und knapp: Assembler und Disassembler

Fassen wir also zusammen: Der **Assembler** ist ein Programm, das die Arbeit der **Programmerstellung** ganz wesentlich **vereinfacht.** Aufgabe eines Assemblers ist es, ein im Assemblercode erstelltes Programm (das sogenannte **Quellprogramm** oder Quell-Listing) **umzusetzen** in den passenden Maschinencode (das sogenannte **Objektprogramm**). Dabei setzt der Assembler für **symbolische Namen** (die Label) **Absolutadressen** ein, die er in einem separaten Arbeitsgang ermittelt. Der Assembler selbst kann weder ein Quellprogramm erstellen noch Änderungen darin vornehmen. Dazu dient ein völlig anderes Programm (der sogenannte Editor), das zwar eng mit dem Assembler zusammenarbeitet, mit ihm selbst aber nichts zu tun hat. **Assemblieren** heißt soviel wie (das Maschinenprogramm) zusammensetzen, es **übersetzen; editieren** bedeutet eine **reine Textbearbeitung**, d.h. Erstellen, Einfügen, Löschen und Modifizieren vorhandener Texte (im hier betrachteten Fall auf das Quellprogramm bezogen).

Um die Begriffsbestimmung komplett zu machen: Ein **Disassembler** ist ebenfalls ein Programm, und zwar eins, das die **umgekehrte Funktion** eines Assemblers übernimmt; einen Disassembler füttert man mit Objektcode (d.h. in Maschinensprache vorliegenden Befehlen), und heraus kommen die mnemotechnischen Assemblercodes (also z.B. 'ADD B', wenn der Disassembler auf den Maschinenbefehl '80h' stößt). Natürlich kann ein Disassembler nicht ahnen, welche Label bei der ürsprünglichen Erstellung des Quellprogramms vergeben worden sind; bei der Rückübersetzung tauchen daher anstelle der symbolischen Namen stumpfsinnig die im Maschinenprogramm enthaltenen Absolutadressen auf.

2. Angebot auf dem Silbertablett: Das Menü

Was Sie bis hierhin über Assembler und Disassembler gelesen haben, ist allgemeingültig, bezieht sich also auf jeden beliebigen Assembler. Speziell für den **MOPPEL-Assembler** gelten die folgenden Aussagen (die natürlich in ähnlicher Form auch auf viele andere Artgenossen zutreffen). Der MOPPEL-Assembler ist ROM-resident, d.h. er steht, fest gespeichert, in einem EPROM abrufbereit zur Verfügung und muß nicht erst geladen werden (z.B. von Diskette). Dieser Assembler ist ein **Drei-Pass-Assembler**, d.h. die Übersetzung in den Maschinencode vollzieht sich in mehreren Durchläufen (das ist nicht etwa ein Nachteil, sondern ein Plus!). Im MOPPEL-Assembler-EPROM ist außerdem ein **Disassembler** enthalten, der die eben erwähnte Rückübersetzung von Maschinenprogrammen übernehmen kann. Dieses EPROM (4 KBytes Kapazität) muß auf **Platz #6 der großen Speicherkarte** eingesetzt werden.

Der **Assembler-Aufruf** von der Monitor-Anweisungs-Ebene aus erfolgt mit der **Anweisung "A"** (oder "a" für "Assembler): 'Anweisung an MOPPEL >a'. Es erscheint dann folgende Bildschirm-Meldung (ein sogenanntes Menü, bei dem Sie das Gewünschte auswählen können):

```
MOPPEL-Assembler V 8.2
Copyright hms'84

 0/1/2/3: Pass #
 D: Disassembler
 E: Editor
 H: HEX-Mode
 I: In/Out  on/off
 M: Monitor
 O: OCT-Mode
 P: Printer on/off
 S: Symbol Table
```

Bei Eingabe von "D" (bzw."d"), E (bzw."e") oder "M" (bzw."m") geht es kommentarlos **weiter zum Disassembler, Editor** oder **zurück zum Monitor**; die **P-Eingabe** schaltet einen (über das serielle Interface) angeschlossenen **Drucker parallel**, so daß alles, was auf dem Bildschirm erscheint, auch auf dem Drucker ausgegeben wird; die P-Funktion hat Flipflop-Verhalten (wie der Druckknopf einer Nachttischlampe), d.h. die nächste Eingabe eines "P" schaltet den Paralleldruck wieder aus. Einen fortlaufenden **Druck halten Sie** durch Druck auf eine beliebige Taste **an**; bei Betätigung von **CTL+C** zusammen **brechen Sie den Druckvorgang ab.**

Ähnliches wie für die Umschaltung per P-Statement gilt für die **I-Anweisung**, die allerdings **für Sondereinsätze reserviert** ist (wenn jemand den Quellcode nicht aus dem RAM, sondern von anderer Quelle bereitstellt; die dafür benötigte Software muß er dann allerdings selbst erstellen).

"H" (bzw. "h") und "O" (bzw."o") geben dem Assembler an, wie er nach erfolgter Umsetzung das Objektprogramm ausgeben soll: Entweder **hexadezimal** (Abkürzung von 4-Bit-Gruppen durch "0...9" und "A...F") **oder oktal** (Abkürzung von 3-Bit-Gruppen durch "0...7"); das oktale Format hat heute fast nur noch historische Bedeutung. In früheren Zeiten waren Computer mit 12-Bit-Wortlänge sehr verbreitet (z.B. PDP-8), wo man ein Wort in vier 3-Bit-Gruppen aufgeteilt hat, um es abgekürzt darzustellen.-

Und wenn Sie das Assembler-Menü betrachten, bleiben jetzt noch genau fünf Anweisungen übrig, die den Assembler direkt betreffen (**Direkt-Kommandos**, im Gegensatz zu den zuvor erläuterten **Verwaltungs-Anweisungen** "D", "E", "M", "P" usf.): **Pass 0/1/2/3** veranlaßt den Assembler, einen der Durchläufe Nr.0, 1, 2 oder 3 auszuführen (s.u.), und "S" (bzw."s") führt zur Ausgabe der **Symboltabelle** (auch dazu finden Sie weiter unten alles Nähere).

