Info4 in 7 Minuten

mutax@cs.tu-berlin.de

22.7.2006

Inhaltsverzeichnis

Intro

Was soll dieses Dokument eigentlich?
Nunja, das ist gar nicht so leicht zu sagen - vor allem handelt es sich um meine Klausurvorbereitung für den Info4-Teil der Informatik B - Klausur.
Der Inhalt soll eine kurze Zusammenstellung der wichtigsten Themen des Semesters mit Code unterlegen und mit in der Klausur helfen - es handelt sich um eine Kofferklausur.
Es ist bei weitem nicht vollständig und deckt nicht den gesamten Stoff ab - dann hätte ich ebensogut das Script abtippen können, vielmehr soll es diverse Codeschnipsel einigermassen übersichtlich vereinigen.

Die Themen der Vorlesung Informatik 4:

- Hardwarenahe Programmierung mit Assembler
- Parallele Programmierung in Java **
- Assembler *
- Einf. Betriebssysteme *
- Nebenläufigkeit und Synchronisation **
- Prozeßbeschreibung
- Prozeßverwaltung
- Leichtgewichtige Prozesse *
- Betriebssystem-Kern
- Schloßvariablen
- Semaphore
  siehe Wikiartikel
- Monitore *
- Nachrichten/Sockets *
- Kommunikationskanäle
- Prozedurfernaufrufe *
- Ein- und Ausgabe
- Speicherverwaltung **
  siehe Script - abtippen bringt nichts
- Betriebsmittelvergabe

*) werden hier behandelt
**) werden hier nur angeschnitten

Teil I
Assembler

1 Motorola 68k - kurze Einführung und Beispiele

1.1 Einführung in den 68k Assembler

Die meisten Befehle lassen sich durch .B, .W und .L modifizieren:

1. MOVE.B - Byteweise (1 Byte)
2. MOVE.W - Wortweise (2 Byte)
3. MOVE.L - Langwort (4 Byte)

Es gibt die Datenregister D0-D7 und die Adressregister A0-A7, wobei A7 der SP ist und daher nicht zur Verfügung steht!
Aliases für Register können per #DEFINE gesetzt werden oder durch direkte Zuweisung:
counter=D0
Die Pre- und Postdekrementgroessen bei Zugriffen wie 21(SP) hängen vom Befehlsmodus ab!
Alle Operationen wie MOVE und ADD geben erst Quelle und dann Ziel an - nicht wie bei x86!

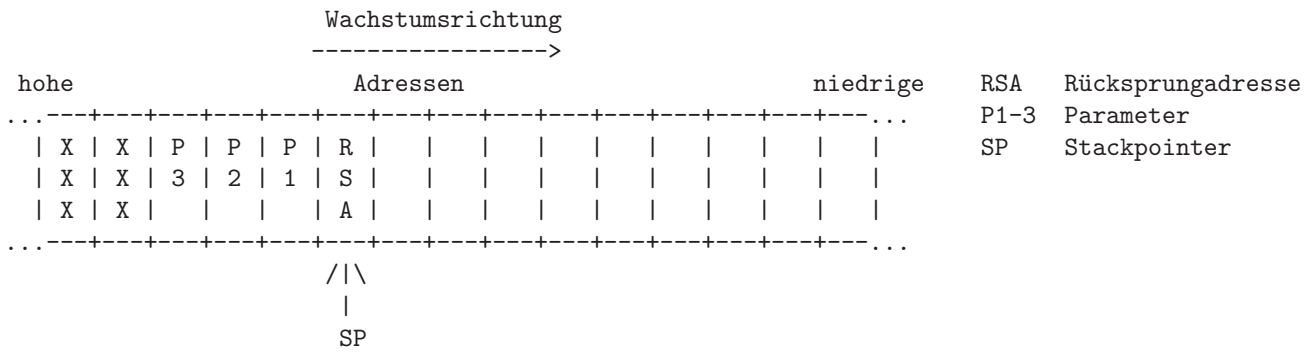
1.2 Benutzung des Stacks

1.2.1 Parameterübergabe per Stack

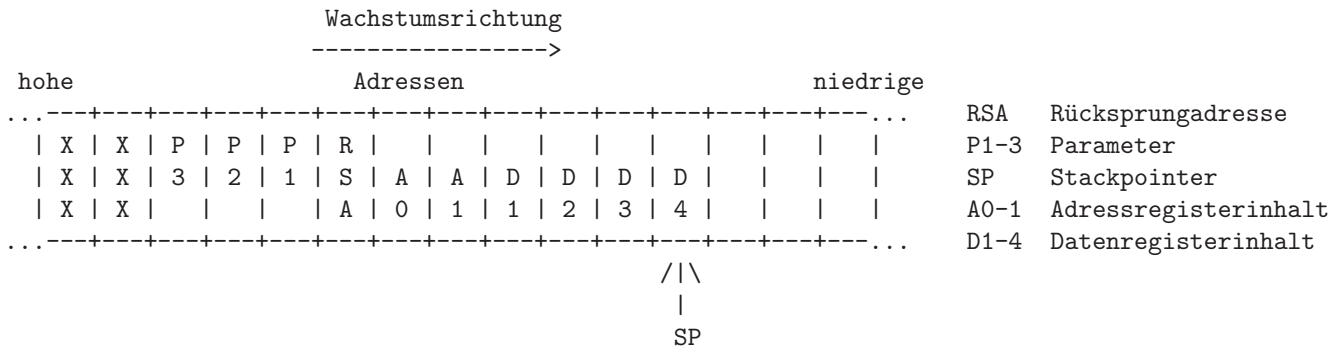
Theoretisch könnten Parameter auch über Register übergeben werden, jedoch klappt dies ausschliesslich für primitive Typen.
Will man jedoch structs oder ganze Strings als Parameter nutzen, so wird lediglich die Adresse in Form eines Pointers als Argument auf dem Stack übergeben.
Mehrere Parameter landen so hintereinander auf dem Stack, der SP zeigt auf die Rücksprungadresse (??) und das Assemblerprogramm kann die Argumente dann nacheinander einlesen.
Nach den Parametern folgt die Rücksprungadresse - die keinesfalls überschrieben werden darf!
Damit dies funktioniert, muss man die Position der Argumente auf dem Stack berechnen - diese hängt aber auch von der Anzahl der auf dem Stack gesicherten Register ab (MOVEM.X).

1.2.2 Visualisierung des Stacks

unmittelbar nach Aufruf eines Unterprogramms mit den Parametern P1,P2,P3 sieht der Stack aus wie folgt:



nach dem Sichern der Register zeigt sich der Stack folgendermassen:



Gesichert werden auf dem Stack immer ganze Langworte (4Byte)
Möchte man also nun auf den Paramter P2 zugreifen, so berechnet man die Adresse relativ zu SP als (8*4) = 32, da wir 6*4 Byte Register und die Rücksprungadresse überspringen müssen und dann noch den Wert P3.
Der Zugriff erfolgt also mit MOVE.L 32(SP),D1

1.2.3 Rückgabewerte

Rückgabewerte sind immer entweder Adressen (also Pointer) oder Grunddatentypen - diese werden im Register D0 bzw. D0/D1 an den Aufrufer übergeben - dieser rechnet damit, weshalb er dafür sorgt, dass in D0 (D0/D1) keine für ihn wichtige Daten stehen.
Sollen Verbunddatentypen zurückgeliefert werden, so wird für diese Platz auf dem Stack reserviert und die Adresse in A1 übergeben, die dann zusätzlich noch in D0 steht.
⇒ Besser bei Grunddatentypen bleiben...

1.3 Beispielprogramm: Hammming-Code

Eine Assembler-Routine soll aus einem C-Programm aus aufgerufen werden.
Die Routine berechnet das kleinste m für das gilt $2^m \geq n + m + 1$.
Der Parameter n wird aus dem C-Programm heraus in Form eines Poniters übergeben, der Aufruf in C dazu lautet:

```
unsigned int hamming(unsigned int *n);
```

Listing 1: Assemblerprogramm zur hamming-Code berechnung

```
1
.GLOBAL _hamming                    | Dem Linker das Symbol bekanntmachen

.EVEN                               | even alignment an geraden speicheradressen
| **
|* Gesucht: ein m dass die Gleicung 2^m >= n+m+1 erfuellt
| **
 _hamming:
            MOVEM.L A0|D1-D3,-(SP)  | Sichern der Register auf Stack
            MOVEA.L 20(SP),A0       | Holen des Pointers
            MOVE.L  (A0),D1         | Laden des Pointerinhaltes (n)
            MOVE.L  #1,D0           | m=1   Initialisierung
            MOVE.L  #2,D2           | 2^m=1
            MOVE.L  #2,D3           |    m+1
            ADD.L   D1,D3           | n+(   )  (n+m+1)

    loop:   CMP.L   D2,D3           | (n+m+1 - 2^m)
            BLS     end             | 2^m <= n+m+1 erfuellt? wenn ja, ende
            ADDQ.L  #1,D0           | naechstes m (m++)
            ADDQ.L  #1,D3           |
            ADD.L   D2,D2           | "2^m" verdoppeln da m++
            BCC     loop            | Carry clear? kein Ueberlauf?
            MOVE.L  #0,D0           | bei Ueberlauf 0 sichern
     end:
            MOVEM.L (SP)+, A0|D1-D3 | Register ruecksichern
            RTS                     | Return from Subroutine, Ergebnis
                                    | in D0
```

1.4 Beispielprogramm: Big- to Little-Endian

Die hier vorgestellte Assembler-Routine soll eine Big- to Little Endian konversion auf dem Motorola Prozessor durchführen, um die Daten per Netz an einen x86 zu senden.
Die Abbruchbedingung besteht aus einem bestimmten Datenwort, dass den Abbruch der Routine signalisiert.
Der Aufruf der Routine in C dazu lautet:

```
void bigtolittle(unsigned void *quelle, unsigned void *ziel);
```

Listing 2: Assemblerprogramm zur Endiankonversion

```
| Das laden und sichern in der loop sind vertauscht - das ist aber
|  korrekt so!
.GLOBAL _bigtolittle          | Dem Linker das Symbol bekanntmachen

.EVEN                         | even alignment an geraden speicheradressen

_bigtolittle:
            MOVEM.L A0-A1,-(SP)    | Sichern der benutzten Register auf Stack
            MOVE.L  32(SP), A1     | Zieladresse in A1
            MOVE.L  28(SP), A0     | Quelladresse in A0
            MOVE.W   (A0)+, D0     | erstes Datenwort lesen ?.W?
      loop:
            CMP.W   D0,0xFEDC      |FEDC ist Abbruchkriterium
            BEQ     End            |wenn ja, ende
            ROL.W   #8,D0          |Konvertierung
            MOVE.W  D0,(A1)+       |sichern und Adresse+1
            MOVE.W  (A0)+,D0       |laden und Adresse+1
            BRA loop
       End:
            ROL.W   #8,D0          |letztes Rotieren
            MOVE.W  D0,(A1)        |letztes Wort sichern
            MOVEM.L (SP)+,A0-A1    |Rueckladen der Register
            RTS
```

Teil II
Betriebssysteme

2 Einführung Betriebssysteme

Das Betriebssystem hat die Aufgabe eine optimale Nutzung der Betriebsmittel zu gewährleisten (CPU, Speicher, Platte,...).
Ausserdem sollte es eine abstrakte Maschine schaffen, welche die Hardware der echten Machine abstrahiert und so den Anwendungen eine homogene Zugriffsschicht bietet.

