RA

stichting
mathematisch
centrum

MC

REKENAFDELING RA CR 25/72 JUNI

J.C. VAN VLIET en D.T. WINTER
EEN FORTRAN SYNTAX VOOR CDC 6000 FORTRAN EXT.

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.

0. Inleiding.

0.0. Wat betekent FORTRAN.

FORTRAN betekent FORmula TRANslation.

0.1. Hoe ziet een FORTRAN programma eruit.

Beschreven wordt de CDC 6000 versie van FORTRAN IV.
Een FORTRAN programma bestaat uit verschillende subprogramma's, waaronder minstens 1 hoofdprogramma. Elk subprogramma bestaat uit statements, al of niet genummerd.
Voor elk subprogramma staat een header (voor identificatie, hierin staan: naam, type subprogramma en eventuele parameters).

Een statement of header staat op 1 kaart, in kolom 7 t/m 72. In kolom 6 staat een 0 of een spatie, in kolom 1 t/m 5 het eventuele statement number. Past de statement of header niet op 1 kaart, dan kan op de volgende kaart(en) in kolom 7 t/m 72 verder gegaan worden, met in kolom 6 een symbool ongelijk aan 0 of spatie. (Er mogen voor 1 statement of voor de header maximaal 20 kaarten worden gebruikt, waartussen geen blanco- of commentkaarten mogen voorkomen).

Commentaar mag verder overal voorkomen. Dit commentaar bestaat uit een kaart met in kolom 1 een C, $ of *. Kolom 73 t/m 80 worden beschouwd als niet behorende tot de FORTRAN-tekst. Indien op 1 kaart meer dan 1 statement past, kan de $ als statement separator worden gebruikt. Het eerste symbool na de $ wordt dan beschouwd als staande in kolom 7 van een nieuwe kaart. (Dit mag niet bij de END-statement). Spaties zijn overal zonder betekenis toegestaan, behalve in een hollerith constante.

0.2. Hoe ziet de syntax eruit.

De syntax is beschreven in de Backus-notatie, (J.W. Backus, The syntax and semantics of the proposed international algebraic language of the Zürich ACM-GAMM conference-IFIP, June 1959) met toevoeging van:
a) de metanotion OPTION.
b) verwijzingen naar andere definities tussen [en].

0.3. FORTRAN symbol set.

a. <FORTRAN symbol>::= <FORTRAN character>|<word symbol>

b. <FORTRAN character>::= <letter>|<digit>|<special character>

c. <letter>::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

d. <digit>::= 0|1|2|3|4|5|6|7|8|9

e. <special character>::= =|+|-|*|/|(|)|,|.|$|<space>

f. <word symbol>::= <header indicator>|<subprogram link>|<indicator>|
<sequential controller>|<io indicator>|<relational operator>|
<logical operator>|<logical value>|<file>|<other>

g. <header indicator>::= PROGRAM|SUBROUTINE|FUNCTION|ENTRY

h. <subprogram link>::= BLOCK DATA|EXTERNAL|CALL|RETURNS|RETURN

i. <indicator>::= TYPE|INTEGER|REAL|DOUBLE PRECISION|LOGICAL|COMPLEX|
ECS|DIMENSION|COMMON|EQUIVALENCE|DATA

j. <sequential controller>::= GO TO|IF|DO|END

k. <io indicator>::= FORMAT|READ|WRITE|PRINT|PUNCH|BUFFER IN|
BUFFER OUT|ENCODE|DECODE|ENDFILE|REWIND|BACKSPACE|NAMELIST

l. <relational operator>::= .EQ.|.NE.|.GT.|.GE.|.LT.|.LE.

m. <logical operator>::= .NOT.|.AND.|.OR.

n. <logical value>::= .TRUE.|.FALSE.

o. <file>::= INPUT|OUTPUT|PUNCH|TAPE

p. <other>::= ASSIGN|TO|CONTINUE|PAUSE|STOP

semantiek:

i. DOUBLE PRECISION mag afgekort worden tot DOUBLE

m. .NOT.,.AND. en .OR. mogen afgekort worden tot respectievelijk .N., .A. en .O.

n. .TRUE. en .FALSE. mogen afgekort worden tot respectievelijk .T. en .F.

o. Door middel van controle kaarten kan het aantal files uitgebreid worden, echter alleen voor gebruik in overlay header.

1. Namen, constanten, variabelen en statement numbers

1.1 Namen

a. <integer name>::= <name>
b. <array name>::= <name>
c. <subroutine name>::= <name>
d. <function name>::= <name>
e. <program name>::= <name>
f. <name>::= <letter[0.3.c]>|<name><letter[0.3.c]>|<name><digit[0.3.d]>

Semantiek:

f. Een naam bestaat uit maximaal 7 symbolen.

Voorbeelden:

f. A, I, I12, A1BCDE2

1.2 Constanten

a. <constant>::= <hollerith constant>|<complex constant>|
<double precision constant>|<real constant>|<integer constant>|
<logical constant>|<octal constant>
b. <hollerith constant>::= <integer constant><hollerith specificator>
<string>
c. <hollerith specificator>::= H|L|R
d. <string>::= <FORTRAN character[0.3.b]><string OPTION>
e. <complex constant>::= (<sign OPTION><real constant>,<sign OPTION>
<real constant>)
f. <double precision constant>::= <integer constant>.
<decimal fraction OPTION><double precision exponent part>
g. <double precision exponent part>::= D<sign OPTION><integer constant>
h. <real constant>::= <integer constant>.<decimal fraction OPTION>
<exponent part OPTION>

i. <exponent part>::= E<sign OPTION><integer constant>

j. <decimal fraction>::= <integer constant>

k. <integer constant>::= <digit[0.3.d]><integer constant OPTION>

l. <sign>::= +|-

m. <logical constant>::= <logical value[0.3.n]>

n. <octal constant>::= <octal number>B

o. <octal number>::= <octal digit><octal number OPTION>

p. <octal digit>::= 0|1|2|3|4|5|6|7

Semantiek:

b. en c. De integer in een Hollerith constant geeft het aantal karakters in een string aan. Er gaan 10 karakters in een machinewoord; is het aantal karakters in een string geen veelvoud van 10 dan gebeurt bij de verschillende specificaties het volgende:

H: de string wordt aan het eind opgevuld met spaties.

L: (left justified) het laatste woord wordt opgevuld met binaire nullen

R: (right justified) de string wordt verschoven, totdat het laatste karakter aan het eind van het laatste woord staat, waarna het begin van het eerste woord wordt gevuld met binaire nullen.

Merk op dat H en L niet identiek zijn, want een spatie wordt niet voorgesteld door binaire nullen.

In een expressie is het aantal karakters in de string tot 10 beperkt.

f. en h. Voor de exponent moet gelden: $-308 \leq \text{exponent} \leq +337$.

Voor de constante moet gelden: $\pm\ 10^{-293} \leq \text{constante} \leq \pm\ 10^{+322}$

Afkortingen:

1° .E mag geschreven worden als E

2° .D mag geschreven worden als D

3° 0.nE mag geschreven worden als .nE

4° 0.nD mag geschreven worden als .nD

Precisie: enkele lengte: 14 à 15 cijfers (48 bits)
dubbel lengte: 28 à 29 cijfers (96 bits)

k. Een integer constant moet kleiner zijn dan $2 \uparrow 59$.

Bij een DO en als subscript moet de constante kleiner zijn dan $2 \uparrow 17 - 1$ (≤ 131070)

o. Een octal number bestaat uit 20 octale cijfers.
Nullen aan het begin mogen weggelaten worden.

Ruimte in beslaggenomen door constanten:
Integer, real, logical en octal: 1 machinewoord (60 bits)
Double precision en complex: 2 machinewoorden
Hollerith: entier ((aantal karakters + 9)/10) machinewoorden.

Voorbeelden:

b. 5 HALLEN, 3LX1Y, 4R+*=,
e. (-3.21, 4.E+2)
h. 5.4, 4.2E-3, 3.
k. 1,205, 32768
m. .TRUE., .F.
n. 7657B, OB

1.3. Variabelen

a. <integer variable> ::= <variable>
b. <variable>::= <name [1.1.f]><subscript part OPTION>
c. <array element>::= <name [1.1.f]><subscript part>
d. <subscript part>::= (<subscript list>)
e. <subscript list>::= <arithmetic expression [2.7.1.a]>|
<arithmetic expression [2.7.1.a]>,<subscript list>

Semantiek:

b. Het subscript deel mag slechts voorkomen als de naam van de variabele in een dimension indication voorkomt. Komt de naam in een dimension indication voor, en worden één of meer [eventueel alle] van de subscripts weggelaten, dan wordt voor de ontbrekende subscripts een 1 ingevuld.

Voorbeelden:

b. I, A(I,J), A(2*J+1, 3*SQRT(3.14))

1.4. Statement numbers.

a. <statement number>::= <integer constant [1.2.k]>

Semantiek:

a. Een statement number bestaat uit maximaal 5 cijfers, niet alle gelijk aan 0.

2. Implied do-loop, parameter packs, statement number pack, function designator en expressions.

2.1. Implied do-loop.

a. <implied do loop>::= (<loop element list>,<integer name [1.1.a]>= <start><finish>)
b. <loop element list>::= <loop element>|<loop element> ,<loop element list>
c. <loop element>::= <implied do loop>|<variable [1.3.c]>
d. <start>::= <integer constant [1.2.k]>|<integer name [1.1.a]>
e. <finish>::= ,<integer constant [1.2.k]>|,<integer name [1.1.a]>
f. <step>::= ,<integer constant [1.2.k]>|,<integer name [1.1.a]>

Voorbeelden:

a. (A(I),I=1,10), (I,(B(I,J),J=1,5),I=1,10)

2.2. Formal parameter pack.

a. <formal parameter pack>::= (<formal parameter list>)
b. <formal parameter list>::= <name [1.1.f]>| <name [1.1.f]>,<formal parameter list>

Semantiek:

b. Het aantal parameters is maximaal 63.

