Albert-Ludwigs-Universität Freiburg
Institut für Informatik

Lehrstuhl für Rechnerarchitektur
Professor Dr. Bernd Becker

Konfiguration eines Multiprozessorsystems und Integration von genetischen Algorithmen

von Tobias Schubert

Diplomarbeit

10. Dezember 1999

Hiermit versichere ich, Tobias Schubert, daß ich die vorliegende Diplomarbeit eigenständig erstellt habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Freiburg, den 10. Dezember 1999

Vorwort

Im alltäglichen Leben, sei es im Beruf, in der Schule oder im Studium, nimmt *Team-Arbeit* einen hohen Stellenwert ein. Nur durch eine geeignete Arbeitsaufteilung und kontinuierlichen Austausch von Erfahrungswerten der beteiligten Personen ist es möglich, eine Gesamtleistung zu erreichen, die das Leistungsvermögen des Einzelnen übertrifft.

Bei dem am Lehrstuhl für Rechnerarchitektur in Freiburg entstandenen Projekt eines flexiblen Multiprozessorsystems wird von obigem Grundgedanken in zweierlei Hinsicht Gebrauch gemacht: Zum einen gliedert sich diese Diplomarbeit in eine ganze Reihe von Studien- und Diplomarbeiten ein, deren Ziel jeweils die Realisierung eines Teilaspektes ist. Zum anderen beschäftigt sich dieses Projekt mit der *Team-Arbeit* auf Hardware-Ebene, d.h. das Multiprozessorsystem soll durch massive Parallelität vieler Prozessoren ein Höchstmaß an Leistung bei den gestellten Anwendungen erbringen.

An dieser Stelle gilt mein Dank all Denen, die mich während meines Studiums begleitet und unterstützt haben.

Freiburg, den 10. Dezember 1999

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Betrachtet man die enorme Leistungssteigerung moderner Computer in den letzten zehn Jahren, so könnte man der irrtümlichen Hoffnung erliegen, alle gestellten Aufgaben mittels purer Rechenkraft exakt lösen zu können. Dennoch existieren viele Problemstellungen, die sich nicht in *akzeptabler* Zeit optimal lösen lassen. Derartige Aufgaben sind in der Realität an vielfältigen Stellen vorzufinden, man denke nur an möglichst sparsame Lagerhaltung, maximale Auslastung von Produktionseinheiten, kostengünstige Anordnung von Schaltkreisen auf Chips oder auch logistische Lösungen im Transportwesen.

Viele der genannten Aufgaben fallen in das Gebiet der sogenannten *NP-vollständigen* Probleme, was bedeutet, daß derzeit kein deterministischer Algorithmus bekannt ist, der in polynomieller Zeit die beste Lösung berechnen kann (unter der Voraussetzung, daß $NP \neq P$ ist). Klassische Beispiele dieser Gruppe sind etwa das in Kapitel 9 implementierte *Travelling Salesman Problem* (kurz: *TSP*-Problem; gegeben ist eine Anzahl an Städten und gesucht ist die kürzeste Rundreise, die jede Stadt genau einmal besucht und wieder am Startort endet) oder auch das am Lehrstuhl für Rechnerarchitektur intensiv betrachtete *Channel Routing Problem*, bei dem zu einer gegebenen Menge von Anschluß-Pins eines Schaltkreises die kürzeste, kreuzungsfreie Verdrahtung der Anschlüsse gesucht wird.
Hervorgerufen wird der gewaltige Rechenaufwand unter anderem durch den zu betrachtenden Suchraum aller möglichen Lösungen, der häufig enorm groß ist. Beispielsweise gibt es beim symmetrischen *TSP*-Problem bei einer Rundreise durch 20 Städte bereits $\frac{1}{2} \cdot 19! \; = \; 6.08 \cdot 10^{16}$ viele unterschiedliche Routen, die bewertet werden müssen.

Gesucht sind also alternative Methoden zum 'schnellen' Lösen derartiger Probleme (mit polynomieller Laufzeit). Prinzipiell sind solche Verfahren dann realisierbar, wenn auf die Forderung nach der bestmöglichen Lösung verzichtet wird und der Algorithmus endet, sobald eine 'sehr gute' Nährung erreicht ist.
Genetische Algorithmen stellen hier eine Alternative dar und folgen in ihrem Grundkonzept einem Vorbild der Natur: der Evolution. Abstrahiert läßt sich sagen, daß von den Individuen einer Gattung, die an der Fortpflanzung beteiligt sind, nur diejenigen den Fortbestand der Art gewährleisten, die besonders 'gut' an die Umgebung angepaßt sind. Die bezüglich einer Güte- oder Fitneßfunktion 'schlechten' Individuen sind demnach zum Aussterben verurteilt (*Darwin'sche Selektionstheorie*, 1859).
Die algorithmische Nachbildung generiert zufällig oder durch problemspezifisches Wissen eine Anfangsmenge von möglichen Lösungen (die Start-*Population*), und unterwirft die Individuen (auch Chromosome genannt) mit Hilfe verschiedener Operatoren der Fortpflan-

zung, um danach nur die besten der neu erzeugten 'Kinder' in die bestehende Population aufzunehmen (*Steady-State-Repräsentation*, vgl. Kapitel 9).
Ein wichtiges Merkmal genetischer Algorithmen ist die Tatsache, daß gleichzeitig immer ein ganzer *Pool* von Lösungsmöglichkeiten betrachtet wird, was sich für den Einsatz von Multiprozessorsystemen eignet. Hier kann die Menge möglicher Lösungen auf die verschiedenen zur Verfügung stehenden Prozessoren aufgeteilt werden, die autonom neue Chromosome kreieren und in bestimmten Intervallen austauschen. Durch diese parallel ausgeführte Arbeit wird eine deutliche Performance-Steigerung erhofft (vgl. Kapitel 10). Abbildung 1.1 zeigt dies schematisch für vier Prozessoren.

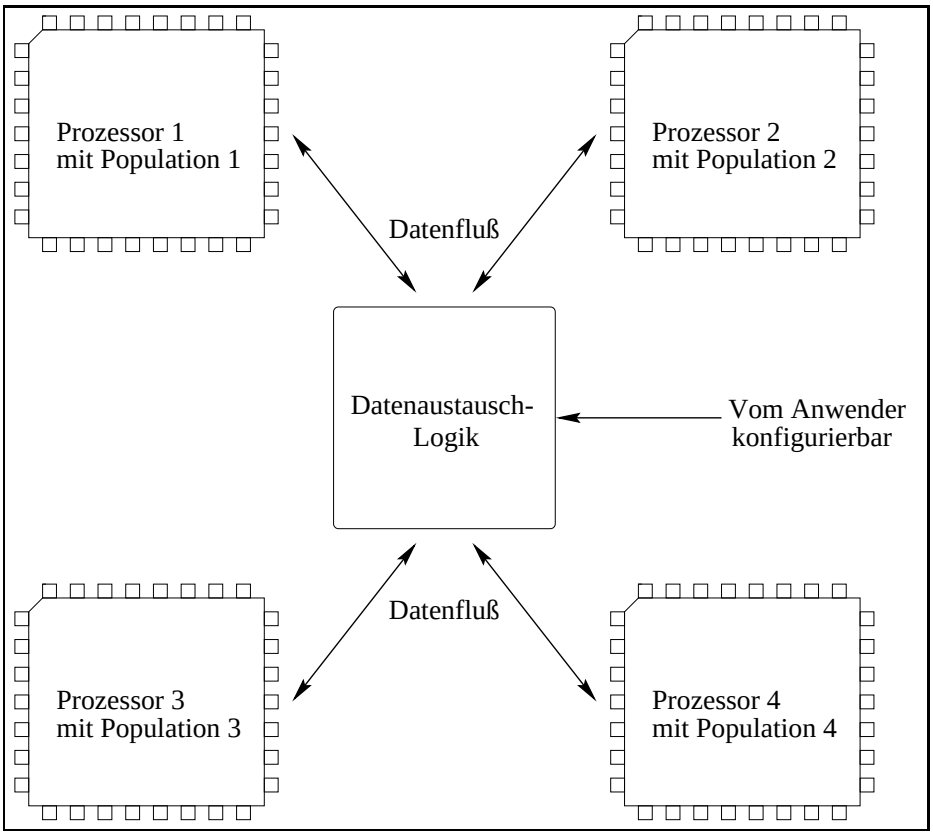


Abbildung 1.1: Blockdiagramm eines Parallelprozessorsystems mit vier Prozessoren

Letztgenannte Idee wird in dieser Diplomarbeit aufgegriffen und ein skalierbares System mit bis zur vier parallel arbeitenden Prozessoren in Betrieb genommen. Diese Prozessoren sollen mit einer Art 'Betriebssytem' programmiert werden, und nach einem Systemstart oder Reset-Befehl vom Anwender mit den eigentlichen Programmen zur Problembearbeitung versorgt werden.

Um die übermittelten Aufgabenstellungen effizient lösen zu können, wird die Datenaustauschlogik der obigen Abbildung so realisiert, daß sie vom Anwender jederzeit geändert werden kann. Abbildung 1.2 zeigt zwei mögliche Strategien des Datenaustausches (Topologie) bei vier parallel arbeitenden Prozessoren.

Die Leistungsfähigkeit wird abschließend am Beispiel des *TSP*-Problemes demonstriert.

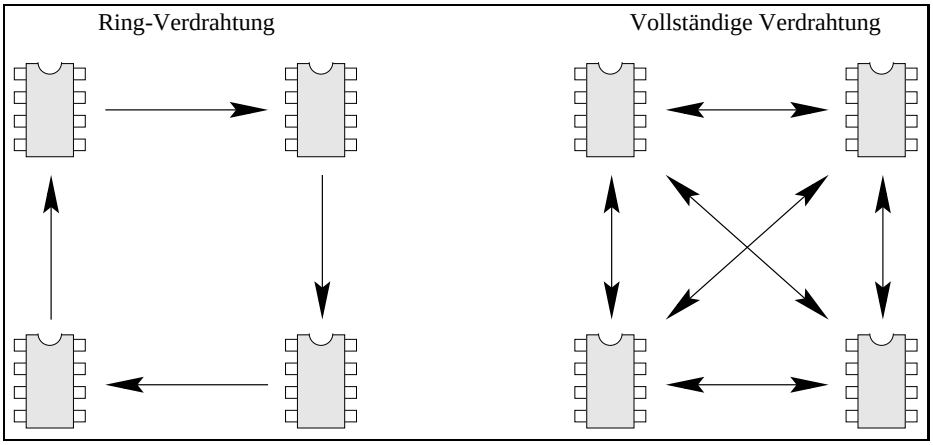


Abbildung 1.2: Zwei mögliche Datenaustausch-Strategien bei vier Prozessoren

Die weiteren Abschnitte dieser Arbeit sind wie folgt gegliedert:

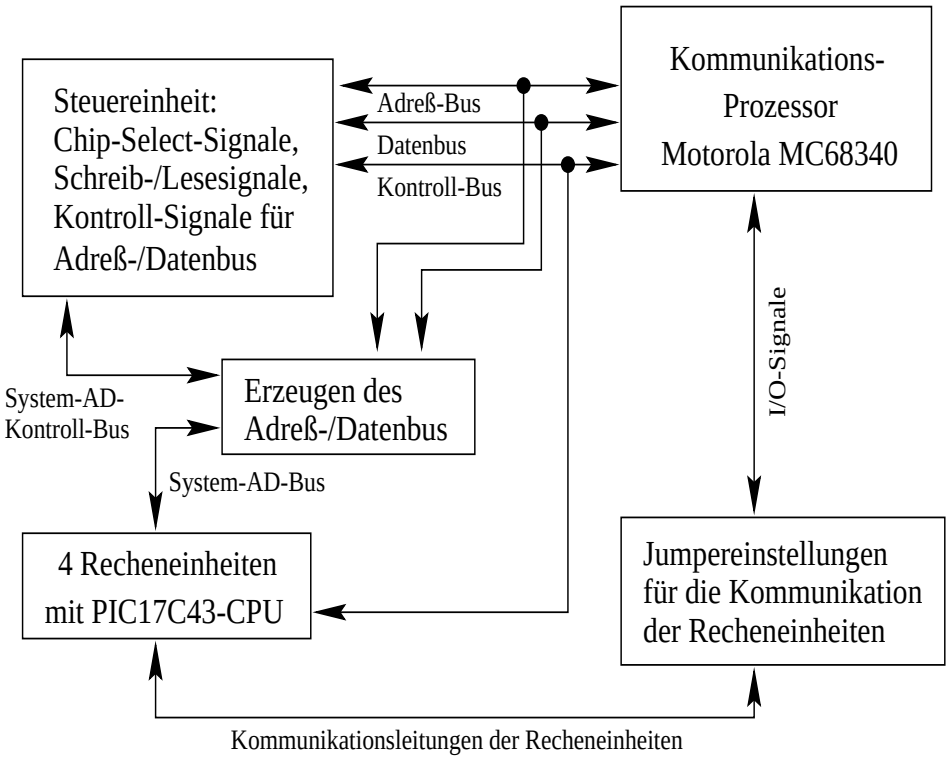
- Kapitel 2 gibt eine Einführung in das zur Verfügung stehende Multiprozessorsystem.
- Die Zielsetzungen der Diplomarbeit sind in Kapitel 3 dargestellt.
- Das entworfene Ablaufschema, nach dem die Prozessortypen programmiert wurden, zeigt Kapitel 4.
- In Kapitel 5 werden die notwendigen Gleichungen für die Steuereinheiten der Prozessorstreifen entwickelt.
- Die Anbindung des Multiprozessorsystemes an einen Computer (Kontakt zum Anwender) wird in Kapitel 6 behandelt.
- Die Programmierung der beiden Prozessoren *Motorola MC68340* und *Microchip PIC17C43* wird in den Abschnitten 7 und 8 erläutert.
- Kapitel 9 und 10 demonstrieren die Leistungsfähigkeit des Systems anhand des *Travelling Salesman Problems*.
- Abschließend werden mögliche Verbesserungen und zukünftige Arbeiten in Kapitel 11 diskutiert.
- Im Anhang ist die Implementation aller Programme im jeweiligen Assembler-Dialekt dargestellt. Zusätzlich ist eine CD-ROM beigefügt, die alle Programmdateien sowie die verwendeten Software-Tools enthält.
  Die Schaltpläne liegen dem Lehrstuhl für Rechnerarchitektur momentan nur in Papierform vor. Aufgrund der schlechten Abbildungsqualität wurde deshalb auf die Darstellung im Anhang verzichtet.

Kapitel 2

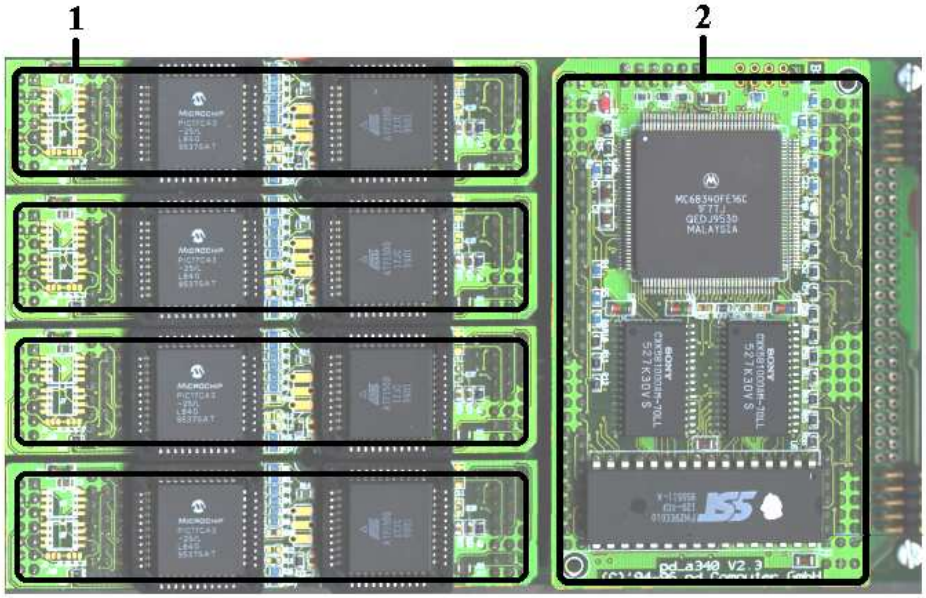
Hardware

Am Lehrstuhl für Rechnerarchitektur werden intensiv genetische Algorithmen und ihre Effizienz bei vielfältigen Aufgabenstellungen untersucht. Besonders sticht hierbei das Eingangs bereits genannte *Channel Routing Problem* hervor. Die bei diesen Arbeiten gewonnenen Erfahrungen mündeten in die Entwicklung eigener Hardware, die speziell derartige Algorithmen-Konzepte unterstützt und zeigen soll, wie die zugrundeliegende Topologie (vgl. Abbildung 1.2) Einfluß auf die Laufzeit und Güte der Lösung nimmt. In Zusammenarbeit mit der Firma *PD Computer - Gesellschaft für Prozeß- und Datentechnik mbH* mit Sitz in Karben bei Frankfurt am Main entstanden dabei die in diesem Kapitel vorgestellten Komponenten.
Abbildung 2.1 zeigt das *kleine* Multiprozessorsystem, das die Plattform für diese Arbeit bildet und mit dem sich maximal vier Recheneinheiten (Punkt **(1)** der Abbildung 2.1) parallel betreiben lassen. Zur Steuerung der verschiedenen Module dient der sogenannte Kommunikationsprozessor, hier mit **(2)** beschriftet. Es ist deutlich ersichtlich, daß die Recheneinheiten und auch der Kommunikationsprozessor in sich eigenständige Funktionsblöcke sind, die auf die Gesamtplatine nur aufgesteckt werden. Die Vorteile dieser modularen, flexiblen Vorgehensweise liegen darin, daß gegebenenfalls die einzelnen Komponenten durch jeweils Leistungsstärkere ausgetauscht werden können, ohne das komplette Parallelprozessorsystem neu entwickeln zu müssen. Dabei muß allerdings gewährleistet sein, daß die Schnittstellen, d.h. die verschiedenen Bus-Architekturen, kompatibel zueinander sind. Für die Programmierung bedeutet dies, daß alle Leistungs-Anforderungen an das System mittels Software realisiert werden müssen, was einen Mehraufwand bei der Integration der Algorithmen und einen Geschwindigkeitsverlust beim Lösen der Problemstellungen bedeuten kann. Dieser Nachteil wird hier aber in Kauf genommen, um in Forschung und Lehre ein breitgefächertes Anwendungsspektrum abdecken zu können.

Die beiden nächsten Abschnitte widmen sich dem Kommunikationsprozessor und den Recheneinheiten, abschließend wird dann das am Lehrstuhl für Rechnerarchitektur entwickelte System vorgestellt, mit dem sich bis zu 81 Prozessoren parallel betreiben lassen.



(a) Blockschaltbild



(b) Foto

Abbildung 2.1: Das kleine Multiprozessorsystem

2.1 Der Kommunikationsprozessor

Die Funktionseinheit *pd_a340, Microcontroller-Modul im Scheckkartenformat mit MC68340* (im folgenden pauschal nur noch als Kommunikationsprozessor bezeichnet), stellt **das** zentrale Modul des Gesamtsystemes dar. Wie aus dem Blockschaltbild in Abbildung 2.2 ersichtlich basiert es auf einem Motorola MC68340-Prozessor und übernimmt vielfältige Aufgaben: Beispiele sind die Generierung der systemweiten Taktfrequenz von 16.78 MHz und der Kontakt zum Anwender, um Steuersignale und zu lösende Aufgabenstellungen entgegenzunehmen und an die Recheneinheiten weiterzugeben. Zusätzlich obliegt dem Baustein die komplette Kontrolle des Datenaustausches zwischen den einzelnen Prozessoren, d.h. es werden deren Kontaktanfragen ausgewertet und entsprechend Daten verschoben.

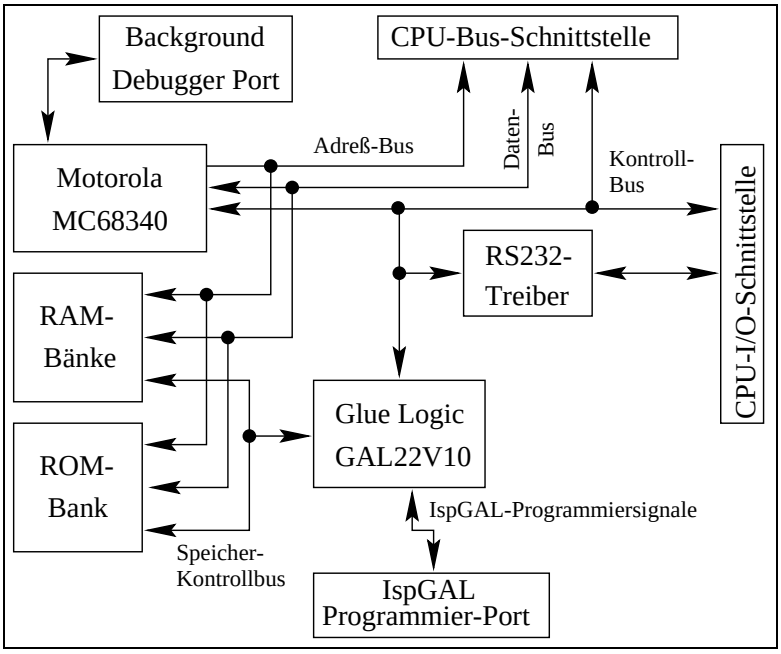
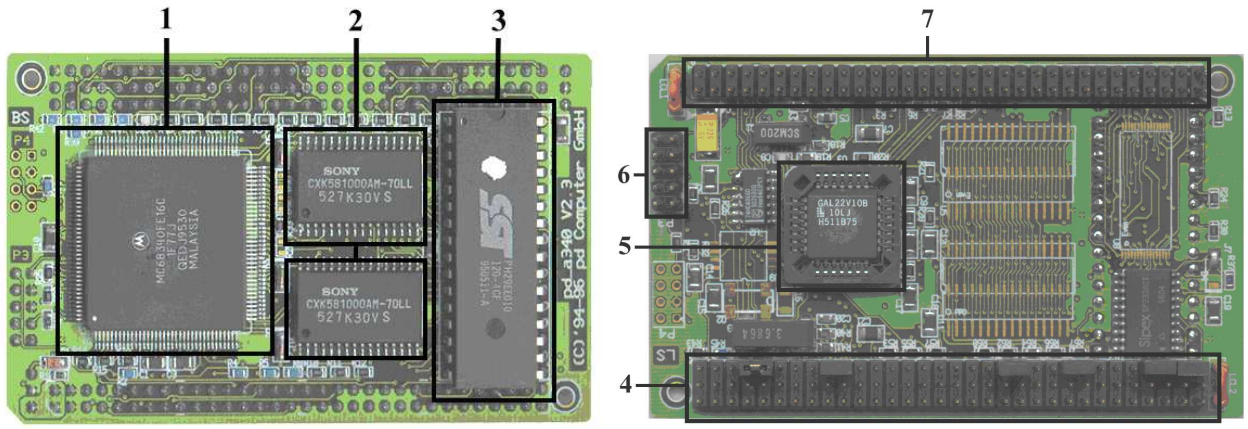


Abbildung 2.2: Blockschaltbild des Kommunikationsprozessors

Die Karte setzt sich aus den nachfolgend aufgeführten Bausteinen zusammen, die kurz erläutert werden. Die Nummerierungen und Bezeichnungen der einzelnen Komponenten sind dabei in den beiden Aufnahmen des Kommunikationsprozessors in Abbildung 2.3 zu finden.

1. **Motorola MC68340.** Hierbei handelt es sich um einen 32 Bit CISC-Prozessor mit zwei DMA-Controllern, zwei seriellen Schnittstellen (von denen nur Kanal B zum Kontakt mit dem angeschlossenen Computer genutzt wird), 4 GByte Adreßraum und insgesamt 256 verschiedenen Interruptvektoren. Ein Blockschaltbild mit den unterschiedlichen Signalgruppen ist in Kapitel 7 enthalten.

2. **CMOS-Speicher SONY CXK581000AM.** Auf dem Kommunikationsprozessor sind zwei RAM-Bänke vorgesehen: die mit Punkt **(2)** markierte, 'obere' Speicherbank (256 kByte Speicherkapazität) und die sogenannte 'untere' Speicherbank, die sich auf der Rückseite der Platine befindet und momentan, wie Abbildung 2.3.b zeigt, nicht bestückt ist.



(a) Ansicht von oben (b) Ansicht von unten

Abbildung 2.3: Ein komplett bestückter Kommunikationsprozessor

3. **EEPROM SST29EE010.** In diesem EEPROM ist ein Mini-Betriebssystem (*Flash Bios*) für den Motorola MC68340 gespeichert. Es stellt einige wichtige Grundfunktionen bereit, wie etwa den Aufbau einer initialen Vektortabelle mit allen Sprungadressen, um bei einem *Power-On-Reset* einen definierten Programmstart zu gewährleisten. Zusätzlich werden den Peripherie-Bausteinen logische Adreßbereiche (*Chip-Select*-Signale) zugeordnet, unter denen sie vom MC68340 aus anzusprechen sind. Im Einzelnen sind dies momentan:

   - CSO: 128 kByte EEPROM im Bereich $\$00FE0000 - \$00FFFFFF$ mit drei *Wait States* bei 8 Bit Datenbreite.
   - $CS1$: 16 kByte I/0-Bereich von $\$FFFF8000$ bis $\$FFFFBFFF$ mit drei *Wait States* bei 8 Bit Datenbreite.
   - $CS2$: Entweder nutzbar für die 'untere' (derzeit nicht bestückte) Speicherbank oder zum Ansprechen des *I-Cube* der großen Multiprozessorkarte ([3]).
   - $CS3$: 256 kByte Speicher im Bereich $\$00000000 - \$0003FFFF$ mit zwei *Wait States* bei 16 Bit Datenbreite.

   Das EEPROM besitzt 128 kByte Speicherkapazität, kann also zusätzlich noch um weitere Funktionen wie das Grundprogramm aus Kapitel 7 erweitert werden.

4. **Jumper-Leiste.** Hier können grundlegende Parameter des Kommunikationsprozessors eingestellt werden. So ist die Größe der genutzten Speicher- und EEPROM-Bausteine zu definieren oder auch wichtige Signalverknüpfungen wie zum Beispiel eine systemweite Motorola-Bestätigung bei eingehenden Interrupts ($\overline{iackin}$) können gesetzt werden. Dies ist für die Verknüpfung mit den Prozessorstreifen von Bedeutung, denn erst zum Zeitpunkt der Bestätigung darf von diesen das Interruptsignal zurückgenommen werden.
   Weil die vorhandenen Jumpereinstellungen momentan ohne Änderung übernommen werden können, sei für deren genaue Bedeutung auf [16] verwiesen.

5. **Glue Logic GAL22V10.** Dieser Baustein generiert aus den vier *Chip-Select*-Leitungen des Motorola MC68340 die entsprechenden Schreib- und Lesesignale für die externen Peripherie-Module. Besonders im Hinblick auf den Kontakt zu den Prozessorstreifen ist die Steuerung von Bedeutung, damit jeder der vier PIC-Prozessoren eindeutig referenziert werden kann.

6. **Background Debugger Port.** In Zusammenarbeit mit der ICD32-Software, im Verzeichnis `Diplomarbeit\Software\Motorola-Debugger`, kann der Benutzer den Prozessor zur Laufzeit von einem Computer aus überwachen. Hierzu wird der Steckverbinder mit einem speziellen Kabel an einen der parallelen Anschluß-Ports des Rechners angeschlossen. Es ist möglich, eigene Programme in den Speicher zu laden, schrittweise abzuarbeiten und sich dabei die veränderten Register und Speicherzellen anzeigen zu lassen. Derzeit wird dieser Zugang genutzt, um das Multiprozessorsystem mit dem Motorola-Betriebssystem aus Kapitel 7 zu starten. Abbildung 2.4 zeigt eine Momentaufnahme: im oberen Teil sind die Spezialregister mit ihren Werten und daneben der gerade aktuelle Programm-Code zu verfolgen.

7. **Systemweite Daten-, Adreß- und Steuerleitungen.** Eine Zuordnung der Pins zu den Signalnamen und ihrer Funktion ist [16] zu entnehmen.

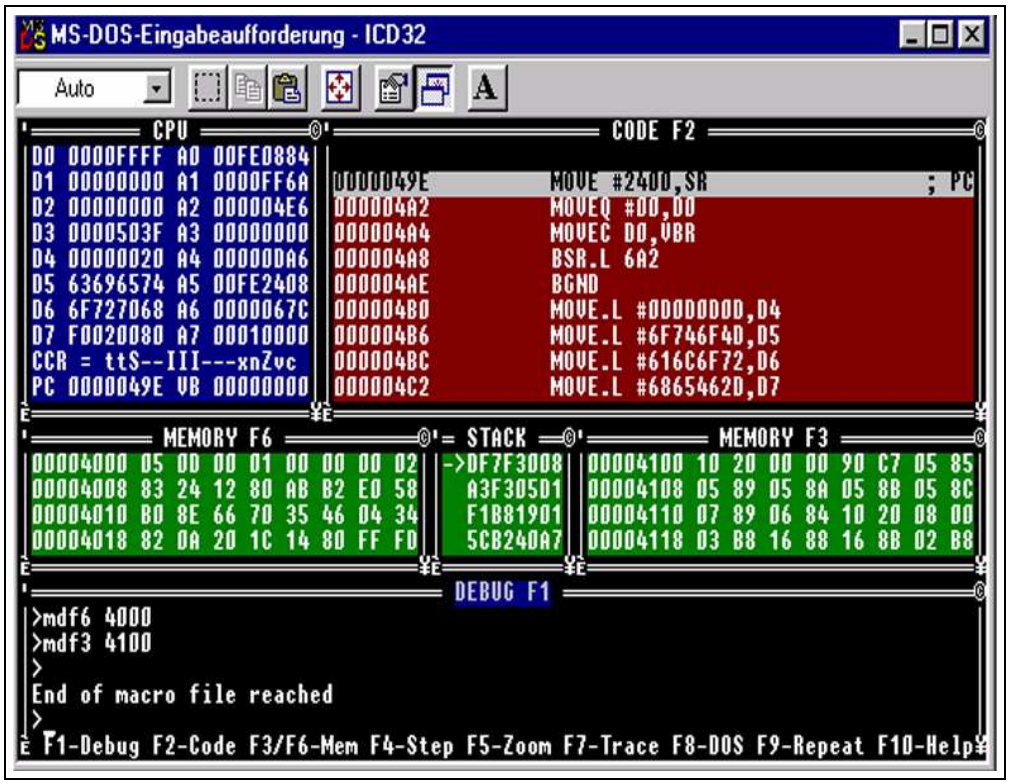
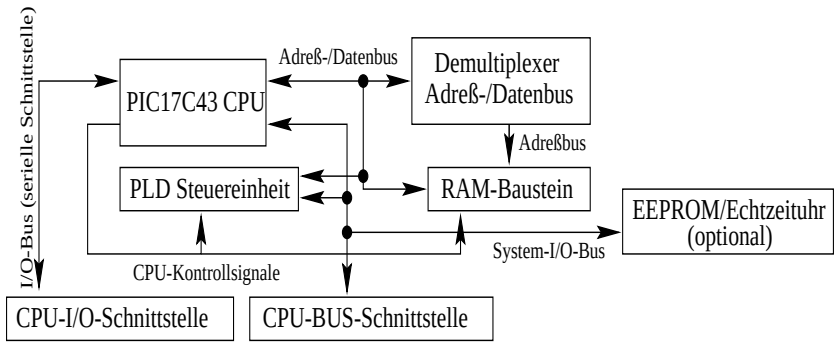


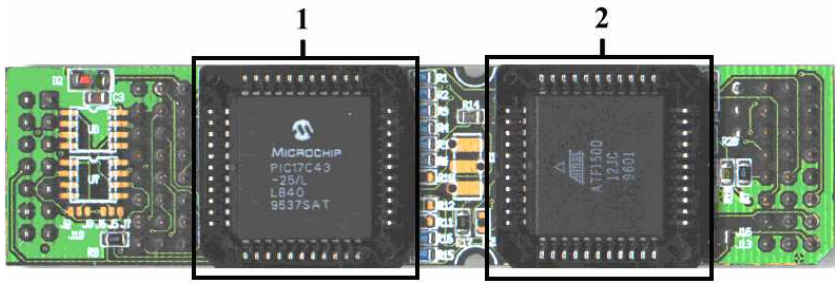
Abbildung 2.4: Momentaufnahme der ICD32-Software

2.2 Die Recheneinheiten

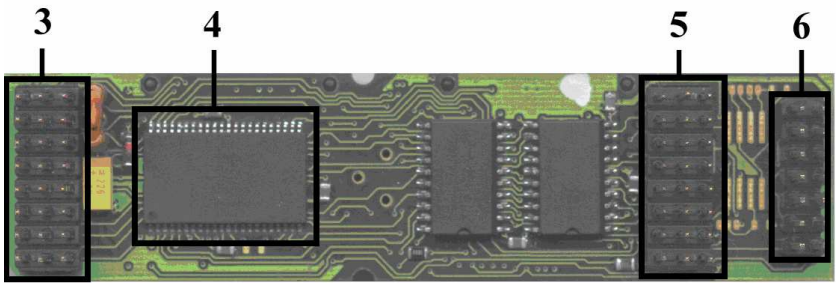
Neben dem Blockschaltbild mit den wichtigsten Daten- und Signalpfaden sind in Abbildung 2.5 die Grundrecheneinheiten noch einmal aus zwei verschiedenen Blickwinkeln dargestellt. Aufgrund ihres Aussehens wird im weiteren Verlauf nur noch der Begriff *Prozessorstreifen* verwendet, in den Datenblättern [15]-[18] sind sie als *pd_cmp43* bezeichnet. Sie dienen dazu, die vom Anwender gestellten Aufgaben zu lösen, und stellen in sich geschlossene kleine Computer dar. Auch hier sind die grundlegensten Bausteine durch die Umrandung mit entsprechender Nummer gekennzeichnet und werden wie folgt vorgestellt:



(a) Blockschaltbild



(b) Ansicht von oben



(c) Ansicht von unten

Abbildung 2.5: Ein komplett bestückter Prozessor-Streifen

1. **Microchip PIC17C43-Prozessor.** Hierbei handelt es sich um eine RISC-CPU, die über folgende Eigenschaften verfügt: 8 Bit Datenbreite, 16.78 MHz Taktfrequenz, 4 KByte EPROM für das zu implementierende 'Betriebssystem' aus Kapitel 8, 11 Interrupt-Quellen, serielle Schnittstelle und fünf frei konfigurierbare I/O-Ports. Ohne Speicherpartitionierung ist ein maximal 128 kByte großer Adreßraum adressierbar.

2. **Steuereinheit Atmel CPLD ATF1500.** Um den Kontakt zwischen den Prozessorstreifen und dem Kommunikationsprozessor aufzubauen und das Zusammenspiel zwischen PIC17C43 und externem Speicher **(4)** zu steuern, wird ein eigener Logikbaustein auf jeder Recheneinheit benötigt. Die notwendigen Verknüpfungen zeigt Kapitel 5.

3. **HSB-CB-Steckerleiste.** Die Signale dieser Pfostenleiste stellen die Verbindung zum Kommunikationsprozessor dar. Gemäß der Ansicht auf Abbildung 2.5 sind die Leitungen wie folgt definiert (die funktionelle Bedeutung ergibt sich aus Kapitel 5):

Reihe A	Reihe B	Reihe C	
/breq	ad7	vcc	Pin 8
/bgnt	ad6	gnd	
/cs	ad5	clk	
/iackin	ad4	/int	
/dtack	ad3	/reset	
/trig	ad2	ale	
sda	ad1	/rd	
scl	ad0	/wr	Pin 1

4. **Statischer Speicher CXK516100** mit 128 kByte Kapazität. Genutzt wird dieses Modul zum Abspeichern temporärer Daten (wie etwa die Chromosome einer Population bei genetischen Algorithmen), sowie zum Ablegen der Anwenderprogramme, die auch von dort ausgeführt werden müssen. Dadurch wird sich herausstellen, daß nur noch 112 kByte des externen RAM effektiv zu nutzen sind (vgl. Kapitel 5.2).

5. **HSB-CI-Steckerleiste.** Diese ist besonders für das *große* Multiprozessorsystem wichtig (vgl. Abschnitt 2.3). Auf diesem Stecker liegen mit den Pins RA4 und RA5 die seriellen Schnittstellen der PIC-Prozessoren an, die zum Datenaustausch über die sogenannte *Switch-Matrix* genutzt werden. Erste Ansätze sind in [5] zu finden. Beim eingesetzten kleinen Parallelprozessorsystem kommt dem HSB-CI-Bus allerdings keine tragende Rolle zu.

6. **I/O-Leiste.** Diese leitet die frei konfigurierbaren I/O-Ports des PIC17C43 nach 'aussen', um eventuell weitere Peripherie-Geräte ansprechen und nutzen zu können. Besondere Bedeutung erhält die Leiste bei Anwendungen der Regelungs- und Steuertechnik. PD-Computer setzt beispielsweise ein ähnlich geartetes System mit entsprechender Hardware zum Auswerten von Stimmzetteln bei Aktionärsveranstaltungen ein. In dieser Arbeit wird die Steckerleiste nicht genutzt.

Für die Umsetzung der PIC17C43-Programme wurde der *PICMASTER In-Circuit-Emulator* der Firma Microchip verwendet, der auch zum 'Brennen' des in Kapitel 8 implementierten Betriebssystemes für die vier Prozessorstreifen genutzt wurde. Eine Kopie der eingesetzten Software ist auf der CD-ROM im Verzeichnis `\Diplomarbeit\Software\Mplab` enthalten, für weitere Informationen zu diesem Prozessortyp sei neben den Datenblättern im Verzeichnis `\Diplomarbeit\Datenblaetter\Microchip PIC17C43` auf die Literaturstellen [9]-[12] verwiesen.

2.3 Das große Multiprozessorsystem

Bei dem in dieser Diplomarbeit vorgestellten Parallelprozessorsystem handelt es sich um eine erste Grundversion, mit der maximal vier Prozessoren parallel betrieben werden können. Das Multiprozessorsystem, das letztendlich am Lehrstuhl für Rechnerarchitektur zum Einsatz kommen soll, ist ein System, bei dem maximal 81 Recheneinheiten parallel arbeiten können. Im folgenden soll dies kurz erläutert werden.

Es handelt sich dabei um eine ISA-Steckkarte, wie sie in Abbildung 2.6 dargestellt ist. Diese läßt sich wie jede Netzwerk- oder sonstige Erweiterungskarte in einen der üblicherweise drei freien ISA-Steckplätze eines Rechners einbauen (Punkt **(4)** der Abbildung 2.6). Punkt **(1)** zeigt die Vorrichtungen für die Recheneinheiten, von denen sich mit einer Karte maximal neun parallel betreiben lassen. In der Mitte ist der schon beschriebene Kommunikationsprozessor zu erkennen, der alle Komponenten des großen Multiprozessorsystemes steuert. Der Datenaustausch zwischen den Recheneinheiten wird bei dieser Modell-Variante über den sogenannten *I-Cube* abgewickelt, der in Abbildung 2.6 leider vom Motorola MC68340 verdeckt wird. Dieser stellt eine Leitungsmatrix (*Switch Matrix*) dar, an der die seriellen Schnittstellen der Recheneinheiten angeschlossen sind, und die sich durch SETZEN oder LÖSCHEN von Kreuzungspunkten zur Laufzeit frei konfigurieren läßt. Hiermit können die PIC-Prozessoren ohne den Umweg über den Motorola direkt Daten/Lösungen miteinander austauschen. Bei der Portierung der in den folgenden Kapiteln entworfenen Algorithmen ist dieses stark differierende Hardwarekonzept mit erheblichen Änderungen der Programme verbunden.

War beim kleinen Multiprozessorsystem die Anzahl der zu betreibenden Prozessorstreifen einzig durch die Kapazität der Karte beschränkt, so bietet sich mit den drei in Punkt **(2)** zusammengefaßten Steckverbindungen die Möglichkeit, Prozessoren weiterer Parallelprozessorkarten zu kontaktieren. In der Endausbaustufe sollen dann 81 Prozessoren Problemstellungen lösen. Erreicht wird dies durch den Einsatz sogenannter *Breakout-Boxen*, die jeweils drei ISA-Steckkarten in einem externen Gehäuse zusammenfassen, und die sich mit einer speziellen Treiberkarte vom Rechner aus ansteuern lassen. Hierdurch können anstelle der üblicherweise drei vorhandenen ISA-Steckplätze dann insgesamt neun Karten parallel mit einem PC genutzt werden.

Für Entwicklungszwecke, wie in [3] und [5] beschrieben, sind die wichtigsten Kontrollsignale der unterschiedlichen Bausteine auf den sogenannten Service-Stecker **(3)** geleitet. Zur Generierung grundlegender Signale wie beispielsweise die Übermittlung der jetzt neun PIC-Interrupts an den Kommunikationsprozessor steht mit der *PLD-Kontroll-Einheit* **(6)** das

Gegenstück zum Atmel ATF1500 der Recheneinheiten bereit. Außerdem wird mit diesem Baustein das *Dual Ported RAM* gesteuert, das die Daten von und zum Rechner zwischenspeichert. Ein weiteres Einsatzgebiet ist die Verteilung des Systemtaktes, der nicht für alle Einheiten von derselben Quelle erzeugt werden muß (vgl. Punkt 4 des Kapitels 11, 'Einsetzen schnellerer Recheneinheiten').

Zusammenfassend läßt sich festhalten, daß die Portierung von Algorithmen von der kleinen Multiprozessorkarte auf die Große aufgrund der unterschiedlichen Hardware-Ausstattungen immer mit einem nicht unerheblichen Programmieraufwand verbunden sein wird, insbesondere wegen der gänzlich unterschiedlichen Kommunikation der Recheneinheiten untereinander und wegen des Kontaktes zum Anwender, der jetzt über ein ISA-Protokoll mit entsprechender Treibersoftware zu realisieren ist.

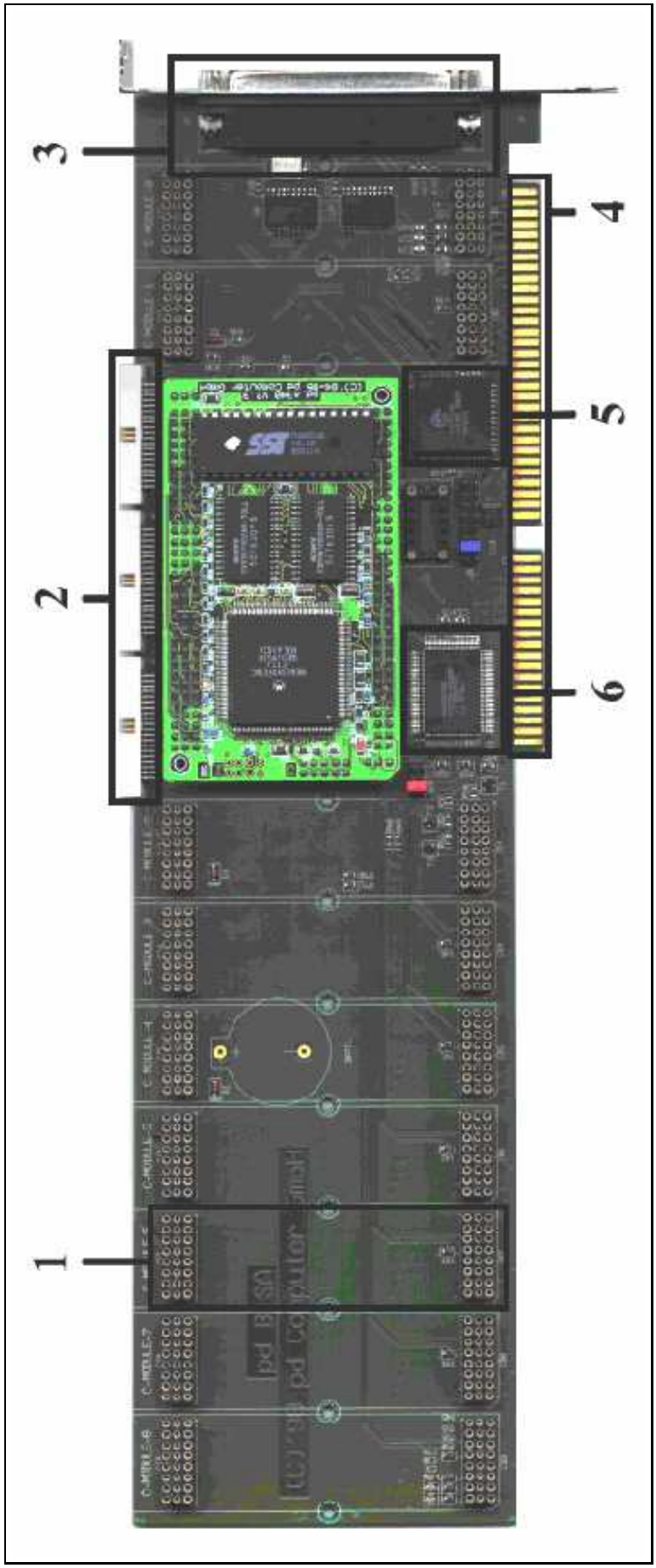


Abbildung 2.6: Das große Multiprozessorsystem

Kapitel 3

Zielsetzungen

Die folgenden Zielsetzungen sollten während der Diplomarbeit realisiert werden:

1. **Ausarbeitung eines Ablaufschemas,** nach dem das Multiprozessorsystem arbeiten soll.

2. **Entwicklung von logischen Gleichungen für den** *ATMEL ATF1500*-**Baustein** (Steuereinheit der Prozessorstreifen), um interruptgestützt zwischen den verschiedenen Recheneinheiten einen Datenfluß zu ermöglichen.

3. **Implementierung der 'Betriebssysteme'** für beide Prozessortypen, wie sie in ihrer Funktionalität im Ablaufschema aus Punkt 1 festgelegt wurden. Bei den Prozessorstreifen soll nach Abschluß dieser Aufgabe das entstandene Programm in den nur einmal beschreibbaren internen PIC17C43-EPROM-Bereich 'gebrannt' werden, um diese Einheiten sowohl auf dem großen wie auch auf dem kleinen Multiprozessorsystem zu nutzen.
   Aus diesem Grund ist eine sehr sorgfältige Analyse der Anforderungen notwendig. Um ein Höchstmaß an Flexibilität bei späteren Einsatzgebieten zu erreichen, wird lediglich eine Endlosschleife bestehend aus Speichertest, Dateiempfang und Programmabarbeitung realisiert (Kapitel 8). Alle aufgabenspezifischen Parameter sind somit in der eigentlichen Anwendung zu kodieren. Dies führt (in Kapitel 8 und 9) zu einer komfortablen Schnittstelle.
   Beim Motorola MC68340 kann das Betriebssystem entweder als *Flash Bios* fest ins EEPROM eingebunden werden, oder es wird bei einem Neustart über den *Background Debugger Port* immer wieder neu ins RAM geladen (für den ersten Weg gibt [5] eine Anleitung).

4. **Erweiterung der in [8] vorgestellten Funktionsbibliothek** mit Grundroutinen für genetische Algorithmen auf Basis des PIC17C43. Die damalige Studienarbeit war auf lediglich einen Prozessor zugeschnitten und muß deshalb insbesondere um Funktionen für den Datenaustausch mit dem Kommunikationsprozessor erweitert werden. Auch sind Anpassungen an das in Punkt 3 spezifizierte PIC-Grundprogramm vorzunehmen.
   Die entsprechenden Passagen sind in Kapitel 9 zusammen mit der Umsetzung der Testanwendung (Punkt 5) erläutert.

5. **Test der Funktionalität aller Komponenten anhand eines Optimierungsproblemes,** das als genetischer Algorithmus implementiert werden soll. Wie schon mehrfach erwähnt, wird das sogenannte *Travelling Salesman Problem* umgesetzt.

6. **Untersuchung der Berechnungsdauer und Lösungsgüte** verschiedener Probleminstanzen bei unterschiedlichen Topologien. Auch soll festgestellt werden, wie sich die Anzahl der verwendeten Prozessoren auf die Gesamtleistung auswirkt. Bei der Dauer der Berechnungen steht weniger der Vergleich mit anderen parallelen Rechnersystemen im Vordergrund als vielmehr 'interne' Messungen: es soll ein Gefühl dafür herausgearbeitet werden, ob die massive Parallelität mit 81 Prozessorstreifen (9 große Multiprozessorkarten) die Nachteile zu beispielsweise 500 Mhz schnellen Pentium-Rechnern ausgleichen kann.

Kapitel 4

Ausarbeitung eines Ablaufschemas

In Gesprächen mit den beteiligten Mitarbeitern des Lehrstuhles hat sich das in Abbildung 4.1 dargelegte Ablaufschema herauskristallisiert, das sich wie folgt beschreiben läßt:
In einem ersten Schritt müssen die zu lösenden Aufgabenstellungen vom Anwender in einem PIC17C43-Programm kodiert und kompiliert werden, damit sie vom Multiprozessorsystem bearbeitet werden können. Nach dem Systemstart wird vom Kommunikationsprozessor nach einer Initialisierungsphase eine Statusmeldung ausgegeben, in der die vom Motorola MC68340 erkannten Prozessorstreifen aufgelistet werden. An dieser Stelle muß der Benutzer entscheiden, ob alle Recheneinheiten dasselbe Problem abarbeiten sollen oder ob jeder Prozessorstreifen eine unterschiedliche Aufgabe zugeordnet bekommt.
Nachdem die verschiedenen Programmdateien über den Kommunikationsprozessor an die Recheneinheiten weitergegeben wurden, beginnt die eigentliche Aufgabenbearbeitung. Hierzu können problemspezifisch Intervalle festgelegt werden, nach deren Erreichen die Recheneinheiten Daten oder auch Ergebnisse austauschen sollen. Die Strategie, welche Prozessoren am Datenaustausch beteiligt sind, kann der Anwender anhand der Zwischenergebnisse jederzeit ändern. Nach Abarbeitung der Aufgaben werden dann die besten Resultate am Bildschirm präsentiert und eventuell ein Neustart des Systemes eingeleitet.
Dem Anwender bietet das festgelegte Vorgehen eine Vielzahl an Möglichkeiten. Insbesondere eignet es sich für den Einsatz von genetischen Algorithmen, weil die Topologie sich zur Laufzeit ändern läßt, die gesamten Problemparameter nur durch die übermittelten Programme bestimmt werden und es zusätzlich möglich ist, jedem Prozessor ein anderes Programm oder andere Daten zuzuordnen. Dies kann dazu genutzt werden, jede Recheneinheit mit unterschiedlichen Parametern (z.B. Populationsgröße und Erzeugung der Startpopulation) auszustatten. Desweiteren ermöglicht dies auch den Einsatz des Systemes im Bereich der Steuerungs- und Regelungstechnik, denn jeder Prozessorstreifen kann folglich die vorhandenen I/O-Ports unterschiedlich nutzen.
Hier könnte eine kleine Einschränkung, besonders bei Echtzeitanwendugen, dadurch entstehen, daß die PIC-Prozessoren zum Beispiel Meßdaten nicht direkt miteinander austauschen können, sondern nur über den Umweg 'Motorola', was natürlich zu Lasten der Geschwindigkeit geht. Allerdings sind zum jetzigen Zeitpunkt des Gesamtprojektes derartige Fragen noch im Zukunftsbereich anzusiedeln.

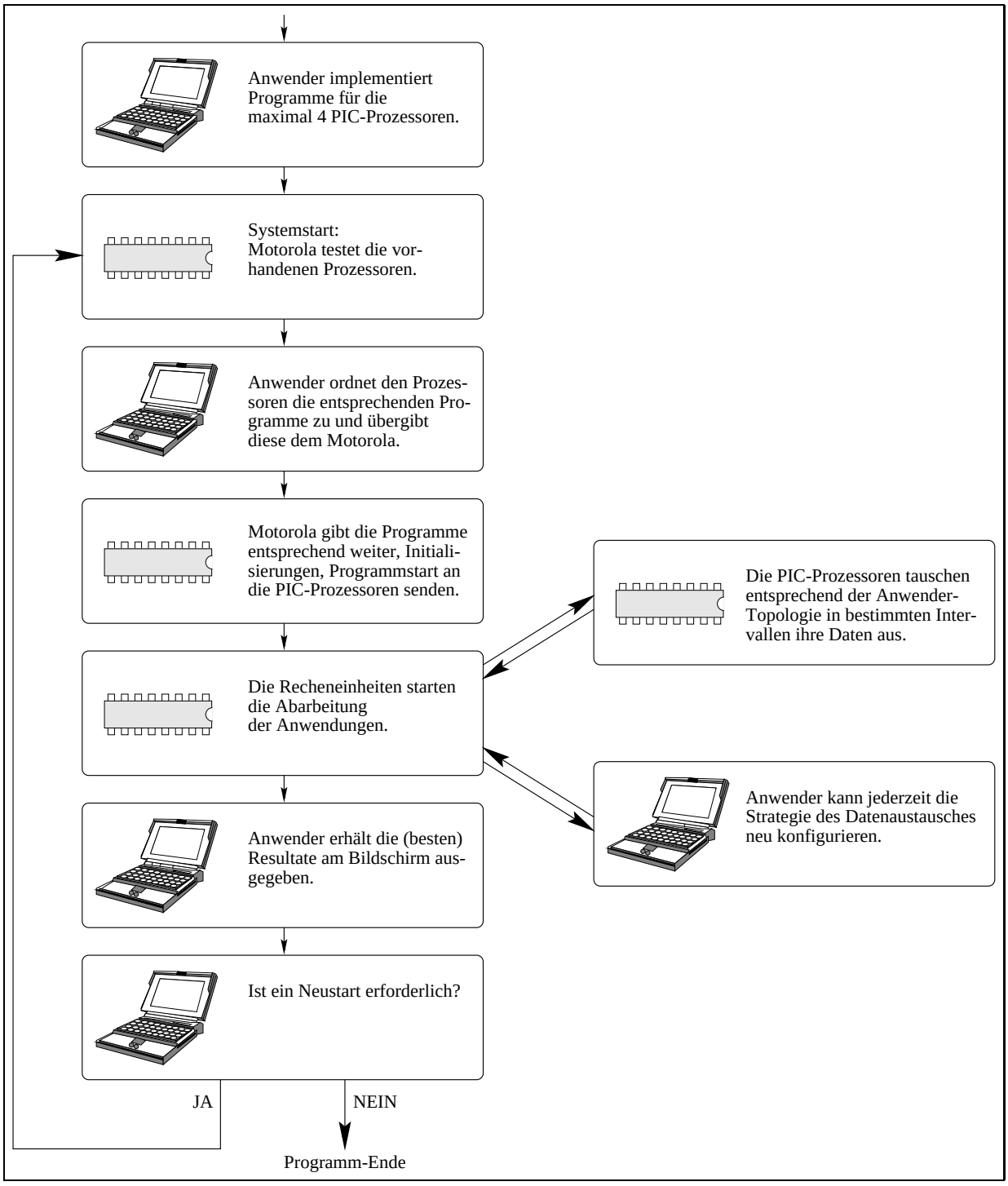


Abbildung 4.1: Ablaufschema des Parallelprozessorsystems

Kapitel 5

Die Steuereinheit der Prozessorstreifen

Mit dem *Atmel CPLD ATF1500* steht auf jedem Prozessorstreifen ein komplexer, programmierbarer Logikbaustein zur Verfügung. Dieser besitzt 32 I/O-Pins mit angeschlossener Makrozelle, die als D-, T- und JK-FlipFlop oder auch als transparentes Latch konfiguriert werden kann. Zusätzlich sind als globale Eingangssignale noch Reset-, Output-Enable- und Clock-Anschlüsse vorhanden. Mittels Rückführungssignalen und Kaskadierungsmöglichkeiten können dadurch komplexe logische Schaltkreise realisiert werden.
Als Entwicklungs- und Programmierwerkzeug steht *WinCupl* zur Verfügung, das als Ausgabe eine *JEDEC*-Datei erzeugt, die von jedem handelsüblichen Universalprogrammiergerät zum (mehrmaligen) 'Brennen' genutzt werden kann. Die im Verzeichnis `\Diplomarbeit\Software\PLD Programmier Software` abgespeicherte Version kann zudem eine Simulation zu einem vorgegebenen Muster an Signalbelegungen vornehmen. Diese Ergebnisse sind allerdings nur für die logische Korrektheit der entworfenen Gleichungen zu nutzen, weil es sich um eine statische Analyse handelt, d.h. die internen Gatterverzögerungszeiten, etc. dabei nicht berücksichtigt werden. Solche Untersuchungen lassen sich aber komfortabel mit dem lehrstuhleigenen *Logic Analyzer* bewerkstelligen. Als Einstieg in die Thematik eignen sich die Literaturverweise [1], [2] und die Datenblätter im Ordner `\Diplomarbeit\Datenblaetter\Atmel PLD ATF1500`.

Der Baustein wird dazu benutzt, um dem PIC17C43 das Ansprechen des externen Speichers zu ermöglichen und den Datenaustausch mit dem Kommunikationsprozessor aufzubauen. Dies erhält eine zusätzliche Bedeutung durch die Tatsache, daß auch die Anwenderaufgaben über den Kommunikationsprozessor an die Recheneinheiten weitergegeben werden müssen. Bevor die Umsetzung der beiden Aufgabenstellungen erläutert wird, sind nachfolgend die Datenpfade zu den Einheiten Microchip PIC17C43 und Motorola MC68340 sowie die Blockdiagramme des ATF1500 (entnommen aus [2]) abgebildet.

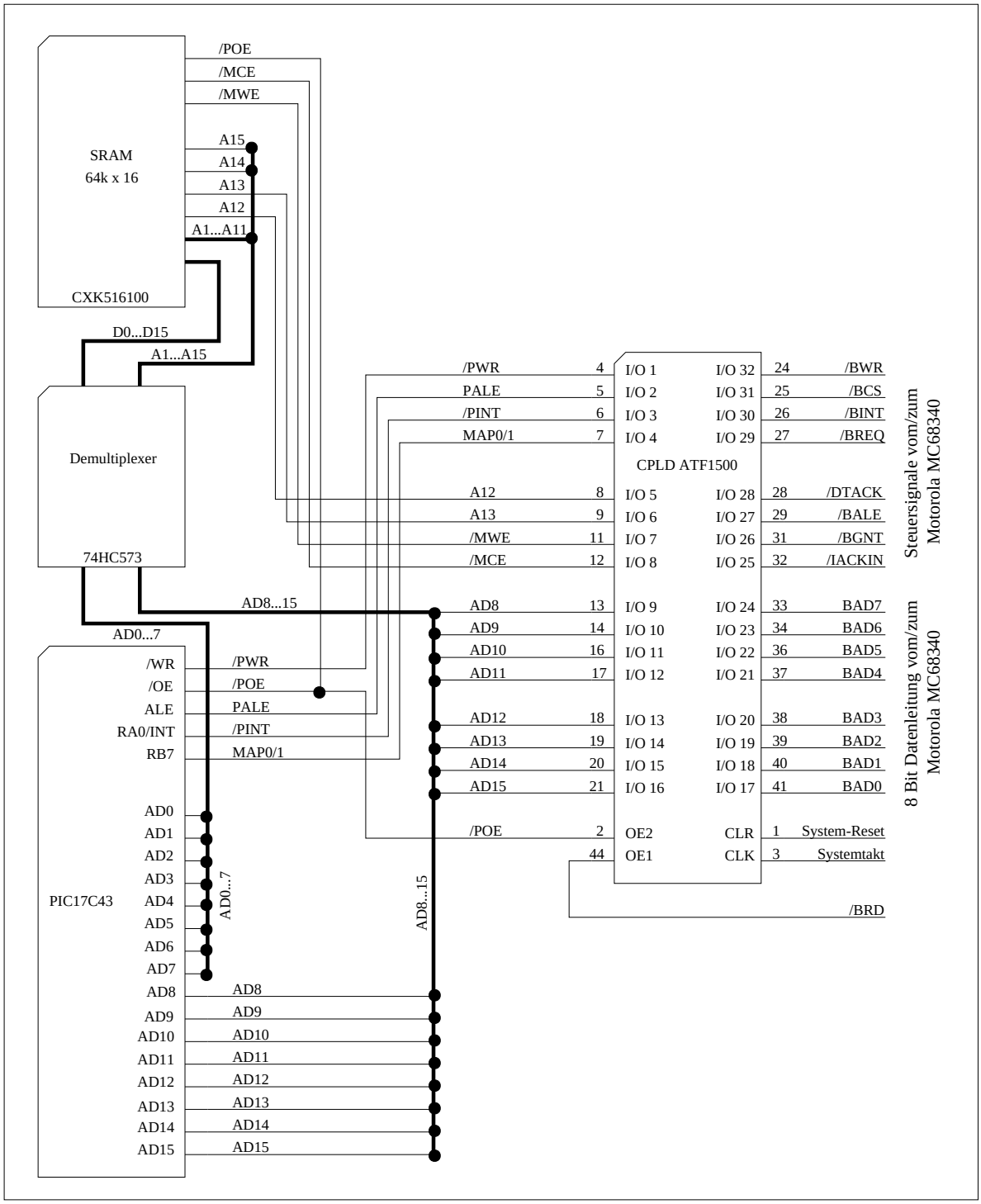


Abbildung 5.1: Anbindung zwischen PIC17C43, Atmel ATF1500 und Motorola MC68340

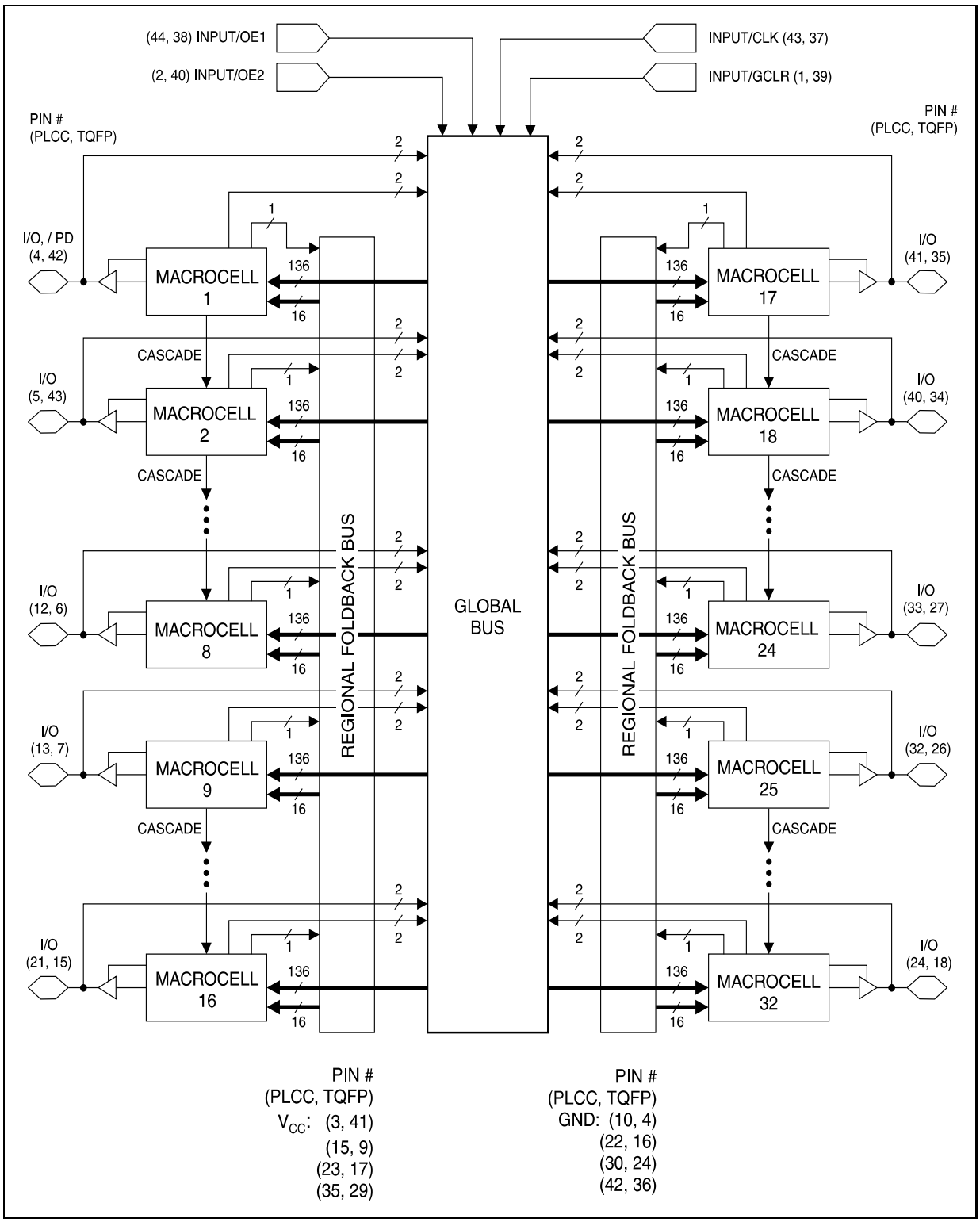


Abbildung 5.2: Blockdiagramm des ATF1500-Bausteines

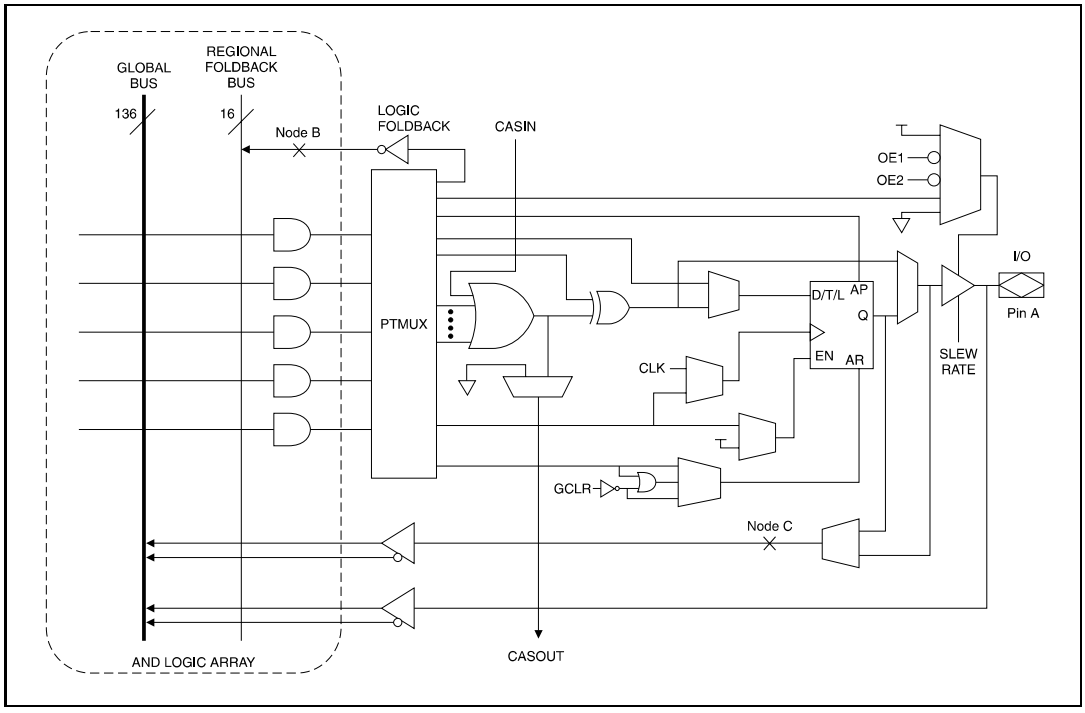


Abbildung 5.3: Schematische Darstellung einer Makrozelle

5.1 Datenaustausch mit dem Motorola MC68340

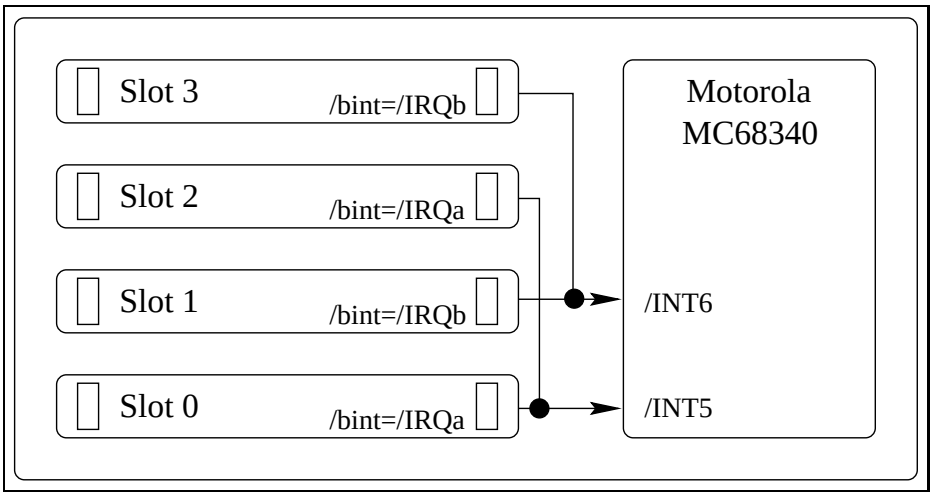
Das Ziel dieser Teilaufgabe besteht darin, den Austausch von Daten zwischen dem PIC-Prozessor und dem Motorola MC68340 zu ermöglichen. Neben der Übergabe von Anwenderprogrammen und Steuersignalen kommt dieser Schnittstelle noch in einer weiteren Hinsicht Bedeutung zu. Die Prozessorstreifen sind nicht wie beim großen Parallelprozessorsystem durch eigene Datenkanäle verbunden (vgl. Abschnitt 2.3), sondern der Austausch von Daten wird über das jeweilige Atmel PLD abgewickelt. Dazu senden die Recheneinheiten in anwendungsspezifisch festgelegten Zeiträumen ihre derzeit beste Lösung an den Motorola MC68340 und erhalten dafür im Gegenzug Daten anderer an der parallelen Abarbeitung beteiligter Prozessoren. Die Prozessorstreifen müssen dabei nicht explizit wissen, an welchen PIC-Prozessor die eigenen Chromosome geschickt werden, beziehungsweise woher die empfangenen Daten stammen.

Dieses Vorgehen hat entscheidende Vorteile für die Entwicklung eines 'Betriebssystemes' für den PIC17C43. Die festgelegte Anwenderstrategie zum Lösungsaustausch muß nur dem Kommunikationsprozessor bekannt sein und die Programme für die Prozessorstreifen können somit sehr einfach gehalten werden. Außerdem müssen keine Berechnungen unterbrochen werden, wenn die Topologie während der Laufzeit geändert wird.

Bezugnehmend auf Abbildung 5.1 erkennt man, daß die Datenübertragung nur über entsprechende Interrupts (beim PIC-Prozessor $\overline{pint}$, auf Seiten des Motorola MC68340 $\overline{bint}$) vonstatten gehen kann. Der eine Prozessor signalisiert mit einem Interrupt dem jeweils anderen, daß er diesem ein Byte Daten zusenden will. Das Auslesen der Daten geschieht

beim PIC-Prozessor über die 8 höherwertigeren Leitungen $AD[8:15]$ und beim Kommunikationsprozessor über die 8 niedrigsten Leitungen $BAD[0:7]$.

Die Interruptstruktur der eingesetzten kleinen Multiprozessorkarte wird durch die folgende Skizze verdeutlicht. Das vom logischen Baustein ausgehende Signal $\overline{bint}$ wird je nach Steckplatz des Prozessorstreifens entweder an $\overline{IRQa}$ oder $\overline{IRQb}$ weitergegeben, das auf Seiten des Motorola MC68340 mit der Leitung $\overline{INT5}$ beziehungsweise $\overline{INT6}$ verbunden ist.
Dadurch besteht prinzipiell die Möglichkeit beim Kommunikationsprozessor in zwei verschiedene Interrupt-Routinen zu verzweigen, d.h. aufgrund der aktivierten Leitung bereits eine Erkennung des auslösenden PIC-Prozessors vorzunehmen. Um eine Kompatibilität zur großen ISA-Steckkarte zu erreichen, wird diese Möglichkeit allerdings nicht umgesetzt, da dort alle eingehenden Interrupts auf $\overline{INT5}$ geführt werden.



Wichtig ist noch die Festlegung, welche Adreßbereiche von beiden Prozessortypen genutzt werden können, um das Atmel PLD der jeweiligen Recheneinheit zu kontaktieren. Beim PIC17C43 wird für diesen Zweck der Bereich $\$1000 \ldots \$1FFF$ reserviert, auf Seiten des Motorola MC68340 hängt dies von den entsprechenden Chip-Select-Definitionen ab. Derzeit werden für die vier Prozessorstreifen vom *Flash Bios* (vgl. Abschnitt 2.1) die folgenden Adressen bereitgestellt:

- PIC-Slot 0: Adreßbereich $\$FFFF8000 \ldots \$FFFF80FF$
- PIC-Slot 1: Adreßbereich $\$FFFF8100 \ldots \$FFFF81FF$
- PIC-Slot 2: Adreßbereich $\$FFFF8200 \ldots \$FFFF82FF$
- PIC-Slot 3: Adreßbereich $\$FFFF8300 \ldots \$FFFF83FF$

Das Schaltbild 5.4 gibt die umgesetzte Realisierung wieder und baut auf folgenden Schritten auf:

1. Aktiviert wird der logische Baustein durch einen PIC-Schreibzugriff auf die Adresse \$1000, dies führt beim MC68340 zu einem Interrupt, der bis zu einer Bestätigung aufrecht erhalten wird.
   Dazu muß zuerst die Gültigkeit der Adreßleitungen geprüft werden, dies wird bei

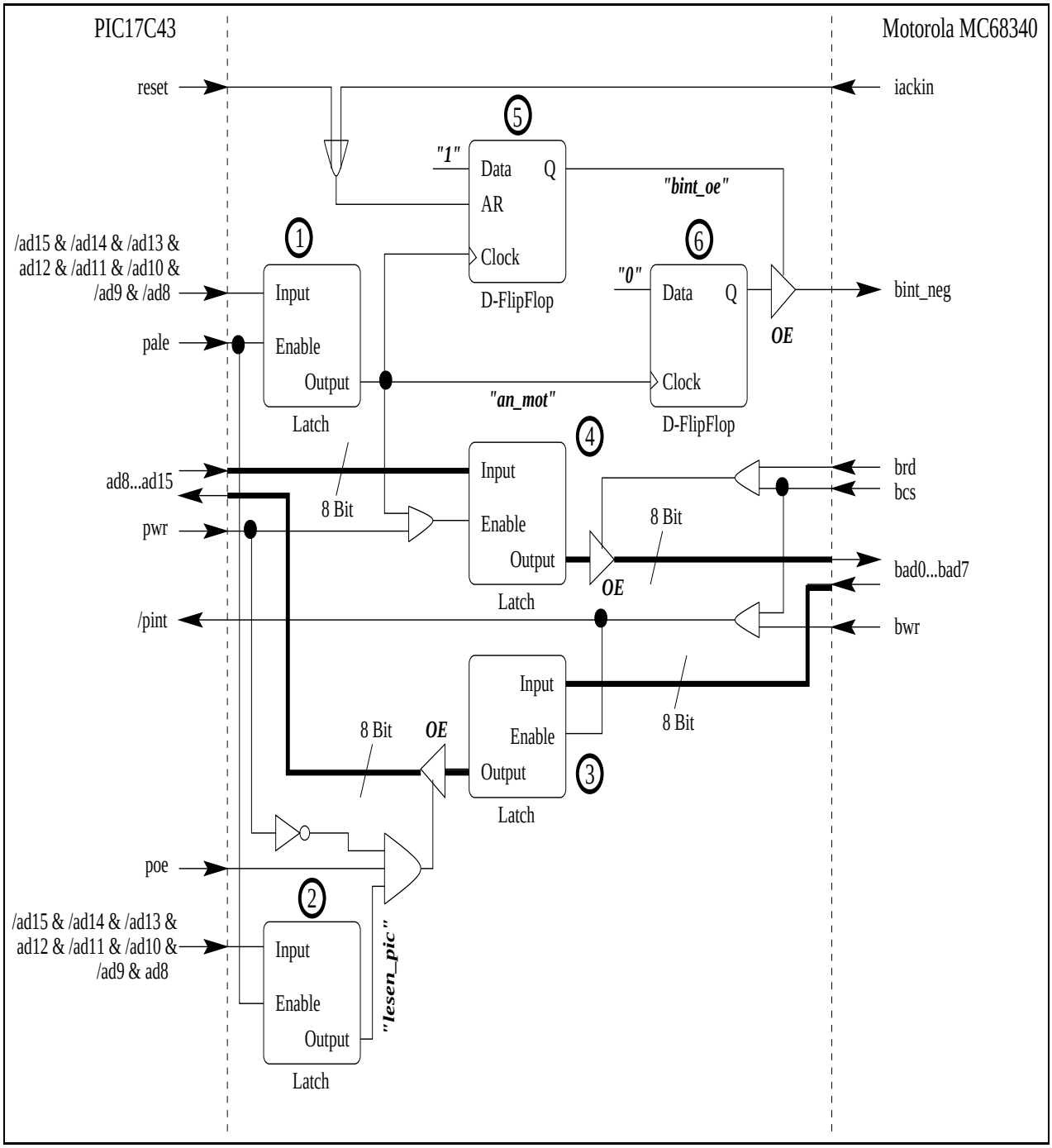


Abbildung 5.4: Schaltbild der Interrupt-Generierung

den kombinierten Adreß- und Datenleitungen des PIC17C43 gemäß dem Timing-Diagramm (Abbildung 5.5) durch das Signal $\overline{pale}$ (*pic address latch enable*) signalisiert. Bei einem positiven Ergebnis gemäß folgender Gleichung

$$\$1000 = \overline{AD15} \wedge \overline{AD14} \wedge \overline{AD13} \wedge AD12 \wedge \overline{AD11} \wedge \overline{AD10} \wedge \overline{AD9} \wedge \overline{AD8} \wedge pale$$

wird daraufhin das interne PLD-Signal *an_mot* erzeugt, das den Motorola-Interrupt auslöst (Bausteine **(1)** und **(6)** der Abbildung 5.4). Die Deaktivierung wird dann

durch $\overline{iackin}$ bewerkstelligt, womit über D-Flip-Flop **(5)** ein Tristate-Zustand erreicht wird. Eine logische Eins würde zwar auch eine Rücknahme des Interrupts bewirken, aber zwei Prozessorstreifen auf einer Leitung zusammengefaßt hätten eine *Bus Contention* zur Folge.

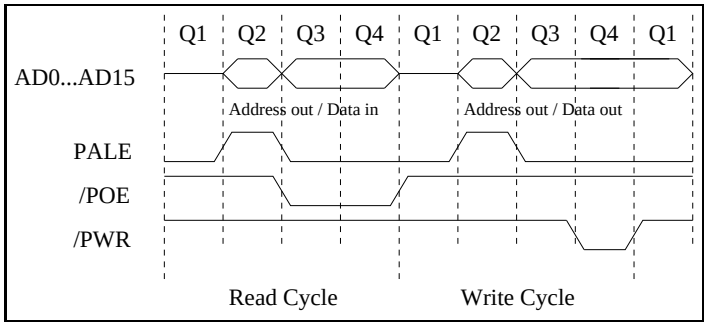


Abbildung 5.5: Timing-Diagramm des PIC17C43 bei Schreibzugriffen

Um ein Gefühl für die Realisierung derartiger Sachverhalte in *WinCupl* zu geben, sind nachfolgend die Gleichungen für *an_mot* dargestellt (Zeilen 69-77). Die vollständige Programmdatei ist dem Anhang A zu entnehmen.

```
/** Dekodierung der anliegenden, gueltigen                          **/
/** (pale = "pic address latch enable" = 1) Adresse.                **/
/** ad15..8 =  $1000  ==> an_mot = 1 ==> Datenuebergabe an Motorola **/
/** ad15..8 <> $1000  ==> an_mot = 0 ==> Ansprechen des externen RAM **/
/** Compiler waehlt nichtbenutzte logische Zelle.                   **/
pinnode   = an_mot;
an_mot.l  = !ad15.io & !ad14.io & !ad13.io & ad12.io &
            !ad11.io & !ad10.io & !ad9.io  & !ad8.io;
an_mot.le = pale;
```

Mit dem Schlüsselwort *pinnode* wählt der Compiler eine freie (bisher nicht genutzte) Makrozelle, deren Ausgang dabei direkt mit dem vergebenen Namen zugänglich ist. Die Suffixe *.l* und *.le* definieren die Zelle als transparentes Latch und stellen den Latcheingang beziehungsweise das 'Latch-Enable' dar. Eine Besonderheit ist die Endung *.io*, mit welcher der entsprechende Pin verwendet wird und nicht der Ausgang der dahinterliegenden Makrozelle, die zwar gleich bezeichnet ist aber durchaus andere Daten speichern kann.

2. Der Zeitpunkt, an dem die Daten beim Schreibvorgang aus vorigem Punkt gültig sind, wird durch *pwr* zusammen mit *an_mot* signalisiert. Das Datenwort wird daraufhin in den acht Latches **(4)** zwischengespeichert. Ein Timing-Diagramm, das diesen Sachverhalt schematisch darstellt ist in Abbildung 5.6 zu sehen.
   Angefordert werden kann das Datenwort vom Motorola MC68340 durch einen Lesezugriff auf die Anfang dieses Abschnittes festgelegten I/O-Zellen. Die Generierung der Chip-Select- und Schreibsignale übernimmt dabei die *Glue Logic Gal22V10* (beschrieben in Teilabschnitt 2.1).

3. Der Datenweg vom Kommunikationsprozessor zu einer der vier Recheneinheiten folgt einem Schema, das analog zu den obigen zwei Punkten implementiert wurde und aus dem unteren Teil der Abbildung 5.4 ersichtlich ist.

4. Abschließend sei erwähnt, daß die Umsetzung keine eigene *Handshake*-Leitung zur Bestätigung eines Datentransfers zwischen den beiden Prozessortypen beinhaltet. Bei den Algorithmen ist dies zu berücksichtigen, um Datenverluste zu vermeiden.

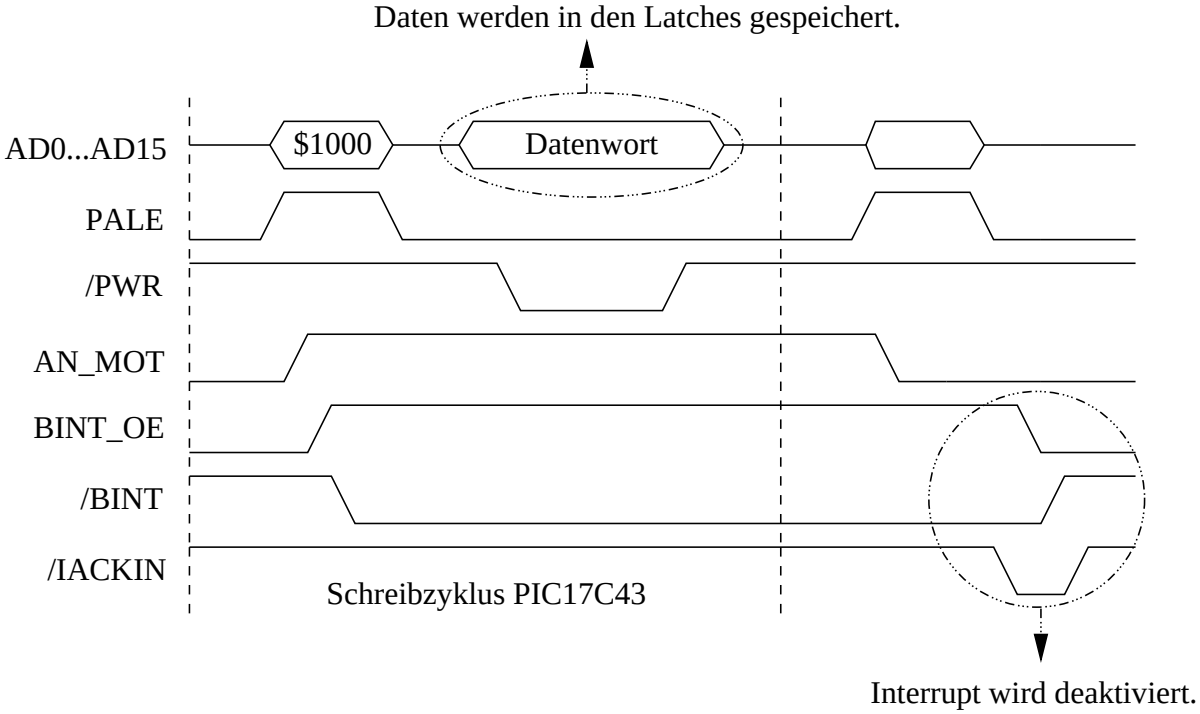


Abbildung 5.6: Vereinfachtes Timing-Diagramm der Interrupt-Generierung

5.2 Ansprechen des externen Speichers *CXK516100*

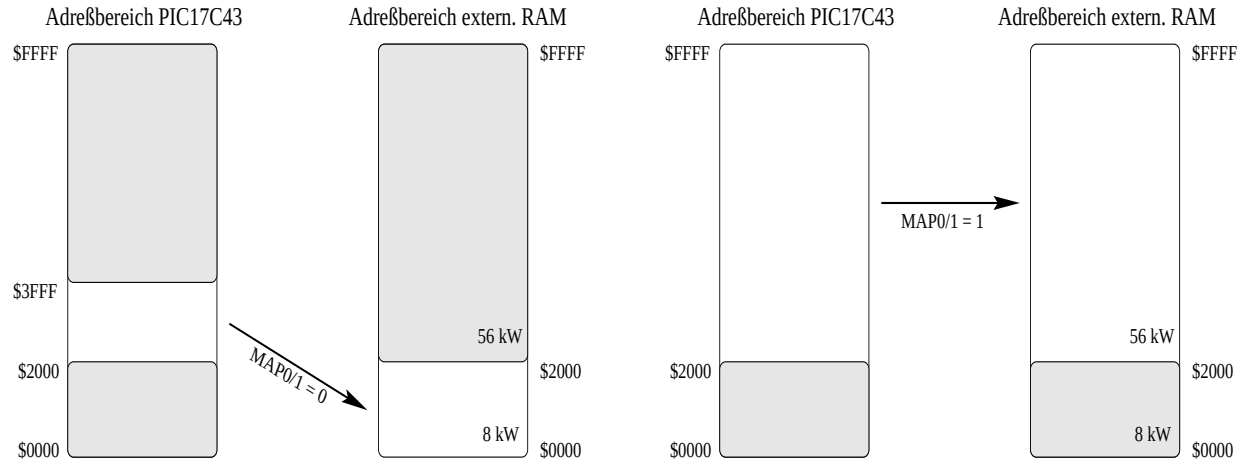
Mit den 16 Bit Adreßleitungen des PIC-Prozessors ist es möglich, einen Adreßbereich von $\$0000 - \$FFFF$ abzudecken. Die Leitungen $AD[0:15]$ werden dazu genutzt, um das externe RAM mit seinen insgesamt 64 kWord Kapazität im gleichen Adreßrahmen zu betreiben. Allerdings sind hierbei zwei Einschränkungen zu machen, welche die Notwendigkeit einer Speicherpartitionierung auf logischer Ebene unumgäglich machen.

Zum einen wird mit den Adressen $\$0000 - \$0FFF$ der PIC-interne 4 kByte große EPROM-Bereich, in dem das 'Betriebssystem' steht, referenziert, d.h. in diesem Speicherbereich liegen keine gültigen Adressen am RAM an (vgl. Abbildung 5.8, entnommen aus [12]). Mit derselben Argumentation stellt zum anderen auch der Bereich $\$1000 - \$1FFF$ keinen Zugriff auf den externen Speicher dar, weil dieser im vorigen Teilabschnitt für die Kommunikation zum Motorola MC68340 reserviert wurde. Ohne eine zusätzliche Logik sind also nur die 'oberen' 56 kWord des Speicherbausteines nutzbar.

Aus diesem Grund wird der Ausgangs-Pin $RB7$ $(= MAP0/1)$ des PIC-Prozessors genutzt, um zusammen mit den Adreß-Pins $AD[0:15]$ anzuzeigen, welcher Bereich des *RAM CXK516100* angesprochen werden soll. Deshalb werden lediglich die Leitungen $AD[1:11]$,

$AD14$ und $AD15$ vom PIC-Prozessor auf den Speicher geführt, während die Signale $AD12$ und $AD13$ vom PLD-Baustein entsprechend der Verschiebung des Adreßbereiches aufbereitet werden (dazu werden zwei neue Leitungen $A12$ und $A13$ definiert; vgl. Abbildung 5.1).

