RUHR - UNIVERSITÄT BOCHUM

Arbeitsberichte

des

Rechenzentrums

Direktor: Prof. Dr. H. Ehlich

Nr. 7206 BO.E2.06

BOGOL-TAS, ein Weg zur systemnahen Programmierung in ALGOL am TR 440

von
Manfred Rosendahl

Bochum, August 1972
MV1
Bochum, Buscheystraße, Gebäude NA

0. Überblick über das System BOGOL-TAS

0.0. Einleitung

Die Algol 60 Implementierung des TR 440 [5] ist eine sehr weitgehende und vollständige Implementierung der Sprache Algol 60 [1] . Für gewisse Zwecke, insbesondere beim Erstellen von Programmsystemen reichen die Möglichkeiten der Sprache Algol 60 ohne weiteres noch nicht aus. Auf der anderen Seite möchte man wegen einiger sehr komfortabler Eigenschaften wie Rekursivität, hohem Dokumentationswert, eleganter Formulierung, leichtem Testen jedoch nicht auf eine andere Sprache, insbesondere nicht auf die Assemblersprache TAS [6] , ausweichen. Um nun die durch die Sprache Algol gegebenen Einschränkungen zu umgehen, ist das System BOGOL (BOchumer-alGOL) gedacht. Bei diesem handelt es sich um ein Unterprogrammpaket. Vorerst sind 3 Teile geplant:

(1) BOGOL-TAS, zur systemnahen Programmierung in Algol.

(2) BOGOL-STRING, zum eleganten Stringhandling in Algol. Dieses System erlaubt in etwa die Möglichkeiten der string-orientierten Sprache SNOBOL [2] in Algol.

(3) BOGOL-LISP. Hier wird den Algol-Programmen eine LISP-Datenstruktur beigegeben. Dies erlaubt es, LISP [3] und AIDA [4] - Funktionen in Algol zu benutzen.

0.1. Zweck des Systems BOGOL-TAS

Für bestimmte Programmierungsaufgaben werden TAS-Programme, d.h. Programme in der maschinennahen Assemblersprache, benötigt. Die Gründe dafür können sein:

(1) Man möchte Befehle ausnutzen, die in den höheren Programmiersprachen nicht ansprechbar sind (z.B. Zeichenverarbeitung (BNZ, CNZ), Systemleistungen mit SSR, Tabellensuchen, Halbwortbefehle, Bitmuster).

(2) Man möchte mit Adressen rechnen.

(3) Dem Programm steht die Syntax der höheren Programmiersprachen entgegen. So ist es z.B. in Algol und Fortran nicht möglich, Programme mit variabler Parameterzahl zu schreiben oder in Abhängigkeit vom Parametertyp das Programm zu verzweigen.

(4) Die Programme sollen schneller laufen.

Für die Fälle (2) und teilweise auch (3) kann man die Programmiersprache BCPL [7] benutzen. Der Anschluß von TAS-Prozeduren etwa um (1) zu erhalten,

ist dort jedoch umständlicher als in Algol oder Fortran.

In Algol und Fortran kann man TAS-Programmteile nur auf Prozedurebene anschließen. Ein solches TAS Unterprogramm setzt sich daher meist zusammen aus den eigentlichen Befehlen, die in Algol nicht ansprechbar sind und einem organisatorischen Rahmen zur Parameterübernahme und Aufbereitung der Daten. Dabei muß jedoch wiederum mit Adressen oder Festkommazahlen gerechnet werden, womit auch dieser Teil nicht in Algol zu schreiben ist.

In BOGOL-TAS wird ein Unterprogrammsystem zur Verfügung gestellt, daß die einzelnen TAS-Leistungen durch Algol-Prozeduren erbringt, wobei die Schnittstellen gewöhnliche Algol Prozedurparameter bzw. Ergebnisse sind. Desgleichen gibt es Unterprogramme zur Aufbereitung der Daten, wie Festkomma- und Halbwortmanipulationen. Um die Einschränkungen, die durch die Syntax der Sprache Algol 60 gegeben sind (feste Parameterzahl, bestimmter Parametertyp) zu umgehen, gibt es ebenfalls Prozeduren zur Aktivierung und Bestimmung der Parameter, so daß diese als formale Parameter nicht mehr aufgeführt werden müssen. Dadurch ist es ohne Änderung des Compilers möglich, gewisse Einschränkungen der Sprache Algol 60 zu umgehen.

0.2. Übersicht über die BOGOL-TAS Prozeduren.

Die Prozeduren lassen sich in 5 Gruppen einteilen:

(1) Prozeduren zur Erweiterung der Algol-60-Sprachmöglichkeiten:

1.1.	PART	Parametertyp
1.2.	PARV	Parameterwert
1.3.	PARZ	Parameteranzahl
1.4.	PARA	Parameterwert (Informationsvektor v. Arrays)
1.5.	HVZV	Haupt- und Zusatzversorgung
1.6.	CALL	formaler Prozeduraufruf
1.7.	FOHVZV	Erzeugen einer formalen Haupt- und Zusatzversorgung
1.8.	LABEL	Definieren einer globalen Marke
1.9.	GOTO	Sprung auf eine globale Marke
1.10.	RETURN	Rücksprung aus Unterprogramm

(2) Prozeduren zur Adressenrechnung:

2.1.	REF	Adresse einer Variablen
2.2.	VAL	Wert einer Adresse (Ganzwort)
2.3.	AS	Zuweisung auf eine Adresse
2.4.	VALI	Wert einer indizierten Adresse
2.5.	REFI	indizierte Adresse

(3) Prozeduren für Festkommazahlen und Nichtzahlwörter:

3.1.	FK	Wandlung Gleitkomma nach Festkomma
3.2.	GK	Wandlung Festkomma nach Gleitkomma
3.3.	AD	Festkommaaddition
3.4.	SB	Festkommasubtraktion
3.5.	ML	Festkommamultiplikation
3.6.	DV	Festkommadivision
3.7.	AUT	Exclusives Oder (bitweise)
3.8.	VEL	Oder (bitweise)
3.9.	ET	Und (bitweise)
3.10.	IE	Hexadezimal- oder Oktadenkonstante

(4) Prozeduren für spezielle Maschinenbefehle:

4.1.	BNZ	Bringe nächste Zeichen
4.2.	CNZ	Speichere nächstes Zeichen
4.3.	TOK	Transportiere Oktaden
4.4.	VOK	Vergleiche Oktaden
4.5.	TK	Typenkennung
4.6.	ZTO	Setze TK = 0
4.7.	ZT1	Setze TK = 1
4.8.	ZT2	Setze TK = 2
4.9.	ZT3	Setze TK = 3
4.10.	ZT	Ganzwort mit neuer Typenkennung

4.11.	SSR	Springe ins System (Systemleistungen)
4.12.	SH	Shiften
4.13.	TLI	Tabellensuchen
4.14.	B2V	Wert eines Halbwortes
4.15.	C2	Zuweisung auf ein Halbwort
4.16.	T	Ausführung eines beliebigen TAS-Befehls
4.17.	TAS	Ablegen einer TAS-Befehlsfolge
4.18.	SU	Unterprogrammsprung auf TAS-Befehlsfolge

(5) Prozeduren für Systemdienste:

5.1.	ALSETZ	Setzen einer Alarmadresse
5.2.	ALTEST	Testen der Alarmursache

0.3. Testmöglichkeiten

Beim Trace erscheint bei Zuweisungen jeweils eine Typenkennungsabhänger-Ausdruck. Es bedeutet:

TK = 0 Gleitkommazahl
TK = 1 boolscher Wert, true oder false
TK = 2 undefiniert
TK = 3 Alphazeichen

Nach Anwendung der BOGOL-Prozeduren haben die Worte mit TK ≠ 0 jedoch auch andere Bedeutungen und ihr Inhalt ist von Interesse. Nach Anmeldung der BOGOL-Bibliothek steht daher ein geändertes Montageobjekt A&TRAC zur Verfügung, das bei Zuweisungen für TK≠0 einen anderen Ausdruck liefert:

TK = 1 : 1*Hexadezimaler Wert / Zahlwert
TK = 2 : 2*Hexadezimaler Wert /
TK = 3 : 3*Hexadezimaler Wert / Oktaden

Beim Ausdruck der Werte werden dabei linksbündige Nullen unterdrückt.

0.4. Deklarationen

Die BOGOL-Prozeduren werden wie gewöhnliche Algol code-Prozeduren in der Form:

‹type› procedure NAME (X) ; code;

deklariert. Wegen der Typenkennung ist jedoch folgendes zu beachten:

Wird einer integer-Größe eine real-Größe zugewiesen, so erfolgt eine Rundung. Diese läuft jedoch für TK ≠ 0 auf einen Alarm. Daher ist zu beachten:

(1) Funktionsprozeduren, deren Wert eine Größe mit TK ≠ 0 sein kann, werden als integer-procedures deklariert.
(2) Variablen, die Werte mit TK ≠ 0 erhalten können, werden als real deklariert.

Dann kann eine Zuweisung von real nach integer nicht auftreten.

Die in den folgenden Kapiteln lediglich zur Beschreibung benutzten Deklarationen haben folgende Bedeutung:

var	Nur einfache oder indizierte Variable, kein Ausdruck erlaubt.
address	Aktueller Wert wird als Adresse benutzt.
fix	Nur Wert mit TK = 1 erlaubt.
type	Wert mit beliebiger TK möglich.
type X	Wert mit TK X möglich.
form	Wert mit TK $\geq$ 1 möglich.
integer	Aktueller Wert muß ganzzahlige Gleitkommazahl sein. Deklaration jedoch beliebig.
real	Aktueller Wert kann beliebige Gleitkommazahl sein.

0.5. Handhabung

Die Prozeduren des BOGOL-Systems liegen als Montageobjekte in der Bibliothek BOGOL auf der LFD (langfristige Datenhaltung) vor.

Nach dem Kommando ¤BIBANM., BOGOL sind sie wie gewöhnliche Algol-Prozeduren der &OEFDB (öffentliche Datenbasis) zu benutzen. In dieser Bibliothek befindet sich auch das geänderte Montageobjekt A&TRAC, das den geänderten Trace-Ausdruck bewirkt.

Um das lästige Hinschreiben aller code-Prozedurdeklarationen zu sparen, ist ein Precompiler geplant, der in einem Algol-Programm alle fehlenden Deklarationen standardmäßig einsetzt.

Mit Ausnahme der Prozedur PARV darf keine Prozedur rekursiv benutzt werden. Teilweise (PARV, BNZ, CNZ, TOK, VOK) verändern sie die Indexzellen 0-7.

1. Zur Erweiterung der Algol 60 Sprachmöglichkeiten

1.1. integer procedure PART (N);

value N; integer N;

PART (N) liefert den PARameter-Typ des N-ten aktuellen Parameters des letzten Aufrufs der Prozedur, in der sich der PART-Aufruf befindet.

Abgeliefert wird eine Zahl T mit $1 \leq T \leq 8$, mit folgender Verschlüsselung:

T	Parametertyp des aktuellen Parameters
1	einfache oder indizierte Variable
2	Konstante oder Ausdruck
3	Name eines Feldes
4	eine Marke, aber kein integer-label
5	integer-label
6	switch-Name
7	Name einer eigentlichen Prozedur oder einer Funktionsprozedur
8	ein string

1.2. procedure PARV (N,A,B);
value N; var A, B; integer N;

PARV (N,A,B) liefert "PARameter-Value" des N-ten Parameters. Die den Parameter beschreibenden Werte werden auf den Variablen A und B abgelegt.
Nach dem Aufruf ist A und B in Abhängigkeit vom Parametertyp wie folgt belegt.