2.1 Arten von Betriebssystemen

1. Stapelbetrieb - Batch Processing

   Kaum Systemeingriff - oberste Maxime: CPU Ausnutzung optimal

2. Dialogbetrieb - Time Sharing

   Interaktion mit dem System jederzeit möglich (Terminal)

   (a) Auskunftssysteme
   selten geänderte Datenbanken (Fahrplanauskunft, Google, ...)

   (b) Transaktionssysteme
   Datenbankinhalte werden ständig geändert (SWIFT, Buchungssysteme)

3. Echtzeitbetrieb - Realtime

   Steuerungssysteme

3 Speicherverwaltung

Die Speicherverwaltung ist neben dem Prozess-scheduling die zweite große Schicht eines modernen BS.
Das Problem: Die Prozesse eines Systems benötigen eventuell mehr Speicher als das system physikalisch zu Verfügung hat. Ausserdem müssen Berechtigungen beim Zugriff geprüft werden, die Benutzung von Shared Memory soll zur Interprozesskommunikation möglich sein, etc.
Lösung: Die Prozesse benutzen virtuelle Speicheradressen die per Hardwareunterstützung (MMU) durch das BS auf den physikalischen Speicher und 'Hintergrundspeicher' abgebildet werden.
Der Hintergrundspeicher kann dabei auf unterschiedliche Weise implementiert werden: mit oder ohne Seitenverwaltung, etc.
Bei der Verwaltung müssen also nicht nur die freien Speicherbereiche sinnvoll verwaltet werden sondern auch das Ein- und Auslagern aus dem Hintergrundspeicher gesteuert werden.

3.1 Strategien für die Speicherzurodnung

1. Ersetzungsstrategie (Replacement)

   Legt fest, welche Bereiche aus dem Hauptspeicher entfernt werden, erzeugt freien Speicher

2. Ladestrategie (Fetch)

   Legt fest wann Daten in den Speicher geladen werden - z.B. auf Anforderung oder im Vorraus

   Ist für Seiten- und und nicht Seitenverwaltete Systeme praktisch gleich.

3. Belegungsstrategie (Placement)

   Legt fest wo Daten im Speicher abgelegt werden, wählt aus den freien Bereichen aus

3.1.1 Belegungsstrategien in nicht seitenverwalteten Systemen

Segmente (unterschiedlicher Länge) werden ständig im Hauptspeicher belegt und wieder freigegeben, der Belegungsalgorithmus hat die Freispeicherliste zu pflegen.
best fit - es wird das kleinste freie Speicherstück belegt, das gross genug ist den Angeforderten Speicher zu liefern - es gibt möglichst kleinen Verschnitt, große Stücke bleiben für große Anforderungen erhalten.
Nachteil: kleinere Stücke des Verschnitts sind eventuell zu klein um sie noch zu nutzen.
worst fit - das größte freie Speicherstück wird belegt, damit ist der Verschnitt für kleinere Belegungen noch benutzbar.
first fit - die Freispeicherliste ist nach Adressen geordnet, das erste passende Stück wird belegt, der Einstiegspunkt in die Liste kann nach jeder Belegung hinter die aktuell belegte Stelle gesetzt werden um Suchzeiten zu verringern.
Simulationsergebnisse: First fit ist schneller als Best Fit und beide sind besser als Worst Fit. Alle haben das Problem der Fragmentierung - Abhilfe: Speicherverdichtung.

3.1.2 Belgungsstrategien seitenverwalteter Systeme

Viel einfacher, da Seiten feste Längen haben, keine Fragmentierung bei den Belegungen, jedoch interne Fragmentierung wenn Prozesse ihre Speicherseiten nicht voll benutzen.

3.1.3 Ersetzungsstrategien - siehe Script

SIEHE SCRIPT S.32

4 Semaphore-Beispiele

4.1 Beispiel: Baustelle und Autos

Gegeben sei eine Baustelle, bei der die Autos nur abwechselnd von Links oder Rechts kommen dürfen. Wartet kein Auto darf eine Baustellenfahrzeug die Strasse überqueren.

```
von rechts              von links                Baufahrzeug

  lock.P                lock.P                    while(warten){
   counter++              counter++                  lock.P
  lock.V                lock.V                     if (counter==0)
                                                     warten=false
  vonRechts.P           vonLinks.P                 else
   //rübermachen          //rübermachen              lock.V
  vonLinks.V            vonRechts.V                fi
                                                  }

                                                  //rübermachen
  lock.P                lock.P
   counter--              counter--                lock.V
  lock.V                lock.V
```

Hinweis: Die lock-Semaphore wird auch zum Strassensperren verwendet, sonst käme noch eine Semaphore hinzu (Vorhalt).

Teil III
Java

5 Leichtgewichtige Prozesse / Threads in Java

5.1 Thread-Beispiel

Aufruf der Threads aus einer statischen Methode heraus:

Listing 3: Info4.Threads.Simple.Main

```
1  /*
   * Created on 22.07.2006
   */
  package Info4.Threads.Simple;

6  public class Main {

      public static void main(String[] args) {

         System.out.println("erzeuge ThreadA");
11
         ThreadA ta = new ThreadA();

         ta.start();

16       System.out.println("erzeuge ThreadB");


         /* Das folgende ist die Loesung, falls diese Klasse
          * von einer anderen Klasse mit extends abgeleitet wird!
21        * -> keine Mehrfachvererbung in Java
          */
         Runnable tb = new ThreadB();
         Thread tb_thread = new Thread(tb);
         tb_thread.start();
26
         System.out.println("Elternprozess terminiert...");

      }

31 }
```

Im folgenden der Quellcode für einen einfachen Hintergrundprozess:

Listing 4: Info4.Threads.Simple.ThreadA

```
/*
 * Created on 22.07.2006
 */
package Info4.Threads.Simple;

public class ThreadA extends Thread {

    public void run(){
        System.out.println("Hier ist das Kind aus ThreadA...!");
    }
}
```

Hier eine weitere Methode zum Erzeugen eines Threads:

Listing 5: Info4.Threads.Simple.ThreadB

```
/*
 * Created on 22.07.2006
 */
package Info4.Threads.Simple;

public class ThreadB implements Runnable {

    public void run(){
        System.out.println("Hier ist das Kind aus ThreadB...!");
    }

}
```

6 Monitore in Java

Monitore haben gegenüber Semaphoren einige Vorteile. Vor allem sind sie aber leichter zu benutzen, da sie intuitiv besser verständlich sind.
Monitore beinhalten im Gegensatz zu Semaphoren den Kritischen Abschnitt in sich und verbergen ihn so vor der 'Manipulation' von aussen.
Die Synchronisation bleibt so vor dem Anwender komplett verborgen, da die Zugriffsmethoden implizit im gegenseitigen Ausschluss realisiert sind.
Im Gegensatz zu Semaphoren kann man hier qasi das paarweise Auftreten der lock und unlock 'Anweisungen' prüfen. Eine Garantie gegen Deadlocks ist das dennoch nicht!
Blockiert ein Thread innerhalb eines Monitors durch wait() wird der Monitor automatisch freigegeben, da es sonst zum Deadlock kommen muss - nur eine Monitorfunktion kann den Monitor freigeben (signal) - die aber solange er belegt ist von keinem Thread aufgerufen werden kann.

6.1 gegenseitiger Ausschluss

In Java besitzt jedes Objekt implizit eine Ereignisvariable, auf der die Methoden notify und wait aufgerufen werden können. Spezialfall: NotifyAll.
Im folgenden wird eine Race-Condition provoziert - ohne die synchronized deklaration der Kasse ergibt sich folgende Ausgabe:

```
Zahle ein...
Anteil Schecks:200.0%
Anteil Schecks:66.66666666666666%
```

Im Gegensatz dazu ergibt der hier abgedruckte Quellcode folgende Ausgabe:

```
Zahle ein...
Anteil Schecks:66.66666666666666%
```

Listing 6: Info4.Monitore.Ausschluss.Main

```
/*
 * Created on 23.07.2006
 */
package Info4.Monitore.Ausschluss;

public class Main {

    public static void main(String[] args) throws InterruptedException{


        Kasse k = new Kasse();
        Concurrent c = new Concurrent();
        c.setKasse(k);
        c.start();

        Thread.sleep(1000);
        System.out.println("Zahle ein...");
        k.einzahlen("Bar",100);
        k.einzahlen("Scheck",200);
        Thread.sleep(1000);

        c.setKasse(null);

    }

}
```

Listing 7: Info4.Monitore.Ausschluss.Kasse

```
/*
 * Created on 23.07.2006
 */
package Info4.Monitore.Ausschluss;

public class Kasse {

   private int bestand=0;
   private int schecks=0;

   public synchronized void einzahlen(String zahlungsmittel, int summe) {

      if (zahlungsmittel.compareToIgnoreCase("scheck")==0){
         schecks+=summe;
      }
      //Zeit fuer eien Race-Condition ;)
      try{
      Thread.sleep(500);
      }catch (InterruptedException i){}

      bestand +=summe;

   }

   public synchronized double getScheckanteil(){
      if (bestand==0) return 0;
      return ((1d*schecks)/(bestand*1d))*100;
   }

}
```

Listing 8: Info4.Monitore.Ausschluss.Concurrent

```
/*
 * Created on 23.07.2006
 */
package Info4.Monitore.Ausschluss;

public class Concurrent extends Thread {

   private Kasse k;

   public void setKasse(Kasse k) {
      this.k = k;
   }

   public void run(){
      double anteil=0;
      double oldanteil=0;

      while (k != null){
         if (k==null) return; //while zu langsam...
         anteil=k.getScheckanteil();
         if (anteil != oldanteil){
         System.out.println("Anteil Schecks:"+anteil+"%");
         oldanteil=anteil;
         }
      }
   }
}
```

6.2 Exklusive Betriebsmittelbelegung

Im folgenden eine Klasse, die eine exklusive Belegung eines Betriebsmittels ermöglicht. Im Prinzip eine primitve Semaphore - ohne Moeglichkeit der initialisierung!
Der Aufrufende Thread schlaeft, solange das Betriebsmittel nicht frei ist.

Listing 9: Info4.Monitore.Betriebsmittel.BMonitor

```
/*
 * Created on 23.07.2006
 */
package Info4.Monitore.Betriebsmittel;

public class BMonitor {

   boolean belegt = false;

   public synchronized void belegen(){
      while (belegt) {
          try{
            this.wait();
          } catch (InterruptedException ie){}
      }
      belegt = true;
   }

   public synchronized void freigeben(){
      belegt=false;
      this.notifyAll();
   }

}
```

Hier der Thread, der versucht das Betriebsmittel zu bekommen. Zur veranschaulichung werden 10 parallel gestartet. Damit sich das ganz nicht so deterministisch verhält wird Math.Random mit herangezogen, um zufällige Wartezeiten zu erzielen.

