Technische Universität Graz
Fakultät für Informatik und Biomedizinische Technik

Institut für Angewandte Informationsverarbeitung
und Kommunikationstechnologie

Rechnerorganisation Konstruktionsübungen 2018

LV Nr. 705.037
Version vom 2. Februar 2018
Karl C. Posch

Latitude: *56.917922* | ***Longitude:*** *18.155165*

Inhaltsverzeichnis

Einleitung

Dieses Dokument dient als Basis für die Aufgaben, welche du im Rahmen der Lehrveranstaltung „KU Rechnerorganisation“ im Sommersemester 2018 machen sollst. Die Aufgaben beziehen sich auf die in der Vorlesung Rechnerorganisation besprochenen Themen. Ich rate also dringend an, Vorlesung und Konstruktionsübungen als ein zusammen gehörendes Ganzes zu betrachten. Wenn du die Themen der Vorlesung wirklich verstehen willst, dann musst du die Übungen machen. Und wenn du die Übungen erfolgreich machen möchtest, dann musst du dich mit dem Vorlesungsstoff auseinander setzen. Erst durch diese gemeinsame Sichtweise bekommst du ein vollständiges Bild vom Thema Rechnerorganisation.

Ich wünsche dir viel Spaß beim Arbeiten.

Leistungsdokumentation

Du bist auf dem Weg, eine Ingenieurin oder ein Ingenieur werden zu wollen. Ein professioneller Zugang zu Ingenieursarbeit beinhaltet auch die Dokumentation der Arbeit. Dazu gibt es folgende Werkzeuge:

Das Ingenieurstagebuch: In einem Ingenieurstagebuch dokumentierst du für dich selbst deine Arbeit am Projekt. Deine Ideen, deine Versuche, Skizzen, Termine, Treffen mit Gruppenmitgliedern und vieles mehr. Du dokumentierst dabei vor allem auch, wann du wie viele Stunden gearbeitet hast. Du möchtest am Ende ja schließlich wissen, wie viel Zeit du für die Lehrveranstaltung aufgewendet hast. Ich werde dich, nachdem du die Note bekommen hast, ebenfalls danach fragen. Denn ich möchte auch wissen, wie viel Zeit Studierende für die Lehrveranstaltung aufwenden. Erst dadurch kann ich meine anfängliche Schätzung mit wirklichen Zeiten vergleichen und daraus lernen. Der geplante Aufwand für Vorlesung und Übungen zusammen für den durchschnittlichen Studierenden ist mit 4,5 ECTS-Punkten vorgegeben. Dies entspricht einem durchschnittlichen Zeitaufwand von etwa 120 Arbeitsstunden. Ob diese Planung stimmt, kann ich nur mit Hilfe der Zeitaufzeichnungen einer größeren Menge von Studierenden im Nachhinein ermitteln. Und mit diesem Ergebnis kann ich das Übungsausmaß für das nachfolgende Jahr kalibrieren.

Die Deliverables: Das sind alle Dokumente, welche du dem Auftraggeber, in unserem Fall hier also dem Lehrer und dem/der Tutorin rechtzeitig liefern (= deliver) musst.

Die Präsentation: Du wirst im Rahmen von zwei Abgabegesprächen Gelegenheit haben, deine Arbeit zu präsentieren und zu verteidigen. Damit kannst du dokumentieren, mit welcher Qualität du die Arbeit gemacht hast.

Termine: Apropos „rechtzeitig liefern“. Ein wesentliches Element von professioneller Qualität einer Arbeit besteht schließlich im Einhalten von Terminen. Ich dränge dich also als Auftraggeber dazu, die Abgabefristen einzuhalten. Der jeweils angegebene Abgabetermin ist der spätestens mögliche. Du darfst auch früher abgeben. Ein guter Rat besteht darin, dass du dir jeweils einen eigenen internen Termin setzt, welcher ein gutes Stück vor dem von mir vorgegebenen Termin liegt. Damit vermeidest du am ehesten extern verursachten Stress. Du hast für die Aufgaben 0 bis 3 zusammen 3 mal 24 Stunden Überziehungsfrist, welche sich nicht auf die Punkte und somit auch nicht auf die Note auswirken. Für die **Gruppenaufgabe** gibt es **keine** Überziehungsfrist. Spare dir diese Überziehungsfrist so lange es geht auf. Du solltest diese Reserve nur in unvorhergesehenen Notfällen „anzapfen“. Jede darüber hinaus gehende Überziehung schlägt sich mit 10 Punkten Abzug (von den

erreichten Gesamtpunkten für die gesamte Übung) pro 24 Stunden nieder. Mehr zum Thema „Zeitplanung“ gibt es gegen Ende dieses Dokuments unter dem Titel „Huch, es geht sich mit der Zeit nicht aus“.

Anmeldefrist zur Übung

Melde dich bis spätestens 9. März 2018 an einer der 5 Übungsgruppen im TUGRAZonline an. Solltest du zu diesem Zeitpunkt noch keinen TUGRAZonline-Zugang besitzen, weil du soeben erst im ersten Semester inskribiert hast, dann schicke vor dem 8. März 2018 eine Mail an Karl.Posch@iaik.tugraz.at.

Wenn du nicht angemeldet bist, dann kannst du keine positive Note bekommen. Eine Note bekommst du, sobald du zumindest eine Abgabe gemacht hast.

Teilnahmepflicht am Tutorium 0

Die Teilnahme am Tutorium 0 ist verpflichtend. Wenn du dieses versäumst, dann kannst du keine Note kriegen. Das Tutorium 0 findet je nach Übungsgruppe an folgenden Terminen statt:

- 14. März 2018, 8:00-9:00 Gruppe Habich Hörsaal i12
- 13. März 2018, 8:00-9:00 Gruppe Hadzic Hörsaal i12
- 14. März 2018, 9:00-10:00 Gruppe Ulbel Hörsaal i12
- 12. März 2018, 8:00-9:00 Gruppe Unterguggenberger Hörsaal i12
- 12. März 2018, 9:00-10:00 Gruppe Weiglhofer Hörsaal i11

Abgabetermine

Aufgabe 0:	Montag,	19. März 2018, 14:00
Aufgabe 1:	Freitag,	13. April 2018, 14:00
Aufgabe 2:	Freitag,	27. April 2018, 14:00
Gruppenmeldung:	Montag,	14. Mai 2018, 14:00
Aufgabe 3:	Montag,	14. Mai 2018, 14:00
Aufgabe 4:	Freitag,	8. Juni 2018, 14:00

Abgabe der Deliverables über das „Student Tick System“ (STicS) unter https://stics.iaik.tugraz.at. Vergiss nicht, dich rechtzeitig vor der ersten Abgabe bei diesem System zu registrieren. Die Möglichkeit zur Registrierung besteht ab dem 13. März 2018. Verwende bei dieser Registrierung als E-Mail deine TU-Graz-E-Mail-Adresse. In besonderen Fällen darfst du deine Abgabe auch per E-Mail an Karl.Posch@iaik.tugraz.at machen.

Gruppenbildung

Die vierte Aufgabe dieser Übung machst du in einer 3-er-Gruppe. Suche dir im Laufe der ersten Wochen des Semesters drei GruppenpartnerInnen. Die Gruppenmeldung erfolgt über das „Student Tick System“ (STicS) unter https://stics.iaik.tugraz.at. Der späteste Termin für diese Meldung ist der 14. Mai 2018, 14:00.

Aufgabe 0:

Vorstellung bei der/dem TutorIn; Kennenlernen von TOY; Auseinandersetzung mit den Prinzipien guter wissenschaftlicher Praxis

1. Besuche das Tutorium 0. Lege fest, ob du das Tutorium weiterhin besuchen willst/kannst oder nicht und teile dies dem/der TutorIn mit. Im Falle der Entscheidung, das Tutorium zu besuchen, machst du deine beiden Abgabegespräche mit dem/der TutorIn. Ansonsten mit dem Lehrer selbst.
2. Lerne Visual X-TOY kennen.
3. Verstehe die Begriffe „Plagiat“ und „Täuschungsversuch“.

Termin des Tutoriums 0

14. März 2018, 8:00-9:00	Gruppe Habich	Hörsaal i12
13. März 2018, 8:00-9:00	Gruppe Hadzic	Hörsaal i12
14. März 2018, 9:00-10:00	Gruppe Ulbel	Hörsaal i12
12. März 2018, 8:00-9:00	Gruppe Unterguggenberger	Hörsaal i12
12. März 2018, 9:00-10:00	Gruppe Weiglhofer	Hörsaal i11

Spätester Abgabetermin

- Montag, 19. März 2018, 14:00.
- Abgabe über das „Student Tick System“ (STicS) unter https://stics.iaik.tugraz.at. Vergiss nicht, dich rechtzeitig vor der ersten Abgabe bei diesem System zu registrieren. Die Möglichkeit zur Registrierung beim „Student Tick System“ besteht ab dem 13. März 2018. Verwende bei dieser Registrierung als E-Mail deine TU-Graz-E-Mail-Adresse. In besonderen Fällen darfst du deine Abgabe auch per E-Mail mit dem Betreff „[RO] Abgabe 1“ an Karl.Posch@iaik.tugraz.at machen.

Vorgangsweise

Diese Aufgabe besteht aus mehreren Teilen:

1. Besuch des Tutoriums 0
2. Kennenlernen von Visual X-TOY
3. Auseinandersetzung mit Prinzipien guter wissenschaftlicher Praxis

Die Resultate dieser Aufgabe gibst du in Form eines Textes ab. Das Dateiformat dieses Textes ist PDF. Der Name der PDF-Datei entspricht deiner Matrikelnummer gefolgt von der Dateierweiterung „.pdf“. Auf der ersten Seite dieses Dokuments steht zuerst der Titel des Dokuments, danach dein Name und deine Matrikelnummer. Das Dokument besteht aus drei Kapiteln. Das erste Kapitel heißt „Einleitung“ und stellt eine Zusammenfassung des Dokumentes dar. Das zweite Kapitel heißt „Kennenlernen von Visual X-TOY“. Das dritte Kapitel heißt „Prinzipien guter wissenschaftlicher Praxis“. Formuliere deinen Text in ganzen Sätzen. Der Text kann in Deutsch oder in Englisch verfasst sein.

Kennenlernen von Visual X-TOY:

- Besorge dir eine Kopie von Visual X-TOY, installiere diese auf deinem Computer und lerne Visual X-TOY kennen. Der Text zur Vorlesung hilft dir bei dieser Arbeit. Sieh dir die eingebauten Beispielprogramme an und simuliere einige dieser Programme. Lerne dabei auch die verschiedenen Modi von Visual X-TOY kennen.

- Schreibe ein Maschinenprogramm für Visual X-TOY, welches die Ziffern deiner Matrikelnummer zusammenzählt und dessen Summe ausgibt. Simuliere dieses Programm auf Visual X-TOY.

- Sieh dir den Inhalt von „Tools→Options→Execution→Exception Handling“ an. Sieh dir die Beschreibung der Einstellungen an und versuche diese zu verstehen. Notiere dir diejenigen Exceptions, welche du nicht verstanden hast.

- Beantworte im Text deiner Abgabe folgende Fragen:

    1. Beschreibe zunächst kurz, was du im Rahmen dieser Arbeit alles gemacht hast.
    2. Wie viele Befehlstypen gibt es in TOY?
    3. Wie groß ist der Hauptspeicher von TOY?
    4. Gibt es bei TOY einen Goto-Befehl?
    5. Wie kann man in TOY eine If-Anweisung realisieren?

Auseinandersetzung mit Prinzipien guter wissenschaftlicher Praxis:

- Setze dich mit den Begriffen „Plagiat“ und „Täuschungsversuch“ auseinander. Studiere dazu die Materialien, welche zum Beispiel unter http://www.plagiarism.org zu finden sind.

- Studiere das Dokument “Richtlinie zur Sicherung guter wissenschaftlicher Praxis” (http://mibla.tugraz.at/15_16/Stk_5/RL_Sicherung_guter_wissenschaftlicher_Praxis.pdf).

- Studiere §9 dieses Dokuments und verstehe die Konsequenzen eines Fehlverhaltens in dieser Lehrveranstaltung.