T	A	B
1	Wert der Variablen	Adresse der Variablen
2	Wert des Ausdrucks	undefiniert
3	Wert des 1. Feld - elements	Adresse des 1. Feld - elements
4	Der Aufruf von PARV auf die Marke	bewirkt einen Sprung
5	Wert der Konstanten in Gleitkomma	Adresse der Marke
6	Aufruf nicht erlaubt	
7	Falls Funktionsprozedur ohne Parameter dann Funktionswert sonst undefiniert	Falls nicht Funktionsprozedur ohne Parameter, dann Startadresse der Prozedur, sonst undefiniert
8	Kopfwort des strings d.i. Länge in Fest - komma	Adresse des Kopfwortes

1.3. integer-procedure PARZ;

PARZ liefert die aktuelle PARameterZahl des letzten Aufrufs der Prozedur,in der PARZ aufgerufen wird.

Anm.: PARZ entspricht in seiner Leistung genau der TR 440 Prozedur PARZAHL (N31.FO.11), die jedoch z.Z. mit den Prozeduren des BOGOL-Systems nicht kompatibel ist.

1.4. PARA (N,A,B)

value N; integer N; var A,B;

Wie PARV, jedoch wird für den N-ten Parameter, falls er ein Array ist, nicht die Adresse und Wert des 1. Elements geliefert, sondern in B wird die Adresse des Informationsvektors abgeliefert.

Beispiel 1.1.

Es soll eine Funktion MAX (siehe N31.FO.11, [8]), die den kleinsten Wert aus den Type-Parametern oder Arrays der Parameterliste liefert, in Algol geschrieben werden.

Diese hat dann (ohne Deklarationen) die Gestalt:

```
real procedure MAX;
begin
F2 := FK(2);
```

```
N := PARZ; M := 10E-150;
for I := 1 step 1 until N do
begin PARA (I,A,B);
      T := PART (N);
      if T ≤ 2 then
      begin if M < A then M := A end
      else if T = 3 then
      begin AA := B2V (B,  -2);
            EA := B2V (B,  -1);
WW:         A  := VAL (AA);
            if M < A then M := A;
            AA = AD (AA, F2);
            if AA ≤ EA then goto WW;
      end else
      begin PRINT ('(' MAX, PARAMETER FALSCH')');
            A = 1/0;
      end;
end;
MAX:= M;
end;
```

Die Parameter werden der Reihe nach auf ihrem Typ getestet. Falls ein Array vorliegt, werden Anfangsadresse (AA) und Endadresse (EA) aus dem Versorgungsvektor bestimmt und in einer Schleife die Werte dazwischen getestet.

Als Beispiel zur Anwendung dieser Prozeduren diene ferner:

Beispiel 1.1

real procedure SUM (A_1, A_2, ..., A_N) soll die Summe von N-Variablen liefern. Der Aufruf soll mit variabler Parameterzahl erfolgen können, die Parameter sollen auf ihren Typ geprüft werden. Die Prozedur sieht dann wie folgt aus:

```
real procedure SUM;
begin
integer procedure PART(X); code;
procedure PARV(X); code;
integer procedure PARZ; code;
real S,A,B;
integer I, OG;
OG := PARZ; S := 0;
for I := 1 step 1 until OG do
begin if PART (I) gt 2 then goto ERROR;
PARV (I,A,B);
S := S+A;
end;
goto ENDE:
ERROR: PRINT ('('FALSCHER TYP, PARAMETER')');
       TYPE (I);
       SUM := 1/0;
ENDE : SUM := S;
end;
```

Der Reihe nach werden die Parameter aktiviert und nur bei einer Variablen oder einem Ausdruck (Typ ≤ 2) wird weitergemacht.

Möchte man nun eine Prozedur DIF (B_1,B_2,B_n,A) schreiben, bei der der Wert ($\sum_{v=1}^{n} B_v$) - A gebildet werden soll und man möchte zur Berechnung der Summe das Programm SUM benutzen, so benötigt man einen Prozeduraufruf mit unbestimmter Parameterzahl. Dies ist ebenfalls normalerweise in Algol 60 nicht möglich. Im BOGOL-System dienen dazu die Prozeduren HVZV und CALL.

Das Aktivieren eines Algol-Parameters erfolgt durch die Ausführung eines Befehls aus einem sogenannten Versorgungsblock, und zwar wird der k-te Parameter durch den K+1. Befehl des Versorgungsblocks aktiviert. Um nun SUM in DIFF aufzurufen, muß zunächst ein Versorgungsblock aufgebaut werden. Dazu dient:

1.5. form procedure HVZV (N);
value N; integer N;
Der Aufruf von HVZV liefert die Haupt- und Zusatzversorgung des N-ten aktuellen Parameters des letzten Aufrufes der Prozedur, in der HVZV aufgerufen wird.

Das 1. Wort des Versorgungsblocks hat folgende Gestalt:

	24	2	8
TK2	FA	SS	PZ

FA : Fehleradresse

SS : Sprachschlüssel

PZ : Parameterzahl

(Siehe auch TAS-Handbuch [6]).

Durch die noch zu behandelnden Operationen zur Festkomma- und Teilwort-Bearbeitung läßt sich dieses 1. Wort erzeugen.

