RA

stichting
mathematisch
centrum

MC

REKENAFDELING RA MR 121/71 FEBRUARI

F.E.J. KRUSEMAN ARETZ
HET OBJECTPROGRAMMA, GEGENEREERD DOOR DE
X8 - ALGOL 60 - VERTALER VAN HET M.C.

RA

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

Voorwoord

Dit rapport beschrijft zo volledig mogelijk het objectprogramma dat door de MC-ALGOL 60-vertaler voor de EL X8 wordt gegenereerd. Slechts die details die voorkennis vereisen van vertaler, run-time-organisatie of enig bedrijfssysteem zijn wel genoemd maar niet uitgewerkt. Uiteraard is kennis van het Revised Report [1] en van de EL X8 wel verondersteld.

Het ontwerp van het objectprogramma en van de run-time-organisatie is in de jaren 1963/1965 gemaakt door J.J.B.M. Nederkoorn en F.E.J. Kruseman Aretz. De vertaler werd geschreven door F.E.J. Kruseman Aretz in 1964/1965, eerst in ALGOL 60 (deze tekst zal nog gepubliceerd worden [3]) en daarna in ELAN. De ELAN-tekst van vertaler, run-time-routines en een eenvoudig bedrijfssysteem is in [2] vastgelegd.

Dit rapport werd grotendeels in 1965 geschreven op het Mathematisch Centrum. Het werd voltooid op het Natuurkundig Laboratorium van de N.V. Philips' Gloeilampenfabrieken te Eindhoven. Graag wil de auteur zijn erkentelijkheid uitspreken jegens de directie van dit laboratorium voor de gelegenheid, hem geboden dit rapport alsnog te voltooien en te publiceren.

De auteur is ook dank verschuldigd aan drs. K.K. Koksma voor vele kritische opmerkingen, met name wat betreft hoofdstuk 6, en aan H.W. Roos Lindgreen voor hulp bij de samenstelling van appendix A.

[1] P. Naur (ed.), Revised Report on the Algorithmic Language ALGOL 60, Regnecentralen, Kopenhagen 1962.

[2] F.E.J. Kruseman Aretz en B.J. Mailloux, The ELAN source text of the MC ALGOL 60 system for the EL X8, Mathematisch Centrum, Amsterdam 1966, rapport MR 84.

[3] F.E.J. Kruseman Aretz, P.J.W. ten Hagen en H.L. Oudshoorn, The ALGOL source text of the MC ALGOL 60 translator for the EL X8, Mathematisch Centrum, Amsterdam 1971.

Inhoud

1. Arithmetische en Boolean expressies

1.1. Met arithmetische expressies correspondeert een stuk objectprogramma, dat de waarde van de expressie in F vormt.
Het objectprogramma, behorend bij Boolean expressies, vormt zijn resultaat in de conditie C.

1.2. De vertaling van de arithmetische expressie "if BE then AE1 else AE2" luidt symbolisch:

```
                C = BE
            N, GOTO (:L0)          " COJU (L0)
                F = AE1
                GOTO (:L1)         " JU (L1)
        L0:     F = AE2
        L1:
```

Analoog resulteert de Boolean expressie "if BE1 then BE2 else BE3" in:

```
                C = BE1
            N, GOTO (:L0)
                C = BE2
                GOTO (:L1)
        L0:     C = BE3
        L1:
```

1.3. De vertaling van arithmetische primary's

1.3.1. de simpele, locale of globale variabele:
Indien deze van het type integer is, beslaat de variabele 1 geheugenplaats, in geval van het type real 2.
De adresseringswijze kan zijn:
a) dynamisch, als de variabele gedeclareerd is in een procedure body of een binnenblok daarvan, maar niet het predicaat own heeft (zie 8.),
b) statisch, als dit niet het geval is (zie 7.4.).
We krijgen zo de opdrachten:

```
statisch, real:         F = M[x]        " TRV (x)
dynamisch, real:        F = Mr[q]

statisch, integer:      G = M[i]        " TIV (i)
dynamisch, integer:     G = Ms[t]
```

1.3.2. de simpele, formele variabele:
Deze is steeds dynamisch geadresseerd, en de vertaling luidt:

```
                        DOS (Mu[v])     " DOS (f)
```

Hiermee wordt een instructie uit de stapel, de z.g. APIC 0-instructie, uitgevoerd; deze instructie is daar gevormd bij het binnenkomen van de procedure, op grond van de aard van de corresponderende actuele parameter (zie 6.3.2.).

1.3.3. de locale of globale subscripted variable:
Bij ieder niet-formeel array behoort in X8-ALGOL een pseudo-variabele, die 1 geheugenplaats beslaat. Deze z.g. array-sleutel bevat het adres van de reference-vector, die de verdere informatie over het array bevat. De lengte van de reference-vector bedraagt 3 × dimensie + 4. De array-sleutel is geadresseerd volgens de regels, vermeld in 1.3.1.
Het ophalen van een subscripted variable gebeurt in twee stappen:
a) het maken van een adresdescriptie,
b) het uitvoeren van de indicering en het opnemen van de gevraagde waarde in het betreffende register.

1.3.3.1. de adresdescriptie:
Het hierbij behorende objectprogramma zet alle indexwaarden, behalve de laatste, op de stapel, neemt de laatste indexwaarde in F, en de array-sleutel in A. Voor de variabele "AR[AE1, AE2]" krijgen we zo symbolisch:

```
                F = AE1
                MC =  F          " STACK
                F = AE2
statisch:       A = M[AR]        " TAK (AR)
```

waarbij de laatste opdracht ook kan luiden:

```
dynamisch:      A = Mm[n]
```

1.3.3.2. de indicering en het halen:
Voor arithmetische array's gebeurt de tweede fase door het objectprogramma:

```
real:           SUB (:IND)       " TSR
                F =  MA

integer:        SUB (:IND)       " TSI
                G =  MA
```

1.3.4. de formele subscripted variable:
In vergelijking met 1.3.3. zijn de verschillen:
a) bij het maken van de adresdescriptie wordt de laatste opdracht vervangen door:

```
                DOS (Mu[v])      " DOS (F)
```

Het uitvoeren van de met [u,v] geadresseerde APIC 0-instructie zal de array-sleutel in A dienen af te leveren,
b) de tweede fase gebeurt door de aanroep:

```
                SUB2 (:TFSU)     " TFSU
```

Hierbij is SUB2 een subroutinesprong als SUB met zijn link op LINK[2] = M[10]. Deze opdracht is in alle hardware aanwezig, maar niet in elke ELAN-assembler opgenomen. De subroutine TFSU voert de indicering uit en neemt, op grond van de analyse van de reference-vector, 1 of 2 woorden in het betreffende register op.

1.3.5. de locale of globale functie-designator (niet-standaardfunctie):
Bij iedere niet-formele functie-designator behoort een statisch adres in het objectprogramma, waar de vertaling van de procedure body begint. De vertaling van de aanroep "P (3, 110)" luidt:

```
SUBC (:P)            " SUBJ (P)
(7 × d20 +   3)      " APD (3)
(7 × d20 + 110)      " APD (110)
```

Direct op de aanroep van de procedure volgen dus enige Actual Parameter Descriptors, waaruit bij binnenkomst van de procedure o.a. de APIC 0-instructies (die in het voorbeeld F = 3 respectievelijk F = 110 luiden) worden afgeleid. Voor het vertaalschema van aanroepen met ingewikkelder parameters verwijzen we naar 3. Zie voorts 10.

1.3.6. de formele functie-designator:
Als Fct een formele parameter is, luidt de vertaling van "Fct (3)":

```
DOS (Mu[v])          " DOS (Fct)
(7 × d20 +   3)      " APD (3)
```

Ook hier volgen de APD's direct op de "aanroep".

1.3.7. unsigned numbers:
De unsigned numbers vallen in twee groepen uiteen:
a) integers < 32768,
b) alle andere numerieke constanten.
De constanten uit de tweede groep worden verzameld in een constantenlijst, die aan het objectprogramma vooraf gaat (zie 7.2.); ze zijn steeds statisch geadresseerd. Voor de kleine integers van groep a is plaats direct in het adresdeel van de "haalopdracht" zelf.
Zo krijgen we:

```
real numbers:        F = M[rn]       " TRC (rn)

grote integers:      G = M[m]        " TIC (m)

kleine integers:     F =  [n]        " TSIC (n)      0 ≤ n < 32768
```

1.3.8. de expressie tussen ronde haakjes:
Hiermee correspondeert een stuk objectprogramma, dat de waarde van de expressie in F vormt.

1.4. De vertaling van arithmetische operaties

1.4.1. Behoudens de in 1.4.2. te bespreken optimaliseringen luidt het symbolische vertaalschema voor arithmetische operaties:

```
+ AE1:               F = AE1

- AE1:               F = AE1
                     F = - F         " NEG

AE1 + AE2:           F = AE1
                     MC =  F         " STACK
                     F = AE2
                     F + MC[-2]      " ADD
```

```
AE1 - AE2:          F = AE1
                    MC =  F          " STACK
                    F = AE2
                    F = - F          " SUB
                    F + MC[-2]

AE1 × AE2:          F = AE1
                    MC =  F          " STACK
                    F = AE2
                    F × MC[-2]       " MUL

AE1 / AE2:          F = AE1
                    MC =  F          " STACK
                    F = AE2
                    MC =  F          " DIV
                    F = MC[-4]
                    F /  MC

AE1 : AE2:          F = AE1
                    MC =  F          " STACK
                    F = AE2
                    SUBC (:IDI)      " IDI

AE1 ↑ AE2:          F = AE1
                    MC =  F          " STACK
                    F = AE2
                    SUBC (:TTP)      " TTP
```

Uit bovenstaande schema's zijn, met inachtneming van de prioriteitsregels, de programma's voor willekeurige expressies af te leiden. Voldaan is aan de regel, dat de primary's worden geevalueerd in de volgorde van links naar rechts (er vindt derhalve geen "verwisseling van operanden" plaats).

1.4.2. optimalisering van arithmetische operaties:

1.4.2.1. de unaire operatie "-":
De opdrachtenparen, bestaande uit een van de macro's TRV, TIV (zie 1.3.1.), TRC, TIC, of TSIC (zie 1.3.7.), gevolgd door de macro NEG (zie 1.4.1.), worden, waar mogelijk, vervangen door de macro's: TNRV, TNIV, TNRC, TNIC, of TNSIC. Deze zijn van de vorm:

```
statisch, real:     F = - M[x]       " TNRV (x)
dynamisch, real:    F = - Mr[q]

statisch, integer:  G = - M[i]       " TNIV (i)
dynamisch, integer: G = - Ms[t]

real numbers:       F = - M[rn]      " TNRC (rn)

grote integers:     G = - M[m]       " TNIC (m)

kleine integers:    F = - [n]        " TNSIC (n)     0 ≤ n < 32768
```

Een voorbeeld van een programma, waarin deze optimalisering niet moge-

lijk is, wordt gegeven door het objectprogramma van "– (if BE then 1 else 2)":

```
                C = BE
            N, GOTO (:LO)          " COJU (LO)
               F =  1              " TSIC (1)
               GOTO (:L1)          " JU (L1)
        LO:    F =  2              " TSIC (2)
        L1:    F = – F             " NEG
```

1.4.2.2. de binaire operaties "+, –, ×, /":
De macro-tripels, gevormd door de macro STACK, gevolgd door een van de macro's TRV, TIV, TRC, TIC, of TSIC, gevolgd door een van de macro's ADD, SUB, MUL, of DIV, worden waar mogelijk vervangen door de ermee corresponderende macro uit de serie ADDRV t/m DIVSIC, waarvan de vorm symbolisch wordt weergegeven door:

```
AE + x:        F = AE
               F +  x          " ADDRV (x)

AE – x:        F = AE
               F –  x          " SUBRV (x)

AE × x:        F = AE
               F ×  x          " MULRV (x)

AE / x:        F = AE
               F /  x          " DIVRV (x)
```

Een volledige lijst is in appendix B opgenomen.

1.4.3. voorbeelden:

"a × b + c × d":
F = a; F × b; MC = F; F = c; F × d; F + MC[–2]

"ar[i] ↑ 2" :
G = i; A = ar; SUB (:IND); F = MA; MC = F; F = 2; SUBC (:TTP)

"a × P (0) + 3":
F = a; MC = F; SUBC (:P); (7 × d20 + 0); F × MC[–2]; F + 3

1.5. De vertaling van Boolean primary's

1.5.1. de simpele, locale of globale variabele:
Deze beslaat steeds 1 geheugenplaats, waarvan de adressering de in 1.3.1. gegeven regels volgt. De waarde van de variabele is true als de geheugenplaats een positief getal bergt, false, als dat getal negatief is. De haalopdrachten luiden:

```
statisch:      S = M[b], P      " TBV (b)
dynamisch:     S = Ms[t], P
```

Het resultaat komt hiermee in de conditie C.

1.5.2. de simpele, formele variabele:
Hiervoor geldt woordelijk het in 1.3.2. vermelde. Het uitvoeren van de APIC 0-instructie zal uiteindelijk een resultaat in C moeten afleveren.

1.5.3. de locale of globale subscripted variable:
Bij Boolean array's beslaan de elementen slechts 1 bit, 0 voor een element true, 1 voor een element false . Deze elementen worden bitsgewijze gepakt in woorden: iedere geheugencel bevat 27 array-elementen.
Afgezien hiervan is de behandeling van Boolean array's geheel analoog aan die van arithmetische array's, en kan hiervoor grotendeels verwezen worden naar 1.3.3. Het maken van de adresdescriptie gebeurt als in 1.3.3.1., het indiceren en het opnemen van de gevraagde waarde in C vindt plaats door de macro:

```
SUB1 (:INDB)          " TSB
S 'x' MA, Z
```

Ter toelichting diene, dat de subroutine INDB (met link op LINK[1] = M[9]) in A het geheugenadres aflevert van het woord, dat het gevraagde element bergt, terwijl in S een masker wordt gevormd met 26 nullen en 1 een, precies op de positie van het gevraagde element in het door A aangewezen woord.

1.5.4. de formele subscripted variable:
Voor de adresdescriptie verwijzen we naar 1.3.4., terwijl de verdere afwerking door de in 1.5.3. beschreven macro TSB gedaan wordt.

1.5.5. de functie-designator:
Het in 1.3.5. en 1.3.6. vermelde is zonder meer van toepassing ook op Boolean functie-designators.

1.5.6. de logische waarde:
Hiermee correspondeert de macro TBC van de volgende gedaante:

```
true:     S = 0, Z        " TBC (true)
false:    S = 1, Z        " TBC (false)
```

die de conditie C passend zet.

1.5.7. de relaties:

1.5.7.1. Behoudens de in 1.5.7.2. te bespreken optimaliseringen luidt het symbolische vertaalschema voor de 6 relaties:

```
AE1 = AE2:    F = AE1
              MC = F          " STACK
              F = AE2
              F - MC[-2], Z   " EQU
```

```
AE1 ≠ AE2:      F = AE1
                MC =  F              " STACK
                F = AE2
                F – MC[–2], Z        " UQU
                S = – T, P

AE1 < AE2:      F = AE1
                MC =  F              " STACK
                F = AE2
                F – MC[–2], P        " LES
              Y, F –  0, P

AE1 ≤ AE2:      F = AE1
                MC =  F              " STACK
                F = AE2
                F – MC[–2], P        " MST
              N, F =  F, Z

AE1 > AE2:      F = AE1
                MC =  F              " STACK
                F = AE2
                F – MC[–2], P        " MDR
              N, F =  F, Z
                S = – T, P

AE1 ≥ AE2:      F = AE1
                MC =  F              " STACK
                F = AE2
                F – MC[–2], Z        " LST
              N, S = – 1, E
```

Ook hier worden dus de primary's geevalueerd van links naar rechts, zonder "verwisseling van operanden".

1.5.7.2. optimalisering van relaties:
De macro–tripels, gevormd door de macro STACK, gevolgd door een van de macro's TRV, TIV, TRC, TIC, of TSIC, gevolgd door een van de macro's EQU, UQU, LES, MST, MDR, of LST, worden waar mogelijk vervangen door de ermee corresponderende macro uit de serie EQURV t/m LSTSIC, waarvan de vorm symbolisch wordt weergegeven door:

```
AE = x:         F =  AE
                F –  x, Z            " EQURV (x)

AE ≠ x:         F =  AE
                F –  x, Z            " UQURV (x)
                S = – T, P

AE < x:         F =  AE
                F –  x, P            " LESRV (x)
              N, F =  F, Z
                S = – T, P
```

```
AE ≤ x:          F = AE
                 F — x, Z          " MSTRV (x)
              N, S = — 1, E

AE > x:          F = AE
                 F — x, P          " MDRRV (x)
              Y, F — 0, P

AE ≥ x:          F = AE
                 F — x, P          " LSTRV (x)
              N, F = F, Z
```

Een volledige lijst is in appendix B opgenomen.

1.5.8. de Boolean expressie tussen ronde haakjes:
Hiermee correspondeert een stuk objectprogramma, dat de waarde van de expressie in C vormt.

1.6. De vertaling van Boolean operaties

1.6.1. Het symbolische vertaalschema voor de Boolean operaties luidt:

```
¬ BE:            C = BE
                 S = — T, P        " NON

BE1 ∧ BE2:       C = BE1
                 S = T             " STAB
                 MC = S
                 C = BE2
              N, B — 1             " AND
              Y, S = MC[—1], P

BE1 ∨ BE2:       C = BE1
                 S = T             " STAB
                 MC = S
                 C = BE2
              Y, B — 1             " OR
              N, S = MC[—1], P

BE1 ⊐ BE2:       C = BE1
                 S = T             " STAB
                 MC = S
                 C = BE2
              Y, B — 1             " IMP
              N, S = — MC[—1], P

BE1 = BE2:       C = BE1
                 S = T             " STAB
                 MC = S
                 C = BE2
                 S = T, P          " QVL
                 S = MC[—1], E
```

Ter toelichting van STAB diene, dat een logische waarde gestapeld

wordt als één woord, waarvan het tekenbit de waarde bepaalt analoog aan de waarde van simpele Boolean variabelen. Boolean operaties worden niet geoptimaliseerd.

