Übungspaket 1

Grundlagen: von der Hardware zum Programmieren

Übungsziele:

1. Die Bedeutung des Programmierens für mein Studium und meine spätere Berufstätigkeit
2. Was ist eine erfolgreiche Lernstrategie?
3. Die wesentlichen Komponenten des Prozessors

Skript:

Kapitel: 1 bis 5

Semester:

Wintersemester 2020/21

Betreuer:

Thomas, Tim und Ralf

Synopsis:

In diesem Übungspaket werden die wesentlichen Grundlagen kurz wiederholt, die für das Programmieren unbedingt benötigt werden. Dazu gehört neben der allgemeinen Motivation und der Bedeutung des Programmierens für das Studium und die spätere Berufstätigkeit auch das Verständnis der Funktionsweise der wesentlichen Hardware-Elemente eines Computers wie Arbeitsspeicher und Prozessor (CPU).

Teil I: Stoffwiederholung

Aufgabe 1: Fragen zur Motivation

Aus welchen Bestandteilen bestehen die Lehrunterlagen?

Zu den Lehrunterlagen gehören das Skript, die Übungsblätter und sonstige Arbeitsblätter, die früher oder später verwendet werden. Auf den zweiten Blick umfassen die Lehrunterlagen aber ebenfalls die Literatur, die auf der Webseite der Lehrveranstaltung angegeben wird. Und auf den dritten Blick kann eigentlich jede zugängliche Dokumentation zur Programmiersprache C als Lehrunterlage aufgefasst werden, beispielsweise auch die `man-pages` auf den Linuxsystemen oder den Webseiten, die sich der Sprache C widmen. Natürlich fragt man sich als Student: „Wie, dass soll ich alles lesen und zur Klausur lernen?“ Nein, die Klausur prüft, ob der Student ein Verständnis von der Programmiersprache C hat. Sie prüft nicht diese oder jene Kapitel aus dieser oder jener Quelle.

Welche Bedeutung haben die Übungsaufgaben?

Die Aufgaben dienen der Anwendung und der Vertiefung des behandelten Lehrstoffs. Es geht nicht nur darum, die Aufgabe zu lösen, sondern auch darum, Alternativen zu probieren. Es geht darum, eine gewisse Neugier zu erzeugen: Warum funktioniert das so? Wie sehen mögliche Alternativen aus? Welche Vor-/Nachteile haben diese? Was passiert, wenn ich bestimmte Zeilen ändere?

Welchen Zweck verfolgen die Musterlösungen? Wofür sind sie nicht geeignet?

Der Zweck der Musterlösungen liegt darin, nur eine mögliche Lösungsvariante vorzustellen und eventuell hier und da noch auf ein paar Besonderheiten einzugehen. Niemand sollte auf die Idee kommen, Musterlösungen für die Klausur auswendig zu lernen. Niemand sollte auf die Idee kommen, dass man C anhand der Musterlösungen lernen kann. Richtig geht man vor, wenn man sich *nach* dem Erarbeiten einer eigenen Lösung die Musterlösung anschaut und mit der eigenen vergleicht. Was wurde wo und warum anders gemacht?

Wie sieht eine sinnvolle Lernstrategie aus?

Gemäß obiger Ausführungen sind die einzelnen Schritte wie folgt:

1. Die jeweils *nächste* Vorlesung durch Lesen der entsprechenden Skriptkapitel *vorbereiten*. Siehe dazu auch die entsprechenden Arbeitsblätter.
2. Zur Vorlesung gehen und gut zuhören *sowie mitarbeiten.*
3. Die Vorlesung durch nochmaliges Lesen der entsprechenden Kapitel *nachbereiten.*
4. Die Übungsaufgaben lösen soweit dies ohne Hilfe der Assistenten geht.
5. *Nach* erfolgreichem Lösen der Übungsaufgabe diese mit der Musterlösung vergleichen und dabei die Unterschiede finden, verstehen und beurteilen.

Warum lerne ich ausgerechnet die Programmiersprache C? Welche Bedeutung wird sie in meinem Studium sowie meinem späteren Berufsleben haben?