In Abhängigkeit von $MAP0/1$ sind zwei Fälle zu unterscheiden:



(a) $MAP0/1 = 0$ (b) $MAP0/1 = 1$

Abbildung 5.7: Partitionierung des externen Speichers *CXK516100*

- $MAP0/1 = 0$: Gültige Adressen sind nur im Bereich $\$2000 - \$3FFF$ zulässig, in Binärschreibweise entspricht das $001X \ldots X$. Um eine Transformation in den Speicherraum $\$0000 - \$1FFF$ $(000X \ldots X)$ zu erhalten, ist lediglich die Negierung von $AD13$ nötig: $A12 = AD12$, $A13 = AD13 \wedge MAP0/1 = 0 = \overline{AD13}$.
- $MAP0/1 = 1$: Gültige Adressen sind nur im Bereich $\$2000 - \$FFFF$ möglich, eine weitere Dekodierung der Signale ist nicht notwendig: $A12 = AD12$,
  $A13 = AD13 \wedge MAP0/1 = AD13$.

Selbstverständlich müssen vom Atmel CPLD ATF1500 noch die zugriffsrelevanten Signale des externen RAM erzeugt werden. Im Einzelnen sind dies $\overline{mce}$ (*Memory Chip Enable*) für Lese- und Schreibzugriffe sowie $\overline{mwe}$ (*Memory Write Enable*) für Schreibzugriffe. Man verifiziert leicht, daß die realisierten Verknüpfungen in Abbildung 5.9 erst die Gültigkeit der Adreßbereiche obiger Fallunterscheidung untersuchen, bevor die Leitungen aktiviert und somit Zugriffe ermöglicht werden.

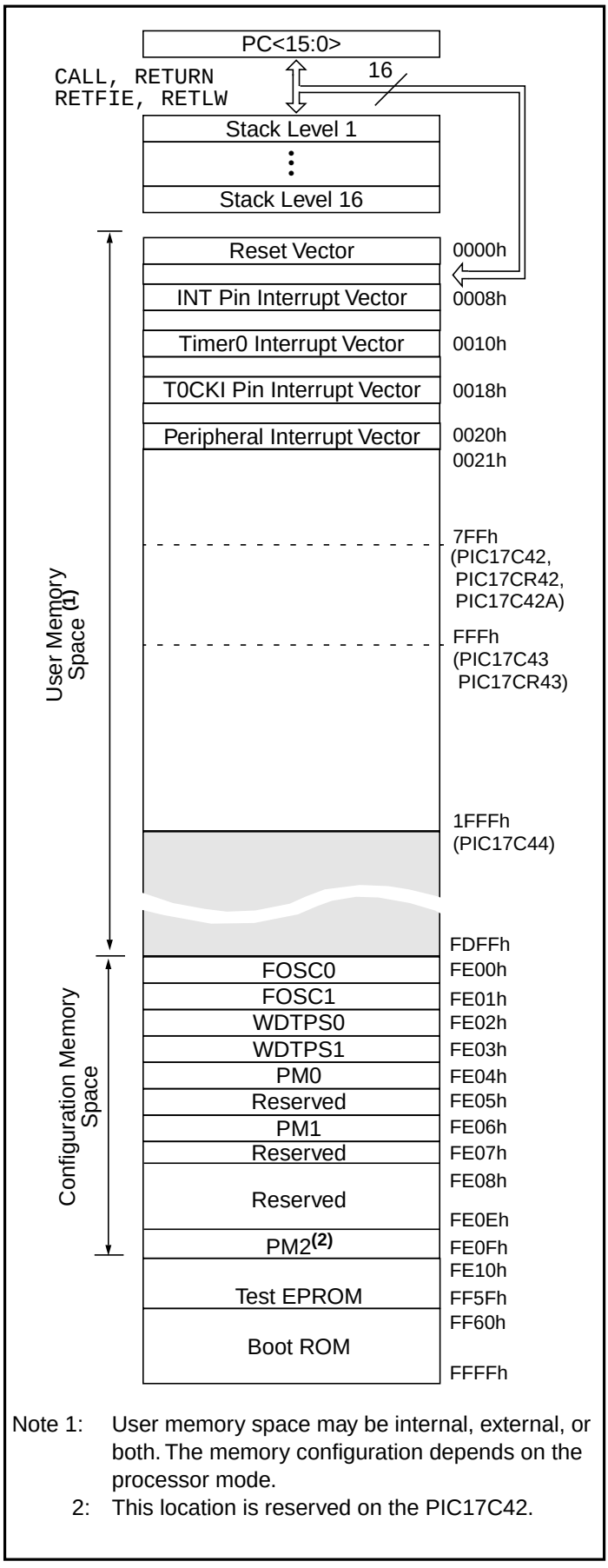


Abbildung 5.8: Speicherorganisation des Microchip PIC17C43

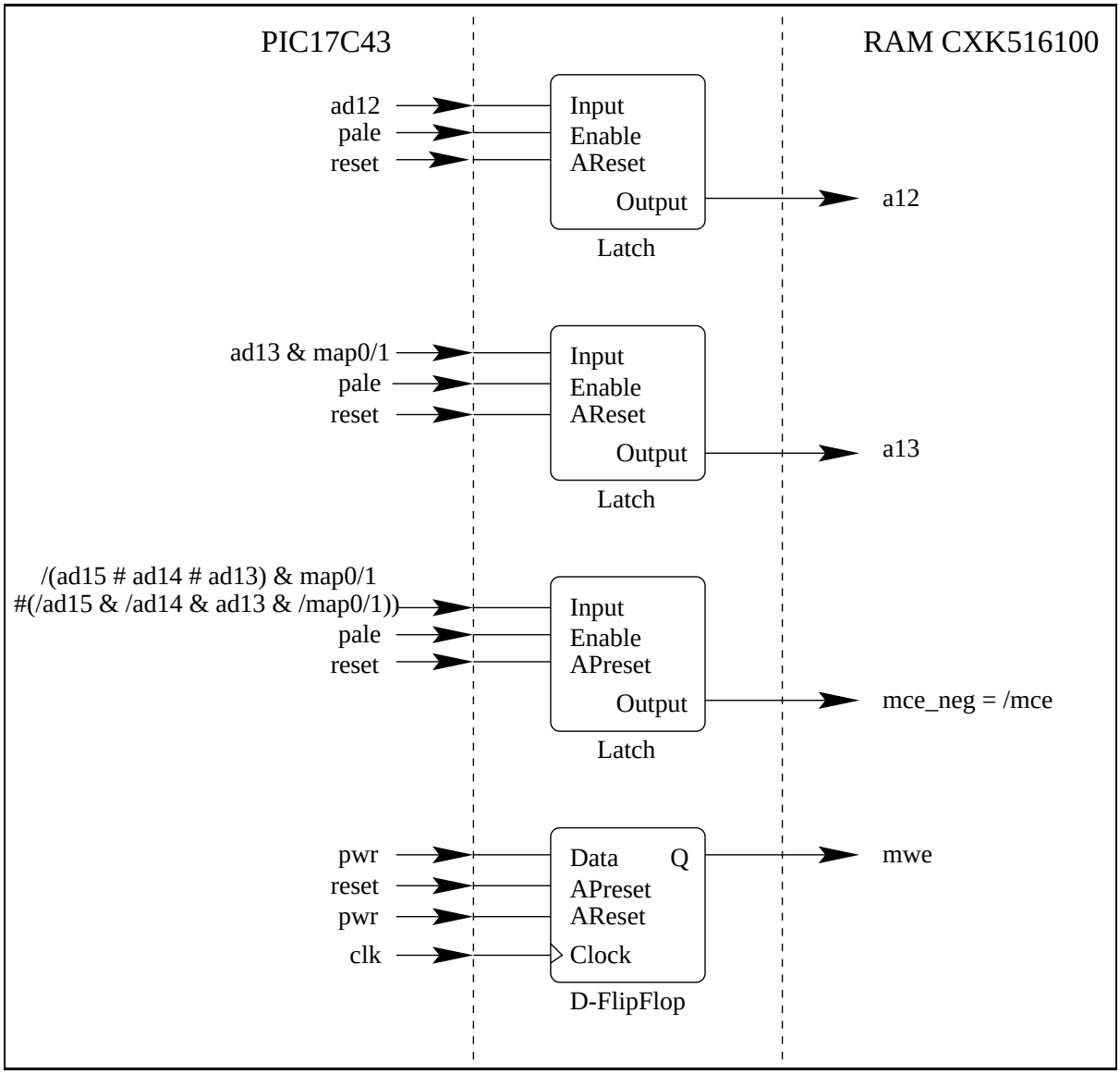
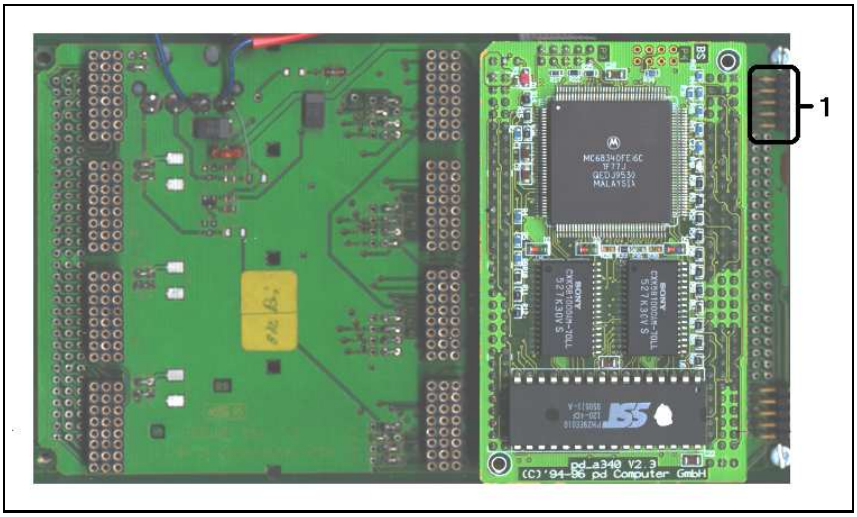


Abbildung 5.9: Schaltbild der Realisierung des Speicherzugriffes

Kapitel 6

Kontakt vom Anwender zum Multiprozessorsystem

Der Kontakt zum Anwender (PC) wird über eine Steckverbindung (Punkt **(1)** der nachfolgenden Zeichnung) hergestellt. Mittels dieses Steckers wird der Kanal B der seriellen Schnittstellen des Motorola MC68340 (Teil des Kommunikationsprozessors) mit dem Pendant auf Seiten des Computers verbunden. Im Normalfall ist dies entweder der Port *COM1* oder *COM2*.



Hierbei ist ein zusätzlicher Baustein vonnöten, weil die TTL/CMOS-Pegel des MC68340 nicht mit denen der RS232-Spezifikation des Rechners übereinstimmen. Zeichnung 6.1 zeigt schematisch die eingesetzte Schaltung.

Nur über diesen Weg ist es dem Anwender mit dem *Mtty*-Terminalprogramm (im Verzeichnis `\Diplomarbeit\Software\Terminal-Programme`) möglich, Anweisungen und Steuerbefehle dem Multiprozessorsystem weiterzugeben. Dies betrifft all diejenigen Punkte des Ablaufschemas 4.1, bei denen der Kommunikationsprozessor auf Entscheidungen des Anwenders angewiesen ist oder Ausgaben am PC-Bildschirm vornehmen soll.

Als Konfiguration wurde ein Datentransfer von 38400 Baud bei 8 Datenbits, einem Stopp-Bit und keiner Parität eingetragen. Abbildung 6.2 zeigt eine Momentaufnahme: dem An-

wender wird vom Kommunikationsprozessor das beste Resultat des 8-Städte-Problemes mitgeteilt.

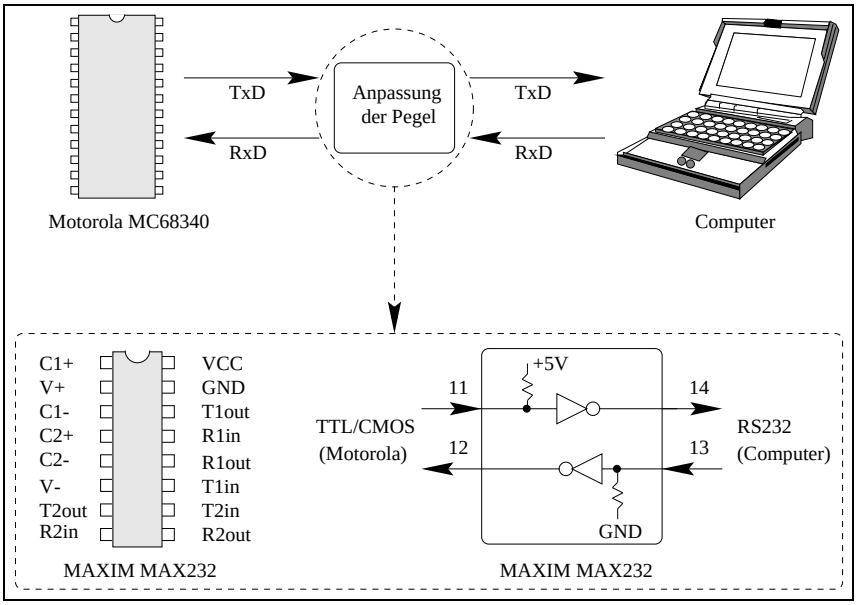


Abbildung 6.1: RS232-Verbindung zwischen Motorola und einem Computer

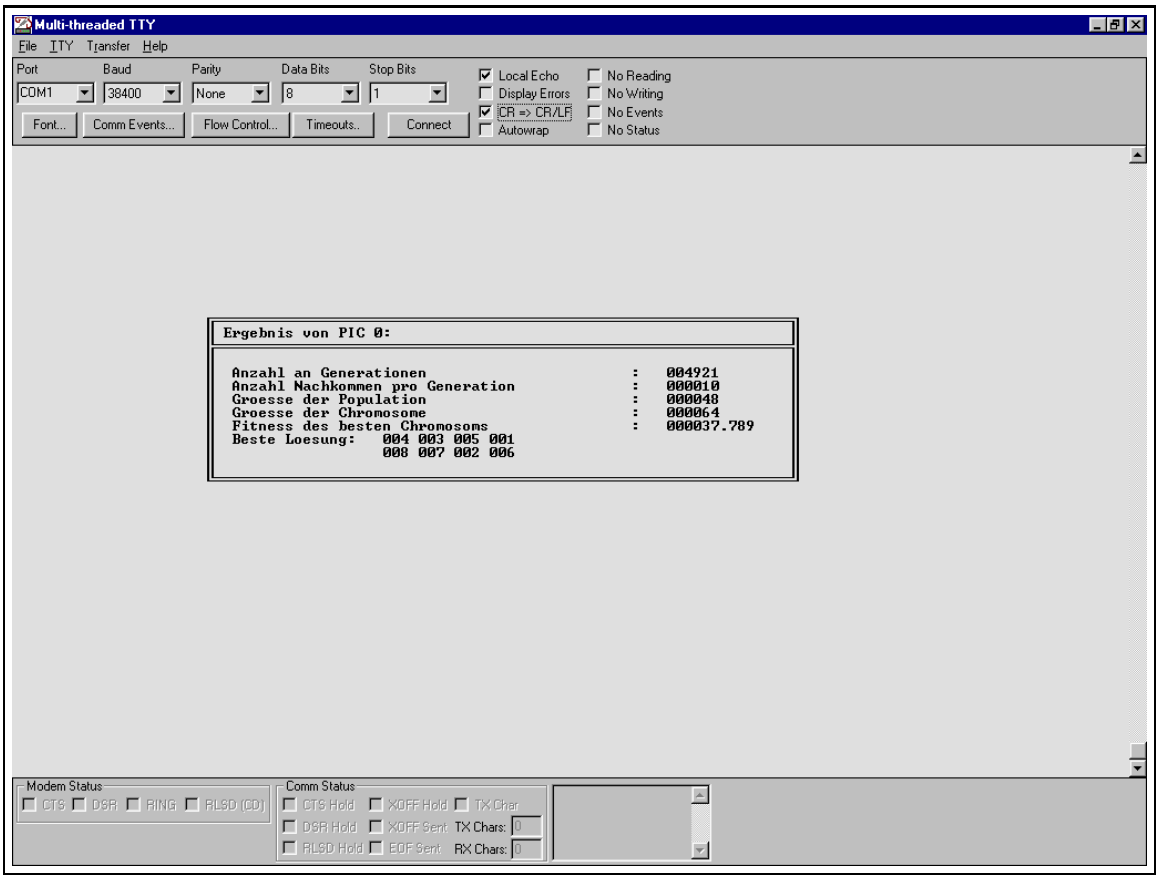


Abbildung 6.2: Das verwendete *Mtty*-Terminalprogramm

Kapitel 7

Programmierung des Motorola MC68340

Die Funktionalität des Kommunikationsprozessors wird maßgeblich durch das Schema 4.1 bestimmt. Dabei wurde festgelegt, daß die Prozessorstreifen bei Aufgabenstellungen, die mit genetischen Algorithmen gelöst werden sollen, in festgelegten Intervallen ihre derzeit besten Resultate gemäß der Anwendertopologie austauschen. Die Endergebnisse werden nach Abschluß des Verfahrens dann dem Benutzer präsentiert.

Diese Vereinbarungen führten zu einem 'Betriebssystem' für den Motorola MC68340, dessen Ablauf die folgende Skizze verdeutlicht. Die in Klammern angegebenen Abschnitte widmen sich den jeweiligen Teilaufgaben und erläutern ihre Umsetzung.

1. **Grundlegende Einstellungen (Abschnitt 7.1).**
2. **Initialisieren der seriellen Schnittstelle (Abschnitt 7.2).**
3. **Testen der belegten PIC-Slots mit anschließender Ausgabe einer Statusmeldung (Abschnitt 7.3).**
4. **Empfang der Programmdateien zur Weitergabe an die Prozessorstreifen (Abschnitt 7.4).**
5. **Hauptschleife: entsprechend der jederzeit änderbaren Anwendertopologie den Datenaustausch zwischen den Recheneinheiten steuern (Abschnitt 7.5).**
6. **Nach Abschluß der Abarbeitung alle Endergebnisse speichern (ebenfalls Abschnitt 7.5).**
7. **Ausgabe der besten Resultate (Abschnitt 7.6).**
8. **Falls ein Neustart erforderlich ist $\Rightarrow$ Punkt 3.**

Wie schon in Kapitel 2 mehrfach erwähnt wurde, bildet der Motorola MC68340, dessen Module und Signalgruppen in Abbildung 7.1 dargestellt sind, den Kern des Kommunikationsprozessors und wurde wie die anderen Software-Realisierungen in Assembler programmiert.

Die Handbücher zu diesem Prozessortyp sind in den Literaturangaben [6], [14], [16], [24] und [25] und auch im PDF-Format auf der CD-ROM im Verzeichnis `Diplomarbeit\Datenblaetter\Motorola MC68340` zu finden. Im Anhang B beziehungsweise im Verzeichnis `Diplomarbeit\Betriebssystem\Motorola` sind die kompletten Programm-Dateien enthalten.

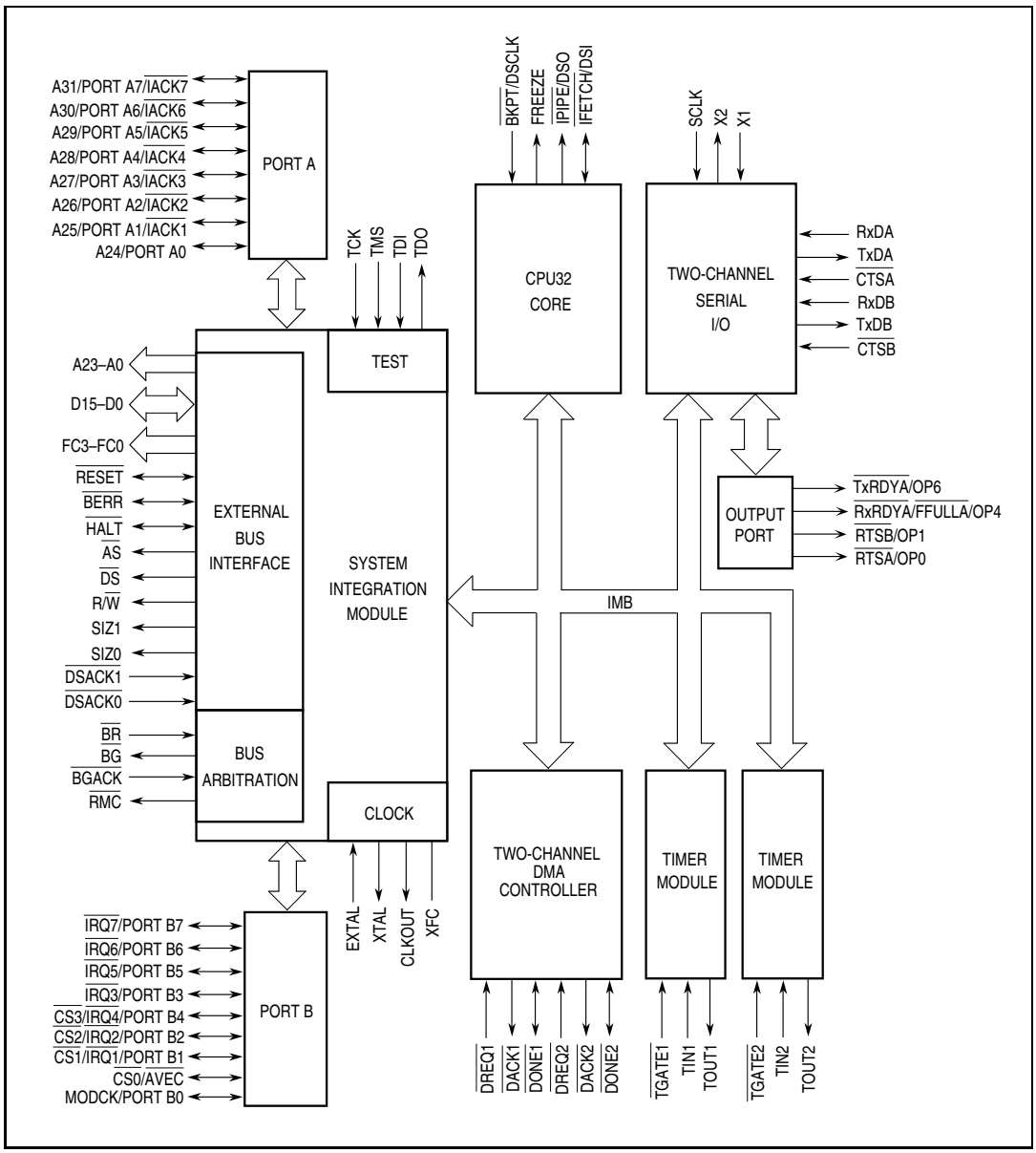
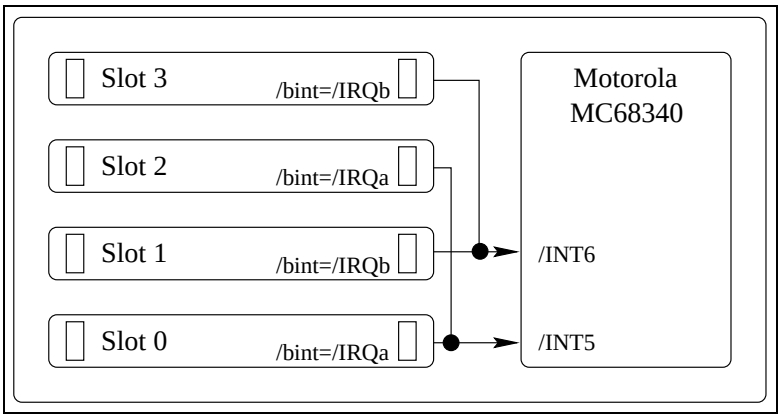


Abbildung 7.1: Module und Signalgruppen des Motorola MC68340

7.1 Notwendige Grundeinstellungen

Die ersten vorzunehmenden Parametereinstellungen dienen der Anpassung an die zugrundeliegende Hardware. Neben der Bereitstellung der *Chip-Select*-Signale zum Ansprechen der Peripherie-Module muß interruptgestützt der Datenaustausch mit den Prozessorstreifen ermöglicht werden. Aus diesem Grund ist noch einmal schematisch das kleine Multipro-

zessorsystem mit den vier Prozessorstreifen und den entsprechenden Interrupt-Leitungen abgebildet:



Die beiden Signale $INT[5:6]$ können dadurch aktiviert werden, daß nur die Maskierung (Deaktivierung) für die Interrupts niederer Priorität ($INT[1:4]$) aufrecht erhalten wird. Die notwendigen Einstellungen werden durch folgenden Befehl übernommen, der das *Status Register* des Motorola MC68340 entsprechend konfiguriert (Zeile 20 in `tonux.a`):

```
move.w #$2400,sr                    * Interrupt level = 4, Supervisor Mode
```

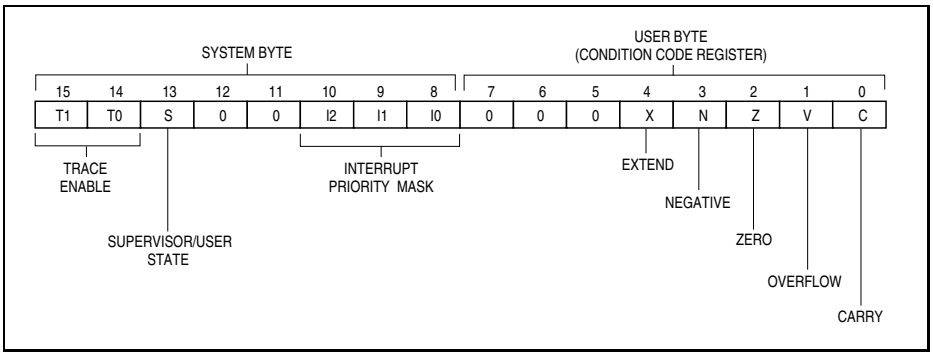


Abbildung 7.2: Das *Status Register* des Motorola MC68340

Hiermit werden prinzipiell Interrupts bis zur CPU weitergeleitet. Als nächstes muß definiert werden, in welche Routine der Kommunikationsprozessor in einem solchen Fall verzweigen soll. Diese Festlegungen werden mit der sogenannten *Vektor-Tabelle* getroffen, in der momentan allerdings nur vier Sprungadressen eingetragen sind (man beachte, daß alle Kontaktanfragen der Prozessorstreifen mit einer Funktion - `picx_interrupt` - ausgewertet werden, vgl. `vector.h`):

```
*** Vektor-Tabelle:                Nr.: Bedeutung
  dc.l STACK_INI                     0: Reset: Initial Stack Pointer
  dc.l ga_start                      1: Reset: Initial Program Counter
  ...                               ...
  dc.l picx_interrupt               29: Level 5 Interrupt Autovector (INT5)
  dc.l picx_interrupt               30: Level 6 Interrupt Autovector (INT6)
  ...                               ...
```

Aufgrund der vorgegebenen Verdrahtung des kleinen Multiprozessorsystemes wird *PORT B* des Motorola MC68340 verwendet, um neben vier Interrupt-Leitungen auch die vier nötigen Chip-Select-Leitungen bereitzustellen (vgl. Abbildung 7.1). Die Zeilen 52-56 der Datei `basics.a` enthalten die notwendigen Befehle und werden durch Abbildung 7.3 verdeutlicht (*PORT A* stellt die Interrupt-bestätigenden Signale $\overline{iackin5/6}$ bereit):

```
move.b  #$60,avr        * Autovektor fuer IRQ-Level 5 und 6
bclr    #12,simcr       * FIRQ = 0
move.b  #$FF,pparb      * PORTB festlegen: 4xIRQ + 4xCS
move.b  #$00,ppara1     * PORTA als "lack" nutzen
move.b  #$FF,ppara2
```

Signal	Pin Function			
	FIRQ = 0 PPARB = 0	FIRQ = 0 PPARB = 1	FIRQ = 1 PPARB = 0	FIRQ = 1 PPARB = 1
IRQ7	PORTB7	IRQ7	PORTB7	IRQ7
IRQ6	PORTB6	IRQ6	PORTB6	IRQ6
IRQ5	PORTB5	IRQ5	PORTB5	IRQ5
IRQ3	PORTB3	IRQ3	PORTB3	IRQ3
CS3	CS3	CS3	PORTB4	IRQ4
CS2	CS2	CS2	PORTB2	IRQ2
CS1	CS1	CS1	PORTB1	IRQ1
CS0	CS0	CS0	AVEC	AVEC
MODCK	PORTB0	MODCK	PORTB0	MODCK

NOTE: MODCK has no function after reset.

Abbildung 7.3: Der *PORT B* des Motorola MC68340

Neben den grundlegenden Hardware-Anpassungen sind die verwendeten Variablen für das Gesamtverständnis der Implementierung des 'Betriebssystemes' wichtig. Tabelle 7.1 zeigt die in der Datei `def.h` vorgenommenen Definitionen und ihre Bedeutung:

Name	Bedeutung
`mode`	Betriebsmodus des Motorola MC68340 $(0, \ldots, 5, 10, \ldots, 13)$
`correct_pics`	Genutzte PIC-Slots (vgl. Abschnitt 7.3)
`results`	Recheneinheiten, welche die Abarbeitung beendet haben
`chr_size`	Länge der Chromosome
`pic0/1/2/3`	I/O-Adresse der jeweiligen PLD-Bausteine
`control0/1/2/3`	Aktuelle Steuersignale (vgl. Abschnitt 7.5)
`topo0/1/2/3`	Topologie (vgl. Abschnitt 7.5)
`fitness0/1/2/3`	Fitneß der derzeit gespeicherten Chromosome
`gen0/1/2/3`	Anzahl benötigter Generationen
`child_size0/1/2/3`	Anzahl Nachkommen pro Generation
`pop_size0/1/2/3`	Populationsgröße

Tabelle 7.1: Die wichtigsten Variablen des MC68340-Betriebssystemes

Wichtig für den korrekten Ablauf ist insbesondere die Variable `Mode`. Ähnlich zu Kapitel 8 wird ein zustandsbasiertes Automatenmodell für den Kommunikationsprozessor entworfen, um die verschiedenen Abarbeitungszustände unterscheiden zu können. Besonders im Hinblick auf die Interrupt-Routine `picx_interrupt`, zum Auswerten eingehender Steuersignale von den Prozessorstreifen, ist die Belegung von größter Bedeutung:

`Mode`	**Bedeutung**
0	Empfang des Speichertest-Ergebnisses von PIC-Slot 0
1	Empfang des Speichertest-Ergebnisses von PIC-Slot 1
2	Empfang des Speichertest-Ergebnisses von PIC-Slot 2
3	Empfang des Speichertest-Ergebnisses von PIC-Slot 3
4	Modus zur Dateiverteilung an die Recheneinheiten
5	'Normal'-Modus: Datenaustausch steuern
10	Chromosom-Empfang von PIC-Slot 0
11	Chromosom-Empfang von PIC-Slot 1
12	Chromosom-Empfang von PIC-Slot 2
13	Chromosom-Empfang von PIC-Slot 3

Tabelle 7.2: Die Belegungen der Variablen `Mode`

Aufgrund des 'Betriebszustandes' ist es somit einfach, die notwendigen Operationen eingehender Kontaktanfragen der Prozessorstreifen vorzunehmen. Abbildung 7.4 stellt das in `irq.a` für die Routine `picx_interrupt` umgesetzte Konzept schematisch vor:

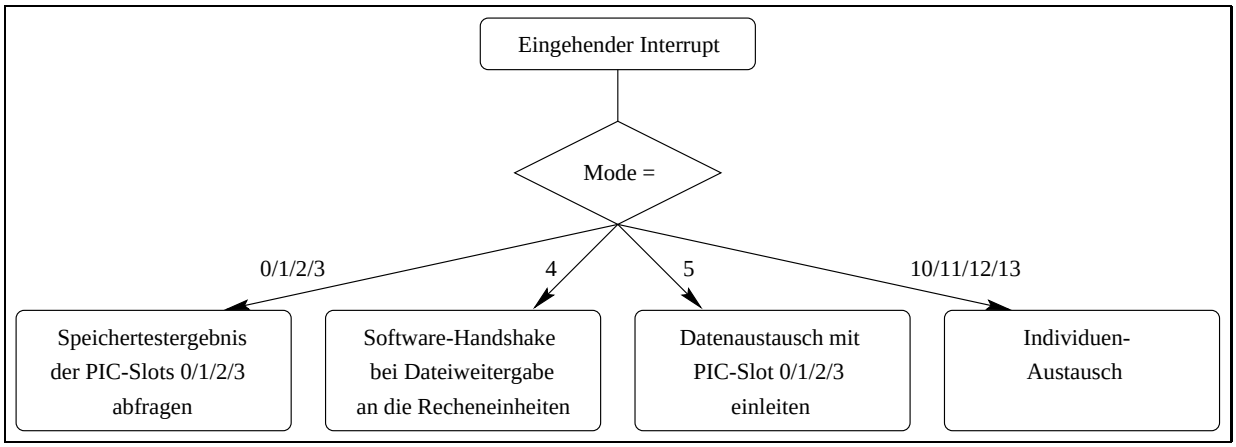


Abbildung 7.4: Die Routine `picx_interrupt`

7.2 Initialisierung der seriellen Schnittstelle

Um den Kontakt zum Benutzer des Multiprozessorsystemes aufzubauen, wird Kanal B der beiden seriellen Schnittstellen des Motorola MC68340 genutzt. Abbildung 7.5 (entnommen aus [14]) zeigt die Hardware mit insgesamt vier Schieberegistern für den Empfang und zwei Registern zum Senden von Datenworten (ASCII-Nummern):

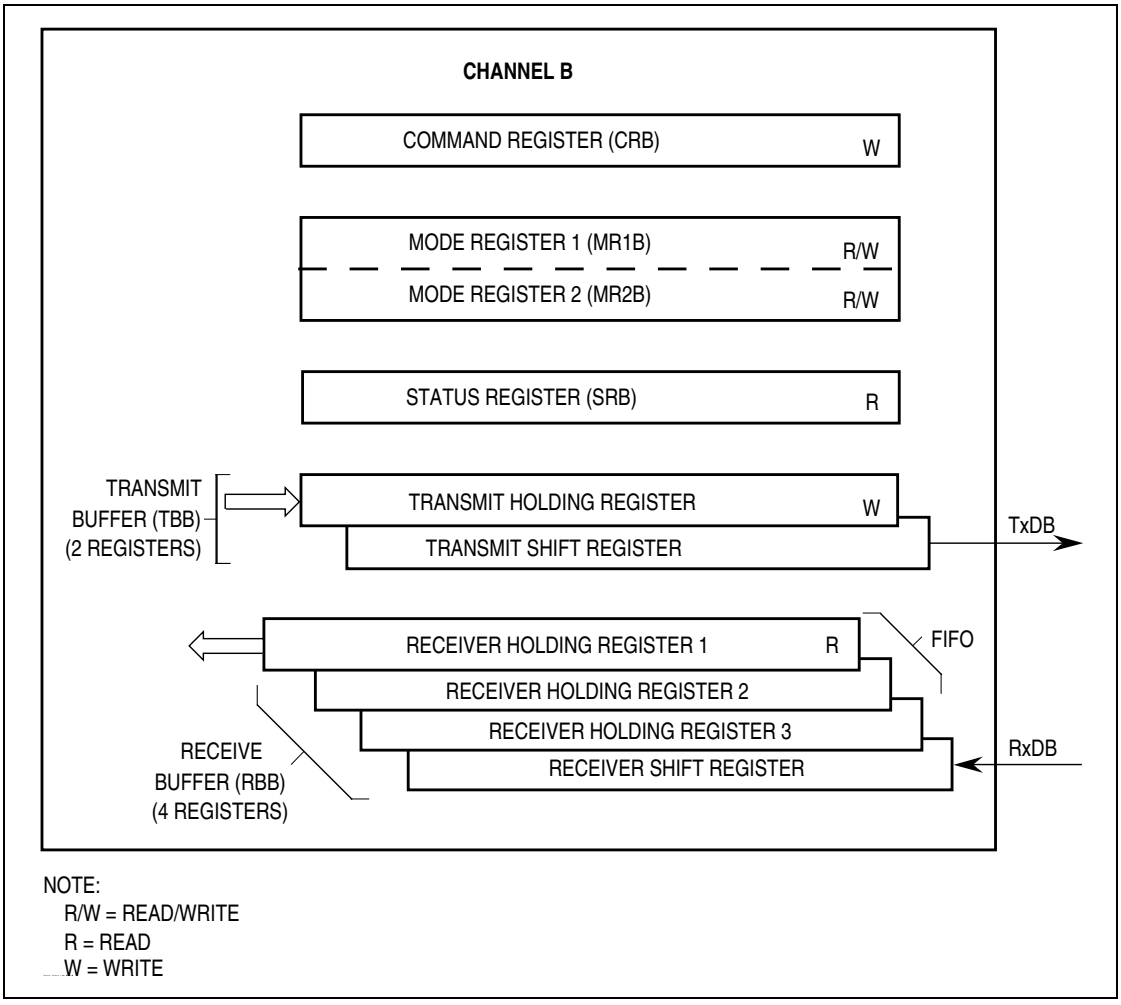


Abbildung 7.5: Die serielle Schnittstelle des MC68340 (Kanal B)

In Kapitel 6 wurde eine Datenübertragung von 38400 Baud bei acht Datenbits, einem Stopp-Bit und keiner Parität festgelegt. Im Einzelnen wird dies in der Datei `uart.a` mit den folgenden Befehlen bewerkstelligt (zum besseren Verständnis sind die konfigurierten Register mit den jeweiligen Seitenzahlen aus [14] angegeben):

1. Wahl des sogenannten 'Baud Rate Generator Set 2':

```
move.b #$80,ser_acr
```

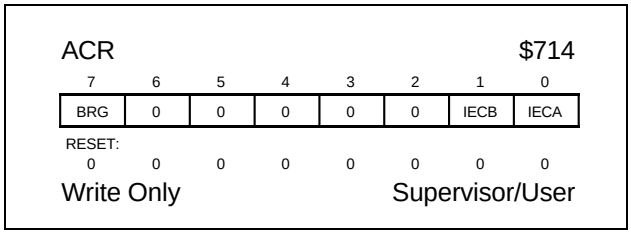


Abbildung 7.6: Das Register `SER_ACR`, 7-32

2. Baudrate auf 38400 Baud festlegen:

```
move.b #$DD,ser_csrb
```

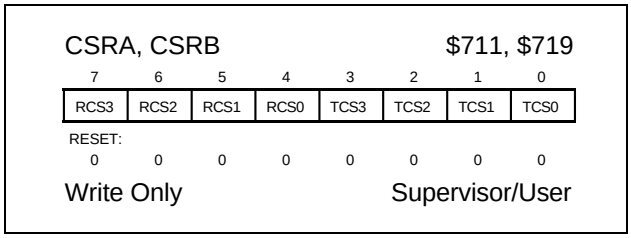


Abbildung 7.7: Das Register SER_CSRB, 7-25

3. Keine Parität bei 8 Datenbits:

```
move.b #$93,ser_mr1b
```

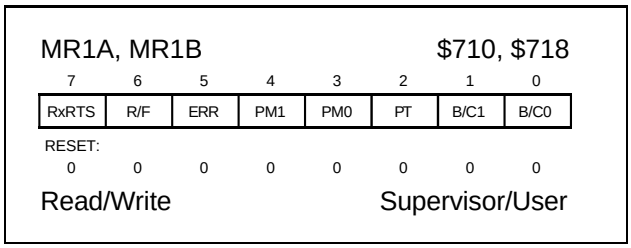


Abbildung 7.8: Das Register SER_MR1B, 7-22

4. Ein Stopp-Bit eintragen:

```
move.b #$07,ser_mr2b
```

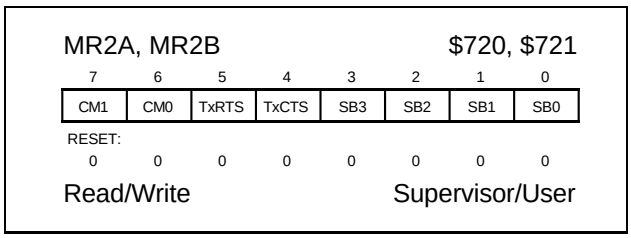


Abbildung 7.9: Das Register SER_MR2B, 7-38

Beim Empfang von Datenworten wird auf eine Interrupt-Auslösung verzichtet. Stattdessen wird 'Polling' angewendet, was aufgrund der eindeutigen Programmstruktur keinerlei Probleme bereitet. Die folgenden Befehle zeigen die dafür entwickelte Funktion RECEIVERB_LOOP. Den Schieberegistern kommt dabei keine Bedeutung zu, das erhaltene Zeichen wird sofort ausgelesen:

```
*************************************************************************
***
*** Name....: RECEIVERB_LOOP
*** Funktion: Wartet auf ein Zeichen des Benutzers.
***           In einer "Neben-Schleife" wird eine Zufallszahl fuer
***           die PICs bestimmt.
*** Register: D6 enthaelt das empfangene Zeichen.
***
RECEIVERB_LOOP
  addi.b #01,random             Zufallswert inkrementieren
  btst.b #0,ser_srb             Bit0 = 0 --> keine (neuen) Daten
  beq    RECEIVERB_LOOP         Bit0 = 1 --> neue empfangene Zeichen
  move.b ser_rbb,d6             ser_rbb = "Receiver Buffer B"
  rts
***
*************************************************************************
```

Für das Senden von Daten stellt die Datei `mot2pc.a` nützliche Routinen bereit, die beispielsweise das Generieren von Rahmen beliebiger Größe ($D7$ = Breite des Rahmens) ermöglichen und im wesentlichen auf die Basisfunktion `SEND_CHB_BYTE` zurückgeführt werden:

```
*********************************************************
***
*** Name....: SEND_CHB_BYTE
*** Funktion: Sendet ein Byte an den PC.
*** Register: zu sendendes Byte muss in D4 an der niedrigsten
***           Position stehen.
***
SEND_CHB_BYTE
  btst.b  #2,ser_srb            Warten bis PC empfangsbereit
  beq     SEND_CHB_BYTE
  move.b  d4,ser_tbb
  rts
***
*********************************************************
```

7.3 Testen der belegten PIC-Slots

Eine ebenfalls in der Initialisierungsphase wichtige Aufgabe ist das Erkennen der PIC-Slots, in denen ein Prozessorstreifen eingesteckt ist und der sich nach einer Testroutine als funktionstüchtig erweist. Denn nur diese Einheiten können im weiteren Verlauf Problemstellungen bearbeiten und lösen. Für Abschnitt 7.5 ist dabei zusätzlich wichtig, daß auch nur aktive Prozessorstreifen Kontaktanfragen an den Kommunikationsprozessor stellen können, was in der dortigen Routine zu beachten ist.

Hierzu führt jeder Prozessorstreifen zu Beginn des Programmes einen Test seines externen Speichers durch, und wird danach vom Motorola MC68340 kontaktiert (d.h. ein Interrupt ausgelöst). Die Antwort muß in einem bestimmten Zeitfenster erfolgen, ansonsten dient dies dem Kommunikationsprozessor als Indiz für eine defekte oder nicht bestückte Recheneinheit. Die Variable `correct_pics` speichert den festgelegten Sachverhalt gemäß folgender Definition:

- Bit $x = 0$ $\Rightarrow$ aktive Recheneinheit in Slot x.
- Bit $x = 1$ $\Rightarrow$ nicht genutzter PIC-Slot x.

Die entsprechenden Statusmeldungen dieser Teilaufgabe zeigt Abbildung 7.10, die Realisierung ist in der Datei `pictest.a` zu finden.

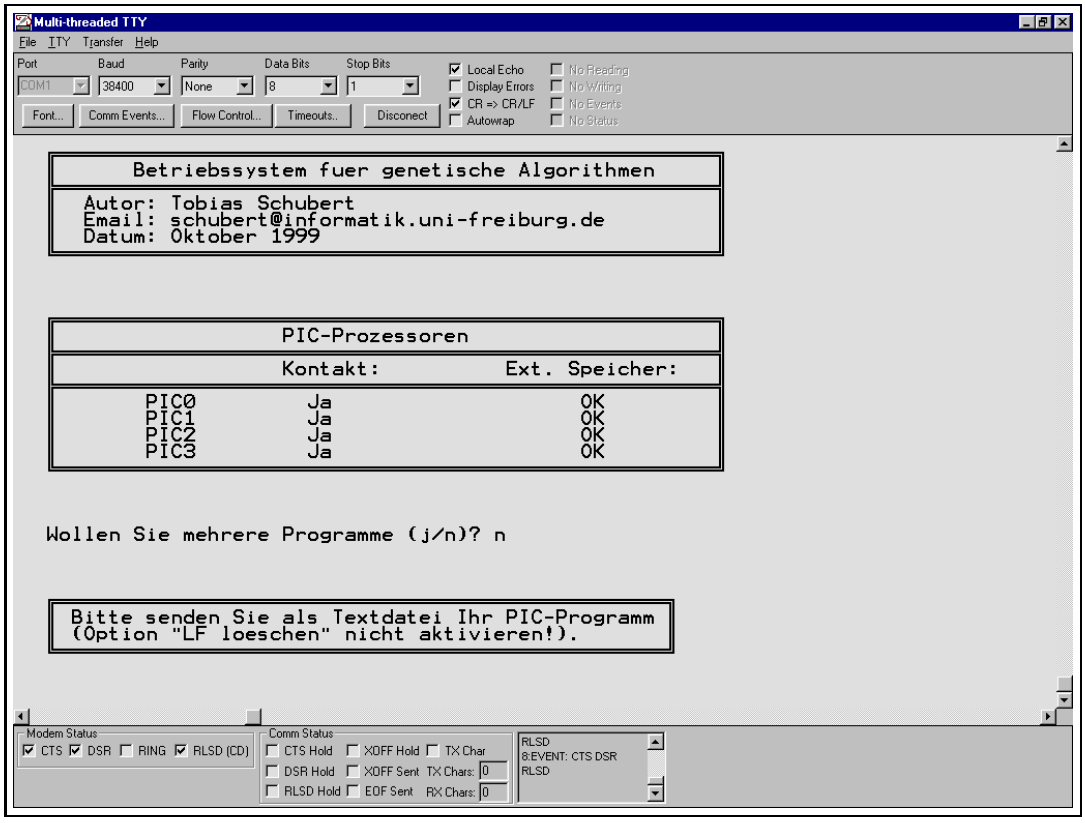


Abbildung 7.10: Übersicht über die eingesetzten Prozessorstreifen

7.4 Empfang der Aufgabenstellung

Die in `getprg.a` und `sendprg.a` realisierte Funktion des Kommunikationsprozessors bietet dem Anwender die Möglichkeit, mit Hilfe des Terminalprogrammes festzulegen, ob er allen Prozessorstreifen dieselbe Aufgabenstellung (Programmdatei) übermitteln möchte oder jedem Prozessor eine eigene Datei zuordnen will.

Dazu werden über das *Mtty*-Programm die Dateien im *INHX8M*-Format (vgl. hierzu Abschnitt 8.1, 'Struktur von PIC17C43-Programmen im Intel-Hex-Format') an den Kommunikationsprozessor gesendet, im externen RAM zwischengespeichert und im Anschluß daran an die jeweiligen Recheneinheiten weitergegeben. Als Startadresse dient hierbei $varstart + \$100$ (der Offset ist nötig, um nicht die eigenen Variablen zu überschreiben), die in `def.h` definiert werden kann. Zu beachten ist lediglich, daß ein Speicherraum von mindestens 52 kWord (112 kByte) bereitgestellt wird, weil dies dem externen Speicher der Prozessorstreifen entspricht. Das Flußdiagramm 7.11 zeigt das Vorgehen unter der Voraussetzung, daß vier PIC-Prozessoren eingesetzt werden.

Der eigentliche Dateiempfang verläuft dabei analog zu Abbildung 8.2 auf Seite 55 (dort aus Sicht des PIC17C43 beschrieben) und kann für den Motorola MC68340 mit Hilfe der im vorigen Teilabschnitt eingeführten Funktion `RECEIVERB_LOOP` elegant realisiert werden:

```
* Neues Zeichen des Anwenders abwarten:
bsr.l RECEIVERB_LOOP

* Konvertierung:
bsr.l A2SYMBOL

* Zeichen verarbeiten:
... ...
```

Die Routine `A2SYMBOL` aus `convert.a` ist notwendig, um aus der übermittelten ASCII-Nummer das entsprechende Zeichen zu generieren. Die im *INHX8M*-Format auftretenden Werte zeigt Tabelle 7.3.
Die Weitergabe der Dateien an die Prozessorstreifen geschieht interruptgestützt, d.h. nach jedem gesendeten Zeichen des Kommunikationsprozessors wird ein bestätigender Interrupt abgewartet, der signalisiert, daß der PIC17C43 das Zeichen verarbeitet hat (die nachfolgenden Befehle zeigen dies schematisch für die Weitergabe an den Prozessorstreifen '1', vgl. `sendprg.a`):

```
1 * A0 = Startadresse im externen RAM, nach jedem Zugriff inkrementieren
2 move.b  (A0)+, pic1
3
4 WAITING_SEND_PRG_PIC1
5   cmp.b #$00,d1    * Bestaetigung durch d1=$01 in irq.a
6   beq   WAITING_SEND_PRG_PIC1
7
8 * naechstes Zeichen:
9 'Sprung in Zeile 2'
```

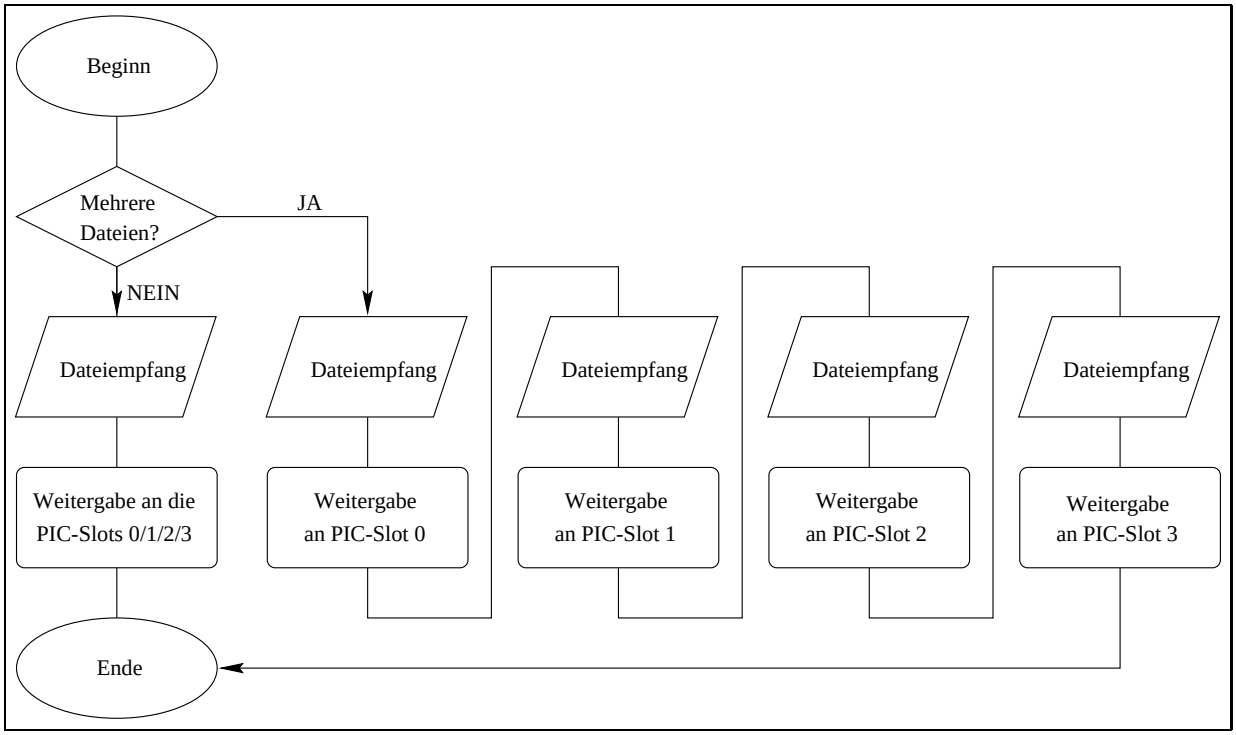


Abbildung 7.11: Flußdiagramm zur Verteilung von Anwenderaufgaben

ASCII-Nummer	Zeichen
30	'0'
31	'1'
32	'2'
33	'3'
34	'4'
35	'5'
36	'6'
37	'7'
38	'8'
39	'9'
41	'A'
42	'B'
43	'C'
44	'D'
45	'E'
46	'F'

Tabelle 7.3: Zuordnung von ASCII-Nummern zu den entsprechenden Zeichen

7.5 Die Hauptschleife

Am Anfang dieses Kapitels wurde angedeutet, daß die Hauptaufgabe des Kommunikationsprozessors in der Kontrolle des Datenaustausches zwischen den Recheneinheiten und dem Empfang der Endergebnisse liegt. Während der Abarbeitung der gestellten Probleme können die PIC-Prozessoren dazu über definierte 'Signalwörter' Anfragen an den Motorola MC68340 stellen. In der nun vorgestellten Umsetzung sind momentan lediglich die beiden folgenden Anfragen möglich (aus Sicht des PIC17C43), eine flexible Erweiterung für zukünftige Arbeiten ist aber problemlos möglich:

- $\$FE$: Datenaustausch erwünscht.
- $\$FF$: Abarbeitung beendet, Übergabe der besten Lösung.

Prinzipiell ist es möglich, jede mittels eines Interrupts eingehende Anfrage sofort zu erkennen und zu beantworten. Bedenkt man aber, daß sich je zwei Prozessorstreifen eine Interrupt-Leitung teilen, so kann der Fall eintreten, daß während der Beantwortung einer Anfrage ein anderer Prozessor 'dazwischenfunkt'. Dieses Signalwort muß dann zwischengespeichert werden, um es später zu bearbeiten. Besonders problematisch wird dieser Sachverhalt, wenn zwei Recheneinheiten zur gleichen Zeit auf derselben Leitung den Kommunikationsprozessor kontaktieren, von denen nur eine erkannt werden kann. Die Folge wären Datenverluste!

Besser ist es, den Motorola MC68340 in mehreren Phasen zu betreiben. In einem ersten Schritt werden lediglich die Anfragen der Recheneinheiten entgegengenommen. Ist von jedem PIC17C43 ein Interrupt ausgelöst worden, beginnt die zweite Phase. Jetzt werden alle Anfragen der Reihe nach beantwortet und danach mit einem 'Startsignal' an alle Einheiten

die weitere Problembearbeitung eingeleitet. Dadurch ist gewährleistet, daß immer nur ein PIC-Prozessor aktiv ist, während die anderen noch auf ihre Antwort beziehungsweise auf das Startsignal zur Fortsetzung ihrer Aufgabe warten.

Nachteilig ist bei dem gewählten Ansatz, daß die Geschwindigkeit des Gesamtsystemes immer durch den 'langsamsten' Prozessorstreifen bestimmt wird, was besonders dann zum Tragen kommt, wenn man unterschiedliche Problemparameter wie die Populationsgröße einsetzt. Ein stabiles und fehlerfreies System stand jedoch vorerst im Vordergrund.

Um den auslösenden Prozessorstreifen eingehender Interrupts identifizieren zu können, stehen dem Kommunikationsprozessor die Variablen `control0/1/2/3` zur Verfügung, die vor jedem neuen Intervall mit dem Wert $00 initialisiert werden. Bei jeder eingehenden Kontaktanfrage werden alle PLD-Latches ausgelesen und die Werte im entsprechenden Register `control0/1/2/3` gespeichert. Auf diesem Weg ist somit ein Abgleich zwischen den neuen und schon gespeicherten Datenwörtern möglich, so daß der auslösende PIC-Prozessor eindeutig identifiziert werden kann. Zudem lassen sich mit diesem Vorgehen Datenverluste vermeiden, weil unabhängig von der Interrupt-Quelle alle PLD-Bausteine der Reihe nach ausgelesen werden. Die Umsetzung ist in der Datei `irq.a` zu finden und besteht aus wenigen Befehlen (Zeilen 83-91):

```
*** Aus dem "Normal"-Modus heraus Steuersignale empfangen
COMMUNICATION
  move.b pic0,control0
  move.b pic1,control1
  move.b pic2,control2
  move.b pic3,control3
  moveq  #00,d3
  move.b #$01,d3                    * Als Bestaetigung beim Motorola
  rte
```

Ist von allen Prozessorstreifen eine Anfrage an den Motorola MC68340 signalisiert worden, müssen entweder die Endergebnisse empfangen oder ein Datenaustausch eingeleitet werden. Im ersten Fall werden die problemspezifischen Parameter wie Fitneß, Anzahl Generationen, etc. in den entsprechenden Registern aus Tabelle 7.1 gespeichert.

Im zweiten Fall wird der Austausch von Lösungen anhand der vom Anwender eingestellten Topologie vorgenommen. Zu diesem Zweck wird der Inhalt der Register `topo0/1/2/3` überprüft, das beste Chromosom des aktuellen Prozessors entgegengenommen und im Gegenzug die beste Lösung eines anderen PIC17C43 verschickt (Abbildung 7.12 zeigt das dazugehörige Flußdiagramm). Die Kodierung der Variablen `topo0/1/2/3` ist wie folgt definiert worden:

- Bit $x = 1$ in topo0: PIC-Slot 0 sendet sein bestes Individuum an Einheit x.
- Bit $x = 1$ in topo1: PIC-Slot 1 sendet sein bestes Individuum an Einheit x.
- Bit $x = 1$ in topo2: PIC-Slot 2 sendet sein bestes Individuum an Einheit x.
- Bit $x = 1$ in topo3: PIC-Slot 3 sendet sein bestes Individuum an Einheit x.

Hinter der vorgestellten Kodierung steckt eigentlich eine Matrix, bei der durch gesetzte Bits eine logische Verdrahtung zwischen zwei oder mehreren Recheneinheiten möglich wird. Per Tastendruck kann der Anwender die aktuelle Topologie jederzeit neu konfigurieren. Die Abbildungen 7.13 und 7.14 zeigen zwei Momentaufnahmen, die beim Abarbeiten der 20-Städte-Instanz des Travelling Salesman Problemes aufgenommen wurden.

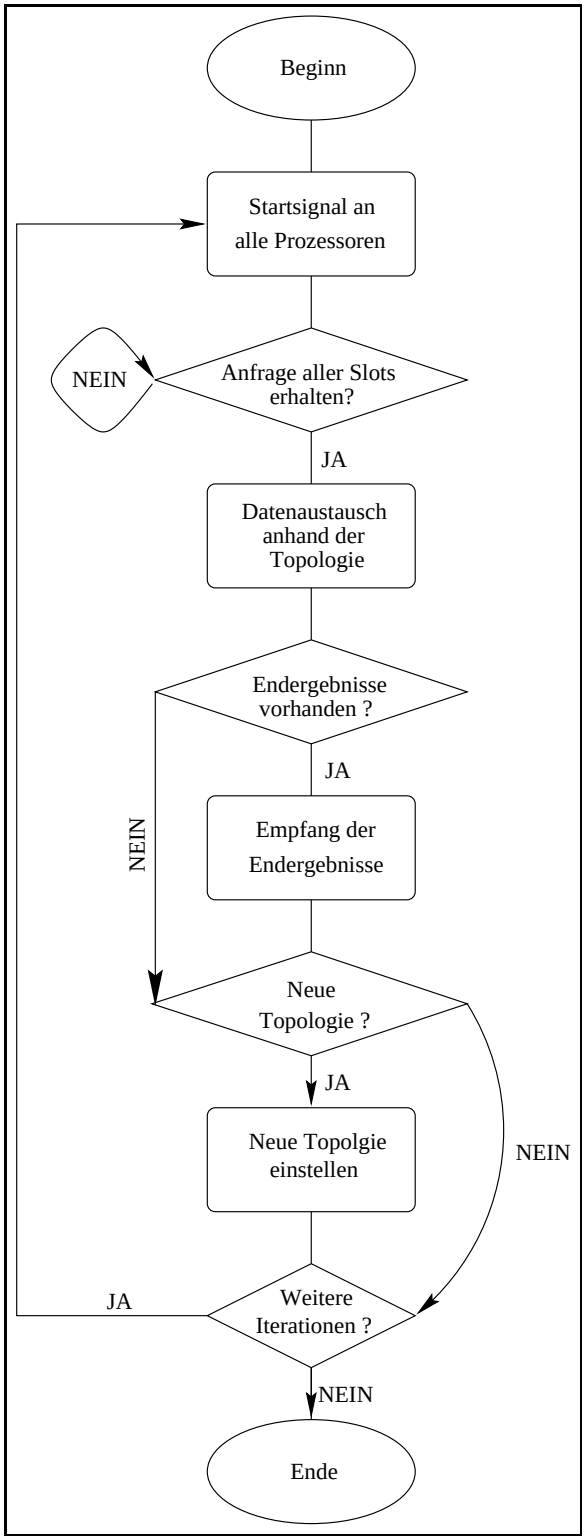


Abbildung 7.12: Das Flußdiagramm der Hauptschleife

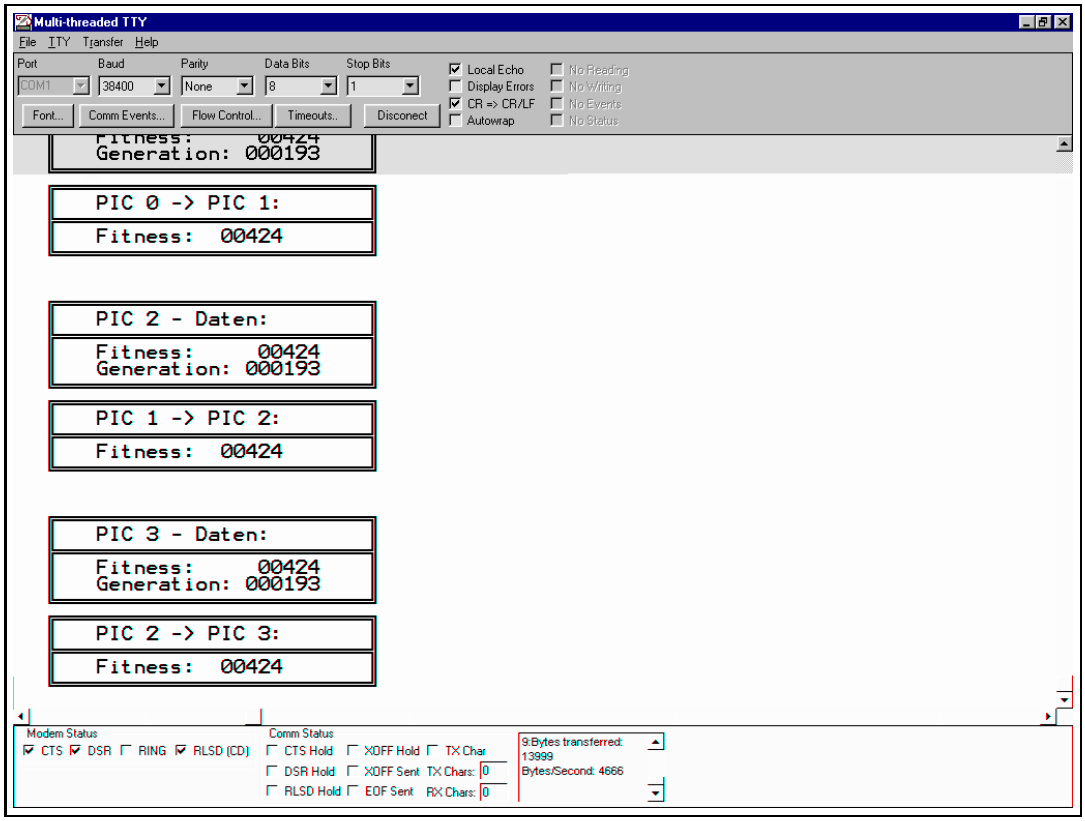


Abbildung 7.13: Datenaustausch zwischen den Recheneinheiten

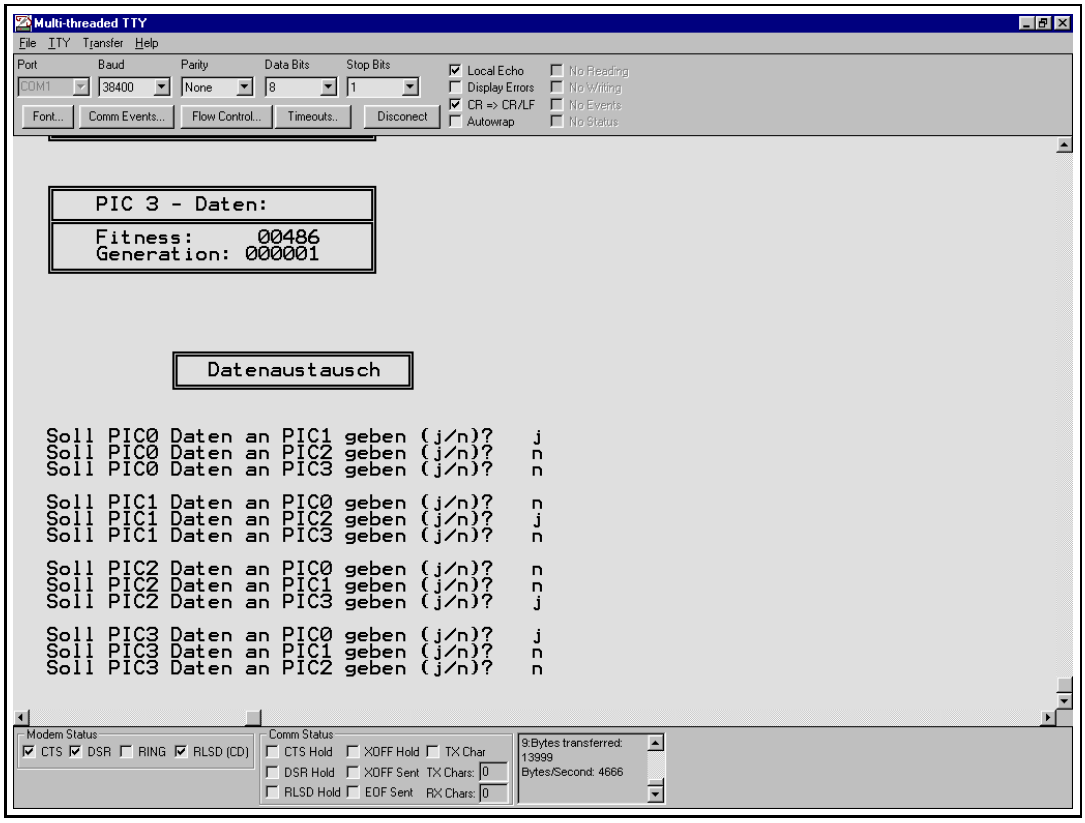


Abbildung 7.14: Konfiguration einer 'Ring'-Topologie

7.6 Präsentation der Endergebnisse

Die Ausgabe der besten Resultate der maximal vier Recheneinheiten dient dem Anwender als Kontrolle der erzielten Güte und stellt zudem ein Indiz für die benötigte Berechnungsdauer dar, die sich aus der Anzahl der durchgeführten Generationsschritte ablesen läßt. Desweiteren werden die eingestellten Problemparameter wie Größe der Population, Größe der Chromosome und Anzahl der Nachkommen pro Iteration aufgeführt, beim Travelling Salesman Problem wird zusätzlich noch die gefundene Rundreise durch die Identitätsnummern der Städte angegeben. Verwendung finden dabei die entsprechenden Variablen aus Tabelle 7.1 wie `pop_size0/1/2/3` oder `child_size0/1/2/3`. Abbildung 7.15 zeigt beispielhaft die Ausgabe für die Prozessorstreifen '2' und '3'.

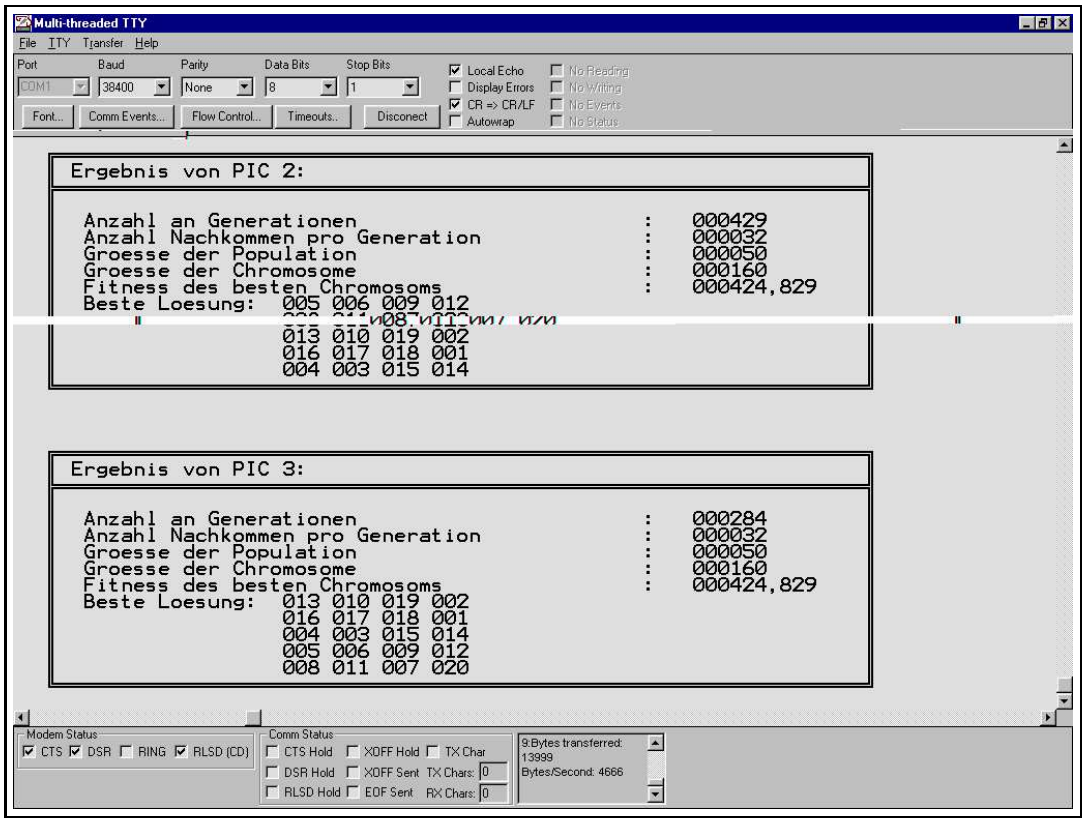


Abbildung 7.15: Das beste Ergebnis des 20-Städte-Problemes

Kapitel 8

Programmierung des Microchip PIC17C43

Die in den Zielsetzungen festgelegten drei Hauptpunkte des PIC17C43-Grundprogrammes sind im Flußdiagramm 8.1 dargestellt.

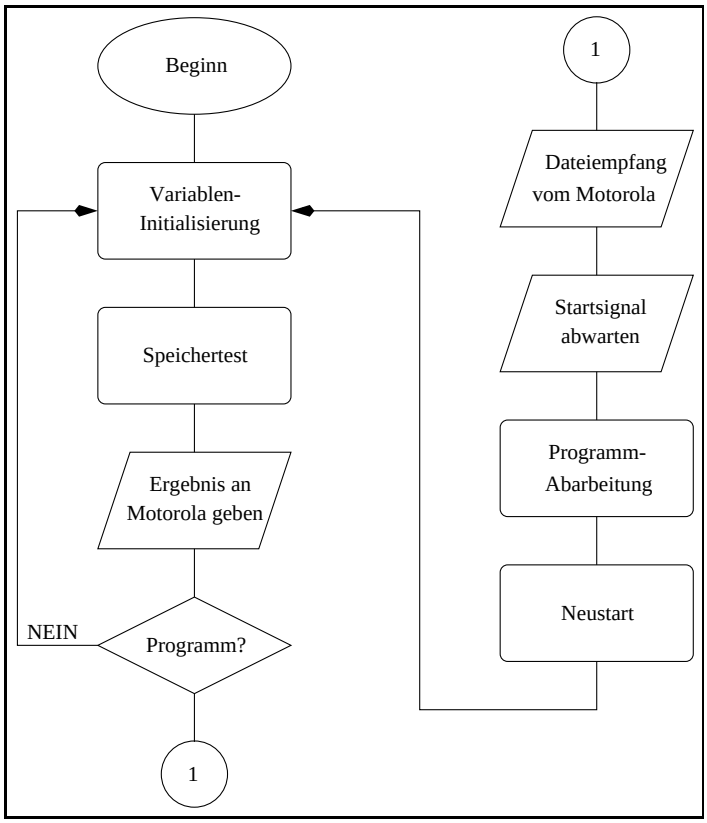


Abbildung 8.1: Flußdiagramm: PIC17C43-Grundprogramm

Es sind dies die Blöcke *Speichertest*, *Dateiempfang vom Motorola MC68340* und *Programm-Abarbeitung* (der gestellten Aufgabe), die in einer Endlosschleife durchlaufen werden. Der Analyse der Struktur eines kompilierten Assemblerprogrammes, wie es dem Prozessorstreifen übergeben wird und abgearbeitet werden soll, und den daraus resultierenden Folgen beim Dateiempfang ist der nächste Teilabschnitt gewidmet. Zunächst stehen jedoch die globalen Einstellungen, wie sie in der Hauptdatei `tonux.asm` vorgenommen werden, im

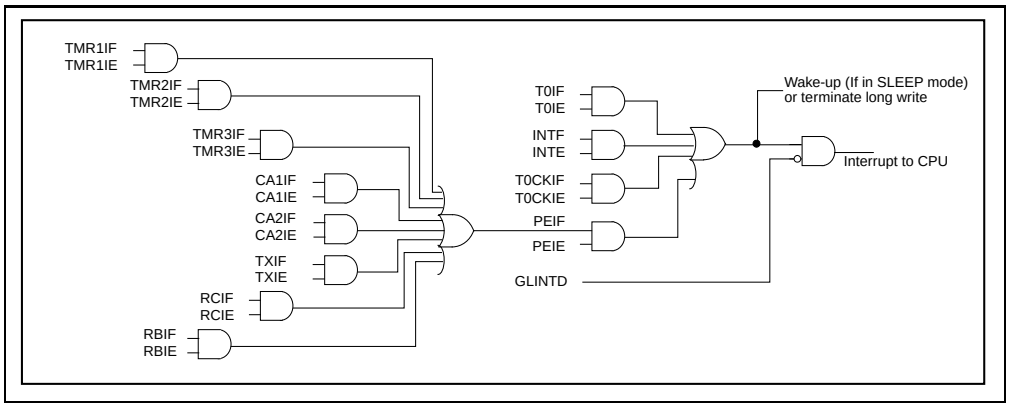
Vordergrund. Das komplette Listing ist im Anhang C aufgeführt. Als einführende Literatur zum Programmieren des PIC17C43 und zum Umgang mit der Software *Mplab* stellen die Verweise [9]-[12] eine gute Ausgangsbasis dar. Als Programmiersprache wird Assembler verwendet, um sprachliche Kompatibilität zur in [8] vorgestellten Funktionsbibliothek zu wahren und die beschränkte Größe von 4 kByte des PIC17C43 EPROMs nicht zu überschreiten.

Als erstes ist der PIC17C43 korrekt an die vorliegende Hardware anzupassen. Dazu müssen der sogenannte *Extended Microcontroller Mode* und der *XT-Timer*-Modus aktiviert werden. Ersteres stellt die einzigst wählbare Variante dar, in welcher der Anwender sowohl das interne EPROM wie auch den externen RAM-Baustein mit den Adressen $AD[0:15]$ ansprechen kann. Die Einstellung *XT-Timer* wählt als Takterzeugung eine externe Quarzkristallquelle (die Frequenz von 16.78 MHz wird hierbei vom Motorola MC68340 für alle vier Recheneinheiten bereitgestellt). Die Konfiguration kann zu Beginn der Datei festgelegt oder wie in dieser Arbeit erst beim 'Brennen' mit der *Microchip Mplab Promate*-Software vorgenommen werden.

Definiert wurde, daß zu lösende Problemstellungen als Assemblerdatei im Intel-Hex-Format (*INHX8M*) ab der externen Speicherzelle $2000 beginnen. Diese Datei soll im 56 kWord großen, 'oberen' RAM-Bereich abgelegt werden. Die Tatsache, daß die Programme auch von dort ausgeführt werden, läßt die Benutzung der 'unteren' RAM-Bank für Anwendungen nicht zu, der effektiv vorhandene Speicher sinkt somit von 128 kByte auf 112 kByte. Um den Speicherbereich auszuwählen, muß gemäß Abschnitt 5 der Pin $RB7 = MAP0/1$ auf eine logische Eins gelegt werden, was in den Zeilen 99 bis 101 geschieht:

```
CLRF   DDRB, 1
MOVLW  B'11111111'
MOVWF  PORTB                    ; MAP0/1=1 --> Speicher: $2000 - $FFFF
```

Um dem späteren Nutzer des Multiprozessorsystems eine komfortable Schnittstelle zu bieten, muß ihm die Kontrolle über alle Interruptvektoren ermöglicht werden, diese stellen neben den jederzeit neu konfigurierbaren I/O-Ports die wichtigste Schnittstelle zur Außenwelt dar. Insbesondere bei Problemen, die mit der Technik der genetischen Algorithmen programmiert wurden, ist der Kontakt zu anderen Prozessoren zwecks Datenaustausch erforderlich. Die nachfolgende Abbildung zeigt das schematische Blockdiagramm der 11 möglichen Auslöserquellen.



Kapitel 5 und Abbildung 5.1 auf Seite 20 zeigen, daß das dort eingeführte Signal *pint* (*Pic Interrupt*), welches der Motorola MC68340 zum Übermitteln von 8 Bit Datenworten nutzt, mit dem Anschluß $RA0/INT$ auf Seiten des PIC17C43 verbunden ist. Zur Konfiguration genügt es, alle Interrupt-Quellen als gültig zu deklarieren (Zeile 104, `BCF CPUSTA, 4`) und den I/O-Port $RA0$ als externen Interrupt höchster Priorität zu definieren (Zeile 102, `BSF INTSTA, 0`). Die Bezeichnungen $CPUSTA$ und $INTSTA$ stehen hierbei für *CPU Status Register* beziehungsweise *Interrupt Status Register*. Momentan wird nur diese eine Leitung genutzt, d.h. mehrere, gleichzeitig aktive Interruptsignale sind deshalb nicht möglich. Prinzipiell wird aber aufgrund der höchsten Priorität immer der Kontakt zum Kommunikationsprozessor vorrangig behandelt.

Problematisch ist, daß die Sprungadressen von der CPU automatisch berechnet werden und zwischen \$0008 (*int_vec*) und \$0020 (*peri_vec*) liegen. Dies stellt den internen EPROM-Bereich dar. Anwenderprogramme hätten nie Zugriff auf Interrupts und ein Austausch von Daten zwischen den Prozessorstreifen wäre somit nicht möglich. Eine Lösung bietet sich an, wenn man neben der Forderung zukünftige Aufgabenstellungen ab der Adresse \$2000 beginnen zu lassen, zusätzlich vereinbart, daß bei eingehenden Interrupts die Sprungadressen um einen Offset von \$2000 verschoben werden. Beispielsweise würde ein aktives *peri_vec*-Signal nicht mehr einen Sprung an die Adresse \$0020 sondern an \$2020 hervorrufen (für die anderen Interruptvektoren analog). Die entsprechenden Zeilen der Dateien `tonux.asm` in der linken Hälfte und `tsp.asm` (Hauptdatei des in Kapitel 9 implementierten *Travelling Salesman Problemes*) zeigen die gemeinsame Verzahnung, wobei momentan natürlich nur zu dem weiter oben zugelassenen *int_vec* eine Interrupt-Routine existiert:

```
         ORG    0000
reset    GOTO START

         ORG    0008
int_vec  GOTO   READ_FROM_MOTOROLA

         ORG    0010
rtcc_vec MOVLW  H'20'
         MOVWF  PCLATH
         LCALL  H'10'
         RETFIE

         ORG    0018
rt_vec   MOVLW  H'20'
         MOVWF  PCLATH
         LCALL  H'18'
         RETFIE

         ORG    0020
peri_vec MOVLW  H'20'
         MOVWF  PCLATH
         LCALL  H'20'
         RETFIE
```

```
         org 2000
             goto    Main_Loop

         org 2008
int_vec      movlw   0x11
             movwf   tblptrh
             clrf    tblptrl, 1
             tablrd  1, 0, temp2
             tlrd    1, temp2
             incf    temp1, 1
             return

         org 2010
rtcc_vec     return

         org 2018
rt_vec       return

         org 2020
peri_vec     return
```

Bei obiger Idee stellt diese Leitung eine Ausnahme dar, weil sie bereits zu Beginn des Betriebssystemes zum Dateiempfang und für alle Steuersignale vom Kommunikationsprozessor benötigt wird. Aus diesem Anlaß ist die Variable `Mode` eingeführt worden, deren Wert in der Interrupt-Routine *READ_FROM_MOTOROLA* entscheidet, ob der eingehende Interrupt während des Ablaufes des Betriebssystemes oder während der Abarbeitung der Anwenderaufgabe eingetreten ist. Tabelle 8.1 zeigt die drei möglichen Belegungen:

Mode	**Funktion**
0	Normalmodus
1	Neue Daten vom Kommunikationsprozessor
2	Anwenderaufgabe wird derzeit bearbeitet, Sprungadresse in $2008 umwandeln

Tabelle 8.1: Die Variable `Mode` und ihre Belegungen

Die Bedeutung wird mit dem nachfolgenden Ausschnitt aus `tonux.asm` deutlich (Zeilen 166-194):

```
;*********************************************************************
;***
;*** Name....: READ_FROM_MOTOROLA
;*** Funktion: Daten vom Motorola aus dem PLD auslesen.
;***           Besonders wichtig auch fuer den Datenaustausch aus
;***           der PIC-Anwendung heraus (MODE = 2).
;*** Register: TBLPTRH, TBLPTRL um mit $1100 das PLD zu aktivieren,
;***           MOT_TO_PIC enthaelt das gelesene Byte.
;***
READ_FROM_MOTOROLA
        MOVLW  H'00'
        CPFSEQ MODE
        GOTO   $+9

        MOVLW  0x11
        MOVWF  TBLPTRH
        CLRF   TBLPTRL, 1              ; "Table Pointer" = $1100
        TABLRD 1, 0, WREG
        TLRD   1, WREG
        MOVWF  MOT_TO_PIC
        INCF   MODE, 1                 ; Gilt als Bestaetigung
        RETFIE

        MOVLW  H'20'                   ; In Anwender-Routine verzweigen
        MOVWF  PCLATH
        LCALL  H'08'
        RETFIE
;***
;*********************************************************************
```

Die ersten drei Befehle springen bei `Mode`$\neq 0$ (aufgrund der Pogrammstruktur bedeutet das `Mode`=2) an die Adresse $2008. Ansonsten wird das gesendete Byte im dafür vorgesehenen Register *MOT_TO_PIC* gespeichert (wie in Abschnitt 5 erläutert mit einem Lesezugriff auf die Adresse $1100), und die Routine regulär mit *RETFIE* beendet.

Wiederholend läßt sich zusammenfassen: Das Betriebssystem, welches in das 4 kByte große EPROM eingespeichert wird, wurde so flexibel und 'unauffällig' wie möglich gestaltet, damit es sich für den späteren Benutzer, der lediglich Optimierungsaufgaben parallel von den Prozessorstreifen lösen lassen möchte, wie ein Teil der Hardware des PIC17C43 darstellt. Dazu muß 'nur' ein Adreß-Offset von $2000 zu Beginn des Assembler-Programmes eingerichtet und die Adresse $0041 (= `Mode`) nicht überschrieben werden.

8.1 Struktur von PIC17C43-Programmen im Intel-Hex-Format

Der Dateiempfang wird in Zeile 128 in `tonux.asm` mit dem Aufruf `CALL GET_PRG_FROM_MOTOROLA` gestartet, und das vom Kommunikationsprozessor gesendete Programm wird an den innerhalb der Datei kodierten Adressen gespeichert. Am einfachsten ist das Dateiformat und der damit einhergehende Ablauf an einem kleinen Beispiel zu verdeutlichen. Die folgende Abbildung zeigt im oberen Teil das Beispielprogramm und darunter die dazugehörige Datei im *INHX8M*-Format.

```
INCLUDE "p17c43.inc"
LIST p=17C43

reg1 equ 0x70;
reg2 equ 0x80;
reg3 equ 0x90;

ORG     2000
  goto $+1
  movlw B'00000011'
  movwf reg1
  movlw B'00000010'
  movwf reg2
  movfp reg1, WREG
  clrf reg3
  addwf reg3,1
  movfp reg2, WREG
  addwf reg3,1
  nop
end




:1040000001C003B0700102B08001706A9029900F66
:06401000806A900F000021
:00000001FF
```

Die Art der Kodierung beinhaltet eine 8-Bit Datei mit einer Kombination aus niederwertigen und höherwertigen Befehlsanweisungen im hexadezimalen Zahlensystem. Jede Zeile dieses Formates hat dabei die folgende Gestalt:

$$: BB\ AAAA\ TT\ HHHH\ ...\ HHH\ CC$$

mit

- BB ist ein Zähler, der die Anzahl an kodierten Befehlen in dieser Zeile angibt. Weil jeder Befehl aus einer 16-Bit Kodierung besteht, d.h. aus einer hexadezimalen Angabe aus dem Bereich $\$0000, ..., \$FFFF$, ist $\frac{BB}{2}$ die tatsächliche Anzahl an kodierten Anweisungen in dieser Zeile.
- $AAAA$ gibt die Startadresse an, ab welcher der Datenblock der restlichen Zeile abgelegt werden soll.
- TT gibt an, ob die vorliegende Datei mit der aktuellen Zeile endet oder nicht:
  00 = weitere Befehlszeilen,
  01 = Programm endet nach dieser Zeile.

- CC ist die Kontrollsumme, um die korrekte Datenübertragung überprüfen zu können. Diese ist das 2er-Komplement der Summe aller vorherigen Werte und ist einfach zu berechnen, indem man alle Bytes der Zeile aufsummiert, das Komplement bildet und dann inkrementiert.
- $HHHH$ ... $HHHH$ ist die Kodierung der einzelnen Befehle, wobei eine Anweisung $H_1H_2H_3H_4$ aus den höherwertigen Bytes H_3H_4 und den niederwertigen Bytes H_1H_2 besteht.

Für das obige Beispielprogramm ergibt das folgende Dekodierung:

: $\underbrace{10}$ $\underbrace{4000}$ $\underbrace{00}$ $\underbrace{01C0}$ $\underbrace{03B0}$ $\underbrace{7001}$ $\underbrace{02B0}$ $\underbrace{8001}$ $\underbrace{706A}$ $\underbrace{9029}$ $\underbrace{900F}$ $\underbrace{66}$
8 Anweisungen, Adresse 4000, Kein Ende, B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, SUM_1

: $\underbrace{06}$ $\underbrace{4010}$ $\underbrace{00}$ $\underbrace{806A}$ $\underbrace{900F}$ $\underbrace{0000}$ $\underbrace{21}$
3 Anweisungen, Adresse 4010, Kein Ende, B_9, B_{10}, B_{11}, SUM_2

: $\underbrace{00}$ $\underbrace{0000}$ $\underbrace{01}$ $\underbrace{FF}$
0 Anweisungen, Adresse 0000, Datei-Ende, SUM_3

mit

$B_1 = \$C001 = (1100\ 0000\ 0000\ 0001)_2 =$ goto $\$ + 1$
$B_2 = \$B003 = (1011\ 0000\ 0000\ 0011)_2 =$ movlw B '00000011'
$B_3 = \$0170 = (0000\ 0001\ 0111\ 0000)_2 =$ movwf 0x70
$B_4 = \$B002 = (1011\ 0000\ 0000\ 0010)_2 =$ movlw B '00000010'
$B_5 = \$0180 = (0000\ 0001\ 1000\ 0000)_2 =$ movwf 0x80
$B_6 = \$6A70 = (0110\ 1010\ 0111\ 0000)_2 =$ movfp 0x70, 0x10
$B_7 = \$2990 = (0010\ 1001\ 1001\ 0000)_2 =$ clrf 0x90
$B_8 = \$0F90 = (0000\ 1111\ 1001\ 0000)_2 =$ addwf 0x90, 1
$B_9 = \$6A80 = (0110\ 1010\ 1000\ 0000)_2 =$ movfp 0x80, 0x10
$B_{10} = \$0F90 = (0000\ 1111\ 1001\ 0000)_2 =$ addwf 0x90, 1
$B_{11} = \$0000 = (0000\ 0000\ 0000\ 0000)_2 =$ nop

Man überprüft leicht, daß auch die drei Kontrollsummen $SUM_1 \ldots SUM_3$ den oben Angegebenen entsprechen. Für die Übergabe und Dekodierung derartiger Dateien dient das auf Seite 55 dargelegte Flußdiagramm als Vorlage.

