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

Hardwarenahe Programmierung I

WS 2021/22
LV 1512

Übungsblatt 1

Sie werden in dieser Übung ein erstes Programm bearbeiten und starten, allerdings (noch) nicht in der Programmiersprache C, sondern zuerst in Assembler.

Ziel ist es, im Praktikum echte Mikrocontroller-Boards zu verwenden, aber anfangs wird diese Hardware in Experimenten noch durch den Simulator `simulavr` ersetzt,

- um schneller zu Erfolgserlebnissen zu gelangen und
- Ihnen die Arbeit von daheim zu ermöglichen.

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 unbedingt Ihre Lösungsschritte und Antworten!

Aufgabe 1.1 (Vorbereitungen):

Die Praktika werden unter Linux durchgeführt, dem Betriebssystem, das Sie auch für die Lehrveranstaltung (LV) „Einführung Informatik“ und Ihr weiteres Studium verwenden.

Im Labor stehen Ihnen Linux-Rechner zur Verfügung, auf denen zusätzlich zur Standardkonfiguration die folgenden Software-Pakete installiert sind:

- `make`
- `gcc-avr`
- `binutils-avr`
- `simulavr` (mind. Version 1.9!)
- `gdb-avr` (ggf. stattdessen `gdb-multiarch`)

Außerdem sind diese Rechner so konfiguriert, dass daran auch „echte“ Microcontroller-Hardware angeschlossen werden kann. Davon wird in späteren Aufgaben auch Gebrauch gemacht.

Wenn Sie auch zu Hause arbeiten möchten, benötigen Sie einen Linux-Rechner mit den oben genannten Software-Paketen. Für den Anfang genügt hierfür auch eine virtuelle Linux-Installation, z.B. in

VirtualBox oder VMware. Auch Linux als App unter Windows wäre anfangs möglich, ist aber auf Dauer nicht ausreichend.

Wir empfehlen deshalb, dass Sie sich auf Dauer ein System installieren, bei dem Linux direkt gebootet werden kann (ggf. als Dual-Boot-Lösung mit Windows).

Um dabei möglichst nahe an den Linux-Installationen der Hochschule zu bleiben, empfehlen wir Ihnen derzeit, Debian 11 zu installieren. Dabei handelt es sich um die aktuelle „stable“ Version, d.h. Sie laufen nicht Gefahr, bald auf eine neue Version umsteigen zu müssen.

Sie können zwar auch andere Linux-Distributionen verwenden, allerdings können wir Sie bei Fragen zur Installation dann ggf. nicht so gut unterstützen und Ihr System könnte sich ggf. anders verhalten, als die Hochschul-PCs.

Schließlich besteht auch noch die Möglichkeit, über das Internet auf den Rechnern des PC-Pools des Labors zu arbeiten. Dies sind die Rechner `its01.local.cs.hs-rm.de` ... `its15.local.cs.hs-rm.de`. Auf Rechnern der anderen Linux-Pools sind nicht alle für ITS notwendigen Tools installiert!

Sie benötigen von außerhalb eine VPN-Verbindung, mit der Sie Ihren PC oder Laptop in das Hochschulnetz bringen, die Anleitung finden Sie unter

`https://doku.cs.hs-rm.de/doku.php?id=openvpn`

Anschließend können Sie mit `ssh`, `sftp` & Co. einloggen oder Daten austauschen oder sich auf dem Desktop eines Pool-PCs anmelden.

Die Anleitungen für `ssh` finden Sie unter

`https://doku.cs.hs-rm.de/doku.php?id=ssh_scp_sftp_kurzanleitung`

Wenn Sie VPN aktiviert haben, können Sie z.B. `its01.local.cs.hs-rm.de` statt `login1.cs.hs-rm.de` verwenden.

Für die Anmeldung auf dem Desktop eines ITS-Rechners lesen Sie folgende Anleitung nach:

`https://doku.cs.hs-rm.de/doku.php?id=rdp`

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

Aufgabe 1.2 (Das erste Programm):

a) Laden Sie von der Website der Lehrveranstaltung das Beispiel-`programm 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.

