



h\_da

# Anwenderdokumentation zur Entwicklungsumgebung im Rechnerarchitektur-Praktikum

Sommersemester 2018  
Prof. Dr. Eva Brucherseifer

## Erste Schritte im Labor

- Computer im Labor starten und Einloggen (siehe Termin 1 / Aufgabe 6)
- Laden Sie die Praktikumsaufgaben von der in der Vorlesung bekannt gegebenen Quelle herunter und speichern Sie diese in Ihrem Home-Verzeichnis (z.B. ~/raSS2018/)
  - Hinweis: Im Pfadnamen keine Leerzeichen verwenden, da der snavigator sonst nicht korrekt funktioniert!
- Öffnen Sie eine Konsole
  - Im Labor und in der virtuellen Maschine haben Sie den Desktop KDE im Einsatz. Dort starten Sie den Filemanager dolphin und öffnen Sie dort mit F4 eine Konsole.
- Wechseln Sie in das Verzeichnis für Termin 3



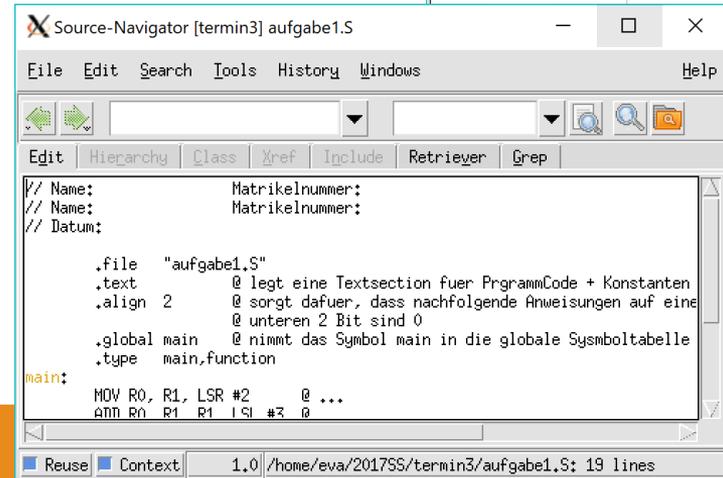
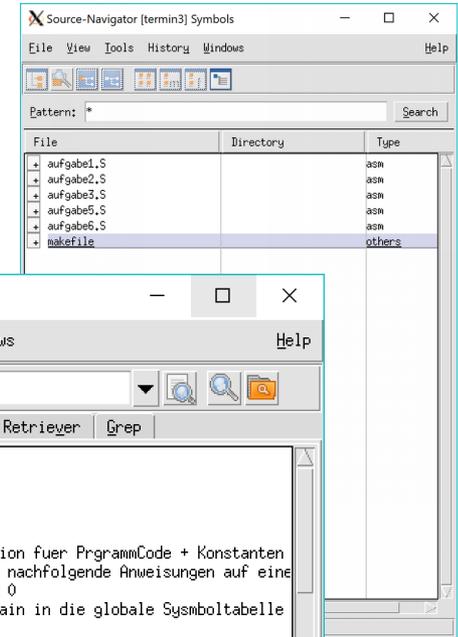
## Alternativ: Erste Schritte auf dem eigenen Laptop

- Installieren Sie die Entwicklungsumgebung, siehe Anleitungen im TechnoWiki der Fachgruppe Technische Informatik (Anmelden erforderlich):  
[Link zum TechnoWiki](#)
  - nativ auf Linux
  - Linux Subsystem in Windows 10
  - Entwicklungsumgebung für VirtualBox  
[Link zur Webseite](#)  
Rechnerarchitektur 32-Bit OVA
- Laden Sie aus Moodle die Praktikumsaufgaben von der in der Vorlesung bekanntgegebenen Quelle herunter und speichern Sie diese auf Ihrem Rechner
- Hinweis: Im Pfadnamen keine Leerzeichen verwenden, da der snavigator sonst nicht korrekt funktioniert! Achtung, Windows legt den Usernamen mit Leerzeichen an (Vor- + Nachname). Dann legen Sie sich besser für die Praktikumsdateien ein separates Verzeichnis an.
- Wechseln Sie in das Verzeichnis für Termin 3