Anhand der Datei `getprg.asm` läßt sich zudem sehr gut veranschaulichen, wie ein gegenseitiger Austausch von Daten mit dem Kommunikationsprozessor aus Sicht des PIC17C43 vonstatten geht.
Mit den zur Verfügung stehenden Routinen `WRITE_BYTE_TO_MOTOROLA` und `WAITING_FOR_MOTOROLA` (beide aus `tonux.asm`) genügt es, die zu sendenden Daten in das dafür vorgesehene Register *PIC_TO_MOT* zu verschieben, den PIC-Betriebsmodus *Mode* auf Null zu setzen und hintereinander beide Routinen aufzurufen. Das folgende Beispiel zeigt den Programmablauf. `Mode`=1 signalisiert dabei eine bestätigende Nachricht des Motorola MC68340, die Routine `WAITING_FOR_MOTOROLA` wird verlassen und die Daten können aus MOT_TO_PIC ausgelesen werden:

```
MOVLW  B'00000001'
MOVWF  PIC_TO_MOT
CALL   WRITE_BYTE_TO_MOTOROLA          ; 'Handshake'-Signal
CLRF   MODE, 1
CALL   WAITING_FOR_MOTOROLA
```

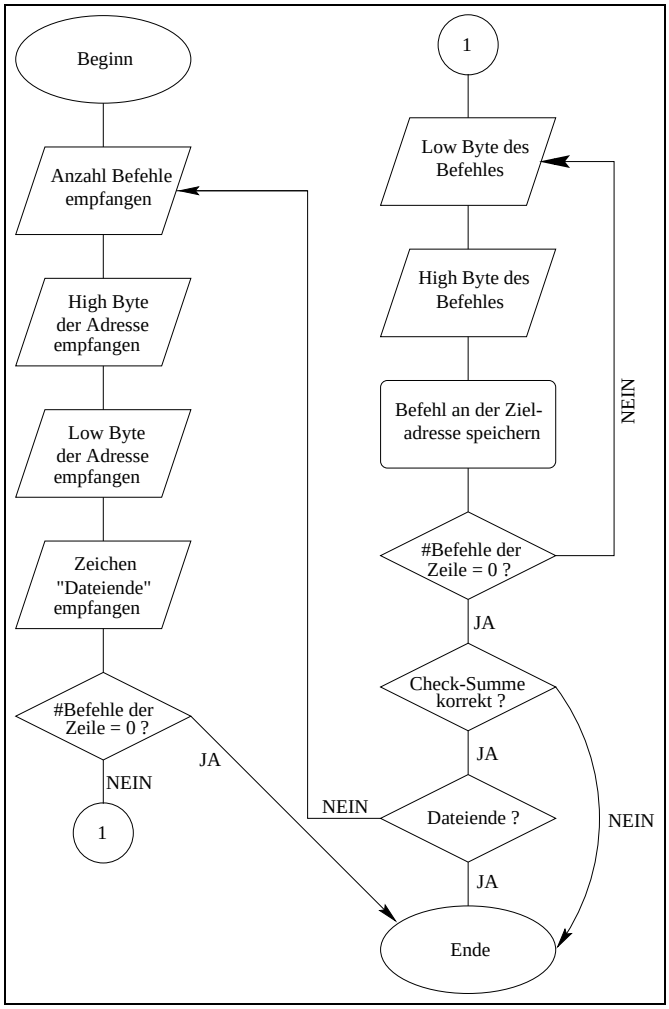
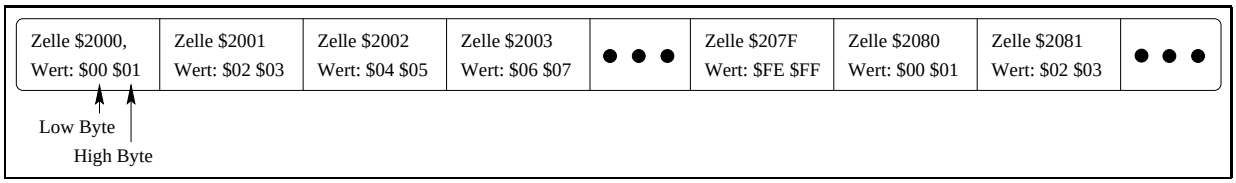


Abbildung 8.2: Flußdiagramm: Dateiempfang vom Kommunikationsprozessor

8.2 Der Speichertest

Der Test (Aufruf mit `CALL MEM_TEST_START` in Zeile 96 in `tonux.asm`) ist ein Teil der gemeinsamen Initialisierungsphase mit dem Kommunikationsprozessor: Nur wenn der Prozessorstreifen in einem festgelegten Zeitfenster mit einem positiven Ergebnis antwortet, setzt der Motorola den Prozessor als vorhanden und nicht defekt voraus (nicht alle vier Steckplätze müssen genutzt werden) und übergibt diesem später Anwendungen.

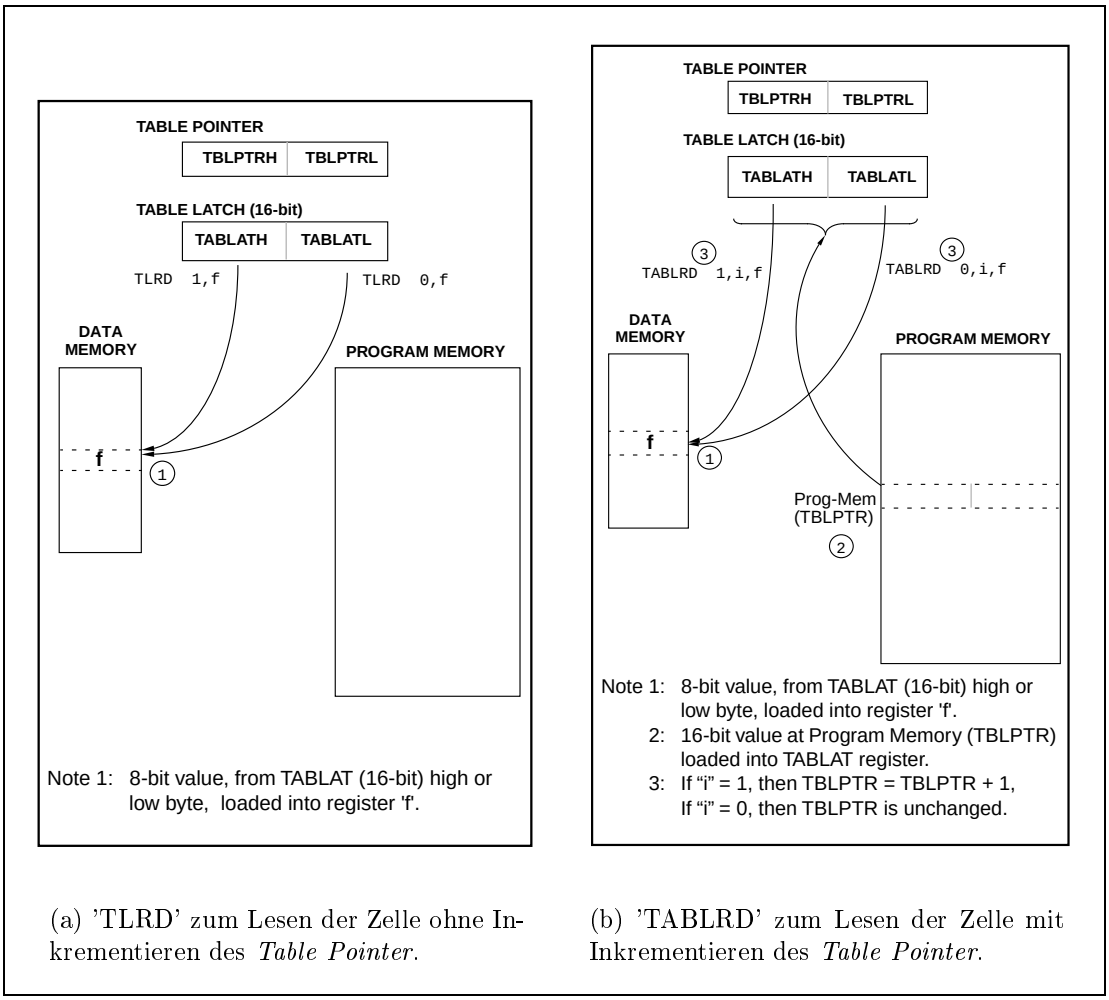
Dabei wird nach einem in diesem Bereich üblichen Schema vorgegangen: Sowohl der 56 kWord 'große' wie auch der 8 kWord 'kleine' externe Speicher werden vollständig mit aufsteigenden Werten gefüllt, die 16 Bit Zellen ausgelesen und anschließend auf ihre Korrektheit hin überprüft.



Ein fehlerfreies Ansprechen des Speichers wird dem Motorola MC68340 mit dem Wert \$00 der Variablen OK angezeigt. Die Belegung mit \$01 entspricht der Tatsache, daß sich beim Schreiben und/oder Lesen von RAM-Zellen Inkonsistenzen ergeben haben. Die Anzahl an falschen Werten ist dann den beiden Variablen $ERROR\_LO$ und $ERROR\_HI$ zu entnehmen, die allerdings nicht weitergegeben werden.

Die Schreib- und Lesezugriffe vom PIC-Prozessor auf Peripherie-Module werden in den Abbildungen 8.3 und 8.4 wiedergegeben, weil sie an vielfältigen Stellen genutzt werden und aufgrund ihres mehrphasigen Ablaufes (zwei oder drei Befehlszyklen) untypisch im Vergleich zu anderen Befehlen sind.

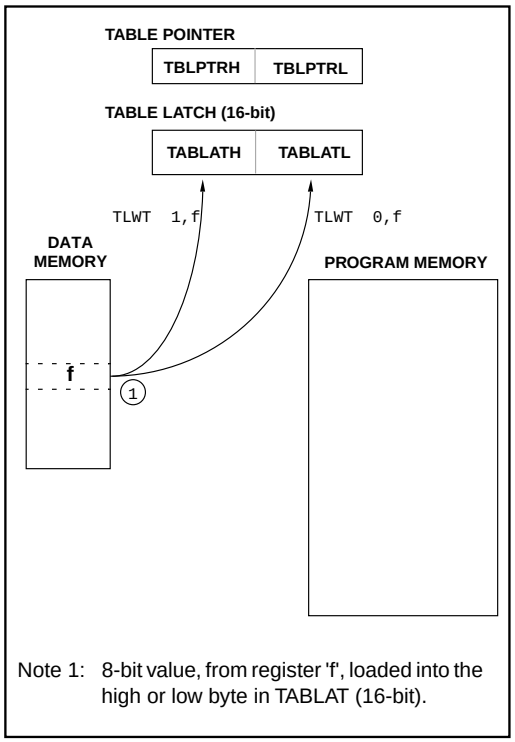
Bei einem Lesezugriff muß zuerst der *Table Pointer*, der die Zieladresse spezifiziert, aktualisiert werden, bevor die momentan aktuellen zwei Datenbyte des Zwischenspeichers (*Table Latch*) in die angegebenen Zielregister verschoben werden und selbiges mit dem gelesenen Datenwort der externen Zelle gefüllt wird (der Schreibzugriff verhält sich analog). Dieser Sachverhalt erfordert folglich beim ersten Auslesen einer externen RAM-Zelle zwei Lesezugriffe auf die gleiche Speicheradresse, um die tatsächlichen und nicht die noch im *Table Latch* vorhandenen Werte zu erhalten.



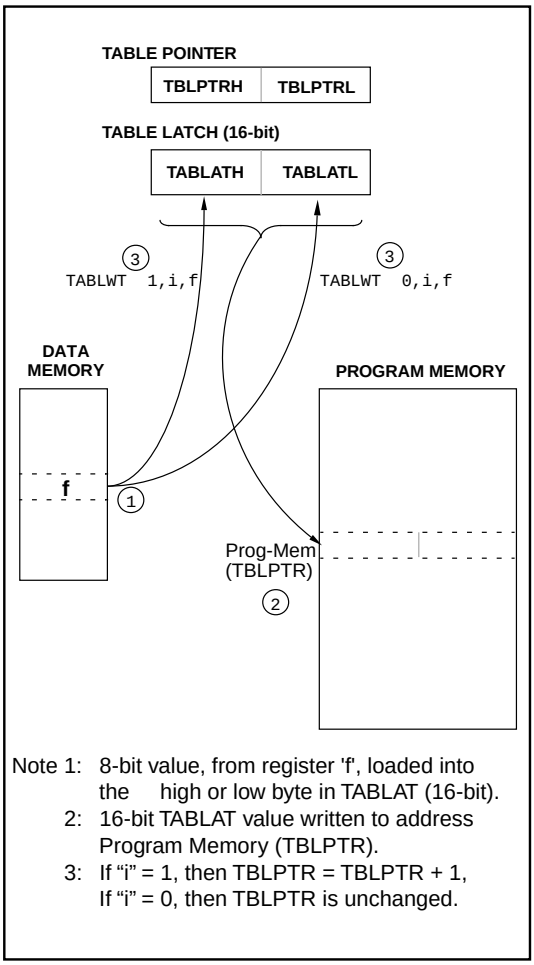
(a) 'TLRD' zum Lesen der Zelle ohne Inkrementieren des *Table Pointer*.

(b) 'TABLRD' zum Lesen der Zelle mit Inkrementieren des *Table Pointer*.

Abbildung 8.3: Lesezugriffe auf externe Module



(a) 'TLWT' zum Schreiben der Zelle ohne Inkrementierung.



(b) 'TABLWT' zum Schreiben der Zelle mit Inkrementierung.

Abbildung 8.4: Schreibzugriffe auf externe Module

Kapitel 9

Das TSP-Problem als genetische Anwendung

Das *Travelling Salesman Problem* (im weiteren Verlauf der Einfachheit halber nur noch als TSP-Problem oder kurz TSP bezeichnet) stellt eines der vielen klassischen Optimierungsprobleme dar, wie sie in Vorlesungen der *Theoretischen Informatik* im Grundstudium untersucht werden. Die folgenden Definitionen und Varianten der Aufgabenstellungen orientieren sich eng an den Notationen aus [21] und [23].

Definition 9.1. *TRAVELLING SALESMAN PROBLEM TSP*: Gegeben sind n Orte und die Kosten $c(i,j) \in \mathbf{N}$, um von Stadt i nach Stadt j zu reisen. Eine Rundreise (eine Tour) ist durch eine Permutation π auf $\{1,\ldots,n\}$ gegeben, ihre Kosten betragen $c(\pi(1),\pi(2)) + c(\pi(2),\pi(3)) + \ldots + c(\pi(n-1),\pi(n)) + c(\pi(n),\pi(1))$.
Gesucht ist also eine Route, von einem beliebigen Startort ausgehend, die alle anderen $(n-1)$ Städte genau einmal besucht und wieder am Ausgangsort endet. Abbildung 9.1 zeigt ein 5-Städte-Problem mit zwei denkbaren Routen.

- Variante 1: Gibt es zu einer vorgegebenen Distanz D eine Rundreise, deren Gesamtlänge kürzer als D ist?
- Variante 2: Berechne die Kosten der billigsten Rundreise.
- Variante 3: Berechne die billigste Rundreise.

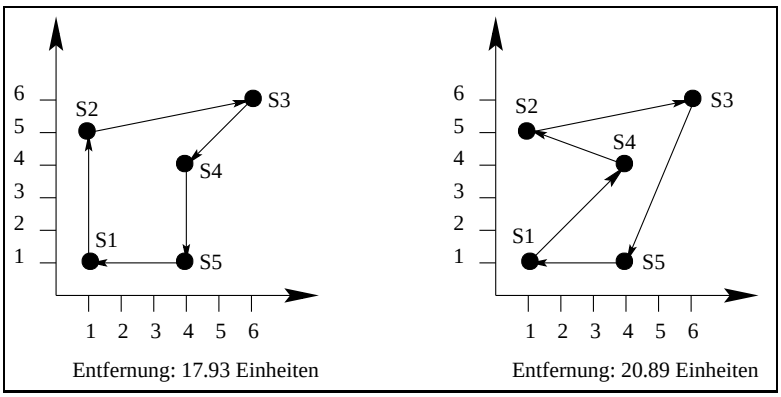


Abbildung 9.1: Zwei Routen eines 5-Städte-Problemes

Zusätzlich zu den obigen drei Varianten sind noch unzählige verschiedene Kostenmaße möglich: beim **symmetrischen** TSP-Problem gilt für alle Paare von Orten i und j, daß $c(i,j) = c(j,i)$, was beim **asymmetrischen** TSP nicht gelten muß. Desweiteren sind Instanzen denkbar, bei denen nicht alle Paare von Städten eine gemeinsame Kante besitzen, d.h. diese Orte sind nicht auf direktem Weg nacheinander zu besuchen (oder zumindest nicht in beide Richtungen).
Eine Verallgemeinerung stellt hierzu das **Sequential Ordering Problem** (SOP) dar, bei dem es zu den Orten, definiert durch ihre Koordinaten, noch weitere Nebenbedingungen gibt, die bei der Lösung eingehalten werden müssen. Ein Beispiel ist eine (partielle) Ordnung auf den Städten, was zur Folge hat, daß gewisse Ortschaften erst nach der Durchreise durch andere besucht werden dürfen.

Die bei dieser Diplomarbeit gewählte Problemstellung ist wie folgt zu charakterisieren:

- Gegeben:
  - n Orte mit $n \in \{4, 8, 12, 16, \ldots, 252\}$ mit paarweise verschiedenen X- und Y-Koordinaten. Hierbei ist die Beschränkung auf maximal 252 Städte auf die spätere Umsetzung zurückzuführen, den gegebenen Punkten wird eine eindeutige 8 Bit große Nummer (Identität) zugeordnet.
  - Zu jeder Ortschaft i sind die Koordinaten x_i und y_i mit $x_i, y_i \in [0, \ldots, 180]$ gegeben. Ein 'nur' 180x180 Einheiten großes Koordinatensystem hat bei der Güte-Bestimmung einer Rundreise den Implementationsvorteil, daß 16 Bit ausreichen.
  - Die Entfernung zwischen zwei Orten i und j ist definiert durch die euklidische Distanz ('Luftlinie'): $\forall i, j$ mit $x_i, x_j, y_i, y_j \in \mathbf{N}$ gilt:

$$c(i,j) \;=\; \sqrt{(x_i - x_j)^2 \;+\; (y_i - y_j)^2}$$

- Gesucht: Die Gesamtlänge und die Route der kürzesten Rundreise, beziehungsweise die beste gefundene Lösung nach Erreichen eines noch zu definierenden Abbruchkriteriums (siehe dazu 9.2).

Die Veranschaulichung durch einen Handelsreisenden, der in jeder ihm bekannten Stadt seine Güter verkaufen und zur Gewinnmaximierung so wenig Reisekosten aufbringen möchte wie möglich, zeigt nicht die Praxisrelevanz dieses Optimierungsproblemes.

Verwandte Themengebiete sind zum Beispiel die Fertigung von Masken für die Leiterplattenproduktion und die anschließende Herstellung der Leiterplatten. Im ersten Fall besteht die Aufgabe darin, eine mittels CAD-Software erstellte Maske mit möglichst wenigen Bewegungen der Stifte auf einem Plotter auszudrucken. Nach der Produktion der Leiterplatte müssen dann noch die Bohrungen für die Bauteile und die meist unumgänglich gewordenen Durchkontaktierungen gefertigt werden. Auch hier soll sich der eingesetzte Bohrkopf möglichst wenig über die Platine bewegen, um den Durchsatz (Anzahl produzierter Einheiten pro Stunde) und damit den Gewinn zu maximieren.

Um ein Gefühl für den enormen Berechnungsaufwand selbst kleiner Probleminstanzen zu geben, zeigt folgende Tabelle die Anzahl 'echt' unterschiedlicher Routen zu einer vorgegebenen Menge von Orten (bei n Städten gibt es $\frac{1}{2} \cdot (n-1)!$ viele verschiedene Rundrei-

semöglichkeiten):

Anzahl Städte	Anzahl unterschiedlicher Routen
4	3
8	2520
12	19958400
16	$6.54 \cdot 10^{11}$
20	$6.08 \cdot 10^{16}$
24	$1.29 \cdot 10^{22}$
28	$5.44 \cdot 10^{27}$
32	$4.11 \cdot 10^{33}$
36	$5.16 \cdot 10^{39}$
40	$1.01 \cdot 10^{46}$
60	$6.93 \cdot 10^{79}$
80	$> 10^{100}$

Tabelle 9.1: Die Anzahl unterschiedlicher Routen bei verschiedenen Probleminstanzen

Geht man von der Annahme aus, daß ein fiktiver Algorithmus pro Sekunde eine Route in ihren Kosten beurteilen könnte, so würde für $n = 100$ bereits das Alter des Universums nicht ausreichen, um die bestmöglichste Lösung bestimmen zu können. Formalisiert wird dieser Sachverhalt durch die Tatsache, daß das Travelling Salesman Problem *NP-vollständig* ist, d.h. unter der Annahme $P \neq NP$ existiert kein deterministischer Algorithmus, der zu jeder vorgegebenen Instanz die kürzeste Rundreise in polynomieller Zeit ermittelt. Ein Beweis hierzu ist in [21] und auch in [23] zu finden.

9.1 Der klassische genetische Grundalgorithmus

Obige Tabelle arbeitet ganz deutlich heraus, daß man auf die Forderung nach der bestmöglichsten Lösung verzichten muß, um zu realisierbaren Verfahren für 'schwierige' Optimierungsprobleme zu gelangen. Dennoch sind ausgeklügelte Mechanismen nötig, damit 'sehr gute' Näherungen erreicht werden.

Eine vielversprechende Technik ist die Umsetzung der biologischen Evolutionstheorie: der genetische Algorithmus. Die Informationen über Struktur und Fähigkeiten eines biologischen Individuums sind bekanntlich in den Chromosomen (Erbgut) kodiert, diese wiederum sind Bestandteil jeder Zelle. Ein Chromosom besteht chemisch vor allem aus einer charakteristischen DNA-Kette und kann aus Sicht der Informatik als ein String über einem Alphabet mit einigen Grundmolekülen angesehen werden. Bei der algorithmischen Nachbildung ist ein Chromosom im allgemeinen binär kodiert, das heißt mit den Zeichen '0' und '1' aufgebaut.

Kreuzungsversuche im 19. Jahrhundert (*Darwin'sche Selektionstheorie*) zeigten schließlich, daß Mutation einzelner Zellen und die Verschmelzung von DNA-Ketten zweier Individuen zu einem neuen Chromosom für die Weiterentwicklung einer Gattung verantwortlich sind. Von den Individuen in ihrer Gesamtheit (Population) 'überleben' dann allerdings immer

nur die an die Umweltbedingungen am besten Angepaßten.

Das für die Informatik daraus abgeleitete Modell baut folglich auf den erläuterten Voraussetzungen auf:

- Zu einer gegebenen Problemstellung gibt es eine Population von vielen Lösungsindividuen über einem definierten Alphabet (z.B. Binärformat).
- Die Chromosome einer Population variieren in ihren Eigenschaften.
- Die Fähigkeit zum Überleben hängt ab von diesen Eigenschaften, das heißt eine problemspezifische Fitneßfunktion beurteilt alle Chromosome in ihrer Lösungsgüte.
- Es stehen Operatoren zur Vermehrung (Reproduktion) zur Verfügung, aufgrund derer eine an der Fitneß orientierte Neuordnung der Population stattfindet.

In den siebziger Jahren war es John Holland, der erstmals derartige Konzepte vorstellte und dessen bis heute gültiges Grundprogramm eines genetischen Algorithmus sich wie folgt skizzieren läßt (angelehnt an [7]):

1. **Initialisierung einer Startpopulation.**
2. **Fitneß-Bestimmung aller Chromosome.**
3. **Solange das Abbruchkriterium noch nicht erfüllt ist, erzeuge eine neue Generation von Individuen, indem immer wieder probabilistisch einer der drei Operatoren angewendet wird:**
   (a) **Reproduktion: Wähle ein Chromosom der aktuellen Population als Bestandteil der Nächsten.**
   (b) **Mutation: Verändere ein Chromosom an einer oder mehreren Stellen und nimm das Resultierende in die neue Population auf.**
   (c) **Rekombination: Bestimme zwei Individuen und erzeuge mittels Crossover-Funktionen zwei Nachkommen, die in die nächste Generation aufgenommen werden.**
4. **Ausgabe der besten Lösung.**

9.2 Die Implementierung des TSP für den PIC17C43

In der hier vorgestellten Arbeit wird obiges Vorgehen leicht abgewandelt und einem als *Steady-State-Repräsentation* bekannt gewordenem Ansatz gefolgt (siehe [19]). Hierbei wird der Selektionsoperator verworfen und die aktuelle Population von möglichen Lösungen als Grundlage für die nächste Generation verwendet. Lediglich die schlechtesten Chromosome werden durch die erzeugten Nachkommen ersetzt, und zwar auch dann, wenn die neuen Individuen keinen besseren Fitneßwert besitzen als die zu Löschenden. Für die Umsetzung hat dies den Vorteil, daß temporär nicht Speicherplatz für zwei komplette Populationen bereitgestellt werden muß, sondern nur für eine und die festgelegte Anzahl an Kindern pro Generation, die im Allgemeinen deutlich kleiner gewählt wird als die Größe der Population.

Der Ablauf wird durch Abbildung 9.2 verdeutlicht und ist auch leicht aus den Zeilen 79-161 der Hauptdatei `tsp.asm` abzulesen.

1. **Berechnung der Distanz zwischen allen Paaren von Städten der vorgegebenen Instanz.**
2. **Initialisierung einer Startpopulation.**
3. **Fitneß-Bestimmung aller Chromosome.**
4. **Sortierung der Individuen nach aufsteigenden Fitneßwerten.**
5. **Bestes Chromosom an den Kommunikationsprozessor weitergeben.**
6. **Generierung der festgelegten Anzahl an Nachkommen wie folgt:**
   (a) **Mit Wahrscheinlichkeit** 50% **führe** *Greedy-Crossover* **aus.**
   (b) **Mit Wahrscheinlichkeit** 30% **führe** *PMX-Crossover* **aus.**
   (c) **Mit Wahrscheinlichkeit** 10% **führe** *Mutation* **aus.**
   (d) **Mit Wahrscheinlichkeit** 10% **optimiere ein Individuum lokal in 'Viererblöcken'.**
7. **Ersetzen der schlechtesten durch die neu erzeugten Chromosome ⇒ nächste Generation.**
8. **Austausch von Daten mit anderen Prozessorstreifen nach einer frei wählbaren Anzahl an Generationen. Verschoben wird hierbei immer nur das aktuell beste Individuum, die Realisierung ist analog zu 8.1.**
9. **Solange sich innerhalb einer festzulegenden Anzahl an Generationen (`IMP_GEN_LO/HI` in `paramet.h`) das beste Chromosom weiter verbessert, gehe zurück zu Schritt 6.**
10. **Ausgabe der besten Lösung.**

Abbildung 9.2: Das umgesetzte Programmschema des TSP-Problemes

Die ausführliche Beschreibung der einzelnen Programmteile in den nächsten Abschnitten ist dazu gedacht, andere Optimierungsprobleme in relativ kurzer Zeit entwickeln zu können, indem lediglich die Dateien überarbeitet werden, die anwendungsspezifische Informationen enthalten. Die Kodierung der Chromosome (Bitstrings) oder auch die an die Aufgabenstellung angepaßte Fitneßfunktion fallen beispielsweise in diese Gruppe.
Als wichtige Literaturquelle dient [8], dessen Realisierung die Grundlage für die Umsetzung der gestellten Optimierungsaufgabe ist. Die beschriebenen Programmdateien sind im Anhang D abgedruckt.

9.2.1 Definition einer Probleminstanz

Die Spezifizierung einer Probleminstanz beinhaltet zwei Schritte. Zum einen muß in der Datei `paramet.h` die Anzahl der Orte bestimmt werden (in der Variablen NO_OF_CITIES) und zum anderen muß deren Position festgelegt werden. Die Datei `tsp006.tsp` zeigt dies für ein 6-Städte-Problem. Zu beachten ist, daß die einzelnen Punkte sich im gültigen Bereich des 180x180-Einheiten großen Koordinatensystemes befinden.
Für die Implementierung wurden zwei Speicheradressen ($2100 und $2200) vereinbart, an denen sich die beiden Koordinatenlisten befinden. Das 'Laden' benötigter Werte geschieht durch die Angabe des gewünschten Elementes (zum Beispiel die X-Koordinate des dritten Ortes $\Rightarrow$ '3') im Arbeitsregister *wreg*, gefolgt von einem Sprung in die entsprechende Liste (`call TABLE_OF_X_KOORD`). Der Rückgabewert wird dann im *wreg* abgelegt, d.h. im unteren Fall wäre $wreg = 173$. Ein derartiger Aufruf ist unter anderem in der Datei `tspfit.asm` in Zeile 228/229 zu finden.
Eingebunden werden die erstellten Instanzen in der Datei `tspinc.asm`, die auch gleichzeitig die notwendige Größe der Chromosome anpaßt.

```
;*** Tabelle der X-Koordinaten:
        org 2100
TABLE_OF_X_KOORD
        addwf   pcl, 1
        retlw   D'152'
        retlw   D'145'
        retlw   D'173'
        retlw   D'170'
        retlw   D'146'
        retlw   D'166'

;*** Tabelle der Y-Koordinaten:
        org 2200
TABLE_OF_Y_KOORD
        addwf   pcl, 1
        retlw   D'141'
        retlw   D'146'
        retlw   D'148'
        retlw   D'134'
        retlw   D'157'
        retlw   D'138'
```

Abbildung 9.3: Definition eines 6-Städte-Problemes: `tsp006.tsp`

9.2.2 Die Kodierung einer Rundreise

Für die Kodierung einer Rundreise erhalten die Städte in der Reihenfolge ihrer Definition in Abschnitt 9.2.1 eine Identität zugeordnet, unter der sie eindeutig referenzierbar sind.

Für den ersten Ort der Liste ist dies folglich die '1', für den achten Ort die '8' und so weiter.
Ein Chromosom, das stellvertretend für eine mögliche Route steht, wird dann ausgehend vom Startpunkt durch die Verkettung der besuchten Nachfolgeorte bestimmt. Zum Beispiel wird eine Reise von Ort '3' zu Ort '2' und danach zu Ort '1' durch den String (3, 2, 1) modelliert. Bei der Umsetzung in PIC-Assembler werden hierzu die einzelnen Nummern im Binärformat abgespeichert, so daß alle 8 Bit ein neue Identität beginnt. Abbildung 9.4 zeigt noch einmal das Einführungsbeispiel mit der Kodierung der zwei unterschiedlichen Rundreisen.

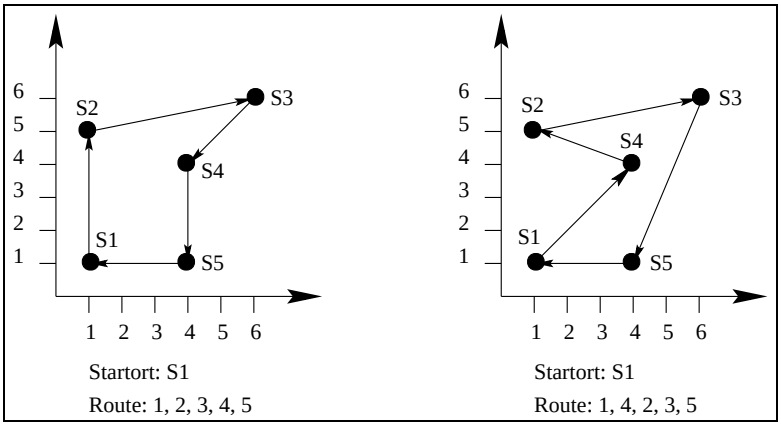


Abbildung 9.4: Kodierung der Routen

9.2.3 Speicheraufteilung

Eine effiziente Organisation des 56 kWord kleinen externen RAM-Speichers der Prozessorstreifen ist von enormer Wichtigkeit, da der Speicher neben den Daten auch noch die eigentliche Programm-Datei speichern muß.

Bei der verwendeten *Steady-State-Repräsentation* wird Platz für eine komplette Population von Chromosomen, die Anzahl an Nachkommen pro Generation und die Fitneßwerte aller Individuen benötigt. Die jeweilige Speicheradresse kann dabei über einen eindeutigen Index abgefragt werden. (siehe [8]). Bei einer Populationsgröße von **p** Chromosomen mit einer Länge von je **l** Bits und **n** Nachkommen pro Generation beträgt der Speicherbedarf im Einzelnen:

- $(\lceil \frac{l}{16} \rceil \cdot p)$ Zellen für die Speicherung der gesamten Population, da jede Speicherzelle 16 Bit speichern kann.
- p Zellen für die einzelnen Fitneßwerte der Chromosome (für die Fitneß ist ein 16 Bit Wert definiert worden).
- $(\lceil \frac{l}{16} \rceil \cdot n)$ Zellen für die Kinder, die pro Generation neu erzeugt werden.
- n Zellen für die Fitneßwerte der Kinder.

Insgesamt stehen für anwendungsrelevante Daten somit nur noch $(56 - \lceil \frac{l}{16} \rceil \cdot (p + n) - p - n)$ kWord zur Verfügung. Damit lassen sich beispielsweise 25 Chromosome der Länge

4 kByte verwalten, wenn man von einer Programmgröße von etwa 3500 Befehlen ausgeht (das TSP-Problem besteht aus weniger als 3000 Befehlen). Die Problemparameter können in `paramet.h` eingestellt werden, die Speicherorganisation wird mit `memptr.asm` gesteuert.

9.2.4 Berechnung der Distanz zwischen allen Paaren von Orten

Um die Entfernung zwischen zwei Orten bei unterschiedlichen Chromosomen nicht jedesmal neu berechnen zu müssen, wird zu Beginn des Algorithmus die Distanz aller Städte zueinander in einer Art Diagonal-Matrix abgelegt.

Wie in der Problemspezifikation am Anfang dieses Kapitels festgelegt wurde, wird für die Entfernung zwischen zwei Orten i und j die euklidische Distanz verwendet:

$$c(i,j) \;=\; \sqrt{(x_i - x_j)^2 \;+\; (y_i - y_j)^2}$$

Bei der Berechnung dieser Formel auf dem PIC17C43 in Assembler ist folgendes Problem zu lösen: Erweiterung des Befehlssatzes um eine standardmäßig nicht enthaltene Wurzel-Funktion, die als Ergebnis auch binäre Nachkommastellen bereitstellt.

Erstgenannte Problematik läßt sich mit einem näherungsweisen, iterativen Vorgehen lösen, bei dem mit der Formel

$$X_{i+1} = \tfrac{1}{2} \cdot (X_i + \frac{A}{X_i})$$

die Wurzel zu einer gegebenen Zahl A bestimmt wird (siehe dazu Abschnitt 6.17 in [25]). Um ein garantiertes Ende der Iterationsschleife zu erhalten, wird die Routine entweder nach 256 Durchläufen oder falls die Werte X_{i+1} und X_i identisch sind beendet.

Die Einführung von Nachkommastellen soll an einem kleinen Beispiel demonstriert werden. Hierzu sei die Wurzel der Zahl 13 ($\sqrt{13} = 3.6055$) mit acht Nachkommastellen zu berechnen. Verwendet wird eine Einbettung in ein 16 Bit Format, die sich durch 16 *Links-Shifts* erreichen läßt und bei der Zahl 13 folgendes Aussehen hat:

128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	Wertigkeit
0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	Bit

Durch Anwendung des obigen iterativen Verfahrens auf diese Zahlendarstellung wird dann die Wurzel berechnet und bei diesem Beispiel insgesamt das Ergebnis 3.6042 zurückgeliefert:

128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	Wertigkeit
0	0	0	0	0	0	1	1	1	0	0	1	1	0	1	1	Bit

Die Tabellen 9.2 und 9.3 zeigen zum einen, daß die Initialisierungsphase nicht wesentlich mehr Zeit in Anspruch nimmt, wenn man zu Beginn die Entfernung zu allen Paaren von Orten bestimmt (als Vergleich zur Nichterstellung einer solchen Matrix). Auf der anderen Seite sinkt die benötigte Zeit, um eine Generation neu zu generieren dadurch drastisch.

Hierbei ist zu beachten, daß pro Generationsschritt die Güte der neu erzeugten Nachkommen beurteilt werden muß, was zu einem enormen Rechenaufwand führt, falls die Distanz zwischen zwei Städten, die in unterschiedlichen Chromosomen nacheinander besucht werden, jedesmal neu berechnet werden muß.

Bei der linken Spalte der Tabellen handelt es sich um die Angabe der untersuchten Probleminstanz, wobei die Nummer die Anzahl der Städte angibt. Auf der CD sind die Daten als vorgefertigte *INHX8M*-Dateien im Verzeichnis `Diplomarbeit\Travelling Salesman Problem\TSP-Instanzen` gespeichert.
Die verwendeten Programmparameter wurden für beide Testläufe wie folgt festgelegt: die Population besteht aus 100 Individuen, pro Generation werden 70 Nachkommen erzeugt und der Startwert für den internen Zufallszahlengenerator ist $3045.

Problem	Aufwand ohne Distanzmatrix	Aufwand mit Distanzmatrix
tsp008	4s	<1s
tsp012	6s	<1s
tsp016	7s	1s
tsp020	10s	1s
tsp024	11s	2s
tsp028	14s	2s
tsp032	15s	3s
tsp036	18s	4s
tsp040	19s	4s
tsp060	29s	10s
tsp100	50s	27s
tsp140	69s	52s
tsp180	90s	84s
tsp220	109s	126s
tsp252	124s	145s

Tabelle 9.2: Zeitaufwand der Initialisierungsphase ohne und mit Distanzmatrix

Ein Nachteil der Berechnung der Diagonalmatrix liegt im immensen Speicherbedarf. Bei einem n-Städte-Problem werden insgesamt $(\frac{n\cdot(n-1)}{2})$ Zellen benötigt, wodurch die maximale Größe der Population und die Länge der Chromosome aus Abschnitt 9.2.3 natürlich dementsprechend kleiner gewählt werden müssen.

9.2.5 Die Fitneßfunktion

Um die Gesamtlänge einer Route $(s_1, s_2, s_3, \ldots, s_n)$ zu bestimmen, kann die im vorigen Abschnitt berechnete Distanzmatrix genutzt werden. Dabei werden in einer Schleife die entsprechende Orts-Einträge ausgelesen und aufsummiert, dies entspricht folgender Formel (Funktion `compute_tsp_fitness` in der Datei `tspfit.asm`):

$$\textit{Länge der Route} \;=\; \left(\sum_{i=2}^{n} c(s_{i-1}, s_i)\right) + c(s_n, s_1)$$

Das Koordinatensystem wurde auf ein 180x180-Einheiten großes Feld festgelegt, somit ist die maximale Länge einer Route nie größer als 65536 Einheiten, 16 Bit sind zur Speicherung eines Fitneßwertes ausreichend.

Problem	1 Generation ohne Matrix	1 Generation mit Matrix
tsp008	6s	<1s
tsp012	10s	<1s
tsp016	11s	<1s
tsp020	17s	<1s
tsp024	19s	<1s
tsp028	22s	1s
tsp032	27s	1s
tsp036	32s	1s
tsp040	37s	2s
tsp060	55s	3s
tsp100	85s	6s
tsp140	139s	16s
tsp180	184s	23s
tsp220	180s	28s
tsp252	267s	42s

Tabelle 9.3: Zeitaufwand eines Generationsschrittes ohne und mit Distanzmatrix

9.2.6 Generierung einer Startpopulation

Prinzipiell existieren zwei Ansätze, um eine initiale Generation zu kreieren. Entweder zufällige Erzeugung der Bitstrings oder Verwendung anwendungsspezifischer Informationen. In der ersten Programmvariante wurden alle Chromosome in einem ersten Schritt mit der Route (1, 2, 3, ..., n) initialisiert, um danach eine bestimmte Anzahl an Mutationen an jedem Individuum auszuführen. Dadurch wird neben einer 'zufälligen' Population auch noch gewährleistet, daß alle Elemente aufgrund der Implementierung des Mutations-Operators gültige Lösungen darstellen, d.h. kein Ort mehrmals besucht wird (siehe dazu auch 9.2.8 und 9.2.10).

Der in der endgültigen Programmversion eingesetzte Ansatz folgt nach dem Initialisierungsschritt einer anderen Strategie. In lokalen Optimierungen wird immer eine Mutation ausgeführt, um danach das originale mit dem veränderten Chromosom bezüglich der Güte zu vergleichen. Das Bessere der beiden wird dann beibehalten und der nächste Durchlauf eingeleitet. Beendet wird die Schleife, falls sich innerhalb von 100 Mutationen keine Verbesserung zur jeweils letzten Rundreise eingestellt hat. In Pseudo-Code stellt sich dieses Vorgehen wie folgt dar (mit $Fitness(X) = Länge\ der\ Route\ X$, Minimierungsproblem):

```
i = 0;
chromosom = (1,2,3,...,n);
while (i ≤ 100) {
  i = i+1;
  next_chromosom = Mutation(chromosom);
  if (Fitness(next_chromosom) ≤ Fitness(chromosom) {
    chromosom = next_chromosom;
    i = 0;
  }
}
```

Tabelle 9.4 stellt einen Vergleich der zuvor beschriebenen Ideen dar. Dies zeigt sehr deutlich das Potential der zweiten Variante. Obwohl die Populationsgröße beim zufallsbasierten Ansatz mit 100 Chromosomen zehnmal so groß ist, ist die Güte des besten Individuums nach Abschluß der Initialisierung bis zu dreimal schlechter.

Problem	**Zufällig (100 Chromosome)**	**mit Optimierung (10 Chromosome)**
tsp008	37	38
tsp012	334	289
tsp016	96	84
tsp020	763	609
tsp024	965	661
tsp028	1328	870
tsp032	1340	818
tsp036	1175	723
tsp040	1450	856
tsp060	2461	1340
tsp100	7213	3305
tsp140	9112	4212
tsp180	14530	5796
tsp220	17485	6122
tsp252	24173	6378

Tabelle 9.4: Güte des besten Individuums der initialen Population

9.2.7 Das Auswahlverfahren zur Bestimmung von Individuen

Bei allen nachfolgenden klassischen Operatoren genetischer Algorithmen wird eine Routine benötigt, die ein oder zwei Elternteile für Rekombinationszwecke auswählt. In dieser Arbeit wird die klassische Funktion *Roulette-Wheel-Selection* benutzt, die in [4], [7], [8] und [13] ausführlich beschrieben ist.

Die Chromosome sollen hierbei für weiterführende Operationen mit einer Wahrscheinlichkeit ausgewählt werden, die proportional zu ihrer Güte ist. Gibt es beispielsweise drei Chromosome mit den Werten $Fit_1 = 10$, $Fit_2 = 5$ und $Fit_3 = 5$, so soll das erste Individuum mit einer doppelt so hohen Wahrscheinlichkeit zufällig bestimmt werden wie die beiden anderen. Die Namensgebung dieser Funktion stammt dabei vom Glücksspiel *Roulette*, bei dem eine Kugel in einem zufälligen Sektor (mit einer Gewinnzahl beschriftet) zum Stillstand kommt. Bei der Umsetzung für genetische Algorithmen wird für jedes Chromosom ein Kreisbogen definiert, dessen Länge proportional zur Fitneß des Chromosomes ist. Der Bereich, in dem die Kugel zum Stehen kommt, ist also für sehr 'gute' Individuen deutlich höher als für schlechte Chromosome. Abbildung 9.5 zeigt die Einteilung der Kreisbögen für das obige Zahlenbeispiel.

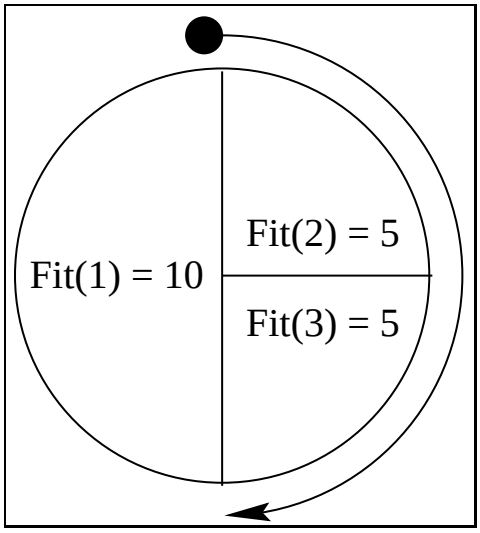


Abbildung 9.5: Das Auswahlverfahren: Roulette-Wheel-Selection

9.2.8 Der PMX-Crossover-Operator

Unter Crossover-Operatoren versteht man Funktionen, die mit Hilfe eines Auswahlverfahrens wie *Roulette-Wheel-Selection* zwei Elternteile (Chromosome) bestimmen und aus diesen Daten zwei Nachkommen erzeugen. Hierzu wird beim Einpunkt-Crossover zufällig eine Kreuzungsposition innerhalb der beiden Individuen gewählt und die zwei Teilstrings hinter dieser Stelle ausgetauscht. Beim Zweipunkt-Crossover gibt es dementsprechend zwei Schnittstellen. In Abbildung 9.6 wird eine solche Operation skizziert.

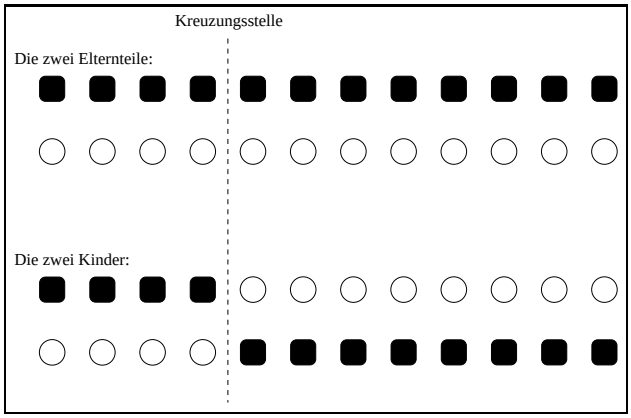


Abbildung 9.6: Einpunkt-Crossover

Man sieht leicht ein, daß die Funktion bei Anwendungen wie dem TSP- oder ähnlichen Problemen zu ungültigen Lösungen führen kann. Eventuell wird dieselbe Stadt mehrmals besucht. Um solche Chromosome zu verhindern, bietet es sich an, diese mit der Fitneßfunktion als sehr schlecht zu bewerten, damit sie möglichst schnell in den nächsten Generationen durch neue (gültige) Individuen ersetzt werden. Im allgemeinen sinkt dadurch jedoch die Konvergenzrate des Algorithmus stark ab, weil der Anteil an Lösungen mit mehrfach auftretenden Orten trotzdem sehr hoch bleibt. Besser ist es, im Anschluß an die eigentliche

Routine gleich eine Reorganisation anzuschließen.

Die in dieser Arbeit eingesetzte PMX-Crossover-Version (*Partially Matched Crossover*) merkt sich alle Paare von Orts-Identitäten **hinter** der Kreuzungsposition, um aufgrund dieser Information gegebenenfalls Duplikate **vor** der Kreuzungslinie aufzulösen.

Bei den beiden Elternteilen (1, 2, 3, 4) und (3, 2, 4, 1) würden bei einem Schnitt hinter der zweiten Stelle die Kinder (1, 2, 4, 1) und (3, 2, 3, 4) entstehen. Um Duplikate erkennen zu können, speichert man sich die Paare an der dritten und vierten Position der Elternteile ($3 \leftrightarrow 4$, $4 \leftrightarrow 1$). Jetzt werden alle doppelten Vorkommen von Städten an den ersten zwei Positionen durch obige Pendants ersetzt, was zu (4, 2, 4, 1) und (4, 2, 3, 4) führt. Ein nochmaliger Reorganisationsschritt liefert dann mit (3, 2, 4, 1) beziehungsweise (1, 2, 3, 4) zwei gültige Nachkommen. Unter Umständen sind also mehrere Durchläufe nötig. Ein Beispiel hierzu zeigt auch Abbildung 9.7.

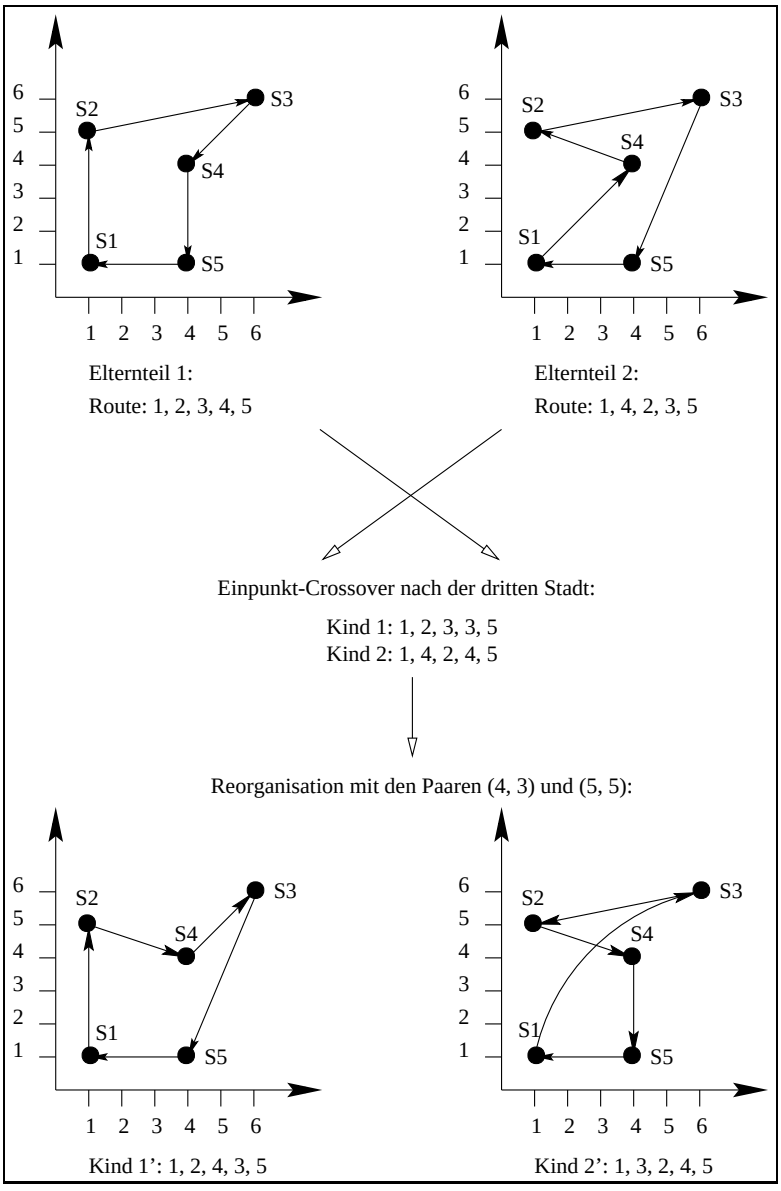


Abbildung 9.7: PMX-Crossover an einem Beispiel

Um sich alle auftretenden 'Tausch-Paare' zu merken, wird bei der Umsetzung ein Speicherbereich für n Städte ($n = \#Orte$) reserviert, der in einer Schleife solange durchlaufen wird, bis alle Positionen bis zur Schnittstelle 'korrekt' sind (Anhang D.13).

9.2.9 Der Greedy-Crossover-Operator

Unter obigen Gesichtspunkten ist der Begriff *Greedy-Crossover* für die nun vorgestellte Funktion etwas mißverständlich, zumal nur ein Kind erzeugt wird, allerdings wurde er als solcher in [4] eingeführt.

Die Idee läßt sich wie folgt beschreiben. Der Startort der entstehenden Rundreise (des Kindes) wird zufällig bestimmt. In den beiden Elternteilen wird daraufhin nach den entsprechenden Nachfolgestädten gesucht und diejenige als nächster Punkt der Route ausgewählt, deren Entfernung zum Startort geringer ist. Dieses Schema wird weitergeführt, um das Kind zu komplettieren, und gültige Lösungen werden dadurch erzwungen, daß nach jedem Schritt die bereits erzeugte Route auf Duplikate geprüft wird. Wird ein doppeltes Vorkommen eines Ortes festgestellt, so wird das zuletzt eingefügte Element durch ein zufällig gewähltes ersetzt (erfordert neuerliche Prüfungsphase).

Greedy-Crossover kann als eine *kantenorientierte* Funktion angesehen werden, bei der kostengünstigen Distanzen zwischen zwei Orten der Vorzug vor längeren gegeben wird. Sind in den Elternteilen bereits sehr gute Teilabschnitte enthalten, so werden diese zu einer größeren Route zusammengesetzt und die Fitneß im Vergleich zu den Eltern im allgemeinen gesteigert.
Weil PMX-Crossover dazu neigt, die relative Position der Städte innerhalb einer Rundreise beizubehalten, ergänzen sich beide Crossover-Varianten ausgezeichnet.

9.2.10 Die Mutation

Der Erfolg der diversen Crossover-Routinen bei der sukzessiven Verbesserung einer Population darf nicht darüber hinwegtäuschen, daß zum 'Gelingen', d.h. zum Finden sehr guter Resultate eines genetischen Algorithmus, die Mutation eine wichtige Rolle spielt. Während die oberen beiden Funktionen die in der Kodierung enthaltene Information 'nur' neu verteilen, können bei der Mutation, sofern sie auf Zufallsentscheidungen basiert, neue Sachverhalte entstehen. Zum Verlassen suboptimaler, lokaler Minima ist dies der einzigste Ausweg.
Bei der hier verwendeten Variante wird ein Chromosom (eine Rundreise) dahingehend modifiziert, daß zufällig zwei Städte gewählt werden, die in ihrer Position vertauscht werden, wie dies auch Abbildung 9.8 grafisch verdeutlicht. Die Funktion ist unter dem Namen `tsp_mutation` im Abschnitt D.13 enthalten.

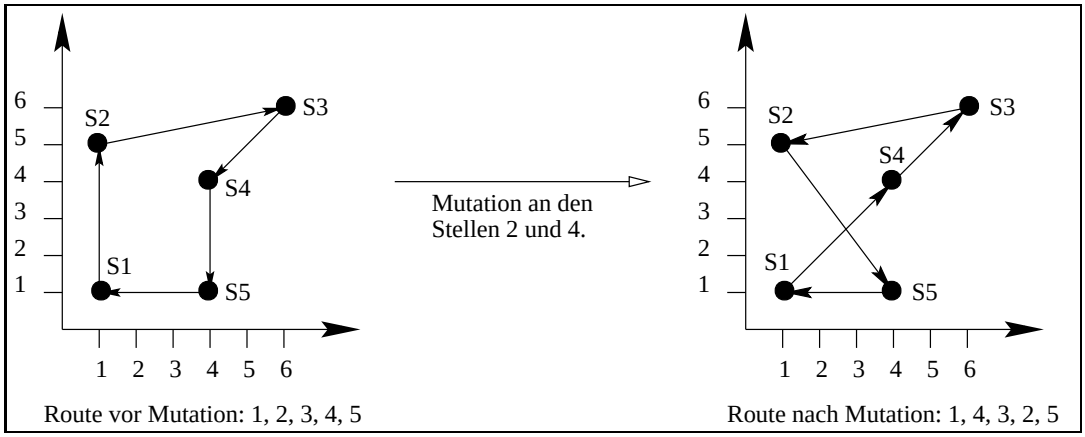


Abbildung 9.8: Mutation an einem Beispiel

Kapitel 10

Ergebnisse

In diesem Abschnitt sollen die Resultate aus den Versuchsreihen dargestellt werden, mit denen das Zusammenspiel zwischen den einzelnen Hardwarekomponenten und der dazugehörigen Software getestet wurde. Auch soll die Leistungsfähigkeit des Multiprozessorsystemes untersucht werden. Im Vordergrund steht hierbei die Strategie (Topologie) des Datenaustausches bei maximal vier eingesetzten Prozessoren. Als Beispielapplikation dient das im vorigen Kapitel implementierte Travelling Salesman Problem.

Weniger von Bedeutung ist zum jetzigen Zeitpunkt des Projektes der Laufzeitvergleich mit anderen Rechner-Netzwerken wie *Workstation-Cluster*. Dies ist mit der geringen Taktfrequenz und der 8 Bit Architektur der Prozessorstreifen zu begründen. Soll beispielsweise die Route eines 100 Städte-Problemes gelesen werden, so bedeutet das ohne weitere Operationen wie Crossover bereits 100 Lesezugriffe (jede Stadt wird mit einer 8 Bit Nummer kodiert $\Rightarrow$ 800 Bit). Die Analyse aus Tabelle 9.3 zeigt die Dauer eines Generationsschrittes und läßt erahnen, daß die benötigte Zeit zum Lösen größerer Instanzen mehrere Stunden in Anspruch nehmen kann. Aus diesem Grund sind auch nur Aufgabenstellungen mit maximal 100 Städten getestet worden. Endgültige Aussagen lassen sich jedoch erst nach einer Portierung auf die ISA-Steckkarten mit maximal 81 Recheneinheiten machen, zumal die vorgestellte Hardware wesentlich günstiger ist als Unix-Rechner.

Um die berechneten Lösungen mit einem Referenzwert vergleichen zu können, wurde in Anlehnung an [7] ein *Threshold Accepting*-Algorithmus in C programmiert und auf alle Probleminstanzen angewendet. Analog zur Generierung einer Startpopulation in Abschnitt 9.2.6 wird dabei **eine** Rundreise X durch Mutation zu einer Route X' verändert. Für die nächsten Iterationen wird X' dann verwendet, falls gilt: $Fitness(X') \leq Fitness(X) + T$ (mit $Fitness(X) = \textit{Länge der Route } X$, Minimierungsproblem). T stellt einen Schwellenwert dar, der das Verlassen lokaler Minima möglich macht und sukzessive verringert wird. Abbildung 10.1 zeigt den zugrundeliegenden Pseudo-Code, das Programm ist auf der CD-ROM im Verzeichnis `Diplomarbeit\Travelling Salesman Problem\Sonstige TSP-Routinen` zu finden.

Tabelle 10.1 und 10.2 zeigen die erzielten Meßergebnisse. Die Probleminstanzen gehen im wesentlichen auf die *TSPLIB95* von Gerhard Reinelt von der Universität Heidelberg zurück, die auf der Seite `http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html` verfügbar ist. Diese Bibliothek stellt für die im vorigen Abschnitt eingeführten TSP-Varianten zahlreiche Problemstellungen mit den jeweils (derzeit) besten Ergebnissen

```
i = 0;
T = 100;
chromosom = (1, 2, 3, ..., n);
while (i ≤ 200000) {
  i = i+1;
  next_chromosom = Mutation(chromosom);
  if (Fitness(next_chromosom) ≤ Fitness(chromosom) + T) {
    T = T − 0.002;
    i = 0;
    chromosome = next_chromosome;
  }
}
```

Abbildung 10.1: Pseudo-Code des *Threshold Accepting*-Algorithmus

bereit. Viele Probleminstanzen sind allerdings nicht kompatibel zum festgelegten 180x180-Einheiten großen Koordinatensystem, so daß die angepaßten Instanzen keinen direkten Vergleich zulassen. Trotzdem bietet die vorhandene Dokumentation der TSPLIB95 einen guten Einstieg in die Thematik und gibt gleichzeitig auch nützliche Programmierhinweise.

Da die Topologie des Datenaustausches bei lediglich vier Prozessoren nicht viel Spielraum bietet, sind bei den Untersuchungen nur die Verdrahtung als *Ring* beziehungsweise die vollständige Kommunikation zwischen allen Recheneinheiten gewählt worden (*Alle*). Eine schematische Darstellung der beiden Möglichkeiten zeigt Abbildung 10.2.

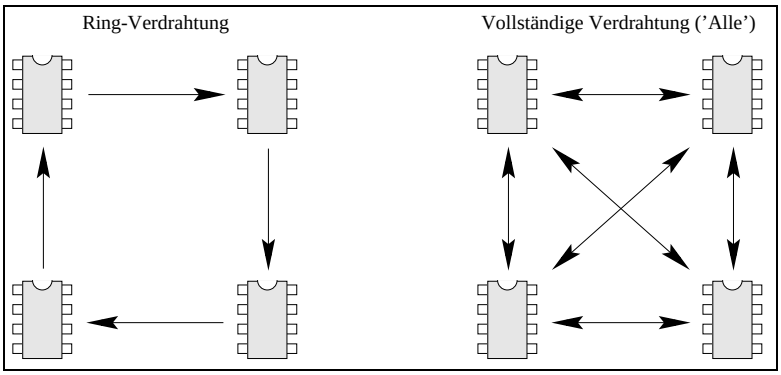


Abbildung 10.2: Zwei Topologien

Bis auf wenige Ausnahmen sind die Meßergebnisse identisch oder sogar deutlich besser als die Referenzwerte, das Verfahren liefert also hervorragende Lösungen. Die 'Ausreißer' sind unter anderem mit den gewählten Parametern zu erklären. Neben einem Datenaustausch nach jeweils 32 Generationen ist das Abbruchkriterium aus Laufzeitgründen so eingestellt worden, daß die Programmausführung nach 200 Iterationen ohne Verbesserung des besten Individuums endet. Dieser enggesteckte Zeitrahmen erlaubt folglich nur wenig Spielraum, um lokale Minima zu verlassen, der Algorithmus stoppt hier sehr früh.

Trotzdem wird besonders bei den 'großen' Instanzen (60- und 100-Städte-Problem) herausgearbeitet, daß vier Recheneinheiten aufgrund der insgesamt vierfachen Populationsgröße

schnell sehr gute Rundreisen ermöglichen. Vereinfacht ausgedrückt: Die hohe, auf viele Prozessorstreifen verteilte Chromosomenvielfalt vergrößert die Wahrscheinlichkeit kurzer Routen. Die Abbildungen 10.3 und 10.4 zeigen die 'Sprünge', um welche sich die Gesamtdistanz der besten derzeitigen Lösung verringert, in Abhängigkeit der Generationsschritte.

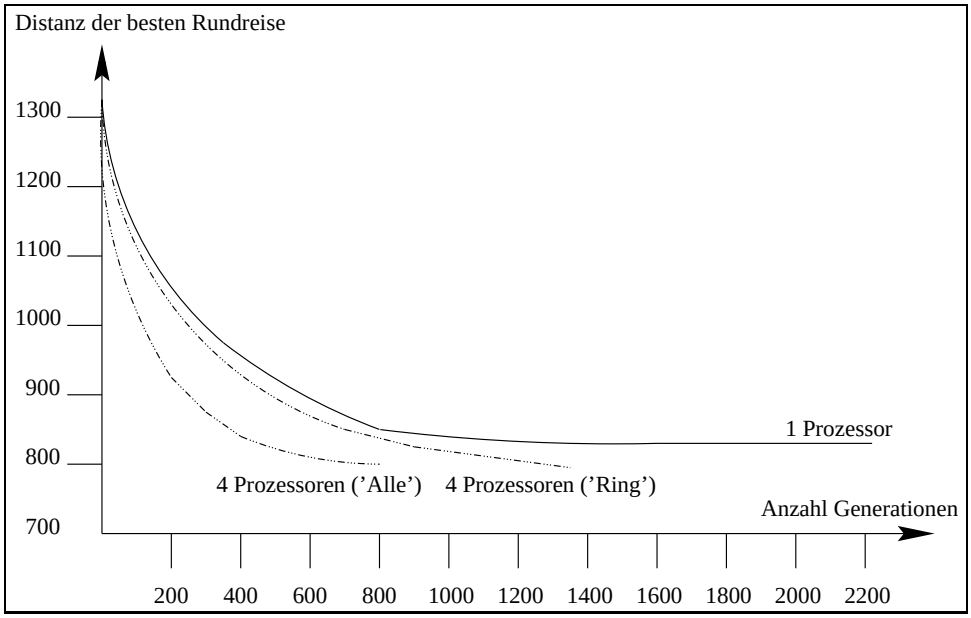


Abbildung 10.3: Verkürzung der besten Route beim 60-Städte-Problem

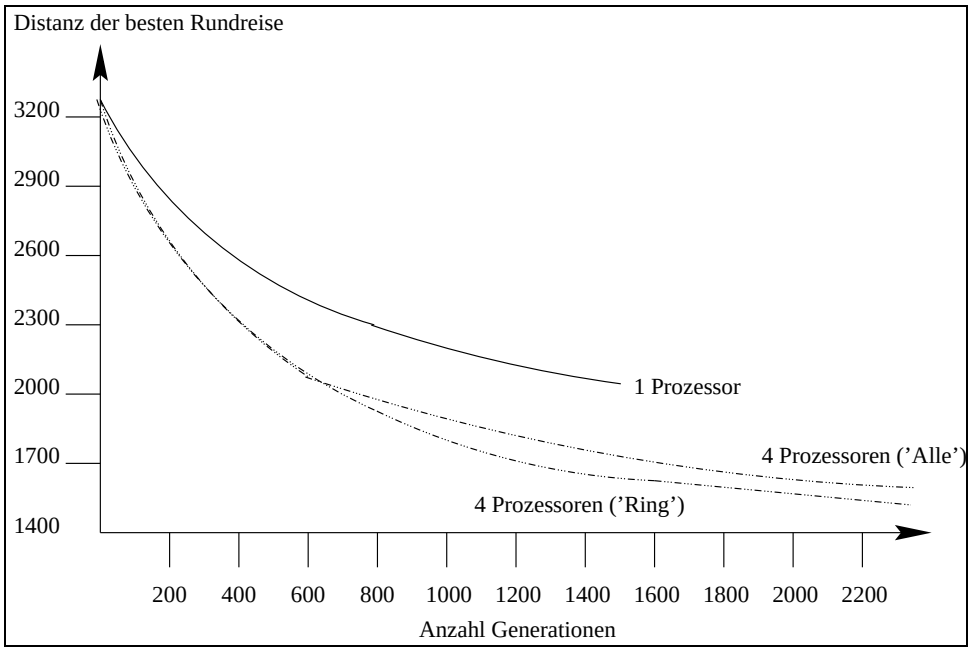


Abbildung 10.4: Verkürzung der besten Route beim 100-Städte-Problem

Name	**#Städte**	**#Chromosome**	**#Kinder**	**#Prozessoren**	**Topologie**	**#Generationen**	**Lösung**	*Threshold Accepting*
tsp012	12	50	32	1		09	285.696	285.715
tsp012	12	50	32	2	Ring	76	285.696	285.715
tsp012	12	50	32	3	Ring	94	285.696	285.715
tsp012	12	50	32	3	Alle	209	285.696	285.715
tsp012	12	50	32	4	Ring	65	285.696	285.715
tsp012	12	50	32	4	Alle	10	285.696	285.715
tsp016	16	50	32	1		294	75.692	75.043
tsp016	16	50	32	2	Ring	72	75.981	75.043
tsp016	16	50	32	3	Ring	54	75.598	75.043
tsp016	16	50	32	3	Alle	30	75.598	75.043
tsp016	16	50	32	4	Ring	94	75.598	75.043
tsp016	16	50	32	4	Alle	138	74.512	75.043
tsp020	20	50	32	1		353	424.829	424.663
tsp020	20	50	32	2	Ring	189	424.625	424.663
tsp020	20	50	32	3	Ring	100	424.829	424.663
tsp020	20	50	32	3	Alle	138	424.625	424.663
tsp020	20	50	32	4	Ring	143	424.625	424.663
tsp020	20	50	32	4	Alle	144	424.829	424.663
tsp024	24	40	26	1		93	548.969	543.174
tsp024	24	40	26	2	Ring	364	540.934	543.174
tsp024	24	40	26	3	Ring	312	542.680	543.174
tsp024	24	40	26	3	Alle	202	539.489	543.174
tsp024	24	40	26	4	Ring	168	540.934	543.174
tsp024	24	40	26	4	Alle	108	540.934	543.174
tsp028	28	40	26	1		622	598.047	606.154
tsp028	28	40	26	2	Ring	351	602.622	606.154
tsp028	28	40	26	3	Ring	814	594.727	606.154
tsp028	28	40	26	3	Alle	325	594.727	606.154
tsp028	28	40	26	4	Ring	538	598.047	606.154
tsp028	28	40	26	4	Alle	389	594.727	606.154

Tabelle 10.1: Güte der untersuchten Probleminstanzen

Name	**#Städte**	**#Chromosome**	**#Kinder**	**#Prozessoren**	**Topologie**	**#Generationen**	**Lösung**	*Threshold Accepting*
tsp032	32	40	26	1		354	468.356	475.162
tsp032	32	40	26	2	Ring	299	456.934	475.162
tsp032	32	40	26	3	Ring	300	456.414	475.162
tsp032	32	40	26	3	Alle	181	476.797	475.162
tsp032	32	40	26	4	Ring	572	456.410	475.162
tsp032	32	40	26	4	Alle	177	470.629	475.162
tsp036	36	40	26	1		442	462.559	483.447
tsp036	36	40	26	2	Ring	409	481.543	483.447
tsp036	36	40	26	3	Ring	228	447.493	483.447
tsp036	36	40	26	3	Alle	301	473.934	483.447
tsp036	36	40	26	4	Ring	284	447.493	483.447
tsp036	36	40	26	4	Alle	99	464.059	483.447
tsp040	40	40	26	1		575	528.250	545.851
tsp040	40	40	26	2	Ring	487	578.079	545.851
tsp040	40	40	26	3	Ring	371	537.497	545.851
tsp040	40	40	26	3	Alle	448	538.598	545.851
tsp040	40	40	26	4	Ring	515	559.910	545.851
tsp040	40	40	26	4	Alle	348	542.012	545.851
tsp060	60	20	14	1		2175	818.243	859.012
tsp060	60	20	14	2	Ring	1235	803.805	859.012
tsp060	60	20	14	3	Ring	1139	801.129	859.012
tsp060	60	20	14	3	Alle	744	795.454	859.012
tsp060	60	20	14	4	Ring	1326	794.731	859.012
tsp060	60	20	14	4	Alle	780	795.774	859.012
tsp100	100	10	6	1		1342	1872.954	1616.419
tsp100	100	10	6	2	Ring	1992	1642.954	1616.419
tsp100	100	10	6	3	Ring	2412	1608.618	1616.419
tsp100	100	10	6	3	Alle	2831	1612.723	1616.419
tsp100	100	10	6	4	Ring	2374	1544.551	1616.419
tsp100	100	10	6	4	Alle	2212	1604.403	1616.419

Tabelle 10.2: Güte der untersuchten Probleminstanzen (Fortsetzung)

Kapitel 11

Ausblick

In dieser Diplomarbeit wurde eine erste Grundversion des Multiprozessorsystemes mit vier Recheneinheiten in Betrieb genommen und die verschiedenen Komponenten programmiert. Als erste Anwendung wurde das *NP-vollständige* Travelling Salesman Problem implementiert. Die dabei erzielten Ergebnisse der Testläufe mit verschiedenen Probleminstanzen lassen Rückschlüsse auf zukünftige Arbeiten am Gesamtprojekt zu. Hierbei erscheint eine Unterteilung in vier Gruppen von Zielrichtungen sinnvoll:

1. **Alternative Einsatzgebiete.** Aufgrund der Tatsache, daß jeder Prozessorstreifen über diverse getrennt programmier- und ansprechbare I/O-Ports verfügt, ist besonders das kleine Parallelprozessorsystem für den Einsatz in der Steuerungs- und Regelungstechnik verwendbar, weil es sich autonom von einem Computer betreiben läßt. Denkbar wäre zum Beispiel die Steuerung einer Fertigungsstraße, eventuell in Kooperation mit dem Institut für Mikrosystemtechnik, das in Freiburg neben dem Institut für Informatik ebenfalls in die neugeschaffene Fakultät für Angewandte Wissenschaft eingegliedert ist.

2. **Erweiterung des Betriebssystemes des Kommunikationsprozessors.** Bislang ist es bei dem implementierten Betriebssystem des Kommunikationsprozessors mit genetischen Algorithmen nur möglich, daß die Recheneinheiten **eine** Lösung austauschen, die unter Umständen immer dieselbe ist. Das ist immer dann der Fall, wenn die Fortpflanzungs-Operatoren kein neues, bestes Chromosom kreieren, was zur Folge hat, daß sich die Populationen zu sehr angleichen. Hier wäre ein Betriebssytem von Vorteil, bei dem der Kommunikationsprozessor pro Schritt mehrere Individuen eines jeden Prozessorstreifens erhält und immer nur diejenigen weitergibt, die im Vergleich zum vorigen Schritt neue Informationen darstellen.

3. **Umsetzung der Algorithmen auf das große Multiprozessorsystem.** Hierzu müssen die in dieser Arbeit vorgestellten Algorithmen teilweise überarbeitet werden, denn die Hardware beider Multiprozessorsysteme unterscheidet sich deutlich. Exemplarisch seien an dieser Stelle zwei Punkte aufgeführt:
   Während die Recheneinheiten des kleinen Systemes direkt über den Kommunikationsprozessor Daten austauschen, wird dies bei den großen Steckkarten über den sogenannten *I-Cube* realisiert, der eine Leitungsmatrix darstellt und die seriellen Schnittstellen der einzelnen Prozessorstreifen direkt miteinander verbindet. Dieser Sachverhalt erfordert somit die Konfiguration der seriellen PIC17C43-Schnittstellen durch geeignete Routinen, die bislang nicht benötigt wurden.

Ein zweiter Aspekt ist das Zusammenspiel zwischen Anwender und System, das über den ISA-Bus und nicht mehr über die serielle MC68340-Schnittstelle realisiert wird. Zu beiden Punkten bieten [3] und [5] Lösungsvorschläge an.

4. **Verwendung leistungsstärkerer Prozessoren.** Die benötigte Laufzeit zum Lösen der Probleminstanzen des TSP-Problemes hat gezeigt, daß die Prozessorstreifen nur bedingt für rechenintensive Aufgabenstellungen geeignet sind. Nachteilig sind besonders der mit 128 kByte zu klein dimensionierte externe Speicher, die mit 16.78 MHz geringe Taktfrequenz, die 8 Bit-Architektur und das Fehlen eines mathematischen Coprozessors.
   Sollten diese Einschränkungen auch beim Einsatz auf dem großen Multiprozessorsystem auftreten (Punkt 3), d.h. nicht durch die massive Parallelität kompensiert werden können, so muß über alternative Hardware nachgedacht werden. Aufgrund der Modularität des Systemes ist es aber möglich, leistungsfähigere Prozessorstreifen zu entwickeln und einzusetzen.
   Übergangsweise könnten aber auch nur einzelne Recheneinheiten ersetzt werden, zum Beispiel durch reine Speicherbausteine, die getrennt von den restlichen Komponenten vom Kommunikationsprozessor angesteuert werden müssten.

Literaturverzeichnis

[1] Atmel Corporation. *Atmel-WinCUPL, User's Manual.* 1999.

[2] Atmel Corporation. *Using the ATF1500(A) CPLD.* 1999.

[3] Christian Faller. *Programmierung der Kontrolleinheit des Multiprozessorsystems.* Studienarbeit. Albert-Ludwigs-Universität Freiburg, 1999.

[4] David E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley Publishing Company Inc., 1989.

[5] Mark Jonas. *Inbetriebnahme eines Mikrocontroller-Systems mit ISA-Schnittstelle, Zugriff auf Hardware unter Windows und Entwicklung einer Arbeitsumgebung.* Diplomarbeit. Albert-Ludwigs-Universität Freiburg, 1999.

[6] Ewald Liess Josef Fuchs. *M68300-Mikrocontroller, Band 3.* Franzis-Verlag GmbH, 1994.

[7] Werner Kinnebrock. *Optimierung mit genetischen und selektiven Algorithmen.* Oldenbourg Verlag GmbH, 1994.

[8] Timo Lakner. *Erstellung einer Programmbibliothek genetischer Algorithmen.* Studienarbeit. Albert-Ludwigs-Universität Freiburg, 1996.

[9] Microchip Technology Inc. *Embedded Control Handbook.* 1994.

[10] Microchip Technology Inc. *Microchip Data Book.* 1994.

[11] Microchip Technology Inc. *MPASM Assembler, User's Guide.* 1995.

[12] Microchip Technology Inc. *PIC17C4X Data Sheet.* 1995.

[13] Tom M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[14] Motorola Inc. *MC68340 Integrated Processor with DMA - User's Manual.* 1992.

[15] PD Computer - Gesellschaft für Prozeß- und Datentechnik mbH. *pd_a340, Bus-Belegung.* 1994.

[16] PD Computer - Gesellschaft für Prozeß- und Datentechnik mbH. *pd_a340, Microcontroller-Modul mit MC68340 im Scheckkartenformat.* 1994.

[17] PD Computer - Gesellschaft für Prozeß- und Datentechnik mbH. *HSB, Hierarchisches System von Bussen.* 1995.

[18] PD Computer - Gesellschaft für Prozeß- und Datentechnik mbH. *pd_cmp43, Hardware-Beschreibung.* 1996.

[19] Gregor Pudelko. *Genetische Algorithmen zur Kanalverdrahtung.* Diplomarbeit. Johann-Wolfgang-Goethe-Universität Frankfurt am Main, 1996.

[20] Michael Rose. *Mikroprozessor PIC17C42, Architektur und Applikation.* Hüthig Buch Verlag, 1994.

[21] Uwe Schöning. *Theoretische Informatik - kurzgefaßt.* Spektrum Akademischer Verlag, Heidelberg, 1995.

[22] Tobias Schubert. *Programmierung des Kommunikationsprozessors.* Studienarbeit. Albert-Ludwigs-Universität Freiburg, 1996.

[23] Ingo Wegener. *Theoretische Informatik.* B.G. Teubner, 1993.

[24] Anton Nausch Werner Hilf. *M68000-Familie, Teil 1: Grundlagen und Architektur.* te-wi Verlag GmbH, 1984.

[25] Anton Nausch Werner Hilf. *M68000-Familie, Teil 2: Anwendung und 68000-Bausteine.* te-wi Verlag GmbH, 1984.

[26] Helmut Will. *Strukturiert programmieren mit Turbo Pascal.* Franzis-Verlag GmbH, München, 1990.

