ÜBUNG 2 FÜR LABOR MIKRO-RECHNER

Einführung in die Assemblerprogrammierung

ZIELE

Ziele dieser Übung sind:
- einfache Assembler-Code-Sequenzen erstellen und korrekt ausführen können

- den Zugriff mit InLine-Assembler-Code auf die Parameter und die lokalen Variablen der umgebenden C-Funktion verstehen.

- Value- resp. Reference- Parametern mit Assembler- Code verarbeiten können

AUFGABEN

1. Implementieren Sie das folgenede C- Unterprogramm:

```
int berechne(int par1, int *par2)
{
  int sum = 0;

  --asm
  {
    for (int i = 0; i < par; i++)
      sum = sum + par2;
  }
  return sum;
}
```

2. Erstellen Sie ein C- Programm, das folgenden Anforderungen erfüllt:

   Hauptprogramm in C:

   1.) einlesen eines Strings vom Keyboard

   2.) aufrufen der C-Funktion "verarbeiten(...)"

```
    void verarbeiten(char *c1, char *c2)
    {
      // C-Code:
       ermitteln der Länge des eingelesenen Strings

       // InLineAssembler
       __asm
       {
         für (i = 1 bis Stringlänge) loop
         {
           laden des letzen Characters des eingelesenen Strings in
             das Register AL
           Gross-/Klein-Wandlung
           verschieben des gewandelten Characters in die erste
             Position des neuen Strings
         }
       }
    }
```

Die C-Funktion "verarbeiten(...)" verschiebt somit den eingelesenen String in einen anderen Speicherbereich und manipuliert dabei den String wie folgt:
- Gross-Buchstaben <-> Klein-Buchstaben
- String umkehren, zB.: aus "Hallo" wird OLLAh"

Verfolgen Sie die Wandlung des Strings im Assembler-Code mit dem Debugger in Einzelschrittmodus

3.) ausgeben des verarbeiteten Stringes

Hinweise:

1.) Deklaration der beiden Strings als globale Variablen im C-Programm als Character-Array's:

```
# define ANZ 50

char s1[ANZ];   // einzulesender String
char s2[ANZ];   // manipulierter String
```

2.) Semantik des Array-Namens:

in C ist der Array-Name ein Pointer auf das erste Element des Array's

```
s1 = &s1[0]
```

3.) einlesen eines Stringes vom Keyboard:

```
...
printf("String eingeben (max. 49 Elemente ohne Space): ");
scanf("%s", s1);
...
```

ein Space wird als Stringterminierung interpretiert

4.) der eingelesene String hat ein Element mehr, als die Anzahl eingetippter Buchstaben;

String name = "Ernst" -> 'E' 'r' 'n' 's' 't' '\0';

beim zusätzlichen Element handelt es sich um das String-Terminierungszeichen (Character null);

5.) ermitteln der Länge eines Stringes in C:

```
int i = strlen(s1);
// strlen(...) ist in der Header-Datei <string.h> definiert
```

- grosse und kleine Buchstaben sind nur im Bit 5 des entsrpechenden ASCII-Codes unterschiedlich (Bit 5 = 1 -> klein; Bit 5 = 0 -> gross)

  Diese Invertierung des Bit 5 ist mit einem geeigneten Maschinen-Befehl vorzunehmen

3. Im Gegensatz zu den HLL stellt der Assembler normalerweise keine Kontroll-Strukturen zum Codieren von Verzweigungen (if...; case..., etc.) und von Iterationen (while..., repeat..., etc.) zur Verfügung. Diese Kontroll-Strukturen werden im Assembler auf der Basis von Sprung-Befehlen nachgebildet.

   Studieren Sie als Vorbereitung auf die nächste Assembler-Übung:

   3.1.) im Vorlesungs-Skript das Kapitel "1.3.4 Elemente des Kontrollflusses insbesondere die dazugehörigen Beilagen:
   - "Kap. 6.4 HIGH-LEVEL LOGIC STRUCTURES", Seite 204 bis 212

   - "Looping" -> zeigt die Wirkungsweise der Loop-Befehle (LOOP, LOOPE, LOOPZ, etc.) und deren Anwendung zur Codierung von For-Loops;

   3.2.) Beilage zur Laborübung: "Conditional-Jump Instructions Used after Compare"

   Bemerkung: dieser Lehrstoff wird in der nächsten Prüfung als bekannt vorausgesetzt und abgefragt.

TERMINE

- Abgabe des Listings und Demo (individuell für jeden Studenten) während des 3. Labornachmittags.
  Muss erfüllt sein für die Testaterlangung.

BEILAGEN, LITERATUR

- Demo-Programm-Gerüst mit der C-Funktion "zugriffDemo(...)"