RUHR - UNIVERSITÄT BOCHUM

Arbeitsbericht

des

Rechenzentrums

Direktor: Prof. Dr. H. Ehlich

Nr. 7503 BO.E1.14

BOTRAN, eine Fortran Spracherweiterung durch Code-Prozeduren

von

H.-D. Sander

Juni 1975

Bochum, Universitätsstr. 150, Gebäude NA

I N H A L T :

Kurzfassung

BOTRAN stellt den Benutzern, die in Fortran programmieren, TAS-Prozeduren zur Verfügung, die

1. rekursive Aufrufe von Fortranunterprogrammen ermöglichen,
2. dynamisch deklarierte Felder zulassen, und
3. den TAS-Befehlscode als Unterprogramme anbieten.

Darüber hinaus gibt es ein Unterprogrammpaket zur Manipulation von Versorgungsblöcken, das es ermöglicht Fortran-Unterprogramme mit variabler Parameterzahl und mit variablen Parametertypen zu schreiben.

Im ersten Teil wird die Möglichkeit der rekursiven Unterprogrammaufrufe beschrieben. In diesem "Rekursiv-Paket" sind die TAS-Prozeduren zur dynamischen Felddeklaration integriert. Selbstverständlich gibt es somit auch lokale Variablen und Felder.

Im zweiten Teil werden "String-Arrays" beschrieben. Dies ist ein System zur dynamischen Felddeklaration, in dem die Feldnamen erst zur Operatorlaufzeit festgelegt werden (also z. B. eingelesen werden) und die Länge der Felder nicht vor der dynamischen Deklaration bekannt sein muß.

Im dritten Teil werden die Möglichkeiten zur Manipulation von Versorgungsblöcken und Informationsunterprogramme vorgestellt.

Im vierten Teil werden Aufrufe für spezielle TAS-Befehle beschrieben. Interessant sind hierbei vor allem die Tabellensuchbefehle und Transportbefehle.

1. Aufbau von Fortran-Unterprogrammen

Ein Fortran-Unterprogramm ist eine Programmeinheit, die ihren eigenen Befehls- und Variablenteil hat. Der Bezug zur rufenden Programmeinheit wird durch die Parameterliste und durch eventuell existierende Commonblöcke hergestellt. Ausgehend von der formalen Parameterliste werden vom Übersetzer Informationsaustauschmöglichkeiten generiert. Das Unterprogramm erhält einen "Wasserkopf", der entsprechend der formalen Parameterliste die aktuelle Parameterliste auswerten soll. Der Wasserkopf holt sich die Werte oder Adressen der aktuellen Parameter, kontrolliert eventuell (bei Dynkon= -Std-), ob der aktuelle Parameter mit dem formalen kompatibel ist, und speichert dann Wert oder Adresse (je nach Typ) auf unterprogrammeigene Speicherplätze. Erst danach wird der eigentliche Befehlsteil durchlaufen. Vor dem Verlassen des Unterprogramms sorgt ein "Wasserfuß" wieder für den Informationsaustausch, d. h. die Werte der Variablen der aktuellen Parameterliste übertragen.

1.1. Versorgungsblock

Damit der Informationsaustausch möglich ist und der Operatorlauf hinter dem Aufruf des Unterprogramms fortgesetzt werden kann, werden folgende Konventionen beachtet:

1. Bei Aufruf eines Unterprogramms werden diesem im Register RB die Rücksprungadresse und in RA die Adresse eines Versorgungsblocks übergeben.

2. Der Versorgungsblock entspricht der aktuellen Parameterliste und hat folgenden Aufbau:

24	4	4	6	2	8	
FA	SS	A	O	B	N	Kopfwort
HV			ZV			Haupt- und Zusatzversorgung für N Parameter

Der Versorgungsblock wird während der Übersetzung des rufenden Programms aufgebaut. Im Kopfwort des Versorgungsblocks steht die Fehleradresse FA, die im Fehlerfall vom aufgerufenen Unterprogramm angesprungen werden soll, der Sprachschlüssel SS (in Fortran ist SS = 1), die Aufrufart B (Function oder Subroutine), der Aufruftyp A (Typ des Function) und die Anzahl N der aktuellen Parameter. Die nächsten N Ganzworte enthalten jeweils die Haupt- und Zusatzversorgung der einzelnen Parameter. Die Hauptversorgung gibt Auskunft über die Adresse des Parameters und besteht zumeist aus einem Versorgungsbefehl, der im Register RB die Adresse des Parameters übergibt. Die Zusatzversorgung gibt die Art des Parameters an (Variable, arithmetischer Ausdruck, Feld, Subroutine ...). Die Zusatzversorgung wird nur bei eincompilierten dynamischen Kontrollen vom aufgerufenen Unterprogramm ausgewertet.

Tabelle der HVZV

Ausdruck	Versorgungsbefehl	0	1 - 8	1
Variable	"	0	1 - 8	2
Function	"	0	1 - 8	3
Subroutine	"	0	-	4
Prozedur	"	0	-	5
Marke	Programmadresse	0	-	6
Literalkonstante	Versorungsbefehl	0	Anzahl der Zeichen	15
Feld	"	1	Adresse der	
Feldelement	"	2	Feldbeschreibung	

Typ-Tabelle

1 integer *2

2 integer *4

3 real *4

5 complex *8

6 complex *16

7 logical *1

8 logical *4

1.1.1. Variablenübergabe

Ist laut formaler Parameterliste ein Parameter eine einfache Variable, so führt der Wasserkopf den Versorgungsbefehl der HV aus, holt sich den Inhalt der Speicherzelle, dessen Adresse in RB steht und speichert in eine eigene Speicherzelle. Steht die formale Variable in " / / ", so wird nur die Adresse der aktuellen Variablen gespeichert. Bei jedem Zugriff auf diese formale Variable, wird die gespeicherte Adresse benutzt, um auf der Speicherzelle im rufenden Programm zu rechnen. Steht auf aktueller Parameterposition ein arithmetischer Ausdruck, so generiert der Übersetzer im rufenden Programm eine Befehlsfolge, die den Ausdruck berechnet und in eine Hilfszelle speichert, dessen Adresse im VB übergeben wird.

1.1.2. Übergabe von Feldern

Jeder DIMENSION-Anweisung entspricht eine Feldbeschreibung.

Typ N	
Anfangsadr.	
1. freie Adresse	Anzahl der Ele. mente der 1. Dim.
Adr. des Dim.-Bl.	

