MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

REKENAFDELING

CURSUS

1955-'56

Programmeren voor automatische rekenmachines

onder leiding van

Prof. Dr. Ir. A. van Wijngaarden en de Heer E.W. Dijkstra

The Mathematical Centre at Amsterdam, founded the 11th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications, and is sponsored by the Netherlands Government through the Netherlands Organization for Pure Research (Z.W.O.) and the Central National Council for Applied Scientific Research in the Netherlands (T.N.O.), by the Municipality of Amsterdam and by several industries.

INHOUD

Errata

1. Algemene inleiding over automatische rekenmachines
2. Het woord
3. Het getal
4. De opdracht
5. Blokschema's
6. Subroutines I
7. In- en Uitvoer van de ARRA
8. Een uitgewerkt voorbeeld
9. Subroutines II
10. Subroutines III
11. Snelheid
12. Een voorbeeld ter illustratie
13. Controle en flexibiliteit
14. De administratieve subroutine
15. De administratieve subroutine II, Integratie
16. Subroutine-aggregaten voor semi-interpretatief werk
17. Superprogramma's I
18. Superprogramma's II

ERRATA

Syllabus	No.	blz.	r.			
Syllabus	No. 1	blz. 0-1	r.1	0-1	moet zijn	1-1
"	" 1	" 0-2	" 1	0-2	" "	1-2
"	" 1	" 0-3	" 1	0-3	" "	1-3
"	" 1	" 0-4	" 1	0-4	" "	1-4
"	" 1	" 0-5	" 1	0-5	" "	1-5
"	" 2	" 1-1	" 1	1-1	" "	2-1
"	" 2	" 1-2	" 1	1-2	" "	2-2
"	" 2	" 1-3	" 1	1-3	" "	2-3
"	" 2	" 1-4	" 1	1-4	" "	2-4
"	" 2	" 1-5	" 1	1-5	" "	2-5
"	" 3	" 2-1	" 1	2-1	" "	3-1
"	" 3	" 2-2	" 1	2-2	" "	3-2
"	" 3	" 2-3	" 1	2-3	" "	3-3
"	" 3	" 2-4	" 1	2-4	" "	3-4
"	" 4	" 4-1	"24	oftewel	" "	ofwel
"	" 4	" 4-3	" 3	[S] 2^{29-n}	" "	[S] 2^{30-n}
"	" 4	" 4-3	"11	schoon maal tien	moet zijn	maal tien
"	" 4	" 4-3	"12	maal tien	" "	schoon maal tien
"	" 4	" 4-4	"15	23,29	" "	23,28
"	" 4	" 4-4	"27	4,300	" "	2,300
"	" 5	" 5-1	"25	vast, voor	" "	vast, terwijl voor
"	" 5	" 5-2	"24	de uitgang	" "	de ingang
"	" 5	" 5-5	fig. 9	$\sum_{j=0}^{2} a_j y^j \not> z$	moet zijn	$\sum_{j=0}^{3} a_j y^j \not> z$
"	" 7	" 7-1	r.4	A. v. Wingaarden	moet zijn	A. van Wijngaarden
"	" 11	" 11-1	r.27	mat	moet zijn	met
"	" 11	" 11-1	" 39	juist	" "	niet
"	" 11	" 11-6	" 34	10	" "	8
"	" 11	" 11-6	" 36	10	" "	8
"	" 12	" 12-4	" 7	naar S,het	" "	naar S, zodat het
"	" 12	" 12-5	" 16	4	" "	5
"	" 12	" 12-5	" 25	-15	" "	-5
"	" 12	" 12-5	" 31	+15	" "	+5
"	" 14	" 14-1	" 38	$x_1 f(x)$	" "	$x, f(x)$
"	" 14	" 14-2	" 9	TYPT f(x)	" "	TYPT $\Delta f(x)$
"	" 14	" 14-2	" 15	TYPT f(x)	" "	TYPT $\Delta f(x)$
"	" 14	" 14-2	tussen regel 25 en 26	hor. streep		
"	" 14	" 14-2	" " 27 " 28	" "		
"	" 14	" 14-2	" " 29 " 30	" "		
"	" 14	" 14-2	" " 31 " 32	" "		
"	" 14	" 14-2	" " 33 " 34	" "		
"	" 14	" 14-2	" " 35 " 37	" "		
"	" 14	" 14-2	regel 36 vervalt			
"	" 14	" 14-2	tussen regel 38 en 39	" "		
"	" 14	" 14-3	" " 5 " 6	" "		
"	" 14	" 14-3	" " 12 " 13	" "		
"	" 14	" 14-3	" " 20 " 21	" "		
"	" 14	" 14-3	" " 27 " 28	" "		
"	" 14	" 14-3	" " 30 " 31	" "		
"	" 14 Administratieve subroutine II	blz.14-1	r. 1	No. 14	moet zijn	No. 15
"	" 14	" 14-2	" 1	14-2	" "	15-2
"	" 14	" 14-3	" 1	14-3	" "	15-3
"	" 14	" 14-4	" 1	14-4	" "	15-4

Syllabus	No.		blz.		r.					
Syllabus	No.	14	blz.	14-5	r.	1	14-5	moet	zijn	15-6
"	"	14	"	14-6	"	1	14-6	"	"	15-6
"	"	14	"	14-7	"	1	14-7	"	"	15-7
"	"	14	"	14-8	"	1	14-8	"	"	15-8
"	"	16	"	16-1	"	6	interpratief	moet	"	interpretatief
"	"	16	"	15-2	"	1	15-2	"	"	16-2
"	"	16	"	15-2	"	35	Complexe	"	"	complexe
"	"	16	"	15-3	"	1	15-3	"	"	16-3
"	"	16	"	15-4	"	1	15-4	"	"	16-4
"	"	18	"	18-1	"	6	Superprogramma's	moet	zijn	Superprogramma's II
"	"	18	"	18-3	"	11	ongekeerde	moet	zijn	omgekeerde
"	"	18	"	18-3	"	20	ingelegen	moet	zijn	ingelezen

Syllabus No.1 van de cursus 1955-'56:

Programmeren voor automatische rekenmachines

onder leiding van

Prof. Dr Ir A. van Wijngaarden en de Heer E.W. Dijkstra

Algemene inleiding over automatische rekenmachines

De oudste rekenmachines, die ontworpen zijn, hadden tot taak de mensen enig slavenwerk, nl. het rekenen met papier en potlood, uit handen te nemen. Het is bekend, dat met de nieuwste machines dit ideaal meer dan ingehaald is: tegenwoordig kan men amper spreken van „werk uit handen nemen": de problemen, waarbij de machtigste der moderne machines pas goed tot hun recht komen, zijn dusdanig omvangrijk, dat men er zonder deze rekenapparatuur nooit aan zou zijn begonnen! Er worden problemen mee aangepakt, die vroeger de meest drieste niet eens als „numeriek probleem" zou durven te beschouwen; en inderdaad, naarmate de methoden, waarop de machines hun resultaten afleveren, geraffineerder worden, raakt het numeriek karakter althans voor de naieve bezoeker, ernstig op de achtergrond: tegen de tijd dat men plaatjes op televisieschermen te zien krijgt..... Maar wij, die ons bezig hebben te houden met de interne organisatie, d.w.z. hoe de brug geslagen moet worden tussen een gegeven probleem en een gegeven machine, doen er goed aan, deze machines, ontdaan van alle spectaculaire luister, onderdeel voor onderdeel onder de loupe te nemen; het zal duidelijk worden, dat zij in hun functionele samenhang een rekenaar, uitgerust met een tafelrekenmachine, vrij aardig nabootsen en dus zijn werk over kunnen nemen.

DE ONDERDELEN.

Ruwweg gesproken bestaat een machine uit vijf onderdelen: de invoer, de uitvoer, het rekenorgaan, het geheugen en de besturing. Even ruw wordt de algemene gang van een berekening weergegeven door:

het opnemen van de noodzakelijke gegevens (via de invoer naar het geheugen);

het verwerken van (= rekenen met) deze gegevens (van het geheugen via het rekenorgaan terug naar het geheugen);

het afleveren van de resultaten (van het geheugen naar de uitvoer).

Het in dit functionele résumé verzwegen onderdeel, de besturing, is minder correct: het zorgt er b.v. voor, dat, als er opgeteld moet worden, het rekenorgaan inderdaad ook gàât optellen, het zorgt ervoor dat het juiste addendum op het juiste tijdstip op de juiste plaats aanwezig is, etc.

Omdat, zoals boven bleek, het geheugen de meest centrale plaats inneemt, zullen wij eerst onze aandacht hieraan schenken. De functie van het geheugen is dus ten eerste het onthouden van de noodzakelijke gegevens: men realisere zich, dat deze informatie tweeërlei is: numerieke informatie (waarden van parameters, coëfficiënten, desnoods complete matrices) en „logische informatie", d.w.z. specificatie, wàt er met al deze getallen gebeuren moet. Ten tweede onthoudt het geheugen informatie, die niet van de invoer afkomstig is, maar door de berekening gevormd is: de tussenresultaten, die niet naar de uitvoer gaan, en tenslotte de werkelijke uitkomsten, die wel naar de uitvoer gaan. Als enige informatie naar het geheugen gestuurd wordt om onthouden te worden, zegt men, „dat het in het geheugen is geschreven". Echter heeft het slechts zin, om informatie - denk aan tussenresultaten van de berekening - in het geheugen te schrijven, als deze informatie ook weer uit het geheugen gelezen kan worden; dit is inderdaad mogelijk. Het moet echter eveneens mogelijk zijn te specificeren wèlk getal in het geheugen - dat er immers honderden, ja duizenden bevatten kan! - gelezen moet worden. Dit zou niet mogelijk zijn, als de getallen in het geheugen geborgen werden gelijk knikkers in een zak; daarom is het geheugen ingedeeld in genummerde vakjes, adressen genaamd. Voor ieder getal; dat door het geheugen onthouden moet worden, wordt een bepaald adres gekozen: dit adres wordt gespecificeerd als het betroffen getal in het geheugen geschreven wordt en dit adres fungeert verder als „naam van het getal": iedere keer, dat de berekening dit getal weer nodig heeft, wordt er onder opgave van het betroffen adres in het geheugen gelezen.

De inhoud van een adres kan onbeperkt vaak gelezen worden en wordt pas vernietigd - vergeten! - als er een nieuw getal in geschreven wordt. Een geheugen dat wel „onthouden", maar niet „vergeten" kan, zou vol raken! Omdat „vergeten" alleen noodzakelijk is om nieuwe geheugenruimte vrij te maken, pleegt „vergeten" geen aparte operatie van de machine te zijn. Het is voldoende, als het schrijven de oude inhoud uitwist. De wijze waarop getallen in de adressen gerepresenteerd worden, verschilt van machine tot machine: b.v. sommige machines werken intern tientallig, an-

dere tweetallig. Tot zover de numerieke informatie, maar wij herinneren ons, dat het geheugen behalve de getallen, waarmede gerekend wordt, ook onthouden moet, wat er gerekend moet worden. Hiervoor wordt nu hetzelfde geheugen gebruikt, nadat voor de verschillende operaties die door het rekenorgaan verricht kunnen worden (zie onder) een nummercode is gekozen (b.v. 0 = optelling, 1 = aftrekking, 18 = vermenigvuldiging), om aan te geven, wat er met het betroffen getal gebeuren moet; met wèlk getal deze operatie uitgevoerd moet worden (dus b.v. welk getal opgeteld moet worden) wordt gespecificeerd door het adres aan te geven, waar dit getal in het geheugen te vinden is. Maar dit adres, het nummer van het vakje, is al een uit cijfers opgebouwd getal. Men kan zich dus misschien voorstellen, dat het mogelijk moet zijn, om de gang van de berekening vast te leggen in een geheugen, dat in eerste instantie alleen numerieke informatie herbergt. Zoals de informatie, die met de invoer word ingebracht tweeërlei is, is ook het contact met het geheugen bij werkende machine tweeërlei: eerst moet een opdracht uit het geheugen gelezen worden; bij de uitvoering van deze opdracht worden als regel een of meer getallen van of naar het geheugen getransporteerd. Als de opdracht voltooid is, komt het signaal, dat de volgende opdracht aan de beurt is, deze wordt gelezen etc En zo worden de opdrachten aan elkaar geregen tot zij samen de berekening volbrengen.

Een belangrijk begrip is "de snelheid van het geheugen", die afhangt van de gemiddelde tijd, die nodig is om in een adres te schrijven of eruit te lezen ("access-time"); hierbij is inbegrepen de eventuele wachttijd, die ontstaat, als elk adres niet continu, maar periodiek beschikbaar is: hoe kleiner deze periode is, hoe sneller het geheugen. De gebruiker moet zic dan rekenschap geven van de snelheid van de geheugens, als de machine geheugens van verschillende snelheid bezit: als de machine over een snel geheugen beschikt, dan is dat - uit financiële overwegingen - meestal vrij klein. Om van de anders knellende beperkingen van een klein geheugen bevrijd te zijn, beschikt de machine dan vaak over een groot, langzaam, geheugen. Door op het juiste moment bepaalde informatie uit het langzame geheugen "naar voren te halen" en in het snelle te copiëren, kan de tijdsduur van een berekening aanmerkelijk bekort worden.

Een tweede indeling is in "dood geheugen" en "levend geheugen". Het geheugen, zoals het boven beschreven is, is het zg. levend geheugen, d.w.z. de inhoud kan omdat erin geschreven kan worden, door de machine worden gewijzigd. Het lijkt erop, dat er

eens emplooi is voor een geheugen, waarvan de inhoud niet door de machine gewijzigd kan worden: geheugen met standaard informatie, zg. dood geheugen.
(In zekere zin is de geponste band, zie onder bij de invoer, te beschouwen als dood geheugen, dat tegen geringe prijs schier onbeperkt uitgebreid kan worden.)

Als voorbeelden van geheugens noemen wij:
Kwik vertragingslijn (acoustisch)
Nikkel vertragingslijn (magnetostrictie)
Statistische magnetische vertragingslijn
Williams-buizen (electrostatisch)
Magnetische trommel
Matrix geheugen (kerntjes met rechthoekige hysteresislus)
Magnetische band
(Geponste band en kaarten (maar één keer te beschrijven))

HET REKENORGAAN

Het is mogelijk, dat het rekenorgaan uitsluitend opereert op de inhoud van adressen: in geval van een optelling worden twee getallen uit het geheugen gelezen en de som wordt op een aangegeven adres geschreven (meer-adres code);

In een andere uitvoering wordt deze bewerking in drieën gesplitst door invoering van een soort totaal register: eerst wordt de andere term erbij opgeteld (de som wordt in het totaal register achtergelaten) en tenslotte wordt de inhoud van het totaal register op het adres, waar het antwoord moet komen te staan, gecopiëerd (één-adres code). Het aantal registers van het rekenorgaan verschilt van machine tot machine. Als het rekenorgaan over een paar de beschikking heeft, pleegt dit vergeleken met één totaal register, de efficiency zeer te verhogen, omdat veel transport vermeden kan worden. Twee registers is een veel voorkomend aantal. (Men kan de registers beschouwen als „speciale hoekjes van het levend geheugen").

De arithmetische bewerkingen waartoe de rekenorganen in staat zijn, verschillen: naast optellen en aftrekken, komen voor: het nemen van de absolute waarde, vermenigvuldigingen (niet altijd!), delen (niet altijd!); in oudere machines eveneens meer gezochte bewerkingen als worteltrekken, machtsverheffen.

INVOER

Hoewel waarschijnlijk alle machines een bedieningspaneel bezitten, waarmede (met schakelaars of telefoonschijven) het mogelijk is elk gewenst getal op een willekeurig adres in te vullen, is dit nimmer de gebruikelijke manier van invoer van gegevens, omdat hier teveel machinetijd mee gemoeid is. Als een bepaald probleem vaker op de machine gezet moet worden, heeft men bovendien liever de noodzakelijke informatie op een duurzaam medium, en wel zo, dat deze informatie zonder menselijke tussenkomst (foutenbron!) door de machine kan worden opgenomen.

De meest gebruikte methoden zijn ponskaarten, ponsband (telexband met vijf of zeven gaatjes) en magnetische band. Alle gegevens worden in een of andere (afgesproken) code geponst, resp. gemagnetiseerd; de keuze van deze code wordt bepaald door overwegingen, waar wij nu niet op in zullen gaan.

Machtig worden deze invoerfaciliteiten eerst dan, als de machine zelf in staat is, zijn eigen invoermedium te beschrijven: als de invoer via telexband geschiedt, is het gewenst, dat de machine ook banden kan ponsen.

UITVOER

Er zijn zoveel verschillende vormen van uitvoer, aangepast aan speciale behoeften, dat wij slechts de meest algemene zullen noemen.

Ten eerste de reeds vermelde uitvoer: schrijven op het eigen invoermedium. Dit is daarom zo belangrijk, omdat men dan in het invoermedium een schier onbeperkte uitbreiding van het geheugen gevonden heeft - zij het met vrij lange access-tijden en soms beperkte „opzoekmogelijkheid".

Ten tweede: het bedrukte papier. Dit is van belang, omdat hier het resultaat in leesbare, reproduceerbare vorm wordt afgeleverd. „Teletype" eenheden of door de machine bediende electrische schrijfmachines zijn veel voorkomende verwezenlijkingen. De schrijfmachine is langzaam: men magnetiseert de getallen sneller op een band. Het volgende arrangement is uitgevoerd: de machine magnetiseert de antwoorden op een band; als deze band vol is, begint de machine een nieuwe band te magnetiseren. Deze banden voert men naar „lees-verhaal-typ" eenheden, die de band uittypen. Hiervan kan de machine er verscheidene aan het werk houden!

Syllabus No. 2 van de cursus 1955-'56:

Programmeren voor automatische rekenmachines

onder leiding van

Prof. Dr Ir A. van Wijngaarden en de Heer E.W. Dijkstra

HET WOORD

In het volgende zal niet naar overdadige algemeenheid worden gestreefd. Het is dus mogelijk (zelfs zeker), dat er machines zijn of zullen zijn, waarvoor het onderstaande te beperkt is geformuleerd. Evenwel is het gemakkelijker zich bij de beschrijving aan iets concreets vast te klampen. Later kan men dan, na de eerste begripsmoeilijkheden te hebben overwonnen, altijd nog mogelijke generalisaties onder ogen te zien. Weinigen hebben moderne algebra geleerd zonder al lang vertrouwd te zijn met de klassieke algebra. Waar wij de puntjes op de i zetten zullen wij dat altijd doen aan de hand van de machine ARRA, waarvan enerzijds de eigenschappen representatief zijn voor een zeer grote klasse machines, zodat wij niet al te zeer hoeven te specialiseren, terwijl anderzijds het voordeel aanwezig is eventueel gemaakte programma's op de machine te onderzoeken op hun correctheid.

In het geheugen en in de rekenkundige registers van de machine bevinden zich „woorden". Een woord is een rij cijfers - op een doodenkele uitzondering na tweetallige cijfers in verband met het feit, dat de physische elementen, waaruit snelle en betrouwbare machines kunnen worden opgebouwd, zodanig zijn, dat ze in twee toestanden kunnen verkeren, welke toestanden we de namen 0 en 1 kunnen geven. Welke betekenis aan een dergelijke cijferrij zal worden toegekend, wordt daarmede geenszins gesuggereerd en inderdaad kan een woord op verschillende wijzen worden geïnterpreteerd al naar gelang de machine ervan gebruik maakt.

Allereerst kan een woord geïnterpreteerd worden als „getal". In het eenvoudigste geval, nl. bij de zg. tweetallige of wel binaire machines wordt dit getal verkregen door de cijferrij te interpreteren als de normale tweetallige schrijfwijze van het getal (over details, zoals interpretatie van teken en plaats van de komma, enz. wordt later gesproken). Bij andere, zg. tientallige machines, worden de cijfers van het woord eerst gegroepeerd gedacht in groepjes van bijv. vier cijfers. Deze groepjes worden geacht de tientallige cijfers 0,1,....9 te representeren en de aldus verkregen rij tientallige cijfers wordt nu weer als op de

normale tientallige wijze geschreven getal geïnterpreteerd.

Een tweede interpretatie van een woord is een „opdracht" of algemener een aantal, bijv. twee opdrachten, een zg. opdrachtenkoppel. Een opdracht is een hoeveelheid informatie, welke de machine leert welke handeling door haar moet worden verricht. Zij bestaat uit een functiegedeelte en een numeriek gedeelte, d.w.z. enige cijfers vormen samen op een als boven beschreven wijze een getal, dat het nummer is van een van de functies waartoe de machine in staat is, bijv. optellen, vermenigvuldigen, terwijl de overige cijfers een of meer getallen representeren, die een nadere specificatie geven van de opdracht, bijv. het adres waarop het getal zich bevindt dat opgeteld moet worden bij dat wat zich op een afgesproken plaats bevindt. Het aantal functies waartoe de meeste machines in staat zijn is tamelijk gering, zodat meestal slechts een klein aantal, bijv. 5, tweetallige cijfers voldoende is om het functiegedeelte van de opdracht te specificeren. Voorts is het aantal adressen in het geheugen ook tamelijk beperkt, zodat bijv. 10 cijfers voldoende zijn voor het numerieke gedeelte. Zo zijn bijv. 15 cijfers voldoende voor de gehele opdracht. Anderzijds zijn er met het oog op het gemakkelijker verwerken van getallen met hoge precisie vrij veel cijfers nodig, bijv. 30 om een getal te specificeren. De woordlengte is dan dus bijv. 30 cijfers en er kunnen twee opdrachten binnen één volledig woord worden geborgen, wat van duidelijk belang is met het oog op de beschikbare geheugenruimte. Een opdrachtlengte die juist even groter is dan de halve getallenlengte is blijkbaar hoogst ongeschikt.

Een derde interpretatie van een woord is die van een „code", d.w.z. volgens een of ander afgesproken systeem wordt een bepaalde hoeveelheid informatie afgebeeld op de cijferrij. Soms kan de machine direct deze informatie interpreteren ter nadere specificatie van een opdracht (dit was het geval bij de oude relaismachine ARRA), maar meestal is het aan speciale programma's overgelaten deze informatie te hanteren. Een eenvoudig voorbeeld van zulk een code is het volgende. Stel men wenst een programma de beschikking te geven over een lijst van priemgetallen. Het meest voor de hand ligt in het geheugen een lijst van priemgetallen op te nemen. Als N het grootste priemgetal uit de lijst is, moeten wij zo ongeveer N/log N getallen bergen. Men kan echter ook een aantal woorden aan elkaar gereid denken tot een grote cijferrij en deze als afbeelding van de natuurlijke getallen zien. Door een cijfer 0 dan wel 1 te kiezen al naar gelang het ermee

corresponderende getal al dan niet priem is, verkrijgen wij nu ook een getrouwe afbeelding van alle priemgetallen tot N. Het ligt trouwens voor de hand alle even getallen over te slaan (eventueel ook 3- en 5-vouden), zodat men N/2 cijfers moet bergen en als n het aantal cijfers per woord voorstelt heeft men N/2n woorden nodig. Is bijv. n = 30, dan heeft men N/60 woorden nodig, wat bijzonder goed afsteekt tegenover N/log N. De machine moet natuurlijk uit de woorden de afzonderlijke cijfers kunnen isoleren of er althans op opereren. Indien zij dit alleen kan langs zuiver arithmetische weg, dan is er niet veel reden om deze woorden niet als getallen te beschouwen. Evenwel kunnen de meeste machines ook operaties op de cijfers van een woord uitvoeren (collatie en schuiven) en indien het programma daarvan gebruik maakt is het woord niet als getal maar als codewoord te beschouwen.

Het is nu nog niet duidelijk hoe de machine kan weten op welke wijze een woord geïnterpreteerd moet worden. Dit wordt verklaard uit het volgend functionele schema, dat geldt voor de meeste machines voor zover het de programmeur betreft, hoewel het technisch bezien niet altijd geheel correct behoeft te zijn. De machine voert achtereenvolgens „handelingen" uit, welke handelingen bestaan uit een aantal deelhandelingen, die volgens een vast patroon in twee fasen verlopen.

Fase 1. De besturing B zoekt een (half) woord in het geheugen op het adres aangegeven in een speciaal register, de opdrachtteller T. Zij plaatst dit (half) woord in een ander register, het opdrachtregister R. Dit woord wordt nu geïnterpreteerd als opdracht. De besturing B hoogt vervolgens de inhoud van T op met $(\frac{1}{2})1$ en gaat over in fase 2.

Fase 2. De besturing „voert opdracht in R uit". Daartoe beschouwt B het functiegedeelte f van de opdracht, dat in het F-gedeelte van R is terechtgekomen en het numerieke gedeelte n van de opdracht, dat in het N-gedeelte van R is terechtgekomen. Aan de hand van f wordt n geïnterpreteerd.

a) Bij de zg. arithmetische opdrachten, d.w.z. van een zeker aantal waarden van f. als optellen, schoon inlezen, vermenig-

vuldigen, wordt opnieuw contact met het geheugen opgenomen. De besturing zoekt nu het woord in het geheugen op het adres aangegeven door n. Dit woord wordt als getal geïnterpreteerd. Het wordt naar het rekenorgaan gebracht en daar verwerkt op de door f aangegeven wijze, bijv. opgeteld bij wat zich al in de accumulator bevindt.

b) Bij de schrijfopdrachten wordt omgekeerd een getal uit het rekenorgaan (nader gespecificeerd door f) geschreven in het geheugen op het adres aangegeven door n.

c) Bij de zg. absolute opdrachten wordt geen contact meer opgenomen met het geheugen. Er geschiedt een of andere operatie in het rekenorgaan gespecificeerd door f en n samen; n is nu geen adres maar een codewoord.

d) Bij de zg. sprongopdrachten wordt de inhoud van N(dus n) getransporteerd naar T; n is nu dus weer als adres geïnterpreteerd, maar er is geen contact met het geheugen opgenomen. Na afloop van een van deze 4 deelhandelingen gaat de besturing over in fase 1.

Men ziet uit dit schema, dat zolang geen sprongopdracht wordt ontmoet de besturing een rij opdrachten uitvoert die zich op opeenvolgende adressen (ev. „halve" adressen) bevinden en wel in de daardoor gedefinieerde volgorde. Men zegt: de besturing voert een „programma" uit. Af en toe ontmoet de besturing een sprongopdracht. Het programma wordt nu voortgezet met de opdrachtenrij die begint op het adres gespecificeerd in de sprongopdracht. Dit kan een nieuw stuk programma zijn, maar ook het zojuist uitgevoerde stuk programma. Een deelprogramma kan dus meerdere malen worden uitgevoerd. Dat is ook wel nodig omdat anders de machine binnen korte tijd het gehele programma dat zich in het geheugen, waar slechts voor een eindig aantal opdrachten plaats is, zou hebben uitgevoerd en dus slechts korte tijd automatisch zou kunnen werken. Zo'n herhaling van een programma houdt overigens in het geheel niet in, dat nu precies hetzelfde rekenproces zou worden uitgevoerd. Een van de handelingen van het programma kan immers zijn het ophogen van een of meer fundamentele parameters, die een rol spelen in de berekening. Bijv. kan zo'n programma, dat steeds herhaald wordt f(x) uitrekenen en uittypen voor aequidistante waarden van x, dus bijv. een tabel van x^2 produceren. Ja zelfs behoeven de opdrachten van het programma na een doorloping ervan niet meer dezelfde te zijn als ervoor! Immers men kan een opdracht of opdrachten-

koppel, dat toch ook maar een woord is en dan ergens op een adres in het geheugen staat evenals de getallen met behulp van andere opdrachten naar het rekenorgaan sturen, daar wijzigen, bijv. door het numerieke gedeelte ervan met 1 te verhogen en weer op de oorspronkelijke plaats neerschrijven. Zo'n opdracht noemen wij een „variabele opdracht". De besturing heeft er geen weet van, dat het de rij cijfers die het als getal naar het rekenorgaan stuurt, daar laat verwerken en weer terugbergt in het geheugen, kortweg, die het „manipuleert" of „verwerkt" in een later stadium van de berekening zal ontmoeten als opdracht, die het moet „gehoorzamen". Deze consequentie echter van het opbergen van opdrachten en getallen in één enkel geheugen is van fundamenteel belang voor het karakter van de machine en is een van de voorwaarden van een werkelijk automatische machine. Het is nu dus duidelijk dat de machine een bepaald programma een aantal keren kan doorlopen en daarbij zelfs steeds wat anders kan uitrekenen maar nog niet hoe het een dergelijke cyclus opdrachten kan verlaten om een geheel ander deel van het totale programma uit te gaan voeren. Dit is met opdrachten van het besproken type weliswaar logisch mogelijk, maar verre van evident en zeer onpractisch. Vandaar dat nog een tweede element belangrijk is om een machine tot een volledig automatische te stempelen. Dit is nl. het aanwezig zijn van minstens één sprongopdracht waarvan de preciese werking afhangt van het al dan niet vervuld zijn van een conditie, bijv. als aan de conditie voldaan is fungeert de opdracht als gewone sprongopdracht, als aan de conditie niet voldaan is fungeert de opdracht als skipopdracht, d.w.z. fase 2 van de besturingscyclus bestaat uitsluitend in de overgang naar fase 1. Wat die conditie is varieert nogal. Vaak is het positief zijn van een getal in een van de registers van het rekenorgaan de conditie, bij de ARRA en ARMAC is het positief zijn van de inhoud van het conditieregister C, die overigens alleen maar bestaat uit een enkel tekencijfer. Dit register ontvangt zijn inhoud als nevenproduct van bepaalde opdrachten, die er het teken van een getal in kunnen laten onthouden.

Syllabus No.3 van de cursus 1955-'56:
Programmeren voor automatische rekenmachines
onder leiding van

Prof. Dr Ir A. van Wijngaarden en de Heer E.W. Dijkstra

HET GETAL

We zullen nu nog ingaan op de juiste samenhang van de grootte van een getal en de cijferrij die het getal representeert. Voorlopig is het voldoende alleen gehele getallen te beschouwen. Voorts zullen we ons beperken tot machines die in het tweetallig stelsel werken, omdat de uitbreiding tot een algemeen talstelsel achteraf gemakkelijk geschiedt. Stel de cijferrij heeft n binaire cijfers $d_0, d_1, d_2 \ldots . d_{n-1}$. Het is duidelijk, dat er slechts $N = 2^n$ verschillende cijferrijen zijn en dat dus ook slechts hoogstens N getallen onderscheiden kunnen worden. Op een of andere wijze moet men zich dus beperken wat betreft het aantal getallen, terwijl voorts om practische redenen zowel positieve als negatieve getallen kunnen worden voorgesteld. Het ligt voor de hand, in verband met deze laatste opmerking om alvast één cijfer, bijv. d_{n-1} te reserveren als tekencijfer bijv. $d_{n-1} = 0$ definieert een positief getal en d_{n-1} definieert een negatief getal, terwijl de resterende cijferrij $d_0, d_1, \ldots d_{n-2}$ op een of andere wijze een positief getal, nl. de absolute waarde van het getal definieert.
Dit stelsel sluit geheel aan bij onze normale schrijfwijze van gehele getallen. Er zijn inderdaad machines, waarbij dit stelsel wordt toegepast, maar er zijn ernstige technische nadelen aan verbonden. Zo moet de machine bijv. naast een opteller ook een aftrekker hebben. Een aantrekkelijker systeem is om alle getallen, die op een veelvoud van een ander getal $M \leqslant N$ na aan elkaar gelijk zijn, door één enkele cijferrij voor te stellen. Men werkt dus met een getalklasse modulo M in plaats van met getallen. Hierdoor is enerzijds een passende beperking bereikt, terwijl er ook anderzijds geen moeilijkheid meer is wat betreft positieve en negatieve getallen. Is bijv. M = 100, dan wordt er geen onderscheid gemaakt tussen de getallen +43, -57, +8143 en -357.

Er zijn nu nog twee vrijheden, nl. de keuze van $M \leqslant N$ en de juiste definitie van de koppeling van getalklasse en cijferrij. Wij beginnen met het laatste en definieren dat een getal D, congruent d modulo M ($0 \leqslant d < N$) wordt voorgesteld door:

$$d = \sum_{k=0}^{n-1} d_k 2^k .$$

Bij gegeven d ($0 \leq d < N$) liggen de cijfers d_k ondubbelzinnig vast. Als M = N, legt D de cijfers d_k ook ondubbelzinnig vast, als M<N kunnen bij één getal D meerdere representaties behoren, omdat er dan onder de getallen 0,1,2,....., N-1 zich meerdere getallen d kunnen bevinden, die met D congruent zijn modulo M.

Er zijn twee interessante waarden voor M, nl. $M = N = 2^n$ en $M = N-1 = 2^n-1$. Als M = N is er zoals reeds gezegd geen dubbelzinnigheid omtrent de representatie van D. Als men twee getallen op de gewone wijze bij elkaar optelt en een eventuele overdracht naar de niet bestaande plaats d_n onderdrukt, krijgt men het juiste resultaat, aangezien dan toch hoogstens precies N zou worden onderdrukt, hetgeen juist de modulus is. Als de som van twee getallen N is, zijn zij te beschouwen als elkaars tegengestelde, immers $N \equiv 0$ (mod. N). Om uit de gegeven representatie van D die van -D te verkrijgen verandert men eerst alle énen in nullen en alle nullen in énen (dan complementeren zij elkaar tot een rij énen, dat wil zeggen tot N-1) en telt er dan nog 1 bij op. Het omdraaien van het teken van een getal vereist dus een rekenkundige bewerking, nl. die optelling.

De tweede mogelijkheid, nl. M = N-1, het zg. inversensysteem, o.a. gebruikt in ARRA en ARMAC leidt allereerst tot één dubbelzinnigheid in notatie. Immers de rij nullen stelt 0 voor, maar de rij énen stelt N-1, dus ook 0 voor. Als men twee getallen optelt en er treedt een overdracht naar de niet bestaande plaats d_n op, dan moet men deze overdracht bij d_0 optellen. Zodoende laat men immers N weg, maar telt 1 bij en vermindert dus de som met de modulus N-1. Deze overdracht noemt men de ringoverdracht (end around carry). Als de som van twee getallen N-1 is, zijn zij te beschouwen als elkaars tegengestelde, immers $N-1 \equiv 0$ (mod. N-1). Om uit de gegeven representatie van D die van -D te verkrijgen, verandert men dus alle nullen in énen en alle énen in nullen.

Het voordeel van het gebruik van het inversensysteem is, dat het omkeren van het teken van een getal, wat een veelvuldig voorkomende operatie is, een zuiver administratieve operatie is. Wij zullen ons hier verder toe beperken, hoewel het systeem der echte complementen (M = N) op vrijwel gelijke wijze verder kan worden behandeld.

Zolang men zich beperkt tot optellen en aftrekken (dus optel-

tellen van het inverse!) behoeft men zich verder niet uit te spreken over de kwestie welk getal uit de getalklasse voorgesteld door de cijferrij, men werkelijk bedoelt. Zodra men echter de vermenigvuldiging introduceert, rijzen de moeilijkheden. Als product van twee getallen van n cijfers verwacht men nl. een getal van 2n cijfers, maar men zou natuurlijk het product ook alleen modulo M kennen, dus alleen aan de minst significante n cijfers zou men betekenis kunnen hechten. Theoretisch onmogelijk wordt het helemaal bij de deling. Daarom moeten we een conventie vastleggen, welke uit iedere getalklasse één enkel getal als echte „waarde" van de cijferrij vastlegt. Om practische redenen moeten zowel positieve als negatieve getallen voorkomen. De conventie is deze, dat altijd dat getal zal worden bedoeld, dat in absolute waarde zo klein mogelijk is. Dit betekent, dat als $d_{n-1} = 0$ is, $D = d$ wordt geinterpreteerd, terwijl als $d_{n-1} = 1$ is, geldt $D = d - M$, of in formule:

$$D = \sum_{k=0}^{n-2} (d_k - d_{n-1}) 2^k .$$

Het cijfer d_{n-1} fungeert dus toch weer als tekencijfer, want $d_{n-1} = 0$ wijst op een positief getal, $d_{n-1} = 1$ op een negatief getal. Alleen in tegenstelling met de primitieve schrijfwijze: tekencijfer, absolute waarde, wordt nu een negatief getal verkregen door niet alleen het tekencijfer, maar ook alle andere cijfers te inverteren. Als n = 5 bijv. stelt 00011 het getal 3 voor en 11100 het getal -3. De twee representaties van nul, nl. de rij nullen resp. de rij enen hebben nu ook een betekenis, nl. die van +0 resp. -0. De nul kan dus zowel bij de positieve getallen worden geteld, als bij de negatieve en onderscheidingen als bijv. > 0; $\gtrsim +0$; $\geqslant 0$; $\leqslant 0$; $\lesssim -0$; < 0 zijn alle verschillend.

Er zij nog eens op gewezen, dat deze verenging van getalvoorstelling alleen strikt noodzakelijk is, als vermenigvuldiging en deling een rol spelen en overigens alleen slechts een gemakkelijke conventie is. Als men bijv. een rij getallen bij elkaar optelt, waarvan men op een of andere wijze weet, dat hun som ligt tussen $\frac{1}{2}N$ en $-\frac{1}{2}N$, dan levert de machine het antwoord correct af, ook al vallen allerlei partiele sommen buiten deze capaciteit.

Alle getallen D voldoen nu dus verder aan

$$-2^{n-1} + 1 \leqslant D \leqslant 2^{n-1} - 1.$$

Dit heeft een ander merkwaardig resultaat. Immers het product van twee getallen is nu in absolute waarde kleiner dan 2^{2n-2}, dus als we het met dezelfde conventies voorstelden, zouden we slechts 2n-1

cijfers nodig hebben, dus één cijfer minder dan met twee normale woordlengten overeenkomt. Dit past niet goed in ons algemene schema van woorden met vaste lengte en daarom wordt kunstmatig een extra cijfer toegevoegd en wel als volgt. Een „dubbele-lengte getal" E wordt voorgesteld door de cijferrij $e_0, e_1 \ldots\ldots e_{2n-1}$, met de nevenconditie $e_{n-1} = e_{2n-1}$. Deze rij wordt gesplitst gedacht in de rij $e_0, e_1, \ldots\ldots e_{n-1}$ en de rij $e_n, e_{n+1}, \ldots\ldots e_{2n-1}$, waaraan op de normale wijze de getallen E_0 en E_1 worden toegevoegd, die dus dankzij de nevenconditie hetzelfde teken hebben.

Tenslotte wordt gedefinieerd

$$E = E_1 . 2^{n-1} + E_0 .$$

De getallen E_0 en E_1 zijn dus de „cijfers" van E in het „2^{n-1}-tallig" stelsel, waarbij de cijfers overigens zelf van een (zelfde) teken zijn voorzien. Natuurlijk kan men deze gedachtengang nu direct overdragen op getallen van willekeurige grootte. Het grote voordeel van dit systeem is, dat men ondanks het feit, dat de woordlengte vast is, toch weer met getallen van willekeurige grootte kan werken, terwijl alle brokstukken van het grote getal weer normale getallen zijn. Afgezien van de eindige grootte van het geheugen van de machine zijn er geen grenzen aan de grootte van de getallen waarmee de machine kan werken.

De beperking dat de cijfers $E_0, E_1, \ldots\ldots$ hetzelfde teken hebben is niet noodzakelijk. Het is alleen een gemakkelijke conventie, waaraan men goed doet zich zo veel mogelijk te houden, zoals later duidelijk zal worden.

