1. Einführung

Dieses Handbuch wurde als Benutzerhandbuch zum Single-Board-Computer SBC-86 sowie seiner Softwarekomponenten verfaßt. Es trägt daher nicht den Charakter eines Lehrbuches zu Hard- und Softwaretechnologien für den I8086. Der Leser sollte deshalb Grundwissen aus entsprechenden Fachbüchern erlangen, um zufriedenstellend mit dem SBC-86 umgehen zu können. Der Einsatz des SBC-86 als Hilfsmittel beim Studium solcher Standardliteratur zum Gebiet der Mikroprozessortechnik wird sicherlich zum leichteren Verständnis beitragen. Ausgewählte Literaturstellen sind im Literaturverzeichnis einzusehen.

Am Anfang jedoch etwas Geschichtliches. Der SBC-86 entstand aus der Idee, die bis dahin eher theoretisch betriebene Lehre zur Mikroprozessortechnik an der Ingenieurhochschule Mittweida etwas aufzulockern und anschaulicher zu gestalten. Erste Gedanken sollten vorhandene Technik nutzbar machen. Dazu wurden IBM-kompatible Rechner zur Lehre vergewaltigt. Das Resultat war, daß Programmierfehler, die in der Lehre nicht auszuschließen, ja sogar wünschenswert sind, in der Regel zu kompletten Systemabstürzen führten und die Lernenden dauernd mit dem Systemneustart beschäftigt waren. Außerdem kommt hinzu, daß hardwarenahe Programmierung, also Programmierung von Peripheriecontrollern, nicht oder nur teilweise realisiert werden konnte, da vom System bereits ein Großteil aller Hardwarekomponenten verplant war. Deshalb mußte diese Methode als unzufriedenstellend eingestuft werden und nach anderen Lehrmethoden gesucht werden. Bei der Suche nach einem geeigneten Lehrprozessor kam man zum Schluß, daß am besten ein fiktiver Prozessor zu entwickeln wäre, der einen minimierten Befehls- sowie Registersatz beinhaltet und der mit Hilfe eines Emulators auf vorhandener Technik programmierbar wäre. Dieses Konzept wurde ebenfalls wieder verworfen, da die Emulation von Peripheriecontrollern zu ähnlichen Problemen wie beim ersten Konzept geführt hätte. Deshalb wurde einem realen Prozessor der Vorzug gegeben, einem Prozessor, der auf einer Vielzahl von vorhandenen Computern eingesetzt wird und der auch in Neuentwicklungen noch als Teilmenge enthalten ist. Dieser Prozessor ist Intel's 8086. Dazu wurde der vorliegende Einplatinenrechner unter der Maßgabe entwickelt, ein Lern- und Lehrsystem zur Verfügung stellen zu können, das transparent ist, also Logikschaltkreise nicht in Logikarrays versteckt sind. Ergebnis dessen war dann der SBC-86 in ziemlich der heutigen Ausstattung. Bei der Entwicklung des Betriebssystemmonitors für den SBC-86 wurden zuerst simple Kommunikationsroutinen geschrieben, die es ermöglichten, den SBC-86 so nach und nach mit Leben zu erfüllen. Dabei entstand fast nebenbei der Prototyp des im Paket enthaltenen Full-Screen-Debuggers FSDEB, damals noch zeilenorientiert. Der Debugger wurde also schon als Entwicklungstool bei der Systemmonitorerstellung eingesetzt. Grundgedanke bei der

Konzeption war, daß ein System zu schaffen ist, daß einerseits abgesetzt vom Muttersystem nutzbar ist und andererseits über eine serielle Schnittstelle vom Muttersystem mit Software versorgt werden kann. Die Programmtestung sollte in der Zielhardware (dem SBC-86) erfolgen, so daß Systemabstürze leicht durch Drücken des Resetknopfes zu korrigieren sind. Bislang haben schon viele Studenten in unzähligen Praktika den SBC-86 und das Softwarepaket auf Herz und Nieren testen können, so daß viele interne Fehlerchen entdeckt und beseitigt werden konnten. Dennoch kann es durchaus vorkommen, daß noch unentdeckte Bugs existieren, die aus der vorliegenden Version noch nicht entfernt wurden.

Das vorliegende Handbuch beschäftigt sich mit der Installation der im Lieferumfang enthaltenen Software, der Hardware des SBC-86 sowie den einzelnen Softwarekomponenten. Kleine Programmbeispiele sollen keineswegs als fertige Lehrbeispiele angesehen werden, vielmehr dienen sie zum Kennenlernen des SBC-86 und der Bedienung der Softwarekomponenten.

In Zukunft sind weitere Softwareergänzungen zum SBC-86 geplant, wie z.B. "Programmierung des SBC-86 unter Turbo-Pascal" und "Ein Echtzeit-Multitasking-Betriebssystem für den SBC-86 mit Turbo-Pascal-Schnittstelle". Entsprechende Informationen können Sie von der TEL-GmbH (Herr Westerholt) erhalten. Tel.-Nr. bzw. Anschrift finden Sie auf der 1. Umschlagseite dieses Handbuches.

Im folgenden Text werden mehrfach Handelsmarken, die eingetragene Warenzeichen darstellen, verwendet. Auf eine Auflistung soll jedoch an dieser Stelle verzichtet werden.

Die zum SBC-86 zugehörige Software kann frei kopiert und weitergegeben werden, da sie ohnehin ohne den SBC-86 nicht verwendbar ist. Kopien sind jedoch nur dann zulässig, wenn alle auf der Installationsdiskette befindlichen Files ohne Veränderungen mitkopiert werden. Eine Änderung der Urheberrechte ist unzulässig und wird strafrechtlich verfolgt.

2. Installation des Programmpaketes FSDEB

Zur Installation von FSDEB sind zunächst einige Vorbereitungen zu treffen, die den Umgang mit dem Debugger wesentlich komfortabler gestalten. Sollten Sie sich bei einigen Arbeitsschritten zunächst nicht ganz über die Wirkungsweise der Aktion im Klaren sein, dann lesen Sie bitte zuerst im entsprechenden Kapitel Ihres DOS-Handbuches über die Auswirkungen auf das System nach. Schließlich soll sich doch Ihr System nach den durchgeführten Änderungen noch genauso verhalten, wie vor der Installation der vorliegenden Software!

Um die Dateien von der Installationsdiskette auf das Ziellaufwerk zu übertragen, wurde ein Installationsprogramm geschrieben, das diesen Vorgang automatisiert. Stecken Sie deshalb die Installationsdiskette in den entsprechenden Laufwerksschlitz und rufen Sie das Installationsprogramm auf:

z.B.:**A:INSTALL**

Nachdem alle Fragen des Installationsprogrammes korrekt beantwortet wurden, müßten alle Dateien auf das Ziellaufwerk übertragen worden sein. Dazu ist es allerdings notwendig, daß das Programm XCOPY in Ihrem DOS-Verzeichnis existiert. Anderenfalls sollten Sie dieses Programm von Ihren DOS-Disketten dorthin übertragen und den Installationsvorgang wiederholen.

Damit FSDEB auch von Verzeichnissen aufrufbar ist, in den das Programm selbst nicht existiert, muß dem Betriebssystem der Suchpfad zu FSDEB bekannt sein. Zu diesem Zweck muß die Datei AUTOEXEC.BAT (befindet sich im Wurzelverzeichnis der Festplatte) geändert werden. Fügen Sie mit Hilfe eines Editors zuerst an den Befehl:

PATH C:\;C:\DOS;.....;

den Suchpfad für den FSDEB an. Die Anweisung hätte dann folgendes Aussehen:

z.B.:**PATH C:\;C:\DOS;.....;C:\FSDEB;**

Jetzt kann der Debugger in jedem beliebigen Unterverzeichnis gestartet werden, ohne den kompletten Pfadnamen eintippen zu müssen.

Der FullScreenDebugger ist damit prinzipiell lauffähig, jedoch kann es vorkommen, daß verschiedene Funktionen nicht, oder nicht richtig ausgeführt werden. Zu diesem Zweck fragt der Initialisierungsmechanismus des FSDEB die DOS-Umgebung nach dem Vorhandensein von speziellen Umgebungsvariablen ab. Also sollte man die Datei AUTOEXEC.BAT mit den entsprechenden Änderungen versehen. Zu diesem Zweck wurden zwei Umgebungsvariable definiert:

FSDEBPATH FSDEBCOM

Erstere hält den Pfadnamen zum FullScreenDebugger. Editieren Sie also erneut die Datei AUTOEXEC.BAT und fügen Sie folgende Befehlszeile ein:

z.B.:**SET FSDEBPATH=C:\FSDEB**

Dadurch kennt FSDEB sein eigenes Verzeichnis und ist in der

Lage, spezielle Dateien, die zur Ausführung von speziellen Kommandos geladen werden müssen, zu finden.

Die zweite Umgebungsvariable dient dazu, dem Debugger mitzuteilen, wenn ein anderer COM-Port, als der standardmäßig eingestellte Port COM1 verwendet werden soll. Für COM2 hätte die Anweisung folgendes Aussehen:

SET FSDEBCOM=2

Diese Anweisung ist ebenfalls an die Datei AUTOEXEC.BAT anzuhängen. Jetzt kommuniziert FSDEB mit dem SBC-86 über COM2. Den gleichen Effekt kann man auch erreichen, wenn FSDEB mit einem Kommandozeilenparameter gerufen wird:

FSDEB /2

Allerdings wirkt das Setzen der Umgebungsvariable permanent (d.h. bis zum nächsten System-Booten), während der Kommandozeilenparameter nur für den aktuellen Aufruf gültig ist.

Jetzt ist FSDEB mit allen Anweisungen nutzbar, Sie müssen nur noch das mitgelieferte RS-232-Kabel mit dem SBC-86 (V24.1) verbinden und FSDEB aufrufen, um mit der Arbeit beginnen zu können. Achten Sie jedoch darauf, daß die für die Kommunikation mit dem Single-Board-Computer vorgesehene COM-Schnittstelle nicht durch residente Programme, wie z.B. dem MOUSE-Treiber belegt ist. Anderenfalls ist der entsprechende Treiber unbedingt vor dem Aufruf von FSDEB.EXE zu entfernen!