- Schreibe den Text des Kapitels 3 deiner Dokumentation, in welchem du erklärst, dass du (1) die in den obigen Punkten beschriebene Arbeit gemacht hast, dass du (2) verstanden hast, was ein Plagiat bzw. ein Täuschungsversuch ist, dass du (3) die Konsequenzen eines Fehlverhaltens in dieser Lehrveranstaltung verstanden hast, und dass du (4) kein Plagiat abgeben wirst. Verwende in diesem Text Sätze mit dem Wort „ich“ als Subjekt. Setze unter deinen Text deinen Namen.

Wozu Aufgabe 0?

Jede/r angemeldete StudentIn sollte explizit die Entscheidung treffen, ob sie/er die Übung innerhalb einer Übungsgruppe mit Betreuung eines/einer TutorIn machen möchte oder ob dies nicht möglich ist. Ein typischer Grund für eine Nicht-Teilnahme wäre eine berufsbedingte Verhinderung.

Die Entscheidung zur Teilnahme sollte zu Beginn der Übung getroffen werden und muss dem Lehrer und der/dem TutorIn mitgeteilt werden. Damit ist es für den Lehrer und für die TutorInnen möglich, sich bestmöglich auf die Betreuung derjenigen Studierenden zu konzentrieren, welche am Tutorium teilnehmen.

Ein lediglich sporadischer Besuch der Tutorien ist unerwünscht.

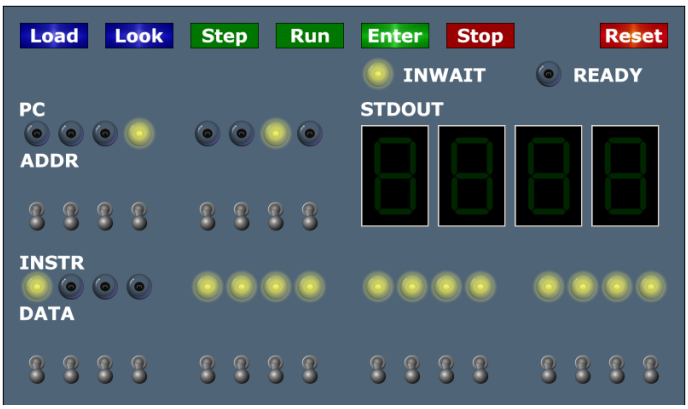
Solltest du – in seltenen Fällen – trotz Teilnahmeentscheidung an einem Tutorium nicht teilnehmen können, dann teilst du dies dem/der TutorIn vorher per Mail mit. Solltest du dich vom Tutorium abmelden wollen, kannst du dies ebenfalls per einfacher Mail an den/die TutorIn sowie an den Lehrer (Karl.Posch@iaik.tugraz.at) machen.

Der wesentliche Grund für diese Vorgangsweise liegt im Ansinnen, Studierende zu einem professionellen Verhalten zu drängen. Professionalität drückt sich dabei folgendermaßen aus: Du lernst, Entscheidungen über deine nähere Zukunft zu treffen und dich dann an diese Entscheidungen zu halten. Du lernst zudem, im Falle von Verhinderung im Vorhinein geeignete Aktionen zu setzen.

Durch die explizite Beschäftigung mit dem Themen „Plagiat“ und „Täuschungsversuch“ lernst du die Grundregeln der wissenschaftlichen Praxis kennen.

Aufgabe 1:

Assemblerprogramm mit Visual X-TOY



(Einzelarbeit)

Entwickle mit Hilfe der Sprache C ein Assemblerprogramm für TOY, welches ASCII-kodierte Dezimalzahlen vom Standard-Input einliest, diese intern in Binärform speichert, gegebenenfalls addiert, und die Zahlen sowie die Summe ASCII-kodiert in Dezimalformat am Standard-Output ausgibt. Die Reihenfolge der am Standard-Input erwarteten Operanden bzw. Operatoren erfolgt gemäß der „umgekehrten polnischen Notation“ („reverse Polish notation“, RPN). Bei der umgekehrten polnischen Notation werden zunächst die Operanden eingegeben und danach der darauf anzuwendende Operator.

Das Assemblerprogramm soll folgende ASCII-Zeichen am Standard-Input verarbeiten können: 0x30 bis 0x39 (stellvertretend für die Ziffern 0 bis 9), 0x0A („newline“, ‚\n‘), 0x2B (Pluszeichen, ‚+‘) und 0x03 („Ende der Eingabe“, EOF).

Will man zum Beispiel die dezimale Zahl „einhundertdreiundzwanzig“ (123) eingeben, so kann dies durch die folgenden 4 ASCII-Zeichen am Standard-Input dargestellt werden: 0x31, 0x32, 0x33, 0x0A. Als Ergebnis dieser Eingabe soll die Zahl 123 in Binärformat gespeichert werden. Will man zu dieser Zahl 123 eine weitere Zahl, zum Beispiel 456 addieren, dann wird die Eingabe folgendermaßen fortgesetzt: 0x34, 0x35, 0x36, 0x2B. Mit dem abschließenden ASCII-kodierten Pluszeichen (0x2B) wird dem Programm mitgeteilt, dass es die Addition der beiden zuvor eingegebenen Zahlen ausführen soll und danach die Summe (479) ASCII-kodiert in Dezimalformat ausgeben soll. Das Programm soll in diesem Fall folgende Zeichen ausgeben: 0x31, 0x32, 0x33, 0x0A, 0x34, 0x35, 0x36, 0x2B, 0x35, 0x37 und 0x39. Es sind dies also alle Zeichen der Eingabe gefolgt vom Ergebnis der Addition.

Das Programm soll in der Lage sein, bis zu 16 eingegebene Dezimalzahlen in Binärformat in einem First-In-Last-Out-Puffer zu speichern. Mit dem Einlesen eines Pluszeichens werden die beiden zuletzt eingegebenen Dezimalzahlen addiert und deren Summe wird ausgegeben. Durch eine Addition wird der zuletzt eingegebene Wert im First-In-Last-Out-Puffer „verbraucht“.

Hier ein Beispiel dazu: Zuerst geben wir die Zahlen 13, 14, 15 und 16 ein. Dies entspricht der folgenden Eingabesequenz: 0x31, 0x33, 0x0A, 0x31, 0x34, 0x0A, 0x31, 0x35, 0x0A, 0x31, 0x36, 0x0A. Es befinden sich jetzt also 4 Zahlen im Last-In-First-Out-Puffer.

Danach geben wir die Zahl 17 gefolgt von einem Pluszeichen ein: 0x31, 0x33, 0x2B. Zunächst erkennt das Programm die Zahl 17 und speichert diese in Binärformat in einer Variablen („X“). Mit dem Pluszeichen soll das Programm die zuletzt eingegebene Zahl (17) zur zuvor eingegebenen Zahl (16) –

das ist die „oberste“ Zahl im First-In-Last-Out-Puffer – addieren. Das Ergebnis (33) soll in der Variablen X gespeichert werden und danach ASCII-kodiert in Dezimalform ausgegeben werden: 0x33, 0x33. In der Variablen X befindet sich nun der binärkodierte Dezimalwert 33. Im First-In-Last-Out-Speicher wurde nun die Zahl 16 „verbraucht“ und es steht die Zahl 15 an oberster Stelle. Die Eingabe eines weiteren Pluszeichens soll nun den Wert in der Variablen X (33) zu 15 addieren, die Summe (48) in X binär kodiert speichern und diese Summe ASCII-kodiert in Dezimalform ausgeben. Am Ausgang soll also 0x34 gefolgt von 0x48 erscheinen.

Die Variable X sowie die 16 Speicherstellen des First-In-Last-Out-Puffers haben zu Beginn den Wert 0.

Sobald das TOY-Programm das Eingabezeichen 0x03 einliest, hält das Programm an.

Assembliere das TOY-Assemblerprogramm und teste das resultierende TOY-Maschinenprogramm mit Visual X-TOY.

Beginne deine Überlegungen zur Lösung indem du zunächst einen C-Code entwickelst. Ausgehend von diesem C-Quellcode sollst du iterativ funktionsidente Veränderungen an diesem C-Code vornehmen. Du beginnst mit dem Ersetzen von Schleifen durch Goto-Anweisungen; sodann sollst du die Array-Zugriffe so modifizieren, dass die in der TOY-Assemblersprache vorhandenen Möglichkeiten im C-Code direkt ersichtlich werden. Das Resultat dieser Modifikationen soll ein funktionierender C-Code sein, der Zeile für Zeile in TOY-Assemblerbefehle übersetzt werden kann.

Verwende in deinem C-Code nur globale Variable. Im Hauptprogramm „main“ sollen nur folgende Funktionsaufrufe vorkommen: getchar() und putchar().

In der ersten Version deines C-Programms sollen nur folgende Kontrollflussanweisungen vorkommen: For-Schleifen, While-Schleifen, If-(Else)-Anweisungen, Break-Anweisungen und Continue-Anweisungen. Es dürfen in der ersten Version keine Goto-Anweisungen verwendet werden.

Abgabe

Abzugeben sind folgende Dateien, welche zusammen in eine ZIP-Datei gepackt werden. Der Name der ZIP-Datei entspricht deiner Matrikelnummer gefolgt von der Dateierweiterung „.zip“; also zum Beispiel so: „1234567.zip“. Folgende Dateien sollen in der ZIP-Datei enthalten sein:

- Der anfängliche C-Code:
  Dateiname ist Matrikelnummer-dot-c; also z.B. „1234567_1.c“
- Der modifizierte, funktionsidente C-Code (Schleifen in Goto-Anweisungen umgeformt):
  Dateiname ist Matrikelnummer-dot-c; also z.B. „1234567_2.c“
- Der modifizierte, funktionsidente C-Code (Array-Zugriffe aufgelöst):
  Dateiname ist Matrikelnummer-dot-c; also z.B. „1234567_3.c“
- Der modifizierte, funktionsidente C-Code (Verwendung von TOY-CPU-Registernamen):
  Dateiname ist Matrikelnummer-dot-c; also z.B. „1234567_4.c“

- Der TOY-Assemblercode:
  Dateiname ist Matrikelnummer-dot-asm; also z.B. „1234567.asm“
- Der Maschinencode:
  Dateiname ist Matrikelnummer-dot-toy; also z.B. „1234567.toy“

Frühabgabe mit Rückmeldung

Solltest du bereits eine Woche vor Abgabe deine Lösung fertig gestellt haben, gibt es eine Möglichkeit eine Rückmeldung zu bekommen. Dafür solltest du bis Freitag, 06. April, 14:00 deine Lösung auf das „Student Tick System“ (STicS) hochgeladen haben. Du findest dann bis Montag den 09. April auf https://teaching.iaik.tugraz.at/ro/start eine Datei mit der Anzahl der bestandenen Testfälle.

Spätester Abgabetermin

Freitag, 13. April 2018, 14:00.

Abgabe über das „Student Tick System“ (STicS) unter https://stics.iaik.tugraz.at. Vergiss nicht, dich rechtzeitig vor der ersten Abgabe bei diesem System zu registrieren. Die Möglichkeit zur Registrierung besteht ab dem 13. März 2018. Verwende bei dieser Registrierung als E-Mail deine TU-Graz-E-Mail-Adresse. In besonderen Fällen darfst du deine Abgabe auch per E-Mail mit dem Betreff „[RO] Abgabe 1“ an Karl.Posch@iaik.tugraz.at machen.

Vorgangsweise

- Lege ein Ingenieurstagebuch an und trage dort immer Notizen zu deinen Arbeiten ein. Samt Datum und aufgewendeter Zeit. Dieses Ingenieurstagebuch musst du zu den Abgabegesprächen mitbringen.
- Lerne, mit Visual X-TOY umzugehen. Studiere die Beispiele, welche mit TOY mitgeliefert wurden. Du findest Visual X-TOY hier: http://introcs.cs.princeton.edu/xtoy/
- Lerne den TOY-Assembler kennen. Es gibt 2 Varianten: das Python-Programm „atoyasm.py“und „toyasm“ (basierend auf dem C-Quellcode „toyasm.c“). Beide Assembler gibt es zusammen mit anderen Tools hier: https://seafile.iaik.tugraz.at/f/76c5da794b/).
- Probiere kleine Beispiele zuerst. Ein Beispiel ist etwa das Erkennen einer Zahl aus einer Folge von ASCII-Zeichen oder die Umwandlung einer Binärzahl in eine ASCII-kodierte Dezimalzahl.
- Vermutlich sind folgende Studienmaterialien hilfreich: https://seafile.iaik.tugraz.at/f/46e1468ef6/. Darin findest du eine detaillierte Beschreibung des TOY-Computers.
- Ein heißer Tipp ist auch der Text „A Lesson on Programming“ (https://seafile.iaik.tugraz.at/f/d1b0e83828/) samt den dazu gehörigen Dateien (https://seafile.iaik.tugraz.at/f/d06df6373c/).

