Ostbayerische Technische Hochschule Regensburg
Fakultät Informatik und Mathematik Studiengang: Technische Informatik

Prüfungsfach: Datenverarbeitungssysteme **Aufgabensteller:** **Prüfungstermin:** **Prüfungsdauer:** 90 Minuten	**Prüfungsteilnehmer** (Alle Angaben bitte in **Druckbuchstaben**) **Name:** ……..…….. **Vorname:** …… **Semester:** ……... **Matrikelnummer:** …… **Platznummer:** ……

Probeklausur

Aufgabe	**1**	**2**	**3**	**4**	**5**	**Summe**	**Note**
mögliche Punkte	45	16	15	6	18	100	
erreichte Punkte							

Hinweise:

- Die Prüfung besteht aus 5 Aufgaben auf 9 Blättern, die alle abzugeben sind.
- Bearbeiten Sie die Aufgaben direkt auf diesen Blättern.
- Schreiben Sie nicht mit Bleistift und nicht mit roten Stiften.
- Geben Sie zu jeder Aufgabe **nur eine** Lösung an.
- Erlaubtes Hilfsmittel: **ASM Instruction Sum x86-64 GAS (ausgedruckte Datei ohne Notizen)**

Wissensfragen (45 Punkte insgesamt)

1.1) (3 Punkte)

a) Aus wie vielen Bytes besteht jeweils ein **Kilobyte** und ein **KibiByte**?
Geben Sie die Ergebnisse in Form einer **Potenz** an:

1 KB = Byte

1 KiB = Byte

1.2) (3 Punkte)

a) Was bedeutet die Abkürzung **ALU**?

b) Geben Sie zwei Gruppen von Operationen an, die in der **ALU** durchgeführt werden:

1.3) (6 Punkte)

Gegeben ist folgender GNU-Assembler-Code:

```
movb $53,  %al
addb $91,  %al #  53 +   91  =  144 (signed   / dec)
               #  35 +   5B  =   90 (unsigned / hex)
```

Geben Sie mit Begründung an, welche der folgenden Flags nach der zweiten Zeile gesetzt sind. Hierbei bedeuten **1** = Flag gesetzt, **0** = Flag nicht gesetzt.

Flag	0 / 1	Begründung
Carry-Flag (CF)		
Overflow-Flag (OF)		
Sign-Flag (SF)		

1.4) (10 Punkte)

Gegeben ist folgende Funktionsdeklaration in C:

```
float * func(float, char, int, int *);
```

In welchen Registern würden Sie den Rückgabetyp und die Parameter speichern?
Achten Sie dabei auf die richtige Größe der Register.

Typ	Register
Rückgabe (`float *`)	
1. Parameter (`float`)	
2. Parameter (`char`)	
3. Parameter (`int`)	
4. Parameter (`int *`)	

1.5) (4 Punkte)

Gegeben ist folgender C-Code:

```
float arrF[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
int arrN[5] = {0};
const int N = 1000;

int main(){
      int n = 5;
      // ...
      return 0;
}
```

Ordnen Sie jeder der angegebenen Variablen **durch Ankreuzen** genau einem Eintrag in der folgenden Tabelle zu und beachten dabei insbesondere den Unterschied zwischen lokalen und globalen Variablen. **Jede Spalte darf nur ein Kreuz enthalten**:

	Stack	**.section**		
		.rodata	**.data**	**.bss**
`arrF`				
`arrN`				
`N`				
`n`				

1.6) (3 Punkte)

Geben Sie den Befehl an, mit dem man ein C-Programm **prog.c** unter Verwendung von **gcc** in ein GNU-Assembler-Programm **prog.s** übersetzt:

1.7) (5 Punkte)

Wie wird das Beenden eines GNU-Assembler-Programms codiert (**`return 0;`**), wenn Sie das Programm

a) mit **as**
b) mit **gcc**

übersetzen?

a)

b)

1.8) (4 Punkte)