Anhang A

Programm-Datei ATMEL ATF1500

```
Name        ATF1500;
Partno      None;
Date        08/08/99;
Revision    None;
Designer    Tobias Schubert;
Company     University of Freiburg;
Assembly    None;
Location    None;
Device      F1500;

/** Kontakt: Tobias Schubert, schubert@informatik.uni-freiburg.de     **/
/**                                                                   **/
/**  Signal-Definitionen:                                             **/
/**  Siehe auch Schaltplan des "PIC-Streifen".                        **/
/**                                                                   **/
/**  Globale Reset-Leitung, die alle Prozessoren zuruecksetzt:        **/
Pin  1  = !reset     ;      /** Globales Reset                        **/

/**  Kontrollsignale vom bzw. zum PIC17C43:                           **/
Pin  2  = !poe       ;      /** pic output enable (low active)        **/
Pin  4  = !pwr       ;      /** pic write (low active)                **/
Pin  5  =  pale      ;      /** pic address latch enable              **/
Pin  6  = !pint      ;      /** pic interrupt (low active)            **/
Pin  7  =  map01     ;
Pin 11  = !mwe       ;      /** memory write enable (low active)      **/
Pin 12  =  mce_neg   ;      /** memory chip enable (low active)       **/

/** Addressleitungen vom bzw. zum PIC17C43 (8 Bit):                   **/
Pin 13  =  ad8       ;
Pin 14  =  ad9       ;
Pin 16  =  ad10      ;
Pin 17  =  ad11      ;
Pin 18  =  ad12      ;
Pin 19  =  ad13      ;
Pin 20  =  ad14      ;
Pin 21  =  ad15      ;

/**  Kontrollsignale vom bzw. zum Motorola 68340:                     **/
Pin 24  = !bwr       ;      /** bus write (low active)                **/
Pin 25  = !bcs       ;      /** bus chip select (low active)          **/
Pin 26  =  bint_neg  ;      /** bus interrupt (low active)            **/
Pin 27  = !breq      ;      /** bus request (low active)              **/
Pin 28  = !dtack     ;      /** data acknowledge (low active)         **/
Pin 29  =  bale      ;      /** bus address latch enable              **/
Pin 31  = !bgnt      ;      /** bus grant (low active)                **/
Pin 32  = !iackin    ;      /** interrupt acknowledge(low active)     **/

/**  Addressignale vom bzw. zum Motorola 68340:                       **/
Pin 33  =  bad7      ;
Pin 34  =  bad6      ;
Pin 36  = !bad5      ;      /** aus Compiler-Gruenden: es ist nicht   **/
Pin 37  =  bad4      ;      /** moeglich einen I/O-Pin negiert und    **/
Pin 38  =  bad3      ;      /** nichtnegiert in logischen Gleichungen **/
```

```
Pin 39  =  bad2      ;      /** zu verwenden, so dass diese          **/
Pin 40  =  bad1      ;      /** schaltungstechnisch realisiert werden **/
Pin 41  =  bad0      ;      /** koennen.                             **/

Pin 43  =  clk       ;      /** Globales Clock-Signal                **/
Pin 44  = !brd       ;      /** bus read (low active)                **/

/** Steuersignale fuer den externen RAM-Baustein, der auf           **/
/** jedem "PIC-Streifen" zur Verfuegung steht (64 kWord).           **/
Pin  8  = a12    ;
Pin  9  = a13    ;

/*********************************************************************/
/** Definition des logischen Verhaltens des Bausteines:             **/

/** Dekodierung der anliegenden, gueltigen                          **/
/** (pale = "pic address latch enable" = 1) Adresse.                **/
/** ad15..8 =  $1000  ==> an_mot = 1 ==> Datenuebergabe an Motorola **/
/** ad15..8 <> $1000  ==> an_mot = 0 ==> Ansprechen des externen RAM **/
/** Compiler waehlt nichtbenutzte logische Zelle.                   **/
pinnode   = an_mot;
an_mot.l  = !ad15.io & !ad14.io & !ad13.io & ad12.io &
            !ad11.io & !ad10.io & !ad9.io  & !ad8.io;
an_mot.le = pale;

/** Steuerung der "Output enable"-Leitung des D-FF zur              **/
/** Erzeugung von Interrupts beim Motorola 68340.                   **/
/** an_mot = 1 ==> bint_oe = 1 ==> Interrupt (s.u.)                 **/
/** an_mot = 0 ==> bint_oe = 0 ==> kein Interrupt                   **/
pinnode    = bint_oe;
bint_oe.d  = 'b'1;
bint_oe.ck = an_mot;
bint_oe.ar = iackin # reset;
bint_oe.oe = 'b'1;

/** Erzeugung von Interrupts an den Motorola 68340.                 **/
/** Der Interrupt wird geloescht durch obiges OE-Signal,            **/
/** dass genau dann wieder inaktiv wird, wenn der Motorola          **/
/** den eingehenden Interrupt bestaetigt hat (!iackin = 1).         **/
bint_neg.d  = 'b'0;
bint_neg.ck = an_mot;
bint_neg.oe = bint_oe;

/** Interruptsignal an den PIC-Prozessor:                           **/
/** Hier ist das Timing weniger wichtig als oben, da                **/
/** der PIC17C43 sofort nach Beendigung des momentanen              **/
/** Befehles auf externe Interrupts eingeht. Beim Motorola          **/
/** kann die Verifikation der ausloesenden Quelle lt.               **/
/** Datenbuch bis zu 44 Takte dauern, in denen der Inter-           **/
/** rupt aktiv gehalten werden muss.                                **/
/** "bwr" symbolisiert einen Schreibbefehl, d.h. das PLD            **/
/** muss Daten vom Motorola entgegen nehmen, waehrend "bcs"         **/
/** automatisch erzeugt wird und den betreffenden PIC-Pro-          **/
/** zessor bzw. das entsprechende PLD eindeutig auswaehlt.          **/
!pint = bwr & bcs;

/** Datentransfer: PIC --> Motorola                                 **/
/** 1. Aufgabe: Daten vom PIC17C43 in 8 Latches speichern.          **/
/**     Dies wird durch "pwr" = "pic write" = 1 und                 **/
/**     "an_mot" = 1 (s.o.) signalisiert.                           **/
/** 2. Aufgabe: Daten an den Motorola 68340 weitergeben.           **/
/**     Dies wird durch "brd" = "bus read" = 1 signalisiert.        **/
/** Als Speicherzellen werden die zum Motorola gehoerenden          **/
/** Datenleitungen bad7..0 = "bus address data7..0" genutzt.        **/
bad7.l  = ad15.io;
bad7.le = pwr & an_mot;
bad7.oe = brd & bcs;

bad6.l  = ad14.io;
bad6.le = pwr & an_mot;
bad6.oe = brd & bcs;

bad5.l  = !ad13.io;
```

```
    bad5.le = pwr & an_mot;
    bad5.oe = brd & bcs;

130 bad4.l  = ad12.io;
    bad4.le = pwr & an_mot;
    bad4.oe = brd & bcs;

    bad3.l  = ad11.io;
    bad3.le = pwr & an_mot;
    bad3.oe = brd & bcs;

    bad2.l  = ad10.io;
    bad2.le = pwr & an_mot;
140 bad2.oe = brd & bcs;

    bad1.l  = ad9.io;
    bad1.le = pwr & an_mot;
    bad1.oe = brd & bcs;

    bad0.l  = ad8.io;
    bad0.le = pwr & an_mot;
    bad0.oe = brd & bcs;

150 /** Datentransfer: Motorola --> PIC                              **/
    /** 1. Aufgabe: Daten vom Motorola in 8 Latches speichern.       **/
    /**      Dies wird durch "bwr" = "bus write" = 1 und             **/
    /**      "bcs" = "bus chip select" = 1 (s.o.) signalisiert.      **/
    /** 2. Aufgabe: Daten an den PIC17C43 weitergeben.               **/
    /**      Dies wird durch "!pwr" = "!pic write" = 1 und durch     **/
    /**      das Hilfssignal "lesen_pic" = 1 realisiert, das aus     **/
    /**      den anliegenden Adressen einen Datentransfer bei        **/
    /**      $1100 erkennt (ansonsten handelt es sich um einen       **/
    /**      Lesebefehl aus dem externen RAM).                       **/
160 /** Als Speicherzellen werden die zum PIC17C43 gehoerenden       **/
    /** Datenleitungen ad15..8 = "address data15..8" genutzt.        **/
    pinnode      = lesen_pic;
    lesen_pic.l  = !ad15.io & !ad14.io & !ad13.io & ad12.io &
                   !ad11.io & !ad10.io & !ad9.io  & ad8.io;
    lesen_pic.le = pale;

    ad8.l  = bad0.io;
    ad8.le = bwr & bcs;
    ad8.oe = !pwr & poe & lesen_pic;
170
    ad9.l  = bad1.io;
    ad9.le = bwr & bcs;
    ad9.oe = !pwr & poe & lesen_pic;

    ad10.l  = bad2.io;
    ad10.le = bwr & bcs;
    ad10.oe = !pwr & poe & lesen_pic;

    ad11.l  = bad3.io;
180 ad11.le = bwr & bcs;
    ad11.oe = !pwr & poe & lesen_pic;

    ad12.l  = bad4.io;
    ad12.le = bwr & bcs;
    ad12.oe = !pwr & poe & lesen_pic;

    ad13.l  = bad5.io;
    ad13.le = bwr & bcs;
    ad13.oe = !pwr & poe & lesen_pic;
190
    ad14.l  = bad6.io;
    ad14.le = bwr & bcs;
    ad14.oe = !pwr & poe & lesen_pic;

    ad15.l  = bad7.io;
    ad15.le = bwr & bcs;
    ad15.oe = !pwr & poe & lesen_pic;

    /** Signale zum Ansprechen des externen PIC-Speichers:           **/
```

```
200 a12.l  = ad12.io;          /** Um RAM-Bereiche auszuwaehlen          **/
    a12.le = pale ;            /** (vgl. Dokumentation).                 **/
    a12.oe = 'b'1 ;
    a12.ar = reset;

    a13.l  = ad13.io & map01; /** Wichtig, um RAM-Zugriffe bei Map0/1 = 0 **/
    a13.le = pale ;            /** im Bereich $2000 - $3FFF in den RAM-  **/
    a13.oe = 'b'1 ;            /** Bereich $0000 - $2000 zu aendern.     **/
    a13.ar = reset;

210 /** RAM-chip-enable: RAM fuer Lese- bzw. Schreibvorgang "enablen".  **/
    /** mce_neg steht fuer "inverted memory chip enable".              **/
    mce_neg.l  = !((ad15.io # ad14.io # ad13.io) & map01
                # (!ad15.io & !ad14.io & ad13.io & !map01));
    mce_neg.le = pale ;
    mce_neg.oe = 'b'1 ;
    mce_neg.ap = reset;        /** Preset, weil mce low-aktiv ist.       **/

    /** RAM-write-enable: RAM fuer Schreibzugriff "enablen".           **/
    /** mwe_neg steht fuer "inverted memory write enable".             **/
220 /** Diesmal wird ein D-FF verwendet, weil der Table Write-         **/
    /** Befehl des PIC17C43 2 Taktzyklen benoetigt, d.h. der           **/
    /** eigentliche Schreibzyklus wird im zweiten Schritt ge-          **/
    /** macht, dort wird also erst !mwe = 0 benoetigt.                 **/
    /** Zusaetzlich wird das Signal bis zur naechsten stei-            **/
    /** genden Clock-Flanke erweitert, um auch schnellere RAMs         **/
    /** korrekt unterstuetzen zu koennen.                              **/

    mwe.d  = pwr  ;
    mwe.ck = clk  ;
230 mwe.oe = 'b'1 ;
    mwe.ap = reset;            /** Preset, weil mwe low-aktiv ist.       **/
    mwe.ar = pwr  ;
```

Anhang B

Motorola MC68340 Betriebssystem

Dateiname	**Funktion**
`tonux.a`	Hauptprogramm, Einbindung der Funktionsblöcke
`68340reg.h`	Register-Definitionen
`def.h`	Variablen-Definitionen
`vector.h`	Definition der Sprungadressen und Interrupt-Routinen
`mot2pc.a`	Funktionen zur Datenausgabe am Bildschirm
`basics.a`	Hauptschleife, Initialisierungen, Datenübergabe, etc.
`uart.a`	Initialisierung der RS232-Schnittstelle
`pictest.a`	Kontaktaufnahme zu den PIC-Prozessoren
`getprg.a`	PIC-Programme vom Anwender/PC empfangen
`sendprg.a`	PIC-Programme an alle Prozessoren weitergeben
`datasize.a`	Chromosomengröße für Datenaustausch festlegen
`topology.a`	Topologie des Datenaustausches festlegen
`clearpld.a`	Speicherzellen der ATF1500-PLDs initialisieren
`initreg.a`	Variablen initialisieren
`convert.a`	Konvertierungsroutinen (z.B. ASCII $\longleftrightarrow$ Hex/Binär)
`messages.a`	Alle Ausgaben und Meldungen vom Motorola zum PC
`irq.a`	Funktionen zur Interrupt-Auswertung der PIC17C43-Prozessoren
`change.a`	Funktionen zum Datenaustausch zwischen den Prozessoren
`getres.a`	Empfang des besten jeweiligen PIC17C43-Ergebnisses
`showres.a`	Ausgabe der besten Resultate am PC-Bildschirm

Tabelle B.1: MC68340 Betriebssystem - Die Programm-Dateien und ihre Funktion

B.1 Die Hauptdatei: tonux.a

```
*******************************************************
***             Hauptprogramm                       ***
*** Autor:      Tobias Schubert                     ***
*** EMail:      schubert@informatik.uni-freiburg.de ***
*** Datum:      29.07.1999                          ***
*** Datei:      tonux.a                             ***
*******************************************************

  org $0
  include inc/68340reg.h        * Register-Definitionen wie PORTA,...
  include inc/def.h             * Eigene Variablen-Definitionen
  include inc/vector.h

  dc.l   end
  dc.b   154
  even

*** Programm-Start:
ga_start:
  move.w #$2400,sr              * Interrupt level = 4, Supervisor Mode
  moveq  #0,d0
  movec  d0,vbr                 * Zeiger auf Interrupt-Tabelle

*** Initialisieren: serielle Schnittstelle, Programm-Uebergabe, ...
  bsr.l  BASICS

*** Programm-Ende mit entsprechender Mitteilung
  bsr.l  END_OF_PROGRAM
  bgnd                          * Breakpoint, Debugger stoppt Programmablauf

  include asm/mot2pc.a          * Routinen zur Datenuebertragung an den PC
  include asm/basics.a          * Initialisierungen, Hauptschleife, ...
end:
```

B.2 Register-Deklarationen: 68340reg.h

```
*************************************************************
***                  Definition der On-Chip-Register      ***
*** Autor:           Tobias Schubert                      ***
*** EMail:           schubert@informatik.uni-freiburg.de  ***
*** Datum:           26.07.1999                           ***
*** Datei-Name:      68340reg.h                           ***
*************************************************************

MBA             equ $FFFFF000   * Baseaddr. of the MC68340-onchip-register
mbar            equ $03ff00     * Module Base Addr. Register (CPU-Space)

*** System Integration Module Registers
simcr           equ MBA+$000    * SI-Module Configuration Register
syncr           equ MBA+$004    * Clock Synthesizer Control Register
avr             equ MBA+$006    * Autovector Register
rsr             equ MBA+$007    * Reset Status Register
porta           equ MBA+$011    * Port A Data Register
ddra            equ MBA+$013    * Port A Data Direction Register
ppara1          equ MBA+$015    * Port A Pin Assignment Register 1
ppara2          equ MBA+$017    * Port A Pin Assignment Register 2
portb           equ MBA+$019    * Port B Data Register
portb1          equ MBA+$01b    * Port B Data  Register (= portb)
ddrb            equ MBA+$01d    * Port B Data Direction Register
pparb           equ MBA+$01f    * Port B Pin Assignment Register
swiv            equ MBA+$020    * Software Interrupt Vector Register
sypcr           equ MBA+$021    * System Protection Control Register
picr            equ MBA+$022    * Periodic Interrupt Control Register
pitr            equ MBA+$024    * Periodic Interrupt Timer Register
swsr            equ MBA+$027    * Sofware Service Register
```

```
30  cs0am           equ MBA+$040    * Chip Select 0 Address Mask
    cs0ba           equ MBA+$044    * Chip Select 0 Base Address
    cs1am           equ MBA+$048    * Chip Select 1 Address Mask
    cs1ba           equ MBA+$04c    * Chip Select 1 Base Address
    cs2am           equ MBA+$050    * Chip Select 2 Address Mask
    cs2ba           equ MBA+$054    * Chip Select 2 Base Address
    cs3am           equ MBA+$058    * Chip Select 3 Address Mask
    cs3ba           equ MBA+$05c    * Chip Select 3 Base Address

    *** DMA Module Registers
40  dma1_mcr        equ MBA+$780    * Channel 1 Module Config. Register
    dma1_intr       equ MBA+$784    * Channel 1 Interrupt Register
    dma1_ccr        equ MBA+$788    * Channel 1 Control Register
    dma1_csr        equ MBA+$78a    * Channel 1 Status Register
    dma1_fcr        equ MBA+$78b    * Channel 1 Fuction Code Register
    dma1_sar        equ MBA+$78c    * Channel 1 Source Address Register
    dma1_dar        equ MBA+$790    * Channel 1 Destination Addr. Register
    dma1_btc        equ MBA+$794    * Channel 1 Byte Transfer Counter
    dma2_mcr        equ MBA+$7a0    * Channel 2 Module Config. Register
    dma2_intr       equ MBA+$7a4    * Channel 2 Interrupt Register
50  dma2_ccr        equ MBA+$7aa    * Channel 2 Control Register
    dma2_csr        equ MBA+$7aa    * Channel 2 Status Register
    dma2_fcr        equ MBA+$7ab    * Channel 2 Fuction Code Register
    dma2_sar        equ MBA+$7ac    * Channel 2 Source Address Register
    dma2_dar        equ MBA+$7b0    * Channel 2 Destination Addr. Register
    dma2_btc        equ MBA+$7b4    * Channel 2 Byte Transfer Counter

    *** Serial Module Registers
    ser_mcr         equ MBA+$700    * Serial Module Config. Register
    ser_ilr         equ MBA+$704    * Interrupt Level Register
60  ser_ivr         equ MBA+$705    * Interrupt Vector Register
    ser_mr1a        equ MBA+$710    * Channel A Mode Register 1
    ser_sra         equ MBA+$711    * Channel A Status Register
    ser_csra        equ MBA+$711    * Channel A Clock-Select Register
    ser_cra         equ MBA+$712    * Channel A Command Register
    ser_rba         equ MBA+$713    * Channel A Receiver Buffer
    ser_tba         equ MBA+$713    * Channel A Transmitter Buffer
    ser_ipcr        equ MBA+$714    * Input Port Change Register
    ser_acr         equ MBA+$714    * Auxiliary Control Register
    ser_isr         equ MBA+$715    * Interrupt Status Register
70  ser_ier         equ MBA+$715    * Interrupt Enable Register
    ser_mr1b        equ MBA+$718    * Channel B Mode Register 1
    ser_srb         equ MBA+$719    * Channel B Status Register
    ser_csrb        equ MBA+$719    * Channel B Clock-Select Register
    ser_crb         equ MBA+$71a    * Channel B Command Register
    ser_rbb         equ MBA+$71b    * Channel B Receiver Buffer
    ser_tbb         equ MBA+$71b    * Channel B Transmitter Buffer
    ser_ip          equ MBA+$71d    * Input Port Register
    ser_opcr        equ MBA+$71d    * Output Port Control Register
    ser_opbs        equ MBA+$71e    * Output Port Bit Set
80  ser_opbr        equ MBA+$71f    * Output Port Bit Reset
    ser_mr2a        equ MBA+$720    * Channel A Mode Register 2
    ser_mr2b        equ MBA+$721    * Channel B Mode Register 2

    *** Timer Module Registers
    t1_mcr          equ MBA+$600    * Timer 1 Module Config. Register
    t1_ir           equ MBA+$604    * Timer 1 Interrupt Register
    t1_cr           equ MBA+$606    * Timer 1 Control Register
    t1_sr           equ MBA+$608    * Timer 1 Status/Prescaler Register
    t1_cntr         equ MBA+$60a    * Timer 1 Counter Register
90  t1_prel1        equ MBA+$60c    * Timer 1 Preload 1 Register
    t1_prel2        equ MBA+$60e    * Timer 1 Preload 2 Register
    t1_com          equ MBA+$610    * Timer 1 Compare Register
    t2_mcr          equ MBA+$640    * Timer 2 Module Config. Register
    t2_ir           equ MBA+$644    * Timer 2 Interrupt Register
    t2_cr           equ MBA+$646    * Timer 2 Control Register
    t2_sr           equ MBA+$648    * Timer 2 Status/Prescaler Register
    t2_cntr         equ MBA+$64a    * Timer 2 Counter Register
    t2_prel1        equ MBA+$64c    * Timer 2 Preload 1 Register
    t2_prel2        equ MBA+$64e    * Timer 2 Preload 2 Register
100 t2_com          equ MBA+$650    * Timer 2 Compare Register
```

B.3 Variablen-Deklarationen: `def.h`

```
*************************************************************
***                   Eigene Definitionen                 ***
*** Autor:            Tobias Schubert                     ***
*** EMail:            schubert@informatik.uni-freiburg.de ***
*** Datum:            11.09.1999                          ***
*** Datei-Name:       def.h                               ***
*************************************************************

RAM_START       equ 0                   * StartAdresse des RAM
STACK_INI       equ $10000              * Zeiger auf oberstes Stack-Element
RAM_END         equ $10000              * Verweis auf Stack

pic0            equ $FFFF8000           * Adresse von PIC0 im I/O-Bereich
pic1            equ $FFFF8100           * Adresse von PIC1 im I/O-Bereich
pic2            equ $FFFF8200           * Adresse von PIC2 im I/O-Bereich
pic3            equ $FFFF8300           * Adresse von PIC3 im I/O-Bereich

var_start       equ $00011000
***                 StartAdresse fuer eigene Variablen.
***                 Die genetischen Programme beginnen
***                 ab var_start+$100. ($11000 selektiert
***                 die "obere" RAM-Bank)

control0        equ var_start+$00       * Speichert PIC0-Signalwort
control1        equ var_start+$01       * Speichert PIC1-Signalwort
control2        equ var_start+$02       * Speichert PIC2-Signalwort
control3        equ var_start+$03       * Speichert PIC3-Signalwort

topo0           equ var_start+$04       * "Topologie"-Matrix:
topo1           equ var_start+$05       * Bit x=1 in topoy -> PIC y sendet
topo2           equ var_start+$06       * sein derzeit bestes Chromosom an
topo3           equ var_start+$07       * den PIC x.

chr_sizeL       equ var_start+$08       * Low-Byte eines Chromosomes
chr_sizeH       equ var_start+$09       * High-Byte eines Chromosomes

fitness0_lo     equ var_start+$0A       * Fitnesswert von PIC-Prozessor 0
fitness0_hi     equ var_start+$0B
fitness1_lo     equ var_start+$0C       * Fitnesswert von PIC-Prozessor 1
fitness1_hi     equ var_start+$0D
fitness2_lo     equ var_start+$0E       * Fitnesswert von PIC-Prozessor 2
fitness2_hi     equ var_start+$0F
fitness3_lo     equ var_start+$10       * Fitnesswert von PIC-Prozessor 3
fitness3_hi     equ var_start+$11

mode            equ var_start+$12       * Betriebsmodus des Motorola
correct_pics    equ var_start+$13       * Bit x=0 --> PIC x ist ansprechbar
sum_instr       equ var_start+$14       * "Sum of instructions"
chk_sum         equ var_start+$15       * "Checksum"
results         equ var_start+$16
***                 In der Hauptschleife Indiz dafuer,
***                 ob ein PIC noch Kontakt aufnehmen will.
***                 Bit x=1 -> PICx hat die Aufgabe "geloest".

gen0_hi         equ var_start+$17       * Anzahl Generationen bei PIC0
gen0_mi         equ var_start+$18
gen0_lo         equ var_start+$19
gen1_hi         equ var_start+$1A       * Anzahl Generationen bei PIC1
gen1_mi         equ var_start+$1B
gen1_lo         equ var_start+$1C
gen2_hi         equ var_start+$1D       * Anzahl Generationen bei PIC2
gen2_mi         equ var_start+$1E
gen2_lo         equ var_start+$1F
gen3_hi         equ var_start+$20       * Anzahl Generationen bei PIC3
gen3_mi         equ var_start+$21
gen3_lo         equ var_start+$22

child_size0     equ var_start+$23       * Nachkommen/Generation bei PIC0
child_size1     equ var_start+$24       * Nachkommen/Generation bei PIC1
child_size2     equ var_start+$25       * Nachkommen/Generation bei PIC2
```

```
child_size3     equ var_start+$26       * Nachkommen/Generation bei PIC3

pop_size0       equ var_start+$27       * Populationsgroesse bei PIC0
pop_size1       equ var_start+$28       * Populationsgroesse bei PIC1
pop_size2       equ var_start+$29       * Populationsgroesse bei PIC2
pop_size3       equ var_start+$2A       * Populationsgroesse bei PIC3

transmitter_pic equ var_start+$2B       * Datentausch in "change.a"
receiver_pic    equ var_start+$2C       * Datentausch in "change.a"
trans_fit_hi    equ var_start+$2D       * Datentausch in "change.a"
trans_fit_lo    equ var_start+$2E       * Datentausch in "change.a"

gen_intvall_hi  equ var_start+$2F       * Intervall zwischen 2 PIC-Anfragen
gen_intvall_lo  equ var_start+$30       * Intervall zwischen 2 PIC-Anfragen

random          equ var_start+$31       * Zufallszahlen-Seed üfr PICs

remainder0      equ var_start+$32       * Nachkommastellen bei PIC0-Fitn.
remainder1      equ var_start+$33       * Nachkommastellen bei PIC1-Fitn.
remainder2      equ var_start+$34       * Nachkommastellen bei PIC2-Fitn.
remainder3      equ var_start+$35       * Nachkommastellen bei PIC3-Fitn.

rbit0           equ var_start+$36       * 1. Nachkommastelle der Ausgabe
rbit1           equ var_start+$37       * 2. Nachkommastelle der Ausgabe
rbit2           equ var_start+$38       * 3. Nachkommastelle der Ausgabe
```

B.4 Vektor-Tabelle der Interrupt-Routinen: vector.h

```
*******************************************************
***             Interrupt/Exception-Tabelle         ***
*** Autor:      Tobias Schubert                     ***
*** EMail:      schubert@informatik.uni-freiburg.de ***
*** Datum:      27.07.1999                          ***
*** Datei-Name: vector.h                            ***
*******************************************************

  opt nol

*** Vektor-Tabelle:              Nr.: Bedeutung
  dc.l STACK_INI                  0: Reset: Initial Stack Pointer
  dc.l ga_start                   1: Reset: Initial Program Counter
  dc.l _uninit_excpt_             2: Bus Error
  dc.l _uninit_excpt_             3: Address Error
  dc.l _uninit_excpt_             4: Illegal Instruction
  dc.l _uninit_excpt_             5: Zero Divide
  dc.l _uninit_excpt_             6: CHK, CHK2 Instructions
  dc.l _uninit_excpt_             7: TRAPcc, TRAPV Instructions
  dc.l _uninit_excpt_             8: Privilege Violation
  dc.l _uninit_excpt_             9: Trace
  dc.l _uninit_excpt_            10: Line 1010 Emulator
  dc.l _uninit_excpt_            11: Line 1111 Emulator
  dc.l _uninit_excpt_            12: Hardware Breakpoint
  dc.l _uninit_excpt_            13: (Reserved for Protocol Violation)
  dc.l _uninit_excpt_            14: Format Error/Uninitialized Interrupt
  dc.l _uninit_excpt_            15: Format Error/Uninitialized Interrupt
  dc.l _uninit_excpt_            16: (Unassigned, Reserved)
  dc.l _uninit_excpt_            17: (Unassigned, Reserved)
  dc.l _uninit_excpt_            18: (Unassigned, Reserved)
  dc.l _uninit_excpt_            19: (Unassigned, Reserved)
  dc.l _uninit_excpt_            20: (Unassigned, Reserved)
  dc.l _uninit_excpt_            21: (Unassigned, Reserved)
  dc.l _uninit_excpt_            22: (Unassigned, Reserved)
  dc.l _uninit_excpt_            23: (Unassigned, Reserved)
  dc.l spurious_interrupt        24: Spurious Interrupt
  dc.l _uninit_excpt_            25: Level 1 Interrupt Autovector
  dc.l _uninit_excpt_            26: Level 2 Interrupt Autovector
  dc.l _uninit_excpt_            27: Level 3 Interrupt Autovector
```

```
40    dc.l _uninit_excpt_            28: Level 4 Interrupt Autovector
      dc.l picx_interrupt            29: Level 5 Interrupt Autovector
      dc.l picx_interrupt            30: Level 6 Interrupt Autovector
      dc.l _uninit_excpt_            31: Level 7 Interrupt Autovector
      dc.l _uninit_excpt_            32: TRAP 0 Instruction Vector
      dc.l _uninit_excpt_            33: TRAP 1 Instruction Vector
      dc.l _uninit_excpt_            34: TRAP 2 Instruction Vector
      dc.l _uninit_excpt_            35: TRAP 3 Instruction Vector
      dc.l _uninit_excpt_            36: TRAP 4 Instruction Vector
      dc.l _uninit_excpt_            37: TRAP 5 Instruction Vector
50    dc.l _uninit_excpt_            38: TRAP 6 Instruction Vector
      dc.l _uninit_excpt_            39: TRAP 7 Instruction Vector
      dc.l _uninit_excpt_            40: TRAP 8 Instruction Vector
      dc.l _uninit_excpt_            41: TRAP 9 Instruction Vector
      dc.l _uninit_excpt_            42: TRAP A Instruction Vector
      dc.l _uninit_excpt_            43: TRAP B Instruction Vector
      dc.l _uninit_excpt_            44: TRAP C Instruction Vector
      dc.l _uninit_excpt_            45: TRAP D Instruction Vector
      dc.l _uninit_excpt_            46: TRAP E Instruction Vector
      dc.l _uninit_excpt_            47: TRAP F Instruction Vector
60    dc.l _uninit_excpt_            48: (Reserved for Coprocessor)
      dc.l _uninit_excpt_            49: (Reserved for Coprocessor)
      dc.l _uninit_excpt_            50: (Reserved for Coprocessor)
      dc.l _uninit_excpt_            51: (Reserved for Coprocessor)
      dc.l _uninit_excpt_            52: (Reserved for Coprocessor)
      dc.l _uninit_excpt_            53: (Reserved for Coprocessor)
      dc.l _uninit_excpt_            54: (Reserved for Coprocessor)
      dc.l _uninit_excpt_            55: (Reserved for Coprocessor)
      dc.l _uninit_excpt_            56: (Reserved for Coprocessor)
      dc.l _uninit_excpt_            57: (Reserved for Coprocessor)
70    dc.l _uninit_excpt_            58: (Reserved for Coprocessor)
      dc.l _uninit_excpt_            59: (Unassigned, Reserved)
      dc.l _uninit_excpt_            60: (Unassigned, Reserved)
      dc.l _uninit_excpt_            61: (Unassigned, Reserved)
      dc.l _uninit_excpt_            62: (Unassigned, Reserved)
      dc.l _uninit_excpt_            63: (Unassigned, Reserved)
      dc.l _uninit_excpt_            64: User-Defined Vector
      dc.l _uninit_excpt_            65: User-Defined Vector
      dc.l _uninit_excpt_            66: User-Defined Vector
      dc.l _uninit_excpt_            67: User-Defined Vector
80    dc.l _uninit_excpt_            68: User-Defined Vector
      dc.l _uninit_excpt_            69: User-Defined Vector
      dc.l _uninit_excpt_            70: User-Defined Vector
      dc.l _uninit_excpt_            71: User-Defined Vector
      dc.l _uninit_excpt_            72: User-Defined Vector
      dc.l _uninit_excpt_            73: User-Defined Vector
      dc.l _uninit_excpt_            74: User-Defined Vector
      dc.l _uninit_excpt_            75: User-Defined Vector
      dc.l _uninit_excpt_            76: User-Defined Vector
      dc.l _uninit_excpt_            77: User-Defined Vector
90    dc.l _uninit_excpt_            78: User-Defined Vector
      dc.l _uninit_excpt_            79: User-Defined Vector
      dc.l _uninit_excpt_            80: User-Defined Vector
      dc.l _uninit_excpt_            81: User-Defined Vector
      dc.l _uninit_excpt_            82: User-Defined Vector
      dc.l _uninit_excpt_            83: User-Defined Vector
      dc.l _uninit_excpt_            84: User-Defined Vector
      dc.l _uninit_excpt_            85: User-Defined Vector
      dc.l _uninit_excpt_            86: User-Defined Vector
      dc.l _uninit_excpt_            87: User-Defined Vector
100   dc.l _uninit_excpt_            88: User-Defined Vector
      dc.l _uninit_excpt_            89: User-Defined Vector
      dc.l _uninit_excpt_            90: User-Defined Vector
      dc.l _uninit_excpt_            91: User-Defined Vector
      dc.l _uninit_excpt_            92: User-Defined Vector
      dc.l _uninit_excpt_            93: User-Defined Vector
      dc.l _uninit_excpt_            94: User-Defined Vector
      dc.l _uninit_excpt_            95: User-Defined Vector
      dc.l _uninit_excpt_            96: User-Defined Vector
      dc.l _uninit_excpt_            97: User-Defined Vector
110   dc.l _uninit_excpt_            98: User-Defined Vector
      dc.l _uninit_excpt_            99: User-Defined Vector
      dc.l _uninit_excpt_           100: User-Defined Vector
```

```
        dc.l _uninit_excpt_             101: User-Defined Vector
        dc.l _uninit_excpt_             102: User-Defined Vector
        dc.l _uninit_excpt_             103: User-Defined Vector
        dc.l _uninit_excpt_             104: User-Defined Vector
        dc.l _uninit_excpt_             105: User-Defined Vector
        dc.l _uninit_excpt_             106: User-Defined Vector
        dc.l _uninit_excpt_             107: User-Defined Vector
120     dc.l _uninit_excpt_             108: User-Defined Vector
        dc.l _uninit_excpt_             109: User-Defined Vector
        dc.l _uninit_excpt_             110: User-Defined Vector
        dc.l _uninit_excpt_             111: User-Defined Vector
        dc.l _uninit_excpt_             112: User-Defined Vector
        dc.l _uninit_excpt_             113: User-Defined Vector
        dc.l _uninit_excpt_             114: User-Defined Vector
        dc.l _uninit_excpt_             115: User-Defined Vector
        dc.l _uninit_excpt_             116: User-Defined Vector
        dc.l _uninit_excpt_             117: User-Defined Vector
130     dc.l _uninit_excpt_             118: User-Defined Vector
        dc.l _uninit_excpt_             119: User-Defined Vector
        dc.l _uninit_excpt_             120: User-Defined Vector
        dc.l _uninit_excpt_             121: User-Defined Vector
        dc.l _uninit_excpt_             122: User-Defined Vector
        dc.l _uninit_excpt_             123: User-Defined Vector
        dc.l _uninit_excpt_             124: User-Defined Vector
        dc.l _uninit_excpt_             125: User-Defined Vector
        dc.l _uninit_excpt_             126: User-Defined Vector
        dc.l _uninit_excpt_             127: User-Defined Vector
140     dc.l _uninit_excpt_             128: User-Defined Vector
        dc.l _uninit_excpt_             129: User-Defined Vector
        dc.l _uninit_excpt_             130: User-Defined Vector
        dc.l _uninit_excpt_             131: User-Defined Vector
        dc.l _uninit_excpt_             132: User-Defined Vector
        dc.l _uninit_excpt_             133: User-Defined Vector
        dc.l _uninit_excpt_             134: User-Defined Vector
        dc.l _uninit_excpt_             135: User-Defined Vector
        dc.l _uninit_excpt_             136: User-Defined Vector
        dc.l _uninit_excpt_             137: User-Defined Vector
150     dc.l _uninit_excpt_             138: User-Defined Vector
        dc.l _uninit_excpt_             139: User-Defined Vector
        dc.l _uninit_excpt_             140: User-Defined Vector
        dc.l _uninit_excpt_             141: User-Defined Vector
        dc.l _uninit_excpt_             142: User-Defined Vector
        dc.l _uninit_excpt_             143: User-Defined Vector
        dc.l _uninit_excpt_             144: User-Defined Vector
        dc.l _uninit_excpt_             145: User-Defined Vector
        dc.l _uninit_excpt_             146: User-Defined Vector
        dc.l _uninit_excpt_             147: User-Defined Vector
160     dc.l _uninit_excpt_             148: User-Defined Vector
        dc.l _uninit_excpt_             149: User-Defined Vector
        dc.l _uninit_excpt_             150: User-Defined Vector
        dc.l _uninit_excpt_             151: User-Defined Vector
        dc.l _uninit_excpt_             152: User-Defined Vector
        dc.l _uninit_excpt_             153: User-Defined Vector
        dc.l _uninit_excpt_             154: User-Defined Vector
        dc.l _uninit_excpt_             155: User-Defined Vector
        dc.l _uninit_excpt_             156: User-Defined Vector
        dc.l _uninit_excpt_             157: User-Defined Vector
170     dc.l _uninit_excpt_             158: User-Defined Vector
        dc.l _uninit_excpt_             159: User-Defined Vector
        dc.l _uninit_excpt_             160: User-Defined Vector
        dc.l _uninit_excpt_             161: User-Defined Vector
        dc.l _uninit_excpt_             162: User-Defined Vector
        dc.l _uninit_excpt_             163: User-Defined Vector
        dc.l _uninit_excpt_             164: User-Defined Vector
        dc.l _uninit_excpt_             165: User-Defined Vector
        dc.l _uninit_excpt_             166: User-Defined Vector
        dc.l _uninit_excpt_             167: User-Defined Vector
180     dc.l _uninit_excpt_             168: User-Defined Vector
        dc.l _uninit_excpt_             169: User-Defined Vector
        dc.l _uninit_excpt_             170: User-Defined Vector
        dc.l _uninit_excpt_             171: User-Defined Vector
        dc.l _uninit_excpt_             172: User-Defined Vector
        dc.l _uninit_excpt_             173: User-Defined Vector
```

```
        dc.l _uninit_excpt_          174: User-Defined Vector
        dc.l _uninit_excpt_          175: User-Defined Vector
        dc.l _uninit_excpt_          176: User-Defined Vector
        dc.l _uninit_excpt_          177: User-Defined Vector
190     dc.l _uninit_excpt_          178: User-Defined Vector
        dc.l _uninit_excpt_          179: User-Defined Vector
        dc.l _uninit_excpt_          180: User-Defined Vector
        dc.l _uninit_excpt_          181: User-Defined Vector
        dc.l _uninit_excpt_          182: User-Defined Vector
        dc.l _uninit_excpt_          183: User-Defined Vector
        dc.l _uninit_excpt_          184: User-Defined Vector
        dc.l _uninit_excpt_          185: User-Defined Vector
        dc.l _uninit_excpt_          186: User-Defined Vector
        dc.l _uninit_excpt_          187: User-Defined Vector
200     dc.l _uninit_excpt_          188: User-Defined Vector
        dc.l _uninit_excpt_          189: User-Defined Vector
        dc.l _uninit_excpt_          190: User-Defined Vector
        dc.l _uninit_excpt_          191: User-Defined Vector
        dc.l _uninit_excpt_          192: User-Defined Vector
        dc.l _uninit_excpt_          193: User-Defined Vector
        dc.l _uninit_excpt_          194: User-Defined Vector
        dc.l _uninit_excpt_          195: User-Defined Vector
        dc.l _uninit_excpt_          196: User-Defined Vector
        dc.l _uninit_excpt_          197: User-Defined Vector
210     dc.l _uninit_excpt_          198: User-Defined Vector
        dc.l _uninit_excpt_          199: User-Defined Vector
        dc.l _uninit_excpt_          200: User-Defined Vector
        dc.l _uninit_excpt_          201: User-Defined Vector
        dc.l _uninit_excpt_          202: User-Defined Vector
        dc.l _uninit_excpt_          203: User-Defined Vector
        dc.l _uninit_excpt_          204: User-Defined Vector
        dc.l _uninit_excpt_          205: User-Defined Vector
        dc.l _uninit_excpt_          206: User-Defined Vector
        dc.l _uninit_excpt_          207: User-Defined Vector
220     dc.l _uninit_excpt_          208: User-Defined Vector
        dc.l _uninit_excpt_          209: User-Defined Vector
        dc.l _uninit_excpt_          210: User-Defined Vector
        dc.l _uninit_excpt_          211: User-Defined Vector
        dc.l _uninit_excpt_          212: User-Defined Vector
        dc.l _uninit_excpt_          213: User-Defined Vector
        dc.l _uninit_excpt_          214: User-Defined Vector
        dc.l _uninit_excpt_          215: User-Defined Vector
        dc.l _uninit_excpt_          216: User-Defined Vector
        dc.l _uninit_excpt_          217: User-Defined Vector
230     dc.l _uninit_excpt_          218: User-Defined Vector
        dc.l _uninit_excpt_          219: User-Defined Vector
        dc.l _uninit_excpt_          220: User-Defined Vector
        dc.l _uninit_excpt_          221: User-Defined Vector
        dc.l _uninit_excpt_          222: User-Defined Vector
        dc.l _uninit_excpt_          223: User-Defined Vector
        dc.l _uninit_excpt_          224: User-Defined Vector
        dc.l _uninit_excpt_          225: User-Defined Vector
        dc.l _uninit_excpt_          226: User-Defined Vector
        dc.l _uninit_excpt_          227: User-Defined Vector
240     dc.l _uninit_excpt_          228: User-Defined Vector
        dc.l _uninit_excpt_          229: User-Defined Vector
        dc.l _uninit_excpt_          230: User-Defined Vector
        dc.l _uninit_excpt_          231: User-Defined Vector
        dc.l _uninit_excpt_          232: User-Defined Vector
        dc.l _uninit_excpt_          233: User-Defined Vector
        dc.l _uninit_excpt_          234: User-Defined Vector
        dc.l _uninit_excpt_          235: User-Defined Vector
        dc.l _uninit_excpt_          236: User-Defined Vector
        dc.l _uninit_excpt_          237: User-Defined Vector
250     dc.l _uninit_excpt_          238: User-Defined Vector
        dc.l _uninit_excpt_          239: User-Defined Vector
        dc.l _uninit_excpt_          240: User-Defined Vector
        dc.l _uninit_excpt_          241: User-Defined Vector
        dc.l _uninit_excpt_          242: User-Defined Vector
        dc.l _uninit_excpt_          243: User-Defined Vector
        dc.l _uninit_excpt_          244: User-Defined Vector
        dc.l _uninit_excpt_          245: User-Defined Vector
        dc.l _uninit_excpt_          246: User-Defined Vector
```

```
    dc.l _uninit_excpt_            247: User-Defined Vector
    dc.l _uninit_excpt_            248: User-Defined Vector
    dc.l _uninit_excpt_            249: User-Defined Vector
    dc.l _uninit_excpt_            250: User-Defined Vector
    dc.l _uninit_excpt_            251: User-Defined Vector
    dc.l _uninit_excpt_            252: User-Defined Vector
    dc.l _uninit_excpt_            253: User-Defined Vector
    dc.l _uninit_excpt_            254: User-Defined Vector
    dc.l _uninit_excpt_            255: User-Defined Vector

    opt l
```

B.5 Funktionen zur Datenausgabe am PC: mot2pc.a

```
*******************************************************
***             Routinen, um dem PC Daten zu senden  ***
*** Autor:       Tobias Schubert                      ***
*** EMail:       schubert@informatik.uni-freiburg.de  ***
*** Datum:       03.08.1999                           ***
*** Datei-Name:  mot2pc.a                             ***
*******************************************************

***
*** Die Daten werden ueber die beim Motorola vorhandene
*** RS232-Schnittstelle versendet, wobei hier aufgrund
*** der vorgegebenen Hardware der Kanal B verwendet wird.
*** Einstellungen bzgl. Baudrate, Anzahl Stop-Bits usw.
*** werden in "uart.a" vorgenommen.
***


*************************************************************
***
*** Name....: SEND_CHB_1LONG
*** Funktion: Sendet ein "long word" an den PC, d.h. 4 Byte.
*** Register: zu sendendes "word" muss in D4 stehen.
***           (Uebermittlung von "rechts" nach "links" aus D4)
***
SEND_CHB_1LONG
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    rts
***
*************************************************************


*************************************************************
***
*** Name    : SEND_CHB_4LONG
*** Funktion: Sendet 4 "long words" an den PC, insgesamt 16 Byte.
*** Register: Daten muessen in D4-D7 stehen, wobei mit D4 begonnen wird.
***           (Uebermittlung von "rechts" nach "links")
***
SEND_CHB_4LONG
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    move.l  d5,d4
```

```
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
60  bsr.l   SEND_CHB_BYTE
    move.l  d6,d4
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    move.l  d7,d4
70  bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    lsr.l   #8,d4
    bsr.l   SEND_CHB_BYTE
    rts
***
*************************************************************
80

*************************************************************
***
*** Name....: SEND_CHB_BYTE
*** Funktion: Sendet ein Byte an den PC.
*** Register: zu sendendes Byte muss in D4 an der niedrigsten
***           Position stehen.
***
SEND_CHB_BYTE
90  btst.b  #2,ser_srb             Warten bis PC empfangsbereit
    beq     SEND_CHB_BYTE
    move.b  d4,ser_tbb
    rts
***
*************************************************************


*************************************************************
***
100 *** Name....: SEND_STARS
*** Funktion: Sendet D7-viele Symbole "*" an den PC.
*** Register: In D7 an der niedrigsten Position.
***
SEND_STARS
    subi    #1,d7
    move.b  #$2A,d4                 = *
    bsr.l   SEND_CHB_BYTE
    cmp.b   #00,d7
    bne     SEND_STARS
110 rts
***
*************************************************************


*************************************************************
***
*** Name....: SEND_SPACE
*** Funktion: Sendet D7-viele Leerzeichen an den PC.
*** Register: In D7 an der niedrigsten Position.
120 ***
SEND_SPACE
    cmp.b   #00,d7
    bne     SEND_SPACE_GOON
    rts
SEND_SPACE_GOON
    subi    #1,d7
```

```
      move.b #$20,d4                  = 'Space'
      bsr.l  SEND_CHB_BYTE
      bra.l  SEND_SPACE
***
**********************************************************


**********************************************************
***
*** Name....: SEND_CR
*** Funktion: Sendet D7-viele "Carriage Return" an den PC.
*** Register: In D7 an der niedrigsten Position.
***
SEND_CR
      cmp.b  #00,d7
      bne    SEND_SPACE_CR
      rts
SEND_SPACE_CR
      subi   #1,d7
      move.b #$0D,d4                  = 'CR'
      bsr.l  SEND_CHB_BYTE
      bra.l  SEND_CR
***
**********************************************************


**********************************************************
***
*** Name....: SEND_UP_LINE
*** Funktion: Sendet eine "obere Begrenzungslinie" mit D7-vielen
***           waagrechten Segmenten.
*** Register: In D7 an der niedrigsten Position.
***
SEND_UP_LINE
      move.b #$C9,d4                  = "|-"
      bsr.l  SEND_CHB_BYTE
SEND_UP_LINE_LOOP
      subi   #1,d7
      move.b #$CD,d4                  = "-"
      bsr.l  SEND_CHB_BYTE
      cmp.b  #00,d7
      bne    SEND_UP_LINE_LOOP
      move.b #$BB,d4                  = "-|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                  = "CR"
      bsr.l  SEND_CHB_BYTE
      rts
***
**********************************************************


**********************************************************
***
*** Name....: SEND_MI_LINE
*** Funktion: Sendet eine "mittlere Begrenzungslinie" mit D7-vielen
***           waagrechten Segmenten.
*** Register: In D7 an der niedrigsten Position.
***
SEND_MI_LINE
      move.b #$CC,d4                  = "||="
      bsr.l  SEND_CHB_BYTE
SEND_MI_LINE_LOOP
      subi   #1,d7
      move.b #$CD,d4                  = "-"
      bsr.l  SEND_CHB_BYTE
      cmp.b  #00,d7
      bne    SEND_MI_LINE_LOOP
      move.b #$B9,d4                  = "=||"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                  = "CR"
      bsr.l  SEND_CHB_BYTE
      rts
***
```

```
200 ********************************************************


    ********************************************************
    ***
    *** Name....: SEND_LO_LINE
    *** Funktion: Sendet eine "untere Begrenzungslinie" mit D7-vielen
    ***           waagrechten Segmenten.
    *** Register: In D7 an der niedrigsten Position.
    ***
210 SEND_LO_LINE
      move.b #$C8,d4                = "|_"
      bsr.l  SEND_CHB_BYTE
    SEND_LO_LINE_LOOP
      subi   #1,d7
      move.b #$CD,d4                = "-"
      bsr.l  SEND_CHB_BYTE
      cmp.b  #00,d7
      bne    SEND_LO_LINE_LOOP
      move.b #$BC,d4                = "_|"
220   bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                = "CR"
      bsr.l  SEND_CHB_BYTE
      rts
    ***
    ********************************************************
```

B.6 Initialisierungen, Hauptschleife: `basics.a`

```
    ************************************************************************
    ***             Hauptschleife: Initialisieren, Programm-Uebergabe    ***
    *** Autor:      Tobias Schubert                                      ***
    *** EMail:      schubert@informatik.uni-freiburg.de                  ***
    *** Datum:      25.09.1999                                           ***
    *** Datei-Name: basics.a                                             ***
    ************************************************************************


 10 ************************************************************************
    ***                                                                  ***
    ***     Programm-Ablauf:                                             ***
    ***    ------------------                                            ***
    ***                                                                  ***
    ***     1.) Initialisieren der Interrupt-Quellen/-Routinen.          ***
    ***     2.) Serielle Schnittstelle initialisieren.                   ***
    ***     3.) "Begruessungsbildschirm"                                 ***
    ***     4.) Kontaktaufnahme zu PIC-Prozessoren / Ergebnis des        ***
    ***         Speichertestes entgegennehmen.                           ***
 20 ***     5.) Die (verschiedenen) Programme an die PIC-Prozessoren     ***
    ***         empfangen und verteilen.                                 ***
    ***     6.) Daten-"Breite" der Chromosome festlegen.                 ***
    ***     7.) Topologie des Datenaustausches vornehmen.                ***
    ***     8.) Zwischenergebnisse gemaess Topologie verteilen.          ***
    ***     9.) Beste Ergebnisse am Bildschirm praesentieren.            ***
    ***    10.) Neustart?                                                ***
    ***                                                                  ***
    ************************************************************************

 30
    ************************************************************************
    ***                                                                  ***
    ***     Definition: "Mode"-Register des MC68340                      ***
    ***    ------------------------------------                          ***
    ***                                                                  ***
    ***     Wert:                   Bedeutung:                           ***
    ***    ------------------------------------------------------------  ***
    ***     0000 0000 = 000         Warten auf Speichertest PIC0         ***
```

```
***      0000 0001 = 001          Warten auf Speichertest PIC1          ***
***      0000 0010 = 002          Warten auf Speichertest PIC2          ***
***      0000 0011 = 003          Warten auf Speichertest PIC3          ***
***      0000 0100 = 004          Verteilen der PIC-Programme           ***
***      0000 0101 = 005          Grundmodus: Chromosome verteilen      ***
***      0000 1010 = 010          Anfrage von PIC0: Empfang d. Daten    ***
***      0000 1011 = 011          Anfrage von PIC1: Empfang d. Daten    ***
***      0000 1100 = 012          Anfrage von PIC2: Empfang d. Daten    ***
***      0000 1101 = 013          Anfrage von PIC3: Empfang d. Daten    ***
***                                                                     ***
*************************************************************************

BASICS:
  move.b #$60,avr                * Autovektor fuer IRQ-Level 5 und 6
  bclr   #12,simcr               * FIRQ = 0
  move.b #$FF,pparb              * PORTB festlegen: 4xIRQ + 4xCS
  move.b #$00,ppara1             * PORTA als "lack" nutzen
  move.b #$FF,ppara2


  bsr.l  SETUP_UART              * Serielle Schnittstelle initialisieren

  bsr.l  START_MESSAGE           * "Begruessungs-Bildschirm" mit Name, ...

  bsr.l  SRAM_TEST_START         * Testen der PICs und deren externem RAM

  bsr.l  GET_PRG                 * Entgegennehmen der (vers.) PIC-Programme

  bsr.l  GET_DATA_SIZE           * "Datenbreite" der auszutauschenden Daten

  bsr.l  INIT_REGISTER           * benoetigte Register auf "Null" setzen

  bsr.l  PIC_START_MESS          * "PICs beginnen mit der Abarbeitung"


*** PLD-Latches mit 0-Vektor initialisieren, Programm-Abarbeitung starten
  move.b #$05,mode               * Betriebsmodus = 5, vgl. obige Tabelle
  bsr.l  CLEAR_PLD
  bsr.l  CLEAR_PLD               * PIC-Zufallswerte uebermitteln

*** Hauptschleife: Anfragen auswerten, Daten verteilen/speichern
MAIN_LOOP

*** nicht vorhandene PICs mit $FF als "fertig" deklarieren
  btst.b #$00,results
  beq    INIT_PIC1_CONTROL       * PIC0 ist korrekt ansprechbar
  move.b #$FF,control0
INIT_PIC1_CONTROL
  btst.b #$01,results
  beq    INIT_PIC2_CONTROL       * PIC1 ist korrekt ansprechbar
  move.b #$FF,control1
INIT_PIC2_CONTROL
  btst.b #$02,results
  beq    INIT_PIC3_CONTROL       * PIC2 ist korrekt ansprechbar
  move.b #$FF,control2
INIT_PIC3_CONTROL
  btst.b #$03,results
  beq    MAIN_TST_PIC0           * PIC3 ist korrekt ansprechbar
  move.b #$FF,control3

*** Tests, ob alle PICs Ihre naechste "Iteration" erreicht haben
MAIN_TST_PIC0
  cmp.b  #$FE,control0           * "Arbeitet" PIC0 noch?
  beq    MAIN_TST_PIC1
  cmp.b  #$FF,control0
  beq    MAIN_TST_PIC1
  bra.l  MAIN_LOOP

MAIN_TST_PIC1
  cmp.b  #$FE,control1           * "Arbeitet" PIC1 noch?
  beq    MAIN_TST_PIC2
  cmp.b  #$FF,control1
  beq    MAIN_TST_PIC2
```

```
      bra.l  MAIN_LOOP

    MAIN_TST_PIC2
      cmp.b  #$FE,control2          * "Arbeitet" PIC2 noch?
      beq    MAIN_TST_PIC3
      cmp.b  #$FF,control2
      beq    MAIN_TST_PIC3
      bra.l  MAIN_LOOP
120
    MAIN_TST_PIC3
      cmp.b  #$FE,control3          * "Arbeitet" PIC3 noch?
      beq    CHANGE_DATA
      cmp.b  #$FF,control3
      beq    CHANGE_DATA
      bra.l  MAIN_LOOP

    *** Datenaustausch fuer die naechste "Iteration
    CHANGE_DATA
130   cmp.b  #$FE,control0
      bne    CHANGE_DATA_PIC1
      move.b #$0A,mode              * Modus = 10 (vgl. obige Tabelle)
      bsr.l  CHANGE_RESULTS_PIC0

    CHANGE_DATA_PIC1
      cmp.b  #$FE,control1
      bne    CHANGE_DATA_PIC2
      move.b #$0B,mode              * Modus = 11 (vgl. obige Tabelle)
      bsr.l  CHANGE_RESULTS_PIC1
140
    CHANGE_DATA_PIC2
      cmp.b  #$FE,control2
      bne    CHANGE_DATA_PIC3
      move.b #$0C,mode              * Modus = 12 (vgl. obige Tabelle)
      bsr.l  CHANGE_RESULTS_PIC2

    CHANGE_DATA_PIC3
      cmp.b  #$FE,control3
      bne    GETTING_RESULTS_FROM_PIC0
150   move.b #$0D,mode              * Modus = 13 (vgl. obige Tabelle)
      bsr.l  CHANGE_RESULTS_PIC3

    *** Bei Beendigung ($FF) die besten Ergebnisse abfragen
    GETTING_RESULTS_FROM_PIC0
      cmp.b  #$FF,control0
      bne    GETTING_RESULTS_FROM_PIC1
      btst.b #$00,results
      bne    GETTING_RESULTS_FROM_PIC1
      move.b #$0A,mode              * Modus = 10 (vgl. obige Tabelle)
160   bsr.l  GET_RES_PIC0
      move.b #$05,mode

    GETTING_RESULTS_FROM_PIC1
      cmp.b  #$FF,control1
      bne    GETTING_RESULTS_FROM_PIC2
      btst.b #$01,results
      bne    GETTING_RESULTS_FROM_PIC2
      move.b #$0B,mode              * Modus = 11 (vgl. obige Tabelle)
      bsr.l  GET_RES_PIC1
170   move.b #$05,mode

    GETTING_RESULTS_FROM_PIC2
      cmp.b  #$FF,control2
      bne    GETTING_RESULTS_FROM_PIC3
      btst.b #$02,results
      bne    GETTING_RESULTS_FROM_PIC3
      move.b #$0C,mode              * Modus = 12 (vgl. obige Tabelle)
      bsr.l  GET_RES_PIC2
      move.b #$05,mode
180
    GETTING_RESULTS_FROM_PIC3
      cmp.b  #$FF,control3
      bne    LOOP_END_TST
      btst.b #$03,results
```

```
      bne     LOOP_END_TST
      move.b #$0D,mode              * Modus = 13 (vgl. obige Tabelle)
      bsr.l   GET_RES_PIC3
      move.b #$05,mode

190 *** Test auf Ende der Hauptschleife
    LOOP_END_TST
      cmp.b   #$0F,results
      beq     SHOW_RESULTS          * Alle PICs sind fertig

    *** Evtl. Tastendruck zur Topologie-Aenderung auswerten.
      btst.b #0,ser_srb             * Bit0=1 -> neue empfangene Zeichen
      beq     RESET_CONTROL_SIGNALS * Bit0=0 -> keine (neuen) Daten/Signale
      move.b ser_rbb,d6             * ser_rbb = "Receiver Buffer B"
      bsr.l   GET_TOPOLOGY
200
    *** Startsignal ($00) fuer die naechste Iteration an die PICs
    *** verschicken und Steuersignale zuruecksetzen auf "0".
    RESET_CONTROL_SIGNALS
      move.b #$05,mode
      btst.b #$00,results
      bne     CALC_BIT1
      subi.b #01,random
      moveq   #00,d3                * D3=1 signalisiert Bestaetigung durch PIC
      move.b random,pic0            * lasse PLD durch PIC0 auf "0" setzen
210 CLEAR_PIC0
      cmp.l   #$00,d3
      beq     CLEAR_PIC0

    CALC_BIT1
      btst.b #$01,results
      bne     CALC_BIT2
      subi.b #01,random
      moveq   #00,d3                * D3=1 signalisiert Bestaetigung durch PIC
      move.b random,pic1            * lasse PLD durch PIC1 auf "0" zuruecksetzen
220 CLEAR_PIC1
      cmp.l   #$00,d3
      beq     CLEAR_PIC1

    CALC_BIT2
      btst.b #$02,results
      bne     CALC_BIT3
      subi.b #01,random
      moveq   #00,d3                * D3=1 signalisiert Bestaetigung durch PIC
      move.b random,pic2            * lasse PLD durch PIC2 auf "0" zuruecksetzen
230 CLEAR_PIC2
      cmp.l   #$00,d3
      beq     CLEAR_PIC2

    CALC_BIT3
      btst.b #$03,results
      bne     MAIN_LOOP
      subi.b #01,random
      moveq   #00,d3                * D3=1 signalisiert Bestaetigung durch PIC
      move.b random,pic3            * lasse PLD durch PIC3 auf "0" zuruecksetzen
240 CLEAR_PIC3
      cmp.l   #$00,d3
      beq     CLEAR_PIC3
      bra.l   MAIN_LOOP
    *** Ende der Hauptschleife


    *** Neustart?
    SHOW_RESULTS
      btst.b #$00,correct_pics
250   bne     SHOW_RESULTS_FROM_PIC1
      bsr.l   SHOW_PIC0_RES
    SHOW_RESULTS_FROM_PIC1
      btst.b #$01,correct_pics
      bne     SHOW_RESULTS_FROM_PIC2
      bsr.l   SHOW_PIC1_RES
    SHOW_RESULTS_FROM_PIC2
      btst.b #$02,correct_pics
```

```
    bne     SHOW_RESULTS_FROM_PIC3
    bsr.l   SHOW_PIC2_RES
SHOW_RESULTS_FROM_PIC3
    btst.b #$03,correct_pics
    bne     RESTART_TST
    bsr.l   SHOW_PIC3_RES

RESTART_TST
    bsr.l   RESTART_MESS
    cmp.b   #$6A,d6                 * (6A) ASCII = 'j'
    beq     ga_start                * Neustart

*** kein Neustart, Programmende
    move.w #$2700,sr                * Interrupt Level = 7, Supervisor Mode
    rts


*** Include-Dateien:
    include asm/uart.a              * serielle Schnittstelle initialisieren
    include asm/pictest.a           * Test des externen PIC-Speichers
    include asm/getprg.a            * PIC-Anwendungen vom PC entgegennehmen
    include asm/sendprg.a           * PIC-Anwendungen an diese verschicken
    include asm/datasize.a          * Daten-/Chromosomengroesse
    include asm/topology.a          * Datenaustauschstrategie festlegen
    include asm/clearpld.a          * "Reset" bei den PLDs hervorrufen
    include asm/initreg.a           * Register initialisieren
    include asm/convert.a           * verschiedene Konvertierungsroutinen
    include asm/messages.a          * Alle Textausgaben an den PC
    include asm/irq.a               * Bearbeiten eingehender Interrupts
    include asm/change.a            * Datenaustausch zwischen den PICs steuern
    include asm/getres.a            * Empfang der besten PIC-Resultate
    include asm/showres.a           * Anzeigen der besten Resultate
```

B.7 Initialisierung der RS232-Schnittstelle: `uart.a`

```
*************************************************************************
***             Initialisieren der seriellen Schnittstelle (Kanal B)  ***
*** Autor:       Tobias Schubert                                       ***
*** EMail:       schubert@informatik.uni-freiburg.de                   ***
*** Datum:       29.09.1999                                            ***
*** Datei-Name:  uart.a                                                ***
*************************************************************************


*************************************************************************
***
*** Name....:  SETUP_UART
*** Funktion: Legt Kanal B fest auf 38400 Baud, keine Paritaet, 8 Datenbits
***            und 1 Stop-Bit (analog zu FAX "AN899" von Motorola).
*** Register: Keines der Register D0-D7 wird benoetigt oder veraendert.
***
SETUP_UART
    move.b  #$80,ser_acr            BRG Set 2
    move.b  #$DD,ser_csrb           B: RX & TX 38400 Baud
    move.b  #$93,ser_mr1b           B: RX-RTS, keine Paritaet, 8 Charakter
    move.b  #$07,ser_mr2b           B: Modus=Normal, 1 Stop-Bit
    rts
***
*************************************************************************


*************************************************************************
***
*** Name....:  RECEIVERB_LOOP
*** Funktion: Wartet auf ein Zeichen des Benutzers.
***            In einer "Neben-Schleife" wird eine Zufallszahl üfr
```

```
***            die PICs bestimmt.
*** Register: D6 enthaelt das empfangene Zeichen.
***
RECEIVERB_LOOP
  addi.b #01,random              Zufallswert inkrementieren
  btst.b #0,ser_srb              Bit0 = 0 --> keine (neuen) Daten
  beq    RECEIVERB_LOOP          Bit0 = 1 --> neue empfangene Zeichen
  move.b ser_rbb,d6              ser_rbb = "Receiver Buffer B"
  rts
***
*******************************************************************************
```

B.8 Kontaktaufnahme zu den Prozessoren: pictest.a

```
*********************************************************
***             Test der PIC-Kommunikation + RAM-Test  ***
*** Autor:      Tobias Schubert                        ***
*** EMail:      schubert@informatik.uni-freiburg.de    ***
*** Datum:      29.07.1999                             ***
*** Datei-Name: pictest.a                              ***
*********************************************************

*******************************************************************************
***                                                                         ***
***             Kodierung des PIC-Status:                                   ***
***            ---------------------------                                  ***
***                                                                         ***
***     <id1, id0, k, b4, b3, b2, b1, s> mit folgender Bedeutung:           ***
***                                                                         ***
***             id1 id0: 00 = PIC0, 01 = PIC1, 02 = PIC2, 03 = PIC3         ***
***                 k=1: kein Kontakt zwischen PIC und Motorola mgl.        ***
***             b4 - b1: nicht genutzt                                      ***
***                 s=1: Fehler beim Beschreiben des externen PIC-RAM       ***
***                                                                         ***
*******************************************************************************


*******************************************************************************
***
*** Name....: SRAM_TEST_START
*** Funktion: Testet Kontakt zu den PICs und gibt Meldung aus bzgl. des
***           jeweiligen externen PIC-Speichers.
*** Register: D4-D7 fuer Ausgabe an den PC,
***           D0-D3 fuer Speicherung des PIC-Status (PIC0 --> D0, ...)
***
SRAM_TEST_START
  moveq  #02,d7
  bsr.l  SEND_CR
  moveq  #10,d7
  bsr.l  SEND_SPACE              * d7-viele Leerzeichen
  moveq  #53,d7
  bsr.l  SEND_UP_LINE            * obere "Begrenzung" eines Rahmens
  moveq  #10,d7
  bsr.l  SEND_SPACE              * d7-viele Leerzeichen
  move.b #$BA,d4                 * = "|"
  bsr.l  SEND_CHB_BYTE
  moveq  #18,d7
  bsr.l  SEND_SPACE              * d7-viele Leerzeichen
  move.l #$2D434950,d4           * "PIC-"
  move.l #$7A6F7250,d5           * "Proz"
  move.l #$6F737365,d6           * "esso"
  move.l #$206E6572,d7           * "ren "
  bsr.l  SEND_CHB_4LONG
  moveq  #19,d7
  bsr.l  SEND_SPACE              * d7-viele Leerzeichen
  move.b #$BA,d4                 * = "|"
```

```
      bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4
      bsr.l   SEND_CHB_BYTE
      moveq   #10,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      moveq   #53,d7
 60   bsr.l   SEND_MI_LINE            * mittlere "Begrenzung" eines Rahmens
      moveq   #10,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      moveq   #18,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.l  #$746E6F4B,d4           * "Kont"
      bsr.l   SEND_CHB_1LONG
      move.l  #$3A746B61,d4           * "akt:"
 70   bsr.l   SEND_CHB_1LONG
      moveq   #10,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.l  #$2E747845,d4           * "ext."
      bsr.l   SEND_CHB_1LONG
      moveq   #01,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.l  #$69657053,d4           * "Spei"
      bsr.l   SEND_CHB_1LONG
      move.l  #$72656863,d4           * "cher"
 80   bsr.l   SEND_CHB_1LONG
      move.b  #$3A,d4                 * = ":"
      bsr.l   SEND_CHB_BYTE
      moveq   #03,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4
      bsr.l   SEND_CHB_BYTE
      moveq   #10,d7
 90   bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      moveq   #53,d7
      bsr.l   SEND_MI_LINE            * mittlere "Begrenzung" eines Rahmens


    *** Variablen-/Register-Initialisierungen:
      move.b  #$00,correct_pics       * Bit x=0 --> PICx kann genutzt werden
      moveq   #00,D0                  * Resultat des Speichertests von PIC0
      moveq   #00,D1                  * Resultat des Speichertests von PIC1
      moveq   #00,D2                  * Resultat des Speichertests von PIC2
100   moveq   #00,D3                  * Resultat des Speichertests von PIC3


    *** Teste PIC0:
      move.b  #$00,mode               * Modus = 000 (vgl. "basics.a")
      move.w  #$A000,D4               * Innerhalb von (A000)h vielen Iterationen
      move.b  #$01,pic0               * muss PIC0 antworten.
    WAIT_PIC0
      subi.w  #01,D4
      cmp.w   #00,D4
110   beq     PIC0_NO_CONTACT         * Zeitlimit erreicht
      cmp.b   #01,mode                * PIC0 antwortet (vgl. "picx_interrupt")
      bne     WAIT_PIC0               * Auf Antwort warten
      bra.l   SRAM_PIC0_MESSAGE       * Kein Fehler
    PIC0_NO_CONTACT
      bset    #05,D0                  * K=1, PIC0 nicht ansprechbar
      bset    #00,correct_pics        * bit0=1 -> PIC0 kann nicht genutzt werden
      addi.b  #01,mode

    *** Ergebnis von PIC0 ausgeben:
120 SRAM_PIC0_MESSAGE
      moveq   #10,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      moveq   #07,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
```

```
      move.l #$30434950,d4          * "PIC0"
      bsr.l  SEND_CHB_1LONG
      btst.b #05,D0
130   bne    PIC0_CONTACT_ERROR     * Kein Kontakt zu PIC0
      moveq  #08,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$20614A20,d4          * " Ja "
      bsr.l  SEND_CHB_1LONG
      btst.b #00,D0
      bne    PIC0_SRAM_ERROR        * Speicherfehler bei PIC0
      moveq  #18,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$204B4F20,d4          * " OK "
140   bsr.l  SEND_CHB_1LONG
      moveq  #08,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE
      bra.l  SRAM_PIC1_CONTACT      * Speichertest bei PIC1 abfragen

    PIC0_SRAM_ERROR
150   bset   #00,correct_pics       * bit0=1 -> PIC0 kann nicht genutzt werden
      moveq  #17,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$6C686546,d4          * "Fehl"
      bsr.l  SEND_CHB_1LONG
      move.l #$20207265,d4          * "er  "
      bsr.l  SEND_CHB_1LONG
      moveq  #05,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.b #$BA,d4                * = "|"
160   bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE
      bra.l  SRAM_PIC1_CONTACT      * Speichertest bei PIC1 abfragen

    PIC0_CONTACT_ERROR
      moveq  #08,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$6E69654E,d4          * "Nein"
      bsr.l  SEND_CHB_1LONG
170   moveq  #17,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$2D2D2D2D,d4          * "----"
      bsr.l  SEND_CHB_1LONG
      move.l #$20202D2D,d4          * "--  "
      bsr.l  SEND_CHB_1LONG
      moveq  #05,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
180   move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE


    *** Teste PIC1:
    SRAM_PIC1_CONTACT
      move.b #$01,mode              * Modus = 001 (vgl. "basics.a")
      move.w #$A000,D4              * Innerhalb von (A000)h vielen Iterationen
      move.b #$01,pic1              * muss PIC1 antworten.
    WAIT_PIC1
190   subi.w #01,D4
      cmp.w  #00,D4
      beq    PIC1_NO_CONTACT        * Zeitlimit erreicht
      cmp.b  #02,mode               * PIC1 antwortet (vgl. "picx_interrupt")
      bne    WAIT_PIC1              * Auf Antwort warten
      bra.l  SRAM_PIC1_MESSAGE      * Kein Fehler
    PIC1_NO_CONTACT
      bset   #05,D1                 * K=1, PIC1 nicht ansprechbar
      bset   #01,correct_pics       * bit1=1 -> PIC1 kann nicht genutzt werden
      addi.b #01,mode
```

```
200
    *** Ergebnis von PIC1 ausgeben:
    SRAM_PIC1_MESSAGE
      moveq  #10,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
      moveq  #07,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.l #$31434950,d4           * "PIC1"
210   bsr.l  SEND_CHB_1LONG
      btst.b #05,D1
      bne    PIC1_CONTACT_ERROR      * Kein Kontakt zu PIC1
      moveq  #08,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.l #$20614A20,d4           * " Ja "
      bsr.l  SEND_CHB_1LONG
      btst.b #00,D1
      bne    PIC1_SRAM_ERROR         * Speicherfehler bei PIC1
      moveq  #18,d7
220   bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.l #$204B4F20,d4           * " OK "
      bsr.l  SEND_CHB_1LONG
      moveq  #08,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                 * = "CR"
      bsr.l  SEND_CHB_BYTE
      bra.l  SRAM_PIC2_CONTACT       * Speichertest bei PIC2 abfragen
230
    PIC1_SRAM_ERROR
      bset   #01,correct_pics        * bit1=1 -> PIC1 kann nicht genutzt werden
      moveq  #17,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.l #$6C686546,d4           * "Fehl"
      bsr.l  SEND_CHB_1LONG
      move.l #$20207265,d4           * "er  "
      bsr.l  SEND_CHB_1LONG
      moveq  #05,d7
240   bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                 * = "CR"
      bsr.l  SEND_CHB_BYTE
      bra.l  SRAM_PIC2_CONTACT       * Speichertest bei PIC2 abfragen

    PIC1_CONTACT_ERROR
      moveq  #08,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
250   move.l #$6E69654E,d4           * "Nein"
      bsr.l  SEND_CHB_1LONG
      moveq  #17,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.l #$2D2D2D2D,d4           * "----"
      bsr.l  SEND_CHB_1LONG
      move.l #$20202D2D,d4           * "--  "
      bsr.l  SEND_CHB_1LONG
      moveq  #05,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
260   move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                 * = "CR"
      bsr.l  SEND_CHB_BYTE


    *** Teste PIC2:
    SRAM_PIC2_CONTACT
      move.b #$02,mode               * Modus = 002 (vgl. "basics.a")
      move.w #$A000,D4               * Innerhalb von (A000)h vielen Iterationen
270   move.b #$01,pic2               * muss PIC2 antworten.
    WAIT_PIC2
      subi.w #01,D4
```

```
      cmp.w   #00,D4
      beq     PIC2_NO_CONTACT         * Zeitlimit erreicht
      cmp.b   #03,mode                * PIC2 antwortet (vgl. "picx_interrupt")
      bne     WAIT_PIC2               * Auf Antwort warten
      bra.l   SRAM_PIC2_MESSAGE       * Kein Fehler
    PIC2_NO_CONTACT
      bset    #05,D2                  * K=1, PIC2 nicht ansprechbar
280   bset    #02,correct_pics        * bit2=1 -> PIC2 kann nicht genutzt werden
      addi.b  #01,mode

    *** Ergebnis von PIC2 ausgeben:
    SRAM_PIC2_MESSAGE
      moveq   #10,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      moveq   #07,d7
290   bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.l  #$32434950,d4           * "PIC2"
      bsr.l   SEND_CHB_1LONG
      btst.b  #05,D2
      bne     PIC2_CONTACT_ERROR      * Kein Kontakt zu PIC2
      moveq   #08,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.l  #$20614A20,d4           * " Ja "
      bsr.l   SEND_CHB_1LONG
      btst.b  #00,D2
300   bne     PIC2_SRAM_ERROR         * Speicherfehler bei PIC2
      moveq   #18,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.l  #$204B4F20,d4           * " OK "
      bsr.l   SEND_CHB_1LONG
      moveq   #08,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4                 * = "CR"
310   bsr.l   SEND_CHB_BYTE
      bra.l   SRAM_PIC3_CONTACT       * Speichertest bei PIC3 abfragen

    PIC2_SRAM_ERROR
      bset    #02,correct_pics        * bit2=1 -> PIC2 kann nicht genutzt werden
      moveq   #17,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.l  #$6C686546,d4           * "Fehl"
      bsr.l   SEND_CHB_1LONG
      move.l  #$20207265,d4           * "er  "
320   bsr.l   SEND_CHB_1LONG
      moveq   #05,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE
      bra.l   SRAM_PIC3_CONTACT       * Speichertest bei PIC3 abfragen

    PIC2_CONTACT_ERROR
330   moveq   #08,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.l  #$6E69654E,d4           * "Nein"
      bsr.l   SEND_CHB_1LONG
      moveq   #17,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.l  #$2D2D2D2D,d4           * "----"
      bsr.l   SEND_CHB_1LONG
      move.l  #$20202D2D,d4           * "--  "
      bsr.l   SEND_CHB_1LONG
340   moveq   #05,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE
```

```
    *** Teste PIC3:
    SRAM_PIC3_CONTACT
350   move.b #$03,mode              * Modus = 003 (vgl. "basics.a")
      move.w #$A000,D4              * Innerhalb von (A000)h vielen Iterationen
      move.b #$01,pic3              * muss PIC3 antworten.
    WAIT_PIC3
      subi.w #01,D4
      cmp.w  #00,D4
      beq    PIC3_NO_CONTACT        * Zeitlimit erreicht
      cmp.b  #04,mode               * PIC3 antwortet (vgl. "picx_interrupt")
      bne    WAIT_PIC3              * Auf Antwort warten
      bra.l  SRAM_PIC3_MESSAGE      * Kein Fehler
360 PIC3_NO_CONTACT
      bset   #05,D3                 * K=1, PIC3 nicht ansprechbar
      bset   #03,correct_pics       * bit3=1 -> PIC3 kann nicht genutzt werden
      addi.b #01,mode

    *** Ergebnis von PIC3 ausgeben:
    SRAM_PIC3_MESSAGE
      moveq  #10,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.b #$BA,d4                * = "|"
370   bsr.l  SEND_CHB_BYTE
      moveq  #07,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$33434950,d4          * "PIC3"
      bsr.l  SEND_CHB_1LONG
      btst.b #05,D3
      bne    PIC3_CONTACT_ERROR     * Kein Kontakt zu PIC3
      moveq  #08,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$20614A20,d4          * " Ja "
380   bsr.l  SEND_CHB_1LONG
      btst.b #00,D3
      bne    PIC3_SRAM_ERROR        * Speicherfehler bei PIC3
      moveq  #18,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$204B4F20,d4          * " OK "
      bsr.l  SEND_CHB_1LONG
      moveq  #08,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.b #$BA,d4                * = "|"
390   bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE
      bra.l  SRAM_PICx_READY        * Abfrage beendet

    PIC3_SRAM_ERROR
      bset   #03,correct_pics       * bit3=1 -> PIC3 kann nicht genutzt werden
      moveq  #17,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$6C686546,d4          * "Fehl"
400   bsr.l  SEND_CHB_1LONG
      move.l #$20207265,d4          * "er  "
      bsr.l  SEND_CHB_1LONG
      moveq  #05,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE
      bra.l  SRAM_PICx_READY        * Abfrage beendet
410
    PIC3_CONTACT_ERROR
      moveq  #08,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$6E69654E,d4          * "Nein"
      bsr.l  SEND_CHB_1LONG
      moveq  #17,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$2D2D2D2D,d4          * "----"
```

```
    bsr.l   SEND_CHB_1LONG
420 move.l  #$20202D2D,d4           * "--  "
    bsr.l   SEND_CHB_1LONG
    moveq   #05,d7
    bsr.l   SEND_SPACE              * d7-viele Leerzeichen
    move.b  #$BA,d4                 * = "|"
    bsr.l   SEND_CHB_BYTE
    move.b  #$0D,d4                 * = "CR"
    bsr.l   SEND_CHB_BYTE


430 SRAM_PICx_READY
    moveq   #10,d7
    bsr.l   SEND_SPACE              * d7-viele Leerzeichen
    moveq   #53,d7
    bsr.l   SEND_LO_LINE            * untere "Begrenzung" eines Rahmens
    bsr.l   RECEIVERB_LOOP          * Auf Signal warten
    rts
***
*****************************************************************************
```

B.9 Programmempfang vom Anwender: getprg.a

```
*************************************************************
***             PIC-Programme empfangen                   ***
*** Autor:      Tobias Schubert                           ***
*** EMail:      schubert@informatik.uni-freiburg.de       ***
*** Datum:      03.08.1999                                ***
*** Datei-Name: getprg.a                                  ***
*************************************************************


 10 *************************************************************
***
*** Name....: GET_PRG
*** Funktion: Empfaengt die vom Benutzer gewaehlte Anzahl an
***           PIC-Programmen und verteilt sie entsprechend.
*** Register: D4, D6 zum Datenempfang
***
GET_PRG
    cmp.b   #15,correct_pics        * kein PIC ansprechbar (alle Bit gesetzt)
    bne     ONLY_ONE_PIC
 20 bsr.l   END_OF_PROGRAM
    bgnd

ONLY_ONE_PIC
    cmp.b   #07,correct_pics        * nur PIC3 vorhanden
    beq     GET_ONLY_ONE_PRG
    cmp.b   #11,correct_pics        * nur PIC2 vorhanden
    beq     GET_ONLY_ONE_PRG
    cmp.b   #13,correct_pics        * nur PIC1 vorhanden
    beq     GET_ONLY_ONE_PRG
 30 cmp.b   #14,correct_pics        * nur PIC0 vorhanden
    beq     GET_ONLY_ONE_PRG

TWO_OR_MORE_PICS
    bsr.l   HOW_MANY_PRGS_MESS      * "Mehrere Programme?"

    bsr.l   RECEIVERB_LOOP          * Auf Symbol warten
    cmp.b   #$6A,d6                 * (6A)hex = 'j'
    bne     TWO_OR_MORE_PICS_ONLY_ONE_PRG

 40 btst.b  #$00,correct_pics       * Bit 0 = 0 --> Programm an PIC0
    bne     SINGLE_PRG_FOR_PIC1
    bsr.l   GET_PRG_MESSAGE         * Aufforderung zur Programmuebergabe
    moveq   #00,d2
    bsr.l   PRG_TO_PICx_MESS
```

```
      move.l #var_start+$100,A0      * StartAdresse ist "varstart"+$100
      move.l A0,A1                   * A1 ist StartAdresse des Programmes
      bsr.l  GETTING_DATA            * Programm-Empfang ueber RS232-Schnittstelle
      move.l A0,A2                   * A2 ist EndAdresse des Programmes
      move.l A1,A0                   * A0 StartAdresse, Programm --> PIC
 50   bsr.l  SEND_PRG_TO_PIC0

    SINGLE_PRG_FOR_PIC1
      btst.b #$01,correct_pics       * Bit 1 = 0 --> Programm an PIC1
      bne    SINGLE_PRG_FOR_PIC2
      bsr.l  GET_PRG_MESSAGE         * Aufforderung zur Programmuebergabe
      moveq  #01,d2
      bsr.l  PRG_TO_PICx_MESS
      move.l #var_start+$100,A0      * StartAdresse ist "varstart"+$100
      move.l A0,A1                   * A1 ist StartAdresse des Programmes
 60   bsr.l  GETTING_DATA            * Programm-Empfang ueber RS232-Schnittstelle
      move.l A0,A2                   * A2 ist EndAdresse des Programmes
      move.l A1,A0                   * A0 StartAdresse, Programm --> PIC
      bsr.l  SEND_PRG_TO_PIC1

    SINGLE_PRG_FOR_PIC2
      btst.b #$02,correct_pics       * Bit 2 = 0 --> Programm an PIC2
      bne    SINGLE_PRG_FOR_PIC3
      bsr.l  GET_PRG_MESSAGE         * Aufforderung zur Programmuebergabe
      moveq  #02,d2
 70   bsr.l  PRG_TO_PICx_MESS
      move.l #var_start+$100,A0      * StartAdresse ist "varstart"+$100
      move.l A0,A1                   * A1 ist StartAdresse des Programmes
      bsr.l  GETTING_DATA            * Programm-Empfang ueber RS232-Schnittstelle
      move.l A0,A2                   * A2 ist EndAdresse des Programmes
      move.l A1,A0                   * A0 StartAdresse, Programm --> PIC
      bsr.l  SEND_PRG_TO_PIC2

    SINGLE_PRG_FOR_PIC3
      btst.b #$03,correct_pics       * Bit 3 = 0 --> Programm an PIC3
 80   bne    SINGLE_PRG_RDY
      bsr.l  GET_PRG_MESSAGE         * Aufforderung zur Programmuebergabe
      moveq  #03,d2
      bsr.l  PRG_TO_PICx_MESS
      move.l #var_start+$100,A0      * StartAdresse ist "varstart"+$100
      move.l A0,A1                   * A1 ist StartAdresse des Programmes
      bsr.l  GETTING_DATA            * Programm-Empfang ueber RS232-Schnittstelle
      move.l A0,A2                   * A2 ist EndAdresse des Programmes
      move.l A1,A0                   * A0 StartAdresse, Programm --> PIC
      bsr.l  SEND_PRG_TO_PIC3
 90
    SINGLE_PRG_RDY
      rts
    ***
    ********************************************************


    ********************************************************
    ***
    *** Name....: TWO_OR_MORE_PICS_ONLY_ONE_PRG
100 *** Funktion: Empfaengt und uebertraegt ein Programm fuer alle PICs
    *** Register: A0 - A2 fuer Start-/End-Adressen der Programme
    ***
    TWO_OR_MORE_PICS_ONLY_ONE_PRG
      move.l #var_start+$100,A0      * StartAdresse ist "varstart"+$100
      move.l A0,A1                   * A1 ist StartAdresse des Programmes
      bsr.l  GET_PRG_MESSAGE         * Aufforderung zur Programmuebergabe
      bsr.l  GETTING_DATA            * Programm-Empfang ueber RS232-Schnittstelle
      move.l A0,A2                   * A2 ist EndAdresse des Programmes

110   btst.b #$00,correct_pics       * Bit 0 = 0 --> Programm an PIC0
      bne    ONLY_ONE_PRG_FOR_PIC1
      move.l A1,A0                   * A0 StartAdresse, Programm --> PIC
      bsr.l  SEND_PRG_TO_PIC0

    ONLY_ONE_PRG_FOR_PIC1
      btst.b #$01,correct_pics       * Bit 1 = 0 --> Programm an PIC1
      bne    ONLY_ONE_PRG_FOR_PIC2
```

```
      move.l A1,A0                    * A0 StartAdresse, Programm --> PIC
      bsr.l  SEND_PRG_TO_PIC1
120
    ONLY_ONE_PRG_FOR_PIC2
      btst.b #$02,correct_pics        * Bit 2 = 0 --> Programm an PIC2
      bne    ONLY_ONE_PRG_FOR_PIC3
      move.l A1,A0                    * A0 StartAdresse, Programm --> PIC
      bsr.l  SEND_PRG_TO_PIC2

    ONLY_ONE_PRG_FOR_PIC3
      btst.b #$03,correct_pics        * Bit 3 = 0 --> Programm an PIC3
      bne    ONLY_ONE_PRG_RDY
130   move.l A1,A0                    * A0 StartAdresse, Programm --> PIC
      bsr.l  SEND_PRG_TO_PIC3

    ONLY_ONE_PRG_RDY
      rts
    ***
    ********************************************************


    ********************************************************
140 ***
    *** Name....: GET_ONLY_ONE_PRG
    *** Funktion: Empfaengt ein PIC-Programm und verschickt es entsprechend.
    *** Register: A0, A1, A2 als ProgrammAdressen
    ***
    GET_ONLY_ONE_PRG
      move.l #var_start+$100,A0       * StartAdresse ist "varstart"+$100
      move.l A0,A1                    * A1 ist StartAdresse des Programmes
      bsr.l  GET_PRG_MESSAGE          * Aufforderung zur Programmuebergabe
      bsr.l  GETTING_DATA             * Programm-Empfang ueber RS232-Schnittstelle
150   move.l A0,A2                    * A2 ist EndAdresse des Programmes
      move.l A1,A0                    * A0 StartAdresse, Programm --> PIC
      cmp.b  #07,correct_pics         * nur PIC3 vorhanden
      beq    SEND_PRG_TO_PIC3
      cmp.b  #11,correct_pics         * nur PIC2 vorhanden
      beq    SEND_PRG_TO_PIC2
      cmp.b  #13,correct_pics         * nur PIC1 vorhanden
      beq    SEND_PRG_TO_PIC1
      cmp.b  #14,correct_pics         * nur PIC0 vorhanden
      beq    SEND_PRG_TO_PIC0
160   rts
    ***
    ********************************************************


    ********************************************************
    ***
    *** Name....: GETTING_DATA
    *** Funktion: Empfang der "Hex-Daten" der PIC-Programme
    ***           (vgl. Microchip Datenblatt fuer die vers. Dateiformate)
170 *** Register: D0-D6, A0
    ***
    GETTING_DATA
      move.b #00,chk_sum
      moveq  #00,d0
      moveq  #00,d1
      moveq  #00,d2
      moveq  #00,d3

      bsr.l  RECEIVERB_LOOP           * Auf Zeichen warten
180   cmp.b  #58,d6                   * (58) hex = ':', 1. Zeichen einer Zeile
      bne    HEX_ERROR                * falsches Zeichen

    *** Anzahl Befehle dieser Zeile empfangen (High Byte)
      bsr.l  RECEIVERB_LOOP           * Auf Zeichen warten
      move.b d6,d3
      bsr.l  A2SYMBOL
      lsl.b  #4,d3                    * ins Highbyte "shiften"
      move.b d3,d0

190 *** Anzahl Befehle dieser Zeile empfangen (Low Byte)
```

```
      bsr.l   RECEIVERB_LOOP          * Auf Zeichen warten
      move.b  d6,d3
      bsr.l   A2SYMBOL
      add.b   d3,d0                   * zum High-Byte addieren
      move.b  d0,(A0)+                * D0 speichern, A0 inkrementieren
      move.b  d0,sum_instr
      add.b   d0,chk_sum              * "chksum" aktualisieren

    *** "High Part" der ZielAdresse empfangen (High Byte)
200   moveq   #00,d1                  * noetig fuer Carry-Bit
      bsr.l   RECEIVERB_LOOP          * Auf Zeichen warten
      move.b  d6,d3
      bsr.l   A2SYMBOL
      lsl.b   #4,d3                   * ins High Byte "shiften"
      move.b  d3,d0

    *** "High Part" der ZielAdresse empfangen (Low Byte)
      bsr.l   RECEIVERB_LOOP          * Auf Zeichen warten
      move.b  d6,d3
210   bsr.l   A2SYMBOL
      add.b   d3,d0                   * komplette High Anteil
      add.b   d0,chk_sum              * "chksum" aktualisieren
      lsr.b   #1,d0                   * Aufgrund Datenformat Adresse halbieren
      bcc     LOW_ADDRESS             * Carry Bit?
      move.b  #$80,d1                 * evtl. Carry Bit sichern

    *** "Low Part" der ZielAdresse empfangen (High Byte)
    LOW_ADDRESS
      move.b  d0,(A0)+                * "High Part" speichern, A0 inkrementieren
220
      bsr.l   RECEIVERB_LOOP          * Auf Zeichen warten
      move.b  d6,d3
      bsr.l   A2SYMBOL
      lsl.b   #4,d3                   * ins High Byte "shiften"
      move.b  d3,d0

    *** "Low Part" der ZielAdresse empfangen (Low Byte)
      bsr.l   RECEIVERB_LOOP          * Auf Zeichen warten
      move.b  d6,d3
230   bsr.l   A2SYMBOL
      add.b   d3,d0                   * kompletter "Low Part"
      add.b   d0,chk_sum              * "chksum" aktualisieren
      lsr.b   #1,d0                   * Adresse halbieren
      add.b   d1,d0                   * Carry Bit addieren
      move.b  d0,(A0)+                * Speichern und A0 inkrementieren

    *** Dateiende? (01 = Dateiende)
      moveq   #00,d2
      bsr.l   RECEIVERB_LOOP          * Auf Zeichen warten
240   move.b  d6,d3
      bsr.l   A2SYMBOL
      lsl.b   #4,d3
      move.b  d3,d0
      bsr.l   RECEIVERB_LOOP          * "Low Part"
      move.b  d6,d3
      bsr.l   A2SYMBOL
      add.b   d3,d0
      move.b  d0,(A0)+                * Speichern und A0 inkrementieren
      move.b  d0,d2                   * Dateiende
250   add.b   d0,chk_sum              * "chksum" aktualisieren

      move.b  sum_instr,d1
      cmp.b   #00,d1                  * Gibt es Befehle in dieser Zeile?
      beq     CHKSUM_CONTROL          * Falls nein, so berechne "chksum"

    *** Befehlsdekodierung
    GETTING_INSTRUCTIONS
      bsr.l   RECEIVERB_LOOP
      move.b  d6,d3
260   bsr.l   A2SYMBOL
      lsl.b   #4,d3                   * ins High Byte "shiften"
      move.b  d3,d0
      bsr.l   RECEIVERB_LOOP
```

```
      move.b  d6,d3
      bsr.l   A2SYMBOL
      add.b   d3,d0
      move.b  d0,(A0)+                * erster Teil des aktuellen Befehls
      add.b   d0,chk_sum

270 *** zweiten Teil des Befehls empfangen
      bsr.l   RECEIVERB_LOOP
      move.b  d6,d3
      bsr.l   A2SYMBOL
      lsl.b   #4,d3                   * ins High Byte "shiften"
      move.b  d3,d0
      bsr.l   RECEIVERB_LOOP
      move.b  d6,d3
      bsr.l   A2SYMBOL
      add.b   d3,d0
280   move.b  d0,(A0)+                * zweiter Teil des aktuellen Befehls
      add.b   d0,chk_sum

      subi.b  #02,d1                  * Zwei Halbbefehle erniedrigen
      bne     GETTING_INSTRUCTIONS    * naechsten Befehl empfangen

    *** "chksum" empfangen und mit der Eigenen vergleichen
    CHKSUM_CONTROL
      bsr.l   RECEIVERB_LOOP
      move.b  d6,d3
290   bsr.l   A2SYMBOL
      lsl.b   #4,d3                   * ins High Byte "shiften"
      move.b  d3,d0
      bsr.l   RECEIVERB_LOOP
      move.b  d6,d3
      bsr.l   A2SYMBOL
      add.b   d3,d0

      move.b  chk_sum,d1
      not.b   d1                      * "chksum" negieren
300   addi.b  #01,d1                  * und inkrementieren (vgl. Datenblatt)
      cmp.b   d0,d1                   * Vergleich der Ergebnisse
      beq     NO_CHKSUM_ERROR
      bsr.l   CHKSUM_ERROR

    NO_CHKSUM_ERROR
      bsr.l   RECEIVERB_LOOP          * 'line feed'
      bsr.l   RECEIVERB_LOOP          * 'carriage return'

    *** Weitere Befehlszeilen?
310   cmp.b   #01,d2
      bne     GETTING_DATA            * Empfang der naechsten Zeile

    HEX_FILE_RDY
      rts
    ***
    ************************************************************
```

B.10 Programm-Übergabe an die Prozessoren: sendprg.a

```
    ************************************************************
    ***             Programm-Uebergabe an die PICs           ***
    *** Autor:      Tobias Schubert                          ***
    *** EMail:      schubert@informatik.uni-freiburg.de      ***
    *** Datum:      02.08.1999                               ***
    *** Datei-Name: sendprg.a                                ***
    ************************************************************


 10 ****************************************************
    ***
```

```
*** Name....: SEND_PRG_TO_PIC0
*** Funktion: Programm an PIC0 uebergeben
*** Register: D2 fuer PIC-IRQ-Bestaetigung,
***           D4 fuer Warteschleife,
***           A0, A2 fuer Programm-Adressen (A2 signalisiert EndAdresse)
***
SEND_PRG_TO_PIC0
  move.b #$00,pic0              * PIC0 das "Startsignal" geben
20
  move.w #$1000,d4              * Warteschleife
SEND_PIC0_LOOP
  subi.w #$0001,d4
  cmp.w  #$0000,d4
  bne    SEND_PIC0_LOOP

SEND_PRG_TO_PIC0_LOOP
  cmp.l  A0,A2                  * Ende des zu uebertragenden Programmes?
  beq    SEND_PRG_PIC0_RDY
30 moveq  #00,d0
  move.b (A0)+,pic0             * Ein Byte des Programmes uebertragen
WAITING_SEND_PRG_PIC0
  cmp.b  #00,d0                 * Bestaetigung des PICs ($01) abwarten
  beq    WAITING_SEND_PRG_PIC0
  cmp.b  #01,d0
  beq    SEND_PRG_TO_PIC0_LOOP
SEND_PRG_PIC0_RDY
  rts
***
40 ******************************************************


******************************************************
***
*** Name....: SEND_PRG_TO_PIC1
*** Funktion: Programm an PIC1 uebergeben
*** Register: D2 fuer PIC-IRQ-Bestaetigung,
***           D4 fuer Warteschleife,
***           A0, A2 fuer Programm-Adressen (A2 signalisiert EndAdresse)
50 ***
SEND_PRG_TO_PIC1
  move.b #$00,pic1              * PIC1 das "Startsignal" geben

  move.w #$1000,d4              * Warteschleife
SEND_PIC1_LOOP
  subi.w #$0001,d4
  cmp.w  #$0000,d4
  bne    SEND_PIC1_LOOP

60 SEND_PRG_TO_PIC1_LOOP
  cmp.l  A0,A2                  * Ende des zu uebertragenden Programmes?
  beq    SEND_PRG_PIC1_RDY
  moveq  #00,d1
  move.b (A0)+,pic1             * Ein Byte des Programmes uebertragen
WAITING_SEND_PRG_PIC1
  cmp.b  #00,d1                 * Bestaetigung des PICs ($01) abwarten
  beq    WAITING_SEND_PRG_PIC1
  cmp.b  #01,d1
  beq    SEND_PRG_TO_PIC1_LOOP
70 SEND_PRG_PIC1_RDY
  rts
***
******************************************************


******************************************************
***
*** Name....: SEND_PRG_TO_PIC2
*** Funktion: Programm an PIC2 uebergeben
80 *** Register: D2 fuer PIC-IRQ-Bestaetigung,
***           D4 fuer Warteschleife,
***           A0, A2 fuer Programm-Adressen (A2 signalisiert EndAdresse)
***
SEND_PRG_TO_PIC2
```

```
      move.b #$00,pic2              * PIC2 das "Startsignal" geben

      move.w #$1000,d4              * Warteschleife
    SEND_PIC2_LOOP
      subi.w #$0001,d4
 90   cmp.w  #$0000,d4
      bne    SEND_PIC2_LOOP

    SEND_PRG_TO_PIC2_LOOP
      cmp.l  A0,A2                  * Ende des zu uebertragenden Programmes?
      beq    SEND_PRG_PIC2_RDY
      moveq  #00,d2
      move.b (A0)+,pic2             * Ein Byte des Programmes uebertragen
    WAITING_SEND_PRG_PIC2
      cmp.b  #00,d2                 * Bestaetigung des PICs ($01) abwarten
100   beq    WAITING_SEND_PRG_PIC2
      cmp.b  #01,d2
      beq    SEND_PRG_TO_PIC2_LOOP
    SEND_PRG_PIC2_RDY
      rts
    ***
    ******************************************************


    ******************************************************
110 ***
    *** Name....: SEND_PRG_TO_PIC3
    *** Funktion: Programm an PIC3 uebergeben
    *** Register: D2 fuer PIC-IRQ-Bestaetigung,
    ***           D4 fuer Warteschleife,
    ***           A0, A2 fuer Programm-Adressen (A2 signalisiert EndAdresse)
    ***
    SEND_PRG_TO_PIC3
      move.b #$00,pic3              * PIC3 das "Startsignal" geben

120   move.w #$1000,d4              * Warteschleife
    SEND_PIC3_LOOP
      subi.w #$0001,d4
      cmp.w  #$0000,d4
      bne    SEND_PIC3_LOOP

    SEND_PRG_TO_PIC3_LOOP
      cmp.l  A0,A2                  * Ende des zu uebertragenden Programmes?
      beq    SEND_PRG_PIC3_RDY
      moveq  #00,d3
130   move.b (A0)+,pic3             * Ein Byte des Programmes uebertragen
    WAITING_SEND_PRG_PIC3
      cmp.b  #00,d3                 * Bestaetigung des PICs ($01) abwarten
      beq    WAITING_SEND_PRG_PIC3
      cmp.b  #01,d3
      beq    SEND_PRG_TO_PIC3_LOOP
    SEND_PRG_PIC3_RDY
      rts
    ***
    ******************************************************
```

