Institut für Industrielle Informationstechnik (IIIT)
Universität (TH) Karlsruhe

Prof. Dr.-Ing. Uwe Kiencke
Prof. Dr.-Ing. habil. K. Dostert

Hertzstr. 16 / Geb. 06.35
76187 Karlsruhe
Tel.: 0721 608 4521
Fax: 0721 608 4500

Praktikum: **„Mikrocontroller und digitale Signalprozessoren“**

Versuche 1+2

Einführung in den Mikrocontroller 80C517A

Inhaltsverzeichnis

1 Einleitung

Die ersten beiden Praktikumsversuche zur digitalen Drehzahlmessung und Frequenzsynthese werden mit dem Platinenboard „Micky(PL) 1.0“ durchgeführt. Es dient als Basisplatine zum Minimodul-537 der Firma Phytec, auf dem sich ein Mikrocontroller vom Typ SAB80C517A der Firma Siemens befindet. Des weiteren ist auf diesem Modul der externe Speicher des Mikrocontrollers untergebracht. Zur Kommunikation mit einem PC besitzt das Board eine serielle asynchrone Schnittstelle. Programme können so auf dem PC erstellt und anschließend in den Programmspeicher des Mikrocontrollers geschrieben werden. Die zahlreichen Ein- und Ausgänge des eingesetzten Mikrocontrollers werden auf der Basisplatine herausgeführt. Über vier auf der Platine integrierte LEDs können programminterne Zustände des Mikrocontrollers angezeigt werden. Die LEDs können über die unteren vier Bits des Port 5 angesprochen werden. Die Platine wird von einem Netzteil mit 12V Gleichspannung versorgt.

Im folgenden soll Ihnen nun ein kurzer Einblick in die einzelnen Komponenten des Versuchsaufbaus gegeben werden. Es wird zuerst der Mikrocontroller 80C517A mit seinen verschiedenen Hardwareeinheiten vorgestellt und anschließend auf die angebundene zusätzliche Hardware wie LCD (**L**iquid **C**rystal **D**isplay) und D/A-Wandler eingegangen. Das letzte Kapitel gibt eine Einführung in die am Versuchstag zur Verfügung gestellte Entwicklungsumgebung.

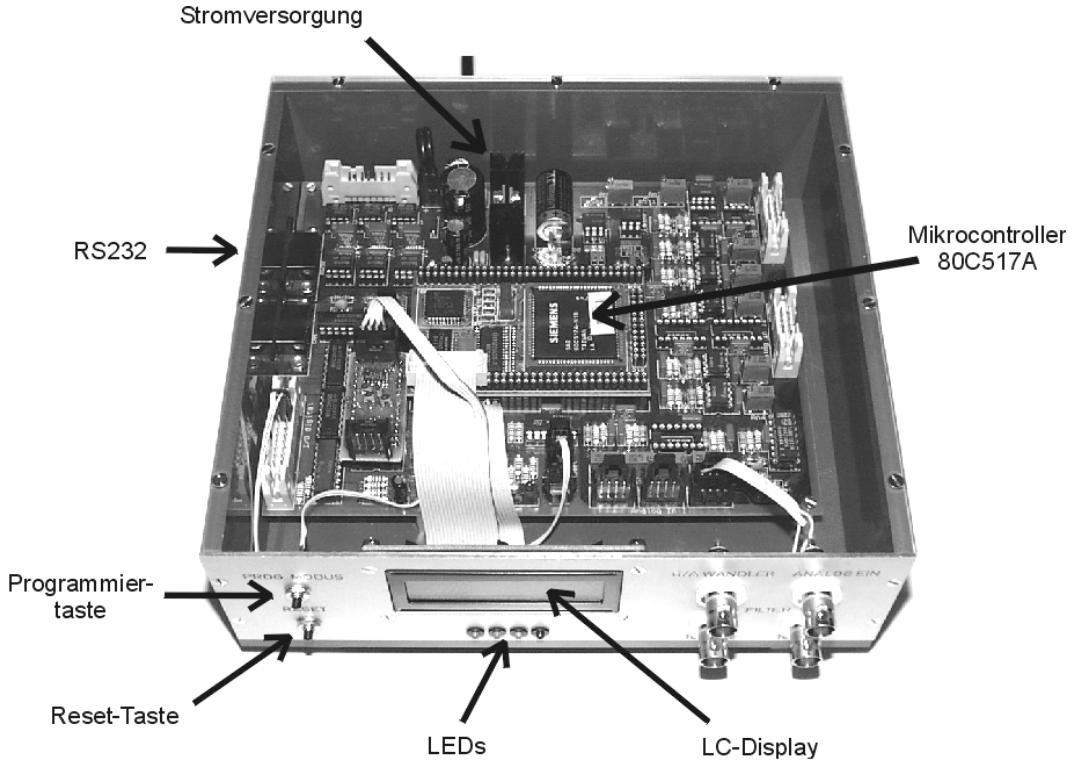

Abbildung 1 Hardware des Praktikumsversuchs

2 Hardwarebeschreibung

2.1 Der Mikrocontroller 80C517A

Für die Lösung der digitalen Signalverarbeitungsaufgaben in den Versuchen 1 und 2 wird der Mikrocontroller 80C517A der Firma Siemens verwendet. Er ist ein High-End-Mikrocontroller, der abwärtskompatibel zur verbreiteten 8bit 8051-Familie ist. Die Architektur des Chips wurde um zusätzliche Komponenten wie einen 10bit A/D-Wandler und eine leistungsfähige Capture-Compare-Einheit erweitert. Für das Praktikum werden nicht alle Funktionen dieses komplexen Bausteins benötigt. Abbildung 2 zeigt den Aufbau des Mikrocontrollers.

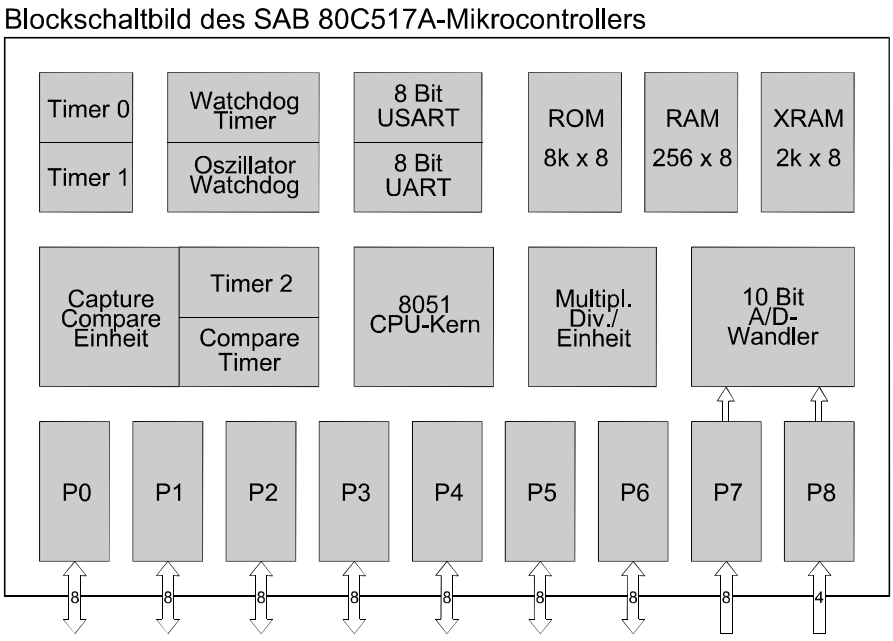

Abbildung 2 Aufbau des Mikrocontrollers 80C517A

Der Mikrocontroller SAB80C517A zeichnet sich durch folgende Leistungsmerkmale aus:

- 56 bidirektionale I/O-Ports und 12 analoge Eingänge
- 256 Byte internes RAM und 2 KByte "On Chip" XRAM
- 256 direkt adressierbare Bits
- 8 Datenzeiger für die Adressierung des externen Speichers
- 17 Interruptquellen, 4 Prioritätsebenen

- 16bit Capture-Compare-Unit (CCU)
- 10bit A/D-Wandler mit 12 gemultiplexten Eingangskanälen (Konversionszeit 12,4 µs)
- 32bit Multiplikations-/Divisionseinheit
- 2 serielle Schnittstellen mit variabler Baudratengenerierung

Der Mikrocontroller wird mit einer Oszillatorfrequenz von 18MHz getaktet. Er besitzt einen umfangreichen Befehlssatz, der im Benutzerhandbuch ab Seite 208 nachgeschlagen werden kann. Die Abarbeitung der Befehle unterteilt sich in Maschinenzyklen, wobei jeder Maschinenzyklus aus zwölf Oszillatorperioden besteht. Die meisten Befehle des SAB80C517A können in einem Maschinenzyklus ausgeführt werden. Andere Befehle wie z.B. MUL und DIV benötigen dagegen bis zu vier Zyklen für ihre Abarbeitung.

2.1.1 Das Registermodell

Akkumulator (A)

Das Register A ist das Standardregister und wird bei der Durchführung sämtlicher arithmetischer und logischer Befehle verwendet. Es hat eine Breite von 8 Bit.

Hilfsregister (B)

Dies ist ein frei benutzbares Hilfsregister, das unter anderem bei der Division benötigt wird.

Stackpointer (SP)

Der Stackpointer ist ein 8bit-Register, das immer auf den obersten Wert des Stapelspeichers zeigt. Er kann nur durch die Befehle PUSH und POP verändert werden.

Programmstatusregister (PSW)

Dieses 8bit-Register enthält alle zur Verfügung stehenden Kennzeichenbits, u.a. das Carry-Flag und das Overflow-Flag.

Befehlszähler (PC)

Im 16bit breiten Programmzählregister steht immer die Adresse des nächsten abzuarbeitenden Befehls.

Datenzeiger (DPTR)

Das DPTR-Register ist ein 16bit-Register, das sich aus zwei 8bit-Registern zusammensetzt. DPH bildet das Highbyte, DPL das Lowbyte. Der Datenzeiger wird zur Adressierung des externen Daten- und Programmspeichers verwendet. Er kann auch direkt mit einem 16bit-Wert geladen werden.

Registersatz R0-R7

Die Register sind jeweils 8bit breit und können frei verwendet werden. Den Registern R0 und R1 kommt dabei besondere Bedeutung zu, da die indirekte Adressierung nur mit R0, R1 oder mit dem Datenzeiger DPTR durchgeführt werden kann. Eine Registerbank kann mittels zweier Bankselect-Bits (Bits RS0 und RS1 im PSW) aktiviert werden. Für die in diesem

Versuch zu lösenden Aufgaben reichen jedoch die acht Register der voreingestellten Bank 0 völlig aus.

