stichting
mathematisch
centrum

MC

REKENAFDELING MR 117/70 MEI

K.K. KOKSMA

EEN ALGOL 60 BIBLIOTHEEK SYSTEEM VOOR DE EL X8

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.

Inhoudsopgave

Voorwoord

Dit rapport is een weergave van de werkbesprekingen in maart en april 1970 over het MC-ALGOL 60 systeem. Daar het MC-ALGOL 60 systeem in voortdurende ontwikkeling is, heeft elke versie een eigen naam gekregen. Zo is in MR84 het PICO systeem beschreven. In dit rapport is het systeem MICRO64KL, kortweg MKL of M64KL beschreven. Het is een uitbreiding van het systeem MICRO48K. In het rapport wordt er dan ook vanuit gegaan dat de lezer met dit systeem bekend is. In het MKL systeem is het mogelijk om op een gemakkelijke wijze ALGOL 60 procedures voor te vertalen en in de bibliotheek op te nemen.

Hiervoor was het noodzakelijk dat de compiler LOAD and GO gemaakt werd. De wijziging van de compiler vergde veel studie van het systeem en van de compiler zelf.

Het werk dat uiteindelijk leidde tot het MKL-systeem begon dan ook met het bestuderen van MR84 samen met de heer E.G.M. Broerse. Eveneens tezamen met de heer Broerse schreef de auteur een voorloper van de gewijzigde compiler in een PICO-achtige omgeving. Het idee om een "kind" (zie blz. 16) in te voeren is ook afkomstig van de heer Broerse.

Toen ongeveer een idee verkregen was hoe de compiler gewijzigd zou moeten worden, werkte de auteur verder aan het bibliotheeksysteem.

Dit werd als uitbouw van MICRO48K op 1 januari 1970 operationeel als het MKL systeem.

De schrijver is veel dank verschuldigd aan de heer R. Brinkhuysen voor zijn hulp bij de voorbereiding van dit geschrift.

I-a Opbouw

Het bibliotheeksysteem is geponst op 28 ponsbanden. Een overzicht hiervan wordt gegeven op de volgende bladzijde.
In het systeem onderscheiden we:

1 monitor met submonitor
2 drumsection met autonome en semiautonome processen
3 running system
4 library + tabellen
5 compiler met catalogue
6 transformator + trafotable
7 systap.

Het bedrijfssysteem M64KL is een wijziging van M48K.
We onderscheiden nu wijzigingen en toevoegingen.

ad1 wijziging monitor zie I-b
toegevoegd de submonitor zie I-d-1

ad2 wijziging drum section zie I-b

ad3 ongewijzigd

ad4 tabellen toegevoegd

ad5 wijziging compiler zie I-c
toevoeging zie I-e

ad6 toegevoegd zie I-c

ad7 wijzigingen zijn van ondergeschikt belang

Overzicht banden MKL systeem

1	assemblageparameters	
2	geheugenverdeling	
	drumbudget	
	interface constanten	
3	monitor	
4	I/O conversion	
5	Drum section	
6		Reader section
7	autonome processen	Punch section
8		Line printer section
9	NXT TAPE SL	
10	RECODE en MICA	
11	submonitor	
12	software protection	CONSIDER, ERM, OBSERVE, DANGER
13	semiautonoom process	RECOVER
14	running system	
15	library	
16	semiautonoom process	DUMP
17	catalogue	
18		compiler
19		
20		
21	analysator	
22	bitstringgenerator	
23		
24		
25		
26		
27	transformator	
28	systap	

De bespreking van M64KL als geheel zonder voorkennis van M48K valt buiten de scope van dit rapport. We gaan er vanuit dat de lezer min of meer op de hoogte is van [1], [2], [3] en [5]. Alleen bij de wijziging van de compiler wijden we enigszins uit over de oude compiler in M48K.
Verder weerspiegelt de opbouw van M64KL zich direct in de opbouw van dit geschrift.
Het is dan ook bedoeld als een technisch rapport, dat van nut kan zijn bij maintenance en extensions.

I-b Wijzigingen in de monitor

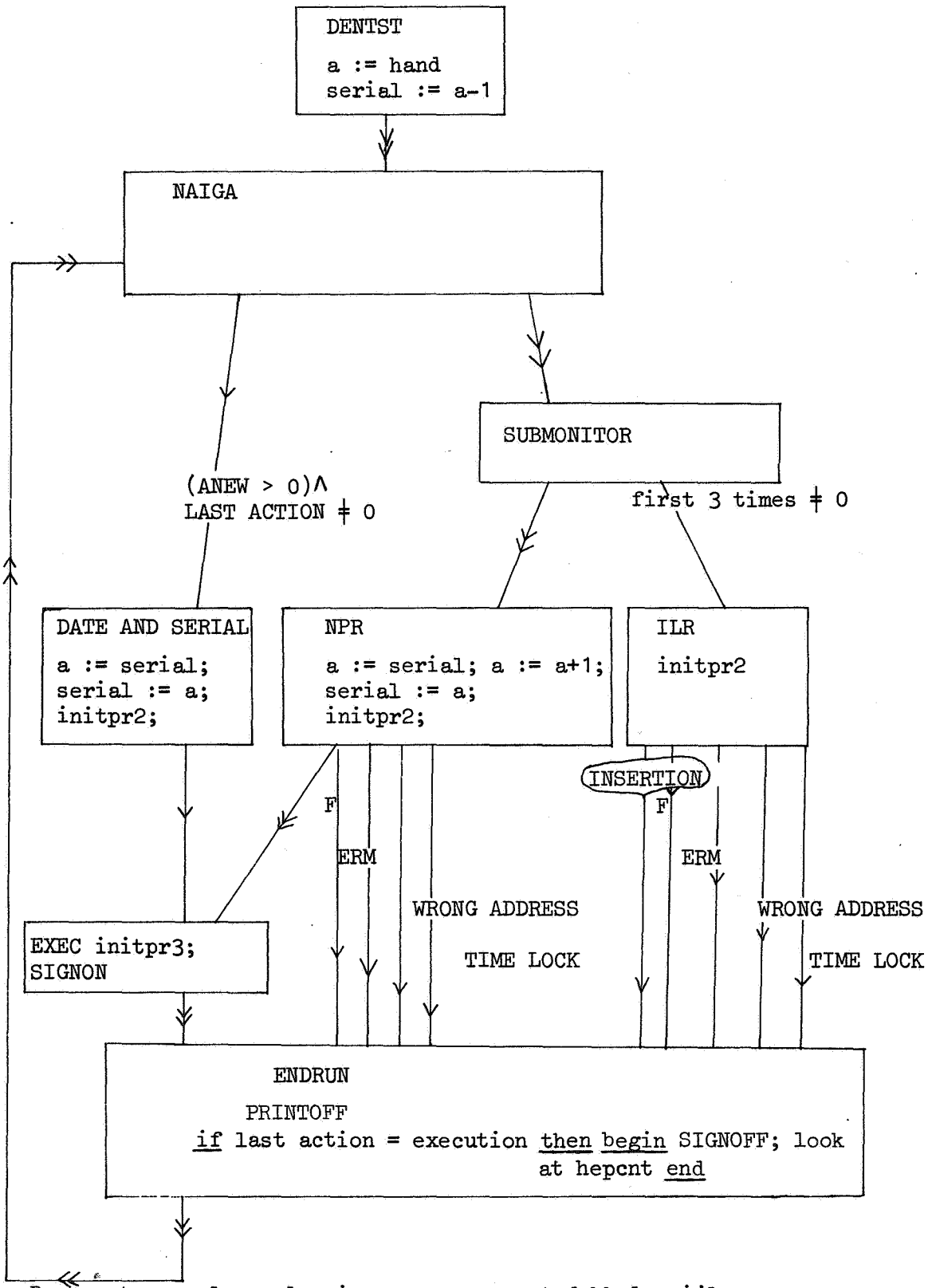
In [1] is MICRO48K verdeeld in een compiler (nagenoeg gelijk aan de compiler in [3]) en de rest waarvan de tekst dan in [1] is afgedrukt.

Wij volgen de verdeling uit sectie I-a. De kern van de monitor wordt gevormd door DENTST, NAIGA en ENDRUN. Doordat de teleprinter-output en de regeldrukker-output enigszins gewijzigd zijn, lijken DENTST, NAIGA en ENDRUN meer veranderd dan in werkelijkheid het geval is.

Het systeem wordt met DENTST gestart en doorloopt vervolgens steeds de cyclus NAIGA, ENDRUN. (zie fig. 1)
Van NAIGA naar ENDRUN lopen echter verschillende paden. Dit uit zich ook in regeldrukker-output, teleprinter-output en manipulatie met date en serial.

Het serialnr. wordt alleen bij ingang van analyse van een normaal programma met 1 opgehoogd. Bij herstart en ILR vindt geen ophoging plaats.

Bovendien krijgt men over de regeldrukker bij een herstart in M64KL geen extra lege pagina meer (met serialnummer van volgende programma) zoals in M48K.
Wel is er in de line printer-section een routine PRINT OFF toegevoegd, die zorgt dat de regeldrukker-output na elke job kan worden afgescheurd.



De meest normale cyclus is aangegeven met dubbele pijlen.

fig.1

De wijzigingen ten behoeve van het bibliotheeksysteem in de monitor zijn veelal van ondergeschikte betekenis. Zo is in de Drumsectie wat gewijzigd omdat de compiler en tabellen niet meer consecutief in het kerngeheugen staan. Ook het drumbudget is gewijzigd met het oog op MKLL.
In DENTST is een instructie SUBC (:FOUND LIBRARY) opgenomen. In FOUND LIBRARY wordt als het ware de bibliotheekadministratie (zie I-f) geinitialiseerd.
Als tegenhanger van de routine ENDRECDU moesten ACTIVE [7] en [8] in INITDR geinitialiseerd worden.
Ook ingelast is een stukje INFORM OPERATOR waarin aan het systeem gemeld wordt dat ILR geslaagd is. (zie I-d).

I-c Wijzigingen in de compiler

I-c-1 Inleiding

I-c-2 Wijziging van de compiler in verband met de bitstring

I-c-3 codeprocedures in M64KL

I-c-4 de transformator.

I-c-1. Inleiding

Voordat we kunnen overgaan tot bespreking van de aangebrachte wijzigingen is een korte inleiding over de compiler in MR84 op zijn plaats. Men leze in dit verband [10].

In MR84 wordt een programma in 3 scans gecompileerd. Ter illustratie diene het volgende programma:

```
begin integer i; goto L; L: EXIT end
```

In de eerste scan wordt een naamlijst aangelegd.

```
(deze bevat  i + gegevens i = een integer)
(            L + gegevens L = een label  )
```

Hierin komen in PRESCAN0 alleen de gedeclareerden voor.

In de tweede scan wordt aan i een adres toegekend. Dit wordt in de naamlijst vastgelegd. Bovendien wordt elke naam, die in de sourcetext voorkomt, opgezocht in de naamlijst. Wordt hij daar niet gevonden dan is hij dus niet gedeclareerd. In dit geval wordt de naam opgezocht in de catalogue van bibliotheeknamen. Dit proces wordt in ons geval dus toegepast op de naam EXIT.

In de derde scan komt dan de eigenlijke compilatie.

Hoe kunnen we ons dit vertaalproces nu voorstellen?
De compiler leest m.b.v. nxt sbl symbool voor symbool. Na een van geval tot geval variërend aantal aanroepen van nxt sbl komt de analysator in één van ± 150 mogelijke situaties. Deze situaties hebben alle een nummer het zgn. macronummer.
Zo zal begin gevolgd door een declarator door de analysator herkend worden als een blokingang.
Dit weerspiegelt zich in de ELAN sourcetext door de instructies:

```
A = 112
SUBC(: MACRO2)
```

In deze situatie is het echter ook nog van belang dat het hier gaat om een block van display level 1. In dit geval zal dan ook bij aanroep van MACRO2 in S de waarde 1 staan. Dit wordt bewerkstelligd door de routine DISP LVL. In de ELAN-tekst staat dan ook:

```
SUBC(: DISP LVL)
A = 112
SUBC(: MACRO2)
```

Dit is maar een eenvoudig voorbeeld van de parametrisering van de situatie. Er kunnen zich de volgende gevallen voordoen:

1 de parameter is een adres uit het objectprogr dat al bekend is. (S < 0).

2 er is een adres uit het objectprogramma vereist, dat nog niet bekend is, dan is de parameter 0 (S = 0)

3 de parameter is een pointer uit de naamlijst. (S > 0)

- 3a in de naamlijst staat een echt adres
- 3b in de naamlijst staat een rangnummer uit de bibliotheek
- 3c in de naamlijst staat een 0. (JU super locale label).

4 de parameter is een getal bv. display level (S > 0)

Door aanroepen van MACRO2 geeft de analysator te kennen dat aan het objectprogramma nu de instructies van situatie 112 moeten worden toegevoegd.

We kunnen de derde scan uit MR84 verdelen in 2 delen:de recognition en de macroprocessor.
In verband met het volgende is het goed enige aandacht te wijden aan de behandeling van constanten in MR84.
Tijdens compilatie van het ALGOL 60 programma wordt het objectprogramma in de kernen opgebouwd. In PRESCAN0 wordt elke constante, die in de sourcetext voorkomt, in het objectprogramma opgenomen. Daar er verder in PRESCAN0 niets aan het objectprogramma wordt toegevoegd begint het objectprogramma met een lijst van constanten. In de vertaalscan wordt nu de constante in deze lijst opgezocht en wordt bij vertaling het zo verkregen adres in het objectprogramma gebruikt.
Het objectprogramma in MR84 groeit door:

1. aanroep van PRODUCE via MACRO2
2. aanroep van SUBSTT of SUBST2
3. een bijzonder geval van 1 APD's, SWORD's
4. constanten, honorering variabelen

I-c-2.
Wijziging van de compiler in verband met de bitstring.

In M64KL is de macroprocessor in tweeën gehakt. In de derde scan is de macroprocessor een bitstring generator. De transformator vormt de bitstring dan later om in objectcode.

In M64KL wordt een bitstring gevormd door:

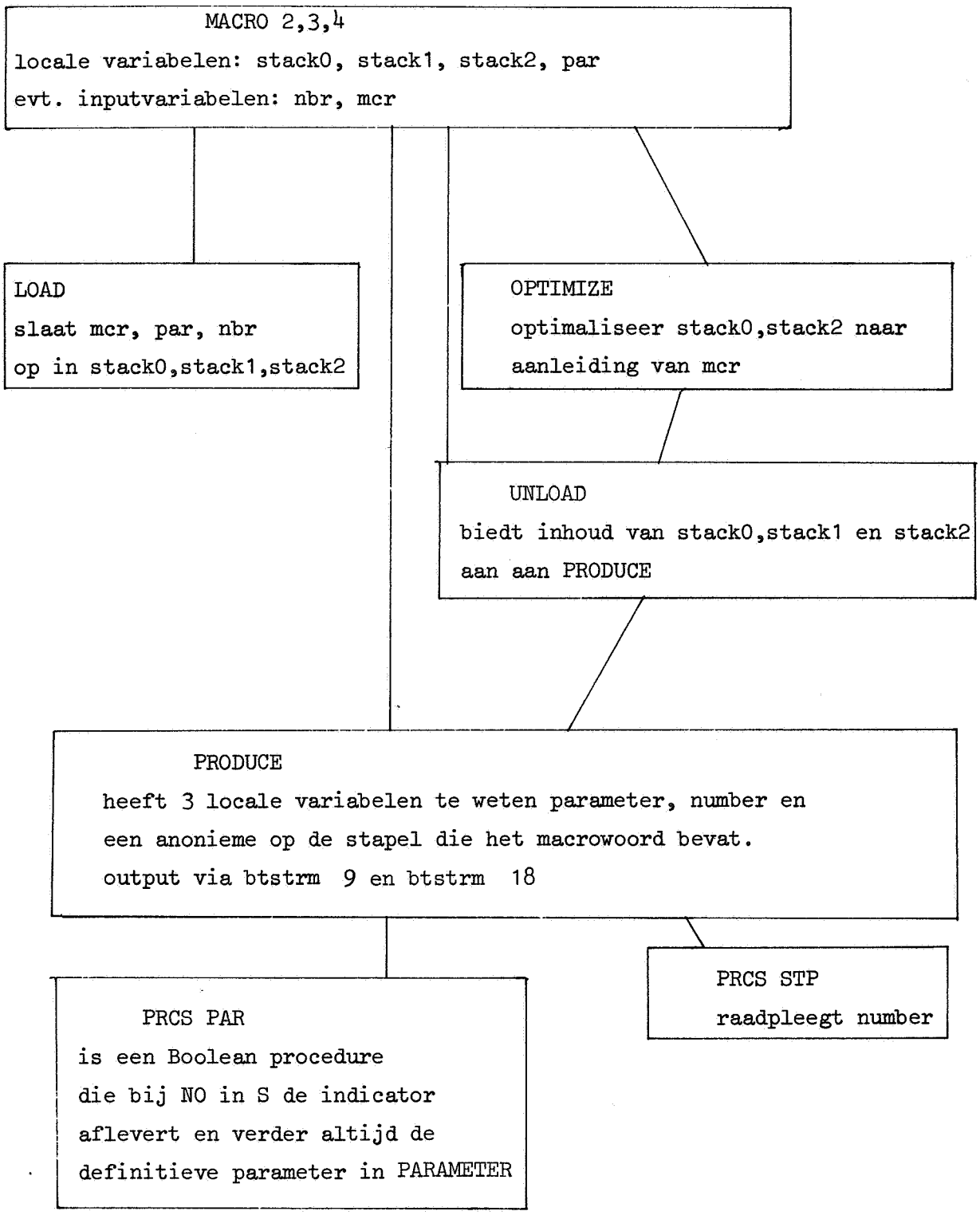
1 aanroep van SUBSTT of SUBST2

2 aanroep van PRODUCE via MACRO2

3 aanroep van PRODUCE via MACRO4: APD's, SWORD's

4 reservering ruimte voor variabelen, constanten

Een schets van de macroprocessor/bitstringgenerator is gegeven op blz. 11 .


