Freie Universität Berlin

Assembler (NASM) Crashkurs

von Sönke Schmidt

Berlin, 4.11.2015

Meine Webseite: http://www.soenke-berlin.de

Assembler – NASM – Was ist das?

Assembler nach Wikipedia:

„Ein Assembler ist ein Programmierwerkzeug, das ein in maschinennaher Assemblersprache geschriebenes Computerprogramm in Maschinensprache (auch Maschinencode oder Nativer Code genannt) übersetzt. Assembler wird häufig auch als Synonym für Assemblersprache benutzt.“

https://de.wikipedia.org/wiki/Assembler_%28Informatik%29

⇒ Sehr nah an Maschinensprache

Maschinensprache für Mensch (quasi) nicht lesbar; Assembler schon

Assembler – NASM – Was ist das?

⇒ Sehr nah an Maschinensprache

– Maschinensprache besteht aus einer Folge von Bytes (die sowohl Befehle als auch Daten repräsentieren)

– Maschinensprache für Mensch (quasi) nicht lesbar; Assembler schon

  – Mnemonics

– NASM: Frei verfügbarer Assembler für x86- und x64-Architekturen

  Ist an die Syntax von Intels ASM86 angelehnt.

Tools zum Programmieren unter Windows

- **Notepad++**
  - Zum schreiben des Quellcodes

- **FileZilla (für sFTP)**
  - Zum Hochladen der Dateien

- **PuTTY (für SSH)**
  - Zum Compilieren und Ausführen am Uni Rechner

Verbinden zu:

andorra.imp.fu-berlin.de

Da Abgaben vermutlich unter andorra (x64) funktionieren müssen, ist es besser gleich dort zu kompilieren und zu testen. Außerdem erspart Ihr euch die NASM Win Installation und sonstige Abhängigkeiten zu berücksichtigen…

Demo

Demo-Zeit

Putty und FileZilla

C Wrapper

- Ein- und Ausgabe wäre mit Assembler anstrengend und aufwendig.

  Zur Vereinfachung darum einen c Wrapper einsetzen, der sich darum kümmert.

Demo

Demo-Zeit

C Wrapper

ASM Register

- Return Wert steht in rax (wird mit "ret" übergeben)

- Übergebene Argumente sind in

  rdi, rsi, rdx, rcx, r8, r9, und dann auf dem Stack (das ist die üblich Reihenfolge)

Übrigens: Register unterscheiden sich zwischen x86 und x64! Was eax bei x86 ist, ist rax bei der x64 Architektur.

Wichtigste Befehle

Mnemonic	Purpose
mov dest, src	Move data between registers, load immediate data into registers, move data between registers and memory.
call *func*	Push the address of the next instruction and start executing func. For local functions, you don't have to say anything special. For functions defined in C/C++, say "extern *func*" first.
Ret	Pop the return program counter, and jump there. Ends a subroutine.
add dest, src	dest = dest + src
mul *src*	Multiply rax and *src* as unsigned integers, and put the result in rax. High 64 bits of product (usually zero) go into rdx.
imul *dest,src*	*dest=dest*src*
div *src*	Divide rax by *src*, and put the ratio into rax, and the remainder into rdx. Bizarrely, on input rdx must be zero, or you get a SIGFPE.
jmp *label*	Goto the instruction *label*:. Skips anything else in the way.
cmp *a,b*	Compare two values. Sets flags that are used by the conditional jumps (below).
jl *label*	Goto *label* if previous comparison came out as less-than. Other conditionals available are: jle (<=), jeq (==), jge (>=), jg (>), jne (!=), and many others. Also available in unsigned comparisons: jb (<), jbe (<=), ja (>), jae (>=).

siehe: https://www.cs.uaf.edu/2006/fall/cs301/support/x86_64/

NASM kompilieren und ausführen

Auf der Konsole ausführen (bei Windows in Putty) :

```
nasm -f elf64 name.asm -o name.o

gcc -o ProgrammName nameC.c name.o

./ProgrammName
```

Das kompiliert die Assembler Datei und anschließend den C-Wrapper

Wichtig:

Es kompiliert hiermit zwar, aber wenn euer Tutor etwas anderes angegeben hat, dann kompiliert es besser so, wie euer Tutor es gesagt hat (Bsp. eines zusätzlichen Flags wäre: gcc −std=c99 …)

Demo

Demo-Zeit

Assembler programmieren und kompilieren

(Beispiele mit mul, imul, div und c Wrapper)

Fragen?

Fragen?

Demo

Demo-Zeit

Dann machen wir jetzt noch ein Beispiel mit „if – else“

Und ein kleines Projekt / Programm eurer Wahl

Vielen Dank für die Aufmerksamkeit!