ÜBUNG 7 FÜR LABOR MIKRO-RECHNER

Unterprogramm-Technik und Einbinden von Assembler-Code in HLL

ZIELE

- Üben des Codierens von Assembler-Unterprogrammen;

- Üben des Einbindens von Assembler-Routinen in ein C-Programm;

Aufgabe 1

Man codiere in Assembler (mustergültig) die Funktion "myFunction" und demonstriere den korrekten Ablauf dieser Funktion

```
  void myFunction (int[] * array, int * resultat)
  {
    // lokale Variablen
    int local[4] = {0, 1, 2, 3};
    int sum = 0;


    for (int i = 0; i < 4; i++)
    {
      sum = sum + local[i]  + *(array + i);
    }

    return sum;
  }
```

Randbedingungen
- Übergabe des Array's by Reference
- Behandlung der lokalen Variablen, wie im Unterricht gelernt
- Rückgabe des Resultates via Pointer
- Konvention: Stdcall

Aufgabe 2

Geg.: - Pointer auf eine parameterlose Prozedur pX: p DD pX

- parameterlose Prozdur pX:

```
pX PROC
  ...
  ; Ausgabe auf Bildschirm: "innerhalb pX"
  ;
  ...
pX ENDP
```

- Prozedur doCall

```
void doCall(p: ProcedurePointer)
{
  ...
  p;      // dh.: CALL p
  ...
}
```

Ges.: Codierung der beiden Prozeduren "p" und "doCall" und Demo des
korrekten Ablaufes

Aufgabe 3

In der Übung 5 wurde das Einbinden von Assembler-Code in ein Java-Programm behandelt. Dies erfolgte indirekt: C-Code, der "Inline-Assembler- Code" enthält, wurde via JNI in das Java-Programm eingebunden.

In dieser Übung wird der Assembler-Code nicht via In-Line-Assemblierung, sondern via "Einbinden eines Assembler-Moduls in ein C (C++) Programm", behandelt.
Die dazu notwendige Theorie findet man im Anhang 1, sowie im Unterrichtstoff Kapitel 7 "modulare Programmierung" und Kapitel 8 "Schnittstelle zwischen HLL und Assembler".

Ein Javaprogramm nimmt Eingabewerte vom Bediener entgegen, berechnet mit Hilfe der Methode "functionName" ein Resultat und gibt dieses aus.

Die Methode "functionName" ist "native" und hat folgende Spezifikation

```
public native int functionName(int par1, int par2);
```

Die Berechnung, die von "functionName" ausgeführt wird, kann man frei wählen (zB.: einfacher arithmetischer Ausdruck mit Additionen und|oder Subtraktionen). Die in C codierte Funktion "functionName" dient aber nur als Hüllfunktion: Sie ruft im Assembler-Modul implementierte Prozeduren auf. Diese führen die eigentliche Berechnung durch.

Im C-/Assembler- Modul ist die Berechnung wie folgt zu codieren:

```
extern int function2 (int par1, int* par2);
```

TERMINE

- Abgabe aller Assembler- Übungen abgeschlossen (unaufgefordert) spätestens am 8. Juli 2005.

Anhang 1

Damit der Linker die vom Assembler resp. C-Compiler erzeugten Obj-Files zusammenbinden kann müssen die folgenden Gesichtspunkte beachtet werden:

1.) das im C-Programm zu verwendende Unterprogramm muss aus dem Assembler-Modul exportiert werden; dazu braucht man die PUBLIC-Direktive;

2.) das im Assembler auscodierte Unterprogramm muss in das C-Modul importiert werden; das erfolgt mit der (optionalen) extern Direktive;

3.) die Namenstransformation des C-Compilers ist zu berücksichtigen; er verändert die vom Programmierer eingeführten Namen: das heisst, der C-Compiler transformiert den Namensraum des Programmierers in den Namensraum des Compilers (die vom Compiler veränderten Namen bezeichnet man als decorierte Namen);
Bsp.:

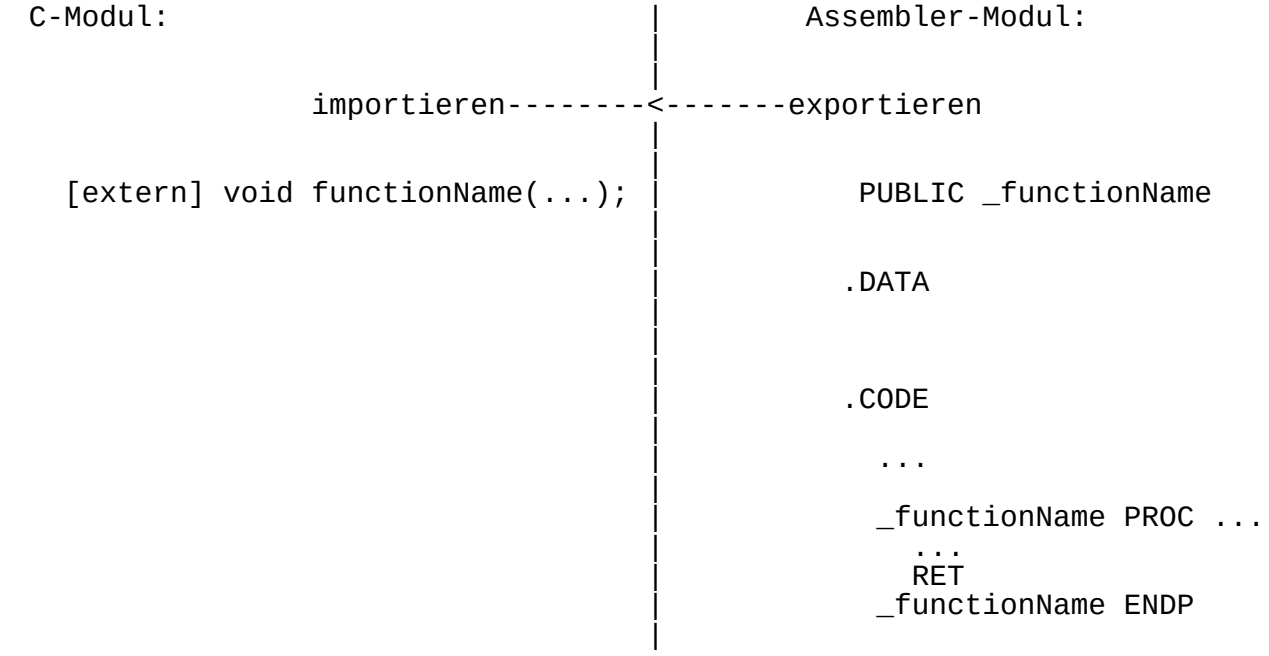
Namen des Programmieres:		Namensraum des Compilers; (decorierte Namen)
int iTmp = 1;	->	_iTmp
int func(void)	->	_func

im Gegensatz zum C-Compiler macht der Assembler keine Namens-Transformation; der Namensraum beim Assembler bleibt unverändert;

Konsequenz: den Namen im C-Programm muss im Assembler ein underscore dazugefügt werden; so heisst zB.: das C-Unterprogramm functionName(...) im Assembler: _functionName

4.) die Unterscheidung von Gross-/Klein-Schreibung ist zu beachten:
- C ist "casesensitive";
- Assembler ist defaultmässig nicht "casesensitive";

Beispiel:



```
int aCfunction()                    |
{                                   |
  ...                               |
  // Aufruf der in Assembler        |
  // implementierten Funktion       |
  functionName(...);                |
  ...                               |
}                                   |
```

Bemerkung zu C++

- das Importieren und Exportieren von Variablen erfolgt nach den gleichen Konventionen wie bei C;

- bei den Prozeduren ist zu beachten, dass die Namenstransformation durch den C++ Compiler anders ist, als beim C-Compiler;

  am einfachsten umgeht man dieses Problem, indem man den C++ Compiler anweist auch bei Prozeduren die Namenstransformation des C-Compilers zu verwenden;

  dies erfolgt mit folgender Syntax:

```
  extern "C"
  {
    int sin(int);
    void printText(char, char, char *, char);
  }
```

  die decorierten (transformierten) Namen lauten damit:

```
    sin       --->   _sin
    printText --->   _printText
```

- bei Unklarheiten:
  - Compiler-Manual konsultieren;
  - ein Assembler-Listing des C++ Compilers konsultieren;