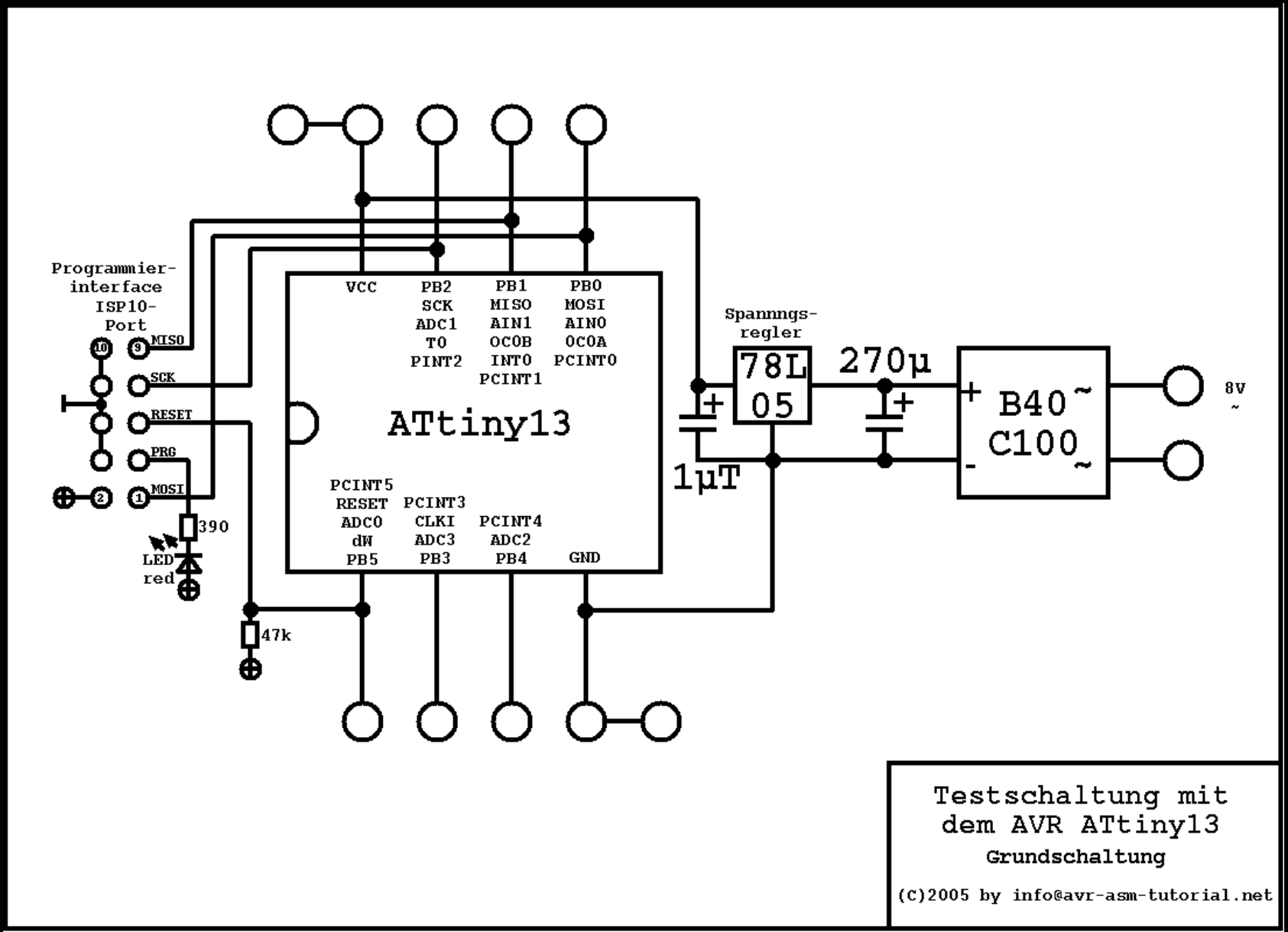
Programmierung von ATMEL AVR Mikroprozessoren am Beispiel des ATtiny13

