STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

REKENAFDELING

MR 55 - Intern Rapport

Objectprogramma, gegenereerd door de M.C. vertaler

Dr.E.W. Dijkstra

Januari 1963

The Mathematical Centre at Amsterdam, founded the 11th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications, and is sponsored by the Netherlands Government through the Netherlands Organization for Pure Research (Z.W.O.) and the Central National Council for Applied Scientific Research in the Netherlands (T.N.O.),by the Municipality of Amsterdam and by several industries.

Objectprogramma, gegenereerd door de MC-vertaler.

In eerste instantie formuleert de MC-vertaler het object-programma in een aantal standaardhandelingen, welke genummerd zijn door een zg. Outputcode (OPC). Wij beginnen met een lijst hiervan

8	ETMR	EXTRANSMARK	RESULT
9	ETMP	EXTRANSMARK	PROCEDURE
10	FTMR	FORMTRANSMARK	RESULT
11	FTMP	FORMTRANSMARK	PROCEDURE
12	RET	RETURN	
13	EIS	END OF IMPLICIT SUBROUTINE	
14	TRAD	TAKE REAL ADDRESS DYNAMIC	
15	TRAS	TAKE REAL ADDRESS STATIC	
16	TIAD	TAKE INTEGER ADDRESS DYNAMIC	
17	TIAS	TAKE INTEGER ADDRESS STATIC	
18	TFA	TAKE FORMAL ADDRESS	
19	FOR0		
20	FOR1		
21	FOR2		
22	FOR3		
23	FOR4		
24	FOR5		
25	FOR6		
26	FOR7		
27	FOR8		
28	GTA	GOTO ADJUSTMENT	
29	SSI	STORE SWITCH INDEX	
30	CAC	COPY BOOLEAN ACC. INTO CONDITION	
31	TRRD	TAKE REAL RESULT DYNAMIC	
32	TRRS	TAKE REAL RESULT STATIC	
33	TIRD	TAKE INTEGER RESULT DYNAMIC	
34	TIRS	TAKE INTEGER RESULT STATIC	
35	TFR	TAKE FORMAL RESULT	
36	ADRD	ADD REAL DYNAMIC	
37	ADRS	ADD REAL STATIC	
38	ADID	ADD INTEGER DYNAMIC	
39	ADIS	ADD INTEGER STATIC	
40	ADF	ADD FORMAL	
41	SURD	SUBTRACT REAL DYNAMIC	
42	SURS	SUBTRACT REAL STATIC	
43	SUID	SUBTRACT INTEGER DYNAMIC	
44	SUIS	SUBTRACT INTEGER STATIC	
45	SUF	SUBTRACT FORMAL	

46	MURD	MULTIPLY REAL DYNAMIC
47	MURS	MULTIPLY REAL STATIC
48	MUID	MULTIPLY INTEGER DYNAMIC
49	MUIS	MULTIPLY INTEGER STATIC
50	MUF	MULTIPLY FORMAL
51	DIRD	DIVIDE REAL DYNAMIC
52	DIRS	DIVIDE REAL STATIC
53	DIID	DIVIDE INTEGER DYNAMIC
54	DIIS	DIVIDE INTEGER STATIC
55	DIF	DIVIDE FORMAL
56	IND	INDEXER
57	NEG	INVERT SIGN ACCUMULATOR
58	TAR	TAKE RESULT
59	ADD	ADD
60	SUB	SUBTRACT
61	MUL	MULTIPLY
62	DIV	DIVIDE
63	IDI	INTEGER DIVISION
64	TTP	TO THE POWER
65	MOR	MORE >
66	LST	AT LEAST ≥
67	EQU	EQUAL =
68	MST	AT MOST ≤
69	LES	LESS <
70	UQU	UNEQUAL ≠
71	NON	NON ¬
72	AND	AND ∧
73	OR	OR ∨
74	IMP	IMPLIES ⊃
75	QVL	EQUIVALENT ≡
76	abs	
77	sign	
78	sqrt	
79	sin	
80	cos	
(81	arctan)	
82	ln	
83	exp	
84	entier	
85	ST	STORE
86	STA	STORE ALSO
87	STP	STORE PROCEDURE VALUE
88	STAP	STORE ALSO PROCEDURE VALUE

```
89    SCC       SHORT CIRCUIT
90    RSF       REAL ARRAYS STORAGE FUNCTION FRAME
91    ISF       INTEGER ARRAYS STORAGE FUNCTION FRAME
92    RVA       REAL VALUE ARRAY STORAGE FUNCTION FRAME
93    IVA       INTEGER VALUE ARRAY STORAGE FUNCTION FRAME
94    LAP       LOCAL ARRAY POSITIONING
95    VAP       VALUE ARRAY POSITIONING

96    START     aan begin van programma
97    STOP      aan einde van programma

98    TFP       TAKE FORMAL PARAMETER
99    TAS       TYPE ALGOL SYMBOL
100   OBC6      OUTPUT BUFFER CLASS 6
101   FLOATER
102   read
103   print
104   TAB
105   NLCR
106 XEEN
107   SPACE
108   stop
109   P21
```

De arithmetiek

De in de ALGOL-tekst benoemde variablen zijn of boolean of integer en dan beslaan ze één woord in het geheugen, of ze zijn van het type real. In het laatste geval worden ze in P9. pakking gerepresenteerd, beslaan in het geheugen twee op-eenvolgende adressen. Bij referentie naar een drijvend getal wordt het adres van het kopwoord gegeven.

