Praktikumsanleitung Rechnerorganisation
Versuch 2: Informationskodierung

Emulator 8086

Studiengänge Informatik und Basic Engineering School

Dr.-Ing. Karsten Henke

M. Sc. Tobias Fäth

Dipl.-Inf. René Hutschenreuter

Inhalt

1. Einleitung 3
2. Ablauf 3
3. Protokoll und Vorbereitung 3
4. Aufgaben 5
   4.1. Aufgabe 1: Summe zweier Zahlen (Register, Operationen und Flags) 5
   4.2. Aufgabe 2: Umsetzung von Kontrollstrukturen (Kontrollstrukturen und Flags) 5
   4.3. Aufgabe 3: Zähler (Interrupts und Ausgabe) 6
   4.4. Aufgabe 4: BCD Korrektur (Zahlensysteme und Kontrollstrukturen) 6
   4.5. Aufgabe 5: Vorzeichenbetragszahlen (Zahlensysteme, Logische Operationen) 7
   4.6. Aufgabe 6: Potenzfunktion (Speicherzugriff) 7
   4.7. Aufgabe 7: Temperatursteuerung (Ein-/Ausgabe) 7

Anhang A 8

1. Einleitung

Im Versuch 2 – Informationskodierung des Praktikums „Rechnerorganisation/Technische Informatik“ soll die systemnahe Programmierung und die Funktion wichtiger Prozessorkomponenten vermittelt werden. Dazu sind mehrere Aufgaben zu bearbeiten, die eines oder mehrere der folgenden, für die Assemblerprogrammierung wichtigen, Themengebiete abdecken:

- Register
- Operationen
- Flags
- Kontrollstrukturen
- Interrupts
- Ein-/Ausgabe
- Speicherzugriffe
- Zahlensysteme
- Maskierung und logische Operationen

Für die gegebenen Aufgabenstellungen ist in der Vorbereitung jeweils ein Programm in einer Assembler-Sprache zu implementieren. Im Praktikum wird das erstellte Programm dann in einer Simulationsumgebung, die die systemnahe Sicht eines 8086-kompatiblen PCs emuliert, getestet. Bei Fragen zum Praktikum melden Sie sich bitte beim Betreuer der auf der Praktikumswebsite unter http://www.tu-ilmenau.de/iks/lehre/ angegeben ist.

2. Ablauf

Zu Beginn des Praktikums wird Ihnen vom Praktikumsbetreuer der praktikumsspezifische Login mitgeteilt, der Ablauf des Praktikums noch einmal detailliert erläutert und die zum Praktikumstermin zu lösenden Aufgaben beschrieben. Danach sind die vorbereiteten Assembler-Programme in die vorinstallierte Umgebung „emu8086“ zu übernehmen. Mithilfe der Emulationsfunktionen simulieren Sie anschließend Ihr erstelltes Programm und machen sich gegebenenfalls Notizen. Nachdem Sie Ihre Lösung überprüft und den Programmablauf erfolgreich simuliert haben, melden Sie sich beim Betreuer. Der Betreuer beurteilt nun die Korrektheit Ihrer Ergebnisse und stellt gegebenenfalls vertiefende Fragen. Nachdem der Betreuer Ihre Ergebnisse erfolgreich abgenommen hat, können Sie die nächste Aufgabe bearbeiten. Das Praktikum gilt als absolviert, wenn alle geforderten Aufgaben vom Betreuer als zufriedenstellend abgenommen wurden.

3. Protokoll und Vorbereitung

Um zu dem von Ihnen gewählten Praktikumstermin zugelassen zu werden ist die Vorbereitung eines umfangreichen Protokolls notwendig. Das Protokoll ist handschriftlich oder in ausgedruckter Form auszuarbeiten und zum Praktikumstermin mitzuführen. Auf jeder Protokollseite müssen Name und Matrikelnummer vermerkt werden. Das Protokoll muss in einer für den Praktikumsbetreuer lesbaren Form vorliegen.

Insbesondere muss zu jeder Aufgabe:

- Die Lösung der Aufgabe in Form eines Assembler-Programmes vorliegen
- Die Funktionsweise der Lösung durch ergänzenden Text oder durch Kommentare im Quelltext erläutert werden
- Sofern Beispieldaten gegeben sind, die Resultate der einzelnen Beispieldaten analysiert werden.
- Alle in den Aufgabenstellungen gegebenen Fragen beantwortet sein.

Für das Absolvieren des Praktikums ist es nötig sich in der Vorbereitung mit der Simulationsumgebung „emu8086“ vertraut zu machen. Dazu sollten Sie sich eine Demoversion von der Website www.emu8086.com herunterladen und mithilfe dieser die Lösungen für die gegebenen Aufgaben anfertigen. Zusätzlich dazu ist auf den PCs in RTK6 die Software „emu8086“ vorinstalliert und kann zur Vorbereitung genutzt werden.

Das erfolgreiche Absolvieren des Praktikums setzt voraus, das Sie sich in Ihrer Vorbereitung alle Praktikumsaufgaben gründlich durchgelesen haben, sich mithilfe von Anhang A mit der Simulationsumgebung „emu8086“ vertraut gemacht haben und ein Protokoll in beschriebener Form und Ausführung ausgearbeitet und mitgebracht haben. Ihr Protokoll wird zu Praktikumsbeginn vom jeweiligen Betreuer bewertet und Ihnen daraufhin mitgeteilt ob Ihr Protokoll Sie für die Teilnahme qualifiziert. Weiterhin wird zu Praktikumsbeginn ein moodle-basierter Kurztest durchgeführt, um das für die Durchführung des Praktikums nötige Wissen zu überprüfen. Nach Überprüfung des Protokolls und erfolgreichem Absolvieren des Kurztests werden Sie zum Praktikum zugelassen. Ihre Kenntnis der Aufgabenstellung und der Simulationsumgebung „emu8086“ wird während des Praktikums durch den Betreuer mittels vertiefender Fragen nachgewiesen.

