INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Projektarbeit

Entwicklung von Übungsaufgaben für die Vorlesungen TGI und Informatik III

Tim Furche

Aufgabensteller: Prof. Dr. Claudia Linnhoff-Popien
Betreuer: Igor Radisic
Abgabetermin: 07. März 2001

Inhaltsverzeichnis

1 Aufgabenstellung, Ziele

Ausgehend von der Neugestaltung der Grundstudiumsvorlesungen „Einführung in die Informatik III“ (jetzt „Informatik III“) und „Technische Grundlagen der Informatik“ am Institut für Informatik der Ludwig-Maximilians-Universität München ab dem Sommersemester 2000 sollten im Rahmen dieser Projektarbeit drei wesentliche Ziele erreicht werden:

1. Identifizierung und Evaluation **möglicher Alternativen** zur Gestaltung und Durchführung der Übungen zu den genannten Vorlesungen. Ergebnis der Evaluations-Phase soll die Auswahl eines – nach den aufzustellenden Kriterien – geeigneten Konzeptes für die Durchführung der Übungen sein.
2. Entwicklung neuer bzw. Anpassung vorhandener **Übungsaufgaben** an die im ersten Schritt ausgewählte Konzeption.
3. **Anwendung** des ausgewählten Konzeptes in einem Jahrgang, so daß vorhandene Schwächen erkannt und, soweit möglich, beseitigt werden können.

Um die Ziele der Projektarbeit zu verdeutlichen, ist ein kurzer Vergleich der Gliederung der beiden Vorlesungen vor und nach der Neugestaltung des Grundstudiums angebracht.
Wie Abbildung 1 zeigt, hatte sich aus historischen Gründen eine ungewöhnliche Aufteilung der Lehrinhalte auf die beiden Vorlesungen ergeben. In beiden Vorlesungen sind Inhalte vermittelt worden, die üblicherweise der Rechnerarchitektur zugerechnet werden. Dabei wurde in „Informatik III“ eine VAX-Architektur (die an der Technischen Universität München entwickelte „Maschine für die Informatikausbildung“, MI vgl. [BGSW 91]), in den „Technischen Grundlagen der Informatik“ eine RISC-Architektur (MIPS nach [PaHe 97]) verwendet. Dies hat sich insbesondere im Hinblick auf die unterschiedlichen Assemblersprachen als zusätzliche und unnötige Hürde erwiesen, zumal die beiden Vorlesungen bisher beide im dritten Studiensemester vorgesehen waren. Auch mußten verschiedene Lehrinhalte doppelt behandelt werden, da eine Synchronisation, aufgrund der unterschiedlichen Architekturen, nur schwer zu erreichen war.
Daher wurde im Zuge der Neuordnung des Grundstudiums für das Sommersemester 2000, die u.a. auch eine Verlegung der Vorlesung „Technische Grundlagen der Informatik“ vom dritten ins zweite Studiensemester mit sich brachte, der gewünschte Lehrinhalt revidiert und umgeschichtet. Das Prinzip der Restrukturierung (siehe Abbildung 2) besteht darin, in den „Technischen Grundlagen der Informatik“ den Schwerpunkt auf die Rechnerarchitektur zu verschieben und dies durch eine ausführliche Behandlung der zu der betrachteten Architektur gehörigen Assemblersprache in den Übungen zu flankieren. Dadurch wird die Vorlesung „Informatik III“ entlastet und es können Teile der bisherigen Hauptstudiumsvorlesung „Betriebssysteme“, die gewiß zum Grundstock eines Studiums der Informatik gerechnet werden können, hierher verlagert werden, so daß der Schwerpunkt nun auf Konzepten der Betriebssysteme und Systemprogrammierung liegt.
Die Restrukturierung führt natürlich zu einer Neuorientierung der Übungen: Im Rahmen der Vorlesung „Technische Grundlagen für Informatik“ werden Grundkenntnisse über den Aufbau moderner Rechnersysteme vermittelt. Dies soll in den Übungen unterstützt werden, indem Basiskenntnisse in einer zur in der Vorlesung verwendeten Rechnerarchitektur passenden *Assembler-Sprache* vermittelt werden und mit Hilfe eines Debuggers oder Simulators die Auswirkungen der einzelnen Befehle untersucht werden können.
Für die Vorlesung „Informatik III“ ist das Ziel, den Studenten einen praktischen Eindruck von den in der Vorlesung kennengelernten Konzepten, wie Scheduling, IPC, Deadlocks etc., zu vermitteln. Insbesondere soll das Problembewußtsein für die Fährnisse *nebenläufiger Programmierung* geweckt werden.
Die Anpassung der Übungen ist in zwei Schritten durchgeführt worden:

1. Auswahl geeigneter Plattformen für die praktische Umsetzung der in den Vorlesungen gelernten Konzepte während der Übungen.
2. Anpassung bzw. Erstellung von Übungsaufgaben für die ausgewählte Plattform und die geänderten Lehrinhalte der Vorlesungen.

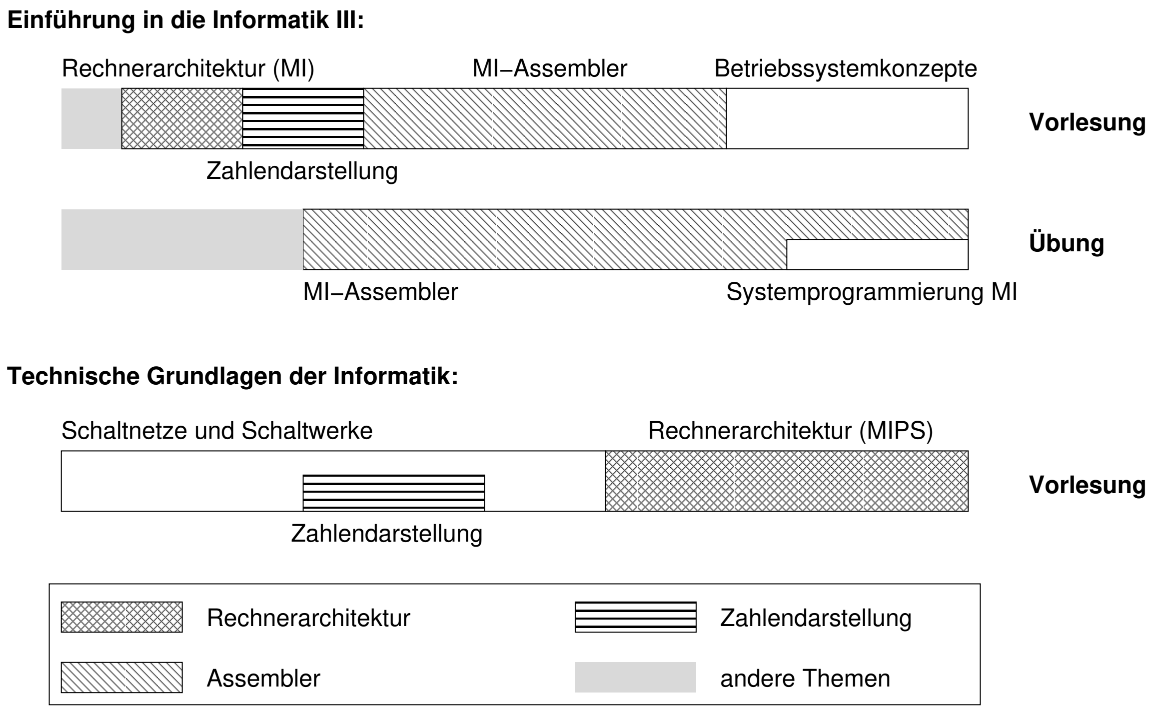


Abbildung 1: Vorlesungsgliederung *vor* der Neugestaltung . . .

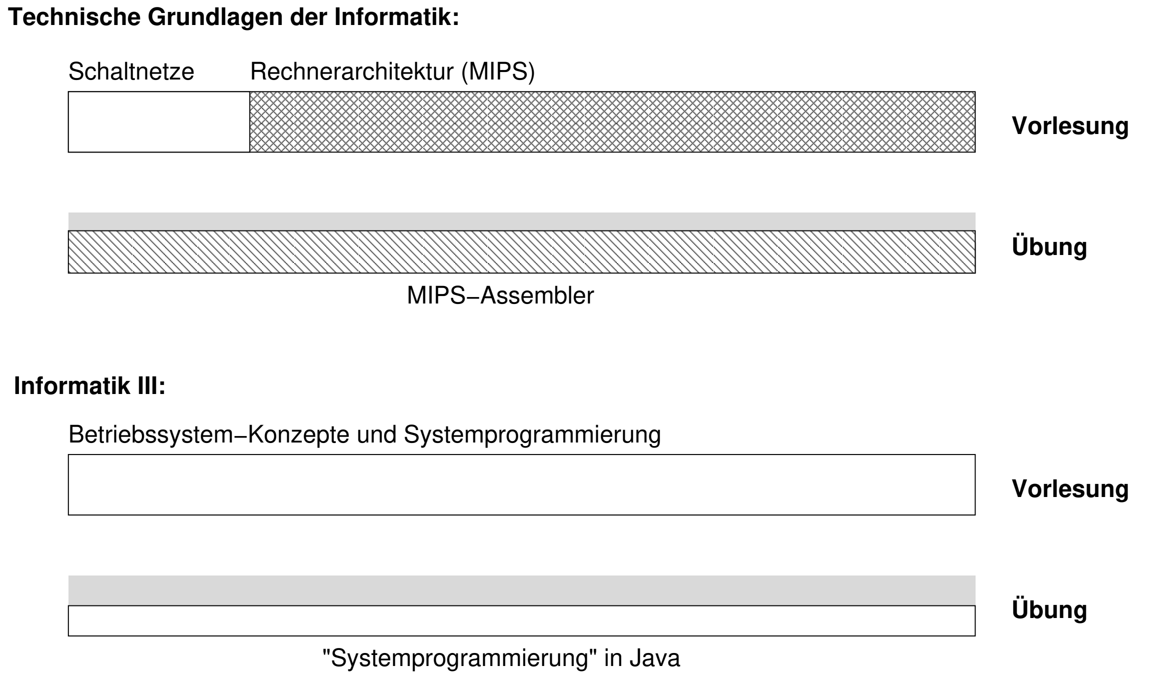


Abbildung 2: . . . und *nach* der Neugestaltung

1.1 Überblick über die Ausarbeitung

Im folgenden Abschnitt soll zuerst ein Blick auf die Auswahlkritieren geworfen wurden, die sich im Laufe der ersten Phase ergeben haben. Dann sollen einige Alternativen für die beiden Übungen vorgestellt und anhand der Kriterien bewertet werden. Abgeschlossen wird mit einem Ausblick auf die Entwicklung der Übungsaufgaben und die bisher gewonnenen Erfahrungen bei der Anwendung im Sommersemester 2000 bzw. Wintersemester 2000/01.

2 Auswahlkriterien

Offensichtlich gibt es eine gewisse Anzahl von **grundlegende Anforderungen**, die unabhängig von den Inhalten der Vorlesung, an eine Plattform zur Durchführung von Übungen im *Grundstudium* gestellt werden müssen:

Anwendung der Lehrinhalte: Es sollten möglichst viele der Lehrinhalte in der Plattform verwendbar sein.

Flache Lernkurve: Es sollte möglich sein, die Grundlagen und Voraussetzungen einer Plattform innerhalb möglichst kurzer Zeit, gewiß jedoch in einem Zeitraum von maximal drei Wochen, zu erlernen, so daß ab diesem Zeitpunkt nicht mehr der Umgang mit der Plattform, sondern die praktische Umsetzung von Lehrinhalten Lernziel sein kann.

Anschaulichkeit der Konzepte: Konzepte aus der Vorlesung sollten einerseits *einfach angewandt* werden können, um ein Verständnis für ihre Bedeutung zu erhalten, andererseits auch einfach hinsichtlich ihrer Realisierung untersucht werden können. So sollte es beispielsweise möglich sein, Monitore einfach zu verwenden, wie auch ihre Implementation (exemplarisch) nachzuvollziehen.

Begleitende Literatur: Offensichtlich sollte für eine geeignete Plattform eine umfassende Dokumentation und Begleitliteratur zur Verfügung stehen, möglichst sowohl einführende wie auch vertiefende.

Begleitende Übungsaufgaben: Günstig wäre die Existenz von exemplarischen Übungsaufgaben für die Plattform, einerseits um einen besseren Eindruck von der Verwendbarkeit zu erhalten, andererseits um ein gewissen Grundstock an Aufgaben zur Verfügung zu haben.

Vertretbarer Betreuungsaufwand: Angesichts der begrenzten Möglichkeiten zur Betreuung ist ein Weg zu finden, den Betreuungsaufwand in vertretbarem Maß zu halten. Insbesondere ist eine Einarbeitung der Tutoren nur begrenzt möglich.

2.1 Spezielle Kriterien für die simulierte Maschine

Neben den grundlegenden Anforderungen lassen sich einige spezifische Forderungen an den Simulator für den Einsatz in „Technische Grundlagen der Informatik“ ermitteln:

Architektur: Es sollte eine Architektur verwendet werden, die es ermöglicht, die wesentlichen in der Vorlesung gelernten Inhalte nachzuvollziehen.

Die Architektur sollte möglichst nahe an existierenden, modernen Rechnerarchitekturen orientiert sein, ohne jedoch durch einen komplexen Aufbau den Blick für die wesentlichen Strukturprinzipien zu trüben.

Einblicke in Wirkungsweise: Der Simulator sollte es ermöglichen, die Abläufe in der simulierten Maschine im Detail zu beobachten. Insbesondere die Veränderung von Speicher und Registern und die Darstellung der Daten im Speicher muß sichtbar gemacht werden können.

Oberfläche: Die Oberfläche sollte so gestaltet sein, daß die intendierten Abläufe einfach einsehbar sind. Beispielsweise sollte es leicht möglich sein, den Zusammenhang zwischen dem Assembler-Code und den ausgeführten Instruktionen zu erkennen.

2.2 Spezielle Kriterien für die Übungsplattform zur Systemprogrammierung

Auch für „Informatik III“ gibt es eine Reihe weiterer konkreter Kriterien:

Nebenläufige Programmierung: Zentrales Ziel der Programmierübungen zu „Informatik III“ ist es, Erfahrungen in nebenläufiger Programmierung zu gewinnen. Daher sollte die verwendete Lösung in diesen Bereich möglichst umfassende Unterstützung bieten.

Praktische Erfahrung: Die Studenten sollen die Möglichkeit erhalten, die praktische Umsetzung der gelernten Konzepte nachzuvollziehen. Wünschenswert wäre es, wenn dies an einem funktionierenden System stattfindet.

„Hands-on experience“: Die Studenten sollten einen möglichst praxisnahen Eindruck von den komplexen Abläufen eines Betriebssystems und den Fährnissen des Eingriffes in solche Funktionen.

Geleitet durch diese Kriterien soll nun ein kurzer Überblick über die genauer untersuchten Lösungsansätze gegeben und die tatsächlich getroffene Auswahl begründet werden.

3 Technische Grundlagen der Informatik

Wie bereits gesagt, soll für die Vorlesung „Technische Grundlagen der Informatik“ ein Simulator verwendet werden, anhand dessen die wesentlichen Konzepte der Rechnerarchitektur verdeutlicht werden können. Bei der Auswahl wurden neben anderen vor allem die folgenden vier Simulatoren betrachtet:

MI: Die **„Maschine für die Informatikausbildung“**, beschrieben in [BGSW 91][1], kurz MI, wird seit 1985 an der Technischen Universität München und seit 1987 an anderen deutschen Universitäten eingesetzt.

Sie basiert auf der VAX-Architektur und weist einen recht komplexen Befehlssatz mit verschiedenen Befehls- und Adressierungsmodi auf:

```
fib:    SEG
        MOVE  W  I H'10000', SP  -- Vorbesetzung Stackpointer
        JUMP     start

n:      DD    W  10              -- Eingabeparameter
ret:    RES      4               -- Resultatparameter
        EQUAL    i     = R0      -- Registervariablen
        EQUAL    f_old = R1      --  => Geschwindigkeit
        EQUAL    f     = R2      -- EQUAL-Makros
        EQUAL    f_new = R3      --  => Lesbarkeit

start:  MOVE  W  I 1, f_old      -- f_old = 1
        MOVE  W  I 0, f          -- f = 0
init:   MOVE  W  I 1, i          -- i = 1
        JUMP     test
for:    ADD   W  f, f_old, f_new -- f_new = f_old + f
        MOVE  W  f, f_old        -- f_old = f
        MOVE  W  f_new, f        -- f = f_new
step:   ADD   W  I 1, i          -- i++
test:   CMP   W  i, n            -- i > n ?
        JLE      for             -- nein: goto for
end:    MOVE  W  f, ret          -- return f
        HALT
        END
```

Der Simulator bietet eine klar gegliederte Oberfläche, in der die wesentlichen Funktionen schnell zugänglich sind (vgl. Abbildung 3).

Die wesentlichen *Nachteile* der MI sind

- Schlechte Dokumentation und fehlende Begleitliteratur. Die zur Verfügung stehende Dokumentation zur MI ([BGSW 91] und das MI-Manual) ist unübersichtlich und an vielen Stellen nicht klar genug. Dieser Mangel wird dadurch noch verschärft, daß es keine für Studenten geeignete Begleitliteratur zur MI gibt, so daß den Studenten der Zugang und insbesondere das tiefere Verständnis ausschließlich in den Tutorstunden vermittelt werden kann.
- Veraltete Architektur, komplexer Befehlssatz. Die MI basiert, wie bereits festgestellt, auf einer CISC-Architektur. Da die in der Vorlesung behandelten Prinzipien wie Pipelining besonders einfach an einer RISC-Architektur erläutert werden können und ein wesentlicher Teil der Begleitliteratur solche Architekturen als Beispiel verwendet, ist die Verwendung eines CISC-Befehlssatzes für die Übungen nicht angeraten.

SPIM: Mit der SPIM, vorgestellt in [PaHe 97], dokumentiert im Anhang A von [PaHe 97] (im Netz verfügbar) und in [Laru 90][2], haben Patterson und Hennessy den bei weitem verbreitetsten Simulator für den Einsatz im Lehrbetrieb entwickelt. Er orientiert sich eng an der MIPS-Architektur und implementiert nahezu den gesamten Befehlssatz, so daß die gesamte Literatur zur Programmierung einer MIPS-Architektur verwendet werden kann. An einer

[1] Eine ausführlichere Dokumentation findet sich im MI-Manual, z.B. unter `http://www3.informatik.tu-muenchen.de/public/lehre/info3/MI/mi-manual-html/mi-manual.toc.html`.

[2] Sourcen finden sich unter `http://www.cs.wisc.edu/~larus/spim.html`.

