

# Einführung in Assembler

für 8051- $\mu$ Controller

*von Dominik Strauß*  
<[www.n3or.de](http://www.n3or.de)>

# AUFBAU EINES $\mu$ CONTROLLERS

- Wichtige Bestandteile eines  $\mu$ Controllers
  - CPU
    - Programmzähler
    - Akkumulator
    - Befehlswerk
    - ALU
    - Register
  - ROM / (interner und externer) RAM / SFRs
  - Stack
  - Peripherie/Timer/Serielle Schnittstelle/..
  - Interrupt Controller

## ASSEMBLER

- „Repräsentiert die Maschinsprache in einer vom Menschen lesbaren Form“
  - *mov a, #12* anstelle von *74 0c*
    - Mnemonic anstelle von Opcodes
- 1:1 Umwandlung von Mnemonic in Opcodes durch Compiler (z.B.  $\mu$ Vision von Keil)
- Opcodes werden vom Befehlswerk interpretiert/ausgeführt
- Assemblerprogramm gespeichert im (EEP)ROM
- Programmzähler gibt den auszuführenden Opcode an
- Opcodes, Einsprungadressen etc. können einer Formelsammlung entnommen werden

# ASSEMBLERPROGRAMME

- Startadresse immer 0h
- Bestimmte Speicherbereiche sollten nicht verwendet werden (da z. B. Einsprungadressen für Interrupts)  
→ bei 0h sollte deshalb ein Sprung zum Hauptprogramm stehen
- Programm muss eine Endlosschleife beinhalten  
→ ansonsten unvorhersehbare Terminierung
- Kombination von Mnemonics und Labels ermöglicht Kontrollstrukturen, wie *if-else* oder *switch-case*
- Erstellung von Arrays sind auch möglich
- Mathematische und logische Operationen (wie inc, dec, add, subb, mul, div, anl, orl, rr, rl) sind im Akkumulator möglich

## TIMER

- Benötigt für Zeitfunktionen (Timer) oder zum Zählen von Signalen (Counter)
- Auf 8051- $\mu$ Controllern i.d.R. 2 Standardzähler
- Besitzen mehrere Modi, wie
  - 16-Bit-Modus
  - 8-Bit-Modus mit Autoreload
- Zählen nach oben
- Zählweite kann durch Startwert beschränkt werden
- Der Timer wird jede  $\mu$ Sekunde inkrementiert

## INTERRUPTS

- Programmunterbrechung zugunsten der Verarbeitung eines Events; nach der Abarbeitung wird an der letzten Stelle fortgesetzt
- Folgende Ereignisse können einen Interrupt auslösen
  - Überlauf eines Timers
  - Externe Eingabe
  - Serielle Schnittstelle
- Benötigen eine globale Freigabe *setb ea*
- Benötigen eine individuelle Freigabe
- Interrupt ausgelöst: Programmzähler wird auf eine spezifische Einsprungadresse gesetzt; *reti* führt das Programm an der unterbrochenen Stelle fort

## BEISPIEL: EINFACHE ASSEMBLERANWENDUNG

- Timer (Timer0; 16-Bit-Modus)
- Kontrollstruktur (if-else)
- Interrupt
- Rotation im Akkumulator
- Abfragen eines Schalters
- „Steuerung“ externer Peripherie

# VORTEILE VON ASM GEGENÜBER HOCHSPRACHEN

- Wenig Ressourcen benötigt  
→ gut geeignet für Embedded-Geräte
- Besseres Verständnis für die Funktionsweise der Einzelkomponenten und deren Zusammenspiel
- Echtzeitanwendungen möglich
- Kein Overhead durch den Compiler
- Debugging: Verhalten im kleinsten Detail nachvollziehbar

# NACHTEILE VON ASM GEGENÜBER HOCHSPRACHEN

- Code großer Anwendungen schlecht wartbar und unübersichtlich
- Alle Funktionen müssen von Grund auf implementiert werden (I/O, mathematische Funktionen [sin, cos..]..)
- Viele Aufgabenstellungen können fast nicht mit purem Assembler umgesetzt werden (z. B. grafische Benutzeroberflächen)
- Keine (komplexen) Datentypen und Datenstrukturen

# NUTZEN VON ASSEMBLER FÜR ENTWICKLER

- Flaschenhalse können mit Assembler beseitigt werden  
→ siehe Spieleentwicklung
- Besseres Gefühl für Ressourcen  
→ weniger Speicherverschwendung möglich  
→ bessere Performance möglich  
da besseres Verständnis für die zugrundeliegende Hardware
- Analyse von Laufzeiten fällt einfacher
- Wechsel auf Assembler erfordert umdenken  
→ trainiert das Lösen von Problemen aus einer anderen Perspektive

Noch Fragen?