K O N I N K L I J K N E D E R L A N D S
M E T E O R O L O G I S C H I N S T I T U U T
D E B I L T

VERSLAGEN
V-220-[redacted]

Speciale Projectgroep Numerieke Voorspelmethoden

Memorandum over Codeprocedures

(Voorlopige Handleiding)

door

Dr. D.J. Bouman

De Bilt, 1969

Kon. Ned. Meteor. Inst.
De Bilt

Inhoud

1. Inleiding
2. Terminologie
3. Verschil met ALGOL-procedures
4. Identifiers
5. Valuelike parameters
6. Macro's
7. Optimized macro's
8. De macro CODE
9. Notaties
10. Procedure ingang
11. Constanten
12. Arithmetische variabelen
13. Boolean variabelen
14. Opstapeling
15. Arithmetische operaties
16. Relationele operaties
17. Boolean operaties
18. Arrays
19. Aanroep van procedures
20. Formele parameters
21. Formele arrays
22. Procedure verlating
23. Labels
24. Conditionele sprongopdrachten
25. Voorbeelden
26. Slotopmerkingen
Appendix

-o-o-o-

KONINKLIJK NEDERLANDS
METEOROLOGISCH INSTITUUT

Verslagen V-220-VI

Speciale Projectgroep Numerieke Voorspelmethoden

Memorandum over Codeprocedures (Voorlopige Handleiding)

door

Dr. D.J. Bouman

1. Inleiding

Het gebruik van procedures waarvan de body in code is gesteld, binnen het kader van een programma in ALGOL-60, is legaal op grond van de volgende passages uit het "REVISED REPORT":
(5.4.1.) "<procedure body>::=<statement>/<code>"
en
(5.4.6.) "Code as procedure body

It is understood that the procedure body may be expressed in non-ALGOL language".
De tot het EL-X8 systeem behorende foutenmeldingen 396 t/m 401 verraden dat de in principe toegestane mogelijkheid inderdaad is geimplementeerd in het systeem.

In dit memorandum wordt een poging gedaan het gebrek aan informatie over het gebruik van code-procedures (CP's) enigermate te ondervangen. Het draagt een voorlopig karakter zoals in de slotparagraaf nader zal worden omschreven.

Als bronnen voor de samenstelling hebben gediend:

1e. F.E.J. KRUSEMAN ARETZ and B.J. MAILLOUX. The ELAN source text of the MC ALGOL 60 system for the EL-X8.
Mathem. Centrum, A'dam. MR.84, 1966.
2e. Een door Prof. Kruseman Aretz beschikbaar gesteld concept voor de eerste 5 hoofdstukken van een publikatie "het objectprogramma, gegenereerd door de X8-ALGOL-60 vertaler van het MC" Het plan tot publikatie van dit document is inmiddels van de baan (persoonlijke mededeling).
3e. Enige voorbeelden van CP's van anderen. (par. 25.2)
4e. De resultaten van een kleine serie experimenten.

Een eerste schets van deze handleiding heeft in handschrift onder een groep van belangstellenden gecirculeerd. Met de opmerkingen van de lezers is in de thans voorliggende versie zoveel mogelijk rekening gehouden.

2. Terminologie

De te gebruiken metalinguistische termen en de daarbij behorende syntaxis worden op de volgende bladzijde gegeven in de bekende BACKUS-notatie.

<procedure body> ::= <statement> / <code> (RR.5.4.1.)
<code> ::= <quote><calling sequence><unquote>
<quote> ::= ⟨
<unquote> ::= ⟩
<calling sequence> ::= <call> / <call> , <calling sequence>
<call> ::= <macronumber> / <macronumber> , <parameter>
<macronumber> ::= <macro> / <optimized macro>
<macro> ::= <unsigned integer ≤ 511>
<optimized macro> ::= <unsigned integer ≥ 512>
<parameter> ::= <namelike> / <valuelike>
<namelike> ::= <identifier>
<valuelike> ::= <integer> / <logical value>
<logical value> ::= <u>true</u> / <u>false</u> (RR.2.2.2.)

In wat gewonere taal:

Er zijn twee soorten macro's: met of zonder parameter.
Macro en parameter worden door een komma gescheiden.
Macro's worden door komma's van elkaar gescheiden.

<u>Voorbeeld</u>:

<u>Prodecure</u> CP; ⟨121,0,84,i,8815872,j,136,50⟩;

<u>Analyse</u>:

⟨	quote
121,0	macro, valuelike (parameter),
84,2	macro, namelike (parameter),
8815872,7,	optimiged macro, namelike,
136,	macro,
50	macro
⟩;	unquote, separator.

<u>Opmerking</u>:

Volgens de ALGOL regels volgt na <unquote> nog de ;<separator> (vgl. R.R. 2.3)

3. <u>Verschil met ALGOL-procedures.</u>

Tijdens het vertaalproces worden in het geval van CP's een aantal bewerkingen die normaal plaats vinden bij de vertaling van een procedure declaratie onderdrukt. Dit geldt in het bijzonder voor de bewerkingen die verband houden met behandeling van de <u>value part</u> en de <u>specification part</u> (R.R. 5.4.1), die hierdoor praktisch tot <u>comment</u> gedegradeerd worden. Zij zouden dus in de declaratie van een CP zonder bezwaar weggelaten kunnen worden.
Het is evenwel voor de leesbaarheid nuttig dit comment toch maar op te nemen in de tekst.

4. <u>Identifiers</u>

Identifiers kunnen optreden als <namelike parameters>.
Slechts die identifiers mogen worden gebruikt, die globaal t.o.v. de procedure zijn gedeclareerd.
<u>Waarschuwing</u>. Uit de vorige paragraaf volgt dat de formele parameters van de procedure niet gebruikt kunnen worden als parameters bij macro's.
Een namelike parameter-alias een identifier verwijst naar een adres.
Men letto goed op het verschil tussen b.v. de volgende macro's:

Macro 92 met parameter i die gelijkwaardig is met A = M [i] en
Macro 93 " " " " " " " A = i.

Bij de uitvoering van de macro 92 wordt de waarde van i in het register A genomen, terwijl macro 93 bewerkt, dat het adres van i in A komt.
1) In par. 10 wordt op dit punt nader ingegaan.

5. Valuelike parameters

Als zodanig kunnen optreden integers en de logical values true en false.
Op een enkele uizondering na worden van de integer valuelike parameters door de vertaler alleen de absolute waarden gebruikt.

6. Macro's

Deze worden aangeduid door middel van een getal $\geqslant 0$ en $\leqslant 511$.
In de praktijk is het getal begrensd door $\leqslant 147$. Elk dezer macro's creeert in het objectprogramma een aantal (meestal 1 en maximaal 4) opdrachten die consecutief worden ontleend aan een lijst van opdrachten de z.g. INSTRUCTION LIST, die in het systeem is opgenomen. In sommige gevallen past de vertaler op een aantal achtereenvolgende macro's een optimalisering toe ter vermindering van het aantal instructies in het object programma. Dit gebeurt volautomatisch zodat de programmeur zich er niet mee behoeft te bemoeien.

7. Optimized macro's

Deze worden aangeduid door een getal dat in de praktijk ligt tussen 8.000.000 en 9.000.000. De vertaler voegt op basis van een analyse van dit getal 1 tot 3 consecutieve opdrachten uit de INSTRUCTION LIST aan het objectprogramma toe. De in de vorige paragraaf genoemde optimalisering verloopt ongezien eveneens via de optimized macro's.
In principe kan de programmeur zelf nieuwe optimized macro's construeren. Dit is echter een arbeidsintensieve bezigheid en in de meeste gevallen zal men met meer gemak van de in de volgende paragraaf te bespreken mogelijkheid gebruik kunnen maken.
In dit memorandum zal daarom slechts eenmaal van een optimized macro gebruik worden gemaakt. (zie 22 slot)

1) "Bij uitvoering van de macro" par abus de language voor "Bij uitvoering van de door de macro in het objectprogramma gegenereerde opdrachten".

8. De macro CODE

De macro met nummer 128 en valuelike parameter (p), meestal symbolisch als CODE(p) aan te duiden, neemt een aparte plaats in.
De vertaler plaatst in dit geval het getal p in de instructies van het objectprogramma. Bij uitvoering van het objectprogramma in de executiefase wordt derhalve die opdracht uitgevoerd waarvan de interne representatie door het getal p wordt weergegeven.

Voorbeeld

De macro + parameter 128, 524290, (of code (524290))
heeft het volgende effect:
Daar 524290 = '2 000 002' = $10000\ 00000\ 00000\ 00010_2$
wordt uitgevoerd de opdracht
A+:M[2] ofwel A+2 (zie ELAN - handleiding).

