Praktikum Rechnerarchitekturen

Versuche zur x86-Assemblerprogrammierung

A1 Lauflicht

A2 Timerbaustein

A3 Matrixtastatur

TU Ilmenau, Fakultät IA

Fachgebiet Rechnerarchitektur

Version von März 2009

Bitte benutzen Sie nur aktuelles Material!

Die neueste Version dieser Anleitung finden Sie stets auf

http://www.tu-ilmenau.de/ra

Inhaltsverzeichnis

1. Einleitung

1.1 Wichtige Hinweise

➔ Für die erfolgreiche Durchführung des Praktikums ist eine **Vorbereitung** notwendig. Diese Anleitung informiert Sie über die Einzelheiten. Arbeiten Sie deshalb rechtzeitig vor dem Praktikumstermin diese Anleitung durch!

➔ Verschaffen Sie sich einen Überblick über das benötigte Wissen und rekapitulieren Sie die entsprechenden Abschnitte der Vorlesung und der Übung. In vielen Fällen werden Sie zusätzliches Wissen benötigen, um zum Beispiel die Funktion nicht behandelter Befehle, Adressierungsarten oder Peripheriebausteine zu erschließen.

➔ Entwerfen Sie die Programme für die Praktikumsaufgaben auf Papier. Eine Vorbereitung am Rechner ist nicht erforderlich. Zum Praktikumstermin haben Sie Zeit für Test und Fehlersuche.

➔ Die Versuche **A2** und **A3** sind in „Grundaufgaben“ und „Fortgeschrittene Aufgaben“ gegliedert. Die „Fortgeschrittenen Aufgaben“ sind für manche Studiengänge obligatorisch, für andere optional. Beachten Sie dazu die Informationen in der aktuellen Praktikumsankündigung!

➔ Bitte arbeiten Sie ehrlich und fertigen Sie die Vorbereitungen selbst an. **Die Verwendung fremder Vorbereitungen ist eine Täuschung und kann zum Verlust des Testats führen.** Erfahrungsgemäß führen fremde Vorbereitungen häufig zu auffälligen Schwierigkeiten bei der Praktikumsdurchführung.

➔ Zur Verhinderung von Täuschungsversuchen ist es nicht erlaubt, vorbereitete Lösungen mittels Datenträgern oder Kommunikationsmedien einzuspielen. Sie müssen Ihre Programme also beim Praktikum selbst eintippen.

➔ Neben der eigenen Vorbereitung benötigen Sie zum Praktikumstermin mindestens diese Anleitung sowie die **„Arbeitsblätter zur Übung“** aus der Lehrveranstaltung. Auf weiteres zweckmäßiges Material wird an den entsprechenden Stellen verwiesen.

➔ Vor der Testatvergabe bzw. Benotung überprüft der Praktikumsbetreuer die Funktion Ihrer Programme und inspiziert gegebenenfalls Ihre notierten Ergebnisse. Er kann auch **Zusatzaufgaben** stellen oder **Hintergrundwissen** abfragen und die Erteilung des Testats bzw. der Note davon abhängig machen.

➔ In bestimmten Fällen wird während des Praktikums eine **Leistungskontrolle** durchgeführt. Beachten Sie dazu die Informationen in der aktuellen Praktikumsankündigung!

➔ Bitte beachten Sie auch die zusätzlichen Hinweise, die eventuell in der aktuellen Praktikumsankündigung enthalten sind. Internet-Adresse: **http://www.tu-ilmenau.de/ra**

1.2 Aufbau dieser Anleitung

Das Kapitel **„Einleitung“** gibt Ihnen wichtige Hinweise und informiert Sie über das Ziel des Praktikums.

Das Kapitel **„Versuchsaufbau und Arbeitsablauf“** stellt Ihnen die verwendete Hard- und Software vor und erklärt den grundsätzlichen Arbeitsablauf bei der Praktikumsdurchführung. Diese Informationen brauchen Sie für das Verständnis der Aufgabenstellungen.

Das Kapitel **„Praktikumsaufgaben“** enthält die Aufgabenstellungen für die einzelnen Versuche. Hier müssen Sie jeweils den Abschnitt durcharbeiten, der zum anstehenden Praktikumsversuch gehört. Die „Fortgeschrittenen Aufgaben“ sind für manche Studiengänge optional (aktuelle Ankündigungen beachten), die „Zusatzaufgaben“ in jedem Falle.

Das Kapitel **„Bedienungsanleitung“** erklärt Ihnen detailliert die Funktion und Bedienung der Praktikumshardware und der Software-Werkzeuge. Auch dieses Kapitel müssen Sie vor dem Praktikumstermin durchlesen. Dabei können Sie die Abschnitte übergehen, die mit der aktuellen Praktikumsaufgabe nichts zu tun haben. Weiterführende Informationen für besonders Interessierte sind als solche gekennzeichnet.

Der **„Anhang“** enthält einige Informationen nochmals in gedrängter Form und ist zum Nachschlagen bei Vorbereitung und Durchführung des Praktikums gedacht.

Bestimmte Informationen werden typografisch hervorgehoben:

- Eigennamen, Zahlen, Programmsymbole usw.: Eigenname
- Menüfunktionen: ‘*Menüname – Menüpunkt*’
- Funktionen von Buttons und Icons: ‘Funktionsname’

1.3 Internet-Adresse, Downloads, Zusatzinformationen

Diese Anleitung und weiteres Material, Verweise auf zusätzliche Informationsquellen sowie die aktuellen Praktikumsankündigungen finden Sie hier:

http://www.tu-ilmenau.de/ra

Neben dieser Anleitung benötigen Sie auf jeden Fall auch die „Arbeitsblätter zur Übung“ aus der Lehrveranstaltung.

1.4 Ziel des Praktikums

Das Praktikum Rechnerarchitekturen dient dazu, die Funktion eines Prozessors und einiger typischer Rechnerbaugruppen am praktischen Beispiel kennen zu lernen. Das geschieht dadurch, dass Sie einzelne Programme in Assemblersprache entwerfen, in Betrieb nehmen und untersuchen. Das Editieren, Übersetzen und Debuggen erfolgt mit professionellen Werkzeugen der Softwareentwicklung.

Passend zum Übungsstoff wird das **32-bit-x86-Programmiermodell** mit „flachem" Speicher benutzt. Die Aufgabenstellungen sind so ausgelegt, dass in der Regel nur die in den „Arbeitsblättern zur Übung" vorhandene Auswahl an Befehlen und Operandenkombinationen benötigt wird. Ausnahmen sind in den Aufgabenstellungen erklärt. Die Quelltextdateien sind bereits so weit vorbereitet, dass eine Konzentration auf das Wesentliche erfolgt.

Als Praktikumshardware werden Trainingssysteme mit einem x86-kompatiblen Prozessor und umfangreicher Peripherie benutzt. Diese sind mit einem PC verbunden, auf dem die Werkzeuge der Softwareentwicklung laufen. Die Trainingssysteme, welche als Laufzeitumgebung für die selbst erstellte Software dienen, sind absichtlich nicht vollständig PC-kompatibel und besitzen kein ausgeprägtes Betriebssystem. Dadurch ist ein direkter Zugriff auf die elementaren Operationen und Hardwarefunktionen möglich.

Diese Situation bei der Programmentwicklung ähnelt derjenigen, die man in der Industrie unter der Bezeichnung „Remote Debugging" beim Entwurf so genannter „Eingebetteter Systeme" vorfindet. Es existieren auch Berührungspunkte zu angrenzenden Wissensgebieten wie z.B. Rechnerorganisation und Rechnertechnik.

Sie müssen sich bei der Praktikumsdurchführung nicht auf die gegebenen Aufgabenstellungen beschränken. Im Rahmen des Zeitlimits können Sie die Aufgabenstellungen variieren und eigene Ideen ausprobieren. Der Praktikumsbetreuer kann Ihnen weitere Hinweise geben.

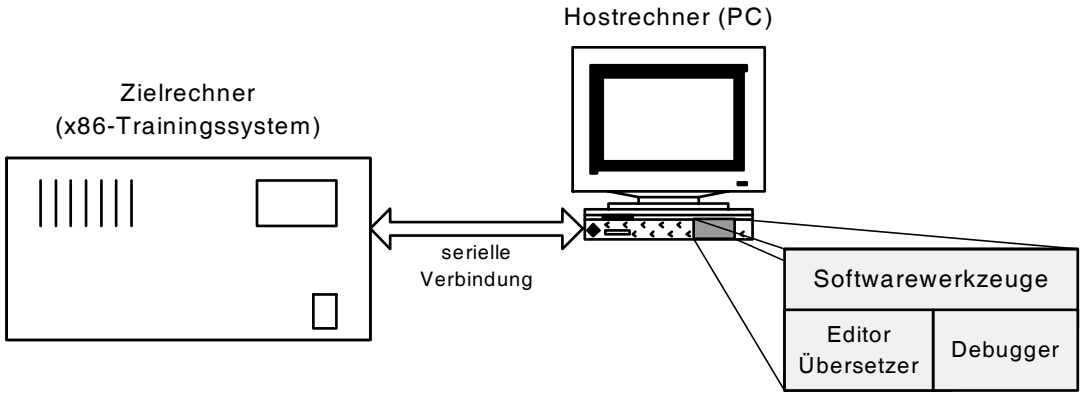
Je nach Studiengang und Praktikumsmodus können die Zeitvorgaben, das Bewertungsschema und der organisatorische Ablauf völlig verschieden sein. Bitte beachten Sie die aktuellen Ankündigungen für Ihren Studiengang.

2. Versuchsaufbau und Arbeitsablauf

2.1 Überblick

Der Versuchsaufbau besteht aus einem gewöhnlichen PC („Hostrechner“) und einem x86-Trainingssystem („Zielrechner“). Beide stehen über eine asynchrone serielle Verbindung (RS232-Standard, in der PC-Welt als „COM-Schnittstelle“ bekannt) in Verbindung.

Die Werkzeuge zur Programmentwicklung befinden sich auf dem Hostrechner. Sie umfassen im Wesentlichen eine Software-Entwicklungsumgebung (mit Quelltext-Editor und Übersetzungsfunktionen) und einen „Debugger“ (Werkzeug für Programmtest und Fehlersuche). Programmeingabe und Übersetzung finden vollständig am Hostrechner statt. Der Debugger besitzt Funktionen, um das übersetzte Programm zum Zielrechner zu übertragen und dort zu starten.



Zur Steuerung der Übertragung und zur Umsetzung der Debuggerfunktionen befindet sich im Zielrechner eine kleine Softwarekomponente, welche als „Monitor“ bezeichnet wird. Diese wird für das getestete Programm in der Regel nicht direkt sichtbar. Letzteres kann uneingeschränkt auf die Ressourcen des Zielrechners zugreifen.

Im Versuchsablauf geben Sie Ihr Assemblerprogramm zunächst mit dem Editor ein. Dabei benutzen Sie einen vorgegebenen Rahmen-Quelltext. Anschließend führen Sie die Übersetzung mit den eingebauten Übersetzungswerkzeugen aus. Dabei können Fehlermeldungen auftreten, welche auf Syntaxfehlern des eingegebenen Quelltextes beruhen. Erst wenn Sie diese Fehler beseitigt haben, kann ein ablauffähiges Maschinencode-Programm erzeugt werden.

Die Untersuchung des erzeugten Programms erfolgt nun mit dem Debugger. Dieser überträgt den Maschinencode in den RAM-Speicher des Zielrechners. Anschließend können vom Hostrechner aus verschiedene Debug-Funktionen ausgelöst werden, welche sich auf das in den

Zielrechner übertragene Programm („Zielprogramm“) auswirken. Beispielsweise können Sie das Zielprogramm starten und stoppen, schrittweise abarbeiten oder bis zu einem vorher festgelegten Haltepunkt („Breakpoint“) laufen lassen. Weiterhin können Sie Register- und Speicherinhalte des Zielrechners anzeigen lassen und manuell modifizieren. Auch ein manueller Zugriff auf die Ein- und Ausgabeports des Zielrechners ist möglich.

Um die Arbeit komfortabel zu gestalten, kann der Debugger die aktuelle Position im Quelltext anzeigen und Informationen über die benutzten symbolischen Bezeichner bereit halten. Beachten Sie aber, dass diese Informationen aus der Entwicklungsumgebung stammen und nicht aus dem Zielrechner. Im Zielrechner selbst ist nur der binäre Maschinencode vorhanden.

Ziel des Debuggens ist es, die korrekte Funktion des erzeugten Programms zu überprüfen. In der Regel wird beim ersten Programmstart noch nicht das gewünschte Ergebnis erscheinen. Dann müssen Sie mit Hilfe der Debuggerfunktionen die mehr oder weniger versteckten Programmfehler suchen. Zur Korrektur der Fehler verlassen Sie den Debugger, editieren den Quelltext und führen die Schritte zum Übersetzen und Debuggen erneut aus.

Die Fehlersuche ist beendet, wenn sich das Programm im normalen Laufmodus wunschgemäß verhält. Um das zu überprüfen, ist im Sinne der jeweiligen Aufgabenstellung mit den Bedien- und Anzeigeelementen des Zielrechners zu experimentieren.

Der Versuchsaufbau und die Praktikumssoftware verfügen über eine Anzahl von Bestandteilen und Funktionen, welche in diesem Praktikum nicht benutzt werden. Diese sind in der Anleitung nicht oder nur am Rande erwähnt. Bitte benutzen Sie keine Funktionen, die in dieser Anleitung nicht erwähnt werden und deren Bedeutung Ihnen nicht klar ist!

2.2 Hardware des Zielrechners

Der Zielrechner ist ein eigenständiges Rechnersystem in einer von-Neumann-Architektur mit Prozessorbaugruppe, Speicherbaugruppen (RAM und ROM) und mehreren Ein- und Ausgabebaugruppen sowie eigener Stromversorgung. In diesem Abschnitt erhalten Sie grundlegende Informationen über die vorhandenen Baugruppen. Detaillierte Beschreibungen der Anschlüsse und Bedienelemente sowie Hardware-Adressen und dergleichen sind im Kapitel „Bedienungsanleitung“ sowie im Anhang enthalten.

Als Prozessor wird der Typ „Élan™ SC400“ der Firma AMD verwendet. Dieser enthält einen CPU-Kern, der zum Intel-Typ i486 kompatibel ist und damit die x86-Programmiermodelle bis einschließlich IA-32 anbietet. Auf dem Prozessorchip sind zusätzlich einige Baugruppen integriert, die bei anderen Rechnerkonzepten mit getrennten Bausteinen realisiert werden müssten. Dadurch wird die Gesamtschaltung vereinfacht.

Der Prozessor befindet sich auf einem Steckmodul von der Größe einer Kreditkarte. Dieses enthält in hoher Packungsdichte eine Reihe weiterer Baugruppen, darunter insbesondere sämtliche Speicherbaugruppen. Die Grundleiterplatte des Gesamtgerätes enthält außerhalb des Steckmoduls fast ausschließlich Ein- und Ausgabebaugruppen, welche in den verschiedenen Praktikumsversuchen benutzt werden sollen.

> Bemerkung:
> Aus Softwaresicht ist es unerheblich, ob sich die angesprochene Speicher- oder Ein-/Ausgabebaugruppe auf der Grundleiterplatte, auf dem Steckmodul oder auf dem Prozessorchip befindet.

