Hochschule RheinMain
Fachbereich Design Informatik Medien
Marcus Thoss, M.Sc.

Hardwarenahe Programmierung I
WS 2019/20
LV 1512

Übungsblatt 1

Sie werden in dieser Übung Ihr erstes Assemblerprogramm bearbeiten und starten. Für den Anfang wird das Mikrocontroller-Board, das wir später verwenden, noch durch einen Simulator ersetzt, um schneller zu Erfolgserlebnissen zu gelangen.

Legen Sie für diese und folgende Übungen geeignete Unterverzeichnisse (z.B. `hwp1/blatt1`) in Ihrem Homeverzeichnis an.

Lesen Sie immer zunächst kurz die gesamte Aufgabe durch und bearbeiten Sie dann jeden Übungsschritt gründlich und vollständig. Notieren Sie sich zu jeder Aufgabe und Frage Ihre Lösungsschritte und Antworten!

Aufgabe 1.1 (Das erste Programm):

a) Laden Sie von der Materialienseite der Lehrveranstaltung auf `http://wwwvs.cs.hs-rm.de` das Beispielprogramm `hello_arduino.S` in Ihr neues Übungsverzeichnis herunter.

   Sehen Sie sich den Programmcode mit einem Texteditor Ihrer Wahl (oder notfalls mit `less`) an - Quellcode, ob Assembler oder C, ist zunächst nur eine einfache Textdatei.

   Lesen Sie sich den Programmcode durch; Sie müssen in der Lage sein, alle Kommentare (vorsicht, Englisch!) sprachlich zu verstehen. Trotz mangelnder Assembler-Kenntnisse sollten Sie mit Hilfe der Kommentare die Bedeutung der einzelnen Schritte ebenfalls möglichst genau verstehen. Rechechieren Sie ggf. in der Atmel-Dokumentation, insbesondere der Übersicht der Assemblerbefehle, die auf der Materialienseite verlinkt ist und diskutieren Sie den Code in der Praktikumsgruppe.

   Sie benötigen für die Bearbeitung der Aufgabe ein geöffnetes Shell-Fenster, in dem Sie in Ihr neues Übungsverzeichnes wechseln müssen.

b) Übersetzen Sie nun den Quelltext mittels `avr-as` zu ausführbarem Maschinencode in einem "Object File" (Objektdatei, `hello_arduino.o`) und linken Sie die Objektdatei zu einem ausführbaren Programm (`hello_arduino`):

```
avr-as -g -o hello_arduino.o hello_arduino.S
avr-ld -o hello_arduino hello_arduino.o
```

Diese beiden Schritte, die wir noch genauer betrachten werden, überführen den im Klartext lesbarem Programmcode in `hello_arduino.S` ("human readable") in eine Programmdatei, die dieselben Instruktionen enthält, diese aber binär codiert.

Diese (in `hello_arduino`) sind für Sie zwar dann nicht mehr als Text lesbar (versuchen Sie es!), liegen dafür aber in dem Format vor, mit dem der Prozessor mit Befehlen "gefüttert" wird.

Fragen:

- Um welchen Dateityp handelt es sich bei der Ausgabedatei?
  Hinweis: Mit welchem Linux-Kommando das herauszufinden ist, wissen Sie vielleicht schon aus dem Linux-Vorkurs oder rechechieren Sie, z.B. Selflinux, Kap. "Dateien unter Linux"
- Versuchen Sie, das Programm auf dem PC unter Linux auszuführen. Was geschieht?
- Untersuchen Sie das Programm mit `avr-objdump`. Können Sie den Assemblerquelltext damit rekonstruieren? Wie viele Bytes Programmcode wurden erzeugt?
- Was geschieht, wenn man bei den Aufrufen `-o` weglässt? Hat die Ausgabedatei noch dasselbe Format? Welche Bedeutung hat demnach die Option `-o`?
- Was geschieht, wenn man bei `avr-as -g` weglässt? Finden Sie heraus, wofür diese Option steht.

Hinweis:
Für die Beantwortung der Fragen müssen Sie "herumexperimentieren", die `man`-Pages der Tools lesen und sich die erzeugten Dateien genauer betrachten. Diese Aufgabe soll Sie bewußt dazu anregen, Ihr Wissen durch Eigeninitiative und kreatives Fragen und Untersuchen zu erweitern (= Lernen im Studium).

c) Starten Sie nun den Simulator zur Simulation eines AVR ATmega16-Prozessors mit folgendem Kommando:

```
simulavr -d atmega16 -P simulavr-disp -g
```

Es erscheint dabei auch ein weiteres Fenster, das den Status der simulierten CPU anzeigt.

Sehen Sie sich aber zunächst die `simulavr`-Ausgaben auf der Kommandozeile an, insbesondere die letzte Zeile, die zeigt, dass der Simulator nun auf einem Netzwerkport auf eine Verbindung mit dem `gdb`-Debugger wartet (das ist das Resultat der Option `-g` beim Starten von `simulavr`). Notieren Sie sich die (vierstelligen) Nummer des Ports.

Sehen Sie sich auch die Details des Simulator-Fensters an. Sie finden dort u.a. die Register (einschließlich r16, r26, r27, dem X-Register und dem Program Counter PC) und in der unteren Fensterhälfte einen Arbeitsspeicherbereich.

d) Öffnen Sie ein neues Shell-Fenster und starten Sie darin den Debugger `avr-gdb`:

```
avr-gdb -tui hello_arduino
```

In `avr-gdb` gibt es eine eigene Eingabekonsole mit speziellen Befehlen. Verbinden Sie den `avr-gdb` mit dem laufenden Simulator, ersetzen Sie dabei `NNNN` durch die im letzten Schritt notierte Portnummer, auf der der Simulator auf Verbindungen wartet.

```
target remote :NNNN
```

Bei Erfolg sollten Sie nun in der oberen Fensterhälfte des `avr-gdb` Ihren Quelltext sehen. Sie müssen das Programm jetzt noch über die aufgebaute Verbindung in den Simulator laden:

```
load
```

Anschließend können Sie das Programm starten:

```
continue
```

Beobachten Sie nun die Effekte des Programmablaufs im Simulator-Fenster und versuchen Sie, sie anhand des Quellcodes zu verstehen und nachzuvollziehen. Sehen Sie sich dazu die Register- und Speicherinhalte und ihre Veränderung an.

Zur weiteren Bearbeitung sollten Sie nun frei üben und sich mit den Möglichkeiten des Debuggers beschäftigen. Beginnen Sie, indem Sie den Programmablauf mit *Strg-C* abbrechen. Sie sehen nun, wo das Programm angehalten hat und können mit dem Kommando `step` schrittweise durch das Programm laufen. Verfolgen Sie jeden Assemblerbefehl und seine Auswirkungen auf den Zustand des simulierten Prozessors.

Die Dokumentation der `gdb`-Kommandos ist in `gdb` mittels `help` abrufbar, versuchen Sie auch in der Linux-Shell `info gdb`, um sich weiter zu informieren. Ein Tipp nebenbei: `gdb`-Kommandos lassen sich auch abkürzen...

e) Modifzieren Sie zur Übung den Assemblerquelltext, experimentieren Sie damit. Ändern Sie beispielsweise die Schleife so, dass immer um 2 hochgezählt wird oder nur 7 Speicherplätze beschrieben werden... seien Sie kreativ!