Technische Universität
Clausthal Institut für Informatik
Prof. G. Kemnitz

13. November 2018

Rechnerarchitektur: Laborübung 5 (Timer, USART)

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 für vorgeführte Experimente vom Betreuer die Punkte auf dem Aufgabenblatt eintragen und geben Sie, wenn Sie fertig sind, alle Blätter ab. Für eine Bescheinigung der erfolgreichen Teilnahme sind in jeder bis auf einer Laborübung mindestens 60% der Punkte zu erreichen.

Name	Matrikelnummer	Studiengang	Punkte von 20	$\geq 60\%$

Aufgabe 5.1: Das nachfolgende Assemblerprogramm berechnet aus zwei »uint16_t« ein vorzeichenfreies 32-Bit-Produkt:

```
1  .global mult
2   mult:
3    MOVW R18, R22 ;  r19:r18 = b
4    MOVW R20, R24 ;  r21:r20 = a
5    MUL R20,R18   ;  r1:r0 = a0*b0
6    MOVW R22,R0   ;  r23:r22 = r1:r0
7    MUL R21,R19   ;  r1:r0 = a1*b1
8    MOVW R24,R0   ;  r25:r24 = r1:r0
9    MUL R20,R19   ;  r1:r0 = a0*b1
10   ADD R23,R0    ;  r23 = r23 + r0
11   ADC R24,R1    ;  r24 = r24 + r1 +c
12   CLR R1        ;  r1 = 0
13   ADC R25,R1    ;  r25 = r25 + c
14   MUL R21,R18   ;  r1:r0 = a1*b0
15   ADD R23,R0    ;  r23 = r23 + r0
16   ADC R24,R1    ;  r24 = r24 + r1 +c
17   CLR R1        ;  r1 = 0
18   ADC R25,R1    ;  r25 = r25 +c
19   ret;
```

Der nachfolgende Testrahmen vereinbart den Funktionsaufruf, definiert globale Variablen für drei Ergebnisse und führt drei Testbeispiele aus:

```
#include <avr/io.h>

uint32_t mult(uint16_t a, uint16_t b);

int main(void){
 uint32_t p1 = mult(0x1652, 0x09AB);
 uint32_t p2 = mult(0xA04E, 0x70F2);
 uint32_t p3 = mult(0x1652, 0x159E);
}
```

a) Bestimmen Sie für die drei Testbeispiele für jede abzuarbeitende Assembleranweisung die Werte, die in die Zielregister übernommen werden und abschließend das 4-Byte-Produkt in R25:R22 und berechnen Sie für jedes Testbeispiel den Sollwert mit dem Taschenrechner. 6P

Nr.	Register	Berechnung p1	Berechnung p2	Berechnung p3	Test zu d)
3	R19:R18				
4	R21:R20				
5	R1:R0				
6	R23:R22				
7	R1:R0				
8	R25:R24				
9	R1:R0				
10	R23				
11	R24				
12	R1				entfällt
13	R25				entfällt
14	R1:R0				
15	R23				
16	R24				
17	R1				
18	R25				
	R25:R22				
	Sollwert				

b) Überprüfen Sie die Zuweisungen aus der Aufgabenstellung zuvor im Schrittbetrieb und haken Sie die korrekt ausgeführten Anweisungen ab[1].

c) In welchen Registern erwartet das Unterprogramm die Operanden und das aufrufende Programm das Ergebnis? Kontrollieren Sie anhand des disassemblierten Codes und im Schrittbetrieb, dass das aufrufende Programm die Faktoren vor dem Unterprogrammaufruf in den richtigen Registern ablegt und das Ergebnis nach dem Rücksprung aus den richtigen Registern ausliest. 1P

d) Ein potenzieller Fehler in dem Multiplikationsprogramm ist, dass die Anweisungen 12 und 13 fehlen. Suchen Sie einen Test (Eingabewerte + Soll-Ergebnis), mit dem der Fehler an einem abweichenden Ergebnis nachweisbar ist. 3P

Punkte Aufgabe 5.1	

Aufgabe 5.2: Für die Addition von zwei »int16_t« soll eine Funktion

```
int16_t add_i16(int16_t a, int16_t b);
```

entwickelt werden, die bei Über- oder Unterlauf des Wertebereich den größten bzw. kleinsten darstellbaren Wert zurückgibt (Sättigungsarithmetik). Das nachfolgende C-Programm definiert dafür Testbeispiele und einen Testrahmen. Die Datenstruktur für einen Einzeltest ist ein Verbund aus den beiden Eingabewerten und dem Soll-Ergebnis. Für die ersten beiden Testbeispiele ist die Summe darstellbar, einmal positiv und $< 2^{15}$, einmal null und einmal negativ und $\geq -2^{15}$. Bei den unteren vier Testbeispielen ist das Ergebnis der größte bzw. kleinste darstellbare Wert. Der Testrahmen arbeitet in einer Schleife alle Testbeispiele ab. Zum Test setzt man am besten einen Unterbrechungspunkt von »err++« (hochzählen des Fehlerzählers). Zur Fehlerlokalisierung ist im Schrittbetrieb die Testnummer »idx« auf den Wert des fehlgeschlagenen Tests zu setzen, in die Assemblersicht zu wecheln und das Programm schrittweise durchzugehen. 10P

[1] Die Sollwerte bei einem Test mit Soll-/Ist-Vergleich sind zur Maskierungsvermeidung auf eine andere Weise als die Istwerte (diversitär), z.B. manuell zu berechnen.

```
struct test{ // Verbund zur Beschreibung eines Tests
        int16_t a, b, sw;
        };

//                      a         b      Summe
struct test t[7] = {{0x2173, 0x1F74, 0x40E7}, // positiv
                    { 0x7FFF, 0x8001,      0}, // null
                    { 0x0FFF, 0x8001, 0x9000}, // negativ
                    { 0x7FFF, 0x0001, 0x7FFF}, // Überlauf
                    { 0x7FFF, 0x7FFF, 0x7FFF}, // Überlauf
                    { 0x8000, 0xFFFF, 0x8000}, // Unterlauf
                    { 0x8000, 0x8000, 0x8000}};// Unterlauf

uint8_t idx, err;
int16_t s1, s2, sum, soll;

int16_t add_i16(int16_t a, int16_t b); // Funktionsvereinb. Testobjekt

int main(void){
 for (idx=0; idx<7; idx++){
  s1 = t[idx].a; s2 = t[idx].b; soll = t[idx].sw;
  sum = add_i16(s1, s2);
  if (soll != sum)
   err++;
 }
}
```

Hinweise:

- Erstellen einer neuen Assemblerdateil.
- Vereinbarung einer Einsprungmarke »add_i16« als »global«.
- Bei einem Über- oder Unterlauf vorzeichenbehafteter Zahlen wird das Überlaufbit gesetzt.
- Ein Überlauf lässt sich von einem Unterlauf am Vorzeichen- (S) oder am Übertrags (C – Carry) -Bit unterscheiden.
- Eine Übersicht über alle bedingten Sprünge steht in der Befehlssatzbeschreibung »techwww.in.tu-clausthal.de/site/Dokumentation/ATmega2560/AVR_Instruction.pdf« auf Seite 10.
- Das zu entwickelnde Assembler-Unterprogramm muss alle vorgegebenen Testbeispiel korrekt abarbeiten.

Punkte Aufgabe 5.2	