Der erste Block der Feldbeschreibung besteht aus 4 Halbwörtern. Im ersten HW steht der Typ des Feldes und die Anzahl der Dimensionen. Das zweite HW enthält die Adresse des ersten Feldelements, das dritte HW gibt die Adresse der ersten nicht mehr vom Feld belegten Speicherzelle an. Das vierte HW verweist auf den Dimensionsblock. In diesem stehen die Dimensionslängen in Anzahl der Elemente. Ein Element kann je nach Typ des Feldes 1 bis 8 HW belegen. Steht in der formalen Parameterliste ein Feld, so wird bei Aufruf des Unterprogramms die Adresse des entsprechenden aktuellen Parameters in das zweite HW der Feldbeschreibung gespeichert. Bei variabler Dimensionierung wird auch der Rest der Feldbeschreibung entsprechend geändert. Bei Zugriff auf ein Feldelement wird die aktuelle Adresse mit Hilfe der Feldbeschreibung berechnet, d. h. das Unterprogramm hat eine eigene Feldbeschreibung, rechnet jedoch auf den Speicherplätzen des rufenden Programms. Daher kann ein Feld im Unterprogramm nie länger sein als im rufenden Programm.

1.2. Sicherung der Rücksprungadresse

Die in RB übergebene Rücksprungadresse wird in eine eigene Speicherzelle gesichert (Kontrollblock). Desgleichen wird die VB-Adresse abgespeichert, um später vom "Wasserfuß" wieder benutzt zu werden.

1.3. Probleme der rekursiven Aufrufe

a) Wird ein Unterprogramm rekursiv aufgerufen, überschreibt es sich die Rücksprungadresse.

b) Die Versorgungsblockadresse wird überschrieben.

c) Es werden lokale Variablen benötigt.

d) Bei Feldern kann man sich lokale, globale und übergebene Felder vorstellen.

Ein im Unterprogramm fest dimensioniertes Feld, das nicht in der Parameterliste erscheint, wird durch alle Aufrufverschachtelungen dieselben Speicherplätze im Unterprogramm belegen. Dies ist also ein globales Feld. Lokale Felder hingegen müssen bei jedem Aufruf des Unterprogramms neu kreiert werden und bei RETURN wieder aufgegeben werden. Um die Lebensdauer der lokalen Felder an die Aufrufverschachtelung zu binden, benötigt man einen dynmisch verwalteten Variablenbereich.

1.3.1. Rekursive Aufrufe

Ein direkter rekursiver Aufruf von Unterprogrammen wird schon vom Übersetzer verhindert. Daher muß man sich an folgende Konventionen halten, um TAS-Prozeduren für rekursive Aufrufe benutzen zu können:

1. Aufbau der formalen Parameterliste:

```
SUBROUTINE UP(UPREK,/P1,/P2/,...)
EXTERNAL UPREK
```

Der erste Parameter muß als External gekennzeichnet sein.

2. Aufruf

Erster Aufruf des Unterprogramms:

```
CALL REK (UP,P1,P2,...)
```

Rekursiver Aufruf:

```
CALL REK (UPREK,P1,P2,...)
```

Beispiel: Rekursiver Aufruf mit REK

```
DIMENSION FELD(10)
EXTERNAL UP
:
CALL REK(UP,I,J,K,FELD)
:
END

SUBROUTINE UP(UPREK,/IA/,/IB/,/IC/,F)
EXTERNAL UPREK
REAL F(10)
:
CALL REK(UPREK,IA,IB,IC,F)
:
END
```

Die TAS-Prozedur REK hat zwei Aufgaben:

1. Der Versorgungsblock wird unverändert an das Unterprogramm weitergegeben, das durch den ersten Parameter identifiziert wird.

2. Die Rücksprungadresse wird gesichert und dem Unterprogramm wird eine Rücksprungadresse mitgegeben, die auf eine TAS-Prozedur verweist. Zur Sicherung des Versorgungsblocks und der Rücksprungadresse wird ein Freispeichergebiet (vergl. BO&FSP) dynamisch, entsprechend der Aufrufverschachtelung, verwaltet.

   Da der Versorgungsblock nicht verändert wird, entspricht UPREK genau dem aufgerufenen Unterprogramm, d. h. auf UPREK wird die Startadresse von UP übergeben. Eine Endebehandlung durch den Wasserfuß entfällt, da die Variablen in Schrägstriche eingeschlossen sind.

Beispiel:

```
IA = 1
IB = 2
CALL REK(TEST,IA,IB)
END
SUBROUTINE TEST(TT,/I/,/J/)
EXTERNAL TT
K = K + 1
IF(K . NE . 1) RETURN
CALL  REK (TT,I,J)
I = I + 1
J = J + 2
RETURN
END
```

1.3.2. Lokale Variable

Lokale Variable bezeichnen Speicherplätze, die mit Aufruf des Unterprogramms generiert und beim Rücksprung wieder aufgegeben werden. Ein Äquivalent zu lokalen Variablen stellt der Aufruf von LØKALV zur Verfügung. LØKALV sichert die Werte der angegebenen Variablen in den Freispeicher. Somit stehen diese Speicherplätze dem Unterprogramm wieder zur freien Verfügung. Beim Rücksprung werden die gesicherten Werte zurückgespeichert. Diese Technik erlaubt sogar noch mehr, als einfach einige Variablen als lokal zu definieren. Ist die erste Anweisung im Unterprogramm der LØKALV-Aufruf, so entspricht dies genau der Deklaration von lokalen Variablen. Da der Aufruf der TAS-Prozedur LØKALV an jeder Stelle im Unterprogramm stehen kann, ist es möglich im Operatorlauf zu entscheiden, welche Variablen ab wann als lokal zu betrachten sind.

1.3.3. Lokale Felder

Da es in Fortran die Möglichkeit gibt, bei Übergang zu einem Unterprogramm bestehende Felder dynamisch neu zu dimensionieren, bietet sich der Aufruf von Unterprogrammen als ideale Schnittstelle an, neue im Freispeicher liegende Felder zu kreieren. Die dafür zuständigen "REKN"-Prozeduren stellen die notwendigen Speicherplätze zur Verfügung und übergeben die entsprechende Anfangsadresse im Versorgungsblock an das aufzurufende Unterprogramm. Die dynamische Kreation von Feldern läßt sich auch gut anwenden für Fortranprogramme, die mit unterschiedlichem Platzbedarf (z. B. für Daten) rechnen müssen. So können die Programme mit optimal genutzten Kernspeichern laufen und müssen nicht immer maximal dimensioniert sein.
Mit der TAS-Prozedur LOKALF ist es möglich bei rekursiven Aufrufen die Lebensdauer neu kreierter Felder an die entsprechende Aufrufverschachtelung zu binden. LØKALF sorgt dafür, daß die Feldbeschreibung in den Freispeicher gesichert wird und bei RETURN das Feld aufgegeben und eventuell die alte Feldbeschreibung wieder zurückgespeichert wird.

