CRAY-Handbuch

Einführung in die Benutzung der CRAY

1. Auflage Mai 1987

Technical Report 87-1

Konrad-Zuse-Zentrum für Informationstechnik Berlin;
Heilbronner Straße 10; D-1000 Berlin 31

VORWORT

Das vorliegende Handbuch wendet sich an alle Benutzer des für den Norddeutschen Vektorrechnerverbund im ZIB installierten Rechners CRAY X-MP/24.

Ein großer Teil der Darstellung ist der Optimierung von Programmen gewidmet, die in FORTRAN geschrieben sind. Gerade weil die CRAY X-MP einer der schnellsten Rechner der Welt ist, wird jeder Benutzer früher oder später selbst auf dieses Thema stoßen, denn die Maschine ist vornehmlich für Probleme bestimmt, die für die meisten anderen Rechner zu komplex sind. Maschinen der CRAY-Leistungsklasse beziehen einen erheblichen Teil ihrer Geschwindigkeit aus der parallelen Arbeit vieler Komponenten. Programme, die die jeweilige Architektur der Maschine berücksichtigen, können um Größenordnungen schneller (also wirtschaftlicher) sein als Programme, deren Autor beim Schreiben einen sogenannten "klassischen" Rechner vor seinem geistigen Auge hatte. Daher wird auch der technische Aufbau der CRAY X-MP im Handbuch erläutert.

Weitere Teile des Handbuches beschäftigen sich mit den Vorrechnern und dem Rechnernetz. Kaum ein Benutzer wird alle Teile des Handbuches für seine Arbeit benötigen, wir hoffen andererseits, daß das Handbuch für jeden potentiellen Benutzer einen leichten "Einstieg" in die Benutzung unseres Rechners ermöglicht, gleichgültig aus welcher DV-Umgebung er kommt.

Ich danke allen, die an der sorgfältigen Ausarbeitung dieses Handbuches beteiligt waren. Jeder Betreiber und jeder Benutzer komplexer Rechnersysteme weiß um den hoffnungslosen Wettlauf zwischen technischer Entwicklung und der Dokumentation für die Benutzer. Wir hoffen, daß wir das Handbuch als aktuelle und nützliche Arbeitsbasis so lange erhalten können, wie wir den Vektorrechner im ZIB betreiben.

Jürgen Gottschewski

Berlin, im Mai 1987

Dieses Handbuch wurde verfaßt unter Verwendung der Schriften

- Rechenzentrum der Universität Stuttgart: CRAY-Handbuch, Einführung in die Benutzung der CRAY, Rev. 1, Dok.-Nr. 84/037, SY 10/200-1, Stuttgart 1986

- Zentralinstitut für Angewandte Mathematik der Kernforschungsanlage Jülich GmbH: Einführung in die Benutzung der CRAY, Jülich 1984

Wir danken für die freundliche Erlaubnis zur Verwendung der Schriften.

Im ZIB waren insbesondere Hubert Busch, Uwe Pöhle und Wolfgang Stech an der Erstellung dieser Schrift beteiligt; darüber hinaus sind einzelne Kapitel von weiteren, z.T. inzwischen ehemaligen Mitarbeitern des ZIB bearbeitet worden. Die redaktionelle Arbeit lag bei Wolfgang Stech.

Das ZIB übernimmt für die Fehlerfreiheit der hier beschriebenen Programme keine Gewährleistung oder Haftung, da es nach dem Stand der Technik nicht möglich ist, Datenverarbeitungsprogramme so zu entwickeln, daß sie fehlerfrei arbeiten.

Inhaltsverzeichnis

1. Zugang zur CRAY

1.1 Allgemeine Information über die CRAY X-MP des ZIB

1.1.1 Rechenarten

Die CRAY X-MP ist ein Vektorrechner. Ihr Befehlsvorrat umfaßt neben den üblichen arithmetischen Befehlen für einzelne Zahlen auch Vektorbefehle, die gleichartige Operationen auf Zahlenkolonnen (Vektoren) ausführen. Jeder dieser Vektorbefehle stößt eine größere Anzahl von Operationen an, die stark überlappt ablaufen. Die Überlappung führt zu einer großen mittleren Anzahl von Operationen je Zeiteinheit und damit zu der hohen Rechenleistung der Anlage.

Die Überlappung von Operationen wird auch in konventionellen Rechnern zur Leistungssteigerung eingesetzt. Im Vektorrechner sind der Strom der Operanden und die Steuerung des Rechenvorgangs jedoch in besonderem Maße aufeinander abgestimmt. Die Zusammenfassung gleichartiger Operationen zu mächtigen Vektorbefehlen ist das besondere Merkmal der Vektorrechner.

Die folgende Schleife wird auf einem konventionellen Rechner durch wiederholte Ausführung einer Folge arithmetischer Befehle abgearbeitet:

```
      DO  100  I=1,50
100         A(I)  =  B(I)*C(I)
```

Auf der CRAY werden die Multiplikationen mit einem einzigen Befehl angestoßen und laufen dann überlappt ab. Beispielsweise beginnt das Rechenwerk mit der Berechnung von B(2)*C(2) kurz nachdem die Berechnung von B(1)*C(1) angelaufen ist, und lange ehe der Wert des Produkts B(1)*C(1) ermittelt ist.

Vektorielle Verarbeitung

Die an einer Vektoroperation beteiligten Datenströme bezeichnet man als Vektoren. Ein Vektor im Sinne des Rechners CRAY X-MP ist eine Folge von Werten, die im Speicher aufeinanderfolgend oder in gleichem Abstand gespeichert sind. In FORTRAN sind dies z.B. eindimensionale Felder, Spalten und Zeilen einer Matrix oder systematische Ausschnitte solcher Einheiten. Vektoren können durch die üblichen arithmetischen Operationen verknüpft werden.

Die hohe Leistung des Vektorrechners kann besonders dann ausgeschöpft werden, wenn die Vektoren lang sind. Für Vektoren mit weniger als fünf Elementen bringen die Vektorbefehle gegenüber wiederholter Ausführung konventioneller Befehle noch keinen Gewinn, da der Prozessor eine gewisse Vorlaufzeit benötigt, ehe er die volle Vektorleistung erreicht. Bei Vektoren der Länge 30 ist der Gewinn erheblich und bei der Länge 64 für die CRAY schon optimal.

Die Vektorlänge 64 spielt für die CRAY deshalb eine zentrale Rolle, weil die Daten für die Verarbeitung in speziellen Vektorregistern zurechtgelegt werden, die je 64 Elemente aufnehmen können. Der Anwender braucht jedoch nicht eine Aufteilung der Vektoren in 64-er Gruppen vorzunehmen: dies erledigt der FORTRAN-Compiler automatisch. Das Augenmerk ist lediglich darauf zu richten, möglichst lange Vektoren zu bearbeiten.

Arithmetische Ausdrücke

Arithmetische Ausdrücke mit Vektoren und Skalaren als Operanden werden dann besonders schnell ausgeführt, wenn möglichst viele Zwischenergebnisse in Registern gehalten werden können, so daß die Anzahl der zeitaufwendigen Zugriffe zum Hauptspeicher gering ist. Dies erreicht man, indem jeweils möglichst viele Operanden zu einem arithmetischen Ausdruck zusammengefaßt werden.

Bei einigen Kombinationen von arithmetischen Verknüpfungen können mehrere Vektorbefehle vollständig gleichzeitig ablaufen, wodurch lokal eine Verdopplung der Leistung erreicht wird, z.B.

```
      A(I) = C(I) + X * D(I)
```

Dies ist immer dann möglich, wenn verschiedene Prozessoreinheiten angesprochen werden, z.B. die Multiplikationseinheit und die Additionseinheit. Dabei brauchen die Operanden nicht voneinander unabhängig zu sein, da der Rechner mit Hilfe einer als "Chaining" bezeichneten Technik die Resultate einer Operation, z.B. der Multiplikation, als Operanden der nächsten Operation, z.B. der Addition, unmittelbar verwendet.

Theoretisch ist jede CPU (Central Processing Unit) der CRAY X-MP in der Lage, bis zu 210 Millionen Gleitkommaoperationen in der Sekunde auszuführen. Dies kann allerdings nur im Idealfall erreicht werden, in dem drei verschiedene Operationen (z.B. Betragsbildung, Addition und Multiplikation) durch Verkettung (chaining) verknüpft werden können. Dem Grenzwert wird man sich jedoch in der Praxis nur in seltenen Fällen und bei sehr speziellen Fragestellungen nähern können. Die maximale Rechenleistung kann auch nur dann erreicht werden, wenn bei der Rechnung nicht zu viele Hauptspeicherzugriffe notwendig sind und wenn die Parallelverarbeitung fortlaufend ausgenutzt werden kann. In der Praxis ist ein mehrfaches von 30 Millionen Gleitkommaoperationen pro Sekunde als mittlere Leistung über einen Programmlauf erreichbar.

Die Höchstleistung bei der Vektorverarbeitung wird stark beeinträchtigt, wenn der Datenstrom für die Verknüpfungen nicht ungehindert fließen kann. Dies ist zum Beispiel der Fall, wenn die Operationen von Bedingungen abhängig sind

```
      DO 100 I=1,3000
      IF (X(I) .GT. 0.0) Y(I)=X(I)*Z
100   CONTINUE
```

oder wenn Resultate als Operanden der gleichen Operation benötigt werden

```
      DO 200 I=2,3000
200     X(I) = X(I-1)*Y(I)
```

In dem letzten Beispiel wird in der Anweisung 200 z.B. das Element X(2) für die Berechnung von X(3) benötigt. Die zweite Berechnung kann also nicht gleichzeitig mit der ersten ablaufen. Der Compiler erkennt derartige Probleme und generiert hier konventionelle Befehle für Einzelbearbeitung. Wenn aus einem dieser oder anderer Gründe Vektorverarbeitung nicht möglich ist, informiert der Compiler über die Ursache in den Übersetzungslisten.

Ausführliche Hinweise über die Möglichkeiten der Autovektorisierung des FORTRAN-Übersetzers und der Einflußnahme durch den Benutzer finden sich im Kapitel 9.

Neben den Techniken zur Optimierung eines Programmes auf einer CPU gewinnen neuerdings Techniken an Bedeutung, die Leistungen mehrerer oder aller Prozessoren eines DV-Systems auf ein einziges Programm zu konzentrieren. Im COS stehen hierfür die Techniken des Multitasking, Macrotasking und Microtasking

zur Verfügung. Da die Anlage des ZIB nur über zwei Prozessoren verfügt, werden nur wenige existierende Anwendungen auf der Anlage des ZIB aus diesen Techniken Gewinn ziehen können. Sie sind daher in dieser Schrift nicht beschrieben. Falls Sie z.B. für künftige Programmentwicklungen Näheres über Multitasking, Macrotasking und Microtasking wissen wollen, wenden Sie sich bitte an Ihr Rechenzentrum oder direkt an das ZIB.

1.1.2 Speicherhierachie

Die Notwendigkeit, den Prozessor mit einem breiten, ungehinderten Datenstrom zu versorgen, spiegelt sich in der feinen Abstufung der Speicherhierarchie und der Transportwege der CRAY X-MP wieder:

Register

Die in dem umfangreichen Registersatz vorliegenden Daten können verzögerungsfrei und simultan vom Prozessor erreicht werden.

Hauptspeicher

Der Hauptspeicher der CRAY X-MP (4 Million Worte zu je 64 Bit) ist durch Überlappung und andere technische Maßnahmen so ausgelegt, daß nach einer gewissen Anlaufzeit pro CPU zweimal 105 Millionen Worte pro Sekunde aus dem Hauptspeicher in Vektorregister und einmal 105 Millionen Worte pro Sekunde vom Vektorregister in den Hauptspeicher geladen werden können.

E/A Subsystem

Durch einen separaten, der CPU vorgeschalteten Ein/Ausgaberechner wird die Zentraleinheit von allen E/A-Vorgängen entlastet. Das E/A-Subsystem bedient insbesondere die Plattenspeicher und die Vorrechner und puffert Daten in erheblichem Umfang, damit die Zentraleinheit möglichst selten durch Transporte unterbrochen oder behindert wird.

Plattenspeicher

An den Rechner CRAY X-MP sind die schnellsten derzeit auf dem Markt erhältlichen Plattenspeicher angeschlossen. Um auch hier eine hohe Transportrate zu erreichen, kann die Transportleistung aller Plattenspeicher simultan erbracht werden. Damit ist es möglich, durch Verteilung der zu einem Programmlauf gehörenden Dateien auf verschiedene Platten auch individuell Transportleistungen bis zu 50 MB/s abzufordern.

Vorrechner

Die letzte Stufe in der Speicherhierarchie bilden die Vorrechner mit den dort angeschlossenen Speichern. Bei den Vorrechnern handelt es sich um handelsübliche Universalrechner (Großrechner im herkömmlichen Sinne). Sie haben die Aufgabe, den Vektorrechner mit Aufträgen zu versorgen, Resultate entgegenzunehmen und weiterzuleiten und Daten zu speichern.

1.1.3 Verbindung mit dem Benutzer

Stations

Das Betriebssystem des Vektorrechners unterhält in jedem Vorrechner ein Station genanntes Teilsystem, welches die Kontaktstelle zwischen Vektorrechner und Benutzer darstellt. Die Station nimmt Batchjobs entgegen, liefert Resultate (Listen, Mitteilungen usw.) ab und vermittelt Dateien zwischen Vektorrechner und Vorrechner. Die Stationsoftware ist ihrem jeweiligen "Wirtssystem" angepaßt: unter NOS/BE z.B. kommuniziert sie unmittelbar mit den Ein-/Ausgabe-Queues des Betriebssystems, so daß der Rechner CRAY X-MP, abgesehen von der abweichenden Steuersprache, als Mitglied eines homogenen Rechnernetzes angesprochen wird.

Programmiertechnik

Die Einfachheit des Zugangs darf nicht darüber hinwegtäuschen, daß Programme, die auf dem Vektorrechner zeitgünstig laufen sollen, zweckmäßig gestaltet sein müssen. Insbesondere ist die Höchstleistung der CRAY X-MP nur mit speziellen Programmen erreichbar. Zwar ist die Maschine auch im nicht vektoriellen Bereich sehr leistungsfähig, jedoch stellen sich im Vergleich zu Universalrechnern erhebliche Verkürzungen der Programmlaufzeiten ein, wenn die Vektorverarbeitung im Programm überwiegt und wenn sie für den Vektorrechner geeignet formuliert ist. Unterstützung für die Formulierung geben spezielle Bibliotheken und Erweiterungen der Programmiersprache FORTRAN sowie die Literatur des Herstellers.

1.2 Konfiguration

1.2.1 CRAY X-MP

Die Rechenanlage CRAY X-MP ist wie folgt konfiguriert:

Zentraleinheit

Die Zentraleinheit CRAY X-MP verfügt über zwei Prozessoren, die mit jeweils 13 unabhängigen und segmentierten Funktionseinheiten ausgerüstet ist:

zwei für Adressrechnung:
- Addition (24-Bit)
- Multiplikation(24-Bit)

vier für skalare Grundoperationen:
- Addition (64-Bit)
- Shift
- Logische Operationen
- Population Count (Anzahl der auf 1 oder 0 gesetzten Bits zählen)

vier für vektorielle Grundoperationen:
- Addition (64-Bit)
- Shift
- Logische Operationen
- Population Count (Anzahl der auf 1 oder 0 gesetzten Bits zählen)

drei sowohl für skalare als auch vektorielle Gleitkommaoperationen:
- Addition
- Multiplikation
- Reziproke Approximation (statt echter Division)

fünf Arten von schnellen Registern:
- 8 Vektorregister (V) mit je 64 Worten a 64 Bit
- 8 Skalarregister (S) a 64 Bit

* 64 Skalarregister (T) a 64 Bit
* 8 Adressregister (A) a 24 Bit
* 64 Adressregister (B) a 24 Bit.

Die Funktionseinheiten holen die Operanden und speichern die Ergebnisse nur aus bzw. in die A-, S- und V-Register. Die B- und T-Register werden als schneller Zwischenspeicher für skalare Operationen verwendet.

Die interne Zykluszeit der CRAY X-MP beträgt 9.5 ns.

Die Hardware ist in ECL-bipolar-Technik realisiert; der Hauptspeicher besteht aus 4 Millionen direkt adressierbarer Worte zu je 64 Bit. Jede CPU verfügt über drei unabhängige Zugriffspfade zum Hauptspeicher; zwei dienen zum Laden von Vektoren, der dritte zum Speichern von Vektoren und zum Laden und Speichern von Skalaren. Der Speicher besteht aus 32 Bänken, die so verschränkt sind, daß adreßmäßig aufeinander folgende Worte in unterschiedlichen Bänken liegen. Jede Bank kann alle 38 ns (4 Zyklen) erneut angesprochen werden. Auf Grund der Verschränkung können daher Elemente von Vektoren in der Regel alle 9.5 ns (der Zykluszeit) geladen oder gespeichert werden.

Ein-/Ausgabesystem und Plattenspeicher

Die CRAY X-MP verfügt über ein Ein-Ausgabesystem, bestehend aus 2 Prozessoren (ein Master Ein-Ausgabe-Prozessor, ein Buffer Ein-Ausgabe-Prozessor) und eine Million Worte Pufferspeicher je 64 Bit. Das Ein-Ausgabesystem dient insbesondere der Steuerung der Plattenlaufwerke und der Kommunikation mit den Vorrechnern.

Als Hintergrundspeicher dienen vier Plattenlaufwerke DD-29, Speicherkapazität je 600 MB, Transferrate 4.5 MB/s sowie drei Plattenlaufwerke DD-49, Speicherkapazität je 1,2 GB, Transferrate 11 MB/s.

Die Anlage CRAY X-MP verfügt (abgesehen von einem angeschlossenen Wartungsrechner mit Drucker und Magnetbandstation) nicht über Peripheriegeräte wie Magnetbandgeräte, Drucker, Kartenleser oder Sichtgeräte.

Die Verbindung mit der Außenwelt geschieht ausschließlich über Kanalverbindungen zwischen dem Ein-Ausgabesystem und den Vorrechnern CD CYBER 170-825 (NOS/BE-System), und Siemens 7.865 (MVS-System). Die theoretische Transferrate der Kanäle beträgt CRAY-seitig je 6 MB/s.

1.2.2 Der Vorrechner CD CYBER 170-825

* Zentraleinheit (CPU) vom Typ 825, 20 periphere Processoren (PPU's), 2M Worte Hauptspeicher je 60 Bit (MOS-Technik) und 24 Kanäle mit Übertragungsraten bis 4 MB/s

* 4 Festplattendoppellaufwerke CD 885-12 mit 2 Steuereinheiten CD 7155-11, die unabhängig auf jedes Laufwerk zugreifen können; Speicherkapazität je Doppellaufwerk 1,4 GB, Übertragungsrate 1,6 MB/s

* 5 Wechselplattenlaufwerke CD 844-41 mit zwei Steuereinheiten CD 7154-1, die unabhängig auf jedes Laufwerk zugreifen können; Speicherkapazität je Laufwerk 237 MB, Übertragungsrate 1.1 MB/s

* 2 Magnetbandstationen CD 669-4 (9-Spur) mit einer Schreibdichte von 800 bzw. 1600 Bit pro Zoll einschließlich einer Steuereinheit CD 7021-22 mit einer Übertragungsrate von 240 KB/s (bei 1600 bpi);

* 2 Magnetbandstationen CD 679-7 (9-Spur) mit einer Schreibdichte von 1600 bzw. 6250 Bit pro Zoll einschließlich einer Steuereinheit CD 7021-31 mit einer Übertragungsrate von 1,25 MB/s (bei 6250 bpi)

* 1 Zeilendrucker

* 2 Rechner ATM CLASSIC 7830 zum Anschluß der CYBER 825 an das Deutsche Forschungsnetz (DFN).

Die CYBER 825 wird mit einer für die Technische Universität Berlin im ZIB betriebenen CYBER 175 als Multimainframe-System mit gemeinsamer Magnetplatten- und Magnetbandperipherie bereit gestellt; die aufgeführten Peripheriegeräte werden zum Teil auch von der CYBER 175 genutzt.

1.2.3 Der Vorrechner Siemens S 7.865

Diese Anlage ist wie folgt konfiguriert:

* Zentraleinheit (CPU) SIEMENS 7.865 (Leistungsklasse 2-2,4 MIPS) mit 16 MB Hauptspeicher in MOS-Technik, zwei Datenstromkanälen (Übertragungsrate drei MB/Sek), fünf Blockmultiplexkanälen (Übertragungsrate 2 MB/Sek) und einem Bytemultiplexkanal

* vier Festplattenspeicher IBM 3380 (Kapazität je 2,5 GB) an einer Steuerung IBM 3880-23, angeschlossen an zwei Datenstromkanäle

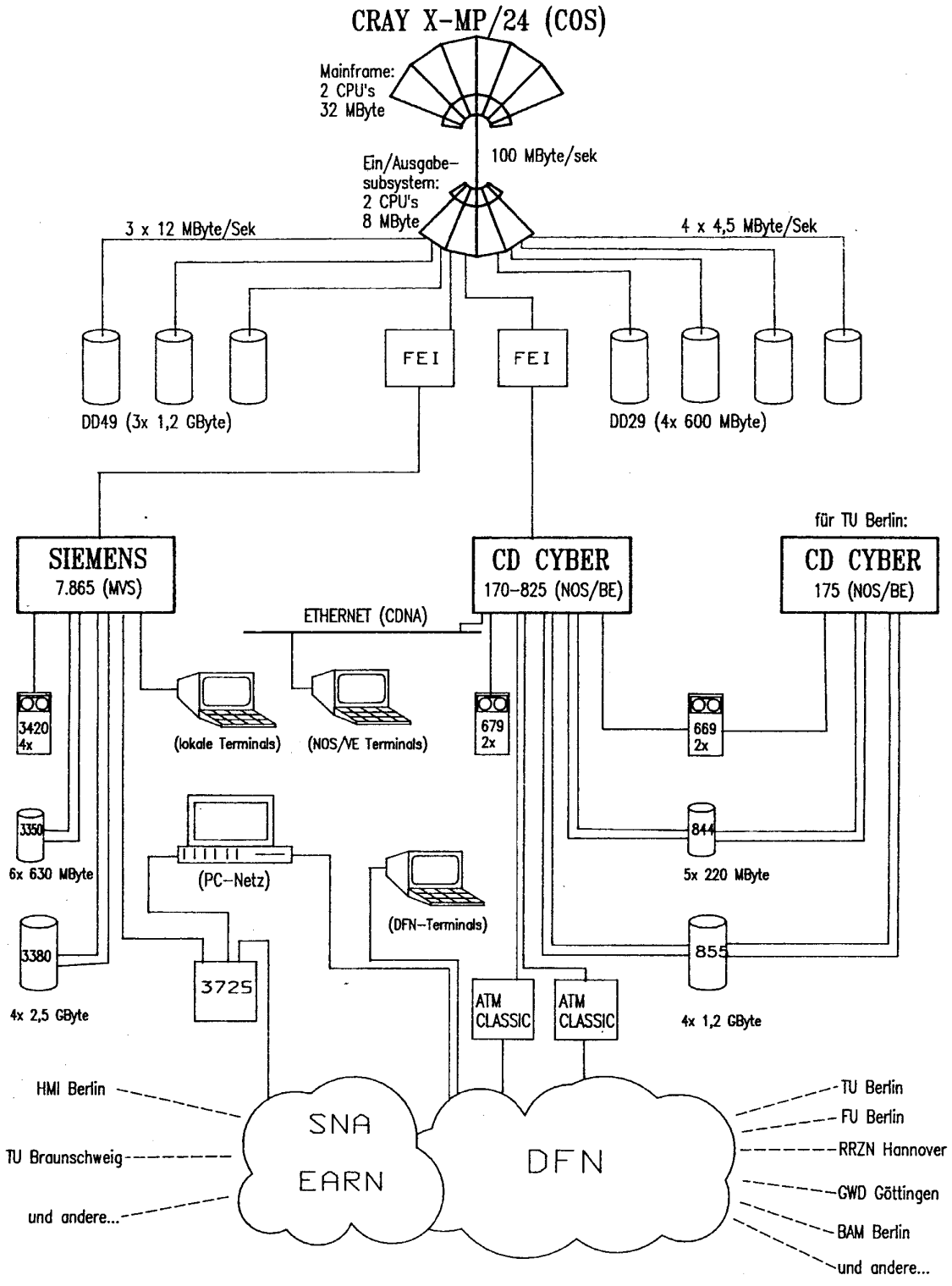
* sechs Festplattenspeicher-Doppellaufwerke IBM 3350 (Kapazität je 630 MB) an einer Steuerung IBM 3880-1, angeschlossen an zwei Blockmultiplexkanäle

* vier Magnetbandstationen IBM 3420 (9-Spur) mit einer Schreibdichte von 1600 bzw. 6250 Bit pro Zoll an einer Steuerung IBM 3803

* ein Datenübertragungsvorrechner IBM 3725 zum Anschluß an DATEX-P, das Europäische Forschungsnetz (EARN/BITNET), sowie verschiedene Stand- und Wählverbindungen

* Lokale Terminals, über Cluster IBM 3274 direkt an die Zentraleinheit angeschlossen

Konrad-Zuse-Zentrum für Informationstechnik Berlin
Rechnerkonfiguration



ZIB/Bereich Anlagenbetrieb/15.4.1987

Stand Anfang 1987

1.3 Einbindung der CRAY in Rechnernetze

1.3.1 Deutsches Forschungsnetz (DFN)

Die Anlage CRAY X-MP ist über den Vorrechner CYBER 825 an das Deutsche Forschungsnetz (DFN) angeschlossen. Das DFN arbeitet im wesentlichen mit den gleichen Protokollen, die vielen Benutzern unter dem Begriff BERNET schon bekannt sind; insbesondere ist der Benutzerzugang gleich geblieben: DFN verwendet lediglich in der Transportebene die T70-Protokolle anstelle von Message-Link bei BERNET. Die Verbindung zum Netz ist über zwei Rechner ATM Classic 7830, Kanalkopplung zur CYBER 825 und über X.25-Untervermittlungen realisiert. Die Siemens 7.865 MVS-Anlage verfügt zur Zeit nur über den aktiven und den passiven Dialogdienst.

Im Rahmen des DFN werden folgende Dienste angeboten:

Remote Job Entry (RJE)

Der RJE-Dienst überträgt Ein- und Ausgabedateien zwischen den am Verbundsystem angeschlossenen Rechenanlagen. Die von einem Job erzeugte Ausgabedatei wird zu einer der angeschlossenen Rechenanlagen übertragen; im Regelfall ist dies die Anlage, von der aus der Job abgeschickt wurde.

Einzelheiten über das Absenden eines Batchjobs von einer dieser Anlagen erfahren Sie beim jeweiligen Rechenzentrum. Für das Absenden von einer NOS/BE-Anlage finden Sie alle Informationen unter DOC,BERNET,RJE; für das Absenden von anderen Systemen her finden Sie die für die CYBER 825 notwendigen Informationen unter DOC,BERNET,CDC. Erreichbar ist die CYBER des ZIB über die -

- DFN-Adresse: 'B ZIB01' (die zwei Leerzeichen hinter B sind signifikant!);
- Datex-P-Adresse: '45300020301' bzw. eine Kurzkennung, die Ihr Rechenzentrum vergeben hat (z.B. 'ZIB').

File Transfer Dienst

Mit Hilfe des FT-Dienstes können codierte (und zum Teil auch binäre) Dateien von einem entfernt stehenden Rechner geholt oder zu diesem gebracht werden. Es gelten für die CYBER 825 die gleichen Adressen wie für RJE. Weitere Informationen erhalten Sie unter DOC,BERNET,FT.

Passiver Dialog

Unter passivem Dialog versteht man die Möglichkeit, den betreffenden Rechner von außen her über die von der Deutschen Bundespost eingesetzten X.28/X.29-Protokolle anzusprechen. Der Benutzer kann von folgenden Geräten aus auf die ZIB-Rechner zugreifen:
Terminals an einem Package Assembler Disassembler (Hardware-PAD), die direkt an eine Untervermittlung des Berliner Rechnernetzes BERNET angeschlossen sind;
Terminals an einem Rechner, der über den aktiven Dialog (Software-PAD) verfügt;
Terminals, die direkten Zugang zum DATEX-P-Dienst der Deutschen Bundespost besitzen.

Sowohl die CYBER 825 als auch die MVS-Anlage sind über den passiven Dialog von außen ansprechbar:

CYBER 825 (NOS/BE)	10260005	(interne BERNET-Adr.)	45300020305 (Datex-P-Adr.)
	10260203	(interne BERNET-Adr.)	
bei Ausfall einer ATM Classic:			
	10260001	(interne BERNET-Adr.)	45300020301 (Datex-P-Adr.)
	10260201		
Siemens 7.865 (MVS)	10260400	(interne BERNET-Adr.)	45300020340 (Datex-P-Adr.)

Aktiver Dialog

Unter aktivem Dialog (Software-Pad) versteht man die Möglichkeit, von einem am eigenen Rechner installierten Dialoggerät über diesen Rechner den X.25-Ausgang und über das DATEX-P-Netz der Deutschen Bundespost (bzw. Standleitungen) den passiven Dialog eines entfernten Rechners zu verwenden. Fragen Sie in Ihrem Rechenzentrum nach, ob Ihr Rechner über den aktiven Dialog verfügt.

Rechner im DFN:

Innerhalb des Norddeutschen Vektorrechnerverbundes sind folgende Rechenanlagen im DFN erreichbar (Quelle: DFN-Informationssystem, Stand März 1987):

Institution	Rechner	Dienste	DFN-Adresse
Berlin:			
Bundesanstalt für Materialprüfung	VAX 11-780	D	
Bundesanstalt für Materialprüfung	VAX 8600	R F D	B BAM03*
Deutsches Bibliotheks-Institut	S 7.570	R D	B DBI05
Freie Universität Berlin:			
- Fachbereich Chemie	VAX 11-750	R F D	B FKR01*
- Zentraleinr. Datenverarbeitung	CDC 170-845	R F D	B FUB03*
- Zentraleinr. Datenverarbeitung	S 7.550	R D	B FUB05*
Fachinformationszentrum Chemie	S 7.531	R D	B FIZ01
Fritz-Haber-Institut	CDC 170-830	R D	B TUB01*
Heinrich-Hertz-Institut	S 7.531	R D	B HHI01*
Hahn-Meitner-Institut	S 7.890	F D	B HMI02
Hahn-Meitner-Institut	VAX 11/785	F D	B HMI13
Hahn-Meitner-Institut	VAX 11/780	F D	B HMI21
Hahn-Meitner-Institut	VAX 11/750	F D	B HMI20
Technische Universität Berlin:			
Zentraleinrichtung Rechenzentrum	CDC170-835	R D	B TUB01*
Zentraleinrichtung Rechenzentrum	CDC180-830	R D	B TUB01*
Zentraleinrichtung Rechenzentrum	S 7.551	D	B TUB05*
Iwan N. Stranski-Institut	PCS Qu 68000	F D	B INS01*
Institut für Bauingenieurmethoden	S 7.551	D	B TUB07*
Niedersachsen:			
Regionales Rechenzentrum Hannover	CDC 180-990	R	H RRZ02*
Regionales Rechenzentrum Hannover	CDC 180-990	R	H RRZ03*
Universität Göttingen	SPERRY	R D	GO GWD01
Schleswig-Holstein:			
Institut für Meereskunde Kiel	VAX 11-750	R F	KI UNI80
Rechenzentrum Universität Kiel	VAX 730	R F D	KI UNI04
Rechenzentrum Universität Kiel	S 7.760	R F D	KI UNI05

Erläuterung: * bedeutet, daß der entsprechende Rechner mit den Rechnern des

ZIB über Standleitungen in Verbindung steht; bei der Benutzung fallen also für den Benutzer keine Übertragungsgebühren an.
* bedeutet, daß der betreffende Anschluß in Vorbereitung ist. Weitere im DFN erreichbare Rechner sind in der Beschreibung DOC,BERNET,RECHNER aufgeführt.

Die DFN-Adresse dient zur eindeutigen Bezeichnung der Datenverarbeitungsanlage. Die ersten drei Zeichen stellen eine Gebietsbezeichnung dar, die von der Kennzeichnung für Kraftfahrzeuge übernommen worden ist; die Leerzeichen in dieser Adresse sind daher signifikant. Die folgenden drei Zeichen bilden eine Institutskennung, während die letzten zwei Ziffern die Unterscheidung zwischen verschiedenen Datenverarbeitungsanlagen des Instituts ermöglichen.

Bei Benutzung des Programms RJE besteht für den Verkehr zwischen CDC-Anlagen unter NOS/BE die Möglichkeit, Kurzkennungen von drei Zeichen Länge anzugeben. Diese können direkt in den Parametern DO= bzw. ST= des RJE-Programms angegeben werden.

1.3.2 European Academic and Research Network (EARN)

EARN ist ein Rechnernetz, das eine Reihe von Rechnern wissenschaftlicher Einrichtungen in Europa untereinander verbindet. Die Kommunikation dieser Rechner basiert überwiegend auf Software des IBM-Leitungsprotokolls BSC sowie den Protokollen der IBM NJE (Network Job Entry), NJI (Network Job Interface) bzw. RSCS (Remote Spooling and Communication Subsystem). An EARN sind neben IBM-Rechnern auch eine größere Anzahl anderer Rechner angeschlossen, auf denen eine Emulation der oben genannten Protokolle im Einsatz ist.

EARN kann zu folgenden Zwecken eingesetzt werden:
Senden von Dateien und Nachrichten an einen Benutzer eines anderen Rechners innerhalb des Netzes;
Senden von Jobs an einen an EARN angeschlossenen Rechner.

Ein Dialog-Betrieb über die EARN-Verbindung ist nicht möglich.

Der MVS-Rechner des ZIB ist zu erreichen über die EARN-Adresse

DBOZIB21

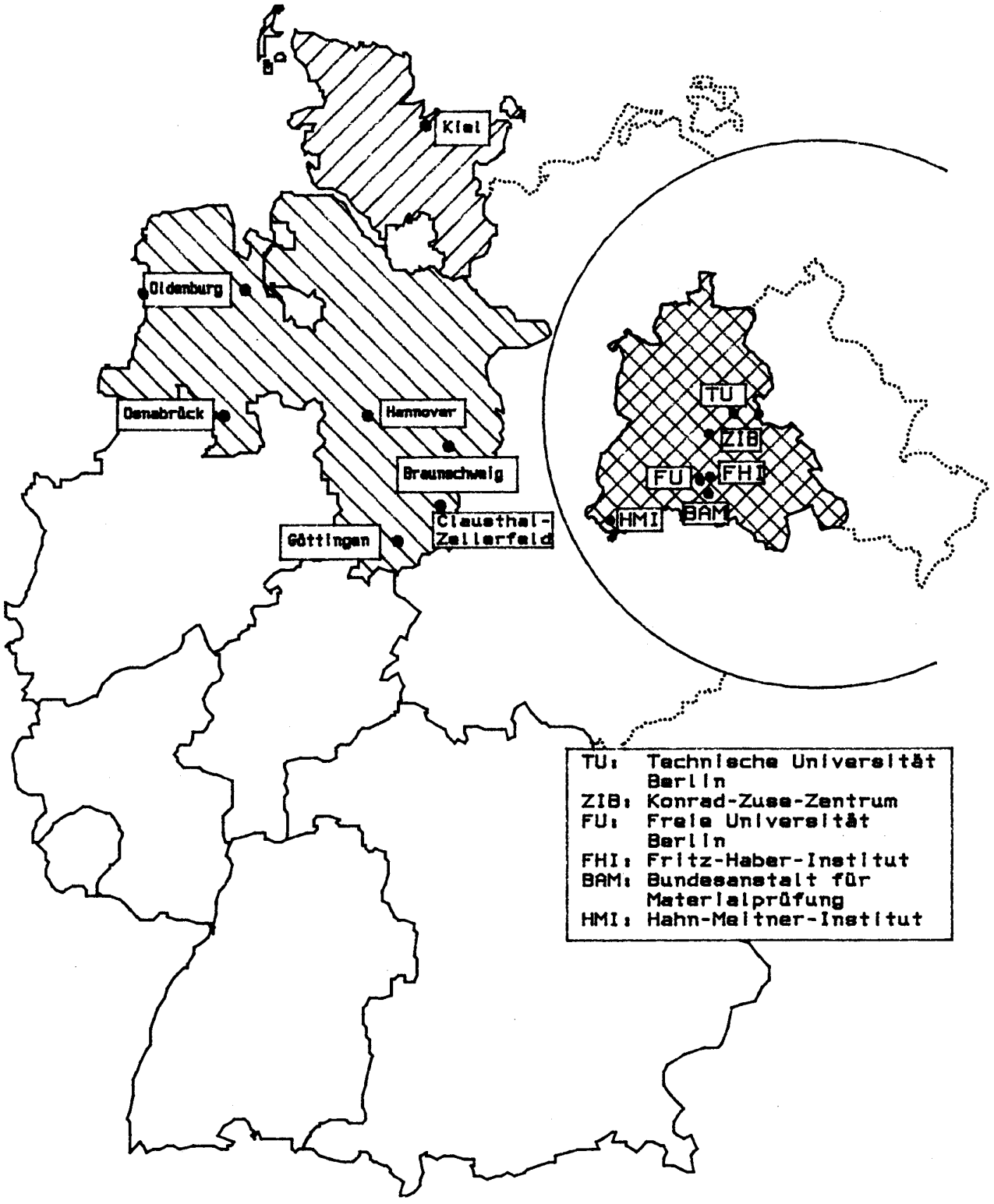
Innerhalb des Norddeutschen Rechnerverbundes stehen folgende Rechner mit EARN in Verbindung:

Institution	Rechner	Betriebssystem	EARN-Adresse
Berlin:			
Fritz-Haber-Institut	CDC	NOS/BE	DBOFHI01
FU Berlin, ZEDAT	CDC	NOS/BE	DBOFUB03
Hahn-Meitner-Institut	SIEMENS	MVS	DBOHMI41
TU Berlin			
- Informatik	IBM	VM	DBOTUI11
- Informatik KBS	UNIX 4.2	DBOTUI6	
- Maschinenbau	IBM	VM	DBOTUM11
- Schiffs- und Meerestechnik	IBM	VM	DBOTUS11
- ZRZ	CDC	NOS/BE	DBOTUZ01

Institution	Rechner	Betriebssystem	EARN-Adresse
Niedersachsen:			
GBF Braunschweig	DEC	VMS	DBSGBF5
Uni Braunschweig		ULTRIX-32	DBSINF6
Uni Braunschweig	IBM	VM/SP	DBSTU1
Uni Clausthal	CGK	BS3	DCZRZTU0
DFVLR Braunschweig	IBM	VM/SP	DFVLRBS1
DFVLR Braunschweig	IBM	MVS/SP	DFVLRBS2
DFVLR Göttingen	IBM	VM/SP	DFVLRG01
DFVLR Göttingen	IBM	MVS/SP	DFVLRG02
GWD Göttingen	SPERRY	OS 1100	DGOGWD01
Med. Hochschule Hannover	IBM	VM/SP	DHVMHH1
Universität Hannover: Institut für Fertigungstechnik		VM/SP	DHVIFW1
Regionales Rechenzentrum für Niedersachsen	CDC	NOS	DHVRRZN0
Regionales Rechenzentrum für Niedersachsen		VM/SP	DHVRRZN1
Hochschule Hildesheim	IBM	VM/SP	DHIURZ1
Universität Oldenburg		VM/SP	DOLUNI1
Universität Osnabrück	CGK	BS 3	DOSUNI
Schleswig-Holstein:			
Uni Kiel	DEC 10	TOPS 10	DKIUNI0
GKSS Forschungszentrum, Geesthacht		BS 3000	DGHGKSS4

Quelle: EARN. Pocket Reference Summary. Third Edition, May 1986. Weitere Einzelheiten erhält man durch DOC,EARN,EARN.

Norddeutscher Vektorrechnerverbund



1.4 Benutzerzugang zur CRAY X-MP

Mit den Berliner Universitäten, dem Hahn-Meitner-Institut für Kernforschung Berlin GmbH und den Ländern Schleswig-Holstein und Niedersachsen bestehen Kooperationsvereinbarungen über die Benutzung des Vektorrechners im ZIB einschließlich seiner Vorrechner. Im Rahmen dieser Kooperationsvereinbarungen wurde festgelegt, daß für die Zulassung und Betreuung von Benutzern aus den Bereichen dieser Kooperationspartner jeweils diese Partner zuständig sind. Das ZIB selbst läßt Benutzer aus Wissenschaft und Industrie zu, wenn dies mit seinen eigenen Forschungsprojekten vereinbar ist. Rechnererlaubnisse erteilen im Rahmen der oben genannten Kooperationsvereinbarungen folgende Stellen (==> 1.9):

für Berlin: Zentraleinrichtung für Datenverarbeitung der Freien Universität Berlin; Zentraleinrichtung Rechenzentrum der Technischen Universität Berlin; Bereich Datenverarbeitung/Mathematik des Hahn-Meitner-Instituts Berlin.

für Niedersachsen: Regionales Rechenzentrum Niedersachsen in Hannover sowie die Rechenzentren der übrigen niedersächsischen Hochschulen;

für Schleswig-Holstein: Universität Kiel - Rechenzentrum;

für Projekte mit dem ZIB: Konrad-Zuse-Zentrum für Informationstechnik, Berlin.

Jeder interessierte Benutzer kann bei einem für ihn zuständigen Rechenzentrum auf Antrag und nach Prüfung eine Erlaubnis zur Zulassung zum Rechner CRAY X-MP sowie für einen oder beide Vorrechner erhalten. Bei Zulassung erhält der Benutzer eine Abrechnungsnummer und ein Kennwort für jeden Rechner. Antragsformulare stehen in den Rechenzentren zur Verfügung.

Abrechnungsnummern und Kennworte sollten zum Schutz vor Mißbrauch geheimgehalten werden. Festgestellter Mißbrauch ist dem eigenen Rechenzentrum und dem ZIB umgehend mitzuteilen. Das Ausscheiden eines Benutzers aus dem Institut ist dem Rechenzentrum anzuzeigen; ohne Genehmigung der Rechenzentren sind Benutzernummern nicht auf andere Personen übertragbar. Die Nutzungsberechtigung gilt in der Regel für ein Kalenderjahr, kann aber jeweils um ein weiteres Jahr verlängert werden.

1.5 Entgelte

Für die Benutzung der CRAY X-MP gelten die Entgeltordnungen der zulassenden Rechenzentren. Zur Zeit (Anfang 1987) werden von Angehörigen der Universitäten in Berlin, Niedersachsen und Schleswig-Holstein noch keine Entgelte erhoben.

Bis zum Erlaß einer Entgeltordnung des ZIB gelten folgende Sonderregelungen für sonstige Benutzer:

Benutzer des Vektorrechners entrichten Entgelte für die verbrauchte Ressource-Zeit, nämlich

- Industriepartner des ZIB	6.000,- DM pro Stunde (zuzüglich MWST)
- Wissenschaftliche Einrichtungen außerhalb Berlins, Niedersachsens und Schleswig-Holsteins	1.000,- DM pro Stunde (zuzüglich MWST)

In diesen Entgelten ist die Benutzung des jeweiligen Vorrechners inbegriffen. Eine Nutzung der Vorrechner zu anderen Zwecken als zum Zugang zum Vektor-

rechner ist nur im Rahmen von Forschungsprojekten des ZIB vorgesehen und muß im Einzelfall gesondert geregelt werden.

1.6 Zugangszeiten zur CRAY X-MP

Die CRAY X-MP sowie beide Vorrechner werden grundsätzlich rund um die Uhr betrieben; dabei

Montags - Freitags 06.30 - 21.30 Uhr

mit Bedienung durch Operateure, außerhalb dieser Zeit ohne Bedienung durch Operateure. Bandgeräte können innerhalb des bedienten Betriebes nur bis 21.00 Uhr benutzt werden.
Regelmäßig findet zu folgenden Zeiten **kein** Benutzerbetrieb statt:

CRAY X-MP:

Montags	07.00 - 11.00 Uhr
Mittwochs	15.00 - 17.00 Uhr
Freitags	07.00 - 09.00 Uhr

Cyber 825:

am ersten Montag im Monat:	
	07.00 - 11.00 Uhr
bei Bedarf werktäglich:	
	08.00 - 09.00 Uhr

MVS-Anlage:

Montags	06.30 - 09.00 Uhr
Dienstags	06.30 - 09.00 Uhr
Freitags	07.30 - 08.45 Uhr

Darüber hinausgehende absehbare Einschränkungen werden im Kopf jeder Jobliste (Header) sowie beim Beginn des Dialogs (Startup-Prozedur) den Benutzern mitgeteilt. Der aktuelle Betriebszustand kann über den Anrufbeantworter des ZIB, Rufnummer (030) 89604 167 abgefragt werden.

1.7 Dokumentation

1.7.1 Literatur zur CRAY

Betriebssystem COS

Bestell-Nummer	Titel	Preis in DM
SR-0011 N	CRAY-OS (COS) Version 1 Reference Manual Beschreibt die Benutzeroberfläche des Betriebssytems COS der CRAY (COS Version 1.15)	84,90
	Nachdruck des ZIB (einschl. Update R.F. SR-0013 in der Version für COS 1.13)	12,00
SR-0013 F	UPDATE Reference Manual Nachdruck ist im COS-Manual Nachdruck enthalten (siehe oben) (COS 1.13 !)	45,50

SR-0039 D	CRAY-OS (COS) Message Manual Beschreibt alle Botschaften des CRAY-OS, die im Outputlisting und im "User Logfile" erscheinen.	51,50
	Nachdruck des ZIB (COS 1.14 !)	7,00
SR-0066 B	Segment Loader (SEGLDR) Reference Manual	47,00
SR-0112	Symbolic Debugging Package Ref. Man. (COS Version 1.15)	33,65

Sprachen

SR-0009 L	CRAY FORTRAN (CFT) Reference Manual Beschreibt den CRAY FORTRAN Compiler und damit zusammenhängende Betriebssystem-funktionen	62,15
SR-0009 L-01	Change Packet	19,90
K	Nachdruck des ZIB (CFT 1.14 !)	14,00
SQ-0021 F	COS/CFT Reference Card	2,75
SR-0113 I	Programmer's Library Reference Manual Beschreibt die vom Hersteller bereitge-stellten Unterprogramme für Benutzer	94,75
SR-0014 I	Nachdruck des ZIB (COS 1.14 !)	19,00
SR-0000 K	CAL Assembler Version 1 Reference Manual Beschreibt den CRAY Assembler	47,50
SR-0060 B	PASCAL Reference Manual (3.0)	59,00
SR-2024	CRAY C Reference Manual Beschreibung der Programmiersprache C auf der CRAY	44,90

Stations

SR-0034 D	CDC NOS/BE Station Reference Manual (Version 1.12) Beschreibt die Leistungen der Kopplungssoftware zwischen CRAY und dem Vorrechner CD CYBER 825	12,50
	Change Packet	
SR-0034 D-01	1. Update zu Rev. D (Version 1.13)	9,25
SR-0034 D-02	2. Update zu Rev. D (Version 1.14)	6,75
SI-0038 D	IBM MVS Station Reference Manual (Version 2.01) Beschreibt die Leistungen der Kopplungssoftware zwischen CRAY und dem Vorrechner IBM	35,10
SI-0108 B	IBM MVS Station Message Manual (Version 2.01) Beschreibt alle Botschaften der MVS-Station-Software	35,00

CRAY Computer Systems Technical Notes

SN-0220 B	CRAY-1 Optimization Guide Enthält zahlreiche Beispiele zur Optimierung und Vektorisierung von Programmen	17,75
SN-0203 B	Complex Fast Fourier Transform Binary Radix Subroutine (CFFT2) Diese Schrift enthält wie die folgenden die Beschreibung der entsprechenden Bibliotheksroutinen	2,50
SN-0204 B	CRAY-1 Real to Complex Fast Fourier Transform Binary Radix Subroutine (RCFFT2)	2,50
SN-0206 B	CRAY-1 Complex to Real Fast Fourier Transform Binary Radix Subroutine (CRFFT2)	2,50
SN-0210 A	CRAY-1 Linear Digital Filters for CFT Usage	2,50
SN-0214 A	Arbitrary Spaced Fast Matrix Multiply (MXMA)	2,75
SN-0216 A	PACK/UNPACK - Packed Word Routines	2,50

Die wichtigsten Manuals sind vom ZIB preisgünstig nachgedruckt worden, allerdings nur in z. Zt. nicht mehr aktuellen Versionen. Für Standard-Anwendungen reichen diese Nachdrucke jedoch noch aus. Für die Nachdrucke gibt es keine Ergänzungslieferungen. Es wird keine Mehrwertsteuer erhoben. Bestellungen sind zu richten an die

Verwaltung des
Konrad-Zuse-Zentrum für
Informationstechnik Berlin (ZIB)
Heilbronner Straße 10
1000 Berlin 31

Die oben aufgeführte Originalliteratur dürfte in Ihrem Rechenzentrum vorhanden sein. Wir empfehlen, zunächst dort oder im ZIB das betreffende Manual anzusehen, wenn Sie Fragen haben, die über das in diesem Handbuch Ausgeführte hinausgehen. Sie können die Schriften auch direkt von

CRAY RESEARCH GMBH
Geschäftsbereich Software
z.H. Frau Claudia Mayer
Kistlerhofstr. 168
8000 München 70
Telefon: (089) 78590-0 (Direktwahl: (089) 78590-131)

beziehen. Die angegebenen Preise wurden uns mit Stand von März 1987 von CRAY mitgeteilt, für ihre Richtigkeit können wir keine Gewähr übernehmen. Der zweite Teil der Bestellnummer (z.B. J-02) bezeichnet jeweils den Änderungsstand des genannten Handbuchs. Bei den Original-Handbüchern wird zusätzlich zu den angegebenen Preisen die Mehrwertsteuer erhoben.

Die aufgeführten Handbücher gehören, sofern nicht im einzelnen anders vermerkt, zu COS 1.15 bzw. CFT 1.15 . Die Technical Notes werden als wichtige Ergänzung der grundlegenden Handbücher für die Lösung spezieller, aber häufig vorkommender Probleme angesehen.

Literatur zur CD CYBER 825 (ebenfalls bei der ZIB-Verwaltung erhältlich):

Titel		Preis (DM) incl. MWST
- FORTRAN 77	Einführung. 3. verbesserte Aufl., 1.1981 Darstellung der wesentlichen Teile des neuen Standards. Autor: H. Wehnes (GHS Wuppertal).	5,50
- FORTRAN 77	Sprachumfang. Ein Nachschlagewerk. 1. Aufl., 2.1983 Interpretierende Übersetzung des FORTRAN 5 Manuals der Firma CDC; auch geeignet als Einführung in den CRAY-Compiler CFT. Herausgeber: RRZN, Universität Hannover.	6,00
- NOS/BE	Benutzeranleitung für das Betriebssystem. 4. aktualisierte und korrigierte Aufl., 1986 Beschreibung der wichtigsten Möglichkeiten des CD- Betriebssystems NOS/BE. Autoren: H.Hoffmann, D.Kehl, H.-J.Mildner, A.Preusser, R.Riedel, S.Schwenkler. Herausgeber: ZRZ der TU Berlin und ZIB.	8,00
- INTERCOM	Instant. 4. überarb. Aufl., 3.1985 Beschreibung aller wichtigen Kommandos im Dialogbetrieb (insbes. FAMILY, TUBE, PAGE). Herausgeber: ZRZ der TU Berlin .	2,00

Die oben aufgeführten Handbücher sind zumeist in den Rechenzentren erhältlich und auch für den Vorrechner CD 170-825 gültig.

Unabhängig davon kann Einblick in die Originalhandbücher der Fa. Control Data ("CD-Manuals") gewährt werden; diese stehen in den Beratungsräumen der Rechenzentren zur Verfügung. Z. T. können diese Handbücher auch ausgeliehen werden.

1.7.2 ZIB-Dokumentationsystem DOC:

Weitere, insbesondere aktuelle Informationen erhält man auf beiden Vorrechnern der CRAY über das Online-Informationssystem DOC. DOC kann an der NOS/BE-Anlage sowohl im Dialog als auch im Stapelbetrieb aufgerufen werden mit

```
DOC,cat,item,L=list,MO=mode,PS=ps,PL=pl,REV=rev.
```

DOC kann an der MVS-Anlage im Dialog bequem im ISPF-Menue 1.5 aufgerufen werden oder direkt durch

```
DOC,cat,item,L(list),MO(mode),PL(pl),REV(rev)
```

oder auch im Stapelbetrieb durch

```
stpname EXEC DOC,cat,item,MO(mode),PL(pl),REV(rev)
```

aufgerufen werden. Insbesondere für den über EARN arbeitenden Benutzer steht die Prozedur DOC zur Verfügung, mit der man eine gewünschte Schrift direkt einem über EARN erreichbaren Rechner senden kann:

```
//      JOB ... ,MSGCLASS=Z, ...
/*ROUTE PRINT DBOZIB21
/*OUTPUT A001 DEST=knoten.user
//      EXEC DOC,CAT=cat,ITEM=item,
//           NODE=knoten,USERID=user
//
```

Parameter: (die Voreinstellung (default) ist unterstrichen; die kursive Schreibweise gilt für die MVS-Anlage)

cat — Category: Sachgebiet, über das der Aufrufende informiert werden möchte.

item — Item: Begriff aus dem o.g. Sachgebiet, über den der Aufrufende informiert werden möchte. Fehlt die Angabe, so erhält er die zum Sachgebiet gehörende Schrift GENERAL, die einen Überblick über das Sachgebiet liefert, oder aber die Schrift INDEX mit einem Inhaltsverzeichnis dieses Sachgebietes. Mehrere Begriffe können durch '/' getrennt in einem Aufruf von DOC angefordert werden.

=* — Information über alle DOCs des Sachgebietes.

L=list
L(list)
=*
(*) — Angabe des Namens der Datei, in die die DOC-Ausgabe erfolgen soll.
Ausgabe auf dem Zentraldrucker. Ohne Angabe dieses Parameters erscheint die gewünschte Schrift bei Aufruf im Dialog auf dem Bildschirm, im Stapelbetrieb in der Ausgabeliste.

MO=mode
MO(mode)
mode:
- TITLE — Titelfeldes
- SHORT — Kurzfassung
- LONG — ausführlichen Dokumentation

Ausgabe des/der:

PS=ps
=11 — Physical Size: Länge der Seite in Zoll

PL=pl
=50 — Print Limit: Anzahl der auszudruckenden DOC-Seiten, falls >50

REV=rev
REV(rev) — Ausgabe nur der DOCs, die nach dem Stichtag tt/mm/jj verändert wurden.

Sachgebiet CRAY: Einige für die Benutzung der CRAY wichtige Dokumente (Stand März 1987)

ADILIB	Routinen zur Optimierung von ADINA-Ein/Ausgabevorgängen
BERNET	Die CRAY im Rechnernetz (Post, RJE, Transfer)
COS114	Besonderheiten des Betriebssystems COS 1.14
COS115	Besonderheiten des Betriebssystems COS 1.15
EARNJOBS	EARN - Zugang zur CRAY
FIDISOL	Lösungen von part. und gewöhnl. Differentialgl.
FORSIM6	Lösungen von part. und gewöhnl. Differentialgl.
GAUS82	Benutzerinformation über GAUSSIAN 82
GRIPS	Grafiksoftware GRIPS der ZRZ der TUB an der CRAY
HEADER	Aktueller Header der CRAY
IMSL	Besonderheiten der IMSL Installation an der CRAY
IO	Hinweise zur Anwendung von Ein-Ausgabe-Methoden an der CRAY
KONFIGUR	Konfiguration der CRAY

LISP	LISP unter COS
MSIO	Effektive Ein-Ausgabe-Routinen für Dateien mit großen Satzlängen
MVSCRAY	Zugang zur CRAY für MVS-Benutzer
NAG	Besonderheiten der NAG Installation an der CRAY
NEWS	Aktuelle Informationen und Fehlermeldungen zur CRAY
NOSBECRY	Kopplung/Transfer zwischen CD 825 und CRAY
NUTZUNG	Nutzung der CRAY im letzten Monat
PASCAL	PASCAL 3.0 für die CRAY
PRIORITY	Definition der Jobklassen auf dem CRAY-Rechner
REDUCE	REDUCE 3.2 für CRAY
SCILIB	Hochoptimierte CRAY-Bibliotheksroutinen (==> 9.5.1), lauffähig auch auf den NOS/BE-Anlagen!
SOFTWARE	Hinweise auf vorhandene und beschaffbare Software
STATION	Kopplung/Transfer zwischen Vorrechnern und CRAY
VEKPACK	Iterative Lösung großer linearer Gleichungssysteme
VORJAHR	Nutzung der CRAY im Vorjahr (vergl. NUTZUNG)
XIO	Routinen zum Ersetzen sequentieller E/A durch E/A mit direktem Zugriff zur Effizienzsteigerung

Sachgebiet COS: Beschreibung der COS-Kommandos sowie wichtiger CRAY-Programme

Dieses Sachgebiet wurde direkt von der Firma CRAY übernommen und inhaltlich nicht überarbeitet. Leichte Abweichungen zu der im ZIB aktuell eingesetzten Betriebssystemversion sind möglich. Maßgeblich ist das COS Reference Manual der jeweiligen Version und des ZIB.

Sachgebiet CFT: Gibt Informationen zum CRAY FORTRAN Compiler CFT und seiner Einbettung in das Betriebssystem COS.

Sachgebiet CRUTIL: Beschreibungen von Dienstprogrammen, die nicht in Standardbibliotheken vorliegen bzw. als Ergänzungen des COS am ZIB existieren.

Sachgebiet DFUE: Zugang zum ZIB über Netze und für Dialoggeräte.

TELEFON	DFUE - Zugang zu den Rechnern des ZIB für Dialoggeräte Telefonanschlüsse

Sachgebiet BERNET: Einbindung der CYBER 825 in BERNET und DFN, Dienste: Remote Job Entry, passiver Dialog, aktiver Dialog, Filetransfer.

Sachgebiet DFN: Einbindung der MVS-Anlage in das DFN; Dienst: aktiver Dialog

Sachgebiet EARN: Einbindung der MVS-Anlage in EARN, Dienste: Remote Job Entry, Filetransfer, Mail-Service.

Sachgebiet ORGANISAT: Allgemeine Informationen für die Anlagen des ZIB, u.a:

ADRESSEN	Anschriften der Partnerrechenzentren
BETRIEB	Übersicht über den Bereich Anlagebetrieb des ZIB
DATENSCH	Datenschutzregelungen
ENTGELT	Entgeltordnung für die Rechenanlagen des ZIB
FREMDMT	Organisatorische Regelungen für Fremdbänder

HEADER	Aktuelle Mitteilungen des ZIB (CYBER 825)
HEAD175	Aktuelle Mitteilungen des ZIB (CYBER 175)
MANUALS	Handbücher (Manuals), Skripte etc.
TELEFON	Telefonnummern im ZIB
ZIB	Verweis auf rechtliche Grundlagen des ZIB

Mit dem Kommando: DOC,ORGANISATION,*,SEL=BENORD. erhalten Sie die Kurzfassung aller Informationen, die Teil der betrieblichen Regelungen des ZIB sind.

Sachgebiet SOFTWARE: Information über die Software, die an den drei wissenschaftlichen Rechenzentren Berlins (an FU, TU und ZIB) verfügbar ist.

Sachgebiet SPRACHEN: Übersicht über die am ZIB zur Verfügung stehenden Programmiersprachen.

Sachgebiet SYSTEM (CD) bzw. BESYSTEM (MVS): Beschreibung einer Reihe von zusätzlichen Programmen auf der CYBER 825.

ANATAPE	Analyse von Magnetbändern
BAENDER	Regelungen zur Benutzung von Magnetbändern
CANCEL	Herausnahme von Jobs aus der Input-Queue und Abbruch von aktiven Jobs durch den Benutzer
CLEARAC	Löschen fremder IDs aus der Erlaubnisliste
COMPACT	Komprimieren von Datenfiles
DOC	Allgemeines Dokumentationsprogramm
FAMILY	Family Indirect File System (Konzept und Kommandos)
FEXPORT	Erzeugen eines Jobs, der Textdateien (Quellen, Daten, Dokumente) Satzorientiert im EBCDIC- oder ASCII-Code so auf ein Magnetband schreibt, daß dieses auch von anderen Betriebssystemen verarbeitet werden kann
FIMPORT	Erzeugen eines Jobs, der Textdateien (Quelle, Daten, Dokumente) Satzorientiert im EBCDIC- oder ASCII-Code von einem Magnetband mit fester Recordlänge und und Blockung liest, wie es von vielen Betriebssystemen geschrieben werden kann.
GIVEAC	Einräumen von Zugriffsrechten an Fremde IDs
LFTRANS	Local-File-Transfer
LIMITS	Richtlinien für die Vergabe von Limits
LISTID	Information über eigene permanente Files
LISTPF	Liste der permanenten Files (Ersatz für AUDIT)
MBO	Anfordern von Magnetbändern im ZIB
MICRO	File-Transfer zwischen Cyber unter NOS/BE und PC
MTT	Senden einer Meldung von einem Job an ein Terminal
PASSWORTE	Passworte im Batchbetrieb
PERMFILES	Regelungen zur Benutzung permanenter Dateien
PRINT	Allgemeines Druckkommando
PRIORITY	Regelungen von Prioritätsfragen
PRO	Page Remote Output (Sichten von Batchjobslistings)
QUEUES	Informationen über Jobs am eigenen Rechnern
RST	Informationen über Jobs an anderen Rechnern
STACK	Manager für Jobumgebung
STARTUP	Startup-Prozedur beim Terminal-LOGIN
STATIS	Informationen über Betriebsmittelinanspruchnahme
STATUS	Informationen über Jobs in der CYBER 170-825
SYSTEM	Organisatorische Aspekte des Betriebssystems NOS/BE
TUBE	Zeileneditor
UMP	Dump von Feldlänge und Exchange Package

Sachgebiet UTILITIES (MVS): Nützliche Programme des ZIB auf der MVS-Anlage.

COMPRESS	Bereinigen von PDS- und SEQ-Dateien, Freigeben nicht belegter Spuren von Dateien
RMTPRT	(ReMoTe PRinT) Drucken von Dateien, Benutzerbeschreibung der Kommandoprozedur RMTPRT
TAPESAVE	Sichern von mehreren Dateien auf ein Band, mit interaktiver Eingabe der Dateinamen
TPINF	Informieren über den Inhalt von Magnetbändern, Benutzerbeschreibung des Programmes TPINF
TRANSFER	Transfer von Dateien und Jobs, Rechnerkopplung CD-Siemens-CRAY des ZIB
TSOINFO	Information über den Terminalbenutzer, Kennung, Name usw.

1.8 Adressen:

Anschriften und Telefonnummern der Kooperationspartner:

Postanschriften:	Telefonnummern:
Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) Bereich Anlagenbetrieb Heilbronner Str. 10 1000 B E R L I N 31	Sekretariat: (030) 8 96 04-131 Leitung: Hr. Gottschewski ...130 Abteilungen: CRAY/CD: Hr. Busch ... 135 MVS/WS : Hr. Kujawa ... 150 Maschinensaalleiter: Hr. Goetz ... 160
Freie Universität Berlin Zentraleinrichtung für Datenverarbeitung (ZEDAT) Fabeckstr. 32 1000 B E R L I N 33	Sekretariat: (030) 838-6055 Leitung: Hr. Giedke ... 4215 Beratung: Bereich- CD: ... 4206 SIEMENS: ... 6029 Benutzerverwaltung: ... 6069
Technische Universität Berlin Zentraleinrichtung Rechenzentrum (ZRZ) Straße des 17. Juni 135 1000 B E R L I N 12	Sekretariat: (030) 314-2703 Leitung: Hr. Gürtler ... 4230 Beratung: ... 5253 Rechnerbetrieb: Hr. Brodnicki ... 4232 Konsole ... 4236
Universität Kiel - Rechenzentrum - Olshausenstraße 40 2300 K I E L 1	Betreuung der CRAY-Benutzer: Hr. Dr. Mordhorst (0431)880-2766 Geschäftszimmer -2768
HAHN-MEITNER-INSTITUT für Kernforschung Berlin GmbH (HMI) Bereich Datenverarbeitung Postfach 39 01 28 1000 B E R L I N 39	Hr. Würz (030) 8009-2515 Fr. Kuß -2545
Regionales Rechenzentrum für Niedersachsen (RRZN) Universität Hannover Schloßwender Str.5 3000 H A N N O V E R 1	Telefon: (0511) 762-2883 Betreuung der CRAY-Benutzer: Hr. Buschmann -4667 Hr. Fischer -5132 Benutzerberatung: -4737

2. CRAY-Job-Ausführung

2.1 Aufbau eines CRAY-Jobs

Ein CRAY-Job liegt in einer Datei (Dataset), die in mehrere Dateiabschnitte (Files) gegliedert sein kann. Diese Untergliederungen werden durch /EOF-Marken (End of File) markiert. Der erste Dateiabschnitt enthält immer die vollständige CRAY-Steuerkartensequenz JCL (Job Control Language). Die folgenden Dateiabschnitte können z.B. FORTRAN-Programme oder Datensätze enthalten. Der Job kann durch die Marke /EOD (End of Data) abgeschlossen werden.

Achtung:
Der Begriff "File" hat im CRAY Betriebssystem COS eine andere Bedeutung als im CD Betriebssystem NOS/BE: Ein COS "Dataset" kann aus mehreren "Files" bestehen, ein NOS/BE "File" kann aus mehreren Sections bestehen. Beim Übergang von NOS/BE zu COS wird aus einer NOS/BE End-of-Section Marke, einer NOS/BE End-of-File Marke oder einer Karte "/EOF" eine COS End-of-File Marke. Eine Karte "/EOD" oder das Ende einer NOS/BE Datei entspricht einem COS End-of-Dataset.

```
JOB.
ACCOUNT,AC=TUB12345678,APW=GEHEIM.
CFT.
LDR.              (besser: SEGLDR,GO.)
/EOF
      PROGRAM TEST
      CHARACTER*25 TEXT
      READ (5,100) TEXT
      WRITE (6,100) TEXT
100   FORMAT (1X,A25)
      STOP
      END
/EOF
DIES IST EIN TESTPROGRAMM
/EOF
/EOD
```

Die im folgenden geschilderte Strukturierung entspricht der gewohnten Gliederung eines Jobs für NOS/BE Anlagen:

Bei Beginn der Jobbearbeitung auf der CRAY wird der erste Dateiabschnitt (die Steuerkarten) auf der Datei $CS abgelegt, die übrigen Dateiabschnitte (Programme, Daten etc.) werden auf der Datei $IN abgelegt. Der Benutzer kann auf die Datei $CS nicht direkt zugreifen; die Datei kann vom Benutzer zwar gelesen, aber nicht beschrieben werden. Die FORTRAN-Kanalnummer 5 ist von Anfang an der Datei $IN zugeordnet.

Die Reihenfolge der Steuer-Anweisungen bestimmt dabei, zu welchem Zeitpunkt auf welche Datei zugegriffen wird. Es können so z.B. mehrere FORTRAN-Programme innerhalb eines Jobs bearbeitet werden. Um die Synchronisation zwischen Steuer-Anweisung und Eingabe-Datei zu gewährleisten, existiert auf jeder der Dateien $CS und $IN ein Zeiger, der auf den jeweils als nächsten zu verarbeitenden Satz (FORTRAN- bzw. Steueranweisung oder Datenkarte) zeigt.