1.6.2. voorbeelden:

"x = y ∧ i > 1":

```
F = x; F – y, Z; S = T; MC = S; G = i; F – 1, P; N, B – 1;
Y, S = MC[–1], P
```

"¬ Bo[i, j – 1]":

```
G = i; MC = F; G = j; F – 1; A = Bo; SUB1 (:INDB);
S 'X' MA, Z; S = – T, P
```

2. Overige expressies

2.1. Stringexpressies

2.1.1. Met stringexpressies correspondeert een stuk objectprogramma, dat een resultaat in A aflevert. Dit resultaat, de "stringwaarde", is het adres van een vector in de contrastapel, die nadere informatie over de geevalueerde string bevat.

2.1.2. De vertaling van de stringexpressie "if BE then SE1 else SE2" luidt symbolisch:

```
          C = BE
      N, GOTO (:L0)          " COJU (L0)
          A = SE1
          GOTO (:L1)         " JU (L1)
L0:       A = SE2
L1:
```

2.1.3. De vertaling van simpele stringexpressies:

2.1.3.1. de simpele, locale of globale stringvariabele:
Deze beslaat in het geheugen 2 opeenvolgende plaatsen, waarvan de eerste de stringwaarde bewaart, terwijl de tweede het adres bevat van het eerst volgende in de z.g. sleutelketting opgenomen object. In deze sleutelketting zijn alle stringvariabelen (ook de geindiceerde) op deze wijze opgenomen.
Stringvariabelen worden geadresseerd conform de in 1.3.1. beschreven regels; de haalopdracht luidt:

```
statisch:     A = M[st]        " TSTV (st)
dynamisch:    A = Mr[q]
```

2.1.3.2. de simpele, formele stringvariabele:
Hiervoor geldt woordelijk het in 1.3.2. vermelde. Het uitvoeren van de APIC 0-instructie zal uiteindelijk een resultaat in A afleveren.

2.1.3.3. de locale of globale subscripted stringvariabele:
Ook subscripted stringvariabelen zijn in de in 2.1.3.1. genoemde sleutelketting opgenomen en derhalve beslaan de elementen van string-array's 2 geheugenplaatsen elk.
De behandeling van string-array's is verder geheel analoog aan die van arithmetische array's. Het maken van de adresdescriptie gebeurt als in 1.3.3.1., het indiceren en het opnemen van de gevraagde stringwaarde in A vindt plaats door de macro:

```
SUB (:IND)          " TSST
A = MA
```

2.1.3.4. de formele subscripted stringvariabele:
Voor de adresdescriptie verwijzen we naar 1.3.4., terwijl de verdere afwerking door de in 2.1.3.3. beschreven macro TSST gedaan wordt.

2.1.3.5. de functie-designator:
Het in 1.3.5. en 1.3.6. vermelde is zonder meer van toepassing ook op functie-designators van type string. Hier zij opgemerkt, dat als neveneffect van de aflevering van het resultaat in A de vlag "loose string" uit het complex de waarde true (> 0) krijgt (zie 6.10.).

2.1.3.6. de constante string:
De karakters van de string worden, in semi-interne representatie (deze wordt in appendix C gegeven), in een aantal opeenvolgende woorden van het objectprogramma opgeslagen, en wel gepakt met 3 karakters per woord. In zo'n woord beslaan de stringkarakters de posities d23 t/m d0; op de meest significante plaats (d23 t/m d16) staat het voorste (meest linkse) karakter. De karakters van de string worden aangevuld met een karakter 255, en zoveel meer karakters 255 als nodig zijn om het laatste woord "vol" te maken. Als we de woorden gevuld met de stringkarakters en de end marker(s) weergeven met × × ×, luidt de complete vertaling van een constante string:

```
        SUBC (:TCST)        " TCST
        GOTO (:L1)          " JU (L1)
L0:     × × ×
        × × ×
L1:
```

De macro TCST vult in de contrastapel een vector van 4 woorden in. Een hiervan bevat het adres L0.

2.1.3.7. de stringexpressie tussen ronde haakjes:
Hiermee correspondeert een stuk objectprogramma, dat de waarde van de expressie in A vormt.

2.2. Designational expressies

2.2.1. Onder de waarde van een label zullen we verstaan de waarde van een pseudo label-variabele. Deze beslaat twee woorden in het geheugen en wordt, indien de label locaal is t.o.v. een procedure of een binnenblok daarvan, dynamisch geadresseerd. In dat geval worden bij iedere introductie van die procedure of dat blok voor de pseudo variabele twee woorden in de stapel gereserveerd en op passende wijze gevuld (zie 5.2., 5.3., 6.4. en 6.5.). Voor statisch te adresseren label-variabelen is dit reeds door de vertaler gedaan (zie 7.4.1.).

De waarde van een designational expressie is het adres van de pseudo label-variabele, behorend bij de label die uiteindelijk door de evaluatie van de expressie wordt aangewezen. Dit adres wordt in het A-register gevormd.

2.2.2. De vertaling van de designational expressie "if BE then DE1 else DE2" luidt symbolisch:

```
        C = BE
N,      GOTO (:L0)          " COJU (L0)
        A = DE1
        GOTO (:L1)          " JU (L1)
L0:     A = DE2
L1:
```

2.2.3. De vertaling van simple designational expressies:

2.2.3.1. de locale of globale label:
De haalopdracht luidt, conform 2.2.1.:

```
statisch:       A = :M[1]          " TLV (1)
dynamisch:      A = :Ms[t]
```

2.2.3.2. de formele label:
Deze is steeds dynamisch geadresseerd, en de vertaling luidt:

```
                DOS (Mu[v])        " DOS (f1)
```

Hiermee wordt de bij f1 behorende APIC 0-instructie uitgevoerd. Deze zal uiteindelijk een resultaat in A afleveren.

2.2.3.3. de switch designator:
De vertaling van de designational expressie "Sw[AE]", waarbij Sw een locale of globale switch identifier is, luidt symbolisch:

```
F =  AE
A = :Sw            " TSWE (Sw)
SUBC (:TSL)        " TSL
```

Ter toelichting diene, dat met iedere switchdeclaratie een stuk objectprogramma correspondeert (zie 5.6.); door de macro TSWE wordt het beginadres hiervan in het A-register genomen. De macro TSL selecteert en evalueert vervolgens het bedoelde switch-element.

Als Sw een formele switch identifier aanduidt, wordt de expressie "Sw[AE]" echter vertaald door:

```
F =  AE
DOS (Mq[r])        " DOS (Sw)
SUBC (:TFSL)       " TFSL
```

waarbij door de DOS-instructie enerzijds de bij Sw behorende APIC 0-instructie wordt uitgevoerd (waardoor het beginadres van een switch-programma in A komt), anderzijds in S het adres van de APIC zelf gevormd wordt, zodat TFSL over nadere gegevens van de met Sw corresponderende actuele parameter kan beschikken.

2.2.3.4. de designational expressie tussen ronde haakjes:
Hiermee correspondeert een stuk objectprogramma, dat de waarde van de expressie in A vormt.

2.3. Expressies van onbekend type

Deze kunnen voorkomen als rechterlid van een assignment of als actuele parameter.

Zodra in de expressie operatoren verschijnen, of identifiers van bekend type voorkomen, is er informatie over het type van de expressie verkregen en zal van deze informatie gebruik gemaakt worden. Het vertaalschema is dan een van de in de vorige secties behandelde constructies.

Evenzo zal bij de vertaling van "if BE then E1 else E2", als uit E1 het type van de expressie blijkt, bij de vertaling van E2 van deze informatie gebruik gemaakt worden.

2.3.1. De vertaling van variabelen en functie-designators verloopt als volgt: ofwel de bijbehorende identifier is locaal of globaal, op grond van het type daarvan volgt de vertaling, als bovengenoemd,
ofwel de bijbehorende identifier is formeel; weet de vertaler het type op grond van voorkomen, dan volgt de vertaling ook uit het voorafgaande, en anders worden simpele identifiers met een DOS-instructie vertaald, geindiceerde objecten door een adres-descriptie en de in 1.3.4. reeds eerder genoemde macro TFSU. Deze laatste is ook in staat in A een designational resultaat te vormen, als uit de analyse blijkt dat de actuele identifier die van een switch is.

2.3.2. De vertaling van constanten verloopt op de normale manier, tenzij het een unsigned integer betreft, in het programma integer labels voorkomen, en deze integer op deze plaats een locale of globale label kan betekenen. In zo'n geval wordt in F de arithmetische waarde, in A de designational waarde van de integer gevormd. Zo luidt bijvoorbeeld de vertaling van de expressie "10" in het onderhavige geval:

```
F = 10          " TSIC (10)
A = :M[ten]     " TLV (ten)
```

3. Actuele parameters

3.1. Iedere actuele parameter wordt in het objectprogramma gekenmerkt door een Actual Parameter Descriptor. Deze APD's volgen direct op de aanroep van de procedure of functie-procedure, voor elke parameter een, in de volgorde van de actuele-parameterlijst. Elke APD bestaat uit één programmawoord, dat als volgt is opgebouwd:

d0 t/m d14	een adres of een waarde
d15 t/m d17	0 (gereserveerd voor eventuele 18-bits-adressen)
d18	dynamisch-bit
d19	0
d20 t/m d25	rangnummer
d26	0

Het is de taak van de procedure, de programmalink passend op te hogen, zodat teruggekeerd wordt naar de eerste op de APD's volgende instructie.

3.2. Sommige actuele parameters zijn te gecompliceerd om uitputtend door een APD beschreven te worden. Voor zulke parameters bestaat dan naast de APD een impliciete subroutine, dit is een stuk objectprogramma, waarvan dan het (begin-) adres in d0 t/m d14 van de APD vermeld staat. Alle impliciete subroutines gaan aan de aanroep van de procedure of functie-procedure vooraf; ze staan in de volgorde van de actuele-parameterlijst. Indien er een of meer van zulke impliciete subroutines zijn, wordt de eerste op zijn beurt voorafgegaan door een "sprong over de impliciete subroutines", dat is een GOTO-instructie, leidende direct naar de aanroep van de procedure.

3.3. Actuele parameters die uitputtend beschreven worden door de APD

3.3.1. de identifier van een simpele, locale of globale variabele:

real, statisch:	$(0 \times d20 + \underline{\vert}M[x])$	" APD (x)
dynamisch:	$(0 \times d20 + d18 + \underline{\vert}Mr[q])$	
integer, statisch:	$(1 \times d20 + \underline{\vert}M[i])$	" APD (i)
dynamisch:	$(1 \times d20 + d18 + \underline{\vert}Ms[t])$	
Boolean, statisch:	$(2 \times d20 + \underline{\vert}M[b])$	" APD (b)
dynamisch:	$(2 \times d20 + d18 + \underline{\vert}Ms[t])$	
string, statisch:	$(3 \times d20 + \underline{\vert}M[st])$	" APD (st)
dynamisch:	$(3 \times d20 + d18 + \underline{\vert}Mr[q])$	

Met de operator $\underline{\vert}$ wordt aangeduid, dat op de bits d0 t/m d14 van de APD het adres van de variabele staat ingevuld, als d18 = 0 de 15 bits van een statisch geadresseerde variabele (zie 1.3.1.), als d18 = 1 de 6 + 9 bits van het dynamisch adres van een dynamisch geadresseerde variabele.

3.3.2. de identifier van een locaal of globaal array:

real, statisch:	(8 × d20 + |M[AR])	" APD (AR)
dynamisch:	(8 × d20 + d18 + ⌊Mm[n])	
integer, statisch:	(9 × d20 + |M[AR])	
dynamisch:	(9 × d20 + d18 + ⌊Mm[n])	
Boolean, statisch:	(10 × d20 + |M[AR])	
dynamisch:	(10 × d20 + d18 + ⌊Mm[n])	
string, statisch:	(11 × d20 + |M[AR])	
dynamisch:	(11 × d20 + d18 + ⌊Mm[n])	

Het adres op de bits d0 t/m d14 van de APD is het statisch of dynamisch adres van de in 1.3.3. besproken array-sleutel.

3.3.3. de identifier van een locale of globale label:

statisch:	(14 × d20 + |M[1])	" APD (1)
dynamisch:	(14 × d20 + d18 + ⌊Ms[t])	

Het adres op de bits d0 t/m d14 van de APD is het statisch of dynamisch adres van de in 2.2.1. besproken pseudo label-variabele.

3.3.4. de identifier van een locale of globale switch (zie ook 11.1.):

(12 × d20 + :Sw) " APD (Sw)

Het adres op de bits d0 t/m d14 van de APD is het statisch adres van het met de switchdeclaratie corresponderende stuk objectprogramma (zie 5.6.).

3.3.5. een formele identifier:

(15 × d20 + d18 + ⌊Mu[v]) " APD (f)

Het adres op de bits d0 t/m d14 van de APD is het dynamisch adres van de bij f behorende APIC (vgl. 6.3.2.).

3.3.6. + unsigned number:

+ real number:	(4 × d20 + |M[j])	" APD (rn)
+ grote integer:	(5 × d20 + ⌊M[i])	" APD (m)
+ kleine integer:	(7 × d20 + n)	" APD (n)
		" $0 \leq n < 32768$

Op de bits d0 t/m d14 van de APD staat een adres uit de constantenlijst, dan wel (bij de integers < 32768) de waarde van de constante zelf (vgl. 7.2.).

3.3.7. unsigned number:
De APD voor een unsigned number is die van + unsigned number, tenzij het een integer betreft, die, volgens de vertaler, op deze plaats ook de betekenis van een label zou kunnen hebben. Dit laatste geval wordt behandeld in 3.4.7.2.

3.3.8. - kleine integer: (13 × d20 + n) " APD (-n)
" 0 ≤ n < 32768

Op de bits d0 t/m d14 staat de waarde van de kleine integer.

3.3.9. een logische waarde:

true: (6 × d20 + 0) " APD (true)
false: (6 × d20 + 1) " APD (false)

3.4. Actuele parameters, beschreven door een impliciete subroutine

3.4.1. Behoudens de gevallen, behandeld in 3.4.2. t/m 3.4.6., kan de structuur van een impliciete subroutine symbolisch worden weergegeven door:

```
SUB2 (:ENTRIS)              " ENTRIS
<standaard vertaling van de als actuele
 parameter fungerende expressie>
GOTO (:EXITIS)              " EXITIS
```

waarbij ENTRIS en EXITIS routines uit het complex zijn, die de voor een goede werking vereiste administratie verzorgen.
Steeds geldt, dat de APD van de vorm (m × d20 + :ADRES) is, waarbij dan m > 15 en ADRES, op de bits d0 t/m d14 van de APD, het adres is van een instructie (en wel meestal de eerste) uit de impliciete subroutine.

3.4.2. de identifier van een (functie-) procedure:

Deze geeft aanleiding tot een impliciete subroutine van de vorm:

```
ISR:  S = MS[1]             " TFD
      A = :Pr               " JU1 (Pr)
      GOTO (:MA[1])
```

met Pr het beginadres van de vertaling van de proceduredeclaratie, terwijl de bijbehorende APD in de te onderscheiden gevallen luidt:

real procedure: (24 × d20 + :ISR)
integer procedure: (25 × d20 + :ISR)
Boolean procedure: (26 × d20 + :ISR)
string procedure: (27 × d20 + :ISR)
non type procedure: (30 × d20 + :ISR)

3.4.3. een locale of globale real, subscripted variable:
De expressie "Ar[AE1, ..., AEn]", waarbij Ar de identifier van een locaal of globaal real array is, leidt tot de volgende, samengestelde impliciete subroutine:

```
ISR:  SUB2 (:ENTRIS)        " ENTRIS
      F = AE1
      MC = F                " STACK
      F = AE2
```

```
                        MC = F                  " STACK
                        .......

                        F = AEn
statisch:               A = M[Ar]               " TAK (Ar)
                        S = :MC[-2Xn]           " EXITSV (- 2n)
                        GOTO (MS)
                LO:     SUBC (:ISR)             " SUBJ (ISR)
real:                   GOTO (:TASR)            " TASR
                L1:     S - 2                   " DECS
                        SUBC (:ISR)             " SUBJ (ISR)
                        GOTO (:FAD)             " FAD
```

terwijl de bijbehorende APD luidt:

real array element: (16 × d20 + :LO) " APD (Ar[AE1, ..., AEn])

Ter toelichting diene, dat een aanroep SUBC (:LO) de waarde van de geindiceerde variabele in F moet nemen, terwijl de aanroep SUBC (:L1) op de stapel een complete adresdescriptie, zoals benodigd voor een assignment aan de geindiceerde variabele (zie 4.1.4. en 4.1.5.), moet afleveren. De macro EXITSV bevat de aan de vertaler bekende dimensie n en stelt de routines ISR van het objectprogramma en FAD uit het complex in staat de programmalinken terug te vinden.

3.4.4. een locale of globale subscripted variable van integer, Boolean, of string type:
Hiervoor is de vertaling geheel analoog aan de in 3.4.3. besproken structuur. Het enige verschil in de impliciete subroutine is de vervanging van de macro TASR door:

```
integer:        GOTO (:TASI)        " TASI

Boolean:        GOTO (:TASB)        " TASB

string:         GOTO (:TASST)       " TASST
```

terwijl de bijbehorende APD's respectievelijk luiden:

integer array element: (17 × d20 + :LO)
Boolean array element: (18 × d20 + :LO)
string array element: (19 × d20 + :LO)

3.4.5. een element van een formeel array:
Het vertaalschema is weer zoals reeds besproken in 3.4.3., op de volgende verschilpunten na:

1) de macro DOS komt in de plaats van de macro TAK,

2) als het de vertaler bekend is, dat de formele geindiceerde variabele van Boolean of string type is, komt de macro TASB of TASST in de plaats van de macro TASR; in alle andere gevallen wordt daar de macro:

```
onbekend type:  GOTO (:TASU)        " TASU
```

gebruikt.