Voorts treden in de arithmetiek de zg. anonyme tussenresultaten op: deze worden opgeborgen in de "accumulatorenstapel". Een dergelijke accumulator kan drie verschillende soorten inhoud hebben; een accumulator uit de stapel beslaat vier opeenvolgende adressen.

1) Inhoud accumulator van type real

```
0   } numeriek gedeelte
1   }
2     binaire exponent
3     = -1
```

Opm. In systeem ALS is plaats 1 ongebruikt.

2) Inhoud accumulator van type integer

```
0     numeriek gedeelte
1   } ongebruikt
2   }
3     = -0
```

De boolean variabele wordt als integer behandeld en wel in de toevoeging

+0 ↔ true
+1 ↔ false

3) Inhoud accumulator een physisch adres

0 adres + "integerbit"
1 } ongebruikt
2 }
3 ≥ + 0

Plaats 0 van de accumulator bevat een physisch adres op de laagste vijftien plaatsen; voorts is d19=1, als hierbij verwezen wordt naar een integer, anders is d19=0. De andere bits van woord 0 zijn altijd =0.

Woord 3 is nonnegatief (ter onderscheiding van het geval van numerieke accumulatorinhoud) en, geloof ik, altijd gelijk gekozen aan woord 0.

Arithmetiek speelt zich in principe af op twee op eenvolgende accumulatoren met numerieke inhoud. In eerste instantie geschiedt de arithmetiek door de operaties 59 t/m 64 (zie OPC-lijst).

Als wij uit moeten voeren "a operatie b" (operatie zijnde +, -, *, /, + of ↑), dan is het accumulator beeld

vlak voor de operatie

α+0 }
1 } a
2 }
3
4 }
5 } b
6 }
7 }
AP → 8

vlak na de operatie

α+0 }
1 } a operatie b
2 }
3
AP → 4
5
6
7
8

De grootheid AP, Accumulator Pointer, wordt onthouden op adres 18 X1; AP is het beginadres van de eerste vrije accumulator, of in andere woorden: het adres van de eerste vrije plaats volgend op de laatst gevulde accumulator. Voor de operatie is AP=α+8, na de operatie is AP=α+4. Behalve dat de operaties 59 t/m 64 de waarde van AP met vier verminderen, nemen zij er uitgebreid notavan: de operanden immers worden geselecteerd en het resultaat wordt afgeleverd onder controle van de heersende waarde van AP.

De operaties 59 t/m 64 zijn "adresloos"; in het objectprogramma worden zij door een enkele subroutinesprong gerepresenteerd. Zij werken bij alle vier de mogelijke

type-combinaties (real ↔ integer) der accumulator-inhouden.

Operaties, die het aantal gevulde accumulatoren met 1 vermeerderen (als onderdeel van hun werkzaamheden dus "AP/=AP+4" bevatten) zijn in eerste instantie de operaties met OPC 14 t/m 18, die de eerst volgende accumulator met een adres vullen, en de operaties 31 t/m 35, die de eerstvolgende accumulator met een getal vullen. Deze outputcodes stellen weer subroutinesprongen voor, in het objectprogramma vergen zij in hun geheel twee opdrachten, omdat voor de subroutinesprong een 15 bit parameter in het S- of B-register van de machine gezet moet worden.

De gevallen 15, 17, 32 en 34 ("STATIC") zijn van toepassing als het physisch adres (van de grootheid) bekend is; voor wat de MC-vertaler produceert beperkt dit zich tot de variabelen, die in het hoofdprogramma gedeclareerd zijn en de own-gedeclareerden.
In het geval "STATIC" moet dit physische adres in het B-register worden meegegeven.

In de gevallen 14, 16, 18, 31, 33 en 35 ("DYNAMIC" of "FORMAL") moet de 15-bits-parameter in het S-register worden meegegeven. Deze parameter is dan

$$32p + bn$$

waarbij bn ($1 \leq bn \leq 31$) als bloknummer een bepaalde PP (=Parameter Pointer) aanwijst, die met p (=positie t.o.v. PP) vermeerderd moet worden, om het physische adres van primaire interesse te geven. Het uiteenrafelen van de parameter in S, het opzoeken van de PP en het opstellen van p is een standaard-onderdeel van alle operaties met het praedicaat DYNAMIC of FORMAL.

In verband met deze non-arithmetische operaties kunnen wij nog noemen TAR, die alleen gegeven mag worden, als de jongste accumulator een adres bevat: onder controle van dit adres wordt in het geheugen een getal opgezocht dat dan in diezelfde accumulator wordt geplaatst. TAR laat het aantal gevulde accumulatoren ongewijzigd en "vervangt de inhoud van de jongste accumulator door de inhoud van zijn inhoud". De vertaler maakt van TAR slechts na Ind (zie later) gebruik.