Abschließend noch eine Bemerkung zur Konfiguration der Pfade des Editors, des Assemblers und des Linkers. Hierzu existiert eine Datei @@ASM.BAT, die durch den Debugger beim Erstellen von Programmcode aufgerufen wird. Diese Datei ist selbsterklärend geschrieben und muß vor Benutzung an Ihre Belange mit Hilfe eines Texteditors angepaßt werden.

Hier das Listing dieser Datei, wie sie auf der
Installationsdiskette vorhanden ist:

Filename: @@ASM.BAT

```
@echo off
if "%1" == "" goto NoParms

rem                   *** Editor ***
rem
rem    Hier ist Pfad- und Filename des zu verwendenten Text-
rem    verarbeitungsprogrammes einzutragen.
rem
rem    Für alle DOS V5.0-Nutzer: Benutzen Sie EDIT ! (s.u.)
rem
rem    Nicht anwendbar sind Textverarbeitungsprogramme, die
rem    eigene Formate verwenden, d.h. welche bestimmte Steuer-
rem    zeichen in den Quelltext einfügen. Vergewissern Sie
rem    sich also, daß der Editor ein DOS-Text-Format ausgeben
rem    kann!

c:\dos\edit %1.asm

rem                   *** Assembler ***
rem
rem    Hier ist der Pfad- und Filename des Assemblers einzu-
rem    tragen. Verwendbar sind prinzipiell alle 8086-Assemb-
rem    ler, jedoch sollte solchen, die mit INTEL-Mnemonics
rem    arbeiten zweckmäßigerweise der Vorzug gegeben werden.
rem
rem    Sehr gut eignen sich Borlands "TASM" (s.u.), sowie
rem    MicroSofts "MASM" (wurde bei älteren DOS-Versionen dazu
rem    geliefert).

c:\tasm\tasm %1,%1 /l/n/z
if errorlevel 1 goto ASMerror
rem                   *** Linker ***
rem
rem    Haben Sie einen geeigneten Assembler, so ist meist auch
rem    ein Linker im Paket integriert. Tragen Sie hier also
rem    Pfad- und Filename des zum Assembler zugehörigen Lin-
rem    kers ein. Beachten Sie bei Assembler und Linker die
rem    gültigen Kommandozeilenparameter!
rem
rem    Der hier eingesetzte TLINK hat die Möglichkeit COM-
rem    Files direkt zu erzeugen. Sollten Sie einen Assembler
rem    benutzen wollen, der diese Fähigkeit nicht besitzt, so
rem    ist das EXE-File-Format in den Debugger zu laden(Kom-
rem    mando "E"), bzw. mit einer zum Assembler gehörigen Da-
rem    tei in das COM-File-Format zu wandeln (z.B. EXE2BIN bei
rem    MASM).
```

```
c:\tasm\tlink %1 /t/x
if errorlevel 1 goto LINKerror
goto AllOk

:NoParms
@echo Kein Filename angegeben !!
goto Ready

:ASMerror
@echo Fehler beim Assemblieren !!
goto Ready

:LINKerror
del %1.obj
@echo Fehler beim Linken !!
goto Ready

:AllOk
del %1.obj
@echo Alles bestens...
:Ready
```

3. Aufbau der Soft- und Hardwarekomponenten des Single-Board-Computers SBC-86

3.1. Die Grundstruktur

Die Grundstruktur des SBC-86 umfaßt folgende Funktionsgruppen:

- 8086-Mikrocomputer
- Hexadezimale Funktionstastatur
- 8-stelliges Siebensegmentdisplay
- Binäre Ein- Ausgabeeinheit
- Paralleles Interface
- 2 Serielle Interfaces nach V.24
- XT-kompatibler Steckplatz
- Netzteil

Bei der Konzeption wurden ausschließlich Bausteine verwendet, die entweder standardmäßig zum 8086-System zugehörig sind, bzw. Methoden digitaler Schaltungstechnik repräsentieren. Sicherlich wäre es möglich gewesen, einen Großteil der eingesetzten Logikschaltkreise in Programmierbaren Logischen Feldern (PAL's, GAL's usw.) zu verstecken, dann würde allerdings deutlich die Transparenz des Schaltungsaufbaues unter dieser Technologie leiden. Mikrocomputer wie heutige 386-er, 486-er usw. bedienen sich natürlich dieser Vorgehensweise, sonst wäre eine solche Miniaturisierung niemals denkbar.

Befassen wir uns also mit den einzelnen Funktionsgruppen, wobei an dieser Stelle nur auf Besonderheiten eingegangen werden soll. Grundsätzliche schaltungstechnische Details können den Literaturquellen im Literaturverzeichnis entnommen werden.

3.2. Der 8086-Mikrocomputer

Für den Betrieb des 8086-Prozessors wurde der Minimum-Modus gewählt, denn in dieser Betriebsart generiert der Prozessor alle benötigten Steuersignale selbst, d.h. es kann auf den Einsatz eines speziellen BUS-Controllers verzichtet werden. Diese Betriebsart wurde speziell für Minimalsysteme vorgesehen.

Betrachten Sie bitte während der folgenden Ausführungen auch die Schaltungen im Anhang (CPU-Board, CPU-RAM (ROM)).

Zur Takterzeugung dient ein 8284, der mit einem Quarz von 15 MHz getaktet wird. Da der 8284 den Takt drittelt, arbeitet die CPU mit einer Taktfrequenz von 5 MHz. Daten-, Adress- und Steuersignale werden durch entsprechende Treiberbausteine vom Prozessor getrennt, verstärkt und den entsprechenden Speicher- und Peripheriebausteinen zugeführt.

Als Programmspeicher wurden ROM's mit insgesamt 4 kByte Speicherkapazität eingesetzt. Diese enthalten Monitorprogramme, die Kommunikation mit externen Geräten, die Möglichkeit der Programmtestung auf niedrigstem Niveau und einige logische Systemtreiber beinhalten. Alle Programme, die vom Anwender erstellt werden, werden in RAM's mit einer Kapazität von 48 kByte gespeichert. Von diesen 48 kByte RAM können 32 kByte ebenso durch ROM ersetzt werden, um beispielsweise Anwenderprogramme fest auf dem SBC-86 zu installieren (wichtig für Lehrzwecke !). Dies muß allerdings mittels kleiner Schalter eingestellt werden (siehe Anhang). Damit die den Adressen entsprechenden Speicherbausteine ausgewählt werden können, wurden 1-aus-8-Dekoder eingesetzt. Zu beachten ist die Besonderheit der unvollständigen Adreßdekodierung, die zur Adreßdekodierung nur die Adreßleitungen berücksichtigt, die unbedingt benötigt werden. Dadurch kann bei Minimalsystemen der Dekodieraufwand verringert werden.

Die Adressen für die Ein- Ausgabeports wurden auf ähnliche Art dekodiert, wie die der Speicher. Jedoch korrespondieren beim 8086 die Ports auf geraden Adressen mit dem niederwertigen Teil des Datenbusses, während sich ungerade Adressen auf den höherwertigen Teil des Datenbusses beziehen. Um die Hardware des Single-Board-Computers zu vereinfachen, wurde vollständig auf ungerade Portadressen verzichtet. Das hat allerdings den Nachteil, daß wortweise Datenein-/ausgabe auf diese Ports nicht den gewünschten Effekt bringt. Beachten Sie dies bitte!

Die Portadressen der Peripheriebausteine sind im Anhang ersichtlich.

3.3. Die hexadezimale Funktionstastatur

Jeder Taste ist bezüglich des Betriebssystemmonitors eine bestimmte Funktion zugewiesen. An dieser Stelle soll jedoch nur auf die elektrische Funktion eingegangen werden. Der Schaltplan ist im Anhang (Keyboard) ersichtlich.

Ein Taktgenerator erzeugt, wenn keine Taste gedrückt ist, Impulse, die von einem 74HCT193 gezählt werden und an dessen Ausgang in binärer Form (3 Bit) anliegen. Ein 1-aus-8-Dekoder steuert in Abhängigkeit des Binärkodes die entsprechenden Spaltenleitungen der Tastaturmatrix an. Die Tastaturzeilen werden normalerweise über Pull-Up-Widerstände auf High-Potential gehalten. Wird jedoch eine Taste gedrückt, so bekommt die Tastaturzeile bei ausgewählter Spalte Low-Pegel, wodurch der Impulsgeber angehalten wird. Somit stehen am Eingabetor die Daten der entsprechenden Taste zur Verfügung.

Hierbei gilt: Bit 0..2 führen die Zeile (1-aus-3 invers-
kodiert),
Bit 3..5 führen die Spalte (3 Bit binär) und
Bit 6..7 führen Low-Pegel.

Wurde keine Taste gedrückt, so führen die Bits 0..2 High-Pegel.

Das Tastaturport ist ein Nur-Lese-Port, d.h. ein Schreiben von Daten bleibt ohne Wirkung. Ein Auslesen der Portadresse 80H bringt das Ergebnis der Tastaturabfrage in folgender Weise zurück:

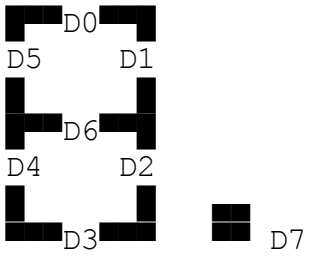
Spalte 7	6	5	4	3	2	1	0	
06H	0EH	16H	1EH	26H	2EH	36H	3EH	Zeile 2
05H	0DH	15H	1DH	25H	2DH	35H	3DH	Zeile 1
03H	0BH	13H	1BH	23H	2BH	33H	3BH	Zeile 0

Beachten Sie bitte, daß jede Taste beim Drücken und Loslassen prellt, d.h. die Tastenkodes erst nach einigen Millisekunden stabil sind. Deshalb sollte bei Verwendung der direkten Tastaturkodes eine softwaremäßige Entprellung realisiert werden. An dieser Stelle jedoch schon einmal vorab: Es existiert im Betriebssystemmonitor eine Routine, die eine exakte Tastaturabfrage realisiert, wobei die softwaremäßige Entprellung bereits realisiert ist. Doch dazu später.

