'n ONDERSOEK NA DIE GESKIKTHEID VAN 'n DATAVLOEI-VERWERKER AS 'n HERSTRUKTUREERBARE SPESIALE VERWERKER

'n Tesis uitgevoer ter behaling van die graad Magister in Ingenieurswese

deur

N. J. Loubser

aan die

UNIVERSITEIT VAN STELLENBOSCH

November 1984

BEDANKINGS

Die skrywer wil graag die volgende persone vir hul raad en bystand bedank: Mnr. PJ Bakkes my promotor, my mede-eksaminator Mnr. WH Steyn en Prof. JJ du Plessis.

Die WNNR sowel as die Universiteit van Stellenbosch word bedank vir hul finansiële bystand gedurende die studietydperk.

OPSOMMING VAN DIE TESIS

Hierdie tesis behels 'n ondersoek na die geskiktheid van 'n datavloei-verwerker om as 'n herstruktureerbare spesiale verwerker te dien.

Die werking van 'n datavloei-verwerkermodel word aan die hand van datavloeikonsepte verduidelik. Die tekortkominge van die model, naamlik die gebrek aan datastruktuur-hanterings, toevoer/afvoer en hertoelatingsmeganismes word uitgelig en moontlike oplossings word gegee.

'n Gemodifiseerde datavloei-model, wat beide struktuur-hantering en toevoer/afvoermeganismes insluit, word voorgestel. Hertoelating word met behulp van 'n datapakket-benamingsmetode bewerkstellig. Om die programmeerbaarheid en die herstruktureerbaarheid van die model te ondersoek, is besluit om 'n datavloei-verwerker te simuleer.

Die model is met behulp van die hoëvlaktaal PASCAL, en bedryfstelselroepe op die VAX 11/780 rekenaar gesimuleer. Parallelle verwerkingskonsepte in beide programmatuur en argitektuur word gedemonstreer.

INHOUDSOPGAWE

HOOFSTUK 1

1 INLEIDING

In die laaste dekade het die vooruitgang in baiegrootskaalse integrasietegnologie 'n waardevolle bydrae gelewer ten opsigte van die ontwerp en bou van rekenaarstelsels. Geïntegreerde stroombane het stelselgrootte en koste drasties laat afneem terwyl spoed en betroubaarheid verhoog is. Ten spyte van hierdie gunstige faktore, is daar min navorsing gedoen om argitektuurkonsepte, wat Von Neumann veertig jaar gelede gedefineer het, te herevalueer.

'n Aanvraag na spesiale verwerkerstelsels, met verwerkingspoede wat etlike ordes groter is as in bestaande stelsels, het ontstaan. Syfervretertoepassings, byvoorbeeld simulasie, syferseinverwerking en ryverwerking, regverdig sulke stelsels. Indien die onderliggende tegnologie in huidige stelsels verbeter word, sal dit nie die gevraagde spoedverhoging lewer nie. Tans word ander moontlikhede, byvoorbeeld die gebruik van groot parallelle verwerkingstelsels, oorweeg.

1.1 Huidige spesiale verwerkingstelsels

In die wedloop om vinniger verwerkers te bou, het stelselontwerpers versuim om 'n van bo-af ("top down") probleembeskouing te volg. Die aandag was toegespits op die verbetering en modifisering van vorige verwerkergenerasies om sodoende die hoof knelpunte te omseil. Voorbeelde van geïmplementeerde modifikasies, is gepyplynde kritiese stadiums, vektorverwerkingseenhede, meervoudige funksionele eenhede, tussengeheues, instruksie-tussengeheues en geheue-invlegging.

Ten spyte van hierdie modifikasies, is argitektuurtipes nog steeds gebaseer op die beheervloeikonsep van sekwensiële

programuitvoering. Twee algemene voorbeelde van huidige tipes spesiale verwerkers, is die ryverwerker en die horisontaal-geprogrammeerde bissnitverwerker.

(i) Ryverwerkers - Hierdie spesiale verwerker bestaan uit 'n aantal verwerkingseenhede wat met behulp van 'n vaste interverbindingsnetwerk gekoppel is. 'n Enkele meester beheer die verwerkers deur instruksies en data, via die verbindingsnetwerk, na hul te stuur. Die verwerkereenhede is programmeerbaar en werk sinchroon, onder direksie van die beheereenheid, aan wiskundige of kommunikasie-operasies.

(ii) Horisontaal-programmeerbare bissnitverwerkers - Die stelsel bestaan uit 'n programmeerbare beheerder en 'n aantal homo- of heterogene verwerkerelemente. Elke verwerkingselement werk op 'n snit van die beheerwoord in. Asinchrone verwerking vind in 'n woordeenheid plaas, maar die taakverwerking word in geheel sekwensieel deur die beheereenheid uitgevoer. Die langste snitbewerking in die stelsel bepaal die maksimum beheerderklokspoed.

1.2 Evaluering van huidige spesiale verwerkers

Die volgende twee kriteria moet beskou word in die evaluasie van huidige of nuwe tipes spesiale verwerkers: Eerstens moet die afbeeldingsproses van die algoritme na die verwerker effektief en eenvoudig wees. Ten tweede moet die algoritme verwerking so spoedig moontlik kan plaasvind.

Omdat die huidige spesiale verwerkers met die tweede kriterium as primêre doel ontwerp was, ontstaan 'n aantal probleme:

(i) Algoritme-afbeelding - Die afbeeldingsproses behels dat die algoritme vooraf ondersoek moet word vir moontlike

inherente parallelisme. Dit is die plig van die programmeerder om alle parallelle verwerkingsmoontlikhede op 'n eksplisiete wyse aan die kompileerder en stelsel oor te dra. Dit impliseer dat die algoritme by die stelsel aangepas moet word, sodat die programmeerder genoodsaak is om 'n goeie kennis van die stelselargitektuur te hê.

(ii) Sinchronisasie - Indien 'n nie-funksionele taak op 'n multiverwerkerstelsel (byvoorbeeld 'n ryverwerker) geloop word, benodig dit baie verwerkingskoördinasie. Heelwat tussenverwerker sinchronisasie (en dus kommunikasie) word vir die koördinasie benodig en dit verlaag effektiewe stelselverwerkingsvermoë.

(iii) Kontensie - Kontensie kom voor indien 'n aantal verwerkereenhede hulpbronne, byvoorbeeld geheue, invoer/afvoer of kommunikasie-eenhede moet deel. Verwerkerleeglooptye (wagtoestande) verlaag effektiewe verwerking.

(iv) Oorhoofse koste - Die koste behels data- en kode-duplisering asook die ekstra kode wat benodig word vir parallelle verwerkings-koördinasie.

As gevolg van hierdie faktore, sal die vermeerdering van die aantal verwerkers in 'n parallelle verwerkingsopstelling nie 'n ooreenkomstige spoedverhoging lewer nie. 'n Alternatiewe verwerker-argitektuur, wat meer geskik vir parallelle verwerking is, word dus gesoek.

1.3 Moontlike oplossings

Twee alternatiewe spesiale verwerkertipes word tans beskou, naamlik reduksieverwerkers en datavloeiverwerkers.

(i) Reduksieverwerkers [1] – Reduksieverwerking behels die verwerking van instruksie-stringe met behulp van lamda algebraïese metodes. Hierdie stelsel bestaan basies uit 'n aantal hoogs gededikeerde subverwerkers, wat 'n gegewe taak teen 'n baie hoë spoed kan uitvoer. 'n Sistoliese verwerkerstelsel [2] is 'n goeie voorbeeld van 'n homogene reduksieverwerker. Die argitektuur bestaan uit 'n aantal eenvoudige verwerkerelemente wat in 'n vaste interverbindingstruktuur gerangskik is. Die verwerkingselemente kan byvoorbeeld slegs sommerings- of vermenigvuldigingsfunksies uitvoer. Die stelsel voer spesifieke take soos korrelasie, FFT's of filtrering uit.

(ii) Datavloei-verwerkers [3] – Die sogenaamde datavloeiverwerker is gebaseer op die konsepsionele datavloeimasjienmodel van Dennis. Die model is basies 'n taalgeoriënteerde verwerker wat datavloeidiagramme kan uitvoer. Die maksimale parallelle verwerking wat 'n gegewe algoritme bied, kan teoreties op hierdie manier benut word. Die konsep behels dat verwerking nie deur beheer aangedryf word nie, maar deur die aanwesigheid van data. Parallelle verwerkingsmoontlikhede volg outomaties indien genoegsame data vir 'n aantal bewerkings gelyktydig bestaan.

Uit die bostaande uitgangspunte blyk datavloei oënskynlik die mees herstruktureerbare van die twee tipes verwerkers te wees, omdat 'n hoër vlak van probleembeskouing gebruik word. Aangesien die Universiteit van Stellenbosch se Elektries en Elektroniese Ingenieursdepartement 'n behoefte het na 'n herstruktureerbare spesiale verwerker om in 'n multiverwerkeropstelling te dien, is die doelstelling van hierdie tesis 'n ondersoek na die konsepte en argitektuur van 'n datavloeiverwerker, as alternatief tot huidige spesiale verwerkers.

HOOFSTUK 2

2 DATAVLOEI-KONSEPTE EN TEORIE

In teenstelling met die beheervloei-begrip in Von Neumann argitekture, het datavloei geen volgorde of sekwensie van programuitvoering nie. Beheervloei berus op die beginsel van 'n programteller wat die programuitvoering dirigeer, terwyl datavloei-programuitvoering deur die beskikbaarheid van data aangedryf word.

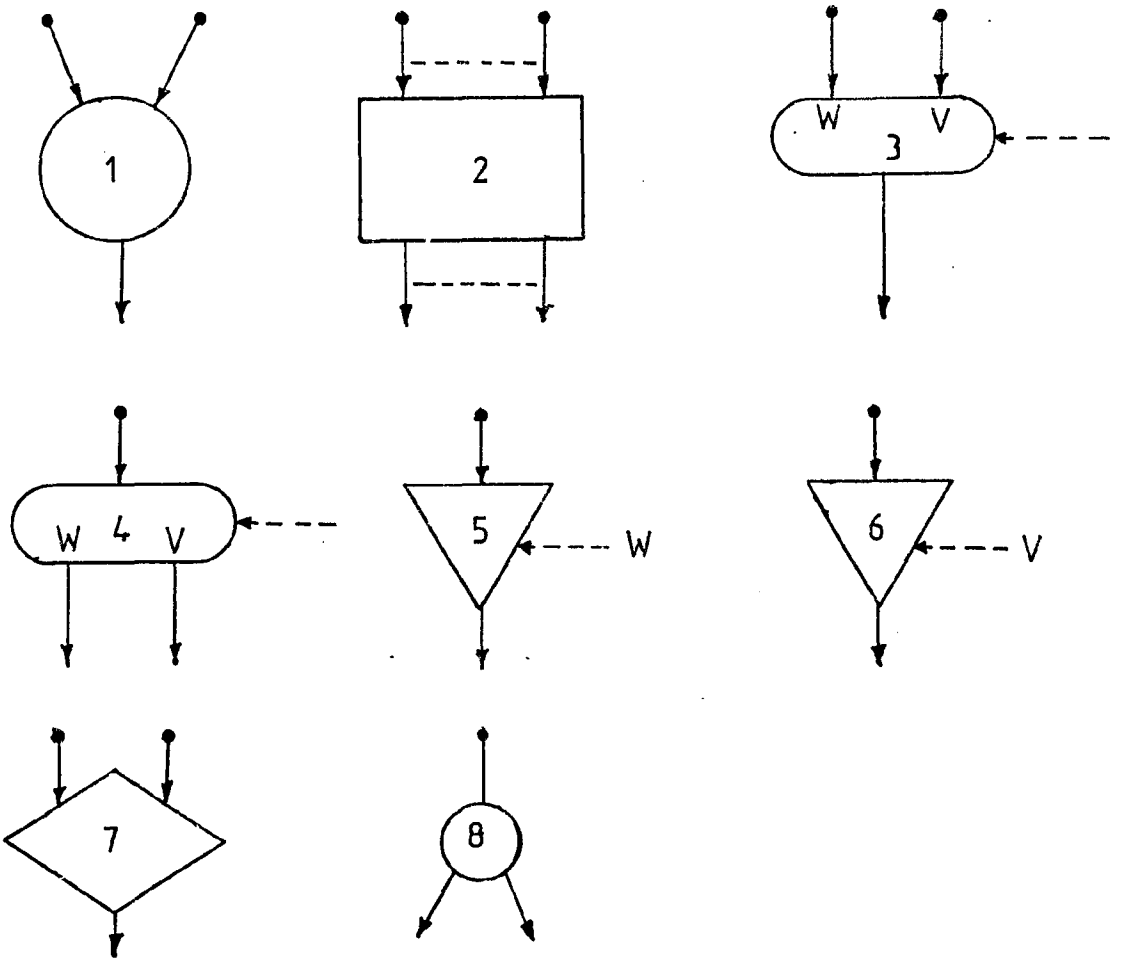
Datavloei het geen begrip van veranderlikes nie, aangesien 'n veranderlike die deel van 'n gemeenskaplike geheue behels. 'n Datavloei-"veranderlike" word voorgestel deur 'n verbindingsboog wat 'n datawaarde van een bewerking na 'n ander dra. Die twee hoof eienskappe van datavloei kan as volg opgesom word:

(i) Asinchrone verwerking - 'n Aantal bewerkings kan parallel plaasvind indien daar min data-afhanklikheid in die betrokke verwerkingstaak voorkom.

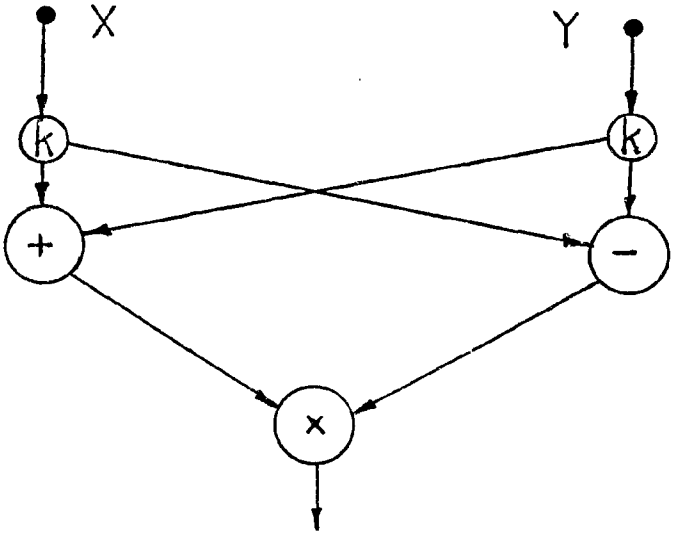
(ii) Funksionele verwerking - Datavloei-verwerking is lokaal aangesien 'n spesifieke bewerking nie invloed op ander bewerkings, behalwe die afvoer-destinasie bewerkings, het nie. Geen newe-effekte is dus in 'n datavloei-omgewing moontlik nie.

2.1 Datavloei-diagramme

Aangesien datavloei-diagramme die grafiese masjientaal van datavloei-verwerkers is, sal die diagrambeginsels [4] aan die hand van 'n paar voorbeelde geïllustreer word. Die basiese boublokke van 'n datavloei-diagram word in figuur 2.1 aangegee. Die boublok funksies word nou gegee:



FIGUUR 2.1: Datavloei-diagram boublokke



$$Z = (X + Y)(X - Y)$$

FIGUUR 2.2: 'n Eenvoudige datavloei-diagram voorbeeld

(1) Bewerkingsoperator - Die operator voer die aangeduide bewerking uit op die toevoerdata en versprei die resultate na die afvoere.

(2) Funksie-operator - Die operator bestaan uit 'n mini-datavloei-diagram en kan as 'n makro-operator beskou word.

(3) Ineenvoeg-operator - Die linker of regter toevoerdata-pakket word, afhangende van die toevoerbeheersein wat aan die operator aangelê word, deurgelaat na die afvoer.

(4) Skakel-operator - Die toevoerdatapakket word, afhangende van die beheersein wat aangelê word, na die linker of regter afvoer deurgelaat.

(5) Waarhek-operator - Indien 'n "waar" beheersein aan hierdie operator aangelê word, laat dit die toevoerdata-pakket deur. Indien 'n "vals" beheerwaarde egter aangelê word, absorbeer die operator die toevoerdatapakket.

(6) Valshek-operator - Indien 'n "vals" beheerwaarde aan hierdie operator aangelê word, laat dit die toevoerdata-pakket deur. Indien 'n "waar" beheerwaarde egter aangelê word, absorbeer die operator die toevoerdatapakket.

(7) Besluitnemings-operator - Hierdie operator voer 'n boolese operasie uit op die toevoerdatapakkette en stuur die resultaat uit na die afvoere.

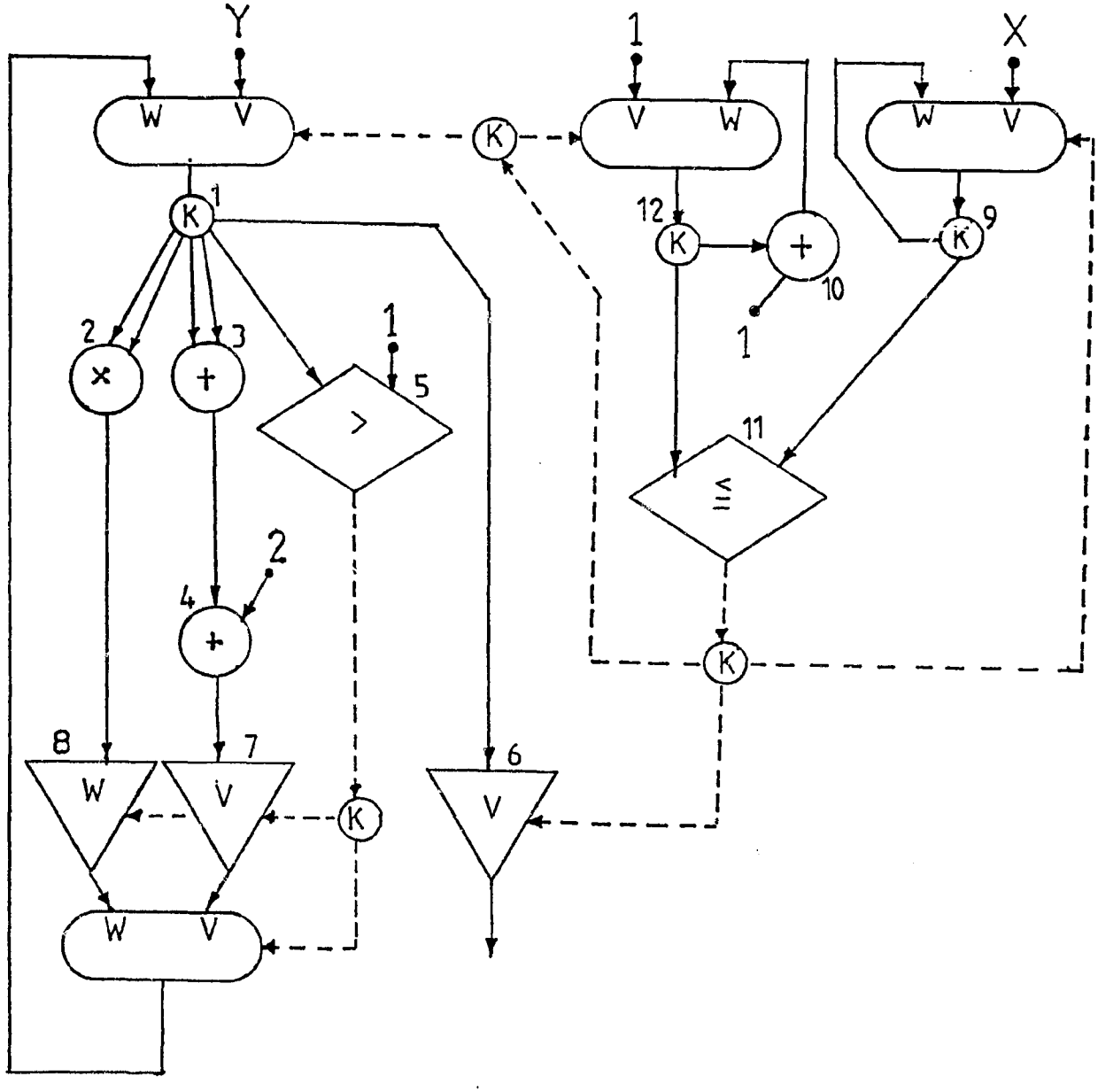
(8) Kopieer-operasie - Die operator kopieer slegs die toevoerdatapakket na die benodigde afvoerdestinasies.

'n Eenvoudige datavloei-diagram word in figuur 2.2 , saam met die wiskundige funksie wat dit verteenwoordig, gegee. Die

```
INPUT X,Y

FOR I = 1 TO X DO
  BEGIN
    IF (Y > 1)
      THEN Y = Y*Y
      ELSE Y = Y + Y + 2
  END

OUTPUT Y
```



FIGUUR 2.3: 'n Datavloei-diagram

vloei van data in die diagram word nou verduidelik. Die diagram is opgebou uit 'n aantal operators wat met behulp van verbindingsboë aanmekaargekoppel is. Data vloei langs die toevoerverbindingsboë na die operators. Die operator word gesneller indien die data op albei sy toevoerverbindingsboë verskyn. Die geaktiveerde operator verwyder nou die toevoer-datapakkette en voer die operasie uit om die resultaat te produseer. Die resultaat word nou via die afvoerverbindingsboë na die daaropvolgende operators versprei.

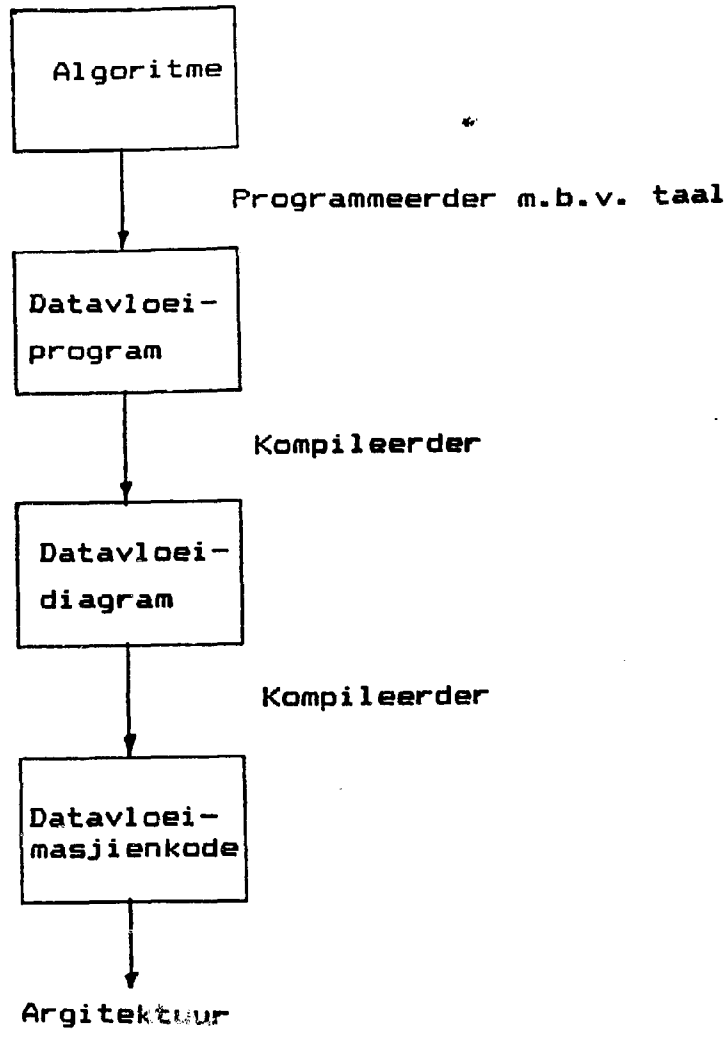
Datavloei-diagramme ondersteun die gebruik van konstantes. Die konstante waardes word op die betrokke toevoerverbindingsboë hergenereer na elke operasie uitvoering.

Let op dat data sleg in een rigting op die verbindingboë vloei. Hierdie eienskap maak dit moontlik om die datavloei met behulp van vloeibeheeroperators te beheer. Iterasies, spronge en voorwaardelike spronge kan met behulp van die vloeibeheer-operators bewerkstellig word. Vloeibeheereenhede word deur die boolese beheerseine (met stippellyne in figuur 2.3 aangedui) van besluitnemings-operators beheer.

'n Meer volledige voorbeeld, 'n algoritme saam met sy datavloei-diagram, word in figuur 2.3 gegee. Die algoritme bestaan uit 'n lus waarin 'n voorwaardelike funksie-uitvoering plaasvind. Let op dat daar konstantes op die toevoer-verbindingsboë van party operators verskyn. Indien die aanvanklike beheerwaardes van die diagram as "vals" aangeneem word, en die beginwaardes word op die betrokke toevoer-verbindingsboë geplaas, volg 'n logiese datavloei deur die diagram.

2.2 Datavloei-afbeeldingsmetodes

Datavloei-afbeelding is die proses waarby 'n algoritme na sy



FIGUUR 2.4: Die afbeeldingsproses

```
PROCEDURE GETRS (X,Y : REAL)
BEGIN
  RS = X*X + Y*Y
  (* RS IS GLOBAAL VERKLAAR *)
END
```

FIGUUR 2.5: 'n Newe-effek voorbeeld

```
1) K = 3
2) Z = K*5
3) K = 9*K/B
4) Z = K/C
```

FIGUUR 2.6: 'n Program wat sewensiele programuitvoering benodig

datavloei-diagram en dan na 'n datavloei-masjientaalprogram afgebeeld word. Die metode word in figuur 2.4 getoon.

Indien die algoritme eenvoudig van aard is, kan die afbeeldingsproses met die hand gedoen word. Die operasie en sy betrokke destinasie adresse word in die diagram geïdentifiseer en na masjienafhanklike programkode afvertaal.

Indien meer komplekse probleme beskou word, raak hierdie proses egter onaanvaarbaar en betaal dit om 'n hoëvlaktaal en kompileerder vir die afbeelding te gebruik. Die algoritme word nou in die taal gedefineer en met behulp van die kompileerder na 'n datavloeidiagram afgebeeld. Hierdie metode vereenvoudig die leesbaarheid asook die konstruksie van datavloei-diagramme.

Ongelukkig is die huidige hoëvlaktale nie baie geskik vir datavloei-afbeelding nie. Die tale is ontwikkel vir beheervloeiverwerkers en kan nie die asinchrone verwerkings-moontlikhede in datavloei-diagramme hanteer nie. 'n Meer gedissiplineerde taal word verlang, wat aan die funksionele sowel as asinchrone programuitvoerings-kosepte sal voldoen. Die huidige hoëvlaktale se tekortkominge kan as volg opgesom word:

(i) Die tale laat die verdeling van data-areas toe. Indien veranderings aan hierdie gedeelde areas aangebring word, affekteer dit die globale taakverwerking. Die verskynsel heet newe-effekte en is te wyte aan hierdie tale se swak funksionele dissipline. 'n Voorbeeld van 'n newe-effek word in figuur 2.5 gedemonstreer.

(ii) In konvensionele tale is die sekwensiële program-uitvoering noodsaaklik vir korrekte resultaatlewering. Indien aan 'n veranderlike byvoorbeeld meermalig waardes

toegeken word, volg dit logies dat hierdie orde sekwensieel moet verloop. 'n Voorbeeld van hierdie verskynsel word in figuur 2.6 gegee.

'n Aantal tale is reeds ontwikkel met die oog op parallelle programuitvoering in 'n multiverwerker omgewing. Voorbeelde hiervan is LISP en CONCURRENT PASCAL. Hierdie tale voldoen in 'n mindere of meerdere mate aan datavloei-vereistes, maar is nie suiwer genoeg om in 'n datavloei-omgewing te dien nie.

Die gebrek aan geskikte tale is deur datavloei-navorsers waargeneem en daar is besluit om nuwe tale te defineer om aan die nodige datavloei-konsepte te voldoen. 'n Aantal voorbeelde van hierdie tale [5] word nou gegee:

(i) VAL - VAL is deur Ackermann ontwikkel en is MIT se nuutste datavloei-taal weergawe. Die struktuur is sterk funksioneel en elke stelling word as 'n funksie beskou. Die taal hanteer ongelukkig nie toevoer/afvoer of rekursie nie.

(ii) ID - Hierdie taal is deur Arvind en Gostelow by die Universiteit van California (Irvine) ontwikkel vir gebruik op hul datavloei-verwerker. Hierdie uitdrukking- en blokstruktuur georïenteerde taal ondersteun voor-waardelike stellings, prosedures en iterasies.

(iii) LAPSE - Hierdie enkeltoewysingstaal is by die Universiteit van Manchester deur Glauert en Gurd ontwikkel. Die taal is soortgelyk aan PASCAL en ondersteun funksionele subroetines, gedissiplineerde iterasies en gestruktureerde datatipes.

(iv) DDMI - Hierdie is by die Universiteit van Utah deur Davis ontwikkel vir hul DDMI datavloei-verwerker.

HOOFSTUK 3

3 'n KONSEPSIONELE DATAVLOEI-VERWERKER

'n Datavloei-verwerkermodel word verlang om datavloei-diagramme so effektief as moontlik te verwerk. Die verwerkings moet op verskeie vlakke parallel aan mekaar kan geskied, soos die asinchrone konsep illustreer. Drie verwerkingsvlakke bestaan, naamlik:

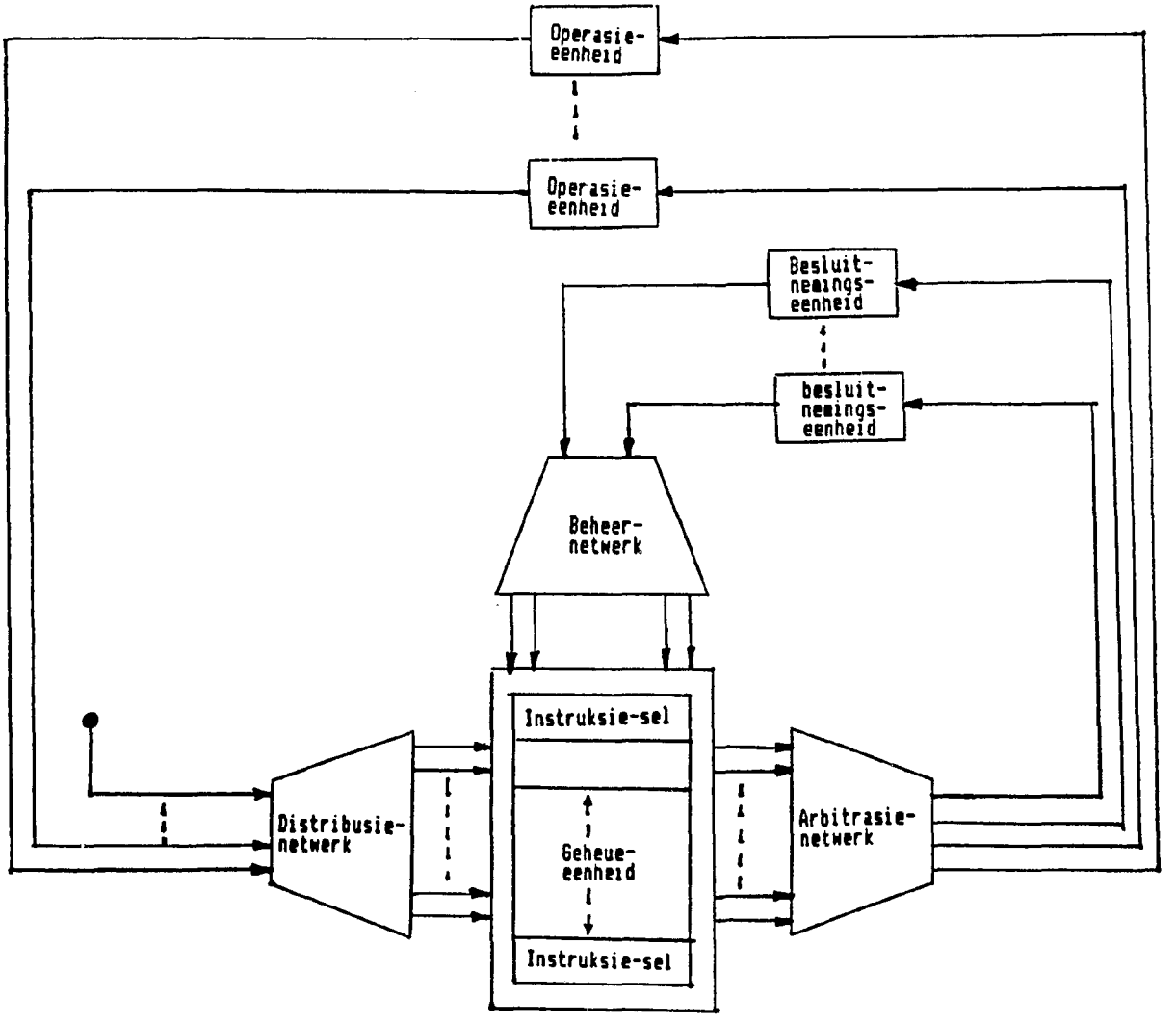
(i) Die masjien instruksievlak.
(ii) Die funksie of prosedurevlak (modulere vlak)
(iii) Die programvlak.