Voorbeelden:

a. (A,B,I)

2.3. Actual parameter pack

a. <actual parameter pack>::= (<actual parameter list>)
b. <actual parameter list>::= <actual parameter>|
<actual parameter>,<actual parameter list>
c. <actual parameter>::= <name [1.1.f]>|<expression [2.7.a]>

Voorbeelden:

a. (A,2,SQRT(3.14))

2.4. Formal statement number pack.

a. <formal statement number pack>::= (<formal statement number list>)
b. <formal statement number list>::= <integer name [1.1.a]>|
<integer name [1.1.a]>, <formal statement number list>

Voorbeelden:

a. (A,I,J)

2.5. Actual statement number pack

a. <actual statement number pack>::= (<actual statement number list>)
b. <actual statement number list>::= <statement number [1.4.a]>|
<statement number [1.4.a]>, <actual statement number list>

Voorbeelden:

a. (10,20,30)

2.6. Function designator.

a. <function designator>::= <function name [1.1.d]>
<actual parameter pack [2.3.a]>

Voorbeelden:

a. DET(A,10,B)

2.7. Expressions.

a. <expression>::= <arithmetic expression>|<relational expression>|
<logical expression>|<masking expression>

2.7.1. Arithmetic expression

a. <arithmetic expression>::= <adding operator OPTION><term>|
<arithmetic expression><adding operator><term>
b. <term>::= <factor>|<term><multiplying operator><factor>
c. <factor>::= <primary>|<factor>**<primary>
d. <primary>::= <unsigned number>|<variable [1.3.b]>|
<function designator [2.6.a]>|(<arithmetic expression>)
e. <unsigned number>::= <integer constant [1.2.k]>|
<real constant [1.2.h]>|<double precision constant [1.2.f]>|
<hollerith constant [1.2.b]>|<octal constant [1.2.p]>|
<complex constant [1.2.e]>
f. <adding operator>::= +|-
g. <multiplying operator>::= *|/

Semantiek:

a. De mode van een arithmetische expressie wordt bepaald door de volgende tabellen:

+-*/	integer	real	double prec.	complex	octal	hollerith
integer	integer	real	double prec.	complex	integer	integer
real	real	real	double prec.	complex	real	real
double prec.	double prec.	double prec.	double prec.	complex	double prec.	double prec.
complex	complex	complex	complex	complex	complex	complex
octal	integer	real	double prec.	complex	integer	integer
hollerith	integer	real	double prec.	complex	integer	integer

**	integer	real	double prec.	complex	octal	hollerith
integer	integer	real	double prec.	complex	X	X
real	real	real	double prec.	complex	X	X
double prec.	double prec.	double prec.	double prec.	complex	X	X
complex	complex	X	X	X	X	X
octal	X	X	X	X	X	X
hollerith	X	X	X	X	X	X

Verboden is:

1. Het verheffen van een negatief getal tot een real, double precision of complexe exponent
2. 0**0
3. oneindige of niet- gedefinieerde operand
4. een element met mathematisch ongedefinieerde waarde, bijv. 1/0

Onder zekere voorwaarden zijn 1 t/m 3 niet fataal.

Een function designator kan beschouwd worden als zijnde van prioriteit 7.

De evaluatie van een expressie is in de regel van links naar rechts, waarbij steeds van de kleinste eenheid opnieuw de mode wordt bepaald (dus 3.14 + 11/3 = 6.14)

c. De prioriteit van ** is 7.

f. De prioriteit van + en - is 5.

g. De prioriteit van * en / is 6.

Voorbeelden:

a. -A*B+C/D**3.14+SQRT(2./4.37)

2.7.2. Relational expression.

a. <relational expression>::= <arithmetic expression [2.7.1.a]>
<relational operator [0.3.1]>
<arithmetic expression [2.7.1.a]>

Semantiek:

a. De arithmetische expressies mogen van mixed mode zijn. Bij complexe elementen wordt alleen het reële deel bekeken, behalve bij .EQ. en .NE.. De prioriteit van de relational operators is 4.

Voorbeelden:

a. A+B*C.NE.10

2.7.3. Logical expression.

a. <logical expression>::= <logical factor>|<logical expression>.OR.
<logical factor>
b. <logical factor>::= <logical secondary>|<logical factor>.AND.
<logical secondary>
c. <logical secondary>::= <logical primary> .NOT.<logical primary>
d. <logical primary>::= <logical constant [1.2.m]>|<variable [1.3.b]>|
<function designator [2.6.a] >|
<relational expression [2.7.2.a]>|(<logical expression>)

Semantiek:

a. .OR. heeft de prioriteit 1.
b. .AND. heeft de prioriteit 2.
c. .NOT. heeft de prioriteit 3.

Voorbeelden:

B.AND. .NOT.C.OR.(D.NE.E)

2.7.4. Masking expression.

a. <masking expression>::= <masking factor>|<masking expression>.OR.
<masking factor>
b. <masking factor>::= <masking secondary>|<masking factor>.AND.
<masking secondary>
c. <masking secondary>::= <masking primary>|.NOT. masking primary
d. <masking primary>::= <arithmetic expression [2.7.1.a]>|
(<masking expression>)

Semantiek:

a. Een masking expression is een bit voor bit operatie, en kan gezien worden als een logische operatie tussen bits, waarbij 1 overeenkomt met .TRUE. en 0 met .FALSE.. Een masking expression bezit geen mode.

Voorbeelden:

a. B.AND.C.OR.D

3. Indications.

a. <indication>::= <first type indication>|<second type indication>|
<format indication [5.a]>

b. <first type indication>::= <mode indication>|<dimension indication>|
<common indication>|<equivalence indication>|
<external indication>

c. <second type indication>::= <data indication>|<namelist indication>|
<function indication>

3.1. Mode indication.

a. <mode indication>::= <statement number [1.4.a] OPTION><type OPTION>
<mode indicator><mode name list>

b. <type>::= TYPE

c. <mode indicator>::= INTEGER|REAL|DOUBLE PRECISION|COMPLEX|LOGICAL|
ECS

d. <mode name list>::= <name [1.1.f]>|<name [1.1.f]>,<mode name list>

Semantiek:

a. De mode van een variabele,function of array wordt bepaald door de mode indication, waarin de naam voorkomt. Staat de naam in geen enkele mode indication, dan worden variabelen,functions en arrays wier naam begint met een der letters I t/m N geacht te zijn van mode INTEGER, en alle andere variabelen, functions en arrays van mode REAL. ECS mode mogen alleen zijn variabelen en arrays. Deze mogen slechts voorkomen als formele en actuele parameter, in een dimension statement en in een common met naam, mits alle andere variabelen en array's in het common ook van mode ECS zijn (zie ook [3.3.c])

Voorbeelden:

a. 100 REAL I,J
TYPE DOUBLE P

3.2. Dimension indication.

a. <dimension indication>::= <statement number [1.4.a] OPTION>
DIMENSION <array list>

b. <array list>::= <array name [1.1.b]><subscript part [1.3.d]>|
<array name [1.1.b]><subscript part [1.3.d]>,<array list>

Semantiek:

a. Staat een naam in een mode indication en tevens in een dimension indication, dan mag het subscript part uit de dimension indication achter de naam in de mode indication gezet worden, waarna de naam en subscript part uit de dimension indication weggelaten mag worden.

b. De subscripts mogen slechts integer constanten of integer name's zijn, en dit laatste alleen als:
1° de arrayname en de integer name's in de formele parameter lijst staan.
2° de integer name's gedurende executie van het subprogramma niet van waarde veranderen.

Voorbeelden:

a. DIMENSION A(10,10),B(N,MAX)

3.3. Common indication.

a. <common indication>::= <statement number [1.4.a] OPTION > COMMON
<common pack>

b. <common pack>::= <common><common pack OPTION>

c. <common>::= / <common name OPTION> / <common list>

d. <common name>::= <name [1.1.f]>|<integer constant [1.2.k]>

e. <common list>::= <common entry>|<common entry>,<common list>

f. <common entry>::= <name [1.1.f]>|<array name [1.1.b]>
<subscript part [1.3.d]OPTION>

Semantiek:

c. COMMON // mag afgekort worden tot COMMON.
Er mag maar een common zijn voor ECS mode variabelen en array's.
Bij een common, waarvoor een naam is gedefinieerd, dient in ieder subprogramma de lengte in machine woorden gelijk te zijn. Indien in een subprogramma twee of meer commons met dezelfde naam worden gedefinieerd, dan worden deze gezamenlijk beschouwd als een common.
Ook: er is slechts een common zonder naam, z.g. blank common.

d. De integer constante mag slechts 7 cijfers bevatten.

f. Volgt achter de array name een subscript part, dan moet deze identiek zijn aan die in de dimension indication, en mag de dimension indication voor het betreffend array weggelaten worden.

Voorbeeld:

a. COMMON /NAME/A(1),B,I/P/C,D

3.4. Equivalence indication.

a. <equivalence indication>::= <statement number [1.4.a] OPTION>
EQUIVALENCE <equivalence pack list>

b. <equivalence pack list>::= <equivalence pack>|
<equivalence pack>,<equivalence pack list>

c. <equivalence pack>::= (<equivalence list>)

d. <equivalence list>::= <equivalence element>,<equivalence element>|
<equivalence element>,<equivalence list>

e. <equivalence element>::= <variable [1.3.b]>

Semantiek:

a. Een equivalence mag noch een common veranderen, noch een common naar voren uitbreiden. Wordt namelijk een array element equivalent gemaakt

aan een variabele in een common, dan wordt het hele array beschouwd als behorend tot het common (en telt dus eventueel ook mee bij de lengte in machinewoorden).

e. Is de variabele een array element, dan moet een subscript part volgen, met één of alle subscripts. Staat er maar een subscript, dan worden de elementen van het array geacht lineair geordend te zijn. (de eerste subscript loopt het snelst!). De subscipts moeten integer constanten zijn.

Voorbeelden:

101 EQUIVALENCE (A,B),(P,Q,R)

3.5. External indication.

a. <external>::= <statement number [1.4.a] OPTION> EXTERNAL
<external list>

b. <external list>::= <procedure name>|<procedure name>,
<external list>

c. <procedure name>::= <function name [1.1.d]>|
<subroutine name [1.1.c]>

Semantiek:

c. De function name moet de naam zijn van een function subprogram.

Voorbeelden:

a. EXTERNAL DET, INV, DETINV

3.6. Data indication.

a. <data indication>::= <statement number [1.4.a] OPTION>DATA<data list>

b. <data list>::= <type one data list>|<type two data list>

c. <type one data list>::= <data elements>/<list of constants>/|
<data elements>/<list of constants>/,<type one data list>

d. <data elements>::= <data element>|<data element>,<data elements>

e. <data element>::= <variable[1.3.b]>|<implied do loop [2.1.a]>

f. <type two data list> ::= (<array> = <list of constants>)|
(<array> = <list of constants>), <type two data list>

g. <array>::= <array name [1.1.f]>|<implied do loop [2.1.a]>

h. <list of constants>::= <constant list element>|
<constant list element>,<list of constants>

i. <constant list element>::= <repetition factor OPTION>
<constant element>|<repetition factor OPTION>
(<list of constants>)

j. <repetition factor>::= <integer constant [1.2.k]>*

k. <constant element>::= <hollerith constant [1.2.b]>|
<sign [1.2.1] OPTION> <double precision constant [1.2.f]>|
<sign [1.2.1] OPTION> <real constant [1.2.h]>|
<sign [1.2.1] OPTION> <integer constant [1.2.k]>|
<logical constant [1.2.m]>|<octal constant [1.2.n]>

Semantiek:

a. Een data indication heeft alleen effect na vertaling, doch voor executie.

c. en f. De overeenkomst tussen constanten en variabelen (of array elementen) is een eenduidig in machinewoorden.

e. en g. De variabelen en array's mogen niet in blank common voorkomen. Variabelen en array's in een ander common mogen alleen geïnitialiseerd worden in het BLOCK DATA subprogram.

e. In de implied do loop zijn alleen de volgende producties van loop element toegestaan:
1° (<implied do loop>); 2° <variable>, waarbij deze laatste subscripted moet zijn.