Recherchieren Sie ggf. in der Atmel-Dokumentation, insbesondere der Übersicht der Assemblerbefehle, die auf der Webseite verlinkt ist. Diskutieren Sie den Code in der Praktikumsgruppe und zeichnen Sie ein Flussdiagramm des Programmes.

Sie benötigen für die Bearbeitung der Aufgabe ein geöffnetes Shell-Fenster, in dem Sie in Ihr neues Übungsverzeichnis 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` in eine Programmdatei, die dieselben Instruktionen enthält, diese aber als „Maschinensprache“ binär codiert.

Diese Instruktionen (in `hello_arduino`) sind für Sie zwar dann nicht mehr als Text lesbar (versuchen Sie es!), liegen dafür aber in genau 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 man das prüft, können Sie z.B. unter
  `https://www.selflinux.org/selflinux/html/dateien_unter_linux.html`
  recherchieren.
- Versuchen Sie, das Programm `./hello_arduino` 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?
  Zur Lösung lesen Sie mit man objdump die Dokumentation des Tools.
- Was geschieht, wenn man bei den Aufrufen `-o` weg lässt? Hat die Ausgabedatei noch dasselbe Format ($\rightarrow$ Typ der Datei)?
  Welche Bedeutung hat demnach die Option -o?
- Was geschieht, wenn man bei `avr-as` die Option `-g` weg lässt? 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 -g
```

Sehen Sie sich aber die `simulavr`-Ausgabe auf der Kommandozeile an 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 (vierstellige) Nummer des Ports.

d) Öffnen Sie ein `weiteres` Shell-Fenster, sorgen Sie dafür, dass es möglichst groß ist, 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
```

Mit dem folgenden Befehl lassen Sie nun zusätzlich zum Quelltext die aktuellen Werte der Register des simulierten Prozessors anzeigen

```
layout reg
```

Mit dem Befehl

```
list main
```

sorgen Sie dafür, dass der Quelltext wieder an der richtigen Stelle dargestellt wird:

Mit `list` können Sie eine Speicherstelle oder den Namen eines Programmteils (hier `main`) angeben, und das Listing des Quellcodes an dieser Stelle anzeigen.

e) Anschließend können Sie das Programm Befehl für Befehl ausführen, indem sie wiederholt `step` eingeben. Mit jedem `step` wird ein Befehl ausgeführt, und die markierte Zeile im Quellcode wandert einen Schritt vorwärts.

Bevor Sie die Zeile `st X,r16` ausführen, lassen Sie sich einen Teil des RAM-Speichers des simulierten Prozessors anzeigen:

```
x/16b 0x00800100
```[1]

Damit werden 16 Bytes ab der Speicheradresse (hexadezimal) 100 angezeigt (was der dezimalen Adresse 256 entspricht).

Der Maschinenbefehl `st X,r16` wird nun eines dieser Bytes verändern und den Wert des Register `r16` in die Stelle schreiben, deren Adresse im `X`-Register (16 Bit, zusammengesetzt aus `r27` und

[1] Wundern Sie sich nicht, dass hier die Adresse 800100_{16} statt der vielleicht von Ihnen erwarteten 000100_{16} verwendet wird: der Speicherbereich wird im Simulator aus technischen Gründen *gespiegelt*!

`r26`) steht. Überlegen Sie, was passieren soll, führen Sie den Befehl mit `step` aus und prüfen Sie, ob Sie mit Ihrer Vermutung richtig lagen.

Das Programm wird im weiteren Verlauf in einer Schleife immer wieder in dieser Zeile vorbeikommen, Sie haben also weitere Gelegenheiten, diese Untersuchung durchzu- führen.

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.

> Für Fortgeschrittene:
>
> Mit sogenannten Hook-Routinen können Sie Debugger-Instruktionen automatisch vor oder nach einem Kommando ablaufen lassen. Versuchen Sie es, indem Sie eine Hook-Routine definieren, die nach dem Ausführen des Kommandos `next` aufgerufen wird.

f) 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 und des Speichers.
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, z.B. `s` statt `step` ...)

g) Modifizieren 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!