Dennis [6,7,8,9,10] (by MIT) het 'n datavloei-verwerkermodel voorgestel, wat eenvoudige datavloei-diagramme kan uitvoer. Die modelkonstruksie sowel as die werking daarvan, word in hierdie hoofstuk bespreek.

3.1 Die modelkonstruksie

Die voorgestelde verwerker het 'n gepyplynde ringvorm en bestaan uit 'n aantal aktiewe funksionele eenhede, soos in figuur 3.1 getoon word. Dit is duidelik dat die verwerkerargitektuur nie 'n geheue of beheerder het soos in die tradisionele Von Neumann verwerkers nie. Die eenhede, funksies en werking word in die volgende seksies bespreek.

(i) 'n Aktiewe geheue-eenheid.
(ii) 'n Distribusienetwerk.
(iii) 'n Arbitrasienetwerk.
(iv) 'n Beheernetwerk.
(v) 'n Aantal besluitnemings- en operasie-eenhede.



FIGUUR 3.1: Die datavloei-masjien

	Operasie-kode	DESTINASIE-ADRESSE		
a)	BEHEER-KODE	Beheer-vlag	Data-vlag	Dataveld
b)	BEHEER-KODE	Beheer-vlag	Data-vlag	Dataveld

FIGUUR 3.2: 'n Geheue-sel

3.1.1 Die aktiewe geheue-eenheid

Die aktiewe geheue bestaan uit 'n aantal geheueselle. Die toepassings datavloei-diagram word in 'n instruksievorm in hierdie geheueselle gestoor. Elke instruksie verteenwoordig 'n datavloei-diagramoperator. Ses verskillende tipes velde bestaan in die sel soos getoon in figuur 3.2.

(i) Operasie - Hierdie veld bevat 'n instruksiekode wat koördineer met die betrokke datavloei-diagram se ooreenkomstige operasiekode.

(ii) Destinasie-adresse - Die bogenoemde operasie-antwoord word na die aangeduide adresse gestuur.

(iii) Beheerkode - Die kode spesifiseer die beheer wat vir die bepaalde sub-sel van krag is. Vier tipes beheerkodes kan bestaan.

D - Die toevoerdatawaarde word direk in die betrokke sub-sel gestoor. (Geen beheer word toegepas nie)

W - Indien die beheervlag van die sub-sel reeds vroeër met behulp van 'n "waar" beheersein gestel was, word die toevoer datawaarde gestoor. Indien die beheerwaarde egter 'n "vals" waarde was, word die data nie gestoor nie.

V - Indien die beheervlag reeds met 'n "vals" beheersein gestel was, word die toevoerdatawaarde gestoor. Indien die beheersein "waar" was, word die data nie gestoor nie.

K - Dui aan dat hierdie sub-sel 'n konstante datawaarde bevat.

(iv) Die beheervlag het basies drie toestande:

(a) Ongestel

(b) Waar

(c) Vals

(v) Die datavlag - Gestel indien die betrokke sub-sel 'n datawaarde ontvang en aanvaar het.

(vi) Die dataveld - Hierdie veld bevat die betrokke sub-sel datawaarde.

3.1.2 Die distribusie-, arbitrasie- en beheernetwerke

DISTRUBUSIENETWERK

Die distribusienetwerk se funksie is basies om as demultiplekseerder te dien. 'n Paar datapakket-toevoerpunte word versprei na 'n groot aantal instruksieselle. Die netwerk roeteer toevoerdatapakkette (invoer van buite die masjien af vanaf verwerkerelemente) na hul destinasie-adresse.

Die netwerk het 'n vermoë om datapakkette te buffer indien die destinasie-adres reeds data bevat. Daar word aanvaar dat 'n hele aantal roeteringsfunksies parallel in die baan kan plaasvind. (Die netwerk is virtueel die datavloei-diagram se verbindingsboë)

ARBITRASIENETWERK

Hierdie eenheid se funksie is soortgelyk aan 'n multipleks effek, waar 'n groot aantal datatoevoerpunte na 'n aantal afvoerpunte verbind word. Geaktiveerde instruksies (saam met die data) word vanaf die instruksieselle na die verwerker en besluitnemings-eenhede geroeteer. Parallelle roetering is ook in hierdie eenheid moontlik.

BEHEERNETWERK

Hierdie netwerk ontvang resulterende beheerseine vanaf die besluitnemings-eenhede en roteer dit na die betrokke geheueselle om die beheervlae in die sub-selle "waar" of "vals" te stel. Hierdie netwerk is van dieselfde tipe as die distribusienetwerk.

3.1.3 Die besluitnemings- en operasie-eenhede

Die besluitnemings-eenhede voer boolese operasies uit op toevoerdata en stuur die resulterende beheerseine via die beheernetwerk na die geheueselle.

Die operasie-eenhede voer rekenkundige operasies op die toevoerdata uit, en kom dus ooreen met datavloei-operators. Die resulterende datapakkette word aan die distribusienetwerk verskaf.

Enige hoeveelheid van hierdie eenhede kan gebruik word om maksimale parallelle verwerking te bewerkstellig. (Met praktiese oorwegings, soos die roeteringnetwerk-grootte, in ag geneem.)

3.1.4 Werking van die datavloei-verwerker

'n Verwerkingsvoorbeeld gaan aan die hand van die algoritme en datavloeidiagram in figuur 2.3 gegee word. Die voorbeeld sal op 'n sekwensiële wyse verduidelik word, met die volgende faktore in ag geneem.

(i) Op enige tydsnit mag meer as een operasie geaktiveer wees en omdat daar meervoudige operasies beskikbaar is, sal parallelle verwerking plaasvind.

1

kopieer	2a,2b,3a,3b,5a,6a	
D		
—	—	—

2

*	7a	
W		
D		

3

+	4a	
W		
D		

4

+	8a	
D		
K		2

5

>	7a,8a	
W		
K		1

6

kopieer	afvoer	
V		
—	—	—

7

kopieer	2a,2b,3a,3b,5a,6a	
W		
—	—	—

8

kopieer	2a,2b,3a,3b,5a,6a	
V		
—	—	—

9

kopieer	11b,9a	
W		
—	—	—

10

+	10a,11a	
V		
K		1

11

<	2a,3a,5a,6a,10a,9a	
D		
D		

FIGUUR 3.3: Die datavloei-masjientaal

(ii) Die orde van uitvoering is nie dieselfde as wat verduidelik word nie. Enige operasie is uitvoerbaar indien die betrokke data aanwesig is.

Die program word geïnisialiseer deur die beginwaarde X en Y voor die distribusienetwerk in te voer. Die datawaardes word afsonderlik na instruksie 1 en instruksie 9 geroeteer. Nadat die datavloeiprogram uitgevoer is, word die resultaat vanaf die afvoerpunt (Instruksie 6) na die benodigde destinasie gestuur. Die elf datavloei-instruksies word in figuur 3.3 gegee. Die inhoud en funksie van elke instruksie word nou bespreek:

INSTRUKSIE 1 - Geen operasie word uitgevoer nie en data word slegs versprei.

INSTRUKSIE 2 - Vermenigvuldig die twee intreewaardes en versprei die resultate.

INSTRUKSIE 3 - Sommeer die twee intreewaardes en versprei die resultate.

INSTRUKSIE 4 - Sommeer die konstante twee en die resultaat van instruksie 2.

INSTRUKSIE 5 - Toets of die intreewaarde groter is as die konstante 1 en versprei die resulterende beheerwaardes na destinasie.

INSTRUKSIE 6 - Indien 'n "vals" beheerwaarde ontvang word, stuur die intreewaarde na 'n afvoerdestinasie-adres.

INSTRUKSIE 7 - Indien 'n "waar" beheerwaarde ontvang word,

versprei die toevoerdata na destinasie-adres.

INSTRUKSIE 8 - Indien 'n "vals" beheerwaarde ontvang word, versprei toevoerdata na destinasie-adres.

INSTRUKSIE 9 - Indien 'n "waar" beheerwaarde ontvang word, versprei toevoerdatawaarde.

INSTRUKSIE 10 - Indien 'n "waar" beheerwaarde ontvang word, sommeer die toevoerdata met die konstante 1 en versprei die resultaat.

INSTRUKSIE 11 - Toets of die toevoerdata groter is as die toevoerwaarde X en stuur hierdie beheerdata na die betrokke destinasie-adresse.

Die programverloop kom ooreen met die logiese eenrigting datavloei in die sirkelvormige gepyplynde verwerker-konstruksie.

HOOFSTUK 4

4 'n UITGEBREIDE DATAVLOEI-VERWERKER

Die konsepsionele datavloei-verwerkermodel van die vorige hoofstuk, het 'n aantal tekortkominge aangaande sy vermoë om as spesiale verwerker te funksioneer. Die drie basiese tekortkominge word gegee:

(i) Geen meganismes vir data- en struktuurverdeling bestaan nie.

(ii) Geen kodehertoeganklikheids-meganisme bestaan nie. ("re-entrant coding")

(iii) Geen vaste toevoer/afvoermeganisme bestaan nie.

Hierdie faktore beperk die bestaande datavloei-verwerkermodel tot baie eenvoudige probleme. In hierdie hoofstuk word oplossings vir die bogenoemde tekortkominge voorgestel en in 'n nuwe model geïnkorporeer.

4.1 Struktuurhantering

Sekere probleme ontstaan indien datastrukture in 'n datavloei-omgewing geïmplementeer moet word. Aangesien datavloei-konsepte geen gemene geheue of dataverdeling toelaat nie, impliseer dit dat datapakkette volledige datastrukture moet transporteer. Hoër kopiërings en roeterings koste is die gevolg, sodat die verwerkings-effektiwiteit drasties verlaag.

Die moontlikheid van dataverdeling (en dus struktuurverdeling) bestaan wel indien 'n LEES ALLEEN voorwaarde op die geheue geplaas word. Indien slegs een element van 'n datastruktuur in die geheue egter verander word, dwing datavloei-

funksionaliteit die konstruksie van 'n nuwe struktuur af.

'n Gedeelde geheue-eenheid is deur ARVIND [11] as struktuurhanteringsmeganisme voorgestel. Die voorwaarde verbonde aan die geheue-eenheid, is dat slegs een toewysing per geheueposisie mag geskied. Die eenheid het die volgende eienskappe:

(i) Indien 'n geheuelees-operasie voorkom en aan die betrokke geheueposisie is reeds 'n waarde toegewys, dan word die leesoperasie bevredig.

(ii) As gevolg van asinchrone operasie-uitvoering, kan 'n leesoperasie byvoorbeeld voorkom voor die betrokke toewysingsoperasie. In so 'n geval word die leesoperasie gebuffer totdat die toewysingsoperasie voorkom.

Die voorgestelde eenheid kan, as gevolg van die bestaande eienskappe, teoreties asinchrone sowel as parallelle lees- en skryfoperasies hanteer.

4.2 'n Hertoeganklikheidsmeganisme

Die voorgestelde model ondersteun nie kode-hertoeganklikheid nie. Die nodigheid van so 'n meganisme word deur die volgende situasie gedemonstreer:

Indien 'n lusinhoud byvoorbeeld uit 'n funksieroep bestaan, verlang die datavloei-verwerkermodel 'n sekwensiële iterasie-uitvoering. Hierdie redenasie volg op die datavloei-konsep dat slegs een datapakket uit enige tydstip per operator toevoerboog mag bestaan.

Drie moontlike oplossings bestaan vir hierdie probleem. Die oplossings is gebaseer op die feit dat meervoudige data-

pakkette op 'n toevoerdataboog moet kan verskyn, om sodoende hertoeganklikheid te bewerkstellig.

(i) Die iterasie- of funksiekode moet gedupliseer word sodat elke datapakketpaar sy unieke operator het.

(ii) Deur 'n EIEU-buffer op elke operator se intreeboog te plaas, sal dit databuffering verseker. Datapakket toevoersekwensie word in die bufferingsproses behou.

(iii) GOSTELOV [12,13] het voorgestel dat alle ooreenstemmende datapakket operatorpare, unieke name toegewys moet word. Korrekte resultate word verseker deur slegs operasies op naamgenoot datapakkette uit te voer.

Die eerste metode vereis dinamiese kodegenerasie, terwyl die tweede metode 'n groot aantal EIEU-buffering behels. Beide hierdie stelsels het 'n hoë geheueverbruik gedurende programlooptyd en is dus meer akademies van aard.

Die derde metode laat gelyktydige gebruik van programkode toe, en is ideaal geskik vir 'n parallelle stelsel waar lusiterasies of funksies asinchroon verwerk moet word. Die metode berus op die beginsel dat elke operasie gelyktydig 'n aantal logies geskeide toevoerboogpare kan besit. Om hierdie metode toe te pas, vereis die volgende ondersteuningsmeganismes:

(i) 'n Naamgenerator wat aan elke lus-iterasie en funksieroep 'n unieke konteks kan voorsien.

(ii) 'n Verbindingseenheid wat naamgenoot datapakkette vinnig en effektief kan saamgroepeer.

(iii) 'n Naambewerkingsmeganisme sodat lus- en funksieroepe

sowel as terugkering bewerkstellig kan word.

Metodes vir die implementering van bostaande meganismes word nou bespreek.

4.2.1 Die naamgenerator

Die naamgenerator kan tipies met behulp van 'n getalsisteem geïmplementeer word. Twee moontlike sisteme word gegee:

(i) 'n Tellersisteem wat byvoorbeeld unieke name (getalle) aan die stelsel verskaf. Die teller moet egter groot genoeg wees om die uniekheid van die name in die stelsel gedurende looptyd te verseker.

(ii) 'n Aantal name (getalle) bestaan in die sisteem en 'n hulpbronbestuurder beheer die gebruik daarvan. Indien 'n konteks aangevra word, verskaf die hulpbronbestuur 'n unieke getal, en teken dit as beset aan. Nadat die getal klaar gebruik is, word dit deur die hulpbronbestuurder herwin, en word as onbeset aangeteken.

Die bogenoemde stelsels sal tipies sentraal geleë wees in 'n multiverwerker-opstelling om konteks uniekheid te verseker. Dit is egter moontlik om 'n aantal konteksbestuureenhede te hê, waar elke bestuurder sy unieke name kan bestuur.

4.2.2 Die verbindingseenheid

Die eenheid se funksie is om naamgenoot datapakkette saam te voeg. Een datapakket sal tipies vir sy maat in die eenheid moet wag om die ontmoeting te bewerkstellig. Indien sy naamgenoot voorkom, sal hulle saamgevoeg en aangestuur word.

Die eenheid het 'n baie effektiewe soekproses nodig om so

vinnig as moontlik die naamgenoot op te spoor. 'n Assosiatiewe geheue is goed geskik vir hierdie taak, aangesien dit moontlik is om die naamgenoot direk met die toevoerdatapakket se naam te adresseer.

4.2.3 Die naambewerkingseenheid

Sekere naambewerkings-operators word benodig om datapakketname dinamies gedurende die looptyd te verander. Die volgende voorbeelde dui die benodigdheid aan:

(i) Om 'n funksie in 'n unieke konteks te roep, moet die funksie toevoerdatapakkette se konteks met 'n nuwe konteks vervang word.

(ii) Indien die funksie afvoerdatapakkette na die roepkonteks wil terugkeer, moet die konteks daarvan deur die oorspronklike konteks vervang word.

Aangesien iterasie-konteksveranderinge baie meer voorkom as funksie-konteksveranderinge , is dit beter om die veld op te deel. Indien die iterasies op dieselfde manier as die funksieroep behandel word, sal die naamgenerator moontlik versadig. (Indien byvoorbeeld slegs een naamgenerator bestaan)

'n Ekstra naamveld, naamlik die iterasieveld, word vir die iterasie-konteksveranderinge geskep. Hierdie veld verseker 'n unieke konteks vir elke iterasie indien dit gewis word voor 'n lus begin, en geïnkrementeer word gedurende elke iterasie. Indien die lus voltooi, word hierdie iterasieveld na die oorspronklike verander. Dit verseker dat die lus afvoerdata-pakkette na die lusroepvlak iterasiediepte kan terugkeer. Die volgende naambewerkings-operators word nou gedefinieer:

K - Hierdie operator vervang die ou konteks met 'n nuwe

en stuur die oue na die INV K operator.

INV K – Hierdie operator vervang die nuwe konteks met oorspronklike.

I – Hierdie operator herstel die iterasieveld na eenheid en stuur die oue na die INV I operator.

INK I – Hierdie operator inkrementeer die iterasieveld.

INV I – Hierdie operator herstel die iterasieveld na die oorspronklike iterasiediepte.

4.2.4 'n Voorbeeld van die hertoelatingsmeganisme

Die algoritmevoorbeeld word, saam met sy verteenwoordigende datavloei-diagram, afsonderlik in figuur 4.1 en figuur 4.2 vertoon.

'n Tyddiagram van die algoritme uitvoering (in die geval waar twee verwerkers beskikbaar is) vir beide die nie-hertoelaatbare en toelaatbare verwerking-metodes word in figuur 4.3 getoon. (Slegs die interne lusbewerkings word in ag geneem) Die metodes word nou verduidelik:

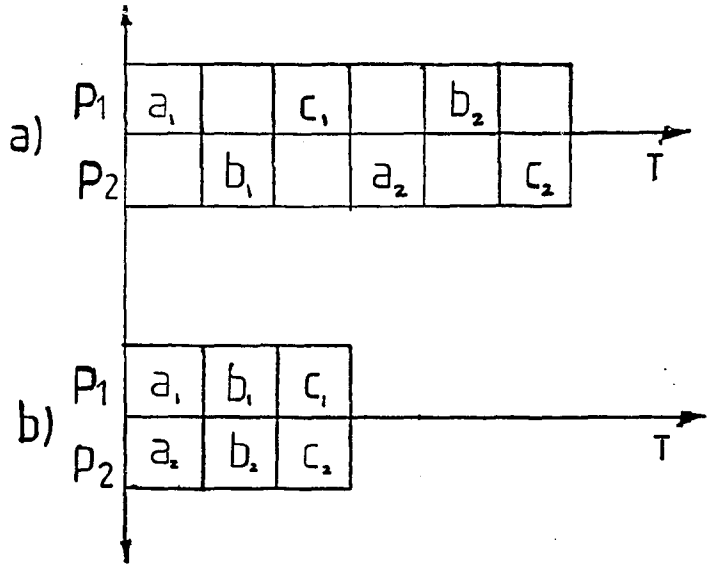
(a) Hertoelating van kode word nie ondersteun nie. Die verwerking stem ooreen met die datavloei-verwerkermodel in die vorige hoofstuk.

(b) Hertoelating word ondersteun met behulp van een van die volgende drie metodes:

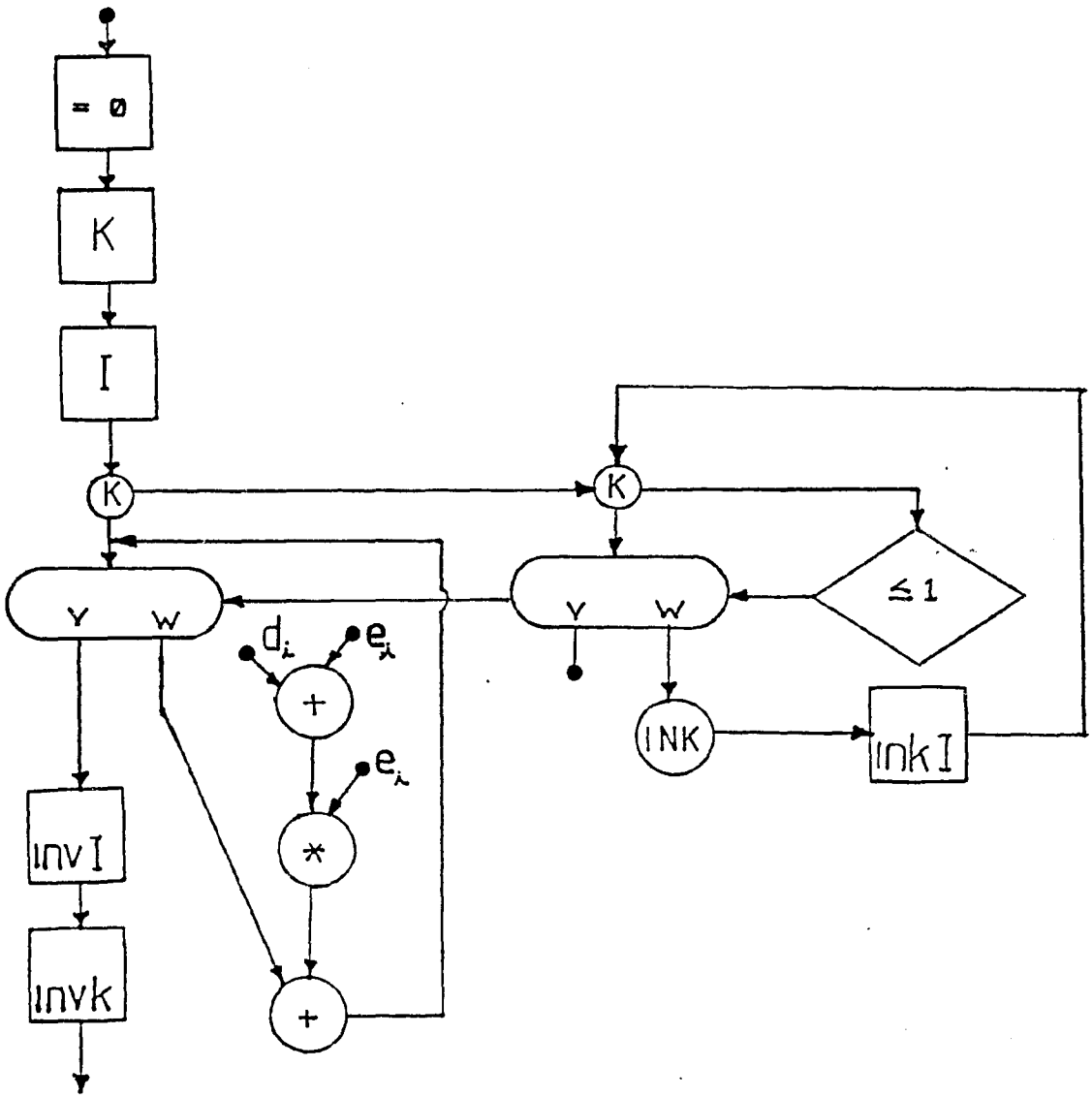
1) EIEU-buffers op alle nodetoevoerboë.
2) Dinamiese kodeduplisering.

```
INPUT D[],E[]
C_I = 0
FOR I = 0 TO 1 DO
  BEGIN
    A_I = D_I + E_I
    B_I = A_I + E_I
    C_I = B_I + C_I
  END
OUTPUT C[]
```

FIGUUR 4.1: Die algoritme



FIGUUR 4.3: Die tyddiagram



FIGUUR 4.2: Die datavloeidiagram

3) Datapakkette dra konteks- en iterasievelde saam.

4.3 Toevoer/afvoer

Toevoer/afvoer is in die voorafgaande datavloei-verwerkermodel bewerkstellig deur die ring tydelik voor die distribusie-netwerk te breek, sodat datapakkette toe- en afgevoer kan word. Dit is duidelik dat hierdie metode minder geskik is vir 'n meer praktiese opstelling. Om 'n toevoer/afvoermeganisme vir 'n nuwe model te ondersoek, moet die volgende probleem-situasies beskou word.

(i) Toevoer - As gevolg van die hertoelatingsmeganisme se dinamiese benaming, moet die naam en die iterasieveld van elke struktuurelement se intreepunt bekend wees om die datapakket-verbindingsproses te bevredig.

(ii) Afvoer - As gevolg van die asinchrone verwerking in die model, kan die datastruktuur-afvoersekwensie nie gewaarborg word nie.

Dit is dus noodsaaklik dat geordende buffering vir beide data-toevoer en afvoer moet geskied. Hierdie toevoer/afvoer-eenheid moet 'n metode hê om die nodigheid vir die betrokke aksie waar te neem. Neem die geval waar 'n afvoerstruktuur gebuffer word. Indien al die struktuur elemente reeds waardes toegeken is, moet die eenheid dit kan waarneem om sodoende die geordende afvoer te bewerkstellig.

Die aanvanklike program-aktiveringswaardes kan steeds in die ring gevoeg word, aangesien die konteks nie aan die begin bekend hoef te wees nie.

4.4 Die nuwe modeluitleg

Die uitgebreide datavloei-verwerkermodel verskil van die vorige model in die volgende aspekte:

(i) Die verwerkerelemente behartig vloeibeheer, benaming asook gewone wiskundige bewerkings.

(ii) Die distribusie- en arbitrasienetwerke word in een skakelaareenheid saamgevat om onnodige stelsel-verspreiding te verhoed.

(iii) Die geheueselstruktuur word vervang met 'n samevoegings- en instruksiehaal-eenheid.

(iv) 'n struktuurhanterings-eenheid word gebruik vir beperkte dataverdeling.

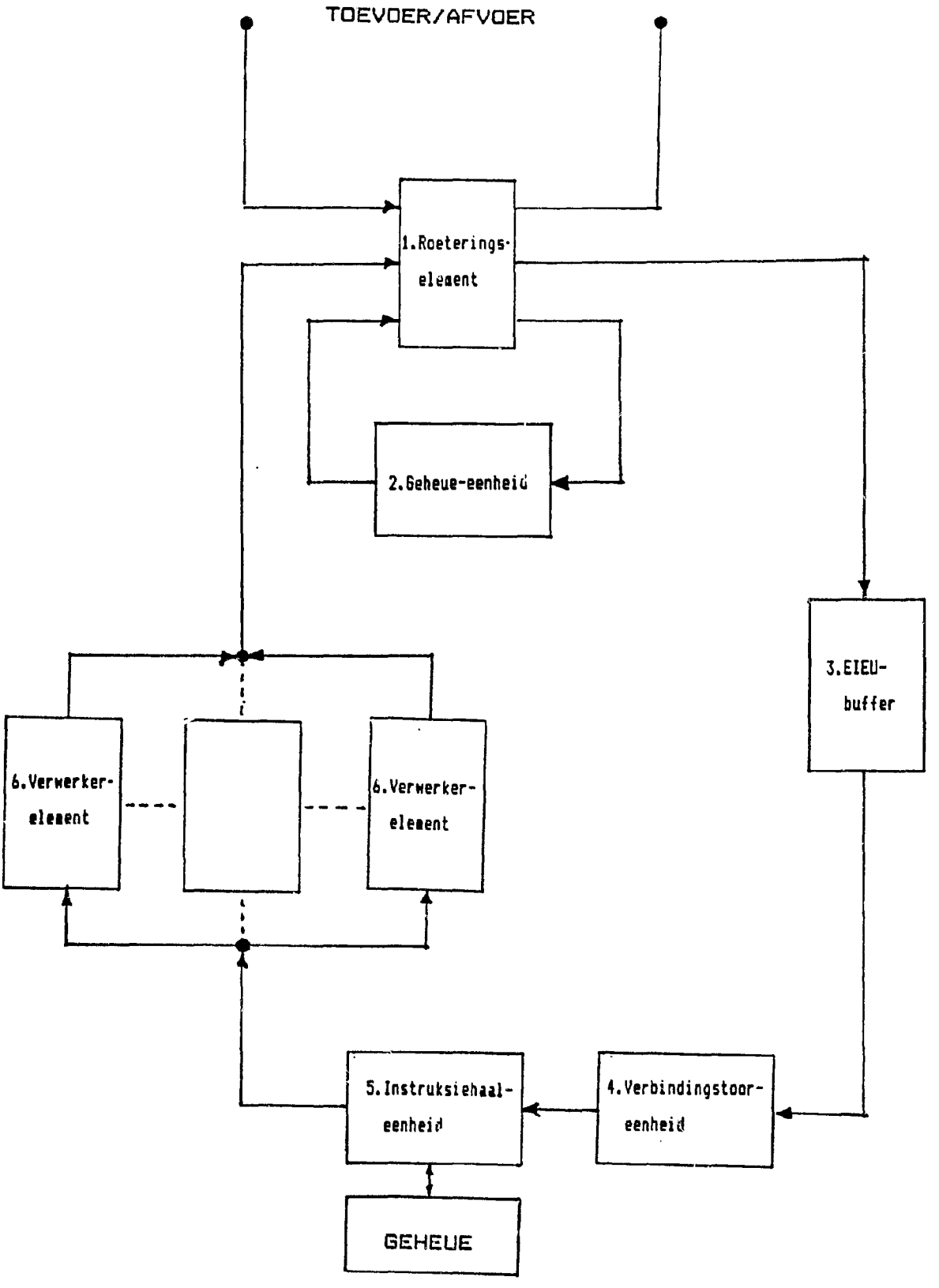
Die datavloei-verwerkereenhede word aan die hand van figuur 4.4 gegee:

(1) Die roeteringselement - Hierdie element roeteer data-pakkette vanaf enige toevoer na enige afvoer

(2) Die lokale struktuurhanteringseenheid - Hierdie eenheid hanteer lokale geheueverdeling sowel as globale geheue-verdeling (met behulp van die roeteringselement).

(3) EIEU-buffer - Hierdie eenheid se taak is om die data-pakketvloei in die ring te vereffen.

(4) Die verbindingstoor - Hierdie eenheid se taak is om naamgenootdatapakkette saam te voeg en aan te stuur.



FIGUUR 4.4: Die datavloei-pyplyn

(5) Instruksiehaal-eenheid - Hierdie eenheid voeg die aangevraagde instruksie, vanuit die programgeheue, by die inkomende datapakket en stuur dit aan.

(6) Die verwerkingselement - Hierdie eenheid voer alle tipes operasies uit wat in die verwerker benodig word, en stuur die resultate na hul betrokke destinasie-adresse.

4.5 Werkverrigting analise van die datavloei-model

Die verbindingstoor is die mees kritiese pyplynstadium omdat die verbindingsproses tydrowend is. Analise word nou gedoen om die verwantskap tussen die verbindingstoordeurset en die verkeersdigthede in die verskillende pyplyngedeeltes te verkry. Die aantal verwerkingselemente vir optimale ring-deurset word ook bereken.

Die model in figuur 4.5 stel die datavloeipyplyn voor. Die volgende aannames word gemaak:

(i) Die struktuurhanteringseenheid word buite rekening gelaat aangesien dit buite die pyplyn voorkom.