Voorbeelden:

a. 1 DATA C, (A(I),I=1,10), D(1)/3 2*(1.,3.), 0.,6*5./
2 DATA (A=2*(1.,3.),0.,5*5.)

3.7. Namelist indication.

a. <namelist indication>::= <statement number [1.4.a] OPTION>
NAMELIST <namelist list>

b. <namelist list>::= /<namelist name>/<namelist><namelist list OPTION>

c. <namelist name>::= <name [1.1.f]>

d. <namelist>::= <namelist element>|<namelist element>,<namelist>

e. <namelist element>::= <variable [1.3.b]>|<array name [1.1.b]>

Semantiek:

a. De namelist is ingevoerd om bij input/output operaties de overdracht te kunnen uitvoeren:
   1 bij invoer naar bepaalde, in de invoer, bij naam genoemde variabelen.
   2 bij uitvoer op een zodanige wijze, dat uitvoer en invoer compatible zijn. (zie unformatted io [6.2.a])

e. Alleen arrays met constante grenzen mogen genoemd worden.

Voorbeelden:

a. NAMELIST/NAME/A,D(1),C

3.8. Function indication.

a. <function indication>::= <statement number [1.4.a] OPTION>
<function name [1.1.d]> <formal parameter pack [2.2.a]>
= <expression [2.7.a]>

4. Statements.

a. <statement>::= <assignment statement>|<goto statement>|
<assign statement>|<if statement>|<do statement>|
<continue statement>|<call>|<pause statement>|
<stop statement>|<entry statement>|<io statement [6.a]>

4.1. Assignment statement.

a. <assignment statement>::= <arithmetic assignment>|
<logical assignment>|<masking assignment>

b. <arithmetic assignment>::= <statement number [1.4.a] OPTION>
<left part><arithmetic expression [2.7.1.a]>

c. <logical assignment>::= <statement number [1.4.a] OPTION><left part>
<logical expression [2.7.3.a]>

d. <masking assignment>::= <statement number [1.4.a] OPTION><left part>
<masking expression [2.7.4.a]>

e. <left part>::= <left part OPTION><variable [1.3.b]>=

Semantiek:

a. Het left part moet bij een logical assignment een logische variabele zijn, anders een niet-logische variabele.

b en e. Na evaluatie van de expressie vindt zonodig mode-conversie plaats. Een assignment aan meerdere variabelen wordt beschouwd als een opeenvolging van assignments, eerst aan de meest rechtse, enz.

=	integer	real	double prec.	complex	hollerith, octal, masking expr.
integer	direct	truncation	truncation	truncation reële deel	bit voor bit
real	direct	truncation	meest sign. woord	reële deel direct	bit voor bit
double prec.	direct naar meest sign.woord,rest=0	idem ←	direct	reële deel naar meest sign. woord rest=0	meest sign.woord, rest=0
complex	direct naar reële deel, im.deel = 0	idem ←	meest sign.woord naar reële deel, rest=0	direct	bit voor bit naar reële deel, rest=0

Tabel voor mode conversie.

Voorbeelden:

b. I(1)= 2*N

c. A=B.A.C

d. F=B.OR.C

4.2. Goto statement.

a. <goto statement>::= <unconditional goto>|<assigned goto>|
<computed goto>

b. <unconditional goto>::= <statement number [1.4.a] OPTION> GO TO
<statement number [1.4.a]>

c. <assigned goto>::= <statement number [1.4.a] OPTION> GO TO <integer
name [1.1.a]>,<actual statement number pack [2.5.a]>

d. <computed goto>::= <statement number [1.4.a] OPTION> GO TO
<actual statement number pack [2.5.a]><comma OPTION>
<arithmetic expression [2.7.1.a]>

e. <comma>::= ,

Semantiek:

c. De integer name behoort sedert een voorafgaande assign statement [4.3.a] een statement number te bevatten, dat in het actual statement number pack voorkomt. Het programma gaat dan verder met de statement aangewezen door dit statementnumber. Bezit de integer name niet een zodanig statement number, dan worden de laatste 18 bits van de waarde die de integer name bezit als een absoluut geheugenadres geïnterpreteerd, en gaat het programma daar verder (!!)

d. Is de expressie een integer variable, en is de waarde kleiner dan 1 of groter dan het aantal statement numbers in het actual statement number pack, dan wordt de executie met een foutmelding beëindigd, anders wordt de expressie geëvalueerd, en vervolgens getrunceerd tot een integer tussen 1 en het aantal statement numbers. Als vervolg van het programma wordt gekozen de statement aangewezen door het statement number aangewezen door de waarde van de expressie.

Voorbeelden:

b. 10 GO TO 11

c. GO TO KL,(10,11,12)

d. GO TO (10,11,12)N

4.3. Assign statement.

a. <assign statement>::= <statement number [1.4.a] OPTION>ASSIGN
<statement number [1.4.a]> TO <integer name [1.1.a]>

Semantiek:

a. De integer name mag niet meer in een expressie worden gebruikt, tenzij hij eerst in een left part van een assignment statement is verschenen.

Voorbeelden:

a. ASSIGN 11 TO KL

4.4. If statement.

a. <if statement>::= <three branch if>|<two branch if>|<simple if>

b. <three branch if>::= <statement number [1.4.a] OPTION> IF
(<arithmetic expression [2.7.1.a]>) statement number [1.4.a]>,
<statement number [1.4.a]>, <statement number [1.4.a]>

c. <two branch if>::= <statement number [1.4.a] OPTION >
IF (<expression [2.7.a]>)<statement number [1.4.a]>,
<statement number [1.4.a]>

d. <simple if>::= <statement number [1.4.a] OPTION>
IF (<logical expression [2.7.3.a]>)<statement [4.a]>

Semantiek:

b. De arithmetic expression mag niet van type octal of hollerith zijn. De expressie wordt geëvalueerd, vervolgens wordt als volgt een statement number gekozen: is de waarde kleiner dan nul, dan het eerste, is de waarde nul, dan het tweede, en anders het derde.

Het programma vervolgt bij de door dit statement number aangewezen statement. Bij een complex getal wordt de waarde van het reële deel genomen.

c. De expressie wordt geëvalueerd, en een statement number wordt gekozen volgens de volgende tabel:

waarde van de expressie	arithmetic expression	logical expression	masking expression octal	hollerith
eerste stat. number	≠0	.TRUE.	Na uitwerking expressie: indien ≠0 (octal→integer getal)	altijd
tweede stat. number	=0	.FALSE.	rest	nooit

d. Als statement zijn uitgesloten: 1. een do statement
2. een end statement
3. een simple if

Voorbeelden:

b. IF (EXPR) 10,11,12
c. IF (EXPR) 10,11
d. IF (BOOL) GOTO 11

4.5. Do statement.

a. <do statement>::= <statement number [1.4.a] OPTION>
DO <statement number [1.4.a]><integer name [1.1.a]>=
<start [2.1.d]> <finish [2.1.e]><step [2.1.f] OPTION>

Semantiek:

a. Het statement number na DO mag niet staan voor een:
1° goto statement
2° three branch if of two branch if
3° return statement, stop statement of pause statement

4° do statement

5° simple if gevolgd door een van de vorige 4.

De integer name mag binnen de do loop niet in een left part verschijnen. <start>, <finish> en step moeten groter dan nul zijn; zijn het integer names, dan mag de waarde binnen de do loop niet veranderen. Wordt <step> weggelaten, dan wordt hiervoor 1 genomen.

De statements binnen een do loop worden minstens eenmaal uitgevoerd (De test vindt dus aan het eind plaats).

Nesting: do loops mogen genest voorkomen, ze mogen echter niet door elkaar heenlopen. Na een sprong via een goto statement, three branch if of two branch if uit de do loop, mag, mits de integer namen voor <start>, finish en step niet van waarde veranderd zijn, weer in de do loop teruggesprongen worden. Na afloop van de do loop is de waarde van de integer name niet gedefiniëerd.

Voorbeelden:

a. 10 DO 11 I= 1, 10, 3

4.6. Continue statement.

a. <continue statement>::= <statement number [1.4.a] OPTION> CONTINUE

Semantiek:

a. Een continue statement is een dummy statement.

Voorbeelden:

a. 11 CONTINUE

4.7. Call.

a. <call>::= <statement number [1.4.a] OPTION> CALL
<subroutine name [1.1.c]><actual parameter pack [2.3.a] OPTION>
<actual returns OPTION>

b. <actual returns>::= ,RETURNS <actual statement number pack [2.5.a]>

Voorbeelden:

a. CALL SUB(A, 10, B), RETURNS(10, 11, 100)

4.8. Pause en stop statement.

a. <pause statement>::= <statement number [1.4.a] OPTION> PAUSE
<octal number [1.2.o] OPTION>
b. <stop statement>::= <statement number [1.4.a] OPTION> STOP
<octal number [1.2.o] OPTION>

Semantiek:

a. en b. Het octal number mag niet groter zijn dan 77777.
a. Het programma stopt, het octale getal wordt afgedrukt op het monitorvel. De operateur kan het programma doorstarten of beëindigen.
b. De executie wordt beëindigd; het octale getal wordt op het monitorvel afgedrukt.

Voorbeelden:

a. PAUSE 7657
b. STOP

4.9. Entry statement.

a. <entry statement>::= <statement number [1.4.a] OPTION> ENTRY
<procedure name>
b. <procedure name>::= <subroutine name [1.1.c]>|<function name [1.1.d]>

Semantiek:

a. Het aantal parameters wordt geacht gelijk te zijn aan het aantal in de header van het subprogramma. (evenzo voor de returns).

b. De mode van de function name is gelijk aan de mode van de name in de header.

Voorbeelden:

a. 10 ENTRY SUB 1

4.10. Return statement.

a. <return statement>::= <statement number [1.4.a] OPTION> RETURN <integer name [1.1.a] OPTION>

Semantiek:

a. De integer name mag alleen gebruikt worden in een subroutine program, en dan nog alleen als hij in het formal statement number pack voorkomt.

Voorbeelden:

a. RETURN
RETURN B

5. Format indication.

a. <format indication>::= <statement number [1.4.a]> FORMAT <format>
b. <format>::= (<format list>)
c. <format list>::= <slashes OPTION><field descriptor list OPTION> <slashes OPTION>
d. <field descriptor list>::= <field descriptor>|<field descriptor> <field separator><field descriptor list>
e. <field descriptor>::= <simple field descriptor>| <replicator OPTION><format>
f. <simple field descriptor>::= <floating point descriptor>| <alphanummeric descriptor>|<hollerith descriptor>| <spacing descriptor>

g. <field separator>::= ,|<slashes>

h. <slashes>::= /<slashes OPTION>

i. <replicator>::= <integer constant [1.2.k]>

j. <field length>::= <integer constant [1.2.k]>

k. <position>::= <integer constant [1.2.k]>

Semantiek:

c. Een format list mag niet leeg zijn. Nesting mag slechts twee diep geschieden. Een slash (/) geeft een nieuw record, d.i.:
   1° regeldrukker: nieuwe regel
   2° kaart ponser: nieuwe kaart
   3° elders: end of record marker bij uitlezen, record overgang bij inlezen.

i, j en k. replicator, field length en position moeten groter dan nul zijn.