Grundsätzlich gehört C zu den wichtigsten Programmiersprachen überhaupt. Das ist dadurch begründet, dass C beispielsweise für nahezu alle Mikrokontroller verwendet werden kann. C ist eine hardwarenahe Abstraktion von Assembler (Maschinencode). Das macht C zu *der* Programmiersprache für Elektrotechniker. C deckt das gesamte Spektrum von der hardwarenahen Programmierung kleiner Mikrokontroller bis hin zu komplexen PC-basierten Anwendungen ab. Weiterhin erleichtert ein gutes Verständnis der Programmiersprache C auch den Umstieg auf nahezu jede andere Programmiersprache, wie beispielsweise Java oder C++. Wer diese Punkte verstanden hat, der wird auch die Bedeutung der Programmiersprache C für das Studium und das spätere Berufsleben einschätzen können.

Aufgabe 2: Hardware

Erkläre in eigenen Worten die folgenden Speicher-Begriffe: Bit, Byte, Wort.

In der Elektronik bezeichnet das *Bit* die kleinste Dateneinheit. Ein Bit kann lediglich die Zustände Null oder Eins annehmen. Dies ist auch die Bedeutung des Bits in der Programmiersprache C. Alle anderen Datenformate wie Zahlen, Zeichen und so weiter sind lediglich definierte *Bitmengen,* die man je nach Verwendung entsprechend interpretiert. Viele Mikrokontroller und PCs betrachten eine Spannung von 0 V als den logischen Wert Null (oder auch „false“ oder „low“) und eine Spannung von beispielsweise 3.3 V als den logischen Wert Eins (alternativ auch „true“ oder „high“). Ein *Byte* ist eine Gruppe, die aus acht Bit besteht. Ein *Wort* ist eine Gruppe von Bytes. Die genaue Anzahl der Bytes ist dabei vom System abhängig. Üblich sind beispielsweise 2 Byte oder 4 Byte.

Benenne die wesentlichen Register eines Prozessors (einer CPU) und erkläre in eigenen Worten deren Bedeutung und Funktionsweise.

Unabhängig von Modell und Hersteller haben wohl alle CPUs die folgenden Register:

1. Im *Befehlsregister (instruction register)* steht, welchen Befehl die CPU gerade ausführt. Beispiel: Setze das Programm an Adresse `xyz` fort.
2. Der *Befehlszähler (program counter, instruction pointer)* gibt an, wo im Arbeitsspeicher sich der nächste Befehl befindet.
3. Die *Adressregister (address register)* enthalten Speicheradressen von Variablen, Zahlen und anderen Werten, auf die die CPU gleich oder in Kürze zugreifen möchte.
4. Das *Stapelregister (stack pointer)* ist ein weiteres Adressregister, das die CPU im Wesentlichen für eigene Zwecke benötigt.
5. Die *Datenregister (Akkumulatoren)* speichern die Daten und Resultate der arithmetischen und logischen Operationen, die die Recheneinheit (ALU) ausführt.

Die meisten CPUs besitzen noch weitere Register wie Status- und Interruptregister, die aber erst in weiterführenden Lehrveranstaltungen behandelt werden.

Was ist ein Programm und auf welchen „Abstraktionsebenen“ kann man programmieren?

Es lassen sich unter anderem die folgenden drei Ebenen identifizieren:

1. Das *direkte Programmieren* mit Nullen und Einsen im Arbeitsspeicher erfordert explizites Wissen, welche Nullen und Einsen welchen Befehl kodieren. Dies ist in der Regel *sehr* zeitraubend, fehleranfällig und sehr frustrierend.
2. Die Assemblerprogramierung ist die direkte Programmierung mittels Maschinenbefehlen. Auch hier ist der Aufwand sehr hoch, da komplexe Operationen Schritt für Schritt eigenständig beschrieben werden müssen. Die Assemblerprogrammierung ist nicht plattformunabhängig, wodurch ein Assemblerprogramm in der Regel für jeden neuen Prozessortyp auch wieder neu geschrieben werden muss. Beispielhafte Assemblerbefehle sind: lade Wert `1` in Akkumulator, lade Wert `2` in Akkumulator, addiere Wert `1` und Wert `2`, speichere Ergebnis an Adresse `xyz`.
3. Die Programmiersprache C gehört zu den sogenannten Hochsprachen, auch wenn es noch wesentlich „höhere“ Programmiersprachen wie beispielsweise Pascal, Ada, SQL etc. gibt. C ist weitestgehend plattformunabhängig und erlaubt verständliche Programmstrukturen. Die Sprache verfügt über Schleifen, Verzweigungen und den Aufruf eigener Funktionen.