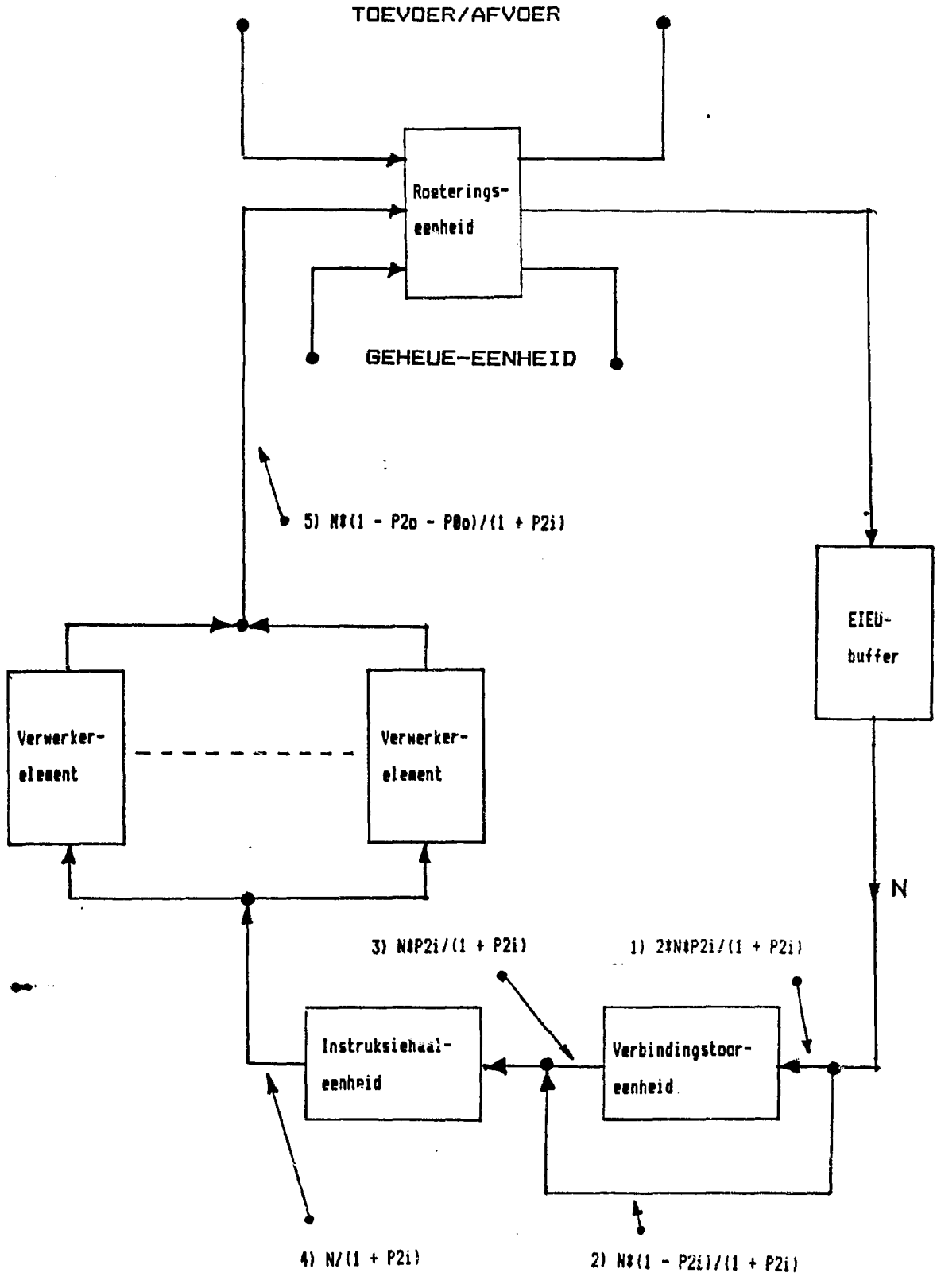
(ii) Indien 'n datapakket nie 'n maat benodig nie, vloei dit om die verbindingseenheid.

(iii) Elke datavloei-verwerkerinstruksie het 'n maksimum van twee toevoer- asook afvoerdatapakkette.

Met behulp van die volgende gegewe waarskynlikhede, word die analise uitgevoer:

a) P2i - Die waarskynlikheid van 'n instruksie met twee data-toevoere.

b) P2o - Die waarskynlikheid van 'n instruksie met twee data-



FIGUUR 4.5: Die analise model

afvoere.

c)P0o - Die waarskynlikheid van 'n instruksie met geen afvoere nie.

d)N - Die aantal afvoerdatapakkette van die EIEU-buffer per tydseenheid.

Twee datapakkete word benodig vir elke instruksie wat twee toevoere benodig. Die verhouding van datapakkette na die verbindingstoor X, tot die datapakkette verby die verbinding-stoor Y, is as volg:

$$X/Y = 2*P2i/(1 - P2i)$$

Die waarskynlikheid dat die data na die verbindingseenheid gaan, is dus:

$$X/(X + Y) = 2*P2i/(2*P2i + (1 - P2i))$$
$$= 2*P2i/(1 + P2i)$$

Om die deurset te verkry, word die waarskynlikheid eenvouding met N vermenigvuldig. Die volgende ringdeursetvergelykings kan met behulp van die gegewe waarskynlikhede (P2i,P2o en P0o) en die bostaande vergelyking afgelei word.

1) $N*2*P2i/(1 + P2i)$

Die aantal datapakkette wat om die verbindingstoor beweeg volg nou as Y/(X + Y):

2) $N*(1 - P2i)/(1 + P2i)$

Die aantal datapakkete wat vanaf die verbindingstoor afgevoer word (die helfte van die aantal toevoere):

3) $N*P2i/(1 + P2i)$

Die som van die afvoere deur en om die verbindingstoor:

4) $N/(1 + P2i)$

Die totale afvoeraantal vanaf die verwerkerelemente:

5) $N*(1 + P2o - P0o)/(1 + P2i)$

Die tempo's word in figuur 4.5 aangedui met hul ooreenstemmende nommers. As 'n voorbeeld, word die waarskynlikhede as volg geneem. (Tipiese programwaardes soos in [14]):

$P2i = 0,5$
$P2o = 0,6$
$P0o = 0,1$

Veronderstel dat die verbindingstoor 'n maksimale inset van een datapakket per tydeenheid kan hanteer. Indien dit in die vergelyking 1) gestel word, sal die roeteringselement en die EIEU-buffer deursette as volg wees:

$N = (1 + P2i)/(2*P2i)$
$= 1,5$ datapakkete/tydeenheid

Uit vergelyking 4) volg die toevoerdatapakketverkeer na die instruksiehaal-eenheid as een datapakket per tydeenheid. Indien byvoorbeeld aangeneem word dat 'n gemiddelde verwerkingselement-operasie ongeveer vyf tydeenhede duur, volg:

Insette vanaf instruksiehaal = 1 Operasie/tydeenheid

en hieruit volg:

(5 tydeenhede/operasie)/ (1 operasies/tydeenheid)
= 5 verwerkerelemente

In die voorbeeld is dit duidelik dat vyf verwerkerelemente benodig word om die pyplyn deurset te optimeer. Uit die analise, word die pyplynsiklus van die verwerker as 1 operasie per tydeenheid verkry. Indien een tydsiklus nou byvoorbeeld 200nS verteenwoordig, is die pyplyndeurset ongeveer 5 MIPS.

4.6 Die uitbreiding van die model na 'n multiverwerker

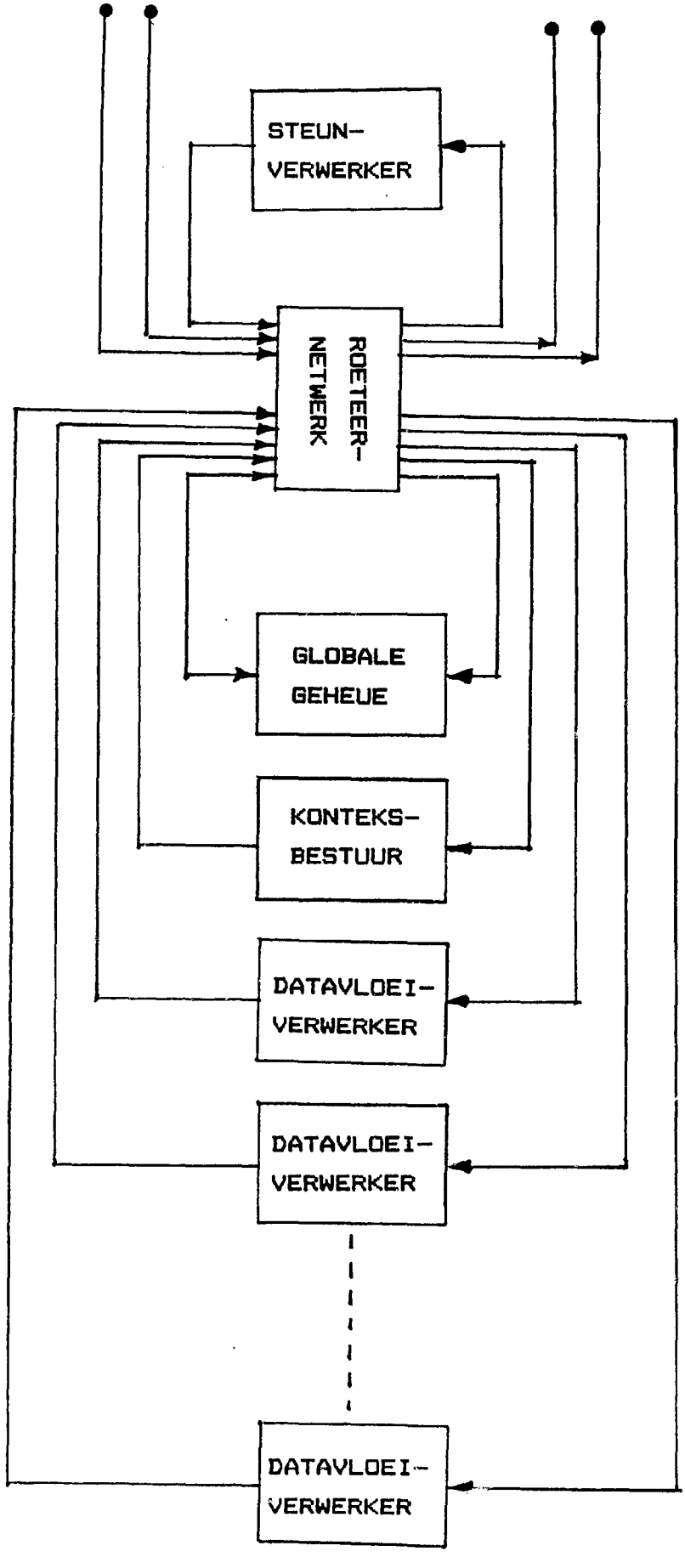
Die datavloei-verwerker beskik oor 'n modulêre argitektuur, wat uiters geskik is vir uitbreiding na 'n multiverwerker. Dit is egter belangrik dat daar altyd sentrale konteks-bestuurders is om die konteksuniekheid te waarborg. Indien die stelsel geheue-intensiewe werk en dataverdeling behartig, is dit raadsaam om 'n gemene geheue te hê, sodat die privaat-geheue van die verwerkereenhede nie met verkeer oorbelaai sal word nie.

In die figuur 4.6 is die voorgestelde datavloei-multiverwerkermodel. Die eenhede word nou beskryf:

(1) Die eenheid is of 'n steunrekenaar wat die stelsel inisialiseer en beheer, of 'n hulpverwerker wat 'n gededikeerde funksie soos skyfhantering behartig.

(2) Die roeteringsnetwerk bestaan uit 'n aantal roeterings-elemente sodat parallelle roetering moontlik is.

(3) Die geheue-eenheid stoor data (soos die struktuur-hanteringseenheid) van gemeenskaplike belang.



FIGUUR 4.6: 'n Datavloei-multiverwerker

(4) 'n Sentrale konteksbestuurder verskaf unieke kontekste aan die verskillende datavloei-verwerkereenhede.

(5) Die datavloei-verwerkereenhede wat bygevoeg of weggeneem kan word, afhangende van die stelselbehoefte.

Dit stelsel werkverrigting kan, ooreenkomstig met die vermeerdering van verwerkereenhede, verhoog word. 'n Aantal multiverwerkingstelsels word tans ondersoek, naamlik die van Arvind, Kiski en Takahashi (sien bylae I).

4.6.1 Afbeelding op die multiverwerkerstelsel

Verskeie metodes bestaan om 'n datavloei-diagram na 'n multi-verwerker-opstelling af te beeld. Die uitvoerverspreidings-kriterium kan op die konteks- of iterasienaamveld van 'n programgedeelte gebasseer word.

(i) In die ekstreme geval van verspreide programuitvoering, kan elke verwerker byvoorbeeld 'n lusiterasie of 'n gedeelte van 'n iterasie uitvoer. Die iterasiekode moet in hierdie geval na alle verwerkers versprei word.

(ii) Die konteksgedeelte van die naamveld kan vir die program-verspreiding gebruik word, maar ook hier is 'n kopie van die funksie in al die verwerkers nodig.

(iii) Die program kan uniform oor 'n aantal verwerkers versprei word. Hierdie metode het die nadeel dat lusiterasies of funksieroepe byvoorbeeld in 'n enkele verwerker sal plaasvind terwyl die program-inisialisasie in 'n ander plaasvind. Die verwerkerslas is dus meer oneweredig versprei.

Die skeduleerder wat benodig word, sal verkieslik 'n

kombinasie van al drie die bogenoemde metodes gebruik om 'n optimale werkslas-verspreiding te bewerkstellig. Die faktore wat oorweeg moet word, sluit in die aantal lusiterasies wat uitgevoer gaan word, die grootte van die funksies of iterasies, en die hoeveelheid kopiëring wat nodig is na die verskillende verwerkers. Die belangrikste oorweging is egter die databeweging in die netwerk, sodat verspreiding ten opsigte hiervan geoptimiseer moet word.

Hieruit volg dat daar 'n afspeling is tussen verspreiding van datavloei-aktiwiteite, en die lokalisering daarvan ten opsigte van roeteringskoste.

HOOFSTUK 5

5 SIMULEERING VAN DIE DATAVLOEI-VERWERKER

'n Datavloei-verwerkereenheid is gesimuleer sodat die volgende parallelle programmatuur en argitektuurkonsepte ondersoek en illustreer kon word.

(i) Die multiprogrammerings-, asook asinchrone-effek waar verskeie take gelyktydig kan verloop. (elke taak loop in sy eie konteks)

(ii) Die multiverwerker-effek van die gepyplynde datavloei-ring.

(iii) Die multiverwerker-effek van die meervoudige verwerker-elemente.

Die model is op die VAX 11/780 Rekenaar geloop en met behulp van 'n ontwikkelde masjientaal geprogrammeer om sodoende die verwerkingsproses te monitor.

5.1 Die model-implementering

Die model bestaan uit PASCAL-geprogrammeerde eenhede wat elk 'n onafhanklike proses (sien bylae A) op die VAX Rekenaar verteenwoordig. Elke eenheid het 'n toevoer/afvoermeganisme wat met behulp van die VAX bedryfstelselroepe geïmplementeer is.

Die eenheid se toevoer/afvoereenhede is in 'n sirkelvorm gerangskik om die datavloei-ring te verkry. Gedurende verwerking, vloei die datapakkette van proses na proses. (afvoer na toevoer)

Sekere aannames is gemaak om die teoretiese datavloei-verwerker-simulasiemodel te realiseer:

(i) Alle datavloei-instruksies het 'n maksimum van twee toevoer en afvoer datapakkette.

(ii) Geen perk bestaan op die grootte en aantal datapakket-datavelde nie.

(iii) Alle bewerkings word met behulp van wisselpuntgetalle gedoen. (alhoewel integers en boolese waardes virtueel ondersteun word)

Die laaste twee aannames is gemaak om die simulasie met PASCAL te vergemaklik. Die prosesse se funksies word nou afsonderlik gegee.

5.1.1 Die monitorproses

Die monitor proses word gebruik (in teenstelling met die steunrekenaar in die vorige hoofstuk se model) om die basiese toevoer/afvoerfunksies soos program inisialisasie en resultaat-afvoering te behartig.

Hierdie proses lees data vanaf die sleutelbord in en vorm 'n datapakket. Die pakket word nou na die roeteringsproses versend. Enige afvoer word vanaf die roeteringsproses gelees, en in 'n voorafbepaalde formaat op die skerm vertoon.

5.1.2 Die roeteringsproses

Die destinasie-adres van alle toevoerdatapakkette word in hierdie proses ondersoek sodat dit na die aangevraagde afvoerdestinasie geroeteer kan word.

5.1.3 Die geheueproses

Hierdie proses inkorporeer basies drie tipes funksies:

(i) Dien as struktuurhanteringseenheid
(ii) Dien as toevoer/afvoermeganisme
(iii) Dien as konteksbestuurder.

Die toevoer/afvoer en konteksbestuurfunksies kan ook in aparte interverbinde prosesse opgedeel word. Aangesien die funksies relatief eenvoudig is, was besluit om dit in hierdie eenheid te implementeer.

DIE STRUKTUUR-HANTERING:

Die struktuurhanterings-gedeelte bestaan uit 'n lineêre geheue, waarin elke geheue-posisie 'n statusveld het om die datavloei-geheuebeperkings af te dwing. 'n Leesbuffer word gebruik om uitgestelde leesoperasies tydelik te stoor totdat die betrokke leesdata wel beskikbaar is.

Omdat slegs een toewysing per geheueposisie toegelaat word, is sekere geheue-opruimingsmetodes belangrik om kontinue geheueverbruik te verseker. "Wis" en "Lees en wis" instruksies word byvoorbeeld gebruik vir hierdie doel.

Sekere sinchronisasiemetodes is in die proses geïmplementeer sodat struktuurproduksie en verbruik (indien alle elemente van 'n spesifieke struktuur byvoorbeeld gelees of geskryf is) waargeneem kan word (sien bylae J vir 'n voorbeeld hiervan). Wanneer alle elemente in 'n struktuur gelees of geskryf is, stuur die eenheid 'n kennisgewing na die aangevraagde destinasie-adres.

Die toepassing hiervan, is die geval waar 'n funksie groot is

en heelwat struktuurbewerking bevat. Alle funksie-toevoerdata kan nou met behulp van 'n struktuur in die geheue gebuffer word, sodat die funksieloop nie onmiddellik sneller nie. Nadat al die toevoerargumente in die struktuur gestoor is, kan die sinchronisasiemetodes die spesifieke funksie nou sneller deur die struktuurargument-wyser na die funksie intreepunt te stuur.

Hierdie metode sal sekwensiële funksie-uitvoering bewerkstellig en sal tipies gebruik word indien die datavloei-verwerker se hulpbronne (byvoorbeeld geheue) beperk is. Die sinchronisasie-metodes kan ook gebruik word om geheue-opruimingsfunksies te sneller. (sien bylae H vir meer detail omtrent die sinchronisasiemetodes)

TOEVOER/AFVOER

Die toevoer/afvoer word in hierdie proses gebuffer, om die sekwensie van die data te behou. Die bostaande sinchronisasie-meganisme word gebruik om die toevoer/afvoer funksie te sneller.

Indien alle toevoerdata se struktuurelemente vir 'n gegewe taak beskikbaar is, word die argumentwyser na die betrokke program gestuur om die verwerking sodoende te sneller.

Indien alle afvoerdata in 'n gegewe struktuur beskikbaar is, word die meganisme gesneller sodat alle data nou sekwensieel na buite gestuur word. Indien die volgorde egter nie belangrik is nie, kan die datawaardes direk na buite geroeteer word (met behulp van die instruksie se destinasieveld).

KONTEKS-BESTUUR

Die proses bestuur die getalgebaseerde konteksverdeling.

Konteksaanvrae word van 'n unieke "naam" voorsien en aan die aangevraagde destinasie-adres gestuur.

5.1.4 Die verbindingstoor-proses

Die proses verbind naam- en destinasiegenoot datapakkette en stuur dit na die instruksiehaalproses. 'n Pseudo-assosiatiewe geheue word vir direkte naamgenoot adressering gebruik, aangesien soekmetodes te stadig is vir hierdie samevoegings-doel.

Die metode behels die gebruik van lineêre geheue en 'n hutsskema ("hashing"), sodat 'n huts-adres vir alle inkomende datapakkette opwek word, waarmee hy sy naamgenoot direk kan adresseer.

Inkomende datapakkette word ondersoek om te sien of dit 'n naamgenoot benodig. Indien nie, word die datapakket na die instruksiehaalproses aangestuur. Indien wel, word 'n hutsadres opgewek waarna drie moontlikhede volg naamlik:

(i) Geen data bestaan by die hutsadres nie, sodat die toevoerdatapakket tydelik in die geheue gestoor word, totdat sy naamgenoot voorkom.

(ii) Die datapakket vind sy maat by die hutsadres. Die gestoorde data word by die toevoerdata gevoeg, en na die instruksiehaalproses aangestuur. Indien die oorvloei-vlag gestel is, (wanneer 'n oorvloei reeds by hierdie adres plaasgevind het) word die datapakket gesoek in die oorvloeibuffer en na die geheue verskuif.

(iii) 'n Hutsbotsing (die datapakket by die hutsadres se konteks en iterasieveld kom nie ooreen met die toevoer datapakket nie). Die toevoer datapakket word nou in die

oorvloei-buffer gestoor. Indien die oorvloeivlag gestel is, word die oorvloeibuffer deursoek vir 'n naamgenoot. indien geen naamgenoot bestaan nie, word die datapakket in 'n vakante oorvloeibuffer posisie gestoor.

Die proses werk stadiger indien die geheue vol is, aangesien baie hutsbotsings en dus oorvloeie sal voorkom. Die oorvloeibuffer moet in so 'n geval elke keer sekwensieel deursoek word vir die naamgenoot.

5.1.5 Die instruksiehaalproses

Die eenheid voeg 'n instruksie (die operasie-gedeelte) by die toevoerdatapaar en stuur dit na die verwerkerselemente.

Die masjientaal van 'n gegewe algoritme word gedurende inisialisasie vanuit 'n programlêer ingelees en in die proses se geheue gestoor. Direkte adresseringmetodes word gebruik om die aangevraagde instruksie te bekom. Dit stem ooreen met die operasie-aktivering van datavloei-diagramme.

Indien 'n inkomende datapakket na 'n instruksie wat 'n konstante waarde bevat verwys, word die konstante saam met die inkomende datapakket gevoeg en aangestuur.

Dinamiese konstante-invoeging is moontlik in die geval waar konstante waardes slegs gedurende looptyd bekend word. (byvoorbeeld N datawaardes) Die inkomende data word in die verwysde instruksie se "konstante-veld" gestoor.

5.1.6 Die verwerkingsproses

Die proses ontvang 'n datapakket vanaf die instruksiehaal-proses, wat beide die betrokke datawaardes en datavloei-instruksie bevat. Alle operasies kan in hierdie proses

uitgevoer word, naamlik:

(i) Alle datapakket-konteksbewerking-operasies
(ii) Alle datavloei-beheeroperasies
(iii) Alle boolese operasies
(iv) Alle wiskundige operasies
(v) Alle datakopiërings-operasies

Die operasieresultate word nou na die aangevraagde destinasie-adresse gestuur, deur die afvoerdatapakkette na die roeterings-proses te stuur (twee moontlike destinasie-adresse).

5.2 Programmering van die simulasie-model

Geen afbeeldingshulpmiddelle is by die US se Departement van Elektries en Elektroniese Ingenieurswese beskikbaar nie (byvoorbeeld funksionele tale en kompileerders) en daarom word die algoritme met die hand na die ontwikkelde datavloei-masjientaal afvertaal. Die tegniek behels die volgende fases:

(i) Trek 'n makro-datavloeidiagram van 'n gegewe taak.

(ii) Stel die detail-datavloeidiagramme op vir die bogenoemde makroblokke.

(iii) Nommer die datavloei-operasies in die diagram.

(iv) Skryf die instruksies op 'n sekwensiële wyse neer en verbind die operasies met behulp van die afvoer-destinasie-adresse.

Die datavloei-program bestaan dus basies uit 'n aantal aanmekaargekoppelde datavloei-masjieninstruksies. Die instruksies word in 'n rekordformaat in 'n lêer gestoor. Enige program kan geloop word, indien die betrokke programlêer

gedurende die opstelfase na die instruksiehaalproses ingelees en in die geheue gestoor word. Die bylae J voorsien 'n volledige programmeringsvoorbeeld met konteks sowel iterasie-veranderinge en die implementering van 'n toevoersinchroni-sasiemeganisme.

5.2.1 'n Gesinchroniseerde funksieroep

Die funksieroep-meganisme kan op een van twee maniere bewerkstellig word:

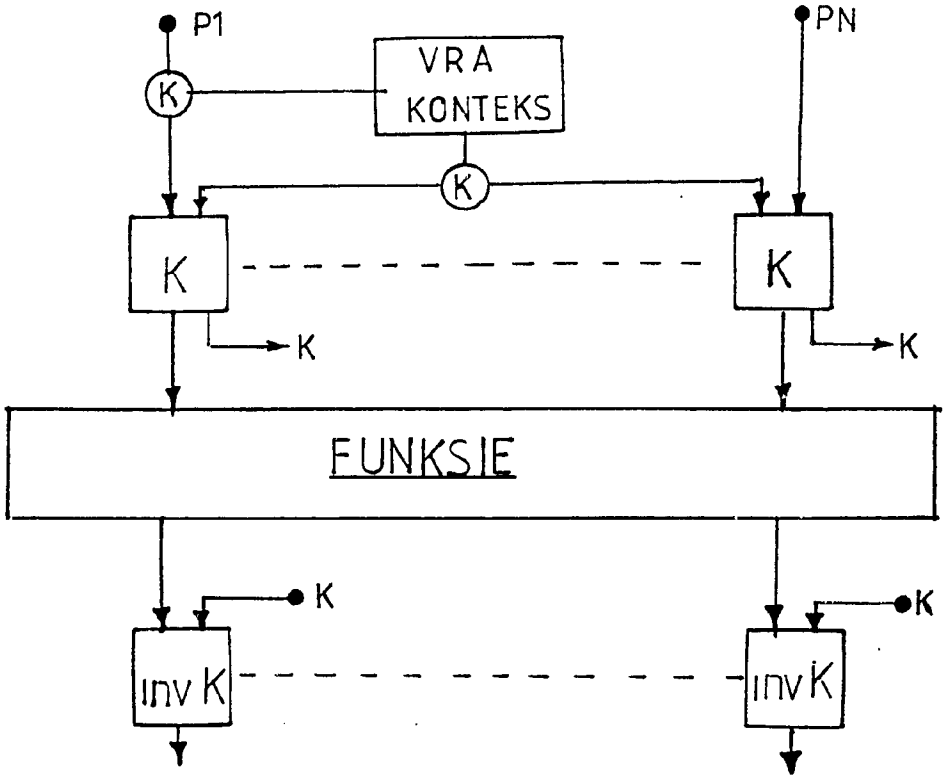
(i) Indien die funksieroepe "klein" is, (met ander woorde min kode en data-aria gebruik) en die datavloei-verwerkerhulpbronne is genoegsaam, word 'n funksieroep bewerkstellig deur slegs die konteks te verander. Die meganismes word getoon in figuur 5.1 .

(ii) Indien die funksieroep "groot" is, en die datavloei-verwerker se hulpbronne is nie genoegsaam nie, moet 'n uitvoerings-sekwensie afgedwing word om die hulpbronne eweredig in tyd te verdeel.

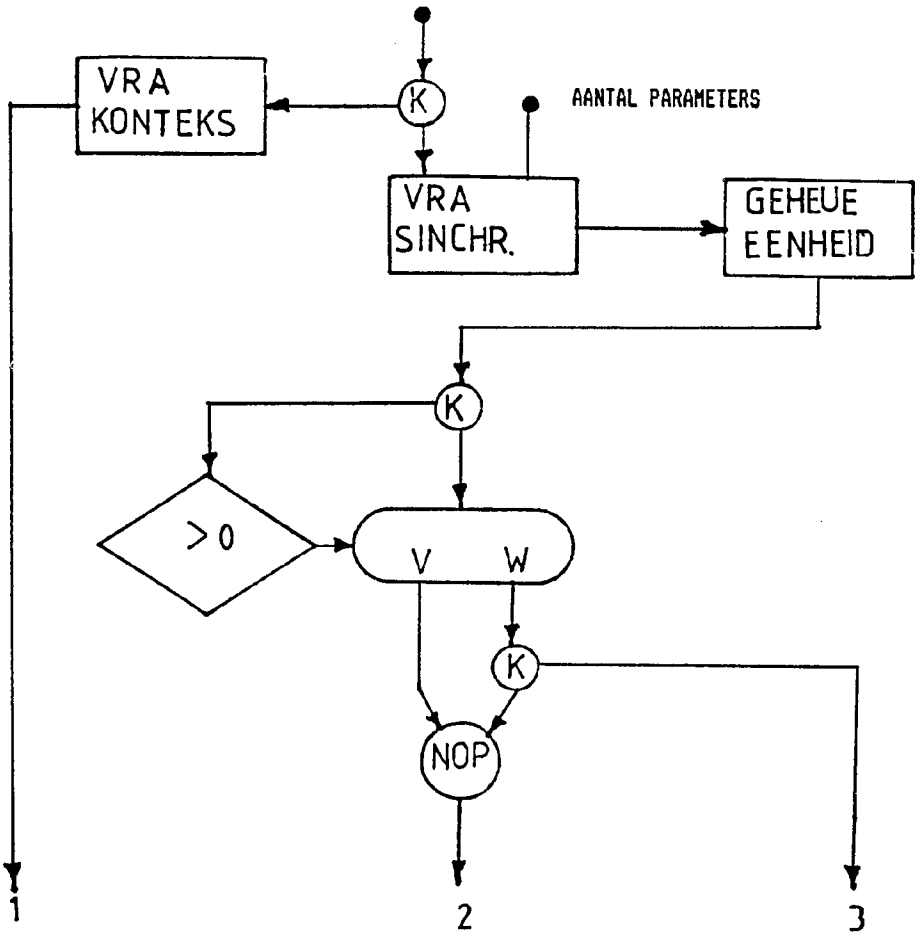
'n Tipiese voorbeeld is om lusoopvouing te verbied in die geval waar elke lusiterasie 'n groot funksie roep. Die sinchronisasie-metodes van die geheueproses word gebruik om die sekwensie van afvoering te verkry. Die meganisme word aan die hand van figuur 5.2 verduidelik, en verloop as volg:

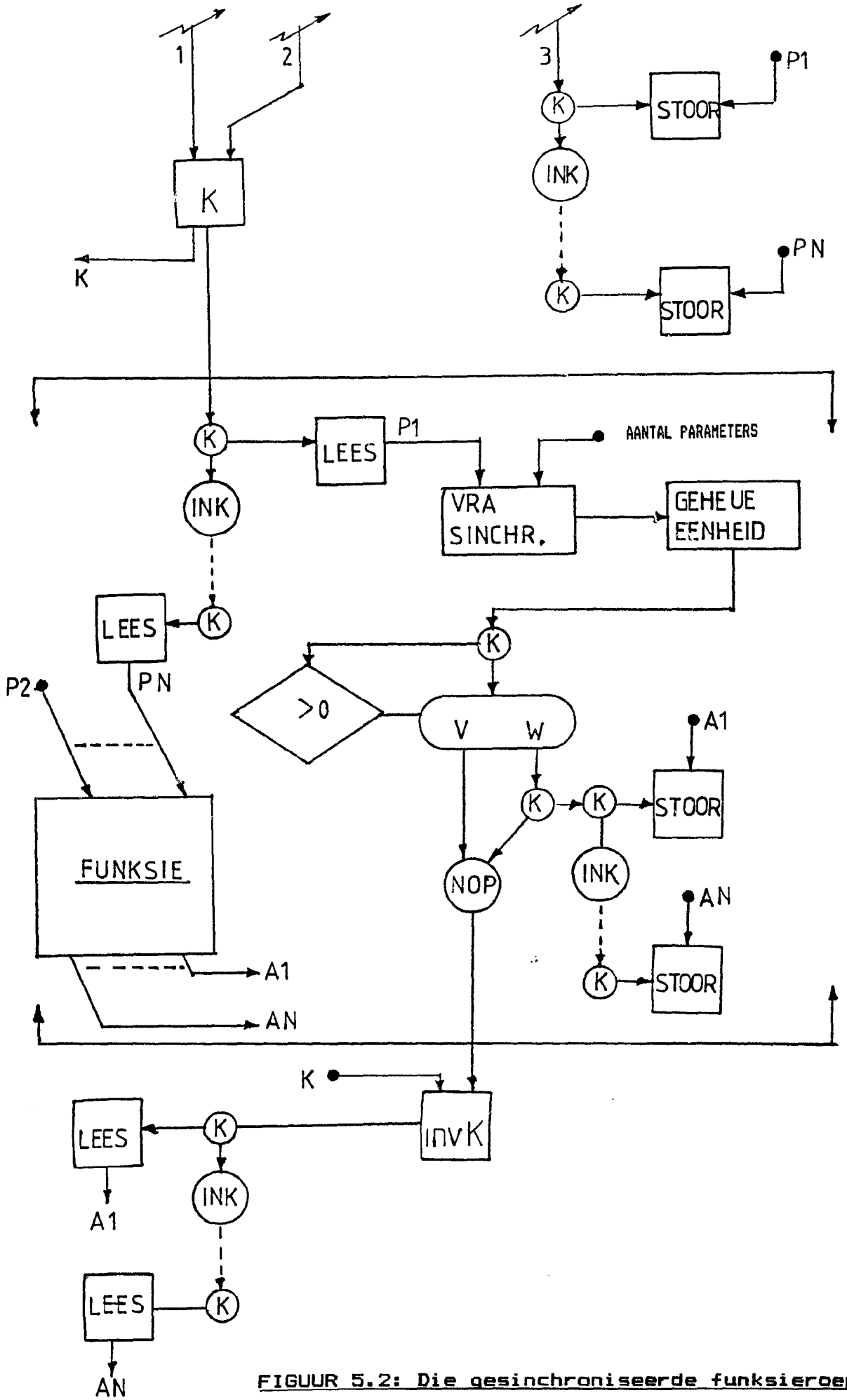
(1) Vra sinchronisasiemeganisme aan, waarna die geheueproses die toevoerdatastruktuurwyser aan die verskeie toevoer-parameter-stooroperasies voorsien.