3) als het de vertaler bekend is, dat de formele geindiceerde variabele van Boolean of string type is, luidt de APD: (18 × d20 + :L0) of (19 × d20 + :L0); in alle andere gevallen luidt de APD:

(32 × d20 + :L0)

3.4.6. een formeel geindiceerd object, dat volgens de vertaler zowel een array element als een switch designator zou kunnen zijn:
De expressie "Fm[AE]" leidt in dit geval tot de volgende impliciete subroutine:

```
ISR:    SUB2 (:ENTRIS)          " ENTRIS
        F = AE
        DOS (Mp[q])             " DOS (Fm)
        stock3 = S              " SAS
        S = :MC[-2]             " EXITSV (-2)
        GOTO (MS)
L0:     SUBC (:ISR)             " SUBJ (ISR)
        GOTO (:TASU)            " TASU
L1:     S - 2                   " DECS
        SUBC (:ISR)             " SUBJ (ISR)
        GOTO (:FAD)             " FAD
```

waarmee als APD correspondeert:

```
        (32 × d20 + :L0)        " APD (Fm[AE])
```

Het enige verschil met 3.4.5. is de extra macro SAS; stock3 is een van de variabelen van het complex; de macro SAS redt in stock3 het adres van de door DOS uitgevoerde APIC 0-instructie ten behoeve van een eventuele aanroep van TFSL (die via TASU geactiveerd kan worden).

3.4.7. Zoals reeds aangeduid in 3.4.1. wordt in alle andere gevallen – waarbij de actuele parameter steeds een expressie is met als enige functie het afleveren van waarden – steeds een impliciete subroutine gegenereerd die het in 1. en 2. behandelde vertaalschema volgt, echter voorafgegaan door de macro ENTRIS en afgesloten door de macro EXITIS. We kunnen hier dus verder volstaan met een opsomming van de verschillende mogelijkheden voor de bijbehorende APD. Desalniettemin zullen we nog enige gevallen speciaal noemen.

3.4.7.1. designational expressie:
De APD luidt:

(28 × d20 + :ISR)

met op de bits d0 t/m d14 het beginadres van de impliciete subroutine. Een voorbeeld is de switch designator met locale of globale switch identifier. De expressie "Sw[AE]" leidt tot de impliciete subroutine:

```
ISR:    SUB2 (:ENTRIS)          " ENTRIS
        F = AE
        A = :Sw                 " TSWE (Sw)
        SUBC (:TSL)             " TSL
        GOTO (:EXITIS)          " EXITIS
```

conform de in 3.4.7. en 2.2.3.3. gegeven regels. Men vergelijke deze impliciete subroutine met de in 3.4.3. t/m 3.4.6. gegeven schema's.

3.4.7.2. expressie van integer of designational type:
Hiervan is sprake, als in een expressie van overigens onbekend gebleven type, integers voorkomen die alle mogelijk op die plaats als label geinterpreteerd moeten worden. In zulke gevallen luidt de APD:

(29 × d20 + :ISR)

met op de bits d0 t/m d14 het beginadres van de impliciete subroutine. Een voorbeeld is het reeds in 3.3.7. genoemde geval van een integer als actuele parameter met mogelijke betekenis van label. In dat geval luidt de impliciete subroutine voor b.v. "32768" :

```
ISR:    SUB2 (:ENTRIS)          " ENTRIS
        F = M[n]                " TIC (32768)
        A = :Mp[q]              " TLV (32768)
        GOTO (:EXITIS)          " EXITIS
```

conform de in 3.4.7. en 2.3.2. gegeven regels.

3.4.7.3. (niet-assigneerbare) expressie van arithmetisch, Boolean of string type:
In deze gevallen luidt de APD respectievelijk:

arithmetisch:	(20 × d20 + :ISR)
Boolean:	(22 × d20 + :ISR)
string:	(23 × d20 + :ISR)

met op de bits d0 t/m d14 steeds het beginadres van de impliciete subroutine. Als voorbeeld geven we de impliciete subroutine van de arithmetische expressie "Ar[AE1, ..., AEn] + 1", waarbij Ar de (statisch geadresseerde) identifier van een real array is:

```
ISR:    SUB2 (:ENTRIS)          " ENTRIS
        F = AE1
        MC =  F                 " STACK
        .......
        F = AEn
        A = M[Ar]               " TAK (Ar)
        SUB (:IND)              " TSR
        F =  MA
        F +   1                 " ADDSIC (1)
        GOTO (:EXITIS)          " EXITIS
```

In de eerste opdrachten is deze impliciete subroutine gelijkluidend met de in 3.4.3. behandelde vertaling van de expressie "Ar[AE1, ..., AEn]". Daar nu geen assignment aan de actuele parameter kan geschieden is de afwerking echter geheel verschillend en komt zij hier overeen met de in 1.3.3.2. gegeven regels.

3.4.7.4. (niet-assigneerbare) expressie van onbekend type:
De APD luidt:

(31 × d20 + :ISR)

met op de bits d0 t/m d14 het beginadres van de impliciete subroutine.

3.5. Tot besluit dient aan de bovenstaande regels nog iets toegevoegd te worden over de gang van het vertaalproces zelf. Bij het vertalen van een actuele-parameterlijst zijn voor de behandeling van iedere actuele parameter twee fasen te onderscheiden:
a) eerst wordt, onder gelijktijdige constructie van de APD, de actuele parameter vertaald zonder ooit gebruik te maken van eventueel aanwezige informatie over de corresponderende formele parameter,
b) na de vertaling van de actuele parameter wordt onderzocht, of die actuele parameter voldoet aan de eisen, die door de ermee corresponderende formele parameter gesteld worden.
Als de actuele-parameterlijst echter behoort bij de aanroep van een formele procedure, is de voor stap b benodigde informatie niet beschikbaar en wordt die stap onderdrukt.
Een gevolg van bovengenoemd vertaalprocédé is, dat in enkele bijzondere gevallen niet de optimale vertaling bereikt wordt. Zo wordt in het geval van de unsigned integer, die, als actuele parameter optredend, ook als label geinterpreteerd kan worden, steeds het in 3.4.7.2. behandelde schema gevolgd, ook als van de formele parameter uit de procedure body bekend is, dat hij van arithmetisch type moet zijn.

3.6. voorbeelden:

"SUM (i, 1, n, 1/i)" :

```
        GOTO (:L0)
ISR:    SUB2 (:ENTRIS); F = 1; G / i; GOTO (:EXITIS)
L0:     SUBC (:SUM)
        (1 × d20 + |i);(7 × d20 + 1);(1 × d20 + |n);(20 × d20 + :ISR)
```

"REMAINDER (-1987635, (24))" :

```
        GOTO (:L0)
ISR1:   SUB2 (:ENTRIS); G = - M[t]; GOTO (:EXITIS)
ISR2:   SUB2 (:ENTRIS); F = 24; GOTO (:EXITIS)
L0:     SUBC (:REMAINDER)
        (20 × d20 + :ISR1);(20 × d20 + :ISR2)
```

"ACKERMANN (m - 1, ACKERMANN (m, n - 1))" :

```
        GOTO (:L1)
ISR1:   SUB2 (:ENTRIS); G = Mp[m]; F - 1; GOTO (:EXITIS)
ISR2:   SUB2 (:ENTRIS); GOTO (:L0)
            ISR3:   SUB2 (:ENTRIS); G = Mp[n]; F - 1; GOTO (:EXITIS)
            L0:     SUBC (:ACKERMANN)
                    (1 × d20 + d18 + |m);(20 × d20 + :ISR3)
        GOTO (:EXITIS)
L1:     SUBC (:ACKERMANN)
        (20 × d20 + :ISR1);(20 × d20 + :ISR2)
```

4. Statements

4.1. De vertaling van assignment statements

4.1.1. De algemene structuur van een assignment statement kan symbolisch worden weergegeven met:

"L1:= L2:= ...:= Ln:= E"

Met een dergelijke statement correspondeert een stuk objectprogramma volgens het schema:

```
<preparatie assignment aan L1>
<preparatie assignment aan L2>
.............................
<preparatie assignment aan Ln>
<evaluatie van E>
<assignment aan Ln>
.....................
<assignment aan L2>
<assignment aan L1>
```

Voor alle assignment-operaties geldt, dat zij de inhoud van het relevante register (de waarde van de expressie) niet veranderen, behoudens het feit, dat bij de assignment aan een integer-variabele afronding van de waarde van F tot een integer plaats heeft.

4.1.2. assignments aan een locale of globale simpele variabele:
De preparatie voor de assignment aan een locale of globale simpele variabele is leeg. De assignment-operatie luidt:

```
statisch, real:          M[x] = F                    " STR (x)

dynamisch, real:         Mr[q] = F

statisch, integer:       S =   F, Z                  " STI (i)
                      N, SUBC (:RND)
                         M[i] = G

dynamisch, integer:      S =   F, Z
                      N, SUBC (:RND)
                         Ms[t] = G

statisch, Boolean:       S =   T                     " STB (b)
                         M[b] = S

dynamisch, Boolean:      S =   T
                         Ms[t] = S

statisch, string:        F = :M[st]                  " STST (st)
                         SUB2 (:STST)

dynamisch, string:       F = :Mr[q]
                         SUB2 (:STST)
```

Bij de assignment aan een variabele van het type integer wordt bovenstaande macro STI gekozen, tenzij, bij een meervoudige assignment, in de rij van assignment-operaties reeds eerder expliciete afronding door de macro STI of door de nog te bespreken macro STSI (zie 4.1.5.) heeft plaats gehad. In dat geval komt in plaats van de macro STI de macro:

statisch, integer, zonder afronding:

```
M[i] = G                    " SSTI (i)
```

dynamisch, integer, zonder afronding:

```
Ms[t] = G
```

4.1.3. assignments aan een procedure identifier:
Aangezien de assignment aan een procedure identifier in feite geschiedt aan een door de vertaler toegevoegde (zie 6.4. en 8.3.) locale simpele variabele van de procedure body, verloopt die assignment geheel volgens 4.1.2. (met echter steeds de dynamische variant).

4.1.4. assignments aan een formele identifier:
De preparatie voor een assignment aan een formele identifier f (met toegevoegd adres [p,q]) bestaat uit de macro:

```
DOS (Mp[q+2])               " DOS2 (f)
```

waarmee een instructie uit de stapel, de z.g. APIC 2-instructie, wordt uitgevoerd. Deze instructie heeft, als de met f corresponderende actuele parameter (uiteindelijk) een simpele variabele is, geen effect; als de actuele parameter een subscripted variable is, is de APIC 2-instructie een stapelende subroutine-aanroep SUBC (:L1) als beschreven in 3.4.3. (zie ook appendix D).
De assignment operatie geschiedt bij de formele identifier f door de macro:

```
DOS (Mp[q+3])               " DOS3 (f)
```

waarmee de z.g. APIC 3-instructie wordt uitgevoerd. Ook deze is afhankelijk van de aard van de met f corresponderende actuele parameter.

4.1.5. assignments aan een subscripted variable:
De preparatie voor een assignment aan een geindiceerde variabele (element van een locaal, globaal of formeel array) bestaat uit het in 1.3.3.1. respectievelijk 1.3.4. beschreven stuk objectprogramma voor het maken van een adresdescriptie, gecomplementeerd door de macro's STACK en STAA. Het volledige schema voor de preparatie van een assignment aan de variabele "Ar[AE1, ..., AEn]", element van een real, statisch geadresseerd array Ar luidt:

```
F = AE1
MC = F                      " STACK
.......

F = AEn
A = M[Ar]                   " TAK (Ar)
MC = F                      " STACK
MC = A                      " STAA
```

(zie ook 11.3.).

De assignment-operatie geschiedt bij de geindiceerde variabele door een van de volgende macro's:

element van locaal of globaal real array:

SUB2 (:STSR) " STSR

element van locaal of globaal integer array:

SUB2 (:STSI) " STSI

element van locaal of globaal Boolean array:

SUB2 (:STSB) " STSB

element van locaal of globaal string array:

SUB2 (:STSST) " STSST

element van een formeel array:
Voorzover het type aan de vertaler bekend is, en hetzij Boolean, hetzij string is, wordt de bovengenoemde macro STSB dan wel STSST gegenereerd; in alle andere gevallen verschijnt de macro:

SUB2 (:STFSU) " STFSU

Al deze macro's breken de door de preparatie gevormde adresdescriptie in de stapel af.
Indien, bij een meervoudige assignment aan een rij variabelen van integer type, in de rij van assignment-operaties reeds eerder expliciete afronding door een van de macro's STI (zie 4.1.2.) of STSI heeft plaats gehad, wordt de macro STSI vervangen door:

SUB2 (:SSTSI) " SSTSI

die niet opnieuw onderzoekt of afronding noodzakelijk is.

4.1.6. voorbeelden:

"i:= j:= 0" :

F = 0; S = F, Z; N, SUBC (:RND); j = G; i = G;

"Ar[i]:= x" :

G = i; A = M[Ar]; MC = F; MC = A; F = x; SUB2 (:STSR)

"f:= f + 1" :

DOS (f[2]); DOS (f); F + 1; DOS (f[3])

4.2. De vertaling van goto statements

Voor goto statements bestaan twee vertaalschema's:
a) de statement "goto <locale label identifier>" wordt vertaald met:

```
GOTO (:L)                " JU (L)
```

waarbij L het (statische) beginadres is van het stuk objectprogramma dat de vertaling vormt van de door de locale label identifier gelabelde statement,
b) alle andere goto statements worden vertaald door een stuk programma dat de <designational expression> evalueert volgens de in 2.2. gegeven regels, gevolgd door de macro JUA. Symbolisch kunnen we dit weergeven met:

```
A = DE
GOTO (:JUA)              " JUA
```

De routine JUA uit het complex is in staat, zo nodig, alle administratieve handelingen te verrichten die behoren bij blokverlating.

4.3. De vertaling van dummy statements

De vertaling van een dummy statement is leeg.

4.4. De vertaling van procedure statements (zie ook 10.)

De vertaling van een procedure statement bestaat uit een aanroep van de procedure, direct gevolgd door een (eventueel lege) lijst van APD's en zonodig voorafgegaan door een sprong en een lijstje impliciete subroutines, geheel volgens de beschrijving van 3. De aanroep van de procedure bestaat uit hetzij:

```
SUBC (:P)                " SUBJ (P)
```

waarbij P het (statische) beginadres van de vertaling van de procedure body aanduidt (zie 6.), hetzij:

```
DOS (Mu[v])              " DOS (fP)
```

als het de aanroep van een met [u,v] geadresseerde formele procedure fP betreft. Bovengenoemde aanroep van de procedure, inclusief impliciete subroutines en APD's, wordt bovendien nog gevolgd door de macro:

```
SUBC (:REJST)            " REJST
```

indien de procedure identifier een formele parameter is dan wel de identifier van een locale of globale stringprocedure is.

4.5. De vertaling van gelabelde statements

De vertaling van een gelabelde statement is identiek met de vertaling van de statement, verkregen door het weglaten van die label (en de er op volgende colon). De enige opdrachten uit het objectprogramma die een gevolg zijn van het voorkomen van labels in een blok kunnen voorkomen bij de vertaling van de blokingang. Zij dienen dan om de in 2.2.1. genoemde pseudo label-variabelen in te voeren en aan deze een

waarde toe te kennen (zie 5.3. en 6.5.).

4.6. De vertaling van compound statements

De vertaling van een compound statement is de opeenvolging van de vertaling van de statements die successievelijk de compound statement vormen.

4.7. De vertaling van conditional statements

De vertaling van de conditional statement "if BE then ST" luidt symbolisch:

```
            C =  BE
        N, GOTO (:L0)          " COJU (L0)
           ST
    L0:
```

De vertaling van de conditional statement "if BE then ST1 else ST2" luidt symbolisch:

```
            C =  BE
        N, GOTO (:L0)          " COJU (L0)
           ST1
           GOTO (:L1)          " JU (L1)
    L0:    ST2
    L1:
```

4.8. De vertaling van for statements

4.8.1. In het geval van een simpele controlled variable v luidt het vertaalschema voor de for statement
"for v:= FLE1, FLE2, ..., FLEn do ST" symbolisch:

```
                <vertaalschema voor FLE1>
                <vertaalschema voor FLE2>
                ........................

                <vertaalschema voor FLEn>
                GOTO (:FINISH)              " JU (FINISH)
    ACTION:     ST
                GOTO (for var)              " IJU (for var)
    FINISH:
```

Aan iedere for statement voegt de vertaler een anonieme pseudo variabele toe. Deze beslaat in het geheugen 1 woord. In het bovenstaande schema is de pseudo variabele "for var" genoemd. De waarde van for var is steeds een adres uit het objectprogramma. Op dit door for var aangewezen adres wordt, iedere keer na het uitvoeren van de statement ST, de behandeling van de for list voortgezet.
De pseudo forvariabelen zijn locaal t.o.v. het kleinste, de for statement omvattende blok, en worden statisch of dynamisch geadresseerd volgens de in 1.3.1. gegeven regels. Een blok heeft precies zoveel verschillende pseudo forvariabelen als nodig is om bij de diepste in het blok optredende nesting van for statements iedere for clause zijn eigen pseudo forvariabele te geven.