2.1 Die Basis für's Assemblieren

Der Assembler soll bestimmungsgemäß die Umsetzung eines vorhandenen Quellprogramms in den Maschinencode übernehmen; dies kann er natürlich nur dann, wenn **folgende Voraussetzungen** erfüllt sind:

1. Auf **Platz #6** der großen Speicherkarte muß das **Assembler-EPROM** eingesetzt sein; zweckmäßigerweise ist auf **Platz #7** der Editor vorhanden, um Modifikationen am Quellprogramm durchführen zu können. Der Editor ist jedoch keine Voraussetzung für den Betrieb des Assemblers!

2. Auf der großen Speicherkarte müssen **ab 8000h mindestens 2 KB RAM** bestückt sein: Im Bereich ab 8000h (bis maximal 8FFFh, also max.4 KBytes) legt der Assembler seine **Symboltabelle** an (s.u.); um den vorgesehenen Platz für die Symboltabelle voll auszuschöpfen, müssen auf der großen Speicherkarte ab 8000h zwei 2-K-RAMs bestückt sein (Plätze #80 und #88).

3. Auf der großen Speicherkarte müssen **ab 9000h mindestens 2 KB RAM** bestückt sein: Im Bereich ab 9000h (bis maximal EFFFh, also max.24 KBytes) erwartet der Assembler das zu übersetzende **Quellprogramm**; je nach Länge Ihres Quellprogramms müssen Sie ab 9000h also RAM-Kapazität vorsehen, wobei jedes Zeichen im Quell-Listing (auch Leerzeichen!) ein Byte im RAM belegt!

4. Der Assembler legt den Maschinencode des **Objektprogramms im Bereich F000...FFFFh** ab (oberste 4 K auf der großen Speicherkarte); normalerweise müssen hier also mindestens 2 KB RAM bestückt sein. Allerdings kann man diesen Bereich auch mit dem Pseudo-Operator 'OFS' (s.u.) **umdefinieren**, wenn beispielsweise ab F000h kein RAM bestückt ist. Es müssen also nicht unbedingt ab F000h RAMs vorhanden sein, weil es die Ausweichmöglichkeit über 'OFS' gibt.

3. Sagt mehr als viele Worte: Ein Beispielprogramm

Die **Wirkung der Assembler-Direkt-Anweisungen** wird wiederum am besten an einem Beispiel deutlich. Dieses Demonstrationsprogramm 'DEMO' soll auf auf dem Bildschirm in sechs Zeilen zu je 16 Zeichen den ASCII-Zeichensatz ausgeben, was folgendermaßen aussieht, wenn es funktioniert (vgl. auch "Tabelle der ASCII-Codes" auf Blatt 18):

```
  ! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
§ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z Ä Ö Ü ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z ä ö ü ß
```

Ein Programm, das dies übernimmt, sieht folgendermaßen aus (s.u.); Es befindet sich als **Quell-Listing ab Adresse 9000h im RAM**, wo es (in der Regel) mit Hilfe des Editors erstellt worden ist. Bei der Programmeingabe und bei der Korrektur etwaiger Tippfehler hat der Editor Hilfsdienste geleistet (er hat im Prinzip den Computer zu einer komfortablen Schreibmaschine gemacht, bei der man überall Zeichen oder Zeilen einfügen, löschen oder ergänzen kann). Beim Aufruf des Assemblers wird das **Quellprogramm** als **geschlossener**, quasi "eingefrorener" Block übernommen, an dem inhaltlich (vom Assembler) nichts mehr geändert werden kann. Diesen **"Eisblock"** von Daten nimmt sich der Assembler dann in den einzelnen Durchläufen vor und meckert eventuell vorhandene Formfehler an (z.B. ein fehlendes "H" nach einer Hexadezimalzahl). Die anschließende **Korrektur eines Fehlers** ist aber nicht mehr Sache des Assemblers, sondern dazu muß wieder der **Editor her**, der seinerseits mit dem Assemblieren nichts zu tun hat (der Editor wird vom Assembler aus per E-Anweisung aufgerufen).

```
0001  *       Demo-Programm Zeichendarstellung
0002  *
0003  org     2800h           *Beginn CPU-RAM
0004  ofs     3800h           *Versatz F000 ./.2800
0005  *
0006  movid   equ   101eh      *Vid.Monitor-Anfang
0007  hordsp  equ   37b6h      *Vid.Monitor:Zeichen pro Zeile
0008  *
0009  demo:   lxi   h,30f0h   *Video-RAM,3.Zeile links
0010          mvi   a,20h     *erster ASCII-Code=Blank
0011          mvi   b,6h      *Zeilenzähler auf 6 setzen
0012  newlin: mvi   c,10h     *Zeichenzähler auf 16 setzen
0013  chrout: mov   m,a       *ASCII-Code ins Video-RAM
0014          inx   h         *RAM-Pointer erhöhen
0015          mvi   m,20h     *Blank als Trennzeichen
0016          inx   h         *Pointer erneut erhöhen
0017          inr   a         *ASCII-Code hochzählen
0018          ani   7fh       *MSB löschen
0019          dcr   c         *Zeichenzähler erniedrigen
0020          jnz   chrout    *nicht 0: das nächste Zeichen
0021          dcr   b         *Zeilenzähler erniedrigen
0022          jz    ready     *alle Zeilen fertig: JMP
0023          push  psw       *Zählerstand retten
0024          lda   hordsp    *Zeichen pro Bildzeile
0025          sui   20h       *32 Z.bereits ausgegeben
0026          mov   e,a       *nach Reg E
0027          mvi   d,0h      *Reg D löschen
0028          dad   d         *Pointer eine Bildzeile tiefer
0029          pop   psw       *ASCII-Zähler wieder holen
0030          jmp   newlin    *nächste Zeile ausgeben
0031  ready:  jmp   movid     *fertig: zum Video-Monitor
0032  *
0033  end                     *Ass.Lauf beenden (Pseudo-Op)
0034  *
```

4. Wichtige Kleinigkeiten am Rande

Wir nehmen an, daß das Programm ohne formelle Fehler im RAM ab 9000h vorliegt und vom Assembler bearbeitet werden kann. Zur Fehlerfreiheit sind zunächst folgende Dinge **von entscheidender Bedeutung:**

1. Der Assembler erwartet als **erstes Statement die ORG-Anweisung**, die angibt, bei welcher Startadresse das spätere Maschinenprogramm einmal beginnen soll; bezogen auf die hier genannte Adresse werden die folgenden Absolutadressen (z.B. für Sprungbefehle) ermittelt.