<u>OPM.</u> De opeenvolging "TRA TAR"] Dit geldt ook voor integers, maar het heeft
is dus equivalent met "TRR"] niet voor formelen te gelden!

Het totaal aantal gevulde accumulatoren blijft eveneens ongewijzigd bij de "geadresseerde arithmetische operaties" (OPC van 36 t/m 55). Zij zijn logisch niet noodzakelijk maar ingevoerd om het objectprogramma wat com-

pacter te kunnen bergen.

Zo zou de operatie SURS (SUBTRACT REAL STATIC) opgebouwd kunnen worden uit de (eveneens geadresseerde) operatie TRRS (TAKE REAL RESULT STATIC) gevolgd door SUB (SUBTRACT). Dit is ook, wat in feite gebeurt: de geadresseerde arithmetiek laat dan "netto" wel het aantal gevulde accumulatoren constant, maar de inhoud van de eerste vrije accumulator wordt daarbij zeker verstoord. (Verwijst het "adres" naar een formele grootheid, dan worden het er, als de overeenkomstige actuele parameter een expressie of een geindiceerde variabele is, zelfs veel meer).

Er is een adresloze operatie, die het aantal gevulde accumulatoren met twee verlaagt, n.l. de STORE. Deze wordt gegeven, als de jongste accumulator een numerieke inhoud heeft, en de op een na jongste een adres. Het getal in de jongste accumulator wordt opgeborgen op de plaats, die in de op een na jongste accumulator wordt aangegeven. Beide accumulatoren worden dan vrijgegeven: hun inhoud is immers definitief afgehandeld.

De operatie "STORE ALSO" begint als de STORE, echter wordt het getal, dat immers actueel blijft, een accumulator teruggeschoven en AP wordt totaal slechts met 4 verlaagd, zodat uiteindelijk de opgeborgen waarde in de dan jongste accumulator behouden blijft.

Locale grootheden

Vier administratieve grootheden zijn van voortdurend belang: het zijn

[17 X1] = WP = Workingspace Pointer
[18 X1] = AP = Accumulator Pointer
[19 X1] = PP = Parameter Pointer
[20 X1] = BN = Block Number

De waarde van BN wordt tijdens vertaling voor elk blok vastgesteld en geeft aan, door hoeveel blokken het blok in kwestie lexicographisch omvat wordt. Het hoofdprogramma heeft dus BN=0; alle MCP's, die gebruikt kunnen worden, alsof ze aan het begin van dit blok gedeclareerd waren, kunnen geacht worden BN=1 te krijgen.

De functie van AP is in het voorafgaande beschreven; een analyse van het vullen en legen van accumulatoren leert ons, dat AP tussen de statements van een zelfde blok steeds terugkeert tot dezelfde waarde.
Deze "rustwaarde", het nulpunt van de accumulatorenstapel, behorende bij deze incarnatie van het blok, wordt vastgelegd in de wijzer WP.

De wijzer PP wijst een punt in de stapel aan, ten opzichte waarvan de locale grootheden van dit blok gelocaliseerd zijn.

Elk blok, elk procedurelichaam, moet in het objectprogramma beginnen met het volgende tweetal opdrachten

```
2B   "b n"  A
    SCC  (= OPC 89)
```

als "bn" het bloknummer van het blok in kwestie is. Voor elke MCP kiezen we bn=1. (Ze zijn immers in het gehele ALGOL-programma ter beschikking gedragen zich dus als procedures, die aan het begin van het hoofdprogramma, dwz in het enige blok 0, gedeclareerd zijn. Eigen bloknummer =1 is dus organisatorisch correct.)

Het stapelbeeld vanaf het punt aangewezen door de dan heersende waarde van PP is na dit tweetal opdrachten als volgt

```
PP → 0  PP     van jongste incarnatie van lexicographisch omvallend
                                                          blok
     1  WP   ]  heersende waarden op het moment van de call,
     2  link |  betrekking hebbend op de jongste incarnatie
     3  PP   |  van het blok, dat de procedure statement be-
     4  BN   ]  vat
     5       ]  1ste PARD
     6       ]
     7       ]  2de PARD
     8       ]
     9
```

AP, WP → 5+2*n als n het aantal parameters van de procedure in kwestie is.

Als het blok een non-procedureblok is, gelden er dus enige relaties: (PP+0)=(PP+3), want een dergelijk blok wordt slechts geactiveerd vanuit het blok, waarin het gedefinieerd is. Voorts geldt AP=WP=PP+5, omdat het aantal parameters nihil is. Aan de gelijkheid AP=WP is, omdat we ons na SCC nog niet "in een statement" bevinden, altijd voldaan, ook aan de ingang van een procedurelichaam.

Indien in het blok een aantal locale scalairen gedeclareerd worden, is de ruimte, die zij in de stapel innemen bekend tijdens vertaling. Zij worden gedacht aansluitend aan de laatste PARD. Nemen zij totaal M geheugenplaatsen in beslag, dan geeft dit in het objectprogramma volgend op SCC aanleiding tot de volgende drie opdrachten

```
2A  "m"  A
4A  17   X1
4A  18   X1
```

dwz. WP en AP worden samen zover opgeschoven dat er in de stapel ruimte voor de locale scalairen vrijkomt.
De introductie van locale arrays komt later aan de orde.