3.4. Das achtstellige Siebensegmentdisplay

Das 8-stellige Siebensegmentdisplay ist ein Nur-Schreib-Port. Ein Lesen von diesem Port bringt kein sinnvolles Resultat. Anhand der Schaltung (siehe Anhang, Display) erkennt man leicht, daß durch den Einsatz von Latches (Zwischenspeicher) jedes Datum für jede Anzeigestelle gespeichert wird. Das hat den Vorteil, daß das Display nicht zyklisch aufgefrischt werden muß, um eine flimmerfreie Anzeige zu erhalten. Außerdem würde dadurch Rechenzeit verloren gehen, Zeitgeberkanäle würden belegt werden usw.

Jede Displaystelle ist über eine eigene Portadresse ansprechbar. Es sind die Portadressen 90H..9EH (nur gerade Adressen) gültig, wobei der rechten Displaystelle die Adresse 90H zugewiesen ist. Der Datenbus korrespondiert folgendermaßen mit den einzelnen Displaysegmenten:



Jedes Segment wird mit High-Pegel eingeschaltet, aus diesem Grund kamen invertierende Latches zum Einsatz.

3.5. Die binäre Ein- Ausgabeeinheit

Die binäre Ein- Ausgabeeinheit kann ziemlich nützliche Dienste erweisen, wenn man sie nur entsprechend nutzt. Es handelt sich hierbei um ein Port, das bei Ausgaben das Ausgabedatum binär auf 8 Luminiszenzdioden darstellt. Außerdem kann man beim Lesen von diesem Port den Status von 8 Schaltern abfragen. Dadurch hat man außer Display und Hex-Tastatur die Möglichkeit der Kommunikation mit diesen Elementen. Die Schaltung (vgl. Anhang, SBC-86 Schalter) weist keinerlei Besonderheiten auf, außer der Tatsache, daß hier eine andere Möglichkeit der Adreßdekodierung angewandt wurde, der Adreßvergleicher. Dieser Vergleicher liefert nur dann ein Freigabesignal, wenn die Adressen vom Adreßbus mit den Adreßschaltern übereinstimmen. Man hat hierbei somit die Möglichkeit, die Portadresse dieser Einheit frei zu wählen solange man beachtet, daß Portein-/ausgaben nur auf geraden (A0=0) Adressen Wirkung zeigen. Das bedeutet auch, daß man die Ein- Ausgabeeinheit zur Kontrolle parallel zu anderen Peripheriebausteinen schalten kann, um somit den Datenaustausch zu beobachten. Andererseits können Daten mit Hilfe der 8 Schalter manipuliert werden, um beispielsweise bestimme Eingabefunktionen zu simulieren. Weiterhin wurde eine Experimentalschnittstelle (X501) zur Verfügung gestellt. Hier

können ebenfalls Ein- und Ausgaben auf der über die Adreßschalter eingestellten Adresse realisiert werden. Die Steckerpins liegen den Lumies bzw. Schaltern rein elektrisch parallel. Die Beschaltung der Experimentalschnittstelle befindet sich im Anhang.

3.6. Das parallele Interface

Das Parallelinterface weist keinerlei Besonderheiten auf (vgl. Seriell-/Parallel-Port im Anhang). Zum Einsatz kam ein PPI 8255, der 3 8-Bit-Ports besitzt, die in verschiedenen Arbeitsmodi benutzbar sind. Die Ausgänge dieses Schaltkreises wurden auf Steckverbinder X1 geführt und sind somit dem Anwender frei zugänglich. Damit sind Sie als Experimentator gefordert, um diese Schnittstelle mit Leben zu erfüllen. Denkbar wäre hier eine einfache binäre Ein-/Ausgabe, mit deren Hilfe verschiedenste Peripheriegeräte steuerbar werden. Beachten Sie, daß die Ports A & B dieses Bausteines Totem-Pole-Ausgänge besitzen, während Port C mit Open-Collector-Ausgängen bestückt ist. Diese Ausgänge vertragen maximal eine TTL-Standardlast, bzw. 4 Low-Power-Schottky-Eingänge.

Die Interruptausgänge (PC0 und PC3) wurden in der Laborversion dem Interruptcontroller zugänglich gemacht, so daß der Schaltkreis auch interruptbefähigt war. In der überarbeiteten Hardwareversion wurde dies zwar vorgesehen, jedoch schaltungstechnisch nicht realisiert. Vielleicht bringt ein zukünftiges Hardwarelayout da eine Verbesserung. Diese Information an dieser Stelle nur deshalb, um Besitzer früherer Hardwarebeschreibungen nicht allzu sehr zu verwundern. Dort war es leider bezüglich der ausgelieferten Hardware nicht ganz korrekt beschrieben.

3.7. Die seriellen Interfaces nach V.24 und der Zeitgeber

Die Schaltung ist im Anhang (Seriell-/Parallel-Port) einzusehen. Es wurden zwei komplett ausgestattete serielle Schnittstellen realisiert, da beim Einsatz des externen Full-Screen-Debuggers bereits eine Schnittstelle für die Kommunikation verplant ist. Zum Einsatz kamen 8251A (SIC), die sowohl für synchrone, als auch für asynchrone serielle Datenübertragungen einsetzbar sind. Selbst beim Einsatz des Full-Screen-Debuggers ist mit ein paar Tricks eine serielle Kommunikation des SBC-86 mit sich selbst möglich.

Die Zeitbasis für die seriellen Interfaces wird über einen Kanal eines Zähler-Zeitgeber-Schaltkreises 8253 (PIT) entsprechend vorgeteilt. Dazu jedoch später. Jeder der drei Kanäle des PIT erhält am Zählereingang eine Frequenz von 1.8432 MHz, die durch einen Quarzoszillator (7.3728 MHz / 4) erzeugt wird. Somit entsteht zum einen der Sende-/Empfangstakt für die beiden SIC OUT 0), zum anderen ein Takt für die Kanäle 1 und 2 des PIT, deren Ausgänge (OUT 1 und OUT 2) mit dem Interruptcontroller verbunden sind. Dadurch ist es möglich, diese Kanäle für interruptgesteuerte Zeitgeberfunktionen zu nutzen. Die Zuordnung zum Interruptcontroller ist im nächsten Punkt beschrieben.

Jetzt aber wieder zu den Datenübertragungsraten (Baudraten). Die Anzahl der pro Sekunde übertragenen Bits wird in Baud (Bd, nach dem Franzosen Baudot benannt) angegeben. Diese Datenübertragungsrate kann durch geeignete Wahl des programmierbaren Teilerfaktors in weiten Grenzen variiert werden. Basis für die Berechnung der Datenübertragungsrate ist die Zählfrequenz von 1.8432 Mhz. Dieser krumme Wert ist für die Realisation von gängigen Datenübertragungsraten überaus günstig. Im SIC selbst wird die Frequenz durch 16 geteilt, so daß der programmierbare Vorteiler des PIT für verschiedene Baudraten leicht errechnet werden kann. Die nachfolgende Tabelle weist die Vorteilerwerte für ausgewählte Baudraten aus:

Baudrate (Bd)	Teilerfaktor
50	36864
75	24576
100	18432
110	16756
300	6144
600	3072
1200	1536
2400	768
4800	384
9600	192
19200	96

Von den SIC (8251A) wurden außer den Sende-Empfangsleitungen auch die Steuersignale RTS und CTS auf die Steckverbinder geführt. Dadurch kann bei der Datenübertragung sowohl Software- als auch Hardwareprotokoll (RTS/CTS-Protokoll) gefahren werden. Das Softwareprotokoll muß jedoch die Synchronisation von Sender und Empfänger mit Hilfe von Steuerdaten realisieren. Dieser Weg wurde auch bei der Kommunikation zwischen SBC-86 und Full-Screen-Debugger gegangen. Sonst dienen RTS und CTS als Steuerleitungen, wobei CTS direkt auf die Freigabe des Senders wirkt. Das hat zur Folge, daß bei Softwareprotokoll die Steuerleitung CTS (low-aktiv) Low-Potential führen muß. Deshalb wurden die Schalter S28 und S29 zur Umschaltung des Signals CTS realisiert. Die entsprechenden Einstellungen sind im Anhang ersichtlich.

Die Aus-und Eingangssignale (TTL-Pegel) der beiden SIC werden V.24-Treibern zugeführt, die die Pegelwandlung V.24 <--> TTL vornehmen. Dem Anwender stehen die Signale an den Steckern X2 und X4 zur Verfügung. Die Belegung der Stecker ist ebenfalls dem Anhang zu entnehmen.

3.8. Der Interruptcontroller

Bekanntlich dient ein Interruptcontroller zur Verwaltung von Unterbrechungsanforderungen der Peripheriecontroller. Zu diesem Zweck wurde ein 8259A in das System integriert, der bereits schaltungstechnisch mit zwei Zeitgeberkanälen und den beiden seriellen Schnittstellen verbunden wurde. Beachten Sie beim Literaturstudium, daß Intel zwei Interruptcontroller (PIC) fast gleicher Bezeichnung, jedoch unterschiedlichen Funktionsinhaltes auf den Markt gebracht hat. Das ist zum einen der 8259, der für 8080-Systeme konzipiert ist und zum anderen der 8259A, der eine Weiterentwicklung hinsichtlich der 8086-Systeme enthält. Die Programmierung beider Controller ist jedoch vollkommen unterschiedlich!

Da der PIC bereits in das System integriert ist, macht sich auch eine softwaremäßige Initialisierung notwendig. Darüber gibt das Kapitel zum Betriebssystemmonitor Auskunft.

