KERNFORSCHUNGSZENTRUM

KARLSRUHE

Oktober 1971 KFK 1488

Institut für Reaktorentwicklung

Vorteile der Dynamisierung von Argumentenlisten und deren Verwirklichung in Fortran und Icetran

A. Pee, U. Schumann

GESELLSCHAFT FÜR KERNFORSCHUNG M. B. H.

KARLSRUHE

GESELLSCHAFT FÜR KERNFORSCHUNG M. B. H.
KARLSRUHE

KERNFORSCHUNGSZENTRUM KARLSRUHE

Oktober 1971 KFK 1488

Institut für Reaktorentwicklung

Vorteile der Dynamisierung von Argumentenlisten und deren Verwirklichung in Fortran und Icetran

A. Pee
U. Schumann

Gesellschaft für Kernforschung m.b.H.,Karlsruhe

Zusammenfassung

Es wird vorgeschlagen, in den höheren Programmiersprachen Argumentenlisten mit variabler Argumentenanzahl zuzulassen. Es soll möglich sein, die Argumentenliste, mit der ein Unterprogramm aufgerufen werden soll, unmittelbar zuvor während der Programmausführung zusammenzustellen und dann im aufgerufenen Unterprogramm abzählen zu können, wieviele Argumente übergeben werden. Die Vorteile einer derartigen Dynamisierung werden diskutiert und es werden die Voraussetzungen für eine entsprechende Verwirklichung angegeben.

Speziell für FORTRAN- und ICETRAN-Programme des IBM 360-ØS werden zwei Assemblerroutinen (NUMBRA und ARLIST) beschrieben, mit denen diese Dynamisierung realisiert werden kann.

Abstract

It is proposed for higher programming languages to allow the use of argument lists with a variable number of arguments. This means that it should be possible to compose the argument list with which a program is to be called just before executing the call and that then the number of arguments passed can be enumerated in the subprogram called. The advantages of such dynamic argument lists are discussed and their preconditions are derived.

Especially for FORTRAN and ICETRAN programs in IBM 360-ØS, two assembler routines (NUMBRA and ARLIST) are described. These routines allow the use of dynamic argument lists.

Inhaltsverzeichnis

		Seite
1.	Situation	1
2.	Vorteile einer Dynamisierung der Argumentenlisten	2
3.	Vorbedingungen für eine Dynamisierung	4
4.	Anwendungsvorschriften der Unterprogramme NUMBRA und ARLIST zur Verarbeitung dynamischer Argumentenlisten	6
5.	Programmlogik der Unterprogramme NUMBRA und ARLIST	9
6.	Anwendungsbeispiele	12
	Literaturangaben	16
	Anhang 1 und 2	18
	Liste der Assemblerprogramme NUMBRA und ARLIST	A3-1

1. Situation

Es ist eine bekannte und vielgenutzte Technik, von einem Programm ein Unterprogramm aufzurufen und dabei diesem eine Liste mit Argumenten zu übergeben. Bei der Übersetzung des Unterprogramms wird diesem in einer besonderen Anweisung mitgeteilt, welche Argumente dieses Unterprogramm erwarten kann; dies geschieht mit sogenannten Dummy-Argumentenlisten. In allen wichtigen höheren Programmiersprachen,wie z.B. FORTRAN, ist die Anzahl der Argumente in der Argumentenliste eines Unterprogramms eine Größe, die zur Zeit der Programmerstellung festgelegt wird. Es ist eine Forderung der Sprachsyntax, daß jedes Unterprogramm stets mit genau allen vorgesehenen Argumenten aufgerufen wird.

Diese Forderung wird jedoch bei der Mehrzahl der Sprachimplementierungen nicht überprüft. Eine derartige Prüfung ist zur Übersetzungszeit prinzipiell nicht möglich, da einzelne Unterprogramme unabhängig von dem aufrufenden Programm übersetzt werden können. Die Überprüfung könnte erst zum Zeitpunkt des "Linkens"[1] der einzelnen Unterprogramme zu einem ausführbaren Modul oder sogar erst beim Aufruf des Unterprogramms zur Ausführungszeit vorgenommen werden. Da diese Überprüfung jedoch zu keiner Zeit vorgesehen ist, kann es passieren, daß ein Unterprogramm versehentlich mit zu wenigen oder auch mit zu vielen Argumenten aufgerufen wird, wobei sich dieser Fehler erst in fehlerhaften Ergebnissen oder einem Abbruch der Programmausführung aufgrund unsinniger Argumentadressen zeigt. Dieser Fehler tritt in der Programmierpraxis sehr häufig auf. Zwar werden sorgfältige Programmierer von Unterprogrammen wenn möglich versuchen, die Richtigkeit der übermittelten Argumentwerte zu prüfen. Ihnen wird jedoch von der Sprache keine Möglichkeit gegeben, die Anzahl der Argumente selbst festzustellen und mit dem Sollwert zu vergleichen.

Zum Druck gegeben am 22.10.1971

Die Situation ist also so, daß die Sprachdefinitionen die Übereinstimmung der Anzahl der Argumente beim Aufruf mit der erwarteten Anzahl der Argumente des Unterprogramms fordern und so auf einige, mögliche Flexibilität verzichten, die Sprachimplementierungen die Einhaltung dieser Forderung jedoch nicht garantieren. Dies führt zu starren und unsicheren Programmen.

2. Vorteile einer Dynamisierung der Argumentenlisten

Eine Dynamisierung der Argumentenlisten muß zwei Dinge ermöglichen:

- Feststellung der Anzahl der übermittelten Argumente
- Aufbau der Argumentenliste, mit der ein Unterprogramm aufgerufen wird, zur Ausführungszeit.

2.1 Vorteile der Feststellung der Anzahl der übermittelten Argumente

Es können folgende Vorteile genannt werden:

- Wie schon oben erwähnt, kann eine Nichtübereinstimmung der Argumentenlisten zu unvorhersagbaren Fehlern führen. Wenn eine Routine bereitsteht, die es erlaubt, die Anzahl der Argumente beim Aufruf zur Ausführungszeit zu ermitteln, können diese Fehler im Unterprogramm erkannt werden und es können hierfür angemessene Reaktionen vorgesehen werden.