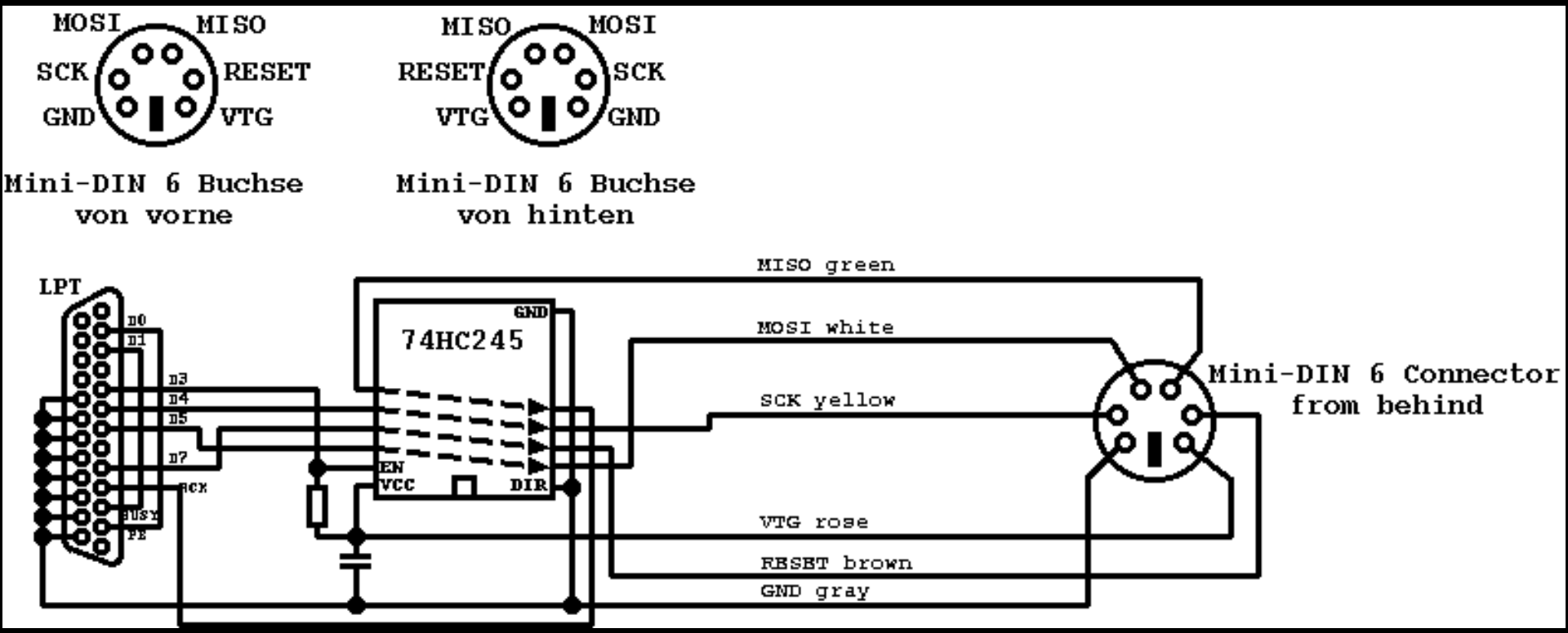
Eine Einführung in Aufbau, Funktionsweise, Programmierung und Nutzen von Mikroprozessoren