Der Zeiger für die Steuer-Anweisung kann vom Benutzer nur bedingt manipuliert werden. Er wird in der Regel auf die nächste Steueranweisung gesetzt, wenn eine Anweisung vollständig abgearbeitet ist. Ausnahmen sind z.B. Fehlerabbrüche, Prozeduraufrufe und bedingte Steueranweisungen.

Die Positionierung des Zeigers auf die Datei $IN geschieht im allgemeinen automatisch. Der Zeiger kann jedoch durch FORTRAN-READ-Anweisungen, BACKSPACE usw., durch Steueranweisungen wie REWIND, SKIPR usw. aber auch durch Steuer-

anweisungen, die Eingabe von $IN erwarten (z.B. den Übersetzer-Aufruf CFT), verändert werden. Bei der Bearbeitung mehrerer Dateiabschnitte von $IN ist auf die korrekte Positionierung des Zeigers auf den zu bearbeitenden Dateiabschnitten zu achten. Gegenüber dem bei NOS/BE gewohnten gilt jedoch bei COS: Positionieren erfolgt satzgenau! Soll z. B. in einem Job ein Programm zweimal mit je einem Datensatz aufgerufen werden und ist nach dem ersten Programmlauf die Datei noch nicht auf /EOF positioniert, so beginnt das zweite Programm beim folgenden Satz im gleichen Dateiabschnitt ("File") und nicht, wie bei NOS/BE, im folgenden Dateiabschnitt.

2.2 Abarbeitung eines CRAY-Jobs

Ein Job erreicht die CRAY in Form einer Datei, die vom Vorrechner übertragen und in der Eingabewarteschlange gespeichert wird; diese erhält eine im System eindeutige Jobnummer (Job Sequence Number), bis die Jobbearbeitung beendet ist. Das Betriebssystem COS der CRAY analysiert die Parameter in der JOB-Anweisung, um den Bedarf des Jobs festzustellen. Die Jobs werden in Abhängigkeit von den Parametern für Rechenzeit, Speicheranforderung, Herkunft sowie ggf. Magnetbandanforderung (auf dem Vorrechner) verschiedenen Klassen zugeteilt und erhalten eine Bearbeitungspriorität (==> 2.3). Wenn die angeforderten Betriebsmittel auf der CRAY bereitstehen und im Sinne einer gerechten Abarbeitungsreihenfolge (==> 2.3) der Job zur Bearbeitung ansteht, wird er in die Ausführungswarteschlange (Executive Queue) eingereiht.

Zu Beginn der Ausführung werden außer den Dateien $CS und $IN noch die Dateien $OUT und $LOG eröffnet. $OUT enthält die normale Ausgabeliste, die FORTRAN-Kanalnummer 6 ist z.B. von Anfang an dieser Datei zugeordnet. $LOG enthält ein Protokoll der Steueranweisungen mit den dazugehörigen Meldungen, der Uhrzeit, sowie den Angaben über verbrauchte Betriebsmittel. Die gesamte Ausgabe des Jobs, also die Dateien $OUT und $LOG, werden in einer Datei unter dem Namen des Jobs abgespeichert, in die Ausgabe-Warteschlange eingereiht und zum Vorrechner übertragen.

Bei der Initialisierung erhält ein Job den Speicherplatz, der für die Analyse der Steueranweisungen benötigt wird. Danach kann er dann soviel Speicher belegen, wie im MFL-Parameter der JOB-Anweisung angegeben ist, jedoch wird die tatsächliche Speicherbelegung vom System dynamisch dem tatsächlichen Speicherbedarf angepaßt.

Tritt eine Fehlerbedingung im Job auf, so wird die Ausführung nicht mit der nächsten Steueranweisung, sondern nach der nächsten EXIT-Anweisung fortgesetzt oder die Bearbeitung abgebrochen.

Unter verschiedenen Umständen kann ein Wiederanlauf eines unterbrochenen Jobs nötig sein (z.B. System Restart). Dieser erfolgt automatisch, wenn der Job nicht seine Wiederaufsetz-Fähigkeit durch Schreiben, Löschen oder Modifizieren von permanenten Dateien oder durch Anfordern von Dateien vom Vorrechner verloren hat.

Nach Systemunterbrechungen kann ein Job an der Stelle seine Berechnungen wieder aufnehmen, an der zum letzten Mal ein komplettes Abbild des Jobs zwischengespeichert wurde (ROLLOUT). Dies erfolgt durch das ZIB automatisch etwa alle 5 Minuten. Vorsicht: nicht alle Jobs sind auf diese Weise restartfähig!

2.3 Klassen- und Prioritäten

Jeder Job, der in die Eingabe-Warteschlange der CRAY gelangt, wird nach den Angaben zu Betriebsmitteln in der JOB-Steuerkarte (CPU- Zeit, Ein-/Ausgabezeit, Hauptspeicherbedarf, Plattenspeicherbedarf, Magnetbandbedarf, BUFFER MEMORY Bedarf) und bzw. oder seiner Herkunft (Freie Universität Berlin, Hahn-Meitner-Institut, Niedersachsen, Schleswig-Holstein, Technische Universität Berlin) einer bestimmten Jobklasse zugeordnet. Achtung: die im folgenden aufgeführte Vorgehensweise gilt zum Zeitpunkt der Drucklegung (April 1987) und wird prinzipiell auch Bestand haben; jedoch werden einzelne Zahlen der Prioritätenregelung bei Bedarf verändert. Die jeweils aktuelle Prioritätenregelung erhalten Sie über DOC,CRAY,PRIORITY.

Für **kurze Entwicklungsjobs** gilt:

Name d. Klasse	CPU in Sek	IO in Sek	Feldlänge Worte,dez.	Platten- Sektoren	Magn. Band	Buffer Memory	Max. Jobs
EXPRESS	<= 2	<= 2	<= 200000	<= 2000	-	-	4
KURZ	<= 20	<= 20	<=1000000	<= 2000	-	<= 414	4

Jobs der Klassen KURZ oder EXPRESS werden vom Betriebssystem COS aus der Eingabe-Warteschlange in die Ausführungs-Warteschlange übernommen, sobald die Jobnummer (Job Sequence Number) die jeweils kleinste der Klasse ist (first in - first out) und die Maximalzahl aktiver Jobs in dieser Klasse noch nicht erreicht ist. In der Regel wird der Job unmittelbar nach Einbringen in die Eingabe-Warteschlange ausgeführt. Bei Überschreiten eines der Limits (auch des IO - Limits) wird der Job jedoch abgebrochen. Jobs, die die **Multitasking**-Eigenschaften der CRAY nutzen, fallen in die Spezialklasse MULTI:

Name d. Klasse	CPU-Zeit in Sek.	Feldlänge Worte,dez.	Magn. Band	Max. Jobs	Jobkenn- zeichnung
FUB	<= 20000	<= 3000000	-	4	Freie Universität Berlin
HMI	<= 20000	<= 3000000	-	1	Hahn-Meitner-Institut
NDS	<= 20000	<= 3000000	-	3	Niedersachsen
SHL	<= 20000	<= 3000000	-	3	Schleswig-Holstein
TUB	<= 20000	<= 3000000	-	5	Techn. Universität Berlin
KURZTP	<= 20	<= 200000	TP	1/0	Tape
LANGTP	<= 2000	<= 400000	TP	1/0	Tape

Die beiden Klassen KURZ oder EXPRESS sind für Jobketten nicht zulässig, denn diese behindern alle Benutzer erheblich; das ZIB wird daher solche Jobketten abbrechen. Alle **übrigen Jobs** werden entsprechend ihrer Herkunft in eine der Klassen abgelegt:

FUB, HMI, NDS, SHL oder TUB

Jobs, die auf einem der Vorrechner ein Magnetband benötigen, müssen den Parameter TP in der Jobkarte gesetzt haben. Die aufgeführten Klassen

FUB,HMI,NDS,SHL,TUB,KURZTP und LANGTP

bilden eine 'Vor-Eingabe-Warteschlange', das heißt Jobs dieser Klassen werden nicht aus diesen Klassen heraus zur Ausführung gebracht. Auf dem Vorrechner CYBER 825 läuft regelmäßig (ca. alle 5 Minuten) ein Überwachungsprogramm mit folgenden Leistungen:

- Berechnen der Resource Priorität RP für jeden Job
- Erhöhen der Alterungspriorität AP eines Jobs jedes Benutzers

- Aufbereiten von Informationen über die Warteschlangen einschließlich der Werte der Prioritäten, Ablage auf der CYBER 825 sowie den CD-Anlagen von FUB, Niedersachsen (RRZN) und TUB, so daß der Benutzer jeweils die Information mit dem Remote Queue Status Programm RST abrufen kann (==> 4.4).

- In Abhängigkeit von der freien Kapazität der Ausführungs-Warteschlange (Zahl der Jobs, Summe der angeforderten Feldlänge aller aktiven Jobs), wird eine gewisse Zahl von Jobs entsprechend der Betriebszeit und der angeforderten Betriebsmittel einer der Jobklassen TAGS, NACHTS oder WOCHEND in der Eingabe-Warteschlange eingeordnet. Die Einordnung erfolgt gemäß der Priorität P (P = RP + AP), wobei berücksichtigt wird, daß je Benutzer nicht mehr als ein Job ausgeführt wird.

- Die Priorität P in der Ausführungs-Warteschlange ist abhängig von der angeforderten Feldlänge eines Jobs (MFL-Parameter), sie bestimmt sich zu

$$P = \frac{MFL}{500000} + 5$$

  womit insbesondere große Programme bei der Abarbeitung weniger behindert werden.

In der Regel werden Jobs in den Klassen TAGS, NACHTS und WOCHEND umgehend zur Ausführung gebracht.

Um die große Anzahl von CRAY-Jobs auf einem praktikablen Maß zu halten und für einzelne Jobs akzeptable Bearbeitungszeiten zu erreichen, gilt folgende Regelung:

- Pro Auftragsnummer dürfen nicht mehr als sechs Jobs in der CRAY (Eingabe- und Ausgabe-Warteschlange) abgelegt sein. Überzählige Jobs werden entfernt, und zwar beginnend beim jüngsten Job.

- Pro Auftragsnummer darf nur je ein Job in den Klassen EXPRESS und KURZ abgelegt sein.

Sonstige Klassen

- Die Klassen JOBSERR, OPERATR, SPECIAL, SYSTEM, BETRIEB, ONLDIAG und *ROLL* haben ausschließlich betriebsinterne Bedeutung und sind für Benutzer nicht zugänglich.

- Ein Job gelangt durch Operateureingriff aus einer beliebigen Klasse in die Klasse PARK, wenn ein vorübergehendes 'Parken' aus betrieblichen Gründen notwendig erscheint.

Die Betriebsmittel-Priorität

Die Betriebsmittel-Priorität RP (Resource Priority) berücksicht die angeforderten Betriebsmittel des Jobs, zur Zeit sind dies die geforderte CPU-Zeit (T-Parameter), die geforderte Ausgabe-Zeit (IO-Parameter) und die geforderte Feldlänge (MFL-Parameter). Diese Priorität berechnet sich wie folgt (Stand März 1987):

$$RP' = 170 - 51 * \ln \frac{(T + IO)}{50000 - (T + IO)} - 105 * \ln \left(\frac{MFL}{4000000 - MFL}\right)$$

ln: natürlicher Logarithmus

```
RP =    0    (für RP' <0)
RP =  RP'    (für 0 <= RP' <= 1000)
RP = 1000    (für RP' > 1000)
```

Einen Uberblick über die Werte von RP liefert die folgende Tabelle (Stand März 1987):

T+IO / MFL	4	10	40	100	400	1000	4000	10000	40000
100000	1000	989	918	871	800	753	679	625	483
200000	960	913	842	796	725	677	603	549	408
300000	914	868	797	750	679	632	558	504	363
400000	881	835	764	717	646	599	525	471	330
500000	855	808	737	691	620	572	498	445	303
600000	833	786	715	668	597	550	476	422	281
700000	813	767	696	649	578	531	457	403	262
800000	796	749	679	632	561	514	440	386	244
900000	780	734	663	616	545	498	424	370	229
1000000	766	719	648	602	531	483	409	356	214
1200000	740	693	622	575	504	457	383	329	188
1400000	716	669	598	551	480	433	359	305	164
1600000	693	646	576	529	458	411	337	283	141
1800000	672	625	554	507	436	389	315	261	120
2000000	651	604	533	486	415	368	294	240	99
2200000	630	583	512	465	394	347	273	219	78
2400000	608	561	491	444	373	325	251	198	56
2600000	586	539	468	421	350	303	229	175	34
2800000	562	515	444	397	326	279	205	151	10
3000000	535	489	418	371	300	253	179	125	0

Auf den Vorrechnern CYBER 825 und MVS des ZIB steht das Programm PRIOCR zur Verfügung, mit dem man sich -auch ohne Taschenrechner- Werte für RP in Abhängigkeit von T, IO und MFL ausrechnen lassen kann. (Sinnvoll ist der interaktive Aufruf von PRIOCR.)

Die Alterungspriorität AP

Jeder neu in der Eingabe-Warteschlange aufgenommene Job erhält die Alterungspriorität AP = 0. Bei jedem Überwachungslauf (ca. alle 5 Minuten) wird bei jedem Benutzer die Alterungspriorität seines ältesten Jobs um 1 bis zu einem Maximalwert von 8000 erhöht. Der Wert von 8000 wird frühestens nach 32 Tagen ereicht. Hat der erste Job die Alterungspriorität 1000 erreicht, kann ein zweiter Job dieses Benutzers anfangen zu altern; hat dieser 1000 erreicht, kann ein dritter Job anfangen zu altern etc. .

Betriebszeiten:

Es werden drei Betriebszeiten 'Tages-, Nacht- und Wochenendbetrieb' sowie drei Jobklassen TAGS, NACHTS und WOCHEND unterschieden.

Tagesbetrieb	montags - freitags jeweils 7.00 bis 20.00 Uhr
Nachtbetrieb	montags - donnerstags jeweils ab 20.00 Uhr

bis 7.00 Uhr des darauffolgenden Tages
Wochenendbetrieb freitags 20.00 Uhr bis sonntags 24.00 Uhr, an den gesetzlichen Feiertagen im Land Berlin sowie an Heiligabend und Silvester.

Klassen:

In Abhängingkeit von der Betriebsmittel-Priorität RP ist jeder Job einer dieser drei Klassen zugeordnet und wird zu folgenden Betriebszeiten gestartet (Stand März 1987):

Klasse	Betriebsmittel-Pri.	Betriebszeit
TAGS	RP >= 700	Tages-, Nacht- oder Wochenendbetrieb
NACHTS	400 <= RP < 700	Nacht- oder Wochenendbetrieb
WOCHEND	RP < 400	Wochenendbetrieb

2.4 Überschreiten des Joblimits

Jobs, die nicht in eine der in Abschnitt 2.3 beschriebenen Klassen fallen, werden bereits bei der Jobeingangskontrolle mit der Fehlermeldung

```
JOB statement error
```

abgebrochen. Daher ergeben sich folgende zulässige Maximalwerte für die einzelnen Parameter der Steuerkarte JOB:

```
T   <=  20000 (Sekunden)
IO  <=  20000 (Sekunden)
MFL <=3000000 (Worte)
BMR <=   1116 (Sektoren)
SZ  <= 200000 (Sektoren)
```

Für Magnetbandjobs gilt:

```
T   <=   2000 (Sekunden) und MFL <= 400000
```

Sollten Ihre Jobs innerhalb dieser Maximalwerte nicht ausführbar sein, so bitten wir um Rücksprache.

Jobs, die den Grenzwert für die CP-Zeit bzw. für die maximale Hauptspeicheranforderung überschreiten, werden automatisch vom Betriebssystem COS mit einer entsprechenden Fehlermeldung beendet bzw. die Abarbeitung der Steuerkarten wird hinter der nächsten EXIT-Anweisung fortgesetzt.

Jobs, die das IO-Timelimit oder das SZ-Limit überschreiten, werden mit der Ausführungspriorität 1 oder 2 "schlafen" gelegt; im RST-Status findet man für einen solchem Job die Angabe "SLEEPING". "Schläft" ein Job eines Benutzers, wird auch kein weiterer ausgeführt. Durch Anruf beim Betrieb der CRAY

```
Telefon: (030) 896 04 160 / 165
```

oder durch Information per Message an die Operateurkonsole kann bei einem Job während der Ausführung das IO-Timelimit und/oder das SZ-Limit erhöht werden (nur zu Zeiten des Operateurbetriebs).

Der Hintergrundspeicherbedarf eines Jobs erscheint im Benutzerprotokoll ($LOG) unter SZ-TOTAL: SECTORS OF DATA CREATED; maßgebend ist jeweils die letzte im Benutzerprotokoll aufgeführte Zeile mit diesem Text.

2.5 Dateien

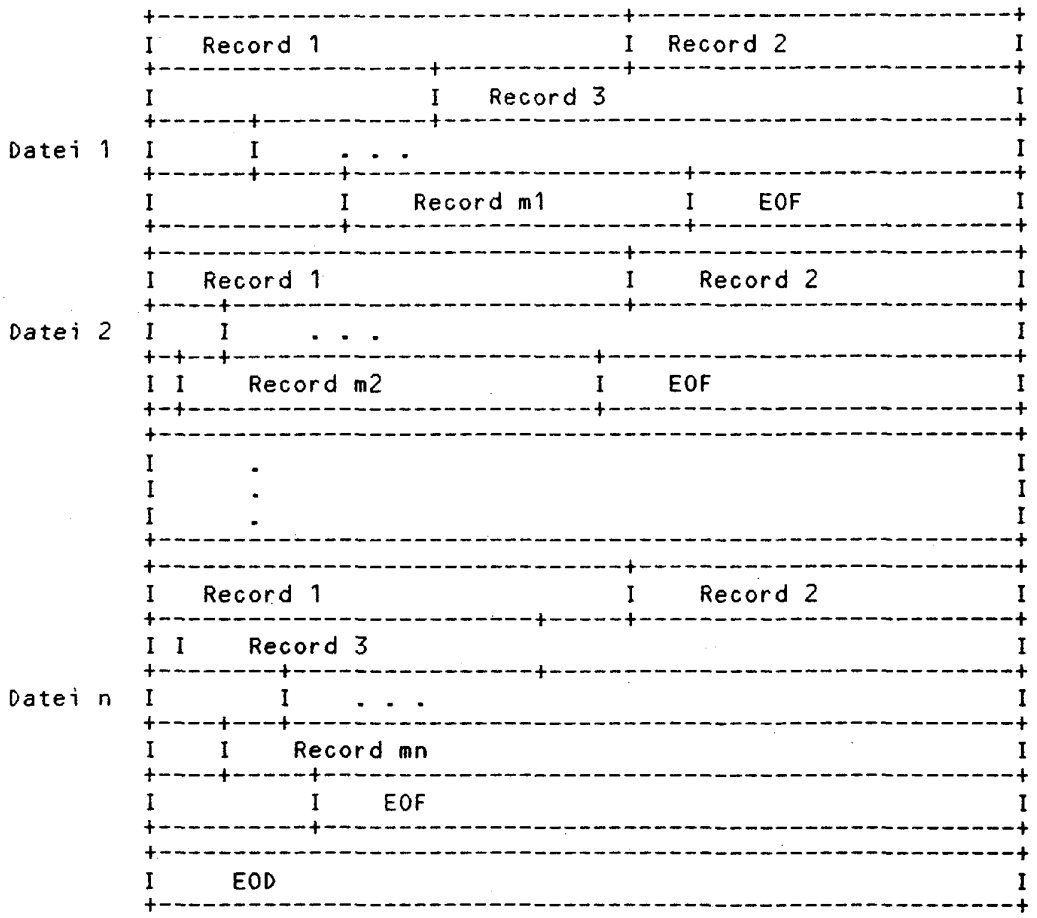
2.5.1 Namenskonvention, Struktur

CRAY Dateien werden vom CRAY Operating System (COS) beim Schreiben oder bei der Definition mit der Job-Steueranweisung ASSIGN automatisch angelegt. Sie werden auf den CRAY Platten gehalten.

Jede lokale Datei wird vom Benutzer durch ihren Namen angesprochen. Dateinamen bestehen aus ein bis sieben alphanumerischen Zeichen, das erste Zeichen muß ein Buchstabe oder eines der Zeichen $, @ oder % sein. Eine Reihe von Dateien, die das Betriebssystem COS anlegt, beginnen mit einem $, z.B. $IN, $OUT, $BLD.

Normalerweise werden die Dateien in geblockter Form erzeugt, d.h. die Datei besteht aus Blöcken zu je 512 CRAY-Worten. Dabei können die einzelnen Sätze beliebig lang werden, insbesondere auch länger als 512 Worte. Im allgemeinen entspricht eine CRAY Datei im Aufbau einem Magnetband, d.h. es werden mehrere einzelne Dateiabschnitte ("Files") zu einer neuen Datei ("DATASET") zusammengefaßt. Dies ist z.B. bei der Eingabe-Datei $IN (==> 2.1 "Aufbau eines CRAY-Jobs") der Fall, bei der die einzelnen Eingabe-Dateiabschnitte (jeweils durch /EOF getrennt) zu einer einzigen Datei zusammengefaßt werden. Normalerweise bestehen jedoch vom Benutzer angelegte Dateien aus genau einem Dateiabschnitt.

Allgemeiner Aufbau einer CRAY-Datei:



2.5.2 Lebensdauer von Dateien, Datensicherheit

Eine neue Datei wird beim ersten Beschreiben angelegt und steht dann bis zum Ende des Jobs als temporäre Datei zur Verfügung. Diese Datei ist für den Job lokal.

Soll diese Datei als permanente Datei bestehen bleiben, so muß sie mit der Steuerkarten-Anweisung SAVE permanent gemacht werden (==> 3.5.1). Sie steht dann solange zur Verfügung, bis sie vom Benutzer mit der Steuerkarten-Anweisung DELETE (==> 3.5.3) gelöscht wird. Da jedoch der CRAY Plattenplatz beschränkt ist, werden alle permanenten Dateien in der Regel **fünf Betriebstage** nach dem Anlegen vom ZIB automatisch gelöscht, gegebenenfalls bei akuten Engpässen früher. Es werden keine Sicherheitskopien von auf der CRAY permanenten Dateien auf Magnetband erstellt.

Der Benutzer sollte daher nach jeder Veränderung einer permanenten CRAY-Datei eine Kopie dieser Datei mit Hilfe der Steuerkarten-Anweisung DISPOSE oder PUTFE (==> 4.1.2) auf einen der Vorrechner übertragen.

Benutzer, die über die CYBER 825 mit der CRAY arbeiten, dürfen an diesem Vorrechner Dateien bis zu einer Größe von 2000 Pru's (ca. 1,3 M Byte) permanent halten. Bei Bedarf kann dieser Wert auf Antrag über das jeweils zulassende Rechenzentrum erhöht werden. Legt der Benutzer mehr permanente Dateien als beantragt an, so werden überzählige Dateien vom ZIB gesperrt oder gelöscht. Dateien, die länger als 125 Tage nicht im Zugriff waren, werden vom ZIB gelöscht. Einzelheiten zum Ablegen von permanenten Dateien auf der CYBER 825 erhält man auf der CYBER 825 über

```
DOC, SYSTEM,PERMFILE.
```

und an der MVS-Anlage über

```
DOC,BESYSTEM,PERMFILE.
```

Benutzer, die über die MVS-Anlage mit der CRAY arbeiten, dürfen an diesm Vorrechner Dateien bis zu einer Größe von 20 Tracks (ca. 380 KByte) permanent halten. Bei Bedarf kann dieser Wert auf Antrag über das jeweils zulassende Rechenzentrum erhöht werden. Dateien, die länger als 125 Tage nicht im Zugriff waren, werden vom ZIB gelöscht. Einzelheiten zum Ablegen von permanenten Dateien auf der MVS-Anlage erhält man über

```
DOC,MVSCRAY,DISPOSE  (auf beiden Anlagen)
DOC,SYSTEM,SYSTEM    (auf der MVS-Anlage).
```

2.5.3 Zugriff

Um auf eine bereits existierende (permanente) Datei zugreifen zu können, muß diese Datei dem Job bekannt (lokal) gemacht werden. Dies geschieht entweder mit der Steuerkarten-Anweisung ACCESS oder mit der Steuerkarten-Anweisung ACQUIRE, die der Datei einen 1- bis 7-stelligen lokalen Namen zuweisen, der mit dem permanenten Namen übereinstimmen oder von ihm verschieden sein kann. Die Anweisung ACQUIRE bewirkt, daß eine Datei, die nicht (mehr) auf den CRAY-Platten zur Verfügung steht, von einem der Vorrechner geholt wird und (wieder) als permanente Datei auf den CRAY-Platten gespeichert wird (==> 4.1). Sobald eine Datei lokal bekannt gemacht wurde, kann sie im Job verwendet werden. Eine Datei bleibt lokal bis zum Jobende oder bis sie freigegeben wird. Dies geschieht entweder mit der Steuerkarten-Anweisung RELEASE oder durch eine Parameterangabe in der DISPOSE-Anweisung.

Zugriffskontrolle

Standardmäßig kann nur derjenige auf eine CRAY Datei zugreifen, der sie angelegt hat. Sollen auch andere Benutzer auf diese Datei zugreifen können, muß man Zugriffberechtigungen (Lesen, Schreiben oder Löschen) für alle Benutzer beim Anlegen (SAVE) oder durch die Steuerkarten-Anweisung (MODIFY) oder Zugriffsberechtigungen für einzelne Benutzer mit der Steuerkarten- Anweisung PERMIT vergeben. Zu permanenten Dateien auf den Vorrechnern kann in ähnlicher Weise standardmäßig ebenfalls nur der anlegende Benutzer zugreifen.

3. Job-Steueranweisungen

Im Rahmen dieser Einführung werden bei der Erklärung der Anweisungen nur die am häufigsten benutzten Parameter aufgeführt. Dem fortgeschritteneren CRAY-Benutzer werden die unter DOC vorliegenden Informationen oder die entsprechende CRAY-Literatur (==> 1.7) als vollständige Referenz empfohlen.

3.1 Syntax der Job-Steuersprache

Der prinzipielle Aufbau der bis zu 80 Zeichen langen CRAY-Anweisungen ist

verb sep_1 *parm* sep_2 *parm* sep_2 *parm* sep_2 ... *term comment*

wobei gilt:

verb	Anweisungsname
sep_1	Komma "," oder öffnende Klammer "("
parm	Parameter, die aus nur einem Schlüsselwort oder aus einem Schlüsselwort bestehen können, dem ein oder mehrere Werte zugewiesen werden. Die jeweils gültige Form hängt von der spezifizierten Anweisung ab.
sep_2	Komma "," oder schließende Klammer ")"
term	Anweisungsende ist ein Punkt, falls sep_1 ein Komma war oder eine schließende Klammer, falls sep_1 eine öffnende Klammer war.
comment	Alle Zeichen hinter dem Anweisungsende werden als Kommentar interpretiert.

Kommentaranweisungen haben die Form:

```
* text
```

"*" ist das Anweisungsverb. Kommentare brauchen nicht durch einen Punkt abgeschlossen zu werden.

Anweisungsfortsetzung:

Werden mehr als 80 Zeichen für eine Anweisung benötigt, so kann durch das Fortsetzungszeichen (MVS: "nicht" -Zeichen, CDC: ^ -Zeichen) angezeigt werden, daß die nächste Zeile ebenfalls zu dieser Anweisung gehört. Eine Anweisung kann sich über beliebig viele Zeilen erstrecken. Ein Fortsetzungszeichen darf erst hinter sep_1 stehen, es darf nicht in einem Schlüsselwort oder in einem Wert vorkommen. Ein "nicht" - bzw. ^ -Zeichen in einer explizit durch Hochkommata gekennzeichneten Zeichenkette wird nicht als Fortsetzungszeichen interpretiert. JOB-, ACCOUNT-, DUMPJOB- und EXIT-Anweisungen können nicht fortgesetzt werden.

Beispiel:

```
JOB.
ACCOUNT,AC=xxxx,APW=PASSY.
*
*  UEBERSETZUNG EINES FORTRAN-PROGRAMMES
*
CFT,ON=F,OFF=P,L=0.
*
*  LADEN DES UEBERSETZTEN PROGRAMMS
*
LDR,SET=INDEF.        besser: SEGLDR,GO. (PRESET=INDEF ist Voreinstellung!)
*
/EOF

FORTRAN-Programm
```

Der Übersetzer liest das Quellprogramm von $IN. Eine Liste des Programms wird nicht erstellt. Der Objektmodul liegt auf der Datei $BLD und wird vom Lader gebunden, geladen und ausgeführt. Die Ausgabe mit dem Protokoll wird nach Beendigung des Jobs zu dem Vorrechner transferiert, von dem der Job abgeschickt wurde.

3.2 Job-Identifikation und Ablaufkontrolle

3.2.1 Identifikation des Jobs (JOB)

Zur Job-Identifikation bei der CRAY dient die Anweisung JOB. Sie muß als erste Anweisung im CRAY-Job erscheinen. Die JOB-Anweisung darf nicht auf einer zweiten Zeile fortgesetzt werden und muß auf Spalte 1 beginnen.

```
JOB<,JN=jn,MFL=mfl,T=tl,OLM=olm,IO=io,SZ=sz,BMR=bmr,TP=tp.FC=fc,HD=hd,CO=co>
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

JN=jn — Jobname (max. siebenstellig), der den CRAY-Job und die zugehörige Ausgabe bezeichnet.
Ausgabe von NOS/BE: 1. ein beliebiger Buchstabe; 2.-5. Auftragsnummer, 6.-7. zwei vom Betriebssystem vergebene alphanumerische Zeichen).
Ausgabe von MVS: 1.-6. Benutzerkennung ("Usernumber"), 7. ein beliebiges alphanumerisches Zeichen.
Normalerweise gibt der Benutzer JN nicht an:
Kommt der Job über den NOS/BE-Vorrechner zur CRAY, so wird der NOS/BE-Jobname auch als CRAY-Jobname verwendet; dies ist für den Weitertransport der Ausgabeliste über BERNET/DFN notwendig. Kommt der Job über den MVS-Vorrechner zur CRAY, so ist die MVS-Benutzerkennung anzugeben und ein beliebiger Buchstabe oder eine beliebige Ziffer anzuhängen, wenn sich der MVS-TSO-Benutzer die Ausgabe an seinem Terminal ansehen will. Ist der COS-Jobname von der TSO-Benutzerkennung verschieden, aber auf der CRAY gültig, so kann die Ausgabe an der ZIB-MVS-Anlage nicht am Terminal betrachtet werden. Wird ein nicht eingetragener Jobname verwendet, wird der COS-Job vorzeitig abgebrochen.

MFL=mfl
=<u>115000</u> — Maximum Field Length: MaximaleSpeicheranforderung in CRAY-Worten (dezimal). Die maximale Speicheranforderung eines Jobs wird auf das nächste Vielfache von 512 gesetzt, das größer

als mfl ist. Der Job wird abgebrochen, falls die Speicheranforderung während der Ausführung größer ist als das zulässige Maximum. Die Speicheranforderung hat Einfluß auf die Reihenfolge der Verarbeitung (==> 2.3, "Klassen und Prioritäten"). Wird MFL ohne Wert angegeben, so ist die Speicheranforderung gleich dem System-Maximum (MFL=3000000).

T=tl
=2

CPU-Time-Limit: Zeitbedarf für den Job in Sekunden. Die Zeitanforderung hat Einfluß auf die Reihenfolge der Verarbeitung (==> 2.3, "Klassen und Prioritäten"). Der Job wird abgebrochen, falls die Zeitanforderung überschritten wird.

OLM=olm

=1024

Maximale Anzahl von Blöcken (1 Block = 1 Sektor = 512 Worte = 4096 Zeichen), die für die Ausgabe zur Verfügung stehen. Maximum: OLM=20000
(Da ein Block durchschnittlich etwa 50 Druckzeilen ergibt, entspricht die Voreinstellung rund 700 Seiten; das Maximum rund 15000 Seiten.)

IO=io
=20

I/O-Time-Limit: Ein- Ausgabezeitbedarf für den Job in Sekunden (io<=20000). Die Bearbeitung des Jobs wird unterbrochen, falls die aktuelle IO-Zeit den angegebenen IO-Zeitbedarf überschreitet. Der Benutzer kann die Weiterbearbeitung eines derartig unterbrochenen Jobs durch telefonische Mitteilung an den Betrieb des ZIB (Tel. (030) 89604 160/165) erreichen. Die vom Job benötigte IO-Zeit erscheint im Benutzerprotokoll ($LOG) unter I/O TIME. Die Anforderung an IO-Zeit hat Auswirkungen auf die Reihenfolge der Abarbeitung der Jobs.

SZ=sz
=2000

Size: Hintergrundspeicherbedarf auf Magnetplatten für den Job in Sektoren (1 Block = 1 Sektor = 512 Worte = 4096 Zeichen); sz<=200000. Die Bearbeitung des Jobs wird unterbrochen, falls der aktuell benötigte Hintergrundspeicher den Wert der Speicheranforderung überschreitet. Der Hintergrundspeicherbedarf eines Jobs erscheint im Benutzerprotokoll ($LOG) unter SZ-TOTAL: SECTORS OF DATA CREATED-
Ein Job wird nur dann zur Ausführung gebracht, wenn der angeforderte Hintergrundspeicher zur Verfügung steht. Keine Angabe des SZ-Parameters wirkt wie SZ=2000.

BMR=bmr
=0

Buffer Memory: Hintergrundspeicherbedarf im Hauptspeicher des IO-Subsystems für den Job in Sektoren (bmr <= 1206). Einzelheiten über eine sinnvolle Nutzung dieses Hintergrundspeichers (==> 10. Ein/Ausgabe-Optimierung).

TP=tp
=0

Tape: Zahl der gleichzeitig benötigten Magnetbandgeräte am Vorrechner (tp<=2). Die Bearbeitung des Jobs wird abgebrochen, falls mehr Magnetbandgeräte angefordert werden. Jobs, die Magnetbänder anfordern, können nur während des bedienten Betriebs bearbeitet werden (Betriebsart TAGS).

FC=fc
=NO

Formatcode: Wirkt nur bei Ausgabe des Jobs auf einem der zentralen Drucker von ZRZ oder ZIB. Beim Ausdruck werden alle Formularsteuerzeichen ignoriert.

=YES

Beim Ausdruck werden alle Formularsteuerzeichen beachtet.

HD=hd	Header: Deckblatt des CRAY-Jobs.
=FULL	Das Deckblatt enthält auch die aktuellen Mitteilungen des ZIB (72 Zeilen).
=PART	Das Deckblatt enthält nur den Jobnamen (ca. 20 Zeilen).
CO=co	Kommentar für das Deckblatt. Die Zeichen nach CO= bis zum nächsten Komma (max. 10 bei NOS/BE, 8 bei MVS) werden in vergrößerter Darstellung zur zusätzlichen Identifikation auf das Deckblatt geschrieben.

Beispiel:

```
JOB,T=20,IO=100.FC=YES,CO=BEISPIEL,HD=PART.
JOB,JN=C2222,T=200,OLM=100.
JOB,JN=ZZZ111A,MFL=200000.
```

3.2.2 Identifikation der Abrechnungsnummer (ACCOUNT)

Als zweite Anweisung im CRAY-Job muß ACCOUNT vorliegen, welche die Abrechnungsnummer und das Abrechnungskennwort enthält.

```
ACCOUNT,AC=ac,APW=apw<,NAPW=napw>.
```

Parameter:	(die Voreinstellung (default) ist unterstrichen)
AC=ac	Account Number: Abrechnungsnummer, Buchstaben und Ziffern wie vom Rechenzentrum zugeteilt.
APW=apw	Account Password: Abrechnungskennwort (ein- bis 15-stellig). Zur Änderung des Kennwortes kann der Parameter NAPW angegeben werden.
NAPW=napw	New Account Password: neues Abrechnungskennwort (ein- bis 15-stellig). Altes und neues Kennwort müssen unterschiedlich sein.

Es ist zu beachten, daß die Änderung des Kennwortes nicht beim Eintritt des Jobs in das System, sondern erst bei der Ausführung des Jobs erfolgt, und somit keine weiteren Jobs zur Ausführung anstehen sollten, wenn das Kennwort geändert wird.

Beispiel:

```
ACCOUNT,AC=xxxx,APW=GEHEIM.
ACCOUNT,AC=xxxx,APW=GEHEIM,NAPW=UNKNACKBAR.
```

Für einige Benutzergruppen ist vom zulassendem Rechenzentrum zusätzlich zur Abrechnungsnummer AC und Abrechnungskennwort APW auch eine Benutzernummer (US: Usernumber) mit zugehörigem Benutzerkennwort (UPW: Userpassword) vergeben; in diesen Fällen lautet die ACCOUNT-Anweisung:

```
ACCOUNT,AC=ac,APW=apw,US=us,UPW=upw<,NAPW=napw,NUPW=nupw>.
```

3.2.3 Bildung von Jobketten (SUBMIT)

Mit Hilfe der Anweisung SUBMIT kann ein auf der CRAY laufender Job dem System eine Datei, die ebenfalls einen kompletten Job enthalten muß, übergeben und somit in die Eingabe-Warteschlange zur Bearbeitung einreihen. Der Job, der übergeben wird, ist unabhängig von dem Job, der die Übergabe ausführt.

Achtung: SUBMIT kollidiert mit dem Konzept von DFN: Jobs, die über DFN und die CYBER 825 zur CRAY gelangen, können Folgejobs mit SUBMIT erstellen; diese Folgejobs werden auch abgearbeitet, jedoch wird die zugehörige Ausgabedatei in der Regel nicht korrekt vom DFN zurücktransportiert. SUBMIT sollte daher nur von Benutzern aufgerufen werden, die direkt auf einem der Vorrechner CYBER 825 oder der MVS-Anlage arbeiten. Die Datei $OUT kann mit LFTRANS (==> 4.1.2) korrekt zurücktransportiert werden; eine andere Möglichkeit für Kettenjobs findet man am Ende des Kapitels 4.2.4.

```
SUBMIT,DN=dn<,NRLS>.
```

Parameter:

DN=dn — Lokaler Name der zu übergebenden Datei. DN muß angegeben werden.

NRLS — No Release: gibt an, daß die Datei, die den zu übergebenden Job enthält, weiterhin dem übergebenden Job zum Lesen zur Verfügung steht.

Die Übergabe erfolgt, wenn die SUBMIT-Anweisung ausgeführt wurde. Die übergebene Datei kann auf der CRAY von der Eingabe-Datei kopiert oder vom Vorrechner angefordert werden.

Beispiel:

```
JOB,JN=A1999,T=500.
ACCOUNT,AC=TUB10331999,APW=OTTO.
COPYF,I=$IN,O=JOBB.
ACCESS,DN=PROGA.
LDR,DN=PROGA.          besser: SEGLDR,GO,CMD='BIN=PROGA'.
SUBMIT,DN=JOBB.
/EOF
JOB,JN=A1999,T=100.
ACCOUNT,AC=TUB10331999,APW=OTTO.
ACCESS,DN=PROGB.
LDR,DN=PROGB.          besser: SEGLDR,GO,CMD='BIN=PROGB'.
/EOD
```

Das abgespeicherte Programm PROGA wird ausgeführt; der zweite Job wird aus der Eingabe-Datei kopiert, abgeschickt und ausgeführt. Beide Jobs senden ihre Ausgabelisten zum Vorrechner CYBER 825. Falls der ursprüngliche Job über DFN zur CYBER 825 gesendet wurde, besteht die Gefahr, daß die Ausgabeliste des zweiten Jobs entweder vernichtet oder fehlerhaft zurücktransportiert wird.

3.2.4 Protokollieren von Anweisungen im LOGFILE (ECHO)

Im Protokoll werden normalerweise die Steuer-Anweisungen mit den dazugehörigen Meldungen protokolliert. Mit der Anweisung ECHO kann dieses Protokoll jedoch ganz oder für bestimmte Meldungs-Klassen unterdrückt werden. Meldungen, die sich nicht klassifizieren lassen, können auch nicht unterdrückt werden. ECHO kann mehrmals während eines Jobs aufgerufen werden, d.h. das Protokoll des Jobablaufs kann ein- und ausgeschaltet werden.

Die folgenen Meldungs-Klassen erkennt das Betriebssystem:

Klasse	Beschreibung
JCL	Diese Klasse enthält das Protokoll der Anweisungen.
ABORT	Meldungen (z.B. Traceback oder ABxxx-Messages), die das Betriebssystem schreibt, wenn der Job abbricht.
PDMINF	Informationen über die Dateien, die vom PDM (Permanent-Dataset-Manager) erzeugt werden.
PDMERR	Fehlermeldungen, die vom PDM erzeugt wurden.

ECHO,ON=$class_1$:...:$class_i$,OFF=$class_1$:...:$class_i$.

Parameter: (die Voreinstellung (default) ist unterstrichen)

ON=$class_i$ =<u>ALL</u>	Nur die Meldungen der Klasse $class_i$ werden auf das Protokoll geschrieben. Die Klassen werden durch Doppelpunkte getrennt angegeben.
OFF=$class_i$ =<u>ALL</u>	Die Meldungen der Klasse $class_i$ werden unterdrückt. Ist nur OFF oder OFF=ALL angegeben ist, wird keine Meldung, auch keine Abbruchmeldung, auf das Protokoll geschrieben. Bei OFF=JCL werden die JCL-Anweisungen im Protokoll nicht wiederholt.

3.3 Übersetzung und Ausführung von Programmen

3.3.1 Der Aufruf des FORTRAN-Übersetzers (CFT)

Der Übersetzer liest das FORTRAN-Quellprogramm von der augenblicklichen Position der durch den Parameter I spezifizierten Datei bis zum Ende des nächsten Datei-Abschnitts (/EOF). Das übersetzte Binärprogramm wird auf die Datei bdn geschrieben, von der es mit Hilfe des Laders (LDR) oder des Segmentladers geladen und ausgeführt werden kann. Der CFT-Übersetzer benutzt die spezielle Vektor-Hardware der CRAY-X/MP, d.h. er optimiert und vektorisiert innere DO-Schleifen.

Der Aufruf des CRAY-FORTRAN-Übersetzers lautet:

CFT<,I=idn,L=ldn,B=bdn,ON=string,OFF=string,OPT=option>.

Parameter: (die Voreinstellung (default) ist unterstrichen; weitergehende Beschreibung ==> Kapitel 7)

I=idn =<u>$IN</u>	Lokaler Name (ein- bis siebenstellig) der Datei, die den Quellcode enthält.
B=bdn =<u>$BLD</u>	Lokaler Name (ein- bis siebenstellig) der Datei, auf die die übersetzten Binärmoduln geschrieben werden.
ON=zeichenkette	Zeichenkette ist eine Folge von Buchstaben. Jeder spezifizierte Buchstabe schaltet eine Übersetzer-Option ein.
OFF=zeichenkette	Zeichenkette ist eine Folge von Buchstaben. Jeder spezifizierte Buchstabe schaltet eine Übersetzer-Option aus.
L=ldn	Lokaler Name der Datei (ein- bis siebenstellig), auf die die

Programm-Liste geschrieben wird.

=$OUT Die Liste wird auf die Ausgabedatei geschrieben.

=0 Es wird keine Liste erstellt.

OPT=option Angabe der Übersetzer-Optionen für die Optimierung. Mehrere Optionen werden angegeben als OPT=opt:opt:opt: ...
Werden mehrere Programmteile hintereinander mit CFT übersetzt, weil z.B. für einzelne Unterprogramme verschiedene Übersetzer-Optionen notwendig sind, so können sie ohne besondere Vorkehrungen mit einem Aufruf des Laders ausgeführt werden.

3.3.2 Der Aufruf des PASCAL-Übersetzers (PASCAL)

Der Übersetzer liest das PASCAL-Quellprogramm von der nächsten Datei aus der Menge der Eingabedateien $IN oder der Datei idn. Das übersetzte Binärprogramm wird auf die Datei bdn geschrieben, von der es mit Hilfe des Laders (LDR/SEGLDR) geladen und ausgeführt werden kann. Der PASCAL-Übersetzer benutzt die spezielle Vektor-Hardware der CRAY-XMP, d.h. er optimiert und vektorisiert innere Schleifen und er bietet Spracherweiterungen zur Nutzung der Vektoroperationen.

Der Aufruf des PASCAL-Übersetzers lautet:

```
PASCAL<,I=idn,L=ldn,B=bdn,O=list>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

I=idn Lokaler Name (ein- bis siebenstellig) der Datei, auf der sich
=$IN die PASCAL-Quelle befindet.

L=ldn Lokaler Name (ein- bis siebenstellig) der Datei, die die
=$OUT Liste des Quellprogrammes enthält.
=0 Unterdrückt eine Liste. 'Fatal-'Fehlermeldungen werden auf $OUT geschrieben.

B=bdn Lokaler Name der Datei, auf die der Binärcode geschrieben
=$BLD wird.

O=list Übersetzeroptionen können durch Doppelpunkte getrennt angegeben werden. Übersetzerdirektiven, die im PASCAL-Programm angegeben werden, überschreiben die Anfangsbesetzungen.

Anmerkung: PASCAL benötigt zur Übersetzung eines kleinen Programms 340000 Worte Hauptspeicher! Weitere Hinweise: DOC,CRAY,PASCAL und CRAY PASCAL Reference Manual (==> 1.7.1).

3.3.3 Laden und Starten von Programmen (LDR/SEGLDR)

Der Lader LDR verarbeitet die vom Übersetzer erstellten Binärmoduln, die von diesem in einer Datei abgelegt sein können oder sich in einer Bibliothek befinden. Die Moduln werden geladen, gebunden und ausgeführt.
Der Segmentlader SEGLDR ist entgegen seinem Namen ein vollwertiger und effizienter Lader sowohl für segmentierte wie auch für nicht segmentierte Programme.
Da von der Firma CRAY zukünftig nur noch der SEGLDR unterstützt wird, wird empfohlen, nach Möglichkeit schon heute ausschließlich den SEGLDR zu benutzen!

Im Anschluß an die SEGLDR-Direktiven sind LDR- und SEGLDR-Steueranweisungen gegenübergestellt!

Der Lader LDR:

```
LDR<,DN=dn,SET=val,LIB=ldn,NOLIB=ldn,AB=adn,NX,^
     MAP=op,L=ldn,NA,USA,E=n,NOECHO>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

DN=dn
=$BLD
: Lokaler Name der Eingabe-Datei, von der die Moduln geladen werden. Fehlt dieser Parameter, so ist die Voreinstellung DN=$BLD wirksam, d.h. die Moduln werden von der Datei geladen, auf der der Übersetzer die Moduln per Voreinstellung ablegt. Befindet sich das zu ladende Programmsystem auf mehreren Dateien, so sind alle Dateinamen anzugeben:

DN=ldn_1:ldn_2:ldn_3: ...

Es können maximal acht Dateien angegeben werden.

SET=val
=INDEF
: Parameter zur Steuerung der Speichervorbesetzung. Der Speicher wird mit einem Bitmuster vorbesetzt, das Gleitpunktüberlauf anzeigt, so daß beim Rechnen mit nicht vorbesetzten Gleitpunkt-Variablen ein Jobabbruch wegen Gleitpunktfehler erfolgt.

=ZERO
: Speichervorbesetzung mit binären Nullen.

=ONES
: Speichervorbesetzung mit -1 (alle Bits auf 1 gesetzt).

LIB=ldn
: Name einer Bibliothek (Library), die zusätzlich zu den Systembibliotheken beim Laden berücksichtigt werden soll. Der Lader sucht und lädt nur die Moduln aus einer Bibliothek heraus, die benötigt werden. Sollen mehrere Bibliotheken angegeben werden, so sind die Namen der Bibliotheken durch Doppelpunkte voneinander zu trennen:

```
...,LIB=MYLIB:IMSL, ...
```

Es dürfen maximal acht Bibliotheken angegeben werden. Bei der Auflösung von Referenzen werden zuerst die bei DN angegebene Datei, dann die bei LIB angegebenen Bibliotheken in der spezifizierten Reihenfolge und danach erst die Systembibliotheken durchsucht. Treten aufgrund der angegebenen Bibliotheken Namenskonflikte auf, so müssen evtl. die Systembibliotheken bei LIB mit angegeben werden, um das Programm aus der gewünschten Bibliothek zu laden. Doppelte Moduln werden übersprungen und eine Meldung ausgedruckt.

NOLIB=ldn
: Mit dem NOLIB-Parameter wird eine Systembibliothek angegeben, die beim Ladevorgang nicht berücksichtigt werden soll. Wenn der NOLIB-Parameter fehlt, werden alle Systembibliotheken (wie $SCILIB,$IOLIB,$ARLIB,...) automatisch beim Ladevorgang eingeschlossen. Wenn nur NOLIB angegeben ist, wird keine Systembibliothek beim Laden berücksichtigt.

AB=adn
: Name des absoluten (binären) Objektmoduls. Der absolute Modul wird auf die Datei adn geschrieben. Es kann später durch Aufruf des Namens ausgeführt werden.

NX
: No execution: Ist dieser Parameter spezifiziert, so wird das Programm geladen aber nicht ausgeführt. Der absolute Modul wird auf die unter AB angegebene Datei geschrieben.

MAP=op	Der MAP-Parameter steuert das Erstellen einer Ladeliste.
=ON	Es wird eine Liste mit den Namen, Längen und Anfangsadressen der Blöcke einschließlich der Referenzen zu den Anfangsadressen (Cross-Reference-List) erstellt.
=FULL	Bedeutungsgleich mit MAP=ON.
=OFF	Es wird keine Liste erstellt.
=PART	Es wird die Block-Liste ohne "Cross-Reference-List" erstellt. Angabe nur des Schlüsselwortes MAP entspricht MAP=PART.
L=ldn	Name der Datei, auf die die Ladeliste geschrieben werden
=$OUT	soll.
NA	No Abort: Der Job wird nicht abgebrochen, wenn beim Laden Fehler gefunden werden, die die Meldungsklasse 3 oder höher haben (siehe E-Parameter).
USA	Unsatisfied External Abort: Der Job wird nach Beendigung des Ladens abgebrochen, wenn versucht wird, nicht vorhandene Blöcke anzusprechen. In der Ladeliste sind alle derartigen nicht gefundenen Referenzen enthalten.
E=n	steuert die Meldungen des Laders: es werden nur Meldungen der Meldungsklasse < n ausgegeben. E=2 unterdrückt die Meldungsklassen COMMENT und NOTE; CAUTION-, WARNING- und FATAL-Meldungen werden ausdruckt. Die Klasse FATAL kann nicht unterdrückt werden.
=1	COMMENT: Der Fehler behindert die Ausführung des Programmes nicht.
=2	NOTE: Der Fehler könnte die Ausführung behindern.
=3	CAUTION: Der Job bricht nach Beendigung des Ladens ab, falls der Parameter NA nicht angegeben ist. Das Programm kann gestartet werden.
=4	ERROR: Der Job bricht nach dem Laden ab, falls der Paramter NA nicht angegeben ist. Der Fehler ist so schwerwiegend, daß das Programm nicht gestartet werden kann.
=5	FATAL: Der Job bricht nach einem schwerwiegenden (fatal) Fehler sofort ab.
NOECHO	Die Anweisung, die das Laden bewirkt, wird nicht auf das Protokoll geschrieben.

Beispiel:

Der erste Job übersetzt das PASCAL-Programm und erzeugt ein absolutes Modul, welches mit dem Namen Jupiter auf Platte permanent gespeichert wird. Der zweite Job holt den absoluten Modul von der Platte und führt ihn aus.

```
JOB,MFL=340000.
ACCOUNT,AC=xxxx,APW=UNKNACKBAR.
* Anweisungen
PASCAL.
LDR,NX,AB.          besser: SEGLDR,CMD='ABD'.
SAVE,DN=$ABD,PDN=JUPITER.
* weitere Anweisungen
/EOF
/EOD
JOB,MFL=340000.
ACCOUNT,AC=xxxx,APW=UNKNACKBAR.
ACCESS,DN=JUPITER.
JUPITER.
/EOF
```

Der Segment-Lader SEGLDR:

SEGLDR<,I=idn,L=ldn,GO,DW=dw,CMD='dirstr'>.

Parameter: (die Voreinstellung (default) ist unterstrichen)

I=idn
=$IN — Input Dataset Name: diese Datei enthält die SEGLDR-Anweisungen. Weglassen des Parameters oder I=0 bedeutet: keine Eingabedirektiven. I ohne Angabe von idn bedeutet I=$IN.

L=ldn
=$OUT — List Dataset Name: Name der Datei für die Druckausgabe. L=0 unterdrückt jede Druckausgabe samt der Fehlermeldungen. Wird L ohne Parameter angegeben, gilt die Voreinstellung.

GO — GO bedeutet Laden und sofortiges Ausführen des Programms, ohne daß der ausführbare Lademodul in einer Datei abgelegt wird. Wird GO nicht angegeben, so wird nur ein ausführbarer Lademodul erzeugt, der nach $ABD geschrieben wird, sofern nicht durch die ABS-Anweisung eine andere Zuordnung getroffen wird.

DW=dw
=80 — Data Width: Anzahl der signifikanten Spalten jeder Zeile, 0<dw<81. Bei DW=72 wird SEGDLR beispielsweise die UPDATE-Zeilennummern (Spalten 73 bis 96) ignorieren.

CMD='dirstr' — Anweisungen für SEGLDR können als Direktive 'dirstr' bereitgestellt werden. Die Direktive wird als erste Anweisung der Eingabe-Datei ausgeführt, und zwar selbst dann, wenn I=0 gesetzt wurde. Die einzelnen Anweisungen werden durch ein Semikolon getrennt; maximal 80 Zeichen sind insgesamt zulässig.
Beispiel:
SEGLDR,CMD='BIN=BIN1,BIN2;LIB=MYLIB;MAP=PART'.

Die Anweisungen für den Segment-Lader

Die Anweisungen für den Segmentlader sind nachfolgend aufgelistet, soweit sie nicht der Beschreibung der Segment-Struktur dienen; die wichtigsten Anweisungen sind näher erläutert. Anweisungen von maximal 80 Zeichen Länge können mit dem Parameter CMD im Aufruf des SEGLDR angegeben werden; die einzelnen Anweisungen werden durch ein Semikolon getrennt. Längere Anweisungen können in der Eingabedatei (==> Parameter I) fortgesetzt oder auch nur dort aufgelistet werden. Kommentare werden durch * eingeleitet. Anweisungen werden durch Semikolon, Zeilenende oder * beendet, Listenelemente durch Kommata getrennt. Anweisungen, die Listen enthalten, können nach einem Komma auf der nächsten Zeile fortgesetzt werden.
Weitere Angaben und Einzelheiten sind dem CRAY Reference Manual 'SEGMENT LOADER (SEGLDR)' SR-0066 zu entnehmen (==> 1.7.1).

Dateizuordnung

Mit den Anweisungen BIN, LIB, NODEFLIB und ABS werden Dateien definiert, die SEGLDR erzeugen und/oder benutzen soll.

BIN=dn_1<,dn_2,dn_3,.....,dn_n> Name der binären Eingabe-Dateien.
=$BLD — Bei Namensgleichheit wird der erste Modul geladen; alle weiteren Modulnamen werden in einer Meldung ausgedruckt.

LIB=lib_1<,lib_2,lib_3,.....,lib_n> Namen der Bibliotheken, die zusätzlich zu

den Systembibliotheken beim Laden berücksichtigt werden sollen. Der Lader sucht und lädt nur die benötigten Moduln einer Bibliothek. Bei der Auflösung von Referenzen werden zuerst die bei BIN angegebenen Dateien, dann die bei LIB angegebenen Bibliotheken in der spezifizierten Reihenfolge und erst dann die Systembibliotheken durchsucht. Treten aufgrund der angegebenen Bibliotheken Namenskonflikte auf, müssen evtl. die Systembibliotheken bei LIB mit angegeben werden, um das Programm aus der gewünschten Bibliothek zu laden. Bei Namensgleichheit wird der zuerst gefundene Modul geladen; über weitere gefundene Modulnamen werden keine Meldungen ausgegeben.

NODEFLIB — ignoriert alle voreingestellten Bibliotheken.

ABS=dn — Datei, auf die der absolute Objektmodul geschrieben wird.
=$ABD

Listensteuerung

Mit den Anweisungen ECHO, MAP und TRIAL wird die Ausgabe von SEGLDR auf die Ausgabedatei gesteuert.

TRIAL — führt einen Testlauf durch, ohne daß eine Ausgabe erzeugt wird. Man erhält dadurch die Ladeliste, die meisten Fehlermeldungen sowie Angaben zum Speicherbedarf.

ECHO=OFF — unterdrückt die Druckausgabe von Eingabe-Anweisungen bzw.
=ON — setzt den Ausdruck fort.

MAP=dir — Durch die Zuweisung unterschiedlicher Parameter zur MAP-Anweisung kann man sich unterschiedliche Daten aus der Ladeliste des SEGLDR ausgeben lassen.

=NONE	es werden keine Ladelisten ausgegeben.
=STAT	Angaben zum gebundenen Programm wie Größe, Datum und Uhrzeit.
=ALPHA	wie STAT, zusätzlich alphabetisch sortierte Ladeliste für alle Unterprogramme.
=ADDRESS	wie ALPHA, nach Adressen sortiert.
=PART	wie ALPHA plus ADDRESS
=EPXRF	Entry Point Crossreference: wie STAT, zuzüglich Kreuzreferenzliste der Eingangspunkte
=CBXRF	Common Block Cross Reference: wie STAT, zuzüglich Kreuzreferenzliste der COMMON-Blöcke.
=FULL	wie PART plus EPXRF plus CBXRF.

Moduln und gemeinsame Speicher-Blöcke ("Common-Blocks")

Mit den Anweisungen MODULES, COMMONS und DYNAMIC kann die Reihenfolge festgelegt werden, in der die zu ladenden Moduln oder (gemeinsamen) Speicherblöcke geladen werden.

MODULES=mod<:dn> — einzelne Moduln können angegeben werden, die zu binden sind. Mehrere Angaben sind durch Kommata zu trennen.

=mod	Der Modul mod soll aus der ersten Datei, in der er gefunden wird, gebunden werden.
=mod:dn	Der Modul mod soll aus der Datei dn gebunden werden, auch wenn er in mehreren Dateien vorkommt.

COMMONS=blk<:size> es werden COMMON-Blöcke in der angegebenen Reihenfolge gebunden. Mehrere Angaben sind durch Kommata zu trennen.

=blk benennt den zu bindenden COMMON-Block.

=blk:size überschreibt zusätzlich die in den Programmquellen angegebene Größe des COMMON-Blockes.

DYNAMIC=blk legt den Anfang des (gemeinsamen) Speicherblocks auf das erste Wort, das dem längsten Segment-Abschnitt folgt. Die Größe dieses (gemeinsamen) Speicherblocks kann vom Benutzer verändert werden.

=// definiert den unbenannten (gemeinsamen) Speicherblock ("Blank Common") als dynamisch.

Fehlerbehandlung

Mit den Anweisungen MLEVEL und USX wird die Fehlerbehandlung gesteuert.

MLEVEL=ERROR Nur Fehler-Meldungen werden ausgegeben; SEGLDR beendet sich augenblicklich. Es wird kein ausführbares Programm erstellt.

=WARNING Fehler- und Warnungs-Meldungen werden ausgegeben. Zwar wird kein ausführbares Programm erstellt, die Verarbeitung wird aber fortgesetzt, so daß weitere Meldungen ausgegeben werden können.

=CAUTION Ausgabe von Fehler-, Warnungs-, und Achtung-Meldungen. Ein ausführbares Programm wird trotz aufgetretenen Fehlers erzeugt.

=NOTE Ausgabe von Fehler-, Warnungs-, Achtung- und Hinweis-Meldungen. SEGLDR wurde z.B. nicht korrekt oder ineffektiv benutzt; keine Auswirkung auf die Ausführung.

=COMMENT Alle Meldungen werden ausgegeben. Keine Auswirkung auf die Ausführung.

USX=WARNING SEGLDR kennzeichnet unbefriedigte Externbezüge (unsatisfied external symbols) und erzeugt kein ausführbares Programm.

=CAUTION SEGLDR kennzeichnet unbefriedigte Externbezüge (unsatisfied external symbols) und erzeugt ein ausführbares Programm.

=IGNORE SEGLDR erzeugt trotz unbefriedigter Externbezüge (unsatisfied external symbols) ohne Warnung ein Programm.

Speichersteuerung

Mit den Anweisungen HEAP und STACK kann die Größe der Speicherbereiche festgelegt werden, die durch die vom System angebotene Heap- und Stackverwaltung dynamisch zur Laufzeit verändert werden kann. Falls die DYNAMIC-Anweisung angegeben ist, ist die Größe des Heap nicht dynamisch veränderbar.

HEAP=init Anfangsgröße in Worten für die Heap-Verwaltung.

=init+inc Increment: Größe der Speichererweiterung, wenn der Heap überläuft. Durch eine DYNAMIC-Anweisung (s.o.) wird inc auf 0 gesetzt.

=init>min Minimum: gibt die kleinste Blockgröße (min größer oder gleich
=init+inc>min 2) in der Freispeicherliste des Heap an.

=2048+2>=2 wird eingesetzt, wenn die Heap-Anweisung ohne Parameter angegeben ist.

STACK=init	Anfangsgröße in Worten für den Teil des Heap, der der Stack-Verwaltung zur Verfügung gestellt wird.
=init+inc	Increment: Größe der Speichererweiterung, wenn der Stack überläuft.
=2048+256	wird eingesetzt, wenn die Stack-Anweisung ohne Parameter angegeben ist.

Vorbelegen von Speicherbereichen

Mit der Anweisung PRESET lassen sich Speicherbereiche vorbelegen:

PRESET=ONES	Vorbelegung mit -1
=ZEROS	Vorbelegung mit 0
=INDEF	Vorbelegung mit einem Wert derart, daß dessen Verwendung in einer Gleitkomma-Operation zur Anzeige eines Gleitkomma-Fehlers führt.
=-INDEF	Vorbelegung wie indef, aber negativ.
=value	eine 16-bit Oktalzahl kann vom Benutzer vorgegeben werden.

Gegenüberstellung von einander entsprechenden Steueranweisungen:

LDR-Steueranweisungen:	SEGLDR-Steueranweisungen:
LDR.	SEGLDR,GO.
LDR,NX.	SEGLDR.
LDR,NX,AB=OUTPUT.	SEGLDR,CMD='ABS=OUTPUT'.
LDR,DN=DOG.	SEGLDR,GO,CMD='BIN=DOG'.
LDR,NX,LIB=MYLIB,MAP=PART.	SEGLDR,CMD='LIB=MYLIB;MAP=ADDRESS'.
LDR,NX,STK=10000,MM=50000.	SEGLDR,CMD='STACK=10000;HEAP=50000'.

Will man bei LDR nicht die Voreinstellungen benutzen, so muß man Parameter angeben; beim SEGLDR spezifiziert man hingegen (Steuer-)Anweisungen, die man entweder über CMD angibt (SEGLDR,CMD='dir_1;dir_2;dir_3; ...;dir_n'.) und/oder in einer Datei bereitstellt, die mit I spezifiziert wird (SEGLDR,I=dn.).

Gegenüberstellung von einander entsprechenden Parametern bzw. Anweisungen:

DN=DS1:DS2:DS3: ...DSn	BIN=DS1,DS2,DS3, ...DSn
AB=OUTPUT	ABS=OUTPUT
LIB=LIB1:LIB2:LIB3: ...LIBn	LIB=LIB1,LIB2,LIB3, ...LIBn
MAP=PART	MAP=ADDRESS
MAP=FULL	MAP=ADRESS,EPXRF
MM=50000:10000	HEAP=50000+10000
STK=2000:1000	STACK=2000+1000

Eine Tabelle aller einander entsprechender LDR-Parameter und SEGLDR-Anweisungen findet man im SEGMENT LOADER REFERENCE MANUAL SR 0066, Seite C2.

3.3.4 Schützen der I/O-Tabellen und -Puffer (IOAREA)

Mit der IOAREA-Anweisung können die I/O-Tabellen (DSP) und die I/O-Puffer vor nicht beabsichtigtem Überschreiben geschützt werden. Wenn der Parameter LOCK angegeben ist, kann ein Benutzerprogramm den Speicherbereich nicht mehr direkt lesen oder beschreiben, der oberhalb des Programmes im Benutzerbereich liegt. Das Benutzerprogramm kann natürlich dennoch Ein- und Ausgaben machen. Die Systembibliotheksroutinen können immer noch auf diesen Bereich zugreifen. Das erfordert aber einen sehr hohen Systemoverhead, da die Systemroutinen bei

jedem I/O-Aufruf die 'High-Limit-Address' verändern müssen. IOAREA,LOCK sollte daher nur zu Testzwecken verwendet werden.

```
IOAREA,LOCK.
IOAREA,UNLOCK.
```

Die Parameter LOCK und UNLOCK schließen einander aus. UNLOCK ist die Voreinstellung.

3.4 Behandlung lokaler Dateien

3.4.1 Datei definieren (ASSIGN, DASSIGN)

Dateien werden automatisch beim ersten Zugriff oder mit Hilfe einer der Anweisungen ASSIGN oder DASSIGN angelegt, wobei ihre Kennwerte festgelegt werden. Bei Dateien auf Magnetplatten verwendet DASSIGN für den Parameter DV anstelle des physikalischen Gerätenamens eine symbolische Zuordnungsnummer. Will man von FORTRAN aus eine Datei ansprechen, der nicht durch die FORTRAN-Anweisung OPEN eine FORTRAN-Kanalnummer zugewiesen wurde, so muß die Zuweisung mit ASSIGN/DASSIGN erfolgen. ASSIGN/DASSIGN muß dann vor dem Aufruf des Laders stehen, auf den es Bezug nimmt, d.h. die Kanalnummer muß vor der Ausführung des Programms definiert sein. Die Zuweisung der Kanalnummer bleibt bis zum Jobende bestehen oder bis sie durch ein neues ASSIGN/DASSIGN überschrieben wird.

