Technische Universität Clausthal
Institut für Informatik
Prof. G. Kemnitz

02/03/2014

Laborübung 5 Rechnerarchitektur

Hinweise: Schreiben Sie die Lösungen, so weit es möglich ist, auf die Aufgabenblätter. Tragen Sie Namen, Matrikelnummer und Studiengang in die nachfolgende Tabelle ein und schreiben Sie auf jedes zusätzlich abgegebene Blatt ihre Matrikelnummer. Lassen Sie vorgeführte Experimente vom Betreuer gegenzeichnen. Für eine Bescheinigung der erfolgreichen Teilnahme sind in jeder einzelnen Laborübung mindestens 40% und insgesamt mindestens 50% der Punkte erforderlich.

Name	Matrikelnummer	Studiengang	Punkte	% von 20

Aufgabe 1:

Das nachfolgende Programm zum Test von Gleitkommaoperationen ist mit Compileroptimierung -O0 zu übersetzen, zu disassemblieren und auszuprobieren:

```
float a=0.175, b=31.71, c=1.11;

int main(void){
  a += b;
  a /= c;
  a -= b;
  a *= c;
}
```

	Wert von a	Hex.-Wert von a
1		
2		
3		
4		
5		

a) Arbeiten Sie es im Schrittbetrieb ab und schreiben Sie in die Tabelle neben dem Programm den Wert der Variablen a und deren hexadezimale Darstellung vor Abarbeitung der Anweisung links daneben. 5P

b) Kontrollieren Sie überschlagsweise (nur mit den vier führenden Mantissebits), dass die Werte und ihre Darstellungen zusammenpassen. 2P

Wert von a	s	$c - 0x7F$	$1.m^*$	Überschlag

Wert: $(-1)^s \cdot 2^{c-0x7F} \cdot 1.m$

* nur eine Hexadezimalstelle nach dem Komma

c) Suchen Sie im disassemblierten Code des Hauptprogramms die Unterprogrammaufrufe der Addition, Division etc., darüber die Aufrufadressen der Unterprogramme, dazu die Adresse des letzten Rücksprungs, bevor das Programm endet (nop's beginnen) und bestimmen Sie daraus, wie viele Befehle die Unterprogramme für die Gleitkomma-Grundoperationen insgesamt benötigen. 2P

	Add.	Div.	Sub.	Mult.	letztes ret
Adresse					

Befehlsanzahl für die Gleitkomma-Grundrechenarten insgesamt:

d) Warum funktioniert es, dass die Aufrufadresse des Unterprogramms für die Subtraktionen nur um eins kleiner ist, als die Aufrufadresse für die Additionen? 1P

Aufgabe 2:

Das nachfolgende in einer Header-Datei vereinbarte Unterprogramm:

```
uint16 UP(uint8_t a, uint16_t b, uint8_t c);
```

soll in Assembler geschrieben werden. In welchen Registern bekommt das Assemblerprogramm die Operanden übergebenen und in welchen Registern muss der Rückgabewert stehen? 4P

Aufrufparameter	a	b.Byte1	b.Byte0	c
Register				
Rückgabewert	Byte 1	Byte 0		
Register				

Hinweise: Vereinbaren Sie die Variablen für die Übergabeparameter und den Rückgabewert global (vor dem Haupt- und Unterprogramm). Schreiben Sie irgend ein c-Unterprogramm mit dem vorgegebenen Aufrufkopf, dass den Rückgabewert aus a, b und c berechnet sowie ein c-Hauptprogramm, das das Unterprogramm aufruft. Bestimmen Sie die Registerzuordnung mit Hilfe des Watch-Fensters im Debugger und des disassemblierten Codes. Kontrollieren Sie das Ergebnis anhand von Foliensatz 2 ab Folie 73.

Aufgabe 3:

Das nachfolgende C-Programm enthält eine while-Schleife, in der eine Variable a solange um 1 erhöht wird, wie ihr Wert kleiner 256 ist. Dazu sind die disassemblierten mit O0 und mit O1 übersetzten Programme gezeigt.

C-Programm mit while-Schleife

```
11 void main(void){
12   uint8_t a;
13   while(a<256){
14       a++;
15   }
16 }
```

Mit O1 compiliertes Programm

```
0007D  RJMP PC-0x0000
```

Mit O0 compiliertes Programm

```
0007D  PUSH R28
0007E  PUSH R29
0007F  PUSH R1
00080  IN R28,0x3D
00081  IN R29,0x3E
00082  LDD R24,Y+1
00083  SUBI R24,0xFF
00084  STD Y+1,R24
00085  RJMP PC-0x0003
```

a) Probieren Sie aus, dass das Programm tatsächlich so übersetzt wird. Warum wird das Programm mit -O1 in eine Endlosschleife übersetzt, die nichts tut? Verhält sich das mit -O0 übersetzte Programm anders? 3P

b) Wie ist das C-Programm zu verändern, damit die Schleife abbricht, wenn a nicht mehr kleiner als 256 ist? Kontrollieren Sie auch anhand des dissasemblierten Programms, dass sich ihr verändertes Programm nach 256 Schleifendurchläufen beendet. 3P