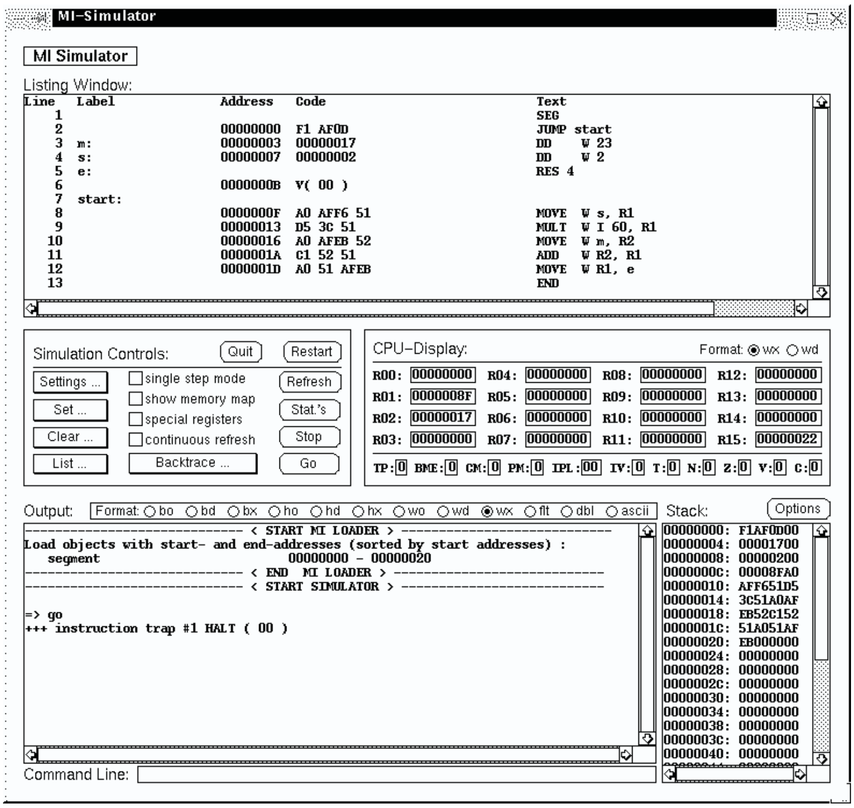


Abbildung 3: MI-Simulator

Vielzahl amerikanischer, europäischer, asiatischer und australischer Universitäten wird er mit Erfolg eingesetzt, wodurch ausreichend Material an Übungsaufgaben und Einleitungen zur Verfügung stehen.

Die wesentlichen Argumente, die für die Verwendung des SPIM-Simulators sprechen, sind:

- Hervorragende Begleitliteratur. [PaHe 97] behandelt alle für die Vorlesung wichtigen Konzepte in anschaulicher Weise und liefert Einblicke in die Motivation für bestimmte Designentscheidungen. Daneben gibt es eine vielzahl einleitender Tutorien zum SPIM, besonders [Nitz 97] der FU Berlin, welches in den Übungen zu „Technische Grundlagen der Informatik“ nun eingesetzt wird, und [Kalr 97] der Universität von Massachusetts.
- Einfache Architektur und eingängiger Befehlssatz. Die MIPS-Architektur gilt als die reinste und kompakteste Umsetzung einer RISC-Architektur. Ihr Befehlssatz ist dank seiner einheitlichen Strukturierung für die Lehre gut geeignet, wie folgender Ausschnitt zeigt:

```
prmpt1: .asciiz "\n\nThis␣program␣computes␣the␣Fibonacci␣function."
prmpt2: .asciiz "\nEnter␣value␣for␣n:␣"
descr:  .asciiz "fib(n)␣=␣"
        .text
        .align 2
main:   # Print the prompts
        li      $v0, 4          # print_str system service ...
        la      $a0, prmpt1     # ... passing address of first prompt
        syscall
        li      $v0, 4          # print_str system service ...
        la      $a0, prmpt2     # ... passing address of 2nd prompt
        syscall

# Read n and call fib with result
        li      $v0, 5          # read_int system service
        syscall

        move    $a0, $v0        # $a0 = n = result of read
        jal     fib             # call fib(n)
        move    $s1, $v0        # $s1 = fib(n)
# Print result
        li      $v0, 4          # print_str system service ...
        la      $a0, descr      # ... passing address of output descriptor
        syscall
        li      $v0, 1          # print_int system service ...
        move    $a0, $s1        # ... passing argument fib(n)
        syscall
# Call system   exit
        li      $v0, 10
        syscall
```

  Hinzu kommt, daß nahezu die gesamte Literatur zur MIPS-Architektur auch für den SPIM-Simulator verwendet werden kann.
- Enfache, intuitive Oberfläche. Die Oberfläche des SPIM-Simulators ist einfach zu bedienen und klar gegliedert, wie in Abbildung 4 gezeigt. Dadurch ist sie auch für Anfänger schnell zugänglich. Es gibt Versionen für unterschiedliche Betriebssysteme, insbesondere für Windows und Linux.

Natürlich stellt auch der SPIM-Simulator keine völlig optimale Lösung dar, es haben sich im Laufe der Anwendung einige Probleme ergeben:

- Es fehlen die Möglichkeit, fortgeschrittenere Konzepte (wie beispielsweise Inhalt der Pipeline) direkt zu visualisieren. Dieser Nachteil ließe sich möglicherweise durch den zusätzlichen Einsatz des DLXSim aufheben.
- Die bisher verwendete Version stellte keine akkurate Simulation einer MIPS dar, insbesondere wurden *delayed branches* und *delayed loads* nicht umgesetzt. Seit Beginn dieses Jahres sind diese Probleme durch die Version 6.3 behoben.

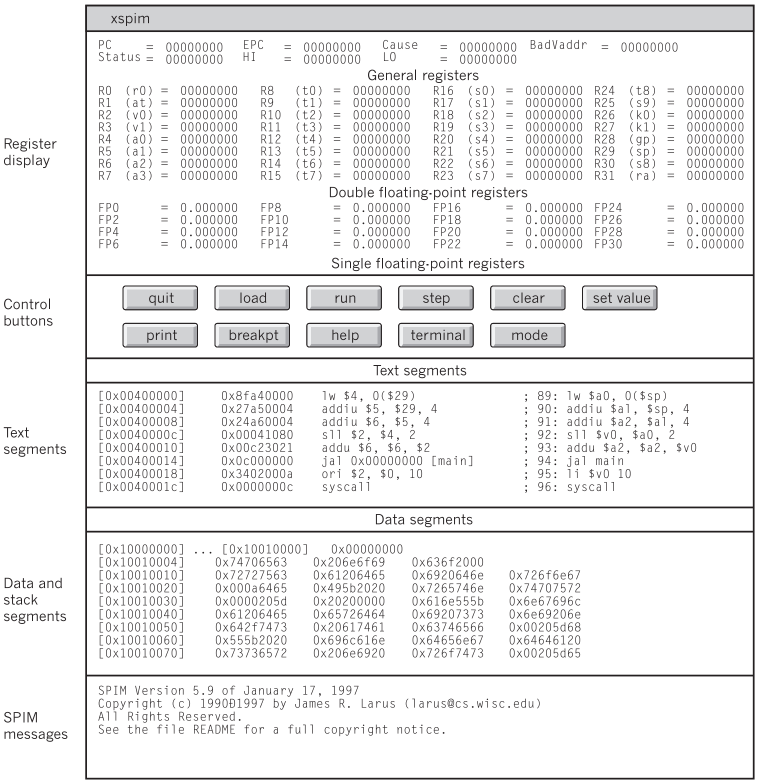


Abbildung 4: Gliederung des SPIM-Simulators

- Einige Details der Oberfläche entsprechen nicht ganz den Anforderungen, so kann man den Inhalt von Speichern und Registern nur in hexadezimaler Darstellung ausgeben, und es ist für den Anfänger unnötig umständlich, den Beginn der eigenen Programme zu identifizieren (siehe Abbildung 5).

  Auch hat es bis zur neusten Version noch deutliche Unterschiede zwischen der Version für Windows und derjenigen für Linux gegeben. Diese sind durch Version 6.3 ebenfalls behoben, wie Abbildung 6 zeigt.

DLXSim: In [HGP 95] stellen Hennessy, Goldberg und Patterson den **DLXSim** („DeLuXe"-Simulator)[3] vor. Wie [HGP 95] dient er in erster Linie zur Demonstration von Design-Prinzipien für Rechnerarchitekturen. Er bietet unter den betrachteten Simulatoren einmalige Möglichkeiten zur Analyse der Verteilung von Befehlen in einer Pipeline, so können einzelne Befehlsfolgen und Pipelinekonfigurationen auf strukturelle Hazards untersucht werden: Der DLXSim basiert, wie der unten vorgestellt SPIM, auf der MIPS-Architektur und verwendet deren Befehlssatz.

Leider gibt es jedoch einige Einschränkungen, die seine Verwendung als Plattform für „Technische Grundlagen der Informatik" unmöglich erscheinen lassen:

- Relativ steile Lernkurve. Aufgrund der umfassenderen Möglichkeiten ist eine längere Einarbeitungszeit erforderlich als bei den anderen Alternativen.
- Uneinheitliche Oberfläche. Die Oberfläche ist auf verschiedenen Systemen nicht einheitlich.

[3]Dokumentation und Sourcen finden sich unter `http://www.mkp.com/books_catalog/ca/hp2e_res.asp\#dlx`.

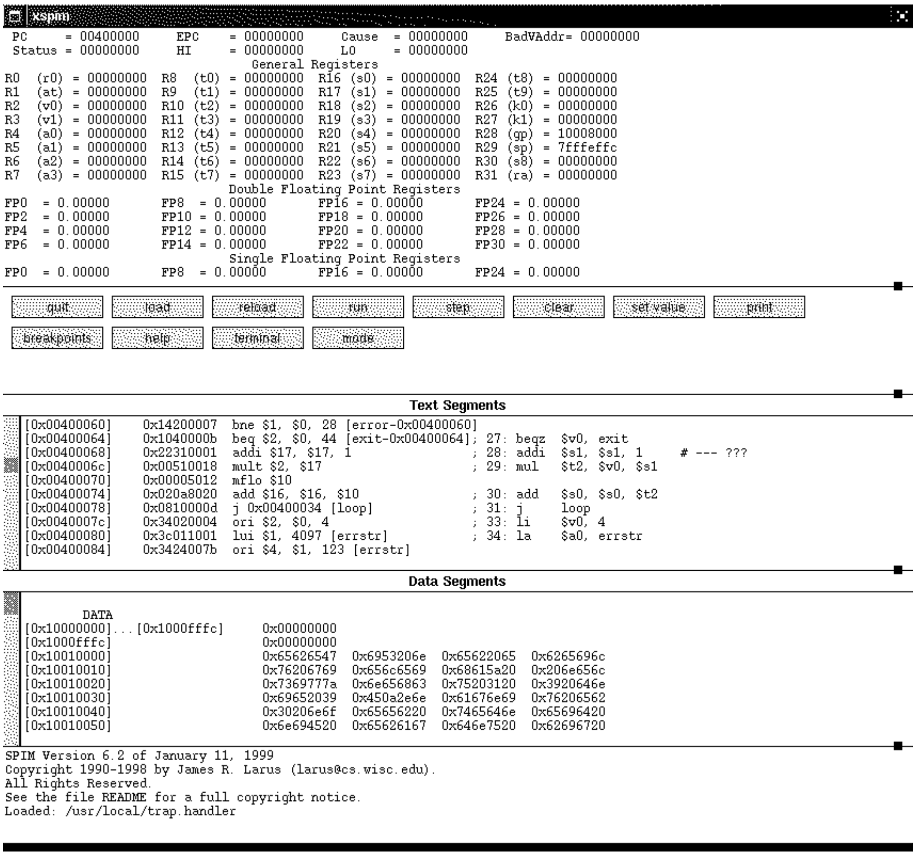


Abbildung 5: MIPS-SPIM Simulator unter UNIX

Abbildung 6: MIPS-SPIM Simulator unter Windows

- Mangelnde Anschaulichkeit bei grundlegenden Funktionen. Leider werden die Grundfunktionen zu Gunsten tiefergehender Analysefunktionen vernachlässigt, teilweise sind entscheidende Informationen vollkommen verborgen.

Es ist jedoch durchaus vorstellbar und wünschenswert den DLXSim in kommenden Jahren gegen Ende der Vorlesung oder in weiterführenden Kursen einzusetzen, da er, bei nahezu gleichem Befehlssatz wie der SPIM, wesentlich tiefere Einblicke in die Auswirkungen von Designentscheidungen einer Architektur auf die Ausführung einzelner Anweisungen und Programme zuläßt.

MIC/IJVM: Tanenbaum stellt in [Tane 98] den Simulator **MIC-1**[4] für die *Integer Java Virtual Machine* (**IJVM**, im wesentlichen der Befehlssatz der *Java Virtual Machine* ohne Gleitkommabefehle) vor. Die Idee ist hierbei, die grundlagen der Rechnerarchitektur an der von Sun spezifizierten virtuellen Maschine für Java zu vermitteln unter Nutzung der zugehörigen Assemblersprache:

```
//
//   echo.jas
//
//   Author
//     Dan Stone
//
//   Description
//     Sample assembly program that reads a key press and echoes the result to
//     standard output.
//


.main
```

[4]Dokumentation in [OnSt 00].

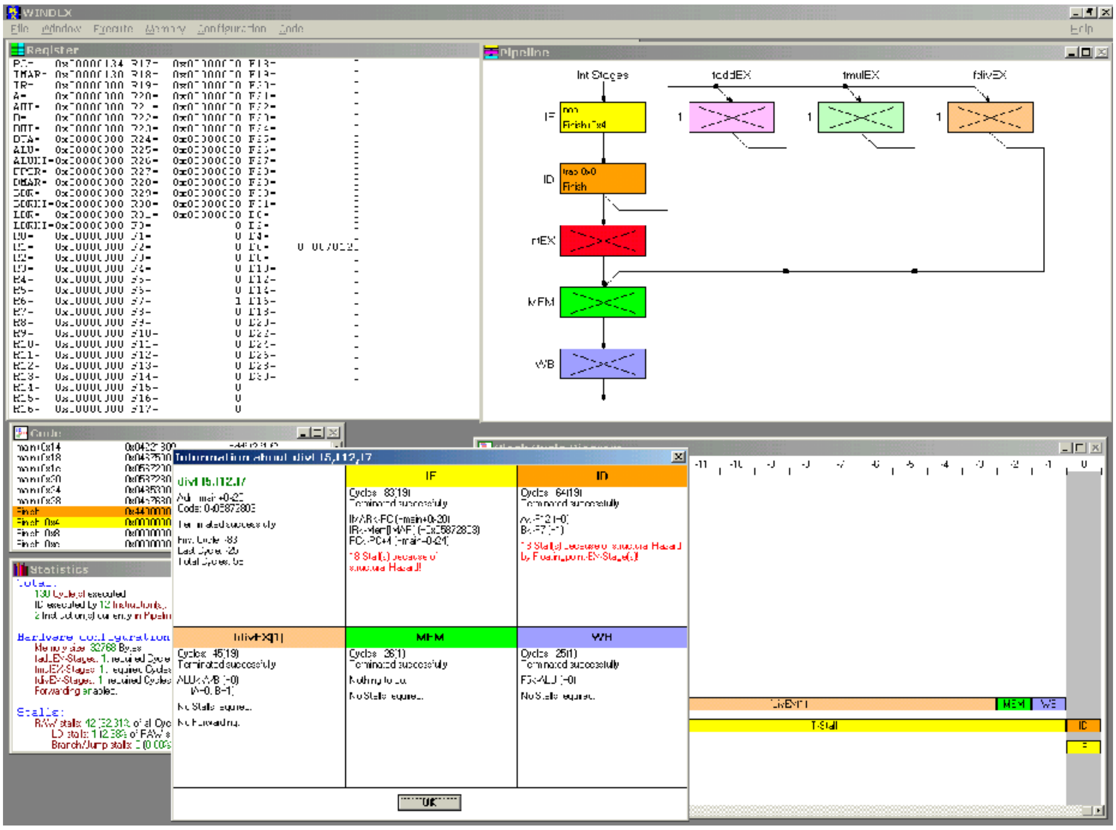


Abbildung 7: Windows-Oberfläche des DLX-Simulators mit Informationen über strukturelle Hazards

```
L1:    IN           // request character input from memory
       DUP          // duplicate top of stack (inputed char) for comparing
       BIPUSH 0x0   // push 0x0 for comparison
       IF_ICMPEQ L2 // if no characters are available for input, loop
       OUT          // else, print character
       GOTO L1      // loop back to beginning of program


L2:    POP          // No key has been pushed, so clear the stack,
       GOTO L1      // and start over
.end-main
```

Während durch die Nutzung der JVM die Studenten ihr Verständnis vertiefen können, indem Java-Programme in die Assemblersprache übersetzt werden, bringt dies etliche Nachteile mit sich:

- Ziel des Entwurfs der JVM war es, eine möglichst einfache zu realisierende Architektur (beispielsweise für *embedded systems*) zur Verfügung zu stellen. Man entschied sich für die Nutzung einer Stackmaschine.

  Dadurch ist die Programmierung der Stackmaschine recht aufwendig, der Lernaufwand eher hoch. Andererseits erhält der Student Einblick in die Funktionsweise einer sehr einfach gehaltenen Stackmaschine.
- Leider gibt es außer [Tane 98] nur wenig für Studenten geeigente Begleitliteratur, insbesondere da diese Lösung bisher nur in geringem Umfang eingesetzt wird.
- Durch die Beschränkung auf Integer-Befehle wird zwar der Befehlssatz übersichtlicher, es lassen sich jedoch einige in der Vorlesung umfangreich behandelte Konzepte (insbesondere Zahlendarstellung nach IEEE 754) nicht angemessen darstellen.
- Es existiert noch keine brauchbare Oberfläche für den Simulator, wodurch einerseits die Lernkurve und andererseits der Betreuungsaufwand massiv ansteigt.

Nach diesem kurzen Überblick über die Alternativen für „Technische Grundlagen der Informatik“ sollen nun die für „Informatik III“ betrachteten Systeme vorgestellt werden.

4 Informatik III

Für die Vorlesung „Informatik III“ ist das Ziel, den Studenten einen praktischen Eindruck von den in der Vorlesung kennengelernten Konzepten, wie Scheduling, IPC, Deadlocks etc., zu vermitteln. Insbesondere soll das Problembewußtsein für die Fährnisse *nebenläufiger Programmierung* geweckt werden.
Mehr noch als bei „Technische Grundlagen der Informatik“ steht hier die Bewertung, was in einer Grundstudiumsübung angesichts der Möglichkeiten der Betreuung und der wachsenden Studentenzahlen realisierbar ist, im Zentrum. Es wurden im wesentlich drei unterschiedliche Ansätze betrachtet:

Lehrbetriebssystem: Die umfassendsten Möglichkeiten bietet der Einsatz eines **Lehrbetriebssystems**. Anhand eines Lehrbetriebssystems lassen sich sämtliche in der Vorlesung behandelten Konzepte praktisch untersuchen bzw. implementieren. So wird an einer großen Anzahl nordamerikanischer Universitäten das an der University of Berkeley entwickelte *Nachos* eingesetzt, wobei die Studenten mehrere Aufgabenblöcke bearbeiten müssen (beispielsweise die Implementation eines RR-Schedulers).