Listing 10: Info4.Monitore.Betriebsmittel.Beleger

```
/*
 * Created on 23.07.2006
 */
package Info4.Monitore.Betriebsmittel;

public class Beleger extends Thread {

   BMonitor monitor = null;

   private boolean beenden = false;

   public void run() {
      int t = (int) (Math.random() * 100);

      while (!beenden) {
         schlaf(t * 2);
         try {
            System.out.println(this.getName() + ": will Betriebsmittel");
            monitor.belegen();
            System.out.println("--> " + this.getName()
                  + ": habe Betriebsmittel");
            schlaf(t * 5);
            System.out.println(this.getName() + ": gebe Betriebsmittel frei");
            monitor.freigeben();
            schlaf(t * 1);
         } catch (NullPointerException n) {/*tritt auf wenn monitor==null*/
         }

      }
      System.out.println(this.getName() + ": terminated");
   }

   private void schlaf(int d) {
      try {
         Thread.sleep(d);
      } catch (InterruptedException i) {
      }
   }

   public void beenden() {
      this.beenden = true;
   }

   public void setMonitor(BMonitor monitor) {
      this.monitor = monitor;

   }

}
```

7 Sockets in Java

7.1 Simpler zeilenbasierter Server/Client

Im folgenden der Quellcode für einen einfachen Client und Server, die Zeilenweise Nachrichten austauschen. Der Server schickt die Nachricht rückwärts wieder zurück.
Wichtiger Hinweis: Reader benutzen, da man sonst mit der Unicode-Codierung u.U. Probleme bekommt (Strings sind 2 Byte breit, Nullbytes werden nicht dargestellt...)

Listing 11: Info4.Sockets.SimpleString.Server

```
/*
 * Created on 22.07.2006
 */
package Info4.Sockets.SimpleString;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.EOFException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {

    /**
     * @param args
     */
    public static void main(String[] args) {
        short port = 4711;

        ServerSocket serversocket = null;

        //socket oeffnen und lauschen:
        try {
            serversocket = new ServerSocket(port);
        } catch (IOException ioe) {
            System.err
                    .println("error binding socket to port " + port + ": " + ioe);
            System.exit(1);
        }

        // nimm 10 mal eine neue verbindung an:
        for (int i = 0; i < 10; i++) {
            System.out.println("warte auf Verbindung nummer " + (1 + i));
            try {
                Socket connection = serversocket.accept();
                handleClient(connection);
            } catch (IOException ioe) {
                System.err.println("error in client communication: " + ioe);
            }
        }
        //schliessen des server-sockets und freigeben des ports:
        try {
            serversocket.close();
        } catch (IOException ioe) {
            System.err.println("error closing socket: " + ioe);
        }
    }

    public static void handleClient(Socket connection) {
```

```
        try {
            BufferedReader in = new BufferedReader(new InputStreamReader(
                    connection.getInputStream()));
            BufferedWriter out = new BufferedWriter(new OutputStreamWriter(
                    connection.getOutputStream()));

            try {
                String line = null;
                while (true) {
                    line = in.readLine();
                    if (line == null)
                        break;
                    System.out.println("received: " + line);
                    String newline = "";
                    for (int i = 0; i < line.length(); i++) {
                        newline = line.charAt(i) + newline;
                    }
                    System.out.println("sending:  " + newline);
                    out.write(newline + "\r\n");
                    out.flush();
                }

            } catch (IOException ioe) {
                System.out.println("connection closed due to IOException:" + ioe);
            }

            in.close();
            out.close();

        } catch (IOException ioe) {
            System.err.println("Error in client connection:" + ioe);
            return;
        }

    }
}
```

Listing 12: Info4.Sockets.SimpleString.Client

```
/*
 * Created on 22.07.2006
 */
package Info4.Sockets.SimpleString;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.Socket;

public class Client {

    /**
     * @param args
     */
    public static void main(String[] args) {
        String host = "localhost";
        short port = 4711;

        try {
```

```
            Socket connection = new Socket(host, port);
            try {
                BufferedReader in = new BufferedReader(new InputStreamReader(
                        connection.getInputStream()));
                BufferedWriter out = new BufferedWriter(new OutputStreamWriter(
                        connection.getOutputStream()));

                out.write("Dies ist ein Test\r\n");
                out.flush();
                String read = in.readLine();
                System.out.println(read);

            } catch (IOException ioe) {
                System.err.println("Error in server connection: " + ioe);
                System.exit(1);
            }

        } catch (IOException ioe) {
            System.err.println("Can not open connection:" + ioe);
            System.exit(1);
        }

    }
}
```

7.2 Server/Client für primitive Datentypen

Es ist besonders zu beachten, dass man nicht wild die unterschiedlichen read-Methoden aufrufen sollte, da die Daten Byteweise übertragen werden und es dann zu 'Mißverständnissen' kommen kann.

Listing 13: Info4.Sockets.PrimitiveInt.Server

```
/*
 * Created on 22.07.2006
 */
package Info4.Sockets.PrimitiveInt;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {

    /**
     * @param args
     */
    public static void main(String[] args) {
        short port = 4711;

        ServerSocket serversocket = null;

        //socket oeffnen und lauschen:
        try {
            serversocket = new ServerSocket(port);
        } catch (IOException ioe) {
            System.err
                    .println("error binding socket to port " + port + ": " + ioe);
            System.exit(1);
        }
```

```
        // nimm 10 mal eine neue verbindung an:
        for (int i = 0; i < 10; i++) {
34          System.out.println("warte auf Verbindung nummer " + (1 + i));
            try {
                Socket connection = serversocket.accept();
                handleClient(connection);
            } catch (IOException ioe) {
39              System.err.println("error in client communication: " + ioe);
            }
        }
        //schliessen des server-sockets und freigeben des ports:
        try {
44          serversocket.close();
        } catch (IOException ioe) {
            System.err.println("error closing socket: " + ioe);
        }
    }
49
    public static void handleClient(Socket connection) {

        try {

54          // In und Output Streams erzeugen:
            DataOutputStream dos = new DataOutputStream(connection
                    .getOutputStream());
            DataInputStream dis = new DataInputStream(connection.getInputStream());

59          int v;
            int ov = -1;

            try {
                while (true) {
64                  v = dis.readInt(); //lies integer ein

                    if (ov == -1) {
                        ov = v;
                    } else {
69                      ov = (ov + v) / 2;
                    }

                    dos.writeInt(ov); //und schreibe ihn leicht veraendert zurueck
                    System.out.println(v + " --> " + ov);
74              }
            } catch (EOFException eof) {
                System.out.println("end of connection");
            }

79          dis.close();
            dos.close();
        }

        catch (IOException ioe) {
84          System.err.println("Error in client connection" + ioe);
        }
    }
}
```

Listing 14: Info4.Sockets.PrimitiveInt.Client

```
/*
 * Created on 22.07.2006
 */
package Info4.Sockets.PrimitiveInt;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

public class Client {

   /**
    * @param args
    */
   public static void main(String[] args) {

      String server = "localhost";
      short port = 4711;

      Socket connection = null;

      try {
         connection = new Socket(server, port);

      } catch (IOException ioe) {
         System.err.println("can not connect: " + ioe);
         System.exit(1);
      }

      int v;
      int nv;

      try {
         DataOutputStream dos = new DataOutputStream(connection
               .getOutputStream());
         DataInputStream dis = new DataInputStream(connection.getInputStream());

         for (int i = 0; i < 10; i++) {
            v = (int) (Math.random() * Integer.MAX_VALUE) / 2;
            dos.writeInt(v);
            nv = dis.readInt();
            System.out.println(i + ":   " + v + " --> " + nv);

         }

      } catch (IOException ioe) {
         System.err.println("Error in server communication: " + ioe);
      }

   }

}
```

7.3 Server/Client um Objekte zu transportieren

Obacht: Alles muss Serialisiert werden - das hat nichts mit RMI zu tun!
Das bedeutet, das Objekt wird komplett uebertragen - das Gegenueber muss die Klassendefinition vorliegen haben - sonst gibt es eine ClassNotFound Exception!

Listing 15: Info4.Sockets.ObjectTransport.Server

```
/*
 * Created on 22.07.2006
 */
package Info4.Sockets.ObjectTransport;

import java.io.EOFException;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {

   /**
    * @param args
    */
   public static void main(String[] args) {
      short port = 4711;

      ServerSocket serversocket = null;

      //socket oeffnen und lauschen:
      try {
         serversocket = new ServerSocket(port);
      } catch (IOException ioe) {
         System.err
               .println("error binding socket to port " + port + ": " + ioe);
         System.exit(1);
      }

      // nimm 10 mal eine neue verbindung an:
      for (int i = 0; i < 10; i++) {
         System.out.println("warte auf Verbindung nummer " + (1 + i));
         try {
            Socket connection = serversocket.accept();
            handleClient(connection);
         } catch (IOException ioe) {
            System.err.println("error in client communication: " + ioe);
         }
      }
      //schliessen des server-sockets und freigeben des ports:
      try {
         serversocket.close();
      } catch (IOException ioe) {
         System.err.println("error closing socket: " + ioe);
      }
   }

   public static void handleClient(Socket connection) {

      try {

         // In und Output Streams erzeugen:
         ObjectOutputStream dos = new ObjectOutputStream(connection
```

```
                .getOutputStream());
57          ObjectInputStream dis = new ObjectInputStream(connection
                .getInputStream());

            int v;
            int ov = -1;
62
            try {
               while (true) {

                  Object o = dis.readObject();
67                System.out.println("Object received:" + o.toString());
                  dos.writeObject(o);

               }
            } catch (EOFException eof) {
72             System.out.println("end of connection" + eof);
            } catch (ClassNotFoundException cnf) {
               System.out.println("Class not found!" + cnf);
            }

77          dis.close();
            dos.close();
         }

         catch (IOException ioe) {
82          System.err.println("Error in client connection" + ioe);
         }
      }
   }
```

Listing 16: Info4.Sockets.ObjectTransport.Client

```
/*
 * Created on 22.07.2006
 */
package Info4.Sockets.ObjectTransport;
5
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
10
public class Client {

   /**
    * @param args
15  */
   public static void main(String[] args) {

      String server = "localhost";
      short port = 4711;
20
      Socket connection = null;

      try {
         connection = new Socket(server, port);
25
      } catch (IOException ioe) {
         System.err.println("can not connect: " + ioe);
         System.exit(1);
      }
```

```
30
        try {
            ObjectInputStream dis = new ObjectInputStream(connection
                    .getInputStream());
            ObjectOutputStream dos = new ObjectOutputStream(connection
35                  .getOutputStream());

            for (int i = 0; i < 10; i++) {

                dos.writeObject(new String("Hallo Nr. " + i));
40              Object o = dis.readObject();
                System.out.println(o);

            }

45      } catch (IOException ioe) {
            System.err.println("Error in server communication: " + ioe);
            ioe.printStackTrace();
        } catch (ClassNotFoundException cnf) {
            System.err.println("Error in server communication: " + cnf);
50          cnf.printStackTrace();
        }

    }

55 }
```