Befreit man zudem die Sprache von der starren Forderung der festen Übereinstimmung der Argumentenanzahlen, so gewinnt man weitere Vorteile:

- Für fehlende Argumente können Standardwerte verwendet werden. Hierzu ist es erforderlich, festzustellen, welche Argumente als fehlend anzusehen sind. Es können z.B. die ersten, die letzten oder jedes n-te sein. Es hat sich bewährt, die letzten Argumente als fehlend anzusehen.
- Mit der aktuellen Anzahl der Argumente kann die Programmausführung gesteuert werden.

- Häufig verwendete Bibliotheksroutinen können für neuere Anwendungen, die zusätzliche Argumente erfordern, erweitert werden, ohne daß der Name des Unterprogramms zu ändern wäre. Beim Aufruf der Routine mit der alten, verkürzten Argumentenliste kann die Routine mit Standardwerten für die neuen Argumente arbeiten. Dieser spezielle Vorteil ist beispielsweise im Programmsystem MAPLIB [8] von besonderer Bedeutung, wie noch erläutert wird (Kap.6.1).

2.2 Vorteile des dynamischen Aufbaues von Argumentenlisten

Wenn man von den strengen Sprachforderungen abgeht und es möglich ist, die Argumentenliste, mit der ein Unterprogramm aufgerufen werden soll, zur Ausführungszeit zusammenstellen zu lassen, so werden damit folgende Vorteile gewonnen:
Unterprogramme können in flexiblerer und doch sicherer Weise programmiert und genutzt werden. Wenn das Unterprogramm die Anzahl der Argumente ermittelt, kann so dessen Ausführung gesteuert werden bzw. mit den vorgesehenen Standardwerten gerechnet werden.

Grundsätzlich läßt sich dieser Vorteil, z.B. in Fortran, auch anders erreichen. Man kann die Aufrufe mit verschiedenen Argumentenlisten fest programmieren und zu dem gewünschten Aufruf durch verschiedene Abfragen und Sprungbefehle gelangen; z.B.:

```
      INTEGER*2 A(5)
      ...
   20 GOTO(1,2,3,4,5),IA
    1 CALL SUB(A(1))
      GOTO 10
    2 CALL SUB(A(1),A(2))
      GOTO 10
      ...
    5 CALL SUB(A(1),A(2),A(3),A(4),A(5))
   10 CONTINUE
```

Wenn die Zahl der möglichen Varianten groß ist, ist diese Art des Aufrufs sowohl für den Programmierer als auch für das

ausführbare Programm sehr aufwendig. Hier ist also eine dynamische Methode wünschenswert.

Eine dynamische Erzeugung bietet zudem den Vorteil, regelmäßig aufgebaute Argumentenlisten loopartig zu erzeugen. Hierfür werden unten noch Beispiele gegeben.

3. Vorbedingungen für eine Dynamisierung

Die Dynamisierung der Argumentenlisten ist nur möglich, wenn einige Voraussetzungen gegeben sind. Diese seien im folgenden diskutiert:

- Die Argumentenliste muß mit einer Kennzeichnung versehen sein, die die Bestimmung der Anzahl der Argumente zur Ausführungszeit überhaupt erst ermöglicht. Im IBM-360-OS-System geschieht dies mit dem Vorzeichenbit, wie noch erläutert wird.

- Ein Unterprogramm kann nur dann mit weniger Argumenten, als maximal erwartet, aufgerufen werden, wenn in dem Unterprogramm auf die fehlenden Argumente nicht zugegriffen wird. D.h. aber nicht nur, daß keine Anweisung des Unterprogrammkörpers, in dem das Argument auftritt, ausgeführt werden darf, sondern auch, daß beim Aufruf nicht die Werte der Argumente übernommen werden ("CALL BY VALUE") sondern nur deren Adressen ("CALL BY NAME"). Bei fehlenden Argumenten in der Argumentenliste werden an deren Stelle im Kernspeicher irgendwelche Bitkombinationen stehen, die zwar als Adressen übernommen werden dürfen (es sei denn, die so interpretierten Kernspeicherplätze liegen nicht in dem Bereich, der dem Programm zugeteilt wurde), aber es darf nicht zu diesen - in der Regel unsinnigen - Adressen zugegriffen werden.
  In Fortran wird CALL BY NAME stets für dimensionierte Felder vorgesehen; hierbei treten also keine Probleme auf. Nicht dimensionierte Variable werden dagegen in der Mehrzahl der Implementierungen BY VALUE übernommen.

Hier muß also das CALL BY NAME irgendwie erzwungen werden. Im IBM-360-Fortran IV kann dies durch Einklammern der Argumente in der Unterprogrammanweisung in Schrägstrichen [2] erreicht werden.

Beispiel:

CALL BY VALUE:	CALL BY NAME:
SUBROUTINE ABC (X,Y)	SUBROUTINE ABC(/X/,/Y/)

Wenn diese Syntaxerweiterung des ASA-Fortran in anderen Implementierungen nicht gegeben ist, so kann jedoch stets eine an sich nichtdimensionierte Variable in einem mit (1) dimensionierten Dummy-Argument erwartet werden und so dieses Argument BY NAME übernommen werden. Beispiel:

```
SUBRØUTINE ABC  (XD, YD)
DIMENSION XD(1), YD(1)
.......
X = XD(1)
Y = YD(1)
```

- Methoden der Dynamisierung der Argumentenlisten sind stets abhängig von Sprache, Übersetzer und eventuell sogar vom Release des Betriebssystems. Sie müssen in der jeweiligen Assemblersprache programmiert werden und die Übersetzer-Konventionen für den Unterprogrammaufruf beachten.
  Die folgenden Angaben gelten daher nur für FORTRAN-IV (E,G u.H) des IBM-360-ØS, Release 19 und früher (bei FORTRAN-E gesichert nur für Release 14) [3] sowie für ICETRAN [5], die Fortran-Dialekt-Sprache des Systems ICES [4] auf dem IBM-360-ØS.