Tot zover de behandeling van het gehele getal. Evenals op een gewone tafelrekenmachine is voor het begrip van de plaats van de komma bij breuken geen speciale apparatuur noodzakelijk. Deze kommaplaats kan men zelf interpreteren waar men wenst. Het eenvoudigst is het meestal (bijv. voor ARRA en ARMAC) de komma te denken tussen het tekencijfer d_{n-1} en het meest significante echte cijfer d_{n-2}, d.w.z. men denkt zich het gehele getal gedeeld door 2^{n-1}. Aangezien het gehele getal in absolute waarde juist kleiner was dan 2^{n-1}, betekent dit, dat de breuk in absolute waarde juist kleiner dan 1 is. De kleinste echt positieve breuk is 2^{-n+1}. Dit is een maat voor de precisie of onderscheidingsvermogen. We noemen haar de peuter.

Het is goed om enkele schrijfwijzen te hebben, waarmee men de interpretatie van het woord kan aangeven. Zijn x en y gehele getallen, dan zij:

(x) De inhoud van adres x, gewoon een woord zonder nadere interpretatie;

$[x]$ de inhoud van adres x, als geheel getal geinterpreteerd;

$\{x\}$ de inhoud van adres x, als breuk geinterpreteerd;
$\{x\} = [x].2^{-n+1}$;

$[x,y]$ de inhoud van adressen x en y, als dubbele lengte geheel getal geinterpreteerd, waarbij $[x,y] = [x]\, 2^{n-1} + [y]$;

$[x,y\} = [x] + \{y\}$;

$\{x,y\} = \{x\} + \{y\}.2^{-n+1}$.

Natuurlijk gaat het niet aan steeds nieuwe haakjes te bedenken om getallen van meer dan dubbele lengte weer te geven. Evenwel blijkt het erg prettig om de meest voorkomende gevallen van enkele en dubbele lengte te onderscheiden, omdat dit een belangrijke rol speelt in de ingebouwde arithmetiek van de machine.

Syllabus No.4 van de cursus 1955-'56:

Programmeren voor automatische rekenmachines

onder leiding van

Prof. Dr. Ir. A. van Wijngaarden en de Heer E.W. Dijkstra

DE OPDRACHT.

De preciese samenhang tussen de cijferrij en het woord als opdracht(en) geïnterpreteerd en de betekenis van die opdracht(en) is zo sterk verschillend bij verschillende machines, dat het nauwelijks mogelijk is een verantwoord overzicht te geven. Anderzijds is het uit ervaring gebleken, dat iemand, eenmaal met de „opdrachtencode" van een bepaalde machine vertrouwd, snel die van een andere machine leert en vooral beter de merites ervan begrijpt. Daarom zullen we ons hier beperken tot de code van een der Nederlandse machines en wel van de ARRA, o.a. omdat de code daarvan wellicht het eenvoudigste is en weinig kennis van de machine zelf vereist. Enkele te speciale opdrachten zullen we zelfs weglaten in onze beschouwing.

De ARRA is een machine, werkend in het tweetallig stelsel, inversensysteem, n=30. Het rekenorgaan bezit twee nagenoeg identieke registers, geheten A en S, die ieder een volledig getal van 30 cijfers bergen. Bovendien is er nog een register C, dat slechts 1 cijfer bergt, geheten conditieregister. De inhoud (C) noemen we „de conditie". Zij is 0 oftewel positief dan wel 1 ofwel negatief. Er is een register G, getalschakelaar, waarin met de hand door omzetting van schakelaars een woord van 30 cijfers kan worden vastgelegd.

Een opdracht beslaat een halfwoord (15 cijfers) en bestaat uit een 5 cijferig functiegedeelte f en een 10 cijferig numeriek gedeelte n, welke beiden als geheel positief getal geïnterpreteerd worden. De opdracht als geheel heeft een getalwaarde 1024f + n, omdat f aan de meest significante zijde van n staat. Twee opdrachten vormen één woord van 30 cijfers, de a-opdracht en de b-opdracht. De b-opdracht bevindt zich aan de meest significante zijde, zodat het opdrachtenkoppel als positief getal geïnterpreteerd gelijk is aan

$$32768\ (1024\ f_b + n_b) + (1024\ f_a + n_a).$$

De mogelijke waarden van f zijn 0(1)24. Als f = 25(1)31 fungeert de opdracht als „onbestaanbare opdracht" en de machine stopt. Dit heeft het voordeel, dat als het programma derailleert en de machine tracht getallen als opdrachten te gaan lezen, dit

doorgaans snel tot stoppen leidt. In het bijzonder werkt het getal -0 als stopopdracht van onschuldig karakter.

De functie van de opdrachten wordt aangegeven, door de inhoud van het (de) register(s) welke verandert te specificeren. Een accent duidt op de inhoud na afloop van de operatie, geen accent op de inhoud voor de operatie. Zo niet speciaal vermeld wordt de inhoud (T) van de opdrachtteller met "$\frac{1}{2}$" vermeerderd d.w.z. als $(T) = n_a$, dan $(T)' = n_b$, en als $(T) = n_b$ dan $(T)' = n + 1_a$.

De belangrijkste opdrachten luiden:

0,n	$(A)' = (A) + (n)$,	(optellen in A);
1,n	$(A)' = (A) - (n)$,	(aftrekken in A);
2,n	$(A)' = (n)$,	(schoon in A);
3,n	$(A)' = -(n)$,	(negatief schoon in A);
4,n	$(n)' = (A)$, $(C)' = \mathrm{sgn}(n)'$,	(schrijven uit A);
5,n	$(n)' = -(A)$, $(C)' = \mathrm{sgn}(n)'$,	(negatief schrijven uit A);
6,n	$(T)' = n_a$ als $(C) = 0$, $(T)' = (T) + \frac{1}{2}$ als $(C) = 1$,	(conditionele a-sprong);
7,n	$(T)' = n_a$,	(a-sprong);
8,n	$(S)' = (S) + (n)$,	(optellen in S);
9,n	$(S)' = (S) - (n)$,	(aftrekken in S);
10,n	$(S)' = (n)$,	(schoon in S);
11,n	$(S)' = -(n)$	(negatief schoon in S);
12,n	$(n)' = (S)$, $(C)' = \mathrm{sgn}\ (n)'$	(schrijven uit S);
13,n	$(n)' = -(S)$, $(C)' = \mathrm{sgn}\ (n)'$,	(negatief schrijven uit S);
14,n	$(T)' = n_b$ als $(C) = 0$, $(T)' = (T) + \frac{1}{2}$ als $(C) = 1$,	(conditionele b-sprong);
15,n	$(T)' = n_b$	(b-sprong);
16,n	$[AS]' = [S][n] + [A]$,	(additief vermenigvuldigen);
17,n	$[AS]' = -[S][n] + [A]$,	(subtractief vermenigvuldigen);
18,n	$[AS]' = [S][n]$,	(schoon vermenigvuldigen);
19,n	$[AS]' = -[S][n]$,	(negatief schoon vermenigvuldigen);
20,n	$[AS] = [S]'[n] + [A]'$, $\mathrm{sgn}\ [A]' = \mathrm{sgn}\ [AS]$, $\lvert [A]' \rvert < \lvert [n] \rvert$,	(delen);
21,n	$[AS] = -[S]'[n] + [A]'$, $\mathrm{sgn}[A]' = \mathrm{sgn}[AS]$, $\lvert [A]' \rvert < \lvert [n] \rvert$,	(negatief delen);
22,n	$[AS]'$ = geheel gedeelte van $[A]2^{29-n}$, $n = 0(1)31$,	(A-schuif);

23,n	$[SA]'^{*}$ = geheel gedeelte van $[S]2^{29-n}$, n = 0(1)31,	(S schuif);
24,0		(Skip);
24,2	Stop als (C) = 0,	(positieve stop);
24,3	Stop als (C) =(1),	(negatieve stop);
24,6	$\langle A\rangle = \begin{cases} 7, & [T] + 1 \\ 0, & 0 \end{cases}$,	(Subroutinesprongvoorbereiding);
24,7	$[A]' = [G]$,	(legen uit getalschakelaars);
24,16	$[AS]' = 10[S] + [A]$,	(schoon maal tien);
24,17	$[AS]' = 10[S]$,	(maal tien);
24,40	$(C)' = \mathrm{sgn}\ (A)$,	($A \geqslant +0$?);
24,104	$(C)'=1$ als $[A] = -0$; $(C)'= 0$ als $[A] \neq -0$,	($A \neq -0$?);
24,n	$512 \leqslant n \leqslant 767$, $[A]' = [A]+n-512$,	(snelle optelling);
	$768 \leqslant n \leqslant 1023$, $[A]'=[A]+n-1023$,	(snelle aftrekking);

Weggelaten zijn in dit overzicht een aantal opdrachten uit de 24-groep (communicatie-opdrachten), welke speciaal van belang zijn voor het bandlezen, bandponsen, typen en controleren van het getypte, waarmee de programmeur practisch nooit te maken heeft.

Een paar kleine opmerkingen moeten nog worden gemaakt om een en ander te preciseren. Allereerst de vraag, of een resultaat 0 afgeleverd wordt als +0 of -0. Bij sommige opdrachten zijn precieze tekenregels meegegeven, waarmee ook over de 0 beslist is. Bij de overige opdrachten geldt, dat -0 het normale antwoord is. Men kan zelf nagaan wanneer +0 ontstaat door zich te realiseren hoe de optelling en de aftrekking verloopt. Deze uitzonderingen luiden:

$(+0) + (+0) = +0$
$(+0) - (-0) = +0$
$(+0) \times n = +0$ als $n \geqslant +0$
$(-0) \times n = +0$ als $n \leqslant -0$

De „schone" opdrachten 2,3,10,11 denke men zich als schoonmaken tot +0 gevolgd door optellen resp. aftrekken.

Bij de schuifopdracht 23,n is $[SA]'^{*}$ geschreven i.p.v. $[SA]'$, omdat in tegenstelling tot de normale afspraken over een dubbele lengtegetal het tekencijfer van A niet als gewoon tekencijfer geldt, maar veeleer $[SA]^{*}$ een getal met één tekencijfer (dat van S) gevolgd door 59 gewone cijfers is, nl. 29 van S en 30 van A.

Een rij opdrachten vormt een programma. We zullen een aantal eenvoudige kleine programma's neerschrijven:

1. Plaats het product $[300][400]$ op adres 500. Gegeven is $|[300][400]| < 2^{29}$.

   10,300
   18,400
   12,500

2. Plaats het product $\{300\}\{400\}$ op adres 500.

   10,300
   18,400
   4,500

3. Plaats het afgeronde product $\{300\}\{400\}$ op adres 500.

   10,300
   18,400
   4,500
   23,29
   8,500
   12,500

4. Deel $[300]$ door $[400]$, plaats het quotiënt op adres 500 en de rest op adres 600.

   10,300
   23,0
   20,400
   12,500
   4,600

5. Deel $\{300\}$ door $\{400\}$, plaats het quotiënt op adres 500. Gegeven is $|\{300\}| < |\{400\}|$.

   4,300
   22,0
   20,400
   12,500

6. Deel $\{300\}$ door $\{400\}$, plaats het afgeronde quotiënt op adres 500. Gegeven is, dat $|\{300\}|$ en $|\{400\}|$ hetzelfde teken hebbe en dat $|\{300\}| < |\{400\}|$.

   10,400
   23,1
   2,300
   20,400
   12,500

7. Wat doet het volgende programma op adressen 300 - 305?

```
300    22,0        360    0,200
        2,360             0,200
301     1,361      361    8,700
        5,302             8,701
302     0,0        362    8,698
        0,0               8,699
303     0,362
       24,40
304     6,301
       23,30
305    24,104
       24,2
```

Syllabus No.5 van de cursus 1955-'56:
Programmeren voor automatische rekenmachines
onder leiding van

Prof.Dr. Ir. A van Wijngaarden en de Heer E.W. Dijkstra

BLOKSCHEMA'S

Als de machine aan een bepaald probleem rekent, bevat het geheugen een programma van honderden, soms duizenden opdrachten. Deze opdrachten bepalen volkomen, wàt er in onderscheiden gevallen gebeure, zij leggen de berekening volledig vast, en wel in de „taal" die voor de machine „direct verstaanbaar" is.
Voor de geinteresseerde, die het programma inkijkt, zelfs voor de programmeur, die het programma opgesteld heeft, houdt deze „verstaanbaarheid" echter niet over: het lezen van een programma van de opdrachten alleen, zonder een blik te slaan in de explicatie, die er gelukkig doorgaans wel naast staat, vergt erkend veel geduld en doorzettingsvermogen. Dit is wel te verklaren.
Een van de redenen, dat men gauw in de veelheid van opdrachten omkomt, is zeker, dat er dikwijls veel opdrachten nodig zijn, om te berekenen, wat de lezer als één logisch geheel beschouwt (bv. x^5 of siny). Een tweede oorzaak, waardoor de structuur van het programma neigt schuil te gaan, is daarin gelegen, dat het programma belast is met veel irrelevante informatie: de plaats, waar alle opdrachten en getallen staan, ligt in een feitelijk programma vast, voor dezelfde gang van de berekening het geheugen best anders ingedeeld had kunnen zijn en alles net zo goed op andere adressen had kunnen staan. Het belangrijkste is echter wel, dat in het programma <u>wel</u> staat, wat er <u>gebeurt</u>, maar <u>niet</u>, wat dit nu allemaal <u>behelst</u>: er staat, onder welke omstandigheden een conditionele besturingsverplaatsing gehoorzaamd wordt, wat echter de zin van deze speciale omstandigheden is, wordt aan de intelligente lezer overgelaten....... Kortom er is een behoefte aan een overzichtelijke weergave van programma's, een notatie, waarin, dankzij verzwijging van de bijkomstigheden, de essentialia niet verdrinken en waar tevens de <u>functie</u> van de stukken programma (en van de „voorzorgen") in aangegeven is.
Een dergelijke notatie bewijst onmisbare diensten, niet alleen bij het bestuderen en bespreken, maar ook bij het maken van een programma; de notatie der zg. <u>blokschema's</u> (= flow diagrams) voorziet in deze behoefte. Dat deze notatie geen direct gebruik maakt van de opdrachtencode van de betroffen machine, verzekert een niet te verwerpen algemeenheid van de blokschema's. Ander-

zijds is de notatie iets minder efficiënt bij uitgekookte, „getructe" programma's: bij standaardprogramma's (bv. voor het berekenen van de logarithme of de tweede machtswortel), bij welker opstelling men het onderste uit de kan wil halen, zal men immers niet schromen, geraffineerd van de - vaak onbedoelde! - speciale eigenschappen van de machine gebruik te maken. Het antwoord op de vraag „waarom nu juist zo?" is hier doorgaans niet te duidelijk zichtbaar, nochtans bewijst ook hier, zoals wij hopen te illustreren, de methode goede diensten.

Ten eerste worden ten dienste van de overzichtelijkheid de opdrachten in zg. blokken ingedeeld, d.w.z. groepen opdrachten, die een onafscheidelijk geheel vormen, onafscheidelijk in die zin, dat bij een rondgang(etje) van het werkende programma deze opdrachten of allemaal, of geen van allen gehoorzaamd worden. Anders beschouwd: de besturing doorloopt het programma langs bepaalde routes (één-richtingsverkeer!), overal waar routes samenvloeien of splitsen, is een zg. knooppunt. Een splitsing is in eerste instantie altijd tweevoudig (conditionele besturingsverplaatsing), anderzijds kunnen in een knooppunt door samenvloeiing willekeurig veel routes samenkomen (men kan van vele punten in de machine de besturing er naar toe laten springen). Een blok nu loopt in eerste instantie van knooppunt tot knooppunt: een blok heeft dus één ingang en één uitgang. De uitgang is enkel- of meervoudig (enkelvoudig als de ingang alleen een knooppunt door splitsing is), de uitgang kan enkel- òf tweevoudig zijn (tweevoudig, als de uitgang een knooppunt door splitsing is, enkelvoudig, als het alleen een knooppunt door samenvloeiing is).

Men tekent blokken door rechthoeken: we spreken af, dat de ingang van elk blok bovenaan, de uitgang onderaan getekend wordt; deze conventie maakt pijltjes in de connecties tussen de blokken overbodig. Als de uitgang tweevoudig is, staat er boven een vraag en we spreken af, dat de besturing het blok via de uitgang rechts onder verlaat, als het antwoord op de vraag bevestigend luidt, anders via de uitgang links onder.

Een stukje blokschema kan er dus als in fig. 1 uitzien. In de blokken (waarin nu alleen een nummer staat) wordt de handeling omschreven (zie onder), de pijltjes in de verbindingen zijn hier dit keer wel getekend.

De besturing komt - uit een extern stuk programma - links boven in blok 1; blok 1 is een cyclus, want het kan enige malen herhaald worden; bij het verlaten van de cyclus doorloopt de besturing blok 2, waarna aan de hand van een of ander criterium

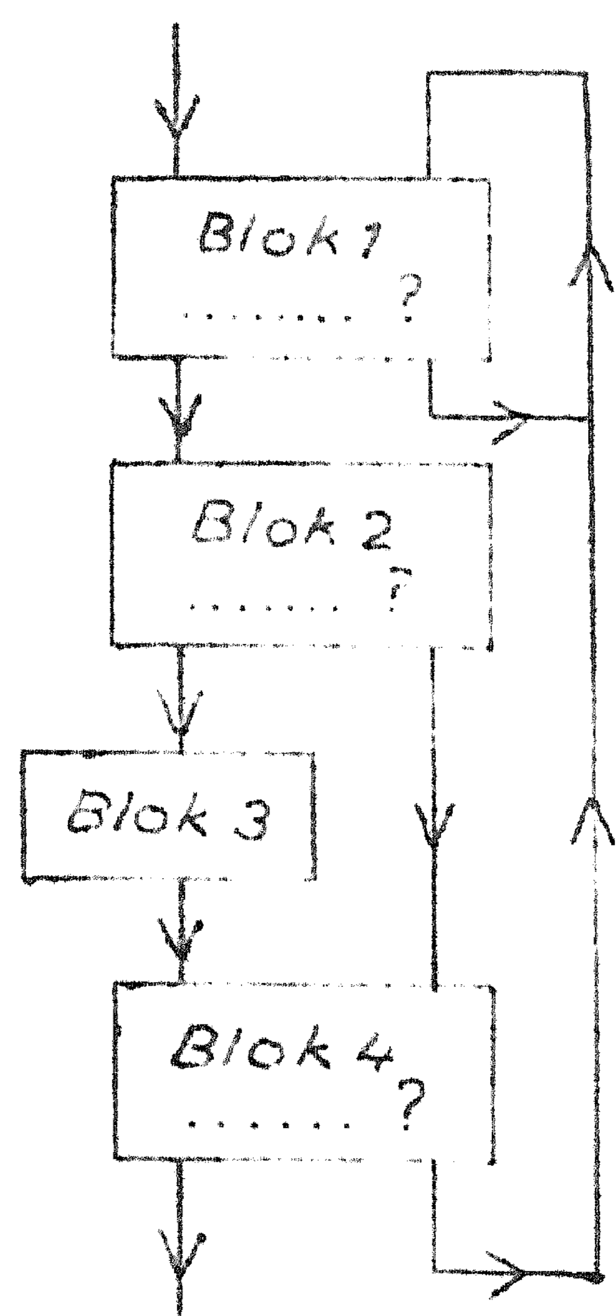


fig.1

uitgemaakt wordt, of blok 3 overgeslagen wordt of niet. Na blok 4 wordt gekeken, of de besturing weer naar blok 1 gestuurd moet worden, of dat dit stuk programma verlaten kan worden (Blok 1,2,3 en 4 kunnen bv. samen een iteratie-schema zijn, en blok 1 een of ander cyclusje.)

Als werkelijk voorbeeld volgt het blokschema, waarmede getest wordt of x = 0 is, in een machine, waarin de directe decisie gemaakt wordt op het non-negatief zijn (fig. 2 en fig. 3). Hierbij is p (fig. 3) het kleinste positieve getal, dat de machine hanteren kan.

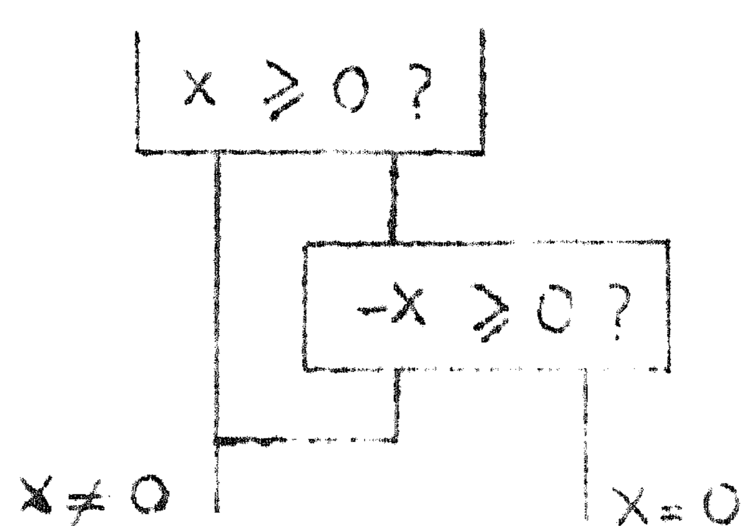


fig.2

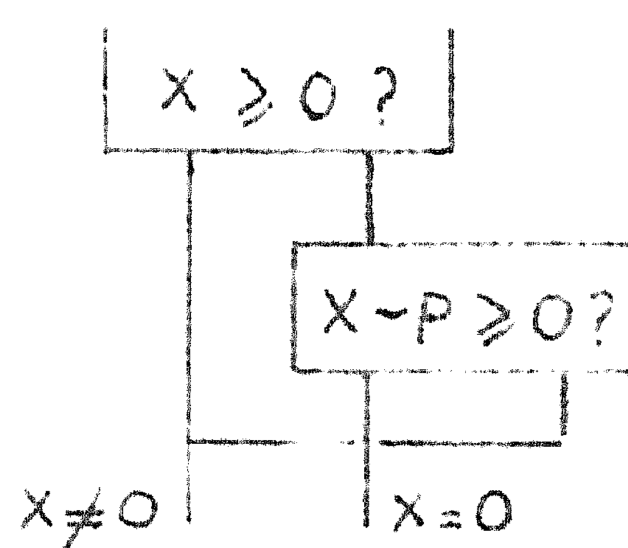


fig.3

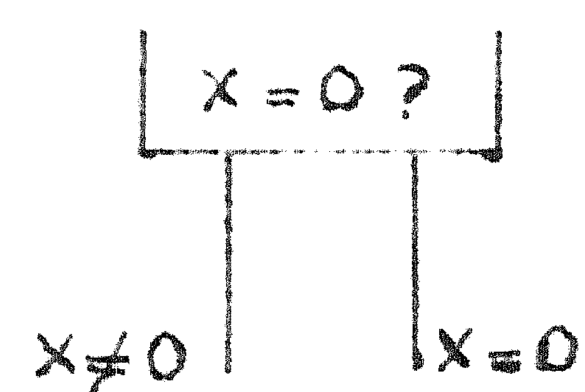


fig.4

Opm.: Ook al heeft de machine geen directe „nultest", toch zal dan de programmeur fig. 2 of fig. 3 in het blokschema vaak comprimeren tot de notatie als in fig. 4. Hetzelfde kan hij doen met fig. 1: als dit de (bekende standaard- of elders precies beschreven) operatie A bewerkstelligt, kan dit in de vorm van fig. 5 gecomprimeerd worden. Het verkort aangevende blok moet wel dezelfde aansluitingen met het externe programma hebben! Hiermede is het begrip blok uitgebreid: om het aantal blokken te verminderen, worden enige „minder belangrijke" knooppunten verwaarloosd.

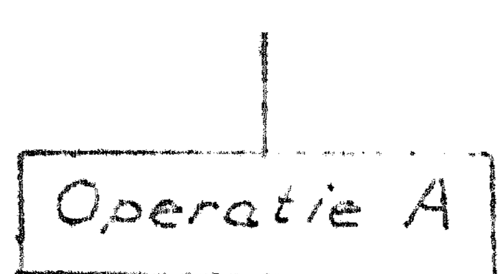


fig.5

De notatie van de decisie's is hiermede aangegeven, echter moet ook voor de arithmetische bewerkingen in elk blok een duidelijke notatie zijn. Een notatie, die slechts voor één uitleg vatbaar is, en nauw aansluit bij de organisatie van vele machines, ontlenen wij aan H. Rutishauser: het „gerichte gelijkteken": $\Rightarrow$ (lees „vervangt"), dat gebruikt wordt ter definitie van een grootheid.

Voorbeelden: $a + b \Rightarrow c$, d.w.z. van nu af aan wordt tot nader aankondiging met c bedoeld a + b; dit wordt verwezenlijkt, doordat er een geheugenplaats gereserveerd wordt voor c, waarin door de opdrachten in dit blok de som a + b geplaatst wordt.

Het programma mag samenvloeien met een ander stuk (route), waarin c door iets anders vervangen is; welke c gebruikt wordt, hangt dus af van de <u>voorgeschiedenis</u>.

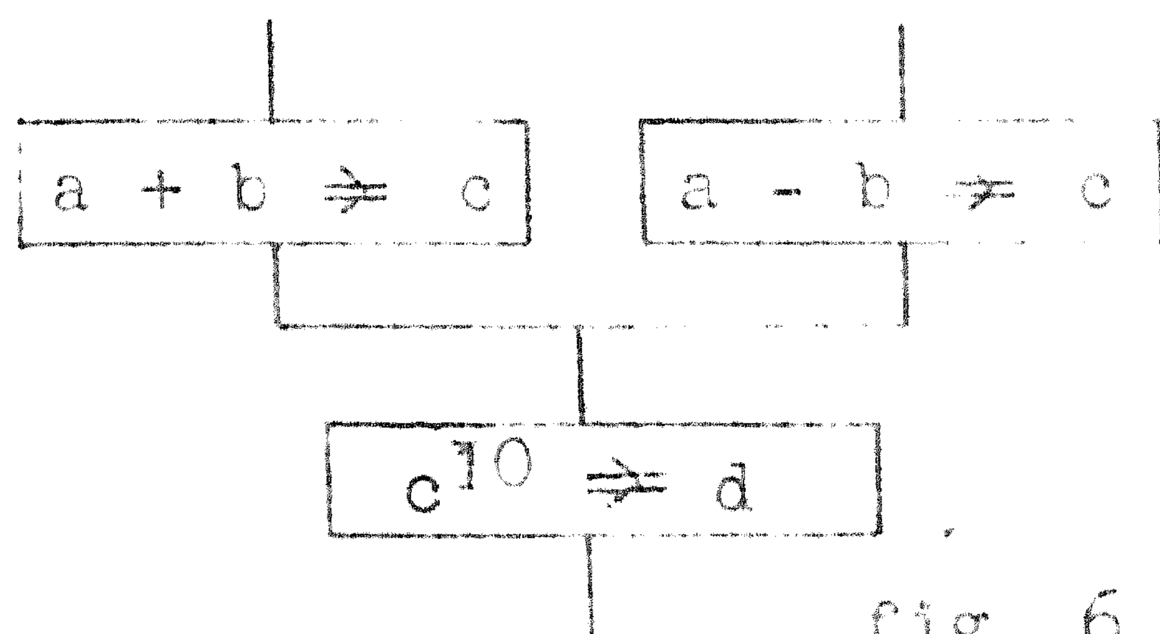


fig. 6

In fig. 6 wordt $(a \pm b)^{10} \Rightarrow d$ uitgerekend, afhankelijk van de ingang (dus van vorige decisie's). Het is bij het <u>gerichte</u> gelijkteken toegestaan, dat links en rechts hetzelfde argument voorkomt: $-p \Rightarrow p$, betekent, dat terwijl p al een of andere waarde had, bij deze route p van teken gewisseld moet worden. Fig. 7 illustrere deze notatie in het blokschema, dat met behulp van de decisie op non-negatief de operatie $|p| \Rightarrow p$ bewerkstelligt, dus p door zijn absolute waarde vervangt.

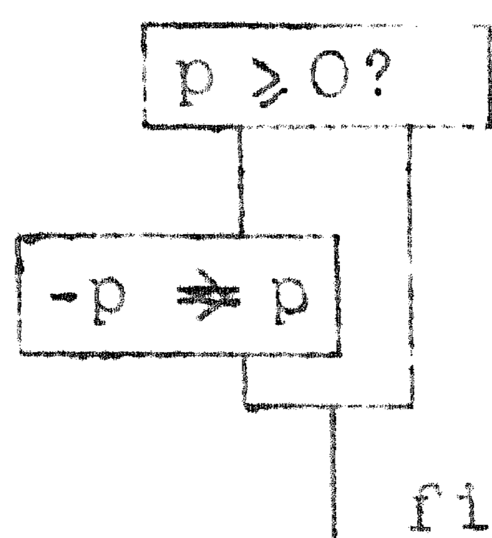


fig. 7

Thans volgt als voorbeeld de berekening van de tweedemachtswortel uit een echte breuk (d.w.z. abs. kleiner dan 1).

<u>Blokschema $|x|^{\frac{1}{2}} \Rightarrow y$</u>; iteratief (met gebruikmaking van de deling).

Hier wordt gebruik gemaakt van de iteratie-formule (2e orde)

$$y_{n+1} = \tfrac{1}{2}\left(y_n + \frac{|x|}{y_n}\right) = y_n + c_n, \text{ met } c_n = \tfrac{1}{2}\left(\frac{|x|}{y_n} - y_n\right)$$

$y_0 = \frac{1}{2} + \frac{1}{2}x$ (òf een waarde, waarvan we mogen verwachten, dat hij als startwaarde y_0 gunstiger is: in vele gevallen immers worden vele wortels getrokken, maar uit een continu veranderend argument $|x|$; maar dan is het antwoord, dat de vorige worteltrekking heeft achtergelaten, waarschijnlijk de beste startwaarde, waar we over beschikken. De test of de vorig afgeleverde y als y_0 bruikbaar is, is of deze y groter is dan $|x|$, omdat anders de eerste deling een quotiënt buiten de capaciteit af zou leveren.

Het blokschema is voor een machine, die werkt met vaste decimale (c.q. binale) punt, terwijl de capaciteit ligt tussen + 1 en - 1; Zodra $c_n = 0$ is, wordt er niet meer geitereerd; hierbij voert de machine een iteratie teveel uit; anderzijds duren iteraties in verband met de eenvoud van de test - als de expliciete nultest is ingebouwd - korter. Het blokschema is in fig. 8 weergegeven.

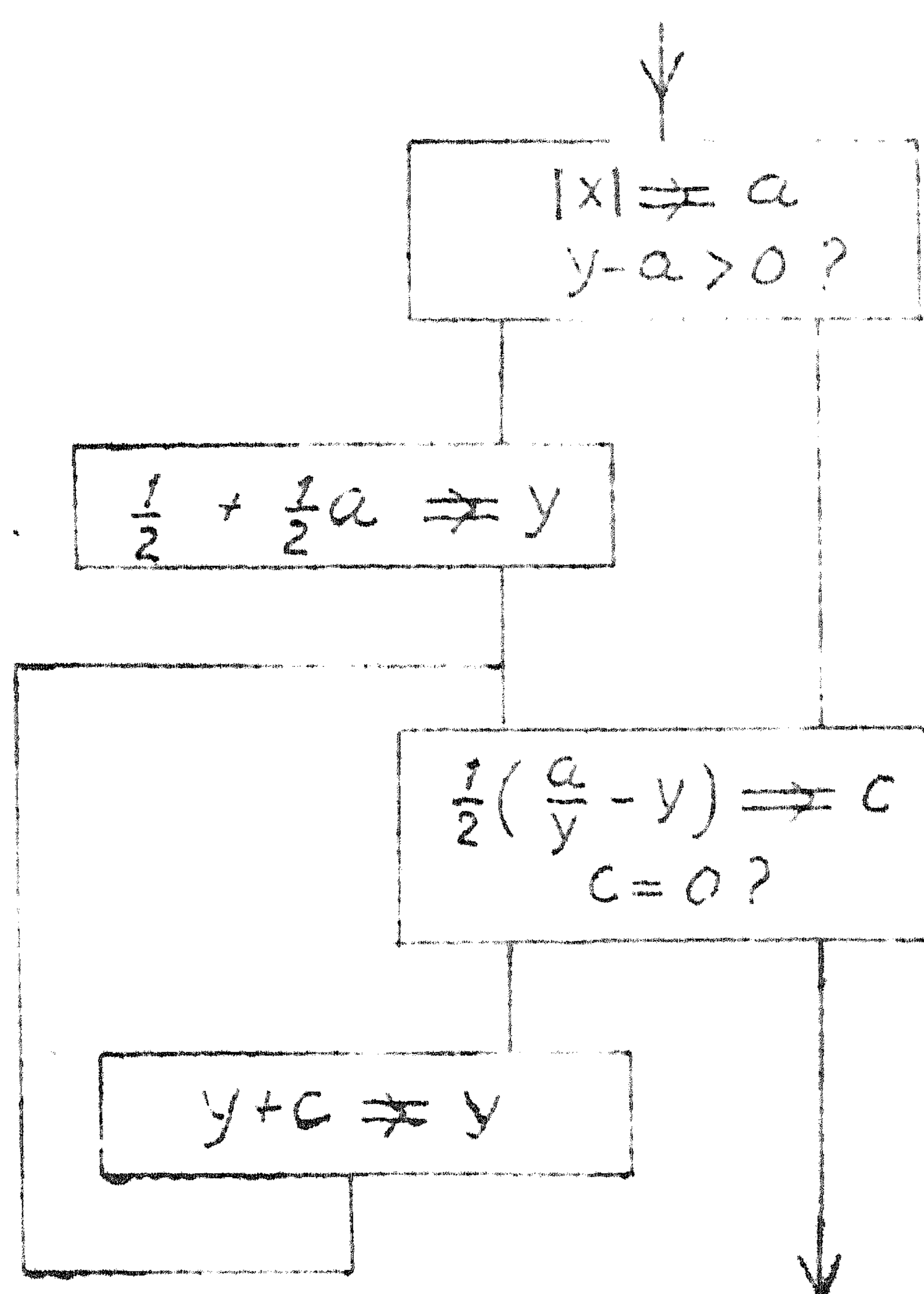


fig. 8 Blokschema $|x|^{\frac{1}{2}} \Rightarrow y$.

Als volgend voorbeeld behandelen we de berekening van een n^{de} graads-polynoom en kiezen in ons voorbeeld $n = 3$. De opgave luidt dus: $a_0 + a_1y + a_2y^2 + a_3y^3 \Rightarrow z$. Als we eerst de machten van y berekenen, deze met de bijbehorende a's vermenigvuldigen en de producten bij a_0 optellen, kost de berekening 5(=2n-1) vermenigvuldigingen, terwijl 3(=n) voldoende is, als men „bij de hoogste macht begint", d.w.z. als men de formule herschrijft tot $a_0 + y\,a_1 + y(a_2 + a_3y) \Rightarrow z$. Het blokschema is in fig. 9 weergegeven; het symbool $\hookrightarrow$ is het gerichte gelijkteken, speciaal met betrekking tot indices (H. Rutishauser). Fig. 10 en fig. 11 geven blokschema's, die arithmetisch hetzelfde verrichten. Bij de methode in fig. 11 is de benodigde programmaruimte wel afhankelijk van n, dankzij het achterwege blijven van de telling in j gaat de berekening mogelijk sneller.

