# Praktikum 5 (10 Punkte) - Assembler

von Tim Steven Meier und Fabian Husemann

### **Aufgabe 1**

### **-00**

Reduziert Kompilierzeit und liefert das erwartete Ergebnis beim Debugging. (Standard)

#### -02

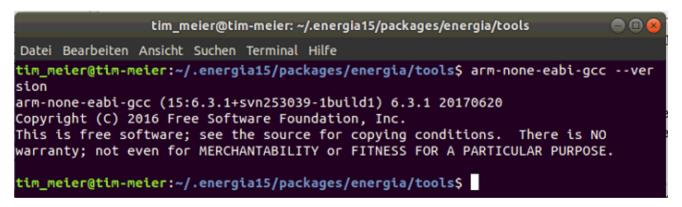
Optimiert mit allen möglichen Optimierungen . Das Kompilieren (GCC) dauert länger aber die Performanz des Codes ist besser.

#### -Os

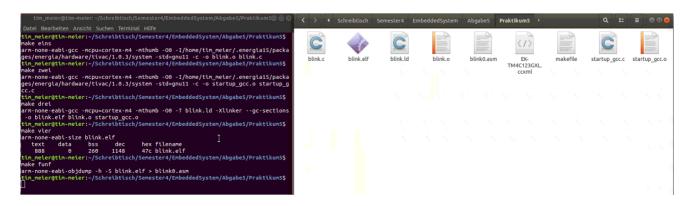
Benutzt die ähnlichen Optimierungen wie -O2 lässt jedoch die Optimireungen weg die die Größe des Codes vergrößern. Erreicht dadurch die kleinst mögliche Größe

- -Os deaktiviert die folgenen Optimierungs flags:
- -falign-functions -falign-jumps -falign-loops -falign-labels -freorder-blocks -freorder-blocks-algorithm=stc freorder-blocks-and-partition -fprefetch-loop-arrays

### **Compiler Version**



### Code



```
2ae:
      4b0d
                 ldr r3, [pc, #52] ; (2e4 <main+0x48>)
2b0:
      2208
                 movs
                       r2, #8
      601a
                 str r2, [r3, #0]
2b2:
                 ldr r3, [pc, #48] ; (2e8 <main+0x4c>)
      4b0c
2b4:
2b6:
     2208
                 movs r2, #8
2b8:
                str r2, [r3, #0]
     601a
2ba:
      4a0c
                ldr r2, [pc, #48] ; (2ec <main+0x50>)
2bc:
     4b0b
               ldr r3, [pc, #44] ; (2ec <main+0x50>)
               ldr r3, [r3, #0]
2be:
     681b
2c0: f043 0308 orr.w r3, r3, #8
2c4: 6013 str r3, [r2, #0]
    f7ff ffd1 bl 26c <delay>
2c6:
2ca:
     4a08 ldr r2, [pc, #32] ; (2ec <main+0x50>)
     4b07
                ldr r3, [pc, #28] ; (2ec <main+0x50>)
2cc:
2ce: 681b ldr r3, [r3, #0]
2d0: f023 0308 bic.w r3, r3, #8
      6013 str r3, [r2, #0]
2d4:
2d6: f7ff ffc9 bl 26c <delay>
```

Die erste Spalte ist die Befehlsadresse im Code.

Die zweite Spalte ist der Befehl in Maschinensprache, hier in Hexadezimal, dabei gibt es zwei verschiedene Längen.