2. Beschreibung der TAS-Prozeduren

2.1. REK

Die aktuelle Parameterliste (besser Versorgungsblock) wird ohne Änderung in den Freispeicher kopiert, die Rücksprungadresse wird ebenfalls gesichert und eine eigene Rücksprungadresse nebst neuer Versorgungsblockadresse (im Freispeicher) wird an das Unterprogramm übergeben, das durch den ersten Parameter identifiziert wird. Daher muß in dem angesprungenen Unterprogramm der erste formale Parameter aus syntaktischen Gründen anders benannt sein als das UP selbst und muß in einer EXTERNAL-Anweisung erscheinen. Falls rekursive Aufrufe folgen, ist es notwendig, sämtliche Variablennamen in der formalen Parameterliste in Schrägstriche einzuschließen, um die übergebenen Variablen des rufenden Programms stets mit den aktuellen Werten besetzt zu halten. Da der Versorgungsblock im Freispeicher nicht schreibgeschützt ist, kann er verändert und direkt an andere Unterprogramme weitergegeben werden (vergl.: VBAD und die Möglichkeiten der Versorgungsblockmanipulationen).Verzichtet man auf die Versorgungsblockkopie im Freispeicher, so kann man auch RUF oder CALL benutzen. Da eine eigene Rücksprungandresse übergeben wird, hat das Programmpaket die Möglichkeit, über entsprechende Verweislisten nach RETURN aus dem Fortran-Unterprogramm die aktuelle Rücksprungadresse wieder aus dem Freispeicher zu holen, andere Endebehandlungen durchzuführen (vergl. LØKALE und LØKALV) und Freispeicherpegel wieder zurückzusetzen.

Beispiel:

```
      EXTERNAL FAK
      INTEGER FAK REK
      READ (,) N
      NFAK = REK (FAK, N)
      WRITE (,) NFAK
      END

      INTEGER FUNCTION FAK (FREK, N)
      EXTERNAL FREK
      INTEGER FREK, REK LØKALV(KØNTRØ, N)
      IF (N.NE.1) GOTO 1
      FAK = 1
      RETURN
    1 FAK = REK (FREK, N-1)*N
      RETURN
      END
```

2.2. REKN und NEW

REKN bringt dieselben Leistungen wie REK und bietet zusätzlich die Möglichkeit dynamisch Felder zu kreieren.

Der in den Freispeicher kopierte aktuelle Versorgungsblock wird zunächst durchsucht, ob ein Function mit dem Namen NEW übergeben werden soll. Steht auf aktueller Parameterposition das Function NEW, so wird der Versorgungsblock so geändert,

daß an dieser Stelle ein Feld übergeben wird. Dieses Feld liegt im Freispeicher. Die auf NEW folgenden Parameter werden als Dimensionsvariablen aufgefaßt, bis wieder ein Function oder Subroutine oder Parameterlistenende gefunden wird. Der Typ des Feldes wird durch den Typ von NEW bestimmt. Damit der Übersetzer die richtige Versorgung für das Function erstellt, muß im rufenden Programm ein expliziter Aufruf dieses Functions stehen. Bei Rücksprung aus dem Unterprogramm werden diese Felder wieder aufgegeben. Um diese Felder dynamisch mit einer Vorbesetzung zu versehen, benutze man VØRBES.

Beispiel:

```
INTEGER NEW
REAL*8 NEW1
EXTERNAL UP
:
CALL REKN (UP, NEW, N1, N2, N3, NEW1, N1, N4)
STOP
I=NEW(I)
F=NEW1(F)
End

SUBROUTINE UP(Uprek, F1, N1, N2, N3, F2, N4, N5)
EXTERNAL UPREK
INTEGER F1 (N1, N2, N3)
Real*8 F2 (N4, N5)
:
END
```

NEW1 ist nur ein anderer Name für NEW (vergl. Liste der äquivalenten Namen), damit Felder verschiedenen Typs kreiert werden können.
Der Aufruf des INTEGER FUNCTION NEW liefert den aktuellen Freispeicherpegel aus.

Arbeitet man zusätzlich noch mit den "ARR"-Prozeduren, so ist außerdem folgende Anwendung von REKN möglich:
Steht in der aktuellen Parameterliste das INTEGER FUNCTION ARRAKT, so wird ein Feld übergeben, das durch den folgenden Parameter identifiziert wird. Die Feldlänge wird auf dem zweiten Parameter hinter ARRAKT übergeben.

Beispiel:

```
:
READ ( ) NAME
CALL REKN (UP, ARRAKT, NAME, N)
:
END
SUBROUTINE UP (UPREK, F, NAME, N)
REAL F(N)
:
```

Vergleiche: Beschreibung des ARR-Systems

2.3. REKNR

REKNR hat nur einen Parameter. Er wird als Adressenangabe eines Versorgungsblocks gedeutet mit dem REK aufgerufen wird.

REKNR

REKNR erwartet als einzigen Parameter die Adresse eines Versorgungsblocks für REKN.

2.4. LØKALV

Der erste Parameter ist eine Kontrollvariable. Ist sie gleich Null, so wird eine Befehlsfolge generiert und im Freispeicher abgelegt. Die Anfangsadresse dieser Befehlsfolge wird auf der Kontrollvariablen zurückgemeldet. Die Befehlsfolge dient zum schnellen Kopieren der restlichen Parameterwerte in den Freispeicher und beim Rücksprung zum Restaurieren der lokalen Variablen. Enthält die Kontrollvariable eine Adresse, so wird der aktuelle Versorgungsblock nicht weiter ausgewertet, sondern die Werte der lokalen Variablen mit Hilfe der schon existierenden Befehlsfolge gesichert.

2.5. LØKALF

Im Gegensatz zu LØKALV kann LØKALF nicht sofort die Feldbeschreibungen der angegebenen Felder retten, sondern merkt sich nur, daß beim nächsten rekursiven Aufruf eine spezielle Endebehandlung für die Felder vorzumerken ist.

Beispiel:

```
190     IV=0
200     CALL REKN (UP,IV,NEW,10)
        :
500     SUBROUTINE UP(UP1,/IV/,F,N)
510     INTEGER F(N)
520     EXTERNAL UP1
530     CALL LOKALF (F)
540     IV=IV+1
550     IF (IV.EQ.1) CALL REKN (UP1,IV,NEW,10)
        :
600     RETURN
```

Durch den LOKALF-Aufruf in Zeile 530 wird nur vermerkt, daß die Feldbeschreibung des Feldes F irgendwann gesichert werden muß. Bevor in Zeile 550 UP rekursiv aufgerufen wird, wird die Feldbeschreibung von F in den Freispeicher gesichert und dafür gesorgt, daß beim Rücksprung diese Feldbeschreibung wieder zurückgeholt wird. Erst danach darf die Feldbeschreibung für das neukreierte Feld F die alte überschreiben. Es ist also zu empfehlen, vor jedem rekursiven Aufruf LØKALF mit den Feldern als Parameter aufzurufen, bei denen sich aktuelle und formale Parameterposition unterscheiden.

