FAKULTÄT ELEKTROTECHNIK

Hochschule für Technik und Wirtschaft Dresden
University of Applied Sciences

Praktikum Elektronik für Informatiker

Teil 3: Mikrocontrollertechnik (MCT)

Versuche: **MCT1: Grundlagen**
MCT2: I/O- Programmierung

1 Versuchsziele

- Kennen lernen der Besonderheiten der Mikrocontroller-Hardware am Beispiel der **8051-Mikrocontroller-Familie**
- Einführung in die Technologie der Projektentwicklung und die Benutzung von Entwicklungssystemen am Beispiel der integrierten Entwicklungsumgebung **µVision2**
- Nutzung ausgewählter Input/Output-Funktionen des **Mikrocontrollers SAB 80C517**
- Entwicklung von Applikationen mit Schwerpunkt I/O-Programmierung
- Programmierpraxis mit einem **8051-Prototyping-Board** und Anwendung des **Remote-Debug**, einer praktischen Methode zur Fehlerbeseitigung mit realer Hardware.

2 Grundlagen

2.1 Motivation

Mikrocontroller erfüllen als Basiskomponente von eingebetteten Systemen – dem Anwender meist verborgen – vielfältige Aufgaben der Steuerung, Kommunikation, Datenerfassung- und Verarbeitung. Das betrifft zahlreiche Technikbereiche, wie die Kommunikations-, die Fahrzeug- oder die Automatisierungstechnik, insbesondere auch die Sensorik oder den Bereich der Mensch-Maschinen-Kommunikation. Die Versuche MCT1 und MCT2 knüpfen deshalb an die in den Lehrveranstaltungen „Elektronik für Informatiker“ und „Grundlagen der Informatik“ vermittelten Grundkenntnisse an und geben eine praktische Einführung in die elementaren Grundlagen der Mikrocontroller-Programmierung und -Anwendung. Schwerpunkt ist dabei die Praxis der I/O-Programmierung, insbesondere für die Funktionen „Messen“ und „Stellen“.

2.2 Voraussetzungen

Zur Vorbereitung auf die Versuche MCT1 und MCT2 ist Folgendes erforderlich:

- Machen Sie sich mit den Arbeitsmöglichkeiten unseres Labors Mikrorechentechnik vertraut. Besuchen Sie dazu die Homepage: www.htw-dresden.de/fe/labors.htm und wählen Sie → **Mikrorechentechnik** aus.
- Arbeiten Sie die Seite → **Praktikum** durch. Nicht notwendig aber hilfreich ist es, die **Download-Angebote** (Software-Entwicklungsumgebung (IDE) und vorgegebenes Programmierprojekt für 8051-Mikrocontroller) zur ergänzenden Vorbereitung zu nutzen.
- Das ausgegebene Lehrmaterial „Laborpraktikum Mikrorechentechnik – Mikrocontroller SAB 80C517A/80C537“ enthält eine **Kurzdokumentation** für die Versuchsdurchführung. Sie ist vor dem ersten Termin zu sichten. Verschaffen Sie sich insbesondere einen Überblick über den **Befehlssatz**, die Einteilung, die Syntax und die Wirkung seiner Befehle (S. 12 bis 14) und ihre Anwendung (S.16 und 17).
- Zu den Versuchsaufgaben der vorliegenden Anleitung sollten erste Lösungsideen entwickelt werden.
- Der **Leitfaden** zur Bedienung der Entwicklungsumgebung (siehe ab Seite 5) ist detailliert durchzuarbeiten.

Wichtiger Hinweis: Das Praktikum findet im **Laborgebäude** (LGS**) Schnorrstraße 29**, Ecke Andreas-Schubert-Straße, in den Räumen **L 216 bis L 218 (Gebäudeeingang Süd)** statt.

3 Versuchsdurchführung

3.1 Allgemeine Zielstellung und Hinweise

Es ist ein Assemblerprogramm [1],[2] für einen Mikrocontroller vom Typ SAB 80C537 [3], [4] zu erarbeiten und zu erproben. Dazu soll der Arbeitsplatz, wie er in Bild 1 dargestellt ist, genutzt werden. Ein Eingangssignal wird mittels Funktionsgenerator konfiguriert und an Port 7.0, einem Analogeingang (AE) des Mikrocontrollers, angelegt. Das Oszilloskop dient zu dessen Darstellung.

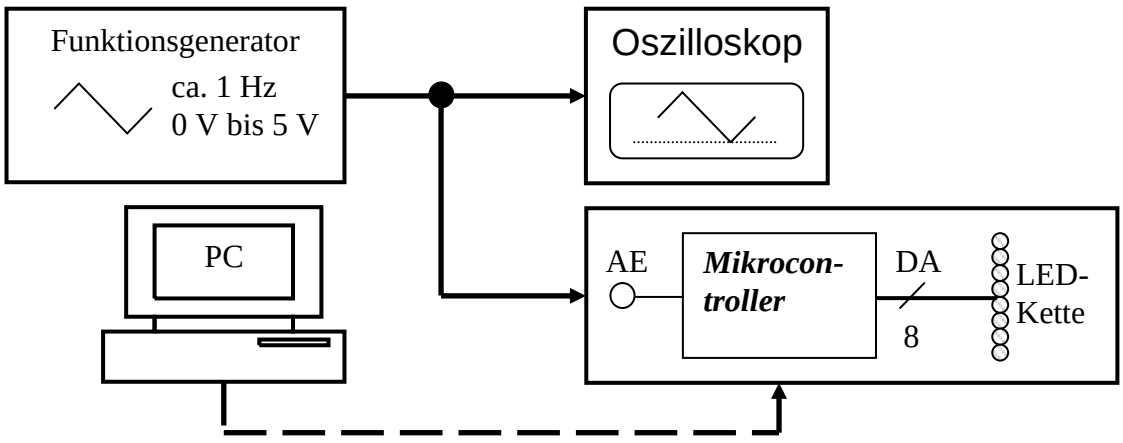

Bild 1 Prinzipschaltbild des Praktikumsplatzes (AE: Analogeingang, DA: Digitalausgang)

Ziel des Praktikums ist es, den an AE anliegenden Messwert zyklisch über ein LED-Modul auszugeben. Es besteht aus einer LED-Kette von acht LEDs und ist an Port 4, einem Digitalausgang (DA) des Mikrocontrollers, angeschlossen. Mit Hilfe des PCs am Arbeitsplatz (auch Host-PC genannt) wird das Assemblerprogramm entwickelt und anschließend in den Mikrocontroller, der sich auf einer Prototypenkarte befindet, geladen. Auf dem PC befindet sich unter dem Laufwerk **D** Ihr persönlicher Arbeitsordner **work_51i**. Starten Sie hier durch Doppelklick die Datei **work.Uv2**. Weitere Anweisungen finden Sie unter Kap. 4. Vergessen Sie nicht, vor dem Beenden des

Praktikums Ihre Daten zu sichern, denn dieser Ordner wird vor jeder Lehrveranstaltung in seinen Ausgangszustand versetzt!