(2) Indien al die toevoerargumente gestoor is, stuur die sinchronisasiemeganisme die struktuurwyser na 'n konteks-bewerkingoperasie. Die datapakket, met die wyser, se



FIGUUR 5.1: Die gewone funksieroep-metode





FIGUUR 5.2: Die gesinchroniseerde funksieroep-metode

konteks word nou met 'n nuwe vervang, sodat die funksie nou met hierdie pakket gesneller word.

(3) Die funksie vra nou vir 'n sinchronisasiemeganisme om op dieselfde wyse weer die resultate na die funksieroep-vlak te laat terugkeer.

(4) Die funksie verwerk die toevoerargumente en stoor die resultate in die afvoerstruktuur.

(5) Sodra alle resultate bereken en in die afvoerstruktuur gestoor is, word die struktuurwyser deur die sinchronisasie-meganisme na die konteksbewerking-operator gestuur. Die nuwe konteks word nou met die ou konteks vervang.

(6) Die afvoerstruktuur word aan al die resultaatlees-operasies in die roepvlak versprei, waarna die twee gebruikte strukture gewis kan word.

6 RESULTATE EN GEVOLGTREKKINGS

'n Totale evaluasie van 'n gegewe analise behels die volgende stappe:

a) Die opstelling van die model
b) 'n Analise van die model se vermoë
c) 'n Simulasie van die model
d) Die bou en toets van die model

In hierdie tesis is 'n datavloei-model opgestel en 'n rowwe analise van die stelselvermoë gedoen. 'n Datavloei-verwerker is gesimuleer om die implementeerbaarheid van die konsepte te demonstreer.

Indien meer eksakte resultate benodig word, kan die bou van 'n datavloei-verwerker oorweeg word. Hierdie tesis gee egter 'n oorsig van die geskiktheid van die datavloei-verwerker as spesiale verwerker. Die bevindinge word bespreek en 'n gevolgtrekking gegee.

6.1 Bespreking

Uit die voorafgaande hoofstukke is dit duidelik dat 'n datavloei-verwerker die meeste inherente argitektuur-tekortkominge, wat in konvensionele stelsels voorkom, ontduik. Die stelsel voer parallelle verwerking op 'n natuurlike wyse, sonder spesiale afbeeldings- of sinchronisasiemeganismes, uit.

GEHEUE-KONTENSIE

Die huidige stelsel-knelpunt, naamlik geheue-kontensie, kom in 'n mindere mate voor aangesien verwerkingsoperators saam met

die betrokke data vervoer word sodat geen geheue-aanvraag per operasie benodig word nie.

Die gedeelde struktuurgeheue veroorsaak nog 'n mate van geheuekontensie, maar as gevolg van die asinchrone-pyplyn-verwerking, kan vertraagde leesoperasies getolereer word. Geaktiveerde operasiebewerking kan gedurende die wagperiode voortgaan indien genoeg parallelle verwerking bestaan om die pyplyn in versadiging te hou.

VERWERKINGS-EFFEKTIWITEIT

Die verwerkings-effektiwiteit hang af van verskeie faktore: Indien die verwerkingstaak nie genoeg inherente parallelisme bevat om die pyplyn te vul nie, sal 'n verlaagde deurset die gevolg wees, sodat die stelseleffektiwiteit verlaag. Die oorhoofse koste wat aan datakopiëring spandeer word, kan slegs regverdig word indien die algoritme 'n subminimum parallel-lisme bevat.

Soortgelyk kan dinamiese iterasie- en funksiebenaming stelsel-oneffektiwiteit teweegbring indien die roepkonteks relatief min verwerking bevat, of rekursief is.

GEHEUE-VERBRUIK

Omdat data saam met die betrokke operasies en beheervelde gebuffer moet word, vind baie geheue-vermorsing, relatief tot konvensionele stelsels, plaas. Dit kan toegeskryf word aan die afwesigheid van 'n gemene geheue, soos wat in die Von Neumann stelsels voorkom [15]. Ongelukkig laat hierdie dataverdeling nie parallelle verwerking toe nie sodat, hierdie probleem onoorkombaar is.

'n Geheueverdelingsmeganisme, naamlik die struktuurhanterings-

meganisme, laat wel beperke geheueverdeling toe, maar omdat elke geheueposisie slegs een toewysing toegelaat word, word geheue-opruimingsfunksies benodig om die geheue te herwin.

Omdat geen lokale registers in 'n datavloei-verwerker beskikbaar is nie, moet alle geheue-operasies se wyser-bewerkings deur die toepassingsprogram self behartig. Hieruit volg dat 'n laer programmeringsvlak as in 'n konvensionele stelsel benodig word.

PROGRAM-ONTFOUTING EN VERIFIERING

Die datavloei-verwerker programontfoutingsproses is een-voudiger as in konvensionele stelsels, aangesien alle verwerking funksioneel plaasvind.

Alle programmeringsfoute kan vanaf die foutpunt na die deteksiepunt teruggespoor word, aangesien geen ander stelsel-bewerkings (behalwe in die geval van data-afhanklikheid) die spesifieke fouttoestand affekteer nie.

OORHOOFSE KOSTES

Soos voorheen reeds genoem is, is die oorhoofse koste verbonde aan die kopiëring en verspreiding van data, asook die dinamiese benamingsproses redelik hoog.

In die struktuurhanteringsmeganisme kom 'n groot oorhoofse koste voor indien data op 'n skryf/lees basis verdeel word. Omdat die datavloei-konsep nie geheue-hertoewysing toelaat nie, word die vorming van 'n nuwe struktuur afgeforseer indien slegs een element in die ou struktuur hertoegewys wil word. Die proses verg die kopiëring van al die onveranderde elemente van die ou struktuur na die nuwe struktuur.

Indien 'n datavloei-verwerker se hulpbronne beperk is en van die sinchronisasiemeganismes gebruik moet word om die funksie verloop sekwensieel af te dwing, verg hierdie proses ook redelike groot oorhoofse kostes, aangesien alle data-oordraging via die geheue plaasvind.

UITBREIBAARHEID

Dit is reeds in hoofstuk 4 geïllustreer dat die datavloei-verwerker, as gevolg van sy modulêre programmatuur en argitektuur, baie geskik is vir implementering in 'n multi-verwerkingsstelsel, omdat verwerking-spoedverhoging ooreenkomstig aan die aantal verwerker wat aangevoeg word, verkry kan word.

6.2 Gevolgtrekkings

'n Datavloei-verwerker het 'n breë toepassingsveld in spesiale verwerkers, aangesien dit die meeste van die bekende parallelle verwerkingsmoontlikhede kan benut. Die verskeie vlakke van parallelle verwerking wat moontlik is in die verwerker, word weereens gegee:

(i) Die argitektuur-georïenteerde fyngrein parallelisme word uitgebuit, deurdat die verwerking slegs deur die data-afhanklikheid in die program beperk word. Geen ander bekende argitektuur kan hierdie tipe verwerkings-moontlikhede so effektief benut nie, aangesien sinchronisasie op hierdie vlak in konvensionele stelsels onsinnig is.

(ii) Outomatiese lus en funksie-ontvouing laat met behulp van die dinamiese pakketbenamingsmetode, 'n makrovlak van parallelle verwerking toe. Hierdie vlak word gewoonlik ook deur konvensionele stelsels uitgebuit, maar heelwat

algoritme analise-, afbeelding- en sinchronisasie-probleme, maak hierdie proses 'n vermoeiende taak.

Die bogenoemde faktore is die parallelle programmatuur-meganismes wat in die model voorkom. Die fisiese uitbuiting hiervan word met behulp van die volgende argitektuur-tegnieke uitgebuit:

(i) Die datavloei-verwerkingsproses is eenvoudig om in aparte prosesse opgedeel te word, sodat dit veral geskik is vir pyplynverwerking. Afsonderlike eenhede werk parallel saam aan operasie dekodering en verwerking. Die verwerker is in staat, indien die pyplyn vol is, om op elke pyplyn-kloksiklus 'n bewerkingresultaat te lewer.

(ii) Omdat instruksie-dekodering oor die algemeen vinniger plaasvind as die werklike datavloei-operasie-uitvoering, volg dat meervoudige verwerkerelemente ingespan kan word om die pyplynsiklus te optimeer.

Indien 'n datavloei-verwerkerontwikkeling beoog word, is dit belangrik om die vlak van die benodigde parallelle verwerking waar te neem. Soos in konvensionele stelsels, is hier ook 'n afspeling tussen herstruktureerbaarheid en verwerkingspoed. In hierdie tesis is egter meer klem gelê op die verwerker herstruktureerbaarheid, sodat die makro- sowel as die mikrovlak van parallelle programuitvoering beskou is.

Ongelukkig is daar geen hoëvlak simulasietale op die steunrekenaar (die VAX) by US beskikbaar nie en daarom is die simulasie in PASCAL gedoen. PASCAL laat die programmeerder egter nie na aan die modelargitektuurvlak toe nie en daarom is geen werkverrigtingsmetings gepoog nie. Soos voorheen genoem is, is slegs konsepte toegepas en geïllustreer. Die simulasie was te grof.

6.3 Fisiese implementering van die model

Indien 'n fisiese implementering van die model beoog word, is dit belangrik om faktore soos koste, herstruktureerbaarheid, spoed en kompleksiteit in ag te neem. Die afspeling is egter tussen spoed en herkonstruktureerbaarheid in die gegewe spesiale verwerker toepasing.

'n Kombinasie van die volgende tegnieke sal tipies in die konstruksie van 'n hoëspoedverwerker gebruik word.

(i) Diskrete logikabane vir hoëspoedbeheer.

(ii) Skuifregister-tegnieke vir 'n hoër vlak van hoëspoed-beheer.

(iii) Bissnit-tegnieke vir die meer herkonstruktureerbare en meer komplekse beheergedeeltes.

Die eenvoudige eenhede van die model, byvoorbeeld die EIEU-buffer, die roeteringselement en die instruksiehaaleenheid, sal tipies met logika geïmplementeer word, aangesien die funksies redelik eenvoudig is en hoëspoed 'n voordeel sal wees.

Die mees komplekse en kritiese pyplyngedeelte, is die verbindingstoor. Hierdie eenheid sal tipies met 'n kombinasie van logika- en skuifregistertegnieke geïmplementeer word om die hoogste spoed moontlik te verkry.

Die mees komplekse asook die grootste getal operasies word in die verwerkingselemente verwerk. Hierdie eenhede word tipies met bissnittegnieke geïmplementeer om aan die herstruktureer-baarheidsvereistes (om verskillende operasies te kan uitvoer)

te voldoen. Verskeie implementeringsprojekte is reeds by heelparty akademiese inrigtings aangepak (sien bylae I).

6.4 Aanbevelings

IN DIE STELSEL

Twee tegnieke is moontlik indien die werkverrigting in die stelsel, met die oog op fisiese implementering, verbeter moet word:

(i) Die operasie-uitvoering word verhoog deur die ringdeurset te verhoog. Die kritiese pyplyngedeeltes, soos die verbindingstoor kan met behulp van beter tegnieke, byvoorbeeld die gebruik van beter hutsskemas (vir minder botsings) of geheuestruktuur [16], bespoedig word. Die optimale aantal verwerker-elemente is wenslik.

(ii) Die ander benadering, volgens moderne tendense, is die definiëring van kragtige instruksies waardeur meer verwerking deur 'n kleiner aantal instruksie verkry word. Die effek daarvan op die datavloei-verwerker is dat die aantal pyplynsiklusse vir 'n gegewe taak verminder, om 'n verlaagde verwerkingtyd tot gevolg te hê. In die gesimuleerde model kom baie kopieeroperasies voor, aangesien 'n operasie slegs twee destinasie-adresse mag hê. Indien instruksies gemodifiseer word om enige aantal destinasie-adresse te bevat, (met behulp van 'n aantal gekoppelde instruksies wat gelyktydig na die verwerkers-elemente oorgedra kan word) sal die aantal ringdeursette aansienlik verminder en programuitvoering bespoedig word.

IN DIE ALGEMEEN

Datavloei-konsepte kan van groot nut wees op 'n hoër vlak van

abstraksie, byvoorbeeld in multiverwerkeropstellings, waar enkelbordrekenaars byvoorbeeld groot operasies soos korrelasies en filtrering kan uitvoer. Die makrovlak parallelle verwerkingsmoontlikhede word op hierdie manier uitgebuit [17].

Indien 'n baie hoë spoed eenvoudige tipe *datavloei-verwerker* beoog word, is dit beter om die dinamiese pakketbenaming met die oorspronklike geheue-selkonsep te vervang. Hierdie verwerkertipe sal slegs die fyngrein parallelle verwerking in die algoritme uitbuit, wat die data-afhanklikheid toelaat.

Die aankoop, aanvraag of ontwikkeling van 'n datavloei-hoëvlaktaal moet sterk oorweeg word, indien projekte soos byvoorbeeld 'n herstruktureerbare datavloei-multiverwerker aangepak word.

BYLAE

BYLAE A

A VAX/VMS stelselroepe

Die twee VAX/VMS begrippe, onafhanklike prosesse en posbusse, word nou hier verduidelik [18].

A.1 'n Onafhanklike proses

'n Onafhanklike proses is die primêre uitvoeringseenheid op die VAX/VMS rekenaar. 'n Verdeling van verwerkertyd (tyddeling) vind onder die verskeie prosesse plaas. Elke proses is dus virtueel besig om die verwerker alleen te gebruik en vandaar die gelyktydige (parallelle) verwerkings-begrip van 'n onafhanklike proses.

Die datavloei-pyplyneenhede is elk as 'n onafhanklike proses gedefinieer om sodoende die pyplyn en meervoudige verwerker-elemente te implimenteer.

A.2 Die posbusbegrip

Die posbusmeganisme word gebruik om die pyplyneenhede se toevoer/afvoer te behartig om sodoende datapakkette in die eenrigting-ring te sirkuleer.

Die posbus word bedryf deur funksies wat data daarnatoe pos, en ander funksies wat die data daarvan lees. Indien die sender pos in die posbus laai, wag hy totdat die posbus-eienaar die pos lees. Indien die eienaar nou die pos lees, bevredig dit die sender asook die ontvanger. Indien die ontvanger reeds vroeër 'n posbus-leesoperasie beveel het, word die posbus-stuuroperasie onmiddellik bevredig.

Elke eenheid in die model, het sy eie posbus waardeur dit pos

kan ontvang. In die pyplyn ontvang elke eenheid data, verwerk dit, en stuur dit na die opvolgende eenheid se posbus.

Dit is noodsaaklik dat elke eenheid sy opvolger-eenheid se posbusadres moet verkry. Die adres is egter eers beskikbaar nadat die opvolgeenheid se posbus geskep is en daarom word verdere stelselroepe vir sinchronisasie-doeleindes gebruik.

A.3 Die VAX/VMS stelselroepe wat in die model gebruik word

Die volgende VAX/VMS stelselroepe word in elke proses gebruik om die boodskap-meganisme te implimenteer.

1) SYS$CREMBX - Hierdie roep skep 'n posbus en ken 'n adres en 'n naam daaraan toe.

2) SYS$ASSIGN - Hierdie roep verkry die adres van die gespesifiseerde posbus.

3) SYS$QIOW - Hierdie roep stuur data na of kry data vanaf 'n gespesifiseerde posbus-adres.

4) SYS$ASCEF - Die roep kry die adres van 'n vlagbank (wat 32 vlaggies bevat) wat in die rekenaar bestaan en deur die prosesse gebruik word.

5) SYS$WAITFR - Hierdie roep wag vir die gespesifiseerde vlag in die vlagbank om gestel te word.

6) SYS$SETEF - Hierdie roep stel die gespesifiseerde vlag in die vlagbank.

7) SYS$READEF - Hierdie roep lees die gespesifiseerde vlag en keer terug met die betrokke status.

A.4 Proses-inisialisasie

Om die pyplyn toevoer/afvoer-meganismes te implimenteer, moet die volgende inisialisasie in elke eenheid plaasvind:

1) Kry die adres van die gespesifiseerde vlagbank (SYS$ASCEF)

2) Skep 'n posbus (SYS$CREMBX)

3) Wag vir 'n vlag om gestel te word in die vlagbank (SYS$WAITFR)

4) Kry adres van opvolgproses se posbus (SYS$ASSIGN)

Die stelsel word geïnisialiseer deur al die prosesse gelyktydig te loop, sodat elke proses sy eie posbus skep. Alle prosesse wag dan vir 'n vlag in die vlagbank wat gestel moet word, waarna die opvolgproses se posbusadres verkry kan word. 'n Tipiese inisialisasie lyk as volg:

```
BEGIN
  rewrite(switch);
  a1 := sys$ascefc (64,'cluster2',0,0);
  a2 := sys$crembx (0,mbx_chan_1,0,0,0,0,'mbxSWITCH');
  a3 := sys$waitfr (64);
  a4 := sys$assign ('mbxHOST',mbx_chan_2,0,0);
  a5 := sys$assign ('mbxMATCH',mbx_chan_3,0,0);
  a6 := sys$assign ('mbxIMEM',mbx_chan_4,0,0);
  a7 := sys$assign ('mbxINOUT',mbx_chan_5,0,0);
```

BYLAE B

B Die stelsel datapakkettipes

Vier verskillende datapakkettipes bestaan in die datavloeiring van die gesimuleerde verwerkermodel. 'n Beskrywing van elk word nou gegee:

TIPE 1

Hierdie datapakkettipe kom na die verwerkerprosesse en voor die verbindingstoorproses voor. Die roeteer- en die geheueproses gebruik dus albei hierdie tipe pakket.

1) KONTEKS - Gee pakket se konteks
2) INDEKS - Gee pakket se iterasie-diepte
3) DESTINASIE - Gee pakket se destinasie
4) GEHEUE-INFORMASIE
    a) FUNKSIE - Geheue funksie
    b) INDEKS - geheue indeks
5) INSTRUKSIE - Volgende instruksie adres
    a) AANTAL - Die aantal datapakkete wat benodig word
    b) LINKS OF REGS - Linker of regter datapakket
    c) ADRES - Die adres
6) DATA
    a) DATATIPE - Tipe data bv.boolees,integer,wisselpunt
    b) DATAWAARDE - Die wisselpunt waarde

TIPE 2

Hierdie datapakkettipe kom voor die instruksie-haal en na die verbindingsproses voor.

1) KONTEKS - Gee pakket se konteks
2) INDEKS - Gee pakket se iterasie-diepte

3) DESTINASIE - Gee pakket se destinasie

4) GEHEUE-INFORMASIE

   a) FUNKSIE - Geheue funksie

   b) INDEKS - geheue indeks

5) INSTRUKSIE - Volgende instruksie adres

   a) AANTAL - Die aantal datapakkete wat benodig word

   b) LINKS OF REGS - Linker of regter datapakket

   c) ADRES - Die adres

6) DATA 1

   a) DATATIPE - Tipe data bv.boolees,integer,wisselpunt

   b) DATAWAARDE - Die wisselpunt waarde

7) DATA 2

   a) DATATIPE -Tipe data

   b) DATAWAARDE

TIPE 3

Hierdie datapakkettipe kom na die instruksiehaal- en voor die verwerkingsprosesse voor.

1) KONTEKS - Gee pakket se konteks

2) INDEKS - Gee pakket se iterasie-diepte

3) DESTINASIE - Gee pakket se destinasie

4) GEHEUE-INFORMASIE

   a) FUNKSIE - Geheue funksie

   b) INDEKS - geheue indeks

5) OPERASIE-TIPE - Spesifiseer die operasie-tipe

6) INSTRUKSIE 1 - Volgende instruksie adres

   a) AANTAL - Die aantal datapakkete wat benodig word

   b) LINKS OF REGS - Linker of regter datapakket

   c) ADRES - Die adres

7) INSTRUKSIE 2 - Volgende instruksie adres

   a) AANTAL - Die aantal datapakkete wat benodig word

   b) LINKS OF REGS - Linker of regter datapakket

c) ADRES - Die adres

8) DATA 1

a) DATATIPE - Tipe data bv.boolees,integer,wisselpunt

b) DATAWAARDE - Die wisselpunt waarde

9) DATA ½

a) DATATIPE -Tipe data

b) DATAWAARDE

TIPE 4

Hierdie tipe datapakket is die formaat van die datavloei-verwerker-masjieninstruksies wat tydens die opstelfase in die instruksiehaalproses se programgeheue ingelees word.

1) DESTINASIE - Gee pakket se destinasie

2) OPERASIE-TIPE - Spesifiseer die operasie-tipe

3) GEHEUE-FUNKSIE - Geheue funksie

4) INSTRUKSIE 1 - Volgende instruksie adres

a) AANTAL - Die aantal datapakkete wat benodig word

b) LINKS OF REGS - Linker of regter datapakket

c) ADRES - Die adres

5) INSTRUKSIE 2 - Volgende instruksie adres

a) AANTAL - Die aantal datapakkete wat benodig word

b) LINKS OF REGS - Linker of regter datapakket

c) ADRES - Die adres

6) KONSTANTE

a) GELDIG - Vlag dui aan of konstante geldig is

b) DATATIPE - Gee tipe bv. boolees,integer,wisselpunt

C) DATA WAARDE

TIPE 1

1	2	3	4		5			6	
			A	B	A	B	C	A	B

TIPE 2

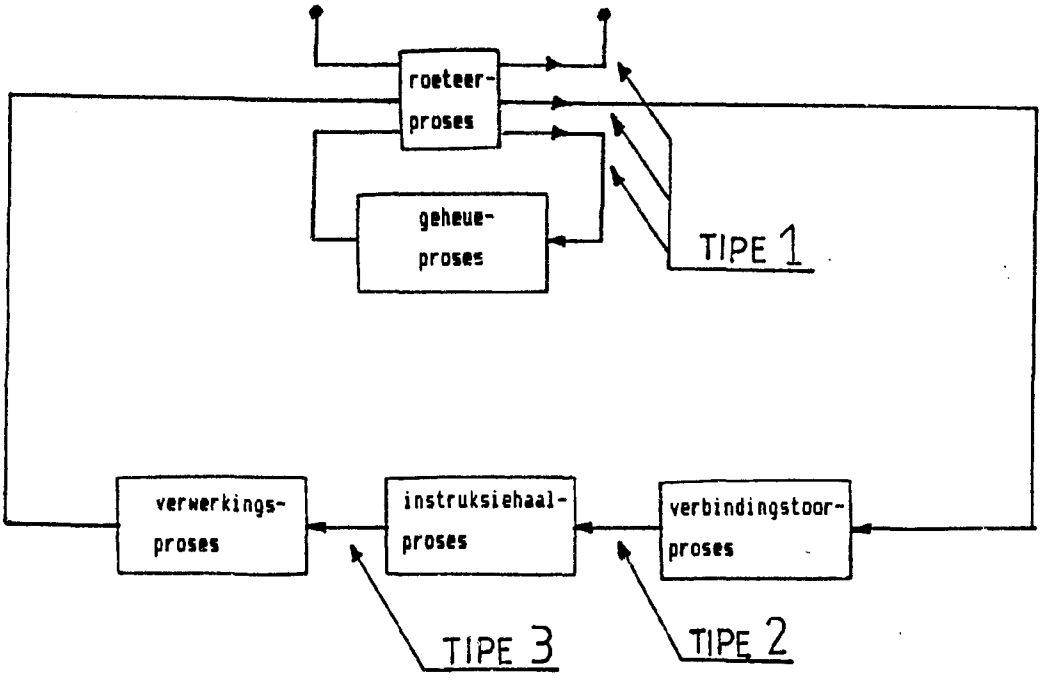
1	2	3	4		5			6		7	
			A	B	A	B	C	A	B	A	B

TIPE 3

1	2	3	4		5	6			7			8		9	
			A	B		A	B	C	A	B	C	A	B	A	B

TIPE 4

1	2	3	4			5			6		
			A	B	C	A	B	C	A	B	C



BYLAE C

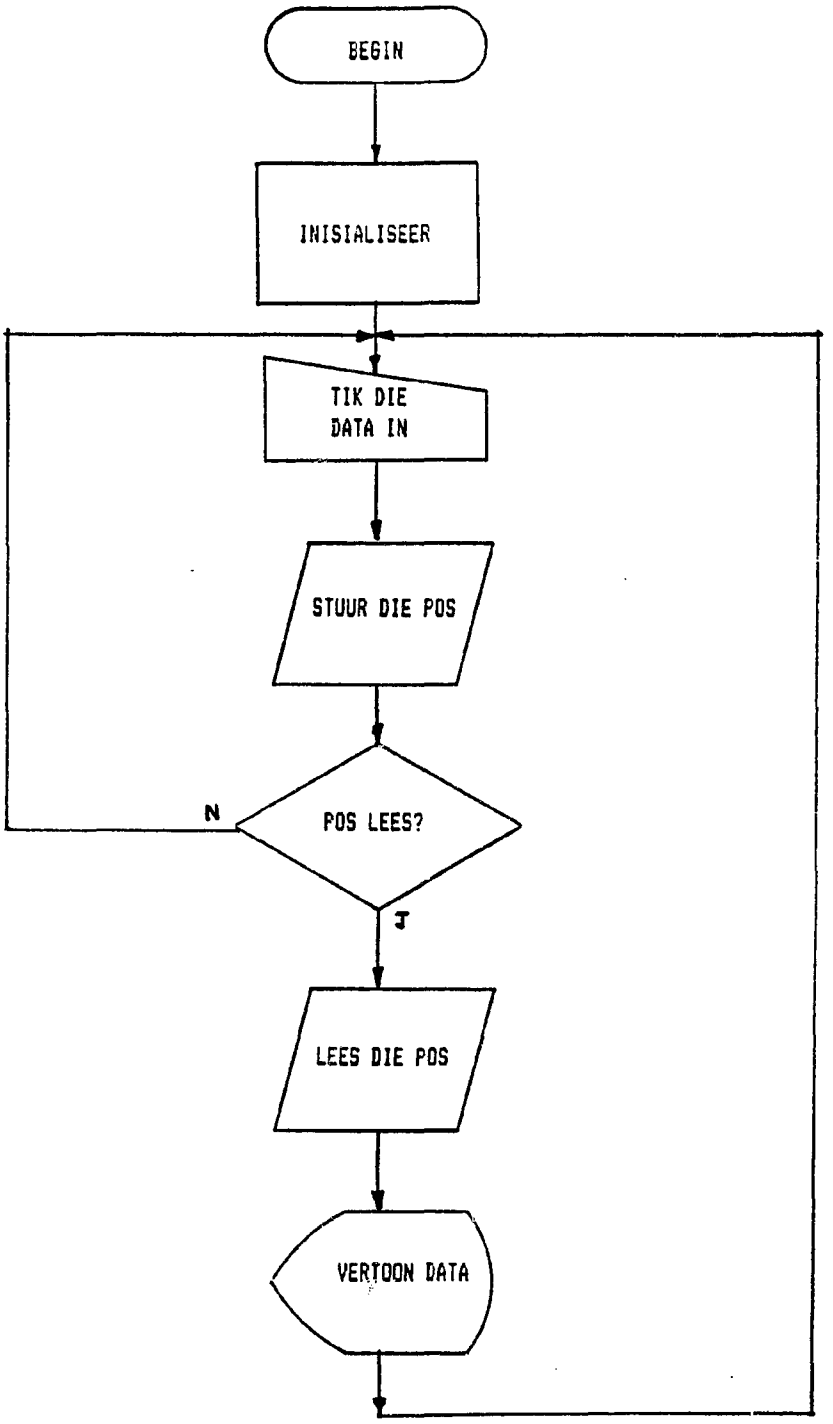
C Die monitorproses

Die monitorproses begin deur sy eie posbus te skep en stel daarna vlagnommer 64 in die vlagbank. Die ander prosesse het reeds vooraf hulle eie posbusse geskep en wag op vlagnommer 64 om die opvolgende proses se posbusadres aan te vra, sodat die hele stelsel nou in 'n ring gekoppel is. Die roeteerproses se posbusadres word dus aangevra.

Die proses gaan nou in 'n oneindige lus wat as volg verloop: Data word vanaf die terminaalskerm ingelees en na 'n data-pakket geformateer. Die datapakket word nou na die roeteer-proses se posbus gestuur vir versending na die aangevraagde destinasie. Die verbruiker word nou gevra of hy afvoerpos wil lees (indien daar wel afvoer behoort te wees). Indien wel, word die pos gelees en die aangevraagde datapakkette word op 'n voorafbepaalde formaat op die skerm vertoon.

Die toevoer- en afvoerdatapakkette is van tipe nr 1 in bylae B.