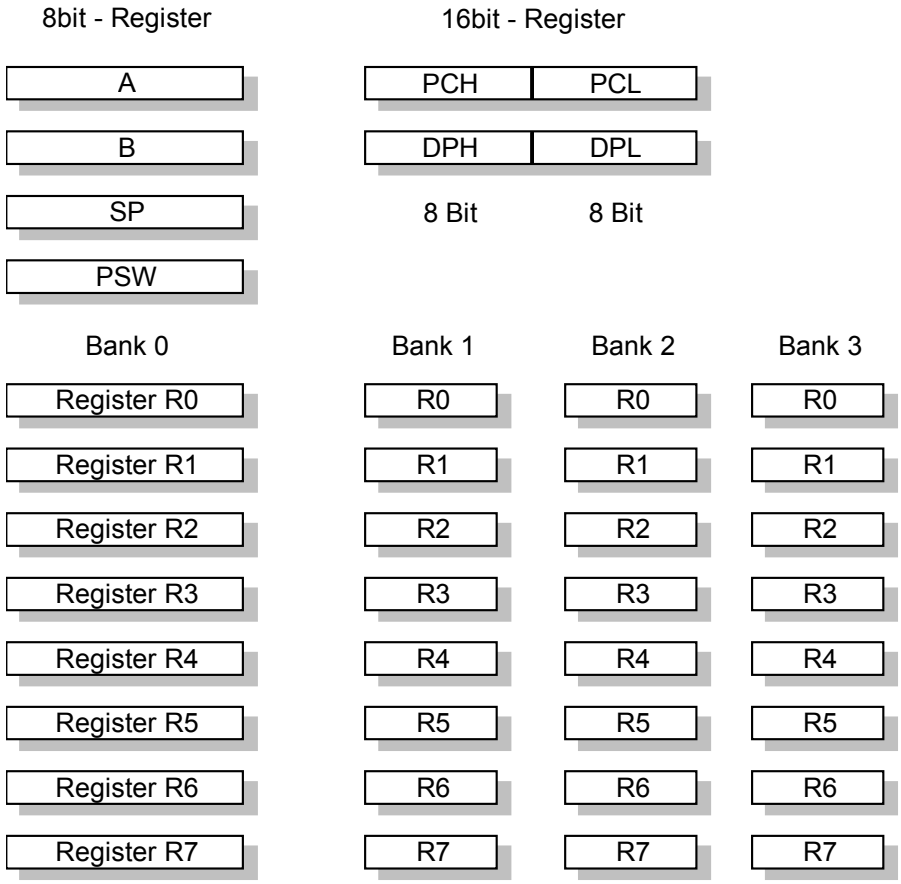

Abbildung 3 Die Register des SAB80C517A

2.1.2 Das Speicherkonzept

Der Speicher teilt sich auf in einen internen und einen externen Anteil.

Der interne Speicher

Der Mikrocontroller 80C517A verfügt über 256 Byte internen Datenspeicher. Das interne RAM kann im Bereich 00h bis 7Fh direkt oder indirekt adressiert werden (siehe 3.1.4 Speicheradressierung). Es besteht aus acht 8bit-Registern, 16 bitadressierbaren Byte (128 Bit) und 80 frei verfügbaren Byte. Das obere interne RAM im Adreßbereich 80h bis FFh enthält Datenbytes, die ebenfalls frei benutzt werden können. Parallel zu diesem Adreßbereich befinden sich 128 Bytes, in denen die Spezialfunktionsregister untergebracht sind. Die Spezialfunktionsregister (SFR) stellen die Verbindung zu der integrierten Peripherie des Controllers wie z.B. Timern oder Ports her. Der gesamte Kontroll- und Datenfluß der Peripherie wird ausschließlich über diese Register abgewickelt. Die SFRs sind im Benutzerhandbuch ab Seite 38 beschrieben.

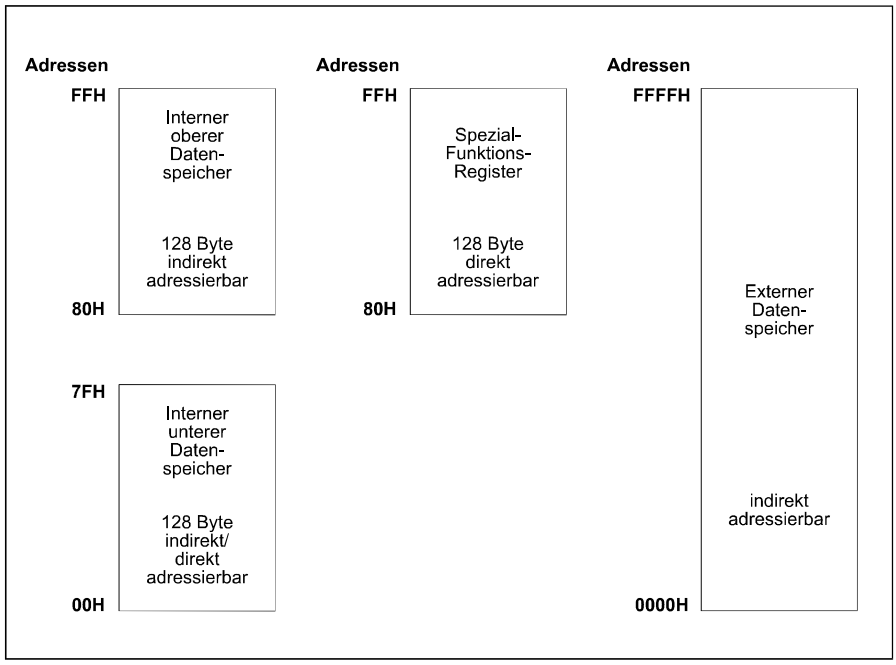

Abbildung 4 Datenspeicher des 80C517A

Die oberen 128 Bytes des internen Datenspeichers teilen sich somit in zwei Funktionsbereiche auf, die lediglich durch die Art der Adressierung unterschieden werden. Wird der Bereich 80h bis FFh direkt adressiert, z.B. durch den Befehl `mov 80h,#01`, so erreicht man die Spezialfunktionsregister. In diesem Speicherbereich befinden sich auch die CPU-Register A, B, DPH und DPL etc. Die parallel zu den SFRs liegenden Datenbytes können nur mit Befehlen erreicht werden, die indirekt adressieren, z.B. durch den Befehl `mov @R0,A` , wobei in R0 die Speicheradresse steht. Die Datenbytes besitzen jedoch die gleichen Adressen wie die SFRs. Insgesamt stehen 81 SFR zur Verfügung, von denen 15 zusätzlich bitadressierbar sind.

Der externe Speicher

Der externe Speicher ist an den Adreß- und Datenbus des Mikrocontrollers angebunden. Er gliedert sich in einen Codespeicher, in dem das Programm abgelegt wird, und einen Datenspeicher, der dem Programmierer uneingeschränkt zur Verfügung steht. Code- und Datenspeicher haben jeweils eine Größe von 64 kByte. Auf den externen Speicher kann nur mittels indirekter Adressierung zugegriffen werden, es ist also stets ein Zeiger zu verwenden. Mit den 8bit-Registern R0 und R1 können nur die unteren 256 Bytes des Adreßraumes erreicht werden. Aus diesem Grund ist für die Adressierung des externen Speichers der Datenzeiger DPTR vorgesehen. Da dieser 16bit breit ist, läßt sich mit ihm der gesamte 64 kByte Speicher adressieren.

Als Programmspeicher kommt ein externes Flash-EEPROM zum Einsatz. Die Flash-Speichertechnologie erlaubt es, den Programmspeicher wie ein EPROM zu beschreiben und wie ein EEPROM zu löschen. Dies hat den Vorteil, daß er bei einer Neuprogrammierung nicht entnommen werden muß. Der Bereich 0000h bis 00ABh des Programmspeichers ist für die Interrupteinsprungvektoren reserviert und darf nicht überschrieben werden. Im Anhang A

befindet sich eine detaillierte Speicherbelegung sowie eine Übersicht der Interrupt-Vektoradressen.

2.1.3 Die Ports

Der Mikrocontroller 80C517A verfügt über neun Ports, auf die mit Hilfe der Spezialfunktionsregister P0 bis P8 zugegriffen wird. Bei den Ports P0 bis P6 handelt es sich um 8bit breite bidirektionale I/O-Ports, wobei P0 und P2 für den Adreß-/Datenbus reserviert sind. Neben diesen digitalen Ports besitzt der Mikrocontroller zwei analoge Ports P7 und P8. An diesen Ports sind die 12 Kanäle des A/D-Wandlers herausgeführt. Den meisten Portpins ist eine alternative Funktion zugeordnet. Dabei handelt es sich um Ein- und Ausgangsleitungen von Peripherieeinheiten wie beispielsweise Timern, seriellen Schnittstellen oder dem Interruptsystem. Die für das Praktikum benötigten Portpins sind in Tabelle 1 aufgelistet.

Port	**Pin**	**Alternative Funktion**
P3.4	T0	Zähleingang Timer 0
P1.1	INT4/CC1	Externer Interrupt 4/Capture-Compare-Register 1
P7.0		A/D-Kanal 0

Tabelle 1 Die Portpins und ihre alternative Funktion

2.1.4 Timer 0 und Timer 1

Die beiden Timer 0 und 1 sind in ihrem Aufbau identisch. Sie unterteilen sich in zwei 8bit-Register, TH0 und TL0 bei Timer 0 bzw. TH1 und TL1 bei Timer1 und können als Zähler oder Zeitgeber programmiert werden. Die Betriebsart kann im SFR TMOD eingestellt werden. In der Betriebsart als Zeitgeber wird das Zählregister in jedem Maschinenzyklus um eins erhöht. Da ein Maschinenzyklus aus zwölf Oszillatorperioden besteht, beträgt die Zählrate 1/12 der Oszillatorfrequenz. In der Betriebsart als Zähler erfolgt ein Zählschritt bei jeder fallenden Flanke am entsprechenden Eingangspin. Für Timer0 ist Pin 3.4 der Zählereingang, für Timer1 ist es Pin 3.5.