Spezialisierte, nebenläufige Programmiersprache: Zur Verdeutlichung der Probleme und Möglichkeiten der nebenläufigen Programmierung bietet sich der Einsatz eines speziell darauf zugeschnittenen Programmiersprache an (beispielsweise BACI – *Ben-Ari Concurrent Interpreter*).

Bekannte Programmiersprache mit Unterstützung für nebenläufige Konzepte: Schließlich lassen sich die Grundlagen der nebenläufigen Programmierung auch an einer den Studenten bereits vertrauten Programmiersprache (*Java*) behandeln.

Wie in Abbildung 8 gezeigt, ergibt sich für jede der Alternativen offensichtlich eine deutlich andere Strukturierung der Übungen. Insbesondere die erste Alternative erfordert einen beträchtlich höheren Betreuungs- und Einarbeitungsaufwand, der, wie sich gezeigt hat, an deutschen Universitäten in einer Grundstudiumsvorlesung anscheinend nicht tolerierbar ist.

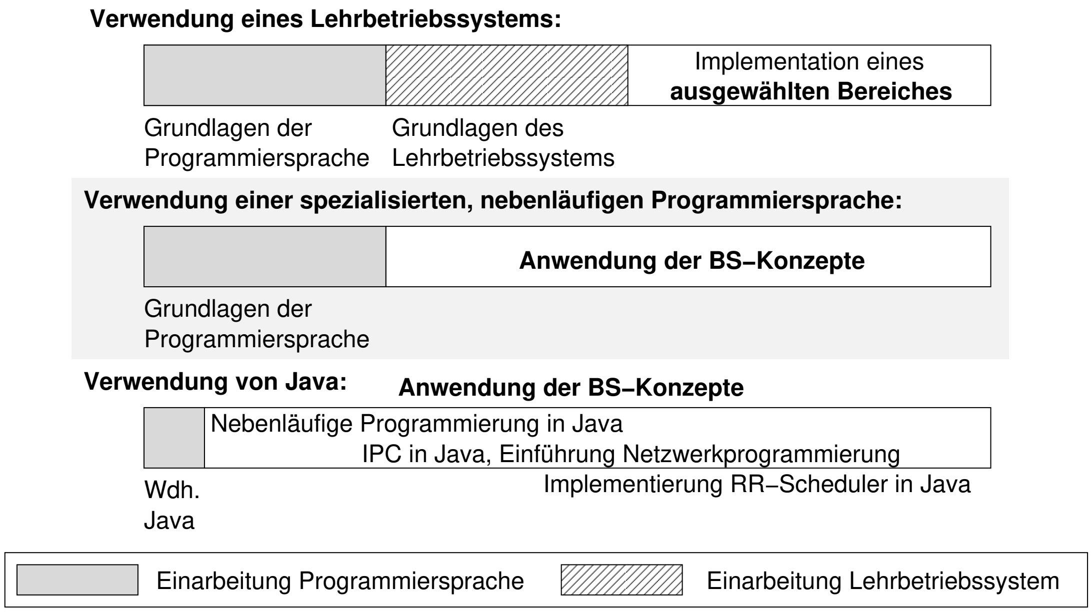


Abbildung 8: Varianten der Übungsgestaltung zu „Informatik III“

Obwohl die Entscheidung letztendlich zugunsten der dritten Alternative gefallen ist, soll nun eine Diskussion der verschiedenen betrachteten Lehrbetriebssysteme folgen, die beispielsweise bei einer Anpassung anderer Veranstaltung nützlich sein könnte.

4.1 Lehrbetriebssysteme

Ursprünglich bestand die Hoffnung, daß es möglich sein könnte, im Rahmen der Umstellung in den Übungen ein Lehrbetriebssystem einzusetzen. Unter einem **Lehrbetriebssystem** (*instructional* oder *educational operating system*) versteht man üblicherweise ein Betriebssystem bzw. einen Betriebssystem-Simulator, dessen Fokus auf dem Einsatz in der Lehre liegen.
In [FCZP 00] werden die wesentlichen Kriterien für ein solches *instructional operating system* genannt:

> **Simplicity:** *Powerful kernels tend to be too large and over-featured. Many algorithms are tuned for speed, lacking a simple formulation. Also, many unneces-sary modules are found in kernels for performance reasons.*
>
> **Readability:** *OS code is often obscured due to its historical origin and lots of options and hardware variants being supported.*
>
> **Hardware independence:** *While most systems claim to be portable, we found that the effort of porting a complex existing system is comparable to building a new simple one.*
>
> **Transparency:** *For didactic reasons we favor clear program code instead of efficient realizations. The basic principles implemented in an OS have to be easily comprehensible by the students. Most OSs are designed for efficiency.*
>
> **Lecture compatibility:** *Since this is a course for students in the second academic year, we have to take into account what they have learned so far.“ ([FCZP 00], S. 5)*

Weitere Anforderungen, die sich im Laufe der Diskussion ergeben haben, sind:

Unterstützung einer großen Anzahl von Betriebssystemkonzepten: Es sollte eine möglichst breite Auswahl an Betriebssystemkonzepten unterstützt werden, ohne die Kernstruktur unübersichtlich zu machen. Besonders wichtig ist dies in Hinblick auf eine Nutzung in weiterführenden Kursen (Systempraktikum, Hauptstudium).

Nähe zu realen Systemen: Es wäre wünschenswert, wenn sich das verwendete System beispielsweise im Hinblick auf Struktur und Systembibliothek an realen Systemen orientiert.

Veranschaulichung von Abläufen: Es wäre wünschenswert, wenn sich zentrale Abläufe (beispielsweise Scheduling) visualisieren lassen, so daß der Student diese einfacher erfassen kann. Dies ist besonders bei komplexen Abläufen sinnvoll.

Anhand dieser Kriterien wurden die folgenden Systeme untersucht:

Linux Kernel: Gerade der Linux-Kernel kann die fünf Grundkriterien nicht erfüllen. Weiterhin erweißt sich die schlechte Modularität der Grundfunktionen des Kernels als Hindernis, da dadurch lange Testzeiten entstehen.

Wie bei allen echten Betriebssystemen, so auch bei MINIX und XINU, (im Gegensatz zu Betriebssystem-Simulatoren wie Nachos, Topsy, RCOS etc.) fällt das Debugging recht schwer, insbesondere sollte ein BIOS-Simulator (wie SimOS (`http://simos.stanford.edu`) oder Bochs x86 (`http://www.bochs.com`) verwendet werden.

MINIX: In [TaWo 97] stellt Tanenbaum die Strukturierung seines UNIX-Abkömmlings MINIX vor:

> *„MINIX is a free UNIX clone that is available with all the source code. Due to its small size, microkernel-based design, and ample documentation, it is well suited to people who want to run a UNIX-like system on their personal computer and learn about how such systems work inside. It is quite feasible for a person unfamiliar with operating system internals to understand nearly the entire system with a few months of use and study.“ [Tane 96]*

Wie bereits die zitierte Vorstellung zeigt, ist MINIX ebenfalls nur mit einem erheblichem Lernaufwand zu überblicken. In [AEG$^+$ 91] wird eine Bewertung seiner Eignung für den Einsatz als Lehrbetriebssystem gegeben. Die dabei gewonnen (eher kritischen) Resultate haben die Entwicklung von Nachos maßgeblich beeinflußt.

XINU: Als Alternative zu MINIX mit einer eher an Windows (NT) angelehnten Struktur und Systembibliothek wird in [Come 83] XINU vorgeschlagen:

> *Xinu is a small, elegant, multitasking Operating System supporting the following features:*
>
> - *Concurrent Processing*
> - *Message Passing*
> - *Ports*
> - *Semaphores*
> - *Memory Management*
> - *Buffer Pools*
> - *Uniform Device I/O*
> - *Shell*
> - *Tcl*
> - *TCP/IP*
>
> *Xinu was originally designed as a vehicle for teaching Operating System design concepts and is used by many educational institutions for this purpose. Later versions supported TCP/IP, these versions are often used in Data Communications courses.*
>
> *Xinu is also well-suited as an environment for teaching advanced programming concepts such as Concurrent Processing, I/O, IPC and Client-Server Interaction. All processes in most Xinu versions (except Version 8) run in the same address space and could therefore be likened to threads in WIN32. In fact, any function in Xinu can be run in its own process, and the mechanism for creating such processes or threads is very similar to the WIN32 mechanism. Of course, major advantages of Xinu are its (relative) simplicity, and that the source code can be modified as needed.“ [Chla 98]*

XINU weißt im Vergleich zu Nachos ähnliche Nachteile wie MINIX auf, insbesondere wegen seiner höheren Komplexität und der Tatsache, daß es sich nicht um ein simuliertes Betriebssystem handelt.

Nachos: Aus der Erfahrung des Einsatzes von MINIX im Lehrbetrieb wurde an der Universität von Californien 1992 (dokumentiert in [CPA 92]) durch Christopher, Procter und Anderson **Nachos** entwickelt:

> *„Nachos is instructional software for teaching undergraduate, and potentially graduate, level operating systems courses. The Nachos distribution comes with:*
>
> - *an overview paper,*
> - *simple baseline code for a working operating system,*
> - *a simulator for a generic personal computer/workstation,*
> - *sample assignments,*
> - *a C++ primer (Nachos is written in an easy-to-learn subset of C++, and the primer helps teach C programmers our subset).*
>
> *The assignments illustrate and explore all areas of modern operating systems, including threads and concurrency, multiprogramming, system calls, virtual memory, software-loaded TLB's, file systems, network protocols, remote procedure call, and distributed systems.“ [CPA 92]*
>
> *„Nachos is instructional software that allows students to study and modify a real operating system. The only difference between Nachos and a 'real' operating system is that Nachos runs as a single Unix process, whereas real operating systems run on bare machines. However, Nachos simulates the general low-level facilities of typical machines, including interrupts, virtual memory and interrupt-driven device I/O.“ [Nart 95]*

Der Erfolg von Nachos an nordamerikanischen und internationalen Universitäten war, insbesondere auch dank der guten Dokumentation durch Nutzer ([Nart 95] und [Kalr 97]), derart groß, daß Silberschatz und Galvin ein Kapitel über Nachos in ihr Betriebssystembuch ([SiGa 98] – Online-Kapitel unter `http://www.bell-labs.com/topic/books/os-book/nachos-dir/index.html`) aufgenommen haben.

Auch heute noch wird Nachos in der überwiegenden Anzahl nordamerikanischer Universitäten eingesetzt, insbesondere zur Begleitung von Kursen des zweiten bis vierten Studiensemesters.

Für unser Einsatzgebiet liegen die wesentlichen Vorteile von Nachos im großen Umfang des Begleitmaterials und der Unterstützung ungewöhnlich vieler Teilaspekte eines Betriebssystems (insbesondere Dateisystem und Netzwerkfunktionalität), der wesentliche Nachteil ist die relativ schlechte Lesbarkeit des Quellcodes (vor allem dank der Verwendung einer C++-Teilmenge) und eine immer noch recht komplexe Struktur.

Topsy: Nach den oben vorgestellten Kriterien wurde an der ETH Zürich durch Fankhauser, Conrad, Zitzler und Plattner Topsy entwickelt:

> *„Topsy is a small operating system which has been designed for teaching purposes (Topsy stands for Teachable OPerating SYstem). It constitutes the framework for the practical exercises related to the course Computer Engineering II. This course is taught at the Department of Electrical Engineering at ETH Zürich and deals with the basic concepts of operating systems.*
>
> *When planning and working out the lecture we thought about the best way how to teach this subject. In our opinion the concepts can best be learned by a combination of theory and practice, so we were looking for an operating system which serves as*
>
> 1. *a basis for practical exercises, enabling the students to apply the knowledge acquired in the lecture lessons, and*
> 2. *an example how basic principles can be implemented in a real operating system.*
>
> *For us it was essential that the desired system improves the student s comprehension in the fields of process parallelism, communication and synchronization, interfacing hardware and memory management.“ [FCZP 00]*

Topsy stellt von den bisher vorgestellten Lehrbetriebssystemen die für unsere Bedürfnisse angemessenste Lösung dar, da es die wesentlichen Anforderungen erfüllt, so

- hervorragende Dokumentation der klaren Struktur,
- schnelle und leichte Einarbeitung für die Studenten,
- gut verständlicher, hervorragend dokumentierter Quellcode,
- umfassende Unterstützung von Betriebssystemkonzepten (einzig Dateisystem bleibt außen vor).

RCOS: Einen ungewöhnlichen Ansatz verfolgt `RCOS.java` (basierend auf dem in [ChJo 96] beschriebenen, in C implementierten RCOS):

> *„`RCOS.java` is a tool designed to help people understand the inner workings of an operating system. It began in 1996 by four people at Central Queensland University. `RCOS.java` is an animated, multi-tasking operating system running on simulated hardware. It is based on the P-Machine, a simple hypothetical computer system. It includes a C/C++ compiler that generates P-Code to be executed by the P-Machine.*
>
> *`RCOS.java` is designed to demonstrate general operating system principles through controlled animation, in the Java programming language. `RCOS.java` is implemented in Java using good code design. This allows students to modify, experiment with and compare different data structures and algorithms.“* `http://rcosjava.sourceforge.net/`

Dank der Animationskomponente ist es hier möglich, komplexe Abläufe für die Studenten eingängig zu visualisieren, beispielsweise das Scheduling, wie in Abbildung 9, oder der Abarbeitung von Instruktionen in der CPU (vgl. Abbildung 10).

Leider ist `RCOS.java` bisher in einem unbrauchbaren Zustand, seit Ende 2000 findet jedoch auf Sourceforge (`http://rcosjava.sourceforge.net/`) wieder eine aktive Weiterentwicklung statt.

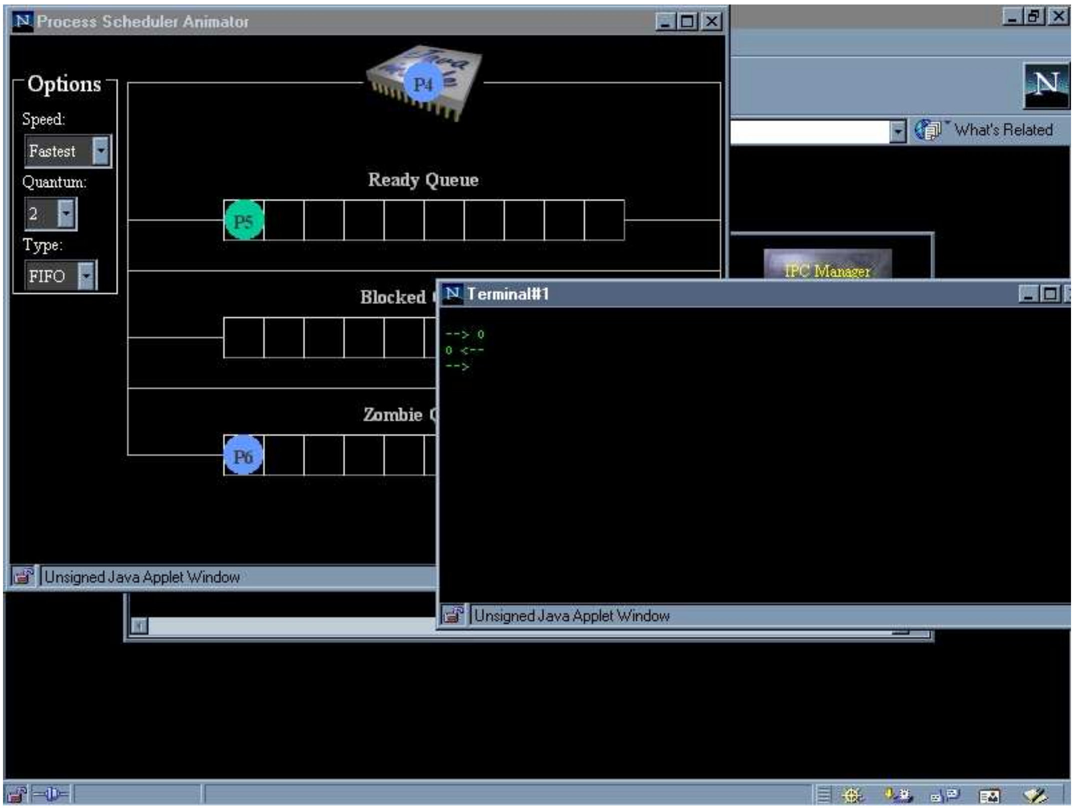


Abbildung 9: Scheduler und Terminal von `RCOS.java`

Bedauerlicherweise mußte auf den Einsatz eines Lehrbetriebssystems verzichtet werden, da hier die Einarbeitungszeit für die Studenten, vor allem jedoch der Betreuungsaufwand den zulässigen Rahmen gesprängt hätte. Weiterhin wäre dadurch das zentrale Ziel der Übungen, die praktische Erfahrung mit nebenläufiger Programmierung, in den Hintergrund getreten.