Bei Nichterfüllung der oben genannten Anforderungen werden Sie des Praktikums verwiesen. Im Fall eines Verweises müssen Sie sich selbständig um einen Ersatztermin bemühen und können an diesem erneut versuchen das Praktikum erfolgreich abzulegen.

4. Aufgaben

Im Folgenden Abschnitt werden die für das Praktikum vorzubereitenden Aufgaben aufgeführt und detailliert beschrieben. Zu jeder Aufgabe existiert ein Template in welches der zu erarbeitende Quellcode einzutragen ist. In diesen Templates finden sich Beispieldaten und eventuelle Zusatzanweisungen oder Befehle zum starten externer Programme.

4.1. Aufgabe 1: Summe zweier Zahlen (Register, Operationen und Flags)

In dieser Aufgabe soll die Summe der Zahlen in den Registern **AL** und **AH** gebildet und in Register **BL** abgelegt werden.

Ergänzen Sie dazu das vorgegebene Template *„A1_Template.asm“*. Im Template sind vier Beispieldatensätze gegeben. Für jeden Beispieldatensatz sind die bei der Addition entstehenden Sign-, Zero-, Overflow- und Carry-Flags zu analysieren. Wann und warum treten diese bei welchen Beispieldaten auf? Machen Sie sich mit der Simulationsumgebung des „emu8086“, insbesondere mit der Anzeige der Flags und der verschiedenen Abarbeitungsmöglichkeiten (Run, Single Step, Step Back) vertraut.

4.2. Aufgabe 2: Umsetzung von Kontrollstrukturen (Kontrollstrukturen und Flags)

Informieren Sie sich über den Befehl **CMP** und die verschiedenen **Sprungbefehle** (JMP, JZ, JNZ, JE, JNE, JA, JB, JG, JL). Setzen Sie folgende Struktogramme in Assemblercode um.

- Inwiefern unterscheiden sich die Befehle **SUB** und **CMP**?
- Wie werden die Sprungbedingungen rechnerintern ermittelt?
- Gibt es bei den Sprungbefehlen redundante Befehle, warum ist das so?
- Setzen sie die Schleifen (Abb. 2 und Abb. 3) jeweils einmal unter Verwendung von **SUB** und **CMP** um.
- Erläutern Sie die in den Struktogrammen verwendeten Assembler-Befehle.

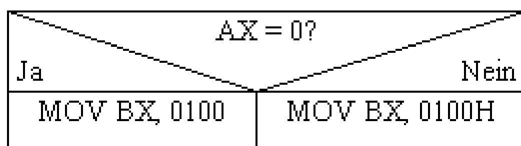


Abbildung 1 Bedingte Verzweigung

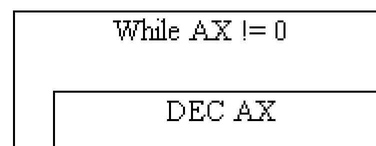


Abbildung 2 Kopfgesteuerte Schleife

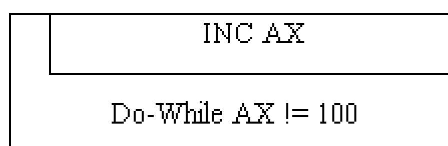


Abbildung 3 Fußgesteuerte Schleife

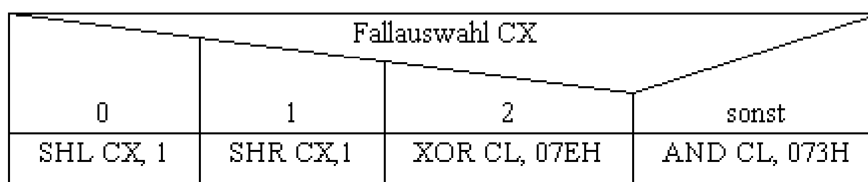


Abbildung 4 Fallauswahl

4.3. Aufgabe 3: Zähler (Interrupts und Ausgabe)

Interrupts sind ein Mechanismus der es Hard- oder Software ermöglicht, die normale Programmabarbeitung in der CPU zu unterbrechen und eine soft-/hardwarebasierte „Interrupt Service Routine“ abzuarbeiten. Machen Sie sich mit der Verwendung des Befehls **INT** in Verbindung mit dem **Interrupt 15** vertraut. Dies soll in dieser Aufgabe mittels **INT 15,86** demonstriert werden. Dieser Interrupt dient dazu den Programmablauf eine angegebene Zeitspanne zu pausieren und wird im Handbuch des „emu8086“ wie folgt beschrieben:

> INT 15h / AH = 86h - *BIOS wait function*
>
> *input:*
>
> - *CX:DX = interval in microseconds*
>
> *return:*
>
> - *CF clear if successful (wait interval elapsed)*
> - *CF set on error or when wait function is already in progress.*

Folgendes Beispiel zeigt die Verwendung des Interrupts:

```
MOV CX, 0FH; 000F4240H = 1.000.000
MOV DX, 4240H
MOV AH, 86H;
INT 15H; Aufruf des Interrupts 15H
```

Machen Sie sich mit dem Befehl **OUT** vertraut. Dieser kann im Template zu Aufgabe 3 benutzt werden, um an eine virtuelle Anzeige an Port **199** eine Zahl zu Senden. Die virtuelle Anzeige kann im Emulator-Fenster unter „virtual devices“ -> „LED_Display.exe“ aktiviert werden. Bei der Benutzung des aufgabenspezifischen Templates öffnet sich diese Anzeige automatisch.

