Aufgabenblatt 4

1 Verwendung von Stack

Implementieren Sie ein Hauptprogramm sowie zwei Funktionen.

Das Hauptprogramm lädt zu Kontrollzwecken die beiden Register R0 und R1 mit den Werten 0x44 sowie 0x55, danach ruft es die erste Funktion drei mal auf. Beim Testen vergewissern Sie sich bitte, dass bei allen drei Aufrufen die Werte und Adressen auf dem Stack gleich belegt sind.

Die erste Funktion legt und initialisiert zuerst 3 lokale Variable an, a (uint8 = 5), b (int16 = -6) sowie c (uint32 = 0). Nach der Initialisierung auf dem Stack werden die Variablen a und b in die Register R0 und R1 geladen und danach die zweite Funktion aufgerufen. Der Rückgabewert der zweiten Funktion soll in der Variable c gespeichert werden. Danach ist die Funktion beendet.

In der zweiten Funktion werden im erstem Schritt die beiden Zahlen aus dem Registern R0 und R1 addiert. Aus dem Ergebnis dieser Addition wird der Absolutbetrag gebildet, der auch der Rückgabewert dieser Funktion ist. Die Rückgabe erfolgt im Register R0.

Testen Sie Ihr Programm auch mit zwei Positiven Werten für a und b

Tipps:

- Am besten wäre, wenn Sie die erste Funktion in mindestens zwei Stufen implementieren. Zuerst vergewissern Sie sich, dass die Variablen korrekt auf den Stack geschrieben wurden (siehe nächster Tipp), erst dann fügen Sie die korrekten Werte ein und rufen die zweite Funktion auf.
- Ob Ihr Stack korrekt belegt wird können Sie z.B. mit folgenden Werten testen a = 0xaa, B= 0xbbbb, c = 0xccccccccc und die Belegung des Speichers im „Memory Window“ überprüfen. In der Abbildung (die Adressen sowie Reihenfolge müssen nicht in Ihrer Lösung 100% übereinstimmen) werden die lokalen Variablen in Rot, die gespeicherten Register R0 und R1

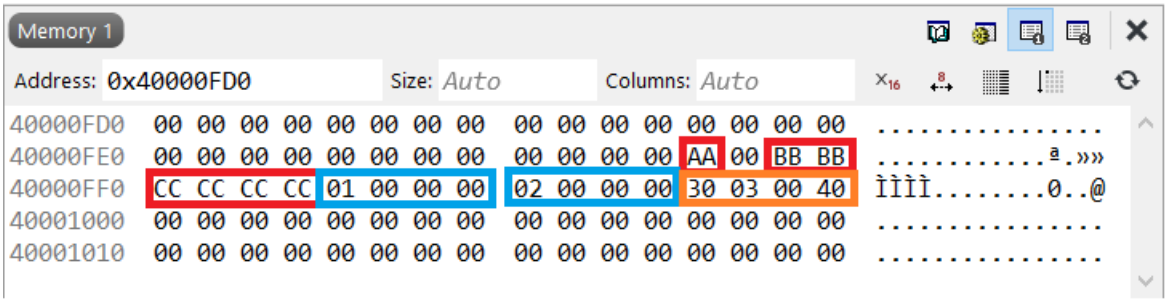

Abbildung 1 Stackbelegung

in Blau, sowie die Rücksprungadresse in Orange angezeigt. Die Werte für R0 undR1 sind in der Abbildung anders als in der Aufgabe verlangt.

- Den Absolutwert in der Funktion 2 können Sie mit der reversen Subtraktion bilden.

Erstellen Sie ein bitte zu dieser Aufgabe passende Struktogramme, laden diese in Ilias hoch bevor Sie diese Aufgabe präsentieren.

Skript
2.3.2 Speichern von Bytes sowie Half-Words
2.2.4.5 Laden von halben Words sowie Bytes
2.2.4.6 Vorzeichenbehaftetes Laden
2.6.6 Umgekehrte Subtraktion

2 Addition von zwei 8 stelligen BCD Zahlen

Implementieren Sie eine Funktion zur Addition zwei 8 stelligen BCD Zahlen (Abbildung 2). Die genaue Funktionsweise des Algorithmus ist im Skript beschrieben. Rufen Sie die Funktion mit unterschiedlichen BCD Zahlen als Parameter auf.

