LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Aspekte der systemnahen Programmierung bei der Spieleentwicklung

Arbeitsblatt 2
20.10.2025 - 27.10.2025

T2.1 Setup und erste Schritte

Um ein einheitliches Setup zu gewährleisten, entwickeln wir auf dem Uni-Server der Rechnerhalle. Mittels einer SSH-Verbindung können Sie auch von Ihrem Laptop aus darauf zuzugreifen. Ansonsten benötigen Sie keine weiteren Tools.

Account Zur Anmeldung benötigen Sie Ihre Informatik-Kennung der Rechnerbetriebsgruppe (RBG) (dies ist *nicht* Ihre TUM-Kennung!). Sollten Sie Ihr Passwort vergessen haben, wenden Sie sich bitte an den RBG-Helpdesk. Weitere Informationen zur `lxhalle` finden Sie im RBG-Wiki[1]

Verbindung unter Windows

Seit dem April-Update 2018[2] unterstützt auch Windows *OpenSSH*[3]. Stellen Sie hierzu sicher, dass der *OpenSSH*-Client aktiviert ist.

1. Öffnen Sie die Windows-Einstellungen und navigieren Sie zu *Programme / Optionale Features verwalten*.
2. Falls Sie in der Liste den Punkt *OpenSSH-Client* nicht sehen, klicken Sie auf *Features hinzufügen*. Wählen Sie dann den *OpenSSH-Client* aus und installieren Sie ihn.

Verbindung unter Windows (OpenSSH), macOS, Linux, BSD, ...

1. Öffnen Sie das Terminal Ihres Vertrauens.
2. Verbinden Sie sich per `ssh username@halle.in.tum.de`

   Username: *Ihre Informatik-Kennung, ohne `@in.tum.de`*
   Password: *Ihr Passwort zu der Informatik-Kennung*
3. Beim ersten Verbinden werden Sie gefragt, ob Sie der Verbindung vertrauen möchten. Überprüfen Sie den angezeigten Fingerabdruck (siehe RBG-Wiki); stimmen diese überein, akzeptieren Sie den Key durch das Eintippen von *yes*.
4. Sie sind nun mit der Rechnerhalle verbunden.

[1] `https://wiki.in.tum.de/Informatik/Helpdesk/Ssh`
[2] Bei früheren Windows-Versionen können sie auf das Tool *PuTTY* zurückgreifen.
[3] `https://docs.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse`

T2.2 Verwendung von Materialien

In dieser und den kommenden Wochen werden wir Ihnen für einige der Programmieraufgaben Materialvorlagen bereitstellen, die Ihnen das bearbeiten der Aufgaben erleichtern. Diese enthalten üblicherweise ein Rahmenprogramm, welches grundlegende I/O-Funktionalitäten bereitstellt, und ein `Makefile`, welches Anweisungen zum Kompilieren des Programms enthält.

In dieser Aufgabe werden wir mithilfe der Kommandozeile[4] die Assembler-Funktion der Aufgabe *Abs* in ein Rahmenprogramm einsetzen.

1. Laden Sie die Vorlage herunter:

   ```
   wget https://asp.caps.in.tum.de/m/abs.tar
   ```

2. Extrahieren Sie die Dateien aus dem Tar-Archiv:

   ```
   tar xf abs.tar
   ```

3. Wechseln Sie in den neu angelegten Ordner `abs`: `cd abs`

4. Führen Sie den Befehl `ls` aus und vergewissern Sie sich, dass sich in diesem Ordner die Dateien `abs.c`, `abs.S` und `Makefile` befinden.

5. Öffnen Sie nun die Datei `abs.c` mit einem Texteditor Ihrer Wahl[5]: `nano abs.c`

   Diskutieren Sie mit Ihrem Tutor die Bedeutung folgender Zeilen:

   - `#include <stdio.h>`
   - `long asm_abs(int n);`
   - `int main(int argc, char** argv)`
   - `printf("abs(%ld) = %ld\n", 5l, asm_abs(5l));`

6. Öffnen Sie nun die Datei `abs.S` und versuchen Sie, jede Zeile zu verstehen.

