Hochschule RheinMain
Fachbereich DCSM - Informatik
Marcus Thoss, M.Sc.
Prof. Dr. Robert Kaiser
Christian Neugebauer

Hardwarenahe Programmierung I

WS 2021/22
LV 1512

Übungsblatt 2

In diesem Aufgabenblatt beschäftigen Sie sich intensiv mit `avr-gcc`, einem C-Compiler für Prozessoren der Atmel-AVR-Familie. Sie müssen sich die benötigten Informationen zum Umgang mit diesem Werkzeug teilweise selbst aus der Dokumentation (Manual Pages oder `info`) erarbeiten. Nutzen Sie diese Möglichkeit, um sich in den Selbsthilfe-Modus zu bringen - dieser wird in Zukunft angesichts der Dynamik Ihres Fachs (der Informatik) Ihre Arbeitsweise wesentlich prägen.

Sie benötigen für das Arbeiten mit dem C-Compiler `avr-gcc` und `simulavr` auf Ihrem System gegenüber den früheren Aufgaben zusätzlich das Installationspaket `avr-libc`. Für die `info`-Dokumentation zu `gcc` können Sie außerdem das Paket `gcc-doc` installieren.

Aufgabe 2.1 (Das erste C-Programm):

a) Legen Sie sich ein neues Verzeichnis und darin ein leere Datei namens `simple.c` an.

b) Öffnen Sie die Datei mit einem Texteditor Ihrer Wahl und fügen Sie folgenden C-Quelltext ein:

```
unsigned char i = 0;

int main(void)
{
  i = 10;

  while( i > 0 )
  {
    --i;
  }

  return 0;
}
```

c) Versuchen Sie den Ablauf des Programms zu verstehen und zeichnen Sie dazu ein Flussdiagramm.

d) Nutzen Sie `avr-gcc`, um zunächst aus `simple.c` die Objektdatei `simple.o` zu erzeugen. Geben Sie beim Aufruf zusätzliche sinnvolle Optionen (Flags) an. Sie sollten, wenn Sie die folgenden Hinweise beachten, fünf Optionen und den Eingabedateinamen `simple.c` als Argumente für den Aufruf von `avr-gcc` verwenden!

Hinweise zu Optionen des `avr-gcc`:

- Verhindern Sie, dass `avr-gcc` nach dem Übersetzen in Maschinensprache automatisch den Linker aufruft, um ein vollständiges, ausführbares Programm zu erzeugen (`man gcc`, suchen nach „..do not link“)
- Ermöglichen Sie das spätere Debuggen des Programms. Die `gcc`-Option hierzu haben Sie zuvor schon für `avr-as` verwendet.
- Geben Sie die Ziel-CPU als `atmega16` vor (`man gcc`, Suche nach „MCU type“).
- Reduzieren Sie die Optimierung des Codes durch den Compiler, damit der erzeugte Assemblercode einfacher zu verstehen ist (Option `-O0`, wird später noch vertieft).
- Schließlich: Lassen Sie den Compiler Ihren Code gründlich prüfen und Mängel melden, hierzu schalten wir alle zunächst sinnvollen Warnmeldungen mit `-Wall` ein.
  Verwenden Sie die Option `-Wall` (oder noch intensivere Warnstufen) ab sofort mit **jedem** Compileraufruf im Praktikum!

e) Linken Sie nun `simple.o` als Programm `simple`. Verwenden Sie hierzu nun auch `avr-gcc` (der den in der letzten Übung verwendenten `avr-ld` intern aufruft). Geben Sie dabei wieder passende Optionen (insgesamt drei) an.

Hinweise zu Optionen des `avr-gcc` zum Linken:

- Der Ausgabedateiname soll `simple` sein.
- Ermöglichen Sie auch beim Linken das spätere Debuggen des Programms.
- Geben Sie wieder die Ziel-CPU als `atmega16` vor.

f) Laden Sie nun wieder Ihr Programm mittels `avr-gdb` in einen laufenden `simulavr`. Setzen Sie diesmal nach dem Auswählen des Targets und dem Laden einen Haltepunkt (Breakpoint) auf die Funktion `main` bevor Sie das Programm mit `continue` laufen lassen.

Hinweis: Sie können sich bestehende Haltepunkte durch `info break` anzeigen lassen und mit `delete` wieder löschen.

g) Steppen Sie durch die `main`-Funktion. Lassen Sie sich zwischendurch den Wert der Variablen `i` ausgeben (mit `print`).

h) Sehen Sie sich den aus dem C-Quelltext in `simple` erzeugten Assemblercode an, indem Sie wie zuvor `avr-objdump` verwenden. Rufen Sie dann `gcc` wie in Schritt c), aber zusätzlich mit der Option `-S` auf. Welche Datei wird erzeugt? Sehen Sie sich die neue Datei an. Was bewirkt also `-S`?
Im `avr-gdb` können Sie übrigens mit dem `layout`-Kommando auch zwischen C- und Assembleransicht wechseln! Finden Sie die Befehle, mit denen die Schleife realisiert wird und vergleichen Sie mit der ersten Praktikumsübung.

i) Schreiben Sie ein neues Skript, um Ihre Compiler-Aufrufe zu automatisieren.

Aufgabe 2.2 (Ein komplexeres C-Programm):

a) Laden Sie von der Materialienseite der Lehrveranstaltung das Beispielprogramm `hwp1_p2.c` in ein neues Übungsverzeichnis herunter.

Sehen Sie sich das Programm im Quelltext an, um es in allen Details zu verstehen.

b) Erzeugen Sie wie in der vorigen Übung ein im Simulator ausführbares Programm daraus und testen Sie es mit `simulavr` und `avr-gdb`.

   Hinweis: Im Debugger werden RAM-Adressen (z.B. von globalen Variablen) mit einem Offset von 0x800000 angezeigt, also z.B. die RAM-Adresse 0x100 als 0x800100. Die ist eine interne Besonderheit des `avr-gcc`, die auf die Adressen in Ihrem C-Quelltext keinen Einfluss hat (dort würde nach wie vor 0x100 als Adresse verwendet).

c) Vergleichen Sie `hwp1_p2.c` mit dem Assembler-Quellcode aus den Praktikumsblatt 1. Finden Sie korrespondierende Quellcode-Abschnitte und vergleichen Sie mit dem aus dem C-Programm erzeugten Assembler-Quellcode. `Beispiel:` Erhöhen des Wertes einer globalen Variablen.

d) Versuchen Sie, Ihre Experimente und Erweiterung des Quellcodes der Assembler-Übungen auch auf den C-Quelltext zu übertragen.