Technische Universität München
Fakultät für Informatik
Rechnerarchitektur-Praktikum
SS 2018

ASSEMBLER POLYNOME

Entwicklerdokumentation

Bearbeitet von:
Oleg Patrascu
Matthias Unterfrauner

07.07.2018

INHALT

1. Kompilierungsprozess 3
2. Verlinkung von Assembly und C Code 3
3. Projektbeschreibung 4
4. Tests für Sonderfälle 7
5. Erweiterungen 8
   4.1 Relativer Fehler 8
   4.2 Grad des Polynoms 9
6. Ressourcen für die Weiterentwicklung 9

1. Kompilierungsprozess

Der Assemblercode und das Makefile wurden für eine 64-Bit Linux Platform entworfen.

Das C-Rahmenprogramm wird mit GCC mit der Option -m64 kompiliert, um eine 64-Bit Kompilierung zu zwingen.

Für NASM übergeben wir die Option --felf um den Assemblercode im ELF Objektformat zu generieren, welches von GCC auf Linux für Objektdateien verwendet wird.

Die vom Makefile generierten Befehle sind:

```
nasm -f elf64 src/norm.asm

gcc -Wall -m64 -std=gnu11 -I headers -fPIC

    test/test.c src/utils.c src/norm.o -o test/test.out

gcc -Wall -m64 -std=gnu11 -I headers -fPIC

   -o main.out src/main.c src/utils.c src/norm.o
```

-I headers damit alle Dateien aus dem headers Verzeichnis eingebunden sind.
-fPIC für Position Independent Code (für „flexible" Adressen).
-std=gnu11 für gnu11 Standard.

2. Verlinkung von Assembly und C Code

Im C-Header **polynom.h** wird die Funktion

```
extern float norm(struct polynom *input, struct polynom *output) asm („norm");
```

deklariert, welche bei der Verlinkung in allen gegebenen Objekt- und Bibliotheksdateien gesucht wird. Die Implementierung dieser Funktion ist in **norm.asm** zu finden.

```
section .text

global norm

norm:

; Assembler Code der Funktion norm() ...
```

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

3. Projektbeschreibung

Projektstruktur:

```
readme
root
    makefile

    headers
      polynom.h
      utils.h
    src
      main.c
      norm.asm
      utils.c
    test
      test.c
      polynom.txt
```

Datei	Beschreibung
main.c	Ermöglicht eine Interaktive Modi zwischen dem Benutzer und Programm und zudem erleichtert das Testen.
norm.asm	Enthält Funktionen die das Input prüfen und das Output-Polynom berechnen, diese Datei ist auch eine gute Dokumentation weil jede Source Code Zeile ausführlich kommentiert ist.
utils.c, utils.h	Enthalten nützliche Funktion um das Polynom schöner darzustellen.
polynom.h	Enthält die Signatur für die norm.asm Funktionen.
test.c	Enthält alle mögliche Testfälle und prüft auf die Richtigkeit des Programms.
polynom.txt	Eine C Implementierung die die norm.asm Funktionen spiegelt, dient nur um das Programm besser zu verstehen.
makefile	Alle Regeln für eine bequeme Kompilierung des Projekts.

Folgender C Algorithmus wurde in Assembly implementiert und liegt in der **norm.asm** Datei:

```
/*
struct polynom {

 int iDegree;
float* p_fCoefficients ;

};
*/

bool isCoeffZero(struct polynom* p, unsigned int i) {

  return fabs(p->p_fCoefficients[i]) < 0.005f;
}

bool areAllCoeffZero(struct polynom* p) {

  for(int i = 0; i <= p->iDegree; ++i) {

    if (isCoeffZero(p, i) == false) return false;
}

  return true;
}
```

```
float norm(struct polynom *input, struct polynom *output) {

 if(areAllCoeffZero(input) || input->iDegree < 0)

    return 0;


 int degree = input->iDegree;

 output->iDegree = degree;

 float a_N = input->p_fCoefficients[degree];


 for (int i=0; i <= degree; ++i) {

       output->p_fCoefficients[i] =  input->p_fCoefficients[i] / a_N;
 }

 return a_N;
}
```

Die Verantwortung um das User-Input zu prüfen, übernehmen die Funktionen:

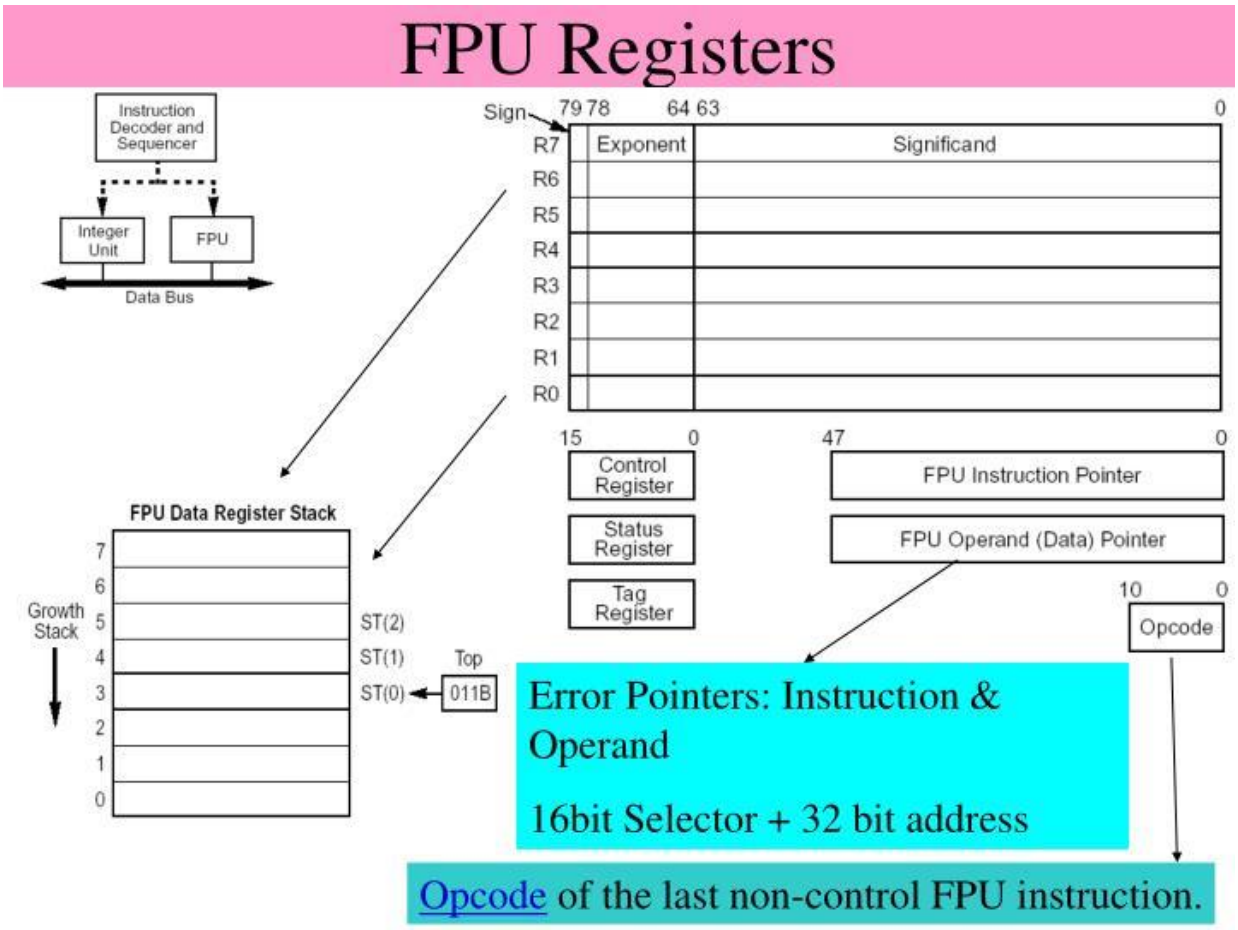
```
bool isCoeffZero(struct polynom* p, unsigned int i) // prüft ob der i-te Koeffizient 0 ist

bool areAllCoeffZero(struct polynom* p) // prüft ob jeder Koeffizient 0 ist
```

deswegen wurden sie auch getrennt als Subroutinen in Assembly implementiert (auch in der Datei **norm.asm**), und damit wird auch *float norm(struct polynom *input, struct polynom *output)* verständlicher und sauberer.

Die Assembly Implementierung ist sehr ausfürlich kommentiert (in der **norm.asm**) und braucht hier keine eigene Sektion.