Teil IV: Programmieren an Beispielen

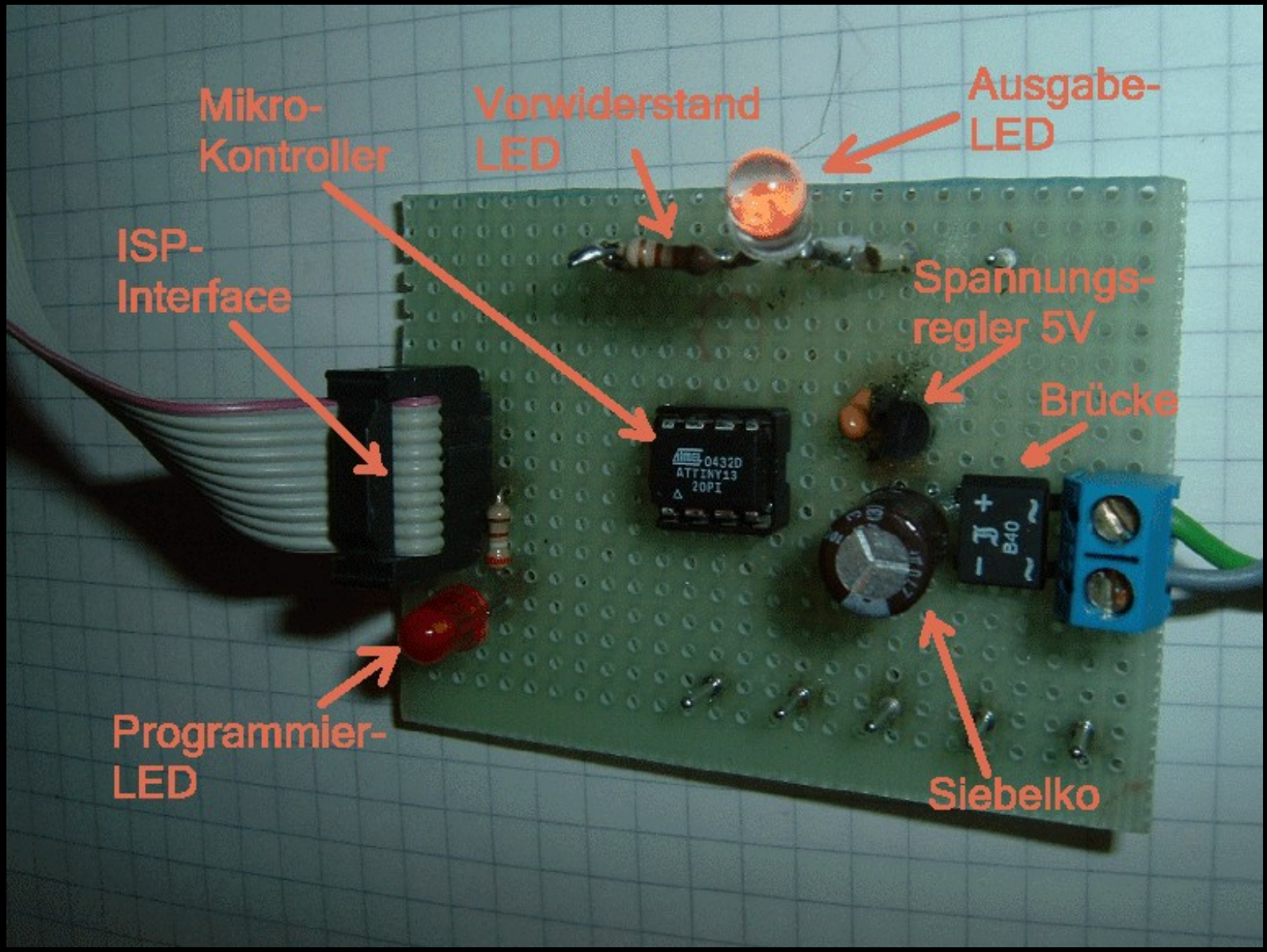
Die Test-Hardware



Das Parallelport-Interface



Testhardware

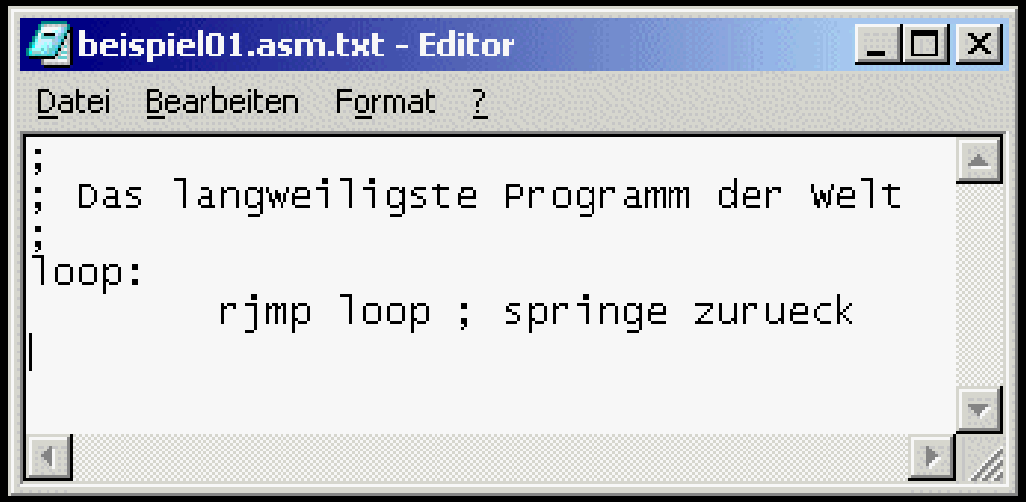


Überblick: Programmieren für Anfänger

1. **Schreiben des Programmcodes mit einem Editor ergibt eine Textdatei**
2. **Übersetzen des Programmcodes mit einem Assembler ergibt ein Programmlisting und eine Hexdatei mit den Maschinenbefehlen**
3. **Einlesen der Hexdatei und Einbrennen der Maschinenbefehle in den Programmspeicher mittels eines Brennprogramms und eines Programmier-Interfaces**
4. **AVR startet neu und führt die Maschinenbefehle aus**

Programmierbeispiel 01: Die unendliche Schleife

Man nehme: einen Editor (z.B. M$: Notepad, Linux: KEdit) und tippe ein:

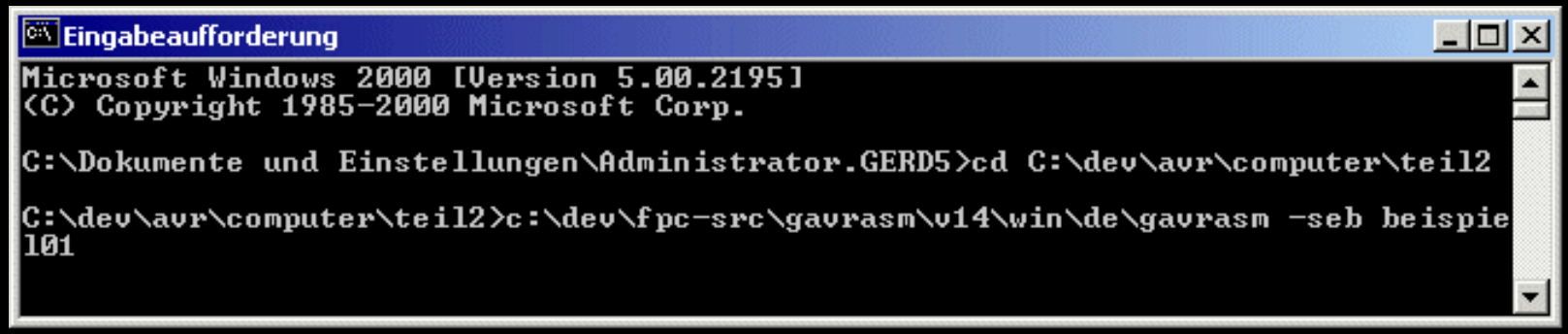


- **Zeilen mit einem Semikolon werden von da an ignoriert (Kommentare)**
- **„loop:“ ist eine Sprungmarke (engl. Label), es dient der Kennzeichnung für den Assembler, Benennung beliebig, immer Doppelpunkt am Ende**
- **„rjmp“ ist eine Instruktion, sie erzeugt beim Übersetzen ein Befehlswort**
- **„rjmp loop“ bewirkt, dass der Prozessor um einen Befehl rückwärts springt (unbedingter Sprung)**
- **Das macht er (und nichts anderes), solange er unter Spannung steht**

Übersetzen des Programms mit gavrasm

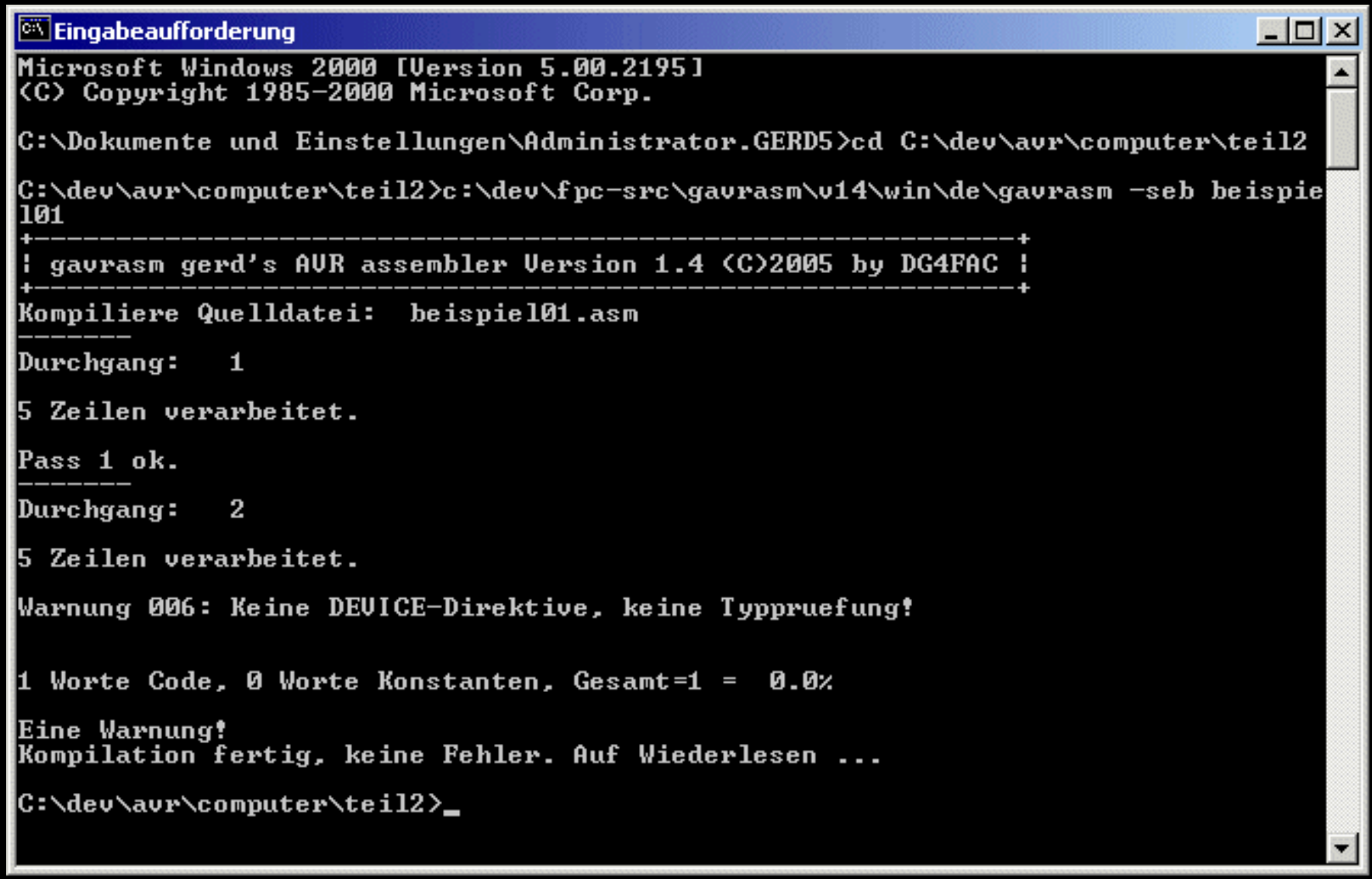
- Man nehme: einen Assembler (wie z.B. meinen eigenen, genannt gavrasm)
- Man öffne: ein Befehlszeilenfenster (M$: Eingabeaufforderung, Linux: bash) und tippe ein (Pfade natürlich entsprechend der eigenen Pfade):