2. Dem Assembler muß mitgeteilt werden, wann er seinen Übersetzungsvorgang beenden soll; das geschieht mit dem **END-Statement**, das **als letztes im Quellprogramm** auftaucht (danach folgende Kommentarzeilen nicht gerechnet). Was nach dieser END-Zeile auch kommen mag, es interessiert den Assembler nicht mehr.

3. **Alle Zahlen** müssen durch ein **angehängtes "H"** (bzw."h") **oder "O"** (bzw."o") gekennzeichnet sein, um dem Assembler mitzuteilen, ob sie hexadezimal (zur Basiszahl 16) oder oktal (zur Basiszahl 8) aufzufassen sind. Das gilt auch für den Zahlenwert 0 (Null). Alle mit "A"..."F" beginnenden **Hexadezimalzahlen** müssen eine **führende Null** bekommen, weil der Assembler sonst denkt, daß es sich um ein (mit einem Buchstaben beginnendes) Label handelt.

4. **Label** sind durch einen **angehängten Doppelpunkt** zu kennzeichnen. Sie dürfen maximal **6 Zeichen** lang sein, müssen aber **mit** einem **Buchstaben beginnen.** Fehlt der Doppelpunkt, kreidet Ihnen das der Assembler nicht als Fehler an, weil er nicht wissen kann, daß Sie ein Label definieren wollen; die Zeile wird in diesem Fall wie eine Zuweisung (per EQU) behandelt.-

5. Alles, was in einer Zeile hinter dem **Kommentarzeichen "*"** steht, dient nur zur Erläuterung für den Programmierer, den Assembler dagegen interessiert das nicht (auf diese Weise können Sie ganze Zeilen mit erklärendem Text einfügen).

6. Sie haben bei der Betrachtung bemerkt, daß das vom **Editor** erstellte Quellprogramm (vgl. Blatt 6) in **Kleinschreibung** vorliegt; das ist dem **Assembler** egal (d.h. Sie können wahlweise klein oder groß schreiben), er **setzt die Assembler-Codes in Großbuchstaben um** (der Kommentar bleibt unverändert).

7. Das im Beispielprogramm eingehaltene **Spaltenformat** für das Quellprogramm wirkt **sehr ordentlich** und aufgeräumt; diese Eingabe erleichtert Ihnen der Editor durch bloßen Druck auf die Tabulator-Taste. Dennoch ist dieses Format für die Arbeit des Assemblers keineswegs erforderlich, im Gegenteil: Jeder Leerschritt vom Zeilenbeginn bis zum eigentlichen Befehl belegt im RAM ein eigenes Byte. Merke: Schönes Aussehen **kostet** etwas, auch wenn es nur **Speicherplatz** ist!

5. Ein Ohr stets an der Tastatur

Bei allen Listings, die Ihr Assembler ausgibt (und deren Ausgabe manchmal schon ganz schön zeitaufwendig sein kann), wird permanent die Tastatur abgefragt; sobald auch nur eine einzige Taste gedrückt wird, stoppt jeder List-Vorgang, egal, ob der Drucker parallel läuft oder nicht **(STOP-Funktion)**. Wenn Sie gleichzeitig die Tasten 'CTL' (Control) und 'C' betätigen, bricht der Assembler sein Listen ganz ab **(BREAK-Funktion)**. Sollten Sie zu irgendeinem Zeitpunkt nicht mehr wissen, welche Eingaben Ihr Assembler akzeptiert, dann drücken Sie auch in diesem Fall CTL+C gleichzeitig, und schon erscheint das Menü auf dem Bildschirm, und Sie können sich eine Anweisung Ihrer Wahl heraussuchen **(HELP-Funktion)**.

5.1 Pass 1 (Symboltabelle anlegen)

Nach der Eingabe einer "1", gefolgt von Return, führt der Assembler den **Pass 1** aus (zu deutsch 'Lauf' oder auch 'Durchlauf'); dabei macht er nichts anderes, als das Quellprogramm auf **vorhandene Label durchzuforsten** und diese (zusammen mit ihren Absolutadressen) **in** die **Symboltabelle** (ab 8000h) einzutragen. Bezogen auf unser Beispiel heißt das folgendes: Das erste Label 'MOVID' (dem per 'EQU' die Absolutadresse 101Eh im Video-Monitor zugewiesen wird), kommt als Folge von ASCII-Zeichen in die Symboltabelle ab 8000h, abgeschlossen durch ein 00-Byte; danach folgt in zwei Bytes die zu diesem Label gehörende Adresse 101Eh (im RAM als '1Eh' abgelegt, gefolgt von '10h'), und danach kommt (ohne weiteres Trennzeichen) die ASCII-Zeichenfolge für das nächste auftauchende Label 'HORDSP' (mit nachfolgendem Null-Byte und Adresse 37B6h). Wenn Sie wollen, können Sie sich diese Symboltabelle ansehen, indem Sie sich den RAM-Inhalt ab 8000h listen lassen; Sie finden dann nacheinander die ASCII-Codes für "M" (=4Dh; vgl. auch Tabelle der ASCII-Codes auf Blatt 18), "O" (=4Fh), "V" (56h), "I" (=49h) und "D" (=44h), dann das Byte "00" und die erste, zu 'MOVID' gehörende Adresse 101Eh:

```
8000  4D 4F 56 49  44 00 1E 10   48 4F 52 44  53 50 00 B6
8010  37 44 45 4D  4F 00 00 28   4E 45 57 4C  49 4E 00 07
8020  28 43 48 52  4F 55 54 00   09 28 52 45  41 44 59 00
8030  27 28 00 00  00 00 00 00   00 00 00 00  00 00 00 00
```