7.4 Ein etwas groesseres Beispiel: Webserver und Client

Listing 17: Info4.Sockets.HTTP.HTTPClient

```
/*
 * Created on 24.11.2005
 *
 * TODO To change the template for this generated file go to
5 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author fabian
10 *
 * HTTPClient
 */
package Info4.Sockets.HTTP;

15 import java.net.Socket;
import java.util.StringTokenizer;
import java.io.*;

public class HTTPClient {
20

    private static int returnCode = 0;

    private static String reasonPhrase = null;

25  private static void statusRequest(String line) {
        //Auswerten der Html Zeile und des Request
        StringTokenizer st = new StringTokenizer(line);
        String token = null;
        if (st.countTokens() >= 3) {
30          token = st.nextToken();
```

```
            //html/V1.0
            token = st.nextToken();
            returnCode = Integer.parseInt(token);
            reasonPhrase = st.nextToken();
            while (st.hasMoreTokens())
                reasonPhrase += " " + st.nextToken();
        } else {
            System.out.println("unknown");
            return;
        }
    }

    public static void main(String[] args) {
        String host = "localhost";
        if (args.length != 1) {
            System.out
                    .println("Please enter your File. / for Index.htm else /documentroot");
            return;
        }
        String path = args[0];
        Socket socket = null;
        try {
            socket = new Socket(host, 4444);
            System.out.println("Server is connected to Client");
            BufferedWriter bos = new BufferedWriter(new OutputStreamWriter(socket
                    .getOutputStream()));
            HTTPInputStream his = new HTTPInputStream(socket.getInputStream());

            bos.write("GET " + path + " HTTP/1.0 \r\n");
            bos.write("header1 \r\n");
            bos.write("header2 \r\n");
            bos.write("header3 \r\n");
            bos.write("\r\n");
            bos.write("blub");
            bos.flush();
            String line = his.readLine();
            System.out.println("Input: " + line);
            statusRequest(line);

            //header
            try {
                while ((line = his.readLine()) != "\r\n") {
                    System.out.println("Input: " + line);
                }
            } catch (IllegalStateException ise) {
                System.out.println("Header End!");
            }

            if (returnCode == 200) {
                //Datei empfangen
                int i;
                while ((i = his.read()) != -1) {
                    System.out.println(i);
                }
            } else {
                System.out.println(reasonPhrase + " - ReturnCode: " + returnCode);
            }

            bos.flush();
            bos.close();
            his.close();
            System.out.println("client is closed");
```

```
        } catch (IOException ioe) {
            System.out.println("␣**␣IOFehler:␣\"" + ioe
                    + "\"␣-␣Programm␣is␣shutting␣down");
            return;
        } finally {
            try {
                socket.close();
            } catch (IOException ioe) {
            }
        }

    }
}
```

Listing 18: Info4.Sockets.HTTP.HTTPServer

```
/*
 * Created on 24.11.2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 Create by Fabian
 */

package Info4.Sockets.HTTP;

import java.io.*;
import java.net.Socket;
import java.net.ServerSocket;
import java.util.StringTokenizer;

public class HTTPServer {

    private static int returnCode = 0;

    private static String reasonPhrase = null;

    private static void createReasonPhrase() {
        //erstellt Reason Phrase anhand des aktuellen ReturnCodes
        switch (returnCode) {
        case (200):
            reasonPhrase = "OK";
            break;
        case (400):
            reasonPhrase = "Bad␣Request";
            break;
        case (401):
            reasonPhrase = "Unauthorized";
            break;
        case (404):
            reasonPhrase = "File␣Not␣Found";
            break;
        default:
            reasonPhrase = "unknown␣Reason";
        }
    }

    private static String statusRequest(String line) {
        //Wertet den HTML-Request aus und gibt das Ziel aus.
        StringTokenizer st = new StringTokenizer(line);
        String token = null;
        String file = null;
```

```
        //String method=null;

        returnCode = 200;
        if (st.countTokens() == 3) {
            token = st.nextToken();
            //if (token.compareToIgnoreCase("get")==0) method="get";
            if (token.compareToIgnoreCase("get") != 0) {
                returnCode = 400;
                return null;
            }

            file = st.nextToken();
            if (file.trim().compareTo("/") == 0 || file.trim().compareTo("") == 0) {
                file = "/index.htm";
            }
            /*     if (file.startsWith("/"))
             file="documentroot"+file;
             else
             file="documentroot/"+file;
             */
            File f = new File(file);
            if (!f.exists())
                returnCode = 404;
            else if (!f.canRead())
                returnCode = 401;
            token = st.nextToken();
            if (token.compareToIgnoreCase("http/1.0") != 0
                    && token.compareToIgnoreCase("http/1.1") != 0)
                returnCode = 400;
        } else
            returnCode = 400;
        return file;
    }

    public static void main(String[] args) {

        ServerSocket ssocket = null;
        Socket socket = null;
        String file;
        try {
            ssocket = new ServerSocket(4444);
        } catch (IOException ioe) {
            System.out.println("**IOFehler:\"" + ioe
                    + "\"!-Program is shutting down");
            return;
        }
        while (true) {

            try {
                while (true) {
                    System.out.println("server ready");
                    socket = ssocket.accept();
                    System.out.println("client connected");

                    //regular output
                    HTTPInputStream his = new HTTPInputStream(socket
                            .getInputStream());
                    BufferedOutputStream bos = new BufferedOutputStream(socket
                            .getOutputStream());
                    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(
                            bos));
```

```
            //Statuszeile
            String line = his.readLine();
            System.out.println("Input: " + line);
            file = statusRequest(line);
            createReasonPhrase();
            System.out.println("Send back: HTTP/1.0 " + returnCode + " "
                  + reasonPhrase + " \r\n");
            bw
                  .write("HTTP/1.0 " + returnCode + " " + reasonPhrase
                        + " \r\n");
            bw.write("header1: value1\r\n");
            bw.write("header2: value2\r\n");
            bw.write("header3: value3\r\n");
            bw.write("header4: value4\r\n");
            bw.write("\r\n");
            bw.flush();

            if (returnCode >= 200 && returnCode < 300) {

               //header
               try {
                  while ((line = his.readLine()) != "\r\n") {
                     System.out.println("Input: " + line);
                  }
               } catch (IllegalStateException ise) {
                  System.out.println("Header finish!");
               }

               //Datei schreiben
               System.out.println("Write File \"" + file + "\"...");
               BufferedInputStream br = new BufferedInputStream(
                     new FileInputStream(file));
               int b;
               while ((b = br.read()) != -1) {
                  //System.out.println(b);
                  bos.write(b);
               }
               bos.flush();
               bw.close();
               br.close();
            } else {

               bw.write("<html><body><h1>Error " + returnCode
                     + "</h1><br><br><h4>" + reasonPhrase
                     + "</h4></body></html>");

               bw.flush();
               bw.close();
            }

            bos.close();
            his.close();
            System.out.println("");
         }
      } catch (IOException ioe) {
         System.out.println(" **IOFehler: \"" + ioe + "\"!");
      } finally {
         System.out.println("connection is closed");
         try {
            socket.close();
         } catch (IOException ioe) {
         }
```

```
            }
        }

    }
}
```

Listing 19: Info4.Sockets.HTTP.HTTPInputStream

```
/* ================================================================
 * HTTPInputStream.java
 * ================================================================
 */

package Info4.Sockets.HTTP;

import java.io.BufferedInputStream;
import java.io.FilterInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;

/**
 * This class simplifies reading a HTTP requests or response from an
 * InputStream. The HTTP request-line or status-line and the HTTP
 * header can be read line by line. Thereafter, the HTTP content must
 * be read bytewise.<br/>
 * For further information on HTTP see RFC 1945 and RFC 2616.<br/>
 * This class contains a new method readline() to read HTTP header line.
 * Other methods are not overwritten and can be used as usual.
 * <i>Note: This class already uses a BufferedInputStream.</i>
 *
 * @author Olaf Goerlitz
 */
public final class HTTPInputStream extends FilterInputStream {

   private StringBuffer buffer = new StringBuffer();

   private boolean endOfHeader = false;

   /**
    * Create a new HTTP input stream.
    * @param in the InputStream to read from
    */
   public HTTPInputStream(InputStream in) {
      super(new BufferedInputStream(in));
   }

   /**
    * Reads a line of US-ASCII encoded text from the HTTP header
    * (including request and status line). A HTTP header line is
    * terminated by a carriage return followed immediately by a
    * linefeed ("CRLF").<br/>
    * This method will throw an <code>IllegalStateException</code>
    * if the end of the header had already been reached. To detect
    * the end of the header one should not catch the exception but
    * test for an empty String. (See RFC 1945 and RFC 2616).
    *
    * @return a String containing the read header line.
    *         Line termination characters (CRLF) are not included.
    * @throws IllegalStateException if the end of the HTTP header
    *         had already been reached
    * @throws IOException if the HTTP header is not correct or
```

```
     *          if an I/O error occurs
     */
    public String readLine() throws IOException {
        if (endOfHeader) {
            throw new IllegalStateException(
                    "YOU CAN NOT READ TEXT LINES AFTER THE END OF THE HTTP HEADER");
        }

        buffer.delete(0, buffer.length());
        int read;
        while ((read = read()) != -1) {
            // HTTP header lines have to be terminated by "CRLF"
            if (read == '\r') { // CR
                if (read() != '\n') { // LF
                    throw new IOException("MALFORMED HTTP HEADER (NEED CRLF)");
                }
                // an empty line (just CRLF) indicates end of header
                if (buffer.length() == 0) {
                    endOfHeader = true;
                }
                return buffer.toString();
            }
            buffer.append((char) read);
        }
        // end of input stream - should not happen
        throw new IOException("INCOMPLETE HTTP HEADER (END OF STREAM)");
    }
}
```

8 Remote Method Invocation / RMI in Java

Bei RMI in Java werden Referenzen auf entfernte Objekte per Netz übertragen. Diese Stubs (Skeletons gibt es in neuen Java-Versionen nicht mehr) sorgen nun dafür, dass bei einem Methodenaufruf dieser per Netz auf der entfernten Maschine ausgeführt wird.
Für den Anwender (Entwlickler) gibt es im Unterschied zu einem lokalen Methodenaufruf eigentlich nur den Unterschied, dass es IOExceptions und RemoteExceptions geben kann.
Ausserdem muss das Objekt von Typ RemoteObject abgeleitet werden.

8.1 ein kurzes Beispiel

Im folgenden der Quellcode für einen einfachen Client und Server:

Listing 20: Info4.RMI.Simple.Server

```
/*
 * Created on 22.07.2006
 */
package Info4.RMI.Simple;

import java.rmi.ConnectException;
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;

public class Server {

   /**
    * @param args
    */
   public static void main(String[] args) {

      int rmiport=1099;

      /*
       * Registry starten -
       * gibt exception und Abbruch wenn schon
       * eine laeuft
      */

      try {
         LocateRegistry.createRegistry(rmiport);
      } catch (Exception e) {
         System.err.println("error starting registry:");
         System.err.println(e);
         System.exit(1);
      }

      registerService("localhost", rmiport, "ReverseService");

   }

   private static void registerService(String host, int port, String ServiceName) {

      String name = "//" + host + ":" + port + "/" + ServiceName;
      try {

         MyRemIface remobj = new MyRemObj();
         Naming.rebind(name, remobj);
         System.out.println("RemoteObject bound, waiting for connections...");

      } catch (ConnectException e) {
         if (e.getCause() instanceof java.net.ConnectException) {
```

```
48              System.err.println("RMI registry not running or wrong host/port: "
                        + host + ":" + port);
                System.exit(1);
            } else {
                System.err.println("Error connecting to " + host + ":" + port);
53              System.err.println("RemoteObject exception: " + e.getMessage());
                System.exit(1);
            }
        }

58      catch (Exception e) {
            if (e.getCause() instanceof java.net.NoRouteToHostException) {
                System.err.println("No Route to RMI-Registry\n"
                        + " - check your network/IP settings!");
                System.exit(1);
63          }
            System.err.println("Exception setting up RMI: " + e.getMessage());
            e.printStackTrace();
        }

68  }

}
```

Listing 21: Info4.RMI.Simple.Client

```
/*
 * Created on 22.07.2006
 */
package Info4.RMI.Simple;
5
import java.rmi.ConnectException;
import java.rmi.Naming;

public class Client {
10
    public static void main(String[] args) {
        String host = "localhost";
        int rmiport = 1099;
        String ServiceName = "ReverseService";
15
        try {
            String name = "//" + host + ":" + rmiport + "/" + ServiceName;
            System.out.println("looking up "+name);
            Object remobj = Naming.lookup(name);
20
            if (remobj!=null && remobj instanceof MyRemIface){


            MyRemIface remIf = (MyRemIface) remobj;
25
            String reverse = remIf.getReverse("Dies ist ein Test!");
            System.out.println(reverse);
            }
            System.out.println("bye");
30
        } catch (ConnectException e) {
            if (e.getCause() instanceof java.net.ConnectException) {
                System.err.println("RMI registry not running or wrong host/port: "
                        + host + ":" + rmiport);
35              System.exit(1);
            } else {
```

```
            System.err.println("Error connecting to" + host + ":" + rmiport);
            System.err.println("ComputeEngine exception: " + e.getMessage());
            System.exit(1);
        }
    } catch (Exception e) {
        System.err.println("exception: " + e.getMessage());
        e.printStackTrace();
    }

}
}
```

Dazu müssen folgende Klassendefinitionen natuerlich auf Client und Server vorhanden sein. Bei RMI besteht sonst die Moeglichkeit, eine Codebase in Form einer URL anzugeben - der Classloader versucht dann per HTTP die Klassen nachzuladen - einen entsprechenden SecurityManager vorausgesetzt.