MACRO 2,3,4
locale variabelen: stack0, stack1, stack2, par
evt. inputvariabelen: nbr, mcr
LOAD
slaat mcr, par, nbr
op in stack0,stack1,stack2
OPTIMIZE
optimaliseer stack0,stack2 naar
aanleiding van mcr
UNLOAD
biedt inhoud van stack0,stack1 en stack2
aan aan PRODUCE
PRODUCE
heeft 3 locale variabelen te weten parameter, number en
een anonieme op de stapel die het macrowoord bevat.
output via btstrm 9 en btstrm 18
PRCS PAR
is een Boolean procedure
die bij NO in S de indicator
aflevert en verder altijd de
definitieve parameter in PARAMETER
PRCS STP
raadpleegt number

Door het optimaliseren van de vertaling kan het zijn dat het drietal dat aan PRODUCE wordt aangeboden niet hetzelfde is als het drietal dat in tijd gezien het kortst daarvoor aan MACRO2 is aangeboden. De eerste 3 horen dus bij MACRO de tweede 3 bij PRODUCE.

Het grootste gedeelte van de bitstring wordt enerzijds gevormd door SUBSTT en SUBST2 die informatie in de bitstring doen opnemen, waarmee de transformator later de ketens kan bijwerken en anderzijds door MACRO2,3,4.

Bij de aanroepen van MACRO2, MACRO3 en MACRO4 kunnen we het volgende onderscheid maken:

a MACRO3 wordt aangeroepen door PAR LIST, SW DEC6, CST STR. Vertaaltechnisch is het de bedoeling dat de gevormde parameter als heel woord wordt opgenomen in het objectprogramma. Daar in een APD of SWORD een adres uit het objectprogramma mag voorkomen, kan het zijn dat de parameter gemodificeerd moet worden met (begin of pr ar -1). Is de condition YES bij aanroep van MACRO3 dan wordt niet gemodificeerd anders wel. Het directief dat aangeeft of er wel of niet gemodificeerd moet worden (258 of 257) wordt opgeslagen in nbr. Het kan tijdens de hele gang door MACRO heen tot het afgeleverd wordt aan PRODUCE ook dienen als nbr. Dit nbr is echter niet meer de entry van het macrowoord CODE (= entry 128 uit MCR LST). Daarom wordt in MACRO3 eerst nbr gezet op grond van het register A en daarna A gezet op 128 om toch later met het macrowoord CODE te werken.

b MACRO4 wordt aangeroepen in TRL CD.
Daar het kan zijn dat de ALGOL 60 programmeur zelf een macrowoord gemaakt heeft moet "mcr" al voor de aanroep van MACRO4 gezet zijn (evt. met zelfgemaakte macro) evenals "nbr" (waar in het geval van de zelfgemaakte macro het directief 259 komt te staan (not listed macro)).

c In alle andere gevallen wordt de macroprocessor aangeroepen door aanroep van MACRO2.

De behandeling van constanten.

Voor de constanten wordt in MKL tijdens PRESCANO een constantenlijst aangelegd. Een constante geeft tijdens PRESCANO aanleiding tot <257><<< value of constant >>><257><<< value of constant1>>>

(Zie ST NUM CS [7] blz. 192). Het begin van deze constantenlijst wordt aangegeven in: START OF CONSTANT LIST- Om aan te geven tot hoever deze lijst gevuld is gebruiken we de variabele DPO (Zie ook I-g).

Aanroep van bibliotheekprocedures.
Er zijn maar 2 macro's te weten IJU1 en ISUBJ die voorkomen in de vertaling van een procedure aanroep. Bij aanroep van MACRO2 is de parameter nog een pointer in de naamlijst. Binnen PRCS PAR wordt middels de routine ADDRSS uit de naamlijst het rangnummer van de bibliotheekprocedure opgehaald. Bovendien wordt een indicator <3> in de bitstring opgenomen. Er ontstaat een rij bitgroepen, die er als volgt uitziet:

<mcrnr IJU1 of ISUBJ> <3><<rangnummer>>

De bitstring

De output van de compiler is in MICRO48KL een bitstring. Deze bitstring is vervolgens input voor de transformator. De betekenis van de bitstring wordt dus als het ware vastgelegd door de werking van de transformator. De transformator mogen we ons voorstellen als een cyclisch programma dat telkens na verwerking van een significant stuk bitstring in de uitgangstoestand terugkeert.
We onderscheiden nu de volgende significante stukken bitstring:
(Evenals in [4] noteren we <informatie> als de informatie verpakt is in 9 bits, <<informatie>> als de informatie verpakt is in 18 bits, en <<<informatie>>> als de informatie verpakt is in 27 bits).

stuk bitstring	toelichting / actie transformator
<macronummer>	als par part = 0 is er geen parameter
<macronummer> <<parameter>>	is de kind 2 of 3 dan geeft de kind alleen genoeg informatie over de behandeling van de parameter
<macronummer> <0> <<parameter>>	de parameter moet gemodificeerd worden
<macronummer> <1> <<parameter>>	de parameter is een dynamisch adres
<macronummer> <2> <<parameter>>	de parameter hoeft niet gemodificeerd te worden
<macronummer> <3> <<parameter>>	de parameter is een rangnummer uit de bibliotheek
<256>	einde bitstring
<257> <<<informatie>>>	De transformator voegt een geheugenwoord aan het objectprogramma in opbouw toe en vult het met informatie
<258> <<<informatie>>>	De transformator voegt een geheugenwoord aan het objectprogramma in opbouw toe en vult het met informatie na modificatie met correctie
<259> <<<not listed macro>>>	Na retrieval van het macronummer werkt de transformator normaal verder met het macrowoord. Daar we in codeprocedures de mogelijkheid hebben zelf een macro te maken moet dat macrowoord apart in de bitstring opgenomen worden. Daarna volgt dan evt. nog een indicator en een parameter.
<260> <<adres>> <<handvat keten>>	De transformator modificeert het adres en het handvat keten. Daarna werkt hij de bijwerkketen beginnende bij handvat keten af en geeft alle instructies het gemodificeerde adres als adresoperand.
<261> <<rangnummer uit bibliotheek>>	De transformator voegt 2 woorden aan het objectprogr in opbouw toe; vult deze met het rangnummer en vult trafotable [rangnummer] met het adres van het eerste net gevulde woord uit het objectprogramma.

<262> <<count>>	De transformator verhoogt de instr cntr met count
<263> <<dp0>>	
<264> <<begin of program>>	
<265> <<end of program>>	
<266> <<reladress>>	
<267> <<handvat keten>>	Analoog aan het directief <260> als adres dient nu de heersende waarde van instr cntr
<268> <handvat keten>	Analoog aan het directief <260> en <267>

In het voorgaande hebben we gezien dat in de bitstring soms een parameter wordt opgenomen.
Deze parameter is echter niet altijd de waarde die in S staat bij aanroep van MACRO2.
Het is de taak van de routine PRCS PAR uit deze waarde van S gecombineerd met het macrowoord de definitieve parameter te maken. Voor deze functie van het macrowoord is de codering van de macrowoorden gewijzigd.
Voor de codering van elk macrowoord afzonderlijk zie IV-a.
De bits in het macrowoord betekenen in MKL:

d26	altijd 0
d25 - d21	Breact
d20 - d12	instr prt
d11 - d10	par prt
d9 - d8	instr nbr
d7 - d4	opt nbr
d3 - d2	kind
d1	opt op
d0	sat

De kind is ingevoerd om aan de transformator hetzij direct in TRANSF zelf hetzij in TRANSL via de indicator door te geven hoe de parameter behandeld moet worden.
De overige bits hebben dezelfde betekenis en functie als in MR84. De bits van "instr prt" bepalen een entry in de INSTRUCT LIST. Hoeveel consecutieve instructies de transformator uit de INSTRUCT LIST zal copieëren vanaf genoemde entry wordt bepaald door "inst nbr". Is "par part" = 0 dan hoeven deze instructies niet geparametriseerd te worden. Is "par part" > 0 dan bepaalt zijn waarde welke instantie geparametriseerd wordt.
De variabele "Breact" wordt aalleen binnen PRCS STP (process stackpointer) geraadpleegd. Hieruit wordt afgeleid hoe deze instructies tijdens runtime de stapel zullen beïnvloeden.

Daar de kind informatie verschaft over de parameter is hij dus eigenlijk alleen van belang als van de corresponderende macro par part > 0 is.
De kind kan nu 0,1,2 of 3 zijn:

kind = 3	de transformator zal de aan de bitstring meegegeven parameter modificeren met correctie
kind = 2	de transformator zal de aan de bitstring meegegeven parameter niet modificeren.
kind = 1	de routine prcs par moet nu aan de hand van de parameter en de waarde van kind (hier dus 1) uitmaken of de parameter met correctie gemodificeerd moet worden of niet.
kind = 0	De routine prcs par moet aan de hand van de parameter uitmaken of we te maken hebben met: 1) een rangnummer uit de bibliotheek 2) een dynamisch adres 3) een absoluut adres Vervolgens moet dit via een indicator in de bitstring worden opgenomen.

<u>Conclusie</u>	par part = 0	geen indicator	geen parameter
	kind 3 of 2	geen indicator	wel een parameter
	kind 1 of 0	een indicator en een parameter.	

I-c-3.

Codeprocedures in M64KL

Codeprocedures kunnen slechts geprecompileerd worden. De compiler geeft dan ook een foutmelding (402) als in de sourcetext van een normaal programma een declaratie van een codeprocedure voorkomt.

Neemt men als bibliotheekprocedure een codeprocedure op dan moet de heading voldoen aan de regels van ALGOL 60. De body moet bestaan uit een aantal getallen of namen onderling gescheiden door komma's en omsloten door string-quote's.
Aan de syntax van een codeprocedure zijn dus betrekkelijk weinig eisen gesteld. Afgezien van de foutmeldingen 396 t/m 401 (Zie het MC-ALGOL 60 systeem voor de X8, voorlopige programmeurshandleiding) worden codeprocedures niet beveiligd vertaald.
Het is zo erg makkelijk een codeprocedure te schrijven die het systeem opblaast.
Dit kan op 2 manieren:

1) door opname van een instructie als

   M[3000] = F

   waarbij de monitor beschadigd wordt
2) doordat hoewel de instructies op zichzelf wel goed zijn de stapel te hoog rijst.

Een normale aanroep van een codeprocedure tijdens executie heeft een "transfer of control" tot effect naar de eerste instructie, die naar aanleiding van de vertaling van de body gemaakt is.
(Wordt de codeprocedure gebruikt als actuele parameter dan is dat de tweede instructie).

Welke bitstring de compiler bij een bepaalde sourcetext vormt wordt bepaald door de routine TRL CD (Translate Code). Hoe de transformator hierop reageert vindt men op blz. 18 .

De zinnige stukjes bitstring, die kunnen ontstaan ten gevolge van de compilatie van een codeprocedure zijn:

a <mcrnr> <<par>>

b <mcrnr>

c <257> <<<getal>>>

d <258> <<<getal>>>

c <259> <<<macrowoord>>> <<par>>

f <259> <<<macrowoord>>>

Men kan dit bereiken door in de code body op te nemen:

in geval a: een mcrnr (Zie IV-a) van een macro met par part > 0, gevolgd door een "identifier" of een getal. Deze identifier moet op dat ogenblik in het programma + systeem zin hebben.

in geval b een mcrnr van een parameterloze macro

in geval c het getal 257 gevolgd door een komma gevolgd door een integer binnen de integercapaciteit.

in geval d het getal 258 gevolgd door een komma gevolgd door een integer binnen de integercapaciteit.

in geval e een getal > 511, dat door de compiler als macrowoord wordt beschouwd, gevolgd door een identifier of getal.

in geval f analoog aan e echter zonder laatste identifier of getal.

I-c-4.
De transformator

Het is de taak van de transformator de bitstring om te zetten in objectcode. De transformator kan van het RECOVER process bits krijgen door de instructies SUBC(BTSTREAM9), SUBC(: BITSTREAM18) of SUBC(: BITSTREAM27). De transformator is een cyclisch programma. Tijdens elke slag wordt één van de rijtjes bitgroepen opgesomd in I-c-2 verwerkt. De meeste slagen geven aanleiding tot groei van het objectprogramma. Naast de rij van instructies, waaruit een programma bestaat, moeten ook de stuurvariabelen bekend zijn.

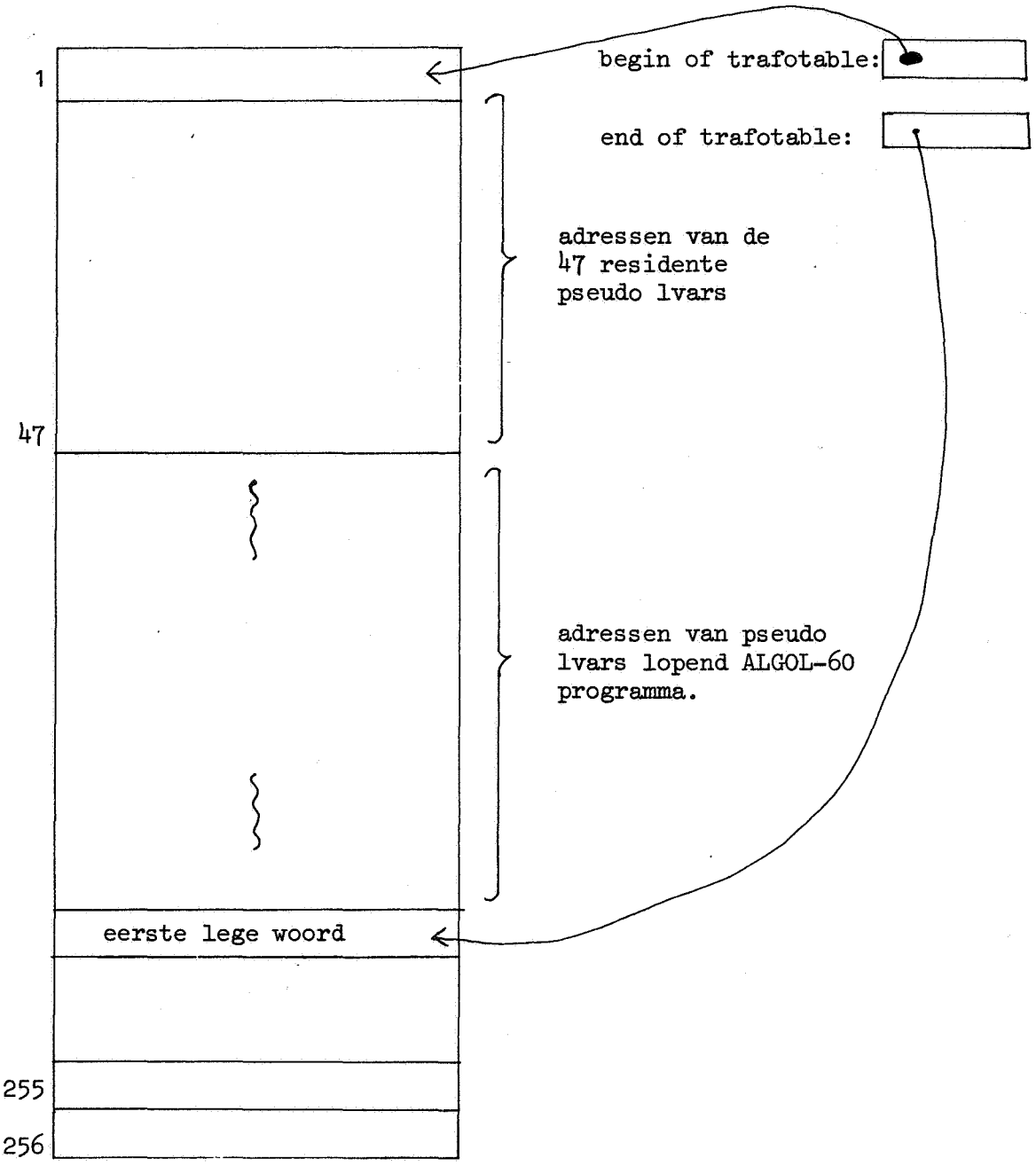
Voor het objectprogramma zijn dit dp0 (= startwaarde D register), begin of program (= startwaarde OT register), end of program (= startwaarde B register), en reladress (wordt gebruikt om end of program van het totale *) programma te berekenen).
Zoals duidelijk zal zijn bij nadere inspectie zijn slechts een zeer klein gedeelte van alle mogelijke rijen bitgroepen relevant voor de transformator. Er is dus zeer veel redundantie. Dit is bijzonder prettig omdat het hier ook om zo'n essentiële stroom gaat. Een stroom namelijk die omgevormd wordt tot een objectprogramma, dat de volledige controle over de machine krijgt en waarin dingen als software protectie moeten werken.
Omdat er nu zoveel redundantie is gaat het zeer snel mis als er een fout in de bitstring ontstaan is en hoeft de bitstring niet ook nog eens in de software bewaakt te worden.
De inrichting van de bitstring is flexibel.