B.11 Festlegung der Daten-Größe: datasize.a

```
*************************************************************
***              Groesse der auszutauschenden Daten abfragen ***
*** Autor:       Tobias Schubert                             ***
*** EMail:       schubert@informatik.uni-freiburg.de         ***
*** Datum:       03.08.1999                                  ***
*** Datei-Name:  datasize.a                                  ***
*************************************************************


*****************************************************
***
*** Name....:  GET_DATA_SIZE
*** Funktion:  Groesse der auszutauschenden Daten abfragen
*** Register:  D0 - D7
***
GET_DATA_SIZE
  bsr.l   DATASIZE_MESSG          * Erklaerung der Eingabe
  move.b  #$00,chr_sizeL          * Groesse der auszutauschenden Chromosome
  move.b  #$00,chr_sizeH
  move.l  #var_start+$100,a0      * Zeiger auf PIC0-"Austausch"-Daten
  move.l  #var_start+$100,a1      * Zeiger auf PIC1-"Austausch"-Daten
  move.l  #var_start+$100,a2      * Zeiger auf PIC2-"Austausch"-Daten
  move.l  #var_start+$100,a3      * Zeiger auf PIC3-"Austausch"-Daten
  moveq   #00,d0                  * Speichert Datengroesse
  moveq   #00,d3
  bsr.l   RECEIVERB_LOOP          * Auf Symbol warten
  move.b  d6,d3
  bsr.l   A2SYMBOL                * ASCII in Zeichen konvertieren
COUNT_100
  cmp.b   #$00,d3
  beq     GET_SYMBOL_10
  subi.b  #$01,d3
  add.l   #$64,d0
  bra.l   COUNT_100
GET_SYMBOL_10
  moveq   #00,d3
  bsr.l   RECEIVERB_LOOP          * Auf Symbol warten
  move.b  d6,d3
  bsr.l   A2SYMBOL                * ASCII in Zeichen konvertieren
COUNT_10
  cmp.b   #$00,d3
  beq     GET_SYMBOL_1
  subi.b  #$01,d3
  add.l   #$0A,d0
  bra.l   COUNT_10
GET_SYMBOL_1
  moveq   #00,d3
  bsr.l   RECEIVERB_LOOP          * Auf Symbol warten
  move.b  d6,d3
  bsr.l   A2SYMBOL                * ASCII in Zeichen konvertieren
  add.l   d3,d0

  lsl.l   #$03,d0
  lsr.l   #$03,d0                 * Jede Speicherzelle speichert 1 Byte
  add.l   d0,a1                   * Adjustieren der 4 Zeiger, damit keine
  add.l   d0,a2                   * Ueberlappungen auftreten.
  add.l   d0,a2
  add.l   d0,a3
  add.l   d0,a3
  add.l   d0,a3

  lsl.l   #$03,d0                 * Datengroesse in High- und Low-Anteil
  move.b  d0,chr_sizeL            * aufteilen und abspeichern
  lsr.l   #$08,d0
  move.b  d0,chr_sizeH
  rts
***
*****************************************************
```

B.12 Festlegung der Topologie: topology.a

```
*************************************************************
***              Empfang der Datenaustausch-Topologie     ***
*** Autor:       Tobias Schubert                          ***
*** EMail:       schubert@informatik.uni-freiburg.de      ***
*** Datum:       04.08.1999                               ***
*** Datei-Name:  topology.a                               ***
*************************************************************


*************************************************************
***
*** Name....:  GET_TOPOLOGY
*** Funktion: Empfaengt Datenaustauschstrategie
*** Register: D0 - D7 fuer Datenuebergabe
***
GET_TOPOLOGY
  move.b #$00,topo0              * Bit x=1 in topo y
  move.b #$00,topo1              * --> PICy gibt Daten an PICx
  move.b #$00,topo2
  move.b #$00,topo3

  cmp.b  #07,correct_pics        * nur PIC3 vorhanden
  beq    NO_TOPO_NEEDED
  cmp.b  #11,correct_pics        * nur PIC2 vorhanden
  beq    NO_TOPO_NEEDED
  cmp.b  #13,correct_pics        * nur PIC1 vorhanden
  beq    NO_TOPO_NEEDED
  cmp.b  #14,correct_pics        * nur PIC0 vorhanden
  beq    NO_TOPO_NEEDED
  bsr.l  TOPO_INTRO_MESS         * "Datenaustausch"
  btst.b #$00,correct_pics       * Bit 0 = 0 --> PIC0 erhielt Programm
  bne    TOPO_PIC1
  btst.b #$01,correct_pics       * Bit 1 = 0 --> PIC1 erhielt Programm
  bne    TOPO_PIC0_TO_PIC2
  moveq  #00,D0
  moveq  #01,D1                  * PIC0 --> PIC1 ?
  bsr.l  TOPO_DATA_MESS
  bsr.l  RECEIVERB_LOOP          * Auf Symbol warten
  cmp.b  #$6A,d6                 * (6A)hex = 'j'
  bne    TOPO_PIC0_TO_PIC2
  bset   #$01,topo0
TOPO_PIC0_TO_PIC2
  move.b #$0D,d4                 * = "CR"
  bsr.l  SEND_CHB_BYTE
  btst.b #$02,correct_pics       * Bit 2 = 0 --> PIC2 erhielt Programm
  bne    TOPO_PIC0_TO_PIC3
  moveq  #00,D0
  moveq  #02,D1                  * PIC0 --> PIC2 ?
  bsr.l  TOPO_DATA_MESS
  bsr.l  RECEIVERB_LOOP          * Auf Symbol warten
  cmp.b  #$6A,d6                 * (6A)hex = 'j'
  bne    TOPO_PIC0_TO_PIC3
  bset   #$02,topo0
TOPO_PIC0_TO_PIC3
  move.b #$0D,d4                 * = "CR"
  bsr.l  SEND_CHB_BYTE
  btst.b #$03,correct_pics       * Bit 3 = 0 --> PIC3 erhielt Programm
  bne    TOPO_PIC1
  moveq  #00,D0
  moveq  #03,D1                  * PIC0 --> PIC3 ?
  bsr.l  TOPO_DATA_MESS
  bsr.l  RECEIVERB_LOOP          * Auf Symbol warten
  cmp.b  #$6A,d6                 * (6A)hex = 'j'
  bne    TOPO_PIC1
  bset   #$03,topo0

TOPO_PIC1
  moveq  #02,d7
  bsr.l  SEND_CR
  btst.b #$01,correct_pics       * Bit 1 = 0 --> PIC1 erhielt Programm
```

```
      bne     TOPO_PIC2
      btst.b  #$00,correct_pics       * Bit 0 = 0 --> PIC0 erhielt Programm
      bne     TOPO_PIC1_TO_PIC2
      moveq   #01,D0
      moveq   #00,D1                  * PIC1 --> PIC0 ?
      bsr.l   TOPO_DATA_MESS
      bsr.l   RECEIVERB_LOOP          * Auf Symbol warten
      cmp.b   #$6A,d6                 * (6A) hex = 'j'
      bne     TOPO_PIC1_TO_PIC2
 80   bset    #$00,topo1
    TOPO_PIC1_TO_PIC2
      move.b  #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE
      btst.b  #$02,correct_pics       * Bit 2 = 0 --> PIC2 erhielt Programm
      bne     TOPO_PIC1_TO_PIC3
      moveq   #01,D0
      moveq   #02,D1                  * PIC1 --> PIC2 ?
      bsr.l   TOPO_DATA_MESS
      bsr.l   RECEIVERB_LOOP          * Auf Symbol warten
 90   cmp.b   #$6A,d6                 * (6A) hex = 'j'
      bne     TOPO_PIC1_TO_PIC3
      bset    #$02,topo1
    TOPO_PIC1_TO_PIC3
      move.b  #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE
      btst.b  #$03,correct_pics       * Bit 3 = 0 --> PIC3 erhielt Programm
      bne     TOPO_PIC2
      moveq   #01,D0
      moveq   #03,D1                  * PIC1 --> PIC3 ?
100   bsr.l   TOPO_DATA_MESS
      bsr.l   RECEIVERB_LOOP          * Auf Symbol warten
      cmp.b   #$6A,d6                 * (6A) hex = 'j'
      bne     TOPO_PIC2
      bset    #$03,topo1

    TOPO_PIC2
      moveq   #02,d7
      bsr.l   SEND_CR
      btst.b  #$02,correct_pics       * Bit 2 = 0 --> PIC2 erhielt Programm
110   bne     TOPO_PIC3
      btst.b  #$00,correct_pics       * Bit 0 = 0 --> PIC0 erhielt Programm
      bne     TOPO_PIC2_TO_PIC1
      moveq   #02,D0
      moveq   #00,D1                  * PIC2 --> PIC0 ?
      bsr.l   TOPO_DATA_MESS
      bsr.l   RECEIVERB_LOOP          * Auf Symbol warten
      cmp.b   #$6A,d6                 * (6A) hex = 'j'
      bne     TOPO_PIC2_TO_PIC1
      bset    #$00,topo2
120 TOPO_PIC2_TO_PIC1
      move.b  #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE
      btst.b  #$01,correct_pics       * Bit 1 = 0 --> PIC1 erhielt Programm
      bne     TOPO_PIC2_TO_PIC3
      moveq   #02,D0
      moveq   #01,D1                  * PIC2 --> PIC1 ?
      bsr.l   TOPO_DATA_MESS
      bsr.l   RECEIVERB_LOOP          * Auf Symbol warten
      cmp.b   #$6A,d6                 * (6A) hex = 'j'
130   bne     TOPO_PIC2_TO_PIC3
      bset    #$01,topo2
    TOPO_PIC2_TO_PIC3
      move.b  #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE
      btst.b  #$03,correct_pics       * Bit 3 = 0 --> PIC3 erhielt Programm
      bne     TOPO_PIC3
      moveq   #02,D0
      moveq   #03,D1                  * PIC2 --> PIC3 ?
      bsr.l   TOPO_DATA_MESS
140   bsr.l   RECEIVERB_LOOP          * Auf Symbol warten
      cmp.b   #$6A,d6                 * (6A) hex = 'j'
      bne     TOPO_PIC3
      bset    #$03,topo2
```

```
TOPO_PIC3
  moveq  #02,d7
  bsr.l  SEND_CR
  btst.b #$03,correct_pics       * Bit 3 = 0 --> PIC3 erhielt Programm
  bne    NO_TOPO_NEEDED
  btst.b #$00,correct_pics       * Bit 0 = 0 --> PIC0 erhielt Programm
  bne    TOPO_PIC3_TO_PIC1
  moveq  #03,D0
  moveq  #00,D1                  * PIC3 --> PIC0 ?
  bsr.l  TOPO_DATA_MESS
  bsr.l  RECEIVERB_LOOP          * Auf Symbol warten
  cmp.b  #$6A,d6                 * (6A)hex = 'j'
  bne    TOPO_PIC3_TO_PIC1
  bset   #$00,topo3
TOPO_PIC3_TO_PIC1
  move.b #$0D,d4                 * = "CR"
  bsr.l  SEND_CHB_BYTE
  btst.b #$01,correct_pics       * Bit 1 = 0 --> PIC1 erhielt Programm
  bne    TOPO_PIC3_TO_PIC2
  moveq  #03,D0
  moveq  #01,D1                  * PIC3 --> PIC1 ?
  bsr.l  TOPO_DATA_MESS
  bsr.l  RECEIVERB_LOOP          * Auf Symbol warten
  cmp.b  #$6A,d6                 * (6A)hex = 'j'
  bne    TOPO_PIC3_TO_PIC2
  bset   #$01,topo3
TOPO_PIC3_TO_PIC2
  move.b #$0D,d4                 * = "CR"
  bsr.l  SEND_CHB_BYTE
  btst.b #$02,correct_pics       * Bit 2 = 0 --> PIC2 erhielt Programm
  bne    NO_TOPO_NEEDED
  moveq  #03,D0
  moveq  #02,D1                  * PIC3 --> PIC2 ?
  bsr.l  TOPO_DATA_MESS
  bsr.l  RECEIVERB_LOOP          * Auf Symbol warten
  cmp.b  #$6A,d6                 * (6A)hex = 'j'
  bne    NO_TOPO_NEEDED
  bset   #$02,topo3

NO_TOPO_NEEDED
  moveq #02,d7
  bsr.l SEND_CR
  rts
***
************************************************************
```

B.13 Löschen der PLD-Speicherzellen: clearpld.a

```
************************************************************************
***              Vor dem Start der PICs deren PLDs auf "Null" setzen  ***
*** Autor:        Tobias Schubert                                      ***
*** EMail:        schubert@informatik.uni-freiburg.de                  ***
*** Datum:        25.09.1999                                           ***
*** Datei-Name:   clearpld.a                                           ***
************************************************************************


************************************************************************
***
*** Name....:  CLEAR_PLD
*** Funktion:  Vor dem Start der PICs deren PLDs auf "Null" setzen, um in
***            der eigentlichen Motorola-Hauptschleife den Inhalt der PLDs
***            nicht faelschlicherweise als Aufforderung zum Datenaustausch
***            zu interpretieren. Zusaetzlich einen Zufallsgenerator-Wert
***            an die PICsü bergeben.
*** Register: D3 als Bestaetigung des kontaktierten PIC-Prozessors
```

```
   ***
20 CLEAR_PLD
     move.b correct_pics,results
   *                        Bit x=1 --> PIC x ist mit seinem Programm "fertig",
   *                        jetzt werden die entsprechenden Bits fuer "vorhan-
   *                        dene" PICs auf 0 gesetzt, d.h. die anderen PICs
   *                        koennen in der Hauptschleife kein Signal an den
   *                        Motorola schicken, da dies nur fuer PICs moeglich
   *                        ist, die in "results" eine 0 "stehen haben".
     btst.b #$00,correct_pics
     bne    CALC_SUM_BIT1
30   moveq  #00,d3              * D3 = 01 signalisiert Bestaetigung durch PIC
     move.b random,pic0         * lasse PLD durch PIC0 auf "Null" zuruecksetzen
   CLEAR_PLD_PIC0
     cmp.l  #$00,d3
     beq    CLEAR_PLD_PIC0

   CALC_SUM_BIT1
     btst.b #$01,correct_pics
     bne    CALC_SUM_BIT2
     subi.b #03,random
40   moveq  #00,d3              * D3 = 01 signalisiert Bestaetigung durch PIC
     move.b random,pic1         * lasse PLD durch PIC1 auf "Null" zuruecksetzen
   CLEAR_PLD_PIC1
     cmp.l  #$00,d3
     beq    CLEAR_PLD_PIC1

   CALC_SUM_BIT2
     btst.b #$02,correct_pics
     bne    CALC_SUM_BIT3
     subi.b #04,random
50   moveq  #00,d3              * D3 = 01 signalisiert Bestaetigung durch PIC
     move.b random,pic2         * lasse PLD durch PIC2 auf "Null" zuruecksetzen
   CLEAR_PLD_PIC2
     cmp.l  #$00,d3
     beq    CLEAR_PLD_PIC2

   CALC_SUM_BIT3
     btst.b #$03,correct_pics
     bne    CALC_SUM_RDY
     subi.b #05,random
60   moveq  #00,d3              * D3 = 01 signalisiert Bestaetigung durch PIC
     move.b random,pic3         * lasse PLD durch PIC3 auf "Null" zuruecksetzen
   CLEAR_PLD_PIC3
     cmp.l  #$00,d3
     beq    CLEAR_PLD_PIC3

   CALC_SUM_RDY
     rts
   ***
   ************************************************************************
```

B.14 Initialisierung der Variablen: `initreg.a`

```
   ************************************************************************
   ***             Vor dem Start der PICs alle Register auf "Null" setzen ***
   *** Autor:       Tobias Schubert                                        ***
   *** EMail:       schubert@informatik.uni-freiburg.de                    ***
   *** Datum:       25.09.1999                                             ***
   *** Datei-Name:  initreg.a                                              ***
   ************************************************************************


10 ************************************************************************
   ***
   *** Name....:  INIT_REGISTER
   *** Funktion: Benoetigte Register auf "Null" setzen
```

```
   *** Register: D0, D1 fuer Schleifenzaehler,
   ***           A5 als Zaehler, um die Zellen zu initialisieren
   ***
   INIT_REGISTER
     move.b #$00,control0          * "controlx" speichert aktuelles Signalwort
     move.b #$00,control1          * des PIC x an den Motorola
20   move.b #$00,control2
     move.b #$00,control3

     move.b #$00,topo0             * Anfaenglich kein Datenaustausch
     move.b #$00,topo1
     move.b #$00,topo2
     move.b #$00,topo3

     move.b #$01,fitness0_lo       * Startwert von 01 ist programmbedingt
     move.b #$00,fitness0_hi
30   move.b #$01,fitness1_lo
     move.b #$00,fitness1_hi
     move.b #$01,fitness2_lo
     move.b #$00,fitness2_hi
     move.b #$01,fitness3_lo
     move.b #$00,fitness3_hi

     move.b #$00,gen0_lo
     move.b #$00,gen0_mi
     move.b #$00,gen0_hi
40   move.b #$00,gen1_lo
     move.b #$00,gen1_mi
     move.b #$00,gen1_hi
     move.b #$00,gen2_lo
     move.b #$00,gen2_mi
     move.b #$00,gen2_hi
     move.b #$00,gen3_lo
     move.b #$00,gen3_mi
     move.b #$00,gen3_hi

50   move.b #$00,child_size0
     move.b #$00,child_size1
     move.b #$00,child_size2
     move.b #$00,child_size3

     move.b #$00,pop_size0
     move.b #$00,pop_size1
     move.b #$00,pop_size2
     move.b #$00,pop_size3

60   move.b #$00,transmitter_pic
     move.b #$00,receiver_pic
     move.b #$00,trans_fit_hi
     move.b #$00,trans_fit_lo

   *** Speicherzellen mit 0-Vektor initialisieren
     move.l a0,a5
     moveq  #00,d0
     moveq  #00,d1
     move.l a3,d0
70   move.b chr_sizeH,d1
     lsl.l  #$08,d1
     add.b  chr_sizeL,d1
     lsr.l  #$03,d1                * 1 Byte pro Speicherzelle
     add.l  d1,d0
   CLEAR_ALL_CHROMO_CELLS
     move.b #$00,(a5)+
     cmp.l  a5,d0
     bne    CLEAR_ALL_CHROMO_CELLS
     rts
80 ***
   ********************************************************************************
```

B.15 Konvertierungsroutinen: convert.a

```
*********************************************************************
***              verschiedene Konvertierungsroutinen              ***
*** Autor:       Tobias Schubert                                  ***
*** EMail:       schubert@informatik.uni-freiburg.de              ***
*** Datum:       14.09.1999                                       ***
*** Datei-Name:  convert.a                                        ***
*********************************************************************


*************************************************************************
***
*** Name....:  A2SYMBOL
*** Funktion:  ASCII-Zeichenkonvertierung, Bsp.: (65) ASCII = "e"
*** Register:  D3 wird als Ein- und Ausgaberegister verwendet.
***
A2SYMBOL
  cmp.l #$30,d3          * = 0
  beq   SYM_0
  cmp.l #$31,d3          * = 1
  beq   SYM_1
  cmp.l #$32,d3          * = 2
  beq   SYM_2
  cmp.l #$33,d3          * = 3
  beq   SYM_3
  cmp.l #$34,d3          * = 4
  beq   SYM_4
  cmp.l #$35,d3          * = 5
  beq   SYM_5
  cmp.l #$36,d3          * = 6
  beq   SYM_6
  cmp.l #$37,d3          * = 7
  beq   SYM_7
  cmp.l #$38,d3          * = 8
  beq   SYM_8
  cmp.l #$39,d3          * = 9
  beq   SYM_9
  cmp.l #$41,d3          * = A
  beq   SYM_A
  cmp.l #$42,d3          * = B
  beq   SYM_B
  cmp.l #$43,d3          * = C
  beq   SYM_C
  cmp.l #$44,d3          * = D
  beq   SYM_D
  cmp.l #$45,d3          * = E
  beq   SYM_E
  cmp.l #$46,d3          * = F
  beq   SYM_F
  bra.l HEX_ERROR
*                        Nichterlaubtes/nichbenoetigtes Zeichen
*                        bei der Datenuebertragung.


SYM_0
  moveq #$00,d3
  rts

SYM_1
  moveq #$01,d3
  rts

SYM_2
  moveq #$02,d3
  rts

SYM_3
  moveq #$03,d3
  rts

SYM_4
```

```
      moveq #$04,d3
      rts

    SYM_5
      moveq #$05,d3
      rts

    SYM_6
      moveq #$06,d3
 80   rts

    SYM_7
      moveq #$07,d3
      rts

    SYM_8
      moveq #$08,d3
      rts

 90 SYM_9
      moveq #$09,d3
      rts

    SYM_A
      moveq #$0A,d3
      rts

    SYM_B
      moveq #$0B,d3
100   rts

    SYM_C
      moveq #$0C,d3
      rts

    SYM_D
      moveq #$0D,d3
      rts

110 SYM_E
      moveq #$0E,d3
      rts

    SYM_F
      moveq #$0F,d3
      rts
    ***
    ************************************************************************

120

    ************************************************************************
    ***
    *** Name....: DECODE_DEZ
    *** Funktion: Anzeigen des besten PIC-Resultates.
    *** Register: D1 als Eingabe, D4 als Ausgabe
    ***
    *** Beispiel: Vor DECODE_DEZ: D1 = 00023
    ***           Nach DECODE_DEZ: D4 = 30 30 30 32 33 --->  an den PC
    ***
130 DECODE_DEZ
      move.l d1,d4
      lsr.l  #$08,d4
      lsr.l  #$08,d4
      bsr.l  SEND_GA_PARAMETER
      move.l d1,d4
      lsr.l  #$08,d4
      lsr.l  #$04,d4
      bsr.l  SEND_GA_PARAMETER
      move.l d1,d4
140   lsr.l  #$08,d4
      bsr.l  SEND_GA_PARAMETER
      move.l d1,d4
      lsr.l  #$04,d4
```

```
    bsr.l   SEND_GA_PARAMETER
    move.l  d1,d4
    bsr.l   SEND_GA_PARAMETER
    rts
***
*************************************************************************
150

*************************************************************************
***
*** Name....: DECODE_DEZ_LONG
*** Funktion: Anzeigen des besten PIC-Resultates.
*** Register: D1 als Eingabe, D4 als Ausgabe
***
*** Beispiel: Vor DECODE_DEZ: D1 = 000023
***           Nach DECODE_DEZ: D4 = 30 30 30 30 32 33 --->  an den PC
160 ***
DECODE_DEZ_LONG
    move.l  d1,d4
    lsr.l   #$08,d4
    lsr.l   #$08,d4
    lsr.l   #$04,d4
    bsr.l   SEND_GA_PARAMETER
    move.l  d1,d4
    lsr.l   #$08,d4
    lsr.l   #$08,d4
170 bsr.l   SEND_GA_PARAMETER
    move.l  d1,d4
    lsr.l   #$08,d4
    lsr.l   #$04,d4
    bsr.l   SEND_GA_PARAMETER
    move.l  d1,d4
    lsr.l   #$08,d4
    bsr.l   SEND_GA_PARAMETER
    move.l  d1,d4
    lsr.l   #$04,d4
180 bsr.l   SEND_GA_PARAMETER
    move.l  d1,d4
    bsr.l   SEND_GA_PARAMETER
    rts
***
*************************************************************************


*************************************************************************
***
190 *** Name....: SEND_GA_PARAMETER
*** Funktion: 1, ..., 9 in entsprechendes ASCII-Symbol verwandeln.
*** Register: D4 als Ein-/Ausgabe
***
*** Beispiel: '5' --> (35)ASCII
***
SEND_GA_PARAMETER
    andi.b  #$0F,d4
    addi.b  #$30,d4
    bsr.l   SEND_CHB_BYTE
200 rts
***
*************************************************************************


*************************************************************************
***
*** Name....: HEX2DEZ
*** Funktion: HEX-Werte in Dezimaldarstellung zur Ausgabe am PC wandeln.
*** Register: D0 als Eingabe, D1 als Ausgabe, D2
210 ***
*** Beispiel:      (F36A)16 = (62374)10
*** Vor  HEX2DEZ: D0 = 0F36A
***               D1 = xxxxx
*** Nach HEX2DEZ: D0 = 00000
***               D1 = 62374
***
```

```
    HEX2DEZ
      addi.w  #$01,d1
      cmp.b   #$0A,d1
220   beq     CARRY_BIT0
      cmp.b   #$1A,d1
      beq     CARRY_BIT0
      cmp.b   #$2A,d1
      beq     CARRY_BIT0
      cmp.b   #$3A,d1
      beq     CARRY_BIT0
      cmp.b   #$4A,d1
      beq     CARRY_BIT0
      cmp.b   #$5A,d1
230   beq     CARRY_BIT0
      cmp.b   #$6A,d1
      beq     CARRY_BIT0
      cmp.b   #$7A,d1
      beq     CARRY_BIT0
      cmp.b   #$8A,d1
      beq     CARRY_BIT0
      cmp.b   #$9A,d1
      beq     CARRY_BIT1
    HEX2DEZ_GOON
240   subi.w  #$01,d0
      cmp.w   #$0000,d0
      bne     HEX2DEZ
      rts

    CARRY_BIT0
      addi.w  #$06,d1
      bra.l   HEX2DEZ_GOON

    CARRY_BIT1
250   move.w  d1,d2
      lsr.w   #$08,d2
      cmp.b   #$09,d2
      beq     CARRY_BIT2
      cmp.b   #$19,d2
      beq     CARRY_BIT2
      cmp.b   #$29,d2
      beq     CARRY_BIT2
      cmp.b   #$39,d2
      beq     CARRY_BIT2
260   cmp.b   #$49,d2
      beq     CARRY_BIT2
      cmp.b   #$59,d2
      beq     CARRY_BIT2
      cmp.b   #$69,d2
      beq     CARRY_BIT2
      cmp.b   #$79,d2
      beq     CARRY_BIT2
      cmp.b   #$89,d2
      beq     CARRY_BIT2
270   cmp.b   #$99,d2
      beq     CARRY_BIT2
      addi.w  #$66,d1
      bra.l   HEX2DEZ_GOON

    CARRY_BIT2
      move.w  d1,d2
      lsr.w   #$04,d2
      lsr.w   #$08,d2
      cmp.b   #$09,d2
280   beq     CARRY_BIT3
      cmp.b   #$19,d2
      beq     CARRY_BIT3
      cmp.b   #$29,d2
      beq     CARRY_BIT3
      cmp.b   #$39,d2
      beq     CARRY_BIT3
      cmp.b   #$49,d2
      beq     CARRY_BIT3
      cmp.b   #$59,d2
```

```
290   beq      CARRY_BIT3
      cmp.b    #$69,d2
      beq      CARRY_BIT3
      cmp.b    #$79,d2
      beq      CARRY_BIT3
      cmp.b    #$89,d2
      beq      CARRY_BIT3
      cmp.b    #$99,d2
      beq      CARRY_BIT3
      addi.w   #$0666,d1
300   bra.l    HEX2DEZ_GOON

    CARRY_BIT3
      addi.l   #$6666,d1
      bra.l    HEX2DEZ_GOON
    ***
    ************************************************************************


    ************************************************************************
310 ***
    *** Name....: HEX2DEZ_LONG
    *** Funktion: HEX-Werte in Dezimaldarstellung zur Ausgabe am PC wandeln.
    *** Register: D0 als Eingabe, D1 als Ausgabe, D2
    ***
    *** Beispiel:      (0000 F36A)16 = (062374)10
    *** Vor HEX2DEZ: D0 = 0000 F36A
    ***               D1 = xxxx xxxx
    *** Nach HEX2DEZ: D0 = 0000 0000
    ***               D1 = 0006 2374
320 ***
    HEX2DEZ_LONG
      addi.l   #$00000001,d1
      cmp.b    #$0A,d1
      beq      CARRY_BIT0_LONG
      cmp.b    #$1A,d1
      beq      CARRY_BIT0_LONG
      cmp.b    #$2A,d1
      beq      CARRY_BIT0_LONG
      cmp.b    #$3A,d1
330   beq      CARRY_BIT0_LONG
      cmp.b    #$4A,d1
      beq      CARRY_BIT0_LONG
      cmp.b    #$5A,d1
      beq      CARRY_BIT0_LONG
      cmp.b    #$6A,d1
      beq      CARRY_BIT0_LONG
      cmp.b    #$7A,d1
      beq      CARRY_BIT0_LONG
      cmp.b    #$8A,d1
340   beq      CARRY_BIT0_LONG
      cmp.b    #$9A,d1
      beq      CARRY_BIT1_LONG
    HEX2DEZ_GOON_LONG
      subi.l   #$00000001,d0
      cmp.l    #$00000000,d0
      bne      HEX2DEZ_LONG
      rts

    CARRY_BIT0_LONG
350   addi.l   #$00000006,d1
      bra.l    HEX2DEZ_GOON_LONG

    CARRY_BIT1_LONG
      move.l   d1,d2
      lsr.l    #$00000008,d2
      cmp.b    #$09,d2
      beq      CARRY_BIT2_LONG
      cmp.b    #$19,d2
      beq      CARRY_BIT2_LONG
360   cmp.b    #$29,d2
      beq      CARRY_BIT2_LONG
      cmp.b    #$39,d2
```

```
      beq       CARRY_BIT2_LONG
      cmp.b     #$49,d2
      beq       CARRY_BIT2_LONG
      cmp.b     #$59,d2
      beq       CARRY_BIT2_LONG
      cmp.b     #$69,d2
      beq       CARRY_BIT2_LONG
370   cmp.b     #$79,d2
      beq       CARRY_BIT2_LONG
      cmp.b     #$89,d2
      beq       CARRY_BIT2_LONG
      cmp.b     #$99,d2
      beq       CARRY_BIT2_LONG
      addi.l    #$00000066,d1
      bra.l     HEX2DEZ_GOON_LONG

    CARRY_BIT2_LONG
380   move.l    d1,d2
      lsr.l     #$00000004,d2
      lsr.l     #$00000008,d2
      cmp.b     #$09,d2
      beq       CARRY_BIT3_LONG
      cmp.b     #$19,d2
      beq       CARRY_BIT3_LONG
      cmp.b     #$29,d2
      beq       CARRY_BIT3_LONG
      cmp.b     #$39,d2
390   beq       CARRY_BIT3_LONG
      cmp.b     #$49,d2
      beq       CARRY_BIT3_LONG
      cmp.b     #$59,d2
      beq       CARRY_BIT3_LONG
      cmp.b     #$69,d2
      beq       CARRY_BIT3_LONG
      cmp.b     #$79,d2
      beq       CARRY_BIT3_LONG
      cmp.b     #$89,d2
400   beq       CARRY_BIT3_LONG
      cmp.b     #$99,d2
      beq       CARRY_BIT3_LONG
      addi.l    #$00000666,d1
      bra.l     HEX2DEZ_GOON_LONG

    CARRY_BIT3_LONG
      move.l    d1,d2
      lsr.l     #$00000008,d2
      lsr.l     #$00000008,d2
410   cmp.b     #$09,d2
      beq       CARRY_BIT4_LONG
      cmp.b     #$19,d2
      beq       CARRY_BIT4_LONG
      cmp.b     #$29,d2
      beq       CARRY_BIT4_LONG
      cmp.b     #$39,d2
      beq       CARRY_BIT4_LONG
      cmp.b     #$49,d2
      beq       CARRY_BIT4_LONG
420   cmp.b     #$59,d2
      beq       CARRY_BIT4_LONG
      cmp.b     #$69,d2
      beq       CARRY_BIT4_LONG
      cmp.b     #$79,d2
      beq       CARRY_BIT4_LONG
      cmp.b     #$89,d2
      beq       CARRY_BIT4_LONG
      cmp.b     #$99,d2
      beq       CARRY_BIT4_LONG
430   addi.l    #$00006666,d1
      bra.l     HEX2DEZ_GOON_LONG

    CARRY_BIT4_LONG
      addi.l    #$00066666,d1
      bra.l     HEX2DEZ_GOON_LONG
```

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


*************************************************************************
***
*** Name....: HEX2BIN
*** Funktion: HEX-Werte in Binaerdarstellung zur Ausgabe am PC wandeln.
*** Register: D0 als Eingabe
***
*** Beispiel: (AF) --> HEX2BIN --> (1010 1111) --> PC
***
HEX2BIN
  btst.b #07,d0
  bne    bit0_1
  move.b #$30,d4
  bsr.l  SEND_CHB_BYTE
  bra.l  BIT1
bit0_1
  move.b #$31,d4
  bsr.l  SEND_CHB_BYTE
BIT1
  btst.b #06,d0
  bne    bit1_1
  move.b #$30,d4
  bsr.l  SEND_CHB_BYTE
  bra.l  BIT2
bit1_1
  move.b #$31,d4
  bsr.l  SEND_CHB_BYTE
BIT2
  btst.b #05,d0
  bne    bit2_1
  move.b #$30,d4
  bsr.l  SEND_CHB_BYTE
  bra.l  BIT3
bit2_1
  move.b #$31,d4
  bsr.l  SEND_CHB_BYTE
BIT3
  btst.b #04,d0
  bne    bit3_1
  move.b #$30,d4
  bsr.l  SEND_CHB_BYTE
  bra.l  BIT4
bit3_1
  move.b #$31,d4
  bsr.l  SEND_CHB_BYTE
BIT4
  btst.b #03,d0
  bne    bit4_1
  move.b #$30,d4
  bsr.l  SEND_CHB_BYTE
  bra.l  BIT5
bit4_1
  move.b #$31,d4
  bsr.l  SEND_CHB_BYTE
BIT5
  btst.b #02,d0
  bne    bit5_1
  move.b #$30,d4
  bsr.l  SEND_CHB_BYTE
  bra.l  BIT6
bit5_1
  move.b #$31,d4
  bsr.l  SEND_CHB_BYTE
BIT6
  btst.b #01,d0
  bne    bit6_1
  move.b #$30,d4
  bsr.l  SEND_CHB_BYTE
  bra.l  BIT7
bit6_1
```

```
    move.b #$31,d4
    bsr.l  SEND_CHB_BYTE
BIT7
    btst.b #00,d0
    bne    bit7_1
    move.b #$30,d4
    bsr.l  SEND_CHB_BYTE
    rts
bit7_1
    move.b #$31,d4
    bsr.l  SEND_CHB_BYTE
    rts
***
*********************************************************************
```

B.16 Benötigte Bildschirmausgaben: messages.a

```
*************************************************************
***             1.) Startbildschirm mit Name, Datum       ***
***             2.) Meldung fuer Programmende             ***
***             3.) Meldung fuer ungueltigen Interrupt    ***
***             4.) Sonstige Meldungen vers. Routinen     ***
*** Autor:       Tobias Schubert                          ***
*** EMail:       schubert@informatik.uni-freiburg.de      ***
*** Datum:       24.10.1999                               ***
*** Datei-Name:  messages.a                               ***
*************************************************************


*******************************************************
***
*** Name....: START_MESSAGE
*** Funktion: Startmeldung, mit Tastendruck zu quittieren.
*** Register: D4-D7 werden benoetigt fuer zu uebergebende Daten.
***
START_MESSAGE:
    moveq  #04,d7
    bsr.l  SEND_CR
    moveq  #10,d7
    bsr.l  SEND_SPACE             * d7-viele Leerzeichen
    moveq  #53,d7
    bsr.l  SEND_UP_LINE           * obere "Begrenzung" eines Rahmens
    moveq  #10,d7
    bsr.l  SEND_SPACE             * d7-viele Leerzeichen
    move.b #$BA,d4                * = "|"
    bsr.l  SEND_CHB_BYTE
    moveq  #05,d7
    bsr.l  SEND_SPACE             * d7-viele Leerzeichen
    move.l #$74654220,d4          * " Bet"
    move.l #$62656972,d5          * "rieb"
    move.l #$73797373,d6          * "ssys"
    move.l #$206D6574,d7          * "tem "
    bsr.l  SEND_CHB_4LONG
    move.l #$72657566,d4          * "fuer"
    move.l #$6E656720,d5          * " gen"
    move.l #$73697465,d6          * "etis"
    move.l #$20656863,d7          * "che "
    bsr.l  SEND_CHB_4LONG
    move.l #$6F676C41,d4          * "Algo"
    bsr.l  SEND_CHB_1LONG
    move.l #$68746972,d4          * "rith"
    bsr.l  SEND_CHB_1LONG
    move.l #$206E656D,d4          * "men "
    bsr.l  SEND_CHB_1LONG
    moveq  #04,d7
    bsr.l  SEND_SPACE             * d7-viele Leerzeichen
    move.b #$BA,d4                * = "|"
```

```
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE
        moveq   #10,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        moveq   #53,d7
        bsr.l   SEND_MI_LINE            * mittlere "Begrenzung" eines Rahmens
        moveq   #10,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
60      move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.l  #$75412020,d4           * "  Au"
        move.l  #$3A726F74,d5           * "tor:"
        move.l  #$626F5420,d6           * " Tob"
        move.l  #$20736169,d7           * "ias "
        bsr.l   SEND_CHB_4LONG
        move.l  #$75686353,d4           * "Schu"
        bsr.l   SEND_CHB_1LONG
        move.l  #$74726562,d4           * "bert"
70      bsr.l   SEND_CHB_1LONG
        moveq   #29,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0d,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE
        moveq   #10,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        move.b  #$BA,d4                 * = "|"
80      bsr.l   SEND_CHB_BYTE
        move.l  #$6D452020,d4           * "  Em"
        move.l  #$3A6C6961,d5           * "ail:"
        move.l  #$68637320,d6           * " sch"
        move.l  #$72656275,d7           * "uber"
        bsr.l   SEND_CHB_4LONG
        move.l  #$6E694074,d4           * "t@in"
        move.l  #$6D726F66,d5           * "form"
        move.l  #$6B697461,d6           * "atik"
        move.l  #$696E752E,d7           * ".uni"
90      bsr.l   SEND_CHB_4LONG
        move.l  #$6572662D,d4           * "-fre"
        bsr.l   SEND_CHB_1LONG
        move.l  #$72756269,d4           * "ibur"
        bsr.l   SEND_CHB_1LONG
        move.l  #$65642E67,d4           * "g.de"
        bsr.l   SEND_CHB_1LONG
        moveq   #09,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        move.b  #$BA,d4                 * = "|"
100     bsr.l   SEND_CHB_BYTE
        move.b  #$0d,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE
        moveq   #10,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        moveq   #01,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        move.l  #$74614420,d4           * " Dat"
110     move.l  #$203A6D75,d5           * "um: "
        move.l  #$6F746B4F,d6           * "Okto"
        move.l  #$20726562,d7           * "ber "
        bsr.l   SEND_CHB_4LONG
        move.l  #$39393931,d4           * "1999"
        bsr.l   SEND_CHB_1LONG
        moveq   #32,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
120     move.b  #$0d,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE
        moveq   #10,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
```

```
      moveq  #53,d7
      bsr.l  SEND_LO_LINE            * untere "Begrenzung" eines Rahmens
      bsr.l  RECEIVERB_LOOP          * Auf Signal warten
      rts
    ***
    ******************************************************
130

    ******************************************************
    ***
    *** Name....: END_OF_PROGRAM
    *** Funktion: Mitteilung, dass das Programm beendet wird.
    *** Register: D4-D7 werden benoetigt fuer zu uebergebende Daten.
    ***
    END_OF_PROGRAM
      moveq  #04,d7
140   bsr.l  SEND_CR
      moveq  #10,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      moveq  #32,d7
      bsr.l  SEND_UP_LINE            * obere "Begrenzung" eines Rahmens
      moveq  #10,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.l #$202020BA,d4           * "|   "
      move.l #$20736144,d5           * "Das "
      move.l #$676F7250,d6           * "Prog"
150   move.l #$6D6D6172,d7           * "ramm"
      bsr.l  SEND_CHB_4LONG
      move.l #$72697720,d4           * " wir"
      move.l #$65622064,d5           * "d be"
      move.l #$65646E65,d6           * "ende"
      move.l #$20202E74,d7           * "t.  "
      bsr.l  SEND_CHB_4LONG
      move.l #$0D20BA20,d4           * " |  "
      bsr.l  SEND_CHB_1LONG
      moveq  #10,d7
160   bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      moveq  #32,d7
      bsr.l  SEND_LO_LINE            * untere "Begrenzung" eines Rahmens
      rts
    ***
    ******************************************************


    ******************************************************
    ***
170 *** Name....: _uninit_excpt_
    *** Funktion: Motorola-Fehler: Ausnahme-Routine nicht definiert.
    *** Register: D4-D7 werden benoetigt fuer zu uebergebende Daten.
    ***
    _uninit_excpt_:
      moveq  #02,d7
      bsr.l  SEND_CR
      moveq  #10,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      moveq  #66,d7
180   bsr.l  SEND_UP_LINE            * obere "Begrenzung" eines Rahmens
      moveq  #10,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
      moveq  #01,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.l #$6F746F4D,d4           * "Moto"
      bsr.l  SEND_CHB_1LONG
      move.l #$616C6F72,d4           * "rola"
190   bsr.l  SEND_CHB_1LONG
      move.l #$6865462D,d4           * "-Feh"
      bsr.l  SEND_CHB_1LONG
      move.l #$3A72656C,d4           * "ler:"
      move.l #$756F5220,d5           * " Rou"
      move.l #$656E6974,d6           * "tine"
      move.l #$63696E20,d7           * " nic"
```

```
      bsr.l   SEND_CHB_4LONG
      move.l  #$64207468,d4           * "ht d"
      move.l  #$6E696665,d5           * "efin"
200   move.l  #$74726569,d6           * "iert"
      move.l  #$7250202E,d7           * ". Pr"
      bsr.l   SEND_CHB_4LONG
      move.l  #$6172676F,d4           * "ogra"
      move.l  #$77206D6D,d5           * "mm w"
      move.l  #$20647269,d6           * "ird "
      move.l  #$6E656562,d7           * "been"
      bsr.l   SEND_CHB_4LONG
      move.l  #$2E746564,d4           * "det."
      bsr.l   SEND_CHB_1LONG
210   move.b  #$20,d4                 * = " "
      bsr.l   SEND_CHB_BYTE
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE
      moveq   #10,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      moveq   #66,d7
      bsr.l   SEND_LO_LINE            * untere "Begrenzung" eines Rahmens
220   bgnd
    ***
    ******************************************************


    ******************************************************
    ***
    *** Name....: spurious_interrupt
    *** Funktion: Motorola-Fehler: Falsches Interrupt-Timing
    *** Register: D4-D7 werden benoetigt fuer zu uebergebende Daten.
230 ***
    spurious_interrupt:
      moveq   #02,d7
      bsr.l   SEND_CR
      moveq   #07,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      moveq   #68,d7
      bsr.l   SEND_UP_LINE            * obere "Begrenzung" eines Rahmens
      moveq   #07,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
240   move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      moveq   #01,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.l  #$6F746F4D,d4           * "Moto"
      bsr.l   SEND_CHB_1LONG
      move.l  #$616C6F72,d4           * "rola"
      bsr.l   SEND_CHB_1LONG
      move.l  #$6865462D,d4           * "-Feh"
      bsr.l   SEND_CHB_1LONG
250   move.l  #$3A72656C,d4           * "ler:"
      move.l  #$6C614620,d5           * " Fal"
      move.l  #$65686373,d6           * "sche"
      move.l  #$6E492073,d7           * "s In"
      bsr.l   SEND_CHB_4LONG
      move.l  #$72726574,d4           * "terr"
      move.l  #$2D747075,d5           * "upt-"
      move.l  #$696D6954,d6           * "Timi"
      move.l  #$202E676E,d7           * "ng. "
      bsr.l   SEND_CHB_4LONG
260   move.l  #$676F7250,d4           * "Prog"
      bsr.l   SEND_CHB_1LONG
      move.l  #$6D6D6172,d4           * "ramm"
      move.l  #$72697720,d5           * " wir"
      move.l  #$65622064,d6           * "d be"
      move.l  #$65646E65,d7           * "ende"
      bsr.l   SEND_CHB_4LONG
      move.l  #$BA202E74,d4           * "t. |"
      bsr.l   SEND_CHB_1LONG
      move.b  #$0D,d4                 * = "CR"
```

```
270   bsr.l   SEND_CHB_BYTE
      moveq   #07,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      moveq   #68,d7
      bsr.l   SEND_LO_LINE            * untere "Begrenzung" eines Rahmens
      bgnd
    ***
    ******************************************************


280 ******************************************************
    ***
    *** Name....: HEX_ERROR
    *** Funktion: Mitteilung: "PIC-Programm hat falsches Format"
    *** Register: D4-D7 werden benoetigt fuer zu uebergebende Daten.
    ***
    HEX_ERROR
      move.w #$2700,sr              * Interrupt-Level = 7, Supervisor Mode
      move.b #$01,pic0              * "Neustart"-Signal an die 4 PICs
      move.b #$01,pic1
290   move.b #$01,pic2
      move.b #$01,pic3
      moveq   #02,d7
      bsr.l   SEND_CR
      moveq   #77,d7
      bsr.l   SEND_UP_LINE          * obere "Begrenzung" eines Rahmens
      move.l #$684920BA,d4          * "| Ih"
      move.l #$44206572,d5          * "re D"
      move.l #$69657461,d6          * "atei"
      move.l #$74616820,d7          * " hat"
300   bsr.l   SEND_CHB_4LONG
      move.l #$63696E20,d4          * " nic"
      move.l #$64207468,d5          * "ht d"
      move.l #$6B207361,d6          * "as k"
      move.l #$6572726F,d7          * "orre"
      bsr.l   SEND_CHB_4LONG
      move.l #$2065746B,d4          * "kte "
      move.l #$6D726F46,d5          * "Form"
      move.l #$202E7461,d6          * "at. "
      move.l #$20726544,d7          * "Der "
310   bsr.l   SEND_CHB_4LONG
      move.l #$6F746F4D,d4          * "Moto"
      move.l #$616C6F72,d5          * "rola"
      move.l #$65656220,d6          * " bee"
      move.l #$7465646E,d7          * "ndet"
      bsr.l   SEND_CHB_4LONG
      move.l #$69657320,d4          * " sei"
      move.l #$4120656E,d5          * "ne A"
      move.l #$69656272,d6          * "rbei"
      move.l #$0DBA2E74,d7          * "t.| "
320   bsr.l   SEND_CHB_4LONG
      moveq   #77,d7
      bsr.l   SEND_LO_LINE          * untere "Begrenzung" eines Rahmens
      bgnd
      bra.l   HEX_ERROR
    ***
    ******************************************************


    ******************************************************
330 ***
    *** Name....: CHKSUM_ERROR
    *** Funktion: Mitteilung: "Die Ubertragung war fehlerhaft"
    *** Register: D4-D7 werden benoetigt fuer zu uebergebende Daten.
    ***
    CHKSUM_ERROR
      move.w #$2700,sr              * Interrupt-Level = 7, Supervisor Mode
      move.b #$01,pic0              * "Neustart"-Signal an die 4 PICs
      move.b #$01,pic1
      move.b #$01,pic2
340   move.b #$01,pic3
      moveq   #02,d7
      bsr.l   SEND_CR
```

```
      moveq  #05,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      moveq  #69,d7
      bsr.l  SEND_UP_LINE            * obere "Begrenzung" eines Rahmens
      moveq  #05,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.l #$694420BA,d4           * "| Di"
350   move.l #$65552065,d5           * "e Ue"
      move.l #$74726562,d6           * "bert"
      move.l #$75676172,d7           * "ragu"
      bsr.l  SEND_CHB_4LONG
      move.l #$7720676E,d4           * "ng w"
      move.l #$66207261,d5           * "ar f"
      move.l #$656C6865,d6           * "ehle"
      move.l #$66616872,d7           * "rhaf"
      bsr.l  SEND_CHB_4LONG
      move.b #$74,d4                 * = "t"
360   bsr.l  SEND_CHB_BYTE
      move.b #$2E,d4                 * = "."
      bsr.l  SEND_CHB_BYTE
      move.b #$20,d4                 * = " "
      bsr.l  SEND_CHB_BYTE
      move.l #$20726544,d4           * "Der "
      move.l #$6F746F4D,d5           * "Moto"
      move.l #$616C6F72,d6           * "rola"
      move.l #$65656220,d7           * " bee"
      bsr.l  SEND_CHB_4LONG
370   move.l #$7465646E,d4           * "ndet"
      move.l #$69657320,d5           * " sei"
      move.l #$4120656E,d6           * "ne A"
      move.l #$69656272,d7           * "rbei"
      bsr.l  SEND_CHB_4LONG
      move.l #$BA202E74,d4           * "t. |"
      bsr.l  SEND_CHB_1LONG
      move.b #$0D,d4                 * = "CR"
      bsr.l  SEND_CHB_BYTE
      moveq  #05,d7
380   bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      moveq  #69,d7
      bsr.l  SEND_LO_LINE            * untere "Begrenzung" eines Rahmens
      bgnd
      bra.l  CHKSUM_ERROR
    ***
    ******************************************************


    ******************************************************
390 ***
    *** Name....: GET_PRG_MESSAGE
    *** Funktion: Aufforderung zur Datenuebertragung
    *** Register: D4-D7 zur Datenuebertragung
    ***
    GET_PRG_MESSAGE
      moveq  #04,d7
      bsr.l  SEND_CR
      moveq  #10,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
400   moveq  #49,d7
      bsr.l  SEND_UP_LINE            * obere "Begrenzung" eines Rahmens
      moveq  #10,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
      moveq  #01,d7
      bsr.l  SEND_SPACE              * d7-viele Leerzeichen
      move.l #$74746942,d4           * "Bitt"
      move.l #$65732065,d5           * "e se"
410   move.l #$6E65646E,d6           * "nden"
      move.l #$65695320,d7           * " Sie"
      bsr.l  SEND_CHB_4LONG
      move.l #$736C6120,d4           * " als"
      move.l #$78655420,d5           * " tex"
      move.l #$74616474,d6           * "tdat"
```

```
      move.l #$49206965,d7          * "ei l"
      bsr.l  SEND_CHB_4LONG
      move.l #$50207268,d4          * "hr P"
      move.l #$502D4349,d5          * "IC-P"
420   move.l #$72676F72,d6          * "rogr"
      move.l #$206D6D61,d7          * "amm "
      bsr.l  SEND_CHB_4LONG
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE
      moveq  #10,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.b #$BA,d4                * = "|"
430   bsr.l  SEND_CHB_BYTE
      move.l #$704F2820,d4          * " (Op"
      move.l #$6E6F6974,d5          * "tion"
      move.l #$464C2220,d6          * " "LF"
      move.l #$656F6C20,d7          * " loe"
      bsr.l  SEND_CHB_4LONG
      move.l #$65686373,d4          * "sche"
      move.l #$6E20226E,d5          * "n" n"
      move.l #$74686369,d6          * "icht"
      move.l #$746B6120,d7          * " akt"
440   bsr.l  SEND_CHB_4LONG
      move.l #$65697669,d4          * "ivie"
      bsr.l  SEND_CHB_1LONG
      move.l #$216E6572,d4          * "ren!"
      bsr.l  SEND_CHB_1LONG
      move.l #$20202E29,d4          * "). "
      bsr.l  SEND_CHB_1LONG
      moveq  #05,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.b #$BA,d4                * = "|"
450   bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE
      moveq  #10,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      moveq  #49,d7
      bsr.l  SEND_LO_LINE           * untere "Begrenzung" eines Rahmens
      moveq  #02,d7
      bsr.l  SEND_CR
      rts
460 ***
    ******************************************************


    ******************************************************
    ***
    *** Name....: HOW_MANY_PRGS_MESS
    *** Funktion: "Wollen Sie mehrere Programme (j/n)?"
    *** Register: D4-D7 zur Datenuebertragung
    ***
470 HOW_MANY_PRGS_MESS
      moveq  #02,d7
      bsr.l  SEND_CR
      moveq  #10,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$6C6C6F57,d4          * "Woll"
      move.l #$53206E65,d5          * "en S"
      move.l #$6D206569,d6          * "ie m"
      move.l #$65726865,d7          * "ehre"
      bsr.l  SEND_CHB_4LONG
480   move.l #$50206572,d4          * "re P"
      move.l #$72676F72,d5          * "rogr"
      move.l #$656D6D61,d6          * "amme"
      move.l #$2F6A2820,d7          * " (j/"
      bsr.l  SEND_CHB_4LONG
      move.l #$203F296E,d4          * "n)? "
      bsr.l  SEND_CHB_1LONG
      rts
    ***
```

```
    ******************************************************
490

    ******************************************************
    ***
    *** Name....:  PRG_TO_PICx_MESS
    *** Funktion: Aufforderung zur Datenuebertragung an PICx
    *** Register: D4-D7 zur Datenuebertragung,
    ***           D2 muss die Nummer von PIC x enthalten
    ***
    PRG_TO_PICx_MESS
500   moveq  #10,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      moveq  #11,d7
      bsr.l  SEND_UP_LINE           * obere "Begrenzung" eines Rahmens
      moveq  #10,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      moveq  #01,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
510   move.l #$50206E61,d4          * "an P"
      bsr.l  SEND_CHB_1LONG
      move.b #$49,d4                * = "I"
      bsr.l  SEND_CHB_BYTE
      move.b #$43,d4                * = "C"
      bsr.l  SEND_CHB_BYTE
      move.b #$20,d4                * = " "
      bsr.l  SEND_CHB_BYTE
      move.b d2,d4
      addi.b #$30,d4
520   bsr.l  SEND_CHB_BYTE
      move.b #$3A,d4                * = ":"
      bsr.l  SEND_CHB_BYTE
      move.b #$20,d4                * = " "
      bsr.l  SEND_CHB_BYTE
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE
      moveq  #10,d7
530   bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      moveq  #11,d7
      bsr.l  SEND_LO_LINE           * untere "Begrenzung" eines Rahmens
      moveq  #02,d7
      bsr.l  SEND_CR
      rts
    ***
    ******************************************************


540 ******************************************************
    ***
    *** Name....:  DATASIZE_MESSG
    *** Funktion: Groesse der auszutauschenden Daten abfragen
    *** Register: D4-D7 zur Datenuebertragung
    ***
    DATASIZE_MESSG
      moveq   #01,d7
      bsr.l   SEND_CR
      moveq   #10,d7
550   bsr.l   SEND_SPACE            * d7-viele Leerzeichen
      moveq   #70,d7
      bsr.l   SEND_UP_LINE          * obere "Begrenzung" eines Rahmens
      moveq   #10,d7
      bsr.l   SEND_SPACE            * d7-viele Leerzeichen
      move.b  #$BA,d4               * = "|"
      bsr.l   SEND_CHB_BYTE
      move.l  #$65472020,d4         * "  Ge"
      move.l  #$206E6562,d5         * "ben "
      move.l  #$20656953,d6         * "Sie "
560   move.l  #$20656964,d7         * "die "
      bsr.l   SEND_CHB_4LONG
```

```
        move.l  #$65746144,d4           * "Date"
        move.l  #$6F72676E,d5           * "ngro"
        move.l  #$65737365,d6           * "esse"
        move.l  #$777A6220,d7           * " bzw"
        bsr.l   SEND_CHB_4LONG
        move.l  #$6964202E,d4           * ". di"
        move.l  #$68432065,d5           * "e Ch"
        move.l  #$6F6D6F72,d6           * "romo"
570     move.l  #$656D6F73,d7           * "some"
        bsr.l   SEND_CHB_4LONG
        move.l  #$6F72676E,d4           * "ngro"
        bsr.l   SEND_CHB_1LONG
        move.l  #$65737365,d4           * "esse"
        bsr.l   SEND_CHB_1LONG
        move.l  #$2E6E6120,d4           * " an."
        bsr.l   SEND_CHB_1LONG
        moveq   #10,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
580     move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE
        moveq   #10,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.l  #$69442020,d4           * "  Di"
        bsr.l   SEND_CHB_1LONG
590     move.l  #$20657365,d4           * "ese "
        bsr.l   SEND_CHB_1LONG
        move.l  #$65747942,d4           * "Byte"
        bsr.l   SEND_CHB_1LONG
        move.b  #$61,d4
        bsr.l   SEND_CHB_BYTE
        move.l  #$68617A6E,d4           * "nzah"
        bsr.l   SEND_CHB_1LONG
        move.l  #$6562206C,d4           * "l be"
        move.l  #$6865697A,d5           * "zieh"
600     move.l  #$69732074,d6           * "t si"
        move.l  #$61206863,d7           * "ch a"
        bsr.l   SEND_CHB_4LONG
        move.l  #$64206675,d4           * "uf d"
        move.l  #$44206E65,d5           * "en D"
        move.l  #$6E657461,d6           * "aten"
        move.l  #$74737561,d7           * "aust"
        bsr.l   SEND_CHB_4LONG
        move.l  #$63737561,d4           * "ausc"
        move.l  #$777A2068,d5           * "h zw"
610     move.l  #$68637369,d6           * "isch"
        move.l  #$64206E65,d7           * "en d"
        bsr.l   SEND_CHB_4LONG
        move.l  #$20206E65,d4           * "en  "
        bsr.l   SEND_CHB_1LONG
        moveq   #01,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
620     bsr.l   SEND_CHB_BYTE
        moveq   #10,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        moveq   #02,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        move.l  #$73434950,d4           * "PICs"
        move.l  #$646E7520,d5           * " und"
        move.l  #$73756D20,d6           * " mus"
630     move.l  #$6C612073,d7           * "s al"
        bsr.l   SEND_CHB_4LONG
        move.l  #$62206F73,d4           * "so b"
        move.l  #$61206965,d5           * "ei a"
        move.l  #$6E656C6C,d6           * "llen"
```

```
      move.l  #$70704120,d7          * " App"
      bsr.l   SEND_CHB_4LONG
      move.l  #$616B696C,d4          * "lika"
      move.l  #$6E6F6974,d5          * "tion"
      move.l  #$67206E65,d6          * "en g"
640   move.l  #$6369656C,d7          * "leic"
      bsr.l   SEND_CHB_4LONG
      move.l  #$65672068,d4          * "h ge"
      move.l  #$68656177,d5          * "waeh"
      move.l  #$7720746C,d6          * "lt w"
      move.l  #$65647265,d7          * "erde"
      bsr.l   SEND_CHB_4LONG
      move.l  #$20202E6E,d4          * "n.  "
      bsr.l   SEND_CHB_1LONG
      move.b  #$BA,d4                * = "|"
650   bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4                * = "CR"
      bsr.l   SEND_CHB_BYTE
      moveq   #10,d7
      bsr.l   SEND_SPACE             * d7-viele Leerzeichen
      move.b  #$BA,d4                * = "|"
      bsr.l   SEND_CHB_BYTE
      moveq   #02,d7
      bsr.l   SEND_SPACE             * d7-viele Leerzeichen
      move.l  #$61676E41,d4          * "Anga"
660   move.l  #$62206562,d5          * "be b"
      move.l  #$65747469,d6          * "itte"
      move.l  #$206E6920,d7          * " in "
      bsr.l   SEND_CHB_4LONG
      move.l  #$422D3631,d4          * "16-B"
      move.l  #$53207469,d5          * "it S"
      move.l  #$69726863,d6          * "chri"
      move.l  #$6E657474,d7          * "tten"
      bsr.l   SEND_CHB_4LONG
      move.l  #$646E7520,d4          * " und"
670   move.l  #$65726420,d5          * " dre"
      move.l  #$65747369,d6          * "iste"
      move.l  #$67696C6C,d7          * "llig"
      bsr.l   SEND_CHB_4LONG
      move.b  #$2E,d4                * = "."
      bsr.l   SEND_CHB_BYTE
      moveq   #19,d7
      bsr.l   SEND_SPACE             * d7-viele Leerzeichen
      move.b  #$BA,d4                * = "|"
      bsr.l   SEND_CHB_BYTE
680   move.b  #$0D,d4                * = "CR"
      bsr.l   SEND_CHB_BYTE
      moveq   #10,d7
      bsr.l   SEND_SPACE             * d7-viele Leerzeichen
      moveq   #70,d7
      bsr.l   SEND_LO_LINE           * untere "Begrenzung" eines Rahmens
      move.b  #$0D,d4                * = "CR"
      bsr.l   SEND_CHB_BYTE
      moveq   #39,d7
      bsr.l   SEND_SPACE             * d7-viele Leerzeichen
690   rts
    ***
    ******************************************************


    ******************************************************
    ***
    *** Name....:  TOPO_INTRO_MESS
    *** Funktion: Festlegen der Topologie
    *** Register: D4-D7 zur Datenuebertragung
700 ***
    TOPO_INTRO_MESS
      moveq   #04,d7
      bsr.l   SEND_CR
      moveq   #20,d7
      bsr.l   SEND_SPACE             * d7-viele Leerzeichen
      moveq   #18,d7
      bsr.l   SEND_UP_LINE           * obere "Begrenzung" eines Rahmens
```

```
      moveq   #20,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
710   move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      moveq   #02,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.l  #$65746144,d4           * "Date"
      move.l  #$7375616E,d5           * "naus"
      move.l  #$73756174,d6           * "taus"
      move.l  #$20206863,d7           * "ch  "
      bsr.l   SEND_CHB_4LONG
      move.b  #$BA,d4                 * = "|"
720   bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE
      moveq   #20,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      moveq   #18,d7
      bsr.l   SEND_LO_LINE            * untere "Begrenzung" eines Rahmens
      moveq   #02,d7
      bsr.l   SEND_CR
      rts
730 ***
    ******************************************************


    ******************************************************
    ***
    *** Name....: TOPO_DATA_MESS
    *** Funktion: Festlegen der Topologie
    *** Register: D4-D7 zur Datenuebertragung,
    ***           D0 enthaelt Datenuebermittelnden PIC (d.h. dessen ID),
740 ***           D1 enthaelt Datenempfangenden PIC
    ***
    TOPO_DATA_MESS
      moveq   #10,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.l  #$6C6C6F53,d4           * "Soll"
      bsr.l   SEND_CHB_1LONG
      move.l  #$43495020,d4           * " PIC"
      bsr.l   SEND_CHB_1LONG
      move.b  d0,d4
750   addi.b  #$30,d4
      bsr.l   SEND_CHB_BYTE
      move.l  #$74614420,d4           * " Dat"
      bsr.l   SEND_CHB_1LONG
      move.l  #$61206E65,d4           * "en a"
      bsr.l   SEND_CHB_1LONG
      move.l  #$4950206E,d4           * "n PI"
      bsr.l   SEND_CHB_1LONG
      move.b  #$43,d4                 * = "C"
      bsr.l   SEND_CHB_BYTE
760   move.b  d1,d4
      addi.b  #$30,d4
      bsr.l   SEND_CHB_BYTE
      move.l  #$62656720,d4           * " geb"
      move.l  #$28206E65,d5           * "en ("
      move.l  #$296E2F6A,d6           * "j/n)"
      move.l  #$2020203F,d7           * "?   "
      bsr.l   SEND_CHB_4LONG
      rts
    ***
770 ******************************************************


    ******************************************************
    ***
    *** Name....: PIC_START_MESS
    *** Funktion: Start der PIC-Programme
    *** Register: D4-D7 zur Datenuebertragung
    ***
    PIC_START_MESS
780   moveq   #04,d7
```

```
      bsr.l   SEND_CR
      moveq   #10,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      moveq   #48,d7
      bsr.l   SEND_UP_LINE            * obere "Begrenzung" eines Rahmens
      moveq   #10,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
790   move.l  #$65694420,d4           * " Die"
      move.l  #$43495020,d5           * " PIC"
      move.l  #$6F72502D,d6           * "-Pro"
      move.l  #$7373657A,d7           * "zess"
      bsr.l   SEND_CHB_4LONG
      move.l  #$6E65726F,d4           * "oren"
      move.l  #$61747320,d5           * " sta"
      move.l  #$6E657472,d6           * "rten"
      move.l  #$65696420,d7           * " die"
      bsr.l   SEND_CHB_4LONG
800   move.l  #$61624120,d4           * " Aba"
      move.l  #$69656272,d5           * "rbei"
      move.l  #$676E7574,d6           * "tung"
      move.l  #$202E2E2E,d7           * "... "
      bsr.l   SEND_CHB_4LONG
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE
      moveq   #10,d7
810   bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      moveq   #48,d7
      bsr.l   SEND_LO_LINE            * untere "Begrenzung" eines Rahmens
      rts
***
*********************************************************


*********************************************************
***
820 *** Name....: RESTART_MESS
*** Funktion: "Moechten Sie einen Neustart (j/n)?"
*** Register: D4-D7 zur Datenuebertragung
***
RESTART_MESS
      moveq   #02,d7
      bsr.l   SEND_CR
      moveq   #10,d7
      bsr.l   SEND_SPACE              * d7-viele Leerzeichen
      move.l  #$63656F4D,d4           * "Moec"
830   move.l  #$6E657468,d5           * "hten"
      move.l  #$65695320,d6           * " Sie"
      move.l  #$6E696520,d7           * " ein"
      bsr.l   SEND_CHB_4LONG
      move.l  #$4E206E65,d4           * "en N"
      move.l  #$74737565,d5           * "eust"
      move.l  #$20747261,d6           * "art "
      move.l  #$6E2F6A28,d7           * "(j/n"
      bsr.l   SEND_CHB_4LONG
      move.l  #$20203F29,d4           * ")?  "
840   bsr.l   SEND_CHB_1LONG
      bsr.l   RECEIVERB_LOOP          * Auf Signal warten
      rts
***
*********************************************************


*********************************************************
***
*** Name....: SHOW_RES_INTRO
850 *** Funktion: "Ergebnis von PICx"
*** Register: D4 - D7
***
SHOW_RES_INTRO
```

```
      moveq  #02,d7                  * = "CR"
      bsr.l  SEND_CR
      moveq  #10,d7
      bsr.l  SEND_SPACE
      moveq  #65,d7
      bsr.l  SEND_UP_LINE
860   moveq  #10,d7
      bsr.l  SEND_SPACE
      move.l #$724520BA,d4           * "| Er"
      move.l #$6E626567,d5           * "gebn"
      move.l #$76207369,d6           * "is v"
      move.l #$50206E6F,d7           * "on P"
      bsr.l  SEND_CHB_4LONG
      move.b #$49,d4                 * "I"
      bsr.l  SEND_CHB_BYTE
      move.b #$43,d4                 * "C"
870   bsr.l  SEND_CHB_BYTE
      move.b #$20,d4                 * " "
      bsr.l  SEND_CHB_BYTE
      rts
    ***
    ********************************************************


    ********************************************************
    ***
880 *** Name....: SHOW_NR_GEN
    *** Funktion: "Anzahl an Generationen"
    *** Register: D4 - D7
    ***
    SHOW_NR_GEN
      moveq  #10,d7
      bsr.l  SEND_SPACE
      move.l #$412020BA,d4           * "|  A"
      move.l #$68617A6E,d5           * "nzah"
      move.l #$6E61206C,d6           * "l an"
890   move.l #$6E654720,d7           * " Gen"
      bsr.l  SEND_CHB_4LONG
      move.l #$74617265,d4           * "erat"
      move.l #$656E6F69,d5           * "ione"
      move.l #$2020206E,d6           * "n   "
      move.l #$20202020,d7           * "    "
      bsr.l  SEND_CHB_4LONG
      move.b #$10,d7
      bsr.l  SEND_SPACE
      move.b #$3A,d4                 * ":"
900   bsr.l  SEND_CHB_BYTE
      move.b #$03,d7
      bsr.l  SEND_SPACE
      rts
    ***
    ********************************************************


    ********************************************************
    ***
910 *** Name....: SHOW_NR_CHILDS
    *** Funktion: "Anzahl Nachkommen pro Generation"
    *** Register: D4 - D7
    ***
    SHOW_NR_CHILDS
      moveq  #10,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
      moveq  #02,d7
920   bsr.l  SEND_SPACE
      move.l #$617A6E41,d4           * "Anza"
      bsr.l  SEND_CHB_1LONG
      move.l #$4E206C68,d4           * "hl N"
      bsr.l  SEND_CHB_1LONG
      move.l #$6B686361,d4           * "achk"
      bsr.l  SEND_CHB_1LONG
```

```
      move.l #$656D6D6F,d4          * "omme"
      move.l #$7270206E,d5          * "n pr"
      move.l #$6547206F,d6          * "o Ge"
930   move.l #$6172656E,d7          * "nera"
      bsr.l  SEND_CHB_4LONG
      move.l #$6E6F6974,d4          * "tion"
      bsr.l  SEND_CHB_1LONG
      move.b #$0D,d7
      bsr.l  SEND_SPACE
      move.b #$3A,d4
      bsr.l  SEND_CHB_BYTE
      move.b #$03,d7
      bsr.l  SEND_SPACE
940   move.b #$30,d4                * "0"
      bsr.l  SEND_CHB_BYTE
      rts
    ***
    ********************************************************


    ********************************************************
    ***
    *** Name....: SHOW_POPSIZE
950 *** Funktion: "Groesse der Population"
    *** Register: D4 - D7
    ***
    SHOW_POPSIZE
      moveq  #10,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      moveq  #02,d7
      bsr.l  SEND_SPACE
960   move.l #$656F7247,d4          * "Groe"
      bsr.l  SEND_CHB_1LONG
      move.l #$20657373,d4          * "sse "
      bsr.l  SEND_CHB_1LONG
      move.l #$20726564,d4          * "der "
      bsr.l  SEND_CHB_1LONG
      move.l #$75706F50,d4          * "Popu"
      move.l #$6974616C,d5          * "lati"
      move.l #$20206E6F,d6          * "on  "
      move.l #$20202020,d7          * "    "
970   bsr.l  SEND_CHB_4LONG
      move.b #$11,d7
      bsr.l  SEND_SPACE
      move.b #$3A,d4
      bsr.l  SEND_CHB_BYTE
      move.b #$03,d7
      bsr.l  SEND_SPACE
      move.b #$30,d4                * "0"
      bsr.l  SEND_CHB_BYTE
      rts
980 ***
    ********************************************************


    ********************************************************
    ***
    *** Name....: SHOW_CHROMOSIZE
    *** Funktion: "Groesse der Chromosome"
    *** Register: D4 - D7
    ***
990 SHOW_CHROMOSIZE
      moveq  #10,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      moveq  #02,d7
      bsr.l  SEND_SPACE
      move.l #$656F7247,d4          * "Groe"
      bsr.l  SEND_CHB_1LONG
      move.l #$20657373,d4          * "sse "
```

```
1000    bsr.l   SEND_CHB_1LONG
        move.l  #$20726564,d4           * "der "
        bsr.l   SEND_CHB_1LONG
        move.l  #$6F726843,d4           * "Chro"
        move.l  #$6F736F6D,d5           * "moso"
        move.l  #$2020656D,d6           * "me  "
        move.l  #$20202020,d7           * "    "
        bsr.l   SEND_CHB_4LONG
        move.b  #$11,d7
        bsr.l   SEND_SPACE
1010    move.b  #$3A,d4
        bsr.l   SEND_CHB_BYTE
        move.b  #$03,d7
        bsr.l   SEND_SPACE
        move.b  #$30,d4                 * "0"
        bsr.l   SEND_CHB_BYTE
        rts
      ***
      ******************************************************

1020
      ******************************************************
      ***
      *** Name....: SHOW_FITNESS
      *** Funktion: "Fitness des besten Chromosomes"
      *** Register: D4 - D7
      ***
      SHOW_FITNESS
        moveq   #10,d7
        bsr.l   SEND_SPACE
1030    move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        moveq   #02,d7
        bsr.l   SEND_SPACE
        move.l  #$6E746946,d4           * "Fitn"
        bsr.l   SEND_CHB_1LONG
        move.l  #$20737365,d4           * "ess "
        bsr.l   SEND_CHB_1LONG
        move.l  #$20736564,d4           * "des "
        bsr.l   SEND_CHB_1LONG
1040    move.l  #$74736562,d4           * "best"
        move.l  #$43206E65,d5           * "en C"
        move.l  #$6D6F7268,d6           * "hrom"
        move.l  #$6D6F736F,d7           * "osom"
        bsr.l   SEND_CHB_4LONG
        move.l  #$20202073,d4           * "s   "
        bsr.l   SEND_CHB_1LONG
        move.b  #$0D,d7
        bsr.l   SEND_SPACE
        move.b  #$3A,d4
1050    bsr.l   SEND_CHB_BYTE
        move.b  #$03,d7
        bsr.l   SEND_SPACE
        move.b  #$30,d4                 * "0"
        bsr.l   SEND_CHB_BYTE
        rts
      ***
      ******************************************************


1060  ******************************************************
      ***
      *** Name....: SHOW_BEST_RES
      *** Funktion: "Beste Loesung"
      *** Register: D4 - D7
      ***
      SHOW_BEST_RES
        moveq   #10,d7
        bsr.l   SEND_SPACE
        move.b  #$BA,d4                 * = "|"
1070    bsr.l   SEND_CHB_BYTE
        moveq   #02,d7
        bsr.l   SEND_SPACE
```

```
      move.l #$74736542,d4          * "Best"
      bsr.l  SEND_CHB_1LONG
      move.l #$6F4C2065,d4          * "e Lo"
      bsr.l  SEND_CHB_1LONG
      move.l #$6E757365,d4          * "esun"
      bsr.l  SEND_CHB_1LONG
      move.l #$20203A67,d4          * "g:  "
1080  bsr.l  SEND_CHB_1LONG
      rts
*****
*******************************************************


*******************************************************
*****
***** Name....: LONG_NO_DATA_MESSG
***** Funktion: "Ausfuehrliche" Anzeige der uebermittelten
1090 *****          Daten eines PICs, der keine neuen Daten erhaelt.
***** Register: D4-D7 zur Datenuebertragung,
*****          A5 um ChromosomenAdresse anzugeben,
*****          D1 als Identitaet der PICs (0=PIC0, 1=PIC1, ...),
*****          D2 speichert Fitness-Wert des aktuellen Chromosomes.
*****          D3 speichert momentane Generation
*****
LONG_NO_DATA_MESSG
      moveq  #02,d7
      bsr.l  SEND_CR                * d7-viele "CR"
1100  moveq  #10,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      moveq  #25,d7
      bsr.l  SEND_UP_LINE           * obere "Begrenzung" eines Rahmens
      moveq  #10,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$202020BA,d4          * "|   "
      bsr.l  SEND_CHB_1LONG
      move.l #$20434950,d4          * "PIC "
      bsr.l  SEND_CHB_1LONG
1110  move.b d1,d4
      addi.b #$30,d4
      bsr.l  SEND_CHB_BYTE
      move.l #$44202D20,d4          * " - D"
      bsr.l  SEND_CHB_1LONG
      move.l #$6E657461,d4          * "aten"
      bsr.l  SEND_CHB_1LONG
      move.b #$3A,d4                * = ":"
      bsr.l  SEND_CHB_BYTE
      moveq  #08,d7
1120  bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE
      moveq  #10,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      moveq  #25,d7
      bsr.l  SEND_MI_LINE           * mittlere "Begrenzung" eines Rahmens

1130  moveq  #10,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
      move.l #$202020BA,d4          * "|   "
      bsr.l  SEND_CHB_1LONG
      move.l #$6E746946,d4          * "Fitn"
      bsr.l  SEND_CHB_1LONG
      move.l #$3A737365,d4          * "ess:"
      bsr.l  SEND_CHB_1LONG
      moveq  #05,d7
      bsr.l  SEND_SPACE             * d7-viele Leerzeichen
1140
      moveq  #00,d0
      move.w #$FFFF,d0
      sub.w  d2,d0                  * D2 speichert Fitness-Wert
      moveq  #00,d1
      moveq  #00,d2
```

```
        bsr.l   HEX2DEZ
        bsr.l   DECODE_DEZ

        moveq   #04,d7
1150    bsr.l   SEND_SPACE
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE

        moveq   #10,d7
        bsr.l   SEND_SPACE
        move.l  #$202020BA,d4           * "|   "
        move.l  #$656E6547,d5           * "Gene"
1160    move.l  #$69746172,d6           * "rati"
        move.l  #$203A6E6F,d7           * "on: "
        bsr.l   SEND_CHB_4LONG
        move.l  d3,d0                   * D2 speichert Generationen-Wert
        moveq   #00,d1
        moveq   #00,d2
        bsr.l   HEX2DEZ_LONG
        bsr.l   DECODE_DEZ_LONG
        moveq   #04,d7
        bsr.l   SEND_SPACE
1170    move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE

        moveq   #10,d7
        bsr.l   SEND_SPACE
        moveq   #25,d7
        bsr.l   SEND_LO_LINE            * "untere" Begrenzungslinie
        rts
1180 ***
     ******************************************************


     ******************************************************
     ***
     *** Name....: LONG_DATA_MESSG
     *** Funktion: "Ausfuehrliche" Anzeige der uebermittelten
     ***           Daten eines PICs, der neue Daten erhaelt.
     *** Register: D4-D7 zur Datenuebertragung,
1190 ***           A6 um ChromosomenAdresse anzugeben,
     ***           D3 speichert Fitness-Wert des aktuellen Chromosomes,
     ***           "receiver_pic" gibt datenempfangenden PIC an,
     ***           "transmitter_pic" gibt datenuebermittelnden PIC an.
     ***
     LONG_DATA_MESSG
        moveq   #10,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        moveq   #25,d7
        bsr.l   SEND_UP_LINE            * obere "Begrenzung" eines Rahmens
1200    moveq   #10,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        move.l  #$202020BA,d4           * "|   "
        bsr.l   SEND_CHB_1LONG
        move.l  #$20434950,d4           * "PIC "
        bsr.l   SEND_CHB_1LONG
        move.b  transmitter_pic,d4
        addi.b  #$30,d4
        bsr.l   SEND_CHB_BYTE
        move.l  #$203E2D20,d4           * " -> "
1210    bsr.l   SEND_CHB_1LONG
        move.l  #$20434950,d4           * "PIC "
        bsr.l   SEND_CHB_1LONG
        move.b  receiver_pic,d4
        addi.b  #$30,d4
        bsr.l   SEND_CHB_BYTE
        move.b  #$3A,d4                 * = ":"
        bsr.l   SEND_CHB_BYTE
        moveq   #07,d7
```

```
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
1220    move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE
        moveq   #10,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        moveq   #25,d7
        bsr.l   SEND_MI_LINE            * mittlere "Begrenzung" eines Rahmens

        moveq   #10,d7
1230    bsr.l   SEND_SPACE              * d7-viele Leerzeichen
        move.l  #$202020BA,d4           * "|   "
        bsr.l   SEND_CHB_1LONG
        move.l  #$6E746946,d4           * "Fitn"
        bsr.l   SEND_CHB_1LONG
        move.l  #$3A737365,d4           * "ess:"
        bsr.l   SEND_CHB_1LONG
        moveq   #02,d7
        bsr.l   SEND_SPACE              * d7-viele Leerzeichen

1240    moveq   #00,d0
        move.w  #$FFFF,d0
        sub.w   d3,d0                   * D3 speichert Fitness-Wert
        moveq   #00,d1
        moveq   #00,d2
        bsr.l   HEX2DEZ
        bsr.l   DECODE_DEZ

        moveq   #07,d7
        bsr.l   SEND_SPACE
1250    move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 *  "CR"
        bsr.l   SEND_CHB_BYTE
        moveq   #10,d7
        bsr.l   SEND_SPACE
        moveq   #25,d7
        bsr.l   SEND_LO_LINE            * "untere" Begrenzungslinie
        rts
    ***
1260 ****************************************************
```

B.17 Auswertung der PIC17C43-Interrupts: `irq.a`

```
    ************************************************************************
    ***             Bearbeiten der eingehenden Interrupts                ***
    *** Autor:       Tobias Schubert                                      ***
    *** EMail:       schubert@informatik.uni-freiburg.de                  ***
    *** Datum:       10.09.1999                                           ***
    *** Datei-Name:  irq.a                                                ***
    ************************************************************************

    ************************************************************************
 10 ***                                                                  ***
    ***     Definition: "Mode"-Register des MC68340                      ***
    ***     ------------------------------------                         ***
    ***                                                                  ***
    ***     Wert:                   Bedeutung:                           ***
    ***     ---------------------------------------------------------    ***
    ***     0000 0000 = 000         Warten auf Speichertest PIC0         ***
    ***     0000 0001 = 001         Warten auf Speichertest PIC1         ***
    ***     0000 0010 = 002         Warten auf Speichertest PIC2         ***
    ***     0000 0011 = 003         Warten auf Speichertest PIC3         ***
 20 ***     0000 0100 = 004         Verteilen der PIC-Programme          ***
    ***     0000 0101 = 005         Grundmodus: Chromosome verteilen     ***
    ***     0000 1010 = 010         Anfrage von PIC0: Empfang d. Daten   ***
```

```
***      0000 1011 = 011          Anfrage von PIC1: Empfang d. Daten    ***
***      0000 1100 = 012          Anfrage von PIC2: Empfang d. Daten    ***
***      0000 1101 = 013          Anfrage von PIC3: Empfang d. Daten    ***
***                                                                     ***
************************************************************************

************************************************************************
*** 
*** Name....: picx_interrupt
*** Funktion: Bearbeiten der eingehenden Interrupts gemaess Motorola-Modus
*** Register: D0 - D3 zur Weitergabe eingehender Daten
***
picx_interrupt
  move.w #$2400,sr              * Interrupt Level = 4, Supervisor Mode

  cmp.b  #$00,mode              * Empfang der Speichertestergebnisse
  beq    READ_SRAM_MESSAGE
  cmp.b  #$01,mode
  beq    READ_SRAM_MESSAGE
  cmp.b  #$02,mode
  beq    READ_SRAM_MESSAGE
  cmp.b  #$03,mode
  beq    READ_SRAM_MESSAGE

  cmp.b  #$04,mode              * PIC-Programme verschicken
  beq    SEND_PRG_INTERRUPT

  cmp.b  #$05,mode              * "Normal"-Modus
  beq    COMMUNICATION

  cmp.b  #$0A,mode
  beq    PIC0_DATA_CHANGE       * Daten an/von PIC0
  cmp.b  #$0B,mode
  beq    PIC1_DATA_CHANGE       * Daten an/von PIC1
  cmp.b  #$0C,mode
  beq    PIC2_DATA_CHANGE       * Daten an/von PIC2
  cmp.b  #$0D,mode
  beq    PIC3_DATA_CHANGE       * Daten an/von PIC3
  rte


*** Empfang von Speichertestergebnis von PIC0 ... PIC3
READ_SRAM_MESSAGE
  move.b pic0,d0
  move.b pic1,d1
  move.b pic2,d2
  move.b pic3,d3
  addi.b #01,mode
  rte


*** an die PICs Programme verschicken, diese antworten mit $01
SEND_PRG_INTERRUPT
  move.b pic0,d0
  move.b pic1,d1
  move.b pic2,d2
  move.b pic3,d3
  rte


*** Aus dem "Normal"-Modus heraus Steuersignale empfangen
COMMUNICATION
  move.b pic0,control0
  move.b pic1,control1
  move.b pic2,control2
  move.b pic3,control3
  moveq  #00,d3
  move.b #$01,d3                * Als Bestaetigung beim Motorola
  rte


*** Motorola tauscht Daten mit PIC0 aus
PIC0_DATA_CHANGE
```

```
  move.b pic0,d0                * 1 Byte Daten
  move.b #$01,d3                * Als Bestaetigung beim Motorola
  rte

100
*** Motorola tauscht Daten mit PIC1 aus
PIC1_DATA_CHANGE
  move.b pic1,d0                * 1 Byte Daten
  move.b #$01,d3                * Als Bestaetigung beim Motorola
  rte


*** Motorola tauscht Daten mit PIC2 aus
PIC2_DATA_CHANGE
110  move.b pic2,d0             * 1 Byte Daten
  move.b #$01,d3                * Als Bestaetigung beim Motorola
  rte


*** Motorola tauscht Daten mit PIC3 aus
PIC3_DATA_CHANGE
  move.b pic3,d0                * 1 Byte Daten
  move.b #$01,d3                * Als Bestaetigung beim Motorola
  rte
120 ***
*************************************************************************
```

B.18 Datenaustausch zwischen den Prozessoren: `change.a`

```
*************************************************************************
***             Datenaustausch zwischen den PICs                      ***
*** Autor:      Tobias Schubert                                       ***
*** EMail:      schubert@informatik.uni-freiburg.de                   ***
*** Datum:      24.10.1999                                            ***
*** Datei-Name: change.a                                              ***
*************************************************************************


 10 *************************************************************************
***
*** Name....: CHANGE_RESULTS_PIC0
*** Funktion: Daten von/an PIC0.
*** Register: A5/A6 als Startadresse fuer Chromosome,
***           D0 - D7 als Datenregister.
***
CHANGE_RESULTS_PIC0
  moveq  #00,d2                 * Chromosomengroesse in D2 speichern
  move.b chr_sizeH,d2
 20  lsl.l  #$08,d2
  add.b  chr_sizeL,d2
  move.l a0,a5                  * a0 = Zeiger auf PIC0-Datenzellen

GET_CHROMO_PIC0
  moveq  #00,d3
  move.b #$00,pic0              * PIC0 zum Datensenden auffordern
CHANGE_PIC0_RESULT_LOOP1
  cmp.l  #$00,d3                * D3 = 01 zeigt IRQ von PIC0 an
  beq    CHANGE_PIC0_RESULT_LOOP1
 30  move.b d0,(a5)+            * D0 = Daten von PIC0
  subi.l #$08,d2                * Chromo-Zaehler um 8 Bit dekrementieren
  cmp.l  #$00,d2
  bne    GET_CHROMO_PIC0

  moveq  #00,d3
  move.b #$00,pic0
CHANGE_PIC0_RESULT_LOOP2
  cmp.l  #$00,d3
```

```
      beq     CHANGE_PIC0_RESULT_LOOP2
40    move.b  d0,fitness0_hi          * Fitness des Chromosomes (High Byte)

      moveq   #00,d3
      move.b  #$00,pic0
    CHANGE_PIC0_RESULT_LOOP3
      cmp.l   #$00,d3
      beq     CHANGE_PIC0_RESULT_LOOP3
      move.b  d0,fitness0_lo          * Fitness des Chromosomes (Low Byte)

      moveq   #00,d3
50    move.b  #$00,pic0
    CHANGE_PIC0_GEN_LOOP0
      cmp.l   #00,d3
      beq     CHANGE_PIC0_GEN_LOOP0
      move.b  d0,gen0_hi              * D0 = Anzahl Generationen (High Byte)

      moveq   #00,d3
      move.b  #$00,pic0
    CHANGE_PIC0_GEN_LOOP1
      cmp.l   #00,d3
60    beq     CHANGE_PIC0_GEN_LOOP1
      move.b  d0,gen0_mi              * D0 = Anzahl Generationen (Middle Byte)

      moveq   #00,d3
      move.b  #$00,pic0
    CHANGE_PIC0_GEN_LOOP2
      cmp.l   #00,d3
      beq     CHANGE_PIC0_GEN_LOOP2
      move.b  d0,gen0_lo              * D0 = Anzahl Generationen (Low Byte)

70    move.b  topo1,d7                * PIC0-Topologie anhand von topo1/2/3
      btst.b  #$00,d7                 * feststellen (vgl. "def.h")
      bne     NEW_DATA_FOR_PIC0
      move.b  topo2,d7
      btst.b  #$00,d7
      bne     NEW_DATA_FOR_PIC0
      move.b  topo3,d7
      btst.b  #$00,d7
      bne     NEW_DATA_FOR_PIC0

80    move.l  a0,a5                   * Startadresse des empfangenen Chromosomes
      moveq   #00,d3                  * D3 speichert Generationen-Anzahl
      moveq   #00,d2                  * D2 speichert Fitness-Wert
      moveq   #00,d1
      move.b  fitness0_hi,d2
      lsl.l   #08,d2
      move.l  d2,d1
      moveq   #00,d2
      move.b  fitness0_lo,d2
      add.l   d1,d2
90    moveq   #00,d0
      move.b  gen0_hi,d0              * Anzahl Generationen
      lsl.l   #08,d0                  * ins High Byte "shiften"
      move.b  gen0_mi,d0
      lsl.l   #08,d0                  * ins High Byte "shiften"
      move.b  gen0_lo,d0
      move.l  d0,d3
      move.b  #$00,d1                 * PIC-Identitaet (PIC0 = 00, PIC1 = 01 ...)
      bsr.l   LONG_NO_DATA_MESSG