7. Kompilieren Sie das Programm mit `make` und führen Sie es aus: `./abs`

   Die Funktion sollte nun für alle Eingaben den Wert 0 zurückgeben.

8. Fügen Sie eine Implementierung der Funktion `asm_abs` ein. Kompilieren Sie das Programm erneut und prüfen Sie es auf Korrektheit.

   Der Assembler-Code kann der entsprechenden Programmieraufgabe aus Woche 1 und dem zugehörigen Lösungsblatt entnommen werden.

[4] Wir gehen davon aus, dass Sie sich selbstständig in die Verwendung einer Linux-Kommandozeile einarbeiten können. Ein paar Basis-Befehle sind hier genannt. Verwenden Sie `man <Befehl>` in der Kommandozeile um nähere Informationen zu den einzelnen Befehlen zu bekommen.

[5] Wir verwenden `nano`, es ist auch `vim` installiert, falls Sie dies bevorzugen sollten.

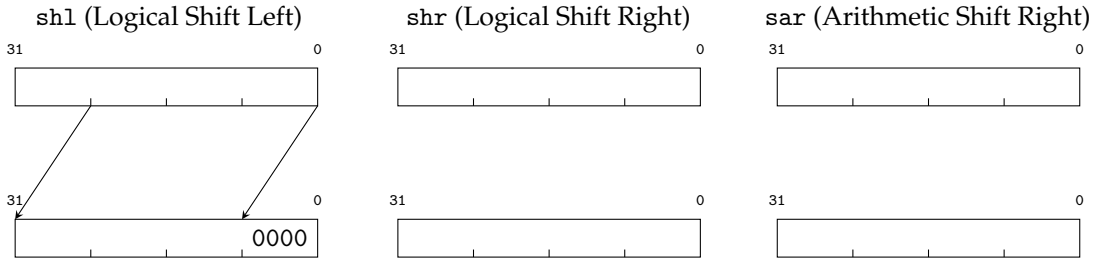
T2.3 Kurzprogramme

Programmieren Sie die folgenden kleinen Funktionen in x86-64-Assembler auf einem Blatt Papier oder in einem digitalen Notizbuch.

1. Der Wert des Registers `ax` soll verdoppelt werden.
2. Das Registers `rax` soll den Konstanten Wert 0x1234 erhalten.
3. Alle Bits des Registers `eax` sollen invertiert werden.
4. `rcx = (eax > edx ? 1 : 0)`

S2.1 Shifts

Skizzieren Sie die Funktionsweisen der unterschiedlichen Schiebeoperationen der x86-64 Architektur in den folgenden Grafiken. Welche mathematische Bedeutung haben die einzelnen Operationen?



S2.2 Weitere x86-64 Befehle

Ziehen Sie das offizielle *Intel Software Development Manual*[6], Volume 2, heran und erläutern Sie, was in den folgenden Kurzprogrammen geschieht.

Hinweis: Wir empfehlen für die Instruktionsreferenz einen PDF-Viewer welcher die in das PDF integrierte Gliederung anzeigt. Dies erleichtert es Ihnen deutlich, bestimmte Instruktionen zu finden. Alternativ können Sie auch eine inoffizielle (und ggf. falsche) Online-Referenz[7] verwenden. Wir verwenden im Praktikum die GNU/Intel-Syntax, bei dieser können mehrere Befehle durch `;` getrennt in eine Zeile geschrieben werden. Zeilenkommentare können hingegen mit `//` oder `#` eingeleitet werden, Blockkommentare mit `/* */`.