Zur Verdeutlichung folgt hier eine Auflistung der wesentlichen Baugruppen und Funktionen:

- Prozessor „Élan™ SC400“ (AMD, Inc.):
  - i486-kompatibler CPU-Kern
  - Memory Management Unit (MMU)
  - Zwei Direct Memory Access Units (DMA)
  - Zwei Programmierbare Interrupt Controller (PIC)
  - Speicherinterface für sRAM, dRAM und ROM
  - Asynchrone serielle Ein- und Ausgabe („Seriell 1“)
  - Businterface für externen Systembus
  - Power-Management

- Viele weitere Funktionen

- Steckmodul „miniModul-486“ (Phytec GmbH):
  - Prozessor „Élan™ SC400“ (33 MHz)
  - Takt- und Resetschaltung
  - Dynamischer RAM
  - Statischer RAM (mit Pufferbatterie)
  - ROM (E²PROM)
  - Asynchrone serielle Ein- und Ausgabe („Seriell 2“)
  - Weitere Funktionen

- Grundleiterplatte:
  - Steckplatz für das Steckmodul
  - Zwei digitale parallele Ausgabebaugruppen mit je acht LEDs (Lichtemittierende Dioden)
  - Digitale parallele Eingabebaugruppe mit acht Schaltern (oben Ein, unten Aus)
  - Digitale parallele Eingabebaugruppe mit acht Tasten (selbsttätiger Rückgang)
  - Digitale parallele Ausgabebaugruppen mit zwölf Sieben-Segment-Anzeigen
  - Digitale Parallele Ein- und Ausgabebaugruppe mit Matrixtastatur (4x4 Tasten) und Anzeige-LEDs für die Zeilen- und Spaltensignale
  - Programmierbarer Paralleler Interfacebaustein (PPI) vom Typ 8255A mit Anzeige-LEDs für die Datenleitungen von Port A sowie Anzeige-LEDs bzw. Bedientasten für die Handshakesignale und Interruptsignale von Port A
  - Programmierbarer Intervalltimer (PIT, auch „Zähler/Zeitgeber“ oder „Timer“ genannt) vom Typ 8254 (aufwärtskompatibel zum 8253) mit Taktgenerator (2 MHz) für die Kanäle 0 und 1, Kaskadierung der Kanäle 1 und 2 sowie Anzeige-LED für das Ausgangssignal von Kanal 2.
  - Analoge Ausgabebaugruppe mit Digital-Analog-Converter (DAC) mit 8 bit Auflösung

- Zwei Anschlussbuchsen für die Ausgangssignale von PIT-Kanal 0 und vom DAC
- Kleinlautsprecher mit Niederfrequenzverstärker und Quellenwahlschaltern für PIT-Kanal 0 und DAC
- Anschlussbuchsen für die beiden seriellen Anschlüsse
- Bedientasten für Reset (Rücksetzen) und NMI (Nichtmaskierbarer Interrupt)
- Stromversorgung und Betriebsanzeige

Die Bauelemente auf der Grundleiterplatte sind durch eine Klarsichtabdeckung hindurch sichtbar. Auf dem Steckmodul sind wegen der beidseitigen engen Bestückung nicht alle Bauelemente erkennbar.

Bei den einzelnen Praktikumsversuchen wird stets nur ein Teil der aufgelisteten Baugruppen und Funktionen benutzt. Die Verbindung zum Hostrechner erfolgt immer am Anschluss „Seriell 1“.

2.3 Ablauf der Softwareentwicklung

Für die Softwareentwicklung wird die professionelle Entwicklungsumgebung CAD-UL Workbench der CAD-UL AG benutzt. Der Start erfolgt über 'Workbench' auf dem Desktop oder im Startmenü-Eintrag '*CAD-UL Embedded Tools - Workbench*'.

Beim Start sollte automatisch ein Projekt mit dem Namen k_prak geöffnet sein. Dieses enthält im Projektordner Source einen Assemblerquelltext mit dem Dateinamen kprak.asm. In dieser Datei sollen Sie Ihre Programme realisieren. Es sind bereits einige Anweisungen vorhanden, die sich um die nötigen Deklarationen und Einstellungen kümmern. Dieses „Skelett" lassen Sie bitte unverändert! Die Stellen zum Einfügen der Programm- und Datenbereiche Ihres Programms sind durch Kommentare gekennzeichnet.

Im Projekt sind weitere Dateien enthalten, die bei der Übersetzung benötigt werden und die Sie nicht verändern sollen. Die eventuell vorhandene Datei README.TXT beschreibt nachträgliche Änderungen und Ergänzungen zu dieser Anleitung, welche Sie zur Kenntnis nehmen sollten.

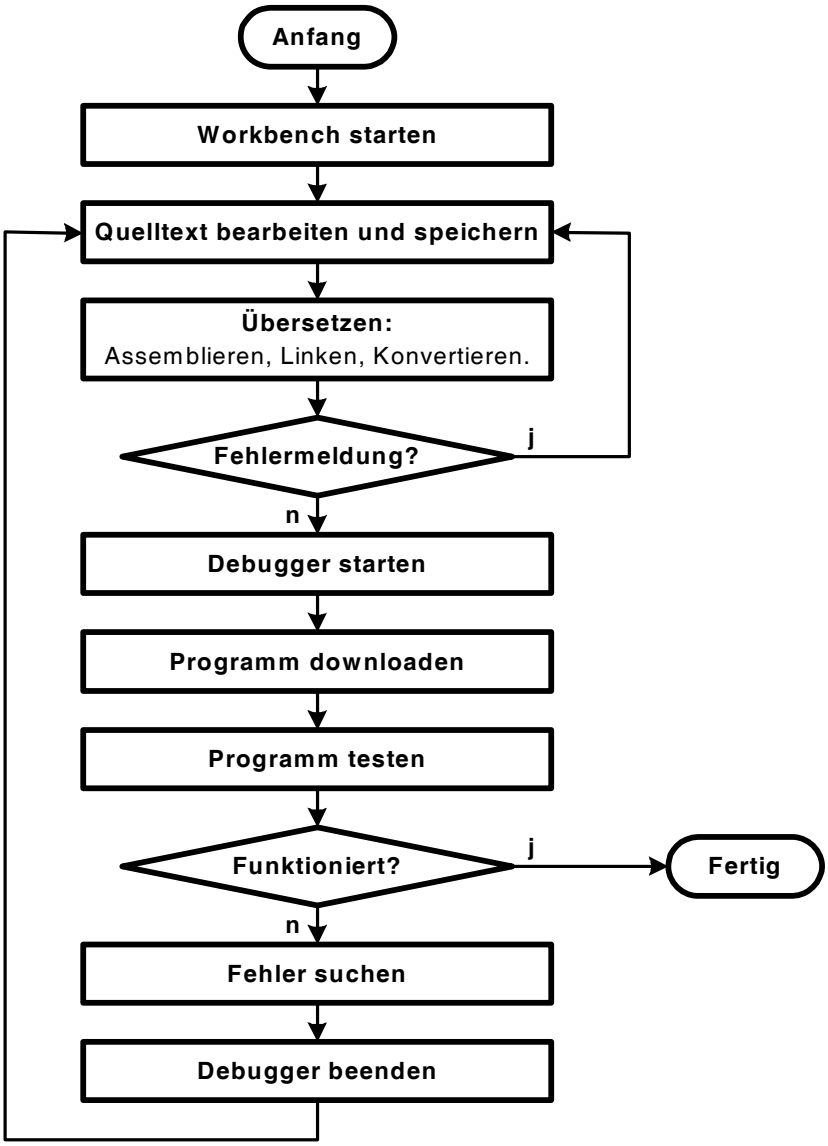
Die Bedienung der Werkzeuge und die Quelltextstruktur werden im Kapitel „Bedienungsanleitung" genauer beschrieben. Bitte beachten Sie, dass die Menüs, Dialoge und Fehlermeldungen ausschließlich in englischer Sprache vorliegen.

Der prinzipielle **Ablauf der Softwareentwicklung** ist im folgenden Bild dargestellt und wird im Weiteren beschrieben.

Der Quelltext mit dem Namen kprak.asm erscheint in einem Dokumentenfenster. Hier können Sie Ihr Programm eingeben, wobei Sie durch einen Editor mit Syntax-Hervorhebung unterstützt werden. Anschließend speichern Sie die Datei ('Save') und starten den Übersetzungsvorgang ('Rebuild all').

Die Übersetzung läuft intern in drei Stufen ab:

- Assemblieren (Werkzeug AS386): Der Assembler-Quelltext kprak.asm wird in den binären Maschinencode übersetzt. Das Ergebnis ist die so genannte Objektdatei kprak.obj. Dieser Programmcode ist aber noch nicht auf dem Zielrechner ausführbar, da z.B. die endgültigen Speicheradressen fehlen.
- Linken (Werkzeug LINK386): Die Objektdatei wird in das so genannte „Objekt Module Format" (OMF) gebracht. Hierbei werden die endgültigen Speicheradressen festgelegt und einige zusätzliche Informationen eingebaut. Bei größeren Projekten können



in diesem Schritt mehrere Objektdateien zu einem Programm verbunden werden. Das Ergebnis ist hier die Datei k_prak.omf. Es sind weitere Hilfsdateien beteiligt, die hier nicht erwähnt werden.

- Konvertieren (Werkzeug OMF386BND): In diesem Schritt wird durch einen „Debug-Präprozessor“ eine ladbare Datei für den Debugger erzeugt. Dabei werden weitere Zusatzinformationen für den Debugger eingebunden. Das Ergebnis ist die Datei k_prak.bd. Diese enthält also den endgültigen binären Maschinencode für den Zielrechner und Informationen für den Debugger.

Diese drei Übersetzungsvorgänge laufen im Praktikum als ein Schritt ab. In einem Outputfenster innerhalb der Arbeitsumgebung kontrollieren Sie, ob alle drei Schritte stattgefunden haben und ob dabei Fehler gemeldet wurden. Im Falle von Fehlermeldungen, welche in der

Regel auf fehlerhafter Syntax des Quelltextes beruhen, müssen Sie die jeweiligen Fehler im Quelltext beseitigen und die Übersetzung erneut starten.

Nach fehlerfreiem Übersetzungsvorgang aktivieren Sie die Funktion ‘Debugger’. Der Debugger erscheint als eigene Applikation mit dem Namen CAD-UL XDB. Übertragen Sie stets zuerst die erzeugte Software auf den Zielrechner (‘Download’). Dabei wird der endgültige Maschinencode aus der zuvor erzeugten Datei in den RAM-Speicher des Zielrechners geschrieben.

Die weitere Funktion des Debuggers besteht darin, den Test des auf dem Zielrechner laufenden Programms vom Hostrechner aus zu ermöglichen und den Benutzer bei der Fehlersuche zu unterstützen. Ihr Programm benutzt also die Hardware des Zielrechners und beeinflusst die dort vorhandenen Ports. Die Steuerung des Testvorgangs erfolgt jedoch vom Hostrechner aus. Dabei setzt der Debugger Ihre Eingaben in spezielle „Fernsteuerbefehle“ um, welche zum Zielrechner übertragen werden. Man nennt diese Art des Programmtests auch „Remote Debugging“.

Durch Auswertung der Zusatzinformationen aus dem Übersetzungsvorgang ist der Debugger in der Lage, in einem Fenster, welches den Quelltext des Programms anzeigt, die aktuelle Position der Abarbeitung zu kennzeichnen. Das ist zu Beginn natürlich der erste Befehl. Bitte beachten Sie, dass das Quelltextfenster hier nur der Information dient. Sie können jetzt keine Änderungen am Quelltext vornehmen, da die entsprechenden Änderungen am Maschinencode eine neue Übersetzung erfordern würden.

Mit Hilfe der Funktionen des Debuggers können Sie nun das Programm untersuchen. Ziel ist es, dass sich das Programm nach dem Starten im Laufmodus (‘Run’) entsprechend der Aufgabenstellung verhält.

Zur Fehlersuche können Sie das Programm anhalten, schrittweise abarbeiten oder bis zu einem vorher festgelegten Haltepunkt („Breakpoint“) laufen lassen. Sie können Register- und Speicherinhalte inspizieren und verändern oder auch manuell auf Ein- und Ausgabeports zugreifen. Für Details sei wieder auf das Kapitel „Bedienungsanleitung“ verwiesen.

Das Finden der mitunter recht versteckten Fehler ist nur durch planvolles Vorgehen möglich. Dazu gehört, dass Sie sich das erwartete Verhalten jeder einzelnen Anweisung überlegen und dieses dann experimentell überprüfen. Es können sich Denkfehler bei der Vorbereitung herausstellen oder auch Tippfehler bei der Quelltexteingabe, welche rein zufällig keinen Syntaxfehler zur Folge hatten.

Zur Korrektur der Fehler muss der gesamte Ablauf wiederholt werden. Dazu beenden Sie den Debugger (‘Exit’). Im Quelltextfenster der Workbench nehmen Sie die erforderlichen Änderungen vor und führen erneut die Übersetzung durch. Danach testen Sie das Ergebnis mit dem Debugger. Diesen Zyklus führen Sie so lange aus, bis Ihr Programm fehlerfrei funktioniert.

3. Praktikumsaufgaben

3.1 Versuch **A1**: Lauflicht

In diesem Versuch lernen Sie den Versuchsaufbau anhand einfacher Assemblerprogramme kennen. Dabei benutzen Sie parallele digitale Ein- und Ausgabebaugruppen, die mit den LED-Reihen, der Tasten- und der Schalterreihe sowie den Sieben-Segment-Anzeigen verbunden sind.

Grundaufgabe a)

Es ist ein fertiges Assemblerprogramm vorgegeben. Dieses finden Sie am Ende von Abschnitt 3.1. Die Kommentare sind absichtlich sehr knapp gehalten, da Sie die Funktion des Programms selbst ermitteln sollen. Zur **Vorbereitung** untersuchen Sie bitte das Programm und notieren die vermuteten Funktionen der einzelnen Abschnitte! Um die Reaktionen des Zielrechners bei den IN- und OUT-Befehlen zu bestimmen, müssen Sie die Informationen aus dem Kapitel „Bedienungsanleitung“ und dem Anhang benutzen.

> Erläuterung zum vorgegebenen Programm:
> Der spezielle Operator OFFSET dient dazu, die **Adresse** zum angegebenen Symbol zu ermitteln und als Zahlenwert zu übergeben. Mit
>
> ```
> MOV     reg32,OFFSET symbol
> ```
>
> befördern Sie also den Zahlenwert der zum Symbol gehörenden Adresse in das Register. Falls das Symbol zu einer Speicherzelle gehört, kann das Register anschließend zur indirekten Adressierung dieser Speicherzelle benutzt werden.

Bei der **Durchführung** sollen Sie dieses Programm eingeben, übersetzen und mit dem Debugger untersuchen. Die Arbeitsschritte sind im Folgenden einzeln beschrieben. Dadurch sollen Sie die Bedienung der Werkzeuge kennen lernen und einige wichtige Funktionen des Debuggers ausprobieren. Außerdem erhalten Sie eine Vorlage für den Test Ihrer eigenen Programmentwürfe.

Der Praktikumsbetreuer wird Sie möglicherweise später zu Ihren Beobachtungen befragen. Machen Sie sich deshalb entsprechende Notizen.