3.2 Versuchsaufgaben

Die Versuchstermine beinhalten jeweils eine Einführung und Diskussion der Grundlagen der Laborversuche und anschließend eine individuelle Aufgabenbearbeitung in den Praktikumsgruppen.

Erster Termin - Laborversuch MCT1:

- Einweisung
- Einführung in die Hardware der Mikrocontroller-Familie 8051 [1], [2]
- Anleitung zur Benutzung der integrierten Entwicklungsumgebung **µVision2** und des Remote-Debug-Systems (Siehe Kap. 4, Vierter Schritt)
- Bearbeitung folgender Aufgabe:
  Die am Portpin AE anliegende veränderlich analoge Messgröße ist zyklisch zu erfassen und auf der LED-Kette wahlweise als Punkt oder Balken linear darzustellen. Dabei soll eine Spannung von 0 V am Digitalausgang DA dem Binäräquivalent "0000 0000", d.h. dem Zustand „alle LED aus", entsprechen. Die obere Spannungsgrenze von 5 V soll an DA mit dem Binäräquivalent "1111 1111" korrespondieren. In diesem Fall ist wahlweise die achte LED (Punktdarstellung) oder sind alle LED (Balkendarstellung) eingeschaltet.

Zweiter Termin - Laborversuch MCT2:

- Einführung in die Timer-Programmierung und Interrupt-Steuerung
- Bearbeitung folgender Aufgabe:
  Erstellen Sie zunächst ein Unterprogramm, das die Erfassung der an AE anliegenden Messgröße übernimmt. Es sind nun nacheinander 256 Analog-Digital-Umsetzungen durchzuführen. Diese Daten sollen anschließend zur Messwertglättung einer Mittelwertbildung unterzogen werden. Als Ergebnis ist der gemittelte Wert im Akku vom Unter- an das Hauptprogramm zu übergeben. Dabei bleibt der Programmteil zur Ausgabe der Punkt- bzw. Balkendarstellung erhalten.
- Optionale Lösung einer Zusatzaufgabe, z.B. Software-Realisierung eines spannungsgesteuerten Oszillators (Schwerpunkte: Timer-Programmierung, Interrupt-Steuerung).

3.3 Vorgaben und Hinweise zum Laborversuch MCT1

Das zu erstellende Programm sollte, wie in Bild 2 dargestellt, aufgebaut sein.

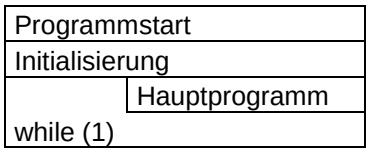

Bild 2 Struktogramm eines einfachen Programms

Aufgrund der Architektur des verwendeten Mikrocontrollers sind einige Besonderheiten beim Programmstart zu beachten [3],[4]. Die Adresse "23h" ist ein Einsprung für die Interrupt-Service-Routine (ISR) „Universal Asynchronous Receiver Transmitter" (UART), serieller Kanal 0, die der Kommunikation mit dem Host-PC dient. Hierfür werden im Programm drei Byte reserviert. Folgebefehle befinden sich also erst ab Adresse "26h". Deshalb ergibt sich im Befehlsspeicher die Struktur aus Bild 3.

PC	Direktiven	Marken	Befehlsspeicher
0	cseg at 0h		jmp start
1			---
…			---
…			---
23	cseg at 23h, ds 3		Debugger-ISR
24			Debugger-ISR

25		Debugger-ISR
26	start	clr ADM
…		…

Bild 3 Befehlsabfolge zum Programmstart

Ein einfaches Rahmenprogramm ohne weitere Interrupt-Einsprünge hat damit folgenden Aufbau:

```
$include(reg517a.inc)         ;Register_Symboldefinitionen

;--------Programmstart-------------------
      cseg  at 0          ;Assembler-Direktive cseg:
                          ;                  ext. Programmspeicher, at: Adr. 0
      jmp   start         ;Überspringe reservierte Codesegmente
      cseg at 23h         ;UART: ser.Kanal 0 (Einsprung ISR - Debugeinsprung!!)
      ds    3             ;3 Byte für Debugger-ISR reservieren

;--------Initialisierung-----------

      ;  <---Hier sind eventuelle Initialisierungen vorzunehmen!!

;--------Hauptprogrammschleife-----------
main:

      ;  <---Hier ist das Programm zu implementieren!!

      jmp   main          ;Rücksprung zu main

;--------Programmende--------------------
      end
```

Folgendes Beispiel liest einen Analogwert von Port 7.0 ein und gibt den 8-Bit-Wert auf Port 1 aus:

```
;Initialisierung ADU
      clr   ADM           ;ADU: einmalige Umsetzung
                          ;     (ADM-Bit (aus Adcon0-Byte) = 0)
      clr   ADEX          ;ADU: sofortiger Start
                          ;     (ADEX-Bit (aus Adcon0-Byte) = 0)
                          ;ADU: Adcon0 nicht mit mov-Befehl laden!
      mov   adcon1,#00h   ;ADU: Kanal 0 (P7.0)
                          ;ADU: erst Adcon0, danach Adcon1 laden!

      ;Analogwert einlesen
      mov   addat1,#0     ;ADU: Lowteil Null setzen (Addat: Ergebnisregister)
      jb    BSY,$         ;ADU: warten bis Busy-Bit (Adcon0-Byte) = 0
      mov   a,addath      ;ADU: Highteil auslesen: 8 Bit und in Akku schreiben

      ;Ausgabe
      mov   p1,a          ;P1: Akku auf P1 ausgeben
...
```

