Hochschule Karlsruhe – Technik und Wirtschaft
Fakultät für Informatik und Wirtschaftsinformatik
Prof. Dr. A. Ditzinger / Dipl.-Inform. (FH) O. Gniot
Prof. Dr. N. Link / Dipl.-Ing. J. Krastel

D i g i t a l l a b o r

Versuch: Erste Schritte mit maschinennaher C166 Programmierung

Ziel: Im heutigen Versuch sollen Sie die internen Abläufe in einem typischen Prozessor verstehen. Sie sollten sich erarbeiten, was die Unterschiede zwischen Konstanten und Variablen sind, und wie der Prozessor mit den internen Registern arbeitet. Weiterhin sollten Sie den Übergabemechanismus von Parameteradressen an Unterprogramme (call by reference) und die Verarbeitung der Parameter per indirekter Adressierung verstehen.

Verwenden Sie die Keil Software mit der vorgegebenen Vorlage. Zum Debuggen brauchen wir heute keine Hardware, der Simulator genügt. Verständnisfragen, die bei den Aufgaben gestellt werden, beantworten Sie in der Ausarbeitung.

Aufgabe 1

Vereinbaren Sie Konstanten mit folgenden Namen und Werten:

op1 = 30000, op2 = 5000, op3 = 40000, op4 = 4999, op5 = -30000

Im Hauptprogramm laden Sie dann einfach die Register R1 bis R5 mit den Konstanten op1 bis op5. Assemblieren und binden Sie Ihr Programm und debuggen Sie im Einzelschritt. Öffnen Sie ggf. das "Register-Window", falls es nicht offen ist. Werden die richtigen Werte in die Register geladen? Welche Adressierungsart mussten Sie verwenden, und an welcher Stelle im übersetzten Programmcode tauchen die Konstanten auf? Schauen Sie zur Beantwortung dieser Frage im Assembler-Listing und mit Hilfe des Debuggers direkt im Programmspeicher nach.

Aufgabe 2

Kopieren Sie Ihr Programm aus Aufgabe 1 in eine neue Datei und übernehmen Sie diese in Ihr Projekt. Löschen Sie die Konstanten aus Ihrem Programm und vereinbaren Sie stattdessen gleichnamige 16 Bit-Variablen, die mit den Werten aus Aufgabe 1 initialisiert sind.

Im Hauptprogramm laden Sie dann wiederum die Register R1 bis R5 mit den Werten der Variablen op1 bis op5. Debuggen Sie erneut und verifizieren Sie, dass die richtigen Werte geladen werden. Welche Adressierungsart wurde diesmal verwendet, und was taucht nun im übersetzten Programmcode an Stelle der Konstanten auf? Ermitteln Sie mit Hilfe des Assembler- und des Linker-Listings die Adresse der Variablen und zeigen Sie den entsprechenden Speicher im "Memory-Window" an. Was fällt auf.

Aufgabe 3

Erweitern Sie das Programm aus Aufgabe 2, indem Sie nach dem Laden der Register R1 bis R5 folgende Rechenoperationen mit den Register-Operanden und den Ziel-Registern R10 bis R14 durchführen:

R10 = op1 + op2; R11 = op1 + op3; R12 = op4 - op2; R13 = op1 + op5, R14 = op3 + op5;

Debuggen Sie Ihr Programm im Single Step. Stimmen die Rechenergebnisse? Notieren Sie nach jeder Rechenoperation die Werte der Flags C, Z, V und N. Wie kommen diese Werte zustande und was bedeuten sie?

Aufgabe 4

Schreiben Sie ein Unterprogramm, das zwei 16 Bit Zahlen addiert. Die Zahlen stehen direkt im Speicher. Das Unterprogramm soll beim Aufruf in R0 einen Pointer auf den ersten Operanden, in R1 einen Pointer auf den zweiten Operanden erhalten. Das Resultat wird als Wert in R2 zurückgeliefert. Ausser R2 und den Flags soll das UP keine Register zerstören.

Nutzen Sie Ihr Programm 4 mal, um die Werte in R10, R11, R13 und R14 zu berechnen.

Aufgabe 5

Na ja, die 16 bittige Rechnerei ist ja bereichsmäßig wohl doch etwas eingeschränkt und es war zugegeben keine gute Idee, ein Unterprogramm zu schreiben, dessen Aufruf-Overhead größer ist, als der Nutzen. Beides wollen wir jetzt ändern und Unterprogramme für 32 Bit Arithmetik erstellen.

Dazu legen Sie zunächst zwei 32 Bit Variablen an, die Sie auf die Werte 120000 und 75000 initialisieren.

Moment mal, 32 Bit Variablen anlegen, das haben wir doch gar nicht besprochen?! Stimmt, denn der Prozessor unterstützt dieses Datenformat nicht. Und wie immer, wenn entweder der Prozessor, oder die Programmiersprache ein Datenformat nicht unterstützt, dann muss man es eben selbst programmieren. Hier legen Sie eben einfach zwei 16 Bit Werte hintereinander in den Speicher. Ganz little Endian mäßig kommen zuerst die unteren 16 Bit, dann die oberen. Das sieht dann so aus:

```
MyVar32     DW     (120000 AND 0xFFFF)     ;untere 16 Bit
            DW     (120000 SHR 16)         ;obere 16 Bit
```

So, nachdem wir nun die beiden 32 Bit Variablen angelegt haben, wollen wir auch mit ihnen rechnen:

Schreiben Sie ein Unterprogramm "add32", das zwei 32 Bit Zahlen addiert. Es erhält die Pointer wie in Aufgabe 4 und liefert das Resultat in R2 (untere 16 Bit) und R3 (obere 16 Bit) zurück. Ausser R2, R3 und den Flags soll es ebenfalls keine Register zerstören.

Rufen Sie das UP mit den beiden Variablen auf und berechnen Sie 120000 + 75000.

Abschliessend kopieren Sie Ihr UP und modifizieren die Kopie zu "sub32", das zwei 32 Bit Zahlen, bei gleicher Aufruf-Struktur subtrahiert.

Berechnen Sie zusätzlich 120000 - 75000 und 75000 - 120000 und geben Sie die Hex-Resultate der 3 Rechnungen in der Ausarbeitung an.