Der formale Prozeduraufruf erfolgt dann mittels

1.6. procedure CALL (P, VBA);
procedure P; var VBA;
P ist die aufzurufende Prozedur.
VBA ist das 1. Element des Versorgungsblocks.
Ist P eine Type-procedure, so erfolgt die Deklaration in der Form:

real procedure CALL (X); code;
bzw.:
integer procedure CALL (X); code;

Das Ergebnis ist das des Aufrufs von P mit dem Versorgungsblock.

Die Prozedur DIF ist dann

Beispiel 1.3

```
real procedure DIF;
begin
real procedure SUM(X); code;
real procedure CALL(X); code;
integer procedure HVZV(X); code;
integer procedure ADD(X); code;
integer procedure FK(X); code;
integer N,I;
real S;
array VB [0:20] ;
N := PARZ;
for I := 1 step 1 until N-1 do
VB [ I] := HVZV(I);
VB [ 0] := ADD (VB [0] , FK(-1));
S := CALL (SUM, VB [0] );
PARV (N,A,B);
S := S-A;
DIFF := S;
end;
```

Die Parameterzahl des Aufrufs von SUM ist um eins niedriger als des von DIF.

Es kann jedoch auch nötig sein, als aktuellen Parameter eines mit CALL erzeugten formalen Prozeduraufrufes eine Größe zu haben, die lokal definiert und nicht wie die obigen Parameter durchgereicht wird. Dann benutzt man:

1.7 form procedure FOHVZV(X):

X kann ein beliebiger Parameter sein.
Die Prozedur liefert die Haupt- und Zusatzversorgung für die Größe X als aktueller Parameter.

Beispiel 1.4

Mit CALL soll ein formaler Aufruf von DIFF erzeugt werden. Der n-te Parameter sei die Zahl 1000. Dies wird erreicht durch

```
VB [ N ] := FOHVZV (1000);
```

Möchte man aus einer getrennt übersetzten Prozedur auf einen label im Hauptprogramm springen, so geht dies nur, wenn der label als Parameter übergeben wird. Für Variablen ist im ALGOL-TR 440 das common-Konzept verwirklicht, nicht jedoch für Marken. Um dies auch für Marken zu bekommen, dienen die Prozeduren LABEL, GOTO.

1.8 procedure LABEL (L,A);

label L; var A;

L ist eine Marke im Hauptprogramm und A eine Variable eines in Haupt- und Unterprogramm erklärten common-Speichers.
Die Information über label L wird auf Variable A abgelegt.

1.9 procedure GOTO (A);
var A;
Falls im Hauptprogramm der Aufruf LABEL (L,A) erfolgte, bewirkt GOTO(A) einen Sprung nach L;

Im Gegensatz zu FORTRAN ist in ALGOL eine Rückkehr aus einer Prozedur jeweils nur am statischen Programmende möglich. Bei Typeprozeduren muß vorher jeweils noch eine Zuweisung erfolgen.
Dies umgeht:

1.10 procedure RETURN;
Es erfolgt sofort Rücksprung in das aufrufende Programm.
Befindet man sich in einer Type-procedure, so erfolgt dies durch

1.9a real (bzw. integer) procedure RETURN (A);
real (bzw. integer) A;
Es erfolgt Rücksprung mit A als Funktionswert.

2. Adressenrechnung

Beim TR 440 - Algol werden alle Zahlen, auch die integer-deklarierten, als Gleitkommazahlen (TK=0) dargestellt. Alle arithmetischen Operationen erfolgen mit Gleitkommabefehlen.

Boolsche Werte sind Festkommazahlen (TK = 1). Ist das rechte Bit besetzt, ist der Wert true, sonst false. Ist die TK=2, so zeigt dies an, daß der Wert undefiniert ist. Die Werte mit TK = 3 bilden die strings.

Die Adressen von Variablen, Parametern u.ä. fallen als Festkommawerte an. Um ein doppeltes Umwandeln Festkomma, Gleitkomma, zu vermeiden, wird auch die Adressenrechnung im BOGOL-System in Festkomma durchgeführt. Die arithmetischen Prozeduren dazu siehe Kapitel 3.

Zur Erzielung einer Adresse dient:

2.1 address procedure REF (X) ;

Der Parameter X kann sein

(1) einfache Variable

(2) Prozedurname, jedoch nicht Funktionsprozedur ohne Parameter

(3) integer label

(4) string

(5) Feldname

Das Ergebnis ist eine Festkommazahl, deren Wert die Adresse der Variablen bzw. des integer-labels oder des Kopfwortes des strings ist.
Bei einer Prozedur ist der Wert die Startadresse der Prozedur.

Eine besondere Bedeutung hat REF, wenn X der Name eines Feldes ist. Der Funktionswert ist dann die Adresse des Informationsvektors. Darüber lassen sich dann alle weiteren Angaben wie Dimension, Länge usw. beschaffen. (Siehe dazu aber auch die Algol-Prozeduren DIMA, GRAD, GRAD 1 (N31.FO:11) [8]). Für den Informationsvektor eines Array-Parameters siehe PARA. Um wieder von einer Adresse auf den Wert zuzugreifen dient:

2.2 type procedure VAL (A);
address A;
A muß eine Adresse sein, die man z.B. mit PARV oder REF erhalten hat. VAL liefert den Inhalt des Ganzwortes mit dieser Adresse.

Um auf eine Adresse eine Zuweisung vorzunehmen dient:

2.3 procedure AS (A,X);
value X; address A; type X;
Der Variablen mit der Adresse A wird der Wert X zugewiesen.

Zum Zugriff auf Feldadressen dient

2.4 type procedure VALI (A,I);
value I,A; address A; integer I;
A wird als Adresse des Feldelements F [0] angenommen. VALI (A,I) liefert dann den Wert des Feldelements F [I] ..