Die hardwaremäßige Beschaltung des PIC wurde folgendermaßen realisiert:

```
IR0.............PIT, Kanal 1 (OUT 1)
IR1.............PIT, Kanal 2 (OUT 2)
IR2.............SIC1, TxRdy
IR3.............SIC1, RxRdy
IR4.............SIC2, TxRdy
IR5.............SIC2, RxRdy
IR6.............frei, (PPI, PC3)
IR7.............frei, (PPI, PC0)
```

Die Klammerwerte für IR6 und IR7 sind für zukünftige Hardwarelayouts denkbar, bzw. könnten verfügbar werden, wenn vom Anwender Drahtbrücken zwischen den Eingängen des PIC und den Ein-bzw. Ausgängen des PPI nachgerüstet würden. Dies hätte die Interruptbefähigung des Parallelinterfaces zur Folge.

Da die Sender-/Empfänger-Bereitschaftsleitungen der beiden SIC ebenfalls zum PIC führen, ist sowohl interruptgesteuertes Senden, als auch interruptgesteuertes Empfangen serieller Daten möglich.

Die Schaltung ist im Anhang (I/O-Board) zu finden.

3.9. Der XT-kompatible Erweiterungssteckplatz

Um Erweiterungen zum SBC-86 nicht verschlossen zu sein, wurde eine Busschnittstelle geschaffen, die weitestgehend XT-kompatibel gehalten wurde. Dadurch können an diese Schnittstelle selbst XT-I/O-Karten angeschlossen werden, wenn sie

- nur 5V-Spannungsversorgung (max. 1A) benötigen,
- keine Wait-Zyklen benötigen und
- Datenverkehr nur über gerade 8-Bit-Portadressen abwickeln.

Somit könnte alter Computerschrott noch nützliche Dienste erweisen! Die Beschaltung des Steckplatzes ist aus dem Schaltplan ersichtlich.

4. Der Betriebssystemmonitor

4.1. Grundlegende Gedanken zum Betriebssystemmonitor

Der Betriebssystemmonitor beinhaltet eine Vielzahl von Softwarekomponenten, die den SBC-86 sowohl als eigenständiges Gerät, als auch in Verbindung mit einem externen Bediensystem nutzbar machen. Es mag komisch erscheinen, aber der externe Full-Screen-Debugger ist eigentlich nichts weiter, als ein einigermaßen komfortables Bediensystem für den SBC-86. Sämtliche Software, die zum Debuggen und Testen von Anwenderprogrammen nötig ist, wurde bereits im Betriebssystemmonitor (in der Folge nur kurz Monitor genannt) implementiert.

Nach dem Einschalten bzw. dem Betätigen des Reset-Knopfes werden zunächst folgende Hardwarekomponenten initialisiert:

Speicher:	Aufbau von Interrupttabelle und Systemstapel
CPU:	IP=SP=0100H, CLI, alle anderen Register gleich 0
PPI:	alle Ports im Ausgabemodus (Mode 0)
SIC1:	Datenformat 8 Bit, keine Parität, 1 Stopbit, Vorteiler 16
PIT:	Zähler 0: 16-Bit-Vorteiler (Mode 3), Teilerfaktor 12
PIC:	alle Interrupts gesperrt, End-Of-Interrupt (EOI)-Modus

Nach der Initialisierung erscheint auf dem Display die Meldung " SBC-86 ", wobei entweder die Eröffnung der Datenkommunikation mit einem externen Rechner über die serielle Schnittstelle (V24.1) oder ein Tastendruck auf dem Keyboard des SBC-86 erwartet wird. Dies dient zur Unterscheidung zwischen dem Stand-Alone- und dem Master-Controlled-Modus. Letzterer wird im nächsten Kapitel (FSDEB) beleuchtet.

4.2. Die Monitorkommandos

Im Stand-Alone-Modus erwartet der Monitor immer dann ein Kommando, wenn folgende Displayausgabe erscheint: "CMD xxxx". CMD steht dabei für Command, xxxx repräsentiert den jeweils aktuellen Stand des Instruction-Pointers (IP). Kommandos werden mit einer der Tasten der unteren Funktionstatenzeile aufgerufen. Alle anderen Tasten dienen nur zur Parametereingabe für diese Kommandos. Jedes Kommando wird mit der Bestätigungstaste (ENTER) abgeschlossen. Manche Kommandos gestatten ein "Blättern" mit Hilfe der Tasten + und -.

Hier eine kurze Aufstellung der Kommandos:

MEM: Speicherinhalte anzeigen und ändern
- MEM betätigen, Ausgabe "MEM xxxx"
- Adresse eintippen, ENTER
- evtl. neuen Speicherinhalt eingeben
- mit + oder - im Speicherraum blättern
- mit ENTER Kommando beenden

REG: Registerinhalte anzeigen und ändern
- REG betätigen, Ausgabe "REG "
- gewünschtes Register durch Betätigen eines Registersymboles auswählen
- evtl. neuen Registerinhalt eintippen
- mit + und - im Registerraum blättern
- mit ENTER Kommando beenden

ACHTUNG: Der SP trägt den zur Anzeigezeit aktuellen Wert, dieser ist vom tatsächlichen, vom Anwenderprogramm 'gesehenen' Wert unterschiedlich!

OUT: Ausgabe eines 8-Bit-Datums auf ein Port
- OUT betätigen, Ausgabe "OU0000 "
- gewünschte Portadresse eintippen, mit ENTER abschließen
- Ausgabedatum eingeben
- Kommando mit ENTER beenden

IN: Eingabe eines 8-Bit-Datums von einem Port
- IN betätigen, Ausgabe "IN0000 "
- gewünschte Portadresse eintippen, mit ENTER abschließen
- das eingelesene Datum erscheint neben der Portadresse
- Kommando mit ENTER beenden

TRACE
into: Einzelschrittbetrieb (Single-Step)
- TRACE into betätigen
- Ausführung des Befehls auf den CS:IP zeigt
- "CMD xxxx" trägt neuen IP

STEP
over: Komplette Ausführung des nächsten Befehles
(Unterprogrammrufe und rückwärtsgerichtete, bedingte Sprünge werden komplett in Echtzeit ausgeführt)
- STEP over betätigen
- Ausführung des Befehls auf den CS:IP zeigt, bis CS:IP auf den logisch nächsten Befehl zeigt
- "CMD xxxx" trägt neuen IP

GO: Programmstart (Echtzeitlauf)
- GO betätigen, Ausgabe "GO 0000"
- evtl. Haltepunktadresse eingeben, bei dem der Echtzeitlauf gestoppt werden soll
- Kommando mit ENTER bestätigen
- Echtzeitlauf bis Zieladresse, wurde keine Zieladresse eingegeben, so wirkt kein Haltepunkt (Abbruch dann mit RESET)

ACHTUNG: Funktioniert (ebenso wie STEP over) nur, wenn Programme im RAM getestet werden. GO ohne Haltepunktadresse funktioniert auch im ROM!

4.3. Die Systeminterrupts

Um einige simple Testmechanismen realisieren zu können, wurde Gebrauch vom Single-Step-Interrupt , vom Break-Point-Interrupt und anderen Interrupts gemacht. Dadurch ergibt sich folgende, initiale Belegung der Interrupttabelle:

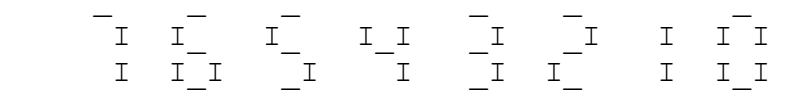
Adresse: Verwendung:

0000:0000 INT 00 Divisionsüberlauf (Standard)
0000:0004 INT 01 Single-Step (vom Monitor belegt)
0000:0008 INT 02 NMI (Standard)
0000:000C INT 03 Breakpoint-Interrupt (vom Monitor belegt)
0000:0010 INT 04 (INTO, Standard)
0000:0014 INT 05 (Tastaturinterrupt, vom Monitor belegt)
0000:0018 INT 06 (Displayinterrupt, vom Monitor belegt)
0000:001C INT 07 (frei)
0000:0020 INT 08 (PIC-IR0, PIT Kanal 1)
0000:0024 INT 09 (PIC-IR1, PIT Kanal 2)
0000:0028 INT 0A (PIC-IR2, SIC1 TxRdy)
0000:002C INT 0B (PIC-IR3, SIC1 RxRdy)
0000:0030 INT 0C (PIC-IR4, SIC2 TxRdy)
0000:0034 INT 0D (PIC-IR5, SIC2 RxRdy)
0000:0038 INT 0E (PIC-IR6, frei (PPI, PC3))
0000:003C INT 0F (PIC-IR7, frei (PPI, PC0))

Ab der Adresse 0000:0040 befinden sich einige Systemzellen, der Systemstapel belegt den Bereich von 0000:004C...0000:00FF. Anwenderprogrammbereich ist ab der Adresse 0000:0100, die Länge variiert je nach Systemausstattung. Der Monitor belegt die Adressen von 0000:C000...0000:CFFF (Achtung: Unvollständige Adreßdekodierung!).

Dem aufmerksamen Leser ist sicher nicht entgangen, daß in der Interrupttabelle Display- und Tastaturinterrupts Einträge belegen. Dies sind die Eintrittspunkte für Softwareinterrupts, die mit diesen peripheren Einheiten korrespondieren. In der Hardwarebeschreibung wurden Sie bereits auf die Tatsache des Prellens der Taster des Keyboards hingewiesen, die mit Hilfe von etwas Software entprellt werden können. Da zum Aufbau des Betriebssystemmonitors sowieso ein kompletter Konsoltreiber geschrieben werden mußte, gibt es also keinen Grund diese Systemfunktionen nicht auch dem Anwender zugänglich zu machen. Zu diesem Zweck existieren diese Systeminterrupts INT 5 und INT 6. Jedem dieser Softwareinterrupts sind Parameter in den Registern AX, BX und DL zu liefern, bzw. es werden Rückgabewerte in diesen CPU-Registern geliefert. Das Register AH dient dabei zur Spezifikation der Unterfunktionsnummer.