Met behulp van de macro CODE kunnen dus alle wettige EL-X8 opdrachten aan het objectprogramma worden toegevoegd.
Om hiervan gebruik te kunnen maken is kennis nodig van de interne representatie van opdrachten. De appendix bevat een tabel waarin de benodigde gegevens zijn opgenomen.
De macro CODE is ook nog voor een ander doel bruikbaar.
Men kan een voldoende aantal malen de macro 128,0, zodanig plaatsen dat zij nooit wordt uitgevoerd. (b.v. na de opdracht tot procedure verlating of in een traject waar overheen wordt gesprongen).
Daardoor wordt aan de CP een eigen werkruimte meegegeven, die men o.a. doormiddel van MT - adressering kan bereiken.
De macro CODE is in de praktijk de enige waarbij een negatieve (valuelike) parameter kan worden gebruikt.
Door een van de lezers van de eerste versie van deze voorlopige handleiding is opgemerkt dat het construeren van opdrachten in de interne representatie neerkomt op het werken met een "eerste generatie methode voor een derde generatie machine", althans bij de thans in gebruik zijnde ALGOL-compiler en bedrijfsysteem.

SED CONTRA:

(1). Het gaat hier om een uitzondering op een uitzondering op de normale gang van zaken. Normaal is immers ALGOL.
Codeprocedures zijn al uitzonderingen. Binnen codeprocedures zijn de opdrachten normaal in de vorm van macro's ≠128
(2). Codeprocedures worden in 2 gevallen gebruikt:
(a) Als het niet anders kan (Available, RANDOM, SETRANDOM, logical-sum, logical product etc.). Dit zijn alle zeer korte procedures.
(b) Als men er zoveel voordeel (b.v. tijdwinst) van verwacht, dat men het extra werk er voor over heeft.
(3). Het alternatief is:
Nog een extra macro b.v. no. 1000 invoeren met als parameter een string van symbolen voorstellende een in ELAN geschreven opdracht. Dan moet men echter over een ELAN compiler beschikken.
(4). De gewraakte constructie blijkt in de praktijk mee te vallen. Door gebruik te maken van oktale getallen kan de constructie nog vereenvoudigd worden. (zie Appendix).

9. Notaties

In het vervolg zullen de volgende notaties worden gebezigd:
Een identifier eindigende op vl zal een valuelike parameter aanduiden.
Een identifier " " nl " " namelike " " .
Aan elke macro wordt tevens een symbolische naam toegekend, b.v. CODE. NAAM (p) zal dan weer geven de macro met naam NAAM en parameter p.

Verder zal dikwijls een tabellarische rangschikking volgens onderstaand model worden gebruikt:

(1)	(2)	(3)	(4)	(5)
10	128,524290,	CODE(524290)	A + 2	↓t,a,x',x",k,↑

De kolommen vermelden respectievelijk:

(1) Rangnummer van de opdracht in de CP (facultatief)
(2) Macro met eventuele parameter
(3) Symbolische aanduiding
(4) ELAN equivalent
(5) Commentaar
} (in latere voorbeelden soms in één kolom ondergebracht)

De kolom (2) bevat dus de tekst van de CP, de rest komt in de eigenlijke programmatekst niet voor.

In kolom (5) zal dikwijls het stapelbeeld worden weergegeven. Hiervoor worden nog de volgende symbolen gebruikt:

↓: posities beneden het laagste woord in de stapel waarvoor interesse bestaat.
x',x": kopwoord respectievelijk staartwoord van een real.
↑: Eerste vrije plaats in de stapel.

Conventie

Voorzover in kolom (4) (ELAN - equivalent) identifiers worden gebruikt die op namelike parameters betrekking hebben moeten deze worden gelezen als adressen.
Als voorbeeld de 2 macro's besproken aan het slot van paragraaf 4.

1	92,i,	TAK (i)	A = M[i]	i = namelike
2	93,i,	TAA (i)	A = i	

De beschreven conventie wekt wellicht enige bevreemding, omdat op het eerste gezicht een andere conventie natuurlijker schijnt te zijn, nl. die waarbij een ALGOL - identifier binnen een ELAN - tekst als een waarde wordt opgevat. De laatste conventie is echter niet vol te houden, omdat een array of een procedure geen waarden heeft.
In de hier gebruikte conventie kan men ook aan procedures en arrays op ongedwongen wijze een adres toekennen t.w. het beginadres van de procedure en het adres waar de later te bespreken arraysleutel te vinden is, waaruit volgt dat alleen met de boven gedefinieerde conventie in alle gevallen zinvol en moeiteloos gewerkt kan worden

10. Procedure ingang

De CP dient te worden geplaatst in het buitenste blok. De eerste opdracht moet steeds luiden:

0	121,0	TDA(0)	S = 0

11. Constanten

Positieve integerconstanten <32767 en logische constanten kunnen in de CP worden geintroduceerd doormiddel van de volgende macro's:

87,ivl,	TSIC(ivl)	F = ivl	Take Small Integer Constant ivl = valuelike ;$+0 \leq$ ivl<32767
89,bvl,	TBC (bvl)	S = bvl,	bvl = true of false; Take Boolean Constant

Normaliter plaatst ook de vertaler bij een gewone ALGOL-procedure zulke constanten op dezelfde wijze in het objectprogramma.
Integerconstanten die niet aan de genoemde condities voldoen en real constanten worden normaal in een aparte constantenlijst geplaatst. De adressering hiervan is statisch waardoor de constantenlijst voor de programmeurs van een CP ontoegankelijk is.
Om dergelijke constanten bereikbaar te maken zijn er twee mogelijkheden:

(1) Binnen de ALGOL tekst de constanten assigneren aan een globaal gedeclareerde variabele.
(2) De constanten opnemen in de eigen werkruimte van de CP.
(Voorbeeld in paragraaf 25.2)

12. Arithmetische variabelen

Voor het ophalen van een bereikbare arithmetische variabele staan de volgende macro's ter beschikking:

84,inl,	TIV(inl)	G = M[inl]	Take Integer Variable
85,rnl,	TRV(rnl)	F = M[rnl]	Take Real Variable
92,inl,	TAK(inl)	A = M[inl]	Take Array Key (zie par 18.4)

Voor de assignatie van een in het register F (G) aanwezige waarde aan een variabele staan de macro's 94, t/m 96 ter beschikking.

1	94,rnl,	STR(rnl)	M[rnl] = F	Store Real
2	95,inl,	STI(inl)	S = F, ~~G~~	al integer? Store Integer
3 opdrachten! 3			N,SUBC(;RND)	Neen, dan afronden
4			M[inl] = G	en assigneren
5	96,inl	SSTI(inl)	M[inl] = G	assigneren zonder af te ronden.

De macro 96 is alleen bruikbaar als men er zeker van is, dat wat in F(G) staat een integer is die binnen de integercapaciteit valt.

De tot het systeem behorende macro

86,ivl,	TIC(ivl)	G = M[ivl]	Take Integer Constant

met value-like parameter zal meestal niet gebruikt kunnen worden, omdat het benodigde statische adres niet bekend is. Zij is evenwel nuttig voor transport naar het register G b.v.

86,59 of TIC(59) komt overeen met G = M[59] en

daar A = M[59] levert dit dus G = A

13. Boolean variabelen

Het ophalen van en assigneren aan booleans wordt verzorgd door de macro's (met resultaat in CON en de daarmede corresponderende bits van T).

88,bnl	TBV(bnl)	S = M[bnl] ,P	Take Boolean Variable
97,bnl	STB(bnl)	S = T M[bnl] = S	2 opdrachten! Store Boolean

14. Opstapeling

Voor het op de stapel zetten kunnen de volgende macro's worden gebruikt.

0,	STACK	MC = F	
14,	STAB	S = T MC = S	2 opdrachten. Stack boolean
20,	STAA	MC = A	STACK A

Ontstapeling geschiedt door het uitvoeren van een binaire operatie. Zij deze aΩb (b.v. a*b). Na het opstapel zetten van a wordt de operatie Ω uitgevoerd tussen de opgestapelde a en de in een register geplaatste b onder aflaging van de stapel. Voorbeelden in overvloed in de volgende paragrafen.

15. Arithmetische operaties

Als regel vinden deze plaats via de stapel en het F(G) register. Beschikbaar zijn de volgende macro's, die met uitzondering van 38, 110 en 115 het resultaat van de operatie afleveren in F(G). De stapel wordt daarbij afgelaagd.

1,	NEG,	F = -F		
2,	ADD,	F + MC[-2]		stack + F
3,	SUB,	F = -F	2 opdrachten,	stack - F
		F + MC[-2]		
4,	MUL,	F x MC[-2]	,	stack * F
5,	DIV,	MC = F	3 opdrachten,	stack / F
		F = MC[-4]		
		F / MC		
38,	DECS,	S - 2		
76,	ABS,	F = F,P	2 opdrachten	
		N, F = -F		
77,	SIGN,	F = F, Z	3 opdrachten	
		N, F = 1,E,		
		N, F = -1		
110,ivl,	DECB(ivl)	B - ivl		
115,ivl,	INCRB(ivl)	B + ivl		
6,	IDI	SUBC(:IDI)		stack + G
7,	TTP	SUBC(:TTP)		stack ↑ F

16. Relationele operaties

Deze vinden als regel eveneens plaats tussen een opgestapelde grootheid en een in een register genomen grootheid. De aflevering geschiedt in CON en de daarmee corresponderende bits van T onder aflaging van de stapel.