Möchte man dagegen die Adresse eines Feldelements:

2.5 address procedure REFI (A,I);
value A,I; address A; integer I;
Unter den gleichen Bedingungen wie bei 2.4 liefert REFI (A,I) die Adresse des Feldelements F [I].

Zur Adressrechnung beim TR 440 sind noch folgende Anmerkungen zu machen:

Ein TR 440 - Ganzwort hat die Form:

TK	Adresse 2n	Adresse 2n+1
2	24	24

Es belegt also 2 Adressen; deshalb beträgt die Adressendifferenz zwischen zwei aufeinanderfolgenden Ganzwörtern 2.

Alle einfachen Algol-Größen real, integer, boolean belegen genau 1 Ganzwort.

3. Erzeugung und Verarbeitung von Festkommazahlen und Nichtzahlwörtern

Die arithmetischen Operationen beziehen sich beim TR 440 - Algol immer auf Gleitkommazahlen. Bei einer Anwendung auf Zahlen mit TK ≠ 0 liefern sie einen Typenkennungsalarm. Im BOGOL-System existieren daher Prozeduren, die die Verarbeitung solcher Wörter erlauben.

Zunächst 2 Prozeduren zur Konvertierung:

3.1. fix procedure FK(X);
value X; real X;
Der Wert der Funktionsprozedur FK ist eine Festkommazahl mit dem Wert X. Der Wert von X muß ganzzahlig sein. Zur Beschleunigung der Rechnung wird diese Annahme gemacht.

3.2. real procedure GK(A);
value A; fix A;
Der Funktionswert ist eine Gleitkommazahl mit dem Wert der Festkommagröße A.

Als arithmetische und boolsche Operatoren existieren:

3.3. form procedure AD(A,B);

value A,B; form A,B;

Funktionswert = A+B (Festkommaaddition)

3.4. form procedure SB(A,B);

value A,B; form A,B;

Funktionswert = A-B (Festkommadivision)

3.5. fix procedure ML (A,B)

value A,B; fix A,B;

Funktionswert = A*B (Festkommamultiplikation)

3.6. fix procedure DV(A,B)

value A,B; fix A,B;

Funktionswert = A/B (Festkommadivision)

Bei den Operationen AD und SB dürfen A und B beliebige Typenkennung ≠ 0 haben. Bei TK=2 oder 3 werden die Größen als positive Zahlen aufgefaßt. Die TK des Ergebnisses ist MAX(TK(A), TK(B)).

Bei ML und DV muß es sich um Festkommazahlen (TK=1) handeln.

Die Vergleichsoperatoren =, ne, lt, le, ge, gt können auch auf Werte mit TK≠0 angewandt werden.

Bei = und ne wird der Wert der Bits 1-48 verglichen.
Bei TK=0 wird zuvor normalisiert.

Bei den Operationen lt, le, ge, gt bestimmt die größte der beiden Typenkennungen den Vergleich.

Bei TK=1 werden die Zahlen mit Vorzeichen verglichen.
Bei TK=2,3 werden die Bits 1-48 als positive Zahlen aufgefaßt und verglichen.

Für die boolschen Operationen mit Bitmustern gibt es:

3.7. type procedure AUT(A,B);
value A,B; type A,B;
Funktionswert = not (A equiv B)(bitweises exklusives oder)

3.8. type procedure VEL (A,B);
value A,B; type A,B;
Funktionswert = A or B (bitweises oder)

3.9. type procedure ET(A,B);
value A,B; type A,B;
Funktionswert = A and B (bitweises und)

Die Größen A und B können beliebige Typenkennung haben. Das Ergebnis hat wiederum die Typenkennung T_{max}. Die

Operationen werden auf den 48 Bits des Ganzwortes bitweise durchgeführt.

Um Festkommakonstanten zu erhalten kann FK benutzt werden. Um aber auch Konstanten aus Tetraden und Oktaden zu erhalten dient die Prozedur IE (Informationseinheit).

3.10. form procedure IE (S,T);
value S,T; string S; integer T;
Der string S wird als Hexadezimalzahl aufgefaßt. D.h. er muß aus den Zeichen 0,1,...,9,A,...,F bestehen. Diese Hexadezimalzahl wird mit der Typenkennung | T | versehen. Falls T < 0, wird sie außerdem linksbündig abgelegt. Der so erhaltene Wert ist der Funktionswert.

Beispiel 3.1

IE('('AF73')',-2) liefert das Ganzwort:

2 AF7300000000

als Funktionswert.

IE('('123 456')',3) liefert das Ganzwort:

3 000000123456

als Funktionswert.

Zwischenräume im string S sind ohne Bedeutung.

Eine besondere Form ist der Aufruf:

3.10 A type 3 procedure IE(S);
value S; string S;
Falls IE mit nur 1 Parameter aufgerufen wird, ist das Prozedurergebnis der string S bzw. die 6 ersten Zeichen des string. Mit Hilfe von REF oder PARV kann auch auf die Adresse dieses strings (d.h. des 1. Wortes aus Alphazeichen) zugegriffen werden. Dabei ist jedoch zu beachten, daß Konstanten schreibgeschützt abgelegt werden. Die gleiche Leistung wie der Aufruf IE(S) erbringt auch EQUIV(S); (Algol-Handbuch [5]).

Für weitere Manipulation an einzelnen Bits sei auf Kapitel 4 und die Algol Prozeduren ASKBIT und SETBIT (N31.FO.11 [8]) verwiesen.

4. Spezielle Maschinenbefehle

Für einige Zwecke, wie Zeichenverarbeitung, Typenkennung, Tabellensuchen, Shiften, gibt es spezielle Hardwarebefehle, die diese Aufgaben schnell erledigen, jedoch von den höheren Programmsprachen aus nicht angesprochen werden können. Dazu gibt es noch die SSR (Springe ins System)-Befehle, die in beliebiger Form auch nur in TAS gegeben werden können. Durch spezielle Prozeduren sollen diese Leistungen auch in Algol ansprechbar sein.