Het aanroepen van een procedure

Voor het aanroepen van een procedure staan ons vier mechanismen ter beschikking, OPC 8 t/m 11.
In alle vier de mechanismen wordt het aantal parameters in het A-register meegegeven. De procedure zelf wordt in de gevallen ETMR en ETMP (de non-formele procedure) gekarakteriseerd doordat in het B-register het physisch beginadres van de procedure wordt meegegeven, in de gevallen FTMR en FTMP (de formele procedure) wordt de positie van de PARD als dynamisch adres in het S-register meegegeven.

Om der wille van de volledigheid geven wij het stapelbeeld, dat door deze vier aanroepmechanismen geschapen wordt.

Bij ETMP:

AP oud → -0
PP nieuw → zie verder pg 7

Bij ETMR: (Netto effect van activering: AP=AP+4)

AP oud → } reservering van accumulator voor het resultaat van de "function designator"

AP oud = beginadres van de resultaat-accumulator
PPnieuw → zie verder pg 7 .

Bij FTMP:

AP oud → link 1
link 2
kop van PARD met d19=0
-0
PP nieuw → zie verder pg 7

Bij FTMR: (netto effect van activering AP=AP+4)

AP oud → } vier gereserveerde plaatsen voor de resultaat-accumulator

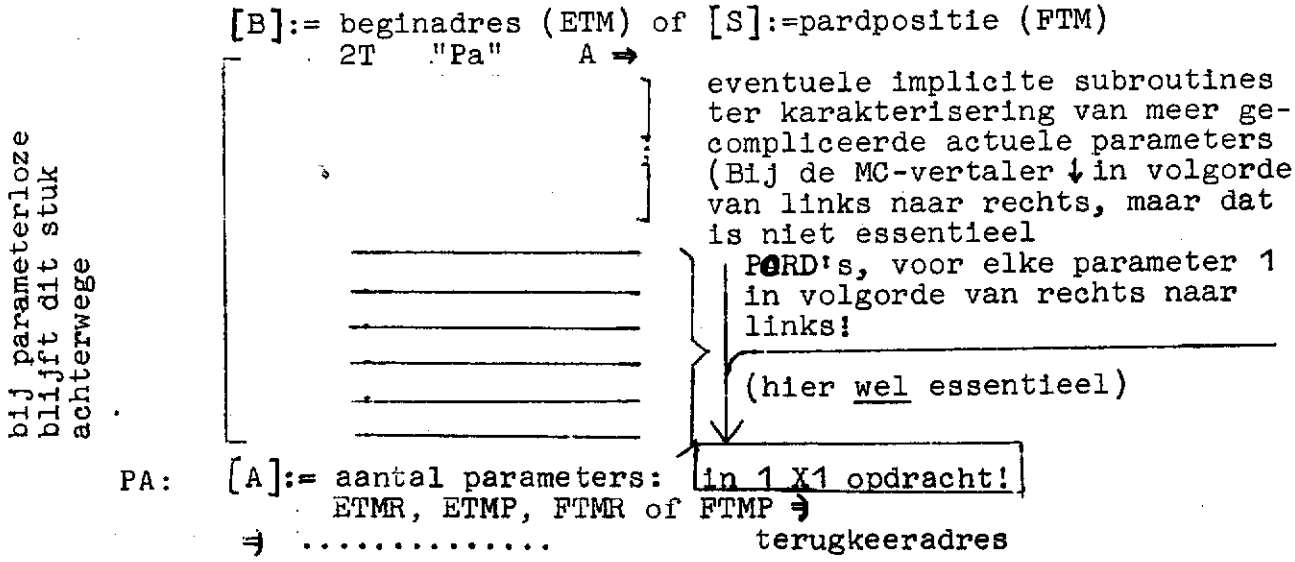
link 1
link 2
kop van PARD met d19=0
AP oud, dur. beginadres van resultaat-accumulator
PP nieuw → zie verder pg

Het laatste stapelbeeld kan ook optreden in TFR en TFA, nl. wanneer de actuele parameter door middel van een implicite subroutine gegeven is (zie later). In de regel "kop van PARD" met d19=0" staat echter "kop van PARD met d19=1" in het geval van TFR, waar de implicite subroutine ~~p= 15 maar~~ een adres blijkt af te leveren (actuele parameter van de vorm "geindi-

ceerde variabele"). In dit geval moet de operatie TAR nog uitgevoerd worden.

In de gevallen FTMR, FTMP, TFA en TFR staat het eigenlijke terugkeeradres op de plaats link 1, want op de plaats "PP nieuw + 2" staat (zie pag. 7)een terugverwijzing naar het FTM-mechanisme, dat o.a. onderzoekt, of de operatie TAR ingelast moet worden en uiteindelijk de besturing via link 1 terugstuurt. Link 2 moet gered en hersteld worden, omdat TFR als onderdeel van arithmetische subroutines (ADF, SUF, MUF en DIF) in actie kan komen.