Für die Realisierung der Dynamisierung unter diesen Vorbedingungen werden im folgenden zwei in Assembler geschriebene Unterprogramme beschrieben:

NUMBRA	zur Bestimmung der Anzahl der übergebenen Argumente
ARLIST	zum Aufbau der Argumentenlisten zur Ausführungszeit

Bevor die Programmlogik dieser Unterprogramme im Detail beschrieben wird, sollen zunächst deren Anwendungsvorschriften angegeben werden.

4. Anwendungsvorschriften

4.1 Feststellung der Zahl der Argumente mit NUMBRA

NUMBRA ist eine INTEGER FUNCTION mit einem Dummy-Argument. Das Argument dient lediglich zur Kennzeichnung von NUMBRA als Funktion. Es wird an diesem Argument nichts geändert, so daß jede beliebige Programmvariable eingesetzt werden kann.

```
Beispiel: MAIN                          SUBROUTINE SUB (U,V,W,X,Y,Z)
            .                               .
            .                               .
            .                               .
          CALL SUB(A,B,C)               NA  =  NUMBRA(DUM)
            .                               .
            .                               .
            .                               .
```

Hier würde NUMBRA den Wert 3 als Ergebnis liefern.

In ICETRAN-Programmen hätte SUB auch mit den Befehlen LINK und BRANCH [5] aufgerufen werden können.

4.2 Aufbau einer Argumentenliste mit ARLIST

ARLIST kann sowohl als Funktion wie auch als Subroutine aufgerufen werden. Das Ergebnis des Funktionsaufrufs sowie der Argumente kann von beliebigem Typ (REAL, INTEGER, LOGICAL) und beliebiger Länge (1,2,4,8 Bytes sowie bei Argumenten auch alle anderen Längen) sein. Beim Aufruf von ARLIST als Funktion ist der Typ im aufrufenden Programm zu deklarieren. ARLIST kann selbst mit einer variablen Argumentenzahl aufgerufen werden.

Anzahl der Argumente: $n = 3*m+2$, $m \geq 0$
m ist die Anzahl der Argumentengruppen, die im folgenden durch Unterstreichung gekennzeichnet werden. Die minimale Argumentenzahl beträgt also 2 und kann in Schritten von 3 vergrößert werden.

Die allgemeine Form des Aufrufs hat folgendes Aussehen:
....ARLIST(feld,$\underline{n_1,i_1,a_1}$, $\underline{n_2,i_2,a_2}$,....,$\underline{n_m,i_m,a_m}$,ext)
Die Punkte deuten an, daß vor ARLIST ein CALL, ein Gleichheitszeichen oder ein anderer Operator stehen kann, je nach Aufruf als Funktion oder Subroutine. Die klein geschriebenen Texte bezeichnen hier Fortran-Variablen, die im folgenden erläutert werden:

-Typ und Länge der Variablen

ext — ist der Name des Unterprogramms, das aufgerufen werden soll; er ist im aufrufenden Programm als EXTERNAL zu deklarieren

feld — ist der Name eines Feldes vom Typ INTEGER, REAL, LOGICAL oder REAL*8; das Feld muß mindestens 4*m Bytes lang sein

n_j — INTEGER, Konstante, Variable oder arithmetischer Ausdruck

i_j — INTEGER, Konstante, Variable oder arithmetischer Ausdruck

a_j — beliebiger Typ, Länge n_j*i_j

- ARLIST verändert nur die in feld gespeicherten Werte
- Wirkung:

Der Aufruf von ARLIST in der obengenannten Form hat in der Regel zur Wirkung, daß das Unterprogramm ext mit einer von ARLIST zusammengestellten Argumentenliste aufgerufen wird.
Falls ARLIST mit nur 2 Argumenten aufgerufen wird, wird das Unterprogramm ext ohne Argumente aufgerufen. Desgleichen gilt, falls alle $n_i=0$ sind.
Falls ARLIST mit mehr als 2 (also 5,8,.....) Argumenten aufge-

rufen wird, wird das Unterprogramm ext mit einer Argumentenliste aufgerufen, deren Aufbau unten beschrieben ist. Bekanntlich (siehe Anhang 1 und [3]) werden bei einem Unterprogrammaufruf die Anfangsadressen der Argumente übergeben. ARLIST erzeugt eine Argumentenliste, die folgende Adressen (in Bytes) enthält:

(A(x) = Adresse des ersten Bytes von x):

$(A(a_1), A(a_1)+i_1, A(a_1)+2*i_1, \ldots\ldots, A(a_1)+(n_1-1)*i_1,$

$A(a_2), A(a_2)+i_2, A(a_2)+2*i_2, \ldots., A(a_2)+(n_2-1)*i_2,$

$\ldots\ldots\ldots$

$A(a_m), A(a_m)+i_m, A(a_m)+2*i_m, \ldots., A(a_m)+(n_m-1)*i_m)$

Diese Adressen werden in dem Feld feld abgelegt. Da jede Adresse ein 4 Byte langes Wort benötigt, muß feld also, wie oben bereits formal angegeben, soviel Platz in Wörtern bereitstellen, wie ext Argumente erwartet. n_j, i_j, a_j seien über diese formale Definition hinaus noch weiter erläutert:

a_j ist das Argument, das als erstes Argument der Argumentengruppe j an ext übergeben werden soll.

i_j für $n_j > 1$ gibt i_j das Increment an, um das die Adresse des nächsten Arguments größer ist als die des vorhergehenden in derselben Argumentengruppe. Sollen beispielsweise die in einem REAL-feld a_j abgespeicherten Werte nacheinander als Argumente auftreten, so ist $i_j=4$, denn ein REAL-Wort umfaßt 4 Bytes. Entsprechend wäre für

a_j = LOGICAL*1	i_j	= 1
INTEGER*2	i_j	= 2
INTEGER,REAL,LOGICAL	i_j	= 4
REAL*8	i_j	= 8
usw.		

Soll jedoch nicht jeder der aufeinanderfolgenden Werte eines REAL-Feldes a_j sondern jeder zwölfte Wert als Argument an ext

übergeben werden, so muß

$$i_j = 4*12 = 48 \text{ sein.}$$

Für $n_j \leqq 1$ ist der Wert von i_j beliebig.