4.8.2. In het geval van een geindiceerde controlled variable "Ar[AE1, ..., AEn]" wordt het in 4.8.1. gegeven vertaalschema voorafgegaan door een stuk objectprogramma van de volgende structuur:

```
                    GOTO (:FIRST ELEMENT)   " JU (FIRST ELEMENT)
ISR:                F = AE1
                    MC = F                  " STACK
                    F = AE2
                    MC = F                  " STACK
                    .......
                    F = AEn
statisch:           A = M[Ar]               " TAK (Ar)
                    S = :MC[-2×n+1]         " EXITSV (-2n + 1)
                    GOTO (MS)
VALUE:              SUBC (:ISR)             " SUBJ (ISR)
real:               GOTO (:TRSCV)           " TRSCV
ADDRESS:            SUBC (:ISR)             " SUBJ (ISR)
                    GOTO (:FADCV)           " FADCV
FIRST ELEMENT:
```

Dit schema is geheel analoog aan de in 3.4.3. besproken impliciete subroutine voor de actuele parameter "Ar[AE1, ..., AEn]". De macro EXITSV bevat ook hier de aan de vertaler bekende dimensie n. Een aanroep SUBC (:VALUE) levert in F de waarde van de controlled variable (benodigd in een step-until-element), een aanroep SUBC (:ADDRESS) levert op de stapel een complete adresdescriptie, zoals benodigd voor een assignment aan een geindiceerde variabele (vgl. 4.1.5.). In het bovenstaande schema dient voor een locaal of globaal integer array de macro TRSCV vervangen te worden door:

```
                    GOTO (:TISCV)           " TISCV
```

Voor een formeel array moet in plaats van TAK de macro DOS het adres van de reference-vector in A vormen, terwijl de macro TRSCV vervangen moet worden door:

```
                    GOTO (:TSCVU)           " TSCVU
```

4.8.3. Het vertaalschema voor een for list element bestaande uit een arithmetische expressie "AE" luidt:

```
                    F = :NEXT ELEMENT       " TSIC (NEXT ELEMENT)
                    for var = G             " SSTI (for var)
                    <preparatie assignment aan v>
                    F = AE
                    <assignment aan v>
                    GOTO (:ACTION)          " JU (ACTION)
NEXT ELEMENT:
```

Hierin staat for var voor de in 4.8.1. genoemde pseudo forvariabele, v voor de controlled variable, terwijl ACTION de eveneens in 4.8.1. gegeven gelijknamige label representeert. Het hier besproken type for list element wijst zelf expliciet zijn opvolger aan. Afhankelijk van de aard van de controlled variable worden de preparatie- en assignment-macro's:

4.8.3.1. de controlled variable is een simpele, locale of globale variabele:
De preparatie voor de assignment aan v is in dit geval leeg; de assignment geschiedt, conform 4.1.2. door:

```
statisch, real:          M[x] = F                    " STR (x)

dynamisch, real:         Mr[q] = F

statisch, integer:       S =   F, Z                  " STI (i)
                       N, SUBC (:RND)
                         M[i] = G

dynamisch, integer:      S =   F, Z
                       N, SUBC (:RND)
                         Ms[t] = G
```

4.8.3.2. de controlled variable is een formele identifier:
De preparatie voor de assignment aan v bestaat in dit geval uit de macro:

```
                         DOS (Mp[q+2])               " DOS2 (f)
```

de assignment geschiedt door de macro:

```
                         DOS (Mp[q+3])               " DOS3 (f)
```

beide geheel in overeenstemming met 4.1.4.

4.8.3.3. de controlled variable is een geindiceerde variabele:
De preparatie voor de assignment aan v geschiedt voor een geindiceerde variabele door de macro:

```
                         SUBC (:ADDRESS)             " SUBJ (ADDRESS)
```

waarbij ADDRESS de met dezelfde naam gelabelde opdracht aanduidt van het in 4.8.2. besproken stuk objectprogramma, dat aan de vertaling van de for list elementen voorafgaat als de controlled variable geindiceerd is.
De assignment aan v geschiedt door een van de macroparen:

```
                         SUB2 (:STSR)                " STSR
                         B -  2                      " DECB (2)

                         SUB2 (:STSI)                " STSI
                         B -  2                      " DECB (2)

                         SUB2 (:STFSU)               " STFSU
                         B -  2                      " DECB (2)
```

al naar gelang de array identifier in de controlled variable behoort bij een locaal of globaal array van type real of van type integer, dan wel een formele identifier is.

4.8.4. Het vertaalschema voor een for list element van de vorm "AE <u>while</u> BE" luidt symbolisch:

```
                F = :NEXT TRIAL          " TSIC (NEXT TRIAL)
                for var = G              " SSTI (for var)
NEXT TRIAL:     <preparatie assignment aan v>
                F =  AE
                <assignment aan v>
                C =  BE
              Y, GOTO (:ACTION)          " YCOJU (ACTION)
NEXT ELEMENT:
```

In het hier behandelde geval wordt door de pseudo forvariabele for var niet het volgende for list element aangewezen, maar het for list element in kwestie zelf. Overgang naar het volgende element van de for list vindt vanzelf plaats zodra de waarde van "BE" false blijkt te zijn. Voor de preparatie van de assignment en de assignment-operatie kunnen we verwijzen naar 4.8.3.1. t/m 4.8.3.3.

4.8.5. Het vertaalschema voor een for list element van de vorm "AE1 step AE2 until AE3" luidt in zijn meest algemene vorm:

```
                F = :STEP UP             " TSIC (STEP UP)
                for var = G              " SSTI (for var)
                <preparatie assignment aan v>
                F = AE1
                GOTO (:ASSIGN)           " JU (ASSIGN)
STEP:           F = AE2
                GOTOR (MC[-1])           " EXIT
STEP UP:        <preparatie assignment aan v>
                <evaluatie van v>
                MC =  F                  " STACK
                SUBC (:STEP)             " SUBJ (STEP)
                F + MC[-2]               " ADD
ASSIGN:         <assignment aan v>
                <evaluatie van v>
                MC =  F                  " STACK
                F = AE3
                F - MC[-2], Z            " TEST1
                SUBC (:STEP)             " SUBJ (STEP)
              N, F =   F, E              " TEST2
              N, F =   F, Z
              Y, GOTO (:ACTION)          " YCOJU (ACTION)
NEXT ELEMENT:
```

Evenals bij het while-element wijst de pseudo forvariabele bij een step-until-element naar een plaats in het for list element zelf, terwijl overgang naar het volgende element van de for list vanzelf plaats heeft, zodra niet langer voldaan is aan:

$$dif = 0 \vee step = 0 \vee sign\ (dif) = sign\ (step)$$

met $dif = -\ (v - AE3)$ en $step = AE2$
(hetgeen ekwivalent is met $(v - AE3) \times sign\ (AE2) < 0$).
Voor de preparatie van de assignment en voor de assignment-operatie kunnen we verwijzen naar 4.8.3.1. t/m 4.8.3.3. De evaluatie van v geschiedt door:

```
statisch, real:          F = M[x]        " TRV (x)
dynamisch, real:         F = Mr[q]
```

```
statisch, integer:      G = M[i]                    " TIV (i)
dynamisch, integer:     G = Ms[t]
```

als v een locale of globale simpele real of integer variabele is, door:

```
                        DOS (Mp[q])                 " DOS (f)
```

als v een formele identifier is, en door:

```
                        SUBC (:VALUE)               " SUBJ (VALUE)
```

als v een geindiceerde variabele is. In dit laatste geval wordt met VALUE de gelijknamige label aangeduid in het in 4.8.2. besproken stuk objectprogramma, dat aan de vertaling van de for list elementen voorafgaat als de controlled variable geindiceerd is.

4.8.6. Vereenvoudigingen bij de vertaling van het step-until-element:

4.8.6.1. Als de step-expressie AE2 uit 4.8.5. bestaat uit een van de primary's: <simpele locale of globale variabele> | <unsigned number>, dan wel bestaat uit een van deze primary's, voorafgegaan door een teken, luidt de vertaling van het step-until-element "AE1 step AE2 until AE3" :

```
                        F = :STEP UP                " TSIC (STEP UP)
                        for var = G                 " SSTI (for var)
                        <preparatie assignment aan v>
                        F = AE1
                        GOTO (:ASSIGN)              " JU (ASSIGN)
STEP:                   F = AE2
STEP UP:                <preparatie assignment aan v>
                        <evaluatie van v>
                        MC = F                      " STACK
                        DO (STEP)                   " DO (STEP)
                        F + MC[-2]                  " ADD
ASSIGN:                 <assignment aan v>
                        <evaluatie van v>
                        MC = F                      " STACK
                        F = AE3
                        F - MC[-2], Z               " TEST1
                        DO (STEP)                   " DO (STEP)
                    N, F =  F, E                    " TEST2
                    N, F =  F, Z
                    Y, GOTO (:ACTION)               " YCOJU (ACTION)
NEXT ELEMENT:
```

4.8.6.2. Als de controlled variable v een locale of globale simpele real of integer variabele is, kan de evaluatie van v direct na de assignment aan v achterwege blijven. De vertaling van het step-until-element "AE1 step AE2 until AE3" luidt dan:

```
                        F = :STEP UP                " TSIC (STEP UP)
                        for var = G                 " SSTI (for var)
                        F = AE1
                        GOTO (:ASSIGN)              " JU (ASSIGN)
```

```
STEP:       F = AE2
            GOTOR (MC[-1])          " EXIT
STEP UP:    <evaluatie van v>
            MC = F                  " STACK
            SUBC (:STEP)            " SUBJ (STEP)
            F + MC[-2]              " ADD
ASSIGN:     <assignment aan v>
            MC = F                  " STACK
            F = AE3
            F - MC[-2], Z           " TEST1
            SUBC (:STEP)            " SUBJ (STEP)
          N, F = F, E               " TEST2
          N, F = F, Z
          Y, GOTO (:ACTION)         " YCOJU (ACTION)
NEXT ELEMENT:
```

4.8.6.3. De in 4.8.6.1. en in 4.8.6.2. besproken vereenvoudigingen zijn onafhankelijk van elkaar.

4.9. De vertaling van blokken wordt gegeven in 5.

4.10. voorbeelden:

"for i:= 1 step 1 until j - 1, j + 1 step 1 until n do Ar[i]:= 0" :

```
            F = :L2; for var = G; F = 1; GOTO (:L3)
L1:         F = 1
L2:         G = i; MC = F; DO (L1); F + MC[-2]
L3:         S = F, Z; N, SUBC (:RND); i = G; MC = F
            G = j; F - 1; F - MC[-2], Z; DO (L1); N, F = F, E; N, F = F, Z
            Y, GOTO (:ACTION)
            F = :L5; for var = G; G = j; F + 1; GOTO (:L6)
L4:         F = 1
L5:         G = i; MC = F; DO (L4); F + MC[-2]
L6:         S = F, Z; N, SUBC (:RND); i = G; MC = F
            G = n; F - MC[-2], Z; DO (L4); N, F = F, E; N, F = F, Z
            Y, GOTO (:ACTION)
            GOTO (:FINISH)
ACTION:     G = i; A = M[Ar]; MC = F; MC = A; F = 0; SUB2 (:STSR)
            GOTO (for var)
FINISH:
```

"for i:= 1 step 1 until n do if i ≠ j then Ar[i]:= 0" :

```
            F = :L2; for var = G; F = 1; GOTO (:L3)
L1:         F = 1
L2:         G = i; MC = F; DO (L1); F + MC[-2]
L3:         S = F, Z; N, SUBC (:RND); i = G; MC = F
            G = n; F - MC[-2], Z; DO (L1); N, F = F, E; N, F = F, Z
            Y, GOTO (:ACTION)
            GOTO (:FINISH)
ACTION:     G = i; G - j, Z; S = - T, P; N, GOTO (:L4)
            G = i; A = M[Ar]; MC = F; MC = A; F = 0; SUB2 (:STSR)
L4:         GOTO (for var)
FINISH:
```

5. Blokken

5.1. De vertaling van ieder blok dat niet een procedure body vormt begint met de standaard blokingang. Op de standaard blokingang volgen, in de opgegeven volgorde, de labellijst, de vertaling van de declaraties van het blok, de afsluiting van de vertaling van de declaraties, de vertaling van de statements van het blok en de standaard blokuitgang. Deze onderdelen worden in de volgende secties successievelijk behandeld.

5.2. De standaard blokingang

De standaard blokingang, die het begin vormt van de vertaling van een blok dat geen procedure body vormt, luidt:

```
A = :MC                          " TBL (display level)
F = :MD[display level]
B + [local space]                " ENTRB (local space)
SUB2 (:ENTRB)
```

Hierin is display level een getal, dat door de vertaler aan ieder blok uit het programma (en ook aan iedere procedure body) wordt toegekend volgens de volgende regels:
1) om het programma heen wordt een extra blok gedacht, dit buitenbuitenste blok krijgt 0 als display level,
2) ieder blok in het programma (en ook iedere procedure body) heeft een display level, dat 1 hoger is dan dat van het kleinste, dat blok omvattende blok of procedure body.

De local space uit de standaard blokingang is een getal, dat door de vertaler aan ieder blok uit het programma wordt toegekend. Voor een blok dat geen procedure body is geschiedt dit volgens de volgende regels:
1) als het blok niet door enige procedure van het programma omvat wordt, is de local space 1,
2) als het blok binnenblok is van een procedure, is de local space:

1 + 2 × het aantal locale, simpele, niet-own real variabelen
  + 1 × het aantal locale, simpele, niet-own integer variabelen
  + 1 × het aantal locale, simpele, niet-own Boolean variabelen
  + 2 × het aantal locale, simpele, niet-own stringvariabelen
  + 1 × het aantal locale, niet-own array's
  + 1 × het aantal locale pseudo forvariabelen
  + 2 × het aantal locale, maar niet-superlocale labels

Het aantal pseudo forvariabelen van een blok is gelijk aan het maximum aantal forclauses in dat blok, dat een statement van dat blok omvat (vgl. 4.8.1.).
Een label van een blok wordt superlocaal genoemd, als het een <identifier> label betreft, die niet op andere wijze gebruikt wordt dan in de combinatie "goto <label identifier>", staande in dat blok zelve (vgl. 4.2.).

5.3. De labellijst

Indien een blok dat binnenblok van een procedure is n locale maar niet-superlocale labels L1, L2, ..., Ln heeft, volgt, direct op de standaard blokingang, een stuk objectprogramma van de vorm:

```
                B = [2×n]              " DECB (2 × n)
                S = [display level]    " LAD (display level)
                SUB (:LAD)
LABEL LIST:     :L1                    " NIL (L1)
                :L2                    " NIL (L2)
                ...
                = (d21 + :Ln)          " LAST (Ln)
```

waarbij geldt, dat in het geval n = 1, dus als in het blok slechts 1 niet-superlocale label L voorkomt, achter LABEL LIST slechts de macro LAST (L) volgt. De adressen in de instructies van de macro's NIL en LAST zijn de beginadressen van de stukken objectprogramma die corresponderen met de vertaling van de met die labels gelabelde statements.

5.4. De vertaling van type declaraties

Type declaraties geven geen aanleiding tot afzonderlijke instructies in het objectprogramma. De geheugenruimte voor simpele locale variabelen wordt hetzij door de vertaler gereserveerd in de ruimte van de statisch geadresseerde objecten (zie 7.4.), hetzij tijdens uitvoering van het programma bij de standaard blokingang gereserveerd (vgl. 5.2.).

5.5. De vertaling van arraydeclaraties

De vertaling van de declaratie "array Ar1, ..., Arm[AE11 : AE21, ..., AE1n : AE2n]" luidt symbolisch:

```
                F = AE11
                MC = F                 " STACK
                F = AE21
                MC = F                 " STACK
                ........
                F = AE1n
                MC = F                 " STACK
                F = AE2n
                MC = F                 " STACK
                F = [m]                " TNA (m)
                S = [n]                " TDA (n)
statisch:       A = :M[Ar1]            " TAA (Ar1)
                SUB3 (:RAD)            " RAD
```

Hierin is m het aantal array's met dezelfde grenzen, n de dimensie, terwijl door de macro TAA het adres van de array-reference van het eerste array in A genomen wordt.
De macro RAD moet, voor declaraties van array's van ander type, vervangen worden door:

```
own real array:          SUB3 (:ORAD)          " ORAD
integer array:           SUB3 (:IAD)           " IAD
own integer array:       SUB3 (:OIAD)          " OIAD
Boolean array:           SUB3 (:BAD)           " BAD
own Boolean array:       SUB3 (:OBAD)          " OBAD
string array:            SUB3 (:STAD)          " STAD
own string array:        SUB3 (:OSTAD)         " OSTAD
```

Tenslotte wordt voor elk <array segment> in de <array list> van een <array declaration> bovenstaand vertaalschema gevolgd; de vertalingen komen direct na elkaar in het objectprogramma te staan, in de volgorde van de <array list>.

5.6. De vertaling van switchdeclaraties

5.6.1. Indien een switchdeclaratie niet onmiddellijk voorafgegaan wordt door hetzij een andere switchdeclaratie, hetzij een proceduredeclaratie, wordt de vertaling van die switchdeclaratie voorafgegaan door de macro:

```
GOTO (:L)                    " JU (L)
```

waarbij L wijst naar de instructie van het objectprogramma die het begin vormt van de vertaling van de eerstvolgende arraydeclaratie, of, zo op de switchdeclaratie geen arraydeclaraties meer volgen, naar de instructie die de vertaling van de declaraties van het blok afsluit (zie 5.8.).
Als een switchdeclaratie onmiddellijk voorafgegaan wordt door een switch- of proceduredeclaratie, komt bovenstaande macro te vervallen.

5.6.2. Het symbolische vertaalschema voor de switchdeclaratie "switch Sw:= DE1, DE2, ..., DEn" luidt, afgezien van nader in 5.6.3. te bespreken complicaties, als volgt:

```
Sw:     + [n]                                  " CODE (n)
        <instructie ter berekening van DE1>    " SWORD (DE1)
        <instructie ter berekening van DE2>    " SWORD (DE2)
        ..................................
        <instructie ter berekening van DEn>    " SWORD (DEn)
```

dwz., het eerste woord bevat het aantal entries uit de switch list; onmiddellijk hierop volgen even zoveel instructies, voor iedere entry van de switch list een. Deze instructies worden SWORD's genoemd. Elk hiervan levert bij uitvoering de waarde van de ermee corresponderende designational expressie in A af. Het beginadres van bovenstaand vertaalschema in het objectprogramma is het adres, dat eerder genoemd is in 2.2.3.3. als parameter van de macro TSWE, en in 3.3.4. als adres op de bits d0 t/m d14 van een met de switch identifier corresponderende APD.

