Technische Universität München
Fakultät für Informatik
Rechnerarchitektur-Praktikum
SS 2015

DICHTEMESSUNG

ENTWICKLERDOKUMENTATION

Bearbeitet von:

Niklas Rosenstein

12.07.2015

INHALTSVERZEICHNIS

1. Kompilierungsprozess und Anforderungen an das System

Der Assemblercode und das Makefile wurden für eine 32-Bit Linux Platform entworfen. (Wichtig: Dies schließt Mac OS **nicht** mit ein). Die plattformabhängigen Unterschiede liegen hauptsächlich bei der Verlinkung der Objektdateien, siehe (2.) für mehr Details. Das Programm kann auch auf 64-Bit System kompiliert werden, allerdings muss dazu vorher das Paket **`libc6-dev-i386`** installiert werden, andernfalls wird die Header-Datei **`<sys/cdefs.h>`** nicht gefunden.

Das C-Rahmenprogramm wird mit GCC mit der Option **`-m32`** kompiliert, um eine 32-Bit Kompilierung zu forcieren. Dieses Argument wird auch bei der Verlinkung der Objektdateien übergeben. Für NASM übergeben wir die Option **`-felf`** um den Assemblercode im ELF Objektformat abzulegen, welches von GCC auf Linux für Objektdateien verwendet wird. Die vom Makefile generierten Befehle sind:

```
nasm -felf -o build/calc.o calc.asm
gcc -m32 -c -o build/read.o read.c
gcc -m32 -o build/main build/calc.o build/read.o
```

2. Verlinkung von NASM und C Code

Im C-Header “read.h” des Rahmenprogramms wird die Funktion **`extern void calc(int*, float*, float*, float*, float*)`** deklariert, welche bei der Verlinkung in allen gegebenen Objekt- und Bibliotheksdateien gesucht wird. Die Implementierung dieser Funktion ist in “calc.asm” zu finden.

```
section .text
global calc
calc:
    ; Assembler Code der Funktion calc() ...
```

Unter Linux werden Funktionsnamen in C nicht “dekoriert”. Auf anderen Plattformen, wie Mac OS oder MS Windows, sucht der C-Linker aufgrund der Namensdekoration nach einem anderen Namen als **`calc`** (etwa **`_calc`** auf Mac OS) weshalb die Verlinkung auf diesen Plattformen fehlschlägt.

3. Details zur Assembler Implementierung

3.1 Makros

Zur Vereinfachung von einigen üblichen Aufgaben und des Debuggens wurden einige häufig verwendeten Makros eingeführt.

`stack_pop(n)`

Dieses Makro entfernt die obersten *n* Bytes vom Stack. Effektiv werden *n* Bytes zum Stack-Pointer Register **`esp`** addiert.

`stack_reserve(n)`

Dieses Makro reserviert *n* Bytes auf dem Stack. Effektiv werden *n* Bytes vom Stack-Pointer Register **`esp`** subtrahiert.

`farg(n)`

Mit diesem Makro wird die Adresse des *n*-ten Funktionsarguments berechnet. Es wird angenommen, dass jedes Argument exakt 4 Bytes belegt. Die Funktion muss vorher mit dem Befehl **`enter`** eingeleitet werden, um das Stack-Base-Pointer Register **`ebp`** auf die richtige Adresse zu setzen.

`fvar(n)`

Dieses Makro berechnet die Adresse der *n*-ten lokalen Variable auf dem Stack. Es wird angenommen, dass jede Variable exakt 4 Bytes belegt. Die Funktion muss vorher mit dem Befehl **`enter`** eingeleitet werden, um das Stack-Base-Pointer Register **`ebp`** auf die richtige Adresse zu setzen.

