Assembler-Versuch innerhalb des Hardware-Praktikums

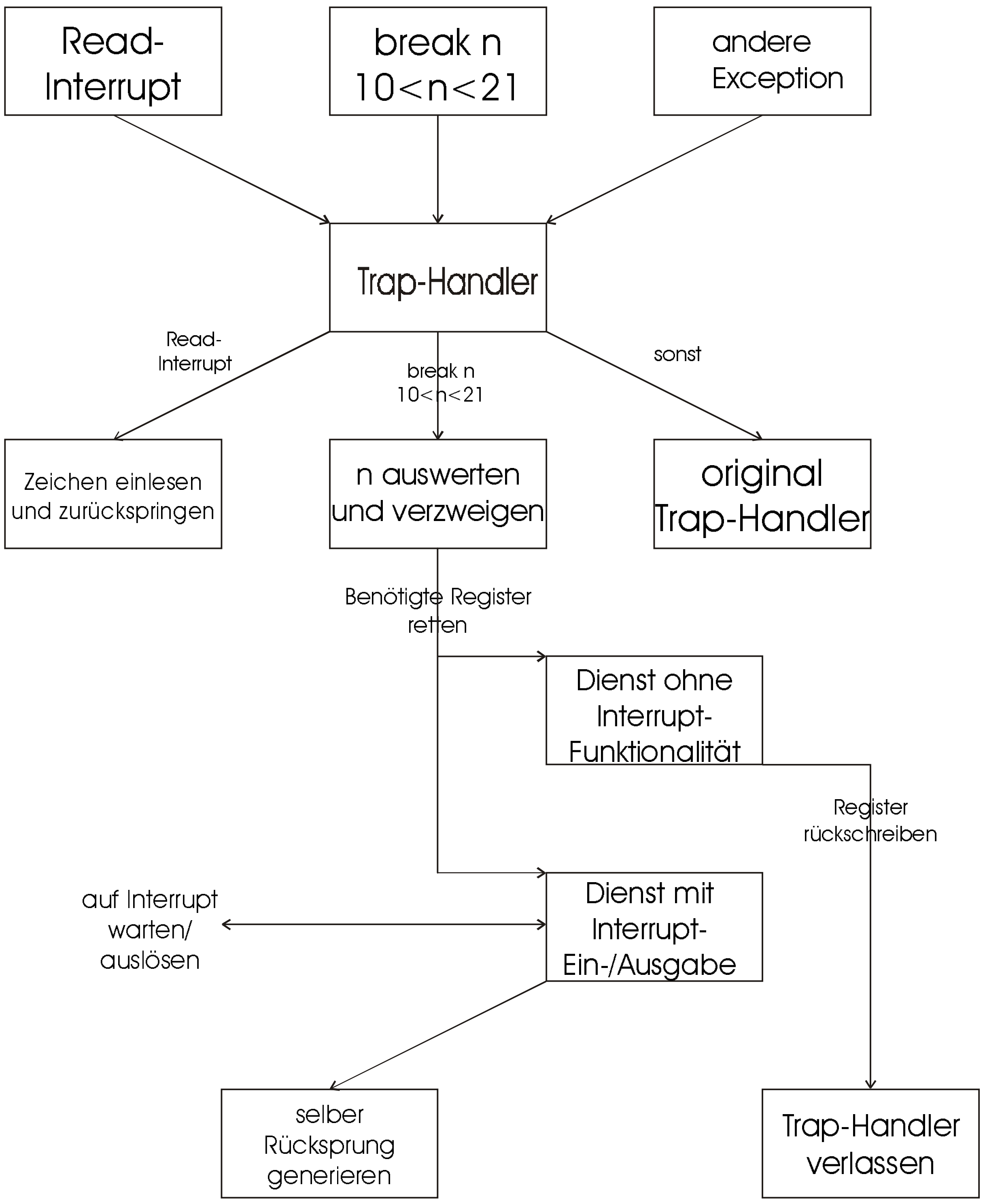
Aufgabe: Entwerfen und implementieren Sie ein SPIM-Programm, welches zusätzliche Dienste durch Befehle break n zur Verfügung stellt.

