AVR-8-bit-Mikrocontroller
Gruppe 300 - Arbeiten mit AVR-Assembler
Teil 301 - Einführung

Teil 301 - Einführung

Teil 302 - Syntax der Assembler-Sprache

Teil 303 - Instruktionen

Teil 304 - Register

Teil 305 - I/O-Ports

Teil 306 - SRAM

Teil 307 - Programmablauf

Teil 308 - Zahlendarstellung

Teil 309 - Bits, Bytes und Zahlen

Teil 310 - Anhang und Beispiele

1 Eine Einführung in die Assembler-Sprache

1.1 Warum Assembler ?

Warum soll man Programme für die Mikrocontroller noch in Assembler schreiben, wo es doch so einen tollen kostenlosen C-Compiler und einen billigen Basic-Compiler gibt?

Assembler oder Hochsprache, das ist hier die Frage. Warum soll man noch eine neue Sprache lernen, wenn man schon welche kann?

Assembler ist für den Einstieg sehr gut geeignet. Nur wenn man Assembler anwendet, lernt man den Aufbau eines Prozessors von der Pike auf kennen und kann ihn dadurch besser nutzen. Außerdem stößt man bei jedem Compiler irgendwann mal auf Probleme, die sich besser durch das Verwenden von Assemblercode lösen lassen. Der umgekehrte Fall kann aber auch auf Compiler zutreffen, wenn sehr komplexe Funktionen verwendet werden müssen. Ggf. hilft ein Mix.

1.2 In der Kürze liegt die Würze - Schnell wie der Wind

Assembler-Instruktionen (Befehle) übersetzen sich 1 zu 1 in Maschinenbefehle. Auf diese Weise macht der Prozessor wirklich nur das, was für den angepeilten Zweck tatsächlich erforderlich ist und was der Programmierer auch ausdrücklich will. Keine extra Schleifen und unnötige Features blasen den ausgeführten Code auf. Wenn es bei begrenztem Programm-Speicher und komplexerem Programm auf jedes Byte ankommt, dann ist Assembler sowieso Pflicht. Kürzere Programme lassen sich wegen schlankerem Maschinencode leichter entwanzen, weil jeder einzelne Schritt Sinn machen und zur Aufmerksamkeit zwingen soll.

Da kein unnötiger Code ausgeführt wird, sind Assembler-Programme maximal schnell. Jeder Schritt ist von voraussehbarer Dauer. Bei zeitkritischen Anwendungen, wie z.B. bei Zeitmessungen ohne Hardware-Timer, die bis an die Grenzen der Leistungsfähigkeit des Prozessors gehen sollen, ist Assembler ebenfalls zwingend. Soll es gemütlich zugehen, kann man programmieren wie man will.

Es ist ein Vorurteil, dass Assemblersprache komplizierter und schwerer erlernbar sei als eine Hochsprache. Das Erlernen einer Assemblersprache macht mit den wichtigsten Grundkonzepten und vor allem mit der Hardware vertraut, was bei Mikrocontrollern auch auf die Anwendung der Hochsprachen zutrifft.

Der erste Code sieht nicht sehr elegant aus, mit jedem Programm sieht das schon besser aus. Da viele Features prozessorabhängig sind, ist Optimierung eine reine Übungsangelegenheit und nur von der Vertrautheit mit der Hardware abhängig. Die ersten Schritte fallen in jeder neu erlernten Sprache nicht leicht und nach wenigen Wochen lächelt man über die Holprigkeit und Umständlichkeit seiner ersten Gehversuche. Manche Assembler-Instruktionen lernt man eben erst nach Monaten richtig nutzen.

Allerdings muss auch erwähnt werden, dass das Programmieren in Assembler besonders fehleranfällig ist und dass es damit besonders lange dauert, bis das Programm erste Ergebnisse liefert. Assembler-Programme sind gnadenlos, weil sie davon ausgehen, dass der Programmierer jeden Schritt mit Absicht so und nicht anders macht. Alle Schutzmechanismen muss man sich selber ausdenken und auch programmieren, die Maschine macht bedenkenlos jeden Unsinn mit. Kein Fensterchen warnt vor ominösen Schutzverletzungen, es sei denn man hat das Fenster selber programmiert.

Denkfehler beim Konstruieren sind aber genauso schwer aufzudecken wie bei Hochsprachen. Das Ausprobieren ist bei den AVR-Mikrocontrollern aber sehr leicht, da der Code rasch um einige wenige Diagnostikzeilen ergänzt und mal eben in den Chip programmiert werden kann. Vorbei die Zeiten mit EPROM löschen, programmieren, einsetzen, versagen und wieder von vorne nachdenken. Änderungen sind schnell gemacht, kompiliert und entweder im AVR Studio simuliert, auf dem Testboard ausprobiert und/oder in der realen Schaltung einprogrammiert, ohne dass sich ein IC-Fuß verbogen oder die UV-Lampe gerade im letzten Moment vor der großen Erleuchtung den Geist aufgegeben hat.

1.3 Grundausstattung und Installation der Werkzeuge

AVR-8-bit-Mikrocontroller
Gruppe 300 - Arbeiten mit AVR-Assembler
Teil 301 - Einführung

Die Details werden in der **Gruppe 200: Einsetzen von AVR-Tools** ausführlich behandelt, so dass hier der Verweis genügt:

Teil 201 - Experimentierboards

Teil 202 - ISP-Programmieradapter

Teil 203 - AVR-ALE-Testboard

Teil 204 - AVR Studio

Teil 205 - Assembler und AVR Studio

Teil 206 - C-Compiler und AVR Studio - hier nicht relevant

Teil 207 - Editor - UltraEdit