`do_printf dataname, text`

Mit diesem Makro wird die C-Funktion **`printf()`** aufgerufen. Dabei wird als letztes Argument der gegebene **`text`** auf den Stack gepusht. Die Angabe von **`dataname`** muss innerhalb eines globalen Labels einzigartig sein, da unter dem angegebenen Namen der **`text`** gespeichert wird. Ein Line-Feed Character und Null-Terminator wird automatisch hinzugefügt.

Beispiel

```
print_xx:  ; void print_xx(int x)
  enter
  mov eax, [farg(0)]
  imul eax, 2
  push eax
  do_printf _txt, “print_xx: %d”
  pop eax
  leave
  ret
```

3.2 Dichte-Berechnung

Zu Beginn der **`calc()`** Funktion wird die Adresse **`int* nlines`** derefenziert und in die gleiche Stack-Adresse verschoben, um zu simulieren, als wäre ein einfacher Integer als Argument übergeben worden.

Algorithmus:

```
for (int i = 0; i < nlines; ++i)
  results1[i] = data1[i] / data2[i];
```

Registerbelegung:

`eax`	Adressen-Offset berechnet aus dem Schleifenindex * 4
`ecx`	Aktueller Schleifenindex **`[0, eax)`**
`esi`	Akkumulator zur Berechnung der Addresse eines Floats
`st0`	Aktueller Gewichtswert aus **`data1`**

3.3 Sortierung

Die Funktion **`sort(int nlines, float* data)`** wird von **`calc()`** Aufgerufen, um **`results1`** aufsteigend zu sortieren.

Algorithmus:

```
for (int i = 0; i < nlines; ++i) {
  for (int j = 0; j < nlines; ++j) {
    if (data[j] < data[i]) {
      float temp = data[i];
      data[i] = data[j];
      data[j] = temp;
    }
  }
}
```

Registerbelegung:

`eax`	Adressen-Akkumulator
`ecx`	Aktueller äußerer und innerer Schleifenindex (temporär)
`st0`	Aktueller äußerer Wert
`st1`	Aktueller innerer Wert

Stackbelegung:

`[fvar(0)]`	Äußerer Schleifenindex
`[fvar(1)]`	Innerer Schleifenindex
`[fvar(2)]`	Adresse des äußereren Werts
`[fvar(3)]`	Adresse des inneren Werts

3.4 Metawert-Berechnung

Nachdem **`results1`** sortiert wurde, müssen nur noch die ersten und letzten 10 Werte vernachlässigt werden um den Durchschnitts-, Differenz-Minimal- und Differenz-Maximalwert zu berechnen.

Algorithmus:

```
results2[0] = 0.0;
for (int i = 10; i < nlines - 10; ++i)
  results2[0] += results1[i];
results2[0] /= (nlines - 20);
results2[1] = results1[nlines - 10] - results2[0];
results2[2] = results2[0] - results1[10];
```

Registerbelegung:

`eax`	Adressen-Akkumulator für **`results1`**
`ebx`	Adressen-Akkumulator für **`results2`**
`ecx`	Aktueller Schleifenindex **`[10, nlines - 10)`**
`edx`	Konstant **`nlines - 10`**
`st0`	Float-Akkumulator, später Durchschnittswert
`st1`	Zweiter Float-Operand

4. Ansätze zur Weiterentwicklung

Da die die Erstellung einer Kopie von **`results1`** recht aufwändig in Assembler zu implementieren wäre, und die Aufgabenstellung die Änderung der Reihenfolge der Ergebniswerte nicht ausschließt, wurde **`results1`** direkt sortiert um die Berechnung der Metawerte zu ermöglichen. Allerdings verliert das Programm damit seinen nutzen, da es nach der Ausführung nicht mehr möglich ist, einen Dichtewert auf das vorherige Gewicht-Volumen Wertepaar zurückzuführen. Hier wäre ein Ansatzpunkt, entweder einen Alternativansatz zur Berechnung der Metawerte zu finden (siehe Lösungsansatz A in der Spezifikation) oder eine Kopie von **`results1`** anzulegen, um diese zu sortieren.