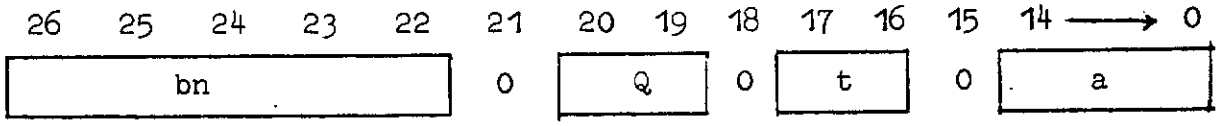
In het objectprogramma ziet de procedure aanroep er als volgt uit:



De PORD's en de PARD's

Het mechanisme TRANSMARK vertaalt PORD's in PARD's. De PORD's staan in het objectprogramma (in teruglopende volgorde, te beginnen bij het terugkeeradres - 3, zie pg 10) en beslaan elk 1 woord, de hieruit afgeleide PARD's worden in oplopende volgorde in de stapel gezet en beslaan elk twee woorden (zie pg 7).

De structuur van een PORD is als volgt:

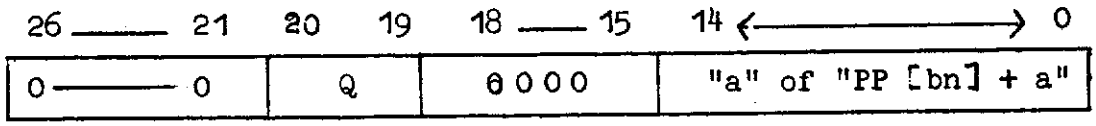


Indien t=0, dan zijn de 15 bits van a een physisch adres (grootheid uit blok 0, beginadres van procedure of impliciete subroutine); de vijf bits van bn doen dan niet ter zake en zijn =0.
De waarde t=2 komt niet voor.
Als t=1 of 3, dan zijn de bits van a een positie tov. van een PP, die door de bits van bn is aangewezen en in eerste instantie wordt dus het physisch adres PP [bn] + a, dat een grootheid in de stapel aanwijst opgebouwd.
(Dit is dus het normale "dynamische adres" met dat verschil, dat de bits voor positie en bloknummer zich in een andere positie bevinden)

Als t=1, dan wordt met het aldus gevormde adres doorgewerkt als in geval t=0 met het direct in a gegeven adres.
Als t=3, - dwz de actuele parameter is op de plaats waar hij gegeven wordt een formele, het zg. "doorgeven van een formele parameter - dan wijst het aldus afgeleide adres gegarandeerd naar een PARD, en deze PARD wordt zonder meer doorgegeven (dus (PP [bn]+ a) en (PP [bn] + a + 1) zijn samen de resulterende PARD).
Als t=3, dan doen de bits van Q niet ter zake, en zijn deze =0.

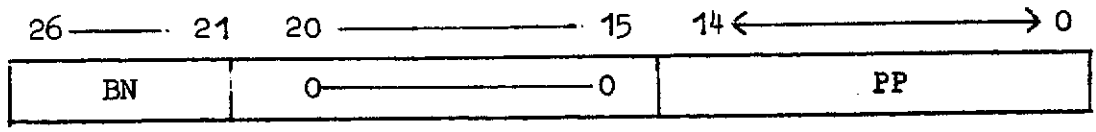
In de gevallen t=0 en 1 bevat het eerste PARD-woord

1ste PARD



in de laagste 15 posities het gecopieerde of afgeleide physische adres en het tweede PARD-woord

2de PARD



dwz. Blocknumber en Parameterpointer, zoals deze heersten op het moment van aanroep. Het tweede PARD-woord, dat altijd gevormd wordt, is slechts van belang, als het physische adres uit het eerste het beginadres is van een procedure of een impliciete subroutine.

Hoe dit physische adres geinterpreteerd moet worden, is in de bits van Q aangegeven.

Q=0 adres van getal in P9-pakking (van real)
Q=1 adres van integer (of boolean)
Q=2 beginadres van procedure of impliciete subroutine zonder of met numeriek resultaat
Q=3 beginadres van impliciete subroutine met een adres als resultaat.

Wie MCP's maakt moet bereid zijn desnoods PARD's te analyseren - doorgaans gebeurt dit in TFR en TFA -; hij moet ook bereid zijn - als zijn MCP een ander aanroept - om PDRD's te construeren. De praktijk heeft geleerd, dat dit werk is, waarbij je erg uit moet kijken.

De impliciete subroutine hoeft niet te beginnen met SCC, want dat is door het aanroepmechanisme van TAKE FORMAL al gebeurd; verhoging van [17 X1] en [18 X1], dwz. WP en AP, hebbenwe evenmin mee te maken en we kunnen dus direct de expressie (of de "left hand side expression" in het geval van een geindiceerde variabele) gaan uitwerken. De impliciete subroutine wordt afgesloten met

EIS (OPC 13),

waardoor het resultaat, dat zich in de jongste accumulator bevindt naar de aanroepende "omgeving" wordt doorgegeven en de RETURN wordt bewerkstelligd.

Dver de implicite subroutines, waartoe labels en switch identifiers als actuele parameter aanleiding geven, zie later.

Labels en switches

Een sprong naar een punt in hetzelfde blok wordt door een normale sprongopdracht vertaald.