3.4 Vorgaben und Hinweise zum Laborversuch MCT2

Das nachfolgende Beispiel veranschaulicht das Prinzip eines Unterprogrammaufrufs:

```
...
;--------Hauptprogrammschleife-----------
main:
      …

      lcall dummy         ;Unterprogrammaufruf
```

```
      …

      jmp   main        ;Rücksprung zu main


;--------Unterprogramm------------------
;Beginn Unterprogramm (Übergabewert: --)
dummy:
      ;…
      push  psw         ;optionale Rettung des PSW
      push  acc         ;optionale Rettung des Akkumulators

      ;Programmbeispiel:
      mov   r0,#0       ;CPU: Register 0 wird mit Konstante 0 geladen

      pop   acc         ;Wiederherstellung des Akkumulators falls gerettet
      pop   psw         ;Wiederherstellung des PSW falls gerettet
      ret               ;Rücksprung zum Hauptprogramm
;Ende Unterprogramm (Ausgabe: Ausgabewert im Register 0)
...
```

3.5 Einige Programmierhinweise

Damit Sie effizient arbeiten können, empfiehlt es sich, die nachstehend genannten Punkte zur Entwicklung eines Softwareprojektes (für kleinere Anwendungen) zu beachten:

1. Aufgabe genau formulieren, ggf. erweitern, und in Teilaufgaben gliedern (Eine präzisierte Aufgabenstellung enthält schon implizit die Lösung!)
2. Umwandlung dieser impliziten Lösung in ein Struktogramm bei entsprechend notwendigen Erweiterungen => Ergebnis ist reale Lösung
   (Beachten Sie: Struktogramm ist mit Struktur verwandt! Die grundlegende Struktur, d.h. Verwendung von Verzweigungen, Sprüngen, notwendige Variablen usw. ist zu entwickeln!)
3. Durchführung einer Beispielberechnung anhand beliebig gewählter Eingangsvariablen, die natürlich im Definitionsbereich liegen müssen! (Das dient sowohl dem Verständnis als auch der Verifikation der geplanten Programmstruktur.)
4. Präzise Dokumentation der einzelnen Arbeitsschritte und der Lösungsfindung
5. Erstellen des eigentlichen, gut auskommentierten Programms (Version 1.0)
   Hieraus ergeben sich weitere Fragen und Anforderungen. Ein Rücksprung zu einem der vorhergehenden Punkte wird dann notwendig.

4 Leitfaden zur Bedienung der Entwicklungsumgebung

Zu Beginn wird die Entwicklungsumgebung (engl. Integrated Development Environment, IDE, [5]) durch Doppelklick auf die Datei **work.Uv2** in Ihrem Arbeitsordner (**D:\work_51i**) geöffnet. Das Projekt einschließlich seiner notwendigen Einstellungen ist bereits für Sie angelegt. Danach erscheint ein Fenster, wie in Bild 4 dargestellt. Es ist in drei Teilbereiche (**A**: Project Workspace, **B**: Editor, **C**: Output Window) gegliedert. Im Editor steht der Code des einfachen Rahmenprogramms aus Abschnitt 3.3.

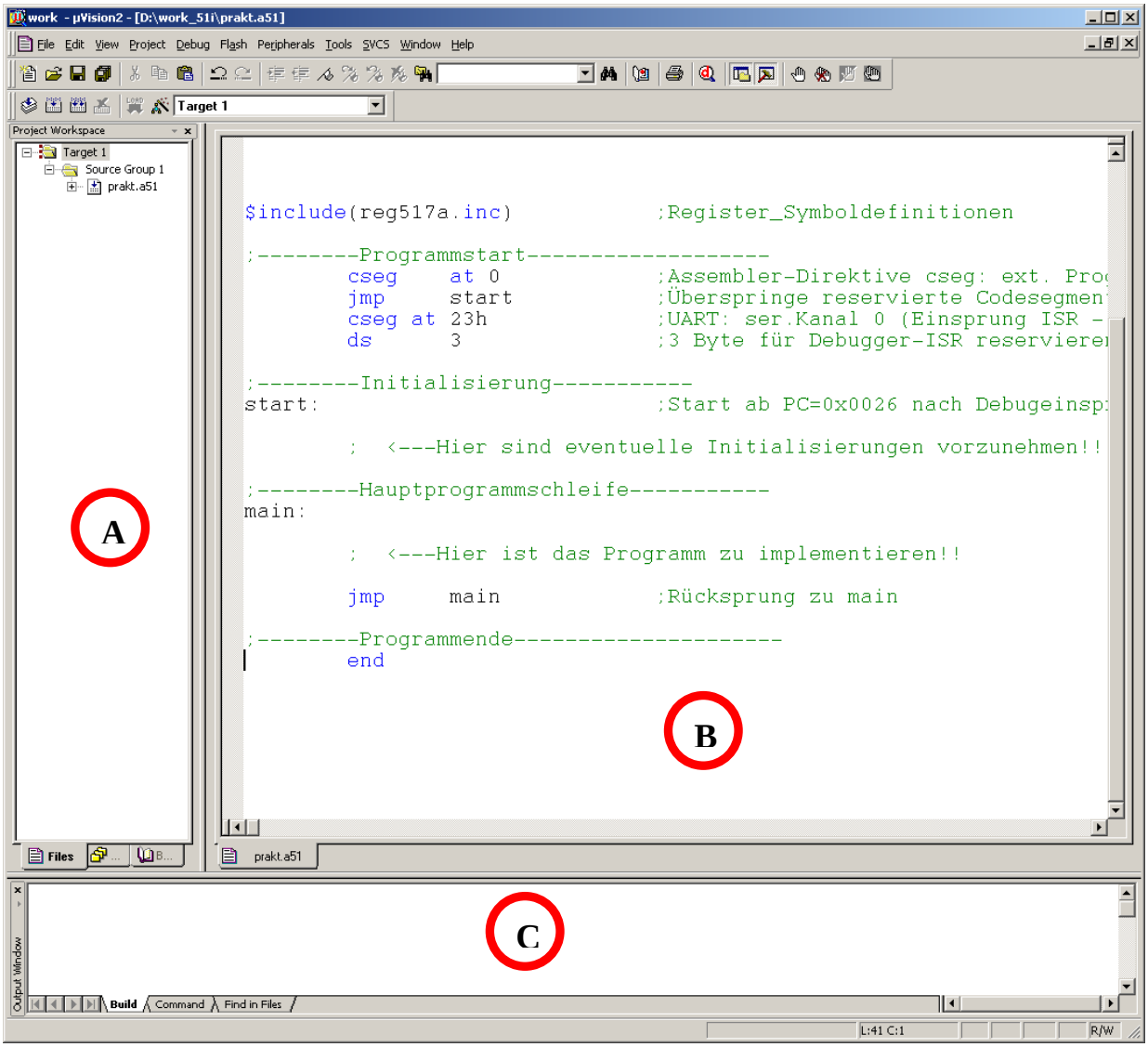

Bild 4 Fenster der Entwicklungsumgebung (A: Project Workspace, B: Editor, C: Output Window)