Er zijn in MKL	202	macronummers en er is plaats voor	255
" " " "	4	indicatoren " " " " "	512
" " " "	13	directieven " " " " "	255

Uiteraard is de software ook zo gemaakt dat toevoeging van een macronummer directief of indicator eenvoudig kan.

*) Voor definitie zie I-d-1

Tot slot schetsen we de structuur van de trafotable.



Opm. In MKL vervult "end of trafotable" geen wezenlijke functie. De trafotable behoort niet tot de bibliotheekadministratie maar tot de werkruimte van de compiler (en wel van de transformator). Het zou later misschien aanbeveling verdienen de totale en primaire behoefte niet aan te geven in de crosstable maar in de trafotable. Hierdoor zou dan de bibliotheekadministratie zelf re-entrant worden. In dat geval gaat end of trafotable natuurlijk wel een rol spelen.

I-d Toevoegingen aan de monitor.

I-d-1 submonitor, make start1 orders
I-d-2 dumpsection
I-d-3 recover section

I-d-1.
De submonitor

In het bedrijfssysteem MICRO64KL kan een job op 2 verschillende wijzen opgevat worden. De submonitor bepaalt deze keus op grond van de variabele "first 3 times" en legt de keus dan verder voor die job vast in de variabele "HMODE" (handling mode). *)
Voor de ene opvatting wordt NPR (normal program) doorlopen, voor de andere ILR (insert library routine) *). Voorts zijn in monitor nog enkele korte routines opgenomen.
Dit zijn: INFORM PROGRAMMER, MAKE START1ORDERS, ASSURE LIBRARY, en UPDATE LIBRARY.

We zullen nu achtereenvolgens NPR en ILR bespreken.

Terminologie. In het volgende spreken we over "totale programma", "hoofd programma" en "bibliotheekprocedures". Er geldt: "totale programma" = "hoofdprogramma" + "bibliotheekprocedures".

*) De submonitor is dus de plaats waar de routines, die uitbreiding van MICRO64KL met bv. de faciliteit van bibliotheekprogramma's beogen, zouden moeten worden opgenomen.

NPR valt uiteen in twee delen.
Het eerste zorgt ervoor dat de bitstring op de trommel komt en de start1-order om het objectprogramma van de trommel terug te halen in USER[-1] en USER[-2] wordt opgebouwd (USER is vast met +0 gevuld). Voor het zetten van de bitstring op de trommel kan NPR beroep doen op de DUMPsectie. Aanzetten van het semi-autonome dumpproces gaat met INIT BITSTRM. Na afloop van de analyse zorgt NPR ervoor dat:

1 indien er nog bits in het stringword zitten deze in de buffer gezet worden (FORCE LAST BITS)

2 uitgerekend wordt hoeveel woorden de bitstring op de trommel in beslag zal nemen (ASK DUMPEND).
Hij hoeft er dus op dat moment nog niet te staan.

3 een eventuele half volle buffer naar de trommel gestuurd wordt (FINISH BITSTRM)

4 gewacht wordt tot de bitstring op de trommel gearriveerd is (END DUMP)

In USER, USER[-1] en USER[-2] staat een start1orderketen, die maar één start1order bevat. Voor het RECOVERprocess is er geen verschil tussen deze start1orderketen en die, die gemaakt wordt door MAKE START1ORDERS.

Het tweede gedeelte van NPR draagt er zorg voor dat de bitstring van de trommel gehaald wordt. Dit gebeurt door aanroepen van de transformator. (TRANSF).
NPR heeft ten opzichte van TRANSF 2 taken:
1. Het passend zetten van de "instr cntr".
2. Het aanzetten van het RECOVER process (INIT BITSTREAM).

Ten opzichte van INIT BITSTREAM en het RECOVER process als geheel heeft NPR de taak voor een start1orderketen te zorgen, waarvan het handvat in de variabele "label" moet staan.
Voor een normaal programma worden 2 start1orderketens gemaakt. De ene staat in USER en de andere direct achter het objectprogramma.

Opmerking MAKE START1ORDERS wordt uitgevoerd nadat het hoofdprogramma getransformeerd is. De informatie om deze "start1orders" te maken is echter al na PRESCAN1 bekend. De hier gekozen oplossing heeft het voordeel dat er in de kernen maar 3 woorden gevuld hoeven te zijn met informatie om de bitstring van een programma van de trommel te halen.
Voor de eenvoud van programmering wordt gewacht tot het hele hoofdprogramma getransformeerd is.
Nadat de START1ORDER keten voor de bibliotheekprocedures gemaakt is, worden deze stuk voor stuk van de trommel gehaald. Voor elke procedure wordt opnieuw de transformator aangeroepen.
Daar het RECOVER process semi-autonoom is hoeft het maar één keer aangezet te worden voor alle procedures. Wel moet NPR het proces na transformatie van de ene procedure in de pas brengen voor de volgende. (SKIP REST OF BUF).
De waarden van dp0 en begin of program na transformatie van het hoofdprogramma moeten gered worden op de stapel, omdat dp0, begin of program, end of program en reladress output parameters van de transformator zijn. ("dp0" en "begin of program" staan na uitvoering van NPR op de waarden die ze hadden na transformatie van het hoofdprogramma. Hiervan wordt aan het begin van de executie in RUN gebruik gemaakt).

NPR heeft nu bovendien de taak om van de omgevormde procedures en het omgevormde hoofdprogramma het totale programma te maken. Dit houdt in dat

1 de SUM OF MAXIMA bepaald wordt, dat is een getal dat aangeeft hoe hoog de stapel binnen één block kan rijzen, voordat er weer op stapelgroei gecontroleerd wordt, en
2 dat de pseudo lvars gevuld worden met de adressen waar de bibliotheekprocedures terechtkomen.

Opmerking Dat de pseudo lvar's gevuld worden op NPR-niveau heeft 2 voordelen:

1 elkaar aanroepende procedure's vormen geen moeilijkheid
2 wordt MICRO48KL uitgebreid met de faciliteit van bibliotheek programma's dan is een geprecompileerde bitstring bestand tegen wijziging van de tekst van gebruikte procedures zolang de crossreference dezelfde blijft.

We gaan nu over tot de bespreking van ILR.

Van een programma, dat aangeboden wordt om procedures in de bibliotheek op te nemen, wordt eerst gecheckt of het aan de Syntax voor bibliotheek-vulprogramma's voldoet (Zie I-e-1). Daarna wordt elke procedure afzonderlijk geprecompileerd. Dit is mogelijk doordat TRUNCATE (een procedure uit PRESCAN0) van het text array als het ware een aantal programma's maakt van de vorm begin <procedure declaration>; end

In de eerste analysegang wordt de catalogue uitgebreid met de namen (en de gegevens over de formelen) van de nieuwe bibliotheekprocedures. Daar dit de enige informatie is, die de compiler nodig heeft om een programma te vertalen, mogen in elk volgend programma deze namen in de sourcetext voorkomen.
De vertaling van bibliotheekroutines, die elkaar wederzijds recursief aanroepen, is hierdoor gewaarborgd.

In de analysegang voor de procedure afzonderlijk wordt:
1 de relocateerbare bitstring gevormd.
2 de primaire behoefte van de procedure bepaald en opgenomen in de infotable.
3 gebruik gemaakt van het feit dat het systeem weet dat het programma correct is. (Er wordt geen statische display aangelegd, geen jump over de declaraties geprogrammeerd en geen blokingang vertaald).

Komt een constante in 2 verschillende procedure's voor dan zal hij dus ook 2* op de trommel staan. *) Het is duidelijk dat een procedure, die zichzelf recursief aanroept of waarin own variabelen gedeclareerd worden, bij afzonderlijke vertaling geen bijzondere logische moeilijkheid vormt.

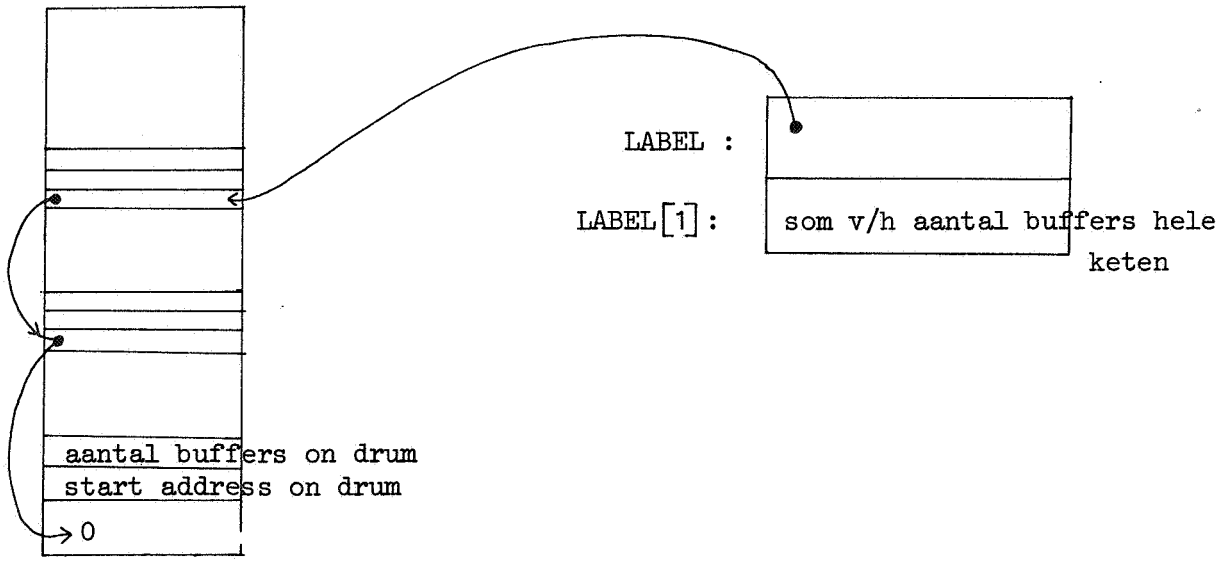
*) Ook in MICRO48K staat een constante zo vaak in het objectprogramma als hij in de sourcetext staat. Het objectprogramma geproduceerd door MICRO64KL kan alleen langer zijn door de pseudo lvars en the leading-zero of the pseudo lvars.

Ten behoeve van de bibliotheekadministratie werkt ILR de crosstable bij.*)
Ten behoeve van de transformator breidt ILR de trafotable uit.
De compiler verwerkt als CMODE=2 een getrunceerd text-array. Dit is natuurlijk niet het geval voor de eerste procedure. Dit wordt echter gecompenseerd doordat ILR beginsafe, wordsafe en shiftsafe passend initialiseert voordat hij de cyclus gelabeld met NEXT PROCEDURE ingaat.

Het DUMP Process is semi-autonoom. Het wordt dan ook aangezet voor de cyclus vertalingen van de afzonderlijke procedures en uitgezet daarna. De procedures FORCE LAST BITS en ASK DUMP END krijgen hier dan ook pas echt zin (Zie NPR en Dumpsectie).
ILR meldt voorts aan de programmeur voor elke procedure:
naam , nummer , lengte in de kernen , lengte op de trommel.
ILR meldt aan de operateur dat de procedures zijn opgenomen door aanroep van INFORM OPERATOR.

MAKE START1ORDERS

In onderstaand plaatje is een START1ORDER keten getekend.
Bij aanroep van INIT BITSTREAM staat het handvat van de START1ORDER keten in de variabele "label".



*) De bibliotheekadministratie bestaat uit 3 tabellen: crosstable, infotable en catalogue. De infotable wordt in TRANSL door FILL INFOTABLE bijgewerkt en de catalogue in INSPECT DECL.

In MAKE START1ORDERS wordt een START1ORDER keten opgebouwd, die voor elke library procedure die op de trommel staat, een start1order bakt. Beslaat een procedure dus 1 woord op de trommel dan wordt toch een hele buffer opgehaald.
De informatie om de start1orders te maken wordt nu gehaald uit de pseudo lvars en de library tabellen.
Om in de rij pseudo lvar's bij te houden, welke verwerkt zijn, wordt "pntr" gebruikt. Deze pointer is tevens output parameter van deze routine en wordt gebruikt in het tweede stuk van NPR om voor alle bibliotheekprocedures ook werkelijk de transformator aan te roepen.

I-d-2. Dump section

Voor het sturen naar de trommel van de bitstring is een dumpsectie ingelast.
In de compiler kan besloten worden om 9, 18 of 27 bits te dumpen. Deze bits moeten meegegeven worden in S gevolgd door SUBC(BTSTRM9) of SUBC(:BTSTRM18) of SUBC(:BTSTRM27).
In het systeem moeten we ons dump denken als een semi-autonoom proces. Het proces wordt dus niet geinitialiseerd in DENTST maar in NPR of ILR nadat CMODE gezet is en voordat de compilatie begint.
De plaats op de trommel, waar de bitstring terechtkomt is voor ILR en NPR verschillend. Tevens is het ook niet zo dat DUMP het "reverse process" kan activeren. Wel is het zo dat, als binnen één bibliotheekprogramma meerdere procedures worden aangeboden, het DUMP proces dan ten opzichte van die compilaties autonoom is. Terwijl hij nog dumpt voor de ene procedure kan hij de andere al wel vast compileren.
Nu is het zo dat we dat stuk string dat met een library procedure correspondeert altijd een geheel aantal woorden beslaat. De meest significante bits van het eerste woord behoren nu altijd tot de eigenlijke bitstring, terwijl de 18 minst significante bits van het laatste woord daar niet toe hoeven te behoren.
De dump sectie behoeft niet sterk resident te zijn. Hij is als het ware een compiler functie. Wat anders is echter de routine END RECDU.

Hier komen we in aanraking met een enigszins interressant probleem. In NAIGA werd vroeger de compiler doorlopen. Nu ILR of NPR. ILR gebruikt DUMP. NPR gebruikt DUMP & RECOVER. Komt het systeem in ENDRUN dan vraagt het of DUMP en RECOVER nog actief zijn, terwijl deze processen misschien nooit gestart zijn. Daarom moet er in DENTST vastgelegd worden dat DUMP en RECOVER niet actief zijn. (Dit gebeurt in INIT DR). Het systeem kan nl. langs verschillende paden in ENDRUN komen: F-inslag, fout tijdens executie, dangerous etc.
(Daarom wordt in END RUN altijd de instructie SUBCD(:ASSURE LIBRARY) uitgevoerd. Het aardige is nl. dat het systeem in ENDRUN alleen maar weet wat de HMODE is maar niet hoever NPR of ILR gevorderd waren. Dit hoeft het ook niet te weten, de prijs is echter dat er soms overbodige instructies gedaan moeten worden).
Daarom moet er naar END DUMP gevraagd kunnen worden zonder dat de DUMP-sectie in de kernen hoeft te zijn. En zonder dat er een aanwezigheidsadministratie van bijgehouden wordt. Hiervoor is er dus een aparte sterk residente routine END RECDU. De bedoeling is als het ware dat in ENDRECDU geverifiërd wordt dat alle semi-autonome processen ophouden. Helaas wordt op ENDTRANS op een andere wijze gewacht.

