

Bitte beachten Sie die allgemeinen Hinweise auf Übungszettel 1

Aufgabe 1: Zahlenbasen

(a) Rechnen Sie die folgenden Zahlen vom Dezimalsystem ins Binär- und Hexadezimalsystem um:

- 113_{10}
- $257, 23_{10}$

siehe Seite 3

(b) Rechnen Sie die folgenden Zahlen vom Binärsystem ins Dezimalsystem um:

- 101011010_2 $2^1+2^3+2^4+2^6+2^8 = 346$
- $10010, 1001_2$ $2^4+2^5+2^6+2^7 = 18,5625$

(c) Rechnen Sie die folgenden Zahlen vom Hexadezimalsystem ins Dezimalsystem um:

- $0xE37A$ $\Rightarrow 10 \cdot 16^0 + 7 \cdot 16^1 + 3 \cdot 16^2 + 14 \cdot 16^3 \Rightarrow 58234$
- $0x39B, 2D8$ $\Rightarrow 11 \cdot 16^0 + 9 \cdot 16^1 + 3 \cdot 16^2, 2 \cdot 16^2 + 13 \cdot 16^3 \Rightarrow 923, 177734375$

(d) Rechnen Sie folgende Binärzahl ohne Umweg über das Dezimalsystem direkt ins Hexadezimalsystem um.

- 0101101010110010111_2 $\Rightarrow 2 \ 13 \ 5 \ 9 \ 7 \Rightarrow 2D597$

(e) Rechnen Sie die folgenden Zahlen ins Dezimalsystem um:

- $HODOR_{26}$ $\Rightarrow 8+15+4+15+18 \Rightarrow 60$
- $STAR, WARS_{36}$ $\Rightarrow 19+20+1+18, 23+1+18+19 \Rightarrow 58,61$

Der Lösungsweg soll stets erkennbar sein.

sei das Alphabet aufsteigend von A nach Z mit 1 bis 26 nummeriert: A=1, B=2, C=3, ..., Z=26

Aufgabe 2: Fibonacci Zahlen

Auf dem letzten Zettel haben Sie die Wiederholung von Instruktionen durch Sprünge kennengelernt. Diese Art der Wiederholung wird *Iteration* genannt. Die andere Art der Wiederholung ist *Rekursion*. Machen Sie sich mit Rekursion auf Assemblerebene und dafür mit dem Callstack vertraut (Befehle: PUSH, POP, CALL, RET).

Die Fibonacci-Zahlen seien wie folgt definiert:

```
f(1) = 1;  
f(2) = 1;  
f(n) = f(n-1) + f(n-2); // für n>2
```

Im KVV wird Ihnen ein C-Framework gestellt, welches unterschiedlich implementierte Fibonacci-Funktionen gegeneinander vergleicht. Schreiben Sie in Assembler eine iterative und eine rekursive Fibonacci-Funktion (siehe Pseudocode unten). Linken Sie die Funktionen mit dem C-Framework. Ihre Funktionen müssen dafür folgende Signatures haben:

```
uint64_t asm_fib_it(uint64_t n);  
uint64_t asm_fib_rek(uint64_t n);
```

Erklären Sie die Zeitunterschiede sowohl zwischen den rekursiven und iterativen Funktionen als auch zwischen den C und Assembler Funktionen. Warum wird Rekursion überhaupt benötigt?

- Pseudocode iterativ:

```
asm_fib_it(n) {  
    x = 0;  
    y = 1;  
    k = 0;  
    while(n > 0) {  
        x = y;  
        y = k;  
        k = x + y;  
        n--;  
    }  
    return k;  
}
```

- Pseudocode rekursiv:

```
asm_fib_rek(n) {  
    if(n < 3) {  
        return 1;  
    } else {  
        return fib_rek(n-1) + fib_rek(n-2);  
    }  
}
```

Hinweis 1: Für das Erklären der Zeitunterschiede sollten Sie sich die Kompilate der C-Funktionen ansehen. Möglichkeiten um sich diese anzusehen sind "gdb <prog>", "objdump -d <prog>" oder "gcc -S fib.c".

Hinweis 2: Sie können beide Funktionen in eine Datei schreiben.