DIE MONITORPROSES



```
                (*************************************************************
                * Die MONITOR proses stuur en ontvang datapakkette na/van *
                * datavlcei-ring vir inisialisasie en afvoer                *
1     C         *************************************************************)
1     C
1     C         PROGRAM monitor (input,output);
1     C
1     C         CONST   writevblk       = 48;
1     C                 readvblk        = 49;
1     C                 stop            = 999;
1     C                 restart         = 1000;
1     C                 host            = 1;
1     C                 dfr1            = 2;
1     C                 mem             = 3;
1     C
1     C         TYPE    sub16           = 1..15;
1     C                 word_range      = 0..65535;
1     C                 string16        = PACKED ARRAY[sub16] of CHAR;
1     C                 word            = PACKED RECORD
1     C                                     w   :word_range;
1     C                                   END;
1     C                 quad_word       = PACKED RECORD
1     C                                     status,aantal  :word_range;
1     C                                     device_dat     :INTEGER;
1     C                                   END;
1     C                 message_types   = (write_switch,read_host);
1     C                 data_types      = (boolean_type,integer_type,real_type);
1     C                 nia_format      = RECORD
1     C                                     nt       :(one,two);
1     C                                     lr       :(left,right);
1     C                                     address  :INTEGER;
1     C                                   END;
1     C                 dat_format      = RECORD
1     C                                     dat_type  :data_types;
1     C                                     dat       :REAL;
1     C                                   END;
1     C                 read_format     = RECORD
1     C                                     u,i,d,func,ex :INTEGER;
1     C                                     nia           :nia_format;
1     C                                     df            :dat_format;
1     C                                   END;
1     C                 message_format  = RECORD CASE typ : message_types OF
1     C                                     write_switch  : (output :read_format);
1     C                                     read_host     : (input  :read_format);
1     C                                   END;
1     C
1     C
1     C         VAR     a1,a2,a3,a4,a5,a6,antw,
1     C                 u,i,d,func,ex,nt,lr,a,t     :INTEGER;
1     C                 dat                         :real;
1     C                 mtx_chan_1,mbx_chan_2       :word;
1     C                 message_buf                 :message_format;
1     C                 inbuf,outbuf                :read_format;
1     C                 r1,r2,r3,w                  :quad_word;
1     C
1     C
2     C         FUNCTION sys$ascefc(%IMMED   efn                          :INTEGER;
2     C                             %STDESCR name                         :PACKED ARRAY[
2     C                             %IMMED   prot,perm                    :INTEGER):INTE
2     C         FUNCTION sys$setef (%IMMED   efn                          :integer):inte
2     C         FUNCTION sys$waitfr(%IMMED   efn                          :INTEGER):INTE
2     C         FUNCTION sys$crembx(%IMMED   prmflg                       :INTEGER;
2     C                             VAR      chan                         :word;
```

```
2     0                          %IMMED   maxmsg,bufquo,promsk,acmode :INTEGER;
2     0                          %STDESCR logname                     :PACKED ARRAY[
2     0    FUNCTION sys$assign(%STDESCR devnam                       :PACKED ARRAY[
2     0                           VAR     chan                        :word;
2     0                          %IMMED   acmode,mbxnam               :INTEGER):INTE
2     0    FUNCTION sys$qiow  (%IMMED   efn                          :INTEGER;
2     0                          %IMMED   chan                        :word;
2     0                          %IMMED   func                        :INTEGER;
2     0                           VAR     iosb                        :quad_word;
2     0                          %IMMED   astadr,astprm               :INTEGER;
2     0                           VAR     p1                          :message_forma
2     0                          %IMMED   p2,p3,p4,p5,p6              :INTEGER):INTE
2     0
1     0    BEGIN
1     1              (* inisialiseer *)
1     1
1     1      a1 := sys$ascefc (64,'cluster2',0,0);
1     1      a2 := sys$crembx (0,mbx_chan_1,0,0,0,0,'mbxHOST');
1     1      a3 := sys$setef  (64);
1     1      a4 := sys$assign ('mbxSWITCH',mbx_chan_2,0,0);
1     1
1     1      REPEAT
1     2               (* Lees data vanaf die sleutelbord in na die proses *)
1     2
1     2        writeln('u,i,d,f,n,l,a,d');
1     2        readln ( u,i,d,func,nt,lr,a,dat);
1     2        outbuf.u     := u;
1     2        outbuf.i     := i;
1     2        outbuf.d     := d;
1     2        outbuf.func := func;
1     2        outbuf.nia.address := a;
1     2        outbuf.df.dat_type := real_type;
1     2        outbuf.df.dat := dat;
1     2        if (nt = 0)
1     2          then outbuf.nia.nt := one
1     2          else outbuf.nia.nt := two;
1     2        if (lr = 0)
1     2          then outbuf.nia.lr := left
1     2          else outbuf.nia.lr := right;
1     2
1     2        message_buf.typ := write_switch;
1     2        message_buf.output := outbuf;
1     2        a1:= sys$qiow(0,mbx_chan_2,writevblk,w,0,0,message_buf,32,0,0,0,0);
1     2
1     2                     (* skryf data uit na die skerm *)
1     2
1     2        repeat
1     3        write('Wil jy n antwoord he ? (ja = 0):');readln(antw);
1     3        if (antw = 0)
1     3          then begin
1     4
1     4             message_buf.typ := read_host;
1     4             a1:= sys$qiow(0,mbx_chan_1,readvblk,r1,0,0,message_buf,32,0,0,0
1     4             inbuf := message_buf.input;
1     4
1     4             if (inbuf.nia.nt = one)
1     4               then nt := 0
1     4               else nt := 1;
1     4             if (inbuf.nia.lr = left)
1     4               then lr := 0
1     4               else lr := 1;
1     4
1     4             u := inbuf.u;
```

```
1    4            i := inbuf.i;
1    4            d := inbuf.d;
1    4            func := inbuf.func;
1    4            ex := inbuf.ex;
1    4            a := inbuf.nia.address;
1    4            dat := inbuf.df.dat;
1    4            writeln(u:5,i:5,func:5,ex:5,nt:5,lr:5,a:5,dat:8:3);
1    4          end;
1    3        until (antw > 0);
1    2      UNTIL false;
1    1    END.
```

BYLAE D

D Die roeteerproses

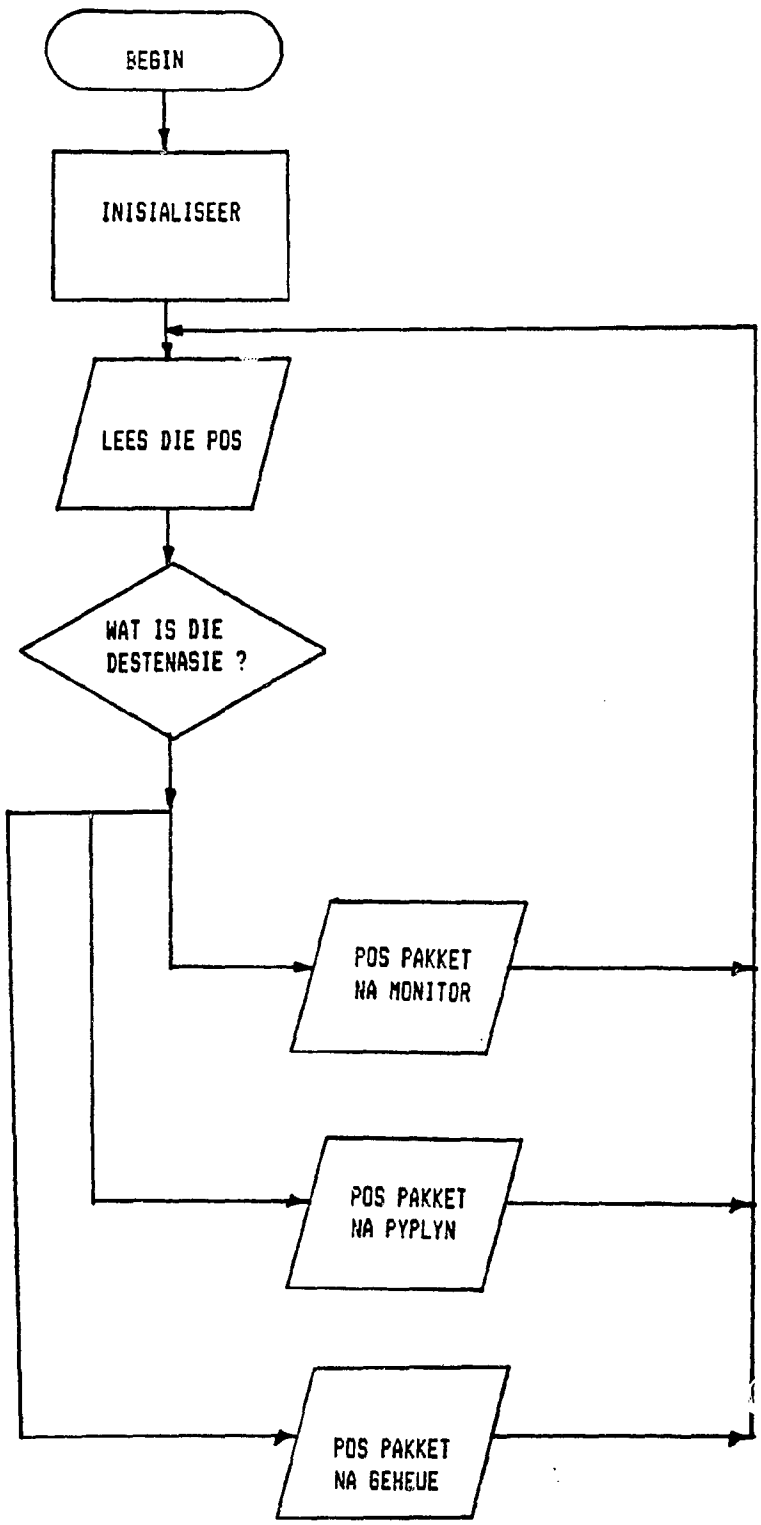
Die proses skep sy eie posbus en wag daarna vir vlagnommer 64 om die adresse van die drie opvolgende prosesse se posbusse te verkry.

Die eenheid gaan in 'n oneindige lus wat as volg verloop: Die toevoerdatapakket (pos) se destinasieadres word ondersoek en na die ooreenkomstige posbusadres geroeteer. Drie moontlike destinasies bestaan:

1) Die monitor (na buite)
2) Die struktuurhanterings-geheue
3) Die datavloei-pyplyn

Die toevoer en afvoer datapakkette is van tipe 1 in bylae B.

DIE ROETEERPROSES



```
                (*********************************************
                * Die skakelrproses skakel die verskillende  *
                * toevoercatapakkette na die coreenstemmende *
                * afvoerdestenasie-adres                     *
1       C       *********************************************)
1       C
1       C       PROGRAM switch (switch);
1       C
1       C       LABEL   999;
1       C
1       C       CONST   writevblk           = 48;
1       C               readvblk            = 49;
1       C               stop                = 999;
1       C               restart             = 1000;
1       C               host                = 1;
1       C               dfr1                = 2;
1       C               mem                 = 3;
1       C
1       C       TYPE    sub16               = 1..15;
1       C               word_range          = 0..65535;
1       C               string16            = PACKED ARRAY[sub16] of CHAR;
1       C               word                = PACKED RECORD
1       C                                       w   :word_range;
1       C                                     END;
1       C               quad_word           = PACKED RECORD
1       C                                       status,aantal  :word_range;
1       C                                       device_dat     :INTEGER;
1       C                                     END;
1       C               message_types       = (read_switch,write_unit);
1       C               data_types          = (boolean_type,integer_type,real_type);
1       C               nia_format          = RECORD
1       C                                       nt       :(one,two);
1       C                                       lr       :(left,right);
1       C                                       address  :INTEGER;
1       C                                     END;
1       C               dat_format          = RECORD
1       C                                       dat_type  :data_types;
1       C                                       dat       :REAL;
1       C                                     END;
1       C               read_format         = RECORD
1       C                                       u,i,d,func,ex :INTEGER;
1       C                                       nia           :nia_format;
1       C                                       df            :dat_format;
1       C                                     END;
1       C               message_format      = RECORD CASE typ:message_types OF
1       C                                       read_switch   :(input  :read_format);
1       C                                       write_unit    :(output :read_format);
1       C                                     END;
1       C
1       C       VAR     a1,a2,a3,a4,a5,a6,a7,i    :INTEGER;
1       C               doomsday                  :BOOLEAN;
1       C               switch                    :TEXT;
1       C               mbx_chan_1,mbx_chan_2,
1       C               mbx_chan_3,mbx_chan_4,
1       C               mbx_chan_5                :word;
1       C               r,w                       :quad_word;
1       C               irbuf                     :read_format;
1       C               message_buf               :message_format;
1       C
1       C       VALUE   doomsday := false;
1       C
1       C   %INCLUDE '[NJL.VOORB]SYS.DEC'
2       C   FUNCTION sys$ascefc(%IMMED   efn        :INTEGER;
```

```
2     C I                          %STDESCR name       :PACKED ARRAY[sub16] of char;
2     C I                          %IMMED   prot,perm  :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$waitfr(%IMMED   efn         :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$setef (%IMMED   efn         :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$clref (%IMMED   efn         :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$readef(%IMMED   efn         :INTEGER;
2     C I                      VAR     state       :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$crembx(%IMMED   prmflg      :INTEGER;
2     C I                      VAR     chan        :word;
2     C I                      %IMMED   maxmsg,bufquo,promsk,acmode :INTEGER;
2     C I                      %STDESCR logname     :PACKED ARRAY[sub16] of char) :I
2     C I FUNCTION sys$assign(%STDESCR devnam      :PACKED ARRAY[sub16] of char;
2     C I                      VAR     chan        :word;
2     C I                      %IMMED   acmode,mbxnam :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$qiow  (%IMMED   efn         :INTEGER;
2     C I                      %IMMED   chan        :word;
2     C I                      %IMMED   func        :INTEGER;
2     C I                      VAR     iosb        :quad_word;
2     C I                      %IMMED   astadr,astprm :INTEGER;
2     C I                      VAR     p1          :message_format;
2     C I                      %IMMED   p2,p3,p4,p5,p6 :INTEGER):INTEGER;EXTERN;
2     C
2     C
1     C   BEGIN
1     1               (* Inisialiseer *)
1     1
1     1     rewrite(switch);
1     1     a1 := sys$ascefc (64,'cluster2',0,0);
1     1     a2 := sys$crembx (0,mbx_chan_1,0,0,0,0,'mbxSWITCH');
1     1     a3 := sys$waitfr (64);
1     1
1     1     a4 := sys$assign ('mbxHOST',mbx_chan_2,0,0);
1     1     a5 := sys$assign ('mbxMATCH',mbx_chan_3,0,0);
1     1     a6 := sys$assign ('mbxIMEM',mbx_chan_4,0,0);
1     1
1     1     REPEAT
1     2             (* Lees toevoerdata *)
1     2
1     2       message_buf.typ := read_switch;
1     2       a1 := sys$qiow(0,mbx_chan_1,readvblk,r,0,0,message_buf,32,0,0,0,0);
1     2       inbuf := message_buf.input;
1     2       message_buf.typ := write_unit;
1     2       message_buf.output := inbuf;
1     2   (* Stuur posafvoerdata na ooreenstemmende destenasies *)
1     2
1     2       CASE inbuf.d  OF
1     3
1     3         host  : a1:= sys$qiow(0,mbx_chan_2,writevblk,w,0,0,message_buf,32.
1     3
1     3         dfr1  : a1:= sys$qiow(0,mbx_chan_3,writevblk,w,0,0,message_buf,32.
1     3
1     3         mem   : a1:= sys$qiow(0,mbx_chan_4,writevblk,w,0,0,message_buf,32
1     3
1     3         OTHERWISE BEGIN
1     4           writeln(switch,'o error',inbuf.i:5,inbuf.i:5,inbuf.d:5,inbuf.fu
      ss:5,inbuf.df.dat:8:3);
1     4           GOTO 999;
1     4         END;
1     3
1     3       END; {case}
1     2     UNTIL coomsday; {repeat}
1     1     999:
1     1   END.
```

BYLAE E

E Die verbindingsproses

Die proses skep sy eie posbus en kry dan die instruksiehaal-proses se posbusadres. Die geheue word nou geïnisialiseer en die proses gaan in 'n oneindige lus.

Die lusbewerking verloop as volg: Die pos word gelees en die toevoerdatapakket word ondersoek. Indien die pakket nie 'n maat benodig nie, word dit onmiddellik na die instruksiehaal-proses gestuur. Indien die pakket 'n maat benodig, word 'n hutsadres daarvoor opgewek. Vier moontlike toestande is die gevolg.

1) Die hutsgeheueadres bevat geen datapakket nie sodat die pakket nou by hierdie posisie gestoor word.

2) Die datapakketmaat bestaan by die adres. Die toevoer-datapakket word by die gestoorde datapakket gevoeg word (onttrek word vanaf die geheue) en na die instruksiehaal-proses gestuur. Indien die oorvloeivlag (die vlag wat aandui dat 'n hutbotsing voorgekom het en dat die betrokke datapakket in die oorvloeibuffer gestoor is) gestel is, word die oorvloeibuffer deursoek vir 'n data-pakket met dieselfde hutsadres (as die opgewekte een). Die datapakket word dan uit die oorvloeibuffer verwyder en in die hoofgeheue gestuur.

3) Hutsbotsing kom voor en die oorvloeivlag is ongestel. Die datapakket word in 'n oop plek in die oorvloeibuffer gestoor terwyl die oorvloeivlag by die betrokke hutsadres gestel word.

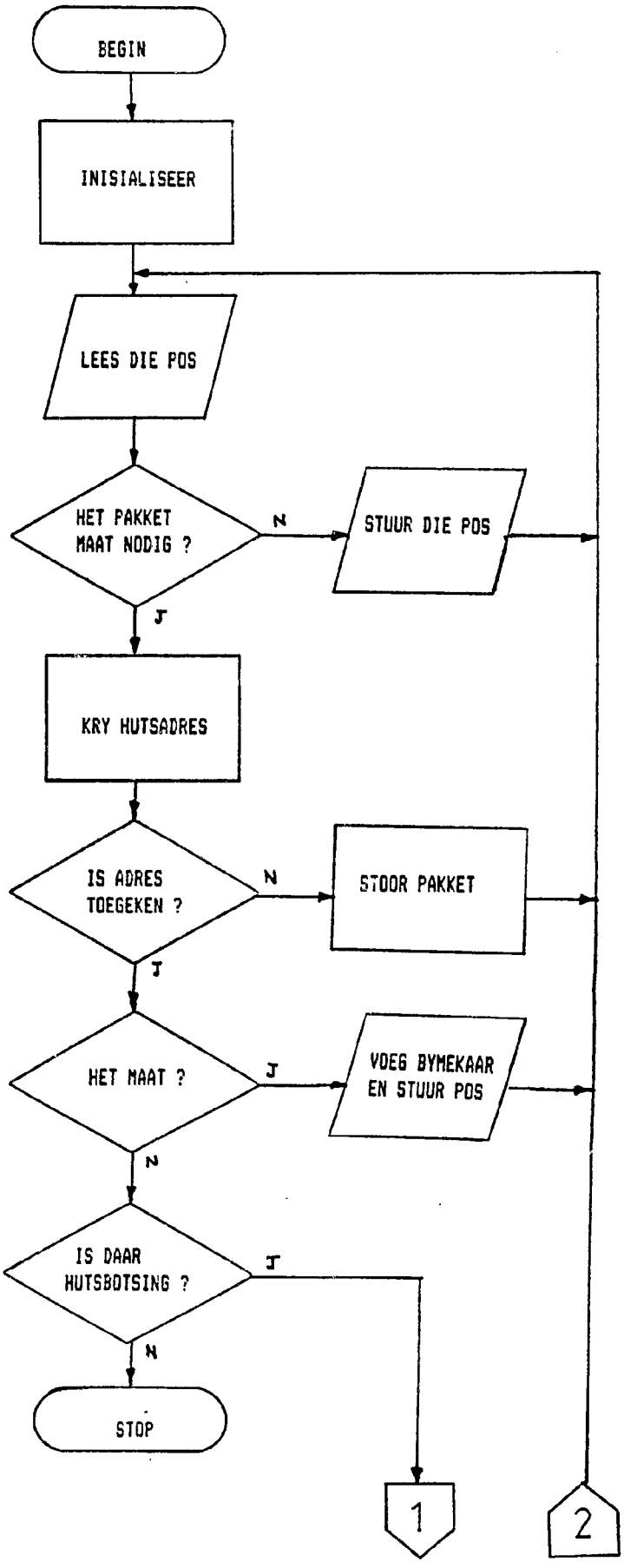
4) 'n Hutsbotsing kom voor en die oorvloeivlag is gestel.

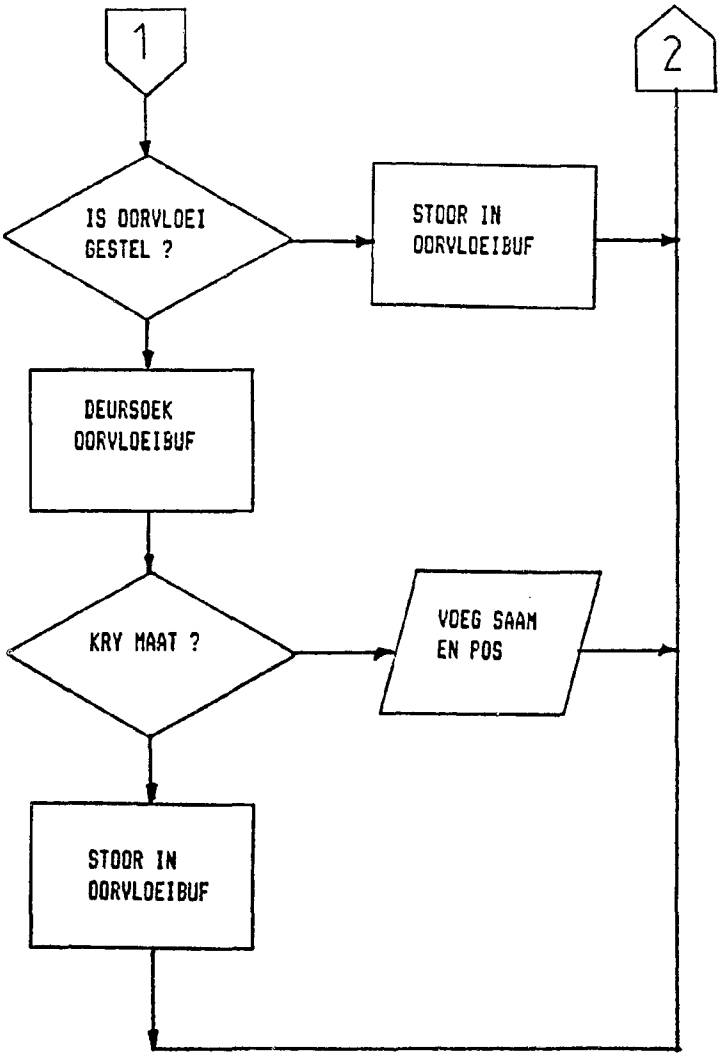
Die oorvloeibuffer word nou deursoek vir 'n maat (met dieselfde hutsadres en naam). Die maat word met die toevoerdatapakket saamgevoeg en na die instruksiehaal-proses gepos. Indien 'n maat nie bestaan nie, word die toevoerdatapakket in 'n oop posisie in die oorvloei-buffer gestoor.

Die proses simuleer 'n assosiatiewe geheue met behulp van hierdie hutsskema. 'n Eenvoudige hutsfunksie, wat die konteks-veld, die iterasieveld en die destinasieadres sommeer, word gebruik om die hutsadres op te wek.

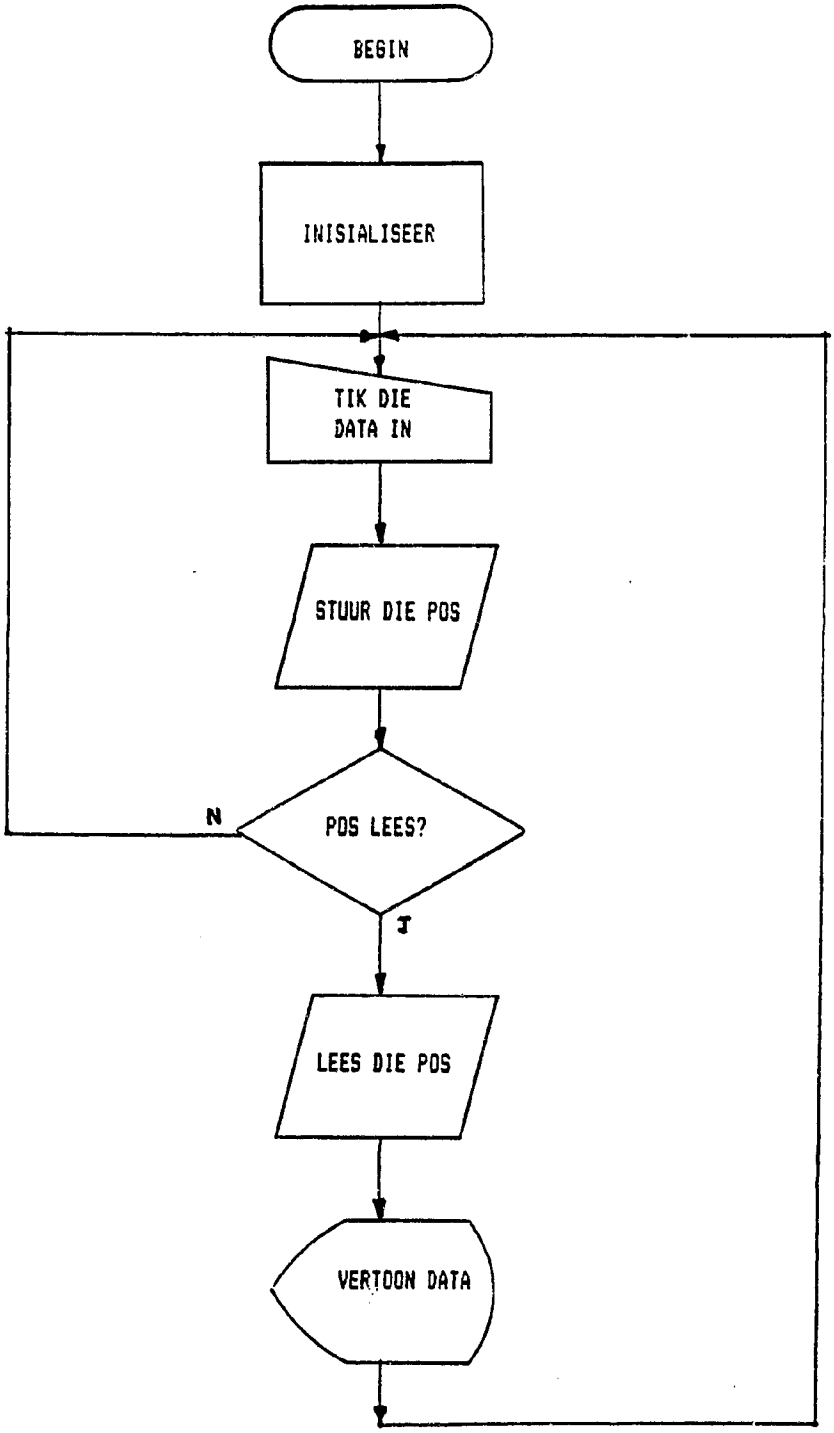
Die toevoerpakketformaat is van tipe 1 en die afvoerpakketformaat is van tipe 2 in bylae B.

DIE VERBINDINGSPROSES





DIE MONITORPROSES



```
                (*************************************************************
                * Die MONITOR proses stuur en ontvang datapakkette na/van *
                * datavlcei-ring vir inisialisasie en afvoer                *
1     C         *************************************************************)
1     C
1     C         PROGRAM monitor (input,output);
1     C
1     C         CONST    writevblk          = 48;
1     C                  readvblk           = 49;
1     C                  stop               = 999;
1     C                  restart            = 1000;
1     C                  host               = 1;
1     C                  dfr1               = 2;
1     C                  mem                = 3;
1     C
1     C         TYPE     sub16              = 1..15;
1     C                  word_range         = 0..65535;
1     C                  string16           = PACKED ARRAY[sub16] of CHAR;
1     C                  word               = PACKED RECORD
1     C                                         w    :word_range;
1     C                                       END;
1     C                  quad_word          = PACKED RECORD
1     C                                         status,aantal  :word_range;
1     C                                         device_dat     :INTEGER;
1     C                                       END;
1     C                  message_types      = (write_switch,read_host);
1     C                  data_types         = (boolean_type,integer_type,real_type);
1     C                  nia_format         = RECORD
1     C                                         nt        :(one,two);
1     C                                         lr        :(left,right);
1     C                                         address   :INTEGER;
1     C                                       END;
1     C                  dat_format         = RECORD
1     C                                         dat_type  :data_types;
1     C                                         dat       :REAL;
1     C                                       END;
1     C                  read_format        = RECORD
1     C                                         u,i,d,func,ex :INTEGER;
1     C                                         nia           :nia_format;
1     C                                         df            :dat_format;
1     C                                       END;
1     C                  message_format     = RECORD CASE typ : message_types OF
1     C                                         write_switch  : (output :read_format);
1     C                                         read_host     : (input  :read_format);
1     C                                       END;
1     C
1     C
1     C         VAR      a1,a2,a3,a4,a5,a6,antw,
1     C                  u,i,d,func,ex,nt,lr,a,t      :INTEGER;
1     C                  dat                          :real;
1     C                  mtx_chan_1,mbx_chan_2        :word;
1     C                  message_buf                  :message_format;
1     C                  inbuf,outbuf                 :read_format;
1     C                  r1,r2,r3,w                   :quad_word;
1     C
1     C
2     C         FUNCTION sys$ascefc(%IMMED   efn                    :INTEGER;
2     C                             %STDESCR name                   :PACKED ARRAY[
2     C                             %IMMED   prot,perm              :INTEGER):INTE
2     C         FUNCTION sys$setef (%IMMED   efn                    :integer):inte
2     C         FUNCTION sys$waitfr(%IMMED   efn                    :INTEGER):INTE
2     C         FUNCTION sys$crembx(%IMMED   prmflg                 :INTEGER;
2     C                             VAR      chan                   :word;
```

```
2    0                          %IMMED   maxmsg,bufquo,promsk,acmode :INTEGER;
2    0                          %STDESCR lognam                     :PACKED ARRAY[
2    0   FUNCTION sys$assign(%STDESCR devnam                        :PACKED ARRAY[
2    0                           VAR     chan                       :word;
2    0                          %IMMED   acmode,mbxnam              :INTEGER):INTE
2    0   FUNCTION sys$qiow  (%IMMED   efn                           :INTEGER;
2    0                          %IMMED   chan                       :word;
2    0                          %IMMED   func                       :INTEGER;
2    0                           VAR     iosb                       :quad_word;
2    0                          %IMMED   astadr,astprm              :INTEGER;
2    0                           VAR     p1                         :message_format
2    0                          %IMMED   p2,p3,p4,p5,p6             :INTEGER):INTE
2    0
1    0   BEGIN
1    1           (* inisialiseer *)
1    1
1    1     a1 := sys$ascefc (64,'cluster2',0,0);
1    1     a2 := sys$crembx (0,mbx_chan_1,0,0,0,0,'mbxHOST');
1    1     a3 := sys$setef  (64);
1    1     a4 := sys$assign ('mbxSWITCH',mbx_chan_2,0,0);
1    1
1    1     REPEAT
1    2           (* Lees data vanaf die sleutelbord in na die proses *)
1    2
1    2       writeln('u,i,d,f,n,l,a,d');
1    2       readln ( u,i,d,func,nt,lr,a,dat);
1    2       outbuf.u     := u;
1    2       outbuf.i     := i;
1    2       outbuf.d     := d;
1    2       outbuf.func := func;
1    2       outbuf.nia.address := a;
1    2       outbuf.df.dat_type := real_type;
1    2       outbuf.df.dat := dat;
1    2       if (nt = 0)
1    2         then outbuf.nia.nt := one
1    2         else outbuf.nia.nt := two;
1    2       if (lr = 0)
1    2         then outbuf.nia.lr := left
1    2         else outbuf.nia.lr := right;
1    2
1    2       message_buf.typ := write_switch;
1    2       message_buf.output := outbuf;
1    2       a1:= sys$qiow(0,mbx_chan_2,writevblk,w,0,0,message_buf,32,0,0,0,0);
1    2
1    2                 (* skryf data uit na die skerm *)
1    2
1    2       repeat
1    3       write('Wil jy n antwoord he ? (ja = 0):');readln(antw);
1    3       if (antw = 0)
1    3         then begin
1    4
1    4           message_buf.typ := read_host;
1    4           a1:= sys$qiow(0,mbx_chan_1,readvblk,r1,0,0,message_buf,32,0,0,0
1    4           inbuf := message_buf.input;
1    4
1    4           if (inbuf.nia.nt = one)
1    4             then nt := 0
1    4             else nt := 1;
1    4           if (inbuf.nia.lr = left)
1    4             then lr := 0
1    4             else lr := 1;
1    4
1    4           u := inbuf.u;
```

```
1    4            i := inbuf.i;
1    4            d := inbuf.d;
1    4            func := inbuf.func;
1    4            ex := inbuf.ex;
1    4            a := inbuf.nia.address;
1    4            dat := inbuf.df.dat;
1    4            writeln(u:5,i:5,func:5,ex:5,nt:5,lr:5,a:5,dat:8:3);
1    4          end;
1    3      until (antw > 0);
1    2    UNTIL false;
1    1  END.
```

BYLAE D

D Die roeteerproses

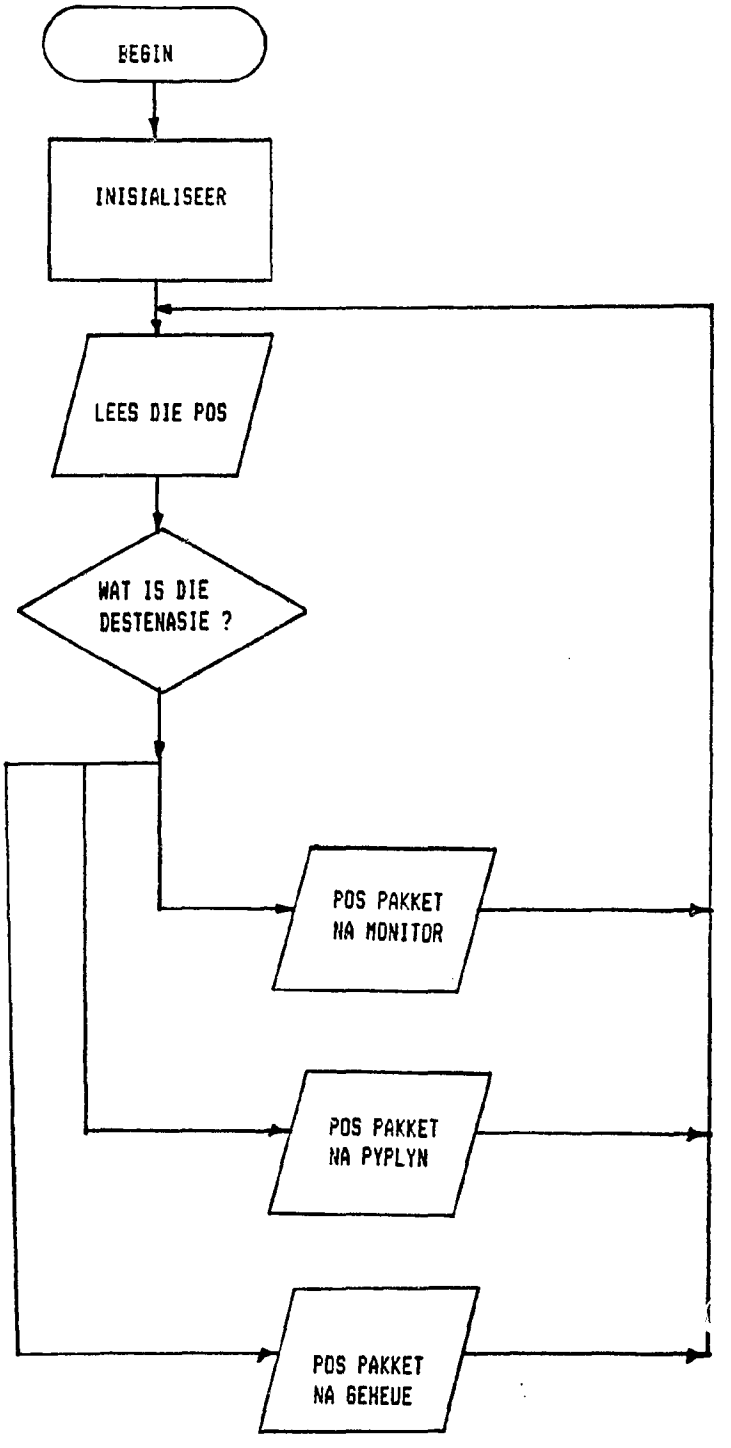
Die proses skep sy eie posbus en wag daarna vir vlagnommer 64 om die adresse van die drie opvolgende prosesse se posbusse te verkry.