Die ersten beiden Label bekommen per **EQU-Anweisung** feste Adressen zugewiesen, da sie nicht im Bereich des hier assemblierten Programms liegen. Erst für die folgenden Label errechnet der Assembler die jeweiligen Absolutadressen, bezogen auf die per **ORG-Anweisung** festgelegte Startadresse. Wenn man weiß, bei welcher Anfangsadresse das Maschinenprogramm beginnt und wieviele Bytes jeder Befehl belegt, dann kann man die zu jedem Label gehörende Adresse durch simples Abzählen ermitteln; und nichts weiter tut Ihr Assembler und merkt sich die Ergebnisse, wie gesagt, in seiner Symboltabelle.

```
>1

>S

MOVID  101E   HORDSP 37B6   DEMO   2800
NEWLIN 2807   CHROUT 2809   READY  2827
```

5.2 Pass 0 (Symboltabelle erweitern)

Nach Eingabe einer "0" (Null), gefolgt von Return, verfährt der Assembler wie im Pass 1, nur mit dem Unterschied, daß die Label mit ihren zugehörigen Adressen nicht ab 8000h abgelegt, sondern an eine bereits vorhandene Symboltabelle angehängt werden (die von einem vorher assemblierten Programm stammt). Auf diese Weise kann ein Programm auf die Label eines getrennt umgesetzten, anderen Programms zugreifen und diese mitbenutzen. Während **Pass 1** eine eventuell vorhandene Symboltabelle **löscht** (überschreibt), **hängt sich Pass 0 daran an**; dies ist der einzige Unterschied zwischen den beiden Durchläufen.

5.3 Symboltabelle listen

Durch Eingabe von "S" (bzw."s"), gefolgt von Return, kann man die **Ausgabe der gesamten Symboltabelle** (Label mit zugehörigen Adressen) veranlassen (auf Wunsch mit parallelem Ausdruck; s.o. "P-Anweisung").

5.4 Pass 2 (Objektcode erzeugen)

Nach Eingabe einer "2", gefolgt von Return, beginnt die **Übersetzung** des Quellprogramms **in den Maschinencode**, den der Assembler **im RAM ab F000h ablegt** (Ausnahme: OFS-Anweisung, siehe Blatt 12). Sofern vorher noch keine Symboltabelle angelegt worden ist (im Pass 1 bzw. Pass 0), holt der Assembler dies hier nach. Nach getaner Arbeit (die sich in Sekundenbruchteilen vollzieht) meldet sich der Assembler mit 'o.k.' zurück und sagt Ihnen gleichzeitig, bis zu welcher Endadresse der Maschinencode reicht (die Anfangsadresse haben Sie ja mit Ihrem ORG-Statement festgelegt):

```
>2 o.k.; Code End: 2829
```

Wenn Sie dieses Maschinenprogramm nun in ein EPROM laden wollen, kennen Sie dazu die betreffende Anfangs- und Endadresse Ihres Programms.

```
>1

>S

MOVID  101E   HORDSP 37B6   DEMO   2800
NEWLIN 2807   CHROUT 2809   READY  2827


>2 o.k.; Code End: 2829
```

5.5 Pass 3 (Quell- und Objektprogramm listen)

Nach dem Austesten eines Programms ist es nicht nur wünschenswert, sondern eine unumgängliche Pflicht, alles auf Papier (und im EPROM oder auf Diskette) festzuhalten. Nach der Eingabe einer "3", gefolgt von Return, erzeugt Ihr Assembler ein **komplettes Listing** von (dezimaler) **Zeilennummer, Maschinencode und Quellprogramm** (einschließlich Label), so daß Sie auf einen Blick Ihr Programm nebst ergänzendem (von Ihnen hinzugefügten!) Kommentar sehen. Sollte vorher noch keine Symboltabelle angelegt worden sein (im Pass 1 bzw. Pass 0), holt der Assembler dies hier selbsttätig nach.-

Das Listen stoppen Sie durch Druck auf eine beliebige Taste; beim gleichzeitigen Betätigen von 'CTL+C' brechen Sie den List-Vorgang ab:

```
>3

0001 0000                    *       Demo-Programm Zeichendarstellung
0002 0000                    *
0003 0000                    ORG     2800H           *Beginn CPU-RAM
0004 2800                    OFS     3800H           *Versatz F000 ./.2800
0005 2800                    *
0006 2800                    MOVID   EQU   101EH      *Vid.Monitor-Anfang
0007 2800                    HORDSP  EQU   37B6H      *Vid.Monitor:Zeichen pr
0008 2800                    *
0009 2800  21 F0 30          DEMO:   LXI   H,30F0H    *Video-RAM,3.Zeile link
0010 2803  3E 20                     MVI   A,20H      *erster ASCII-Code=Blan
0011 2805  06 06                     MVI   B,6H       *Zeilenzähler auf 6 set
0012 2807  0E 10             NEWLIN: MVI   C,10H      *Zeichenzähler auf 16 s
0013 2809  77                CHROUT: MOV   M,A        *ASCII-Code ins Video-R
0014 280A  23                        INX   H          *RAM-Pointer erhöhen
0015 280B  36 20                     MVI   M,20H      *Blank als Trennzeichen
0016 280D  23                        INX   H          *Pointer erneut erhöhen
0017 280E  3C                        INR   A          *ASCII-Code hochzählen
0018 280F  E6 7F                     ANI   7FH        *MSB löschen
0019 2811  0D                        DCR   C          *Zeichenzähler erniedri
0020 2812  C2 09 28                  JNZ   CHROUT     *nicht 0: das nächste Z
0021 2815  05                        DCR   B          *Zeilenzähler erniedrig
0022 2816  CA 27 28                  JZ    READY      *alle Zeilen fertig: JM
0023 2819  F5                        PUSH  PSW        *Zählerstand retten
0024 281A  3A B6 37                   LDA   HORDSP    *Zeichen pro Bildzeile
0025 281D  D6 20                      SUI   20H       *32 Z.bereits ausgegeber
0026 281F  5F                         MOV   E,A       *nach Reg E
0027 2820  16 00                      MVI   D,0H      *Reg D löschen
0028 2822  19                         DAD   D         *Pointer eine Bildzeile
0029 2823  F1                        POP   PSW        *ASCII-Zähler wieder ho
0030 2824  C3 07 28                  JMP   NEWLIN     *nächste Zeile ausgeben
0031 2827  C3 1E 10          READY:  JMP   MOVID      *fertig: zum Video-Moni
0032 282A                    *
0033 282A                    END                      *Ass.Lauf beenden (Pseu
```