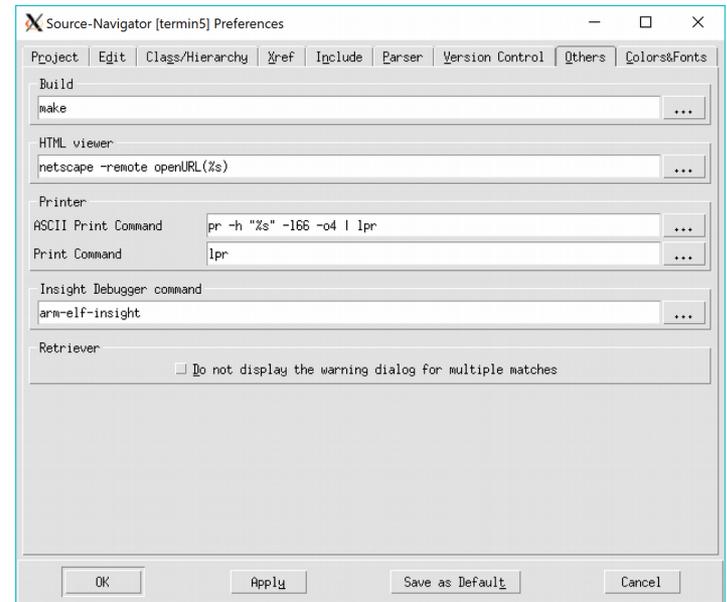
# Entwicklungsumgebung: snavigator

- Auf der Kommandozeile den **snavigator** (IDE) starten.
- Legen Sie ein neues Projekt an (z.B. termin3).  
Prüfen Sie die im Dialog vorgeschlagenen Einstellungen.  
Wenn der snavigator im Projektverzeichnis gestartet wurde, können diese übernommen werden.
- Öffnen Sie eine Datei durch Doppelklick und editieren Sie die Source-Dateien.
- Source-Dateien haben die Endung `.S`



# Entwicklungsumgebung: snavigator

- Konfigurieren Sie den Debugger im snavigator:
  - Öffnen Sie File → Project Preferences... → Others
  - Ändern Sie den Inhalt von „Insight Debugger Command“ auf `arm-elf-insight`
- Hinweis: Es gibt eine alternative Toolchain, die auch Floating Point-Berechnungen unterstützt. Hier wird in den Befehlsnamen „arm-elf“ durch „arm-eb63-elf“ ersetzt.

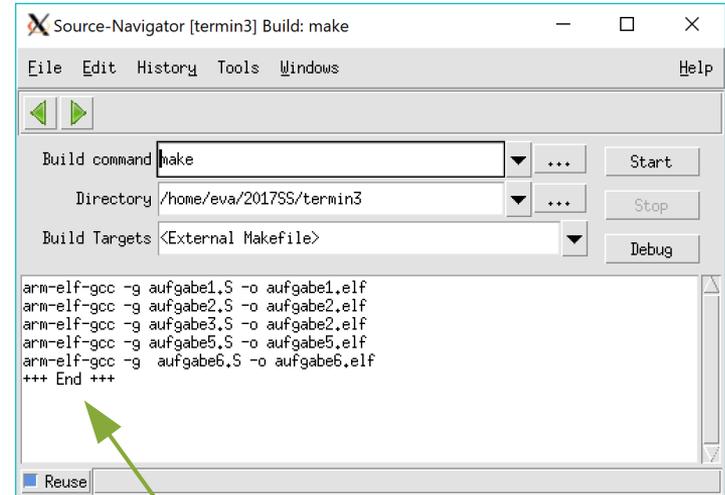


## Entwicklungsumgebung: snavigator

- Editieren Sie die Source-Dateien und führen Sie zum Compilieren make aus:

Tools → Build... → Start

- Die gebauten Binaries haben die Endung .elf
- Im Makefile ist der Build-Vorgang per Skript definiert.