8,	EQU,	F - MC[-2],Z		stack = F?
9,	UQU,	F - MC[-2],Z	2 opdrachten,	stack ≠ F?
		S = -T,P		
10,	LESS,	F - MC[-2],P	2 opdrachten,	stack < F?
		Y, F - 0, P		
11,	MST	F - MC[-2],P	2 opdrachten,	stack ≤ F?
		N, F = F,Z		
12,	MOR,	F - MC[-2],P	3 opdrachten,	stack > F?
		N, F = F,Z		
		S = -T		
13,	LST,	F - MC[-2],Z	2 opdrachten,	stack ≥ F?
		N, S = -1,E		
51,	TEST1,	F - MC[-2],Z,		als 8
52,	TEST2,	N, F = F,E	2 opdrachten,	
		N, F = F,Z		

17. Boolean operaties

15,	NON,	S = -T,P	CON: = ¬CON
16,	QVL,	S = T,P S = MC[-1],E	2 opdrachten, stack ≡ CON?
17,	IMP,	Y, B - 1; N, S = -MC[-1],P	2 opdrachten, stack ⇒ CON?
18,	OR,	Y, B - 1; N, S = MC[-1],P	2 opdrachten, stack ∨ CON?
19,	AND,	N, B - 1 Y, S = MC[-1],P	2 opdrachten, stack ∧ CON?

Aflevering in CON en de daarmede corresponderende bits van T, onder afslaging van de stapel.

18. Arrays

18.1. Mapping function

Zij a een n- dimensionaal array (integer of real array)[1)]

a[l1 : u1, l2 : u2, ... ln : un];

Defineer:

delta[0] = if integer array then 1 else 2,

delta[1] = (u1 - l1 + 1) * delta[0]

delta[2] = (u2 - l2 + 1) * delta[1]

delta[n - 1] = (u(n - 1) - l(n - 1) + 1) * delta[n - 2]

Zij vervolgens:

:a [s1,....sn] het adres van het vaste arrayelement a[s1,....sn] en

:a [i1,....in] het adres van het lopende arrayelement a[i1,....in].

De mapping function die aan elk arrayelement een adres toewijst is dan:

$$:a[i1,....in] = :a[s1,....sn] + \sum_{k=1}^{n} delta[k-1] * [ik - sk] \quad (MF1)$$

Hieruit volgt: Kennis van 1 adres is voldoende om alle andere te kunnen berekenen. In beginsel kunnen 3 verschillende technieken worden gevolgd:

1. Telkens als een array-adres nodig is wordt het met behulp van de beschikbare macro's berekend. (te noemen Lazy Man).
2. Eénmaal wordt een adres berekend. Dit wordt ergens veilig opgeborgen. Voor volgende adressen wordt van formule (MF1) gebruik gemaakt.
3. De grootheden a:[s1,....sn] , delta[0 : n - 1] worden ontleend aan de later te behandelen referentievector waarna (MF1) kan worden toegepast.

De drie methodes zullen in de volgende paragrafen nader worden uitgewerkt.

1) Boolean arrays blijven buiten beschouwing.

18.2. Lazy Man

Voor de behandeling van een array element moeten in deze methode, waarin de gang van zaken zoals die normaliter door de vertaler wordt gevolgd vrijwel ongewijzigd wordt overgenomen, de volgende macro's worden ingelast in de CP:

Opstapeling (als real) van de eerste index.
Opstapeling (" ") " " tweede " .

Opstapeling (" ") " " voorlaatste index.
Laatste index in F(G).

```
92,arrayname,     TAK (arrayname)     A = M[arrayname]

     21,              TSR,                SUB(:IND)        Take Subscripted Real
                                          F = MA           (alleen voor real ar-
                                                           rays)(2 opdrachten)
     22,              TSI,                SUB(:IND)        Take Subcripted
                                                                    Integer
                                          G = MA           (alleen voor
                                                            integer arrays)
                                                           (2 opdracht)
```

Het effect hiervan in de executie fase is:
1. De stapel (van de indices) wordt weer afgebroken.
2. In A komt het adres van het array element.
3. In F(G) komt de waarde van het array element.

Voorbeeld:

Gegeven zijn de globaal gedeclareerde integers i,j en w en een globaal gedeclareerd (3-dimensionaal) integer array piet
Te coderen:

```
w: = piet[1,i,j];

   87,1,     TSIC(1)      F = 1              zie no: 11.
    0,       STACK        MC = F                     14.
   84,i,     TIV(i)       G = M[i]                   12.
    0,       STACK        MC = F                     14.
   84,j,     TIV(j)       G = M[j]                   12.
92,piet,     TAK(piet)    A = M[piet]          12, 18.2.
   22,       TS I         SUB(:IND)                18.2.
                          G = MA
   95,w,     STI(w)       S = F,Z                    12.
                          N,SUBC(:RND)
                          M[w] = G
```

De macro's TSI en TSR verzorgen via de tot het complex behorende subroutine IND o.a. de controle op het lezen en schrijven buiten het array en zijn daardoor tijdrovend. In het algemeen zal deze methode, die voor elk voorkomend array element opnieuw moet worden toegepast, geen of weinig tijdwinst opleveren daar de normale gang van zaken op de voet moet worden gevolgd.

18.3 Gebruik van MF

Hierbij wordt dus eenmaal de methode van paragraaf 18,2. gevolgd. Daarna gebruikt men (MF1) of past hierop gebaseerde bewerkingen toe. Hierbij kan met succes gebruik gemaakt worden van alle toevallige omstandigheden die zich voordoen.

Voorbeeld

u een 1 - dimensionaal integer array
te coderen j: = u[10] + u[11] + u[12];

```
        87,11,      TSIC(11)            F = 11
        92,u,       TAK(u)              A = M[u]
        22,         TSI                 SUB(:IND)
                                        G = MA
128,-31918334,      CODE(-31918334)     G + MA[1]
128,-31918336,      CODE(-31918336)     G + MA[-1]
        95,j,       STI(j)
```

18.4 Referentievector

Na uitvoering van de macro TAK(arrayname) (ook zonder voorafgaande manipulaties met indices) is in A een getal verschenen dat arraykey of arraysleutel of kortweg key wordt genoemd.
Het getal is gelijk aan het (statisch) kopadres van de integer referentievector.

M[A] = ref[0] of A = : ref[0].

Het aantal elementen van de referentievector is gelijk aan

4 + 3 * arraydimensie

Zij zijn achterste voren opgeborgen in het geheugen. Hieronder volgt een tabel waarin de betekenis van de afzonderlijke elementen is weergegeven voor een 1, 2 en een 3-dimensionaal array. Zoals uit de tabel blijkt, werd door de auteurs van de vertaler voor de grootheid :a[s1,....sn] uit(MF1) gekozen het adres van het laatste element van het array.

dim = 1	dim = 2	dim = 3	
ref[0]	ref[0]	ref[0]	adres van het laatste element
ref[-1]	ref[-1]	ref[-1]	type (5=integer, 8=real, 6=boolean)
ref[-2]	ref[-2]	ref[-2]	dimensie
		ref[-3]	delta[3] = (u3-l3+1)*delta[2] aantal woorden.
		ref[-4]	u3
		ref[-5]	l3 - 1
	ref[-3]	ref[-6]	delta[2] = (u2-l2+1)*delta[1]
	ref[-4]	ref[-7]	u2
	ref[-5]	ref[-8]	l2 -1

ref[-3]	ref[-6]	ref[-9]	delta[1]=(u1-l1+1)*delta[0]
ref[-4]	ref[-7]	ref[-10]	u1
ref[-5]	ref[-8]	ref[-11]	l1 - 1
ref[-6]	ref[-9]	ref[-12]	if real then -0 else +0

Voor elke dimensie is ref[-3] gelijk aan het aantal woorden. Het gebruik van de in de referentievector opgeslagen gegevens wordt toegelicht aan de hand van het volgende voorbeeld. Gevraagd de waarde van jan[i,j] (integer array)

92,jan,	TAK(jan)	A = M[jan]	A=key van jan=ref[0]
84,i,	TIV(i)	G = M[0]	
128,20971578,	CODE(20971578)	S = G	transport naar S
84,j,	TIV(j)	G = M[i]	
128,-29821187,	CODE	G - MA[-4]	(j - u2)
128,-1514119,	CODE	G * MA[-6]	(j-u2) * delta[1]
128,20477689,	CODE	S - MA[-7]	(i-u1)
128,-31918335,	CODE	G + MA	laatste adres + (j-u2) * delta[1]
128,33521603,	CODE	G + S	+(i-u1)=adres jan[i,j]
128,-27724543	CODE	G = MG	en waarde jan[i,j].

Indien voor het uitvoeren van de laatste opdracht het gegeven op een geschikte plaats wordt weggeschreven kan dit later weer worden gebruikt zonder dat opnieuw een adresberekening nodig is. Het adres is dan bovendien beschikbaar voor manipulaties als beschreven in de vorige paragraaf. Door combinatie van de methodes beschreven in de beide laatste paragrafen kan men, als men alle toevallige omstandigheden uitbuit, dikwijls aanzienlijke tijdwinst behalen t.o.v. een ALGOL-procedure.[1]
N.B. In de hierboven weergegeven vorm van adresberekening vindt geen controle op lezen/schrijven buiten het array plaats.