Voorbeelden:

a. 100 FORMAT(//5F10.2,I5)

5.1. Floating point descriptor.

a. <floating point descriptor>::= <scaling factor OPTION>
   <replicator [5.i] OPTION><floating type indicator>
   <field length [5.j]><fractional part length OPTION>

b. <scaling factor>::= <minus OPTION><integer constant [1.2.k]> P

c. <minus>::= -

d. <floating type indicator>::= F|E|D|G

e. <fractional part length>::= .<integer constant [1.2.k]>

Semantiek:

b. Bij invoer: een gelezen getal zonder expliciet vermelde exponent wordt voor verdere verwerking vermenigvuldigd met 10 ↑ (-factor).

Bij uitvoer: Er verschijnen (factor) decimalen voor de punt (is factor negatief, dan (- factor) nullen achter de punt).
Bij E en D conversie, en indien bij G conversie overgegaan wordt op E conversie, ook bij G conversie, wordt de exponent met factor verlaagd. De factor is de waarde van de laatst verwerkte scaling factor, deze waarde is bij ingang van een format nul.

d. F conversie:
invoer: als bij E conversie, een exponent mag echter niet in de invoer staan.
uitvoer: het getal wordt geconverteerd tot een rij decimale cijfers met, indien fractional part length is gedefinieerd, een punt en (fractional part length) cijfers achter de punt. Nullen aan het begin, tot de punt, worden weggelaten, en indien het getal negatief is, dan wordt voor het eerste cijfer een min geplaatst.
Is het aantal zo verkregen symbolen kleiner dan (field length), dan wordt de symbolenrij naar voren opgevuld met spaties, is de symbolenrij te lang, dan worden de laatste (field length - 1) symbolen bewaard, van voren aangevuld met een *.
De zo verkregen symbolenrij wordt uitgevoerd. Is het getal ongedefinieerd, dan wordt een I uitgevoerd, is het getal absoluut groter dan $2^{48}-1$, dan wordt een R uitgevoerd.
E conversie:
invoer: Er worden (field length) symbolen ingevoerd, deze symbolenrij moet van een der volgende vormen zijn:
1° <sign [1.2.1] OPTION><integer constant [1.2.k]>
2° <sign [1.2.1] OPTION><real of double precision constant [1.2.h en f]>
(In de exponent mag de D of E weggelaten worden, mits er een teken in staat.)
3° <sign [1.2.1] OPTION><integer constant [1.2.k] OPTION><single of double precision exponent part [1.2.i. en g]>
(Indien de integer constant wordt weggelaten, is de ingelezen waarde nul.)
Een spatie wordt als nul geïnterpreteerd. De zo verkregen waarde wordt, indien er in de descriptor een fractional part length

voorkomt èn er in de ingevoerde symbolenrij geen punt voorkwam, vermenigvuldigd met 10 ↑ (- fractional part length).

Uitvoer: Het getal wordt geconverteerd tot een rij decimale cijfers, ter lengte(fractional part length), voorafgegaan door, indien fractional part length is gedefinieerd, een punt, en, indien het getal negatief was, een min-teken.

De rij symbolen wordt gevolgd door:

1° indien de exponent groter dan 99 was een teken, gevolgd door de exponent in drie cijfers.

2° anders een E, een teken, gevolgd door de exponent in twee cijfers.

Is het aantal zo verkregen symbolen kleiner dan (field length), dan wordt de rij naar voren opgevuld met spaties, is de symbolenrij te lang, dan worden het mogelijke teken van het fractional part en een voldoende aantal minst significante cijfers van het fractional part weggelaten, en aan de rij wordt een * aan het begin toegevoegd. De zo verkregen symbolenrij wordt uitgevoerd. Is het getal ongedefinieerd, dan wordt een I uitgevoerd; is het getal oneindig, dan wordt een R uitgevoerd.

D conversie: als E conversie, bij uitvoer mogelijke vervanging van E door D.

G conversie:

Invoer: als E conversie.

Uitvoer: uitvoer volgens Gw.d gaat als volgt:

1° $0.1 \leq |\text{waarde getal}| < 10\uparrow d$:
als F(w-4).entier(d $-{}^{10}\log|\text{waarde getal}|+1$), 4x, zie [5.4.a]

2° anders als Ew.d

Voorbeelden:

a. -5P5E20.4

5.2. Alphanumeric descriptor.

a. <alphanumeric descriptor>::= <replicator [5.i] OPTION>
<alphanumeric indicator>

b. <alphanumeric indicator>::= I|O|A|R|L

Semantiek:

b. I conversie:

invoer: (field length) symbolen worden ingelezen, deze symbolen moeten vormen: <sign [1.2.1] OPTION><integer constant>, waarbij een spatie als nul wordt geïnterpreteerd.

uitvoer: Het getal wordt geconverteerd tot een rij decimale cijfers, waarbij nullen aan het begin worden weggelaten, behalve de laatste als het getal 0 is, deze rij wordt, indien het getal negatief is, voorafgegaan door een min. Is de lengte van de symbolenrij kleiner dan (field length), dan wordt de rij naar voren opgevuld met spaties, is de rij te lang, dan worden de laatste (field length - 1) symbolen bewaard, voorafgegaan door een *. De zo verkregen symbolenrij wordt uitgevoerd. Is de absolute waarde van het getal groter dan $2^{48}-1$, dan wordt een X afgedrukt.

O conversie: Als I conversie, met de volgende verschillen:

1° in plaats van decimale cijfers mogen overal slechts octale cijfers voorkomen.

2° is de symbolenrij te lang, dan worden de laatste (field length) symbolen afgedrukt.

3° min-teken wordt nooit afgedrukt.

4° er wordt nooit een X afgedrukt.

A conversie:

invoer: (field length) symbolen worden ingevoerd, indien (field length) kleiner is dan 10, dan gevolgd door voldoende spaties, is (field length) groter dan 10, dan worden de laatste 10 symbolen bewaard. De zo verkregen symbolenrij wordt beschouwd als hollerith constante.

uitvoer: Het uit te voeren woord wordt beschouwd als hollerith constante, de symbolen worden stuk voor stuk uitgevoerd, indien (field length) groter is dan 10, voorafgegaan door spaties. Is (field length) kleiner dan 10, dan worden slechts de eerste (field length) symbolen uitgevoerd.

R conversie: Als A conversie, echter met de volgende uitzondering:
als (field length) kleiner is dan 10, dan:
bij invoer: de symbolen worden voorafgegaan door binaire nullen.
bij uitvoer: de laatste (field length) symbolen worden uitgevoerd.
L conversie:
invoer: (field length) symbolen worden ingevoerd. Is het eerste symbool, dat geen spatie is, een T, dan is de waarde .TRUE., anders, ook als er alleen maar spaties zijn, .FALSE.
uitvoer: (field length - 1) spaties worden uitgevoerd, gevolgd door een T voor .TRUE. of een F voor .FALSE.

Voorbeelden:

a. I10
   5R8

5.3. Hollerith descriptor.

a. <hollerith descriptor> ::= <hollerith constant [1.2.b]> |
   <string [1.2.d]> | ≠ <string [1.2.d]>≠

Semantiek:

a. ≠ is géén FORTRAN character, op verschillende printers zal het symbool verschillend afgedrukt worden.
   invoer: hollerith constant: er worden evenveel symbolen ingelezen als de integer constant aangeeft.
   andere versie: er wordt een symbolenrij ingelezen, ter lengte van de string. De verkregen symbolenrij wordt over de string geschreven.
   uitvoer: De string wordt symbool voor symbool uitgevoerd. Voor een hollerith constant is het maximum aantal symbolen 136.

Na een hollerith constant is de komma als field separator optioneel.

Voorbeelden:

a. 6HALLENb
STRING

5.4. Spacing descriptor.

a. <spacing descriptor>::= <field length [5.j]> X|T <position [5.k]>

Semantiek:

a. X conversie: bij invoer worden (field length) symbolen overgeslagen, bij uitvoer worden (field length) spaties uitgevoerd.
1° space X ≡ 1X; 2° komma als field separator is optioneel.
T conversie: De positie pointer krijgt de waarde (position) voor het record. Is (position) nul dan wordt T1 aangenomen. (position) is maximaal 136.
Bij kaartinvoer worden, indien (position) groter dan 80 is, door volgende field descriptors alleen maar spaties gevonden, tot:
1° door een slash overgang naar nieuwe kaart wordt gedwongen.
2° bij ingang van een format indication of heringang van een format een nieuwe kaart wordt begonnen.
3° door een T de positie weer kleiner dan 80 is geworden.
Bij regeldrukker uitvoer wordt de positie (position) - 1.

Voorbeelden:

a. 6X
T79

6. Input/output statements.

a. <io statement>::= <formatted io>|<namelist io>|<unformatted io>|
<mass storage handling>|<conversion statement>

b. <io with unit>::= READ|WRITE

c. <io without unit>::= READ|PRINT|PUNCH

d. <format specification>::= <statement number [1.4.a]>|
<array name [1.1.b]>

e. <io list>::= ,<io element list>

f. <io element list>::= <io element><io list OPTION>

g. <io element>::= <variable [1.3.b]>|<implied do loop [2.1.a]>|
<array name [1.1.b]>|(<io element list>)

h. <unit>::= <integer constant [1.2.k]>|<integer variable [1.3.a]>

Semantiek:

b. io via magnetische tape.

c. io via resp. kaartlezer, regeldrukker en kaartponser.

d. Het statementnumber moet staan voor een format indication [5.a]. Indien er een array name staat, moet in dit array een format [5.b] ingelezen zijn volgens A-conversie [5.2.b]. Voor output via de regeldrukker (en ook naar tapes voor latere verwerking over regeldrukker)geldt, dat het eerste symbool van elk record gebruikt wordt voor carriage control:

symbool	actie voor format	actie na afloop format
A	nieuwe regel	nieuwe pagina
B	nieuwe regel	ga naar laatste regel van pagina
1	nieuwe pagina	--
2	ga naar laatste regel van pagina	--
+	--	--
0	2 nieuwe regels	--
-	3 nieuwe regels	--
spatie	1 nieuwe regel	--

Wanneer na een A of B een aantal slashes worden gegeven, dan wordt hiervan één niet geëffectueerd. Andere symbolen:

Q paginering opheffen

R weer pagineren

andere symbolen onduidelijk of onbekend, de meest verrassende effecten zijn te verwachten. Aangeraden wordt deze niet te gebruiken.

f. Een io element list mag eenmaal genest voorkomen. Het aantal elementen in de io element list wordt geacht gelijk te zijn aan het aantal floating point descriptors plus het aantal alphanumeric descriptors. Is dit niet het geval, dan gebeurt het volgende:
   a) Is het format genest zonder replicator, dan wordt met een nieuw record begonnen op deze plaats in het format (dus na de meest rechtse openingshaak, waar geen replicator voor staat), anders:
   b) Het format wordt op een nieuw record weer van voren af aan begonnen.