- Ein guter Tipp ist auch der Text „From C to TOY Machine Code: A Workout in 22 Sections“ (https://seafile.iaik.tugraz.at/f/ce6ddd6e1e/)
- Eventuell gefällt dir auch folgendes Dokument (vor allem ab Kapitel 3: „Die Zerlegung einer For-Schleife“): https://seafile.iaik.tugraz.at/f/e873638306/.
- Beginne mit einem Modell des zu entwickelnden Programms in der Sprache C. Wandle den C-Code dann sukzessive so um, dass der TOY-Assemblercode sichtbar wird. Assembliere den TOY-Assemblercode mit *toyasm* oder *atoyasm.py.*
- Beginne dabei mit einem Teil des Codes. Das könnte zum Beispiel das Lesen der Eingabedaten sein.
- Verstehe, wie man den First-In-Last-Out-Puffer programmiert.
- Teste den Maschinen-Code auf Visual X-TOY.

Eine alternative Spezifikation des Problems

```
////////////////////////////////////////////////////////
//  RO 2018
//  Assignment 1: "Addition with reverse polish notation"
//
//  KC Posch, Dec 2017
//
//  Program reads ASCII input from standard input.
//  Program expects only:
//      - ASCII 0x30...0x39 (digits 0...9)
//      - '\n' (0x0A)
//      - '+'  (0x2B)
//
//  In case any other character is found on standard input,
//  the program's function is not defined.
//
//  As soon as the program finds "end of file" (EOF) on
//  standard input, it quits.
//
//  Actions:
//    Program reads ASCII-coded digits from its standard input
//    and interprets those as a decimal number. E.g. 0x31 followed by
//    0x32 is interpreted as (decimal) "12" (= twelve).
//    If this is followed by 0x33, the number becomes
//    (decimal) "123" (one hundred and twenty three).
//
//    This number is stored internally in binary format
//    in a register ("X"). E.g. "12" should be stored as 0xC.
//
//    As soon as the input is a '\n', the number in
//    register X gets copied into a buffer (M).
//    The buffer's maximum size is 16.
//
//    After storing a number, a new number can be entered
//    in the same way as described above.
//
//    With the plus sign ('+'), the last number added to the buffer
//    is added to the number in X; the sum is stored in register X.
//    At the same time, the number is removed from the buffer.
//    Thus, the buffer acts like a stack.
//
//    After adding, the program first prints a '\n' to the standard
//    output; moreover, the value in register X gets converted from
//    binary to decimal and also gets ASCII-printed to standard output.
//
//    All characters read from standard input get written
//    to standard output as soon as they got read.
//
//    Example:
//
//      Consider the following ASCII characters read from standard input:
```

```
//
//      0x31 0x32 0x33 0x0A 0x34 0x35 0x36 0x2B EOF
//
//      This corresponds to the following input:
//
//      123
//      456+
//
//      The output created by the program should then be:
//
//      0x31 0x32 0x33 0x0A 0x34 0x35 0x36 0x2B 0x0A 0x35 0x37 0x39
//
//      On the output screen this looks like this:
//
//      123
//      456+
//      579
//
//    Another example:
//
//      Consider the following ASCII characters arising on standard input:
//
//      0x31 0x0A 0x2B 0x0A 0x2B 0x0A 0x2B 0x0A 0x2B EOF
//
//      This corresponds to the following input:
//
//      1
//      +
//      +
//      +
//      +
//
//      With the characters '\n', the value in X gets copied to the
//      buffer. With '+', the value in the buffer gets added to the
//      value in register X. In this case, we expect the program to
//      produce the following output:
//
//      1
//      +
//      2
//      +
//      4
//      +
//      8
//      +
//      16
//
//  In C, use the data type "int" for storing numbers in
//  X and in the buffer.
//
//  In TOY, you use the 16-bit words for storing numbers.
//
//  If the sizes for the numbers exceed the data type "int"
//  (or in TOY the 16 bits), the programs can behave
//  unexpectedly; thus, we do not check for validity of
//  data.
//
////////////////////////////////////////////////////////
```

Tastatureingabe, Bildschirmausgabe und EOF („End of file“)

Bei vielen Programmen erwarten wir, dass wir beim Drücken der Taste „a“ auf der Tastatur auch am Bildschirm ein „a“ sehen. Dies entspricht der Erwartungshaltung und wurde ursprünglich durch die Verwendung einer mechanischen Schreibmaschine geprägt. Bei genauerer Betrachtung erkennen wir jedoch, dass ein Programm die Entscheidung trifft, den Eingabewert von der Tastatur (Standard-Input) zu erkennen und eventuell einen geeigneten Wert an den Standard-Output zu schicken.

Betrachten wir zum Beispiel die „Shell“, welche mit dem Starten eines Terminal-Fensters ausgeführt wird. Bei mir ist dies zum Beispiel „bash“ („Bourne Again Shell“). Dieses Programm schreibt typischerweise einen „Prompt“ auf den Bildschirm und wartet dann auf Eingabe vom Standard-Input. Dies sind im typischen Fall ein oder mehrere Zeichen von der Tastatur. Jedes Zeichen von der

Tastatur wird vom Programm „bash“ erkannt und üblicherweise als Echo auf dem Standard-Output, also auf dem Bildschirm ausgegeben.

Mit dem Befehl „stty –echo“ kannst du die Echo-Funktion des Programms „bash“ auch ausschalten. Und mit dem Befehl „stty echo“ kannst du die Echo-Funktion wieder einschalten. Probiere es mal aus: Wenn du zu Beispiel das Kommando „pwd“ mit eingeschalteter Echofunktion eintippst, siehst du auch im Terminalfenster die drei eingegebenen Buchstaben „pwd“. Wenn du danach die Return-Taste drückst, wird das Kommando „pwd“ von der Shell ausgeführt und am Standard-Output erscheint das Resultat. Schalte nun die Echo-Funktion durch Tippen von „stty –echo“ gefolgt von „Return“ aus. Als Resultat erscheint der „Prompt“; das bedeutet, dass die Shell auf User-Input wartet. Tippe danach wiederum „pwd“ ein. Die Shell hat jetzt zwar die drei Zeichen „pwd“ vom Standard-Input gelesen, doch kein entsprechendes Echo auf den Standard-Output geschickt. Vorläufig wartet die Shell auf weiteren Input. Wenn du jetzt die „Return“-Taste drückst, erkennt die Shell, dass es den eingebauten Befehl „pwd“ ausführen soll. Als Resultat erscheint das Ergebnis von „pwd“ im Terminal-Fenster. Mit „stty echo“ kannst du die Echofunktion wieder einschalten.

Sehen wir uns ein sehr einfaches C-Programm an, welches dieses Echoverhalten der Shell simuliert:

```
/*****************************************
getchar_putchar_ohne_Ende.c
*****************************************/
#include <stdio.h>

int main() {
     char c;

     while(1) {
       c = getchar();
       putchar(c);
     }
}
```

In einer Endlosschleife liest das Programm ein Zeichen vom Standard-Input und gibt dieses Zeichen am Standard-Output aus. Wenn wir dieses Programm übersetzen und ausführen, dann landen wir in der Endlosschleife – es gibt kein “normales” Ende im Programmcode. Wir können das Programm lediglich durch eine “Signal” des Betriebssystems abbrechen. Das Signal sollte den „Prozess“ – so nennen wir unser „laufendes“ Programm aus der Sicht des Betriebssystems – beenden. Bei meiner Shell kann ich dieses Signal durch die Tastatureingabe „Control-C“ auslösen. In Wikipedia findet man unter https://en.wikipedia.org/wiki/Control-C den folgenden Text:

Control-C is a common computer command. It is generated by pressing the C key while holding down the Ctrl key on most computer keyboards.

In graphical user interface environments that use the control key to control the active program, control-C is often used to copy highlighted text to the clipboard. In many command-line interface environments, control-C is used to abort the current task and regain user control. It is a special sequence which causes the operating system to send a signal to the active program. Usually the signal causes it to end, but the program may "catch" it and do something else, typically returning control to the user.

Wir probieren das Programm einfach aus:

```
$gcc getchar_putchar_ohne_Ende.c
$./a.out
1
1
2
```

```
2
3
3
4
4
^C
$
```

Nach dem Starten des Programms habe ich die Tasten “1”, “2”, “3” und “4” gefolgt von „Control-C“ eingegeben. Wir sehen, dass es zwei Echos gibt: Einmal von der Shell selbst und ein zweites Mal von unserem „putchar(c)“ innerhalb unseres Programms.

Der Abbruch eines Prozesses mit „Control-C“ kann bei Endlosschleifen ganz nützlich sein. Üblicherweise gibt es jedoch einen „geordneten Ausstieg“. Man kann dies zum Beispiel durch die Eingabe eines bestimmten Zeichens machen. Im folgenden Code verwenden wir „E“ als Befehl zur Beendigung des Programms:

```
/*****************************************
getchar_putchar_mit_Ende.c
*****************************************/
#include <stdio.h>

int main() {
  char c;

  while(1) {
     c = getchar();
     if (c == 'E') break;
     putchar(c);
  }
}
```

Wir testen das Programm:

```
$gcc getchar_putchar_mit_Ende.c
$./a.out
1
1
2
2
3
3
E
$
```

Anstatt die Zeichen immer per Hand auf der Tastatur einzutippen, generieren wir eine Datei mit den gewünschten Eingabedaten und verwenden sodann diese beim Testen:

```
$echo "123E"
123E
$echo "123E" > input.txt
$cat input.txt
123E
$hexdump input.txt
0000000 31 32 33 45 0a
0000005
$
$./a.out < input.txt
123$
```

Wir generieren eine Datei mit dem Namen “input.txt”. Der Inhalt dieser Datei besteht aus 4 ASCII-Zeichen (0x31 für „1“, 0x32 für „2“, 0x33 für „3“, 0x45 für „E“) gefolgt von 0x0a (“newline”). Wir erzeugen diese kleine Datei mit dem Kommando “echo”. Du könntest jedoch auch einen normalen Texteditor zur Erzeugung verwenden. Mit dem Kommando „cat“ überprüfen wir den Inhalt der Datei. Mit dem Kommando „hexdump“ können wir den Inhalt der Datei „input.txt“ auch Byte für Byte ansehen. Schließlich leiten wir den Standard-Input von der Tastatur zur Datei „input.txt“ um. Dazu verwenden wir das Zeichen „<“. Als Ergebniserhalten wir „123“.

Wir können den Output unseres Programms ebenfalls in eine Datei umleiten. Dazu verwenden wir das Zeichen „>“:

```
$./a.out < input.txt > output.txt
$cat output.txt
123$
$hexdump output.txt
0000000 31 32 33
0000003
$
```

Anstatt des Zeichens „E“ für die Ende der Eingabe verwendet man bei C-Programmen üblicherweise oft „EOF“ („end of file“). Sehen wir uns folgenden Code an:

```
/*****************************************
getchar_putchar_mit_EOF.c
*****************************************/
#include <stdio.h>

int main() {
     char c;

     while(1) {
       c = getchar();
       if (c == EOF) break;
       putchar(c);
     }
}
```

Wir übersetzen das Programm und testen mit “input.txt”:

```
$gcc getchar_putchar_with_EOF.c
$
$./a.out < input.txt > output.txt
$
$hexdump input.txt
0000000 31 32 33 45 0a
0000005
$
$hexdump output.txt
0000000 31 32 33 45 0a
0000005
$
$cat input.txt
123E
$
$cat output.txt
123E
$
```

Wir sehen, dass alle Zeichen der Datei “input.txt” in die Datei “output.txt” kopiert werden. Vielleicht überrascht dich dieses Ergebnis. Wo ist EOF? Gibt es kein eigenes Zeichen für EOF? Und wie soll man EOF von der Tastatur aus auslösen?

Wir googeln nach „EOF“ und lernen, dass „EOF“ eine Funktion des verwendeten Betriebssystems ist. Nachstehend ein Screenshot von Wikipedia (https://de.wikipedia.org/wiki/End_of_File):

Mit **EOF** (**E**nd **o**f **F**ile) wird das Ende einer Quelle signalisiert, welche in der Regel eine Datei oder ein Datenstrom ist.