Beispiel:

```
SUBROUTINE UP(UP1,F,N,F2,N2,F3,N3)
INTEGER F(N),F2(N2),F3(N3)
:
:       weil F und F3 vertauscht werden
CALL LOKALF (F,F3)
CALL REK (UP1,F3,N3,F2,N2,F,N)
:
:       weil F neu deklariert wird und F2, F3 vertauscht
CALL LOKALF (F,F2,F3)
CALL REKN (UP1, NEW,N,F3,N3,F2,N2)
```

2.6. DYNAM

Bevor irgendeine Prozedur des Montageobjekts SREK aufgerufen wird, müssen die internen Listen initialisiert werden. Zusätzlich wird die angegebene Marke als Fehlerlabel betrachtet.

Beispiel:

```
      CALL DYNAM  (&999)
      :
      :
999   STOP 'Fehler in SREK'
      END
```

2.7. VBAD

Um den aktuellen Versorgungsblock zu verändern und weiter zu übergeben, liefert VBAD die Versorgungsblockadresse des letzten REK- bzw. REKN-Aufrufs.

```
      SUBROUTINE TEST (TT, /I/,/J/)
      :
      CALL VBAD (IVB)
      CALL CALLRF (TT, IVB)
      :
      :
```

2.8. CALL

Für den nichtrekursiven Aufruf eines Unterprogramms mit einer varialben aktuellen Parameterliste steht die TAS-Prozedur CALL zur Verfügung.

Beispiel:

```
VB(1) = IKOPF (       )
VB(2) = IHVZV(P1)
VB(3) = IHVZV(P2)
CALL CALL (UP,VB)
```

Das Unterprogramm UP wird mit der Parameterliste (P1,P2) aufgerufen.

2.9. CALLRF

Kennt man nur die Versorgungsblockadresse, so verwendet man CALLRF.

Beispiel:

```
VB(1) = IKOPF (       )
VB(2) = IHVZV(P1)
VB(3) = IHVZV(P2)
IVB = IREF(VB)
CALL CALLRF(UP,IVB)
```

Dies entspricht dem Aufruf

```
CALL UP(P1,P2)
```

2.10. Liste der Äquivalenten Namen

```
REK    : RE1, RE2, RE3, RE4, RE5
REKN   : RE1N,RE2N,RE3N,RE4N,RE5N
NEW    : NEW1,NEW2,NEW3,NEW4,NEW5
REKR   : RE1R,RE2R,RE3R,RE4R,RE5R
REKNR  : RE1NR,RE2NR,RE3NR,RE4NR,RE5NR
CALL   : CALL1,CALL2,CALL3
```

3. Beschreibung des MO BOTRAN&ARRAY

Zweck: Dynamische Deklaration von Feldern, deren Länge, Typ und Namen erst zur Operatorlaufzeit festgelegt wird.

3.1. Initialisierungsaufruf: CALL DYNARR(N,LNG,&FEHL ,FELD)

In einem Freispeichergebiet wird Platz reserviert für die Beschreibung von N Feldern. LNG bezeichnet die voraussichtliche Länge der Felder (in Anzahl der Elemente). &FEHL ist ein Fehlerlabel, das im Fehlerfall angesprungen wird (vergl. DYNTST,DYNFEH,DYNRET). Ist als letzter Parameter ein Feld angegeben, so wird dieses anstelle des Freispeichers benutzt.

Werden die Prozeduren ARRDEK, ARREND, ARRVSG, ARRINF als Integer-Function aufgerufen, so liefern sie als Functions-Wert einen Fehlerschlüssel (vergl. DYNFEH).

Die dynamisch deklarierten Felder sind nur in Unterprogramme als Fortran-Felder ansprechbar.

3.2. Deklaration eines Feldes: ARRDEK(NAME,N,TYP,LADR)

Auf Name wird ein Bitmuster erwartet, das weiterhin zur Identifikation dient.

N gibt die maximale Länge des Feldes an.
Ist N=0, so wird N=LNG gesetzt. (Vergl. ARRKØR)

Typ gibt die Anzahl der Halbworte pro Feldelement an in Zweier-Potenzen:

INTEGER*2	0
INTEGER*4	1
REAL *4	1
REAL *8	2
COMPLEX*8	3
COMPLEX*16	4

Auf IADR wird die Anfangsandresse des Feldes zurückgemeldet.

Mögliche Fehlerschlüssel: 3,4,5

Bevor weitere Felder deklariert werden, muß dieses aktuelle Feld abgeschlossen werden.

3.3. Abschluß eines Feldes: ARREND(NAME,N)

Das Feld, das durch Name identifiziert wird und aktuell sein muß, wird abgeschlossen und ist dann genau N Element lang.

Der ARREND-Aufruf muß entweder in derselben Programmeinheit stehen wie der zugehörige ARRDEK-Aufruf oder falls der ARREND-Aufruf in einer untergeordneten Programmeinheit steht, muß (trotz möglicher Compilerwarnung) die entsprechende Dimensionsvariable ebenfalls auf den aktuellen Wert gesetzt werden.

Mögliche Fehlerschlüssel: 1,2,

3.4. Erstellung der Feldversorgung

Um ein deklariertes Feld an ein Unterprogramm zu übergeben benötigt man die Versorgung für das Feld und die Feldlänge.

```
ARRVSG(NAME,HVF,HVN)
```

Auf HVF wird die Hauptversorgung für das Feld, das durch Name identifiziert wird, und auf HVN die Hauptversorgung für die Feldlänge zurückgemeldet.

Mögliche Fehlerschlüssel: 1,

Beispiel:

```
CALL DYNARR(10,100,&999)
:
READ(    )NAME
CALL ARRDEK(NAME,300,1,LADR)
CALL ARRVSG(NAME,VB(3),VB(4))
VB(1) = IKØPF(0, , ,3)
VB(2) = IHVZV(NAME)
CALL     (UP,VB)
:

SUBROUTINE UP(NAME,F,N)
INTEGER F(N)
:

CALL ARREND(NAME, I)
N=I
END
```

3.5. Information über ein Feld

```
ARRINF(NAME,LADR,N)
```

liefert die Anfangsadresse und die Länge des Feldes aus.

3.6. Übergabe von Feldern an Unterprogramme

Eine bequemere Methode Felder zu übergeben, als sich den Versorgungsblock selbst aufzubauen, ist der Aufruf von Unterprogrammen mit REKN. Tritt im Versorgungsblock für REKN die Versorgung für das INTEGER FUNCTION ARRAKT auf, so wird auf dieser Parameterposition das Feld übergeben, das durch den nachfolgenden Parameter identifiziert wird. Außerdem wird die Versorgung für die Feldlänge in den Versorgungsblock eingetragen.

Beispiel:

```
INTEGER ARRAKT
CALL REKN(UP,...,ARRAKT,NAME,DUMMY,...)
:
STOP
LAUF=ARRAKT(NAME)
END
SUBROUTINE UP(UPREK,...,F,NAME,N,...)
INTEGER F(N)
```

Damit der Übersetzer ARRAKT als FUNCTION erkennt, ist ein expliziter Aufruf nötig (nur statisch). Der Aufruf von ARRAKT als INTEGER FUNCTION liefert die Anfangsadresse des Feldes.