Arbeitsschritte:

- **Programm eingeben:** Starten Sie die Workbench, öffnen Sie im vorbereiteten Projekt die Datei kprak.asm (evtl. ist sie bereits sichtbar) und geben Sie das vorgegebene Programm ein. Der Programmbereich und der Datenbereich müssen jeweils an den vorgesehenen Stellen des vorbereiteten Quelltextes eingefügt werden. Die Kommentare brauchen Sie nicht mit einzutippen. Die Groß- und Kleinschreibung ist unerheblich. Vergessen Sie nicht das regelmäßige Speichern der Datei.
- **Programm übersetzen und Fehler im Quelltext korrigieren:** Starten Sie mit 'Rebuild all' den Übersetzungsvorgang, warten Sie dessen Ende ab und inspizieren Sie den Output. Falls Errors oder Fatals gemeldet werden, müssen Sie den Quelltext korrigieren und erneut übersetzen. Warnings und Notes erfordern meistens keine Eingriffe. *(Hinweis: Achten Sie immer auf diese Fehlerausgabe. Wurde Ihr Programm wegen Fehlern nicht vollständig übersetzt, lädt der Debugger die vorhergehende Version und arbeitet mit dieser, was die weitere Fehlersuche erschwert.)*
- **Zielsystem einrichten:** Am Zielrechner (x86-Trainingssystem im Koffergehäuse) muss die Betriebsanzeige leuchten. Drücken Sie die Reset-Taste am Zielrechner **(nicht am PC!)** und warten Sie einige Sekunden auf das Verlöschen aller sonstigen Anzeigen. *(Hinweis: Dieser Schritt ist nur beim ersten Start des Programms nötig.)*.
- **Debugger starten**: Starten Sie den Debugger XDB mit der Funktion 'Debugger'. Bestätigen Sie den Start im erscheinenden kleinen Fenster.
- **Programm auf das Zielsystem übertragen:** Drücken Sie im Debugger den Button 'Download' und warten Sie ab, bis zuerst das Fenster mit dem Quelltext und dann im Command-Fenster der Prompt erscheinen. Der binäre Maschinencode Ihres Programms befindet sich nun im RAM-Speicher des Zielssystems.
- **Anzeige von Registerinhalten:** Aktivieren Sie die Menüfunktion '*Display - Register*'. Orientieren Sie sich über die interessierenden Register (siehe „Arbeitsblätter zur Übung", Abschnitt 2.1).
- **Anzeigen von Speicherinhalten:** Aktivieren Sie zweimal nacheinander die Menüfunktion '*Display - Memory*' mit folgenden Parametern:
  - Address: ziff Size: Byte Format: hex. Auto Update
  - Address: verzoe Size: Long Format: hex. Auto Update

- Überprüfen Sie, ob die Speicherinhalte aus dem Datenbereich des Quelltextes auffindbar sind. Ordnen Sie alle Fenster sinnvoll an.
- **Schrittweises Abarbeiten des Programms:** Führen Sie einige Befehle im Schrittbetrieb aus ('Instruction step'). Beobachten Sie bei jedem Schritt die Position im Quelltext und die Veränderungen in Registern und Speicher (Reaktion abwarten). Der OUT-Befehl sollte eine Reaktion am Zielrechner bewirken. Vergleichen Sie alle Beobachtungen mit Ihrer Vorbereitung. *(Hinweis: Das Aktualisieren der Anzeige nach dem Abarbeiten eines Befehls braucht einige Zeit. Bitte warten Sie nach jedem Klick so lange, bis im Command-Fenster wieder der Prompt erscheint.)*
- **Programmabarbeitung an vorgegebener Stelle anhalten:** Wenn Sie innerhalb des Unterprogramms zeit angekommen sind, setzen Sie einen Unterbrechungspunkt (Breakpoint) auf den Befehl RET. Das geschieht durch einen Doppelklick auf den blauen Punkt am Zeilenanfang. Es erscheint ein rotes „Stoppschild".
- Drücken Sie jetzt auf den Button 'Run'. Die Ausführung startet und bleibt nach kurzer Zeit am Breakpoint stehen. Betrachten Sie erneut die Registerinhalte.
- Drücken Sie wieder 'Run' und warten Sie bis zum nächsten Anhalten. Am Zielrechner sollte sich eine Änderung zeigen.
- **Programm frei laufen lassen:** Entfernen Sie jetzt den Breakpoint, indem Sie das „Stoppschild" doppelklicken. Wenn Sie jetzt 'Run' drücken, beginnt das Programm ungebremst zu laufen. Schauen Sie sich die Reaktionen am Zielrechner an.
- Eine der blauen Tasten in der Tastenreihe am Zielrechner bewirkt eine Reaktion des Programms. Bei Ihrer Vorbereitung sollten Sie erkannt haben, welche Taste das ist. Drücken Sie diese Taste und beobachten Sie die Wirkung.
- **Verzweigungen beobachten:** Halten Sie das Programm mit dem 'Stop'-Button an. Setzen Sie einen Breakpoint auf den IN-Befehl. Lassen Sie das Programm mit 'Run' bis zum Breakpoint weiterlaufen.
- Halten Sie die oben erwähnte blaue Taste gedrückt und führen Sie einige Einzelschritte aus, bis der Sprungbefehl stattgefunden hat. Beobachten Sie dabei insbesondere das Carry-Flag (Buchstabe C ganz rechts bei EFLAGS).

- Lassen Sie das Programm nochmals mit 'Run' bis zum Breakpoint laufen (Zeit abwarten!) und führen Sie die gleichen Einzelschritte jetzt ohne gedrückte Taste aus. Beobachten und erklären Sie die Unterschiede.
- Entfernen Sie jetzt den Breakpoint. Das Programm soll noch angehalten bleiben.
- **Direkter Zugriff auf Ein- und Ausgabe:** Verwenden Sie die Funktion '*Debug – Read/Write Port*', um auf das Byte-Ausgabeport mit der Adresse B0H (hier als 0xB0 anzugeben) ein Byte mit dem Wert Null zu schreiben. Beobachten Sie, ob eine Reaktion auftritt. Schließen Sie dann das Dialogfenster.
- **Ändern von Speicherinhalten:** Verändern Sie den Inhalt der Speicherzelle verzoe, indem Sie den aktuellen Wert im entsprechenden Speicherfenster doppelklicken. Der neue Wert soll 100000H sein (einzugeben als 0x00100000). *(Hinweis: Auch Registerinhalte lassen sich per Doppelklick ändern.)*
- Starten Sie das Programm erneut mit 'Run' und beobachten Sie die Veränderungen.
- **Debugger beenden:** Halten Sie das Programm mit dem 'Stop'-Button an, wählen Sie danach 'Exit' und bestätigen Sie das Beenden des Debuggers im kleinen Fenster.

Sollten Sie während des Debuggens Fehler in Ihrem Programm feststellen, müssen Sie den Debugger beenden, die Fehler im Quelltext korrigieren und alle Schritte ab dem Übersetzen wiederholen.

Wenn Sie alle Schritte erfolgreich absolviert haben, bringen Sie den Quelltext in den Originalzustand und gehen zur Aufgabe b) über.

Grundaufgabe b)

Jetzt sollen Sie ein eigenes Programm entwerfen, testen und weiterentwickeln. Nachfolgend finden Sie eine mehrstufige Aufgabenstellung. In der **Vorbereitung** notieren Sie bitte das selbst entworfene Programm.

Bei der **Durchführung** sollen Sie das Programm mit den inzwischen bekannten Methoden eingeben und testen. Gehen Sie dabei stufenweise vor. Das bedeutet, dass Sie jede Teilfunktion nach der Realisierung bereits testen sollten, bevor Sie mit der Eingabe fortfahren. Dadurch vereinfacht sich die Fehlersuche.

Aufgabenstellung:

- Es soll ein „Lauflicht“ programmiert werden. Auf der rechten LED-Reihe soll ein sichtbarer Lichtpunkt von links nach rechts laufen und dann immer wieder von links beginnen.

  Bemerkung:
  Für eine erkennbare Anzeige benötigen Sie eine Zeitverzögerung. Sie können sich am Unterprogramm zeit in Aufgabe a) orientieren. Der Parameter für die Verzögerung muss sinnvoll gewählt werden, da der Variationsbereich von einigen hundert Nanosekunden bis zu mehreren Minuten reicht.

- Jetzt implementieren Sie eine Umschaltfunktion für die Geschwindigkeit. Mit dem linken Schalter der Schalterreihe soll sich die Geschwindigkeit zwischen „schnell“ (Schalter oben) und „langsam“ (Schalter unten) umschalten lassen.

  Bemerkung:
  Beachten Sie, dass die IN- und OUT-Befehle (für Bytes) nur mit dem Register AL arbeiten können.

- Eine weitere Umschaltfunktion soll jetzt die Bewegungsrichtung des Lichtpunktes steuern. Der rechte Schalter der Schalterreihe soll zwischen „nach links“ und „nach rechts“ wählen. Die Geschwindigkeitsumschaltung soll weiterhin funktionieren.

- Nun soll das Drücken jeder beliebigen Taste der blauen Tastenreihe dazu führen, dass das Anzeigemuster invertiert wird. Damit ist gemeint, dass anstelle des hellen Punktes ein dunkler Punkt durchläuft. Die invertierte Darstellung soll nur so lange erscheinen, wie die Taste gedrückt gehalten wird. Die anderen Umschaltungen sollen weiterhin funktionieren.

- Führen Sie Ihr Programm dem Praktikumsbetreuer vor.

Als **Zusatzaufgabe** erweitern Sie das Programm nach eigenen Ideen. Im Folgenden erhalten Sie dafür einige Anregungen:

- Weitere Bewegungsvarianten, z.B. hin und her pendelnd, symmetrisch zur Mitte oder unter Verwendung beider LED-Reihen.

- Automatischer Wechsel unterschiedlicher Bewegungsvarianten.

- Bewegungseffekte auf den Sieben-Segment-Anzeigen (z.B. durchlaufende Ziffern).

- Kontinuierliche Steuerung der Anzeigehelligkeit durch sehr schnelles Ein- und Ausschalten mit variablem Hell-Dunkel-Verhältnis.

Hinweis:
Mitunter ist es zweckmäßig, bei den IN- und OUT-Befehlen die Portadresse als Registerinhalt angeben zu können. Für diesen Zweck existieren folgende Operandenkombinationen:

```
IN    AL,DX
OUT   DX,AL
```

In beiden Fällen enthält DX die Portadresse und AL das übertragene Datenbyte. Andere Registerzuordnungen sind bei 8 bit Datenbreite nicht möglich.

Vorgegebenes Programm für Versuch A1, Aufgabe a)

; Programmbereich:

```
anf:    MOV   EDX,400000H
        MOV   [verzoe],EDX

m1:     MOV   EDI,10
        MOV   ESI,OFFSET ziff   ;Adresse von ziff ins Register bringen
                                ;(siehe Erlaeuterung)
m2:     MOV   AL,[ESI+EDI-1]
        OUT   0B0H,AL
        CALL  zeit
        DEC   EDI
        JNZ   m2

        MOV   AL,0FFH
m3:     OUT   5CH,AL
        NOT   AL
        OUT   5DH,AL
        CALL  zeit

        MOV   BL,AL         ;Inhalt von AL wird noch gebraucht
        IN    AL,59H
        BT    EAX,7
        MOV   AL,BL         ;AL bekommt alten Wert zurueck (CF bleibt)
        JC    m1
        JMP   m3
```

;zeit ist ein Unterprogramm, welches nur Zeit verbrauchen soll:

```
zeit:   MOV   ECX,[verzoe]
z1:     DEC   ECX
        JNZ   z1
        RET
```

; Datenbereich:

```
verzoe DD     ?             ;Eine Speicherzelle (Doppelwort).
                            ;Anfangsbelegung ist undefiniert.

ziff   DB     3FH,03H,6DH,67H,53H,76H,7EH,23H,7FH,77H

                            ;Zehn Speicherzellen (Bytes).
                            ;Anfangsbelegung sind die Sieben-Segment-
                            ;Kodierungen der Ziffern Null bis Neun.
```

3.2 Versuch **A2**: Timerbaustein

In diesem Versuch arbeiten Sie mit einem programmierbaren Interfacebaustein, der über eigene Register angesprochen wird. Als Beispiel dient ein Programmierbarer Intervalltimer (PIT, auch als „Zähler-Zeitgeber-Baustein“ oder „Timerbaustein“ bezeichnet) vom Typ 8254. Dieser ist eine kompatible Weiterentwicklung des Typs 8253, der in den „Arbeitsblättern zur Übung“ auszugsweise beschrieben ist.

Diese Beschreibung ist für den vorliegenden Versuch ausreichend. Werfen Sie aber auch einen Blick in den Abschnitt „Zähler- und Zeitgeberbaugruppen“ der Vorlesung. Bei besonderem Interesse können Sie auf der Webseite (siehe 1.3) das vollständige Datenblatt des Bausteins 8254 finden. Die Bedienungsanleitung des unten erwähnten Oszilloskops kann ebenfalls der Webseite entnommen werden.

In der **Vorbereitung** ermitteln Sie bitte die Zählkonstanten für die benötigten Frequenzteilungen und dann die Initialisierungs-Sequenzen für den Timerbaustein für die einzelnen Aufgaben. Notieren Sie selbst entworfene Programme, in denen diese Initialisierungs-Sequenzen umgesetzt werden.

> Hinweis:
> Beachten Sie, dass zum vollständigen Initialisieren **eines** Kanals des PIT drei Ausgabeoperationen in einer festen Reihenfolge notwendig sind:
> - Steuerbyte mit Kanalnummer, Betriebsart und Ankündigung von LSB und MSB: auszugeben auf die Steuerbyte-Adresse
> - LSB (niederwertiges Byte) der Zählkonstante: auszugeben auf die Kanal-Adresse
> - MSB (höherwertiges Byte) der Zählkonstante: auszugeben auf die Kanal-Adresse

Grundaufgabe a)

Der Kanal 0 des Timerbausteins soll als programmierbarer Frequenzgenerator benutzt werden. Dazu wird die Betriebsart „Mode 3“ verwendet (Frequenzteiler mit symmetrischer Rechteckschwingung am Output). Die Output-Frequenz soll 440 Hz betragen (in der Musik entspricht das dem so genannten Kammerton a’). Als Input benutzen Sie den eingebauten 2-MHz-Generator.

Die Initialisierung soll durch ein kurzes Assemblerprogramm erfolgen, welches nach dem Initialisierungsvorgang in einer zunächst leeren Endlosschleife läuft.

Bei der **Durchführung** geben Sie das Programm ein und testen, ob die gewünschte Funktion erreicht wird. Benutzen Sie vor jedem Programmstart den Button ‘Stop_Timer’ im Debugger,

um die Kanäle des PIT in den Grundzustand zu bringen! Der Timer läuft nämlich beim Anhalten des Programms weiter.

Das Output-Signal des PIT-Kanals 0 können Sie hören, wenn Sie den Wahlschalter PIT einschalten (obere Schalterstellung). Dazu muss aber ein Signal mit einer hörbaren Frequenz vorliegen.

Falls der Praktikumsplatz mit einem Oszilloskop ausgestattet ist, so benutzen Sie dieses zur zusätzlichen Kontrolle. Schließen Sie es an der Ausgangsbuchse PIT an.

Zum Experimentieren mit der Initialisierungssequenz können Sie auch die Debuggerfunktion '*Debug - Read/Write Port*' benutzen. Sie brauchen dann nicht ständig das Programm zu ändern.

Grundaufgabe b)

Schalten Sie die Tonausgabe zunächst wieder ab und erweitern Sie das Programm um die Initialisierung der PIT-Kanäle 1 und 2. Die am Output des Kanals 2 angeschlossene LED soll mit einer Periodendauer von 0,5 s blinken. Es ist wiederum Mode 3 zu benutzen. Da beide Kanäle hintereinander geschaltet (kaskadiert) sind, müssen Sie die benötigte Frequenzteilung auf beide Kanäle aufteilen. Außer der LED haben Sie diesmal keine weitere Kontrollmöglichkeit.