Entwickeln Sie, unter Zuhilfenahme von **INT 15** und **OUT**, einen Zählalgorithmus in Assembler der im Sekundentakt zyklisch von **-20 bis 20** zählt und dies auf dem virtuellen Display anzeigt. Beschreiben Sie Ihre Implementierung. Wie werden positive und negative Zahlen in der 8086 CPU dargestellt? Welche Werte kann das virtuelle Display maximal und minimal anzeigen?

4.4. Aufgabe 4: BCD Korrektur (Zahlensysteme und Kontrollstrukturen)

Betrachten Sie das Template zur Aufgabe 4. Hier wird eine Addition zweier Zahlen in den Registern **AL** und **BL** durchgeführt. Das Resultat der Addition soll in BCD dargestellt werden. Dazu wurde eine Prozedur „bcdcorrect“ angelegt, welche Sie implementieren sollen. Das Resultat der Prozedur soll das Ergebnis der Addition in BCD-Kodierung sein (inklusive des Carry-Flags in Register **AX** sein). Dazu ist es notwendig beide Tetraden des Registers **AL** zu überprüfen und ggf. zu korrigieren. Die Prozedur wird mittels des Befehls „CALL“ aufgerufen und endet mit dem Befehl „RET“. Zur Überprüfung Ihrer Prozedur wurden 4 Testfälle angelegt. Die erwarteten Lösungen nach Abarbeitung der Prozedur wurden in den Kommentaren vermerkt.

Zu Beginn der Prozedur wird im Template das Carry-Flag in Register CL und das Auxillary-Flag in das Register CH gesichert. Bitte informieren Sie sich über die Funktion des Auxillary-Flags zur Korrektur einer BCD Addition.

4.5. Aufgabe 5: Vorzeichenbetragszahlen (Zahlensysteme, logische Operationen)

Gegeben seien in **AX** und **BX** zwei Vorzeichenbetragszahlen. Beispielsweise:

AX: 8024H = -36
BX: 0137H = 311

Wie werden vorzeichenbehaftete Zahlen in den Registern einer 8086-CPU dargestellt? Entwickeln Sie eine Routine, welche Vorzeichenbetragszahlen (wie im Beispiel gezeigt) in die rechnerinterne Darstellung (2K-Format) umwandeln und zurückwandeln kann. Erstellen Sie mithilfe dieser Routine ein Assembler-Programm, welches zwei beliebige Vorzeichenbetragszahlen in den Registern **AX** und **BX** mittels rechnerinterner Darstellung (2K-Format) addiert, das Ergebnis wiederum in eine Vorzeichenbetragszahl umwandelt und in das Register **AX** schreibt.

4.6. Aufgabe 6: Potenzfunktion (Speicherzugriff)

Informieren Sie sich in den Arbeitsblättern Seite 26 ff. über die Möglichkeiten des Speicherzugriffs mittels des **MOV-**Befehls. Welche Adressierungsarten gibt es? Wie kann in der Simulationsumgebung „emu8086“ der Speicherinhalt angezeigt werden? Entwickeln Sie ein Programm zur Berechnung einer Potenzfunktion. Die Basis B befindet sich zu Programmbeginn im Speicher an Adresse [BP]. Ihr Programm soll nun im Speicher aufsteigend, beginnend an der Adresse [BP+1], die Potenzen dieser Basis ablegen, sodass sich folgendes Schema ergibt:

$$[BP + i + 1] = B^i$$

Ihr Programm soll genau dann terminieren, wenn die berechnete Potenz nicht mehr in einer Speicherzelle darstellbar ist. Ermitteln Sie dazu die höchste in einer Speicherstelle darstellbare **nicht-negative** Zahl.

4.7. Aufgabe 7: Temperatursteuerung (Ein-/Ausgabe)

In dieser Aufgabe soll eine Temperatursteuerung für einen Gasbrenner implementiert werden.

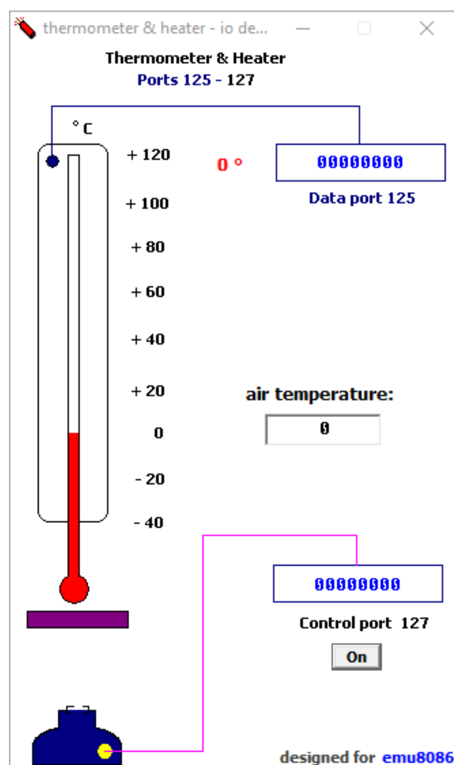
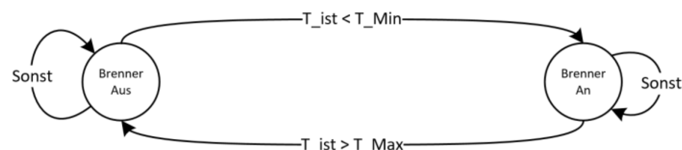


Abbildung 5: Temperatursteuerung

Beim Start der Emulation der Template-Datei zu Aufgabe 7 wird automatisch das links dargestellte Fenster geöffnet. Darin ist die Simulation eines Temperaturregelkreises dargestellt. Der momentane Temperaturwert kann mittels **IN**-Befehl von Port **125** ausgelesen werden. Der Brenner kann mittels **OUT**-Befehl an Adresse **127** an- und ausgeschaltet werden. Hierbei entspricht das Schreiben einer **1** an Port **127** dem Einschalten und das Schreiben einer **0** dem Ausschalten des Brenners. Folgender Automat beschreibt das Verhalten des Regelalgorithmus:



Der Wert T_Min befindet sich zu Programmstart im Speicher an Adresse [BP], der Wert T_Max an Adresse [BP+1].