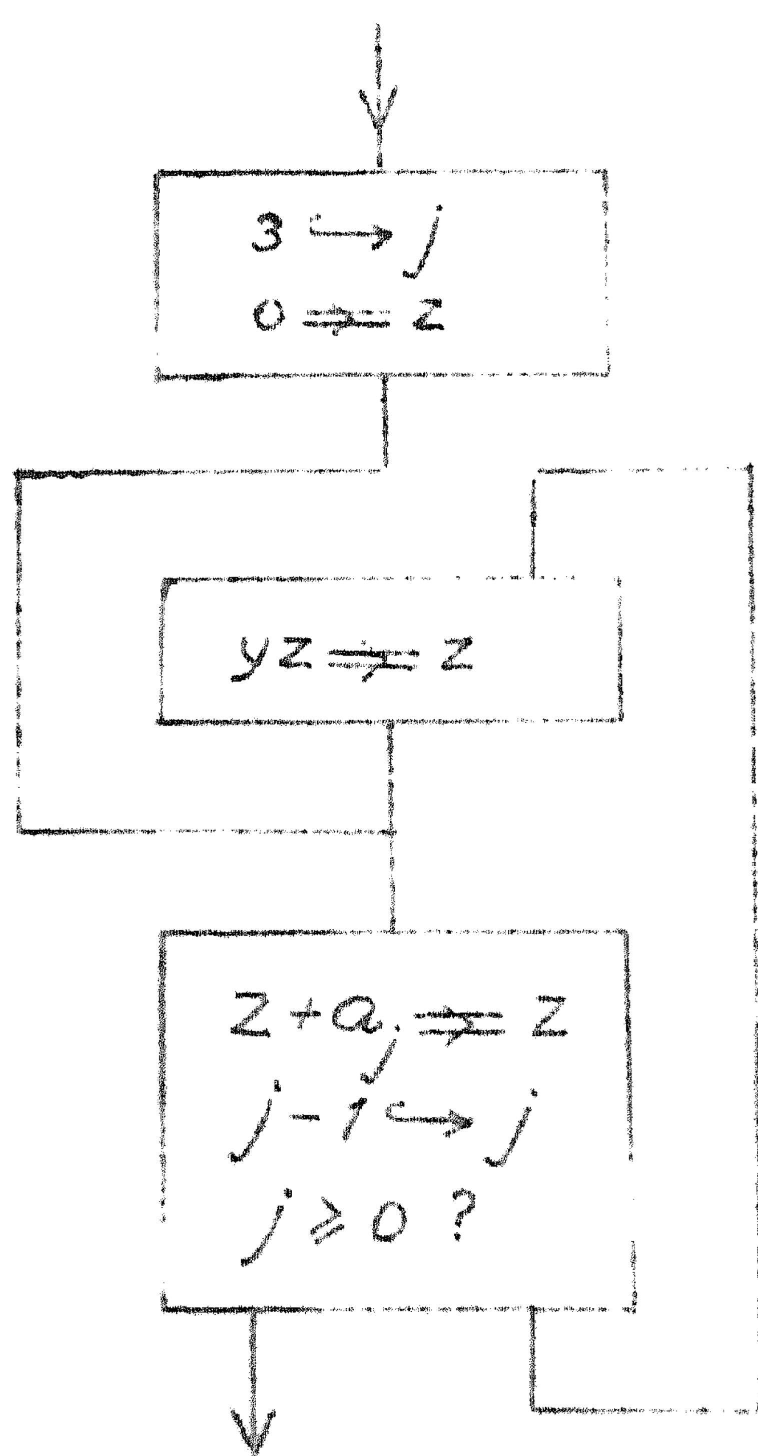


fig. 9 $\sum_{j=0}^{2} a_j y^j \Rightarrow z$

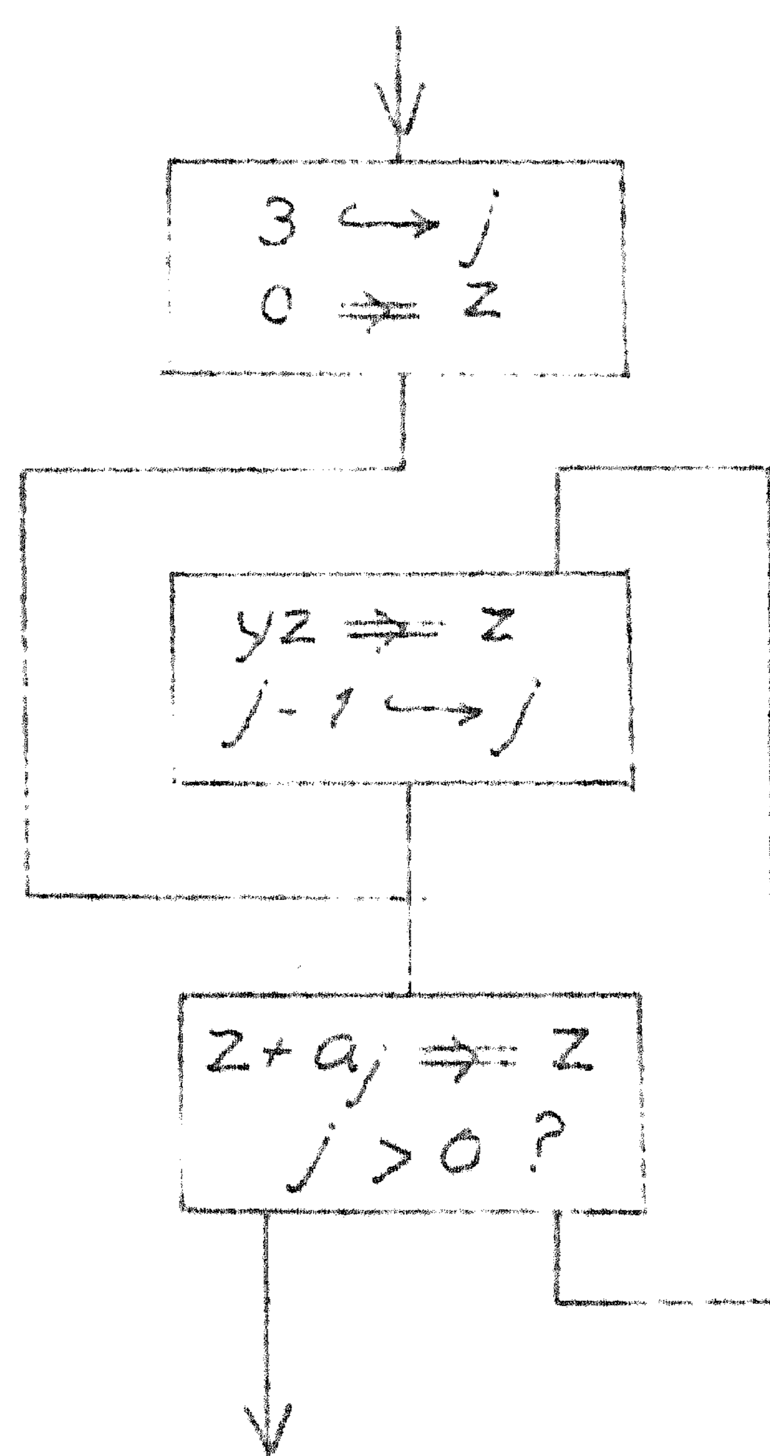


fig. 10

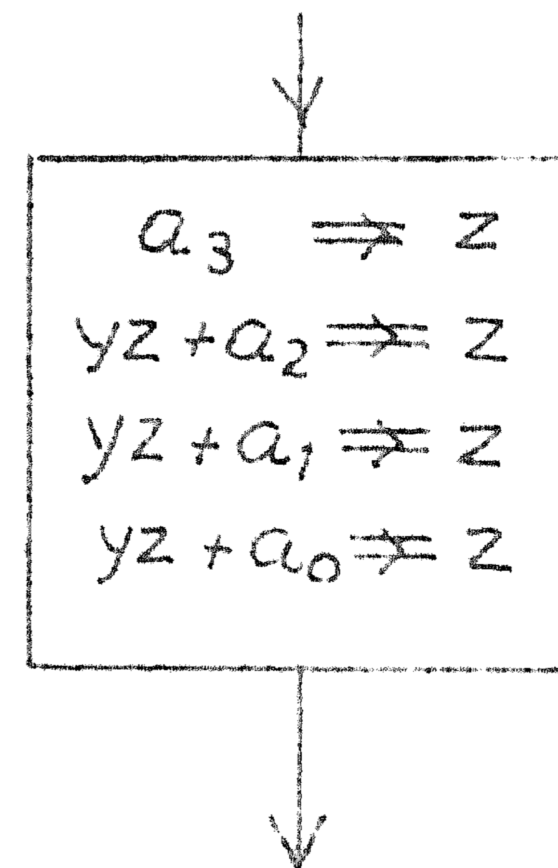


fig. 11

In ARRA-code zou fig. 10 er bv. als volgt uit kunnen zien:

```
150:   a0       ⇒ 160:   2/154        3 ⇒ A
                         10/156       0 ⇒ S
151:   a1         161    15/163  ⇒
           166b ⇒        18/158       yS ⇒ A
152:   a2         162    22/29        A → S
                          2/159       j ⇒ A
153:   a3         163     1/155       j-1 ⇒ A
           161a →         4/159       (j-1 =)A ⇒ j
154:   + 3        164     0/157       vormt  var. opdracht
                          4/165       plaatst var. opdracht.
155:   + 1        165    (8/153-150   S + aj ⇒ S
                          2/159)      j ⇒ A
156:   + 0        166     4/159       j > 0?
                         14/161 →     ja
157: 8/150        167    12/159       S ⇒ z
     2/159               ......
158: ( y )

159: ( j,z)
```

Haakjes duiden op variabele inhoud, (153 t/m 157) zijn administratieve constanten. De besturing komt op de 160 a binnen. De opdracht op 164 a vormt de variabele aanhaalopdracht voor a_j, die op 164 b plaatst deze „voor de voeten" op 165. z wordt in S uitgerekend en na afloop als de werkruimte voor j niet meer nodig is, daar ingevuld.

Stel dat gevraagd is een derde graard polynoom te tabelleren, met als argument $x^{\frac{1}{2}}$, voor x = 0 (0,01)0.99. Om de 10 regels is een extra regel blank gewenst. We maken gebruik van de typsignalen TWNR (= Terug Wagen, Nieuwe Regel) en Tab = (Tabuleert) en een typprogramma. We kunnen de beide behandelde blokschema's incorporeren. De index i telt de regels in een „blokje" van 10. In fig. 12 is het blokschema weergegeven. De gecomprimeerde blokken zijn door stippellijnen aangegeven. Men realisere zich, dat (bv. als de machine in het tweetallig stelsel werkt), voor de operatie Typ mogelijkerwijs meer dan één opdracht - een stuk programma dus - nodig is.

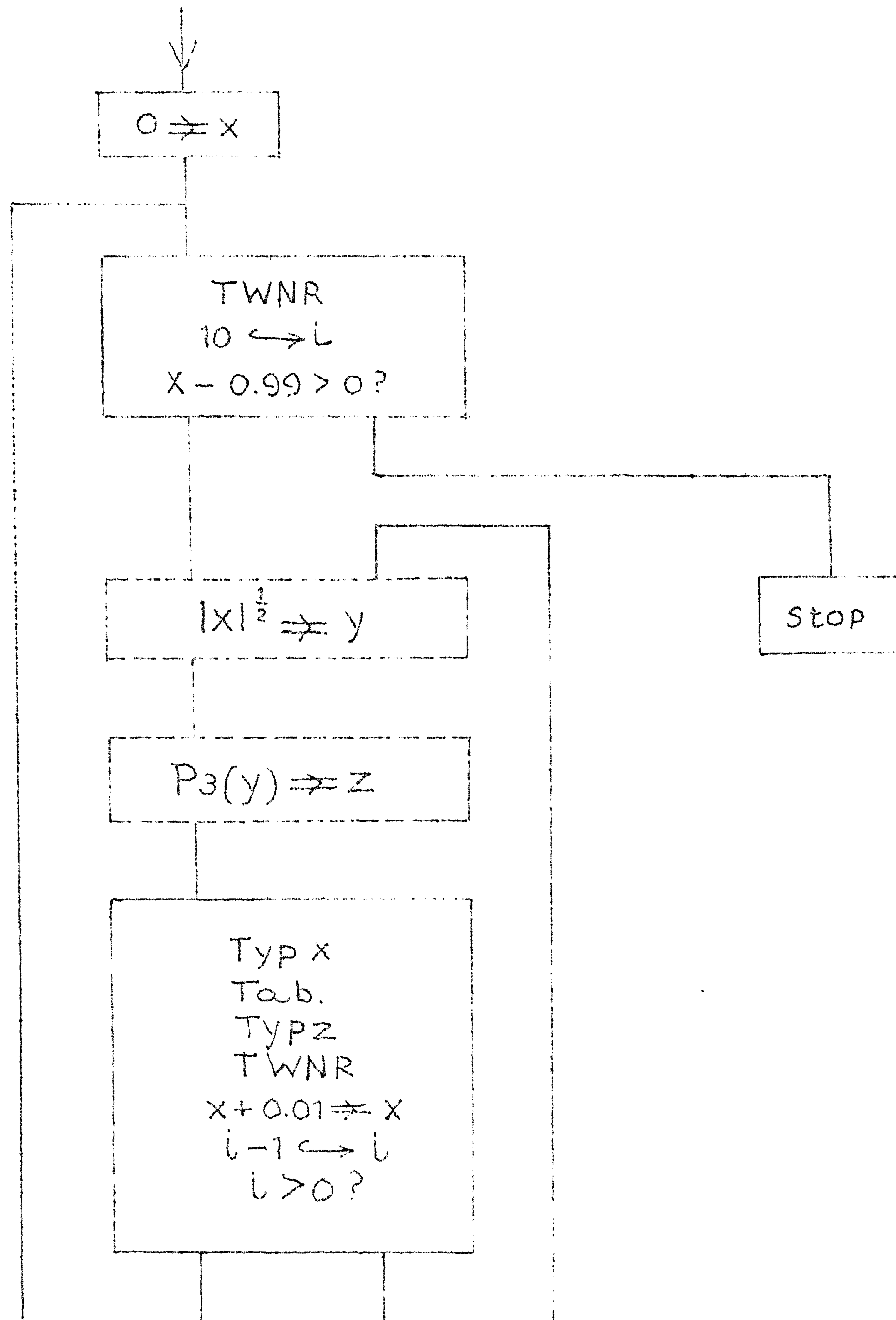


fig. 12.

Syllabus No.6 van de cursus 1955-'56:
Programmeren voor automatische rekenmachines
onder leiding van
Prof. Dr. Ir. A. van Wijngaarden en de Heer E.W. Dijkstra

SUBROUTINES I

Het is de taak van de programmeur om een bepaald rekenproces op te bouwen met behulp van de opdrachtencode van de betroffen machine. Hij zou echter nodeloos zwaar belast worden, als hij iedere keer weer elk programma op moest bouwen uit deze minuscule bouwsteentjes, omdat haast elke berekening gesplitst kan worden in grotere onderdelen, waarvan sommige een zo algemene functie hebben, dat soortgelijke rijen opdrachten zeker in andere programma's voorkomen. Als voorbeeld noemen wij het berekenen van de (co)sinus, het trekken van de tweede- en hogeremachtswortel, delen (als de machine niet over een directe deling beschikt), het vermenigvuldigen van twee complexe getallen (d.w.z. het berekenen van reeel en imaginair deel, als van beide factoren reeel en imaginair deel gegeven zijn), het berekenen van de logaritme, van de e-macht (al of niet voor een complex argument), het berekenen van een integraal, b.v. volgens Simpson (waarbij de integraal nader gespecificeerd dient te worden) enz. enz.

Voor dergelijke problemen zijn geschikte numerieke procedures uitgezocht en geprogrammeerd: deze programmaatjes worden standaard-subroutines of kortweg subroutines genoemd. Een subroutine is nooit een zelfstandig programma, maar moet in een „echt" programma (het zg. hoofdprogramma) geïncorporeerd worden. Om na te gaan, hoe dit incorporeren het soepelste verloopt, vergelijken wij twee machines, W en X, die alleen daarin verschillen, dat machine W wel in staat is, direct een tweedemachtswortel te trekken, terwijl dit met machine X door een of ander (bv. iteratief) proces moet gebeuren: wij nemen aan, dat een dergelijk proces voor machine X geprogrammeerd is in de vorm van een standaardsubroutine. Machine X kan elk programma voor machine W zonder meer overnemen, zolang de worteltrekopdracht er niet in voorkomt. Als deze worteltrekopdracht er wel in voorkomt, zouden we het programma voor machine W aan machine X aan kunnen passen, door overal, waar deze opdracht voorkomt, de standaardsubroutine voor de worteltrekking in te lassen. Als het programma veel worteltrekkingen op verschillende plaatsen in het programma bevat, wordt het programma in de versie voor machine X wel heel veel langer, misschien wel zoveel langer, dat de benodigde programmaruimte de capaciteit

van het geheugen overschrijdt. Maar onze oplossing is in dit opzicht ook duidelijk verkwistend geweest: we hebben het geheugen halfvol geschreven met duplicaten van hetzelfde programmaatje, terwijl men zich af zou kunnen vragen, of één keer niet genoeg was. Dit is inderdaad het geval, maar niet zonder enige extravoorziening. Laat de worteltreksubroutine (in enkelvoud!) op een bepaalde plaats in het geheugen staan; iedere keer, dat in het programma een wortel getrokken moet worden, springt de besturing naar de standaardsubroutine. De informatie, die „de subroutine meekrijgt" is nu tweeerlei: ten eerste het getal, waaruit de wortel getrokken moet worden, ten tweede, welke worteltrekking uit het programma uitgevoerd wordt, m.a.w. waar na afloop van de worteltrekking de berekening voortgezet moet worden. Conclusie: aan het einde van de worteltrekking ontmoet de besturing een sprongopdracht met variabel adres. In het hoofdprogramma staat een sprong naar de subroutine, die (samen met al, wat er speciaal staat ter verstrekking van de noodzakelijke informatie aan de subroutine) de aanroep wordt genoemd; het gebruikelijke arrangement is, dat de besturing na het doorlopen van de subroutine, springt naar de eerste opdracht, volgend op de aanroep. Zo wordt in ARRA en ARMAC aan de subroutine in een van beide aritmetische registers (om precies te zijn, in de lage helft van het A-register) de zg. koppelopdracht (= linkorder) meegegeven, d.w.z. de inconditionele sprongopdracht naar de eerste opdracht na de aanroep. Aan het begin van de subroutine wordt de inhoud van A (= koppelopdracht) op een aan het einde van de subroutine opengelaten adres geschreven, zodat na de voltooiing van de berekening in de subroutine, de besturing „weer heengaat, vanwaar hij gekomen was" (Het effect van de opdracht, die de koppelopdracht in A plaatst, is afhankelijk van de plaats, waarop hij staat, want de opdrachtteller - de teller, die de plaats bijhoudt, waarvandaan op dat ogenblik de opdrachten gehaald worden - wordt uitgelezen.)

Opm.: Het meegeven van de koppelopdracht - d.w.z. het plaatsen van de koppelopdracht in A, geschiedt in ARRA en ARMAC verschillend. In ARRA gebeurt dit door de „communicatie-opdracht" (nl. „adresloos") 24/6, met de volgende functie:

24/6 als a-opdracht op adres x: $\langle A \rangle' = 0/0;\ 7/x + 1$ en
24/6 " b " " " x: $\langle A \rangle' = 0/0;\ 15/x + 1$.

De volgende opdracht is altijd de sprongopdracht naar de subroutine bv.:

```
        ......
205 a   24/6     <A>' = 0/0; 7/206
    b    7/480 =)   naar subroutine op 480 e.v.
=) 206 a ......     hier komt de besturing terug van de sub-
routine.
```

Dankzij de structuur van de opdracht 24/6 kan een subroutine gelijkelijk van de a- als van de b-plaats aangeroepen worden.

In ARMAC is de functie van 24/6 gecombineerd met de volgende sprongopdracht: ARMAC kent nog twee inconditionele sprongopdrachten, de zg. routine-aanroepen: behalve dat inconditioneel gesprongen wordt naar de a- resp. b-plaats van het door het numerieke gedeelte bepaalde adres, wordt in de lage helft van het A-register de sprongopdracht naar de „onmiddellijk" volgende opdracht geplaatst. (Bij deze laatste operatie wordt de opdrachtteller uitgelezen; dit gebeurt, voordat de inhoud hiervan gewijzigd wordt, d.w.z. de sprong effectief gemaakt wordt).

Voor machine X hebben we ons doel bereikt: het programma wordt iets langer, omdat de subroutine voor de worteltrekking ergens in het geheugen moet staan, maar aan de andere kant beschikken we nu over een machine, die logisch even machtig is als machine W. De aanroep van de subroutine fungeert als uitbreiding van de opdrachtencode! Tevens zal men de subroutine volgens een of andere conventie vastleggen op het (cq. een) invoermedium van de machine; voor ARRA hoeft men een subroutine, die men wil gebruiken, niet meer te ponsen; men gebruikt de standaard-band of een - automatisch geponste - copie. Op deze wijze reduceert men het ponswerk en daarmede een foutenbron.

Een laatste voordeel van standaardsubroutines is, dat men vertrouwen kan, dat ze goed zijn: het zijn op al hun verrichtingen geteste stukken programma, ingebracht met behulp van banden uit de biblioteek. Deze wetenschap verlicht de localisatie van fouten in een programma aanzienlijk: men weet, waar men niet hoeft te zoeken.

Een subroutine is een blok met in het algemeen een veelvoudige uitgang, nl. met een multipliciteit gelijk aan het aantal punten in het programma, vanwaar hij wordt aangeroepen. De normale blokschemanotatie (max. dubbele uitgang!) met de subroutine als gecomprimeerd blok, is dus niet aan het gebruik van subroutines aangepast. Veeleer worden in blokschemaas subroutines ingelast, als uitbreiding van de opdrachtencode, en zo vaak genoteerd,

als ze worden aangeroepen. Men duidt dan aan, dat het hier een subroutine betreft door het blok met stippellijnen te omgeven. Dit zullen we illustreren door de gecontroleerde berekening van sinx en cosx. In dit voorbeeld verwaarlozen we alle problemen wat betreft de capaciteit van de registers en de reductie van argumenten modulo 2π, etc. (fig. 1). De berekening mag niet gecontroleerd worden door verificatie van de identiteit $\cos^2 x + \sin^2 x = 1$: als nl. bv. sinx dicht bij nul ligt, worden door kwadratering de cijfers van sinx niet in de controle betrokken; het teken ontsnapt aan deze controle altijd. Een betere controle is de verificatie van de relatie:

$$\sin(x + \tfrac{1}{4}\pi) = \tfrac{1}{2}\sqrt{2}(\sin x + \cos x)$$

sin S⇒S omschrijve de functie van de sinussubroutine, die, na optelling van $\frac{1}{2}\pi$ ook voor de berekening van cosx gebruikt kan worden. (Als de controle faalt, wordt in dit voorbeeld het mislukte stuk van de berekening opnieuw geprobeerd). In de normale procedure zal de subroutine een of ander codenummer (catalogus-nummer) bezitten, dat in het blokschema wordt vermeld.

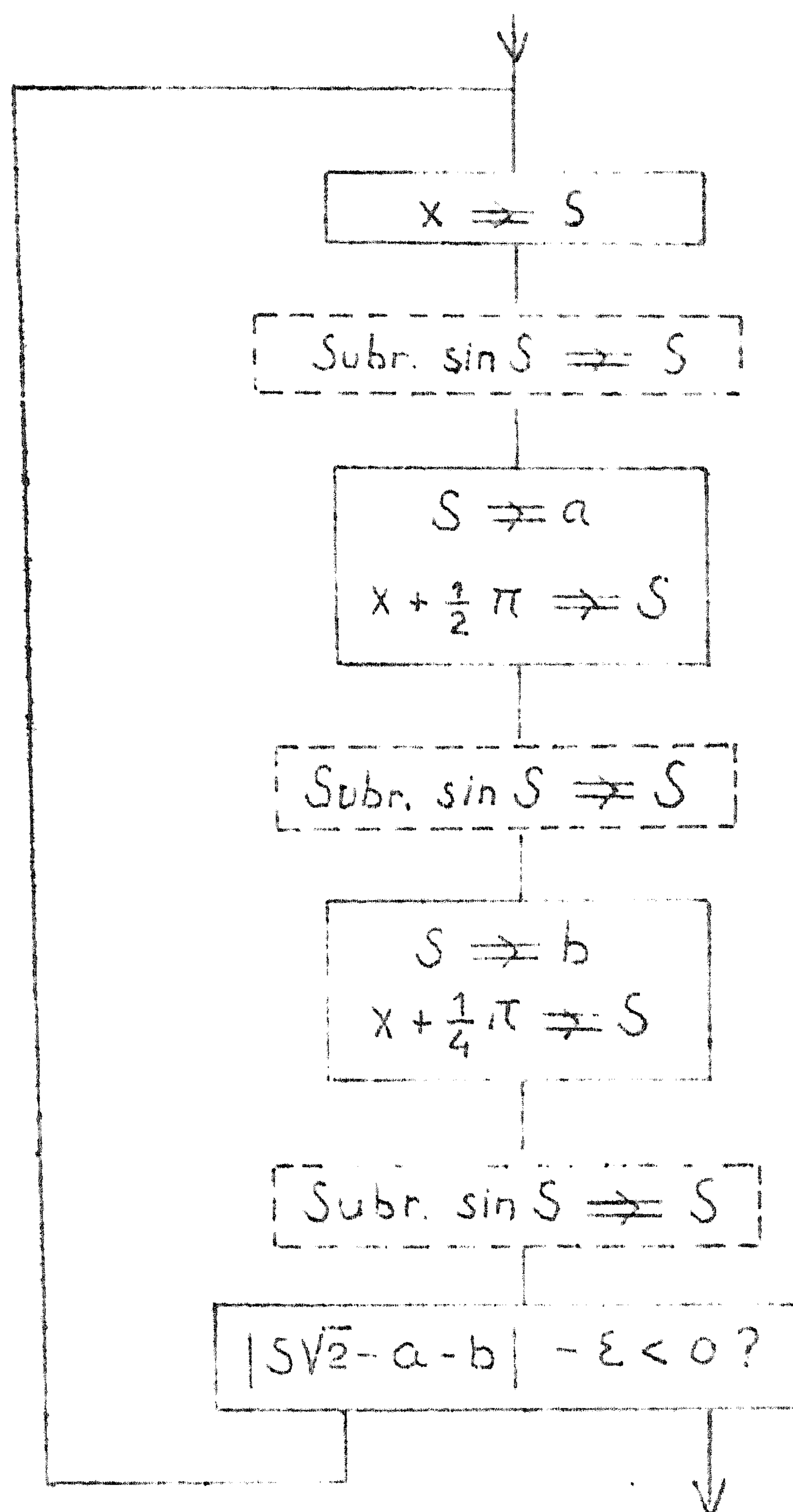


fig. 1 sinx ⇒ a
cosx ⇒ b
(gecontroleerd)

De zojuist gebruikte sinussubroutine behoort tot een grote groep subroutines van hetzelfde type, wat betreft de hoeveelheid informatie, die erdoor verwerkt en afgeleverd wordt: uitgerekend wordt één (reële) functie van één (reëel) argument: f(x)⇒y. In ARRA en ARMAC geldt als conventie, dat hiervoor het S-register gebruikt wordt, zodat hun algemene gedaante luidt f(S)⇒S. Onder deze groep vallen b.v.:

1. de sinus en de cosinus;
2. de arcsinus en de arccosinus;
3. de exponentiële functie (grondtal bv. e, 2 of 10);

4. de logaritme (grondtal bv. e, 2 of 10);
5. de n^{de}-machtswortel;
6. het n^{de}-machtspolynoom. met bepaalde coëfficiënten;
7. het interpoleren in een bepaalde tafel.

De laatste twee voorbeelden vallen strikt genomen slechts ten dele onder deze groep: bijvoorbeeld de routine voor het vijfdegraadspolynoom $\sum_{i=0}^{5} a_i x^i \Rightarrow y$ gebruikt behalve het in S meegegeven argument **x** bovendien de 6 coëfficiënten a_i. Omdat echter deze subroutine in één probleem doorgaans gebruikt zal worden voor vele waarden van het argument X, maar met dezelfde coëfficiënten a_i, valt, zolang het programma de coëfficiënten a_i niet wijzigt, deze routine onder het boven genoemde type. De vijfdegraadspolynoom-subroutine bestaat uit een stukje programma, benevens 6 voor de coëfficiënten gereserveerde adressen. Bij het inlezen van de subroutine worden deze coëfficiënten ter plaatse ingevuld: in de bibliotheek bevindt zich een bandje ter berekening van het algemene polynoom van de vijfde graad; door toevoeging van een bandje met de coëfficiënten brengt de gebruiker de subroutine ter berekening van een bepaald polynoom van de vijfde graad in. De interpolatie-subroutine gebruikt nog meer vaste informatie. Behalve, dat de gehele tafel in het geheugen moet staan (bv. voor met constant interval opklimmende waarden van het argument op achtereenvolgende geheugenplaatsen), moet dan bv. nog gespecificeerd zijn:
het interval in het argument, de argumentwaarde voor het begin van de tafel, en het adres, waar de bijbehorende functiewaarde geborgen staat. Weer geldt, dat in de bibliotheek van standaardsubroutines zich het bandje met de algemene interpolatie-subroutine bevindt, waar tijdens het inlezen een bandje met parameters aan toegevoegd wordt. En zo is dan de subroutine ter interpolatie in een bepaalde tafel ingelezen. Men noemt deze parameters de zg. vaste parameters (= prefixed parameter), in tegenstelling tot de zg. programma-parameters, d.w.z. voor de subroutine vereiste informatie, die van aanroep tot aanroep wisselt, en dus „door het programma" iedere keer meegegeven wordt. (Een voorbeeld van een enkele programma-parameter is het in S meegegeven argument in de boven gegeven voorbeelden van $f(S) \Rightarrow S$).

De bovengenoemde parameters vertegenwoordigen uitersten: het is eveneens mogelijk, dat een of andere parameter in de loop van de hele berekening af en toe eens wijzigt, denk bv. aan het

interval in de onafhankelijk veranderlijke, waarmede een stapsgewijze integratie wordt uitgevoerd. Doordat een speciaal hiervoor gereserveerd adres het interval bevat - onthoudt - is de integratie op een bepaald interval „ingesteld", bevindt de integratie-routine zich in een bepaalde toestand. Hetzelfde doet zich bv. voor bij de subroutine, die, door tussen het typen van de getallen, volgens een bepaald patroon Tab- of TWNR-signalen te geven de zg. pagina-indeling regelt: hier wordt, aan de hand van een of meer getallen onthouden, op welke plaats van de pagina, op welk punt van het Tab- en TWNR-patroon men is. Dergelijke routines hebben verschillende ingangen, m.a.w. verschillende aanroepen. Men kan de besturing op verschillende plaatsen laten inspringen, op deze manier verschillende functies onderscheidend. Bij de routines, die zich in verschillende toestanden kunnen bevinden, zullen naast aanroepen, waarbij de heersende toestand zich doet gelden, een of meer aanroepen voorkomen, waarmee de instelling op een bepaalde toestand bewerkstelligd wordt. Verschillende ingangen zijn echter niet beperkt tot zg. administratieve subroutines (een type waartoe de twee zojuist genoemde onder vallen); de cosinus-subroutine berekent cosinus S, door $\frac{1}{2}\pi$ bij het argument op te tellen, en er dan de sinus van te berekenen. Deze routine kent tevens een sinus-aanroep, waarbij de aanvankelijke optelling van $\frac{1}{2}\pi$ achterwege blijft. Op deze wijze bespaart men òf het inlezen van twee routines, òf - andere mogelijkheid - de optelling van $\frac{1}{2}\pi$ door het hoofdprogramma.

Een indeling, die om der wille van de volledigheid gegeven wordt, is naar de zg. orde: de eenvoudige subroutines (als sinus of logaritme), waar geen aanroep van een andere subroutine in voorkomt, is een subroutine van de nulde orde. De subroutine echter, die op zijn beurt „sub-routines" aanroept, is van de orde n + 1, als n de hoogste orde van de hulpsubroutines is. Een algemene integratie-subroutine is dus altijd van hogere orde, omdat de berekening van de (speciale) integraal als subroutine zal zijn geprogrammeerd en deze subroutine door de integratie wordt aangeroepen. Bij elk programma is de hierarchie van subroutines een-duidig.

In dit laatste voorbeeld is een nieuw element ingevoegd, de berekening van de integrand met behulp van een subroutine: dit zal over het algemeen geen standaardsubroutine zijn! De programmeur maakt zijn eigen subroutines, waarvan de functie te speciaal is, om ze standaard-subroutines te noemen. Deze procedure komt de overzichtelijkheid en de flexibiliteit van het program-

ma zeer ten goede, maar daarover later.

In het voorafgaande is niet gepoogd een sluitende classificering van subroutines te geven. Het is meer bedoeld als een overzicht in vogelvlucht, voordat we enige speciale subroutines in detail gaan beschouwen.

Syllabus No.7 van de cursus 1955-'56:

Programmeren voor automatische rekenmachines

onder leiding van

Prof. Dr. Ir. A. v. Wingaarden en de Heer E.W. Dijkstra

IN- EN UITVOER VAN DE ARRA

Elk werkelijk probleem begint ermee, dat de noodzakelijke informatie-d.w.z. opdrachten en getallen - te bestemder plaatse in het geheugen wordt ingevuld, en eindigt met het afleveren van de resultaten. Deze sluitstukken („invoer en uitvoer") zijn thans aan de orde. De invoer is van belang, omdat deze nauw verband houdt met de wijze, waarop de programmeur zijn programma opschrijft: hij doet dit in een zodanig symbolisme dat van zijn vellen „de banden onmiddellijk geponst kunnen worden".

De invoer van de ARRA geschiedt nl. via geponste telexband. In de dwarsrichting van de band kan in vijf posities al dan niet een gaatje geponst worden: de ponsapparatuur heeft dienovereenkomstig $2^5 = 32$ toetsen. Deze toetsen dragen de opschriften 0,1,2,.... 23,24,A,B(+),C(-),D(+.),E(-.),F en X. (Vier toetsen, de zg. tekentoetsen, zijn dus dubbel benoemd.) Elke toets ponst een speciale pentade.

Het invoerprogramma (dat zich al in het geheugen bevindt!) leest groepjes pentades, die samen een zg. molecuul informatie vormen, en combineert deze pentades volgens vaste regels tot een geheel. Deze regels hebben hun neerslag in de ponsconventies, die nu volgen.

Een molecuul informatie kan zijn

1e: een woord

2e: een controlecombinatie.

Zolang geen controlecombinaties voorkomen, worden de moleculen (woorden dus) in de volgorde, waarin ze van de band gelezen worden, op successieve adressen in het geheugen geborgen.

1e Een woord (dat dus juist een adres vulle) bestaat uit

a) een getal

b) een opdrachtenpaar.

Beide types woorden mogen elkaar op de band afwisselen.

a) Het getal

Er zijn vier soorten getallen, ingeleid door een van de vier tekens:

positief geheel getal	+
negatief geheel getal	-
positieve breuk	+.
negatieve breuk	-.

Elk getal wordt geponst in drieën:

1e: teken (+, -, +. of -.)

2e: de decimale cijfers (uit de groep 0 t/m 9)

3e: de sluitletter X (om het einde aan te geven)

Nullen aan het begin van het decimale gedeelte mogen weggelaten worden; om ± 0 in te brengen, hoeft geen enkele decimale toets aangeslagen te worden. Breuken moeten in 8 cijfers achter de komma worden ingevoerd. (Als de breuk in meer cijfers gegeven is, moet hij op 8 decimalen achter de komma worden afgerond; als hij in minder cijfers gegeven is, moet hij met nullen worden aangevuld.) Ook bij breuken mogen nullen direct volgend op de punt tot aan het eerste cijfer ≠ 0, worden weggelaten.

b) Het opdrachtenpaar

Bij elkaar behorende a- en b-opdracht werden immer tesamen ingebracht: het is onmogelijk om een losse a- of b-opdracht in te brengen. Van dit tweetal wordt eerst de a-opdracht, dan de b-opdracht geponst, en wel volgens dezelfde conventies; slechts hun onderlinge positie bepaalt, wie a-opdracht, wie b-opdracht is. De opdracht wordt in vieren geponst, in de volgorde: functie, plaats, sluitletter, kanaalcorrectie.

De „functie" wordt geponst door één aanslag uit de groep 0 t/m 24. (Omdat het mogelijke functiecijfer loopt van 0 t/m 24, komen deze toetsen op het toetsenbord voor. Het inbrengen van de onbestaanbare opdrachten 25 t/m 31 wordt dus niet aangemoedigd; langs een omweg is het wel mogelijk).

De overige drie, plaats, sluitletter en kanaalcorrectie, bepalen het adres.

De „plaats" wordt geponst met een variabel aantal pentades door aanslag op de decimale toetsen 0 t/m 24. Als „plaats" uit meer dan één decimaal cijfer bestaat, maar het getal, gevormd door de hoogste (= eerste) twee decimale cijfers kleiner dan 25 is, mag dit tweetal met een van de toetsen 10 t/m 24 geponst worden. Voor de volgende cijfers beperke men zich tot de toetsen 0 t/m 9; combinatie is daar niet mogelijk.

De „sluitletter" is één aanslag (uit de groep A t/m F en X) met als eerste functie het indiceren van het einde van „plaats"; voor de tweede functie, zie onder.

De „kanaalcorrectie" is eveneens één aanslag, nu uit de groep

0 t/m 24. (Evenals de „functie" moet de kanaalcorrectie altijd met één pentade geponst worden en mag de 0 niet weggelaten worden) De kanaalcorrectie geeft aan het 32-voud, waarmede de „plaats" vermeerderd wordt. Zo kan bv. de opdracht, die we tot nog toe noteerden als 24/104 op vele manieren geponst worden (de aanslagen zijn door komma's gescheiden).

24, 1, 0, 4, X, 0
24,10, 4, X, 0
24, 7, 2, X, 1
24, 4, 0, X, 2
24, 8, X, 3

Dit laatste is de normale ponsing, we noteren en onthouden de opdracht als 24 8 X3. (Het is gebruik sluitletter en kanaalcorrectie zonder ruimte ertussen te schrijven.)

Als tweede voorbeeld: 24/512 + n is de opdracht voor de zg. snelle optelling: $[A] + n \Rightarrow [A]$, als $0 \leqslant n \leqslant 255$. Omdat 512 = 16x32, noteren we deze opdracht als

24 n X16

Als n hier de 31 overschrijdt, gaan we geen 32-vouden afsplitsen voor de snelle optelling van 70 schrijven (en dus ponsen) we

24 70 X16 en niet 24 6 X18

De kanaalcorrectie ontleent zijn naam aan de (physische) indeling van het geheugen, dat bestaat uit een magnetische trommel, waarlangs 32 koppen zijn opgesteld: elke kop bestrijkt 32 successieve adressen, die samen een zg. kanaal vormen. Het geheugen is ingedeeld in 32 kanalen: 0 t/m 31 vormen het nulde kanaal, 32 t/m 63 het eerste kanaal, etc.; elk adres is bepaald door kanaal (hoogste vijf binalen van het adres) en plaats in het kanaal (laagste vijf binalen). De kanaalcorrectie is dus te interpreteren als „zoveel kanalen verder".

De sluitletter X geeft alleen aan, dat de laatste „plaatspentade" gelezen is; de sluitletters A,B,C,D,E en F vermeerderen tevens het adres met een parameter, waarvan de programmeur zelf de waarde vast kan stellen; deze waarden moet hij voordat de banden worden ingelezen, met een apart bandje, de zg. voorponsing, inbrengen. Voor het onthouden van deze zes waarden zijn 6 plaatsen in het 31ste kanaal gereserveerd, te weten

1001 X0 voor A
1002 X0 voor B
1003 X0 voor C
1004 X0 voor D
1005 X0 voor E
1006 X0 voor F

Deze faciliteit maakt het mogelijk, om van standaard-subroutines standaard-bandjes te hebben, waarop in het midden is gelaten, waar in het geheugen de subroutine dit keer komt te staan. Alle subroutines gebruiken de sluitletter F; elke subroutine wordt voorafgegaan door een eigen voorponsinkje, waardoor op adres 1006 het gewenste beginadres van de subroutine wordt ingevuld. Dan volgt de standaardband, die de nu geldende waarde voor de sluitletter F in rekening brengt. Voor subroutines is een dergelijke faciliteit noodzakelijk: hoe het geheugen verdeeld wordt verschilt immers van programma tot programma, voor de subroutine, die in beide programma's voorkomt, is soms hier, soms daar een stukje geheugen beschikbaar. Als er meer subroutines in een programma voorkomen, wisselt de bij F behorende parameter dus tijdens het inlezen.

De sluiletters verrichten echter ook hun functie bij het opstellen van het zg. hoofdprogramma: elk probleem valt immers altijd uiteen in diverse min of meer gescheiden stukken: om deze stukken achter elkaar in het geheugen te zetten, moet men weten, hoelang ze zijn, hoeveel adressen ze beslaan. Maar dat is pas bekend, als het programma gemaakt is! Daarom is het gewenst, de beslissing, waar die stukken zullen beginnen, uit te stellen totdat het programma klaar is. De normale gang van zaken is, dat men deze verschillende stukken een eigen sluitletter als label geeft. Men merkt dan wel, hoeveel kanalen zij beslaan. Men adresseert relatief t.o.v. het begin van het stuk

Omdat in de machine de adressen in binale vorm staan, wordt het opsporen van fouten in het programma (en in de machine) zeer vergemakkelijkt, als de parameters die bij de sluitletters horen, 32-vouden zijn, dus nulde adressen van kanalen. Dan is het ook mogelijk om van kanaal A0, van kanaal B2 te spreken. Programmabladen bevatten ruimte voor 32 adressen, genummerd van 0 t/m 31. Bij een bepaald probleem hoort dan een staatje, dat er bv. uitziet:

A0 = 5 (d.w.z. 0A0 = 0X5, dus 1001 = 160 = 5 x 32)
B0 = 9
C0 =16
E0 =18

Dit geeft de beslissingen weer, die pas gemaakt zijn, toen het programma klaar was: kanaal A0 wordt nu kanaal X5, A1 wordt X6, A2 wordt X7 en A3 wordt X8, etc. (er waren dus kennelijk 4 A-kanalen, 7 B-kanalen en 2 C-kanalen.) Deze nomenclatuur wordt, voordat de andere banden ingebracht worden, ingelezen met behulp van de zg. <u>generale voorponsing</u>, die aan de sluitletters de vaste parameters toekent, in tegenstelling tot de <u>specifieke voorponsingen</u>

der standaardsubroutines.

2e Controlecombinaties zijn molecules informatie, die niet als woord opgebouwd en geborgen worden, maar anders verwerkt worden. ARRA kent er in eerste instantie twee:

<u>De adrescombinatie</u> Eerder is vermeld, dat woorden van de band op successieve plaatsen worden geborgen Dit impliceert, dat voor het eerste woord van een reeks een andere aanduiding nodig is. Dit is de zg. adresindicatie, die vastlegt de plaats waar het eerstvolgende woord, dat van de band gelezen wordt, opgeborgen moet worden. Dan ligt de het in te vullen adres voor alle volgende woorden tot aan de eerstvolgende adresindicatie vast.
De adres-indicatie bestaat uit het adres, dat geponst wordt, evenals adressen in opdrachten, voorafgegaan door een A (dit geschiedt met dezelfde toets; de A heeft al openingsponsde een heel andere functie, dan als sluitletter!) De A gevolgd door het adres moet in duplo geponst worden! Zo begint de band van elke standaardsubroutine met

	A	F0
	A	F0

Het bandje dat de bovengegeven generale voorponsing inbrengt kan er als volgt uitzien:

	A	1001	X0
	A	1001	X0
1001	0		X5
	0		X0
1002	0		X9
	0		X0
1003	0		X16
	0		X0
	A	1005	X0
	A	1005	X0
1005	0		X18
	0		X0

of (als sluitletter D in het hele probleem niet voorkomt, en men de tafel van 32 kent)

A	1001	X0
A	1001	X0
+	160	X
+	288	X
+	512	X
+		X
+	576	X

De F-sprong: Staat op de band een F, gevolgd door een adres, dan springt de besturing naar de a-opdracht van het betroffen adres. Een voorbeeld hiervan is:

F 28 X0

waarna de machine stopt. (Slotcombinatie).
Opm.: Het communicatieprogramma (d.w.z. het programma dat in- en uitvoer verzorgt) is permanent in de machine aanwezig en beslaat de kanalen 0 t/m 4 (vaste inhoud) en 31 (variabele inhoud). De opdrachten, die de besturing ontmoet, als naar a 28 X0 gesprongen wordt, zijn dus altijd dezelfde. Vandaar dat vastgesteld kan worden, dat de machine stopt.