In der Betriebsart 1 arbeitet Timer0 als 16bit-Timer/Zähler, in der Betriebsart 2 im 8bit-Autoreload. Im Autoreload-Modus besteht das Zählregister nur aus TL0. Bei einem Überlauf dieses Registers, also dem Übergang vom Zählerstand FFh zu 00h, wird der Wert des Registers TH0 in TL0 geschrieben. Dies geschieht automatisch durch die Hardware. Soll der Zähler nach der Zeit *t* überlaufen, so berechnet sich der Reloadwert für TH0 zu:

$$\text{Reloadwert} = 256 - t * \frac{f_{OSZ}}{12}$$

2.1.5 Das Interruptsystem

Der SAB80C517A besitzt ein umfangreiches Interruptsystem mit sieben externen Interruptleitungen. Interrupts können durch steigende oder fallende Flanken an den Interrupteingängen oder durch die interne Peripherie, z.B. durch einen Timer-Überlauf oder das Ende eines Wandlungsvorgangs im A/D-Wandler, ausgelöst werden. Tritt ein solches Ereignis auf, wird ein speziell ihm zugeordnetes Bit, das Interrupt-Request-Flag, gesetzt. Eine Interruptanforderung wird jedoch nur bearbeitet, falls der Interrupt zugelassen ist. Jeder Interrupt kann individuell zugelassen und gesperrt werden, indem das korrespondierende Interrupt-Enable-Flag gesetzt bzw. gelöscht wird. Zusätzlich können alle Interrupts gemeinsam gesperrt werden, indem das globale Interrupt-Flag EAL gelöscht wird.

Eine Unterbrechungsanforderung veranlaßt den Mikrocontroller, die momentane Befehlssequenz zu unterbrechen und eine Interruptserviceroutine (ISR) auszuführen. Zunächst wird die Rücksprungadresse auf den Stack gerettet. Danach verzweigt der Programmlauf auf eine fest zugeordnete Adresse im Codespeicherbereich 0003h bis 00ABh, die sogenannte Interrupt-Vektoradresse. An dieser Adresse steht ein Sprungbefehl zu der eigentlichen Interruptserviceroutine, die an einem beliebigen Platz im Codespeicher stehen kann. Eine Übersicht der Interrupt-Vektoradressen des SAB80C517A finden Sie im Anhang. Nach Abarbeitung der ISR wird wieder zum ursprünglichen Programm zurückgekehrt. Das Interrupt-Flag, das für die Auslösung eines Interrupts verantwortlich war, wird bei einigen Interruptquellen automatisch zurückgesetzt, bei anderen muß dies softwaremäßig erfolgen. Geschieht dies nicht, wird direkt nach Abarbeitung der Interruptroutine erneut ein Interrupt ausgelöst und das Programm „hängt“ sich in der ISR auf. Die Ausführung der Interruptroutine ist vergleichbar mit dem Aufruf eines Unterprogramms. Der Unterschied besteht darin, daß eine Interruptroutine mit dem Befehl RETI (Return from Interrupt) anstatt RET (Return from Subroutine) abgeschlossen werden muß. An diesem Befehl erkennt der Prozessor das Ende der ISR, restauriert die zuvor gesicherten Stackinformationen und kehrt ordnungsgemäß ins Hauptprogramm zurück.

Sind mehrere Interruptquellen zugelassen, kann es bei ihrer Abarbeitung zu Überschneidungen kommen. Einerseits kann ein Interrupt durch einen anderen unterbrochen werden, andererseits können zwei Interrupts gleichzeitig ausgelöst werden. Um dies zu verhindern, werden den Interruptquellen Prioritäten zugeordnet. Die Interruptquellen sind in Gruppen zu je drei Interrupts zusammengefaßt. Jeder Gruppe kann eine der vier Prioritätsstufen zugewiesen werden. Dies geschieht durch Setzen der entsprechenden Bits in den Registern IP0 und IP1. Die höchste Priorität für einen Interrupt ist die Ebene 3, die niedrigste die Ebene 0. Interrupts, die einer höheren Ebene zugeordnet sind, können nicht von Interrupts niedrigerer Ebenen unterbrochen werden. Sie werden abgewiesen.

Interrupts gleicher Ebene werden nach einer prozessorintern festgelegten Abarbeitungsfolge angenommen, die in Tabelle 2 dargestellt ist. Jeweils drei Interruptquellen sind zu einer Gruppe zusammengefaßt. Die Priorität innerhalb einer Gruppe nimmt von links nach rechts ab, die Priorität unter den Gruppen nimmt von oben nach unten ab.

Priorität	Hoch	→	Niedrig
Hoch	Externer Interrupt 0	Serielle Schnittstelle 1	A/D-Wandler Interrupt
	Timer 0 Interrupt	——	Externer Interrupt 2
	Externer Interrupt 1	CM0-CM7-Interrupt	Externer Interrupt 3
↓	Timer1 Interrupt	Compare-Timer Interrupt	Externer Interrupt 4
	Serielle Schnittstelle 0	COMSET Interrupt	Externer Interrupt 5
Niedrig	Timer 2 Interrupt	COMCLR Interrupt	Externer Interrupt 6

Tabelle 2 Priorität für Interruptquellen gleicher Ebene

Timer0 und Timer1-Interrupt

Diese Interrupts werden durch Überläufe der beiden Timer ausgelöst, also beim Übergang des Zählerstandes von FFh auf 00h. Die zugehörigen Interrupt-Request-Flags werden automatisch zurückgesetzt.

Der A/D-Wandler-Interrupt

Dieser Interrupt wird vom A/D-Wandler ausgelöst, wenn das Ergebnis einer Wandlung vorliegt. Das zugehörige Interrupt-Flag IADC muß per Software zurückgesetzt werden. Ist der Wandler auf kontinuierliche Wandlung eingestellt, so wird das IADC-Flag nach jedem Wandlungszyklus gesetzt.

Externer Interrupt 4

Dieser Interrupt wird durch eine positive Flanke am Pin INT4 ausgelöst und setzt das Interrupt-Request-Flag IEX4. Dieses Flag wird durch die Hardware automatisch zurückgesetzt.

Timer2-Interrupt

Dieser Interrupt kann von zwei verschiedenen Ereignissen erzeugt werden. Der Überlauf von Timer2 setzt das Interrupt-Request-Flag TF2, eine negative Flanke am Pin T2EX setzt das Interrupt-Request-Flag EXF2. Beide Ereignisse verzweigen auf dieselbe Interruptroutine (002Bh). Um die Interruptquelle in der Serviceroutine erkennen zu können, werden die Flags nicht automatisch zurückgesetzt. Dieses Flag muß softwaremäßig vor Ende der Interruptroutine zurückgesetzt werden.

2.1.6 Die Capture/Compare-Einheit

Eine der leistungsstärksten Einheiten des 80C517A ist die Capture/Compare-Einheit (CCU). Sie stellt ein komfortables Hilfsmittel zur Verarbeitung externer Ereignisse und zur Erzeugung von digitalen Signalen wie z.B. Impulsfolgen oder Pulsweitenmodulation dar. Als Zeitbasis dienen zwei Timer, der Compare-Timer und Timer2. Beide Timer haben eine Breite von 16 Bit. Da im Praktikum der Compare-Timer nicht benötigt wird, wird im folgenden nur die Funktion des Timer2 beschrieben.

Die maximale Taktfrequenz von Timer2 beträgt $f_{OSZ}/12$. Timer2 bietet in seinen Standardfunktionen eine Vielzahl von Steuerungsmöglichkeiten. Er besteht aus einem 16bit-Zählregister (TH2, TL2), das als intern gesteuerter Zeitgeber (Timer), extern freigegebener Zeitgeber (gated timer) oder als extern getakteter Zähler (event counter) programmiert werden kann. Die Betriebsart des Timer2 läßt sich im Spezialfunktionsregister T2CON einstellen. Die Erhöhung des Zählerstandes kann entweder über 1/12 oder 1/24 der Oszillatorfrequenz erfolgen, d.h. mit jedem Maschinenzyklus oder nur mit jedem zweiten. Die Einstellung erfolgt durch das Bit T2PS im SFR T2CON. Wird Timer2 im Zählermodus verwendet, muß das Bit T2PS1 im Register CTCON zu Null gesetzt werden. Timer2 besitzt einen Reloadmodus. Bei einem Überlauf wird der Inhalt des Registers CRCL und CRCH automatisch in das Zählregister geladen. Dieser Überlauf ermöglicht es, Interrupts in äquidistanten Zeitabständen auszulösen. Die Zeit zwischen zwei Interrupts berechnet sich zu

$$t_{Overflow} = \frac{65536 - (CRCL + 256 * CRCH)}{f_{OSZ}/12} * 2^{(T2PS+2*T2PS1)}$$

Der Autoreload kann alternativ auch durch ein externes Signal ausgelöst werden. Die CCU besitzt zwei Funktionen, die Compare- und die Capturefunktion. Im folgenden wird die in Versuch 1 benötigte Capture-Funktion beschrieben.

In dieser Betriebsart dient Timer2 als Zeitbasis. Bei Eintritt eines externen Ereignisses wird der momentane Zählerstand in das zugeordnete Capture-Register geschrieben und dort gespeichert. Dieses Ereignis kann z.B. eine steigende Flanke an Pin P1.1 (CC1) sein. Zusätzlich wird bei einem Capture-Ereignis das entsprechende Interrupt-Flag gesetzt. Ist der zugehörige Input-Capture-Interrupt freigegeben, so wird dieser bei jeder aktiven Flanke am Triggereingang ausgelöst. In der Interrupt-Routine kann nun der Inhalt des Capture-Registers ausgelesen und verarbeitet werden.

Timer2 ist mit den Capture/Compare-Registern CC1 bis CC4 und dem CRC-Register verknüpft. Somit können bis zu fünf Signale gleichzeitig ausgemessen werden. Die Funktion der CCx-Register wird im SFR CCEN eingestellt. Mit der Capture/Compare-Einheit können somit ohne großen Softwareaufwand Zeitdifferenzen zwischen Signalen ausgemessen werden. Bei der Periodendauermessung z.B. ist die zu messende Zeitdauer durch den Abstand zweier Signalflanken gleicher Polarität (entweder steigende oder fallende Flanke) vorgegeben.

2.1.7 Der A/D-Wandler

Der integrierte A/D-Wandler des Mikrocontrollers 80C517A besitzt 12 gemultiplexte Eingangsleitungen bei einer Auflösung von 10 Bit. Er arbeitet nach dem Prinzip der sukzessiven Approximation und besitzt eine Wandlungszeit von ca. 12,4 µs. Der maximale Fehler einer Wandlung beträgt ±2 LSB (Least Significant Bits). Die Analogeingänge sind an den Ports 7 und 8 herausgeführt.