6.1. Formatted io.

a. <formatted io>::= <statement number [1.4.a] OPTION><io specification>

b. <io specification>::= <io with unit [6.b]><list with unit>|
   <io without unit [6.c]><list without unit>

c. <list with unit>::= (<unit [6.h]>,<format specification [6.d]>)
   <io element list [6.f] OPTION>

d. <list without unit>::= <format specification [6.d]>
   <io list [6.e] OPTION>

Semantiek:

b. Indien de list niet leeg is, moet in het bijbehorende format minstens één floating point descriptor of alphanumeric descriptor staan.

Voorbeelden:

a. 10 WRITE(6,100) A, (B(I), I= 1,5)
   READ 100, C

6.2. Namelist io.

a. <namelist io>::= <statement number [1.4.a] OPTION>
<namelist io specification>

b. <namelist io specification>::= <io with unit [6.b]>
<namelist with unit>|
<io without unit [6.c]><namelist name [3.7.c]>

c. <namelist with unit>::= (<unit [6.h]>, <namelist name [3.7.c]>)

Semantiek:

a. Bij de invoer kunnen aan variabelen waarden toegekend worden, dit gebeurt als volgt:
1°<variable>= <sign OPTION><constant>
2°<variable with subscript>= <sign OPTION><constant>,...,
<sign OPTION><constant . Vanaf de aangegeven index wordt het array gevuld.
3°<array name>= <sign OPTION><constant>, ...,<sign OPTION>
<constant>. Als 2° vanaf index 1.
In de gevallen 2° en 3° mag een repetitiefaktor k* gebruikt worden. Het geheel moet voorafgegaan worden door een $, gevolgd door de namelist name, en afgesloten door een $. Constanten van type hollerith of octal mogen niet voorkomen. Logical en complexe constanten moeten aan logical en complexe variabelen toegekend worden, bij de rest is mixed mode mogelijk. De constanten met eventuele repetitie factor mogen geen blanks bevatten. Meer dan een record per lees opdracht is mogelijk, mits ieder record eindigt met een, (behalve na de afsluitende $). Van ieder record wordt het eerste symbool overgeslagen.
Uitvoer: Het volgende wordt uitgevoerd:
1° een record met in positie 2 een $ gevolgd door de namelist name.
2° een of meer records, steeds beginnend in positie 2, die op overeenkomstige wijze ingelezen kunnen worden.
3° een record met in positie 2 een $ gevolgd door END.

Voorbeelden:

a. READ(6, NAME)
   PUNCH NAME

6.3. Unformatted io.

a. <unformatted io>::= <statement number [1.4.a] OPTION>
   <io with unit [6.b]>(<unit [6.h]>)
   <io element list [6.f] OPTION>

Semantiek:

a. Invoer en uitvoer geschieden binair. Is de io element list leeg, dan wordt bij invoer een record overgeslagen, en bij uitvoer een leeg record geschreven.

Voorbeelden:

a. READ(5) B, C
   WRITE(6)

6.4. Mass storage handling.

a. <mass storage handling>::= <rewind statement>|<backspace statement>|
   <end file statement>|<buffer statement>
b. <rewind statement>::= <statement number [1.4.a] OPTION>REWIND
   <unit [6.h]>
c. <backspace statement>::= <statement number [1.4.a] OPTION>BACKSPACE
   <unit [6.h]>
d. <endfile statement>::= <statement number [1.4.a] OPTION>ENDFILE
   <unit [6.h]>
e. <buffer statement>::= <statement number [1.4.a] OPTION>
   <buffer indicator>(<unit [6.c]>,<parity>)
   (<first address>,<last address>)
f. <buffer indicator>::= BUFFER IN|BUFFER OUT
g. <parity>::= 0|1|<integer name [1.1.a]>

h. <first address>::= <address>

i. <last address>::= <address>

j. <address>::= <integer constant [1.2.k]>|<integer name [1.1.a]>

Semantiek:

e. Een buffer statement transporteert informatie van een unit in de mode aangegeven door parity, naar geheugenadres <first address> tot en met <last address> of omgekeerd. Het voordeel van deze statement is, dat het programma doorrekent tijdens het transport. Men moet echter wel, voordat men van de betreffende informatie in het geheugen gebruik gaat maken, d.m.v. de UNIT function controleren of het transport afgelopen is. Op de betreffende unit mag verder alleen andere <mass storage handling> toegepast worden.

g. De integer name mag alleen de waarden 0 (even pariteit) en 1 (oneven pariteit) aannemen.

Voorbeelden:

b. REWIND 6

c. BACKSPACE 5

d. ENDFILE I

e. BUFFER IN (5,I) (A,B)

6.5. Conversion statement.

a. <conversion statement>::= <statement number [1.4.a] OPTION>
<convertor> (<character length>,<format specification [6.d]>,
<starting address>)<io element list [6.f]>

b. <convertor>::= ENCODE|DECODE

c. <starting address>::= <variable [1.3.b]>

d. <character length>::= <integer constant [1.2.k]>|
<integer name [1.1.a]>

Semantiek:

a. Een conversion statement is te vergelijken met een formatted READ/ WRITE statement, alleen gaat het hier om verplaatsing binnen het geheugen van een aantal records, elk ter lengte <character length>. ENCODE verplaatst de inhoud van de io element list naar een brok geheugen, beginnend op <starting address>, DECODE doet het omgekeerde.

d. <character length> is het aantal symbolen per record. Een record bestaat uit een geheel aantal woorden, in elk woord gaan maximaal 10 symbolen.

Voorbeelden:

a. DECODE(10, 100, A) B,C

7. Program.

a. <program>::= <subprogram list OPTION><main program>
<subprogram list OPTION>

b. <subprogram list>::= <subprogram><subprogram list OPTION>

c. <subprogram>::= <subroutine subprogram>|<function subprogram>|
<block data subprogram>

d. <first indication list>::= <first type indication [3.b]>
<first indication list OPTION>|<format indication [5.a]>
<first indication list OPTION>

e. <second indication list>::= <second type indication [3.c]>
<second indication list OPTION>|<format indication [5.a]>
<second indication list OPTION>

f. <statement list>::= <statement [4.a]><statement list OPTION>|
<format indication [5.a]><statement list OPTION>

g. <end marker>::= END

7.1. Main program.

a. <main program>::= <main program header OPTION>

<first indication list [7.d] OPTION>
<second indication list [7.e] OPTION>
<statement list [7.f] OPTION><end marker [7.g]>

b. <main program header> ::= PROGRAM <program name [1.1.e]>
<program parameter pack OPTION>

c. <program parameter pack> ::= (<program parameter list>)

d. <program parameter list> ::= <program parameter> | <program parameter> ,
<program parameter list>

e. <program parameter> ::= <filename> | <filename> = <filename> |
<filename> = <bufferlength>

f. <filename> ::= INPUT | OUTPUT | PUNCH | TAPE <integer constant [1.2.k]>

g. <bufferlength> ::= <integer constant [1.2.k]> |
<octal constant [1.2.n]>

Semantiek:

a. Wordt de header weggelaten, dan vult de compiler in: PROGRAM START (INPUT, OUTPUT, PUNCH)

b. Wordt het parameter pack weggelaten, dan vult de compiler in: (INPUT, OUTPUT, PUNCH)

e. Wordt <filename> gebruikt, dan vult de compiler in:
<filename> = 1025.
<filename> = <filename>:
De invoer resp. uitvoer, die normaal gaat via de file met als naam de linker filename, gaat nu via de file met als naam de rechter filename. De rechter filename moet in de parameter list al eerder voorgekomen zijn.

f. De integer constant moet groter dan 0 en kleiner dan 100 zijn.
INPUT moet gedefinieerd zijn, als in het programma een READ statement zonder unit voorkomt.
OUTPUT moet gedefinieerd zijn, als in het programma een PRINT statement voorkomt.
PUNCH moet gedefinieerd zijn, als in het programma een PUNCH statement voorkomt.
TAPE u moet gedefinieerd zijn, als in het programma een io statement voor unit u voorkomt.

Voorbeelden:

b. PROGRAM VOORB(INPUT, OUTPUT = 4096, TAPE 1 = OUTPUT)

7.2. Subroutine subprogram.

a. <subroutine subprogram>::= <subroutine header>
<first indication list [7.d] OPTION>
<second indication list [7.e] OPTION>
<statement list [7.f] OPTION><end marker [7.g]>

b. <subroutine header>::= SUBROUTINE <subroutine name [1.1.c]>
<formal parameter pack [2.2.a] OPTION><formal returns OPTION>

c. <formal returns>::= ,RETURNS <formal statement number pack [2.5.a]>

Semantiek:

a. Een subroutine mag niet recursief zijn.

Voorbeelden:

b. SUBROUTINE DET(A, N, P), RETURNS (I)

7.3. Function subprogram.

a. <function subprogram>::= <function header>
<first indication list [7.d] OPTION>
<second indication list [7.e] OPTION>
<statement list [7.f] OPTION><end marker [7.g]>

b. <function header>::= <mode indicator [3.1.c] OPTION> FUNCTION
<function name [1.1.d]><formal parameter pack [2.2.a]>

Semantiek:

a. Een function mag niet recursief zijn.

b. De mode van een function mag niet ECS zijn. De mode van een function wordt bepaald door hetzij de mode indicator, hetzij de eerste letter van de function name, op precies dezelfde wijze als bij variables, zie [3.1.a]. Deze function name moet ook voorkomen in een mode

indication in elk programma dat deze function gebruikt, tenzij de mode door de eerste letter bepaald kan worden. De function name mag in elke statement in het function subprogram gebruikt worden als variabele, een subscript part is dan echter (uiteraard) verboden. Hetzelfde geldt voor een function name uit een entry statement in het function subprogram.

Voorbeelden:

b. DOUBLE PRECISION FUNCTION ERF(X)

7.4. Block data subprogram.

a. <block data subprogram>::= <block data header>
<first indication list[7.d]><second indication list [7.e]>
<end marker [7.g]>

b. <block data header>::= BLOCK DATA

Semantiek:

a. Een block data subprogram dient om variabelen uit een common te initialiseren met behulp van een <data indication [3.6.a]>.
Blank common mag echter niet geïnitialiseerd worden.
Toegestaan zijn alleen:
1° <common indication [3.3.a]>
2° <data indication [3.6.a]>
3° <equivalence indication [3.4.a]>
4° <mode indication [3.1.a]>
5° <dimension indication [3.2.a]>
en 2° t/m 5° alleen voorzover het betreft variabelen uit een common gedefinieerd in de common indication, en deze laatste alleen voor commons die geheel of gedeeltelijk met de data indication worden geïnitialiseerd.