Zur Zeichenverarbeitung gibt es:

4.1. fix procedure BNZ (IL,IR);
fix IL;address IR;
(Bringe Nächstes Zeichen)
Der Aufruf wirkt wie der Hardware-Befehl BNZ IL IR. Der neue Akkuinhalt ist das Funktionsergebnis.

IR ist die Adresse des Ganzwortes aus dem das Zeichen geholt wird.

IL = f*4096+b in Festkomma.
Dabei ist a die Anzahl der bits, die die Zeichen besitzen sollen. Also:

4	Tetrade
6	Hexade
8	Oktade
12	Viertelwort.

b ist die Nummer des Zeichens im Wort, also
0,1,...,d wenn d = (48/f)-1 ist.

Nach Ausführung des Befehls haben die Variablen IL und IR folgende neue Werte:

b := b+1, wenn b < d

b := +0
⟨IR⟩ := ⟨IR⟩ +2
} wenn b = d

Beispiel 4.1

Aus einer Zeichenfolge soll ab Adresse A die 3. bis 5. Tetrade nach B, C, D gespeichert werden.

```
IL := FK(4*4096+3);
IR := A;
B  := BNZ(IL,IR);
C  := BNZ(IL,IR);
D  := BNZ(IL,IR);
```

Analog arbeitet die Prozedur zum Wegspeichern von Zeichen.

4.2. procedure CNZ(IL,IR,W);

value W; form W; adress IR; fix IL;

Das Zeichen W wird entsprechend IL und IR weggespeichert.

Die Typenkennung des Wortes,in das gespeichert wurde, wird auf 3 gesetzt.

Für die Anwendung von BNZ, CNZ

Beispiel 4.2

Von Adresse A Zeichen Nr. 2 sollen N Oktaden nach Adresse B Zeichen Nr. 4 transportiert werden.

```
IL := FK(8*4096+2);
IR := A;
ILC := FK(8*4096+4);
IRC := B;
for i := 1 step 1 until N do
CNZ(ILC, IRC, BNZ (IL,IR));
```

Für den Transport von Oktaden existieren noch spezielle Prozeduren:

4.3. procedure TOK (A,M,B,N,K,V);

address A,B;fix M,N,K; real V;

A sei die Adresse der Oktade A_0,

B die Adresse der Oktade B_0.

Ist nun V ≥ 0 erfolgt ein Transport der Oktaden:

$$A_{M+I} \longrightarrow B_{N+I} \quad \text{für } 0 \leq i \leq K.$$

Ist dagegen V < 0 , so erfolgt der Transport

$$I_{M+I} \longrightarrow B_{N+I} \quad \text{für } 0 \geq i \geq -K.$$

(Also Rückwärtstransport).

Die Werte von M und N dürfen dabei auch größer als 5 sein. Intern werden dabei die Hardwarebefehle BNZ und CNZ aufgerufen. Im Gegensatz zu den Prozeduren BNZ und CNZ erspart man die jeweilige Parameterübergabe.

Eine analoge Prozedur zum Vergleich von Oktadenstrings ist:

4.4. boolean procedure VOK (A,M,B,N,K);
address A; B; fix M,N,K;
A,M,B,N,K haben die gleiche Bedeutung wie in Prozedur TOK. Nur wird hier kein Transport ausgeführt, sondern die strings werden verglichen.
Ist:

$$A_{M+I} = B_{N+I} \quad \text{für} \quad 0 \leq I \leq K ,$$

so ist das Ergebnis true, sonst false.

Beispiel 4.3

Die Prozedur VSTR soll 2 Algol-strings auf Gleichheit in Länge und Inhalt prüfen.

```
boolean procedure VSTR;
begin
procedure PARV(X); code;
boolean procedure VOK(X); code;
integer procedure FK(X); code;
procedure RETURN(X); code;
real A,B,C,D;
PARV(1,A,B);
PARV(2,C,D);
if C ne A then RETURN (false) else
RETURN (VOK(B,FK(6),D,FK(6),A));
end;
```

Ein besonderes Kennzeichen der TR 440 Hardware ist die Typenkennung. Für den Algol-Programmierer ist diese bisher nur über ASKBIT und SETBIT zugänglich.

Zur Behandlung der Typenkennung dienen die Prozeduren

4.5. integer procedure TK(A);
value A; type A;
Das Funktionsergebnis ist die TK von A in Gleit-

kommadarstellung.

4.6. procedure ZT0(A);
type A;
Die TK von A wird auf 0 gesetzt.
Analog für TK 1,2,3 arbeiten die Prozeduren:

4.7. procedure ZT1(A);
type A;

4.8. procedure ZT2(A);
type A;

4.9. procedure ZT3(A);
type A;
Außerdem als Funktion:

4.10. type procedure ZT(A,T);
value A,T; type A; integer T;
Das Funktionsergebnis hat den Wert von A, aber mit der Typenkennung T. T muß den Wert 0,1,2 oder 3 haben.

Um Dienstleistungen des Betriebssystems anzusprechen, dient der Hardwarebefehl SSR (Springe ins System und Reserviere). In BOGOL wird er durch folgende Prozedur zugänglich:

kommadarstellung.

4.6. procedure ZTO(A);
type A;
Die TK von A wird auf O gesetzt.
Analog für TK 1,2,3 arbeiten die Prozeduren:

4.7. procedure ZT1(A);
type A;

4.8. procedure ZT2(A);
type A;

4.9. procedure ZT3(A);
type A;
Außerdem als Funktion:

4.10. type procedure ZT(A,T);
value A,T; type A; integer T;
Das Funktionsergebnis hat den Wert von A, aber mit der Typenkennung T. T muß den Wert 0,1,2 oder 3 haben.