Achten Sie auch jetzt darauf, vor jedem Test den Button 'Stop_Timer' zu betätigen!

Grundaufgabe c)

Die Tonausgabe von Kanal 0 wird wieder eingeschaltet. Sie soll jetzt aber nur noch dann aktiv sein, wenn gerade eine beliebige Taste in der blauen Tastenreihe gedrückt ist. Dazu müssen Sie in der Endlosschleife des Programms eine entsprechende Abfrage einbauen.

> Hinweis:
> Der PIT-Kanal stoppt automatisch nach der Ausgabe des Steuerbytes und startet wieder nach der vollständigen Ausgabe der Zählkonstanten.

Fortgeschrittene Aufgabe d)

Erweitern Sie das Programm dann so, dass den einzelnen Tasten unterschiedliche Frequenzen zugeordnet sind. Als Vorlage können Sie die in der nachfolgenden Tabelle gezeigte Tonleiter verwenden. Es wird angenommen, dass nicht mehrere Tasten gleichzeitig gedrückt werden.

Das Blinken der LED von Aufgabe b) soll weiterhin funktionieren. Führen Sie dem Praktikumsbetreuer das fertige Programm vor.

Frequenzen für die C-Dur-Tonleiter (eingestrichene Oktave, gerundete Werte)								
Ton	c’	d’	e’	f’	g’	a’	h’	c’’
f (Hz)	261,6	293,7	329,6	349,2	392,0	440,0	493,9	523,2

Als **Zusatzaufgabe** erweitern Sie das Programm nach eigenen Ideen. Im Folgenden erhalten Sie dafür einige Anregungen:

- Verwendung von Schaltern aus der Schalterreihe, um den Tonumfang durch Umschaltung zu erweitern (andere Tonarten, andere Oktaven)
- Frequenzeinstellung der blinkenden Leuchtdiode durch Bedienelemente
- Automatisches Abspielen von Tonfolgen (z.B. Martinshorn, Sirene) oder Melodien
- „Melodietrainer“: Ein Leuchtpunkt zeigt die nächste zu drückende Taste an.
- Tonerzeugung mit „Vibrato“ (Frequenzmodulation mit einer Modulationsfrequenz von einigen Hertz und einem Frequenzhub von bis zu drei Prozent)
- Realisierung anderer Kurvenformen des Tonsignals (z.B. Dreieck oder Sägezahn) unter Verwendung des Analogausganges (DAC)

Hinweis für „musikalische“ Experimente:
Die Frequenzen der zwölf Halbtöne der chromatischen Tonleiter bilden eine geometrische Folge mit dem Faktor $2^{1/12}$ (etwa 1,05946). Ein Oktavschritt entspricht exakt einer Halbierung bzw. Verdoppelung der Frequenz. Ausgehend von a’ = 440 Hz können Sie damit die Frequenzen sämtlicher Töne berechnen.

3.3 Versuch **A3**: Matrixtastatur

Eine Matrixtastatur ist eine spezielle Art der Ansteuerung eines Tastenfeldes über parallele digitale Ein- und Ausgabebaugruppen. Die Tasten bilden eine gedachte zweidimensionale Matrix und werden mit Zeilen- und Spaltenleitungen verbunden. Mit n Zeilenleitungen und m Spaltenleitungen kann man n x m Tasten bedienen. Besonders bei großen Tastenfeldern ist das eine deutliche Verringerung des Aufwandes.

Der Vorteil wird jedoch durch eine kompliziertere Software erkauft. Um eine gedrückte Taste zu identifizieren, müssen alle Zeilen der Matrix nacheinander abgefragt werden. Bei jedem Abfrageschritt erhält man die Information über die gedrückten Tasten jeweils einer Zeile.

Bei diesem Versuch probieren Sie dieses Prinzip mit der 4x4-Matrixtastatur des Versuchsaufbaus aus. Zur **Vorbereitung** informieren Sie sich bitte in dieser Anleitung über die genaue Funktion der vorhandenen Matrixtastatur und notieren selbst entworfene Programme für die einzelnen Aufgaben.

Grundaufgabe a)

Zunächst werden zwei Unterprogramme benötigt, welche die Abfrage der Matrixtastatur durchführen. Das **Unterprogramm matr** soll alle Zeilen der Matrix je einmal abfragen und dann unverzüglich zurückkehren. Falls eine gedrückte Taste erkannt wurde, soll in einem selbst gewählten Byteregister eine von Null verschiedene Tastennummer übergeben werden.

Die Tastennummern legen Sie selbst fest. Sie müssen keine zusammenhängende Folge bilden. Es ist lediglich notwendig, dass es sich um unterschiedliche 8-bit-Zahlen handelt. Natürlich müssen Sie die festgelegten Tastennummern notieren.

Falls keine Taste gedrückt ist, soll der Wert Null übergeben werden. Das gleichzeitige Drücken mehrerer Tasten brauchen Sie nicht gesondert auszuwerten.

Ein darauf aufbauendes **Unterprogramm wmatr** soll in ähnlicher Weise funktionieren. Allerdings soll es so beschaffen sein, dass es erst beim Erkennen eines Tastendrucks zurückkehrt. Das bedeutet, dass das Unterprogramm das Drücken einer Taste abwartet und dann deren Nummer übergibt. Falls beim Aufruf bereits eine Taste gedrückt ist, soll zunächst deren Loslassen und dann das Drücken der nächsten Taste abgewartet werden.

Bei der **Durchführung** geben Sie diese Unterprogramme ein und testen diese nacheinander mit Hilfe eines kurzen Hauptprogramms. Das Hauptprogramm soll innerhalb einer „ewigen“ Schleife jeweils eines der Unterprogramme aufrufen und die erkannte Tastennummer binär auf einer der LED-Zeilen anzeigen. Untersuchen Sie damit nacheinander die unterschiedliche Funktion Ihrer beiden Unterprogramme. Richten Sie sich darauf ein, dass der Praktikumsbetreuer diesen Test sehen möchte.

Im Einzelschrittbetrieb können Sie den Vorgang der Tastenabfrage an den LEDs neben der Matrixtastatur genau beobachten.

> Hinweise:
> Das Hauptprogramm muss immer am Beginn des Programmbereichs stehen. Achten Sie unbedingt darauf, dass sich die Programmabarbeitung nach dem Ende des Hauptprogramms nicht einfach in das erste Unterprogramm fortsetzen darf. Die „ewige Schleife“ laut Aufgabenstellung soll das verhindern. Über die Gestaltung von Unterprogrammen können Sie sich z.B. in der Vorlesung und in der Befehlsliste der „Arbeitsblätter zur Übung“ (Befehle CALL und RET) informieren.

Grundaufgabe b)

Ordnen Sie einigen Tasten der Matrixtastatur die Ziffern 0 bis 9 zu. Notieren Sie diese Anordnung. Realisieren Sie ein Programm, welches die jeweils gedrückte Ziffer in lesbarer Darstellung auf der linken Stelle der Sieben-Segment-Anzeigen anzeigt.

> Bemerkung:
> Denken Sie daran, dass die zuvor erstellten Unterprogramme bestimmte Register verändern können. Es zahlt sich jetzt aus, wenn Sie vorher ein wenig Mühe für Planung und Dokumentation aufgewendet haben. Um die Übersichtlichkeit zu erhalten, sollten Sie Ihr Programm mittels weiterer Unterprogramme strukturieren.

Fortgeschrittene Aufgabe c)

Erweitern Sie das Programm jetzt so, dass die gedrückten Ziffern der Reihe nach nebeneinander angezeigt werden. Nach Erreichen der letzten Stelle soll der Vorgang wieder links beginnen. Dabei können Sie evtl. die Anzeige löschen, das ist aber nicht notwendig. Das Drücken einer nicht als Ziffer definierten Taste soll eine leere Stelle erzeugen.

Fortgeschrittene Aufgabe d)

Vielleicht ist Ihnen aufgefallen, dass bei einem einzigen Tastendruck manchmal mehr als eine Ziffer erscheint. Ursache ist das so genannte „Prellen“ der Tasten. Die mechanisch bewegten Kontakte haben im Moment des Schließens und Öffnens kurzzeitig eine unsichere Kontaktgabe, so dass der logische Pegel innerhalb kurzer Zeit mehrfach wechseln kann. Die Software deutet das evtl. als mehrere Tastendrücke.

Diese Erscheinung sollen Sie jetzt beseitigen. Eine Lösungsmöglichkeit besteht darin, in das Unterprogramm wmatr an zwei Stellen eine Wartezeit von jeweils etwa 0,1 s einzufügen:

- Direkt am Beginn des Unterprogramms (vor der Abfrage, ob bereits eine Taste gedrückt ist).
- Nach dem Erkennen des Loslassens einer evtl. bereits gedrückt gewesenen Taste (vor der nächsten Abfrage).

Bei richtiger Realisierung findet der Prellvorgang stets innerhalb der Verzögerungszeit statt und bleibt wirkungslos.

> Bemerkung:
> Den genauen Zahlenwert der Verzögerung müssen Sie ausprobieren. Bei zu kurzer Zeit bleibt das Prellen bestehen, bei zu langer Zeit reagieren die Tasten spürbar träge.

Je nach der genauen Gestaltung der Software kann auch ein anderer Aufbau der Entprellung zweckmäßig sein. Testen Sie die Wirksamkeit Ihrer Lösung und führen Sie das Ergebnis dem Praktikumsbetreuer vor.

Als **Zusatzaufgabe** können Sie einen sehr einfachen „Taschenrechner“ realisieren. Dieser soll z.B. einstellige nichtnegative Dezimalzahlen addieren und das ein- bis zweistellige Ergebnis auf der Sieben-Segment-Anzeige anzeigen. Definieren Sie zusätzliche Tasten für ‚+’ und ‚=’. Das Rechnen kann dann mit Tastenfolgen der folgenden Art erfolgen:

Ziffer + Ziffer =

Nach Drücken des Gleichheitszeichens sollte auf der Sieben-Segment-Anzeige das Ergebnis der Addition als Dezimalzahl erscheinen. Danach kann die nächste Rechnung beginnen.

Mögliche Erweiterungen sind das Rechnen mit mehrstelligen Zahlen, weitere Rechenoperationen und dergleichen mehr.

Alternativen zum Taschenrechner sind z.B. eine Digitaluhr oder andere Ideen.

4. Bedienungsanleitung

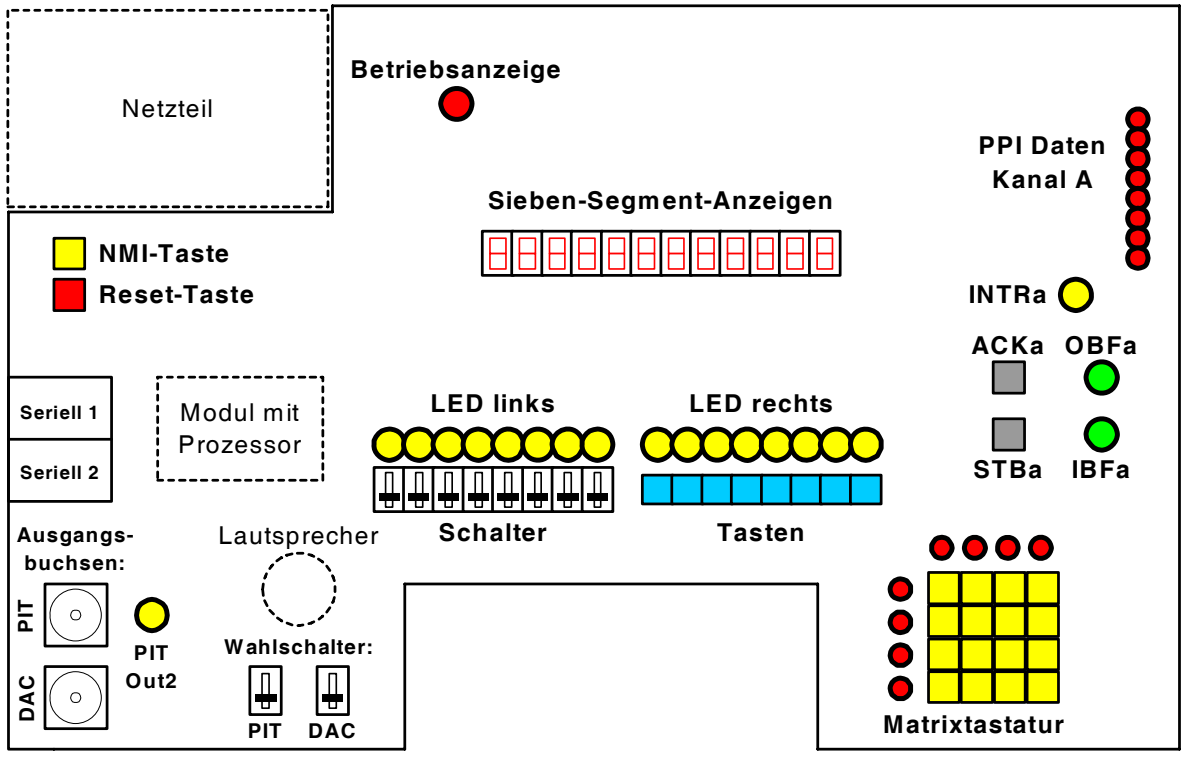
4.1 Bedienung des Zielrechners

Der Zielrechner ist ein vollständiges Rechnersystem mit einem x86-kompatiblen Prozessor, RAM- und ROM-Speicher und zahlreichen Ein- und Ausgabebaugruppen. Für die Bedienung im Rahmen dieses Praktikums sind vor allem die Ein- und Ausgabebaugruppen wichtig. Einen Überblick haben Sie bereits in Abschnitt 2.2 erhalten.

Jetzt erfahren Sie, wie die Bedienelemente funktionieren und wie die Ein- und Ausgabebaugruppen durch die Software angesprochen werden. Für die Portzugriffe werden ausschließlich Ein- und Ausgabezyklen mit **8 bit Datenbreite** verwendet.

4.1.1 Anordnung der Bedienelemente

Das folgende Bild zeigt die Anordnung der Bedien- und Anschlusselemente auf der Frontplatte des Gerätes. Anschließend werden diese Elemente und die dazugehörenden Baugruppen näher beschrieben. Bei besonderem Interesse finden Sie auf der Webseite (siehe 1.3) die Schaltbilder der Grundleiterplatte.



4.1.2 Stromversorgung und Rücksetzen

Das Gerät verfügt über ein internes Netzteil und wird an das 230-V-Netz angeschlossen. Die Netzspannung wird außerhalb des Geräts ein- und ausgeschaltet, so dass kein Netzschalter existiert. Die **Betriebsanzeige** meldet das Vorhandensein der Betriebsspannung.

Die **Reset-Taste** löst einen Hardware-Reset aus. Der anschließende Initialisierungsvorgang dauert einige Sekunden. Er ist abgeschlossen, wenn alle Anzeigen außer der Betriebsanzeige verlöschen. Erst dann ist das Gerät betriebsbereit.

Die Reset-Funktion aktivieren Sie zu Beginn des Praktikums sowie bei Totalabstürzen. Während des Debuggens sollte sie nur benutzt werden, wenn die Stopp-Funktion des Debuggers nicht mehr reagiert. Anschließend ist mindestens 'Recapture Target' und 'Reload' erforderlich. Es ist aber sicherer, den Debugger zu beenden und neu zu starten.