```
DASSIGN,DN=dn,DV=dv<,DEF=def,A=FTxx,LM=lm,U,RDM,MR,BS=blk>.
ASSIGN,DN=dn<,DV=dv,A=FTxx,LM=lm,U,RDM,MR,BS=blk>.
```

Parameter:	(die Voreinstellung (default) ist unterstrichen)
DN=dn	Ein- bis siebenstelliger lokaler Name der Datei.
DV=dv	bei DASSIGN: symbolische Kanal-Nummer. bei ASSIGN: physikalische Gerätenummer, z.B. D0-A1-21 (Magnetplatte) oder BMR-0-20 (Buffer Memory). Eine Liste der vergebenen Nummern erhält man unter DOC,CRAY,KONFIGUR.
DEF=def	nur bei DASSIGN: bestimmt den Gerätetyp, z.B. DEF=DD29 (Übertragungsgeschwindigkeit: 4,5 MB/s) oder DD49 (11 MB/s).
A=FTxx	xx bezeichnet die FORTRAN Kanalnummer, mit der die Datei von FORTRAN aus angesprochen werden kann; xx ist eine zweistellige Zahl zwischen 00 und 99.
LM=lm	Maximum Size Limit: Maximale Größe einer Datei in Blöcken zu 512 Worten.
=40000	Maximum: LM=160000 Blöcke
U	Unblocked Dataset Structure: Es wird eine Datei angelegt, die eine ungeblockte Struktur hat. (==> 2.5.1).
RDM	Random Dataset: Die Datei wird mit wahlfreiem Zugriff gelesen oder beschrieben. Wenn der Parameter RDM fehlt, kann nur sequentiell oder mit FORTRAN "Direct Access" gelesen oder geschrieben werden.
MR	Memory Resident Dataset: Wenn eine Datei so klein ist, daß sie im Puffer-Bereich Platz hat und viel mit ihr gearbeitet

wird, ist es sinnvoll, sie ausschließlich im Hauptspeicher zu halten. Der Puffer wird erst auf Platte geschrieben, wenn er voll ist. Normalerweise wird der Puffer geleert, wenn er halb voll ist, um die Ausgabe zu beschleunigen.

BS=blk
=4 — Buffer Size: blk gibt die Größe des Puffers in 512-Wort-Blöcken an (==> IO-Optimierung, Kap. 9.5).

3.4.2 Datei freigeben (RELEASE)

RELEASE gibt eine im Job lokal angelegte Datei sowie ihren Pufferplatz im Hauptspeicher frei. Der von einer nicht permanenten Datei belegte Speicherplatz wird freigegeben. Eine permanenten Datei wird nur dann zurückgeschrieben, wenn sie zuvor verändert worden ist; vorausgesetzt, sie ist sequentiell und hat geblocktes Format. Das System wird über die Änderung informiert (ADJUST).

RELEASE,DN=dn_1:dn_2:...:dn_n.

Parameter: (die Voreinstellung (default) ist unterstrichen)

DN=dn_i — Lokaler Name der freizugebenden Datei. Es können maximal acht Dateien angegeben werden.

3.4.3 Datei zurückspulen (REWIND)

Eine Datei wird mit REWIND auf den Anfang der Daten positioniert. Dateien werden nach der Bearbeitung i.a. nicht automatisch vom System an den Anfang positioniert (außer bei Jobende). Soll z.B. eine Datei von mehreren Programmen innerhalb eines Jobs benutzt werden, so ist jeweils ein REWIND erforderlich.

REWIND,DN=dn_1<:dn_2:...:dn_n>.

Parameter: (die Voreinstellung (default) ist unterstrichen)

DN=dn_i — Lokaler Name der Datei. DN muß angegeben werden.

3.4.4 Sätze kopieren (COPYR)

Einzelne Sätze (RECORDS) können mit Hilfe der COPYR-Anweisung von einer Datei auf eine andere Datei kopiert werden; dabei beginnt der Kopiervorgang auf beiden Dateien an der augenblicklichen Position; bei noch nicht benutzten Dateien am Anfang. Nach dem Kopieren stehen die Zeiger in beiden Dateien hinter dem letzten kopierten Satz.

COPYR<,I=idn,O=odn,NR=nr,S=s>.

Parameter: (die Voreinstellung (default) ist unterstrichen)

I=idn
=$IN — Lokaler Name der Eingabe-Datei, von der kopiert wird.

O=odn
=$OUT — Lokaler Name der Ausgabe-Datei, auf die die Kopie geschrieben wird.

NR=n
=1 — Dezimale Anzahl von Sätzen, die kopiert werden sollen. Wird NR ohne Angabe eines Wertes angegeben, wird solange kopiert, bis die nächste EOF-Marke erreicht wird. Steht der

Zeiger der Eingabe-Datei innerhalb eines Satzes, so zählt dieser Satz als ganzer Satz.

S=s
=0 — S gibt die Anzahl der Leerschritte (ASCII) am Satzanfang an.

3.4.5 Dateiabschnitte (Files) kopieren (COPYF)

Dateiabschnitte (Files) können mit Hilfe der COPYF-Anweisung von einer Datei auf eine andere Datei kopiert werden. Dabei beginnt der Kopiervorgang auf beiden Dateien an der augenblicklichen Positionierung, d.h. bei noch nicht benutzten Dateien am Anfang. Nach dem Kopieren stehen die Zeiger beider Dateien hinter der EOF-Marke des letzten kopierten Dateiabschnitt.

```
COPYF<,I=idn,O=odn,NF=n,S=s>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

I=idn
=$IN — Lokaler Name der Eingabe-Datei, von der kopiert wird.

O=odn
=$OUT — Ein- bis siebenstelliger lokaler Name der Ausgabe-Datei, auf die die Kopie geschrieben wird.

NF=n
=1 — Dezimale Anzahl von Dateiabschnitten, die kopiert werden. Wird nur der Parameter NF angeben, wird solange kopiert, bis die EOD-Marke erreicht wird. Steht der Zeiger in der Eingabe-Datei mitten auf einem Dateiabschnitt, so zählt dieser wie ein ganzer Dateiabschnitt.

S=s
=0 — S gibt die Anzahl der Leerschritte (ASCII) am Satzanfang an.

3.4.6 Dateien kopieren (COPYD)

Dateien (Datasets) werden mit Hilfe der COPYD-Anweisung kopiert. Dabei beginnt der Kopiervorgang auf beiden beteiligten Dateien an der augenblicklichen Positionierung, d.h. bei noch nicht benutzten Dateien am Anfang. Nach dem Kopieren stehen die Zeiger beider Dateien hinter der EOF-Marke des letzten kopierten Dateiabschnittes.

```
COPYD<,I=idn,O=odn>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

I=idn
=$IN — Lokaler Name der Eingabe-Datei, von der kopiert wird.

O=odn
=$OUT — Lokaler Name der Ausgabe-Datei, die die Kopie aufnimmt.

3.4.7 Ungeblockte Dateien kopieren (COPYU)

Die COPYU-Anweisung kopiert die angegebene Anzahl von (physikalischen) Sektoren oder die ganze Datei bis zum Ende (EOD). Der Kopiervorgang beginnt bei der augenblicklichen Position beider Dateien. Nach Beendigung stehen die Zeiger beider Dateien hinter den letzten kopierten Sektor.

```
COPYU,I=idn,O=odn<,NS=ns>.
```

Parameter:	(die Voreinstellung (default) ist unterstrichen)
I=idn	Name der Datei, die kopiert werden soll.
O=odn	Name der Zieldatei, auf die kopiert werden soll.
NS=ns =1	Dezimale Anzahl der zu kopierenden Sektoren. Wenn die Datei weniger als die angegebene Anzahl von Sektoren enthält, wird der Kopiervorgang bei EOD beendet. Wird nur NS angegeben, so wird bis EOD (Dateiende) kopiert.

3.4.8 Verändern der Zeigerposition (SKIPR, SKIPF, SKIPD, SKIPU)

Die Anweisung SKIPR (Skip Records) überspringt die angegebene Anzahl von Sätzen in der spezifizierten Datei.

```
SKIPR<,DN=dn,NR=n>.
```

SKIPR überspringt keine EOF-Marken. Wird eine EOF-Marke erreicht, bevor die angegebene Anzahl von Sätzen erreicht wurde, so bleibt die Datei hinter dieser EOF-Marke positioniert.

Die Anweisung SKIPF (Skip Files) überspringt die angegebene Anzahl von Dateiabschnitten der spezifizierten Datei. Lag die Zeigerposition in der Mitte eines Dateiabschnittes, so zählt dieser als ganzer Dateiabschnitt.

```
SKIPF<,DN=dn,NF=n>.
```

Die SKIPD-Anweisung (Skip Dataset) positioniert die angegebene Datei an das Ende (EOD), d. h. hinter die letzte EOF-Marke der Datei.

```
SKIPD<,DN=dn>.
```

Die SKIPU-Anweisung (Skip Unblocked) überspringt die angegebene Anzahl von Sektoren.

```
SKIPU<,DN=dn,NS=ns>.
```

Parameter:	(die Voreinstellung (default) ist unterstrichen)
DN=dn =$IN	Lokaler Name der Datei.
NR=n =1	Dezimale Anzahl von Sätzen, die übersprungen werden sollen. Wird NR ohne Angabe eines Wertes spezifiziert, so wird die Datei hinter der letzten EOR-Marke des aktuellen Dateiabschnittes positioniert. Ist n negativ, so wird rückwärts gesprungen.
NF=n	Dezimale Anzahl von Dateiabschnitten, die übersprungen werden sollen.

=1	Wird NF ohne Angabe eines Wertes spezifiziert, so wird hinter die letzte EOF-Marke positioniert. Ist n negativ, so wird rückwärts gesprungen.
NS=ns	Dezimale Anzahl von Sektoren, die übersprungen werden sollen.
=1	Wird NS ohne Angabe eines Wertes spezifiziert, so wird die Datei hinter den letzten Sektor positioniert. Ist n negativ, so wird rückwärts gesprungen.

3.5 Behandlung permanenter Dateien

3.5.1 Abspeichern (SAVE)

Eine lokale Datei wird mit Hilfe der Anweisung SAVE abgeschlossen und permanent auf CRAY-Platten gespeichert; die Positionierung bleibt unverändert. Eine permanente Datei ist eindeutig gekennzeichnet durch ihren Namen, evtl. eine zusätzliche Identifikation und eine Versionsnummer.
Beachten Sie bitte, daß permanente Benutzerdateien auf der CRAY im ZIB spätestens 5 Tage nach dem Anlegen automatisch gelöscht werden; für eine langfristige Datenhaltung stehen im ZIB die beiden Vorrechner zur Verfügung. Verwenden Sie dazu z.B. DISPOSE oder PUTFE (==> 4.1.2, ==> 2.3).

Zugriffsberechtigungen können bei SAVE angegeben werden; sie können aber auch später mit MODIFY (für alle Benutzer) oder mit PERMIT (für einzelne Benutzer) verändert werden. Die Zugriffskontrolle auf die permanente Datei erfolgt in zwei Stufen: zum einen kann der erlaubte Zugriffsmodus für einzelne Benutzer bzw. für die Allgemeinheit mit dem PAM-Parameter (Public Access Mode) definiert werden, möglich sind "None" oder eine Kombination der Werte "Execute only", "Read only", "Write only" und "Maintenance only". Zum anderen kann für jede der drei Zugriffsarten "Read", "Write" und "Maintenance" ein separates Kennwort definiert werden, das jeder Benutzer, auch der Eigentümer der Datei, beim Aufruf angeben muß.

```
SAVE,DN=dn<,PDN=pdn,ID=id,ED=ed,PAM=mode,NA,^
     R=rd,W=wt,M=mn,TA=opt,NOTES=notes>.
```

Parameter:	(die Voreinstellung (default) ist unterstrichen)
DN=dn	Lokaler Name der Datei.
PDN=pdn	Ein- bis fünfzehnstelliger Name der permanenten Datei.
=dn	
ID=id	Ein- bis achtstellige zusätzliche Identifikation der Datei zur weiteren Gruppierung von Dateien.
ED=ed	Edition Number: Versionsnummer, die der Datei beim Abspeichern zugeordnet wird (1 bis 4095).
=1	Wird ED nicht angegeben, so wird für eine neue Datei der Wert 1 eingesetzt;
=ed+1	wenn die Datei bereits existiert, wird ed um 1 erhöht.
PAM=mode	Public Access Mode: Allgemeine Zugriffsrechte zu dieser Datei für *alle* Benutzer:
=N	No Public Access: keine allgemeinen Zugriffsrechte.
=E	Execute only: Datei darf nur ausgeführt werden.
=R	Read only: Datei darf nur gelesen werden.
=W	Write only: Datei darf nur beschrieben werden.
=M	Maintenance only: Kataloginformation der Datei darf geändert und die Datei auch gelöscht werden.

Werden verschiedene Zugriffsarten benötigt, so müssen sie einzeln angegeben werden, z.B.: PAM=R:W. Die Verwendung von E schließt jedoch alle anderen Zugriffsrechte aus.

NA — No Abort: Der Job wird nicht abgebrochen, wenn eine Fehlerbedingung auftritt, z.B. falls die Datei nicht existiert.

R=rd — Read-Password: Definition eines Lese-Kennwortes (1-8 alphanumerische Zeichen). Das Lesekennwort der permanenten Datei mit der höchsten Versionsnummer gilt auch für alle Dateien mit niedrigerer Versionsnummer.

W=wt — Write-Password: Das Schreib-Kennwort (1-8 alphanumerische Zeichen) gilt für alle Versionen einer Datei.

M=mn — Maintenance Control Word: Das Wartungs-Kennwort (1-8 alphanumerische Zeichen) muß für alle nachfolgenden Versionen einer Datei angebeben werden.

TA=opt
=NO — Track Accesses: ein Zähler kann ein- (TA=YES) oder ausgeschaltet (TA=NO) werden, der die Aufrufe anderer Benutzer zählt. Die Anzahl wird in AUDIT aufgelistet.

NOTES=notes — In der Zeichenkette NOTES können zusätzliche Informationen angegeben werden. Die Informationen werden in AUDIT aufgelistet. Bei einem Programm kann z.B. der Aufruf erklärt werden. Die Zeichenkette kann bis zu 480 Zeichen lang sein.

3.5.2 Zugriff (ACCESS)

Permanente Dateien werden mit Hilfe der Anweisung ACCESS einem Job lokal zugeordnet. Nach ACCESS kann die Datei unter dem lokalen Namen angesprochen werden. ACCESS überprüft, ob die Datei existiert und der zugreifende Benutzer die erforderlichen Zugriffsrechte hat.

ACCESS,DN=dn<,PDN=pdn,ID=id,ED=ed,OWN=userid,NA,UQ,R=rd,W=wt,M=mn>.

Parameter: (die Voreinstellung (default) ist unterstrichen)

DN=dn — Ein- bis siebenstelliger lokaler Name der Datei.

PDN=pdn
=Ldn — Ein- bisfünfzehnstelliger Name der permanenten Datei.

ID=id — Zusätzliche Identifikation der Datei (1-8 alphanumerische Zeichen) zur weiteren Gruppierung von Dateien.

ED=ed — Versionsnummer der Datei, auf die zugegriffen werden soll (1 bis 4095). Wird ED nicht angegeben, so wird auf die Datei mit der höchsten existierenden Versionsnummer zugegriffen.

OWN=userid — Ownership Value: Soll eine Datei eines anderen Benutzers angesprochen werden, so muß OWN angegeben werden. Für diese Datei müssen entsprechende Zugriffsrechte definiert sein.

UQ — Unique Access: Es wird der exklusive Zugriff auf die Datei angefordert. Wenn auf eine Datei geschrieben oder die Datei mit DELETE gelöscht werden soll, so muß der Job die Datei mit UQ im Zugriff haben. Wird UQ nicht angegeben, so kann die Datei von mehreren Jobs gleichzeitig gelesen werden.

Beachten Sie bitte, daß Ihr Job ggf. in der Ausführung unterbrochen wird, wenn Sie eine permanente Datei mit exklusivem Zugriff anfordern, die zu diesem Zeitpunkt von einem anderen Job verwendet wird!

NA — No Abort: Der Job wird nicht abgebrochen, wenn eine Fehlerbedingung auftritt, z.B. falls die Datei nicht existiert.

R=rd — Read-Password: Wenn beim Anlegen einer Datei mit SAVE ein Lese-Kennwort angegeben wurde, so muß dieses Kennwort bei ACCESS auch angegeben werden, um die Datei lesen zu können.

W=wt — Write-Password: Um auf eine Datei schreiben zu können, die beim SAVE mit einem Schreib-Kennwort geschützt wurde, muß der W-Parameter und der UQ-Parameter angegeben werden. Ein ADJUST benötigt einen Schreibzugriff.

M=mn — Maintenance Control Word: Das Wartungs-Kennwort ist anzugeben, wenn es beim SAVE angegeben wurde. Um eine Datei zu löschen, muß der M-Parameter zusammen mit dem UQ-Parameter angegeben sein.

3.5.3 Löschen (DELETE)

Lokale permanente Dateien:

Mit DELETE können permanente Dateien gelöscht werden. Vorausgesetzt ist, daß der Job die Datei im exklusiven Zugriff (Unique Access, ==> 3.5.2) und der Benutzer die Erlaubnis zum Löschen hat.

```
DELETE,DN=dn<,NA>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

DN=dn — Lokaler Name einer Datei, die gelöscht werden soll und mit Unique Access im Zugriff ist.

NA — No Abort: Der Job wird nicht abgebrochen, wenn eine Fehlerbedingung auftritt, z.B. falls die Datei nicht existiert.

Nicht lokale permanente Dateien:

Mit dem DELETE-Format für nicht lokale Dateien können permanente Dateien auch dann gelöscht werden, wenn der Benutzer sie nicht im Zugriff hat.

```
DELETE,PDN=pdn<,NA,ERR,MSG,ID=id,OWN=owner,ED=ed,M=mn>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

PDN=pdn — Name der permanenten Datei, die gelöscht werden soll.

NA — No Abort: Der Job wird nicht abgebrochen, wenn eine Fehlerbedingung auftritt, z.B. falls die Datei nicht existiert.

ERR — Error Message: wird dieser Parameter angegeben, werden Meldungen über einen Fehler-Abbruch unterdrückt

MSG — Termination Message: Unterdrückt die Ende-Meldungen.

ID=id — Zusätzliche Identifikation der Datei (1-8 alphanumerische Zeichen) zur weiteren Gruppierung von Dateien.

OWN=owner — Besitzer der permaneneten Datei. Wird das DELETE-Kommando nicht vom Besitzer der zu löschenden Datei verwendet, ist der Parameter anzugeben und eine Zugriffsberechtigung muß vorliegen.

ED=ed — Edition Number: Voreinstellung ist die höchste vorhandene Versionsnummer. Folgende Angaben sind möglich:

=vorzeichenlose Integer (z.B. ED=2: die Version 2 wird gelöscht).
=negative Integer: alles Löschen mit Ausnahme der ed-höchsten Versionen. (z.B. ED=-2: mit Ausnahme der zwei höchsten Versionen).
=Positive Integer: die ed-höchsten Versionen. (z.B. ED=+2: die zwei höchsten Versionen werden gelöscht)
=ALL alle Versionen werden gelöscht.

M=mn — Maintenance Control Word: das Wartungskennwort muß angegeben werden, wenn für die Datei ein Wartungs-Kennwort eingerichtet ist.

3.5.4 Zugriffsrechte definieren (PERMIT)

Die Anweisung PERMIT erlaubt einem Benutzer, explizit anzugeben, wer (außer ihm selbst) seine permanenten Dateien ansprechen kann. PERMIT bezieht sich auf alle Versionen einer Datei. Die Datei braucht bei der Ausführung von PERMIT dem Job nicht lokal zur Verfügung zu stehen. Mit PERMIT erteilte Zugriffsrechte ändern für den angegebenen Benutzer die bei SAVE oder MODIFY mit PAM für alle Benutzer vergebenen Rechte.

```
PERMIT,PDN=pdn<,ID=id,AM=m,RP,USER=userid,NA>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

PDN=pdn — Name einer existierenden permanenten Datei.

ID=id — Ein- bis achtstellige zusätzliche Identifikation der Datei.

AM=m — Access Mode: Zugriffsberechtigung für andere Benutzer. Folgende Modi sind erlaubt:

=N — No public access: keine Zugriffsrechte:
=E — Execute only: Datei darf nur ausgeführt werden.
=R — Read only: Datei darf nur gelesen werden.
=W — Write only: Datei darf nur beschrieben werden.
=M — Maintenance only: Kataloginformation der Datei darf geändert und Datei gelöscht werden.

Werden mehrere verschiedene Zugriffsarten benötigt, so müssen sie einzeln angegeben werden, z.B. AM=R:W:M.

RP — Remove Permit: Die Zugriffsrechte werden dem angegebenen Benutzer entzogen; entspricht der Angabe von AM=N.

USER=userid — Benutzer, für den die Zugriffsrechte definiert werden.

NA — No Abort: Der Job wird nicht abgebrochen, wenn eine Fehlerbedingung auftritt, z.B. falls die Datei nicht existiert.

3.5.5 Kataloginformation ändern (MODIFY)

Die MODIFY-Anweisung ändert die Kataloginformation einer permanenten Datei, die bei SAVE oder vorhergehendem MODIFY angegeben wurde. Der Job muß die Datei exklusiv im Zugriff (unique access) haben und der Benutzer muß zur Änderung der Kataloginformation berechtigt sein. Ein ausgeführtes MODIFY gilt für alle weiteren Versionen einer Datei.

```
MODIFY,DN=dn<,PDN=pdn,ID=id,ED=ed,PAM=mode,NA^
       R=rd,W=wt,M=mn,NOTES=notes>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

DN=dn	Lokaler Name der Datei, deren Kataloginformation geändert werden soll.
PDN=pdn	Ein- bis fünfzehnstelliger permanenter Name der Datei.
ID=id	Ein- bis achtstellige Identifikation der Datei.
ED=ed	Die Datei soll eine andere Versionsnummer erhalten.
PAM=mode	Public Access Mode: Zugriffsberechtigung zu dieser Datei für alle Benutzer. Folgende Modi sind erlaubt:
=N	No Public Access. keine Zugriffsrechte.
=E	Execute only: Datei darf nur ausgeführt werden. **Achtung:** E schließt alle anderen Parameter aus!!
=R	Read only: Datei darf nur gelesen werden.
=W	Write only: Datei darf nur beschrieben werden.
=M	Maintenance only: Kataloginformation der Datei darf geändert und die Datei gelöscht werden. Werden mehrere verschiedene Zugriffsarten benötigt, so müssen sie einzeln angegeben werden, z.B. PAM=R:W.
NA	No Abort: Der Job wird nicht abgebrochen, wenn eine Fehlerbedingung auftritt, z.B. falls die Datei nicht existiert.
R=rd	Neues Lese-Kennwort. Fehlt der Parameter, so gilt das bestehende Kennwort. Wird nur R angegeben, so wird das alte Kennwort gelöscht.
W=wt	Neues Schreib-Kennwort.
M=mn	Neues Wartungs-Kennwort.
NOTES=notes	In der Zeichenkette NOTES können zusätzliche Informationen mit angegeben werden. Die Informationen können im AUDIT mit aufgelistet werden. Bei einem Programm kann z.B. der Aufruf erklärt werden. Die Zeichenkette kann bis zu 480 Zeichen groß sein.

3.5.6 Kataloginformation auflisten (AUDIT)

Die Anweisung AUDIT gibt Informationen über permanente Dateien eines Benutzers aus. Wird mehr als ein Parameter angegeben, so werden nur die Dateien aufgelistet, die alle Kriterien erfüllen. AUDIT gibt die Namen der permanenten Dateien, ihre Identifikation, Versionsnummern, Dateigröße, allgemeine Zugriffsrechte, Erstellungsdatum, letzten Zugriff und erteilte Zugriffsberechtigungen aus.

AUDIT<,L=ldn,PDN=pdn,ID=id,SZ=dsz,OWN=ov,LO=opt:opt:...>.

Parameter:	(die Voreinstellung (default) ist unterstrichen)
L=ldn =$OUT	Name der Datei, auf die die Kataloginformationen geschrieben werden.
PDN=pdn	Name der permanenten Datei, über die die Information aufgelistet werden soll.
ID=id	Ein- bis achtstellige zusätzliche Identifikation der Datei, mit der die Datei gekennzeichnet ist.
SZ=dsz =0	Es werden nur die Dateien gelistet, die größer oder gleich der angegebenen Länge (in Worten) sind.
OWN=ov	Ownership Value: Listet alle Dateien des Benutzers ov. Fehlt der Parameter, so werden die eigenen Dateien gelistet. Es werden nur die Dateien aufgelistet, für die der Benutzer Zugriffsrechte besitzt.
LO=opt:opt:...	List Options: geben an, welche Informationen aufgelistet werden sollen:
=S	Short List: nur Name und Versionsnummer. (S kann nicht in Verbindung mit anderen Optionen benutzt werden)
=L	Long List: ausführliche Liste.
=P	Permit List: Angaben über vergebene Zugriffsrechte
=A	Access Tracking: Die Liste enthält Informationen über andere zugreifende Benutzer mit Zugriffsanzahl und Zeit des letzten Zugriffs, sofern beim Anlegen TA=YES gesetzt wurde.
=N	Note List: Auflisten des "Note"-Feldes.
=X	Extended Long List: Erweiterte ausführliche Liste.

Beispiel:

```
   AUDIT.
AUDIT              COS 1.14            04.09.86  09:50:37            PAGE      1

  OWN = ZZZuserid

-------------------------------------------------------------------------------
PDN                 ID          ED                LAST     LAST     LAST     DEVICE
        SZ     RT     ACC TA PAM   CREATED ACCESSED MODIFIED  DUMPED
-------------------------------------------------------------------------------

$LOOPLIB                              1  27.11.85 28.11.85          14.07.86  DD-A1-25
      4608      2       4 N  R       11:42:09 10:26:06          10:09:56

$LOOPPRE                              1  26.11.85 28.11.85                    DD-A1-24
     57344      2       8 N  E       11:52:46 10:23:25

BESYSTEM                              1  03.09.86 03.09.86                    DD-A1-22
     54580      2       2 N  N       09:58:40 10:04:12

JOBINDE                               1  01.09.86 01.09.86                    DD-A1-24
       561      2       2 N  N       14:54:43 14:54:56

PTEST                                 1  04.09.86 04.09.86                    DD-A1-22
      7680      2       1 N  R       09:49:48 09:49:48

S                                     1  29.08.86 29.08.86                    DD-A1-22
      1899      2       2 N  N       14:27:30 14:28:19

VVV                                   1  02.09.86 02.09.86                    DD-A1-22
       211      2       2 N  N       10:01:56 10:03:11

        7 DATASETS,       250 BLOCKS,              126883 WORDS
```

4. Zugang zur CRAY über die Vorrechner

Der Benutzer kann nicht direkt auf die CRAY X/MP des ZIB zugreifen; der Zugang ist nur moglich über zwei Vorrechner, eine CD CYBER 170-825, betrieben unter dem Betriebssystem NOS/BE (und später auch unter NOS/VE) und eine Siemens 7.865, betrieben unter dem Betriebssystem MVS. Beide Vorrechner sind in Rechnernetze (BERNET/DFN bzw. EARN) integriert und bieten damit auch dem entfernt arbeitenden Benutzer die Möglichkeit, Jobs und Dateien zur CRAY zu senden und Ergebnislisten, Dateien und Statusinformationen zu erhalten.

Auf beiden Vorrechnern arbeiten Software-Produkte, die sogenannten "Stations", die Stapelauftrage entgegen nehmen, Listen und Mitteilungen liefern und den Austausch von Dateien vornehmen. Die jeweilige Station ist dem Wirtssystem angepaßt: Unter NOS/BE z. B. kommuniziert sie unmittelbar mit den Eingabe- und Ausgabewarteschlangen dieses Betriebssystems.

4.1 Kommandos zum Dateitransfer zwischen CRAY und Vorrechner

Die Plattenlaufwerke der CRAY dienen der schnellen Ein-/Ausgabe und nicht der langfristigen Datenhaltung. Der Betrieb des ZIB löscht daher regelmäßig alle auf den CRAY-Platten abgelegten langfristigen Benutzerdateien. Zur Zeit (erstes Halbjahr 1987) werden normal große Dateien fünf Tage nach dem Anlegen, extrem große Dateien (>10000 Sektoren bzw. > 40 Mbyte) nach einem Tag gelöscht. Der Benutzer sollte langfristige Dateien daher nur auf einem der beiden Vorrechner ablegen.

Zur leichteren Handhabung von Dateitransfers wurde die Prozedur TRANSFER geschaffen, mit deren Hilfe Jobs und Daten zwischen MVS-Anlage, CRAY und CD ausgetauscht werden können. Die dafür notwendigen Kommandoprozeduren werden automatisch generiert, so daß der Anwender kaum Kenntnisse der verschiedenen Kommandosprachen benötigt. Diese Schrift erhält man durch:

```
DOC,UTILITIE,TRANSFER
```

4.1.1 Dateitransfer zur CRAY

Mit den Standard-CRAY-Kommandos FETCH und ACQUIRE sowie dem ZIB-Kommando GETFE werden Dateien zur CRAY übertragen (Besonderheiten ==> 4.2.4).

FETCH und ACQUIRE sind flexibel aber aufwendig in der Bedienung. FETCH stellt eine Datei als lokale Datei auf der CRAY bereit; nach Ende des Jobs steht sie auf der CRAY nicht mehr zur Verfügung. ACQUIRE sucht zunächst unter den permanenten Dateien auf der CRAY nach einer Datei, die den geforderten Angaben (PDN, ID, ED) entspricht; ist sie gefunden, wird ein ACCESS (==> 3.5.2) durchgeführt; wird sie nicht gefunden, wird wie beim FETCH die Datei vom Vorrechner zur CRAY transferiert und auf der CRAY unter den genannten Bezeichnungen permanent gemacht (vgl. SAVE, ==> 3.5.1).

Standard-COS-Kommandos FETCH und ACQUIRE

```
FETCH,DN=dn,TEXT='text'<,MF=mf,DF=df,SDN=sdn,TID=tid>.

ACQUIRE,DN=dn,TEXT='text'<,MF=mf,DF=df,PDN=pdn,TID=tid,ID=id,^
       ED=ed,R=rd,W=wt,M=mn,OWN=own,UQ,PAM=mode>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

DN=dn Name (1 bis 7 stellig), unter dem die Datei auf der CRAY verfügbar ist.

TEXT='text' 'text' enthält die Kommandos zur Erzeugung des Transferjobs auf dem Vorrechner. Für die CRAY ist 'text' nur eine Zeichenkette, die nicht weiter interpretiert wird, jedoch an den Vorrechner zur Ausführung übertragen wird. Die Syntax dieser Zeichenkette ist deshalb in den entsprechenden Abschnitten (NOS/BE ==> 4.2.4, MVS ==> 4.3.3) aufgeführt.

MF=mf
=CD
=IB Mainframe: Bezeichnung des Rechners, von dem die Datei übertragen wird (2 Zeichen). Wird MF nicht angegeben, so ist dies der Vorrechner, von dem der Job zur CRAY gesendet wurde. Die CYBER 825 hat die Bezeichnung 'CD', die MVS-Anlage Siemens 7.865 'IB'. Das Übertragen einer Datei vom MVS-Vorrechner ist nur möglich, wenn der CRAY-Job auch von dort abgeschickt wurde; in diesem Fall ist unbedingt der Parameter TID zu setzen.

DF=df Dataset-Format. Dieser Parameter bestimmt das Übertragungsformat.

=CB Character Blocked: Es wird erwartet, daß die Datei codiert vorliegt und ein übliches Zeilenformat besitzt. Beim Transfer wird vom Vorrechner eine Zeichenkonvertierung durchgeführt, (NOS/BE: DISPLAY bzw. MVS: EBCDIC in ASCII), vorrechnertypische Dateimarken (/EOF oder *EOR) werden in COS 'END-OF-FILE'-Kennzeichen umgewandelt und die üblichen CRAY-Block- und Satzkennzeichen hinzugefügt.

=BB Binary Blocked: Es wird keine Zeichenkonvertierung vorgenommen, jedoch die üblichen CRAY-Block- und Satzkennzeichen hinzugefügt.

=TR Transparent: die Datei wird ohne Konvertierung zur CRAY geschickt.

SDN=sdn
=dn Staged Dataset Name (nur bei FETCH und nur bei NOS/BE). Dateiname, unter dem die Datei in den NOS/BE Steuerkarten angesprochen wird (maximal 7 Zeichen lang).

TID=tid Terminal Identifier. Dieser Parameter ist vorrechnerabhängig (NOS/BE ==> 4.2.4, MVS ==> 4.3.3).

PDN=pdn
=dn Name der permanenten Datei (nur bei ACQUIRE, 1- bis 15- stellig). Die ersten 7 Zeichen bilden den NOS/BE-Dateinamen.

ID=id zusätzliche Identifikation der Datei (nur bei ACQUIRE, vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)

ED=ed Versionsnummer (nur bei ACQUIRE, vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)

R=rd Kennwort für die Leseerlaubnis (nur bei ACQUIRE, vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)

W=wt — Kennwort für die Schreiberlaubnis (nur bei ACQUIRE, vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)

M=mn — Wartungs-Kennwort (nur bei ACQUIRE, vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)

UQ — Unique Access: Alleinzugriff (nur bei ACQUIRE, vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)

OWN=ov — Ownership value: Benutzerkennung; nur bei ACQUIRE, vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)
Achtung: Ist eine Datei auf der CRAY nicht vorhanden und wird daher vom Vorrechner übertragen, so wird sie anschließend unter der Kennung des Benutzers eingetragen und die Usernumber wird zu Ownership Value; die Angabe von OWN wird bei SAVE ignoriert. Ein erneuter Aufruf von ACQUIRE mit Angabe von OWN wird dann abermals zum Laden der Datei vom Vorrechner und abermals zum permanenten Ablegen (SAVE) führen.

PAM=mode — Public Access Mode: allgemeine Zugriffserlaubnis (nur bei ACQUIRE, vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)

Die Anweisung FETCH ist insbesondere dann der Anweisung ACQUIRE vorzuziehen, wenn man auf jeden Fall auf die am Vorrechner gespeicherte Datei zugreifen will, z.B. weil sie eventuell einen aktuelleren Stand als die vielleicht an der CRAY noch vorhandene Datei gleichen Namens hat. Außerdem sollte man Dateien an der CRAY nur dann permanent machen, wenn man sie in mehreren aufeinander folgenden Jobs benötigt.

Die Anweisung ACQUIRE sollte verwendet werden, wenn an einem Tag mehrere Programmläufe mit dieser Datei (in unveränderter Version) vorgesehen sind. Man beachte, daß das jeweilige Übertragen der Datei mehr Zeit kostet als der Zugriff auf eine permanente Datei an der CRAY.
Anmerkung: In einer Sekunde können transparent maximal 2 Mbit (60 Sektoren bzw. 500 Pru's) oder codiert maximal 200 Kbit (6 Sektoren oder 50 Pru's) übertragen werden.

Das COS-Kommando GETFE des ZIB

Das COS-Kommando GETFE (GET from Front-End: Datei vom Vorrechner übertragen) ist ein vom ZIB geschaffenes bequemeres Werkzeug als die CRAY-Kommandos FETCH und ACQUIRE, allerdings mit vermindertem Leistungsumfang: Der Benutzer spart die mühsame und fehleranfällige Erstellung der transferspezifischen Angaben (z.B. Erstellen eines vollständigen NOS/BE-Jobs) im CRAY-Textfeld. Zur Zeit arbeitet GETFE nur mit dem NOS/BE-Vorrechner; eine Erweiterung der Dienste auf den MVS-Vorrechner ist vorgesehen. GETFE kann nur eine Datei aus einer NOS/BE Family laden; es arbeitet ähnlich wie ACQUIRE, d.h. GETFE stellt dem CRAY-Job eine lokale Datei zur Verfügung. Existiert die Datei bereits als permanente Datei an der CRAY, wird diese angesprochen. Existiert die Datei nicht auf der CRAY, so wird sie vom Vorrechner übertragen, auf der CRAY permanent gemacht und dem Job zur Verfügung gestellt. GETFE bricht den Job ab, wenn die Datei lokal schon existiert oder wenn auf die Datei nicht zugegriffen werden kann. Lassen Sie sich nicht durch die Vielzahl der Parameter abschrecken. Für fast alle Anwendungen sind geeignete Voreinstellungen gewählt!

```
GETFE,DN=dn<,NOLOG,NOSAVE,NEW,PDN=pdn,ID=id,M=m,R=r,W=w,OWN=own,UQ,PAM=pam,^
      MF=mf,DF=df,CDATNR=cdatnr,CDACCOUNT=cdaccount,CDJOBPW=cdjobpw,^
      CDFAM=cdfam,CDFAMID=cdfamid,CDIFN=cdifn,CDFAMPW=cdfampw.>
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

DN=dn	Name der lokalen Datei
NOLOG	Der Protokolleintrag vom Vorrechner wird nur im Fehlerfall in das CRAY-Protokoll kopiert.
NOSAVE	Die Datei wird auf der CRAY nicht permanent gemacht (entspricht FETCH). Ist der Parameter nicht angegeben, wird die Datei permanent angelegt (entspricht ACQUIRE).
NEW	Die Datei wird in jedem Fall erst vom Vorrechner geholt. Ist der Parameter nicht angegeben, wird zunächst nach einer auf der CRAY vorhandenen permanenten Datei mit den angegebenen Kennungen gesucht und diese ggf. dem Benutzer zur Verfügung gestellt.
PDN=pdn =dn	Name der permanenten Datei (1- bis 15- stellig).
ID=id	zusätzliche Identifikation der Datei (vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)
R=rd	Kennwort für die Leseerlaubnis (vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)
W=wt	Kennwort für die Schreiberlaubnis (vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)
M=mn	Wartungskennwort (vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)
UQ	Unique Access: Alleinzugriff (vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)
OWN=ov	Ownership Value: Benutzerkennung, (vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)
=usernumber	Achtung: Ist diese Benutzerkennung verschieden von der Benutzernummer (Usernumber, ==> 3.2.2) und ist diese Datei auf der CRAY nicht vorhanden (und wird daher vom Vorrechner geholt), so wird abweichend vom ACQUIRE kein SAVE für diese Datei ausgeführt.
PAM=mode	Public Access Mode: Allgemeine Zugriffsrechte. (vgl. ACCESS ==> 3.5.2 und SAVE ==> 3.5.1)
MF=mf	Angabe des Vorrechners, von dem die Datei geholt werden soll. Ist dieser Parameter nicht angegeben, so wird die Datei von dem Vorrechner geholt, von dem der CRAY-Job gestartet wurde. Achtung: zur Zeit kann GETFE nur Dateien von der CYBER 825 holen!
DF=df	Dataset-Format: bestimmt das Übertragungs-Format
=CB	Character Blocked: Es wird erwartet, daß die Datei codiert vorliegt und ein übliches Zeilenformat besitzt. Beim Transfer wird vom Vorrechner eine Zeichenkonvertierung durchgeführt, (NOS/BE: DISPLAY bzw. MVS: EBCDIC in ASCII), vorrechnertypische Dateimarken (/EOF oder *EOR) werden in COS 'END-OF-FILE'-Kennzeichen umgewandelt und die üblichen CRAY-Block- und Satzkennzeichen hinzugefügt.
=BB	binary blocked: Es wird keine Zeichenkonvertierung vorgenom-

men, jedoch werden die üblichen CRAY-Block- und Satzkennzeichen hinzugefügt.

=TR transparent: die Datei wird ohne Konvertierung zur CRAY geschickt. (ACHTUNG: Andere Voreinstellung als bei FETCH!)

Parameter für den NOS/BE-Job

CDATNR=cdatnr Auftragsnummer für den intern erzeugten NOS/BE-Job; fehlt dieser Parameter, so wird die Auftragsnummer aus dem CRAY-Jobnamen entnommen.

CDACCNT=cdaccnt Accountnumber: Abrechnungsnummer für den intern erzeugten NOS/BE-Job, wenn für diese Auftragsnummer eine eigene Abrechnungsnummer notwendig ist; fehlt dieser Parameter, so wird sie für Benutzer aus der FU Berlin aus der CRAY-Abrechnungsnummer gebildet.

CDJOBPW=cdjobpw Jobkennwort für den NOS/BE-Job: Fehlt der Parameter, wird das Abrechnungskennwort des CRAY-Jobs verwendet.

CDFAM=cdfam Family, aus der die Datei von dem CD-Vorrechner geholt werden
=CRAYFAM soll.

CDIFN=cdifn Name der indirekten Datei in der Family cdfam, die vom CD-
=id_pdn Vorrechner geholt werden soll. Wird der Name des indirekten Files aus id_pdn gebildet, so werden dabei für id und pdn nichtalphanumerische Zeichen auf '_' abgebildet. Ausnahme: ein dabei entstandenes '_' an erster Stelle wird auf '4' abgebildet. Entstehen mehrere '_' aufeinanderfolgend, so wird jedes zweite '_' auf '4' abgebildet.

=pdn wenn keine id, sondern nur pdn angegeben wurde.

CDFAMID=cdfamid Identifikation der Family auf dem CD-Vorrechner
=cdatnr

CDFAMPW=cdfampw Kennwort der Family cdfam. Benötigt die Family mehrere Kenn-
=cdjobpw worte, so müssen diese in Hochkommata eingeschlossen und durch Kommata getrennt werden.

Beispiel: die Binärdatei LOOP wird aus der Family CRAYBIN geholt.

```
GETFE,DN=LOOP,MF=CD,DF=TR,CDFAM=CRAYBIN,CDFAMID=8888,^
CDATNR=8888,CDJOBPW=pw-825.
```

bzw. unter Ausnutzung der Voreinstellungen:

```
GETFE,DN=LOOP,CDFAM=CRAYBIN,CDJOBPW=pw-825.
```

4.1.2 Dateitransfer zum Vorrechner

Dateien, die auf der CRAY bearbeitet wurden (z.B. Ausgabelisten, Programmdaten oder Jobs) können zu einem der Vorrechner übertragen werden. Auf diese Weise können einerseits CRAY-Dateien beim Vorrechner permanent gespeichert werden, Ausgabelisten gedruckt oder über DFN (BERNET) bzw. EARN weitergeleitet werden, andererseits können Jobs in die Eingabe-Warteschlange des Vorrechners eingereiht werden. Die Übertragung dieser Dateien erfolgt mit einem der Kommandos DISPOSE, PUTFE oder LFTRANS.

Das Standard-COS-Kommando DISPOSE ist flexibel aber unbequem in der Handhabung (vgl. FETCH ==> 4.1.1). Der Benutzer stellt im Parameter TEXT alle Transferinformationen bereit. Die beiden Kommandos PUTFE und LFTRANS sind einfacher in der Handhabung, können z. Zt. jedoch nur für den Transfer zu NOS/BE-Vorrechnern verwendet werden. PUTFE dient zum Ablegen der Datei in eine Family an der CYBER 825 (analoges Kommando zu GETFE), LFTRANS dient zum Übertragen der Datei mittels BERNET bzw. DFN zu einer entfernt stehenden NOS/BE Anlage. Dort kann die übertragene Datei als permanente Datei, als indirekte Datei einer Family oder in der Ausgabe-Warteschlange abgelegt werden.

Das Standard-COS-Kommando DISPOSE

Das Standard-COS-Kommando DISPOSE dient zum flexiblen Übertragen der Dateien von der CRAY zum Vorrechner.

```
DISPOSE,DN=dn,TEXT='text'<,MF=mf,DF=df,DC=dc,SDN=sdn,^
        TID=tid,NOWAIT,NRLS,DEFER>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

DN=dn — Name (1 bis 7 stellig), unter dem die Datei auf der CRAY verfügbar gemacht wird.

TEXT='text' — 'text' enthält die Kommandos zur Erzeugung des Transferjobs auf dem Vorrechner. Für die CRAY ist 'text' nur eine Zeichenkette, die nicht weiter interpretiert wird, jedoch an den Vorrechner zur Ausführung übertragen wird. Die Syntax dieser Zeichenkette ist deshalb in den entsprechenden Abschnitten (NOS/BE ==> 4.2.4, MVS ==> 4.3.3) aufgeführt.

MF=mf
=CD
=IB — Mainframe: Bezeichnung des Rechners, zu dem die Datei übertragen wird (2 Zeichen). Wird MF nicht angegeben, so ist dies der Vorrechner, von dem der Job zur CRAY gesendet wurde. Die CYBER 825 hat die Bezeichnung 'CD', die MVS-Anlage Siemens 7.865 'IB'. Das Übertragen einer Datei zum MVS-Vorrechner ist nur möglich, wenn der CRAY-Job auch von dort abgeschickt wurde; in diesem Fall ist unbedingt der Parameter TID zu setzen.

DF=df — Dataset Format: bestimmt das Übertragungsformat.
=<u>CB</u> — Character Blocked: Beim Transfer wird vom Vorrechner eine Zeichenkonvertierung durchgeführt: ASCII wird in DISPLAY bzw. EBCDIC, COS 'END-OF-FILE'-Kennzeichen werden in vorrechnertypische Dateimarken (/EOF oder *EOR) umgewandelt.
=BB — binary blocked: Es wird keine Zeichenkonvertierung vorgenommen; die üblichen CRAY- Satz- und Blockkennzeichen werden jedoch entfernt.
=TR — transparent: die Datei wird ohne Konvertierung zur CRAY übertragen.

DC=dc — Disposition Code: Verteilungsschlüssel für die Datei.
=<u>PR</u> — Print: Die Datei ist Ausgabeliste und wird auf dem Vorrechner in die Ausgabewarteschlange eingereiht (DF=CB!). Auf dem NOS/BE-Vorrechner erhält sie als Zielrechnerkennung (Destination ID) die Zeichen 1 bis 3 und als Terminal-ID die Zeichen 4 und 5 des Parameters TID; Angaben im Parameter TEXT werden ignoriert. Auf dem MVS-Vorrechner wird die Datei dem Benutzer zugestellt, dessen Benutzerkennung (Usernumber) im TID-Parameter enthalten ist. Die logische Struktur wird dem TEXT-Parameter entnommen, die Voreinstellung lautet SYSOUT=A,

DCB=CREC,M=VBA,LRECL=4092,BLKSIZE=4096 (==> 4.3.3).

=IN — Input: Die Datei soll auf dem Vorrechner in die Eingabewarteschlange eingereiht werden. Auf dem NOS/BE-Vorrechner werden Angaben im Parameter TEXT ignoriert (für MVS ==> 4.3.3).

=ST — Stage: die Datei soll auf dem Vorrechner permanent gehalten werden. Einzelheiten für den Transfer müssen unter dem Parameter TEXT= vom Benutzer angegeben werden (==> 4.2.4 bzw. 4.3.3).

=MT — Magnetic Tape: Die Datei soll am Vorrechner auf Magnetband geschrieben werden; Einzelheiten für den Transfer müssen unter dem Parameter TEXT= vom Benutzer angegeben werden (==> 4.2.4, 4.3.3). Achtung: in der CRAY-Jobkarte muß der Parameter TP gesetzt sein, sonst wird der CRAY-Job abgebrochen.

SDN=sdn
=dn — Staged Dataset Name (nur NOS/BE): Dateiname, unter dem die Datei in den Steuerkarten angesprochen wird (maximal 7 Zeichen lang).

TID=tid — Terminal Identification. Dieser Parameter ist vorrechnerabhängig (NOS/BE ==> 4.2.4, MVS ==> 4.3.3).

NOWAIT — Der CRAY-Job, der das DISPOSE-Kommando abgesetzt hat, arbeitet weiter, ohne auf das Ende des Transfers zu warten. Der COS-Job erhält keine Mitteilung, wenn die Übertragung abgebrochen wird.
Ist NOWAIT nicht angegeben, wartet der Job, bis die Datenübertragung abgeschlossen ist. Bei Abbruch der Übertragung wird auch der wartende COS-Job abgebrochen und ggf. nach einer EXIT-Anweisung fortgesetzt.

NRLS — No Release: gibt an, daß die Datei auch nach der Übertragung auf der CRAY verfügbar bleibt. Während der Datenübertragung kann auf die Datei nicht geschrieben werden. Ist NRLS nicht angegeben, so wird die Datei vom System zurückgegeben (entspricht RELEASE ==> 3.4.2)

DEFER — Die Datei wird erst nach der Freigabe durch ein RELEASE-Kommando (==> 3.4.2) oder bei Jobende übertragen.

Es empfiehlt sich, aufwendig wiederherzustellende Dateien vor der Übertragung auf der CRAY permanent zu machen und nach erfolgreicher Übertragung wieder zu löschen.

Das COS-Kommando PUTFE des ZIB

Das ZIB-Kommando PUTFE sichert eine lokale Datei als permanente Datei auf der CRAY und zusätzlich auf dem Vorrechner. Die lokale Datei bleibt dem Job erhalten. Der Job bricht ab, wenn die lokale Datei nicht existiert oder die lokale Datei nicht auf dem Vorrechner gesichert werden konnte. Gelingt es hingegen nicht, die Datei auf der CRAY permanent zu machen, wird sie trotzdem auf dem Vorrechner gesichert. PUTFE steht zur Zeit nur für den Transfer zum NOS/BE-Vorrechner zur Verfügung. Am Vorrechner CYBER 825 wird die Datei in einer Family abgelegt.

```
PUTFE,DN=dn<,NOLOG,NOSAVE,PDN=pdn,ID=id,M=m,R=r,W=w,UQ,PAM=pam,MF=mf,^
      DF=df,CDATNR=cdatnr,CDACCOUNT=cdaccount,CDJOBPW=cdjobpw,^
      CDFAM=cdfam,CDFAMID=cdfamid,CDIF=cdif,CDFAMPW=cdfampw>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

DN=dn	Name der lokalen Datei
NOLOG	Der Protokolleintrag vom Vorrechner wird nur im Fehlerfall in das CRAY-Protokoll kopiert.
NOSAVE	Die Datei wird auf der CRAY nicht permanent gemacht. Ist der Parameter nicht angegeben, wird die Datei permanent angelegt.
PDN=pdn =dn	Name der permanenten Datei (1- bis 15- stellig).
ID=id	zusätzliche Identifikation der Datei (vgl. ACCESS und SAVE ==> 3.5.2 und 3.5.1)
R=rd	Kennwort für die Leseerlaubnis (vgl. ACCESS und SAVE ==> 3.5.2 und 3.5.1)
W=wt	Kennwort für die Schreiberlaubnis (vgl. ACCESS und SAVE ==> 3.5.2 und 3.5.1)
M=mn	maintenance: Kennwort für die Wartung (vgl. ACCESS und SAVE ==> 3.5.2 und 3.5.1)
UQ	Unique Access: Alleinzugriff (vgl. ACCESS und SAVE ==> 3.5.2 und 3.5.1)
MF=mf	Angabe des Vorrechners, an dem die Datei gesucht werden soll. Ist dieser Parameter nicht angegeben, so wird die Datei zu dem Vorrechner übertragen, von dem der CRAY-Job gestartet wurde. Achtung: zur Zeit kann PUTFE nur Dateien zur CYBER 825 übertragen.
DF=df	Dataset-Format: bestimmt das Übertragungs-Format.
=CB	Character Blocked: Es wird erwartet, daß die Datei codiert vorliegt und ein übliches Zeilenformat besitzt. Beim Transfer wird vom Vorrechner eine Zeichenkonvertierung durchgeführt, (ASCII in CD-DISPLAY bzw. MVS: in EBCDIC), vorrechnertypische Dateimarken (/EOF oder *EOR) werden in COS 'END-OF-FILE'-Kennzeichen umgewandelt und die üblichen CRAY-Block- und Satzkennzeichen hinzugefügt.
=BB	binary blocked: Es wird keine Zeichenkonvertierung vorgenommen, jedoch werden die üblichen CRAY-Block- und Satzkennzeichen hinzugefügt.
=TR	transparent: die Datei wird ohne Konvertierung zum Vorrechner geschickt. (ACHTUNG: Andere Voreinstellung als bei DISPOSE!)
PAM=mode	Public Access Mode. (vgl. ACCESS und SAVE ==> 3.5.2 und 3.5.1)

Parameter für den NOS/BE-Job:

CDATNR=cdatnr	Auftragsnummer für den intern erzeugten NOS/BE-Job. Fehlt dieser Parameter, so wird die Auftragsnummer aus dem CRAY-Jobnamen entnommen.
CDACCNT=cdaccnt	Accountnumber: Abrechnungsnummer für den intern erzeugten NOS/BE-Job, wenn für diese Auftragsnummer eine eigene Abrechnungsnummer notwendig ist; fehlt dieser Parameter, so wird

sie für Benutzer aus der FU Berlin aus der CRAY-Abrechnungsnummer gebildet.

CDJOBPW=cdjobpw — Jobkennwort für den NOS/BE-Job. Fehlt der Parameter, wird als Jobkennwort das Abrechnungskennwort des CRAY-Jobs verwendet.

CDFAM=cdfam
=CRAYFAM — Family, in die die Datei auf dem CD-Vorrechner kopiert werden soll.

CDFAMID=cdfamid
=cdatnr — Identifikation der Family auf dem CD-Vorrechner

CDIF=cdif
=id_pdn — Name der indirekten Datei in der Family cdfam, in die die Datei kopiert werden soll. Wird der Name des indirekten Files aus id_pdn gebildet, so werden dabei für id und pdn nichtalphanumerische Zeichen auf '_' abgebildet. Ausnahme: ein dabei entstandenes '_' an erster Stelle wird auf '4' abgebildet. Entstehen mehrere '_' aufeinanderfolgend, so wird jedes zweite '_' auf '4' abgebildet.

CDFAMPW=cdfampw
=cdjobpw — Kennwort der Family cdfam. Benötigt die Family mehrere Kennworte, so müssen diese in Hochkommata eingeschlossen und durch Kommata getrennt werden.

Beispiel: Sichern des absoluten Lademoduls ABS in der Standard-Family CRAYFAM auf der CYBER 825 unter dem Namen CRAYABS. Der Benutzer verwende auf der CYBER 825 das gleiche Jobkennwort wie auf der CRAY.

```
...
CFT.
LDR,AB=ABS,NX.      besser: SEGLDR,CMD='ABS'.
PUTFE,DN=ABS,PDN=CRAYABS.
...
```

Falls das Jobkennwort auf der CYBER 825 anders lautet als auf der CRAY, muß es bei PUTFE zusatzlich angegeben werden:

```
PUTFE,DN=ABS,PDN=CRAYABS,CDJOBPW=pw-825.
```

Das Kommando LFTRANS des ZIB

Das Kommando LFTRANS dient dazu, eine Datei von der CRAY zu einer über das DFN erreichbaren Berliner NOS/BE-Anlage zu übertragen. Die zu transferierende Datei kann entweder am Zielrechner langfristig als permanente Datei oder als indirekte Datei einer Family abgespeichert oder in einer Eingabe- oder Ausgabe-Warteschlange des Zielrechners abgelegt werden.

Ein ähnliches Kommando für den Transfer einer Datei von der CRAY zu einer entfernt stehenden NOS/VE-Anlage ist in Vorbereitung; verfolgen Sie bitte die aktuellen Hinweise im Kopf der CRAY-Ausgabelisten!

```
LFTRANS<,LF=lf,PF=pf,IF=if,JPW=jpw,JID=jid,ANR=anr,T=t,IO=io,^
       TO=to,ID=id,PW=pw,CP=cp,XR=xr,MTT=mtt,DOJPW=dojpw,^
       DOJID=dojid,DOTID=dotid,PURGE,DC=dc,DCTID=dctid,DF=df>.
```

Lassen Sie sich nicht durch die Vielzahl der Parameter abschrecken. Für fast alle Anwendungen sind geeignete Voreinstellungen vorhanden! Eine vollständige Beschreibung erhält man mit:

DOC,CRUTIL,LFTRANS.

Parameter: (die Voreinstellung (default) ist unterstrichen)

LF=lf
=LOCAL — Name der zu übertragenden Datei.

PF=pf
=lf — Name der passenden NOS/BE-Datei bzw. der Family an der Zielanlage.

IF=if — Name der indirekten Datei für die Family pf am Zielrechner; falls IF angegeben wurde, wird der Parameter PF als Name einer Family interpretiert und die zu transferierende Datei wird in dieser Family unter dem durch IF spezifizierten indirekten Dateinamen am Zielrechner abgelegt. Fehlt dieser Parameter, wird die Datei als permanente Datei abgelegt.

JPW=jpw — Jobpassword: Jobkennwort für die Zielanlage. Die Angabe ist für alle Zielanlagen obligatorisch. Zugleich ist JPW auch das Abrechnungskennwort für den FU-Benutzer. Es ist auch die Angabe APW= möglich.

JID=jid — Jobidentifikation bzw. Auftragsnummer, unter der der Job an der Zielanlage ablaufen soll.
=<Auftragsnr.> Ist der Zielrechner die Anlage der FUB, so dient JID zur Kennzeichnung der Ergebnisliste.

ANR=anr — Accountnumber: Abrechnungsnummer für den zu generierenden NOS/BE Job; dieser Parameter ist nur für Benutzer interessant, deren Job am Zielrechner der FUB eine Abrechnungs-Anweisung benötigt.

T=t
=30 — Rechenzeitschranke in Sekunden für die Jobkarte.

IO=io
=20 — Ein/Ausgabe-Zeitschranke in Sekunden für die Jobkarte.

TO=to — Zielanlage, zu der die Datei übertragen werden soll; zulässige Werte sind:
FUB, (FU3,) TUB, (TU1,) ZIB, (WR8, WR5)
=TUB falls ANR nicht angegeben
=FUB falls ANR angegeben

ID=id — Benutzer-Identifikation an der Zielanlage, unter der die Datei abgelegt werden soll.

PW=pw — Password: Kennwort der permanenten Datei bzw. der Family am Zielrechner; falls mehrere Kennworte notwendig sind, müssen sie gemeinsam in Hochkommata eingeschlossen werden, z.B. PW='GEHEIM,LET'. Die Angabe ist nur sinnvoll, falls bereits ein Datei oder eine Family mit Kennworten versehen am Zielrechner existieren.

CP=cp — Komprimierung: die Datei wird vor dem Übertragen komprimiert. Insbesondere bei Textfiles, Programmquellen, Family's und GRAFIK-METAFILES ergibt sich in der Regel eine erhebliche Verkleinerung; die Übertragungsgeschwindigkeit steigt und die Übertragungsleitung wird entlastet. (==> DOC,SYSTEM,COMPACT.)
=ON Die Datei wird vor der Übertragung komprimiert und am Zielrechner wieder dekomprimiert.

=OFF	keine Komprimierung
=PRE	Komprimierung nur vor der Übertragung (z. B. Sparen von permanenten Plattenplatz am Zielrechner)
XR=xr	Kennworte für das Anlegen einer permanenten Datei (weitere Informationen siehe NOS/BE Reference Manual).
MTT=mtt	Message to Terminal; mit Hilfe dieses Parameters ist es
=JA	möglich, die standardmäßig erzeugte Erfolgs- oder Mißerfolgsmeldung des Dateitransfers an dem durch TO spezifizierten Rechner abzuschalten.
=N	keine Meldung erwünscht.
DOJPW=dojpw	Jobkartenkennwort für die Rückmeldung
=jpw	
DOJID=dojid	Auftragsnummer für die Rückmeldung
=jid	
DOTID=dotid	Terminalkennung für die Rückmeldung
=tid	
PURGE	Nach erfolgreichem Transfer wird am Zielrechner eine ggf. vorhandene alte permanente Datei (LC=1) gleichen Namens gelöscht.
DC=dc	Disposition-Code: Verteilungsschlüssel für die Datei.
=IF	Ablage der Datei lf in die durch den Parameter PF spezifizierte Family unter dem durch den Parameter IF angegebenen indirekten Dateinamen.
=PF	Ablage der Datei als permanente Datei unter dem durch den Parameter PF spezifizierten Namen (falls der Parameter IF nicht angegeben wurde)
=PR	Ablage der Datei lf in der Ausgabe-Warteschlange des Zielrechners unter der durch den Parameter DCTID angegebenen Terminal-ID.
=PU	Ablage der Datei lf in der Ausgabe-Warteschlange PUNCH des Zielrechners unter der durch den Parameter DCTID angegebenen Terminal-ID.
=IN	Ablage der Datei lf in der Eingabe-Warteschlange am Zielrechner.
DCTID=dctid	Terminal-ID, unter der die zu transferierende Datei am Zielrechner abgelegt werden soll.
=C	falls DC=PR oder DC=PU
=tid	falls DC=IN
DF=df	Dateiformat (vgl. DISPOSE, ==> 4.2.4)
=CB	Character Blocked
=BB	Binary Blocked
=TR	Transparent

Beispiel 1.: Übertragen der lokalen Datei TESTFIL von der CRAY an die CD der TUB in eine Family.

```
A7777,STCRY.
JOB,T=20,MFL=1000000.FC=YES,CO=LFTRANS
ACCOUNT,AC=<ac>,APW=<apw>.
...
LFTRANS,LF=TESTFIL,PF=DATENFAM,IF='DATEN_1',JPW=<tub-jpw>,PW='pw1,pw2'.
```

Die auf der CRAY existierende Datei TESTFIL wird zur TUB übertragen und unter der ID=7777 in der Family DATENFAM mit dem Namen DATEN_1 abgelegt. Der Zugriff auf die Family DATEN sei nur mit PW=pw1,pw2 moglich. Die Parameter DO und TID müssen nicht angegeben werden, da der CRAY-Job von der TUB stammt und die Ausgabeliste zur TUB zurück geschickt werden soll.

Beispiel 2.: Übertragen der lokalen Datei TEST an die CD der FUB.

```
A7777,STCRY.
JOB,T=20.FC=YES,CO=LFTRANS
ACCOUNT,AC=<ac>,APW=<apw>.
...
LFTRANS,LF=TEST,PF=CRAYRESULTS,ANR=<ac.FUB>,ID=<ac.FUB>,^
PW='pw1,pw2',TO=FUB,TID=<tid.FUB>,L,PURGE.
...
```

Die Datei soll dort als permanente Datei mit dem Namen CRAYRESULTS unter der ID=<ac.FUB> abgelegt werden. Falls eine permanente Datei gleichen Namens existiert, wird die Version mit der kleinsten Versionsnummer (LC=1) nach erfolgreichem Ablegen (CATALOG) gelöscht. Die Datei CRAYRESULTS sei mit den Schlüsselworten PW=pw1,pw2 geschützt. Sollte keine permanente Datei mit dem Namen CRAYRESULTS existieren, wird beim CATALOG-Kommando zusätzlich ein TK-Kennwort mit TK=pw1 eingesetzt.

4.2 Der NOS/BE Vorrechner

Die Verbindung von der CD CYBER 825 zur CRAY wird auf der CYBER 825 über ein ständig aktives Systemprogramm CRSTAT abgewickelt. Dem Benutzer stehen eine Reihe von Kommandos unter NOS/BE zur Verfügung, um die einzelnen Transferleistungen anzufordern. Diese werden im Folgenden beschrieben:

4.2.1 Abschicken von Jobs an die CRAY

Die CRAY-CD-Station-Software sorgt dafür, daß sämtliche Jobs aus der Eingabe-Warteschlange der CYBER 825, die mit der NOS/BE Zielrechnerkennung (Destination-ID) CRY versehen sind, in die Eingabe-Warteschlange der CRAY transportiert werden. Dazu muß vor den CRAY-Job folgende Steuerkarte geschrieben werden:

```
xnnnn,STCRY.
```

Dabei ist x ein beliebiger Buchstabe zur Identifizierung von verschiedenen Jobs eines Benutzers, nnnn die unter NOS/BE gültige, maximal vierstellige Auftragsnummer des Benutzers. Die Ergebnisliste des Jobs wird, sofern vom Benutzer nicht ausdrücklich anders vermerkt, zur CYBER 825 zurückgelenkt. Der Transfer des Jobs bzw. der Ergebnisliste zwischen CYBER 825 und einem entfernten Rechner erfolgt nach den üblichen Konventionen von BERNET bzw. DFN.

Beispiel: Der Benutzer habe an einer der NOS/BE-Anlagen (FU Berlin, TU Berlin oder ZIB) den folgenden CRAY-Job in der Datei CRAYJOB abgelegt:

```
A1999,STCRY.                          Jobkarte für die CYBER 825 des ZIB
JOB,T=2.                              Jobkarte für die CRAY
ACCOUNT,AC=TUB10331999,APW=pw-cray.   Account-Karte für die CRAY
ACQUIRE,DN=VEKTOR,TEXT='A8888.ACCOUNT,FU2034BB,,pw=825.ATTACH,VEKTOR,'^
    'ID=8888.CTASK.'.                 Anstoß zum Filetransfer
CFT.                                  Aufruf des CRAY-FORTRAN-Compilers
```

```
LDR.                                    besser: SEGLDR,GO. Aufruf des Laders
                                        zum Laden und Ausführen
/EOF                                    CRAY End-of-File (oder CD EOS-Karte)
      PROGRAM SKLPRO
      DIMENSION A(5),B(5)
      READ (*,100) A,B
C     LIEST VON $IN DIE NAECHSTEN BEIDEN EINGABESATZE
100   FORMAT (5F5.2)
      S=0.
      DO 10 I=1,5
10    S = S + A(I)*B(I)
      WRITE (*,200) S
C     SCHREIBT NACH $OUT, $OUT WIRD ZURUCKGESENDET
200   FORMAT (' SKALARPRODUKT ',F10.2)
      STOP
      END
/EOF                                    CRAY End-of-File (oder CD EOS-Karte)
1.0  2.0  3.0  4.0  5.0
1.0  2.0  3.0  4.0  5.0
end-of-file                             Ende der Datei oder CD End-of-File
```

Dieser CRAY-Job kann mit Hilfe des BERNET- (DFN-) Kommandos RJE zur CYBER 825 des ZIB und damit auch zur CRAY übertragen werden; nach Bearbeitung des Jobs wird das Ergebnisprotokoll (die Datei $OUT mit der Übersetzungsliste von CFT, der Ausgabe des Programms und dem Logfile - entsprechend dem Dayfile bei CD) als "Remote Output File" an den Vorrechner übertragen. Der gleiche Aufruf von RJE kann auch verwendet werden, wenn direkt an der CYBER 825 des ZIB gearbeitet wird.

Beispiel:

```
RJE,CRAYJOB. (bei Aufruf an der FUB:  RJE,CRAYJOB,ST=ZIB)
```

Im Rahmen des DFN ist die CYBER 825 des ZIB zur Zeit (Anfang 1987) von ca. 20 Rechenanlagen in der Bundesrepublik Deutschland einschließlich Berlin (West) zu erreichen; neben den schon genannten NOS/BE-Rechnern sind dies insbesondere die NOS/VE-Rechner beim RRZN Hannover, die SPERRY-Anlage bei der GWD Göttingen sowie einige VAX-Rechenanlagen. Einzelheiten dazu finden Sie in ==> 1.3.1 "Deutsches Forschungsnetz" (DFN).

4.2.2 Status-Abfrage

CSTAT - globale Statusinformation

Mit Hilfe des Programms CSTAT kann sich der Benutzer auf der CYBER 825 des ZIB über den aktuellen Status aller Jobs in den verschiedenen Warteschlangen informieren.

Achtung: CSTAT existiert nur an der CYBER 825 des ZIB, wo die Informationen aktuell von der CRAY abgerufen werden. Sollte zum Zeitpunkt des Aufrufs die Verbindung nicht aktiv sein, so konnen keine Statusinformationen ausgegeben werden und CSTAT wird mit einer entsprechenden Meldung beendet.

```
CSTAT<,q,L=lfn>
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

q=I Auflisten aller Jobs in der Eingabe-Warteschlange der CRAY

=E	Auflisten aller Jobs in der Ausführungs-Warteschlange der CRAY
=O	Auflisten aller Jobs in der Ausgabe-Warteschlange der CRAY
=R	Auflisten aller Jobs, die Dateien von der CYBER 825 erhalten
=S	Auflisten aller Jobs, die Dateien zur CYBER 825 senden
=A	Auflisten aller Jobs der CRAY
L=lfn	Die Information wird in die Datei lfn geschrieben.
=<Bildschirm>	(beim Aufruf im Dialog)
=OUTPUT	(beim Aufruf im Stapelbetrieb)

CJOB - Statusinformation eines einzelnen Jobs

Mit Hilfe des Programms CJOB kann man sich auf der CYBER 825 des ZIB ausführlich über den Status von Jobs informieren, die unter der eigenen Kennung auf der CRAY laufen.

```
CJOB,jn<,jsq>
```

Parameter:

jn	Name des Jobs
jsq	Jobsequencenumber: Nummer des Jobs, die man mit CSTAT (==>4.2.2) oder RST (==> 4.4) erhalten kann.