4.2 Nebenläufige Programmierung

Um die Probleme und Möglichkeiten der nebenläufigen Programmierung zu vermitteln, standen uns im wesentlichen zwei Ansätze zur Verfügung.
Die erste Alternative ist der Einsatz einer spezialisierten Programmiersprache mit der Unterstützung möglichst vieler der in der Vorlesung behandelten Konzepte zur Synchronisierung und Kommunikation. Ein Beispiel dieser Klasse, welches näher betrachtet wurde, ist **BACI**, der *Ben-Ari Concurrent Interpreter*, entwickelt nach [Ben 90] an der Colorado School of Mines ([ByCa 99]):

> *„Concurrency concepts and synchronization techniques are important issues in computer science. Due to the increasing emphasis on parallel and distributed computing, understanding concurrency and synchronization is more necessary than ever. To obtain a thorough understanding of these concepts, practical experience writing concurrent programs is needed. BACI is an option to obtain this desired 'hands-on' experience with concurrent programs.*
>
> *BACI stands for Ben-Ari Concurrent Interpreter. The compiler and interpreter originally were procedures in a program written by M. Ben-Ari, based on the original Pascal compiler by*

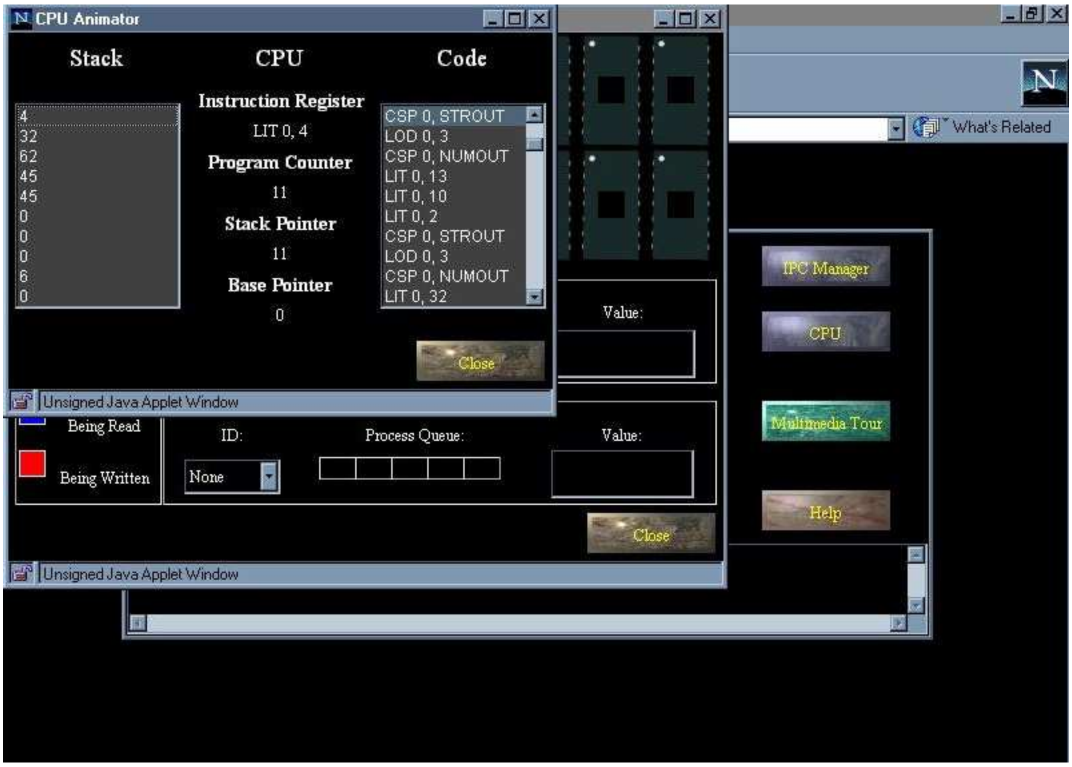


Abbildung 10: CPU Animator `RCOS.java`

> *Niklaus Wirth. The original version of the BACI compiler and interpreter was created from that source code and was hosted on a PRIME mainframe. After several modifications and additions, this version was ported to a PC version in Turbo Pascal, to Sun Pascal, and to C. Finally, the compiler and interpreter were split into two separate programs. Recently, a C– compiler has been added to the BACI suite of programs to compile source programs written in a restricted dialect of C++ into PCODE object code executable by the interpreter. Compared with other concurrent languages, BACI offers a variety of synchronization techniques with a syntax that is usually familiar. Any experienced C or Pascal programmer could use BACI within hours.“ [ByCa 99]*

```
const int m = 5;

      int n;

void incr (char id) {

    int i;

    for (i = 1; i <= m; i = i + 1) {
          n = n +1;
          cout << id << "␣n␣=" << n  <<  "␣i␣=";
          cout << i  << "␣"    << id << endi;
    }
}

void main() {
    n = 0;
    cobegin {
        incr('A' ); incr( 'B' ); incr( 'C');
    }
    cout << "The␣sum␣is␣" << n << endl;
}
```

Neben den offensichtlichen Vorteilen, wie der einfacheren Anwendung der verschiedenen Konzepte zur Synchronisation, haben jedoch derartige Lösung einige Probleme:

- Die Studenten müssen sich mit einer neuen Syntax vertraut machen. Dies ist nachteilig, insbesondere da in den vorangehenden Semestern keine imperative Programmiersprache, wie Pascal oder C, betrachtet wird.
- Die Dokumentation und Begleitliteratur ist eher unzureichend.
- Es können zwar die Konzepte der Vorlesung eingängig vermittelt werden, jedoch fehlt die Erfahrung, wie diese Konzepte in „realen“ Programmiersprachen umgesetzt werden.

Letztendlich ist die Entscheidung zugunsten des Einsatzes von **Java** gefallen. Java bietet in dem gegebenen Umfeld eine Reihe von Vorteilen:

- Java enthält eine API zum Umgang mit Threads und zur Synchronisation nebenläufiger Vorgänge.
- Die Studenten werden bereits im zweiten Semester mit den Grundlagen der Programmierung in Java vertraut gemacht und nehmen zu einem großen Anteil parallel zur Vorlesung an einem Programmierpraktikum unter Verwendung von Java teil.
- Es gibt ausreichend Literatur zum Umgang mit Threads unter Java, davon wurde neben der allzu oft benötigten *Java Language Specification* [GJSB 00] in erster Linie auf [Lea 00] und [Holu 00] zurückgegriffen. Besonders günstig hat sich hier erwiesen, daß es mit [SGG 00] ein Betriebssystembuch gibt, das nahezu alle behandelten Algorithmen in Java vorstellt.
- Auch der Betreuungsaufwand ist bei Verwendung von Java überschaubarer, da weder Übungsleiter noch Tutoren sich in eine neuen Sprache bzw. System einarbeiten müssen.

Anders als in „Technische Grundlagen der Informatik“ kann Java jedoch unsere Zielvorstellungen zu einem bedeutenden Anteil nicht erfüllen, hier mußte aus oben genannten Gründen auf vieles verzichtet werden. Die wesentlichen Probleme, die sich durch den Einsatz von Java ergeben haben, sind:

- Durch die Struktur von Java ist es nahezu unmöglich einen Einblick in die Implementierung der Thread-API zu werfen. Insbesondere Fragen wie Scheduling, Kontextswitch oder Speicherverwaltung sind implementationsabhängig und nicht in Java realisiert.

  Weiterhin ist es auch schwer möglich, die Studenten beispielsweise den Ablauf des Schedulings beobachten zu lassen, da [GJSB 00] nur grobe Rahmenannahmen vorschreibt, alle anderen Abläufe extrem stark abhängig sind von der jeweils verwendeten JVM-Implementation.
- Der Wunsch nach „plattformunabhängigem“ Zugriff auf Threads führt dazu, daß viele entscheidende Fragen, insbesondere beim Scheduling, von Implementation zu Implementation anders gehandhabt werden, so unterstützen beispielsweise eine Reihe von JVM-Implementationen unter Linux kein Timeslicing, sondern verwenden kooperatives Mulithreading. Dies führt dazu, das sämtliche Aufgaben, die Gefahren des Multithreadings demonstrieren sollen (z.B. unsynchronisierter Zugriff auf gemeinsame Speicherbereiche), nur zum Teil verwendet werden können.

  Schließlich ist der Umgang mit Threads in einigen JVM-Implementationen unter Linux nicht einmal konform zu [GJSB 00].
- Es gibt überraschend wenig andere Universitäten, die Java in einem Betriebssystemkurs einsetzen, einen (wenn auch nicht mehr aktuellen) Überblick mag `http://www.gac.edu/~max/java-os-courses.html` geben. Dadurch müssen nahezu alle Java Aufgaben neu entwickelt oder portiert werden.

- Die Java-Thread-API zählt nicht zu den Glanzpunkten von Java. Dies zeigt sich beispielsweise an den drastischen Änderungen an der API, die in Version 1.2 durchgeführt (`Thread.stop`, `Thread.suspend` und `Thread.resume` deprecated) wurden und auf heftige Kritik von Seiten der Nutzer (so in [Holu 00]) getroffen sind.

  Auch wirft die Verwendung von Threads an vielen Stellen Probleme auf, meist aufgrund von der Implemtation der Threads (beispielsweise Thread-Cache).

- Am gravierendsten im vergangenen Semester war jedoch der erschreckende Mangel an Vorkenntnissen auf Seiten der Studenten. Es scheint nicht gelungen zu sein, der Mehrzahl der Studenten in „Informatik II“ Grundkenntnisse in Java zu vermitteln, auf denen zügig aufgebaut werden kann. Im Laufe des Semesters scheint sich dies (u.U. dank des Programmierpraktikums) etwas gebessert zu haben.

Nachdem nun ein Überblick über die untersuchten Plattformen für „Technische Grundlagen der Informatik“ und „Informatik III“ gegeben und deren Vor- und Nachteile für den Einsatz im Rahmen der Übungen zu den genannten Vorlesungen diskutiert wurden, soll nun noch ein kurzer Überblick über den Entwurf der Übungsblätter anhand einiger Beispiele gegeben werde.

5 Entwurf der Übungsblätter

In beiden Veranstaltungen wird in den Übungen ein großer Schwerpunkt auf die praktische Umsetzung der Konzepte der Vorlesung durch Programmieraufgaben gelegt. Dies hatte einige Folgen für die Strukturierung der Übungsblätter:

- Um den Studenten die Zuordnung der Aufgaben und die Wiederholung für Klausur bzw. Prüfung zu erleichtern, werden die Übungsaufgaben in vier Klassen eingeteilt (vgl. Abbildung

  Hausaufgaben (H): Hausaufgaben dienen zur Wiederholung der Vorlesung, können i.a. in recht kurzer Zeit bearbeitet werden und sollen durch die Studenten vor der jeweiligen Übungsstunde bearbeitet werden, so daß in der Übungsstunde allenfalls noch Fragen und Probleme behandelt werden müssen.

  Klausuraufgaben (K): Klausuraufgaben sollen den Studenten als Orientierung und Prüfung ihrer Kenntnisse dienen. Es sind im wesentlichen Hausaufgaben auf dem Niveau einer Klausur.

  Tutoraufgaben (T): Wesentlich neue Lehrinhalte oder komplexere Problemstellungen werden in Tutoraufgaben behandelt. Diese sind durch die Studenten allein nur mit deutlich höherem Aufwand zu lösen und sollen in der Übungsstunde ausführlich besprochen werden. Häufig dienen sie zur Einführung von für die Programmierung benötigten Konzepten, die in der Vorlesung gar nicht oder nur kurz behandelt werden.

  Programmieraufgaben (P): Durch die Programmieraufgaben sollen die Studenten schließlich die in der Vorlesung bzw. in den Tutoraufgaben eingeführten Konzepte praktisch Umsetzen. Dazu ist auf jedem Übungsblatt wenigstens eine Programmieraufgabe vorgesehen, es gibt jedoch durchaus auch Übungsblätttter mit drei oder sogar vier Aufgaben dieses Typs.

- Da der Schwierigkeitsgrad vor allem der Programmieraufgaben stark variieren kann und mit Rücksicht darauf, daß in Zukunft vielleicht doch wieder eine Bewertung der Übungsblätter vorgenommen wird, wird als Orientierung für die Studenten zu jeder Teilaufgabe eine Punktzahl angegeben, die eine Einschätzung des Bearbeitungsaufwands möglich macht.

- Das letzte Übungsblatt eines Semesters soll als Wiederholung und Test zur Feststellung der eigenen Lücken für die Studenten dienen, wie Abbildung 12 zeigt.

- Für jedes Übungsblatt wird eine Liste von Literatur angegeben, die gelesen werden soll. Dabei wird unterschieden in Pflichtliteratur, Literatur zur Wiederholung und solche zur Vertiefung (vgl. Abbildung 13).

Basierend auf diesen Grundprinzipien wurden insgesamt 13 Übungsblätter mit etwa 121 Aufgaben für „Technische Grundlagen der Informatik“ und 14 Übungsblätter mit über 110 Übungsaufgaben (dazu über 55 Java-Programme) für „Informatik III“ übernommen, angepaßt, portiert oder neu erstellt.

6 Erfahrungen

Im Sommersemester 2000 bzw. Wintersemester 2000/01 konnten bereits Erfahrungen mit der neuen Struktur der Vorlesung und der Übungen gesammelt werden. Es wurden keine wesentlichen Probleme in der Übungsstruktur aufgedeckt, außer der – bereits oben angedeuteten – Schwächen bei der Benutzung von Java.

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Sommersemester 2000
Übungsblatt 10
11.07.2000

Technische Grundlagen der Informatik

Achtung: Bitte beachten Sie die **Hinweise zur Prüfungsanmeldung** auf der Homepage von Prof. Dr. Linnhoff–Popien (über die TGI–Homepage erreichbar), falls Sie eine Vordiplomsprüfung in TGI/Info3 in den **kommenden Semesterferien** in betracht ziehen.

Lesen: Tutorium Kap. 13.2
Wiederholen: Schleifen, ISA, Rekursion
Vertiefen: PatHen Kap. 5.1 – 5.3; Tanenbaum Kap. 2.2.4, Kap. 4.1

Aufgabe 80: (H) Fehlerkorrekturcode (Hamming) (6 Pkt.)

Der sog. *Hamming Algorithmus* kann zur Entwicklung eines Fehlerkorrekturcodes für Speicherwörter beliebiger Größe herangezogen werden. D.h., dass aufgetretene Fehler nicht nur erkannt sondern auch korrigiert werden können. Der Algorithmus funktioniert folgendermaßen:

– Die Bits werden mit 1 beginnend durchgezählt, wobei Bit 1 das höherwertige Bit ist (ganz links).

– Alle Bits, deren Bitnummer eine Potenz von 2 ist, sind *Paritätsbits*, die zur Überprüfung der Datenbits im Speicherwort herangezogen werden. Alle übrigen Bits sind die tatsächlichen Datenbits

– Um bei einem Speicherwort der Länge m eine Fehlerkorrektur durchführen zu können, sind demnach zusätzliche r Paritätsbit notwendig; es ergibt sich demnach ein Speicherwort der Länge $m + r$.

– Ein Bit b an der Stelle n wird von denjenigen Paritätsbits geprüft, deren Summe der Indizes die Stellennummer n des Bits ergeben. Demnach wird das Bit an der Stelle 5 von den Bits an den Stellen 1 und 4 geprüft, da $1 + 4 = 5$. Analoges gilt für Bit 6, dass von den Bits 2 und 4 geprüft wird.

– Das Paritätbit ist gesetzt, falls die Anzahl der gesetzten und von diesem zu überprüfenden Datenbits *ungerade* ist.

– Eine einfache Methode zum Auffinden von falschen Bits ist, die Summe der Indizes der fehlerhaften Paritätsbits zusammenzuzählen. Die Summe ergibt die Stelle des falsch gesetzten (= invertierten) Datenbit. Wären die Paritätsbits 1 und 4 falsch, jedoch die restlichen Paritätsbits richtig, so wäre das Bit 5 fälschlicherweise invertiert worden.

a. Geben Sie die Anzahl und die Position der Paritätsbits bei einem ursprünglich 16–Bit langem Speicherwort an. Wie groß ist der Overhead in % ?

b. Geben Sie zu den folgenden 16–Bit Speicherworten die aus obigem Algorithmus resultierenden und mit Paritätsbits versehenen Speicherworte an:

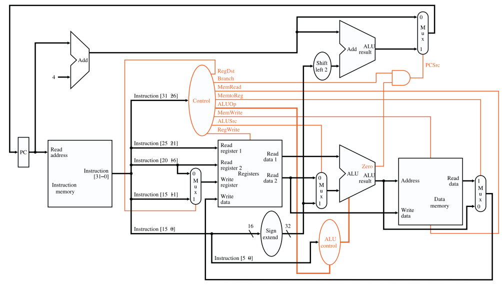
(i) 1111000010101110
(ii) 1010010011001000
(iii) 1000001010010001
(iv) 1000001010010001

c. Prüfen Sie, ob die folgenden Speicherworte fehlerhafte Bits enthalten:

(i) 111110100101
(ii) 111100100000
(iii) 001000010010
(iv) 110000011110

Aufgabe 81: (H) Single-cycle Datapath (6 Pkt.)

Fügen Sie die Instruktion `bne` (***branch if not equal***) zu dem folgenden *single-cycle* Datenpfad hinzu:



Zeigen Sie auch, welche Änderungen sich für die folgende Abbildung aus ihrer Erweiterung ergeben:

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1

Aufgabe 82: (H) Beschränkungen des Datenpfads (3 Pkt.)

Warum ist es nicht möglich, den single-cycle Datenpfad so zu modifizieren, daß eine `swap`-Anweisung (vertauscht zwei Register) realisiert wird, ohne das Registerfile zu verändern?

Aufgabe 83: (T) Multi-cycle Datenpfad (7 Pkt.)

Häufig stellt der Speicherzugriff für Lade- und Speichervorgänge den zeitkritischen Teil des multi-cycle Datenpfads dar. Implementiert man Lade- und Speichervorgänge in einem Zyklus, so wird dadurch die Länge des Datenpfades unnötig erhöht. Eine Lösung ist es, alle Lade und Speichervorgänge in zwei Zyklen aufzuspalten.
Untersuchen Sie, ob diese Idee tatsächlich zu einer Leistungsverbesserung führt unter den folgenden Annahmen:

– Die bisherige Implementation beschränkt die Geschwindigkeit auf 500 MHz. Durch die Aufteilung läßt sich die Geschwindigkeit auf 750 MHz erhöhen.

– Benutzen Sie den folgenden Anweisungs-Mix (für den `gcc`-Compiler) für Ihre Bewertung:

Anweisungstyp	Häufigkeit	CPI 500 MHz
Loads	21%	5
Stores	12%	4
R-type	46%	4
Jump/branch	21%	3

– Nehmen Sie schließlich an, daß alle Jumps/Branches dieselbe Anzahl Zyklen benötigen und daß alle übrigen Anweisungen als R-type- (Register-) Instruktionen realisiert sind.

Aufgabe 84: (K) Operationen auf Feldern (6 Pkt.)

Das folgende Programmfragment durchläuft ein Feld und schreibt zwei Werte in die Register `$v0` und `$v1`. Nehmen Sie an, daß das Feld aus 5000 Wörtern, indiziert von 0 bis 4999, besteht, seine Anfangsadresse in `$a0` und seine Größe in `$a1` gespeichert wird. Beschreiben Sie unter diesen Annahmen, was das Programmfragment bewirkt, und welche Werte in `$v0` und `$v1` zurückgegeben werden:

```
        add     $a1, $a1, $a1
        add     $a1, $a1, $a1
        add     $v0, $zero, $zero      # Wie lautet die Pseudoinstr.?
        add     $t0, $zero, $zero
outer:  add     $t4, $a0, $t0
        lw      $t4, 0($t4)
        add     $t5, $zero, $zero
        add     $t1, $zero, $zero
inner:  add     $t3, $a0, $t1
        lw      $t3, 0($t3)
        bne     $t3, $t4, skip
        addi    $t5, $t5, 1            # Wie lautet die Pseudoinstr.?
skip:   addi    $t1, $t1, 4
        bne     $t1, $a1, inner
        slt     $t2, $t5, $v0
        bne     $t2, $zero, next       # Wie lautet die Pseudoinstr.?
        add     $v0, $t5, $zero
        add     $v1, $t4, $zero
next:   addi    $t0, $t0, 4
        bne     $t0, $a1, outer
```

Aufgabe 85: (P) Endrekursion (3+5+1 Pkt.)