Der Zugriff auf ein Feld in der deklarierenden Programmeinheit ist nur durch eigene Adressrechnung möglich.

Beispiel:

```
      WERT1 = 1
      WERT2 = 2.
      CALL ARRDEK('FELD',200,1,LADR)
C     Besetzung der ersten 2 Feldelemente
      CALL AS(LADR,WERT1)
      CALL AS(LADR+2,WERT2)
      :
      CALL ARREND('FELD',2)
      :
      CALL ARRINF('FELD',LADR,N)
      W1=RVAL(LADR)
      W2=RVAL(LADR+2)
```

Danach hat W1 und W2 die Werte 1. und 2. und N hat den WERT2. (Vergl. AS,RVAL.)

3.7. Aktivieren von dynamischen Kontrollen in BOTRAN&ARRAY

```
DYNTST(MODUS,/FEHL/)
```

MODUS = 0 TESTE aktivieren (Voreinstellung)
1 Teste abschalten

Im MODUS 0 wird kontrolliert, ob in

ARRDEK Namen doppelt deklariert werden,
zu viele Felder deklariert werden

ARREND Namen nicht definiert sind,
Felder schon abgeschlossen sind.

IFEHL bestimmt das Fehlerverhalten. Ist im Fehlerfall der aktuelle Fehlerschlüssel größer als IFEHL, wird das Fehlerlabel aus DYNARR angesprungen, sonst wird der Fehlerschlüssel als Functionswert übergeben.

3.8. Informieren über Fehler

```
DYNFEH(IFS,NGW)
```

liefert den aktuellen Fehlerschlüssel IFS und löscht den Fehlerindikator. Auf NGW wird die Anzahl der belegten Ganzworte übergeben.

IFS	Bedeutung
0	Kein Fehler
1	Name nicht definiert
2	Unzulässiger Zugriff auf abgeschlossenes Feld
3	Feldname schon vergeben
4	Anzahl der zu kreierenden Felder wird größer als in DYNARR angegeben.
5	Es kann kein weiterer Kernspeicher zur Verfügung gestellt werden.

3.9. Fortsetzen des Operatorlaufs nach Ansprung des Fehlerlabels

Falls nach Ansprung des Fehlerlabels nur die Prozeduren DYNKOR, DYNFEH und DYNTST aufgerufen werden, kann der Operatorlauf an der Unterbrechungsstelle fortgesetzt werden. Die ursprünglich erwünschte Leistung wird nicht erbracht. Nach CALL DYNRET (IBFS) wird hinter den Aufruf der fehlererzeugenden Prozedur gesprungen und der beliebige benutzereigene Fehlerschlüssel IBFS als Functions-Wert übergeben. So ist es möglich, bei zentraler Fehlerbehandlung trotzdem den Operatorlauf vernünftig fortzusetzen.

3.10. Verlängern von aktuellen Feldern

ARRKØR(NAME,NZUS)

Ein noch nicht abgeschlossenes Feld kann dynamisch über NMAX (von ARRDEK) hinaus verlängert werden. Es werden NZUS Elemente zusätzlich zur Verfügung gestellt.

Mögliche Fehlerschlüssel: 1,2,5

3.11. Umbenennen von Feldern

ARRUMB (NAMEALT,NAMENEU)

Mögliche Fehlerschlüssel: 1,3

3.12. Namenstausch

ARRTAU(NAME1,NAME2)

Fehlerschlüssel: 1

3.13. Felder löschen

ARRLOE(NAME)

Fehlerschlüssel: 1

Es wird nur der Verweis gelöscht. Zur Freigabe der Speicherplätze siehe ARRGRB.

3.14. Garbage Collection

ARRGRB

Fehlerschlüssel: keine

Der Aufruf ist nur sinnvoll, falls vorher Felder gelöscht oder Felder verkürzt worden sind. Die deklarierten Felder werden um die freien Speicherplätze aufgerückt. Damit ändern sich auch die Anfangsadressen der Felder.

4. Informationen über Versorgungsblöcke

4.1. INF(IVB,FA,SS,A,B,N)

IVB enthält die Versorgungsblockadresse, dessen Kopfwort ausgewertet werden soll.

FA Fehleradresse, normalerweise S&CC+8

SS Sprachschlüssel in Fortran: 1

A Typ des Function bei Subroutine: A=0

B Aufrufart

- 0 keine Aussage
- 1 Aufruf als Function
- 2 Aufruf als Subroutine

N Anzahl der Parameter

Sämtliche Parameter werden als Festkommazahlen (INTEGER*4) übergeben. INF wird als Subroutine aufgerufen.

4.2. N = IPARZ(IVB)

liefert die Parameterzahl des angegebenen Versorgungsblocks als Functions-Wert.

4.3. PINF(IVB,N,IAD,SCH,TYP,ART)

liefert sämtliche Informationen über den N-ten Parameter.

IVB enthält die Versorgungsblockadresse

N bezeichnet den Parameter über den die Informationen ausgeliefert werden soll.

IAD Programmadresse des Parameters

SCH Schlüssel

- 1 falls Feld
- 2 falls Feldelement
- 0 sonst

TYP Typ des Parameters oder Anzahl der Zeichen

ART Art des Parameters wie im Versorgungsblock angegebenen oder falls SCH ≠ 0 Adresse der Feldbeschreibung.

4.4. IAD = IPARA(IVB,N)

liefert die Adresse des N-ten Parameters

4.5. ZV = IPART(IVB,N)

liefert die Zusatzversorgung des N-ten Parameters

4.6. FB = IPARAF(IVB,N)

liefert die Zusatzversorgung ohne die beiden Schlüsselbits. Im Falle, daß der N-te Parameter ein Feld bzw. Feldelement ist, wird also die Adresse der Feldbeschreibung übergeben, sonst identisch mit IPART.

4.7. IWERT = IPARV(IVB,N)
RWERT = RPARV(IVB,N)

liefern jeweils den Wert des N-ten Parameters.

5. Erstellen von Versorgungsblöcken

5.1. VBKOPF = IKØPF(FA,A,B,N)

erstellt das Kopfwort für einen Versorgungsblock

FA	Fehleradresse falls FA=0 wird die Adresse S&CC+8 genommen
A	Typ der Function oder bei Subroutine 0
B	Aufrufart
N	Anzahl der Parameter

5.2. VB = IHVZV(X)

erstellt die Haupt- und Zusatzversorgung zum Parameter X. X darf kein Label und keine Literalkonstante sein.

5.3. VB = IHVRF(LAD)

erstellt die Hauptversorgung zu einem Parameter, dessen Adresse auf LAD übergeben wird.