4.2.3 Abbrechen und Beenden von Jobs

Abbrechen von Jobs

Das Programm CDROP erlaubt dem Benutzer von der CYBER 825 des ZIB aus, seinen gerade in Ausführung befindlichen CRAY-Job abzubrechen. Die bisher erzeugte Ausgabe bleibt erhalten. Die Jobausführung wird nach der nächsten EXIT-Anweisung fortgesetzt, falls eine vorhanden ist.

```
CDROP,jsq.
```

jsq	Jobsequencenumber: Nummer des Jobs, die man mit CSTAT (==>4.2.2) oder RST (==> 4.4) erhalten kann.

Beenden von Jobs

Die Wirkung des NOS/BE Programms CKILL hängt vom aktuellen Status des Jobs ab: Ein Job in der Eingabe-Warteschlange wird aus dieser entfernt (Löschen von $IN); ein Job in der Ausführungs-Warteschlange wird abgebrochen, der Benutzer erhält als Ausgabe nur das Protokoll (Logfile). Von einem Job in der Ausgabe-Warteschlange wird die gesamte Ausgabe gelöscht (Löschen von $OUT).

```
CKILL,jsq.
```

jsq	Jobsequencenumber: Nummer des Jobs, die man mit CSTAT (==>4.2.2) oder RST (==> 4.4) erhalten kann.

4.2.4 Besonderheiten des Dateitransfers bei NOS/BE

In den Abschnitten 4.1.1 und 4.1.2 wurden die COS-Kommandos FETCH, ACQUIRE und DISPOSE zum Transfer von Dateien vom bzw. zum Vorrechner allgemein beschrieben. Die Besonderheiten für einen NOS/BE-Vorrechner werden hier aufgeführt:

TEXT-Parameter:

Für jeden Dateitransfer wird auf der NOS/BE-Anlage ein vollständiger Batchjob gestartet; die Steuerkarten für diesen Job müssen vom Benutzer im TEXT-Parameter des COS-Transferkommandos angegeben werden. Die Zeichenfolge wird durch ' (Apostroph) eingeschlossen; man beachte, daß bei Verwendung von Folgekarten das Zeichen für eine Fortsetzung (^) außerhalb der Zeichenkette stehen muß. (==> das folgende Beispiel zu ACQUIRE). Der Parameter text darf maximal 240 Zeichen lang sein, der zu bildende NOS/BE Job darf maximal 8 Steuerkarten enthalten. Die Anweisung CTASK dient dabei zum Kopieren der Datei und muß an der Stelle in den Steuerkarten stehen, an der eine Kopierroutine stehen würde.

```
CTASK,ALL,ASCII8.
```

Parameter:

ALL	Auflisten des CD-Protokolls (Jobdayfile) im CRAY-Protokoll. Keine Angabe: Auflisten des CD-Protokolls (Jobdayfile) im Fehlerfall.
ASCII8	(nur bei DF=CB, codierte Übertragung). Übertragung im CD-ASCII95-Code mit Groß- und Kleinschreibung. Keine Angabe: Übertragung im CD-Display-Code, nur Großschreibung.

Unter NOS/BE wird die Datei unter dem durch SDN festgelegten Namen angesprochen; fehlt SDN, unter dem durch DN festgelegten Namen. In dem zu generierenden Job dürfen nur folgende Steuerkarten vorkommen:

```
ACCOUNT, ATTACH, CATALOG, CTASK, FAMILY, GET, LABEL, LFTRANS, LIMIT,
MOUNT, PURGE, PUT, REPLACE, REQUEST, RST, VSN, XDRAW und XMIT.
```

Die für den Dateitransfer notwendige CD-Jobkarte im Textfeld darf keine Angaben zu T, IO, CM oder ST enthalten; zulässig sind nur die Parameter PW und NT.

TID-Parameter:

Terminal Identificataion. Die CRAY-Terminal-Identification besitzt die Form "ssstt", dabei kennzeichnet "sss" den Herkunftsrechner (die NOS/BE Source-ID) und "tt" die Terminal-ID des zugehörigen NOS/BE Jobs. Wird TID nicht angegeben, so stehen hier die entsprechenden Werte des ursprünglichen NOS/BE Jobs bzw. die Benutzerkennung (User-ID) des MVS-Jobs. Falls der CRAY-Job von der MVS-Anlage abgeschickt wurde und die Datei von einer CD-Anlage übertragen werden soll, muß TID geeignet angegeben werden.

Beispiel zu ACQUIRE:

```
ACQUIRE,DN=VEKTOR,TEXT='A8888.ACCOUNT,FU2034BB,,pw-825.ATTACH,VEKTOR,'^
    'ID=8888.CTASK.'.                    Anstoß zum Filetransfer
```

Das Kommando ACQUIRE sucht zunächst unter den permanenten Dateien mit der Benutzerkennung (ownership value) FU2034BB eine Datei VEKTOR. Ist eine vorhan-

den, wird sie mittels ACCESS dem Job unter dem Namen VEKTOR zur Verfügung gestellt. Ist sie nicht vorhanden, wird auf der CYBER 825 ein Job unter der Kennung des Benutzers gestartet, der die Datei dort sucht (ATTACH), die Blockung und Codierung vom CD-Format in das CRAY-Format umwandelt, sie zur CRAY sendet, dort permanent macht (wie mit einem SAVE,DN=VEKTOR) und dem Programm übergibt.

Beispiel für das Abspeichern einer Datei auf Magnetband am Vorrechner:

```
DISPOSE,DN=MATRIX,DF=TR,DC=MT,TEXT='A7777,NT1,PW=pw-825.'^
      'LABEL,MATRIX,R,L=CRAYMATRIX,D=DE,VSN=W09999,RING.CTASK,ALL.'.
```

Das Kommando DISPOSE sendet an die CYBER 825 einen Job, der die Datei MATRIX (die auch unformatiert sein kann) transparent (d.h. ohne Codekonvertierung und ohne Umblockung) im CRAY-Format auf ein Magnetband schreibt (CD-Tape, 9-Spur, 6250 bpi). Das Programm CTASK führt die eigentliche Übertragung durch.

Beispiel zur Erzeugung eines Folgejobs (Ersatz für SUBMIT):

```
Job1:
   C8999,STCRY.
   JOB.
   ACCOUNT,AC=FU1244AA,APW=cray-pw.
   COPYF,I=$IN,O=JOBB,NF=2.            Kopiere 2. Job in die Datei JOBB
   CFT,B=PROG.
   SAVE,DN=PROG.                       Hebe Binärdatei für 2. Job auf
   LDR,DN=PROG.                        besser: SEGLDR,GO,CMD='BIN=PROG'.
   DISPOSE,DN=JOBB,DC=ST,TEXT='D8999.ACCOUNT,FU1244AA,AA,pw-cd.CTASK.'^
                     'RJE,JOBB,ST=ZIB,DO=FUB.'.
                                       Übertrage 2. Job zur CYBER 825, sende
                                       ihn mittels RJE wieder zur CRAY und
                                       sorge dafür, daß auch die Ergebnisliste
                                       dieses Jobs zurück zur FU findet
   /EOF

Job2:
   E8999,STCRY.
   JOB.
   ACCOUNT,AC=FU1244AA,APW=cray-pw.
   ACCESS,DN=PROG.                     Verwende Binärprogramm vom 1. Job
   LDR,DN=PROG.                        besser: SEGLDR,GO,CMD='BIN=PROG'.
   /EOF
   Daten für den 2. Job
   /EOF
   Fortran-Programm
   /EOF
   Daten für den 1. Job
```

Beispiel für das Übertragen eines Zwischenergebnisses zu einer NOS/BE-Anlage, die über DFN erreichbar ist:

```
...
DISPOSE,DN=TAPE20,DC=PR,TID=TU1CP.
...
```

In diesem Beispiel steht das Zwischenergebnis in der Datei TAPE20 und wird als Ausgabedatei zum Rechner TU1 an die Terminal-ID CP geschickt. Die Datei **muß** im Sinne von DFN druckbare Ausgabe enthalten, im Zweifelsfall verwende man LFTRANS (==> 4.1.2).

Beispiel zur Bildung von Jobketten:

```
K<nnn>,STCRY..
JOB,T=<time>,MFL=<mfl>.
ACCOUNT,AC=<acc>,APW=<apw>.
*  Steuerkarten für den 'eigentlichen' Job
*  Steuerkarten für die Fortsetzung der Jobkette
ACCESS,DN=JOBZAHL,UQ.
DELETE,DN=JOBZAHL.
ASSIGN,DN=JOBZAHL,A=FT54.
ASSIGN,DN=JOBZ,A=FT55.
CFT,L=0.
LDR.                                    besser: SEGLDR,GO.
SAVE,DN=JOBZ,PDN=JOBZAHL.
FETCH,DN=NICHTS,TEXT='C<nnn>,PW=<pw>.FAMILY,FAM,ID=<nnn>.'^
'GET,FOLGE.RJE,FOLGE,DO=TU1,TID=<tid>.CTASK.'.
EXIT.
*  ENDE DER JOBKETTE
/EOF
/EOF
      PROGRAM JOBKET
      READ(54,'(I2)') I
      IF (I+1 .GT. 14) CALL ABORT
      WRITE(6,'(A,I2)') ' LAUF NUMMER : ',I+1
      WRITE(55,'(I2)') I+1
      END
/EOF
```

Bemerkungen:
In der Datei JOBZAHL auf der CRAY wird die laufende Nummer des Jobs mit dem Fortranprogramm JOBKET weitergezählt (der erste Job kreiert die Datei und trägt einen geeigneten Anfangswert ein). Das FETCH- Kommando wird dazu "missbraucht", am Vorrechner den nächsten Job aus der Datei FOLGE der Family FAM per RJE zu starten (das Kommando CTASK im TEXT- Parameter darf nicht fehlen!). Die Ausgabeliste soll an die CD 835 der TUB gesendet werden.

Die Jobkette wird nach dem 14. Job beendet (Subroutine ABORT im Fortranprogramm).

ACHTUNG: Jobketten sind in speziellen Klassen unzulässig (==> 2.3)! Fehler in Jobketten können zu kritischen Betriebssituationen führen. Daher empfehlen wir dringend, vor dem ersten Produktionslauf einen Test mit geeigneten Betriebsmittelangaben bei Tage zu machen, um bei fehlerhaftem Ablauf (z.B. Joblawine) notfalls die Jobkette durch einen Operateur (Tel. (030) 896 04 165) abbrechen zu lassen.

4.3 Der MVS-Vorrechner

Die Verbindung von der MVS-Anlage zur CRAY wird auf der MVS-Anlage über ein ständig aktives Systemprogramm 'CRAY' abgewickelt. Für den Benutzer stehen eine Reihe von Kommandos unter MVS zur Verfügung, um die einzelnen Transferleistungen anzufordern. Diese werden im Folgenden beschrieben. Damit ist es möglich, Jobs und Dateien sowohl von der MVS Anlage zur CRAY als auch in entgegengesetzter Richtung zu schicken.

Die Verbindung zwischen der MVS-Station-Software und dem Benutzer wird im Dialogsystem (TSO) über eigene Kommandos geschaffen. Im Batch steht dafür ein Programm zur Verfügung, das Anweisungen verarbeitet, die in Form und Inhalt

mit dem jeweils entsprechenden TSO-Kommando identisch sind. Das Programm kann über eine Systemprozedur (catalogued procedure) aufgerufen werden.

Dem Anwender stehen folgende Station-Kommandos zur Verfügung:

CRSUBMIT	Übertragung einer sequentiellen Datei (PS-Datei) oder eines Elementes einer untergliederten Datei (Member einer PO-Datei), in der ein oder mehrere vollständige COS-Jobs stehen, zur CRAY, wo der (die) Job(s) in die COS Eingabewarteschlange eingereiht werden.
CRAY	Liefert Informationen über COS-Jobs und Dateien und erlaubt für eigene Jobs Manipulationen wie das Abbrechen des Jobs.

Eine ausführliche Darstellung der Station-Software enthält die Schrift IBM MVS STATION Reference Manual SI 0038 sowie

```
DOC,MVSCRAY,KOMPLETT
```

In ihm werden auf ca. 80 Seiten Wirkungsweise und Kommandos der MVS-Station-Software detailliert beschrieben.

4.3.1 Abschicken von Jobs an die CRAY

Mit dem Kommando CRSUBMIT kann ein Job von der MVS-Anlage zur CRAY geschickt werden. Die Übertragung dieses COS-Jobs erfolgt zweistufig: zunächst wird der Job von der Station zwischengespeichert; erst nach erfolgreichem Verbindungsaufbau erfolgt der eigentliche Jobtransfer zwischen den Anlagen. Ist die Verbindung hergestellt, werden zunächst alle älteren Transferaufträge abgearbeitet, ehe der Job übertragen wird. Der Benutzer kann sich über die zwischengespeicherten Jobs und über die noch zu übertragenden Dateien mit dem Subkommando TRANSFER innerhalb des Kommandos CRAY informieren.

Im ZIB ist für einen COS-Job die MVS-Benutzerkennung anzugeben und ein beliebiger Buchstabe oder eine beliebige Ziffer anzuhängen, wenn sich der MVS-Dialog-Benutzer die Ausgabe des COS Jobs an seinem Terminal ansehen will. Ist der COS-Jobname zwar auf der CRAY gültig, aber von der MVS-Benutzerkennung verschieden, so kann die Ausgabe nicht am Terminal betrachtet werden. Wird ein falscher Jobname verwendet, wird der COS Job vorzeitig abgebrochen mit der Meldung:

```
AC120 - INVALID JOB NAME
```

CRSUBMIT

Ein oder mehrere vollständige COS-Jobs können auf der S 7.865 in einer MVS-Datei beschrieben werden. Dabei ist es belanglos, ob die Jobs in einer sequentiellen Datei oder in einem Element einer untergliederten Datei (Member eines partitioned dataset) zusammengestellt werden. Als Satzformate werden dabei von der Station unterstützt:

```
RECFM = F (fixed), FB (fixed blocked), V (variable), VB (variable blocked),
        U (undefined) (==> 4.3.3)
```

Zwischen einzelnen COS-Jobs in einer Datei muß je eine /EOD-Marke stehen.

```
CRSUBMIT (dsname<(mname)> <DEST(destination) PRINT(classname)
         HOLD/NOHOLD NOTIFY<(userid)>/NONOTIFY APW<(apw)>/NAPW<(napw)>
         UPW<(upw)>/NUPW<(nupw)>>)
```

dsname — Dataset Name: Name einer katalogisierten MVS-Datei, in dem der zu transferierende COS Job steht. Das Datei kann eine sequentielle Datei oder eine untergliederte Datei (partitioned dataset) sein.

(mname) — Member Name: Name eines Elementes einer untergliederten Datei, in dem der zu transferierende Job steht. Wird kein Element-Name angegeben, wird TEMPNAME eingesetzt.

DEST(destination) Bestimmungsort für die Ausgabe des Jobs.

PRINT(classname) SYSOUT Klassenname für die Druck-Ausgabe des COS Jobs (Datei $OUT). Die hier angegebene Klasse kann im TEXT-Parameter einer im Job befindlichen DISPOSE-Anweisung verändert werden. Wird PRINT nicht angegeben, so wird die Ausgabe in die im ZIB voreingestellte Klasse 'A' gelenkt.

HOLD — Die Job-Ausgabe wird in einer Warteschlange gehalten, damit sich der Dialog-Benutzer die Ausgabe am Terminal ansehen kann. Eine Druckausgabe erfolgt nur mittels Kommando OUTPUT oder im SPF-Menue 3.8.

NOHOLD — Die Job-Ausgabe kommt nicht in eine Warteschlange. Wird vom Benutzer weder HOLD noch NOHOLD angegeben, so wird NOHOLD als Voreinstellung eingesetzt.

NOTIFY<(userid)> Benutzerkennung des MVS Benutzers, der die von der MVS-Station ausgegebenen Meldungen des Jobs erhalten soll. Wird keine Benutzerkennung angegeben, so wird die Kennung desjenigen Benutzers eingesetzt, der das CRSUBMIT-Kommando gab.

APW<(apw)> — account password (1 bis 15 alphanumeriche Zeichen): soll in die ACCOUNT-Anweisung des COS-Jobs eingetragen werden. Wird 'apw' nicht angegeben, so wird im Dialog (jedoch nicht im Batch!) der Benutzer zur Eingabe des Kennwortes aufgefordert.

NAPW<(napw)> — new account password (1 bis 15 alphanumerische Zeichen): soll in die ACCOUNT-Anweisung des COS Jobs eingetragen werden.

UPW<(upw)> — user password (1 bis 15 alphanumerische Zeichen): soll in die ACCOUNT-Anweisung des COS-Jobs eingetragen werden. Wird 'upw' nicht angegeben, so wird im Dialog (jedoch nicht im Batch!) der Benutzer zur Eingabe des Kennwortes aufgefordert.

NUPW<(nupw)> — new user password (1 bis 15 alphanumerische Zeichen): soll in die ACCOUNT-Anweisung des COS-Jobs eingetragen werden.

Beispiel für das Senden eines Jobs an die CRAY im *Dialog:*

Ein COS-Job, der in einer sequentiellen Datei steht, soll zur CRAY übertragen werden.

Inhalt der Datei 'ZIB122.CRAYJOB.TEST' :

```
JOB,JN=ZIB122P.
ACCOUNT,AC=COSUSER77,US=COSNUM,APW=NEU,UPW=XYZ.
* vollständiger COS-Job
```

Um den COS-Job von der MVS-Anlage zur CRAY zu schicken, setzt der Benutzer mit der Kennung ZIB122 folgendes Dialog-Kommando ab:

```
CRSUBMIT CRAYJOB.TEST
```

Wegen der Voreinstellungen wirkt es wie das Kommando

```
CRSUBMIT CRAYJOB.TEST PRINT(A) NOHOLD NOTIFY(ZIB122)
```

Dieselbe Leistung erhält man durch Aufruf des Kommandoprozessors CRAY (==> 4.3.2) und die Eingabe von:

```
SUBMIT CRAYJOB.TEST   oder, abgekürzt:    SU CRAYJOB.TEST
```

darauf erhält man die Rückmeldung:

```
'CSS235I JOB ZIB122P (xxxxx) QUEUED FOR TRANSFER'
```

wobei 'xxxxx' die von der Station vergebene Transfernummer 'trn' (==> 4.3.2) ist.

Beispiel für das Senden des Jobs aus einer Datei an die CRAY im *Batch:*

```
//ZIB122C    JOB,'CRAYJOB IM BATCH',TIME=1,MSGLEVEL=(1,1),
//           MSGCLASS=P,PASSWORD=<pw>
//*********************************************************
//COSJOB     EXEC CRAY
//SYSPRINT   DD   SYSOUT=*
//SYSIN      DD   *
  CRSUBMIT   CRAYJOB.TEST
/*
//
```

Beispiel für das Senden des vollständigen Jobs an die CRAY im *Batch:*

Der folgende Batchjob erbringt die gleiche Leistung, wobei auf die Verwendung der sequentiellen Datei 'ZIB122.CRAYJOB.TEST' verzichtet wird. Der CRAY-Job ist hier als Datenbereich im MVS-Job enthalten:

```
//ZIB122C    JOB,'CRAYJOB IM BATCH',TIME=1,MSGLEVEL=(1,1),
//                MSGCLASS=P,PASSWORD=YYYYY
//*********************************************************
//COSJOB     EXEC CRAY
//SYSPRINT   DD   SYSOUT=*
//SYSUT1     DD   *
JOB,JN=ZIB122P.
ACCOUNT,AC=COSUSER77,US=COSNUM,APW=NEU.
      ... vollständiger COS-Job
          .
          .
          .
/*
//SYSIN      DD   *
  CRSUBMIT
/*
//
```

Beispiel für das Senden des vollständigen Jobs an die CRAY über *EARN:*

Die MVS-Anlage des ZIB ist unter der Kennung DBOZIB21 von anderen Rechenanlagen, die ebenfalls an EARN bzw. BITNET angeschlossen sind, zu

erreichen. Über EARN kann man nur im Batch auf die MVS-Anlage zugreifen, insoweit gelten beide Batch-Beispiele auch für den Zugang über EARN. Diese Jobs müssen wie folgt ergänzt werden:

Ein Standardjob für den EARN-Benutzer zum Senden eines COS-Jobs an die CRAY hat den folgenden Aufbau:

```
//<userid>xx  JOB 0,...                    <userid>=6-stellige Userid an der MVS-
                                           Anlage des ZIB, gefolgt von einem oder
                                           zwei Zeichen (A,...,Z,0,...,9)
/*ROUTE XEQ DBOZIB21                       Die ROUTE-XEQ-Karte ist nur dann zu
                                           verwenden, wenn auf dem Ausgangsrechner
                                           des Jobs das MVS/JES2 läuft
/*ROUTE PRINT <homenode>.<homeuserid>      Diese ROUTE PRINT-Karte wird empfohlen,
                                           wenn der Ausgangsrechner unter VM-
                                           System läuft. Es sind einzusetzen:
              <homenode>=                  Knotenname des Ausgangsrechners,
              <homeuserid>=                Die eigene Benutzerkennung am Ausgangs-
                                           rechner
//<stepname> EXEC CRAY                     <stepname>=Beliebige alphanum. Zeichen-
                                           folge, beginnend mit einem Buchstaben.
//SYSPRINT   DD   SYSOUT=*
//SYSUT1     DD   *
COS-Job
/EOF
/*
//SYSIN      DD   *
CRSUBMIT                                   CRSUBMIT ist die Direktive für die
                                           CRAY-Station Software.
/*
```

Die vom COS-Job erzeugte Ausgabeliste wird in diesem Standardjob an das gleiche Ziel geschickt wie die Ausgabe des MVS-Jobs, wenn der COS-Job keine DISPOSE-Steuerkarte für die Ausgabedatei $OUT enthält.

Wird auf dem Rechner, von dem der Job kam, ein VM-System betrieben, so kann der Benutzer die Ausgabeliste des COS- und MVS-Jobs mit dem RECEIVE-Kommando in eine Datei in seinem Benutzerbereich übertragen, wenn er die ROUTE-PRINT-Karte wie oben angegeben verwendet hat. Ohne diese Karte gelangen beide Ausgabelisten auf den Zentraldrucker des RSCS, welches die EARN-Leitungen bedient. Die Option NOH (NOHEADER) muß angegeben werden, damit das RSCS der Jobkarte keinen eigenen Header voranstellt, da das zu einem Fehler auf dem MVS-Zielrechner führen würde!

Enthält der MVS-Job von einem MVS-Rechners keine ROUTE-PRINT-Karte, so gehen COS- und MVS-Ausgabelisten an diejenige Stapelstation zurück, an der der Job eingelesen wurde. Wurde der Job durch SUBMIT im Dialog gesendet, geht die Ausgabeliste auf einen lokalen Drucker des Ausgangsrechners.
Weitere Infomationen, z.B. zum Absenden eines Jobs für ein MVS-System von einem VM-System aus, sind in DOC,EARN,EARNJOBS enthalten.

4.3.2 Status-Abfrage und Steuerungsmöglichkeiten

Mit dem Dialog-Kommandoprozessor CRAY können Kommandos abgesetzt werden, mit deren Hilfe ein Dialog-Benutzer seine COS-Jobs und COS-Dateien überwachen und teilweise steuern kann. Es ist möglich, sich über Transferaufträge zu informieren, diese abzubrechen oder, falls sie noch unerledigt sind, zurückzunehmen.

Das Kommando CRAY

Das Kommando CRAY steht jedem Benutzer nach LOGON zur Verfügung. Voraussetzung für die gewünschten Leistung ist jedoch, daß die MVS-Station-Software auf der MVS Anlage aktiv ist und - bei den meisten Subkommandos erforderlich - Verbindung zur CRAY hat.

Nach Eingabe des Dialog-Kommandos 'CRAY' meldet sich der CRAY-Kommandoprozessor mit

```
CSS256I CRAY ENVIRONMENT ENTERED
- CSS 2.01      - <Datum Uhrzeit>
```

und anschließend, je nach Zustand der Verbindung zwischen MVS und COS:

```
CSS757I STATION IS NOT LOGGED ON TO THE CRAY
```

oder

```
CSS257I STATION IS LOGGED ON TO THE CRAY
CSS258I STATION ID: IB, STATION TYPE: BOTH
```

Danach wird mit der Meldung des Kommandoprozessors die Eingabe verlangt (Prompting):

```
CRAY:
```

Das Subkommando wird eingegeben. Danach meldet sich erneut der Kommandoprozessor:

```
CRAY:
```

Subkommandos von CRAY

Folgende Kommandos können innerhalb des Kommandoprozessors aufgerufen werden, wobei das Subkommando TRANSFER auch bei nicht bestehender Verbindung zur CRAY aktiviert werden kann; alle anderen Subkommandos arbeiten nur bei aktiver Verbindung zwischen MVS-Anlage und CRAY. Der Parameter jsq steht jeweils für die Jobsequencenumber des eigenen CRAY-Jobs; man erhält sie entweder mit dem Subkommando STATUS oder über das ZIB-Kommando RST (==> 4.4).

Abfragen, ob eine bestimmte Datei auf der CRAY als permanente Datei existiert:

```
DATA<SET> pdn <ID(id)> <ED(ed)> <OV(ov)>
```

Abbrechen eines eigenen COS-Jobs, die bisher erzeugte Ausgabeliste $OUT wird zugestellt. Anweisungen im Job werden bis zur nächsten EXIT-Anweisung -falls vorhanden- übergangen, sonst bis zum Jobende):

```
DROP jsq
```

Information über einen <bestimmten> eigenen COS-Job:

```
JOB jobname <jsq>
```

Information über den Jobstatus eines <bestimmten> eigenen COS-Jobs:

```
JSTAT <jsq>
```

Sofortiges Abbrechen eines eigenen COS-Jobs: die bisher erzeugte Ausgabeliste wird vernichtet.

```
KILL jsq
```

Eintragen von Nachrichten in das Protokoll eines eigenen COS-Jobs:

```
MES<SAGE>      JOB     jobname jsq 'message'
               BOTH
```

Sofortiges Abbrechen der Verarbeitung eines eigenen Jobs und Zurückstellen des Jobs in die Eingabe-Warteschlange

```
RERUN <jsq>
```

Statusanzeige von COS-Jobs, COS-Eingabe- und Ausgabedateien:

```
STAT<US>  Q(<queueliste>)
```

Setzen und Löschen von logischen Schaltern (sense switches) eines COS-Jobs:

```
SWI<TCH> jsq ssw       ON
                       OF<F>
```

Absenden eines Jobs zur CRAY; die möglichen Parameterwerte entsprechen denen von CRSUBMIT (==> 4.3.1).

```
SUBMIT
```

Mit dem Subkommando TRANSFER Q trn kann man sich über seinen eigenen Job mit der Transfernummer 'trn' informieren, solange dieser auf der MVS-Anlage in der Transferwarteschlange gehalten wird.
Mit 'CRAYCMD TRANSFER CANCEL trn' kann der Dialog-Benutzer einen eigenen Transferauftrag aus der Warteschlange zurücknehmen (löschen).

```
TRANSFER       Q trn
               CANCEL trn
```

4.3.3 Besonderheiten des Dateitransfers bei MVS

In den Abschnitten 4.1.1 und 4.1.2 wurden die COS-Kommandos FETCH, ACQUIRE und DISPOSE zum Transfer von Dateien allgemein beschrieben. Die Besonderheiten für einen MVS-Vorrechner sind hier aufgeführt.

Der aktuelle Wert des Parameters TEXT wird vom Betriebssystem COS der MVS-Station übergeben und von dieser als Operand einer MVS-Dateidefinitions-Anweisung interpretiert. Diese Anweisung beschreibt die Datei, auf die zugegriffen werden soll. Die Länge der im TEXT-Parameter übergebenen Zeichenfolge beträgt maximal 240 Zeichen ohne die begrenzenden Apostrophe. Ein Leerzeichen darin wird als Endekennzeichen der Zeichenfolge interpretiert. TEXT ist ein obligater Parameter, wenn von COS aus auf Dateien der MVS Anlage zugegriffen werden soll.

Dateidefinitionsanweisung (DD-Statement)

Im Folgenden wird eine Auswahl der am häufigsten benutzten Parameter aufgelistet, die von der MVS-Station in einer Dateidefinitionsanweisung unterstützt werden. Alle in der Liste angegebenen Werte können in den COS-Anweisungen FETCH, ACQUIRE und DISPOSE dem TEXT-Parameter als aktuellen Wert übergeben werden.

In der Schrift OS/VS2 MVS JCL (Bestellnr. GC28-1300-2) sind die hier benutzten Bezeichnungen ausführlich erklärt.

Anzahl der Kopien beim Drucken:

```
COPIES=copies
```

Mit dem Dateisteuerblock DCB wird der Aufbau einer Datei beschrieben:

```
DCB=(BLKSIZE=blksize,DEN=den,DSORG=dsorg,LRECL=lrecl,RECFM=recfm)
```

Parameter:

BLKSIZE=blksize	Blockgröße: min = 18, max = 32760 Bytes. Bei variabel geblockten Sätzen (VB/VBA) muß die Blockgröße die Satzlänge um mindestens 4 Bytes überschreiten.
DEN=den	Density: Schreib-/Lesedichte für Magnetbänder in Byte/Zoll
DSORG=dsorg	Data Set Organization: Dateiorganisationsformen sind hauptsächlich Physical Sequential (PS) und Partitioned Organized (PO).
LRECL=lrecl	Logical Record Length: Satzlänge.
RECFM=recfm	Record Format: Satzformat.
=U	undefinierte Länge
=V	variable Länge
=F	feste Länge
=VB	variabel geblockt
=FB	fest geblockt
=VBA	variabel geblockt mit ANSI-Vorschubsteuerzeichen
=FBA	fest geblockt mit ANSI-Vorschubsteuerzeichen

Auswahl eines Ausgabegerätes, z.B. Remote Drucker R1 oder R3 oder Schnelldrucker LOCAL; kann nur zusammen mit dem SYSOUT-Parameter benutzt werden:

```
DEST=dest
```

Zustand und Zustandsänderung einer Datei:

```
DISP=(status<,normal-disposition,abnormal-disposition>)
```

Der Positionsparameter status kann folgende Werte annehmen:

NEW	wenn die Datei neu angelegt werden soll
OLD	wenn die Datei schon vorhanden ist; exclusiver Schreibzugriff auf die Datei.
SHR	Share: Lesezugriff auf sequentielle Dateien; Lese- und Schreibzugriff auf untergliederte Dateien
MOD	Modification: Anfügen an eine bestehende sequentielle Datei

Positionsparameter für normal-disposition und abnormal-disposition: Zustand nach normalem oder nach fehlerhaftem Jobende.

KEEP	Wenn Status Old: Datei soll nach Jobende erhalten bleiben;
DELETE	Wenn Status NEW: Datei soll nach Jobende gelöscht werden.

CATLG	Catalog: Datei soll im System-Katalog eingetragen werden.
UNCATLG	Katalog-Eintrag soll gelöscht werden.

Ein Dateiname kann sich aus mehreren Dateinamen zusammensetzen. Jeder einzelne Name darf nicht länger als acht Zeichen sein; das erste Zeichen jedes Namens muß ein Buchstabe oder eines der Zeichen $,#,@ sein. Die Namen werden durch Punkte voneinander getrennt. Insgesamt darf der Dateiname einschließlich Punkten maximal 44 Zeichen lang sein. Beispiel: ZZZ611.CRAY.AUSGABE.JOB

```
DSNAME=dsname
```

In einer untergliederten Datei muß der Membername (=Elementname) mit angegeben werden. Der Name des Elements darf nicht länger als acht Zeichen sein; sein erstes Zeichen muß ein Buchstabe oder eines der Zeichen $,#,@ sein.

```
DSNAME=dsname(membername)
```

Beispiele:

```
DISP=(SHR)
DISP=(NEW,CATLG,DELETE)
DISP=(OLD,KEEP)
```

Bandmarke für Magnetbanddateien:.

```
LABEL=(dst-sqnb,la)
```

Positionsparameter:

dst-sqnb	Data Set Sequence Number: relative Position einer Datei auf dem Band.
la	Label-Art: SL bedeutet IBM-Standard-Label; NL bedeutet ohne Label.

Angaben zum Platz, den eine Datei auf der Magnetplatte belegt:

```
SPACE=(länge,(pq,sq,dir),RLSE,CONTIG)
```

Positionsparameter:

länge	Angabe der Einheiten durch TRK (Tracks) oder CYL (Cylinder) oder eine Zahlenangabe für die Blocklänge
pq	Primary Quantity: Menge der benötigten Einheiten (w.o.) beim Anlegen einer Datei
sq	Secondary Quantity: Speichererweiterung um die angegebene Menge von Einheiten bei Überlauf
dir	Directory: nur für untergliederte Datei: reserviert den für das Inhaltsverzeichnis benötigten Platz
RLSE	Release: Freigeben des nicht benötigten Speicherplatzes am Ende des Jobs (optional)
CONTIG	Contiguous: es wird ein zusammenhängender Speicherbereich angelegt. Dies bezieht sich nur auf die erste Reservierung (Primary Allocation) des Speicherplatzes (optional)

Beispiele:

Untergliederte Datei anlegen:

```
SPACE=(TRK,(10,10,10),RLSE)
```

sequentielle Datei mit zusammenhängendem Speicherbereich anlegen:

```
SPACE=(TRK,(100,10),RLSE,CONTIG)
```

Mit SYSOUT wird einer Datei eine Ausgabeklasse zugewiesen. Der Parameter SYSOUT darf nicht in Kombination mit DSNAME, LABEL, VOLUME oder UNIT verwendet werden.

```
SYSOUT=sysout
```

sysout=A	die Datei wird sofort gedruckt.
=0	die Datei kann an einem Sichtgerät angesehen werden. Sie wird am nächsten Tag gelöscht.
=P	die Datei kann an einem Sichtgerät angesehen werden. Sie wird am nächsten Tag gedruckt.
=E	die Datei kann an einem Sichtgerät angesehen werden. Sie wird nur dann gelöscht, wenn Platzmangel besteht.

Angabe des Speichermediums:

```
UNIT=unit
```

unit=3350	Plattentyp mit max. Blocklänge von 19069 Bytes.
=3380	Plattentyp mit max. Blocklänge von 32760 Bytes.
=DISK	beliebiger Plattentyp
=TAPE	beliebiges Bandgerät
=SYSDA	beliebiges Gerät mit wahlfreiem Zugriff

Logischer Gerätename und Zugriffskriterien:

```
VOLUME=(PRIVATE,RETAIN,SER=ser)
```

Positionsparameter:

PRIVATE	Bedeutung nur für Bandgeräte; optionaler Parameter; Erläuterung siehe RETAIN:
RETAIN	muß angegeben werden, wenn ein Band in mehr als einem Jobstep benötigt wird und bis zum Ende des letzten Jobsteps auf dem Bandgerät verbleiben soll.
SER=ser	Angabe eines Platten- oder Bandnamens; Zur Zeit sind die Platten DISK01 bis DISK14 verfügbar.

Beispiele:

```
VOL=(,RETAIN,SER=MYTAPE)
VOL=(SER=DISK05)
```

Standardwerte des TEXT-Parameters im DISPOSE-Statement

Die zu einem COS-Job gehörende Ausgabedatei $OUT wird, wenn nichts anderes angegeben wurde, zu der MVS- Station zurückgeschickt, von der der COS-Job stammt; dort wird die Datei dem Job Entry Subsystem (JES2) übergeben. Das JES2 teilt die Ausgabeklasse in MVS entsprechend der Parametereinstellung zu.
Wird im DISPOSE-Statement ein DC-Parameter (disposition code) angegeben und keine Angabe zu TEXT gemacht, so wird nur die Voreinstellung wirksam.

In der folgenden Tabelle sind die Voreinstellungen notiert:

COS Job System Datei	COS disp. code (dc)	Datei Format (df)	Voreingestellter Wert im TEXT-Feld des DISPOSE
$OUT	PR	CB	SYSOUT=A,DCB=(RECFM=VBA, LRECL=4092,BLKSIZE=4096)
$PUNCH	PU	CB	SYSOUT=B,DCB=(RECFM=FB, LRECL=80,BLKSIZE=3120)
$PLOT	PT	CB	SYSOUT=Y,DCB=(RECFM=FB, LRECL=133,BLKSIZE=2660)
	ST	CB	keine Voreinstellung, TEXT-Parameter muß angegeben werden
	MT	CB	keine Voreinstellung, TEXT-Parameter muß angegeben werden
	IN	CB	SYSOUT=(A,INTRDR)

Die hier gezeigten Voreinstellungen können überschrieben werden durch die Angabe des PRINT, PUNCH bzw. PLOT Parameters im CRSUBMIT. Wenn ein 'disposition code' DC=PR/PU/PT angegeben wird und im TEXT-Parameter weiter Angaben folgen, so muss SYSOUT=... im TEXT angegeben werden.

Terminal-Kennung: Voreingestellt ist die Übernahme der MVS-Benutzerkennung des Benutzers, der den COS Job abgeschickt hat. Die Voreinstellung kann im CRSUBMIT durch NOTIFY(userid) geändert werden.

```
TID=tid
```

Beispiel zu FETCH und ACQUIRE, Zugriff auf eine permanente Datei:

```
ACQUIRE,DN=LOCAL,PDN=IBMFILE,ID=COSUID,ED=8,R=NICHTS,¬
        W=ZU,M=MACHEN,UQ,TEXT='DSN=ZIB999.JCL.CNTL,DISP=SHR'.
```

Dabei stehen im TEXT-Parameter alle Angaben, die zum Auffinden der MVS-Datei erforderlich sind.
Hinweis: Fortsetzungszeichen auf der MVS Anlage für COS JCL ist das logische 'nicht'-Zeichen X'5F'.

FETCH oder ACQUIRE auf eine katalogisierte Datei:

```
JOB,JN=...
  :
FETCH,DN=LOCAL,TEXT='DSN=ZIB999.JCL.CNTL,DISP=SHR'.
```

oder

```
JOB,JN=...
  :
ACQUIRE,DN=LOCAL,PDN=IBMFILE,ID=COSUID,ED=8,R=NICHTS,¬
  W=ZU,M=MACHEN,UQ,TEXT='DSN=ZIB999.JCL.CNTL,DISP=SHR'.
```

Beispiel: Es soll eine lokale COS-Datei mittels DISPOSE auf der MVS-Anlage in eine schon bestehende Datei eingetragen werden. Dabei soll eine weitere Datei in eine bereits katalogisierte untergliederte Datei geschrieben werden, bzw. an das Ende einer sequentiellen Datei angefügt werden. Die sequentielle Datei

sei nicht katalogisiert, sie soll auf der Platte DISK08 liegen (UNIT=3380).

Bei Eintrag in die untergliederte Datei (partitioned Dataset):

```
DISPOSE,DN=LOADMOD,DC=ST,DF=BB,TEXT='DSN=ZIB999.JOB.LIB(GOGO),DISP=OLD'.
```

Bei Eintrag in die sequentielle Datei (an das Ende anfügen):

```
DISPOSE,DN=LOADMOD,DC=ST,DF=BB,TEXT='DSN=ZIB999.JOB.MODS,DISP=MOD,' ¬
        'UNIT=3380,VOL=SER=DISK08'.
```

4.4 Remote Status Information RST

RST ist ein vom ZIB erstelltes Programm, das den Benutzer über den Status von Jobs auf den Rechnern der eigenen oder anderer Institutionen informiert, u.a. auch über den Status von Jobs auf der CRAY. Die Information ist in der Regel nicht älter als fünf Minuten. RST kann an folgenden Rechnern aufgerufen werden:

FUB:	CYBER 845
TUB:	CYBER 835/830
ZIB:	CYBER 825, MVS-Anlage

Eine Bereitstellung auch auf den NOS/VE-Rechnern ist in Vorbereitung; beachten Sie bitte die aktuellen Informationen (z.B. in DOC,CRAY,NEWS).

Im folgenden werden nur die Parameter von RST beschrieben, die im Zusammenhang mit dem Status von CRAY-Jobs benötigt werden. (Eine vollständige Beschreibung erhält man mit: DOC,SYSTEM,RST.)

```
RST,JID=jid,DC=dc,MF=mf,DMF=dmf,SMF=smf,TID=tid,US=us,CLASS=class,L=l.
```

Parameter: (die Voreinstellung (default) ist unterstrichen; * bedeutet, daß alle möglichen Parameterwerte bei der Ausgabe berücksichtigt werden.)

JID=jid	Jobname oder Auftragsnummer:
=*	keine Einschränkung bzgl. des Jobnamens oder der Auftragsnummer.
DC=dc	Disposition Code: führe nur Jobs mit dem angegebenen Verteilungsschlüssel auf:
=IN	nur Eingabe-Warteschlange
=EX	nur Ausführungs-Warteschlange
=*	Eingabe- und Ausführungs-Warteschlange
MF=mf	Mainframe: führe nur Jobs auf, die sich zur Zeit des Aufrufs auf der angegebenen Anlage befinden:
=CRY	CRAY
=WR8	CYBER 825
=*	beliebig
SMF=smf	Source Mainframe: führe nur Jobs auf, die von folgender Anlage abgesendet wurden:
=FU	FUB CYBER 845
=GW	GWD Göttingen
=HO	RRZN
=IB	ZIB S7.865 (MVS)
=NE	sonstige BERNET/DFN Anlage
=TU	TUB CYBER 835

=WR	ZIB CYBER 825 (NOS/BE)
=*	keine Einschränkung bzgl. der Source Mainframes. (Bezeichnungen: Stand April 1987)
TID=tid	Führe nur Jobs mit der angegebenen (NOS/BE) Terminal-ID auf
=*	keine Einschränkung bzgl. der Terminal-ID.
DMF=dmf	Führe nur Jobs auf, die zu folgender Anlage (Destination Mainframe) gesendet werden sollen:
=FU	FUB CYBER 845
=GW	GWD Göttingen
=HO	RRZN
=IB	ZIB S7.865 (MVS)
=NE	sonstige BERNET/DFN Anlage
=TU	TUB CYBER 835
=WR	ZIB CYBER 825 (NOS/BE)
=*	keine Einschränkung bzgl. der Destination Mainframes. (Bezeichnungen: Stand April 1987)
US=us	Usernumber: Führe nur CRAY-Jobs mit der angegebenen Benutzerkennung auf
CLASS=class	Führe nur Jobs in der angegebenen Klasse auf
=*	keine Einschränkung bzgl. der Klasse
L=l	Dateiname für die Ausgabe;
=<Bildschirm>	(beim Aufruf im Dialog)
=OUTPUT	(beim Aufruf im Stapelbetrieb)

Bei der Angabe der Werte für den Auswahlparameter wird das Zeichen * zum Darstellen eines beliebigen Zeichens verwendet. Sterne am Ende eines Parameterwertes können weggelassen werden. Der Aufruf von RST ohne Parameter bewirkt (abweichend von den oben beschriebenen Standardwerten) den Aufruf:

RST,SMF=<Anlage des Aufrufenden>,TID=<Terminal ID des Aufrufenden>.

Bei der Verwendung von RST **auf der MVS-Anlage** des ZIB gelten folgende zusätzliche Hinweise:

RST ist als Dialog-Kommando bereitgestellt.

Voraussetzung für die erfolgreiche Ausführung des Kommandos ist die volle Betriebsbereitschaft der Kopplungen zwischen CYBER 825 und CRAY sowie zwischen CRAY und MVS-Anlage.

Da die Statusinformation von der CYBER 825 geholt wird, entsteht zwischen dem Kommandoaufruf und der Informationsausgabe eine Wartezeit von ca. 30 Sekunden. Wenn die Information nach 120 Sekunden noch nicht eingetroffen ist, wird das Kommando mit einer entsprechenden Meldung abgebrochen.

Ein Aufruf im Batch ist über die Standardschnittstelle zum Aufruf von Dialog-Kommandos im Batch möglich.

Folgende Abweichungen sind auf der MVS-Anlage zu beachten:

- Der Parameter L (listfile) wird nicht voll unterstützt. Wenn L=string angegeben wurde, wird die lokale Arbeitsdatei auf der MVS-Anlage nicht gelöscht. Eine weitergehende Auswertung des als 'string' angegebenen Dateinamens erfolgt nicht. Der aus Datum und Uhrzeit des Aufrufes gebildete Name der Arbeitsdatei wird nach Beendigung des Kommandos mit einer entsprechen-

den Meldung ausgegeben.

- Unabhängig von Angaben im Kommando wird der Parameter MF immer auf den Wert CRY gesetzt.

- Apostroph im String wird auf Leerzeichen abgebildet.

Erläuterungen der auftretenden Meldungen erhält man an der MVS-Anlage über DOC,SYSTEM,RST.

Beispiele:

```
RST.              Status aller eigenen Jobs

RST,US=N.         Status aller CRAY-Jobs aus Niedersachsen

RST,DC=EX,MF=CRY. Status aller CRAY-Jobs in der Ausführungs-Warteschlange
```

5. Weitere Kommandos für Bibliotheken und Dateien

COS enthält zwei Systemprogramme, die zur Bibliothekenverwaltung dienen. Das eine ist UPDATE, welches zum Verwalten von Quellprogrammen und Texten dient; das andere ist BUILD, welches zum Verwalten von übersetzten Programmen, also Objektmodulen, dient.

5.1 Bearbeiten von Objektbibliotheken (BUILD).

Mit BUILD können Objektbibliotheken erstellt, verändert und erweitert werden. Der Unterschied zwischen sequentiell auf einer Datei stehenden Moduln und einer Bibliothek ist der, daß am Ende einer Bibliothek ein Inhaltsverzeichnis (Directory) steht. Die Bedeutung der Bibliotheken liegt nun darin, daß der Lader im Stande ist, nur die Moduln herauszusuchen, die benötigt werden. Bei einer sequentiellen Datei werden alle Moduln geladen. Der Lader durchsucht dabei das Verzeichnis der Bibliothek und stellt so sehr schnell fest, ob ein Modul vorhanden ist und an welcher Stelle es in der Datei steht. Das Verwaltungsprogramm wird durch die Anweisung BUILD aufgerufen. BUILD-Anweisungen stehen separat auf einem Dateiabschnitt. Mit diesen Anweisungen können noch zusätzliche Moduln oder Bibliotheken angegeben werden; eine neue Bibliothek ($NBL) kann kopiert, weggelassen oder aufgelistet werden. In den meisten Fällen werden jedoch keine Anweisungen und nur wenige Parameter benötigt.

BUILD<,I=idn,L=ldn,OBL=obn,NBL=nbl,SORT,NODIR,REPLACE>.

Parameter: (die Voreinstellung (default) ist unterstrichen)

I=idn =$IN	Input: Name der Datei, die die BUILD-Anweisungen enthält. Wenn I=0 ist, werden keine Anweisungen erwartet.
L=ldn =$OUT	List: Name der Datei, auf die die Liste geschrieben wird. Bei L=0 wird keine Liste erzeugt.
OBL=odn =$OBL	Old Binary Library: Name der ersten Eingabe-Datei; normalerweise die alte Bibliothek, die verändert werden soll. OBL=0 bedeutet, daß eine neue Bibliothek angelegt werden soll.
B=bdn =$BLD	Binary: Name der zweiten Eingabe-Datei, deren Moduln addiert werden oder andere Moduln aus der ersten Eingabe-Datei ersetzen.
NBL=nbl =$NBL	New Binary Library: Name der Ausgabe-Datei; normalerweise ist das eine neue Bibliothek. Wenn zusätzlich der Parameter NODIR angegeben wird, hat die Ausgabe-Datei kein Bibliotheksformat. NBL=0 bedeutet, daß keine Ausgabe-Datei angelegt wird.
SORT	Die Moduln werden alphabetisch aufgelistet. SORT hat nur auf die Liste Einfluß, nicht auf die neue Bibliothek selbst.
NODIR	No Directory: zu der Ausgabe-Datei wird kein Verzeichnis erzeugt. Das Ergebnis ist eine sequentielle Datei wie $BLD.
REPLACE	Die Ausgabe-Bibliothek enthält die Moduln in der gleichen Reihenfolge wie die Eingabe-Bibliothek.

5.1.1 Anlegen einer Binär-Bibliothek

Im folgenden Beispiel wird eine neue Bibliothek angelegt. Die Eingabe-Datei ist die Datei ($BLD), die den Objektcode enthält, der von CAL, CFT oder einem anderen Übersetzer erzeugt wurde.

Beispiel:

```
JOB.
ACCOUNT,AC=xxxx,APW=UNKNACKBAR.
CFT,ON=A.
BUILD,I=0,OBL=0.
SAVE,DN=$NBL,PDN=ZUGLIB.
/EOF
      SUBROUTINE LOK
      ...
      SUBROUTINE PACKW
      ...
/EOF
```

Anlegen einer Objektbibliothek: CFT schreibt die Moduln auf die Datei $BLD. Diese Datei wird als Eingabe-Datei genommen und auf die Ausgabe-Datei, die neue Bibliothek ($NBL), kopiert. BUILD fügt noch das Verzeichnis ein. Die Bibliothek ist nun angelegt und kann vom Lader verwendet werden.

Beispiel:

```
JOB.
ACCOUNT,AC=xxxx,APW=UNKNACKBAR.
ACCESS,DN=ZUGLIB.
CFT,ON=A.
LDR,LIB=ZUGLIB.     besser: SEGLDR,GO,CMD='LIB=ZUGLIB'.
/EOF
         PROGRAM BAHN
         ...
         CALL LOK
         ...
         END
/EOF
```

In dem FORTRAN-Programm wird das Unterprogramm LOK aufgerufen. Der Lader lädt nur der Modul LOK aus der Bibliothek.

5.1.2 Verändern einer Binär-Bibliothek

Die Anweisung BUILD erkennt zwei Eingabe-Dateien. Die erste ist die alte Objektbliothek, die mit dem Parameter OBL auf der Steuerkarte angegeben wird. Die zweite wird mit dem Parameter B angegeben und enthält Objektmoduln. Wenn beide Dateien angegeben werden bzw. die Voreinstellung verwendet wird, werden beide Dateien auf die neue Bibliothek kopiert, sofern durch eine Anweisung nichts ausgeschlossen wird. Kommen einzelne Moduln in beiden Dateien vor, so werden die Moduln der zweiten Datei (B-Parameter) auf die neue Bibliothek (NBL) kopiert. Wenn ein Modul aus der zweiten Datei ein Modul aus der alten Bibliothek an der gleichen Stelle ersetzen soll, ist der Parameter REPLACE zu verwenden.

Beispiel:

```
JOB.
ACCOUNT,AC=xxxx,APW=UNKNACKBAR.
ACCESS,DN=$OBL,PDN=ZUGLIB,UQ.
CFT,ON=A.
BUILD,I=0,REPLACE.
SAVE,DN=$NBL,PDN=ZUGLIB.
DELETE,DN=$OBL.
/EOF
   SUBROUTINE LOK
   ...
   SUBROUTINE KLASSE1
   ...
/EOF
```

Erweitern einer Bibliothek: Der Modul KLASSE1 wird zu den Moduln aus der alten Bibliothek addiert. Die alte Bibliothek enthält schon ein Modul mit dem Namen LOK, dies wird durch den Modul LOK auf der zweiten Eingabe-Datei ausgetauscht.

5.1.3 BUILD-Anweisungen

BUILD kann auch mit Anweisungen gesteuert werden, die auf einer mit dem I-Parameter angegebenen Datei liegen. Die Voreinstellung ist $IN. Eine Anweisung besteht aus einem Schlüsselwort und eventuell einer Liste von Dateinamen. Die Liste muß vom Schlüsselwort mindestens durch ein Leerzeichen getrennt werden. Auf einer Zeile können mehrere durch Punkt oder Strichpunkt zu trennende Direktiven angegeben werden; jedoch darf eine Anweisung nicht über das Zeilenende hinausgehen.

Die FROM-Anweisung

Die FROM-Anweisung gibt eine Liste von Eingabe-Dateien an. Es gibt zwei Möglichkeiten für diese Liste. Die eine Möglichkeit ist, daß nur eine Datei angegeben wird. Auf diese Datei beziehen sich dann nachfolgende Anweisungen wie COPY, OMIT oder LIST. In Abhängigkeit von diesen Anweisungen werden Moduln aus der angegebenen Datei mit auf die neue Bibliothek $NBL geschrieben. Die andere Möglichkeit ist, daß mehrere Dateinamen angegeben werden. Dabei werden alle Dateien bis auf die zuletzt angegebene auf die Ausgabe-Datei $NBL kopiert. Die zuletzt angegebene Datei wird wie bei der ersten Möglichkeit behandelt.

FROM dn_1 <,dn_2 ,dn_3 ,...,dn_n ;OMIT fn_1 ;COPY fn_2 >

Die Eingabe-Dateien (dn_i) können entweder Bibliotheken (mit Verzeichnis) oder auch sequentielle Dateien (wie $BLD) sein. BUILD überprüft, ob ein Verzeichnis vorhanden ist und benutzt dieses dann auch.

Beispiel:

```
JOB.
ACCOUNT,AC=xxxx,APW=UNKNACKBAR.
ACCESS,DN=LIBEINS.
ACCESS,DN=LIBZWEI.
ACCESS,DN=LIBDREI.
ACCESS,DN=ZUGLIB.
BUILD,I,OBL=0,B=0.
SAVE,DN=$NBL,PDN=BIGLIB.
/EOF
FROM LIBEINS,LIBZWEI,LIBDREI,ZUGLIB;OMIT LOK
/EOF
```

Zusammenfügen von Bibliotheken: Alle vier Bibliotheken werden zu einer Bibliothek zusammengefügt. Bei der letzten wird der Modul LOK nicht kopiert.

Die OMIT-Anweisung

Mit der OMIT-Anweisung können Moduln vom Kopieren ausgeschlossen werden. Eine OMIT-Anweisung bezieht sich nur auf die augenblickliche Eingabe-Datei.

OMIT fn_1 ,fn_2 ,fn_3 ,...,fn_n

Parameter:

fn_i

* Ein einzelner Name wie z. B. LOK oder PACKW, dabei werden die Moduln explizit vom Kopieren ausgeschlossen.

* Eine Gruppe von Namen. Sie kann gebildet werden, indem für Zeichenkombinationen ein Ersatzzeichen (- oder *) gesetzt wird. Anstelle eines "-" können mehrere Kombinationen von Zeichen stehen. Der "*" ersetzt genau ein, jedoch jedes beliebige Zeichen, außer dem Leerzeichen . Z.B. HA- schließt HASE, HAB, HATTE, HA selbst usw. ein; *IEB** ersetzt SIEBEN, LIEBEN, GIEBEL, aber nicht HIEB. Eine COPY-Anweisung kann explizit wieder einzelne Moduln einer Gruppe zum Kopieren einbeziehen.

Beispiel:

```
JOB.
ACCOUNT,AC=xxxx,APW=UNKNACKBAR.
ACCESS,DN=$OBL,PDN=ZUGLIB.
BUILD,B=0.
SAVE,DN=$NBL,PDN=ICLIB.
/EOF
OMIT KLASSE2
/EOF
```

Löschen eines Moduls in einer Bibliothek: Die neue Bibliothek ICLIB enthält nicht mehr der Modul KLASSE2.

Die COPY-Anweisung

Mit der COPY-Anweisung werden nur die auf der Anweisung stehenden Moduln von der augenblicklichen Eingabe-Datei auf die Ausgabe-Datei $NBL kopiert. Dabei kann der Benutzer einzelne Moduln, Gruppen oder Bereiche von Moduln angegeben.

COPY fn_1,fn_2,fn_3,...,fn_n

Parameter:

fn_i

* Ein einzelner Modulname; dabei werden die Moduln explizit kopiert, auch wenn der Modulname zu einer von OMIT ausgeschlossenen Gruppe gehört.

* Eine Gruppe von Moduln; dabei werden alle Moduln dieser Gruppe kopiert, es sei denn, sie sind explizit in einer vorhergehenden OMIT-Anweisung ausgeschlossen. (Erläuterung der Gruppen siehe bei der OMIT-Anweisung.)

Die folgenden beiden Spezialfälle sind auch möglich:

old=new — Mit der Form OLDNAME=NEWNAME kann ein Modul umbenannt werden. Der Name wird im Ausgabe-Verzeichnis und im ProgrammBeschreibungs-Verzeichnis des Moduls geändert.

first,last — Alle Moduln, die in diesem Bereich sind, werden kopiert, es sei denn, sie sind durch ein vorhergehendes OMIT explizit ausgeschlossen.

Beispiele für Bereiche:

(VONA,BISZ) — kopiert alle Moduln von VONA bis BISZ.

(ABHIER,) — kopiert alle Moduln von ABHIER bis zum Ende der Eingabe-Datei.

(,BISHIER) — kopiert alle Moduln von der augenblicklichen Position bis BISHIER.

(,) — kopiert alle Moduln von der augenblicklichen Position bis zum Ende der Eingabe-Datei.

Beispiel:

```
JOB.
ACCOUNT,AC=xxxx,APW=UNKNACKBAR.
ACCESS,DN=$OBL,PDN=ZUGLIB.
BUILD,B=O,N=$BLD,NODIR.
...
LDR.                    (besser: SEGLDR,GO.)
/EOF
COPY -W
/EOF
```

Herausnehmen von Moduln aus einer Bibliothek: Die Ausgabe-Datei enthält alle Moduln der Bibliothek ZUGLIB, die mit W enden, in sequentiellen Format. Ein Verzeichnis wird nicht angehängt.

Die LIST-Anweisung

LIST

Durch die LIST-Anweisung wird ein Inhaltsverzeichnis der augenblicklichen Eingabe-Datei auf die Ausgabe-Datei ldn für die Liste geschrieben. (Die von BUILD standardmäßig erzeugte Listdatei enthält ein Inhaltsverzeichnis der neuen Bibliothek $NBL.)

5.2 Bearbeiten von Quellenbibliotheken (UPDATE)

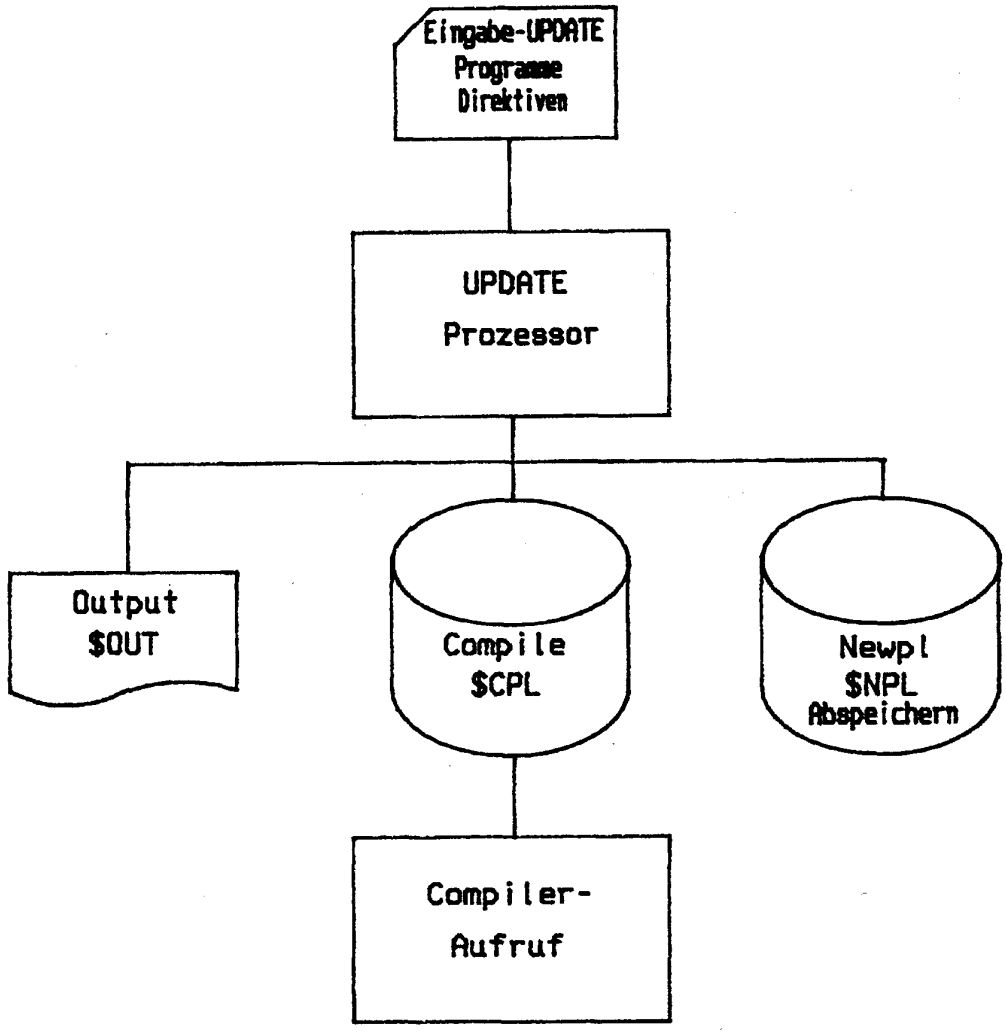
Das Systemprogramm UPDATE verwaltet Programme in ihrer Quellform, d.h. in lesbarer Form, wie man sie z. B. im Editor erstellt. Die Quellprogramme werden in Dateien (Datasets), den Quellprogrammbibliotheken, verwaltet. UPDATE kann solche Programmbibliotheken erzeugen und verändern, d. h. einzelne Quellzeilen entfernen, einfügen oder durch andere ersetzen.
Der Benutzer hat nicht mehr riesige Editor-Dateien zu bearbeiten und verliert auch nicht mehr die Übersicht über seine Programmänderungen; denn UPDATE verwaltet alle Programme in einer einzigen Datei in komprimierter Form und führt stets eine Liste über alle durchgeführten Anderungen mit.

Alle Quellzeilen werden zu Einheiten (Decks) zusammengefaßt, bekommen ein Gruppenkennzeichen (Deckname) und werden innerhalb einer Gruppe automatisch durchnumeriert, so daß sie entweder einzeln oder in Gruppen entfernt oder ersetzt werden können. Jede Korrektur wird unter einem eindeutigen Namen (IDENT) durchgeführt, so daß sie jederzeit wieder identifiziert und angesprochen werden kann. UPDATE ist unabhängig von Programmiersprachen und kann auch dazu benutzt werden, beliebige Datenbestände zu verwalten. Nachfolgend werden die wichtigsten Möglichkeiten beschrieben. In speziellen Fällen ziehe man das CRAY-Update Reference Manual SR-0013 (==> 1.7.1) zu Rate.

5.2.1 Die Arbeitsweise von UPDATE

Anlegen einer Quellbibliothek (Erstellungs-Lauf)

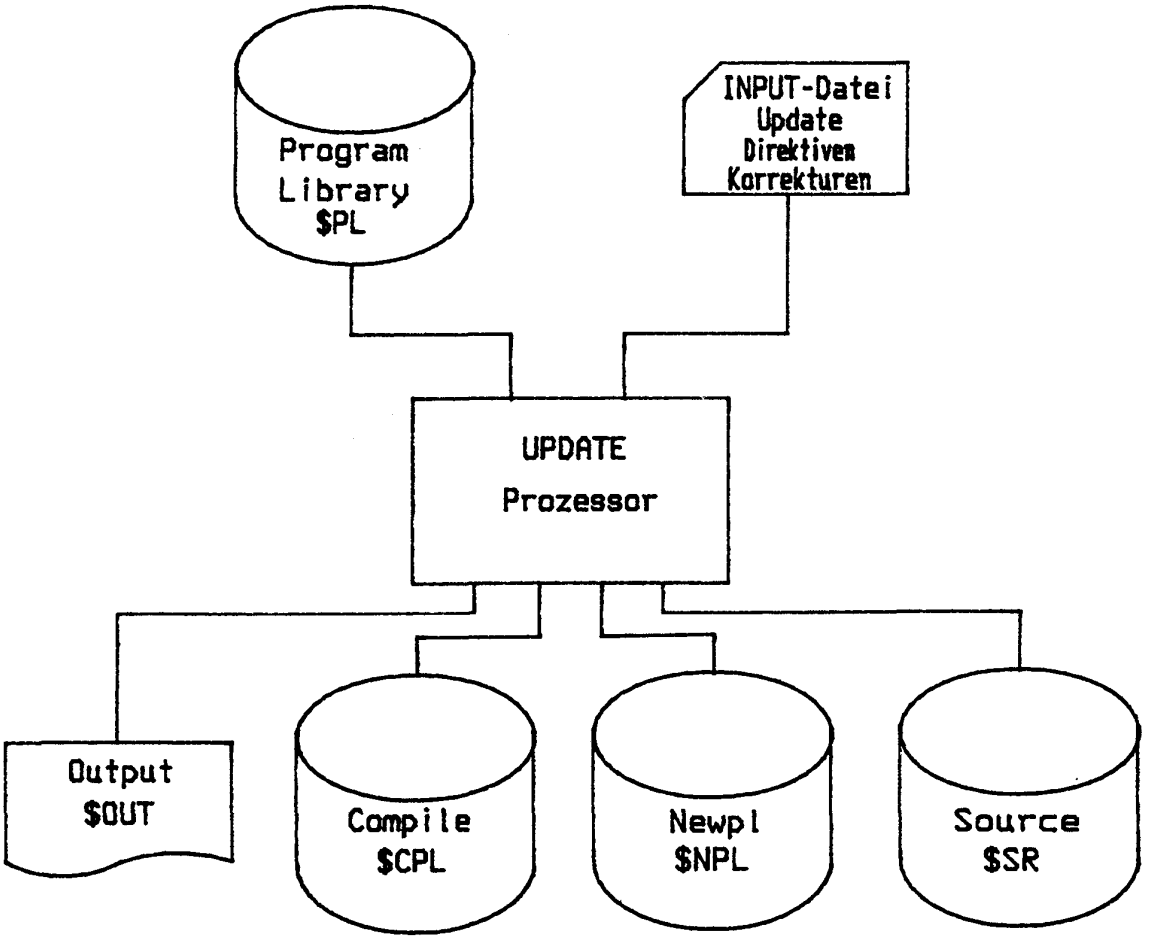
UPDATE liest die in Decks unterteilten Programme von der Eingabe-Datei, normalerweise $IN, und erzeugt daraus eine Programmbibliothek, normalerweise $PL. Der Ablauf wird auf der Ausgabe-Datei, normalerweise $OUT, protokolliert. Standardmäßig werden die Quellzeilen zusätzlich auf die COMPILE-Datei geschrieben, die, falls es sich um Programme handelt, als Eingabe-Datei für den Übersetzer dient.



UPDATE-Erstellungs-Lauf: Erzeugen einer neuen Programmbibliothek

Verändern einer Quell-Bibliothek (UPDATE-Lauf)

UPDATE liest seine Anweisungen von der Eingabe-Datei, normalerweise $IN, und die zu bearbeitenden Programme von der Programmbibliothek, normalerweise $PL, und erzeugt daraus eine neue Programmbibliothek, normalerweise $NPL. Der Ablauf wird auf die Ausgabe-Datei protokolliert, normalerweise $OUT. Falls gewünscht, kann eine Quell-Datei erstellt werden, die alle aktiven Quellzeilen der Bibliothek mit UPDATE-Anweisungen enthält, so daß daraus z.B. eine neue Bibliothek (Erstellungs-Lauf) erzeugt werden kann.



UPDATE-Correction-Run: Modifizieren einer Programmbibliothek

5.2.2 Der UPDATE-Aufruf

Steuerung über den Aufruf von Update:

UPDATE<,P=pln,I=idn_1:idn_2:...:id_n,C=cdn,N=nln^
L=ldn,E=edn,S=sdn,*=m,/=c,DW=dw,DC=dc,ML=n,^
(F IQ=dk_1:dk_2:...:dk_n I^
Q=dk_1,dk_2,...dk_j.dk_k,...),options>.