Gewisse rekursive Prozeduren können bekanntlich einfach in eine Iteration umgewandelt werden, wodurch im allgemeinen der Aufwand für die Berechnung drastisch gesenkt werden kann. Betrachten Sie zum Beispiel das folgende Programmfragment:


```
PROCEDURE sum(n, acc :  INTEGER): INTEGER;

BEGIN
   IF (n > 0) THEN
      RETURN sum(n - 1, acc + n)
   ELSE
      RETURN acc;
END; { sum }
```

Angenommen es wird `sum(3,0)` berechnet. Das resultiert in rekursiven Aufrufen `sum(2,3)`, `sum(1,5)` und `sum(0,6)`, das Resultat 6 wird dann vier mal zurückgegeben. Ein derartiger rekursiver Aufruf wird als Endaufruf (*tail call*), die Rekursion als Endrekursion (*tail recursion*) bezeichnet. Diese kann sehr effizient als Iteration realisiert werden:

```
sum:    beq     $a0, $zero, sum_exit   # sum_exit, wenn n = 0
        add     $a1, $a1, $a0          # addiere n auf acc
        addi    $a0, $a0, -1           # subtrahiere 1 von n
        j       sum
sum_exit:
        move    $v0, $a1               # gibt acc zurück
        jr      $ra                    # zurück zum Caller
```

a. Schreiben Sie ein MIPS-Programm, das die n-te Fibonacci-Zahl $F(n)$ berechnet. Ihr Algorithmus sollte auf der folgenden offensichtlichen, aber hoffnungslos ineffizienten Lösung basieren:

```
PROCEDURE fib(n :  INTEGER): INTEGER;

BEGIN
   IF (n = 0) THEN
      RETURN 0
   ELSE
      IF (n = 1) THEN
         RETURN 1
      ELSE
         RETURN fib(n-1) + fib(n-2);
END; { fib }
```

b. Schreiben Sie nun ein MIPS-Programm, daß ebenfalls die n-te Fibonacci-Zahl berechnet, aber auf der folgenden Prozedur basiert:

```
PROCEDURE fib_iter(prev, preprev, count:  INTEGER): INTEGER;

BEGIN
   IF (count = 0) THEN
      RETURN preprev
   ELSE
      RETURN fib_iter(prev + preprev, prev, count-1);
END; { fib_iter }
```

Die beiden ersten Parameter enthalten die vorherigen beiden Fibonacci-Zahlen. Um $F(n)$ zu berechnen, muß man die Prozedur mit `fib_iter(1,0,n)` aufrufen. Optimieren Sie Ihr Programm!

c. Schätzen Sie den Geschwindigkeitsunterschied zwischen den beiden Programmen.

Abbildung 11: „Technische Grundlagen der Informatik“, Blatt10, Aufgabenblatt, Seiten 1-4

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Wintersemester 2000/01
Übungsblatt 13
30.01.2000

Informatik III

Wiederholen: Silberschatz, Stallings, Tannenbaum

Aufgabe 66: (H) Wiederholung (3+1+1+8+4+4+4+6+6+6 Pkt.)

Die folgenden Aufgaben stammen zum größten Teil aus Klausuren der Central Queensland University, Australien:

a. Ergänzen Sie folgenden Lückentext:

1) Die zwei gültigen Opertionen auf einer Semaphore sind .... und ....
2) Zwei Beispiele für das Management von virtuellem Speicher sind .... und ....
3) Ein geläufiges Programmierkonstrukt zur Implementation von wechselseitigem Ausschluß unter Nutzung von Bedingungsvariablen wird als ....

b. Der SJF-Algorithmus ist

1) ein präemptiver CPU-Schedulingalgorithmus.
2) ein Disk-Schedulingalgorithmus.
3) eine Modifikation des Round Robin CPU-Schedulingalgorithmus.
4) ein nicht-präemptiver CPU-Schedulingalgorithmus.

c. Welche der folgenden Prozeßzustandsübergänge sind ungültig?

1) ready → running.
2) ready → blocked.
3) blocked → ready.
4) running → dead.

d. Erklären bzw. beantworten Sie kurz die folgenden Fragen:

(1) Multiprogramming.
- Warum wird es verwendet?
- Welche Probleme entstehen durch seine Verwendung?

(2) Semaphoren.
- Was sind Semaphoren?
- Wozu dienen Sie?
- Welche Operationen auf ihnen gibt es und welche Aufgabe haben diese?

(3) Fassen Sie die Argumente zusammen für und gegen kleine bzw. große *page size*.

(4) Starvation.

- What versteht man darunter?
- Wodurch entsteht es?
- Wie kann ein Betriebssystem/Algorithmus es vermeiden?
- Welcher andere Ausdruck wird dafür auch verwendet?

e. Was ist ein PCB? Wozu dient er? Welche Informationen werden dort gespeichert? Wozu dienen diese Informationen? Welche Operationen können auf ihm ausgeführt werden? Erläutern Sie ausführlich!

f. Angenommen zwei Prozesse A und B enthalten beide die folgenden Befehle in einer Schleife:

```
int Numres = 0;
Numres = Numres + 1;
if Numres == 50 then Numres = 0;
```

Nehmen Sie weiterhin an, daß beide die Variable `Numres` teilen und beide aktiv sind. Erklären Sie, wie selbst bei nur einer CPU es eintreten kann, daß `Numres` nie auf 0 zurückgesetzt wird.

g. Beschreiben Sie die Ideen und Konzepte im Umfeld von Monitoren.

h. Schreiben Sie (in *Pseudocode*) einen Monitor, der sich wie eine binäre Semaphore verhält.

i. Ein System habe drei identische Drucker, so daß ein Prozeß unterschiedslos auf jedem der drei Druck drucken kann. Schreiben Sie (in *Pseudocode*) einen Monitor, der Prozessen erlaubt, einen Drucker sicher zu „belegen“, zu verwenden und dann „freizugeben“. Die Procedure-Deklarationen sollten wie folgt aussehen:

```
procedure acquire(var id : integer);
procedure release(id : integer);
```

j. Betrachten Sie eine Variante des Round Robin-Schedulingalgorithmus, in der die Einträge in der Warteschlange Zeiger auf die PCBs sind.

(i) What wäre der Effekt, wenn man zwei Zeiger auf den selben Prozeß in die Warteschlange einfügt.
(ii) Was wären die Hauptvor- bzw. -nachteile dieses Verfahrens?
(iii) Wie würden Sie den ursprünglichen Round Robin-Algorithmus verändern, um den selben Effekt ohne doppelte Zeiger zu erhalten.

Aufgabe 67: (H) Java-Certification Exam (10 Pkt.)

Die folgenden Aufgaben sind Übungsaufgaben zur Vorbereitung der Java-Certification Prüfung:

a. Was wird passieren, wenn Sie versuchen den folgenden Code zu kompilieren und auszuführen?

```
abstract class Base{
        abstract public void myfunc();
        public void another(){
        System.out.println("Another method");
        }
}

public class Abs extends Base{
        public static void main(String argv[]){
```



```
        Abs a = new Abs();
        a.amethod();
        }
        public void myfunc(){
                System.out.println("My Func");
                }
        public void amethod(){
        myfunc();
        }
}
```

1) Der Code wird kompiliert und ausgeführt, wobei „My Func“ ausgedruckt wird.]
2) Der Compiler bemängelt, daß die `Base`-Klasse keine abstrakte Methode hat.
3) Der Code wird kompiliert, aber es tritt ein Laufzeitfehler auf: Die `Base`-Klasse hat keine abstrakten Methoden.
4) Der Compiler wird bemängeln, daß die Method `myfunc` in der `Base`-Klasse keinen Funktionskörper besitzt.

b. Was wird passieren, wenn Sie versuchen den folgenden Code zu kompilieren und auszuführen?

```
public class MyMain{
public static void main(String argv){
        System.out.println("Hello cruel world");
        }
}
```

1) Der Compiler bemängelt, daß `main` ein reserviertes Wort ist und nicht für eine Klasse verwendet werden kann.
2) Der Code wird kompiliert und bei der Ausführung „Hello cruel world“ ausgeben.
3) Der Code wird kompilieren, aber zur Laufzeit wird bemängelt, daß kein Konstruktor definiert ist.
4) Der Code wird kompilieren, aber zur Laufzeit wird bemängelt, daß `main` nicht korrekt definiert ist.

c. Welche Gründe gibt es, eine Methode als *native* zu definieren:

1) Um Zugriff auf Hardware zu bekommen, die von Java nicht unterstützt wird.
2) Um einen neuen Datentyp wie z.B. `unsigned integer` zu definieren.
3) Um optimierten Code zu schreiben.
4) Um die Beschränkung des `private` Gültigkeitsbereichs einer Methode aufzuheben.

d. Sie wollen den Wert des letzten Elements eines Arrays mit dem folgenden Code herausfinden. Was passiert, wenn Sie ihn kompilieren und ausführen?

```
public class MyAr{
public static void main(String argv[]){
        int[] i = new int[5];
        System.out.println(i[5]);
        }
}
```

e. Was wird passieren, wenn Sie versuchen den folgenden Code zu kompilieren und auszuführen?


```
public class Bground extends Thread{
public static void main(String argv[]){
                Bground b = new Bground();
                b.run();
        }
        public void start(){
                for (int i = 0; i <10; i++){
                        System.out.println("Value of i = " + i);
                }
        }
}
```

f. Was kann einen Thread dazu bringen, die Ausführung zu unterbrechen?

1) Das Programm beendet sich mit einem Aufruf von `System.exit(0);`.
2) Einem anderen Thread wird höhere Priorität gegeben.
3) Ein Aufruf der `stop`-Methode des Threads.
4) Ein Aufruf der `halt`-Methode des Threads.

g. Unter welchen Umständen ist es sinnvoll die `yield`-Methode der Thread-Klasse aufzurufen.

1) Aufruf auf den aktuell laufenden Thread, um einem anderen Thread der gleichen oder höherer Priorität die Abarbeitung zu erlauben.
2) Aufruf auf einem wartenden Thread, um ihm zur Ausführung zuzulassen.
3) Um einem Thread mit höherer Priorität die Abarbeitung zu erlauben.
4) Aufruf auf dem aktuell laufenden Thread mit einem Parameter, der kennzeichnet, welcher Thread als nächstes ausgeführt werden soll.

h. Welche der folgenden Aussagen über Threads sind wahr?

1) Man kann einen wechselseitigen, exklusiven Lock für Methoden in einer Klasse erhalten, die die `Thread`-Klasse erweitert oder das Interface `runnable` implementiert.
2) Man kann einen wechselseitigen, exklusiven Lock für jedes Objekt erhalten.
3) Ein Thread kann einen wechselseitigen, exclusiven Lock für eine `synchronized` Method eines Objekts erhalten.
4) Thread scheduling algorithms are platform dependent

i. Welche der folgenden sind Methoden der Thread-Klasse?
(1) `yield()`, (2) `sleep(long msec)`, (3) `go()`, (4) `stop()`.

j. Welche der folgenden Aussagen beschreibt am besten die Funktionsweise des `synchronized`-Schlüsselwortes?

1) Erlaubt zwei Prozessen parallel ausgeführt zu werden, aber miteinander zu kommunizieren.
2) Stellt sicher, daß nur ein Thread zur selben Zeit auf eine Methode oder ein Objekt zugreifen kann.
3) Stellt sicher, daß zwei oder mehr Prozesse zur selben Zeit starten und enden werden.
4) Stellt sicher, daß zwei oder mehr Threads zur selben Zeit starten und enden werden.

k. An welchen Methodenaufrufen für die Klasse Thread sind zwei, an welchen nur ein Prozeß beteiligt?

Abbildung 12: „Informatik III“, Blatt14, Aufgabenblatt, Seiten 1-4 (Seiten 5-6 ausgelassen)

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Sommersemester 2000
Übungsblatt 7
20.06.2000

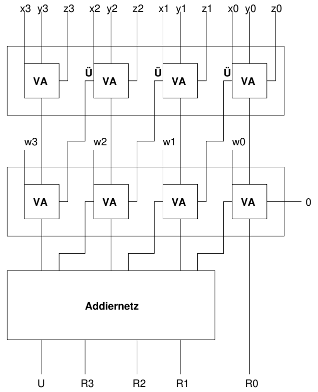
Technische Grundlagen der Informatik

Achtung:
- Die **Anmeldung zur 2. Klausur** ist noch bis zum **Freitag, 23. Juni 12 Uhr** möglich. Die Anmeldung ist verpflichtend!
- Die 2. Klausur findet am Dienstag, 27. Juni um **13 Uhr s.t.** in den Räumen E04, E05, 122 (Theresienstr. 39) statt. Die Aufteilung der Studierenden auf die Räume wird ab Montag, 26. Juni 12 Uhr auf der TGI–Homepage veröffentlicht.
- Es sind **keine Hilfsmittel** erlaubt. Es wird allerdings eine Liste mit den wichtigsten SPIM–Befehlen ausgeteilt.
- **Mitzunehmen** sind unbedingt der **aktuelle Studentenausweis** und ein **gültiger Personalausweis**.

Lesen: Tutorium Kapitel 11/12
Wiederholen: Arrays, Linearisierung von Arrays (siehe Info3–Skript WS 1999/2000, Kapitel 7.2), Keller (Info3–Skript, Kap. 7.3)
Vertiefen: ObVos Kap. 2.3 u. 4.2, PatHen Anh. B4-B5; Tannenbaum Kap. 5

Aufgabe 34: (H) Carry-Save-Addiernetz (2+2+2+4 Pkt.)

Betrachten Sie die Funktionsweise und den Aufbau des Carry-Save-Addiernetzes:



a. Welcher funktionelle Unterschied besteht zum Ripple-Carry-Addierer?

b. Welche Vor- und Nachteile hat es gegenüber den in der Vorlesung behandelten Ripple-Carry-Addierern bezüglich Zeitverhalten bzw. Schaltungsaufwand?

c. Berechnen und vergleichen Sie die Zeit, die die einzelnen Addiernetze zur Addition von vier 8-Bit Zahlen benötigen. Nehmen Sie dazu an, daß die Zeitverzögerung für jedes grundlegende Schaltelement 1 Takt beträgt.

   Hinweis: Halb- und Volladdierer gelten nicht als grundlegende Schaltelemente.

Aufgabe 35: (H) Test auf Überlauf (5 Pkt.)

Bei den bekannten Addierern kann man als Test auf Überlauf einfach überprüfen, ob der eingehende Übertrag (CarryIn) des höchstwertigen Bits ungleich zu seinem ausgehenden Übertrag (CarryOut) ist.
Beweisen Sie, daß dieser Test korrekt ist, daß also genau dann ein Überlauf eintritt.

Aufgabe 36: (H) Berechnung der durchschnittlichen CPI (2+2 Pkt.)

Betrachten Sie die folgende Tabelle über den Instruktionsmix für ein gegebenes Programm A und über die Geschwindigkeit eines gegebenen Prozessors:

Instruktions-Typ	Zyklen	Häufigkeit in Programm A
ADD/SUB	2	50%
Sprünge	4	25%
Verschiedene	12	25%

a. Berechnen Sie die durchschnittliche Zahl der Taktzyklen pro Instruktion (CPI).

b. Nehmen Sie an, daß der Prozessor mit 200 MHz läuft und das obige Programm in 30 Sekunden ausgeführt wird, wie viele Instruktionen werden dann in diesem Programm ausgeführt.

Aufgabe 37: (T) Einfache ALU (4 Pkt.)

Entwerfen Sie eine einfache 1-Bit ALU, die den folgenden Spezifikationen genügt:

i. Operationen: `AND`, `OR`, Addition und Subtraktion.

ii. Inputs: Operanden a und b, CarryIn (Übertrag aus einer vorgeschalteten ALU), gewisse Steuerleitungen (z.B. zur Auswahl des Typs der Operation).

iii. Outputs: Resultat, CarryOut (Übertrag).

Hinweis: Die Steuerleitungen sind ebenfalls anzugeben. Beschreiben Sie die Arbeitsweise Ihrer ALU. Beschreiben Sie mögliche Erweiterungen.

Aufgabe 38: (T) Mehrdimensionale Felder (5+10+2 Pkt.)

Sei die folgende Typdefinition für ein mehrdimensionales Feld gegeben:

```
TYPE T = ARRAY I1, I2, ..., Ik OF T0.
```

`T0` heißt Grundtyp des Felds, `I1` bis `Ik` heißen die k Indextypen des Felds (endliche Untermengen der natürlichen Zahlen), ihre Kardinalitäten seien durch `s1` bis `sk` bezeichnet. Mit `LOP` (*length of operand*) bezeichnet man die Länge des Grundtyps `T0` in Byte.
Die Selektion einzelner Elemente erfolgt durch Angabe konkreter Werte für *sämtliche* Indizes, z.B. `a[i1, ..., ik]`, in *konstanter* Zeit.
Beispiel: Betrachten Sie die folgende Deklaration der Feldvariablen `A`:

```
VAR A = ARRAY[0..5,0..78,0..42] OF INTEGER.
```

Hier ist `INTEGER` der Grundtyp, damit `LOP` $= 4$ (für einen Rechner mit 32-Bit Integerzahlen), die Indextypen sind `I1 = [0..5]`, `I2 = [0..87]` und `I3 = [0..42]`, ihre Kardinalitäten also 6, 79 bzw. 43. Man beachte, daß die untere bzw. obere Grenze der Indextypen konstant sein müssen (da sonst die Adressberechnung nicht mehr in konstanter Zeit garantiert werden kann).

Unter diesen Voraussetzungen bearbeiten Sie die folgenden Aufgaben:

a. Überlegen Sie sich, wie sie das dreidimensionale Feld aus dem Beispiel geschickt auf einen linearen Speicher abbilden können. Nehmen Sie an, daß die Anfangsadresse des Felds 4000 sei. Berechnen Sie die Adresse des Elements `A[3,13,42]`.

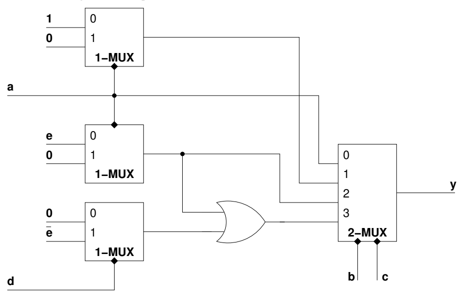
b. Überlegen Sie sich, wie ein mehrdimensionales Feld derartig (mit Hilfe einer Adressfunktion) auf einen linearen Speicher abgebildet werden kann, daß der wahlfreie Zugriff auf die Elemente des Felds in konstanter Zeit möglich ist. Erläutern Sie die Idee ihrer Adressfunktion.

   Hinweis: Sie können davon ausgehen, daß die unteren Grenzen aller Indextypen 0 sind.

c. Was müssen Sie an ihrer Adressfunktion verändern, wenn nicht alle unteren Grenzen der Indextypen 0 sind?

Aufgabe 39: (K) Multiplexer-Schaltnetz (6 Pkt.)

Gegeben sei das folgende Multiplexer-Schaltnetz:



Ermitteln Sie aus dem MUX-Schaltnetz den booleschen Ausdruck für die Schaltfunktion $y = f(a, b, c, d, e)$ und formen Sie diesen nach den Regeln der Booleschen Algebra in eine zweistufige disjunktive Form um. Vereinfachen Sie das Ergebnis soweit wie möglich.

Aufgabe 40: (P) Arithmetische Ausdrücke (10+2+1+2 Pkt.)