Erster Schritt: Editieren von Quellcode

Ergänzen Sie den Code in der Hauptprogrammschleife nach der Marke **main** um folgende Zeilen:

```
add   a,#1        ;CPU: Addiere zum Akku 1
mov   p4,a        ;P4: Akku auf Port P4 ausgeben
```

Dieses kleine Beispielprogramm besteht im Wesentlichen aus einer Schleife ohne Abbruchbedingung. Der Befehl **add** führt eine Addition des Akku-Inhaltes sowie der Konstante **1** aus und der Befehl **mov** transportiert den Akku-Inhalt nach Port 4. Somit wird der Akku bei jedem Schleifendurchlauf innerhalb der Struktur um den Wert **1** erhöht und anschließend dessen Inhalt auf dem Port 4, an den das LED-Modul angeschlossen ist, ausgegeben.

Zweiter Schritt: Assemblieren und Linken

Nach dem Editieren soll das Assemblerprogramm übersetzt werden. Dazu ist die Taste [F7] zu drücken oder der Menüeintrag **Project → Build target** bzw. **Project → Rebuild all target files** zu wählen. Als Konsequenz erscheint im Output Window der Text:

```
Build target 'Target 1'
assembling prakt.a51...
linking...
Program Size: data=8.0 xdata=0 code=12
"work" - 0 Error(s), 0 Warning(s).
```

Dritter Schritt: Korrektur syntaktischer Fehler
Ersetzen Sie in Zeile 36 den richtigen Befehl **mov** durch das Wort **move** (Hinweis: Zeilen- und Spaltennummer des Cursors werden im Fenster rechts unten angezeigt (siehe Bild 5)). Starten Sie erneut den Assembler und beobachten Sie die Ausgabe im Output Window.
Durch Doppelklick auf die Fehlerbotschaft (in diesem Fall ist es ein Syntaxfehler) erscheint am linken Rand des Editors ein blauer Pfeil, der, wie im Bild 5 gezeigt, auf die Fehlerursache verweist. Berichtigen Sie den Fehler!