- Dann wird der Assembler aufgerufen mit:



- „Pfad\gavrasm“ ruft den Assembler auf, „beispiel01[.asm]“ ist die Quelldatei
- -seb bedeutet: s=erzeuge eine Symbolliste, e=gib erweiterte Errormeldungen aus, b=gib Kommentare für Beginner aus

Hier spricht der Assembler

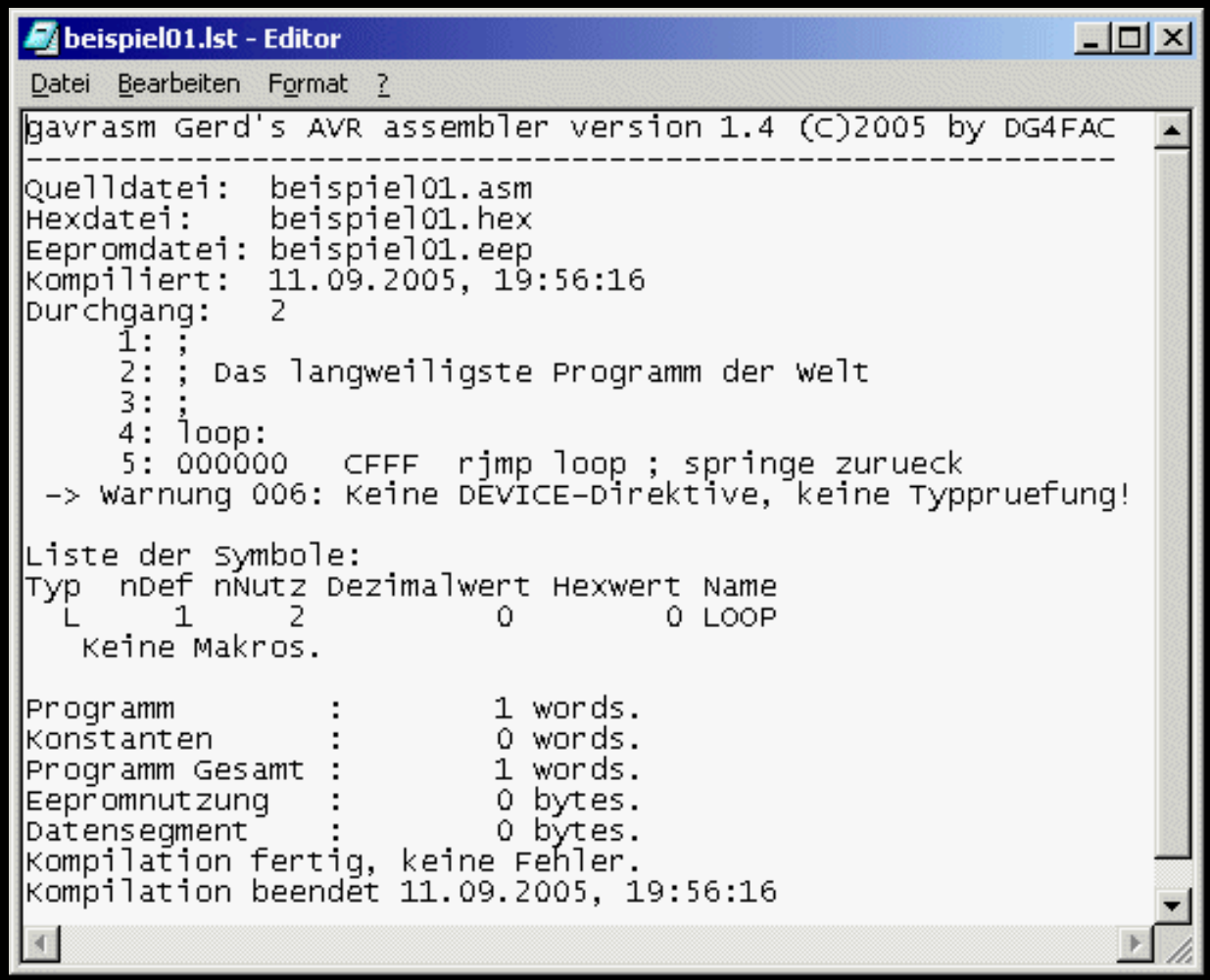


Was will er uns sagen?