100   moveq   #00,d3                  * PLD-Latch zuruecksetzen
      move.b  #$00,pic0
    CLEAR_PIC0_LATCH
      cmp.l   #00,d3
      beq     CLEAR_PIC0_LATCH
      move.b  #$05,mode
      rts


    NEW_DATA_FOR_PIC0
110   move.l  a0,a5                   * Startadresse des empfangenen Chromosomes
      moveq   #00,d3                  * D3 speichert Generationen-Anzahl
```

```
      moveq  #00,d2                  * D2 speichert Fitness-Wert
      moveq  #00,d1
      move.b fitness0_hi,d2
      lsl.l  #08,d2
      move.l d2,d1
      moveq  #00,d2
      move.b fitness0_lo,d2
      add.l  d1,d2
120   moveq  #00,d0
      move.b gen0_hi,d0              * Anzahl Generationen
      lsl.l  #08,d0                  * ins High Byte "shiften"
      move.b gen0_mi,d0
      lsl.l  #08,d0                  * ins High Byte "shiften"
      move.b gen0_lo,d0
      move.l d0,d3
      move.b #$00,d1                 * PIC-Identitaet (PIC0 = 00, PIC1 = 01 ...)
      bsr.l  LONG_NO_DATA_MESSG

130   moveq  #00,d3
      move.b #$01,pic0
    FITNESS_PIC0_LOOP1
      cmp.l  #00,d3
      beq    FITNESS_PIC0_LOOP1

      moveq  #00,d1                  * Fitness-Werte speichern: PIC0 in D0,
      moveq  #00,d2                  * PIC1 in D1, PIC2 in D2, ...
      moveq  #00,d3                  * A5 = Adresse des besten Chromosomes
      moveq  #00,d5                  * D5 = beste Fitness
140   move.b fitness1_hi,d1
      lsl.w  #$08,d1
      add.b  fitness1_lo,d1
      move.b fitness2_hi,d2
      lsl.w  #$08,d2
      add.b  fitness2_lo,d2
      move.b fitness3_hi,d3
      lsl.w  #$08,d3
      add.b  fitness3_lo,d3

150   move.b topo1,d7                * TOPO1 testen:
      btst.b #$00,d7                 * Bit0=1 -> PIC0 erhaelt Daten von PIC1
      beq    HIGH_PART_FITNESS02
      move.w d1,d5
      move.l a1,a5
      move.l a1,a6                   * A6 = Startadresse zur Bildschirmausgabe
      move.b #$01,transmitter_pic    * Gibt datenuebergebenden PIC an
      move.b fitness1_hi,trans_fit_hi
      move.b fitness1_lo,trans_fit_lo

160 HIGH_PART_FITNESS02
      move.b topo2,d7                * TOPO2 testen:
      btst.b #$00,d7                 * Bit0=1 -> PIC0 erhaelt Daten von PIC2
      beq    HIGH_PART_FITNESS03
      cmp.l  d5,d2
      blt    HIGH_PART_FITNESS03
      move.w d2,d5
      move.l a2,a5
      move.l a2,a6                   * A6 = Startadresse zur Bildschirmausgabe
      move.b #$02,transmitter_pic    * Gibt datenuebergebenden PIC an
170   move.b fitness2_hi,trans_fit_hi
      move.b fitness2_lo,trans_fit_lo

    HIGH_PART_FITNESS03
      move.b topo3,d7                * TOPO3 testen:
      btst.b #$00,d7                 * Bit0=1 -> PIC0 erhaelt Daten von PIC3
      beq    HIGH_PART_FITNESS00
      cmp.l  d5,d3
      blt    HIGH_PART_FITNESS00
      move.w d3,d5
180   move.l a3,a5
      move.l a3,a6                   * A6 = Startadresse zur Bildschirmausgabe
      move.b #$03,transmitter_pic    * Gibt datenuebergebenden PIC an
      move.b fitness3_hi,trans_fit_hi
      move.b fitness3_lo,trans_fit_lo
```

```
    HIGH_PART_FITNESS00
      moveq  #00,d3
      move.w d5,d3
      lsr.w  #$08,d5
190   move.b d5,pic0                  * High Part der besten Fitness uebergeben
      move.w d3,d5
      moveq  #00,d3
    FITNESS_PIC0_LOOP2
      cmp.l  #00,d3
      beq    FITNESS_PIC0_LOOP2

      move.b d5,pic0                  * Low Part der besten Fitness uebergeben

      move.b chr_sizeH,d1             * Chromosomengroesse festlegen
200   lsl.w  #$08,d1
      add.b  chr_sizeL,d1

    SEND_NEW_CHROMO0
      moveq  #00,d0
      moveq  #00,d3
    SEND_NEW_CHROMO_LOOP0
      cmp.l  #00,d3
      beq    SEND_NEW_CHROMO_LOOP0

210   move.b (a5)+,d0
      move.b d0,pic0                  * 8 Bit des besten Chromosomes weitergeben

      subi.w #$08,d1
      cmp.w  #$0000,d1
      bne    SEND_NEW_CHROMO0

      move.b #$00,receiver_pic        * PIC-Identitaet (PIC0 = 00, PIC1 = 01 ...)
      moveq  #00,d3                   * D3 speichert Fitness-Wert
      moveq  #00,d1
220   move.b trans_fit_hi,d3          * Fitness-Wert des ausgewaehlten Chromosomes
      lsl.l  #08,d3
      move.l d3,d1
      moveq  #00,d3
      move.b trans_fit_lo,d3
      add.l  d1,d3
      bsr.l  LONG_DATA_MESSG
      rts
    ***
    ************************************************************************
230

    ************************************************************************
    ***
    *** Name....: CHANGE_RESULTS_PIC1
    *** Funktion: Daten von/an PIC1.
    *** Register: A5/A6 als Startadresse fuer Chromosome,
    ***           D0 - D7 als Datenregister.
    ***
    CHANGE_RESULTS_PIC1
240   moveq  #00,d2                   * Chromosomengroesse in D2 speichern
      move.b chr_sizeH,d2
      lsl.l  #08,d2
      add.b  chr_sizeL,d2
      move.l a1,a5                    * a1 = Zeiger auf PIC1-Datenzellen

    GET_CHROMO_PIC1
      moveq  #00,d3
      move.b #$00,pic1                * PIC1 zum Datensenden auffordern
    CHANGE_PIC1_RESULT_LOOP1
250   cmp.l  #$00,d3                  * D3 = 01 zeigt IRQ von PIC1 an
      beq    CHANGE_PIC1_RESULT_LOOP1
      move.b d0,(a5)+                 * D0 = Daten von PIC1
      subi.l #$08,d2                  * Chromo-Zaehler um 8 Bit dekrementieren
      cmp.l  #$00,d2
      bne    GET_CHROMO_PIC1

      moveq  #00,d3
```

```
      move.b #$00,pic1
    CHANGE_PIC1_RESULT_LOOP2
260   cmp.l  #$00,d3
      beq    CHANGE_PIC1_RESULT_LOOP2
      move.b d0,fitness1_hi          * Fitness des Chromosomes (High Byte)

      moveq  #00,d3
      move.b #$00,pic1
    CHANGE_PIC1_RESULT_LOOP3
      cmp.l  #$00,d3
      beq    CHANGE_PIC1_RESULT_LOOP3
      move.b d0,fitness1_lo          * Fitness des Chromosomes (Low Byte)
270
      moveq  #00,d3
      move.b #$00,pic1
    CHANGE_PIC1_GEN_LOOP0
      cmp.l  #00,d3
      beq    CHANGE_PIC1_GEN_LOOP0
      move.b d0,gen1_hi              * D0 = Anzahl Generationen (High Byte)

      moveq  #00,d3
      move.b #$00,pic1
280 CHANGE_PIC1_GEN_LOOP1
      cmp.l  #00,d3
      beq    CHANGE_PIC1_GEN_LOOP1
      move.b d0,gen1_mi              * D0 = Anzahl Generationen (Middle Byte)

      moveq  #00,d3
      move.b #$00,pic1
    CHANGE_PIC1_GEN_LOOP2
      cmp.l  #00,d3
      beq    CHANGE_PIC1_GEN_LOOP2
290   move.b d0,gen1_lo              * D0 = Anzahl Generationen (Low Byte)

      move.b topo0,d7                * PIC1-Topologie anhand von topo0/2/3
      btst.b #$01,d7                 * feststellen (vgl. "def.h")
      bne    NEW_DATA_FOR_PIC1
      move.b topo2,d7
      btst.b #$01,d7
      bne    NEW_DATA_FOR_PIC1
      move.b topo3,d7
      btst.b #$01,d7
300   bne    NEW_DATA_FOR_PIC1

      move.l a1,a5                   * Startadresse des empfangenen Chromosomes
      moveq  #00,d3                  * D3 speichert Generationen-Anzahl
      moveq  #00,d2                  * D2 speichert Fitness-Wert
      moveq  #00,d1
      move.b fitness1_hi,d2
      lsl.l  #08,d2
      move.l d2,d1
      moveq  #00,d2
310   move.b fitness1_lo,d2
      add.l  d1,d2
      moveq  #00,d0
      move.b gen1_hi,d0              * Anzahl Generationen
      lsl.l  #08,d0                  * ins High Byte "shiften"
      move.b gen1_mi,d0
      lsl.l  #08,d0                  * ins High Byte "shiften"
      move.b gen1_lo,d0
      move.l d0,d3
      move.b #$01,d1                 * PIC-Identitaet (PIC0 = 00, PIC1 = 01 ...)
320   bsr.l  LONG_NO_DATA_MESSG

      moveq  #00,d3                  * PLD-Latch zuruecksetzen
      move.b #$00,pic1
    CLEAR_PIC1_LATCH
      cmp.l  #00,d3
      beq    CLEAR_PIC1_LATCH
      move.b #$05,mode
      rts

330 NEW_DATA_FOR_PIC1
```

```
      move.l a1,a5                    * Startadresse des empfangenen Chromosomes
      moveq  #00,d3                   * D3 speichert Generationen-Anzahl
      moveq  #00,d2                   * D2 speichert Fitness-Wert
      moveq  #00,d1
      move.b fitness1_hi,d2
      lsl.l  #08,d2
      move.l d2,d1
      moveq  #00,d2
      move.b fitness1_lo,d2
340   add.l  d1,d2
      moveq  #00,d0
      move.b gen1_hi,d0               * Anzahl Generationen
      lsl.l  #08,d0                   * ins High Byte "shiften"
      move.b gen1_mi,d0
      lsl.l  #08,d0                   * ins High Byte "shiften"
      move.b gen1_lo,d0
      move.l d0,d3
      move.b #$01,d1                  * PIC-Identitaet (PIC0 = 00, PIC1 = 01 ...)
      bsr.l  LONG_NO_DATA_MESSG
350
      moveq  #00,d3
      move.b #$01,pic1
    FITNESS_PIC1_LOOP1
      cmp.l  #00,d3
      beq    FITNESS_PIC1_LOOP1

      moveq  #00,d0                   * Fitness-Werte speichern: PIC0 in D0,
      moveq  #00,d2                   * PIC1 in D1, ...
      moveq  #00,d3                   * A5 = Adresse des besten Chromosomes
360   moveq  #00,d5                   * D5 = Beste Fitness
      move.b fitness0_hi,d0
      lsl.w  #$08,d0
      add.b  fitness0_lo,d0
      move.b fitness2_hi,d2
      lsl.w  #$08,d2
      add.b  fitness2_lo,d2
      move.b fitness3_hi,d3
      lsl.w  #$08,d3
      add.b  fitness3_lo,d3
370
      move.b topo0,d7                 * TOPO0 testen:
      btst.b #$01,d7                  * Bit1=1 -> PIC1 erhaelt Daten von PIC0
      beq    HIGH_PART_FITNESS12
      move.w d0,d5
      move.l a0,a5
      move.l a0,a6                    * A6 = Startadresse zur Bildschirmausgabe
      move.b #$00,transmitter_pic     * Gibt datenuebergebenden PIC an
      move.b fitness0_hi,trans_fit_hi
      move.b fitness0_lo,trans_fit_lo
380
    HIGH_PART_FITNESS12
      move.b topo2,d7                 * TOPO2 testen:
      btst.b #$01,d7                  * Bit1=1 -> PIC1 erhaelt Daten von PIC2
      beq    HIGH_PART_FITNESS13
      cmp.l  d5,d2
      blt    HIGH_PART_FITNESS13
      move.w d2,d5
      move.l a2,a5
      move.l a2,a6                    * A6 = Startadresse zur Bildschirmausgabe
390   move.b #$02,transmitter_pic     * Gibt datenuebergebenden PIC an
      move.b fitness2_hi,trans_fit_hi
      move.b fitness2_lo,trans_fit_lo

    HIGH_PART_FITNESS13
      move.b topo3,d7                 * TOPO3 testen:
      btst.b #$01,d7                  * Bit1=1 -> PIC1 erhaelt Daten von PIC3
      beq    HIGH_PART_FITNESS11
      cmp.l  d5,d3
      blt    HIGH_PART_FITNESS11
400   move.w d3,d5
      move.l a3,a5
      move.l a3,a6                    * A6 = Startadresse zur Bildschirmausgabe
      move.b #$03,transmitter_pic     * Gibt datenuebergebenden PIC an
```

```
      move.b fitness3_hi,trans_fit_hi
      move.b fitness3_lo,trans_fit_lo

    HIGH_PART_FITNESS11
      moveq  #00,d3
      move.w d5,d3
410   lsr.w  #$08,d5
      move.b d5,pic1                * High Part der besten Fitness uebergeben
      move.w d3,d5
      moveq  #00,d3
    FITNESS_PIC1_LOOP2
      cmp.l  #00,d3
      beq    FITNESS_PIC1_LOOP2

      move.b d5,pic1                * Low Part der besten Fitness uebergeben

420   move.b chr_sizeH,d1           * Chromosomengroesse festlegen
      lsl.w  #$08,d1
      add.b  chr_sizeL,d1

    SEND_NEW_CHROMO1
      moveq  #00,d0
      moveq  #00,d3
    SEND_NEW_CHROMO_LOOP1
      cmp.l  #00,d3
      beq    SEND_NEW_CHROMO_LOOP1
430
      move.b (a5)+,d0
      move.b d0,pic1                * 8 Bit des besten Chromosomes weitergeben

      subi.w #$08,d1
      cmp.w  #$0000,d1
      bne    SEND_NEW_CHROMO1

      move.b #$01,receiver_pic      * PIC-Identitaet (PIC0 = 00, PIC1 = 01 ...)
      moveq  #00,d3                 * D3 speichert Fitness-Wert
440   moveq  #00,d1
      move.b trans_fit_hi,d3        * Fitness-Wert des ausgewaehlten Chromosomes
      lsl.l  #08,d3
      move.l d3,d1
      moveq  #00,d3
      move.b trans_fit_lo,d3
      add.l  d1,d3
      bsr.l  LONG_DATA_MESSG
      rts
    ***
450 ************************************************************************


    ************************************************************************
    ***
    *** Name....: CHANGE_RESULTS_PIC2
    *** Funktion: Daten von/an PIC2.
    *** Register: A5/A6 als Startadresse fuer Chromosome,
    ***           D0 - D7 als Datenregister.
    ***
460 CHANGE_RESULTS_PIC2
      moveq  #00,d2                 * Chromosomengroesse in D2 speichern
      move.b chr_sizeH,d2
      lsl.l  #$08,d2
      add.b  chr_sizeL,d2
      move.l a2,a5                  * a2 = Zeiger auf PIC2-Datenzellen

    GET_CHROMO_PIC2
      moveq  #00,d3
      move.b #$00,pic2              * PIC2 zum Datensenden auffordern
470 CHANGE_PIC2_RESULT_LOOP1
      cmp.l  #$00,d3                * D3 = 01 zeigt IRQ von PIC2 an
      beq    CHANGE_PIC2_RESULT_LOOP1
      move.b d0,(a5)+               * D0 = Daten von PIC2
      subi.l #$08,d2                * Chromo-Zaehler um 8 Bit dekrementieren
      cmp.l  #$00,d2
      bne    GET_CHROMO_PIC2
```

```
      moveq  #00,d3
      move.b #$00,pic2
480 CHANGE_PIC2_RESULT_LOOP2
      cmp.l  #$00,d3
      beq    CHANGE_PIC2_RESULT_LOOP2
      move.b d0,fitness2_hi          * Fitness des Chromosomes (High Byte)

      moveq  #00,d3
      move.b #$00,pic2
    CHANGE_PIC2_RESULT_LOOP3
      cmp.l  #$00,d3
      beq    CHANGE_PIC2_RESULT_LOOP3
490   move.b d0,fitness2_lo          * Fitness des Chromosomes (Low Byte)

      moveq  #00,d3
      move.b #$00,pic2
    CHANGE_PIC2_GEN_LOOP0
      cmp.l  #00,d3
      beq    CHANGE_PIC2_GEN_LOOP0
      move.b d0,gen2_hi              * D0 = Anzahl Generationen (High Byte)

      moveq  #00,d3
500   move.b #$00,pic2
    CHANGE_PIC2_GEN_LOOP1
      cmp.l  #00,d3
      beq    CHANGE_PIC2_GEN_LOOP1
      move.b d0,gen2_mi              * D0 = Anzahl Generationen (Middle Byte)

      moveq  #00,d3
      move.b #$00,pic2
    CHANGE_PIC2_GEN_LOOP2
      cmp.l  #00,d3
510   beq    CHANGE_PIC2_GEN_LOOP2
      move.b d0,gen2_lo              * D0 = Anzahl Generationen (Low Byte)

      move.b topo0,d7                * PIC2-Topologie anhand von topo0/1/3
      btst.b #$02,d7                 * feststellen (vgl. "def.h")
      bne    NEW_DATA_FOR_PIC2
      move.b topo1,d7
      btst.b #$02,d7
      bne    NEW_DATA_FOR_PIC2
      move.b topo3,d7
520   btst.b #$02,d7
      bne    NEW_DATA_FOR_PIC2

      move.l a2,a5                   * Startadresse des empfangenen Chromosomes
      moveq  #00,d3                  * D3 speichert Generationen-Anzahl
      moveq  #00,d2                  * D2 speichert Fitness-Wert
      moveq  #00,d1
      move.b fitness2_hi,d2
      lsl.l  #08,d2
      move.l d2,d1
530   moveq  #00,d2
      move.b fitness2_lo,d2
      add.l  d1,d2
      moveq  #00,d0
      move.b gen2_hi,d0              * Anzahl Generationen
      lsl.l  #08,d0                  * ins High Byte "shiften"
      move.b gen2_mi,d0
      lsl.l  #08,d0                  * ins High Byte "shiften"
      move.b gen2_lo,d0
      move.l d0,d3
540   move.b #$02,d1                 * PIC-Identitaet (PIC0 = 00, PIC1 = 01 ...)
      bsr.l  LONG_NO_DATA_MESSG

      moveq  #00,d3                  * PLD-Latch zuruecksetzen
      move.b #$00,pic2
    CLEAR_PIC2_LATCH
      cmp.l  #00,d3
      beq    CLEAR_PIC2_LATCH
      move.b #$05,mode
      rts
```

```
550
    NEW_DATA_FOR_PIC2
      move.l  a2,a5                   * Startadresse des empfangenen Chromosomes
      moveq   #00,d3                  * D3 speichert Generationen-Anzahl
      moveq   #00,d2                  * D2 speichert Fitness-Wert
      moveq   #00,d1
      move.b  fitness2_hi,d2
      lsl.l   #08,d2
      move.l  d2,d1
      moveq   #00,d2
560   move.b  fitness2_lo,d2
      add.l   d1,d2
      moveq   #00,d0
      move.b  gen2_hi,d0              * Anzahl Generationen
      lsl.l   #08,d0                  * ins High Byte "shiften"
      move.b  gen2_mi,d0
      lsl.l   #08,d0                  * ins High Byte "shiften"
      move.b  gen2_lo,d0
      move.l  d0,d3
      move.b  #$02,d1                 * PIC-Identitaet (PIC0 = 00, PIC1 = 01 ...)
570   bsr.l   LONG_NO_DATA_MESSG

      moveq   #00,d3
      move.b  #$01,pic2
    FITNESS_PIC2_LOOP1
      cmp.l   #00,d3
      beq     FITNESS_PIC2_LOOP1

      moveq   #00,d0                  * Fitness-Werte speichern:
      moveq   #00,d1                  * PIC0 in D0, PIC1 in D1, ...
580   moveq   #00,d3                  * A5 = Adresse des besten Chromosomes
      moveq   #00,d5                  * D5 = Beste Fitness
      move.b  fitness0_hi,d0
      lsl.w   #$08,d0
      add.b   fitness0_lo,d0
      move.b  fitness1_hi,d1
      lsl.w   #$08,d1
      add.b   fitness1_lo,d1
      move.b  fitness3_hi,d3
      lsl.w   #$08,d3
590   add.b   fitness3_lo,d3

      move.b  topo0,d7                * TOPO0 testen:
      btst.b  #$02,d7                 * Bit2=1 -> PIC2 erhaelt Daten von PIC0
      beq     HIGH_PART_FITNESS21
      move.w  d0,d5
      move.l  a0,a5
      move.l  a0,a6                   * A6 = Startadresse zur Bildschirmausgabe
      move.b  #$00,transmitter_pic    * Gibt datenuebergebenden PIC an
      move.b  fitness0_hi,trans_fit_hi
600   move.b  fitness0_lo,trans_fit_lo

    HIGH_PART_FITNESS21
      move.b  topo1,d7                * TOPO1 testen:
      btst.b  #$02,d7                 * Bit2=1 -> PIC2 erhaelt Daten von PIC1
      beq     HIGH_PART_FITNESS23
      cmp.l   d5,d1
      blt     HIGH_PART_FITNESS23
      move.w  d1,d5
      move.l  a1,a5
610   move.l  a1,a6                   * A6 = Startadresse zur Bildschirmausgabe
      move.b  #$01,transmitter_pic    * Gibt datenuebergebenden PIC an
      move.b  fitness1_hi,trans_fit_hi
      move.b  fitness1_lo,trans_fit_lo

    HIGH_PART_FITNESS23
      move.b  topo3,d7                * TOPO3 testen:
      btst.b  #$02,d7                 * Bit2=1 -> PIC2 erhaelt Daten von PIC3
      beq     HIGH_PART_FITNESS22
      cmp.l   d5,d3
620   blt     HIGH_PART_FITNESS22
      move.w  d3,d5
      move.l  a3,a5
```

```
      move.l a3,a6                    * A6 = Startadresse zur Bildschirmausgabe
      move.b #$03,transmitter_pic     * Gibt datenuebergebenden PIC an
      move.b fitness3_hi,trans_fit_hi
      move.b fitness3_lo,trans_fit_lo

    HIGH_PART_FITNESS22
      moveq  #00,d3
630   move.w d5,d3
      lsr.w  #$08,d5
      move.b d5,pic2                  * High Part der besten Fitness uebergeben
      move.w d3,d5
      moveq  #00,d3
    FITNESS_PIC2_LOOP2
      cmp.l  #00,d3
      beq    FITNESS_PIC2_LOOP2

      move.b d5,pic2                  * Low Part der besten Fitness uebergeben
640
      move.b chr_sizeH,d1             * Chromosomengroesse festlegen
      lsl.w  #$08,d1
      add.b  chr_sizeL,d1

    SEND_NEW_CHROMO2
      moveq  #00,d0
      moveq  #00,d3
    SEND_NEW_CHROMO_LOOP2
      cmp.l  #00,d3
650   beq    SEND_NEW_CHROMO_LOOP2

      move.b (a5)+,d0
      move.b d0,pic2                  * 8 Bit des besten Chromosomes weitergeben

      subi.w #$08,d1
      cmp.w  #$0000,d1
      bne    SEND_NEW_CHROMO2

      move.b #$02,receiver_pic        * PIC-Identitaet (PIC0 = 00, PIC1 = 01 ...)
660   moveq  #00,d3                   * D3 speichert Fitness-Wert
      moveq  #00,d1
      move.b trans_fit_hi,d3          * Fitness-Wert des ausgewaehlten Chromosomes
      lsl.l  #08,d3
      move.l d3,d1
      moveq  #00,d3
      move.b trans_fit_lo,d3
      add.l  d1,d3
      bsr.l  LONG_DATA_MESSG
      rts
670 ***
    ********************************************************************


    ********************************************************************
    ***
    *** Name....: CHANGE_RESULTS_PIC3
    *** Funktion: Daten von/an PIC3.
    *** Register: A5/A6 als Startadresse fuer Chromosome,
    ***           D0 - D7 als Datenregister.
680 ***
    CHANGE_RESULTS_PIC3
      moveq  #00,d2                   * Chromosomengroesse in D2 speichern
      move.b chr_sizeH,d2
      lsl.l  #$08,d2
      add.b  chr_sizeL,d2
      move.l a3,a5                    * a3 = Zeiger auf PIC3-Datenzellen

    GET_CHROMO_PIC3
      moveq  #00,d3
690   move.b #$00,pic3                * PIC3 zum Datensenden auffordern
    CHANGE_PIC3_RESULT_LOOP1
      cmp.l  #$00,d3                  * D3 = 01 zeigt IRQ von PIC3 an
      beq    CHANGE_PIC3_RESULT_LOOP1
      move.b d0,(a5)+                 * D0 = Daten von PIC3
      subi.l #$08,d2                  * Chromo-Zaehler um 8 Bit dekrementieren
```

```
      cmp.l   #$00,d2
      bne     GET_CHROMO_PIC3

      moveq   #00,d3
700   move.b  #$00,pic3
    CHANGE_PIC3_RESULT_LOOP2
      cmp.l   #$00,d3
      beq     CHANGE_PIC3_RESULT_LOOP2
      move.b  d0,fitness3_hi          * Fitness des Chromosomes (High Byte)

      moveq   #00,d3
      move.b  #$00,pic3
    CHANGE_PIC3_RESULT_LOOP3
      cmp.l   #$00,d3
710   beq     CHANGE_PIC3_RESULT_LOOP3
      move.b  d0,fitness3_lo          * Fitness des Chromosomes (Low Byte)

      moveq   #00,d3
      move.b  #$00,pic3
    CHANGE_PIC3_GEN_LOOP0
      cmp.l   #00,d3
      beq     CHANGE_PIC3_GEN_LOOP0
      move.b  d0,gen3_hi              * D0 = Anzahl Generationen (High Byte)

720   moveq   #00,d3
      move.b  #$00,pic3
    CHANGE_PIC3_GEN_LOOP1
      cmp.l   #00,d3
      beq     CHANGE_PIC3_GEN_LOOP1
      move.b  d0,gen3_mi              * D0 = Anzahl Generationen (Middle Byte)

      moveq   #00,d3
      move.b  #$00,pic3
    CHANGE_PIC3_GEN_LOOP2
730   cmp.l   #00,d3
      beq     CHANGE_PIC3_GEN_LOOP2
      move.b  d0,gen3_lo              * D0 = Anzahl Generationen (Low Byte)

      move.b  topo0,d7                * PIC3-Topologie anhand von topo0/1/2
      btst.b  #$03,d7                 * feststellen (vgl. "def.h")
      bne     NEW_DATA_FOR_PIC3
      move.b  topo1,d7
      btst.b  #$03,d7
      bne     NEW_DATA_FOR_PIC3
740   move.b  topo2,d7
      btst.b  #$03,d7
      bne     NEW_DATA_FOR_PIC3

      move.l  a3,a5                   * Startadresse des empfangenen Chromosomes
      moveq   #00,d3                  * D3 speichert Generationen-Anzahl
      moveq   #00,d2                  * D2 speichert Fitness-Wert
      moveq   #00,d1
      move.b  fitness3_hi,d2
      lsl.l   #08,d2
750   move.l  d2,d1
      moveq   #00,d2
      move.b  fitness3_lo,d2
      add.l   d1,d2
      moveq   #00,d0
      move.b  gen3_hi,d0              * Anzahl Generationen
      lsl.l   #08,d0                  * ins High Byte "shiften"
      move.b  gen3_mi,d0
      lsl.l   #08,d0                  * ins High Byte "shiften"
      move.b  gen3_lo,d0
760   move.l  d0,d3
      move.b  #$03,d1                 * PIC-Identitaet (PIC0 = 00, PIC1 = 01 ...)
      bsr.l   LONG_NO_DATA_MESSG

      moveq   #00,d3                  * PLD-Latch zuruecksetzen
      move.b  #$00,pic3
    CLEAR_PIC3_LATCH
      cmp.l   #00,d3
      beq     CLEAR_PIC3_LATCH
```

```
      move.b #$05,mode
770   rts

    NEW_DATA_FOR_PIC3
      move.l a3,a5                    * Startadresse des empfangenen Chromosomes
      moveq  #00,d3                   * D3 speichert Generationen-Anzahl
      moveq  #00,d2                   * D2 speichert Fitness-Wert
      moveq  #00,d1
      move.b fitness3_hi,d2
      lsl.l  #08,d2
      move.l d2,d1
780   moveq  #00,d2
      move.b fitness3_lo,d2
      add.l  d1,d2
      moveq  #00,d0
      move.b gen3_hi,d0               * Anzahl Generationen
      lsl.l  #08,d0                   * ins High Byte "shiften"
      move.b gen3_mi,d0
      lsl.l  #08,d0                   * ins High Byte "shiften"
      move.b gen3_lo,d0
      move.l d0,d3
790   move.b #$03,d1                  * PIC-Identitaet (PIC0 = 00, PIC1 = 01 ...)
      bsr.l  LONG_NO_DATA_MESSG

      moveq  #00,d3
      move.b #$01,pic3
    FITNESS_PIC3_LOOP1
      cmp.l  #00,d3
      beq    FITNESS_PIC3_LOOP1

      moveq  #00,d0                   * Fitness-Werte speichern: PIC0 in D0,
800   moveq  #00,d1                   * PIC1 in D1, ...
      moveq  #00,d2                   * A5 = Adresse des besten Chromosomes
      moveq  #00,d5                   * D5 = Beste Fitness
      move.b fitness0_hi,d0
      lsl.w  #$08,d0
      add.b  fitness0_lo,d0
      move.b fitness1_hi,d1
      lsl.w  #$08,d1
      add.b  fitness1_lo,d1
      move.b fitness2_hi,d2
810   lsl.w  #$08,d2
      add.b  fitness2_lo,d2

      move.b topo0,d7                 * TOPO0 testen:
      btst.b #$03,d7                  * Bit3=1 -> PIC3 erhaelt Daten von PIC0
      beq    HIGH_PART_FITNESS31
      move.w d0,d5
      move.l a0,a5
      move.l a0,a6                    * A6 = Startadresse zur Bildschirmausgabe
      move.b #$00,transmitter_pic     * Gibt datenuebergebenden PIC an
820   move.b fitness0_hi,trans_fit_hi
      move.b fitness0_lo,trans_fit_lo

    HIGH_PART_FITNESS31
      move.b topo1,d7                 * TOPO1 testen:
      btst.b #$03,d7                  * Bit3=1 -> PIC3 erhaelt Daten von PIC1
      beq    HIGH_PART_FITNESS32
      cmp.l  d5,d1
      blt    HIGH_PART_FITNESS32
      move.w d1,d5
830   move.l a1,a5
      move.l a1,a6                    * A6 = Startadresse zur Bildschirmausgabe
      move.b #$01,transmitter_pic     * Gibt datenuebergebenden PIC an
      move.b fitness1_hi,trans_fit_hi
      move.b fitness1_lo,trans_fit_lo

    HIGH_PART_FITNESS32
      move.b topo2,d7                 * TOPO2 testen:
      btst.b #$03,d7                  * Bit3=1 -> PIC3 erhaelt Daten von PIC2
      beq    HIGH_PART_FITNESS33
840   cmp.l  d5,d2
      blt    HIGH_PART_FITNESS33
```

```
      move.w d2,d5
      move.l a2,a5
      move.l a2,a6                    * A6 = Startadresse zur Bildschirmausgabe
      move.b #$02,transmitter_pic     * Gibt datenuebergebenden PIC an
      move.b fitness2_hi,trans_fit_hi
      move.b fitness2_lo,trans_fit_lo

    HIGH_PART_FITNESS33
850   moveq  #00,d3
      move.w d5,d3
      lsr.w  #$08,d5
      move.b d5,pic3                  * High Part der besten Fitness uebergeben
      move.w d3,d5
      moveq  #00,d3
    FITNESS_PIC3_LOOP2
      cmp.l  #00,d3
      beq    FITNESS_PIC3_LOOP2

860   move.b d5,pic3                  * Low Part der besten Fitness uebergeben

      move.b chr_sizeH,d1             * Chromosomengroesse festlegen
      lsl.w  #$08,d1
      add.b  chr_sizeL,d1

    SEND_NEW_CHROMO3
      moveq  #00,d0
      moveq  #00,d3
    SEND_NEW_CHROMO_LOOP3
870   cmp.l  #00,d3
      beq    SEND_NEW_CHROMO_LOOP3

      move.b (a5)+,d0
      move.b d0,pic3                  * 8 Bit des besten Chromosomes weitergeben

      subi.w #$08,d1
      cmp.w  #$0000,d1
      bne    SEND_NEW_CHROMO3

880   move.b #$03,receiver_pic        * PIC-Identitaet (PIC0 = 00, PIC1 = 01 ...)
      moveq  #00,d3                   * D3 speichert Fitness-Wert
      moveq  #00,d1
      move.b trans_fit_hi,d3          * Fitness-Wert des ausgewaehlten Chromosomes
      lsl.l  #08,d3
      move.l d3,d1
      moveq  #00,d3
      move.b trans_fit_lo,d3
      add.l  d1,d3
      bsr.l  LONG_DATA_MESSG
890   rts
    ***
    ************************************************************************
```

B.19 Entgegennahme der besten Resultate: `getres.a`

```
    ************************************************************************
    ***              Die besten Resultate der PICs empfangen             ***
    *** Autor:       Tobias Schubert                                     ***
    *** EMail:       schubert@informatik.uni-freiburg.de                 ***
    *** Datum:       11.09.1999                                          ***
    *** Datei-Name:  getres.a                                            ***
    ************************************************************************


 10 ************************************************************************
    ***
    *** Name....:  GET_RES_PIC0
    *** Funktion: Empfang der besten Loesung von PIC0.
```

```
   *** Register: D0, D3, A5 um die empfangenen Daten zu "verteilen".
   ***
   GET_RES_PIC0
     moveq  #00,d3
     move.b #$00,pic0
   RES_PIC0_LOOP0
20   cmp.l  #00,d3
     beq    RES_PIC0_LOOP0
     move.b d0,gen0_hi              * D0 = Anzahl Generationen (High Byte)

     moveq  #00,d3
     move.b #$00,pic0
   RES_PIC0_LOOP1
     cmp.l  #00,d3
     beq    RES_PIC0_LOOP1
     move.b d0,gen0_mi              * D0 = Anzahl Generationen (Middle Byte)
30
     moveq  #00,d3
     move.b #$00,pic0
   RES_PIC0_LOOP2
     cmp.l  #00,d3
     beq    RES_PIC0_LOOP2
     move.b d0,gen0_lo              * D0 = Anzahl Generationen (Low Byte)

     moveq  #00,d3
     move.b #$00,pic0
40 RES_PIC0_LOOP3
     cmp.l  #00,d3
     beq    RES_PIC0_LOOP3
     move.b d0,child_size0          * D0 = Anzahl Nachkommen pro Generation

     moveq  #00,d3
     move.b #$00,pic0
   RES_PIC0_LOOP4
     cmp.l  #00,d3
     beq    RES_PIC0_LOOP4
50   move.b d0,pop_size0            * D0 = Populationsgroesse

     moveq  #00,d3
     move.b #$00,pic0
   RES_PIC0_LOOP5
     cmp.l  #00,d3
     beq    RES_PIC0_LOOP5
     move.b d0,chr_sizeH            * Chromosomengroesse (High Byte)

     moveq  #00,d3
60   move.b #$00,pic0
   RES_PIC0_LOOP6
     cmp.l  #00,d3
     beq    RES_PIC0_LOOP6
     move.b d0,chr_sizeL            * Chromosomengroesse (Low Byte)

     moveq  #00,d3
     move.b #$00,pic0
   RES_PIC0_LOOP7
     cmp.l  #00,d3
70   beq    RES_PIC0_LOOP7
     move.b d0,fitness0_hi          * Fitness (High Byte)

     moveq  #00,d3
     move.b #$00,pic0
   RES_PIC0_LOOP8
     cmp.l  #00,d3
     beq    RES_PIC0_LOOP8
     move.b d0,fitness0_lo          * Fitness (Low Byte)

80   moveq  #00,d3
     move.b #$00,pic0
   RES_PIC0_LOOPA
     cmp.l  #00,d3
     beq    RES_PIC0_LOOPA
     move.b d0,remainder0           * Nachkommastellen
```

```
      move.l a0,a5                    * Zeiger auf Datenzellen
      moveq  #00,d1
      move.b chr_sizeH,d1
 90   lsl.w  #$08,d1
      add.b  chr_sizeL,d1

    BEST_CHROMO_PIC0
      moveq  #00,d3
      move.b #$00,pic0
    RES_PIC0_LOOP9
      cmp.l  #00,d3
      beq    RES_PIC0_LOOP9
      move.b d0,(a5)+                 * 1 Byte des besten Resultats speichern
100   subi.w #$08,d1
      cmp.w  #$0000,d1
      bne    BEST_CHROMO_PIC0

    *** PLD-Latch zuruecksetzen
      moveq  #00,d3
      move.b #$00,pic0
    RES_PIC0_LOOP10
      cmp.l  #00,d3
      beq    RES_PIC0_LOOP10
110
      bset.b #$00,results
      rts
    ***
    ********************************************************************************


    ********************************************************************************
    ***
    *** Name....: GET_RES_PIC1
120 *** Funktion: Empfang der besten Loesung von PIC1.
    *** Register: D0, D3, A5 um die empfangenen Daten zu "verteilen".
    ***
    GET_RES_PIC1
      moveq  #00,d3
      move.b #$00,pic1
    RES_PIC1_LOOP0
      cmp.l  #00,d3
      beq    RES_PIC1_LOOP0
      move.b d0,gen1_hi               * D0 = Anzahl Generationen (High Byte)
130
      moveq  #00,d3
      move.b #$00,pic1
    RES_PIC1_LOOP1
      cmp.l  #00,d3
      beq    RES_PIC1_LOOP1
      move.b d0,gen1_mi               * D0 = Anzahl Generationen (Middle Byte)

      moveq  #00,d3
      move.b #$00,pic1
140 RES_PIC1_LOOP2
      cmp.l  #00,d3
      beq    RES_PIC1_LOOP2
      move.b d0,gen1_lo               * D0 = Anzahl Generationen (Low Byte)

      moveq  #00,d3
      move.b #$00,pic1
    RES_PIC1_LOOP3
      cmp.l  #00,d3
      beq    RES_PIC1_LOOP3
150   move.b d0,child_size1           * D0 = Anzahl Nachkommen pro Generation

      moveq  #00,d3
      move.b #$00,pic1
    RES_PIC1_LOOP4
      cmp.l  #00,d3
      beq    RES_PIC1_LOOP4
      move.b d0,pop_size1             * D0 = Populationsgroesse

      moveq  #00,d3
```

```
160   move.b #$00,pic1
    RES_PIC1_LOOP5
      cmp.l  #00,d3
      beq    RES_PIC1_LOOP5
      move.b d0,chr_sizeH           * Chromosomengroesse (High Byte)

      moveq  #00,d3
      move.b #$00,pic1
    RES_PIC1_LOOP6
      cmp.l  #00,d3
170   beq    RES_PIC1_LOOP6
      move.b d0,chr_sizeL           * Chromosomengroesse (Low Byte)

      moveq  #00,d3
      move.b #$00,pic1
    RES_PIC1_LOOP7
      cmp.l  #00,d3
      beq    RES_PIC1_LOOP7
      move.b d0,fitness1_hi         * Fitness (High Byte)

180   moveq  #00,d3
      move.b #$00,pic1
    RES_PIC1_LOOP8
      cmp.l  #00,d3
      beq    RES_PIC1_LOOP8
      move.b d0,fitness1_lo         * Fitness (Low Byte)

      moveq  #00,d3
      move.b #$00,pic1
    RES_PIC1_LOOPA
190   cmp.l  #00,d3
      beq    RES_PIC1_LOOPA
      move.b d0,remainder1          * Nachkommastellen

      move.l a1,a5                  * Zeiger auf Datenzellen
      moveq  #00,d1
      move.b chr_sizeH,d1
      lsl.w  #$08,d1
      add.b  chr_sizeL,d1

200 BEST_CHROMO_PIC1
      moveq  #00,d3
      move.b #$00,pic1
    RES_PIC1_LOOP9
      cmp.l  #00,d3
      beq    RES_PIC1_LOOP9
      move.b d0,(a5)+               * 1 Byte des besten Resultats speichern
      subi.w #$08,d1
      cmp.w  #$0000,d1
      bne    BEST_CHROMO_PIC1
210
    *** PLD-Latch zuruecksetzen
      moveq  #00,d3
      move.b #$00,pic1
    RES_PIC1_LOOP10
      cmp.l  #00,d3
      beq    RES_PIC1_LOOP10

      bset.b #$01,results
      rts
220 ***
    ********************************************************************************


    ********************************************************************************
    ***
    *** Name....:  GET_RES_PIC2
    *** Funktion: Empfang der besten Loesung von PIC2.
    *** Register: D0, D3, A5 um die empfangenen Daten zu "verteilen".
    ***
230 GET_RES_PIC2
      moveq  #00,d3
      move.b #$00,pic2
```

```
    RES_PIC2_LOOP0
      cmp.l   #00,d3
      beq     RES_PIC2_LOOP0
      move.b  d0,gen2_hi              * D0 = Anzahl Generationen (High Byte)

      moveq   #00,d3
      move.b  #$00,pic2
240 RES_PIC2_LOOP1
      cmp.l   #00,d3
      beq     RES_PIC2_LOOP1
      move.b  d0,gen2_mi              * D0 = Anzahl Generationen (Middle Byte)

      moveq   #00,d3
      move.b  #$00,pic2
    RES_PIC2_LOOP2
      cmp.l   #00,d3
      beq     RES_PIC2_LOOP2
250   move.b  d0,gen2_lo              * D0 = Anzahl Generationen (Low Byte)

      moveq   #00,d3
      move.b  #$00,pic2
    RES_PIC2_LOOP3
      cmp.l   #00,d3
      beq     RES_PIC2_LOOP3
      move.b  d0,child_size2          * D0 = Anzahl Nachkommen pro Generation

      moveq   #00,d3
260   move.b  #$00,pic2
    RES_PIC2_LOOP4
      cmp.l   #00,d3
      beq     RES_PIC2_LOOP4
      move.b  d0,pop_size2            * D0 = Populationsgroesse

      moveq   #00,d3
      move.b  #$00,pic2
    RES_PIC2_LOOP5
      cmp.l   #00,d3
270   beq     RES_PIC2_LOOP5
      move.b  d0,chr_sizeH            * Chromosomengroesse (High Byte)

      moveq   #00,d3
      move.b  #$00,pic2
    RES_PIC2_LOOP6
      cmp.l   #00,d3
      beq     RES_PIC2_LOOP6
      move.b  d0,chr_sizeL            * Chromosomengroesse (Low Byte)

280   moveq   #00,d3
      move.b  #$00,pic2
    RES_PIC2_LOOP7
      cmp.l   #00,d3
      beq     RES_PIC2_LOOP7
      move.b  d0,fitness2_hi          * Fitness (High Byte)

      moveq   #00,d3
      move.b  #$00,pic2
    RES_PIC2_LOOP8
290   cmp.l   #00,d3
      beq     RES_PIC2_LOOP8
      move.b  d0,fitness2_lo          * Fitness (Low Byte)

      moveq   #00,d3
      move.b  #$00,pic2
    RES_PIC2_LOOPA
      cmp.l   #00,d3
      beq     RES_PIC2_LOOPA
      move.b  d0,remainder2           * Nachkommastellen
300
      move.l  a2,a5                   * Zeiger auf Datenzellen
      moveq   #00,d1
      move.b  chr_sizeH,d1
      lsl.w   #$08,d1
      add.b   chr_sizeL,d1
```

```
    BEST_CHROMO_PIC2
      moveq  #00,d3
      move.b #$00,pic2
310 RES_PIC2_LOOP9
      cmp.l  #00,d3
      beq    RES_PIC2_LOOP9
      move.b d0,(a5)+              * 1 Byte des besten Resultats speichern
      subi.w #$08,d1
      cmp.w  #$0000,d1
      bne    BEST_CHROMO_PIC2

    *** PLD-Latch zuruecksetzen
      moveq  #00,d3
320   move.b #$00,pic2
    RES_PIC2_LOOP10
      cmp.l  #00,d3
      beq    RES_PIC2_LOOP10

      bset.b #$02,results
      rts
    ***
    ******************************************************************************

330
    ******************************************************************************
    ***
    *** Name....: GET_RES_PIC3
    *** Funktion: Empfang der besten Loesung von PIC3.
    *** Register: D0, D3, A5 um die empfangenen Daten zu "verteilen".
    ***
    GET_RES_PIC3
      moveq  #00,d3
      move.b #$00,pic3
340 RES_PIC3_LOOP0
      cmp.l  #00,d3
      beq    RES_PIC3_LOOP0
      move.b d0,gen3_hi            * D0 = Anzahl Generationen (High Byte)

      moveq  #00,d3
      move.b #$00,pic3
    RES_PIC3_LOOP1
      cmp.l  #00,d3
      beq    RES_PIC3_LOOP1
350   move.b d0,gen3_mi            * D0 = Anzahl Generationen (Middle Byte)

      moveq  #00,d3
      move.b #$00,pic3
    RES_PIC3_LOOP2
      cmp.l  #00,d3
      beq    RES_PIC3_LOOP2
      move.b d0,gen3_lo            * D0 = Anzahl Generationen (Low Byte)

      moveq  #00,d3
360   move.b #$00,pic3
    RES_PIC3_LOOP3
      cmp.l  #00,d3
      beq    RES_PIC3_LOOP3
      move.b d0,child_size3        * D0 = Anzahl Nachkommen pro Generation

      moveq  #00,d3
      move.b #$00,pic3
    RES_PIC3_LOOP4
      cmp.l  #00,d3
370   beq    RES_PIC3_LOOP4
      move.b d0,pop_size3          * D0 = Populationsgroesse

      moveq  #00,d3
      move.b #$00,pic3
    RES_PIC3_LOOP5
      cmp.l  #00,d3
      beq    RES_PIC3_LOOP5
      move.b d0,chr_sizeH          * Chromosomengroesse (High Byte)
```

```
380   moveq  #00,d3
      move.b #$00,pic3
    RES_PIC3_LOOP6
      cmp.l  #00,d3
      beq    RES_PIC3_LOOP6
      move.b d0,chr_sizeL          * Chromosomengroesse (Low Byte)

      moveq  #00,d3
      move.b #$00,pic3
    RES_PIC3_LOOP7
390   cmp.l  #00,d3
      beq    RES_PIC3_LOOP7
      move.b d0,fitness3_hi        * Fitness (High Byte)

      moveq  #00,d3
      move.b #$00,pic3
    RES_PIC3_LOOP8
      cmp.l  #00,d3
      beq    RES_PIC3_LOOP8
      move.b d0,fitness3_lo        * Fitness (Low Byte)
400
      moveq  #00,d3
      move.b #$00,pic3
    RES_PIC3_LOOPA
      cmp.l  #00,d3
      beq    RES_PIC3_LOOPA
      move.b d0,remainder3         * Nachkommastellen

      move.l a3,a5                 * Zeiger auf Datenzellen
      moveq  #00,d1
410   move.b chr_sizeH,d1
      lsl.w  #$08,d1
      add.b  chr_sizeL,d1

    BEST_CHROMO_PIC3
      moveq  #00,d3
      move.b #$00,pic3
    RES_PIC3_LOOP9
      cmp.l  #00,d3
      beq    RES_PIC3_LOOP9
420   move.b d0,(a5)+              * 1 Byte des besten Resultats speichern
      subi.w #$08,d1
      cmp.w  #$0000,d1
      bne    BEST_CHROMO_PIC3

    *** PLD-Latch zuruecksetzen
      moveq  #00,d3
      move.b #$00,pic3
    RES_PIC3_LOOP10
      cmp.l  #00,d3
430   beq    RES_PIC3_LOOP10

      bset.b #$03,results
      rts
    ***
    ********************************************************************************
```

B.20 Ausgabe der besten Resultate: showres.a

```
*************************************************************************
***              Anzeigen der besten Resultate der PICs am PC-Terminal ***
*** Autor:        Tobias Schubert                                       ***
*** EMail:        schubert@informatik.uni-freiburg.de                   ***
*** Datum:        24.10.1999                                            ***
*** Datei-Name:   showres.a                                             ***
*************************************************************************

*************************************************************************
***
*** Name....:  SHOW_PIC0_RES
*** Funktion:  Anzeigen des besten PIC0-Resultates
*** Register:  D0 - D7, A5
***
SHOW_PIC0_RES
    bsr.l   SHOW_RES_INTRO          * "Ergebnis von PIC"
    move.b  #$30,d4
    bsr.l   SEND_CHB_BYTE
    move.l  #$2020203A,d4           * ":   "
    bsr.l   SEND_CHB_1LONG
    moveq   #42,d7
    bsr.l   SEND_SPACE
    move.b  #$BA,d4                 * = "|"
    bsr.l   SEND_CHB_BYTE
    move.b  #$0D,d4                 * = "CR"
    bsr.l   SEND_CHB_BYTE
    moveq   #10,d7
    bsr.l   SEND_SPACE
    moveq   #65,d7
    bsr.l   SEND_MI_LINE
    moveq   #10,d7
    bsr.l   SEND_SPACE
    move.b  #$BA,d4                 * = "|"
    bsr.l   SEND_CHB_BYTE
    moveq   #65,d7
    bsr.l   SEND_SPACE
    move.b  #$BA,d4                 * = "|"
    bsr.l   SEND_CHB_BYTE
    move.b  #$0D,d4                 * = "CR"
    bsr.l   SEND_CHB_BYTE

    bsr.l   SHOW_NR_GEN             * "Anzahl an Generationen"
    moveq   #00,d0
    moveq   #00,d1
    move.b  gen0_hi,d0              * Anzahl Generationen
    lsl.l   #$08,d0                 * ins High Byte "shiften"
    move.b  gen0_mi,d0
    lsl.l   #$08,d0                 * ins High Byte "shiften"
    move.b  gen0_lo,d0
    moveq   #00,d1
    moveq   #00,d2
    bsr.l   HEX2DEZ_LONG            * Datenkonvertierung, bessere Lesbarkeit
    bsr.l   DECODE_DEZ_LONG
    moveq   #08,d7
    bsr.l   SEND_SPACE
    move.b  #$BA,d4                 * = "|"
    bsr.l   SEND_CHB_BYTE
    move.b  #$0D,d4                 * = "CR"
    bsr.l   SEND_CHB_BYTE

    bsr.l   SHOW_NR_CHILDS          * "Anzahl Nachkommen pro Generation"
    moveq   #00,d0
    move.b  child_size0,d0
    moveq   #00,d1
    moveq   #00,d2
    bsr.l   HEX2DEZ                 * Datenkonvertierung, bessere Lesbarkeit
    bsr.l   DECODE_DEZ
    moveq   #08,d7
    bsr.l   SEND_SPACE
```

```
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE

        moveq   #00,d0
        move.b  pop_size0,d0
        bsr.l   SHOW_POPSIZE            * "Groesse der Population"
        moveq   #00,d1
 80     moveq   #00,d2
        bsr.l   HEX2DEZ
        bsr.l   DECODE_DEZ
        moveq   #08,d7
        bsr.l   SEND_SPACE
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE

 90     moveq   #00,d0
        move.b  chr_sizeH,d0
        lsl.l   #08,d0
        move.l  d0,d1
        moveq   #00,d0
        move.b  chr_sizeL,d0
        add.l   d1,d0
        bsr.l   SHOW_CHROMOSIZE         * "Groesse der Chromosome"
        moveq   #00,d1
        moveq   #00,d2
100     bsr.l   HEX2DEZ
        bsr.l   DECODE_DEZ
        moveq   #08,d7
        bsr.l   SEND_SPACE
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE

        moveq   #00,d0
110     moveq   #00,d1
        move.b  fitness0_hi,d0
        lsl.l   #08,d0
        move.l  d0,d1
        moveq   #00,d0
        move.b  fitness0_lo,d0
        add.l   d1,d0
        move.w  d0,d2
        moveq   #00,d0
        move.w  #$FFFF,d0
120     sub.w   d2,d0
        bsr.l   SHOW_FITNESS            * "Fitness des besten Chromosoms"
        moveq   #00,d1
        moveq   #00,d2
        bsr.l   HEX2DEZ
        bsr.l   DECODE_DEZ

        move.b  #$2C,d4                 * = ","
        bsr.l   SEND_CHB_BYTE
        moveq   #00,d0
130     move.b  remainder0,d0
        bsr.l   REMAINDER_OUTPUT

        moveq   #03,d7
        bsr.l   SEND_SPACE
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE

140     move.l  a0,a5
        bsr.l   SHOW_BEST_RES           * "Beste Loesung"
        moveq   #00,d6
        move.b  chr_sizeH,d6
```

```
      lsl.w   #$08,d6
      add.b   chr_sizeL,d6
      move.b #$04,chk_sum            * chk_sum dient als Schleifenzaehler
      bra.l   SHOW_CHROMO_PIC0_SMALL_LOOP

    SHOW_CHROMO_PIC0
150   moveq   #32,d7
      bsr.l   SEND_SPACE
      move.b #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      move.b #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE
      move.b #$04,chk_sum            * chk_sum dient als Schleifenzaehler
      moveq   #10,d7
      bsr.l   SEND_SPACE
      move.b #$BA,d4                 * = "|"
160   bsr.l   SEND_CHB_BYTE
      moveq   #18,d7
      bsr.l   SEND_SPACE

    SHOW_CHROMO_PIC0_SMALL_LOOP
      moveq   #00,d0
      move.b (a5)+,d0
      moveq   #00,d1
      moveq   #00,d2
      bsr.l   HEX2DEZ                * Datenkonvertierung, bessere Lesbarkeit
170   move.l d1,d4
      lsr.l   #$08,d4
      bsr.l   SEND_GA_PARAMETER
      move.l d1,d4
      lsr.l   #$04,d4
      bsr.l   SEND_GA_PARAMETER
      move.l d1,d4
      bsr.l   SEND_GA_PARAMETER

      subi.w #$08,d6
180   cmp.w   #$0000,d6
      beq     SHOW_CHROMO_PIC0_GOON
      subi.b #$01,chk_sum
      cmp.b   #$00,chk_sum
      beq     SHOW_CHROMO_PIC0
      move.b #$20,d4
      bsr.l   SEND_CHB_BYTE
      bra.l   SHOW_CHROMO_PIC0_SMALL_LOOP

    SHOW_CHROMO_PIC0_GOON
190   moveq   #32,d7
      bsr.l   SEND_SPACE
      move.b #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      move.b #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE
      moveq   #10,d7
      bsr.l   SEND_SPACE
      moveq   #65,d7
      bsr.l   SEND_LO_LINE
200   bsr.l   RECEIVERB_LOOP         * Auf Signal warten
      rts
    ***
    ******************************************************************************


    ******************************************************************************
    ***
    *** Name....: SHOW_PIC1_RES
    *** Funktion: Anzeigen des besten PIC1-Resultates
210 *** Register: D0 - D7, A5
    ***
    SHOW_PIC1_RES
      bsr.l   SHOW_RES_INTRO         * "Ergebnis von PIC"
      move.b #$31,d4
      bsr.l   SEND_CHB_BYTE
      move.l #$2020203A,d4           * ":   "
```

```
      bsr.l   SEND_CHB_1LONG
      moveq   #42,d7
      bsr.l   SEND_SPACE
220   move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE
      moveq   #10,d7
      bsr.l   SEND_SPACE
      moveq   #65,d7
      bsr.l   SEND_MI_LINE
      moveq   #10,d7
      bsr.l   SEND_SPACE
230   move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      moveq   #65,d7
      bsr.l   SEND_SPACE
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE

      bsr.l   SHOW_NR_GEN             * "Anzahl an Generationen"
240   moveq   #00,d0
      moveq   #00,d1
      move.b  gen1_hi,d0              * Anzahl Generationen
      lsl.l   #08,d0                  * ins High Byte "shiften"
      move.b  gen1_mi,d0
      lsl.l   #08,d0                  * ins High Byte "shiften"
      move.b  gen1_lo,d0
      moveq   #00,d1
      moveq   #00,d2
      bsr.l   HEX2DEZ_LONG            * Datenkonvertierung, bessere Lesbarkeit
250   bsr.l   DECODE_DEZ_LONG
      moveq   #08,d7
      bsr.l   SEND_SPACE
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE

      bsr.l   SHOW_NR_CHILDS          * "Anzahl Nachkommen pro Generation"
      moveq   #00,d0
260   move.b  child_size1,d0
      moveq   #00,d1
      moveq   #00,d2
      bsr.l   HEX2DEZ                 * Datenkonvertierung, bessere Lesbarkeit
      bsr.l   DECODE_DEZ
      moveq   #08,d7
      bsr.l   SEND_SPACE
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4                 * = "CR"
270   bsr.l   SEND_CHB_BYTE

      moveq   #00,d0
      move.b  pop_size1,d0
      bsr.l   SHOW_POPSIZE            * "Groesse der Population"
      moveq   #00,d1
      moveq   #00,d2
      bsr.l   HEX2DEZ
      bsr.l   DECODE_DEZ
      moveq   #08,d7
280   bsr.l   SEND_SPACE
      move.b  #$BA,d4                 * = "|"
      bsr.l   SEND_CHB_BYTE
      move.b  #$0D,d4                 * = "CR"
      bsr.l   SEND_CHB_BYTE

      moveq   #00,d0
      move.b  chr_sizeH,d0
      lsl.l   #08,d0
      move.l  d0,d1
```

```
290   moveq  #00,d0
      move.b chr_sizeL,d0
      add.l  d1,d0
      bsr.l  SHOW_CHROMOSIZE          * "Groesse der Chromosome"
      moveq  #00,d1
      moveq  #00,d2
      bsr.l  HEX2DEZ
      bsr.l  DECODE_DEZ
      moveq  #08,d7
      bsr.l  SEND_SPACE
300   move.b #$BA,d4                  * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                  * = "CR"
      bsr.l  SEND_CHB_BYTE

      moveq  #00,d0
      moveq  #00,d1
      move.b fitness1_hi,d0
      lsl.l  #08,d0
      move.l d0,d1
310   moveq  #00,d0
      move.b fitness1_lo,d0
      add.l  d1,d0
      move.w d0,d2
      moveq  #00,d0
      move.w #$FFFF,d0
      sub.w  d2,d0
      bsr.l  SHOW_FITNESS             * "Fitness des besten Chromosomes"
      moveq  #00,d1
      moveq  #00,d2
320   bsr.l  HEX2DEZ
      bsr.l  DECODE_DEZ

      move.b #$2C,d4                  * = ","
      bsr.l  SEND_CHB_BYTE
      moveq  #00,d0
      move.b remainder1,d0
      bsr.l  REMAINDER_OUTPUT

      moveq  #03,d7
330   bsr.l  SEND_SPACE
      move.b #$BA,d4                  * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                  * = "CR"
      bsr.l  SEND_CHB_BYTE

      move.l a1,a5
      bsr.l  SHOW_BEST_RES            * "Beste Loesung"
      moveq  #00,d6
      move.b chr_sizeH,d6
340   lsl.w  #$08,d6
      add.b  chr_sizeL,d6
      move.b #$04,chk_sum             * chk_sum dient als Schleifenzaehler
      bra.l  SHOW_CHROMO_PIC1_SMALL_LOOP

    SHOW_CHROMO_PIC1
      moveq  #32,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                  * = "|"
      bsr.l  SEND_CHB_BYTE
350   move.b #$0D,d4                  * = "CR"
      bsr.l  SEND_CHB_BYTE
      move.b #$04,chk_sum             * chk_sum dient als Schleifenzaehler
      moveq  #10,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                  * = "|"
      bsr.l  SEND_CHB_BYTE
      moveq  #18,d7
      bsr.l  SEND_SPACE

360 SHOW_CHROMO_PIC1_SMALL_LOOP
      moveq  #00,d0
      move.b (a5)+,d0
```

```
      moveq  #00,d1
      moveq  #00,d2
      bsr.l  HEX2DEZ                 * Datenkonvertierung, bessere Lesbarkeit
      move.l d1,d4
      lsr.l  #$08,d4
      bsr.l  SEND_GA_PARAMETER
      move.l d1,d4
370   lsr.l  #$04,d4
      bsr.l  SEND_GA_PARAMETER
      move.l d1,d4
      bsr.l  SEND_GA_PARAMETER

      subi.w #$08,d6
      cmp.w  #$0000,d6
      beq    SHOW_CHROMO_PIC1_GOON
      subi.b #$01,chk_sum
      cmp.b  #$00,chk_sum
380   beq    SHOW_CHROMO_PIC1
      move.b #$20,d4
      bsr.l  SEND_CHB_BYTE
      bra.l  SHOW_CHROMO_PIC1_SMALL_LOOP

SHOW_CHROMO_PIC1_GOON
      moveq  #32,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
390   move.b #$0D,d4                 * = "CR"
      bsr.l  SEND_CHB_BYTE
      moveq  #10,d7
      bsr.l  SEND_SPACE
      moveq  #65,d7
      bsr.l  SEND_LO_LINE
      bsr.l  RECEIVERB_LOOP          * Auf Signal warten
      rts
***
*******************************************************************************
400


*******************************************************************************
***
*** Name....: SHOW_PIC2_RES
*** Funktion: Anzeigen des besten PIC2-Resultates
*** Register: D0 - D7, A5
***
SHOW_PIC2_RES
      bsr.l  SHOW_RES_INTRO          * "Ergebnis von PIC"
410   move.b #$32,d4
      bsr.l  SEND_CHB_BYTE
      move.l #$2020203A,d4           * ":   "
      bsr.l  SEND_CHB_1LONG
      moveq  #42,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                 * = "CR"
      bsr.l  SEND_CHB_BYTE
420   moveq  #10,d7
      bsr.l  SEND_SPACE
      moveq  #65,d7
      bsr.l  SEND_MI_LINE
      moveq  #10,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
      moveq  #65,d7
      bsr.l  SEND_SPACE
430   move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                 * = "CR"
      bsr.l  SEND_CHB_BYTE

      bsr.l  SHOW_NR_GEN             * "Anzahl an Generationen"
```

```
        moveq   #00,d0
        moveq   #00,d1
        move.b  gen2_hi,d0              * Anzahl Generationen
        lsl.l   #08,d0                  * ins High Byte "shiften"
440     move.b  gen2_mi,d0
        lsl.l   #08,d0                  * ins High Byte "shiften"
        move.b  gen2_lo,d0
        moveq   #00,d1
        moveq   #00,d2
        bsr.l   HEX2DEZ_LONG            * Datenkonvertierung, bessere Lesbarkeit
        bsr.l   DECODE_DEZ_LONG
        moveq   #08,d7
        bsr.l   SEND_SPACE
        move.b  #$BA,d4                 * = "|"
450     bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE

        bsr.l   SHOW_NR_CHILDS          * "Anzahl Nachkommen pro Generation"
        moveq   #00,d0
        move.b  child_size2,d0
        moveq   #00,d1
        moveq   #00,d2
        bsr.l   HEX2DEZ                 * Datenkonvertierung, bessere Lesbarkeit
460     bsr.l   DECODE_DEZ
        moveq   #08,d7
        bsr.l   SEND_SPACE
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE

        moveq   #00,d0
        move.b  pop_size2,d0
470     bsr.l   SHOW_POPSIZE            * "Groesse der Population"
        moveq   #00,d1
        moveq   #00,d2
        bsr.l   HEX2DEZ
        bsr.l   DECODE_DEZ
        moveq   #08,d7
        bsr.l   SEND_SPACE
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
480     bsr.l   SEND_CHB_BYTE

        moveq   #00,d0
        move.b  chr_sizeH,d0
        lsl.l   #08,d0
        move.l  d0,d1
        moveq   #00,d0
        move.b  chr_sizeL,d0
        add.l   d1,d0
        bsr.l   SHOW_CHROMOSIZE         * "Groesse der Chromosome"
490     moveq   #00,d1
        moveq   #00,d2
        bsr.l   HEX2DEZ
        bsr.l   DECODE_DEZ
        moveq   #08,d7
        bsr.l   SEND_SPACE
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE
500
        moveq   #00,d0
        moveq   #00,d1
        move.b  fitness2_hi,d0
        lsl.l   #08,d0
        move.l  d0,d1
        moveq   #00,d0
        move.b  fitness2_lo,d0
        add.l   d1,d0
```

```
      move.w d0,d2
510   moveq  #00,d0
      move.w #$FFFF,d0
      sub.w  d2,d0
      bsr.l  SHOW_FITNESS            * "Fitness des besten Chromosoms"
      moveq  #00,d1
      moveq  #00,d2
      bsr.l  HEX2DEZ
      bsr.l  DECODE_DEZ

      move.b #$2C,d4                 * = ","
520   bsr.l  SEND_CHB_BYTE
      moveq  #00,d0
      move.b remainder2,d0
      bsr.l  REMAINDER_OUTPUT

      moveq  #03,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                 * = "CR"
530   bsr.l  SEND_CHB_BYTE

      move.l a2,a5
      bsr.l  SHOW_BEST_RES           * "Beste Loesung"
      moveq  #00,d6
      move.b chr_sizeH,d6
      lsl.w  #$08,d6
      add.b  chr_sizeL,d6
      move.b #$04,chk_sum            * chk_sum dient als Schleifenzaehler
      bra.l  SHOW_CHROMO_PIC2_SMALL_LOOP
540
SHOW_CHROMO_PIC2
      moveq  #32,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                 * = "CR"
      bsr.l  SEND_CHB_BYTE
      move.b #$04,chk_sum            * chk_sum dient als Schleifenzaehler
      moveq  #10,d7
550   bsr.l  SEND_SPACE
      move.b #$BA,d4                 * = "|"
      bsr.l  SEND_CHB_BYTE
      moveq  #18,d7
      bsr.l  SEND_SPACE

SHOW_CHROMO_PIC2_SMALL_LOOP
      moveq  #00,d0
      move.b (a5)+,d0
      moveq  #00,d1
560   moveq  #00,d2
      bsr.l  HEX2DEZ                 * Datenkonvertierung, bessere Lesbarkeit
      move.l d1,d4
      lsr.l  #$08,d4
      bsr.l  SEND_GA_PARAMETER
      move.l d1,d4
      lsr.l  #$04,d4
      bsr.l  SEND_GA_PARAMETER
      move.l d1,d4
      bsr.l  SEND_GA_PARAMETER
570
      subi.w #$08,d6
      cmp.w  #$0000,d6
      beq    SHOW_CHROMO_PIC2_GOON
      subi.b #$01,chk_sum
      cmp.b  #$00,chk_sum
      beq    SHOW_CHROMO_PIC2
      move.b #$20,d4
      bsr.l  SEND_CHB_BYTE
      bra.l  SHOW_CHROMO_PIC2_SMALL_LOOP
580
SHOW_CHROMO_PIC2_GOON
```

```
      moveq  #32,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE
      moveq  #10,d7
      bsr.l  SEND_SPACE
590   moveq  #65,d7
      bsr.l  SEND_LO_LINE
      bsr.l  RECEIVERB_LOOP         * Auf Signal warten
      rts
    ***
    ********************************************************************************


    ********************************************************************************
    ***
600 *** Name....: SHOW_PIC3_RES
    *** Funktion: Anzeigen des besten PIC3-Resultates
    *** Register: D0 - D7, A5
    ***
    SHOW_PIC3_RES
      bsr.l  SHOW_RES_INTRO         * "Ergebnis von PIC"
      move.b #$33,d4
      bsr.l  SEND_CHB_BYTE
      move.l #$2020203A,d4          * ":   "
      bsr.l  SEND_CHB_1LONG
610   moveq  #42,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE
      moveq  #10,d7
      bsr.l  SEND_SPACE
      moveq  #65,d7
      bsr.l  SEND_MI_LINE
620   moveq  #10,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      moveq  #65,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE
630
      bsr.l  SHOW_NR_GEN            * "Anzahl an Generationen"
      moveq  #00,d0
      moveq  #00,d1
      move.b gen3_hi,d0             * Anzahl Generationen
      lsl.l  #08,d0                 * ins High Byte "shiften"
      move.b gen3_mi,d0
      lsl.l  #08,d0                 * ins High Byte "shiften"
      move.b gen3_lo,d0
      moveq  #00,d1
640   moveq  #00,d2
      bsr.l  HEX2DEZ_LONG           * Datenkonvertierung, bessere Lesbarkeit
      bsr.l  DECODE_DEZ_LONG
      moveq  #08,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE

650   bsr.l  SHOW_NR_CHILDS         * "Anzahl Nachkommen pro Generation"
      moveq  #00,d0
      move.b child_size3,d0
      moveq  #00,d1
      moveq  #00,d2
```

```
        bsr.l   HEX2DEZ                 * Datenkonvertierung, bessere Lesbarkeit
        bsr.l   DECODE_DEZ
        moveq   #08,d7
        bsr.l   SEND_SPACE
        move.b  #$BA,d4                 * = "|"
660     bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE

        moveq   #00,d0
        move.b  pop_size3,d0
        bsr.l   SHOW_POPSIZE            * "Groesse der Population"
        moveq   #00,d1
        moveq   #00,d2
        bsr.l   HEX2DEZ
670     bsr.l   DECODE_DEZ
        moveq   #08,d7
        bsr.l   SEND_SPACE
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE

        moveq   #00,d0
        move.b  chr_sizeH,d0
680     lsl.l   #08,d0
        move.l  d0,d1
        moveq   #00,d0
        move.b  chr_sizeL,d0
        add.l   d1,d0
        bsr.l   SHOW_CHROMOSIZE         * "Groesse der Chromosome"
        moveq   #00,d1
        moveq   #00,d2
        bsr.l   HEX2DEZ
        bsr.l   DECODE_DEZ
690     moveq   #08,d7
        bsr.l   SEND_SPACE
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE

        moveq   #00,d0
        moveq   #00,d1
        move.b  fitness3_hi,d0
700     lsl.l   #08,d0
        move.l  d0,d1
        moveq   #00,d0
        move.b  fitness3_lo,d0
        add.l   d1,d0
        move.w  d0,d2
        moveq   #00,d0
        move.w  #$FFFF,d0
        sub.w   d2,d0
        bsr.l   SHOW_FITNESS            * "Fitness des besten Chromosomes"
710     moveq   #00,d1
        moveq   #00,d2
        bsr.l   HEX2DEZ
        bsr.l   DECODE_DEZ

        move.b  #$2C,d4                 * = ","
        bsr.l   SEND_CHB_BYTE
        moveq   #00,d0
        move.b  remainder3,d0
        bsr.l   REMAINDER_OUTPUT
720
        moveq   #03,d7
        bsr.l   SEND_SPACE
        move.b  #$BA,d4                 * = "|"
        bsr.l   SEND_CHB_BYTE
        move.b  #$0D,d4                 * = "CR"
        bsr.l   SEND_CHB_BYTE
```

```
      move.l a3,a5
      bsr.l  SHOW_BEST_RES          * "Beste Loesung"
730   moveq  #00,d6
      move.b chr_sizeH,d6
      lsl.w  #$08,d6
      add.b  chr_sizeL,d6
      move.b #$04,chk_sum           * chk_sum dient als Schleifenzaehler
      bra.l  SHOW_CHROMO_PIC3_SMALL_LOOP

    SHOW_CHROMO_PIC3
      moveq  #32,d7
      bsr.l  SEND_SPACE
740   move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE
      move.b #$04,chk_sum           * chk_sum dient als Schleifenzaehler
      moveq  #10,d7
      bsr.l  SEND_SPACE
      move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      moveq  #18,d7
750   bsr.l  SEND_SPACE

    SHOW_CHROMO_PIC3_SMALL_LOOP
      moveq  #00,d0
      move.b (a5)+,d0
      moveq  #00,d1
      moveq  #00,d2
      bsr.l  HEX2DEZ                * Datenkonvertierung, bessere Lesbarkeit
      move.l d1,d4
      lsr.l  #$08,d4
760   bsr.l  SEND_GA_PARAMETER
      move.l d1,d4
      lsr.l  #$04,d4
      bsr.l  SEND_GA_PARAMETER
      move.l d1,d4
      bsr.l  SEND_GA_PARAMETER

      subi.w #$08,d6
      cmp.w  #$0000,d6
      beq    SHOW_CHROMO_PIC3_GOON
770   subi.b #$01,chk_sum
      cmp.b  #$00,chk_sum
      beq    SHOW_CHROMO_PIC3
      move.b #$20,d4
      bsr.l  SEND_CHB_BYTE
      bra.l  SHOW_CHROMO_PIC3_SMALL_LOOP

    SHOW_CHROMO_PIC3_GOON
      moveq  #32,d7
      bsr.l  SEND_SPACE
780   move.b #$BA,d4                * = "|"
      bsr.l  SEND_CHB_BYTE
      move.b #$0D,d4                * = "CR"
      bsr.l  SEND_CHB_BYTE
      moveq  #10,d7
      bsr.l  SEND_SPACE
      moveq  #65,d7
      bsr.l  SEND_LO_LINE
      bsr.l  RECEIVERB_LOOP         * Auf Signal warten
      rts
790 ***
    ************************************************************************


    ************************************************************************
    *** Hilfsfunktion, um Nachkommastellen auszugeben.
    REMAINDER_OUTPUT
      move.b #$00,rbit0             * 1. Nachkommastelle
      move.b #$00,rbit1             * 2. Nachkommastelle
      move.b #$00,rbit2             * 3. Nachkommastelle
800
```

```
      btst.b #$00,d0
      beq    RTEST_B1
      addi.b #$04,rbit2
    RTEST_B1
      btst.b #$01,d0
      beq    RTEST_B2
      addi.b #$08,rbit2
      cmp.b  #$0A,rbit2
      blt    RTEST_B2
810   addi.b #$01,rbit1
      subi.b #$0A,rbit2
    RTEST_B2
      btst.b #$02,d0
      beq    RTEST_B3
      addi.b #$01,rbit1
      addi.b #$06,rbit2
      cmp.b  #$0A,rbit2
      blt    RTEST_B3
      addi.b #$01,rbit1
820   subi.b #$0A,rbit2
    RTEST_B3
      btst.b #$03,d0
      beq    RTEST_B4
      addi.b #$03,rbit1
      addi.b #$01,rbit2
      cmp.b  #$0A,rbit2
      blt    RTEST_B4
      addi.b #$01,rbit1
      subi.b #$0A,rbit2
830 RTEST_B4
      btst.b #$04,d0
      beq    RTEST_B5
      addi.b #$06,rbit1
      addi.b #$03,rbit2
      cmp.b  #$0A,rbit2
      blt    RTEST_B4_SEC_CHK
      addi.b #$01,rbit1
      subi.b #$0A,rbit2
    RTEST_B4_SEC_CHK
840   cmp.b  #$0A,rbit1
      blt    RTEST_B5
      addi.b #$01,rbit0
      subi.b #$0A,rbit1
    RTEST_B5
      btst.b #$05,d0
      beq    RTEST_B6
      addi.b #$01,rbit0
      addi.b #$02,rbit1
      addi.b #$05,rbit2
850   cmp.b  #$0A,rbit2
      blt    RTEST_B5_SEC_CHK
      addi.b #$01,rbit1
      subi.b #$0A,rbit2
    RTEST_B5_SEC_CHK
      cmp.b  #$0A,rbit1
      blt    RTEST_B6
      addi.b #$01,rbit0
      subi.b #$0A,rbit1
    RTEST_B6
860   btst.b #$06,d0
      beq    RTEST_B7
      addi.b #$02,rbit0
      addi.b #$05,rbit1
      cmp.b  #$0A,rbit1
      blt    RTEST_B7
      addi.b #$01,rbit0
      subi.b #$0A,rbit1
    RTEST_B7
      btst.b #$07,d0
870   beq    RTEST_RDY
      addi.b #$05,rbit0
    RTEST_RDY
      move.b rbit0,d4
```

```
      addi.b #$30,d4
      bsr.l  SEND_CHB_BYTE
      move.b rbit1,d4
      addi.b #$30,d4
      bsr.l  SEND_CHB_BYTE
      move.b rbit2,d4
880   addi.b #$30,d4
      bsr.l  SEND_CHB_BYTE
      move.b #$20,d4                  = "␣"
      bsr.l  SEND_CHB_BYTE
      rts
    ***
    ********************************************************************************
```

Anhang C

Microchip PIC17C43 Betriebssystem

Dateiname	Funktion
`tonux.asm`	Hauptschleife: Speichertest, Programm-Empfang, -Abarbeitung
`memtest.asm`	Testroutine des externen Speichers
`getprg.asm`	Programmempfang vom MC68340

Tabelle C.1: PIC17C43 Betriebssystem - Die Programm-Dateien und ihre Funktion

C.1 Die Hauptdatei: `tonux.asm`

```
;*****************************************************************
;***              PIC17C43-Betriebssystem fuer genetische Algorithmen  ***
;*** Autor:       Tobias Schubert                                      ***
;*** EMail:       schubert@informatik.uni-freiburg.de                  ***
;*** Datum:       14.09.1999                                           ***
;*** Datei-Name:  tonux.asm                                            ***
;*****************************************************************

;*****************************************************************
;***    Von Routine "memtest.a" genutzte Variablen/Speicherzellen :    ***
;***    ----------------------------------------------------------     ***
;***                                                                   ***
;***    WBUFFER    equ 0x21     Schreib-Register                      ***
;***    RBUFFERH   equ 0x22     Lese-Register (High-Byte)             ***
;***    RBUFFERL   equ 0x23     Lese-Register (Low Byte)              ***
;***    ENDOFRAM   equ 0x24     End-Adresse des Speichers             ***
;***    ERROR_LO   equ 0x25     Anzahl Fehler (Low Byte)              ***
;***    ERROR_HI   equ 0x26     Anzahl Fehler (High Byte)             ***
;***    OK         equ 0x27     Status: OK=0 --> kein Fehler          ***
;*****************************************************************

;*****************************************************************
;***    Festlegungen :                                               ***
;***    --------------                                               ***
;***                                                                 ***
;***    Modus = 0    "Normal"-Betriebsmodus des PIC17C43             ***
;***    Modus = 1    Bestaetigung eines Motorola-Interrupts          ***
;***    Modus = 2    Abarbeitung einer Anwendung                     ***
;***                                                                 ***
;***    Interrupt ausloesen mit Schreibzugriff auf Adresse $1000     ***
;***    Daten vom Motorola auslesen mit Lesezugriff auf $1100        ***
;*****************************************************************
```

```
        LIST p=PIC17C43,f=INHX32
        INCLUDE "p17c43.inc"

    ;*** Programmier-Einstellungen:
    ;*** Device:              PIC17C43
    ;*** Oscillator:          XT
    ;*** Watchdog:            TMR
    ;*** Processor Mode:      Extended Microcontroller

    ;*** Variablen-Definition:
    ;*** Wichtig: "MODE" sollte an 0x41 liegen, da diese Zelle in den
    ;*** Anwendungen nicht benutzt wird und "Mode" bei eingehenden IRQs
    ;*** getestet wird.
    SUM_INSTR   equ 0x34                    ; Anzahl Befehle in einer Zeile "Hex-Daten"
    ADDR_H      equ 0x35                    ; Ziel-Adresse der zu speichernden Befehle
    ADDR_L      equ 0x36                    ; Ziel-Adresse der zu speichernden Befehle
    ENDOFFILE   equ 0x37                    ; Ende des empfangenen PIC-Programmes = 01
    INSTR_L     equ 0x38                    ; Befehl (low byte)
    INSTR_H     equ 0x39                    ; Befehl (high byte)
    MODE        equ 0x41                    ; Modus des PIC-Prozessors
    PIC_TO_MOT  equ 0x3A                    ; Daten-Register: Daten an den Motorola
    MOT_TO_PIC  equ 0x1D                    ; Daten-Register: Daten vom Motorola

            ORG     0000
    reset   GOTO START

            ORG     0008
    int_vec GOTO    READ_FROM_MOTOROLA      ; externer Interrupt vom Motorola,
                                            ; ausgeloest Daten auslesen aus PLD

            ORG     0010
    rtcc_vec MOVLW  H'20'
            MOVWF   PCLATH
            LCALL   H'10'
            RETFIE

            ORG     0018
    rt_vec  MOVLW   H'20'
            MOVWF   PCLATH
            LCALL   H'18'
            RETFIE

            ORG     0020
    peri_vec MOVLW  H'20'
            MOVWF   PCLATH
            LCALL   H'20'
            RETFIE

    ;*** Programmstart:
            ORG     0050
    START   CLRF    PIC_TO_MOT, 1           ; Register initialisieren
            CLRF    MOT_TO_PIC, 1
            CLRF    MODE, 1
            CLRF    SUM_INSTR, 1
            CLRF    ADDR_H, 1
            CLRF    ADDR_L, 1
            CLRF    ENDOFFILE, 1
            CLRF    INSTR_L, 1
            CLRF    INSTR_H, 1

    ;*** externen PIC-Speicher testen, OK=0 --> kein Lese-/Schreib-Fehler
            CALL    MEM_TEST_START

    ;*** "Interrupt-Handling" und "Memory Map" festlegen
            CLRF    DDRB, 1                 ; PORTB als Ausgang fuer MAP0/1-Steuerung
            MOVLW   B'11111111'
            MOVWF   PORTB                   ; MAP0/1=1 --> Speicher: $2000 - $FFFF
            BSF     INTSTA, 0               ; Bit0 von PORTA setzen, d.h. als
                                            ; "int_vec" verwenden
            BCF     CPUSTA, 4               ; alle Interrupt-Quellen erlauben
```

```
        MOVLW  B'00000000'
        ADDWF  OK, 0                    ; Ergebnis des Speichertestes
        MOVWF  PIC_TO_MOT               ; in Register fuer den Motorola speichern
        CLRF   MODE, 1
110
;*** Warten bis der Motorola das Ergebnis des Speichertestes anfordert
        CALL   WAITING_FOR_MOTOROLA
        MOVLW  B'00000001'
        CPFSEQ MOT_TO_PIC
        BSF    PIC_TO_MOT, 5            ; Test der Datenuebergabe des Motorola
        CALL   WRITE_BYTE_TO_MOTOROLA

;*** Motorola signalisiert mit 00, dass der PIC Programm empfangen soll
;                          mit 01, dass der PIC keine Aufgabe erhaelt.
120     CLRF   MODE, 1
        CALL   WAITING_FOR_MOTOROLA
        MOVLW  B'00000000'
        CPFSEQ MOT_TO_PIC
        GOTO   START                    ; MOT_TO_PIC = (01)hex


;*** Empfang des abzuarbeitenden PIC-Programmes
        CALL   GET_PRG_FROM_MOTOROLA

130
;*** "MODE" aendern, um Motorola-IRQs "umzuleiten"
;*** (vgl. "READ_FROM_MOTOROLA")
        MOVLW  H'02'
        MOVWF  MODE


;*** Anwendung abarbeiten (Startadresse ist (2000)hex)
        MOVLW  H'20'
        MOVWF  PCLATH
140     LCALL  H'00'


;*** --> Programm-Anfang
        CLRWDT                          ; "Watchdog Timer" initialisieren
        GOTO   START


;*************************************************************************
;***
150 ;*** Name....: WRITE_BYTE_TO_MOTOROLA
;*** Funktion: Beim Motorola Interrupt ausloesen und "PIC_TO_MOT"
;***           uebergeben.
;*** Register: TBLPTRH, TBLPTRL um mit $1000 das PLD zu aktivieren,
;***           PIC_TO_MOT enthaelt das zu uebergebende Byte.
;***
WRITE_BYTE_TO_MOTOROLA
        MOVLW  10
        MOVWF  TBLPTRH
        CLRF   TBLPTRL, 1               ; "Table Pointer" = $1000
160     TABLWT 1, 0, PIC_TO_MOT         ; PLD generiert beim Motorola
        RETURN                          ; einen Interrupt.
;***
;*************************************************************************


;*************************************************************************
;***
;*** Name....: READ_FROM_MOTOROLA
;*** Funktion: Daten vom Motorola aus dem PLD auslesen.
170 ;***           Besonders wichtig auch fuer den Datenaustausch aus
;***           der PIC-Anwendung heraus (MODE = 2).
;*** Register: TBLPTRH, TBLPTRL um mit $1100 das PLD zu aktivieren,
;***           MOT_TO_PIC enthaelt das gelesene Byte.
;***
READ_FROM_MOTOROLA
        MOVLW  H'00'
        CPFSEQ MODE
        GOTO   $+9
```

```
180           MOVLW  0x11
              MOVWF  TBLPTRH
              CLRF   TBLPTRL, 1              ; "Table Pointer" = $1100
              TABLRD 1, 0, WREG
              TLRD   1, WREG
              MOVWF  MOT_TO_PIC
              INCF   MODE, 1                 ; Gilt als Bestaetigung
              RETFIE

              MOVLW  H'20'                   ; In Anwender-Routine verzweigen
190           MOVWF  PCLATH
              LCALL  H'08'
              RETFIE
    ;***
    ;*********************************************************************


    ;*********************************************************************
    ;*** Name....: WAITING_FOR_MOTOROLA
    ;*** Funktion: Auf Motorola-IRQ warten und zur Bestaetigung
200 ;*** "MODE" inkrementieren.
    ;***
    WAITING_FOR_MOTOROLA
              MOVLW  H'01'
              CPFSEQ MODE
              GOTO   WAITING_FOR_MOTOROLA
              RETURN
    ;*********************************************************************


210 ;*** Include-Dateien:
              INCLUDE "memtest.asm"          ; Speichertest
              INCLUDE "getprg.asm"           ; PIC-Programm vom Motorola empfangen
              END
```

C.2 PIC17C43-Speichertest: `memtest.asm`

```
    ;*********************************************************************
    ;***              Test des externen PIC-Speichers                  ***
    ;*** Autor:       Tobias Schubert                                  ***
    ;*** EMail:       schubert@informatik.uni-freiburg.de              ***
    ;*** Datum:       09.09.1999                                       ***
    ;*** Datei-Name:  memtest.asm                                      ***
    ;*********************************************************************


 10 ;*********************************************************************
    ;***                                                               ***
    ;***    Idee: Schreibe aufsteigende Werte in alle Speicherzellen und ***
    ;***          lese diese danach wieder aus (beide Speicherbloecke). ***
    ;***    Bsp.:                                                      ***
    ;***             $(2001) = 02 03                                   ***
    ;***             $(2002) = 04 05                                   ***
    ;***             $(2003) = 05 06                                   ***
    ;***             $(2004) = 06 07                                   ***
    ;***             $(2005) = 07 08                                   ***
 20 ;***             $(2006) = 09 0A usw.                              ***
    ;***                                                               ***
    ;*********************************************************************


    ;*********************************************************************
    ;***                                                               ***
    ;***    Register:                                                  ***
    ;***   ------------                                                ***
    ;***                                                               ***
```

```
;***    OK = 00                 kein Fehler                          ***
;***    OK = 01                 Fehler beim Lesen/Schreiben          ***
;***                                                                 ***
;***    ERROR_LO                Anzahl Fehler (Low Byte)             ***
;***    ERROR_HI                Anzahl Fehler (High Byte)            ***
;***                                                                 ***
;*********************************************************************


;*** Variablen-Definitionen:
WBUFFER     equ 0x21            ;Schreib-Register
RBUFFERH    equ 0x22            ;Lese-Register (High-Byte)
RBUFFERL    equ 0x23            ;Lese-Register (Low Byte)
ENDOFRAM    equ 0x24            ;End-Adresse des Speichers
ERROR_LO    equ 0x25            ;Anzahl Fehler (Low Byte)
ERROR_HI    equ 0x26            ;Anzahl Fehler (High Byte)
OK          equ 0x27            ;Status: OK=0 --> kein Fehler