Entwerfen Sie einen Regelalgorithmus in Assembler, der das gewünschte Verhalten umsetzt.

5. Anhang A – emu8086 Kurzhandbuch

Dieses Dokument dient als kurze Einführung in die Bedienungsweise des Programmes Emu8086.

5.1. Editorfunktionen

Startet man das Programm emu8086, so erscheint ein Fenster mit vier Buttons.

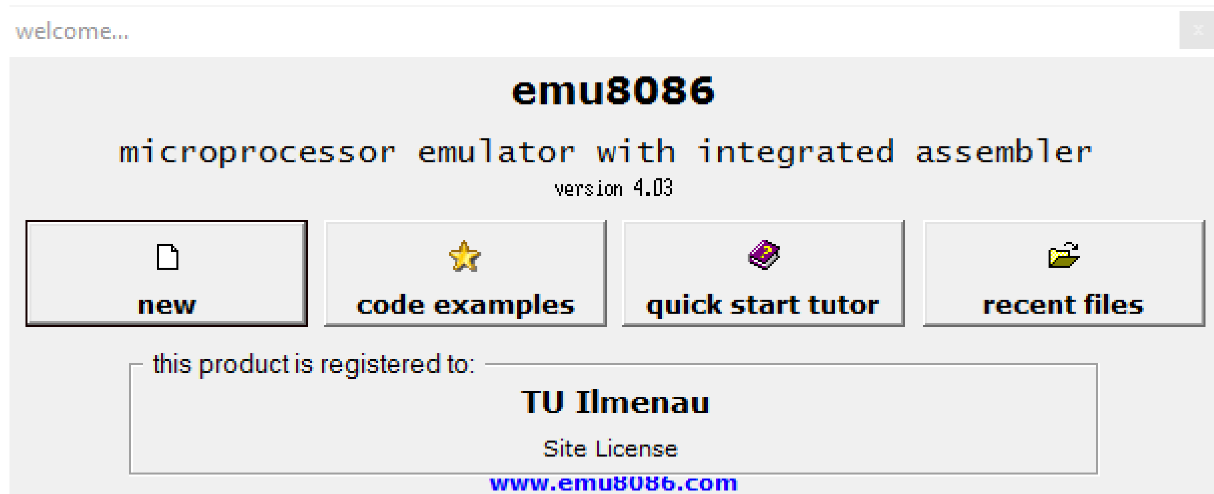


Abbildung 6: Dialog nach dem Start des emu8086

Im Rahmen des Praktikums sind nur die beiden Buttons "new" und "recent files" von Interesse. Mit dem Button "new" erzeugt man eine neue, leere Quellcodedatei. Mit dem Button "recent files" öffnet man bereits vorhandene Quellcodedateien. Da im Rahmen des Praktikums eigene Programme immer ausgehend von dem vorgegebenen Template erstellt werden sollten, muss man zu Beginn das Template mit Hilfe des Buttons "recent files" öffnen. Klickt man auf diesen Button, so öffnet sich eine Pull-Down-Liste, in der sich die zuletzt bearbeiteten Dateien befinden (wenn man das Programm zum ersten mal nach der Neuinstallation startet, ist die Liste natürlich leer). Außerdem befindet sich am Ende der Liste der Eintrag "other files...".

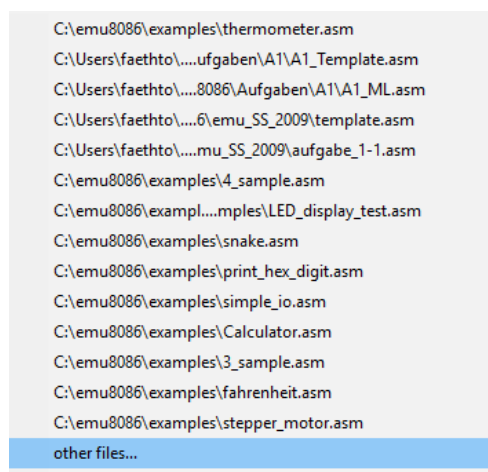


Abbildung 7: Dialog zum Laden einer Datei

Wählt man diesen Eintrag (per Mausklick) aus, so öffnet sich ein Dialogfenster, in dem man die zu bearbeitende Datei auswählen kann. Wenn Sie mit der Bearbeitung einer Aufgabe beginnen, sollten Sie mit Hilfe dieses Dialoges das vorgegebene Template suchen und öffnen. Nachdem Sie das Template geöffnet haben, befinden Sie sich im Editormodus. In diesem Modus können Sie den Quellcode des

Templates bearbeiten und so Ihren eigenen Quellcode einfügen. Mit Hilfe des "save"-Symbols kann man die Datei unter ihrem aktuellen Namen speichern, durch einen Klick auf den kleinen Pfeil rechts neben dem "save"-Symbol kann man die Datei unter einem anderen Namen speichern. Um das leere Template, welches Sie noch als Vorlage benötigen, nicht zu überschreiben, sollten Sie Ihren Quellcode unter einem **anderen Dateinamen** speichern.