5.4. VERSBL(VB,P1,P2,...)

erstellt einen Versorgungsblock zu der Parameterliste (P1,P2,...) und speichert ihn auf das Feld VB.

VB	Feldname
P1,P2...	beliebige Parameter

Beispiele:

```
INTEGER FELD(10)
INTEGER VB(5)
CALL VERSBL(VB,I,&99,'TEXT',FELD)
:
:
```

ist äquivalent zu

```
INTEGER FELD(10)
INTEGER VB(5)
VB(1) = IKØPF(0,0,2,4)
VB(2) = IHVZV(I)
CALL LABEL(&99,VB(3))
CALL SHIFT4(VBC3),160*256+24,VBC3))
CALL LITADR('TEXT',VB(4))
VB(4) = IHVRF(VB(4))
VB(5) = IHVZV(FELD)
```

6. Adressrechnung

6.1. IAD = IREF(X)

liefert die Adresse des Parameters X. X darf kein Label sein.

6.2. IWERT = IVAL(IAD)

RWERT = RVAL(IAD)

liefert das durch IAD gekennzeichnete Ganzwort aus.

6.3. AS(IAD,WERT)

weist der Adresse, auf die IAD zeigt, den angegebenen Wert zu.

Beispiel:

```
IAD = IREF(K)
CALL AS(IAD,10)
```

ist gleichbedeutend mit: K = 10

6.4. LABEL(&LAB,IAD)

weist auf IAD die Programmadresse des Labels zu.

6.5. LITADR('TEXT',IAD)

weist auf IAD die Anfangsadresse der Literalkonstante zu.

6.6. Sprung auf eine Adresse

CALL GØTØ (ADR)

Beispiel:

```
      CALL LABEL(&99,K)
      :
      CALL GØTØ(K)
oder
      CALL LABEL(&10,K)
      CALL UP( , ,K,...)
10    CONTINNE
      :
      END
      SUBROUTINE UP( , ,K,...)
      :
      :
      CALL GØTØ(K)
      :
```

Es ist somit möglich, den "Wasserfuß" zu vermeiden.

6.7. Ausführung von TAS-Befehlen

Aufruf: CALL TUE(Feld)

oder X = TUE (Feld)

Voraussetzung:

Das Feld enthält den Interncode von TAS-Befehlen.
Der letzte Befehl muß sein:

SE TUEEND

Beispiel:

Das Feld G soll mit dem TLI durchsucht werden auf den Wert von SUCH, die gefundene Adresse wird in RA übergeben.

```
      INTEGER G(1000),F(2)
      INTEGER 2 F2(4)
      EQUIVALENCE(F,F2)
      DATA F/Z2 710000EC0000,Z2 967029BC0000/
      F2(1) = F2(1) + IREF(SUCH)
      F2(2) = F2(2) + IREF(G)
      F2(4) = F2(4) + IREF(TUEEND)
      EXTERNAL TUEEND
      :
      ADR = TUE(F)
```

Mit DATA wurde auf F abgelegt: BD 0, TLI 0, R B B,SE 0

7. Typenkennung

1. ZTO(X) setzt die Typenkennung 0 auf X.
2. ZT1, ZT2, ZT3 setzt die Typenkennungen 1,2,3
3. TK = IASKTK(X) liefert die Typenkennung von X aus.

8. Wortgruppentransport

8.1. WTV(ZIEL,QUELLE,N)

transportiert N Ganzworte des Quellfeldes auf das Zielfeld. Der Aufruf entspricht den Fortran-Statements:

```
      DØ 1 I = 1,N
1     Ziel(I) = Quelle(I)
```

WTR(ZIEL,QUELLE,N)

transportiert N Ganzworte, die vor der Quelle liegen, auf Adressen, die vor dem Ziel liegen.

Beispiel:

```
      CALL WTR(Z(100),Q(150),30)
```

ist äquivalent zu

```
      DØ 1 I = 1,30
1     Z(100+1-I) = Q(150+1-I)
```

```
      WTVREF(ZADR,QADR,N)
      WTRREF(ZADR,QADR,N)
```

Im Gegensatz zu WTV und WTR werden hier keine Felder erwartet, sondern Variable , die die Ziel- und Quelladressen angeben.

8.2. VØRBES(FELD,WERT,N)

Unter Benutzung des WTV werden die ersten N Ganzworte des Feldes FELD mit WERT besetzt. Die Typenkennung wird nicht abgeprüft.

9. Schiften

9.1. Schiften eines Ganzwortes

```
CALL SHIFT4(X,ART,ZIEL,&FEHL)
```

X wird ins Register RA gebracht und dort gemäß ART geschiften. Das Ergebnis wird unter ZIEL abgelegt. Tritt ein BÜ-Alarm auf, wird nach Speicherung des Ergebnisses das Fehlerlabel angesprungen.

9.2. Schiften zweier Ganzwörter

```
CALL SHIFT8(X1,X2,ART,Z1,Z2,&FEHL)
```

X1 wird nach RA,
X2 wird nach RQ transportiert.

Nach dem Schift gemäß ART wird der Inhalt von RA nach Z1 und der Inhalt von RQ nach Z2 gebracht. Trat ein BÜ-Alarm auf, so wird das Fehlerlabel angesprungen.

ART setzt sich zusammen aus den Spezifikationen und der Anzahl der Stellen, um die geschiftet werden soll. ART = SPEZ*256+P.

P Anzahl der Stellen.

SPEZ ist die Summe der einzelnen Spezifikationen

A	128	='80'	Schift in RA
Q	64	='40'	Schift in RQ
Z	8	='08'	Schift in RAQ
AQ	192	='C0'	Schift in RA und RQ getrennt
L	32	='20'	Schift nach links
K	16	='10'	Kreisschift
R	4	='04'	mit Rundung
U	2	='02'	unabhängig von TK
B	1	='01'	zählen der aus RA geschifteten 'L'-Bits im RY

Beispiel:

X soll um 10 Bits nach links geschiftet werden, die vorn herausgeschifteten Bits sollen hinten nachgezogen werden (unabhängig von TK)

```
      Art=(128+32+16+2)*256+10
      CALL SHIFT4(X,Art,X,&1)
    1 COUTINNE
```

10. Boolsche Operationen (bitweise)

Neben den Operationen ET,VEL und AUT stehen noch IR, IR8 und ZUS zur Verfügung.

Aufrufe:

```
ERG=ET(X,Y)
ERG=VEL(X,Y)
ERG=AUT(X,Y)
ERG4=IR(X)
ERG8=IR8(X)
ERG=ZUS(MASKE,X,Y)
```

IR invertiert ein Ganzwort (entspricht dem logischen NØT).

IR8 invertiert zwei Ganzwörter, die hintereinander liegen. Das Ergebnis benötigt also auch 2 GW. (IR8 ist also als REAL*8 bzw. CØMPLX zu definieren)

ZUS setzt X und Y gemäß MALSKE zusammen.