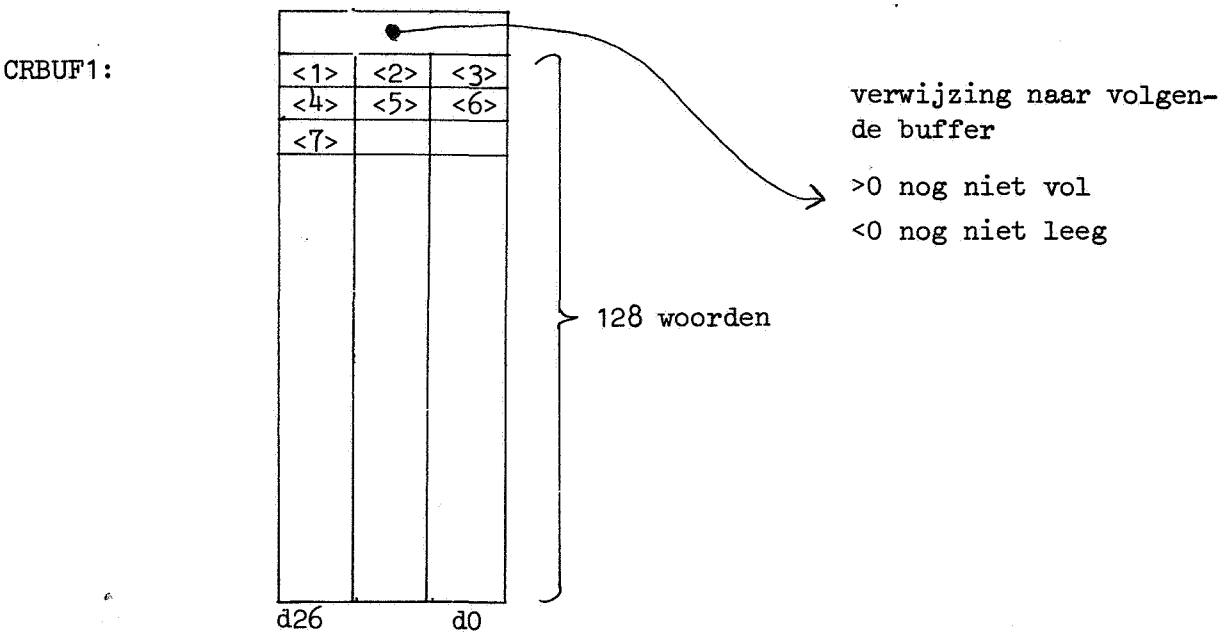
Bij een normaal programma moet de startlorder als resultaat in de kernen overblijven. Bij een bibliotheekprocedure moet de bibliotheekadministratie opgehoogd worden.
In beide gevallen moet eerst voor een bitstring van een geheel aantal worden gezorgd via aanroep van FORCE LAST BITS. Vervolgens moet uitgerekend worden tot hoever de trommel met deze string gevuld gaat worden.

Opmerking

We onderscheiden autonome en semiautonome processen. De autonome processen worden op gang geholpen voor zover mogelijk in DENTST. De semiautonome processen worden op last van de submonitor of het vertaalde objectprogramma op gang geholpen en in ENDRUN wordt gewacht tot ze tot rust gekomen zijn.

Dit gebeurt aan de hand van de DUMP administratie in één doofheid (je moet weten of de buffer op de trommel of in de kernen moet tellen).
In NPR moet na analyse geverifieerd worden of er nog een half lege buffer naar de trommel moet (FINISH BITSTRM).
In ILR wordt doorgegaan met de compilatie van de volgende procedure. Dit maakt het DUMP-proces dus echt een semi-autonoom proces.
De routines TO DRUM en PROCESS8 zijn zeer gelijkend op analoge routines in M48K.
In de dumpsituatie kan echter de trommel vollopen. Het dumpen moet dan gestopt worden terwijl de analyse doorgaat.
Dit wordt gedaan door het vullen van BTSTRM9 met :STOP DUMPING. Wordt tijdens analyse een fout ontdekt dan wordt eveneens het dumpen stopgezet, hetgeen een kleine tijdbesparing geeft bij "foute programma's".
Aan het eind van de analyse is de consequentie van het feit dat het dumpen stopgezet kan worden, dat nagegaan moet worden in FORCE LAST BITS of het überhaupt wel zinnig is iets te "forcen"

<u>De structuur van de dumpbuffers</u>.
Wordt aangeboden <1> <2> <3> <4> ....
dan ziet een volle dumpbuffer er als volgt uit:



Bij buffers die "gepakt" zijn spelen altijd:

1 een marker die aangeeft of hij er is

2 een shift

3 een positie wijzer

4 een methode om uit te maken of de buffer vol is

5 wanneer gaat informatie naar de buffer

ad 2 BTSTRM9 is een variabele gevuld met

of :NEXT STRWRD

of :EMPTY WORD

of :NINE INWORD

of :STOP DUMPING

doordat als er iets gedumpt moet worden dit hoofdzakelijk gebeurt middels de instructie SUBC(BTSTRM9) wordt in BTSTRM9 als het ware de shift bijgehouden.

ad 3 Als positiewijzer fungeert de variabele DUPOS. Komt de buffer aan de beurt om vanuit de compiler gevuld te worden dan wordt DUPOS geïnitialiseerd op:CRBUF1. In de wijzer staat dus steeds een absoluut adres. Dit heeft een voordeel ad5 maar in zekere zin een nadeel ad4.

ad 4 De variabele BUFND wordt tegelijkertijd met DUPOS geïnitialiseerd op CRBUF1 + :RECBULEN. Door steeds bij ophoging te vragen of DUPOS nu gelijk is geworden aan BUFND weet het systeem of de buffer nu vol is of niet.

ad 5 De eerste 9 bits worden in een "tussenbuffertje" van 1 woord opgeslagen. Hiervoor fungeert de variabele STRWRD. Ze worden meteen op de meest significante bits gezet. De volgende <informatie> komt op de volgende 9 bits te staan.
Worden daarna in S aan BTSTRM nog eens 9 bits aangeboden dan tellen we bij S logisch het STRWRD op en vullen een nieuw bufferwoord.

Redelijk vaak zullen in plaats van 9 bits 18 bits worden aangeboden. In dit geval kan het zijn dat er juist 9 bits in het "tussenbuffertje" STRWRD zitten dat wil zeggen dat BTSTRM9 gevuld is met :NINE IN WORD.
Logisch is het duidelijk dat we in dit geval bij de 18 bits in S de 9 bits uit STRWRD kunnen optellen en een nieuw geheugenwoord in de buffer vullen. Hiertoe dient dan ook de optimalisatie in BTSTRM18.

Opgemerkt zij dat als BTSTRM9 gevuld is met :NINE IN WORD hij niet gevuld kan zijn met :STOP DUMPING, d.w.z. er wordt nooit meer iets in een buffer gezet als BTSTRM9 gevuld is met :STOP DUMPING.

Daar 27 bits minder vaak voorkomen is daar geen optimalisatie voor geprogrammeerd.

I-d-3.
De recoversectie.

De recoversectie lijkt sterk op de dumpsectie. Toch is er een logisch verschil.
Bij de start van de analyse zijn de dumpbuffers present voor de compiler.
Bij de start van de opbouw van het objectprogramma moet gewacht worden tot de buffers van de trommel zijn gekomen.

Is bij de DUMP-sectie een buffer vol dan wordt er gewacht tot de volgende buffer naar de trommel getransporteerd is (leeg is).

Is bij de RECOVER-sectie een buffer leeg dan wordt er niet gewacht tot de volgende vol is. Wel worden RECPOS en BUFEND geïnitialiseerd op de volgende buffer. Pas als het eerste nieuwe woord uit de buffer gehaald moet worden wordt gewacht tot de buffer present is.

Bij de 128 aanvragen om één nieuw woord uit de buffer zijn er dus bij RECOVER 2 bijzondere: de eerste en de laatste aanvraag.
Door nu BUFEND te initialiseren op RECPOS+1 wordt in één vraag uitgemaakt of we middenin de buffer zitten of niet. Is dit niet zo dan moeten we natuurlijk nog uitmaken of we het eerste of laatste woord vragen.

Tussen DUMP en RECOVER is voorts nog een ander verschil. DUMP vult een consecutief stuk van de trommel en vertelt aan de compiler via ASK DUMP END hoeveel hij gevuld heeft.
RECOVER werkt een start1order keten af, die niet consecutief op de trommel hoeft te staan (Zie hiervoor start1orders in de submonitorsectie).
Bij het aanhaken van nieuwe buffers tellen we DRROOM7 af. Hierdoor wordt een buffer alleen in de QUEU gehangen als hij ook gevuld moet worden. Bij de terugmelding van het transport wordt in de start1order bijgehouden hoeveel buffers er echt van de trommel komen.
Hierdoor weten we of we ACTIVE af mogen zetten of niet.

I-e. Toevoegingen aan de compiler

Toegevoegd zijn:

I-e-1 Drie compilers

I-e-2 INSPECT DECL

I-e-3 TRUNCATE

I-e-4 CRF

I-e-1.

'Drie compilers"

Met behulp van de syntax gegeven in het Revised Report kan men uitmaken of een rij basic symbols een ALGOL 60 programma is of niet.
In het MC-ALGOL 60 systeem is deze taak toegewezen aan de compiler.
Naast ALGOL 60 programma's kent het M64KL systeem ook bibliotheekvulprogramma's.
Een rij symbolen, aangeboden als één van de eerste 3 programma's van de dag, die voldoet aan de syntax hieronder, heeft tot effect dat de daarin vermelde procedure's voor de rest van de dag in de bibliotheek zijn opgenomen.

Wat een bibliotheekvulprogramma is wordt vastgelegd door de volgende syntax:

<bibliotheekvulprogramma> ::= begin <restricted declaration>; end

<restricted declaration> ::= <procedure declaration>|<restricted declaration> <procedure declaration>

Binnen het systeem wordt dan ook nog gemanipuleerd met symbolenrijen van de vorm:

begin <procedure declaration>; end

In zekere zin zou men dus kunnen spreken van 3 compilers. Voor een systematische bespreking is het beter te denken aan een geparametriseerde compiler.
De keuze "Wie van de drie"? wordt bij ingang van de analyse dubbel vastgelegd in CMODE en base.
De 3 mogelijkheden zijn:

CMODE = 1
base = base0

De vertaler analyseert een ALGOL 60 programma. Het resultaat van de vertaling is een bitstring. Voor elke bibliotheekprocedure van de totale behoefte (Zie I-g-4) is een pseudo lvar in de bitstring opgenomen.

CMODE = 2
base = base2

Het resultaat van de vertaling is een bitstring, die opgeslagen wordt op de trommel in de ruimte gereserveerd voor de bibliotheek. De routine CRF neemt in de infotable de primaire behoefte van deze procedure op.

CMODE = 8
base = base1

De vertaler analyseert een bibliotheekvulprogramma. Het enige resultaat van de vertaling is dat de catalogue uitgebreid wordt met de namen van de aangeboden bibliotheekprocedures. Er wordt dus geen bitstring gevormd. De routine CRF doet niets.

Opmerking In het laatste geval is WANTED negatief. WANTED heeft verder dezelfde betekenis als in MICRO48K.

Na afloop van analyse staat CMODE nog steeds op de meegegeven waarde. Dit geldt niet voor base.
De compilemode's verschillen in 2 aspecten:

1. de syntax die herkend wordt. Hiervoor wordt base gebruikt.
2. wat de compiler moet doen, waar text vandaan komt etc. Hiervoor wordt de CMODE gebruikt.

Opmerking De waarde van CMODE wordt bepaald door de SUBMONITOR aan de hand van HMODE. De handling mode, geeft informatie over hoe een bepaald stuk string als job geïnterpreteerd moet worden. Daar de compiler in het geval van bibliotheekprocedures binnen één job op verschillende syntaxen werkt is er een variabele nodig om aan te geven welke syntax geanalyseerd wordt dat is base. Binnen één job kan PRESCAN0 nu ook meerdere malen doorlopen worden. Het text array wordt echter maar één keer gevuld. Vandaar CMODE.

Het effect van de invoering van de variabele base voor gewone ALGOL 60 programma's is bijna niets. In plaats van SUBC(:PROGRAM) in TRANSL wordt nu gedaan S = base gevolgd door SUBC(MS). Uiteraard staat dan in S het adres van PROGRAM.

Iets anders wordt het bij compilemode 2. Met voordeel kan hier gebruik gemaakt worden van het feit dat het programma correct is en bovendien van de vorm

begin <procedure declaration>; end

is. Met deze speciale syntax wordt dan ook in de 4 instructies na PROGRAM3CM2 op blz.271 van [7] afgerekend. Het "base"-mechanisme biedt hier tevens het grote voordeel dat de overhead voor een bibliotheekprocedure beperkt wordt tot de 2 geheugenwoorden voor de pseudo lvar. Dit betekent dat het voor een gebruiker, die uit het geheugen "barst" geen zin heeft om de declaraties van de bibliotheekprocedures in zijn programma op te nemen en dan opnieuw zijn programma aan te bieden.

Bij compilemode 8 moeten we tenvolle het probleem van foute programma's onder ogen zien. We zouden hier dus eigenlijk de opvatting in alle scans hetzelfde moeten hebben. (Zie [5]). Dit is niet gedaan. Dat het systeem goed werkt is dus een "toevalligheid" omdat de verwachtingsstructuur (bij de LOAD and GO compiler alleen de naamlijst) niet relevant veranderd wordt. Waar wordt nu de variabele base bij compilemode 8 gebruikt? wel o.a. bij indirecte aanroep PROGRAM3CM8 op blz. 270 van [7]. Hier is nu eens mooi het principe van dezelfde opvatting te demonstreren. Beter zou nl. geweest zijn een ELAN-sourcetext als hieronder weergegeven.
Stel dat als bibliotheekvulprogramma wordt aangeboden:

library : begin procedure a; ; end .

De label library is nu in PRESCAN0 en PRESCAN1 gezien en opgenomen in de naamlijst (heeft sporen achtergelaten in de verwachtingsstructuur). Bij de ELAN sourcetext van blz.270 wordt hij in de vertaalscan niet gezien. Gelukkig wordt er toch een foutmelding gegeven en zal insertie - laat staan executie-nooit plaats vinden. Nog gelukkiger is het dat in de vertaalscan zelf er nergens op gerekend kan worden dat aan deze label een adres zou moeten zijn toegekend.

```
PROGRAM3CM8:        U,  S - 104 , Z
                    N,  A = 801
                    N,  SUBC(:ERRORM)
PROGRAM3:
PROGRAM:                A = letter last symbol, Z
                    Y,  SUBC(:IDF)
PROGRAM[2]:         Y,  A = last symbol
                    Y,  A - 90 , Z   "last symbol = colon
                    Y,  SUBC(:LABDEC)
                    Y,  GOTO(:PROGRAM)
                        A = digit last symbol, Z
                    Y,  SUBC(:UNS NUM)
                    Y,  SUBC(:IN NM LST)
                    Y,  S = int lab
```

```
Y,  GOTO(:PROGRAM[2])
    S = last symbol
U,  S - 104,  Z   "last symbol = begin?
    SUBC(:NXT SBL)
N,  JUMP(-3)
    SUBC(:DECL LS)
```

en wat daar verder volgt.

I-e-2 INSPECT DECL.

Als de compilemode 8 is moet de catalogue uitgebreid worden. Dit gebeurt binnen de routine INSPECT DECL.
Om de "opvatting" en de "synchronisatie" van herkenning met nxt sbl niet te verstoren wordt ervoor gezorgd dat alleen iets gedaan wordt als er reden voor is (d.w.z. een procedure declarator op display level 1).
Volgt er dan geen identifier dan wordt alsnog INSPECT DECL verlaten.
INSPECT DECL verifieert uiteraard of de naam uniek is en of er ruimte is in het stuk gereserveerd voor de catalogue.
Het is voorts essentieel op te merken dat als INSPECT DECL in de kernen mag schrijven, dezelfde systeemvariabelen uit de verwachtingsstructuur gebruikt worden als dit ook echt gebeurt.
(De S waarde op regel 25 van blz. 268 is afgeleid uit dezelfde bits uit de naamlijst als op regel 52).