18.5 Assignatie

Indien het adres van het arrayelement waaraan geassigneerd moet worden bekend is, of volgens het voorafgaande eenvoudig te berekenen is, verloopt de assignatie volgens (met adres in A)

	128,-10946815,	CODE	MA = G
of	128,-10979583,	CODE	MA = F

De vertaler (en dus de Lazy Man) werkt volgens onderstaand schema.
1. Alle indicis (nu ook de laatste) op stapel
2. 92, arrayname alias TAK (arrayname)
3. 20, |STAA| MC = A
4. Bereken de te assigneren waarde
5. Een van de macro's

1) Zie ook: T.J. Dekker, ALGOL60 Procedures in numerical Analysis, Part 1. Mathem. centre Tracts 22, A'dam 1968, waar voorbeelden van zevenvoudige versnelling worden vermeld.

29,	STSR	SUB2(:STSR)	if real array Store Subscripted Real
30,	STSI	SUB2(:STSI)	if integer array Store " Integer

De assignatie wordt uitgevoerd, zonodig na afronding tot integer, onder afbraak van de stapel.

19. Aanroep van procedures

Hierbij moet een onderscheid gemaakt worden tussen een aantal gevallen.

19.1 Systeem procedures met macronummers

De volgende tot het systeem behorende procedures worden aangeroepen via een macronummer. (ook normaliter!)

78,	ENTIER	130,	COS	135,	FLOP	140,	REHEP
79,	SQRT	131,	ARCTAN	136,	PUNCH	141,	PUHEP
80,	EXP	132,	READ	137,	PUNLCR	142,	HAND
81,	LN	133,	FIXP	138,	PUSPACE	143,	XEEN
129,	SIN	134,	ABSFIXP	139,	RUNOUT	144,	STOP

Zij komen geheel overeen met die welke in de handleidingen voor het EL-X8-ALGOL-systeem staan. De volgende regels moeten in acht worden genomen.

1. Voor de aanroep via het macronummer moet de laatste (c.q.de enige) parameter in F(G) staan.
2. Eventuele andere parameters moeten real op stapel staan, de eerste onderop.
3. De uitvoering van de opdracht breekt de stapel weer af.
4. procedures die een waarde afleveren doen dit in F(G).

Voorbeeld:

```
      ABSFIXP (5,2,x);
87,5,    TSIC(5)     F = 5
 0,      STACK       MC = F
87,2,    TSIC(2)     F = 2
 0,      STACK       MC = F
83,x,    TRV(x)      F = M[x]
 134,    ABSFIXP     SUBC(:ABSFIXP)
```

De in paragraaf 15 genoemde macro's 76(ABS) en 77(SIGN) zijn geen echte subroutinecalls; zij worden uitgewerkt in het objectprogramma door de vertaler geplaatst.

19.2 Gedeclareerde procedures

Het aanroepen binnen een CP van een andere gedeclareerde procedure (die zelf ook weer een CP mag zijn) geschiedt doormiddel van

108,procedurenaam, | SUBJ(procedurenaam) | SUBC(:M[procedurenaam])

Zonder verdere complicaties gaat dit goed voor procedures die geen parameters hebben. Voor procedures met parameters is i.h.a. de aanroep te gecompliceerd om hier besproken te kunnen worden.

19.3 Systeemprocedures zonder macronummer

De niet onder 19.1 opgenomen systeemprocedures worden behandeld als onder 19.2 beschreven.

20. Formele parameters

Reeds in paragraaf 4 is vermeld dat formele parameters van een procedure niet als namelikeparameters van een macro kunnen optreden. Dat hangt samen met het feit dat in geval van een CP de vertaler geen verband organiseert tussen de formele parameters en de bij de call optredende actuele parameters.
Het schema waarmede een van formele parameters voorzien CP moet worden behandeld is alsvolgt. (Voor formele arrays zie volgende par.)

```
Na          ↓121,0,                    TDA(0)     S = 0
              128,13138688,            A = :MC
```

moet in volgorde van de actual parameter list voor elke parameter een van de volgende macro's volgen:

```
53,   CRV    SUBC(:CRV)      neven effect B:=B+2   Call real Var
54,   CIV    SUBC(:CIV)                 B:=B+1     Call Integer Var
55,   CBV    SUBC(:CBV)                 B:=B+1     Call Boolean Var

58,   CEN    SUB (:CEN)                 B:=B+2     Call Expression
59,   CLPN   SUB1(:CLPN)                B:=B+4                by name
```

De keuze geschiedt op grond van onderstaande criteria:
1. voor een call by value van een real, integer of boolean resp.: CRV, CIV, CBV
2. voor een call by name van een real, integer of boolean waaraan niet tijdens de CP geassigneerd wordt CEN
3. in andere gevallen CLPN

De macro's CRV, CIV en CBV hebben bij executie tot resultaat dat de waarde van de parameter op de stapel wordt geplaatst.
De uitvoering van CEN heeft tot gevolg dat op de stapel 2 instructies worden geplaast APIC[0:1]
De uitvoering van CLPN heeft tot gevolg dat op de stapel 4 instructies worden geplaatst APIC[0:3].
Voorbeeld:
Zij klaas (i,j,x) een procedure waarbinnen alleen aan j geassigneerd wordt. De eerste macro's van klaas zijn dan:

```
↓121,0,           TDA(0)    S = o          stapelbeeld:
128,13138688,     CODE      A = :MC
 54,              CIV       SUBC(:CIV)     ↓,i,↑
 59,              CLPN      SUB1(:CLPN)    ↓,i,APIC[0],APIC[1],APIC[2],APIC[3],↑
 53,              CRV       SUBC(:CRV)     ↓,i,APIC[0],APIC[1],APIC[2],APIC[3],
                                                              x',x",↑
```

Voor de manipulaties met de APIC's gelden de volgende regels:
1. de uitvoering van de APIC-instructies moet gebeuren via de executieve DOS-opdracht.
2. Het ophalen van een met een formele parameter corresponderende actuele gebeurt door uitvoering van APIC[0] (dus DOS(APIC 0))
3. Het assigneren aan een met een formele parameter corresponderende actuele gaat in 3 stappen:

3.1. Uitvoering van APIC[2]
3.2. Berekening van de te assigneren waarde (resultaat in F(G))
3.3. Uitvoering van APIC[3]

4. Uitvoering van de APIC's breekt de stapel niet af.

Voor het doen uitvoeren van de APIC's heeft men nodig:

128,-34193407 + n | CODE (-34193407 + n) | DOS(M[B + n])

In het bovenstaande voorbeeld moet dus voor assignatie aan j allereerst
128,-34193411, | DOS(M[B-4]) | DOS(APIC[2])
en na berekening van de te assigneren waarde:
128.-34193410, | DOS(M[B-3]) | DOS(APIC[3])
worden gebruikt.
Nog 2 opmerkingen:
1. Het uitvoeren van de macro's 53, 54, 55, 58 of 59, in de goede volgorde is verplicht, ook al zou van de op de stapel gezette instructies of waarden geen gebruik worden gemaakt.

2. Voor het verlaten van de CP moet de stapel worden afgebroken b.v. met:
110,n, | DEC B(n) | B - n | (zie paragraaf 15)
In beide gevallen is de straf voor nalatigheid, dat na het verlaten van de CP een sprong naar een verkeerd adres plaats vindt.

Toelichting:
In het objectprogramma volgen na de procedure call een aantal instructies (APD's = actual parameter descriptor) die gegevens bevatten over de actuele parameters. Door tussenkomst van A = :MC (zie boven) wordt de eerste hiervan aangewezen (parameterpointer). Het adres waarnaar verwezen wordt, wordt door de call's van CIV, CRV, CEN en CLPN telkens met 1 verhoogd en is dus na alle calls automatisch het adres waar het programma na procedureverlating moet worden hervat.

21. Formele arrays (geen value arrays)

Bij het opstellen van de reeks CEN etc voor de formele parameters moet ingeval van een formeel array steeds CEN worden gekozen. (Aan een array wordt nooit geassigneerd, dit gebeurt aan de array elementen). Door DOS(APIC[0]) wordt de arraysleutel in A gebracht. In de meeste gevallen heeft men hieraan genoeg om verder te werken zoals beschreven in par. 18. Bovendien kan nog van de volgende macro's gebruik worden gemaakt

25, | TFSU | SUB2(:TFSU)

die zowel TSR als TSI (zie par. 18.2) vervangt en van

34, | STFSU | SUB2(:STFSU) die zowel STSR als STSI (zie par. 18.5) vervangt in geval van formele arrays.