Bitte beachten Sie, dass einige Ein- und Ausgabebaugruppen nicht durch das Reset-Signal erfasst werden.

Die **NMI-Taste** soll einen „Nichtmaskierbaren Interrupt" auslösen. Diese Funktion ist für künftige Erweiterungen vorgesehen und derzeit ohne Bedeutung.

4.1.3 Schalterreihe und Tastenreihe

Das Gerät besitzt zwei digitale parallele Eingabebaugruppen mit je 8 bit Breite. Diese sind mit einer Reihe aus acht **Schaltern** und einer Reihe aus acht **Tasten** verbunden. Die Portadressen sind 58H für die Schalter und 59H für die Tasten.

Die Schalter und Tasten sind je einem Bit zugeordnet. Die Anordnung entspricht der Wertigkeit im Byte (höchstwertiges Bit links, niedrigstwertiges Bit rechts). Die Schalter ergeben in der oberen Stellung Eins-Pegel und in der unteren Stellung Null-Pegel. Die Tasten ergeben gedrückt Eins-Pegel und losgelassen Null-Pegel.

4.1.4 LED-Reihen

Die LED-Reihen (LED = Lichtemittierende Diode) **LED links** und **LED rechts** sind mit zwei digitalen parallelen Ausgabebaugruppen von je 8 bit Breite verbunden. Die Portadressen sind 5CH für die linke Reihe und 5DH für die rechte Reihe.

Die einzelnen LEDs sind je einem Bit zugeordnet. Die Anordnung entspricht der Wertigkeit im Byte (höchstwertiges Bit links, niedrigstwertiges Bit rechts). Eins-Pegel ergibt eine leuchtende Stelle, Null-Pegel eine dunkle Stelle.

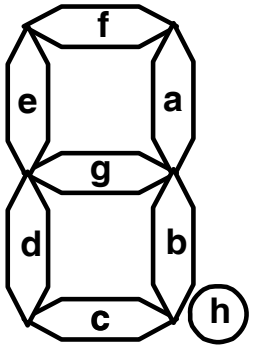
Die LED-Reihen sind auf der Frontplatte den Schaltern und Tasten zugeordnet, es besteht jedoch kein elektrischer oder logischer Zusammenhang.

4.1.5 Sieben-Segment-Anzeigen

Die **Sieben-Segment-Anzeigen** ermöglichen die Darstellung von Dezimalziffern und weiteren Zeichen auf maximal zwölf Stellen. Jede einzelne Anzeige ist mit einer digitalen parallelen Ausgabebaugruppe von 8 bit Breite verbunden und damit einzeln ansteuerbar. Die Portadressen sind B0H (rechte Stelle) bis BBH (linke Stelle).

Die einzelnen Segmente sind je einem Bit zugeordnet. Eins-Pegel ergibt ein leuchtendes Segment, Null-Pegel ein dunkles Segment. Die Zuordnung der Bitpositionen zu den Segmenten ergibt sich aus der folgenden Abbildung.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
h	g	f	e	d	c	b	a



4.1.6 Matrixtastatur

Die **Matrixtastatur** am vorliegenden Gerät ist ein Tastenfeld aus 16 Tasten, welches in Form einer 4x4-Matrix organisiert ist. Dieses weit verbreitete Prinzip ermöglicht insbesondere bei größeren Tastenfeldern eine Verringerung des Hardware-Aufwands.

Die Tasten befinden sich an den Kreuzungspunkten von 4 Zeilenleitungen und 4 Spaltenleitungen. Die Zeilenleitungen sind mit 4 Bits einer digitalen parallelen Ausgabebaugruppe (Adresse 5AH) verbunden. Die Spaltenleitungen sind mit 4 Bits einer digitalen parallelen Eingabebaugruppe (Adresse 5BH) verbunden. Die Funktion beruht darauf, zunächst am Ausgabeport ein Abfragemuster auszugeben und dann am Eingabeport das Resultat zu lesen.

Ohne gedrückte Tasten haben alle Spaltenleitungen Null-Pegel. Beim Drücken einer Taste wird die betreffende Spaltenleitung dann und nur dann auf Eins-Pegel gesetzt, wenn die betreffende Zeilenleitung im aktuellen Abfragemuster Eins-Pegel aufweist. Für mehrere gedrückte Tasten pro Spalte gilt die ODER-Verknüpfung, was freilich nur bei Abfragemustern mit mehreren Eins-Pegel führenden Zeilenleitungen von Interesse ist.

Der normale Ablauf einer Tastenabfrage benutzt vier Abfragemuster, bei denen jeweils nur eine einzige Zeilenleitung Eins-Pegel aufweist. Aus den Resultaten, die jeweils am Eingabeport gelesen werden, können dann die gedrückten Tasten bestimmt werden.

Die Zuordnung der Zeilen und Spalten zu den einzelnen Bitpositionen ist in der folgenden Tabelle dargestellt. Zur Demonstration der Funktion und als Hilfe beim Debuggen werden die Pegel der Zeilen- und Spaltenleitungen durch LEDs angezeigt (high-aktiv). Bei besonderem Interesse können Sie auf der Webseite die vollständige Schaltung finden (Schaltbilder des Zielrechners, Seite 9).

		Spaltenleitungen Eingabeport 5BH			
		2^0	2^1	2^2	2^3
Zeilenleitungen Ausgabeport 5AH	2^0				
	2^1				
	2^2				
	2^3				

4.1.7 Programmierbarer Intervalltimer (PIT)

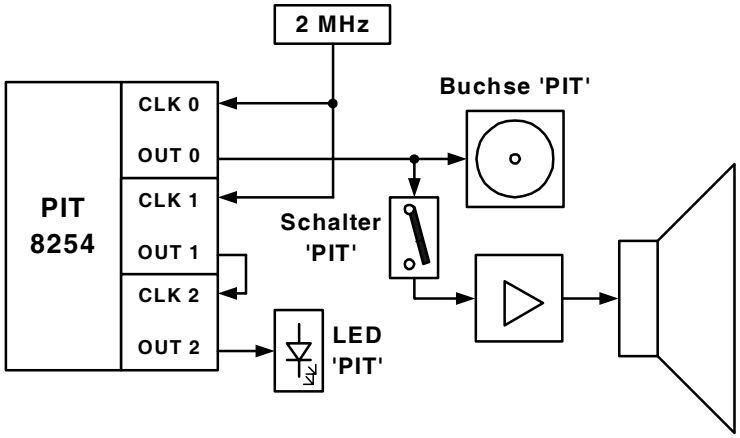
Das Gerät besitzt eine Zähler- und Zeitgeberbaugruppe mit einem **PIT** vom Typ 8254. Dieser Typ ist eine aufwärtskompatible Weiterentwicklung des 8253, der in den „Arbeitsblättern zur Übung“ enthalten ist. Das vollständige Datenblatt des 8254 finden Sie bei Interesse auf der unter 1.3 angegebenen Webseite. Der PIT belegt die Ein- und Ausgabeadressen 54H bis 57H.

Die Takteingänge der Kanäle 0 und 1 werden intern mit einer quarzstabilisierten Frequenz von 2 MHz versorgt. Der Ausgang von Kanal 0 ist an der **Ausgangsbuchse PIT** angeschlos-

sen. Falls eine geeignete Frequenz erzeugt wird, kann der Ton über den eingebauten Lautsprecher hörbar gemacht werden. Dazu ist der **Wahlschalter PIT** einzuschalten (obere Schalterstellung entspricht „Ein“).

Die Kanäle 1 und 2 sind „kaskadiert“, das heißt dass der Takteingang von Kanal 2 mit dem Ausgang von Kanal 1 verbunden ist. Am Ausgang von Kanal 2 befindet sich die LED **PIT Out 2**. Die Gate-Eingänge sämtlicher Kanäle sind fest mit Eins-Pegel belegt.

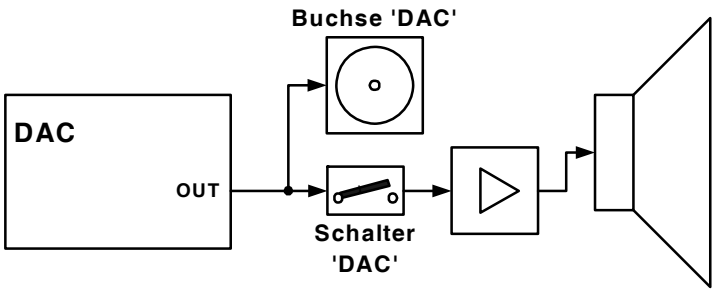
Der im Bild dargestellte vereinfachte Schaltungsauszug verdeutlicht die Zusammenhänge.



4.1.8 Analog-Ausgang

Der Analogausgang besitzt einen **DAC** (Digital-Analog-Converter) mit einer Datenbreite von 8 bit. Die Ansteuerung erfolgt durch Ausgabe eines Bytes auf die Portadresse 5EH. Die Analogspannung entspricht dem vorzeichenlosen Zahlenwert des Bytes multipliziert mit 10 mV. Der Ausgangsspannungsbereich beträgt damit 0,00 V (Byte = 00H) bis 2,55 V (Byte = FFH).

Der Ausgang ist an der **Ausgangsbuchse DAC** angeschlossen (siehe Bild). Falls eine Wechselspannung mit einer geeigneten Frequenz erzeugt wird, kann dieser Ton über den eingebauten Lautsprecher hörbar gemacht werden. Dazu ist der **Wahlschalter DAC** einzuschalten (obere Schalterstellung entspricht „Ein“).



4.1.9 Programmierbares Parallelinterface (PPI)

Das Gerät besitzt ein Programmierbares Parallelinterface (programmierbare digitale Ein- und Ausgabe) mit einem **PPI** vom Typ 8255A. Dieser Typ ist in den „Arbeitsblättern zur Übung“ enthalten. Das vollständige Datenblatt finden Sie bei Interesse auf der Webseite (siehe 1.3). Das PPI belegt die Ein- und Ausgabeadressen 50H bis 53H.

Die Baugruppe ist zur manuellen Untersuchung von Ein- und Ausgabeprotokollen mit Zwei-Draht-Handshake gedacht. Zu diesem Zweck sind die Handshake-Steuersignale von Port A an einzelnen Tasten und LEDs angeschlossen. Die Eingangssignale **ACKa** (für Ausgabe) und **STBa** (für Eingabe) werden mit Tasten bedient. Die Ausgangssignale **OBFa** (für Ausgabe) und **IBFa** (für Eingabe) sowie das zur Interruptauslösung gedachte Signal **INTRa** werden mit LEDs angezeigt. Alle Signale werden als high-aktiv behandelt, unabhängig von der elektrischen Realisierung.

Die acht Datenleitungen von Port A sind an die LED-Reihe **PPI Daten** angeschlossen. Das höchstwertige Bit befindet sich oben. Diese Anzeige macht die ausgegebenen Daten sichtbar, wenn Port A in einer Ausgabebetriebsart betrieben wird.

Bitte beachten Sie, dass sämtliche Tasten und Anzeigen nur dann einen Sinn ergeben, wenn Port A in geeigneter Weise initialisiert ist. Sämtliche Eingangssignale mit Ausnahme von ACKa und STBa sind unbestimmt. Die Zuordnung der Signalnamen können Sie den „Arbeitsblättern zur Übung“ entnehmen.

4.1.10 Serielle Anschlüsse

Das Gerät besitzt zwei duplexfähige Baugruppen für asynchrone serielle Kommunikation entsprechend dem RS232-Standard. Diese sind über die Anschlussbuchsen **Seriell 1** und **Seriell 2** zugänglich.

Am Anschluss Seriell 1 wird stets der Hostrechner angeschlossen. Dieser Anschluss darf in keiner Weise gestört oder manipuliert werden. Anschluss Seriell 2 wird derzeit nicht benutzt. Die Portadressen und Registerstrukturen der seriellen Baugruppen ähneln den „COM-Schnittstellen“ eines herkömmlichen PC und werden hier nicht weiter beschrieben.

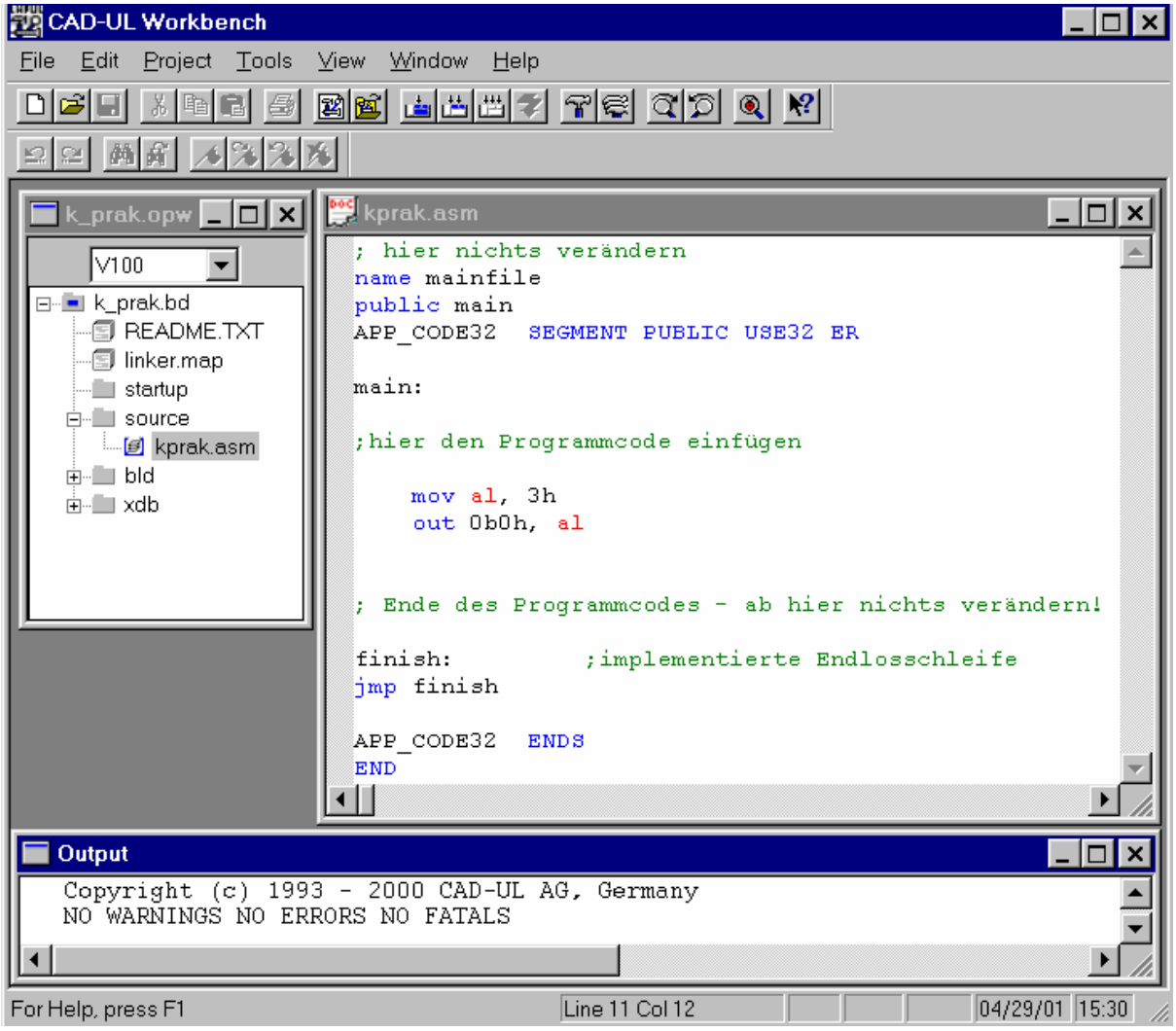
4.2 Bedienung der Entwicklungsumgebung

Im Abschnitt 2.3 haben Sie bereits den Ablauf der Softwareentwicklung in diesem Praktikum kennen gelernt. Hier finden Sie detaillierte Informationen zur Bedienung der Entwicklungsumgebung und zur Struktur des Quelltextes. Bitte beachten Sie, dass die Menüs, Dialoge und Fehlermeldungen ausschließlich in englischer Sprache vorliegen.