Vor einer Wandlung muß der Eingangskanal gewählt werden, der in den Registern ADCON0 oder ADCON1 eingestellt werden kann. Durch einen Schreibzugriff auf das Register ADDATL wird eine Wandlung gestartet. Das BSY-Flag wird im nächsten Zyklus nach Eintreffen des Startsignals gesetzt und mit dem Ende der A/D-Wandlung wieder zurückgesetzt. Danach kann das Ergebnis aus den Registern ADDATH und ADDATL

ausgelesen werden. Für das Praktikum genügen die oberen 8 Bits des Wandlungsergebnisses, welche im ADDATH-Register stehen. Die nächste Wandlung beginnt mit einem erneuten Schreibzugriff auf ADDATL.

2.1.8 Die Multiplikations-/Divisionseinheit

Der 80C517A besitzt eine 32bit Arithmetikeinheit, mit der schnelle Multiplikationen und Divisionen durchgeführt werden können. Während die Befehle MUL und DIV nur 8bit-Operanden zulassen, unterstützt die Arithmetikeinheit 32bit-Division und 16bit-Multiplikation. Sie ist als Coprozessor ausgeführt und arbeitet unabhängig von der CPU. Für die Operanden stehen die SFR MD0 bis MD5 zur Verfügung. Die Berechnung kann zwischen 4 und 6 Maschinenzyklen benötigen. In dieser Zeit kann der Mikrocontroller andere Aufgaben erledigen. Erst danach kann das Ergebnis aus den Registern MD0 bis MD5 gelesen werden.

2.2 Die Anzeigeneinheit

Zur Anzeige von Meßergebnissen des Mikrocontrollers und zur Ausgabe von Meldungen an den Benutzer ist das Board mit einem Anzeigenmodul (LCD) ausgestattet. Es ist über einen Adreßdekoder an den Controllerbus angebunden und wird wie ein externer RAM-Baustein angesprochen. Die Integration in den Adreß-/Datenbus hat den Vorteil, daß kostbare Portanschlüsse eingespart werden und die Ansteuerung des Displays durch die Software wesentlich schneller und komfortabler erfolgen kann. Das Anzeigenmodul besteht aus einer LCD-Punkt-Matrix, einem Grafikcontroller, einem Anzeigenspeicher und einer integrierten Ansteuerlogik. Es ist in der Lage, zwei Zeilen mit jeweils 16 Zeichen darzustellen. Der Grafikcontroller besitzt ein Charakter-ROM, in dem die Zeichencodes der darstellbaren Zeichen gespeichert sind. Ein Zeichen setzt sich aus einer 5x7-Punktmatrix zusammen.

Im Anzeigenspeicher stehen die Codes der Zeichen, die auf dem Display dargestellt werden sollen. Er besitzt eine Größe von 80 Bytes. Jedes Byte repräsentiert einen Anzeigenplatz auf dem Display, es bildet also einen Vektor in das Charakter-ROM, von wo sich der Grafikcontroller das entsprechende Muster für das Zeichen holt. Der Zeichensatz umfaßt mit kleinen Ausnahmen den ASCII-Code (Zeichencodes 20h bis 7Dh). Der Adreßbereich erstreckt sich für die 16 Zeichen der ersten Zeile von 00h bis 0Fh, für die 16 Zeichen der zweiten Zeile von 40h bis 4Fh.

Der Grafikcontroller besitzt einige Befehle, mit denen er die Anzeige steuern kann, z.B. um das Display ein- und auszuschalten oder den Anzeigenspeicher zu löschen (siehe Anhang). Neben den Befehlsbytes empfängt er Datenbytes. Zur Unterscheidung werden die Leitungen RS (Register Select) und R/W (Read/Write) verwendet, die an die Adreßleitungen A0 und A1 des Adreßbusses angeschlossen sind. Für die Ansteuerung der Anzeigeneinheit sind die Adressen FE00h bis FE03h im I/O-Adreßbereich des Mikrocontrollers reserviert. Die Ankopplung des Displays an den Adreß-/Datenbus zeigt Abbildung 5.

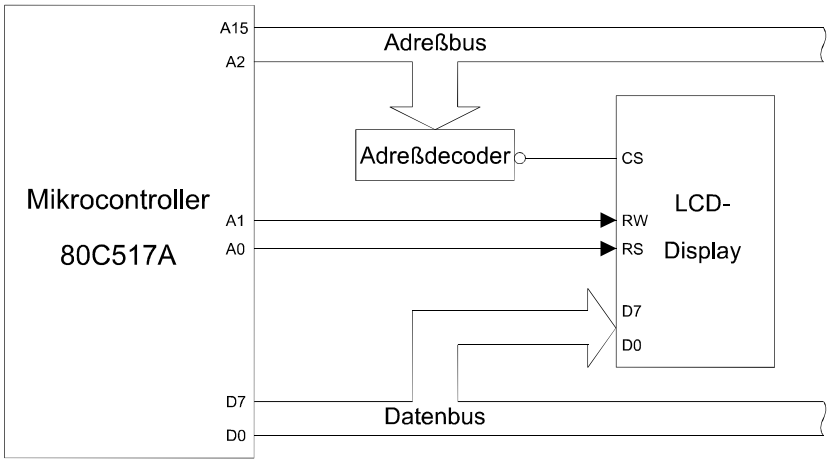

Abbildung 5 Anbindung des LCD-Displays an den Adreß-/Datenbus des 80C517A

2.3 Der D/A-Wandler

Zur Ausgabe von analogen Spannungen wird der D/A-Wandler AD7801 der Firma Analog Devices verwendet. Dabei handelt es sich um einen Wandlerbaustein mit 8bit Auflösung und einer Genauigkeit von ±1 LSB. Er liefert eine Ausgangsspannung von 0V bis 5V und besitzt eine Wandlungszeit von 2µs.

Der D/A-Wandler ist wie die Anzeigeneinheit in den Adreß-/Datenbus des Mikrocontrollers 80C517A integriert und wird wie ein RAM-Baustein angesprochen. Er erhält die Datenbytes vom 8bit-Datenbus des Mikrocontrollers und wird über den Adreßdekoder durch einen Schreibzugriff auf die I/O-Adresse FF00h des externen Datenspeichers angesprochen.

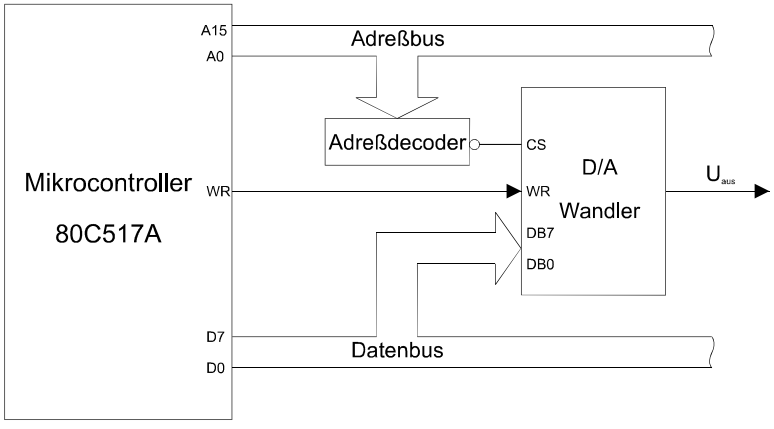

Abbildung 6 Anbindung des D/A-Wandlers an den Adreß-/Datenbus des 80C517A

3 Die Entwicklungsumgebung

In diesem Kapitel wird die Ihnen zur Verfügung stehende Software zur Erstellung der Programme für den Mikrocontroller vorgestellt. Die Softwareentwicklung für den Mikrocontroller 80C517A geschieht in Assembler. Im folgenden sollen einige Besonderheiten der Assemblerprogrammierung hervorgehoben und eine kurze Einführung in die Programmiersprache gegeben werden. Eine detaillierte Beschreibung würde den Rahmen dieses Versuches sprengen. Es sei auf einschlägige Literatur zu diesem Thema verwiesen.

3.1 Aufbau eines Assemblerprogramms

3.1.1 Befehlsformat

Der Programmcode besteht aus Assemblerbefehlen, die der Assembler beim Compilieren in Maschinencode übersetzt und diesen in einer Objektdatei speichert. Assemblerbefehle haben folgenden Aufbau:

```
[Label:] Mnemonic [Operand] [, Operand] [, Operand] [;Kommentar]
```

Label

Ein Label ist eine virtuelle Adresse, deren physikalischer Wert erst beim Compilieren ermittelt wird. Labels sind Namen und müssen mit einem Doppelpunkt enden. Diese Stelle des Programms kann dann durch Angabe des Labels angesprochen werden. Labels werden z.B. bei Sprungbefehlen zur Angabe von relativen Adressen benutzt. Dort muß allerdings der Doppelpunkt weggelassen werden.

Mnemonic

Ein Mnemonic ist ein Assemblerbefehl, z.B. mov, der beim Assemblieren in einen Maschinencode übersetzt wird. Dieser Assemblerbefehl kann bis zu drei Operanden besitzen.

Operanden

Operanden sind Argumente oder Ausdrücke, die im Zusammenhang mit Assemblerdirektiven oder Befehlen verwendet werden.

Kommentare

Ein Assemblerprogramm sollte gut dokumentiert werden. Kommentare erleichtern die Bearbeitung des Quellcodes und die anschließende Fehlersuche erheblich. Sie werden mit einem Semikolon eingeleitet und können an beliebiger Stelle stehen. Alle Zeichen, die hinter diesem Semikolon stehen, werden beim Assemblieren ignoriert und erzeugen keinen Programmcode.

3.1.2 Zahlen

Zahlen können Konstanten oder Adreßangaben sein. Konstanten sind immer vom Typ Byte, während Adressen vom Typ Byte oder Word sein können. Adressen im internen RAM sind dagegen stets vom Typ Byte.

Bytes können als Dezimal-, Binär- oder Hexadezimalzahl angegeben werden. Auch die Angabe als Charakterzeichen ist erlaubt:

```
       120   =   120
       28d   =    28
       3Fh   =    63
       ´A´   =    65
  10111001b   =   185
```

Bei der Angabe von Hexadezimalzahlen, die mit einem Buchstaben beginnen, ist zu beachten, daß der Assembler eine führende Null erwartet. Einem konstanten Wert muß stets ein Nummernzeichen (´#´) vorangestellt werden, ansonsten wird er vom Assembler als Adresse interpretiert.

3.1.3 Speichertypen

Der Assembler unterstützt die nachfolgenden Speichertypen. Jede Variable kann durch den Speichertyp explizit im internen und externen Speicher plaziert werden. Zugriffe auf den internen Datenspeicher laufen wesentlich schneller ab als Zugriffe auf den externen Datenspeicher. Deshalb ist es sinnvoll, häufig benutzte Variablen im internen Datenspeicher unterzubringen. Große Datenmengen wie z.B. Tabellen müssen dagegen im externen Speicher plaziert werden.