Im Listing eben erscheinen alle Speicheradressen und -Inhalte in hexadezimaler Schreibweise; nach Eingabe eines "O" (bzw."o"), gefolgt von Return, schaltet der Assembler um auf **oktale Darstellungsform.** Diese behält er so lange bei, bis Sie dies per H-Eingabe wieder rückgängig machen. Nach dem Assembler-Aufruf ist automatisch der HEX-Mode eingestellt. Wenn Sie das Pass-3-Listing einmal in oktaler Form sehen wollen - bitte schön, hier ist es:

```
>O

>3

0001 000000                    *       Demo-Programm Zeichendarstellung
0002 000000                    *
0003 000000                    ORG     2800H           *Beginn CPU-RAM
0004 050000                    OFS     3800H           *Versatz F000 ./.2800
0005 050000                    *
0006 050000                    MOVID   EQU   101EH      *Vid.Monitor-Anfang
0007 050000                    HORDSP  EQU   37B6H      *Vid.Monitor:Zeichen pr
0008 050000                    *
0009 050000  041 360 060 DEMO:  LXI   H,30F0H   *Video-RAM,3.Zeile link
0010 050003  076 040            MVI   A,20H     *erster ASCII-Code=Blan
0011 050005  006 006            MVI   B,6H      *Zeilenzähler auf 6 set
0012 050007  016 020      NEWLIN:MVI   C,10H     *Zeichenzähler auf 16 s
0013 050011  167          CHROUT:MOV   M,A      *ASCII-Code ins Video-R
0014 050012  043                 INX   H        *RAM-Pointer erhöhen
0015 050013  066 040             MVI   M,20H    *Blank als Trennzeichen
0016 050015  043                 INX   H        *Pointer erneut erhöhen
0017 050016  074                 INR   A        *ASCII-Code hochzählen
0018 050017  346 177             ANI   7FH      *MSB löschen
0019 050021  015                 DCR   C        *Zeichenzähler erniedri
0020 050022  302 011 050         JNZ   CHROUT   *nicht 0: das nächste Z
0021 050025  005                 DCR   B        *Zeilenzähler erniedrig
0022 050026  312 047 050         JZ    READY    *alle Zeilen fertig: JM
0023 050031  365                 PUSH  PSW      *Zählerstand retten
0024 050032  072 266 067         LDA   HORDSP   *Zeichen pro Bildzeile
0025 050035  326 040             SUI   20H      *32 Z.bereits ausgegebe
0026 050037  137                 MOV   E,A      *nach Reg E
0027 050040  026 000             MVI   D,0H     *Reg D löschen
0028 050042  031                 DAD   D        *Pointer eine Bildzeile
0029 050043  361                 POP   PSW      *ASCII-Zähler wieder ho
0030 050044  303 007 050         JMP   NEWLIN   *nächste Zeile ausgeben
0031 050047  303 036 020 READY:  JMP   MOVID    *fertig: zum Video-Moni
0032 050052                    *
0033 050052                    END             *Ass.Lauf beenden (Pseu
```

Es ist nicht Aufgabe dieser Beschreibung, die Grundlagen verschiedener Zahlensysteme aufzurollen. Aber hinter dem etwas aufgeblähten Zahlensalat der oktalen Darstellung verbergen sich wiederum nur Bitkolonnen aus Nullen und Einsen. Fassen Sie es als Denksportaufgabe auf, ein oktal wiedergegebenes Wort zu entschlüsseln! Das Ergebnis muß natürlich dasselbe sein wie bei hexadezimaler Schreibweise.

6. Pseudo-Operatoren (Assembler-Anweisungen im Quellprogramm)

Alle nachfolgend aufgeführten Anweisungen sind sogenannte **Pseudo-Operatoren**, die im Quellprogramm auftauchen, und die den Assembler zu bestimmten Reaktionen veranlassen, ohne im späteren Maschinenprogramm in irgendeiner Form wieder zu erscheinen.

6.1 Anfang und Ende definieren

Die beiden Anweisungen **'ORG'** und **'END'** kennen Sie bereits als einrahmende Statements zur Festlegung von späterer Anfangsadresse und Kennzeichen für das Ende des Quellprogramms; beim Erreichen von 'END' setzt sich der Assembler zur Ruhe, alles (im Quell-Listing) darauf Folgende kümmert ihn nicht mehr.