In ISO-C können Datei- und IO-Operationen einen Wert zurückgeben, der dem symbolischen EOF entspricht und damit anzeigt, dass das Ende erreicht wurde. Der tatsächliche Wert beträgt häufig –1, dies ist allerdings systemabhängig.

In Unix kann ein EOF über die interaktive Shell (Unix-Shell) durch Strg + D produziert werden (konventioneller Standard). Strg + D entspricht dem ASCII-Steuerzeichen **End of Transmission** 0x04. In Microsofts DOS wird ein EOF mittels Strg + Z erzeugt. Historisch fügte Microsoft DOS das ASCII-Steuerzeichen **SUB** 0x1A (eben Strg + Z) tatsächlich an das Ende einer Textdatei. Die Kompatibilität zu älteren Systemen (z. B. CP/M) wäre sonst nicht zu gewährleisten gewesen. AmigaDOS benutzt Strg + \, das entspricht dem ASCII-Steuerzeichen **File Separator** 0x1C.

In der Job Control Language von Großrechnerumgebungen werden Eingaben über den Standardinput SYSIN mit „/*“ und das Jobende mit „/&“ abgeschlossen.

Wir probieren dies mal auf einem Unix-ähnlichen Betriebssystem aus: Ich gebe „1“, „2“ und „3“ gefolgt von „Control+D“ („Steuerung+D“) ein:

```
$./a.out
1
1
2
2
3
3
$D
$
```

Wie zu erwarten sehen wir das Echo unserer Eingaben gefolgt vom Output unseres Programmes. Und tatsächlich konnten wir mit „Control-D“ das Ende unserer Eingabe (= EOF) auf der Tastatur „eingeben“. Wir lernen damit also, dass EOF in einer Datei kein eigenes Zeichen darstellt, sondern unserem Programm vom Betriebssystem her auf geeignete Weise mitgeteilt wird.

Für die Neugierigen: Wir können das C-Programm kompilieren und uns den vom Compiler erzeugten Assemblercode ansehen. Darin erkennen wir, dass bei Unix-ähnlichen Betriebssystemen der Wert von EOF gleich „-1“ ist.

Bei TOY gibt es kein Betriebssystem. Wir müssen deshalb definieren, wie unser Programm das Ende der Eingabe, also EOF erkennt. Deshalb definieren wir für TOY den Wert von „EOF“ mit dem Hex-Wert 0x03.

dc - an arbitrary precision calculator

In „bash“ und in vielen anderen Shells gibt es Kommandos, welche eine „Taschenrechnerfunktion“ darstellen. „bc“ und „dc“ sind solche Kommandos. „bc“ kennt alle üblichen Operatoren wie Addition, Subtraktion etc. und „versteht“ die sogenannte In-Fix-Notation. Das Kommando „dc“ startet einen „Taschenrechner“, welcher die umgekehrte Polnische Notation versteht.

Du kannst mit dem Konsolenkommando „man bc“ alle Details zu „bc“ erfahren. Nachfolgend ein kleines Beispiel:

```
$bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
123+456
579
^D$
$
```

Nach dem Starten von „bc“ gebe ich den String „123+456“ ein und erhalte nach der Eingabe von „Enter“ die Summe, also „579“.

Wir probieren aus, ob „bc“ auch mit Eingaben aus einer Datei funktioniert. Zuerst erstellen wir eine Textdatei mit dem Inhalt „123+456“. Der Einfachheit halber nehmen wir statt einem Texteditor das Kommando „echo“ und leiten dessen Output in die Datei „bc_input0.txt“ um:

```
$echo "12+34" > bc_input0.txt
$
```

Wir überprüfen den Inhalt der Datei mit dem Namen „bc_input0.txt“ mit dem Kommando „cat“:

```
$cat bc_input0.txt
12+34
$
```

Jetzt starten wir „bc“ und holen die Eingabe aus der Datei „bc_input0.txt“:

```
$bc < bc_input0.txt
46
$
```

Wir sehen, dass „bc“ die Summe von 12 und 34 ausgibt. Perfekt.

Wir können die Ausgabe von „bc“ auch in eine Datei umleiten:

```
$bc < bc_input0.txt > bc_output0.txt
$
```

Welche ASCII-Zeichen finden wir in den beiden Dateien?

```
$hexdump bc_input0.txt
0000000 31 32 2b 33 34 0a
0000006
$hexdump bc_output0.txt
0000000 34 36 0a
0000003
$
```

In der Datei „bc_input0.txt“ sehen wir 0x31 und 0x32 für die Ziffern „1“ und „2“ gefolgt von 0x2b für das Pluszeichen. Schließlich finden wir 0x33 für die Ziffer „3“ und 0x34 für die Ziffer „4“ gefolgt von 0x0A („newline“).

In der Datei „bc_output0.txt“ finden wir das von „bc“ erzeugte Resultat ASCII-codiert in Dezimalform: Es sind die Ziffern „4“ und „6“; und wir „lesen“ „sechsundvierzig“.

Sehen wir uns das Kommando „dc“ an. Vermutlich ist auch dieses Kommando in deiner Shell vorinstalliert. Dieses versteht eine etwas andere Eingabesprache. Statt der In-Fix-Notation von „bc“ erwartet „dc“ eine Post-Fix-Notation. Diese ist auch als „umgekehrte Polnische Notation“ („UPN“, „reverse Polish notation“, „RPN“) bekannt. Statt „123+456“ schreibt man „123 456 +“. „dc“ verlangt zudem ein explizites Kommando für die Ausgabe des Resultates. Dies ist „p“ für „print“. Du findest wie üblich alle Details zu „dc“ mit „man dc“.

Ich probiere „dc“ zunächst mit der Eingabe von der Konsole:

```
$dc
123
456
+
p
579
$
```

Als nächstes versuche ich es mit der Umleitung der Eingabe. Ich generiere zunächst eine Datei mit dem Namen „dc_input.txt“. Dazu verwende ich, wie schon vorhin beim Beispiel mit „bc“ gezeigt, das Kommando „echo“:

```
$echo "123
> 456
> +
> p
> " > dc_input0.txt
```

Sehen wir uns den Inhalt der Datei mit “cat” an:

```
$cat dc_input0.txt
123
456
+
p
```

Mit “hexdump” können wir uns auch den genauen Inhalt der Datei ansehen. Wir finden alle Zeichen in ASCII-kodierter Form:

```
$hexdump dc_input0.txt
0000000 31 32 33 0a 34 35 36 0a 2b 0a 70 0a 0a
000000d
```

Wir starten “dc” und übergeben den Inhalt der Datei mit dem Namen “dc_input0.txt”:

```
$dc < dc_input0.txt
579
```

Das hat offensichtlich funktioniert. Wir finden das Resultat “fünfhundertneunundsiebzig” auf der Konsole. Doch in Wirklichkeit sehen wir nur die Ziffern “5”, “7” und “9”. Die dezimale Interpretation entsteht in unserem Kopf.

Sehen wir nach: Wir leiten die Ausgabe von “dc” in die Datei “dc_output0.txt” um:

```
$dc < dc_input0.txt > dc_output0.txt
```

Schließlich sehen wir uns den Inhalt dieser Datei mit “hexdump” an:

```
$hexdump dc_output0.txt
0000000 35 37 39 0a
0000004
$
```

Das Programm “dc” hat lediglich drei ASCII-kodierte Ziffern gefolgt von “newline” ausgegeben.

Offensichtlich haben sowohl “bc” als auch “dc” zunächst eine Folge von ASCII-kodierten Ziffern als Zahl “verstanden”, danach mit diesen Zahlen eine Addition durchgeführt, und schließlich das

Ergebnis in ASCII-kodierter Form als Dezimalzahl wieder ausgegeben. Soviel zum Hintergrund der Aufgabenstellung dieser Aufgabe.

ASCII-kodierte Dezimalzahlen und deren Binäräquivalent

Wenn jemand die Ziffer „5“ auf Papier schreibt, dann liest man „fünf“. Wenn man dann rechts von dieser Ziffer die Ziffer „6“ dazu schreibt, dann liest man, sofern es sich um das Dezimalsystem handelt, die Zahl „sechsundfünfzig“; also 5 mal 10 plus 6. Diesen „Denkprozess“ musst du in deinem Programm nachvollziehen.

Umwandlung einer Binärzahl in eine ASCII-kodierte Dezimalzahl

In dieser Aufgabe sollst du Zahlen in Binärform speichern. Die Dezimalzahl „56“ ist ja hexadezimal 0x38. Dies lässt sich als 32-Bit-Binärzahl so darstellen:
0000_0000_0000_0000_0000_0000_0011_1000.

Du kannst dies mit folgendem Programm überprüfen:

```
/**********************************
 binary_56.c
**********************************/
#include <stdio.h>

int a = 56;

int main() {
    printf("The binary pattern of %d looks like this:\n", a);

    for (int i = 31; i >= 0; i--) {
        if (a >> i & 0x1)
            putchar('1');
        else
            putchar('0');

        if ((i%4==0) && (i!=0))
            putchar('_');
    }
    putchar('\n');
}
```

In TOY arbeiten wir mit 16-Bit-Zahlen. Die Binärdarstellung von dezimal 56 sieht dann also so aus: 0000_0000_0000_0011_1000.

In dieser Aufgabe musst du also einen Weg finden, wie du aus 0x38 die Ausgabe von zwei ASCII-Zeichen erzeugst: 0x33 und 0x38.

Vielleicht hilft dir der folgende Hinweis: Mit 16 Bit können wir (bei vorzeichenloser Darstellung) die Zahlen von 0 bis $2^{16} - 1$ darstellen. Dies sind alle Zahlen von 0 bis irgendwas über 65000. Als erste Ziffer musst du also – im Falle, dass die Zahl größer als dezimal 10000 ist, eine Ziffer aus der Menge {1,2,3,4,5,6} in ASCII-kodierter Form ausgeben. Falls die Zahl kleiner dezimal 10000 ist, soll das Programm keine Ziffer ausgeben. In TOY stehen lediglich Addition und Subtraktion als Maschinenbefehle zur Verfügung.

Standard-Testfälle

Zusammen mit diesem Dokument kannst du auch 6 Testfälle für Aufgabe 1 von der Course-Webpage runterladen. Du solltest im Verzeichnis „test_inputs“ folgende Dateien anfinden:

```
$cd test_inputs
$ls
input_rpn0.txt  input_rpn1.txt  input_rpn2.txt  input_rpn3.txt  input_rpn4.txt
     input_rpn5.txt
$
$cat input_rpn0.txt
123
456+$
$
$cat input_rpn1.txt
1
+
+
+
+$
$
$cat input_rpn2.txt
100
1000+$
$
$cat input_rpn3.txt
100
1000+
123
456+
1
2+$
$
$cat input_rpn4.txt
1
2
3
4
5
6
7
8+++++++$
$
$cat input_rpn5.txt
1
+
+
+
+
+
+
+$
$
$
```

Im Verzeichnis „expected_outputs“ findest du die richtigen Ausgaben zu obigen Eingaben:

```
$cd expected_outputs/
$ls
expected_output_rpn0.txt  expected_output_rpn2.txt  expected_output_rpn4.txt
expected_output_rpn1.txt  expected_output_rpn3.txt  expected_output_rpn5.txt
$
$cat expected_output_rpn0.txt
123
456+
579
$
$cat expected_output_rpn1.txt
1
+
2
+
4
```

```
+
8
+
16
$
$cat expected_output_rpn2.txt
100
1000+
1100
$
$cat expected_output_rpn3.txt
100
1000+
1100
123
456+
579
1
2+
3
$
$cat expected_output_rpn4.txt
1
2
3
4
5
6
7
8+
15+
21+
26+
30+
33+
35+
36
$
$cat expected_output_rpn5.txt
1
+
2
+
4
+
8
+
16
+
32
+
64
+
128
$
$
```

Es gibt zudem auch ein Verzeichnis mit dem Namen „toy_test_inputs“. In diesem Verzeichnis gibt es die äquivalenten Testdaten für den Standard-Input von TOY:

```
$cd toy_test_inputs/
$ls
toy_input_rpn0.txt   toy_input_rpn2.txt   toy_input_rpn4.txt
toy_input_rpn1.txt   toy_input_rpn3.txt   toy_input_rpn5.txt
$
$cat toy_input_rpn0.txt
0031
0032
```

```
0033
000a
0034
0035
0036
002b
0003$
$
```

Wozu Aufgabe 1?