Die eenheid gaan in 'n oneindige lus wat as volg verloop: Die toevoerdatapakket (pos) se destinasieadres word ondersoek en na die ooreenkomstige posbusadres geroeteer. Drie moontlike destinasies bestaan:

1) Die monitor (na buite)
2) Die struktuurhanterings-geheue
3) Die datavloei-pyplyn

Die toevoer en afvoer datapakkette is van tipe 1 in bylae B.

DIE ROETEERPROSES



```
                (*********************************************
                * Die skakelrproses skakel die verskillende  *
                * toevoercatapakkette na die coreenstemmende *
                * afvoerdestenasie-adres                     *
1     C         *********************************************)
1     C
1     C         PROGRAM switch (switch);
1     C
1     C         LABEL    999;
1     C
1     C         CONST    writevblk          = 48;
1     C                  readvblk           = 49;
1     C                  stop               = 999;
1     C                  restart            = 1000;
1     C                  host               = 1;
1     C                  dfr1               = 2;
1     C                  mem                = 3;
1     C
1     C         TYPE     sub16              = 1..15;
1     C                  word_range         = 0..65535;
1     C                  string16           = PACKED ARRAY[sub16] of CHAR;
1     C                  word               = PACKED RECORD
1     C                                         w    :word_range;
1     C                                       END;
1     C                  quad_word          = PACKED RECORD
1     C                                         status,aantal  :word_range;
1     C                                         device_dat     :INTEGER;
1     C                                       END;
1     C                  message_types      = (read_switch,write_unit);
1     C                  data_types         = (boolean_type,integer_type,real_type);
1     C                  nia_format         = RECORD
1     C                                         nt        :(one,two);
1     C                                         lr        :(left,right);
1     C                                         address   :INTEGER;
1     C                                       END;
1     C                  dat_format         = RECORD
1     C                                         dat_type  :data_types;
1     C                                         dat       :REAL;
1     C                                       END;
1     C                  read_format        = RECORD
1     C                                         u,i,d,func,ex :INTEGER;
1     C                                         nia           :nia_format;
1     C                                         df            :dat_format;
1     C                                       END;
1     C                  message_format     = RECORD CASE typ:message_types OF
1     C                                         read_switch   :(input  :read_format);
1     C                                         write_unit    :(output :read_format);
1     C                                       END;
1     C
1     C         VAR      a1,a2,a3,a4,a5,a6,a7,i    :INTEGER;
1     C                  doomsday                  :BOOLEAN;
1     C                  switch                    :TEXT;
1     C                  mbx_chan_1,mbx_chan_2,
1     C                  mbx_chan_3,mbx_chan_4,
1     C                  mbx_chan_5                :word;
1     C                  r,w                       :quad_word;
1     C                  irbuf                     :read_format;
1     C                  message_buf               :message_format;
1     C
1     C         VALUE    doomsday := false;
1     C
1     C     I %INCLUDE '[NJL.VOORB]SYS.DEC'
2     C     I FUNCTION sys$ascefc(%IMMED   efn        :INTEGER;
```

```
2     C I                          %STDESCR name       :PACKED ARRAY[sub16] of char;
2     C I                          %IMMED   prot,perm  :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$waitfr(%IMMED   efn         :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$setef (%IMMED   efn         :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$clref (%IMMED   efn         :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$readef(%IMMED   efn         :INTEGER;
2     C I                      VAR     state       :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$crembx(%IMMED   prmflg      :INTEGER;
2     C I                      VAR     chan        :word;
2     C I                      %IMMED   maxmsg,bufquo,promsk,acmode :INTEGER;
2     C I                      %STDESCR logname     :PACKED ARRAY[sub16] of char) :I
2     C I FUNCTION sys$assign(%STDESCR devnam      :PACKED ARRAY[sub16] of char;
2     C I                      VAR     chan        :word;
2     C I                      %IMMED   acmode,mbxnam :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$qiow  (%IMMED   efn         :INTEGER;
2     C I                      %IMMED   chan        :word;
2     C I                      %IMMED   func        :INTEGER;
2     C I                      VAR     iosb        :quad_word;
2     C I                      %IMMED   astadr,astprm :INTEGER;
2     C I                      VAR     p1          :message_format;
2     C I                      %IMMED   p2,p3,p4,p5,p6 :INTEGER):INTEGER;EXTERN;
2     C
2     C
1     C   BEGIN
1     1                 (* Inisialiseer *)
1     1
1     1     rewrite(switch);
1     1     a1 := sys$ascefc (64,'cluster2',0,0);
1     1     a2 := sys$crembx (0,mbx_chan_1,0,0,0,0,'mbxSWITCH');
1     1     a3 := sys$waitfr (64);
1     1
1     1     a4 := sys$assign ('mbxHOST',mbx_chan_2,0,0);
1     1     a5 := sys$assign ('mbxMATCH',mbx_chan_3,0,0);
1     1     a6 := sys$assign ('mbxIMEM',mbx_chan_4,0,0);
1     1
1     1     REPEAT
1     2             (* Lees toevoerdata *)
1     2
1     2       message_buf.typ := read_switch;
1     2       a1 := sys$qiow(0,mbx_chan_1,readvblk,r,0,0,message_buf,32,0,0,0,0);
1     2       inbuf := message_buf.input;
1     2       message_buf.typ := write_unit;
1     2       message_buf.output := inbuf;
1     2   (* Stuur posafvoerdata na ooreenstemmende destenasies *)
1     2
1     2       CASE inbuf.d  OF
1     3
1     3         host  : a1:= sys$qiow(0,mbx_chan_2,writevblk,w,0,0,message_buf,32.
1     3
1     3         dfr1  : a1:= sys$qiow(0,mbx_chan_3,writevblk,w,0,0,message_buf,32.
1     3
1     3         mem   : a1:= sys$qiow(0,mbx_chan_4,writevblk,w,0,0,message_buf,32
1     3
1     3         OTHERWISE BEGIN
1     4           writeln(switch,'o error',inbuf.i:5,inbuf.i:5,inbuf.d:5,inbuf.fu
      ss:5,inbuf.df.dat:8:3);
1     4           GOTO 999;
1     4         END;
1     3
1     3       END; {case}
1     2     UNTIL coomsday; {repeat}
1     1     999:
1     1   END.
```

BYLAE E

E Die verbindingsproses

Die proses skep sy eie posbus en kry dan die instruksiehaal-proses se posbusadres. Die geheue word nou geïnisialiseer en die proses gaan in 'n oneindige lus.

Die lusbewerking verloop as volg: Die pos word gelees en die toevoerdatapakket word ondersoek. Indien die pakket nie 'n maat benodig nie, word dit onmiddellik na die instruksiehaal-proses gestuur. Indien die pakket 'n maat benodig, word 'n hutsadres daarvoor opgewek. Vier moontlike toestande is die gevolg.

1) Die hutsgeheueadres bevat geen datapakket nie sodat die pakket nou by hierdie posisie gestoor word.

2) Die datapakketmaat bestaan by die adres. Die toevoer-datapakket word by die gestoorde datapakket gevoeg word (onttrek word vanaf die geheue) en na die instruksiehaal-proses gestuur. Indien die oorvloeivlag (die vlag wat aandui dat 'n hutbotsing voorgekom het en dat die betrokke datapakket in die oorvloeibuffer gestoor is) gestel is, word die oorvloeibuffer deursoek vir 'n data-pakket met dieselfde hutsadres (as die opgewekte een). Die datapakket word dan uit die oorvloeibuffer verwyder en in die hoofgeheue gestuur.

3) Hutsbotsing kom voor en die oorvloeivlag is ongestel. Die datapakket word in 'n oop plek in die oorvloeibuffer gestoor terwyl die oorvloeivlag by die betrokke hutsadres gestel word.

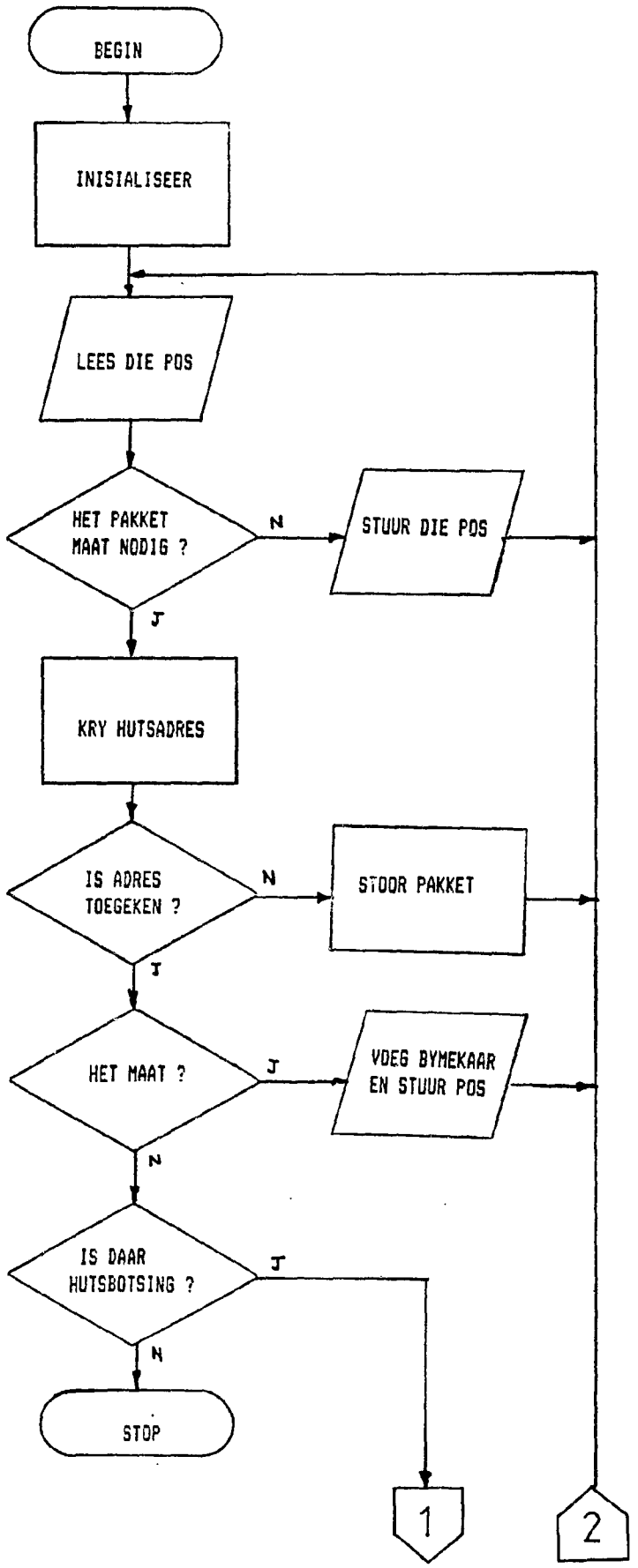
4) 'n Hutsbotsing kom voor en die oorvloeivlag is gestel.

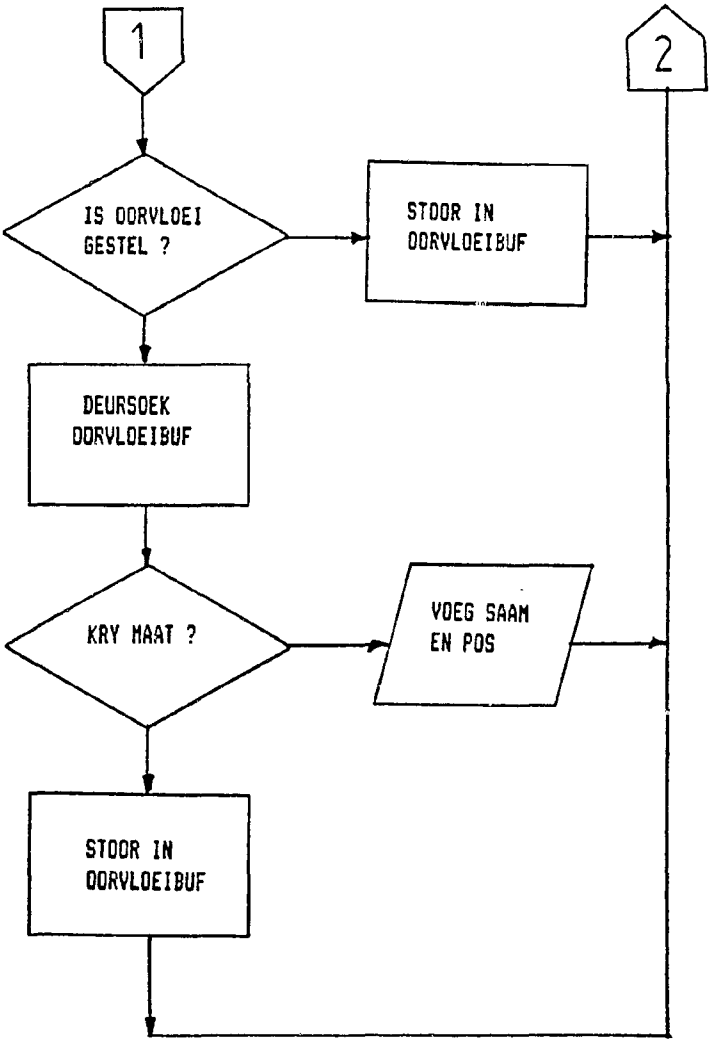
Die oorvloeibuffer word nou deursoek vir 'n maat (met dieselfde hutsadres en naam). Die maat word met die toevoerdatapakket saamgevoeg en na die instruksiehaal-proses gepos. Indien 'n maat nie bestaan nie, word die toevoerdatapakket in 'n oop posisie in die oorvloei-buffer gestoor.

Die proses simuleer 'n assosiatiewe geheue met behulp van hierdie hutsskema. 'n Eenvoudige hutsfunksie, wat die konteks-veld, die iterasieveld en die destinasieadres sommeer, word gebruik om die hutsadres op te wek.

Die toevoerpakketformaat is van tipe 1 en die afvoerpakketformaat is van tipe 2 in bylae B.

DIE VERBINDINGSPROSES





4. Die struktuurwyser wat na die geheueposisie wys waar die struktuur begin.

Fase 2: Die lees- en skryfoperasies (spesiale lees/skryf-operasies) word nou na die geheueproses gestuur. Met operasie word die getal struktuur-elemente, in die register, gedekrementeer. Indien die veld nou zero raak, word 'n erkenningspakket met 'n zero datawaarde na die aangevraagde destinasie-adres gestuur.

DIE GEHEUE-OPRUIMFUNKSIES

Twee tipes geheue-opruimfunksies kom voor. 'n Leesoperasie, wat 'n geheueposisie lees en daarna wis, word gebruik sodat leesoperasies slegs een maal na 'n struktuur moontlik is.

'n Geheuewisoperasie word gebruik, wat geheue vanaf 'n gespesifiseerde beginposisie tot by 'n gespesifiseerde eindposisie wis. Hierdie funksie kan met behulp van die sinchronisasiemeganisme se erkenningspakkette geaktiveer word om sodoende gebruikte struktuurareas te herwin.

H.2 Toevoer/afvoerfunksies

Hierdie funksies funksioneer eenders aan die sinchronisasie-meganismes van die vorige afdeling. Die erkenningspakket begrip word ook hier gebruik.

1) Toevoer - Die toevoer-register word aangevra tesame met die aantal toevoere wat verlang word. Erkenning vir die aanvraag word met behulp van 'n datapakket, wat die struktuurwyser bevat, gestuur. Elke toevoerwaarde dekrementeer die register se struktuur-aantal, totdat die

waarde zero word. Indien die waarde zero is, word 'n erkenningpakket met 'n zero datawaarde na die aangevraagde destinasie gestuur.

2) Afvoer - Hierdie proses is dieselfde as die vorige en indien alle afvoerwaardes gestoor is, word dit sekwensieel na die monitorproses gestuur.

H.3 Konteksbestuur

Twee konteksbestuurfunksies bestaan. Die konteksaanvraag-operasie inkrementeer die konteksregister en verkry dus 'n unieke konteks wat na die aangevraagde destinasie-adres verstuur word. Die konteksherstel-operasie, herstel die kontekswaarde na 'n eenheid.

H.4 Die geheueproses-instruksies

0) NOP - Geen operasie word uitgevoer nie.
1) LEES - Lees 'n geheueposisie
2) LEES EN WIS - Lees 'n geheueposisie en wis daarna
3) LEES EN DEKREMENTEER - Lees 'n geheueposisie en dekrementeer die sinchronisasiemeganisme se register.
4) LEES, WIS EN DEKREMENTEER - Lees 'n geheueposisie, dekrementeer die register, en wis die geheueposisie
5) SKRYF - Ken 'n waarde aan 'n geheueposisie toe
6) SKRYF EN DEKREMENTEER - Ken 'n waarde aan 'n geheueposisie toe en dekrementeer die sinchronisasiemeganisme se register
7) KONTEKS HERSTEL - Herstel konteks na eenheid
8) KONTEKS AANVRAAG - Vra 'n nuwe konteks aan
9) LEES SINCHRONISASIE-REGISTER AANVRAAG - Vra die register aan
10) SKRYF SINCHRONISASIE-REGISTER AANVRAAG - Vra die register aan

11) TOEVOER SINCHRONISASIE-REGISTER AANVRAAG - Vra die register aan

12) AFVOER SINCHRONISASIE-REGISTER AANVRAAG - Vra die register aan

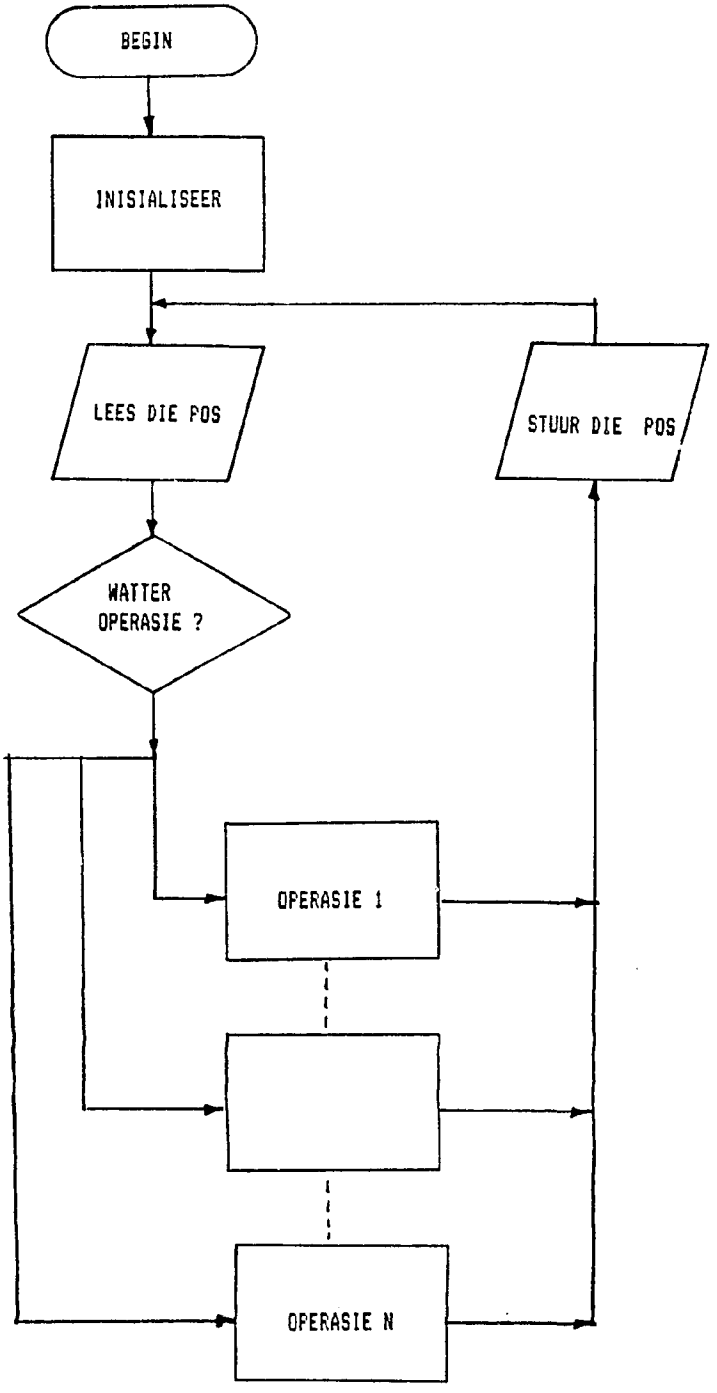
13) TOEVOER - Ken die gegewe posisie 'n datawaarde toe en dekrementeer die register

14) AFVOER - Lees die gegewe geheueposisie en dekrementeer die register

15) WIS - Wis die geheue tussen die gegewe adresse.

```
1     C     (*********************************************************
1     C      * Die verwerkingsproses doen alle naam,logies sowel  *
1     C      * as wiskundige bewerkings                           *
1     C      *********************************************************)
1     C
1     C     PROGRAM pe1 (pe1);
1     C
1     C     LABEL   0,999;
1     C
1     C     CONST   writevblk       = 48;
1     C             readvblk        = 49;
1     C             stop            = 999;
1     C             restart         = 1000;
1     C             host            = 1;
1     C             df1             = 2;
1     C             mem             = 3;
1     C
1     C
1     C     TYPE    sub16           = 1..15;
1     C             word_range      = 0..65535;
1     C             string16        = PACKED ARRAY[sub16] of CHAR;
1     C             data_types      = (boolean_type,integer_type,real_type);
1     C             message_types   = (read_pe,write_switch);
1     C             word            = PACKED RECORD
1     C                                 w   :word_range;
1     C                               END;
1     C             quad_word       = PACKED RECORD
1     C                                 status,aantal  :word_range;
1     C                                 device_dat     :INTEGER;
1     C                               END;
1     C             nia_format      = RECORD
1     C                                 nt,lr    :BOOLEAN;
1     C                                 address  :INTEGER;
1     C                               END;
1     C             dat_format      = RECORD
1     C                                 dat_type  :data_types;
1     C                                 dat       :REAL;
1     C                               END;
1     C             read_format     = RECORD
1     C                                 u,i,d,func,ex,instr  :INTEGER;
1     C                                 nia1,nia2            :nia_format;
1     C                                 df1,df2              :dat_format;
1     C                               END;
1     C             write_format    = RECORD
1     C                                 u,i,d,func,ex :INTEGER;
1     C                                 nia           :nia_format;
1     C                                 df            :dat_format;
1     C                               END;
1     C             message_format  = RECORD CASE typ  :message_types OF
1     C                                 read_pe        :(input   :read_format );
1     C                                 write_switch   :(output  :write_format);
1     C                               END;
1     C
1     C     VAR     a1,a2,a3,a4                 :INTEGER;
1     C             dcomsday                    :BOOLEAN;
1     C             mbx_chan_1,mbx_chan_2       :word;
1     C             r,w                         :quad_word;
1     C             inbuf                       :read_format;
1     C             outbuf                      :write_format;
1     C             antw1,antw2                 :dat_format;
1     C             message_buf                 :message_format;
1     C             pe1                         :text;
1     C
```

DIE VERWERKINGSPROSES



16) >= - D1 groter of gelyk aan D2

17) <= - D1 kleiner of gelyk aan D2

18) <> - D1 nie gelyk aan D2

19) = - D1 gelyk aan D2

20) NOT - Nie D1

21) OR - D1 of D2

22) AND - D1 en D2

23) MOVE - Verskuif D1 na funksieveld en D2 na D1 se veld

24) COPY - Kopieer D1 na die twee destinasie-adresse

25) SWITCH - Afhangende van die beheer gaan die een na adres 1 of adres 2

26) KONTEKS - Vervang ou konteks met nuwe. Stuur ou konteks na INV K operasie

27) I - Verstel indeksveld en stuur ou indeks na INV I operasie

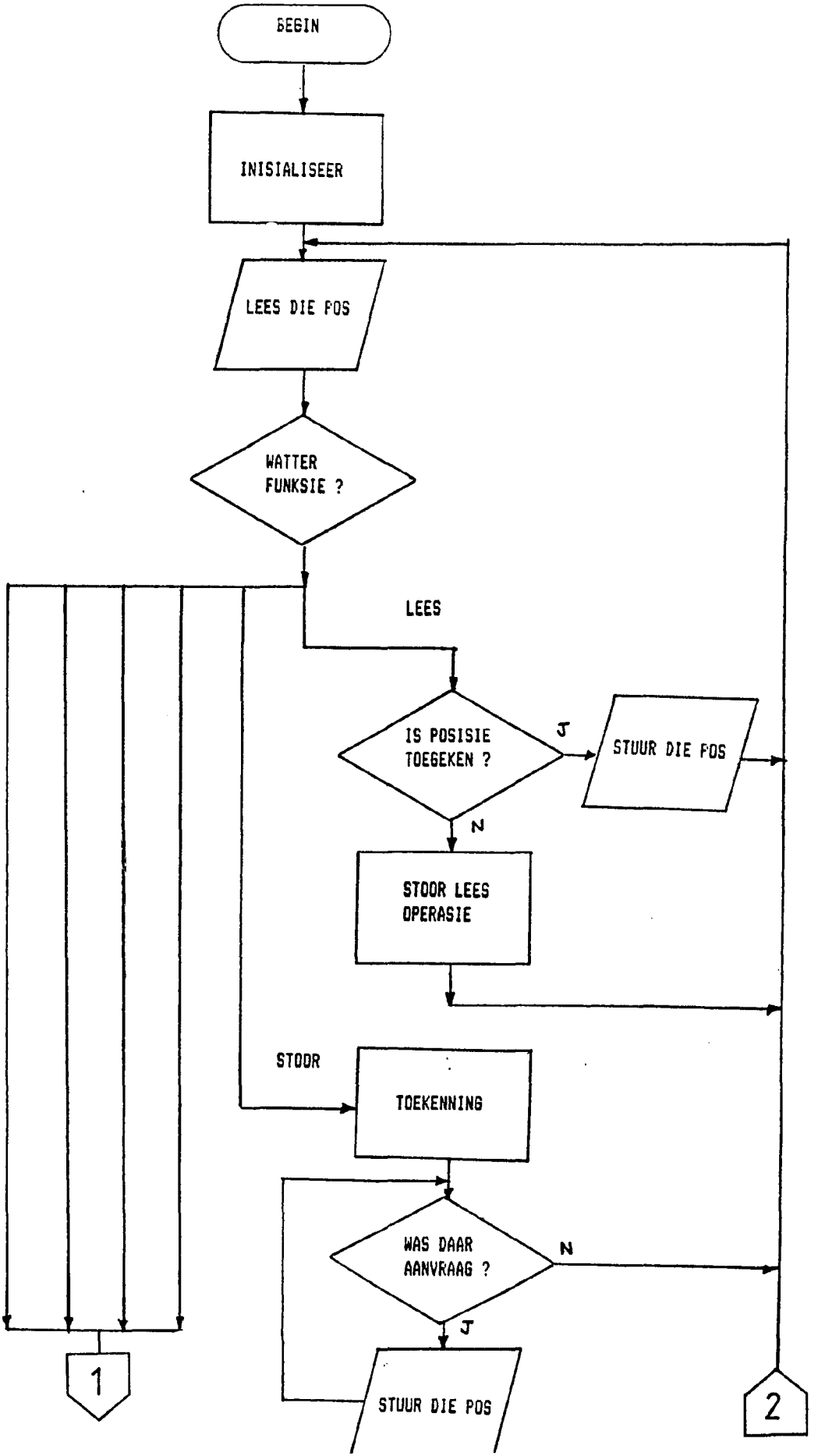
28) INK I - Inkrementeer indeksveld

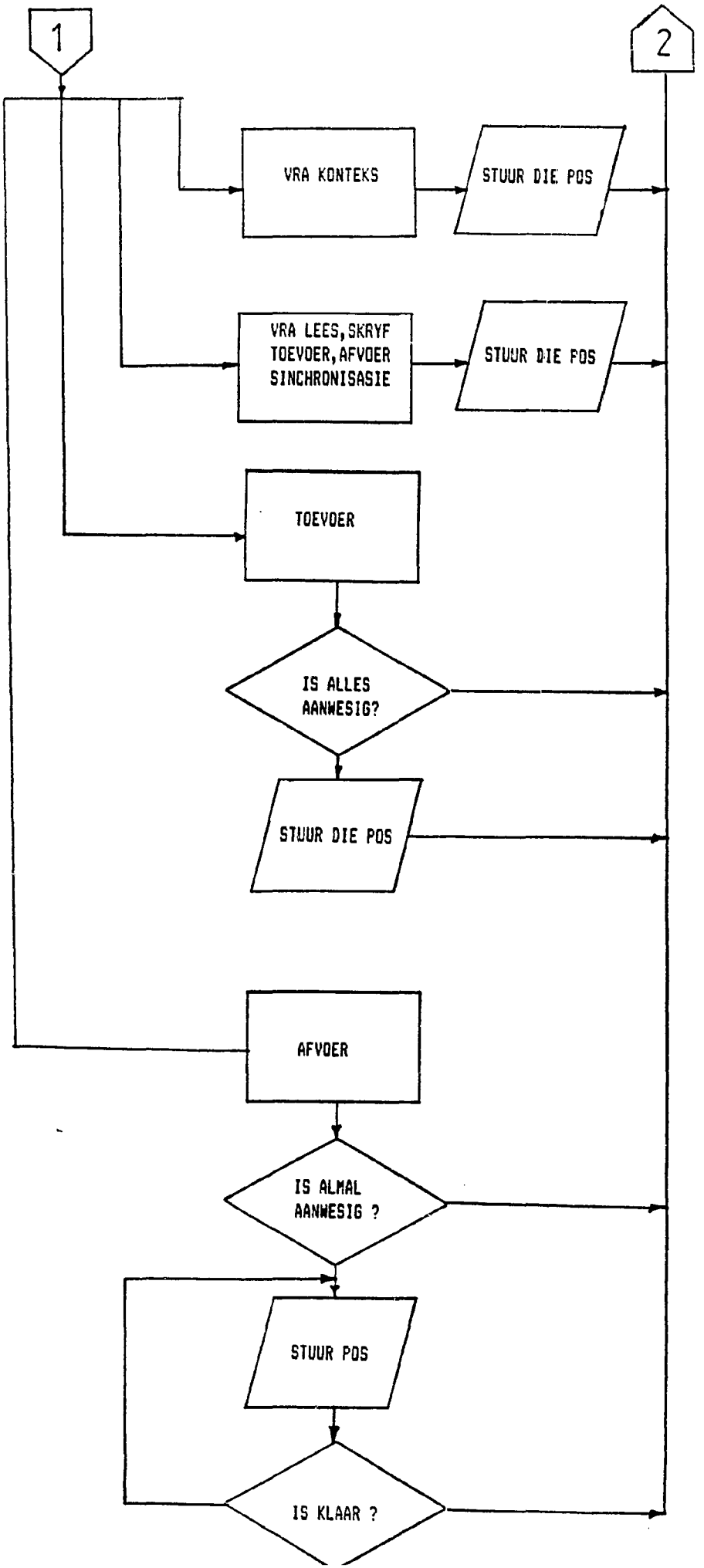
29) INV I - Herstel indeksveld na oorspronklike