Speichertyp	Adreßbereich	Beschreibung
DATA	0000h – 007Fh	Direkt adressierbarer interner Speicher
BIT	0020h – 002Fh	Bitadressierbarer interner Speicher
IDATA	0000h – 00FFh	Indirekt adressierbarer interner Speicher
XDATA	0000h – FFFFh	64 KB externer Datenspeicher
CODE	0000h – FFFFh	64 KB Programmspeicher
CONST	0000h – FFFFh	Wie CODE, kann nur für ROM-Konstanten verwendet werden

Tabelle 3 Speichertypen

3.1.4 Speicheradressierung

Die Speicheradressierung ist für die Bearbeitung der Versuchsaufgaben von großer Bedeutung, da neben dem internen und externen Speicher auch das LCD und der D/A-Wandler über den Adreß-/Datenbus angesprochen werden. Es gibt verschiedene Arten der Speicheradressierung, die im folgenden vorgestellt werden.

Unmittelbare Adressierung

Bei der unmittelbaren Adressierung wird die Adresse als Konstantenwert angegeben, z.B.

```
mov 80h,#10
```

Direkte Adressierung

Die direkte Adressierungsart ermöglicht den Zugriff auf die SFR-Register und die unteren 128 Bytes des internen Datenspeichers. Direkte Adressen sind stets 8bit-Werte. Für die Spezialfunktionsregister sind Symbole definiert, die die Adressierung vereinfachen. Ein Schreibzugriff auf Port1 hat z.B. folgende Gestalt

```
mov P1,#25
```

Indirekte Adressierung

Als Zeigerregister für die indirekte Adressierung werden die 8bit-Register R0, R1 sowie der 16bit breite Datenzeiger DPTR verwendet. Mit R0 und R1 kann nur auf das interne RAM oder auf 256 Byte große Blöcke des externen RAMs zugegriffen werden. Der 16bit breite DPTR ermöglicht dagegen den Zugriff auf den gesamten 64kByte-Speicherbereich.

interner RAM-Zugriff:	@R0, @R1
externer RAM-Zugriff:	@R0, @R1, @DPTR

Alle Befehle zum Ansprechen des externen Programm- oder Datenspeichers arbeiten nur über den Akkumulator A. Auf den Programmspeicher wird mit dem movc-Befehl zugegriffen, auf externe Daten mit dem movx-Befehl.

Der DPTR kann mit einer absoluten Adresse oder einem Label geladen werden, z.B.

```
mov  dptr,3F00h
movx A,@dptr            ;Inhalt von Adresse 3F00h in Akku holen
mov  dptr,#LABEL
movx @dptr,A            ;Inhalt von Akku an Adresse, die durch Label
                        ;spezifiziert ist, schreiben
```

Indirekt indizierte Adressierung

Sind Werte in Form einer Tabelle im Speicher abgelegt, bietet sich die indirekt indizierte Adressierung mit Hilfe des movc-Befehls an:

```
movc A,@A+<Basisadresse>
```

Die Basisadresse entspricht der niedrigsten Tabellenadresse und ist ein 16bit-Wert. Dies erzwingt die Verwendung des Datenzeigers DPTR. Der Inhalt des Akkumulators wird zur

Basisadresse hinzuaddiert und die Summe als Adresse interpretiert. Der Inhalt der Adresse wird in den Akku geladen. Der Akkumulator dient dabei als Index auf die Tabelle.

```
movc A,@A+dptr
```

Hinweis: Da der movx-Befehl im Gegensatz zum movc-Befehl nicht indirekt indiziert arbeitet, müssen Wertetabellen immer im Programmspeicher abgelegt werden.

Bitadressierung

Bitadressen repräsentieren die exakte Adresse eines Bits im Speicher. Auf Bits eines Bytes, das sich im bitadressierbaren Speicherbereich (20h bis 2Fh) befindet, kann über einen Punkt ´.´ zugegriffen werden.

```
setb 20h.0          ;Bit 0 der Adresse 20h setzen
clr  ACC.7          ;Bit 7 im Akkumulator löschen
```

3.1.5 Direktiven

Der Assembler kennt sämtliche Direktiven, welche die Definition von Symbolen, die Reservierung von Speicher und die Plazierung von Programmcode erlauben. Die Direktiven dürfen nicht mit Befehlen verwechselt werden. Sie erzeugen keinen Programmcode.

Die Include-Direktive

Die Include-Direktive fügt die angegebene Quelldatei beim Compilieren an dieser Stelle ein.

```
$include(Dateiname)
```

Mit dieser Direktive werden z.B. die Symboldefinitionen der SFRs des 80C517A, die sich in der Datei REG517A.INC befinden, in das Hauptprogramm eingebunden.

Die EQU-Direktive

Die Equate-Direktive ermöglicht es, Symbole zu verwenden, die numerische oder Zeichenkettenkonstanten repräsentieren. Sie erzeugt symbolische Konstanten.

```
name  equ  ausdruck
```

Symbole können mit den Direktiven EQU und SET erzeugt werden. Symbole, die mit der EQU-Direktive erzeugt wurden, können im Verlauf des Programms nicht mehr neu definiert werden. Symbole, die durch SET erzeugt wurden, können beliebig oft definiert werden.

Die Segment-Direktive

Ein Segment ist ein Block im Daten- oder Programmspeicher, den der Assembler aus dem Quellcode erstellt. Der Mikrocontroller besitzt verschiedene Speicherbereiche wie z.B. Programmspeicher, Datenspeicher, Spezialfunktionsregister und externen Datenspeicher. Segmente werden verwendet, um Programmcode, Konstanten und Variablen in diesen Speicherbereichen zu plazieren. Es gibt zwei verschiedene Arten von Segmenten: relokatible und absolute Segmente:

Relokatible Segmente besitzen einen Namen und einen Speichertyp. Relokatible Segmente, die den gleichen Namen besitzen, jedoch aus unterschiedlichen Objektdateien stammen,

werden beim Link-Prozeß zusammengefügt. Diese Segmente werden folgendermaßen erzeugt:

```
prog  segment  code
```

definiert ein Segment namens `prog` mit dem Speichertyp `code`. Das bedeutet, daß die Daten im Segment `prog` im Programmspeicher abgelegt werden. Haben Sie ein Segment definiert, müssen Sie dieses auswählen. Dies geschieht mit der RSEG-Direktive. Ein Segment wird folgendermaßen aktiviert:

```
rseg  prog
```

Diese Direktive gilt für den nachfolgenden Code, bis das Segment gewechselt wird.

Absolute Segmente befinden sich in festen Speicherbereichen und besitzen keinen Namen. Der Linker kann sie im Gegensatz zu relokatiblen Segmenten nicht verschieben und auch nicht mit anderen Segmenten kombinieren. Absolute Segmente werden mit den Direktiven `CSEG`, `DSEG` oder `XSEG` erzeugt. Mit ihnen können Programmcode und Daten an festen Speicheradressen plaziert werden.

```
cseg  at  adresse
```

Die Using-Direktive

Mit der Using-Direktive kann zwischen den vier vorhandenen Registerbänken des Mikrocontrollers gewechselt werden. Registerbank 0 ist voreingestellt und reicht für die Bearbeitung der Praktikumsaufgaben vollkommen aus.

Die Public/Extrn-Direktive

Die Public-Direktive macht ein Symbol, eine Variable oder ein Label für andere Objektmodule bekannt. Um dieses Symbol in anderen Objektmodulen zu verwenden, muß es dort mit der Extrn-Direktive eingebunden werden.

Symbole können einen Wert, einen Textblock, eine Adresse oder ein Register repräsentieren. Sie können auch für numerische Konstanten und Ausdrücke verwendet werden.

```
PUBLIC  label
EXTRN  segment(label)
```

Die DS-Direktive

Die DS-Direktive reserviert eine bestimmte Anzahl von Bytes an der aktuellen Adresse im aktuellen Segment.

```
variable:  DS  2          ;reserviert 2 Bytes für variable
```

Die DB/DW-Direktive

Die DB-Direktive und die DW-Direktive erlauben die Definition von Konstanten im Programmspeicher. Zur Initialisierung von Datenbytes wird die DB-Direktive verwendet. Mit ihr werden üblicherweise Zeichenkettenvariablen initialisiert. Der Initialisierungswert wird als String-Konstante oder als Folge von Bytes angegeben. Das folgende Beispiel zeigt die verschiedenen Möglichkeiten:

```
konstante1:  DB  'a','b','c'     ;als Zeichen
konstante2:  DB   97,98,99       ;als ASCII-Werte
konstante3:  DB  'abc'           ;als String
```

Die DW-Direktive initialisiert Codespeicher mit 16bit Words.

Die DBIT-Direktive

Die DBIT-Direktive reserviert Speicher in einem Bit-Segment.

```
flag:  DBIT  1            ;reserviert Speicher für ein Bit
```

Die ORG-Direktive

Die ORG-Direktive (Origin) wird verwendet, um Programmcode ab einer bestimmten Adresse im Speicher starten zu lassen.

```
ORG  adresse
```

Trifft der Assembler auf eine `ORG`-Anweisung, übernimmt er den Wert und ordnet dem folgenden Befehl die genannte Adresse zu.

Die END-Direktive

Die END-Direktive kennzeichnet das Ende des Quelltextes. Alle nachfolgenden Befehle werden ignoriert.

```
END
```

3.2 Der Stack

Der Stapelspeicher (Stack) ist ein in seiner Größe limitierter Speicherbereich, der 8bit breit organisiert ist. Auf ihn kann nur mit dem Stapelzeiger (SP), einem ebenfalls 8bit breiten Register, zugegriffen werden. Er wird hauptsächlich beim Aufruf von Unterprogrammen und Interruptroutinen benötigt, kann aber auch zur temporären Speicherung von Registern verwendet werden.