Teil II: Quiz

Beurteile aus eigener Sicht, wie zutreffend die folgenden Aussagen sind:

Es ist unwichtig, zur Übung zu gehen.	überhaupt nicht zutreffend
Das Ansehen der Musterlösungen ist völlig ausreichend.	überhaupt nicht zutreffend
In meinem späteren Berufsleben spielt das Programmieren keine große Rolle.	überhaupt nicht zutreffend
Die Assistenten korrigieren mir meine Programmierfehler.	überhaupt nicht zutreffend
Es ist völlig überflüssig, die Übungsaufgaben zu Hause vorzubereiten bzw. zu lösen.	überhaupt nicht zutreffend
Die Klausuraufgaben sind jedes Jahr gleich.	überhaupt nicht zutreffend
Es ist ganz besonders hilfreich und sinnvoll, die Aufgaben der letzten Klausuren (auswendig) zu lernen.	überhaupt nicht zutreffend

Teil III: Fehlersuche

Unser Newbie Grand Master C hat unsere Unterlagen durchgearbeitet und sich folgenden „Arbeitsplan“ zurechtgelegt:

1. Programmieren ist cool. Am besten setze ich mich gleich an den Rechner, denn dann habe ich alles zusammen und kann loslegen.
2. Die Musterlösungen sind sehr gut. Statt selbst zu programmieren, arbeite ich diese lieber durch; bringt mehr und ist beides, effizienter und effektiver.
3. Der Besuch der Vorlesungen ist Pflicht, damit mich der Professor kennenlernt und mir eine gute Note gibt.
4. Der Besuch der Übungen ist fakultativ, denn dort sind nur Doktoranden, die keinen Einfluss auf die Note haben.
5. Ich besorge mir die alten Klausuren, denn auch Professoren sind bequem und denken sich keine neuen Aufgaben aus.
6. Ich lass es locker angehen, denn wir haben ja einen Freiversuch.
7. Ich soll das Skript lesen? Keine Zeit, die Dozenten erzählen doch alles in der Vorlesung...
8. Die Übungsblätter zu Hause oder gar in der Freizeit bearbeiten? No way! Ist auch quatsch, wir haben doch genügend viele Übungsstunden.
9. Wenn es (zeitlich) eng wird, nehme ich oder besorge mir die Musterlösungen und schreibe einfach ab. Die Doktoranden werden schon ihre eigenen Musterlösungen mögen.

Aus unserer Lehrkörpersicht können wir dieser Arbeitshaltung nicht so recht zustimmen. Finde, erläuter und korrigiere Grand Master C's Missverständnisse auf den folgenden Seiten :-)

Nr.	Kommentierung	Korrigierte Aussage
1.	Einer der Hauptfehler ist es, sich gleich an den Rechner zu setzen. In diesem Fall weiß man noch gar nicht, was man eigentlich programmieren soll, sodass man den Wald vor lauter Bäumen nicht sieht. So scheitert man, obwohl man viele Stunden am Rechner verbringt.	Ich bereite mich mit meinen Studienkollegen gut vor, löse alle Aufgaben auf einem Zettel und setze mich erst dann an den Rechner, um zu programmieren.
2.	Das Abschreiben von Programmen und Musterlösungen aus dem Internet kann man sich sparen, da man dabei so rein gar nichts lernt! Derartige Übungen sind zu 100 % Zeitverschwendung.	Meine Freunde und ich lösen die Aufgaben zusammen. Erst wenn ich diese erfolgreich umgesetzt habe, vergleiche ich meine Lösungen mit den Musterlösungen, um auf Alternativen aufmerksam zu werden.
3.	Der Professor freut sich über jeden Studenten, der zur Vorlesung kommt und beantwortet auch gerne alle Fragen. Es nutzt aber rein gar nichts, sich beim Professor bekannt zu machen, denn es gibt keine Gummipunkte. Die Anwesenheit in der Vorlesung hat keinen Einfluss auf die Prüfungsnote, da diese *ausschließlich* von der Klausur abhängt.	Ich gehe zur Vorlesung, um die wesentlichen Inhalte nochmals zu hören und ggf. Fragen zu stellen, die ich dort beantwortet bekomme.
4.	Die Teilnahme am Übungsbetrieb ist weit aus wichtiger als das Zeitabsitzen in der Vorlesung. Bei den Übungen können die Doktoranden individuell auf meine Programmierfertigkeiten und -probleme eingehen.	Ich gehe zu den Übungsterminen, um die letzten Fehler meiner Programme zu beseitigen. Bei den Übungsterminen bekomme ich zwar keine fertigen Lösungen kann aber Fragen stellen und bekomme *individuelle* Hilfestellungen, Hinweise und Anregungen.
5.	Es mag sein, dass es Vorlesungen gibt, die so funktionieren. In der Veranstaltung *Einführung in die Praktische Informatik* denken sich die Doktoranden und der Professor *immer* neue Prüfungsfragen aus. Bisher hat es noch keinem Studenten geholfen, die alten Klausur auswendig zu lernen. Nur glauben etwa 50 % der Studenten dies nicht.	Um erfolgreich zu sein, lerne ich und bearbeite die Übungspakete *selbstständig*.