30) INV K - Herstel konteksveld na oorspronklike

31) CLR I - Herstel indeksveld na eenheid

DIE GEHEUEPROSES





```
1     C     (*****************************************************
1     C      * Die geheueproses hanteer die struktuurhantering, *
1     C      * die korteksbeheer en die toevoer/afvoer.         *
1     C      *****************************************************)
1     C
1     C     PROGRAM imem (imem);
1     C
1     C     LABEL     0,999;
1     C
1     C     CONST     writevblk          = 48;
1     C               readvblk           = 49;
1     C               maks_1             = 128;
1     C               maks_2             = 32;
1     C               stop               = 999;
1     C               restart            = 10CC;
1     C               out                = 1;
1     C               dfr1               = 2;
1     C               mem                = 3;
1     C
1     C     TYPE      sub16              = 1..15;
1     C               word_range         = 0..65535;
1     C               range_1            = 1..maks_1;
1     C               range_2            = 1..maks_2;
1     C               string16           = PACKED ARRAY[sub16] of CHAR;
1     C               message_types      = (read_imem,write_switch);
1     C               status_types       = (empty,wait,written);
1     C               data_types         = (boolean_type,integer_type,real_type);
1     C               word               = PACKED RECORD
1     C                                      w    :word_range;
1     C                                    END;
1     C               quad_word          = PACKED RECORD
1     C                                      status,aantal  :word_range;
1     C                                      cevice_dat     :INTEGER;
1     C                                    END;
1     C               nia_format         = RECORD
1     C                                      nt,lr     :BOOLEAN;
1     C                                      address   :INTEGER;
1     C                                    END;
1     C               cat_format         = RECORD
1     C                                      dat_type  :data_types;
1     C                                      dat       :REAL;
1     C                                    END;
1     C               read_format        = RECORD
1     C                                      u,i,d,func,ex :INTEGER;
1     C                                      nia           :nia_format;
1     C                                      cf            :dat_format;
1     C                                    END;
1     C               buf_format         = RECORD
1     C                                      status      :status_types;
1     C                                      pionter     :INTEGER;
1     C                                      cf          :dat_format;
1     C                                    END;
1     C               ext_buf_format     = RECORD
1     C                                      valid     :BOOLEAN;
1     C                                      pionter   :INTEGER;
1     C                                      token     :read_format;
1     C                                    END;
1     C               reg_format         = RECORD
1     C                                      count,addr :INTEGER;
1     C                                      token      :read_format;
1     C                                    END;
1     C               message_format     = RECORD CASE typ:message_types OF
1     C                                      read_imem     :(input   :read_format);
```

```
1     C                                   write_switch :(output :read_format);
1     C                                 END;
1     C           buf_type          = ARRAY[range_1] OF buf_format;
1     C           ext_buf_type      = ARRAY[range_2] OF ext_buf_format;
1     C
1     C   VAR     a1,a2,a3,a4,i,j,k,konteks_u    :INTEGER;
1     C           ccomsday                       :BOOLEAN;
1     C           mbx_chan_1,mbx_chan_2          :word;
1     C           r,w                            :quad_word;
1     C           inbuf,outbuf                   :read_format;
1     C           buf                            :buf_type;
1     C           ext_buf                        :ext_buf_type;
1     C           read_reg,write_reg,
1     C           input_reg,output_reg           :reg_format;
1     C           s                              :status_types;
1     C           message_buf                    :message_format;
1     C           imem                           :text;
1     C
1     C
1     C I %INCLUDE '[NJL.VOORB]SYS.DEC'
2     C I FUNCTION sys$ascefc(%IMMED    efn        :INTEGER;
2     C I                     %STDESCR  name       :PACKED ARRAY[sub16] of char;
2     C I                     %IMMED    prot,perm  :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$waitfr(%IMMED    efn        :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$setef (%IMMED    efn        :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$clref (%IMMED    efn        :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$readef(%IMMED    efn        :INTEGER;
2     C I                      VAR      state      :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$crembx(%IMMED    prmflg     :INTEGER;
2     C I                      VAR      chan       :word;
2     C I                     %IMMED    maxmsg,bufquo,promsk,acmode :INTEGER;
2     C I                     %STDESCR  lognam     :PACKED ARRAY[sub16] of char) :I
2     C I FUNCTION sys$assign(%STDESCR  devnam     :PACKED ARRAY[sub16] of char;
2     C I                      VAR      chan       :word;
2     C I                     %IMMED    acmode,mbxnam :INTEGER):INTEGER;EXTERN;
2     C I FUNCTION sys$qiow  (%IMMED    efn        :INTEGER;
2     C I                     %IMMED    chan       :word;
2     C I                     %IMMED    func       :INTEGER;
2     C I                      VAR      iosb       :quad_word;
2     C I                     %IMMED    astadr,astprm :INTEGER;
2     C I                      VAR      p1         :message_format;
2     C I                     %IMMED    p2,p3,p4,p5,p6 :INTEGER):INTEGER;EXTERN;
2     C
2     C
2     C   PROCEDURE read_mail;
2     C
2     C   (* Hierdie prosedure lees 'n datapakket uit die toevoerposbus *)
2     C
2     C   BEGIN
2     1     message_buf.typ := reac_imem;
2     1     a1 := sys$qiow (0,mbx_chan_1,readvblk,r,0,0,message_buf,32,0,0,0,0);
2     1     inbuf := message_buf.input;
2     1   writeln(imem,'begin (u,i,f,ex,a,dat) ',inbuf.u:5,inbuf.i:5,inbuf.d:5,int
          .address:5,inbuf.df.dat:8:3);
2     1   END; {reac_mail}
2     C
2     C   PROCEDURE send_mail;
2     C
2     C   (* Hierdie prosedure lees 'n datapakket uit na die afvoerposbus *)
2     C
2     C   BEGIN
2     1     message_buf.typ := write_switch;
2     1     message_buf.input := outbuf;
```

```
2  1    a1 := sys$qiow (0,mbx_chan_2,writevblk,w,0,0,message_buf,32,0,0,0,0);
2  1  END; {send_mail}
2  0
2  0  FUNCTION search :integer;
2  0
2  0  (* hierdie funksie soek 'n ocp spasie in die ocrvloeibuffer *)
2  0
2  0  VAR     found :BOOLEAN;
2  0          z     :INTEGER;
2  0
2  0  BEGIN
2  1    z := 1;
2  1    found := false;
2  1    REPEAT
2  2      IF (ext_buf[z].valid = false)
2  2        THEN found := true
2  2        ELSE z := z + 1;
2  2      IF (z > maks_2)
2  2        THEN BEGIN
2  3          writeln (imem,'ext_buf overflow');
2  3          GOTO 999;
2  3        END;
2  2    UNTIL(found = true);
2  1    search := z;
2  1  END; {search}
2  0
2  0  PROCEDURE read_buf(token :read_format);
2  0
2  0  (*  hierdie prosedure hanteer alle geheuelese *)
2  0
2  0  VAR  i,j,k,l  :INTEGER;
2  0
2  0  BEGIN
2  1    i := inbuf.ex;
2  1    IF (i > maks_1)
2  1      THEN BEGIN
2  2        writeln (imem,'Ongeldige i in read_buf :',i);
2  2        GOTO 999;
2  2      END;
2  1
2  1    CASE buf[i].status OF
2  2
2  2    empty :BEGIN
2  3             j := search;
2  3             buf[i].pionter := j;
2  3             buf[i].status := wait;
2  3             ext_buf[j].valid := true;
2  3             ext_buf[j].token := token;
2  3             ext_buf[j].pionter := 0;
2  3           END;
2  2
2  2    wait  :BEGIN
2  3             j := buf[i].pionter;
2  3             REPEAT
2  4               k := ext_buf[j].pionter;
2  4               IF (k > maks_2)
2  4                 THEN BEGIN
2  5                   writeln (imem,'ext_buf pionter k ongeldig',k);
2  5                   GOTO 999;
2  5                 END;
2  4               IF (k = 0)
2  4                 THEN BEGIN
2  5                   l := search;
```

```
2     5                    ext_buf[j].pionter := l;
2     5                    ext_buf[l].valid := true;
2     5                    ext_buf[l].token := token;
2     5                    ext_buf[l].pionter := 0;
2     5                  END;
2     4                j := k;
2     4              UNTIL (k = 0);
2     3            END;
2     2
2     2   written :BEGIN
2     3              outbuf.u    := token.u;
2     3              outbuf.i    := token.i;
2     3              outbuf.d    := dfr1;
2     3              outbuf.func := token.func;
2     3              outbuf.nia  := token.nia;
2     3              outbuf.df   := buf[i].df;
2     3              send_mail;
2     3
2     3   (* lees en herstel lese instruksies 2,4 *)
2     3              IF (token.func = 2)OR(token.func = 4)
2     3                THEN BEGIN
2     4                  (*clr die geheue ,dus tou ander wagtende lese*)
2     4                  buf[i].status   := empty;
2     4                  buf[i].pionter  := 0;
2     4                END;
2     3
2     3   (* lees en inkrimenteer die sinchronisasieregister instruksies 3,4  *)
2     3              IF (token.func = 3)OR(token.func = 4)
2     3                THEN BEGIN
2     4                  IF (read_reg.count = 0)
2     4                    THEN BEGIN
2     5                      writeln(imem,'read_reg.count is reeds 0');
2     5                      GOTO 999;
2     5                    END;
2     4                  read_reg.count := read_reg.count - 1;
2     4                  IF (read_reg.count = 0)
2     4                    THEN BEGIN
2     5                      outbuf.u      := read_reg.token.u;
2     5                      outbuf.i      := read_reg.token.i;
2     5                      outbuf.d      := dfr1;
2     5                      outbuf.func   := read_reg.token.func;
2     5                      outbuf.ex     := round(read_reg.token.df.dat);
2     5                      outbuf.nia    := read_reg.token.nia;
2     5                      outbuf.df.dat := 0;
2     5                      send_mail;
2     5                    END;
2     4                END;
2     3            END; {written}
2     2
2     2            OTHERWISE BEGIN
2     3              writeln(imem,'token.status ongeldig in proc read_buf');
2     3              GOTO 999;
2     3            END;
2     2
2     2     END; {case}
2     1   END;  {procedure read_buf}
2     0
1     0   BEGIN
1     1            (* Inisialiseer *)
1     1
1     1     rewrite(imem);
1     1     a1 := sys$ascefc (64,'cluster2',0,0);
```

```
1     1     a2 := sys$crembx (0,mbx_chan_1,0,0,0,0,'mbxIMEM');
1     1     a3 := sys$waitfr (64);
1     1
1     1     a4 := sys$assign ('mbxSWITCH',mbx_chan_2,0,0);
1     1     writeln(imem,'ascefc,crembx,waitfr,assign');
1     1     writeln(imem,a1:6,a2:6,a3:6,a4:6);
1     1
1     1   0:
1     1
1     1     doomsday := false;
1     1     konteks_u := 0;
1     1     FOR i := 0 to maks_1 DO
1     1       BEGIN
1     2         buf[i].status := empty;
1     2         buf[i].pionter := 0;
1     2       END;
1     1     FOR i := 0 to maks_2 DO
1     1       BEGIN
1     2         ext_buf[i].valid   := false;
1     2         ext_buf[i].pionter := 0;
1     2       END;
1     1
1     1   REPEAT
1     2       read_mail;
1     2       IF (inbuf.func in [1,2,3,4])
1     2          THEN read_buf(inbuf)
1     2          ELSE CASE inbuf.func OF
1     3
1     3   {nop}
1     3                C  :;
1     3
1     3   {write,write_dec}
1     3               5,6  :BEGIN
1     4                       i := inbuf.ex;
1     4                       IF (i > maks_1)
1     4                         THEN BEGIN
1     5                           writeln (imem,'i ongeldig',i);
1     5                           GOTO 999;
1     5                         END;
1     4
1     4                       IF (buf[i].status = written)
1     4                         THEN BEGIN
1     5                           writeln(imem,'buf status reeds "written"');
1     5                           GOTO 999;
1     5                         END;
1     4
1     4                       s := buf[i].status;
1     4                       buf[i].df := inbuf.df;
1     4                       buf[i].status  := written;
1     4
1     4                       IF (s = wait)
1     4                         THEN BEGIN
1     5                           j := buf[i].pionter;
1     5                           IF (j > maks_2)OR(j = 0)
1     5                             THEN BEGIN
1     6                               writeln(imem,'ongeldige j in write',j);
1     6                               GOTO 999;
1     6                             END;
1     5                           REPEAT
1     6                             k := j;
1     6                             j := ext_buf[k].pionter;
1     6                             reac_buf(ext_buf[k].token);
1     6                             ext_buf[k].pionter := 0;
```

```
1     6                                 ext_buf[k].valid := false;
1     6                               UNTIL (j = 0);
1     5                             END;
1     4
1     4                           {inc instruction}
1     4                           IF (inbuf.func = 6)
1     4                             THEN BEGIN
1     5                               IF (write_reg.count = 0)
1     5                                 THEN BEGIN
1     6                                   writeln(imem,'write_reg.count is reeds 0');
1     6                                   GOTO 999;
1     6                                 END;
1     5                               write_reg.count := write_reg.count - 1;
1     5                               IF (write_reg.count = 0)
1     5                                 THEN BEGIN
1     6                                   WITH outbuf DO
1     6                                     BEGIN
1     7                                       u       := write_reg.token.u;
1     7                                       i       := write_reg.token.i;
1     7                                       d       := dfr1;
1     7                                       func    := write_reg.token.func;
1     7                                       ex      := 0;
1     7                                       nia     := write_reg.token.nia;
1     7                                       df.dat  := 0;
1     7                                     END;
1     6                                   send_mail;
1     6                                 END;
1     5                             END;
1     4                         END; {write,write_dec}
1     3
1     3   (* konteksbeheer *)
1     3
1     3   {konteks,herstel_konteks}
1     3              7,8    :BEGIN
1     4                       IF (inbuf.func = 7)
1     4                         THEN konteks_u := 0
1     4                         ELSE BEGIN
1     5                           konteks_u := konteks_u + 1;
1     5                           IF (konteks_u > 1000) THEN konteks_u := 1;
1     5                           WITH outbuf DO
1     5                             BEGIN
1     6                               u    := inbuf.u;
1     6                               i    := inbuf.i;
1     6                               d    := dfr1;
1     6                               func := inbuf.func;
1     6                               ex   := 0;
1     6                               nia  := inbuf.nia;
1     6                               df.cat := konteks_u;
1     6                             END;
1     5                           send_mail;
1     5                         END;
1     4                       END; {konteks,herstel_konteks}
1     3
1     3   (* vra lees sinchronisasie register aan *)
1     3
1     3   {req_read_reg}
1     3                9     :BEGIN
1     4                       IF (read_reg.count > 0)
1     4                         THEN BEGIN
1     5                           writeln(imem,'read reg is nog nie leeg nie');
1     5                           GOTO 999;
1     5                         END;
1     4
```

```
1     4                         WITH read_reg DO
1     4                           BEGIN
1     5                             count := inbuf.ex;
1     5                             addr  := round(inbuf.df.dat);
1     5                             token := inbuf;
1     5                           END;
1     4
1     4                         WITH outbuf DO
1     4                           BEGIN
1     5                             u       := inbuf.u;
1     5                             i       := inbuf.i;
1     5                             d       := dfr1;
1     5                             func    := inbuf.func;
1     5                             ex      := 0;
1     5                             nia     := inbuf.nia;
1     5                             df.dat  := inbuf.df.dat;
1     5                           END;
1     4                         send_mail;
1     4                       END;
1     3
1     3   (* vra skryf sinchronisasie register aan *)
1     3
1     3   {req_write_reg}
1     3                10   :BEGIN
1     4                       IF (write_reg.count > 0)
1     4                         THEN BEGIN
1     5                           writeln(imem,'write reg is nog nie leeg nie');
1     5                           GOTO 999;
1     5                         END;
1     4
1     4                       WITH write_reg DO
1     4                         BEGIN
1     5                           count := inbuf.ex;
1     5                           addr  := round(inbuf.df.dat);
1     5                           token := inbuf;
1     5                         END;
1     4
1     4                       WITH outbuf DO
1     4                         BEGIN
1     5                           u       := inbuf.u;
1     5                           i       := inbuf.i;
1     5                           d       := dfr1;
1     5                           func    := inbuf.func;
1     5                           ex      := 0;
1     5                           nia     := inbuf.nia;
1     5                           df.dat  := inbuf.df.dat;
1     5                         END;
1     4                       send_mail;
1     4                     END;
1     3
1     3   (* vra toevoer sirchronisasie register aan *)
1     3
1     3   {req_input_reg}
1     3                11   :BEGIN
1     4                       IF (input_reg.count > 0)
1     4                         THEN BEGIN
1     5                           writeln(imem,'reg is nog nie leeg nie');
1     5                           GOTO 999;
1     5                         END;
1     4
1     4                       WITH input_reg DO
1     4                         BEGIN
1     5                           count := inbuf.ex;
```

```
1     5                          addr  := round(inbuf.df.dat);
1     5                          token := inbuf;
1     5                        END;
1     4
1     4                      WITH outbuf DO
1     4                        BEGIN
1     5                          u       := inbuf.u;
1     5                          i       := inbuf.i;
1     5                          d       := out;
1     5                          func    := inbuf.func;
1     5                          ex      := inbuf.ex;
1     5                          nia     := inbuf.nia;
1     5                          df.dat  := inbuf.df.dat;
1     5                        END;
1     4                      send_mail;
1     4                    END;
1     3
1     3   (* vra afvoer sinchronisasie register aan *)
1     3
1     3   {req_output_reg}
1     3              12   :BEGIN
1     4                      IF (output_reg.count > 0)
1     4                        THEN BEGIN
1     5                          writeln(imem,'reg is nog nie leeg nie');
1     5                          GOTO 999;
1     5                        END;
1     4
1     4                      WITH output_reg DO
1     4                        BEGIN
1     5                          count := inbuf.ex;
1     5                          addr  := round(inbuf.df.dat);
1     5                          token := inbuf;
1     5                        END;
1     4
1     4                      WITH outbuf DO
1     4                        BEGIN
1     5                          u       := inbuf.u;
1     5                          i       := inbuf.i;
1     5                          d       := dfr1;
1     5                          func    := inbuf.func;
1     5                          ex      := 0;
1     5                          nia     := inbuf.nia;
1     5                          df.dat  := inbuf.df.dat;
1     5                        END;
1     4                      send_mail;
1     4                    END;
1     3   (* toevoer *)
1     3   {input}
1     3              13   :BEGIN
1     4                      IF (input_reg.count > 0)
1     4                        THEN BEGIN
1     5                          buf[input_reg.addr].df := inbuf.df;
1     5                          buf[input_reg.addr].status  := written;
1     5                          input_reg.count := input_reg.count - 1;
1     5                          input_reg.addr  := input_reg.addr  + 1;
1     5                          IF (input_reg.count = 0)AND(input_reg.token.nia.e
1     5                            THEN BEGIN
1     6                              WITH outbuf DO
1     6                                BEGIN
1     7                                  u       := input_reg.token.u;
1     7                                  i       := input_reg.token.i;
1     7                                  d       := dfr1;
1     7                                  func    := input_reg.token.func;
```

```
1    7                                    ex      := 0;
1    7                                    nia     := input_reg.token.nia;
1    7                                    df.dat := input_reg.token.df.dat;
1    7                                  END;
1    6                                send_mail;
1    6                              END;
1    5                            END;
1    4                          END; {input}
1    3
1    3    (* afvoer *)
1    3    {output}
1    3                  14    :BEGIN
1    4                      IF (output_reg.count = 0)
1    4                        THEN BEGIN
1    5                          writeln(imem,'reg is alreeds leeg');
1    5                          GOTO 999;
1    5                        END;
1    4
1    4                      i := inbuf.ex;
1    4                      IF (i > maks_1)
1    4                        THEN BEGIN
1    5                          writeln (imem,'i ongeldig',i);
1    5                          GOTO 999;
1    5                        END;
1    4
1    4                      buf[i].df := inbuf.df;
1    4                      buf[i].status  := written;
1    4                      output_reg.count := output_reg.count - 1;
1    4                      IF (output_reg.count = 0)
1    4                        THEN BEGIN
1    5                          j := output_reg.addr;
1    5                          k := output_reg.token.ex;
1    5                          FOR i := j to (j + k - 1) DO
1    5                            BEGIN
1    6                              outbuf.u    := output_reg.token.u;
1    6                              outbuf.i    := output_reg.token.i;
1    6                              outbuf.d    := out;
1    6                              outbuf.func := output_reg.token.func;
1    6                              outbuf.ex   := 0;
1    6                              outbuf.nia  := output_reg.token.nia;
1    6                              outbuf.df   := buf[i].df;
1    6                              send_mail;
1    6                            END;
1    5
1    5                          IF (output_reg.token.nia.address > 0)
1    5                            THEN BEGIN
1    6                              WITH outbuf DO
1    6                                BEGIN
1    7                                  u       := output_reg.token.u;
1    7                                  i       := output_reg.token.i;
1    7                                  d       := dfr1;
1    7                                  func    := output_reg.token.func;
1    7                                  ex      := 0;
1    7                                  nia     := output_reg.token.nia;
1    7                                  df.dat := 0;
1    7                                END;
1    6                              send_mail;
1    6                            END;
1    5                        END;
1    4                    END; {output}
1    3
1    3    {delete}
1    3                  15    :BEGIN
```

```
1    4                          j := round(inbuf.df.dat);
1    4                          k := inbuf.ex + j - 1;
1    4                          IF (k > maks_1)
1    4                            THEN BEGIN
1    5                              writeln(imem,'delete veld te groot',j,k);
1    5                              GOTO 999;
1    5                            END;
1    4                           FOR i := j TO k DO
1    4                            BEGIN
1    5                              buf[i].status := empty;
1    5                              buf[i].pionter := 0;
1    5                            END;
1    4                       END;
1    3
1    3                  OTHERWISE BEGIN
1    4                    writeln(imem,'func bestaan nie',inbuf.func);
1    4                    GOTO 999;
1    4                  END;
1    3              END; {case}
1    2    UNTIL doomsday;
1    1
1    1    999:
1    1    END. {imem}
```

BYLAE I

I HUIDIGE DATAVLOEI-SISTEME

Die huidige datavloei-verwerkersisteme kan basies in twee argitektuur organisasies opgedeel word, naamlik:

(i) Die pakketkommunikasie- en geheueselkonsep van die MIT model.

(ii) Die pakketkommunikasie- en pakketbenamingsmetode van GOSTELOW [12,13].

'n Paar van die bekendste model realiserings word nou gegee, soos gegee deur Treleaven [1], saam met 'n vlugtige verduideliking van die werking. Voorbeelde van tipe een:

(1) "Texas Instruments Distributed Data Processor"
(2) "Toulouse LAU sisteem"

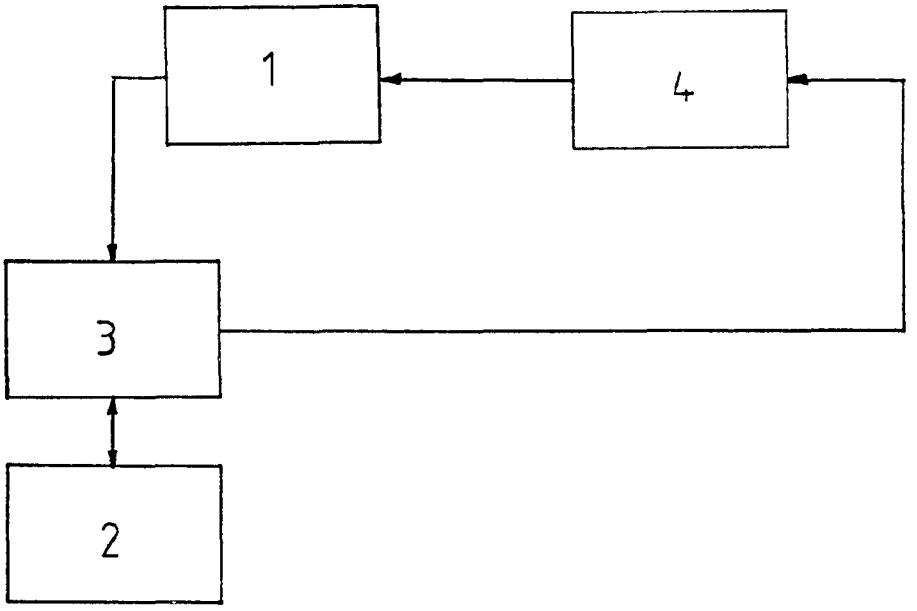
Voorbeelde van tipe twee:

(1) "Irvine Data Flow Machine"
(2) "Manchester Data-Flow Computer"

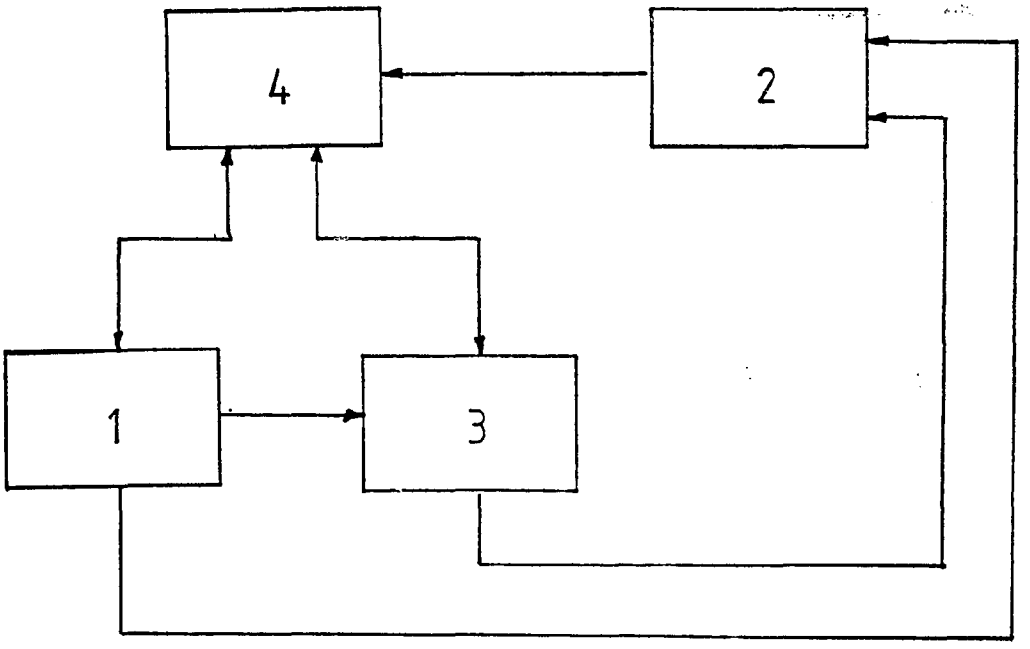
Aangesien van die bostaande stelsels multiverwerkers is, sal slegs een van die datavloei-verwerkers in so 'n geval bespreek word.

I.1 Texas Instruments Distributed Data Processor

Die model is soortgelyk aan die MIT model en bevat ook 'n geheue eenheid met instruksieselle. Die verwerker bestaan uit die volgende eenhede:



Texas Instruments Distributed Data Processor



Toulouse LAU Sistem

(1) Die verwerkingseenhede

(2) Die programgeheue

(3) Die "updata" beheerder

(4) Die geaktiveerde instruksiewagtou-eenheid

Die model se instruksies bestaan uit die volgende velde:

(1) Die operasie kode van die instruksie

(2) Die datapakket tellerveld wat die aantal pakketwaardes aandui vir instruksie aktivering

(3) Die oorspronklike aantal datapakkette wat vir instruksie aktivering benodig word

(4) Die lyskoppeling veld (vir 'n "linked list")

(5) Die datavelde vir die betrokke instruksiedata

(6) Die onderskeie destinasie-adresse waarheen die resultate gestuur moet word

Die verwerkingsproses geskied as volg: Die verwerker element verwyder 'n wagtende geaktiveerde instruksie vanaf die tou-eenheid en voer die gespesifiseerde operasie op die data uit. Die resultate word nou na die sogenaamde "updata" beheerder gestuur. Die beheerder stoor die resultaat-data in 'n dataveld van die opvolgende instruksie en dekrementeer die aktiverings hoeveelheid veld. Indien hierdie veld nou zero is, word die instruksie as geaktiveer beskou en 'n kopie daarvan op die geaktiveerde-instruksiewagtou geplaas. Die gestoorde instruksie word nou herstel na sy oorspronklike toestand en wag op die volgende datawaardes. Indien die instruksietou vol word, (indien die verbruikersalgoritme heelwat inherente parallelisme bevat) kan die instruksiekoppelveld gebruik om 'n

lys van geaktiveerde instruksies te koppel.

I.2 Toulouse LAU sisteem

Hierdie stelsel is ook soortgelyk aan die MIT model en maak weereens gebruik van die geheueselkonsep. Alhoewel die model ook sekere beheerkonsepte, byvoorbeeld die deel van data-arias toelaat, moet dit beklemtoon word dat die beskikbaarheid van data die program-uitvoering dirigeer. Beheerseine en data word afsonderlik hanteer en albei dra tot instruksieaktivering by.

Die verwerker bestaan uit vier eenhede naamlik:

(1) Die instruksiebeheergeheue - stoor beheervlae
(2) Die geaktiveerde instruksiewagtou
(3) Die databeheergeheue - stoor datawaardes
(4) Die verwerkerelemente

Die model instruksies bestaan uit die volgende velde:

(1) Die operasiekode
(2) Die twee geheue adresvelde van die toevoerdata
(3) Die datadestinasie-adres vir die operasieresultate

Konstantes word in die tweede geheue-adresveld gestoor en na elke instruksieaktivering hergenereer. Die geheueselle bestaan uit 'n dataveld en twee koppelvelde wat die dataveld aan twee afsonderlike instruksies kan koppel. Ooreenkomstig aan elke instruksie, is daar 'n aantal beheervlae wat gebruik word om die instruksie uitvoering te sinchroniseer. Die beheervlae C1 en C2 gee afsonderlik die aanwesigheid van datawaardes een en twee in die datageheue aan, terwyl C0 'n omgewingbeheervlag is (word in lusiterasie-uitvoering gebruik).