I-e-3 TRUNCATE.

In het systeem M64KL kunnen willekeurige ALGOL 60 procedures als bibliotheekprocedures op de trommel worden opgeslagen. Dit kan met behulp van "bibliotheekvulprogramma's (zie def. in I-e-1).
Het is nu geoorloofd dat bibliotheekprocedures binnen één bibliotheekvulprogramma elkaar wederzijds recursief aanroepen.
Om na te gaan dat dit voor wat betreft aantal en soort van de parameters etc. correct gedaan is, wordt het bibliotheekvulprogramma eerst als één geheel geanalyseerd.

Daar elke procedure afzonderlijk relocateerbaar moet zijn (met al zijn constanten etc.) is daarna nog voor elk een aparte analysegang vereist.
Uiteraard is TRUNCATE een routine van PRESCAN0.
TRUNCATE zorgt er nu voor:

1. dat het na het verlaten van TRUNCATE voor de compiler net lijkt alsof een programma van de vorm

   begin <procedure declaration>; end

   is aangeboden.

2. dat, als de bibliotheekprocedure niet de laatste is van het bibliotheek-vulprogramma beginsafe, shiftsafe, wordsafe en evt. word1safe zo gevuld worden dat het na de inloop van het hoofdprogramma van PRESCAN0 bij de volgende analysegang net lijkt alsof het bibliotheekvulprogramma begint met de volgende bibliotheekprocedure.

We hebben dus op 3 niveaux te maken met de manipulatie met text array.

1. voor elke scan in INIT text array pointer wordt gezet op begin of text array
   shift " " " first shift
2. hoofdprogramma PRESCAN0 begin of text array " " " begin safe
   first shift " " " shift safe
3. ILR zorgt eerste keer voor goede waarden beginsafe en shiftsafe
   TRUNCATE zorgt alle andere keren " " " " "

Voor de verdere bespreking van TRUNCATE maken we gebruik van het volgende bibliotheekvulprogramma.

```
begin  procedure  a;
       begin .....
             .....
       end   (;)

       procedure  b;
       begin .....
             .....
             .....
       end   (;)
end
```

TRUNCATE wordt aangeroepen bij het verlaten van de PRESCAN0 routine BLOCK als CMODE=2 en display level weer op 1 gezet wordt.
Als gebruikelijk is bij het verlaten van een syntactisch eenheid het eerste symbool van de naasthogere syntactische eenheid al gelezen.
M.a.w. vlak voor de aanroep van TRUNCATE heeft de aanroep van NXT SBL op regel 13 van blz.185 van [7] in "last symbol" de interne representatie van de semicolon afgeleverd. Bij bovenstaande sourcetext wordt TRUNCATE dus 2* elke keer na het omcirkelde symbool aangeroepen.
Bij aanroep van TRUNCATE weten we dat het programma correct is en dat NXT SBL via NXT BSC SBL en IN SBL direct uit text array pakt.
Hieruit volgt:

1. de te analyseren bibliotheekprocedure is de laatste van het bibliotheekvul programma $\Longleftrightarrow$ symbool afgeleverd door eerste symbool in TRUNCATE= end

2. Na de eerste aanroep van NXT SBL binnen TRUNCATE bepalen text array pointer en shift de plaats van het eerstvolgend symbool na het symbool afgeleverd door de eerste aanroep.

I-e-4.

CRF (Crossreference)

0. Terminologie:

De primaire behoefte aan bibliotheekprocedures van een ALGOL 60 programma bestaat uit die bibliotheekprocedures, waarvan de identifiers in de sourcetext van het genoemde ALGOL 60 programma voorkomen.
De totale behoefte aan bibliotheekprocedures van een ALGOL 60 programma bestaat uit die bibliotheekprocedures, die tijdens runtime nodig zijn om het programma uit te voeren.
We spreken van de compilemode is 8 als CMODE de waarde 8 heeft.

1. Aanroep van CRF heeft voor elke compilemode een ander effect. Is de compilemode 8 dan hoeft de analysator alleen maar te verifieëren of het programma aan de restricte syntax vermeld in I-e-1 voldoet en is de crossreference irrelevant. Vandaar dat er in dit geval niets wezenlijks gebeurd.
Is de compilemode 2 dan wordt er dus echt een bibliotheekprocedure in de drumlibrary opgenomen. We moeten nu de infotable uitbreiden aan de hand van de primaire behoefte.
Is de compilemode 1 dan moet eerst aan de hand van de primaire behoefte de totale behoefte bepaald worden. Dit gebeurt in de recursieve routine CRSS. Is de totale behoefte bekend dan kunnen de bitgroepen voor de pseudo lvars overeenkomstig I-e-3 gemaakt worden. Dit gebeurt in de routine PSEUDO LVAR.

2. De input voor CRF is dus eigenlijk de primaire behoefte (af te lezen uit de crosstable). Om de primaire behoefte te kunnen bepalen moet de crosstable eerst op "0 gezet" worden bij ingang van de analyse. Dit is de taak van de routine CLEAR CRSTAB uit PRESCAN0.
Staat in de sourcetext een aanroep van een bibliotheekprocedure dan zal dat bij analyse tijdens PRESCAN1 tot gevolg hebben dat ASK LIBR wordt aangeroepen.

Door nu in ASK LIBR voor elke bibliotheekprocedure in de corresponderende entry van de crosstable d20 = 1 te maken is aan het begin van de vertaalscan de primaire behoefte bekend. In te zien dat de routine CRSS goed werkt eist inzicht in de structuur van de crosstable en infotable. Dat de routine termineert is duidelijk. Gaan we voor één bp de cross-reference uitwerken dan vragen we of van de corresponderende crosstable entry d21 nul is. Zo nee, dan verlaten we deze incarnatie en zo ja, dan gaan we de body verder in. Het eerste wat dan gedaan wordt is bit d21 op te zetten. Voor één en dezelfde procedure kan deze "sluis" dus niet 2* gepasseerd worden. Daar er maar eindig vele bibliotheekprocedures zijn, termineert deze routine dus onherroepelijk en dus ook de routine NP.

I-f.
Bibliotheek en bibliotheekadministratie.

I-f-1 Bibliotheekadministratie

I-f-2 Gang van zaken bij gebruik van een bibliotheekprocedure

I-f-3 Gang van zaken bij opname van een bibliotheekprocedure (zie I-d-1 ILR).

I-f-1.
de bibliotheek-administratie

De bibliotheekadministratie bestaat uit:

1. de "state of library"
   Dit zijn de 6 variabelen:
   nbr of routines, end of catalogue, end of trafotable,
   end of crosstable, end of infotable, end of drumlibrary

   Deze variabelen worden alleen gewijzigd binnen ILR en geïnitialiseerd in FOUND LIBRARY.
   Wel is geprobeerd een wijziging in de "state" altijd zo laat mogelijk te doen. Zo worden "end of trafotable" en "end of crosstable" pas na de analysegang met compilemode 8 gewijzigd.

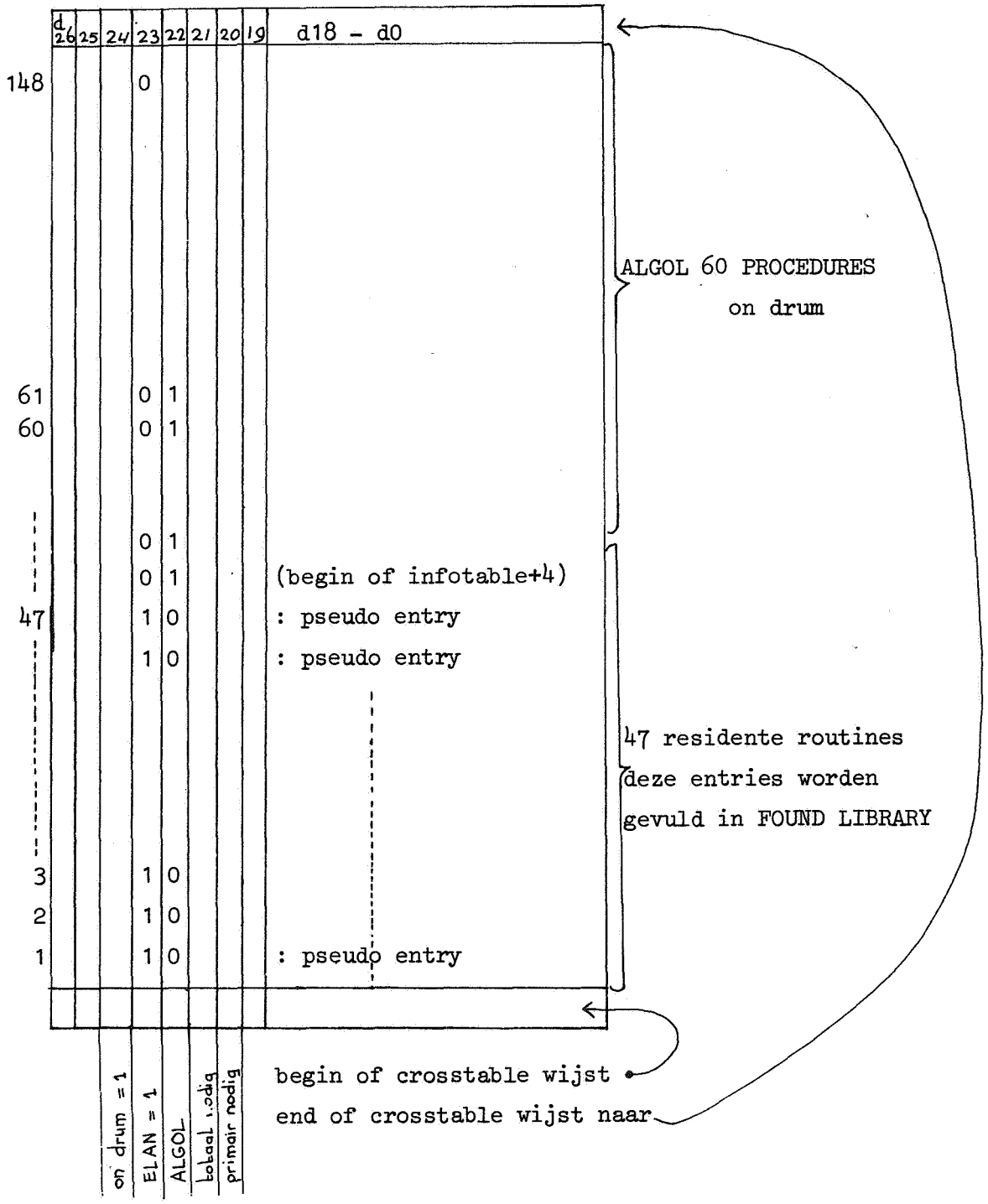
2. de "lvars".
   Het array lvar vormt de schaduw van de "state". Het is als het ware steeds de "old state".
   De lvars worden dan ook niet gewijzigd binnen ILR.
   Voordat ILR aangeroepen wordt zijn "state" en "old state" gelijk.
   Binnen ILR kan dan iets aan de "state" gewijzigd worden.
   Wordt ILR nu op een of andere wijze onderbroken dan wordt er in ENDRUN door aanroep van ASSURE LIBRARY voor gezorgd dat de "state" weer gelijk wordt aan de "old state".
   Wordt ILR echter niet onderbroken dan worden de lvars in INFORM OPERATOR door aanroep vanUPDATE LIBRARY gezet op de waarden van de state.
   In hun functie als "old state" worden de lvars geïnitialiseerd in FOUND LIBRARY. Deze procedure eindigt dan ook met GOTO(:UPDATE LIBRARY)

3. de crosstable.
   Deze bevat voor elke procedure een ingang in de infotable. Voor de residente routines is de crossreference niet belangrijk. Hun entries in de crosstable verwijzen dan ook naar dezelfde in het comment op blz. 13 [7] genoemde pseudo entry.
   Deze entries worden gevuld in FOUND LIBRARY. Hierdoor kan men ook de capaciteit van de bibliotheek zo makkelijk wijzigen. (Zie III-b).

4. de infotable
   Deze bevat voor elke niet-residente routine de lengte in de kernen, trommeladres en primaire behoefte.
   De eerste entry is de zgn. pseudo-entry (zie pt.3).
   De eerste 3 entries worden dan ook met nullen gevuld om aan te geven dat de crossreference irrelevant is.

5. de catalogue.
   Tot hoever de catalogue zinnig gevuld is wordt aangegeven door "end of catalogue".
   Het eerste stuk van de catalogue wordt meegeassembleerd. Echter vanwege de non-LOAD and GO assembler wordt tijdens LOADEN de catalogue ook :DELTA1 plaatsen hoger ingezet.

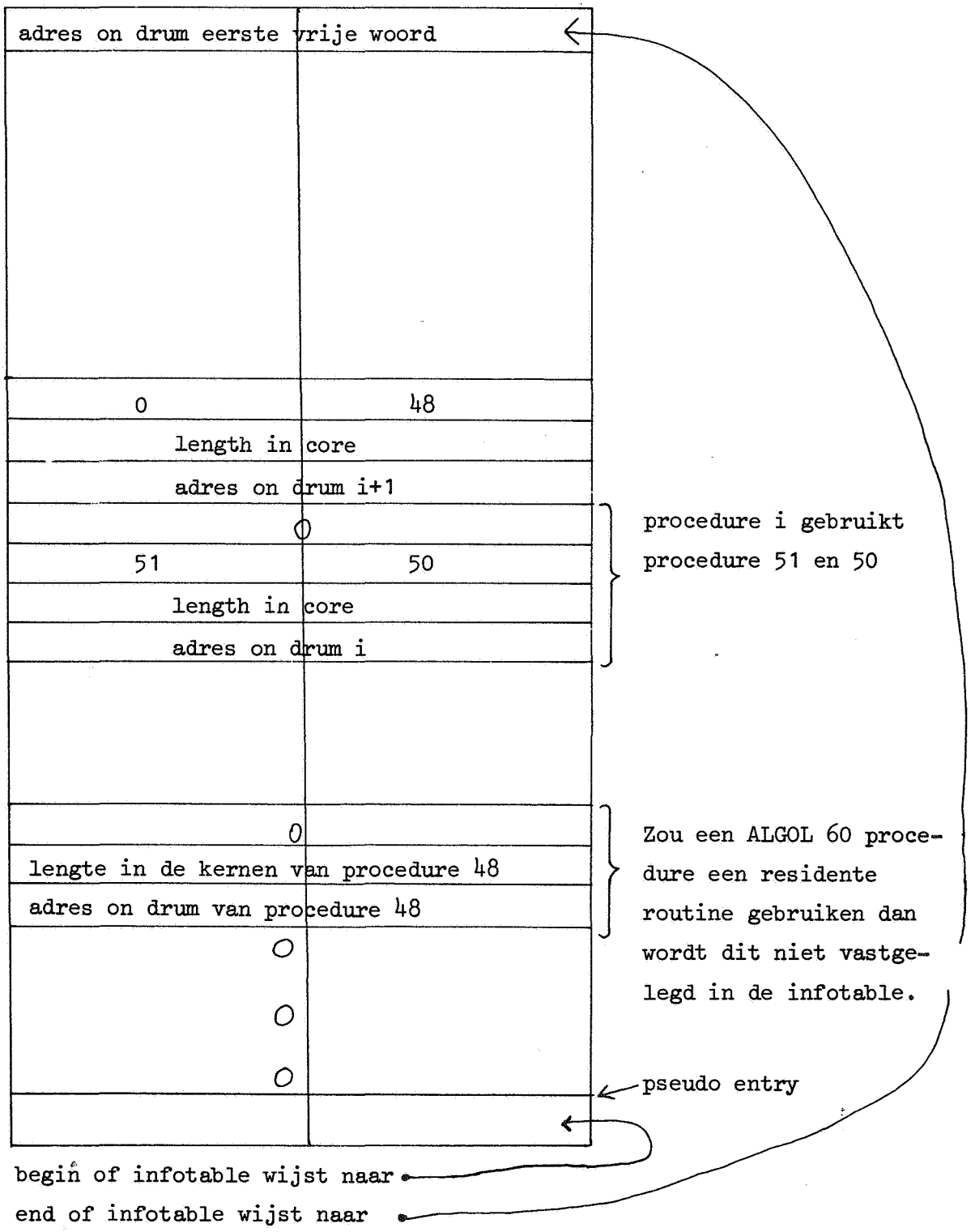
De structuur van de crosstable (als er 148 routines in de bibliotheek zitten).
Opm. d21 en d20 van elke entry horen eigenlijk niet bij de bibliotheek-administratie maar zijn werkvariabelen van de compiler.