Schrijven en controleren.

Het invoerprogramma kent twee „standen": òf elk woord wordt ter plaatse geschreven òf elk van de band opgebouwd woord wordt vergeleken met de inhoud van het betroffen adres: als deze inhoud, die er al staat, afwijkt van het geassembleerde woord, stopt de machine. Incidentele fouten tijdens het inlezen worden aldus ondervangen. Start men de machine op a 0X1, dan gaat hij schrijvend, op a1X1, dan controlerend bandlezen. De stand kan ook van de band beinvloed worden door de zg. wissel

F 30 X3

waardoor de stand van het invoerprogramma verspringt (dit verspringen wordt pas effectief bij de eerstvolgende adres indicatie.) Door een band met F 30X3 af te sluiten, en dan het geheel er nog eens achter te ponsen, maakt men zg. zelfcontrolerende banden.

De X.

Zoals men kan nagaan, komt de X niet als openingspentade van een molecuul voor. Vooraan elk molecuul (niet tussen a-opdracht en volgende b-opdracht!) mag een willekeurig aantal pentades X geponst worden. Deze worden tijdens het bandlezen geskipt. Omdat de pentade X bestaat uit vijf gaatjes (Erase) kan men zo tijdens het ponsen gemerkte misslagen herstellen.
Thans de uitvoer m.a.w. het typen. De typroutines typen de absolute waarde van (S') (Opm.: de inhoud van S voor de aanroep geven we met een enkel accent, na de aanroep met een dubbel accent aan). Door de aanroep kan men regelen:

a) of eerst al dan niet het teken getypt wordt
b) of (S') als geheel getal [S'] of als breuk {S'} geinterpreteerd wordt.

Hiermede corresponderen vier aanroepen:

```
     24    6   X0
      7        X3 =)        Typ G; zonder teken.
  =) ............

     24    6   X0
     15    2   X3 =)        Typ G; met teken.
  =) ............

     24    6   X0
     15   13   X3 =)        Typ B; zonder teken.
  =) ............

     24    6   X0
      7   16   X3 =)        Typ B; met teken.
  =) ............
```

De hierboven gebruikte onderscheiding tussen geheel getal (G) en breuk (B) slaat op de interpretatie van (S') en niet op het beeld op de pagina: bij gehele getallen kan men een punt inlassen, bij breuken kan men de punt enige plaatsen naar rechts verschoven typen, of zelfs geheel onderdrukken. De gedaante waarin de gehele getallen en de breuken uitgetypt worden, wordt bepaald door zg. typ-constanten (in kan. 31) die verdeeld zijn in twee groepen: de ene groep specificeert de handeling „typ G" nauwkeuriger, de andere groep de handeling „typ B". Men voert dus in een bandje „typconstantes G" en een bandje „typconstantes B". Zonder tussentijdse wijziging van de typconstantes kan een programma dus een „genre" G, en geheel onafhankelijk daarvan een „genre" B uittypen, terwijl het teken naar believen dan wel, dan weer niet getypt wordt.

Bij „typ G" moet [S'] in absolute waarde kleiner zijn dan 10^q, als q het (totale) aantal getypte cijfers is.

Bij „typ B" wordt {S'} exact op het laatste decimale cijfer afgerond. {S'} mag niet zo dicht bij $\pm$ 1 liggen, dat door deze afronding capaciteitsoverschrijding optreedt.

Een onderscheiding in facultatief typen en imperatief typen slaat op de cijfers voor de punt of op het totale getal, als geen punt getypt wordt. Als deze imperatief getypt worden verschijnt het aangegeven aantal cijfers op het papier; bij facultatief typen worden eventuele nullen aan de hoge kant door spaties vervangen, met uitzondering van het laatste cijfer, dat altijd getypt wordt (opdat bij alle cijfers = 0 - en geen punt! -

het papier niet schoon blijft). Als één cijfer voor de punt getypt wordt, is facultatief en imperatief dus hetzelfde. Voor facultatieve cijfers kan men desgewenst een extra spatie inlassen.

Typcontrole en regelindeling

Dit onderdeel van het communicatieprogramma heeft een dubbele functie

a) hier vindt de uiteindelijke controle op het typen plaats

b) er worden n getallen per regel getypt.

De subroutine kent twee aanroepen:

De inloopsaanroep:

```
      24    6   X0
       7   19   X2 =)
   =) ...........            met de functie:
```

1) Telling wordt ingesteld op het begin van de regel

2) TWNR, gevolgd door een vertraging wordt bewerkstelligd, waarna een TAB.

Deze aanroep dient de besturing altijd te ontmoeten aan het begin van een probleem.

Opm.: De regel wordt geacht te beginnen bij de eerste tabulatorstop, circa vijf plaatsen rechts van de kantlijn. Dit omdat, als de wagen terugloopt, hij bij de kantlijn soms door de schok weer een plaats terugspringt.

Na elke aanroep van een typroutine volgt:

de controleaanroep

```
      24    6   X0
       7        X2 =)
   =) ...........   + met de functie
```

1) zolang (S') = -0 geeft de routine elke keer een TAB-signaal (of steeds een SPATIE), uitgezonderd elke n^{de} keer, dat de controleaanroep gehoorzaamd wordt: dan volgt nl. TWNR, vertraging en dan TAB. De besturing komt met de conditie negatief in het hoofdprogramma terug.

2) als een keer (S') ≠ -0 (d.w.z. een getal foutief getypt is) dan geeft de routine TWNR (gevolgd door vertraging) en j TAB signalen, als de fout is opgetreden tijdens het typen van het j^{de} getal in de regel. De besturing komt met de conditie positief in het hoofdprogramma terug.

Opm.: Bij een TAB signaal springt de wagen minstens 3 plaatsen. Als men heel veel getallen op een regel wil typen, kan het zijn, dat zoveel ruimte tussen de getallen niet gepermitteerd kan worden. Dan scheidt men de getallen door een spatie. De tabulatorstoppen dienen met zorg geplaatst te worden, omdat deze wel

gebruikt worden, zodra een fout gesignaleerd is.
De controle op het typen is mogelijk, doordat de typroutines, als alles goed gegaan is, in het hoofdprogramma terugkomen met exact dezelfde inhoud, als waarmede ze werden aangeroepen. De controle op het typen van bv. OBO als geheel getal zonder teken wordt als volgt geprogrammeerd: (programma op OAO en volgende).

```
                  3AO a ——> OAO       a    10    BO
                                      b    24 6  XO
                            1AO       a     7    X3 =) typ G.
van de typroutine              =)     b     9    BO
                            2AO       a    24 6  XO
                                      b     7    X2 =) naar de typ-controle
van de typcontrole                    a     6    AO ——>
                            3AO =)    b    . .....
```

Als het typen goed gegaan is, vormt de 9-opdracht - 0 in S; in dit geval komt de besturing in 3AO a aan met het teken van het laatst weggeschreven getal negatief en de besturingsverplaatsing wordt genegeerd.
Is echter tijdens het typen een fout opgetreden, dan wordt door de 9-opdracht geen - 0 in S gevormd, en in 3AO a arriveert de besturing met het teken van het laatst weggeschreven getal positief, zodat de besturingsverplaatsing wel gehoorzaamd wordt, en opnieuw hetzelfde getal getypt wordt. Inmiddels echter is het papier van de schrijfmachine een regel opgevoerd, en de wagen is in die positie gebracht, waarin hij stond, toen aan het typen van het mislukte getal begonnen werd.
De typconstanten zijn dus drieërlei:

1e Typ G.
2e Typ B.
3e Typcontrole en regelindeling.

In het rapport van de rekenafdeling MR21 is de in- en uitvoer van de ARRA uitvoeriger beschreven. Tevens bevat dit rapport de complete tekst van het programma, dat de boven beschreven functies verricht, en een uitgebreide tabel van typconstanten. Uit deze tabel nemen wij als voorbeeld over:

<u>G s 8 fac</u> (Geh. get. in 8 cijfers, facultatief, voorafgegaan door spatie.)

```
   A 230      X24
   A 230      X24
 + 100000000  X
   7  29      X2
   7          X4
```

	B	2.	3	fac
	A	244		X24
	A	244		X24
	7	26		X[illegible]
	15	24		X4
+2	6840	0000		X
+		2684		X
+	10	0000		X
	7	20		X2
	7	23		X4

(breuk in 5 cijfers, 2 voor, 3 na de punt, facultatief)

	n = 4; Tab	
A	242	X24
A	242	X24
+	53	X
+	167	X

(Typcontrole: vier getallen per regel, Tab zo goed)

Syllabus No. 8 van de cursus 1955-'56:
Programmeren voor automatische rekenmachines
onder leiding van

Prof. Dr. Ir. A. v. Wijngaarden en de Heer E.W. Dijkstra

EEN UITGEWERKT VOORBEELD

Gevraagd te onderzoeken, op welke wijze(n) het gehele non-negatieve getal G geschreven kan worden als de som van twee quadraten. Presieser: gevraagd op te lossen
$x^2 + y^2 = G$, als G een nonnegatief getal is, met de nevencondities: 1: x en y zijn gehele getallen
2: x en y voldoen aan $0 \leqslant x \leqslant y$
(Als de eerste conditie niet gesteld was, had het probleem oneindig veel oplossingen; als de tweede conditie niet gesteld was, zou over het algemeen een oplossing in 8 niet essentieel verschillende vormen gevonden worden, nl. met negatieve tekens en verwisseling van x en y. Wanneer zijn er zo minder dan 8 gelijkwaardige oplossingen, en hoeveel?)

De oplossingen worden gevonden door systematisch zoeken. Bij een gekozen waarde van y ligt x vast: het is de vraag, of deze $x = \sqrt{G - y^2}$ een geheel getal is. Dit wordt onderzocht door bij die y enige successieve x-waarden te proberen, d.w.z. te testen, of voor die bepaalde x en y aan $G - x^2 - y^2 = 0$ voldaan is. Zo ja, dan hebben we een oplossing gevonden, zo nee, dan weten we, dat bij deze y geen gehele x bestaat, zodra $G - x^2 - y^2$ van teken is gewisseld, en we gaan de volgende y onderzoeken. We systematiseren dit onderzoek, door met $x = 0$ en een te grote y te beginnen, d.w.z. $y^2 = x^2 + y^2 > G$. Zolang $x^2 + y^2 > G$, trekken we 1 van y af; als $x^2 + y^2 < G$, tellen we 1 bij x op. Zo tasten we het stukje cirkelboog $x^2 + y^2 = G$ af (zie fig. 1, waar G = 74) door het „trapje", dat door deze afspraak de cirkelboog zovaak mogelijk snijdt, zodat alle roosterpunten op de cirkelboog inderdaad gevonden worden (gekenmerkt door $x^2 + y^2 = G$). Hoe we dan door moeten gaan, specificeert de boven gemaakte afspraak niet; we maken de keuze (voor de motivering, zie later) dat dan eerst x met 1 vermeerderd wordt.
Ons „aftasten" moet nog in twee opzichten gecomplementeerd worden:

1: voor het begin: de constructie van een geschikte y bij $x = 0$;

2: voor het einde: een test, zodat door de oplossingen aan de ongelijkheid $x \leqslant y$ voldaan wordt. We zoeken een start-

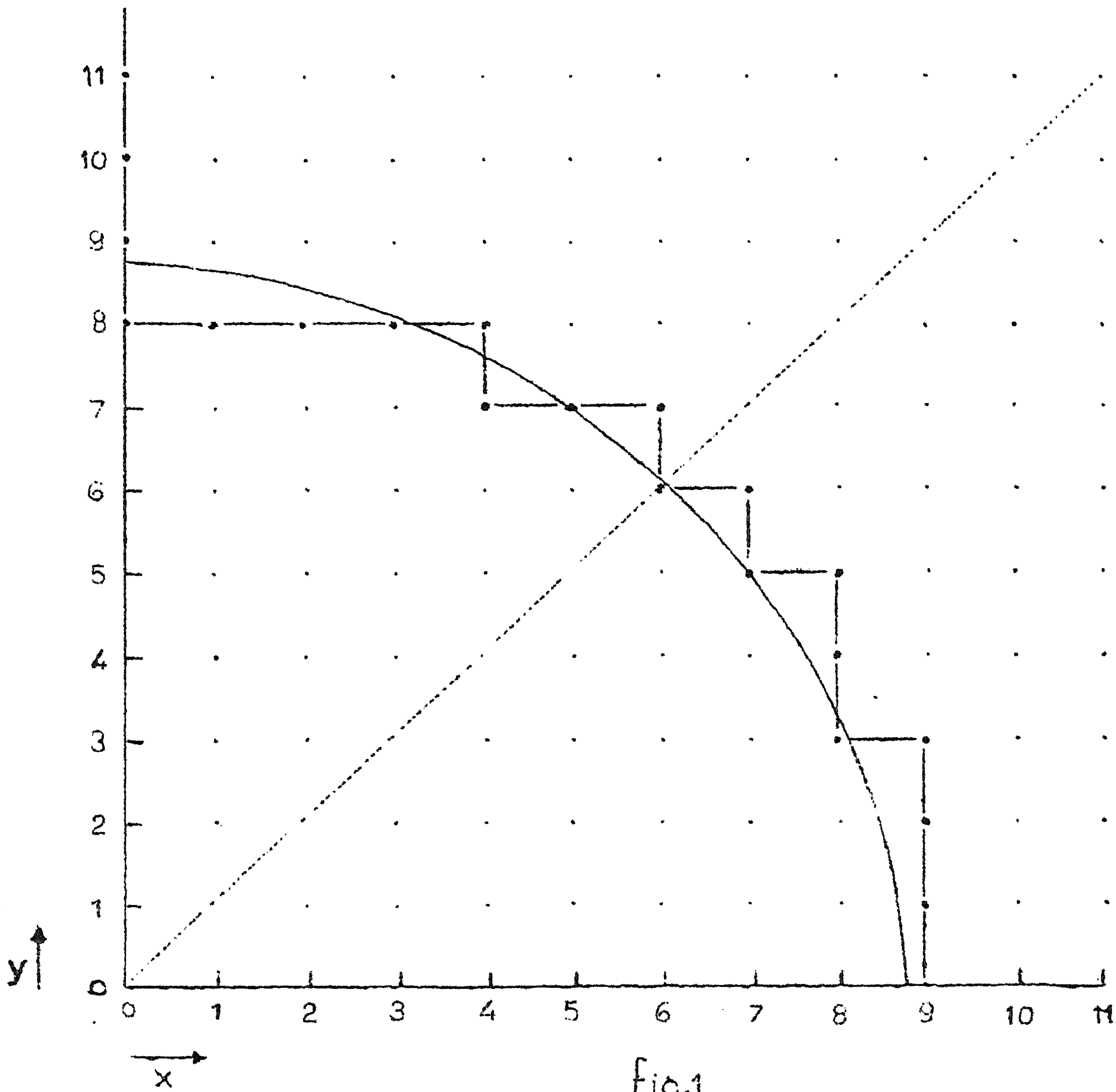


fig.1

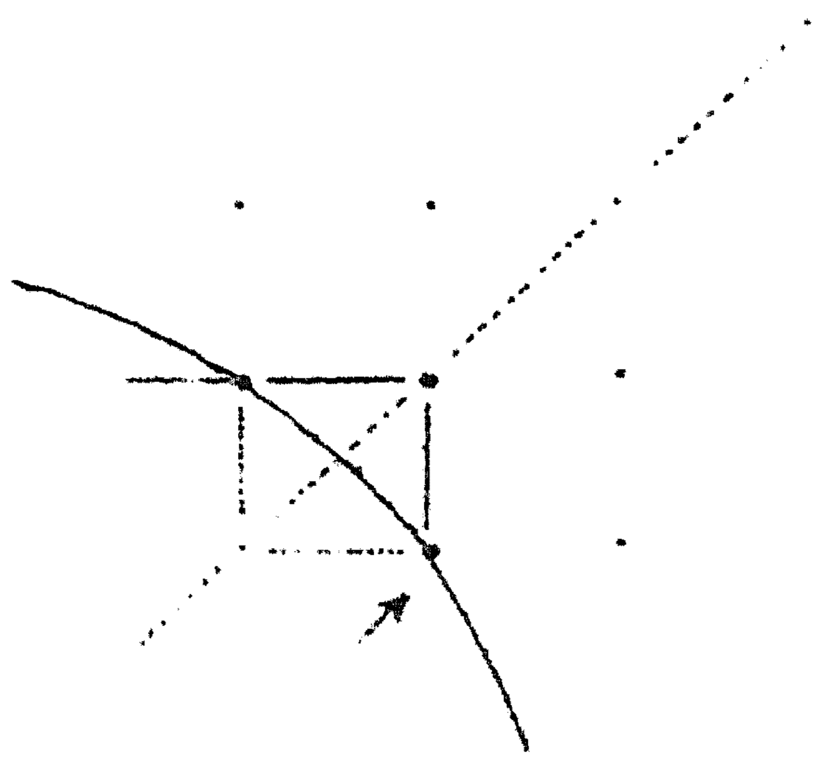


fig.2

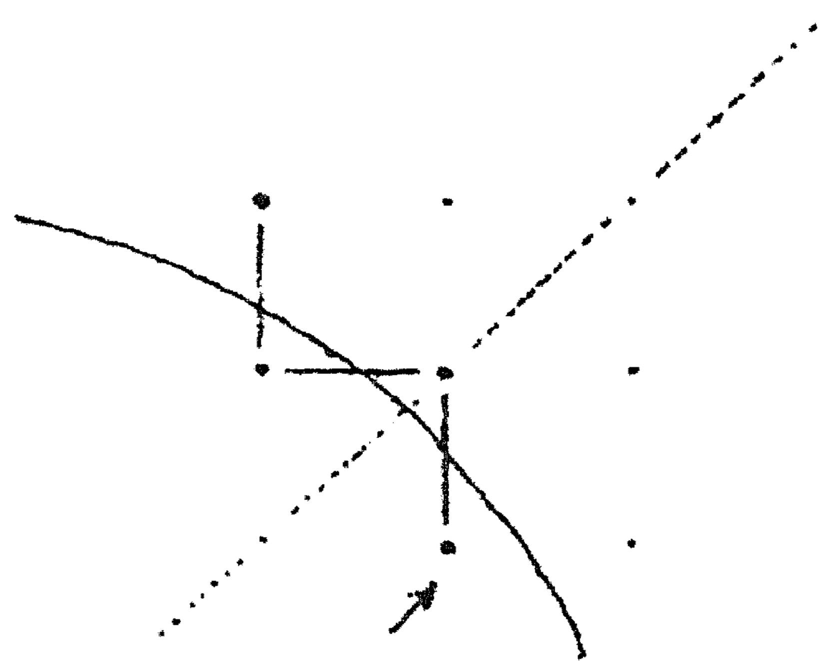


fig.3

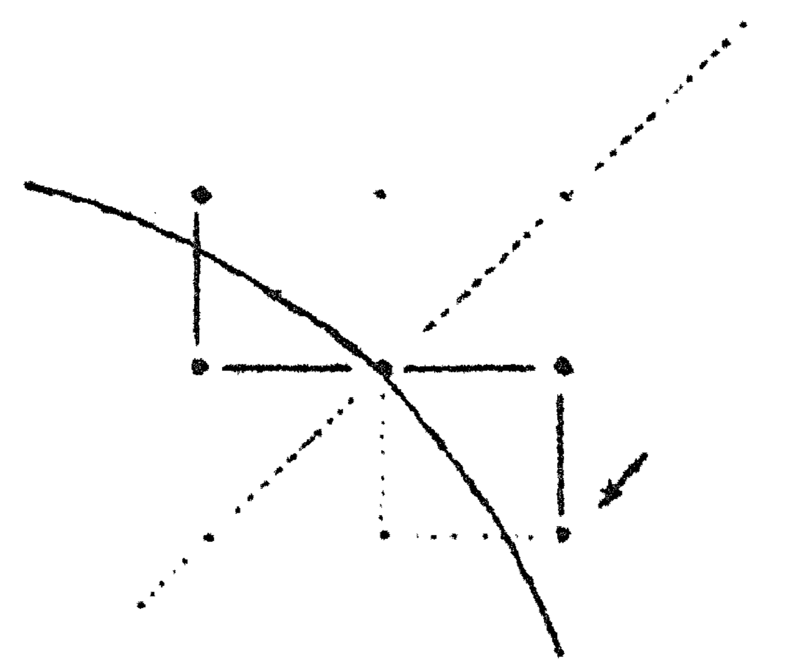


fig.4

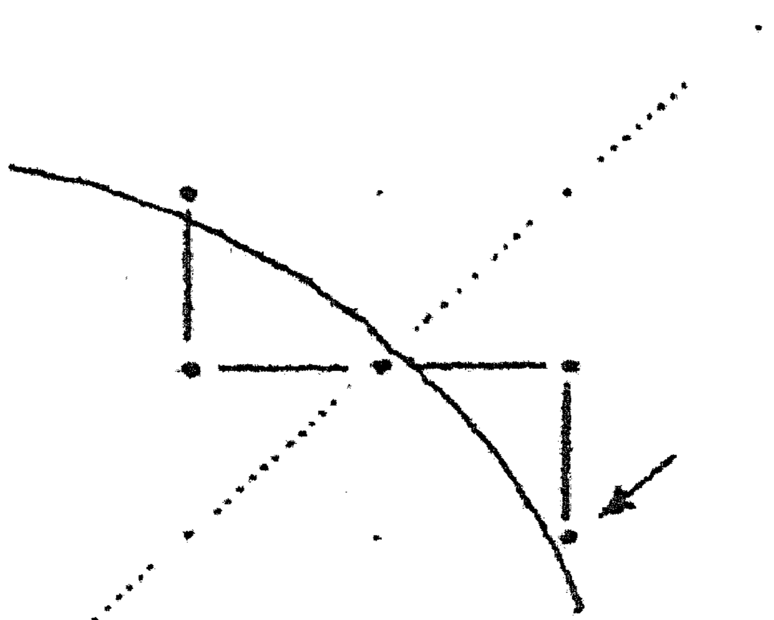


fig.5

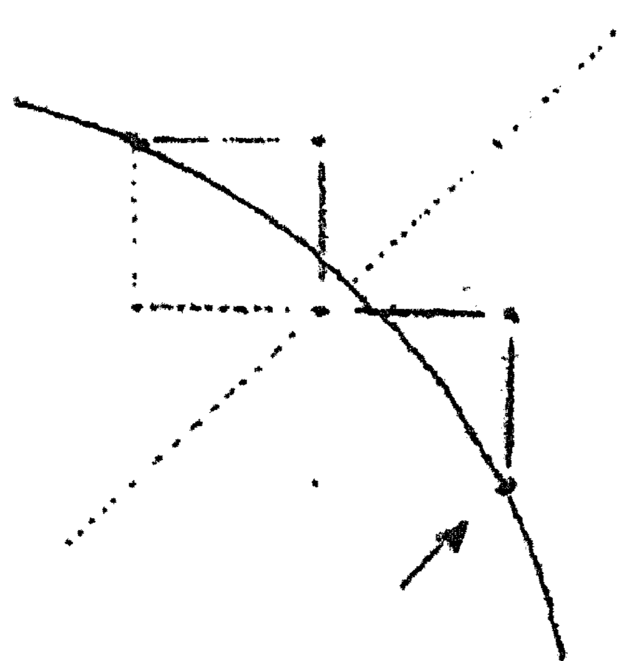


fig.6

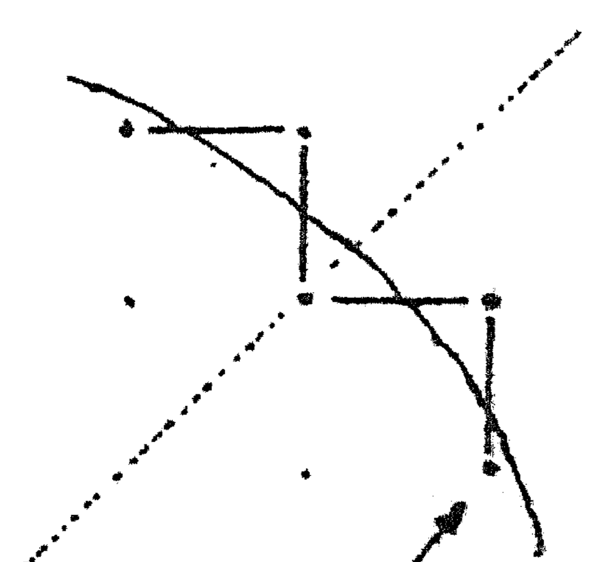


fig.7

waarde door y van nul met sprongen te vermeerderen, totdat $y^2 - G > 0$ ($y^2 - G = 0$, wat zich alleen voordoet, als G van de gedaante $G = n^2s^2$ is, laten we schieten. Dit is in tijd gemeten een, zij het heel klein, efficiencyverlies.)

Om te onderzoeken, hoe we het beste het einde kunnen testen, bekijken we, op hoeveel verschillende manieren de cirkel $x^2 + y^2 = G$ bij de lijn $x - y = 0$ door en langs de roosterpunten kan gaan.

In fig. 2 t/m fig. 7 zijn zes opeenvolgende gevallen van gedrag in de buurt van het snijpunt getekend. (Gestippeld is de route aangegeven, die gevolgd zou zijn, als na een roosterpunt op de cirkel eerst y zou zijn afgelaagd). Omdat (fig. 4) op de lijn $x = y$ nog een oplossing kan liggen, moet het zoeken naar roosterpunten pas ophouden, als $x > y$. Zodra door een nieuw punt aan deze ongelijkheid voldaan is, wordt dit punt niet meer onderzocht en is de berekening klaar. Inspectie toont aan, dat dit „alarm" niet te laat komt, als het gegeven wordt bij de in figuren 2 t/m 7 met pijltjes aangegeven punten. Dankzij onze extra-afspraak, dat na een oplossing eerst x opgehoogd wordt, worden deze punten alle bereikt na aflaging van y. Dit betekent, dat de test op het einde hoeft te geschieden na aflaging van y, d.w.z. slechts bij 30% van het totale aantal onderzochte roosterpunten. Dit is dan ook de reden, waarom de extra-afspraak juist zo gemaakt werd.

In fig. 8 is de berekening in blokschema weergegeven. Hier wordt met „TWNR" de inloopsaanroep van de standaardtypcontrole bedoeld, de Tab en TWNR na het typen is verzwegen; n, het aantal getallen per regel, is even. Bij fig. 8 staat het onderschrift „Het logische blokschema", omdat de arithmetische operaties hierin slechts vaag zijn aangeduid. Het is kennelijk niet de bedoeling, dat per rondgang door het cyclusje twee vermenigvuldigen (nl. voor x^2 en y^2) uitgevoerd worden: van deze twee quadraten is x^2 resp. y^2 de vorige keer al berekend, terwijl $(y + 1)^2$ resp. $(x - 1)^2$ toen gevormd werd, als nu y^2 resp. x^2 moet worden berekend: we gaan de successieve waarde van $T = G - x^2 - y^2$ berekenen, door steeds bij de vorige T de wijziging (met inbegrip van teken) op te tellen.

Hiertoe gaan we met differenties werken, elke (nieuwe) x, resp. y karakteriserend door de laatst gebruikte differentie van x^2, resp. y^2. De drie ophogingen worden dan:

1:tijdens $y + s \Rightarrow y$

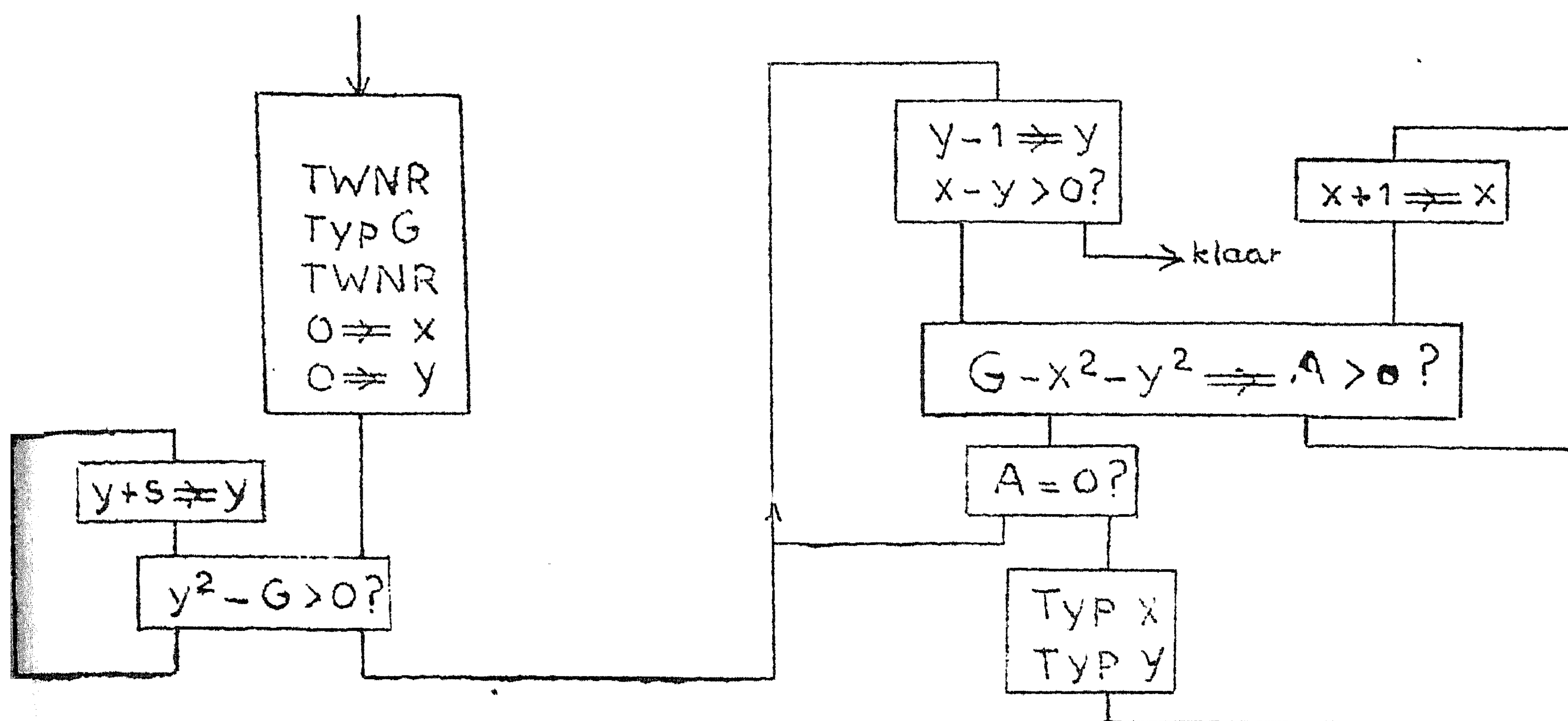


fig. 8 Het logische blokschema.

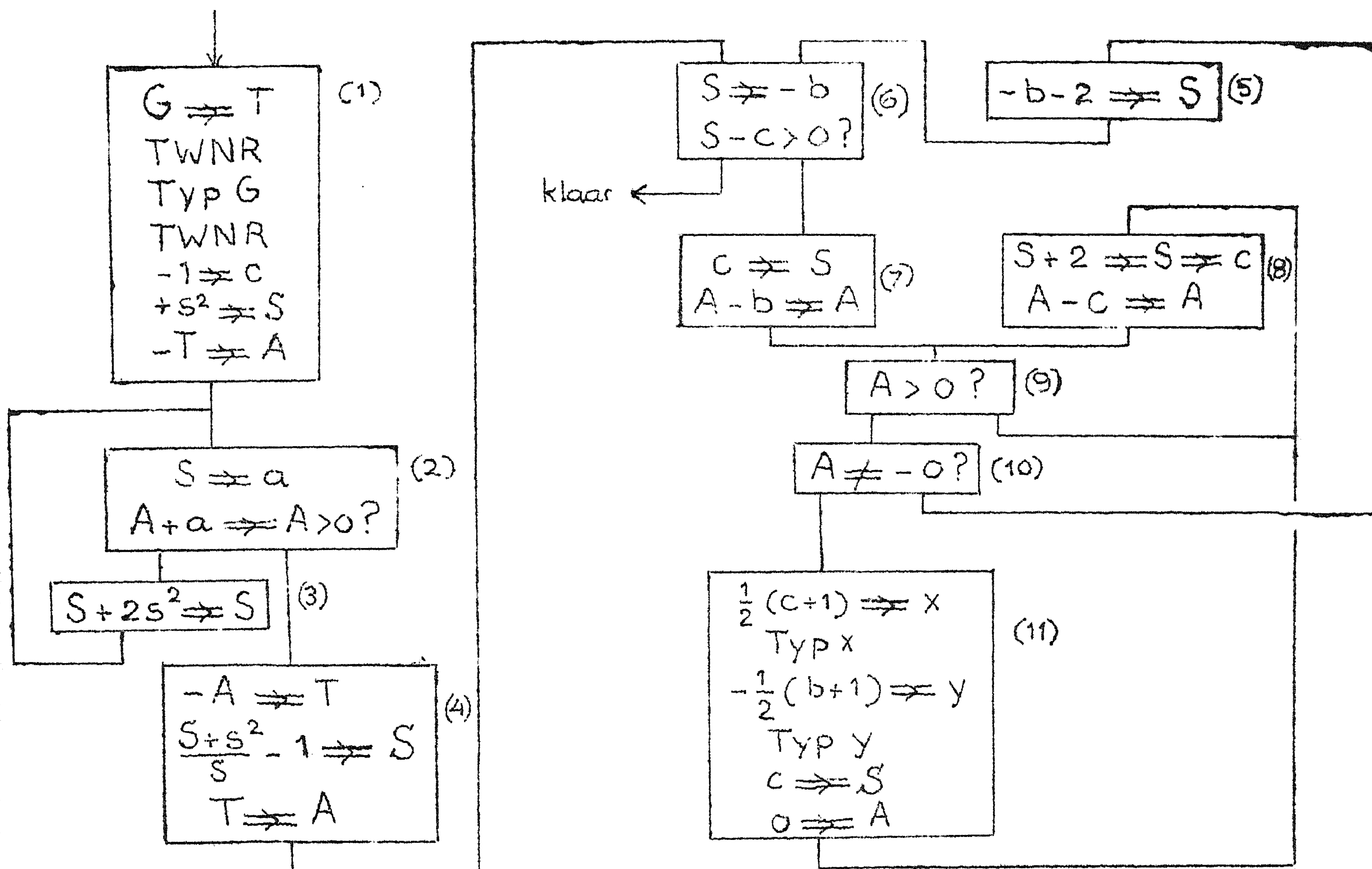


fig. 9 Het arithmetische blokschema.

Het verschil met het vorige quadraat:

$a \equiv y^2 - (y - s)^2 = 2sy - s^2$

dus $0 \Rightarrow y$ wordt $-s^2 \Rightarrow a$

en $y + s \Rightarrow y$ wordt $a + 2s^2 \Rightarrow a$

2: tijdens $y - 1 \Rightarrow y$

Het verschil met het vorige quadraat:

$b \equiv y^2 - (y + 1)^2 = -2y - 1$

De overgang van a naar b als voor y karakteristieke grootheid impliceert de substitutie

$$\frac{-(a + s^2)}{s} - 1 \Rightarrow b$$

waarna $y - 1 \Rightarrow y$ wordt $b + 2 \Rightarrow b$

3; tijdens $x + 1 \Rightarrow x$.

Het verschil met het vorige quadraat:

$c \equiv x^2 - (x - 1)^2 = 2x - 1$

dus $0 \Rightarrow x$ wordt $-1 \Rightarrow c$

en $x + 1 \Rightarrow x$ wordt $c + 2 \Rightarrow c$

Bij de operatie „typ x" moet x uit c berekend worden: $\frac{1}{2}(c + 1) \Rightarrow x$ evenzo bij „typ y" met gebruikmaking van b: $-\frac{1}{2}(b + 1) \Rightarrow y$.

Hier schuilt de reden, waarom we bij het constateren van de beginwaarde van y het geval, dat we op een roosterpunt terecht kwamen, hebben laten schieten: dan is y nl. nog door a bepaald. Voor een relatief zeldzaam geval zou het programma merkbaar langer worden.

Deze gang van zaken wordt nu weergegeven in het arithmetisch blokschema (fig. 9). Omdat dit een klein blokschema is, kunnen we ons veroorloven, hier al gedetailleerd aan te geven, in welke registers de berekening zich afspeelt. (Uit snelheidsoverweging willen we hier nl. het aantal operaties, zoveel als makkelijk mogelijk is, beperken. Een arithmetisch ingewikkeld blokschema zou door een zo gedetailleerde beschrijving er alleen maar onduidelijker op worden.)

Als regel reserveren we het A-register voor $T = G - x^2 - y^2$. Omdat we tijdens de cyclus $y + s \Rightarrow y$ moeten testen, of $y^2 - G = x^2 + y^2 - G = -T > 0$? plaatsen we hier $-T$ in A. Tijdens het typen is A bezet, maar dan weten we, dat $A = 0$ is. Tijdens de cyclus $y + s \Rightarrow y$ is S voor a aangewezen. Omdat " in de rechterhelft" de cyclus $x + 1 \Rightarrow x$ het meest doorlopen wordt, reserveren we daar het S-register voor c; zodra we in de cyclus $y - 1 \Rightarrow y$ komen, voeren we de daar aangegeven bewerkingen in S uit, d.w.z. wijziging van b, en uitvoering van

de test. De oorspronkelijke voorwaarde luidde:

<u>klaar</u> als $x - y > 0$

dus " $x - y - 1 \geq 0$

" " $2x - 2y - 2 = (2x - 1) + (-2y - 1) \geq 0$

" " $b + c \geq 0$

" " $-b - c \leq 0$, dus

<u>doorgaan</u> als $-b - c > 0$.

En hieruit volgt, dat we bij de wijziging van y dus - b in S moeten vormen om er meteen c van af te kunnen trekken.
Voor s kiezen we 16; de deling door S is dan als schuifoperatie uit te voeren.

In het programma zijn de blokken door stippellijntjes gescheiden. De rangschikking van de opdrachten, evenals het inlassen van enkele skipopdrachten, en het plaatsen der getallen is niet op de meest voor de hand liggende wijze gebeurd. Dit is gedaan uit snelheidsoverwegingen. (Omdat elk adres periodiek beschikbaar is, kan men door goede relatieve plaatsing van opdracht en getal, soms "een omwenteling winnen".)