5.6.3. Indien een entry uit de switch list niet uitsluitend gevormd wordt door de identifier van een locale of globale label of door een formele identifier, beschouwt de vertaler deze entry als te ingewikkeld om door een SWORD-instructie uitputtend beschreven te worden; in dat geval last de vertaler, voorafgaande aan het in 5.6.2. gegeven vertaal-

schema (maar wel volgend op de eventueel ingelaste macro JU als in 5.6.1. beschreven), ten behoeve van die entry een impliciete subroutine in, die de <designational expression> evalueert volgens de in 2.2. gegeven regels en afgesloten wordt door de macro EXIT. Symbolisch kunnen we dit weergeven met:

```
ISR:    A = DE
        GOTOR (MC[-1])          " EXIT
```

Een aantal van zulke impliciete subroutines kunnen zo de eigenlijke vertaling van de switchdeclaratie voorafgaan; ze staan zelf in volgorde van de switch list. Het is hier (in tegenstelling tot het geval van impliciete subroutines voor actuele parameters) uiteraard niet nodig de collectie impliciete subroutines weer te laten voorafgaan door een macro JU, al zal deze er soms, maar dan om andere redenen (zie 5.6.1.), wel staan.

5.6.4. Er zijn drie verschillende typen SWORD's:
Voor een entry uit de switch list die een locale of globale label l is, luidt de SWORD:

```
statisch:       A = :M[l]           " TLV (l)
dynamisch:      A = :Ms[t]
```

conform 2.2.3.1.

Voor een entry uit de switch list die een met [u,v] geadresseerde formele label fl is, luidt de SWORD:

```
                DOS (Mu[v])         " DOS (fl)
```

conform 2.2.3.2.

Voor alle andere entries uit de switch list luidt de SWORD:

```
                GOTO (:ISR)         " JU (ISR)
```

waarbij met ISR het beginadres van de, in 5.6.3. besproken, met die entry corresponderende impliciete subroutine is aangeduid.

5.6.5. Alle bij de vertaling van een switchdeclaratie voorkomende dynamische adressen zijn steeds van de vorm:

$$Mp[q] \qquad \text{met } 2 \leq p < 58$$

en derhalve nooit van de vorm MD[q] (vgl. 8.2.).

5.7. De vertaling van proceduredeclaraties

Aan de vertaling van proceduredeclaraties is een afzonderlijk hoofdstuk (6.) gewijd. We vermelden hier alleen, dat, net zoals bij de switchdeclaratie (zie 5.6.1.), de vertaling van een proceduredeclaratie voorafgegaan wordt door de macro:

```
                GOTO (:L)           " JU (L)
```

tenzij die proceduredeclaratie onmiddellijk wordt voorafgegaan door hetzij een switchdeclaratie, hetzij een andere proceduredeclaratie. Het adres L wijst weer naar de instructie, die het begin vormt van de vertaling van de eerstvolgende arraydeclaratie, of, zo op de proceduredeclaratie geen arraydeclaraties meer volgen, naar de instructie die de vertaling van de declaraties van het blok afsluit.

5.8. De afsluiting van de vertaling van de declaraties

Indien onder de declaraties van het blok arraydeclaraties voorkomen, luidt de afsluiting van de vertaling van de declaraties:

```
MO[512×display level] = B        " SWP (display level)
```

Anders is de afsluiting leeg en volgt op de vertaling van de declaraties van het blok onmiddellijk de vertaling van de eerste statement.

5.9. De vertaling van de statements van het blok

Deze is in hoofdstuk 4. reeds behandeld.

5.10. De standaard blokuitgang (zie ook 11.3.)

Op de vertaling van de laatste statement van het blok volgt de standaard blokuitgang. Deze bestaat in twee versies. De eerste hiervan luidt:

```
B = MD[display level]            " EXITB (display level)
```

en wordt geproduceerd als in het blok geen locale simpele of geindiceerde stringvariabelen gedeclareerd worden. Als dit wel gebeurt, luidt de standaard blokuitgang:

```
B = MD[display level]            " EXITC (display level)
SUBC (:FREE CHAIN)
```

5.11. voorbeeld:

```
"begin  integer  i; array  Ar[1 : 10], Eps[1 : 2];
        switch  Sw:= if i > 0 then  AA else  BB, CC;
        goto  Sw[1];
 AA: BB: CC:  goto  DD;
 DD:
 end"
```

Als het blok geen binnenblok van enige procedure is, luidt de vertaling:

```
A = :MC; F = :MD[3]; B + 1; SUB2 (:ENTRB)
F = 1; MC = F; F = 10; MC = F; F = 1; S = 1; A = :M[Ar];
        SUB3 (:RAD)
F = 1; MC = F; F =  2; MC = F; F = 1; S = 1; A = :M[Eps];
        SUB3 (:RAD)
GOTO (:L2)
```

```
ISR:     G = 1; G - 0, P; N, GOTO (:LO)
         A = :M[vAA]; GOTO (:L1)
LO:      A = :M[vBB]
L1:      GOTOR (MC[-1])
Sw:      + 2; GOTO (:ISR); A = :M[vCC]
L2:      M3 = B
         F = 1; A = :Sw; SUBC (:TSL); GOTO (:JUA)
AA:BB:CC: GOTO (:DD)
DD:      B = MD[3]
```

Als het blok binnenblok van een procedure is, luidt de vertaling:

```
         A = :MC; F = :MD[3]; B + 10; SUB2 (:ENTRB)
         B - 6; S = 3; SUB (:LAD); :AA; :BB; - (d21 + :CC)
         F = 1; MC = F; F = 10; MC = F; F = 1; S = 1; A = :M3[2];
                SUB3 (:RAD)
         F = 1; MC = F; F =  2; MC = F; F = 1; S = 1; A = :M3[3];
                SUB3 (:RAD)
         GOTO (:L2)
ISR:     G = M3[1]; G - 0, P; N, GOTO (:LO)
         A = :M3[4]; GOTO (:L1)
LO:      A = :M3[6]
L1:      GOTOR (MC[-1])
Sw:      + 2; GOTO (:ISR); A = :M3[8]
L2:      M3 = B
         F = 1; A = :Sw; SUBC (:TSL); GOTO (:JUA)
AA:BB:CC: GOTO (:DD)
DD:      B = MD[3]
```

6. Proceduredeclaraties

6.1. Onafhankelijk van het feit of de procedure body van een proceduredeclaratie de vorm van een blok heeft of niet, wordt door de vertaler aan de procedure body een blok van een zeer speciale vorm toegevoegd, het z.g. bodyblok. Het display level van dit bodyblok volgt de in 5.2. gegeven regels: het is 1 hoger dan het display level van het blok (of bodyblok) waarin de procedure gedeclareerd wordt.
In het bodyblok zijn steeds ondergebracht: een display, alle formele parameters van de procedure en in het geval van een type procedure de in 4.1.3. reeds genoemde simpele variabele ten behoeve van assignments aan de procedure identifier. Met alle formele parameters die in de value list voorkomen corresponderen gepre-assigneerde locale variabelen van het bodyblok, met elke name parameter correspondeert een instructiecel van 2 of 4 woorden, de reeds vaak genoemde Actual Parameter Instruction Cell (zie o.a. 1.3.2., 1.3.5. en 4.1.4.). De vertaling van een proceduredeclaratie begint dan ook met displayreservering en displaytransport (6.2.) en met de behandeling van de formele parameters (6.3.).
Indien de procedure body de vorm heeft van een gelabeld blok, zijn in het bodyblok verder nog alleen de labels ondergebracht, waarmee die body gelabeld is; het ongelabelde restant wordt in dit geval vertaald als echt binnenblok van het bodyblok, met een display level dat 1 hoger is dan dat van het bodyblok.
In alle andere gevallen worden alle eventuele locale variabelen en labels in het bodyblok ondergebracht.
Op de behandeling van de formele parameters volgt steeds de standaard bodyblokingang (6.4.), de behandeling van de labellijst (6.5.), en het value-arraytransport (6.6.). Hierna komt, in overeenstemming met het voorgaande, hetzij de vertaling van de procedure body als statement volgens de in 4. gegeven regels, hetzij, indien de procedure body de vorm heeft van een ongelabeld blok, achtereenvolgens de vertaling van de declaraties van dat blok (6.7.), de afsluiting van de vertaling van de declaraties van dat blok (6.8.), en de vertaling van de statements uit dat blok (6.9.).
De vertaling van een proceduredeclaratie wordt afgesloten door de standaard procedureverlating (6.10.).

Het transport van value array's geschiedt, ook dynamisch, na de behandeling van de formele parameters, maar voorafgaande aan de afhandeling van i.h.b. de arraydeclaraties van de procedure body. Bijgevolg vinden enerzijds de value-arraytransporten plaats nadat alle andere formele parameters uit de value list hun waarde gekregen hebben, zodat alle wijzigingen die dit laatste als side effects met zich mee brengt in de met de value array's corresponderende actuele parameters, in de uiteindelijke elementen van de value array's teruggevonden worden. Anderzijds heeft het uitvoeren van het value-arraytransport voorafgaande aan de behandeling van de arraydeclaraties uit de procedure body tot gevolg dat tijdens de evaluatie van de bound pair lists daarvan wel, via side effects, de elementen van een met een formeel array uit de value list corresponderende actuele parameter van waarde kunnen veranderen, zonder dat dit konsekwenties heeft voor de waarde van de elementen van dat value formele array.

In het volgende worden de onderdelen waaruit de vertaling van een proceduredeclaratie bestaat in successie behandeld. Deze onderdelen zijn als genoemd: displayreservering en -transport, de behandeling van formele parameters, de standaard bodyblokingang, de labellijst, het value-arraytransport, de vertaling van de declaraties van de procedure body, de afsluiting van de vertaling van die declaraties, de vertaling van de statements van de body, en de standaard procedureverlating. Afgezien van deze onderdelen wordt de vertaling van een proceduredeclaratie soms voorafgegaan door de macro JU, zoals uiteengezet in 5.7.

6.2. De displayreservering en het displaytransport

De vertaling van een proceduredeclaratie begint met:

```
Pr:     S =  D                          " DPTR (display level)
Pr[1]:  A = - [display level - 2], Z
        SUB (:DPTR)
        B + [top of display]            " INCRB (top of display)
```

Hierin is display level het display level van het bodyblok (zie 6.1.). De top of display is een getal, door de vertaler aan iedere procedure uit het programma toegekend. De waarde van top of display is 1 + het maximale aantal in elkaar geschakelde echte binnenblokken van een procedure die niet binnenblok zijn van een andere proceduredeclaratie. Door de macro INCRB wordt een element in de display van de procedure gereserveerd voor elk blokniveau binnen de procedure dat behoort tot die procedure, te beginnen met dat voor het bodyblok zelf.

6.3. De behandeling van formele parameters

Als de procedure formele parameters heeft, volgt op de displayreservering en het displaytransport een aantal instructies, nl. aanroepen van subroutines uit het complex, en wel voor iedere parameter precies een instructie, in de volgorde van de formal parameter list. Welke van de mogelijke aanroepen van het complex gekozen wordt voor een formele parameter wordt enerzijds bepaald door het al of niet voorkomen van die parameter in de value list, anderzijds door het gebruik van die parameter binnen de procedure body. De regels hiervoor zijn:

6.3.1.

formele parameters uit de value list:
De keuze van de aanroep van het complex wordt voor value parameters geheel bepaald door de specificatie. Het verband wordt gegeven in de volgende tabel:

specificatie	instructie	macro
real (procedure)	SUBC (:CRV)	" CRV
integer (procedure)	SUBC (:CIV)	" CIV
Boolean (procedure)	SUBC (:CBV)	" CBV
string (procedure)	SUB1 (:CSTV)	" CSTV
array	SUB (:CEN)	" CEN
<type> array	SUB (:CEN)	" CEN
label	SUBC (:CLV)	" CLV

Behalve in het geval van een specificatie als array of als <type> array wordt door de instructie één enkele waarde afgeleid uit de bij de actuele parameter behorende APD; deze waarde wordt op de stapel gelegd; de zo gereserveerde en gepre-assigneerde geheugenplaats(en) wordt (worden) bij de vertaling van de body verder behandeld als een normale, dynamisch geadresseerde variabele of pseudo variabele van het bodyblok. Het effect van de macro CEN wordt beschreven in 6.3.2.

6.3.2. formele parameters called by name:
De keuze van de aanroep van het complex wordt voor name parameters geheel bepaald door het feit, of ergens in de procedure body de formele parameter voorkomt als left part van een assignment statement of als controlled variable van een for statement. De twee mogelijke macro's zijn:

bij voorkomen als left part of als controlled variable:	SUB1 (:CLPN)	" CLPN
anders:	SUB (:CEN)	" CEN

Beide macro's construeren tijdens de executie van het objectprogramma, op grond van de bij de actuele parameter behorende APD, de z.g. Actual Parameter Instruction Cell op de stapel, en wel CEN een APIC van twee geheugenwoorden, CLPN een uitgebreidere APIC van vier woorden. Het eerste woord van een APIC bevat steeds een instructie, de reeds in 1.3.2. genoemde APIC 0-instructie; het met de formele parameter corresponderende dynamische adres wijst naar deze instructie. In geval van een uitgebreidere APIC bevatten het derde en vierde woord instructies ter voorbereiding en ter uitvoering van een assignment aan de met de formele parameter corresponderende actuele parameter (zie 4.1.4.). Een volledige lijst van alle enkelvoudige en uitgebreidere APIC's, in afhankelijkheid van de APD, is opgenomen in appendix D.

6.4. De standaard bodyblokingang

Op de instructies voor de behandeling van de formele parameters, of, bij een empty formal parameter part, aansluitend aan de displayreservering en -transport, volgt de standaard bodyblokingang. Deze bestaat uit de volgende 3 instructies:

```
F = :MA[display level]      " TDL (display level)
B + [local space]           " ENTRPB (local space)
SUB2 (:ENTRPB)
```

Hierin is display level het reeds in 6.2. gebruikte display level van het bodyblok. De local space is een getal, dat door de vertaler aan ieder bodyblok wordt toegekend. Bij de bepaling van die local space is het van belang of de procedure body de vorm heeft van een gelabeld blok of niet (zie 6.1.). In het eerste geval worden nl. alle locale grootheden van dat gelabelde blok ondergebracht in een zelfstandig, echt binnenblok van het bodyblok, en dragen slechts de labels die dat blok labelen en eventueel de pseudo variabele bestemd voor het onthouden van de functiewaarde, bij tot de local space. De local space bedraagt:

(if <type> procedure declaration
then (if <type> = real ∨ <type> = string then 2 else 1)
else 0)
+ 2 × het aantal locale, simpele, niet-own real variabelen
+ 1 × het aantal locale, simpele, niet-own integer variabelen
+ 1 × het aantal locale, simpele, niet-own Boolean variabelen
+ 2 × het aantal locale, simpele, niet-own stringvariabelen
+ 1 × het aantal locale, niet-own array's
+ 1 × het aantal locale pseudo forvariabelen
+ 2 × het aantal locale, maar niet-superlocale labels

Geheel als in 5.2. is het aantal pseudo forvariabelen van een body-blok gelijk aan het maximale aantal forclauses in dat bodyblok dat een statement van dat blok omvat. Evenzo heet een label van een bodyblok superlocaal, als het een <identifier> label betreft, die niet op andere wijze gebruikt wordt dan in de combinatie "goto <label identifier>", staande in het bodyblok zelve (vgl. 4.2.). Bij de local space behoren voor een <type> procedure de geheugen-plaatsen bestemd voor de pseudo <type> variabele, waaraan assignments aan de procedure identifier in feite worden gedaan (zie 4.1.3. en 6.10.). Het aantal locale, simpele, niet-own variabelen en array's kan alleen dan van nul verschillen als de procedure body de vorm heeft van een ongelabeld blok.

6.5. De labellijst

Op de standaard bodyblokingang volgt, indien het aantal locale, niet-superlocale labels van het bodyblok van nul verschilt, het stuk objectprogramma dat reeds eerder in 5.3. besproken is.

6.6. Het value-arraytransport

Na de, eventueel lege, labellijst volgt pas de reservering van de ruimte, benodigd voor value array's, en het toekennen van de waarde aan alle elementen van deze array's. Voor elk array uit de value list waarvan de specificatie gegeven wordt als integer array geschiedt het value-arraytransport door het macropaar:

```
                        A = :Mp[q]          " TAA (f)
integer array:          SUB4 (:TIAV)        " TIAV
```

Bij alle andere specificaties voor een formeel array uit de value list geschiedt het value-arraytransport door het macropaar:

```
                        A = :Mp[q]          " TAA (f)
niet-integer array:     SUB4 (:TAV)         " TAV
```

SUB4 duidt een subroutinesprong aan analoog aan SUB, met link op LINK[4] = M[12]. Het dynamische adres p, q wijst naar de tijdens de behandeling van de formele parameters geconstrueerde APIC 0-instructie (zie 6.3.1., en voor het effect van CEN ook 6.3.2.). Via het adresgedeelte van deze instructie is de reference-vector (zie 1.3.3.) van het actuele array toegankelijk. Hieraan worden door TAV en TIAV de dimensie en de subscript bounds van het value array ontleend. Uiteindelijk wordt door TAV en TIAV de door het adres p, q aangewezen APIC 0-instructie overschreven door het adres van de

reference-vector van het nieuw geconstrueerde array. Deze geheugenplaats fungeert dan verder als array-sleutel van het nieuwe array en wordt bij de vertaling van de body verder behandeld als een normale, dynamisch geadresseerde array-sleutel.
De value-arraytransporten geschieden in de volgorde van de formal parameter list.

6.7. De vertaling van de declaraties van het bodyblok

Uitsluitend als de procedure body de vorm heeft van een ongelabeld blok, volgt nu de vertaling van de declaraties van dat blok. Deze volgt geheel uit hetgeen reeds beschreven is in 5.4. t/m 5.7.