Listing 22: Info4.RMI.Simple.MyRemIface - das Remote-Interface

```
/*
 * Created on 22.07.2006
 */
package Info4.RMI.Simple;

import java.rmi.Remote;
import java.rmi.RemoteException;


public interface MyRemIface extends Remote {

    public String getReverse(String s) throws RemoteException;

}
```

Listing 23: Info4.RMI.Simple.MyRemObj - ein mögliches RemoteObject

```
/*
 * Created on 22.07.2006
 */
package Info4.RMI.Simple;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class MyRemObj  extends UnicastRemoteObject implements MyRemIface{

    public MyRemObj() throws RemoteException{
        super();
    }

    public String getReverse(String s) throws RemoteException {
        if (s==null) return s;

        String result = "";
        for (int i=0;i<s.length();i++)
        result = s.charAt(i)+result;

        return result;
    }

}
```

A Motorola 68k- Informationen - plaintext

A.1 Motorola 68k Quickreference

Motorola 68000 Instruction Set.

Instruction	Description	Assembler Syntax	Data Size	Condition Codes X N Z V C
ABCD	Add BCD with extend	Dx,Dy -(Ax),-(Ay)	B--	* U * U *
ADD	ADD binary	Dn,<ea> <ea>,Dn	BWL	* * * * *
ADDA	ADD binary to An	<ea>,An	-WL	- - - - -
ADDI	ADD Immediate	#x,<ea>	BWL	* * * * *
ADDQ	ADD 3-bit immediate	#<1-8>,<ea>	BWL	* * * * *
ADDX	ADD eXtended	Dy,Dx -(Ay),-(Ax)	BWL	* * * * *
AND	Bit-wise AND	<ea>,Dn Dn,<ea>	BWL	- * * 0 0
ANDI	Bit-wise AND with Immediate	#<data>,<ea>	BWL	- * * 0 0
ASL	Arithmetic Shift Left	#<1-8>,Dy Dx,Dy <ea>	BWL	* * * * *
ASR	Arithmetic Shift Right	...	BWL	* * * * *
Bcc	Conditional Branch	Bcc.S <label> Bcc.W <label>	BW-	- - - - -
BCHG	Test a Bit and CHanGe	Dn,<ea> #<data>,<ea>	B-L	- - * - -
BCLR	Test a Bit and CLeaR	...	B-L	- - * - -
BSET	Test a Bit and SET	...	B-L	- - * - -
BSR	Branch to SubRoutine	BSR.S <label> BSR.W <label>	BW-	- - - - -
BTST	Bit TeST	Dn,<ea> #<data>,<ea>	B-L	- - * - -
CHK	CHecK Dn Against Bounds	<ea>,Dn	-W-	- * U U U
CLR	CLeaR	<ea>	BWL	- 0 1 0 0
CMP	CoMPare	<ea>,Dn	BWL	- * * * *
CMPA	CoMPare Address	<ea>,An	-WL	- * * * *
CMPI	CoMPare Immediate	#<data>,<ea>	BWL	- * * * *
CMPM	CoMPare Memory	(Ay)+,(Ax)+	BWL	- * * * *
DBcc	Looping Instruction	DBcc Dn,<label>	-W-	- - - - -
DIVS	DIVide Signed	<ea>,Dn	-W-	- * * * 0
DIVU	DIVide Unsigned	<ea>,Dn	-W-	- * * * 0
EOR	Exclusive OR	Dn,<ea>	BWL	- * * 0 0
EORI	Exclusive OR Immediate	#<data>,<ea>	BWL	- * * 0 0
EXG	Exchange any two registers	Rx,Ry	--L	- - - - -
EXT	Sign EXTend	Dn	-WL	- * * 0 0
ILLEGAL	ILLEGAL-Instruction Exception	ILLEGAL		- - - - -
JMP	JuMP to Affective Address	<ea>		- - - - -
JSR	Jump to SubRoutine	<ea>		- - - - -
LEA	Load Effective Address	<ea>,An	--L	- - - - -
LINK	Allocate Stack Frame	An,#<displacement>		- - - - -

LSL	Logical Shift Left	Dx,Dy	BWL	* * * 0 *
		#<1-8>,Dy		
		<ea>		
LSR	Logical Shift Right	...	BWL	* * * 0 *
MOVE	Between Effective Addresses	<ea>,<ea>	BWL	- * * 0 0
MOVE	To CCR	<ea>,CCR	-W-	I I I I I
MOVE	To SR	<ea>,SR	-W-	I I I I I
MOVE	From SR	SR,<ea>	-W-	- - - - -
MOVE	USP to/from Address Register	USP,An	--L	- - - - -
		An,USP		
MOVEA	MOVE Address	<ea>,An	-WL	- - - - -
MOVEM	MOVE Multiple	<register list>,<ea>	-WL	- - - - -
		<ea>,<register list>		
MOVEP	MOVE Peripheral	Dn,x(An)	-WL	- - - - -
		x(An),Dn		
MOVEQ	MOVE 8-bit immediate	#<-128.+127>,Dn	--L	- * * 0 0
MULS	MULtiply Signed	<ea>,Dn	-W-	- * * 0 0
MULU	MULtiply Unsigned	<ea>,Dn	-W-	- * * 0 0
NBCD	Negate BCD	<ea>	B--	* U * U *
NEG	NEGate	<ea>	BWL	* * * * *
NEGX	NEGate with eXtend	<ea>	BWL	* * * * *
NOP	No OPeration	NOP		- - - - -
NOT	Form one's complement	<ea>	BWL	- * * 0 0
OR	Bit-wise OR	<ea>,Dn	BWL	- * * 0 0
		Dn,<ea>		
ORI	Bit-wise OR with Immediate	#<data>,<ea>	BWL	- * * 0 0
PEA	Push Effective Address	<ea>	--L	- - - - -
RESET	RESET all external devices	RESET		- - - - -
ROL	ROtate Left	#<1-8>,Dy	BWL	- * * 0 *
		Dx,Dy		
		<ea>		
ROR	ROtate Right	...	BWL	- * * 0 *
ROXL	ROtate Left with eXtend	...	BWL	* * * 0 *
ROXR	ROtate Right with eXtend	...	BWL	* * * 0 *
RTE	ReTurn from Exception	RTE		I I I I I
RTR	ReTurn and Restore	RTR		I I I I I
RTS	ReTurn from Subroutine	RTS		- - - - -
SBCD	Subtract BCD with eXtend	Dx,Dy	B--	* U * U *
		-(Ax),-(Ay)		
Scc	Set to -1 if True, 0 if False	<ea>	B--	- - - - -
STOP	Enable & wait for interrupts	#<data>		I I I I I
SUB	SUBtract binary	Dn,<ea>	BWL	* * * * *
		<ea>,Dn		
SUBA	SUBtract binary from An	<ea>,An	-WL	- - - - -
SUBI	SUBtract Immediate	#x,<ea>	BWL	* * * * *
SUBQ	SUBtract 3-bit immediate	#<data>,<ea>	BWL	* * * * *
SUBX	SUBtract eXtended	Dy,Dx	BWL	* * * * *
		-(Ay),-(Ax)		
SWAP	SWAP words of Dn	Dn	-W-	- * * 0 0
TAS	Test & Set MSB & Set N/Z-bits	<ea>	B--	- * * 0 0
TRAP	Execute TRAP Exception	#<vector>		- - - - -
TRAPV	TRAPV Exception if V-bit Set	TRAPV		- - - - -
TST	TeST for negative or zero	<ea>	BWL	- * * 0 0
UNLK	Deallocate Stack Frame	An		- - - - -

```
Symbol   Meaning
------   -------

   *     Set according to result of operation
   -     Not affected
   0     Cleared
   1     Set
   U     Outcome (state after operation) undefined
   I     Set by immediate data

<ea>     Effective Address Operand
<data>   Immediate data
<label>  Assembler label
<vector> TRAP instruction Exception vector (0-15)
<rg.lst> MOVEM instruction register specification list
<displ.> LINK instruction negative displacement
...      Same as previous instruction
```

```
Addressing Modes                                    Syntax
----------------                                    ------

Data Register Direct                                  Dn
Address Register Direct                               An
Address Register Indirect                            (An)
Address Register Indirect with Post-Increment        (An)+
Address Register Indirect with Pre-Decrement         -(An)
Address Register Indirect with Displacement          w(An)
Address Register Indirect with Index                b(An,Rx)
Absolute Short                                         w
Absolute Long                                          l
Program Counter with Displacement                   w(PC)
Program Counter with Index                         b(PC,Rx)
Immediate                                             #x
Status Register                                       SR
Condition Code Register                               CCR

Legend
------
   Dn    Data Register        (n is 0-7)
   An    Address Register     (n is 0-7)
    b    08-bit constant
    w    16-bit constant
    l    32-bit constant
    x    8-, 16-, 32-bit constant
   Rx    Index Register Specification, one of:
            Dn.W  Low 16 bits of Data Register
            Dn.L  All 32 bits of Data Register
            An.W  Low 16 bits of Address Register
            An.L  All 32 bits of Address Register
```

```
         Condition Codes for Bcc, DBcc and Scc Instructions.
         ---------------------------------------------------
```

Condition Codes set after CMP D0,D1 Instruction.

Relationship	Unsigned	Signed
D1 < D0	CS - Carry Bit Set	LT - Less Than
D1 <= D0	LS - Lower or Same	LE - Less than or Equal
D1 = D0	EQ - Equal (Z-bit Set)	EQ - Equal (Z-bit Set)
D1 != D0	NE - Not Equal (Z-bit Clear)	NE - Not Equal (Z-bit Clear)
D1 > D0	HI - HIgher than	GT - Greater Than
D1 >= D0	CC - Carry Bit Clear	GE - Greater than or Equal
	PL - PLus (N-bit Clear)	MI - Minus (N-bit Set)
	VC - V-bit Clear (No Overflow)	VS - V-bit Set (Overflow)
	RA - BRanch Always	
DBcc Only -	F - Never Terminate (DBRA is an alternate to DBF)	
	T - Always Terminate	
Scc Only -	SF - Never Set	
	ST - Always Set	

Parts from "Programming the 68000" by Steve Williams. (c) 1985 Sybex Inc.
Parts from BYTE Magazine article.