6.2 Gleichheit anordnen

Erwähnt worden ist ferner die **EQU-Anweisung** (vom engl. 'Equal' für 'Gleichsetzen'); wenn man einem Label einen festen Wert zuweisen will (eine Adresse oder eine 16-Bit-Konstante), benutzt man dieses Statement. Im Beispielprogramm dient es dazu, dem Assembler zwei feste Adressen zu übergeben, von denen er beim besten Willen nicht wissen kann, woher er sie sonst nehmen soll: 'MOVID' ist der Einsprung-Punkt für den Video-Monitor, bei dem der MOPPEL seine Bereitmeldung ausgibt ('Anweisung an MOPPEL >'); die zugehörige Adresse geht aus der Symboltabelle zum Video-Monitor (EPROM gelb) hervor. Und ähnliches gilt für die Speicheradresse 'HORDSP' (vom engl. 'Horizontal Display'), in der die Anzahl der pro Bildzeile dargestellten Zeichen abgelegt ist (40d Zeichen beim kleinen Bildformat, 80d Zeichen beim großen); auch die Absolutadresse hierfür finden Sie in der Symboltabelle zum gelben Monitor. Der EQU-Operator hört es nicht gern, aber Sie müssen es schließlich wissen: 'EQU' ist jederzeit entbehrlich, d.h. dem Assembler ist es egal, ob Sie schreiben 'MOVID EQU 101Eh' oder ob Sie auf das 'EQU' verzichten: 'MOVID 101Eh'; in beiden Fällen führt er die Wertzuweisung der genannten Adresse 101Eh zum Label 'MOVID' durch. Warum es 'EQU' dann überhaupt gibt? In erster Linie aus Gründen der Ordentlichkeit; man hat als Programmierer ein besseres Gefühl, wenn man die Zuweisung ausdrücklich betont (durch Einfügen von 'EQU'). Fachmännisch heißt das "explizit" benennen, im Gegensatz zu "implizit", was im Fall von 'EQU' eben auch ginge.

6.3 Versetzung nie gefährdet

Bereits mehrfach angeklungen ist hier die **OFS-Anweisung**, die, sofern sie eingebaut wird, stets hinter der jeweiligen ORG-Anweisung stehen muß (ansonsten bleibt 'OFS' wirkungslos, da 'ORG' immer für OFS=0 sorgt). 'OFS' steht für 'Offset' (soviel wie 'Versatz', gebräuchlich ist hierfür auch 'Displacement'); hinter 'OFS' steht eine Hexadezimalzahl (bzw. Oktalzahl, falls gewünscht), die angibt, um welchen **Versatz** (bezogen auf F000h) der Maschinencode beim Pass 2 abgelegt werden soll. Sie erinnern sich: Im **Pass 2** erzeugt der Assembler aus dem Quell-Listing den **Maschinencode**, den er ab F000h im RAM ablegt. Mit Hilfe von 'OFS' können Sie diesen Transfer **in** einen **anderen Zielbereich** umleiten. Beispiel: Das Beispielprogramm auf Blatt 10 soll im RAM-Bereich

ab 2800h laufen - warum soll es der Assembler nicht gleich dorthin transportieren und nicht, wie es sonst seine Art ist, nach F000h? Dann geben Sie einfach vor 'OFS 3800h', und statt nach F000h bringt der Assembler den Maschinencode um 3800h versetzt ins RAM - und das ist gerade der 2800er-Bereich (F000h + Versatz 3800h = 2800h)! Auf diese Weise können Sie sich auch weiterhelfen, wenn bei F000h gar kein RAM eingesetzt ist; Sie veranlassen dann einfach die Ablage des Maschinencodes in einen anderen, vorhandenen Bereich!

6.4 Bytes definieren und reservieren

Beim Programmieren kann es vorkommen, daß man im Maschinenprogramm bestimmte Bytes mit einem festen Bitmuster benötigt, etwa zur Verarbeitung von Konstanten. Um ein einzelnes Byte als eine solche **Konstante** zu **definieren**, bedient man sich des **DB-Operators** (vom engl.'Define Byte'); wenn im Quellprogramm die Zeile auftaucht 'DB EQU 1Eh', dann wird an dieser Stelle im späteren Maschinenprogramm das HEX-Byte '1Eh' eingefügt, unabhängig davon, was davor oder dahinter an Befehlen auch stehen mag. Die DB-Anweisung läßt sich auch in folgender Form verwenden, um einen **ASCII-Code als Datenbyte zu definieren: DB EQU "a"** sorgt dafür, daß anstelle des in Anführungszeichen (ASCII-Marker) stehenden ASCII-Zeichens der äquvalente ASCII-Code (=61h) ins spätere Maschinenprogramm eingefügt wird; der Assembler nimmt Ihnen hierbei die Arbeit des Code-Heraussuchens ab.

Ähnlich verfahren Sie beim Einbau von **Doppelwort-Konstanten** (Definition von zwei aufeinander folgenden Datenbytes). Dabei greifen Sie zur **DW-Anweisung:** 'DW EQU 101Eh' sorgt dafür, daß ins spätere Maschinenprogramm nacheinander die beiden Datenbytes '1Eh' und '10h' eingefügt werden (das niederwertige '1Eh' zuerst, wie beim 8085 generell üblich). Hiermit können Sie, beispielsweise in Sprungtabellen, reihenweise Festadressen in Ihr Programm einbauen.

Wem ein oder zwei Datenbytes in Folge noch nicht ausreichen, der greift zum **DS-Operator** (vom engl. 'Define Storage'); mit 'DS 20h' beispielsweise veranlassen Sie, daß der Assembler im späteren Maschinenprogramm 32 Bytes (= hexadezimal 20h) freiläßt zu Ihrer (oder jedermanns) Verfügung. Dahinter steht folgender Sinn: In vielen Programmen legt man sich einen RAM-Bereich an, in dem Werte zwischengespeichert werden, etwa Zählerstände oder Eingaben bzw. Zwischenergebnisse. Um hier einen **längeren Bereich aufeinander folgender Bytes** freizuschaufeln, bedient man sich der DS-Anweisung.