Der Stack arbeitet nach dem LIFO-Prinzip (**L**ast **I**n **F**irst **O**ut), d.h. es kann immer nur auf das zuletzt auf den Stack gelegte Register zugegriffen werden. Für Stackoperationen stehen die beiden Befehle PUSH und POP zur Verfügung.

```
PUSH  ACC
POP   ACC
```

Der Befehl PUSH inkrementiert den Stapelzeiger und rettet den Akkumulator auf den Stack. Der POP-Befehl holt den gespeicherten Wert wieder vom Stack und dekrementiert anschließend den Stapelzeiger, so daß dieser stets auf den obersten Wert des Stapelspeichers zeigt. Die Befehle PUSH und POP verlangen als Operatoren die absoluten Adressen der Register. Diese lauten

`ACC`	für den Akkumulator
`AR0`	für das Register R0
`ARx`	für die Register Rx

Bei einem Interrupt wird beispielsweise vor der Ausführung der Interruptroutine die Rücksprungadresse auf den Stack gesichert, an der das Programm nach Abarbeitung der Routine wieder fortgeführt wird. Zusätzlich müssen alle Register, die in dieser Routine verändert werden, beim Eintritt in die ISR auf den Stack gesichert werden, da der Interrupt das Hauptprogramm jederzeit unterbrechen kann. Vor dem Rücksprung in das Hauptprogramm müssen die Register wieder restauriert werden.

Für den Stapelspeicher muß ein ausreichend großer Speicherbereich im internen RAM reserviert werden. Dies geschieht folgendermaßen:

```
STACK   SEGMENT  IDATA          ;Stacksegment definieren
        RSEG     STACK
        DS       20h            ;32 Byte für Stapelspeicher reservieren
```

Der Stapelzeiger wird zu Beginn des Hauptprogramms mit folgender Befehlszeile initialisiert, so daß der erste Push-Befehl den Wert an den Anfang des reservierten Speicherbereiches legt:

```
start:
   mov   SP,#STACK-1
```

3.3 Programmstrukturen

Programmverzweigungen

Programmverzweigungen werden mit den Sprungbefehlen des Mikrocontrollers realisiert. Er unterscheidet dabei zwischen unbedingten und bedingten Sprüngen. Mit den unbedingten Sprüngen ACALL und AJMP kann nur innerhalb eines 2kByte-Adreßbereiches verzweigt werden, während die Sprungbefehle LCALL und LJMP den gesamten Speicher adressieren können. Zur Steuerung des Programmablaufs besitzt der Mikrocontroller die bedingten Sprungbefehle. Die folgenden Beispiele zeigen die Umsetzung von typischen Programmstrukturen in Assembler-Code.

Eine If-Anweisung läßt sich in Assembler folgendermaßen realisieren:

IF-Anweisung	analoge Assembler-Routine

`if (Akku > 0) then`
`  zaehler = zaehler+1`
	`  jz  weiter`
	`  inc  zaehler`
	`weiter:`

Beispiel 1 IF-Anweisung

Ist die Bedingung (Akku > 0) nicht erfüllt, wird ein Sprung zu der durch das Label *weiter* angegebenen Adresse durchgeführt, ansonsten wird mit der Abarbeitung des nächsten Befehls fortgefahren.

IF (AND-Verknüpfung)	analoge Assembler-Routine

`if (Akku > 0) and (R0 = 1)`
`  then`
`  zaehler = zaehler+1`
	`  jz  weiter`
	`  cjne  R0,#1,weiter`
	`  inc  zaehler`
	`weiter:`

Beispiel 2 AND-Verknüpfung

Bei der logischen AND-Verknüpfung wird zum Label *weiter* verzweigt, wenn mindestens eine der beiden Bedingungen (Akku > 0) oder (R0 = 1) nicht erfüllt ist. Andernfalls wird die Variable *zaehler* um eins erhöht.

IF (OR-Verknüpfung)	analoge Assembler-Routine

`if (Akku > 0) or (R0 = 1)`
`  then`
`  zaehler = zaehler+1`
	`  jz  bed2`
	`  jmp  incr`
	`bed2:`
	`  cjne  R0,#1,weiter`
	`incr:`
	`  inc  zaehler`
	`weiter:`

Beispiel 3 OR-Verknüpfung

Die logische OR-Verknüpfung wird implementiert, indem zuerst die Bedingung (Akku > 0) geprüft wird. Ist diese erfüllt, erfolgt sofort der Sprung zum Label *incr* und die Variable *zaehler* wird erhöht. Ist sie nicht erfüllt, wird die zweite Bedingung (R0 = 1) geprüft und bei Übereinstimmung zum Label *incr* verzweigt. Eine If-Else-Anweisung besitzt in Assembler folgende Struktur:

IF-ELSE-Anweisung	analoge Assembler-Routine

`if (Akku > 0) then`
`  zaehler = zaehler+1`
`else zaehler = zaehler-1`
	`  jz  decr`
	`  inc  zaehler`
	`  jmp  weiter`
	`decr:`
	`  dec  zaehler`
	`weiter:`

Beispiel 4 IF-ELSE-Anweisung

Eine For-Schleife wird mit dem Assemblerbefehl *djnz* gelegt. Dieser Befehl dekrementiert das angegebene Register und prüft es anschließend auf Null. Ist es ungleich Null, wird das Label *schleife* angesprungen.

FOR-Schleife	analoge Assembler-Routine

`for i = N downto 1 do`
`begin`
`  anweisung1`
`  anweisung2`
`  ...`
`end`
	`  mov  R0,#N`
	`schleife:`
	`  ...`
	`  ...`
	`  djnz  R0,schleife`

Beispiel 5 FOR-Schleife

Das letzte Beispiel zeigt eine While-Struktur. Die While-Schleife wird erst dann verlassen, wenn die Bedingung (Akku > 0) nicht mehr erfüllt ist.

WHILE-Schleife	Analoge Assembler-Routine

`while (Akku > 0) do`
`  Akku = Akku-1`
	`loop:`
	`  jz  weiter`
	`  dec  A`
	`  jmp  loop`
	`weiter:`

Beispiel 6 WHILE-Schleife

Interrupt-Zählvariablen

Um Zeiten messen zu können, die größer sind als die Umlaufzeit eines Referenzzählers, ist es notwendig, die Anzahl der Timerüberläufe softwaremäßig mitzuzählen.

$T_{ges} = t_2 - t_1$ sei die Zeit, nach welcher die ISR abgearbeitet werden soll. Um die Anzahl der Timerüberläufe mitzuzählen, wird eine Variable angelegt, die bei jedem Interruptaufruf heruntergezählt wird. Ist diese Variable gleich 0, dann wird die ISR abgearbeitet, ansonsten wird zum Ende der Routine gesprungen. Die Zeitdauer ist dann durch folgende Formel bestimmt:

$$T_{ges} = (Z_2 - Z_1 + \ddot{U} * 2^Z) * \frac{1}{f_{Timer}}$$

$\ddot{U}$ = Anzahl der ermittelten Überläufe
Z = Zählerbreite des Referenzzählers in Bit
Z_1 = Zählregisterinhalt zum Zeitpunkt t_1
Z_2 = Zählregisterinhalt zum Zeitpunkt t_2

3.4 Ausmaskierung von Bits

In Spezialfunktionsregistern werden häufig nur bestimmte Bits verändert, während die restlichen Bits nicht beeinflußt werden dürfen. Bits, die sich in bitadressierbaren Spezialfunktionsregistern befinden, können mit den Assemblerbefehlen `setb` und `clr` gesetzt bzw. gelöscht werden. Bei SFRs, die nicht bitadressierbar sind, müssen die Bits durch die logischen AND- und OR-Verknüpfungen ausmaskiert werden.

Bits in einem Byte werden gelöscht, indem das Byte mit einer Maske AND-verknüpft wird, in der an den Bitpositionen der zu löschenden Bits Nullen stehen. Die Bitpositionen, die sich nicht verändern sollen, enthalten eine eins.

Im folgenden Beispiel werden die obersten zwei Bits gelöscht.

```
          11111001    Byte
&         00111111    Maske
          ________
          00111001
```

Bits in einem Byte werden gesetzt, indem das Byte mit einer Maske OR-verknüpft wird, in der die Bitpositionen der zu setzenden Bits eine eins enthalten. Im nächsten Beispiel werden die unteren drei Bits gesetzt.

```
          10110010    Byte
v         00000111    Maske
          ________
          10110111
```

3.5 Softwareerstellung

Die Erstellung des Programmcodes gliedert sich in drei Schritte:

1. Der Assembler

Im ersten Schritt wird der Programmcode in einer Quelldatei mit der Endung (.a51) editiert. Der Assembler A51 übersetzt diesen Quellcode in einen Objektcode. Er erzeugt eine Objektdatei (.obj) und eine Listingdatei (.lst), in der die Ergebnisse der Assemblierung protokolliert werden. Die Objektdatei ist relokatibel, d.h. es werden noch keine absoluten Adressen für den Programmcode erzeugt.

2. Der Linker

Der Linker/Locater L51 bindet mehrere Programm-Module, die als Objektdateien vorliegen, zu einem ablauffähigen Programm zusammen. Dabei werden Extrn/Public-Bezüge aufgelöst

und die relokatiblen Programmteile auf absolute Adressen plaziert. Die zu kombinierenden Module können sowohl in Assembler als auch in der Programmiersprache C geschrieben sein. Der Linker ermöglicht eine modulare Programmierung, d.h. die Zerlegung des Quellcodes in mehrere Module. Dadurch bleibt auch ein sehr großes Programm übersichtlich.

Der Linker erzeugt eine absolute Objektdatei (.abs) und eine Map-Datei (.m51), die die Ergebnisse des Link-Prozesses enthält.

3. Der Objekt-Hex-Konverter

Der Programmcode, der sich in der Objektdatei befindet, wird in Hexadezimalwerte konvertiert und im Intel-HEX-Dateiformat ausgegeben. Die HEX-Datei kann von EPROM-Programmiergeräten oder von den im Praktikum verwendeten Flashtools in den Speicher des Mikrocontrollers geschrieben werden. Abbildung 7 zeigt den Ablauf der Quellcode-Erstellung.