Het programma worde gestart op a OAO, met [S] = G. (via het toetsenbord is het nl. mogelijk een getal in S te brengen) De typconstantes voor de typcontrole is ingesteld op 6 getallen, d.w.z. 3 oplossingen per regel. Door de tabulatorstoppen geschikt te plaatsen, kan men bijbehorende kolommen dichter bij elkaar zetten (vanaf de kantlijn 5,9,12,9,12,9,(9)).

Voorbeeld Cursus: splitsing in twee quadraten.

		A		AO	
		A		AO	
	0 AO	12	5	A1	(1) T
		24	6	XO	
	1	7	19	X2	=) TWNR
b4AO-+		10	5	A1	T
	2	24	6	XO	
		7		X3	=) Typ G
	3	9	5	A1	T
		24	6	XO	
	4	7		X2	=) Typcontrole
		14	1	AO	-+
	5	24	6	XO	
		7	19	X2	=)
	6	3	13	A1	1
		10	12	A1	$s^2 = 256$
	7	4	10	A1	C
		3	5	A1	T
a11AO-+	8	12	6	A1	(2) a
		0	6	A1	a
	9	24		XO	
		24	8	X1	
	10	14	11	AO	-+
		8	8	A1	(3)$2s^2 = 512$
	11	7	8	AO	=+
a10AO=+		8	12	A1	(4)$s^2 = 256$
	12	5	11	A1	T
		23	4	XO	
	13	9	13	A1	1
		2	11	A1	T*
	14	7	16	AO	=+
a23AO=+		11	4	A1	(5)2
	15	24		XO	
		9	7	A1	b

Kanaal AO

a14AO-+	16 AO	13	7	A1	(6)b
		9	10	A1	c
	17	13	15	A1	S ≤ - 0?
		1	7	A1	b
	18	24	2	XO	Stop als kla
		10	10	A1	(7)c
	19	15	21	AO	=+
a22AO, b3A1 =+		8	9	A1	(8)2
	20	12	10	A1	c
		1	10	A1	c
	21	24		XO	
a19AO-+		24	8	X1	(9)
	22	14	19	AO	-+
		24	8	X3	(10)
	23	14	14	AO	-+
		2	10	A1	(11)c
	24	24	1	X16	+1
		22	1	XO	
	25	4	15	A1	x
b28AO-+		10	15	A1	x
	26	24	6	XO	
		7		X3	=) Typ x
	27	9	15	A1	x
		24	6	XO	
	28	7		X2	=) Typcontro
		14	25	AO	-+
	29	2	7	A1	b
		24	1	X16	+1
	30	22	1	XO	
		5	14	A1	y
a2AO-+	31	10	14	A1	y
		24	6	XO	

Voorbeeld Cursus: splitsing in twee quadraten.

```
      |  A           A1 |                      A     230    X24 )     Kanaal A1
      |  A           A1 | -                    A     230    X24 )
------+-----------------+------------          + 1000000      X  }    Typconstantes
 OA1  |  7           X3 | =) Typ y                                )   G  6  fac.
      |  9    14     A1 |    y                 7      29     X2 )
   1  | 24     6     X0 |                      7       5     X4 )
      |  7           X2 | =) Typcontrole       A     242    X24 )
   2  |  6    31     A0 | -+                   A     242    X24 )   Typconstantes
      | 10    10     A1 |   c                  +      85      X  }  n = 6
   3  | 23           X0 |  0 ⇏ A                                 )   TAB.
      | 15    19     A0 | =+                   +     251      X )
   4  | +      2      X |
   5  |                 | =T;
   6  |                 | =a
   7  | A      8     A1 | =b
      | A      8     A1 |
   8  | + 512         X | =2s^2
   9  | +      2      X |
  10  |                 | =c
  11  | A     12     A1 |   T*
      | A     12     A1 |
  12  | + 256         X |  =s^2
  13  | +      1      X |
  14  |                 |  =y
  15  |                 |  wr. test einde;
      |                 |  tevens = x
```

Syllabus No. 9 van de cursus 1955-'56:
Programmeren voor automatische rekenmachines
onder leiding van
Prof. Dr. Ir. A. van Wijngaarden en de Heer E.W.Dijkstra

SUBROUTINES II

Thans is aan de orde de bespreking van enkele subroutines ter berekening van zg. elementaire functies. Het gaat hier doorgaans om functies van één onafhankelijk veranderlijke, een „argument", zoals wij zeggen, (dat wij steeds in S zullen meegeven), terwijl ook het antwoord als regel uit één getal zal bestaan (dat dan, der wille van de uniformiteit, in S achtergelaten zal worden.) Wij zullen onze aandacht speciaal schenken aan die gevallen, waar de herleiding tot de directe operatie's (optelling, vermenigvuldiging etc.) der machine niet evident is, dus met voorbijgaan van die functies, waar, als b.v. bij de polynoomberekening, de werkelijke berekening practisch geheel in het wiskundig formalisme ligt opgesloten. Een voorbeeld hiervan hebben we hiervan gezien, nl. een iteratieve methode ter berekening van de vierkantswortel; voor de volledigheid nemen we ook deze op in ons overzicht.

De tweedemachtswortel.

Eerste methode (iteratief, met deling)

De iteratie geschiedt volgens:

$$\tfrac{1}{2}\left(\frac{a}{x_n} + x_n\right) = \tfrac{1}{2}\left(\frac{a}{x_n} - x_n\right) + x_n = x_{n+1} \qquad \lim x_n = a^{\frac{1}{2}}$$

Bewijs: Stel $x_n = a^{\frac{1}{2}}(1 + d_n)$ d.w.z. d_n is de zg. relatieve fout

van de n^{de} schatting x_n; dan is

$$\frac{a}{x_n} = \frac{a^{\frac{1}{2}}}{1 + d_n} = a^{\frac{1}{2}}(1 - d_n + d_n{}^2 - \ldots\ldots..)$$

waaruit wij vinden:

$$x_{n+1} = \tfrac{1}{2}\left(x_n + \frac{a}{x_n}\right) = a^{\frac{1}{2}}(1 + \tfrac{1}{2}d_n{}^2 - \ldots..) = a^{\frac{1}{2}}(1 + d_{n+1}).$$

Omdat in eerste benadering geldt $d_{n+1} = \frac{1}{2}d_n{}^2$ noemt men dit iteratieschema quadratisch convergent. Per iteratiestap moet een deling uitgevoerd worden. De eerste schatting x_0 kan verschillend ge-

maakt worden: essentieel is dat de eerste deling geen capaciteits-overschrijding ten gevolge heeft. Verfijning van de constructie van x_0 is de meest effectieve versnelling van dit procédé, dat mits de machine beschikt over een snelle ingebouwde deling, als regel te verkiezen is boven de volgende methodes.

Tweede methode (iteratief, zonder deling)

Als de machine niet over een ingebouwde deling beschikt - en deze dus (zie later) geprogrammeerd moet worden, is uit tijds-overweging de eerste methode minder aantrekkelijk. Zonder delingen zijn er iteratieschema's voor de reciproke wortel $a^{-\frac{1}{2}}$, bv.

$$b_n\left(\frac{3}{2}-\frac{1}{2}ab_n^2\right) = b_{n+1} \quad \text{(quadratisch)} \quad b = \lim b_n = a^{-\frac{1}{2}}$$

$$b_n\left\{\frac{15}{8} - \frac{10}{8}(ab_n^2) + \frac{3}{8}(ab_n^2)^2\right\} = b_{n+1} \text{ (cubisch) } b = \lim b_n = a^{-\frac{1}{2}}$$

Men bewijst deze formules door $b_n = a^{-\frac{1}{2}}(1 + d_n)$ te stellen en in de uitdrukkingen voor b_{n+1} te substitueren; tevens dient d_0 voldoende klein te zijn, om convergentie van d_n naar nul te verzekeren.

Door het iteratieschema vooraf te laten gaan door

$$|x| \Rightarrow a$$

door het af te sluiten met de vermenigvuldiging

$$ab \Rightarrow y$$

is de opgave

$$|x|^{\frac{1}{2}} \Rightarrow y$$

volbracht.

Echter: essentieel wordt volgens deze methodes eerst de reciproke van ons antwoord uitgerekend. Dit heeft enige directe gevolgen ten eerste is het onmogelijk, om op deze manier de wortel uit het getal nul te trekken; ten tweede is het, zeker in een machine met vaste komma (capaciteit tussen -1 en + 1) gewenst, dat a zo dicht mogelijk bij +1 ligt (als a dicht bij 0 ligt, wortd immers $a^{-\frac{1}{2}}$ verschrikkelijk groot!) In de hieronder uitgewerkte cubische iteratie wordt dit bereikt door de eerste substitutie te vervangen door

$$|x|2^n \Rightarrow a \quad (n \geq 0\text{ , bepaald door } \tfrac{1}{2} \leq a < 1)$$

Deze „normering" van a wordt na afloop in rekening gebracht („gecompenseerd") door

$$ab.2^{-\frac{1}{2}n} \Rightarrow y$$

Door het probleem te herleiden tot een breuk in het afgeperkte

gebied tussen $\frac{1}{2}$ en 1 is tweeerlei winst geboekt: ten eerste is het nu mogelijk eenvoudig en snel (d.w.z. zonder vermenigvuldiging) een goede beginschatting te construeren, ten tweede kan de iteratieformule nu zo herschreven worden, dat alle tussenresultaten in absolute waarde < 1 zijn; dit laatste zou zonder het gegeven $\frac{1}{2} \leqslant a < 1$ heel lastig zijn geweest. Wij geven nu de herschreven iteratie:

Beginschatting:

$$0.9510555 - a \Rightarrow c_0$$

(in plaats van het getal b_n hanteert de machine $c_n = b_n - 1$: de beginschatting b_0, die overeenkomt met de boven gegeven c_0 benadert $a^{-\frac{1}{2}}$ met een relatieve fout, kleiner dan 0.049, mits $\frac{1}{2} \leqslant a < 1$! Maar aan deze voorwaarde is voldaan.)

Iteratieschema

$$\begin{aligned} ac_n &\Rightarrow z_n \\ c_n z_n + 2z_n + a - 1 &\Rightarrow q_n \\ \frac{3}{8} q_n^2 - \frac{1}{2} q_n &\Rightarrow t_n \\ c_n t_n + c_n + t_n &\Rightarrow c_{n+1} \end{aligned} \qquad c = \lim c_n = a^{-\frac{1}{2}} - 1$$

Voltooiing $ac + a \Rightarrow y$, waarmede de opgave $a^{\frac{1}{2}} \Rightarrow y$ volbracht is.

Opm.: Met de notatie $ac + a$ is bedoeld, dat het getal a bij het product ac wordt opgeteld; dit ter onderscheiding van $a(c + 1)$. De factor $(c + 1)$ zou nl. de capaciteit overschrijden. (Er is aangenomen, dat bij de vorming van een gedurige som de partiele sommen wel de capaciteit mogen overschrijden, mits het eindantwoord gegarandeerd weer binnen de capaciteit valt.) De rechtvaardiging en motivering van juist deze formules zal later gegeven worden; voorlopig kan de lezer controleren, dat ze aequivalent zijn met de formule voor de cubische iteratie en dat op deze wijze geen capaciteitsoverschrijding optreedt.

Een laatste opmerking: dankzij de relatief al heel goede beginschatting is na twee iteratie's de relatieve fout $|d_2| < 0.6 \times 10^{-10}$. Als deze precisie vereist en voldoende is, kan overwogen worden, om niet door te itereren tot het verschil $c_{n+1} - c_n$ klein genoeg is, maar om altijd twee iteratiestappen uit te voeren, zodat dan alle tests overbodig zijn. Het gebied voor a, waarin één iteratie al genoeg zou zijn, is nl zo klein, dat het waarschijnlijk inefficiënt is, om voor ieder geval de testen uit te voeren, ter versnelling van slechts één op de zoveel.

Ter explicatie van het blokschema (fig. 1) dat deze methode in beeld brengt, het volgende. Dit is - dat weten we, omdat anders

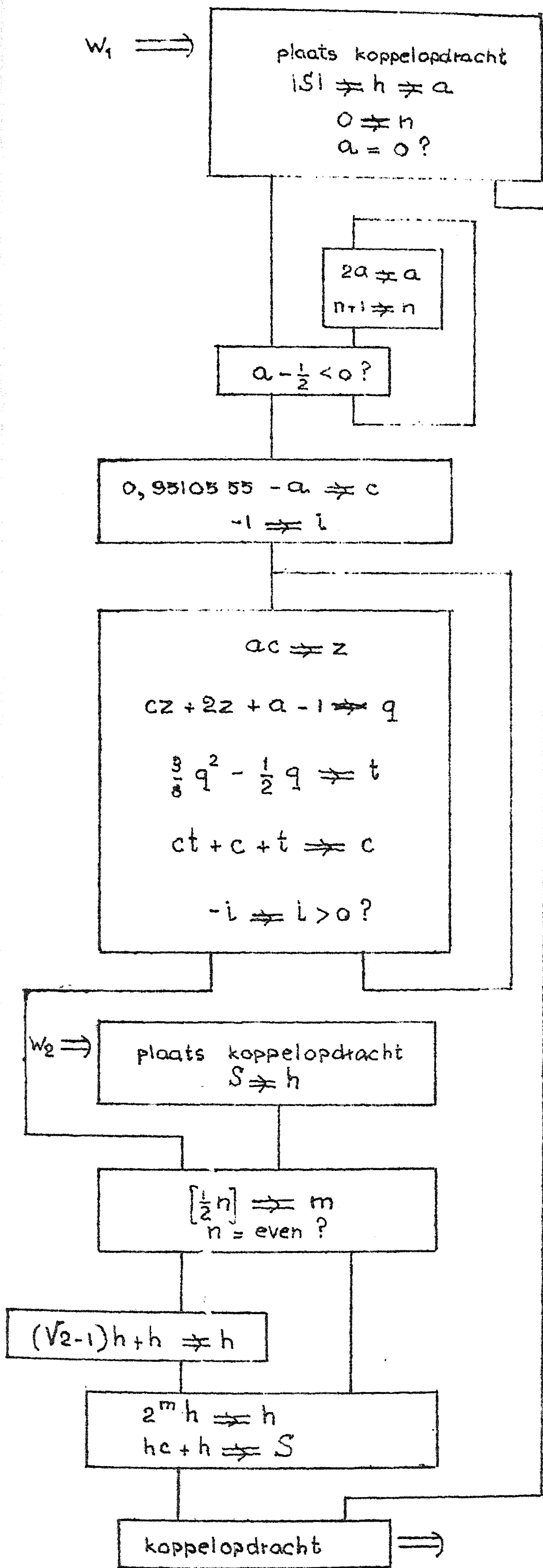


fig.1 Blokschema $|S|^{\frac{1}{2}} \Rightarrow S$
d.w.z. Worteltrekking uit een echte breuk (iteratief).

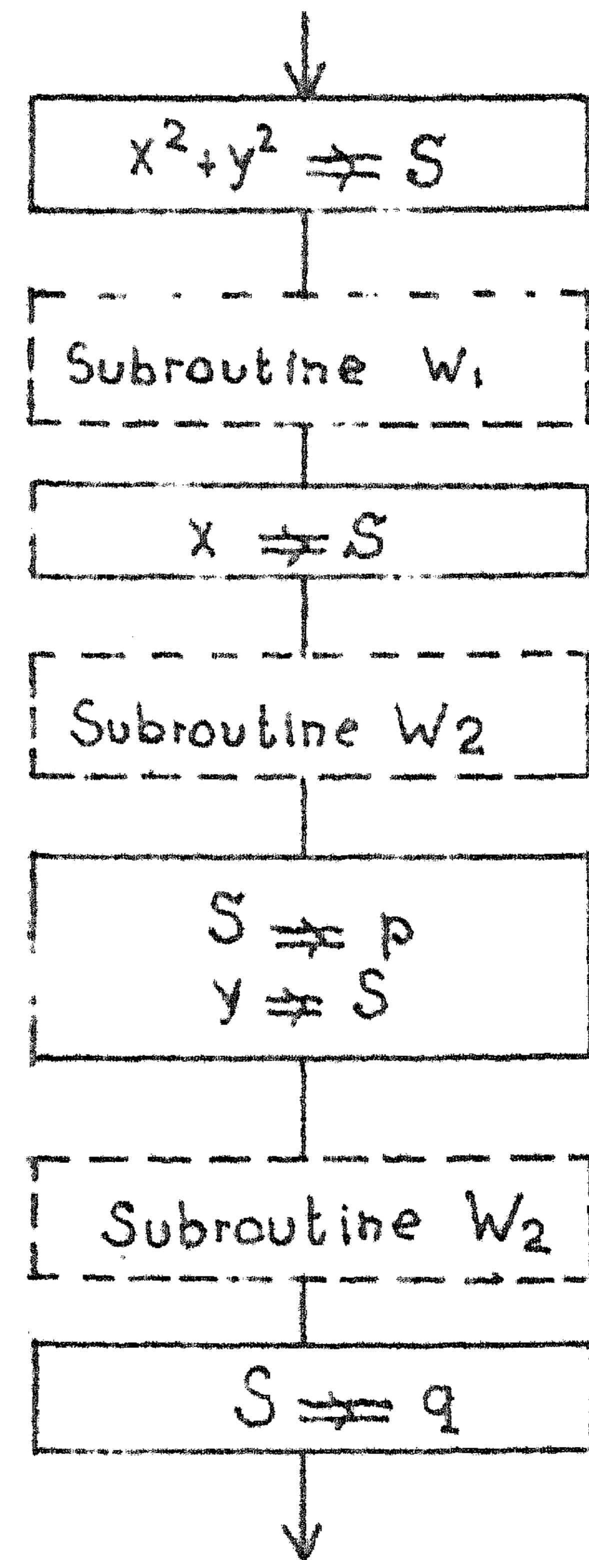


fig.2 Blokschema $\frac{x}{\sqrt{x^2+y^2}} \Rightarrow p$; $\frac{y}{\sqrt{x^2+y^2}} \Rightarrow q$

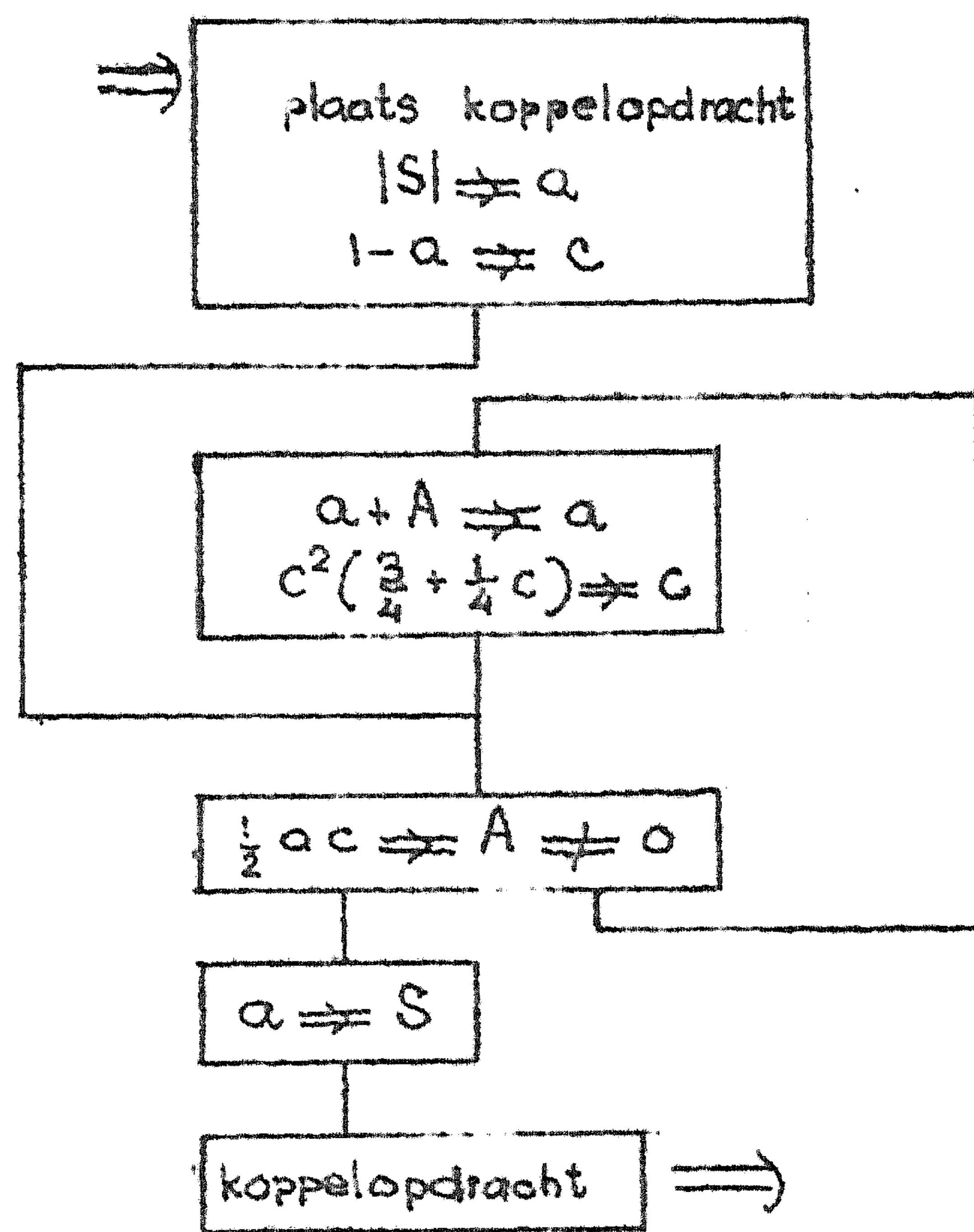


fig.3 $|S|^{\frac{1}{2}} \Rightarrow S$; (repetitief)

de eerste methode gekozen zou zijn! - een blokschema voor een machine, die geen ingebouwde deling bezit. Daarom ligt het voor de hand, om in het geval, dat door de wortel gedeeld moet worden, te benutten, dat eigenlijk eerst de reciproke wortel uitgerekend wordt.

We kunnen dit verwezenlijken door bv. de subroutine 2 aanroepen te geven, nl.

W1 met de functie $|\{S\}|^{\frac{1}{2}} \Rightarrow \{S\}$

W2 " " " $\{S\}\, w^{-1} \Rightarrow \{S\}$, als w de laatst getrokken wortel is.

(Door elke worteltrekking geraakt de W-subroutine „in een bepaalde toestand": kan nl. elk gewenst aantal malen nog door deze wortel delen; w wordt dus op een of andere manier „onthouden".)
Het gebruik van de aanroep W2 wordt geillustreerd in fig. 2.

<u>Derde methode</u> (Repetitief; zonder deling)

De tweedemachtswortel uit een echte breuk kan eveneens zonder delingen berekend worden met een zg. <u>repetitief proces</u>; het quadratisch convergerende laten wij hieronder volgen. Wederom wordt in eerste instantie de reciproke wortel uitgerekend.

De relatie

$$\frac{1}{(1 - c_n)^{\frac{1}{2}}} = (1 + \tfrac{1}{2}c_n)\,\frac{1}{(1 - c_{n+1})^{\frac{1}{2}}}$$

wordt opgevat als definitievergelijking van c_{n+1}; oplossing van c_{n+1} geeft

$$c_n^{\,2}(\tfrac{3}{4} + \tfrac{1}{4}c_n) = c_{n+1}$$

Vatten wij deze relatie op als recurrente betrekking, dan geldt voor $|c_0| < 1$ zeker $\lim c_n = 0$, m.a.w.

$$\lim \frac{1}{(1 - c_n)^{\frac{1}{2}}} = 1$$

Door echter de formule in zijn eerste gedaante herhaald toe te passen, vindt men, als $c_0 = 1 - a$ $(0 < a \leqslant 1)$, na vermenigvuldiging van beide leden met a:

$$a^{\frac{1}{2}} = \frac{a}{1 - c_0} = a(1 + \tfrac{1}{2}c_0)\frac{1}{1 - c_1} = a(1 + \tfrac{1}{2}c_0)(1 + \tfrac{1}{2}c_1)(1+\tfrac{1}{2}c_2)$$

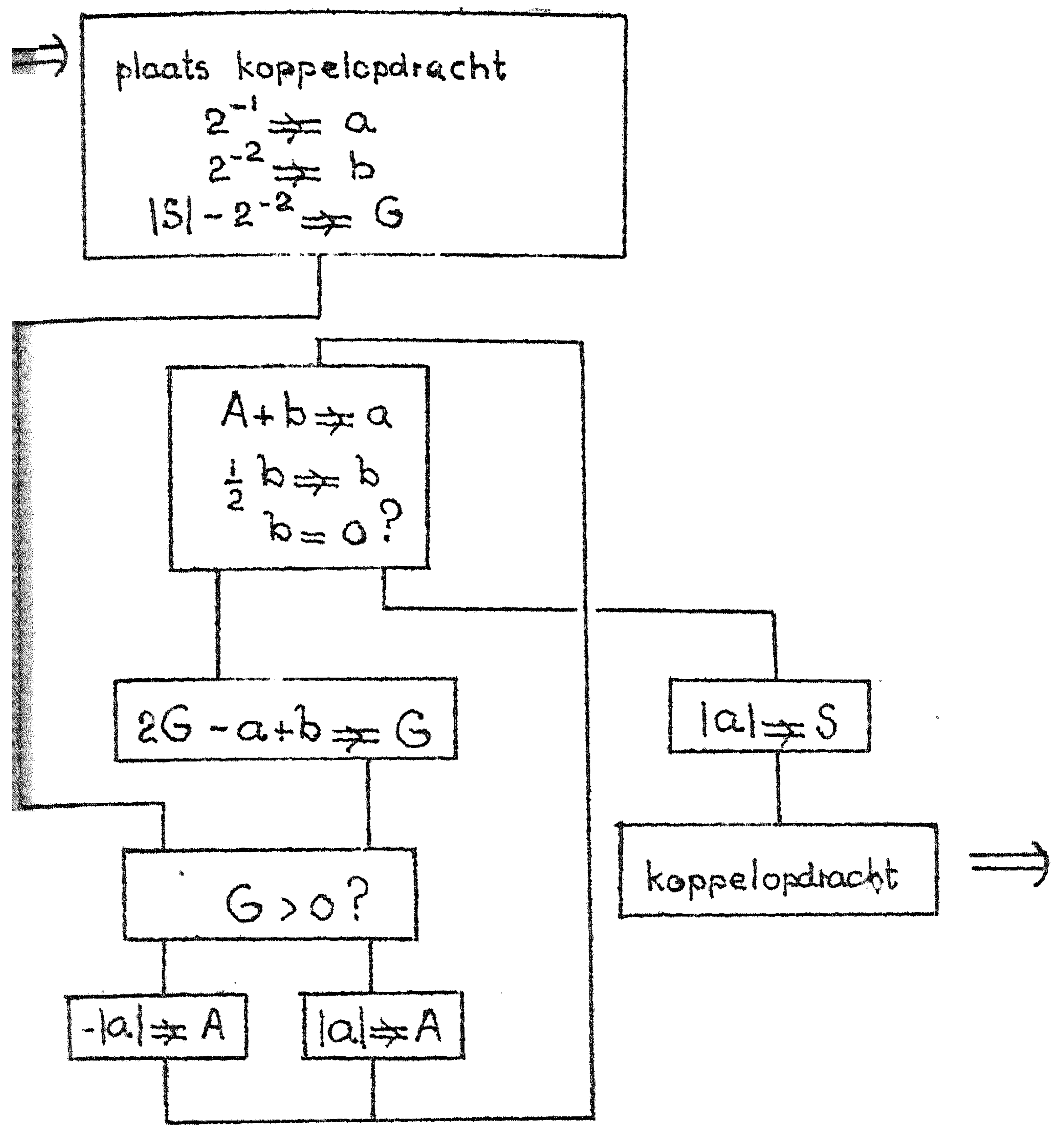


fig.4 De geprogrammeerde worteltrekking.

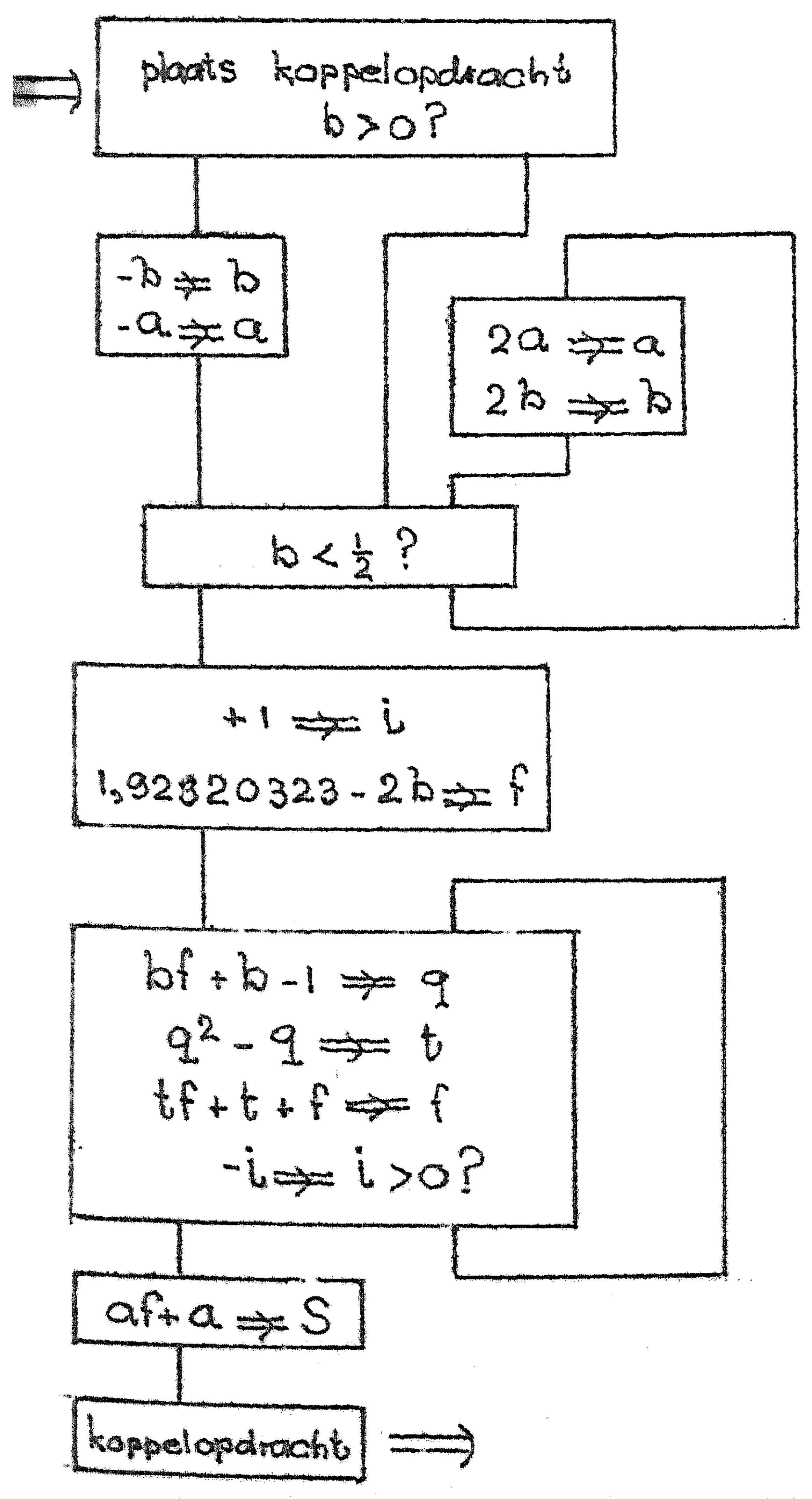


fig.5 De iteratieve deling.

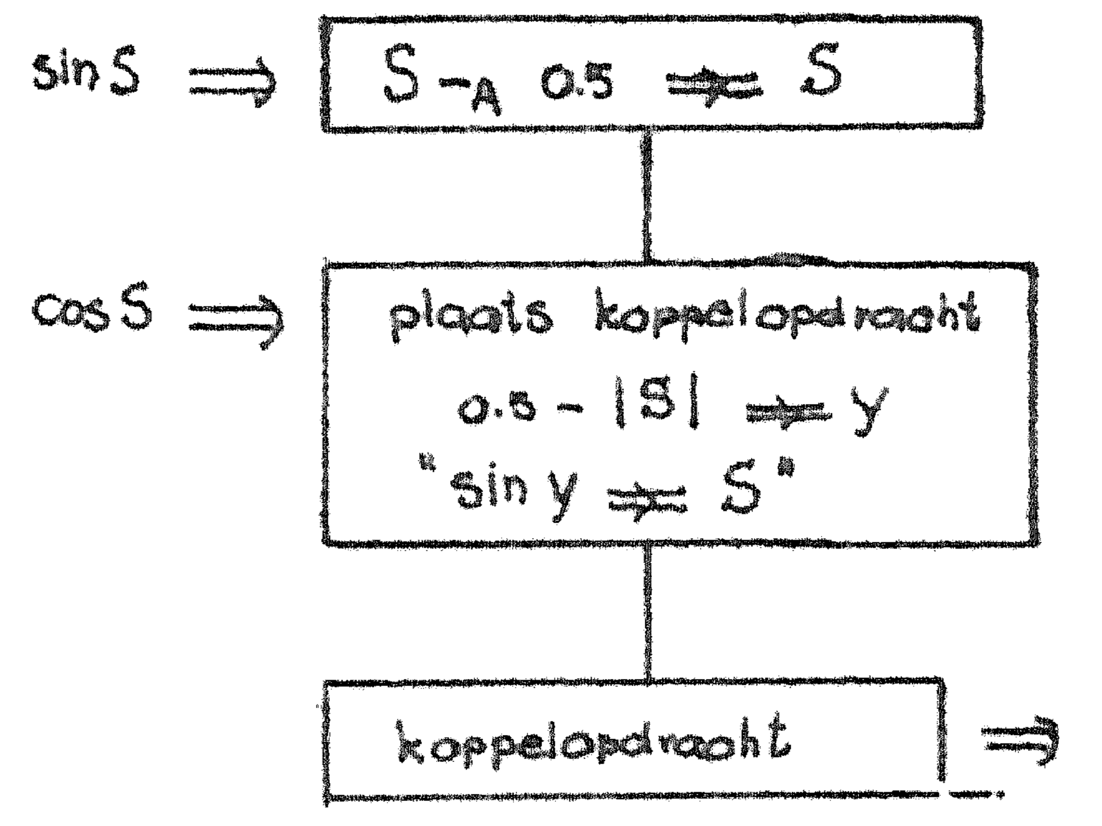


fig.6 De herleiding bij sin x en cos x van het argument in het 1e of 4e quadrant.

Hiermede is de wortel als oneindig, quadratisch convergerend product geschreven; zonder delingen kan de wortel hiermede getrokken worden.
In formule luidt dit proces:

na de slotwaarden:

$a \Rightarrow a_0 \quad 1 - a \Rightarrow c_0$

de repetitie:

$a_n + \frac{1}{2}a_n c_n \Rightarrow a_{n+1} \qquad c_n^2(\frac{3}{4} + \frac{1}{4}c_n) \Rightarrow c_{n+1}$

dan $\lim a_n = a^{\frac{1}{2}}$

In eenvoudige vorm is deze berekening in fig. 3 weergegeven. Hoewel $c_0=1$ als $a=0$, hoeft voor dit uitzonderingsgeval geen extra voorzorg genomen te worden. Door weer a dichtbij +1 te nemen, wordt de convergentie versneld. (Het verschil tussen het iteratieve en het repetitieve proces is, dat bij het iteratieve proces het argument in de formule van de cyclus voorkomt, terwijl de beginschatting - betrekkelijk - willekeurig is, terwijl bij het repetitieve proces het argument zich juist in de startwaarden doet gelden, via de startwaarden „wordt medegedeeld". Een repetitief proces is dus niet als het iteratieve te versnellen door een listige „eerste schatting".)

Vierde methode (De „geprogrammeerde" worteltrekking)

In fig. 4 is het blokschema weergegeven van de zg. geprogrammeerde worteltrekking. Dit is een versie van de „schoolworteltrekking", met dat verschil, dat de wortel niet decimaal voor decimaal. maar binaal voor binaal wordt (De methode is geinspireerd op een binale machine). De wortel wordt opgebouwd als $\sum d_i 2^{-i}$ met $d_i = \pm 1$ (en niet $d_i = 0$ of 1). De halveringen van b (zie fig. 4) worden niet afgerond uitgevoerd: op een gegeven ogenblik is b (in de precisie van de machine) = 0; dit is een indicatie voor het einde. fig. 4 geeft een blokschema voor een machine waar het nemen van de absolute waarde een directe operatie is. (Het blokschema is onverfijnd, wat betreft het einde: als men eisen van afronding stelt, of deze methode op gehele getallen loslaat en voor de kleinste positieve rest geinteresseerd is, moet het einde geraffineerder)

De geprogrammeerde worteltrekking convergeert lineair: de bepaling van elke binaal duurt even lang als de vorige. Men past evenwel dergelijke methodes toe als ook deling en vermenigvuldiging

niet zijn ingebouwd en analoog „geprogrammeerd" moeten worden. Men mag aannemen dat met de opdrachten, die dan wel in de machine zitten, het interne cyclusje heel compact geprogrammeerd kan worden.

De deling

Indien de machine niet beschikt over een ingebouwde deling, moet ook dit proces geprogrammeerd worden. De onderstaande methodes hebben betrekking op de deling van twee breuken, waarvan we aannemen dat het quotient in absolute waarde kleiner is dan 1.

Eerste methode (iteratief)

De iteratieve processen ter bediening van de reciproke b^{-1}, die in aanmerking komen, zijn:

$$c_n(2 - bc_n) = c_{n+1} \qquad \text{(quadratisch)} \qquad c = \lim c_n = b^{-1}$$

$$c_n\{3 - 3(bc_n) + (bc_n)^2\} = c_{n+1} \quad \text{(cubisch)} \qquad c = \lim c_n = b^{-1}$$

Opm.: Het bewijs van deze formules levert men gemakkelijk, door $c_n = b^{-1}(1 + d_n)$ in de linkerleden te substitueren: d_0 moet voldoende klein zijn, om convergentie te verzekeren. In de volgende uitwerking wordt $|d_0|$ heel veel kleiner gekozen, om snelle convergentie te garanderen.
Hier doen zich, bij de aanpassing aan een machine met vaste komma dezelfde moeilijkheden voor, als bij de iteratieschema's voor de reciproke wortel; zij worden op dezelfde wijze opgelost.
Men past de iteratie alleen toe voor $\frac{1}{2} \leq b < 1$ (moet dus elk geval hierop herleiden door teller en noemer met $\pm 2^n$ te vermenigvuldigen) en vindt na herschrijving:

Beginschatting:

$$1{,}92820\ 323 - 2b \Rightarrow f_0$$

(in plaats van c_n manipuleert de machine $f_n = c_n - 1$; de beginschatting, die met de boven gegeven f_0 overeenkomt, benadert b^{-1} met een relatieve fout, kleiner dan 0,072, mits $\frac{1}{2} \leq b < 1$)

Iteratieschema.

$$bf_n + b - 1 \Rightarrow q_n$$

$$q_n^2 - q_n \Rightarrow t_n$$

$$t_nf_n + t_n + f_n \Rightarrow f_{n+1} \qquad f = \lim f_n = b^{-1} - 1$$

Voltooiing: $af + a \Rightarrow y$, waarmede de opgave $\frac{a}{b} \Rightarrow y$ volbracht is.

Het blokschema in fig. 5 is zo uitgevoerd, dat steeds twee maal geitereerd wordt; de relatieve fout is dan (mathematisch!) maximaal $0.5.10^{-10}$, voldoende voor de precisie van ARMAC.