Um Dienstleistungen des Betriebssystems anzusprechen, dient der Hardwarebefehl SSR (Springe ins System und Reserviere). In BOGOL wird er durch folgende Prozedur zugänglich:

Möglich sind also die Aufrufe:

SSR (TYP,VB1)	keine Ergebnisübergabe
A := SSR (TYP,VB1)	Ergebnis in A
SSR (TYP,VB1,REG1)	Ergebnis in REG1 und folgenden Zellen.

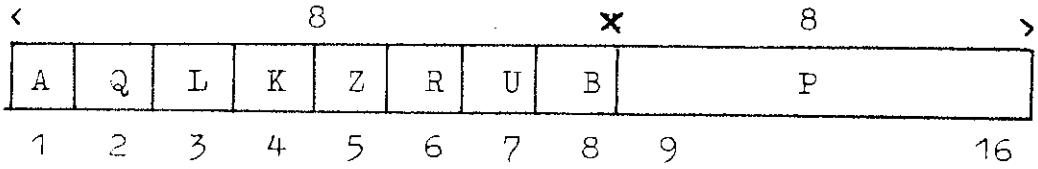
Bezüglich der verschiedenen SSR-Befehle siehe [9] .

Zum Shiften dient die Prozedur:

4.12. procedure SH(A,S,P);
value S,P; type A; form S,P;
Der Inhalt von S ist die Adresse des Shift-Befehls. Falls der Aufruf mit 3 Parametern erfolgt, wird dazu noch der Inhalt von P addiert.

A ist das zu shiftende Wort und wird ins A-Register gebracht, das Q-Register wird mit dem nächsten Ganzwort gefüllt. Nach dem Shiften werden A- und Q-Register wieder zurückgespeichert. Ein Shiften im Q-Register verändert also auch das auf A folgende Wort.

Der Adressteil eines Shift-Befehls hat folgende Gestalt:



Bezüglich der Bedeutung der einzelnen Bits siehe Befehlsliste [10] Seite 17 und 23.

Ein einfaches Shiften von Zahlen (TK=1) ist natürlich auch durch Multiplikation oder Division möglich.

Eine wesentliche Beschleunigung des Tabellensuchens bedeutet die Prozedur

4.13. fix procedure TLI(A,B);
value A,B; address A; type B;
Von der Adresse von A an wird in einer Liste solange gesucht, bis entweder der Wert B gefunden wird oder ein Ganzwort mit einer von B verschiedenen Typenkennung. Falls der Wert B gefunden wird,wird die Adressendifferenz der Zelle (mit Inhalt B) zu A übergeben, wenn nicht der Wert -1 (in Festkomma).

Ähnlich, nur mit anderer Ergebnisübergabe, arbeitet die Prozedur SULI(BO.E2.03 [11]).

Beispiel 4.4

Bezüglich zweier globaler Ganzwortlisten A und B soll die Prozedur ASSOC (X) jeweils das X $\in$ A zugeordnete Y $\in$ B als Funktionswert liefern. Falls X $\notin$ A soll der Funktionswert 0 sein.

```
common ASSOC
array A,B [1:1000] ;
real procedure ASSOC (X);
value X; real X;
begin
integer procedure TLI(X); code; integer procedure VAL(X); code;
integer procedure REF(X); code; integer procedure AD(X); code;
integer procedure FK(X); code; real Z;

Z := TLI(REF(A [1] ),X);
if Z < FK(0) then ASSOC := 0
else ASSOC := VAL (AD(REF(B [1] ),Z));
end;
```

Um einen Zugriff zu Halbworten zu bekommen, gibt es die Halbwortbefehle. Diese werden angesprochen durch:

4.14. fix procedure B2V(A,I);
value A,I;address A; integer I;
Der Funktionswert ist der Inhalt des Halbwortes mit der Adresse A+I.
I wird in Gleitkomma angegeben.

Der Aufruf

4.14 A B2V(A);

liefert den Wert des Halbwortes mit Adresse A.

4.15. procedure C2(A,W);
value W; address A; type W;
Der Wert W wird auf das Halbwort mit der Adresse A abgespeichert.

Um schließlich einen beliebigen Maschinenbefehl ansprechen zu können, dient die Prozedur

4.16. procedure T (OP,AD,REG1).
value OP,AD; form OP,AD; var REG1;
Der Befehl mit dem Operationteil OP und der Adresse AD wird ausgeführt. Beide sind in Internform anzugeben. Sie können jedoch z.B. mit LOC und FK beschafft werden. Vor Ausführung des Befehls werden

die Registerstände mit dem Inhalt aus Zelle REG1 und folgende gefüllt, und nach Ausführung des Befehls werden sie wieder dorthin gespeichert. (Bezügl. Registerabspeicherung vgl. Prozedur SU 4.18.).

Beispiel 4.5

Als Beispiel für die Ausführung eines beliebigen TAS-Befehls soll die doppelt-genaue Gleitkomma-Multiplikation genommen werden. Dazu dient der Befehl DML. Der Interncode von DML ist F2, dezimal 242. Der 1. Faktor steht im A-und Q-Register. Der 2. Faktor in 2 Ganzwörtern mit aufeinanderfolgenden Adressen.

Das Resultat wieder in A und Q.

Die Prozedur hat 6 Parameter.