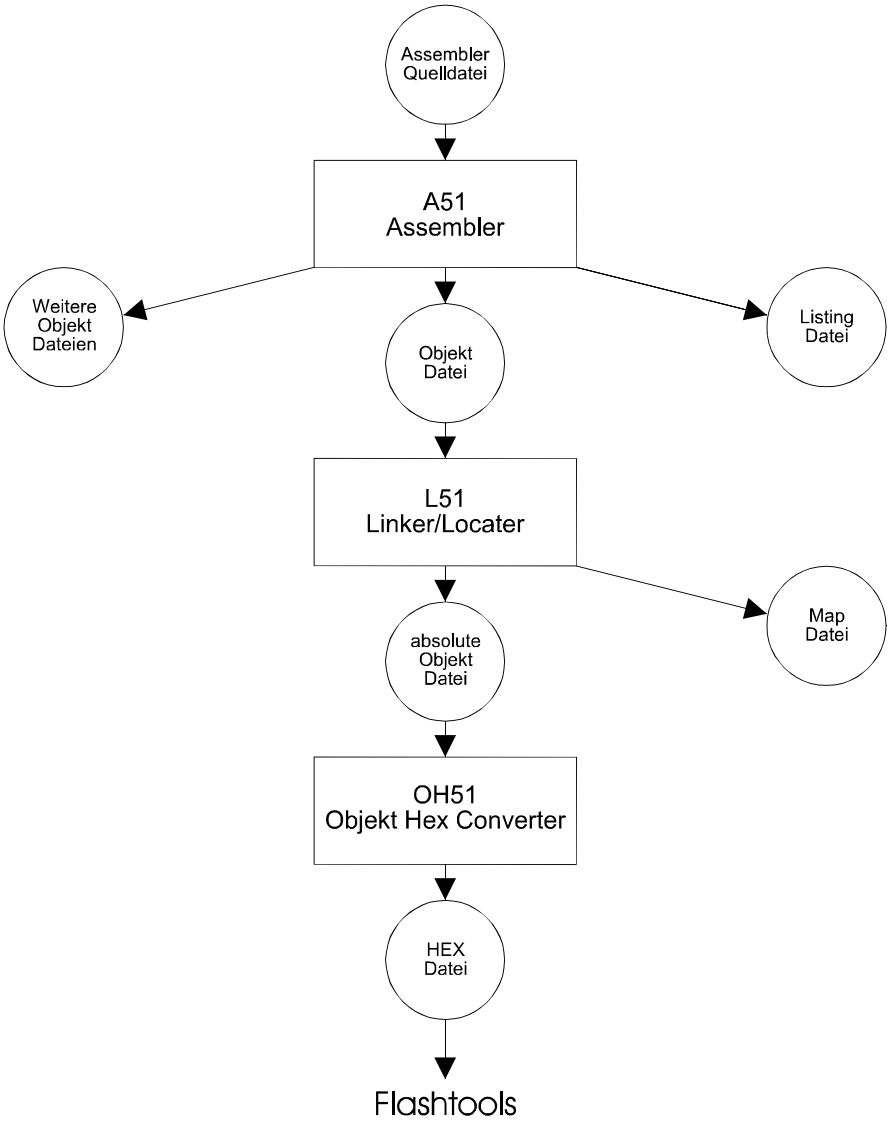

Abbildung 7 Programmerstellung für den 80C517A

3.6 Das Entwicklungstool Proview

Zur Erstellung der Assemblerprogramme für die Versuchsaufgaben wird das Softwaretool Proview verwendet. Es unterstützt das Compilieren, Assemblieren und Linken von Quellcode-Dateien in einem Projekt. Ein Projekt gliedert sich in mehrere Dateien, die beim Linken zu einer Objektdatei zusammengefügt werden. Ein in Proview integrierter Simulator ermöglicht es, die erstellten Programme zu testen und erleichtert die Fehlersuche erheblich.

Der Umgang mit Proview soll anhand eines Beispiels demonstriert werden:

- Öffnen Sie mit „Project|Open“ das Projekt *beispiel.prj* im Verzeichnis *c:\beispiel*. Es erscheint das Projektfenster. Klicken Sie mit der rechten Maustaste auf das Projektfenster und wählen Sie „Add file“. Fügen Sie dem Projekt auf diese Weise die Module *cpuinit.c*, *delay.a51*, *lcd.c* und *led.c* sowie das Hauptprogramm *main.c* hinzu.
- Compilieren und Linken Sie das Projekt mit „Project|Build all“. Proview erstellt die Zwischencode-Dateien (.obj) und die Listingdateien (.lst) nach Abbildung 7. Die benötigte Datei im Intel-Hex-Format wird automatisch miterzeugt. Ein Message-Fenster gibt Auskunft über den Compilier- und Linkprozeß.
- Der Programmcode, der nun als Intel-Hex-Datei vorliegt, kann mit Hilfe der Flashtools der Firma Phytec über die RS-232-Schnittstelle des PC in den Flashspeicher des Mikrocontrollers geladen werden. Der Flashspeicher besteht aus 2 Bänken à 64 KB, die wiederum in 4 Sektoren unterteilt sind. Die erste Bank (Bank 0) ist schreibgeschützt, da sich in ihr die Firmware befindet, die für die Kommunikation mit dem PC zuständig ist. Die zweite Bank ist unbelegt, in sie wird der Programmcode geladen. Der Mikrocontroller bootet nach einem Hardware-Reset stets aus dieser Bank. Gehen Sie ins Menü „Tool|Run Tool|Flash“ und starten Sie die Flashtools.
- Quittieren Sie die folgende Meldung mit „o.k.“. Bevor eine Kommunikation mit dem Mikrocontroller aufgebaut werden kann, müssen Sie diesen zuerst in den Programmiermodus bringen. Dies geschieht, indem Sie die Programmiertaste an der Frontplatte des Gehäuses gedrückt halten und anschließend den Reset-Knopf für ca. 2s betätigen. Der Programmiermodus wird durch vier leuchtende LEDs angezeigt.
- Jetzt können Sie den Download des Programms starten. Wenn die Übertragung beendet ist, können Sie die Flashtools verlassen. Das Programm befindet sich nun im Speicher des Mikrocontrollerboards und kann durch Betätigung der Reset-Taste (ca. 2s) gestartet werden. Es bleibt auch nach Abschalten der Spannungsversorgung erhalten.

Den integrierten Debugger von Proview starten Sie durch den Aufruf von „Debug|Start“. Wählen Sie „Virtual Machine (Simulator)“ und geben Sie unter Mikrocontroller „80C517“ und unter Frequency „18 MHz“ an. Nun können Sie Ihr Programm starten, indem Sie auf das Symbol „GO“ klicken. Der Debugger beginnt mit der Abarbeitung des Programms an der Adresse, die im Register PC steht. Mit „Step into“ und „Step over“ kann das Programm Zeile für Zeile abgearbeitet werden oder es können Breakpoints gesetzt werden. Dabei können Sie sich über das Menü „View“ sämtliche Registerwerte, den Inhalt des Daten- und Programmspeichers und den Status der einzelnen Peripherieeinheiten anzeigen lassen. Sie

haben die Möglichkeit, den Inhalt der CPU-Register und des internen Speichers zu editieren. Die Funktion „Add Watch“ im Menü „Debug“ ermöglicht die Überwachung von Programmvariablen. Mit „Reset“ im gleichen Menü läßt sich ein Hardware-Reset simulieren. Danach kann das Programm erneut gestartet werden.

Den Debugger verlassen Sie mit „Debug|Terminate“.

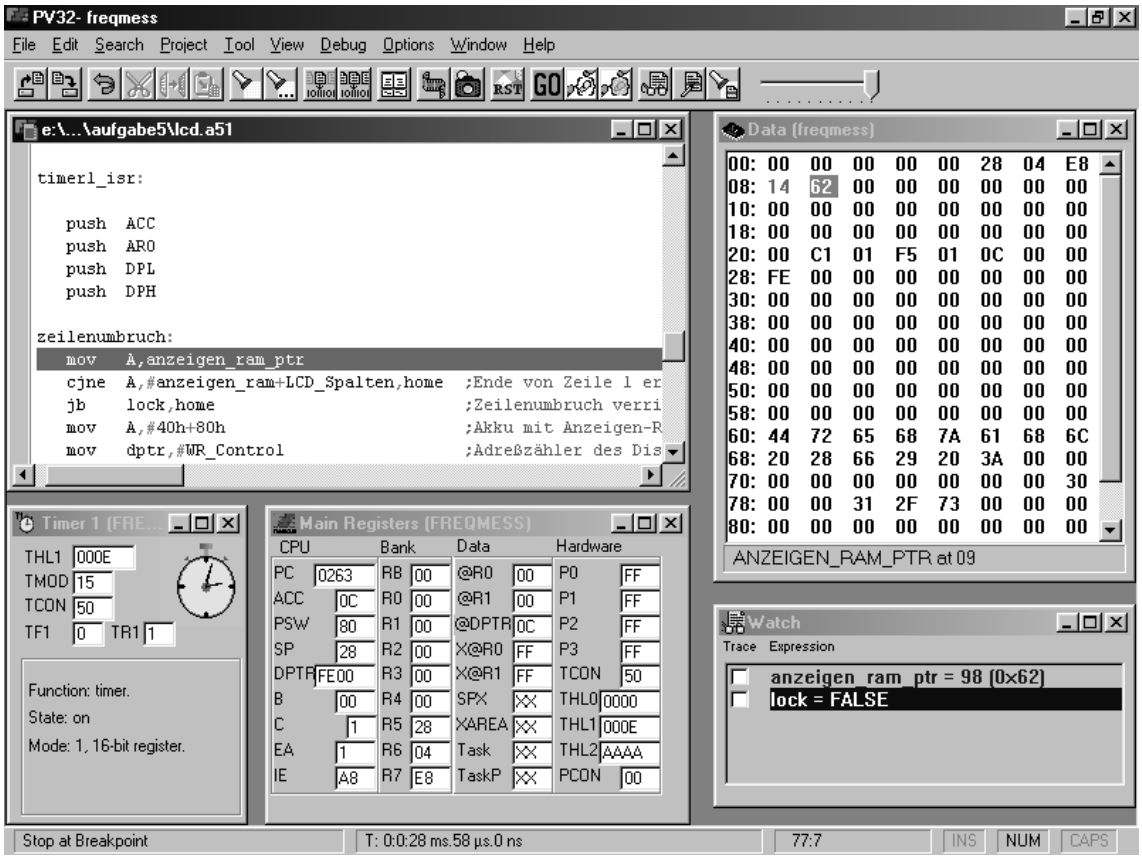

Abbildung 8 Der Simulator von Proview

4 Anhang

Vektoradresse	Interruptquelle	Request-Flag
00ABh	Compare match COMCLR	ICR
00A3h	Compare match COMSET	ICS
009Bh	Compare-Timer Overflow	CTF
0093h	Compare match CM0-CM7	ICMP0 - ICMP7
0083h	serieller Port Interrupt 1	RI1/TI1
006Bh	externer Interrupt 6	IEX6
0063h	externer Interrupt 5	IEX5
005Bh	externer Interrupt 4	IEX4
0053h	externer Interrupt 3	IEX3
004Bh	externer Interrupt 2	IEX2
0043h	A/D-Wandler-Interrupt	IADC
002Bh	Überlauf Timer 2	TF2+EXF2
0023h	Serieller Port Interrupt 0	RIO+TIO
001Bh	Timer 1 Interrupt	TF1
0013h	externer Interrupt 1	IE1
000Bh	Timer 0 Interrupt	TF0
0003h	externer Interrupt 0	IE0
0000h	Reset	