Torsten Keil
TU Dresden
Fakultät Informatik

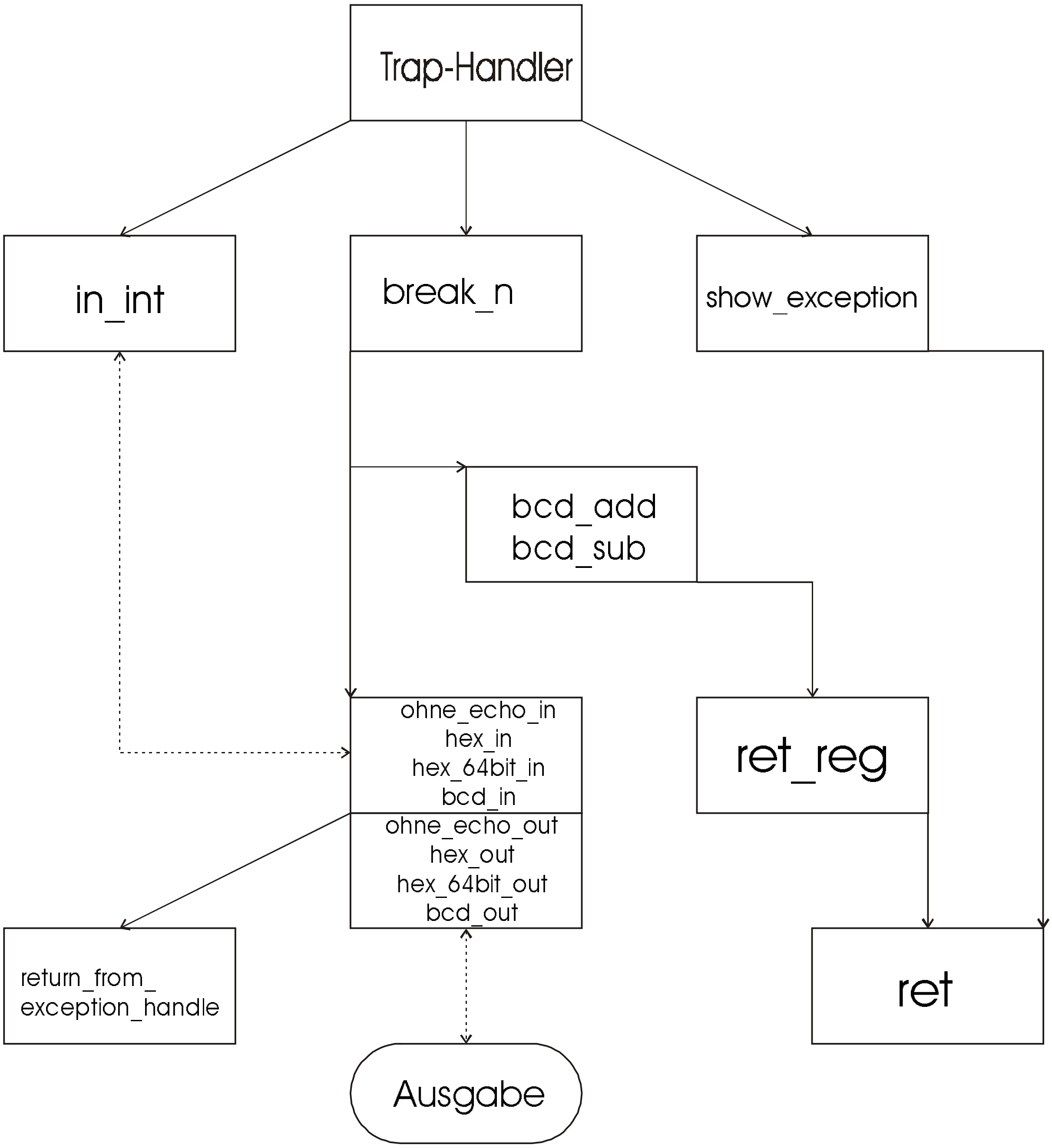
Anlagen:

- Aufgabenstellung
- Bedienungsanleitung - Prüfprogramm
- Aufwandsabschätzung
- Probleme & Vorschläge
- Programmfluß-Darstellung
- Quellcode: trap_ext.asm
- Quellcode: testprgm.asm
- Diskette mit folgenden Dateien:
  Dokumentation.ps; aufgabe.html; trap_ext.asm; testprgm.asm; traptest.asm

Programmfluß-Darstellung schematisch


Read-Interrupt
break n 10<n<21
andere Exception
Trap-Handler
Read-Interrupt
break n 10<n<21
sonst
Zeichen einlesen und zurückspringen
n auswerten und verzweigen
original Trap-Handler
Benötigte Register retten
Dienst ohne Interrupt-Funktionalität
Register rückschreiben
auf Interrupt warten/ auslösen
Dienst mit Interrupt-Ein-/Ausgabe
selber Rücksprung generieren
Trap-Handler verlassen