$ERG_i = X_i$ für $MASKE_i = 0$

$ERG_i = Y_i$ für $MASKE_i = 1$

11. Teilwortoperationen

11.1. Transporte

a) Bringe Teilwort

K = BT(MASKE,X)

$K_i := X_i$ für $MASKE_i = 0$

anschließend wird K um P Stellen nach rechts geschiftet. P ist die Anzahl der 'L'-Bits die rechtsbündig in der Maske stehen.

b) Speichere Teilwort

CALL CT(MASKE,X,ZIEL)

Der dem 'L'-Feld der MASKE entsprechende Teil von ZIEL wird nicht verändert. Der Inhalt von X wird zunächst um p Stellen nach links geschiftet (P ist die Anzahl der rechtsbündig in der MASKE stehenden 'L'-Bits). Dann wird der dem 'O'-Feld der MASKE entsprechende Teil ins Ziel übernommen. X steht danach wieder in unveränderter Form zur Verfügung.

c) Bringe Halbwort

K = B2(X,MØD)

Der Wert von MØD wird zur Adresse von X addiert und das so andressierte Halbwort gebracht.

d) Speichere Halbwort

CALL C2(QUELLE,MØDQ,ZIEL,MØDZ)

Beispiel:

Rechtes Halbwort von A (INTEGER*4, REAL*4)

K = B2(A,1)

linkes Halbwort von A (INTEGER*4, REAL*4)

K = B2(A,0)

linkes Halbwort des Imaginaerteils von A (CØMPLX)

K = B2(A,2)

rechtes HW von A (INTEGER*4) soll zum rechten HW des 3. Feldelements von B(REAL*4) transportiert werden.

CALL C2(A,1, B,5) oder CALL C2(A,1, B(3),1)

11.2. Arithmetik

a) Addiere Teilwort

K = AT(MASKE,X,Y)

b) Subtrahiere Teilwort

K = SBT(MASKE,X,Y)

Sei qr die MASKE um p Stellen im Kreis nach rechts geschiftet (p ist die Anzahl der rechtsbündigen 'L'-Bits der MASKE) und nr bezeichne Y um p Stellen nach rechts geschiftet. Dann gilt

$K_i = X_i \pm nr_i$ für $qr_i = O$

$K_i = O$ für $qr_i = L$

12. Mit den Tabellensuchbefehlen können Felder ganzwortweise durchsucht werden.

Das Tabellenende wird durch eine andere TK gekennzeichnet.

Bezeichnungen:

FELD: Angabe eines Feldes/Feldelementes

WERT: Angabe des Suchwortes

MASKE: Angabe eines Bitmusters, es werden nur die Bits beim Suchvorgang beachtet, die im Nullfeld der MASKE liegen.

DEHNUNG: Angabe der Schrittweite (in HW), da nur Ganzwörter betrachtet werden, ist Dehnung als gerade Zahl anzugeben. Im Normalfall ist DEHNUNG=2, d. h. jedes GW wird betrachtet.

INDEX: Rückmeldung als wievieltes GW das gesuchte Wort gefunden wurde. Wird ein Feld von Anfang an durchsucht, so wird der gefundene Feldindex zurückgemeldet.

&FEHL: Angabe eines Fehlerlabels, es wird angesprungen, falls kein Wort gefunden wird.

AADR: Der Wert von AADR wird als Anfangsadresse der Tabelle interpretiert.

IADR: Adresse des gefundenen GW.

12.1. TLI: Tabelle durchsuchen auf Identität

Aufruf:

```
CALL TLI(FELD,WERT,INDEX,&FEHL)
INDEX=ITLI(FELD,WERT)

CALL TLIR(AADR,WERT,IADR,&FEHL)
LADR=ITLIR(AADR,WERT)
```

Der Suchvorgang wird abgebrochen, sobald ein Wort aus der Tabelle eine andere TK als 'WERT' hat. Falls kein Wort gefunden wird, so wird der Index des ersten Elements hinter der Tabelle ausgeliefert, bzw. die Adresse des ersten GW hinter der Tabelle.

12.2. TLD: Tabelle durchsuchen mit Dehnung

Suchkriterium: Feld (Index) ≥ Wert

Aufruf: CALL TLD(FELD,WERT,DEHNUNG,INDEX,&FEHL)

INDEX=ITLD(FELD,WERT,DEHNUNG)

CALL TLDR(AADR,WERT,DEHNUNG,IADR,&FEHL)

IADR=ITLDR(AADR,WERT,DEHNUNG)

12.3. TDM: Tabelle durchsuchen mit Dehnung und Maske

Suchkriterium: Feld(Index) = Wert für die Bits im Nullfeld der Maske

Aufruf: CALL TDM(FELD,WERT,MASKE,DEHNUNG,INDEX,&FEHL)

INDEX=ITDM(FELD,WERT,MASKE,DEHNUNG)

CALL TDMR(AADR,WERT,MASKE,DEHNUNG,IADR,&FELD)

IADR=ITDMR(AADR,WERT,MASKE,DEHNUNG)

12.4. TMAX: Tabelle durchsuchen auf Maximum

Das Tabellenende wird durch eine TK ungleich dem ersten Tabellenwort gekennzeichnet.

Aufruf: CALL TMAX(FELD,MASKE,DEHNUNG,WERT,INDEX)

INDEX=ITMAX(FELD,MASKE,DEHNUNG)

CALL TMAXR(AADR,MASKE,DEHNUNG,WERT,IADR)

IADR=ITMAXR(AADR,MASKE,DEHNUNG)

Auf WERT wird das gefundene Maximum zurückgemeldet.

TMIN: Tabelle durchsuchen auf Minimum (siehe TMAX)

12.5. TLOG: Tabelle logorithmisch durchsuchen

Aufruf: CALL TLOG(FELD,WERT,MASKE,LÄNGE,WERT2,INDEX,&FEHL)

INDEX=ITLOG(FELD,WERT,MASKE,LÄNGE)

CALL TLOGR(AADR,WERT,MASKE,LÄNGE,WERT2,IADR,&FEHL)

ITLOGR(AADR,WERT,MASKE,LÄNGE)

Auf LÄNGE wird die Tabellenlänge in GW angegeben.

Auf WERT2 wird das gefundene Wort zurückgemeldet.

Wird nur ein Wort Wert gefunden: Sprung auf das Fehlerlabel (WERT2 und INDEX werden zurückgemeldet).

Wir kein Wort = Wert gefunden: Sprung auf das Fehlerlabel.

13. Zeichenverarbeitung

Es können Zeichen, die aus 4, 6, 8 oder 12 Bits bestehen, bearbeitet werden. Aus einem Feld können sequentiell Zeichen gelesen werden (d. h. in einem Ganzwort stehen 4, 6, 8 oder 12 Zeichen). Entsprechend können auch Zeichen geschrieben werden.