Schreiben Sie ein Programm, daß einen ungeklammerten Postfix-Ausdruck ($(4+3)-2$ wird notiert als $43+2-$) als Zeichenkette übergeben bekommt und ihn auswertet. Die Zeichenkette enthält nur die Ziffern 0 bis 9 und die Zeichen für die vier Grundrechenarten $+$, $-$, $\cdot$ und $/$. Als Eingabewerte sind zur Vereinfachung nur Ziffern zugelassen, also die Zahlen 0 bis 9.

a. Testen Sie das Programm durch mit dem folgenden arithmetischen Ausdruck (Eingabe ohne Leerzeichen):

```
4 5 + 2 6 * * 3 7 2 1 * - 5 + * 2 1 8 - 8 * + * -
```

   Geben Sie den Parse-Baum dafür an.

b. Überlegen Sie sich, ob ihr Programm auch die Summe der ersten n Zahlen berechnen kann ($1\ 2\ 3\ \ldots\ n-1\ n\ +\ +\ \ldots\ +$) für n beliebig (begrenzt nur durch den vorhandenen Speicher).

c. Überlegen Sie sich welche Vor- und Nachteile das Programm hat, insbesondere bezüglich Speicherzugriffen und Programmlänge. Wann ist es sinnvoll Kellerzugriffe zu verwenden, wann nicht?

Abbildung 13: „Technische Grundlagen der Informatik“, Blatt07, Aufgabenblatt, Seiten 1-4

A Erster Anhang: Verzeichnisstruktur

```
/ --------------------------------------------------------------------
  Makefile:            Aufruf von make <...> führt diesen Befehl in allen *blatt*
Unterverzeichnissen aus.
  Makefile.sub: Regeln für Makefiles in blatt* Unterverzeichnissen.
  Makefile.sub.DEF: Definitionen für Makefiles in blatt* Unterverzeichnissen.

/.makros -------------------------------------------------------------
  standard.sty:        Dieses Stylesheet lädt einige allgemeine Pakete und stellt Befehle
bzw. Abkürzungen für häufig benötigte Aufgaben bereit.
  tgi_uebungen.sty: In diesem Stylesheet werden die spezifischen Anpassungen für
die Übungsblätter durchgeführt und die Umgebungen und Kommandos
definiert.

  /.makros/listings -------------------------------------------------
    LISTINGS-Paket, Version 0.20, 12.07.1999
Ermöglicht den "Pretty-Print" von Programmfragmenten.

/.tmp ----------------------------------------------------------------
  In diesem Verzeichnis werden von LaTeX bzw. PDFLaTeX erzeugte temporäre Dateien abgelegt.
  Aufgrund der Beschränkungen von PDFLaTeX müssen alle .pdf-Dateien in $(PICDIR) (Standard
  /.repos/abbild) hierher gelinkt werden. Dies erfolgt automatisch im Makefile.

/blattXX -------------------------------------------------------------
  Makefile: Makefile zur Erzeugung des jeweiligen Blattes, definiert $(TOP), ansonsten
include von /Makefile.sub.
  [blattXX.answer.pdf]: PDF-Version des Lösungsblattes.
  [blattXX.answer.ps]: PS-Version des Lösungsblattes.
  blattXX.answer.tex:   LaTeX-Version der Lösungsblattes. Im wesentlichen Einschluß von body.tex,
\usepackage[answers,ziele]{tgi_uebungen}.
  [blattXX.pdf]: PDF-Version des Übungsblattes.
  [blattXX.ps]: PS-Version der Übungsblattes.
  blattXX.tex: LaTeX-Version der Übungsblattes. Im wesentlichen Einschluß von body.tex,
\usepackage{tgi_uebungen}.
  body.tex: Enthält den eigentlichen Inhalt des Übungsblattes. Hier werden die Parameter
des Übungsblattes wie \nummerUebungsblatt \nummerErsteFrage \date etc. definiert.
Weiterhin wird /.repos/_VorlesungsParameter.tex eingeschlossen.
Schließlich werden die einzelnen Aufgaben über \input{Aufgabe} eingeschlossen.
Es ist nicht nötig bei den \input's Verzeichnisse mit anzugeben.

/doc -----------------------------------------------------------------
  index.html: Diese Datei.

  /doc/template ----------------------------------------------------

    /doc/template/blattXX -------------------------------------------
    Vorlagenverzeichnis für die Erstellung eines neuen Blattes.
      Makefile:   Makefile zur Erzeugung des jeweiligen Blattes, definiert $(TOP), ansonsten
  include von /Makefile.sub.
      blattXX.answer.tex: LaTeX-Version der Lösungsblattes. Im wesentlichen Einschluß von body.tex,
  \usepackage[answers,ziele]{tgi_uebungen}.
      blattXX.tex:   LaTeX-Version der Übungsblattes. Im wesentlichen Einschluß von body.tex,
  \usepackage{tgi_uebungen}.
      body.tex:   Enthält den eigentlichen Inhalt des Übungsblattes. Hier werden die Parameter
  des Übungsblattes wie \nummerUebungsblatt \nummerErsteFrage \date etc. definiert.
  Weiterhin wird /.repos/_VorlesungsParameter.tex eingeschlossen.
  Schließlich werden die einzelnen Aufgaben über \input{Aufgabe} eingeschlossen.
  Es ist nicht nötig bei den \input's Verzeichnisse mit anzugeben.

    /doc/template/.repos --------------------------------------------
      _VorlesungsParameter.tex: Enthält die allgemeinen Parameter der Vorlesung wie \Professor, \Semester,
  \Vorlesung etc. Es wird in body.tex der einzelnen Blätter eingeschlossen.

      /doc/template/.repos/blattXX ----------------------------------
        BXX_Template.tex: Vorlage zur Erstellung von Aufgaben-Dateien.

/.repos --------------------------------------------------------------
  Die Verzeichnisstruktur des Repositories steht in keinem Zusammenhang zu den tatsächlichen Blättern,
  in jedem Blatt kann _jede_ Aufgabe eingeschlossen werden, unabhängig von ihrer Position im
  Dateisystem.
  Die Verzeichnisse sollen nur als Hinweise dienen, wo eine Aufgabe etwa sinnvoll ist.

  _VorlesungsParameter.tex: Enthält die allgemeinen Parameter der Vorlesung wie \Professor, \Semester,
    \Vorlesung etc. Es wird in body.tex der einzelnen Blätter eingeschlossen.

  /.repos/abbild ---------------------------------------------------
  Verzeichnis für Abbildungen. Alle Abbildungen werden hier gespeichert.
  Eine weitere Unterteilung ist (aufgrund von Beschränkungen in PDFLaTeX) nicht möglich!
    *.fig:   xfig-Dateien als Vorlagen. Aus ihnen werden automatisch .eps und .pdf erzeugt.
    *.eps:   eps-Dateien, teils generiert aus .fig, teils Originale.
    *.pdf:   pdf-Dateien, werden automatisch aus .fig generiert. Bei eps-Dateien, für die
  keine .fig-Datei vorliegt, muß die Konversion von Hand mit
       convert XXX.eps XXX.pdf bzw.
    epstopdf XXX.eps
  durchgeführt werden.
    *.fig.bak:   Sicherungsdateien von xfig
    *.cdr:   CorelDraw-Dateien, meist für Version 7.0.

  /.repos/blattXX --------------------------------------------------
    BXX_*.tex:   Die eigentlichen Aufgaben-Dateien. Ihre Struktur wird unten beschrieben.
    *.s:   SPIM-Assembler-Dateien.
    *.java:   Java-Dateien.
    *.pas:   Pascal-Pseudocode-Dateien.
    /.repos/blattXX/*/ ---------------------------------------------
    Unterverzeichnis für Quellen von Beispielprogrammen u.ä.
```

B Zweiter Anhang: Vorlagen-Dateien

```
%          +------------------------------------------+
```

```
%       | ü            bungsblatt XX                  |
%       +--------------------------------------------+
%
% Stand:     XX.XX.XXXX
% Version:   0
% Autor:     XXXX XXXXXXX
%
% Reine Wrapper-Datei üfr die Generierung desÜ bungsblattes.
%
% +--------------------------------------------------------------------+
% | Article als Basisklasse                                            |
\documentclass{article}
% | Dieses Stylesheet äldt einige allgemeine Pakete und                |
% | stellt Befehle bzw. üAbkrzungen üfr ähufig öbentigte Aufgaben      |
% | bereit. Zu den Optionen siehe dort.                                |
\usepackage[nonums,article,layout]{standard}
% | In diesem Stylesheet werden die spezifischen Anpassungen üfr die   |
Ü% | äbungsbltter üdurchgefhrt und die Umgebungen und Kommandos        |
% | definiert. Zu den Optionen siehe dort.                             |
\usepackage{tgi_uebungen}
% |                                                                    |
% | ßSchlielich noch eigentliche Inhalt ...                            |
\input{body}
% +--------------------------------------------------------------------+

%       +--------------------------------------------+
%       | ö            Lsungsblatt XX                 |
%       +--------------------------------------------+
%
% Stand:     XX.XX.XXXX
% Version:   0
% Autor:     XXXX XXXXXXX
%
% Reine Wrapper-Datei üfr die Generierung des öLsungsblattes.
%
% +--------------------------------------------------------------------+
% | Article als Basisklasse                                            |
\documentclass{article}
% | Dieses Stylesheet äldt einige allgemeine Pakete und                |
% | stellt Befehle bzw. üAbkrzungen üfr ähufig öbentigte Aufgaben      |
% | bereit. Zu den Optionen siehe dort.                                |
\usepackage[nonums,article,layout]{standard}
% | In diesem Stylesheet werden die spezifischen Anpassungen üfr die   |
Ü% | äbungsbltter üdurchgefhrt und die Umgebungen und Kommandos        |
% | definiert. Zu den Optionen siehe dort.                             |
\usepackage[answers,ziele]{tgi_uebungen}
% |                                                                    |
% | ßSchlielich noch eigentliche Inhalt ...                            |
\input{body}
% +--------------------------------------------------------------------+

%       +--------------------------------------------+
%       |        BODYü bungsblatt XX                  |
%       +--------------------------------------------+
%
% Stand:     XX.XX.XXXX
% Version:   0
% Autor:     XXXX XXXXXXX
%
% Hier werden die spezifischen Parameter einesÜ bungsblattes angegeben und
% die Aufgaben eingeschlossen.

% +--------------------------------------------------------------------+
% |               Parameter desÜ bungsblattes                          |
% |                                                                    |
% | Vorlesungsparameter, öknnenü berschrieben werden:                  |
\input{_VorlesungsParameter}
% |                                                                    |
% | Variante Parameter desÜ bungsblattes:                              |
% |                                                                    |
% | \nummerUebungsblatt{NUMMER}      REQUIRED                          |
% |   NUMMER desÜ bungsblattes                                         |
% |   NUMMER:           INTEGER                                        |
\nummerUebungsblatt{8}
% | \date{DATUM}                     OPTIONAL (DEFAULT)                |
% |   DATUM der _Ausgabe_, wenn nicht gesetzt, Datum derÜ bersetzung   |
% |   DATUM:            TT.MM.JJJJ                                     |
\date{22.06.2000}
% | \Abgabetermin{DATUM}             OPTIONAL                          |
% |   DATUM der _Abgabe_                                               |
```

```
% |   DATUM:           TT.MM.JJJJ                                     |
%\Abgabetermin{29.06.2000}
% | \nummerErsteFrage{NUMMER}        REQUIRED
% |   NUMMER des ersten Frage auf diesemÜ bungsblatt                  |
% |   NUMMER:          POS. INTEGER                                   |
\nummerErsteFrage{73}
% | \Ankuendigungen{ANKUENDIGUNG}    OPTIONAL                         |
% |   ANKUENDIGUNG im Titel vor zuLesen etc.                          |
% |   ANKUENDIGUNG:    TeX-Parameter-Content                          |
\Ankuendigungen{Beim Zusammenstellen des SPIM--Tutorials
  (Stand: 02.05.2000) ist uns ein Fehler unterlaufen, so dass z.T.
  sowohl Kapitel mittendrin als auch am ßSchlu komplett fehlen. Bitte
  laden Sie sich die neueste Version (Stand: 27.06.2000) des Tutorials von der
  TGI--Homepage herunter und drucken Sie sich die fehlenden Kapitel aus.}
% | \zuLesen{TEXT}                   OPTIONAL                         |
% |   Leseempfehlungen                                                |
% |   TEXT:            TeX-Parameter-Content                          |
\zuLesen{Tutorium Kap. 13; \textsl{Register Usage and Procedures}}
% | Analog \zuWiederholen und \zuVertiefen                            |
\zuWiederholen{mehrdimensionale Arrays, Linearisierung von Arrays}
\zuVertiefen{PatHen Kap. A.6}
% +-------------------------------------------------------------------+

\begin{document}


% +-------------------------------------------------------------------+
% | Erzeugung der Titelzeile. Hier werden die Parameter verwendet.    |
\maketitle
% +-------------------------------------------------------------------+


% +-------------------------------------------------------------------+
% |                          INTRO                                    |
% | \begin{intro} TeX-Environment-Content \end{intro}                 |
% | öErmglicht das Setzen von älngeren üEinfhrungstexten, die nicht   |
% | in Ankuendigungen passen. Wird etwas kleiner gesetzt.             |
\begin{intro}
  In den folgenden Aufgaben sollen verschiedene hochsprachliche Programme in eine Minimalsprache
  umgewandelt werden, die ßausschlielich die folgenden Konstrukte ßumfat.
\end{intro}
% +-------------------------------------------------------------------+

% +-------------------------------------------------------------------+
% |                          AUFGABEN                                 |
% | Aufgaben werden mittels \input eingebunden. Es ßmu kein Pfad       |
% | angegeben werden, wenn die Datei innerhalb des Repositories       |
% | (.repos) liegt. Es spielt keine Rolle, in welchem Unterverzeichnis|
% | des Repositories sie liegt (insbesondere kein Zusammenhang        |
% | zwischen Blattnummer im Repository und in /aufgaben.              |
% |                                                                   |
% | Ein Teilpfad (z.B. blattXX/XXXX.tex) kann bei Namenskonflikten    |
% | verwendet werden.                                                 |
% |                                                                   |
% | Mit Hilfe folgender Befehle kann der Typ (T|H|P|K) der änchsten   |
% | Aufgabe ägendert werden, ohne die Aufgabendatei zu äverndern:     |
% | \setHausaufgabe                    \setTutoraufgabe               |
% | \setProgrammieraufgabe             \setKlausuraufgabe             |
\input{BXX_Template}
% +-------------------------------------------------------------------+


\end{document}

%       +-------------------------------------------+
%       |        DEFINITION EINER AUFGABE           |
%       +-------------------------------------------+
%
% Stand:      XX.XX.XXXX
% Version:    0
% Autor:      XXXX XXXXXXX
%
% Hier wird eine Aufgabe definiert. Dazu stehen die in tgi_uebungen.sty
% definierten Kommandos und Umgebungen zur üVerfgung.

% +-------------------------------------------------------------------+
% |                         AUFGABE                                   |
% | \begin{aufgabe}{BEZEICHNUNG}{PUNKTE}{TYP}                         |
% |    TeX-Environment-Content                                        |
% |    \ziele{TeX-Parameter-Content}                                  |
```

```
% |     \begin{antwort}                                               |
% |       TeX-Environment-Content                                     |
% |     \end{antwort}                                                 |
% |  \end{aufgabe}                                                    |
% |                                                                   |
% |  Definiert eine Aufgabe mit Titel BEZEICHNUNG, PUNKTE Punkten und |
% |  dem gegebenen TYP.                                               |
% |                                                                   |
% |  TEILAUFGABEN werden per ENUMERATE erzeugt, es wird automatisch   |
% |  alpha-Nummerierung verwendet. Um zwischen den Nummerierungen     |
% |  verwende man \alphaEnum bzw. \romanEnum                          |
% |                                                                   |
% |    BEZEICHNUNG:       TeX-Parameter-Content.                      |
% |    PUNKTE:            TeX-Parameter-Content, meist X+X+X üfr Teil- |
% |                       aufgaben.                                   |
% |    TYP:               H|P|T|K üfr Haus-, Programmier-, Tutor- bzw. |
% |                       Klausuraufgabe.                             |
\begin{aufgabe}{Minimierung einer Schaltfunktion}{2+4+4}{H}
  Ein \strong{Karnaugh-Diagramm} einer Booleschen Funktion $f: B^n
  \abbm B$ mit [...]

\ziele{}
\begin{antwort}
  [...]
\end{antwort}
\end{aufgabe}
% +-------------------------------------------------------------------+

%        +-------------------------------------------+
%        |      Vorlesungsweite Parameter            |
%        +-------------------------------------------+
%
% Stand:     18.02.2001
% Version:   1
% Autor:     Tim Furche
%
% Hier werden vorlesungsweite Parameter konfiguriert. Zu den Parametern und ihrer Verwendung s.u.
%
% +-------------------------------------------------------------------+
% |  \Professor{VERANSTALTER}                  REQUIRED               |
% |    VERANSTALTER der Vorlesung, wird in der Titelzeile verwendet.  |
% |    VERANSTALTER:    TeX-Parameter-Content                         |
\Professor{Prof.\ Dr.\ Claudia Linnhoff-Popien}
% |  \Semester[KURZBEZEICHNUNG]{BEZEICHNUNG}  REQUIRED               |
% |    Semester der Vorlesung.                                        |
% |    BEZEICHNUNG wird in der Titelzeile, die KURZBEZEICHNUNG in der |
% |    Kopfzeile verwendet.                                           |
% |    BEZEICHNUNG:      TeX-Parameter-Content                        |
% |    KURZBEZEICHNUNG: TeX-Parameter-Content, Format: (SS|WS) JAHR4  |
\Semester[SS 2000]{Sommersemester 2000}
% |  \Vorlesung{BEZEICHNUNG}                   REQUIRED               |
% |    BEZEICHNUNG der Vorlesung, in der Titelzeile/Kopfzeile verwendet|
% |    BEZEICHNUNG:      TeX-Parameter-Content                        |
\Vorlesung{Technische Grundlagen der Informatik}
% |  \Universitaet{BEZEICHNUNG}                REQUIRED               |
% |    BEZEICHNUNG der Universitaet, in der Titelzeile verwendet.     |
% |    BEZEICHNUNG:      TeX-Parameter-Content                        |
\Universitaet{Ludwig-Maximilians-äUniversitt üMnchen}
% |  \Institut{BEZEICHNUNG}                    REQUIRED               |
% |    BEZEICHNUNG des Instituts, in der Titelzeile verwendet.        |
% |    BEZEICHNUNG:      TeX-Parameter-Content                        |
\Institut{Institut üfr Informatik}
% +-------------------------------------------------------------------+

% Hier öknnen auch alle in BODY definierten Parameter desÜ bungsblattes angegeben werden, bsp.
% mit \Ankuendigung eine wiederkehrende üAnkndigung oder \zuLesen ein stets zu lesendes Werk.
%
% Diese dienen dann als Standardwerte und öknnen in denÜ äbungsbltternü berschrieben werden!
```