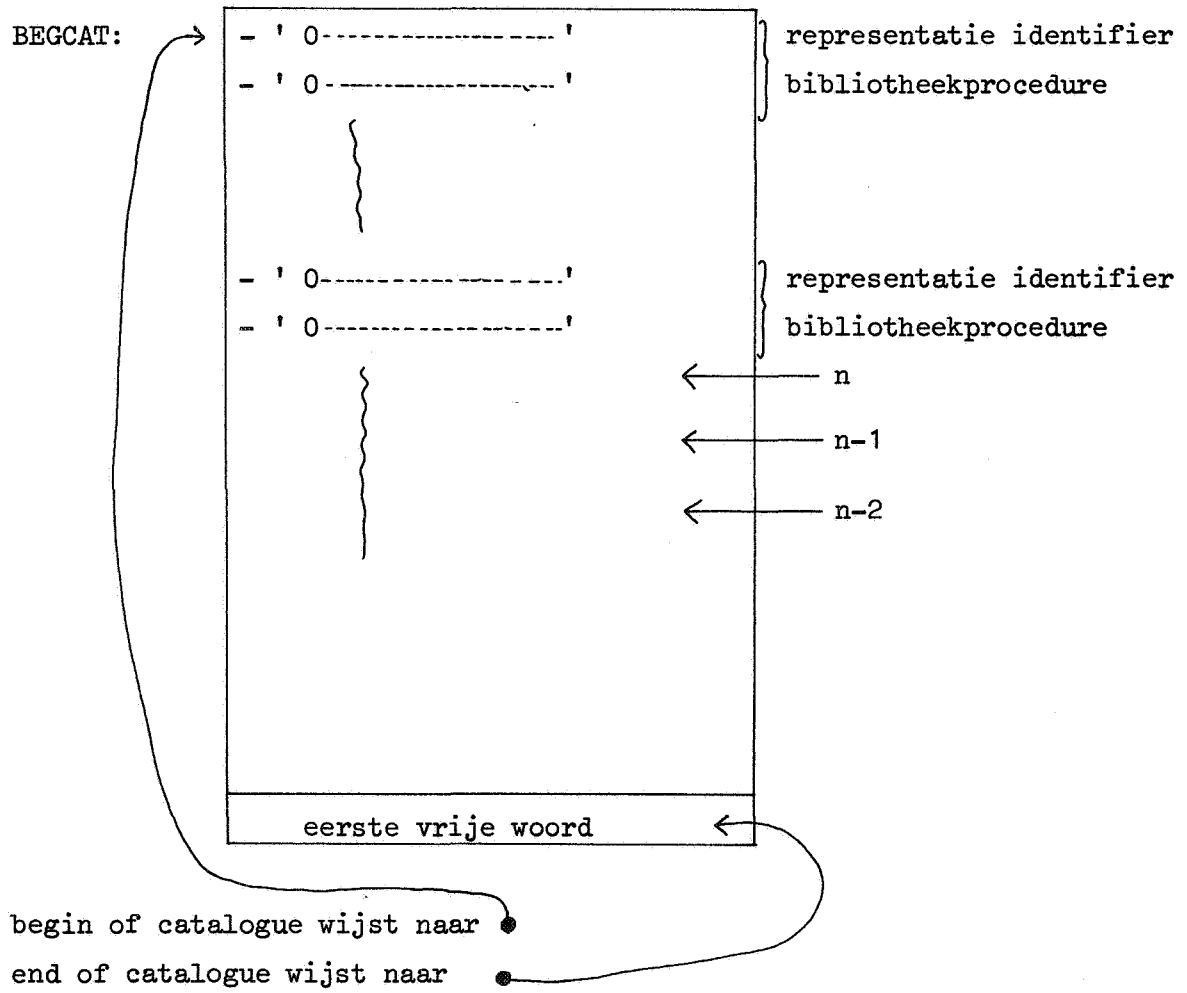
M[end of crosstable] wijst altijd naar end of infotable
M[end of infotable] wijst altijd naar end of drumlibrary

De structuur van de infotable

Aangenomen wordt dat adres on drum van proc i < adres on drum van proc j
als i < j



De structuur van de catalogue



In het woord waar n naar toewijst staat:

voor een gewone procedure : 24 * d19 + rangnr uit de bibliotheek
integer " : 17 * d19 + " " " "
real " : 16 * d19 + " " " "
boolean " : 18 * d19 + " " " "

In PRESCAN1 staat na opzoeken van de procedure identifier in de catalogue in het register S het adres n. De routine ADDRSS levert daarom in A het rangnummer van de bibliotheek procedure af.

Opm. d18 (M[n]) = 0; Dit houdt in dat voor een bibliotheek procedure identifier SUBC(:DYNAMIC) de conditie NO oplevert.

In het woord waar n-1 naar toewijst staat:

bit	stand	betekenis
d26	0	----
d25	1	geeft aan het is een bibliotheekprocedure
d24	0	onderscheide code en ¬code is zinloos
d23	1	operator like
	0	¬operator like
d22	1	outside declaration
d21	0	---
d20	0	---
d19	1	de procedure is gedeclareerd
d18-d0		list length + 1

In het algemeen staat er voor ¬operator like dus op n-1:

'222 000 000' + list length+1

In het woord waar n-2 naar toewijst staat

in het geval van operator like, een entry in de mcrlist.
in het geval van parameterloos een 0.
in het geval van parameters een -1.

Voor elke parameter zijn 3 woorden opgenomen

-1	pseudo letters
char * d19	character
0	afsluiting

Per procedure is als afsluiting van het totaal dan nog één woord met +0 gevuld.

De catalogue wordt naamlijstachtig genoemd. Het is dan ook niet zo dat voor elke procedure het stuk naamlijst, dat gegenereerd wordt, bij eerste vertaling gelijk is aan het stuk dat door INSPECT DECL aan de catalogue wordt toegevoegd. Dit zou ook onzin zijn want de gegevens nodig om de assignment aan de function designator te vertalen zijn bv. alleen van belang bij de precompilatie. Zo hoeft het systeem na precompilatie ook de identifier van de formelen niet meer te kennen.

Hoe passen nu de operator like macro's in dit algemene schema?

Als elke bibliotheekprocedure kunnen ze op 2 manieren aangeroepen worden:

(1) als de procedure in een procedure statement.
De analysator zet de parameters op de stapel en roept vervolgens MACRO2 aan met het macronr van deze macro.
Het par part van deze macro is in MR 84 0.

(11) als actuele parameter in een procedure call; In dit geval wordt de aanroep vertaald met <IJU1> <3> <<n>> en zal de transformator het adres uit de trafotable pakken.
In de trafotable staat het adres van de corresponderende pseudolvar.
Het eerste woord van de pseudolvar zal echter nooit gebruikt worden.

I-f-2.
Gang van zaken bij gebruik van een bibliotheekprocedure

In de sourcetext van een ALGOL 60 programma mag in het M64KL systeem een identifier van een bibliotheekprocedure voorkomen.

Dit kan leiden tot 2 verschillende aanroepen van MACRO2 nl.
MACRO2(IJU1,n) voor een actuele parameter en
MACRO2(ISUBJ,n) voor een procedure call.
In de sourcetext van een bibliotheekvulprogramma mag de procedure identifier ook op vrij willekeurige wijze in de text van een code procedure voorkomen.

We zullen eerst beschrijven hoe het systeem op het gebruik van een bibliotheekprocedure in een correct ALGOL 60 programma reageert.

1. PRESCAN0	de identifiers van bibliotheekprocedures worden geskipt
2. PRESCAN1	een niet in het programma gedeclareerde naam wordt door ASK LIBR in de catalogue opgezocht. Komt de naam in de catalogue voor dan worden de gegevens van die naam uit de catalogue gecopieërd en in de naamlijst opgenomen. In de crosstable wordt geregistreerd dat de bewuste naam primair nodig is.
3. CRF (begin van TRANSL)	uitgaande van de primaire behoefte wordt de totale behoefte bepaald. In de bitstring wordt altijd opgenomen: <257> <<<0>>> de "leading zero". en voor een bibliotheekprocedure met rangnummer n: <261> <<n>> De instr cntr wordt verhoogd met 2* aantal bibliotheekprocedures+1
4. TRANSL	voor de echte aanroep wordt in de bitstring opgenomen: <mcrnr> <3> <<n>>

5. TRANSF

De bitstring wordt door de transformator in dezelfde volgorde in objectcode gezet als ze tijdens analyse ontstaan is. Eerst komen dus de directieven gemaakt in CRF en dan de bitgroepen gemaakt bij echte aanroep. Bij het maken van objectcode wordt van deze wetenschap gebruik gemaakt.
Dus:
<257> <<<0>>> Er wordt één geheugen woord met +0 gevuld. Vervolgens worden voor elke bibliotheekprocedure 2 geheugenwoorden met het rangnummer gevuld en het adres van het eerste woord wordt gezet in de corresponderende entry van de trafotable.
Bij de bitgroepen <mcrnr> <3> <<<n>>> wordt de parameter van de bij mcrnr horende macro nu op entry n van de trafotable gevonden.

6. NPR (na TRANSF)

Vul de pseudo lvar met de adressen waar de procedurebodies terecht gekomen zijn.

Bij een bibliotheekvulprogramma worden 5 en 6 uiteraard niet gedaan. Tijdens CRF wordt aan de hand van de primaire behoefte de infotable uitgebreid en worden geen bitgroepen in de bitstring opgenomen.

I-g.
Enkele belangrijke variabelen

variabele	betekenis
base	zie blz. 32
begin of crosstable	geeft het begin van de crosstable aan. Daar dit vastgelegd wordt in FOUND LIBRARY kan de systeembeheerder de crosstable verplaatsen.
begin of infotable	geeft het begin van de infotable aan. Dit wordt vastgelegd in FOUND LIBRARY. Er geldt hetzelfde als voor begin of crosstable
beginsafe	zie blz. 36
cmode	compile mode zie blz. 32
correctie	inputparameter voor de transformator en geeft aan na welk woord de transformator de objectcode mag neerzetten
dp0	PRESCAN0: dp0 vervult de rol van "constant list pointer" d.w.z. geeft het eerste vrije woord na de constantenlijst aan. PRESCAN1 dp0 fungeert als "end of constant list" TRANSL bij het opzoeken van een constante dp0 TRANSF: geeft het begin van de statische displayvector van het hoofdprogramma aa d.w.z. de startwaarde van D bij executie

end of catalogue end of crosstable . end of drumlibrary : end of infotable end of trafotable	Deze variabelen vormen samen met "nbr of routines" de werkvariabelen van het systeem bij het opnemen van een bibliotheekprocedure. (zie blz. 39). Hun functie ligt in hun naam besloten. end of catalogue wordt verlaagt in INSPECT DECL; end of infotable wordt verhoogt in FILL INFOTABLE; end of crosstable en end of infotable worden verhoogt in ILR na de analyse van het bibliotheekvulprogramma; end of drumlibrary wordt verhoogt in ILR na opname van een bibliotheekprocedure op de trommel
end of text array	geeft aan tot hoever text array gegroeid is
first shift	zie I-e-3
first 3 times	bepaalt "hmode" first 3 times wordt geïnitialiseerd in FOUND LIBRARY op -3. Voor elke doorloop van ILR wordt hij met 1 verhoogt.
hepcnt	outputparameter van SIGNOFF. Er wordt alleen naar gekeken als SIGNON ook doorlopen is d.w.z. na een executie
hmode	handlingmode zie blz. 21
instr cntr	tijdens analyse relatief tijdens transf absoluut in ENDRUN na een executie end of program na een vertaling length of program
label	zie blz. 25

last action	= 0 ⟷ laatste actie een executie
lvar	zie blz. 40
pntr	tijdens MAKE START1ORDERS wordt hij geïnitialiseerd op begin of program en dan steeds met 2 verlaagd tot op (pntr+1) de leading zero van de pseudo lvars is gevonden. Daarna wordt pntr gebruikt om stuk voor stuk de bibliotheekprocedures te transformeren
reladrss	hulpvariabele van de compiler in PSEUDO LVAR. Er wordt in uitgerekend hoeveel woorden de bibliotheekprocedures aan het objectprogramma zullen toevoegen.
runnumber	1000 nog niets gedaan 100 prescan0 200 prescan1 300 transl 500 executie In ENDRUN wordt runnumber geraadpleegd om te beslissen of er iets gedaan is. Daarom is na SUBC(:REHEP) in NAIGA de instructie runnumber = B opgenomen om aan te geven dat TRANSL als het ware al begonnen is
shift	zie I-e-3

De consequenties van het in bedrijf nemen van MICRO64KL.

II-a Consequenties voor de gebruikers
II-b Behandeling bibliotheekvulprogramma's
II-c Geheugengebruik
II-d Consequenties voor de operateur

II-a Consequenties voor de gebruikers.

De consequenties voor de gebruikers volgen uit:
"Aanvullingen II op MR81".
Deze aanvullingen zijn hieronder opgenomen.
Voor een toelichting op het geheugengebruik zie blz. 54
Tevens zijn code procedures nu echt verboden.
Het objectprogramma dat een procedure uit de bibliotheek gebruikt is maar 2 geheugenwoorden langer dan het objectprogramma verkregen door die ALGOL 60 procedure zelf in de sourcetext op te nemen.
(Hierbij is er uiteraard van uitgegaan dat het regelnr in beide gevallen tijdens executie niet wordt bijgehouden).
Het zelf in de sourcetext opnemen van procedures kan dus alleen zin hebben bij het "tracen" van foute programma's.

Aanvullingen II op MR81

Met ingang van 1 maart 1970 is het MC-ALGOL 60-systeem voorzien van een gemakkelijk uit te breiden bibliotheek van ALGOL 60 procedures, welke zonder declaratie in een ALGOL 60 programma aangeroepen kunnen worden.
In deze bibliotheek zijn dan, behalve de procedures van sectie 7 en de procedures van de eerste aanvullingen, opgenomen:

1 De procedures beschreven in:

T.J. Dekker, ALGOL 60 procedures in numerical algebra
part 1, MC Tracts 22
Mathematisch Centrum Amsterdam 1968.

T.J. Dekker, W. Hoffmann,
ALGOL 60 procedures in numerical algebra
part 2, MC Tracts 23
Mathematisch Centrum Amsterdam 1968.

2 De procedure available, waarvan de heading luidt:
integer procedure available;
Deze levert het aantal nog beschikbare geheugenwoorden op het moment van aanroep.
Men zij zeer voorzichtig in het gebruik van available in verband met:

a) Een ALGOL 60 programma heeft voor zijn uitvoering "leefruimte" nodig; dit is meer naarmate de expressies ingewikkelder zijn, dit kan zeer groot zijn wanneer procedures recursief worden aangeroepen. In vele gevallen is de reservering van 100 geheugen woorden voor die "leefruimte" voldoende zodat men de rest van het geheugen kan gebruiken voor het declareren van arrays.

b) Het huidige bedrijfssysteem is niet definitief. Het resultaat van het volgende programma:
begin print(available) end
behoeft daarom niet elke dag hetzelfde te zijn.
Er wordt echter naar gestreefd om op zijn minst ca 35 000 woorden ter beschikking te stellen voor het objectprogramma en zijn werkruimte. Men zie voorts sectie 6.1 waar het getal 18 000 vervangen moet worden door 35 000. De mededeling betreffende "zonder protectie" in sectie 6.2 is vervallen.

Treedt tijdens executie binnen een bibliotheek procedure een fout op dan wordt in de foutmelding (zie 4.2.3 en 4.2.4) het eventuele regelnummer vermeld van de "laatst aangevangen statement" uit de programma tekst.