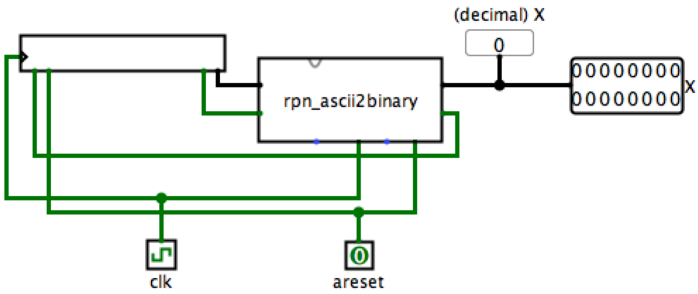
Der TOY-Computer wird in der Lehrveranstaltung als durchgehendes Beispiel verwendet, um über die Organisation von Rechenmaschinen, also von Computern zu sprechen. TOY ist sehr einfach und ein guter Startpunkt ist die Beschäftigung mit dem TOY-Simulator *Visual X-TOY*. Mit dem Wissen, welches du in der ersten Aufgabe kriegst, versetzt du dich in die Lage, viel besser den nachfolgenden Themen folgen zu können. Wir werden in der Vorlesung den TOY-Computer und dabei speziell die TOY-CPU bis zur Logik-Ebene hin entwerfen.

In diesem Beispiel beschäftigst du dich auch mit der Übersetzung zwischen C-Code und Assemblercode. Damit lernst du einige typischen Schritte dieser Übersetzung kennen; diese übernimmt ja zumeist der Compiler. Du wirst das in dieser Aufgabe erworbene Wissen bei den Aufgaben 2, 3 und 4 dieser Übung brauchen. Auch dort beschäftigen wir uns mit dem gleichen Problem.

Als Ausgangspunkt der Aufgabe 1 musst du dir grundlegende Gedanken über die Darstellung von Information machen.

Aufgabe 2:

Logiksimulation mit Logisim: „rpn_ascii2binary“



(Einzelarbeit)

In dieser Aufgabe beschäftigst du dich mit der Simulation digitaler Schaltungen auf Registertransferebene und Logikebene. Die Aufgabe besteht aus mehreren Teilen.

Zuerst sollst du dich in das Werkzeug „Logisim“ einarbeiten. Sinnvolle Aufgabenstellungen für diese ersten Aktivitäten mit Logisim bestehen in der Realisierung einfacher synchroner Automaten und deren Simulation. Sobald du klar verstanden hast, wie man ausgehend von einem ASM-Diagramm einen synchronen Automaten bestehend aus den Teilen „Datenpfad“ und „Kontrolllogik“ entwickelt, in Logisim eingibt, und danach simuliert, kannst du dich dem zweiten Teil widmen.

Der zweite Teil dieser Aufgabe knüpft an deine Arbeit in Aufgabe 1 an. Du sollst einen synchronen Automaten bauen, welcher die von einer Tastatur ausgegebenen ASCII-Daten verarbeitet und deren Ergebnis X in Binärform ausgibt. Die Semantik der Eingabe entspricht – wie schon bei Aufgabe 1 gemacht – der umgekehrten Polnischen Notation. Der Automat soll Additionen ausführen können.

Als Eingabegerät soll das in Logisim vorhandene Bauteil „Keyboard“ verwendet werden. Dein Automat soll die in diese Tastatur eingegebenen Werte der Reihe nach „abholen“, diese entsprechend der umgekehrten Polnischen Notation interpretieren und das Resultat in 16-Bit-Unsigned-Format ausgeben. Der Automat muss folgende Zeichen von der Tastatur sinnvoll erkennen: Die Ziffern 0 bis 9, „newline“ und „+“. Alle anderen Tastatureingaben soll der Automat ignorieren.

Beispiel: Die Eingabefolge „1“, „2“, „3“ und „newline“ soll als „einhundertdreiundzwanzig“ in Binärform im Register X gespeichert werden. Im Register X befindet sich nach der Eingabe also das Binärmuster „0000_0000_0111_1011“. Wenn danach von der Tastatur die ASCII-Codes „4“, „5“, „6“ gefolgt von „+“ kommen, dann sollte im Register X der Binärwert für die Dezimalzahl „fünfhundertneunundsiebzig“ erzeugt werden. Im Register X befindet sich danach also die Binärzahl „0000_0010_0100_0011“.

Wie bei Aufgabe 1 brauchst du also auch hier ein zweites Register, in welchem die zuerst eingegebene Zahl gespeichert wird. In Aufgabe 1 war dies eine First-In-Last-Out-Puffer, in welchem man bis zu 16 Werte speichern konnte. In diesem Beispiel kann dieser Puffer lediglich 1 Wert speichern.

Strukturiere deinen Automaten in Datenpfad und Kontrolllogik. Die Zustände der Kontrolllogik sollen binärkodiert sein. Im Bild oben siehst du, wie der Top-Level-View deiner Schaltung aussehen sollte.

Zusammenfassung:

- Dein synchroner Automat "rpn_ascii2binary" hat also folgende Inputs und Outputs:
  - Input von Tastatur: ASCII-Zeichen und das 1-Bit-Kontrollsignal „available". Input von der Tastatur in ASCII-Format: Ziffern 0-9, newline (0x0A) oder Pluszeichen (0x2B).
  - Output zur Tastatur: "read_enable".
  - Output: binäre 16-Bit-Zahl in vorzeichenlosem Format, welche das Rechenergebnis darstellt.
- Es gibt ein Register X und ein Register M. X und M können je eine 16-Bit-Zahl in vorzeichenlosem Format speichern.
- Syntax für Input wie bei Beispiel 1: "Reverse Polish Notation"; nur Addition.

Abzugebende Dateien

Abzugeben sind folgende Dateien, welche zusammen in eine ZIP-Datei gepackt werden. Der Name der ZIP-Datei entspricht deiner Matrikelnummer gefolgt von der Dateierweiterung „.zip"; also zum Beispiel so: „1234567.zip". Folgende Dateien sollen in der ZIP-Datei enthalten sein:

- Dein Logisim-Modell: Dateiname ist Matrikelnummer-dot-circ, also z.B. „1234567.circ".
- Die PDF-Datei „1234567.pdf" mit dem ASM-Diagramm deines Automaten.

Spätester Abgabetermin

- Freitag, 27. April 2018, 14:00.
- Abgabe über das „Student Tick System" (STicS) unter https://stics.iaik.tugraz.at. In besonderen Fällen darfst du deine Abgabe auch per E-Mail mit dem Betreff „[RO] Abgabe 2" an Karl.Posch@iaik.tugraz.at machen.

Vorgangsweise

- Vergiss nicht, Eintragungen in dein Ingenieurstagebuch zu machen. Datum und Zeitaufwand sind wichtig.
- Studiere die Unterlagen. Dabei sind speziell der Vorlesungstexte „Primer on Logic Functions", „Primer on Finite State Machines" und „Primer on Algorithmic State Machines" wichtig.
- Probiere die in den Unterlagen beschriebenen Beispiele in Logisim aus.
- Verstehe, was es bedeutet, eine „synchrone Schaltung" zu bauen.
- Lerne mit den Begriffen, welche in den Unterlagen vorkommen, umzugehen.

- Verstehe den Zusammenhang und den Unterschied zwischen Modellierung auf Registertransferebene (RTL) und Logikebene.

Wozu Aufgabe 2?

Mit dem Simulator Logisim lernst du „digitale Hardware von unten“ her kennen. Ausgehend von einfachen Elementen wie logischen Gattern, Multiplexern und den dazu gehörigen Wahrheitstabellen sollst du größere Schaltungen „zusammen löten“ (so hätte man dies vor vielen Jahren ausgedrückt). Logisim ist vom Prinzip her sehr einfach und betont die strukturelle Bottom-Up-Sichtweise von digitalen Schaltungen. Die Beschäftigung mit Logisim soll dir helfen, die Grundelemente von digitalen Schaltungen kennen zu lernen und diese auch zu verwenden.

Ziel dieser Übung ist, ein Verständnis für den Aufbau synchroner digitaler Automaten zu entwickeln. Um einen speziellen Automaten zu entwerfen, beginnt man am besten mit einem Modell in einer Sprache wie z.B. C. Aus diesem Modell entwickelt man schrittweise den Automaten auf Registertransferebene bzw. Logikebene.

Bei der Umwandlung eines funktionalen Modells in eine Registertransferdarstellung bzw. in ein Logikmodell lernst du Schaltungen kennen, welche Bits speichern können (Flipflop, Register) oder auch logische Schaltungen, welche einfache arithmetische Operationen ausführen können. Dieses Wissen hilft dir, die Verwendung der Hardware-Modellierungssprache Verilog besser zu verstehen. Das Werkzeug Logisim solltest du als Lernwerkzeug betrachten.

Materialien zu Aufgabe 2

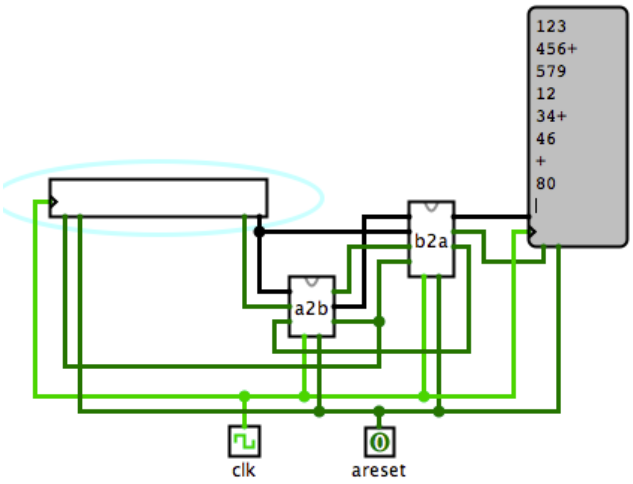
Folgende Materialien brauchst du:

- Die Vorlesungsunterlagen https://teaching.iaik.tugraz.at/ro/start, speziell die Materialen der ersten 5 Wochen des Semesters.
- Logisim (http://ozark.hendrix.edu/~burch/logisim/)

Aufgabe 3:

Kontrolllogik mit One-Hot-Encoding:

„rpn_ascii2ascii“



(Einzelarbeit)

„binary2ascii“: Im ersten Schritt zu Aufgabe 3 sollst du einen synchronen endlichen Automaten bauen, welcher eine vorzeichenlose 16-bit-Binärzahl in deren ASCII-kodiertes Dezimaläquivalent umformt und diese Dezimalzahl auf einer Konsole ausgibt. Dieser Automat besteht aus Datenpfad und Kontrolllogik. Die Kontrolllogik soll mit One-Hot-Encoding realisiert werden. Da die maximal darstellbare Binärzahl etwas über 65000 ist, kann die Dezimalstelle mit dem Wert 10000 lediglich eine der Ziffern 0, 1, 2, 3, 4, 5 oder 6 sein. Durch (eventuell mehrmalige) Subtraktion kannst du diese Ziffer ermitteln. Auf ähnliche Weise kannst du die 1000er-Stelle durch (eventuell mehrmalige) Subtraktion ermitteln. Wie üblich, sollen führende Nullen einer Zahl nicht angezeigt werden. Der Automat besitzt folgende Eingänge: Die 16-Bit-Binärzahl X und das Kontrollsignal „start“. Als Ausgänge gibt es 7 Bits für die ASCII-kodierte Ziffer („digit“) zusammen mit dem Signal „putchar“, welches für einen Taktzyklus aktiv wird, sobald eine neue Ziffer am Ausgang „digit“ erscheint. Damit kannst du dieses ASCII-Zeichen auf der Konsole ausgeben. Zusätzlich liefert der Automat einen Kontrollausgang „ready“. Dieser wird aktiv, sobald die Umwandlung einer Binärzahl fertig ist.

„rpn_ascii2binary_mod“: Im zweiten Schritt sollst du deinen Automaten aus Aufgabe 1 umbauen, so dass dessen Ausgabe X als Eingabe von „binary2ascii“ verwendet werden kann. „binary2ascii“ braucht ja als Eingangssignal „start“ und aktiviert bei Beendigung der Umwandlung einer Binärzahl in deren dezimales ASCII-kodiertes Äquivalent seinen Ausgang „ready“. Der modifizierte Automat „rpn_ascii2binary_mod“ muss also im richtigen Taktzyklus „start“ aktivieren und dann auf die Aktivierung von „ready“ warten.