22. Procedure verlating

Het verlaten van de CP kan bewerkstelligd worden door:

50,	EXIT	GOTOR (MC[-1])
128,-44336384,	CODE	Y, GOTOR (MC[-1])
128,-44303616,	CODE	N, GOTOR (MC[-1])

uiteraard zonodig voorafgegaan door:
110,nvl | DECB(nvl) | B - nvl (zie par. 12)

Voor een type procedure is het niet nodig expliciet aan de procedurenaam te assigneren. Voldoende is dat op moment van verlating de te assigneren waarde in F(G) is.
Voorbeeld real procedure: RR; RR: = sqrt (READ + READ);

In code:

```
real procedure RR;
        121,0,      TDA(0)      S = 0
        132,        READ        SUBC(:READ)
          0,        STACK       MC = F
        132,        READ        SUBC(:READ)
          2,        ADD         F + MC[-2]
         79,        SQRT        SUBC(:SQRT)
         50);       EXIT        GOTOR (MC[-1])
```

In de ALGOL-tekst kan dan zonder meer een statement als

```
y: = 10+ RR;
```

worden opgenomen.
Moet een integer worden afgeleverd dan is een afronding nodig.
Dit kan met:

```
      8987136,    optimized    S = F,Z
                               N, SUBC(:RND)
```

zijnde de beloofde enige geoptimaliseerde macro's in dit geschrift (zie par. 7 slot).

23. Labels

Een sprong naar een label van het ALGOL-programma (mits volgens de regels geoorloofd) kan aldus worden bereikt.

91, labelname,	TLV(labelname)	A= labelname	(vertaler zorgt zonodig zelf voor dynamisch adres)
74,	JUA	GOTO(:JUA)	jump to A

De subroutine uit het complex JUA verzorgt zonodig alle bewerkingen behorende bij procedureverlating, blokverlating enz.
Het is niet mogelijk labels in een CP aan te brengen. Hoe men dit bezwaar kan ondervangen wordt in de volgende paragraaf behandeld.

24. Conditionele en sprongopdrachten

Vanwege de nauwe samenhang worden deze twee typen van opdrachten gezamenlijk behandeld. Twee verschillende methoden komen in aanmerking.

24.1 Gebruik van maskers

Declareer de globale intigers ju, coju en ycoju en assigneer hieraan voordat de CP wordt aangeroepen:

```
ju: = -45613055;
coju: = -45514751;
ycoju: = -45547519;
```

De geassigneerde waarden zijn de interne representaties van de opdrachten (In ELAN).

```
GOTO(:M[0])       "JU
N,GOTO(:M[0])     "COJU
Y,GOTO(:M[0])     "YCOJU
```

Met behulp van de maskers ju, coju en ycoju kan men nu sprongen organiseren zoals uit onderstaand voorbeeld blijkt:

Voorbeeld

```
if x = 0 then i: = j;
```

```
        83,x,          TRV(x)        F = M[x]
         2,            STACK)        MC = F
        87,0,          TSIC(0)       F = 0
         8,            EQU           F - MC[-2],Z
-2      92,coju,       TAK(coju)     A = M[coju]
-1      128,8944899,   CODE          A + :MT[3]
 0      111,59,        DO(59)        DO(M[59]) = DO(A)
 1      84,i,          TIV(i)        G = M[i]
 2      96,j           SSTI(j)       M[=j] = G
 3
```

Bij opdracht (-2) krijgt A de waarde coju en stelt dus N,GOTO(M[0]) voor. In (-1) wordt hierbij opgeteld :MT[3] dat wil dus zeggen het statisch adres van de opdracht (3). Opdracht (0) is dan de conditionele opdracht naar dit adres te springen.
Hoewel dit systeem werkt, kleven er toch enige bezwaren aan. In de eerste plaats vereist het nogal wat werk. Bij lange sprongen is het gevaar van vertellen aanwezig. Bovendien moet rekening gehouden worden met eventueel door de vertaler aan te brengen optimalisaties. De in de volgende subparagraaf te bespreken methode werkt eenvoudiger.

24.2 Procedure namen als labels

In de hier te bespreken methode wordt gebruik gemaakt van de macro:

```
103,naam,  JU1(naam)  A = naam          2 opdrachten: JUmp
                      Goto(:MA[1] )
```

Door voor "naam" de naam van een procedure te kiezen vindt een sprong plaats naar procedurenaam [1] d.w.z. naar de opdracht volgende op TDA(0) (S=0) en wel - zeer essentieel - zonder dat de stapel wordt gewijzigd en i.h.b. zonder dat hieraan een nieuwe link wordt toegevoegd of de oude link wordt overschreven. De werkwijze wordt eerst in ELAN beschreven.
Zij te coderen:

```
if x = 0 then i: = j else m: = n;
```

De normale codering in ELAN zou alsvolgt kunnen verlopen:

```
        G = M[x],Z
        N, GOTO(:L1)
        G = M[j]
        M[i] = G
        GOTO(:L2)
L1:     G = M[n]
        M[m] = G
L2:
```

Dit kan vervangen worden door:

```
        SUBC(:TEST)
LO:     G = M[n]              "ADRES: Lo
        M[m] = G
        GOTO(:L2)
```

```
TEST:  G = M[x],z
       N,GOTOR(MC[-1])        "TERUG NAAR ADRES: Lo
       B - 1                  "VERWIJDER LINK"
       G = M[j]
       M[i] = G
  L2:
```

Nog een stap verder dan is onder gebruikmaking van de aangegeven macro JU1, de codering als CP gemakkelijk uit te voeren:

```
procedure P; ⊀121,0      TDA(0)        S = 0
    :
   100      108,TEST,    SUBJ(TEST)    SUBC(:TEST)
   101       84,n,       TIV(n)        G = M[n]
             96,m,       SSTI(m)       M[m] = G
         103,vervolg⊁;   JU1(VERVOLG)  A = vervolg
                                       GOTO(:MA[1])     GOTO VERVOLG[1]

procedure TEST; ⊀121,0,  TDA(0)        S = 0
                 84,x,   TIV(x)        G = M[x]
                  0,     STACK         MC = F
                 87,0,   TSIC(0)       F = 0
                  8,     EQU           F-MC[-2],z
          128,-44303616, CODE          N,GOTOR(MC[-1]) terug naar P[101]
                110,1    DECB(1)       B - 1           weg met link
                                                            naar P
                 84,i,   TIV(i)        G = M[i]
                 96,j,   SSTI(j)       M[j] = G
          103,VERVOLG ⊁; JU1(VERVOLG)  A = VERVOLG
                                       GOTO(= MA[1])    goto VERVOLG[1]

procedure VERVOLG;
                ⊀121,0,  TDA(0)        S = 0
```

N.B. In VERVOLG heeft de opdracht GOTOR(MC[-1]) nog steeds hetzelfde effect als ze in P had, m.a.w. aan het terugkeeradres in het programma is niets veranderd. (De bewaarnemer moet dezelfde zaak welke hij ontvangen heeft teruggeven. B.W.art. 1751 lid 1).
Bij analyse blijkt dat deze techniek neerkomt op het volgende:

1. Alle labels worden procedurenamen.
2. Testen op conditie in een subroutine. Bij een der condities terug naar de calling procedure, bij de andere blijven in de subroutine, daar de link verwijderen. Beide takken verenigen zich weer door een sprong binnen een volgende procedure waarvan de naam de functie van een label heeft.

Deze methode is bijzonder geschikt voor het organiseren van cycli zoals blijkt uit het voorbeeld dat gelijkwaardig beoogt te zijn aan de ALGOL-procedure:

```
procedure P; begin integer i;
                    fori: = 10 step-1 until 1 do  PUNCH(READ)
                   end P;
```

In code:

```
procedure P;
        ⊀121,0,        TDA(0)        S = 0
          87,10,       TSIC(10)      F = 10 } initialiseer i
           2,          STACK         MC = F }
         103, PP⊁;     JU1(PP)       A = PP; GOTO(:MA[1])
                                              einde P;

procedure PP;
        ⊀121,0,       TDA(0)         S = 0
         108,PPP,     SUBJ(PPP)      SUBC(:PPP)
         132,         READ
         136,         PUNCH
          87,1        TSIC(1)        F = 1
           3,         SUB            F = -F, F + MC[-2]
           0,         STACK          MC = F
         103,PP⊁;     JU1(PP)        A=PP; GOTO(:MA[1])
                                          terug PP[1]

procedure PPP;
        ⊀121,0                TDA(0)
         128,-28262401,       CODE          G = M[B-2] dwzG = M[i]
           0,                 STACK         MC = F
          87,0                TSIC(0)       F = 0
           8,                 EQU           i = 0?
         128,-44303616,       CODE          N, terug naar PP[2]
         110,1                DECB(1)       Verwijder link naar PP
         50  ;                GOTOR(MC[-1]) Wel te verstaan naar het
                                            terugkeeradres dat bij
                                            de call van P werd ge-
                                                        steld.
```

Enkele experimenten toonden aan dat dit op het eerste gezicht nogal vreemde gebruik van procedurenamen als labels soepel werkt.