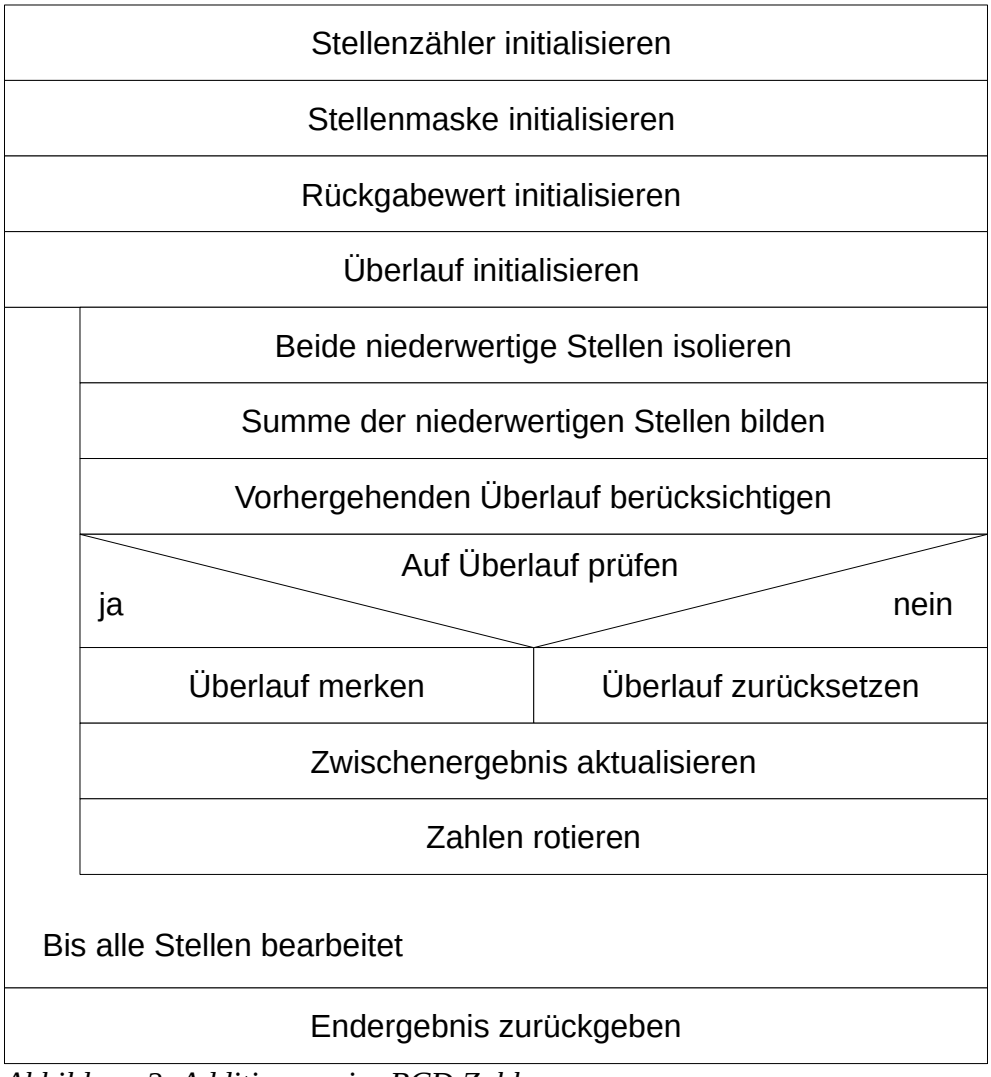

Abbildung 2: Addition zweier BCD Zahlen

Testen Sie Ihre Funktion im Hauptprogramm mit mehreren geeigneten Zahlenkombinationen.

Skript:
4.6 BCD Zahlen
4.6.3 BCD Addition zweier 4-stelliger Zahlen
2.1.2.4 Rechts rotieren

3 Datenstrom Binarisieren

Ausgehend von der Aufgabe 2.3 erweitern Sie das dort implementierte Programm um die Fähigkeit 8 Bit Datenpunkte direkt im Speicher zu binarisieren (in Werte 0 oder 1 umwandeln). Die zu bearbeitenden Daten liegen in einem Datenblock vor (Abbildung 3). Das erste Byte (Index 0) eines Datenblock bestimmt die Menge der zu bearbeitenden Punkte. Das zweite Byte des Blocks enthält die Markierung,ob dieser Block bereits bearbeitet wurde (5), oder noch nicht (0). Ab dem dritten Byte (Index 2) liegen dann die zu bearbeitenden 8 Bit Daten. Im Unterschied zu der A2.3 müssen diesmal die Daten nicht gepackt werden, sondern behalten die ursprüngliche Datengröße.