;*********************************************************************
;***
;*** Name....: MEM_TEST_START
;*** Funktion: Externen Speicher des "PIC-Streifens" testen
;*** Register: TBLPTRH, TBLPTRL um Speicherzellen auszuwaehlen
;***           und obige Variablen-Definitionen
;***
MEM_TEST_START
        CLRF    DDRB, 1         ; PORTB als Ausgang, um mit MAP0/1
                                ; Speicherbereich zu waehlen
        CLRF    PORTB, 1        ; MAP0/1 = 0, Speicher: $2000 - $3FFF
        CLRF    WBUFFER, 1
        CLRF    RBUFFERH, 1
        CLRF    RBUFFERL, 1
        CLRF    ERROR_LO, 1
        CLRF    ERROR_HI, 1
        CLRF    OK, 1
        MOVLW   20
        MOVWF   TBLPTRH
        CLRF    TBLPTRL, 1      ; "Table Pointer" = $2000
        MOVLW   40
        MOVWF   ENDOFRAM


;*** Aufsteigende Werte ins RAM schreiben
WRITE_TO_RAM
        TLWT    1,WBUFFER       ; Table Latch (High Byte) = WBUFFER
        INCF    WBUFFER, 1      ; WBUFFER = WBUFFER + 1
        TABLWT  0,1,WBUFFER     ; Table Latch (High Byte) = WBUFFER
        INCF    WBUFFER, 1      ; WBUFFER = WBUFFER + 1
        MOVFP   ENDOFRAM, WREG  ; Ende des Speicherblockes?
        CPFSEQ  TBLPTRH
        GOTO    WRITE_TO_RAM    ; WREG <> TBLPTRH
        MOVLW   0x00
        CPFSEQ  TBLPTRL
        GOTO    WRITE_TO_RAM    ; WREG <> TBLPTRL

        MOVLW   40
        CPFSEQ  ENDOFRAM
        GOTO    PREPARE_TO_READ ; Lesen/Vergleichen der Werte

        MOVLW   B'11111111'     ; WREG = 255
        MOVWF   PORTB           ; MAP0/1 = 1, Speicher: $2000 - $FFFF
        MOVLW   20
        MOVWF   TBLPTRH
        CLRF    TBLPTRL, 1      ; "Table Pointer" = $2000

        CLRF    ENDOFRAM, 1
        GOTO    WRITE_TO_RAM


;*** Lesen und Vergleichen der geschriebenen Werte
PREPARE_TO_READ
        CLRF    DDRB, 1         ; PORTB als Ausgang
```

```
        CLRF    PORTB, 1          ; MAP0/1 = 0, Speicher: $2000 - $3FFF
        MOVLW   20
        MOVWF   TBLPTRH
        CLRF    TBLPTRL, 1        ; "Table Pointer" = $2000
        CLRF    WBUFFER, 1
        MOVLW   40
        MOVWF   ENDOFRAM
110
READ_FROM_RAM
        MOVFP   WBUFFER, WREG
        TABLRD  1,1,RBUFFERH
        TLRD    1,RBUFFERH
        TLRD    0,RBUFFERL
        CPFSEQ  RBUFFERH
        CALL    SRAM_ERROR        ; WREG<> RBUFFERH
        INCF    WBUFFER, 1
        MOVFP   WBUFFER, WREG
120     CPFSEQ  RBUFFERL
        CALL    SRAM_ERROR        ; WREG<> RBUFFERL
        INCF    WBUFFER, 1
        MOVFP   ENDOFRAM, WREG    ; Ende des Speicherbereiches?
        CPFSEQ  TBLPTRH
        GOTO    READ_FROM_RAM     ; WREG<> TBLPTRH
        MOVLW   0x00
        CPFSEQ  TBLPTRL           ; WREG<> TBLPTRL
        GOTO    READ_FROM_RAM

130     MOVLW   40
        CPFSEQ  ENDOFRAM
        RETURN                    ; Ende der Funktion

        MOVLW   B'11111111'       ; WREG = 255
        MOVWF   PORTB             ; MAP0/1 = 1, Speicher: $2000 - $FFFF
        MOVLW   20
        MOVWF   TBLPTRH
        CLRF    TBLPTRL, 1        ; "Table Pointer" = $2000

140     CLRF    ENDOFRAM, 1
        GOTO    READ_FROM_RAM


;*** Bei Fehlern wird OK=$01 gesetzt
SRAM_ERROR
        BSF     OK, 0
        CLRF    WREG, 1
        INCF    ERROR_LO, 1
        ADDWFC  ERROR_HI, 1
150     RETURN
;***
;*********************************************************************
```

C.3 Programmempfang vom MC68340: `getprg.asm`

```
;*********************************************************************
;***              Vom Motorola PIC-Anwendung empfangen             ***
;*** Autor:       Tobias Schubert                                  ***
;*** EMail:       schubert@informatik.uni-freiburg.de              ***
;*** Datum:       09.09.1999                                       ***
;*** Datei-Name:  getprg.asm                                       ***
;*********************************************************************


 10 ;*********************************************************************
;***
;*** Name....:  GET_PRG_FROM_MOTOROLA
;*** Funktion:  Genetische Anwendung vom Motorola empfangen
;*** Register: TBLPTRH, TBLPTRL um Speicherzellen auszuwaehlen
```

```
;***            und Variablen-Definitionen aus "tonux.asm"
;***
GET_PRG_FROM_MOTOROLA
          CLRF    MODE, 1
          CALL    WAITING_FOR_MOTOROLA
          MOVFP   MOT_TO_PIC, WREG
          MOVWF   SUM_INSTR
          MOVLW   B'00000001'
          MOVWF   PIC_TO_MOT
          CALL    WRITE_BYTE_TO_MOTOROLA          ; 'Handshake'-Signal

          CLRF    MODE, 1
          CALL    WAITING_FOR_MOTOROLA
          MOVLW   B'00000000'
          CPFSEQ  SUM_INSTR
          GOTO    $+2
          GOTO    $+3
          MOVFP   MOT_TO_PIC, WREG
          MOVWF   ADDR_H
          MOVLW   B'00000001'
          MOVWF   PIC_TO_MOT
          CALL    WRITE_BYTE_TO_MOTOROLA          ; 'Handshake'-Signal

          CLRF    MODE, 1
          CALL    WAITING_FOR_MOTOROLA
          MOVLW   B'00000000'
          CPFSEQ  SUM_INSTR
          GOTO    $+2
          GOTO    $+3
          MOVFP   MOT_TO_PIC, WREG
          MOVWF   ADDR_L
          MOVLW   B'00000001'
          MOVWF   PIC_TO_MOT
          CALL    WRITE_BYTE_TO_MOTOROLA          ; 'Handshake'-Signal
          MOVLW   B'00001000'

          CLRF    MODE, 1
          CALL    WAITING_FOR_MOTOROLA
          MOVFP   MOT_TO_PIC, WREG
          MOVWF   ENDOFFILE
          MOVLW   B'00000001'
          MOVWF   PIC_TO_MOT
          CALL    WRITE_BYTE_TO_MOTOROLA          ; 'Handshake'-Signal


;*** Gibt es ueberhaupt Befehle zu dekodieren?
          MOVLW   B'00000000'
          CPFSGT  SUM_INSTR
          GOTO    MORE_LINES


INSTR_L_LOOP1
          CLRF    MODE, 1
          CALL    WAITING_FOR_MOTOROLA
          MOVFP   MOT_TO_PIC, WREG
          MOVWF   INSTR_L
          MOVLW   B'00000001'
          MOVWF   PIC_TO_MOT
          CALL    WRITE_BYTE_TO_MOTOROLA          ; 'Handshake'-Signal

          CLRF    MODE, 1
          CALL    WAITING_FOR_MOTOROLA
          MOVFP   MOT_TO_PIC, WREG
          MOVWF   INSTR_H
          MOVFP   ADDR_L, WREG
          MOVWF   TBLPTRL
          MOVFP   ADDR_H, WREG
          MOVWF   TBLPTRH
          MOVLW   B'00000000'
          TLWT    1, INSTR_H
          TABLWT  0, 1, INSTR_L
          INCF    ADDR_L, 1
          ADDWFC  ADDR_H, 1
```

```
            MOVLW   B'00000001'
            MOVWF   PIC_TO_MOT
            CALL    WRITE_BYTE_TO_MOTOROLA          ; 'Handshake'-Signal

            DECFSZ  SUM_INSTR, 1
            DECFSZ  SUM_INSTR, 1
            GOTO    INSTR_L_LOOP1

    MORE_LINES
            MOVLW   B'00000001'
            CPFSEQ  ENDOFFILE
            GOTO    GET_PRG_FROM_MOTOROLA
            RETURN
    ;***
    ;*****************************************************************************
```

Anhang D

Programm-Dateien zum "Travelling Salesman Problem"

Dateiname	**Funktion**
`tsp.asm`	Hauptschleife des genetischen Algorithmus
`paramet.h`	Parameter des genetischen Algorithmus
`var.h`	Variablen-Definitionen (1)
`names.h`	Variablen-Definitionen (2)
`tspinc.asm`	Einbinden der Probleminstanz
`memptr.asm`	Externen Speicher partitionieren
`hlpfkt.asm`	Hilfsfunktionen für indirekte Adressierung
`random.asm`	Generierung von Zufallszahlen
`mathe.asm`	Mathebibliothek
`index.asm`	Zuordnen von Indizes zu allen Chromosomen
`fitness.asm`	Zuordnen von Fitneßwerten zu den Chromosomen
`game.asm`	Genetische Basisfunktionen wie *Roulette-Wheel-Selection*
`operator.asm`	Rekombinations-Operatoren wie PMX-Crossover, etc.
`kom.asm`	Funktionen zum Datenaustausch mit dem MC68340
`bestres.asm`	Beste Lösung an den Motorola übermitteln
`tspfit.asm`	Fitneßfunktion des Travelling Salesman Problems
`initops.asm`	Vorbereitende Befehle, um Mutation, etc. einzuleiten

Tabelle D.1: Travelling Salesman Problem - Die Programm-Dateien und ihre Funktion

D.1 Die Hauptdatei: tsp.asm

```
;*************************************************************
;***              Hauptprogramm: TSP-Problem               ***
;*** Autor:       Tobias Schubert                          ***
;*** EMail:       schubert@informatik.uni-freiburg.de      ***
;*** Datum:       22.10.1999                               ***
;*** Datei-Name:  tsp.asm                                  ***
;*************************************************************

        LIST    p=17C43, f=INHX8M
10

        include "17c43.h"               ; PIC17C43 Header-Datei
        include "paramet.h"             ; Parameter fuer den gen. Algorithmus
        include "var.h"                 ; Speicherplatz reservieren
        include "names.h"               ; Variablen definieren/ueberladen


        org 2000
            goto    Main_Loop
20
        org 2008
int_vec     movlw   0x11                ; Motorola-IRQ: Daten auslesen
            movwf   tblptrh
            clrf    tblptrl, 1
            tablrd  1, 0, temp2
            tlrd    1, temp2            ; temp2 = Motorola-Daten
            incf    temp1, 1            ; temp1 = 1 = "IRQ-Bestaetigung"
            return

30      org 2010
rtcc_vec    return

        org 2018
rt_vec      return

        org 2020
peri_vec    return

        include "tspinc.asm"            ; Einbinden der TSP-Probleminstanz
40
        org 2300
        include "memptr.asm"            ; Zeiger auf externe Speicherbereiche,
                                        ; Schreiben/Lesen des externen RAM
        include "hlpfkt.asm"            ; Einige nuetzliche Hilfsfunktionen
        include "random.asm"            ; Erzeugen von Zufalls-Bitstrings
        include "mathe.asm"             ; Mathe-Bibliothek
        include "index.asm"             ; Chromosomen Indizes zuordnen
        include "fitness.asm"           ; Chromosomen Fitness zuordnen
        include "game.asm"              ; Typische genetische Funktionen
50      include "operator.asm"          ; Mutation, Crossover, etc.
        include "kom.asm"               ; Routinen zum Datenaustausch mit
                                        ; anderen PICs ueber den Motorola
        include "bestres.asm"           ; Uebergabe der besten Loesung an den
                                        ; Motorola nach Algorithmus-Beendigung
        include "tspfit.asm"            ; Fitness-Fkt. fuer das TSP-Problem
        include "initops.asm"           ; Genetische Operatoren einleiten

Main_Loop
        ;*** PLD-Latches mit 0-Vektor initialisieren,
60      ;*** Zufallszahlen empfangen.
        clrf    intsta, 1
        bsf     intsta, 0               ; Bit 0 von PORTA als Interrupt
        bcf     cpusta, 4               ; Interrupts zulassen
        clrf    temp1, 1                ; IRQ-Bestaetigung
        call    MOTOROLA_WAITING        ; Auf Motorola-IRQ warten (temp1=1)
        movfp   temp2, wreg
        movwf   RandLo                  ; Motorola-Seed
        call    SET_TABLE_POINTER       ; tblptrl/h = $1000
        clrf    wreg, 1
70      tablwt  1, 0, wreg              ; 0-Vektor an Motorola
```

```
        clrf    temp1, 1                ; IRQ-Bestaetigung
        call    MOTOROLA_WAITING        ; Auf Motorola-IRQ warten (temp1=1)
        movfp   temp2, wreg
        movwf   RandHi                  ; Motorola-Seed
        call    SET_TABLE_POINTER       ; tblptrl/h = $1000
        clrf    wreg, 1
        tablwt  1, 0, wreg              ; 0- Vektor an Motorola

        ;*** Entfernungen zwischen allen Staedten berechnen.
        call    preprocessing

        ;*** Initialisieren einer Population.
        call    init_pop

        ;*** Berechnen der Fitness der initialen Population.
        call    compute_new_fitness

        ;*** Berechnen der Summe der Fitnesswerte, Sortieren der Indizes.
        call    update_population

        ;*** Initiale Population an den Motorola uebergeben.
        call    NEW_RESULT              ; Datentausch

        ;*** Hauptschleife des genetischen Algorithmus.
gen_alg_loop
        clrf    loop_reg4, 1
        clrf    children_counter, 1
children_loop
        call    Random100               ; Nichtdeterminismus bei Operator-Wahl
        movlw   D'51'
        cpfslt  cross_posLo
        goto    $+6

        ;*** Greedy-Crossover:
        call    make_greedy_crossover   ; 01 =< Zufallszahl =< 050
        incf    children_counter, 1
        call    make_greedy_crossover
        incf    children_counter, 1
        goto    next_ga_loop

        movlw   D'81'
        cpfslt  cross_posLo
        goto    $+5

        ;*** PMX-Crossover:
        call    make_crossover          ; 51 =< Zufallszahl =< 080
        movlw   H'02'
        addwf   children_counter, 1
        goto    next_ga_loop

        movlw   D'91'
        cpfslt  cross_posLo
        goto    $+06

        ;*** Lokale Verbesserung eines Chromosomes:
        call    make_local_improvement  ; 81 =< Zufallszahl =< 090
        incf    children_counter, 1
        call    make_local_improvement
        incf    children_counter, 1
        goto    next_ga_loop

        ;*** Mutation:
        call    make_mutation           ; 91 =< Zufallszahl =< 100
        incf    children_counter, 1
        call    make_mutation
        incf    children_counter, 1

next_ga_loop
        movlw   H'02'                   ; Anzahl Kinder inkrementieren
        addwf   loop_reg4, 1
        movlw   NO_OF_CHILDREN
        cpfseq  loop_reg4
        goto    children_loop           ; Weitere Kinder "erzeugen"
```

```
        ;*** "Steady-State-Repraesentation", d.h. die schlechtesten
        ;*** Chromosome werden pauschal durch die Kinder ersetzt.
        call    delete_n_last

        ;*** Berechnen der Summe der Fitnesswerte, Sortieren der Indizes,
        ;*** Inkrementieren des Generationszaehlers.
        call    update_population

        ;*** Datentausch einleiten?
        movlr   bank0
        incf    data_change_periode, 1  ; "Generationszaehler" erhoehen
        movlw   CHG_GEN
        cpfslt  data_change_periode
        call    NEW_RESULT              ; Datentausch

        ;*** Abbruchbedingung testen (Beachte "GOTO-Sprung"):
        goto    Improve_Test


        ;*** Ende des genetischen Algorithmus.
end_gen_alg                             ; Abbruchbedingung erreicht!
        call    get_best_index
        call    compute_tsp_fitness     ; Bestimmen der optimalen Fitness
        call    out_size                ; Parameter in den Ausgabeblock
        call    SEND_BEST_RESULTS       ; beste Loesung an Motorola geben
        return                          ; Zurueck zum "Betriebssystem"
        end
```

D.2 Parameter des Algorithmus: `paramet.h`

```
;*************************************************************
;***              Parameter des genetischen Algorithmus      ***
;*** Autor:       Tobias Schubert                            ***
;*** EMail:       schubert@informatik.uni-freiburg.de        ***
;*** Datum:       22.10.1999                                 ***
;*** Datei-Name: paramet.h                                   ***
;*************************************************************


;*************************************************************
;*** Anmerkung zu den Parametern:
;*** Speicherbereich: $2000 - $FFFF
;*** Groesse des Speichers: $FFFF-$2000 = $DFFF = d57343
;***                        = 114686 Byte
;*** PreProcessing-Matrix:  Bei max. 252 Staedten = (252*251)/2
;***                                               = d31626
;*** Max. Anzahl Chromosome: 100 --> 100*126 = d12600
;*** Max. Anzahl Kinder:      70 --> 70*126  = d8820
;*** Sonstiger Speicherbedarf: 100 Indizes, 100 Fitness-Werte
;***                           --> 100+100*2 = d300
;*** Sonstiges: d200
;***
;*** Freier Speicher: 57343 - 31626 - 12600 - 8820 - 300 - 200
;***                  = d3797
;*** Groesse des Programmes: =< 3797 Befehle, d.h. die letzte
;***                         im "Hex-File" angesprochene
;***                         Adresse darf hoechstens $5DAA sein.
;*************************************************************


;*************************************************************
;*** Gueltige Problemgroessen fuer das TSP-Problem:
;*** Max. 110 Staedte (jeweils Vielfache von 4!) mit
;*** Koordinaten aus dem Bereich 0,0 ... 180,180.
;*************************************************************
        CONSTANT        NO_OF_CITIES=D'060'
```

```
        CONSTANT        MAX_NO_OF_CITIES=D'252'


;*************************************************************
;*** Groesse der Population festlegen:
;*** Gueltig im Bereich zwischen 1 und MAX_POP_SIZE.
;*************************************************************
        CONSTANT        POP_SIZE=0x32
        CONSTANT        MAX_POP_SIZE=0x64


;*************************************************************
;*** Anzahl Nachkommen pro Generation festlegen:
;*** Gueltig im Bereich zwischen 1 und POP_SIZE-1
;*** und zwar programmbedingt eine gerade Anzahl.
;*************************************************************
        CONSTANT        NO_OF_CHILDREN=0x20
        CONSTANT        MAX_NO_OF_CHILDREN=0x46


;*************************************************************
;*** Anzahl an Generationen ohne Verbesserung des besten
;*** Chromosomes (Abbruchbedingung) angeben.
;*** Gueltig im Bereich zwischen 1 und MAX_IMP_G_LO/HI.
;*************************************************************
        CONSTANT        IMP_G_LO=0xC8           ; C8h = 200d
        CONSTANT        IMP_G_HI=0x00
        CONSTANT        MAX_IMP_G_LO=0xFF
        CONSTANT        MAX_IMP_G_HI=0xFF


;*************************************************************
;*** Anzahl an Generationen zwischen zwei Datenanfragen
;*** an den Motorola angeben.
;*** Gueltig im Bereich zwischen 1 und MAX_CHG_GEN.
;*************************************************************
        CONSTANT        CHG_GEN=0x20            ; 20h = 32d
        CONSTANT        MAX_CHG_GEN=0xFF


;*************************************************************
;*** Ueberpruefen obiger Programm-Parameter auf ihre Gueltigkeit:
;*************************************************************
        if      POP_SIZE==0 || POP_SIZE>MAX_POP_SIZE
         ERROR "Falscher Parameter: POP_SIZE"
        endif

        if      NO_OF_CHILDREN==0 || NO_OF_CHILDREN > MAX_NO_OF_CHILDREN
         ERROR "Falscher Parameter: NO_OF_CHILDREN"
        endif

        if      IMP_G_HI==0 && IMP_G_LO==0
         ERROR "Falscher Parameter: IMP_G_LO/HI"
        endif

        if      IMP_G_HI>MAX_IMP_G_HI
         ERROR "Falscher Parameter: IMP_G_HI"
        endif

        if      MAX_IMP_G_HI==IMP_G_HI && IMP_G_LO>MAX_IMP_G_LO
         ERROR "Falscher Parameter: IMP_G_LO"
        endif

        if      CHG_GEN==0 || CHG_GEN > MAX_CHG_GEN
         ERROR "Falscher Parameter: CHG_GEN"
        endif
```

D.3 Variablen-Definitionen (1): `var.h`

```
;*********************************************************************
;***              Variablen/Parameter des Hauptprogrammes einstellen ***
;*** Autor:       Tobias Schubert                                    ***
;*** EMail:       schubert@informatik.uni-freiburg.de                ***
;*** Datum:       22.10.1999                                         ***
;*** Datei-Name:  var.h                                              ***
;*********************************************************************

RandLo                  equ     0x1A            ; Zufallszahlen-Generator
RandHi                  equ     0x1B

        CONSTANT        TEMP_PTR=0x1C           ; Register fuer die Mathebibliothek
        CBLOCK          TEMP_PTR
                        temp1 , temp2 , temp3 , temp4
                        temp5 , temp6 , temp7 , temp8
                        temp9 , temp10, temp11, temp12
                        temp13, temp14, temp15, temp16
                        temp17, temp18, temp19, temp20
        ENDC

        CBLOCK          0x30                    ; Globale genetische Parameter
                        chromosom_size_lo, chromosom_size_hi
                        population_size, number_of_children
                        chrom_index, chromfitLo, chromfitHi
                        child_index, childfitLo, childfitHi
        ENDC


        CONSTANT        FITSUM_PTR=0x3A         ; Wird in "game.asm" benoetigt
        CBLOCK          FITSUM_PTR
                        fitsum1,fitsum2,fitsum3
                        generation_counterLo
                        generation_counterMi
                        generation_counterHi
                        children_counter
        ENDC

;*** Registerzelle 0x41 muss fuer das "Betriebssystem freigehalten werden!

        CBLOCK          0x42                    ; Daten des besten Chromosomes
                        best_index,bestfitLo,bestfitHi
        ENDC


        CBLOCK          0x45                    ; Adresse des besten Chromosomes
                        BEST_CHROM_PTR_LO,BEST_CHROM_PTR_HI
        ENDC


CHROM_PTR_LO            equ     0x47            ; Zeiger fuer vers. Routinen
CHROM_PTR_HI            equ     0x48
INDEX_PTR_LO            equ     0x49
INDEX_PTR_HI            equ     0x4A
FIT_PTR_LO              equ     0x4B
FIT_PTR_HI              equ     0x4C
CHILDREN_CHROM_PTR_LO   equ     0x4D
CHILDREN_CHROM_PTR_HI   equ     0x4E
CHILDREN_INDEX_PTR_LO   equ     0x4F
CHILDREN_INDEX_PTR_HI   equ     0x50
CHILDREN_FIT_PTR_LO     equ     0x51
CHILDREN_FIT_PTR_HI     equ     0x52
adrLo                   equ     0x53
adrHi                   equ     0x54


        CBLOCK          0x55                    ; Lokale Parameter vers. Routinen
                        chrom1Lo, chrom1Hi, chrom2Lo, chrom2Hi
                        chrom_index1, chrom_index2, no_of_child
                        cross_posLo, cross_posHi, save_alusta, save_remainder
        ENDC
```

```
        CONSTANT        CHILD1_CHROM=0x55        ; Zeiger auf ch_chrom1Lo/Hi
        CONSTANT        CHILD2_CHROM=0x57        ; Zeiger auf ch_chrom2Lo/Hi


        CBLOCK          0x60                     ; Zaehler fuer die "Haupt-Schleife"
                        loop_reg1, loop_reg2
                        loop_reg3, loop_reg4
        ENDC


        CBLOCK          0x64
                        data_change_periode      ; Periode zw. 2 Datentausch-Operationen
                        improve_fitness_lo       ; Max. Anzahl Generationen ohne Ver-
                        improve_fitness_hi       ; besserung des besten Chromosomes.
                        best_fit_old_lo, best_fit_old_hi
                        best_old_remainder
                        temp21, temp22
        ENDC

        CONSTANT        TSP_FITNESS_PTR=0x6C     ; Zeiger auf Fitnesswerte
        CBLOCK          TSP_FITNESS_PTR
                        tsp_fit_Lo, tsp_fit_Hi
                        tsp_temp1, tsp_temp2
                        tsp_temp3, tsp_temp4
                        tsp_temp5, tsp_temp6
                        tsp_temp7, tsp_temp8
                        start_city, new_child_index
                        next_city1, next_city2
                        next_city3, next_city4
                        tsp_ptr1_Hi, tsp_ptr1_Lo
                        tsp_ptr2_Hi, tsp_ptr2_Lo
                        tsp_ptr3_Hi, tsp_ptr3_Lo
                        tsp_ptr4_Hi, tsp_ptr4_Lo
                        distanceLo, distanceHi
                        PMX_PTR_LO, PMX_PTR_HI
                        dist_tmpLo, dist_tmpMi
                        dist_tmpHi, dist_tmpSHi
                        dist_remainder, find4opt_tmp
                        sqrtLo, sqrtMi, sqrtHi, sqrtSHi
                        old_sqrtLo, old_sqrtMi, old_sqrtHi, old_sqrtSHi
                        city1, city2, city3, city4
                        ncity1, ncity2, ncity3, ncity4
                        greedyHi, greedyLo, greedy_temp
                        pp1, pp2, msLo
                        PP_PTR_LO, PP_PTR_HI
                        tmp_ptr_Lo, tmp_ptr_Hi
        ENDC
        CONSTANT        memptr1=0xA8
```

D.4 Variablen-Definitionen (2): names.h

```
;************************************************************
;***             Variablen-Namen definieren/ueberladen    ***
;*** Autor:      Tobias Schubert                          ***
;*** EMail:      schubert@informatik.uni-freiburg.de      ***
;*** Datum:      08.10.1999                               ***
;*** Datei-Name: names.h                                  ***
;************************************************************


;************************************************************
;*** Verschiedene Funktionen nutzen die Hilfsspeicherplaetze
;*** "tempxx" unter anderen verstaendlicheren Namen.
;************************************************************
```

```
   ;*** Benutzt in "mathe.asm".
   numerator                  equ     temp1           ; Zaehler
   denominator                equ     temp2           ; Nenner
   quotiont                   equ     temp3           ; Ergebnis, Quotient
20 remainder                  equ     temp4           ; Rest
   cntr8                      equ     temp5           ; Zaehler
   numerator16Lo              equ     temp1
   numerator16Hi              equ     temp2
   denominator16Lo            equ     temp3
   denominator16Hi            equ     temp4
   quotiont16Lo               equ     temp5
   quotiont16Hi               equ     temp6
   remainder16Lo              equ     temp7
   remainder16Hi              equ     temp8
30 cntr16                     equ     temp9
   numerator24Lo              equ     temp1
   numerator24Mi              equ     temp2
   numerator24Hi              equ     temp3
   denominator24Lo            equ     temp4
   denominator24Mi            equ     temp5
   denominator24Hi            equ     temp6
   quotiont24Lo               equ     temp7
   quotiont24Mi               equ     temp8
   quotiont24Hi               equ     temp9
40 remainder24Lo              equ     temp10
   remainder24Mi              equ     temp11
   remainder24Hi              equ     temp12
   cntr24                     equ     temp13
   numerator32Lo              equ     temp1
   numerator32Mi1             equ     temp2
   numerator32Mi2             equ     temp3
   numerator32Hi              equ     temp4
   denominator32Lo            equ     temp5
   denominator32Mi1           equ     temp6
50 denominator32Mi2           equ     temp7
   denominator32Hi            equ     temp8
   quotiont32Lo               equ     temp9
   quotiont32Mi1              equ     temp10
   quotiont32Mi2              equ     temp11
   quotiont32Hi               equ     temp12
   remainder32Lo              equ     temp13
   remainder32Mi1             equ     temp14
   remainder32Mi2             equ     temp15
   remainder32Hi              equ     temp16
60 cntr32                     equ     temp17
   add16Lo                    equ     temp1
   add16Mi                    equ     temp2
   add16Hi                    equ     temp3
   add24Lo                    equ     temp1
   add24Mi1                   equ     temp2
   add24Mi2                   equ     temp3
   add24Hi                    equ     temp4
   mul24Lo                    equ     temp1
   mul24Mi1                   equ     temp2
70 mul24Mi2                   equ     temp3
   mul24Hi                    equ     temp4
   sub24Lo                    equ     temp1
   sub24Mi                    equ     temp2
   sub24Hi                    equ     temp3
   dummy                      equ     0xFF

   ;*** Benutzt in "roulette_wheel_selection".
   cum_sumLo                  equ     temp5
   cum_sumMi                  equ     temp6
80 cum_sumHi                  equ     temp7
```

D.5 Einbinden der Probleminstanz: tspinc.asm

```
;*********************************************************
;***              TSP-Instanzen einbinden              ***
;*** Autor:       Tobias Schubert                      ***
;*** EMail:       schubert@informatik.uni-freiburg.de  ***
;*** Datum:       17.10.1999                           ***
;*** Datei-Name:  tspinc.asm                           ***
;*********************************************************


        if      NO_OF_CITIES==D'004'
                include "tsp-in~1\tsp004.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0x20
                CONSTANT        CHROM_SIZE_HI=0x00
        endif

        if      NO_OF_CITIES==D'008'
                include "tsp-in~1\tsp008.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0x40
                CONSTANT        CHROM_SIZE_HI=0x00
        endif

        if      NO_OF_CITIES==D'012'
                include "tsp-in~1\tsp012.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0x60
                CONSTANT        CHROM_SIZE_HI=0x00
        endif

        if      NO_OF_CITIES==D'016'
                include "tsp-in~1\tsp016.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0x80
                CONSTANT        CHROM_SIZE_HI=0x00
        endif

        if      NO_OF_CITIES==D'020'
                include "tsp-in~1\tsp020.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0xA0
                CONSTANT        CHROM_SIZE_HI=0x00
        endif

        if      NO_OF_CITIES==D'024'
                include "tsp-in~1\tsp024.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0xC0
                CONSTANT        CHROM_SIZE_HI=0x00
        endif

        if      NO_OF_CITIES==D'028'
                include "tsp-in~1\tsp028.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0xE0
                CONSTANT        CHROM_SIZE_HI=0x00
        endif

        if      NO_OF_CITIES==D'032'
                include "tsp-in~1\tsp032.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0x00
                CONSTANT        CHROM_SIZE_HI=0x01
        endif

        if      NO_OF_CITIES==D'036'
                include "tsp-in~1\tsp036.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0x20
                CONSTANT        CHROM_SIZE_HI=0x01
```

```
        endif

        if      NO_OF_CITIES==D'040'
                include "tsp-in~1\tsp040.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0x40
                CONSTANT        CHROM_SIZE_HI=0x01
        endif

        if      NO_OF_CITIES==D'060'
                include "tsp-in~1\tsp060.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0xE0
                CONSTANT        CHROM_SIZE_HI=0x01
        endif

        if      NO_OF_CITIES==D'100'
                include "tsp-in~1\tsp100.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0x20
                CONSTANT        CHROM_SIZE_HI=0x03
        endif

        if      NO_OF_CITIES==D'140'
                include "tsp-in~1\tsp140.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0x60
                CONSTANT        CHROM_SIZE_HI=0x04
        endif

        if      NO_OF_CITIES==D'180'
                include "tsp-in~1\tsp180.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0xA0
                CONSTANT        CHROM_SIZE_HI=0x05
        endif

        if      NO_OF_CITIES==D'220'
                include "tsp-in~1\tsp220.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0xE0
                CONSTANT        CHROM_SIZE_HI=0x06
        endif

        if      NO_OF_CITIES==D'252'
                include "tsp-in~1\tsp252.tsp"
                ;*** Chromosomengroesse in 16Bit-Schritten festlegen:
                CONSTANT        CHROM_SIZE_LO=0xE0
                CONSTANT        CHROM_SIZE_HI=0x07
        endif


        ;*** Test der Parameter aus "paramet.h", ob die Fitnesswerte und/oder
        ;*** die Indizes der Population im GPR der Bank1 gehalten werden koennen,
        ;*** oder ob beide Bereiche ins externe RAM verlegt werden muessen.
        CONSTANT        SIZE=(CHROM_SIZE_LO/8)+(CHROM_SIZE_HI*0x20)
                                        ; SIZE = Anzahl Register fuer ein Chromosom

help1=0xFF - 0x20                       ; Verfuegbarer Speicher des GPR der Bank1
help2=help1-3*POP_SIZE                  ; Speicher ohne Indexblock und Fitnessblock

        if      help2>=2*SIZE
         MESSG  "Index- und Fitnessblock koennen in Bank1 resident gehalten werden."
         CONSTANT       IND=0x20        ; Zeiger auf GPR der Bank1
         CONSTANT       FIT=IND+POP_SIZE
         CONSTANT       CHROM=FIT+2*POP_SIZE
         #define        IND_FIT_RESIDENT_
         #define        IND_RESIDENT_
        else
help2=help1-POP_SIZE
         if     help2>=2*SIZE
          MESSG "Indexblock kann in Bank1 resident gehalten werden."
          CONSTANT      IND=0x20        ; Zeiger auf GPR der Bank1
```

```
          CONSTANT      FIT=IND+POP_SIZE
          CONSTANT      CHROM=FIT
          #define       IND_RESIDENT_
         else
          MESSG "Kein kompletter Block wird in Bank1 resident gehalten."
          CONSTANT      IND=0x20                ; Zeiger auf GPR der Bank1
          CONSTANT      FIT=IND
          CONSTANT      CHROM=IND
          #define       NOT_RESIDENT_
         endif
        endif
```

D.6 Partitionierung des externen Speichers: memptr.asm

```
;*************************************************************
;***             Zeiger auf das externe RAM initialisieren,  ***
;***             Schreiben/Lesen des externen RAM.           ***
;*** Autor:       Tobias Schubert                            ***
;*** EMail:       schubert@informatik.uni-freiburg.de        ***
;*** Datum:       23.09.1999                                 ***
;*** Datei-Name: memptr.asm                                  ***
;*************************************************************

;*********************************************************
;*** Funktion:  init_mem_ptr
;*** Job:       Initialisieren derjenigen Zeiger, die Speicherzellen
;***            des externen RAMs referenzieren.
init_mem_ptr
        clrf    BEST_CHROM_PTR_LO, 1
        clrf    BEST_CHROM_PTR_HI, 1
        clrf    CHROM_PTR_LO, 1
        clrf    CHROM_PTR_HI, 1
        clrf    INDEX_PTR_LO, 1
        clrf    INDEX_PTR_HI, 1
        clrf    FIT_PTR_LO, 1
        clrf    FIT_PTR_HI, 1
        clrf    CHILDREN_CHROM_PTR_LO, 1
        clrf    CHILDREN_CHROM_PTR_HI, 1
        clrf    CHILDREN_FIT_PTR_LO, 1
        clrf    CHILDREN_FIT_PTR_HI, 1
        clrf    CHILDREN_INDEX_PTR_LO, 1
        clrf    CHILDREN_INDEX_PTR_HI, 1
        clrf    temp1, 1
        clrf    temp2, 1

        ;*** Berechnung der Startadresse im externen Program Memory. Es wird nur soviel
        ;*** Speicher fuer die Daten bereitgestellt, wie auch wirklich gebraucht wird.
        movlw   (SIZE+1)/2
        movwf   temp1                   ; Platz fuer bestes Chromosom
        mullw   POP_SIZE
        movfp   prodLo,wreg
        addwf   temp1, 1                ; Platz fuer Chromosomen der Population
        movfp   prodHi,wreg
        addwfc  temp2, 1
        ifndef  IND_RESIDENT_
         movlw  (POP_SIZE+1)/2          ; Platz fuer Indizes
         addwf  temp1, 1
         clrf   wreg, 1
         addwfc temp2, 1
        endif
        ifndef  IND_FIT_RESIDENT_
         movlw  POP_SIZE                ; Platz fuer Fitnesswerte der Chromosomen
         addwf  temp1, 1
         clrf   wreg, 1
         addwfc temp2, 1
        endif
```

```
        movlw   (SIZE+1)/2              ; Platz fuer Kinder
        mullw   NO_OF_CHILDREN
        movfp   prodLo,wreg
        addwf   temp1, 1
        movfp   prodHi,wreg
        addwfc  temp2, 1
        movlw   (NO_OF_CHILDREN+1)/2    ; Platz fuer Indizes der Kinder
        addwf   temp1, 1
        clrf    wreg, 1
        addwfc  temp2, 1
        movlw   NO_OF_CHILDREN          ; Fitness der Kinder
        addwf   temp1, 1
        clrf    wreg, 1
        addwfc  temp2, 1

        setf    temp3, 1                ; temp3 = 0xFF
        movfp   temp1,wreg
        subwf   temp3, 0                ; wreg = 0xFF - temp1
        movwf   BEST_CHROM_PTR_LO
        movfp   temp2,wreg
        subwfb  temp3, 0
        movwf   BEST_CHROM_PTR_HI       ; Adresse von BEST_CHROM_PTR_LO/HI

        movlw   (SIZE+1)/2
        addwf   BEST_CHROM_PTR_LO, 0
        movwf   CHROM_PTR_LO
        movfp   BEST_CHROM_PTR_HI,wreg
        addwfc  CHROM_PTR_HI, 1         ; CHROM_PTR_LO/HI = BEST_CHROM_PTR_LO/HI
                                        ;                   + (SIZE+1) DIV 2

        movlw   (SIZE+1)/2
        mullw   POP_SIZE
        movfp   prodLo,wreg
        addwf   CHROM_PTR_LO, 0
        movwf   temp1
        movfp   prodHi,wreg
        addwfc  CHROM_PTR_HI, 0
        movwf   temp2                   ; INDEX_PTR_LO/HI = CHROM_PTR_LO/HI
                                        ;       + ((SIZE+1) DIV 2)* POP_SIZE

        ifndef  IND_RESIDENT_
         movpf  temp1,INDEX_PTR_LO
         movpf  temp2,INDEX_PTR_HI

         movlw  (POP_SIZE+1)/2
         addwf  temp1, 1
         clrf   wreg, 1
         addwfc temp2, 1
        endif

        ifndef  IND_FIT_RESIDENT_
         movpf  temp1,FIT_PTR_LO
         movpf  temp2,FIT_PTR_HI        ; FIT_PTR_LO/HI = INDEX_PTR_LO/HI
                                        ;       + (POP_SIZE+1)/2
         movlw  POP_SIZE
         addwf  temp1, 1
         clrf   wreg, 1
         addwfc temp2, 1
        endif

        movpf   temp1,CHILDREN_CHROM_PTR_LO
        movpf   temp2,CHILDREN_CHROM_PTR_HI

        movlw   (SIZE+1)/2
        mullw   NO_OF_CHILDREN
        movfp   prodLo,wreg
        addwf   CHILDREN_CHROM_PTR_LO, 0
        movwf   CHILDREN_INDEX_PTR_LO
        movfp   prodHi,wreg
        addwfc  CHILDREN_CHROM_PTR_HI, 0
        movwf   CHILDREN_INDEX_PTR_HI   ; CHILDREN_INDEX_PTR_LO/HI =
                                        ; CHILDREN_CHROM_PTR_LO/HI
                                        ;       + (SIZE+1)/2 * NO_OF_CHILDREN
```

```
            movlw   (NO_OF_CHILDREN+1)/2
            addwf   CHILDREN_INDEX_PTR_LO, 0
            movwf   CHILDREN_FIT_PTR_LO
130         movfp   CHILDREN_INDEX_PTR_HI,wreg
            addwfc  CHILDREN_FIT_PTR_HI, 1   ; CHILDREN_FIT_PTR_Hi =
                                             ; CHILDREN_INDEX_PTR_LO/HI
                                             ;                + (NO_OF_CHILDREN+1)/2
            ;*** PMX-Pointer berechnen.
            movfp   BEST_CHROM_PTR_LO, wreg
            movwf   PMX_PTR_LO
            movfp   BEST_CHROM_PTR_HI, wreg
            movwf   PMX_PTR_HI
            movlw   (SIZE/2)+1
140         subwf   PMX_PTR_LO, 1
            clrf    wreg, 1
            subwfb  PMX_PTR_HI, 1
            return
    ;*************************************************************


    ;*************************************************************
    ;*** Funktion:   Data_Copy
    ;*** Job:        Ein Chromosom von tsp_ptr1_Lo/Hi nach
150 ;***            tsp_ptr2_Lo/Hi kopieren.
    Data_Copy
            clrf    loop_reg2, 1
    Data_Copy_Loop
            incf    loop_reg2, 1
            movfp   tsp_ptr1_Lo, wreg
            movwf   tblptrl
            movfp   tsp_ptr1_Hi, wreg
            movwf   tblptrh
            tablrd  0, 0, tsp_temp1
160         tlrd    0, tsp_temp1
            tlrd    1, tsp_temp2
            movfp   tsp_ptr2_Lo, wreg
            movwf   tblptrl
            movfp   tsp_ptr2_Hi, wreg
            movwf   tblptrh
            tlwt    0, tsp_temp1
            tablwt  1, 0, tsp_temp2
            incf    tsp_ptr1_Lo, 1
            clrf    wreg, 1
170         addwfc  tsp_ptr1_Hi, 1
            incf    tsp_ptr2_Lo, 1
            clrf    wreg, 1
            addwfc  tsp_ptr2_Hi, 1
            movlw   SIZE/2
            cpfseq  loop_reg2
            goto    Data_Copy_Loop
            return
    ;*************************************************************
```

D.7 Funktionen für indirekte Adressierung: hlpfkt.asm

```
    ;*************************************************************
    ;***              Hilfsfunktionen                          ***
    ;*** Autor:       Tobias Schubert                          ***
    ;*** EMail:       schubert@informatik.uni-freiburg.de      ***
    ;*** Datum:       01.09.1999                               ***
    ;*** Datei-Name:  hlpfkt.asm                               ***
    ;*************************************************************


 10 ;*** Autoinkrement von fsr0, fsr1.
    auto_inc_fsr
            bsf     alusta,4
```

```
        bcf     alusta,5
        bsf     alusta,6
        bcf     alusta,7
        return


;*** Beende Autoinkrement oder-dekrement von fsr0 und fsr1.
no_change_fsr
        bsf     alusta,5
        bsf     alusta,7
        return


;*** Autoinkrement fsr0
auto_inc_fsr0
        bsf     alusta,4
        bcf     alusta,5
        return


;*** Autodekrement fsr0
auto_dec_fsr0
        bcf     alusta,4
        bcf     alusta,5
        return


;*** Autoinkrement fsr1
auto_inc_fsr1
        bsf     alusta,6
        bcf     alusta,7
        return


;*** Beende Autoinkrement/-dekrement von fsr0.
no_change_fsr0
        bsf     alusta,5
        return


;*** Beende Autoinkrement/-dekrement von fsr1.
no_change_fsr1
        bsf     alusta,7
        return


;*** Hilfsfunktion fuer "Roulette_Wheel_Selection"
set_cnt
        clrf    temp22,1
        bsf     temp22,0                ; temp22 = 1
        return
```

D.8 Generierung von Zufallszahlen: `random.asm`

```
;************************************************************
;***             Erzeugen von Pseudozufallssequenzen      ***
;*** Autor:      Tobias Schubert                          ***
;*** EMail:      schubert@informatik.uni-freiburg.de      ***
;*** Datum:      23.09.1999                               ***
;*** Datei-Name: random.asm                               ***
;************************************************************


;************************************************************
;*** Funktion:  Random16
;*** Job:       16-Bit Pseudo Zufallsgenerator (Microchip AN544).
;***            Noetig fuer Gauss-Zufallszahlengenerator.
```

```
Random16
        rlcf    RandHi, W
        xorwf   RandHi, W
        rlcf    wreg, F

        swapf   RandHi, F
        swapf   RandLo, W
        rlncf   wreg, F
        xorwf   RandHi, W
        swapf   RandHi, F
        andlw   0x01
        rlcf    RandLo, F
        xorwf   RandLo, F
        rlcf    RandHi, F
        return
;***********************************************************
```

D.9 Mathebibliothek: mathe.asm

```
;***********************************************************
;***             Mathe-Bibliothek                        ***
;*** Autor:      Tobias Schubert                         ***
;*** EMail:      schubert@informatik.uni-freiburg.de     ***
;*** Datum:      08.10.1999                              ***
;*** Datei-Name: mathe.asm                               ***
;***********************************************************


;***********************************************************
;*** Funktion:  compare8
;*** Job:       Vergleicht zwei 8 Bit grosse Zahlen miteinander, deren
;***            Speicheradressen sich in den Hilfsregistern "temp1" und
;***            "temp2" befinden muessen. Als Ergebnis wird die Adresse der
;***            groesseren Zahl nach "temp1" geschrieben, die Adresse
;***            der kleineren Zahl nach "temp2".
compare8
        movfp   temp1,fsr0
        movfp   temp2,fsr1

        movfp   indirekt1, wreg         ; wreg = Zahl2
        cpfslt  indirekt0               ; skip if Zahl1 < Zahl2
         goto   end_compare8            ; Hier gilt: Zahl1 >= Zahl2

        ;*** Vertauschen der Adressen
        movpf   fsr1,temp1
        movpf   fsr0,temp2

end_compare8
        return
;***********************************************************


;***********************************************************
;*** Funktion:  compare16
;*** Job:       Vergleicht zwei 16 Bit grosse Zahlen miteinander, deren
;***            Speicheradressen sich in den Hilfsregistern "temp1" und
;***            "temp2" befinden muessen. Als Ergebnis wird die Adresse der
;***            groesseren Zahl nach "temp1" geschrieben, die Adresse der
;***            kleineren Zahl nach "temp2". Eine 16-Bit grosse Zahl muss
;***            in zwei aufeinanderfolgenden Registern gespeichert sein.
compare16
        movfp   temp1,fsr0
        movfp   temp2,fsr1
        incf    fsr0, 1                 ; High-Byte der 16-Bit Zahl1
        incf    fsr1, 1                 ; High-Byte der 16-Bit Zahl2

        movfp   indirekt1,wreg          ; wreg = Zahl2Hi
```

```
        cpfslt  indirekt0               ; skip if Zahl1Hi < Zahl2Hi
 50      goto   test_equalHi16          ; Hier gilt: Zahl1Hi >= Zahl2Hi
        goto    change_pos16

test_equalHi16
        cpfseq  indirekt0               ; skip if Zahl1Hi = Zahl2Hi
         goto   end_compare16           ; Hier gilt: Zahl1Hi > Zahl2Hi
        call    compare8
        goto    end_compare8

change_pos16
 60     movfp   temp2,wreg             ; Tausch der Adressen beider Zahlen
        movfp   temp1,temp2
        movfp   wreg,temp1             ; temp1 = temp2

end_compare16
        return
;*****************************************************************


;*****************************************************************
 70 ;*** Funktion:  compare24
;*** Job:         Vergleicht zwei 24 Bit grosse Zahlen miteinander, deren
;***              Speicheradressen sich in den Hilfsregistern temp1 und
;***              temp2 befinden muessen. Als Ergebnis wird die Adresse der
;***              groesseren Zahl nach temp1 geschrieben, die Adresse der kleineren
;***              Zahl nach temp2. Benoetigt die Register temp1 und temp2.
compare24
        movfp   temp1,fsr0
        movfp   temp2,fsr1
        incf    fsr0, 1
 80     incf    fsr0, 1                ; High-Byte der 24-Bit Zahl1
        incf    fsr1, 1
        incf    fsr1, 1                ; High-Byte der 24-Bit Zahl2

        movfp   indirekt1,wreg         ; wreg = Zahl2Hi
        cpfslt  indirekt0              ; skip if Zahl1Hi < Zahl2Hi
         goto    test_equalHi24        ; Hier gilt: Zahl1Hi >= Zahl2Hi
        goto    change_pos24

test_equalHi24
 90     cpfseq  indirekt0              ; skip if Zahl1Hi = Zahl2Hi
         goto   end_compare24           ; Hier gilt: Zahl1Hi > Zahl2Hi
        call    compare16
        goto    end_compare24

change_pos24
        movfp   temp1,wreg             ; wreg  = temp1
        movfp   temp2,temp1            ; temp1 = temp2
        movfp   wreg,temp2             ; temp2 = wreg = temp1

100 end_compare24
        return
;*****************************************************************


;*****************************************************************
;*** Funktion:  add16
;*** Job:         Addiert zwei 16 Bit grosse Zahlen, deren Low-Byte-
;***              Speicheradressen sich in "fsr0" und "fsr1" befinden
;***              muessen und schreibt das Ergebnis in die Register add16Lo,
110 ;***              add16Mi, add16Hi.
add16
        clrf    add16Lo, 1
        clrf    add16Mi, 1
        clrf    add16Hi, 1
        call    auto_inc_fsr           ; Autoinkrement von fsr0, fsr1
        movpf   indirekt0,wreg
        addwf   indirekt1, 0
        movwf   add16Lo                ; add16Lo = Zahl1Lo + Zahl2Lo
        movpf   indirekt0,wreg
120     addwfc  indirekt1, 0
        movwf   add16Mi                ; add16Mi = Zahl1Hi + Zahl2Hi + carry
```

```
        clrf    add16Hi, 1
        btfsc   _carry
         incf   add16Hi, 1              ; add16Hi = add16Hi + carry

        call    no_change_fsr           ; Autoinkrement von fsr0, fsr1 beenden
        return
;*****************************************************************

;*****************************************************************
;*** Funktion:   add16_24
;*** Job:        Addiert eine 16-Bit Zahl, deren Adresse in "fsr0"
;***             stehen muss, zu einer 24-Bit Zahl, deren Adresse in "fsr1"
;***             stehen muss, und schreibt das Ergebnis nach add24Lo/Mi1/Mi2/Hi.
add16_24
        clrf    add24Lo, 1
        clrf    add24Mi1, 1
        clrf    add24Mi2, 1
        clrf    add24Hi, 1
        call    auto_inc_fsr            ; Autoinkrement von fsr0, fsr1
        movpf   indirekt0,wreg
        addwf   indirekt1, 0
        movwf   add24Lo                 ; add24Lo = Zahl16Lo + Zahl24Lo
        movpf   indirekt0,wreg
        addwfc  indirekt1, 0
        movwf   add24Mi1                ; add24Mi1 = Zahl16Hi + Zahl24Mi + carry
        clrf    wreg, 1
        addwfc  indirekt1, 0
        movwf   add24Mi2                ; add24Mi2 = Zahl24Hi + carry
        clrf    wreg, 1
        addwfc  add24Hi, 1
        call    no_change_fsr           ; Autoinkrement von fsr0, fsr1 beenden
        return
;*****************************************************************


;*****************************************************************
;*** Funktion:   sub24_16
;*** Job:        Subtrahiert eine 16-Bit-Zahl, deren Adresse in "fsr0"
;***             stehen muss, von einer 24-Bit Zahl, deren Adresse in "fsr1"
;***             stehen muss, und schreibt das Ergebnis nach "temp1" bis "temp3".
;***             Schema der Rechnung:
;***             Zahl24 = Zahl24Hi * 2^16 + Zahl24Mi * 2^8 + Zahl24Lo
;***             Zahl16 = Zahl16Hi * 2^8  + Zahl16Lo
;***             Zahl24-Zahl16 = Zahl24Hi*2^16 +
;***                             (Zahl24Mi-Zahl16Hi)*2^8 + (Zahl24Lo-Zahl16Lo)
sub24_16
        call    auto_inc_fsr            ; Autoinkrement von fsr0, fsr1

        clrf    sub24Lo, 1
        clrf    sub24Mi, 1
        clrf    sub24Hi, 1

        movpf   indirekt0,wreg          ; wreg = Zahl16Lo
        subwf   indirekt1, 0            ; wreg = Zahl24Lo - Zahl16Lo
        movwf   sub24Lo
        movpf   indirekt0,wreg          ; wreg = Zahl16Hi
        subwfb  indirekt1, 0            ; wreg = Zahl24Mi - Zahl16Hi - /c
        movwf   sub24Mi
        clrf    wreg, 1
        subwfb  indirekt1, 0            ; wreg = Zahl24Hi - 0 - /c
        movwf   sub24Hi

        call    no_change_fsr           ; Autoinkrement von fsr0, fsr1 beenden
        return
;*****************************************************************


;*****************************************************************
;*** Funktion:   div8
;*** Job:        Berechnet  Zaehler DIV Nenner (<= 8 Bit grosse Zahlen)
;***             und schreibt das Ergebnis nach quotiont, den Rest der
;***             Division nach remainder. Der Zaehler muss dabei in numerator
```

```
;***            stehen und der Nenner in denominator.
;***            Temp1 bis temp5 werden fuer Rechnung benoetigt.
div8
        clrf    quotiont,1              ; quotiont = Ergebnis
        clrf    remainder,1             ; remainder = Rest
200     clrf    cntr8,1                 ; cntr8 dient als Zaehler
        bsf     cntr8,3                 ; cntr8=8

div8loop
        bcf     _carry                  ; carry = 0
        rlcf    numerator,1
        rlcf    remainder,1
        movfp   denominator,wreg
        subwf   remainder,0             ; wreg = Rest - Teiler
        btfss   _carry                  ; skip if alusta[0] = 1
210      goto   no_sub8

sub_zahl_rest8
        movfp   denominator,wreg
        subwf   remainder,1             ; remainder = remainder - Teiler
        bsf     _carry

no_sub8
        rlcf    quotiont,1
        decfsz  cntr8,1
220      goto   div8loop
        return
;*****************************************************************


;*****************************************************************
;*** Funktion:   mod8
;*** Job:        Berechnet   Zahl MOD Teiler   und schreibt das Ergebnis
;***             nach remainder. Die Zahl muss dabei in numerator stehen
;***             und der Teiler in denominator.
230 mod8
        call    div8
        return
;*****************************************************************


;*****************************************************************
;*** Funktion:   div16
;*** Job:        Berechnet Zahl1 DIV Zahl2, wobei Zahl1 eine 16 Bit Zahl,
;***             die sich in numerator16Lo/Hi befindet, Zahl2 ist eine 16
240 ;***         Bit Zahl in denominatorLo/Hi. Das Ergebnis wird nach
;***             quotiont16Lo/Hi geschrieben, der Rest der Division nach
;***             remainder16Lo/Hi. Temp1 bis temp9 werden benoetigt.
div16
        clrf    quotiont16Lo,1
        clrf    quotiont16Hi,1          ; Ergebnis
        clrf    remainder16Lo,1
        clrf    remainder16Hi,1         ; Rest
        clrf    cntr16,1                ; cntr16 dient als Zaehler
        bsf     cntr16,4                ; cntr16 = 16
250
div16loop
        bcf     _carry                  ; loesche carry-Bit
        rlcf    numerator16Lo,1         ; rotate left with carry Zahl1Lo
        rlcf    numerator16Hi,1         ; rotate left with carry Zahl1Hi
        rlcf    remainder16Lo,1
        rlcf    remainder16Hi,1
        movfp   denominator16Hi,wreg    ; wreg = Zahl2Hi
        subwf   remainder16Hi,0         ; wreg = remainder16Hi - zahl2Hi
        btfss   _zero                   ; skip if alusta[2]=1
260      goto   not_zero16
        movfp   denominator16Lo,wreg    ; wreg = Zahl2Lo
        subwf   remainder16Lo,0         ; wreg = remainder16Lo - Zahl2Lo

not_zero16
        btfss   _carry                  ; skip if alusta[0]=1 -> carry,
                                        ; falls Zahl2<Rest.
         goto   no_sub16
```

```
    sub_z2_r16
270         movfp   denominator16Lo,wreg     ; wreg   = Zahl2Lo
            subwf   remainder16Lo, 1         ; remainder16Lo = remainder16Lo - denominator16Lo
            movfp   denominator16Hi,wreg
            subwfb  remainder16Hi, 1         ; remainder16Hi = remainder16Hi - Zahl2Hi - /c
            bsf     _carry                   ; alusta[0] = 1

    no_sub16
            rlcf    quotiont16Lo, 1
            rlcf    quotiont16Hi, 1
            decfsz  cntr16, 1
280          goto   div16loop
            return
    ;***************************************************************


    ;***************************************************************
    ;*** Funktion:  mod16
    ;*** Job:       Berechnet Zahl1 MOD Zahl2, wobei Zahl1 eine 16 Bit
    ;***            Zahl in numerator16Lo und numerator16Hi ist und Zahl2
    ;***            eine 16 Bit Zahl an Speicheradresse in denominator16Lo
290 ;***            und denominator16Hi ist. Das Ergebnis wird nach
    ;***            remainder16Lo und remainder16Hi geschrieben.
    mod16
            call    div16
            return
    ;***************************************************************


    ;***************************************************************
    ;*** Funktion:  div24
300 ;*** Job:       Berechnet Zahl1 DIV Zahl2, wobei Zahl1 eine 24 Bit
    ;***            Zahl, die sich in numerator24Lo/Mi/Hi befindet, Zahl2
    ;***            ist eine 24 Bit Zahl in denominator24Lo/Mi/Hi. Das
    ;***            Ergebnis wird nach quotiont24Lo/Mi/Hi geschrieben, der
    ;***            Rest der Division nach remainder24Lo/Mi/Hi.
    ;***            Temp1 bis temp13 werden benoetigt.
    div24
            clrf    quotiont24Lo, 1
            clrf    quotiont24Mi, 1
            clrf    quotiont24Hi, 1          ; Ergebnis
310         clrf    remainder24Lo, 1
            clrf    remainder24Mi, 1
            clrf    remainder24Hi, 1         ; Rest

            movlw   .24
            movwf   cntr24                   ; cntr24 = 24, dient als Zaehler

    div24loop
            bcf     _carry                   ; loesche carry-Bit
            rlcf    numerator24Lo, 1
320         rlcf    numerator24Mi, 1
            rlcf    numerator24Hi, 1         ; rotate left with carry Zahl1
            rlcf    remainder24Lo, 1
            rlcf    remainder24Mi, 1
            rlcf    remainder24Hi, 1         ; rotate left with carry Rest

            movfp   denominator24Hi,wreg
            subwf   remainder24Hi, 0
            btfss   _zero
             goto   not_zero24
330         movfp   denominator24Mi,wreg
            subwf   remainder24Mi, 0
            btfss   _zero
             goto   not_zero24
            movfp   denominator24Lo,wreg
            subwf   remainder24Lo, 0         ; Pruefen, ob Zahl2 > Rest

    not_zero24
            btfss   _carry                   ; carry gesetzt, falls Zahl2 < Rest.
             goto   no_sub24
340
```

```
    sub_z2_r24
            movfp   denominator24Lo,wreg     ; wreg   = Zahl2Lo
            subwf   remainder24Lo, 1         ; remainder24Lo = remainder24Lo - Zahl2Lo
            movfp   denominator24Mi,wreg
            subwfb  remainder24Mi, 1
            movfp   denominator24Hi,wreg
            subwfb  remainder24Hi, 1         ; denominator24Lo = denominator24Lo
                                             ; - Zahl2Hi - /c
            bsf     _carry                   ; alusta[0] = 1
350
    no_sub24
            rlcf    quotiont24Lo, 1
            rlcf    quotiont24Mi, 1
            rlcf    quotiont24Hi, 1
            decfsz  cntr24, 1
             goto   div24loop
            return
    ;***************************************************************

360
    ;***************************************************************
    ;*** Funktion:  mod24
    ;*** Job:       Berechnet Zahl1 (numerator24Lo/Mi/Hi) MOD Zahl2
    ;***            (denominator24Lo/Mi/Hi) und schreibt
    ;***            das Ergebnis nach remainder24Lo/Mi/Hi.
    mod24
            call    div24
            return
    ;***************************************************************
370
    ;***************************************************************
    ;*** Funktion:  div32
    ;*** Job:       Berechnet Zahl1 DIV Zahl2. Zahl1 muss in numerator32Lo/Mi1/Mi2/Hi
    ;***            stehen, Zahl2 in denominator32Lo/Mi1/Mi2/Hi. Das Ergebnis wird
    ;***            nach quotiont32Lo/Mi1/Mi2/Hi geschrieben, der Rest der Division
    ;***            nach remainder32Lo/Mi1/Mi2/Hi.
    div32
            clrf    quotiont32Lo, 1
            clrf    quotiont32Mi1, 1
380         clrf    quotiont32Mi2, 1
            clrf    quotiont32Hi, 1          ; Ergebnis
            clrf    remainder32Lo, 1
            clrf    remainder32Mi1, 1
            clrf    remainder32Mi2, 1
            clrf    remainder32Hi, 1         ; Rest
            clrf    cntr32, 1                ; cntr32 dient als Zaehler
            bsf     cntr32, 5                ; cntr32 = 32

    div32loop
390         bcf     alusta, 0                ; loesche carry-Bit

            rlcf    numerator32Lo, 1
            rlcf    numerator32Mi1, 1
            rlcf    numerator32Mi2, 1
            rlcf    numerator32Hi, 1         ; rotate left with carry Zahl1

            rlcf    remainder32Lo, 1
            rlcf    remainder32Mi1, 1
            rlcf    remainder32Mi2, 1
400         rlcf    remainder32Hi, 1         ; rotate left with carry Rest

            movfp   temp8,wreg
            subwf   remainder32Hi, 0
            btfss   alusta,2
             goto   not_zero32
            movfp   temp7,wreg
            subwf   remainder32Mi2, 0
            btfss   alusta,2
             goto   not_zero32
410         movfp   temp6,wreg
            subwf   remainder32Mi1, 0
            btfss   alusta,2
             goto   not_zero32
```

```
        movfp   temp5,wreg
        subwf   remainder32Lo, 0        ; Pruefen, ob Zahl2 > Rest

not_zero32
        btfss   alusta,0                ; skip if alusta[0]=1 --> carry,
                                        ; falls Zahl2 < Rest.
420      goto   no_sub32

sub_z2_r32
        movfp   temp5,wreg
        subwf   remainder32Lo, 1
        movfp   temp6,wreg
        subwfb  remainder32Mi1, 1
        movfp   temp7,wreg
        subwfb  remainder32Mi2, 1
        movfp   temp8,wreg
430     subwfb  remainder32Hi, 1        ; Rest = Zahl2 - Rest
        bsf     alusta,0                ; alusta[0] = 1

no_sub32
        rlcf    quotiont32Lo, 1
        rlcf    quotiont32Mi1, 1
        rlcf    quotiont32Mi2, 1
        rlcf    quotiont32Hi, 1         ; Rotate left with carry Ergebnis

        decfsz  cntr32, 1
440      goto   div32loop
        return
;*****************************************************************
```

D.10 Index-Zuordnung: index.asm

```
;*****************************************************************
;***              Index-Berechnungen                           ***
;*** Autor:       Tobias Schubert                              ***
;*** EMail:       schubert@informatik.uni-freiburg.de          ***
;*** Datum:       02.09.1999                                   ***
;*** Datei-Name:  index.asm                                    ***
;*****************************************************************


;*****************************************************************
;*** Funktion:  get_index
;*** Job:       Schreibt die Adresse des Index des wreg-besten Elements
;***            der aktuellen Population in das Zeigerregister fsr0 und
;***            den wreg-besten Index selbst ins wreg.
;***            Vor: Indexblock steht vollstaendig in Bank1.
get_index
        sublw   IND+POP_SIZE            ; wreg = IND+POP_SIZE-wreg
        movwf   fsr0
        movlr   bank1
        movpf   indirekt0,wreg
        movlr   bank0
        return
;*****************************************************************


;*****************************************************************
;*** Funktion:  get_index_ext
;*** Job:       Schreibt die Adresse des Index des wreg-besten
;***            Elements der aktuellen Population nach tblptrl/h
;***            und den wreg-besten Index selbst ins wreg.
get_index_ext
        movwf   numerator
        movlw   (POP_SIZE+1)/2 - 1
        addwf   INDEX_PTR_LO, 0
        movwf   adrLo
```

```
        clrf    wreg, 1
        addwfc  INDEX_PTR_HI, 0
        movwf   adrHi                   ; adrLo/Hi = Adresse des Index an
                                        ; hoechster Position im Indexblock
        if      POP_SIZE%2==.0          ; POP_SIZE gerade
         decf   numerator, 1
        endif

        bcf     _carry
        rrcf    numerator, 0
        movwf   quotiont                ; quotiont=numerator/2
        movpf   alusta,save_alusta      ; Retten des alusta Zustandes
        subwf   adrLo, 0                ; wreg = adrLo/Hi - quotiont
        movwf   tblptrl
        clrf    wreg, 1
        subwfb  adrHi, 0
        movwf   tblptrh                 ; Adresse des gesuchten Index
                                        ; im externen Speicher.
        tablrd  0,0,dummy               ; updates tablatch
        btfsc   save_alusta,0           ; skip if Rest=0 --> read High_Byte
         goto   read_low_byte
        tlrd    1,wreg
        goto    end_get_ind_ext

read_low_byte
        tlrd    0,wreg

end_get_ind_ext
        return
;*************************************************************
```

D.11 Fitneßwert-Zuordnung: `fitness.asm`

```
;*************************************************************
;***             Fitness-Funktionen                        ***
;*** Autor:      Tobias Schubert                           ***
;*** EMail:      schubert@informatik.uni-freiburg.de       ***
;*** Datum:      02.09.1999                                ***
;*** Datei-Name: fitness.asm                               ***
;*************************************************************


;*************************************************************
;*** Funktion:  fitness
;*** Job:       Schreibt Adresse von FitLo des Chromosoms mit Index
;***            wreg in das Zeigerregister fsr0 und ins wreg. Der
;***            Index des Chromosoms muss vor dem Aufruf im wreg stehen.
;***            Vor: Die Fitnesswerte muessen in Bank1 stehen.
fitness
        mullw   2
        movfp   prodLo,wreg
        addlw   FIT-2           ; wreg = FIT + 2*index - 2
        movwf   fsr0
        return
;*************************************************************


;*************************************************************
;*** Funktion:  fitness_ext
;*** Job:       Schreibt externe Programmspeicheradresse von FitLo
;***            des Chromsomos mit Index wreg nach tablptrl/h.
fitness_ext
        decf    wreg, 1
        addwf   FIT_PTR_LO, 0
        movwf   tblptrl
        clrf    wreg, 1
        addwfc  FIT_PTR_HI, 0
```

```
        movwf   tblptrh                 ; tblptrl/h = FIT_PTR_LO/HI + (i-1)
        tablrd  0,0,dummy               ; updates tablatch
        return
;***********************************************************
```

D.12 Genetische Basisfunktionen: game.asm

```
;***************************************************************
;***             Bibliothek typischer genetischer Funktionen ***
;*** Autor:       Tobias Schubert                             ***
;*** EMail:       schubert@informatik.uni-freiburg.de         ***
;*** Datum:       22.10.1999                                  ***
;*** Datei-Name: game.asm                                     ***
;***************************************************************


;***************************************************************
;*** Funktion:  init_pop
;*** Job:       Zufaellige Initialisierung einer Population, Initialisierung
;***            des Indexblockes, des Fitnessblockes und des besten Elements.
init_pop
        ;*** Initialisieren der Zeigerregister fuer indirekte Adressierung.
        ;*** Adressen geben Speicherzellen im externen Program Memory an.
        call    init_mem_ptr

        ;*** Register loeschen/initialisieren.
        movlr   bank0
        clrf    data_change_periode,1   ; Periode zw. 2 Datentausch-Operationen
        clrf    improve_fitness_lo,1    ; Max. Anzahl Generationen ohne Ver-
        clrf    improve_fitness_hi,1    ; besserung des besten Chromosomes.
        clrf    best_fit_old_lo,1
        clrf    best_fit_old_hi,1
        clrf    best_old_remainder,1

        ;*** Initialisieren der Chromosome mit einer Zufallssequenz der Groesse
        ;*** CHROM_SIZE, beginnend ab Adresse "CHROM_PTR". Die einzelnen
        ;*** Chromosome koennen anschliessend mittels CHROM_PTR + (index-1)*SIZE
        ;*** angesprochen werden.

        ;*** Idee: Erzeuge einen String mit Stadt-Indizes aufsteigend,
        ;*** d.h. 00 01 02 03.

        ;*** Startadresse fuer die Chromsome.
        movfp   CHROM_PTR_HI,wreg
        movwf   tblptrh
        movfp   CHROM_PTR_LO,wreg
        movwf   tblptrl

        ;*** Hauptschleife: POP_SIZE-oft.
        movlr   bank0
        clrf    loop_reg1,1
popinitloop
        incf    loop_reg1,1             ; Chromosomenzaehler

        ;*** Erzeugen eines Chromosomes.
        movfp   loop_reg1,1

        clrf    loop_reg2,1
popinitloop2
        incf    loop_reg2,1
        movfp   loop_reg2,wreg
        rlncf   wreg,1
        tlwt    1,wreg
        decf    wreg,1
        tablwt  0,1,wreg                ; Schreiben + Zeiger inkrementieren

        movlw   NO_OF_CITIES/2
```

```
        cpfseq  loop_reg2
        goto    popinitloop2

        ;*** Ende der POP_SIZE-vielen Chromosome?
        movlw   POP_SIZE
        cpfseq  loop_reg1
        goto    popinitloop             ; Auslassen, falls loop_reg1 = POP_SIZE

        ;*** Jedes Chromosom durchlaeuft vielfache Mutationen, bei denen
70      ;*** die bessere Route immer gespeichert wird (Abbruch: 100 Mutationen,
        ;*** ohne Verbesserung der jeweils letzten Route).
        movlr   bank0                   ; Hauptschleife: POP_SIZE-oft
        clrf    loop_reg1, 1
MS_Chromo_Loop
        incf    loop_reg1, 1            ; Chromosomenzaehler

        ;*** Aktuelle Fitness bestimmen.
        movfp   loop_reg1, wreg         ; Bestimme Fitness von Chromo. mit Index wreg
        call    compute_tsp_fitness     ; Bestimmen der Fitness der akt. TSP-Route
80      movfp   tsp_fit_Hi, wreg
        movwf   greedyHi                ; greedyHi = High-Part der Fitness
        movfp   tsp_fit_Lo, wreg
        movwf   greedyLo                ; greedyLo = Low-Part der Fitness

        clrf    msLo, 1                 ; Iterations-Zaehler (Low Byte)
MS_Loop
        ;*** Zeiger aktualisieren.
        movfp   loop_reg1, wreg
        call    get_indiv
90      movfp   tblptrh, wreg
        movwf   tsp_ptr1_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr1_Lo

        movlw   H'01'
        call    get_child               ; tblptrl/h = adrLo/Hi = Adresse des Kindes
        movfp   tblptrh, wreg
        movwf   tsp_ptr3_Hi
        movwf   tsp_ptr2_Hi
100     movfp   tblptrl, wreg
        movwf   tsp_ptr3_Lo
        movwf   tsp_ptr2_Lo

        ;*** Chromosom an den Platz des Kindes kopieren.
        call    Data_Copy

        ;*** Mutation ausfuehren.
        call    tsp_mutation            ; Mutation ausfuehren

110     ;*** Neues Chromosom beurteilen.
        clrf    tsp_fit_Lo, 1
        clrf    tsp_fit_Hi, 1
        movlw   H'01'
        call    get_child               ; tblptrl/h = adrLo/Hi = Adresse des Kindes
        call    compute_tsp_fitness_start2

        ;*** Ist das neue Chromosom besser?
        movfp   greedyHi, wreg
        cpfsgt  tsp_fit_Hi
120     goto    $+2
        goto    NEW_BETTER_CHR          ; tsp_fit_Hi > greedyHi
        movfp   greedyHi, wreg
        cpfseq  tsp_fit_Hi
        goto    NEXT_TURN
        movfp   greedyLo, wreg          ; tsp_fit_Hi = greedyHi
        cpfsgt  tsp_fit_Lo
        goto    NEXT_TURN

NEW_BETTER_CHR
130     ;*** Neues Chromosom an die Stelle des Alten setzen,
        ;*** Fitness-Werte aktualisieren.
        movfp   tsp_fit_Lo, wreg
        movwf   greedyLo
```

```
        movfp   tsp_fit_Hi, wreg
        movwf   greedyHi
        clrf    msLo, 1
        movfp   loop_reg1, wreg
        call    get_indiv
        movfp   tblptrh, wreg
        movwf   tsp_ptr2_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr2_Lo
        movlw   H'01'
        call    get_child               ; tblptrl/h = adrLo/Hi = Adresse des Kindes
        movfp   tblptrh, wreg
        movwf   tsp_ptr1_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr1_Lo
        call    Data_Copy
        goto    $+2

NEXT_TURN
        incf    msLo, 1
        movlw   H'64'
        cpfseq  msLo
        goto    MS_Loop

        ;*** Ende der POP_SIZE-vielen Chromosome?
        movlw   POP_SIZE
        cpfseq  loop_reg1
        goto    MS_Chromo_Loop          ; Auslassen, falls loop_reg1 = POP_SIZE

        ;*** Initialisieren des Indexblockes: Index von Chromosom j ist j.
        call    auto_dec_fsr0
        movlw   IND+POP_SIZE-1
        movwf   fsr0
        movlw   POP_SIZE
        movwf   loop_reg1

indexinit
        movfp   loop_reg1,wreg
        movlr   bank1
        movwf   indirekt0
        movlr   bank0
        decfsz  loop_reg1, 1
         goto   indexinit
        call    no_change_fsr0

        ;*** Speichern des Indexblockes im externen Program Memory.
        ifndef  IND_RESIDENT_
         movlw  POP_SIZE
         movwf  temp1
         movlw  IND
         movwf  fsr0
         movfp  INDEX_PTR_LO,tblptrl
         movfp  INDEX_PTR_HI,tblptrh
         movlr  bank1
         call   auto_inc_fsr0
         tlwt   0, indirekt0
         decfsz temp1, 1
         goto   $+2
         goto   $+5
         tablwt 1, 1, indirekt0
         decfsz temp1, 1
         goto   $-6
         goto   $+2
         tablwt 1, 1,temp1
         call   no_change_fsr
         movlr  bank0
        endif

        ;*** Initialisieren der Fitness der Chromosome und des besten Individuums.
        ;*** Initialisieren der Summe der Fitness.
        clrf    fitsum1, 1
        clrf    fitsum2, 1
        clrf    fitsum3, 1              ; Initialisieren der Fitsum
```

```
        clrf    bestfitHi, 1            ; Fitness des besten Elements = 0
        clrf    bestfitLo, 1
        clrf    temp2, 1

init_fit
        movlw   POP_SIZE
        movwf   loop_reg1

        ifndef  IND_FIT_RESIDENT_
         movfp  FIT_PTR_LO,tblptrl
         movfp  FIT_PTR_HI,tblptrh
fitinitloop1
         tlwt   0,temp2
         tablwt 1,1,temp2
         decfsz loop_reg1, 1
          goto  fitinitloop1
        else
         call   auto_inc_fsr0
         movlw  FIT
         movwf  fsr0
fitinitloop2
         movlr  bank1
         movfp  temp2,indirekt0
         movfp  temp2,indirekt0
         movlr  bank0
         decfsz loop_reg1, 1
          goto  fitinitloop2
         call   no_change_fsr0
        endif

        clrf    best_index, 1

        ;*** Generationszaehler und Children Counter auf 0 setzen.
        clrf    generation_counterLo, 1
        clrf    generation_counterMi, 1
        clrf    generation_counterHi, 1
        clrf    children_counter, 1

        return
;*****************************************************************


;*****************************************************************
;*** Funktion:  get_chrom
;*** Job:       Schreibt die Adresse des wreg besten Chromosoms
;***            in die Register tblptrl/h.
get_chrom
        ifdef   IND_RESIDENT_
         call   get_index               ; wreg = Index des wreg-besten Chromosoms
        else
         call   get_index_ext
        endif
        call    get_indiv               ; tblptrl/h = CHROM_PTR_LO/HI
        return                           ;             + ((SIZE+1)/2)*(i-1))
;*****************************************************************


;*****************************************************************
;*** Funktion:  get_best_index
;*** Job:       Schreibt die Adresse des Index des besten Chromosoms
;***            der aktuellen Population in das Zeigerregister fsr0
;***            bzw. nach tblptrl/h und den Index selbst in das wreg.
get_best_index
        movlw   .1
        ifdef   IND_RESIDENT_
         call   get_index
        else
         call   get_index_ext
        endif
        return
;*****************************************************************
```

```
280
    ;*****************************************************************
    ;*** Funktion:   get_best_chrom
    ;*** Job:        Schreibt die Adresse des besten Chromosoms nach tblptrl/h.
    get_best_chrom
            movlw   .1
            call    get_chrom
            return
    ;*****************************************************************

290
    ;*****************************************************************
    ;*** Funktion:   get_worst_index
    ;*** Job:        Schreibt die Adresse des Index des schlechtesten Chromosoms
    ;***             in das Zeigerregister fsr0 bzw. nach tblptrl/h und den Index
    ;***             selbst in das wreg.
    get_worst_index
            movlw   POP_SIZE
            ifdef   IND_RESIDENT_
             call   get_index
300         else
             call   get_index_ext
            endif
            return
    ;*****************************************************************


    ;*****************************************************************
    ;*** Funktion:   get_worst_chrom
    ;*** Job:        Schreibt die Adresse des schlechtesten Chromosoms nach tblptrl/h.
310 get_worst_chrom
            movlw   POP_SIZE
            call    get_chrom
            return
    ;*****************************************************************


    ;*****************************************************************
    ;*** Funktion:   get_indiv
    ;*** Job:        Schreibt die Adresse von Chromosom mit Index wreg
320 ;***             in die Register tblprtl/h.
    get_indiv
            decf    wreg, 1
            mullw   (SIZE+1)/2
            movfp   prodLo,wreg
            addwf   CHROM_PTR_LO, 0
            movwf   tblptrl
            movfp   prodHi,wreg
            addwfc  CHROM_PTR_HI, 0
            movwf   tblptrh                 ; tblptrl/h = CHROM_PTR_LO/HI
330         tablrd  0,0,dummy               ;             + ((SIZE+1)/2 * (i-1))
            return
    ;*****************************************************************


    ;*****************************************************************
    ;*** Funktion:   get_child
    ;*** Job:        Schreibt die Adresse des Kindes mit Index wreg in die
    ;***             Register tblptrl/h und nach adrLo/Hi.
    get_child
340         decf    wreg, 1
            mullw   (SIZE+1)/2
            movfp   prodLo,wreg
            addwf   CHILDREN_CHROM_PTR_LO, 0
            movwf   tblptrl
            movwf   adrLo
            movfp   prodHi,wreg
            addwfc  CHILDREN_CHROM_PTR_HI, 0
            movwf   tblptrh                 ; tblptrl/h = CHILDREN_CHROM_PTR_LO/HI
            movwf   adrHi                   ;             + ((SIZE+1)/2*(i-1))
350         tablrd  0,0,dummy
            return
    ;*****************************************************************
```

```
;*****************************************************************
;*** Funktion:   child_fit
;*** Job:        Schreibt die Adresse der Fitnesswerte von Child
;***             wreg in die Register tblptrl/h und die Fitnesswerte
;***             selbst in die Table Latches.
child_fit
        decf    wreg, 1
        addwf   CHILDREN_FIT_PTR_LO, 0
        movwf   tblptrl
        clrf    wreg, 1
        addwfc  CHILDREN_FIT_PTR_HI, 0
        movwf   tblptrh                 ; tblptrl/h = CHILDREN_FIT_PTR_LO/HI + (i-1)
        tablrd  0,0,dummy
        return
;*****************************************************************

;*****************************************************************
;*** Funktion:   out_size
;*** Job:        Schreibt die Daten POP_SIZE, CHROM_SIZE_LO/HI,
;***             NO_OF_CHILDREN in das Ausgabefenster.
out_size
        movlw   POP_SIZE
        movwf   population_size
        movlw   NO_OF_CHILDREN
        movwf   number_of_children
        movlw   CHROM_SIZE_LO
        movwf   chromosom_size_lo
        movlw   CHROM_SIZE_HI
        movwf   chromosom_size_hi
        return
;*****************************************************************


;*****************************************************************
;*** Funktion:   insert_fitness
;*** Job:        Einfuegen eines Fitness-Wertes, der zu Chromosom mit
;***             Index wreg gehoert. Die Adresse des einzufuegenden
;***             Fitnesswertes steht in fsr1 (Bank0).
insert_fitness
        ifdef   IND_FIT_RESIDENT_
         call   fitness                 ; fsr0 = Adresse der Fitnesswerte
         movfp  indirekt1,wreg
         movlr  bank1
         movwf  indirekt0               ; Speichern des Low-Wertes
         movlr  bank0
         incf   fsr0, 1
         incf   fsr1, 1
         movfp  indirekt1,wreg
         movlr  bank1
         movwf  indirekt0               ; Speichern des High-Wertes
         movlr  bank0
        else
         call   fitness_ext
         tlwt   0,indirekt1
         incf   fsr1, 1
         tablwt 1,0,indirekt1
        endif
        return
;*****************************************************************


;*****************************************************************
;*** Funktion:   sum_of_fitness
;*** Job:        Berechnet die Summe der Fitnesswerte der aktuellen
;***             Population und schreibt das Ergebnis in das Ausgabefenster.
sum_of_fitness
        ifndef  IND_FIT_RESIDENT_
         movfp  FIT_PTR_LO,tblptrl
         movfp  FIT_PTR_HI,tblptrh
        endif
```

```
        clrf    loop_reg1, 1
        clrf    fitsum1, 1              ; Least significant register
        clrf    fitsum2, 1
        clrf    fitsum3, 1              ; Most significant register

fitsumloop
        ifdef   IND_FIT_RESIDENT_
         movfp  loop_reg1,wreg
         mullw  2
         movfp  prodLo,wreg
         addlw  FIT
         movwf  fsr0                    ; fsr0 = Adresse der 16 Bit Zahl
         movlr  bank1
         movpf  indirekt0,wreg
         movlr  bank0
         movwf  temp5
         incf   fsr0, 1
         movlr  bank1
         movpf  indirekt0,wreg
         movlr  bank0
         movwf  temp6
        else
         tablrd 0,0,dummy
         tlrd   0,temp5
         tablrd 1,1,temp6
        endif

        movlw   TEMP_PTR+4
        movwf   fsr0                    ; fsr0 = Adresse der 16-Bit Zhal FitLo/Hi
        movlw   FITSUM_PTR
        movwf   fsr1                    ; fsr1 = Adresse der 24-Bit Zahl FitSum

        call    add16_24                ; Ergebnis steht in temp1 bis temp4.

        movpf   temp1,fitsum1
        movpf   temp2,fitsum2
        movpf   temp3,fitsum3           ; FitSum = FitSum + Fitness[k]

        incf    loop_reg1, 1
        movlw   POP_SIZE-1
        cpfsgt  loop_reg1               ; skip if loop_reg1 >= POP_SIZE
         goto   fitsumloop

        return
;*****************************************************************


;*****************************************************************
;*** Funktion:   inc_generation
;*** Job:        Inkrementieren des Generation Counters.
inc_generation

        incf    generation_counterLo, 1
        clrf    wreg, 1
        addwfc  generation_counterMi, 1
        clrf    wreg, 1
        addwfc  generation_counterHi, 1
        return
;*****************************************************************


;*****************************************************************
;*** Funktion:   index_sort
;*** Job:        Die Funktion sortiert die Indizes des Indexblocks
;***             aufsteigend nach Fitnesswerten und aktualisiert
;***             gegebenenfalls das beste Individuum. Falls der Indexblock
;***             nicht schon in der Bank1 steht, wird er dorthin kopiert.
;***             Angewandtes Sortierverfahren : Auswahlsort.
index_sort
        ;*** Kopieren des Indexblocks in Data Memory.
        ifndef  IND_RESIDENT_
         movfp  INDEX_PTR_LO,tblptrl
```

```
          movfp   INDEX_PTR_HI,tblptrh
          movlw   POP_SIZE
          movwf   temp1
          movlw   IND
          movwf   fsr0
          movlr   bank1
          tablrd  0, 1, dummy
          call    auto_inc_fsr0
          tlrd    0, indirekt0
          decfsz  temp1, 1
          goto    $+2
          goto    $+4
          tablrd  1, 1, indirekt0
          decfsz  temp1, 1
          goto    $-6
          call    no_change_fsr0
          movlr   bank0
         endif

         movlw    .1
         movwf    loop_reg1
         movlr    bank0

indexsortloop
         movfp    loop_reg1,temp4          ; In temp4 wird Position des
         movfp    loop_reg1,wreg           ; Index des Minimums gespeichert.
         incf     wreg, 1
         movwf    loop_reg2                ; loop_reg2 = loop_reg1 + 1

indexloop
         movlr    bank1

         ;*** Fitness des Chromosoms, dessen Index in IND+(temp4-1) steht
         movfp    temp4,wreg
         addlw    IND-1                    ; wreg  = IND + (min-1)
         movwf    fsr0                     ; fsr0  = wreg
         movpf    indirekt0,wreg           ; wreg  = Index von Chromosom
         ifndef   IND_FIT_RESIDENT_
          movlr   bank0
          call    fitness_ext              ; tblptrl/h = Adresse der gesuchten Fitness
          movlr   bank1
          movlw   IND+POP_SIZE             ; Adresse, an die Fitness geschrieben wird
          movwf   fsr0
          tlrd    0,indirekt0
          incf    fsr0, 1
          tlrd    1,indirekt0
          decf    fsr0, 1
         else
          call    fitness                 ; fsr0  = wreg = Adresse der Fitnesswerte
         endif
         movwf    temp1                    ; temp1 = Adresse von FitLo eines Chrom.
         movwf    temp3                    ; temp3 = temp1, Zwischenspeicher

         ;*** Fitness des Chromosoms, dessen Index in IND+(loop_reg2-1) steht
         movlr    bank0
         movlw    IND-1
         addwf    loop_reg2, 0
         movwf    fsr0
         movlr    bank1
         movpf    indirekt0, wreg          ; wreg  = Index von Chromosom
         ifndef   IND_FIT_RESIDENT_
          movlr   bank0
          call    fitness_ext
          movlr   bank1
          movlw   IND+POP_SIZE+2           ; Fitness des zweiten Chromosoms
          movwf   fsr0
          tlrd    0,indirekt0
          incf    fsr0, 1
          tlrd    1,indirekt0
          decf    fsr0, 1
         else
          call    fitness                 ; fsr0  = Adresse der Fitnesswerte
         endif
```

```
        movwf   temp2                   ; temp2 = Adresse von FitLo eines Chrom.

        call    compare16               ; Vergleicht 16 Bit Zahlen mit Adresse in
                                        ; temp1 und temp2.
        movfp   temp3,wreg              ; wreg = temp3
        subwf   temp1, 0                ; wreg = temp1 - temp3
        movfp   temp4,wreg              ; wreg = altes Minimum
        movlr   bank0
580     btfsc   _zero                   ; skip if alusta[2] = 0
         movfp  loop_reg2,wreg          ; Hier gilt: temp1 <  temp2
        movwf   temp4

        incf    loop_reg2, 1
        movlw   POP_SIZE
        cpfsgt  loop_reg2
         goto   indexloop               ; Schleife indexloop

        ;*** Vertauschen zweier Indizes.
590     movlw   IND-1
        addwf   loop_reg1, 0
        movwf   fsr0
        movfp   temp4,wreg              ; wreg = min
        addlw   IND-1
        movwf   fsr1
        movlr   bank1
        movpf   indirekt0,temp1
        movfp   indirekt1,indirekt0
        movpf   temp1,indirekt1
600     movlr   bank0

        incf    loop_reg1, 1
        movlw   POP_SIZE-1
        cpfsgt  loop_reg1
         goto   indexsortloop           ; Schleife indexsortloop

        ;*** Kopieren des neu angeordneten Indexblocks in das Program Memory.
        ifndef  IND_RESIDENT_
         movlw  IND
610      movwf  fsr0
         movlw  POP_SIZE
         movwf  temp1
         movfp  INDEX_PTR_LO,tblptrl
         movfp  INDEX_PTR_HI,tblptrh
         movlr  bank1
         call   auto_inc_fsr0
         tlwt   0, indirekt0
         decfsz temp1, 1
         goto   $+2
620      goto   $+5
         tablwt 1, 1, indirekt0
         decfsz temp1, 1
         goto   $-6
         goto   $+2
         tablwt 1, 1,temp1
         call   no_change_fsr
         movlr  bank0
        endif

630     ;*** Kopieren des besten Chromosoms in den Best-Chrom-Block.
        call    get_best_index          ; wreg  = Index des besten Individuums
        movwf   temp4                   ; temp4 : Zwischenspeicherung des Index.

        ifdef   IND_FIT_RESIDENT_
         call   fitness                 ; wreg  = FIT_PTR + 2*index - 2
                                        ;       = Adresse von FitLo
        else
         call   fitness_ext
         movlw  IND+POP_SIZE+2
640      movwf  fsr0
         movlr  bank1
         tlrd   0,indirekt0
         incf   fsr0, 1
         tlrd   1,indirekt0
```