Nieuwe foutmeldingen zijn:

493 het programma is na toevoeging van bibliotheek procedures te lang voor het beschikbare geheugen. In plaats van het laatst ingevoerde getal wordt de lengte van het totale objectprogramma afgedrukt

495 het programma is te lang

II-b.
Behandeling bibliotheekvulprogramma's

De foutmeldingen, die kunnen optreden bij behandeling van bibliotheekvulprogramma's zijn:

801 bibliotheekvulprogramma begint niet met begin

802 bibliotheekvulprogramma is een compound statement

803 het compound tail van een bibliotheekvulprogramma bestaat niet alleen uit end

804 bibliotheekvulprogramma bevat een declarator, die geen procedure declareert op display level 1

805 de naam van een aangeboden bibliotheekprocedure komt al voor in de bibliotheek

806 capaciteit infotable overschreden
(Zie III-a)

807 capaciteit crosstable en trafotable overschreden (Zie III-a)

808 vervallen

809 capaciteit catalogue overschreden (Zie III-a)

495 capaciteit trommelruimte uitgeput.

Het laatste vel regeldrukker-output bevat geen schatting over het aantal woorden in beslag genomen door het objectprogramma.

Geheugengengebruik.

GEHEUGEN VERDELING MKLL

bij lopend systeem

	'260'
INGREEPSTAPEL	'400'
constanten geheugenverdeling, drumbudget	'1000'
achtereenvolgens: monitor, I/O conv, drum section, lezer section, ponser section, regeldrukker section, klein stuk van RECODE section	'14206'
De Q variabelen uit MICA	'14236'
	'14251'
MICA (overschrijft FREEWO)	'14640'
	'14745'

achtereenvolgens submonitor, system,
transformator part below 32K

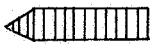
'15425' FREEWO1

Ip SHORTEN TRAJECT

'15430'

Ip MKL(L) → VUMKL(L)

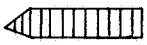
'15446'

'15503' 

RECOVER

RUNVAR

'16316' FREEWO2

'16405' 

achtereenvolgens RUN syst, biblio-
theekbodies

'21320'

DUMP

COMPILER

catalogue space	SYSTAP permanent catalogue

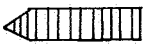
objectprog.

'116551'

transformator part up 32K

achtereenvolgens : macro list, instruct
list, trafotab, crosstab, infotab.

'126111' = M[MEMORYUSABLETO]

Opmerking een inzetaanwijzing is aangegeven door 

In MICRO48K wordt het geheugen gebruikt door systeem + MICA tot adres
'130366'

Voor objectcode + naamlijst + text array staan '45163' woorden ter beschikking.
Voor objectprogramma in executie staan '104117' woorden ter beschikking

Opm. We zien hier even af van de buffers uit de RECODE section.
Dit doen we ook bij M64KL

In M64KL wordt het geheugen gebruikt door systeem + MICA tot adres
'126111'

Voor naamlijst + text array + constant list staan '54547' woorden ter beschikking.
Voor objectprogramma in executie staan '104571' woorden ter beschikking.

Toelichting De vergelijking is niet helemaal eerlijk omdat MICA [9] in MICRO48K niet zo laag mogelijk is ingezet.

Wel is het juist dat M64KL, zoals het op 2/4/70 geassembleerd is, aan 3 kasten geheugenruimte genoeg heeft.

II-d.

Consequenties voor de operateur

In plaats van het M64KL systeem, wordt als bedrijfssysteem gebruikt het MKLL systeem (Zie ook IV-b).

Wil men echter het M64KL systeem gebruiken dan kan men dit eenvoudig verkrijgen door na inlezen van het MKLL systeem en voor de inslag van DATE op geheugenadres '01607' weer SUBCD(: FOUND LIBRARY) te zetten.
Bij aanbod van een bibliotheekvulprogramma met SUM, INPROD etc. gevolgd door 2* het flopprogramma (begin integer i; L: gotoL end) krijgt men het volgende verslag over de COTEL.

```
SYSTEEM MKLL
DATE
SERIAL
INPUT TAPE
LIBRARY
E
  41015  WORDS ON DRUM
LOAD      2
INPUT TAPE
LIBRARY
    804
    803
E
LOAD      1
INPUT TAPE
LIBRARY
    804
    803
E
LOAD      1
INPUT TAPE
```

III Aanwijzingen voor uitbreiding MICRO64KL

III-a

Het opnemen van een residente routine.

In M64KL zijn 47 residente bibliotheekprocedures opgenomen. Dit houdt in dat hun bodies voortdurend in de kernen aanwezig zijn. Ze zijn alle in ELAN gecodeerd en tegelijk met het systeem geassembleerd.
Het wordt in het algemeen afgeraden dit aantal uit te breiden.
Er zijn 2 gevallen waarin men zonder meer een nieuwe routine resident zal maken.

1. Bij toevoeging van I/O routines van nieuwe randapparaten.
   (Deze routines zal het systeem zelf waarschijnlijk ook gebruiken).
2. Toevoeging van een hele snelle routine, waarvan men de instructies via het macrolijst mechanisme in de objectcode wil insereren.
   (zoals abs, sin en cos).

Voor één routine met b.v. rangnummer 48 moeten de volgende wijzigingen worden aangebracht:

a wijzig "nbr of permanent routines" van +47 in +48.

b wijzig de catalogue (Zie aldaar letters negatief)

c verlaag "permanent catalogue end" met het vereiste aantal woorden.

d voeg een residente pseudo lvar toe (< 32K)

e vul trafotable [48] met het adres van de pseudolvar.

Toelichting:
Wordt de identifier van een bibliotheekprocedure gebruikt als actuele parameter dan zal de vertaling anders luiden dan bij gewoon gebruik.
Een pseudo lvar is een geheugentraject van 2 woorden. Dit traject mag overal in het geheugen staan. In het eerste woord staat het adres van de ingang bij gewoon gebruik. In het tweede woord staat het adres bij gebruik als actuele parameter.

Het spreekt vanzelf dat wijziging b in geval 1 afwijkt van geval 2.
Bovendien zal in geval 2 het eerste woord van de pseudo lvar nooit gebruikt worden (Zie I-f-1).

III-b.
Uitbreiding van de maximumcapaciteit van de bibliotheek.

Het systeem is beveiligd tegen een te groot aanbod van bibliotheekprocedures. Hieronder volgt een lijst van foutmeldingen, die kunnen optreden en welke geheugenplaatsen men mag wijzigen om de capaciteit van de bibliotheek te vergroten.
Voor foutnr 807 en 806 moet men dit bij stilstaande machine doen en daarna LS NB geven. (Hierna wordt de bibliotheek opnieuw ingelezen). Hoe men dit in een meer definitieve wijziging kan vastleggen is evident.
Voor foutnr 809 is aangegeven een wijziging met nieuwe assemblage.

Opmerking De actie op een foutmelding kan uiteraard weer een andere foutmelding veroorzaken.

foutnr	807
betekenis	aantal aangeboden bibliotheekprocedures is te groot
mogelijke wens van de systeem-beheerder	maximale aantal te vergroten tot p
te nemen actie	a controleer dat geldt: 2p+3 ≤ INFOTABEND - TRAFOTABBEG b maak max nbr of routines gelijk aan (1+p) c geef LS NB

Opm. de variabele max nbr of routines is dus 1 groter dan het aantal procedures dat opgenomen kan worden.

foutnr	806
betekenis	capaciteit van de infotable overschreden
mogelijke wens van de systeembeheerder	deze capaciteit te vergroten met 128 woorden
te nemen actie	a verhoog INFO TAB END met 128 b verhoog LENGTH OF UP 32K met 128 c verhoog DR LIBR TAB END met 128 d verhoog DR BEG8 met 128 e verifieër of nu nog steeds geldt: INFO TAB END < MEMORY USABLE TO DR BEG8 < DREND8 f geef LS NB

foutnr	809
betekenis	capaciteit van de catalogue overschreden
mogelijke wens van de systeembeheerder	deze capaciteit te vergroten met p woorden.
te nemen actie	Assembleer opnieuw met i.p.v. de inzetaanwijzing M['4000']: DELTA1: de inzetaanwijzing: M[('4000'+p)]: DELTA1:

Opm. men kan nieuwe assemblage als volgt vermijden:

1) lees systeemband in
2) maak een ipcorrectie die alle constanten op blz.1 en 2 van [7] DELTA1 in expressie hebben staan met hetzelfde getal verhoogt
3) LS; BGA op : SYSTAP
   en we hebben een nieuwe systeemband

Wel moet natuurlijk nog steeds gelden DR LIBR BEG < DR LIBR END

Opm.2 Een analoge truc voor DELTA geeft meer werk daar in de transformator DELTA in veel expressies voorkomt.

IV-a Overzicht macrowoorden.

Het volgende programma heeft twee inputbanden. Op de eerste staan alle namen van MACRO's. Komen hierin ook cijfers voor dan worden die bij het printen niet afgedrukt. Waar nodig zijn ze bijgeschreven. Op de tweede band staat de ELAN-tekst van de MACROLIST, zoals het systeem geassembleerd is. In de inzetaanwijzing MCR LST [90]: zijn echter eerst met de hand heptaden corresponderend met resp. "9" en "0" omgezet in "erase". De betekenis van de outputkolommen is als volgt:

kolom 2 : macronaam
3 : macrowoord
4 : B-reactie
5 : entry
6 : par-part
7 : instr. nbr
8 : opt. nbr
9 : kind
10 : opt. op.
11 : sat

Voor notatie en betekenis zie de verslagen van de bespreking MR 84.

```
begin comment R 1352, K.K. Koksma MC(tet87);

    integer procedure bitstring(kn, n, codewoord);
    integer kn, n, codewoord;
    begin integer k;
        k:= codewoord : kn;
        bitstring:= (codewoord - k × kn) : n
    end bitstring;

    integer procedure Breaction(macro); integer macro;
    Breaction:= macro : 2097152;

    procedure A(p, q);
    ABSFIXT(5, 0, bitstring(p, q, macro));

    integer array entrynamel[0:203];
    integer array namel[1:203 × 15];

    integer procedure octal number;
    begin integer intermediate;
        intermediate:= 0;
        for symbol:= RESYM while symbol ≠ apostrophe do if
        symbol < 7 then intermediate:= intermediate × 8 +
        symbol; octal number:= intermediate
    end;

    integer procedure unsigned number;
    begin integer intermediate;
        intermediate:= symbol;
        for symbol:= RESYM while symbol ≠ nlcr ∧ symbol ≠
        semicolon do if symbol < 9 then intermediate:=
        intermediate × 10 + symbol;
        unsigned number:= intermediate
    end;

    integer procedure next macro;
    begin
    next: symbol:= RESYM;
        if symbol = apostrophe then next macro:= octal
        number else if symbol > 0 ∧ symbol < 9 then next
        macro:= unsigned number else goto next
    end;
```

```
    Boolean procedure letter;
    begin letter:= symbol < 62 ∧ symbol > 10;
        if symbol ≥ 36 then symbol:= symbol - 27
    end;

    integer group, line, mcrnr, pos, entry, macro,
    apostrophe, nlcr, symbol, semicolon;
    Boolean first;

    Boolean procedure separator(s);
    separator:= s = 87 ∨ s = 93 ∨ s = 118 ∨ s = 119;

    apostrophe:= 120; nlcr:= 119; semicolon:= 91; mcrnr:= 0;
    entry:= 1; first:= true;
    for symbol:= RESYM while symbol ≠ 76 do if
    separator(symbol) ∧ first then
    begin first:= false; namel[entry]:= - 1;
        entry:= entry + 1; entry namel[mcrnr]:= entry;
        mcrnr:= mcrnr + 1
    end
    else if letter then
    begin first:= true; namel[entry]:= symbol;
        entry:= entry + 1
    end;
    namel[entry]:= - 1; mcrnr:= 0;
Nxt page: for group:= 1 step 1 until 8 do
    begin for line:= 1 step 1 until 5 do
        begin pos:= 0; ABSFIXT(3, 0, mcrnr);
            for entry:= entrynamel[mcrnr], entry + 1 while
            namel[entry] ≠ - 1 do
            begin PRSYM(namel[entry]); pos:= pos + 1 end;
            SPACE(15 - pos); macro:= next macro;
            ABSFIXT(12, 0, macro);
            ABSFIXT(5, 0, Breaction(macro));
            A(2097152, 4096); A(4096, 1024); A(1024, 256);
            A(256, 16); A(16, 4); A(4, 2); A(2, 1);
            mcrnr:= mcrnr + 1; if mcrnr = 203 then EXIT;
            NLCR;
        end;
        NLCR
    end;
    NEWPAGE; goto Nxt page;
end
```

0	stack	12583168	6	0	0	1	0	0	0	0
1	neg	8401168	4	3	0	1	1	0	0	0
2	add	4210978	2	4	0	1	2	0	1	0
3	sub	4207154	2	3	0	2	3	0	1	0
4	mul	4215106	2	5	0	1	4	0	1	0
5	div	4195154	2	0	0	3	5	0	1	0
6	idi	4219136	2	6	0	1	0	0	0	0
7	ttp	4223232	2	7	0	1	0	0	0	0
8	equ	4227426	2	8	0	1	6	0	1	0
9	uqu	4227698	2	8	0	2	7	0	1	0
10	les	4235906	2	10	0	2	8	0	1	0
11	mst	4244114	2	12	0	2	9	0	1	0
12	mor	4244386	2	12	0	3	10	0	1	0
13	lst	4256434	2	15	0	2	11	0	1	0
14	stab	10555904	5	17	0	2	0	0	0	0
15	non	8425728	4	9	0	1	0	0	0	0
16	qvl	6369792	3	19	0	2	0	0	0	0
17	imp	6377984	3	21	0	2	0	0	0	0
18	or	6386176	3	23	0	2	0	0	0	0
19	and	6394368	3	25	0	2	0	0	0	0
20	staa	11334144	5	207	0	2	0	0	0	0
21	tsr	23183872	11	28	0	2	0	0	0	0
22	tsi	23192064	11	30	0	2	0	0	0	0
23	tsb	23200256	11	32	0	2	0	0	0	0
24	tsst	23208448	11	34	0	2	0	0	0	0

25	tfsu	23216384	11	36	0	1	0	0	0	0
26	tsl	8540416	4	37	0	1	0	0	0	0
27	tfsl	8544512	4	38	0	1	0	0	0	0
28	tcst	8548608	4	39	0	1	0	0	0	0
29	stsr	21135616	10	40	0	1	0	0	0	0
30	stsi	21139712	10	41	0	1	0	0	0	0
31	sstsi	21143808	10	42	0	1	0	0	0	0
32	stsb	21147904	10	43	0	1	0	0	0	0
33	stsst	21152000	10	44	0	1	0	0	0	0
34	stfsu	21156096	10	45	0	1	0	0	0	0
35	entris	25354496	12	46	0	1	0	0	0	0
36	tfd	8581376	4	47	0	1	0	0	0	0
37	sas	8585472	4	48	0	1	0	0	0	0
38	decs	8589568	4	49	0	1	0	0	0	0
39	fad	8593664	4	50	0	1	0	0	0	0
40	tasr	8597760	4	51	0	1	0	0	0	0
41	tasi	8601856	4	52	0	1	0	0	0	0
42	tasb	8605952	4	53	0	1	0	0	0	0
43	tasst	8610048	4	54	0	1	0	0	0	0
44	tasu	8614144	4	55	0	1	0	0	0	0
45	exitis	27492608	13	56	0	1	0	0	0	0
46	fadcv	8622336	4	57	0	1	0	0	0	0
47	trscv	8626432	4	58	0	1	0	0	0	0
48	tiscv	8630528	4	59	0	1	0	0	0	0
49	tscvu	8634624	4	60	0	1	0	0	0	0

50	exit	8638720	4	61	0	1	0	0	0	0
51	test1	4227328	2	8	0	1	0	0	0	0
52	test2	8643072	4	62	0	2	0	0	0	0
53	crv	12845312	6	64	0	1	0	0	0	0
54	civ	10752256	5	65	0	1	0	0	0	0
55	cbv	10756352	5	66	0	1	0	0	0	0
56	cstv	12857600	6	67	0	1	0	0	0	0
57	clv	12861696	6	68	0	1	0	0	0	0
58	cen	12865792	6	69	0	1	0	0	0	0
59	clpn	17064192	8	70	0	1	0	0	0	0
60	tav	8679680	4	71	0	1	0	0	0	0
61	tiav	8683776	4	72	0	1	0	0	0	0
62	rad	21270784	10	73	0	1	0	0	0	0
63	iad	21274880	10	74	0	1	0	0	0	0
64	bad	21278976	10	75	0	1	0	0	0	0
65	stad	21283072	10	76	0	1	0	0	0	0
66	orad	21287168	10	77	0	1	0	0	0	0
67	oiad	21291264	10	78	0	1	0	0	0	0
68	obad	21295360	10	79	0	1	0	0	0	0
69	ostad	21299456	10	80	0	1	0	0	0	0
70	los	8720640	4	81	0	1	0	0	0	0
71	exitp	8724736	4	82	0	1	0	0	0	0
72	exitpc	8728832	4	83	0	1	0	0	0	0
73	rejst	8732928	4	84	0	1	0	0	0	0
74	jua	8737024	4	85	0	1	0	0	0	0