25. Voorbeelden

25.1. AVAILABLE

```
integer procedure available;
        ⊀121,0,              TDA(0)          S = 0
          87,16363,          TSIC(16363)     F = 16363
         128,-31424450,      CODE            G-M[61]≡G - B
          50⊁;               EXIT            GOTOR(MC[-2])
```

comment In 16363 is een veiligheidsmarge verborgen.
De afgeleverde waarde houdt geen rekening met het eventuele gebruik van een contrastapel. Een oudere versie, tweemaal zo lang, is eerder reeds aan geinteresseerden medegedeeld;

25.2. RANDOM en SETRANDOM[1)]

```
procedure SETRANDOM(x); value x; real x;

                 ⊀121,0              S = 0
                  128,13138688,      A = :MC
                   53,               CRV                 ↓x',x",↑
                   87,0,             TSIC(0)              F = 0
                    2,               ADD                  F + MC[-2]          F = M[x]
                   76,               ABS                                      voor veilig-
                   93,RANDOM         TAA(RANDOM)          A = RANDOM                heid
                  128,-15173878,     F× MA[9]                                 RANDOM[9] dus
                   89 87136,                               S = F,Z;N,(SUBC:RND) zie par. 22 slot
                  128,-10946804,     MA[11] = G                                store in RANDOM
                   50⊁;              EXIT;                                              [11]

real procedure RANDOM;

          0.  ⊀121,0,               TDA(0)               S = 0
          1.    128,4718593,        CODE                 A = 1
          2.    128,22576392,        "                   S = MT[8]           S = RANDOM[11]
          3.    128,35159300,        "                   MULAS(MT[4])        MULAS(RANDOM[9])
          4.    128,30964998,        "                   MT[6] = S           RANDOM[11] = S
          5.     86,60               TIC(60)              G = M[60]           G = S
          6.    128,-13075197,       CODE                F/MT[2]             F/RANDOM[9]
          7.     50,                 EXIT                 GOTOR(MC[-1])
hier      8.    128,26353589                                                  AP121
begint    9.    128,1                                                         constante uit
werk-    10.    128,0                                                           samen 2↑26
ruim-    11.    128,33554432⊁;                           2↑25                overschrijf-
te                                                                                 baar.
```

Overigens: De wet staat geene regtsvordering toe, terzake van eene schuld uit spel of uit weddenschap voortgesproten. B.W. art. 1825.

1) De procedures RANDOM en SETRANDOM zijn ontleend aan een programmatekst die de N.V. Electrologica toevallig had achtergelaten in de bunker.

25.3. Logisch product

```
integer procedure logprod (x,y); value x,y; intiger x,y.

0.  ⊀121,0,          TDA(0)      S = 0
1.    128,13138688,  CODE        A = :MC          parameterwijzer
2.     54,           CIV
3.     54,           CIV                          ↓, x, y↑
4.    128,5798655,   CODE        A = MC[-1]       ↓, x, ↑
5.    128,47741695,  CODE        A'x'MC[-1]        stapel leeg
6.     86,59,        TIC(59)     G = M[59]         G = A
7.     50⊁,          EXIT        GOTOR(MC[-1])
```

N.B. logische som: opdracht 5 wijzigen in 128, 43547 391,

25.4. Een leesprocedure

Onderstaandeprocedure leest alleen integers van de band. Reals worden zonder meer overgeslagen tot een integer op de band wordt aangetroffen.

```
integer procedure intread;
        ↓121,0,          TDA(0)          S = 0
          132,           READ            F = READ
          128,21333721,  CODE            S = F,Z
          128,-44336384, CODE            Y, EXIT ;Y, klaar en weg
          103, intread,  JU1(intread)    terug naar intread[1]
           50↑;          EXIT
```

25.5. EVEN

Beschouw de ALGOL procedure
integer procedure EVEN(n); value n, integer n;EVEN = if n:2*2 = n
then 1 else -1;
In plaats hiervan zou men ook (om de tijdrovende heling te vermijden) kunnen gebruiken:
even: = if(n>0) ≡ (laatste bit(n) = 0) then 1 else -1
als dit in ALGOL codeerbaar was. Helaas kan dit niet in ALGOL maar wel in een CP aldus:
integer procedure even(n); value n; integer n;

```
        ↓121,0,          TDA(0)   S = 0
          128, 13138688, CODE     A = :MC
           54,           CIV
          128,5929727,   CODE     A = MC[-1], P
           14,           STAB     stack boolean
          128,46923777,  CODE     A'=' 1,Z
           16,           QVL      stack ≡ CON?
           87,1          TSIC(1)  F = 1
          128,-44336384, CODE     Y, GOTOR(MC[-1])
            1,           NEG      F = -F
           50↑;          EXIT     GOTOR(MC[-1])
```

10000 EVEN's duurden 5.7 sec.
10000 events duurden 1.8 sec., zodat hiermede een tijdwinst van 68,5% bereikt wordt.

25.6. Faculteit

```
  real procedure fac(n); value n; integer n;

        ↓121,0           TDA(0)          S = 0
          128,13138688,                  A = :MC
           54,           CIV             ↓,n,↑
           87,1          TSIC(1)         initialisatie
            0,           STACK           ↓, n, 0, 1,↑
          103,fac1↑;     JU1(fac 1)      goto fac 1[1]
procedure
fac 1; 0. ↓121,0         TDA(0)
        1. 108, fac 2,   SUBJ(fac 2)     SUBC(:M[fac 2]) ga testen
        2. 128,-28262402,                G = M[B-3]
             4,          MUL
             0,          STACK           stack nieuwe waarde
          128,-28262402,                 G = M[B-3]
             0,          STACK
            87,1,        TSIC(1)
             3,          SUB             n: = n-1
          128,-11485186,                 M[B-3] = G
          103, fac 1↑;   JU1 (fac 1)     goto fac 1[1]
```

```
procedure
fac 2  0. 4121,0              TDA(0)
       1.  128,-28262433,              G = M[B-4]
            0,                STACK
           87,1,              TSIC(1)
           11,                MST        n ≤1?                    fac1[2]
          128,-44303616,                 N,GOTOR(MC[-1]) terug naar
          128,-28295170,                 F = M[B-3] eindresultaat
          110,4               DECB(4)    B-4. Weg met de stapel
           50,;               EXIT       EXIT fac.
```

Bovenstaande CP voor de faculteit is weinig efficient. Dit komt omdat bij de constructie het objectprogramma (zoals dat door de vertaler uit het corresponderende ALGOL programma wordt opgemaakt) te veel model gestaan heeft.

Een betere versie is de volgende:

```
real procedure fac(n); value n; integer n;

          4121,0,           TDA(0)
           128,13138688,    CODE          A = :MC
            54,             CIV           ↓, n,↑
            87,1,           TSIC(1)       F = 1          initialisatie
           103, fac 1 ;     JU1(fac 1)                   goto fac 1[1]

procedure
fac 1;     4121,0,           TDA(0)
           128,4718593,      CODE          A=:M[1]=1
           128,11714559,                   U,M[B-1] -A,P n > 1?
           128,-64389120,  N, B-1
           128,-44303616,  N,EXIT
           128,-15679488,  G* M[B-1]                    F: = F * n
           128,11550719,   M[B-1] -A                    n: = n-1
           103, fac 1 ,    JU1 (fac 1)                  goto fac 1[1]
```

Door vergelijking met het vorige voorbeeld komt men tot deze conclusie:

Slaafs de vertaler nadoen leidt tot inefficiente constructies. De hoofdoorzaak is te vinden in het feit dat de vertaler alle arithmetiek via het F(G) register laat lopen waardoor de overige registers (A en S vooral) op inefficiente wijze op nonactief worden gesteld.

Eenzelfde kritiek kanmuitgeoefend worden op een aantal eerder gegeven voorbeelden, die echter allen gmakkelijk in een meer efficiente vorm kunnen worden gegoten.

25.7. Ackermann

Een experiment is ondernomen met de funtie van ACKERMANN. In ALGOL kan deze aldus worden gedefinieerd:

```
integer procedure ACK(m,n); value m,n; integer m,n;
ACK: = if m = 0 then n + 1 else
    (if n = 0 then ACK(m-1,1) else ACK(m-1, ACK(m,n-1));
```

De Ackermann-funktie is nogal berucht geworden omdat zijn ondanks haar eenvoudige definitie zelfs voor kleine waarden van de parameters tot onvoorstelbaar lange berekeningen leidt.
(Men trachtte A(2,2) te berekenen!). In elk geval eist de berekening van ACK(4,1) meer geheugenruimte dan in de EL-x8 (thans) beschikbaar is.