Generell werden die Displaystellen über eine logische Stellennummer adressiert:



Auch der Tastaturtreiber liefert nunmehr logische Tastenkodes:

Spalte 7	6	5	4	3	2	1	0	
08H	09H	0AH	0BH	0CH	0DH	0EH	0FH	Zeile 2
00H	01H	02H	03H	04H	05H	06H	07H	Zeile 1
17H	16H	15H	14H	13H	12H	11H	10H	Zeile 0

Zu beachten ist die Drehung der unteren Kodezeile. Die Kommandotasten belegen Kodes ≥ 10H.

Da der Konsoltreiber eine logische Programmierschnittstelle bietet, sollte auf diese Systemfunktionen zurückgegriffen werden. Generell gilt: Tastatureingaben werden mit INT 5 realisiert, während sämtliche Displayausgaben über INT 6 abgewickelt werden.

4.4. Die Funktionen des Konsoltreibers

INT 5, Funktion 0 - Tastaturstatus ermitteln
- Aufruf: AH=00
- Ergebnis: AL=00, keine Taste betätigt
  AL=FF, Taste betätigt

INT 5, Funktion 1 - Warten auf Tastendruck, Tastenkode lesen
- Aufruf: AH=01
- Ergebnis: Tastenkode in AL

INT 5, Funktion 2 - Eingabe eines Hex-Wortes, Echo auf Display
- Aufruf: AH=02
  Vorgabewert in BX
  Anzeige ab Displaystelle DL
- Ergebnis: Hex-Wort in AX
  Quittungskode in DL (10H, 16H, 17H)

INT 5, Funktion 3 - Eingabe eines Hex-Bytes , Echo auf Display
- Aufruf: AH=03
  Vorgabewert in BL
  Anzeige ab Displaystelle DL
- Ergebnis: Hex-Byte in AL
  Quittungskode in DL (10H, 16H, 17H)

INT 6, Funktion 0 - Display löschen (Clear Screen)
Aufruf: AH=00
Ergebnis: Display dunkel

INT 6, Funktion 1 - Ausgabe eines ASCII-Zeichens
Aufruf: AH=01
Zeichenkode in AL
Anzeige auf Displaystelle DL
Ergebnis: Ausgabe

INT 6, Funktion 2 - Ausgabe eines ASCIIZ-Strings
(durch 00H abgeschlossene Zeichenkette)
Aufruf: AH=02
Stringadresse in CS:BX
Anzeige ab Displaystelle DL
Ergebnis: Ausgabe bis Endekennung 00H

INT 6, Funktion 3 - Ausgabe eines Hex-Wortes
Aufruf: AH=03
Hex-Wort in BX
Anzeige ab Displaystelle DL
Ergebnis: Ausgabe

INT 6, Funktion 4 - Ausgabe eines Hex-Bytes
Aufruf: AH=04
Hex-Byte in BL
Anzeige ab Displaystelle DL
Ergebnis: Ausgabe

4.5. Ein Programmbeispiel

Realisiert werden soll ein intelligenter Lichtschalter, der die Stellung der Schalter der Experimentalschnittstelle auf die dortigen Luminiszenzdioden abbildet. Eine Nutzung von Keyboard und Display verbietet sich leider bei Verwendung der Debug-Mechanismen des Monitors von selbst.

Unter der Annahme, der Adreßschalter der Experimentalschnittstelle wäre auf Adresse 00H eingestellt, könnte das Programm folgendes Aussehen haben:

```
        .model tiny
        .code
        org 100h
Switch: mov dx,0        ; Portadresse fuer Ein- und Ausgaben
Again:  in al,dx        ; Schalterbelegung einlesen
        out dx,al       ; und auf Lumis ausgeben
        jmp short Again ; und immer wieder

        end Switch
```

In dieser Form läßt sich das Programm allerdings nicht in den SBC-86 transportieren, so daß wir uns zunächst das Programmlisting betrachten, so wie es ein beliebiger Assembler erzeugen könnte:

```
1  0000           .model tiny
2  0000           .code
3                 org 100h
4  0100  BA 0000  Switch: mov dx,0     ; Portadresse fuer
                                       ; Ein- und Ausgaben
5  0103  EC       Again:  in al,dx     ; Schalterbelegung
                                       ; einlesen
6  0104  EE               out dx,al    ; und auf Lumis ausgeben
7  0105  EB FC            jmp short Again ; und immer wieder
8                 end Switch
```

In Zeile 4 beginnt der eigentliche Programmcode. Wir haben also mit Hilfe der MEM-Funktion ab Adresse 100H folgende Bytesequenz einzutippen:

```
BA 00 00 EC EE EB FC
```

Jetzt kann zunächst mit dem GO-Befehl (ohne Haltepunkt) die Funktion des Programmes getestet werden. Anschließend können selbst mit Hilfe dieses einfachen Programmes andere Funktionen des Betriebssystemmonitors ausprobiert werden oder Änderungen am Programmcode vorgenommen werden. Man wird jedoch sehr schnell erkennen, daß diese Methode ziemlich umständlich und sehr zeitintensiv ist. Es müßte also eine Möglichkeit existieren, die es gestattet, Programmcode auf einem externen

Rechner zu entwickeln und anschließend in den SBC-86 zu laden. Genau aus diesem Grund wurde der Full-Screen-Debugger FSDEB geschaffen, dessen Funktion und Bedienung im nächsten Kapitel beschrieben ist.

5. Der Full-Screen-Debugger FSDEB

5.1. Allgemeines

Der Full-Screen-Debugger FSDEB wurde speziell für den SBC-86 entwickelt und kann deshalb auch nur in Zusammenhang mit diesem eingesetzt werden. Eigentlich ist jegliche Debug-Software im Betriebssystemmonitor enthalten. FSDEB dient somit nur zur komfortablen Steuerung dieses Monitors und wird über die serielle Schnittstelle mit dem SBC-86 verbunden. Zur Datenkommunikation wird ein Softwareprotokoll verwendet. Der Verzicht auf Hardware-Handshaking hat seine Ursache in der Entwicklungsgeschichte des SBC-86, wo ursprünglich auch stromgekoppelte Schnittstellen (IFSS) zum Einsatz kommen sollten. Somit wurde zwischen beiden Komponenten eine Art "Auftrag-Bestätigung"-Protokoll eingesetzt. Die Sende- und Empfangsroutinen arbeiten im Polling-Modus, d.h. es entfielen auch Puffermechanismen, wie sie z.B. bei interruptgesteuerter Datenübertragung nötig gewesen wären. Durch diese relativ simple Art der Datenübertragung kann es jedoch durchaus vorkommen, daß die Synchronität der Datenübermittlung verletzt wird. Deshalb an dieser Stelle ein paar Hinweise, die eine Verklemmung der Datenübertragung verhindern helfen sollen. Vor dem Start von FSDEB muß der SBC-86 eingeschaltet sein, da sonst überhaupt keine Datenübertragung zustande kommen kann. Man unterscheidet weiterhin zwischen einem Hardware-Reset (Drücken des Reset-Knopfes am SBC-86) und einem Software-Reset (FSDEB-Kommando R). Software-Resets wirken auf beide Komponenten und zwar immer. Hardware-Resets werden nur dann auch von FSDEB erkannt, wenn die Reset-Taste während der Ausführung eines Kommandos (Kommunikation mit dem Monitor nötig) gedrückt wurde. Deshalb sollte nach einem Hardware-Reset anschließend auch ein Software-Reset durchgeführt werden. In bestimmten Fällen erkennt FSDEB jedoch diese Resets selbständig (Ausgabe "Reset detected"), so daß ein Soft-Reset sich unnötig macht. Asynchronität im Datenaustausch äußert sich immer in einer "sinnlosen" Ausgabe in den Fenstern des Full-Screen-Debuggers und kann jederzeit per Kommando "R" behoben werden!

Jetzt aber endlich einige grundlegende Worte zu FSDEB. Nach dem Aufruf des Programmes erscheint zunächst ein Meldeschirm, den man nur per Tastendruck wegbekommt. Danach versucht FSDEB die Eröffnung der Kommunikation mit dem SBC-86. Deshalb sollte sich der SBC-86 in einem definierten Zustand befinden, den man am besten durch Druck auf den Reset-Taster erzeugt. Kommt keine Kommunikation zustande kann man an dieser Stelle mit CTRL-Break abbrechen und den Ladevorgang wiederholen.

Der Schirm des Full-Screen-Debuggers ist in verschiedene Fenster unterteilt:

- *CPU-Fenster (Register/Flags)*, gestattet Einblick in die CPU-Register und Flags
- *Status-Fenster (Debuginfo)*, widerspiegelt eingestellte Parameter
- *Eingabe-Fenster (Command)*, hier findet die Kommunikation zwischen Nutzer und FSDEB statt
- *Nachrichten-Fenster (Message)*, hier landen alle Fehlernachrichten und Aufforderungen an den Nutzer
- *Quellkode-Fenster (Source)*, zeigt ein (versuchtes) Reassembling des geladenen Maschinenkodes, ungültiger 8086-Kode wird als DB (define byte) ausgegeben
- *Speicher-Fenster (Memory)*, gestattet Einblick in die Speicherbausteine (Darstellung hexadezimal und ASCII)
- *Hilfe-Fenster*, stellt in der 25. Bildschirmzeile die wichtigsten Kommandos, die über Funktionstasten erreichbar sind, dar

Weiterhin existieren noch 2 Hilfeschirme, die jederzeit über die Taste F1 abrufbar sind und die FSDEB-Kommandos beschreiben.

Die Kommandos, über die die Aktionen des Debuggers gesteuert werden, haben eine bestimmte Syntax, die durch einen Kommandointerpreter geprüft wird. Jedes Kommando wird außerdem beim Eintippen durch eine Online-Hilfe (Angabe der Syntax im Nachrichtenfenster) unterstützt. Wurde ein Kommando syntaktisch unkorrekt eingegeben, so gibt FSDEB im Nachrichtenfenster eine Fehlermeldung aus. Das Datenformat für numerische Eingaben ist prinzipiell hexadezimal.