Zodra door een sprong een blok verlaten wordt, worden twee opdrachten voor de sprong ingelast, nl.

```
"2B "bn"  A    "bn"=BN van bestemmingsblok
 GTA  (OPC 28: GOTO ADJUSTMENT)
```

Het effect van de GOTO ADJUSTMENT is dat de vitale administratieve toestandsgrootheden (PP, WP, AP, en BN) ingesteld worden op de waarden, die voor de voortzetting van het bestemmingsblok vereist zijn. Hierbij moet vermeld worden dat

1° Impliciete subroutines beschouwd worden als "binnenblok" van het blok, waarin de procedure statement voorkomt ("de parameterhaken fungeren als blokhaken").
2° De range van een for-statement geldt eveneens als binnenblok van het blok, waarin de for-statement voorkomt. Indien een for-statement door een goto-statement verlaten wordt, geeft dit in het objectprogramma dus aanleiding tot blokbeeindiging via een GTA.

Een switch declaratie geeft aanleiding tot een stukje programma, waar we slechts naar toe springen als de waarde van de index (verhoogd met 1) is opgeborgen op het vaste adres 16 X1: op het inspringpunt van het switchprogramma staat de opdracht IT 16 X1; onmiddellijk voor het insprongpunt staat dus een bestemmingslijst (als lijst van sprongopdrachten). Indien een sprong naar een switchelement in het objectprogramma uitgevoerd moet worden, dan wordt eerst de index berekend in de jongste accumulator, dan volgt de operatie

SSI (OPC = 29)

dwz STORE SWITCH INDEX, die de inhoud van de jongste accumulator met 1 verhoogd overneemt in adres 16 X1 (na zonodig op integer representatie te zijn overgegaan). Dan volgt de sprong naar het switchprogramma (eventueel onmiddellijk voorafgegaan door een GTA). Het punt is, dat, hoewel in de declaratie waar een switch gedefinieerd wordt door de opsomming van zijn elementen en deze elementen zelf "designational expressions" zijn, dus mogelijk weer naar switches refereren, er bij de tijdelijke uitvoering niets gerest hoeft te worden en we kunnen volstaan met de reservering van een vaste plaats voor de heersende waarde van de "switch-index".

In de designational expression wordt de switch identifier dus eigenlijk behandeld als label, nl. als label van het bijbehorende switchprogramma. Dit komt het duidelijkst tot uiting als we een label of switchidentifier als actuele parameter meegeven aan een procedure. Dit geeft nl. aanleiding tot een impliciete subroutine, waarin na een

GTA de sprong naar het punt in het programma, dan wel de switch gemaakt wordt. (De vertaler sluit deze impliciete opdracht af met de in dit geval overbodige opdracht EIS, END of IMPLICIT SUBROUTINE).
Moet nu in het procedurelichaam een sprong uitgevoerd worden naar een formele label of het element van een formele switch, dan zorgen we dat (via TFR) deze impliciete subroutine geentameerd wordt we komen hieruit nooit in het procedurelichaam terug. De PORD van een dergelijke impliciete subroutine heeft Q=2.

De dynamische afsluiting van een blok

De normale manier om een blok te verlaten, een procedure te beëindigen is de handeling RET (OPC=12), genaamd "RETURN".

Impliciete subroutines worden echter afgesloten met "EIS" (OPC=13) "END OF IMPLICIT SUBROUTINE", (zie later).

Het blok van de for-statement wordt afgesloten met FOR 8; anders kan een blok alleen maar definitief verlaten worden door tussenlassing van een GTA.

De functie-procedure

ALGOL 60 staat toe, dat een functie-procedure op twee manieren geactiveerd wordt, of via TMR (TRANSMARK RESULT), als onderdeel van een expresse, of via TMP (TRANSMARK PROCEDURE), dwz. als zelfstandige statement. In het laatste geval stelt het aanroepende programma geen belangstelling in de functiewaarde.

In het eerste geval worden vier plaatsen (een accumulator) in de stapel gereserveerd, in het laatste niet (zie pg *g* ~~en~~). In welke van de twee gevallen we ons bevinden, is te vinden op adres "PP nieuw -1". Deze adresplaats moet onderzocht worden, als in het procedurelichaam "assignment to its own identifier" plaats vindt. Is (PP nieuw - 1)=-0, dan is de functie procedure als zelfstandige statement aangeroepen, en moet de waardetransport naar buiten onderdrukt worden. Deze analyse (en eventueel volgende transport) geschiedt door
OPC 87 STP STORE PROCEDURE VALUE
of
OPC 88 STAP STORE ALSO PROCEDURE VALUE,
waarbij het verschil is, dat bij de laatste de aflaging AP:=AP-4 onderdrukt wordt, omdat het tussen resultaat nog actueel blijft.

Omdat echter "assignment to the procedure identifier" in een binnenblok van het procedurelichaam is toegestaan, moeten we bij STP en STAP wel aangeven, welke (PP nieuw - 1)we moeten onderzoeken. Dit geschiedt,

door STP en STAP in te gaan met in het B-register het bloknummer van het blok, waarin de procedure in kwestie gedeclareerd is. Voor een MCP, die zelf BN=1 heeft, wordt dit dus

```
2B   0    A
STP  of  STAP
```

De Operatie EIS is een afkorting van

```
2B  "BN-1"   A
    STP
    RET
```

als "BN" de ter plaatse heersende waarde van het bloknummer is. De vertaler voert deze verkorting uitsluitend in aan het einde van een impliciete subroutine; in MCP's zal van deze verkorting met vrucht gebruik gemaakt kunnen worden.