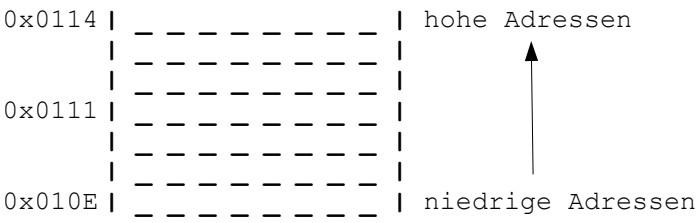
Wie lautet der Inhalt des Registers **rax** nach Ausführung des folgenden Befehls, wenn **rax** und **rdi** jeweils mit der Dezimalzahl **12** vorbelegt sind:

```
leaq 20(%rax, %rdi, 4), %rax
```

Inhalt von **rax** (in hexadezimaler Schreibweise): | _ _ _ _ | _ _ _ _ | _ _ _ _ | _ _ _ _ |

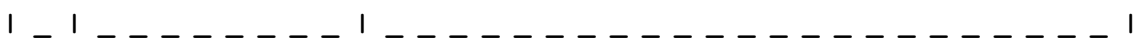
1.9) (4 Punkte)

Angenommen, der **Stack** hat eine Breite von **8 Bit**. Wie würde dann die **`unsigned int`**-Zahl **`0xABCD0123`** an der Adresse **`0x0111`** unter Verwendung von **Little Endian** dargestellt? Geben Sie die Zahl in **Binärdarstellung** an.



1.10) (3 Punkte)

Geben Sie die die Binärdarstellung von **INIFINITY** nach IEEE 754 bzgl. des C-Datentyps **`float`** an:



Platz für Nebenrechnungen

Praktischer Teil

2) (16 Punkte)

Schreiben Sie ein vollständiges GNU-Assembler-Programm, das in einer **for-Schleife** die Zahlen des (C-)Arrays **`long arrL[] = {200, 55, 13, -17, 24}`** aufaddiert und die Gesamtsumme mit der C-Funktion **`printf()`** in der Form "Result = xxx" ausgibt (xxx steht hierbei für das Ergebnis).

3) (15 Punkte)

Gegeben ist der folgende C-Code:

```
struct S {
  char arrC[3];
  int  n;
  short h;
  long l;
};

struct S svar={{'A', 'B', 'C'}, 5, 20, 8000};
```

Legen Sie die Strukturvariable **`svar`** in GNU-Assembler unter Verwendung von **natural alignment an** (**`.align 8`**). Speichern Sie anschließend in **main** die **20** aus **`svar`** im Register **`ax`**.

4) (6 Punkte)

Gegeben ist das folgende GNU-Assembler-Programm:

```
.section .data
.align 16
farr0:
  .float 0.0, 0.0, 0.0, 0.0
.align 16
farr1:
  .float 40.0, 30.0, 20.0, 10.0
.align 16
farr2:
  .float 80.0, 70.0, 60.0, 50.0

.section .text
.globl _start
.type _start, @function

_start:
  # Füllen Sie die nächsten drei Zeilen
  # Belegung von xmm0
  # Belegung von xmm1
  # Belegung von xmm2
  # shuffle
  vshufps $0x34, %xmm2, %xmm1, %xmm0

_end:
  # ...
```

Pos. 0	Pos. 1	Pos. 2	Pos. 3	Register
40	30	20	10	xmm1
80	70	60	50	xmm2
40	**30**	**50**	**80**	**xmm0**

← Ergebniszeile

a) Geben Sie hier die fehlenden drei Zeilen an: (3P)

b) Geben Sie die nötigen (Rechen-)Schritte an, die von der hex-Zahl **0x34** zu der **Ergebniszeile in xmm0** geführt haben: (3P)

5) (18 Punkte)

Gegeben ist folgender **C-Code**:

```
#include <stdio.h> // diese Zeile nicht übersetzen

void calc(long * result, long a, long b) {
  a <<= 2;
  b ^= 5;
  *result = a * b;
}

int main(){
  long result ;
  calc(&result, 0x4, 0x15);
  printf("result = %ld\n", result); // diese Zeile nicht übersetzen
  return 0;
}
```

a) Übersetzen Sie das Programm in GNU-Assembler. Beachten Sie dabei die Calling Conventions. (16P)
b) Was würde der **`printf`**-Befehl ausgeben? (2P)