Im nächsten Abschnitt werden alle Kommandos detailliert beschrieben. Die Kommandosyntax (siehe Hilfeschirm und Online-Hilfe) enthält dabei bestimmte Sonderzeichen, die wie folgt interpretiert werden sollen:

- das Zeichen "|" bedeutet soviel wie ODER
- Ausdrücke in geschweiften Klammern ({}) schliessen durch ODER (|) verknüpfte Ausdrücke ein, dabei bewirkt jeder Ausdruck die gleiche Funktion
- Ausdrücke in eckigen Klammern ([]) müssen eingegeben werden, diese Ausdrücke sind Funktionsparameter wie z.B. Adressen, Werte, DOS-Pfade usw.
- <CR> bezeichnet die Eingabetaste (ENTER)

Alle anderen Angaben sind selbsterklärend. Jedes Debugger-Kommando kann durch einen Satz von Editier-Kommandos verändert werden. Dazu existiert ein Hilfeschirm, der die entsprechenden Steuertasten beschreibt. Wichtig ist dabei die Rückhol-Funktion, die bereits eingetippte Kommandos, sofern sie

parameterbehaftet sind, rückholbar macht. Dadurch kann man sich bei oft wiederkehrenden Kommandos ein bischen Schreibarbeit sparen.

5.2. Der FSDEB-Kommandosatz

5.2.1. Assemblerruf - A

Dieses Kommando kann ebenfalls durch die Taste F5 aufgerufen werden. Dabei wird die Datei @@ASM.BAT ausgeführt. Die Struktur dieser Datei ist in Abschnitt 2 bereits ausführlich beschrieben. @@ASM.BAT sollte sich in dem Pfad befinden, der der Umgebungsvariable FSDEBPATH zugewiesen wurde. Wurde diese Variable nicht eingestellt, so sucht FSDEB die Datei @@ASM.BAT im Aufrufverzeichnis.

```
Syntax:   {A|F5}  ; FSDEB erwartet die Eingabe eines
                  ; Dateinamens
          Eingabe des Dateinamens<CR>

Beispiel: A
          TEST  ; Ohne Dateierweiterung, da @@ASM.BAT die
                ; Erweiterung selbst generiert
```

5.2.2. Breakpoint setzen/löschen - B

Breakpoints sind permanente Unterbrechungspunkte, bei denen der Programmlauf unterbrochen wird. Läuft ein Programm beim Echtzeitlauf auf einen solchen Breakpoint auf, so wird dies im Status-Fenster kenntlich gemacht. Die Angabe der Adresse bezieht sich nur auf den Offset im Code-Segment. Ein gesetzter Unterbrechungspunkt wird mit dem Kommando "-B" wieder gelöscht.

```
Syntax:   B[adr]<CR>

Beispiel: B1200  ; setzt den Breakpoint auf die Adresse 1200H
                 ; im Code-Segment
```

Anmerkung: Breakpoints wirken nicht auf ROM-Adressen!

5.2.3. DOS-Kommando ausführen - C

Dieses Kommando dient zur Ausführung externer Kommandos, z.B. von DOS-Befehlen. Dabei verbleibt FSDEB resident im Hauptspeicher, so daß der komplette Zustand des SBC-86 nach Ausführung des externen Kommandos erhalten bleibt. FSDEB selbst sollte jedoch nicht als Tochterprozeß gestartet werden.

Nach Aufruf des Kommandos C erwartet FSDEB die Eingabe eines externen Kommandos. Wird kein Parameter geliefert, so wird der Kommandoprozessor des Betriebssystems geladen. Dies ist dann sinvoll, wenn mehrere externe Kommandos ausgeführt werden

sollen. Soll jedoch nur ein einziges externes Kommando ausgeführt werden (z.B. das Listen des Inhaltsverzeichnisses der Festplatte), so kann man dieses Kommando (inklusive aller Kommandozeilenparameter) auch direkt angeben. Die Funktionstaste F6 hat die gleiche Wirkung.

```
Syntax:   {C|F6}  ; FSDEB erwartet die Eingabe eines externen
                  ; Kommandos
          Eingabe des externen Kommandos<CR>

Beispiel: C
          DIR A:  ; Listen des Dateiverzeichnisses von
                  ; Laufwerk A
```

Nach Ausführung eines Einzelkommandos übernimmt FSDEB dann wieder die Steuerung, wenn eine beliebige Taste gedrückt wurde. Für den Fall, daß der Kommandoprozessor die Steuerung übernommen hat, ist mit "EXIT" der Kommandoprozessor zu verlassen.

Der Nutzer wird auf erwartete Tastatureingaben jeweils hingewiesen, so daß ein "Abtauchen" in das Betriebssystem ebenfalls durch Online-Hilfetexte unterstützt wird.

5.2.4. Speicherbereich anzeigen - D

Mit Hilfe dieses Befehls kann der Anzeigebereich im Speicher-Fenster geändert werden. Die Adreßangabe bezieht sich auf den Offset im Datensegment. Außerdem hat man die Möglichkeit, mit der Taste "ENDE" (numerischer Zehnerblock) im Speicher-Fenster zu blättern.

```
Syntax:   {D[adr]<CR>|Ende}

Beispiel: D2000  ; Zeigt den Speicherinhalt ab Adresse DS:2000H
```

5.2.5. EXE-File laden - E

EXE-Files sind relokalisierbare Dateien, die durch die Dateierweiterung .EXE gekennzeichnet sind. In jeder dieser Dateien befindet sich eine Relokalisationstabelle, die beim Laden auf zu ändernde Adreßbezüge verweist. Dadurch wird das Programm bezüglich der Speicheradressen verschieblich und kann auf nahezu beliebige Speicheradressen geladen werden. FSDEB verfügt über einen Lader für solche Dateien und führt die Relokalisation prinzipiell auf Adresse 10H:0 aus. Die entsprechenden Prozessorregister tragen nach dem Laden die entsprechenden Werte (CS, IP, SS, SP). Die Erzeugung solcher Dateien weist einige Besonderheiten in der Assemblerprogrammierung auf. Entsprechende Informationen erhält man aus den Handbüchern z.B. von MASM oder TASM.

Da FSDEB ursprünglich nur für Dateien im COM-Format (siehe unten) konzipiert wurde, ergibt sich eine Besonderheit bei der Benutzung von Unterbrechungspunkten. Diese Unterbrechungspunkte werden zum Zeitpunkt der Kommandoauswertung anhand des aktuellen Wertes des Codesegment-Registers und des vom Nutzer gelieferten Offsets in den Programmcode eingesetzt. Da für die Angabe eines Unterbrechungspunktes jedoch nur ein Speicher-Offset erwartet wird, kann das zu Bedienungsfehlern führen. Vergessen Sie deshalb nicht, beim Setzen von Unterbrechungspunkten in einem anderen, als dem aktuellen Codesegment, auch den Wert des Codesegment-Registers manuell einzustellen (Funktion Register ändern).

Übersteigt die Länge des Programmes den verfügbaren Speicher, so wird die Funktion mit einer Fehlermeldung abgebrochen. Das funktioniert jedoch nur, wenn der SBC-86 mit allen Speicherbausteinen bestückt ist. Konnte die EXE-Datei nicht gefunden werden, so generiert FSDEB ebenfalls eine Fehlermeldung.

```
Syntax:   E  ; FSDEB erwartet die Eingabe des Dateinamens
          Eingabe des EXE-Filenamens<CR>

Beispiel: E
          TEST  ; bei dieser Funktion wird zuerst nach dem an-
                ; gegebenen Dateinamen gesucht, ist er nicht
                ; vorhanden wird nach Dateiname+.EXE gesucht
```

5.2.6. Füllen eines Speicherbereiches - F

Sollen mehrere Speicherzellen mit dem gleichen Wert beschrieben werden, so ist dieses Kommando nutzbar. Gefüllt wird dabei immer in Richtung höherer Speicheradressen. Eine Überprüfung der Speichergrenzen erfolgt dabei nicht. Die Startadresse ist datensegmentär.

```
Syntax:   F[von_adr],[anzahl],[füllbyte]<CR>

Beispiel: F1000,20,FF  ; Füllt ab Adresse DS:1000H 32 Speicher-
                       ; zellen mit 0FFH
```

5.2.7. Echtzeitlauf (bis Zieladresse) - G

Dieses Kommando hat wieder eine Doppelfunktion. Erstens kann durch Drücken auf die Funktionstaste F9 ein Echtzeitlauf bewirkt werden. Dieser kann nur durch Breakpoints oder durch Drücken der Reset-Taste angehalten werden. Die zweite Möglichkeit ist die Angabe einer Zieladresse, bei der der Echtzeitlauf unterbrochen werden soll. Diese Funktion wird durch einen temporären Breakpoint erzeugt. Deshalb gilt an dieser Stelle alles, was zu permanenten Breakpoints mit EXE-Files gesagt wurde. Der Unterschied zum permanenten

Unterbrechungspunkt ist der, daß bei Erreichen der Zieladresse FSDEB die Unterbrechnungsadresse wieder "vergißt". Dies aber nur dann, wenn nicht zufällig ein permanenter Breakpoint auf die gleiche Adresse gesetzt war. Der als Parameter erwartete Adreßoffset ist codesegmentär.

Syntax: {G[zieladr]<CR>|F9} ; Goto bzw. Run

Beispiel: G190 ; Programmlauf bis CS:190H

Anmerkung: Goto auf ROM-Adressen angewandt führt zum Run.

5.2.8. Hilfe / Hexadezimalkonverter - H

Als kleine Hilfs-Utility wurde ein Hexadezimalkonverter eingebaut. Die als Parameter gelieferte Hexadezimalzahl wird sowohl dezimal als auch in versuchter ASCII-Darstellung angezeigt. Dies ist dann nützlich, wenn Hexadezimalwerte mal schnell in das Dezimalsystem zu übersetzen sind, um mit dem Quelltext vergleichen zu können, wo möglicherweise Dezimalwerte eingesetzt wurden.

Syntax: H[hexwert]<CR>

Beispiel: H1234 ; Ausgabe: 4460d " 4"ascii

Den eigentlichen Hilfe-Schirm erreicht man durch Drücken der Funktionstaste F1. Die Benutzung der Hilfe-Funktion ist einfach und selbsterklärend.