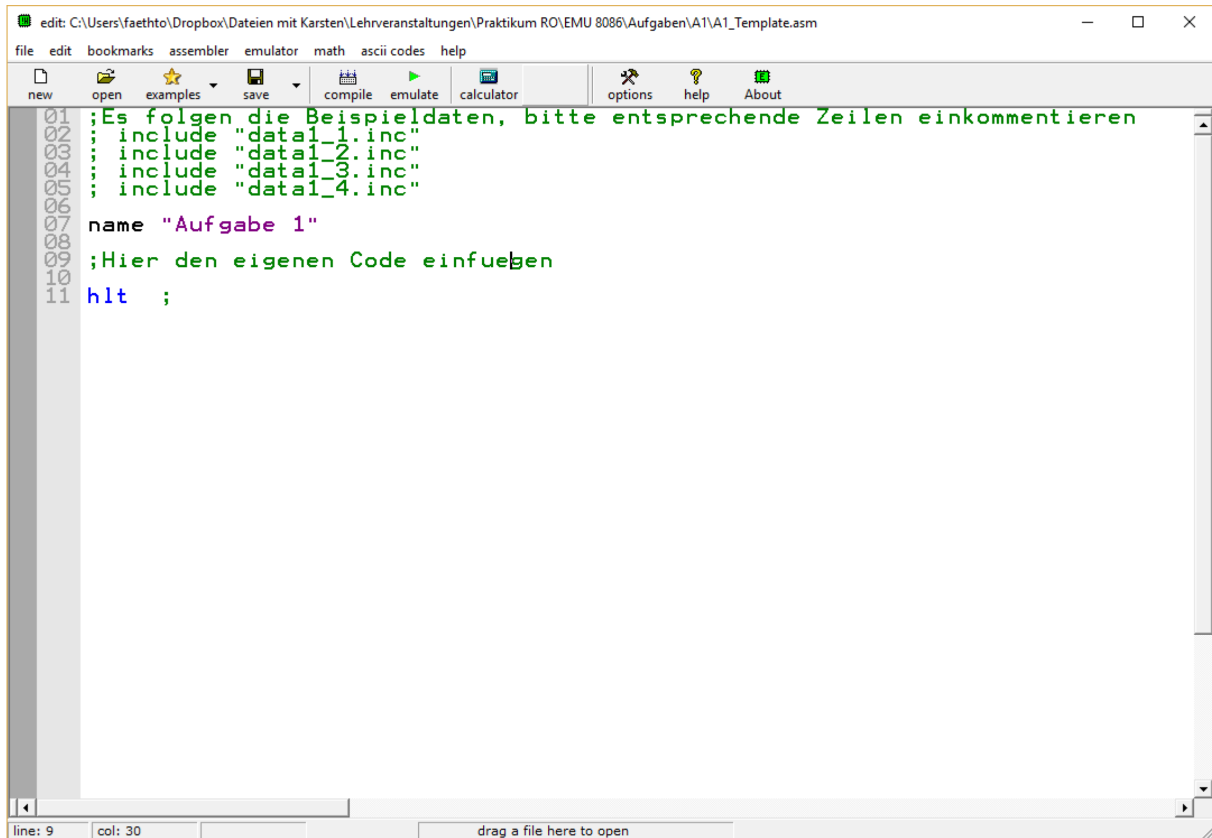


Abbildung 8: Das Template von Aufgabe 1 im Editormodus

5.2. Emulation

Nachdem Sie Ihren Quellcode fertig gestellt und gespeichert haben, können Sie die Emulatorfunktionen des Programmes dazu nutzen, Ihr Programm zu emulieren. Durch einen Klick auf das "emulate"-Symbol, führt das Programm einen Syntaxcheck durch und startet, insofern der Syntaxcheck erfolgreich war, den Emulator. Schlägt der Syntaxcheck fehl, so erscheint ein Dialog, dem Sie entnehmen können, welche Stellen Ihres Quellcodes Fehler enthalten.Die Zahl innerhalb der Klammern gibt die fehlerhafte Quellcodezeile an. Danach folgt eine kurze Beschreibung des Fehlers.

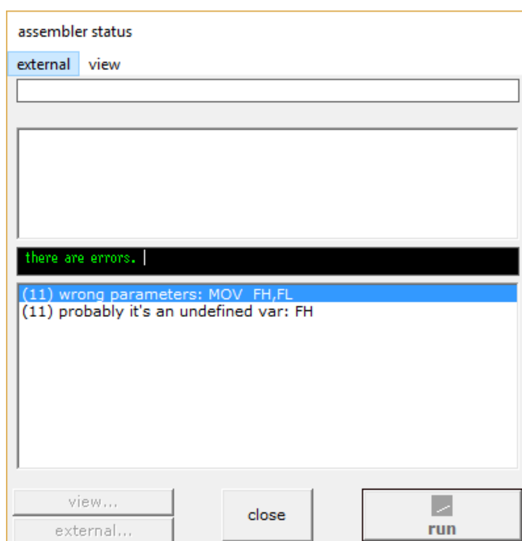


Abbildung 9: Meldung bei fehlerhafter Syntax

Nachdem Sie alle Fehler beseitigt haben, und wiederum das "emulate"-Symbol betätigen, beginnt der Emulator mit der Emulation Ihres Programmes. Es erscheinen nun zwei neue Fenster: zum einen das Hauptfenster des Emulators erkennbar am Fenstertitel "emulator: ...", zum anderen ein Fenster, welches Ihren Quellcode anzeigt, erkennbar am Fenstertitel "original source code".