Falls ARLIST mit nur einem Argument aufgerufen wird oder falls die Zahl der Argumente nicht der Formel n = 3*m+2 entspricht, liegt ein Fehler vor. In diesem Fall wird sofort in das aufrufende Programm zurückgekehrt.
Zur weiteren Erläuterung der Wirkung und Form des Aufrufs sei auf die folgenden Beispiele verwiesen.

Warnung: ARLIST überprüft nicht, ob feld einen ausreichend großen Speicherbereich bezeichnet.

5. Programmlogik

5.1 Allgemeines

Zum Verständnis der Programmlogik beider Unterprogramme ist es erforderlich, zu wissen, wie die Argumentenliste im Kernspeicher angelegt wird und wie die Register und die Save-Area verwendet werden. Hierüber gibt Anhang 1 Auskunft. Siehe auch [3,6].
Wesentlich ist hier insbesondere, daß die Argumentenliste die Adressen der Kernspeicherplätze enthält, an denen die Werte der Argumente abgelegt sind. Die letzte Adresse in der Argumentenliste ist dadurch von den vorhergehenden unterschieden, daß das erste Bit gleich 1 und nicht wie sonst gleich 0 ist. Diese Kennzeichnung macht die Feststellung der Zahl der Argumente überhaupt erst möglich.
Register 1 zeigt auf den Beginn der Argumentenliste.

5.2 NUMBRA

Für die Erläuterung der Programmlogik seien zunächst zwei Arbeitsbegriffe definiert:

Der "CALLY" sei das Programm, das NUMBRA aufrief.
Der "CALLER" sei das Programm, das den CALLY aufrief (entweder direkt mit CALL oder über den ICES-Systemkern mit BRANCH oder LINK).

NUMBRA muß sich als erstes die Registerinhalte beschaffen, die beim Aufruf des CALLY gültig waren. Diese befinden sich in der Save-Area des CALLER.
Anschließend muß festgestellt werden, ob CALLY überhaupt eine Argumentenliste übermittelt bekam. Dies wird festgestellt, indem abgefragt wird, ob 10 Bytes vor der Return-Adresse des CALLY ein Befehl existiert, der die Adresse der Argumentenliste in Register 1 lädt. Dies ist ein LA-Befehl dessen ersten 3 Halbbytes die hexadezimalen Werte '411' enthalten müssen.(Dies ist genau die Stelle an der NUMBRA vom Compiler und eventuell sogar dessen Release abhängig ist; gegenwärtig ist diese Art der Argumentenübermittlung jedoch für alle Fortran-Compiler des IBM-360-OS-Systems gültig). Wird der genannte Befehl nicht gefunden, so liefert NUMBRA den Wert 0 als Ergebnis (in Register 0). Andernfalls ist nun bekannt, daß Register 1 auf die Argumentenliste zeigt. Dort wird nun in einem Loop abgefragt, welches wievielte Wort das Vorzeichenbit '1' enthält. Diese Zahl entspricht dann der Anzahl der Argumente, die als Ergebnis in Register 0 an den CALLY übermittelt wird.

Besondere Vorkehrungen sind zu treffen für den Fall, daß der CALLER den CALLY mit einem LINK- oder BRANCH-Befehl aufgerufen hat. In diesem Fall wird die Kontrolle nicht direkt an den CALLY sondern über ein ICES-Systemkernprogramm QQFINCH sowie ein Load-Modul-Zwischenprogramm übergeben. Dieses Zwischenprogramm zeichnet sich dadurch aus, daß es ab der Adresse 5 relativ zum Entry-Point den Text 'SETUP,b'enthält. Dies ist der Ort, an dem üblicherweise der Programmname abgespeichert wird. NUMBRA kann also abfragen, ob das Namensfeld diese Zeichenkette enthält und daraus sicher schließen, ob der CALLY über einen CALL oder über einen der Befehle LINK bzw. BRANCH aufgerufen wurde. Im letzteren Fall sind

die beim Aufruf des CALLY gültigen Registerinhalte in einer höheren SAVE AREA zu suchen.

5.3 ARLIST

Da ARLIST selbst mit einer variablen Anzahl von Argumenten aufgerufen wird, muß ARLIST diese Anzahl zunächst bestimmen. Dies geschieht nach dem gleichen Prinzip wie bei NUMBRA, jedoch ist dies hier etwas einfacher, da ARLIST nicht mit LINK oder BRANCH aufgerufen werden kann und stets erwartet wird, daß mindestens ein Argument übermittelt wurde.

Ist die Anzahl der Argumente gleich 1 oder entspricht sie nicht der Formel n=3*m+2, so wird sofort in das rufende Programm zurückgesprungen. Ist die Anzahl gleich 2, so wird zur Marke NULLARG verzweigt, wo das Unterprogramm ohne Argumente aufgerufen wird.

Ist die Anzahl größer als 2, so wird die Argumentenliste gemäß Kap.4.2 mit Loops über J=1,m (m, die Anzahl der Argumentengruppen) und I=1,NJ (NJ,die Anzahl der zu erzeugenden Argumente pro Argumentengruppe) aufgebaut. Hierbei wird die Summe $K = \sum_{i=1}^{m} n_i$ gebildet. K ist die Anzahl der neuen Argumente. Ist K größer als 0, so wird das Unterprogramm mit der neuen Argumentenliste aufgerufen, nachdem zuvor das Vorzeichenbit des letzten Arguments auf 1 gesetzt wurde. Ist K gleich 0, d.h.waren alle n_i=0, so wird zu NULLARG verzweigt (siehe oben).

Um ARLIST sowohl als Funktion wie als Subroutine aufrufen zu können, werden das allgemeine Register 0 sowie die Gleitkommaregister 0 und 2 nicht verändert.

Für die Verwendung in ICETRAN-Programmen war es zudem erforderlich, das allgemeine Register 4 nicht zu verändern, da hier die COMMON-Adresse erwartet wird.