„rpn_ascii2ascii“: In einem dritten Schritt kannst du dann „rpn_ascii2binary_mod“ und „binary2ascii“ hintereinander schalten. Dieser Automat kann sodann ASCII-kodierte Tastatureingaben lesen und das Ergebnis in ASCII-kodierter Form ausgeben.

Auf dem Bild oben siehst du den Konsolen-Output, welcher die folgende Tastatur-Eingabe verarbeitet hat: „1“, „2“, „3“, „newline“, „4“, „5“, „6“, „+“, „1“, „2“, „newline“, „3“, „4“, „+“, „+“. Bei dieser Schaltung wurde zusätzlich zum Output von „binary2ascii“ die unveränderten Tastatureingaben „dazu gemischt“. Dies entspricht also einem einfachen Taschenrechner, ganz ähnlich wie schon bei Aufgabe 1 realisiert.

Solltest du Schwierigkeiten haben, sowohl Tastatureingaben wie auch Ergebnis auf einer Konsole darstellen zu können, so ist es OK, wenn du mit 2 Konsolen arbeitest: Auf einer Konsole gibst du alle Tastatureingaben aus und auf der anderen Konsole gibst du die Ausgaben von „binary2ascii“ aus.

Abzugebende Dateien

Abzugeben sind folgende Dateien, welche zusammen in eine ZIP-Datei gepackt werden. Der Name der ZIP-Datei entspricht deiner Matrikelnummer gefolgt von der Dateierweiterung „.zip“; also zum Beispiel so: „1234567.zip“. Folgende Dateien sollen in der ZIP-Datei enthalten sein:

- Dein Logisim-Modell „binary2ascii“: Dateiname ist Matrikelnummer gefolgt von „_b2a.circ“, also z.B. „1234567_b2a.circ“.
- Dein Logisim-Modell „rpn_ascii2binary“: Dateiname ist Matrikelnummer gefolgt von „_a2b.circ“, also z.B. „1234567_a2b.circ“.
- Dein Logisim-Modell „rpn_ascii2ascii“: Dateiname ist Matrikelnummer gefolgt von „_a2a.circ“, also z.B. „1234567_a2a.circ“.
- Die PDF-Datei „1234567.pdf“ mit den ASM-Diagrammen deiner beiden Automaten „binary2ascii“ und „rpn_ascii2binary_mod“.

Spätester Abgabetermin

- Montag, 14. Mai 2018, 14:00.
- Abgabe über das „Student Tick System“ (STicS) unter https://stics.iaik.tugraz.at. In besonderen Fällen darfst du deine Abgabe auch per E-Mail mit dem Betreff „[RO] Abgabe 3“ an Karl.Posch@iaik.tugraz.at machen.

Vorgangsweise

- Vergiss nicht, Eintragungen in dein Ingenieurstagebuch zu machen. Datum und Zeitaufwand sind wichtig.
- Entwickle ein Verständnis für das Standardschema für Kontrolllogik mit One-Hot-Encoding.
- Entwickle ein ASM-Diagramm, welches eine vorzeichenlose 16-Bit-Binärzahl in deren Dezimaläquivalent umwandelt.
- Implementiere den Datenpfad dieses ASM-Diagramms.
- Implementiere die verbleibende Kontrolllogik mit One-Hot-Encoding.
- Baue Datenpfad und Kontrolllogik zusammen und teste die Schaltung mit verschiedenen Binäreingaben. Du kannst die „digits“ auf einer Konsole anzeigen.

- Modifiziere das ASM-Diagramm aus Aufgabe 2, sodass es ein Signal „start“ ausgibt und auf das Signal „ready“ reagiert.
- Modifiziere die Implementierung deiner Lösung zu Aufgabe 2.
- Schalte die beiden Automaten „in Kette“ und teste das Ergebnis verschiedener Tastatureingaben.
- Optional: Überlege, wie du die „rohen Tastatureingaben“ zusammen mit den „digits“ von „binary2ascii“ mit einem Multiplexer „mischen“ kannst. Als dritte Komponente könntest du auch ein „newline“ zu geeigneten Zeitpunkten per Multiplexer „dazu mischen“.

Wozu Aufgabe 3?

Aufgabe 3 gibt dir die Gelegenheit, Kontroller mit One-Hot-Encoding zu studieren. Zusätzlich siehst du, wie man mehrere Automaten verbinden kann. Mit Hilfe von Handshake-Signalen kann ein Automat Aktionen bei einem anderen „starten“ bzw. auf dessen „ready“ warten. Diese Situationen gibt es zwischen der Tastatur und „rpn_ascii2binary_mod“ und auch zwischen „rpn_ascii2binary_mod“ und „binary2ascii“.

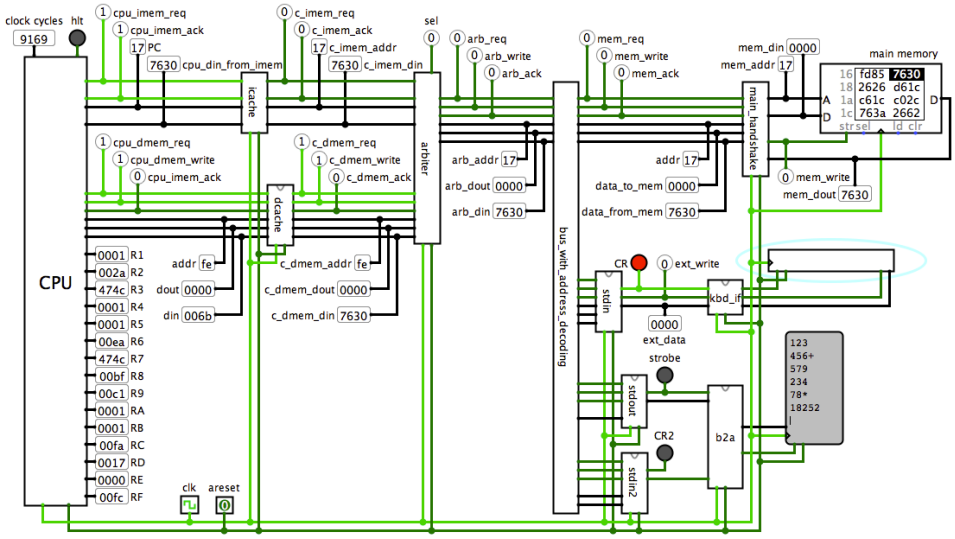
Materialien zu Aufgabe 3:

Folgende Materialien brauchst du:

- Die Vorlesungsunterlagen https://teaching.iaik.tugraz.at/ro/start, speziell die Materialen der ersten 5 Wochen des Semesters.
- Logisim (http://ozark.hendrix.edu/~burch/logisim/)

Assignment 4:

„TOY Deluxe“ meets „binary2ascii“



(Gruppenarbeit)

- **Organisatorisches zum Beginn:**
  Der innerhalb der Gruppe geplante Prozess zur Lösung der Aufgabe, die geplante Arbeitsaufteilung innerhalb der Gruppe, die Abweichungen zwischen Realität von Plan sowie die erzielten Resultate sind zu dokumentieren und sind Bestandteil der Abgabe. Am besten ist es, wenn ihr die Ergebnisse der einleitenden Diskussion zu organisatorischen Fragen schriftlich dokumentiert und jedes Gruppenmitglied sich per Unterschrift sich zu diesem Dokument bekennt.

- **Und jetzt zum technischen Teil:**
  Schließe die Logisim-Tastatur über ein „Keyboard-Interface“ an den Standard-Input des TOY-Computers an. Verbinde den TOY-Computer mit dem Automaten „binary2ascii“ über Standard-Input/Output. „binary2ascii“ funktioniert ähnlich wie in Aufgabe 3. Als Basis für die Modelle dient der TOY-Computer mit der Deluxe-CPU (Modell von „TOY Deluxe: A Pipelined TOY“, Unterkapitel 11; Logisim-Datei „pipelined_toy_with_io_caches_arbiter_and_mainmemory.circ“).

  Das TOY-Programm ist ein einfacher Taschenrechner für ganzzahlige vorzeichenlose Zahlen, welcher die umgekehrte Polnische Notation versteht. Als Operatoren gibt es Addition und Multiplikation. Der Puffer soll – wie schon bei Aufgabe 1 – in der Lage sein, 16 Werte zu speichern. Die darstellbaren Zahlen sind im Intervall $[0, 2^{15}-1]$. Ein Überlauf dieses Intervalls muss vom Programm nicht erkannt werden.

  Das Programm kriegt von der Tastatur ASCII-kodierte Zeichen. Das Programm schickt an den Standard-Output binär kodierte Zahlen im Intervall $[0, 2^{15}-1]$ sowie die Zeichen ‘+’, ‘*’ und ‘\n’. Alle binär kodierten Zahlen im darzustellenden Intervall haben eine führende Null. Um das Pluszeichen von einer Zahl unterscheiden zu können, kodieren wir dieses mit 0x802B. Dieses Bitmuster hat eine führende Null. In gleicher Weise kodieren wir das Multiplikationszeichen mit 0x802A und das Newline-Zeichen mit 0x800A.

  Dein Programm soll Unterprogrammaufrufe für die Funktionen „getchar()“ und „printf()“ beinhalten. Für den Aufruf dient die TOY-Instruktion „Jump and Link“ (JL). Für die Rückkehr aus dem Unterprogramm dient die TOY-Instruktion „Jump Register“ (JR). Die Parameterübergabe soll

über den Stack gemacht werden. Als Stack-Pointer dient das CPU-Register RF. Da TOY-Deluxe keine Push- und Pop-Operationen kennt, musst du diese mit den Instruktionen ADD, SUB, STI und LDI per Software „nachbauen“. Baue den Stack entsprechend der üblichen Konventionen für den Stack-Frame einer Funktion: Stack-Pointer (CPU-Register RF), Base-Pointer (CPU-Register RE), Return-Adresse am Stack, Base-Pointer-Wert der aufrufenden Funktion am Stack, lokale Variablen am Stack. Das Register RD soll für den Return-Wert bei den Instruktionen „Jump and Link“ und „Jump Register“ verwendet werden.

Deine Funktion „getchar()“ soll den Wert im CPU-Register RC zurückgeben. Innerhalb der Funktion soll auch das Polling stattfinden.

Deine Funktion „printf()“ soll bei deren Aufruf folgende Parameter verwenden: Der erste Parameter ist ein Pointer auf einen 16-Bit-Wert im Hauptspeicher. Dieser 16-Bit-Wert entspricht dem zu druckenden Zeichen. Der zweite Parameter ist ein Integer-Wert, welcher entweder 0 oder 1 ist. Damit kannst du beim Aufruf von „printf“ sagen, ob nach dem Drucken des Wertes ein Newline-Zeichen folgen soll oder nicht. Die Funktion hat also folgende Signatur: „printf(int* value, int nl)“. Die Funktion „printf()“ soll auch das Polling beinhalten, mit welchem die Beendigung des Druckens ermittelt wird.

Fertige einen Bericht[1] an, welche die Ergebnisse der Arbeiten an Aufgabe 4 präsentiert.

Abzugebende Dateien

Abzugeben sind folgende Dateien, welche zusammen in eine ZIP-Datei gepackt werden. Der Name der ZIP-Datei ist „RO_Aufgabe4_<XXX>.zip; für die Gruppe „hadzic01“ soll diese Datei also beispielsweise so heißen: „RO_Aufgabe4_hadzic01.zip“. Die ZIP-Datei soll folgende Dateien enthalten:

- Der kommentierte und optimierte Assemblercode. Dateiname ist der Gruppenname, gefolgt von dot-asm, also z.B. „hadzic01.asm“.
- Die Maschinenprogramme „hadzic01.toy“.
- Die Logisim-Versionen des Maschinenprogramms: „hadzic01_image“.
- Dein Logisim-Modell. Dateiname ist der Gruppenname gefolgt von dot-circ, also z.B. „hadzic01.circ“.
- Eine Dokumentation zu Aufgabe 4. Dateiname ist der Gruppenname, dann „dok“ und schließlich „dot“ und „pdf“, also z.B. „hadzic01_DOK.pdf“.

Spätester Abgabetermin

- Freitag, 8. Juni 2018,14:00.

[1] Details über die Form dieses Berichtes findest du auf Seite 34.