C Dritter Anhang: Übungsmakros in LaTeX

```
%        +-------------------------------------------+
%        |     STYLESHEET ÜFRÜ BUNGSAUFGABEN         |
%        +-------------------------------------------+
%
% Stand:     17.02.2001
% Version:   21
% Autor:     Tim Furche
```

```
%
% In diesem Stylesheet werden die spezifischen Anpassungen üfr
% dieÜ äbungsbltter üdurchgefhrt und die Umgebungen und Kommandos
% definiert.
%
% +------------------------------------------------------------------+
% |                        Fontauswahl                               |
% |                                                                  |
% | Aktuell: auskommentieren => Standardfonts                        |
\RequirePackage{euler}
\RequirePackage{charter}
\RequirePackage{courier}
% |                                                                  |
% | Alternativ: Palatino mit eigenem mathemat. Font                  |
%\RequirePackage{palatino}
%\RequirePackage{mathpple}
%\RequirePackage{avant}
%\RequirePackage{courier}
%
% +------------------------------------------------------------------+


% +------------------------------------------------------------------+
% | Anpassungen multicol                                             |
\renewcommand{\columnseprule}{0pt}
\setlength{\columnsep}{1cm}
% +------------------------------------------------------------------+


% +------------------------------------------------------------------+
% |                       COMMENT-Paket                              |
% | Notwendig üfr die Auskommentierung der Aufgaben üfr              |
% | Studentenversion:
\RequirePackage{comment}
% +------------------------------------------------------------------+


% +------------------------------------------------------------------+
% |                      LISTINGS-Paket                              |
% | öErmglicht den Pretty-Print der Programmcodes                    |
% |                                                                  |
% | Fix, um einen Bug im LISTINGS-Paket zu behandeln:                |
\newcounter{chapter}
% |                                                                  |
\usepackage{listings}
\lstloadlanguages{Modula-2, C, make,csh}
% |                                                                  |
% | Definition der neuen "Programmiersprache" MIPS                   |
% |       hoffentlich ävollstndig ...
\lstdefinelanguage{MIPS}%
  {keywords={add,move,div,divu,mult,multu,mul,mulo,mulou,%
      addu,addiu,subu,addi,sub,%
      abs,neg,negu,rem,remu,%
      and,andi,nor,or,ori,xor,xori,not,rol,ror,%
      sll,sllv,srl,srlv,sra,srav,%
      seq,sne,sge,sgeu,sgt,sgtu,sle,sleu,slt,stlu,%
      sltu,stli,sltui,%
      b,j,beq,beqz,bne,bnez,bge,bgeu,bgez,bgt,bgtu,%
      bgtz,ble,bleu,blez,blt,bltu,bltz,%
      jr,jal,jalr,bgezal,bltzal,la,lwc,l.d,l.s,%
      swc,s.d,s.s,mfc,mfcl.d,mtc,mov.d,mov.s,%
      add.d,add.s,sub.d,sub.s,mul.d,mul.s,div.d,%
      div.s,abs.d,abs.s,neg.d,neg.s,bct,bcf,%
      c.eq.d,c.eq.s,c.le.d,c.le.s,c.lt.d,c.lt.s,%
      rfe,break,%
      lw,li,lb,lbu,lh,lhu,ld,ulw,ulh,ulhu,lwr,lwl,lui,%
      sw,sb,sh,sd,swl,swr,ush,usw,%
      mfhi,mflo,mthi,mtlo,bclt%
      syscall,%
      .word,.data,.text,.ascii,.asciiz,.half,.double,.space,%
      DIFF, store, load, add, push, pop%
      },%
   sensitive,%
   commentline=\#,%
   stringizer=[b]"%
  }[keywords,comments,strings]%
% |                                                                  |
% | Standard üfr die Ausgabe der Listings:                           |
\lstset{basicstyle=\small\ttfamily, keywordstyle=\bfseries,
  commentstyle=\slshape, stringstyle=\itshape,
```

```
  labelstyle=\tiny\ttfamily, labelstep=2, labelsep=5pt, extendedchars=true}
% | Listing-INCLUDEs                                                     |
\newcommand{\includeMODULA}[1]{\lstset{language=Modula-2}\lstinputlisting{#1}}
\newcommand{\includeMIPS}[1]{\lstset{language=MIPS}\lstinputlisting{#1}}
\newcommand{\includeC}[1]{\lstset{language=C}\lstinputlisting{#1}}
\newcommand{\includeJava}[1]{\lstset{language=Java}\lstinputlisting{#1}}
\newcommand{\includeMake}[1]{\lstset{language=make}\lstinputlisting{#1}}
\newcommand{\includeShell}[1]{\lstset{language=csh}\lstinputlisting{#1}}
% +--------------------------------------------------------------------+


% +--------------------------------------------------------------------+
% |                 Interne Kommandos und IFs                          |
% | Diese Kommandos werden ßausschlieslich intern verwendet.           |
\newif\ifanswers                         % Sollen Antworten ausgegeben werden?
\answersfalse                            % Standard: NEIN
\newif\ifziele                           % Sollen Ziele ausgeben werden?
\zielefalse                              % Standard: NEIN
\newif\ifpoints                          % Sollen Punkte ausgegeben werden?
\pointstrue                              % Standard: JA
\newif\ifnummerNotSet                    % Ist keine Aufgabennummer angegeben?
\nummerNotSettrue                        % Standard: JA
\newif\iflwvSet                          % Ist zuLesen, zuWiederholen oder zuVertiefen gesetzt?
\lwvSetfalse                             % Standard: NEIN
% | Makro zum Test, ob ein Argument leer ist:                          |
% | \@ifemptyarg{#1} {Argument ist leer}{Argument ist nicht leer}      |
\providecommand\@ifemptyarg[1]{%
  \ifx\@empty#1\@empty
  \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
% +--------------------------------------------------------------------+


% +--------------------------------------------------------------------+
% |                 Nummerierung                                       |
% | Teilaufgaben werden per ENUMERATE erzeugt. Dazu wird enumi auf     |
% | Buchstaben-Nummerierung geschaltet. Die beiden folgenden Befehle   |
% | öermglichen ein einfaches Umschalten.                              |
\newcommand{\alphaEnum}{\renewcommand{\theenumi}{\alph{enumi}}%
  \renewcommand{\theenumii}{\roman{enumii}}}
\newcommand{\romanEnum}{\renewcommand{\theenumi}{\roman{enumi}}}
% +--------------------------------------------------------------------+


% +--------------------------------------------------------------------+
% |                      Paket-Optionen                                |
% | answers:      Antworten werden ausgegeben                          |
\DeclareOption{answers}{\answerstrue}
% | ziele:        Ziele werden ausgegeben                              |
\DeclareOption{ziele}{\zieletrue}
% | nopoints:     Punkte werden _nicht_ ausgegeben                     |
\DeclareOption{nopoints}{\pointsfalse}
\ProcessOptions
% +--------------------------------------------------------------------+


% +--------------------------------------------------------------------+
% | Implementation der answers-Option:                                 |
% | Ist \answersfalse, so wird die Umgebung antwort als Kommentar      |
% | definiert, ansonsten entsprechend ausgegeben.                      |
\ifanswers
  \newenvironment{antwort}%
  {%
    \par\medskip\alphaEnum\answerfont\makebox[1.5cm]{\textbf{Antwort:}\hfill}\nopagebreak%
  }%
  {%
  }
\else
  \excludecomment{antwort}
\fi
% +--------------------------------------------------------------------+


% +--------------------------------------------------------------------+
% |                 Redefinition von MAKETITLE                         |
% | üUnterstttzung der folgenden neuen Kommandos:                      |
% | \Professor          \Universitaet              \Institut           |
% | \Semester           \Vorlesung                 \date (aus Std.)    |
% | \nummerUebungsblatt \nummerErsteFrage          \Abgabetermin       |
% | \Ankuendigungen     \zuLesen                   \zuWiederholen      |
% | \zuVertiefen                                                       |
```

```
\newcommand{\@Professor}{}
\newcommand{\@Universitaet}{}
\newcommand{\@Institut}{}
\newcommand{\@Semester}{}
\newcommand{\@ShortSemester}{}
\newcommand{\@Vorlesung}{}
\newcommand{\@nummerUebungsblatt}{}
\newcommand{\@Abgabetermin}{}
\newcommand{\@Besprechung}{}
\newcommand{\@Ankuendigungen}{}
\newcommand{\@zuLesen}{}
\newcommand{\@zuWiederholen}{}
\newcommand{\@zuVertiefen}{}
%
\newcommand{\Professor}[1]{\renewcommand{\@Professor}{#1}}
\newcommand{\Universitaet}[1]{\renewcommand{\@Universitaet}{#1}}
\newcommand{\Institut}[1]{\renewcommand{\@Institut}{#1}}
\newcommand{\Semester}[2][]{\renewcommand{\@Semester}{#2}
  \@ifemptyarg{#1}{\renewcommand{\@ShortSemester}{#2}}{\renewcommand{\@ShortSemester}{#1}}}
\newcommand{\Vorlesung}[1]{\renewcommand{\@Vorlesung}{#1}}
% | MARKE A                                                              |
\newcommand{\nummerUebungsblatt}[1]%
  {\renewcommand{\@nummerUebungsblatt}{#1}\nummerNotSetfalse}
\newcommand{\Abgabetermin}[1]%
  {\@ifemptyarg{#1}%
    {}
    {\renewcommand{\@Abgabetermin}{%
        {\large\textbf{Abgabetermin:} #1}}
    }
}
\newcommand{\Besprechung}[1]%
  {\@ifemptyarg{#1}%
    {}
    {\renewcommand{\@Besprechung}{%
        {\large\textbf{Besprechung:} #1}}
    }
}
\newcommand{\Ankuendigungen}[1]%
  {\@ifemptyarg{#1}%
    {}%
    { \renewcommand{\@Ankuendigungen}{%
      \medskip \noindent{\makebox[2cm]{\textbf{Achtung:}\hfill}} #1
      \medskip}
    }
  }
\newcommand{\zuLesen}[1]%
{\@ifemptyarg{#1}{}{\lwvSettrue\renewcommand{\@zuLesen}%
{\item[{\makebox[2cm]{\sffamily Lesen:\hfill}}] #1}}}
\newcommand{\zuWiederholen}[1]%
{\@ifemptyarg{#1}{}{\lwvSettrue\renewcommand{\@zuWiederholen}%
{\item[{\makebox[2cm]{\sffamily Wiederholen:\hfill}}] #1}}}
\newcommand{\zuVertiefen}[1]%
{\@ifemptyarg{#1}{}{\lwvSettrue\renewcommand{\@zuVertiefen}%
{\item[{\makebox[2cm]{\sffamily Vertiefen:\hfill}}] #1}}}
%
%
% |                                                                      |
\renewcommand*\maketitle{
    \@maketitle}
%
% | Das eigentliche Layout des Titels                                    |
\def\@maketitle{%
% |     Test, obÜ bungsnummer gesetzt ist, vgl. MARKE A                  |
  \thispagestyle{empty}%
  \ifnummerNotSet
    \PackageError{tgi_uebungen}{%
      Sie haben keine Nummer fuer das Uebungsblatt angegeben.
      }{%
      Geben Sie mit \\nummerUebungsblatt eine Nummer an.
      }
  \fi
% | Informationszeile                                                    |
  {
    \large
    {\scshape
    \begin{tabular}[t]{c}
      \@Universitaet \\
      \@Institut \\
      \@Professor
    \end{tabular}}
```

```
    \hfill
    \begin{tabular}[t]{c}
      \@Semester \\Ü
      bungsblatt \@nummerUebungsblatt \\
      \@date
    \end{tabular}
  }
  \par\bigskip\bigskip
% | Titelzeile                                                    |
  {\Large
  \begin{center}
    {\LARGE\textbf{\@Vorlesung}\par}\smallskip
    \@Abgabetermin\@Besprechung
  \end{center}}
  %
  \par\bigskip
  {
    \small
% | üAnkndigungen                                                 |
    \@Ankuendigungen
  %
    \par\bigskip
% | Lesen/Wiederholen/Vertiefen:                                  |
    \iflwvSet
      \begin{description}
        \@zuLesen
        \@zuWiederholen
        \@zuVertiefen
      \end{description}
    \fi
    \bigskip
  }
}
% +---------------------------------------------------------------------+


% +---------------------------------------------------------------------+
% |                         Intro                                       |
\newcommand{\Introfont}{\small}
\newenvironment{intro}{\par\Introfont}{\par\normalsize\normalfont}
% +---------------------------------------------------------------------+


% +---------------------------------------------------------------------+
% |                    Anpassung Kopfzeilen                             |
\pagestyle{fancy}
\renewcommand{\sectionmark}[1]{}
\renewcommand{\headrulewidth}{0.5pt}
\renewcommand{\footrulewidth}{0pt}
\fancyhf{}
\fancyhead[LE,RO]{{\sffamily \thepage}}
\fancyhead[LO]{{\sffamily\@Vorlesung\,--\,\@ShortSemesterÜ, bungsblatt
    \@nummerUebungsblatt}}
\fancyhead[RE]{{\sffamily\@Vorlesung, \@ShortSemesterÜ, bungsblatt
    \@nummerUebungsblatt}}
% +---------------------------------------------------------------------+


% +---------------------------------------------------------------------+
% |                    AUFGABEN-Umgebung                                |
\newif\ifexists@ziele
\newcounter{aufgaben@nummer}
\newcommand{\nummerErsteFrage}[1]{\setcounter{aufgaben@nummer}{#1}\addtocounter{aufgaben@nummer}{-1}}
\newcommand{\aufgabenfont}{\Large\bfseries}
\newcommand{\aufgabentitelfont}{\Large}
\newcommand{\answerfont}{\sffamily\small}
\newcommand{\@ziele}{}
\newcommand{\ziele}[1]{\renewcommand{\@ziele}{#1}\exists@zieletrue}
% |                                                                     |
\newif\ifChange@Type
\newcommand{\@examType}{}
\newcommand{\setHausaufgabe}{\Change@Typetrue\renewcommand{\@examType}{H}}
\newcommand{\setTutoraufgabe}{\Change@Typetrue\renewcommand{\@examType}{T}}
\newcommand{\setKlausuraufgabe}{\Change@Typetrue\renewcommand{\@examType}{K}}
\newcommand{\setProgrammieraufgabe}{\Change@Typetrue\renewcommand{\@examType}{P}}
\newenvironment{aufgabe}[3]%
{%
  \stepcounter{aufgaben@nummer}\exists@zielefalse
  \renewcommand{\@ziele}{}\alphaEnum
  \par\bigskip\medskip\noindent
```

```
  {\aufgabenfont Aufgabe \theaufgaben@nummer:\ }
  {\aufgabenfont
    \ifChange@Type
      (\@examType)
    \else
      (#3)
    \fi}
  {\aufgabentitelfont #1}
  \hfill
  \ifpoints
    \@ifemptyarg{#2}{}{(#2 Pkt.)}
  \fi
  \par\nopagebreak\medskip
}%
{%
  \ifziele
    \ifexists@ziele
      \par\smallskip
      {
        \answerfont
        \textbf{Ziele:\ }
        \@ziele
      }
    \fi
  \fi\exists@zielefalse
  \smallskip\Change@Typefalse
}
% +--------------------------------------------------------------------+
```