4.2.1 Workbench

Für die Softwareentwicklung wird die professionelle Entwicklungsumgebung CAD-UL Workbench der CAD-UL AG benutzt. Der Start erfolgt über 'Workbench' auf dem Desktop oder im Startmenü-Eintrag '*CAD-UL Embedded Tools - Workbench*'.

Die Workbench enthält unter anderem Funktionen zum Editieren und Übersetzen von Programmen für x86-Systeme in den Sprachen Assembler, C und C++ sowie Funktionen zur Pro-



jektverwaltung und zum Ankoppeln weiterer Werkzeuge. Sämtliche Bedienvorgänge und Anzeigen erfolgen innerhalb des im Bild gezeigten Hauptfensters. Die folgende Beschreibung beschränkt sich auf Funktionen, die im vorliegenden Praktikum eine Rolle spielen können.

Aktivieren Sie bitte keine unbekannten Funktionen und ändern Sie keine Einstellungen und Optionen!

Im Bild sehen Sie drei verschiedene Arten von Unterfenstern. Das **Projektfenster** (im Bild links oben) zeigt die Struktur des geöffneten Projekts in Form einer aufklappbaren Baumansicht. Zum Praktikumsbeginn sollte das Projekt k_prak aktiv sein. Ist das nicht der Fall, so benutzen Sie die Funktion 'Open Project', um das Projektfile C:\Kprak\k_prak.opw zu öffnen.

Durch Aufklappen des Baumes gelangen Sie zum Ordner source, in welchem sich die Datei kprak.asm befindet. Ein Doppelklick auf den Dateinamen öffnet ein **Dokumentenfenster** mit dem Dateiinhalt (im Bild rechts). Die im Hauptzweig des Projekts sichtbare Datei README.TXT enthält möglicherweise nachträgliche Ergänzungen zu dieser Anleitung und wird ebenfalls durch Doppelklick geöffnet. Weitere eventuell enthaltene Dateien bleiben hier unerwähnt und dürfen nicht verändert werden.

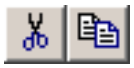
> Bemerkung:
> Wahrscheinlich sind die erwähnten Dokumentenfenster beim Start bereits sichtbar.
>
> Hinweis für Interessierte:
> Falls Sie mehrere Versionen Ihres Programms verwalten wollen, können Sie mit der Menüfunktion '*Project / Create New Version*' eine neue Version definieren. Die neue Version entsteht durch Kopieren einer schon vorhandenen Version und erhält einen eigenen Namen. Das Umschalten zwischen den Versionen geschieht dann im Kopf des Projektfensters. Die erste Version heißt immer V100.
>
> Bitte speichern Sie **nicht** einfach die Quelltextdatei unter einem neuen Namen, da in diesem Fall trotzdem die alte Datei übersetzt wird und die neu gespeicherte Version unberücksichtigt bleibt.

Im **Outputfenster** (im Bild unten) erscheinen während des Übersetzens die Meldungen der Übersetzungswerkzeuge. Dazu gehören auch Fehlermeldungen, welche Sie unbedingt beachten müssen. Sie müssen das Fenster vergrößern und/oder zurück scrollen, um alle Meldungen zu erfassen. Genauere Erläuterungen zu den Meldungen erhalten Sie weiter unten.

Die Abfolge der Arbeitsschritte bei der Programmentwicklung ist im Abschnitt 2.3 dargestellt. Die dazu verwendeten Funktionen der Workbench sind über Buttons in der **Werkzeugleiste** (siehe folgendes Bild) zugänglich. Aufgrund der Konfigurierbarkeit können geringe Unterschiede in der Anordnung auftreten. Falls die Symbole völlig anders aussehen, so han-

delt es sich vermutlich nicht um das Fenster CAD-UL Workbench (Fenstertitel kontrollieren).

Die folgende Tabelle beschreibt eine Auswahl der Funktionen. Einige nicht benötigte Funktionen wurden weggelassen. Die fett gedruckten Namen entsprechen jeweils der Bezeichnung der Funktion und werden auch in dieser Anleitung benutzt.

Symbol	Beschreibung
	**Open:** Öffnen einer nicht zum Projekt gehörenden Datei in einem Dokumentenfenster.
	**Save:** Speichern der Datei im gerade aktiven Dokumentenfenster.
	**Cut / Copy:** Ausschneiden / Kopieren des markierten Textbereiches in die Zwischenablage.
	**Paste:** Einfügen eines Textbereiches aus der Zwischenablage.
	**Open Project:** Öffnen eines ganzen Projekts.
	**Rebuild all:** Starten des kompletten Übersetzungsvorganges (im kleinen Fenster bestätigen).
	**Jump next / prev error:** Springen zur nächsten / vorigen fehlerhaften Quelltextzeile (falls das Output-fenster Fehlerinformationen enthält).
	**Debugger:** Starten des Debuggers XDB. Es erscheint ein kleines Fenster, in welchem der Start zu bestätigen ist. Der Debugger läuft dann als eigene Applikation.
	**Help:** Aktivieren der kontextbezogenen Hilfefunktion.
	**Undo / Redo:** Stornieren / Wiederherstellen der letzten Aktion beim Editieren.
	**Find / Find Next:** Suchen von Zeichenfolgen mit / ohne Abfrage des Suchmusters.
	**Toggle / Next / Previous / Clear Bookmark:** Setzen / Suchen vorwärts / Suchen rückwärts / Gesamtlöschen von „Lesezeichen“ (Navigationsmarken im Dokumentenfenster)

4.2.2 Quelltextstruktur und Speicheraufteilung

Im Originalzustand des Projekts enthält die Quelltextdatei kprak.asm einen Rahmenquelltext, welcher die benötigten Einstellungen und Deklarationen enthält. Der Rahmenquelltext hat folgendes Aussehen:

```
; ===================================================================
;   Dateiname kprak.asm
;   Quelltextvorlage fuer das Praktikum x86-Assemblerprogrammierung
;
;   Letzte Aenderung: Maerz 2003
;   TU Ilmenau, FG Rechnerarchitekturen
; ===================================================================

; Veraenderungen nur an den gekennzeichneten Stellen.
; Bitte keine Umlaute benutzen!!


name   mainfile
public main

APP_CODE32       SEGMENT PUBLIC USE32 ER
ASSUME DS:APP_DATA32,ES:APP_DATA32,FS:APP_DATA32,GS:APP_DATA32,SS:APP_DATA32

main:

; *** Ab hier den Programmbereich einfuegen ***



; *** Ende des Programmbereiches - ab hier nichts veraendern! ***

finish:          jmp finish          ; vorbereitete Endlosschleife
APP_CODE32       ENDS

APP_DATA32       SEGMENT PUBLIC USE32 RW

; *** Ab hier den Datenbereich einfuegen ***



; *** Ende des Datenbereichs - ab hier nichts veraendern! ***

                 db 0                ; Datenbereich soll nicht leer sein
APP_DATA32       ENDS

END              main
```

Aufgrund von Weiterentwicklungen können Unterschiede auftreten. Falls zum Praktikumsbeginn kein derartiger Quelltext existiert, konsultieren Sie bitte die schon erwähnte Datei README.TXT oder den Praktikumsbetreuer.

Das Einfügen Ihres Programms soll **nur** in den beiden durch Kommentare eingegrenzten Bereichen erfolgen! Änderungen außerhalb dieser Bereiche führen zu Problemen beim Überset-

zen und Debuggen. Die genaue Bedeutung der bereits vorbereiteten Teile des Quelltextes wird hier nicht behandelt.

Im **Programmbereich** fügen Sie den Teil Ihres Quelltextes ein, der die Assemblerbefehle Ihres Programms enthält. Es handelt sich also um Assemblerzeilen mit der folgenden prinzipiellen Struktur:

```
symbol:     befehl     operand,operand     ; kommentar
```

Im **Datenbereich** fügen Sie den Teil Ihres Quelltextes ein, der die Speicherzellen für Ihr Programm definiert. Es handelt sich also um Assemblerzeilen mit der folgenden prinzipellen Struktur:

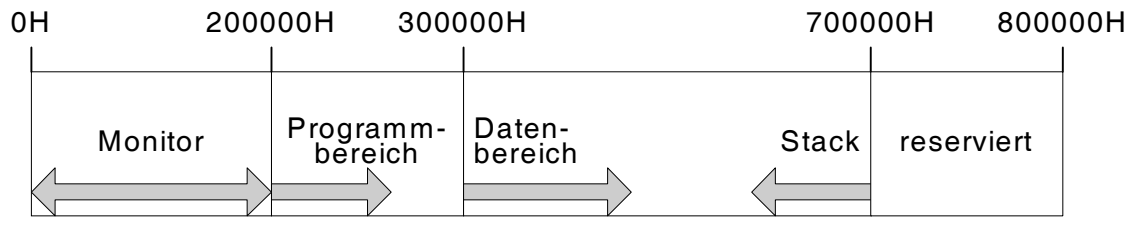
```
symbol      Dx     anfangsbelegung(en)     ; kommentar
```

Für den „Pseudobefehl“ Dx kommen die Speicherzellen-Definitionsbefehle DB, DW und DD in Frage (siehe „Arbeitsblätter zur Übung“). Nicht definierte Anfangsbelegungen werden durch Fragezeichen dargestellt.

Wenn außer dem Hauptprogramm ein oder mehrere Unterprogramme existieren, so müssen auch diese im Programmbereich stehen. Da die Ausführung immer beim ersten Befehl des Programmbereichs startet, sollte das Hauptprogramm jeweils am Anfang stehen. Beachten Sie aber, dass die Ausführung am Ende des Hauptprogramms nicht mit dem dahinter stehenden Unterprogramm fortgesetzt werden darf. Zweckmäßig ist deshalb die Gestaltung des Hauptprogramms als Endlosschleife.

> Bemerkung für besonders Interessierte:
> Der Quelltextrahmen definiert ein Codesegment und ein Datensegment. Das Datensegment wird für sämtliche Segmentregister außer CS benutzt. Die Laufzeitumgebung setzt die Basisadressen beider Segmente auf Null, so dass die Segmentregister nicht weiter beachtet werden müssen. In dieser Situation können die Offsets als alleinige Speicheradressen betrachtet werden, was dem Konzept des „flachen Speichers“ entspricht.

Das folgende Bild zeigt zur Information die **Speicheraufteilung**, welche das erstellte Programm zur Laufzeit vorfindet. Sämtliche Adresseinstellungen werden automatisch beim Übersetzen und/oder beim Laden im Debugger vorgenommen, so dass Sie sich nicht selbst dar-



um kümmern müssen.

Die unteren 2 MiByte stehen dem so genannten **Monitor** zur Verfügung, welcher die Kommunikations- und Debugfunktionen realisiert. Diesen Speicherbereich soll das zu testende Programm unberührt lassen.

Für den **Programmbereich** des zu testenden Programms ist 1 MiByte vorgesehen. Die Start-Adresse (Anfangsbelegung des Befehlszeigers EIP) ist stets 200000H. Der **Datenbereich** und der **Stack** (Stapelspeicher) haben zusammen 4 MiByte zur Verfügung. Die im Datenbereich definierten Speicherzellen werden aufeinander folgend ab 300000H angeordnet.

Der Stackpointer ESP wird automatisch auf den Wert 6FFFFCH voreingestellt, so dass sich der Stack unterhalb von 700000H in absteigender Richtung ausbreitet. Der Stack muss auch dann funktionsfähig sein, wenn das Programm keine Stackbefehle benutzt! **Das Register ESP darf deshalb nicht anderweitig benutzt oder in sonstiger Weise manipuliert werden!**

Normalerweise sollte der Platz für sämtliche erwähnten Speicherbereiche völlig ausreichend bemessen sein. Manche Programmfehler führen jedoch zur unkontrollierten Ausdehnung insbesondere des Stacks. Dabei werden möglicherweise andere Speicherbereiche überschrieben, was meistens zum Totalabsturz führt. Überprüfen Sie deshalb bei „unerklärlichen“ Abstürzen auch die Balance der Stack- und Unterprogrammbefehle.

4.2.3 Meldungen beim Übersetzen

Während des Übersetzungsvorganges werden durch die Übersetzungswerkzeuge Meldungen erzeugt, die im Output-Fenster gesammelt werden. Diese Meldungen müssen Sie anschließend unbedingt inspizieren, da Fehlermeldungen enthalten sein können. Falls beim Übersetzen Fehler gemeldet wurden, ist in der Regel kein ausführbares Programm erzeugt worden. Der Debugger würde jetzt ungefragt eine ältere Variante laden, was sehr irritierend sein kann. Bei Fehlermeldungen müssen Sie vor dem Debuggen den Quelltext korrigieren und erneut übersetzen, bis die Fehler beseitigt sind.

Bei den Meldungen werden folgende Kategorien unterschieden:

- **Fatals:** Besonders schwere Fehler, die unbedingt korrigiert werden müssen.
- **Errors:** Sonstige Fehler, die unbedingt korrigiert werden müssen.

- **Warnings:** Warnungen, die auf Auffälligkeiten hinweisen. Eine Korrektur ist nicht unbedingt notwendig. Warnungen enthalten aber manchmal Hinweise auf versteckte Fehler.
- **Notes:** Sonstige Bemerkungen, die meistens keinen Eingriff erfordern.

An dieser Stelle können keinesfalls alle denkbaren Fehlermeldungen behandelt werden. Die Darstellung erfolgt deshalb am Beispiel. Der folgende Textkasten zeigt zunächst ein Beispiel für die Meldungen im Output-Fenster für den Fall einer **fehlerfreien Übersetzung:**

```
---------------------- Build k_prak.bd -----------------

---- Processing kprak.obj ----
AS386 - CAD-UL Cross Assembler 80386/80486 - Version: V401C

Copyright (c) 1990 - 1999 CAD-UL AG, Germany

0 WARNINGS  0 ERRORS  0 FATALS  IN C:\Kprak\V100\k_prak.bd\source\kprak.asm


---- Processing k_prak.omf ----
LINK386 - Optimizing Cross Linker and System Builder - Version: V237

Copyright (c) 1992 - 2000 CAD-UL AG, Germany


LINK386-W-WARNING541:DS register not initialized by module MAINFILE

LINK386-W-WARNING541:SS register not initialized by module MAINFILE


linkage completed with 0 errors, 2 warnings and 0 notes


---- Processing k_prak.bd ----
OMF386BND - CAD-UL Debug Preprocessor for x86 - Version: V324C

Copyright (c) 1993 - 2000 CAD-UL AG, Germany

NO WARNINGS NO ERRORS NO FATALS


-- Build completed successfully --
```

Es sind nacheinander die Meldungen der drei Übersetzungswerkzeuge entsprechend Abschnitt 2.3 erkennbar:

- Assembler AS386: Erfolgreicher Ablauf ohne Meldungen.

- Linker LINK386: Erfolgreicher Ablauf mit zwei Warnungen. Die hier gezeigten Warnungen sind unkritisch, da die vermissten Initialisierungen in der Laufzeitumgebung erfolgen werden.
- Konvertierer (Debug-Präprozessor) OMF386BND: Erfolgreicher Ablauf ohne Meldungen.