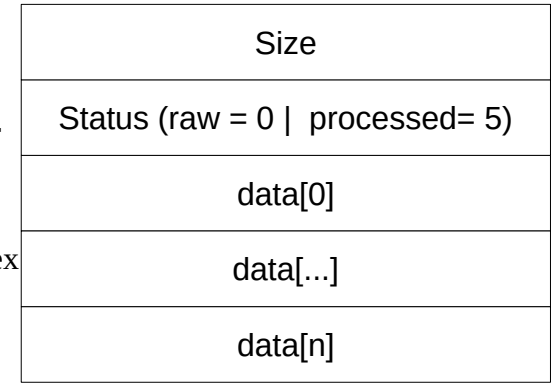

Abbildung 3:

Implementieren Sie eine Funktion, die einen noch nicht bearbeiteten Datenblock automatisch in Binärwerte umwandelt (Abbildung 5). Ein (oder besser Mehrere) Datenblöcke für Ihre Tests können Sie mit dem Präprozessor anlegen. Beispiel:

```
Datablock_1:
.byte 4, 5, 1, 2, 3, 4
```

bzw. für alle die es bevorzugen Ordentlich zu Arbeiten:

```
Datablock_1:
.byte 4                 // Size
.byte 5                 // Status
.byte 1, 2, 3, 4        // Data
```

Bitte beachten Sie auch die Tatsache, dass je nach dem wie groß ihr Datenblock ist, bzw. wo dieser Platziert ist entsprechendes Padding vor dem Folgendem Assemblercode notwendig ist. Anderseits könnten Sie mit dieser Fehlermeldung belohnt werden :„ ***Error: unaligned opcodes detected in executable segment***“. Was an sich bedeutet das Ihr Linker versucht Assemblerbefehle auf nicht durch 4 teilbaren Adressen zu legen – der im Beispiel gezeigte Datenblock ist nur 6 Byte groß, also es fehlen noch 2 Byte um den Linker glücklich zu machen.

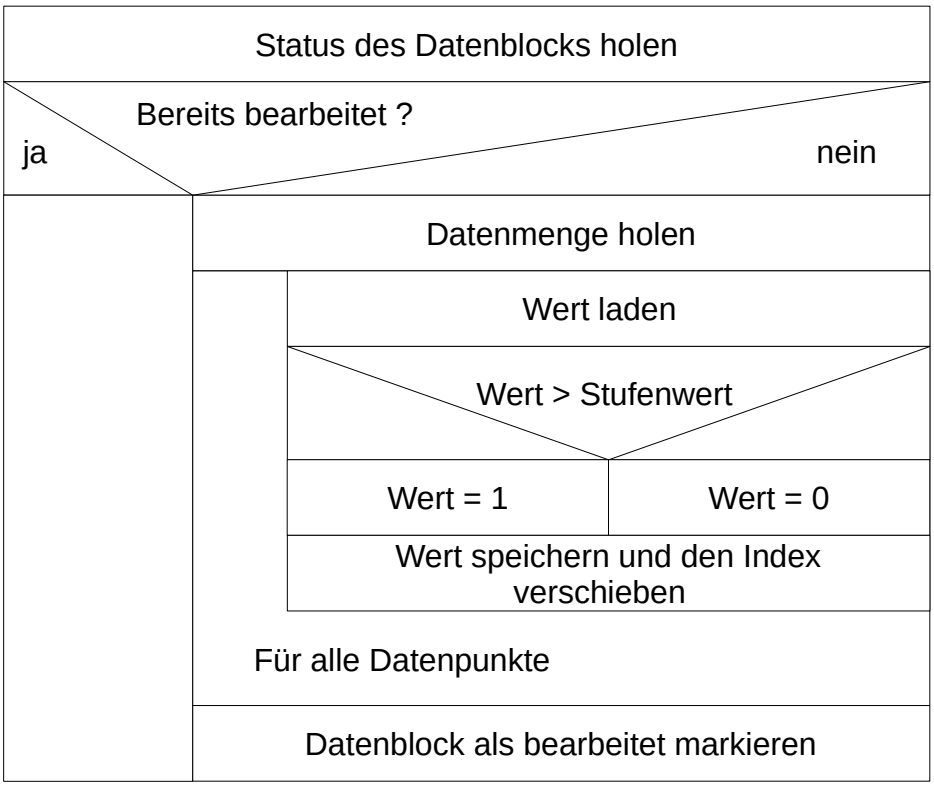

Abbildung 4: Funktion „Daten Binarisieren“

Tipps:

Bei den Anweisungen ***ldr*** bzw.. ***str*** können Sie auch zwei Register gleichzeitig als Index verwenden z.B. ***ldr Rx,[Ry,Rz]***. In diesem Fall haben Sie gleichzeitig einen Basis- und einen Elementzeiger zur Verfügung.

Skript:
2.2.4.4 Indirekte Adressierung mit Index