LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Aspekte der systemnahen Programmierung bei der Spieleentwicklung

Arbeitsblatt 1
13.10.2025 - 20.10.2025

Aufgabentypen:

- Typ **T**: Aufgaben zur Bearbeitung während des Tutoriums (beginnend ab Woche 2).
- Typ **S**: Aufgaben zum Selbststudium, welche meist durch Videos begleitet werden.
- Typ **P**: Programmieraufgaben, die mithilfe des Aufgabentesters oder (auf späteren Blättern) mit bereitgestellten Materialien bearbeitbar sind. Zwecks Notenbonus bewertete Programmieraufgaben werden *nur über den Aufgabentester* angenommen und bewertet!
- Typ **Q**: Quizze auf der Praktikumswebsite.
- Typ **X**: Zusatzaufgaben, die das Verständnis erweitern, aber nicht prüfungsrelevant sind.

S1.1 Binärsystem und Zweierkomplement

Vervollständigen Sie folgende Tabelle, wobei die Binär-/Hexadezimalrepräsentation auf 8 Bit beschränkt sein und für negative Zahlen das Zweierkomplement verwendet werden soll.

Binär	Hexadezimal	Dezimal (mit Vorzeichen)	Dezimal (kein Vorzeichen)
0111 1111			
1111 1111			
0010 1010			
1000 0000			

S1.2 Grundlagen der x86-64-Architektur

In diesem Praktikum wird die x86-64 Architektur behandelt, welche die 64-Bit-Erweiterung der x86-Architektur darstellt. Gehen Sie foldende Instruktionen durch und stellen Sie sicher, dass Sie deren Funktionsweise nachvollziehen können. Beachten Sie insbesondere die Unterschiede zwischen `imul` mit einem Operanden und `imul` mit mehreren Operanden sowie etwaige Beschränkungen bei der Verwendung von *Immediate-Operanden*

`mov`	`add`	`sub`
`(i)mul`	`(i)div`	`jmp`
`cmp`	`jcc`	`and`
`or`	`xor`	`not`

1. rax = 32 Bit + eax Bei rsi ist bei den letzten 16 Bits keine Aufteilung zwischen higher und lower
eax = 16 Bit + ax (bei rax ah und al), sondern es haben nur die letzten 8 Bits eine
ax = ah + al = 8Bit · 2 Bezeichnung - sil.

1. Aus welchen Teilregistern besteht das Register `rax`? Wie unterscheidet sich die Aufteilung des Registers `rsi` von der Aufteilung von `rax`?

2. Was berechnet die Instruktion `imul rdx, r12`? Handelt es sich um eine vorzeichenbehaftete oder vorzeichenlose Multiplikation? Es wird rdx = rdx * r12 berechnet.
Es ist nicht-erweiternd, da dst & src gleich groß. Bei der nicht-erweiternden Mul. gibt es
3. Ist die Instruktion `add rax, edx` codierbar? Wenn nein, warum nicht? keinen Unterschied zwischen
nicht codierbar: dst, src müssen bei add gleich groß sein. signed/unsigned.
4. Warum kann die Instruktion `sub eax, 0x80000000` codiert werden, die Instruktion `sub rax, 0x80000000` hingegen nicht?
eax ⇒ Zahl wird als 32 Bit interpretiert mit 2er-Komplement ⇒ 0x8000 0000 = -2^{31}
rax ⇒ Zahl wird als 64 Bit interpretiert ⇒ 0x8000 0000 = 2^{31} ⇒ nicht mehr darstellbar

P1.1 Kleiner Gauß

Ziel dieser Aufgabe soll es sein, die Funktion `gauss100`, welche die Summe der natürlichen Zahlen von 1 bis n (mit $0 \leq n \leq 100$) berechnet, in x86-64 Assembly zu implementieren. Der Parameter n wird im Register `rdi` übergeben, das Ergebnis wird am Ende der Berechnung in `rax` erwartet.

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

1. Erinnern Sie sich an die *Gaußsche Summenformel*, mit welcher der zu berechnende Wert leicht bestimmt werden kann. Vollziehen Sie nach, weshalb diese Formel korrekt ist.

$$\sum_{k=0}^{n} k = \frac{n\,(n+1)}{2}$$

2. Kopieren/verschieben Sie zunächst den Parameter von `rdi` in das Register `rax`. Welche Instruktion bietet sich an?

3. Addieren Sie nun die Konstante 1 auf das Register `rax`.

4. Multiplizieren Sie nun den Wert des Registers `rax` mit dem Wert von Register `rdi`. Nutzen Sie hierbei eine *nicht-erweiternde* Multiplikation.

5. Als letzter Berechnungsschritt muss das Ergebnis der Multiplikation durch 2 dividiert werden, verwenden Sie hierzu die Instruktion `div`. Beachten Sie, dass die Instruktion `div` bestimmte Register implizit verwendet und keine *Immediate-Operanden* unterstützt.

6. Stellen Sie sicher, dass sich das Ergebnis im Register `rax` befindet und beenden Sie die Funktion mit der Instruktion `ret`.

P1.2 Abs

Ziel dieser Aufgabe soll es sein, die Funktion `asm_abs`, welche den Betrag einer Ganzzahl zurückgibt, in x86-64 Assembly zu implementieren. Der Parameter wird im Register `rdi` übergeben und das Ergebnis wird im Register `rax` erwartet.

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

1. Prüfen Sie zunächst, ob der Parameter negativ ist und realisieren Sie eine Fallunterscheidung. Verwenden Sie hierzu die Instruktionen `cmp` und `jcc`. Welche Sprungbedingung bietet sich an?
2. Definieren Sie ein lokales Label als Zielort für den bedingten Sprung.
3. Falls der Parameter negativ ist, berechnen Sie $0 - rdi$; andernfalls brauchen Sie keine Berechnung durchzuführen.
4. Schreiben Sie am Ende das Ergebnis in das Register `rax` und beenden Sie die Funktion mit der Instruktion `ret`.
5. Überlegen Sie sich, welche Randfälle existieren. Verhält sich Ihr Programm in allen Fällen wie erwartet?

Q1.1 Quiz (siehe Praktikumswebsite)