6. Anwendungsbeispiele

6.1 NUMBRA

Die Anwendung von NUMBRA ist dermaßen simpel, daß weitere Beispiele zur Erläuterung der Anwendungsvorschriften überflüssig sind. Stattdessen soll hier erläutert werden, wie NUMBRA in einer speziellen Anwendung die Sicherheit und Flexibilität eines in Fortran geschriebenen Programmsystems entscheidend erweiterte:

In MAPLIB [7], einem Programmsystem zur Bereitstellung von Stoffdaten für Rechenprogramme, wird NUMBR$ [8], eine ältere und einfachere Version von NUMBRA, zur Feststellung der Zahl der Argumente benutzt. Außer zur Überprüfung der Zahl der angelieferten Argumente zur Vermeidung von Fehlern, wird hier in einigen Stoffdatenfunktionen aufgrund der mit NUMBR$ ermittelten Zahl entschieden, ob für die letzten, evtl. fehlenden Argumente Standardwerte verwendet werden sollen. Wird beispielsweise die Stoffdatenfunktion ESH2OV, die die Entropie von Wasserdampf berechnet [9] und die die Argumente T (Temperatur)und P(Druck) erwartet, mit nur einem Argument aufgerufen, so wird für den Druck der zu dieser Temperatur T gehörige Sättigungsdruck berechnet und statt eines explizit angegebenen Wertes für P verwendet. Anhang 2 zeigt dieses Beispiel im Detail.

Da die Namensgebung der Stoffdatenfunktionen in MAPLIB nach strengen Normen [10] erfolgt, kann für den Fall, daß eine Funktion aufgrund neuerer Messungen nun zusätzlich einen weiteren Parameter als Argument berücksichtigen soll, diese Funktion nicht mit einem anderen Namen versehen werden. Andererseits sollen bei einer Erweiterung einer Funktionsargumentenliste nicht alle bisherigen Benutzerprogramme verändert werden. Auch hier wurde mit NUMBR$ das Problem gelöst: Wird festgestellt, daß die Funktion mit der alten Argumentenanzahl aufgerufen wird, wird nach der alten Methode oder mit einem Standardwert für das neue Argument gerechnet; ist die neue Argumentenanzahl angegeben, so wird nach der neuen Methode gerechnet.

In MAPLIB hat sich die Möglichkeit, die Zahl der Argumente in einer Fortran-Function feststellen zu können, bereits sehr bewährt.

6.2 ARLIST

Die folgenden Beispiele sollen insbesondere die Anwendungsvorschriften von ARLIST erläutern und die damit gewonnenen Möglichkeiten demonstrieren.

1. Aufruf von SUB entsprechend dem Programmabschnitt in Kap.2.2

```
      INTEGER*2 A(5)
      DIMENSION FELD(5)
      EXTERNAL SUB
      ...
   20 CALL ARLIST(FELD,IA,2,A,SUB)
   10 CONTINUE
```

2. Aufruf von $$$$$$ [7,8]

   -Direkter Aufruf X=$$$$$$(EIG,STØFF,A(1),A(2),...,A(IA))

   -Aufruf mit ARLIST für IA≤48

```
DIMENSION F(50)
EXTERNAL $$$$$$
......
X=ARLIST(F,1,2,EIG,1,4,STØFF,IA,4,A,$$$$$$)
```

3. Aufruf von ISTAT [5]

   - Direkter Aufruf J=ISTAT(NS,NR,DYA,QQ0000,IND(1),....,IND(IL))
   - Aufruf mit ARLIST für IL≤100

```
DIMENSION F(104)
EXTERNAL ISTAT
INTEGER ARLIST
....
J=ARLIST(F,1,4,NS,1,4,NR,1,8,DYA,1,8,QQ0000,IL,4,IND,ISTAT)
```

4. Wirkung von $i_j = 0$ bei $n_j > 1$

-Direkter Aufruf CALL ABC(X,X,X,X,X,X,X,X,X,X)
(Es soll 10 mal dasselbe Argument übergeben werden)

-Aufruf mit ARLIST

```
EXTERNAL ABC
DIMENSION(F)10
.......
CALL ARLIST(F,10,0,X,ABC)
```

Hier und in den folgenden Beispielen erspart ARLIST also bei vielen Argumenten Schreibarbeit.

5. Variable Dimensionierung in FORTRAN-Programmen

Eine bewährte Technik zur flexiblen Speicherplatzbelegung in Fortran zeigt folgendes Beispiel. Zur Anpassung des Programms an Problem und verfügbaren Speicherplatz sind lediglich das DIMENSION-Statement für A sowie die Wertzuweisung für K und J zu ändern:

```
      DIMENSION A(30500)
      DIMENSION N(10)
      K=60
      J=100
      N(1)=1
      DO 1 I=2,5
    1 N(I)=N(I-1)+K*J
      DO 2 I=6,10
    2 N(I)=N(I-1)+J
      CALL PGM(K,J,A(N(1)),A(N(2)),A(N(3)),A(N(4)),A(N(5)),
     1 A(N(6)),A(N(7)),A(N(8)),A(N(9)),A(N(10)) )
      STOP
      END
      SUBROUTINE PGM(K,J,A,B,C,D,E,F,G,H,S,T)
      DIMENSION A(K,J),B(K,J),C(K,J),D(K,J),E(K,J),
     1 F(J),G(J),H(J),S(J),T(J)
      ...
```

Der Aufruf von PGM ist mühsam zu programmieren, wenn die Zahl der zu dimensionierenden Felder groß ist. Mit ARLIST würde ein entsprechender Aufruf unter besonderer Ausnutzung des Parameters i_j folgendermaßen aussehen:

```
DIMENSION A(30500)
DIMENSION N(12)
EXTERNAL PGM
K=60
J=100
CALL ARLIST(N,1,4,K,1,4,J,5,K*J,A,5,J,A(5*K*J+1),PGM)
STOP
END
SUBROUTINE PGM ( ... WIE OBEN ...
```

Literaturangaben

[1] Wettstein, H.:
Die Organisation von Programmüberlagerungen als Teilaufgabe des Binders
Elektron.Rechenanl.13,(1971), 170-177

[2] IBM SYSTEM/360
FORTRAN IV Language
Form C28-6515

[3] IBM System/360 Operating System
FORTRAN IV(Gand H) Programmer's Guide
Form C28-6817