Parameter: (die Voreinstellung (default) ist unterstrichen)

P=pln — Program Library Name: Gibt den lokalen Namen der Datei an, der die Programmbibliothek enthält.

=$PL — Wird P=0 angegeben, handelt es sich um einen Erstellungs-Lauf, d. h. es gibt keine alte Programmbibliothek.

I=idn_1:idn_2:...:idn_n — Input Dataset Names: Diese Dateien enthalten die Direktiven und Quellzeilen für den UPDATE-Lauf und werden in der angegebenen Reihenfolge gelesen. Es können maximal 100 Eingabe-Dateien angegeben werden.

=$IN — Wird I=0 angegeben, so werden keine Anweisungen oder Quellzeilen von der Eingabe-Datei gelesen. (I=0 ist beim Erstellungs-Lauf nicht erlaubt.)

C=cdn — Compile Dataset Name: Gibt den Namen der COMPILE-Datei an, auf die die zu übersetzenden Programme bzw. die zu verarbeitenden Daten geschrieben werden sollen.

=$CPL — Wird C=0 angegeben, so wird keine COMPILE-Datei erzeugt.

N=nln — New Library Name: Gibt den Namen der Datei an, auf die die neue Programmbibliothek geschrieben werden soll. Voreinstellungen:
1. Anderungs-Lauf:
wird nur N angegeben, so wird die neue Programmbibliothek auf die Datei $NPL geschrieben;
wird N=0 angegeben oder fehlt der Parameter, so wird keine neue Programmbibliothek erzeugt.
2. Erstellungs-Lauf:
fehlt der Parameter oder wird nur N angegeben, so wird die neue Programmbibliothek auf die Datei $NPL geschrieben;
wird N=0 angegeben, so wird keine neue Programmbibliothek erzeugt.

L=ldn
=$OUT — Listing Dataset Name: Gibt den Namen der Ausgabe-Datei an, auf die UPDATE sein Ablaufprotokoll schreiben soll. Wird L=0 angegeben, so wird kein Ablaufprotokoll erzeugt.

E=edn
=$OUT — Error Listing Dataset Name: Gibt den Namen der Datei an, auf die nur die Fehlermeldungen von UPDATE geschrieben werden sollen. Wird E=0 angegeben, so werden die Fehlermeldungen auf die Ausgabe-Datei geschrieben.

Anmerkung: Wird bei E und L die gleiche Datei angegeben, so gilt L=dn und E=0.

S=sdn — Source Dataset Name: Es wird eine Datei erzeugt, die bei einem nachfolgenden Anderungs-Lauf als Eingabe benutzt werden kann, um eine bereinigte Bibliothek zu erstellen. Sie enthält die ursprünglichen Quellzeilen ohne UPDATE-Numerierung sowie die notwendigen UPDATE-Anweisungen zur Erzeugung einer Bibliothek.
Fehlt der Parameter oder wird S=0 angegeben, so wird keine

SOURCE-Datei erzeugt. Wird nur S angegeben, so gilt S=$SR.

*=m — Master Zeichen: Definiert das UPDATE-Steuerzeichen. Fehlt der Parameter beim Erstellungs-Lauf, so gilt *=*. Fehlt der Parameter beim Anderungs-Lauf, so wird das in der Programmbibliothek definierte Zeichen als Steuerzeichen verwendet. Wird *=m bei einem Anderungs-Lauf angegeben, so muß es mit dem in der Bibliothek verwendeten Steuerzeichen übereinstimmen. Es braucht aber in diesem Falle nicht angegeben zu werden. "*" allein ist nicht zulässig.

/=c
=/ — Comment Zeichen: Definiert das Zeichen für Kommentare in UPDATE-Anweisungen. Fehlt der Parameter, so gilt /=/. "/" allein ist nicht zulässig.

DW=dw
=72 — Data Width Value: Definiert die Länge der Quellzeilen in der COMPILE- und SOURCE-Datei. Die Information steht innerhalb der Spalten 1 bis dw. Die Spalten dw+1 bis dw+15 der COMPILE-Datei enthalten Leerzeichen sowie die UPDATE-Numerierung. Die SOURCE-Datei enthält nur die Dateninformation von Spalte 1 bis dw, es sei denn, die SQ-Option wurde gesetzt (==> SQ-Option).
Anmerkung: Fehlt der Parameter oder wird nur DW bzw. DW=dw angegeben, so enthalten die Spalten dw+1 bis dw+8 die Kennzeichnung rechtsbündig mit führenden Leerzeichen, dw+9 enthält einen Punkt, und dw+10 bis dw+15 enthalten die Numerierung, linksbündig mit nachfolgenden Leerzeichen. Wenn dw in der Form Ldw angegeben wird, so wird die gesamte UPDATE-Numerierung linksbündig ausgegeben.

DC=ON/OFF
=OFF — Declared Modifications Option: Dieser Parameter stellt sicher, daß jede Modifikation nur die gewünschte Stelle verändert und nicht aus Versehen andere Teile der Bibliothek verändert werden, d.h. jede Einheit, die von einer Modifikation betroffen ist, muß zuerst deklariert werden. Fehlt der Parameter oder wird DC=OFF angegeben, so ist die Sicherheit ausgeschaltet, d. h. die betreffende Einheit muß nicht für die Modifikation deklariert werden.

ML=n — Message Level: Ebene, ab der die Fehlermeldungen in der Ausgabe unterdrückt werden sollen. Protokoll-Meldungen werden davon nicht berührt.

=1	COMMENT	zur Zeit nicht benutzt.
=2	NOTE	Informationen, die sich nicht auf Fehler beziehen.
=3	CAUTION	Bemerkungen zu möglichen Fehlern.
=4	WARNING	Warnung bei wahrscheinlichen Fehlern.
=5	ERROR	Fataler Fehler.

F — Full-Erstellungs-Lauf: die gesamte Bibliothek soll bearbeitet werden.

Q<=$dk_1:dk_2:...:dk_n$>
='$dk_1,dk_2,...,dk_j\,dk_k,...,dk_n$' Quick-Erstellungs-Lauf: nur die Bibliothekseinheiten, die durch den Q-Parameter auf der Steuerkarte oder durch die COMPILE-Anweisung angegeben wurden, werden bearbeitet.

Anmerkung: Ist weder F noch Q angegeben, werden nur solche Bibliothekseinheiten bearbeitet, die entweder modifiziert wurden, durch COMPILE-Anweisungen spezifiziert wurden oder die andere modifizierte Einheiten aufrufen.

options:

NA	No Abort: bei Fehlern im UPDATE-Lauf soll nicht abgebrochen werden. Alle verlangten Dateien werden erstellt.
NR	No Rewind: die SOURCE- und COMPILE-Datei sollen vor und nach dem UPDATE-Lauf nicht an den Anfang positioniert werden.
IF	If: die Zusammenfassung aller bedingten Texte (Conditional Text) soll auf die Ausgabe-Datei geschrieben werden.
IN	Input: die Eingabe für UPDATE soll auf die Ausgabe-Datei geschrieben werden.
ID	Identifier: die Zusammenfassung aller UPDATE-Kennzeichen soll auf die Ausgabe-Datei geschrieben werden.
ED	Edited: die Zusammenfassung aller editierten Quellzeilen soll auf die Ausgabe-Datei geschrieben werden.
CD	Compile Directives: alle Anweisungen, die die COMPILE-Datei betreffen, sollen auf die Ausgabe-Datei geschrieben werden.
UM	Unprocessed Modifications: alle nicht bearbeiteten UPDATE-Modifikationen sollen auf die Ausgabe- und/oder Fehler-Datei geschrieben werden. Im Full- oder Normal-Modus wird diese Option ignoriert.
K	Die Bibliothekseinheiten werden in der Reihenfolge auf die COMPILE-Datei und die neue Bibliothek geschrieben, in der sie beim Q-Parameter oder der COMPILE-Anweisung angegeben wurden. Diese Option wird im Full- oder Normal-Modus ignoriert.
SQ	Sequencing: die Ausgabe auf der SOURCE-Datei hat ab Spalte DW+1 eine Numerierung. SQ hat keine Auswirkung auf die COMPILE-Datei. Anmerkung: Wenn eine Modifikation zwei oder mehr Bibliothekseinheiten betrifft und die K-Option eingeschaltet ist, kann die Numerierung der eingefügten Zeilen inkonsistent mit der Numerierung ohne K-Option sein.

5.2.3 Die UPDATE Anweisungen

UPDATE-Anweisungen sind Steueranweisungen an das Programm UPDATE, die die Bearbeitung einer Bibliothek beeinflussen. Sie stehen auf der Eingabe-Datei und werden durch das Master-Zeichen, das UPDATE-Steuerzeichen, in Spalte 1 gekennzeichnet. Die Anweisung selbst folgt unmittelbar auf das Steuerzeichen, also ab Spalte 2 und kann entweder ausgeschrieben oder auch abgekürzt angegeben werden. Die Parameter einer Anweisung werden entweder durch Komma oder ein Leerzeichen von der Anweisung selbst und untereinander durch Komma getrennt. Ein eventueller Kommentar auf der Anweisungs-Zeile muß durch ein oder mehrere Leerzeichen von den Parametern getrennt sein.

Die Syntax der UPDATE-Anweisung:

```
+-+---------+-+----+-+----+-+----+-+     +-+-----------+
!*!Anweisung! ! P1 !,! P2 !,! P3 !,! ....! !<Kommentar>!
+-+---------+-+----+-+----+-+----+-+     +-+-----------+
```

Die Kennzeichnung einer Quellzeile:

Jede Modifikation und jedes Deck bzw. Common-Deck besitzt eine eindeutige Kennzeichnung. Dieses Kennzeichen ist der Name aus der entsprechenden DECK-, COMDECK- oder IDENT-Anweisung. Die Numerierung einer Zeile ergibt sich aus ihrer Position innerhalb des Decks bzw. Common-Decks. Die Numerierung einer eingefügten oder ersetzten Zeile ergibt sich aus ihrer Position innerhalb der Modifikation (MODIFICATION-SET). Eine Zeile wird angesprochen durch

Kennung.Nummer

Zulässige Kennungen:

Jede Kennung eines Decks, Common-Decks oder einer Modifikation ist ein ein bis acht Zeichen langer Name, der beim erstmaligen Auftreten zugewiesen wird. Solche Namen dürfen keine Kommata, Punkte, Leerzeichen, Doppelpunkte oder Gleichheitszeichen enthalten. Jedes andere Zeichen aus dem ASCII-Zeichensatz im Bereich von 41 (oktal) bis 176 (oktal) ist zulässig (==> Anhang C).

Bei manchen Anweisungen können einzelne Namen oder aber Bereiche angegeben werden. Wird als Parameter ein Bereich angegeben, so besteht er aus dem Namen der ersten Kennung, einem Punkt und dem Namen der letzten Kennung:

erstername.letztername

Die Anweisungen (in alphabetischer Reihenfolge):

Die BEFORE-Anweisung

*B<EFORE> id.seq

Parameter:

id	Name des Decks oder der Modifikation
seq	Zeilennumerierung

Die BEFORE-Anweisung gibt an, daß die nachfolgenden Quellzeilen vor die angegebene Zeile id.seq eingefügt werden.

Die CALL-Anweisung

*CAll comdeck

Parameter:

comdeck	Name des Common-Decks

Die CALL-Anweisung veranlaßt, daß das Common Deck comdeck an dieser Stelle in den Quelltext auf der COMPILE-Datei eingefügt wird. Dies erlaubt dem Benutzer, eine einzige Version eines Common Decks in der Bibliothek zu halten und dieses an verschiedenen Stellen in andere Decks einzufügen, so daß allgemeine Programmteile nur einmal zentral bearbeitet werden müssen. Ein Common-Deck kann CALL-Anweisung auf andere Common-Decks enthalten, jedoch nicht einen CALL auf sich selbst.

Die COMDECK-Anweisung

*C<OM>D<EC>K cmdk,NOPROP

Parameter:

cmdk	Name des Common-Decks
NOPROP	NO PROPagation: dieser Parameter bedeutet, wenn gesetzt, daß Decks, die ein solches Common-Deck rufen, nicht automatisch als modifiziert betrachtet werden sollen, wenn das Common-Deck modifiziert wurde. Wurde dieser Parameter nicht gesetzt, so werden alle Decks, die ein modifiziertes Common-Deck rufen, ebenfalls als modifiziert betrachtet.

Die COMDECK-Anweisung leitet ein Common-Deck ein. Alle nachfolgenden Quellzeilen bis zur nächsten DECK-, COMDECK-, IDENT-, INSERT-, DELETE-, BEFORE-Direktive oder bis zum Dateiende werden diesem Common-Deck zugeordnet. Alle anderen Anweisungen werden interpretiert, aber beenden nicht das Common-Deck.

Die COMDECK-Anweisung ist die erste Zeile des Common-Decks und erhält die Zeilennummer 1.

Die COMPILE-Anweisung

*C<OMPILE> $name_1$,$name_2$, ...,$name_j$.$name_k$, ... $name_n$

Parameter:

$name_1$	Name eines Decks oder Common-Decks.
$name_j$.$name_k$	Bereich von Decks und/oder Common-Decks.

Die Reihenfolge der Parameter ist nur von Bedeutung, wenn die K-Option auf der Steuerkarte angegeben wurde. In diesem Falle werden die Decks in der bei *COMPILE angegebenen Reihenfolge auf die COMPILE-Datei und/oder neue Bibliothek geschrieben. Im anderen Falle werden sie in der Reihenfolge übernommen, in der sie auf der alten Bibliothek standen.

Die COPY-Anweisung

*COPY name,id_1 .seq_1 ,id_2 .seq_2 (Format 1)
*COPY name,id_1 .seq_1 ,id_2 .seq_2 ,dn<,SEQ> (Format 2)

Parameter:

name	Name des Decks oder Common-Decks, aus dem Zeilen kopiert werden sollen.
id_1 .seq_1	Anfang des zu kopierenden Bereichs.
id_2 .seq_2	Ende des zu kopierenden Bereichs.
dn	Name der Datei, auf die die Zeilen kopiert werden sollen. Diese Datei darf nicht gleichzeitig in anderer Weise auf der UPDATE-Steueranweisung verwendet werden.
SEQ	muß angegeben werden, wenn die Zeilennumerierung mit übernommen werden soll.

Die COPY-Anweisung hat zwei unterschiedliche Formate mit verschiedenen Funktionen:
Format 1 kopiert Zeilen von der alten Bibliothek in die neue Bibliothek so, als ob sie als Eingabe stehen würden. In diesem Fall muß COPY zusammen mit INSERT, BEFORE, RESTORE, DELETE oder DECK bzw. COMDECK benutzt werden. Die Zeilen werden kopiert, bevor irgendwelche Modifikationen daran gemacht werden, deshalb können Zeilen von einer Stelle zu einer anderen kopiert werden und gleichzeitig an der ursprünglichen Stelle geloscht werden.
Format 2 kopiert Zeilen von der alten Bibliothek auf eine in der Anweisung angegebene Datei. Die Zeilennumerierung wird mit übernommen, wenn SEQ angegeben wurde. Die Zeilenlänge entspricht der der SOURCE-Datei.

Die CWEOF-Anweisung

*CWEOF

Die CWEOF-Anweisung schreibt eine EOF-Marke auf die COMPILE-Datei, sofern die Datei nicht hinter einer EOF-Marke positioniert ist oder wenn sie nicht am Anfang positioniert ist. Die CWEOF-Anweisung wird in ein Deck oder Common-Deck eingebaut und erhält eine Numerierung. Die Anweisung wird ignoriert, wenn keine COMPILE-Datei verlangt wurde.

Die DECK-Anweisung

*D<EC>K deckname

Parameter:

deckname Name des neu zu erstellenden Decks.

Die DECK-Anweisung leitet ein neues Deck ein. Alle Quellzeilen bis zur nächsten DECK-, COMDECK-, IDENT-, INSERT-, DELETE-, BEFORE-Anweisung oder bis zum Ende der Eingabe-Datei werden diesem Deck zugerechnet. Das neue Deck wird in der neuen Programmbibliothek an das Ende der vorhandenen Decks und Common-Decks angehängt. Decks können andere Anweisungen wie CALL, CWEOF oder WEOF enthalten. Andere Anweisungen werden interpretiert, aber schliessen das Deck nicht ab. Die DECK-Anweisung ist die erste Zeile des Decks und erhält die Nummer 1.

Die DECLARE-Anweisung

*D<E>C<LARE> name

Parameter:

name Name eines Decks oder Common-Decks.

Die DECLARE-Anweisung stellt sicher, daß nachfolgende Modifikationen nur das angegebene Deck betreffen.

Die DEFINE-Anweisung

*DEFINE $defname_1$ <,$defname_2$,...,$defname_n$ >

Parameter:

defname Definierter Name.

Die DEFINE-Anweisung definiert einen Namen, der in einer IF-Anweisung abgefragt werden kann. Ein solcher Name muß sich nicht unbedingt von einem DECK-, COMDECK- oder Modifikations-Namen unterscheiden. Definierte Namen sind nur in dem UPDATE-Lauf bekannt, in dem sie definiert wurden; sie werden nicht in der Bibliothek gespeichert.

Die DELETE-Anweisung

```
*D<ELETE> id1.seq1,id2.seq2   (Bereich)
*D<ELETE> id1.seq1,seq2       (Bereich,Kurzform)
*D<ELETE> id1.seq1            (einzelne Zeile)
```

Parameter:

id_1	Name eines Decks oder einer Modifikation.
seq_1,seq_2	Zeilennummern.

Die DELETE-Anweisung erlaubt es, Zeilen oder Zeilenbereiche zu löschen (deaktivieren) und gegebenenfalls durch andere Zeilen, die der DELETE-Anweisung folgen, zu ersetzen.
Eine gelöschte Zeile wird mit auf die neue Bibliothek übernommen. Sie behält ihre Kennung, wird jedoch als inaktiv gekennzeichnet. Inaktive Zeilen erscheinen nicht auf der COMPILE- und SOURCE-Datei. Ein Bereich von zu löschenden Zeilen darf nicht das Ende eines Decks überschreiten.

Die EDIT-Anweisung

```
*ED<IT> name1,name2,...,namej.namek,...,namen
```

Parameter:

$name_1$	Name eines Decks oder Common-Decks.
$name_j$.$name_k$	Bereich von Decks und/oder Common-Decks.

Die EDIT-Anweisung entfernt alle inaktiven Zeilen von Decks und/oder Common-Decks. Es wird nicht neu durchnumeriert. Es werden nur die Decks editiert, die explizit bei der EDIT-Anweisung angegeben wurden. Mit EDIT entfernte Zeilen können nicht mehr zurückgeholt werden.

Die ELSE-Anweisung

```
*ELSE
```

Die ELSE-Anweisung prüft mit der alternativen Bedingung einer vorausgehenden IF- oder ELSEIF-Anweisung, um zu entscheiden, ob die nachfolgenden Zeilen auf die COMPILE-Datei zu schreiben sind oder nicht. Innerhalb eines IF-Blocks kann nur ein ELSE stehen, dieses ELSE muß nach allen eventuell vorhandenen ELSEIFs dieses Blocks kommen.

Die ELSEIF-Anweisung

```
*ELSEIF typ,name<,...,boolop,typ,name>
```

Parameter:

typ	Typ des Bedingungsnamens, entweder DECK, IDENT oder DEF. Bei

DECK muß name der Name eines Decks oder Common-Decks sein, bei IDENT muß name der Name einer Modifikation sein. Wird DEF angegeben, so muß name ein mit DEFINE definierter Name sein. Ein Minuszeichen vor typ negiert die Bedingung.

name — Ein DECK- oder COMDECK-Modifikationsname oder ein mit DEFINE erklärter Name, abhängig von typ. Jede Klausel der Bedingung ist wahr, wenn der Name des richtigen Typs bekannt ist, bzw. bei Negation, wenn der Name unbekannt ist.

boolop — Logischer Operator: AND, OR oder XOR.

Die ELSEIF-Anweisung spezifiziert eine Bedingung, die nur dann ausgewertet wird, wenn keine vorherige Bedingung im selben IF-Block erfüllt wurde. Ist die Bedingung erfüllt, so werden die nachfolgenden Zeilen auf die COMPILE-Datei geschrieben; alle anderen Zeilen, die weiteren ELSEIF- und ELSE-Anweisungen im selben IF-Block folgen, werden übersprungen. Ist die Bedingung nicht erfüllt, so werden alle Zeilen bis zum nächsten IF, ELSEIF, ELSE oder ENDIF ignoriert. ELSEIF muß einem entsprechenden IF zugeordnet sein und darf nicht auf ein ELSE folgen.

Die ENDIF-Anweisung

```
*ENDIF
```

Die ENDIF-Anweisung beendet einen IF-Block, der mit der IF-Anweisung eingeleitet wurde.

Die IDENT-Anweisung

```
*ID<ENT> ident,U=id1:id2:id3:...:idn,
              K=id4:id5:id6:...:idm,DC=name
```

Parameter:

ident — Name der Modifikation.

U=id — Unknown Modification IDs: die Modifikation wird ignoriert, wenn eine hier angegebene ID in der Bibliothek gefunden wird.

K=id — Known Modification IDs: die Modifikation wird ignoriert, wenn eine hier angegebene ID in der Bibliothek nicht gefunden wird.

Die Parameter U und K können auch in folgender Weise angegeben werden:

```
U=id1,U=id2,...,K=id6,...
```

DC=name — Deklariertes Deck oder Common-Deck, für das die nachfolgenden Modifikationen gelten sollen:
Wenn keine der Bedingungen erfüllt werden, überspringt UPDATE alle Zeilen und Anweisungen bis zur nächsten IDENT-Anweisung oder bis zum Dateiende und erzeugt eine diesbezügliche Protokoll-Meldung.

Die IF-Anweisung

```
*IF typ,name<,...,boolop,typ,name>
```

Parameter:

typ	Typ des Bedingungsnamens, entweder DECK, IDENT oder DEF. Bei DECK muß name der Name eines Decks oder Common-Decks sein, bei IDENT muß name der Name einer Modifikation sein. Wird DEF angegeben, so muß name ein mit DEFINE definierter Name sein. Ein Minus-zeichen vor typ negiert die Bedingung.
name	Der Name eines Decks, eines Common-Decks, ein Modifikationsname oder ein mit DEFINE erklärter Name, abhängig von typ. Jede Klausel der Bedingung ist wahr, wenn der Name des richtigen Typs bekannt ist, bzw. bei Negation, wenn der Name unbekannt ist.
boolop	Logischer Operator: AND, OR oder XOR.

Die IF-Anweisung leitet einen IF-Block ein und definiert die Bedingung, unter der die nachfolgenden Zeilen auf die COMPILE-Datei geschrieben werden. Ein IF-Block kann ELSEIF- und ELSE-Anweisungen enthalten und muß mit der ENDIF-Direktive abgeschlossen werden. IF-Blöcke können beliebig geschachtelt werden. Ist die IF-Bedingung nicht erfüllt, so werden alle Zeilen bis zur nächsten IF-, ELSEIF-, ELSE- oder ENDIF-Anweisung ignoriert und nicht auf die COMPILE-Datei geschrieben. Alle übersprungenen Zeilen werden jedoch auf die neue Bibliothek oder in die SOURCE-Datei übernommen.

Die INSERT-Anweisung

```
*I<NSERT> id.seq
```

Parameter:

id	Name eines Decks oder einer Modifikation.
seq	Zeilennumerierung.

Die INSERT-Anweisung veranlaßt, daß die nachfolgenden Zeilen nach der angegebenen Zeile eingefügt werden.

Die LIST-/NOLIST-Anweisung

```
*LIST
*NOLIST
```

Die LIST- bzw. NOLIST-Anweisungen starten bzw. unterdrücken das Auflisten der Zeilen aus der Eingabe-Datei. Diese Anweisungen können an beliebiger Stelle in der Eingabe-Datei erscheinen. Sie steuern die Auflistung der Eingabe und haben sonst keine Bedeutung.

Die MASTER-Anweisung

*MASTER m

Parameter:

m	neues "Master-Zeichen", d. h. neues UPDATE-Steuerzeichen in der Eingabe-Datei.

Die MASTER-Anweisung ändert das UPDATE-Steuerzeichen für Anweisungen auf der Eingabe-Datei. Anweisungen auf der Bibliothek sowie auf der SOURCE-Datei werden davon nicht betroffen.

Die MOVEDK-Anweisung

*MOVEDK $name_1$:$name_2$
*MOVEDK $name_1$:.

Parameter:

$name_1$	Name des Decks oder Common-Decks, das an eine andere Stelle gebracht werden soll.
$name_2$	Name des Decks oder Common-Decks, hinter welches das Deck $name_1$ plaziert werden soll.
.	Das Deck $name_1$ soll an den Anfang der Bibliothek plaziert werden.

Die MOVEDK-Anweisung veranlaßt UPDATE, ein vollständiges Deck von seiner gegenwärtigen Position an eine andere Stelle der Bibliothek zu verlagern. Die Numerierung innerhalb des verlagerten Decks bleibt unverändert.

Die PURGE-Anweisung

*PURGE id_1 <,id_2 ,...id_j .id_k ,...,id_n ,..>

Parameter:

id_1	Name einer Modifikation.
id_j .id_k	Bereich von Modifikationen (inklusiv).
id_n ..	Modifikation id_n und alle Modifikationen, die auf id_n folgen.

Die PURGE-Anweisung macht alle angegebenen Modifikationen rückgängig und stellt den Zustand davor wieder her. PURGE beginnt mit der letzten angegebenen Modifikation und arbeitet in umgekehrter Reihenfolge. Die Anweisung arbeitet ähnlich wie YANK, jedoch ist das Ergebnis im Gegensatz zu YANK irreversibel.

Die PURGEDK-Anweisung

*PURGEDK name

Parameter:

name Name des Decks oder Common-Decks, das gelöscht werden soll.

Die PURGEDK-Anweisung veranlaßt UPDATE dazu, ein komplettes Deck aus der Bibliothek zu entfernen.

Die READ-Anweisung

*R<EA>D dn

Parameter:

dn Name der Datei, von der gelesen werden soll.

Die READ-Anweisung sorgt dafür, daß UPDATE von der angegebenen Datei liest, beginnend ab der augenblicklichen Position. Bei Dateiende schaltet UPDATE wieder auf die vorherige Eingabe-Datei um. READ-Anweisungen können überall auftreten, dürfen aber nicht rekursiv sein.

Die RESTORE-Anweisung

*RESTORE id_1.seq_1,id_2.seq_2 (Bereich)
*RESTORE id_1.seq_1,seq_2 (Bereich, Kurzform)
*RESTORE id_1.seq_1 (einzelne Zeile)

Parameter:

id_1 Name eines Decks oder einer Modifikation.

seq_1,seq_2 Zeilennummern.

Die RESTORE-Anweisung reaktiviert Zeilen bzw. Zeilenbereiche, die zuvor mit DELETE gelöscht wurden; außerdem kann sie neue Zeilen hinzufügen. Neu einzufügende Zeilen folgen unmittelbar auf die RESTORE-Anweisung und werden nach den reaktivierten Zeilen eingefügt. Ein Bereich von zu reaktivierenden Zeilen darf das Ende eines Decks nicht überschreiten.

Die REWIND-Anweisung

*REWIND dn

Parameter:

dn Name der Datei, die an den Anfang positioniert werden soll.

Die REWIND-Anweisung positioniert eine lokale Datei an den Anfang.

Die SEQ-/NOSEQ-Anweisung

```
*SEQ
*NOSEQ
```

Die SEQ-/NOSEQ-Anweisungen starten bzw. beenden das Schreiben von Zeilennumerierungen auf der COMPILE-Datei. Sie werden ignoriert, wenn keine COMPILE-Datei verlangt wurde.

Die SKIPF-Anweisung

```
*SKIPF dn<,n>
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

dn	Name der Datei, auf der ein Dateiabschnitt übersprungen werden soll.
n=1	Anzahl der Dateiabschnitte, die übersprungen werden sollen.

Die SKIPF-Anweisung überspringt einen oder mehrere Dateiabschnitte auf einer lokalen Datei.

Die WEOF-Anweisung

```
*WEOF
```

Die WEOF-Anweisung sorgt dafür, daß eine EOF-Marke auf die COMPILE-Datei geschrieben wird. Diese Anweisung wird in ein Deck, Common-Deck oder einen Eingabe-Text eingefügt und erhält eine Zeilennumerierung.

Die WIDTH-Anweisung

```
*WIDTH dw
```

Parameter:

dw	Data Width (Zeilenbreite) auf der COMPILE-Datei. Die Spalten q.dw+1 bis dw+15 enthalten die UPDATE-Numerierung.

Die WIDTH-Anweisung ändert die Zeilenbreite auf der COMPILE-Datei. Sie wird ignoriert, wenn keine COMPILE-Datei erzeugt wird. Die Zeilenbreite in der Bibliothek oder auf der SOURCE-Datei wird davon nicht betroffen. Wird eine größere Zeilenbreite definiert, als in der Bibliothek vorhanden, so wird auf der COMPILE-Datei mit Leerzeichen aufgefüllt; wird eine kleinere Zeilenbreite definiert, so wird rechts abgeschnitten.

Die YANK-/UNYANK-Anweisung

```
*YANK id1 ,id2 ,...idj .idk ,...,idn ,..
*UNYANK id1 ,id2 ,...idj .idk ,...,idn ,..
```

Parameter:

id_1	Name eines Decks, Common-Decks oder einer Modifikation.
id_j .id_k	Bereich von Decks, Common-Decks oder Modifikationen (inklusiv).

id_n.. Deck oder Modifikation idn und alle IDs, die auf idn folgen.

Die YANK-/UNYANK-Anweisungen löschen (deaktivieren) bzw. restaurieren (aktivieren) Decks, Common-Decks oder Modifikationen einer Bibliothek. Die YANK-Anweisung löscht (deaktiviert) alle Zeilen eines Decks oder Common-Decks unabhängig davon, ob sie Orginalzeilen oder Modifikationen sind. Die beiden Anweisungen können an beliebiger Stelle als Eingabe erscheinen. Sie können innerhalb einer Modifikation stehen, werden aber keiner Modifikation zugeordnet.

Die COMMENT-Anweisung

*/ optionaler Kommentar

Die Comment-Anweisung wird mit auf die Ausgabe-Datei geschrieben. Das gültige Kommentarzeichen wird von der UPDATE-Steueranweisung übernommen.

5.2.4 Beispiele

In diesem Abschnitt sollen ein paar der gebräuchlichsten UPDATE-Anwendungen in Form von Beispieljobs vorgestellt werden.

Erstellen einer Bibliothek

Es soll eine Bibliothek mit Namen EISNBART, welche das Deck GEDICHT und zwei Common-Decks namens VERS1 und VERS8 enthält, erstellt und auf dem MVS-Vorrechner transparent, also auf der MVS-Anlage nicht lesbar, gespeichert werden.

```
JOB,JN=userid.
ACCOUNT,AC=account,APW=a-geheim,US=userid,UPW=u-geheim.
UPDATE,P=0.
DISPOSE,DN=$NPL,SDN=EISNBART,DC=ST,^
DF=TR,MF=IB,TEXT='DSN=nnn.EISNBART,DISP=OLD',WAIT.
/EOF
*DECK GEDICHT
DOKTOR EISENBART

(UNBEKANNTER VERFASSER)

*CALL VERS1

*CALL VERS8
*COMDECK VERS1
ICH BIN DER DOKTOR EISENBART,
KURIER DIE LEUT NACH MEINER ART;
KANN MACHEN, DASS DIE BLINDEN GEHN
UND DASS DIE LAHMEN WIEDER SEHN.
*COMDECK VERS8
ES HAT EIN WEIB IN LANGENSALZ
EIN' ZENTNERSCHWEREN KROPF AM HALS;
DEN SCHNUERT' ICH MIT DEM HEMMSEIL ZU,
PROBATUM EST: SIE HAT NUN RUH.
/EOD
```

Modifizieren einer Bibliothek:

Es soll die Bibliothek EISNBART, die auf dem MVS-Vorrechner lagert, geholt und modifiziert werden. Es soll ein weiteres Common-Deck namens VERS4 und im bereits vorhandenen Deck GEDICHT ein paar Zeilen hinzugefügt werden.

```
JOB.
ACCOUNT,AC=account,APW=a-geheim,US=userid,UPW=u-geheim.
FETCH,DN=$PL,SDN=EISNBART,MF=IB,DF=TR,^
TEXT='DSN=nnn.EISNBART,DISP=OLD'.
UPDATE,F,N,C=0.
DISPOSE,DN=$NPL,SDN=EISNBART,DC=ST,^
DF=TR,MF=IB,TEXT='DSN=nnn.EISNBART,DISP=OLD',WAIT.
/EOF
*COMDECK VERS4
DES KUESTERS SOHN ZU DIDELDUM,
DEM GAB ICH ZEHN PFUND OPIUM;
DRAUF SCHLIEF ER JAHRE, TAG UND NACHT
UND IST BIS JETZT NOCH NICHT ERWACHT.
*IDENT KORR
*D GEDICHT.3
*I GEDICHT.7
*CALL VERS4
/EOF
```

Herausziehen von Decks zur Kompilation

Es sollen alle Decks der Bibliothek EISNBART auf die COMPILE-Datei geschrieben werden; Das Ergebnis steht in $CPL an der CRAY und wird mit der Dispose-Anweisung codiert, also auf der MVS-Anlage lesbar, auf die Datei COMP an der MVS-Anlage übertragen.

```
JOB,JN=nnnA.CO=BART3
ACCOUNT,AC=account,APW=a-geheim,US=userid,UPW=u-geheim.
FETCH,DN=$PL,SDN=EISNBART,MF=IB,DF=TR,^
TEXT='DSN=nnn.EISNBART,DISP=OLD'.
UPDATE,F,S,C,SQ.
DISPOSE,DN=$CPL,DC=ST,^
DF=CB,MF=IB,TEXT='DSN=nnn.COMP,DISP=OLD',WAIT.
/EOF
```

Und das ist das Ergebnis:

```
CSP               CRAY-1M SN 6/36 KONRAD ZUSE-ZENTRUM FUER INFORMATIONSTECHNIK
                                                              BERLIN  23.09.86
CSP
CSP           CRAY OPERATING SYSTEM            COS 1.15  ASSEMBLY DATE 19.09.86
CSP
CSP
CSP      JOB,JN=nnnA.CO=BART3
CSP      ACCOUNT,AC=,APW=,US=,UPW=.
CSP      FETCH,DN=$PL,SDN=EISNBART,MF=IB,DF=TR,^
CSP      TEXT=.
CSP      TEXT FIELD SENT TO FRONT END =
CSP      DSN=nnn.EISNBART,DISP=OLD
CSP
SCP      IEF237I 241  ALLOCATED TO SYS00023
SCP      SS004 - DATASET RECEIVED FROM FRONT END
SCP      IEF285I   nnn.EISNBART                                  KEPT
SCP      IEF285I   VOL SER NOS= DISK02.
CSP      UPDATE,F,S,C,SQ.
```

```
USER    UD001 - PL: $PL        PL DATE: 18.09.86   LAST ID: KORR
USER    UD003 - EMPTY INPUT FILE, DN =  $IN
CSP     DISPOSE,DN=$CPL,DC=ST,^
CSP     DF=CB,MF=IB,TEXT=,WAIT.
CSP     TEXT FIELD SENT TO FRONT END =
CSP     DSN=nnn.COMP,DISP=OLD
CSP
SCP     IEF237I 241  ALLOCATED TO SYS00025
SCP     IEF285I   nnn.COMP                                  KEPT
SCP     IEF285I   VOL SER NOS= DISK02.
CSP     END OF JOB
CSP
CSP
USER        JOB NAME -                          nnnA
USER        USER NUMBER -                       userid
USER        TIME EXECUTING IN CPU -             0000:00:00.1227
USER        TIME WAITING TO EXECUTE -           0000:01:16.8675
USER        I/O TIME -                          0000:00:01.8850
USER        TIME WAITING IN INPUT QUEUE -       0000:00:00.0805
USER        MEMORY * CPU TIME (MWDS*SEC) -              0.00582
USER        MEMORY * I/O TIME (MWDS*SEC) -              0.10560
USER        MINIMUM JOB SIZE (WORDS) -             32768
USER        MAXIMUM JOB SIZE (WORDS) -             97792
USER        MINIMUM FL (WORDS) -                   28672
USER        MAXIMUM FL (WORDS) -                   93696
USER        MINIMUM JTA (WORDS) -                   4096
USER        MAXIMUM JTA (WORDS) -                   4096
USER        DISK SECTORS MOVED -                     754
USER        FSS SECTORS MOVED -                        0
USER        USER I/O REQUESTS -                      108
USER        USER I/O SUSPENSIONS -                   224
USER        OPEN CALLS -                              15
USER        CLOSE CALLS -                             15
USER        MEMORY RESIDENT DATASETS -                 0
USER        TEMPORARY DATASET SECTORS USED -           3
USER        PERMANENT DATASET SECTORS ACCESSED -     225
USER        PERMANENT DATASET SECTORS SAVED -          0
USER        SZ-TOTAL: SECTORS OF DATA CREATED          5
USER        SECTORS RECEIVED FROM FRONT END -          1
USER        SECTORS QUEUED TO FRONT END -              1
USER        RESOURCE TIME -                     0000:00:00.5374
```

5.3 CRAY-spezifische Unterprogramme

In diesem Abschnitt werden einige systemnahe Unterprogramme aufgeführt, die in speziellen Systembibliotheken liegen und die nicht zum FORTRAN Standard gehören. Im Kapitel 9.5 sind die wichtigsten Routinen der Bibliothek für wissenschaftliche Anwendungen $SCILIB aufgeführt. Eine vollständige und ausführliche Dokumentation aller Routinen findet man im Library Reference Manual SR-0113.

Diese Routinen stehen ohne besondere Maßnahmen jedem Benutzerprogramm zur Verfügung, da sie in Systembibliotheken residieren, die beim Laden (LDR bzw. SEGLDR Aufruf) automatisch benutzt werden. Benutzerprogramme mit gleichen Namen haben beim Laden Vorrang, da der Lader die Systembibliotheken standardmäßig als letzte durchsucht. Werden die Systemroutinen gewünscht, obwohl in anderen beim Laden angegebenen Bibliotheken Routinen gleichen Namens vorhanden sind, so ist durch eine geeignete Reihenfolge sicherzustellen, daß die Systemroutinen verwendet werden.

5.3.1 System-Hilfsroutinen

CLEARFI — CALL CLEARFI unterdrückt Gleikomma-Unterbrechungen und verhindert so die Feststellung von Zerodivide und Overflow. CALL CLEARFI macht unter Umständen die Vektorverarbeitung einer darauf folgenden Schleife erst möglich.

SETFI — CALL SETFI schaltet o.g. Zustand wieder aus.

Beispiel:

```
CFT,OPT=FULLIFCON.
...
CALL CLEARFI
DO 1 I=1,N
Y(I)=1.
IF (X(I) .NE. 0.) Y(I)=SIN(X(I))/X(I)
1 CONTINUE
CALL SETFI
```

SMACH — Abfragen der Maschinenkonstanten (FUNCTION)
Aufruf: SMACH(I)
I=1 Resultat: .7105E-0014 (kleinste Zahl e, so daß 1+-e ≠ 1 ist)
I=2 Resultat: .1290E-2449 (kleinste Zahl größer als 0)
I=3 Resultat: .7750E+2450 (größte Zahl)

NUMARG — Anzahl der Argumente (FUNCTION). Aufruf: K=NUMARG( ). Ein Unterprogramm kann mit weniger Argumenten aufgerufen worden sein, als es Formalparameter hat. NUMARG liefert die Anzahl der Argumente beim Aufruf.

TRBK — Traceback als Testhilfe. CALL TRBK bewirkt eine Ausgabe auf dem Protokoll, aus der unter anderem die Aufrufstelle (Unterprogramm, Block) hervorgeht.

JNAME — Identifizierung des Benutzers und des Jobs. Aufruf: CALL JNAME(I). Man erhält den Jobnamen.

DATE — Aktuelles Datum. Aufruf: CALL DATE(I). Man erhält das aktuelle Datum in der Form 'dd.mm.yy' (Berliner Modifikation).

JDATE — Aktuelles Julianisches Datum. Aufruf: CALL JDATE(I). Man erhält das Julianische Datum in der Form 'yyddd'.

CLOCK — Aktuelle Uhrzeit. Aufruf: CALL CLOCK(I). Man erhält die Uhrzeit in der Form 'hh:mm:ss'.

TIMEF — Realzeitdifferenz. Aufruf: CALL TIMEF(R). Man erhält die Realzeitdifferenz zwischen dem aktuellen Aufruf und dem ersten Aufruf von TIMEF in Millisekunden.

SECOND — Verbrauchte CPU-Zeit. Aufruf: CALL SECOND(R). Man erhält die bisher verbrauchte CPU-Zeit des Jobs in Sekunden.

TREMAIN — Verbleibende CPU-Zeit. Aufruf: CALL TREMAIN(R). Man erhält die Restzeit bis zum Ablauf der Jobzeit in Sekunden.

5.3.2 Aufruf von COS-Steuerkarten aus Fortran-Programmen

Mit dem Aufruf dieser Unterprogramme kann man vom FORTRAN-Programm aus Steueranweisungen geben. Folgende Routinen stehen zur Verfügung:

ACCESS	ADJUST	DELETE	MODIFY	SAVE	PERMIT
ACQUIRE	DISPOSE	FETCH	SUBMIT	ECHO	
ASSIGN	RELEASE				

Beispiel, wie die Argumente übergeben werden:

```
CALL ASSIGN(I,'DN'L,'$OUT'L,'A'L,'FT07'L)
```

Vom Programm wird "ASSIGN,DN=$OUT,A=FT07." veranlaßt. I enthält nach Ausführung des Kommandos einen Hinweis, inwieweit die Ausführung erfolgreich war:

I=0	Anweisung erfolgreich ausgeführt.
I>0	Anweisung fehlerhaft beendet. Eine Erläuterung des Fehlers findet man im COS Message Manual, SR-0039, unter 'AB<I>'.

5.4 Analysieren von Dateien

5.4.1 Vergleichen von Dateien (COMPARE)

Mit der COMPARE-Anweisung können geblockte Dateien miteinander verglichen werden. Die Ausgabe enthält eine Liste mit den Stellen, an denen Unterschiede gefunden wurden, die verschiedenen Teile selbst und den Nummern der Meldungen, die die Art des Unterschiedes bezeichnen. Die Meldungen sind im CRAY-OS MESSAGE MANUAL, SR-0039 beschrieben. COMPARE positioniert beide Eingabe-Dateien vor und nach dem Vergleich an den Anfang.

```
COMPARE,A=adn,B=bdn,<L=ldn,DF=df,ME=maxe,CP=cpn,CS=csn,CW=cw1:cw2,Abort=ac>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

A=adn/B=bdn	Namen der Dateien, die verglichen werden sollen.
L=ldn =$OUT	Name der Datei, die eine Liste der Unterschiede enthält. ldn kann nicht zugleich adn oder bdn sein.
DF=df	Format der Eingabe-Dateien.
=B	Binär: Die Eingabe-Dateien werden bitweise miteinander verglichen. Wenn Unterschiede auftreten, werden die unterschiedlichen Worte oktal und als ASCII-Zeichen ausgedruckt. Der Wortzähler ist dezimal und beginnt bei dem ersten Wort der Datei mit 1.
=T	Text: Die Eingabe-Dateien werden als Text verglichen, z.B. wird ein Datensatz, bei dem Leerzeichen komprimiert dargestellt sind, wie ein Datensatz angesehen, bei dem die Leerzeichen einzeln dargestellt sind. Der Zähler ist dezimal und beginnt bei dem ersten Satz mit 1.
ME=maxe =100	Maximale Anzahl an Unterschieden, die gedruckt werden sollen.
CP=cpn =0	Bereich, der mit ausgedruckt werden soll: cpn Sätze vor und nach der Stelle, an der Unterschiede auftreten, werden gedruckt. CP ist nur bei Text-Dateiabschnitten erlaubt.

CS=csn =0	Bereich, der mit durchsucht wird: csn Sätze vor und nach der Stelle, an der der Unterschied gefunden wurde, werden auf Übereinstimmungen durchsucht. Wenn eine Übereinstimmung in dem angegebenen Bereich gefunden wurde, wird bei beiden Dateien an der Stelle der Übereinstimmung weiter verglichen.
CW=cw (CW=cw1:cw2) =1:133	Wenn CW=cw angegeben ist, werden die Spalten 1 bis cw verglichen. Wenn CW=cw1:cw2 angeben ist, werden nur die Spalten cw1 bis cw2 miteinander verglichen. CW gilt nur für Text-Dateiabschnitte.
ABORT=ac	Wenn mehr als ac Unterschiede gefunden wurden, wird der Vergleich abgebrochen. Nur ABORT ist identisch mit ABORT=1.

5.4.2 Datei-Dump (DSDUMP)

Mit der DSDUMP-Anweisung kann eine Datei ganz oder teilweise symbolisch aufgelistet werden. Der Inhalt der geblockten oder ungeblocken Datei wird Wort für Wort als oktale oder hexadezimale Integerzahl sowie ASCII-Zeichenkette interpretiert.

Es gibt zwei Gruppen von Parametern. Die Ix-Parameter verweisen auf das Wort, den Satz, Dateiabschnitt oder Sektor, von dem ab gelistet werden soll. Die Länge des Ausdrucks wird mit den Nx-Parametern gesteuert. Die Werte der beiden Parametern können in den folgenden Arten angegeben werden:

* als einfache Zahl, z.B. 1234 (wird als dezimale Zahl interpretiert),
* als explizit gekennzeichnete dezimale Zahl (z.B. D'1234'),
* als explizit gekennzeichnete oktale Zahl (z.B.O'2322).

```
DSDUMP,I=idn<,O=odn,DF=df,IW=n,NW=n,IR=n,NR=n,^
       IF=n,NF=n,IS=n,NS=n,Z,DB=db,DSZ=sz>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

I=idn (DN=idn)	Name der Datei, die gelistet werden soll.
O=odn (L=ldn) =$OUT	Name der Datei, die die Liste erhält.
DF=df =B =U	Datei-Blockung geblockt ungeblockt
IW=n	Nummer des Wortes, von dem ab jeder Satz bzw. Sektor der Eingabe-Datei gelistet wird. Voreinstellung ==> Z-Parameter.
NW=n =1	Anzahl der Worte eines jeden Satzes bzw. Sektors, die gelistet werden sollen. Wenn bis zum Ende eines jeden Satzes bzw. Sektors gelistet werden soll, kann das Schlüsselwort allein angegeben werden.

Parameter, die zusammen mit DF=B angegeben werden können:

IR=n	Nummer des Satzes, von dem ab auf jedem Dateiabschnitt der Eingabe-Datei gelistet wird. Voreinstellung ==> Z-Parameter.
NR=n =1	Anzahl der Sätze eines jeden Dateiabschnitts, von denen eine Liste erzeugt werden soll. Wenn die Liste bis zum Ende jedes

Dateiabschnitts gewünscht wird, kann das Schlüsselwort allein angegeben werden.

IF=n — Nummer des Dateiabschnitts, von dem ab die Eingabe-Datei gelistet wird. Voreinstellung ==> Z-Parameter.

NF=n
=1 — Anzahl der Dateiabschnitte einer Datei, die gelistet werden sollen. Wenn bis zum Ende der Datei gelistet werden soll, kann das Schlüsselwort allein angegeben werden.

Parameter, die zusammen mit DF=U angegeben werden können:

IS=n — Nummer des Sektors, von dem ab die Eingabe-Datei gelistet wird. Voreinstellung ==> Z-Paramter.

NS=n
=1 — Anzahl der Sektoren einer Datei, die gelistet werden soll. Wenn bis zum Ende jeder Datei eine Liste gewünscht wird, kann das Schlüsselwort allein angegeben werden.

Z — Wenn Z angegeben ist, beziehen sich alle Ix-Parameter (IW,IR,IF und IS) auf die Basis 0, ansonsten auf 1. Die zwei folgenden DSDUMP-Anweisungen zeigen auf das gleiche erste Wort, von dem ab die Datei-Liste ausgeführt wird.

```
DSDUMP,...,IW=4096.
DSDUMP,...,Z,IW=4095.
```

Der Z-Parameter hat keinen Einfluß auf die Nx-Parameter.

DB=db — db gibt die Zahlendarstellung an, in welchem die Liste ausgedruckt wird:
=OCTAL oder O — oktal
=HEX oder X — hexadezimal

DSZ=sz — sz bezeichnet die Formate der einzelnen Werte:
=WORD oder W — in Worten (64 Bits)
=PARCEL oder P — in Paketen zu 16 Bit

Das Format der Ausgabedatei ist:

Wortzähler	oktale Interpretation	ASCII-Zeichen Interpretation

DB,DSZ	numerische Interpretation	ASCII-Interpretation
OCTAL, WORD	vier 22-stellige oktale Zahlen	32 ASCII-Zeichen
HEX, WORD	vier 16-stellige hexadez. Zahlen	32 ASCII-Zeichen
OCTAL, PARCEL	16 vier-stellige oktale Zahlen	keine (wegen Platzmangels)
HEX, PARCEL	16 vier-stellige hexadez. Zahlen	32 ASCII-Zeichen

Eine Reihe mit Sternchen bedeutet, daß eine oder mehrere Gruppen mit der vorausgehenden übereinstimmen.

5.4.3 Listen der Struktur einer Datei (ITEMIZE)

Die ITEMIZE-Anweisung druckt einen formatierten Report der Struktur einer geblockten Datei.

ITEMIZE<,DN=dn,L=odn,NREW,NF=n,T,BL,E,B,X>.

Parameter: (die Voreinstellung (default) ist unterstrichen)

DN=dn =$OBL	Name der lokalen Datei, die untersucht werden soll.
L=odn =$OUT	Name der Datei, auf die die Ausgabe geschrieben werden soll.
NREW	NO REWIND. Gibt an, daß die angebene Datei nicht an den Anfang positioniert werden soll. Fehlt der Parameter, so wird die zu untersuchende Datei vor und nach dem ITEMIZE-Lauf an den Anfang positioniert.
NF=n =1	Number of Files. Anzahl der Dateiabschnitte, die auf der Datei untersucht werden sollen. Wird nur NF angegeben, so werden alle Dateiabschnitte der Datei untersucht.
T	Truncation. Veranlaßt, daß alle Zeilen auf der Ausgabedatei auf 80 Zeichen begrenzt werden. Dieser Parameter schließt die Parameter E, B und X aus.
BL	Burstable Listing. Wird dieser Parameter angegeben, so beginnt die Ausgabe eines jeden Dateikopfes immer auf einer neuen Seite. Standardmäßig wird eine kompakte Ausgabe erzeugt, wobei nur dann auf einer neuen Seite angefangen wird, wenn die laufende Seite fast voll ist.
E	Entry-Points. Wird E angegeben, so werden bei binären Programmbibliotheken die Eingangspunkte aller Programme mit ausgegeben.
B	Blocks. B gibt an, daß bei binären Programmbibliotheken alle Eingangspunkte und COMMON-Block-Informationen mit ausgegeben werden. B überschreibt den E-Parameter.
X	Externals. X veranlaßt, daß bei binären Programmbibliotheken zusätzlich zu den bei B angegebenen Werten alle EXTERNAL-Informationen mit ausgegeben werden. X überschreibt den B-Parameter.

Anmerkung: Eine UPDATE-Bibliothek wird nur dann als solche erkannt, wenn sie der einzige Dateiabschnitt in der Datei ist.

6. Kommandoprozeduren

Der Zweck von Kommandoprozeduren besteht in erster Linie darin, Kommandos nach eigenen Bedürfnissen zu erstellen. Eine bestimmte Reihenfolge von Anweisungen, die vom Benutzer häufig benötigt werden, kann mit Hilfe der Kommandosprache (Job Control Language JCL) durch eine Kommandoprozedur ersetzt werden.

6.1 Aufbau der Kommandosprache

Die COS-Kommandosprache erlaubt drei grundlegend verschiedene Strukturen:

1. Einfache sequentielle Anweisungen: dabei werden Anweisungen eine nach der anderen abgearbeitet.

2. Bedingte Anweisungsblöcke: Eine Reihe von Anweisungen werden in Abhängigkeit von einer Bedingung ausgeführt. Dazu werden die Anweisungen IF, ELSE, ELSEIF und ENDIF benutzt.

3. Iterative Anweisungsblöcke: Eine Reihe von Anweisungen wird so oft ausgeführt, bis eine bestimmte Bedingung erfüllt ist. Iterative Anweisungsblöcke sind Schleifen, die mit LOOP, ENDLOOP und EXITLOOP gebildet werden.

Es gibt einfache und komplexe Prozeduren. Einfache Prozeduren bestehen aus einer Reihe von Steueranweisungen, bedingten Anweisungsblöcken oder Schleifen, die auf einer Datei liegen. Die Prozedur wird mit der CALL-Anweisung aufgerufen. Sie darf weder PROC noch ENDPROC enthalten.

Komplexe Prozeduren bestehen aus der Anweisung PROC, einer Parameteranweisung, dem Prozedurrumpf mit den Steueranweisungen und eventuell Daten. Eine komplexe Prozedur kann auf zwei Arten aufgerufen werden:

1. Aufruf des Prozedurnamens, dazu muß die Prozedur dem System bekannt sein, entweder dadurch, daß sie auf der Datei $PROC steht oder auf einer lokalen Datei, die als Parameter mit der LIBRARY-Anweisung aufgerufen wird. Beim Aufruf werden Parameter ersetzt.

2. Aufruf mit der CALL-Anweisung: Bei der CALL-Anweisung wird der Name der Datei angegeben, auf dem die Prozedur steht. Zusätzlich muß noch der Parameter CNS (Crack Next Statement) angegeben werden, damit die Parameteranweisung erkannt und ausgeführt wird.

6.2 Ausdrücke und Variablen

Wie bei Programmiersprachen gibt es auch bei der Kommandosprache Ausdrücke und Variablen. Ein Ausdruck ist als Kette von Operanden und Operatoren definiert.

6.2.1 Operanden

Operanden von Ausdrücken teilen sich in die folgenden Typen ein:

* Konstanten

  Beispiel:

  +19, -108 (dezimal), 17B, 104B (oktal)

* Zeichenkonstanten bestehen aus einem bis acht Zeichen

Beispiel:

'ABC'L linksbündig, mit Nullen aufgefüllt,
'ABC'R rechtsbündig, mit Nullen aufgefüllt,
'ABC'H linksbündig, mit Leerzeichen aufgefüllt.

* Symbolische Variable sind in der folgenden Tabelle aufgeführt:

Symbol	gesetzt von	Beschreibung
J0-J7	Benutzer	Lokales Job-Pseudoregister
G0-G7	Benutzer	Globales Job-Pseudoregister
FL	COS	Augenblickliche Feldlänge
FLM	COS	Maximale Feldlänge
SYSID	(Systemkonstante)	COS System Level "COS X.XX"
SID	COS	ID der Jobherkunft (MF)
ABTCODE	COS	Job-Abort-Code
TRUE	(Systemkonstante)	True-Wert => -1
FALSE	(Systemkonstante)	False-Wert =>0
TIME	COS	Zeit: hh:mm:ss
DATE	COS	Datum: dd.mm.yy
TIMELEFT	COS	Verbleibende Zeit des Jobs in ms

* Unterausdrücke sind Ausdrücke, deren Ergebnisse wiederum Operanden sind.

6.2.2 Operatoren

Operatoren werden in die folgenden Arten eingeteilt und wie bei der Programmiersprache FORTRAN verwendet.

1. Arithmetische Operatoren:
   + Addition (64 Bit)
   + als Vorzeichen
   - Subtraktion (64 Bit)
   - als Vorzeichen
   * Multiplikation (64 Bit)
   / Division (64 Bit)

2. Vergleichende Operatoren:
   .EQ. gleich
   .NE. nicht gleich
   .LT. kleiner als
   .GT. größer als
   .LE. kleiner oder gleich
   .GE. größer oder gleich

3. Logische Operatoren:
   .OR. oder
   .AND. und
   .XOR. exklusives oder
   .NOT. nicht

6.2.3 Abarbeitung von Ausdrücken

Ausdrücke werden von links nach rechts abgearbeitet; dabei werden Klammern und die Hierarchie der Verküpfungen berücksichtigt.

1. Multiplikation und Division
2. Addition, Subtraktion und Negation
3. Vergleiche
4. Komplementieren (.NOT.)
5. (.AND.)
6. (.OR.)
7. (.XOR.)

Anmerkung: Das Betriebssystem COS macht keine Typenüberprüfung bei Ausdrücken der Kommandosprache.

6.2.4 Strings

Ein String ist eine Kette von druckbaren Zeichen und wird als Parameterwert verwendet. Strings werden durch Hochkommata oder durch Klammern begrenzt. Hochkommata werden nie als Teil des Wertes eines Strings betrachtet, wogegen Klammern bedingt Wert eines Strings sein können.

```
'Alles im Eimer'
'Wie geht''s'
DN=($BLD)
KEY=(ABC.DEF)     ABC.DEF wird KEY zugewiesen
KEY=((ABC.DEF))   (ABC.DEF) wird KEY zugewiesen
KEY='ABC.DEF'     ABC.DEF wird KEY zugewiesen
```

6.3 Anweisungsblöcke

Es gibt zwei Arten von Anweisungsblöcken:

1. Bedingte Anweisungsblöcke:
   - IF definiert den Anfang eines bedingten Blockes.
   - ENDIF definiert das Ende eines bedingten Blockes.
   - ELSE wird benutzt, um die alternative Bedingung zu definieren.
   - ELSEIF definiert eine alternative Bedingung, die getestet wird, wenn eine vorhergehende Bedingung nicht erfüllt ist.
   - EXITIF definiert eine Bedingung. Der bedingte Anweisungsblock wird verlassen, wenn die Bedingung erfüllt ist.

2. Wiederholungsblöcke:
   - LOOP definiert den Anfang eines iterativen Anweisungsblockes,
   - ENDLOOP definiert das Ende eines iterativen Blockes,
   - EXITLOOP definiert eine Bedingung. Die Schleife wird beendet, wenn die Bedingung erfüllt ist.

6.3.1 Bedingte Anweisungsblöcke

IF - Beginn eines bedingten Blockes

Eine IF-Anweisung definiert den Beginn eines bedingten Blockes. Für jede IF-Anweisung muß es eine dazugehörige ENDIF-Anweisung geben.

```
IF (Ausdruck).
```

Parameter:

Ausdruck ist ein logischer oder vergleichender Ausdruck.

ENDIF - Ende eines bedingten Blockes

Eine ENDIF-Anweisung definiert das Ende eines bedingten Blockes.

```
ENDIF.
```

ELSE - Alternative Bedingung

Die ELSE-Anweisung wird benutzt, um die alternative Bedingung zu einer vorhergehenden IF- oder ELSEIF-Anweisung zu definieren. Die Anweisungen hinter der ELSE-Anweisung werden ausgeführt, wenn die Bedingungen aller vorhergehenden IF- und ELSEIF-Anweisungen nicht erfüllt wurden.

```
ELSE.
```

ELSEIF - Alternative Bedingung

Die ELSEIF-Anweisung definiert eine alternative Bedingung, die geprüft wird, wenn eine vorhergehende IF- oder ELSEIF-Bedingung nicht erfüllt ist. In einem bedingten Block können mehrere ELSEIF-Anweisungen stehen. Eine eventuell angegebene ELSE-Anweisung muß hinter allen ELSEIF-Anweisungen stehen. Die Anweisungen hinter der ELSEIF-Anweisung werden dann ausgeführt, wenn die IF-Bedingung und alle vorhergehenden ELSEIF-Bedingungen nicht erfüllt sind und diese ELSEIF-Bedingung erfüllt ist.

```
ELSEIF(Ausdruck).
```

6.3.2 Wiederholungsblöcke

EXITIF - Beenden einer Bedingung

Mit der EXITIF-Anweisung wird eine Bedingung angegeben. Wenn diese Bedingung erfüllt oder keine Bedingung angegeben ist, werden die nächsten Anweisungen bis zum Ende des IF-Blocks übersprungen. EXITIF springt also zum ENDIF.

```
EXITIF.
EXITIF(Ausdruck).
```

LOOP - Beginn eines iterativen Blockes

Die LOOP-Anweisung wird benötigt, um einen Schleifenanfang zu kennzeichnen. LOOP-Anweisungen können geschachtelt werden. Für jedes LOOP muß es jedoch in der gleichen Ebene eine ENDLOOP-Anweisung geben.

```
LOOP.
```

ENDLOOP - Ende eines iterativen Blockes

Die ENDLOOP-Anweisung beendet einen iterativen Anweisungsblock. Die Ausführung einer ENDLOOP-Anweisung bewirkt, daß zur dazugehörigen LOOP-Anweisung zurückgesprungen wird.

```
ENDLOOP.
```

EXITLOOP - Beenden der Iteration

Mit der EXITLOOP-Anweisung wird eine Bedingung angegeben. Wenn die Bedingung erfüllt ist, wird die Schleife an der Stelle beendet. In geschachtelten Schleifen bezieht sich eine EXITLOOP-Anweisung auf die innerste Schleife.

```
EXITLOOP.
EXITLOOP(Ausdruck).
```

Wenn der Parameter Ausdruck weggelassen wird, bedeutet das ein unbedingtes Ende der Schleife.

6.4 Prozeduren

Wie schon in 6.1 "Aufbau der Kommandosprache" beschrieben, gibt es zwei Arten von Prozeduren:

1. einfache Prozeduren und
2. komplexe Prozeduren.

Das folgende Bild zeigt den Aufbau von komplexen Prozeduren:

```
PROC.
prototype statement
    .
    .
COS-Anweisungen
    .
    .
&DATA,dn.
    .
    .
Daten
    .
    .
ENDPROC.
```

Da einfache Prozeduren nur aus dem Prozedurrumpf bestehen, also dem COS-Anweisungsteil, werden in diesem Abschnitt nur die komplexen Prozeduren beschrieben, die mit PROC. beginnen und mit ENDPROC. enden. Die im Prozedurrumpf zu ersetzenden Parameter aus dem Prozedurkopf werden durch ein vor den Parameter gestelltes & (kaufm. und) gekennzeichnet. Von eventuell folgendem Text

müssen Parameternamen durch das Trennzeichen _ (underline) getrennt werden.

PROC - Beginn der Definition einer Prozedur

Die PROC-Anweisung definiert den Beginn einer Prozedur

PROC.

Prototype Anweisung - Erklärung der Prozedur

Die Prototype-Anweisung hat zwei Aufgaben. Zum ersten wird der Name der Prozedur erklärt, zum zweiten können formale Parameter angegeben werden, die bestimmen, wie die Prozedur aufgerufen und welche Voreinstellungswerte verwendet werden.

name,p_1,p_2,p_3,...,p_n.

Parameter:

name	Prozedurname; Der Prozedurname kann ein bis acht Zeichen lang sein.
p_i	Formaler Parameter, der eines der nachfolgenden Formate hat. Formale Positionsparameter müssen vor Parametern angegeben werden, die mit einem Schlüsselwort beginnen.
key_i	Positionsparamter
key_i =dwert:kwert	Parameter, die mit einem Schlüsselwort beginnen.
dwert	Voreinstellung, die eingesetzt wird, wenn das Schlüsselwort beim Aufruf der Prozedur nicht angegeben wird.
kwert	Wert, der eingesetzt wird, wenn nur das Schlüsselwort angegeben wird.
key_i =	diese Angabe bewirkt, daß kein Voreinstellungswert eingesetzt wird; der Wert des Parameters wird erst beim Aufruf der Prozedur angegeben.

RETURN - Verlassen einer Prozedur

Mit der RETURN-Steueranweisung wird das Abarbeiten von Anweisungen einer Prozedur beendet und in die rufende Prozedur bzw. in den rufenden Steueranweisungsteil zurückgesprungen.

RETURN.

Nach Abarbeiten der RETURN-Steueranweisung ohne Parameter wird die Verabeitung mit der auf den Prozeduraufruf folgenden Steueranweisung fortgesetzt.

RETURN,ABORT.

Wird der Parameter ABORT bei der RETURN-Anweisung angegeben, so wird die Verarbeitung der Steueranweisungen hinter der nächsten EXIT-Steueranweisung fortgesetzt, die auf den Prozeduraufruf folgt.

Hinweis: dieses Kommando ist nicht identisch mit dem RETURN-Kommando an der NOS/BE-Anlage, sondern entspricht dort dem REVERT-Kommando!

&DATA - Prozedurdaten

Mit der &DATA-Anweisung konnen Daten in der Prozedur angegeben werden, die auf eine lokale Datei geschrieben wurden. Diese Datei kann von Programmen wie z.B. CFT oder UPDATE verwendet werden. Die Bedeutung der &DATA-Anweisung liegt darin, daß die Daten erst nach Ersetzen der Parameter mit den Werten des Aufrufs auf die lokale Datei geschrieben werden. Es konnen mehrere &DATA-Anweisungen bei einer Prozedur-Definition angegeben werden.

```
&DATA,dn.
```

dn — Name der Datei, auf die die Daten geschrieben werden, die der &DATA-Anweisung folgen. dn muß angegeben werden.

ENDPROC - Ende der Prozedurdefinition

Mit der ENDPROC-Anweisung wird das Ende einer Prozedur-Definition angegeben.