Abgabe über das „Student Tick System“ (STicS) unter https://stics.iaik.tugraz.at. In besonderen Fällen darfst du deine Abgabe auch per E-Mail mit dem Betreff „[RO] Abgabe 4“ an Karl.Posch@iaik.tugraz.at machen.

Vorgangsweise

- Vergiss nicht, Eintragungen in dein persönliches Ingenieurstagebuch zu machen. Datum und Zeitaufwand sind wichtig. Diese Eintragungen sind für die Erstellung des Berichts zu Aufgabe 4 vermutlich sehr nützlich.
- Setze dich rechtzeitig mit der Erstellung des Berichtes auseinander. Als Ausgangspunkt dazu kannst du den Text auf Seite 34 nehmen.
- Setze dich mit „TOY Deluxe“ auseinander. Dazu studierst du am besten die Unterkapitel 7 bis 11 der Materialien „TOY Deluxe: A Pipelined TOY“. Mache zuerst die in diesen Materialien beschriebenen Experimente, damit du die Eigenheiten einer Pipeline verstehst.
- Erweitere „TOY Deluxe“ mit einer „Keyboard-Interface-Logik“, damit du ASCII-kodierte Zeichen ähnlich wie mit „getchar()“ in Aufgabe 1 von einem Keyboard einlesen kannst. Dabei wirst du auf das Problem stoßen, wie ein TOY-Programm erkennen kann, dass ein Zeichen vom Keyboard vorhanden ist. Studiere dazu das Thema „Polling“ aus den Vorlesungsunterlagen. Du wirst auch auf das Problem stoßen, dass du Daten, welche vom Standard-Input eingelesen werden, nicht in den Cache kopieren darfst. Studiere dazu ebenfalls die Vorlesungsunterlagen.
- Hänge den Automaten „binary2ascii“ an den Standard-Output von „TOY Deluxe“ an und verbinde diesen Automaten mit einer Konsole. Du wirst dabei auf das Problem stoßen, wie dieser Automat dem TOY-Computer mitteilt, dass er mit der Umwandlung einer Binärzahl in deren ASCII-kodiertes Dezimaläquivalent fertig ist. Dazu musst du den TOY-Computer mit einem zweiten Standard-Input-Gerät ausstatten.
- Entwickle dein Programm Schritt für Schritt und teste alle Zwischenschritte auf deinem modifizierten „TOY Deluxe“. Du arbeitest in dieser Übung an der Schnittstelle zwischen Hardware und Software. Du siehst, wie man Software durch Hardware ersetzen kann: In Beispiel 1 hast du die Umwandlung einer Binärzahl in deren Dezimaläquivalent per Software gemacht. In diesem Beispiel verwendest du dazu einen Hardware-Automaten. Du siehst aber auch, wie man fehlende Hardware-Funktionen, wie etwa die in der CPU von TOY-Deluxe nicht eingebauten Instruktionen CALL, RET, PUSH und POP, durch andere vorhandene CPU-Instruktionen, also durch Software „emulieren“ kann.

Wozu Aufgabe 4?

Du kannst im Rahmen der Arbeit bei dieser Aufgabe Fertigkeiten in vielfacher Hinsicht entwickeln.

(1) Du lernst bei dieser Aufgabe, wie man Modifikation bei einem bestehenden Modell vornimmt. Dies ist sehr oft üblich. Man kommt als IngenieurIn neu in ein Team und als erstes Projekt muss man sich oft in eine bestehende Umgebung einarbeiten und danach ein paar technische Veränderungen

oder eine Weiterentwicklung vornehmen. Dieser Sachverhalt ist in dieser Übung vorgegeben. Man startet also nicht bei „Null“, sondern man macht eine Variante von etwas Bestehendem. Das „Einarbeiten“ in eine neue Umgebung funktioniert am besten, wenn man eine gute Dokumentation des Bestehenden vorfindet. Durch die eigenen Erweiterungen entsteht die „Umgebung“ für die Zukunft. Daraus resultiert auch die Notwendigkeit der Dokumentation der eigenen Arbeiten.

(2) Diese Aufgabe sollst du in einer Gruppe bestehend aus 3 Personen durchführen. Dabei soll es keine Arbeitsteilung geben. Alle sollen alles machen und alles verstehen. Als Vergleich verwende ich hier eine Bergsteigergruppe. Die 3er-Gruppe geht auf einen Berg. Es müssen also alle hinauf. Es gilt nicht, dass nur einer rauf geht und den anderen danach Bilder vom Gipfel zeigt. Und diese dann womöglich auch noch daran glauben, dass sie auch oben waren. Warum also dann eine Gruppe, wenn eh jeder hinauf muss? Ja warum? Wenn eine/r Schwierigkeiten beim Aufstieg hat, dann helfen die anderen. Wenn eine/r Motivationsschwächen hat, dann helfen die anderen. Wenn eine/r sich weh tut, dann helfen die anderen. Und es ist meist lustiger, gemeinsam den Ausblick zu genießen und darüber zu sprechen. Doch kommen wir zurück zur Realität: Als Ingenieur arbeitet man – ich würde sagen – nie alleine. Man arbeitet immer im Kontext einer Gruppe. Und es ist wichtig, zu üben, wie man in einer Gruppe agiert. Es ist wichtig, zu sehen, dass andere anders arbeiten als man es selbst macht. Schnelle kommen drauf, dass es Langsamere gibt und umgekehrt. Verlässlichere kommen drauf, dass es Unzuverlässige gibt – und umgekehrt. Es ist wichtig, früh im Leben damit umgehen zu lernen. Es ist wichtig, zu lernen, Abmachungen mit anderen zu treffen. Aus Gruppenerfolgen und auch aus Gruppenmisserfolgen lernt man und wird Stück für Stück professioneller.

(3) Du beschäftigst dich in dieser Aufgabe mit einem typischen Problem, welches in allen Computern vorkommt: Wie kriegt man Daten in einen Computer „hinein“ und wie kriegt man Resultate wieder „hinaus“. Zusätzlich hast du Gelegenheit, dich mit der Pipeline-Version der TOY-CPU samt Cache und vielen anderen typischen Hardware-Komponenten auseinanderzusetzen. Dieses Wissen wird dir nützen, moderne Computersysteme samt deren Betriebssystem und allen damit verbundenen Problemen verstehen zu lernen. Auch Begriffe wie „Meltdown“ und „Spectre“ werden dir damit leichter zugänglich erscheinen. Am Institut für Angewandte Informationsverarbeitung und Kommunikationstechnologie (IAIK) beschäftigen wir uns recht intensiv mit IT- Sicherheit. Das Beispiel soll dir Lust auf mehr zu diesem Thema machen.

Materialien zu Aufgabe 4:

- Die Vorlesungsunterlagen zu „TOY Deluxe“ (Kapitel 17) und „Cache-Speicher (Kapitel 10). Du findest diese unter https://teaching.iaik.tugraz.at/ro/start.

Abgabegespräche

Es gibt 2 Abgabegespräche. Eines nach der Abgabe der 2. Aufgabe und ein zweites am Ende der Übung. Du triffst dich dabei zusammen mit deinen Gruppenmitgliedern mit dem/der TutorIn oder mit dem Lehrer. Typischerweise dauert so ein Gespräch eine halbe Stunde. Du machst zusammen mit deinen Gruppenmitgliedern mit dem/der TutorIn oder dem Lehrer einen individuellen Termin für dieses Gespräch aus.

Neben der Leistungsüberprüfung sollte mit der Möglichkeit der Präsentation ein positiver Stimulus geschaffen werden: Die aktive Rolle der Studierenden sollte in den Vordergrund gerückt werden. Bei offensichtlichem Leistungsdefizit sollte dem Studierenden ein „Warnschild“ signalisiert werden.

Die (positive und/oder negative) Kritik zur Präsentation beim ersten Abgabegespräch dient zweierlei:

- Verbesserung bzw. Korrektur der Präsentation mit dem Ziel, bei der zweiten Präsentation am Ende des Kurses besser abzuschneiden. Eine nicht gelungene Präsentation sollte bei vorhandenem Wissen zu keinem Punkteabzug führen.
- Beurteilung des Detailwissens über die Aufgaben. Bewertet wird auch die Existenz von Teillösungen. Dies mündet in der Anzahl der Punkte für die Note.

Beim zweiten Abgabegespräch gibt es folgende Aspekte:

- Die Studierenden haben die Chance, durch eine entsprechende Präsentation ihrer Arbeit aktiv zu bestimmen, welchen Verlauf das Gespräch nimmt.
- Die Möglichkeit zur Präsentation der Arbeit sollte von den Studierenden als Chance begriffen werden, ihren Freiraum zu erhöhen. Eine nicht gelungene Präsentation führt zu keinem Punkteabzug, jedoch zu einer prüfungsähnlichen Atmosphäre, da der/die TutorIn bzw. der Lehrer gezwungen ist, intensiver als Fragesteller aufzutreten.
- Bestandteile des Gesprächs sind typischerweise:
  - Erläuterung der Lösungen
  - Demonstration am Rechner
  - Besprechung aufgetretener Schwierigkeiten
  - Vorgangsweise beim Entwurf
  - Benötigte Arbeitszeit
  - Aufteilung der Arbeit unter den Gruppenmitgliedern
  - Diskussionsphase

Punkte

Wenn Tutorium 0 nicht besucht wurde, gibt es 0 Punkte und keine Note. Wenn Tutorium 0 besucht wurde und ansonsten keine Aufgabe abgegeben wurde, gibt es ebenfalls keine Note. Sobald eine der Aufgaben abgegeben wurde, gibt es eine Note.

Aufgabe 0	keine Punkte
Aufgabe 1	max. 15 Punkte
Aufgabe 2	max. 30 Punkte
Aufgabe 3	max. 15 Punkte
Aufgabe 4	max. 40 Punkte

Notenschlüssel

Zur Erreichung einer positiven Note müssen alle 4 Aufgaben abgegeben werden und bei jeder Aufgabe muss ein „ernsthafter Versuch“ erkennbar sein, die Aufgabe lösen zu wollen. Als „ernsthafter Versuch“ bei den Aufgaben 1 bis 3 zählt $^1/_3$ der maximal zu erreichenden Punkte. Bei Aufgabe 4 müssen für einen „ernsthaften Versuch“ zumindest 20 Punkte erreicht werden.

- 0 – 50 Punkte Nicht genügend
- 51 – 62 Punkte Genügend
- 63 – 75 Punkte Befriedigend
- 76 – 87 Punkte Gut
- 88 – 100 Punkte Sehr gut

Schwindeln, Täuschungsversuch, Plagiat

Wenn ein/e Student/in seine/ihre Arbeit erschwindelt, wird dies mit der Note "U" ("Ungültig/Täuschung") bewertet. Wenn ein Team seine Arbeit erschwindelt, werden alle Teammitglieder mit der Note "U" ("Ungültig/Täuschung") bewertet.

Huch, es geht sich mit der Zeit nicht aus

Ein wesentlicher Aspekt auf dem Weg zum Profi ist der richtige Umgang mit dem Faktor „Zeit“. Jedes Projekt hat einen Liefertermin. Das ist der spätestens mögliche Liefertermin. Wenn du einmal verstanden hat, dass du ja auch früher liefern darfst, dann hast du bereits viel gewonnen. Du darfst also in deinem eigenen Kalender durchaus einen „Privatliefertermin“ eintragen, welcher zum Beispiel eine Woche früher als der vom Lehrer geforderte Termin ist. Die zweite wichtige Komponente ist der „Plan“. Dazu musst du die Arbeit eines Projektes anfänglich abschätzen, einen bestimmten Risikofaktor einberechnen, und dann die geplanten Arbeitszeiten in deinem Kalender festlegen.

Nimm als Beispiel die Arbeit an Aufgabe 1. Du liest die Aufgabenstellung durch und findest heraus, dass du „keine Ahnung“ hast, wie lange die Arbeit daran werden wird. „Keine Ahnung“ bedeutet zweierlei: (1) Du setzt dich rasch – man nennt dies „as soon as possible“ (ASAP) – hin, um eine „Ahnung“ zu kriegen. (2) Je weniger „Ahnung“, desto höher musst du den „Risikofaktor“ setzen. Daraus folgt für mich die wichtigste Grundregel: Bei „wenig bis gar keine Ahnung“ muss ich mich „sofort“ (= ASAP) darum kümmern, wenn es mir wichtig ist, zeitgerecht liefern zu können. Du könntest also zum Beispiel 5 Stunden investieren um eine „Ahnung“ zu bekommen, wie viel Aufwand sich hinter Aufgabe 1 verbirgt.