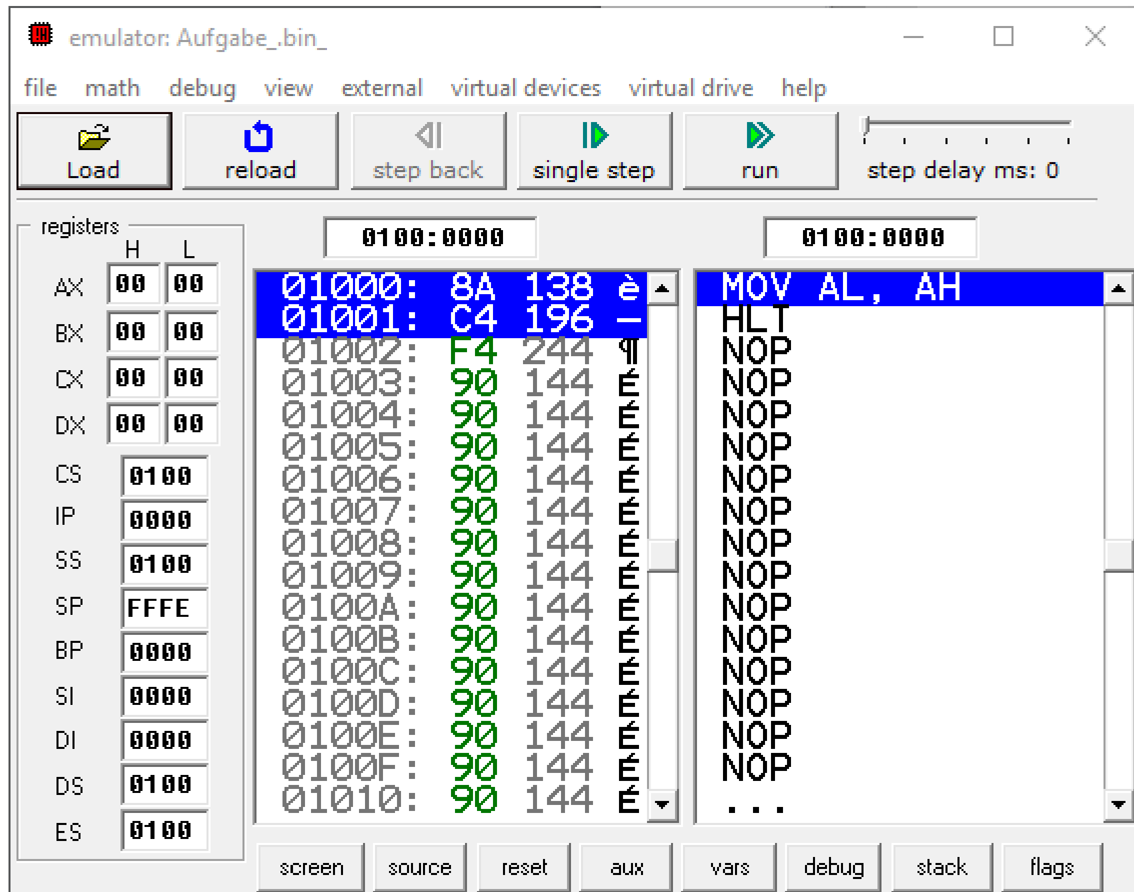


Abbildung 10: Das Hauptfenster des Emulators

Im Hauptfenster kann man den Emulationsvorgang steuern. In dem anderen Fenster wird parallel zur Emulation der gerade aktive Befehl markiert. Am linken Rand des Hauptfensters sind die aktuellen Inhalte der Register dargestellt. Diese ändern sich entsprechend des zuletzt ausgeführten Befehls. Im mittleren Teil des Fensters befindet sich eine Tabelle. Diese Tabelle illustriert den Inhalt des Speichers. In der linken Spalte befindet sich eine fünfstellige hexadezimale Zahl gefolgt von einem Doppelpunkt, welche die Speicheradresse (nicht in der Schreibweise "segment:offset", sondern als physische Speicheradresse) der entsprechenden Zeile darstellt. Nach dem Doppelpunkt folgt der Inhalt der Speicherzelle, welche durch die entsprechende Adresse in der ersten Spalte adressiert wird, als zweistellige hexadezimale Zahl. In der dritten und vierten Spalte wird ebenfalls der Inhalt der Speicherzelle dargestellt, hier aber in dezimaler Schreibweise (in der dritten Spalte) bzw. in Form von ASCII-Zeichen (in der vierten Spalte). Am rechten Rand des Hauptfensters befindet sich eine weitere Darstellung des Speichers. Hier werden alle Speicherinhalte als Maschinenbefehle interpretiert. Dabei ist zu beachten, dass es sich nicht tatsächlich auch um Maschinenbefehle handeln muss. Es wird hier einfach "blind" eine Reihe aufeinanderfolgender Bytes als Befehl interpretiert (auch, wenn Sie vom Programmierer eigentlich als Daten interpretiert werden würden), so wie es auch die CPU eines realen Rechners tun würde, falls der Befehlszeiger auf diese Speicherstelle zeigen würde. Über den beiden

Tabellen befindet sich jeweils eine Box, in der jeweils die Anfangsadresse des gerade betrachteten Speicherbereichs steht. Es gibt zwei Modi, um nun das Programm ablaufen zu lassen.