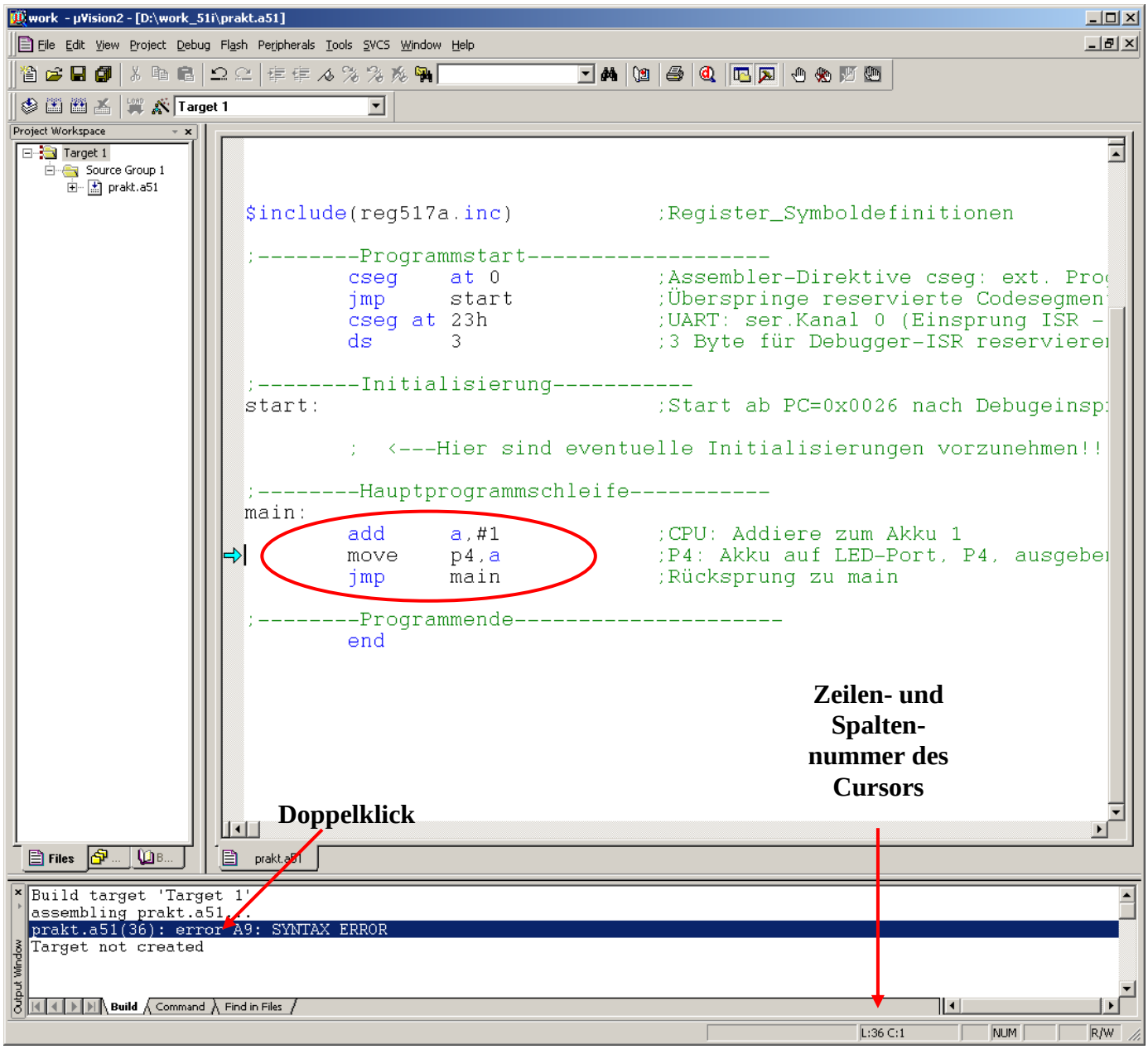

Bild 5 IDE nach einem fehlerhaften Übersetzungsversuch

Vierter Schritt: Anwendung des Remote-Debug-Systems
Diese Methode ermöglicht einen Programmtest mit realer Zielhardware. Dazu wird zusätzlich zum zu testenden Programm ein Monitorprogramm in den Mikrocontroller geladen, das die Kontrolle über eine Schnittstelle (z.B. seriell: RS232) zum Host-PC erhält. Beide Programme teilen sich die Ressourcen des Mikrocontrollers. Deshalb stehen eine Schnittstelle und ein Teil des Speichers nicht für das zu testende Programm zur Verfügung. Jedoch bietet ein derartiges Monitorprogramm gute Debug-Möglichkeiten, wie Software-Breakpoints, Registerzugriff und Einzelschrittausführung. Der

Zugriff des Hostrechners erfolgt im Praktikum mittels Entwicklungsumgebung **µVision2** über ein zusätzliches Bedien-Interface.
Nun soll der Editiermodus der IDE verlassen und der Debug-Modus aufgerufen werden. Drücken Sie dazu nach einem fehlerfreien Übersetzungslauf die Tastenkombination [Strg]+[F5] oder wählen Sie den Menüeintrag **Debug → Start/Stop Debug Session**! Im linken unteren Fensterrand wird nun für kurze Zeit ein blauer Fortschrittsbalken eingeblendet, der das Laden des Maschinencodes vom Host-PC in den Mikrokontroller widerspiegelt. Bild 6 zeigt, wie im Debug-Modus mit der Entwicklungsumgebung gearbeitet werden kann: Mit Taste [F5] bzw. dem Menüeintrag **Debug → Go** wird der Maschinencode ausgeführt und mit **Debug → Stop Running** wieder gestoppt. Während das Programm abläuft, leuchten zunächst alle LEDs scheinbar gleichzeitig. Eine Anzeigeunterbrechung ist nicht zu erkennen.

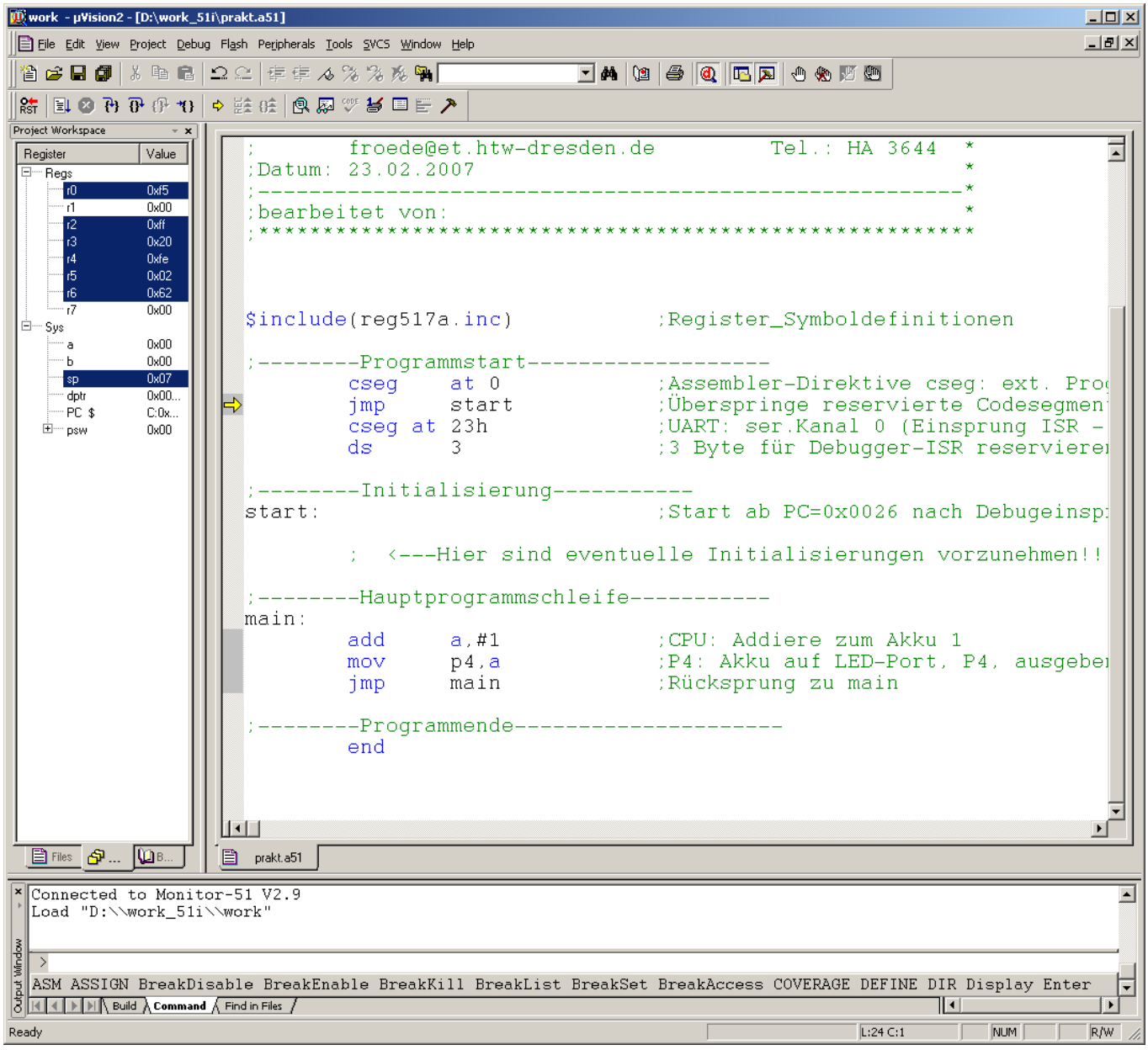

Bild 6 IDE nach dem Wechsel in den Debug-Modus

Betätigen Sie nun am Gehäuse der Mikrocontrollerbaugruppe den Taster links unten. Es werden alle Register rückgesetzt und der Controller nimmt einen definierten Ausgangszustand ein.
Betätigen Sie **Debug → Reset** und führen Sie eine Einzelschrittsteuerung des Debuggers durch, indem Sie die Taste [F10] (bzw. Menüeintrag: **Debug → Step into**) mehrfach nacheinander drücken. Nach dem ersten Tastendruck springt der gelbe Pfeil im linken Rand des Editors zur Zeile