Abbildung 9 Die Interrupt-Vektoradressen des SAB80C517A

Instructions	Code										Description	Execution time
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears all display and returns the cursor to the home position (Address 0).	82 µs ~ 1.64 ms
Return home	0	0	0	0	0	0	0	0	1	-	Returns the cursor to the home position (Address 0). Also returns the display being shifted to the original position. DD RAM contents remain unchanged.	40 µs ~ 1.6 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and specifies or not to shift the display. These operations are performed during data write and read.	40 µs
Display ON/OFF control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF of all display (D), cursor ON/OFF (C), and blink of cursor position character (B).	40 µs
Cursor and display shift	0	0	0	0	0	1	S/C	R/L	-	-	Moves the cursor and shifts the display without changing DD RAM contents.	40 µs
Function set	0	0	0	0	1	DL	N	F	-	-	Sets interface data length (DL) number of display lines (L) and character font (F).	40 µs
Set CG RAM address	0	0	0	1	A_{CG}						Sets the CG RAM address. CG RAM data is sent and received after this setting.	40 µs
Set DD RAM adress	0	0	1	A_{DD}							Sets the DD RAM address. DD RAM data is sent and received after this setting.	40 µs
Read busy flag & address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	40 µs
Write data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	40 µs
Read data to CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	40 µs
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift. S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift right R/L = 0: Shift left DL = 1: 8 bits DL = 0: 4 bits N = 1: 2 lines N = 0: 1 line F = 1: 5x10 dots F = 0: 5x7 dots BF = 1: Internally operating BF = 0: can accept instruction										DD RAM: Display data RAM CG RAM: Character generator RAM A_{CG}: CG RAM address A_{DD}: DD RAM address corresponds to cursor address. AC: Address counter used for both DD and CG RAM address.	

Abbildung 10 Befehle des Grafikcontrollers

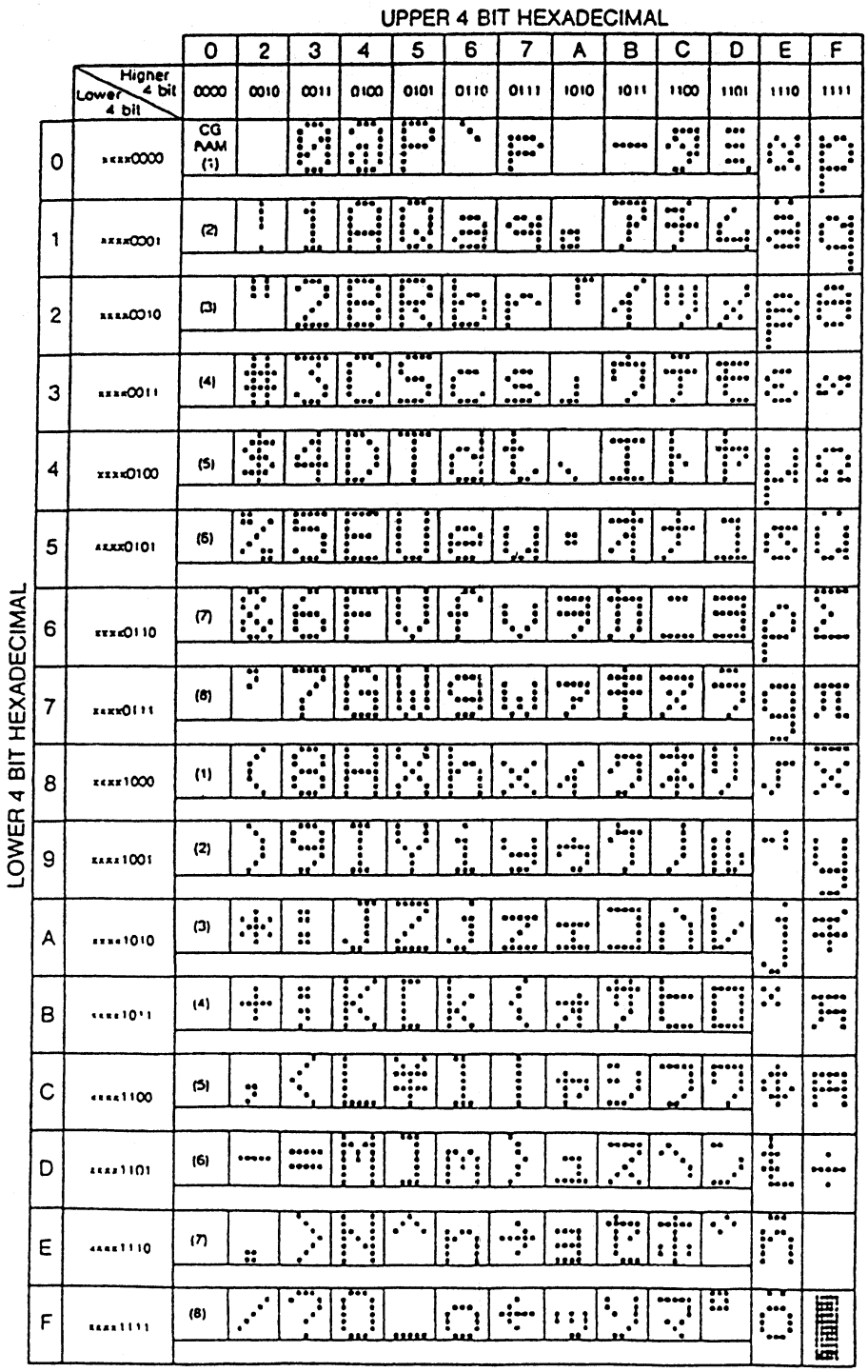

Abbildung 11 Zeichensatz des LCD

```
;*******************************************************************************
;*   Programmrahmen für ein Hauptprogramm, das auf dem Mikrocontroller 80C517A   *
;*   laufen soll. Die Datei ´REG517A.INC´, die die Registernamen des 80C517A     *
;*   enthält, muß bei der Assemblierung im aktuellen Verzeichnis stehen.         *
;*   Kommentare beginnen mit einem Semikolon. Alles, was danach folgt, wird      *
;*   beim Assemblieren ignoriert. Labels, die das Hauptprogramm aus anderen      *
;*   Modulen verwendet, werden mit der Direktive EXTRN eingebunden. Die Module   *
;*   müssen zuvor dem Projekt hinzugefügt werden. Text, der kursiv dargestellt   *
;*   ist, muß entsprechend ergänzt werden.                                       *
;*******************************************************************************
$NOMOD51                          ;Registerbelegung des 8051 abschalten
$include(REG517A.INC)             ;SFR Symbole von 80517A benutzen

NAME  <Hier muß der Name des Programms stehen>

;*******************************************************************************
;*                               Extern-Definitionen                           *
;*******************************************************************************
EXTRN   CODE(delay)               ;Modul delay.a51
<Hier können weitere Labels stehen, auf die das Hauptprogramm zugreift und die
 sich in einem anderen Modul befinden>

;*******************************************************************************
;*                               Segment-Definitionen                          *
;*******************************************************************************
PROG    SEGMENT  CODE             ;Codesegment definieren
STACK   SEGMENT  IDATA            ;Stacksegment definieren
VAR     SEGMENT  DATA             ;Datensegment definieren
BITS    SEGMENT  BIT              ;Bitsegment definieren
using 0                           ;Registerbank 0 auswählen

;*******************************************************************************
;*                             Konstanten-Deklaration                          *
;*******************************************************************************
<Hier werden die benutzten Konstanten mittels der EQU-Direktive definiert>

;*******************************************************************************
;*                             Stack-Segment                                   *
;*******************************************************************************
        RSEG   STACK
        DS     20h                ;32 Byte für Stapelspeicher reservieren

;*******************************************************************************
;*                             Variablen-Deklaration                           *
;*******************************************************************************
        RSEG   VAR
<Hier kann Platz für die notwendigen Variablen reserviert werden>

;*******************************************************************************
;*                              absolute Segmente                              *
;*******************************************************************************
CSEG  AT  0000h                       ;Startadresse nach einem Hardware-Reset
   ljmp  start
CSEG  AT  <Interruptvektoradresse>    ;Einfügen einer Interruptserviceroutine
   <Hier steht der Sprungbefehl zur eigentlichen ISR>

;*******************************************************************************
;*                                Code-Segment                                 *
;*******************************************************************************
        RSEG   PROG
        ORG    0100h
start:
   mov   SP,#STACK-1                  ;Stack initialisieren
<Hier beginnt das Hauptprogramm>

END
```

Listing 1 Programmrahmen für ein Hauptprogramm

```
;*******************************************************************************
;*   Programmrahmen für ein Unterprogramm, das auf dem Mikrocontroller 80C517A   *
;*   laufen soll. Die Datei ´REG517A.INC´, die die Registernamen des 80C517A     *
;*   enthält, muß bei der Assemblierung im aktuellen Verzeichnis stehen.         *
;*   Kommentare beginnen mit einem Semikolon. Alles, was danach folgt, wird      *
;*   beim Assemblieren ignoriert. Labels, die das Unterprogramm für das Haupt-   *
;*   programm bereitstellt, müssen mit der Direktive PUBLIC bekannt gemacht      *
;*   werden. Text, der kursiv dargestellt ist, muß entsprechend ergänzt werden.  *
;*******************************************************************************
$NOMOD51                          ;Registerbelegung des 8051 abschalten
$include(REG517A.INC)             ;SFR Symbole von 80517A benutzen

NAME  <Hier muß der Name des Unterprogramms stehen>

;*******************************************************************************
;*                              Public-Definitionen                            *
;*******************************************************************************
PUBLIC  <Hier müssen die Labels eingetragen werden, die für andere Module
         bekannt gemacht werden sollen>

;*******************************************************************************
;*                              Segment-Definitionen                           *
;*******************************************************************************
PROG    SEGMENT  CODE             ;Codesegment definieren
VAR     SEGMENT  DATA             ;Datensegment definieren
BITS    SEGMENT  BIT              ;Bitsegment definieren
using 0                           ;Registerbank 0 auswählen

;*******************************************************************************
;*                             Konstanten-Deklaration                          *
;*******************************************************************************
<Hier werden die benutzten Konstanten mittels der EQU-Direktive definiert>

;*******************************************************************************
;*                             Variablen-Deklaration                           *
;*******************************************************************************
<Hier kann Platz für die notwendigen Variablen reserviert werden>

;*******************************************************************************
;*                              Code-Segment                                   *
;*******************************************************************************
        RSEG   PROG

<Hier beginnt der Programmcode der Unterroutine>

END
```

Listing 2 Programmrahmen für ein Unterprogramm