Mit Hilfe des Buttons "single step" kann man das Programm im Einzelschrittmodus ausführen. Dabei wird bei Betätigung des Buttons immer der sowohl in den beiden Tabellen des Hauptfensters, als auch im "original source code"-Fenster markierte Befehl ausgeführt. Nach dessen Ausführung wird der Programmlauf gestoppt und der nächste Befehl markiert. Im zweiten Modus wird das Programm vom Anfang bis zum Programmende durchlaufen, so wie es auch auf einer realen CPU ablaufen würde. Dieser Modus wird mit Hilfe des Buttons "run" gestartet. Rechts neben dem "run"- Button befindet sich ein Slider, mit dem man die Geschwindigkeit des Programmlaufes einstellen kann. Der nächste Befehl wird immer mit der dort eingestellten Verzögerung ausgeführt. Auch im "run"-Modus wird der aktuell ausgeführte Befehl im Hauptfenster und im "original source code"-Fenster.

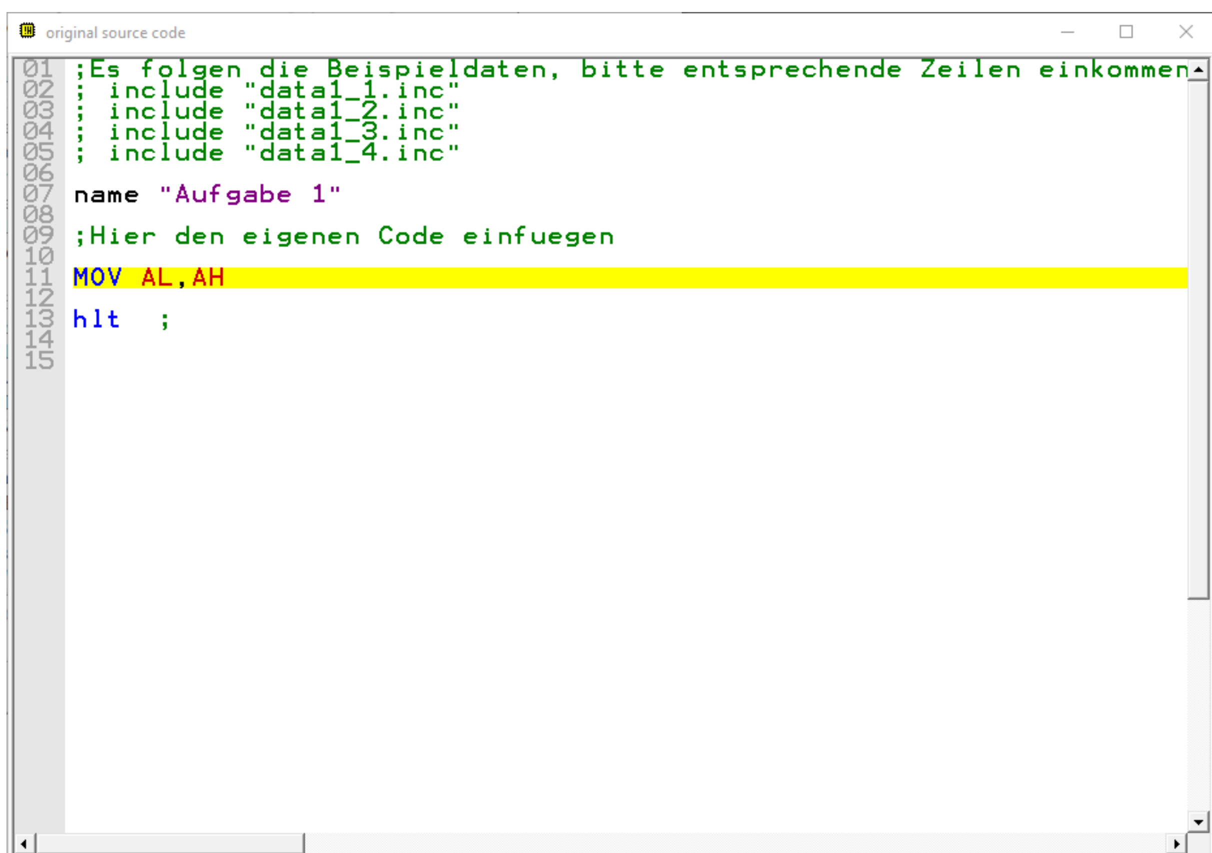


Abbildung 11: Quellcode mit aktuellem Befehl markiert

Um im "run"-Modus den Programmlauf an einer bestimmten Stelle des Programmes automatisch stoppen zu lassen, kann man einen sogenannten Breakpoint setzen. Im Menü des Haupfensters findet man unter "debug"->"set breakpoint" die entsprechende Funktion. Um den Programmlauf nun vor der Ausführung eines bestimmten Befehls zu stoppen, markiert man den Befehl im Hauptfenster oder im "original source code"-Fenster durch einen Klick mit der linken Maustaste auf den Befehl und wählt dann "debug"->"set breakpoint" im Menü des Hauptfensters aus. Nachdem man den Breakpoint wie beschrieben gesetzt hat, startet man den Programmlauf mit Hilfe des "run"-Buttons. Das Programm wird nun emuliert und hält vor der Ausführung des entsprechenden Befehls an. Mit Hilfe des "single step"- Buttons kann man nun den nun markierten Befehl ausführen. Betätigt man wieder den "run"-Button, so wird die Emulation fortgeführt. Falls man den Breakpoint innerhalb einer Schleife setzt, so wird der Programmlauf jedesmal, wenn er den entsprechenden Befehl passiert, angehalten. Um das

Programmverhalten während der Emulation genauer untersuchen zu können, stellt der Emulator weitere Werkzeuge zu Verfügung. Die wichtigsten sind das Memory-Fenster, des Flag-Fenster und das Stack-Fenster. Diese Fenster lassen sich über das "view"-Menü des Hauptfensters öffnen. Das Memory-Fenster (siehe Abb. 7) kann mit Hilfe des Eintrages "view"->"memory" geöffnet werden.