5.2.9. Eingabeport lesen - I

Mit diesem Kommando kann manuell ein Byte von einem Eingabeport eingelesen werden. Ein wortweises Lesen ist auf Grund der Systemkonfiguration (siehe Hardwarebeschreibung) nicht möglich. Zu beachten ist, daß zwar 16-Bit-Portadressen als Parameter geliefert werden können, jedoch nur die untersten 8 Bit signifikant sind. Dies hat seine Ursache in der unvollständigen Adreßdekodierung, wodurch im System nur 8-Bit-Portadressen Verwendung fanden. Mit der Gültigkeit des Kommandos durch Betätigen der Enter-Taste wird das Ergebnis angezeigt und bleibt bis zum nächsten Tastendruck sichtbar.

Syntax: I[portadr]<CR>

```
Beispiel: IC0  ; Nach Betätigen der Enter-Taste erscheint
               ; IC0=04 (=PIC-ICW1)
```

5.2.10. COM-File laden - L

Diese Funktion dient zum Laden von Speicherabbildern. Solche Dateiformate sind eigentlich geschichtlich überholt, jedoch in DOS-Systemen noch anzutreffen. Für Lehrzwecke eignet sich jedoch das COM-File-Format, da es sehr einfach ist. Es handelt sich um ein Speicherabbild, daß auf Offset 100H in jedes beliebige Codesegment geladen werden kann. Im SBC-86 ist das die Segmentadresse 0. Die Besonderheit bei COM-Files ist, daß Daten, Programmkode und Stapel sich in einem Segment befinden müssen. Intersegmentäre Sprünge sind also unzulässig.

Die Ladefunktion selbst ähnelt dem EXE-Laden, nur daß keine Relokalisation erfolgen muß. Die Registerinitialisierung setzt CS=0, IP=100H und SP=100H, so daß für einfache Testprogramme der Systemstapel verwendet werden kann.

```
Syntax:   L  ; FSDEB erwartet die Eingabe des Dateinamens
          Eingabe des Dateinamens<CR>

Beispiel: L
          TEST  ; bei dieser Funktion wird zuerst nach dem an-
                ; gegebenen Dateinamen gesucht, ist er nicht
                ; vorhanden wird nach Dateiname+.COM gesucht
```

5.2.11. Verschieben eines Speicherbereiches - M

Um Programm- oder Datenbereiche im Speicher verschieben zu können, existiert diese Kopierfunktion. Dabei erfolgt keinerlei Kontrolle, ob sich Quell- oder Zielbereich überlappen oder überschreiben. Die Funktion wird in Richtung aufsteigender Speicheradressen ausgeführt. Die als Adressen erwarteten Parameter sind datensegmentär. Es können maximal 64 kByte Daten (Anzahl = 0) auf einmal kopiert werden.

```
Syntax:   M[von_adr],[nach_adr],[anzahl]<CR>

Beispiel: M100,200,20  ; Kopiert 32 Byte Daten ab DS:100H nach
                       ; DS:200H
```

5.2.12. Nächsten Programmschritt ausführen - N

Dieses Kommando sollte nicht mit dem Einzelschrittbetrieb verwechselt werden. Den "nächsten Programmschritt" auszuführen bedeutet, daß versucht wird, den Befehlszähler auf den Befehl zu setzen, der dem aktuellen Befehl im Codesegment physisch folgt. Da zu Realisation dieser Funktion intern der Breakpoint-Interrupt verwendet wird, funktioniert diese Funktion nur im RAM. Dabei gilt für diese Funktion: Unterprogramme nach CALL-Befehlen werden komplett ausgeführt, ebenso wie Schleifen mit rückwärts gerichteten, bedingten Sprungbefehlen. Für alle anderen Befehle wird ein Einzelschritt ausgeführt. Die Funktionstaste F8 ruft diese Funktion ebenfalls auf.

```
Syntax:   {N|F8}

Beispiel:    .........
          Start:  MOV  CX,1000H      ; Zähler auf 1000H
          Again:  LOOP Again         ; <--- PC
             .........               ; <--- PC nach "Next"
```

Im obigen Beispiel wäre der echte Einzelschrittbetrieb sehr mühsam, weil dann der LOOP-Befehl 1000H-mal auszuführen wäre. Wendet man den Next-Befehl an, so wird die Programmschleife in Echtzeit abgearbeitet.

5.2.13. Ausgabeport schreiben - O

Dieses Kommando erlaubt die Ausgabe eines Bytes auf ein Ausgabeport. Die Besonderheiten der Port-Eingabe sind an dieser Stelle ebenfalls gültig. Die Funktion erwartet als Parameter ein Byte, welches auf die im Befehl spezifizierte Portadresse ausgegeben wird.

```
Syntax:   O[portadr],[ausgabebyte]<CR>

Beispiel: O90,FF  ; Bringt alle Segmente der Siebensegment-
                  ; anzeige an Stelle 0 zum Leuchten
```

5.2.14. DOS-Pfadnamen ändern - P

Soll der aktive DOS-Pfad, der für das Lesen von Dateien, ausführen von Kommandos usw. gültig ist, geändert werden, so ist dieses Kommando anzuwenden. Das Kommando ist prinzipiell dem DOS-Kommando "CD" (Change Directory) äquivalent.

```
Syntax:   P[pfadname]<CR>

Beispiel: P
          FSDEB  ; Wechsel in das Unterverzeichnis \FSDEB
```

Der aktive DOS-Pfad wird jeweils im Status-Fenster angezeigt. Fehleingaben werden mit einer Fehlermeldung belohnt.

5.2.15. Verlassen von FSDEB - Q

Dieser Befehl bewirkt nach Abfrage einen Austritt aus FSDEB und gibt die Steuerung an DOS zurück. Eine Sicherung der aktuell eingestellten Parameter des Debuggers erfolgt dabei nicht.

```
Syntax:   {Q|F10}[{Y|N}]

Beispiel: Q  ; Frage, ob wirklich raus...
          Y  ; bewirkt das Verlassen des Debuggers
```

5.2.16. SBC-86- und FSDEB-Soft-Reset - R

Dieses Kommando wurde einleitend bereits beschrieben. Es dient zum softwaremäßigen Rücksetzen des SBC-86-Monitors, sowie des Full-Screen-Debuggers. Da auch ein Kommunikations-Reset ausgeführt wird, kann eine Asynchronität in der Datenübertragung leicht behoben werden. Zumeist kommt das dann vor, wenn der Reset-Knopf des SBC-86 gedrückt wurde, ohne daß dies der FSDEB automatisch erkannt hat. Ein FSDEB-Befehl (außer natürlich -R-) bringt dann die Synchronität der Datenübertragung aus dem Trott. Deshalb gilt: Nach einem Hardware-Reset sollte sogleich ein Software-Reset durchgeführt werden. Das erübrigt sich nur dann, wenn von FSDEB nach dem Hardware-Reset "Reset detected" gemeldet wurde.

```
Syntax:   R[{Y|N}]

Beispiel: R  ; Frage, ob wirklich Reset
          Y  ; Ja als Antwort
```

5.2.17. Speicher beschreiben - S

Will man manuell Daten oder Programmcode in den Speicher eintragen, so wendet man diese Funktion an. Das "Store"-Kommando erwartet zunächst einen Adreßoffset im Datensegment (ausschließlich!). Dann können ab dieser Adresse einzelne Bytes geändert werden. Zunächst wird der jeweilige alte Speicherinhalt gemeldet, der durch eine Eingabe überschrieben werden kann. Wird die Vorgabe nur mit <CR> beantwortet, so wird der Speicherinhalt nicht geändert. Die ESC-Taste dient zum Abbruch dieser Funktion.

```
Syntax:   S[adr]<CR>
          [{<CR>|<ESC>|neuer_wert<CR>}]

Beispiel: S100  ; Speicher ab Adr. DS:100H beschreiben
          01    ; neuer Wert 01
          02    ; neuer Wert 02
          <CR>  ; alter Wert bleibt bestehen
          03    ; neuer Wert 03
          <ESC> ; fertig
```

5.2.18. Einzelschritt ausführen - T

Will man einen einzigen logischen Programmschritt ausführen, so kann dieses Kommando oder die Funktionstaste F7 verwendet werden. Der Unterschied zwischen dem Einzelschritt und dem Next-Befehl wurde bereits in Punkt 5.2.12. ausführlich beschrieben.

Syntax: {T|F7}

```
Beispiel:    .........
          Start:  MOV  CX,1000H      ; Zähler auf 1000H
          Again:  LOOP Again         ; <--- PC, auch nach "T",
             .........               ; wenn CX<>0
```

5.2.19. Programmcode reassemblieren - U

Im Quellcode-Fenster wird im Normalfall ein versuchtes Reassembling ab Adresse CS:IP angezeigt. Der IP zeigt dabei auf die invers dargestellte Mnemonic. Will man einen anderen Speicherbereich mnemonisch darstellen, so verwendet man dieses Kommando. Mit der Taste PgDn kann man außerdem im Programmspeicher blättern. Drückt man die Enter-Taste, so korrespondiert der angezeigte Speicherbereich erneut mit dem Befehlszeiger. Dadurch kann man schnell Folgebefehle, die aufgrund der Größe des Qellcode-Fensters nicht dargestellt wurden, sichtbar machen und zurück zum aktuellen Befehl schalten.

Syntax: {U[adr]<CR>|PgDn}

```
Beispiel: U300 ; IP derzeit auf CS:100H
          <CR> ; und wieder zurück
```

Es kann nur 8086-Programmcode reassembliert werden. Folgeprozessoren nutzen die Lücken der Dekodiermatrix des 8086, so daß solcher Programmcode nicht mnemonisch darstellbar ist. Nicht reassemblierbarer Code wird als DB xx dargestellt. Dies gilt auch für Koprozessorbefehle.

