Universität Mannheim
Lehrstuhl für Praktische Informatik IV
Prof. Dr. W. Effelsberg
Christoph Kuhmünch, Gerald Kühne

Praktische Informatik II
SS 2000

Übungsblatt 9 **Ausgabe: Mi, 05.07.00** **Abgabe: Di, 11.07.00, 18 Uhr**

Aufgabe 1: Assembler-Programmierung [10 Punkte]

Das C-Programm auf Seite 2 wurde von einem Standardcompiler in 68000-Maschinencode übersetzt und von uns etwas gekürzt (siehe Seiten 3-4). Beantworten Sie folgende Fragen:

(a) [1 Punkt] Was leistet dieses Programm?

(b) [2 Punkte] Wie wird Speicherplatz für die lokalen Variablen der Prozeduren geschaffen? Wie sind die Variablen `laenge`, `eingabe` und `sort` auf dem Stack in `main` angeordnet?

(c) [1 Punkt] Wie werden Variablen-Werte an Unterprogramme übergeben?

(d) [1 Punkt] Wie werden Ergebnisse an das Hauptprogramm zurückgeliefert?

(e) [4 Punkte] Skizzieren Sie den Stack an der mit $\longrightarrow$`*****` gekennzeichneten Stelle. Gehen Sie dabei von einem eingelesenen Wert von 7428 aus.

(f) [1 Punkt] Wie wird der Unterprogrammaufruf durchgeführt? Wie wird die Rücksprungadresse gespeichert?

Die Syntax des Assemblerlistings auf den Seiten 3 und 4 ist etwas anders als gewohnt:

- Sedezimalkonstanten werden nicht durch ein führendes Dollarzeichen, sondern durch 0x gekennzeichnet.
- Die Kürzel für die Länge der Operanden (l,w,b) werden nicht mit einem Punkt vom Operator getrennt (Aus $add.l$ wird $addl$).
- Die Adressierungsart „Adressregister indirekt mit Distanz“ wird in der Form (z.B.) $a6@(0x1c)$ anstatt $\$1c(a6)$ angegeben.
- sp (Stackpointer) steht für das Register $a7$

C-Programm:

```
#include <stdio.h>

long Einlesen() {
  long zahl;
  printf("Bitte Zahl eingeben:");
  scanf("%ld", &zahl);
  return zahl;
}

int Zerlegung(long zahl, int *vektor) {
  int i,j;
  for (j=0; zahl>0; j++) {
    vektor[j]=zahl%10;
    zahl=zahl/10;
  }
  return (j-1);
}

void Sortiere(int *vektor, int ende) {
  int i,j, hilf;
  for (i=0; i<ende; i++)
    for (j=i+1; j<=ende; j++)
      if (vektor[j] < vektor[i]) {
        hilf = vektor[j];
        vektor[j] = vektor[i];
        vektor[i] = hilf;
      }
}

void Ausgeben(int *vektor, int ende) {
  int i;
  for (i=0; i<=ende; i++)
    printf("%d", vektor[i]);
}

int main (){
  int laenge;
  long eingabe;
  int sort[10];

  eingabe = Einlesen();
  laenge = Zerlegung(eingabe, sort);
  Sortiere(sort, laenge);
  Ausgeben(sort, laenge);
}
```

Assemblerprogramm:

```
.globl _Einlesen
_Einlesen:
                link a6,#-4
                ;liest eine Integer - Zahl (4Bytes) ein
                ...
                unlk a6
                rts

.globl _Zerlegung
_Zerlegung:
                link a6,#-8
                moveml #0x3c00,sp@-
                nop
                clrl a6@(-8)
L3:
                tstl a6@(8)
                jgt L6
                jra L4
L6:
                movel a6@(-8),d0
                movel d0,d1
                movel d1,d0
                asll #2,d0
                movel a6@(12),a0
                movel a6@(8),d1
**********************************STACK angeben***************
                moveq #10,d2
                divs  d2,d1
                asrl  #16,d1
                movel d1,a0@(d0:l)
                movel a6@(8),d0
                moveq #10,d1
                divs d1,d0
                andl #$0000FFFF,d0
                movel d0,a6@(8)
L5:
                addql #1,a6@(-8)
                jra L3
L4:
                movel a6@(-8),d1
                subql #1,d1
                movel d1,d0
                jra L2
L2:
                moveml a6@(-24),#0x3c
                unlk a6
                rts
```

```
.globl _Sortiere
_Sortiere:
                link a6,#-12
                ...
                unlk a6
                rts

.globl _Ausgeben
_Ausgeben:
                link a6,#-4
                ...
                unlk a6
                rts

.globl _main
_main:
                link a6,#-48
                jbsr ___main
                jbsr _Einlesen       ; gibt eingelesenen Wert in D0 zurueck
                movel d0,a6@(-8)     ; Variable eingabe steht in D0
                lea a6@(-48),a0
                movel a0,sp@-
                movel a6@(-8),sp@-
                jbsr _Zerlegung
                addqw #8,sp
                movel d0,a6@(-4)
                movel a6@(-4),sp@-
                lea a6@(-48),a0
                movel a0,sp@-
                jbsr _Sortiere
                addqw #8,sp
                movel a6@(-4),sp@-
                lea a6@(-48),a0
                movel a0,sp@-
                jbsr _Ausgeben
                addqw #8,sp
L22:
                unlk a6
                rts
```

Aufgabe 2: Assembler-Programmierung [10 Punkte]

Der Binomialkoeffizient ist rekursiv wie folgt definiert:

$$\begin{pmatrix} n \\ k \end{pmatrix} = \begin{pmatrix} n-1 \\ k \end{pmatrix} + \begin{pmatrix} n-1 \\ k-1 \end{pmatrix}$$

Desweiteren gilt:

$$\begin{pmatrix} n \\ 0 \end{pmatrix} = \begin{pmatrix} n \\ n \end{pmatrix} = 1$$

Entwerfen Sie ein 68000-Assemblerprogramm, das zwei Zahlen n und k ($n > k$) einliest und den zugehörigen Binomialkoeffizienten rekursiv berechnet und abspeichert.

Nachfolgend ist ein Programmfragment angegeben, daß Sie für Ihre Implementierung verwenden sollen. Geben Sie Ihren Quelltext **ohne** die Zeilen „Konstantendefinitionen 1–3“ per abox unter dem Programmnamen `binom.s` ab.

```
          ORG     $0
          DC.L    $8000           Stack pointer value after a reset
          DC.L    START           Program counter value after a reset


          ORG     $2000           Start at location 2000 Hex

START     MOVE.W  N,D0
          MOVE.W  K,D1
          ...                     Berechnung ueber Unterprogramm
          ...
          MOVE.L  D6,ERGEBNIS
          BREAK


N         DC.W    8               Konstantendefinition 1
K         DC.W    2               Konstantendefinition 2
ERGEBNIS  DS.L    1               Konstantendefinition 3
```