6.8. De afsluiting van de declaraties van het bodyblok

Uitsluitend als de procedure body de vorm heeft van een ongelabeld blok, en als bovendien onder de declaraties van dat blok arraydeclaraties voorkomen, wordt de vertaling van de declaraties van dat blok afgesloten met de macro:

```
MO[512×display level] = B        " SWP (display level)
```

geheel analoog aan 5.8. Hierin is display level het display level van het bodyblok.
Merk op, dat de aanwezigheid van value array's onder de formele parameters niet van invloed is op het al dan niet opnemen van de macro SWP.

6.9. De vertaling van de statements van de body

Deze is reeds in 4. behandeld. Als de procedure body de vorm heeft van een gelabeld blok, volgt de vertaling van dat blok geheel uit 5.

6.10. De standaard procedureverlating

Deze bestaat in principe uit een van de macro's EXITP en EXITPC. Bij een <type> procedure wordt echter vooraf de waarde van de bij de procedure identifier behorende pseudo variabele (vgl. 6.4. en 4.1.3.) in het bij dat type behorende register genomen door een van de macro's:

real:	F = MD[q]	" TRV (Pr)
integer:	G = MD[q]	" TIV (Pr)
Boolean:	S = MD[q], P	" TBV (Pr)
string:	A = MD[q]	" TSTV (Pr)
	loose string = B	" LOS

Bij deze laatste macro is loose string een vlag uit het complex die de waarde true krijgt bij het verlaten van een string procedure (vgl. 2.1.3.5.). De vlag wordt false gezet o.a. door de macro REJST (zie 4.4.) en door de macro's STST (4.1.2.), STSST (4.1.5.), en CSTV (6.3.1.).
De eigenlijke procedureverlating gebeurt door:

```
                    GOTO (:EXITP)          " EXITP
```

tenzij het een string procedure betreft, of van de formele parameters een value parameter als string of string array gespecificeerd is, of de procedure body de vorm heeft van een ongelabeld blok en in dat blok locale simpele of geindiceerde stringvariabelen gedeclareerd worden. In dat geval wordt de procedure verlaten door de macro:

```
                    GOTO (:EXITPC)         " EXITPC
```

6.11. voorbeeld:

```
"integer procedure factorial (n); value n; integer n;
 begin  integer i, fac;
        fac:= 1;
        for i:= 2 step 1 until n do fac:= i × fac;
        factorial:= fac
 end"
```

```
FACTORIAL: S = D; A = - 0, Z; SUB (:DPTR); B + 1
           SUBC (:CIV)                                  " value n
           F = :MD[2]; B + 4; SUB2 (:ENTRPB)
           F = 1; S = F, Z; N, SUBC (:RND); MD[7] = G   " fac:= 1
           F = :STEP UP; MD[5] = G                      " for i:=
           F = 2; GOTO (:ASSIGN)                        " 2
STEP:      F = 1                                        " step 1
STEP UP:   G = MD[6]; MC = F; DO (STEP); F + MC[-2]
ASSIGN:    S = F, Z; N, SUBC (:RND); MD[6] = G
           MC = F; G = MD[3]; F - MC[-2], Z             " until n
           DO (STEP); N, F = F, E; N, F = F, Z
           Y, GOTO (:ACTION); GOTO (:FINISH)            " do
ACTION:    G = MD[6]; G × MD[7]
           S = F, Z; N, SUBC (:RND); MD[7] = G          " fac:= i×fac
           GOTO (MD[5])
FINISH:    G = MD[7]                                    " factorial:=
           S = F, Z; N, SUBC (:RND); MD[4] = G          " fac
           G = MD[4]; GOTO (:EXITP)
```

7. ALGOL 60-programma's

7.1. De vertaling van een ALGOL 60-programma is uit de volgende onderdelen opgebouwd: de constantenlijst, de primaire display, de statische ruimte, het objectprogramma in engere zin, en het ademgetal. Deze onderdelen worden in 7.2. t/m 7.6. besproken.
Door de vertaler worden bovendien nog drie adressen afgeleverd, en wel in de volgende variabelen:
in "dp0" het beginadres van de primaire display,
in "begin of program" het beginadres van het objectprogramma in engere zin, en
in "instr cntr" het adres van de op het ademgetal volgende geheugenplaats (het eerste "vrije" woord).

7.2. De constantenlijst

Elke <unsigned number> uit een ALGOL-programma wordt voor elk voorkomen in de tekst van dat programma als 2 woorden (in de standaardrepresentatie van floating point getallen) opgenomen in de constantenlijst, en wel in de volgorde waarin ze in die tekst worden aangetroffen. Echter worden <unsigned integer>'s met een waarde < 32768 $(= 2 \uparrow 15)$ niet in de constantenlijst opgenomen. Het adres van een in de constantenlijst opgenomen constante (vgl. 1.3.7. en 3.3.6.) is het adres van de laagst gelegen cel van 1 of 2 woorden (respectievelijk voor integers ≥ 32768 en voor real numbers) die die constante bevat.

7.3. De primaire display

Voor elk programma reserveert de vertaler, direct aansluitend aan de (misschien lege) constantenlijst, ruimte voor de z.g. primaire display. Het aantal hiervoor gereserveerde geheugenplaatsen bedraagt:

1 + het maximale display level voor blokken die geen binnenblok zijn van een procedure

Het beginadres van deze primaire display wordt door de vertaler in de variabele "dp0" afgeleverd (vgl. 7.1.).

7.4. De statische ruimte

7.4.1. In de statische ruimte wordt plaats gereserveerd voor alle simpele, niet-own variabelen, voor de array-sleutels (zie 1.3.3.) van alle niet-own array's, voor alle pseudo label-variabelen (zie 2.2.1.), en voor alle pseudo forvariabelen (zie 4.8.1.), voor zover deze grootheden locaal zijn t.o.v. een blok dat geen binnenblok is van een procedure, en tevens voor alle simpele own variabelen en voor de array-sleutels en de array-reference-vectoren van alle own array's, ongeacht t.o.v. welk blok of welke procedure ze locaal zijn. Al deze grootheden worden statisch geadresseerd.
Het aantal woorden dat voor elk van de bovengenoemde grootheden in de statische ruimte wordt gereserveerd is in de volgende tabel weergegeven (vgl. 5.2., 6.4. en 8.3.):

soort grootheid	aantal woorden
simpele, own of niet-own real variabele	2
simpele, own of niet-own integer variabele	1
simpele, own of niet-own Boolean variabele	1
simpele, own of niet-own stringvariabele	2
niet-own array	1
own array met dimensie n	3 × n + 6
pseudo forvariabele	1
pseudo label-variabele	2

Er zij hier op gewezen dat, in tegenstelling tot 5.2. en 6.4., bij blokken die geen binnenblok zijn van een procedure, geen onderscheid wordt gemaakt tussen superlocale en niet-superlocale labels: voor elke label die locaal is t.o.v. een dergelijk blok wordt, ongeacht zijn gebruik, door de vertaler een pseudo label-variabele van 2 woorden gereserveerd in de statische ruimte. Bovendien wordt deze pseudo label-variabele door de vertaler geinitialiseerd. Het eerste woord bevat het beginadres van het stuk objectprogramma dat correspondeert met de vertaling van de met die label gelabelde statement. Het tweede woord wordt gevuld met:

(display level × d18 + dp0)

waarin display level het display level van het blok aangeeft waarin die label locaal is en dp0 het beginadres van de primaire display (zie 7.3.) aanduidt.

Het aantal pseudo forvariabelen van een blok is steeds gelijk aan het maximale aantal forclauses in dat blok dat een statement van dat blok omvat.

7.4.2. de volgorde van reservering:
De reservering van plaatsruimte voor statisch geadresseerde grootheden in de statische ruimte geschiedt door de vertaler in een tekstinspectie (van "links" naar "rechts") die aan de produktie van het objectprogramma in engere zin voorafgaat, en wel aan het einde van de inspectie van het blok waarin die grootheden locaal zijn, met uitzondering van de ruimte voor own array's en voor labels, die direct bij inspectie van respectievelijk hun declaratie en de door die label gelabelde statement hun plaats toegewezen krijgen. Dit houdt in dat simpele (own of niet-own) variabelen, niet-own array's en pseudo forvariabelen die locaal zijn t.o.v. binnenblokken van een blok eerder geadresseerd worden en dus een lager adres krijgen dan dezelfde grootheden van dat blok zelf.
De volgorde waarin genoemde grootheden aan het eind van de inspectie van een blok geadresseerd worden is: eerst de pseudo forvariabelen, daarna simpele locale variabelen en locale niet-own array's, in volgorde van declaratie, en ten slotte de locale labels, eveneens in volgorde van voorkomen als label voor een statement.
Bij proceduredeclaraties en bij binnenblokken daarvan worden slechts simpele own variabelen en own array's statisch geadresseerd. De ruimtereservering vindt hierbij plaats: voor own array's bij inspectie van hun declaratie, en voor simpele own variabelen bij het eind van de inspectie van de proceduredeclaratie of van het binnenblok daarvan, weer in volgorde van de <type declaration>.

7.5. Het objectprogramma in engere zin

Het objectprogramma in engere zin bestaat uit de vertaling van de compound statement of van het blok waaruit het ALGOL-programma bestaat, volgens de regels van 4.6. of 5., afgesloten door de macro:

```
. GOTO (:ENDRUN)                    " END
```

waardoor de besturing teruggegeven wordt aan het bedrijfssysteem. Het beginadres van het objectprogramma in engere zin wordt door de vertaler in de variabele "begin of program" afgeleverd (vgl. 7.1.).

7.6. Het ademgetal

Tijdens de executie van een objectprogramma zal de stapel groeien en slinken. Geregeld zal in deze fase getest worden, of de stapel de toegestane omvang niet te boven gaat. Met name gebeurt dit door de macro's ENTRB (5.2.), ENTRPB (6.4.), ENTRIS (3.4.), RAD, IAD, BAD, STAD, ORAD, OIAD, OBAD, OSTAD (5.5.), TIAV en TAV (6.6.).
Het ademgetal is een door de vertaler voor elk ALGOL-programma vastgestelde bovengrens (die zeker geen kleinste bovengrens is) voor het aantal woorden, waarmee de stapel kan groeien tussen twee zulke controles. Bij elke controle wordt vastgesteld, of er nog ten minste dat aantal woorden beschikbaar is in het ongebruikte gedeelte van het geheugen direct boven de top van de stapel.
Het ademgetal wordt in 1 woord als integer opgeslagen, direct aansluitend aan de macro END (zie 7.5.). Dit woord is tevens het laatste woord van het objectprogramma; het adres van het er op volgende woord wordt door de vertaler in de variabele "instr cntr" afgeleverd (vgl. 7.1.).
De vertaler-algoritme die het ademgetal afleidt uit de tijdens de vertaling van het programma geproduceerde string van macro's valt buiten het bestek van dit rapport.

7.7. voorbeeld:

```
"begin integer i;
      integer procedure nfac (n); value n; integer n;
      nfac:= if n = 0 then 1 else n × nfac (n - 1);
      i:= nfac (5)
 end"
```

```
PR DSP:    'SKIP' 1              "                  primaire display
           'SKIP' 1
I:         'SKIP' 1              "                  i
START:     A = :MC               " TBL (1)          take block level
           F = :MD[1]
           B +  1                " ENTRB (1)        entrance block
           SUB2 (:ENTRB)
           GOTO (:L4)            " JU (L4)          sprong over body
NFAC:      S =  D                " DPTR (2)         display transport
           A = - 0, Z
           SUB (:DPTR)
           B +  1                " INCRB (1)        increase B (voor top)
           SUBC (:CIV)           " CIV              call integer by value
           F = :MA[2]            " TDL (2)          take display level
```

```
            B +   1              " ENTRPB (1)     entrance proc. body
            SUB2 (:ENTRPB)
            G = MD[3]            " TIV (n)        take integer value
            F -   0, Z           " EQUSIC (0)     equal small int. cnst.?
         N, GOTO (:L0)           " COJU (L0)      conditional jump
            F =   1              " TSIC (1)       take small int. cnst.
            GOTO (:L3)           " JU (L3)        jump (over else-deel)
L0:         G = MD[3]            " TIV (n)        take integer value
            MC =  F              " STACK
            GOTO (:L2)           " JU (L2)        sprong over impl. subr.
L1:         SUB2 (:ENTRIS)       " ENTRIS         entrance implicit subr.
            G = MD[3]            " TIV (n)        take integer value
            F -   1              " SUBSIC (1)     subtract sml int. cnst.
            GOTO (:EXITIS)       " EXITIS         exit implicit subr.
L2:         SUBC (:NFAC)         " SUBJ (NFAC)    subroutine jump
            (20 × d20 + :L1)     " CODE (APD)     actual parameter descr.
            F × MC[-2]           " MUL            multiply
L3:         S =    F, Z          " STI (NFAC)     store integer
         N, SUBC (:RND)          "                (rond zonodig af)
            MD[4] = G            "                (assignment to nfac)
            G = MD[4]            " TIV (NFAC)     (function value)
            GOTO (:EXITP)        " EXITP          exit procedure
L4:         SUBC (:NFAC)         " SUBJ (NFAC)    subroutine jump
            (7 × d20 + 5)        " CODE (APD)     actual parameter descr.
            S =   F, Z           " STI (I)        store integer
         N, SUBC (:RND)          "                (rond zonodig af)
            I =   G
            B = MD[1]            " EXITB (1)      exit block
            GOTO (:ENDRUN)       " END
            +     9              " CODE (9)       ademgetal
STCK BEG:
```

De vertaler levert verder af:

```
in "dp0":                :PR DSP
in "begin of program":   :START
in "instr cntr":         :STCK BEG
```

8. Dynamische adressering

8.1. Aan elke dynamisch geadresseerde grootheid wordt door de vertaler, voorafgaande aan de productie van het objectprogramma in engere zin (zie 7.1. en 7.5.), bloksgewijze een dynamisch adres [p,q] toegekend (steeds $2 \leq p \leq 57$, $1 \leq q \leq 255$). Hierin is p het display level van de procedure of van het binnenblok daarvan ten opzichte waarvan de grootheid lokaal is, en dus dezelfde voor alle dynamisch geadresseerde grootheden van een blok, terwijl q binnen een blok van grootheid tot grootheid verschilt en bepaald wordt door de volgorde van adrestoekenning in een blok. De regels hiervoor worden in 8.3. gegeven.

8.2. optimalisering van dynamische adressen:
Tijdens de productie van het objectprogramma in engere zin wordt bij elk voorkomen van een dynamisch geadresseerde grootheid in de tekst in plaats van zijn dynamische adres [p,q] het adres [D,q] (dus [63,q]) gebruikt, mits die grootheid locaal is t.o.v. een procedure body en het betreffende voorkomen
a) niet deel uitmaakt van een (andere) proceduredeclaratie ten opzichte waarvan de grootheid (dus) globaal is,
b) niet deel uitmaakt van een switchdeclaratie (vgl. 5.6.5.).

Door dat gebruik van het adres [D,q] i.p.v. het adres [p,q] wordt tijdens executie van het programma bij de adresberekening een geheugenaccess uitgespaard in die gevallen dat (volgens de vertaler) zeker MD[p] dezelfde waarde zal bevatten als D.

8.3. volgorde van adrestoekenning:
Voorafgaande aan de adrestoekenning aan de grootheden van een dynamisch geadresseerd blok wordt q geinitialiseerd op:

display level + top of display	voor een procedure body
1	voor elk gewoon binnenblok daarvan

Na elke toekenning van een adres aan een grootheid wordt q opgehoogd met een bedrag dat afhangt van de soort grootheid.

De volgorde van adrestoekenning aan de grootheden van een procedure body is:
1) de formele parameters, in volgorde van de formal parameter list,
2) in geval van een type procedure: de simpele variabele ten behoeve van assignments aan de procedure identifier (zie 4.1.3. en 6.10.),
3) pseudo forvariabelen (zie 4.8.1.),
4) simpele niet-own variabelen en array-sleutels (zie 1.3.3.) van niet-own array's, in volgorde van declaratie,
5) pseudo label-variabelen (zie 2.2.1.) voor locale maar niet-superlocale labels (zie 5.2.), in volgorde van voorkomen als label voor een statement.

De totale ophoging van q voor de grootheden genoemd sub 2 t/m 5 is juist local space (zie 6.4.).

De volgorde van adrestoekenning aan de grootheden van een gewoon binnenblok van een procedure is:
1) pseudo forvariabelen,
2) simpele niet-own variabelen en array-sleutels van niet-own array's,

3) pseudo label-variabelen.
De totale ophoging van q bedraagt in dit geval juist local space - 1 (zie 5.2.), zodat in een dergelijk blok steeds $1 \leq q < \text{local space}$.

De ophoging van q per grootheid wordt voor elke soort grootheid in onderstaande tabel gegeven:

soort grootheid	q-ophoging
formele parameters uit de value list:	
specificatie integer (procedure)	1
Boolean (procedure)	1
overige	2
formele parameters called by name:	
bij voorkomen als left part of als controlled variable	4
anders	2
simpele variabele ten behoeve van assignment aan de procedure identifier:	
real procedure en string procedure	2
integer procedure en Boolean procedure	1
pseudo forvariabelen	1
simpele variabelen:	
real en string	2
integer en Boolean	1
array-sleutels	1
pseudo label-variabelen	2

9. Het bijhouden van regelnummers tijdens executie

9.1. Ten behoeve van foutlocalisatie in geval van een foutmelding tijdens de executiefase van een programma kan het objectprogramma desgewenst aangevuld worden met instructies die tijdens executie moeten zorgen voor het bijhouden van een regelnummer. Hiermee kan in een foutmelding verwezen worden naar de regel van de ALGOL-tekst waarop de laatst in executie genomen maar nog onvoltooide statement of array segment begint. Door genoemde extra instructies wordt het objectprogramma langer (b.v. 10 %) en iets langzamer (b.v. 1 %).

De vertaler zal het objectprogramma al dan niet aanvullen met instructies voor het bijhouden van regelnummers tijdens de executiefase alnaargelang de variabele "wanted" true (> 0) of false (< 0) is.