Voorbeelden:

b. BLOCK DATA

8. Program pack.

a. <program pack>::= <program [7.a]>|<overlay list>|
<segmented program>

8.1. Overlay.

a. <overlay list>::= <overlay><overlay list OPTION>
b. <overlay>::= <overlay header><program [7.a]>
c. <overlay header>::= OVERLAY (<file overlay OPTION><primary level>,
<secondary level><start indication OPTION>)
d. <primary level>::= <octal number [1.2.o]>
e. <secondary level>::= <octal number [1.2.o]>
f. <start indication>::= , C <octal number [1.2.o]>
g. <file overlay>::= <file [0.3.o]>,

Semantiek:

a. en b. zie [A.1.i]
b. Files die in meer dan een overlay gebruikt worden, moeten voorkomen op de program kaart van het overlay, dat alle overlay's kan aanroepen.
c. De eerste overlay moet een filename bevatten, de volgende overlay's worden dan ook op deze file gezet.
d. en e. Primary level en secondary level (p,s) moeten kleiner zijn dan 100. Er moet een overlay (0,0) zijn, en is er een overlay (p,s) dan moet er ook een overlay (p,0) zijn. Overlay (0,s) is verboden.
f. De start indication geeft aan hoeveel woorden na blank common het overlay geladen moet worden.

Voorbeelden:

c. OVERLAY(17, 0)

8.2. Segment.

a. <segmented program>::= <segement indication list><program>

b. <segment indication list>::= <zero segment indication>
<non zero segment indication list>

c. <non zero segment indication list>::= <non zero segment indication>
<non zero segment indication list OPTION>|<section indication>
<non zero segment indication list OPTION>

d. <segment name>::= <name>

e. <section name>::= <name>

Semantiek:

a. zie [A.1.i]

8.2.1. Zero sement indication.

a. <zero segment indication>::= SEGZERO(<segment name [8.2.d]>
<zero segment list>)

b. <zero segment list>::=, <program name [1.1.e]>
<zero segment list OPTION>|, <subroutine name [1.1.c]>
<zero segment list OPTION>|, <function name [1.1.d]>
<zero segment list OPTION>|, <section name [8.2.e]>
<zero segment list OPTION>

Semantiek:

a. Het zero segment zit altijd in het kerngeheugen, het level is 0.

b. Program name moet voorkomen, de function name moet de naam zijn van een function subprogram.

Voorbeelden:

a. SEGZERO (MAIN, VOORB, ERF, SEC1)

8.2.2. Non zero segment indication.

a. <non zero segment indication>::= SEGMENT (<segment name [8.2.d]>
<non zero segment list>)

b. <non zero segment list>::= , <subroutine name [1.1.c]>
<non zero segment list OPTION>|, <function name [1.1.d]>
<non zero segment list OPTION>|, <section name [8.2.e]>
<non zero segment list OPTION>

Semantiek:

a. Het level is altijd groter dan 0.

b. De function name moet de naam zijn van een function subprogram.

Voorbeelden:

a. SEGMENT(FIRST, DET, SOL, DETSOL)

8.2.3. Section indication.

a. <section indication>::= SECTION(<section name [8.2.e]><section list>)

b. <section list>::= ,<subroutine name [1.1.c]><section list OPTION>
,<function name [1.1.d]><section list OPTION>

Semantiek:

b. De function name moet de naam zijn van een function subprogram.

Voorbeelden:

a. SECTION(SEC1, ERFC, GAMMA)

A. Standaard subroutine en functions.

A.1. Standaard subroutines.

a. sense switches en sense lights.

SLITE (i) : $0 \leq i \leq 6$, i=0: zet alle sense lights af
i≠0: zet sense light i aan

SLITET (i,j): $0 < i \leq 6$, als sense light i aan: j=1, zet sense light i af.
anders: j=2

SSWITCH(i,j): $0 < i \leq 6$, als sense switch i aan: j=1, anders: j=2

b. exit:

EXIT: beëindig het programma.

c. display:

REMARK (H): H is hollerith constante, maximaal 80 karakters, wordt op het monitorvel afgedrukt.

DISPLA(H,k): H is hollerith constante, maximaal 10 karakters, wordt op het monitorvel afgedrukt.
k is een real of integer, en zijn waarde wordt ook afgedrukt op het monitorvel.

d. random:

RANSET(n) initialiseer de randomrij op het eerste oneven getal ≥ n (n is real).

RANGET(n) n krijgt de waarde van het volgende randomgetal.

e. dump:

DUMP $(a_1, b_1, f_1, \ldots, a_n, b_n, f_n)$:

PDUMP $(a_1, b_1, f_1, \ldots, a_n, b_n, f_n)$:

a_i is het eerste, b_i is het laatste woord van een stuk geheugen dat gedumpt moet worden.

f_i = 0,3: octaal dumpen
= 1: real
= 2: integer
= 4: octaal, a_i en b_i zijn statement numbers, door een ASSIGN statement gedefiniëerd.

Na DUMP wordt het programma beëindigd, na PDUMP gaat het programma verder.

f. error check:

ERRSET (A,B): Het maximale aantal fouten in de invoer is B, de fouten worden geteld in A. Als A>B: exit.

g. ecs input/output.

READEC (a,b,n) lezen c.q. schrijven van Extended Core Storage naar
WRITEC (a,b,n) geheugen.

a is een variabele die het eerste geheugenadres aangeeft.

b is een ECS variabele, geeft het eerste adres in het ECS aan.

n is een integer expressie, geeft het aantal te transporteren woorden aan.

h. mass stoarge.

OPENMS (u, ix, l, p): u = unit,
ix = eerste adres van de geheugen index (array name),
l = lengte index,
p = 0: records hebben nummers, p = 1: records hebben namen.

READMS (u, fwa, n, i): u = unit,
WRITMS (u, fwa, u, i)
fwa = variabele of array naam, geeft het eerste geheugenadres aan,
n = aantal te transporteren woorden,
i = record-nummer of-naam.

STINDX (u, ix, l): parameters als bij OPENMS.

functie: OPENMS initialiseert, de master index wordt in array ix gezet.
READMS en WRITMS lezen/schrijven records van/op file u.
STINDX: inplaats van de oude index wordt nu de nieuwe index ix gebruikt. Indien deze index nieuw aangemaakt moet worden, moet hij eerst op 0 gezet worden. Voor beëindiging van het programma moet m.b.v. STINDX weer de master index teruggezet worden. De master index wordt na afloop van het programma automatisch weggeschreven.

i. overlays en segments.

OVERLAY (fn, I, J, p, l): fn = integer variabele die de filenaam bevat,
I = eerste niveau, J = tweede niveau.
Als p = 6HRECALL, wordt het overlay, als het al in het geheugen aanwezig is, niet opgehaald.
Als l weggelaten wordt of nul is, wordt het overlay uit file fn gehaald, anders uit de library file.

OVERLAY laadt het betreffende overlay (=een volledig programma) en voert het uit.
Overlays mogen allen aangeroepen worden vanuit het hoofd overlay (=overlay(0,0)) en verder mag overlay (I,J) aangeroepen worden vanuit overlay (I,0).

SEGMENT (fn, e, a, lib, m): fn = integer variabele die de filenaam bevat,
e = level, octaal getal, ≤ 77 octaal.
a = variabele of array name die de lijst van namen van subprogramma's, segmenten of sections bevat, afgesloten door een 0. Als de eerste 0 is, worden alle subprogramma's geladen.
lib = 0, spatie of weggelaten (dit laatste kan alleen als m ook weggelaten wordt): zo mogelijk worden alle aanroepen van subprogramma's verbonden met subprogramma's uit de bibliotheek.
m = 0, spatie of weggelaten: er wordt geen map van het segment gegeven.

SEGMENT laadt een aantal subprogramma's, die d.m.v. het levelnummer (e) een level krijgen. Het hoofdprogramma heeft level 00.
Wordt een segment aangeroepen met kleiner levelnummer dan het laatst

geladen segment, of met levelnummer gelijk hieraan , dan worden alle segmenten met levelnummer groter of gelijk aan het levelnummer van het aangeroepen segment verwijderd. Een segment mag dus nooit een segment aanroepen met levelnummer kleiner of gelijk aan het eigen levelnummer. COMMON met een naam mag niet in verschillende segmenten voorkomen. De lengte van blank common is de lengte, zoals die vermeld wordt in het eerste segment waarin blank common voorkomt.

j. datum en tijd.

De functies uit [A.2.g] mogen ook als subroutines worden gebruikt.

k. debugging subroutine.

STRACE geeft een map van de levels van de plaats waar hij aangeroepen wordt tot en met het hoofdprogramma.

l. labelen van een file.

LABEL (u, fwa): u = unit
fwa = eerste adres van 4 woorden, die de label informatie bevatten.

Bij input wordt de label vergeleken met die van de betreffende file, bij output wordt de label geschreven.

m. intercom - interface.

CONNEC (lfn) lfn = unitnummer of
DISCON (lfn) hollerithconstante (left justified), of een integer variabele, een van beide vorige bevattend.

File lfn wordt met de terminal van de gebruiker verbonden (CONNEC), of wordt weer losgekoppeld (DISCON).

A.2. Standaard functions.

a. absolute waarde, modulus e.d.

ABS (x)	x = real parameter; waarde real
IABS (i)	i = integer parameter; waarde integer.
DABS (d)	d = double precision parameter; waarde double precision.
CABS (c)	c = complexe parameter; waarde real: $\sqrt{(Re(c)^2 + Im(c)^2)}$
MOD (i1, i2)	i1, i2 = integer parameters; waarde integer: (i1 mod(i2) * teken (i1).
AMOD(r1, r2)	r1, r2 = real parameters; waarde real als boven.
ISIGN(i1,i2)	i1, i2 = integer parameters; waarde integer: teken (i2) * |i1|.
SIGN(r1, r2)	r1, r2 = real parameters; waarde real als boven. teken (+0) = +1; teken (-0) = -1.

b. conversie functies.

INT (r1)	r1 = real parameter; waarde integer: entier(|r1|) * teken (r1).
AIN (r1)	r1 = real parameter; waarde real als boven.
IDINT (d1)	d1 = double precision parameter; waarde integer als boven.
IFIX (r)	r = real parameter, waarde integer: afgeronde waarde van r .
FLOAT (i)	i = integer parameter; waarde double presision.
SNGLE (d)	d = double precision parameter; waarde real: meest significante deel.
CMPLX(r1,r2)	r1,r2 = real parameters; waarde complex: r1 + i * r2.
REAL (c)	c = complexe parameter; waarde real: reële deel.
AIMAG (c)	c = complexe parameter; waarde real: imaginaire deel.
CONJG (c)	c = complexe parameter; waarde complex: gecon-jugeerde.

c. minima en maxima e.d.

MAXO (i1,..., in) i1,..., in integer; waarde integer; n ≥ 2.
MAX1 (r1,..., rn) r1,..., rn real; waarde integer.
AMAXO (i1,..., in) i1,..., in integer; waarde real.
AMAX1 (r1,..., rn) r1,..., rn real; waarde real.
DMAX1 (d1,..., dn) d1,..., dn double precision; waarde double precision.
MINO (i1,..., in)
MIN1 (r1,..., rn)
AMINO (i1,..., in)
AMIN1 (r1,..., rn)
DMIN1 (d1,..., dn)
IDIM (i1, i2) i1, i2 integer parameters; waarde integer: i1 - min(i1,i2).
DIM (r1, r2) r1, r2 real parameters; waarde real.

d. bitmanipulatie.

