EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Vorlesung Informatik der Systeme
SS 2015
Prof. Dr. M. Menth
Dipl.-Inform. W. Braun

Exercise 8

9. Juni 2015

Abgabe: 16. Juni 2015, 10.00 Uhr

Problem 8.1: *Übersetzungsvorgang*

In dieser Aufgabe werden die einzelnen Schritte beim Compilieren eines C-Programms genauer untersucht. Der praktische Teil kann z.B. mit dem C-Compiler aus der freien *GNU Compiler Collection* und dem Programm `objdump` gelöst werden. Diese sind für alle weit verbreiteten Betriebssysteme verfügbar, auf einigen davon bereits vorinstalliert. Spezifische Installationsanleitungen lassen sich online leicht finden; unter Ubuntu werden z.B. mit den Paketen `build-essential` und `binutils` alle für die Übung notwendigen Programme installiert. Zudem sind `gcc` und `objdump` auf den Rechnern in den Linux-Pools des ZDV installiert, damit wurde die Aufgabe getestet.

Informieren Sie sich (durch Aufrufen von `gcc -help`, in Handbüchern, in den Man-Pages, mit einer Webrecherche, ...), wie der Compiler bedient wird, insbesondere wie Sie den Compiler dazu bringen, den Output des Präprozessors oder nur den generierten Assemblercode anzuzeigen.

Verwenden Sie relevante Ausschnitte aus den Programm-Outputs und den generierten Dateien um Ihre Antworten zu belegen. Damit zeigen Sie außerdem, dass Sie die praktischen Aufgaben bearbeitet haben. Wenn Sie einen anderen Compiler verwenden, geben Sie bitte kurz an, um welchen es sich handelt, und beachten Sie, dass der Compiler in den ersten Teilen ohne Optimierung laufen soll.

Falls Sie es nicht schaffen, den Code zu Übersetzen, können Sie im Moodle eine Zip-Datei herunterladen, die einige der geforderten Dateien und Ausgaben enthält; damit können Sie zumindest einen Teil der Aufgaben bearbeiten. Wenn Sie Ihre Lösung mit der in der Zip-Datei vergleichen: Es ist normal, dass sich die Dateien stark unterscheiden (auch im Assembler-Syntax), da sie wahrscheinlich für verschiedene Architekturen erstellt wurden.

Als C-Code für diese Aufgabe wird folgendes Programm verwendet (um den generierten Code klein zu halten, erfolgt die Ausgabe nicht wie üblich als Textausgabe, sondern als Rückgabewert der `main`-Funktion):

```c
#define UPPER 15
const int lower = 12;

int sum = 0;

int main(void) {
  int i;
  for(i = lower; i < UPPER; i++) {
    sum += i;
  }
  return sum;
}
```

1. Präprozessor — 1 Points

   Lassen Sie sich den Output des Präprozessors anzeigen, wenn er auf den oben angegebenen C-Code angewendet wird.

   Was hat sich im Vergleich zum Originalcode geändert?

   Welche Aufgaben hat der Präprozessor allgemein; warum ist dieser zusätzliche Schritt beim übersetzen sinnvoll?

2. Erzeugen von Assembler-Code — 4 Points

   Erzeugen Sie nun aus dem C-Code Assembler-Code und analysieren Sie diesen: Welche Konstrukte im Assembler-Code entsprechen `i`, `sum`, `lower` und `UPPER`? Finden Sie die Stellen, die den 3 Anweisungen `i = lower`, `sum += i` und `i < UPPER` entsprechen. Geben Sie die entsprechenden Assembler-Anweisungen an und erläutern Sie jede dieser Anweisungen kurz. Wie wurde die `for`-Schleife auf Assembler-Ebene abgebildet? Erklären Sie den groben Ablauf. Wahrscheinlich gibt es am Anfang des Schleifen-Äquivalents einen Jump-Befehl, der nicht von irgendwelchen Bedingungen abhängt. Warum wird hier an eine andere Stelle des Codes gesprungen (welche Nachteile hätte eine alternative Implementierung ohne diesen Jump-Befehl)?

3. Optimierter Code — 1 Points

   Erzeugen Sie nun (bei `gcc` mit der zusätzlichen Option `-O2`) optimierten Assembler-Code. Was macht das Assemblerprogramm nun? Erklären Sie dieses Verhalten.

4. Disassemblierung — 2 Points

   Mit dem Unix-Befehl `objdump -d <programm>` (oder einem Äquivalent für Ihren Compiler) kann ein Programm in Maschinensprache disassembliert werden, so dass Maschinensprache-Instruktionen und rekonstuierter Assembler-Code zusammen angezeigt werden. Erzeugen Sie aus dem C-Code ein ausführbares Programm und untersuchen Sie anschließend den disassemblierten Maschinen- und Assembler-Code. Wieviele Byte Maschinencode werden benötigt um zwei Register (auf manchen Architekturen evlt. auch Speicherzellen) zu addieren?

Problem 8.2: *Inline-Funktionen*

```
1  int max(int a, int b) {
2    if (a > b)
3      return a;
4    else
5      return b;
6  }
7
8  int min(int a, int b) {
9    if (a < b)
10     return a;
11   else
12     return b;
13 }
14
15 int minOfMax(int a, int b, int c, int d) {
16    int tmp1 = max(a, b);
17    int tmp2 = max(c, d);
18    int tmp3 = min(tmp1, tmp2);
19    return tmp3;
20 }
```

1. Übersetzen Sie diese Funktionen nach MIPS! Wieviele Instruktionen werden benötigt? 4 Points
2. Schreiben Sie die Funktion minOfMax in C-ähnlichen Pseudo-Code um, indem Sie die enthaltenen Funktionsaufrufe als inline Funktionen abbilden! 4 Points
3. Übersetzen Sie die modifizierte Funktion ebenfalls nach MIPS! Wieviele Instruktionen werden jetzt benötigt? 4 Points

Die Fachschaft möchte Sie zum diesjährigen Sommerfest der Informatik am Freitag, 26.06.2015 ab 18 Uhr auf der WSI-Sonnenterasse (vor dem Raum A104) herzlich einladen!

Total: 20 Points