Compiled by Diego Barros. e-mail : alien@zikzak.apana.org.au
Revision 2.1 22 May, 1994

A.2 Motorola 68k Adressierungsarten

```
; REFERENZTABELLE FÜR DIE 68000er PROGRAMMIERUNG

Synthetisch zusammengefaßt: Die Adressierungen:

 move.l #123,xxxx   ; Immediate: die Zahl 123 kommt sofort ins xxxx
 move.l xxxx,$50000   ; Absolut long
 move.l xxxx,$500.w   ; Absolut kurz (weniger als $7FFF)
 move.l xxxx,D0   ; Datenregister direkt
 move.l xxxx,A0   ; Adressregister direkt
 move.l xxxx,(A0)   ; Datenregister indirekt
 move.l xxxx,(A0)+   ; Adressregister indirekt mit Post-Inkrement
 move.l xxxx,-(A0)   ; Adressregister indirekt mit Pre-Dekrement
 move.l xxxx,$123(A0)   ; Adressregister indirekt mit Offset (Adressdistanz)
 move.l xxxx,$12(a0,d0.w)    ; Adressregister indirekt mit Offset und Index
 move.l Offset(PC),xxxx      ; Relativ zum PC mit Offset
 move.l Offset(PC,d0.w),xxxx ; Relativ zum PC mit OFFSET

---

* Die verschiedensten Adressierungsarten kann man in Befehlen mit
Datenquelle und Datenziel "mischen", z.B. "move.l -(A0),12(a0,d3.l)".

---

* Die Dezimalzahlen werden von keinem Symbol angeführt (z.B. 123), die
Hexadezimalzahlen von einem $ (z.B. $1a0). Hexzahlen enthalten auch die
Buchstaben von A bis F. Binärzahlen werden von einem % angeführt, z.B.
%10010110, sie bestehen nur aus 0 und 1 (Strom oder nicht im Draht!). Die
Konvertierung untereinander der drei Zahlensysteme bereitet keine
Probleme, da es unter dem ASMONE den "?"-Befehl gibt, gefolgt von der zu
konvertierenden Zahl. Als Resultat erhält man das Äquivalente in Dezimal,
Hexadezimal und ASCII, also CHARAKTERN: denn auch die Buchstaben wie
"ABCDabcde..." sind nur durch ein Byte dargestellt. So ist z.B. das "Z"
$5a (Probiert ?"z"). Um Charakter anzugeben setzt man sie unter
Gänsefüßchen ("" oder ''), und man kann sie mit den Befehlen kombinieren
(z.B. MOVE.B #"a",Label1) oder mit dem DC.B direkt in den Speicher geben
(DC.B "Ein Text im Speicher").

---

* In Assembler wird die Multiplikation durch * dargestellt, die Division
durch /, und man kann runde Klammern verwenden, wieviel man will, z.B:
move.l #(100/2*(12+$41-32)+%01101010),RESULTAT

* 1 byte = 8 bit ($00 = %00000000; $FF = %11111111)
  1 word = 16 bit ($0000 = %0000000000000000; $FFFF = %1111111111111111)
  1 long = 32 bit, ossia 2 words ($00000000 = %00000000000000000000000000000000)

---

Bei Bits zählt man folgens: von 0 rechts nach links: z.B. ein Byte, das
Bit 5 auf 1 hat (oder High): $00100000. Bei einem Byte gehen die Bits von
0 (niederwertigsten) zum siebten (höchstwertigsten), ein Word von 0 bis
15, ein Longword von 0 bis 31. Um Bits leicht numerieren zu können, könnt
ihr diesen Trick verwenden:
```

```
; 5432109876543210 - ein word
move.l  #%1000010000110000,d0   ; bit 15,10,5 e 4 High (auf 1)
```

---

*Adressen werden per Konvention durch Hexzahlen dargestellt.

---

* Befehle mit dem "#"-Symbol, wie etwa MOVE.L #123,d0, CMP.L #10,LABEL1 etc. betrachten die Zahl nach dem Lattenzaun (#) wie eine konstante Zahl, also wirklich als "Nummer", nicht als Adresse, zum Unterschied wenn kein # vorhanden ist: move.b $12,$45 kopiert das Byte aus Adresse $12 in Adresse $45, während move.b #$12,$45 die Zahl $12 in Adresse $45 kopiert.

---

* Die DATENREGISTER und die ADRESSREGISTER sind alle 32 Bit lang, also ein Longword. Auf Adressregistern kann man nur mit .W oder .L arbeiten, nicht mit .B.

---

* Auf ungeraden Adressen kann man nicht mit .W oder .L-Instruktionen arbeiten, nur mit .B. Ein move.l #1,$10001 schickt den computer in GURU, während ein move.b #1,$10001 keine Probleme verursacht.

---

* Ein Byte kann eine Zahl zwischen $00 und $FF (255) enthalten, wenn dann noch etwas addiert wird, startet die Zahl wieder bei NULL. Das gleiche gilt für das Word, bei dem $FFFF die größte, darstellbare Zahl ist, und für das Longword. Dies hat max. $FFFFFFFF.

---

* Das LABEL, die KOMMENTARE nach den ";" und die DC.x sind keine 68000erBefehle, aber Assemblerbefehle, die es uns ermöglichen, Punkte im Listing (z.B. Daten oder Routinen) zu markieren, Kommentare einzufügen, um das Listing klarer und verständlicher zu gestalten oder Bytes, Words oder Longwords direkt an einen bestimmten Punkt im Speicher zu geben (DC.x). Das kann verifiziert werden, indem man den Speicher mit dem Befehl "D $xxxx" oder "D LABEL" disassembliert.

** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** **

```
; ADRESSIERUNGEN DES 68000 (Beispiele)

; Adressierungen mit absoluten Adressen, .L (Longword)

 move.l #$123,$50000 ; wir geben $00000123 in $50000. Die Nullen links sind
; Optional, denn move.l #$00000123,xxx unterscheidet
; sich nicht von move.l #$123,xxx, im Speicher werden
; die Nullen immer trotzdem angehängt.
; ZU BEACHTEN ist, daß mit diesem .L-Befehl vier
```

```
; Bytes im Speicher verändert werden, also die Bytes an
; Adresse $50000, $50001, $50002 und $50003, die
; folgende Werte erhalten:
; $50000 = $00
; $50001 = $00
; $50002 = $01
; $50003 = $23

---

; Adressierungen mit absoluten Adressen, .W (Word)

 move.w #$123,$50000 ; Wir geben $0123 in Adresse $50000 - Mit dieser
; .W - Instruktion haben wir zwei Bytes verändert,
; da ein Word 2 Bytes lang ist, und zwar die
; Adressen $50000 und $50001:
; $50000 = $01
; $50001 = $23

---

; Adressierungen mit absoluten Adressen, .B (Byte)

 move.B #$12,$50000 ; Wir geben $12 in Adresse $50000. Mit diesem .b-Befehl
; haben wir 1 Byte modifiziert, und zwar das an
; Adresse $50000 = $12.
; PASST GUT AUF DIE UNTERSCHIEDE AUF, DIE EINTRETEN,
; WENN IHR EINFACH .L, .W UND .B VERTAUSCHT. In der Tat
; liegen oft Fehler der Anfänger darin, diese drei
; Typen zu vertauschen oder ihrer falschen Einschätzung
; Verwendet den Debugger ("AD"), dann die > Taste um
; auch die letztn Zweifel auszuschalten.

 move.l $40000,$50000 ; In diesem Fall geben wir den Inhalt aus Byte
; $40000, $40001, $40002 und $40003 in die vier
; Bytes ab $50000, also in das Byte $50000, $50001,
; $50003 und $50004. Wenn z.B. $40000 = 00102305 war:
; $50000 = $00
; $50001 = $10
; $50002 = $23
; $50003 = $04
; Auf die gleiche Weise kopieren wir mit einem
; .W oder .B von einer Adresse zu anderen jeweils
; zwei Bytes oder eines.

---

BEMERKUNG: Wenn wir LABEL verwenden, um Daten im  Speicher  zu  verändern,
werden  sie  vom Assembler in die EFFEKTIVEN ADRESSEN umgewandelt, die sie
darstellen. Da Label ja Punkte im Speicher markieren, wie  Etiketten  oder
Schildchen,  werden wir uns auf genau diesen Punkt beziehen, wenn wir eine
in irgend einer Art ansprechen oder aufrufen. Befehle  wie  die  folgenden
sind dann auch bei der absoluten Adressierung mit beinhaltet:

MOVE.L LABEL1,$50000
MOVE.W #$123,LABELBLAU
MOVE.B LABELHUND,LABELKATZE
```

Diese werden im Speicher dann immer in ähnlicher Weise dastehen:

```
MOVE.L $64230,$50000 ; angenommen LABEL1 sei auf $64230
MOVE.W #$123,$726e0 ; angenommen LABEL1 sei auf $726e0
MOVE.B $23450,$3a010 ; wie oben...
```

Also, mit Bytes, Words oder Longwords, die mit Labels gekennzeichnet sind, müßt ihr umgehen, als seien es Adressen, den, einmal ASSEMBLIERT, SIND ES ADRESSEN!!!

Deswegen wird bei folgendem Befehl

```
MOVE.L #LABEL1,$dff080 ; Verwendet, um unsere Copperlist
; "anzupeilen"
```

in $dff080 die Adresse von LABEL1 gegeben, und nicht die vier Bytes, die ab LABEL1 stehen: weil LABEL1 in ihre äquivalente Adresse konvertiert wird, und da es nach einem # steht, wird diese Adresse als ("konstante") Zahl betrachtet, und somit wird diese Zahl in $dff080 kopiert. Machen wir ein Beispiel:

```
MOVE.L #LABEL1,LABEL2
MOVE.L LABEL1,LABEL2
```

Werden so assembliert: (Für das Label werden hypotetische Adressen angenommen)

```
MOVE.L #$42300,$53120 ; In $53120 kommt die Zahl $42300,
; also die Adresse des Label
MOVE.L $42300,$53120 ; In $53120 wird das Longword kopiert,
; das sich ab Adresse $42300 befindet
```

Es ist möglich, sich auf elegantere Weise auf absolute Adressen zu beziehen, wenn sie kleiner als das Word sind, also $7FFF, indem man ein .W nach der Adresse anhängt: das ist z.B. der Fall bei Move.L 4.w,A6, das die ExecBase in A6 ladet, aber jede Instruktion, die mit Adressen mit der Länge des Word operieren, können so abgekürzt werden. Die Ersparnis der linken vier Nullen wirkt sich Geschwindigkeitssteigernd aus. Schauen wir uns den Unterschied an:

```
(assembliert)
MOVE.B #10,$123 -> MOVE.B #10,$00000123
MOVE.B #10,$123.w -> MOVE.B #10,$0123 -OHNE ÜBERFLÜSSIGEN
 NULLEN
```