Tweede methode (Repetitief)

Analoog aan het repetitieve proces voor de worteltrekking kan de reciproke b^{-1} uitgerekend worden als bv. quadratisch convergerend product: men vindt dan

$$(1 - c)^{-1} = (1 + c)(1 + c^2)(1 + c^4)(1 + c^8)........$$

De deling $a.b^{-1} \Rightarrow y$ kan dan in de volgende stappen worden ontleed:

1e. Wissel a en b van teken, als b negatief is

2e. Vervang $a.2^n \Rightarrow a$ en $b.2^n \Rightarrow b$, zodat $\frac{1}{2} \leq b < 1$ (n 0)

3e. Zet beginwaarde $a \Rightarrow y_0$

$1 - b \Rightarrow c_0$

4e. Repeteer volgens $y_n + y_n c_n \Rightarrow y_{n+1}$

$c_n^2 \Rightarrow c_{n+1}$

tot $y_n c_n = 0$

Omdat $\lim y_n = y = ab^{-1}$ is hiermede het probleem opgelost.

Opm.1 De tweede stap is niet essentieel noodzakelijk. De convergentie kan er (bij kleine b) aanzienlijk mee versneld worden.

Opm.2 De nauwkeurigheid, waarmede het antwoord wordt afgeleverd is niet ideaal. Door de formules wat anders op te schrijven, is dit te verbeteren.

Het opstellen van het blokschema laten wij aan de lezer over.

Derde methode (De „geprogrammeerde" deling)

In twee gevallen bestaat de mogelijkheid dat een geprogrammeerde versie van de „schooldeling" de voorkeur verdient. Of als de machine niet over een snelle (d.w.z. ingebouwde) vermenigvuldiging beschikt (anders zijn de eerste twee methodes sneller) of als men gehele getallen op elkaar wil delen en in de rest geinteresseerd is.

Opm.1 Als normeren (d.w.z. bepalen hoeveel plaatsen een getal naar links geschoven worde, opdat het „kopcijfer"- het hoogste significante cijfer - tegen de komma aanstaat) een snelle operatie is, kan men hiermede de plaats van het hoogst mogelijke quotientcijfer localiseren.

Opm.2 Men moet in elke cijferpositie „aftrekken tot het niet meer kan", d.w.z. men trekt een keer te veel af, om tekenwisseling

te constateren. In plaats van die aftrekking ongedaan te maken, kan men ook de volgende keer gaan optellen. Dit versnelt het proces aanzienlijk; bij een tweetallige machine wordt het resultaat, dat er in elke cijferpositie steeds één keer wordt afgetrokken of opgeteld, afhankelijk van het teken van de „partiele rest": het aantal keren hoeft niet meer geteld worden.
Wij volstaan met deze aanduidingen, en laten de fijnere analyse aan de geinteresseerde lezer over.

<u>De sinus en de cosinus.</u>
Voor sin x en cos x bestaan snel convergerende machtreeksen. Men kan met de laagste macht beginnen en zolang volgende termen erbij uitrekenen, tot men een term gevonden heeft, waarvan de bijdrage verwaarloosbaar is. Op deze wijze spendeert men twee vermenigvuldigingen per term. Is echter van te voren bekend, welk aantal termen voldoende is, dan kan men de normale methode voor de polynoomberekening toepassen, beginnend bij de hoogste macht; deze wijze van werken kost één vermenigvuldiging per term. Om die reden is het voordeliger om een vast aantal - d.w.z. in elk geval genoeg - termen mee te nemen: men berekent sin x en cos x met behulp van een polynoombenadering voor een bepaald interval van de x (In de praktijk zijn de coefficienten van deze polynomen niet exact de laagste coefficienten van de Taylorontwikkeling, maar zijn ze wat gewijzigd: dit ter verhoging van de precisie). Algemeen geldt, dat de graad van het benaderende polynoom toeneemt met de vereiste preciesie, èn met de grootte van het interval voor x, waarin de benadering moet gelden. Daarom wordt, indien dit elegant mogelijk is, gebruik makend van de te benaderen functie, het interval voor x, waarin de polynoombenadering toegepast wordt, gereduceerd. Zo zal men sin x met een polynoom berekenen voor $-\frac{1}{2}\pi \leq x \leq \frac{1}{2}\pi$; in dit interval doorloopt immers sin x al zijn mogelijke waarden, de sinusberekening voor x buiten dit interval wordt op dit geval herleid, evenals de berekening van cos x = sin $(x + \frac{1}{2}\pi)$. Het zijn deze herhalingen binnen het interval, waar wij nu onze aandacht aan zullen schenken. De volgende herleiding is gebonden aan de logische constructie van de machine, nl. het gedrag bij capaciteistsoverschrijding door optelling (en aftrekking). In ARRA en ARMAC werkt de optelling Modulo 2 (exact $2-2^{-29}$, resp. $2-2^{-33}$, hier zien wij nu van af); de inhoud van het register wordt geinterpreteerd als liggend tussen -1 en +1. De optelling 0.7 + 0.4 geeft 1.1 - 2 = - 0.9. Omdat wij van deze optelling expliciet gebruik zullen maken, voorzien wij in deze beschouwing de + en

- tekens ter herinnering van de index A, dus bv. $0.7 +_A 0.4 \Rightarrow -0.9$. Omdat de periode van de functie sin x gelijk is aan 2π, ligt het nu voor de hand, om het argument x uit te drukken in eenheden π : zoals automatisch het antwoord tussen -1 en +1 geinterpreteerd wordt, wordt nu de som van een aantal hoeken automatisch tussen $-\pi$ en $+\pi$ geinterpreteerd: de reductie modulo 2 geschiedt nu zonder extra voorzorgen. Voor de sinusberekening wordt nu gebruik gemaakt (argumenten in radiaal uitgedrukt) van de relatie

$$\sin x = \sin y, \text{ als } 0.5 - |x -_A 0.5| \Rightarrow y$$

(en niet van $\sin x = \sin y$, als $|x +_A 0.5| - 0.5 \Rightarrow y$ in verband met de cosinusberekening, zie onder). Ter verificatie: als $x = 0.9$, dan $X -_A 0.5 = 0.4$ en $y = 0.1$; inderdaad is $\sin 0.9 = \sin 0.1$; als $x = -0.8$, dan $x -_A 0.5 = +0.7$ en $y = -0.2$. Aan $y \leqslant 0.5$ is kennelijk voldaan. Omdat $\cos x = \sin(x +_A 0.5)$ kunnen we de cosinus berekenen door optelling van 0.5; deze optelling effectueren we echter door de aftrekking van 0.5, waar de sinusberekening mee begint, over te slaan. In rudimentaire vorm is de gecombineerde subroutine $\sin\{S\}\pi \Rightarrow \{S\}$ en $\cos\{S\}\pi \Rightarrow \{S\}$ in fig, 6 weergegeven. De laatste regel „sin $y \Rightarrow S$" staat voor de polynoombenadering; omdat dit polynoom oneven is bespaart men vermenigvuldigingen door eerst y^2 uit te rekenen, dan een polynoom in y^2, dat tenslotte met y vermenigvuldigd wordt.

Zo wordt voor ARRA de volgende benadering gebruikt:

$$\tfrac{1}{2} \sin x = c_1 x + c_3 x^3 + c_5 x^5 + c_7 x^7 + c_9 x^9$$

waar x het argument uitdrukt in eenheden $\frac{1}{2}\pi$, de formule alleen toegepast mag worden in het 1^e^ en 4^e^ quadrant, en de c's gegeven zijn door:

$c_1 = .7853\ 9816$
$c_3 = -.3229\ 8186$
$c_5 = .0398\ 4484$
$c_7 = -.0023\ 3688$
$c_9 = .0000\ 7574$

Syllabus No. 10 van de cursus 1955-'56:
Programmeren voor automatische rekenmachines
Onder leiding van
Prof. Dr. Ir. A. van Wijngaarden en de Heer E.W. Dijkstra

SUBROUTINES III

Als voorbeeld van de goniometrische functies hebben we enige aandacht gewijd aan de sinus en de cosinus. Thans is het inverse probleem aan de orde: de cyclometrische functies, d.w.z. gegeven de waarde, die de bepaalde goniometrische functie heeft, wordt gevraagd, voor welke hoek de betroffen goniometrische functie deze gegeven waarde aanneemt. Wij zullen als voorbeeld de arctangens (d.w.z. de inverse functie van de tangens) behandelen.

Arctan $\frac{y}{x}$

In fig. 1 is (volgens definitie van $\tan\varphi$)

$$\tan\varphi = z, \text{ als } z = \frac{y}{x}$$

In woorden is dus φ een hoek, waarvan de tangens gelijk is aan z, wat in formule ook uitgedrukt kan worden (volgens de definitie van de cyclometrische functies) door

$$\arctan z = \varphi$$

Indien we echter. door de "pijl" naar R een halve slag om de oorsprong te draaien, ($180^\circ =$) π radiaal optellen, wisselen zowel x, als y van teken; hun quotient blijft daarbij echter gelijk; dus geldt

$$\tan(\varphi+\pi) = \tan\varphi$$

algemeen (tevens draaiingen naar de andere kant uitvoeren)

$$\tan(\varphi+n\pi) = \tan\varphi \quad \text{voor elke gehele } n\text{ .}$$

Dit is niet anders, dan het welbekende feit, dat de tangens van een hoek niet verandert door optelling of aftrekking van een geheel aantal malen π . In de taal van de arctan z betekent dit, dat deze functie door z niet eenduidig bepaald is, maar slechts op een veelvoud van π na. In de wiskunde is deze dubbelzinnigheid (beter "oneindigveelzinnigheid") opgeheven doordat een speciale keuze is gemaakt.

φ = arctan z is bepaald door $\tan\varphi = z$ en $-\frac{1}{2}\pi < \varphi < \frac{1}{2}\pi$ waar het geval $\varphi = \pm\frac{1}{2}$ hier buiten beschouwing gelaten is, omdat daar "$z = \pm\infty$" is.
(De situatie is analoog aan die bij de vierkantswortel van x, gedefinieerd als dat getal, dat in het quadraat gebracht, x oplevert: -2 is even goed $\sqrt{4}$ als +2; We spreken echter af, dat we met het symbool $\sqrt{x}$ de positieve wortel bedoelen dus in formule
$y = \sqrt{x}$ is bepaald door $y^2 = x$ en $|y| = y$.)

Voor automatisch rekenen is bij de arctangens echter een andere "bij afspraak" wenselijk en mogelijk. Daartoe beschouwen we de overgang tussen cartesiaanse coordinaten (x en y) en poolcoordinaten (r en φ) in het platte vlak (fig. 2). De positie van een punt P kan men specificeren door de projecties op de zg. coordinaat-assen; zo verkrijgt men de cartesiaanse coordinaten x en y. Een andere, veel gebruikte wijze is, dat men specificeert de zg. poolcoordinaten:

1. r = de afstand van P tot de oorsprong (= de lengte van de zg."voerstraal"OP)
2. φ = de hoek, tussen de voerstraal OP en de richting van de positieve x-as, precieser: de hoek, waarover men de positieve x-as (in de richting tegen de wijzers van de klok in positief gemeten) draaien moet, opdat hij samenvalle met de richting OP.

Omdat r hier per definitie positief is, is φ bepaald op een veelvoud van 2π na (met uitzondering van het geval r = 0, waar φ onbepaald is).
Als r en φ gegeven is, volgen x en y (zie fig. 2) uit de betrekkingen:

$x = r\cos\varphi$ en $y = r\sin\varphi$

Als x en y gegeven zijn, kunnen we omgekeerd r en φ berekenen. Volgens de stelling van Pythagoras geldt voor de voerstraal

$r = \sqrt{x^2 + y^2}$

Voor φ geldt wel $\tan\varphi = \frac{y}{x}$; echter $\varphi = \arctan\frac{y}{x}$ is niet zonder meer juist! Volgens boven gegeven conventie geldt $-\frac{1}{2}\pi < \varphi < \frac{1}{2}\pi$, d.w.z. we vinden alleen punten in het rechterhalfvlak (P_1' in plaats van P_1). Daarom definieren wij de arctan anders, als functie van twee veranderlijken, nl.

$$\varphi = \arctan\frac{y}{x} \quad \text{als} \quad \tan\varphi = \frac{y}{x}, \quad \text{en} \quad \begin{cases} 0 < \varphi < \pi & \text{als } y > 0 \\ -\pi < \varphi < 0 & \text{als } y < 0 \end{cases}$$

Aldus gedefinieerd loopt φ "helemaal rond". φ is positief in het bovenhalfvlak, negatief in het onderhalfvlak.

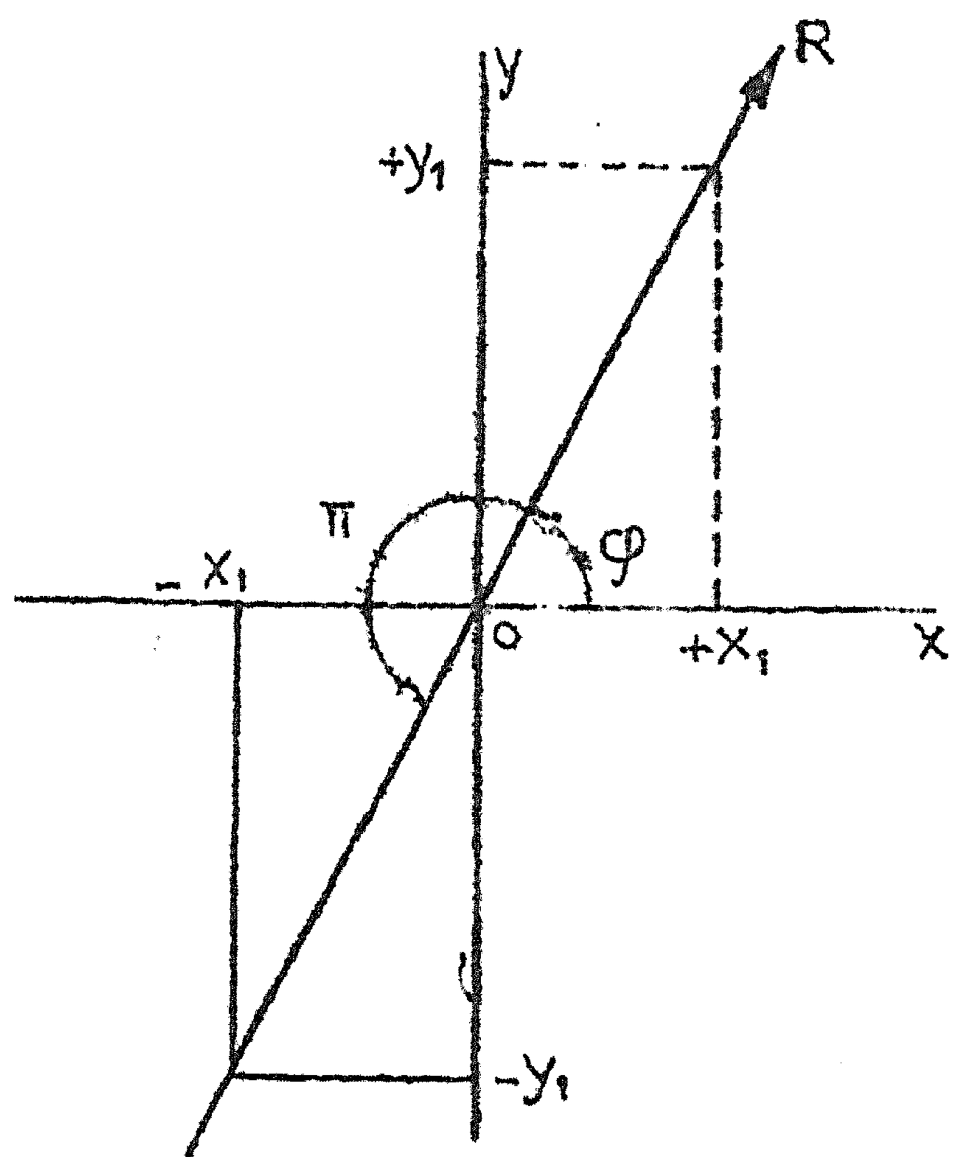


fig.1 $\tan\varphi = \frac{y_1}{x_1} = \frac{-y_1}{-x_1} = \tan(\varphi+\pi)$

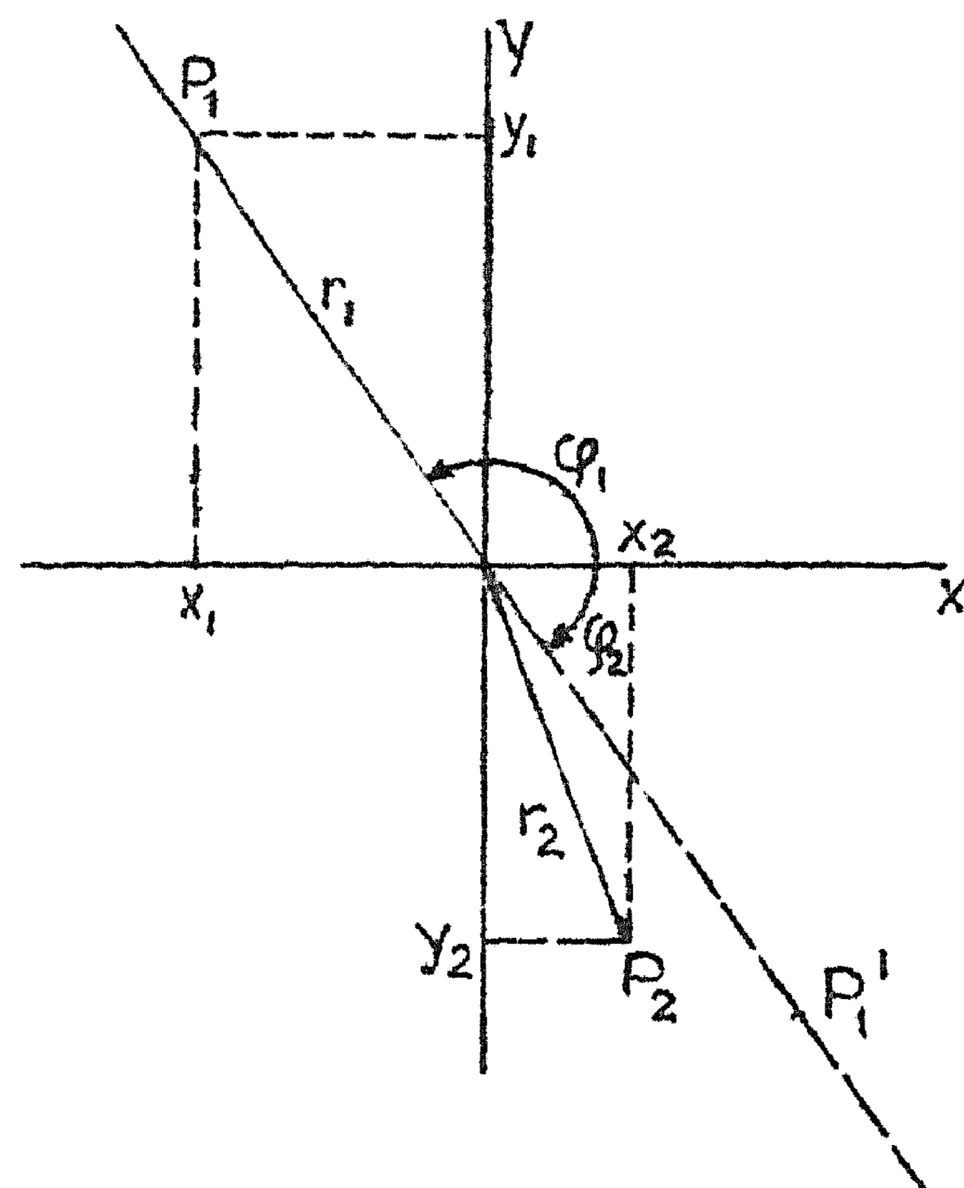


fig.2 Cartesiaanse coordinaten en poolcoordinaten.

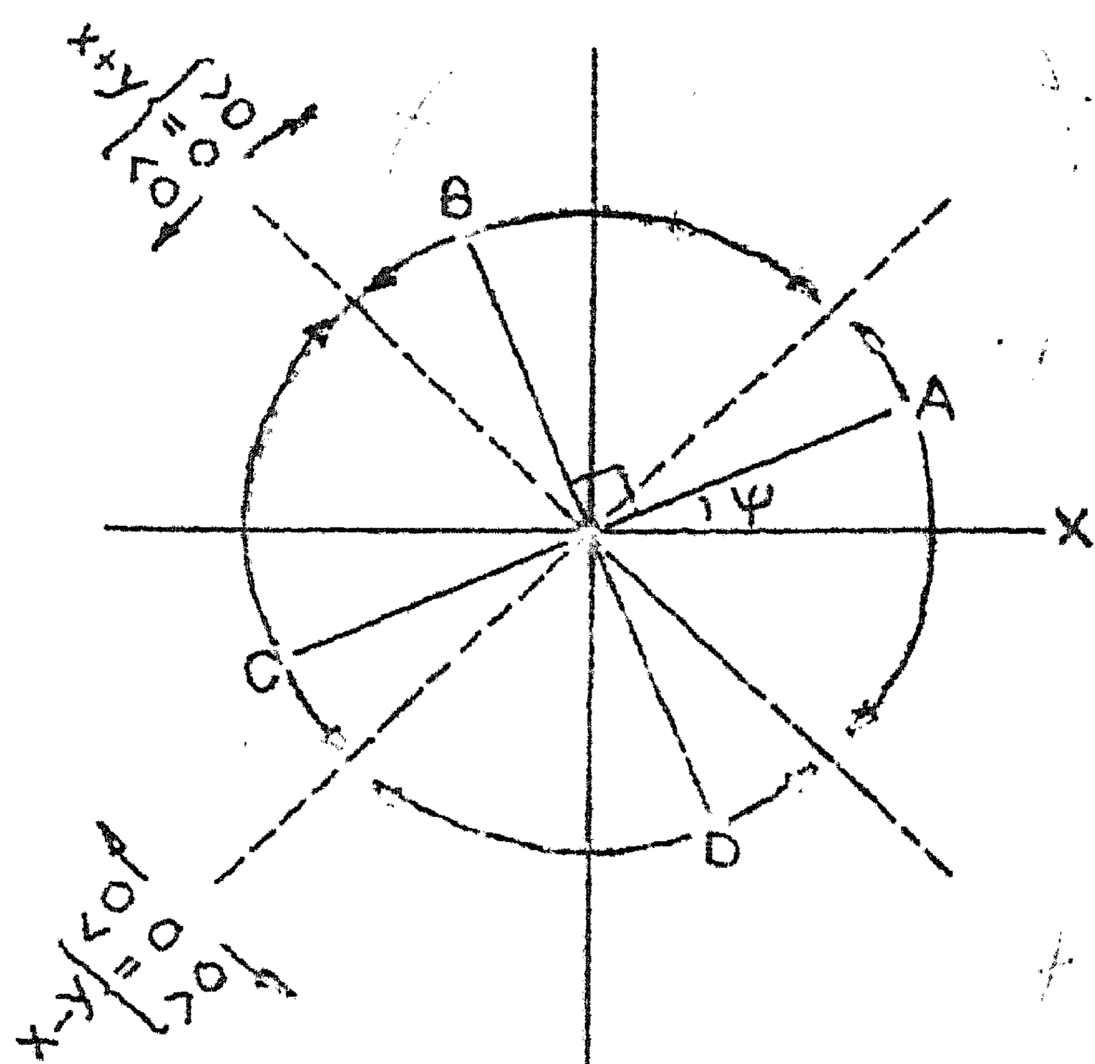


fig.3 Bij de herleiding tot $|\psi| \leq \frac{1}{4}\pi$

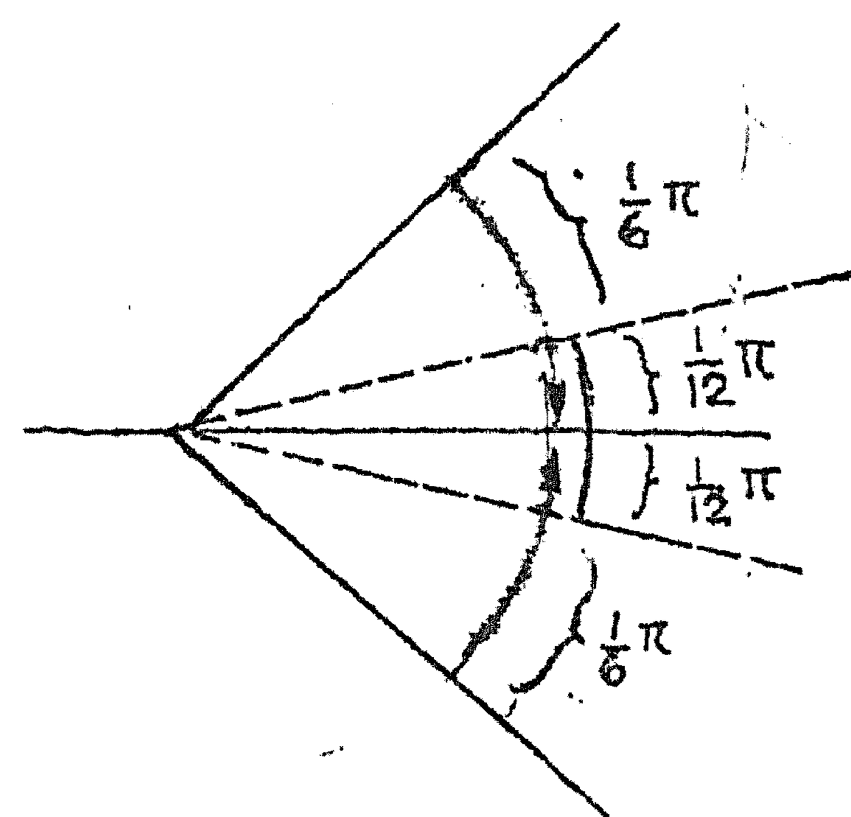


fig.5 Gebiedsverkleining voor de arctan z.

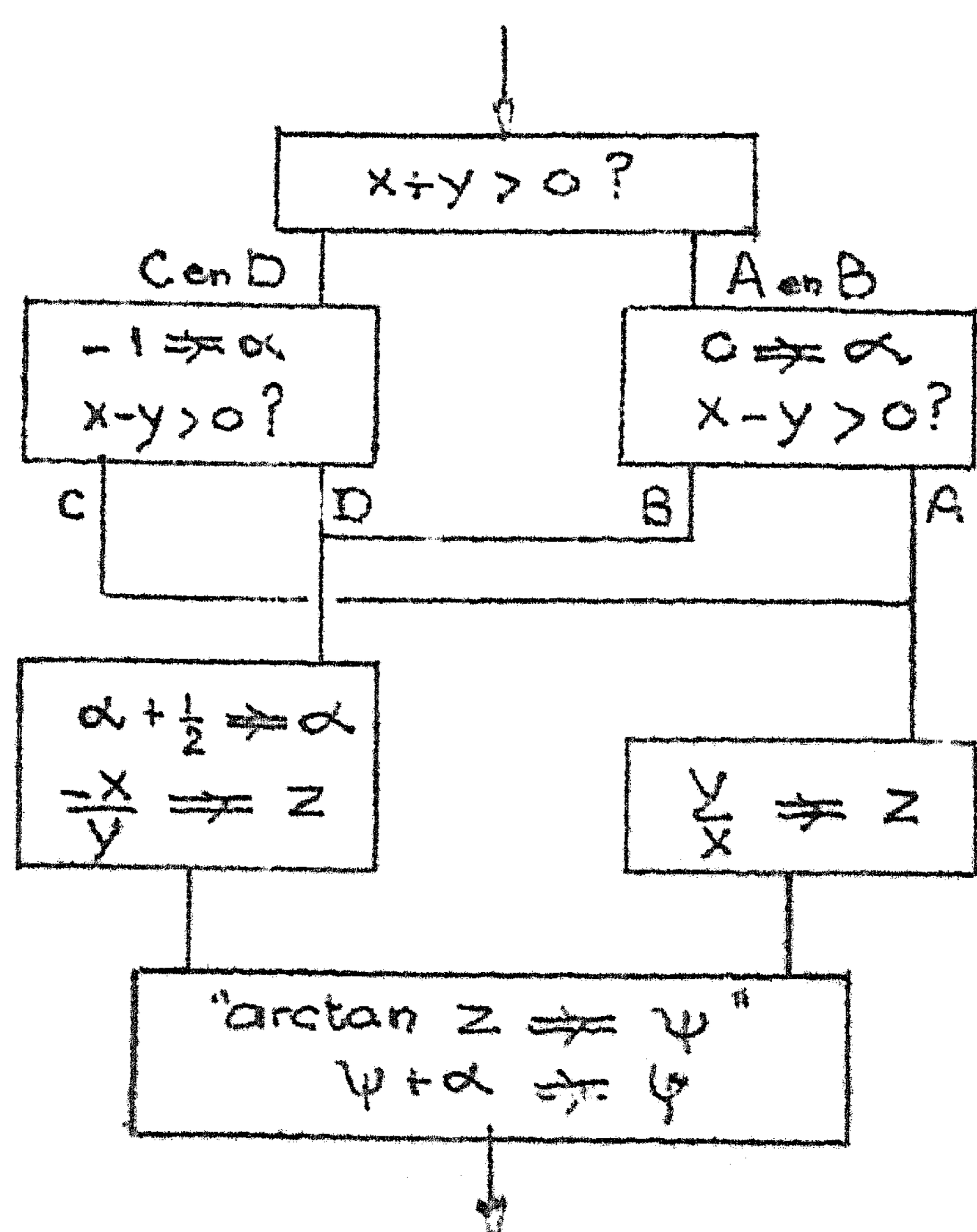


fig.4 Het "grove" blokschema: arctan $\frac{y}{x} \Rightarrow \varphi$

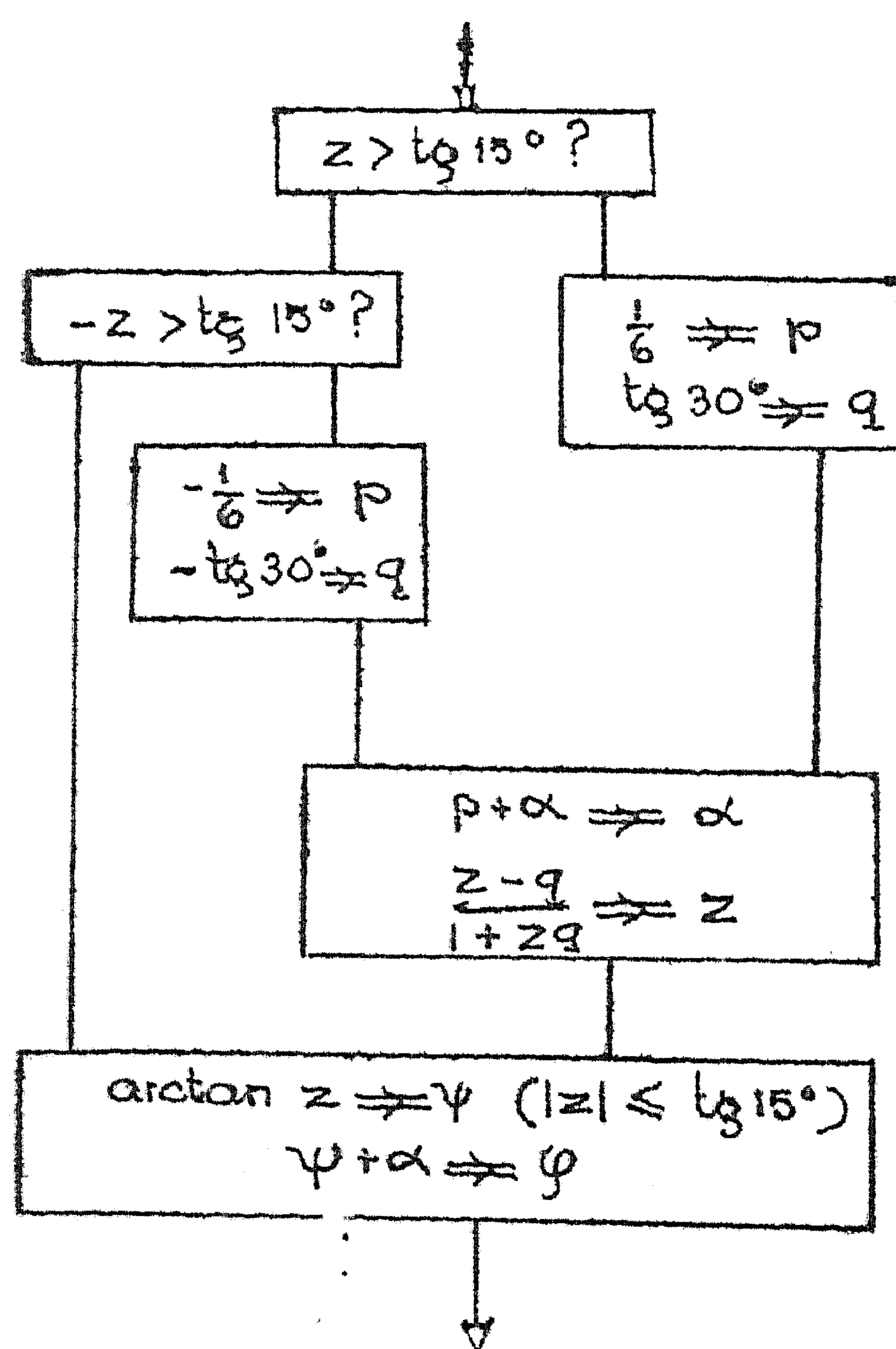


fig.6 De uitwerking van het laatste blok in fig.4 ; (zie fig.5)

(Opm.: Bij complexe getallen vinden wij exact dezelfde situatie terug, na de terminologieverandering:

- x - reeel deel
- y - imaginair deel
- r - modulus
- φ - argument

Ook de modulus is per definitie non-negatief; het argument is op een veelvoud van 2π na bepaald.)

Het ligt in een machine met vaste komma bovendien voor de hand om niet arctan z te vragen, maar arctan $\frac{y}{x}$ (y en x afzonderlijk gegeven); immers in de helft van de gevallen zou z de capaciteit overschrijden ($|z| \to \infty$ als $\varphi \to \pm\frac{1}{2}\pi$!) De berekening wordt uitgevoerd door slechts φ = arctan z voor kleine $|z|$ (dus $|\varphi|$ dicht bij 0) met een polynoombenadering van niet te hoge graad te berekenen en alle andere gevallen hierop te herleiden. Dit herleiden betekent:

1. bepalen, hoe uit x en y een geschikte z volgt,
2. tevens bepalen hoe uit φ = arctan z (klassieke def.) onze arctan $\frac{y}{x}$ (spec. def.) gevonden wordt.

Dit geschiedt in twee stappen: eerst wordt de berekening herleid tot die van een hoek tussen $-\frac{1}{4}\pi$ en $+\frac{1}{4}\pi$ (de bijbehorende tangens loopt van -1 tot +1). We maken gebruik van

$$\tan(\varphi \pm \pi) = \tan\varphi \quad \text{en} \quad \tan(\varphi \pm \tfrac{1}{2}\pi) = \frac{-1}{\tan\varphi}$$

In fig. 3 is de volledige 2π door de stippellijnen verdeeld in 4(gedraaide) quadranten. We beschouwen het over 90° draaibare kruis met de richtingen A,B,C en D; de hoek, die het "been A" met de positieve X-as maakt, noemen wij ψ ($|\psi| \leq \frac{1}{4}\pi$), $\tan\psi = z$. Bij een gegeven richting φ kan het kruis altijd zo gedraaid worden, dat precies één van de benen A,B,C of D met deze richting samen valt. (Het grensgeval, dat φ langs een van de stippellijnen valt, laten wij even buiten beschouwing). De situatie splitst zich dus in vier gevallen (we noteren van nu af aan hoeken in eenheden π ; optelling van hoeken geschiedt dus zonder extra voorzorg modulo 2π)

Geval A : $\varphi = \psi \to \tan\psi = \tan\varphi$ of $z = \frac{y}{x}$

Geval B . $\varphi = \psi + \frac{1}{2} \to \tan\psi = \frac{-1}{\tan\varphi}$ of $z = \frac{-x}{y}$

Geval C : $\varphi = \psi \pm 1 \to \tan\psi = \tan\varphi$ of $z = \frac{y}{x}$

Geval D : $\varphi = \psi - \frac{1}{2} \to \tan\psi = \frac{-1}{\tan\varphi}$ of $z = \frac{-x}{y}$

Welk van deze vier gevallen zich voordoet, onderzoekt men door te kijken, aan welke kant van de ene, en tevens aan welke kant van de andere stippellijn men zich bevindt. Omdat de punten op de stippellijnen door $x + y = 0$, resp. $x - y = 0$ gekarakteriseerd zijn, komt dit neer op het testen van de tekens van $x + y$ resp. $x - y$. Tot zover wordt de berekening in beeld gebracht door het blokschema, fig. 4.

Opm.1: Als de richting exact langs een stippellijn ligt, geldt bij het blokschema in fig. 4:

tussen A en B	als B	resumerend: "zodicht mogelijk bij de negatieve x-as."
tussen B en C	als C	
tussen C en D	als C	
tussen D en A	als D	

Opm.2: In ARRA moet [illegible] $- \frac{1}{2}\pi$ berekend worden, omdat $|z| = 1$ nog mogelijk is. In deze beschrijving blijven wij ons echter van z bedienen.

Opm.3: De testen op de tekens van $x + y$ en $x - y$ moeten voorzichtig uitgevoerd worden, in verband met mogelijke capaciteitsoverschrijdingen.

We hadden "het klokje rond" een arctangens gedefinieerd. In fig. 4 is de analyse gegeven, hoe deze samenhangt met de normale (klassieke) arctan z, en wel voor $|z| \leq 1$. Voor dit gebied wordt de gezochte ψ (in eenheden van π radiaal) gegeven door

$$\psi = \pi^{-1}\left(z - \frac{z^3}{3} + \frac{z^5}{5} - \frac{z^7}{7} + \cdots\cdots\right)$$

Voor de operatie "arctan $z \Rightarrow \psi$" is deze reeks niet zonder meer bruikbaar, omdat we voor grote z een onwaarschijnlijk groot aantal termen mee zouden moeten nemen, om de gewenste precisie te bereiken. Voor kleine z is de formule evenwel goed bruikbaar; dankzij de additieformule voor de tangens

$$\operatorname{tg}(\alpha + \beta) = \frac{\operatorname{tg}\alpha + \operatorname{tg}\beta}{1 - \operatorname{tg}\alpha \cdot \operatorname{tg}\beta}$$

kunnen we het gebied voor z, waarin we de polynoomberekening werkelijk toepassen bv. als volgt beperken (fig. 5).
De hoek van $-\frac{1}{4}\pi$ tot $+\frac{1}{4}\pi$ wordt verdeeld in drie gelijke stukken, elk dus van $\frac{1}{6}\pi$; ligt de nog uit te rekenen hoek (waar we de tangens van kennen) in het bovenste part, dan berekenen we de

tangens van de hoek, die $\frac{1}{6}\pi$ kleiner is: $|z|$ is dan klein, we berekenen arctan z met de machtreeks, en tellen bij de uitkomst weer $\frac{1}{6}\pi$ op; deze $\frac{1}{6}\pi$ wordt bij α, het toekomstig addendum opgeteld: de benadering voor arctan z hoeft uitsluitend in het middelste part toegepast te worden. Ligt de toekomstige hoek in het onderste part, dan wordt mutatis mutandis gelijkelijk gehandeld. In fig. 6 is het blokschema voor deze herleiding gegeven.

