



## 6. Aufgabenblatt

05.06.2023

Compilieren, Assemblieren und Linken

### Aufgabe 1: Theoriefragen

- (a) Welche (wichtigen) Informationen werden im Header eines Objektprogramms (z. B. ELF-Format) abgelegt.
- (b) Erläutern Sie die Aufgaben eines Binders/Linkers und eines Laders.

### Aufgabe 2: Codeanalyse (optimierender) Compiler

Gegeben ist folgendes Code-Fragment:

```
#include <stdio.h>

int main (void)
{
    int n = 20;
    int erg;

    if(n == 20) {
        erg = n * 2;
    }

    printf("Ergebnis_%d\n",erg);

    return 0;
}
```

Im Folgenden sollen ein paar Analysen vorgenommen werden.

- a) Wie kann der Ausdruck `erg = n * 2;` des C-Programms effizient berechnet werden?

- 
- b) Analysieren Sie den vom GCC Compiler generierten Code. Das Programm soll mit `gcc test.c` (also ohne Optionen) übersetzt werden.<sup>1</sup> Disassemblieren Sie den vom GCC erzeugten Code (`objdump -S ./a.out`). Geben Sie das Programmfragment in Assembler an, welches den Ausdruck `erg = n * 2;` berechnet.
- c) Der Ausdruck im obigen C-Programm hat nun die Form `erg = n * 8;`. Disassemblieren Sie den vom GCC erzeugten Code (`objdump -S ./a.out`). Geben Sie das Programmfragment in Assembler an, welches den Ausdruck `erg = n * 8;` berechnet.
- d) Der Ausdruck im obigen C-Programm hat nun die Form `erg = n / 8;`. Disassemblieren Sie den vom GCC erzeugten Code (`objdump -S ./a.out`). Geben Sie das Programmfragment in Assembler an, welches den Ausdruck `erg = n / 8;` berechnet. Erklären Sie das Programmfragment.

### Aufgabe 3: Reverse Engineering

Gegeben ist folgender Object Dump:

```
00010440 <up>:
10440: e92d4800  push {fp, lr}
```

---

<sup>1</sup> Nutzer\*innen des clientssh-arm bitte daran denken, den `arm-linux-gnueabi-hf-gcc` aufzurufen.

```

10444: e28db004  add fp, sp, #4
10448: e24dd010  sub sp, sp, #16
1044c: e50b0010  str r0, [fp, #-16]
10450: e51b3010  ldr r3, [fp, #-16]
10454: e3530000  cmp r3, #0
10458: 1a000001  bne 10464 <up+0x24>
1045c: e3a03000  mov r3, #0
10460: ea000010  b 104a8 <up+0x68>
10464: e51b3010  ldr r3, [fp, #-16]
10468: e3530001  cmp r3, #1
1046c: 1a000001  bne 10478 <up+0x38>
10470: e3a03001  mov r3, #1
10474: ea00000b  b 104a8 <up+0x68>
10478: e51b3010  ldr r3, [fp, #-16]
1047c: e2433002  sub r3, r3, #2
10480: e1a00003  mov r0, r3
10484: ebffffed  bl 10440 <up>
10488: e50b0008  str r0, [fp, #-8]
1048c: e51b3010  ldr r3, [fp, #-16]
10490: e2433001  sub r3, r3, #1
10494: e1a00003  mov r0, r3
10498: ebffffe8  bl 10440 <up>
1049c: e1a02000  mov r2, r0
104a0: e51b3008  ldr r3, [fp, #-8]
104a4: e0823003  add r3, r2, r3
104a8: e1a00003  mov r0, r3
104ac: e24bd004  sub sp, fp, #4
104b0: e8bd8800  pop {fp, pc}

```

```

000104b4 <main>:
104b4: e92d4800  push {fp, lr}
104b8: e28db004  add fp, sp, #4
104bc: e24dd008  sub sp, sp, #8
104c0: e3a03014  mov r3, #20
104c4: e50b3008  str r3, [fp, #-8]
104c8: e51b0008  ldr r0, [fp, #-8]
104cc: ebffffdb  bl 10440 <up>
104d0: e50b000c  str r0, [fp, #-12]
104d4: e51b100c  ldr r1, [fp, #-12]
104d8: e59f0010  ldr r0, [pc, #16] ; 104f0 <main+0x3c>
104dc: ebffff81  bl 102e8 <printf@plt>
104e0: e3a03000  mov r3, #0
104e4: e1a00003  mov r0, r3
104e8: e24bd004  sub sp, fp, #4
104ec: e8bd8800  pop {fp, pc}
104f0: 00010564  .word 0x00010564

```

a) Analysieren Sie den Object Dump. Was für eine Funktion wird realisiert.

- b) Implementieren Sie das Programm in C und testen Sie die Funktionsweise.


## Aufgabe 4: Vervollständigen von Codestücken

Gegeben ist nachfolgend ein Programm als C-Code und Assembler-Code. In Beiden fehlen jeweils einzelne Codestücke.

<pre>#include &lt;stdio.h&gt;  #define EZIS 2  int d[EZIS] = {6, 12}; int e[EZIS] = {1, 0};  int a() {     int c = 0;     for (int f = 0; f &lt; EZIS; f++) {         _____     }     return c; }  int b(int f[EZIS]) {     _____     for (int h = 0; h &lt; EZIS; h++) {         c += f[h];     } }</pre>	<pre>.global main .data varZ: _____ varY: _____ string: .asciz "%d\n" .text label_1:     _____     mov r0, #0     mov r1, #0     ldr r2, =varZ     ldr r3, =varY     _____     cmp r1, #2     bge label_3     _____     ldr r5, [r2, r4]     ldr r6, [r3, r4]     mul r5, r6, r5</pre>
--	--

<pre>     }     return c; }  int main() {     -----     f = f * b(e);     -----     return 0; } </pre>	<pre>         add r0, r0, r5         add r1, r1, #1         b label_2 label_3:     -----         bx lr label_4:         push {r4}         mov r2, r0         mov r0, #0         mov r1, #0 label_5:         cmp r1, #2         bge label_6     -----     -----         add r1, r1, #1         b label_5 label_6:         pop {r4}         bx lr main:         push {lr}         bl label_1         mov r1, r0         push {r1}         ldr r0, =varY     -----         pop {r1}         mul r1, r0, r1         ldr r0, =string         bl printf         mov r0, #0         pop {lr}         bx lr </pre>
--	--

a) Vollständigen Sie den C-Code und Assembler-Code.



---

Tipp: Verwenden Sie dazu die Informationen aus dem Code der jeweils anderen Sprache.



- 
- b) Benennen Sie die Variablen und Funktionen bzw. Labels entsprechend ihrer Funktionalität um. Kommentieren Sie den Assembler-Code.