SHIFT (x,i)	x = eenwoords parameter: integer, real, octal of hollerith.
	i = integer parameter; waarde octal.
	als i ≥ + 0: x links rondschuiven.
	als i ≤ - 0: x rechts schoonschuiven. (helen door 2^i)
AND (x1,x2)	x1, x2 = eenwoords parameter; waarde octaal.
	AND: logisch .AND. bit voor bit.
OR (x1, x2)	OR: logisch .OR. bit voor bit.
COMPL (x1)	COMPL: logisch .NOT. bit voor bit.
MASK (i)	i = integer parameter; waarde octaal.
	(er gebeurt in feite niets).

e. functies:

steeds:	r = real parameter; d = double precision parameter;
	c = complex parameter.
SQRT (r)	wortel trekking: waarde resp. real, double
DSQRT (d)	precision en complex.
CSQRT (c)	

EXP (r)	exponentiële functie: waarde resp. real, double precision
DEXP (d)	en complex.
CEXP (c)	
ALOG (r)	natuurlijke logaritme: waarde resp. real, double precision
DLOG (d)	en complex.
CLOG (c)	
ALOG10 (r)	10-logaritme: waarde resp. real en double precision.
DLOG10 (d)	
SIN (r)	sinus: waarde resp. real, double precision en complex.
DSIN (d)	
CSIN (c)	
COS (r)	cosinus: waarde resp. real, double precision en complex.
DCOS (d)	
CCOS (c)	
TAN (r)	tangens: waarde real.
TANH (r)	tangens hyperbolicus: waarde real.
ASIN (r)	arcsinus: waarde real.
ACOS (r)	arccosinus: waarde real.
ATAN (r)	arctangens: waarde resp. real en double precision.
DATAN (d)	
ATAN2 (r1,r2)	arctangens van de eerste parameter gedeeld door de
DATAN2(d1,d2)	tweede parameter: waarde resp. real en double precision.

f. <u>random:</u>

RANF (x)	x willekeurig, verandert niet; waarde: real, het eerstvolgende random getal. (zie ook RANGET [A.1.d]).

g. <u>datum en tijd:</u>

DATE (d)	d krijgt als waarde de datum, d integer of real; waarde van functie is hollerith.
TIME (t)	t krijgt als waarde de tijd, t integer of real; waarde van functie is hollerith.
SECOND (r)	r real parameter; waarde van functie is de waarde die r krijgt, real: de tijd in seconden sedert aanvang programma.

h. diversen.

LEGVAR (r) — r = real parameter; waarde:
-1 als r oneindig;
+1 als r ≥ 2 ↑ 48;
anders 0.

LOCF (v) — v = variabele; waarde: adres (integer),

UNIT (u) — u = unit: integer variable of constante:

EOF (u) — UNIT: real: -1: unit gereed,

IOCHEC (u) — 0: end of file bij laatste operatie,
+1: pariteits fout.

EOF: real: 0: end of file bij laatste operatie,
-1: geen end of file.

IOCHEC: integer: 0: geen pariteits fout,
+1: pariteits fout.

UNIT voor units met buffer (BUFFER IN, BUFFER OUT).
EOF, IOCHEC voor units zonder buffer.

LENGTH (u) — u = unit: integer variabele of constante;
waarde: integer; aantal woorden, de laatste keer gelezen.

B. Voorbeelden.

B.1. Priemgetallen.

```
C       INITIALISATIE PAGE, I, J
        BLOCK DATA
        COMMON/LABEL/PAGE, I, J
        INTEGER PAGE (10,50)
        DATA I,J/2*1/,PAGE/500*0/
        END
C       BEREKENING PRIEMGETALLEN
        PROGRAM PRIMES (OUTPUT)
        COMMON/LABEL/PAGE, I, J
        LOGICAL PR(4999)
        INTEGER PAGE (10,50), A, B
        DATA PR/4999*.TRUE.
        CALL STORE (2)
        DO 200 A=3, 100, 2
          IF (.NOT.PR((A-1))/2)) GO TO 200
          CALL STORE (A)
          K = (A*A-1)/2
          DO 300 B=K, 4999, A
  300       PR(B) = .FALSE.
  200     CONTINUE
        DO 201 B=50, 4999
         IF (.NOT.PR(B)) GO TO 201
         K=B*2+1
         CALL STORE (K)
  201    CONTINUE
        IF (I.NE.1) GO TO 100
        IF (J.EQ.1) STOP
  100 CALL OUT (K)
        STOP
        END
```

```
C       OPBERGEN EN AFDRUKKEN PRIEMGETALLEN
C       GETALLEN WORDEN OPGEBORGEN IN ARRAY PAGE,
C       IS PAGE GEHEEL GEVULD, DAN WORDT PAGE
C       UITGEVOERD OP EEN NIEUWE  PAGINA.
C       OP EEN PAGINA KOMEN 10 KOLOMMEN VAN 50
C       GETALLEN, NA IEDERE REGEL VOLGT EEN EXTRA
C       BLANCO REGEL.
C       BIJ AANROEP VAN DE NAAM OUT, WORDT DE
C       LAATSTE, NIET GEHELE PAGINA AFGEDRUKT.
C       OP NOG NIET GEVULDE PLAATSEN STAAT EEN 0.
        SUBROUTINE STORE (X)
        COMMON/LABEL/PAGE, I, J
        INTEGER PAGE (10,50), X
        PAGE (J,I) = X
        I= I+1
        IF (I.LT.51) RETURN
        J= J+1
        I= 1
        IF (J.LT.11) RETURN
        ENTRY OUT
        PRINT 90000, PAGE
90000   FORMAT (1H1, 10( 5(10I8/)/))
        DO 300 I= 1, 50
          DO 300 J= 1, 10
  300       PAGE (J, I)= 0
        I= 1
        J= 1
        RETURN
        END
```

B.2. binaire boom.

```
      OVERLAY (HANSDIK, 0,0)
      PROGRAM OUTBOOM (INPUT, TAPE1, OUTPUT)
      COMMON BOOM (1000), BOOMP, N, VERW(20), VOORW(20), P,Q, WEL
      INTEGER BOOM, BOOMP, N, VERW, VOORW, P, Q
      LOGICAL WEL
      CALL OVERLAY (7HHANSDIK, 1,0)
      CALL OVERLAY (7HHANSDIK, 2,0)
      WRITE (1) BOOM
      STOP
      END

      OVERLAY (1,0)
      PROGRAM BOUBOOM
      COMMON BOOM (1000), BOOMP, N, VERW (20), VOORW (20), P, Q, WEL
      INTEGER BOOM, BOOMP, N, VERW, VOORW, P, Q
      LOGICAL WEL
C
C     DIT PROGRAMMA LEEST EEN GEORDENDE VERZAMELING
C     WOORDEN IN EN PLAATST DEZE IN EEN BINAIRE BOOM,
C     DIT GAAT OP DE VOLGENDE MANIER
C     1) EEN VERWIJZING NAAR DE LINKERTAK (-1= AARDE)
C     2) EEN VERWIJZING NAAR DE RECHTERTAK (-1= AARDE)
C     3) HET BETREFFENDE WOORD, AFGESLOTEN DOOR EEN 0
C
      N= Q= VOORW(1)= BOOM(1)= 0
      VOORW(2)= BOOMP= 1
      P= 2
      WEL= .TRUE.
   10 CALL ZET, RETURNS (12)
      GO TO 10
   12 IF (P .GT. 2) BOOM (VOORW(4))= VERW(Q-1)
      IF (P .LT. 6) GO TO 14
      DO 13 N= 6, P, 2
   13 BOOM (VOORW(N))= -1
```

```
   14 BOOM(1)= VERW(1)
C       LOSSE EINDEN AARDEN, BOOM(1) VERWIJST NAAR DE TOP
        END

        SUBROUTINE ZET, RETURNS(L)
        COMMON BOOM (1000), BOOMP, N, VERW(20), VOORW(20), P, Q, WEL
        INTEGER BOOM, BOOMP, N, VERW, VOORW, P, Q, HULP
        LOGICAL WEL, LEES
        HULP = BOOMP + 3
        N = N + 1
        IF (N/2 * 2.NE.N) GO TO 100
        CALL STORE (VERW(Q-1))
        CALL STORE (VERW(Q))
        IF (.NOT. LEES(1)) RETURN L
C       LAATSTE WOORD INGELEZEN
        IF ((N + BOOM(VOORW(P)))/2 .NE. VOORW(P-1)) GO TO 11
        BOOM (VOORW(P)) = HULP
        Q = Q-2
        GO TO 12
   11   VERW(Q-1)= HULP
        VERW(Q)= N
        P= P+2
   12   VOORW(P-1)= N
        VOORW(P)= HULP - 1
        RETURN
  100   CALL STORE(-1)
        CALL STORE(-1)
        IF(.NOT.LEES(1)) RETURN L
C       LAATSTE WOORD INGELEZEN
        WEL= .NOT. WEL
        IF(WEL) GO TO 101
        Q= Q+2
        VERW(Q-1)= HULP
        VERW(Q)= N
        RETURN
  101   BOOM(VOORW(P))= HULP
```

```
      P= P-2
      RETURN
      END

      LOGICAL FUNCTION LEES (J)
      INTEGER AUX (40)
      LEES = .FALSE.
      READ 11, AUX
   11 FORMAT (40R1)
      PRINT 12, AUX
   12 FORMAT (1X, 40R1)
      DO 10 I= 1,40
      IF (AUX(I) .EQ. 55B) GO TO 9999
C     TESTEN OP EEN SPATIE, DEZE HEEFT BCD-WAARDE 55 OCTAAL.
      LEES= .TRUE.
   10 CALL STORE (AUX(I))
 9999 CALL STORE(0)
      RETURN
      END

      SUBROUTINE STORE (I)
      COMMON BOOM (1000), BOOMP
      INTEGER BOOM, BOOMP
      IF (BOOMP .GE. 1000) STOP 7777
      BOOMP = BOOMP + 1
      BOOM (BOOMP) = I
      RETURN
      END

      OVERLAY (2,0)
      PROGRAM OPTBOOM
      COMMON BOOM (1000)
      INTEGER BOOM, N, P, Q
C
C     OPTIMALISEREN DOOR OP ELK NIVEAU DE LINKER- EN DE
C     RECHTERTAK ZOVEEL MOGELIJK MET ELKAAR IN EVENWICHT
C     TE BRENGEN
C
```

```
      Q = 2
      P = BOOM (Q-1)
   11 IF (NIVEAU (BOOM (P-2)  .LE. NIVEAU (BOOM(P-1)) GO TO 10
      N = BOOM (P-2)
      BOOM (P-2) = BOOM (N-1)
      BOOM (N-1) = P
      Q = BOOM (N-1)= N
      GO TO 11
   10 Q = P
      P = BOOM (Q-1)
      IF (P .NE. -1) GO TO 11
      END

      FUNCTION NIVEAU (I)
      COMMON BOOM (1000)
      INTEGER BOOM
      J = I
      NIVEAU = 0
   11 IF (J .EQ. -1) RETURN
      NIVEAU = NIVEAU + 1
      J = BOOM (J-2)
      GOTO 11
      END
```