75	empty	8388608	4	0	0	0	0	0	0	0
76	abs	8741376	4	86	0	2	0	0	0	0
77	sign	8749824	4	88	0	3	0	0	0	0
78	entier	8761600	4	91	0	1	0	0	0	0
79	sqrt	8765696	4	92	0	1	0	0	0	0
80	exp	8769792	4	93	0	1	0	0	0	0
81	ln	8773888	4	94	0	1	0	0	0	0
82	end	8777984	4	95	0	1	0	0	0	0
83	trv	8783105	4	96	1	1	0	0	0	1
84	tiv	8787217	4	97	1	1	1	0	0	1
85	trc	8783149	4	96	1	1	2	3	0	1
86	tic	8787261	4	97	1	1	3	3	0	1
87	tsic	8791369	4	98	1	1	4	2	0	1
88	tbv	8963328	4	140	1	1	0	0	0	0
89	tbc	8967432	4	141	1	1	0	2	0	0
90	tstv	8971520	4	142	1	1	0	0	0	0
91	tlv	8975616	4	143	1	1	0	0	0	0
92	tak	8971520	4	142	1	1	0	0	0	0
93	tswe	8975620	4	143	1	1	0	1	0	0
94	str	8983808	4	145	1	1	0	0	0	0
95	sti	8990464	4	146	3	3	0	0	0	0
96	ssti	8996096	4	148	1	1	0	0	0	0
97	stb	9001472	4	149	2	2	0	0	0	0
98	stst	8791552	4	98	1	2	0	0	0	0
99	dos	9008384	4	151	1	1	0	0	0	0

100	dos2	9012480	4	152	1	1	0	0	0	0
101	dos3	9016576	4	153	1	1	0	0	0	0
102	ju	9020676	4	154	1	1	0	1	0	0
103	ju/	8975876	4	143	1	2	0	1	0	0
104	iju	9024768	4	155	1	1	0	0	0	0
105	iju/	9028864	4	156	1	1	0	0	0	0
106	coju	9032968	4	157	1	1	0	2	0	0
107	ycoju	9037060	4	158	1	1	0	1	0	0
108	subj	9041156	4	159	1	1	0	1	0	0
109	isubj	9045248	4	160	1	1	0	0	0	0
110	decb	9049352	4	161	1	1	0	2	0	0
111	do	9053452	4	162	1	1	0	3	0	0
112	tbl	9058824	4	163	2	2	0	2	0	0
113	entrb	9065992	4	165	1	2	0	2	0	0
114	dptr	30046984	14	167	2	3	0	2	0	0
115	incrb	9065736	4	165	1	1	0	2	0	0
116	tdl	9086216	4	170	1	1	0	2	0	0
117	entrpb	32159240	15	171	1	2	0	2	0	0
118	nil	9098500	4	173	1	1	0	1	0	0
119	last	9102596	4	174	1	1	0	1	0	0
120	lad	9106952	4	175	1	2	0	2	0	0
121	tda	9106696	4	175	1	1	0	2	0	0
122	tna	8791304	4	98	1	1	0	2	0	0
123	taa	8975616	4	143	1	1	0	0	0	0
124	swp	9114888	4	177	1	1	0	2	0	0

125	exitb	9225992	4	204	1	3	0	2	0	0
126	exitc	9119240	4	178	1	2	0	2	0	0
127	exitsv	28001800	13	180	1	2	0	2	0	0
128	code	9098504	4	173	1	1	0	2	0	0
129	sin	9134336	4	182	0	1	0	0	0	0
130	cos	9138432	4	183	0	1	0	0	0	0
131	arctan	9142528	4	184	0	1	0	0	0	0
132	read	9146624	4	185	0	1	0	0	0	0
133	fixp	762112	0	186	0	1	0	0	0	0
134	absfixp	766208	0	187	0	1	0	0	0	0
135	flop	770304	0	188	0	1	0	0	0	0
136	punch	9163008	4	189	0	1	0	0	0	0
137	punlcr	9167104	4	190	0	1	0	0	0	0
138	puspace	9171200	4	191	0	1	0	0	0	0
139	runout	9175296	4	192	0	1	0	0	0	0
140	rehep	9179392	4	193	0	1	0	0	0	0
141	puhep	9183488	4	194	0	1	0	0	0	0
142	hand	9187584	4	195	0	1	0	0	0	0
143	xeen	9191680	4	196	0	1	0	0	0	0
144	stop	9195776	4	197	0	1	0	0	0	0
145	slnc	9202176	4	198	2	2	0	0	0	0
146	rlnc	9209344	4	200	1	2	0	0	0	0
147	lnc	9217544	4	202	1	2	0	2	0	0
148	tnrv	8799488	4	100	1	1	0	0	0	0
149	tniv	8803584	4	101	1	1	0	0	0	0

150	tnrc	8799500	4	100	1	1	0	3	0	0
151	tnic	8803596	4	101	1	1	0	3	0	0
152	tnsic	8807688	4	102	1	1	0	2	0	0
153	addrv	8811776	4	103	1	1	0	0	0	0
154	addiv	8815872	4	104	1	1	0	0	0	0
155	addrc	8811788	4	103	1	1	0	3	0	0
156	addic	8815884	4	104	1	1	0	3	0	0
157	addsic	8819976	4	105	1	1	0	2	0	0
158	subrv	8824064	4	106	1	1	0	0	0	0
159	subiv	8828160	4	107	1	1	0	0	0	0
160	subrc	8824076	4	106	1	1	0	3	0	0
161	subic	8828172	4	107	1	1	0	3	0	0
162	subsic	8832264	4	108	1	1	0	2	0	0
163	mulrv	8836352	4	109	1	1	0	0	0	0
164	muliv	8840448	4	110	1	1	0	0	0	0
165	mulrc	8836364	4	109	1	1	0	3	0	0
166	mulic	8840460	4	110	1	1	0	3	0	0
167	mulsic	8844552	4	111	1	1	0	2	0	0
168	divrv	8848640	4	112	1	1	0	0	0	0
169	diviv	8852736	4	113	1	1	0	0	0	0
170	divrc	8848652	4	112	1	1	0	3	0	0
171	divic	8852748	4	113	1	1	0	3	0	0
172	divsic	8856840	4	114	1	1	0	2	0	0
173	equrv	8860928	4	115	1	1	0	0	0	0
174	equiv	8869120	4	117	1	1	0	0	0	0

175 equrc	8860940	4	115	1	1	0	3	0	0
176 equic	8869132	4	117	1	1	0	3	0	0
177 equsic	8877320	4	119	1	1	0	2	0	0
178 uqurv	8861184	4	115	1	2	0	0	0	0
179 uquiv	8869376	4	117	1	2	0	0	0	0
180 uqurc	8861196	4	115	1	2	0	3	0	0
181 uquic	8869388	4	117	1	2	0	3	0	0
182 uqusic	8877576	4	119	1	2	0	2	0	0
183 lesrv	8886016	4	121	1	3	0	0	0	0
184 lesiv	8898304	4	124	1	3	0	0	0	0
185 lesrc	8886028	4	121	1	3	0	3	0	0
186 lesic	8898316	4	124	1	3	0	3	0	0
187 lessic	8910600	4	127	1	3	0	2	0	0
188 mstrv	8922624	4	130	1	2	0	0	0	0
189 mstiv	8930816	4	132	1	2	0	0	0	0
190 mstrc	8922636	4	130	1	2	0	3	0	0
191 mstic	8930828	4	132	1	2	0	3	0	0
192 mstsic	8939016	4	134	1	2	0	2	0	0
193 morrv	8947200	4	136	1	2	0	0	0	0
194 moriv	8955392	4	138	1	2	0	0	0	0
195 morrc	8946956	4	136	1	1	0	3	0	0
196 moric	8955148	4	138	1	1	0	3	0	0
197 morsic	8910088	4	127	1	1	0	2	0	0
198 lstrv	8885760	4	121	1	2	0	0	0	0
199 lstiv	8898048	4	124	1	2	0	0	0	0

200 lstrc	8885772	4	121	1	2	0	3	0	0
201 lstic	8898060	4	124	1	2	0	3	0	0
202 lstsic	8910344	4	127	1	2	0	2	0	0

IV-b.
PROTOCOL "AANMAAK MKLL"

Op het MC is een programma om bedrijfssystemen op magneetband te zetten. Dit is beschreven in [6].

Vrij spoedig na in gebruikname van MKL was een meer rigide vorm van het bibliotheeksysteem gewenst.
Met behulp van [6] en een kleine Ipcorrectie kan men, als men onderstaand PROTOCOL volgt, het systeem MKLL maken.
MKLL is het systeem corresponderend met de
Aanvullingen II op MR81 (zie II-a).
Leest men het systeem MKLL van magneetband dan kan na intikken van "date" en "serial" met de verwerking van ALGOL 60 programma's begonnen worden. Aan de bibliotheek kan men dan ook niets meer veranderen. Wil men een nieuwe bibliotheekvulling maken dan zal men het PROTOCOL opnieuw (wellicht enigszins aangepast) moeten volgen.

In het PROTOCOL is sprake van de Ipband "PROTOCOL AANMAAK MKLL". De ELAN sourcetext van deze band is hieronder gegeven:

```
                    'BEGIN' SHORTEN TRAJECT

M[512]:             -1
                    '001 000 000'
M['1607']:          JUMP(0)
M['3761']:          GOTO(:SHORTEN TRAJECT)

M['15425']:
SHORTEN TRAJECT:
                    A = M['1037']
                    M['1036'] = A
                    GOTOR(MC[-1])

                    'END'
```

Voorts is er sprake van het flopprogramma.
Dat is het ALGOL 60 programma:

begin integer i ; L: goto L end

In pt. 6 van het PROTOCOL wordt M['1036'] dat is length of compiler met '50 000' verhoogd. De bibliotheek van MKLL (6/4/70) is voor ± 18K gevuld en past daar dus ruim in.
Wil men meer dan 20K gebruiken dan dient men er op bedacht te zijn dat length of compiler genoeg wordt opgehoogd zonder dat de tables overschreven worden.

Uiteraard is de ELAN sourcetext van de vorige pagina afhankelijk van het assemblaat.
Men raadplege dan ook de labellijst uit [7].

Het PROTOCOL luidt nu als volgt:

1. Vul het geheugen met +0.
2. LS; IP; papertape biband M64KL inlezen;
3. LS; IP; SVA↑; MICA-ipband inlezen; SVA↓;
4. LS; NB;
   eerste 3 bibliotheekvulprogramma's inlezen.
5. flop prog 2* inlezen; laat eerste in executie komen.
6. SVA↑; lees S uit; Maak M['1036'] gelijk aan M['1036'] + '50 000';
   SGA op SUBMONITOR = '15001'; Herstel S;
   SVA↓; BVA; F-inslag op Commandoteleprinter;
7. Na stop op SGA; LS; IP; SVA↑; Ipcorrectie "PROTOCOL AANMAAK MKLL" inlezen
8. LS; NB;
9. systeem wegschrijven met verplaatsing als beschreven in [6].
   vrije traject op '126111'.

IV-c MKL(L) → VUMKL(L).

Naast de systemen MKL en MKLL bestaan ook de systemen VUMKL en VUMKLL.

De eigenschappen van VUMKL en VUMKLL zijn:

1 TO DRUM & FROM DRUM in ALGOL heeft beschikking over 300K op de trommel.

2 het objectprog heeft beschikking over ± 53K in de kernen.

3 op regel 3 van het laatste output-vel (dat door het systeem aan de regeldrukker-output wordt toegevoegd) wordt, indien het programma in executie is geweest, afgedrukt hoeveel woorden tijdens executie in beslag genomen zijn.

Opmerking De tabellen staan bij VUMKL en VUMKLL op dezelfde plaats als bij MKL en MKLL. Voor een programma met een groot text array is het VU-systeem dus niet geschikt.
Zo'n systeem kan gemakkelijk gemaakt worden door MKL opnieuw te assembleren.

Men verkrijgt het VUMKLL systeem door:

1 LS; IP systeemlezer; lees MKLL in;
2 LS; IP; SVA↑; lees MKL(L) → VUMKL(L) in; SVA↓;
3 LS; NB;
en men heeft het VUMKLL systeem.

```
M[521]:          'BEGIN' CONSIDER, AFXT6, BCHECK, RUN, MEMORY USABLE TO,
                         CONSIDER DUTY, RUN DUTY, ENDRUN DUTY,
                         BEGIN OF PR AR, maxB, last action, INITRE,
                         FROM DRUM RE, PROCESS4, INITPR1, FROM DRUM PR,
                         PROCESS1, MAX4, MAX, ENTRIS4


                 '000 230 000'              " 76 K
                 '000 230 000'              " 76 K
                 '000 430 000'              " 140 K
                 '000 430 000'              " 140 K
                 '000 540 000'              " 180 K
                 '000 540 000'              " 180 K
                 '000 640 000'              " 212 K
                 '000 640 000'              " 212 K
                 '002 000 000'              " 512 K

" FOR EVERY PROCESS THE RELATION DREND - DRBEG = :BUFLENGTH ×:MAX MUST HOLD

M['5243']:       INITRE:
M['5405']:       FROM DRUM RE:
M['5425']:       PROCESS4:
M['12726']:      INITPR1:
M['13351']:      FROM DRUM PR:
M['13365']:      PROCESS1:
M[512]:          MAX4:
M[128]:          MAX:
M['16442']:      ENTRIS4:
INITRE[7]:                          A = :MAX4
FROM DRUM RE[11]:                N, a - :MAX4, Z
PROCESS4[19]:                    N, S - :MAX4, Z

INITPR1[8]:                      A = :MAX
FROM DRUM PR[7]:              N, A -:MAX, Z
PROCESS1[21]:                 N, S - :MAX, Z
M['1043']:       begin of pr ar:
M['14123']:      AFXT6:
M['1202']:       last action:
M['15335']:      CONSIDER:                  GOTO(:CONSIDER DUTY)
M['17604']:      RUN:
M['15334']:      BCHECK:
M['1045']:       MEMORY USABLE TO:          '177 774'
ENTRIS4[0]:      GOTO(:CONSIDER)
```

```
RUN[1]:
M['1762']:
M['15430']:      maxb:
CONSIDER DUTY:


RUN DUTY:


ENDRUN DUTY:



      'END'
```

```
   GOTO(:RUN DUTY)
N, SUBC(:ENDRUN DUTY)
   'SKIP' 1
U, B - maxb, P
Y, MAXB = B
U, B - BCHECK, P
   GOTO(:CONSIDER[1])
   maxb = B

   A = MEMORY USABLE TO
   GOTO(:RUN[2])
   SUBC(:AFXT6)
   S = last action, Z
   G = maxb
   G - begin of pr ar
Y, GOTO(:AFXT6)
   GOTOR(MC[-1])
```

Litteratuur

[1] MICRO48K up to date 14-7-69,
regeldrukkerafdruk ELAN sourcetext MC-ALGOL 60 systeem

[2] Verslagen bespreking MC-bedrijfssysteem MICRO48K
dedato 4/7, 8/7, 10/7, 14/7, 15/7, 21/7, 23/7 / 1969

[3] Kruseman Aretz F.E.J. and Mailloux B.J.
The ELAN sourcetext of the MC ALGOL 60 system
for the EL X8, MR84, Mathematisch Centrum Amsterdam 1966

[4] Koksma K.K.
RA W44 Het bedrijfssysteem MICRO48KL, Mathematisch Centrum
Amsterdam 1969

[5] Verslagen bespreking MR84, 1969

[6] Brinkhuysen R.
Systemen op magneetband (in voorbereiding)

[7] Koksma K.K.
The ELAN sourcetext of MICRO64KL, Mathematisch Centrum
Amsterdam 1970.

[8] Revised Report on the Algorithmic Language ALGOL 60
P. Naur, editor, Regnecentralen, Copenhagen 1962

[9] Roos Lindgreen H.W.
MICA Een aanvulling en wijziging van MICRO48K (en MICRO48KL) ten
behoeve van het gebruik van de kaartlees apparatuur, NR10,
Mathematisch Centrum, 1970

[10] Kruseman Aretz F.E.J.
ALGOL 60 Translation for Everybody, elektronische datenverarbeitung
Heft 6/1964.