9.2. Het true-zijn van wanted brengt de volgende veranderingen teweeg in het objectprogramma als beschreven in de voorafgaande hoofdstukken:

a) na het toewijzen van een dynamisch adres [p,q] aan de anonieme simpele variabele bij een type procedure waaraan assignments aan de procedure identifier geschieden (zie 4.1.3. en 6.10.) wordt q steeds met 3 opgehoogd en niet met 1 of 2 afhankelijk van het type van de procedure (zie 8.3.),

b) bij de vertaling van de declaratie van een type procedure Pr (zie 6.) worden ingelast:
tussen value-arraytransport (6.6.) en de vertaling van de declaraties van het bodyblok (6.7.):

```
S = linecounter          " SLNC (Pr)
MD[q+2] = S
```

en direct voorafgaande aan de macro EXITP of EXITPC (6.10.):

```
S = MD[q+2]              " RLNC (Pr)
linecounter = S
```

In beide gevallen wordt met q de q-component aangeduid van het aan de onder a genoemde simpele variabele toegekende dynamische adres [p,q]. In linecounter wordt tijdens executie het regelnummer bijgehouden.

c) de vertaling van een statement die geen compound statement en geen dummy statement is, en de vertaling van een array segment wordt, waar nodig, voorafgegaan door:

```
S = [lnc]                " LNC (lnc)
linecounter = S
```

Hierin is lnc het regelnummer van de regel waarop het eerste basic symbol van de statement, of het basic symbol direct voorafgaande aan het array segment (dus "array" of ",") staat.

d) na de vertaling van de statement achter "do" in een for statement maar voor de macro IJU (zie 4.8.1.) wordt, waar nodig, ingelast:

```
S = [lnc]                " LNC (lnc)
linecounter = S
```

waarbij lnc het regelnummer aanduidt van de regel waarop het corresponderende basic symbol "for" staat.

De vertaler-algoritme die bepaalt op welke plaatsen de sub c en d genoemde inlassen moeten geschieden valt buiten het bestek van dit rapport.

10. Bibliotheekprocedures en codeprocedures

10.1. Er zijn twee vertaalschema's voor procedure statements en function designators indien de betreffende procedure niet in het programma gedeclareerd wordt maar in de bibliotheek is opgenomen. Welk van deze twee schema's gevolgd moet worden is voor elke bibliotheekprocedure vastgelegd in de catalogus van de bibliotheek. Het eerste, algemene schema (10.3.) is steeds toepasbaar, het tweede (10.4.) uitsluitend indien alle formele parameters in de value list zijn opgenomen en hetzij real, hetzij integer gespecificeerd zijn.

De wijze waarop procedures in de bibliotheek en in de catalogus daarvan kunnen worden opgenomen is systeemafhankelijk; de beschrijving hiervan valt buiten het bestek van dit rapport.

Bij elke in het programma gebruikte bibliotheekprocedure die het algemene schema volgt behoren twee opvolgende woorden in het geheugen. Beide woorden bevatten het adres van een ingang van de met de procedure corresponderende subroutine; het eerste dat voor directe aanroepen van de procedure, het tweede dat voor aanroepen via de correspondentie formele-actuele parameter.

Bij elke in het programma gebruikte bibliotheekprocedure die het tweede schema volgt behoort een woord in het geheugen dat het adres van de formele procedure-ingang bevat. Dit woord wordt slechts gebruikt bij de vertaling van de bibliotheeknaam als actuele parameter.

Genoemde woorden kunnen door de vertaler gereserveerd en ingevuld worden in de statische ruimte (zie 7.4.) of direct aansluitend aan het ademgetal (zie 7.6.); in de eenvoudigste opzet van een bibliotheeksysteem kunnen echter permanent gereserveerde (en ingevulde) woorden gebruikt worden.

10.2. De vertaling van een bibliotheeknaam als actuele parameter is analoog aan 3.4.2.; slechts wordt de macro JU1 vervangen door de macro IJU1, zodat de impliciete subroutine er in dit geval als volgt uitziet:

```
ISR:    S = MS[1]              " TFD
        GOTO (LP[1])           " IJU1 (LP)
```

Hierin verwijst LP[1] naar het tweede woord van de twee bij de procedure behorende woorden als deze het eerste schema volgt en naar het enige woord dat bij een procedure hoort die het tweede schema volgt.

10.3. Bij een directe aanroep, hetzij als function designator, hetzij als procedure statement, van een bibliotheekprocedure die het algemene schema volgt worden actuele parameters vertaald als beschreven in 3. De aanroep van de procedure wordt echter niet met de macro SUBJ vertaald (zie 1.3.5. en 4.4.), maar met behulp van:

```
        SUBC (LP)              " ISUBJ (LP)
```

waarbij LP het eerste van de twee bij de procedure behorende woorden aanduidt. Bovendien wordt, net als in 4.4., in geval van een procedure statement bovengenoemde aanroep van de bibliotheekprocedure,

inclusief impliciete subroutines en APD's, nog gevolgd door de macro REJST indien de procedure een string procedure is.

10.4. Bij een directe aanroep, hetzij als function designator, hetzij als procedure statement, van een bibliotheekprocedure die het tweede schema volgt worden eerst alle actuele parameters vertaald volgens het vertaalschema voor arithmetische expressies (zie 1.), elk gevolgd door de macro STACK behalve de laatste; de vertaling van de laatste actuele parameter wordt gevolgd door een bij de procedure behorende macro (meestal een subroutine-aanroep). Zo luidt de vertaling van de statement "ABSFIXP (AE1, AE2, AE3)" symbolisch:

```
F = AE1
MC =  F                  " STACK
F = AE2
MC =  F                  " STACK
F = AE3
SUBC (:ABSFIXP)          " ABSFIXP
```

Dit tweede schema vermijdt het parametermechanisme en is speciaal ontworpen voor de standaardfuncties "abs", "sign", "sqrt", "sin", "cos", "arctan", "ln", "exp" en "entier". Voor deze functies luiden de bijbehorende macro's:

```
   F =   F, P            " ABS
N, F = - F

   F =   F, Z            " SIGN
N, F =   1, E
N, F = - 1

   SUBC (:SQRT)          " SQRT

   SUBC (:SIN)           " SIN

   SUBC (:COS)           " COS

   SUBC (:ARCTAN)        " ARCTAN

   SUBC (:LN)            " LN

   SUBC (:EXP)           " EXP

   SUBC (:ENTIER)        " ENTIER
```

die alle een transformatie van F naar F uitvoeren.
Om historische redenen volgen ook READ, ABSFIXP, FIXP, FLOP, PUNCH, PUNLCR, PUSPACE, RUNOUT, PUHEP, HAND, XEEN en STOP het tweede schema.

10.5. codeprocedures:
Deze zijn toegestaan en hebben de volgende vorm: op de procedure heading volgen een quote symbol, een rij macro-omschrijvingen onderling gescheiden door komma's, en tenslotte een unquote symbol. Elke macro-omschrijving bestaat uit een unsigned integer (het "macro-nummer" van de macro), eventueel gevolgd door een komma en een "macro-parameter". Deze laatste is of een unsigned integer, of een

minteken gevolgd door een unsigned integer, of een identifier. Een nadere beschrijving van codeprocedures valt buiten het bestek van dit rapport: voor het schrijven ervan is kennis van de macro-processor in de vertaler en van de run-time-organisatie vereist.

11. Varianten

11.1. switch identifier als actuele parameter

De huidige vertaler is niet in staat de in 3.3.4. beschreven APD voor een switch identifier als actuele parameter te construeren indien de declaratie van die switch identifier niet aan dat optreden als actuele parameter voorafgaat. De eenvoudigste remedie is de volgende wijziging van het objectprogramma:
een switch identifier als actuele parameter geeft steeds aanleiding tot een impliciete subroutine van de vorm:

```
ISR:    A = :Sw                    " TSWE (Sw)
```

waarin Sw het statische adres van het met de switchdeclaratie corresponderende stuk objectprogramma (5.6.) aangeeft. De bijbehorende APD heeft dan de vorm:

```
switch identifier:    (12 × d20 + :ISR)    " APD (Sw)
```

Bovendien is een kleine wijziging van de run-time-subroutine CEN nodig.

11.2. werkstapel boven 32 K

Het in de voorafgaande hoofdstukken gedefinieerde objectprogramma dient noodzakelijk in de eerste 32768 woorden van het kerngeheugen te staan, terwijl tijdens executie ook de "werkstapel" niet buiten genoemd geheugengedeelte mag komen. Om althans dit laatste wel mogelijk te maken kan volstaan worden met de volgende wijziging van de dynamische adressering in het objectprogramma (zie 8.3.):
na de toekenning van een dynamisch adres [p, q] aan een formele identifier die niet in de value list voorkomt en in de procedure body niet als left part of als controlled variable gebruikt wordt moet q niet met 2 maar met 3 opgehoogd worden.
Bovendien moeten de run-time-subroutines CEN, CLPN, CRV, CIV, CBV, CSTV, CLV en TAV, de systeemprocedure CONSIDER en allicht ook enige bibliotheekprocedures aangepast worden.
Door genoemde wijzigingen wordt het aanroepen van een procedure met parameters iets langduriger.

11.3. TO DRUM/FROM DRUM

Ten behoeve van een automatische synchronisatie bij transporten van en naar een stuk achtergrondgeheugen op de trommel kan gebruik gemaakt worden van het fout-adres-ingreepmechanisme van de X8 (het voordeel van deze methode van synchronisatie is dat het indiceren van array's, toch al een relatief dure operatie, niet langer gaat duren). Hiervoor dienen de macro's STAA (zie 4.1.5.) en EXITB (zie 5.10.) als volgt gewijzigd te worden:

```
        U, S = :MA                    " STAA
          MC =  A
```

```
   B = MD[display level]          " EXITB (display level)
U, B + DREF[1], P
N, JUMP (-2)
```

terwijl voorts de run-time-routines EXITPB, EXITPC, FREE CHAIN, FAD, FADCV en JUA dienen te worden aangepast.

Appendix A

In deze appendix worden alle macro's getabelleerd die door de vertaler gegenereerd worden op enige in 10.4. genoemde macro's voor in- en uitvoer na. Ze zijn alfabetisch op naam gesorteerd. Voor elke macro wordt gegeven: de instructies waaruit de macro is opgebouwd, een omschrijving van de macro en een of meer verwijzingen. Vele macro's bevatten een macro-parameter. Deze is in de instructies aangegeven met "par" voor een geheugenoperand, met ":par" als het een adresoperand is en met "[par]" als het een getaloperand betreft. De verwijzingen geven de plaatsen in het rapport waar de macro primair (dwz. niet in een voorbeeld of in een variant) gebruikt wordt.

APD's (zie 3.) zijn niet als macro opgevat; het zijn door de vertaler geproduceerde codewoorden die in het objectprogramma opgenomen worden door de macro CODE.

naam	instructies	omschrijving	verwijzingen
ABS	F = F, P N, F = - F	absolute value	10.4
ADD	F + MC[-2]	add	1.4.1 4.8.5
ADDIC	G + par	add integer constant	B
ADDIV	G + par	add integer variable	B
ADDRC	F + par	add real constant	B
ADDRV	F + par	add real variable	B
ADDSIC	F + [par]	add small integer constant	B
AND	N, B - 1 - Y, S = MC[-1], P	and	1.6.1
ARCTAN	SUBC (:ARCTAN)	arctangent	10.4
BAD	SUB3 (:BAD)	Boolean array declaration	5.5
CBV	SUBC (:CBV)	call Boolean by value	6.3.1
CEN	SUB (:CEN)	call expression by name	6.3.1 6.3.2
CIV	SUBC (:CIV)	call integer by value	6.3.1
CLPN	SUB1 (:CLPN)	call left part by name	6.3.2
CLV	SUBC (:CLV)	call label by value	6.3.1
CODE	[par]	code	5.6.2 (zie ook A)
COJU	N, GOTO (:par)	conditional jump	1.2 2.1.2 2.2.2 4.7

naam	instructies	omschrijving	verwijzingen
COS	SUBC (:COS)	cosine	10.4
CRV	SUBC (:CRV)	call real by value	6.3.1
CSTV	SUB1 (:CSTV)	call string by value	6.3.1
DECB	B — [par]	decrease B	4.8.3.3 5.3 6.5
DECS	S — 2	decrease S	3.4.3 3.4.6
DIV	MC = F F = MC[—4] F / MC	divide	1.4.1
DIVIC	G / par	divide by integer constant	B
DIVIV	G / par	divide by integer variable	B
DIVRC	F / par	divide by real constant	B
DIVRV	F / par	divide by real variable	B
DIVSIC	F / [par]	divide by small integer constant	B
DO	DO (par)	do	4.8.6.1
DOS	DOS (par)	dos	1.3.2 1.3.4 1.3.6 1.5.2 2.1.3.2 2.2.3.2 2.2.3.3 3.4.5 3.4.6 4.4 4.8.2 4.8.5 5.6.4
DOS2	DOS (par[2])	dos2	4.1.4 4.8.3.2
DOS3	DOS (par[3])	dos3	4.1.4 4.8.3.2
DPTR	S = D A = — ([par]—2), Z SUB (:DPTR)	display transport	6.2
END	GOTO (:ENDRUN)	end of program	7.5
ENTIER	SUBC (:ENTIER)	entier	10.4
ENTRB	B + [par] SUB2 (:ENTRB)	enter block	5.2
ENTRIS	SUB2 (:ENTRIS)	enter implicit subroutine	3.4.1 3.4.3 3.4.6
ENTRPB	B + [par] SUB2 (:ENTRPB)	enter procedure body	6.4
EQU	F — MC[—2], Z	equal	1.5.7.1

naam	instructies	omschrijving	verwijzingen
EQUIC	G — par, Z	equal to integer constant	B
EQUIV	G — par, Z	equal to integer variable	B
EQURC	F — par, Z	equal to real constant	B
EQURV	F — par, Z	equal to real variable	B
EQUSIC	F — [par], Z	equal to small integer constant	B
EXIT	GOTOR (MC[—1])	exit	4.8.5 5.6.3
EXITB	B = MD[[par]]	exit block	5.10 (zie ook 11.3)
EXITC	B = MD[[par]] SUBC (:FREE CHAIN)	exit block using counter stack	5.10
EXITIS	GOTO (:EXITIS)	exit implicit subroutine	3.4.1
EXITP	GOTO (:EXITP)	exit procedure	6.10
EXITPC	GOTO (:EXITPC)	exit procedure using counter stack	6.10
EXITSV	S = :MC[[par]] GOTO (MS)	exit subscripted variable	3.4.3 3.4.6 4.8.2
EXP	SUBC (:EXP)	exponential function	10.4
FAD	GOTO (:FAD)	finish address description	3.4.3 3.4.6
FADCV	GOTO (:FADCV)	finish address description of controlled variable	4.8.2
IAD	SUB3 (:IAD)	integer array declaration	5.5
IDI	SUBC (:IDI)	integer division	1.4.1
IJU	GOTO (par)	indirect jump	4.8.1
IJU1	GOTO (par[1])	indirect jump	10.2
IMP	Y, B — 1 N, S = — MC[—1], P	implies	1.6.1
INCRB	B + [par]	increase B	6.2
ISUBJ	SUBC (par)	indirect subroutine jump	10.3
JU	GOTO (:par)	jump	1.2 2.1.2 2.1.3.6 2.2.2 3.2 4.2 4.7 4.8.1 4.8.2 4.8.3 4.8.5 5.6.1 5.6.4 5.7

naam	instructies	omschrijving	verwijzingen
JU1	A = :par GOTO (:MA[1])	jump	3.4.2
JUA	GOTO (:JUA)	jump via A	4.2
LAD	S = [par] SUB (:LAD)	label declaration	5.3 6.5
LAST	– (d21 + :par)	last element of label list	5.3 6.5
LES	F – MC[–2], P Y, F – 0, P	less	1.5.7.1
LESIC	G – par, P N, F = F, Z S = – T, P	less than integer constant	B
LESIV	G – par, P N, F = F, Z S = – T, P	less than integer variable	B
LESRC	F – par, P N, F = F, Z S = – T, P	less than real constant	B
LESRV	F – par, P N, F = F, Z S = – T, P	less than real variable	B
LESSIC	F – [par], P N, F = F, Z S = – T, P	less than small integer constant	B
LN	SUBC (:LN)	natural logarithm	10.4
LNC	S = [par] linecounter = S	set linecounter	9.2
LOS	loose string = B	loose string	6.10
LST	F – MC[–2], Z N, S = – 1, E	at least	1.5.7.1
LSTIC	G – par, P N, F = F, Z	at least integer constant	B
LSTIV	G – par, P N, F = F, Z	at least integer variable	B
LSTRC	F – par, P N, F = F, Z	at least real constant	B
LSTRV	F – par, P N, F = F, Z	at least real variable	B

naam	instructies	omschrijving	verwijzingen
LSTSIC	F – [par], P N, F = F, Z	at least small integer constant	B
MOR	F – MC[–2], P N, F = F, Z S = – T, P	more	1.5.7.1
MORIC	G – par, P	more than integer constant	B
MORIV	G – par, P Y, F – 0, P	more than integer variable	B
MORRC	F – par, P	more than real constant	B
MORRV	F – par, P Y, F – 0, P	more than real variable	B
MORSIC	F – [par], P	more than small integer constant	B
MST	F – MC[–2], P N, F = F, Z	at most	1.5.7.1
MSTIC	G – par, Z N, S = – 1, E	at most integer constant	B
MSTIV	G – par, Z N, S = – 1, E	at most integer variable	B
MSTRC	F – par, Z N, S = – 1, E	at most real constant	B
MSTRV	F – par, Z N, S = – 1, E	at most real variable	B
MSTSIC	F – [par], Z N, S = – 1, E	at most small integer constant	B
MUL	F × MC[–2]	multiply	1.4.1
MULIC	G × par	multiply by integer constant	B
MULIV	G × par	multiply by integer variable	B
MULRC	F × par	multiply by real constant	B
MULRV	F × par	multiply by real variable	B
MULSIC	F × [par]	multiply by small integer constant	B
NEG	F = – F	invert sign	1.4.1