Die dritte Spalte zeigt den Befehl

Alles nach einem Semikolon ist ein Kommentar

Befehl	Erläuterung
ldr r3, [pc, #52]	Lädt den Wert der Adresse des Programmcounters mit einem Offset von 52 (GPIO_PORTF_DIR_R Adresse) in das Register r3
movs r2, #8	Lädt den Wert 8 in das Register r2
str r2, [r3, #0]	Speichert r2 in r3
ldr r3, [pc, #48]	Lädt den Wert der Adresse des Programmcounters mit einem Offset von 48 (GPIO_PORTF_DEN_R Adresse) in das Register r3
movs r2, #8	Lädt den Wert 8 in das Register r2
str r2, [r3, #0]	Speichert den Wert von r2 in r3
ldr r2, [pc, #48]	Lädt den Wert der Adresse des Programmcounters mit einem Offset von 48 (GPIO_PORTF_DEN_R Adresse) in das Register r2
ldr r3, [pc, #44]	Lädt den Wert der Adresse des Programmcounters mit einem Offset von 44 in das Register r3
ldr r3, [r3, #0]	Lädt den Wert des Registers R3 in R3
orr.w r3, r3, #8	Der Befehl verknüpft r3 und 8 mit einem Logischen OR. Das .w steht dafür das der Assambler ein 32 Bit Encoding verwendet
str r3, [r2, #0]	Speichert den Wert von r3 in r2
bl 26c <delay></delay>	Der Programmcounter wird auf die Adresse 26c (delay) gesetzt
ldr r2, [pc, #32]	Lädt den Wert der Adresse des Programmcounters mit einem Offset von 32 in das Register r2
ldr r3, [pc, #28]	Lädt den Wert der Adresse des Programmcounters mit einem Offset von 28 in das Register r3
ldr r3, [r3, #0]	Lädt den Wert des Registers R3 in R3
bic.w r3, r3, #8	Das Register r3 wird gecleared. Das .w steht dafür das der Assambler ein 32 Bit Encoding verwendet
str r3, [r2, #0]	Speichert den Wert von r3 in r2

Befehl	Erläuterung
bl 26c <delay></delay>	Der Programmcounter wird auf die Adresse 26c (delay) gesetzt

## **Aufgabe 2**

```
tim_meier@tim-meier: ~/Schreibtisch/Semester4/EmbeddedSystem/Abgabe5/Praktikum5_2 🗐 🕕
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
blink.c blink.elf blink.ld EK-TM4C123GXL.ccxml
                                                    makefile
tim_meier@tim-meier:~/Schreibtisch/Semester4/EmbeddedSystem/Abgabe5/Praktikum5_2
$ make eins
arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -O2 -I/home/tim_meier/.energia15/packa
ges/energia/hardware/tivac/1.0.3/system -std=gnu11 -c -o blink.o blink.c
tim_meier@tim-meier:~/Schreibtisch/Semester4/EmbeddedSystem/Abgabe5/Praktikum5_2
$ make zwei
arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -O2 -I/home/tim_meier/.energia15/packa
ges/energia/hardware/tivac/1.0.3/system -std=gnu11 -c -o startup_gcc.o startup_g
cc.c
tim_meier@tim-meier:~/Schreibtisch/Semester4/EmbeddedSystem/Abgabe5/Praktikum5_2
$ make drei
arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -O2 -T blink.ld -Xlinker --gc-sections
 -o blink.elf blink.o startup gcc.o
tim_meier@tim-meier:~/Schreibtisch/Semester4/EmbeddedSystem/Abgabe5/Praktikum5_2
$ make vier
arm-none-eabi-size blink.elf
   text
           data
                    bss
                            dec
                                    hex filename
    844
                    260
                           1104
                                    450 blink.elf
tim_meier@tim-meier:~/Schreibtisch/Semester4/EmbeddedSystem/Abgabe5/Praktikum5_2
 make funf
arm-none-eabi-objdump -h -S blink.elf > blink2.asm
   <u>_meier@tim-meier:~/Schreibtisch/Semester4/EmbeddedSystem/Abgabe5/Praktikum5_2</u>
```

Durch -O2 hat sich der Programmcode um 44 Bytes verkleinert.

In der neu erzeugten Datei gibt es keinen eigenen Abschnitt mit , dadurch ist die Main Methode zwar etwas größer, aber insgesamt ist es dadurch kleiner.

# **Aufgabe 3**

volatile ist in C ein Zusatz beim deklarieren von Variablen und gibt an, dass sich der Wert der Variablen jederzeit ändern kann. Dadurch führt der Compiler an der Stelle keinen Optimierungen durch. volatile ist nun nicht mehr vorhanden und dadurch gibt es auch an jeder Stelle Optimierungen

In der neu erzeugten Datei ohne volatile gibt es einen Abschnitt mit delay, welcher in der main zweimal aufgerufen wird. Dadurch ist die main kleiner, da der Code nicht zweimal aufgeschrieben werden musste. Ansonsten sind beide Dateien ziemlich ähnlich.