Arrays

Arrays worden bereikt via hun zg. Storage Function; voor een n-dimensionaal array beslaat de storage function n+3 consecutieve adressen, Referentie naar de storage function geschiedt door opgave van het eerste adres van de storage function.

Laat de indexgrenzen voor het n-dimensionale array A

$$[\, l_o{:}u_o,\ l_1{:}u_1, \ldots\ldots, l_{n-1}{:}u_{n-1} \,]$$

zijn. Dan bevat de storage function

$\alpha + 0$	beginadres van het array	$-\sum_{i=0}^{n-1} l_i * \Delta_i$
$+ 1$	beginadres van het array	
$+ 2$	$\Delta_o = 1$ of 2	
$+ 3$	$\Delta_1 = (1 + u_o - l_o) * \Delta_o$	
	⋮	
$+(n+1)$	$\Delta_{n-1} = (1+u_{n-2}-l_{n-2}) * \Delta_{n-2}$	
$+(n+2)$	$-\Delta_n = -(1+u_{n-1}-l_{n-1}) * \Delta_{n-1}$	

Het adres van element A $[\, j_o, j_1, \ldots, j_{n-1} \,]$ is dan

$$(\alpha+1) + \sum_{i=0}^{n-1} j_i * \Delta_i \quad ;$$

$(\alpha+0)$=het adres van element A $[\, l_o, l_1, \ldots\ldots, l_{n-1} \,]$.

Omdat elke $u_i \geq l_i$ zijn alle Δ's positief; omdat de laatste met gewisseld teken is opgeborgen, bepaalt de storage-function de dimensie van het array. De waarde van Δ_o wordt door de bijbehorende typedeclaratie beheerst.

Indien het array in het hoofdprogramma gedeclareerd is - dan wel van de marker "own" voorzien is - dan is het array tijdens vertaling vast gelocaliseerd en de storagefunction bevindt zich in de constantenlijst, Als het een normaal lokaal array is, dan wordt de storagefunction door het objectprogramma in de stapel opgebouwd.

Dit geschiedt in twee phasen: eerst wordt het "frame van de strorage function" opgebouwd, later wordt de plaats van het array pas vastgelegd.

Het frame van de storagefunction is alles, wat afgeleid kan worden uit type en indexgrenzen: het is de storagefunction, bepaald op een additieve constante in de eerste twee woorden na.

Er zijn twee operaties, die uit een rij onder- en bovengrenzen het frame van de storage function afleiden. Het zijn

OPC 90 RSF REAL ARRAYS STORAGE FUNCTION FRAME
ISF INTEGER ARRAYS STORAGE FUNCTION FRAME

Als parameter krijgen zij in het S register mee, hoeveel replica's van dit frame in de stapel gegenereerd moeten worden (immers de zelfde lijst van indexgrenzenparen kan de afmetingen van een aantal arrays bepalen). Als dit "m" arrays zijn van dezelfde afmetingen, dan ziet het gegenereerde programma er als volgt uit:

$\curvearrowright l_0$
$\curvearrowright u_0$
$\curvearrowright l_1$
$\curvearrowright u_1$

$\curvearrowright l_{n-1}$
$\curvearrowright u_{n-1}$
2S "m" A
RSF of ISF

(waarin met het symbool "$\curvearrowright$" (TAKE) het vullen van de volgende accumulator is aangegeven.
Uit het verschil van WP en AP kan RSF of ISF de dimensie van de arrays vaststellen en de frame's kunnen gemaakt worden. De frame's overschrijven de accumulatoren met indexgrenzen geheel of gedeeltelijk; na afloop van RSF of ISF zijn WP en AP weer gelijk aan elkaar, maar nu wijzen ze naar de eerste vrije plaats achter het laatste frame (WP is met m * (n+3) opgehoogd).

De vertaler kent echter de dimensies van deze arrays, dus de lengte van de storage functions en weet dus, tov. PP van het eigen blok, waar de verschillende storage functions beginnen.

Tot nog toe is er in de stapel alleen ruimte gereserveerd voor de storagefunctions; de (statisch onbekende) hoeveelheid ruimte voor de arrays zelf moet nog gereserveerd worden. Dit nu wordt uitgesteld, totdat alle storage functionsframe's (ook voor value arrays, zie later) gemaakt zijn. Dan wordt een maal voor elk array de adreshebbende operatie

OPC 94 LAP LOCAL ARRAY POSITIONING

uitgevoerd. Deze operatie krijgt mee in het S-register de dynamische (localisatie (32 * positie + bn) van de storage function, waarvan het frame inmiddels is opge-

bouwd, de heersende waarde van AP=WP wordt daarin als "beginadres van het array" ingevuld, waarna deze grootheden met de omvang van het array (Δ_n) verhoogd worden.