Als CP in de volgende gebruikt:

```
integer procedure ACK(m,n); value m,n; integer m,n;
        ←121,0,          TDA(0)
          128,13138688,  A = MC
           54,           CIV
           54,           CIV                              ↓, m, n,↑
          108, ack 1,    SUBJ(ack 1)     SUBC(:ack 1)
          110,2,         B - 2
           50→;          EXIT

procedure
ack 1;  ←121,0           TDA(0)
          108,ack 2,     SUBJ(ack 2)                      test m
          108,ack 3,     SUBJ(ack 3)                      test n
          128,5259261,   A = M[B-3]                       m
           20,           STAA            MC = A
          128,5259261,   A = M[B-3]                       n
          128,2621441,   A - 1                            n - 1
           20,           STAA            MC = A           ↓,m,n,link,
                                                              m, n-1,↑
          108, ack 1,    SUBJ(ack 1)     SUBC(:ack 1)     bereken ack
          110,2,         B - 2                            (m,n-1) stapel
          128,-11485185, M[B-2] = G                       afbraak resul-
          121,1,         TDA(1)          S = 1            taat als nieuwe
          128,28327933,  M[B-3] = S                       m: = m-1 parameter
          103, ack 1→;   JU1 (ack 1)                      goto ack 1[1]

procedure
ack 2;  ←121,0,          TDA(0)                           test op m
          128,5554172,   U, A = M[B-4] ,Z                 m = 0?
          128,-44303616, N, EXIT                          N,goto ack1[2]
           87,1,         TSIC(1)         F = 1
          128,-32456706, G+M[B-3]                         n + 1
          110,1,         B-1               laatste link kan weg
           50→;          EXIT

procedure
ack 3;  ←121,0,          TDA(0)                           test op n
          128,5554713,   U, A=M[B-3] ,Z                   n = 0?
          128,-44303616, N,EXIT                           N,goto ack1[3]
          110,1,         B - 1             laatste link kan weg
          121,1,         TDA(1)          S = 1
          128,30425086,  M[B-2] = S      n: = 1          } nieuwe
          128,28327933,  M[B-3] = S      m: = m-1        } parameters
          103, ack 1→;   JU1 (ack 1)                      goto ack 1[1]
```

Wat betreft de resultaten het volgende:

```
      for i: = 0 step 1 until 3 do for j: = 0 step 1 until 5 do
      ACKERMANN(i,j,);
kostte
      met ACK(i,j,)          38,27 sec
  en  met ack(i,j,)           9,68 sec
waarvan
          ACK(3,5)           25,48 sec      (d.i. bijna een halve minuut)
  en      ack(3,5)            3,27 sec
kostte.
```

26. Slotopmerkingen

De samenstelling van dit memorandum werd bemoeilijkt door een gebrek aan beschikbare documentatie. Dit is voor een groot deel gecompenseerd kunnen worden door een aantal korte experimenten. Niettemin blijft het als handleiding een voorlopig karakter behouden. Bij beste weten van de samensteller zijn alle beweringen òf uit hetgeen wel beschikbaar was overgenomen èn/òf getoetst door een klein experiment.
Het geschrevene werd aanvankelijk samengesteld door de schrijver om aldoende te ontdekken waar precies de hiaten in zijn kennis zaten. Bij de realisatie bleek, dat er toch een tamelijk volledig beeld kon worden gegeven van de mogelijkheden. Het is vooral daarom onwaarschijnlijk dat voor de praktijk belangrijke elementen ontbreken.
Voorzover hiaten bekend zijn is geconstateerd dat deze betrekking hebben op constructies die in de praktijk niet nodig zullen zijn of die als te gecompliceerd ongewenst zijn. Daarbij moet bedacht worden dat er in de praktijk slechts 2 situaties zijn waarin het gebruik van een CP kan worden overwogen:

1. De overeenkomstige ALGOL-procedure kan niet worden samengesteld (b.v. available)
2. De overeenkomstige ALGOL-procedure is inefficient, hetgeen vooral voorkomt in procedures met geneste cycli waarbinnen met arrays wordt gemanipuleerd. Als de CP dan niet eenvoudig is, wordt er toch geen winst behaald.

De conclusie is dus dat ondanks het z.g. voorlopig karakter van de "handleiding" zij toch praktisch het nodige bevat.
(Opmerking:
Enkele tot veel complicaties aanleiding gevende constructies zijn bewust niet beschreven, hoewel ze wel bekend waren (Voorbeeld: slot par. 19.2.))

De bemoeienis met het onderhavige onderwerp is vooral ontstaan uit de overweging dat de SPG - NVM in toekomstige projecten (baroklinе modellen) snelle codeprocedures om redenen van tijdwinst mogelijk niet zal kunnen missen zodat het gewenst was tijdig over informatie en ervaring te beschikken.

APPENDIX

A1. Tabel van besproken macro's.

nr.	SYMBOOL	para.	nr.	SYMBOOL	para.	nr.	SYMBOOL	para.
0	STACK	14	38	DECS	15	96	SSTI	12
1	NEG	15	50	EXIT	22	97	STB	13
2	ADD	15	51	TEST1	16	103	JU1	24.2
3	SUB	15	52	TEST2	16	108	SUBJ	19.2
4	MUL	15	53	CRV	20	110	DECB	15
5	DIV	15	54	CIV	20	111	DO	24.1
6	IDI	15	55	CBV	20	115	INCRB	15
7	TTP	15	58	CEN	20	121	TDA	10;20
8	EQU	16	59	CLPN	20	128	CODE	8
9	UQU	16	74	JUA	23	129	SIN	19.1
10	LESS	16	76	ABS	15	130	COS	19.1
11	MST	16	77	SIGN	15	131	ARCTAN	19.1
12	MOR	16	78	ENTIER	19.1	132	READ	19.1
13	LST	16	79	SQRT	19.1	133	FIXP	19.1
14	STAB	17	80	EXP	19.1	134	ABSFIXP	19.1
15	NON	17	81	LN	19.1	135	FLOP	19.1
16	QVL	17	84	TIV	12	136	PUNCH	19.1
17	IMP	17	83	TRV	12	137	PUNLCR	19.1
18	OR	17	86	TIC	12	138	PUSPACE	19.1
19	AND	17	87	TSIC	11	139	RUNOUT	19.1
20	STAA	14	88	TBV	13	140	REHEP	19.1
21	TSR	18.2	89	TBC	11	141	PUHEP	19.1
22	TSI	18.2	91	TLV	23	142	HAND	19.1
25	TFSU	21	92	TAK	12;18.2	143	XEEN	19.1
29	STSR	18.5	93	TAA	19	144	STOP	19.1
30	STSI	18.5	94	STR	12	8987136	afronden	22
34	STFSU	21	95	STI	12			

A2. Interne representatie van opdrachten

A2.1. Opdrachten

A + x	0	B + x	-67 108 863
A - x	2 097 152	B - x	-65 011 711
A = x	4 194 304	B = x	-62 914 559
A = -x	6 291 456	B = -x	-60 817 407
x + A	8 388 608	x + B	-58 720 255
x - A	10 485 760	x - B	-56 623 103
x̂ = A	12 582 912	x̂ = B	-54 525 951
x = -A	14 680 064	x = -B	-52 428 799
S-opdrachten	A-opdracht + 16 777 216	F + x	-33 554 431
		F - x	-31 457 279
MULAS(x)	33 554 432	F = x	-29 360 127
MULAS(-x)	35 651 584	F = -x	-27 262 975
MULS(x)	37 748 736	F * x	-16 777 215
MULS(-x)	39 845 888	F/x	-14 680 063
A' + 'x	41 943 040	x = F	-12 582 911
A' + '-x	44 040 192	x = -F	-10 485 759
A' * 'x	46 137 344	G-opdrachten	als U, F-opdracht
A' * '-x	48 234 496		
DIVAS(x)	50 331 648		
DIVAS(-x)	52 428 800		
DIVA(x)	54 525 952		
DIVA(-x)	56 623 104		
S' + 'x	58 720 256		
S' + '-x	60 817 408		
S' * 'x	62 914 560		
S' * '-x	65 011 712		
PLUSSA	als U, x=A		
MINA	als U, x=-A		

N.B. Voor niet vermelde opdrachten zie ELAN-handleiding. De meeste niet vermelde opdrachten kunnen eenvoudig via macro's worden gerealiseerd.

x = geheugenoperand in geoorloofde adressering.

A2.2. Varianten

A2.2.1. Adresseringsvarianten

statisch		0
: statisch	(:M[n] met $0 \leq n \leq 32762$)	524 288 + n
STATB	(M[B + n] met $\|n\| \leq 16383$)	1064 960 + n
DYN	(GAST-varianten)	1572 864
	met voor	
	MG[n]	29 952 + n
	MA[n]	30 464 + n
	MS[n] ($\|n\| < 512$)	30 976 + n
	MC[n]	31 488 + n
	MT[n]	32 000 + n
	MD[n]	32 512 + n [1]
	Mp[q]	p*512 + 256 + q

: DYN (in opdracht) als uit opdracht met :STAT variant

A2.2.2 Conditiezettende varianten

P	131 072
Z	262 144
E	393 216

A2.2.3 Conditie volgende varianten

U	32 768
Y	65 536
V	98 304

De bij de varianten vermelde constanten moeten worden opgeteld bij de codering van de opdracht, ongeacht het teken van de interne codering van de opdracht.

Voorbeeld:

G = MA?		
F = x	⇒	-29 360 127
DYNVAR	⇒	1 572 864
wegens G, U-VAR	⇒	-27 787 263
		32 768
MA[0]	⇒	-27 754 495
		30 464
Dus G = MA = 128,		-27 724 031

1) Het gebruik van D,Mp[q] en MD-adressering moet worden afgeraden wegens gevaar van overschrijving van door de vertaler reeds in gebruik gestelde geheugenruimte.