13.1. BNZ - Bringe nächstes Zeichen

Aufruf: `ZEICH=BNZ(FADR,ART)`

oder `ZEICH=BNZ(X)`

Dabei bedeutet:

- FADR: Variable, die eine Feldadresse beinhaltet
- ART: Variablen mit dem Wert f*4096+N
- f : Anzahl der Bits pro Zeichen (4,6,8,12)
- N : Laufende Nummer eines Zeichens in einem Ganzwort (0 bis (48/f-1))

Bei sequentiellem Lesen wird N um 1 erhöht. Falls N > 48/f-1 wird N auf Null gesetzt und der Wert von FADR um 2 erhöht. Zusätzlich werden die Werte von FADR um 2 erhöht. Zusätzlich werden die Werte von FADR und ART als Voreinstellung für den nächsten BNZ-Aufruf übernommen. Wird BNZ nur mit einem Parameter aufgerufen, so wird die Voreinstellung benutzt und eine neue Voreinstellung erzeugt. Der angegebene Parameter ist bedeutungslos und wird auch nicht verändert.

Beispiel:

Aus dem Feld F soll ab dem dritten Zeichen zehn Zeichen gelesen und einzeln abgespeichert werden.

```
      IADR=IREF(F)
      IART=8*4096+2
      Z(1)=BNZ(IADR,IART)
      DO 10 I=2,10
10    Z(I)=BNZ(X)
```

13.2. CNZ-Speichere nächstes Zeichen

Aufruf: `CALL CNZ(FADR,ART,ZEICH)`

oder `CALL CNZ(ZEICH)`

Das Zeichen wird gemäß FADR und ART abgespeichert. Erfolgt der Aufruf nur mit einem Parameter, so wird die Voreinstellung benutzt.

14. SSR - Befehle

Aufruf: CALL SSR(ART,REG,VB)

ART bestimmt den SSR-Befehl, der ausgeführt werden soll.

ART=LADR*256+RADR

wobei LADR der Linksadressteil und RADR der Rechtsadressteil ist.

REG ist ein Feld, in dem die Registerstände nach dem SSR-Befehl zurückgemeldet werden, wie der QCR-Befehl die Register abspeichert.

VB ist der Versorgungsblock für den SSR-Befehl

Beispiel:

SSR 4 32 zur Bestimmung der operatorlaufrelativen Nettozeit in interner Darstellung.

```
      INTEGER*2 VB(2)
C     FEHLERADRESSE
      CALL LABEL(&99,IADR)
      VB(1)=IADR
      VB(2)=5
      INTEGER REG(6)
      IART=4*256+32
      CALL SSR(IART,REG,VB)
      ITIME=REG(2)
      :
99    STOP'FEHLER'
      END
```

15. Dreiecksmatrizen

In Verbindung mit BØDAT(STRANS, etc.) gibt es folgende Möglichkeit, große Dreiecksmatrizen (ohne Hauptdiagonale) stückweise zu bearbeiten. Die Dreiecksmatrix A(i,j) mit i > j sei folgendermaßen abgespeichert (Gebiet, Datei):

A(2,1),A(3,1),...,A(n,1),A(3,2),A(4,2),...

Im Kernspeicher wird A auf einem eindimensionalen Feld F segmentweise verarbeitet. Die zu dem Element A(i,j) gehörige Segmentnummer SEGNR und den entsprechenden Index im Feld F erhält man durch den Aufruf:

CALL SINDEX(J,I,SEGNR,INDEX)

Die Umrechnung von Segmentnummer und Index auf die Indices der Dreiecksmatrix erfolgt durch

CALL QINDEX(SEGNR,INDEX,J,I).

Die Segmentlänge wird eingestellt durch:

CALL SEGLNG(LNG)

wobei LNG die Länge in Elementen bezeichnet. Da die Segmente von eins an gezählt werden, ergibt sich die Umrechnung der Segmentnummer in die Blocknummer bei STRANS als:

BLNR=(SEGNR-1) LNG/128.

Beispiel: siehe nächste Seite

Beispiel:

Bestimmung des maximalen Elements

```
      COMMON/CACHTO/F(2049)
      INTEGER F
      :
      CALL SEGLNG(2048)
      MWERT=-2**46-1
      MSEG = 0
      IBLNR= -16
      CALL ZT2(F(2049))

      DØ  10  ISEG=1,NN
      IBLNR=IBLNR+16
      CALL STRANS(1,F,IBLNR,-16)
      CALL TMAX (F,0,2,IWERT,INDEX)
      IF(MWERT.GE.IWERT)GØTØ10
      MWERT=IWERT
      MSEG=ISEG
      MIND=INDEX
10    CONTINUE
      CALL QINDEX(MSEG,MIND,J,I)
      WRITE(6,1)I,J,MWERT
 1    FØRMAT(' DAS MAXIMUM LIEGT AUF A('I5,','15')','UND HAT DEN WERT:',I5)
      STOP
      END
```

Nr. 7301: R. Mannshardt, K.-H. Mohn, H. Münch, P. Pottinger
Einführung in die Benutzung des Teilnehmer Rechensystems TR 440
2. geänderte Auflage (vergriffen)

Nr. 7302: K.-H. Mohn
Über einige Anwendungen des Computers in der Medizin

Nr. 7303: R. Buchmann
BODAT, ein schnelles und platzsparendes System zur Datenmanipulation und -speicherung in ALGOL 60 und FORTRAN

Nr. 7304: M. Hauenschild
Ansätze zur komplexen Kreisarithmetik

Nr. 7305: R. Buchmann
RB&QUELLHALT, ein TR440-Datenbanksystem zur platzsparenden Quellhaltung auf Datenträgern mit direktem Zugriff (LFD,WSP)

Nr. 7306: 6. Jahresbericht des Rechenzentrums (1.7.1972 bis 31.12.1973)

Nr. 7401: R. Buchmann
Der Systemoperator BO&BS3OP
Messungen und Steuerungen des Betriebssystems auf Operatorebene

Nr. 7402: R. Mannshardt
Herleitung und Prüfung spezieller Runge-Kutta-Verfahren mit impliziten Rechenschritt

Nr. 7403: R. Buchmann, H. Wupper
Unzulänglichkeiten des TR 440 Programmiersystems und ihre Umgehung

Nr. 7404: R. Green, K.-H. Mohn
Quellbezogene FORTRAN Optimierungen für den Compiler des TR 440

Nr. 7405: R. Buchmann
BODAT, ein schnelles und platzsparendes System zur Datenmanipulation und -speicherung in ALGOL 60 und FORTRAN (2. ergänzte Auflage)

Nr. 7501: R. Buchmann
Zur Theorie der Montage von Programmoduln

Nr. 7502: 7. Jahresbericht des Rechenzentrums (1.1. bis 31.12.1974)

Nr. 7503: H.-D. Sander
BOTRAN, eine Fortran Spracherweiterung durch Code-Prozeduren