Value arrays

Voor de storage function van een value array worden om vertaaltechnische redenen altijd 8 plaatsen ter beschikking gesteld: dit is de achtergrond van de restrictie, dat een value array niet meer dan 5-dimensionaal mag zijn.

Hier moet een storage function frame opgebouwd worden op grond van het in de procedure heading gespecificeerde type en verder op grond van het "aanbod van buiten", dwz. de actuele parameter.

Hiervoor staan ons twee adreshebbende operaties ter beschikking, nl.

OPC 92 RVA REAL VALUE ARRAY STORAGE FUNCTION FRAME
OPC 93 IVA INTEGER VALUE ARRAY STORAGE FUNCTION FRAME

In S krijgen zij als parameter mee de dynamische lokalisering van de PARD in kwestie.

(in de objectprogramma's, die door de vertaler gegenereerd worden, worden RVA en IVA slechts helemaal aan het begin van een blok aangeroepen, dus nog vóór de stapel reserveringen voor de lokale variabelen; dit is niet essentieel. Van essentieel belang is, dat het gebeurd is, voordat door LAP een stuk van statisch onbekende lengte is geïntroduceerd.)

Zodra alle (dus ook de lokale) frame's in de stapel gegenereerd zijn, (en dus eventuele aanroepen LAP kunnen komen) dan moeten we voor alle value array de adreshebbende operatie

OPC 95 VAP VALUE ARRAY POSITIONING

gegeven worden. Evenals LAP krijgt VAP in het S-register de dynamische localisering van de betreffende nog onaffe storage function mee. "POSITIONING"is wat betreft VAP wel een eufemisme, want behalve dat de physische plaats der array elementen nu wordt vastgesteld, wordt bovendien het "buiten array" op het nu locale "binnen array" gecopieerd. Bij deze copiering wordt zonodig de overgang van real naar integer of omgekeerd elementsgewijs bewerkstelligd.

Referentie naar een array element

Referentie naar het array element A[i_0, i_1,....,i_{n-1}] geschiedt door vulling van n+1 opeenvolgende accumulatoren.

De eerste accumulator wordt met een van de operaties TA (OPC 14/18) gevuld met het kopadres van de storage function; tevens bevat deze accumulator in bit 19 van het eerste woord zoals gebruikelijk, of we hier integers of reals bedoelen. De volgende accumulatoren worden in volgorde gevuld met de numerieke waarden der indices i_0, i_1,.... i_{n-1}.
Daarna wordt de operatie IND gegeven (OPC 56).

Deze onderzoekt accumulatoren van de stapel, te beginnen bij de jongst gevulde, net zo lang totdat een accumulator gevonden is met een adres als inhoud. Het aantal indices is dan bekend, eveneens is bekend, waar zich de storage function bevindt, en het adres van het array element wordt dan bepaald. Dit adres vervangt de oudste van de n+1 accumulatoren en de n accumulatoren, die index-waarden bevat hebben, worden vrij gegeven.

```
↷ # A
↷  i0
↷  i1
    :
↷  in-1
   IND
```

is equivalent met ↷ # A[i_0, i_1,...,i_{n-1}]

waarbij met "#" het adres van (STORAGE FUNCTION, resp. variabele) is aangegeven.

Komt de geindiceerde variabele in een expressie voor, dwz. niet links van een := teken, zodat we de numerieke waarde van het array element willen hebben, dan moet de operatie IND door TAR (OPC 58) gevolgd worden.

De for-statement

De macroscopische structuur van

```
        for  v:=  ,  ,  ,   do Σ ;

        goto  Fa  ⇒
Fb:     ↷ # v
        FOR1
        goto  Fc  ⇒
Fa :    2A  0  A
        2B "Fd" A, dwz adres van de FORO
```

```
          ETMP ⇒
              ]                    AAnsluitend aan ETMP
              ]lijstelementen
              ]
          2S  "Fe"  A    adres van het vervolg
           FOR8  ⇒
Fd:        FORO
           goto Fb ⇒
Fc:             ]
                |
                |  STATEMENT
                |
                ]
           goto Fb ⇒
Fe:        .........   voortzetting
```

Dit is de structuur, zoals de vertaler het maakt; je kunt dit wel bekorten, door het stuk volgend op Fb: achter FORO te zetten; dit scheelt drie sprongopdrachten (goto Fa, goto Fc, en de 1ste goto Fb).
De lijstelementen worden aansluitend op ETMP en in volgorde gerepresenteerd door

1) "Arithmetic expression", E_1, door

```
↷ E1
FOR2 ⇒
```

2) "While element", E_2 while B, door

```
↷ E2
  FOR3 ⇒
↷  B
  FOR4 ⇒
```

3) "Step-until-element" , E_3 step E_4 until E_5

```
↷ E3
  FOR5 ⇒
↷ E4
  FOR6 ⇒
↷ E5
  FOR7 ⇒
```

FORO fungeert als speciale blokingang aan het begin van de statement, FOR8 als speciale exit van het blok; FOR8 markeert tevens het einde van de lijst.

Voor het maken van MOP's lijkt dit ternauwernood van interesse: door efficientere realisatie van de for-statement kan je juist verdienen.