D Vierter Anhang: Beispiele

E Fünfter Anhang: Makefiles

```
#       +-----------------------------------------+
#       |      MASTER-MAKEFILE UeBUNGSAUFGABEN     |
#       +-----------------------------------------+
#
# Stand:     17.02.2001
# Version:   9
# Autor:     Tim Furche
#
# Dieses Makefile dient dazu, in allen *blatt*-Unterverzeichnissen make mit dem
# jeweiligen Target aufzurufen.
#
# Die definierten Targets sind:
# make all / make:       Erzeugt alle Uebungsblaetter und Loesungsblaetter.
# make blatt:            Erzeugt alle Uebungsblaetter
# make answer:           Erzeugt alle Loesungsblaetter
# make blatt(.ps|.pdf):  Erzeugt nur die PS|PDF-Version der Uebungsblaetter
# make answer(.ps|.pdf): Erzeugt nur die PS|PDF-Version der Loesungsblaetter
# make clean:            Loescht alle temporaeren Dateien
# make distclean:        Loescht alle regenerierbaren Dateien (auch die
#                        Uebungs- und Loesungsblaetter)
# make cleanemacs:       Loescht Emacs-Backup-Files und auto-Verzeichnisse in allen
#                        Unterverzeichnissen

SHELL   = /bin/sh
ROOTDIR = $(shell pwd)
WWWDIR  = $(ROOTDIR)/../www-internal/tgi
WWWBLATTDIR  = $(WWWDIR)/Blaetter
WWWANSWERDIR = $(WWWDIR)/Loesungen

.PHONY: all answer clean distclean cleanemacs blatt.ps blatt.pdf answer.ps answer.pdf

all:
        @for DATEI in *blatt* ; do \
                if [ -d $$DATEI ]; then \
                        (       echo "================================================"; \
                                cd $$DATEI && ( make ) ; \
                                echo "" \
                        ) ; \
                fi ; \
        done

blatt:
        @for DATEI in *blatt* ; do \
                if [ -d $$DATEI ]; then \
                        (       echo "================================================"; \
                                cd $$DATEI && ( make blatt ) ; \
                                echo "" \
                        ) ; \
                fi ; \
        done

blatt.ps:
        @for DATEI in *blatt* ; do \
                if [ -d $$DATEI ]; then \
                        (       echo "================================================"; \
                                cd $$DATEI && ( make blatt.ps ) ; \
                                echo "" \
                        ) ; \
```

```
                fi ; \
        done

blatt.pdf:
        @for DATEI in *blatt* ; do \
                if [ -d $$DATEI ]; then \
                        (       echo "=================================================="; \
                                cd $$DATEI && ( make blatt.pdf ) ; \
                                echo "" \
                        ) ; \
                fi ; \
        done



answer:
        @for DATEI in *blatt* ; do \
                if [ -d $$DATEI ]; then \
                        (       echo "=================================================="; \
                                cd $$DATEI && ( make answer ) ; \
                                echo "" \
                        ) ; \
                fi ; \
        done

answer.ps:
        @for DATEI in *blatt* ; do \
                if [ -d $$DATEI ]; then \
                        (       echo "=================================================="; \
                                cd $$DATEI && ( make answer.ps ) ; \
                                echo "" \
                        ) ; \
                fi ; \
        done


answer.pdf:
        @for DATEI in *blatt* ; do \
                if [ -d $$DATEI ]; then \
                        (       echo "=================================================="; \
                                cd $$DATEI && ( make answer.pdf ) ; \
                                echo "" \
                        ) ; \
                fi ; \
        done



clean:
        @for DATEI in *blatt* ; do \
                if [ -d $$DATEI ]; then \
                        (       echo "=================================================="; \
                                cd $$DATEI && make clean ; \
                                echo "$$DATEI␣is␣now␣clean."; \
                                echo "" \
                        ) ; \
                fi ; \
        done

distclean:
        @for DATEI in *blatt* ; do \
                if [ -d $$DATEI ]; then \
                        (       echo "=================================================="; \
                                cd $$DATEI && make distclean ; \
                                echo "$$DATEI␣is␣now␣really␣clean."; \
                                echo "" \
                        ) ; \
                fi ; \
        done

emacsclean:
        @find . -name "*~" -exec rm "{}" "-rf" ";"
        @find . -name "auto" -exec rm "{}" "-rf" ";"
        @echo "Alle␣~-Dateien␣und␣auto-Verzeichnisse␣geloescht."

webcopy:
        @for DATEI in *blatt*; do \
                if [ -d $$DATEI ]; then \
                        (       echo "=================================================="; \
                                WEBCOPY_TARGET=`echo $$DATEI | gawk '{ print toupper(substr($$1,1,1)) substr($$1,2) }'` ;\
                                cd $$DATEI && \
                                if [ -f $$DATEI.ps ]; then \
                                        cp $$DATEI.ps $(WWWBLATTDIR)/$$WEBCOPY_TARGET && \
                                        echo "$$DATEI.ps␣has␣been␣copied␣to␣web."; \
                                else \
                                        echo "$$DATEI.ps␣was␣not␣available."; \
                                fi ; \
                                if [ -f $$DATEI.pdf ]; then \
                                        cp $$DATEI.pdf $(WWWBLATTDIR)/$$WEBCOPY_TARGET && \
                                        echo "$$DATEI.pdf␣has␣been␣copied␣to␣web."; \
                                else \
                                        echo "$$DATEI.pdf␣was␣not␣available."; \
                                fi ; \
                                if [ -f $$DATEI.answer.ps ]; then \
                                        cp $$DATEI.answer.ps $(WWWANSWERDIR)/$$WEBCOPY_TARGET && \
                                        echo "$$DATEI.answer.ps␣has␣been␣copied␣to␣web."; \
                                else \
                                        echo "$$DATEI.answer.ps␣was␣not␣available."; \
                                fi ; \
                                if [ -f $$DATEI.answer.pdf ]; then \
                                        cp $$DATEI.answer.pdf $(WWWANSWERDIR)/$$WEBCOPY_TARGET && \
                                        echo "$$DATEI.answer.pdf␣has␣been␣copied␣to␣web."; \
                                else \
                                        echo "$$DATEI.answer.pdf␣was␣not␣available."; \
                                fi ; \
                                echo "" \
                        ) ; \
```

```
                fi ; \
        done


#       +---------------------------------------------+
#       |        SUB-MAKEFILEÜ BUNGSAUFGABEN          |
#       +---------------------------------------------+
#
# Stand:     17.02.2001
# Version:   1
# Autor:     Tim Furche
#
# Dieses Makefile wird durch die Makefiles in den Unterverzeichnissen verwendet.
# Es äenthlt die Regeln.
#
# In den jeweiligen Unterverzeichnissen wird im Makefile nur noch
# TOP definiert. Dies zeigtü blicherweise auf den Namen des Blattes.
# TOP    = blatt01
#
# Neben den Targets aus dem MASTER-MAKEFILE gibt es noch die folgenden weiteren:
# make $(TOP).dvi:              Erzeugt DVI desÜ bungsblattes in $(TMPDIR)
# make $(TOP).answer.dvi:       Erzeugt DVI des öLsungsblattes in $(TMPDIR)
# make showconfig:              Zeigt die Definitionen des Makefiles an
include ../Makefile.sub.DEF

all:    blatt answer
        @echo "␣␣␣␣␣␣␣␣␣␣+------------------------------------------------+␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣|␣$(TOP)␣erfolgreich␣erzeugt.␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣|␣␣␣$(TOP).ps␣␣␣␣␣␣␣␣␣ok␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣|␣␣␣$(TOP).pdf␣␣␣␣␣␣␣␣ok␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣|␣␣␣$(TOP).answer.ps␣␣ok␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣|␣␣␣$(TOP).answer.pdf␣ok␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣+------------------------------------------------+␣␣␣␣␣␣␣␣"

blatt:  $(TOP).ps $(TOP).pdf
        @echo "␣␣␣␣␣␣␣␣␣␣+------------------------------------------------+␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣|␣$(TOP)␣erfolgreich␣erzeugt.␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣|␣␣␣$(TOP).ps␣␣␣␣␣␣␣␣␣ok␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣|␣␣␣$(TOP).pdf␣␣␣␣␣␣␣␣ok␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣+------------------------------------------------+␣␣␣␣␣␣␣␣"

answer: $(TOP).answer.ps $(TOP).answer.pdf
        @echo "␣␣␣␣␣␣␣␣␣␣+------------------------------------------------+␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣|␣$(TOP)␣erfolgreich␣erzeugt.␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣|␣␣␣$(TOP).answer.ps␣␣ok␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣|␣␣␣$(TOP).answer.pdf␣ok␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣"
        @echo "␣␣␣␣␣␣␣␣␣␣+------------------------------------------------+␣␣␣␣␣␣␣␣"

blatt.ps: $(TOP).ps

blatt.pdf: $(TOP).pdf

answer.ps: $(TOP).answer.ps

answer.pdf: $(TOP).answer.pdf

$(TOP).answer.ps: $(TOP).answer.dvi
        @test -d $(TMPDIR) || mkdir $(TMPDIR)
        cd ${TMPDIR}; dvips -o ${ROOTDIR}/$@ $<

$(TOP).answer.pdf: ${PDFFILES} ${TEXFILES}  $(TOP).answer.tex body.tex
        @test -d $(TMPDIR) || mkdir $(TMPDIR)
        -@ln -s $(PICDIR)/*.pdf $(TMPDIR) > /dev/null 2>&1
        cd ${TMPDIR};${PDFLATEX} ${ROOTDIR}/$(TOP).answer.tex
# Solange es noch Label-Changed-Warnings gibt, ßla LaTeX nochmal üdrberlaufen.
        @while ( fgrep -sx "LaTeX␣Warning:␣Label(s)␣may␣have␣changed.␣Rerun␣to␣get␣cross-references␣right." $(TMPDIR)/$(TOP).answer.log ); do   \
          cd $(TMPDIR); $(PDFLATEX) $(ROOTDIR)/$(TOP).answer.tex;\
        done
        @mv -f $(TMPDIR)/$(TOP).answer.pdf $(ROOTDIR)

$(TOP).answer.dvi: ${EPSFILES} ${TEXFILES} $(TOP).answer.tex body.tex
        @test -d $(TMPDIR) || mkdir $(TMPDIR)
        cd ${TMPDIR};${LATEX} ${ROOTDIR}/$(TOP).answer.tex
# Solange es noch Label-Changed-Warnings gibt, ßla LaTeX nochmal üdrberlaufen.
        @while ( fgrep -sx "LaTeX␣Warning:␣Label(s)␣may␣have␣changed.␣Rerun␣to␣get␣cross-references␣right." $(TMPDIR)/$(TOP).answer.log ); do   \
          cd $(TMPDIR); $(LATEX) $(ROOTDIR)/$(TOP).answer.tex;\
        done

$(TOP).ps: $(TOP).dvi
        @test -d $(TMPDIR) || mkdir $(TMPDIR)
        cd ${TMPDIR}; dvips -o ${ROOTDIR}/$@ $<

# Dependencie $(TOP).dvi nur zur Regeneration nach CLEAN u.a.
$(TOP).pdf: ${PDFFILES} ${TEXFILES}  $(TOP).tex body.tex
        @test -d $(TMPDIR) || mkdir $(TMPDIR)
        -@ln -s $(PICDIR)/*.pdf $(TMPDIR) > /dev/null 2>&1
        cd ${TMPDIR}; ${PDFLATEX} ${ROOTDIR}/$(TOP).tex
# Solange es noch Label-Changed-Warnings gibt, ßla LaTeX nochmal üdrberlaufen.
        @while ( fgrep -sx "LaTeX␣Warning:␣Label(s)␣may␣have␣changed.␣Rerun␣to␣get␣cross-references␣right." $(TMPDIR)/$(TOP).log ); do  \
          cd $(TMPDIR); $(PDFLATEX) $(ROOTDIR)/$(TOP).tex;\
        done
        @mv -f $(TMPDIR)/$(TOP).pdf $(ROOTDIR)

$(TOP).dvi: ${EPSFILES} ${TEXFILES} $(TOP).tex body.tex
        @test -d $(TMPDIR) || mkdir $(TMPDIR)
        cd ${TMPDIR};${LATEX} ${ROOTDIR}/$(TOP).tex
# Solange es noch Label-Changed-Warnings gibt, ßla LaTeX nochmal üdrberlaufen.
        @while ( fgrep -sx "LaTeX␣Warning:␣Label(s)␣may␣have␣changed.␣Rerun␣to␣get␣cross-references␣right." $(TMPDIR)/$(TOP).log ); do  \
          cd $(TMPDIR); $(LATEX) $(ROOTDIR)/$(TOP).tex;\
        done


${EPSFILES}: %.eps: %.fig $(FIGFILES)
        @fig2dev -L eps $< $(basename $<).eps


${PDFFILES}: %.pdf: %.eps  $(EPSFILES)
```

```
	@test -f $(basename $<).eps && ( test -f $(basename $<).pdf || \
	  epstopdf $(basename $<).eps --outfile=$(basename $<).pdf )

.PHONY: clean distclean showconfig all blatt.ps blatt.pdf


showconfig:
	@echo "TEXINPUTS:␣${TEXINPUTS}"
	@echo "␣"
	@echo "SRCDIRS:␣${SRCDIRS}"
	@echo "␣"
	@echo "Xfig␣files␣in␣SRCDIRS:"
	@for name in $(FIGFILES) ""; do echo "␣␣"$${name}; done
	@echo "PDF␣files␣in␣SRCDIRS:"
	@for name in $(PDFFILES) ""; do echo "␣␣"$${name}; done
	@echo "EPS␣files␣in␣SRCDIRS:"
	@for name in $(EPSFILES) ""; do echo "␣␣"$${name}; done
	@echo "LaTeX␣files␣in␣SRCDIRS:"
	@for name in $(TEXFILES) ""; do echo "␣␣"$${name}; done

clean:
	@test -d $(TMPDIR) || mkdir $(TMPDIR)
	@${RM} ${TMPDIR}/*
	@echo "All␣temporary␣files␣have␣been␣deleted."

distclean: clean
	@$(RM) $(ROOTDIR)/*.pdf
	@$(RM) $(ROOTDIR)/*.ps
	@echo "All␣regenerable␣files␣have␣been␣deleted."


#	+----------------------------------------------+
#	|      DEFINITIONS-MAKEFILEÜ BUNGSAUFGABEN      |
#	+----------------------------------------------+
#
# Stand:	17.02.2001
# Version:	1
# Autor:	Tim Furche
#
# In diesem Makefile werden die wesentlichen Definitionen üfr
# Makefile.sub festgelegt.
#
# +------------------------------------------------------------------+
# |                         Verzeichnisse                            |
# | - ROOTDIR zeigt auf das aktuelle Verzeichnis. Man beachte, ßda   |
# |   dies das jeweilige Blatt-Verzeichnis ist!                      |
ROOTDIR = $(shell pwd)
# | - SRCDIR zeigt auf das Repository mit den Aufgaben               |
SRCDIR  = ${ROOTDIR}/../.repos
SRCDIRS = $(sort $(dir $(wildcard ${SRCDIR}/*/)))
# | - STYLEDIR zeigt auf das Verzeichnis mit den TeX-Makros          |
STYLEDIR= $(ROOTDIR)/../.makros
# | - PICDIR zeigt auf das Verzeichnis im Repository, das die Bilder |
# | ä enthlt.                                                        |
PICDIR  = $(SRCDIR)/abbild
# | - TMPDIR zeigt auf das Verzeichnis üfr ätemporre Dateien         |
TMPDIR  = ${ROOTDIR}/../.tmp
# +------------------------------------------------------------------+


# +------------------------------------------------------------------+
# |                             Kommandos                            |
SHELL   = /bin/sh
RM      = rm -f
GREP    = grep
LATEX   = latex
PDFLATEX= pdflatex
# +------------------------------------------------------------------+

# +------------------------------------------------------------------+
# |                  Erweiterung der Suchpfade                       |
vpath %.fig ${SRCDIRS}
vpath %.tex ${SRCDIRS}
vpath %.eps ${SRCDIRS} ${TMPDIR}
vpath %.dvi ${TMPDIR}
# +------------------------------------------------------------------+

# +------------------------------------------------------------------+
# |                  Expansion der Dateinamen                        |
FIGFILES=$(notdir $(foreach x,${SRCDIRS},$(wildcard $x/*.fig)))
TEXFILES=$(notdir $(foreach x,${SRCDIRS},$(wildcard $x/*.tex)))
EPSFILES=$(patsubst %.fig,%.eps,${FIGFILES})
PDFFILES=$(patsubst %.fig,%.pdf,${FIGFILES})
#EPSFILES=${EPSFILES_FROM_FIG} ${EPSFILES_FROM_EPS}
#EPSFILES_FROM_EPS=$(notdir $(foreach x,${SRCDIRS},$(wildcard $x/*.eps)))
#EPSFILES_FROM_FIG=$(patsubst %.fig,%.eps,${FIGFILES})
# +------------------------------------------------------------------+

# +------------------------------------------------------------------+
# |                Anpassung des Environments                        |
TEXINPUTS:=$(STYLEDIR)//:${SRCDIR}//:$(ROOTDIR)//:${TEXINPUTS}
export TEXINPUTS
# Nur sinnvoll, wenn spezielle Versionen der verwendeten Pro-
# gramme verwendet werden sollen.
#PATH   :=/usr/local/dist/DIR/tetex-0.9/bin:${PATH}
# +------------------------------------------------------------------+
SHELL   = /bin/sh


#	+----------------------------------------------+
#	|          MAKEFILE ÜTGI-BUNGSAUFGABEN          |
#	+----------------------------------------------+
#
# Stand:	17.02.2001
# Version:	1
# Autor:	Tim Furche
#
```

```
# In diesem Makefile wird nur noch der Name des Blattes festgelegt.
#
TOP     = blattXX
include ../Makefile.sub
```

Literatur

[AEG+ 91] AGUIRRE, G., M. ERRECALDE, R. GUERRERO, C. KAVKA, G. LEGUIZAMON, M. PRINTISTA und R. GALLARD: *Experiencing MINIX as a Didactial Aid for Operating Systems Courses*. Technischer Bericht, Operating Systems Review, 25:32-39, Juli 1991.

[Ben 90] BEN-ARI, M.: *Principles of Concurrent and Distributed Programming*. Prentice Hall, London, 1 Auflage, März 1990.

[BGSW 91] BORGHOFF, U., T. GASTEIGER, A. SCHMALZ und P. WEIGELE: *MI – Eine Maschine für die Informatikausbildung*. Technischer Bericht, Institut für Informatik, Technische Universität München, 1991.

[ByCa 99] BYNUM, B. und T. CAMP: *An Introduction to BACI*. Technischer Bericht, Colorado School of Mines, September 1999
`http://www.mines.edu/fs\_home/tcamp/baci/`.

[ChJo 96] CHERNICH, R. und D. JONES: *RCOS – Ron Chernich's Operating System*. Technischer Bericht, Central Queensland University, Juni 1996
`http://cq-pan.cqu.edu.au/david-jones/Projects/rcos/`.

[Chla 98] CHLAP, C.: *XINU Homepage*. Technischer Bericht, University of Canberra, Oktober 1998
`http://willow.canberra.edu.au/~chrisc/xinu.html`.

[Come 83] COMER, D.: *Operating System Design: the Xinu Approach*. Prentice Hall, London, 1 Auflage, November 1983.

[CPA 92] CHRISTOPHER, W. A., S. J. PROCTER und T. E. ANDERSON: *The Nachos Instructional Operating System*. Technischer Bericht, University of California, März 1992
`http://http.cs.berkeley.edu/~tea/nachos/nachos.ps`.

[FCZP 00] FANKHAUSER, G., C. CONRAD, E. ZITZLER und B. PLATTNER: *Topsy – A Teachable Operating System*. Technischer Bericht, Computer Engineering and Networks Laboratory, ETH Zürich, März 2000
`http://www.tik.ee.ethz.ch/~topsy/Book/Topsy\_1.1.pdf`.

[GJSB 00] GOSLING, J., B. JOY, G. STEELE und G. BRACHA: *The Java Language Specification*. Addison–Wesley, Reading, Mass., 2 Auflage, jun 2000
`http://java.sun.com/docs/books/jls/second\_edition/html/j.title.doc.html`.

[HGP 95] HENNESSY, J. L., D. GOLDBERG und D.A. PATTERSON: *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., San Francisco, 2 Auflage, August 1995.

[Holu 00] HOLUB, A.: *Taming Java Threads*. Springer, Berlin, 1 Auflage, Februar 2000.

[Kalr 97] KALRA, A.: *Salsa – An Operating System Tutorials*. Technischer Bericht, University of Massachusetts, Mai 1997
`http://saa-www.cs.umass.edu/salsa/index.html`.

[Laru 90] LARUS, J. R.: *SPIM S20: A MIPS R2000 Simulator*. Technischer Bericht, University of Wisconsin-Madison, 1990
`http://www.cs.wisc.edu/~larus/SPIM/spim\_documentation.pdf`.

[Lea 00] LEA, D.: *Concurrent Programming In Java*. Addison–Wesley, Reading, Mass., 2 Auflage, Februar 2000.

[Nart 95] NARTEN, T.: *A Road Map Through Nachos*. Technischer Bericht, Levine Science Research Center, Duke University, Januar 1995
http://www.cs.duke.edu/~narten/110/nachos/main.ps.

[Nitz 97] NITZSCHE, R.: *Einführung in die Assemblerprogrammierung mit dem MIPS-Simulator SPIM*. Technischer Bericht, Freie Universität Berlin, September 1997.

[OnSt 00] ONTKO, R. und D. STONE: *MIC1 – IJVM Simulator*. Technischer Bericht, Prentice Hall, Februar 2000
http://www.ontko.com/mic1/.

[PaHe 97] PATTERSON, D. A. und J. L. HENNESSY: *Computer Organization and Design*. Morgan Kaufmann Publishers, Inc., Reading, Mass., 2 Auflage, Februar 1997.

[SGG 00] SILBERSCHATZ, A, P. B. GALVIN und G. GAGNE: *Applied Operating System Concepts*. John Wiley & Sons, Inc., New York, 1 Auflage, 2000.

[SiGa 98] SILBERSCHATZ, A und P. B. GALVIN: *Operating System Concepts*. Addison–Wesley, Reading, Mass., 5 Auflage, 1998.

[Tane 98] TANENBAUM, A. S.: *Structured Computer Organization*. Prentice Hall, London, 4 Auflage, 1998.

[Tane 96] TANENBAUM, A. S.: *MIC1 – IJVM Simulator*. Technischer Bericht, November 1996
http://www.cs.vu.nl/~ast/minix.html.

[TaWo 97] TANENBAUM, A. S. und A. S. WOODHULL: *Operating Systems: Design and Implementation*. Prentice Hall, London, 2 Auflage, Januar 1997.