naam	instructies	omschrijving	verwijzingen
NIL	:par	element in label list	5.3 6.5
NON	S = – T, P	not	1.6.1
OBAD	SUB3 (:OBAD)	own Boolean array declaration	5.5
OIAD	SUB3 (:OIAD)	own integer array declaration	5.5
OR	Y, B – 1 N, S = MC[–1], P	or	1.6.1
ORAD	SUB3 (:ORAD)	own real array declaration	5.5
OSTAD	SUB3 (:OSTAD)	own string array declaration	5.5
QVL	S = T, P S = MC[–1], E	equivalent	1.6.1
RAD	SUB3 (:RAD)	real array declaration	5.5
REJST	SUBC (:REJST)	reject string	4.4 10.3
RLNC	S = par[2] linecounter = S	restore linecounter	9.2
SAS	stock3 = S	save S	3.4.6
SIN	SUBC (:SIN)	sine	10.4
SIGN	F = F, Z N, F = 1, E N, F = – 1	sign	10.4
SLNC	S = linecounter par[2] = S	save linecounter	9.2
SQRT	SUBC (:SQRT)	square root	10.4
SSTI	par = G	simple store integer	4.1.2 4.8.3 4.8.4 4.8.5
SSTSI	SUB2 (:SSTSI)	simple store subscripted integer	4.1.5
STAA	MC = A	stack A	4.1.5 (zie ook 11.3)
STAB	S = T MC = S	stack Boolean	1.6.1
STACK	MC = F	stack F	1.3.3.1 1.4.1 1.5.7.1 3.4.3 4.1.5 4.8.2 4.8.5 5.5 10.4

naam	instructies	omschrijving	verwijzingen
STAD	SUB3 (:STAD)	string array declaration	5.5
STB	S = T par = S	store Boolean	4.1.2
STFSU	SUB2 (:STFSU)	store formal subscripted of unknown type	4.1.5 4.8.3.3
STI	S = F, Z N, SUBC (:RND) par = G	store integer	4.1.2 4.8.3.1
STR	par = F	store real	4.1.2 4.8.3.1
STSB	SUB2 (:STSB)	store subscripted Boolean	4.1.5
STSI	SUB2 (:STSI)	store subscripted integer	4.1.5 4.8.3.3
STSR	SUB2 (:STSR)	store subscripted real	4.1.5 4.8.3.3
STSST	SUB2 (:STSST)	store subscripted string	4.1.5
STST	F = :par SUB2 (:STST)	store string	4.1.2
SUB	F = – F F + MC[–2]	subtract	1.4.1
SUBIC	G – par	subtract integer constant	B
SUBIV	G – par	subtract integer variable	B
SUBJ	SUBC (:par)	subroutine jump	1.3.5 3.4.3 3.4.6 4.4 4.8.2 4.8.3.3 4.8.5
SUBRC	F – par	subtract real constant	B
SUBRV	F – par	subtract real variable	B
SUBSIC	F – [par]	subtract small integer constant	B
SWP	MO[512×[par]] = B	set working space pointer	5.8 6.8
TAA	A = :par	take address of array key	5.5 6.6
TAK	A = par	take array key	1.3.3.1 3.4.3 4.1.5 4.8.2
TASB	GOTO (:TASB)	take actual subscripted Boolean	3.4.4
TASI	GOTO (:TASI)	take actual subscripted integer	3.4.4

naam	instructies	omschrijving	verwijzingen
TASR	GOTO (:TASR)	take actual subscripted real	3.4.3
TASST	GOTO (:TASST)	take actual subscripted string	3.4.4
TASU	GOTO (:TASU)	take actual subscripted of unknown type	3.4.5 3.4.6
TAV	SUB4 (:TAV)	transport array called by value	6.6
TBC	S = [par], Z	take Boolean constant	1.5.6
TBL	A = :MC F = :MD[[par]]	take block level	5.2
TBV	S = par, P	take Boolean variable	1.5.1 6.10
TCST	SUBC (:TCST)	take constant string	2.1.3.6
TDA	S = [par]	take dimension of array	5.5
TDL	F = :MA[[par]]	take display level	6.4
TEST1	F = MC[-2], Z	test controlled variable	4.8.5
TEST2	N, F = F, E N, F = F, Z	test controlled variable	4.8.5
TFD	S = MS[1]	take formal display pointer	3.4.2 10.2
TFSL	SUBC (:TFSL)	take formal subscripted label	2.2.3.3
TFSU	SUB2 (:TFSU)	take formal subscripted of unknown type	1.3.4 2.3.1
TIAV	SUB4 (:TIAV)	transport integer array called by value	6.6
TIC	G = par	take integer constant	1.3.7 3.4.7.2
TISCV	GOTO (:TISCV)	take integer subscripted controlled variable	4.8.2
TIV	G = par	take integer variable	1.3.1 4.8.5 6.10
TLV	A = :par	take label variable	2.2.3.1 2.3.2 3.4.7.2 5.6.4
TNA	F = [par]	take number of arrays	5.5
TNIC	G = - par	take negative integer constant	1.4.2.1

naam	instructies	omschrijving	verwijzingen
TNIV	G = – par	take negative integer variable	1.4.2.1
TNRC	F = – par	take negative real constant	1.4.2.1
TNRV	F = – par	take negative real variable	1.4.2.1
TNSIC	F = – [par]	take negative small integer constant	1.4.2.1
TRC	F = par	take real constant	1.3.7
TRSCV	GOTO (:TRSCV)	take real subscripted controlled variable	4.8.2
TRV	F = par	take real variable	1.3.1 4.8.5 6.10
TSB	SUB1 (:INDB) S '×' MA, Z	take subscripted Boolean	1.5.3
TSCVU	GOTO (:TSCVU)	take subscripted controlled variable of unknown type	4.8.2
TSI	SUB (:IND) G = MA	take subscripted integer	1.3.3.2
TSIC	F = [par]	take small integer constant	1.3.7 4.8.3 4.8.4 4.8.5
TSL	SUBC (:TSL)	take subscripted label	2.2.3.3
TSR	SUB (:IND) F = MA	take subscripted real	1.3.3.2
TSST	SUB (:IND) A = MA	take subscripted string	2.1.3.3
TSTV	A = par	take string variable	2.1.3.1 6.10
TSWE	A = :par	take switch entry	2.2.3.3 11.2
TTP	SUBC (:TTP)	to the power	1.4.1
UQU	F – MC[–2], Z S = – T, P	unequal	1.5.7.1
UQUIC	G – par, Z S = – T, P	unequal to integer constant	B
UQUIV	G – par, Z S = – T, P	unequal to integer variable	B
UQURC	F – par, Z S = – T, P	unequal to real constant	B

naam	instructies	omschrijving	verwijzingen
UQURV	F – par, Z S = – T, P	unequal to real variable	B
UQUSIC	F – [par], Z S = – T, P	unequal to small integer constant	B
YCOJU	Y, GOTO (:par)	yes–conditional jump	4.8.4 4.8.5

Appendix B

Uit de in deze appendix gegeven tabel kan men de macro-naam aflezen van de macro's die ontstaan door optimalisatie (zie 1.4.2.2. en 1.5.7.2.) van macro-tripels gevormd door de macro STACK, een van de macro's TRV, TIV, TRC, TIC of TSIC en een van de macro's ADD, SUB, MUL, DIV, EQU, UQU. LES, MST, MOR of LST. Met behulp van de macro-naam kunnen vervolgens in appendix A de bijbehorende instructies opgezocht worden.

	TRV	TIV	TRC	TIC	TSIC
ADD	ADDRV	ADDIV	ADDRC	ADDIC	ADDSIC
SUB	SUBRV	SUBIV	SUBRC	SUBIC	SUBSIC
MUL	MULRV	MULIV	MULRC	MULIC	MULSIC
DIV	DIVRV	DIVIV	DIVRC	DIVIC	DIVSIC
EQU	EQURV	EQUIV	EQURC	EQUIC	EQUSIC
UQU	UQURV	UQUIV	UQURC	UQUIC	UQUSIC
LES	LESRV	LESIV	LESRC	LESIC	LESSIC
MST	MSTRV	MSTIV	MSTRC	MSTIC	MSTSIC
MOR	MORRV	MORIV	MORRC	MORIC	MORSIC
LST	LSTRV	LSTIV	LSTRC	LSTIC	LSTSIC

Appendix C

Alle karakters van een ALGOL-tekst worden door de vertaler geconverteerd naar een semi-interne representatie die gegeven wordt door onderstaande tabel. In strings is de conversie daarmee voltooid, buiten strings vindt een assemblage tot basic symbols in interne representatie plaats. Deze laatste komt in het objectprogramma nergens voor.

karakter	representatie	karakter	representatie
0	0	10	89
...	...	:	90
9	9	;	91
a	10	spatie	93
...	...	(	98
z	35	)	99
A	37	[	100
...	...	]	101
Z	62	tab	118
+	64	twnr of LF	119
-	65	'	120
×	66	"	121
/	67	?	122
↑ 1)	69	&	123
=	70	↓ 2)	124
<	72	#	125
>	74		126
¬	76	⌈	127
∨	79	@	128
∧	80	!	129
,	87	%	132
.	88	$	133

1) ISO-code 5.14
2) ISO-code 6.0

Appendix D

In onderstaande tabel is het verband gegeven tussen actuele-parametersoort, APD, en APIC. CEN (zie 6.3.1. en 6.3.2.) construeert uit de APD (zie 3.) een APIC van twee woorden, CLPN (zie 6.3.2.) een APIC van vier woorden. Het eerste en tweede woord van een APIC worden zowel door CEN als door CLPN geconstrueerd, APIC 2 en APIC 3 uitsluitend door CLPN; voor de meeste soorten actuele parameters is echter verwerking van de APD door CLPN niet toegestaan. Met [dp] wordt de display pointer op moment van aanroep van de procedure aangeduid, met (lw|0|1) de expressie "if lw then 0 else 1".

actuele parameter	APD	APIC
real variable identifier (x)	(0 × d20 + adres x)	0 F = M[adres x] 1 (0 × d20 + [dp]) (2 A = [adres x]) (3 M[adres x] = F)
integer variable identifier (i)	(1 × d20 + adres i)	0 G = M[adres i] 1 (1 × d20 + [dp]) (2 A = [adres i]) (3 SUBC (:STAI))
Boolean variable identifier (b)	(2 × d20 + adres b)	0 S = M[adres b], P 1 (2 × d20 + [dp]) (2 A = [adres b]) (3 SUBC (:STABO))
string variable identifier (st)	(3 × d20 + adres st)	0 A = M[adres st] 1 (3 × d20 + [dp]) (2 A = [adres st]) (3 SUB2 (:STAST))
floating point constante (fp)	(4 × d20 + adres fp)	0 F = M[adres fp] 1 (4 × d20 + [dp])
integer constante (n)	(5 × d20 + adres n)	0 G = M[adres n] 1 (5 × d20 + [dp])
logische waarde (lw)	(6 × d20 + (lw|0|1))	0 S = (lw|0|1), P 1 (6 × d20 + [dp])
kleine positieve integer (k)	(7 × d20 + k)	0 F = [k] 1 (7 × d20 + [dp])
real array identifier (ar)	(8 × d20 + adres ar)	0 A = M[adres ar] 1 (8 × d20 + [dp])
integer array identifier (ai)	(9 × d20 + adres ai)	0 A = M[adres ai] 1 (9 × d20 + [dp])
Boolean array identifier (ab)	(10 × d20 + adres ab)	0 A = M[adres ab] 1 (10 × d20 + [dp])
string array identifier (ast)	(11 × d20 + adres ast)	0 A = M[adres ast] 1 (11 × d20 + [dp])

actuele parameter	APD	APIC
switch identifier (sw)	(12 × d20 + adres sw)	0 A = [adres sw] 1 (12 × d20 + [dp])
kleine negatieve integer (– k)	(13 × d20 + k)	0 F = – [k] 1 (13 × d20 + [dp])
label identifier (lb)	(14 × d20 + adres lb)	0 A = [adres lb] 1 (14 × d20 + [dp])
formele identifier (f)	(15 × d20 + adres f)	0 [copie van APIC0] 1 [copie van APIC1] (2 [bijbehorende APIC2]) (3 [bijbehorende APIC3])
element van real array	(16 × d20 + adres ISR)	0 SUBC (:ISR) 1 (16 × d20 + [dp]) (2 SUBC (:ISR[2])) (3 SUB3 (:STASR))
element van integer array	(17 × d20 + adres ISR)	0 SUBC (:ISR) 1 (17 × d20 + [dp]) (2 SUBC (:ISR[2])) (3 SUB3 (:STASI))
element van Boolean array	(18 × d20 + adres ISR)	0 SUBC (:ISR) 1 (18 × d20 + [dp]) (2 SUBC (:ISR[2])) (3 SUB3 (:STASB))
element van string array	(19 × d20 + adres ISR)	0 SUBC (:ISR) 1 (19 × d20 + [dp]) (2 SUBC (:ISR[2])) (3 SUB3 (:STASST))
niet-assigneerbare arithmetische expressie	(20 × d20 + adres ISR)	0 SUBC (:ISR) 1 (20 × d20 + [dp])
niet-assigneerbare Boolean expressie	(22 × d20 + adres ISR)	0 SUBC (:ISR) 1 (22 × d20 + [dp])
niet-assigneerbare string expressie	(23 × d20 + adres ISR)	0 SUBC (:ISR) 1 (23 × d20 + [dp])
real procedure identifier	(24 × d20 + adres ISR)	0 SUBC (:ISR) 1 (24 × d20 + [dp])
integer procedure identifier	(25 × d20 + adres ISR)	0 SUBC (:ISR) 1 (25 × d20 + [dp])
Boolean procedure identifier	(26 × d20 + adres ISR)	0 SUBC (:ISR) 1 (26 × d20 + [dp])
string procedure identifier	(27 × d20 + adres ISR)	0 SUBC (:ISR) 1 (27 × d20 + [dp])

actuele parameter	APD	APIC
designational expressie	(28 × d20 + adres ISR)	0 SUBC (:ISR) 1 (28 × d20 + [dp])
integer constante of designational expr.	(29 × d20 + adres ISR)	0 SUBC (:ISR) 1 (29 × d20 + [dp])
procedure identifier	(30 × d20 + adres ISR)	0 SUBC (:ISR) 1 (30 × d20 + [dp])
niet-assigneerbare expr. van onbekend type	(31 × d20 + adres ISR)	0 SUBC (:ISR) 1 (31 × d20 + [dp])
mogelijk-assigneerbare expressie van onbekend type	(32 × d20 + adres ISR)	0 SUBC (:ISR) 1 (32 × d20 + [dp]) (2 SUBC (:ISR[2])) (3 SUB3 (:STASU))

De in 11.1. behandelde variant verandert het geval van de switch identifier als volgt:

switch identifier	(12 × d20 + adres ISR)	0	DO (ISR)
		1	(12 × d20 + [dp])

De in 11.2. behandelde variant verandert het totale beeld als volgt:
CEN construeert, op grond van de APD, een APIC van drie woorden. Indien het rangnummer van de APD het rangnummer van een doorgegeven formele is (15), worden drie woorden uit een vroegere, door CEN of CLPN geconstrueerde APIC gecopieerd. In alle andere gevallen gelden de volgende regels:
a) CEN vult de bits d17 t/m d0 van APIC 2 steeds met het (zonodig van dynamisch tot statisch herleide) adres uit de APD. Van de bits d26 t/m d18 worden alleen bit d22 en d19 gelijk aan een gemaakt ('0220 00000' is het bitpatroon van de opdracht A = :M[0]),
b) CEN vult APIC 1 als vanouds met [(codebits uit APD), (heersende dp)],
c) als het getal gevormd door de bits d17 t/m d0 van APIC 2 kleiner is dan 32768 wordt APIC 0 als vanouds ingevuld; anders wordt APIC 0 door CEN ingevuld met:

rangnummer APD	APIC 0	toelichting
0	SUB (:TAR)	take actual real variable
1	SUB (:TAI)	take actual integer variable
2	SUB (:TABO)	take actual Boolean variable
3	SUB (:TAST)	take actual string variable
8, 9, 10, 11	SUB (:TAA)	take array address
14	A = MS[2]	take address label variable

Bij de niet-vermelde rangnummers kan de situatie dat d17 t/m d0 van APIC 2 een getal > 32768 bevatten zich niet voordoen zolang het objectprogramma zich niet boven de 32 K mag uitstrekken.
CLPN maakt als vanouds een APIC van vier woorden. APIC 0 en APIC 1 worden door CEN gemaakt. Voor de rangnummers 16, 17, 18, 19 en 32 is de inhoud van APIC 2 als vanouds SUBC (:ISR[2]), terwijl APIC 3 als vanouds respectievelijk

de instructie SUB3 (:STASR), SUB3 (:STASI), SUB3 (:STABO), SUB3 (:STASST) of SUB3 (:STASU) bevat. Voor de rangnummers 0, 1, 2 en 3 wordt APIC 2 door CEN gemaakt. De bits d17 t/m d0 bevatten dus het (statische) adres van de variabele, de waarde van de overige bits zorgen ervoor dat een executie van APIC 2 snel en ongevaarlijk is (ten hoogste wordt, behalve de inhoud van A, ook de conditie C gewijzigd). De inhoud van APIC 3 is voor de rangnummers 1, 2 en 3 als vanouds de instructie SUBC (:STAI), SUBC (:STABO) of SUBC (:STAST) respectievelijk. Voor het rangnummer 0 wordt, indien het adres in de bits d17 t/m d0 van APIC 2 kleiner dan 32768 is als vanouds de instructie M[0] = F, passend gemodificeerd met dat adres, gegenereerd en anders de instructie SUBC (:STAR). Deze instructie wordt in APIC 3 geschreven.