Übungen X86-Assembler

Einleitung / Hilfestellungen

- Verbinden Sie sich via Cisco Anyconnect mit dem Lehre-Profil der DHBW Mosbach
- Benutzen Sie einen SSH-Client um sich auf der EPYC-Maschine einzuloggen
  IP: `10.50.15.120`, Benutzer: `vl-ra`, Passwort: `P8Ic9aTn`
- Legen Sie für Ihre Experimente einen eigenen Unterordner an mit `mkdir gruppenname`, Gruppenname ist frei wählbar.
  Wechseln Sie in den Unterordner mit `cd`, benutzen Sie diesen als Arbeitsordner
- Erzeugen Sie mit cp eine Arbeitskopie der Dateien helloworld.c und matrixtest.c in Ihr Arbeitsverzeichnis (`cp ../helloworld.c ../matrixtest.c .`)

Wichtige Linux-Befehle

- Directory-Listing: `ls`
- Verzeichnis wechseln via `cd`
- `mkdir, rmdir, rm`
- Hilfe zu Befehlen: `man befehl`, z.B. `man ls`
- Text-Editor: `nano` oder `vi`, `mousepad` für X11 (wenn Sie z.B. mobaxterm verwenden)
- Information über die CPU: `lscpu`
- Informationen über die NUMA-Konfiguration: `numactl -H`
- Kompilieren von C-Programmen: `gcc -o programm programm.c`
- Gnu Debugger: `gdb programm`
- `man <Befehl>`, z.B. `man ls` zeigt die Hilfeseite an.

Hilfe zu X86-Assembler

- https://bob.cs.sonoma.edu/IntroCompOrg-x64/book.html
- Kurzbeschreibung der wichtigsten Befehle:
  https://www.cs.virginia.edu/~evans/cs216/guides/x86.html

Hilfe zum Gnu Debugger GDB

Sobald Sie Ihr Programm kompiliert haben, können Sie es mit Hilfe des Gnu Debuggers starten:

```
gdb programm
```

Wichtige Kommandos für GDB:

- `b main` setzt einen Breakpoint in main
- `run` startet die Ausführung, `si` (`stepi`) eine einzelne Assembler-Instruktion
- `set disassembly-flavor intel` ändert die Disassemblierung, sodass die Reihenfolge der Operanden richtig ist (Intel-Stil) → mov ziel, quelle
- `tui enable` öffnet das text user interface
- `layout split` zeigt gleichzeitig Quellcode, Assembly & Kommandozeile an
- `layout regs` zeigt die Register dauerhaft an, `info registers` schreibt diese auf die Konsole, `info register rax` nur z.B. rax
- `p variablenname` zeigt den Inhalt einer Variablen an, z.B. `p i`
- `help` gibt Hilfe aus, `help befehl` zu einem konkreten Befehl
- `quit` beendet gdb

Kompilieren von C-Programmen

Zum Kompilieren eines Programms für das Debugging mit Hilfe von gdb:

```
gcc -O0 -g -fno-asynchronous-unwind-tables -fcf-
protection=none -o helloword helloworld.c
```

Damit werden Optimierungen und weitere für die Untersuchung von Assembler unnötige Funktionen ausgeschaltet.

Der Befehl erstellt eine Datei „helloworld“ mit Debugging-Symbolen, die direkt ausgeführt werden kann:

```
./helloworld
```

oder via Debugger:

```
gdb ./helloworld
```

Es kann auch eine kombinierte Ausgabe von Assembler und C-Sourcecode erzeugt werden:

```
gcc -O0 -g -Wa,-adhls -masm=intel -fno-asynchronous-
unwind-tables -fcf-protection=none helloworld.c >
helloworld.s
```

Dies erzeugt eine Datei helloworld.s mit Maschinencode, Assembler, sowie Hochsprache-Anweisungen in einer Datei. Diese ist allerdings durch die vielen Informationen auch schwerer lesbar.

Eine deutlich besser lesbare Version ohne Maschinencode und Hochsprache-Anweisungen lässt sich erzeugen via:

```
gcc -O0 -masm=intel -fno-asynchronous-unwind-tables -fcf-
protection=none -S helloworld.c
```

Damit wird die Datei helloworld.s mit reinem Assembler-Code erstellt.