5.2.20. Programmversion verifizieren - V

Dieses Kommando ist nur zur Kontrolle des Softwareausgabedatums gedacht und ist für die Funktion des Debuggers unbedeutend.

Syntax: V

5.2.21. Register und Flags ändern - X

Alle im CPU-Fenster dargestellten Prozessoregister und Flags können mit diesem Kommando verändert werden. Eine Besonderheit ist das Hilfs-Segmentregister. Es kann beliebig als Zwischenspeicher benutzt werden, vor allem dann, wenn Segmentregister verändert werden müssen, wobei der alte Segmentwert erhalten bleiben soll. Verwenden Sie dieses Register also nur als Hilfsregister und versuchen Sie nicht, es mit Prozessorbefehlen zu verändern.

Die Hauptregister des 8086 sind in zwei 8-Bit-Register teilbar. Auch diese 8-Bit-Register können durch diesen Befehl angesprochen werden. Für die Gesamtheit des Flagregisters wurde die Abkürzung FL verwendet. Alle Flags können auch einzeln angesprochen werden. Dazu ist dem dargestellten Flagnamen ein "F" nachzustellen, d.h. C wird CF usw..

Jedem Register kann der Wert eines passenden (Bitbreite!) anderen Registers, sowie ein Direktwert zugewiesen werden. Einzelflags können nur Werte übertragen werden. Das gesamte Flagregister FL hat den Status eines normalen 16-Bit-Registers.

```
Syntax:    X[{register|flag}]=[{wert|reg}]<CR>

Beispiele: XAX=1234 ; AX:=1234H
           XDS=AX   ; DS:=AX
           XFL=FFFF ; Flags:=0FFFFH
           XBX=FL   ; BX:=Flags
           XCF=0    ; Carry-Flag:=0
           XDF=1    ; Dirction-Flag:=1
           XDH=55   ; DH:=55H
           XCL=DH   ; CL:=DH
```

5.2.22. Displayanzeige umschalten - Z

Dieses Kommando schaltet die Darstellung im Speicherfenster zwischen ASCII- und IBM-Zeichensatz um. Steuerzeichen werden gefiltert und als "." dargestellt. Die Nützlichkeit dieses Kommandos wird man erst bei häufigerem Gebrauch des Debuggers erkennen, insbesondere dann, wenn man Patches auf Maschinenebene ausführen möchte.

5.3. Anmerkungen zu den Editorfunktionen

Auf die Möglichkeit, Kommandos mit Hilfe spezieller Tastatureingaben zu editieren, wurde bereits eingegangen. Hier eine kurze Liste dieser Editiermöglichkeiten. Es wird nur die Bezeichnung der engl. Tastatur verwendet.

```
Pfeil links / rechts       - Cursor nach links / rechts
Pfeil oben / unten         - im Kommandopuffer blättern
```

Backspace	- Zeichen links vom Cursor löschen
Delete	- Zeichen unter Cursor löschen
Ctrl-Pfeil rechts	- Löschen bis Zeilenende
Escape	- ganze Zeile löschen
Home	- Cursor an Zeilenanfang
End	- Cursor an Zeilenende

5.4. Programmbeispiele

An dieser Stelle soll nicht die Bedienung des Debuggers anhand von Programmbeispielen gezeigt, sondern die Benutzung von Betriebssystemfunktionen demonstriert werden. Da jedes einzelne Kommando des Full-Screen-Debuggers bereits beschrieben wurde, könnten diese Programmbeispiele auch dazu dienen, um sich in die Bedienung des Debuggers einzuarbeiten.

Das erste Beispiel soll mit Hilfe des Displayinterrupts eine Zeichenkette ('-SBC-86-') auf der Siebensegmentanzeige des SBC-86 blinken lassen. Hier das einfache Quellprogramm dazu:

```
        .model tiny
        .code
        org 100h
Start:  jmp short Begin

Text    db '-SBC-86-',0         ; ASCIIZ-String

Delay:  push cx                 ; Prozedur zur Zeitverzoegerung
        xor cx,cx               ; Zaehler
        loop $
        loop $
        pop cx
        ret

Begin:  xor ah,ah               ; ClrScr
        int 6
        call Delay              ; und etwas warten
        mov ah,2                ; ASCIIZ-String ausgeben
        mov dl,7                ; ab erster Displaystelle links
        mov bx, offset Text     ; Textadresse
        int 6
        call Delay              ; und wieder warten
        jmp short Begin         ; fuer immer

        end Start
```

Da das Programm kommentiert ist, wird an dieser Stelle auf eine Beschreibung des Programmablaufes verzichtet. Es ist einfach zu einfach, um dafür Platz zu verschwenden.

Das zweite Programmbeispiel ist da etwas anspruchsvoller. Es soll eine einfache, hardwaregesteuerte Uhr realisiert werden.

Dazu sind die Zeitgeberdienste zu verwenden, die Zeitausgabe soll in der Form 'hh mm ss' auf dem Siebensegmentdisplay des SBC-86 erfolgen.

Zunächst einige Überlegungen zur Hard- und Softwareinitialisierung:

1. Programmierung des Zeitgebers als 16-Bit-Rückwärtszähler, Zeitkonstante 16-Bit-binär.

2. Die Bezugsfrequenz des Zeitgebers beträgt 1.8432 MHz. Bei einer Zeitkonstante von 18432 (der Einfachheit halber) ergibt sich eine Frequenz von 100Hz, so daß der Sekundentakt in der Interruptserviceroutine durch einen Vorteiler softwaremäßig erzeugt werden muß.

3. Die Initialisierung des Interruptsystems umfaßt die Komponenten Interrupttabelle, Interruptcontroller und CPU.

Die Displayausgabe soll prinzipiell im Hintergrundprozeß (Hauptprogramm) erfolgen, während die Zeitberechnung in der Interruptserviceroutine erledigt wird. Der Einfachheit halber wird dezimal gezählt, so daß die Displayausgabe direkt, d.h. ohne Hexadezimal-Dezimalwandlung über eine Systemfunktion möglich ist. Hier ein Vorschlag eines lauffähigen Programmes:

```
     .model tiny
     .code
     org 100h

TimeConst = 18432
InTab = 20h

Start: xor ax,ax
       mov ds,ax           ; IMMED geht leider nicht mit SEGREG
                    call Init
       mov ah,0                ; ClrScr
       int 6
Again: mov bl,[cs:Hour]        ; Stunden ausgeben
       mov ah,4
       mov dl,7
       int 6
       mov bl,[cs:Min]         ; Minuten ausgeben
       mov ah,4
       mov dl,4
       int 6
       mov bl,[cs:Sec]         ; Sekunden ausgeben
       mov ah,4
       mov dl,1
       int 6
       jmp short Again         ; und immer wieder
```

```
Init:  cli
       mov al,01110110b        ; PIT, Kanal 1, Mode 3, 16-Bit-
; binaer
       out 0a6h,al
       mov al,LOW(TimeConst)   ; erst niederwertiger Teil
       out 0a2h,al
       mov al,HIGH(TimeConst)  ; dann der hoeherwertige Teil
       out 0a2h,al
       mov al,11111110b        ; Kanal 0, PIC freigeben
       out 0c2h,al

; jetzt ein umstaendlicher Weg, aber die Assembler tun sich
; schwer mit direkter Adressierung beim Tiny-Model

       mov bx,InTab            ; Adressbasis Int.-Tabelle
       mov [bx],offset ISR     ; Adr. ISR in Int.-Tabelle
       mov [bx+2],cs
       sti
       ret
ISR:   push ax
       inc [cs:Teiler]         ; Softwareteiler /100
       cmp [cs:Teiler],100     ; schon 100 ?
       jnz I1
       mov [cs:Teiler],0       ; ja, dann von vorne
       mov al,[cs:Sec]
       add al,1                ; Sekunden dezimal +1
       daa
       mov [cs:Sec],al
       cmp al,60h              ; schon 60 ?
       jnz I1
       mov [cs:Sec],0          ; und wieder von vorne
       mov al,[cs:Min]         ; das gleiche fuer die Minuten
       add al,1
       daa
       mov [cs:Min],al
       cmp al,60h
       jnz I1
       mov [cs:Min],0
       mov al,[cs:Hour]        ; und nochmal bei den Stunden
       add al,1
       daa
       mov [cs:Hour],al
       cmp al,24h
       jnz I1
       mov [cs:Hour],0
I1:    mov al,20h              ;EOI an PIC
       out 0c0h,al
       pop ax
       iret

Sec    db 0                    ; Sekunden
Min    db 0                    ; Minuten
```

```
Hour   db 0                    ; Stunden
Teiler db 0                    ; Softwareteiler /100

       end Start
```

Will man das Programm zunächst im Einzelschrittbetrieb testen, so sollte man beachten, daß nach einem Segmentregister-Transportbefehl (hier mov ds,ax) keine Programmunterbrechung möglich ist, also der Einzelschritt dieses Befehls auch gleich den nächsten Befehl mit ausführt! Dies nur als Tip, da man ohne diese Kenntnis leicht andere Ursachen sucht...

6. Schlußbemerkungen

Der Full-Screen-Debugger wurde bis jetzt auf folgenden Betriebssystemen eingesetzt, ohne daß Probleme auftraten:

MS-DOS V3.x, V4.x, V5.0 und OS/2 V2.0 .

Für die Zukunft wird sich am derzeitigen Funktionsumfang des FSDEB wohl wenig ändern. Das Konzept hat sich in der Ausbildung im Lehrfach Mikroprozessortechnik bewährt. Eine "zeitgemäße" Bedieneroberfläche würde die Bedienung des Debuggers nur komplizierter gestalten.

7. Literaturverzeichnis

J.W. Coffron: Programmierung des 8086/8088
Sybex-Verlag GmbH Düsseldorf 1984

H. Stein: Der 8086 in der Praxis
Verlag Markt & Technik München 1982

H.-J. Blank, H. Bernstein: PC-Schaltungstechnik in der Praxis
Markt & Technik Verlag AG 1989

H. Bernstein: Hardware-Handbuch PC-XT-AT und Kompatible
Markt & Technik Verlag AG 1990