Die Übersetzung ist dann erfolgreich, wenn alle drei Werkzeuge gearbeitet haben, die Anzahl sämtlicher Fatals und Errors Null ist und die „Erfolgsmeldung“ in der letzten Zeile erscheint.

Jetzt folgt ein Beispiel für eine **Übersetzung mit Fehlern:**

```
---------------------- Build k_prak.bd -----------------

---- Processing kprak.obj ----
AS386 - CAD-UL Cross Assembler 80386/80486 - Version: V401C

Copyright (c) 1990 - 1999 CAD-UL AG, Germany

AS386-E-ERROR075:C:\...\kprak.asm:26:illegal operator combination

AS386-E-ERROR130:C:\...\kprak.asm:26:illegal register combination

AS386-E-ERROR144:C:\...\kprak.asm:26:illegal address size combination

AS386-E-ERROR072:C:\...\kprak.asm:30:expression contains an undefined label (TB)

AS386-E-ERROR072:C:\...\kprak.asm:34:expression contains an undefined label (AFH)

AS386-W-WARNING614:C:\...\kprak.asm:90:only 7bit ascii are accepted, value: 0374

1 WARNINGS  5 ERRORS  0 FATALS  IN C:\Kprak\V100\k_prak.bd\source\kprak.asm


undefined labels

  line     name

    30     TB

    34     AFH

-- Build completed with errors -
```

Bemerkung:
Die Pfadangaben wurden aus Platzgründen manuell gekürzt.

Hier hat der Assembler fünf Fehlermeldungen und eine Warnung generiert. Daraufhin wurde der Übersetzungsvorgang abgebrochen, die anderen Werkzeuge haben also nicht gearbeitet.

Die Fehlermeldungen sind jeweils mit der entsprechenden **Zeilennummer** des Quelltextes versehen. (Diese erscheint beim Editieren auch in der Fußzeile des Hauptfensters der Workbench). Mit 'Jump next/prev error' kann man direkt zu den fehlerhaften Zeilen springen.

Zur Erläuterung der gemeldeten Fehler werden jetzt zusätzlich die Quellzeilen angegeben, die den jeweiligen Fehler verursacht haben:

- Zeile 26: **`MOV  ECX,BL`**

  Dieser MOV-Befehl enthält eine unzulässige Operandenkombination, nämlich zwei Register unterschiedlicher Breite. Derselbe Fehler hat hier drei verschiedene Meldungen verursacht.

- Zeile 30: **`CALL tb`**

  Das Symbol TB, welches als Ziel dieses Unterprogrammaufrufs dient, ist offensichtlich nicht definiert. Entweder ist das Symbol falsch geschrieben oder das Unterprogramm fehlt. Die Groß- und Kleinschreibung spielt dabei übrigens keine Rolle.

- Zeile 34: **`CMP  DL,AFH`**

  Hier konnte das Symbol AFH nicht zugeordnet werden. Gemeint war jedoch ein hexadezimaler Zahlenwert. Damit dieser erkannt wird, ist die Schreibweise 0AFH nötig. Man sieht, dass unterschiedliche Fehler zu den gleichen Meldungen führen können.

- Zeile 90: **`; für Zifferneingabe`**

  Diese Warnung wurde durch die unzulässige Verwendung eines Umlauts verursacht. Das ist im Kommentar zwar kein ernsthaftes Problem, man sollte es aber trotzdem korrigieren.

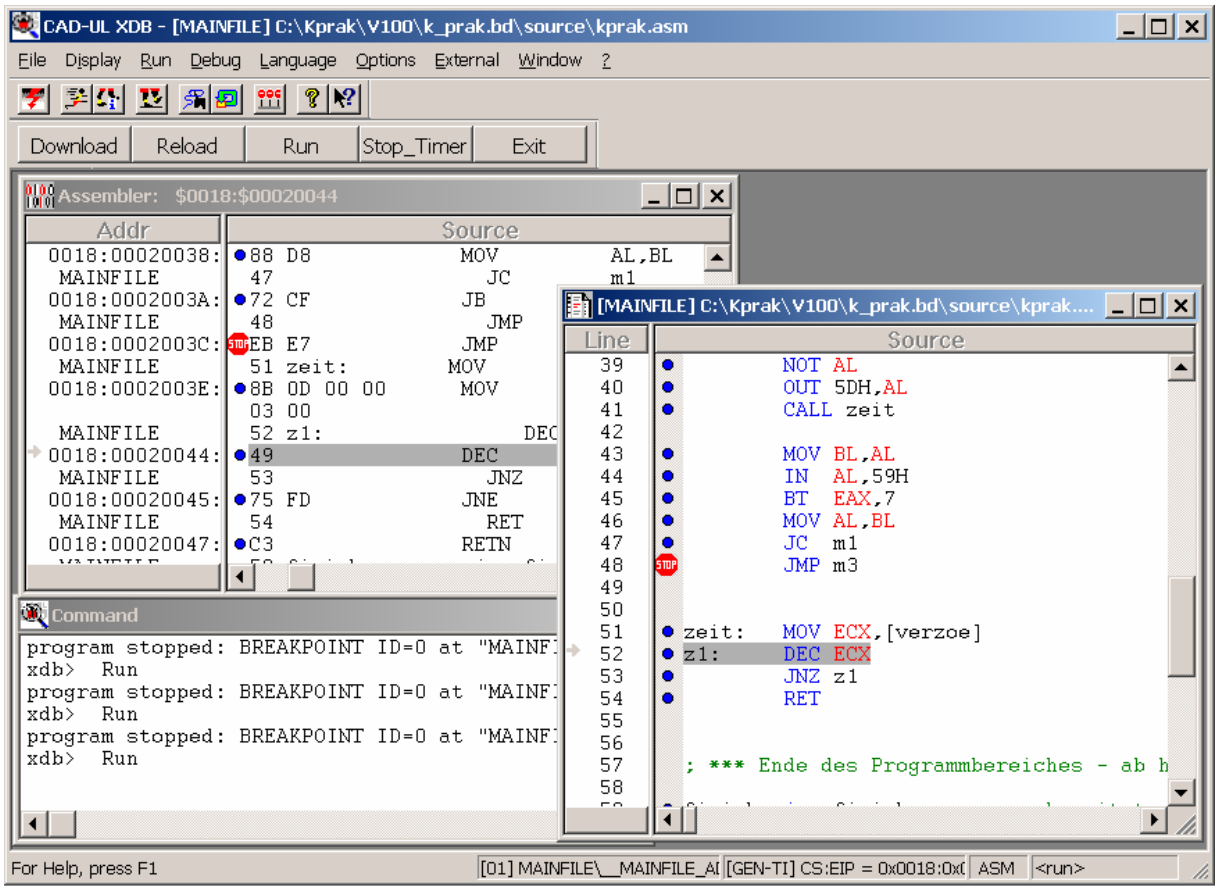
Die „Treffsicherheit" der Fehlertexte lässt manchmal zu wünschen übrig. Sie müssen dann detektivisch arbeiten und den Hintergrund des Problems ergründen. Bei großer Zahl an Meldungen gehen Sie schrittweise vor und korrigieren zunächst nur einen Teil der Fehler. Häufig sind verschiedene Meldungen voneinander abhängig und verschwinden gleichzeitig.

Probleme beim Linken und Konvertieren beruhen meistens auf Beschädigungen der vorgegebenen Quelltextstruktur oder anderer Projektdateien. Vielleicht haben Sie aber auch überflüssige Deklarationsanweisungen oder dergleichen benutzt.

4.3 Bedienung des Debuggers

4.3.1 Aufruf und Erscheinungsbild

Der Remote-Debugger CAD-UL XDB wird aus der Workbench heraus mit dem Button ‘Debugger’ gestartet. Im erscheinenden kleinen Fenster ist der Start zu bestätigen, ohne irgendwelche Änderungen vorzunehmen. Anschließend erscheint der Debugger als selbstständige Applikation in einem eigenen Hauptfenster mit mehreren Unterfenstern. Das Bild zeigt ein Beispiel für das Aussehen des Debugger-Hauptfensters.



Das als erstes erscheinende **Commandfenster** (im Bild links unten, Fensterüberschrift: Command) listet in Art einer Konsole fortlaufend die Aktionen des Debuggers auf. Sie brauchen sich hier nicht für alle Details zu interessieren. Es sind aber Informationen über den aktuellen Status des Debuggers zu entnehmen. Wenn am Beginn der letzten Zeile der Prompt **xdb>** und dahinter nur der Kursor steht, befindet sich der Debugger im Grundzustand und kann bedient werden. Anderenfalls läuft gerade eine Aktivität, deren Ende abgewartet werden muss. Ein Fehlercode in der Art E-2267 mit anschließendem Fehlertext weist auf einen Fehlerzustand hin.

Hinweis:
Das rechte Feld in der Fußleiste des Debugger-Hauptfensters enthält ebenfalls eine Information über gerade laufende Aktivitäten. Im oben stehenden Bild ist das <run>. Im Grundzustand ist dieses Feld leer.

Direkt nach dem Start versucht sich der Debugger mit dem Zielrechner („Target“) zu verbinden. Falls das misslingt, erscheint ein Dialog TRY AGAIN? Drücken Sie YES, nachdem Sie folgendes getan haben:

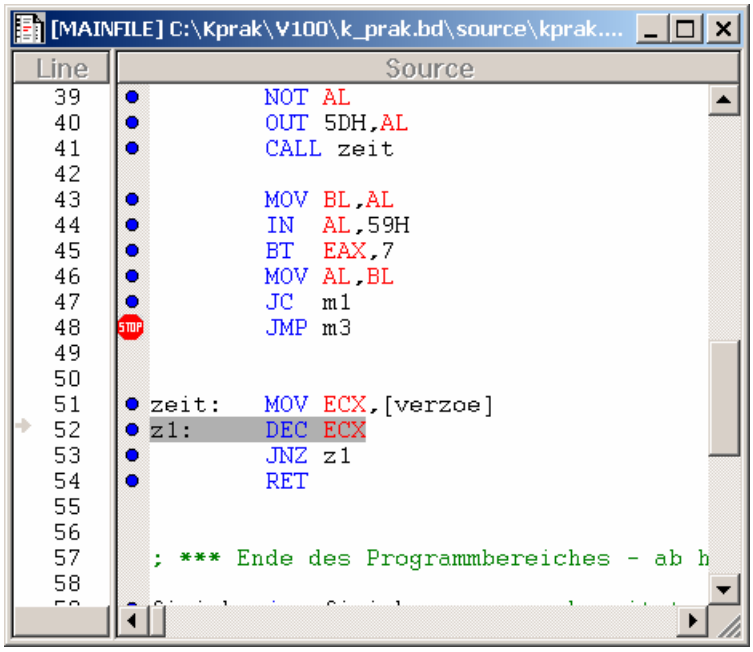
- Kabelverbindung am Anschluss „Seriell 1“ des Zielrechners überprüfen,
- Betriebsanzeige am Zielrechner überprüfen,
- Reset-Taste am Zielrechner drücken und Verlöschen aller Anzeigen außer der Betriebsanzeige abwarten,
- Andere Instanzen des Debuggers auf dem Hostrechner beenden (XDB darf immer nur einmal aktiv sein).

Falls das Problem bestehen bleibt, informieren Sie den Praktikumsbetreuer.

Bei erfolgreicher Verbindung erscheint das **Assemblerfenster** (im Bild links oben, Fensterüberschrift: Assembler). Dieses zeigt im linken Teil Speicheradressen aus dem Programmbereich und im rechten Teil die in den zugehörigen Speicherzellen vorhandenen Maschinencodes und daraus rückübersetzte Assemblerbefehle. So lange kein Programm geladen ist, erscheint ein willkürlicher Inhalt. Sie brauchen dem Assemblerfenster keine Aufmerksamkeit zu widmen, da der Programmtest wesentlich komfortabler mit Hilfe des weiter unten beschriebenen Quelltextfensters erfolgt.

Im normalen Arbeitsablauf aktivieren Sie jetzt die Funktion ‘Download’, um das zuvor übersetzte Programm in den Zielrechner zu laden. Dabei werden gleichzeitig die speziellen Debug-Informationen im Debugger registriert. Nach erfolgtem Laden erscheint das **Quelltextfenster** mit der Fensterüberschrift [MAINFILE]. Das geladene Programm ist jetzt betriebsbereit und kann z.B. mit ‘Run’ oder im Schrittbetrieb ausgeführt werden.

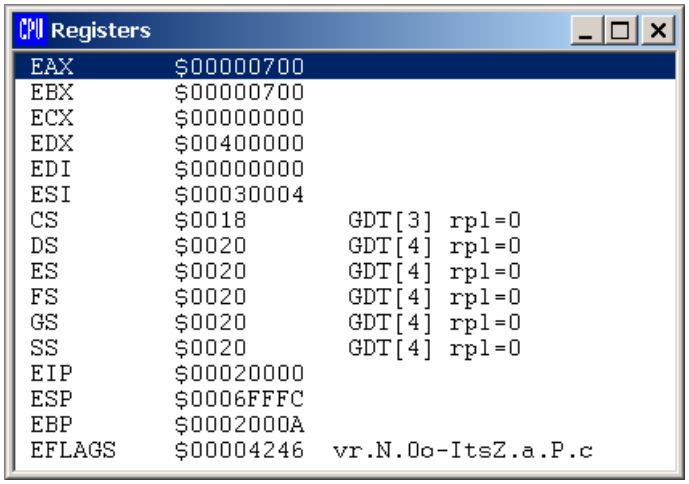
Das Quelltextfenster (siehe auch folgendes Bild) ist das wichtigste Hilfsmittel beim Debuggen. Es zeigt in der rechten, größeren Spalte den Quelltext des Programms, so wie Sie ihn zuvor im Editor eingegeben haben. Die linke Spalte enthält die Zeilennummern. Die graue Unterlegung und ganz links der graue Pfeil kennzeichnen die **aktuelle Position** der Abarbeitung. Der gekennzeichnete Befehl ist stets der als nächstes auszuführende Befehl, er hat also noch nicht stattgefunden. Nach dem Laden ist das zunächst der erste Befehl des Programmbe-



reichs. Die Aktualisierung dieser Anzeige erfolgt beim Anhalten des Programms und beim Schrittbetrieb.

Die blauen Punkte in der mittleren Spalte ermöglichen das Festlegen von **Breakpoints** (Unterbrechungspunkten). Ein Doppelklick auf einen solchen Punkt bewirkt das Erscheinen eines symbolischen „Stoppschilds“. Wenn das Programm mit ‘Run’ gestartet wird und dann einen solchen Breakpoint erreicht, so hält es vor dem Ausführen des gekennzeichneten Befehls automatisch an. Ein erneuter Doppelklick auf das „Stoppschild“ entfernt den Breakpoint.

Eine weitere nützliche Informationsquelle ist das **Registerfenster** (siehe folgendes Bild), welches mit der Menüfunktion ‘*Display – Register*’ aufgerufen wird. Es zeigt die aktuellen Registerinhalte der wichtigsten x86-Register und wird beim Anhalten des Programms und beim Schrittbetrieb aktualisiert, wobei geänderte Werte rot hervorgehoben werden. Die Anzeige erfolgt hexadezimal in der maximalen Breite des jeweiligen Registers. Bei Doppelklick



auf einen angezeigten Wert erscheint ein Dialog, mit dem der Wert geändert werden kann. Die nicht behandelten „Segmentregister“ CS bis SS ignorieren Sie bitte.