C. controle kaarten en job indeling.

1. algemeen

Een job bestaat uit een file op kaarten, weer onderverdeeld in verschillende records. (Onder record wordt hier iets anders verstaan, dan onder record als vermeld in hoofdstuk 5.)
De job wordt afgesloten door een EOF kaart (een kaart met in kolom 1 een ponsing 6-7-8-9). Een record wordt afgesloten door een EOR kaart (een kaart met in kolom 1 een ponsing 7-8-9). Aangeraden wordt om de EOF kaart verkeerd om te ponsen. (d.w.z. ponsen in kolom 80 en kaart met blanke zijde naar voren houden.)
De controle kaarten bevinden zich in het eerste record van de job.
De volgorde is meestal:
<job kaart><request kaarten><rewind kaarten><verzoek tot vertaling>
<reduce kaart><verzoek tot executie><unload kaart>
Bestaat de job uit meerdere programma's, dan wordt na het verzoek tot executie toegevoegd:
<aanvraag geheugenruimte><verzoek tot vertaling><reducekaart>
<verzoek tot executie>, dit mag willekeurig herhaald worden.
Is een programma reeds vertaald, dan worden weggelaten:
<aanvraag geheugenruimte><verzoek tot vertaling><reduce kaart>.
Bij een verzoek tot vertaling wordt een record gelezen van de file, als gespecificeerd in de betreffende kaart (meestal INPUT, d.i. het kaartendek). Gebruikt een programma invoer van de file INPUT, dan wordt hiervoor ook een record gelezen. Iedere controle kaart bestaat uit een indicatie, gevolgd door eventuele parameters. Ontbreken de parameters, dan wordt de indicatie gevolgd door een . of een), staan er wel parameters, dan wordt de lijst voorafgegaan door een willekeurig symbool, mits geen letter, cijfer, . ,) of * , onderling gescheiden door een willekeurig symbool, mits geen letter, cijfer, . ,) , * of spatie; en afgesloten door een . of).
(Aangeraden wordt, om zonder parameters de . te gebruiken als afsluiter, en met parameters hetzij
1° lijst vooraf laten gaan door (, afsluiten door) of
2° lijst vooraf laten gaan door , en afsluiten door .;
en de parameters onderling te scheiden door een ,.)

Spaties mogen niet voorkomen, tenzij gebruikt als beginaanduiding van een parameter lijst.
Alle symbolen na de afsluitende . of) worden opgevat als commentaar (behalve eventueel symbolen in kolommen 79 en 80 van de jobkaart, zie appendix D).
Filenamen zijn een <name [1.1.f]>.

2. de job kaart.

De indicatie van de job kaart bestaat uit de job naam
(een <name [1.1.f]>).
Door de rekendienst wordt aan iedere gebruiker een job naam toegekend. Mogelijke parameters zijn:

1° Verzoek om tijd: T <octal number>, de tijd wordt opgegeven in seconden, als een octaal getal.

2° Verzoek om geheugenruimte: CM <octal number>, de geheugenruimte wordt opgegeven in woorden, als een octaal getal.

3° Verzoek om magnetische tape units: MT <octal number>, het aantal units mag uiteraard nooit meer zijn, dan het aantal aanwezige units.

4° Verzoek om extended core storage (ECS): EC <octal number>, het aantal aangevraagde ECS woorden is <octal number>*512.
(ECS is niet bij elke installatie aanwezig, waar wel, mag het opgevat worden als uitbreiding van het kerngeheugen).

Gebruik is, om als commentaar aan de jobkaart de naam van de gebruiker toe te voegen.

3. de request kaart:

REQUEST, <file naam>, MT.
(Er zijn andere vormen mogelijk, dit is de meest voorkomende vorm).
Verzocht wordt de file met als naam <filenaam> op een magnetische band te zetten. Voor een FORTRAN programma is de file naam altijd TAPE <octal number>, en eventuele in het verzoek tot vertalen, of verzoek tot executie vermelde files. Files, wier naam niet in een request kaart staan, worden op een disk gezet, en na afloop van de job vernietigd. OUTPUT, INPUT, PUNCH en PUNCHB mogen nooit vermeld

worden. Het aantal tapes aangevraagd in request kaarten moet ook aangevraagd worden in de job kaart.

4. de rewind kaart:

REWIND, <filenaam>, <filenaam>,... .
Dit geldt alleen voor files op magnetische tape: de tapes worden aan het begin ingesteld.

5. verzoek tot vertaling:

De simpelste vorm voor FORTRAN vertaling is FTN.
Eventuele parameters: (alleen de belangrijkste worden vermeld)

1° I= <filenaam>: Het programma staat niet op INPUT, maar op de betreffende file (in de vorm van ponskaart beelden).

2° B= <filenaam>: Het objectprogramma (d.i. vertaalde programma) wordt niet op LGO, maar op de betreffende file gezet (is de <filenaam> PUNCHB, dan wordt het programma geponst).

3° ROUND = <lijst van operatoren>: Indien deze parameter aanwezig is, worden deze operatoren vertaald in instructies, die ervoor zorgen dat het resultaat afgerond wordt, i.p.v. getrunceerd. Als operatoren zijn toegelaten: +, -, * en /

6. reduce kaart:

REDUCE.
Het aantal beschikbare geheugenwoorden wordt gereduceeerd van het aantal, als aangegeven in de jobkaart, tot het aantal, dat volgens de vertaler voldoende is voor het vertaalde programma. Dit zorgt ervoor dat het programma niet nodeloos veel geheugen in beslag neemt. (b.v. voor vertaling zijn 43000_8 woorden nodig, dit aantal is natuurlijk vrijwel nooit ook bij executie nodig).

7. verzoek tot executie:

Meest eenvoudige vorm: LGO.
Heeft de vertaler het vertaalde programma echter niet op LGO, maar op een andere file gezet, dan moet deze file vermeld worden. Was de file PUNCHB, dan kan het programma niet meer in deze job uitgevoerd

worden, bij een volgende job, kan het kaartenrecord uitgevoerd worden door de opdracht INPUT. (<filenaam>. is een afkorting voor de twee opdrachten:
LOAD,<filenaam>.
EXECUTE.
Wordt deze laatste vorm gekozen, dan moet een eventuele reducekaart tussen de load kaart en de execute kaart gezet worden.)

8. aanvraag geheugen ruimte:

RFL, <octal number>.
Deze kaart is nodig, wanneer, b.v. door een reduce kaart, het beschikbare geheugen te klein is geworden voor het volgende programma. Aangeraden wordt om voor 2e en volgende programma's in een job altijd opnieuw geheugen ruimte aan te vragen: (voor vertaling is 43000_8 woorden meestal wel voldoende).

9. unload kaart:

UNLOAD, <filenaam>,<filenaam>,... .
Dit geldt alleen voor files op magnetische tapes: de unit wordt ontladen.

10. diversen:

a. SWITCH, <digit>.
   Hier moet gelden: $1 \leq \text{digit} \leq 6$.
   Deze kaart is van belang als het programma gebruik maakt van de subroutine SSWITCH, in de normale situatie staan de switches uit, door deze controle kaart wordt de betreffende switch aangezet.
b. COMMENT.
   Deze kaart dient alleen voor het plaatsen van commentaar (zoals gebruikelijk na de . en t/m kolom 80).
   Er mogen willekeurig veel comment kaarten in het controle kaarten record voorkomen.

Voorbeeld:

```
Z07MC(CM43000, T170, MT1) WINTER-VAN VLIET
REQUEST(TAPE1, MT) TAPE P027R AANBRENGEN
REWIND(TAPE1)
COMMENT. TAPE1 STAAT NU KLAAR VOOR DE VOLGENDE PROGRAMMA'S
FTN (I= TAPE1, ROUND= +-*/, B=SCRATCH)

REDUCE.
SCRATCH. HET VERTAALDE PROGRAMMA OP SCRATCH WORDT UITGEVOERD.
UNLOAD (TAPE1)
RFL (43000) NODIG VOOR VERTALING
FTN.
SWITCH (1)
REDUCE.
LGO. NORMALE SITUATIE
7/8/9

<invoer voor eerste programma, dat eventuele resultaten op diskfile
 TAPE2 zet. Dit programma staat vermeld in ponskaartenbeelden op de
 file TAPE1:
 PROGRAM EERSTE (INPUT,TAPE2)>
7/8/9

<tweede programma, maakt gebruik van de diskfile TAPE2:
 PROGRAM TWEEDE (TAPE2, OUTPUT)>
6/7/8/9
```

D. codering symbolen:

symbool	H026	H029	symbool	H026	H029
A	12-1	12-1	Y	0-8	0-8
B	12-2	12-2	Z	0-9	0-9
C	12-3	12-3	/	0-1	0-1
D	12-4	12-4	+	12	12-6-8
E	12-5	12-5	-	11	11
F	12-6	12-6	0	0	0
G	12-7	12-7	1	1	1
H	12-8	12-8	2	2	2
I	12-9	12-9	3	3	3
J	11-1	11-1	4	4	4
K	11-2	11-2	5	5	5
L	11-3	11-3	6	6	6
M	11-4	11-4	7	7	7
N	11-5	11-5	8	8	8
O	11-6	11-6	9	9	9
P	11-7	11-7	=	3-8	6-8
Q	11-8	11-8	≠	4-8	7-8
R	11-9	11-9	.	12-3-8	12-3-8
S	0-2	0-2	)	12-4-8	11-5-8
T	0-3	0-3	$	11-3-8	11-3-8
U	0-4	0-4	*	11-4-8	11-4-8
V	0-5	0-5	,	0-3-8	0-3-8
W	0-6	0-6	(	0-4-8	12-5-8
X	0-7	0-7	spatie	geen	geen

Standaard is code H026; wordt code H029 gebruikt, dan moet in de job kaart in kolommen 79 en 80 een 29 geponst worden (aangeraden wordt om bij code H026 daar een 26 te ponsen, in verband met mogelijke toekomstige verrassingen). Het is mogelijk in een job naar een andere code over te gaan: pons in kolommen 79 en 80 van een EOR kaart een 26 of 29, en de volgende records worden in de betreffende code verwacht.

Index.

<procedure name>	4.9.b
<program>	7.a
<program name>	1.1.e
<program pack>	8.a
<program parameter>	7.1.e
<program parameter list>	7.1.d
<program parameter pack>	7.1.c
<real constant>	1.2.h
<relational expression>	2.7.2.a
<relational operator>	0.3.1
<repetition factor>	3.6.j
<replicator>	5.i
<return statement>	4.10.a
<rewind statement>	6.4.b
<scaling factor>	5.1.b
<second indication list>	7.e
<second type indication>	3.c
<secondary level>	8.1.e
<section indication>	8.2.3.a
<section list>	8.2.3.b
<section name>	8.2.e
<segment indication list>	8.2.b
<segment name>	8.2.d
<segmented program>	8.2.a
<sequential controller>	0.3.j
<sign>	1.2.1
<simple field descriptor>	5.f
<simple if>	4.4.d
<slashes>	5.h
<special character>	0.3.e
<spacing descriptor>	5.4.a
<start>	2.1.d
<start indication>	8.1.f
<starting address>	6.5.c