[4] Roos, D.(ed.):
ICES System: General Description
MIT, Department of Civil Engineering, Sept.1967

[5] Jordan, J.C.(ed.):
ICES: Programmers Reference Manual
MIT Department of Civil Engineering, Oct.1967

[6] IBM System/360 Operating System
Supervisor and Data Management Services
Form C28-6646

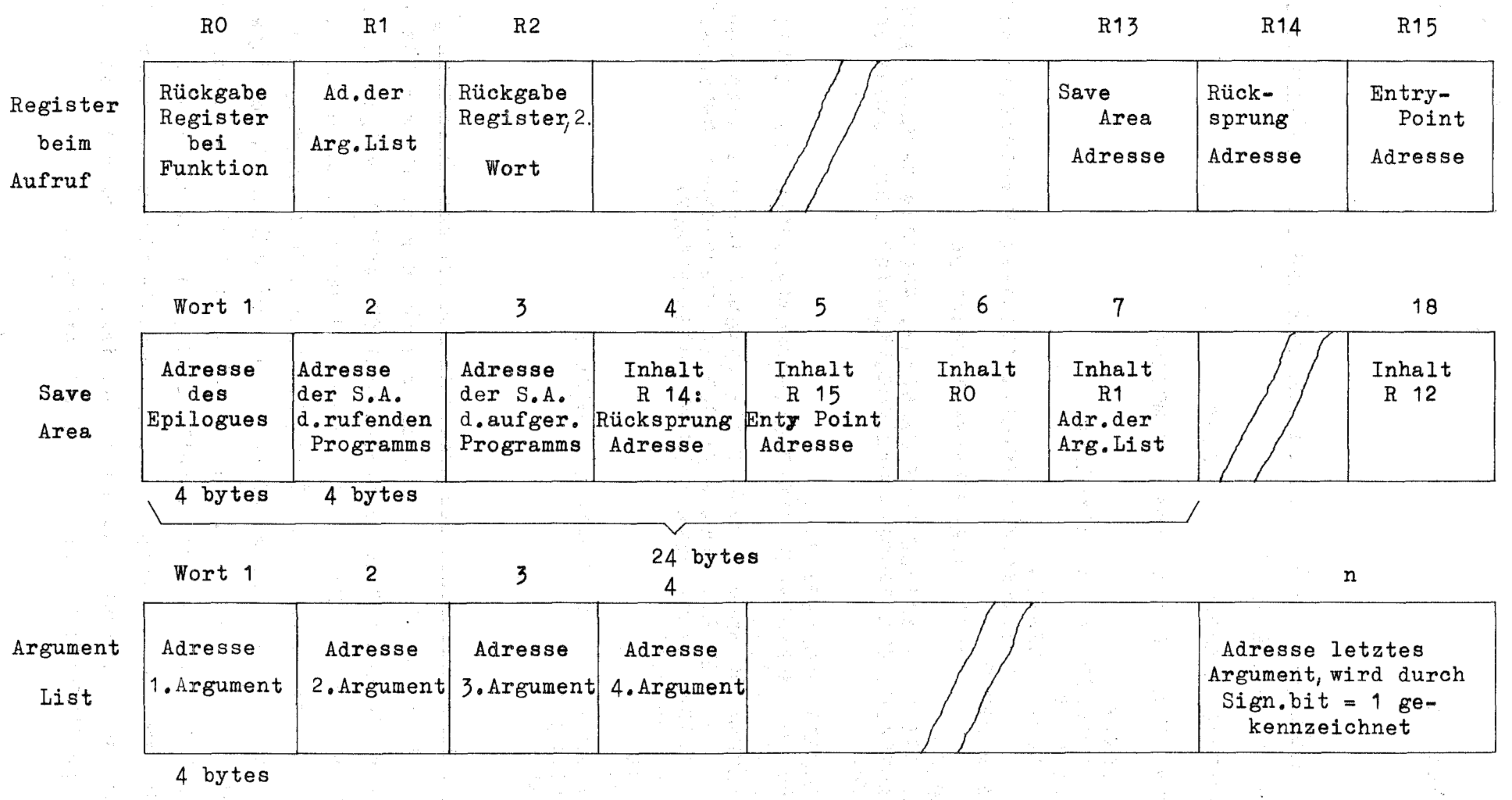
[7] Pee, A., Schumann, U.:
An Integrated Program Library for Material Property Data
Nucl.Eng.Design 14,(1970), 99-103

[8] Schumann, U.:
MAPLIB - Ein Programmsystem zur Bereitstellung von Stoffdaten für Rechenprogramme
Kernforschungszentrum Karlsruhe, Sept.1970, KFK 1253

[9] Zimmerer, W.:
MAPLIB - Funktionen zur Berechnung der Zustandsgrößen von Helium, Luft, Kohlendioxid und Wasser
Kernforschungszentrum Karlsruhe, Mai 1971, KFK 1403

[10] Schumann, U.:
MAPLIB - A Data Bank of FORTRAN-Functions describing Material Properties
(zur Veröffentlichung in "Software Practice and Experience" angenommen).

Schematische Darstellung der Register, der Save Area und der Argumentenliste



Anhang 2

Beispiel für die Anwendung einer Funktion zur Bestimmung der Anzahl der Argumente

```
      FUNCTION ESH2OL(/T     /,/P     /)
C
CN    ***  M A P L I B  ***   FUNCTION
C
CD    10.09.71
CA    H. SPILKER  &  W. ZIMMERER
C$P   ENTROPIE                                                  J/KG.K
C$M   WASSER, FLUESSIG
CP    T        273.<=AMAX1(VTH2O(P),273.16)<=TK<=1073.15<=1074.
CP    TEMPERATUR                                                K
CP    P       1.E-30<=1.E-30<=PN<=1.E8<=1.E8
CP    DRUCK                                                     N/M2
CL    H. SPILKER, IRE-BERICHT NR.62/68, PROGRAMMBESCHREIBUNG NR.151
CS    $NUMBR,NUMBR$,RANGE,WUDZ$,FUNCT$,$ITER,$SUB1,$SUB2,$SUB3,$SUB4,
CS    $WARN
C$N   1
      INTEGER JSOLL(2)/1,2/
      CALL $NUMBR(ESH2OL,'ESH2OL',NUMBR$(2),JSOLL,2,I,&111)
      IF (I-1) 1,1,2
CB    SUBROUTINE $NUMBR PRUEFT DIE ARGUMENTENZAHL; WIRD NUR ARGUMENT
CB       T GELIEFERT, DANN WIRD DER SATTDAMPFDRUCK BERECHNET UND
CB       DIESER DRUCK-WERT NACH PS UMGESPEICHERT
    1 PS= VPH2O(T)
      GOTO 3
    2 PS= P
    3 ESH2OL= WUDZ$(T,PS,3,1,'ESH2OL')
  111 RETURN
      END
```