Falls wie bei matrixtest.c CPU-spezifische Instruktionen verwendet werden, muss die ISA vorgegeben werden. Diese kann man entweder explizit durch eine konkrete Auswahl wie „knl“ wählen, oder mit „native“ den GCC bitten, diese passend zur aktuellen CPU zu setzen:

```
gcc -O0 -Wall -march=native -fno-asynchronous-unwind-
tables -fcf-protection=none -g -o matrixtest matrixtest.c
```

Aufgabe 1

1a

Loggen Sie sich in das System ein. Finden Sie heraus, was es für eine Architektur ist, welche CPU im Einsatz ist, wieviele Cores die Maschine besitzt (sowohl SMT als auch die Anzahl echter Cores).

1b

Kompilieren Sie das Programm `helloworld.c` und erzeugen Sie eine Assembler-Ausgabe.

Was sind die Unterschied im Vergleich zu MIPS-Assembler? Sie können den gdb verwenden, um den Code Instruktion für Instruktion auszuführen.

Ändern Sie die Datentypen der Variablen `i`, `j` und `k` auf `int8_t`, sowie `int64_t`. Was ändert sich am Assembler-Code? Was ändert sich, wenn die Variablen mit dem Schlüsselwort `register` deklariert werden?

Wie ist der Zugriff auf unterschiedliche Datengrößen im X86 realisiert? Welche Unterschiede beim Umgang mit dem Hauptspeicher fallen im Vergleich zu MIPS auf?

1c

Fügen Sie eine Methode `int hello()` zu (diese kann einmal leer sein, bzw. lokale Variable enthalten; geben Sie einen beliebigen int-Wert zurück). Generieren Sie die Assembler-Ausgabe hierzu und erläutern Sie die hinzu gekommenen Instruktionen. Wie sieht der Prozeduraufruf bei X86 aus? Was passiert, wenn eine lokale Variable `int i` angelegt wird? Was, wenn ein Array `char c[32] = {0}` deklariert wird?

Hilfe: ergänzen Sie hierfür die gcc-Direktiven um das Argument `-fno-stack-protector` um einfacheren Code zu generieren.

Was geschieht, wenn die Methode `hello` einen Parameter bekommt? Wie sieht die Aufrufkonvention bei X86 aus?

1d

Kompilieren Sie das Programm mit dem char-Array und der Option `-fstack-protector` anstatt `-fno-stack-protector`. Erläutern Sie die Bedeutung der Zeilen

```
mov  rax, QWORD PTR fs:40
mov  QWORD PTR -8[rbp], rax
```

sowie des zugehörigen Codes am Ende der Prozedur (es kann je nach Compiler & Programm auch ein anderes Register sein):

```
mov  rdx, QWORD PTR -8[rbp]
sub  rdx, QWORD PTR fs:40
je   .L2
call __stack_chk_fail@PLT
```

1e

Schreiben Sie ein kleines C-Programm, das eine Schleife von 0 bis 10 enthält. Generieren Sie das entsprechende Assembler-Listing daraus. Wie unterscheidet sich hier der bedingte Sprung im Vergleich zu MIPS-Assembler? Was ist die Bedeutung des rflags-Registers?

Aufgabe 2

Kompilieren Sie das Programm `matrixtest.c`. Hierzu müssen ggf. andere Parameter für gcc verwendet werden.

Testen Sie das Programm mit verschiedenen Matrixgrößen.

Diskutieren Sie eventuelle Geschwindigkeitsunterschiede zwischen den 3 dgemm-Implementierungen.

Gibt es eine Erklärung für die Laufzeitunterschiede der zwei trace-Varianten? Um mit größeren Matrizen zu arbeiten, können Sie auch den Code für die dgemm-Aufrufe auskommentieren.

Bewertung & Abgabe

Dokumentieren Sie Ihre Ergebnisse in einer PDF-Datei (erwartet: ca. 3-5 Seiten). Sie können in Gruppen von bis zu 3 Personen abgeben. Für die Abgabe können bis zu 15 Extra-Punkte auf die Klausur angerechnet werden (selbstverständlich nur bis maximal 100 Punkte ;)).

Bitte laden Sie die Abgabe in den entsprechenden Moodle-Container hoch. Die Abgabe sollte selbstverständlich die Matrikelnummer der Gruppenmitglieder enthalten.