Nr.	Kommentierung	Korrigierte Aussage
6.	Ja, das erleben wir leider all zu oft. Ja, es gibt einen Freiversuch und ja, ich kann die Klausur öfters wiederholen. Aber, das Studium ist vollgepackt mit Lehrveranstaltungen. Und da bleibt keine Zeit, diese in späteren Semestern zu wiederholen.	Ich lass es nicht locker angehen sondern werde mich jede freie Minute mit dem Programmieren beschäftigen. Ich werde mich auch mit meinen Freunden in der Kneipe verabreden, um einzelne Übungsteile bei einem leckeren Bier gemeinsam zu bearbeiten.
7.	Das ist die ganz falsche Herangehensweise. In den Vorlesungen ist nur Zeit, einige ausgewählte Aspekte zu behandeln. Wenn man sich den Stoff nicht *vorher* intensiv angeschaut hat, kann man nur schwer folgen und bekommt wenig mit.	Ich werde also die relevanten Skriptteile *vor* jeder Vorlesung durch*arbeiten*, um vorbereitet zu sein und dem Dozenten folgen zu können. Ich werde jedesmal mindestens eine Frage vorbereiten; mal sehen, ob er sie beantworten kann ;-)
8.	Man mag es nicht glauben, aber Programmieren ist eine sehr zeitaufwändige Tätigkeit. Dabei vergeht die Zeit wie im Fluge. Wenn ich nicht schon alles zu Hause vorbereitet habe, reicht die Zeit bei weitem nicht aus. Ein Hauptproblem ist es, die Programmierfehler erst einmal zu finden...	Die Aufgabenblätter werde ich (zusammen mit meinen Freunden) zu Hause vorbereiten, um die Übungszeit zum Testen meiner Programme zu nutzen.
9.	Das Abschreiben irgendwelcher Lösungen bringt gar nichts! Die Doktoranden interessiert es auch so rein gar nicht, ihre eigenen Lösungen zu sehen, denn sie können schon programmieren. Ferner muss ich in der Klausur in der Lage sein, neue, mir unbekannte Aufgaben unter Zeitdruck selbstständig zu lösen; sonst falle ich *gnadenlos* durch.	Die vorhandenen Musterlösungen *arbeite* ich erst durch, wenn ich die Aufgaben (mit Hilfe der Doktoranden) selbst gelöst habe. Bei Unklarheiten frage ich die Doktoranden.

Teil IV: Anwendungen

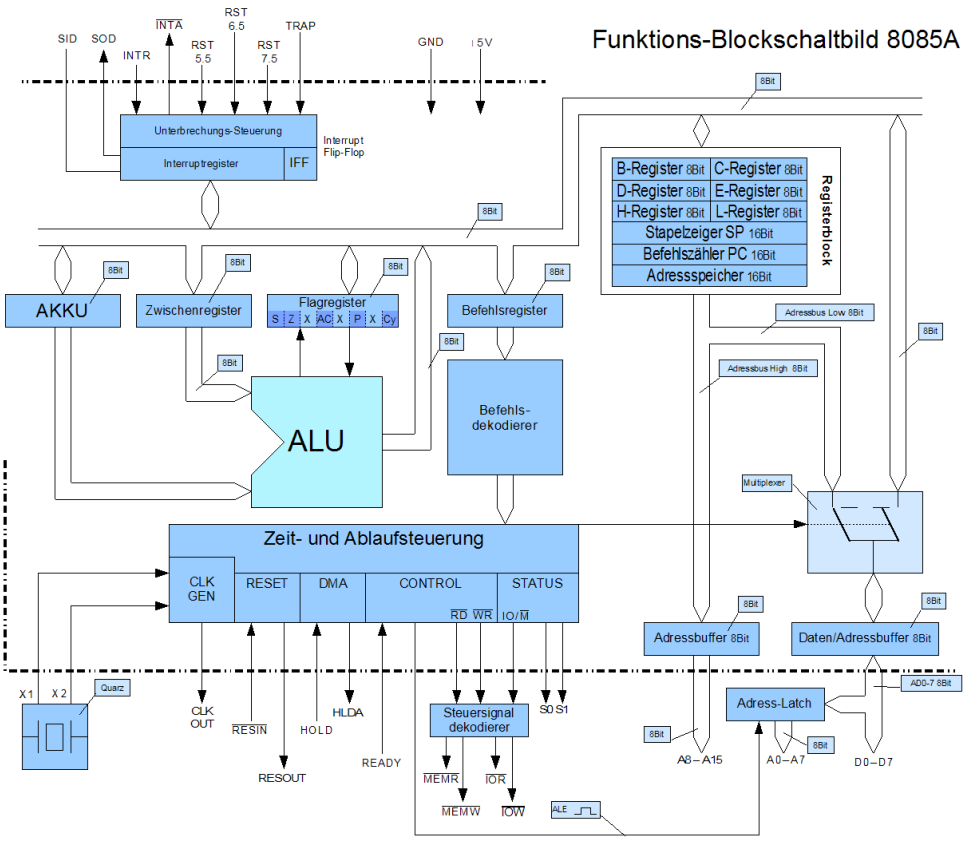
Die folgenden Aufgaben lassen sich mit Hilfe einer einfachen Internet-Recherche relativ einfach lösen.

