Übungsblatt 5

Aufgabe A – C & Assembler

Die Programmieraufgaben dieses Teil sollen mit dem 64Bit-„Netwide Assembler“- kurz NASM – gelöst werden. NASM ist x_86 Assembler der Intelsyntax verwendet. Wenn Ihnen also ein x86-PC mit einem 64bit Linux zur Verfügung steht, können Sie sich die NASM-Toolchain auf diesem installieren, ansonsten können Sie die Unirechner verwenden.
Folgende Programme benötigen Sie zum NASM-Programmieren. Machen Sie sich mit ihnen vertraut.

- nasm
- gcc
- gdb
- (objdump)

Weiterhin benötigen sie noch einen Editor zum Schreiben der Programme. Es gibt Editoren, die in einer Konsole ausgeführt werden und auch welche, die als „normales“ graphisches Fenster ausgeführt werden. Beispiele:

- nano (Konsole)
- gedit (graphisch)
- kate (graphisch)
- vim (Konsole)

Intro Der C-Compiler wird auf der Kommandozeile mit cc aufgerufen. Er nimmt u. A. diese Optionen:

- `-o output` Der Compiler wir angewiesen das Resultat (kompiliertes Programm) als „output“ zu speichern.
- `-c` Es wird keine ausführbare Datei erstellt, sondern lediglich eine übersetzte `.o` -Datei.
- `-l foo` Es wird die Bibliothek „„libfoo“ dazugelinkt werden, diese muss in einem Suchpfad für Bibliotheken vorhanden sein
- `-L/path/to/libs` Der Pfad „/path/to/libs“ wird als Suchpfad für Bibliotheken hinzugefügt

- `-I/path/to/incs` Analog wird dieser Pfad für Header-Dateien hinzugefügt

Zusätzlich werden die zu kompilierenden Objekte als Operand dazu übergeben, bspw.:

```
$ cc -o programm source1 .c source2 .c ...
```

Während cc der standardmäßig als der auf dem System verwendete C-Compiler eingestellt ist, gibt es auf vielen System mehrere Compiler. Auf den Poolrechnern sind dies gcc und clang, wobei ersteter als cc eingesetzt wird. Diese Compiler unterstützen viele spezifische Flags, nützlich sind vor allem die, die den Compiler anweisen, einen bestimmten C-Standard zu benutzen:

- `-std=c11` Benutzt bei der Kompilation den Standard C11, weitere sind C99, C89 und nicht standardkonforme GNU-CC-Varianten davon.
- `-pedantic` Lässt den Compiler „strikt“ nach Standard arbeiten und lässt keine nicht- standardkonformen Programme zu.

Außerdem kann man Diagnostics anschalten, also Warnungen bei Code der möglicherweise falsch ist.

- `-Wall` Schaltet viele sinnvolle Warnungen an
- `-Wextra` ... und noch mehr
- `-Weverything (Clang)` Schaltet alle Warnungen an

C Hosted Environment Programmeintrittspunkte
Es gibt verschiedene Möglichkeiten den Programmeintrittspunkt zu schreiben:

- `int main(void)`
- `int main(int argc, char *argv[])`

Erstere Variante gibt an, dass das Programm alle vom Nutzer übergebenen Parameter ignoriert, da die Funktion main() keine Möglichkeit hat auf diese zuzugreifen. Möchte man dies jedoch tun, nutzt man die andere Variante, hierbei ist argc der Argument Counter, und gibt somit an wie viele Argumente der Nutzer übergeben hat. Der Argument Vector argv hält die eigentlichen Argumente als Zeichenketten vor, wobei argv[0] (das erste Element im „Array“) der Programmaufruf ist, also:

```
$ ./ programm arg1 "arg2 in quotes "
argv [0]: "./ programm "
argv [1]: "arg1"
argv [2]: "arg2 in quotes "
```

Die zweite Variante der main()-Funktion kann man auch anders schreiben:

- `int main(int argc, char **argv)`
- `int main(const int argc, const char *const argv[argc+1])`

Alle Varianten von main() geben int zurück. Während man jedoch in allen anderen Funktionen die etwas zurückgeben nicht vergessen darf, das auch zu tun (mit return), ist diese Funktion speziell: Es wird implizit 0 zurückgegeben. Der Rückgabewert von main() hat außerdem eine besondere Bedeutung, denn er gibt an, ob das Programm fehlerfrei (Rückgabewert 0) oder fehlerhaft (Rückgabewert 1–255) ablief (mache Programme nutzen den Exit-Code anders, bspw. test(1)).

C-Standardbibliothek: stdio.h

Der Standard-I/O–Header enthält diejenigen Funktionen der C-Standardbibliothek, die wichtig für einfache Ein- und Ausgabe (i. d. R. auf der Konsole) sind.

Die Funktion puts() Nimmt einen String und gibt ihn auf der Standardausgabe stdout aus, und beendet die Zeile mit einem ’\n’.

```
int puts( const char *s);
```

Die Funktion printf() Nimmt als erstes Argument einen Formatstring. Enthält dieser Platzhalter werden die dafür benötigten Werte in den weiteren Argumenten in Reihenfolge der Platzhalter übergeben. Hängt kein terminierendes '\n' an!

```
int printf ( const char *format , ...);
```

Beispiel für printf()

```
int main(void) {
        float pi , daumen ;
        pi = 3.141;
        daumen = 13.374;
        int antwort = pi* daumen ;
        printf ("Die Antwort auf die Frage nach dem Leben , "
        /* Stringliterale die aufeinanderfolgen werden einfach
        zusammengesetzt und zaehlen als eine lange Konstante */
        "dem Universum und dem ganzen Rest: %d\n", antwort );
}
```

Um float auszugeben benutzt man den Platzhalter %f, für char* (also Strings) %s. Kontrollstrukturen **Schleifen** `for`-Schleife Gut geeignet zum iterieren mit Zählvariablen. Dazu werden drei Ausdrücke benutzt, eine Initialisierung, eine Schleifenbedingung und ein Inkrement. Die Initialisierung wird zuerst ausgeführt, danach wird die Schleifenbedingung getestet. Ist sie wahr, wir der Schleifenkörper ausgeführt, danach der Inkrement ausgewertet und dann wieder die Bedingung ausprobiert – solange, bis die Bedingung falsch wird.

```
for ( Ausdruck 1 ; Ausdruck 2 ; Ausdruck 3 ) {
/* Anweisungen */
}
```

Beispielcode

```
# include <stdio .h>

int main(int argc , char *argv []) {
        for (int i = 0; i < argc; i++) {
                printf ("argv [%d]: \"%s\"\n", i, argv[i]);
        }
        return (0);
}
```