7. Es gibt auch gehörige Schelte

Der **Assembler** ist ein **übersetzerprogramm**, das sich seines Wertes wohl bewußt ist - es ist **pingelig bis zum Letzten** und läßt in keiner Weise mit sich handeln; für Sie als Programmierer bedeutet das, daß Ihr Quellprogramm bis auf's I-Tüpfelchen genau den Erwartungen Ihres Assemblers entsprechen muß. Andernfalls gibt es Schelte vom Assembler, die in Form von Fehlermeldungen über Sie hereinbricht (was im Wiederholungsfall schon ganz schön nerven kann!): 'Error # Axx in Line 1234' heißt es da, und das "A" vor der Fehlernummer erinnert Sie daran, daß es Komplikationen im Assembler gibt (im Gegensatz zu den Editor-Fehler-Codes, die mit "E" beginnen). Die angegebene Zeilennummer verweist Sie auf die fehlerträchtige Zeile, die Sie (nach dem Rücksprung in den Editor) sofort inspizieren und korrigieren sollten.-

Was der Assembler bemängeln kann (und dies auch mit Akribie tut!), das sind **Formfehler** im Quell-Listing, d.h. falsche Schreibweise der Befehle, Doppeldefinition eines Labels o.ä. **Kein** Programm der Welt kann **Denkfehler** in Ihrem Programm **aufspüren**, die aufgrund falscher Strukturen zum Nichtfunktionieren führen. Ehe Sie diesen Funktionstest eines Programms durchführen können, müssen Sie also erst einmal die richtige Form bei der Programmeingabe wählen!

Error # A00 in Line 0000

Hoppla, die von Ihnen eingegebene **Anweisung kennt der Assembler gar nicht** ("Line 0000" ist hier überflüssig)! Was er kennt, entnehmen Sie bitte seinem **Menü**, das Sie **jederzeit abrufen** können, indem Sie CTL+C gleichzeitig drücken (HELP-Funktion, s.o.): Es sind die Ziffern 0,1,2 und 3 für die Durchläufe 1...3 sowie die Buchstaben D,E,H,I,M,O,P und S für die weiter oben beschriebenen Funktionen - andere Eingaben quittiert Ihnen Ihr Assembler mit dieser Fehlermeldung.

Error # A01 in Line xxyy

Da haben Sie ja ganz schön gehudelt! Sie tun so, als ob Sie ein **Label** definieren wollen (durch Setzen eines Doppelpunktes), und dann **vergessen** Sie, es hinzuschreiben! Dem Assembler fehlt in diesem Fall ganz einfach das zum Doppelpunkt gehörende Label.

Error # A02 in Line xxyy

Es ist so weit: Sie haben so viele symbolische Namen (Label) in Ihrem Programm verwendet, daß die **Symboltabelle überläuft**; wie Sie wissen, legt der Assembler diese Symboltabelle ab 8000h im RAM an. Haben Sie im 8000er-Bereich nur ein RAM bestückt (auf Platz #80 der großen Speicherkarte)? Dann hilft es Ihnen weiter, ein zweites RAM für die Symboltabelle zu spendieren (und auf Platz #88 einzusetzen). Im anderen Fall müssen sie Ihr Programm teilen oder Label abspecken (d.h. überflüssige, die keine Sprungziele sind, wieder entfernen).

Error # A03 in Line xxyy

Da ist Ihnen ein peinlicher Fehler unterlaufen: **Ein symbolischer Name wird zweimal definiert**, und der Assembler kann sich für keinen entscheiden! Taufen Sie das Label in einer der beiden Zeilen um, und die Welt ist wieder in Ordnung. **Achtung!** Wenn Sie in Ihrem Quellprogramm (mit Hilfe des Editors) Änderungen vorgenommen haben, sollten Sie nach der Rückkehr in den Assembler unbedingt erst den Pass 1 durchlaufen, um die Symboltabelle auf Vordermann zu bringen; sonst kann es passieren, daß eine vorher angelegte Symboltabelle mit dem inzwischen aktualisierten Quellprogramm nicht mehr korrespondiert und der Fehler "03" ausgeworfen wird!

Error # A04 in Line xxyy

Flüchtigkeitsfehler! Vor dem Kommentar fehlt das obligate "*" oder bei einer Zahl fehlt z.B. die Angabe, ob Sie sie hexadezimal (angehängtes "H") oder oktal (angehängtes "O") verstanden wissen wollen; bitte in Ordnung bringen! Übrigens: Konsequenterweise muß auch die Null ein "H" oder "O" nachschleppen; hier ein fehlendes "H" oder "O" zu tolerieren, hieße erhöhten Programmaufwand für das Assembler-Programm treiben, und der läßt sich im hautengen 2732er-EPROM leider nicht mehr unterbringen!

Dasselbe gilt für die Überprüfung auf andere Schreibfehler; hier mußte der Aufwand wiederum auf ein vernünftiges Maß beschränkt werden, so daß beispielsweise Fehler wie 'LXI C,134h' zu einer unsinnigen Übersetzung führen. Da es den Code 'LXI' ebenso gibt wie die Registerangabe 'C', reimt sich der Assembler seinen Vers daraus zusammen; daß die Kombination von 'LXI' plus 'C' nicht zulässig ist, übersteigt dabei seinen Horizont!

Error # A05 in Line xxyy

In dieser Zeile steckt der Wurm drin! **Eins der Felder** (Label, Assembler-Code oder Kommentar) ist vollkommen verkorkst und damit für den Assembler **nicht zu identifizieren.** Dieser Fehler liegt beispielsweise vor, wenn Sie im Feld für Label ausschließlich andere als die erlaubten alphanumerischen Zeichen verwenden oder wenn Sie im Quell-Listing eine Zeile eingebaut haben, die beim besten Willen weder als Kommentar noch als Assembler-Code eines Befehls zu deuten ist.

Error # A06 in Line xxyy

Sie haben sich bei der **Schreibweise eines Befehls** vertan, z.B. eine Registerangabe vergessen oder gar ein Komma zwischen der Nennung zweier Register. Das läßt sich aber schnell und ohne viel Aufwand wieder beheben!

Error #07 in Line xxyy

Da haben Sie doch tatsächlich vergessen, daß ein **Label** mit einem **Buchstaben anzufangen** hat! Im vorliegenden Fall aber beginnt eins mit irgendeinem anderen Zeichen als den 26 unseres Alphabets!