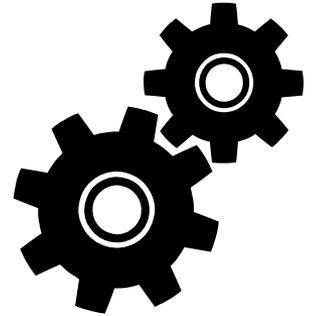


```
Source-Navigator [termin3] Build: make
File Edit History Tools Windows Help
Build command make
Directory /home/eva/2017SS/termin3
Build Targets <External Makefile>
arm-elf-gcc -g aufgabe1.S -o aufgabe1.elf
arm-elf-gcc -g aufgabe2.S -o aufgabe2.elf
arm-elf-gcc -g aufgabe3.S -o aufgabe2.elf
arm-elf-gcc -g aufgabe5.S -o aufgabe5.elf
arm-elf-gcc -g aufgabe6.S -o aufgabe6.elf
+++ End +++
Reuse
```

Hier wird die Ausgabe von make angezeigt, auch Fehlermeldungen, falls der Build-Prozess fehl schlägt.

## Buildtool: make

- Ein Buildtool dient dazu, aus Sourcen Binaries zu erstellen.
- Wenn make in einem Verzeichnis ausgeführt wird, liest es die Datei Makefile ein und führt die dort angegebenen Anweisungen aus:
  - Ein Buildtarget ist eine Datei, die mit Hilfe des Makefiles erstellt werden soll. Targets werden durch den Doppelpunkt danach gekennzeichnet.
  - Ein Target kann Abhängigkeiten zu Dateien oder anderen Targets haben.
  - Die für ein Target auszuführenden Befehle mit Tab oder Leerzeichen müssen eingerückt sein.
  - Die Befehle sind Kommandozeilen-Befehle, z.B. für die Linux Bash.
  - Variablen werden mit Gleichzeichen definiert und können in den Befehlen mit \$(VARIABLE) genutzt werden.



## Buildtool: make

```
# Makefile für Rechnerarchitektur...  
# von: Manfred Pester  
# vom: 12.02.2013 letzte Aenderung...
```

Kommentare

```
# Variable fuer den zu nutzenden Compiler  
GCC = arm-elf-gcc
```

Variable, die unten genutzt wird

```
all: aufgabe1.elf aufgabe2.elf aufgabe3.elf  
aufgabe5.elf Zusatzaufgabe.elf
```

1. Buildtarget: „all“

Das erste Target wird automatisch aufgerufen, wenn nur „make“ getippt wird. „all“ hat Abhängigkeiten zu den anderen Targets, diese werden also nacheinander ausgeführt.

```
aufgabe1.elf: aufgabe1.S  
$(GCC) -g aufgabe1.S -o aufgabe1.elf
```

2. Buildtarget: „aufgabe1.elf“ = zu bauende Dateien.

Sofern die Datei „aufgabe1.S“ vorhanden ist und neuer als die Zielfeile (aufgabe1.elf) ist, werden die folgenden Kommandozeile ausgeführt.

```
[...]  
Zusatzaufgabe: Zusatzaufgabe.S  
$(GCC) -g Zusatzaufgabe.S -o  
Zusatzaufgabe.elf
```

Die Kommandozeile ruft den oben in der Variable angegebenen Befehl auf mit den angegebenen Parametern. .S ist die Source-Datei, .elf das Binary.

```
clean:  
rm *.o  
rm *.elf
```

Die Targets „all“ und „clean“ sind keine Dateien, die gebaut werden sollen, sondern nur Marken. Daher werden diese als „phony“ markiert.

```
.PHONY: all clean
```

