Rechnerorganisation

Sommersemester 2023

Prof. Stefan Roth, Ph.D.

TECHNISCHE UNIVERSITÄT DARMSTADT

7. Aufgabenblatt

12.6.2023
Compilieren, Assemblieren und Linken

Aufgabe 1: Maschinensprache vs. Hochsprache

a) Wann ist es sinnvoll, Programme direkt in Assembler (statt in C oder JAVA) zu schreiben?

b) Nennen Sie einige Vorteile einer Hochsprache wie C gegenüber Assembler.

Aufgabe 2: Parity-Bit

In der Netzwerktechnik werden oft verschiedene Arten von Parity-Bits genutzt, um sicher zu gehen, dass die Datenübertragung einwandfrei funktioniert hat. Das Parity-Bit in seiner einfachsten Form ist die Vervollständigung einer Bitfolge auf eine gerade Anzahl von 1er durch anfügen eines Bits. Als Beispiel wird die Bitfolge `1011` auf `1011 1` und `0110` auf `0110 0` erweitert. Schreiben Sie ein ARM-Assemblerprogramm, das eine gegebene 4-Bit Zahl in `r0` um das oben beschriebene Parity-Bit erweitert und das Ergebnis wieder in `r0` ablegt. Sprünge oder konditionale Befehle stehen dabei nicht zur Verfügung. Kommentieren Sie Ihren Code.

Aufgabe 3: Inline Assembler Code

Es ist möglich und manchmal auch sehr sinnvoll, Assemblercode in Hochsprachen wie z. B. C zu verwenden. Folgendes Beispiel zeigt Ihnen, wie man Assemblercode einbetten kann. Formulieren Sie das Programm aus Aufgabe 2 als Unterprogramm in C. Benutzen Sie dabei die Möglichkeit des Inline Assembler Code.

```
#include <stdio.h>

int mad(int i, int j, int k)
{
  int res = 0;
  __asm ("MUL␣r3,␣%[input_i],␣%[input_j]\n"
         "ADD␣%[result],␣r3,␣%[input_k]"
    : [result] "=r" (res) /* outputs: '+' = rw, '=' = wo */
    : [input_i] "r" (i), [input_j] "r" (j), [input_k] "r" (k) /* inputs */
    : "r3" /* other used registers */
  );
  return res;
}

int main(void)
{
  int a = 23;
  int b = 42;
  int c = 666;
  int d = 0;

  d = mad(a,b,c);

  printf("Result␣of␣%d␣*␣%d␣+␣%d␣=␣%d\n", a, b, c, d);
  return 0;
}
```

Aufgabe 4: Konditionale Befehlsausführung

In der Vorlesung haben Sie folgende Möglichkeit der Abbildung eines if/then Konstrukts in Assembler kennengelernt:

```
/* Hochsprache */
if (apples == oranges)
  f = i + 1;
else
f = f - i;
----------------------
/* ARM - Assemblersprache */
/* r0=apples; r1=oranges, r2=f, r3=i */
cmp r0,r1
bne L1
add r2,r3,#1
b L2
L1:
sub r2,r2,r3
L2:
```

Der ARM-Prozessor erlaubt, wie viele moderne Prozessoren, eine sogenannte konditionale Befehlsausführung. Das bedeutet, dass Befehle nur dann ausgeführt werden, wenn bestimmte Bedingungen erfüllt sind.

a) Implementieren Sie obiges if/then Konstrukt in ARM-Assembler ohne die Nutzung von Sprungbefehlen. Zur Implementierung dieses Programms ist es sinnvoll, sich im Reference Guide[1] die Konditionen für die Befehlsausführung anzuschauen.

b) Welche Vorteile hat die Übersetzung mit konditionalen Befehlen?

[1] https://moodle.tu-darmstadt.de/pluginfile.php/1809290/mod_folder/content/0/Learning%20Nugget%2002%20-%20ARM%20Assembler%20Befehle/arm_instruction_set_reference_guide.pdf

Aufgabe 5: Nutzung der Statusflags

Für die Abbildung von Kontrolloperationen in Assembler oder zur Implementierung von Mehrwort-Addition stehen verschiedene Statusflags zur Verfügung. Diese sind:

- C – Carryflag (Übertragsflag)
- Z – Zeroflag (Nullflag)
- N – Negativflag (Vorzeichenflag)
- V – Overflowflag (Überlaufflag)

Der Befehl, der bei der Abbildung von Kontrolloperationen zur Setzung der Statusflags verwendet wird, heißt `cmp`. ARM-Prozessoren haben die Besonderheit, dass z. B. die Befehle zur Addition (`add`) und Subtraktion (`sub`) die Statusflags nicht automatisch setzen. Um zu erreichen, dass diese Befehle die Statusflags setzen muss dem Befehl ein s angefügt werden. Die Syntax eines Additionsbefehls lautet dann `adds`.

In der Vorlesung haben Sie folgende Möglichkeit der Abbildung eines if/then Konstrukts in Assembler kennengelernt:

```
/* Hochsprache */
if (apples == oranges)
  f = i + 1;
else
f = f - i;
----------------------
/* ARM - Assemblersprache */
/* r0=apples; r1=oranges, r2=f, r3=i */
cmp r0,r1
bne L1
add r2,r3,#1
b L2
L1:
sub r2,r2,r3
L2:
```

Implementieren Sie das Assemblerprogramm ohne die Nutzung von `cmp`.

Aufgabe 6: Current Program Status Register

Das Current Program Status Register enthält u. a. auch die Statusflags. Der Befehl `mrs` kopiert den Inhalt des Current Program Status Register in ein beliebiges Register. Erweitern Sie das Programm aus Aufgabe 5 und lassen Sie sich das Zeroflag ausgeben.