- Dabei bedeutet:
- „gavrasm gerd's AVR assembler Version 1.4 (C)2005 by DG4FAC“: die Version des Assemblers
- „Kompiliere Quelldatei: beispiel01.asm“: die assemblierte Textdatei
- „Durchgang: 1, 5 Zeilen verarbeitet.“: der Assembler hat alle Instruktionen verstanden, kein Fehler ist aufgetreten.
- „Durchgang: 2, 5 Zeilen verarbeitet.“: der Assembler hat beim zweiten Durchgang auch alle Sprungmarken, Konstanten und Definitionen komplett verstanden und gefunden.
- „Warnung 006: Keine DEVICE-Direktive, keine Typprüfung!“: es wurde kein spezifischer AVR-Typ mit der DEVICE-Direktive festgelegt, deshalb wurde nicht überprüft, ob die verwendeten Instruktionen bei diesem AVR-Typ zulässig sind.
- „1 Worte Code, 0 Worte Konstanten, Gesamt=1 = 0,0%“: Es wurde eine 16-Bit-Instruktion (ein Befehlswort) erzeugt, im Programm werden keine Tabellen mit Konstanten verwendet, und insgesamt ist der verfügbare Flash-Speicher zu 0,0% durch den Code belegt.

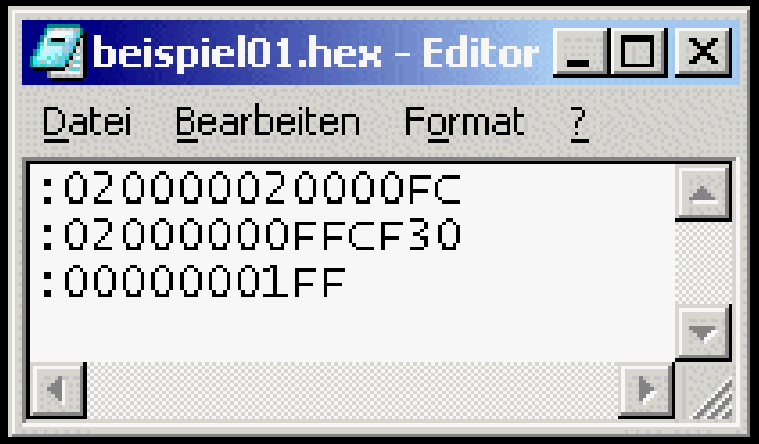
Wer‘s schriftlich lieber mag ...

- In dem Verzeichnis befinden sich jetzt zwei neue Dateien: „beispiel01.lst“ und „beispiel01.hex“.
- beispiel01.lst“: Das Listing beim Übersetzen:



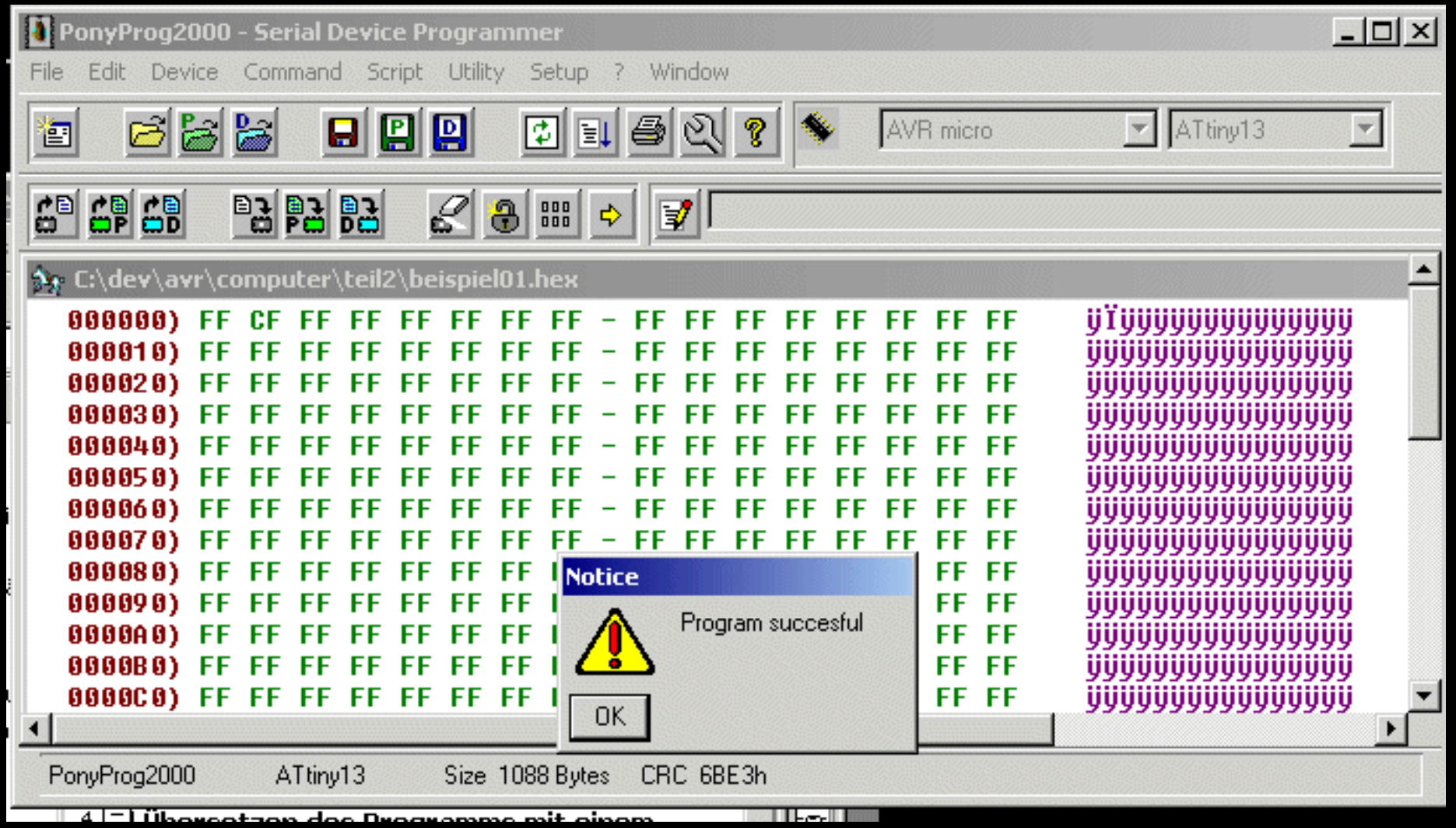
Kryptisch: Maschinencode in Hex

- Der einzige Maschinenbefehl an Adresse 000000 lautet also „CFFF“
- Die Hex-Datei enthält die Maschinenbefehle in folgendem Format:

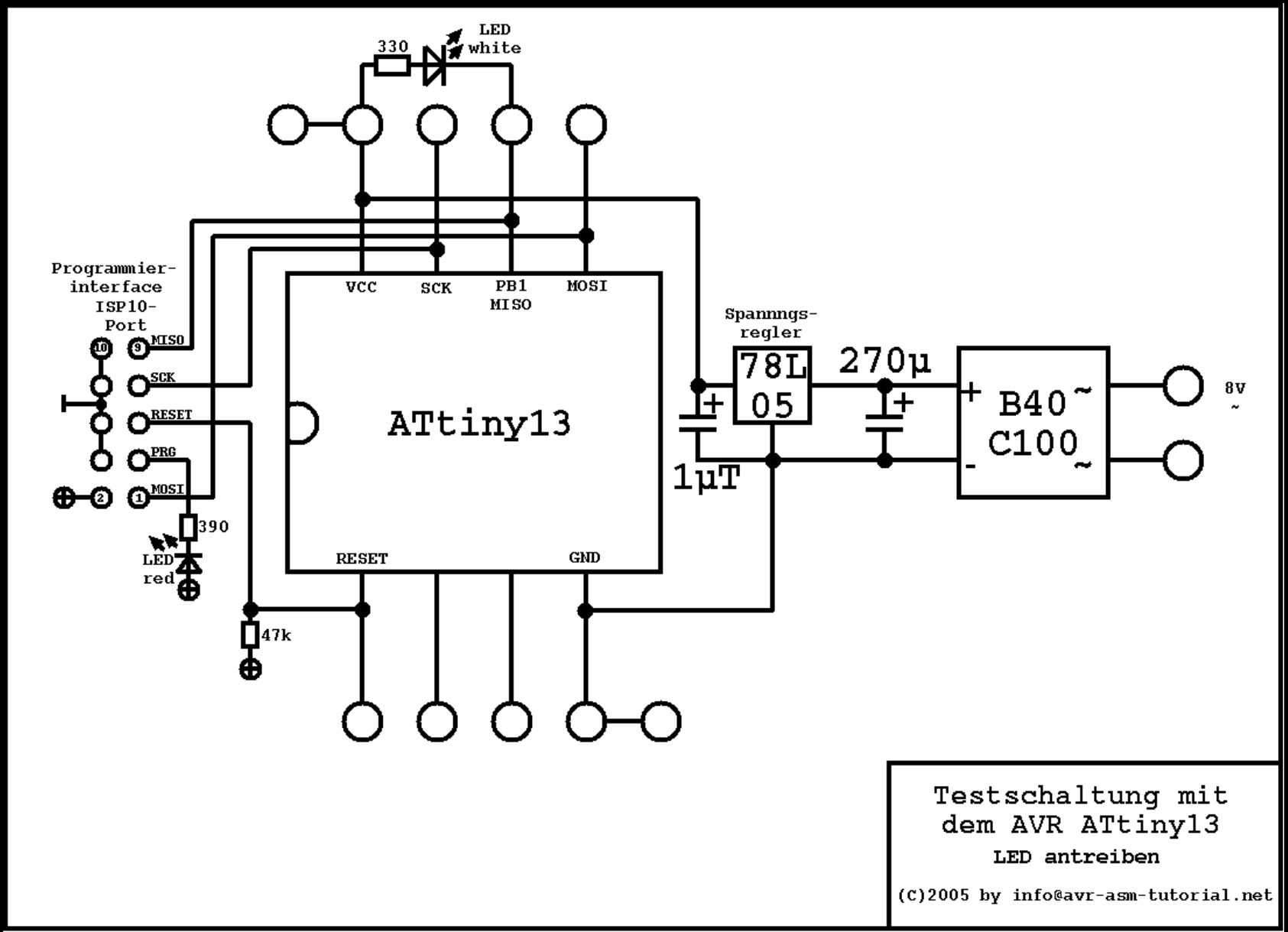


- Diese Datei enthält alle nötigen Daten für die Programmierung in den Flash-Speicher des Chips plus Adressangaben und Prüfbytes.
- Sie wird vom Brennprogramm eingelesen, ausgewertet und der enthaltene Maschinencode (hier: CFFF) in den Flash-Speicher übertragen.
- Als Beispiel für ein Brennprogramm wird PonyProg2000 verwendet.

Burn it in ...


PonyProg2000 - Serial Device Programmer
File Edit Device Command Script Utility Setup ? Window
AVR micro
ATtiny13
C:\dev\avr\computer\teil2\beispiel01.hex
000000) FF CF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000010) FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000020) FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000030) FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000040) FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000050) FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000060) FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000070) FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000080) FF FF FF FF FF FF FF FF FF FF
000090) FF FF FF FF FF FF FF FF FF FF
0000A0) FF FF FF FF FF FF FF FF FF FF
0000B0) FF FF FF FF FF FF FF FF FF FF
0000C0) FF FF FF FF FF FF FF FF FF FF
Notice
Program succesful
OK
PonyProg2000 ATtiny13 Size 1088 Bytes CRC 6BE3h

Beispiel 02: Die Test-Hardware



Beispiel 02: LED an

bsp02_led.asm - Editor

Datei Bearbeiten Format ?

```
; *********************************************************
; * Das zweitlangweiligste Programm der Welt: LED an!     *
; * (C)2005 by info@avr-asm-tutorial.net                  *
; *********************************************************
;
.INCLUDE "tn13def.inc"
;
; Schaltbild:
;                   ATMEL ATtiny13
;                     ___   ____
;            ___    1/   |_|    |8
;+5 Volt o--|___|----|Res    Vcc|----o + 5 Volt
;                    |          |
;                    |PB3    PB2|
;                    |          |6  ___
;                    |PB4    PB1|--|___|--|<|--o + 5 Volt
;                   4|          |   330   LED
;        0 Volt o----|Gnd    PB0|
;                    |__________|
;
;
;
        sbi DDRB,1 ; Bit 1 Port B ist Ausgang
        cbi PORTB,1 ; Ausgang auf logisch 0
loop:
        rjmp loop
;
; End of source code
```

Beispiel 02: LED an, Programmschritte

- **„.INCLUDE „tn13def.inc““:**
  Liest die Prozessor-spezifischen Definitionen des ATtiny13 ein (zulässige Instruktionen, Nummern der I/O-Ports, spezielle Labels, etc.)

- **„sbi DDRB,1“:**
  setze das Bit 1 im Datenrichtungsregister von Port B auf Eins, das macht dieses Portbit (PB1, Pin 6) zu einem Ausgang.

- **„cbi PORTB,1“:**
  setze das Portbit PB1 auf Null, der Ausgang wird damit auf Null Volt gezogen und die LED zieht aus der Versorgungsspannung über den Widerstand Strom und leuchtet.