A und AA bilden den 1. Faktor,

B und BB den 2. Faktor und

C und CC das Ergebnis.

```
procedure DOPMULT (A,AA,B,BB,C,CC);
begin
real RB,RA,RQ,RD,RH; real OP1,OP2;
procedure T(X); code;
RA := A;
RQ := AA;
```

```
OP1 := B;
OP2 := BB;
T(FK(242),REF(OP1),RB);
C   := RA;
CC  := RQ;
end;
```

Um jedoch TAS-Sequenzen aus mehreren Befehlen evtl. öfter zu durchlaufen, dienen die Prozeduren:

4.17. procedure TAS(B1,OP_1,AD_1,...,OP_n,AO_n);
value OP_1,AD_1,...,OP_n,AD_n; var B1;
form OP_1, AD_1,...,OP_n,AD_n;
Die Befehle mit Operationsteilen OP_v und den Adressen AD_v werden auf Zelle B1 und folgende abgelegt. Insgesamt werden n+1 Halbworte benötigt. Der Ansprung erfolgt durch einen SUE-Befehl.

Zur Ausführung dieser TAS-Sequenz dient dann die Prozedur

4.18. SU (B1, REG1);
value B1; var B1, REG1;

Die TAS-Sequenz ab Zelle B1 wird durchlaufen. Zuvor werden die Register aus REG1 geholt und anschließend wieder dorthin sichergestellt.

Benötigt man nicht einen bestimmten Registerinhalt, so kann man den Aufruf:

4.18 A SU(B1)

wählen.

Die Register werden von REG1 an jeweils in folgender Form abgespeichert:

Die Adresse von REG1 sei n. Dann wird:

$\langle n \rangle_{25-48}$ = B
$\langle n+2 \rangle$ = A-Register
$\langle n+3 \rangle$ = Q-Register
$\langle n+4 \rangle$ = D-Register
$\langle n+6 \rangle$ = H-Register

5. Systemdienste

Allgemein können Systemdienste durch die Prozedur SSR angesprochen werden. Für die Alarmbehandlung existieren jedoch spezielle Prozeduren.

5.1. procedure ALSETZ(L);
integer label L;
Die Alarmadresse des Operators, die sich normalerweise in der Kontrollprozedur befindet, wird auf die Adresse des integer-labels L umgesetzt. Beim Aufruf als integer-procedure erhält man als Funktionsergebnis die alte Alarmadresse (mit TK=2) geliefert.
Achtung: Der Label L sollte im äußersten Block des Hauptprogramms liegen.

5.2. procedure ALTEST(I);
var I;
Hierbei wird mittels SSR 4 8 die Alarmursache untersucht und entsprechend Variable I gesetzt. Dabei bedeutet:

i	Alarmursache
0	Interrupt
1	Arithmetischer Alarm
2	Typenkennungs-Alarm
3	Speicherschutz-Alarm
4	Überlauf Register U
5	Befehlsalarm
6	Dreierprobenalarm

Einen Interrupt erhält man, falls nach ␣ XAN ␣. die Anweisung HALT, ⟨ OLN ⟩ gegeben wurde.

Nach einem Alarm ist zunächst ALTEST aufzurufen, danach muß ALSETZ gegeben werden.

Literaturverzeichnis

[1] Bachus, J.W. et.al.:
Revised Report on the Algorithmic Language ALGOL 60
Numerische Mathematik 4 (1963), p.420
Comm. ACM 6 (1963) pp. 1-17

[2] D.J. Farber, R.E. Griswold and I.P. Poloysky
SNOBOL, a string manipulation language
Journal of the ACM, 11(1964), pp.21-30

[3] J.McCarthy et.al.
LISP 1.5 Programmer's Manual
MIT Press 1962

[4] M.Jäger, M.Rosendahl und R.Staake
Einführung in die Listenverarbeitung anhand der Dialogsprache AIDA
Arbeitsberichte des Rechenzentrums der Ruhr-Universität Bochum, 7203, 1972

[5] TR 440 Algol-Sprachbeschreibung
TR 440 Unterlagensammlung N31.D1.04

[6] TR 440 Telefunken Assemblersprache TAS
TR 440 Unterlagensammlung N31.D2.11

[7] M. Richards
BCPL: A tool for compiler writing and system programming.
AFIPS Vol. 35 Spring Joint Comp. Conference 1969, pp. 557-566

Zum BOGOL-TAS-System werden in unregelmäßiger Folge Ergänzungen und Änderungen erscheinen. Diese werden in der COMPuterPOST angekündigt und in der Programmbibliothek bereitgestellt.

Bisher erschienene Arbeitsberichte des Rechenzentrums der Ruhr-Universität Bochum :

Nr. 7101 : K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, eine Dialogsprache für den TR 440

Nr. 7102 : K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, ein Dialogsystem und seine Implementierung in ALGOL

Nr. 7103 : K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, Manual für den Benutzer

Nr. 7104 : 4. Jahresbericht des Rechenzentrums (Juni 1970 bis Juni 1971)

Nr. 7105 : H. Wupper
WR MBO2 - Ein einfaches Band-Betriebssystem für einen mittleren Rechner

Nr. 7201 : H. Windauer
Existenzsätze zur (0,1,...,R-2,R) - Interpolation

Nr. 7202 : W. Schelongowski
DIATRACE, Ein System zur interaktiven Assemblerprogrammierung

Nr. 7203 : M. Jäger, M. Rosendahl, R. Staake
Einführung in die Listenverarbeitung anhand der Dialogsprache AIDA

Nr. 7204 : R. Mannshardt, P. Pottinger
Einführung in die Benutzung des Teilnehmer-Rechensystems TR 440 in der RUB

Nr. 7205 : 5. Jahresbericht des Rechenzentrums (1.7.1971 bis 30.6.1972)

Nr. 7206 : M. Rosendahl
BOGOL-TAS, ein Weg zur systemnahen Programmierung in ALGOL am TR 440

Nr. 7207 : W. Stark
ILW, Programmsystem zur Berechnung des Instationären Ladungswechsels von Verbrennungskraftmaschinen (Modulbeschreibung und Eingabekonventionen)