Die hier verwendete Subroutine $NUMBR ist ausführlich in [8] beschrieben. Ihre Hauptaufgabe besteht darin, die mit NUMBR$ (Funktionsaufruf im CALL-Statement für $NUMBR) festgestellte Anzahl der Argumente mit den im Feld JSOLL enthaltenen Sollwerten zu vergleichen. I enthält hier beim Rücksprung den Wert 1 oder 2, je nachdem JSOLL(1) oder JSOLL(2) gleich dem Istwert ist. Stimmt der Istwert mit keinem Sollwert überein, wird eine Fehlernachricht erzeugt und eventl. zu der im letzten Argument angegebenen Statementnummer gesprungen. Man beachte hier insbesondere die Tatsache, daß das Argument P nur dann referiert wird, wenn I=2 ist. Die Funktion VPH2Ø(T) berechnet den Sattdampfdruck.

Anhang 3

Listen der Assemblerroutine NUMBRA und ARLIST

```
 1 NUMBRA   START 0
 2          BC    15,12(15)
 3          DC    X'7'
 4          DC    CL7'NUMBRA'
 5          STM   14,12,12(13)
 6          BALR  10,0          SET REGISTER 10 BASIS REGISTER
 7          USING *,10
 8 *                ***************************************************
 9 *                *                                                 *
10 *                *    AUTOR : PEE , A.                             *
11 *                *                                                 *
12 *                *    DATUM : 01.10.71                             *
13 *                *                                                 *
14 *                *    TEST PROCESSOR: IBM 360/65 AND IBM 360/85    *
15 R0       EQU   0    ***************************************************
16 R4       EQU   4    *   CALLING SEQUENCE :                          *
17 R5       EQU   5    ***********************                         *
18 R6       EQU   6    *                                               *
19 R7       EQU   7    *        N = NUMBRA(DUMMY)                      *
20 R11      EQU   11   *-----------------------------------------------*
21 R12      EQU   12   *  WHERE :                                       *
22 R13      EQU   13   *    N          : INTEGER*4= NUMBER OF ARGUMENTS,  *
23 R14      EQU   14   *    DUMMY      : DUMMY-ARGUMENT FOR COMPILER      *
24 R15      EQU   15   ***************************************************
25         SR    R11,R11       R11=0
26         L     R5,=F'-10'
27         L     R6,4(0,13)    LOAD R6: S.A. ADR. OF CALLY
28         L     R4,12(0,R6)   LOAD R4: RETURN ADR. IN CALLY
29         L     R14,4(R6,0)   LOAD R14: S.A. ADR. OF CALLER
30         L     R15,16(R14,0) LOAD R15: E.P. ADR. OF CALLY
31         CLC   SETUP(4),9(R15)   TEST FOR CALLY = ICES.MOD.ENTRY
32         BC    7,NOLINK      IF NO THEN GOTO NOLINK
33         L     R4,12(R14,0)  LOAD R4: RETURN ADR. IN CALLER
34         L     R14,4(R14,0)  LOAD R14: ADR.  S.A. OF CALLER
35         L     R15,16(R14,0) LOAD R15: E.P. ADR. OF CALLER
36         CLI   0(R4),X'47'   TEST FOR BRANCH
37         BC    8,NOLINK      IF YES THEN GOTO NOLINK
38         L     R4,8(R4,0)    LOAD R4: RETURN ADR. IN CALLER
39 NOLINK  LH    R12,0(R5,R4)  R12= 2. INST FOR BALR TO SUBPR.
40         SRA   R12,4(0)      SHIFT RIGHT 1 HALF BYTE
41         L     R5,=F'1041'   R5='411'=3 FIRST BYTE OF LA 1,....
42         CR    R12,R5        COMPARE R5,R12
43         BC    8,TEST        ARGUMENTS: BRANCH TO TEST FOR NUMBER
44         BC    15,FUNC       GO TO FUNC
45 TEST    L     R7,24(R6,0)   LOAD R7: ADR. OF ARGLIST IN CALLER
46 LOOP    LH    R5,0(R7,R11)  LOAD R5: 1.HALFWORD OF ARGUMENT ADR.
47         LA    R11,4(R11,0)  LOAD R11: R11+4
48         LTR   R5,R5         TEST SIGN OF ARG. ADR.
49         BC    2,LOOP        >0 LOOP
50         SRA   R11,2(0)      R11=R11/4
51 FUNC    LR    R0,R11        NUMBER OF ARGS IN FUNCTION REGISTER
52         L     R14,12(R13,0) LOAD R14: RETURN ADR.
53         LM    1,12,24(13)   RESTORE REGISTERS
54         MVI   12(13),X'FF'   SET RETURN CODE
55         BCR   15,14         RETURN
56 SETUP   DC    X'D76B40E5'
57         END
```

```
*
*DATUM    01.10.71
*
*AUTOR    SCHUMANN
*
*BESCHREIBUNG
*         ARLIST DIENT ZUM ERZEUGEN VON ARGUMENTENLISTEM MIT
*         VARIABLER ARGUMENTENANZAHL UND VARIABLEN ARGUMENTEN-
*         ADRESSEN.
*
ARLIST     START
*       AUFRUF
*       ALS FUNCTION ODER ALS SUBROUTINE
*       ARLIST(FELD,N1,I1,A1,N2,I2,A2,....,NM,IM,AM,EXTERNAL)