# Debugger: insight

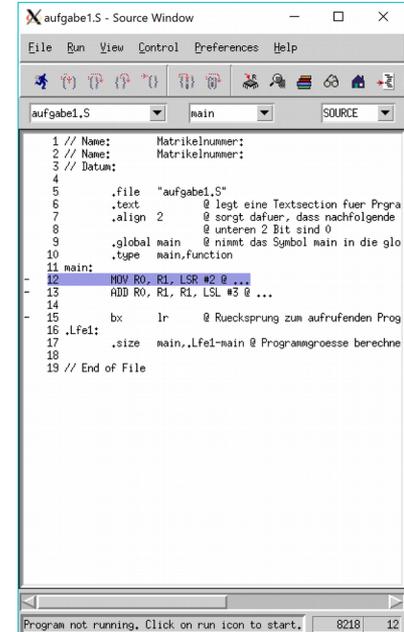
- Im navigator können Sie den Debugger (insight) über

Tools → Debugger

starten und im folgenden Dialog das zu debuggende Binary auswählen.

- Alternativ zum navigator können die Dateien auch mit einem Editor wie z.B. kate oder Notepad++ bearbeitet werden und make sowie insight von der Kommandozeile aufgerufen werden.
- Insight ist Teil der ARM-Toolchain und wird mit folgendem Befehl gestartet:

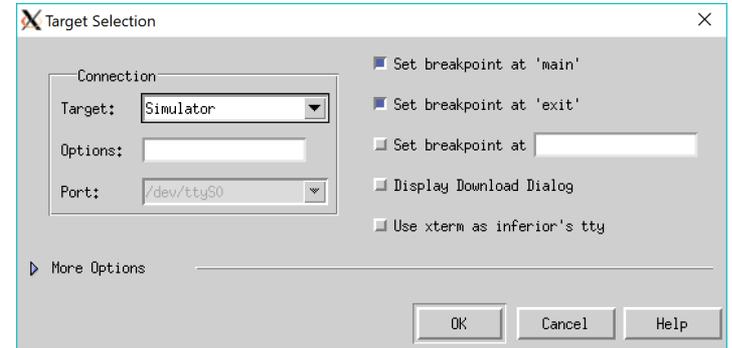
`arm-elf-insight`



```
1 // Name:      Matrikelnummer:
2 // Name:      Matrikelnummer:
3 // Datum:
4
5 .file "aufgabe1.5"
6 .text       @ legt eine Textsection fuer Progra
7 .align 2    @ sorgt da fuer, dass nachfolgende
8             @ unteren 2 Bit sind 0
9 .global main @ nimmt das Symbol main in die glo
10 .type main,function
11 main:
12 MOV R0, R1, LSR #2 @
13 ADD R0, R1, R1, LSL #3 @ ...
14
15 bx lr       @ Ruecksprung zum aufrufenden Prog
16 .Lfe1:
17 .size main, .Lfe1-main @ Programgroesse berechne
18
19 // End of File
```

## Debugger: insight

- In Insight die Zielplattform konfigurieren:  
File → Target Settings...
- Als Target  
    **Simulator**  
auswählen.



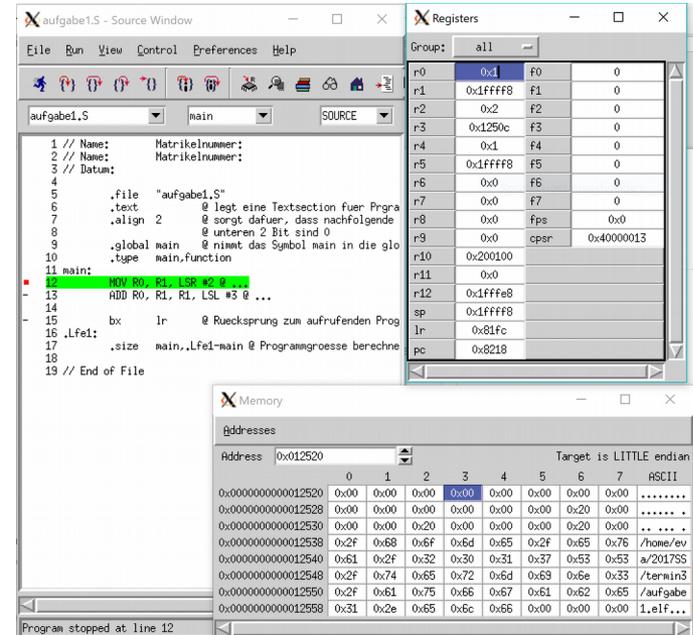
# Debugger: insight

- Unter View folgende Fenster öffnen:

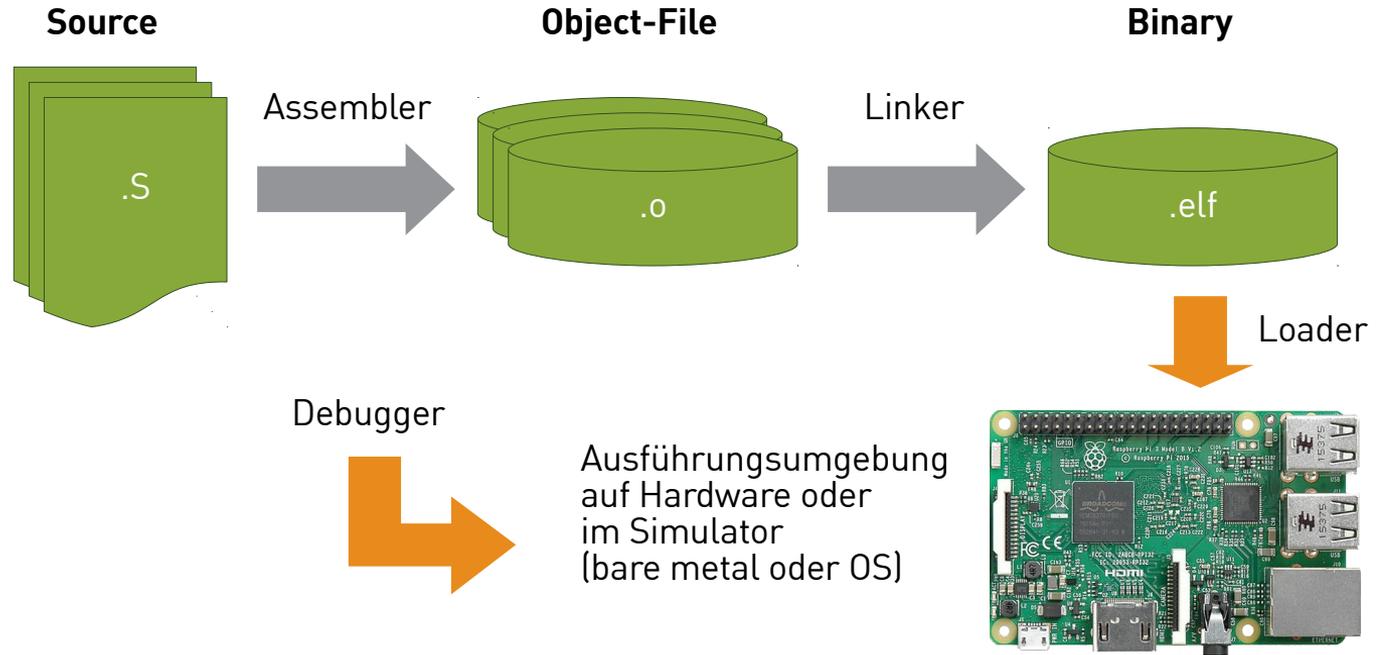
- Register
- Memory

Hier können Sie beim Debuggen die Register und den Speicherinhalt einsehen und auch verändern

- Mit den Buttons in der Toolbar können Sie nun den Debugger ausführen, z.B. schrittweise.
- Breakpoints können Sie direkt im Source-Code setzen.

