

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



## **Teil 301 - Einführung**

### **1 Eine Einführung in die Assembler-Sprache**

#### **1.1 Warum Assembler ?**

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

#### **1.3 Grundausstattung und Installation der Werkzeuge**

## **Teil 302 - Syntax der Assembler-Sprache**

### **2 Struktur und Syntax der Assembler-Sprache**

#### **2.1 Struktur der Assembler-Zeilen**

#### **2.2 Struktur der Programm-Datei**

#### **2.3 Kommentare**

#### **2.4 Kopf des Programms**

#### **2.5 Assembler-Direktive (Directivs)**

#### **2.6 Assembler-Ausdrücke (Expressions)**

##### **2.6.1 Operanden**

##### **2.6.2 Operatoren**

##### **2.6.3 Funktionen**

##### **2.6.4 Operationen**

#### **2.7 Daten- und Adress-Struktur**

##### **2.7.1 Nomenklatur: Abkürzungen für Register und Operanden**

##### **2.7.2 AVR-Speicher-Räume aus Assembler-Sicht**

###### **2.7.2.1 Programm-Speicher**

###### **2.7.2.2 Daten-Speicher**

###### **2.7.2.3 EEPROM-Speicher**

##### **2.7.3 Adress-Modi**

###### **2.7.3.1 Direkte Adressierung eines einzelnen Arbeits-Registers**

###### **2.7.3.2 Direkte Adressierung von 2 Arbeits-Registern**

###### **2.7.3.3 Direkte Adressierung eines I/O-Registers**

###### **2.7.3.4 Direkte Daten-Adressierung im SRAM**

###### **2.7.3.5 Indirekte Daten-Adressierung mit Displacement**

###### **2.7.3.6 Indirekte Daten-Adressierung**

###### **2.7.3.7 Indirekte Daten-Adressierung mit Pre-Decrement**

###### **2.7.3.8 Indirekte Daten-Adressierung mit Post-Decrement**

###### **2.7.3.9 Adressierung des Programm-Speichers**

###### **2.7.3.10 Adressierung des Programm-Speichers mit Post-Increment**

###### **2.7.3.11 Direkte Adressierung des Programm-Speichers**

###### **2.7.3.12 Indirekte Adressierung des Programm-Speichers**

###### **2.7.3.13 Relative Adressierung des Programm-Speichers**

## **Teil 303 - Instruktionen**

### **3 Mnemotechnische Abkürzungen der Instruktionen**

#### **3.1 Arithmetische und logische Instruktionen**

#### **3.2 Verzweigungs-Instruktionen**

#### **3.3 Instruktionen für den Daten-Transfer**

#### **3.4 Bit- und Bit-Test-Instruktionen**

## **Teil 304 - Register**

### **4 Register-Datei**

#### **4.1 Arbeits-Register**

#### **4.2 Unterschiede der Arbeits-Register**

#### **4.3 Pointer-Register**

#### **4.4 Empfehlungen zu Registerwahl**

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

### **Teil 305 - I/O-Ports**

- 5 Die Ein-/Ausgabe-Ports der AVR's
  - 5.1 Die Ports der AVR's
  - 5.2 Die Ports als I/O-Register
  - 5.3 Das Status-Register SREG

### **Teil 306 - SRAM**

- 6 Statisches RAM - SRAM
  - 6.1 Beschreibung
  - 6.2 Verwendung von SRAM
  - 6.3 Zugriff auf das SRAM
  - 6.4 Verwendung des SRAM als Stack
  - 6.5 Einrichtung des Stapels
  - 6.6 Verwendung des Stapels
  - 6.7 Fehlermöglichkeiten beim (Hoch-)Stapeln

### **Teil 307 - Programmablauf**

- 7 Steuerung des Programmablaufs
  - 7.1 Der Reset
  - 7.2 Linearer Programmablauf und Verzweigungen
  - 7.3 Zeitzusammenhänge beim Programmablauf
  - 7.4 Makros im Programmablauf
  - 7.5 Unterprogramme (Subroutines)
  - 7.6 Interrupts im Programmablauf

### **Teil 308 - Zahlendarstellung**

- 8 Zahlendarstellungen im Assembler
  - 8.1 Dual-Zahlen
  - 8.2 Hexadezimal-Zahlen im ASCII-Format
  - 8.3 Oktal-Zahlen
  - 8.4 BCD-Zahlen (binär kodierte Dezimal-Zahlen)
  - 8.5 Zahlen im ASCII-Format
  - 8.6 Gleitkomma-Zahlen

### **Teil 309 - Bits, Bytes und Zahlen**

- 9 Umwandlungen von Bits, Bytes und Zahlen
  - 9.1 Bitmanipulationen, Schieben und Rotieren
  - 9.2 Umwandlung einer Dual-Zahl in eine ungepackte BCD-Zahl
  - 9.3 Umwandlung einer BCD-Zahl in eine Dual-Zahl
  - 9.4 Umwandlung einer Dual-Zahl in eine Dezimal-Zahl im ASCII-Format
    - 9.4.1 Lösung 1
    - 9.4.2 Lösung 2
    - 9.4.3 Lösung 3
  - 9.5 Umwandlung einer Dezimal-Zahl im ASCII-Format in eine Dual-Zahl
  - 9.6 Umwandlung einer Dual-Zahl in eine Hexadezimal-Zahl im ASCII-Format
  - 9.7 Umwandlung einer Hexadezimal-Zahl im ASCII-Format in eine Dual-Zahl
  - 9.8 Lineare Umwandlung

### **Teil 310 - Anhang und Beispiele**

- 10. Anhang und Beispiele
  - 10.1 Rechnen in Assembler
    - 10.1.1 Addition, Subtraktion und Vergleich
    - 10.1.2 Multiplikation
    - 10.1.3 Division

## 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 schlankem 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 Grundausrüstung 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**