*       FELD  = SPEICHERBEREICH FUER NEUE ARGUMENTENLISTE
*           ERFORDERLICHE LAENGE:
*         SOVIELE WOERTER, WIE DIE EXTERNAL-ROUTINE ARGUMENTE
*         ERWARTET.
*
*       J=1,M,1
*       NJ    = ANZAHL DER WERTE IM FELD AJ
*       IJ  = INCREMENT DER ADRESSEN IM FELD AJ
*       AJ    = FELD MIT DEN ARGUMENTEN
*
*       EXTERNAL    = MIT NEUER ARGUMENTENLISTE AUFZURUFENDE ROUTINE
*
*
*      ICETRAN- PROGRAMME ERWARTEN, DASS REGISTER VIER DIE COMMON
*      -ADRESSE ENTHAELT, DAHER WIRD REGISTER VIER NICHT VERAENDERT
*
*
*      NAMENS-ABSPEICHERUNG UND SAVE-AREA-VERKETTUNG
           BC     15,12(15)
           DC     X'7'
           DC     CL7'ARLIST'
           STM    14,12,12(13)
           USING ARLIST,15
HSA        EQU    5
           LR   HSA,13
           LA     13,SAVE
           ST     13,8(HSA)
           ST     HSA,4(13)
NULL       EQU    0          WIRD NICHT VERAENDERT
ARGALT     EQU    1          ADR. DER ALTEN ARGUMENTENLISTE
J          EQU    2
K          EQU    3
COMMON     EQU    4     WIRD NICHT VERAENDERT
I          EQU    5
INCR1      EQU  6
N          EQU 7
INCR12     EQU 8
NL         EQU 9
ACHT       EQU 14
ARGNEU     EQU    10         ADR. DER NEUEN ARGUMENTENLISTE
LAENGE     EQU    12
VIER       EQU    14         ENTHAELT 4
*
*      FESTSTELLUNG DER ANGELIEFERTEN ARGUMENTENZAHL
*
```

```
          SR     9,9          REG9=0
          LA     3,*+4        REG3=LOOPADR.
          LH     5,0(1,9)     LADE 1. HALBWORT D. ARGADR. IN REG.5
          LA     9,4(9)       REG9 =REG9+4
          LTR    5,5          PRUEFE  REG. 5
          BCR    2,3          FALLS  >0  LOOP
*         REG9 =  4*ARGUMENTENANZAHL
          LA ACHT,8
          SR 9,ACHT
          LA INCR12,12
          LTR 11,9
          BL RETURN ANZAHL DER ARGUMENTE=1  --> FEHLER
          SR 10,10
          DR 10,INCR12
          MR 10,INCR12
          CR 9,11
          BNE RETURN DIE ANZAHL DER ARGUMENTE N BEIM AUFRUF VON ARLIST
*                    ENTSPRICHT NICHT DER FORMEL N=3*M+2 --> FEHLER
          SR NL,ACHT
*
*       AUFBAU DER NEUEN ARGUMENTENLISTE
*
          L ARGNEU,0(1)
          LCR J,ACHT
          LA VIER,4
          LCR K,VIER
          LA INCR1,1
*
*     LOOP UEBER DIE ARGUMENTE VON ARLIST
LOOPJ      BXH    J,INCR12,LOOPEND
           L      N,0(ARGALT,J)
*                              N=NJ
           L      N,0(N)
          LTR  N,N TEST VON NJ
          BNH LOOPJ  FALLS NJ=0 ODER NEGATIV  IST, WIRD AUS DER J-TEN
*                    ARGUMENTENGRUPPE KEIN ARGUMENT ERZEUGT
           L      LAENGE,4(ARGALT,J)
*                              LAENGE=IJ
           L      LAENGE,0(LAENGE)
*                              R11=ADR(AJ)
           L      11,8(ARGALT,J)
           AR     K,VIER
           ST     11,0(ARGNEU,K)
*
*      LOOP UEBER NJ
          LR    I,INCR1
LOOPI      BXH    I,INCR1,LOOPJ
           AR     11,LAENGE
           AR     K,VIER
           ST     11,0(ARGNEU,K)
           B      LOOPI
*
LOOPEND  EQU *
*  ABFRAGE , OB DIE SUBROUTINE MIT NULL ARGUMENTEN AUFGERUFEN WERDEN
*           SOLL
          LTR    K,K
          BL     NULLARG  AUFRUF OHNE ARGUMENTE
          AR     K,ARGNEU  ADRESSE DES LETZTEN ARGUMENTS
*          KENNZEICHNUNG DES LETZTEN ARGUMENTS
           OI     0(K),B'10000000'
```

```
*
*       AUFRUF DER EXTERNAL ROUTINE
*
            L      15,0(ARGALT,J)
*
            LA    ARGALT,0(ARGNEU)
*       DIESER BEFEHL ENTSPRICHT DEM FORTRAN E/G/H BEFEHL, RELEASE 14/19,
*       DAMIT NUMBRA ANGEWENDET WERDEN KANN
*
            L   15,0(15)
            BALR  14,15
            NOP 2
*
*       RUECKSPEICHERN DER REGISTER 13, 14 UND 1 BIS 12
*
RETURN    EQU *
            L      13,4(13)
            L      14,12(13)
          RETURN  (1,12),T,RC=0
NULLARG   EQU   *
*    CALL DER EXTERNAL ROUTINE OHNE ARGUMENTENLISTE
          LA    3,RETURN
          L 15,0(ARGALT,J) LADE ADR. DER ADR. DES ENTRY-POINTS
          L      15,0(15)       LADE ENTRY-POINT-ADRESSE
          BALR  14,15           UND SPRINGE ZUM UNTERPROGRAMM
           NOP 4
         BR    3    SPRUNG NACH RETURN
*
*       DAS ALLGEMEINE REGISTER NULL UND DIE FLOATING-POINT
*       REGISTER WURDEN NICHT VERAENDERT, DAHER KANN
*       ARLIST SOWOHL ALS FUNCTION ALS AUCH ALS SUBROUTINE
*       AUFGERUFEN WERDEN.
*
SAVE        DC     18F'0'
            END
```