35. Er zeigt immer die Befehlszeile an, die als nächstes ausgeführt wird. Im Project Workspace (linkes Fenster, Bild 6) werden nun die Registerinhalte dargestellt. Der aktuelle Inhalt des Akkumulators ist der hexadezimale Wert **00**.
Nach dem zweiten Betätigen von Taste [F10] steht der gelbe Pfeil in Zeile 36 (siehe Bild 7). Der Befehl **add** wurde also schon bearbeitet, so dass der aktuelle Inhalt des Akkumulators jetzt dem hexadezimalen Wert **01** entspricht. Wie weiterhin aus Bild 7 zu erkennen ist, wird der Befehl **mov** nach einem erneuten Drücken der Taste [F10] bearbeitet.

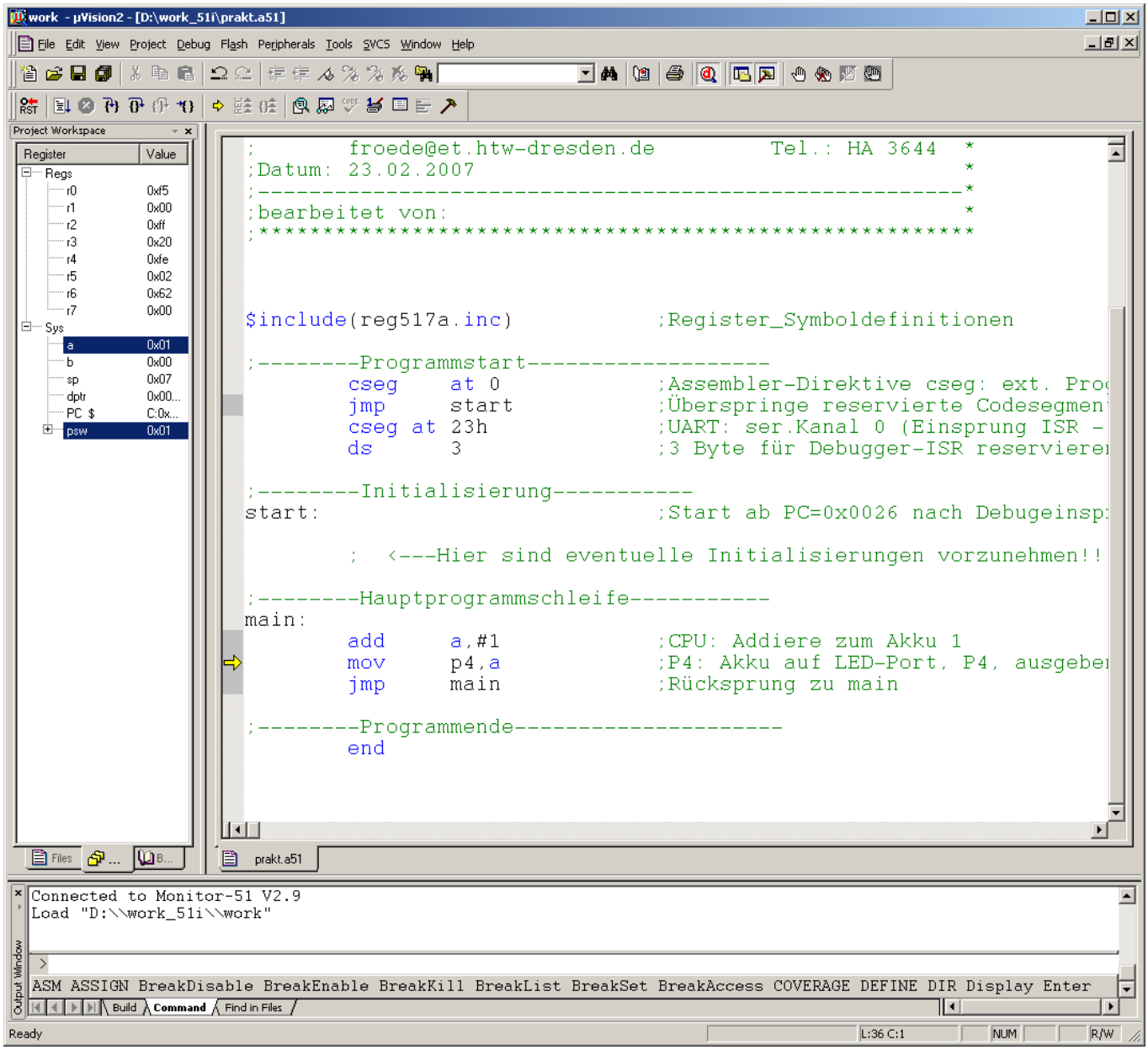

Bild 7 IDE im Debug-Modus nach der zweiten Einzelschrittausführung

Nach dem dritten Einzelschritt steht der Zeiger in Zeile 37 des Editors vor dem Befehl **jmp**. Wie Sie leicht überprüfen können, stellt die LED-Kette den Akku-Inhalt (hexadezimal **01)** dar, d.h. nur die unterste LED leuchtet.
Nach einem weiteren Schritt (Taste [F10]) ergibt sich der Zustand, wie in Bild 8 gezeigt. Der Zeiger steht erneut auf Zeile 35 aber diesmal im zweiten Schleifendurchlauf.

Nachdem **add** ausgeführt wurde (s. Bild 8), steht im Akku jetzt der hexadezimale Wert **02**. Kontrollieren Sie die Ausgabe am LED-Modul nach weiteren Programmschritten.