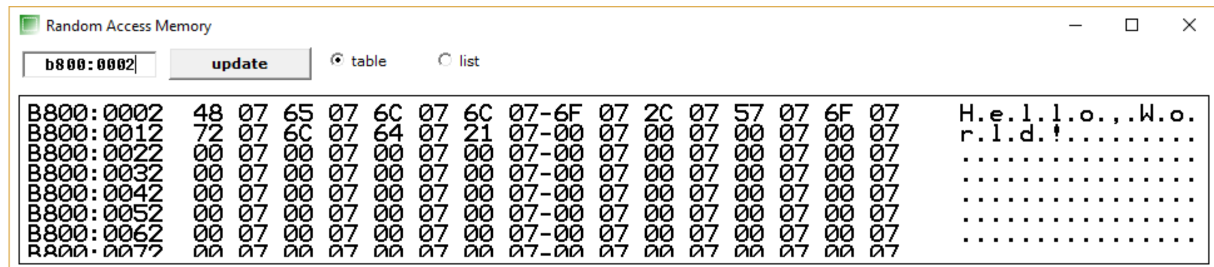


Abbildung 12: Das Memory Fenster

Dieses Fenster zeigt einen Ausschnitt des dem Programm zur Verfügung stehenden Hauptspeichers. Hier können die Auswirkungen von Speicheroperationen während des Programmlaufes beobachtet werden. Im oberen Teil des Fensters befindet sich eine Box, in der die Anfangsadresse des anzuzeigenden Speicherbereiches steht. Trägt man hier eine andere Adresse ein, so wird der entsprechende Speicherbereich angezeigt. Werden nun während des Programmlaufes Speicheroperationen auf dem aktuell angezeigten Speicherbereich ausgeführt, so kann man deren Auswirkungen hier beobachten. Der Speicherbereich wird in Form einer Tabelle angezeigt. Die erste Zahl in jeder Spalte gibt dabei jeweils die Anfangsadresse der in der Zeile dargestellten Byte-Folge an. Rechts davon folgen 16 Bytes, von denen jedes jeweils als zweistellige hexadezimale Zahl codiert ist. Dabei hat das erste Byte die am Anfang der Zeile angegebene Adresse, das zweite Byte die am Zeilenanfang angegebene Adresse plus eins, das dritte Byte plus zwei, ..., das letzte Byte plus 15. Am rechten Rand der Tabelle wird der Speicherinhalt der entsprechenden Zeile in ASCII-codierter Form dargestellt. Mit Hilfe des Stack-Fensters, welches sich über den Menüeintrag "view"->"stack" öffnen lässt, kann man den als Stack verwendeten Speicherbereich inspizieren. Natürlich ist dies auch mit dem Memory-Fenster möglich, da es sich ja auch nur um einen Speicherbereich handelt, allerdings stellt das Stack-Fenster eine "stack-gerechtere" Ansicht zur Verfügung. Nachdem die erste Stack-Operation (üblicherweise der Befehl PUSH, da der Stack zu Beginn noch leer ist) ausgeführt wurde, springt der Pfeil im rechten Teil des Stack-Fensters auf den "obersten" Stackeintrag (also der Eintrag, der als letztes auf den Stapel gelegt wurde). Der Pfeil wird bei einem PUSH-Befehl um eine Zeile nach unten versetzt, bei einem POP-Befehl um eine Zeile nach oben (Man beachte hier, wie der Stack im Speicherbereich organisiert ist. Er beginnt im hier gezeigten Beispiel am Ende des Segments, welches die Segmentadresse 0100 trägt. Mit einem PUSH-Befehl wird der Wert auf den Stack gelegt und der Stack-Zeiger um eins verringert.). Mit Hilfe des Flags-Fensters (kann man den aktuellen Zustand der Prozessor-Status-Flags nach jedem Befehl inspizieren.

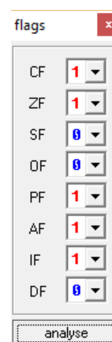


Abbildung 13: Das Flags Fenster

Nach der Ausführung eines Befehls werden die Inhalte der einzelnen Flags hier angezeigt. Die Flags, die sich mit der Ausführung des Befehls geändert haben, werden rot markiert, alle anderen blau. Neben den hier aufgezählten Fenstern, kann man im "view"-Menü noch weitere finden, die aber weniger häufig benötigt werden.

5.3. Typische Ablauf der Emulation eines Programms

Im Folgenden soll exemplarisch der typische Ablauf einer Emulation dargestellt werden. Typischerweise beginnt man mit der Erstellung des Quellcodes im Editor-Fenster. Nachdem man den Quellcode fertig gestellt und alle Fehler beseitigt hat, startet man die Emulation mit Hilfe des "emulate"-Buttons. Im Emulator öffnet man nun mindestens das Memory-Fenster, falls nötig auch weitere, wie das Stack-Fenster und das Flags-Fenster. Will man nun die Wirkung eines bestimmten Teils des Quellcodes genauer untersuchen, setzt man den Breakpoint auf den Anfang des zu untersuchenden Bereiches und startet den Programmlauf im "run"-Modus. Nachdem der Breakpoint erreicht wurde, führt man den zu untersuchenden Code-Bereich im Einzelschrittmodus aus und beobachtet dabei den Zustand des Programmes mit Hilfe der zuvor geöffneten Fenster (Memory-Fenster, Stack-Fenster, ...).