De e-macht

Uit herhaald toepassen van $a^2 - b^2 = (a + b)(a - b)$ vindt men

$$1-e^{-x} = (1+e^{\frac{-x}{2}})(1+e^{\frac{-x}{2^2}})(1+e^{\frac{-x}{2^3}})\cdots\cdots(1+e^{\frac{-x}{2^n}})(1-e^{\frac{-x}{2^n}})$$

of na inlassen van n factoren 2 en n factoren $\frac{1}{2}$:

$$1-e^{-x} = \frac{(1+e^{\frac{-x}{2}})}{2}\,\frac{(1+e^{\frac{-x}{2^2}})}{2}\,\frac{(1+e^{\frac{-x}{2^3}})}{2}\cdots\cdots\underbrace{\frac{(1+e^{\frac{-x}{2^n}})}{2}\,\underbrace{(1-e^{\frac{-x}{2^n}})2^n}_{u_n}}_{u_{n-1}}$$

Definieren wij $u_n = (1-e^{\frac{-x}{2^n}})2^n$, dan is gemakkelijk af te leiden, dat geldt:

$$u_{n-1} = u_n - \frac{u_n^2}{2^{n+1}}$$

Tevens volgt uit de wiskundige definitie van de e-macht, dat $\lim_{n=\infty} u_n = x$ In de precisie van de ARRA is deze limiet bereikt bij n = 28. Kiest men nu $u_{28} = x$ (als we niet afronden, is $u_{27} = x$ voldoende) en rekent men volgens bovenstaande recurrente betrekking u_{27}, dan u_{26} etc, uit, dan vindt men uiteindelijk

$$u_0 = 1-e^{-x} \quad \text{dus} \quad e^{-x} = 1-u_0$$

Dit procedé is in een binaire machine geschikter dan in een decimale. Het heeft het voordeel, dat de benodigde geheugenruimte klein is; het proces is echter door het grote aantal vermenigvuldigingen niet snel.

Een snellere methode vindt men door gebruikmaking van

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots\cdots \quad (n! = n(n-1)(n-2)\cdots\cdots 3.2.1)$$

Omdat de coëfficiënten in deze machtreeks al heel snel naar nul gaan, hoeft men van deze reeks niet te veel termen mee te nemen, om de benodigde precisie te bereiken; hoe groot dit aantal is, wordt bepaald door de maximale waarde van $|x|$. Het gebied voor x, waarin we een polynoombenadering toepassen, is weer te verkleinen, dankzij de analytische eigenschappen van de functie.

Omdat we van ons antwoord eisen, dat het binnen de capaciteit ligt, kan e^x alleen uitgerekend worden voor negatieve x, dus $-1 < x < 0$. Dit interval verdelen we in tweeen, van $-1 < x < -\frac{1}{2}$ en $-\frac{1}{2} \leq x < 0$. Als aan de laatste ongelijkheden voldaan is, berekenen we e^x met een polynoombenadering; als x in het eerste interval ligt, bedenken we, dat

$$e^x = e^{x+\frac{1}{2}-\frac{1}{2}} = e^{-\frac{1}{2}} \cdot e^{(x+\frac{1}{2})}$$

Nu ligt de exponent $(x + \frac{1}{2})$ in het tweede interval en we kunnen deze e-macht met de polynoombenadering berekenen. Na afloop vermenigvuldigen we met de constante $e^{-\frac{1}{2}}$. Omdat in de polynoombenadering nu vijf of zes termen meegenomen moeten worden, is dit programma aanzienlijk sneller. Behalve programma moeten nu ook constantes in het geheugen geborgen worden.

Opm.: De subroutine-bibliotheek voor binaire machines bevat vaak routines ter berekening van exponentiele functies met grondtal 2 inplaats van e; deze routines bevatten niets essentieel nieuws.

De logarithme met grondtal 2: $\log_2 x$

Als aan $\frac{1}{2} < x < 1$ beslist voldaan is, kan men $\log_2 x$ berekenen. Als $0 < x < 1$ geldt, moet $\log_2 x$ door een of andere factor gedeeld worden (2^5 bv.) om te zorgen, dat het antwoord binnen de capaciteit blijft. In dit geval begint men x zo vaak te verdubbelen, totdat het resultaat minimaal $\frac{1}{2}$ is, en trekt evenzoveel (geschaalde) gehelen van de logarithme af: dus

$$y = 2^n x \text{ waar } \frac{1}{2} \leq y < 1.$$

Kennelijk geldt dan $\log_2 x = -n + \log_2 y$. In het volgende beperken wij ons tot het berekenen van $\log_2 x$ als aan $\frac{1}{2} \leq x < 1$ voldaan is.

Eerste methode (binaal voor binaal).

Start met $x = x_0$ $0 = a_0$ en vervolg met de recurrente

$$\text{betrekkingen:} \begin{cases} \text{als } x_n \geq \frac{1}{2}: & x_n^2 = x_{n+1}; \quad a_n = a_{n+1} \\ \text{als } x_n < \frac{1}{2}: & (2x_n)^2 = x_{n+1}; \quad a_n - 2^{-n} = a_{n+1} \end{cases}$$

Als $\frac{1}{2} \leq x < 1$, dan is $\lim a_n = \log_2 x$.

Dit proces heeft weer het nadeel van lineaire convergentie; voor ARRA-precisie zou het 29 vermenigvuldigingen kosten. Het voordeel is de geringe geheugenruimte, die het programma inneemt.

Tweede methode (polynoombenadering met deling).

Wij weten dat $\log\frac{1}{x} = -\log x$; de logarithme is dus een functie, die van teken wisselt, als men x door de reciproke vervangt. Een eenvoudige functie met deze eigenschap is

$$y(x) = \frac{x-1}{x+1}.$$

Tevens heeft deze functie, evenals log x, een nulpunt voor x = 1. Wij mogen dus hopen op de mogelijkheid de logarithme in de buurt van x = 1 te benaderen door een polynoom in y van oneven graad. Deze hoop blijkt bij narekening gerechtvaardigd; met een polynoom van de 7^e of de 9^e graad kan de gebruikelijke precisie bereikt worden.

Opm.: Als boven gegeven loopt het gebied, waarin de polynoombenadering geldig is, van de grenzen $z \leqslant x \leqslant \frac{1}{z}$. Herleiding tot het gebied van $\frac{1}{2}$ tot 1 impliceert, dat men schrijft:

$$\tfrac{1}{2} + \log x = \text{oneven polynoom in } y\text{, met } \frac{\frac{1}{2}\sqrt{2} - x}{\frac{1}{2}\sqrt{2} + x} = y$$

Derde methode (polynoombenadering, zonder deling).

Men kan ook log x in een beperkt gebied door een polynoom in x benaderen; dit wordt dan van aanmerkelijk hogere graad. Een voor de hand liggende manier om deze graad te verlagen is verkleining van het interval, wat weer dankzij de analytische eigenschappen van de te benaderen functie mogelijk is: bv. in het gebied $\frac{1}{2}\sqrt{2} \leqslant x < 1$ past men een polynoombenadering voor $\log_2 x$ toe; als $\frac{1}{2} \leqslant x < \frac{1}{2}\sqrt{2}$, dan vermenigvuldigt men eerst x met $\sqrt{2}$ (x ligt dan in het gebied, waar we de polynoombenadering toe mogen passen); de bij dit argument berekende logarithme $\log_2 x$ vermindert men na afloop met $\frac{1}{2}$.

Syllabus No. 11 van de cursus 1955-'56:
Programmeren voor automatische rekenmachines

Onder leiding van
Prof. Dr. Ir. A. van Wijngaarden en de Heer E.W. Dijkstra

SNELHEID

In het voorgaande hebben wij een overzicht gegeven van de opdrachtencode en de uitbreiding ervan door middel van subroutines. Daarmede is echter nog lang niet alles gezegd wat van belang is voor de opmaak van een programma. Immers daarbij spelen allerlei factoren een rol, als bijv.:

de grootteorde van alle (!) getallen welke in de berekening optreden en het al of niet maatregelen nemen ("schalen") om al de getallen in absolute waarde kleiner dan de eenheid te maken;

kwesties betreffende de "service" die het programma biedt, bijv. of het programma gemakkelijk uitwendig in zijn loop beinvloed kan worden op grond van decisies die de gebruiker maakt naar aanleiding van getypte resultaten, met name of het gemakkelijk gestart kan worden of herstart kan worden nadat bijv. een fout is opgetreden en i.h.a. de flexibiliteit van het programma, d.w.z. de eigenschap dat het gebruikt kan worden onder een grote verscheidenheid van omstandigheden dan wel sterk gespecialiseerd is;

de programmering van controles op de berekening en het typen;

de snelheid;

de gemakzucht van de programmeur, enz.

Wij beginnen met de behandeling van de snelheid, enerzijds omdat deze a priori van groot belang is en voorts omdat bij de bespreking van de overige factoren er steeds rekening mee dient te worden gehouden. Het lijkt wellicht op het eerste gezicht vreemd, dat de programmeur anders dan door het weglaten van overbodige opdrachten de snelheid van het programma noemenswaard zou kunnen beinvloeden. Immers deze snelheid is in eerste instantie een gegeven eigenschap van de speciale machine waarmee men werkt. Toch is het niet de bedoeling, dat de programmeur de snelheid van zijn programma tracht op te voeren door constructiewijzigingen in de machine aan te brengen. De situatie is nl.deze, dat bij vele machines de tijd benodigd voor het uitvoeren van een opdracht juist alleen van de functie van die opdracht afhangt, maar ook van het verschil tussen het adres waarop de opdracht zich bevindt en het adres gespecificeerd in de opdracht. Door hier rekening mede te houden kan vaak een veel betere tijd gescored worder

dan die waarop men gemiddeld "recht heeft". Voorts kan vaak mirabele dictu de tijd drastisch beperkt worden door het aantal opdrachten drastisch maar op geschikte wijze te vergroten. Dit geldt zelfs voor bijna alle machines.

We zullen eerst de tijd benodigd voor het uitvoeren van een enkele opdracht eens bezien. Deze uitvoering valt meestal uiteen in vier processen, te weten:

1. Het halen van de opdracht uit het geheugen.
2. Het interpreteren van de opdracht
3. Het halen of wegbergen van een getal uit of in het geheugen;
4. Het uitvoeren van de door de opdracht gespecificeerde bewerking.

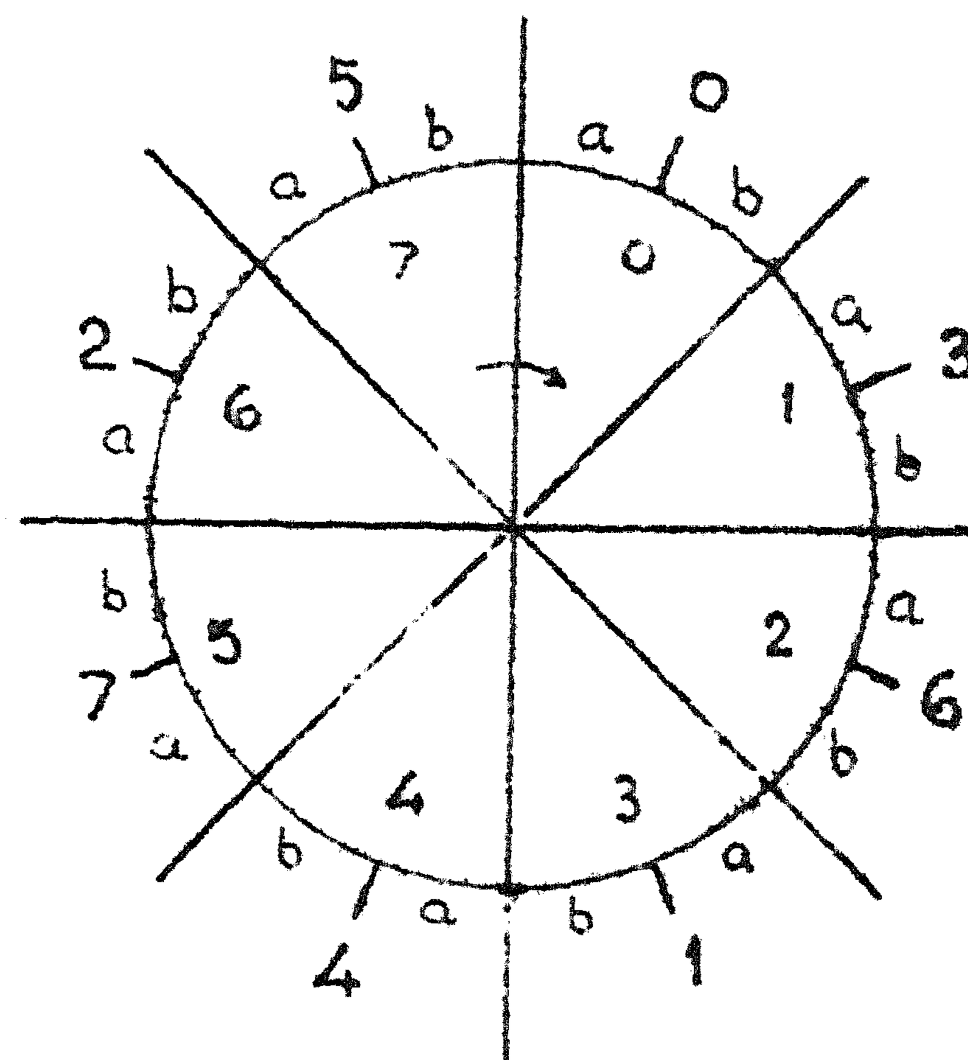
Op de tijd benodigd voor de punten 2 en 4 valt, gegeven de machine, niet af te dingen. Dit zijn tijden direct resulterend uit de intrinsieke snelheid van de machine. Wat betreft de tijd benodigd voor de punten 1 en 3 ligt de situatie echter geheel anders. Vele machines, bijv. ARRA en FERTA hebben een geheugen, waarin de verschillende adressen slechts op gezette tijden van een bepaalde tijdscyclus (trommelomwenteling) beschikbaar zijn. Wanneer nu geen bepaalde keuze uit adressen van tevoren is gemaakt zal zowel voor punt 1 als punt 3 gemiddeld een halve cyclus gewacht worden, d.w.z. gemiddeld totaal één cyclus. Bij de meeste opdrachten is de duur van de punten 2 en 4 zo kort met betrekking tot de cyclus, dat het uitvoeren van de opdracht dan gemiddeld ruim één cyclus kost, waarvan het grootste gedeelte van de tijd besteed is aan wachten op het geheugen. Veelal kan nu deze wachttijd sterk worden gereduceerd, wat dus een geweldige snelheidswinst ten gevolge heeft.

De mate waarin een dergelijk slim kiezen van adressen (optimum programmeren, minimum latency coding) mogelijk is varieert sterk voor verschillende machines. I.h.a. wordt deze mogelijkheid enorm vergroot door het beschikbaar zijn van een klein "snel" geheugen, d.w.z. een geheugen, dat op ieder ogenblik toegankelijk is voor de machine. Evenwel kan ook door andere organisatievormen heel wat in deze richting worden bereikt. Dit stelsel is toegepast op ARRA en FERTA. Wij zullen dit hierna behandelen.

De genoemde cyclus is hier een omwenteling van de magnetische trommel. Deze duurt 20 ms. Op de trommel staan 32 woorden met opeenvolgende adressen op een spoor of track.

Dit spoor is onderverdeeld in 8 getaltijden (2,5 ms),

genummerd 0,1,....7 en op iedere getaltijd bevinden zich 4 woorden met een kleine verschuiving door elkaar heengeschreven. Deze verschuiving is voor de programmeur van belang, want de machine compenseert haar, maar alleen kennelijk nodig, omdat men nu eenmaal niet twee cijfers op dezelfde plaats kan bergen. Deze vier woorden behoren bij adressen, die modulo 8 congruent zijn. Voorts staan de woorden behorend bij de adressen congruent a modulo 8 op de getaltijden congruent 3a modulo 8. Het tijdschema kan dus worden voorgesteld door nevenstaande horoscoop. Buiten de cirkel staan de adressen modulo 8 vermeld met a- en b-helften. Binnen de cirkel staan de getaltijden, voor de programmeur van minder belang.



Wij kunnen nu overgaan tot de beschrijving van de tijdsduur van de opdrachten. Allereerst nemen wij het geval, dat de opdracht op een a-plaats staat, een zg. a-opdracht. Voor het inlezen is alvast een halve getaltijd nodig. Is de opdracht een skipopdracht 24X0 of een niet gehoorzaamde conditionele sprong (6n of 14n met negatieve conditie), dan is de machine nog juist in staat om de volgende b-opdracht in te lezen. Deze opdrachten kosten ons "geen tijd", om precieser te zijn, zij kosten niet meer dan nu eenmaal nodig is om er kennis van te nemen. Iedere andere a-opdracht kost normaliter 1 omwenteling. Immers er moet nu iets gebeuren en de b-opdracht flitst ondertussen voorbij en kan pas de volgende omwenteling weer gelezen worden. In enkele gevallen duurt de a-opdracht nog langer. Bij de opdrachten 0,1,2,3,10, 11,12,13 kan het nl. gebeuren, dat het adres vermeld in de opdracht modulo 8 congruent is met het adres waarop de opdracht zelf zich bevindt. Dit getal komt dus pas na een omwenteling ter beschikking, maar terwijl het wordt uitgelezen en door de opteller heen loopt, flitst ook de b-opdracht weer voorbij en wij moeten dus weer een omwenteling wachten. Dit is een zg. ongunstig geval, waarin een opdracht, die slechts 1 omwenteling behoorde te kosten er 2 kost. Met de schrijfopdrachten 4,5,12 en 13 is natuurlijk hetzelfde aan de hand. Evenwel geldt hier bovendien, dat wanneer men 5 adrestijden (dus 7 getaltijden)

verder opschrijft een ongunstige situatie optreedt. Na schrijven wordt nl. de leesapparatuur gedurende 1 getaltijd geblokkeerd om de storingen veroorzaakt door de schrijfstroom te laten verdwijnen. Ook dan kan dus de volgende b-opdracht dus juist niet worden gelezen.

De vermenigvuldig- en deelopdrachten 16,17,18,19,20 en 21 behoeven voor hun eigenlijke werking precies 4 omwentelingen. Men gaat gemakkelijk na, dat als het een a-opdracht betreft, er dan totaal precies 5 omwentelingen voor nodig zijn ongeacht het adres, waarop het vermenigvuldigtal staat.

Voor de gehoorzaamde sprongopdrachten op de a-plaats (6 of 14 met positieve conditie en 7 of 15) geldt dat de eerste bruikbare plaats de a-opdracht 2 getaltijden verder op is. Vanaf 0a kan dus zonder omwenteling gesprongen worden naar 6a.

Bij de b-opdrachten is de situatie aanzienlijk gunstiger. Weliswaar is de machine niet instaat om een getal te lezen, dat zich op de onmiddelijk volgende getaltijd bevindt (bij de a-opdracht had de machine nog een halve getaltijd beschikbaar om zich voor te bereiden) maar anderzijds zijn er nu 2 getaltijden extra beschikbaar alvorens de volgende a-opdracht moet worden gelezen. Het resultaat is, dat alvast de genoemde ongunstige situaties zich meer voordoen. Daarvoor in de plaats, en hier schuilt de clou, treden er nu een aantal bijzonder gunstige situaties op.

Allereerst de schrijfopdrachten 4,5,12 en 13. Deze duren altijd precies 1 omwenteling. Hetzelfde geldt voor de schuifopdrachten 22n en 23n en de vermenigvuldigingen met tien, nl. 24/16 en 24/17. De opdrachten 0,1,2,3,8,9,10 en 11 kosten één omwenteling behalve als het in de opdracht vermelde adres modulo 8 congruent 6 is met het adres waarop de opdracht zich bevindt. Men ziet dan nl. in de horoscoop, dat de optelling (of zoiets) juist gepleegd wordt voor de volgende a-opdracht arriveert en dus behalve de altijd aanwezige draaiing, die nu eenmaal noodzakelijk is om van de opdrachten kennis te nemen geen tijd kost.

Bij de vermenigvuldigingen en delingen (16,17,18,19,20 en 21) ziet men op analoge wijze, dat er 2 gunstige posities voor het vermenigvuldigtal zijn, nl. congruent 1 of 6 modulo 8 met het adres van de opdracht. In deze gunstige gevallen kost de vermenigvuldiging of deling ons 4 omwentelingen i.p.v. 5.

De snelste sprong vanaf een b-plaats is naar de a-plaats 3 getaltijden verder (dus de "volgende opdracht"). Tenslotte

zijn er nog de adresloze communicatieopdrachten 24. Uitgezonderd reeds vermelde "maal tien" opdrachten 24 16 X0 en 24 17 X0 zijn deze op de b-plaats alle "snel", d.w.z. de volgende a-opdracht wordt zonder meer gehaald.

Om nu de tijdsduur van een gegeven programma zonder sprongen te berekenen gaat men het beste uit van een skipprogramma, d.w.z. een programma geheel bestaande uit skipopdrachten 24 X0. De "uitvoering" hiervan vergt wel degelijk tijd, nl. 3 omwentelingen per 8 opdrachtenkoppels. Dit is het minimum wat te bereiken valt. Vervolgens noteert men bij iedere opdracht of hij al of niet 1,2,4 of 5 omwentelingen langer kost en vindt zodoende de totale tijd.
Als voorbeeld kiezen wij de berekening van de tijd van de cyclus in het programma op pag. 4-5, welke telkens twee getallen bij

```
→ 301    1   361    *
         5   302    *
  302  ( 8     n )  *
       ( 8   n+1 )  *
  303    0   362    *
        24    40
  304    6   301    →
         .......
```

de inhoud van (S) optelt. Afgezien van de 5 extra omwentelingen, welke door een * zijn aangegeven kost het programma 2 omwentelingen. Totaal duurt de cyclus dus 7 omwentelingen of 140 ms. Iedere optelling kost dus 70 ms. Hierbij dient te worden opgemerkt, dat de a-opdracht op adres 302 één keer in de vier gevallen ongunstig is als n even is en de b-opdracht dan nimmer gunstig is, terwijl omgekeerd als n oneven is, de a-opdracht nimmer ongunstig is maar de b-opdracht eens in de vier keer gunstig is. Per optelling komt er dus gemiddeld nog 2,5 ms. bij dan wel gaat er 2,5 ms. af al naar gelang n even dan wel oneven is.

Door een kleine wijziging in het programma aan te brengen boeken wij altijd winst.

```
  301    ........
→        1   363
  302   24    X0
         5   303    *
  303  ( 8     n )  *
       ( 8   n+1 )  *
  304    0   362    *
        24 8  X1
  305   14   301    →
```

Immers het nevenstaande programma, waarin de constante op 361 verzet is naar 363 en waarin één skipopdracht 24 X0 is ingelast heeft een omwenteling minder nodig en vergt dus 60 ms. per optelling. Men zou kunnen menen, dat men nog een omwenteling zou kunnen besparen door op 304a nog een skip in te voeren en 362 te veranderen in 366. Evenwel komt dan de 24 8 X1 opdracht op 305a en kost een omwenteling. Men moet daar dan

dus ook een skip zetten maar dan komt de sprongopdracht
14 301 op 306a en deze sprong is net te kort en men verliest
de omwenteling dan dus daar.

Na enige oefening verkrijgt men al spoedig grote vaardigheid in het wegpraten van omwentelingen. Natuurlijk heeft het geen zin al te veel tijd eraan te besteden voor een eenmaal te gebruiken programma dat weinig machinetijd in beslag neemt, waarvoor subroutines en programma's die lang duren of veelvoudig gebruikt worden loont het vaak zeer de moeite.

Een vreselijk krachtdadige methode, die lang niet altijd toe te passen is, maar zo dat wel het geval is enorme tijdsbesparing op kan leveren, is het zg. strekken van programma's. Als wij het bovenstaande voorbeeld nog eens bezien, valt het op, dat wij voor de eigenlijke 2 optellingen in de cyclus ruim 2 omwentelingen gebruiken, terwijl de rompslomp (administratie, red tape) er 6 omwentelingen van maakt. Als de geheugenruimte zulks toelaat kunnen wij veel eenvoudiger de optelopdrachten stomweg achter elkaar neerschrijven. Wij krijgen dus bijv. het onderstaande programma.

```
300  10  500  * *
      8  501  *
301   8  502  *
      8  503  *
302   8  504  *
      8  505  *
303   8  506  *
      8  507  *
304   8  508  *
      8  509  *
305   8  510  *
      8  511  *
306   8  512  *
      8  513  *
307   8  514  *
      8  515  *
```

Wij zien dat per 8 optelkoppels 16 omwentelingen nodig zijn plus de 3 omwentelingen die zonder meer nodig zijn, d.w.z. 19 omwentelingen of 380 ms. per 16 optellingen of 24 ms. per optelling.

Wie het breed heeft laat het breed hangen. Als men toch geheugenruimte genoeg heeft, kan men het volgend programma gebruiken:

```
294  24   X0
     10  500
295  24   X0
     10  501
296  24   X0
     10  502
...  ...  ...
```

Nu is iedere opdracht snel. Dit programma vergt 3 omwentelingen dus 60 ms. per 8 optellingen, of wel 7,5 ms. per optelling. Een sneller optelprogramma is op de ARRA niet mogelijk, maar het zal niet vaak mogelijk en wenselijk zijn om zo drastisch te werk te gaan. Toch valt er nog iets zeer waardevols aan de laatste 2 programma's op te merken, nl. dat ze geen variabele opdrachten bevatten. Dit betekent, dat de kanalen, die dit programma bevatten van de getalselectie kunnen worden afgeschakeld, zodat dit gedeelte van het programma veiliggesteld kan worden tegen fouten.

Syllabus No. 12 van de cursus 1955-'56
Programmeren voor automatische rekenmachines
onder leiding van
Prof. Dr. Ir. A. van Wijngaarden en de Heer E.W. Dijkstra

EEN VOORBEELD TER ILLUSTRATIE

Wij laten nu een voorbeeld volgen van een practisch vraagstuk, waarbij snelheid een rol speelt. Zorgvuldig ontdaan van alle termen die het practisch nut evident zouden maken kan het aldus worden geformuleerd: Er zijn 24 punten (x,y,z), x = 0 of 1, y = 0 of 1, z = 0,1,2,3,4, of 5. Van deze 24 punten zijn er k roodgeverfd. Hiertoe behoort in elk geval de oorsprong (0,0,0). Aan elk punt wordt een gewicht toegekend, te weten een bescheiden positief getal (varierend van enkele eenheden tot enkele honderden). Hangt een gewicht in een rood punt, dan laat men het daar, maar hangt het in een ander punt, dan moet het in een rood punt worden gehangen. Voor dit verhangen geldt echter een spelregel, nl. dat geen van de coordinaten van het rode punt groter mag zijn dan de corresponderende coordinaten van het punt waar het gewicht oorspronkelijk hing, m.a.w. er mag niet verschoven worden in een richting waarin één of meer coordinaten toenemen. Voorts wordt door dit verhangen een verlies geleden. Ondergaat nl. een gewicht g_i verminderingen van zijn coordinaten ter grootte Δx_i, Δy_i, Δz_i (alle dus nonnegatief) dan lijdt dit gewicht per definitie een verlies

$$g_i(2\Delta x_i + \Delta y_i + 2\Delta z_i)$$

Zijn alle gewichten in rode punten gehangen dan is per definitie het (totale) verlies gelijk aan de som van de verliezen van alle gewichten

Nu kan men allereerst gemakkelijk nagaan hoe men elk gewicht moet verhangen om het verlies dat het lijdt minimaal te maken. Immers vanuit een bepaald punt kunnen slechts een aantal andere worden bereikt en deze kunnen worden gerangschikt in volgorde van toenemend of althans niet afnemend verlies. Gaat men bijv. uit van het punt (0,1,3), dan luidt deze rij, waarbij tussen haakjes vermeld wordt het verlies geleden door een gewicht ter grootte van de eenheid:

013(0), 003(1), 012(2), 002(3), 011(4), 001(5), 010(6), 000(7).

Voor het punt (1,1,1) luidt de rij:

111(0), 101(1), 011(2), 110(2), 001(3), 100(3), 010(4), 000(5).

Voor het punt (1,0,2) luidt zij:

102(0), 002(2), 101(2), 001(4), 100(4), 000(6).

Men kan gemakkelijk inzien, dat zulk een rij verliescijfers

hetzij gedurig met 1, dan wel gedurig met 2 opklimt. Men kan deze 24 reductierijen opstellen gemakshalve ineens met gewichten (die gedurende circa een half jaar constant blijven) vermenigvuldigd en nu in ieder rijtje het eerste rode punt opzoeken. Dan heeft men de goedkoopste reductie gevonden.

Tot zover is het vraagstuk zo simpel, dat het zeker niet de moeite loont het te programmeren. Nu volgt echter het eigenlijke vraagstuk: Gegeven k (het aantal rode punten), bepaal dan welk k-tal punten roodgeverfd moet worden opdat het minimum verlies, zoals boven bepaald, minimaal wordt. Voor k achtereenvolgens de waarden 1,2,3,....9 te kiezen.

Dit vraagstuk is plotseling vreselijk naar. Theorie is er practisch niet van te geven en de enige methode die kan worden toegepast is domweg de verschillende k-tallen te onderzoeken. Maar voor k = 9 bedraagt het aantal k-tallen (waarbij men bedenken moet, dat de oorsprong altijd aanwezig moet zijn) niet minder dan $\binom{23}{8} = 490314$. Dit getal is zo groot, dat het onbegonnen werk is het vraagstuk op deze manier te benaderen. Nu moet men dus wat water in de wijn doen en het vraagstuk zo reformuleren: Zoek een k-tal rood te verven punten, zodat het minimum verlies heel klein wordt. Om nu een redelijke beantwoording te geven moet men echter toch nog altijd wel heel veel combinaties onderzoeken. Aan de hand van de resultaten hiervan kan men nu gissen, welke veranderingen aangebracht moeten worden om tot betere resultaten te geraken. Zo kan men al zoekende en verschillende strategieën opstellende en proberende te werk gaan. Dit vereist een zo snel mogelijk en zo beinfluenceerbaar mogelijk programma. We zullen nu een aantal snapshots uit dit programma tonen, waarbij tevens duidelijk wordt, hoe men een dergelijk nauwelijks rekenkundig vraagstuk in machinetaal kan omzetten.

Om te beginnen willen we liever zo min mogelijk cijfers typen en daarom vervangen we de combinatie xy door één cijfer, dus 00 = 0, 01 = 1, 10 = 2, 11 = 3. Een punt wordt dan voorgesteld door 2 cijfers bijv. 25, te interpreteren als een getal in het 6-tallig stelsel geschreven en achteraf eventueel ook weer te interpreteren als punt nummer 17, nu geschreven in het tientallig stelsel. Met deze 24 punten zullen wij nu 24 adresplaatsen OBO, 1BO, 2BO,....23BO laten corresponderen en afspreken dat het punt rood geverfd is als zich op het corresponderende adres een negatief getal bevindt en dat het punt niet roodgeverfd is als dat getal positief is. Let wel voor het onderscheid van een binair kenmerk (rood, niet rood) is het teken van het getal voldoende en wij hoeven ons dus verder over het getal niet

uit te laten. Daar zullen wij later nog gebruik van kunnen maken.

Voorts moet de machine weten welk k-tal wij willen onderzoeken. Dit kan natuurlijk via de ponsband geschieden, maar dit is hopeloos inflexibel en traag in vergelijking met het gebruik van de getalschakelaars. Hier hebben wij immers 30 binaire elementen aanwezig door de machine gemakkelijk te lezen en heel eenvoudig in te stellen door de mens. Wij bestemmen dus de minst significante 24 schakelaars om de punten 00, 01,....35 te representeren. Staat de schakelaar omhoog (1) dan is het punt rood, zo niet dan is het punt niet rood. Aangezien voorts het punt 00 altijd rood is, laten wij dit punt verder buiten beschouwing, daar het programma hiervoor kan zorgen.

Het is bij de meeste strategiekeuzen geschikt om slechts k-1 punten rood te verven en dan het effect te zien van het roodverven van achtereenvolgens steeds een ander punt van de overgebleven punten. Daarom willen wij eerst getypt zien welke k-2 schakelaars omhoog staan en daarachter het corresponderende verlies en vervolgens daaronder een aantal regels telkens vermeldend een toegevoegd rood punt en het corresponderend verlies.

```
0A0  24       X0
     24   6   X0
  1   7  19   X2 =) TWNR
     24   6   X0
  2   7  19   X2 =) TWNR
     24   7   X0 [G]⇒(A)
  3  22  23   X0 2^6(A)⇒(S)
     12   1   B0 1B0 even werk-
..............      ruimte
  4   8   1   B0
     12   1   B0 (1B0)< 0 als
                 schakelaar 01
  5  14   8   A0→omhoog stond
      2       D0 (0D0) = 0 ⇒(A)
  6  24   8   X0 Typt 0
     24   1  X16 (A)+1=1⇒(A)
  7  24   8   X0 Typt 1
     24  14  X16 (A)+14=15⇒(A)
  8  24   8   X0 Typt spatie
.................
      8   1   B0
  9  12   2   B0 (2B0)< 0 als
                 schakelaar 02
      6  13   A0→omhoog stond
 10   2       D0
     24   8   X0 Typt 0
 11  24   2  X16
     24   8   X0 Typt 2
 12  24  13  X16
     24   8   X0 Typt spatie
```

Het begin van het programma op 0A0 voert 2 maal het papier op en gaat nu de schakelaars onderzoeken en daarvan rekenschap afleggen. Het leest die inhoud in A en schuift het 23 plaatsen naar S, het cijfer corresponderend met de schakelaar 01 direct achter het tekencijfer van S staat, Het verdubbelt nu de inhoud van S, even 1B0 als werkruimte gebruikend, en schrijft deze nieuwe inhoud nu definitief op 1B0. Als 01 een rood punt was is deze inhoud negatief en wordt de sprong op 5aA0 niet gehoorzaamd en wordt getypt 01 gevolgd door een spatie en het programma arriveert op 8bA0. Was 01 geen rood punt dan is de inhoud van 1B0 positief en dan werd de sprong op 5aA0 wel gehoorzaamd. Er werd dan niet getypt en het programma arriveerde dan ook op 8bA0. Hier wordt (S) weer verdubbeld, nu weggeschreven op 2B0 en zo wordt de kleur van het punt 02 vastgelegd, enz. Let op, dat alleen het teken van het weggeschreven getal betekenis heeft! Dit geschiedt zo 23 maal. Natuurlijk wordt het grootste gedeelte van het programma, nl. de 6 opdrachten voor het typen, in het merendeel van de gevallen overgeslagen en meestal dus alleen de 3 opdrachten benodigd voor de definitie van de rode punten uitgevoerd. Dit is klaar na de behandeling van punt 35 (of 23 in tientallige schrijfwijze) op adres 11aA3. Het programma gaat dan als volgt verder op 11bA3:

```
11A3  24   8  X0
      ..........
      24   6  X0
  12   7      E0 =)
      ..........
       3   1  B0
  13  24   8  X1
       6  18  A3 →
  14   4   1  B0
       2      D0
  15  24   8  X0  Typt 0
      24   1  X16
  16  24   8  X0  Typt 1
      24   6  X0
  17   7      E0 =)
       4   1  B0
     ...........
  18   3   2  B0
      24   8  X1
  19  14  23  A3 →
       4   2  B0
          enz.
```

Nu wordt een subroutine, die begint op OEO aangeroepen. Deze rekent bij een gegeven situatie het verlies uit en typt dit als geheel getal van 4 cijfers voorafgegaan door een spatie en geeft vervolgens een TWNR. Hier komen wij nog op terug. Op 12bA3 wordt nu nagegaan of het punt 01 rood was of niet. Zo ja, dan wordt gesprongen naar 18aA3 waar het punt Q2 onderzocht wordt. Is 01 niet rood, dan wordt het tijdelijk op 14aA3 rood geverfd. Dan wordt 01 getypt, zonder spatie ditmaal en de subroutine wordt weer aangeroepen. Deze beoordeelt de situatie weer, maar nu met het rode punt 01 erbij. De spatie tussen 01 en het erachter getypte verlies komt door de typconstanten van zelf in orde en na de TWNR komt de subroutine terug op 17bA3. De TWNR is echter tot stand gekomen via de normale aanroep 24 6 X0, 7 19 X2 en deze TWNR-routine komt terug met [A] = -15, als resultaat van de telling voor de tabulatievertraging. Dit is nu eenmaal zo en de opdracht op 17bA3 wast de rode verf dus van het punt 01 af! Merk weer op, dat weer alleen het teken ons interesseert. Dat de inhoud van 01 nu precies +15 bedraagt is van geen belang.

Op precies dezelfde wijze werden de volgende punten afgewerkt, d.w.z. overgeslagen als ze rood zijn en tijdelijk even roodgeverfd om het effect van toevoeging ervan te tonen. Tenslotte wensen we er zeker van te zijn, dat we niet voor niets zitten te rekenen doordat de constanten van het programma bedorven zijn. Daartoe telt uiteindelijk een volkomen optimaal programma deze 72 constanten en hun negatieve som op, controleert of het resultaat -0 is en stopt na een punt getypt te hebben zo het klopte en zonder punt getypt te hebben als het niet klopte. Deze controle kost slechts 600 ms.

```
17E1   3   9  BO  * *
      24   8  X1
  18   6  31  E1 →
       8   8  C1
   . . . . . . . . . .
  19   3   3  BO  * *
      24   8  X1
  20   6  31  E1 →
       8  10  C1
   . . . . . . . . . .
  21   3   8  BO  *
      24   8  X1
  22   6  31  E1 →
       8  12  C1
   . . . . . . . . . .
  23   3   2  BO  *
      24   8  X1
  24   6  31  E1 →
       8  14  C1
   . . . . . . . . . .
  25   3   7  BO  *
      24   8  X1
  26   6  31  E1 →
       8   8  C1
   . . . . . . . . . .
  27   3   1  BO  *
      24   8  X1
  28   6  31  E1 →
       8  10  C1
   . . . . . . . . . .
  29   3   6  BO  *
      24   8  X1
  30   6  31  E1 →
       8  12  C1
```

We moeten nu nog even de subroutines bezien. Deze wordt circa 20 maal doorlopen per programma en bepaalt bijna de gehele tijdsduur van het programma. Dus loont het hier zeer de moeite op tijd te letten.
De routine begint met S schoon te maken en dan voor ieder punt op de geschetste wijze het verlies te sommeren in S. Als voorbeeld geven wij het gedeelte, dat het punt 1,3 (decimaal 9) behandelt. Het gewicht van dat punt is gegeven op 8C1, 10C1, 12C1 en 14C1. Telkens worden vier opdrachten uitgevoerd, de eerste kost 1 of een enkele maal 2 omwentelingen, de overige 3 zijn alle kort (zo de sprong genegeerd wordt). Gemiddeld kosten zij 10 ms per opdracht.
Het resultaat van e.e.a. is een extreem programma. Normalerwijze geprogrammeerd zou het vele malen korter in plaatsruimte en vele malen langer in tijdsduur geweest zijn. Dit laatste zou fnuikend zijn voor de toepassing.

Syllabus No 13 van de cursus 1955-'56:
Programmeren voor automatische rekenmachines
Onder leiding van
Prof. Dr. Ir. A. van Wijngaarden en de Heer E.W. Dijkstra

CONTROLE EN FLEXIBILITEIT

Onder de geprogrammeerde controle vallen alle maatregelen, die de programmeur treft, om er zo goed als zeker van te kunnen zijn, dat de resultaten, die door de machine afgeleverd worden, overeenkomen met de bedoelde resultaten.