Der Effekt des Befehles ÄNDERT SICH NICHT! Es ändert sich nur die "Form", die schlanker und schneller erscheint. Wenn man vergißt, das .w bei den "kurzen" Befehlen anzuhängen, dann produziert man nur Code, der einige Word länger ist, nicht mehr.

```
**    **    **    **    **    **    **    **    **    **    **

; Datenregister, .L (Longword)
```

```
 move.l #$123,d0 ; Datenregister direkt (wir geben $123 in D0)

 move.l d1,d0 ; Datenregister direkt, wir geben den Wert, der in
; d1 enthalten ist, in d0)

; Datenregister, .W (Word) (Bemerkung: Man nennt die rechte Hälfte des
Long das "niederwertige Word", die linke
das "höherwertige Word": $HOCH+NIEDER,
.L = 4 Byte = 2 Word)

 move.w #$123,d0 ; In diesem Fall haben wir nur das niederwertige
; Word von d0 verändert: wenn d0 $0012fe3c war, und
; wir nur auf dem niederwertigen Word agieren, also
; $fe3c, dann wird d0 danach so aussehen: $00120123

 move.w d1,d0 ; Das Gleiche, wir kopieren das niederwertige Word
; von d1 ins niederwertige Word von d0. Wenn d1
; $12345678 enthält, und d0 $9abcdef0, dann wird nach
; diesem Befehl d0 folgendes enthalten: $9abc5678
     ^^^^ WORD!

; Datenregister, .B (Byte)

 move.b #$12,d0 ; In diesem Fall ändern wir nur das Byte ganz rechts,
; wenn d0 z.B. $0012fe3c war, nur auf dem ersten
; Byte zugreifend, wird es so verändert: d0=$0012fe12

 move.b d1,d0 ; Das Gleiche, wir kopieren das erste Byte von d1
; in das erste Byte von d0. Wenn d1 $12345678 enthält,
; während d0 $9abcdef0, dann wird nach dieser Instr.
; d0 so aussehen: $9abcde78
 ^^ Byte!
```

Die Adressregister a0,a1,a2,a3,a4,a5 und a6 (VERWENDET NICHT A7, auch SP genannt - Stack Pointer) verhalten sich wie die Datenregister, nur kann man auf ihnen NICHT mit .B zugreifen. Man kann darin auch Daten geben, auch wenn sie für Adressen vorgesehen sind.

```
**    **    **    **    **    **    **    **    **    **    **

; INDIREKTE ADRESSIERUNGEN MITTELS ADRESSREGISTERN

 move.w #123,(a0) ; Bei diesem Move wird die Zahl 123 in das Word
; kopiert, das sich ab der Adresse befindet, die
; in a0 steht. Man sagt indirekt dazu, weil die
; Zieladresse nicht direkt angegeben ist, sondern
; Mittels Register, das die Adresse enthält. Das
; geschieht nur, wenn das Adressregister in Klammern
; geschrieben steht, ansonsten würde man 123 in das
; Register selbst schreiben. Ein DATENREGISTER kann
; NICHT dazu verwendet werden, eine indirekte
; Adressierung zu verrichten.
; Man kann sagen, daß das Ragister a0 als ZEIGER
; auf eine Speicherzelle verwendet wurde, es ZEIGT
; also wie der Mauspointer oder ein Spürhund in
```

```
; Richtung der Beute: man nennt eine Adresse oder
; ein Register "ZEIGER", wenn dessen Inhalt eine Adres.
; von irgend etwas enthält,auf das man zugreift,
; indem man den Zeiger fragt, wo sich dieses befindet.
; Zeiger werden meist auch als "POINTER" bezeichnet.
; Z.B. die Copperlist hat ein Pointerregister, das
; $dff080, in das die Adresse der Copperlist gegeben
; wird. Der Copper schaut bei jedem Fotogramm in
; $dff080 nach, wo sich die Copperlist befindet.

 move.l (a0),(a1) ; In diesem Fall wird das Long, das sich ab Adresse a0
; befindet, in Adresse a1 kopiert. Wenn vor der
; Ausführung dieses Befehles in a0 die Adresse $100
; gestanden hätte, und in a1 $200, dann wäre dieser
; gleichwertig mit einem
; MOVE.L $100,$200, oder, noch raffinierter,
; MOVE.L $100.w,$200.w...

---

; INDIREKTE ADRESSIERUNG MIT POST-INKREMENTIERUNG (Erhöhung der Adresse NACH
    Ausführung)

 move.w #123,(a0)+ ; Auf diese Art wird die Zahl 123 in das Word kopiert,
; das sich ab Adresse a0 befindet, und DANACH wird
; a0 um ein WORD INKREMENTIERT. Wenn die Anweisung ein
; .B gewesen wäre, dann würde nach dem Move a0 um nur
; ein Byte erhöht, bei einem .L um 4 Bytes, also ein
; Long.

 move.l (a0)+,(a1)+ ; Mit dieser Anordnung kopieren wir das Long, das sich
; ab Adresse a0 befindet, in Adresse a1 und folgende,
; und danach werden beide Register, a0 und a1, um
; jeweils vier Byte erhöht (Long).
; Praktisch bewegen wir uns auf die darauffolgenden
; Longword im Speicher. Mit einer Serie solcher
; Instruktionen könnte man ein Stück Speicher kopieren:

lea $50000,a0 ; Quell-Adressee
lea $60000,a1 ; Ziel-Adresse
move.l  (a0)+,(a1)+
move.l  (a0)+,(a1)+
move.l  (a0)+,(a1)+
move.l  (a0)+,(a1)+
move.l  (a0)+,(a1)+
; Jetzt haben wir 5 Longwords von $50000 nach
; $60000 kopiert.

---

; INDIREKTE ADRESSIERUNG MIT PRE-DEKREMENT (Adresse wird VOR Ausführung
     erniedrigt)

 move.w #123,-(a0) ; ALS ERSTES WIRD A0 UM 2 BYTES DEKREMENTIERT
; (verringert, abgezogen), ALSO UM EIN WORD,
; und DANACH wird 123 in das Word kopiert, das
; sich ab der Adresse, die in a0 steht, befindet.
```

```
; Wenn die Anweisung eine .B gewesen wäre, dann
; würde dem a0 nur 1 Byte abgezogen, bei einem
; .L hingegen 4 Byte (Long).

 move.l -(a0),-(a1) ; a0 und a1 werden beide um jeweils 4 Bytes
; dekrementiert (Long), und dann wird der Inhalt,
; der sich ab der nun resultierenden Adresse a0
; befindet, in die jetzt resultierende Adresse a1
; kopiert.

; Mit einer Reihe solcher Befehle könnte man, wie im vorigen
; Fall, ein Stück Speicher kopieren, aber mit dem
; Unterschied, daß man rückwärts vorgehen würde, so
; wie Krebse. Wir müßten bei der Adresse starten, die
; das Ende der Kopie ist, und nach hinten gehen, bis
; wir den Anfang erreicht haben, im Beispiel $50000 und
; $60000. Setzten wir also $50014 und $60014 als Startwert,
; und dann kopieren wir solange ein Long "nach hinten", bis
; wir bei $50000 bzw. $60000 angekommen sind: um die Adresse
; zu berechnen, bei der wir starten müssen, habe ich zum
; Anfangswert (5*4) dazugezählt, also = $14, praktisch
; 5 Longwords * 4 Bytes pro Long. Zu Beachten, daß im Speicher
; $50000+(5*4) als $50014 assembliert wird, denn während der
; Assemblierfase werden auch eventuelle mathematische
; Operationen durchgeführt.

lea $50000+(5*4),a0 ; Quelladresse am ENDE
lea $60000+(5*4),a1 ; Zieladresse am ENDE
move.l  -(a0),-(a1)
move.l  -(a0),-(a1)
move.l  -(a0),-(a1)
move.l  -(a0),-(a1)
move.l  -(a0),-(a1)

; In diesem Fall haben wir 5 Longwords von $50000
; nach $60000 kopiert, aber sind bei $50014 gestartet
; und bis $50000 nach "hinten" gegangen. Der Unter-
; schied zum vorigen Beispiel ist wie der Unterschied,
; der darin besteht, den Flur von Links oder von
; Rechts her zu putzen: in beiden Fällen "kopieren" wir
; den Schmutz in den Eimer, aber in zwei verschiedenen
; Richtungen.

---

; INDIREKTE ADRESSIERUNG MIT ADRESSIERUNGSDISTANZ (OFFSET) UND INDEX

 move.w #12,5(a0,d0.w)  ; Bei dieser Anweisung wird 12 in das Word kopiert,
; das sich ab der Adresse befindet, die sich aus
; der Summe von 5 + a0 + Word in d0 bildet. Wenn z.B.
; in a0 $50000 stehen würde, und in d0 $1000, dann
; würde 12 auf die Adresse $51005 kopiert.
; Das Offset kann hier aber nur zwischen -128 und +127
; variieren.
; Praktisch wird zur Summe, die die Adresse ergibt,
; auch noch ein Register hinzugezogen,das sowohl
; DATEN- wie auch ADRESSREGISTER sein kann, bei dem
```

```
; der ganze Inhalt verwendet werden kann (.L) wie auch
; nur ein Word (.W).
; Byteweise ist es nicht verwendbar. Man nennt dieses
; zusätzliche Register INDEX.

  EINIGE BEISPIELE:

lea $50000,a3
move.w  #$6000,d2
move.l  #123,$30(a3,d2.w) ; kopiert 123 in $56030
*
lea $33000,a1
move.w  #$2000,a2
move.l  #123,$10(a1,a2.w) ; kopiert 123 in $35010
*
lea $33000,a1
lea $20010,a2
move.l  #123,-$10(a1,a2.l) ; kopiert 123 in $53000

**    **    **    **    **    **    **    **    **    **    **

; ADRESSIERUNGEN RELATIV ZUM PC (mit automatischem Offset)
```

Diese Art der Adressierungen werden vom ASMONE automatisch in Ordnung gebracht, sie werden unbemerkt übergangen: z.B. schaut euch den Unterschied zwischen diesen beiden Anweisungen an:

```
MOVE.L  LABEL1,d0 ; ABSOLUTE ADRESSE
MOVE.L  LABEL1(PC),d0 ; ADRESSE RELATIV ZUM PC
```