Aufgabe 1: CPU Register

In der Vorlesung und Skript (Kapitel 5) haben wir einige Register besprochen, die in *jeder* CPU in der einen oder anderen Form vorhanden sein *müssen*.

Frage: Welche Register hat der Intel Prozessor 8085? Recherchiere ein wenig im Internet und beschreibe oder illustriere kurz den Aufbau des Prozessors.

Folgende Grafik des Intel 8080 Prozessors stammt von der deutschsprachigen Wikipedia-Seite `http://de.wikipedia.org/wiki/Intel_8085`. Im Wesentlichen sind die Register zu finden, die wir schon im ersten Aufgabenteil besprochen haben.



Aufgabe 2: Taktfrequenzen

Ein weiterer interessanter technischer Parameter beschreibt die Geschwindigkeit, mit der eine CPU arbeitet.

Frage: Mit welchen Taktfrequenzen arbeiten moderne Prozessoren und deren angeschlossenen Arbeitsspeicher (auch häufig als RAM bezeichnet).

Im April 2013 ergab eine kleine Recherche folgende Taktfrequenzen für handelsübliche Komponenten.

Prozessoren:

- IBM POWER7 – Quad Core: 4,14 GHz
- Intel Xeon E3-1290 v2 – Quad Core: 3,7 GHz
- AMD FX-8150 – 8-Core: 3,6 GHz

Übertaktete Prozessoren (Kühlung mittels flüssigem Stickstoff):

- AMD FX-8150 : 9,144 GHz (www.radeon3d.org/forum/thread-2896.html)
- Intel Celeron LGA775 352: 8,505 GHz (www.hwbot.org)
- AMD FX-8350: 8,502GHz (www.hwbot.org)

Arbeitsspeicher: Im Mai 2012 wurde DDR4 als neueste Speichertechnologie vorgestellt. DDR4 erlaubt Werte von bis zu 400 MHz für den Speichertakt. DDR4 arbeitet allerdings mit einem als „Prefetch“ bezeichnetem Verfahren. Dabei werden bei einer Leseanforderung acht mal mehr Daten an der Schnittstelle zwischen Speicher und System bereitgestellt, als eigentlich angefordert. Das beruht auf der Annahme, dass die folgenden Daten auch noch benötigt werden könnten. Dadurch ergibt sich ein effektiver Takt von 3,2 GHz. Als Systemtakt zwischen Speicher und CPU ist aber nur die halbe Taktrate, also 1,6 GHz notwendig, da Daten zwei mal pro Takt (sowohl bei der fallenden als auch bei der steigenden Taktflanke) übertragen werden.

Aufgabe 3: Speichermedien

Wie groß ist heute üblicherweise der Arbeits- und Plattenspeicher moderner PCs?

Im Frühjahr 2013 hatte jeder Spiele-PC, der etwas auf sich hält, mindestens 16 GB Arbeitsspeicher. Als Größe für Plattenspeicher (Einzellaufwerke) haben sich Werte im unteren einstelligen Terrabyte (TB) etabliert, die Topmodelle (beispielsweise Seagate Constellation) bringen 4 TB mit.