Error # A08 in Line 0000

Soso, Sie wollen also Ihren **Drucker** parallelschalten, und der ist entweder gar nicht angeschlossen oder nicht eingeschaltet oder aus sonstigen Gründen **nicht betriebsbereit.** Sie wissen doch: Der an den MOPPEL anzuschließende Drucker muß über eine serielle Schnittstelle verfügen und auf eine Übertragungsrate von 4800 Baud eingestellt sein; der Anschluß am MOPPEL erfolgt über die 25polige, rückwärtige Buchsenleiste am Bus, und zur Ansteuerung muß das Serielle Interface eingesetzt sein. Wenn Sie das alles sichergestellt haben, versuchen Sie das P-Kommando erneut, diesmal bleibt die Fehlermeldung "08" dann aus!

8. Der Disassembler

Vom Assembler aus erreichen Sie den (im Assembler-EPROM mit untergebrachten) Disassembler über die **D-Anweisung**, auf die unmittelbar die entsprechende Bereitmeldung folgt (s.u.); Sie brauchen nur noch die **Start- und Endadresse** des Speicherbereichs **anzugeben** (durch Komma trennen!), den Sie rückübersetzen wollen, und schon bekommen Sie die Befehle im bekannten mnemotechnischen Format gelistet (Speicheradresse plus Befehlswort und Assembler-Code). Der Disassembler übernimmt übrigens die Einstellung des Drucker-Flipflops vom Assembler, d.h. wenn Sie dort den Paralleldruck eingeschaltet haben, werden auch im Disassembler sämtliche Ausgaben auf dem Drucker protokolliert. Das **Listen** können Sie jederzeit durch Betätigen einer beliebigen Taste **anhalten**; Sie **brechen den List-Vorgang ab**, indem Sie gleichzeitig **'CTL+C'** betätigen. Ein erneutes 'CTL+C' bringt Sie zurück zum Assembler-Menü, von wo aus Sie entweder weiter assemblieren oder ganz zurück in die Monitor-Anweisungs-Ebene springen können (M-Anweisung).

Im Beispiel unten hat sich der **Disassembler selbst disassembliert** (vergleichbar mit einem Doktor, der sich selbst untersucht); im ersten Befehl in 6C00h wird der Textpointer H&L auf den Anfang der Disassembler-Bereitmeldung gesetzt; der folgende Unterprogramm-Aufruf veranlaßt die Ausgabe des Textes (bis im Text das End-Kennzeichen '00' auftaucht), und das bei 1036h beginnende Unterprogramm stellt sicher, daß der Cursor eingeschaltet ist und artig blinkt...

```
MOPPEL-Disassembler V 9.0
Copyright hms'84

Start,End >6c00,6c15

          6C00   21   LXI  H,6E12
          6C03   CD   CALL 6DF1
          6C06   CD   CALL 1036
          6C09   CD   CALL 6006
          6C0C   FE   CPI  1B
          6C0E   C2   JNZ  6C17
          6C11   CD   CALL 1036
          6C14   C3   JMP  6000
```

Auch der Disassembler läßt sich nicht alles gefallen; bei fehlerhaften Adreß-Eingaben beschwert er sich mit **'Input Error'**, und unbekannte 8085-Codes quittiert er mit einem **Fragezeichen:**

```
Start,End >2800,28xx

Input Error


Start,End >2800,2807

          2800   21   LXI  H,30F0
          2803   10   ?10
          2804   18   ?18
          2805   20   RIM
          2806   28   ?28
          2807   30   SIM
```

9. Tabelle der ASCII-Codes

Aus der untenstehenden Tabelle geht der Zusammenhang zwischen ASCII-Zeichen und zugehöriger Codierung hervor. 'ASCII' kommt vom amerikanischen 'American Standard Code For Information Interchange' und bedeutet soviel wie 'Amerikanischer Standard-Code für den Informationsaustausch'; das spricht man 'Askie' und nicht etwa 'Ah - Es - Zeh - Zwo', wie es immer wieder zu hören ist!! - Um beispielsweise den ASCII-Code für das "d" herauszusuchen, gehen Sie in die 6xer-Zeile (d.h. der betreffende Code beginnt mit einer "6") und in die x4er-Spalte: Demnach hat das "d" den ASCII-Code '64h'.

Die in den ersten beiden Zeilen stehenden Zeichen sind Steuercodes, deren Erläuterung Sie unten finden; bis auf den Leerschritt (das Blank) dienen diese Zeichen ausschließlich zur Verwaltung der Peripheriegeräte (z.B. Papier um eine Zeile hochschieben), ohne daß diese Zeichen anderweitig ausgegeben werden (z.B. auf dem Bildschirm oder Drucker).

Der Standard-ASCII-Zeichensatz umfaßt (einschließlich Steuercodes) 128 Zeichen, die durch sieben Bits repräsentiert werden (ASCII-Codes '00...7Fh'); das höchstwertige Bit (MSB vom engl.'Most Significant Bit') ist hierbei immer auf Null.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x								BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x		DC1	DC2	DC3	DC4	NAK						ESC				
2x	<O>	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	§	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	Ä	Ö	Ü	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	ä	ö	ü	ß	DEL

<O>: Leerschritt (Blank)
BEL: Akustisches Signal (Bell)
BS: Rückwärtsschritt (Back Space)
HT: Horizontaler Tabulator (Horizontal Tabulation)
LF: Zeilenvorschub (Line Feed)
VT: Vertikaler Tabulator (Vertical Tabulation)
FF: Seitenvorschub (Form Feed)
CR: Wagenrücklauf (Carriage Return)
SO: Shift Out
SI: Shift In
DC1: Device Control #1
DC2: Device Control #2
DC3: Device Control #3
DC4: Device Control #4
NAK: Negative Acknowledge
ESC: Escape Prefix
DEL: Zeichen löschen (Delete)