Wie schon in der Spezifikation erwähnt, unsere Lösung ist einfach, da wir nur durch ein Speicherbereich durchlaufen und damit eine O(N) Laufzeit erreichen.
Es werden keine SIMD – Instruktionen verwendet , obwohl die deutlich unsere Implementierung beschleunigen können aber damit auch die Komplexität des Assembly-Programms.

Zuerst man muss verstehen wie der FPU-Stack überhaupt funktioniert und dann soll man die **norm.asm** anschauen.



Eine Liste mit allen FPU Befehlen, findet man unter
http://www.website.masmforum.com/tutorials/fptute/fpuchap8.htm#fdiv.

4. Tests für Sonderfälle

Alle Tests befinden sich in der **test.c** Datei, folgende Fälle sind von uns abgedeckt.

Fall 1

Wenn das Programm den Polynomgrad verlangt, man kann z.b. 3 als Grad eingeben.

Ein gültiger Polynom mit Grad 3 sieht so aus : *5 + 2x + 3x^2 + 7x^3*
aber der Benutzer kann auch folgendes eingeben : *5 + 2x + 0x^2 + 0x^3*

In diesem Fall, es ist sinnvoll die *iDegree* Variable von *3* auf *1* zu setzen. Dieser Fall impliziert auch den

Fall 1.1

Der benutzer kann alle Koeffizienten auf 0 setzen und trotzdem ein Grad größer als 0 haben, dann kriegt er auch eine Fehlermeldung, dass solche Polynome nicht gültig sind.

Fall 2

Ein anderer Sonderfall ist wenn der Benutzer einen negativen Grad eingibt wie z.B. *-2*, die Funktion *norm*, wird einfach nichts in das Output-Polynom schreiben und es wird auch kein Speicher für den Output-Polynom reserviert, d.h. wenn wir auf irgendwelche Felder des Output-Polynoms zugreiffen, wie *iDegree* oder *p_fCoefficients* stoßen wir auf „Memory Garbage".

5. Erweiterungen

4.1 Relativer Fehler

Aktuell das Programm nutzt einen relativen Fehler von 0.005f.

Angenommen, dass nach einer Division unser Ergebniss 0.0009 ist, es wird als 0.0 interpretiert.
Man könnte noch den relativen Fehler als Eingabe des Benutzers anfordern.

Vorschlag
Entweder eine globale Variable wie *relative_error* oder ein zusätzliches *relative_error* Parameter hinzuzufügen für alle Funktionen die die Polynome berechnen , nämlich:

Alte Signaturen:
*float norm(struct polynom *input, struct polynom *output);*

bool isCoeffZero(struct polynom p, unsigned int i);*

bool areAllCoeffZero(struct polynom p);*

Neue Signaturen:
*float norm(struct polynom *input, struct polynom *output, float rel_error);*

bool isCoeffZero(struct polynom p, unsigned int i, float rel_error);*

bool areAllCoeffZero(struct polynom p, float rel_error);*

4.2 Grad des Polynoms

Wenn das Programm den Polynomgrad verlangt, man kann z.b. 3 als Grad eingeben.

Ein gültiger Polynom mit Grad 3 sieht so aus : *5 + 2x + 3x^2 + 7x^3*
aber der Benutzer kann auch folgendes eingeben : *5 + 2x + 0x^2 + 0x^3*

In diesem Fall, es ist sinnvoll die *iDegree* Variable von *3* auf *1* zu setzen.
Zurzeit das Programm verlangt eine neue Eingabe weil sie ungültig war.

Vorschlag :
Eine neue Funktion, die den Grad falls notwendig dekrementiert.

```
bool isCoeffZero(struct polynom p, unsigned int i) {
  return fabs(p.p_fCoefficients[i]) < 0.005f; //rel error
}

void decrementDegree(struct polynom p) {
++p.iDegree;

while(isCoeffZero(p, --p.iDegree;));
}
```

6. Ressourcen für die Weiterentwicklung

Sehr hilfreiche Links die während der Implementierung verwendet wurden und dienen als **Hauptressourcen** für eine mögliche Weiterentwicklung :

https://en.wikipedia.org/wiki/X86_calling_conventions#List_of_x86_calling_conventions

http://www.website.masmforum.com/tutorials/fptute/fpuchap8.htm#fdiv

http://www.cs.virginia.edu/~evans/cs216/guides/x86.html

https://msdn.microsoft.com/en-us/library/7kcdt6fy.aspx