Die verwerking verloop as volg: 'n Instruksie word geaktiveer

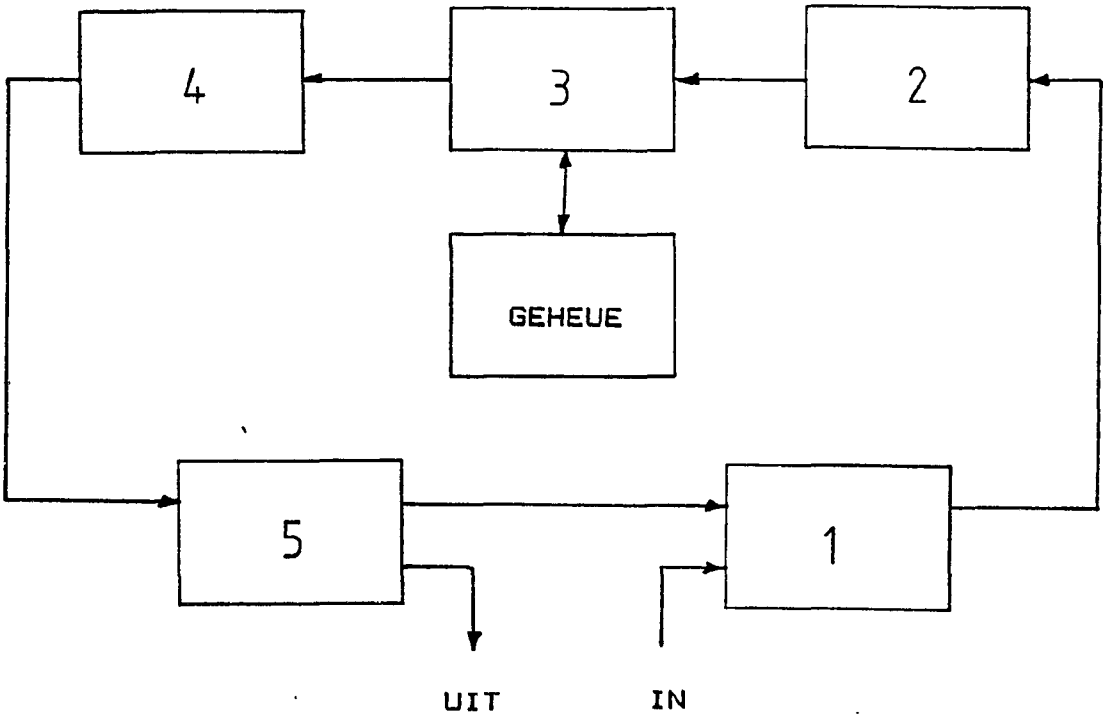
indien al sy beheervlae in die beheergeheue-eenheid gestel is (wanneer C0,C1,C2 voorgestel word deur 111). Die eenheid neem die geaktiveerde instruksie waar en plaas die instruksie-adres in 'n wagtou. Die databeheergeheue neem hierdie geaktiveerde instruksie-adres, voeg die instruksie daarby, en sit dit in 'n geaktiveerde instruksiewagtou. Een van die verwerkerelemente neem nou die instruksie en voer die gespesifiseerde operasie uit deur die betrokke datawaardes uit die geheue-eenheid aan te vra. Die resultaat instruksie-adres word gebruik om die opvolgende instruksie in die databeheergeheue te adresseer. Die opvolgende instruksie bevat 'n dataverwysingsveld waarin die adres van resultaatdatawaarde se datasel gegee word. Die datawaarde word nou in die datasel gestoor en die koppeladres ooreenkomstig verander. Parallel hieraan word die opvolgende instruksiebeheervlae in die instruksiebeheergeheue gestel om die opvolgende instruksie te aktiveer.

I.3 Irvine Data Flow Machine

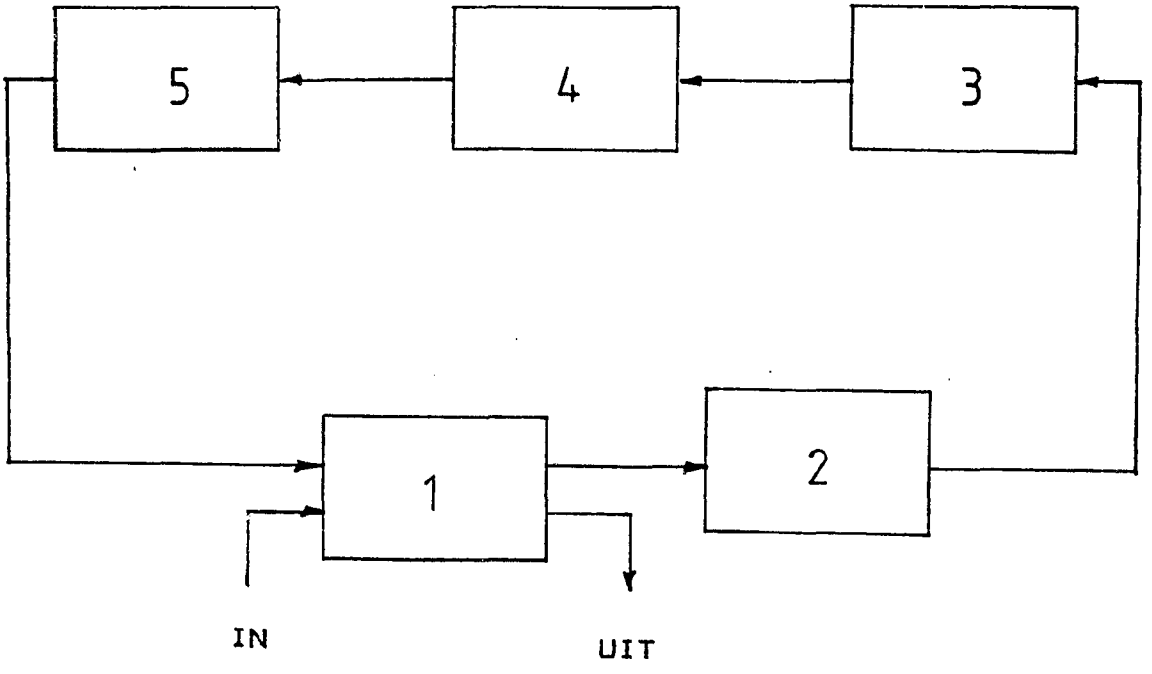
Die programorganisasie is suiwer datavloei sodat die instruksies onmiddelik geaktiveer raak indien die data bekikbaar word. Die argitektuurorganisasie impliseer 'n pakketkommunikasie pakketbenamingsmetode. Die pakketbenaming word gebruik om kode hertoeganklikheid te bewerkstellig, wat basies FIFO buffering op die operator toevoerboë simuleer. Die verwerker bestaan uit die volgende eenhede:

(1) Die toevoerseksie
(2) die verbindingstoor
(3) Die instruksiehaalseksie
(4) Die verwerkingseenheid
(5) Die afvoerseksie

Die verwerker ondersteun struktuurhantering deur middel van die "I Structures" van Arvind. Soos reed verduidelik is, kan



Irvine Data Flow Machine



Manchester Data Flow Computer

hierdie eenheid asinchrone leesoperasies tollereer, maar slegs een toewysing mag per geheueposisie geskied. Die datapakkette bestaan uit die volgende velde:

(1) 'n Naamveld wat uit die volgende sub-velde bestaan
    a) 'n Kodebloknaam
    b) 'n Blokinstruksienommer
    c) 'n Konteksveld wat 'n funksieroep konteks aandui
    d) 'n iterasieveld wat 'n lusiterasiediepte aandui
(2) 'n Destenasie instruksie-adres
(3) 'n Dataveld met 'n waarde of 'n verwysing

Die verwerking verloop as volg: Inisiële toevoerwaardes word by die invoerseksie ingevoer en na die programintreepunte gestuur. Die instruksiehaal-eenheid voeg die betrokke data by die aangevraagde instruksie, waarna die geaktiveerde instruksie na die verwerkereenheid gestuur word. Die instruksie bevat 'n operasie en die resultaatadresse. Die verwerker voeg die verwysde data by die operasie. Die operasie word uitgevoer en die resultate indien dit benodig word, in die geheue gestoor. resultate word as data of geheuewysers in datapakkette aangestuur na die afvoerseksie. Die afvoere word na buite of terug na die verwerker invoerseksie geroeteer. Indien twee datapakkette benodig word vir 'n instruksie-uitvoering, ontmoet die datapakketnaamgenote mekaar in die verbindingstoor, waar enkele datapakkette tydelik gebuffer word todat sy maat voorkom.

I.4 Manchester Data-Flow Computer

Hierdie programorganisasie is ook suiwer datavloei, deurdat instruksies geaktiveer word indien die data toevoere beskikbaar is. Die argitektuurorganisasie is weereens pakketkommunikasie met pakketbenaming. Die verwerker bestaan uit die volgende eenhede [19,20,21]:

(1) Die skakelaar vir toevoer/afvoer
(2) Die datapakkettou
(3) Die verbindingstoor
(4) Die instruksiehaaleenheid
(5) Die verwerkingselemente

Elke instruksie benodig een of twee toevoerdatapakkette om geaktiveer te word. 'n Datapakket is soortgelyk aan die Irvine model, en bevat 'n naamveld, 'n dataveld en 'n destinasie-adres. Die datavloei-instruksie bestaan uit die volgende velde:

(1) 'n Operasiekode
(2) 'n Destenasie-adres
(3) 'n Tweede destinasie of 'n konstante

Die verwerking geskied as volg: Inisiële toevoerwaarde word weereens na die programintreepunte gestuur om programuit-voering te sneller. Die instruksiehaal-eenheid onttrek die verwysde instruksie uit die programgeheue en voeg dit by die inkomende data. Die verwerkerelemente neem nou die geaktiveer-de instruksie en voer die gespesifiseerde operasie uit. Die resultate word nou na buite of na die datapakkettou geroeteer.

Die datapakkette uit die wagtou word na die verbindingstoor gestuur. Indien 'n gegewe pakket 'n maat benodig, word dit in die verbindingstoor met sy naamgenoot, indien dit aanwesig is, verbind. Indien die maat nie in die stoor voorkom nie, word die pakket tydelik hier gebuffer todat die maat voorkom.

I.5 Ander datavloei-projekte

Heelwat projekte vir die fisiese implementering van datavloei-stelsels was reeds, en is tans besig by Europese, Japanese en Amerikaanse navorsingsinstansies, onder andere:

(1) Die multidatavloei-verwerker van Arvind [12,22] by die Laboratory of Computer Science MIT Cambridge, Massachusetts, USA.

(2) Die DDDP van Kishi [23] by die Systems Laboratory OKI Electric Industry Co., Ltd Minato-ku, Tokyo 108, Japan.

(3) Die datavloei-ryverwerker van Takahashi [24] by Musashino Elec. Communication Lab. Nippon Telegraph and Telephone Public Corporation 3-9-11 Midoricho Musashino-shi Tokyo 180 Japan.

(4) Die syferseindatavloei-verwerker van Kronlof [17] by Helsinki Univerity of Technology, Department of Technical Physics, SF-02150 Espoo 15, Finland.

(5) Bell Laboratories (AT&T) se EMSP syferseindatavloei-verwerker [27].

(6) Die Utah Data Driven Machine (DDM1) [25,26] van Burroughs Interactive Research Center, La Jolla, California.

(7) Die Newcastle Data-control Flow Computer by die Universiteit van Newcastle upon Tyne.

'n Komersiële datavloeiverwerker [28] word tans by die NEC Electronics Natick (Mass.) Technology Center ontwikkel. Die PD7281 is basies 'n enkel geïntegreerde stroombaan, wat die datavloei-pyplyn voorstel.

BYLAE J

I 'n Voorbeeld

Aangesien die programmering tydsaam is, word slegs 'n eenvoudige voorbeeld verduidelik. Die gemiddeld van N getalle word geneem. Die volgende konsepte word gedemonstreer.

1) Eenvoudige toevoer/afvoer-meganismes
2) Die hertoelatings-meganismes (lusoopvouing)
3) Die asinchrone pyplyn verwerking
4) Struktuurhantering

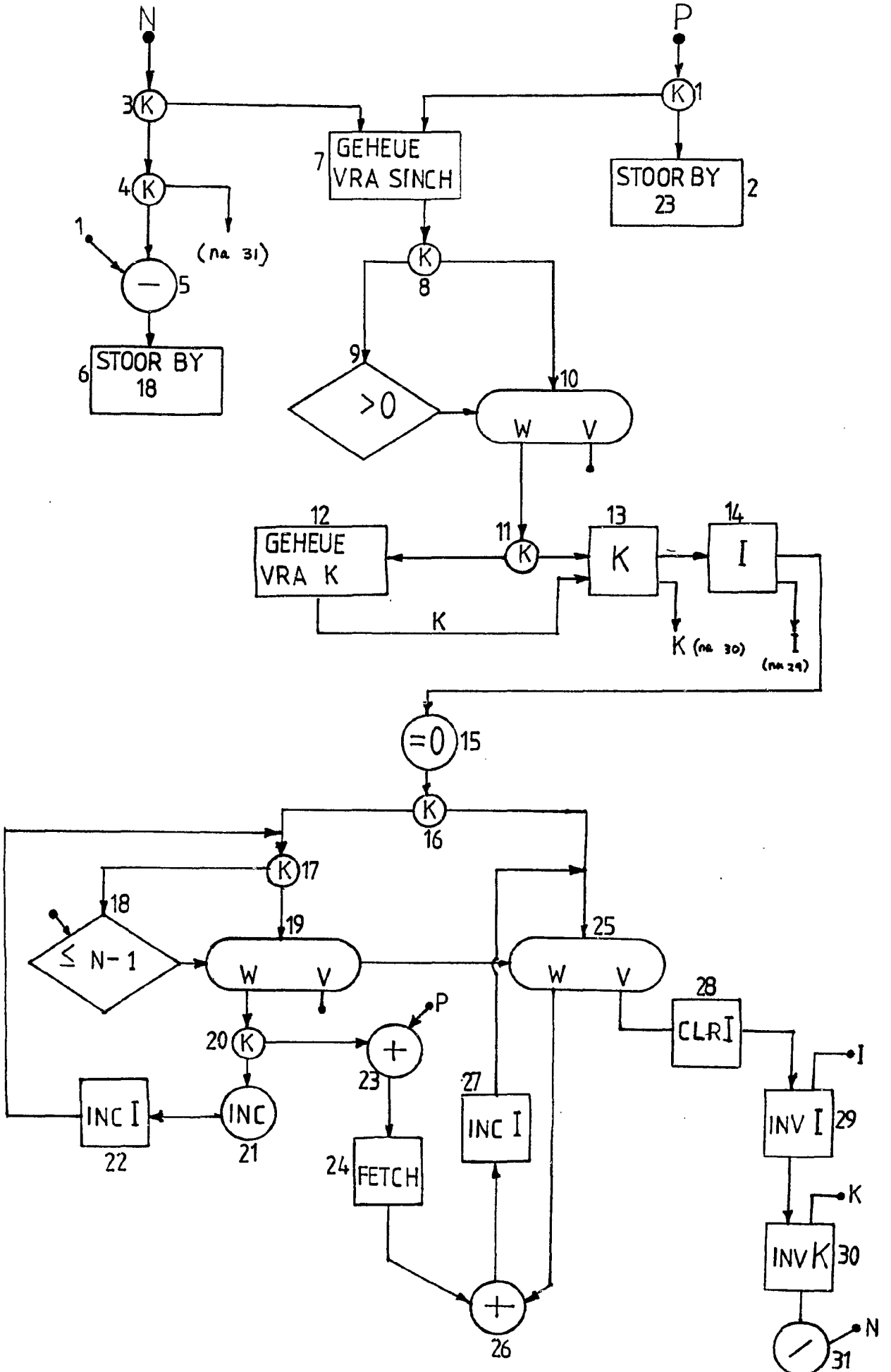
Die programmeringsmetode was as volg: Die datavloei-diagram van die funksie is getrek waarna die operators genommer is. Die operators is in sekwensie neergeskryf en met behulp van die destinasie-adresse verbind. Hierdie datavloei-masjientaal stel dus die vloeidiagram van die funksie voor. Die masjientaal word in 'n leêr gestoor, wat tydens die model opstelfase in die instruksiehaalproses se programgeheue gelaai word.

Die inisiële datawaardes (N -datawaardes en P - die struktuur-wyser) word by die monitor ingevoer en na die afsonderlike programintreepunte, instruksie 1 en 3 gestuur. Die data word nou na die geheueproses gestuur (ook via die monitor) en sodra al die data beskikbaar is, word die programverloop gesneller. Die program algoritme is as volg:

```
INPUT X,N

I   = 0
SOM = 0
REPEAT
  SOM = SOM + 1
  I   = I   + 1
UNTIL (I > N - 1)
ANTW = SOM/N

OUTPUT ANTW
```



— DIE VOORBEELD —

Die programinstruksies word gegee:

1) KOPIEER - Die toevoerwyser word gekopieer.
2) KONSTANTE STOOR - Stoor konstante by instruksie 23
3) KOPIEER
4) KOPIEER
5) MINUS - Die konstante 1 word van die toevoerwaarde afgetrek
6) KONSTANTE STOOR - Stoor konstante by instruksie 18
7) VRA SINCHRONISASIE - vra intree sinchronisasie
8) KOPIEER
9) BESLUITNEMING - Is die intree groter as 0
10) SKAKEL - Skakel die toevoer na een van die afvoere
11) KOPIEER
12) KONTEKS AANVRAAG - Vra konteks van geheueproses
13) KONTEKS VERVANGING - Vervang ou konteks met nuwe en stuur die ou konteks na instruksie 30
14) INDEKS VERVANGING - Herstel indeks na eenheid en stuur ou indeks na instruksie 29
15) NOP - Geen operasie maar herstel toevoerwaarde na 0
16) KOPIEER
17) KOPIEER
18) BESLUITNEMING
19) SKAKEL
20) KOPIEER
21) INKREMEMTEER - Inkrementeer toevoerwaarde
22) INKREMENTEER INDEKS - inkrementeer toevoerpakket indeksveld
23) SOMMEER
24) GEHEUE-AANVRAAG - Vra 'n geheueposisie datawaarde aan
25) SKAKEL
26) SOMMEER
27) INKREMENTEER INDEKS
28) HESTEL INDEKS - hestel indeksveld na eenheid
29) INVERSE INDEKS - Vervang indeks met oue
30) INVERSE KONTEKS - Vervang konteks met oue
31) DEEL

Die programverloop

Die inisiële toevoerwaardes P en N word gebruik om die geheueproses vir sinchronisasie aan te vra (instruksie 7), om sodoende geordende toevoer te bewerkstellig. Die waardes word ook gebruik vir verspreiding na die opvolgende instruksies 18,23 en 31.

'n Zero waarde word nou onmiddelik aan die kopieerinstruksie 8 gestuur om aan te dui dat die toevoersinchronisasieregister in die geheueproses toegeken is. Die waarde is gelyk aan zero by instruksie 9 en daarom word die toevoer van instruksie 10 na die "vals" afvoer geroeteer. Die toevoerdatawaardes (waarvan die gemiddeld verkry wil word) word nou direk na die geheueproses gestuur. Indien al N waardes ontvang is (met behulp van die "intree" funksie in die geheueproses), detek die geheueproses hierdie toestand en stuur die toevoer-struktuurwyser P aan na instruksie 8. Die toevoerdatawaarde by instruksie 9 is groter as zero en daarom word instruksie 10 se toevoer na instruksie 11 afgevoer.

Instruksie 11 versprei die toevoer waarde na instruksie 13 en 12 waarvan die laasgenoemde 'n konteksaanvraag na die geheue-proses stuur. Die geheueproses stuur nou die unieke konteks na konteksoperasie instruksie 13, waar die konteksveld van die afvoerdatapakket met hierdie nuwe konteks vervang en na die opvolgende instruksie gestuur word. Die ou konteks word ook nou deur instruksie 13 (met behulp van 'n datapakket) na die inverskonteksoperasie by instruksie 30 gestuur. Die indeks-operator by instruksie 14 herstel die indeksveld van die toevoerdatapakket na eenheid en stuur dit aan na instruksie 15. Die ou indeksdiepte word ook na die inversindeksoperator by instruksie 29 getuur.

Instruksie 15 herstel die toevoerdatawaarde na zero en roeteer

die afvoerwaarde na die kopieeroperasie, instruksie 16. Die toevoerdatawaardes word nou na instruksie 17 en 25 geroeteer, om onderskeidelik die indekswaarde (moet nie verwar met die indeksveld nie) en die som van die algoritme voorstel.

Instruksie 17 kopieer die toevoer na instruksie 18 en 19. Indien die toevoerdatawaarde na instruksie 18 kleiner of gelyk aan die aangeduide waarde is, word die toevoer van instruksie 19 na die kopieeroperasie instruksie 20 geroeteer (indien nie, word geen afvoerroetering gedoen nie). Die toevoer van instruksie 21 word met eenheid geïnkrementeer en na instruksie 22 aangestuur. Instruksie 22 inkrementeer die indeksveld van die toevoerpakket en stuur dit aan die kopieeroperasie instruksie 17, vanwaar die lokale lus weereens verloop.

Intussen word die indekswaarde en die struktuurbeginwyser gesomeer by instruksie 23 om die absolute geheue-adres van die stelseltoevoerdata te verkry (wat reed in die geheue gestoor is). Hierdie wyser word nou na die geheueproses gestuur om 'n geheueleesoperasie te bewerkstellig. Die geheueproses lees die gespesifiseerde posisie en stuur dit aan na sommerinsoperasie, instruksie 26.

Intussen is die toevoerdatawaarde na instruksie 25 (die "som") na (afhangende van die beheerwaarde vanaf instruksie 18 soos instruksie 19) ook na die sommeringsoperasie, instruksie 26 gestuur. Die ou som word in effek met die nuwe datawaarde (vanaf die geheueproses) gesommeer. Die antwoord word hierna na instruksie 27 geroeteer. Instruksie 27 inkrementeer die toevoerpakket se indekveld en stuur dit aan instruksie 25 om sodoende die lus te voltooi.

Die lus loop todat die toevoerdata na instruksie 18 nie meer kleiner of gelyk aan die gegewe bedrag is nie en die twee skakelaaroperasies, instruksies 19 en 25 roeteer hul

toevoerdata na die ondeskeie "vals"afvoere. Instruksie 19 het geen afvoer nie, terwyl instruksie 25 sy toevoer na instruksie 28 roeteer.

Instruksie 28 herstel die indeksveld na eenheid sodat die en voer die toevoer af na instruksie 29, sodat die afvoer met die ander naamgenoot wat vir instruksie 28 bestem is kan koppel. Instruksie 29 vervang die toevoerpakket se indeksveld met die ou indeksdiepte en stuur dit na instruksie 30 aan. Instruksie 30 vervang op sy beurt weer die toevoerpakket se konteksveld met die ou konteks en stuur dit na instruksie 31. Die instruksie 31 deel die toevoerdatawaarde deur N en roeteer die afvoer na buite (die monitorproses).

'n Afdruk van die program op die VAX word gegee (pakket tipe 4 in bylae B) saam met die vloeidiagram wat dit verteenwoordig.

rstr. nr.	1	2	3	4a	b	c	5a	b	c	6c	c
1	[illegible]	[illegible]	24	[illegible]	[illegible]	5	1	1	7	0	0.0000
2	[illegible]	[illegible]	0	0	0	23	0	[illegible]	0	0	0.0000
3	[illegible]	[illegible]	24	0	[illegible]	4	1	[illegible]	[illegible]	0	0.0000
4	[illegible]	[illegible]	24	0	[illegible]	5	1	1	31	0	0.0000
5	[illegible]	[illegible]	2	[illegible]	0	6	0	[illegible]	0	2	1.0000
6	[illegible]	[illegible]	0	0	0	10	0	0	0	0	0.0000
7	[illegible]	11	23	0	0	8	0	[illegible]	0	0	0.0000
8	[illegible]	0	24	0	0	9	1	1	10	0	0.0000
9	[illegible]	[illegible]	14	1	0	10	0	0	0	2	0.0000
10	[illegible]	0	25	[illegible]	0	11	0	[illegible]	0	0	0.0000
11	[illegible]	0	24	0	0	12	1	1	13	0	0.0000
12	[illegible]	[illegible]	[illegible]	1	0	13	0	[illegible]	0	0	0.0000
13	[illegible]	0	26	1	0	30	0	0	14	0	0.0000
14	[illegible]	0	27	1	0	29	0	1	15	0	0.0000
15	[illegible]	[illegible]	0	0	0	19	0	0	0	2	0.0000
16	[illegible]	0	24	0	0	17	1	1	29	0	0.0000
17	[illegible]	0	24	0	0	18	1	1	19	0	0.0000
18	[illegible]	0	17	1	0	19	1	0	25	0	0.0000
19	[illegible]	0	25	0	0	20	0	0	0	0	0.0000
20	[illegible]	0	24	0	0	21	0	0	23	0	0.0000
21	[illegible]	[illegible]	[illegible]	0	0	22	0	[illegible]	0	0	0.0000
22	[illegible]	0	28	0	0	17	0	0	0	0	0.0000
23	[illegible]	0	1	0	0	24	0	0	0	0	0.0000
24	[illegible]	1	25	1	0	26	0	0	0	0	0.0000
25	[illegible]	0	25	1	1	26	0	0	28	0	0.0000
26	[illegible]	[illegible]	1	0	0	27	0	0	0	0	0.0000
27	[illegible]	0	[illegible]	1	1	35	0	0	0	0	0.0000
28	[illegible]	0	31	1	1	29	0	0	0	0	0.0000
29	[illegible]	0	29	1	1	30	0	0	0	0	0.0000
30	[illegible]	0	30	1	0	31	0	0	0	0	0.0000
31	1	0	4	0	[illegible]	39	0	0	0	0	0.0000

BIBLIOGRAFIE

BIBLIOGRAFIE

[1] TRELEAVEN P.C., BROWNBRIDGE D.R., and HOPKINS R.P.
"Data-Driven and Demand-Driven Computer Architecture"
Computing Surveys, Vol. 14, Nr. 1, Maart 1982

[2] "Parallel Processing"
Computer, Januarie 1982

[3] MYERS G.J.
"Advances in Computer Architecture",
Second edition, John Wiley and Sons, bl.463-494

[4] DAVIS A.L. and KELLER R.M.
"Data Flow Program Graphs"
Computer, Vol.15, Nr.2, Feb 1982, bl.26-41

[5] ACKERMAN W.B.
"Data Flow Languages",
Proceedings of the NCC. Montvale, NJ: AFIPS, 1979, bl.1087-1095

[7] DENNIS J.B. and MISUNAS D.P.
"A preliminary architectute for a basic DATA FLOW Processor",
Procedings of the 2nd Annual Conference on Computer Architecture, ACM, 1975, bl.126-132

[8] DENNIS J.B. and WENG K.
"An abstract implementation for Concurrent Computation with Streams",
Proceedings of the 1979 International Conference on parallel Processing, bl.35-45

[9] DENNIS J.B.
"Data Flow Supercomputers",
IEEE Computer, 1980, 13(11), bl.48-56

[10] DENNIS J.B., BOUGHTON G.A. and LEUNG C.K.C
"Building Blocks for Data Flow Prototypes",
Proceedings of the 7th Annual Symposium on Computer Architecture, ACM, 1980, bl.1-8

[11] ARVIND, IANNUCCI R.A.
"A Critique of Multiprocessing Von Neumann Style",
Proceedings of the 10th. Annual Symposium on Computer Architecture, ACM, 1983, bl.426-436

[12] ARVIND and GOSTELOW K.P.
" The U-interpreter",
Computer, Vol.15, Nr.2, Vol.6, Nr.1, Feb. 1982, bl.42-49

[13] GOSTELOW K.P. and THOMAS R.E.
"A View of Data Flow",
Proceedings of the NCC. Montvale, NJ:AFIPS, 1979, bl.629-636

[14] WATSON I. and GURD J.
"A Prototype Data Flow Computer with Token Labeling",
Proceedings of the NCC. Montvale, NJ:AFIPS, 1979, bl.623-628

[15] GAJSKI D.D., PADUA D.A., KUCK D.J., KUKN R.H.
"A Second Opinion on Data Flow Machines and Languages",
Computer, Vol.15, Nr.2, Feb 1982, bl.58-69

[16] GOTO E., and IDA T.
"Parallel Hashing Algorithms",
Information Processing Letters 6, 1 (Feb 1977),
bl.8-13

[17] KRONLOF K.
"Execution Control and Memory Management of a Data Flow Signal Processor"
Proceedings of the 10th Annual Symposium on Computer Architecture, ACM, 1983, bl.230-235

[18] VAX/VMS
"System Services Reference Manual"
DEC, Maart 1980

[19] GURD J. and WATSON I.
"Data Driven System for High Speed Parallel Computing - Part 1 :Structuring Software for Parallel Execution",
Computer Design, Junie 1980, bl.91-100

[20] GURD J. and WATSON I.
"Data Driven System for High Speed Parallel Computing - Part 2 :Hardware Design",
Computer Design, Julie 1980, bl.97-106

[21] WATSON I. and GURD J.
"A Practical Data Flow Computer",
Computer, Vol.15, Nr.2, Feb. 1982, bl.51-57

[22] ARVIND and KATHAIL V.
"A Multiple Processor Dataflow Machine that Supports Generalized Procedures",
Proceedings of the 8th. International Symposium on Computer Architecture, 1981, bl.291-302

[23] KISHI M., YASUHARA H. and KAWAMURA Y.
"DDDP : A Distributed Data Driven Processor",
Proceedings of the 10th Annual Symposium on Computer Architecture, ACM, 1983, bl.236-242

[24] TAKAHASHI N. and AMAMIYA M.
"A data Flow Processer Array System : Design and Analysis",
Proceedings of the 10th Annual Symposium on Computer Architecture, ACM, 1983, bl.243-250

[25] DAVIS A.L.
"The Architecture and System Method of DDM1: A Recursively Structured Data Driven Machine",
Proceedings of the 5th. Annual Symposium on Computer Architecture, ACM, 1978, bl.210-215

[26] DAVIS A.L
"A Data Flow Evaluation System Based on the Concept of Recursive Locality",
Proceedings of the NCC. Montvale, NJ:AFIPS, 1979, bl.1079-1086

[27] "Advanced Computer Technology: Architectures"
Electronic Design, Mei 1984, bl.170-194

[28] Meshach w.
"Data-flow IC makes short work of tough processing chores"
Electronic Design, Mei 1984, bl.191-206

BYLAE F

E Die instruksiehaalproses

Die proses begin deur sy eie posbus te skep en daarna die adres van die opvolgende verwerkerprosesse se posbusadresse te verkry. Die aangevraagde programinstruksieleêer word daarna in die programgeheue gelees en die proses gaan in 'n oneindige lus.

Die lusverloop is as volg: Die destinasieadres van die geleesde toevoerdatapakket word gebruik om die verwysde instruksie aan te spreek. Die inkomende datapakket word saam met die instruksie gevoeg en na die verwerkingsprosesse aangestuur. Indien die toevoer-pakkette 'n enkel datawaarde bevat en die destinasie-instruksie 'n konstante bevat, word die konstante tesame met die toevoerdata en die instruksie aan die verwerkerprosesse gestuur.

Indien die toevoerpakket na 'n instruksie verwys waarvan die funksieveld 'n "konstante stoor" funksie gevat, word die data onmiddellik in die verwysde data-instruksie se "konstante-veld" gestoor.

Die toevoerdatapakket is van tipe 2 en die afvoerdatapakket is van tipe 3 in bylae B.