1. `add rbx,rdx; adc rax, rcx`

[6] `https://intel.com/sdm`
[7] z.B. `https://www.felixcloutier.com/x86/`

2. `imul edx, esi, 10` (Ist diese Multiplikation vorzeichenbehaftet?)
3. `xor eax, eax`
4. `xchg rax, rax`
5. `mov dword ptr [rbx], 0`
6. `add qword ptr [rdx], 1`
7. `lea rax, [rbx+rdx*8-18]`
8. `cmovl edi, r9d`
9. `popcnt rax, rdi`

S2.3 Speicherzugriff und Pointerarithmetik in C

Im Gegensatz zu Java erlaubt C einen direkten Zugriff auf den Speicher. Eine Adresse entspricht in C einem sogenannten *Pointer*. In diesem Zusammenhang ist folgende Syntax relevant:

C Source	Erläuterung
`int v = 5;`	Definition einer normalen Variable
`int* a;`	Deklaration eines Pointers der auf ein (mehrere) `int` zeigen kann
`a = &v;`	Der Pointer wird auf die Adresse von `v` gesetzt
`int q = *a;`	Der Pointer wird *dereferenziert*, d.h. der Wert aus dem Speicher geladen
`*a = 2;`	Der Wert im Speicher auf den `a` zeigt wird auf 2 gesetzt

Zum Speichern mehrerer Werte gleichen Typs gibt es auch in C Arrays. Durch die Deklaration eines Arrays wird zu Ausführungsbeginn ein entsprechend großer Bereich im Speicher reserviert. Der Array-Name entspricht dabei der Adresse dieses Speicherbereichs.

Welche Werte nehmen die Variablen in folgendem Code-Ausschnitt an?

C Source	Wert der Variable
`int array[4] = { 10, 20, 30, 40 };`	`0xfffe1000` *(beispielhaft!)*
`int val1 = array[0];` `int val2 = array[1];`	

`int val3 = *array;`
`int val4 = (*array)+1;`
`int val5 = *(array+1);`	

`int* ptr1 = array;` `int val6 = *ptr1;`	
`int* ptr2 = &array[2];` `int val7 = *ptr2;`	
`int* ptr3 = array + 3;` `int val8 = *ptr3;`	

Hinweis:

Ihre Implementierung der folgenden Aufgaben darf nur die folgenden Register verwenden: `rax, rcx, rdx, rsi, rdi, r8-r11`.

P2.1 Min

Schreiben Sie in x86-64 Assembly eine Funktion `min`, welche das Minimum der vorzeichenbehafteten Werte in den Registern `rdi`, `rsi` und `rdx` bestimmt und in `rax` zurück gibt.

Aufgabentester: Nutzen Sie den Aufgabentester, um diese Funktion zu entwickeln.

P2.2 Gauß

Schreiben Sie in x86-64 Assembly eine Funktion `gauss`, welche die Summe der Zahlen von 1 bis `rdi` berechnet und das Ergebnis in `rax` schreibt.

Aufgabentester: Nutzen Sie den Aufgabentester, um diese Funktion zu entwickeln.

P2.3 Strlen

In dieser Aufgabe soll die Funktion `strlen` in Assembler implementiert werden. Ein String wird in C durch ein `char`-Array realisiert. Das Ende des Strings wird dabei durch das Nullzeichen `'\0'` markiert, welches den numerischen Wert 0 hat.

```
const char* text = "Hallo";
```

Die Funktion `strlen_asm` gibt die Länge eines Strings zurück, indem die Zeichen bis zum Nullzeichen gezählt werden.

Aufgabe: Implementieren Sie die Funktion `strlen_asm`. Die Startadresse steht in Register `rdi`, das Ergebnis wird in `rax` erwartet.

Vorlage: `https://asp.caps.in.tum.de/m/strlen.tar` – Für die Bearbeitung der Aufgabe müssen Sie lediglich die enthaltene Assembler-Datei bearbeiten.

Aufgabentester: Sie können den Aufgabentester nutzen, um diese Funktion zu entwickeln und abzugeben. Wir empfehlen, dass Sie die Aufgabe dennoch zunächst lokal mit der bereitgestellten Vorlage bearbeiten.

Q2.1 Quiz (siehe Praktikumswebsite)