Programmfluß-Darstellung
Routinen und Dienste


Trap-Handler
in_int
break_n
show_exception
bcd_add
bcd_sub
ohne_echo_in
hex_in
hex_64bit_in
bcd_in
ret_reg
ohne_echo_out
hex_out
hex_64bit_out
bcd_out
return_from_
exception_handle
ret
Ausgabe

Bedienungsanleitung - Prüfprogramm:

Das beiliegende Programm „testprgm.asm“ enthält Vorschläge, wie die Erweiterungen des Trap-Handlers genutzt bzw. getestet werden können. Das Programm ist in Sinneinheiten (Abschnitten) gegliedert. Dabei stehen im ersten Teil der Datei Beispiele für einen Test. Im zweiten Teil findet man hingegen eine Art Beispiel-Umgebung, mit vorbereiteten helfenden Ausgaben auf der Console. Die dafür notwendigen Daten stehen im .data Abschnitt zu Beginn des Programms. Allerdings ist diese Art der Kombination denkbar ungüstig, da bei den dafür notwendigen Syscall-Routinen die Register $v0 und $a0 benötigt werden.
Jegliche Befehle der main-Routine wurden auskommentiert, d.h. um das Programm nutzen zu können, muß man in einem Editor den entsprechenden Abschnitt „aktivieren“. Dies hat den Hintergrund, daß man sich mit dem Aufruf eines Dienstes vorher auseinandersetzen muß - ihn zuvor versteht - um seine genaue Wirkung zu erkennen. Außerdem schützt es den Benutzer davor, beim Ablauf des Prüfprogramms, von der Kette von Befehlen und den damit verbundenen Ein- und Ausgaben „erschlagen“ zu werden.

Folgende Schritte sind nötig, um den erweiterten Trap-Handler und das Prüfprogramm zu nutzen:

- starten eines Editors und laden der Datei “testprgm.asm“
- aktivieren eines Dienstes und speichern der Datei
- starten eines SPIM-Simulators mit -mapped_io
- einstellen der Datei „trap_ext.asm“ als Standard Trap-Handler
- laden und starten des Prüfprogramms

Aufwandsabschätzung:

- Speicherbedarf Trap-Handler (zusätzlich zum Original-Trap-Handler):
  11 x 4byte für Retten der Register in .kdata
  10 x 4byte für den Stack in .kdata
  1 x 4byte für den Stack-Pointer in .kdata
  sowie etwa 3200 x 4byte in .ktext für Trap-Handler
- Speicherbedarf Prüfprogramm:
  Kommentare für Console in .data
  sowie Programmcode in .text

- Entwicklungszeit:
  Die Entwicklungszeit für die Trap-Handler-Erweiterung lag mit Programmierphase, Test, Fehlersuche und erstellen der Dokumentation bei ca. 30~35 Stunden.
  Nicht eingerechnet sind dabei Analysen (z.B. von Bsp.-Programmen, der SPIM-Dokumentation) in Pausen, Vorlesungen oder Straßenbahnfahrten.

Probleme:

Wesentliche Probleme während der Entwicklung waren:

- keine Breakpoints im Kernel möglich
- keine Schrittweise Abarbeitung durch Interruptsteuerung möglich
- erkennen der Funktionsweise der Interruptgesteuerten Ein-/Ausgabe
- Analyse der Aufgabenstellung
- Während der Entwicklung trat noch folgendes Phänomen auf:
  Meine eigene Programmierung führte ich zu Hause auf einem PC mit PCSpim durch. Dabei schlich sich ein Programmierfehler ein der vom PCSpim richtig interpretiert wurde. Der anschließende Probelauf auf einer DEC-Station der Fakultät schlug allerdings fehl und brachte falsche Ergebnisse. Die Suche nach der Ursache dauerte mehrere Stunden.

Vorschläge:

- Integration weiterer Dienste bzw. ähnlicher Dienste mit anderem Rücksprung bei Abbruch des Dienstes
- einführen von Statusvariablen z.B. ob Abbruch bei Eingabe
- verlagern der Eingabeaufforderungen in den Trap-Handler (günstiger, da für Syscall-Aufrufe unmittelbar vor und nach dem break n Befehl die Register $v0 und $a0 benötigt werden.