(a) Rechnen Sie die folgenden Zahlen vom Dezimalsystem ins Binär- und Hexadezimalsystem um:

- 113_{10}
- $257,23_{10}$

$$113 : 16 = 7 \quad \text{Rest } 1 \quad \uparrow$$
$$7 : 16 = 0 \quad \text{Rest } 7$$

$$113 = 71$$

$$257 : 16 = 16 \quad \text{Rest } 1 \quad \uparrow$$
$$16 : 16 = 1 \quad \text{Rest } 0$$
$$1 : 16 = 0 \quad \text{Rest } 1$$

$$257 = 101$$

$$0,23 \cdot 16 = 3,68 \quad z_1$$

$$0,68 \cdot 16 = 10,88 \quad z_2$$


$$0,88 \cdot 16 = 14,08 \quad z_3$$

$$0,08 \cdot 16 = 1,28 \quad z_4 \quad \downarrow$$


$$0,23 \approx 0,3AE1 \quad \Rightarrow \quad 257,23 \Rightarrow 101,3AE1$$

$$113 : 2 = 56 \quad \text{Rest } 1 \quad \uparrow$$
$$56 : 2 = 28 \quad \text{Rest } 0$$
$$28 : 2 = 14 \quad \text{Rest } 0$$
$$14 : 2 = 7 \quad \text{Rest } 0$$
$$7 : 2 = 3 \quad \text{Rest } 1$$
$$3 : 2 = 1 \quad \text{Rest } 1$$
$$1 : 2 = 0 \quad \text{Rest } 1$$

$$113 = 1110001$$

$257 : 2 = 128$	Rest 1	
$128 : 2 = 64$	Rest 0	
$64 : 2 = 32$	Rest 0	
$32 : 2 = 16$	Rest 0	
$16 : 2 = 8$	Rest 0	
$8 : 2 = 4$	Rest 0	
$4 : 2 = 2$	Rest 0	
$2 : 2 = 1$	Rest 0	
$1 : 2 = 0$	Rest 1	

$$257 = 100000001$$

$0,23 \cdot 2 = 0,46$	Rest 0	
$0,46 \cdot 2 = 0,92$	Rest 0	
$0,92 \cdot 2 = 1,84$	Rest 1	
$0,84 \cdot 2 = 1,68$	Rest 1	
$0,68 \cdot 2 = 1,36$	Rest 1	
$0,36 \cdot 2 = 0,72$	Rest 0	
$0,72 \cdot 2 = 1,44$	Rest 1	

$$0,23 \approx 0011101$$

$$\Rightarrow 257,23 \Rightarrow 100000001,0011101$$

Erklären Sie die Zeitunterschiede sowohl zwischen den rekursiven und iterativen Funktionen als auch zwischen den C und Assembler Funktionen. Warum wird Rekursion überhaupt benötigt?

Rekursive Funktionen sind meist langsamer und verbrauchen mehr Speicher als Iterationen. Das liegt daran, dass sie sich immer wieder selbst aufrufen und ihre Ergebnisse zwischenspeichern. Erst nachdem die erste Rekursion abgeschlossen wurde (ein Return ausgegeben wurde) werden die Ergebnisse zusammengerechnet und der Speicher wieder entleert. Iterationen hingegen benutzen Loops, welche Speicherfreundlicher sind.

Dies ist auch an unseren Funktionen `asm_fib_it` und `asm_fib_rek` zu sehen. Je höher der eingegebene Wert (Anzahl an Rekursionen) desto Speicher- und Zeitaufwändiger die Ausführung der `asm_fib_rek` Funktion. Bei der `asm_fib_it` lässt sich dies im Vergleich nur minimal feststellen.

Die iterative C Funktion läuft meist minimal langsamer als unsere `asm_fib_it`. Das liegt vermutlich daran, dass wir direkt im Assembler code geschrieben haben und die C Datei erst von C nach Assembler umrechnen muss. Die rekursive C Funktion läuft jedoch, besonders bei einer hohen Eingabe um einiges schneller. Dort ist der C Code wahrscheinlich einfach besser kompiliert.

Rekursionen sind meist leichter zu kompilieren und bieten manchmal bessere Lesbarkeit. Iterationen sind schneller, dafür schwerer zu kompilieren und bieten manchmal schlechtere Lesbarkeit .