```
            movlr   bank0
            movlw   IND+POP_SIZE+2
           endif

           movwf    temp3                   ; temp1/3 = Zwischenspeicherung der
650                                         ; Adresse von FitLo
           movwf    temp1                   ; des besten Chromosoms, fuer
                                            ; compare-Routine gebraucht

           ifndef   IND_FIT_RESIDENT_
            movlw   IND+POP_SIZE
           else
            movlw   FIT+2*POP_SIZE
           endif
           movwf    temp2                   ; temp2 = Adresse v. FitLo des
660                                         ; bisher besten Chrom.
           movwf    fsr0
           movfp    bestfitLo,wreg
           movlr    bank1
           movwf    indirekt0
           incf     fsr0, 1
           movlr    bank0
           movfp    bestfitHi,wreg          ; Kopieren des besten Individuums nach FIT
           movlr    bank1
           movwf    indirekt0
670
           call     compare16
           movlr    bank0

           movfp    temp3,wreg
           subwf    temp1, 0

           btfss    _zero
            return

680 new_best
           movpf    temp4,best_index        ; neuer bester Index

           movfp    temp1,fsr0              ; fsr0 = Adresse von FitLo
                                            ; des neuen besten Chrom.
           movlr    bank1
           movpf    indirekt0,wreg
           movlr    bank0
           movwf    bestfitLo               ; bestfitLo
           incf     fsr0, 1
690        movlr    bank1
           movpf    indirekt0,wreg
           movlr    bank0
           movwf    bestfitHi               ; bestfitHi

           call     get_best_chrom          ; tblptrl/h = Adresse des besten Chromosoms
           movfp    tblptrl, wreg
           movwf    tsp_ptr1_Lo
           movfp    tblptrh, wreg
           movwf    tsp_ptr1_Hi
700        movfp    BEST_CHROM_PTR_LO,wreg
           movwf    tsp_ptr2_Lo
           movfp    BEST_CHROM_PTR_HI,wreg
           movwf    tsp_ptr2_Hi
           call     Data_Copy
           goto     end_index_sort

    end_index_sort
           return                           ; Ende von index_sort
    ;*****************************************************************
710

    ;*****************************************************************
    ;*** Funktion:  update_population
    ;*** Job:       Berechnet die Population der naechsten Generation
    update_population
           call     sum_of_fitness
           call     index_sort
```

```
        call    inc_generation
        return
;*************************************************************
```

```
;*************************************************************
;*** Funktion:   delete_n_last
;*** Job:        Einfuegen der n neu erzeugten Chromosome des Childrenblockes
;***             in die Population. Die schlechtesten n Chromosome der
;***             aktuellen Population werden dabei geloescht.
delete_n_last
        movlw   .1
        movwf   loop_reg4
deleteloop
        movfp   loop_reg4,wreg
        call    get_child
        movfp   tblptrl, wreg
        movwf   tsp_ptr1_Lo             ; Startadresse
        movfp   tblptrh, wreg
        movwf   tsp_ptr1_Hi

        movfp   loop_reg4,wreg
        sublw   POP_SIZE+1
        call    get_chrom
        movfp   tblptrl, wreg
        movwf   tsp_ptr2_Lo             ; Zieladresse
        movfp   tblptrh, wreg
        movwf   tsp_ptr2_Hi

        call    Data_Copy

        ;*** Fitness der Kinder berechnen.
        movfp   loop_reg4, wreg
        sublw   POP_SIZE+1
        ifdef   IND_RESIDENT_
         call   get_index
        endif
        ifndef  IND_RESIDENT_
         call   get_index_ext
        endif                           ; wreg = entsprechender Index
        movwf   new_child_index
        call    compute_tsp_fitness     ; Bestimmen der Fitness
        movlw   TSP_FITNESS_PTR
        movwf   fsr1                    ; fsr1 = Adresse der einzufuegenden Fitness
        movfp   new_child_index, wreg   ; wreg = Index des akt. Chromosomes
        call    insert_fitness          ; Fitness aktualisieren

        incf    loop_reg4, 1
        movlw   NO_OF_CHILDREN
        cpfsgt  loop_reg4
         goto   deleteloop
        return
;*************************************************************
```

D.13 Rekombinations-Operatoren: operator.asm

```
;*************************************************************
;***             Operatoren genetischer Algorithmen        ***
;*** Autor:      Tobias Schubert                           ***
;*** EMail:      schubert@informatik.uni-freiburg.de       ***
;*** Datum:      07.10.1999                                ***
;*** Datei-Name: operator.asm                              ***
;*************************************************************
```

```
;*************************************************************
;*** Funktion:   find4opt
```

```
;*** Job:        Bestimme an zufaelliger Stelle eine optimale
;***             4-Staedte-Teilstrecke.
find4opt
        movfp   tsp_ptr1_Lo, wreg
        movwf   tblptrl
        movfp   tsp_ptr1_Hi, wreg
        movwf   tblptrh

        ;*** Die vier Staedte einlesen.
        tablrd  0, 1, next_city1
        tlrd    0, next_city1
        tlrd    1, next_city2
        tablrd  0, 1, next_city3
        tlrd    0, next_city3
        tlrd    1, next_city4

        ;*** Staedte zwischenspeichern.
        movfp   next_city1, wreg
        movwf   city1
        movfp   next_city2, wreg
        movwf   city2
        movfp   next_city3, wreg
        movwf   city3
        movfp   next_city4, wreg
        movwf   city4

        ;*** Berechne Strecke next_city1, next_city2, next_city3, next_city4.
        call    find4opt_init

        movfp   city1, wreg
        movwf   next_city1
        movfp   city2, wreg
        movwf   next_city2
        call    compute_distance
        movfp   city2, wreg
        movwf   next_city1
        movfp   city3, wreg
        movwf   next_city2
        call    compute_distance
        movfp   city3, wreg
        movwf   next_city1
        movfp   city4, wreg
        movwf   next_city2
        call    compute_distance

        clrf    find4opt_tmp, 1
        movfp   tsp_fit_Lo, wreg
        movwf   find4opt_tmp

        movfp   city1, wreg
        movwf   ncity1
        movfp   city2, wreg
        movwf   ncity2
        movfp   city3, wreg
        movwf   ncity3
        movfp   city4, wreg
        movwf   ncity4

        ;*** Berechne Strecke next_city1, next_city2, next_city4, next_city3.
        call    find4opt_init

        movfp   city1, wreg
        movwf   next_city1
        movfp   city2, wreg
        movwf   next_city2
        call    compute_distance
        movfp   city2, wreg
        movwf   next_city1
        movfp   city4, wreg
        movwf   next_city2
        call    compute_distance
        movfp   city4, wreg
        movwf   next_city1
```

```
        movfp   city3, wreg
        movwf   next_city2
        call    compute_distance

        movfp   tsp_fit_Lo, wreg
        cpfslt  find4opt_tmp
        goto    $+02                    ; skip, if find4opt_tmp<tsp_fit_Lo
        goto    $+0B
        movfp   tsp_fit_Lo, wreg
        movwf   find4opt_tmp
        movfp   city1, wreg
        movwf   ncity1
        movfp   city2, wreg
        movwf   ncity2
        movfp   city4, wreg
        movwf   ncity3
        movfp   city3, wreg
        movwf   ncity4

        ;*** Berechne Strecke next_city1, next_city3, next_city2, next_city4.
        call    find4opt_init

        movfp   city1, wreg
        movwf   next_city1
        movfp   city3, wreg
        movwf   next_city2
        call    compute_distance
        movfp   city3, wreg
        movwf   next_city1
        movfp   city2, wreg
        movwf   next_city2
        call    compute_distance
        movfp   city2, wreg
        movwf   next_city1
        movfp   city4, wreg
        movwf   next_city2
        call    compute_distance

        movfp   tsp_fit_Lo, wreg
        cpfslt  find4opt_tmp
        goto    $+02                    ; skip, if find4opt_tmp<tsp_fit_Lo
        goto    $+0B
        movfp   tsp_fit_Lo, wreg
        movwf   find4opt_tmp
        movfp   city1, wreg
        movwf   ncity1
        movfp   city3, wreg
        movwf   ncity2
        movfp   city2, wreg
        movwf   ncity3
        movfp   city4, wreg
        movwf   ncity4

        ;*** Die vier Staedte in ihrer (neuen) Reihenfolge speichern.
        movfp   tsp_ptr1_Lo, wreg
        movwf   tblptrl
        movfp   tsp_ptr1_Hi, wreg
        movwf   tblptrh

        tlwt    0, ncity1
        tablwt  1, 1, ncity2
        tlwt    0, ncity3
        tablwt  1, 1, ncity4
        return

;*** Hilfsfunktion:
find4opt_init
        clrf    tsp_fit_Lo, 1
        clrf    tsp_fit_Hi, 1
        clrf    dist_remainder, 1
        return
;*****************************************************************
```

```
;*****************************************************************
;*** Funktion:   tsp_mutation
;*** Job:        Am Chromosom mit Startadresse tsp_ptr3_Lo/Hi wird
;***            Mutation durchgefuehrt, d.h. 2 Staedte in ihrer
;***            Position vertauscht. Benoetigt: tsp_ptr1/2/3_Lo/Hi
;***            und tsp_temp1/2/3.
tsp_mutation
        movfp   tsp_ptr3_Lo, wreg
        movwf   tsp_ptr2_Lo
        movwf   tsp_ptr1_Lo
        movfp   tsp_ptr3_Hi, wreg
        movwf   tsp_ptr2_Hi
        movwf   tsp_ptr1_Hi

        ;*** 1. Position/Stadt finden.
        call    find_pos_city
        movfp   cross_posLo, wreg
        movwf   next_city1
        decf    cross_posLo, 1
        movlw   H'08'
        mulwf   cross_posLo             ; Bestimme "Start-Bit" der
        movfp   prodHi, wreg            ; "zufaelligen" Stadt.
        movwf   numerator16Hi
        movfp   prodLo, wreg
        movwf   numerator16Lo
        movlw   H'10'
        movwf   denominator16Lo
        clrf    denominator16Hi, 1
        call    div16

        movfp   remainder16Lo, wreg
        movwf   tsp_temp3

        movfp   quotiont16Lo, wreg
        addwf   tsp_ptr1_Lo, 1          ; Zeiger justieren
        clrf    wreg, 1
        addwfc  tsp_ptr1_Hi, 1          ; evtl. Carry
        movfp   quotiont16Hi, wreg
        addwf   tsp_ptr1_Hi, 1

        movfp   tsp_ptr1_Lo, wreg
        movwf   tblptrl
        movfp   tsp_ptr1_Hi, wreg
        movwf   tblptrh

        clrf    wreg, 1
        cpfseq  remainder16Lo
        goto    $+4
        tablrd  0, 0, tsp_temp1         ; High-Byte ist gewaehlte Pos.
        tlrd    0, tsp_temp1
        goto    $+3
        tablrd  1, 0, tsp_temp1         ; Low-Byte ist gewaehlte Pos.
        tlrd    1, tsp_temp1

        ;*** 2. Position/Stadt finden.
        call    find_pos_city
        movfp   next_city1, wreg        ; Ausschliessen gleicher gewaehlter Staedte
        cpfseq  cross_posLo
        goto    $+2                     ; skip, if next_city1 = cross_posLo
        goto    $-7                     ; skip, if next_city1 <> cross_posLo
        decf    cross_posLo, 1
        movlw   H'08'
        mulwf   cross_posLo             ; Bestimme "Start-Bit" der
        movfp   prodHi, wreg            ; "zufaelligen" Stadt.
        movwf   numerator16Hi
        movfp   prodLo, wreg
        movwf   numerator16Lo
        movlw   H'10'
        movwf   denominator16Lo
        clrf    denominator16Hi, 1
        call    div16
```

```
        movfp   quotiont16Lo, wreg
        addwf   tsp_ptr2_Lo, 1          ; Zeiger justieren
        clrf    wreg, 1
        addwfc  tsp_ptr2_Hi, 1          ; evtl. Carry
        movfp   quotiont16Hi, wreg
        addwf   tsp_ptr2_Hi, 1

        movfp   tsp_ptr2_Lo, wreg
        movwf   tblptrl
240     movfp   tsp_ptr2_Hi, wreg
        movwf   tblptrh

        clrf    wreg, 1
        cpfseq  remainder16Lo
        goto    $+4
        tablrd  0, 0, tsp_temp2         ; High-Byte ist gewaehlte Pos.
        tlrd    0, tsp_temp2
        goto    $+3
        tablrd  1, 0, tsp_temp2         ; Low-Byte ist gewaehlte Pos.
250     tlrd    1, tsp_temp2

        ;*** Position der Staedte austauschen.
        movfp   tsp_ptr1_Lo, wreg
        movwf   tblptrl
        movfp   tsp_ptr1_Hi, wreg
        movwf   tblptrh
        clrf    wreg, 1
        cpfseq  tsp_temp3
        goto    $+6
260     tablrd  1, 0, wreg
        tlrd    1, wreg
        tlwt    1, wreg
        tablwt  0, 0, tsp_temp2
        goto    $+5
        tablrd  0, 0, wreg
        tlrd    0, wreg
        tlwt    0, wreg
        tablwt  1, 0, tsp_temp2

270     movfp   tsp_ptr2_Lo, wreg
        movwf   tblptrl
        movfp   tsp_ptr2_Hi, wreg
        movwf   tblptrh
        clrf    wreg, 1
        cpfseq  remainder16Lo
        goto    $+6
        tablrd  1, 0, wreg
        tlrd    1, wreg
        tlwt    1, wreg
280     tablwt  0, 0, tsp_temp1
        return
        tablrd  0, 0, wreg
        tlrd    0, wreg
        tlwt    0, wreg
        tablwt  1, 0, tsp_temp1
        return
;*************************************************************


290 ;*************************************************************
;*** Funktion:   greedy_crossover
;*** Job:        Es wird ausgehend von einer zufaelligen Startstadt
;***             immer diejenige naechste Stadt eines Elternteiles
;***             gewaehlt, die "naeher" liegt.
greedy_crossover
        ;*** Die Elternteile sind durch Indizes chrom_index1/2 bestimmt.
        movfp   children_counter, wreg
        incf    wreg, 1
        call    get_child               ; tblptrl/h = Adresse des Kindes
300     movfp   tblptrh, wreg
        movwf   tsp_ptr1_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr1_Lo
```

```
        ;*** Zufaellige Startstadt waehlen.
        call    find_pos_city
        movfp   cross_posLo, wreg
        movwf   city1                   ; city1 = Startstadt

        ;*** Nachfolger von city1 in Route 1 finden (in chrom1Lo).
        call    get_next_city_in_route1

        ;*** Nachfolger von city1 in Route 2 finden (in chrom2Lo).
        call    get_next_city_in_route2

        ;*** Bestimme zu waehlenden Nachfolger (in chrom1Lo).
        call    get_next_city

        ;*** Teil-Route speichern.
        call    write_new_part

        movlw   H'02'
        movwf   greedy_temp
Greedy_Crossover_Loop
        ;*** Chrom1Lo wird zum neuen Ausgangspunkt.
        movfp   chrom1Lo, wreg
        movwf   city1

        ;*** Nachfolger von city1 in Route 1 finden (in chrom1Lo).
        call    get_next_city_in_route1

        ;*** Nachfolger von city1 in Route 2 finden (in chrom2Lo).
        call    get_next_city_in_route2

        ;*** Bestimme zu waehlenden Nachfolger (in chrom1Lo).
        call    get_next_city

        ;*** Nur gueltige Loesungen zulassen.
        call    correct_solutions

        ;*** Chrom1Lo wird zum neuen Ausgangspunkt.
        movfp   chrom1Lo, wreg
        movwf   city1

        ;*** Nachfolger von city1 in Route 1 finden (in chrom1Lo).
        call    get_next_city_in_route1

        ;*** Nachfolger von city1 in Route 2 finden (in chrom2Lo).
        call    get_next_city_in_route2

        ;*** Bestimme zu waehlenden Nachfolger (in chrom1Lo).
        call    get_next_city

        ;*** Nur gueltige Loesungen zulassen.
        call    correct_solutions

        ;*** Teil-Route speichern.
        call    write_new_part

        ;*** Weitere Iterationen?
        incf    greedy_temp, 1
        incf    greedy_temp, 1
        movlw   NO_OF_CITIES
        cpfseq  greedy_temp
        goto    Greedy_Crossover_Loop
        return

;*** Hilfsfunktion: Nur gueltige Loesungen erlauben.
correct_solutions
        movfp   children_counter, wreg
        incf    wreg, 1
        call    get_child               ; tblptrl/h = Adresse des Kindes

correct_solutions_loop
        tablrd  0, 1, chrom2Lo
        tlrd    0, chrom2Lo
```

```
        tlrd    1, chrom2Hi

        movfp   chrom1Lo, wreg
        cpfseq  chrom2Lo
        goto    $+2
        goto    create_new_edge
        movfp   chrom1Lo, wreg
        cpfseq  chrom2Hi
        goto    $+2
        goto    create_new_edge

        movfp   tblptrh, wreg
        cpfseq  tsp_ptr1_Hi
        goto    correct_solutions_loop
        movfp   tblptrl, wreg
        cpfseq  tsp_ptr1_Lo
        goto    correct_solutions_loop
        return

create_new_edge
        ;*** Zufaellige Stadt waehlen (in cross_posLo).
        call    find_pos_city

        ;*** Ist cross_posLo = city1 ?
        movfp   city1, wreg
        cpfseq  cross_posLo
        goto    $+2                     ; Skip if cross_posLo = city1
        goto    create_new_edge

        movfp   children_counter, wreg
        incf    wreg, 1
        call    get_child               ; tblptrl/h = Adresse des Kindes
new_edge_loop
        tablrd  0, 1, chrom2Lo
        tlrd    0, chrom2Lo
        tlrd    1, chrom2Hi

        movfp   cross_posLo, wreg
        cpfseq  chrom2Lo
        goto    $+2
        goto    create_new_edge
        movfp   cross_posLo, wreg
        cpfseq  chrom2Hi
        goto    $+2
        goto    create_new_edge

        movfp   tblptrh, wreg
        cpfseq  tsp_ptr1_Hi
        goto    new_edge_loop
        movfp   tblptrl, wreg
        cpfseq  tsp_ptr1_Lo
        goto    new_edge_loop

        movfp   cross_posLo, wreg
        movwf   chrom1Lo
        return

;*** Hilfsfunktion: Neue Teilroute city1 -> chrom1Lo speichern.
write_new_part
        movfp   tsp_ptr1_Hi, wreg       ; Im neuen Kind speichern
        movwf   tblptrh
        movfp   tsp_ptr1_Lo, wreg
        movwf   tblptrl
        tlwt    0, city1
        tablwt  1, 1, chrom1Lo
        movfp   tblptrh, wreg
        movwf   tsp_ptr1_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr1_Lo
        return

;*** Hilfsfunktion: Bessere naechste Stadt aus chrom1Lo und chrom2Lo bestimmen.
get_next_city
```

```
450         clrf    tsp_fit_Lo, 1
            clrf    tsp_fit_Hi, 1
            clrf    dist_remainder, 1
            movfp   city1, wreg
            movwf   next_city1
            movfp   chrom1Lo, wreg
            movwf   next_city2
            call    compute_distance        ; Entfernung city1 <-> chrom1Lo

            movfp   tsp_fit_Lo, wreg
460         movwf   greedyHi
            movfp   dist_remainder, wreg
            movwf   greedyLo

            clrf    tsp_fit_Lo, 1
            clrf    tsp_fit_Hi, 1
            clrf    dist_remainder, 1
            movfp   city1, wreg
            movwf   next_city1
            movfp   chrom2Lo, wreg
470         movwf   next_city2
            call    compute_distance        ; Entfernung city1 <-> chrom2Lo

            movfp   tsp_fit_Lo, wreg        ; tsp_fit_Lo gehoert zu chrom2Lo
            cpfsgt  greedyHi                ; greedyHi gehoert zu chrom1Lo
            goto    $+2                     ; Skip, if greedyHi > tsp_fit_Lo
            goto    $+8                     ; greedyHi > tsp_fit_Lo

            movfp   tsp_fit_Lo, wreg
            cpfseq  greedyHi
480         return                          ; Skip, if greedyHi = tsp_fit_Lo

            movfp   dist_remainder, wreg
            cpfslt  greedyLo
            goto    $+2                     ; Skip, if greedyLo < dist_remainder
            return

            movfp   chrom2Lo, wreg
            movwf   chrom1Lo
            return
490
    ;*** Hilfsfunktion: Nachfolger von city1 in Route 1 finden.
    get_next_city_in_route1
            clrf    tsp_temp6, 1
            movfp   chrom_index1, wreg
            call    get_indiv

            tablrd  0, 0, start_city
            tlrd    0, start_city

500         tablrd  0, 1, chrom1Lo
            tlrd    0, chrom1Lo
            tlrd    1, chrom1Hi

            incf    tsp_temp6, 1
            incf    tsp_temp6, 1

            movfp   city1, wreg
            cpfseq  chrom1Lo
            goto    $+2                     ; skip, if chrom1Lo = city1
510         goto    $+0D
            movfp   city1, wreg
            cpfseq  chrom1Hi
            goto    $-0B                    ; skip, if chrom1Hi = city1

            movlw   NO_OF_CITIES
            cpfseq  tsp_temp6
            goto    $+4
            movfp   start_city, wreg
            movwf   chrom1Lo
520         return

            tablrd  0, 1, chrom1Lo
```

```
        tlrd    0, chrom1Lo
        return

        movfp   chrom1Hi, wreg
        movwf   chrom1Lo
        return

;*** Hilfsfunktion: Nachfolger von city1 in Route 2 finden.
get_next_city_in_route2
        clrf    tsp_temp6, 1
        movfp   chrom_index2, wreg
        call    get_indiv

        tablrd  0, 0, start_city
        tlrd    0, start_city

        tablrd  0, 1, chrom2Lo
        tlrd    0, chrom2Lo
        tlrd    1, chrom2Hi

        incf    tsp_temp6, 1
        incf    tsp_temp6, 1

        movfp   city1, wreg
        cpfseq  chrom2Lo
        goto    $+2                     ; skip, if chrom2Lo = city1
        goto    $+0D
        movfp   city1, wreg
        cpfseq  chrom2Hi
        goto    $-0B                    ; skip, if chrom2Hi = city1

        movlw   NO_OF_CITIES
        cpfseq  tsp_temp6
        goto    $+4
        movfp   start_city, wreg
        movwf   chrom2Lo
        return

        tablrd  0, 1, chrom2Lo
        tlrd    0, chrom2Lo
        return

        movfp   chrom2Hi, wreg
        movwf   chrom2Lo
        return
;*****************************************************************


;*****************************************************************
;*** Funktion:   tsp_crossover
;*** Job:        Es wird One-Point-Crossover an zwei Chromosomen,
;***             deren Indizes in chrom_index1/2 stehen, durchgefuehrt.
tsp_crossover
        ;*** Kopieren von Eltern-Chromosom 1 an Child[children_counter+1].
        movfp   chrom_index1,wreg
        call    get_indiv               ; tblptrl/h = Adresse von Eltern-Chromosom 1
        movfp   tblptrh, wreg
        movwf   tsp_ptr1_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr1_Lo
        movfp   children_counter, wreg
        incf    wreg, 1
        call    get_child               ; tblptrl/h = Adresse des Kindes
        movfp   tblptrh, wreg
        movwf   tsp_ptr2_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr2_Lo
        call    Data_Copy               ; Chromosom kopieren

        ;*** Kopieren von Eltern-Chromosom 2 an Child[children_counter+2].
        movfp   chrom_index2,wreg
        call    get_indiv               ; tblptrl/h = Adresse von Eltern-Chromosom 2
        movfp   tblptrh, wreg
```

```
        movwf   tsp_ptr1_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr1_Lo
        movfp   children_counter, wreg
600     incf    wreg, 1
        incf    wreg, 1
        call    get_child               ; tblptrl/h = Adresse des Kindes
        movfp   tblptrh, wreg
        movwf   tsp_ptr2_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr2_Lo
        call    Data_Copy               ; Chromosom kopieren

        ;*** Kreuzungspunkt finden: Vor(!) Stadt "cross_posLo".
610     call    find_pos_city
        decf    cross_posLo, 1
        movlw   H'08'
        mulwf   cross_posLo             ; Bestimme "Start-Bit" der
        movfp   prodHi, wreg            ; "zufaelligen" Stadt.
        movwf   numerator16Hi
        movfp   prodLo, wreg
        movwf   numerator16Lo
        movlw   H'10'
        movwf   denominator16Lo
620     clrf    denominator16Hi, 1
        call    div16

        ;*** tsp_ptr1_Lo/Hi zeigt auf entspr. Kind1-Stelle.
        movfp   children_counter, wreg
        incf    wreg, 1
        call    get_child               ; tblptrl/h = Adresse des Kindes
        movfp   tblptrh, wreg
        movwf   tsp_ptr1_Hi
        movfp   tblptrl, wreg
630     movwf   tsp_ptr1_Lo
        movfp   quotiont16Lo, wreg
        addwf   tsp_ptr1_Lo, 1          ; Zeiger justieren
        clrf    wreg, 1
        addwfc  tsp_ptr1_Hi, 1          ; evtl. Carry
        movfp   quotiont16Hi, wreg
        addwf   tsp_ptr1_Hi, 1

        ;*** tsp_ptr2_Lo/Hi zeigt auf entspr. Kind2-Stelle.
        movfp   children_counter, wreg
640     incf    wreg, 1
        incf    wreg, 1
        call    get_child               ; tblptrl/h = Adresse des Kindes
        movfp   tblptrh, wreg
        movwf   tsp_ptr2_Hi
        movwf   tsp_ptr3_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr2_Lo
        movwf   tsp_ptr3_Lo
        movfp   quotiont16Lo, wreg
650     addwf   tsp_ptr2_Lo, 1          ; Zeiger justieren
        clrf    wreg, 1
        addwfc  tsp_ptr2_Hi, 1          ; evtl. Carry
        movfp   quotiont16Hi, wreg
        addwf   tsp_ptr2_Hi, 1

        ;*** Daten austauschen.
        call    Read_Registers

        clrf    wreg, 1
660     cpfseq  remainder16Lo
        goto    $+0F                    ; Nur High-Byte austauschen

        ;*** Beide Teile austauschen.
        call    Change_both_parts
        movfp   PMX_PTR_LO, wreg
        movwf   tblptrl
        movfp   PMX_PTR_HI, wreg
        movwf   tblptrh
```

```
            tlwt    0, chrom1Lo
670         tablwt  1, 1, chrom2Lo
            tlwt    0, chrom1Hi
            tablwt  1, 1, chrom2Hi
            movfp   tblptrl, wreg
            movwf   tsp_ptr4_Lo
            movfp   tblptrh, wreg
            movwf   tsp_ptr4_Hi
            goto    $+12

            ;*** Nur High-Byte austauschen.
680         movfp   PMX_PTR_LO, wreg
            movwf   tblptrl
            movfp   PMX_PTR_HI, wreg
            movwf   tblptrh
            tlwt    0, chrom1Hi
            tablwt  1, 1, chrom2Hi
            movfp   tblptrl, wreg
            movwf   tsp_ptr4_Lo
            movfp   tblptrh, wreg
            movwf   tsp_ptr4_Hi
690         movfp   chrom1Lo, wreg
            movwf   tsp_temp4
            movfp   chrom2Lo, wreg
            movwf   chrom1Lo
            movfp   tsp_temp4, wreg
            movwf   chrom2Lo
            call    Change_both_parts

            ;*** Ende von Chromosom schon erreicht?
            movfp   tsp_ptr3_Lo, wreg
700         cpfseq  tsp_ptr1_Lo
            goto    Change_Next_Register    ; Skip, if tsp_ptr1_Lo = tsp_ptr3_Lo
            movfp   tsp_ptr3_Hi, wreg
            cpfseq  tsp_ptr1_Hi
            goto    Change_Next_Register    ; Skip, if tsp_ptr1_Hi = tsp_ptr3_Hi
            goto    PMX_Check               ; Gewollt ist PMX-Crossover

            ;*** Weitere Register austauschen.
    Change_Next_Register
            call    Read_Registers
710         movfp   tsp_ptr4_Lo, wreg
            movwf   tblptrl
            movfp   tsp_ptr4_Hi, wreg
            movwf   tblptrh
            tlwt    0, chrom1Lo
            tablwt  1, 1, chrom2Lo
            tlwt    0, chrom1Hi
            tablwt  1, 1, chrom2Hi
            movfp   tblptrl, wreg
            movwf   tsp_ptr4_Lo
720         movfp   tblptrh, wreg
            movwf   tsp_ptr4_Hi
            call    Change_both_parts
            movfp   tsp_ptr3_Lo, wreg       ; Ende des Chromosomes?
            cpfseq  tsp_ptr1_Lo
            goto    Change_Next_Register    ; Skip, if tsp_ptr1_Lo = tsp_ptr3_Lo
            movfp   tsp_ptr3_Hi, wreg
            cpfseq  tsp_ptr1_Hi
            goto    Change_Next_Register    ; Skip, if tsp_ptr1_Hi = tsp_ptr3_Hi

730         ;*** Nur gueltige Loesungen erzeugen, d.h. solche, in denen
            ;*** jede Stadt nur einmal auftritt.
    PMX_Check
            movfp   children_counter, wreg
            incf    wreg, 1
            call    get_child               ; tblptrl/h = Adresse des 1. Kindes
            movfp   tblptrh, wreg
            movwf   tsp_ptr1_Hi
            movfp   tblptrl, wreg
            movwf   tsp_ptr1_Lo
740         call    PMX_Corrections1        ; PMX-Bedingung fuer Kind1 herstellen
```

```
        movfp   children_counter , wreg
        incf    wreg, 1
        incf    wreg, 1
        call    get_child               ; tblptrl/h = Adresse des 2. Kindes
        movfp   tblptrh , wreg
        movwf   tsp_ptr1_Hi
        movfp   tblptrl , wreg
        movwf   tsp_ptr1_Lo
750     call    PMX_Corrections2        ; PMX-Bedingung fuer Kind2 herstellen
        return

;*** Hilfsroutine: Austauschen beider Speicherzellen-Register.
Change_both_parts
        movfp   tsp_ptr1_Lo , wreg
        movwf   tblptrl
        movfp   tsp_ptr1_Hi , wreg
        movwf   tblptrh
        tlwt    0, chrom2Lo             ; Kind1 erhaelt Kind2-Daten
760     tablwt  1, 1, chrom2Hi
        movfp   tblptrh , wreg
        movwf   tsp_ptr1_Hi
        movfp   tblptrl , wreg
        movwf   tsp_ptr1_Lo

        movfp   tsp_ptr2_Lo , wreg
        movwf   tblptrl
        movfp   tsp_ptr2_Hi , wreg
        movwf   tblptrh
770     tlwt    0, chrom1Lo             ; Kind2 erhaelt Kind1-Daten
        tablwt  1, 1, chrom1Hi
        movfp   tblptrh , wreg
        movwf   tsp_ptr2_Hi
        movfp   tblptrl , wreg
        movwf   tsp_ptr2_Lo
        return

;*** Hilfsroutine: 2 Speicherzellen der beiden Kinder auslesen
Read_Registers
780     movfp   tsp_ptr1_Lo , wreg
        movwf   tblptrl
        movfp   tsp_ptr1_Hi , wreg
        movwf   tblptrh
        tablrd  0, 0, chrom1Lo
        tlrd    0, chrom1Lo
        tlrd    1, chrom1Hi

        movfp   tsp_ptr2_Lo , wreg
        movwf   tblptrl
790     movfp   tsp_ptr2_Hi , wreg
        movwf   tblptrh
        tablrd  0, 0, chrom2Lo
        tlrd    0, chrom2Lo
        tlrd    1, chrom2Hi
        return

;*** Hilfsroutine: PMX-Korrekturen an Kind1 ausfuehren
PMX_Corrections1
        clrf    tsp_temp4, 1
800 PMX_LOOP_CHILD1
        movfp   tsp_ptr1_Lo , wreg
        movwf   tblptrl
        movfp   tsp_ptr1_Hi , wreg
        movwf   tblptrh
        tablrd  0, 0, chrom1Lo
        tlrd    0, chrom1Lo
        tlrd    1, chrom1Hi

        ;*** Testen, ob chrom1Lo oder chrom2Lo in den getauschten
810     ;*** Datensaetzen bereits vorkommen.
        movfp   tsp_temp4, wreg
        cpfseq  cross_posLo
        goto    $+2
        return
```

```
            movfp   PMX_PTR_HI, wreg
            movwf   tsp_ptr2_Hi
            movfp   PMX_PTR_LO, wreg
            movwf   tsp_ptr2_Lo
820
    test_chrom1Lo
            movfp   tsp_ptr4_Lo, wreg
            cpfseq  tsp_ptr2_Lo
            goto    $+5
            movfp   tsp_ptr4_Hi, wreg
            cpfseq  tsp_ptr2_Hi
            goto    $+2
            goto    test_chrom1Hi_start

830         movfp   tsp_ptr2_Lo, wreg
            movwf   tblptrl
            movfp   tsp_ptr2_Hi, wreg
            movwf   tblptrh
            tablrd  0, 1, tsp_temp1
            tlrd    0, tsp_temp1
            tlrd    1, tsp_temp2
            movfp   tblptrl, wreg
            movwf   tsp_ptr2_Lo
            movfp   tblptrh, wreg
840         movwf   tsp_ptr2_Hi

            movfp   tsp_temp2, wreg
            cpfseq  chrom1Lo
            goto    test_chrom1Lo

            movfp   tsp_temp1, wreg
            movwf   chrom1Lo
            movfp   tsp_ptr1_Lo, wreg
            movwf   tblptrl
850         movfp   tsp_ptr1_Hi, wreg
            movwf   tblptrh
            tlwt    0, chrom1Lo
            tablwt  1, 0, chrom1Hi
            movfp   PMX_PTR_HI, wreg
            movwf   tsp_ptr2_Hi
            movfp   PMX_PTR_LO, wreg
            movwf   tsp_ptr2_Lo
            goto    test_chrom1Lo

860 test_chrom1Hi_start
            incf    tsp_temp4, 1
            movfp   tsp_temp4, wreg
            cpfseq  cross_posLo
            goto    $+2
            return

            movfp   PMX_PTR_HI, wreg
            movwf   tsp_ptr2_Hi
            movfp   PMX_PTR_LO, wreg
870         movwf   tsp_ptr2_Lo

    test_chrom1Hi
            movfp   tsp_ptr4_Lo, wreg
            cpfseq  tsp_ptr2_Lo
            goto    $+5
            movfp   tsp_ptr4_Hi, wreg
            cpfseq  tsp_ptr2_Hi
            goto    $+2
            goto    More_Tests
880
            movfp   tsp_ptr2_Lo, wreg
            movwf   tblptrl
            movfp   tsp_ptr2_Hi, wreg
            movwf   tblptrh
            tablrd  0, 1, tsp_temp1
            tlrd    0, tsp_temp1
            tlrd    1, tsp_temp2
```

```
            movfp   tblptrl , wreg
            movwf   tsp_ptr2_Lo
890         movfp   tblptrh , wreg
            movwf   tsp_ptr2_Hi

            movfp   tsp_temp2 , wreg
            cpfseq  chrom1Hi
            goto    test_chrom1Hi

            movfp   tsp_temp1 , wreg
            movwf   chrom1Hi
            movfp   tsp_ptr1_Lo , wreg
900         movwf   tblptrl
            movfp   tsp_ptr1_Hi , wreg
            movwf   tblptrh
            tlwt    1, chrom1Hi
            tablwt  0, 0, chrom1Lo
            movfp   PMX_PTR_HI , wreg
            movwf   tsp_ptr2_Hi
            movfp   PMX_PTR_LO , wreg
            movwf   tsp_ptr2_Lo
            goto    test_chrom1Hi
910
    More_Tests
            incf    tsp_ptr1_Lo , 1
            clrf    wreg, 1
            addwfc  tsp_ptr1_Hi , 1
            incf    tsp_temp4 , 1
            goto    PMX_LOOP_CHILD1

    ;*** Hilfsroutine : PMX-Korrekturen an Kind2 ausfuehren
    PMX_Corrections2
920         clrf    tsp_temp4 , 1
    PMX_LOOP_CHILD12
            movfp   tsp_ptr1_Lo , wreg
            movwf   tblptrl
            movfp   tsp_ptr1_Hi , wreg
            movwf   tblptrh
            tablrd  0, 0, chrom1Lo
            tlrd    0, chrom1Lo
            tlrd    1, chrom1Hi

930         ;*** Testen , ob chrom1Lo oder chrom2Lo in den getauschten
            ;*** Datensaetzen bereits vorkommen.
            movfp   tsp_temp4 , wreg
            cpfseq  cross_posLo
            goto    $+2
            return

            movfp   PMX_PTR_HI , wreg
            movwf   tsp_ptr2_Hi
            movfp   PMX_PTR_LO , wreg
940         movwf   tsp_ptr2_Lo

    test_chrom1Lo2
            movfp   tsp_ptr4_Lo , wreg
            cpfseq  tsp_ptr2_Lo
            goto    $+5
            movfp   tsp_ptr4_Hi , wreg
            cpfseq  tsp_ptr2_Hi
            goto    $+2
            goto    test_chrom1Hi_start2
950
            movfp   tsp_ptr2_Lo , wreg
            movwf   tblptrl
            movfp   tsp_ptr2_Hi , wreg
            movwf   tblptrh
            tablrd  0, 1, tsp_temp1
            tlrd    0, tsp_temp1
            tlrd    1, tsp_temp2
            movfp   tblptrl , wreg
            movwf   tsp_ptr2_Lo
960         movfp   tblptrh , wreg
```

```
            movwf   tsp_ptr2_Hi

            movfp   tsp_temp1, wreg
            cpfseq  chrom1Lo
            goto    test_chrom1Lo2

            movfp   tsp_temp2, wreg
            movwf   chrom1Lo
            movfp   tsp_ptr1_Lo, wreg
 970        movwf   tblptrl
            movfp   tsp_ptr1_Hi, wreg
            movwf   tblptrh
            tlwt    0, chrom1Lo
            tablwt  1, 0, chrom1Hi
            movfp   PMX_PTR_HI, wreg
            movwf   tsp_ptr2_Hi
            movfp   PMX_PTR_LO, wreg
            movwf   tsp_ptr2_Lo
            goto    test_chrom1Lo2
 980
    test_chrom1Hi_start2
            incf    tsp_temp4, 1
            movfp   tsp_temp4, wreg
            cpfseq  cross_posLo
            goto    $+2
            return

            movfp   PMX_PTR_HI, wreg
            movwf   tsp_ptr2_Hi
 990        movfp   PMX_PTR_LO, wreg
            movwf   tsp_ptr2_Lo

    test_chrom1Hi2
            movfp   tsp_ptr4_Lo, wreg
            cpfseq  tsp_ptr2_Lo
            goto    $+5
            movfp   tsp_ptr4_Hi, wreg
            cpfseq  tsp_ptr2_Hi
            goto    $+2
1000        goto    More_Tests2

            movfp   tsp_ptr2_Lo, wreg
            movwf   tblptrl
            movfp   tsp_ptr2_Hi, wreg
            movwf   tblptrh
            tablrd  0, 1, tsp_temp1
            tlrd    0, tsp_temp1
            tlrd    1, tsp_temp2
            movfp   tblptrl, wreg
1010        movwf   tsp_ptr2_Lo
            movfp   tblptrh, wreg
            movwf   tsp_ptr2_Hi

            movfp   tsp_temp1, wreg
            cpfseq  chrom1Hi
            goto    test_chrom1Hi2

            movfp   tsp_temp2, wreg
            movwf   chrom1Hi
1020        movfp   tsp_ptr1_Lo, wreg
            movwf   tblptrl
            movfp   tsp_ptr1_Hi, wreg
            movwf   tblptrh
            tlwt    1, chrom1Hi
            tablwt  0, 0, chrom1Lo
            movfp   PMX_PTR_HI, wreg
            movwf   tsp_ptr2_Hi
            movfp   PMX_PTR_LO, wreg
            movwf   tsp_ptr2_Lo
1030        goto    test_chrom1Hi2

    More_Tests2
            incf    tsp_ptr1_Lo, 1
```

```
        clrf    wreg, 1
        addwfc  tsp_ptr1_Hi, 1
        incf    tsp_temp4, 1
        goto    PMX_LOOP_CHILD12
;*************************************************************

1040
;*************************************************************
;*** Funktion:   Roulette_Wheel_Selection
;*** Job:        Fuehrt die ausgewaehlte Roulette-Wheel-Selection aus.
roulette_wheel_selection

        ;*** Berechnung von 24-Bit Zufallszahl MOD Fitsum.
        movlw   TEMP_PTR+3
        movwf   temp1                   ; temp1 = Adresse, an die Zufallszahl
                                        ;         geschrieben wird
1050    call    Random16
        movfp   RandLo,numerator24Lo
        movfp   RandHi,numerator24Mi
        call    Random16
        movfp   RandHi,numerator24Hi    ; numerator24Lo/Mi/Hi =
                                        ;       24-Bit Zufallszahl

        movfp   fitsum1,denominator24Lo
        movfp   fitsum2,wreg
        movwf   denominator24Mi
1060    movfp   fitsum3,wreg
        movwf   denominator24Hi         ; denominator24Lo/Mi/Hi = Fitsum1/2/3

        call    div24                   ; temp10 bis temp12 = Zahl MOD Teiler
                                        ;                   = Zufallszahl z
        clrf    cum_sumLo, 1
        clrf    cum_sumMi, 1
        clrf    cum_sumHi, 1            ; cum_sumLo/Mi/Hi =
                                        ; Speicherung der Zwischensumme

1070    ifdef   IND_FIT_RESIDENT_
         movlw  FIT
         movwf  fsr0
         movwf  temp13                  ; temp13 = Speicherung der Adresse
        else                            ;               der aktuellen Fitness
         movfp  FIT_PTR_LO,tblptrl
         movfp  FIT_PTR_HI,tblptrh
        endif

        movlw   POP_SIZE
1080    movwf   temp22                  ; temp22 = POP_SIZE = Schleifen-Zaehler
        clrf    temp21, 1               ; temp21 = Position im Indexblock

loop_roulette_max
        movlw   memptr1
        movwf   fsr1

        ifdef   IND_FIT_RESIDENT_
         movlr  bank1
         movfp  indirekt0,wreg
1090     movlr  bank0
         movwf  indirekt1
         incf   fsr0, 1
         incf   fsr1, 1
         movlr  bank1
         movfp  indirekt0,wreg
         movlr  bank0
         movwf  indirekt1
        else
         tablrd 0,0,dummy
1100     tlrd   0,indirekt1
         incf   fsr1, 1
         tablrd 1,1,indirekt1
        endif

        movlw   memptr1
        movwf   fsr0
```

```
            movlw   TEMP_PTR+4
            movwf   fsr1                    ; fsr1 = Adresse der Zwischensumme
1110                                        ;      = temp5 bis temp7

            call    add16_24                ; temp1 bis temp4 =
                                            ;       Fitness[i]+Zwischensumme

            movpf   temp1,cum_sumLo
            movpf   temp2,cum_sumMi
            movpf   temp3,cum_sumHi         ; Zwischensumme+=Fitness[i],
                                            ; Zwischensumme aktualisiert

1120        ;*** Pruefen, ob Zwischensumme >= Zufallszahl z.
            movlw   TEMP_PTR+9
            movwf   temp1                   ; temp1 = Adresse der Zufallszahl
                                            ; z = temp10 bis temp12
            movwf   temp3                   ; Zwischenspeicherung von temp1
            movlw   TEMP_PTR+4
            movwf   temp2                   ; temp2 = Adresse der Zwischensumme

            call    compare24

1130        movfp   temp3,wreg
            subwf   temp1, 0                ; wreg = temp1 - temp3
            btfss   _zero                   ; skip if wreg = 0
             call   set_cnt                 ; Hier gilt: temp1<>temp3
                                            ;       -->Zwischensumme > z

            incf    temp21, 1               ; Inkrementieren des Indize-Speichers

            ifdef   IND_FIT_RESIDENT_
             incf   temp13, 1
1140         incf   temp13, 1
             movfp  temp13,fsr0             ; fsr0 = Adresse der Fitness
                                            ; des naechsten Chromosoms
            endif

            decfsz  temp22, 1
             goto   loop_roulette_max

            ;*** temp1 = Index des ausgewaehlten Chromosoms.
            movfp   temp21,temp1
1150        return
    ;*************************************************************


    ;*************************************************************
    ;*** Funktion:  find_two_chromosoms
    ;*** Job:       Liefert zwei Indizes chrom_index1/2 fuer "tsp_crossover",
    ;***            die mittels Roulette_Wheel_Selection ermittelt werden.
    find_two_chromosoms
            call    roulette_wheel_selection
1160        movpf   temp1,chrom_index1
    second_index
            call    roulette_wheel_selection
            movfp   temp1,wreg
            cpfseq  chrom_index1
             goto   second_index_ok
            goto    second_index

    second_index_ok
            movpf   temp1,chrom_index2
1170        return
    ;*************************************************************


    ;*************************************************************
    ;*** Funktion:  find_pos_city
    ;*** Job:       Bestimmen einer Zufallszahl zwischen 1 und NO_OF_CITIES.
    ;***            In cross_posLo wird die Zahl gespeichert.
    find_pos_city
            ;*** Erzeugen einer Zufallszahl z mit Groesse 0 <= z < NO_OF_CITIES+1
```

```
1180          call    Random16
              movfp   RandLo, wreg
              movwf   numerator
              movlw   NO_OF_CITIES+1
              movwf   denominator
              call    div8
              clrf    wreg, 1
              cpfsgt  remainder
              incf    remainder, 1            ; remainder=0 ausschliessen
              movfp   remainder,wreg
1190          movwf   cross_posLo
              return
    ;*****************************************************************


    ;*****************************************************************
    ;*** Funktion:   Random100
    ;*** Job:        Bestimmen einer Zufallszahl zwischen 0 und 100.
    ;***             In cross_posLo wird die Zahl gespeichert.
    Random100
1200          ;*** Erzeugen einer Zufallszahl z mit Groesse 0 <= z < 101
              call    Random16
              movfp   RandLo, wreg
              movwf   numerator
              movlw   H'65'
              movwf   denominator
              call    div8
              movfp   remainder,wreg
              movwf   cross_posLo
              return
1210 ;*****************************************************************
```

D.14 Funktionen zum Datenaustausch: `kom.asm`

```
    ;*****************************************************************
    ;***             Routinen zur Kommunikation mit anderen PICs ***
    ;*** Autor:      Tobias Schubert                             ***
    ;*** EMail:      schubert@informatik.uni-freiburg.de         ***
    ;*** Datum:      25.09.1999                                  ***
    ;*** Datei-Name: kom.asm                                     ***
    ;*****************************************************************


  10 ;*****************************************************************
    ;*** Funktion:   NEW_RESULT
    ;*** Job:        Datenaustausch mit anderen PICs.
    NEW_RESULT
              CLRF    data_change_periode, 1  ; "Datentausch-Zaehler"

              CLRF    intsta, 1
              BSF     intsta, 0               ; Bit 0  von PORTA als Interrupt
              BCF     cpusta, 4               ; Interrupts zulassen

  20          CALL    out_size                ; Parameter in den Ausgabeblock
              CLRF    temp1, 1                ; IRQ-Signal: $01 = neue Motorola-Daten
              CLRF    temp2, 1                ; temp2 = Daten vom Motorola

              MOVFP   BEST_CHROM_PTR_LO, wreg
              MOVWF   temp3
              MOVFP   BEST_CHROM_PTR_HI, wreg
              MOVWF   temp4

              CALL    SET_TABLE_POINTER
  30          MOVLW   0xFE                    ; $FE = Datenaustausch
              TABLWT  1, 0, wreg

    BEST_CHROMO
```

```
        CALL    MOTOROLA_WAITING

        MOVFP   temp3, wreg
        MOVWF   tblptrl
        MOVFP   temp4, wreg
        MOVWF   tblptrh
40
        TABLRD  0, 0, temp1
        TLRD    0, temp1
        CALL    SET_TABLE_POINTER
        MOVFP   temp1, wreg
        TABLWT  1, 0, wreg
        CLRF    temp1, 1
        CALL    MOTOROLA_WAITING

        MOVFP   temp3, wreg
50      MOVWF   tblptrl
        MOVFP   temp4, wreg
        MOVWF   tblptrh

        TABLRD  1, 0, temp1
        TLRD    1, temp1
        CALL    SET_TABLE_POINTER
        MOVFP   temp1, wreg
        TABLWT  1, 0, wreg
        CLRF    temp1, 1
60
        MOVLW   B'00000001'
        ADDWF   temp3, 1
        CLRF    wreg, 1
        ADDWFC  temp4, 1

        MOVFP   CHROM_PTR_LO, wreg
        CPFSEQ  temp3
        GOTO    BEST_CHROMO
        MOVFP   CHROM_PTR_HI, wreg
70      CPFSEQ  temp4
        GOTO    BEST_CHROMO
        CALL    MOTOROLA_WAITING

        CLRF    temp1, 1
        CALL    SET_TABLE_POINTER
        MOVFP   bestfitHi, wreg
        TABLWT  1, 0, wreg
        CALL    MOTOROLA_WAITING

80      CLRF    temp1, 1
        CALL    SET_TABLE_POINTER
        MOVFP   bestfitLo, wreg
        TABLWT  1, 0, wreg
        CALL    MOTOROLA_WAITING


        CLRF    temp1, 1
        CALL    SET_TABLE_POINTER
        MOVFP   generation_counterHi, wreg
90      TABLWT  1, 0, wreg
        CALL    MOTOROLA_WAITING

        CLRF    temp1, 1
        CALL    SET_TABLE_POINTER
        MOVFP   generation_counterMi, wreg
        TABLWT  1, 0, wreg
        CALL    MOTOROLA_WAITING

        CLRF    temp1, 1
100     CALL    SET_TABLE_POINTER
        MOVFP   generation_counterLo, wreg
        TABLWT  1, 0, wreg
        CALL    MOTOROLA_WAITING

        CLRF    wreg, 1
        CPFSEQ  temp2
```

```
          GOTO   GET_THE_NEW_FITNESS      ; Neues Chromosom und dessen Fitness empfangen

    ;*** Keine neuen Daten, PLD-Latch zuruecksetzen
110       CALL   SET_TABLE_POINTER
          CLRF   wreg, 1
          TABLWT 1, 0, wreg

    ;*** Auf das Start-Signal fuer die naechste Iteration warten
          CLRF   temp1, 1
          CALL   MOTOROLA_WAITING
          MOVFP  temp2, wreg
          ADDWF  RandLo, 1               ; Motorola-Seed
          CLRF   wreg, 1
120       ADDWFC RandHi, 1
          TABLWT 1, 0, wreg
          RETURN

    ;*** Fitness empfangen
    GET_THE_NEW_FITNESS
          CALL   get_worst_index
          MOVWF  temp7                   ; = Index des schlechtesten Chromosomes

          CLRF   temp1, 1
130       CALL   SET_TABLE_POINTER
          CLRF   wreg, 1
          TABLWT 1, 0, wreg

          CALL   MOTOROLA_WAITING

          MOVFP  temp2, wreg
          MOVWF  temp6                   ; = High Byte der Fitness

          CLRF   temp1, 1
140       CALL   SET_TABLE_POINTER
          CLRF   wreg, 1
          TABLWT 1, 0, wreg

          CALL   MOTOROLA_WAITING

          MOVFP  temp2, wreg
          MOVWF  temp5                   ; = Low Byte der neuen Fitness
          MOVLW  0x20                    ; = Zeiger auf temp5
          MOVWF  fsr1
150       MOVFP  temp7, wreg
          CALL   insert_fitness

    ;*** Empfang des Chromosomes
          CALL   get_worst_chrom
          MOVFP  tblptrh, wreg
          MOVWF  temp4
          MOVFP  tblptrl, wreg
          MOVWF  temp3

160       CLRF   temp5, 1
          CLRF   temp6, 1

    GET_THE_NEW_CHROMOSOME
          CLRF   temp1, 1
          CALL   SET_TABLE_POINTER
          CLRF   wreg, 1
          TABLWT 1, 0, wreg

          CALL   MOTOROLA_WAITING
170
          MOVFP  temp4, wreg
          MOVWF  tblptrh
          MOVFP  temp3, wreg
          MOVWF  tblptrl
          TABLWT 0, 0, temp2

          CLRF   temp1, 1
          CALL   SET_TABLE_POINTER
          CLRF   wreg, 1
```

```
180           TABLWT 1, 0, wreg

              CALL   MOTOROLA_WAITING

              MOVFP  temp4, wreg
              MOVWF  tblptrh
              MOVFP  temp3, wreg
              MOVWF  tblptrl
              TABLWT 1, 0, temp2

190           MOVLW  B'00000001'
              ADDWF  temp3, 1
              CLRF   wreg, 1
              ADDWFC temp4, 1

              MOVLW  B'00010000'
              ADDWF  temp5, 1
              CLRF   wreg, 1
              ADDWFC temp6, 1

200           MOVFP  chromosom_size_hi, wreg
              CPFSGT temp6
              GOTO   $+2
              GOTO   CALCULATE_ALL_THE_REST
              MOVFP  chromosom_size_hi, wreg
              CPFSEQ temp6
              GOTO   GET_THE_NEW_CHROMOSOME
              MOVFP  chromosom_size_lo, wreg
              CPFSLT temp5
              GOTO   CALCULATE_ALL_THE_REST
210           GOTO   GET_THE_NEW_CHROMOSOME

    CALCULATE_ALL_THE_REST

    ;*** Auf das Start-Signal fuer die naechste Iteration warten
              CLRF   temp1, 1
              CALL   MOTOROLA_WAITING
              MOVFP  temp2, wreg
              ADDWF  RandLo, 1              ; Motorola-Seed
              CLRF   wreg, 1
220           ADDWFC RandHi, 1
              TABLWT 1, 0, wreg

              CALL   sum_of_fitness         ; Noetig fuer "Roulette-Wheel-Selection"
              CALL   index_sort             ; Neusortierung der gesamten Population
              RETURN
    ;*************************************************************


    ;*************************************************************
230 ;*** Funktion:  SET_TABLE_POINTER
    ;*** Job:       "Table Pointer" fuer Datenuebergabe vorbereiten
    SET_TABLE_POINTER
              MOVLW  H'10'
              MOVWF  tblptrh
              CLRF   tblptrl, 1
              RETURN
    ;*************************************************************


240 ;*************************************************************
    ;*** Funktion:  MOTOROLA_WAITING
    ;*** Job:       Warten auf Motorola-Signal/-Daten
    MOTOROLA_WAITING
              MOVLW  B'00000001'
              CPFSEQ temp1
              GOTO   MOTOROLA_WAITING
              RETURN
    ;*************************************************************
```

D.15 Übergabe der besten Lösung: bestres.asm

```
;*************************************************************************
;***              Beste Loesung an den Motorola schicken                ***
;*** Autor:        Tobias Schubert                                       ***
;*** EMail:        schubert@informatik.uni-freiburg.de                   ***
;*** Datum:        04.10.1999                                            ***
;*** Datei-Name:   bestres.a                                             ***
;*************************************************************************


;*************************************************************************
;*** Funktion:   SEND_BEST_RESULTS
;*** Job:        Beste Loesung der genetischen Anwendung an den Motorola schicken.
;***
SEND_BEST_RESULTS
        CLRF    temp1, 1
        CALL    SET_TABLE_POINTER
        MOVLW   H'FF'                   ; $FF->"Signal", um beste Loesung zu schicken
        TABLWT  1, 0, wreg
        CALL    MOTOROLA_WAITING

;*** 1. Schritt: Anzahl benoetigter Generationen
        CLRF    temp1, 1
        CALL    SET_TABLE_POINTER
        MOVFP   generation_counterHi, wreg
        TABLWT  1, 0, wreg
        CALL    MOTOROLA_WAITING

        CLRF    temp1, 1
        CALL    SET_TABLE_POINTER
        MOVFP   generation_counterMi, wreg
        TABLWT  1, 0, wreg
        CALL    MOTOROLA_WAITING

        CLRF    temp1, 1
        CALL    SET_TABLE_POINTER
        MOVFP   generation_counterLo, wreg
        TABLWT  1, 0, wreg
        CALL    MOTOROLA_WAITING

;*** 2. Schritt: Anzahl "neuer" Chromosome pro Generation
        CLRF    temp1, 1
        CALL    SET_TABLE_POINTER
        MOVFP   number_of_children, wreg
        TABLWT  1, 0, wreg
        CALL    MOTOROLA_WAITING

;*** 3. Schritt: Groesse der Population
        CLRF    temp1, 1
        CALL    SET_TABLE_POINTER
        MOVFP   population_size, wreg
        TABLWT  1, 0, wreg
        CALL    MOTOROLA_WAITING

;*** 4. Schritt: Groesse der Chromosome
        CLRF    temp1, 1
        CALL    SET_TABLE_POINTER
        MOVFP   chromosom_size_hi, wreg
        TABLWT  1, 0, wreg
        CALL    MOTOROLA_WAITING

        CLRF    temp1, 1
        CALL    SET_TABLE_POINTER
        MOVFP   chromosom_size_lo, wreg
        TABLWT  1, 0, wreg
        CALL    MOTOROLA_WAITING


;*** 5. Schritt: Fitness des besten Chromosomes
        CLRF    temp1, 1
        CALL    SET_TABLE_POINTER
```

```
            MOVFP  bestfitHi, wreg
            TABLWT 1, 0, wreg
            CALL   MOTOROLA_WAITING

            CLRF   temp1, 1
            CALL   SET_TABLE_POINTER
            MOVFP  bestfitLo, wreg
            TABLWT 1, 0, wreg
            CALL   MOTOROLA_WAITING
 80
            CLRF   temp1, 1
            CALL   SET_TABLE_POINTER
            MOVFP  dist_remainder, wreg
            TABLWT 1, 0, wreg

    ;*** 6. Schritt: Das beste Chromosom
            MOVFP  BEST_CHROM_PTR_LO, wreg
            MOVWF  temp7
            MOVFP  BEST_CHROM_PTR_HI, wreg
 90         MOVWF  temp8

    CHROMO_LOOP
            CALL   MOTOROLA_WAITING

            MOVFP  temp7, wreg
            MOVWF  tblptrl
            MOVFP  temp8, wreg
            MOVWF  tblptrh

100         TABLRD 0, 0, temp10
            TLRD   0, temp10

            CLRF   temp1, 1
            CALL   SET_TABLE_POINTER
            MOVFP  temp10, wreg
            TABLWT 1, 0, wreg
            CALL   MOTOROLA_WAITING

            MOVFP  temp7, wreg
110         MOVWF  tblptrl
            MOVFP  temp8, wreg
            MOVWF  tblptrh

            TABLRD 1, 0, temp10
            TLRD   1, temp10

            CLRF   temp1, 1
            CALL   SET_TABLE_POINTER
            MOVFP  temp10, wreg
120         TABLWT 1, 0, wreg

            MOVLW  H'01'
            ADDWF  temp7, 1
            CLRF   wreg, 1
            ADDWFC temp8, 1

            MOVFP  CHROM_PTR_LO, wreg
            CPFSEQ temp7
            GOTO   CHROMO_LOOP
130         MOVFP  CHROM_PTR_HI, wreg
            CPFSEQ temp8
            GOTO   CHROMO_LOOP

    ;*** 7. Schritt: PLD-Register wieder zuruecksetzen mit 0-Vektor
            CALL   MOTOROLA_WAITING

            CLRF   temp1, 1
            CALL   SET_TABLE_POINTER
            clrf   wreg, 1
140         TABLWT 1, 0, wreg
            RETURN
    ;***
    ;*****************************************************************************
```

D.16 Problemspezifische Fitneßfunktion: `tspfit.asm`

```
;************************************************************
;***              Fitness-Funktion fuer das TSP-Problem    ***
;*** Autor:       Tobias Schubert                          ***
;*** EMail:       schubert@informatik.uni-freiburg.de      ***
;*** Datum:       22.10.1999                               ***
;*** Datei-Name:  tspfit.asm                               ***
;************************************************************


;************************************************************
;*** Funktion:   compute_tsp_fitness
;*** Job:        Bestimmen des Fitness-Wertes eines Chromosomes
;***             bzw. einer TSP-Route mithilfe der euklidischen
;***             Distanz D = sqrt( (x1-x2)^2 + (y1-y2)^2 ).
compute_tsp_fitness
        clrf    tsp_fit_Lo, 1
        clrf    tsp_fit_Hi, 1
        clrf    dist_remainder, 1

        call    get_indiv               ; tblptrl/h = Adresse von Chromosom
                                        ; mit Index wreg

compute_tsp_fitness_start2
        tablrd  0, 0, start_city        ; Ausgangsstadt der Route lesen
        tlrd    0, start_city

        clrf    tsp_temp5, 1
city_loop
        incf    tsp_temp5, 1            ; Staedtezaehler

        ;*** Staedte-ID lesen.
        tablrd  0, 1, next_city1
        tlrd    0, next_city1
        tlrd    1, next_city2

        ;*** Berechne Entfernung zwischen next_city1 und next_city2.
        call    compute_distance

        ;*** Ende der NO_OF_CITIES-vielen Staedte?
        movlw   NO_OF_CITIES/2
        cpfseq  tsp_temp5
        goto    $+2                     ; Auslassen, falls loop_reg2 = POP_SIZE
        goto    $+5                     ; Entfernung zum Startpkt. noch addieren

        tablrd  0, 0, next_city1
        tlrd    0, next_city1
        call    compute_distance
        goto    city_loop

        movfp   start_city, wreg
        movwf   next_city1
        call    compute_distance

        movfp   tsp_fit_Lo, wreg
        movwf   tsp_temp1
        movlw   H'FF'
        movwf   tsp_fit_Lo
        movfp   tsp_temp1, wreg
        subwf   tsp_fit_Lo, 1

        movfp   tsp_fit_Hi, wreg
        movwf   tsp_temp1
        movlw   H'FF'
        movwf   tsp_fit_Hi
        movfp   tsp_temp1, wreg
        subwf   tsp_fit_Hi, 1
        return
;************************************************************


```

```
;*************************************************************
;*** Funktion:   compute_distance
;*** Job:        Bestimme Entfernung zwischen next_city1 und
;***             next_city2 mithilfe der zu Beginn erzeugten
;***             Diagonal-Matrix.
compute_distance
        ;*** Aktuellen Zeiger von "compute_tsp_fitness" speichern.
        movfp   tblptrl, wreg
        movwf   tmp_ptr_Lo
        movfp   tblptrh, wreg
        movwf   tmp_ptr_Hi

        movfp   PP_PTR_LO, wreg
        movwf   tblptrl
        movfp   PP_PTR_HI, wreg
        movwf   tblptrh

        movfp   next_city1, wreg
        cpfsgt  next_city2
        goto    $+4                     ; Skip, if next_city2 > next_city1
        movfp   next_city1, wreg        ; next_city2 > next_city1
        movwf   pp1
        goto    $+3
        movfp   next_city2, wreg        ; next_ciyt2 =< next_city1
        movwf   pp1

        ;*** pp1 enthaelt kleineren Index beider Staedte.
        movlw   NO_OF_CITIES-1
        movwf   pp2

        movlw   H'01'
        cpfseq  pp1
        goto    $+02
        goto    $+0C

        clrf    tsp_temp6, 1
address_hi_loop
        movfp   pp2, wreg
        addwf   tblptrl, 1
        clrf    wreg, 1
        addwfc  tblptrh, 1

        decf    pp2, 1

        incf    tsp_temp6, 1
        movfp   pp1, wreg
        decf    wreg, 1
        cpfseq  tsp_temp6
        goto    address_hi_loop

        movfp   next_city1, wreg
        cpfsgt  next_city2
        goto    $+6                     ; Skip, if next_city2 > next_city1
        movfp   next_city2, wreg        ; next_city2 > next_city1
        movwf   pp1
        movfp   next_city1, wreg
        subwf   pp1, 1
        goto    $+5
        movfp   next_city1, wreg        ; next_ciyt2 =< next_city1
        movwf   pp1
        movfp   next_city2, wreg
        subwf   pp1, 1

        movfp   pp1, wreg
        decf    wreg, 1
        addwf   tblptrl, 1
        clrf    wreg, 1
        addwfc  tblptrh, 1

        ;*** Distanz auslesen.
        tablrd  0, 0, pp1
        tlrd    0, pp1                  ; "Vorkommastellen"
        tlrd    1, pp2                  ; Nachkommastellen
```

```
        movfp   pp2, wreg
        addwf   dist_remainder, 1
        btfsc   alusta, 0               ; C=1, d.h. Nachkommastellen >= 1 ?
        goto    $+2                     ; Skip, if Carry = 0
        goto    $+4                     ; Nachkommastellen =< 1.000
150
        incf    tsp_fit_Lo, 1
        clrf    wreg, 1
        addwfc  tsp_fit_Hi, 1

        movfp   pp1, wreg
        addwf   tsp_fit_Lo, 1
        clrf    wreg, 1
        addwfc  tsp_fit_Hi, 1

160     ;*** Urspruengliche Zeiger wieder herstellen.
        movfp   tmp_ptr_Lo, wreg
        movwf   tblptrl
        movfp   tmp_ptr_Hi, wreg
        movwf   tblptrh
        return
;*****************************************************************


;*****************************************************************
170 ;*** Funktion:  preprocessing
;*** Job:       Bestimme Entfernung zwischen allen Staedten und
;***            speichere die Ergebnisse in einer Matrix.
preprocessing
        ;*** Zeiger auf die "Matrix" initialisieren.
        movlw   H'5D'
        movwf   tblptrh
        movwf   PP_PTR_HI
        movlw   H'AB'
        movwf   tblptrl
180     movwf   PP_PTR_LO

        ;*** Entfernung zwischen je 2 Staedten berechnen.
        movlw   H'01'
        movwf   pp1
next_city1_loop
        movfp   pp1, wreg
        movwf   next_city1
        incf    wreg, 1
        movwf   pp2
190 next_city2_loop
        movfp   pp2, wreg
        movwf   next_city2

        ;*** Berechne Strecke zwischen next_city1 und next_city2.
        clrf    tsp_fit_Lo, 1
        clrf    tsp_fit_Hi, 1
        clrf    dist_remainder, 1
        call    pp_distance

200     ;*** Berechnete Strecke speichern.
        tlwt    0, tsp_fit_Lo           ; "Vorkommastellen"
        tablwt  1, 1, dist_remainder    ; Nachkommastellen

        incf    pp2, 1
        movlw   NO_OF_CITIES+1
        cpfseq  pp2
        goto    next_city2_loop         ; Skip, if pp2 = NO_OF_CITIES+1

        incf    pp1, 1
210     movlw   NO_OF_CITIES
        cpfseq  pp1
        goto    next_city1_loop         ; Skip, if pp1 = NO_OF_CITIES

        return
;*****************************************************************
```

```
;*************************************************************
;*** Funktion:   pp_distance
;*** Job:        Bestimme Entfernung zwischen next_city1 und
;***             next_city2 mithilfe der euklidischen
;***             Distanz D = sqrt( (x1-x2)^2 + (y1-y2)^2 ) und
;***             addiere das Ergebnis zu tsp_fit_Lo/Hi.
;***             (8 binaere Nachkommastellen)
pp_distance
        ;*** X-Koordinaten aus Tabelle auslesen.
        movfp   next_city1, wreg
        decf    wreg, 1
        call    TABLE_OF_X_KOORD
        movwf   tsp_temp1
        movfp   next_city2, wreg
        decf    wreg, 1
        call    TABLE_OF_X_KOORD
        movwf   tsp_temp2

        ;*** Berechne tsp_temp1 = abs( tsp_temp1 - tsp_temp2 ).
        movfp   tsp_temp1, wreg
        cpfslt  tsp_temp2
        call    change_tsp_temp1_tsp_temp2      ; skip, if temp2<temp1
        movfp   tsp_temp2, wreg
        subwf   tsp_temp1, 1

        ;*** Berechne x*x und speichere das Ergebnis in distanceLo/Hi.
        clrf    distanceLo, 1
        clrf    distanceHi, 1
        movfp   tsp_temp1, wreg
        mulwf   tsp_temp1
        movfp   prodLo, wreg
        movwf   distanceLo
        movfp   prodHi, wreg
        movwf   distanceHi

        ;*** Y-Koordinaten aus Tabelle auslesen.
        movfp   next_city1, wreg
        decf    wreg, 1
        call    TABLE_OF_Y_KOORD
        movwf   tsp_temp1
        movfp   next_city2, wreg
        decf    wreg, 1
        call    TABLE_OF_Y_KOORD
        movwf   tsp_temp2

        ;*** Berechne tsp_temp1 = abs( tsp_temp1 - tsp_temp2 ).
        movfp   tsp_temp1, wreg
        cpfslt  tsp_temp2
        call    change_tsp_temp1_tsp_temp2      ; skip, if temp2<temp1
        movfp   tsp_temp2, wreg
        subwf   tsp_temp1, 1

        ;*** Berechne y*y und addiere das Ergebnis zu distanceLo/Hi.
        movfp   tsp_temp1, wreg
        mulwf   tsp_temp1
        movfp   prodLo, wreg
        addwf   distanceLo, 1
        clrf    wreg, 1
        addwfc  distanceHi, 1
        movfp   prodHi, wreg
        addwf   distanceHi, 1

        ;*** Zur Wurzel-Berechnung distanceLo/Hi nach dist_tmpLo/Mi/Hi/SHi
        ;*** kopieren und mit 65536 multiplizieren, da sqrt(65536)=256, was
        ;*** dann bewirkt, dass Bit0..7 acht binaere Nachkommastellen sind.
        clrf    dist_tmpSHi, 1
        clrf    dist_tmpHi, 1
        movfp   distanceHi, wreg
        movwf   dist_tmpMi
        movfp   distanceLo, wreg
        movwf   dist_tmpLo
```

```
290         ;*** Multiplikation mit 65536 bedeutet 16 Links-Shifts (65536 = 2^16).
            clrf    tsp_temp6, 1
    shift_loop
            rlcf    dist_tmpLo, 1
            rlcf    dist_tmpMi, 1
            rlcf    dist_tmpHi, 1
            rlcf    dist_tmpSHi, 1
            bcf     alusta, 0
            incf    tsp_temp6, 1
            movlw   D'16'
300         cpfseq  tsp_temp6
            goto    shift_loop

            ;*** Wurzel von dist_tmpLo/Mi/Hi/SHi berechnen mit folgender Iterations-
            ;*** formel: x(i+1) = 0.5 * (x(i) + dist_tmp/x(i)).
            ;*** Startwert: 1. Abbruch, falls x(i+1) = x(i) oder nach $FF Iterationen.
            ;*** x wird in sqrtLo/Mi/Hi/SHi gespeichert.
            clrf    tsp_temp6, 1            ; Zaehler
            clrf    sqrtSHi, 1
            clrf    sqrtHi, 1
310         clrf    sqrtMi, 1
            movlw   H'01'
            movwf   sqrtLo

    Sqrt_Loop
            movfp   sqrtSHi, wreg
            movwf   old_sqrtSHi
            movfp   sqrtHi, wreg
            movwf   old_sqrtHi
            movfp   sqrtMi, wreg
320         movwf   old_sqrtMi
            movfp   sqrtLo, wreg
            movwf   old_sqrtLo

            ;*** Berechne dist_tmp/x(i).
            movfp   dist_tmpLo, wreg
            movwf   numerator32Lo
            movfp   dist_tmpMi, wreg
            movwf   numerator32Mi1
            movfp   dist_tmpHi, wreg
330         movwf   numerator32Mi2
            movfp   dist_tmpSHi, wreg
            movwf   numerator32Hi

            movfp   sqrtSHi, wreg
            movwf   denominator32Hi
            movfp   sqrtHi, wreg
            movwf   denominator32Mi2
            movfp   sqrtMi, wreg
            movwf   denominator32Mi1
340         movfp   sqrtLo, wreg
            movwf   denominator32Lo
            call    div32

            ;*** Berechne x(i) + dist_tmp/x(i).
            movfp   sqrtLo, wreg
            addwf   quotiont32Lo, 1
            clrf    wreg, 1
            addwfc  quotiont32Mi1, 1
            movfp   sqrtMi, wreg
350         addwf   quotiont32Mi1, 1
            clrf    wreg, 1
            addwfc  quotiont32Mi2, 1
            movfp   sqrtHi, wreg
            addwf   quotiont32Mi2, 1
            clrf    wreg, 1
            addwfc  quotiont32Hi, 1
            movfp   sqrtSHi, wreg
            addwf   quotiont32Hi, 1

360         ;*** Berechne x(i+1) = 0.5 * (x(i) + dist_tmp/x(i)).
            rrcf    quotiont32Hi, 1
            rrcf    quotiont32Mi2, 1
```

```
        rrcf    quotiont32Mi1, 1
        rrcf    quotiont32Lo, 1

        movfp   quotiont32Hi, wreg
        movwf   sqrtSHi
        movfp   quotiont32Mi2, wreg
        movwf   sqrtHi
370     movfp   quotiont32Mi1, wreg
        movwf   sqrtMi
        movfp   quotiont32Lo, wreg
        movwf   sqrtLo

        ;*** Weitere Iterationen (maximal 256)?
        incf    tsp_temp6, 1
        clrf    wreg, 1
        cpfseq  tsp_temp6
        goto    $+02                    ; Skip, if tsp_temp6 = H'FF'
380     goto    $+0D

        movfp   old_sqrtSHi, wreg
        cpfseq  sqrtSHi
        goto    Sqrt_Loop               ; Skip, if old_sqrtSHi = sqrtSHi
        movfp   old_sqrtHi, wreg
        cpfseq  sqrtHi
        goto    Sqrt_Loop               ; Skip, if old_sqrtHi = sqrtHi
        movfp   old_sqrtMi, wreg
        cpfseq  sqrtMi
390     goto    Sqrt_Loop               ; Skip, if old_sqrtMi = sqrtMi
        movfp   old_sqrtLo, wreg
        cpfseq  sqrtLo
        goto    Sqrt_Loop               ; Skip, if old_sqrtLo = sqrtLo

        ;*** Fitness der Route aktualisieren (zuerst die 8 Nachkomma-Stellen).
        movfp   sqrtLo, wreg
        addwf   dist_remainder, 1
        btfsc   alusta, 0               ; C=1, d.h. Nachkommastellen >= 1 ?
        goto    $+2                     ; Skip, if Carry = 0
400     goto    $+4                     ; Nachkommastellen =< 1.000

        incf    tsp_fit_Lo, 1
        clrf    wreg, 1
        addwfc  tsp_fit_Hi, 1

        clrf    tsp_temp6, 1
backshift_loop
        rrcf    sqrtSHi, 1
        rrcf    sqrtHi, 1
410     rrcf    sqrtMi, 1
        rrcf    sqrtLo, 1
        bcf     alusta, 0
        incf    tsp_temp6, 1
        movlw   D'8'
        cpfseq  tsp_temp6
        goto    backshift_loop

        movfp   sqrtLo, wreg
        addwf   tsp_fit_Lo, 1
420     clrf    wreg, 1
        addwfc  tsp_fit_Hi, 1
        movfp   sqrtMi, wreg
        addwf   tsp_fit_Hi, 1
        return

;*** Hilfsfunktion:
change_tsp_temp1_tsp_temp2
        movfp   tsp_temp1, wreg
        movwf   tsp_temp3
430     movfp   tsp_temp2, wreg
        movwf   tsp_temp1
        movfp   tsp_temp3, wreg
        movwf   tsp_temp2
        return
;*****************************************************************
```

D.17 Hilfsfunktionen der Operatoren: `initops.asm`

```
;*************************************************************
;***              Genetische Operatoren einleiten          ***
;*** Autor:        Tobias Schubert                          ***
;*** EMail:        schubert@informatik.uni-freiburg.de      ***
;*** Datum:        18.10.1999                               ***
;*** Datei-Name:   initops.asm                              ***
;*************************************************************


;*************************************************************
;*** Fitness der gesamten Population bestimmen:
compute_new_fitness
        movlw   H'01'
        movwf   loop_reg4               ; Chromosomen-Zaehler
first_fitness
        movfp   loop_reg4, wreg         ; Bestimme Fitness von Chromo. mit Index wreg
        call    compute_tsp_fitness     ; Bestimmen der Fitness der akt. TSP-Route
        movlw   TSP_FITNESS_PTR
        movwf   fsr1                    ; fsr1 = Adresse des einzufuegenden Fitnesswertes
        movfp   loop_reg4, wreg         ; wreg = Index des akt. Chromosomes
        call    insert_fitness          ; Fitness aktualisieren
        incf    loop_reg4, 1
        movlw   POP_SIZE
        cpfsgt  loop_reg4
        goto    first_fitness           ; naechstes Chromosom bewerten
        return
;*************************************************************


;*************************************************************
;*** PMX-Crossover:
make_crossover
        call    find_two_chromosoms     ; Zwei Elternteile bestimmen
        call    tsp_crossover           ; Crossover-Operation
        return
;*************************************************************


;*************************************************************
;*** Greedy-Crossover:
make_greedy_crossover
        call    find_two_chromosoms     ; Zwei Elternteile bestimmen
        call    greedy_crossover        ; Crossover-Operation
        return
;*************************************************************


;*************************************************************
;*** Mutation:
make_mutation
        ;*** Geeignetes Chromosom auswaehlen
        call    roulette_wheel_selection

        movpf   temp1, wreg             ; temp1 = wreg = Index obiges Chromosom
        movwf   new_child_index
        call    get_indiv               ; tblptrl/h = Adresse von Chromosom
                                        ; mit Index wreg
        movfp   tblptrh, wreg
        movwf   tsp_ptr1_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr1_Lo
        movfp   children_counter, wreg
        incf    wreg, 1
        movwf   no_of_child
        call    get_child               ; tblptrl/h = adrLo/Hi = Adresse des Kindes
        movfp   tblptrh, wreg
        movwf   tsp_ptr3_Hi
        movwf   tsp_ptr2_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr3_Lo
```

```
        movwf   tsp_ptr2_Lo

        ;*** Chromosom an den Platz des Kindes kopieren.
        call    Data_Copy

        call    tsp_mutation            ; Mutation ausfuehren
        return
;*************************************************************

80
;*************************************************************
;*** Lokale Verbesserung eines Chromosomes:
make_local_improvement
        ;*** Geeignetes Chromosom auswaehlen
        call    roulette_wheel_selection

        movpf   temp1, wreg             ; temp1 = wreg = Index obiges Chromosom
        movwf   new_child_index
        call    get_indiv               ; tblptrl/h = Adresse von Chromosom
90                                      ; mit Index wreg
        movfp   tblptrh, wreg
        movwf   tsp_ptr1_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr1_Lo
        movfp   children_counter, wreg
        incf    wreg, 1
        movwf   no_of_child
        call    get_child               ; tblptrl/h = adrLo/Hi = Adresse des Kindes
        movfp   tblptrh, wreg
100     movwf   tsp_ptr2_Hi
        movwf   tsp_ptr3_Hi
        movfp   tblptrl, wreg
        movwf   tsp_ptr2_Lo
        movwf   tsp_ptr3_Lo

        ;*** Chromosom an den Platz des Kindes kopieren.
        call    Data_Copy

        movfp   tsp_ptr3_Hi, wreg
110     movwf   tsp_ptr1_Hi
        movfp   tsp_ptr3_Lo, wreg
        movwf   tsp_ptr1_Lo

        clrf    loop_reg3, 1            ; interner Zaehler
Find4opt_Loop
        call    find4opt                ; lokale Verbesserung
        incf    tsp_ptr1_Lo, 1
        clrf    wreg, 1
        addwfc  tsp_ptr1_Hi, 1
120     incf    loop_reg3, 1
        movlw   SIZE/4
        cpfseq  loop_reg3
        goto    Find4opt_Loop
        return
;*************************************************************


;*************************************************************
;*** Abbruchbedingung pruefen: Hat sich die Fitness innerhalb
130 ;*** einer bestimmten Anzahl von Generationen verbessert?
Improve_Test
        movlr   bank0
        incf    improve_fitness_lo, 1   ; "Alter" des aktuellen besten Chromosomes
        clrf    wreg, 1
        addwfc  improve_fitness_hi, 1

        movfp   bestfitHi, wreg
        cpfslt  best_fit_old_hi
        goto    SECOND_TST              ; best_fit_old_hi = bestfithi
140     clrf    improve_fitness_hi, 1
        clrf    improve_fitness_lo, 1
        movfp   bestfitHi, wreg
        movwf   best_fit_old_hi
```

```
        movfp   bestfitLo, wreg
        movwf   best_fit_old_lo
        movfp   dist_remainder, wreg
        movwf   best_old_remainder
        goto    gen_alg_loop

SECOND_TST
        movfp   bestfitLo, wreg
        cpfslt  best_fit_old_lo
        goto    THIRD_TST               ; best_fit_old_lo = bestfitlo
        clrf    improve_fitness_hi, 1
        clrf    improve_fitness_lo, 1
        movfp   bestfitHi, wreg
        movwf   best_fit_old_hi
        movfp   bestfitLo, wreg
        movwf   best_fit_old_lo
        movfp   dist_remainder, wreg
        movwf   best_old_remainder
        goto    gen_alg_loop

THIRD_TST
        movfp   dist_remainder, wreg
        cpfslt  best_old_remainder
        goto    FOURTH_TST              ; best_old_remainder = dist_remainder
        clrf    improve_fitness_hi, 1
        clrf    improve_fitness_lo, 1
        movfp   bestfitHi, wreg
        movwf   best_fit_old_hi
        movfp   bestfitLo, wreg
        movwf   best_fit_old_lo
        movfp   dist_remainder, wreg
        movwf   best_old_remainder
        goto    gen_alg_loop

FOURTH_TST
        movlw   IMP_G_HI
        cpfseq  improve_fitness_hi
        goto    gen_alg_loop            ; improve_fitness_hi < IMP_G_HI
        movlw   IMP_G_LO                ; improve_fitness_hi = IMP_G_HI
        cpfseq  improve_fitness_lo
        goto    gen_alg_loop            ; improve_fitness_lo < IMP_G_LO
        goto    end_gen_alg             ; Abbruchbedingung  erfuellt
;*************************************************************
```

Anhang E

Programm-Datei zum Threshold-Accepting-Algorithmus

```
/* Heuristisches Loesen des TSP-Problems */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* Anzahl Staedte */
int no_of_cities;

/* Bestes Ergebnis der besten Route */
float distance;

/* Beste Route */
int best_route[253];

/* Koordinaten der Staedte */
int x[253], y[253];

/* Aktuelle Route */
int akt_route[253];

/* Aktuelle Distanz */
float akt_distance;

/* Anzahl Iterationen, Threshold */
int iteration;
float threshold;

/* Mutations-Positionen: */
int pos1, pos2, tmp;

/* Schleifen-Variablen */
int i;

/* Hauptprogramm: */
int main(void)
        {
        printf("\n\n Heuristische TSP-Berechnung: \n\n");
        printf(" Wieviele Staedte: ");
        scanf("%d", &no_of_cities);
        printf("\n");
        for(i=1; i<no_of_cities+1; i=i+1)
                {
                printf(" X-Koordinate von Stadt %d: ",i);
                scanf("%d", &x[i]);
                }
        printf("\n");
        for(i=1; i<no_of_cities+1; i=i+1)
                {
                printf(" Y-Koordinate von Stadt %d: ",i);
                scanf("%d", &y[i]);
```

```
        }

    /* Start-Route festlegen */
    for (i=1; i<no_of_cities+1; i=i+1) {best_route[i] = i;}

    /* Start-Distanz bestimmen */
    distance = 0.0;
    for (i=2; i<no_of_cities+1; i=i+1)
        {
60      distance = distance + sqrt( (x[i]-x[i-1])*(x[i]-x[i-1])
                                   +(y[i]-y[i-1])*(y[i]-y[i-1]) );
        }
    distance = distance + sqrt( (x[1]-x[no_of_cities])*(x[1]-x[no_of_cities])
                              +(y[1]-y[no_of_cities])*(y[1]-y[no_of_cities]) );
    iteration = 0;
    threshold = 0.005 * distance;
    while(iteration<250000)
        {
        iteration = iteration+1;
70      for (i=1; i<no_of_cities+1; i=i+1) {akt_route[i] = best_route[i];}
        pos1 = 0;
        pos2 = 0;
        while(pos1 == 0 || pos1 == no_of_cities+1)
            {
            pos1 = random() % no_of_cities+1;
            }
        while(pos2 == 0 || pos2 == no_of_cities+1 || pos2 == pos1)
            {
            pos2 = random() % no_of_cities+1;
80          }
        tmp = akt_route[pos1];
        akt_route[pos1] = akt_route[pos2];
        akt_route[pos2] = tmp;
        akt_distance = 0.0;
        for (i=2; i<no_of_cities+1; i=i+1)
            {
            akt_distance = akt_distance +
                           sqrt( (x[akt_route[i]]-x[akt_route[i-1]])
                                *(x[akt_route[i]]-x[akt_route[i-1]])
90                              +(y[akt_route[i]]-y[akt_route[i-1]])
                                *(y[akt_route[i]]-y[akt_route[i-1]]) );
            }

        akt_distance = akt_distance +
                       sqrt( (x[akt_route[1]]-x[akt_route[no_of_cities]])
                            *(x[akt_route[1]]-x[akt_route[no_of_cities]])
                            +(y[akt_route[1]]-y[akt_route[no_of_cities]])
                            *(y[akt_route[1]]-y[akt_route[no_of_cities]]) );

100     if(akt_distance < distance+threshold)
            {
            iteration = 0;
            if(threshold > 0) {threshold = threshold - 0.002;}
                    else      {threshold = 0.0;}
            distance = akt_distance;
            for (i=1; i<no_of_cities+1; i=i+1)
                    {best_route[i] = akt_route[i];}
            printf("\n Aktuelle Loesung: %09.3f, ", distance);
            printf("Momentaner Threshold = %09.3f",threshold);
110         }
        }
    printf("\n\n Beste Loesung: %09.3f \n", distance);
    printf(" Beste Route: ");
    for(i=1; i<no_of_cities+1; i=i+1) {printf(" %d", best_route[i]);}
    printf("\n\n");
    getchar(); getchar();
    return 0;
    }
```

Anhang F

Inhalt der CD-ROM

Abschliessend ist an dieser Stelle eine CD-ROM beigefügt, die alle vorgestellten Programme, Entwicklungs- und Programmier-Tools enthält.
Abbildung F.1 zeigt schematisch die Verzeichnisstruktur, für genauere Informationen zu den einzelnen Ordnern und Programmen sei auf die *README*-Datei im Ursprungsverzeichnis der CD-ROM verwiesen.

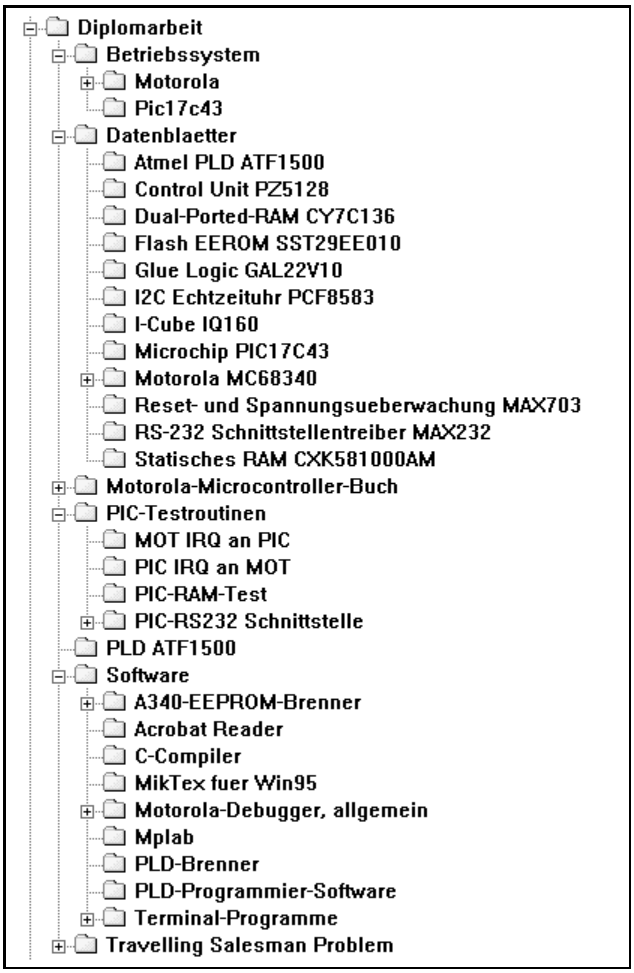


Abbildung F.1: Verzeichnisstruktur der CD-ROM