Prof. Dr. Wolfgang Effelsberg
Dirk Stegemann Mannheim, den 27.06.2005

Vorlesung **Programmierkurs 2**, Sommersemester 2005

Übungsblatt 11

Abgabe spätestens am Montag, 04. Juli 2005, um 13.45 Uhr.

Aufgabe 1: Ampel

(6 Punkte, Abgabedatei `ampel.s`)

Schreibe ein MSP430-Assemblerprogramm, das den Sensorknoten als Verkehrsampel betreibt. Die LEDs sollen dabei die Phasen ROT - ROTGELB - GRÜN - GELB - ROT zyklisch durchlaufen, bis der Knoten abgeschaltet wird.

Hinweise:

- Wie bereits auf Übungsblatt 10 erwähnt, empfiehlt es sich, einfach den Bereich *Hauptprogramm* in der Datei `userapp.s` zu überschreiben. Alle umstehenden Befehle sollten dagegen nur verändert werden, wenn man genau weiß, was man tut.
- Die drei Leuchtdioden des Sensorknotens werden über das Byte an der (absoluten) Adresse `0x0029` im Speicher des Prozessors gesteuert. Das niederwertigste Bit (Bit 0) an dieser Adresse steuert die rote, das Bit 1 die grüne und das Bit 2 die gelbe Leuchtdiode. Eine Leuchtdiode ist genau dann eingeschaltet, wenn das entsprechende Bit auf 0 gesetzt ist.
- Um die Leuchtdauer der Dioden zu steuern, kannst du die Unterprozedur `wait` verwenden. Die Wartedauer (=Anzahl der Ausführungen des Befehls `nop`) wird über die Register `R14` und `R15` übergeben, wobei die höherwertigen 16 Bit der Wartedauer in `R15` erwartet werden.
- Der `mspgcc`-Assembler interpretiert die Integer-Konstanten

  `0b10` als Binärdarstellung von 2,

  `0x10` als Hexadezimaldarstellung von 16 und

  `10` als Dezimalzahl 10.
- Für `call` und `br` müssen im Gegensatz zu den `jump`-Befehlen `jmp`, `jge`, `jz`, etc. die Sprungmarken mit führendem `'#'` angegeben werden, d.h. man springt an eine mit dem Label `func` bezeichnete Stelle im Quelltext mit den Befehlen `call #func` bzw. `jmp func`.

Aufgabe 2: 64-Bit-Addition

(6 Punkte, Abgabedatei `addition.s`)

Schreibe eine Funktion in MSP430-Assembler, die zwei 64-Bit-Integer addiert. Als Eingabeparameter erhält sie

- in `R7` die Adresse des ersten Operanden,
- in `R8` die Adresse des zweiten Operanden und
- in `R9` die Adresse, unter der das Ergebnis gespeichert werden soll.

Von der Übungswebseite und aus dotlrn kann ein Testprogramm `userapp-addition.s` (zum Testen in `userapp.s` umbenennen) für die Funktion heruntergeladen werden, in das die Additionsfunktion `add64` an der gekennzeichneten Stelle eingefügt werden soll. Das Programm verwendet die serielle Schnittstelle, das Programm `minicom` muss also beim Test laufen. Beachte, dass vor dem Verlassen der Funktion alle ursprünglichen Registerinhalte wiederhergestellt werden müssen.

Aufgabe 3: Notenspiegel

(8 Punkte, Abgabedatei `noten.s`)

Schreibe eine Funktion `notsp` in MSP430-Assembler, die aus der Ergebnisliste einer Klausur den Notenspiegel berechnet, d.h. eine Tabelle, die angibt, wie oft welche Note vergeben worden ist.

Der Anfang der Ergebnisliste wird in Register `R10` und die Anzahl der Listeneinträge in Register `R11` übergeben. Jeder Listeneintrag der Ergebnisliste besteht aus zwei Bytes für die Matrikelnummer und zwei Bytes für die Note. Der Einfachheit halber enthalte die Liste nur Noten aus $\{1, 2, 3, 4, 5\}$.

Der Notenspiegel soll an der Adresse gespeichert werden, die in Register `R12` übergeben wird. Jeder Listeneintrag des Notenspiegels besteht aus zwei Bytes für die Note und zwei Bytes für die Anzahl der Studenten, die diese Note erzielt haben. Nimm an, dass der Speicherbereich, in dem der Notenspiegel abgelegt werden soll, uninitialisiert ist.

Beispiel:

Parameter

```
R10: 0x0300
R11: #6
R12: 0x0400
```

Ergebnisliste

Adresse	Inhalt
0x0300	#16215
0x0302	#3
0x0304	#16216
0x0306	#5
0x0308	#16220
0x030a	#2
0x030c	#16227
0x030e	#3
0x0310	#16230
0x0312	#2
0x0314	#16245
0x0316	#1

Notenspiegel

Adresse	Inhalt
0x0400	#1
0x0402	#1
0x0404	#2
0x0406	#2
0x0408	#3
0x040a	#2
0x040c	#4
0x040e	#0
0x0410	#5
0x0412	#1

Von der Übungswebseite und aus dotlrn kannst du ein Testprogramm `userapp-noten.s` (zum Testen in `userapp.s` umbenennen) herunterladen, in das die Funktion `notsp` an der gekennzeichneten Stelle eingefügt werden soll. Wie in der vorherigen Aufgabe verwendet das Programm die serielle Schnittstelle, und die ursprünglichen Registerinhalte sollen vor dem Verlassen der Funktion wiederhergestellt werden.