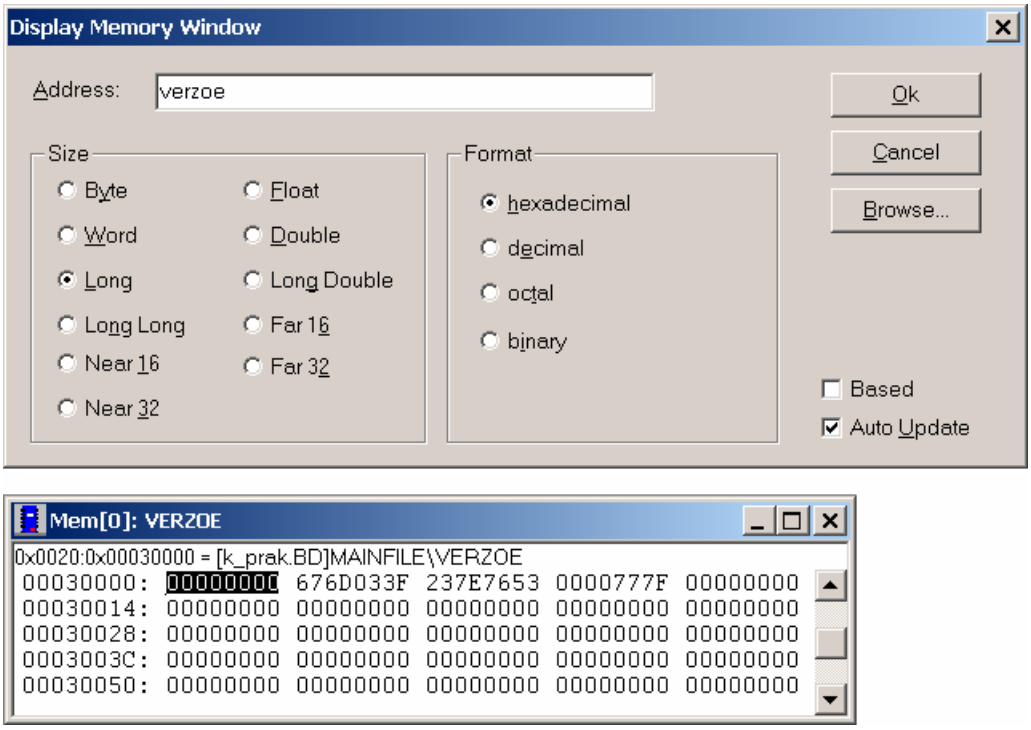
Die Zustände der einzelnen Flags sehen Sie rechts neben dem hexadezimalen Wert von EFLAGS. Jedes Flag wird durch seinen Anfangsbuchstaben dargestellt (siehe „Arbeitsblätter zur Übung“, Abschnitt 2.1). Ein Großbuchstabe steht für ein gesetztes Flag, ein Kleinbuchstabe für ein rückgesetztes Flag. Aus dem im Bild gezeigten Zustand lässt sich also z.B. entnehmen, dass das Zero-Flag gesetzt ist und das Carry-Flag rückgesetzt ist:

```
vr.N.0o-Its Z .a.P. c
```

Mit Doppelklick erreichen Sie einen ausführlicheren Dialog mit der Möglichkeit der Veränderung. Die nicht in der Lehrveranstaltung behandelten Flags können Sie ignorieren.

> Bemerkung für Interessierte:
> Die Anordnung entspricht den Bits im Flagregister. Punkte stehen für unbenutzte Positionen. Das Flag DF wird abweichend zum Gesagten als **+** oder **–** angezeigt, der IOPL als Ziffer.

Mit der Menüfunktion ‘*Display – Memory*’ können Sie ein **Speicherfenster** anfordern, welches die Inhalte eines bestimmten Speicherbereichs anzeigt. Der zunächst erscheinende Dialog (oberes Fenster im Bild) fragt mit Address die Anfangsadresse ab. Sie können den Zahlenwert oder ein Symbol angeben (siehe Abschnitt 4.3.3). Als Size kommen Byte (8 bit), Word (16 bit) oder Long (32 bit, entspricht dem Doppelwort) in Frage. Bei Format wählen

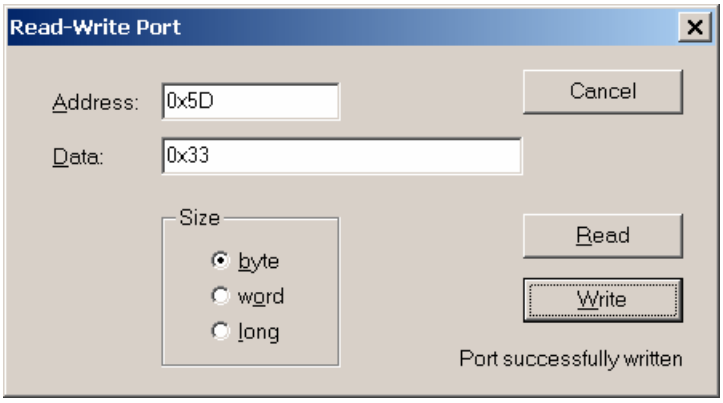


Sie das gewünschte Zahlensystem für die Anzeige.

Das eigentliche Speicherfenster (unteres Fenster im Bild), das bei Klick auf Ok erscheint, zeigt links die Speicheradressen und rechts die Speicherinhalte (im gewählten Format). Mit Doppelklick erscheint ein Dialog zur Änderung des jeweiligen Speicherinhalts. Sie können das Fenster scrollen und skalieren. Der Inhalt wird wie bei den anderen Fenstern aktualisiert (falls Sie Auto Update nicht abgewählt hatten). Beachten Sie aber, dass das Aktualisieren mit wachsender Fenstergröße immer länger dauert!

> Hinweis:
> Die Anzeige im angewählten Format erfolgt ohne Rücksicht auf die Art des dargestellten Inhalts. Es spielt also keine Rolle, ob und wie der betreffende Speicherbereich im Quelltext definiert wurde.

Zuletzt soll noch das **Portfenster** (siehe folgendes Bild) betrachtet werden. Es erscheint bei '*Debug - Read/Write Port*' und ermöglicht die manuelle Ausführung von Ein- und Ausgabebefehlen. Bei der vorliegenden Hardware ist nur die Größe byte sinnvoll. Unter Address ist stets die Portadresse einzutragen. Die Taste Read löst dann eine Eingabeoperation aus und zeigt die gelesenen Daten bei Data an. Für eine Ausgabeoperation trägt man die Daten selbst ein und drückt dann auf Write.



4.3.2 Funktionsübersicht

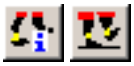
Die Bedienung des Debuggers erfolgt über Symbol- und Textbuttons und über Menüfunktionen. Dieser Abschnitt stellt eine Auswahl der Funktionen tabellarisch zusammen. Bitte benutzen Sie keine Funktionen, über deren Bedeutung Sie sich nicht informiert haben.

Unterlassen Sie Manipulationen an den Einstellungen und Optionen des Debuggers!

Über Buttons aufzurufende Funktionen tauchen im Anleitungstext in der Form 'Funktionsname' auf. Menüfunktionen sind als '*Menüname – Menüpunkt*' referiert.

Das folgende Bild zeigt das Aussehen der **Werkzeugleiste** von XDB. Aufgrund der Konfigurierbarkeit sind geringe Abweichungen möglich.

Die folgende Tabelle erläutert die Funktionen der einzelnen Buttons. Sämtliche Funktionen außer ‘Stop’ sind nur bei angehaltenem Programm ausführbar!

Button	Beschreibung
	**Stop:** Programm anhalten. Es kann eine spürbare Reaktionszeit auftreten.
 Run	**Run:** Programm starten: ungebremster Lauf ab aktueller Position.
	**Instruction Step / Step Into:** Schrittbetrieb eines Maschinenbefehls / einer Quellzeile. Bei Assemblerprogrammen ist das in der Regel kein Unterschied.
	**Recapture Target:** Wiederherstellen der Verbindung zum Zielrechner nach Problemen. Bei Misserfolg: Debugger beenden, Zielrechner rücksetzen und Debugger neu starten.
	**Restart Target:** Neustart des Monitorprogramms. Zur Behebung unklarer Fehlerzustände, falls die Verbindung zum Zielrechner noch funktioniert. Nur selten hilfreich.
	**Show Breakpoint Dialog:** Aufrufen eines Dialogfensters zur übersichtlichen Verwaltung von Breakpoints.
	**About / Help:** Versionsinformation / Hilfefunktion.
Download	**Download:** Laden des Zielprogramms in den Zielrechner und Einrichten des Debuggers. Nur für das erste Laden zu Beginn einer Debuggersitzung.
Reload	**Reload:** Erneuter Download in derselben Debuggersitzung.
Stop_Timer	**Stop Timer:** Anhalten aller Kanäle des PIT 8254 im Zielrechner.
Exit	**Exit:** Beenden des Debuggers (mit zusätzlicher Bestätigung).

Die folgende Tabelle listet einige Funktionen aus den **Menüs** des Debuggers auf, welche zusätzlich von Interesse sein können. Funktionen, die auch über Buttons erreichbar sind, werden dabei nicht wiederholt.

Menüname	Menüpunkt	Beschreibung
Display	Register	Öffnen / Schließen des Registerfensters.
	Memory	Öffnen eines Speicherfensters.
	Toolbar	Werkzeugleisten einstellen. Bitte nicht verändern! Grundzustand: Toolbar praktikum und User-Toolbar aktiv.
Run	Run until	Lauf bis zu einer bestimmten Zeilennummer, Adresse oder Symbol.
Debug	Evaluate	Anzeige des Zahlenwertes für Symbole und Ausdrücke.
	Read/Write Port	Manuelle Ein- und Ausgabeoperationen.
	Symbols	Symbolübersicht anzeigen. Scope sollte dabei auf Global stehen.
	Data	Operationen im Speicher: Füllen, Suchen, Kopieren, Vergleichen.
Window	Windows-typische Funktionen zum Arrangieren der Unterfenster.	

4.3.3 Schreibweise von Zahlen und Adressen

Bei der Eingabe und Anzeige von Zahlenwerten im Debugger XDB sind einige Besonderheiten zu beachten.

- **Hexadezimalzahlen:**

  Die **Anzeige** von Hexadezimalzahlen im Debugger kann an verschiedenen Stellen in unterschiedlicher Schreibweise erfolgen:

  - „Assemblerschreibweise“: 1234H
  - „C/C++ - Schreibweise“: 0x1234
  - „Pascal-Schreibweise“: $1234
  - Ohne Kennzeichnung: 1234

Die Verwendung der verschiedenen Formate ist leider nicht einheitlich. Der letztgenannte Fall tritt meistens dann auf, wenn das Format vorher explizit abgefragt wurde (z.B. im Speicherfenster). Bei der **Eingabe** von Hexadezimalzahlen können die ersten drei Fälle vorkommen. Richten Sie sich nach den Vorgaben in den Dialogfeldern.

Bemerkung:
Im Assemblerquelltext ist stets die „Assemblerschreibweise“ zu verwenden.

- **Speicheradressen:**

  Speicheradressen werden oft in der Form 0018:00200009 oder DS:0x00300002 angezeigt. In diesem Fall ist **nur** die Zahl **hinter** dem Doppelpunkt als Adresse zu betrachten. Der Ausdruck vor dem Doppelpunkt enthält eine Segmentinformation und ist beim „flachen“ Speichermodell uninteressant.

- **Symbole:**

  In vielen Fällen können auch symbolische Bezeichner aus dem Quelltext eingetragen werden, wenn nach einer Zahl oder einer Adresse gefragt wird.

4.3.4 Maßnahmen bei Problemen

Das Zielprogramm und der Debugger besitzen unmittelbaren Zugriff auf alle Hardware-Funktionen des Zielrechners. Deshalb führen Programm- oder Bedienfehler in vielen Fällen zu Abstürzen und anderen schweren Problemen. Wenn sich der Debugger nicht mehr normal bedienen lässt oder sonstige Fehlersymptome zeigt, gehen Sie schrittweise nach dem folgenden Stufenplan vor, bis das Problem gelöst ist.

- Überprüfen Sie, ob evtl. ein einzelnes Dialogfenster des Debuggers über den Bedienfokus (exklusive Bedienbarkeit) verfügt.
- Warten Sie mindestens zehn Sekunden.
- Drücken Sie den Button ‘Stop’ mehrmals im Abstand einiger Sekunden.
- Drücken Sie den Button ‘Reload’ und beobachten Sie das Command-Fenster.
- Drücken Sie den Button ‘Recapture Target’ und dann wiederum den Button ‘Reload’.
- Beenden Sie den Debugger mit ‘Exit’ (kann einige Sekunden dauern), drücken Sie die Reset-Taste am Zielrechner **(nicht am PC!!)**, warten Sie bis zum Verlöschen der Anzeigen und starten Sie den Debugger erneut.

Anhang

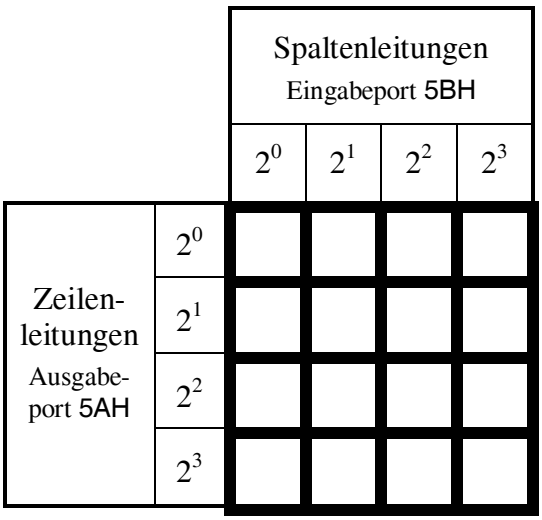
A) Ein- und Ausgabeadressen des Zielrechners

Baugruppe	Teilfunktion	Read/Write	Datenbreite	Adresse	Abschnitt
PPI 8255A	Port A	R/W	8 bit	50 H	4.1.9
	Port B			51 H	
	Port C			52 H	
	Steuerbyte	W		53 H	
PIT 8254	Kanal 0	R/W	8 bit	54 H	4.1.7
	Kanal 1			55 H	
	Kanal 2			56 H	
	Steuerbyte	W		57 H	
Schalterreihe (links)		R	8 bit	58 H	4.1.3
Tastenreihe (rechts)				59 H	
Matrixtastatur	Zeilen	W	8 bit	5A H	4.1.6
	Spalten	R		5B H	
LED-Reihen	Links	W	8 bit	5C H	4.1.4
	Rechts			5D H	
Digital-Analog-Converter (DAC)		W	8 bit	5E H	4.1.8
Sieben-Segment-Anzeigen	Stelle 0 (rechts)	W	8 bit	B0 H	4.1.5
	...			...	
	Stelle 11 (links)			BB H	

B) Hardware-Details einzelner Baugruppen

Matrixtastatur

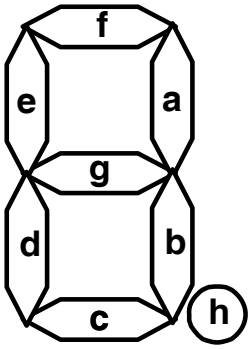
Siehe Abschnitt 4.1.6.



Sieben-Segment-Anzeigen

Siehe Abschnitt 4.1.5.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
h	g	f	e	d	c	b	a



C) Anordnung der Bedienelemente