```
ENDPROC.
```

6.4.1 Anlegen von Prozedurbibliotheken

Eine komplexe Prozedur wird wie Steueranweisungen im Anweisungsteil des Jobs geschrieben. Wenn das System eine PROC-Anweisung findet, werden die nachfolgenden Anweisungen bis zum Ende der Prozedur (ENDPROC) nicht ausgeführt, sondern auf die Datei $PROC gelegt ("INLINE-Prozedur"). Die Prozeduren stehen dort in lesbarer Form, durch End-of-File-Marken voneinander getrennt. Als letzter Dateiabschnitt wird das Verzeichnis angehängt. Die Prozedur-Bibliothek $PROC ist immer vorhanden, sofern sie nicht durch eine LIBRARY-Anweisung ausgeschlossen wurde.

Die Prozedur steht jedoch nur dem Job zur Verfügung, der sie definiert hat. Wenn sie auch von anderen Jobs aufgerufen werden soll, so muß die Datei $PROC permanent gemacht werden.

Beispiel für eine INLINE-Prozedur:

```
H999,STORY.
JOB.
ACCOUNT,AC=,APW=,US=,UPW=.
PROC.
PROTO,P1=:ABC.
CFT,I=&P1.
LDR.                        (besser: SEGLDR,GO.)
RETURN.
&DATA,DFC.
      PROGRAM MTD
      WRITE(*,'(A)')' MESSAGE FROM J C L '
      END
ENDPROC.
PROTO,P1=DFC.
SAVE,DN=$PROC,PDN=PROCTESTLIB.
*EOR
```

Anlegen einer Prozedur-Bibliothek: Die Prozeduren werden im Anweisungsteil des Jobs definiert und auf der Datei $PROC abgelegt. Die Prozedur Bibliothek wird unter dem Namen PROCTESTLIB permanent gespeichert.

Die Definition der Prozerdur wird im Protokoll gelistet:

```
CSP      CRAY X-MP/24     KONRAD-ZUSE-ZENTRUM FUER INFORMATIONSTECHNIK
                                  BERLIN
CSP
CSP                CRAY OPERATING SYSTEM              COS 1.15  ASSEMBLY DATE
                                                               19.01.87
CSP
CSP
CSP      JOB,JN=HxxxYLE.
CSP      ACCOUNT,AC=,APW=,US=,UPW=.
CSP      <DEFINITION>     PROC.
CSP      <DEFINITION>     PROTO,P1=:ABC.
CSP      <DEFINITION>     CFT,I=&P1.
CSP      <DEFINITION>     LDR.
CSP      <DEFINITION>     RETURN.
CSP      <DEFINITION>     &DATA,DFC.
CSP      <DEFINITION>           PROGRAM MTD
CSP      <DEFINITION>           WRITE(*,'(A)')' MESSAGE FROM J C L '
CSP      <DEFINITION>           END
CSP      <DEFINITION>     ENDPROC.
CSP      PROTO,P1=DFC.
CSP      CS136 - &DATA CREATED DSN DFC
CSP      CFT,I=DFC.
USER     CF000 - CFT VERSION -   20.01.87 1.14
USER     CF998 - CFT 1.14 EDITION 229.85
USER     CF001 - COMPILE TIME =    0.0045 SECONDS
USER     CF002 -         3 LINES,         3 STATEMENTS
USER     CF003 -     53738 WORDS,     14542 I/O BUFFERS USED
CSP      LDR.
USER     LD000 - BEGIN EXECUTION
CSP      SAVE,DN=$PROC,PDN=PROCTESTLIB.
PDM      PD000 - PDN = PROCTESTLIB       ID =           ED =    2  OWN =
PDM      PD000 - SAVE     COMPLETE
CSP      RETURN.
CSP      END OF JOB
*EOR
```

Die folgenden Beispiele zeigen, wie die Bibliothek von weiteren Jobs verwendet werden kann:

```
JOB.
ACCOUNT,AC=XXXX,APW=YYYY,US=XXX,UPW=YYY.
   :
ACCESS,DN=$PROC,PDN=PROCTESTLIB.
   :
PROTO,P1=DFC.
   :
/EOF
```

Aufruf einer Prozedur aus der Bibliothek: Die Prozedur-Bibliothek wird als lokale Datei $PROC angemeldet. Da die Datei $PROC als globale Bibliothek vorhanden ist, kann die Prozedur PROTO nach dem ACCESS aufgerufen werden.

Wenn gleichzeitig mehrere Prozedurdateien vorhanden sein sollen oder der lokale Prozedurname nicht $PROC ist, müssen die Prozedurdateien mit der LIBRARY- Anweisung dem System bekannt gemacht werden.

6.4.2 Erweitern einer Prozedur-Bibliothek

Es ist für eine spatere Version von COS vorgesehen, daß auch Prozedur-Bibliotheken mit BUILD bearbeitet werden konnen. Bis jetzt müssen Prozeduren im Anweisungsteil definiert werden und das System legt sie auf der Datei $PROC ab. Deshalb muß die zu erweiternde Datei als lokale Datei $PROC mit Schreiberlaubnis (UQ-Parameter beim ACCESS) vorhanden sein. Weitere im Anweisungsteil definierte Prozeduren werden vom System an diese Datei angehängt und der letzte Dateiabschnitt, das Verzeichnis, wird auf den neuen Stand gebracht. Wenn eine definierte Prozedur denselben Namen hat wie eine in der Bibliothek bereits vorhandene, gilt immer die neu hinzugekommene. Da ein physikalisches Ersetzen nicht moglich ist, werden Prozeduren verändert, indem die geänderte Version hinzugefugt wird. Die Datei wird in diesem Falle immer großer.

Beispiel:

```
JOB.
ACCOUNT,AC=XXXX,APW=YYYY,US=XXX,UPW=YYY.
  :
ACCESS,DN=$PROC,PDN=PROCTESTLIB,UQ.
  :
PROC.                                   besser:
LOAD,NOGO=:NX,LIBRARY=:FORTLIB.         LOAD,GO=:GO,LIBRARY=:FORTLIB.
LDR,&NOGO,LIB=&LIBRARY.                 SEGLDR,&GO,CMD='LIB=&LIBRARY'.
ENDPROC.
  :
/EOF
```

Erweitern einer Prozedur-Bibliothek: Die Prozedur wird an die Bibliothek angehangt und das Verzeichnis wird auf den neuen Stand gebracht.

6.5 Nützliche COS-Anweisungen für Kommandoprozeduren

6.5.1 Aufrufen von Anweisungen von einer Datei (CALL)

Eine CALL-Anweisung bewirkt, daß das System beginnt, die Anweisungen auf der angegebenen Datei auszufuhren. Das System liest und bearbeitet die Anweisungen auf der Datei, bis eine End-of-File-Marke oder eine RETURN-Anweisung erreicht ist. Danach wird wieder von der rufenden Datei gelesen. CALL-Anweisungen konnen bis zu siebenfach geschachtelt werden.

Auf der angegebene Datei konnen entweder nur Steueranweisungen oder auch Kommandoprozeduren stehen. Eine Parametersubstitution bei Prozeduren ist moglich, wenn der Parameter CNS angegeben wird. In diesem Fall muß die erste Anweisung auf der Datei eine Prototype-Anweisung sein (==> 6.4).

```
CALL,DN=dn<,CNS>.
```

Parameter:

DN=dn	Name der Datei, von welcher das System liest.
CNS	Crack Next Statement. Wenn dieser Parameter angegeben ist, wird angenommen, daß auf der Datei DN eine komplexe Prozedur steht. Eine Parametersubstitution wird in Abhängigkeit der Anweisung ausgeführt, die der CALL-Anweisung folgt, welche den Prozedur-Aufruf enthalt.

Beispiel für eine komplexe Prozedur:

```
H999,STCRY.
JOB.
ACCOUNT,AC=XXXX,APW=YYYY.
COPYF,I=$IN,O=PRF.
REWIND,DN=PRF.
CALL,DN=PRF,CNS.
*,P1=DFC.                dies ist sonst eine kommentaranweisung!
/EOF
PROTO,P1=:ABC.
CFT,I=&P1.
LDR.                      (besser: SEGLDR,GO.)
&DATA,DFC.
      PROGRAM MTD
      WRITE(*,'(A)')' MESSAGE FROM J C L '
      END
*EOR
```

6.5.2 Setzen von System-Variablen (SET)

Mit der Set-Anweisung konnen die Systemvariablen verändert werden, die gemäß Tabelle auf Seite 6-2 als "gesetzt vom Benutzer" gekennzeichnet sind.

```
SET(symbol=Ausdruck)
```

Beispiele:

```
SET(J1=J1+1)
```

Zum lokalen Pseudo-Register wird eins addiert.

```
SET(G1=(FL.GT.180000).AND.(G2.GT.2))
```

Dem globalen Pseudoregister wird ein Wert zugewiesen, der abhängig ist von FL und G2.

6.5.3 Drucken von System-Variablen (PRINT)

Mit der PRINT-Anweisung konnen Ausdrücke und die Systemvariablen (siehe Tabelle auf Seite 6-2) im Protokoll ausgedruckt werden. Der Wert wird auf drei verschiedenen Arten ausgedruckt: als eine dezimale Integerzahl, als 22-stelliger Oktalwert oder als ASCII-Zeichenkette.

```
PRINT(Ausdruck)
```

6.5.4 Globale Prozedurbibliotheken (LIBRARY)

Mit der LIBRARY-Anweisung werden Prozedur-Bibliotheken deklariert. Die angegebenen Bibliotheken sind solange global erklärt, bis eine Neudefinition durch eine neue LIBRARY-Anweisung erfolgt. Anmerkung (für CDC-Benutzer): mit der LIBRARY-Anweisung werden nur Prozedur-Bibliotheken global erklärt und nicht, wie bei NOS/BE, Objektbibliotheken. Diese werden beim LDR-Aufruf mit dem Parameter LIB=lb_1:lb_2:.. angegeben.
Prozeduren, die in einer Bibliothek vorhanden sind, können während des Joblaufes aufgerufen werden. Wenn eine Anweisung weder eine COS-Anweisung noch im System ist, werden die angegebenen Prozedur-Bibliotheken nach der Anweisung durchsucht. Die Reihenfolge der Angabe auf der LIBRARY-Anweisung entspricht

der Suchreihenfolge. Wenn die Bibliothekenliste verändert oder erweitert werden soll, ersetzt ein Stern die Angabe der bereits vorhandenen Bibliotheken. Zusätzlich kann mit LIBRARY auch die Suchreihenfolge auf das Protokoll geschrieben werden. Die Voreinstellungsliste enthält die Dateien $PROC.

LIBRARY,DN=dn_1:dn_2:...:dn_{64},V.

Parameter:

DN=dn_i Name der Bibliothek, die mit durchsucht werden soll. Maximal konnen 64 Namen angegeben werden.

V Die im Augenblick gultige Suchreihenfolge wird auf den Protokolldatei geschrieben.

Beispiel:

```
CSP CRAY X-MP/24     KONRAD-ZUSE-ZENTRUM FUER INFORMATIONSTECHNIK BERLIN
CSP                  CRAY OPERATING SYSTEM       COS 1.15  ASSEMBLY DATE....
CSP
CSP
CSP      JOB,JN=HxxxY07.
CSP      ACCOUNT,AC=,APW=,US=,UPW=.
CSP      <DEFINITION>     PROC.
CSP      <DEFINITION>     PROTO,P1=:ABC.
CSP      <DEFINITION>     CFT,I=&P1.
CSP      <DEFINITION>     LDR.
CSP      <DEFINITION>     RETURN.
CSP      <DEFINITION>     &DATA,DFC.
CSP      <DEFINITION>           PROGRAM MTD
CSP      <DEFINITION>           WRITE(*,'(A)')' MESSAGE FROM J C L '
CSP      <DEFINITION>           END
CSP      <DEFINITION>     ENDPROC.
CSP      ACCESS,DN=MYPROC,PDN=PROCTESTLIB.
PDM      PD000 - PDN = PROCTESTLIB       ID =           ED =    2  OWN =
PDM      PD000 - ACCESS  COMPLETE
CSP      LIBRARY,DN=*:MYPROC,V.
CSP      PROTO,P1=DFC.
CSP      CS136 - &DATA CREATED DSN DFC
CSP      CFT,I=DFC.
USER     CF000 - CFT VERSION -   20.02.87 1.15
USER     CF998 - CFT 1.15BF1     EDITION 239.88
USER     CF001 - COMPILE TIME =    0.0047 SECONDS
USER     CF002 -         3 LINES,        3 STATEMENTS
USER     CF003 -    67626 WORDS,    14542 I/O BUFFERS USED
CSP      LDR.
USER     LD000 - BEGIN EXECUTION
CSP      RETURN.
CSP      END OF JOB
```

Aufruf einer Prozedur aus der Bibliothek: Die Prozedur-Bibliothek wird als lokale Datei MYPROC angemeldet. Da die Datei MYPROC nicht als globale Bibliothek vorhanden ist, muß sie mit der LIBRARY-Anweisung dem System bekannt gemacht werden. Die Bibliothek MYPROC wird an die vorhandene Bibliotheks-hierarchie als letzte angehangt. Erst dann kann die Prozedur PROTO aufgerufen werden.

7. CFT Übersetzer-Optionen und -Direktiven

Der einfachste Aufruf des CFT Übersetzers unter Benutzung der Standard-Werte für die beteiligten Dateien lautet:

Beispiel:

```
CFT.
```

(==> 3.3.1 FORTRAN Übersetzer-Aufruf (CFT)).

Arbeitsweise und Leistungen des Übersetzers können einerseits durch Optionen beim Aufruf des Übersetzers und andererseits durch Direktiven in der Programmquelle gesteuert werden.

Übersetzer-Optionen steuern das globale Verhalten für einen Übersetzerlauf wie Ein-/Ausgabe des Übersetzers, Optimierung, insbesondere Vektorisierung der Programme, Hilfen zur Überwachung und Fehlersuche sowie Varianten zur Kompatibilität von Programmen.

Übersetzer-Direktiven passen das Verhalten des Übersetzers lokalen Besonderheiten im Programm an.

Die folgende Beschreibung berücksichtigt die Version CFT 1.15; bei den Parameterwerten ist jeweils die Voreinstellung (default) unterstrichen!

7.1 Übersetzer-Optionen

Es gibt im wesentlichen zwei verschiedene Arten, um Übersetzer-Optionen zu setzen:

1) Explizite Angabe in der Form

   PARAMETERNAME=Wert

Beispiel:

```
L=0                  oder
OPT=NOIFCON:FASTMD
```

2) Ein- bzw. Ausschalten einer Reihe von Optionen in der Form

   ON=Zeichenfolge bzw. **OFF=Zeichenfolge**

Beispiel:

```
ON=AFZ               oder
OFF=P
```

   Dabei bezeichnet jeder auf der rechten Seite des Gleichheitszeichens spezifizierte Buchstabe eine Option.

Ein Übersetzeraufruf kann dann bei Benutzung von Optionen so aussehen:

Beispiel:

```
CFT,L=0,OPT=NOIFCON:FASTMD,ON=AFZ,OFF=P.
```

7.1.1 Optionen zur Steuerung der Ein-Ausgabe

I=idn =$IN	Input: bezeichnet den Namen der Datei, in der die Quelle steht.
L=ldn =$OUT =0	List: bezeichnet den Namen der Datei, in die die Übersetzerliste geschrieben wird. bewirkt, daß keine Liste erzeugt wird.
B=bdn =$BLD =0	Binary: bezeichnet den Namen der Datei, in die der Übersetzer die binaren Lademodule schreibt. bewirkt, daß keine binaren Lademodule erzeugt werden.
C=cdn	Code: bezeichnet den Namen der Datei, in der Pseudo-Code für den CRAY-Assembler (CAL) abgelegt wird. Der erzeugte Text kann nach wenigen Handkorrekturen als Eingabe für CAL dienen. DATA-Anweisungen werden nicht unterstützt.
LOOPMARK<=lmsg>	Wenn LOOPMARK angegeben ist, klammert CFT alle DO-Schleifen in der Ausgabeliste.
=MSGS	Druckt zu allen inneren Schleifen Meldungen, ob sie vektorisiert oder ersetzt wurden, beziehungsweise warum sie nicht vektorisierbar sind.
=NOMSGS	Bringt keine Meldungen bei nicht vektorisierten Schleifen.
AIDS=aids	steuert die Zahl der Meldungen, die ausgegeben werden, wenn bei einer inneren Schleife die Vektorisierung verhindert ist.
=LOOPNONE	keine Meldungen
=LOOPPART	bis zu 3 Meldungen für jede innere Schleife; bis zu 100 Meldungen fur eine Übersetzung
=LOOPALL	alle Meldungen

ON=Zeichenfolge bzw. **OFF=Zeichenfolge**

ON/OFF=B	Block: gibt die Zeilennummer und Speicheradresse jedes Compilationsblocks aus (sinnvoll in Verbindung mit einigen Optionen zur Fehlersuche; es laßt sich so eine Verbindung zwischen Zeilennummer und Speicheradresse herstellen).
ON/OFF=C	Common: druckt Namen und Langen von COMMON-Blöcken.
ON/OFF=D	DO: erzeugt eine Tabelle aller DO-Schleifen.
ON/OFF=G	Generate: druckt den erzeugten Code für jede Programmeinheit (==> 7.2.1 Direktiven CODE und NOCODE).
ON/OFF=H	Head line: bewirkt, daß nur die erste Zeile jeder Programmeinheit sowie Fehlermeldungen gedruckt werden; H überschreibt L=0 (sinnvoll bei der Übersetzung großer Programme, von denen schon eine Übersetzerliste existiert).
ON/OFF=I	Internal labels: gibt die vom Übersetzer generierten Anweisungsmarken aus. Ahnlich wie ON=B ist diese Option in Verbindung mit einigen Optionen zur Fehlersuche sinnvoll, da sie einen Bezug zwischen vom Übersetzer generierten Marken und den zugehorigen Adressen herstellt.

ON/OFF=L — List directives: ermoglicht die Erkennung von Direktiven zur Steuerung der Übersetzerlisten (==> 7.2.1).

ON/OFF=S — Source: druckt die FORTRAN-Quelle.

ON/OFF=T — Symbol table: gibt nach jeder Programmeinheit eine Symboltabelle aus (bei großen Programmen sehr umfangreich).

ON/OFF=X — Cross reference: erzeugt eine Kreuzreferenz-Tabelle aller Symbole einer Programmeinheit; überschreibt T.

7.1.2 Optionen zur Vektorisierung und Optimierung

OPT=option — Folgende Werte für 'option' sind erlaubt (werden mehrere Werte spezifiziert, so sind sie durch Doppelpunkte zu trennen):

=NOZEROINC
=ZEROINC

Mit diesen beiden Optionen wird CFT mitgeteilt, ob Constant Increment Variables (CIV ==> 9.3.2) mit einer Variablen vom Wert Null inkrementiert werden konnen oder nicht.
Bei ZEROINC geht der Übersetzer davon aus, daß in der Form

```
CIV = CIV + variable
```

die Variable auch den Wert Null haben kann und daher nicht vektorisiert werden darf.

=NOIFCON
=PARTIALIFCON
=FULLIFCON

Mit diesen Optionen wird die Behandlung von IF-Anweisungen gesteuert (==> 9.3.4 und 7.2.2 Direktiven NOIFCON und RESUMEIFCON).
Bei NOIFCON werden IF-Anweisungen der Form

```
IF (Bedingung) Zuweisung
```

nur vektorisiert, falls der Übersetzer diese durch die Intrinsic Funktionen MAX bzw. MIN ersetzen kann.

Bei PARTIALIFCON vektorisiert CFT alle IF-Anweisungen dieser Form, sofern die Zuweisung nicht Divisionen oder Aufrufe von Standardfunktionen enthält.

Der Übersetzer vektorisiert bei FULLIFCON auch solche IF-Anweisungen der obigen Form, die in der Zuweisung Divisionen oder Standardfunktionen enthalten.

=SLOWMD
=FASTMD

Diese beiden Optionen wählen die Genauigkeit und die Geschwindigkeit der INTEGER-Multiplikation und -Division.
Der Übersetzer benutzt bei FASTMD die schnelleren Operationen mit 46 Bit. Ein Überlauf wird hierbei nicht erkannt.
Der Übersetzer führt bei SLOWMD alle INTEGER-Operationen mit der vollen 64-Bit Genauigkeit aus.

=SAFEDOREP
=FULLDOREP
=NODOREP

Einzeilige DO-Schleifen werden nur dann durch eine Routine aus $SCILIB ersetzt, wenn keine Abhängigkeiten (==> 9.3.3) und kein EQUIVALENCE vorliegen.
Wenn FULLDOREP angegeben ist, werden beim Ersetzen dieser DO-Schleifen Abhangigkeiten und EQUIVALENCE nicht berücksichtigt. (==> 7.2.2 Direktiven NODOREP und RESUMEDOREP).
NODOREP verhindert jede automatische Ersetzung durch Routinen aus $SCILIB.

=SAFEIF =UNSAFEIF	Zur Optimierung konnen Befehle vertauscht werden. Bei SAFEIF werden Befehle nicht vor Sprunganweisungen verschoben.
=INVMOV =NOINVMOV	Mit diesen beiden Optionen wird festgelegt, ob invarianter Code in einer Schleife vor die Schleife gezogen werden soll.
=BL =NOBL	Bottom Load: In skalaren Schleifen konnen die nachsten Operanden schon geholt werden, bevor das Endekriterium abgefragt wird. Durch NOBL wird 'Bottom Load' und der dabei mogliche 'Out Of Range'-Fehler verhindert.
=NOBTREG =BTREG	B- und T-Register: Alle vom Benutzer definierten Variablen liegen im Speicher. B- und T-Register werden nur fur Hilfsvariable und Zwischenergebnisse verwendet. Wenn BTREG angegeben ist, konnen bis zu 24 lokale Variable eines Unterprogramms, die nicht in SAVE-, DATA-, COMMON- oder NAMELIST-Anweisungen vorkommen, in T-Registern abgelegt werden. Sie werden am Anfang nicht vorbesetzt und sind nach RETURN oder END undefiniert.
=CVL =NOCVL	Conditional Vector Loops: Fur Schleifen mit ungewissen Abhangigkeiten (==> 9.3.3) wird vektorieller und skalarer Code erzeugt. Zur Laufzeit entscheidet sich, welche Version zu durchlaufen ist. NOCVL legt fest, daß fur Schleifen mit ungewissen Abhangigkeiten nur skalarer Code erzeugt wird.
=KEEPTEMP =KILLTEMP	Bei KEEPTEMP haben Variable,die als temporare Vektoren (==> 9.3.2) verwendet werden, nach Durchlauf der Vektorschleife den korrekten Wert. Bei KILLTEMP sind sie nach Durchlauf der Vektorschleife undefiniert.
MAXBLOCK=mb =4000	CFT kann Blocke von Anweisungen mit einer Lange bis zu "mb" Worten optimieren und vektorisieren. Werte großer als die Voreinstellung 4000 konnen die Optimierung verbessern, aber auch zum Abbruch des Übersetzers führen.
=1	verhindert jede Optimierung.
UNROLL=r =3 =0	Innere DO-Schleifen mit einer konstanten Zahl von bis zu "r" Durchlaufen werden abgerollt (==> 9.3.4). "r" ist maximal 9. verhindert das automatische Abrollen innerer Schleifen.

ON=Zeichenfolge bzw. **OFF=Zeichenfolge**

ON/OFF=V	Vectorize: Schaltet die Vektorisierung von DO-Schleifen ein bzw. aus. Das Ausschalten dieser Option kann in Einzelfallen sinnvoll sein, um den Vektorisierungsgrad eines Programms zu ermitteln.

7.1.3 Optionen zur Überwachung und Fehlersuche

E=n — Error level: gibt das hochste Niveau der Meldungen an, die CFT unterdrucken soll.

- =0 keine Unterdrückung
- =1 Kommentare zur Effizienz des Programms werden unterdrückt.
- =2 Zusatzlich werden Anmerkungen über mogliche Schwierigkeiten mit anderen Übersetzern (z.B. nicht ANSI) unterdruckt.
- =3 Zusatzlich werden Hinweise auf mogliche Fehler (z.B. eine Anweisung, die nicht durchlaufen wird) unterdruckt.
- =4 Zusatzlich werden Warnungen vor wahrscheinlichen Fehlern (z.B. Benutzung eines Feldes mit zu wenig Indizes) unterdruckt.

Fehler (Niveau 5) konnen nicht unterdruckt werden.

EDN=edn — Error dataset name: bezeichnet den Namen der Datei, in die Fehlermeldungen mit hoherem Niveau als "E=n" zusatzlich zu "ldn" (==> 7.1.1 Option L) ausgegeben werden.

DEBUG — erzeugt eine Numerierung für die ausführbaren FORTRAN-Anweisungen und schreibt sie in die Symbol-Tabelle fur den Symbolischen Interaktiven Debugger (SID).
DEBUG verhindert Vektorisierung und Optimierung.

ON=Zeichenfolge bzw. **OFF=Zeichenfolge**

ON/OFF=A — Abend: Der Job wird nach der Übersetzung abgebrochen, falls ein fataler Fehler bei der Übersetzung aufgetreten ist. Ist OFF=A gesetzt (Standard), so werden nachfolgende Verarbeitungsschritte (z.B. die Programmausfuhrung) auch dann ausgeführt, wenn schwerwiegende Fehler bei der Übersetzung aufgetreten sind.

ON/OFF=F — Flowtrace: Schaltet die Messung des Zeitverbrauchs aufgeschlusselt nach den einzelnen Programmmoduln ein bzw. aus. Dies ist eine fur Optimierungszwecke sehr nutzliche Option, um die rechenzeitintensiven Programmteile zu ermitteln. Sie sollte jedoch nur für einzelne Programmlaufe eingesetzt werden, da sie im allgemeinen zu einem erheblichen zusatzlichen Zeitbedarf fuhrt (in Abhangigkeit von der Anzahl der Unterprogrammaufrufe wahrend der Laufzeit).
Anmerkung: Voraussetzung für die Verwendung der Flowtrace-Option ist das Vorhandensein einer PROGRAM-Anweisung im FORTRAN-Hauptprogramm. Zur Ausgabe der erzeugten Information benotigt man das Kommando FLOWTRACE (==> 8.4.3). Mit dem FLOWTRACE-Kommando werden die erzeugten Tabellen ausgedruckt.

ON/OFF=O — Array bounds checking: Schaltet die Überprufung von Feldindizes auf Bereichsuberschreitung ein bzw. aus. Diese Option verhindert die Vektorisierung jeglicher Schleifen und verbraucht daher sehr viel CPU-Zeit. Sie sollte nur in Einzelfallen benutzt werden.
Anmerkung: Felder vom Typ CHARACTER, Feldelemente als aktuelle Parameter oder innerhalb einer Ein-/Ausgabeliste werden nicht überprüft.

ON/OFF=Q — bricht die Übersetzung nach 100 Fehlern ab.

ON/OFF=Z — Symbol table: Erzeugt eine DEBUG-Symboltabelle zur Fehlersuche. Diese Option verbraucht während der Ausführung weder nennenswert CPU-Zeit noch Speicherplatz; sie kann daher auch bei Produktionslaufen eingeschaltet bleiben.

7.1.4 Optionen zur Programmportabilität

ANSI — Alle Anweisungen, die nicht dem ANSI-Standard entsprechen, werden gemeldet.

CPU=cputyp:cpuchar kennzeichnet den Rechnertyp und mögliche Eigenschaften der Zielmaschine, auf der der erzeugte Code laufen soll.
Wenn die Option nicht gesetzt ist, wird Code für die Maschine erzeugt, auf der CFT läuft, beziehungsweise für die, die durch die COS-Anweisung TARGET festgelegt wurde.
Ohne Angabe von CPU nimmt CFT die folgenden Eigenschaften der im ZIB installierten Anlage an:

CRAY-XMP	Maschinentyp
IBUFSIZE=32	Instruction buffer size: Größe des Befehlspuffers
MEMSPEED=14	Memory speed: Geschwindigkeit des Speichers in Takten
NOEMA	Extended memory addressing: keine erweiterte Speicheradressierung
NOCIGS	Compressed index / gather scatter: Keine Hardware für komprimierte Indizes und Gather-, Scatter-Operationen (==> 9.3.2)
VPOP	Vector population: Vektoreinheit zum Zählen gesetzter Bits vorhanden
NOAVL	Additional vector logical unit: Nur eine Einheit für logische Vektoroperationen
NOVRECUR	Vector recursion: Keine Vektorrekursion

ON=Zeichenfolge bzw. **OFF=Zeichenfolge**

ON/OFF=J — Bewirkt, daß alle DO-Schleifen abweichend vom Standard mindestens einmal durchlaufen werden (sinnvoll bei der Umstellung von FORTRAN-IV auf FORTRAN 77).

ON/OFF=M — Machine characteristics: Gibt die Eigenschaften der eingestellten Zielmaschine aus.

ON/OFF=P — Ermöglicht oder verhindert DOUBLE PRECISION-Arithmetik.
OFF=P bewirkt folgendes:

- Alle DOUBLE PRECISION-Deklarationen werden wie REAL-Deklarationen behandelt.
- Doppelt genaue Funktionen werden in die entsprechenden einfach genauen Versionen umgewandelt.
- Doppelt genaue Konstanten werden in einfach genaue konvertiert.
- D-Formate werden in E-Formate umgewandelt.

Diese Option ist für Programme sinnvoll, die an Anlagen entstanden sind, an denen wegen geringerer Maschinengenauigkeit mit DOUBLE PRECISION gerechnet wurde. An der CRAY Anlage ist DOUBLE PRECISION wegen der hohen Genauigkeit (64 Bit Worte für einfache Genauigkeit) im allgemeinen nicht nötig. Wird es trotzdem eingesetzt, führt dies zu erheblicher Verlangsamung des Programmablaufs.

7.1.5 Sonstige Optionen

TRUNC=nn
=0 — Zahl der Bits zwischen 0 und 47, die bei Gleitpunktoperationen gerundet werden sollen.

INT=64
=24 — Lange von INTEGER-Zahlen in Bits.
Anmerkung: Ein Überlauf über $2^{23}-1$ wird nicht erkannt.

ALLOC=allocation — bestimmt die Art, wie Variable im Speicher abzulegen sind.
=STATIC — Allen Variablen wird Speicher fest zugewiesen.
=STACK — Konstanten und solchen Variablen, die in DATA-, SAVE- oder COMMON-Anweisungen vorkommen, wird Speicher fest zugewiesen. Alle anderen Variablen werden dynamisch in einem Keller ("stack") verwaltet.

INDEF — Besetzt den Kellerbereich so vor, daß Gleitpunktoperationen mit undefinierten REAL-Variablen und Feldzugriffe mit undefinierten INTEGER-Variablen zu Fehlermeldungen führen.

SAVEALL — wirkt wie eine SAVE-Anweisung mit leerer Liste, d.h. allen vom Benutzer definierten Variablen wird Speicher fest zugewiesen.
OPT=BTREG wird überlagert (==> 7.1.2).
Im Zusammenhang mit ALLOC=STACK werden nur vom Übersetzer erzeugte Variable auf den "stack" gelegt.

ON=Zeichenfolge bzw. **OFF=Zeichenfolge**

ON/OFF=**R** — Round: Multiplikationsergebnisse werden gerundet.

ON/OFF=**U** — Bei OFF=U werden INTEGER*2-Variable in 64 Bit abgelegt, sonst in 24 Bit.

ON/OFF=**W** — Reelle, einfach genaue Gleitpunktoperationen können als Unterprogrammsprunge in externe Routinen ausgeführt werden.

7.2 Übersetzer-Direktiven

Übersetzer-Direktiven erlauben es, einige Übersetzer-Optionen für bestimmte Programmabschnitte ein- bzw. auszuschalten und eröffnen außerdem zusätzliche Möglichkeiten. Sie werden in der Form

CDIR$ value

ins FORTRAN Quellprogramm eingestreut. Sie beginnen stets ab Spalte 1 und werden daher von anderen Übersetzern wie Kommentare behandelt; sie stören die Portabilität der Programme nicht.

Alle Übersetzer-Direktiven werden nur erkannt, wenn beim CFT-Aufruf ON=E (Standard) eingestellt ist (==> 7.1.5). Weiter Einschränkungen für Gruppen von Direktiven sind möglich (siehe unten).

Einige wichtige Übersetzer-Direktiven sollen hier vorgestellt werden. Man beachte auch die Erklärungen, die in Zusammenhang mit der Erläuterung von Optimierungs- und Vektorisierungsmöglichkeiten im Abschnitt "9. Einführung in die Optimierung der Rechenzeit" gegeben werden.

In den folgenden Abschnitten sind mögliche Werte für '**value**' angegeben.

Beispiele:

```
CDIR$ NOLIST
CDIR$ NOCODE,IVDEP
```

7.2.1 Direktiven zur Steuerung der Ausgabeliste

Diese Direktiven werden nur erkannt, wenn beim CFT-Aufruf ON=L (Voreinstellung) gesetzt ist (==> 7.1.1).

EJECT — beim Auflisten der Quelle wird eine neue Seite begonnen.

NOLIST
LIST — NOLIST unterdruckt die Ausgabeliste für den folgenden Teil der Programmquelle. Wirkt über die folgende END-Anweisung hinaus. Überschreibt die Übersetzer-Optionen zur Steuerung der Ausgabeliste.
LIST schaltet die Erzeugung der Ausgabeliste wieder ein.

NOCODE
CODE — NOCODE unterdruckt die Liste des generierten Codes vom Anfang des nachsten Optimierungsblocks an.
CODE erzeugt vom laufenden Optimierungsblock an die Liste des generierten Codes bis zu einer NOCODE-Direktive.

7.2.2 Direktiven zur Vektorisierung und Optimierung

Die Direktiven zur Vektorisierung setzen voraus, daß ON=V (Standard) als Übersetzer-Option gewählt wurde (==> 7.1.2).

NOVECTOR <=n>
VECTOR — Mit Hilfe dieser Direktive wird festgelegt, ob eine Vektorisierung von DO-Schleifen verhindert werden soll.
Ohne die Angabe n wird eine Vektorisierung in den auf die Direktive folgenden Programmteilen generell verhindert.
Bei Angabe von n, einer ganzen Zahl zwischen 0 und 64, bezieht sich die Verhinderung der Vektorisierung nur auf Schleifen, deren Wiederholungszahl I ("iteration count") für den Übersetzer erkennbar eine Konstante ist und für die I<=n gilt.
Wird die Direktive nicht verwendet, ist n auf 1 gesetzt.
VECTOR hebt NOVECTOR<=n> wieder auf, d.h. innere DO-Schleifen mit Wiederholungszahl großer als 1 werden möglichst vektoriell abgearbeitet (==> 7.1.2 Option UNROLL).

NORECURRENCE <=n>
=14 — Diese Direktive legt fest, ob Schleifen, die eine akkumulierende Zuweisung enthalten ("recurrence relation"), skalar oder vektoriell abgearbeitet werden sollen.
Akkumulierende Zuweisungen der Form

```
S = S + e   oder   S = S * e
```

kann CFT vektorisieren, wenn
S eine skalare Variable und
e ein vektorisierbarer Ausdruck ist.

Ohne die Angabe von "n" wird eine Vektorisierung aller folgenden akkumulierenden Schleifen verhindert.
Bei Angabe von "n" werden nur die akkumulierenden Schleifen skalar abgearbeitet, deren Wiederholungszahl I ("iteration count") für den Übersetzer erkennbar eine Konstante ist, für die I <= n gilt.

IVDEP	Ignore vector dependency: In der folgenden inneren Schleife werden Abhangigkeiten von Feldindizes, die einer Vektorisierung im Wege stehen konnen, nicht beachtet (==> 9.3.3 Abhangigkeiten). *Anmerkung:* Die unbedachte Verwendung von IVDEP kann zu falschen Ergebnissen fuhren (==> 9.3.4).
NEXTSCALAR	Mit dieser Direktive wird die Vektorisierung fur die folgende DO-Schleife abgeschaltet.
SHORTLOOP	Die Verwendung dieser Direktive kann zu einer kürzeren Ausfuhrungszeit der nachsten inneren Schleife fuhren, falls diese Schleife vektorisiert wird. Sie darf nur dann verwendet werden, wenn bekannt ist, daß diese Schleife mindestens 1 und hochstens 64 mal durchlaufen wird. Andernfalls sind die Ergebnisse nicht definiert.
FASTMD SLOWMD	Diese Direktiven entscheiden uber INTEGER-Multiplikation und -Division im laufenden Optimierungsblock (==> 7.1.2 Optionen OPT=FASTMD und OPT=SLOWMD). Bei FASTMD haben die Ergebnisse nur eine Genauigkeit von 46 Bit. Uberlauf wird nicht festgestellt. Bei SLOWMD wird die langsamere Ganzwortarithmetik (64 Bit) verwendet.
UNSAFEIF SAFEIF	Die Direktiven UNSAFEIF und SAFEIF ermoglichen bzw. verhindern, daß in einem Block einzelne Befehle (außer Speichern und Division) uber den Sprung einer IF-Anweisung hinaus verschoben werden konnen. Beide Direktiven uberlagern nur fur den Block, in dem sie stehen, die Einstellung beim CFT-Aufruf (==> 7.1.2).
BL NOBL	Bottom load: Die Direktiven BL und NOBL ermoglichen bzw. verhindern, daß Operanden fur den nachsten Durchlauf einer skalaren Schleife schon vor Abfrage des Endekriteriums geladen werden konnen. Beide Direktiven uberlagern nur fur den Block, in dem sie stehen, die Einstellung beim CFT-Aufruf (==> 7.1.2).
ALIGN	Die Direktive ALIGN bestimmt, daß die folgende Anweisung an den Anfang des Befehlspuffers gelegt wird, wenn diese eine SUBROUTINE-, PROGRAM-, FUNCTION-, ENTRY- oder DO-Anweisung oder ein Sprungziel ist.
NOIFCON RESUMEIFCON	IF conditions: Mit der Direktive NOIFCON kann die Optimierung von Anweisungen der Form IF (Bedingung) Zuweisung abgeschaltet und mit RESUMEIFCON auf die beim CFT-Aufruf angegebene Option zuruckgeschaltet werden (==> 7.1.2 Optionen OPT= NOIFCON, OPT=PARTIALIFCON und OPT=FULLIFCON).
NODOREP RESUMEDOREP	DO loop replacement: Die Direktive NODOREP verhindert die Ersetzung einzeiliger DO-Schleifen durch eine Routine aus $SCILIB. RESUMEDOREP kehrt zu der Einstellung beim CFT-Aufruf zurück (==> 7.1.2 Optionen OPT=SAFEDOREP und OPT=FULLDOREP).

NOCVL
CVL
: Conditional vector loops: Mit diesen beiden Direktiven konnen die gleichnamigen Optionen beim CFT-Aufruf (==> 7.1.2) überschrieben werden.
CVL heißt, daß fur Schleifen mit ungewissen Abhangigkeiten (==> 9.3.3) vektorieller und skalarer Code erzeugt wird. Zur Laufzeit entscheidet sich, welche Version zu durchlaufen ist.
Nach NOCVL wird fur Schleifen mit ungewissen Abhangigkeiten nur skalarer Code erzeugt.

ROLL
: Nach ROLL werden keine DO-Schleifen abgerollt.

UNROLL
: Nach UNROLL konnen innere DO-Schleifen mit einer konstanten Zahl von Durchlaufen gemaß der Angabe beim CFT-Aufruf abgerollt werden (==> 7.1.2 Option UNROLL).

7.2.3 Direktiven zur Überwachung und Fehlersuche

Voraussetzung für die Benutzung der Laufzeituberwachung durch FLOWTRACE ist, daß die Ubersetzer-Option F auf ON gesetzt ist; für das Uberwachen der Feldgrenzen (Array Bound Checking) ist die Ubersetzer-Option O Voraussetzung.

NOFLOW
FLOW
: Mit NOFLOW kann die FLOWTRACE-Option (F) des Ubersetzers fur einzelne Unterprogramme abgeschaltet werden. CDIR$ NOFLOW muß nach einer END- und vor einer SUBROUTINE- oder FUNCTION-Anweisung stehen. Diese Direktive ist sinnvoll fur kleine, haufig gerufene Unterprogramme, um den zusatzlichen Rechenzeitbedarf der FLOWTRACE-Option zu begrenzen.
FLOW hebt NOFLOW wieder auf.

BOUNDS
: Fur alle Felder wird bei jedem Zugriff gepruft, ob die Feldgrenzen eingehalten werden (sehr rechenzeitaufwendig, keine Vektorisierung).

BOUNDS()
: Abschalten der Uberprufung von Feldgrenzen. Gilt nur innerhalb einer Programmeinheit.

BOUNDS(a,b,c)
: Einschranken der Uberprufung von Feldgrenzen auf die Felder der Liste (a,b,c). Gilt nur innerhalb einer Programmeinheit.

7.2.4 Sonstige Direktiven

INT24 <v, ..>
INT64 <v, ..>
: Diese Direktiven bestimmen die Lange von INTEGER-Zahlen in Bits entweder fur alle implizit vereinbarten Variablen, die mit den Buchstaben I bis N anfangen, oder für die angegebenen Variablen "v" (==> 7.1.5 Option INT).
Anmerkung: Bei INT24 wird ein Uberlauf uber $2^{23}-1$ nicht erkannt.

8. Fehlersuche in FORTRAN-Programmen

Es kann keine einheitliche Strategie zur Fehlersuche empfohlen werden; hier sollen nur verschiedene Möglichkeiten vorgestellt werden, durch die man Informationen und Daten erhält, die das Aufspüren eines Fehlers ermöglichen. Diese Informationen können zu verschiedenen Zeitpunkten ausgegeben werden:

* nach Ende oder Abbruch der Übersetzung
* nach Ende oder Abbruch des Programmlaufes
* zur Programmlaufzeit

Die Auswahl der jeweils gewünschten Art der Fehlerverfolgung erfolgt nicht nur durch entsprechende Kommandos, sondern auch durch die vorbereitende Spezifikation von Anweisungen in anderen Kommandos bzw. im Programm selbst.

Beispiel: Erstellen eines DUMPS:

CFT,ON=Z.	Vorbereitung, Compiler legt Symboltabellen an.
LDR,SET=INDEF.	Vorbereitung, Lader besetzt Variable mit erkennbar falschen werten vor (Voreinstellung im ZIB).
EXIT.	Vorbereitung, die folgenden Kommandos werden nur im Fehlerfall ausgeführt.
DUMPJOB.	Erzeugt die Datei $DUMP mit einem Speicherabzug.
DEBUG.	Wertet $DUMP aus und legt eine lesbare Form des DEBUG in eine Datei ab.

Zusätzlich zu den im Kapitel 7. beschriebenen Möglichkeiten der Fehlersuche mit Compiler-Anweisungen sollen hier Kommandos gezeigt werden, die zusammen mit dem Compiler bzw. nach dem Übersetzungslauf wirken (==> 7.1.1, 7.1.3, 7.2.1, 7.2.3).
Eine ausführliche Darstellung der verschiedenen Befehle zur Fehlersuche mit Beispielen und den jeweils erzeugten Ausgabelisten enthält die CRAY-Schrift "Symbolic Debugging Package Reference Manual SR-0112" (==> 1.7.1). Darin wird auch auf verschiedene neue Werkzeuge hingewiesen, die sich noch in der Entwicklung befinden (Dynamic Runtime Debugger, Dynamic Dump Analyzer).

Voraussetzung zum Erzeugen der Dumps ist, daß eine der Compileroptionen ON=Z oder DEBUG angegeben wurde. Beide bewirken, daß der Compiler symbolische Tabellen anlegt, d.h. die vom Benutzer vergebenen Namen bleiben erhalten und können im Dump den Speicherinhalten zugeordnet werden. Die Wirkung der Kommandos ist unterschiedlich: unter ON=Z läuft das Programm in der Weise ab, die COS als optimal festgelegt hat; unter DEBUG wird jede Optimierung und Vektorisierung unterdrückt. Das Programm läuft damit so ab, wie es vom Benutzer geschrieben wurde. Mit dem Dump läßt sich dann der Programmablauf schrittweise nachvollziehen, besonders in den DO-Schleifen. Leider kann es in Ausnahmefällen geschehen, daß ein Programm nur unter einem Modus (optimiert oder schrittweise) fehlerhaft abläuft.

8.1 Lader-Optionen zur Fehlersuche

Folgende Parameter werden beim Lader LDR bzw. bei dem vorzuziehenden Segment-Lader SEGLDR zur Fehlersuche verwendet (==> 3.3.3):

Beim Lader LDR wird mit den Anweisungen NX und SET die Fehlerbehandlung gesteuert:

```
LDR<,DN=dn,SET=val,LIB=ldn,NOLIB=ldn,AB=adn,NX,^
     MAP=op,L=ldn,NA,USA,E=n,NOECHO>.
```

Parameter zur Fehlerbehandlung (die Voreinstellung ist unterstrichen):

NX	No Execution: Das Programm wird nach dem Laden nicht ausgeführt. Dies ermöglicht, offene Bezüge festzustellen (z.B. vergessene Labels)
SET=val	dient der Vorbesetzung aller Variablen im Speicher (mit Ausnahme derjenigen, die bei der Übersetzung vorbesetzt wurden).
=ZERO	Vorbesetzung mit binären Nullen.
=INDEF	Hier erfolgt die Vorbesetzung mit zu großen Gleitpunktzahlen (out of range). Wird auf diese zugegriffen, erfolgt ein Fehler-Abbruch (floating point error). Im symbolischen Dump (==> 8.3) erscheint als Wert der so vorbesetzten Variablen ein 'R'.

Beim Segment-Lader SEGLDR wird mit den Anweisungen (Direktiven) MLEVEL und USX die Fehlerbehandlung gesteuert. Sie werden der Option CMD als Zeichenkette (dirstring) übergeben:

```
SEGLDR<,I=idn,L=ldn,GO,DW=dw,CMD='MLEVEL=m,USX=u,PRESET=p'>.
```

Anweisungen zur Fehlerbehandlung (die Voreinstellung ist unterstrichen):

MLEVEL=ERROR	Nur Fehler-Meldungen werden ausgegeben; SEGLDR beendet sich augenblicklich. Es wird kein ausführbares Programm erstellt.
=WARNING	Fehler- und Warnungs-Meldungen werden ausgegeben Es wird kein ausführbares Programm erstellt; die Verarbeitung wird aber weitergeführt, so daß weitere Meldungen ausgegeben werden können.
=CAUTION	Ausgabe von Fehler-, Warnungs-, und Achtung-Meldungen. Ein ausführbares Programm wird trotz aufgetretenen Fehlers erzeugt.
=NOTE	Ausgabe von Fehler-, Warnungs-, Achtung- und Hinweis-Meldungen. SEGLDR wurde möglicherweise nicht korrekt oder ineffektiv benutzt; keine Auswirkung auf die Ausführung.
=COMMENT	Alle Meldungen werden ausgegeben. Keine Auswirkung auf die Ausführung.
USX=WARNING	SEGLDR kennzeichnet unbefriedigte Externbezüge (unsatisfied external symbols) und erzeugt kein ausführbares Programm.
=CAUTION	SEGLDR kennzeichnet unbefriedigte Externbezüge (unsatisfied external symbols) und erzeugt ein ausführbares Programm.
=IGNORE	SEGLDR erzeugt trotz unbefriedigte Externbezüge (unsatisfied external symbols) ohne Warnung ein ausführbares Programm.
PRESET=ONES	Vorbelegung mit -1
=ZEROS	Vorbelegung mit 0
=INDEF	Vorbelegung mit einem Wert derart, daß dessen Verwendung in einer Gleitkomma-Operation zur Anzeige eines Gleitkomma-Fehlers führt.
=-INDEF	Vorbelegung wie indef, aber negativ.
=value	eine 16-bit Oktalzahl kann vom Benutzer vorgegeben werden.
GO	bewirkt die sofortige Programmausführung. Wird GO nicht angegeben, so entspricht das der Angabe von NX (No Execution) beim LDR.

8.2 Kommando EXIT

Aufgrund der Bearbeitungs-Strategie für Kommandofolgen kommt dem Kommando EXIT im Fehlerfall besondere Bedeutung zu: tritt bei der Bearbeitung eines Kommandos ein Fehler auf, werden alle nachfolgenden Kommandos bis zum /EOF oder bis zum nächsten EXIT übersprungen, d.h. Kommandos zur Fehleraufbereitung, die auf EXIT folgen, werden nur im Fehlerfall ausgeführt.

Die EXIT-Anweisung definiert also die Position in den Steuerkarten, an der die Jobbearbeitung beim Auftreten eines Fehlers fortgesetzt wird. Erfolgt ein Jobabbruch wegen Überschreitung der angeforderten Rechenzeit, so stehen dem Job noch drei Sekunden für die Abarbeitung der nach EXIT angegebenen Anweisungen zur Verfügung. Tritt kein Jobabbruch auf, so bezeichnet EXIT das normale Ende der Jobbearbeitung.

```
EXIT.
```

Beispiel:

```
JOB.
ACCOUNT,AC=xxxx,APW=UNKNACKBAR.
CFT,ON=AZ.
LDR.                    besser: SEGLDR,GO.
EXIT.
DUMPJOB.
DEBUG.
.
.  JCL Anweisungen
.
/EOF
.  FORTRAN Programm
.
```

Die hinter EXIT angegebenen Anweisungen zur Erzeugung von Hilfen zur Fehlersuche ("symbolischer Debug") werden nur dann ausgeführt, wenn die Übersetzung des FORTRAN-Programms mit CFT oder die Ausführung des Programms mit LDR/SEGLDR zu einem Jobabbruch führen.

Mit dem Aufruf eines nicht vorhandenen Programms, z.B. namens BLUFF, kann man das EXIT,U. bei CDC simulieren:

```
BLUFF.
EXIT.
```

hier werden die Steuerkarten nach dem EXIT. in jedem Fall ausgeführt - entweder bei Fehlerabbruch vor BLUFF. oder aber deswegen, weil kein Programm BLUFF existiert.

8.3 Kommando DUMPJOB

DUMPJOB erzeugt an der Stelle, an der es aufgerufen wurde, die Datei $DUMP. Diese Datei erhält ein Abbild des Hauptspeicherbereiches und der Job-Table-Area (JTA) des Jobs. Diese Zwischendatei kann von anderen Programmen wie DUMP und DEBUG zu einem formatierten Dump bzw. einem symbolischen Dump weiterverarbeitet werden.

Die DUMPJOB-Anweisung wird nicht ausgeführt, wenn das Programm von einer Datei geladen wurde, die mit "Execute-Only" angelegt wurde. Es ist also nicht möglich, einen Speicherabzug eines Programmes zu erhalten, das nur ausgeführt werden darf.

```
DUMPJOB.
```

8.4 Erzeugen eines DUMPs nach Fehlerabbruch

8.4.1 Symbolischer DUMP (DEBUG)

Bei der Verwendung der CFT-Compiler-Option ON=Z werden vom Compiler spezielle Symboltabellen zur Fehlersuche in FORTRAN Programmen erzeugt. Diese Tabellen enthalten Namen von Variablen, Feldern, Common-Blöcken usw. sowie den Bezug zwischen diesen Namen und den zugehörigen Speicheradressen. Abbruchadressen können so in symbolischer Form ausgegeben werden (bezogen auf FORTRAN-Labels: z.B. Abbruch zwischen Label 10A und 10B; 10A bezeichnet hier den Anfang einer DO-Schleife mit Label 10, 10B bezeichnet deren Ende). Zur Ausgabe dieses symbolischen Dumps im Fehlerfall sind einige zusätzliche Steuersprachen-Kommandos im CRAY-Job erforderlich:

```
  :
CFT,ON=Z.
LDR.                    (besser: SEGLDR,GO)
  :
  :                     Programm, das mit Fehler abbricht
  :
EXIT.
DUMPJOB.
DEBUG.
  :
```

Das Kommando arbeitet nur einwandfrei, wenn zum Laden der Segment-Lader SEGLDR aufgerufen wurde. Diese Form von DEBUG gilt ab Version 1.15 des COS (die Namen einzelner Optionen sind geändert gegenüber älteren Versionen).

```
DEBUG<,S=sdn,L=ldn,DUMP=ddn,CALLS=n,SYMS=sym,NOTSYMS=nsym,MAXDIM=dim,^
      BLOCKS=blk,NOTBLKS=nblk,PAGES=np,COMMENTS='string'>.
```

Parameter: Werden mehrere Werte spezifiziert, sind die Werte durch Doppelpunkte voneinander zu trennen. (die Voreinstellung (default) ist unterstrichen).

S=sdn
=$DEBUG — Name der Datei, die die Symbol-Tabelle enthält. Diese Datei wird bei der Ausführung des Programmes, das zum Abbruch führte, erstellt.

L=ldn
=$OUT — Name der Datei, auf die der symbolische Dump geschrieben wird.

DUMP=ddn
=$DUMP — Name der Datei, die den von DUMPJOB erzeugten Hauptspeicherabzug enthält.

CALLS=n
=50 — gibt die Tiefe der Schachtelung der Routinen an, die vom symbolischen Dump zurückverfolgt und ausgewertet wird. Vorausgesetzt wird, daß für die Routinen Informationen in den symbolischen Tabellen vorliegen.

SYMS=sym — Es werden die einzelnen Symbolnamen oder Gruppen von Namen

angegeben, die in den Dump mit aufgenommen werden. Ausdrücklich angegeben werden können zwanzig Namen. Voreinstellung: alle Symbole werden gedumpt.

NOTSYMS=nsym — Mit dem SYMS entgegengesetzt wirkenden Parameter NOTSYMS können einzelne Symbolnamen oder Gruppen von Namen vom DUMP ausgeschlossen werden. Der NOTSYMS-Parameter hat Vorrang vor dem SYM-Parameter. Ausdrücklich angegeben werden können zwanzig Namen.

MAXDIM=dim — Gibt die maximale Anzahl von Feldelementen in jeder Dimension an, die bei FORTRAN-Feldern gedruckt werden sollen.
=20:5:2:1:1:1:1

Beispiel: MAXDIM=3:2:50

BLOCKS=blk — blk gibt die Namen der Common-Blöcke an, die gedruckt werden sollen. Ausdrücklich angegeben werden können zwanzig Namen. Wird BLOCKS ohne Angabe eines Namens spezifiziert, so werden alle Common-Blöcke gedruckt.
Voreinstellung: es werden keine Common-Blöcke gedruckt.

NOTBLKS=nblk — Im Gegensatz zu BLOCKS können mit NOTBLKS Common-Blöcke vom Dump ausgeschlossen werden. Ausdrücklich angegeben werden können zwanzig Namen.
Voreinstellung: es werden keine Common-Blöcke ausgeschlossen. NOTBLKS hat Vorrang vor BLOCKS.

Werden beide Parameter nicht angegeben, werden alle Blöcke gedruckt!

RPTBLOCKS — bewirkt, daß die mit den beiden vorgenannten Parametern zur Ausgabe vorgesehenen COMMON-Blöcke für jedes Unterprogramm gesondert ausgegeben werden.

PAGES=np — Die maximale Seitenanzahl eines Dumps wird angegeben.
=70

Bei den Parametern SYMS, NOTSYMS, BLOCKS und NOTBLKS können Gruppen von Namen in einer Kurznotation angeben werden. Eine Gruppe von Namen kann gebildet werden, indem für Zeichenkombinationen ein Ersatzzeichen (- oder *) gesetzt wird. Anstelle eines "-" können mehrere Kombinationen von Zeichen stehen. Der "*" ersetzt ein Zeichen, ausgenommen das Leerzeichen.

Beispiele:

SYMS=ABC** — Alle Variablen, die mit ABC beginnen und aus fünf Buchstaben bestehen, werden ausgegeben.

NOTSYMS=AB-C — Alle Variablen, die mit AB beginnen und mit C enden, werden vom Dump ausgenommen.

8.4.2 Speicher- und Register-Auszug (DUMP)

Die DUMP-Anweisung liest und formatiert den Inhalt der von DUMPJOB erzeugten Datei $DUMP und schreibt die Informationen auf eine andere Datei. Es ist zweckmäßig DUMPJOB und DUMP (wie bei DEBUG) nach einer EXIT-Anweisung zu schreiben, denn so erhält man einen DUMP, unabhängig davon, an welcher Stelle der Job zu einem Fehlerabbruch geführt hat.

```
DUMP<,I=idn,O=odn,FW=fwa,LW=lwa,JTA,NXP,V,DSP,FORMAT=f,CENTER>.
```

Parameter: (die Voreinstellung (default) ist unterstrichen)

I=idn =$DUMP	Name der Datei, die das Hauptspeicherabbild enthält.
O=odn =$OUT	Name der Datei, auf die der Dump geschrieben wird.
FWA=fwa =0	Oktale Anfangsadresse des Hauptspeicherbereichs, der gedumpt werden soll.
LWA=lwa =200	Oktale Endadresse des Hauptspeicherbereichs, der ausgegeben werden soll. Wird nur LWA angegeben, gilt die Adresse des JOB-Limits.
JTA	Die Job Table Area (JTA) wird ausgegeben.
NXP	Das Exchange Package wird nicht ausgegeben.
V	Die Vektorregister werden ausgegeben.
DSP	Die Logical File Tables (LFTs) und die Dataset Parameter Areas (DSPs) werden ausgegeben.
FORMAT=f	Format in dem der Hauptspeicherbereich (Parameter FWA und LWA) dargestellt wird. Die folgenden Formate sind möglich:
=O	Oktale Integerzahlen und ASCII-Zeichen,
=D	Dezimale Integerzahlen und ASCII-Zeichen,
=X	Hexadezimale Integerzahlen und ASCII-Zeichen,
=G	Gleitkommazahlen und ASCII-Zeichen,
=P	16-Bit-Packete (Parcels),
=M	gemischt sedezimal und oktal.
CENTER	100 (oktal) Worte unterhalb und oberhalb der im P-Register stehenden Adresse werden im P-Format gedumpt.

8.4.3 Die Wiederherstellung der FLOWTRACE-Tabellen (FLODUMP)

Die FLODUMP-Anweisung stellt die FLOWTRACE-Tabellen wieder her und druckt sie aus, wenn ein Programm, das mit FLOWTRACE gestartet wurde, abbricht. Die FLOWTRACE-Tabellen werden im FORTRAN FLOWTRACE Format ausgedruckt. Wird FLODUMP benutzt, so muß auf der CFT-Anweisung die F-Option gesetzt worden sein (==> FORTRAN CFT 7.1.3).

```
FLODUMP.
```

Beispiel der Jobaufbau bei Verwendung von FLODUMP:

```
JOB.
ACCOUNT,AC=xxxx,APW=GEHEIM.
CFT,ON=F.
LDR.                    (besser: SEGLDR,GO)
EXIT.
DUMPJOB.
FLODUMP.
/EOF
PROGRAM LOOPING
END
/EOF
```

8.5 Erzeugen von Dumps zur Programmlaufzeit

Das Bibliotheksprogramm SYMDEBUG dient der Erzeugung von "schnappschußartigen" Speicherabzügen während der Laufzeit des Programms. An der Stelle des FORTRAN-Programms, an der ein Abbild des Hauptspeicherbereiches zur späteren Untersuchung erzeugt werden soll, gibt man die Anweisung

```
CALL SYMDEBUG('opt<,opt...>.')
```

Die Optionen opt sind die gleichen, die im Kommando DEBUG gegeben werden können, mit Ausnahme von DUMP (==> 8.4.1). Die Voreinstellung und die Syntax stimmen ebenfalls überein. Vorsicht bei Aufruf aus DO-Schleifen: SYMDEBUG benötigt Speicherplatz im "Stack"; ggf. müssen im Kommando SEGLDR die Angaben STACK und HEAP verändert werden (==> 3.3.3)!

Beispiel:

```
CALL SYMDEBUG('L=DUMP5,CALLS=3,BLOCKS.')
```

9. Einführung in die Optimierung der Rechenzeit

In der angewandten Mathematik werden Objekte durch arithmetische und logische Operationen verknüpft und in neue Objekte umgewandelt. Diese Objekte sind hauptsächlich Skalare (Zahlen), Vektoren und Matrizen.

Vektoroperationen lassen sich auf skalare Operationen mit den einzelnen Komponenten der Vektoroperanden zurückführen. Ebenso können Matrizenoperationen als Vektoroperationen mit den einzelnen Zeilen und Spalten der Matrizen beziehungsweise als skalare Operationen mit den einzelnen Elementen der Matrizen aufgefaßt werden.

Herkömmliche Rechner und die dafür entwickelten Programmiersprachen wie FORTRAN bieten Darstellungen für Skalare (Variable), Vektoren und Matrizen (Felder oder Arrays) aber nur Operationen mit Skalaren. Vektor- und Matrizenoperationen müssen als skalare Operationen programmiert werden (z.B. in FORTRAN durch DO-Schleifen).

Im Abschnitt 9.1 wird vorgestellt, wie die CRAY-Architektur auf die effiziente Bearbeitung von Vektoroperationen ausgerichtet ist. Abschnitt 9.2 erläutert, wie Vektoroperationen auf der CRAY ablaufen und wie Parallelarbeit den gesamten Rechenvorgang beschleunigt.

In den folgenden Abschnitten werden dann die wichtigsten Programmiertechniken für die vektorielle Verarbeitung mit FORTRAN-Programmen zusammengestellt. Abschnitt 9.3 zeigt, wie der CFT-Autovektorisierer in sequentiellen FORTRAN-Programmen vektorielle Operationen erkennt und wie er vom Programmierer unterstützt werden kann. Weitere Optimierungsmöglichkeiten werden in Abschnitt 9.4 vorgestellt. Abschnitt 9.5 weist auf die Bibliothek von optimierten Unterprogrammen hin.

Die Begriffe Multitasking, Makrotasking und Mikrotasking haben nichts mit der Optimierung der Rechenzeit zu tun. Beide Techniken dienen vielmehr dazu, die geforderte Leistung auf zwei oder mehr Teilaufträge (tasks) zu verteilen. Diese können dann mehrere vorhandene Prozessoren unabhängig voneinander ausnutzen.

Da die Aufteilung in Teilaufträge und die Synchronisation Zeit kosten, steigt im allgemeinen die Gesamtrechenzeit. Aber die Verweilzeit im Rechner kann deutlich (bei zwei Prozessoren bis etwa zum Faktor 1,9) sinken.

Dringend anzuraten sind diese Techniken bei sehr großen Programmen, die den verfügbaren Hauptspeicher der CRAY ausschöpfen.

9.1 Systemarchtitektur der CRAY X-MP

Zur möglichst effizienten Ausnutzung der Rechenleistung sind grundlegende Kenntnisse über Hardware-Eigenschaften der CRAY X-MP sehr nützlich.

Zwei identische Rechenwerke teilen sich den gemeinsamen Hauptspeicher und das Ein-Ausgabesystem. Sie sind in Adreßteil, Skalarteil und Vektorteil mit jeweils eigenen Registern und Funktionseinheiten gegliedert.

Die wichtigsten Bestandteile und Arbeitsweisen der Rechenwerke werden in den folgenden Abschnitten erläutert.

9.1.1 Vektorteil des Rechenwerks

Zur Bearbeitung von Vektoren gibt es
- 8 V-Register mit je 64 Worten (je 64 Bit)
- 1 Vektorlängenregister (7 Bit)
- 1 Vektormaskenregister (64 Bit)

Die Vektorregister können blockweise vom Speicher oder elementweise aus S-Registern geladen werden.

Vektorregister, die als Operanden dienen, sind für die Dauer der Operation belegt. Vektorregister, die das Ergebnis einer Operation aufnehmen, können gleichzeitig als Operanden für die nächste Operation dienen. Die größte Wirkung dieser Verkettung (chaining) ergibt sich, wenn die zweite Operation beginnt, sobald das erste Ergebniselement der ersten Operation vorliegt.

Das Vektorlängenregister ermöglicht Operationen mit weniger als 64 Elementen.

Jedes Bit des Vektormaskenregisters entspricht dem Element eines Vektorregisters. Es steuert die Auswahl der Elemente bei der vektoriellen Mischoperation (vector merge ==> 9.3.4 IF-Anweisungen).

Für verschiedene Operationen gibt es voneinander unabhängige Funktionseinheiten, die alle gleichzeitig arbeiten können. Sie erhalten ihre Operanden von den Registern und liefern ihr Ergebnis in einem anderen Register ab.

Alle Funktionseinheiten sind segmentiert, d.h. in Verarbeitungsstufen geteilt, die jeweils einen Maschinentakt für ihren Arbeitsanteil brauchen. Damit kann jede Funktionseinheit in jedem Takt das nächste Element des oder der Operanden entgegennehmen und nach einer für jede Operation festen Zahl von Takten in jedem Takt ein Ergebnis abliefern.

Für die Vektorverarbeitung gibt es sieben Funktionseinheiten
- Gleitpunktaddition/-subtraktion (6 Takte)
- Gleitpunktmultiplikation (7 Takte)
- Reziproke Approximation (14 Takte)
- ganzzahlige Addition/Subtraktion (3 Takte)
- Verschiebeoperation (3 bis 4 Takte)
- logische Verküpfung (2 Takte)
- Zählen der gesetzten Bits (5 Takte)

Die drei Funktionseinheiten für Gleitpunktoperationen werden auch für skalare Operationen benutzt.

Einfach genaue Gleitpunktzahlen bestehen aus einem Vorzeichenbit, 15 bit Exponent zur Basis 2 und 48 Bit Mantisse (==> B.3). Die Rechenergebnisse sind immer normalisert. Für doppelt genaue Berechnungen ist keine Hardware vorgesehen.

Die Reziproke Approximation vollzieht die ersten drei Schritte eines Newton-Verfahrens. Dessen Ergebnis ist auf 30 Bit genau. Für die volle Genauigkeit von 48 Bit wird ein weiterer Newton-Schritt in der Multiplikationseinheit vollzogen.

Für eine Division werden also nach der Reziproken Approximation noch drei Multiplikationen ausgeführt.

9.1.2 Skalarteil des Rechenwerks

Der Skalarteil zum Rechnen mit Zahlen besteht aus
8 S-Registern (je 64 Bit) zum Rechnen
64 T-Registern (je 64 Bit) als Zwischenspeicher
und vier Funktionseinheiten
Addition/Subtraktion (3 Takte)
Verschiebeoperation (2 Takte)
logische Verknüpfung (1 Takt)
Zählen von gesetzten oder nicht gesetzten Bits (3 bis 4 Takte)

Die Additionseinheit verarbeitet ganzzahlige 64-Bit-Werte ohne Überlauf. Eine Multiplikationseinheit für ganzzahlige 64-Bit-Werte gibt es nicht.

Für Gleitpunktoperationen mit Skalaren werden die Funktionseinheiten des Vektorteils mitbenutzt.

9.1.3 Adreßteil des Rechenwerks

Für Adreß- ind Indexrechnungen sowie Schleifenkontrollen stehen zwei Sätze von Registern
8 A-Register (je 24 Bit) zum Rechnen
64 B-Register (je 24 Bit) als Zwischenspeicher
und zwei Funktionseinheiten
Addition/Subtraktion (2 Takte)
Multiplikation (4 Takte)
zur Verfügung.

Die beiden Funktionseinheiten bieten ganzzahlige 24-Bit-Arithmetik auf den A-Registern ohne Überlauferkennung.

9.1.4 Befehlspuffer und Befehlsabarbeitung

Jedes Rechenwerk hat vier Befehlspuffer für je 128 aufeinanderfolgende Befehlspakete zu je 16 Bit (=32 Worte).

Wenn der nächste auszuführende Befehl in einem anderen Puffer als der vorhergehende liegt, tritt eine Verzögerung von zwei Takten ein.

Wenn der nächste Befehl in keinem Puffer liegt, wird ein Puffer ausgewählt und vom Speicher aus neu geladen. Bis der Befehl dann vorliegt, vergehen 16 bis 19 Takte.

Sprünge vorwärts oder rückwärts innerhalb eines Puffers bedeuten keine Verzögerung.

9.1.5 Hauptspeicher

Der Hauptspeicher von vier Millionen Worten zu je 72 Bit (64 Datenbits und 8 Prüfbits SECDED, single error correction and double error detection) hat eine Zykluszeit von vier Maschinentakten (=38 ns). So lange ist eine Speichereinheit belegt für den Transport eines Wortes.

Zugriffszeiten für Transporte vom Speicher in die Register:

14 Takte	für Adressen und Skalare
17 + vektorlänge	für Vektoren
16 + blocklänge	für Blockübertragung in B- und T-Register

Maximale Übertragungsraten:

B-, T- und V-Reg.	1 Takt für 3 Worte
A- und S-Register	2 Takte für 1 Wort
Befehlspuffer	4 Takte für 128 Pakete (=32 Worte)
Ein-/Ausgabe	1 Takt für 2 Worte

Organisation des Hauptspeichers

Der Hauptspeicher ist in vier Abschnitte zu je acht Bänke geteilt. Jedes der beiden Rechenwerke hat je eine Verbindung zu jedem der vier Abschnitte:

Im Abschnitt 0 liegen die Bänke 0,4,8,...,28. Im Abschnitt 1 liegen die Bänke 1,5,9,... u.s.w.

Jedes Rechenwerk hat vier Zugriffspfade zum Speicher, nämlich drei für die Registerübertragung:

A	Vektor- und B-Register laden
B	Vektor- und T-Register laden
C	Skalare laden und alle Speicheroperationen

und einen davon unabhängigen Pfad für Ein- und Ausgabe.

Gleichzeitiges Laden und Speichern von Vektoren oder Blöcken (bidirectional memory mode) kann zu Überlappungen und Fehlern führen, die durch die Programmierung vermieden werden müssen. Normalerweise verhindert der FORTRAN-Übersetzer diese Überlappungen (==> 9.3.4).

Für skalare Übertragungen müssen die drei Pfade A, B und C frei sein, um nicht mit Blockübertragungen zu überlappen.

Wenn Befehle zu laden sind, werden alle acht Verbindungen zum Hauptspeicher gesperrt (also für beide Rechenwerke): nach spätestens drei Takten sind die laufenden Vorgänge abgeschlossen und in den nächsten vier Takten werden dann 32 Worte von allen 32 Bänken geladen.

Speicherzugriffskonflikte

In der CRAY X-MP sind drei Arten von Speicherzugriffskonflikten möglich:

Bank belegt: Wenn ein Zugriffspfad eine noch aktive Bank ansprechen will, werden alle Pfade dieses Rechenwerks für ein, zwei oder drei Takte angehalten, bis die Bank wieder frei ist.

Gleiche Bank: Wenn zwei Pfade aus verschiedenen Rechenwerken dieselbe Bank ansprechen, muß ein Rechenwerk einen Takt warten. Anschließend liegt ein Bank-belegt-konflikt vor.

Gleicher Abschnitt: Wenn zwei Pfade aus einer CPU Bänke in demselben Abschnitt ansprechen, muß ein Pfad einen Takt warten.

Welche Pfade und Rechenwerke jeweils zu warten haben, wird durch geeignete Prioritäten geregelt.

9.1.6 Kommunikation der Rechenwerke untereinander

Allen Rechenwerken der CRAY X-MP stehen die Uhr und drei Sätze gemeinsamer Register zur Verfügung. Jeder Satz besteht aus
8 gemeinsamen Adreßregistern (je 24 Bit)
8 gemeinsamen Skalarregistern (je 64 bit)
32 gemeinsamen Semaphore-Registern (je 1 Bit)

Diese Register werden unter anderem auch beim Mikrotasking verwendet.

9.2 Prinzip der Vektorverarbeitung

Vektoren im Sinne der CRAY sind zusammengehörige Folgen von ganzen oder einfach-genauen Gleitpunktzahlen (Skalaren), deren Elemente mit konstantem Abstand im Speicher abgelegt sind. Dabei ist es völlig gleichgültig, ob es sich um Darstellungen von Vektoren im mathematischen Sinne handelt. In FORTRAN: eindimensionale Felder, Spalten oder Zeilen einer Matrix.

Beispiel:

Sollen in einem herkömmlichen Rechner zwei Vektoren A und B der Länge N addiert und das Ergebnis dem Vektor X zugewiesen werden, dann muß der Rechner:

je eine Komponente von A und eine von B in (skalare) Register laden,
die beiden Register im Rechenwerk addieren,
das Ergebnisregister in die zugehörige Komponente von X speichern
und diese Befehle in einer Schleife N-mal durchlaufen.

in FORTRAN:

```
      DIMENSION X(500), A(200),B(100)
      DO 10 I = 1,100,2
10    X(I) = A(I) + B(I)
```

Die Felder A, B und X sind *Vektoren*. Die *Vektorlänge* ist im allgemeinen nicht gleich der Dimension der Felder, sondern gleich der Anzahl der Schleifendurchläufe. Im Beispiel ist die Vektorlänge 50. Der *konstante Abstand*, mit dem die Vektorelemente im Speicher abgelegt sind, ist im Beispiel gleich der Schrittweite der DO Schleife (also = 2).

An die Stelle dieser Folge von Skalaroperationen (sequentielle Architektur) treten bei CRAY Vektoroperationen (Vektorarchitektur).

9.2.1 Vektoroperationen auf der CRAY

Bei dem obigen Beispiel der Vektoraddition werden die Vektoren X und Y durch je einen Ladebefehl in je ein Vektorregister geladen. Durch einen Additionsbefehl werden die beiden Register im Rechenwerk addiert und wird das Ergebnis in einem dritten Vektorregister gesammelt. Ein Speicherbefehl schreibt dann den Ergebnisvektor zurück.

Wenn die Vektorlänge größer als 64 ist, sehen die Vektoroperationen nicht ganz so einfach aus. Da die Vektorregister der CRAY nur 64 Worte lang sind, müssen längere Vektoren intern in Abschnitten der Länge 64 bearbeitet werden (==> 9.1.1).

Entsprechend den Funktionseinheiten des Vektorteils (==> 9.1.1) gibt es vektorielle Operationen für einfach-genaue Gleitpunktvektoren:
   Addition und Subtraktion,
   Multiplikation und
   Approximation der Reziproken
und für ganzzahlige Vektoren
   Addition und Subtraktion
sowie bitweise logische Verknüpfung.

9.2.2 Wichtige Begriffe der Vektorverarbeitung

Pipeline

Die Komponenten eines Vektors werden in der CRAY nicht parallel im Sinne von gleichzeitig verarbeitet. Zentral für das Verständnis der Vektorverarbeitung ist der Begriff der Pipeline und damit die Segmentierung der Funktionseinheiten (==> 9.1.1).

Beispiel:

Eine Einheit zur Vektoraddition von Gleitpunktzahlen sei der Einfachheit halber in vier Segmente geteilt:
   Vergleich der Exponenten,
   Angleichung der Mantissen,
   Addition der Mantissen und
   Normalisierung.

Zunächst werden die Exponenten der jeweils ersten Komponenten verglichen. Im nächsten Takt werden die Mantissen entsprechend angepaßt und gleichzeitig die Exponenten der jeweils zweiten Komponenten verglichen.

Wenn im vierten Takt die ersten Komponenten normalisiert werden, können bereits die Exponenten der vierten Komponenten verglichen werden. Nach dieser Anlaufzeit liefert das Rechenwerk mit jedem Maschinentakt eine Ergebniskomponente.

Verkettung (chaining)

Die unabhängigen Funktionseinheiten der Vektoreinheit können auch zu längeren Pipelines verkettet werden. Wenn ein arithmetischer Ausdruck mit verschiedenen Vektoren und unterschiedlichen Operationen zu berechnen ist, können
- das Laden der Vektorregister,
- die Multiplikation,
- die Addition,
- die Bildung des Absolutbetrages und
- das Speichern

miteinander verknüpft werden.

Wenn die ersten Vektorkomponenten geladen werden, können sie gleichzeitig in die erste Funktionseinheit, z.B. die Multiplikation, eintreten. Sobald die Komponenten eine Funktionseinheit verlassen, können sie in einer anderen Funktionseinheit, z.B. der Addition, weiterverarbeitet oder abgespeichert werden, während immer noch die weiteren Komponenten der ersten Operanden geladen werden.

Diese Verknüpfung heißt bei CRAY Verkettung (Chaining). Sie setzte bei der CRAY 1-M voraus, daß mit jedem Maschinentakt die nächste Vektorkomponente vom Register oder Speicher bereitstand beziehungsweise dorthin abfließen konnte. Sonst wurde die Verkettung unterbrochen und die Vektoroperationen wurden nacheinander abgearbeitet. An der CRAY X-MP können Operationen auch ohne diese strenge Voraussetzung verkettet werden.

9.3 Vektorisierung von FORTRAN-Programmen

Standard-FORTRAN bietet keine Sprachmittel für Vektoroperationen. Im CRAY-FORTRAN CFT wurden sie auch nicht als Spracherweiterungen eingeführt.

Aus der Erkenntnis, daß Vektoroperationen in FORTRAN komponentenweise - häufig in DO-Schleifen - programmiert sind, versucht der Autovektorisierer von CFT, aus DO-Schleifen Vektoroperationen herauszulesen. Ist er erfolgreich, so spricht man davon, daß die Schleife vektorisiert.

In einigen Zweifelsfällen, die erst zur Laufzeit entscheidbar sind, erzeugt der Übersetzer sowohl skalaren als auch vektoriellen Code mit einer Abfrage, die beim Ablauf die sequentielle oder die parallele Variante auswählt. Man spricht dann von bedingter Vektorisierung.

Vorteil dieses Vorgehens ist die Portabilität, d.h. daß in Standard-FORTRAN (sequentiell) geschriebene Programme die Vektoreigenschaften der CRAY nutzen können und weiterhin auf sequentiellen Rechnern ablauffähig sind.

Nachteil ist der Umstand, daß es schwierig bis unmöglich sein kann, in den sequentiellen Anweisungen automatisch Vektoroperationen zu erkennen und die volle Leistungsfähigkeit der CRAY zu nutzen.

Zur Minderung dieses Nachteils bietet CRAY einerseits eine Reihe von optimierten Routinen (==> 9.5) und andererseits im CFT Möglichkeiten, mit denen der Programmierer den Autovektorisierer unterstützen kann. Sie werden in den folgenden Abschnitten vorgestellt.

9.3.1 Vorgehensweise bei der Optimierung

Ziel bei der FORTRAN-Programmierung an der CRAY ist, einen hohen Anteil an vektorisierbarem Code mit langen Vektoren zu erreichen, um hohe Verarbeitungsgeschwindigkeiten zu erzielen. Folgende Vorgehensweise ist dabei sinnvoll:

a) Das Programm muß richtig laufen, d.h. die richtigen Ergebnisse liefern

b) Lasse den Autovektorisierer von CFT innere DO-Schleifen vektorisieren. Er gibt Meldungen aus, wenn sie nicht vektorisierbar sind.

c) Stelle mit FLOWTRACE fest, in welchen Routinen die meiste Rechenzeit verbraucht wird (==> 7.1.3 Option ON=F).

d) Analysiere nichtvektorisierte DO-Schleifen.

e) Unterstütze den Autovektorisierer durch Optionen und Direktiven.

f) Verwende geeignete optimierte Routinen aus $SCILIB (==> 9.5).

g) Prüfe andere Optimierungsmöglichkeiten, z.B. Abrollen von Schleifen.

Wenn diese Maßnahmen zu keinem befriedigenden Ergebnis führen:

h) Überarbeite das Programm und verwende z.B. einen anderen Algorithmus oder eine Struktur, die sich besser zur Parallelverarbeitung eignen.

Generell ist zu bedenken:

i) Nutze möglichst vorhandene optimierte Software.

9.3.2 Was vektorisiert der CFT-Autovektorisierer?

Die Kriterien zur Entscheidung, ob der Übersetzer Vektoroperationen erkennen kann, können sich relativ schnell ändern. Der Autovektorisierer wird mit jeder neuen Version des Übersetzers verbessert und erweitert.

Die folgende Aufstellung enthält Bedingungen, die für CFT 1.15 erfüllt sein müssen, damit DO-Schleifen vektorisiert werden können (die auftretenden Begriffe werden nach dem Beispiel erläutert):

* Nur innere DO-Schleifen sind vektorisierbar.
* Mindestens eine Vektorreferenz muß in der Schleife links von einem Gleichheitszeichen stehen oder eine Vektorreduktionen vorliegen.
* Alle Variablen sind Vektorreferenzen, CIVs, Temporäre Vektoren, Invarianten oder Pseudovektoren.
* Die arithmetischen Ausdrücke verwenden REAL- (64 Bit), INTEGER- (46 Bit) und COMPLEX-Arithmetik (zweimal 64 Bit).
* Alle Aufrufe von FORTRAN Intrinsic Funktionen mit Vektorversion (SIN, COS, ABS, MAX, SQRT usw.) haben vektorisierbare Ausdrücke als Argumente.

Einfache IF-Anweisungen wie bedingte Zuweisungen, Suchschleifen und bedingte Blöcke (IF-THEN-ELSE) verhindern die Vektorisierung im allgemeinen nicht (==> 9.3.4).

Anweisungsfunktionen werden durch die entsprechenden Ausdrücke mit den Aktualparametern ersetzt. Die Vektorisierbarkeit der resultierenden Anweisung wird mit obigen Regeln überprüft.

Beispiel für eine vektorisierbare Schleife:

```
      S = 0.
      K = 200
      DO 1 M = 100,200,3
        A(M) = 3.5                    Feldelement mit CIV M
        B(K) = A(M)+SIN(A(M))         SIN ist vektorisierbare Funktion
        T = C(K) ** 2 + 1.2           T ist Temporärer Vektor
        S = S + C(K)                  Vektorreduktion
        K = K - 1                     K ist CIV
        A(M-1) = 23.3+SQRT(T)         SQRT ist vektorisierbare Funktion
        E(M,K) = 0.0                  E  ist Pseudovektor,  da mit zwei  CIVs
                                      indiziert
1     CONTINUE
```

Invariante

Invariant heißt eine Konstante oder eine Variable, die in einer Schleife benutzt aber nicht verändert wird.

Beispiel:

```
      DO 20 I = 1, 100
20    A(I) = B(I) + X*Z**K            X, Z und K sind Invariante
```

CIV (constant increment variable)

Die Elemente eines für Vektoroperationen geeigneten Feldes müssen im Speicher mit konstantem Abstand (increment) abgelegt sein. Zur Adressierung dieser Felder sind daher nur Variable geeignet, deren Wert bei jedem Durchlauf der Schleife um einen invarianten Ausdruck erhöht oder vermindert werden.

Solche Variablen vom Typ INTEGER oder REAL werden als CIV bezeichnet. Der definierende Ausdruck darf nur die Operationen Plus und Minus und keine Klammern enthalten. Er hat also den Typ:

```
CIVa = CIVa +- Invarianter Ausdruck
CIVa = CIVb +- Invarianter Ausdruck
```

Beispiele:

```
      DO 10 I = 10,30
        J  = J - 8
        K  = 4 + M * I
 10     A(I) = B(J) * C(K)
```

I, J und K sind CIVs, M ist invariant unddie Schleife vektorisiert.

```
      DO 20 I = 10,30
           J = J - I
           K = 4 + K * I
 20        A(I) = B(J) * C(K)
```

I ist CIV aber J und K verändern sich mit wachsenden Abständen und verhindern die Vektorisierung.

Temporärer Vektor (scalar temporary)

Temporäre Vektoren sind Variable, die die folgenden Bedingungen erfüllen: Sie werden innerhalb einer Schleife nur einmal durch einen vektorisierbaren Ausdruck definiert und führen, wenn sie an weiteren Stellen in der Schleife durch den definierenden Ausdruck ersetzt werden, auch dort zu vektorisierbaren Anweisungen.

Temporäre Vektoren werden in einem Vektorregister gehalten. Am Schluß der Schleife wird der Wert der letzten Komponente in der skalaren Variablen gespeichert.

Beispiel:

```
      Die Schleife                    wirkt wie

      DO 70 I = 50,10,-3                DO 70 I = 50, 10, -3
        S = C(K) + 4.           70      B(I) = X - A(I) - 1.35
        T = A(I) + 1.35                 DO 71 I = 50, 10, -3
        B(I) = X - T            71      D(I) = SIN(C(K) + 0.65 + A(I) - B(I))
        D(I)= SIN(S+T-B(I))
70    CONTINUE
```

Im obigen Beispiel sind T und S Temporäre Vektoren. Die Schleife wird vektorisiert.

Vektorreferenz (vector array reference)

Als Vektorreferenz wird eine Feldreferenz bezeichnet, bei der ein Indexausdruck eine CIV enthält und alle weiteren Indexausdrücke invariant sind.

Der Indexausdruck, der die CIV enthält, muß ein linearer Ausdruck sein, der in die Form

```
+- invarianter Ausdruck1 * CIV +- invarianter Ausdruck2
```

überführt werden kann, wobei nur Ausdruck1 Multiplikationen enthalten darf.

Beispiel:

```
      DO 40 K = 100,1,-1
      I = I + 1
40    A(K) = B(J*K + L,J) + C(I,K)
```

A und B sind in Vektorreferenzen verwendet, C nicht.

Ausdruck1 und Ausdruck2 müssen von relativ einfacher Form sein, d.h. Klammern in dem CIV-Indexausdruck können die Vektorisierung verhindern. Klammerausdrücke, die in die folgende Form überführt werden können, lassen die Vektorisierung zu:

```
+- variable  +*- ( variable  +*- variable )
   konstante       konstante     konstante
```

Beispiele:

Feldreferenz	Vektorreferenz
A((I+2))	A(I+2)
A(3-(I+2))	A(1-I)
A(J*(I+K))	A(J*I+J*K)
A((I-2)+3)	A(I+1)
A(K+(I*3))	A(K+I*3)

Aber kompliziertere CIV-Indexausdrücke mit mehr Termen oder mehr Klammern verhindern die Vektorisierung.

Beispiele:

Feldreferenz	keine Vektorreferenz, da
A((I+(3-(J)))	Klammern zu tief geschachtelt
A(2*(I+K+2))	zu viele Terme in der Klammer

In diesen beiden Fällen meldet CFT 'too many indirect references'.

Vektorreduktion (vector reduction)

Die Verknüpfung der Komponenten eines Vektors in einen Skalar durch eine arithmetische Operation in der Form

skalar = skalar +-*/ (vektorisierbarer Ausdruck)

erzeugt aus Vektoren eine skalare Größe. Dieser Vorgang wird in der CRAY-Literatur als Vektorreduktion bezeichnet. Die skalare Reduktionsvariable darf in dem vektorisierbaren Ausdruck nicht auftreten. Für INTEGER-Größen sind nur Plus und Minus erlaubt.

Beispiel:

```
      X = 0.0
      DO 50 I = 1,N
50    X = X + A(I)*B
```

Pseudovektor

Pseudovektor heißt eine Feldreferenz, die nicht die Kriterien für eine Vektorreferenz erfüllt, die aber einen vektorisierbaren Indexausdruck hat. Die Feldreferenz wird dann als skalare Unterschleife in der Vektorschleife behandelt, wobei der Indexausdruck als Vektorausdruck in der Vektorschleife berechnet wird.

Maschinen mit Hardware für einen sogenannten komprimierten Index (compressed index hardware, am ZIB nicht installiert) können die skalare Unterschleife mit einem Befehl berechnen.

9.3.3 Was vektorisiert der CFT-Autovektorisierer nicht?

DO-Schleifen, die eines der nachfolgenden Elemente enthalten, werden derzeit nicht vektorisiert:

* eingeschachtelte DO-Schleifen
* Abhängigkeiten in den Feldindizes
* Nicht-lineare Feldindizes, insbesondere
  - Temporäre Vektoren als Index
  - Indizierte Indizierung
* Ein-/Ausgabe-Anweisungen
* CALL-Anweisungen sowie Aufrufe von Funktionen, die keine Vektorversion haben
* geschachtelte IF-Anweisungen und ELSEIF-Anweisungen
* Rückwärtssprünge innerhalb der Schleife
* Variable, Felder oder Funktionen vom Typ CHARACTER
* eine der Anweisungen RETURN, STOP oder PAUSE
* Überprüfung der Feldindizes für ein Feld (==> 7.1.3 Option ON=0)
* Option DEBUG (==> 7.1.3)
* wenn die Schleife länger ist als die Größe MAXBLOCK (==> 7.1.2)

Die verwendeten Begriffe werden im folgenden erläutert.

Abhängigkeiten in den Feldindizes (Dependencies)

Besonders sorgfältig muß man Schleifen betrachten, in denen die gleiche indizierte Variable als Operand in mehreren Anweisungen auftritt. Bei normaler Abarbeitung der Schleife wird in manchen Fällen der in einer vorangehenden Anweisung veränderte Wert dieser Variablen in einer nachfolgenden Anweisung verwendet. Bei vektorieller Verarbeitung wird jede Anweisung für alle Elemente des Vektors ausgeführt, ehe die nächste Anweisung ausgeführt wird.

Die unterschiedliche Reihenfolge der Ausführung der Anweisungen kann zu unterschiedlichen Ergebnissen führen. Dabei gilt immer das Ergebnis der sequentiellen Verarbeitung als richtig.

Derartige Zusammenhänge werden als Abhängigkeiten bezeichnet. Die folgenden Beispiele verdeutlichen den Sachverhalt und zeigen, wie Abhängigkeiten umgangen werden können.

Beispiel a: (Abhängigkeit)

Das Resultat des vorigen Schleifendurchlaufs liegt noch nicht vor.

```
      DO 10 I = 1,N
        B(I) = A(I-1)              nicht vektorisierbar
        A(I) = C(I)
10    CONTINUE
```

Bei sequentieller Verarbeitung wird A(I-1) im jeweils vorangehenden Schleifendurchlauf durch A(I) neu definiert. Bei Vektorverarbeitung wird dagegen zunächst der ganze Vektor A referiert und erst danach neu definiert.

Die vektorielle Verarbeitung würde also wirken wie

```
      DO 10 I = 1,N
10    B(I) = A(I-1)
      DO 20 I = 1,N
20    A(I) = C(I)
```

Da dieses ein anderes Ergebnis liefern würde, vektorisiert CFT die Schleife nicht.

Beispiel b: (Rekursion)

Rekursion ist ein Spezialfall der Abhängigkeit.

```
      DO 10 I = 1,N
10    B(I) = B(I-1)* ...          nicht vektorisierbar
```

Beispiel c:

Der folgende Fall sieht ähnlich aus, wird aber seit CFT 1.15 vektorisiert.

```
      DO 10 I = 1,N
        A(I) = C(I)
        B(I) = A(I+1)             vektorisierbar
10    CONTINUE
```

Bei sequentieller Verarbeitung wird A(I+1) erst im jeweils nächsten Schleifendurchlauf durch C(I) überschrieben. Bei Vektorverarbeitung wird intern ein temporärer Vektor zum Zwischenspeichern von A(I+1) gebildet, als hätte man geschrieben:

```
      DO 10 I = 1,N
        T = A(I+1)
        A(I) = C(I)               vektorisierbar
        B(I) = T
10    CONTINUE
```

Beispiel d: (Abhängigkeit durch unbekannten Index)

```
      K = 2
      DO 10 I = 1,N
10    A(I) = A(I+K)               bedingt vektorisierbar
```

Der Übersetzer erkennt nicht, ob K positiv oder negativ sein wird und erzeugt skalaren und vektoriellen Code mit einer Abfrage zur Laufzeit, welcher Zweig zu durchlaufen ist.

Die folgenden formalen Regeln legen fest, wann eine Abhängigkeit beim Zugriff auf ein Feld A innerhalb einer DO Schleife vorliegt.

I sei innerhalb der Schleife des folgenden Diagramms ein CIV. Die Schleife enthält bezogen auf die Zuweisung zur Zielvariablen A(I) zwei Blöcke. Der erste Block enthält alle Elemente (Variablen, Feldelemente usw.), die zeitlich vor der Zuweisung zu A(I) auf der rechten oder linken Seite einer Zuweisung verwendet werden (previous); der zweite Block enthält diejenigen Elemente, die zeitlich nach der Zuweisung zu A(I) verwendet werden (subsequent).

Wird I positiv (oder negativ) inkrementiert, so liegt eine Abhängigkeit vor, falls im ersten Block ein Zugriff auf ein Feldelement A(I-k) (oder A(I+k)) erfolgt. CFT meldet in diesen Fällen 'previous minus with incrementing subscript' beziehungsweise 'previous plus with decrementing subscript'.

Abhängigkeit bei positiv inkrementiertem I (n > 0):

```
        DO 10 I = 1,N
         ___previous_____________
        !      ... A(I-k) ...    !      Abhängigkeit
        !_____                   !
        A(I) =      !____________!
         ___subsequent___________
        !      ... A(I+k) ...    !      keine Abhängigkeit
        !________________________!
10      CONTINUE
```

9.3.4 Maßnahmen zur Unterstützung der Vektorisierung

Die hier zusammengestellten Hinweise und Ratschläge können veralten und hinfällig werden, wenn der Autovektorisierer in zukünftigen Versionen verbessert wird.

Abhängigkeiten

Wenn sich eine nichtvektorisierte Schleife ohne Änderung der Ergebnisse umprogrammieren läßt, oder wenn aus zusätzlichen Informationen über Daten und Programmablauf bekannt ist, daß sich eine vorhandene Abhängigkeit nicht auswirken kann, läßt sich die Vektorisierung oft noch erreichen.

zu Beispiel 9.3.3.a:

```
      DO 10 I = 1,N
        A(I) = C(I)
        B(I) = A(I-1)            vektorisierbar
 10   CONTINUE
```

Die Verarbeitungsreihenfolge im modifizierten Code stimmt bei sequentieller Bearbeitung und bei Vektorverarbeitung überein.

zu Beispiel 9.3.3.d:

```
      DO  10  I  = 1,N
        A(I)  = A(I+2)           vektorisierbar
 10   CONTINUE
```

Die mögliche Abhängigkeit kann aufgelöst werden, wenn eine Konstante eingesetzt wird.

Vorsicht bei Direktive CDIR$ IVDEP

Wenn der Programmierer sicher weiß, daß die Abhängigkeit im Ablauf mit den möglichen Daten nie auftreten kann, läßt sich die Schleife mit der Direktive IVDEP (==> 7.2.2) trotz Abhängigkeit vektorisieren. Aber aus zwei Gründen ist Vorsicht geboten.

a) Bei Verwendung von IVDEP wird die Schleife auch dann vektorisiert wenn eine echte Abhängigkeit besteht; dies führt zu falschen Resultaten.

b) An der CRAY X-MP kann es wegen der Möglichkeit des gleichzeitigen Lesens und Schreibens vom und zum Hauptspeicher bei der Vektorverarbeitung mit einem einzigen Feld zu Überlappungen und damit ebenfalls zu falschen Ergebnissen kommen.

Die Übersetzer-Direktive CDIR$ IVDEP sollte nicht oder nur nach sehr sorgfältiger Prüfung der jeweiligen Situation benutzt werden. Ergebnisvergleiche werden dringend empfohlen.

IF-Anweisungen

Bedingte Anweisungen lassen sich zum Teil automatisch durch die Vektorfunktionen MAX oder MIN oder durch die Vektormischoperationen (conditional vector merge) CVMGT, CVMGP, CVMGM, CVMGZ und CVMGN ersetzen.

Anweisungen der Form
```
   IF (var .op. ausdruck) var = ausdruck
```
werden überführt in
```
   var = funktion( var, ausdruck)
```
mit funktion=MIN / MAX, wenn op ein Vergleichsoperator (GT, GE, LT, LE) ist und var und ausdruck denselben Typ haben.

Beispiele:

```
IF( A(I) .GT. B(I) ) A(I) = B(I)       wird zu   A(I) = AMIN1(A(I),B(I))
IF( (I+3)*R1 .GT. R2 ) R2 = (I+3)*R1   wird zu   R2 = AMAX1(A2,(I + 3) * R1)
```

Die allgemeinere Form
```
   IF (bedingung) var = ausdruck
```
wird überführt in
```
   var = CVMGT ( ausdruck, var, bedingung )
```

Beispiel:

```
   IF( (B(I) .GT. C(I)) .OR. (B(I) .LT. A(I)) B(I) = ABS(A(I)*C(I))
```
wird zu
```
   B(I) = CVMGT(ABS(A(I) * C(I)),B(I),(B(I) .GT. C(I)) .OR. (B(I) .LT. A(I)))
```

Die Vektorisierung bewirkt hier, daß der Ausdruck auf der rechten Seite der Zuweisung zunächst für alle Vektorelemente berechnet wird; lediglich die Zuweisung zur Zielvariablen wird in Abhängigkeit von der IF-Bedingung durch das Vektormaskenregister (==> 9.1.1) gesteuert.

Diese Vorgehensweise kann einen der folgenden beiden Nachteile haben:
* Ist die Bedingung nur in wenigen Fällen erfüllt, so kann dies zu einem erheblichen Mehraufwand führen.
* Enthält der Ausdruck auf der rechten Seite Divisionen oder Aufrufe von Intrinsic Funktionen, so kann die vektorielle Verarbeitung zu Laufzeitfehlern, z.B. wegen einer Division durch Null, führen.

Deshalb kann der Anwender den automatischen Einsatz der Vektormischfunktionen steuern durch die Übersetzer-Optionen NOIFCON, PARTIALIFCON und FULLIFCON (==> 7.1.2) und die Direktiven NOIFCON und RESUMEIFCON (==> 7.2.2).

Beispiel a:

```
      DO  10 I = 1,N
        IF (B(I) .GT.  C(I)) A(I) = A(I)  +  B(I)
10    CONTINUE
```

Wird vektorisiert bei Verwendung der Übersetzer-Option PARTIALIFCON oder FULLIFCON.

Beispiel b:

```
      DO 10 I = 1,N
        A(I) = B(I)
        IF (B(I) .GT.  0.) A(I) =  1.  /  B(I)
 10   .CONTINUE
```

Wird nur bei Verwendung der Übersetzer-Option FULLIFCON vektorisiert.

9.4 Weitere Optimierungsmöglichkeiten an der CRAY

9.4.1 Vergrößerung der Vektorlänge

Die Verarbeitungsgeschwindigkeit bei Vektorverarbeitung hängt u.a. wesentlich von der Länge der Vektoren (Anzahl der Schleifendurchläufe) ab. Je länger die Vektoren sind, desto höher ist die Verarbeitungsgeschwindigkeit pro durchgeführter Operation. Ist die Vektorlänge kleiner als 4, so lohnt sich eine Vektorisierung nicht. Da nur innere DO Schleifen vektorisiert werden, sollten die längsten Vektoren in den inneren Schleifen verarbeitet werden. Oft läßt sich die Vektorlänge auch dadurch vergrößern, daß geschachtelte Schleifen auf einfache Schleifen reduziert werden, indem mehrdimensionale Felder auf eindimensionale Felder abgebildet werden.

Beispiel:
nach Melenk, H.: Strategie bei der Multiplikation vieler kleiner Matrizen mit Hilfe von CRAY-Vektorrechnern, ZIB, interner Bericht 1984

Hundert kleine Matrizen (3,3) seien miteinander zu multiplizieren.

```
      DO 300 M = 1,100
        DO 230 K  = 1,3
          DO 200 J  = 1,3
            C(M,K,J)  = 0.0
200       CONTINUE
          DO 220 I  = 1,3
            DO 210 J = 1,3
              C(M,K,J) = C(M,K,J) + A(M,K,I) * B(M,I,J)
210         CONTINUE
220       CONTINUE
230     CONTINUE
300   CONTINUE
```

Hier werden die Matrixprodukte einzelnen nacheinander berechnet. Die innere Schleife 210 vektorisiert mit der unzureichenden Vektorlänge von 3. Eine bessere Leistung erhält man durch:

```
      DO 140 K = 1,3
        DO 130 J = 1,3
          DO 120 M = 1,100
            C(M,K,J) = 0.0
120       CONTINUE
130     CONTINUE
140   CONTINUE
      DO 300 K = 1,3
        DO 230 J  = 1,3
          DO 220 I  = 1,3
            DO 210 M = 1,100
              C(M,K,J) = C(M,K,J) + A(M,K,I) * B(M,I,J)
210         CONTINUE
220       CONTINUE
230     CONTINUE
300   CONTINUE
```

In der inneren Schleife werden jetzt hundert Matrizen simultan mit der Vektorlänge 100 berechnet.

Abrollen von Schleifen (unrolling)

Schleifen abrollen bedeutet, Schleifen zu beseitigen oder die Zahl der Schleifendurchläufe dadurch zu vermindern, daß die zu wiederholenden Operationen ausdrücklich hingeschrieben werden.

Vorteilhaft ist es, wenn durch das Abrollen in der innersten Schleife mehr Operationen und kompliziertere arithmetische Ausdücke entstehen. Der Übersetzer kann dann die Register und die Verkettung von Operationen besser ausnutzen. Ein weiterer Vorteil kann durch die Beseitigung kurzer innerer Schleifen entstehen, wenn dadurch eine Schleife mit größerer Vektorlänge zur innersten wird.

Innere Schleifen mit einer konstanten kleinen Zahl von Durchläufen kann schon der Übersetzer abrollen (==> 7.1.2 Option UNROLL). Das folgende Beispiel zeigt, wie durch die Verminderung von Durchläufen einer äußeren Schleife die innere Schleife mit Vektoroperationen angereichert wird. Dabei sorgt die Klammerung für die beste Ausnutzung der Register.

Beispiel:
nach Dongarra, J.J.; Eisenstat. S.C.: Squeezing the most out of an Algorithm in CRAY-FORTRAN, ACM TOMS, Vol. 10 No.3 1984, S. 219-230

```
      SUBROUTINE SMXPY4 (N1, Y, N2, LDM, X, M)
      REAL Y(*), X(*), M(LDM,*)
C
C   PURPOSE:
C     Multiply matrix M times vector X and add the result to vector Y.
C
C   PARAMETERS:
C     N1 INTEGER, number of elements in vector Y, and number of rows in
C         matrix M
C     Y REAL(N1), vector of length N1 to which is added the product M*X
C     N2 INTEGER, number of elements in vector X, and number of columns
C         in matrix M
C     LDM INTEGER, leading dimension of array M
C     X REAL(N2), vector of length N2
C     M REAL(LDM,N2), matrix of N1 rows and N2 columns
C
```

```
C    Cleanup odd vector
C
       J = MOD(N2,2)
       IF (J .GE. 1) THEN
          DO 10 I = 1, N1
             Y(I) = (Y(I)) + X(J)*M(I,J)
    10    CONTINUE
       ENDIF
C
C    Cleanup odd group of two vectors
C
       J = MOD(N2,4)
       IF (J .GE. 2) THEN
          DO 20 I = 1, N1
             Y(I) = ( (Y(I))
      $                + X(J-1)*M(I,J-1)) + X(J)*M(I,J)
    20    CONTINUE
       ENDIF
C
C    Main loop - groups of four vectors
C
       JMIN = J+4
       DO 40 J = JMIN, N2, 4
          DO 30 I = 1, N1
             Y(I) = ((( (Y(I))
      $             + X(J-3)*M(I,J-3)) + X(J-2)*M(I,J-2))
      $             + X(J-1)*M(I,J-1)) + X(J)  *M(I,J)
    30    CONTINUE
    40 CONTINUE
C
       RETURN
       END
```

Nutzt die Register und Verkettung von Operationen besser als die Version:

```
       SUBROUTINE SMXPY (N1, Y, N2, LDM, X, M)
       REAL Y(*), X(*), M(LDM,*)
C
       DO 20 J = 1, N2
          DO 10 I = 1, N1
             Y(I) = (Y(I)) + X(J)*M(I,J)
    10    CONTINUE
    20 CONTINUE
C
       RETURN
       END
```

9.4.2 Speicherzugriffskonflikte

Bei der CRAY 1-M hatte der Speicher acht Bänke und jede Bank war für einen Zugriff acht Takte belegt. Daher traten Bank-belegt-Konflikte (==> 9.1.5) sofort auf, wenn die Vektorelemente einen geradzahligen Abstand hatten. Bei durch acht teilbaren Abständen konnte die Rechenleistung auf ein Drittel sinken.

Für die CRAY X-MP wurde dieses Problem gemildert. Aber in ungünstigen Fällen können sich auch hier Speicherzugriffskonflikte bemerkbar machen.

Die Leistung der folgenden naiv programmierten Matrizenmultiplikation ist abhängig von der Dimensionierung der Matrizen (= Abstand der Elemente der Zeilenvektoren der Matrix B).

Beispiel:

```
      REAL A(M,M), B(M,M), C(M,M)
C
      DO 100 K = 1, N
      DO 100 J = 1, N
        A(K,J) = 0
      DO 100 I = 1, N
        A(K,J) = A(K,J) + B(K,I) * C(I,J)
  100 CONTINUE
```

Messungen mit Vektorlänge N=250 und verschiedenen Dimensionen haben ergeben:

Abstand teilbar durch	Speicher-abschnitte benutzt	Speicher-bänke benutzt	Leistung
ungerade	4	32	100%
2	2	16	91%
4	1	8	84%
8	1	4	84%
16	1	2	71%
32	1	1	48%

Bei den Abständen, die durch zwei, vier oder acht teilbar sind, werden nicht mehr alle möglichen Zugriffspfade genutzt. Sobald sich der Vektor auf weniger als vier Bänke verteilt, treten Bank-belegt-Konflikte auf.

Speicherzugriffskonflikte werden durch ungerade Abstände der Vektorelemente oder durch Umstellen von Operationen (z.B. Abrollen) vermieden. Die Matrizenmultiplikation kann z.B. auch mit Hilfe der oben angegebenen Routinen SMXPY oder SMXPY4 von Dongarra programmiert werden:

Beispiel:

```
      DO 20 J = 1, N
         DO 10 I = 1, N
            A(I,J) = 0.0
   10    CONTINUE
         CALL SMXPY (N, A(1,J), N, M, C(1,J), B)
   20 CONTINUE
```

Sowohl mit SMXPY als auch mit SMXPY4 ist die Multiplikation nicht empfindlich für Zugriffskonflikte.

Vergleicht man übrigens die Rechenleistung (Operationen pro Sekunde) dieser Varianten und bezieht die für Matrizenmultiplikation spezialisierte Routine MXMA aus der Bibliothek $SCILIB (==> 9.5.1) noch ein, so erhält man für Vektorlänge 250:

100% naive Version (Abstand=1, keine Zugriffskonflikte)
157% mit SAXPY, da die innerste Schleife Vektorergebnisse statt Skalarergebnisse erzeugt
194% mit SAXPY4, da Abrollen die Register besser ausnutzt
284% mit MXMA (der Vorteil ist bei kürzeren Vektorlängen noch größer)

9.4.3 Operationen ohne Hardware-Unterstützung

INTEGER-Multiplikationen und -Divisionen vermeiden

INTEGER-Multiplikationen und -Divisionen sind auf der CRAY erheblich langsamer als die entsprechenden REAL-Operationen (Multiplikation: bis Faktor 7, Division: bis Faktor 20). Deshalb sollten in diesen Fällen möglichst REAL-Arithmetik oder 46-Bit INTEGER benutzt werden.

Die Übersetzer-Option FASTMD (==> 7.1.2) bewirkt, daß für ganzzahlige Multiplikation die Gleitpunkteinheit verwendet wird, und daß damit die Zahlen durch die Mantissenlänge auf 2^{46} beschränkt sind (==> 9.1).

In Einzelfällen kann auch die Verwendung von INTEGER*2 (24-Bit INTEGER) sinnvoll sein. Wenn ganze Zahlen zu verarbeiten sind, die sicher kleiner als 2^{24} bleiben, können auch die Funktionseinheiten des Adreßteils verwendet werden (==> 9.1 und 7.1.2 Übersetzer-Direktive INT24).

DOUBLE PRECISION vermeiden

Für DOUBLE PRECISION-Operationen stehen auf der CRAY keine Hardware-Instruktionen zur Verfügung. Daraus ergeben sich erhebliche Verlangsamungen bei der Ausführung gegenüber normaler REAL-Arithmetik (Addition, Subtraktion und Division: bis Faktor 60; Multiplikation: bis Faktor 35).

DOUBLE PRECISION-Operationen sollten deshalb nur verwendet werden, wenn einfache Genauigkeit nicht ausreicht (Doppelte Genauigkeit der CRAY entspricht ungefähr der vierfachen Genauigkeit bei IBM).

Mit Hilfe der Option OFF=P (==> 7.1.4) kann ein Programm, das in doppelter Genauigkeit geschrieben wurde, mit der einfach genauen Arithmetik übersetzt werden.

9.5 Benutzung optimierter CRAY Routinen: die Bibliothek $SCILIB

Einige Situationen in FORTRAN-Programmen lassen eine Vektorisierung auch nach Modifikation des FORTRAN-Quellprogramms nicht zu oder die volle Leistung kann erst in Assembler erreicht werden. Insbesondere in diesen Fällen ist die Verwendung hochoptimierter CRAY-Bibliotheksroutinen sinnvoll.

Diese Routinen ermöglichen in einigen Fällen eine Vektorisierung auch da, wo in FORTRAN eine Vektorisierung nicht möglich wäre (z.B. Minimum- / Maximumsuche in einem Vektor); in anderen Fällen erlauben sie zumindest eine effizientere (nicht-vektorisierende) Bearbeitung von Problemen, als sie in FORTRAN möglich wäre (z.B. Auflösen von linearen Rekursionen). Unter Umständen empfiehlt sich die Benutzung von anderen CRAY-optimierten Programmen, z.B. aus der linearen Algebra (Berechnung des Skalarprodukts, Matrizenprodukte, Lösung linearer Gleichungssysteme etc.).

In diesem Abschnitt wird ein Überblick über Unterprogramme gegeben, die nicht zum FORTRAN Standard gehören. Sie sind von CRAY zur Verfügung gestellt, um bestimmte Aufgaben in einer dem Rechner und seiner Architektur angepaßten Weise lösen zu können. Eine vollständige und ausführliche Dokumentation findet man im Library Reference Manual (==> 1.7).

Diese Routinen stehen ohne besondere Maßnahmen jedem Benutzerprogramm zur Verfügung, da sie in Systembibliotheken residieren, die beim Laden automatisch benutzt werden. Benutzerprogramme mit gleichen Namen haben beim Laden Vorrang, da der Lader die Systembibliotheken standardmäßig als letzte durchsucht. Werden die Systemroutinen gewünscht, obwohl in anderen beim Laden angegebenen Bibliotheken Routinen gleichen Namens vorhanden sind, so ist durch eine geeignete Reihenfolge sicherzustellen, daß die Systemroutinen verwendet werden (==> 3.3.3). Die in diesem Abschnitt angegebenen Routinen residieren in $SCILIB.

Die folgende Übersicht enthält die wichtigsten Routinen. Es ist zu beachten, daß bei neueren Betriebssystemversionen neue Routinen hinzukommen können. Andererseits verlieren unter Umständen einige Routinen durch verbesserte Fähigkeiten des Übersetzers CFT ihre Berechtigung (augenblicklich COS 1.15 und CFT 1.15). Die Bibliothek $SCILIB steht nicht nur an der CRAY, sondern sowohl an der NOS/BE-Anlage wie auch der MVS-Anlage des ZIB zur Verfügung, womit die Portabilität der optimierten CRAY-Programme sichergestellt ist.

Teilweise sind die Aufrufe mit Parametern angegeben. Dabei bedeuten:

N	Vektorlänge
SX, SY	Vektoren vom Typ REAL
INCX, INCY	Inkremente für SX bzw. SY (Für aufeinanderfolgende Elemente =1)
SA, R	REAL Variable
I	INTEGER Variable

Grundoperationen der linearen Algebra

Einige Routinen besitzen ein Äquivalent für komplexe Arithmetik. Diese Namen wurden hier nicht aufgenommen.

Name	Funktion
SASUM	Summe der Absolutbeträge eines Vektors Aufruf: SASUM(N,SX,INCX)
SAXPY	Multiplikation einer Konstanten mit einem Vektor und Addition eines anderen Vektors (Triade). Y= a*X + Y Aufruf: CALL SAXPY(N,SA,SX,INCX,SY,INCY)
SDOT	Skalarprodukt zweier Vektoren Aufruf: SDOT(N,SX,INCX,SY,INCY)
SNRM2	Euklidische Norm eines Vektors Aufruf: SNRM2(N,SX,INCX)
SSUM	Summe der Elemente eines Vektors Aufruf: SSUM(N,SX,INCX)

Weitere Routinen bieten Givens-Rotation und Basisoperationen für die LU-Zerlegung und die Lösung dünnbesetzter linearer Gleichungssysteme.

Lineare Rekursionen

Name	Funktion
FOLR FOLRN	Auflösung linearer Rekursionen erster Ordnung vom Typ C(I) = -A(I)*C(I-1) + B(I) Kann zur Berechnung des Hornerschemas benutzt werden.
SOLR SOLRN	Auflösung linearer Rekursionen zweiter Ordnung vom Typ C(I+2)= A(I)*C(I+1) + B(I)*C(I)
SOLR3	Auflösung linearer Rekursionen zweiter Ordnung mit drei Gliedern vom Typ C(I)= C(I) + A(I-2)*C(I-1) + B(I-2)*C(I-2)

Numerische Algorithmen

Die mit *) bezeichneten Routinen sind neben der Dokumentation im Library Reference Manual zusätzlich durch 'Technical Notes', herausgegeben von der Firma CRAY, erläutert.

Name	Funktion
MINV*)	Lösung eines linearen Gleichungssystems mit mehreren rechten Seiten. Berechnung der Inversen und der Determinante einer quadratischen, regulären Matrix.
MXM*) MXMA*)	Produkt Matrix*Matrix
MXV	Produkt Matrix*Vektor

MXVA

FILTERG*) FILTERS*) OPFILT*)	Berechnungen für lineare Filterverfahren
CFFT2*) RCFFT2*) CRFFT2*)	Fourier Transformationen (Analyse und Synthese) für reelle und komplexe Argumente

GATHER und SCATTER

Wenn Vektoren zu verarbeiten sind, die nicht gleichabständig im Speicher liegen, sondern über ein Indexfeld indiziert werden, helfen diese optimierten Routinen beim Umspeichern. Das lohnt sich bei großen Vektorlängen und/oder, wenn der umgespeicherte Vektor mehrfach in einem vektorisierbaren Ausdruck verwendet wird.

Hardware zur Unterstützung von GATHER und SCATTER ist am ZIB nicht installiert. Daher sollten Vektoren möglichst immer gleichabständig im Speicher liegen.

Name	Funktion

GATHER	Sammeln von Vektorelementen mit indizierter Indizierung. Diese Routine ersetzt FORTRAN-Code der Form
	`DO 1 I = 1,N`
	`1  A(I) = B(INDEX(I))`

SCATTER	Verteilen von Vektorelementen mit indizierter Indizierung. Diese Routine ersetzt FORTRAN-Code der Form
	`DO 1 I = 1,N`
	`1  A(INDEX(I)) = B(I)`

Such- und Sortierroutinen

Name	Funktion
ISMIN ISMAX	Indexbestimmung des kleinsten bzw. größten Vektorelements Aufruf: ISMxx (N,SX,INCX)
ISAMIN ISAMAX	Indexbestimmung des betragsmäßig kleinsten bzw. größten Vektorelements Aufruf: ISAMxx (N,SX,INCX)
IILZ	Anzahl der Werte =0 (oder = .FALSE.) vor dem ersten Wert ≠0
ILLZ	Anzahl der Werte >=0 (oder = .FALSE.) vor dem ersten Wert < 0
ILSUM	Anzahl der Werte <0 (oder = .TRUE.) in einem Vektor
ISRCHxx	Suchen der Position des ersten Elements eines Vektors, das zu einem vorgegebenen Wert in der Beziehung .EQ., .NE., .LT., .LE., .GT. oder .GE. steht.
WHENxx	Suchen der Anzahl und der Positionen aller Elemente eines Vektors, die zu einem vorgegebenen Wert in der Beziehung

.EQ., .NE., .LT., .LE., .GT. oder .GE. stehen.

ORDERS — Bestimmen der Sortierindizes für Records fester Länge, die in einem zweidimensionalen Feld gespeichert sind.

LINPACK

Von dem Unterprogrammpaket LINPACK stehen alle einfach genauen Routinen in einer für CRAY optimierten Form zur Verfügung (1. Buchstabe der Routinennamen ist S). LINPACK ist ein Paket zur Analyse und Lösung linearer Gleichungssysteme. Die Dokumentation befindet sich in folgendem Buch:

LINPACK User's Guide
J.J. Dongarra, J.R. Bunch, C.B. Moler, G.W. Stewart
SIAM, 1979
ISBN 0-89871-172-X

Folgende Matrizenklassen (bzw. Zerlegungen) werden behandelt (die beiden angegebenen Buchstaben stehen an 2. und 3. Stelle im Routinennamen):

GE	General
GB	General band
PO	Positive definite
PP	Positive definite packed
PB	Positive definite band
SI	Symmetric indefinite
SP	Symmetric indefinite packed
TR	Triangular
GT	General Tridiagonal
PT	Positive definite tridiagonal
CH	Cholesky decomposition
QR	Orthogonal triangular decomposition
SV	Singular value decomposition

EISPACK

Von dem Unterprogrammpaket EISPACK stehen die Routinen für reelle Matrizen zur Verfügung, die das Eigenwertproblem

$$Ax = \lambda x$$

lösen. Diese Routinen sind für die CRAY Anlage optimiert. EISPACK ist dokumentiert in:

B.T. Smith, J.M. Boyle et al.
Matrix Eigensystem Routines -- Eispack Guide
Volume 6, second edition
und
B.S. Garbow, J.M. Boyle et al.
Matrix Eigensystem Routines -- Eispack Guide extension
Volume 51

Beide Bände sind im Springer Verlag in der Reihe "Lecture Notes in Computer Science" erschienen.

Folgende Matrizenklassen werden behandelt:

General
Symmetric
Symmetric band
Symmetric packed
Symmetric tridiagonal
Special tridiagonal
Upper Hessenberg

10. Ein-/Ausgabeoptimierung

10.1 Allgemeines

Der CRAY-Rechner besitzt zwei sehr schnelle CPU's, die ca. 210 000 000 Gleitkommaoperationen in der Sekunde durchführen kann. Bei der Ausführung von vielen Programmen besteht jedoch der Engpaß nicht bei der CPU-Leistung, sondern im Bereich der Datenmengen und ihrer Verwaltung. Daher wird häufig die Ausführungszeit eines Jobs nicht so sehr von der Inanspruchnahme der CPU bestimmt als viel mehr von den Ein- Ausgabeanforderungen an externe Speicher (Ein-/Ausgabe-Operationen). Diese Tatsache wird besonders deutlich, wenn man sich klar macht, daß ein Plattenzugriff im allgemeinen etwa 30 ms kostet und in dieser Zeit etwa 3 Millionen Gleitkommaoperationen durchgeführt werden können. Häufig muß die CPU auf den Abschluß eines Ein-/Ausgabevorgangs warten. Im Normalfall ist die CPU während dieser Wartezeit nicht untätig, sondern wird von einem anderen Job belegt. Sind jedoch mehrere Jobs aktiv, die häufig Zwischenergebnisse auf Magnetplatten speichern, so kann es zu einer gegenseitigen Behinderung kommen, da die Anlage nur eine begrenzte Zahl von Laufwerken und Steuereinheiten besitzt. Unter den in der Ausführungsphase befindlichen Jobs ist unter Umständen keiner, der die CPU nutzen kann, weil alle auf nicht abgeschlossene Ein- Ausgabevorgänge warten. Dies führt zu einer insgesamt ungünstigen Auslastung der Anlage; die abgegebene Leistung wird vermindert.

Aus diesen Gründen muß neben den Vektorisierungsverfahren für die CPU auch den verwendeten Ein-/Ausgabe-Methoden besondere Aufmerksamkeit gewidmet werden. Hier soll ein Überblick über die Möglichkeiten an der CRAY in diesem Bereich gegeben werden.

Im Einzelfall kann die Konzipierung oder Optimierung eines Programms im Ein-/Ausgabe-Bereich eine schwierige Aufgabe bedeuten. Es kann daher durchaus sinnvoll sein, sich bei größeren Vorhaben vorher an die Beratung der Rechenzentren zu wenden.

10.2 Einige Begriffe

Die kleinste Einheit, die zwischen einer Magnetplatte und dem Arbeitsspeicher transportiert wird, ist ein Sektor. Ein Sektor besteht aus 512 Worten zu je 64 Bit. Auf der Platte vom Typ DD29 sind in einer Spur (Track) 18 Sektoren, auf der Platte vom Typ DD49 sind es 42 Sektoren. Die meisten Angaben zur Größe von Dateien erfolgen in Sektoren.

Ein Satz (record) ist die vom Benutzerprogramm in einem Befehl angesprochene Datenmenge.

Request ist eine Anforderung des Betriebssystems, daß Daten übertragen werden sollen.

Ein Ein-Ausgabepuffer ist ein Bereich im Arbeitsspeicher, der zur Zwischenspeicherung beim Transport vom oder zum externen Speichermedium benutzt wird. Durch die Pufferung werden die Ein- Ausgabeanforderungen des Benutzerprogramms an die Erfordernisse des Betriebssystems angepaßt. Die Puffergröße kann im allgemeinen durch den Benutzer festgelegt werden und ist bei Optimierungsmaßnahmen in Abhängigkeit von den Satzlängen und der Arbeitsweise (sequentiell oder wahlfrei) zu setzen. Die Voreinstellung beträgt im allgemeinen 4 Sektoren.

Sequentielle Arbeitsweise bedeutet, daß die Sätze der externen Dateien vom Benutzerprogramm in aufsteigender Reihenfolge ohne Lücken geschrieben und gelesen werden. Bei sequentieller Arbeitsweise sollte die Puffergröße mindestens

so groß gewählt werden, daß zwei Sätze in den Puffer passen.

Wahlfreie (Random)-Arbeitsweise bedeutet, daß der Zugriff auf die Sätze in den externen Dateien in beliebiger Reihenfolge erfolgt. Bei dieser Arbeitsweise sollte die Puffergröße etwa 'maximale Satzlänge + 2 Sektoren' betragen. Größere Puffer können bei zeitweise mehr sequentiell orientiertem Lesen oder Schreiben zu einer Verminderung der Ein- Ausgabeaktivitäten führen, wenn die verlangte Information schon durch den vorangegangenen Request zur Verfügung gestellt wurde. Bei reiner Arbeitsweise im wahlfreien Zugriff sind große Puffer nicht von Vorteil.

Asynchrone Arbeitsweise bedeutet, daß das Benutzerprogramm in der für die Datenübertragung benötigten Zeit weiterarbeitet, das heißt entweder Rechnungen ausführt oder Daten anderer Dateien überträgt.

Im Normalfall liegen alle Dateien im geblockten Format vor. Ungeblockt bedeutet, daß vom CRAY-internen Format der 512-Wort Blöcke mit vorangestelltem 'Block control word' abgewichen wird. Ungeblockte Verarbeitung bedeutet immmer auch 'ungepuffert' und ist immer nur für Sätze mit ganzen Vielfachen von 512 Worten möglich. Diese Arbeitsweise ist nur für die Übertragung von langen Sätze geeignet und bringt dort wegen der geringeren Verwaltungsarbeit des Systems unter Umständen Vorteile.

10.3 Angaben im Protokoll

Einige Angaben über die Ein-/Ausgabe-Aktivitäten eines Jobs lassen sich dem Benutzer-Protokoll entnehmen. Hier ist vor allem auf den Wert

```
DISK SECTORS MOVED
```

zu achten, der zur Berechnung der Ein-/Ausgabe-Zeit herangezogen wird, welche ebenfalls ausgewiesen wird. Zur Zeit gilt folgende Formel:

```
I/O-Time<sec> = 1.5*kSect (kSect= 1000 Sectors moved).
```

Sie gibt eine grobe Abschätzung für die Zeit, die von den Ein-/Ausgabe-Vorgängen beansprucht wurde. Im Einzelfall ist es aber durchaus möglich, daß die tatsächliche Belastung des Systems mit Hilfe dieser Formel nur unzureichend erfaßt ist, da noch andere Faktoren eine wichtige Rolle spielen, beispielsweise die Anforderung selbst sowie das benutzte Speichermedium und die Art und Weise der Speicherung.

Um weitere Informationen zu erhalten, sollte der Benutzer die Steueranweisung

```
OPTION,STAT=ON.
```

verwenden. Diese Anweisung bewirkt, daß für jede benutzte Datei eine Statistik ins Protokoll geschrieben wird, die als Grundlage für Optimierungsmaßnahmen benutzt werden kann (==> COS Reference Manual SR-0011). Die im Protokoll erscheinenden Angaben sollen hier kurz erläutert werden:

ldn	Datei-Name
x WORDS	Größe der Datei in Worten
x IOS	Anzahl der Job-Unterbrechungen wegen ausstehenden Ein-/Ausgabe-Vorgängen dieser Datei
x REQ	Anzahl der tatsächlichen Ein-/Ausgabe-Anforderungen
x SECTRS	Anzahl der übertragenen Sektoren
x.x SEC	Ein-/Ausgabe-Wartezeit: Zeit in Sekunden, die der Job wegen Ein-/Ausgabevorgängen für diese Datei warten mußte
ldv	Name des Gerätes, auf dem die Datei liegt

x SECTRS Größe des für die Datei reservierten Plattenspeichers in Sektoren, zugewiesen in Einheiten von 'Tracks' (gerätespezifisch).

Die Ein-/Ausgabe-Wartezeiten können für eine Datei je nach Betriebssituation stark differieren. Bei Optimierungsmaßnahmen sollte vor allem auf eine Verminderung der Anforderungen und der transportierten Sektoren geachtet werden.

10.4 Auswahl von Speichermedien

Erfolgt keine Angabe, werden Dateien auf einer Magnetplatte angelegt. Daneben bestehen noch die Möglichkeiten, das 'Buffer Memory' des Ein-/Ausgabe-Subsystems und den Arbeitsspeicher der CRAY ('Memory resident datasets') als Speichermedium einzusetzen.

Durch die Anweisung

```
DASSIGN,DN=dn,DV=n,DEF=def,.....
```

kann man Dateien bestimmten Platten zuordnen (==> Kapitel 3.4.1). Dies kann sinnvoll sein, wenn man gleichzeitig mehrere Dateien benutzt, die auf verschiedenen Platten liegen sollen, um eine gegenseitige Behinderung bei der Übertragung auszuschließen. 'n' ist eine symbolische Kanalnummer der gewünschten Platte. Dateien mit gleichen symbolischen Kanalnummern liegen auf gleichen physikalischen Geräten, Dateien mit unterschiedlichen symbolischen Kanalnummern liegen auf unterschiedlichen Geräten (sofern noch verfügbar). Die Angabe zu DEF bestimmt den Gerätetyp, der vergeben werden soll, z.B. DEF=DD29 (Transferrate 4,5 MB/s) oder DD49 (Transferrate 11 MB/s) Die tatsächlichen Namen der verwendeten Geräte findet man in der von 'OPTION,STAT=ON.' gelieferten Statistik als Device-Namen.

Das Buffer Memory des Ein-/Ausgabe-Subsystem wird benutzt, indem man

```
BMR=nn
```

in der CRAY-Jobkarte angibt, wobei nn die Anzahl der benötigten Sektoren angibt (Maximum: 1200, ca. 600 000 Worte). In der ASSIGN-Anweisung für die entsprechende Datei muß man

```
DV=BMR-0-20
```

einsetzen. Das IO-Subsystem besitzt eine Datenübertragungsrate von ca. 50 MB/s gegenüber 4,5 MB/s (DD 29) bzw. 11 MB/s (DD 49) zu Magnetplatten.

Kleinere Dateien kann man unter Umständen durch

```
ASSIGN,...,MR,BS=blk,...
```

im Arbeitsspeicher (Memory Resident) halten. Die Puffergröße muß so festgesetzt werden, daß die gesamte Datei und zusätzlich 1 Sektor in den Puffer passen. Bei Überschreitung des im Buffer-Memory oder im Arbeitsspeicher zur Verfügung stehenden Speichers wird automatisch auf Magnetplatten ausgelagert.

10.5 Charakterisierung einiger Ein-/Ausgabe-Verfahren

Sequentielle Dateibearbeitung mit READ/WRITE war bereits im FORTRAN Standard von 1966 enthalten. Sie sollte bevorzugt bei Dateien mit kleinerem Ein-/Ausgabe-Aufkommen eingesetzt werden oder wenn Kompatibilität mit anderen Rechnersystemen wichtig ist. Für effiziente sequentielle Verarbeitung stehen die BUFFERIN/BUFFEROUT Befehle zu Verfügung (==> CFT Reference Manual SR-0009). Sie ermöglichen auch asynchrone Verarbeitung und wahlfreien Zugriff mit SETPOS (==> Library Reference Manual SR-0113).

Direkt-Zugriff (Direct Access) ist die in FORTRAN 77 standardisierte Möglichkeit, Dateien im wahlfreien Zugriff zu bearbeiten (OPEN(...,ACCESS='DIRECT',...). Von dieser Möglichkeit sollte an der CRAY nur sparsamer Gebrauch gemacht werden, da diese Methode zur Zeit ineffizient implementiert ist. Statt dessen sollten bevorzugt die READMS/WRITMS Befehle benutzt werden, die in mancher Hinsicht auch ein erweitertes Leistungsspektrum bieten. Die Puffergröße sollte entsprechend den Erfordernissen der überwiegenden Zugriffsart (wahlfrei oder sequentiell) und der Satzlängen über die Routine WOPEN festgelegt werden. Dabei wird der Benutzer durch Statistiken unterstützt, die vom READMS/WRITMS-System geliefert werden. Wegen ihrer zentralen Bedeutung wird die Arbeitsweise mit diesen Routinen im nächsten Abschnitt eingehender beschrieben.

Soll ein großes Feld, unter Umständen unter Benutzung von wechselnden Satzlängen, auf einem externen Medium verwaltet werden, so stehen die GETWA/PUTWA Routinen zur Verfügung (==> Library Reference Manual SR-0113).

Für ungeblockte Verarbeitung und wahlfreien Zugriff existieren die READDR/WRITDR Routinen, die sich besonders für die Übertragung großer Datenmengen eignen. (Dokumentation ==> Library Reference Manual SR-0113 und nächsten Abschnitt).

Für READMS/WRITMS, GETWA/PUTWA und READDR/WRITDR gibt es jeweils Versionen oder Ergänzungen, die asynchrone Verarbeitung erlauben.

Weitere Eigenschaften dieser verschiedenen Ein-/Ausgabe-Methoden, außerdem Hinweise zu ihrer Anwendung und Dokumentation findet man mit Hilfe des Kommandos

```
DOC,CRAY,IO.
```

am Vorrechner.

10.6 Routinen für wahlfreie Ein-/Ausgabe (READMS/WRITMS)

Diese Routinen sollten an der CRAY bevorzugt für die Behandlung von Aufgaben eingesetzt werden, die häufigen wahlfreien Zugriff zu einer externen Datei erfordern. Sie sind im allgemeinen wesentlich effizienter als die vom FORTRAN 77 Standard angebotenen READ/WRITE-Befehle im ACCESS='DIRECT'-Modus. READMS/WRITMS ist kompatibel mit den auf CDC Rechnern vorhandenen Routinen gleichen Namens.

Zur Übertragung von großen Datenmengen, oder wenn sich die Pufferung als ineffizient herausstellt, sollten die READDR/WRITDR-Routinen eingesetzt werden. Sie besitzen im wesentlichen die gleiche Parameterversorgung wie die unten beschriebenen READMS/WRITMS-Routinen.

READMS/WRITMS wird hier nur in soweit behandelt, daß die vom FORTRAN 77 Standard mit READ/WRITE-Befehlen gebotenen Möglichkeiten genutzt werden können. READMS/WRITMS erlaubt allerdings nur unformatiertes Lesen und

Schreiben.
WOPEN setzt die Puffergröße fest und ist vor OPENMS aufzurufen:

```
CALL WOPEN(dn,ibuff,istats)
```

Parameter:

dn	Spezifizierung der Datei. Entweder Kanalnummer als INTEGER oder Dateiname als Hollerith-Konstante. Beispiele: 7 entspricht FT07; 5HFILEA entspricht FILEA
ibuff	Größe des Puffers in Blöcken zu 512 Worten als INTEGER Beispiel: 16, es wird ein Puffer von 16*512 Worten eingerichtet.
istats	Anforderung einer Statistik vom GETWA/PUTWA-Paket, das intern von READMS/WRITMS aufgerufen wird. Ist istats verschieden von Null, wird die Statistik auf die Datei $STATS geschrieben (INTEGER-Größe).

OPENMS eröffnet eine READMS/WRITMS-Datei:

```
OPENMS(dn,index,length,it)
```

Parameter:

dn	Dateiname: ==> WOPEN.
index	Feld vom Typ INTEGER, das vom Benutzer dem READMS/WRITMS System als Arbeitsspeicher zur Verfügung gestellt werden muß. Länge: ==> Parameter length.
length	Maximale Zahl der Sätze in der Datei (INTEGER).
it	0, wenn die Sätze über ihre Nummer angesprochen werden (andere Werte erlauben die Identifizierung über einen alphanumerischen Namen und asynchrone Arbeitsweise).

READMS liest einen Satz:

```
READMS(dn,record,nwords,irec)
```

Parameter:

dn	Dateiname: ==> WOPEN.
record	Feld der Länge >=nwords, wird durch Lesevorgang gefüllt.
nwords	Anzahl der zu lesenden Worte (INTEGER)
irec	Nummer des zu lesenden Satzes (INTEGER)

WRITMS schreibt einen Satz:

```
WRITMS(dn,record,nwords,irec,iflag,dummy)
```

Parameter:

dn	Dateiname: ==> WOPEN.

record	Feld der Länge >=nwords, wird auf die Datei geschrieben.
nwords	Anzahl der zu schreibenden Worte (INTEGER).
irec	Nummer des zu schreibenden Satzes (INTEGER).
iflag	In Abhängigkeit vom Vorhandensein und der physikalische Länge des Satzes in der Datei soll der neue Satz den alten Satz entweder überschreiben oder nach EOD geschrieben werden. -1 kann im Standardfall angegeben werden, weitere zulässigen Werte: ==> Library Reference Manual SR-0113 Seite 12-56.
dummy	Variable ohne Bedeutung. (Kann bei späteren Implementierungen zur Kennzeichnung von Subindizes benutzt werden.)

CLOSMS schließt eine READMS/WRITMS-Datei

```
CALL CLOSMS(dn)
```

Parameter:

dn	Dateiname: ==> WOPEN.

Allen hier angegebenen Parameterlisten kann ein weiterer Parameter ierr hinzugefügt werden, der bei Auftreten eines Fehlers während der Ausführung der entsprechenden Routine auf einen Wert verschieden von Null gesetzt wird.

Beispiel für READMS/WRITMS:

```
A9999,STCRY.
JOB,T=2,MFL=200000.
ACCOUNT,AC=xxxxxxxxx,APW=xxxxxxxx.
CFT.
*PRINT IO STATISTIC TO LOGFILE
OPTION,STAT=ON.
LDR.                    besser: SEGLDR,GO.
*PRINT STATISTIC FILE
REWIND,DN=$STATS.
COPYD,I=$STATS.
/EOF
      PROGRAM MSTEST
C
      DIMENSION INDEX(100), REC(2000)
C
C     CREATE MS-FILE
C
C     SET BUFFER SIZE
      CALL WOPEN(1,32,1)
C     OPEN FILE
      CALL OPENMS(1,INDEX,100,0)
C     WRITE 100 RECORDS WITH INCREASING LENGTH AND RECORD NUMBER
      DO 100 I=1,100
         CALL WRITMS(1,REC,1000+I*10,I,-1,DUMMY)
 100  CONTINUE
C
C
C     READ 1000 RECORDS BY RANDOM ACCESS
      DO 200 J=1,1000
```

```
        IREC= RANF()*100  + 1
        CALL READMS(1,REC,1000+IREC*10,IREC)
 200  CONTINUE
C
C
C     WRITE 1000 RECORDS BY RANDOM ACCESS
      DO 300 J=1,1000
        IREC= RANF()*100  + 1
        CALL WRITMS(1,REC,1000+IREC*10,IREC,-1,DUMMY)
 300  CONTINUE
C     CLOSE FILE
      CALL CLOSMS(1)
      END
```

In diesem Beispiel wird der Puffer für die Datei mit der Kanalnummer 1 auf 32 Sektoren gesetzt (WOPEN). Die Datei FT01 wird als MS-Datei eingerichtet mit maximal 100 Sätze (OPENMS). 100 Sätze werden sequentiell mit unterschiedlicher Satzlänge auf die Datei geschrieben (das Feld REC wird der Einfachheit halber in diesem Demonstrationsprogramm nicht mit Werten gefüllt). Anschließend werden 1000 Sätze in wahlloser Reihenfolge wieder gelesen. Die im System vorhandene FORTRAN Function RANF liefert dafür Zufallszahlen im Bereich 0 bis 1, die auf den Bereich 1 bis 100 transformiert werden. Zuletzt werden 1000 Sätze ebenso in wahlloser Reihenfolge neu geschrieben. Die Satzlängen sind identisch mit den ursprünglichen Längen.

Im folgenden werden die vom READMS/WRITMS erzeugten Statistiken und ein Auszug aus dem Protokoll wiedergegeben. Da READMS/WRITMS intern das WA-Paket benutzt, erscheint zunächst die Statistik für GETWA/PUTWA, dann die für READMS/WRITMS. Normalerweise ist das Druckbild auf 132 Schreibstellen in einer Zeile ausgerichtet. Dieses Format mußte hier aufgebrochen werden.

GETWA/PUTWA Statistik:

```
 SUMMARY FOR W.A. DATASET     FT01::::  BUFFERS USED      =          32
          GETS          =         1000  PUTS              =        1103
  APUTWA CALLS          =            0  APUTWA OVERFLOWS  =           0
  WFIND SUMMARY: HITS   =          140  MISSES            =        1702
  WORDS READ            =      1511940  WORDS WRITTEN     =     1655268
  DISK READS            =         1878  DISK WRITES       =         962
  DISK WORDS READ       =      3566080  DISK WDS WRITTEN  =     2046976
  AVER PHYS DISK READ   =         1898  AV PHYS DISK WRIT=        2127
  TOTAL ACCESS TIME     =       59.429  AVER ACCESS TIME  =      20.926
  IN SECONDS.                           IN MILLISECONDS.