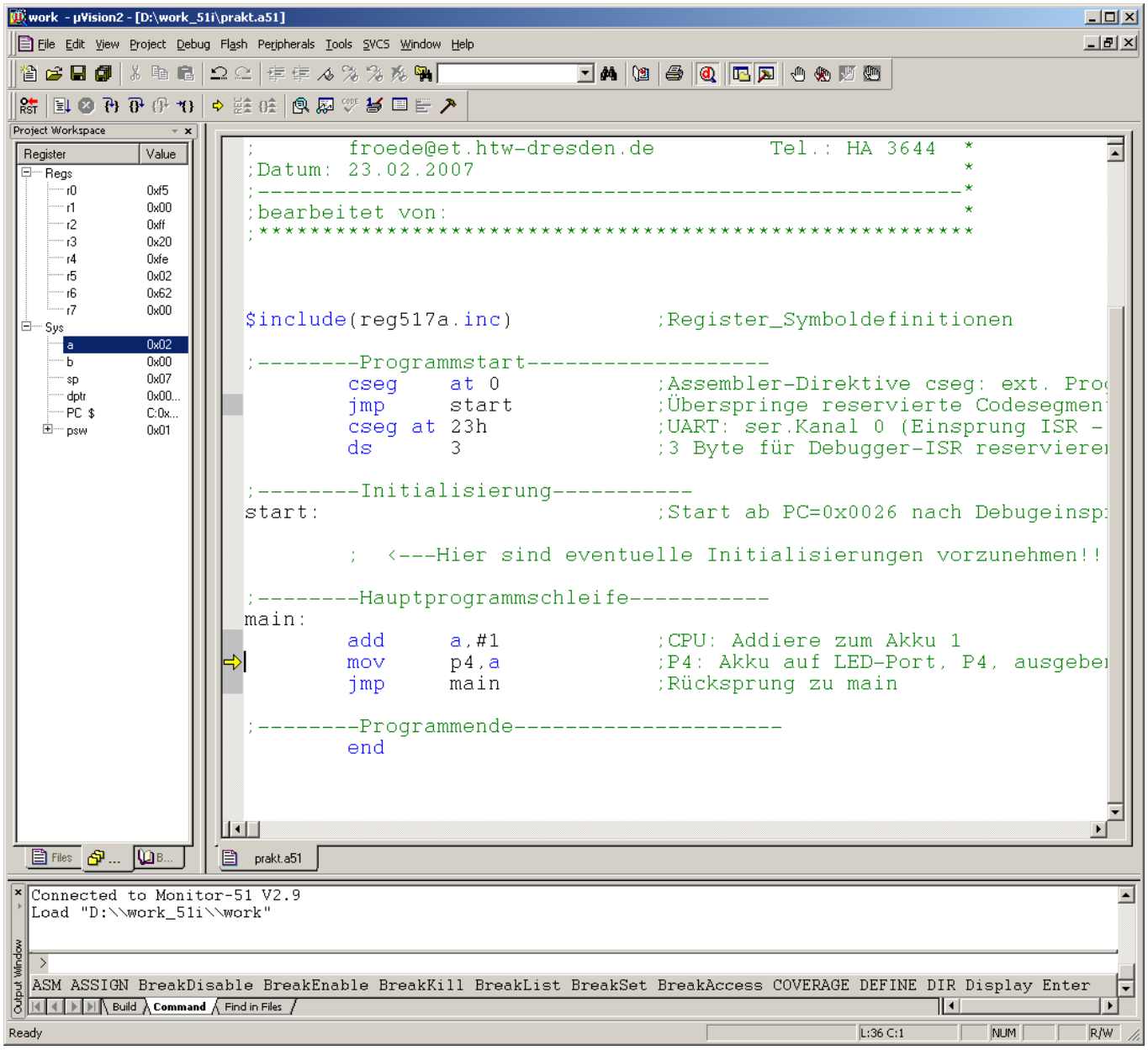

Bild 8 IDE im Debug-Modus nach der fünften Einzelschrittausführung

Wie anhand der Einzelschrittausführung festzustellen ist, arbeitet der Mikrocontroller zwar so, wie im Programm vorgeschrieben, jedoch ist bei einem Programmlauf ohne Schrittbetrieb aufgrund der Taktrate des Controllers nicht zu erkennen, wann welche LED leuchtet. Fügen Sie deshalb folgende Zeilen zwischen Befehl **mov** und Befehl **jmp** ein:

```
rot1: ;Rotierschleife1
rot2: ;Rotierschleife2
      djnz  r1,rot2
            ;CPU: Erniedrige r1 um 1; springe zu rot2, wenn r1 ungleich 0
      djnz  r0,rot1
            ;CPU: Erniedrige r0 um 1; springe zu rot1, wenn r0 ungleich 0
```

Nach Verlassen des Debug-Modus (**Stop Debug Session**) assemblieren und laden Sie das Projekt erneut, so dass sich die Darstellung wie in Bild 9 ergibt! Wechseln Sie anschließend in den Debug-Modus, bringen Sie mit Taste [F5] bzw. dem Menüeintrag **Debug → Go** das Programm zur Ausführung und beobachten Sie die LED-Kette.