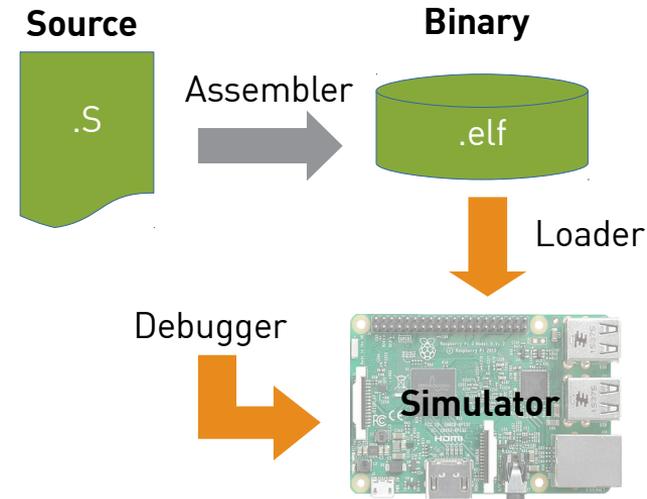


# Toolchain am Beispiel GNU ARM (Allgemeinfall)



## Toolchain im Praktikum

- **Sonderfall im Praktikum:** Da in den Praktikumsaufgaben immer nur eine Source-Datei vorhanden ist, gibt es keine Objektdateien und das Binary kann direkt ohne Linker gebaut werden.
- Die Source-Datei wird vom Assembler in ein Binary für die Zielplattform compiliert. Damit auf einem Intel für ARM (cross-)compiliert werden kann, wird eine **Cross-Toolchain** verwendet.
- Der Debugger führt das Binary in einem Simulator der Zielplattform aus.  
Dadurch kann  
**ARM-Software**  
auf einem  
**Intel-Host-System** ausgeführt werden.



## Tools im Praktikum

### snavigator

- Projekt
- Editor
- startet make
- startet insight

### insight

- Frontend für gdb

### make

- führt die Toolchain aus

Source



### as

Assembler



Binary

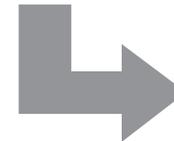


Loader

### gdb

- führt das Binary im Simulator aus
- debuggt

Debugger



## Tools im Praktikum

- IDE/ Editor:           snavigator
  - Build Tool:            make
  - Compiler:             gcc                    Binary: arm-elf-gcc     (ruft arm-elf-as auf)
  - Debugger:            gdb                    Binary: arm-elf-gdb
  - Debugger GUI:        insight                Binary: arm-elf-insight
- 
- Die Programme mit „arm-elf“ im Dateinamen sind Teil der Cross-Toolchain.
  
  - Hinweis:  
Es gibt eine alternative Toolchain, die auch Floating Point-Berechnungen unterstützt.  
Hier wird in den Befehlsnamen „arm-elf“ durch „arm-eb63-elf“ ersetzt.

## Dateinamen

.s	Assembler Source-Code
.S	Assembler Source-Code, vorher wird der GNU Präprozessor ausgeführt. Das erlaubt z.B. Kommentare im Code
.o	Object-Dateien
.elf	ARM-Binaries im elf-Format
Makefile	Anweisungsdatei für make

## Typische Fehler

- Leerzeichen im Pfadnamen oder Datei, das führt zu Problemen mit snavigator  
=> umbenennen
- Mit snavigator kein neues Projekt angelegt, sondern ein Projekt von einem anderen Rechner übernommen  
=> neu erstellen
- Der snavigator wurde nicht im Projektordner gestartet (Symbols-Fenster voll mit Dateien...)  
=> snavigator schließen und im Projektordner nochmal neu starten oder  
=> beim Anlegen des Projekts darauf achten, dass alle Pfade stimmen

## Weitere Informationsquellen

- Es gibt eine Reihe Anleitungen zur Installation und Nutzung der Entwicklungsumgebung:  
im TechnoWiki der Fachgruppe Technische Informatik (Anmelden erforderlich):  
<https://wiki.h-da.de/fbi/technische-systeme/index.php/Rechnerarchitektur>
- Offene Labore  
=> Tutoren oder Laboringenieure fragen