  TOTAL ACCESSES    =         2103
  FINDS             =            0
  SEEK OVERFLOWS    =            0
  PARTIAL HITS      =          273
  TOTAL WORDS       =      3167208
  BUFFER FLUSHES    =          170
  TOTAL DISK XFERS  =      5613056
  BUFFER BONUS %    =        56.00
  EOI BLOCK NUMB.   =          296
```

READMS/WRITMS Statistik

```
  READMS/WRITMS SUMMARY FOR DATASET FT01::::
  TOTAL ACCESSES =            2100  READS =                1000
                                    WRITES =               1100

  SEQUENTIAL READS =            11  SEQUENTIAL WRITES =     106
  REWRITES IN PLACE =         1000  WRITES TO EOI =         100
  TOTAL WORDS MOVED =      3166590
  MINIMUM RECORD  =           1010  MAXIMUM RECORD  =      2000

  TOTAL ACCESS TIME =       59.818  SECONDS.
  AVERAGE ACCESS TIME =     28.485  MILLISECONDS
```

Protokollauszug: (Anmerkung: es handelt sich um die alte Form (COS 1.14)!)

```
USER    UT009 - W.A. FILE OPENED. FILE = FT01  EOI= -1 BUFFER=0000064000
USER    UT009 - W.A. FILE CLOSED. FILE = FT01
USER        JOB NAME -                         AnnnYBX
USER        USER NUMBER -                      ZZZname
USER        TIME EXECUTING IN CPU -            0000:00:00.6514
USER        TIME WAITING TO EXECUTE -          0000:00:06.1570
USER        I/O TIME -                         0000:00:40.4770
USER        TIME WAITING IN INPUT QUEUE -      0000:00:00.0042
USER        MEMORY * CPU TIME (MWDS*SEC) -              0.03361
USER        MEMORY * I/O TIME (MWDS*SEC) -              1.90974
USER        MINIMUM JOB SIZE (WORDS) -             21504
USER        MAXIMUM JOB SIZE (WORDS) -             84480
USER        MINIMUM FL (WORDS) -                   17920
USER        MAXIMUM FL (WORDS) -                   80896
USER        MINIMUM JTA (WORDS) -                   3584
USER        MAXIMUM JTA (WORDS) -                   3584
USER        DISK SECTORS MOVED -                   11438
USER        USER I/O REQUESTS -                     2915
USER        OPEN CALLS -                              30
USER        CLOSE CALLS -                             29
USER        MEMORY RESIDENT DATASETS -                 0
USER        TEMPORARY DATASET SECTORS USED -           0
USER        PERMANENT DATASET SECTORS ACCESSED -      97
USER        PERMANENT DATASET SECTORS SAVED -          0
USER        SECTORS RECEIVED FROM FRONT END -          0
USER        SECTORS QUEUED TO FRONT END -              0
USER        RESOURCE TIME -                    0000:00:10.0272
EXP     SY005 - $IN     DD-A1-24     512W     1R      1S     .013 SEC
EXP     SY005 - $BLD    DD-A1-21     201W     2R      2S     .082 SEC
EXP     SY005 - FT01    DD-A1-24      296S  1878RR    6965SR   962WR    3998SW
                                                                        (49.32S
EXP     SY005 - $STATS  DD-A1-24        1S     1RR       1SR     1WR       1SW
                                                                         (.01S
```

Auf einige für Optimierungsmaßnahmen besonders wesentliche Angaben soll hier hingewiesen werden:

WA-Statistik:

In 140 Fällen 'Hits' war die verlangte Information im Puffer. In diesen Fällen hat also kein Plattenzugriff stattgefunden. Dagegen stehen 1702 'Misses' und 273 'Partial Hits'.
Der 'Buffer Bonus' ist das Verhältnis zwischen 'Total Words' und 'Total Disk

Xfers', das heißt zwischen den vom Benutzerprogramm angeforderten Worten und den tatsächlich transportierten Worten.

Dieses Verhältnis ist im Beispiel mit 56% relativ ungünstig. Das liegt hier an dem hohen Anteil von WRITE-Operationen (WRITMS), die einen erheblich größeren Aufwand erfordern als READ-Befehle. Schreib- wie auch Leseoperationen können nämlich vom System nur so ausgeführt werden, daß vollständige Sektoren von je 512 Worten bewegt werden. Jedem WRITE geht daher intern ein Lesen voraus, um beim Schreibvorgang sicherzustellen, daß Information durch das Schreiben von ganzen Sektoren nicht vernichtet wird. Verdoppelt man im Beispiel die Puffergröße auf 64 Sektoren, das heißt auf 22% der Dateigröße, so steigt der Bufferbonus nur um 7%. Dieser geringe Anstieg ist darin begründet, daß die Anzahl der 'Hits' nicht wesentlich ansteigt, das heißt, das Programm arbeitet stark 'wahlfrei-orientiert'.

Der Buffer-Bonus ist ein Maß für die Effizienz der Pufferung. Werte von 90 bis 100% können als normal angesehen werden. Dabei ist der Anteil der Schreibbefehle und die Arbeitsweise des Programms (mehr sequentiell oder mehr wahlfrei) zu berücksichtigen. Bei Werten unter 90% empfiehlt sich ein Versuch mit READR/WRITDR. Im allgemeinen (so im Beispiel) ist dafür nur erforderlich, auf den Aufruf von WOPEN zu verzichten und OPENMS, READMS und WRITMS durch OPENDR, READDR und WRITDR zu ersetzen. Die Parameterversorgung der Routinen kann beibehalten werden, wenn man sich auf den oben angegebenen Teilmenge der Nutzungsmöglichkeiten beschränkt.

Als generelle Faustregeln für den Umgang mit MS- und WA-Dateien bleibt festzuhalten, daß größere Puffer zu einer Verminderung der Ein-/Ausgabe-Operationen führen. Daher sollte im Zweifelsfall der Puffer immer möglichst groß angelegt werden.

Nutzung des Haupspeichers

Am besten ist es - falls die drei Millionen Worte Arbeitsspeicher hierfür ausreichen - daß die Datei völlig hineinpaßt: damit entfällt jede Ein-/Ausgabe. Dieses Ablegen einer Datei im Arbeitsspeicher geschieht bei MS-Dateien *nicht* mit ASSIGN (==> 3.4.1), sondern - genau wie bei MSTEST - durch CALL WOPEN vor dem CALL OPENMS. Als Pufferlänge in Blöcken im zweiten Argument von WOPEN wählt man hierfür

```
KBLOCKS=(Anzahl der Worte der Datei + 511)/512 + 1.
```

Sehr zu empfehlen ist auch die permanente MS-Datei mit voller (Puffer-) Kopie im Arbeitsspeicher. Auch hier spart man alle Ein-/Ausgabe bis auf die unbedingt notwendige: beim OPENMS wird die permanente MS-Datei in der Puffer gelesen, beim CLOSMS der aktuelle Stand der Puffer-Kopie auf die permanenten Datei zurückgeschrieben, d.h. nach dem CLOSMS ist die permanente Datei ohne jeden weiteren Aufruf auf dem neuesten Stand.

Falls man in einem längeren Lauf mit permanenter MS-Datei FT01 Zwischenstände sichern möchte, erreicht man dies durch die Sequenz

```
:
:
:
CALL CLOSMS(1)
CALL WOPEN(1,KBLOCKS,1)
CALL OPENMS(1, ...)
```

an der entsprechenden Stelle des Programms.

Zum ersten Anlegen der permanenten MS-Datei auf Steuerkarten-Ebene genügt ein

```
SAVE,DN=FT01,PDN=ZWISCHEN.
```

nach der Karte LDR.; beim Fortsetzen ein

```
ACCESS,DN=FT01,PDN=ZWISCHEN,UQ.
```

vor der Karte LDR. (UQ gibt die Erlaubnis, die permaneneten Datei zu überschreiben).

10.7 Routinen für wahlfreie Ein-/Ausgabe sehr großer Satzlängen (MSIO)

MSIO ist ein Paket zur Optimierung der Übertragungszeit von Ein-/Ausgaben mit sehr langen Sätzen, wobei die Sätze unterschiedliche Längen haben können. Auf die Sätze kann wahlfrei zugegriffen werden. MSIO ist funktionskompatibel mit READMS/WRITMS. MSIO sollte nur für solche Dateien eingesetzt werden, bei denen die jeweils zu übertragende Information sehr groß ist (ab ca. 40 Sektoren pro Satz).

Zur Erhöhung der Transferleistung zu und von den Platten zerlegen MSIO-Ein-/Ausgaberoutinen die jeweils zu übertragende Information in eine gewisse Anzahl gleichgrosser Teile, die asynchron, d.h. zeitlich parallel, auf bzw. von verschiedenen Platten übertragen werden (Striping). Dabei wird jedem Teil der Information ein Datenstrom und ein Plattenlaufwerk (bzw. der Pufferspeicher des E/A-Subsystems) fest zugeordnet. Die wählbare Anzahl verwendeter Datenströme wird durch den Parameter IST bei Eröffnen des zugehörigen (logischen) Datasets bzw. 'unit number' mit MSOPEN festgelegt.

MSIO enthält z.Zt. folgende Unterprogramme :

```
MSOPEN (NRTAPE,INDEX,LENGTH,IST,INC,IERR)
MSINDX (NRTAPE,INDEX,LENGTH,IERR)
MSWRIT (NRTAPE,A,N,POS,RRFLAG,IERR)
MSREAD (NRTAPE,A,N,POS,IERR)
MSCLOS (NRTAPE,IERR)
```

MSIO arbeitet nur mit Scratch-Dateien, d.h. nach Aufruf von MSCLOS ist die geschriebene Information verloren (ggf. vor Aufruf von MSCLOS eine Kopie mit Standartsprachmitteln anlegen!). In der jetzigen Version können gleichzeitig maximal 7 logische Dateien (unit numbers) bearbeitet werden.

Aufgrund der aktuellen Konfiguration der Berliner CRAY sind folgende Werte für den Striping-Parameter IST möglich:

```
-6 <= IST <= 7.
```

Das Beispielprogramm aus Abschnitt 10.6 läßt sich bei Einsatz von MSIO wie folgt realisieren:

Beispiel:

```
1     1.        PROGRAM MSTEST
2       C
3     2.        DIMENSION INDEX (100), REC(2000)
4       C
5       C       CREATE MS-FILE
6       C
7       C       SET STRIPING PARAMETER
8       C
```

```
9     3.        IST = - 3
10          C
11          C     OPEN FILE
12          C
13     4.         CALL MSOPEN (1,INDEX,100,IST,'42'L)
14          C
15          C     WRITE 100 RECORDS WITH INCREASING LENGTH AND RECORD NUMBER
16          C
17     5.         DO 100 I=1,100
18     6.            CALL MSWRIT (1,REC,1000+I*10,I,-1,DUMMY)
19     7.   100   CONTINUE
20          C
21          C
22          C     READ 1000 RECORDS BY RANDOM ACCESS
23          C
24     8.         DO 200 J=1,1000
25     9.            IREC = RANF()*100+1
26    10.            CALL MSREAD (1,REC,1000+IREC*10,IREC)
27    11.   200   CONTINUE
28          C
29          C
30          C     WRITE 1000 RECORD BY RANDOM ACCESS
31          C
32    12.         DO 300 J=1,1000
33    13.            IREC = RANF()*100+1
34    14.            CALL MSWRIT (1,REC,1000+IREC*10,IREC,-1,DUMMY)
35    15.   300   CONTINUE
36          C
37          C     CLOSE FILE
38          C
39    16.         CALL MSCLOS (1)
40    17.         END

 CRAY X-MP/24    KONRAD-ZUSE-ZENTRUM FUER INFORMATIONSTECHNIK BERLIN 05.05.87

        CRAY OPERATING SYSTEM           COS 1.15  ASSEMBLY DATE 27.02.87


JOB,T=2,MFL=200000,JN=AxxxYJD.FC=YES
ACCOUNT,AC=,APW=,US=,UPW=.
CFT.
CF000 - CFT VERSION -   24.03.87 1.15BF2
CF998 - CFT 1.15 BF2 ED 257.K02
CF001 - COMPILE TIME =    0.0214 SECONDS
CF002 -       40 LINES,       17 STATEMENTS
CF003 -    67050 WORDS,    10444 I/O BUFFERS USED
OPTION,STAT=ON.
ACCESS,DN=TUBLIB,OWN=TUB.
PD000 - PDN = TUBLIB           ID =           ED =    8  OWN = TUB
PD000 - ACCESS  COMPLETE
SEGLDR,GO,CMD='LIB=TUBLIB'.
SY005 - BRLPROC       921 WRDS,    1 IOS,      1 REQ,       1 SECTRS,   .071 SEC
SY005 - SEGLDR      52351 WRDS,   25 IOS,      3 REQ,     103 SECTRS,   .381 SEC
SG000 - SEGLDR VERSION 2.0  - 04.05.87
SG001 - BEGIN EXECUTION
MS000 - USING MS-ROUTINES ON TAPE     1 IST= -3
MS10C - CLOSING MS-TAPE     1
MS10C -  2100RC               147S  1000RR    1100WR  3093SD    7254ST
SY005 - ZZZZMA1     75264 WRDS, 1337 IOS,   2100 REQ,    3093 SECTRS, 18.421 SEC
      - D4-A1-30      168 SECTRS READ:      1014 REQ,    1472 SECTRS, 15.834 SEC
```

```
      -                                WRITE:     1113 REQ,    1621 SECTRS,  4.446 SEC
SY005 - ZZZZMB1     75264 WRDS,  971 IOS,    2100 REQ,    3093 SECTRS,  3.018 SEC
      - D4-A1-31      168 SECTRS READ:      1014 REQ,    1472 SECTRS, 12.933 SEC
      -                                WRITE:     1113 REQ,    1621 SECTRS,  1.926 SEC
SY005 - ZZZZMC1     75264 WRDS, 1080 IOS,    2100 REQ,    3093 SECTRS,  8.176 SEC
      - D4-A1-32      168 SECTRS READ:      1026 REQ,    1472 SECTRS, 17.325 SEC
      -                                WRITE:     1127 REQ,    1621 SECTRS,  6.760 SEC
SY005 - $IOLIB      31633 WRDS,   26 IOS,       7 REQ,      99 SECTRS,   .272 SEC
SY005 - $UTLIB      84310 WRDS,   53 IOS,       8 REQ,     216 SECTRS,   .776 SEC
SY005 - $SYSLIB     33361 WRDS,   34 IOS,      12 REQ,     136 SECTRS,   .323 SEC
SY005 - $ARLIB      16577 WRDS,   14 IOS,       4 REQ,      57 SECTRS,   .109 SEC
SY005 - $FTLIB       1425 WRDS,    2 IOS,       2 REQ,       6 SECTRS,   .026 SEC
SY005 - $PSCLIB     13215 WRDS,    7 IOS,       1 REQ,      26 SECTRS,   .026 SEC
SY005 - $SCILIB     84709 WRDS,   43 IOS,       2 REQ,     174 SECTRS,   .270 SEC
SY005 - $ZIBLIB      2285 WRDS,    3 IOS,       2 REQ,       7 SECTRS,   .127 SEC
END OF JOB


SY005 - CHARGES     22245 WRDS,   10 IOS,       2 REQ,      44 SECTRS,   .066 SEC
    JOB NAME -                          A3xxxJD
    USER NUMBER -                       xxxxxxxxxx
    TIME EXECUTING IN CPU -             0000:00:00.7200
    TIME WAITING TO EXECUTE -           0000:00:23.5081
    I/O TIME -                          0000:00:15.7935
    TIME WAITING IN INPUT QUEUE -       0000:00:00.0186
    MEMORY * CPU TIME (MWDS*SEC) -                0.07057
    MEMORY * I/O TIME (MWDS*SEC) -                0.89320
    MINIMUM JOB SIZE (WORDS) -                  35840
    MAXIMUM JOB SIZE (WORDS) -                 197632
    MINIMUM FL (WORDS) -                        31232
    MAXIMUM FL (WORDS) -                       192512
    MINIMUM JTA (WORDS) -                        4608
    MAXIMUM JTA (WORDS) -                        5120
    DISK SECTORS MOVED -                        10529
    FSS SECTORS MOVED -                             0
    USER I/O REQUESTS -                          6387
    USER I/O SUSPENSIONS -                       3710
    OPEN CALLS -                                   29
    CLOSE CALLS -                                  28
    MEMORY RESIDENT DATASETS -                      0
```

MSIO befindet sich in der Bibliothek TUBLIB und kann von allen CRAY-Benutzern mittels der Steuerkarten

```
ACCESS,DN=TUBLIB,OWN=TUB.
LDR,LIB=TUBLIB.     besser: SEGLDR,GO,CMD='LIB=TUBLIB'.
```

in das eigene Programm eingebunden werden.

Häufig wird man ein Programm bzw. dessen Ein-/Ausgabe-Aktivität überwachen wollen. Zu diesem Zweck wurde in MSIO eine DEBUG-Möglichkeit implementiert: wird auf der Ebene der Steuersprache vor Aufruf des zu überwachenden Programms der Schalter 6 gesetzt (SWITCH,6=ON), so gibt MSIO bei jedem Schreib- bzw. Lesevorgang eine Zeile DEBUG-Information auf $OUT aus.

Copyrigth : Technische Universität Berlin, Zentraleinrichtung Rechenzentrum, 1985, Autor: R.R. Abel.
Ausführliche Informationen erhalten Sie mit DOC,CRAY,MSIO,MO=LONG.

10.8 Ersatz von FORTRAN-BACKSPACE durch Routinen mit Direktzugriff (XIO)

Die unter dem Sammelbegriff XIO beschriebenen Routinen XREWIND, XBACKSP, XWRITE und XREAD bieten die Möglichkeit, die nicht streng sequentielle unformatierte Ein-/Ausgabe von CFT-Programmen (wie. z.B. Rückwärtslesen einer Datei) durch Ausnutzung der Möglichkeiten des Direktzuriffs weitaus effizienter zu gestalten als dies die CFT-Sprachelemente REWIND, BACKSPACE, WRITE und READ erlauben. Alleinige Voraussetzung für das Ersetzen der jeweiligen Ein-/Ausgabebefehle durch Aufrufe der XIO-Routinen ist dabei, daß innerhalb der an XIO übertragenen Phase der Verarbeitung alle Sätze gleiche Länge aufweisen, was bei den in Frage stehenden Algorithmen (wie z.B. Rückwärtssubstituieren bei der Lösung eines Gleichungssystems mit einem 'Out-of-core'-Löser) häufig der Fall ist. Darüberhinaus bieten die Routinen XPUT/XGET die Möglichkeit, wahlfrei über Angabe einer Satznummer POS auf einzelne Sätze der jeweiligen Datei zuzugreifen. Einzige Einschränkung dabei ist, daß die Satzlänge konstant bleibt und der 'initial-write' mit XPUT sequentiell, d.h. Satz POS=i nach Satz POS=i-1, vorgenommen wird.

Aufruf:		CFT-Sprachelement:
CALL XREWIND	(NRT)	REWIND NRT
CALL XBACKSP	(NRT)	BACKSPACE NRT
CALL XWRITE	(NRT,A,N)	WRITE (NRT) (A(I),I=1,N)
CALL XREAD	(NRT,A,N)	READ (NRT) (A(I),I=1,N)
CALL XPUT	(NRT,A,N,POS)	WRITE (NRT,REC=POS) (A(I),I=1,N)
CALL XGET	(NRT,A,N,POS)	READ (NRT,REC=POS) (A(I),I=1,N)
CALL XIOSUB	(NRT,N)	XIO-Tape NRT im Buffermemory des IO-Subsystem mit Satz-Länge N anlegen Parameter N ist wahlfrei.
CALL XSKIPR	(NRT)	CALL SKIPR (IERROR,'DN'L,'FTNRT'L)
CALL XRELEASE	(NRT)	CALL RELEASE (IERROR,'DN'L,'FTNRT'L)
CALL XIO (0)		Statistik-Informationen über alle aktiven XIO-Tapes auf $OUT ausgeben

Parameter:

NRT	Kanalnummer als symbolische Ein-/Ausgabeeinheit; Konstante oder Variable vom Typ INTEGER
A	symbolische Adresse des Array, das ausgegeben/eingelesen wird
N	Satzlänge des Transfers als Zahl von Hauptspeicherworten
POS	Position (Nummer) des Satzes.

XIO befindet sich in der Bibliothek TUBLIB und kann von allen CRAY-Benutzern in das eigene Programm eingebunden werden mittels der Steuerkarten

```
ACCESS,DN=TUBLIB,OWN=TUB.
LDR,LIB=TUBLIB.     besser: SEGLDR,GO,CMD='LIB=TUBLIB'.
```


Ausführliche Informationen erhalten Sie mit DOC,CRAY,XIO,MO=LONG.

11. Umstellung von FORTRAN-Programmen

11.1 Umstellung von FORTRAN-Programmen CDC -> CRAY

Die geringsten Schwierigkeiten beim Übergang von CDC Cyber 170 Anlagen auf eine CRAY Anlage entstehen, wenn der FORTRAN 77 Standard (ANSI X3.9 - 1978) eingehalten wird. Es wird daher empfohlen, zunächst auf der CDC Anlage die Einhaltung dieses Standards zu überprüfen. Dies geschieht entweder durch Aufruf des FTN5 Compilers mit der ANSI Option:

```
FTN5,ANSI,...
```

oder bei der Benutzung des M77 Compilers durch die Angabe von EL=0:

```
M77,EL=0,...  .
```

Bevor das Programm zur CRAY transferiert wird, sollte man alle Befehle, die nicht dem Standard entsprechen, durch geeignete Standardsprachmittel ersetzen.

Auf einige beachtenswerte Sachverhalte soll im folgenden aufmerksam gemacht werden:

Bei der Verwendung von Direct-Access-Dateien (im FORTRAN 77 Sprachumfang enthalten) wird die Satzlänge bei CDC für unformatierte Sätze in Worten angegeben.

Beispiel:

```
OPEN(3,ACCESS='DIRECT',RECL=100)
```

Für CRAY muß die Satzlänge immer in Zeichen angegeben werden (1 Wort = 8 Zeichen). Bei unformatierten Dateien muß die Anzahl der Zeichen ein Vielfaches von 8 sein. Das obige Beispiel muß für CRAY lauten:

```
OPEN(3,ACCESS='DIRECT',RECL=800)
```

Im PROGRAM Statement werden die Angaben über Dateien und die zugehörigen Ein- und Ausgabekanäle, die bei CDC üblich sind, vom CRAY CFT-Compiler nicht ausgewertet. Daher sollte man möglichst das Standardsprachmittel OPEN benutzen. Bei der häufig benutzten Ein- und Ausgabe über INPUT und OUTPUT (CRAY: $IN und $OUT) genügt es im Normalfall, bei der READ bzw. WRITE Anweisung statt der Kanalnummer '*' einzusetzen, wie im FORTRAN 77 Standard vorgesehen. (5 für $IN und 6 für $OUT ist an der CRAY auch zulässig.)

Die (nicht Standard-) SHIFT Funktionen zur Bitmanipulation sind unterschiedlich. Neben der Tatsache, daß sie sich bei CRAY auf 64-Bit Worte beziehen (CDC 60 Bit) und CRAY eine andere interne Zeichendarstellung verwendet, ist zu beachten, daß es bei CRAY keinen Rechts-Shift mit Vorzeichenausdehnung bei negativem Zähler gibt ("arithmetischer Shift"). In diesem Zusammenhang sei auch auf die PACK/UNPACK Routinen von CRAY hingewiesen (==> Library Reference Manual).

Werden noch Hollerith-Zeichen (xH...) im Programm verwendet (möglichst durch Größen vom Typ 'Character' ersetzen), so ist darauf zu achten, daß an der CRAY Anlage nur 8 Zeichen pro Wort (= 1 Variable vom Typ REAL oder INTEGER) gespeichert werden können. Das Beispiel 8H12345678 zeigt die maximale Anzahl von Zeichen, die an der CRAY in einem Wort gehalten werden können.

Zeichenketten (CHARACTER) werden bei CRAY mit 8 Zeichen pro Wort im ASCII-Code gespeichert (8 Bit), bei CD (NOS/BE) mit 10 Zeichen pro Wort im Display-Code

(6 Bit). Bei CHARACTER*n darf bei CRAY n nicht größer als 504 werden. Weil nicht derselbe Zeichencode benutzt wird, sind fast alle Programmstücke zwischen CD und CRAY inkompatibel, welche die Intrinsic-Funktionen ICHAR und CHAR benutzen - meist wird mit Übersetzungstabellen gearbeitet. Größer-kleiner-Vergleiche von Zeichenketten fallen nur dann auf beiden Rechnern gleich aus, wenn darin entweder nur Buchstaben oder nur Ziffern vorkommen. Wenn diese Bedingung nicht erfüllt ist, sollte man die Vergleiche mit Hilfe der Intrinsic-Funktionen LGE, LGT, LLE, LLT durchführen; auf beiden Maschinen wird dann beim Vergleich die Sortierfolge ASCII benutzt. Achtung: Die Anwendung dieser Funktionen liefert auf der CDAnlage andere Ergebnisse als die der entsprechenden Vergleichsoperatoren .GE., .GT., .LE., .LT. !

Die meisten CD-Spracherweiterungen müssen vermieden werden. Sie werden zum Teil bei eingeschalteter Compileroption ANSI sichtbar. Einige grundsätzliche Regeln dazu werden bereits hier genannt:

* Initialisieren nur mit DATA.

* Arithmetische Daten nicht mit Zeichenketten (CHARACTER) initialisieren oder gleichsetzen (mit EQUIVALENCE). Im allgemeinen sind solche Anwendungen durch den fehlenden Datentyp CHARACTER in FORTRAN IV begründet, und man kann in solchen Fällen REAL*n und INTEGER*n durch CHARACTER*n ersetzen.

  Beispiele:

```
LOGICAL*1 L1(2)                -> CHARACTER*1 L1(2)
INTEGER*2 I2 /' '/             -> CHARACTER*2 I2
EQUIVALENCE(I2,L1(1)) bleibt -> EQUIVALENCE(I2,L1(1))
                               -> DATA I2/' '/
```

* Keine Sedezimalkonstanten benutzen (meist markieren sie sowieso Maschinenabhängigkeit)! Das Z-Format dagegen darf man verwenden:

  Beispiele:

```
CHARACTER*16 C
C='2000800000000000'
READ(C,'(Z16)') FLOMIN
```

Die Maschinen besitzen Unterschiede in der internen Zahlendarstellung und Arithmetik. Die wichtigsten Auswirkungen für Benutzer sind:

Durch die um 4 Bits längeren CRAY Worte erweitert sich der Darstellungsbereich für REAL Zahlen. Exponenten bis etwa +/- 2400 können dargestellt werden. Die Anzahl der Bits für die Mantisse ist gleich (48 Bit, etwa 14 Dezimalstellen). Siehe hierzu die Routine SMACH (==> 5.3.1: System-Hilfsroutinen), die die Maschinenkonstanten liefert.

Sollten Programmentscheidungen auf der Gleichheit von berechneten REAL-Werten aufbauen, so kann es durch die abweichende Arithmetik zu unterschiedlichen Ergebnissen kommen.

Die CRAY Anlage ist ein Vektorrechner. Nach dem ersten erfolgreichen Lauf an der CRAY sollte man die im Kapitel 9 (Einführung in die Optimierung der Rechenzeit) genannten Hinweise lesen, verstehen und anwenden.

11.2 Umstellung von FORTRAN-Programmen MVS -> CRAY

11.2.1 Zielsetzung der Umstellung

Im folgenden soll dargestellt werden, was bei der Umstellung von FORTRAN-Programmen von einem MVS-System zur CRAY X-MP des ZIB beachtet und getan werden muß, um die Programme wenn möglich auf beiden Anlagen lauffähig zu machen. Die Umstellung sollte in zwei Schritten erfolgen:

Zum einen muß das Programm ggf. von FORTRAN IV auf FORTRAN 77 umgestellt werden, am besten noch auf einer MVS-Anlage. Es gilt dann, nicht-CRAY-kompatible Erweiterungen zu eliminieren (==> "VSFORTRAN Application Programming: Language Reference"). Hierbei können schon Anpassungen an die CRAY, soweit sie in MVS nicht stören, vorgenommen werden. In vielen Fällen kann man sich auf Maßnahmen dieser Art beschränken.

Zum zweiten ist in bestimmten Fällen beim Übergang zur CRAY auch die Einführung von Sprachelementen notwendig, die bei VSFORTRAN (MVS) und CFT (CRAY) nicht gleich sind oder nicht zum gleichen Ergebnis führen. Diese Änderungen können eindeutig erkennbar vorgenommen werden, so daß sie auch wieder rückgängig gemacht werden können.

Ziel sollte es nicht nur sein, ein auf der CRAY lauffähiges Programm zu erhalten; vielmehr soll man ev. das Programm später auch wieder auf einer MVS-Anlage ausführen lassen können. Ein Vergleich von Programmergebnissen zwischen MVS und der CRAY ist wichtig - manchmal auch lange nach der ersten Umstellung. Dafür gibt es mehrere Gründe:

* Unterschiedliche Genauigkeit der Rechenanlagen bei REAL-Arithmetik
* Verschiedene Zahlenbereiche sowohl bei REAL als auch bei INTEGER
* Unterschiede beim Erkennen von Programmfehlern
  (Beispiel: nicht initialisierte Variablen)
* Mögliche CRAY-Compiler-Fehler
* Änderung der Resultate durch Programmoptimierung
  (z.B. Verwenden von hochoptimierten CRAY-Routinen)

11.2.2 Umstellen auf CRAY-kompatibles FORTRAN 77

Erhöhte Genauigkeit

REAL*4 und REAL*8 werden bei der CRAY immer wie REAL, also als ein Wort (64 Bit, davon 48 Bit Mantisse, entsprechend 14 bis 15 Dezimalen), realisiert. Die kleinste positive normalisierte Gleitkommazahl ist ungefähr 0.5E-2466, die größte ungefähr 0.5E+2466 (Zahlendarstellung ==> Anhang B). Die Genauigkeit entspricht etwa REAL*8 bzw. DOUBLE PRECISION bei MVS. COMPLEX*16 und COMPLEX*8 werden immer für COMPLEX genommen (Ein-Wort-Genauigkeit).
Unter Wirkung der Standard-Compiler-Option "ON=P" werden bei der CRAY sowohl DOUBLE PRECISION als auch REAL*16 auf zwei Worte mit 96 Bit Mantisse abgebildet, also etwa 29 Dezimalstellen; dasselbe gilt für Konstanten mit D und doppeltgenaue Intrinsic-Funktionen. Dies entspricht ungefähr der erweiterten Genauigkeit REAL*16 bei MVS. Diese hohe Genauigkeit wird bei der CRAY nur mit Mitteln der Software verwirklicht, ist daher sehr langsam und noch weniger zu empfehlen als bei MVS.

Hier soll nur von Programmen die Rede sein, die mit einfacher CRAY-Genauigkeit auskommen. Auch wo im Programm DOUBLE PRECISION, eine Konstante mit D oder eine doppeltgenaue Intrinsic-Funktion (z.B. DSIN) steht, soll nur mit einfacher CRAY-Genauigkeit gearbeitet werden. Man erreicht dies durch die Angabe von OFF=P in der CFT-Anweisung (==> 7.1.4).

Der Übergang zur CRAY entspricht dann weitgehend der Compilation auf einem MVS-System mit der Option AUTODBL(DBLPAD4), was ab VSFORTRAN Release 4 möglich ist. REAL*4-Größen (ebenso COMPLEX*8) werden aufgrund von DBLPAD4 wie REAL*8 (COMPLEX*16) behandelt; andere Datentypen werden bei EQUIVALENCE etc. mit Füllworten verlängert. Unterschied: mit OFF=P wird REAL*16 ebenfalls zu einfacher CRAY-Genauigkeit reduziert; COMPLEX*32, Konstanten mit Q und erweitertgenaue Intrinsic-Funktionen werden nicht verstanden.

Komplexe Rechnung

Komplexe Rechnungen mit doppelter Genauigkeit sieht die FORTRAN-Norm und auch CFT nicht vor. Weil sie also gar nicht bekannt sind, müssen die folgenden Intrinsic-Funktionen überall durch ihr für alle Datentypen geeignetes Äquivalent ersetzt werden:

```
CDABS  ->  CABS
CDCOS  ->  CCOS
CDEXP  ->  CEXP
CDLOG  ->  CLOG
CDSIN  ->  CSIN
CDSQRT ->  CSQRT
DCONJG ->  CONJG
DIMAG  ->  IMAG
DREAL  ->  REAL
```

INTEGER-Arithmetik

Im Hauptspeicher der CRAY wird bei jeder Deklarationsart (INTEGER, INTEGER*4, INTEGER*2) ein Wort (64 Bit) benutzt; die größte Zahl ist 9223372036854775807 (Zahlendarstellung ==> Anhang B). Dies kann bei MVS-Programmen Änderungen notwendig machen, wenn z.B. EQUIVALENCE mit INTEGER*2 und einem anderen Datentyp verwendet oder die 32-Bit-Darstellung im Algorithmus ausgenutzt wurde.

Zur Optimierung des Programms auf der CRAY sind, falls Integer-Multiplikationen und -Divisionen einen wesentlichen Anteil ausmachen, besondere Maßnahmen nötig. Dabei sollten folgende Ratschläge beachtet werden, wenn das Programm portabel bleiben soll:

Wenn die Integer-Rechnung in Gleitkomma-Arithmetik umgewandelt wird, sollte man INTEGER*4 durch REAL*8 und INTEGER*2 durch REAL ersetzen . Auf diese Art bleibt auch bei MVS-Systemen die Stellenzahl ausreichend groß. Häufig muß die Intrinsic-Funktion AINT - ganzzahliger Anteil in REAL-Darstellungen - in Ausdrücke eingefügt werden (nach Division; Warnung: REAL(I*K)/REAL(I) ist oft ungleich K).

Oftmals kann man mit der Compilerangabe OPT=FASTMD ohne viel Mühe eine Beschleunigung erreichen, wobei nun die Ergebnisse nur mit 46 Bit Genauigkeit berechnet werden. Allerdings werden aufgerufene externe Intrinsic-Funktionen (z.B. MOD) nicht schneller.

Weiter kann man auch eine CFT-Option INT=24 angeben; hierbei werden nur 24 Bit berücksichtigt. Damit kann man in skalaren und auch Vektorschleifen nochmals eine Beschleunigung erreichen, vor allem bei Divisionen.

Mit beiden Optionen werden auch die Integer-Konstanten erfaßt. Das kann man mit dem wie eine Deklaration zu verwendenden "CDIR$ INT24 (v1,v2,...)" nicht erreichen, auch nicht mit der INTEGER*2-Deklarationsanweisung. Beide Deklarationen bewirken, daß nur die 24 niederwertigen Bit verwendet werden. Die Erfahrung gebietet, hiermit vorsichtig zu sein.

Warnung: Integer-Zahlen größer als 2.1E14 ($3*2^{46}$) werden nicht korrekt in Gleitkomma Zahlen umgewandelt (bei Zuweisung und auch Funktion REAL). Ebenso ist die Umwandlung von REAL in INTEGER ab 7E13 (2^{46}) falsch.

Arbeiten mit den Datentypen LOGICAL und CHARACTER

Die FORTRAN-Norm kennt nur LOGICAL; LOGICAL*1 muß je nach der Bedeutung durch CHARACTER*1 ersetzt oder darf belassen werden. LOGICAL*1 im Sinne von "1 Byte" gibt es auf der CRAY nicht; es wird wie auch bei LOGICAL und LOGICAL*4 ein ganzes Wort belegt. Vorsicht bei EQUIVALENCE! Das LOGICAL-Wort wird als .TRUE. gedeutet, wenn das erste Bit 1 ist.

Zeichenketten (CHARACTER) werden bei CRAY mit 8 Zeichen je Wort gespeichert (8 Bit), jedoch nicht im EBCDIC-Code wie bei MVS, sondern im ASCII-Code, der eigentlich nur 7 Bit benötigt. Bei CHARACTER*n darf bei CRAY n nicht größer als 504 werden, bei MVS gewöhnlich nicht größer als 500 - mit der Compileroption CHARLEN jedoch bis 32767. Weil nicht derselbe Zeichencode benutzt wird, sind fast alle Programmstücke zwischen MVS und CRAY inkompatibel, welche die Intrinsic-Funktionen ICHAR und CHAR benutzen - meist wird mit Übersetzungstabellen gearbeitet.

Größer-kleiner-Vergleiche von Zeichenketten fallen nur dann auf beiden Rechnern gleich aus, wenn darin entweder nur Buchstaben oder nur Ziffern vorkommen. Wenn diese Bedingung nicht erfüllt ist, sollte man die Vergleiche mit Hilfe der Intrinsic-Funktionen LGE, LGT, LLE, LLT durchführen; auf beiden Maschinen wird dann beim Vergleich die Sortierfolge ASCII benutzt.
Achtung: Die Anwendung dieser Funktionen liefert auf dem MVS-System andere Ergebnisse als die der Vergleichsoperatoren.GE., .GT., .LE., .LT. !

Die meisten MVS-Spracherweiterungen müssen vermieden werden. Sie werden zum Teil bei eingeschalteter MVS-Compileroption FIPS(F) erkennbar. Einige grundsätzliche Regeln dazu:

* Initialisieren nur mit DATA.

* Arithmetische Daten nicht mit Zeichenketten (CHARACTER) initialisieren oder gleichsetzen (mit EQUIVALENCE). Im allgemeinen sind solche Anwendungen durch den fehlenden Datentyp CHARACTER in FORTRAN IV begründet. Man kann in solchen Fällen REAL*n und INTEGER*n durch CHARACTER*n ersetzen.

  Beispiele:

```
LOGICAL*1 L1(2)                -> CHARACTER*1 L1(2)
INTEGER*2 I2 /' '/             -> CHARACTER*2 I2
EQUIVALENCE(I2,L1(1)) bleibt      EQUIVALENCE(I2,L1(1))
                               -> DATA I2/' '/
```

* Arithmetische Datentypen und CHARACTER nicht im gleichen COMMON-Block benutzen!

* Keine Sedezimalkonstanten benutzen (meist markieren sie sowieso Maschinenabhängigkeit)! Das Z-Format dagegen darf man verwenden:

  Beispiele:

```
CHARACTER*16 C
C='2000800000000000'
READ(C,'(Z16)') FLOMIN
```

Ein/Ausgabe

Solange man nur Standard FORTRAN 77 verwendet, müssen auf FORTRAN-Ebene nur wenige Unterschiede zwischen MVS- und CRAY-Ein/Ausgabe beachtet werden:

Mit Format beschriebene und gelesene Sätze (FORMATTED) dürfen bei der CRAY nicht länger als 152 Zeichen sein;

Das Z-Format gibt es bei CRAY nicht für CHARACTER;

Während für das E-Format Ew.d bei MVS w >= d+7 gilt, sollte bei CRAY w >= d+8 sein, weil der Exponent größer sein kann. Man sollte also z.B. statt E23.16 jetzt E23.15 schreiben;

Gleitkommawerte in der I/O-Liste können nicht -wie bei MVS entgegen der Norm- einem I-Formatelement zugeordnet werden (der Speicherinhalt der Gleitkommaspezies wird dabei als INTEGER gedeutet). Auch andere ähnliche Normverstöße sind nicht erlaubt;

Undefinierte REAL-Variablen werden bei CRAY als R ausgegeben (da im ZIB zum Laden die Vereinbarung LDR,SET=INDEF. gilt.
besser: SEGLDR,GO,CMD='PRESET=INDEF').

Programmierung

Das Hauptprogramm sollte mit der PROGRAM-Anweisung beginnen. Ohne sie versagt die FLOW-Zeitanalyse bei der CRAY, mit deren Hilfe man erfährt, wieviel Prozent der Rechenzeit die einzelnen Unterprogramme und das Hauptprogramm verbrauchen. Wegen einer möglichen Zeitanalyse mit LOOPTOOL sollte der Programmablauf mit STOP enden.
Vier Intrinsic-Funktionsnamen haben sich beim Übergang von FORTRAN IV zu FORTRAN 77 geändert. Der MVS-Compiler bringt keine deutliche Fehlermeldung; auf jeden Fall sollten die alten Namen wegen der CRAY ersetzt werden:

```
ARSIN  -> ASIN
DARSIN -> DASIN
ARCOS  -> ACOS
DARCOS -> DACOS
```

Hinweis zur Programm-Optimierung:
Um die Vektorisierbarkeit von Programm-Elementen auf der CRAY zu ermöglichen, ist es öfters erforderlich, den Programm-Code zu verändern, nicht aber die Logik (==> 9. und Optimization-Guide). Dies kann in einzelnen Fällen dazu führen, daß dieser Teil des Programmes bei skalarer Bearbeitung mehr Rechenzeit benötigt als zuvor (z.B. auf dem MVS-System).

Speicherplatzbedarf

Die CRAY X-MP des ZIB besitzt für zwei Prozessoren gemeinsam 4 Megaworte Hauptspeicher. In diesem Speicher müssen das Betriebssystem COS und ein oder mehrere Benutzerprogramme Platz finden.

Bei der Abschätzung des Platzbedarfs ist zu berücksichtigen, daß für INTEGER*2 viermal, für LOGICAL*1 achtmal, für INTEGER*4, REAL*4 und LOGICAL doppelt soviel Platz in Byte zu berechnen ist wie bei einem MVS-System. Erst nach einer solchen Umrechnung kann man ungefähr 1 Megawort = 8 Megabyte setzen.

Bei sehr großen Programmen kann man mit dem Segment-Lader (SEGLDR) in relativ einfacher Weise ein wechselndes Uberlagern von Programmstücken und Daten, die nicht gleichzeitig verwendet werden, im Speicher ermöglichen (Overlay).

Platz für lokale Variable in Unterprogrammen kann man in vielen Fällen mit Hilfe der Compiler-Optionen OPT=BTREG und ALLOC=STACK sparen, diese sind ggf. mit der Anweisung SAVE zu modifizieren (==> 7.1.2 und 7.1.5).

11.2.3 Nicht-MVS-kompatible Änderungen

Nach den bisherigen Maßnahmen kann man beim Ändern der Programme von MVS-Systemen zur CRAY und umgekehrt mehrere Gruppen von Programmen bilden:

Programme, die auf der CRAY übersetzt werden sollen (mit OFF=P):

In diesen Programmen sind möglicherweise noch einige Namen von Intrinsic-Funktionen und Hilfsroutinen zu ändern, die die CRAY nicht kennt:

```
COTAN(arg)  -> COT(arg)          TREMAI  -> TREMAIN
IMAG(arg)   -> AIMAG(arg)        FILTEG  -> FILTERG und andere optimierte
LGAHMA(arg) -> ALGAMA(arg)                  CRAY-Programme
```

Wenn sie selten aufgerufen werden, kann man sich diese Umstellungen ersparen, indem man bei der CRAY Zwischenprogramme mit den MVS-Namen hinzunimmt.

Beispiel:

```
FUNCTION COTAN(X)
COTAN=COT(X)
END
```

Innerhalb der gesamten für die CRAY bestimmten Programme sind noch zu unterscheiden:

Portable Programme mit normaler Arithmetik (für MVS und CRAY gleich);

Programme, die nur für die CRAY bestimmt sind: CRAY-Version von Programmen mit ICHAR, CHAR, Bitfunktionen sowie Zwischenprogramme mit MVS-Namen (siehe oben);

Ersatzprogramme für optimierte CRAY-Routinen (z.B. FOLR, GATHER, etc.) stehen für MVS im allgemeinen nicht bereit.

MVS-Version von Programmen mit CHAR, ICHAR, Bitfunktionen.

Änderungen bei diesen Programmen:

* CHAR und ICHAR
  Die Intrinsic-Funktionen CHAR und ICHAR sind von dem Code abhängig, der für CHARACTER benutzt wird. Bei MVS ist es der EBCDIC-Code, bei CRAY dagegen ASCII. Von Programmen mit CHAR und ICHAR wird es also meist zwei Versionen geben müssen, eine für MVS und eine für CRAY.

* Bitfunktionen (nur in wirklich notwendigen Fällen verwenden!). In VSFORTRAN sind einige bitweise auf INTEGER*4 wirksame logische und Shift-Funktionen verfügbar. Auch bei CRAY gibt es solche Funktionen; sie haben aber andere Namen, zum Teil andere Aufgaben, und wirken statt auf 32-Bit-INTEGER auf 64 Bit jeden Datentyps. Man muß wieder zwei Versionen der Programme besitzen.

Liste der Bitfunktionen für beide Rechner:

MVS-Funktion	CRAY-Funktion
IAND(I1,I2)	AND(w1,w2)
IOR (I1,I2)	OR(w1,w2)
IEOR(I1,I2)	XOR(w1,w2),NEQV(w1,w2)
NOT(IEOR(I1,I2))	EQV(w1,w2)
NOT(I)	COMPL(w)
ISHFT(NOT(0).32-m)	MASK(n) n < 64: n 1-en links
ISHFT(NOT(0),m-32)	MASK(128-n) : n 1-en rechts
IAND(ISHFT(1,m),ISHFT(I,m-32))	SHIFT(w,n) zirkular
ISHFT(I,m) m>0	SHIFTL(w,n) links, <-- 0-en
ISHFT(I,-m) m>0	SHIFTR(w,n) rechts, 0-en -->
	POPCNT(w) Anzahl aller 1-en
	POPPAR(w) Anzahl 1-en modulo 2
	LEADZ(w)Zahl führender 0-en
IOR(IAND(I,K),IAND(J,NOT(K)))	CSMG(i,j,k) selective merge
BTEST(I,n) testen Bit n	-
IBSET(I,n) setzen Bit n	-
IBCLR(I,n) clear Bit n	-

11.2.4 Rückumwandlung von CRAY zu MVS

Wenn Programme, die auf der CRAY laufen, umgekehrt auf den MVS-Rechnern ausgeführt werden sollen, müssen einige Punkte bedacht werden: Auch wenn das Programm nur auf der CRAY läuft, muß die FORTRAN-Norm beachtet werden. Die Norm und MVS lassen für Variablennamen nur 6 alphanumerische Zeichen zu, CRAY aber 8! Der (kleinere) Gleitkomma-Zahlenbereich der MVS (10**-78 bis 10**75) sollte auch auf der CRAY, wenn möglich, nicht verlassen werden.

Faustregeln zum MVS-gerechten Programmieren auf der CRAY; Ziel: Genauigkeit von 14 bis 15 Stellen auf beiden Rechnern.

* einfache Genauigkeit als REAL*8 angeben oder auf MVS-Seite mit AUTODBL(DBLPAD4) übersetzen;
* OFF=P setzen;
* REAL-Konstanten mit D schreiben, als wären sie doppelt genau oder wieder auf MVS-Seite AUTODBL(DBLPAD4) benutzen;
* generische Intrinsic-Funktionen wählen (darauf kann man verzichten, wenn man mit AUTODBL arbeitet).

Nötige Änderung von Unterprogrammnamen (möglicherweise in allen Programmen), wenn man nicht Ersatzprogramme mit MVS-Namen verwandt hat:

```
COT(arg)    -> COTAN(arg)     TREMAIN -> TREMAI
AIMAG(arg)  -> IMAG(arg)      FILTERG -> FILTEG und andere optimierte CRAY-
ALGAMA(arg) -> LGAMMA(arg)               Programme
```

Austausch der Programme mit CHAR, ICHAR oder Bitfunktionen gegen eine MVS-Version.
Optimierte CRAY-Routinen (FOLR, FILTERG etc.) sind nicht direkt durch MVS-Routinen ersetzbar.

A. Vom ZIB bereitgestellte Programmpakete

Anwendungssoftware aus dem wissenschaftlichen Bereich ist für CRAY-Rechner bereits in großem Umfang verfügbar. Der Grad der Anpassung an die speziellen Vektorrechnereigenschaften ist dabei unterschiedlich. Für die von CRAY selbst beziehbare Anwendungssoftware gibt der "Application Software Library Catalog" einen Überblick. Eine umfangreiche Aufstellung über weitere, auch von dritter Seite angebote Software, enthält das "Directory of Applications Software". Beide Schriften liegen in den Rechenzentren zur Einsicht bereit. Interessenten an bestimmten Anwendungspaketen sollten sich an die Beratung ihres Rechenzentrums oder direkt an das Konrad-Zuse-Zentrum (ZIB) wenden. Es wird dann gemeinsam das zweckmäßige Vorgehen besprochen werden.

Folgende Software ist an der CRAY verfügbar (in Klammern der ACM-Index - vgl. DOC,SOFTWRB):

ADINA (T4)

ADINA (Automatic Dynamic Incremental Nonlinear Analysis) ist ein Programmsystem zur statischen und dynamischen Analyse der Verschiebungen und Spannungen von Festkörpern und Strukturen sowie zur Analyse von Flüssigkeiten. Die derzeit an der CRAY bereitgestellte Version von ADINA ist nur für Benutzer aus der Bundesanstalt für Materialprüfung verfügbar. Sofern auch von anderen Benutzern Interesse an der Nutzung von ADINA besteht, muß geprüft werden, ob eine Bereitstellung für alle Benutzer des ZIB durchgeführt werden kann.

BIZEPS2 (J5)

Oberhalb von GKS angesiedelte Programmbibliothek für allgemeine grafische Anwendungen.

Weitere Information: DOC,BIZEPS2.

BLAS (F1)

Basic Linear Algebra Subprograms: 22 CRAY-Assembler Routinen, die einen Teil der BLAS-Routinen, hochoptimiert für die Nutzung auf der CRAY, bilden. BLAS ist Bestandteil von ==> SCILIB sowie ==> LINPACK.

Literatur: CRAY Programmer's Library Reference Manual SR-0113

EISPACK (F2)

EISPACK ist ein Unterprogrammpaket für FORTRAN zur Berechnung von Eigenwerten und Eigenvektoren von verschiedenen Klassen von Matrizen (z.B. reell symmetrisch, komplex, Hessenberg-Form, usw.). Auf der CRAY ist EISPACK Bestandteil von SCILIB.

Literatur: Smith et al, Matrix Eigensystem Routines-EISPACK Guide, Lecture Notes in Computer Science, Springer-Verlag, 1974
- H. Wilkinson, C. Reinsch: Handbook for Automatic Computation Volume 2, Linear Algebra, Springer-Verlag 1971
- CRAY Programmer's Library Reference Manual, SR0113

FIDISOL (D3)

Lösung gewöhnlicher und partieller Differentialgleichungen mit selbstadaptiven Differenzenverfahren, Nachfolgeprodukt von SLDGL für Vektorrechner (speziell an die CRAY-Anlage angepaßte Version).

Weitere Information: DOC,CRAY,FIDISOL.

FORSIM 6 (H4)

FORSIM ist ein FORTRAN-Programmpaket zur Lösung von Systemen gekoppelter partieller und/oder gewöhnlicher Differentialgleichungen.
FORSIM wird zur Simulation von kontinuierlichen Zeitsystemen

verwendet, die durch entsprechende Differentialgleichungen beschrieben werden können.

Weitere Information: DOC,CRAY,FORSIM6

Literatur: Benutzungsanleitung für FORSIM VI (RRZN-Umdruck PRO.SIM 6) Herausgeber: Regionales Rechenzentrum für Niedersachsen / Universität Hannover, Dezember 1984, Telefon: (0511) 762 2883

GAUSSIAN 82 (T2) Programmsystem für "ab initio-" Berechnungen in der Quantenchemie, entwickelt an der Carnegie-Mellon-University.

Weitere Information: DOC,CRAY,GAUS82.

GRIPS (J5) Die Unterprogrammbibliothek GRIPS (Grafik für interaktive und passive Systeme) kann auf der CRAY bei allgemeinen grafischen Anwendungen von FORTRAN - Programmen aufgerufen werden. GRIPS basiert auf dem Graphischen Kernsystem GKS. (An der CRAY-Anlage ist nur die passive Variante GRIPS1 verfügbar.)

Weitere Information: DOC,GRIPS

IMSL (V0) Die Programmbibliothek IMSL besteht aus FORTRAN-Unterprogrammen aus verschiedenen Gebieten der Mathematik und Statistik.

Weitere Information: DOC,IMSL und DOC,CRAY,IMSL

LINPACK (F4) LINPACK ist ein Unterprogrammpaket für die Lösung und Analyse von Linearen Gleichungssystemen. Auf der CRAY ist LINPACK Bestandteil von SCILIB.
Literatur: J.J.Bongarra, C.B.Moler, J.R.Bunch, G.W.Stewart: LINPACK User's Guide; Philadelphia 1979:
Society for Industrial and Applied Mathematics (SIAM)

Literatur: CRAY Programmer's Library Reference Manual SR-0113

NAG (V0) NAG ist eine umfangreiche Programmbibliothek bestehend aus FORTRAN-Unterprogrammen.

Weitere Information: DOC,NAG und DOC,CRAY,NAG

SCILIB (V0) Die Scientific Applications Subprograms ist eine von der Firma CRAY Research entwickelte bzw. angepaßte Programmbibliothek; Die Routinen sind für die Nutzung auf der CRAY hochoptimiert und in allen bekannten Fällen entsprechenden Routinen anderer Bibliotheken überlegen.

SCILIB enthält Routinen aus folgenden Gebieten:
- lineare Algebra
- funktionale und lineare Rekursion
- LINPACK
- EISPACK
- Fast Fourier Transformation
- Filler Routinen
- Such- und Sortierroutinen

Literatur: CRAY Programmer's Library Reference Manual SR-0113

An der CD-Anlage von TU und ZIB existiert darüber hinaus eine weitgehend aufrufkompatible Nachbildung wichtiger Teile der

von CRAY bereitgestellten Bibliothek SCILIB für CD-Anlagen, die insbesondere für die Programmentwicklung eingesetzt werden kann.

Weitere Information: DOC,SCILIB

TUBLIB (LO) TUBLIB ist eine Unterprogrammsammlung, hochoptimiert für die Nutzung an der CRAY, enthält z.Zt.:

XIO: Routinen zum Ersetzen sequentieller Ein/Ausgabe mit fester Satzlänge durch E/A mit Direktzugriff.

Weitere Information: DOC,CRAY,XIO

MSIO: Routinen zur Ein/Ausgabe großer Datenmengen mit Hilfe einer logischen Striping-Technik.

Weitere Information: DOC,CRAY,MSIO

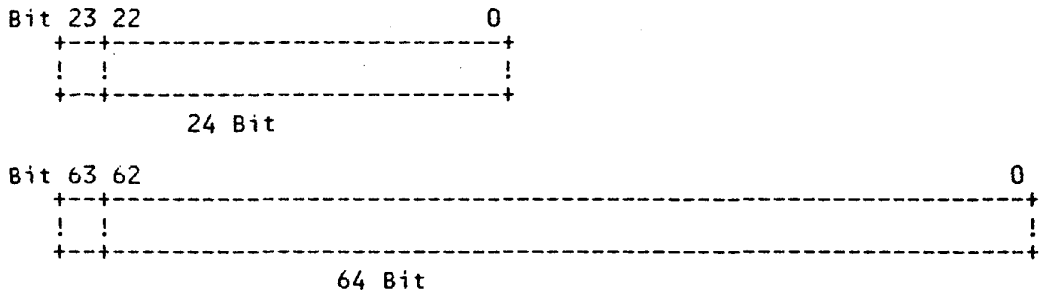
B. Zahlenbereiche, -Genauigkeit und -Speicherung

An der CRAY hat jedes Hauptspeicherwort 64 binäre Ziffern (Bits). Die Register haben entweder 64 Bits (V, T und S) oder 24 Bits (A und B). Darin werden Daten gespeichert. Die Mehrzahl der Programme verarbeitet die gespeicherten Daten als ganze Zahlen (INTEGER), Gleitkommazahlen (REAL), Buchstaben (CHARACTER) oder logische Werte (BOOLEAN). Alle Typen haben eine andere Darstellung im Hauptspeicher oder in den Registern. Die folgenden Kapitel zeigen die interne Darstellung dieser Typen auf der CRAY. Die Darstellung ist bei anderen Rechnern verschieden.

B.1 Ganze Zahlen (INTEGER)

Ganze Zahlen (24 Bit oder 64 Bit) werden im Zweierkomplement verarbeitet. Dabei werden positive ganze Zahlen durch ihr binäres Äquivalent dargestellt. Eine negative ganze Zahl erhält man durch Komplementieren der absoluten Zahl und Addition von Eins. Daraus ergibt sich, daß das am weitesten links stehende Bit (Bit 63) den Wert Null für positive und Eins für negative Zahlen hat. Dieses Bit wird deshalb Vorzeichenbit genannt.

Die Darstellung von ganzen Zahlen:



Das folgende Beispiel zeigt die Darstellung der Zahl -29 in 24 Bit:

```
+29         ----> 000 000 000 000 000 000 011 101
Komplement  ----> 111 111 111 111 111 111 100 010
Addiere 1   ----> 111 111 111 111 111 111 100 011 = -29
```

Die Additions- und Multiplikationseinheiten für die Adreßrechnung verarbeiten nur 24-Bit Integerzahlen. Die Skalar- und Vektoradditionsfunktionseinheit verarbeiten 64 Bit Integerzahlen. Ganze Zahlen, dargestellt in 64 Bit, liegen bei der CRAY im Bereich

-2^{63} (-9.223.372.036.854.775.807) <= N <= $2^{63}-1$ (9.223.372.036.854.775.806)
(2^{63} ist ungefähr gleich 10^{18})

Bei CFT werden die Indizes von DO-Loops oder von Feldelementen in den Adressfunktionseinheiten berechnet und dürfen deshalb $2^{23}-1$ nicht übersteigen.

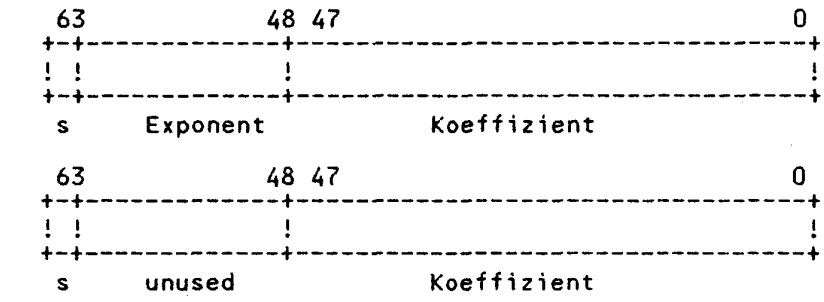
B.2 Gleitkommazahlen (REAL)

Eine Gleitkommazahl wird normalerweise als Dezimalbruch multipliziert mit einem Exponent zur Basis 10 dargestellt. Beispielsweise wird 3580.00 zu 0.3580*10(4). Eine interne Gleitkommazahl an der CRAY wird aufgespalten in einen binären Koeffizienten und einen Exponenten mit der Basis zwei. Der Koeffizient ist eine 48-Bit-Mantisse. Das Vorzeichen der Mantisse steht in Bit

63. Ein negativer Koeffizient wird nicht komplementiert, sondern der absolute Wert steht als Binärzahl im Koeffizientenfeld und nur das Vorzeichenbit wird gesetzt.

Einfach genaue Gleitkommazahl

Doppelt genaue Gleitkommazahlen



s = Vorzeichen des Koeffizienten

Die Darstellung von Gleitkommazahlen:

Der Exponent steht als geschobene (biased) Integerzahl in Bit 48 bis 62. In den zur Verfügung stehenden 15 Bits kann (oktal) ein Wert im Bereich

00000 bis 77777

dargestellt werden. Darin liegt der Exponent, der sowohl positiv als auch negativ sein kann. Die Mitte des Bereiches (40000) wird einem Exponenten mit dem Wert null (2^0) gleichgesetzt. Zu jedem Exponenten wird also 40000 addiert, damit diese Darstellung erreicht wird. Der dem Benutzer zur Verfügung stehende Bereich für einen Exponenten liegt zwischen:

	20000	bis	57777 (verschobener Exponent)
oder	-20000	bis	+17777 (echter Exponent)
also	2^{-20000}	bis	2^{17777} (Zahlbereich)

Werte unter 20000 und über 57777 werden benutzt, Überläufe "underflow" und "overflow" darzustellen.

Die interne Darstellung von Überläufen:

Wenn Bit 62 und 61 gleich sind, so bedeutet das einen Überlauf.

		Bit 62 61 60
00000		0 0 0
:	underflow	
17777		0 0 1
20000	-20000	0 1 0
:		
40000	0	1 0 0
:		
57777	17777	1 0 1
60000		1 1 0
:	overflow	
77777		1 1 1

Gleitkommazahlen bei CRAY liegen im Bereich von ungefähr:

$0{,}458401*10^{-2466}$ bis $0{,}545374*10^{2466}$

Bei einfacher Genauigkeit kann bei der 48-Bit Mantisse mit 14 signifikanten Dezimalstellen gearbeitet werden. Bei doppelter Genauigkeit und 2*48-Bit-Mantisse erhöht sich die Anzahl der signifikanten Stellen auf 29.

Das folgende Bild zeigt noch Beispiele, wie eine REAL-Zahl intern gespeichert ist:

Dezimal	zur Basis 2	Intern (oktal)
unnormalisiert		
1.0	(=0.00001*2 ^{5})	0 40005 0200000000000000
normalisiert		
1.0	(=0.1 *2 ^{1})	0 40001 4000000000000000
17.0	(=0.10001*2 ^{5})	0 40005 4200000000000000

B.3 Logische Größen

Logische Variablen bei CRAY-FORTRAN (CFT) können die Werte TRUE und FALSE annehmen. TRUE wird intern als ein negativer Wert und FALSE als Null oder ein positiver Wert dargestellt. Anders gesagt: Wenn das Vorzeichenbit 1 ist, so ist der Wert TRUE; wenn das Vorzeichenbit 0 ist, so ist der Wert FALSE.

Anhang C. Zeichendarstellung

Zeichen (CHARACTER) werden auf der CRAY in ASCII dargestellt. Ein Zeichen wird in acht Bit gespeichert; in ein CRAY-Wort passen also acht Zeichen. In der folgenden Tabelle stehen die 128 möglichen ASCII-Zeichen, sowohl druckende Zeichen als auch Steuerzeichen.

ZEICHEN	ASCII CODE oktal	BESCHREIBUNG	
NUL	000	Null	\
.	.	.	|
.	.	.	> Steuer-Zeichen
.	.	.	|
US	037	Unit Seperator	/
"blank"	040	Leerzeichen	(Space)
!	041	Ausrufezeichen	(Exclamation Mark)
"	042	Anführungszeichen	(Quotation Marks)
#	043	Balkenkreuz	(Number Sign)
$	044	Dollar	
%	045	Prozent	
&	046	kaufm. und	(Ampersand)
'	047	Hochkomma/Accent aigu	(Apostrophe)
(	050	offene Klammer	
)	051	geschlossene Klammer	
*	052	Stern	(Asterisk)
+	053	Plus	
,	054	Komma	
-	055	Minus	(Bindestrich)
.	056	Punkt	
/	057	Schrägstrich	(Slash)
0	060	null	
1	061	eins	
2	062	zwei	
3	063	drei	
4	064	vier	
5	065	fünf	
6	066	sechs	
7	067	sieben	
8	070	acht	
9	071	neun	
:	072	Doppelpunkt	(Colon)
;	073	Strichpunkt	(Semicolon)
<	074	Kleinerzeichen	(Less than)
=	075	Gleichheitszeichen	(Equal)
>	076	Größerzeichen	(Greater than)
?	077	Fragezeichen	(Question mark)
@	100	"Klammeraffe"	
A	101	\	
B	102	|	
C	103	|	
D	104	|	
E	105	> Großbuchstaben	
F	106	|	
G	107	|	
H	110	|	
I	111	|	

Zeichendarstellung bei CRAY (ASCII) (Teil 1 von Anhang C.)

ZEICHEN	ASCII CODE oktal	BESCHREIBUNG
J	112	|
K	113	|
L	114	|
M	115	|
N	116	|
O	117	|
P	120	|
Q	121	> Großbuchstaben
R	122	|
S	123	|
T	124	|
U	125	|
V	126	|
W	127	|
X	130	|
Y	131	|
Z	132	/
[	133	Eckige Klammer auf
\	134	(Backslash)
]	135	Eckige Klammer zu
^	136	Accent circonflexe
_	137	Unterstreichung (Underline)
`	140	Accent grave
a	141	\
b	142	|
c	143	|
d	144	|
e	145	|
f	146	|
g	147	|
h	150	|
i	151	|
j	152	|
k	153	|
l	154	|
m	155	> Kleinbuchstaben
n	156	|
o	157	|
p	160	|
q	161	|
r	162	|
s	163	|
t	164	|
u	165	|
v	166	|
w	167	|
x	170	|
y	171	|
z	172	/
{	173	geschweifte Klammer auf
|	174	Senkrechter Strich
}	175	geschweifte Klammer zu
~	176	Tilde
DEL	177	Delete (Steuerzeichen)

Zeichendarstellung bei CRAY (ASCII) (Teil 2 von Anhang C.)

Sehr geehrter Benutzer der CRAY,

wir freuen uns, wenn Sie an der Verbesserung unseres Handbuches mitarbeiten. Für Mitteilungen an uns haben wir, dem amerikanischen Beispiel folgend, die Rückseite dieses Blattes als Brief an uns vorbereitet: bitte trennen Sie dazu dieses Blatt heraus und schreiben darauf, wenn Sie z.B. -

* Fehler gefunden haben,

* Erweiterungen oder Streichungen vorschlagen,

* einzelne Teile diese Handbuchs allgemein kommentieren möchten.

Bitte falten Sie dann dieses Blatt an den punktierten Linien nach hinten, stecken es in einen Fensterumschlag und schicken es an unsere bereits eingetragene Adressse.

Vielen Dank!

.....

Absender:

An das
Konrad-Zuse-Zentrum für
Informationstechnik Berlin
zu H. von Herrn Wolfgang Stech
Heilbronner Straße 10

1000 Berlin 31

.....

Betr.: CRAY-Handbuch des ZIB, Kapitel - , Seite - , . Absatz

Sehr geehrter Herr Stech,