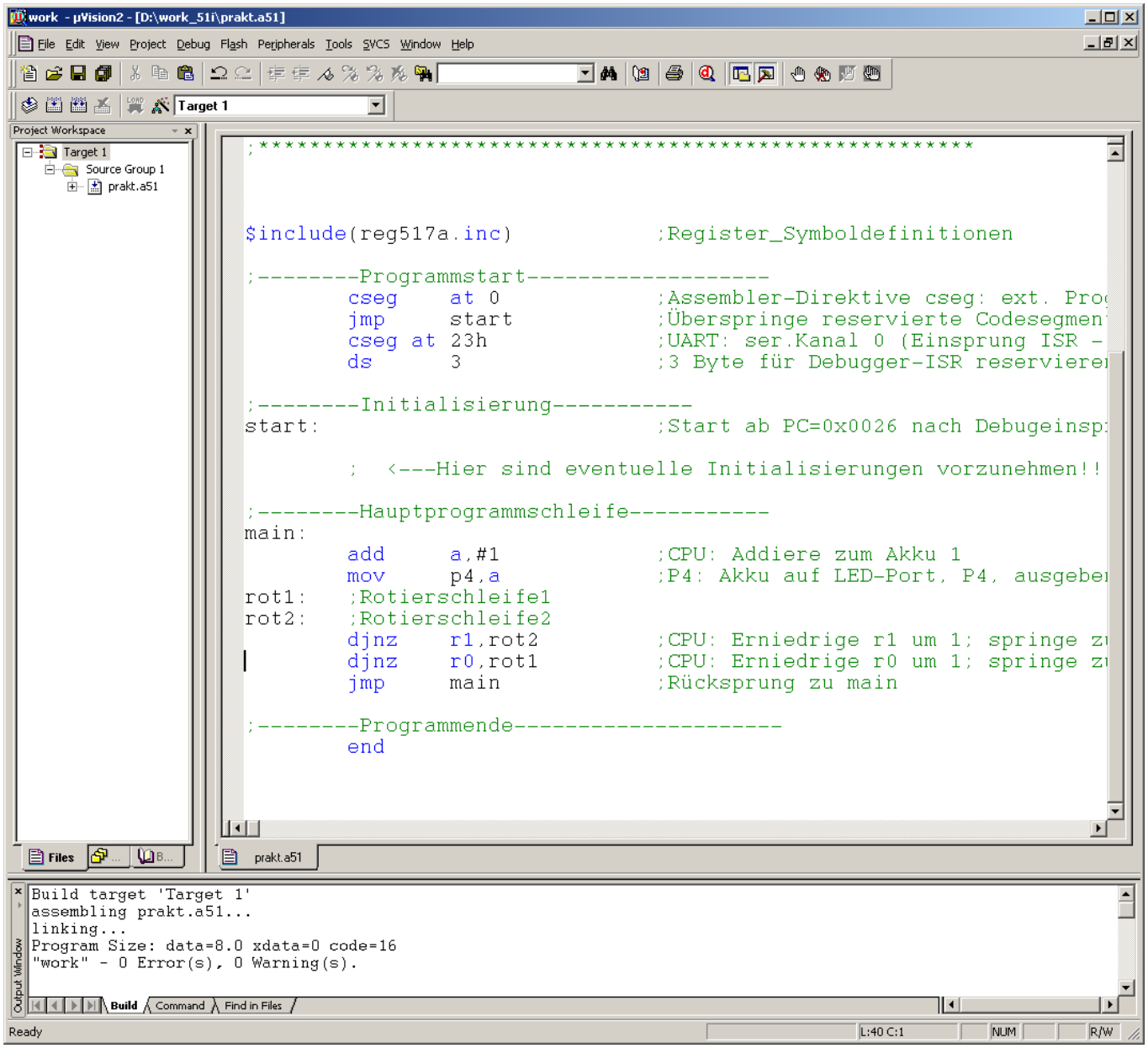

Bild 9 IDE nach Übersetzungslauf mit Programmerweiterung

Was stellen Sie fest? Wie funktioniert der hinzugefügte Code? Nutzen Sie zur Veranschaulichung gegebenenfalls die Einzelschrittausführung!
Ersetzen Sie nun Zeile 40 durch den folgenden „Schreibfehler“:

```
djnz r1,rot1
```

Wie reagiert der Assembler? Bringen Sie auch dieses Programm zur Ausführung auf dem Mikrocontroller, beobachten Sie wiederum die LED-Kette und analysieren Sie das Fehlverhalten.

Weitere Hinweise zur IDE-Bedienung:
Experimentieren Sie mit weiteren Befehlen des Debuggers, die Sie im Menü **Debug** finden. Während des Programmlaufes besteht die Möglichkeit, die Ausführung bei einem bestimmten Befehl mittels Haltepunkt zu unterbrechen. Dazu klicken Sie in die gewünschte Zeile des Editors mit der rechten Maustaste. Wählen Sie im erscheinenden Menü **Insert/Remove Breakpoint**. Der Haltepunkt ist im Editorfenster am linken Rand durch einen roten Punkt gekennzeichnet. Um diesen zu entfernen, wählen Sie wieder **Insert/Remove Breakpoint**.

Zur Darstellung der Signalwerte am Digitalausgang Port 4 nutzen Sie den Menüpunkt **Peripherals → I/O-Ports → Port 4**. Zur Anzeige von Variablen-Werten Ihres Programmcodes muss auf die entsprechende Variable mit der rechten Maustaste geklickt werden. Im erscheinenden Menü ist **Add „Variable“ to Watch Window... → #1** bzw. **#2** zu wählen. Um ein **Watch Window** darzustellen, ist der Menüpunkt **View → Watch & Call Stack Window** zu aktivieren. Danach erscheint im **Output Window** ein neues Fenster, in dem der Reiter **Watch #1** bzw. **Watch #2** zu wählen ist.

Den integrierten **Simulator** können Sie auch ohne die im Labor Mikrorechentechnik eingesetzte Keil-Prototypenkarte zur Unterstützung Ihrer Programmentwicklung einsetzen. Wählen Sie dazu unter **Project → Options for Target → Debug → Use Simulator.**

Um eine Sitzung zu beenden, wählen Sie
Debug → Stop Debug Session und **Project → Close Project** sowie **File → exit**.

Quellenverzeichnis

[1] Müller, H.; Walz, L.: Mikroprozessortechnik – Elektronik 5. Vogel Buchverlag, Würzburg, 2005
[2] Walter, J.: Mikrocomputertechnik mit der 8051-Controller-Familie. Springer-Verlag Berlin Heidelberg, 2. Auflage, 1996
[3] Handbuch - SAB 80C537
[4] Kurzdokumentation: Laborpraktikum Mikrorechentechnik, Mikrocontroller SAB 80C517A/80C537
[5] Wikipedia: http://de.wikipedia.org/wiki/Entwicklungsumgebung

Abkürzungsverzeichnis

ADU	Analog-Digital-Umsetzer
AE	Analogeingang
DA	Digitalausgang
IDE	Integrated Development Environment
ISR	Interrupt Service Routine
LED	Light Emitting Diode
UART	Universal Asynchronous Receiver Transmitter

Erarbeitet: Versuchsanleitung_MPT-Inf_3.1.doc, Prof. Dr.-Ing. K. Feske, M.Sc. A. Fröde, 01.07.2010