Diese beiden Anweisungen tun das Geliche, aber die mit dem (PC) ist kürzer und schneller als die Erste. Sie ist relativ zum PC, denn die basiert auf einer Adressierungs-Distanz (Offset) in Bezug auf das PC-Register, dem PROGRAM COUNTER, das ist das Register, in dem der 68000 Buch führt, wo er gerade mit der Ausführung ist. Das Offset wird automatisch von Assembler errechnet, und im Speicher landet dann gleich schon das richtige Offset, um sich zwischen den Label und anderen Anweisungen richtig zu orientieren. Die Instruktion enthält nun nicht mehr die Adresse der Label, sondern die Anzahl der Bytes, die die Entfernung davon nach vorne/hinten angeben. Der Unterschied ist klar: wenn wir den ganzen Code in eine andere Speicherregion verlegen, dann verschieben sich die absoluten Adressen, die Distanzen zwischen den Labels und den (PC)-Befehlen aber bleibt gleich groß, deswegen "funktioniert" diese Methode immer, während ein nicht zum PC-relatives Programm, wenn an einen anderen Ort im Speicher versetzt, alles in Chaos stürzt. Denn ein move.l LABEL1,d0 wird als (angenommen) MOVE.L $23000,d0 übersetzt, also befindet sich das Label auf Adresse $23000. Wenn wir nun das ganze Programm, das z.B. bei $20000 startete und bei $25000 endete, um $10000 nach vorne verschieben, dann werden bei der Ausführung nicht indifferente Fehler auftreten, da sich ein MOVE.L $23000,d0 nicht mehr auf LABEL1 bezieht, das liegt jetzt ja auf $33000! Aber wenn der Code vollständig relativ zum PC erstellt wurde, dann hätte sich das MOVE immer auf das Label bezogen, also auf $33000, da es die - immer gleichbelibende -Distanz zum Label berechnet hätte. Auch Befehle wie BRA, BSR, BNE, BEQ sind relativ zum PC, ein BSR.W ROUTINE1 wird im Speicher z.B. als BSR (50 Bytes weiter vorne) assembliert, und nicht BSR $30000. Adressen werden von Befehlen assembliert, die äquivalent zum BSR sind, wie etwa JSR: ein JSR LABEL1 wird mit der Adresse von Label1

assembliert, genauso ein wird JMP (SPRINGE-Äquivalent zum BRA) die REALE ADRESSE von LABLE1 bekommen. Aber wieso wird nicht immer einfach eine Adressierung relativ zum PC vorgezogen, also ein BSR einem JSR? Weil die Adressierungen mit PC das Limit haben, sich nur auf Adressen beziehen zu können, die maximal 32767 Bytes nach vorne oder -32768 Bytes nach hinten liegen. Für weiter entfernte Label müssen Move mit absoluter Adresse oder JSR/JMP eingesetzt werden. Aber, wie schon gesagt, werden all diese Rechenaufgaben vom Assembler übernommen, deswegen interessieren sie uns nicht, wir müssen uns nur erinnern, DAß WENN MÖGLICH, IMMER ein (PC) gesetzt werden sollte, und BSR und BRA an Stelle von JSR und JMP verwendet werden sollten. Sollte die Distanz zu groß sein, dann meldet der Assembler einen Fehler, und wir müssen das (PC) entfernen oder schlimmstenfalls das BRA/BSR durch JMP/JSR ersetzen, das die größten Entfernungen erreichen kann. Man könnte auch nur mit JMP/JSR und ohne (PC) programmieren, aber der Code würde länger und um einen Augenblick langsamer erscheinen, deswegen ist immer besser, alles so gut als möglich zu machen!!! Das Problem der RELOCATION, also des Verschiebens im Speicher, wird vom Betriebssystem übernommen: wenn wir unser Programm mit WO als Ausführbares abspeichern, dann speichern wir ein File ab, das von der Shell aus aufgerufen werden kann, indem man seinen Namen eingibt. Das Betriebssystem kümmert sich dann darum, es an einen freien Platz im Speicher zu geben, der irgendwo sein kann, und reallociert (A.d.Ü: tut mir leid, mir fällt für ALLOCATE kein deutsches Wort ein. Es bedeutet soviel wie "ansiedeln", "zuteilen", "anweisen". Wer einen Deutsche Ausdruck kennt, bitte "Readme.Deutsch" lesen!) das Programm, es passt also die Adressen der JSR und der nicht zum PC relativen Move an, um den neuen Gegebenheiten (andere Adressen) zu entprechen. Deswegen kann man auch programmieren, ohne sich den Kopf darüber zu zerbrechen, überall die (PC) für Labels zu setzen, die sich in anderen SECTIONS befinden: z.B. die COPPERLIST befindet sich in einer anderen SECTION, und sie kann nur geändert werden, wenn man ohne (PC) arbeitet, weil das Betriebssystem die Sektionen auf unvorhersehbare Distanzen setzt (allocate...), die vielleicht sogar größer sind als 32768, also dem LIMIT der RC-Realtiven Adressierung.

BEISPIELE FÜR PC-RELATIVE ADRESSIERUNGEN:

```
MOVE.L  LABEL1(PC),LABEL2 ; Bemerkung: man kann das (PC) nicht
; für Labels verwenden, die als Ziel
; stehen!
; move.l a0,LABEL(PC) ist ein Fehler!
ADD.L LABELBAU(PC),d0 ; Geht weil das Label die QUELLE ist
SUB.L #500,LABEL ; KEIN PC, WEIL HIER DAS LABEL
; EIN ZIEL IST
CLR.L LABEL ; hier kann kein PC gesetzt werden.
; Praktisch kann man ein (PC) nur
; einsetzen, wenn es vor einem
; Beistrich steht!

; ADRESSIERUNGEN RELATIV ZUM PC MIT OFFSET UND INDEX
```

Diese Adressierung ist das Gleiche wie vorhin, nur mit INDEX, also einem Register, das zum (PC) und zum Offset summiert wird, genauso wie es beim Offset+Index mit Adressregistern geschieht:

```
MOVE.L LABEL1(PC,d0.w),LABEL2  ; Wie die PC-Adressierung, nur muß
; noch das Word in d0 dazugezählt
```

```
; werden, wir beziehen uns also nicht
; auf LABEL1, sondern auf ein Label,
; das d0 von LABEL1 entfernt ist.
ADD.L LABELWAU(PC,a0.l),d0 ; Wie vorher, a0.l wird als Index
; verwendet.
```

Das ist alles, was die Adressierungen angeht.

```
**    **    **    **    **    **    **    **    **    **    **
```

GEBRÄUCHLICHSTEN BEFEHLE:

```
MOVE.x QUELLE,ZIEL ; Kopiert ein Byte, ein Word oder
; ein Longword

LEA Adresse,Ax ; Ladet eine Adresse: Diese Anweisung
; kann nur mit Adressregistern verwendet
; werden. Sie dient dazu, die entsprechende
; Adresse (sei sie nun in Form eines Label
; oder einer Zahl gegeben, z.B. $50000)
; ins Register zu geben.
; Das gleiche wie : MOVE.L #Adresse,a0
; aber schneller!

CLR.x Ziel ; Dieser Befehl löscht das Ziel
; (setzt es auf 0) CLR = CLEAR = "REINIGE"
```

BEDINGTE SPRÜNGE MIT EINEM TST, BTST, CMP

```
CMP.x Quelle, Ziel ; Vergleicht zwei Operanden, die ein
; Label oder ein Register sein können, oder
; sonst eine absolute Zahl (#)  mit einem
; Register uvm. POITIVES Ergebnis, wenn die
; zwei Operanden GLEICH sind (für folgende
; BEQ/BNE)

TST.x Register.Label/Adresse  ; Kontrolliert, ob der fragliche
; Operand gleich NULL ist, wenn ja,
; Positives Ergebnis

BTST #x,Adresse/Dx ; Kontrolliert, ob Bit x der Adresse
; auf NULL steht; wenn ja POSITIVES
; Ergebnis. Man kann ein BTST auch auf
; ein Datenregister ausführen, in diesem
; Fall ist ein Test auf einem der 32 möglichen
; Bits (0-31) erlaubt. Wird das BTST auf eine
; Speicherzelle angewandt, so muß man sich mit
; einem Byte (0-7) begnügen.
```

Sofort nach dem CMP, TST oder BTST steht immer ein BNE, ein BEQ oder ein anderer, ähnlicher Befehl. Im Falle des BNE oder des BEQ kann man Verzweigungen oder konditionierte Sprünge vom TST/CMP aus machen. DIE BEW/BNE/BSR/BRA können sowohl .w wie auch .b sein, jenachdem, wie weit die Routine vom Aufrufenden Befehl entfernt ist. Wenn sie sehr nahe liegen, kann auch ein .b verwendet werden, das oft als .s geschrieben wird. Es ist das Gleiche (s= SHORT, -> KURZ).

```
BSR.x label ; Führe die Routine LABEL aus, und wenn
; du auf ein RTS am Ende der Routine stößt,
; dann kehre zurück.

BEQ.x label ; Wenn das Resultat der vorherigen Abfrage
; POSITIV war, dann springe zum Label
; (DANACH ABER KEHRE NICHT ZURÜCK, WIE IM
; FALLE DES BSR, HIER WIRD GEWÄHLT ZWISCHEN
; SPRINGEN ODER NICHT

BNE.x label ; Wenn das Resultat NICHT positiv war,
; dann springe zum Label
; (DANACH ABER KEHRE NICHT ZURÜCK, WIE IM
; FALLE DES BSR, HIER WIRD GEWÄHLT ZWISCHEN
; SPRINGEN ODER NICHT

BRA.x label ; Springe IMMER zum Label (wie JMP)

ADD.x Operand1,Ziel ; Mit diesem Befehl wird ein Wert zum
; Ziel addiert

SUB.x Operand1,Ziel ; Mit diesem Befehl wird ein Wert
; von Ziel subtrahiert

SWAP Dx ; Vertauscht die 2 Words des Longwords
; in einem DATENREGISTER, braucht kein
; .B, .W oder .L
```

SWAP kommt aus dem Englichen (Wunder, Wunder), und bedeutet "Vertausche", in der Tat vertauscht es die zwei Words, aus dem ein Longword besteht:

```
MOVE.L #HUNDMAUS,d0 ; in d0 kommt das Longword HUNDMAUS

SWAP d0 ; Wir vertauschen die Words: in d0
; steht jetzt MAUSHUND !!!!
```

*

Bemerkung: Es existieren Anweisungen, die den Adressregistern gewidmet sind: z.B. müssen wir CMPA.W d0,a0 schreiben und nicht CMP.W d0,a0,auf die gleiche Art und Weise ADDA.W a2,a0 und nicht ADD.W a2,a0. Für Konstanten hingegen (#xxxx) müssen wir CMPI.x #10,d0 verwenden, und nicht CMP.x #10,d0. Genauso SUBI.x #123,d2 und nicht SUB.x #123,d2, aber der ASMONE assembliert immer AUTOMATISCH die richtige Anweisung, auch wenn wir nur cmp/add/sub schreiben. Also, keine Sorgen, wenn in einem Listing einmal CMPI auftaucht, weiter unten dann CMP alleine, oder adda und add, weil der ASMONE immer alles zum Rechten biegt. Zum Testen assembliert die folgenden Zeilen und disassembliert mit "D PROBE", der ASMONE wird nach den Regeln assemblieren.

```
PROBE:
CMP.W d0,a0
ADD.W a1,a2
SUB.L #123,$10000
CMP.b #20,d4
```

Wird so assembliert werden:

```
CMPA.W D0,A0
ADDA.W A1,A2
SUBI.L #$0000007B,$00010000
CMPI.B #$14,D4
```

Bemerkung2: Gewisse Instruktionen, die das gleiche tun, können auf verschiedene Weise geschrieben werden: z.B. hat der 68000er Befehle, die bestimmten Situationen angepasst sind, und dort schneller sind:

```
1) ADDQ.x #Zahl,Ziel ; Der Befehl ADDQ.x kann für Additionen
; verwendet werden, dessen Zahlen kleiner
; sind als 8 (Q = Quick, "Schnell")

2) SUBQ.x #Zahl,Ziel ; Der Befehl SUBQ.x kann für Subtraktionen
; mit Zahlen von 1 bis 8 verwendet werde,
; (wie oben...)

3) MOVEQ #Zahl,dx ; Das MOVEQ kann dazu verwendet werden, um
; das MOVE.L #Zahl,d0 zu ersetzen, wobei
; Zahl zwischen -128 und +127 liegen muß.
; Das MoveQ ist immer .L, deswegen braucht
; es kein .B, .W oder .L.
```