A3. Interne representatie van enige opdrachten, oktaal geschreven.

A + x	0		B + x	400 000 000	MD[0]	77 400	
A - x	10 000 000		B - x	410 000 000	MT[0]	76 400	
A = x	20 000 000		B = x	420 000 000	MC[0]	75 400	
A = -x	30 000 000		B = -x	430 000 000	MS[0]	74 400	
x + A	40 000 000		x + B	440 000 000	MA[0]	73 400	
x - A	50 000 000		x - B	450 000 000	MG[0]	72 400	
x = A	60 000 000		x = B	460 000 000			
x = -A	70 000 000		x = -B	470 000 000			
					dec. adres:	oktaal adres:	
S + x	100 000 000						
S - x	110 000 000		F + x	600 000 000	F=M[57]	M[71]	
S = x	120 000 000		F - x	610 000 000	G=M[58]	M[72]	
S = -x	130 000 000		F = x	620 000 000	A=M[59]	M[73]	
x + S	140 000 000		F = -x	630 000 000	S=M[60]	M[74]	
x - S	150 000 000		F * x	700 000 000	B=M[61]	M[75]	
x = S	160 000 000		F/x	710 000 000	T=M[62]	M[76]	
x = -S	170 000 000		x = F	720 000 000	D=M[63]	M[77]	
MULAS(x)	200 000 000		x = -F	730 000 000			
MULAS(-x)	210 000 000						
MULS(x)	220 000 000		DYN variant	6 000 000			
MULS(-x)	230 000 000		STAB variant	4 000 000	+ 40 000 = 4040 000		
A '+' x	240 000 000		:STAT variant	2 000 000			
A '+' -x	250 000 000		E- variant	1 400 000			
A '*' x	260 000 000		Z- variant	1 000 000			
A '*' -x	270 000 000		P- variant	400 000			
DIVAS(x)	300 000 000		N- variant	300 000			
DIVAS(-x)	310 000 000		Y- variant	200 000			
DIVA(x)	320 000 000		U- variant	100 000			
DIVA(-x)	330 000 000						
S '+' x	340 000 000						
S '+' -x	350 000 000						
S '*' x	360 000 000						
S '*' -x	370 000 000						

x = geheugen operand in geoorloofde adressering.

Gebruiksaanwijzing

1. Tel alle bijdragen op. Eventueel toe te voegen constanten (adressen) natuurlijk ook oktaal behandelen. Dus MA[17] moet worden MA[21].
2. Wordt het getal aldus verkregen ≥ 400 000 000 dan trekke men het af van 777 777 777 en plaatse er een minteken voor. (*)
3. Converteer tot decimaal getal.
4. Voor de conversie decimaal ⇆ oktaal bestaat een tweetal beproefde recepten:

4.1.
decimaal-oktaal door deling door 8 en restbepaling
voorbeeld:

12735 decimaal = ?

8	12735	7 (rest)
8	1591	7
8	198	6
8	24	0
	3	

Dus $12735_{10} = 30677_8$

4.2 oktaal-decimaal

30677_8 = ?

30 677

2 * 3 = 6 _

24 677

2 * 24 = 4 8 _

19 877

2 * 198 = 3 96 _

15 917

2 * 1591 = 3 182 _

$12\ 735_{10}$

(Dit schema berust op n * 8 = n * (10 - 2))

(*) $377\ 777\ 777_8 = 67\ 108\ 863_{10}$ = integercapaciteit.

Voorbeeld

Gevraagd A = :MC

Volgens de regels moet een :DYN opdracht gecodeerd worden als de corresponderende uit opdracht met :STAT = variant.

We krijgen dus:

x = A	60 000 000
:STAT	2 000 000
MC[0]	75 400
	$62\ 075\ 400_8$

Begin conversie:

12
50 075 400
10 0
40 075 400
8 00
32 075 400
6414
25 661 400
5 1322
20 529 200
4105 84
16 423 360
3 284 672
$13\ 138\ 688_{10}$ naar behoren.

Automatisering

Uit het bovenstaande blijkt dat het enige wat bij het gebruik van oktale getallen vervelend is, de conversie is naar decimale getallen. Maar dat kan wel geautomatiseerd worden. Aangenomen mag daarbij worden dat zelfs punt 2) van de gebruiksaanwijzing aan de machine kan worden overgelaten.
Aan werk blijft dan slecht over wat voor "Begin conversie" staat. En de hierbij uit te voeren optelling gaat zonder moeite vanwege overmaat aan nullen op gunstig gelegen posities.
Maar, waarom nog niet één stap verder gegaan. Dan komt men terecht bij een soort compiler die in de volgende appendix wordt beschreven.

A.4 Compiler

Laat een of meer codeprocedure op één band geponst worden met inachtneming van de volgende afspraken:

1. Alle tot nu toe gedefinieerde macro's hebben de gewone betekenis. (Ook 128!)
2. Er is 1 nieuwe macro, nummer 148, die dezelfde functie heeft als 128, maar waarvan de (valuelike) parameter oktaal wordt geschreven. Daarbij behoeft geen maatregel genomen worden tegen eventuele overflow boven 377 777 777. Wel moet de parameter altijd uit 9 cijfers bestaan en dus zonodig met 1 of meer nullen beginnen.
3. Van de geoptimaliseerde macro's (nummer $>$ 148) mogen alleen die voorkomen, die niet van een parameter zijn voorzien.
4. De ponsband die deze procedures (en niets anders) bevat wordt afgesloten met het symbool vraagteken.

Indien de aldus geponste band als getallenband wordt gebruikt bij de executie van het hieronder volgende programma getiteld compile, dan bestaat de uitvoer uit een reproductie van de meegeleverde tekst met dien verstande dat de macro (148, parameter, oktaal) getransformeerd is tot macro (128, zelfdeparameter, decimaal, gecorrigeerd voor overflow).

Het extra werk dat verricht moet worden voor het gebruik van opdrachten in interne representatie is daardoor tot een minimum teruggebracht. Om met de Hatter uit Alice in Wonderland te spreken "I can't go no lower, I'm on the floor as it is".

-o-o-o-

```
begin   comment BOUD-compile-130569.
                Compiler voor codeprocedures. Versie 2
                Gebruiksaanwijzing: zie handleiding;
        integer i,j,number,symbol,balk,quote,unquote,oktal,comma,separator,line,plus,minus;
        boolean neg;
        boolean array parameter,valuelike[0:148];

        procedure LINE; begin PUNLCR; PUSPACE(8) end;

        procedure OUT(x); value x; integer x;
        begin     if x < 0 then FIXP(8,0,x) else ABSFIXP(8,0,x) end;

        integer procedure N(x); value x; integer x;
        N:= if neg then 7-x else x;

        integer procedure NUMBER;
        begin   neg:= false;
        LL:     j:= RESYM; if j > 9 ∧ j ≠ plus ∧ j ≠ minus then goto LL;
                if j > 9 then begin neg:= j = minus; j:= 0 end;
                i:= j;
        KK:     j:= RESYM; if j ≤ 9 then
                begin i:= i×10+j; goto KK end;
                separator:= j; NUMBER:= if neg then -i else i
        end;

        procedure final(m); value m; integer m;
        begin
        SS:     if m ≠ line then PUSYM(m); if m = comma then goto incode;
                if m = balk then goto outcode; m:= RESYM; goto SS
        end;

        RUNOUT; PUNLCR; LINE;
        PUTEXT(<comment output compiler codeprocedures;>); PUNLCR;
        balk:= 127; quote:= 72; unquote:= 74; comma:= 87; oktal:= 148; plus:= 64; minus:= 65; line:= 119;

        for i:= 0, i+1 while i ≤ 148 do
        begin parameter[i]:= valuelike[i]:= false end;
        for i:= 83 step 1 until 128,145 step 1 until 148 do parameter[i]:= true;

        for i:= 85,86,87,89,102,106,107,110step 1 until 122,125,126,127,128,145 step 1 until 148 do valuelike[i]:= true;

outcode: symbol:= RESYM; if symbol = 122 then goto KLAAR; PUSYM(symbol);
        if symbol ≠ balk then goto outcode;
        symbol:= RESYM; PUSYM(symbol);
        if symbol ≠ quote then goto outcode;

incode: LINE; number:= NUMBER; if number > oktal then
        begin OUT(number); final(separator) end;
        ABSFIXP(3,0,if number = oktal then 128 else number);
        if ¬ parameter[number] then final(separator); PUSYM(comma);
        if ¬ valuelike[number] then
        begin
        TT:     if separator = comma then final(RESYM);
                separator:= RESYM; goto TT
        end;
        if number ≠ oktal then
        begin OUT(NUMBER); final(separator) end;
PP:     j:= RESYM; if j > 9 then goto PP;
        neg:= j ≥ 4; i:= N(j);
QQ:     j:= RESYM; if j ≤ 9 then
        begin i:= i×8 + N(j); goto QQ end;
        OUT(if neg then -i else i); final(j);

KLAAR:  PUNLCR; RUNOUT
end
```