Nehmen wir ein zweites Beispiel: Du hast schon mal auf Assemblerebene programmiert und hast also bereits „ein bisschen Ahnung“ von Aufgabe 1. Du schätzt, dass es wohl in etwa 3 Tage Arbeit sein wird, doch so richtig sicher bist du dir dabei nicht. Ich würde in einem solchen Fall das Risiko bewerten und statt der 3 Tage vermutlich sicherheitshalber 9 Tage planen. Auf jeden Fall würde ich die 9 Tage Arbeit in meinem Kalender einplanen. Im „Hinterkopf“ kannst du ja behalten, dass – wenn du gut drauf bist – ein Teil dieser 9 Tage nicht gebraucht wird.

Je öfter du das Spiel mit „den geplanten Zeitaufwand im Voraus schätzen“, „den tatsächlichen Zeitaufwand messen“ und schließlich „aus dem Vergleich zwischen Schätzung und Messung lernen“ machst, desto besser lernst du schätzen und planen. Die Fähigkeit zu planen wirst du dein ganzes Leben lang brauchen können.

Die obige Vorgangsweise führt mit großer Wahrscheinlichkeit zur erfolgreichen, rechtzeitigen Fertigstellung von Arbeiten und damit zur fristgerechten Lieferung.

Das Eintreffen unerwarteter Ereignisse kann jedoch auch einen guten Plan ins Wanken bringen. Wenn dies (hoffentlich selten) der Fall sein sollte, dann solltest du folgende Punkte beachten: Ein Agieren vor dem Liefertermin ist viel besser als ein Agieren nach dem verpassten Liefertermin. Ein Brief an den Lehrer vor dem Abgabetermin ist wesentlich effizienter also ein Brief danach. Eventuell ist ein persönliches Vorsprechen sogar noch am besten; damit vermeidest du den Eindruck von „so wichtig ist mir das aber nicht“. Professionell ist, wenn man vorbereitet ist, einen Ersatzliefertermin anzubieten; diesen muss man dann aber auch wirklich einhalten.

Nehmen wir also an, du bist mit dem Projekt zu Aufgabe 1 „on track“, doch zum Abgabetermin fehlt noch „ein bisschen“. Dann kannst du bei dieser Lehrveranstaltung dein Lieferverzögerungskonto (insgesamt maximal 3 Tage für die gesamte KU) anzapfen. Solltest du dies machen, dann solltest du gleichzeitig auch analysieren, wie es dazu kommen konnte. Daraus kannst du für die Zukunft lernen. Vielleicht bist du dem „20-80-Problem“ auf dem Leim gegangen. Es kommt oft vor, dass man für die ersten 80% eines Projektes lediglich 20% der zu Verfügung stehenden Zeit braucht; doch die verbleibenden 20% des Projekts verbrauchen 80% der Zeit. Wie gesagt: Lerne, dieses Problem in den Griff zu kriegen.

OK. Du hast geliefert. Doch kurz danach, jedoch noch vor dem Abgabegespräch, kommst du drauf, dass deine Lieferung fehlerhaft bzw. ungenügend ist. In diesem Fall würde ich „proaktiv“ handeln: Ausbessern bzw. nachbessern und – egal, wie spät – nochmals in verbesserter Form abgeben. Damit hast du klar zum Ausdruck gebracht, dass du „auf Draht“ bist. Mag sein, dass du für die verbesserte, nachträgliche Abgabe keine Extrapunkte kriegst; auf jeden Fall kannst du jedoch für das Abgabegespräch selbst mit deinem „augmentierten Wissen“ „punkten“. Ich kenne keine Situation, in welcher ein solcher „proaktiver“ Ansatz nicht geholfen hätte.

Jetzt hast du das Abgabegespräch mit der Tutorin gemacht und hast unglücklicherweise knapp das Punkteminimum verfehlt. Und dies trotz deiner Extraabgabe nach Abgabefrist. In diesem besonderen Fall solltest du bereit sein, durch geeignete, nachträgliche „Zusatzlieferung“ über das Limit zu kommen. Du könntest der Tutorin zum Beispiel vorschlagen, innerhalb von ein paar Tagen noch „jede Menge auf die Waage zu legen“. Deine Tutorin wird auch in diesem Fall kooperativ sein.

Kommen wir zum unerwünschten Fall: Du bist jung und willst die Grenzen ausloten. Die Abgabefrist verstreicht und du möchtest wissen, was passiert, wenn du einfach „durchtauchst“. In der Schule gab es ja auch immer wieder eine Lösung; egal, wie weit weg du vom Soll warst. Na ja, in diesem Fall haben wir es einfach: Solltest du nie liefern, dann kriegst du keine Note. Solltest du zumindest einmal geliefert haben, dann kriegst du eine negative Note.

Und wie lang soll die Dokumentation sein?

Die (aus Sicht der StudentIn) „blöde“ Antwort ist vermutlich auch die weiseste: So kurz wie möglich, aber nicht kürzer. Meistens drückt diese Frage ein Defizit im Verständnis des Sinns einer Dokumentation aus. Deshalb möchte ich hier etwas breiter ausholen.

Eine Dokumentation ist ein schriftlicher Bericht. Der Zweck eines Berichts ist zu informieren oder zu überzeugen. Als StudentIn möchtest du einerseits über deine Arbeit informieren, andererseits aber auch die LehrerIn von der Qualität deiner Arbeit überzeugen.

Ein Bericht ist ein Aufsatz, welcher eine bestimmte Form hat. Die Form drückt sich in der Verwendung einer typischen Struktur und einer geeigneten Sprache aus. Die sofort sichtbaren Elemente einer typischen Struktur sind Titel der Arbeit, Autorennennung, Kapitelüberschriften und Quellenangabe. Das erste Kapitel heißt immer „Einleitung“ und das letzte Kapitel immer „Zusammenfassung“. Illustrationen werden immer mit einer „Caption“ versehen. Es steht also „Abbildung 3.4“ oder so ähnlich dabei. Im Text geht man dann auf die Illustration ein, indem man etwa schreibt: „Wie in Abbildung 3.4 zu sehen ist, ...“.

Strukturelemente findet man jedoch auch innerhalb eines Kapitels. Schau dir den Text, welchen du soeben liest an. Der Text ist in Absätze gegliedert. Als Zeichen für einen neuen Absatz wurde ein etwas größerer Zeilenabstand gewählt. Jeder Absatz drückt eine „Idee“ aus. Ein Absatz besteht typsicherweise aus mehreren Sätzen. Meistens bezieht sich ein Satz auf seinen Vorgänger. Der Leser sollte in die Lage versetzt werden, beim Lesen ohne Schwierigkeiten einen sinnvollen Zusammenhang zwischen den einzelnen Sätzen zu erkennen. Damit dies de Leser leichter gemacht wird, verwenden wir verschiedene Tricks: Überleitungswörter wie „jedoch“, „außerdem“, „beispielsweise“ usw. sind eine Möglichkeit.

Auch innerhalb eines Satzes gibt es eine fixe Struktur. Die Wörter eines Satzes sind durch die dazwischen liegenden Leerzeichen und sonstigen Satzzeichen erkennbar. Die Reihenfolge der Wörter wird durch die Grammatik bestimmt. Jedes Wort folgt seinen eigenen Rechtschreibregeln. Und jeder Buchstabe eines Wortes ist Bestandteil eines Zeichensatzes („Font“).

Es gibt auch eine Reihe von Regeln, was man im ersten Kapitel, also in der „Einleitung“ eines Berichtes schreibt. Eine typische Reihenfolge könnte so aussehen: Worum geht es in der Arbeit? Warum ist diese Arbeit wichtig/interessant? Welche Voraussetzungen sind notwendig? In welcher Reihenfolge wird die Arbeit präsentiert?

In den weiteren Kapiteln werden dann, wie in der Einleitung festgelegt, die einzelnen Details beschrieben. Das letzte Kapitel fasst die Arbeit zusammen. Es hilft dem Leser nach dem Aufsaugen aller Details „zur Oberfläche“ zurückzukehren. In der Zusammenfassung kann man dem Leser also nochmals das Wichtigste der Arbeit „in Erinnerung“ rufen.

Kommen wir zur Frage der Sprache. Die Sprache sollte informativ und nüchtern sein. Die Zeitform ist (meist) das Präsens. Das wichtigste an der Sprache in der Wissenschaft ist Genauigkeit. Man sagt genau das, was man sagen möchte. Die Sprache sollte auch „klar“ sein. Damit ist gemeint, dass alles Nicht-Notwendige weggelassen wird. Ein weiteres Attribut der Sprache sollte „Aufrichtigkeit“ und „Offenheit“ sein. Die Sprache sollte also nicht „geheimnisvoll“, „zweideutig“ oder „augenzwinkernd“

sein. Schließlich sollte die Sprache „geläufig“ und „vertraut“ sein. Du sollt durchaus einfache Wörter verwenden.

Am Anfang dieses Kapitels stand die Frage nach der Länge der Dokumentation. Wir haben bis jetzt jedoch nur über Struktur und Sprache gesprochen. Mit diesen Mitteln sollst du nun einen Text machen, der „eine Geschichte erzählt“. Die „Geschichte“ besteht aus den Ergebnissen deiner Arbeit im Rahmen der Konstruktionsübung. Dem Leser deiner „Geschichte“ sollte auf möglichst einfache und rasche Art und Weise klar werden, was du gemacht hast und welche Ergebnisse bei deiner Arbeit rausgekommen sind. Ohne Schnörksel. Ohne Rufzeichen. Ohne Abschweifungen. Also „kurz und bündig“.

Möglicherweise bist du nach wie vor der Meinung, dass das Anfertigen einer Dokumentation im Rahmen der Konstruktionsübung „überflüssig“ ist. „Schikane“ sozusagen. Dieser Ansicht halte ich Folgendes entgegen: Ein Profi im technisch-wissenschaftlichen Umfeld ist in der Lage, sich rasch in ein neues Problem einzuarbeiten und geeignete Lösungen vorzuschlagen oder auch anzufertigen. Hier ist also „technisches Know-How“ gefragt. Das übst du im Rahmen jeder Konstruktionsübung im Detail. Jedes Projekt wird immer auch durch eine geeignete Dokumentation begleitet. Spätestens am Ende eines Projektes sollte alles Wesentliche zum Projekt in einem Bericht festgehalten werden. Dieser Bericht dient hauptsächlich dazu, Resultate für die mögliche Weiterarbeit am Projekt zu einem späteren Zeitpunkt festzuhalten. Eine neue Mitarbeiterin – das kannst auch du selbst sein – kommt ins Projekt und „liest sich ein“, um darauf hin „am Projekt weiterzuarbeiten“. Eine professionelle Dokumentation dient also dazu, in einer Gruppe effizient arbeiten zu können.

Eine spezielle Form eines Berichts ist „der Antrag“. In einem Antrag versucht der Schreiber den Leser zu überzeugen, Ressourcen (Zeit, Mitarbeiter, Geld, Geräte) zur Verfügung gestellt zu kriegen. Ziel eines Antrags ist dessen „Genehmigung“. Es geht also um die Kunst des Überzeugens. Auch ein Antrag hält sich an die meisten der oben skizzierten Tipps zu Struktur und Sprache.

Im Rahmen der Konstruktionsübung sollst du also auch die Gelegenheit haben, die Anfertigung einer Dokumentation zu üben. Die Fertigkeit, einen wissenschaftlichen Text zu schreiben, kann man nur dadurch erreichen, indem man es tut. Man muss also schreiben.

Als Abschluss meiner Überlegungen zur Dokumentation möchte ich die Frage „Englisch“ oder „Deutsch“ beleuchten. Die Fachsprache im IT-Bereich ist Englisch. Die meisten Texte sind in Englisch verfasst. Je früher du das Schreiben von Berichten in englischer Sprache übst, umso besser. Solltest du im Rahmen dieser Konstruktionsübung jedoch davor zurückscheuen und den Bericht auf Deutsch verfassen, so ist dies erlaubt. Als Lehrer möchte ich dir jedoch in diesem Falle mitteilen, dass dir mit großer Wahrscheinlichkeit das Erlernen des Verfassens von Berichten in englischer Sprache noch bevorsteht. Je älter du jedoch wirst, umso schwieriger wird es für dich, eine neue Sprache zu erlernen.