De noodzaak van dergelijke maatregelen is evident: een machine, die een hoeveelheid cijfers produceert, maar zonder enige garantie, dat deze cijfers inderdaad iets voorstellen, is een vrij onnut apparaat. Het antwoord ontleent toch immers zijn waarde aan de garantie, dat het het goede antwoord is.

Het is kennelijk inefficiënt, zo niet ondoenlijk, om de resultaten van de machine met de hand na te rekenen. Ter bestrijding van fouten in dusdanige hoeveelheden rekenwerk moet men machtiger wapens ter hand nemen en het enige wapen, hier tegen opgewassen, is....de machine zelf: men bant Satan met Beëlzebub uit, met alle gevolgen van dien.

Het programma, dat zijn eigen verrichtingen controleert moet iets extra doen; het is dus langer, zowel in geheugenruimte als in tijdsduur. De machine berekent in het controlerende programma meer dan het hoognodige:als regel rekent hij af en toe een controlegrootheid uit, die als alles goed is gegaan = 0 is; of aan deze gelijkheid voldaan is, wordt dan met behulp van de opdracht 24 8 X3 getest. Wordt door deze opdracht de conditie positief gezet, dan is een fout gedetecteerd, is de conditie negatief, dan nemen we aan, dat de berekening feilloos is uitgevoerd. Absolute zekerheid hebben we echter niet! Het is b.v. denkbaar, dat de machine meer dan één fout gemaakt heeft, maar dat deze fouten elkaar juist zò compenseren, dat het controleresultaat toch weer = 0 is. Bij een (goede) controle is de kans hierop echter verwaarloosbaar klein. Minder ver gezocht is bv. het geval, dat de opdracht 24 8 X3 niet juist functionneert, b.v. steeds de conditie negatief zet, ongeacht de inhoud van A. Vandaar, dat wij in de eerste alinea schreven "zo goed als zeker".

Voor de keuze van het controleresultaat is geen vaste regel te geven, deze hangt uiteraard te veel van de aard van de berekening af. Wij moeten ons tot enkele opmerkingen beperken.

Een denkbare methode van controle is de berekening twee maal uitvoeren en de beide resultaten van elkaar aftrekken.

Deze methode is om twee redenen verwerpelijk, ten eerste, omdat zo de rekentijd minstens verdubbeld wordt (terwijl het vaak veel zuiniger kan), ten tweede (en dat is doorslaggevend), omdat hier de kans op elkaar compenserende fouten niet verwaarloosbaar klein moet worden geacht: de machine hoeft nu slechts tweemaal dezelfde fout te maken.

Voorts dient het controleresultaat zo van alle grootheden, waarover deze controle een garantie hoopt uit te spreken, af te hangen, dat een afwijking in een willekeurige dezer grootheden het controleresultaat beduidend van nul doet afwijken; anders is de controle voor zo'n grootheid niet "gevoelig genoeg". Mathematisch betekent dit, dat de partiele afgeleiden van het controleresultaat naar elk der te controleren grootheden in absolute waarde van de orde van grootte 1 moeten zijn. Dit is de reden, waarom we de berekening van s = sinx en c = cosx niet mogen controleren door te onderzoeken of $f=s^2+c^2-1$ wel voldoende = 0 is. Een der partiele afgeleiden

$$\frac{\partial f}{\partial s} = 2s \quad \text{en} \quad \frac{\partial f}{\partial c} = 2c$$

is veel te klein, als de hoek in de buurt van een veelvoud van $\frac{1}{2}\pi$ ligt. Daarentegen voldoet iedere somcontrole wel aan deze eis.

Wat tot zover gezegd is, geldt algemeen voor elke machine De specifieke machine, waarvoor het programma bestemd is, bepaalt echter (o.a.) "wat en waar" men controleert. In ARRA is geen enkele controle ingebouwd, in tegenstelling tot ARMAC, waar bij het lezen uit het geheugen steeds een "parity-check" wordt uitgevoerd. De wijze, waarop registers en adressen hun informatie prijsgeven, dus gelezen worden, drukt ook een stempel op de controle. Uit sommige geheugens wordt "niet destructief" gelezen (magnetische trommel), uit andere kan men alleen maar "destructief" lezen (matrixgeheugen, de A- en S-registers) d.w.z. wordt door de leesoperatie het betroffen stuk van het geheugen in eerste instantie schoongemaakt, maar wat wordt afgeleverd, wordt onmiddellijk teruggeschreven. In een destructief geheugen kan men alleen maar garanderen, dat een bepaalde grootheid er op het moment van de controle goed stond, niet dat hij er na de controle nog goed staat.

In ARRA wordt de controle daarom altijd uitgeoefend op getallen in het (niet destructieve) geheugen. (Voor ARMAC is de situatie nog niet te overzien.) De vaak gehoorde opmerking, dat in een machine, waarvan ook het rekenorgaan (in duplo uit-

gevoerd) gecontroleerd is, geprogrammeerde controle volledig overbodig is, is slechts ten dele juist. Via de controle ondervangt men n.l. niet alleen fouten in de werking van de machine, maar ook fouten in het programma, zowel "blunders" (foute coderingen, onjuiste ponsingen) als subtielere fouten, als "verlies van cijfers" d.w.z. onvoldoende precisie.

Een derde facet van de controle wordt bepaald door de faciliteiten, die men eist in het geval, dat de machine een fout maakt. Man kan eisen, dat na detectie van een fout de machine automatisch een stukje terug opnieuw begint en de mislukte "moot" opnieuw probeert (zg. controle in vivo); een lichtere eis stelt men, als men tevreden is met de zekerheid, dat het afgeleverde antwoord goed is, maar aan de operateur overlaat, welke stappen genomen moeten worden. als de controle faalt (z.g. controle post mortem).

Een voorbeeld moge het verschil tussen deze methodes illustreren. Gevraagd een rij getallen, die in het geheugen staat, met een getal L te vermenigvuldigen en de antwoordenrij op dezelfde plaats in het geheugen achter te laten. Men kiest in beide gevallen een somcontrole en begint met de negatieve som achter de rij te plaatsen. Controleert men post mortem, dan vervangt men elk getal in de aangevulde rij door zijn L-voud en controleert of de som van de elementen der (aangevulde) rij weer (binnen de tolerantie der afrondingen) = 0 is. Als aan deze gelijkheid niet voldoende voldaan is, kan dit stukje berekening niet automatisch worden overgedaan, want de oorspronkelijke getallen zijn niet meer beschikbaar! Controleert men in vivo, dan moeten in elke moot de <u>gebruikte</u> grootheden tot aan het einde van de moot beschikbaar blijven. Een oplossing hiervoor is het splitsen in twee moten: in de eerste moot plaatst men het L-voud op een rijtje hulpwerkruimtes en controleert deze operatie, zo goed; dan gaat de besturing over naar de tweede moot, waar de L-vouden naar de oorspronkelijke plaatsen getransporteerd worden en deze transporten worden gecontroleerd.

Al met al zal men de controle in vivo vanwege de omslachtigheid vermijden, waar mogelijk. In rekenprocessen met een "ketting-structuur", waar het afgeleverde resultaat steeds weer de uitgangsgrootheden voor de volgende phase zijn (b.v. grote repetities, eigenwaarden en reductie van matrices), kan men er moeilijk onderuit, kortom bij die berekeningen, waar het alleen maar zin heeft om door te gaan, als de resultaten tot zover gegarandeerd goed zijn. Als de eisen van de controle in

vivo te zwaar zijn, is de volgende werkwijze te overwegen, bedenkend, dat het verschil tussen post mortem en in vivo controleren ligt in de mogelijkheid een eindje terug opnieuw te beginnen. Men kan op gezette tijden (b.v. ieder uur) de vitale informatie uitponsen, die nodig is om vanaf het dan bereikte stadium door te rekenen en verder het programma post mortem controleren. Als de controle een fout detecteert, leest men de laatst geponste band in en begint van daar opnieuw te rekenen. De operateur heeft hier wel de plicht om alle banden, die de machine produceert zorgvuldig te controleren, te beschrijven en op te bergen, dit ter vermijding van de complete chaos, die maar al te licht ontstaat. Deze methode is des te voordeliger, naarmate de staat van de machine beter is; als de machine heel goed is, kan men b.v. slechts om de twee uur uitponsen.

Waar het om gaat een groot aantal korte, onafhankelijke berekeningen, zal men altijd post mortem controleren. Een eenvoudig voorbeeld moge dit illustreren: gevraagd een getal in priemfactoren te ontbinden. Men doet dit, door het getal door de successievelijke priemgetallen (2,3,5,7,11,....) te delen; als een rest $\neq 0$ gevonden wordt, gaat men het volgende priemgetal proberen, als een rest = 0 gevonden wordt, typt men de betroffen deler en opereert verder op het quotient, te beginnen met de zojuist uitgedeelde priemfactor, die er nog een keer in kan zitten. Men eindigt, òf als er bij opgaande deling een quotient = 1 vindt òf als bij niet opgaande deling het quotient niet groter is dan de deler: in dit geval is het overgebleven getal de grootste (laatste) priemfactor. Deze berekening kan men controleren, door

a) de gevonden factoren met elkaar te vermenigvuldigen en het product met het oorspronkelijke getal te vergelijken.

b) als de factorisatie op de tweede manier is geeindigd, tevens te onderzoeken of de "overgebleven" factor inderdaad priem is.

(Alleen als het oorspronkelijke getal ondeelbaar is, doet de machine dubbel werk.)

<u>Opm.:</u> a controleert of we niet "te veel", b controleert of we niet "te weinig" factoren hebben gevonden.

Er zijn zelfs gevallen. dat wij de geprogrammeerde controle geheel achterwege laten, b.v. als wij vele punten op een lijn uit moeten rekenen, de berekening van elk punt een zelf-

standige berekening is (d.w.z. onafhankelijk van de uitkomsten van buur-punten) en bovendien per lijn een paar punten bekend zijn. Als een enkel punt bij uitzetten buiten de lijn valt, is de berekening kennelijk hier fout. Als de bekende punten netjes op de lijn liggen, kan dit een voldoende garantie zijn, dat er geen systematische fout in alle punten zit. Het programma is dan uiteraard zo uitgevoerd, dat na de berekening van het ene punt de machine zonder menselijk ingrijpen het volgende punt gaat uitrekenen; als men de machine normaal start zal hij aan het eerste punt van een lijn beginnen. Wil men later één of meer punten liever nog eens overgerekend zien, dan is het wel prettig, als de programmeur bij het opstellen van het programma met dit verlangen rekening heeft gehouden, m.a.w. het programma moet gemakkelijk op een bepaald punt te starten zijn. Men zal in zo'n geval een tweede startadres specificeren: het startadres voor het meegegeven beginpunt (men brengt dan een of andere parameter in S of plaatst hem in de getalschakelaars.)

Hier hebben wij een voorbeeld van een "speciale start". Dergelijke faciliteiten zijn des te belangrijker, naarmate de operateur een grotere rol is toegedacht. Een algemene eis, waar met enige zorg makkelijk aan te voldoen is, is de z.g. herstartbaarheid. Hieronder wordt verstaan, dat het mogelijk moet zijn, om een programma,"dat al even gedraaid heeft", te stoppen - b.v. omdat het papier scheeft in de schrijfmachine bleek te zitten - en de hele berekening opnieuw te beginnen, door alleen maar op de startknop te drukken, en niet eerst weer alle banden in te lezen. Dit impliceert, dat met de banden alleen die informatie wordt ingevuld, die door het programma niet wordt gewijzigd. Een voorbeeld moge dit illustreren: als in een bepaald punt iets steeds negen keer moet worden overgeslagen, edoch elke tiende keer wel moet gebeuren, dan kan dit geschieden door 1 aftrekken van een telgrootheid: men begint met deze grootheid = 10 te stellen: zolang de aftrekking met 1 een positief resultaat aflevert, slaat men het betroffen stukje over, zodra de aftrekking een antwoord = 0 aflevert, doet men het betroffen stukje wel en zet de telgrootheid weer = 10 voor de volgende telling (men "herstelt de telling".) Als men op het adres van de telgrootheid tijdens de invoer van het probleem ("van de band") tien invult, en het probleem start, gaat alles goed, maar het programma is niet herstartbaar: als men de machine op een

willekeurig moment stopt, zal het adres van de telgrootheid over het algemeen geen tien bevatten! Het is een kleine moeite om bij de eerste opdrachten na de start een paar opdrachten in te lassen, die op de plaats van deze telgrootheid 10 invullen: het initiale "zetten" van de telling dient door het programma te gebeuren. Hetzelfde geldt voor het = 0 maken van een adres, waarin men een som gaat opbouwen, etc. Zich aan te wennen ieder programma aan de eis van de herstartbaarheid te laten voldoen, is ook om een andere reden een gezonde discipline: het herstartbare stuk programma is een zelfstandig geheel, dat geen extra-administratieve voorzorgen behoeft. (Wil men b.v. in het geval van de sommatie meer sommatie's uitvoeren, - met andere parameterwaarden b.v. - dan hoeft het programma "van hoger orde" dat steeds de machine in deze sommatie's lanceert, niet alle volgende keren het opbouwadres schoon te maken.) Dit laatste argument is misschien nog sterker dan het eerste, omdat bij een werkelijk snelle machine het inlezen van het programma nu niet zo erg veel werk is. Bij een heel snelle machine vervalt nl. misschien ook het argument, dat herstartbaarheid een eis is voor het "bij de machine opsporen" van fouten in een programma: als dan een programma fout blijkt te zijn, kon men wel eens overgaan tot uittikken van de geheugeninhoud en de programmeur met deze papieren naar zijn kamer terugsturen, om daar op zijn gemak te laten reconstrueren, waar het programma gederailleerd is. Het herstartbaar maken is niet anders dan bij elkaar behorende "voorbereiding en modificatie" inderdaad aan elkaar te verbinden. Voorbereiding en modificatie bakenen zich _tezamen_ af als zelfstandig onderdeel; in het programma aan deze saamhorigheid gehoor geven komt de overzichtelijkheid alleen maar ten goede en maakt het mogelijk om zonder al te veel hoofdbrekens dit stuk op een andere (logische) plaats in het programma te laten functionneren als aanvankelijk voorzien en bedoeld was. En hier zijn we beland bij een tweede aspect der flexibiliteit.

Werd tot nu toe de flexibiliteit beschouwd als eis, waaraan we verplicht waren te voldoen door onvolkomenheid van machine en programmeur, in werkelijkheid is de situatie nog gecompliceerder. Het in detail behandelde voorbeeld (de splitsing van getallen in de som van twee quadraten) was een idealisatie: we wisten van begin af aan het juiste formularium en wat we als antwoord wensten. Aan beide condities is vaak niet

voldaan: de uitkomst van de proefberekeningen is veelal dat men foute formules heeft opgekregen. Als deze fouten achterhaald zijn - wat wijzigingen in het programma impliceert - gebeurt het vaak, dat aan de hand van de uitgetypte resultaten (of op eigen gelegenheid) de opdrachtgever tot de ontdekking komt, dat hij dat en dat ook graag uitgetypt zag. Als het aanvankelijke programma zo flexibel is, dat de gewenste wijzigingen kunnen worden aangebracht, zonder het gehele programma om te gooien, bespaart men zich veel werk, tijd en ergernis.

Syllabus No. 14 van de cursus 1955-'56:
Programmeren voor automatische rekenmachines
Onder leiding van
Prof. Dr. Ir. A. van Wijngaarden en de Heer E.W. Dijkstra

DE ADMINISTRATIEVE SUBROUTINE

De tot nu toe behandelde subroutines dienden meestal ter berekening van speciale functies als cos x, log x of om een andere operatie op een getal toe te passen, dus bijv. typ x op een of andere manier. Het getal x werd in S meegegeven aan de subroutine. Het hoofdprogramma roept de subroutine aan, deze doet haar werk zonder andere subroutines nodig te hebben en keert terug.

Nu komen wij tot subroutines welker taak coördinerend is. Zij roepen op hun beurt weer andere subroutines aan, of zelfs delen van het hoofdprogramma. De algemeenste routines van deze klasse zijn zelfs superieur aan het eigenlijke hoofdprogramma; het hoofdprogramma wordt dan gedegradeerd tot een stel subroutines ten dienste van de administratieve subroutines. De reden, waarom deze dan nog als subroutine zijn uitgevoerd is om nog algemenere coördinerende routines er mee te laten werken. Uiteindelijk is het ideaal een "hoofdprogramma" als volgt in ARRA-taal:

```
24   6   X0
 7       F0 =) Ga naar hoogstgeordende administratieve
               subroutine
24   2   X0
24   3   X0    Stop
```

De allermachtigste van deze routines hebben derhalve slechts één aanroep. De minder machtige daarentegen zijn met het oog op de flexibiliteit meestal gekenmerkt door een groot aantal aanroepmogelijkheden.

We zullen enige voorbeelden geven, te beginnen met enkele van matige orde, zodat het hoofdprogramma er op verschillende manieren mee kan spelen.

TYPE ARGUMENT, FUNCTIE, DIFFERENTIE

Wij nemen aan, dat het hoofdprogramma x en f(x) voor opvolgende waarden van x berekent en $x_1 f(x)$ en $\nabla f(x)$ uitgetypt moeten worden. Daarbij moet bedacht worden. dat op de eerste regel natuurlijk $\nabla f(x)$ ontbreken moet. Het hoofdprogramma moet voorts de mogelijkheid hebben om via de subroutine een regel nog eens over te typen, bijv. bovenaan een nieuwe pagina. Tevens is er voor gezorgd dat er ingangen zijn, waarbij

het typen van $\nabla f(x)$ of van x, of van x en $\nabla f(x)$ wordt onderdrukt. Als het typen van $\nabla f(x)$ wordt onderdrukt, wordt er altijd voor in de plaats een door de regelindelingsubroutine van het communicatieprogramma getelde tabulatie gegeven.

De aanroepen zijn:

```
NORMAAL:  24    6  XO
           7       FO =) TYPT x, f(x), BEWAART f(x), BEREKENT,
                         BEWAART EN TYPT   f(x).
HERHAAL:  24    6  XO
           7   19  FO =) TYPT x, f(x), ∇f(x) ZOALS BEWAARD, LAAT
                         ALLES ONGEREPT.
AANLOOP: 2(10) 25 FO     TE GEVEN VOOR DE REKENCYCLUS VAN HET
         4(12)  8 FO     HOOFDPROGRAMMA. ONDERDRUKT EENMAAL HET
                         TYPEN VAN   f(x).
```

Als in plaats van 7 FO gebruikt wordt

7 27 FO, dan wordt $\nabla f(x)$ niet getypt,

7 28 FO, dan worden x en $\nabla f(x)$ niet getypt,

7 29 FO, dan wordt x niet getypt.

De voorponsing specificeert op welke wijze men x, f(x) en $\nabla f(x)$ wenst te typen, dus als geheel getal of als breuk en zonder of met teken, alsmede de adressen, waar x en f(x) te vinden zijn.

```
A   1001  XO
A   1001  XO
... ....  X3   "A" TYPE x
O         XO
... ....  X3   "B" TYPE f(x)
O         XO
... ....  X3
O         XO   "C" TYPE ∇f(x)
O   ....  ..
O         XO   "D" ADRES x
O   ....  ..
O         XO   "E" ADRES  f(x)
O         XO
O   ....  ..
O         XO   "F" ADRES SUBROUTINE.
F     30  X3
```

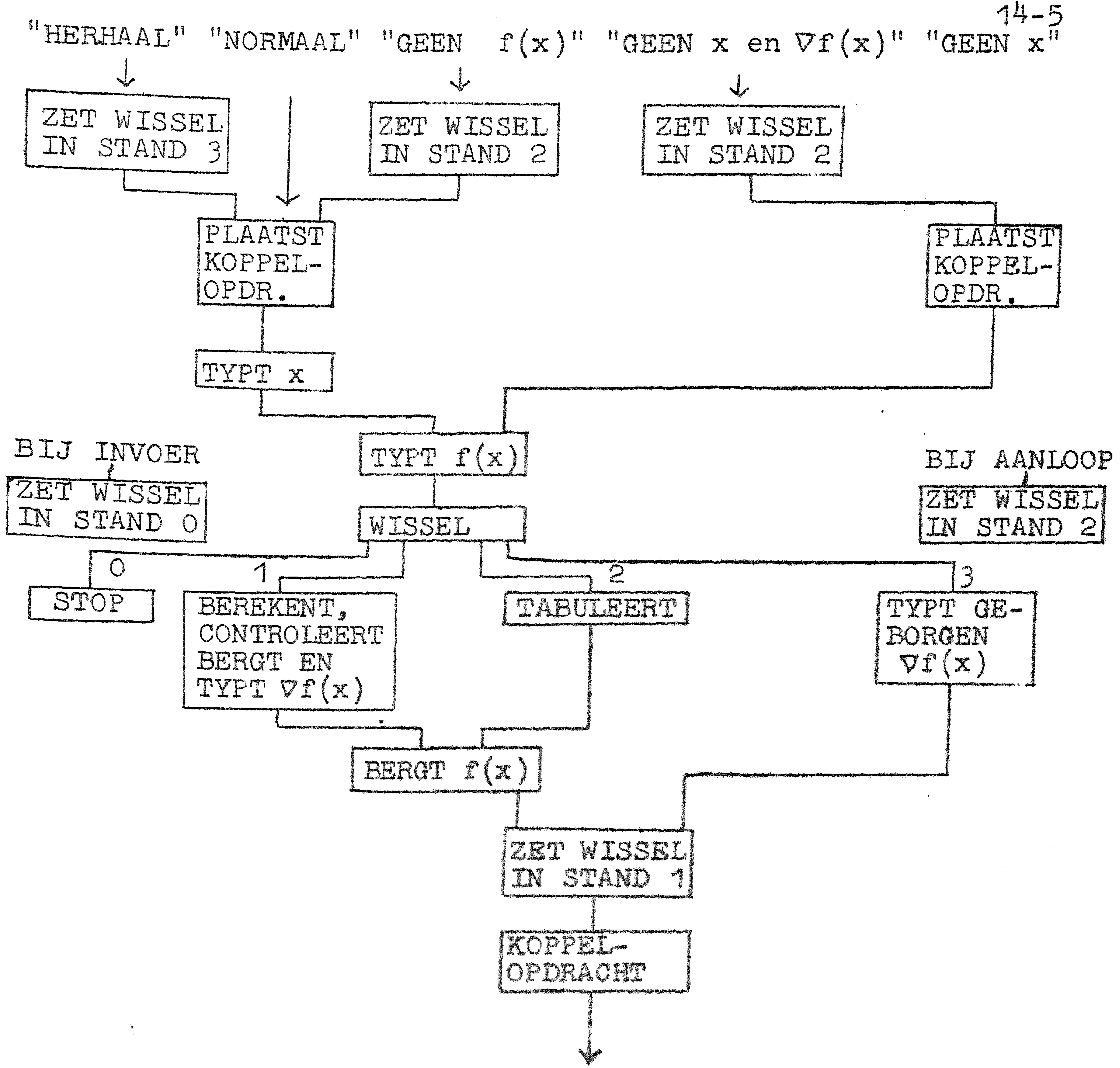
Het programma, dat hieronder volgt, bevat vele controle-elementen, die als illustratie van het in de vorige syllabus behandelde zijn te beschouwen, alsmede een merkwaardige programmawissel, die niet eenvoudig door een blokschema is te beschrijven. Het is tevens een goede oefening in het gebruiken van sluitletters.

```
                     A       FO
                     A       FO
"NORMAAL"  =)0       4   18  FO
3b FO  -+           10       DO    x
                  1 24    6  XO
                     0       AO  =) TYPT x
                  2  9       DO    x
                    24    6  XO
                  3  7       X2 =)
                    14       FO  -+
                  4 24       XO
7bFO  -+
27bFO               10       EO  f(x)
                  5 24    6  XO
                     0       BO  =) TYPT f(x)
                  6  9       EO   f(x)
                    24    6  XO
                  7  7       X2 =)
                    14    4  FO  -+
13aFO -+
23aFO             8 (-       X)
                  9 12   31  FO  ∇f(x)
                    10   31  FO
                 10 24    6  XO
                     0       CO  =) TYPT  ∇f(x)
                 11  9       EO  f(x)
                     8   30  FO  f(x)  oud
8aFO--+          12 24    6  XO
                     7       X2  =)
                 13  6    8  FO  -+
16aFO-+              2       EO  f(x)
                 14  4   30  FO  f(x)  oud
                     2   30  FO
                 15  1       EO  f(x)
                    24    8  X3
                 16 14   13  FO  -+
23bFO-+              2   24  FO
                 17 24       XO
                     4    8  FO
                 18 (+       X)  = +

"HERHAAL"=)19       10   26  FO
27bFO-+             12    8  FO
                 20  7       FO  =+
8bFO=+              24    6  XO
```

```
                     21   0       CO =) TYPT. ∇f(x)
                          9  31   FO    ∇f(x)
                     22  24   6   XO
                          7       X2 =)
                     23   6   8   FO -+
                         15  16   FO =+
                    "24  10       EO
                          9  30   FO
                    "25   7  12   FO
                         24       XO
                    "26  10  31   FO
                         15  20   FO
"GEEN ∇f(x)"=)27         10  25   FO
                         15  19   FO =+
"GEEN x en
∇f(x)"=)             28  10  25   FO
                         12   8   FO
"GEEN x" =)          29   4  18   FO
                         15   4   FO
                     30  (+       X) f(x) oud

                     31  (+       X)  ∇f(x)
                          F  30   X3
```



"HERHAAL"
"NORMAAL"
"GEEN f(x)"
"GEEN x en ∇f(x)"
"GEEN x"
ZET WISSEL IN STAND 3
ZET WISSEL IN STAND 2
ZET WISSEL IN STAND 2
PLAATST KOPPEL-OPDR.
PLAATST KOPPEL-OPDR.
TYPT x
TYPT f(x)
BIJ INVOER
ZET WISSEL IN STAND 0
BIJ AANLOOP
ZET WISSEL IN STAND 2
WISSEL
0
1
2
3
STOP
BEREKENT, CONTROLEERT BERGT EN TYPT ∇f(x)
TABULEERT
TYPT GE-BORGEN ∇f(x)
BERGT f(x)
ZET WISSEL IN STAND 1
KOPPEL-OPDRACHT

Syllabus No. 14 van de cursus 1955-'56:
Programmeren voor automatische rekenmachines

Onder leiding van

Prof. Dr. Ir. A. van Wijngaarden en de Heer E.W. Dijkstra

DE ADMINISTRATIEVE SUBROUTINE II.
INTEGRATIE

In wetenschappelijke berekeningen treedt vaak een integraal op, dus een uitdrukking van de vorm

$$\int_a^b f(x)dx.$$

Als gegeven is, hoe f(x) uit x volgt en wat a en b zijn, staat hier een volledig bepaald getal. Natuurlijk kan men geen routine maken, die alle integralen uitrekent zonder f(x) te definieren. We zullen derhalve aannemen, dat er een subroutine bestaat, die met (S) = x aangeroepen terugkeert met (S) = f(x). Geven we dan nog a en b op een of andere wijze, dan moet het berekenen van de integraal op een of andere wijze uniform kunnen geschieden en daarvoor kan een subroutine gemaakt worden, zodat de programmeur zich daar verder geen zorgen meer over behoeft te maken, maar het integraalbegrip kan inlijven bij zijn standaardroutines voor de machine.

Voor we de werking van deze subroutine analyseren, moeten we ons echter eerst realiseren, dat, willen we onze f(x) niet beperken tot een behalve voor het candidaatsexamen oninteressante kleine groep van functies, die in gesloten vorm geintegreerd kunnen worden, terwijl anderzijds het niet de bedoeling is om andere dan Riemannse integralen te beschouwen, het enerzijds nodig maar anderzijds ook voldoende is om de numerieke analyse te hulp te roepen. Dit betekent, dat we niet kunnen verwachten het precieze antwoord voor de integraal te vinden, d.w.z. de op een voorgeschreven aantal decimalen (of binalen) afgeronde waarde van het exacte resultaat, maar dat we alleen kunnen eisen, dat althans zeer waarschijnlijk het gevonden antwoord van het juiste antwoord niet meer verschilt dan een opgegeven tolerantie. Deze tolerantie zal dus ook worden gespecificeerd, maar het wordt tot de taak van de subroutine geacht ervoor zorg te dragen, dat deze tolerantie niet wordt overschreden.
De subroutine mag dus niet meer informatie vragen, dan iedere geschoolde rekenaar zou doen, nl. de betekenis van het symbool f(x), de waarden van a en b en tenslotte de tolerantie van het antwoord.

Waarschijnlijk zal een dergelijke subroutine met het oog op haar algemeenheid niet voor iedere functie en onder alle omstandigheden het allersnelste werken, maar zij spaart onnoemelijk veel denktijd en programmeertijd en kan natuurlijk zorgvuldig en snel geprogrammeerd worden, zodat ze toch niet zo heel gemakkelijk overtroefd zal worden met een ad hoc programma.

Men moet dus van een of andere numerieke integratieformule gebruik maken en in het te behandelen voorbeeld is de formule van SIMPSON gebruikt. Integreert men van C tot C + 4h met stappen van 2h, h of 0 (d.w.z. exact), dan geldt voor deze deelintegraal:

$$\int_C^{C+4h} f(x)dx = 4h\left\{F(2h) + R(2h)\right\}$$

$$= 4h\left\{\frac{1}{6}\ f(C) + \frac{2}{3}\ f(C + 2h) + \frac{1}{6}\ f(C + 4h) + R(2h)\right\};$$

$$\int_C^{C+4h} f(x)dx = 4h\left\{F(h) + R(h)\right\}$$

$$= 4h\left\{\frac{1}{12}\ f(C) + \frac{1}{3}\ f(C + h) + \frac{1}{6}\ f(C + 2h) + \frac{1}{3}\ f(C + 3h) + \frac{1}{12}\ f(C + 4h) + R(h)\right\};$$

$$\int_C^{C+4h} f(x)dx = 4h\ F(0)$$

Wat we graag zouden weten is F(0), maar berekenbaar zijn alleen F(2h) en F(h). Evenwel weten wij, dat bij benadering voor de overigens onbekende resttermen geldt:

$$16\ R(h) \approx R(2h)$$

en wel des te nauwkeuriger naarmate h kleiner is. Een schatting van R(h) volgt dus uit:

$$F(h) - F(2h) = R(2h) - R(h) \approx 15R(h).$$

De procedure is dus, dat men F(2h) en F(h) beide berekent, toeziet, dat het verschil klein is en zo dit het geval is hieruit een benaderde waarde van R(h) berekent. Afgezien van pathologische gevallen vindt men zo een schatting van F(0), welke aanzienlijk beter is dan het verschil F(h) - F(2h). Laten wij daarom een interval h slechts als genoegzaam klein accepteren als

$$\left|F(h) - F(2h)\right| < \Delta_{max},$$

Waarin Δ_{max} een gegeven klein positief getal is. Ongeacht, hoe h dan ook moge varieren over het totale integratiegebied van a tot b, is

$$\left|b - a\right| \Delta_{max}$$

een pessimistische schatting van de fout welke gemaakt wordt in de volledige integraal van a tot b. Hoe kleiner fout getolereerd wordt, des te kleiner moet Δ_{max} gekozen worden, des te kleiner h wordt getolereerd en des te zekerder is het, dat de werkelijke fout aanzienlijk kleiner zal zijn.

De gedachtengang van de subroutine is nu, dat zij zelf h tracht te kiezen door de bovenstaande ongelijkheid te testen. Is daaraan niet voldaan, dan verwerpt zij h, halveert h en probeert het opnieuw. Alleen als $|h|$ gelijk mocht zijn aan 1 peuter, dan wordt noodgedwongen dit "grove" interval geaccepteerd. Is omgekeerd aan de gelijkheid wel voldaan, dan wordt h geaccepteerd en de bijdrage tot de integraal wordt alvast verdisconteerd. Evenwel wordt daarna getest of soms

$$|F(h) - F(2h)| < \Delta_{min},$$

Waarin Δ_{min} een kleine fractie van Δ_{max} is. Is dit nl. het geval, dan betekent het, dat h wel groter had mogen zijn. De keuze van $\Delta_{min}/\Delta_{max}$ is een strategisch probleem. Kiest men Δ_{min} zeer klein t.o.v. Δ_{max} dan wordt h onnodig lang klein gehouden. Kiest men Δ_{min} weinig kleiner dan Δ_{max}, dan loopt men grote kans dat de verdubbelde stap 2h direct weer wordt afgekeurd, wat eveneens onnodig rekenwerk tengevolge heeft. Omdat verdubbeling van h de restterm ongeveer met 16 vermenigvuldigt, is $\Delta_{max} = 16\,\Delta_{min}$ ongeveer de kritieke verhouding. Het programma kiest $\Delta_{max} = 20\,\Delta_{min}$.

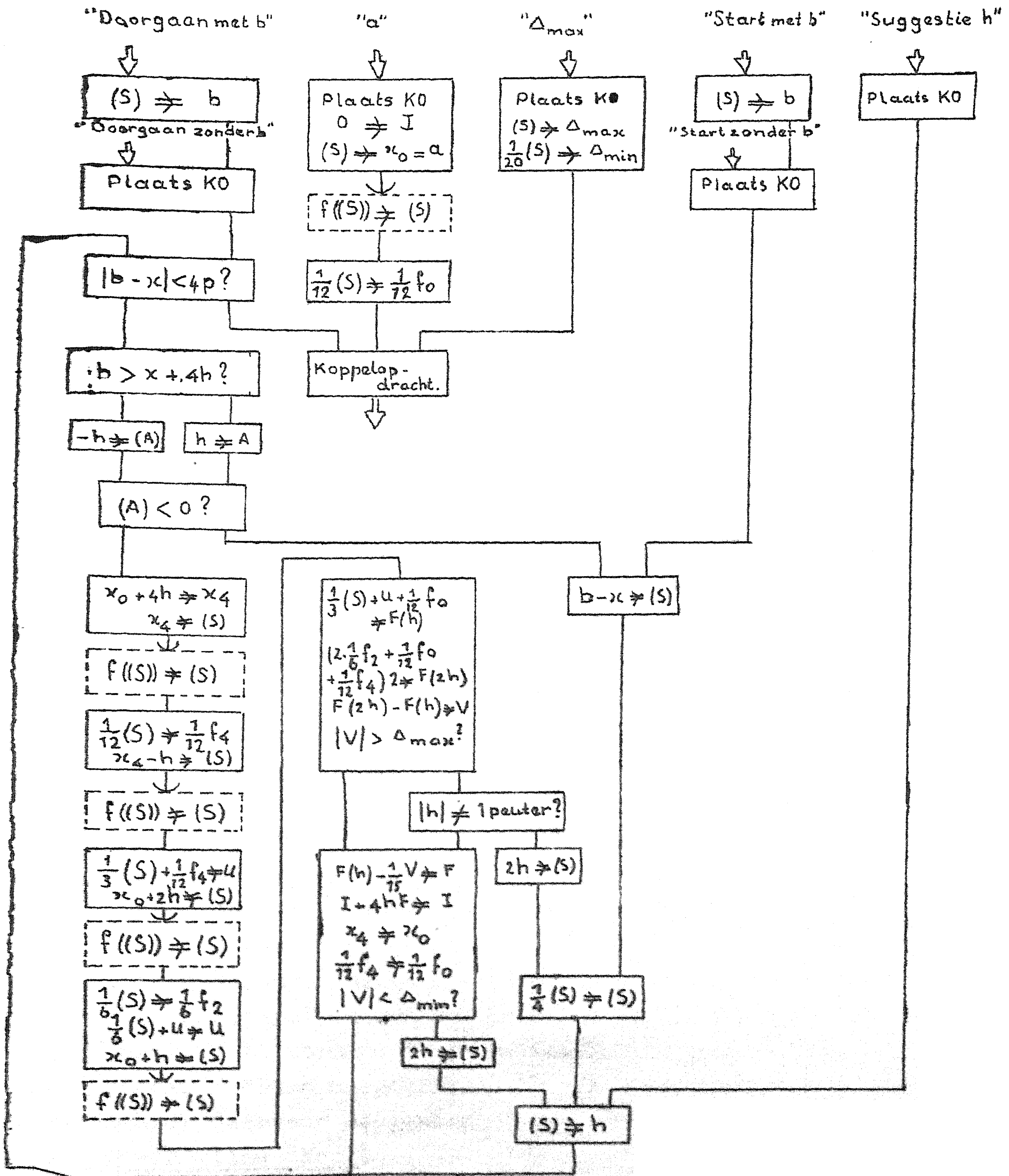
Zodra een integratiestap van C tot C + 4h is geaccepteerd, moet de gevonden F nog met 4h worden vermenigvuldigd, om de bijdrage tot de integraal te vinden. Deze vermenigvuldiging wordt in dubbele lengte uitgevoerd. Dit heeft een dubbele reden. Ten eerste is er de kwestie, dat in de omgeving van de punten waar f(x) zich wild gedraagt, h zeer klein wordt. Een deel van de integraal kan dus worden opgebouwd uit een zeer groot aantal zeer kleine bijdragen. Om de precisie niet te laten lijden is dubbele lengte rekenen dus al noodzakelijk. Ten tweede wil men vaak de integraal achteraf nog met een of andere factor vermenigvuldigen bijv. met $\pi/2$ of de subroutine levert eigenlijk niet f(x) maar bijv. $f(\pi x)$ of iets dergelijks, zodat de x die als integratievariabele fungeert eigenlijk slechts op een factor na de x is die voor f(x) wordt gebruikt. Dit betekent ook, dat de integraal met een of andere factor moet worden vermenigvuldigd. Om aan deze moeilijkheid tegemoet te komen, wordt afgesproken, dat a en b in dezelfde maat worden uitgedrukt als de x welke aan de subroutine voor f(x) moet worden verstrekt en ook h wordt in die maat geteld. De bijdrage tot de integraal zelf,

dus de h die in 4h voorkomt wordt evenwel vermenigvuldigd met een factor φ , die men van te voren moet geven in dubbele lengte, nl. als een geheel gedeelte φ' en een breukdeel φ''. Als niets bijzonders aan de hand is, neemt men $\varphi'' = 1$ en $\varphi'' = 0$, uit welke conventie dan de keuze van φ in andere gevallen volgt. De integraal wordt ook afgeleverd in geheel gedeelte I' en breukdeel I". De subroutine heeft verschillende aanroepen. Allereerst is er de aanroep "Δ_{max}", die (S) als Δ_{max} bergt, hieruit Δ_{min} berekent en bergt. Vervolgens is er de aanroep "a", die (S) als a bergt, I (= I',I") schoonmaakt en vast $\frac{1}{12}$ f(a) uitrekent en bergt als $\frac{1}{12} f_0$. Aan iedere integratie moet natuurlijk één maal het zetten van φ' , φ'' , Δ_{max} en a voorafgaan. Het zetten van φ' en φ'' is te triviaal om door een inloop te laten verzorgen. De aanloop ziet er in ARRA-code als volgt uit, als de subroutine op F0 t/m F3 staat:

```
 2   ..  ..  φ'
10   ..  ..  φ"
 4    3  F3  Zet φ'
12    5  F3  Zet φ"
10   ..  ..  Δmax
24    6  X0
 7   11  F2  =) Zet Δmax en Δmin
10   ..  ..  a
24    6  X0
 7    4  F2  =) Zet a, I en f0
```

De subroutine is gereed om te gaan integreren. Wil men één enkele integraal, dan roept men haar aan met (S) = b op de ingang "Start met b". De subroutine bergt b, kiest h = (b - a)/4 om te beginnen em gaat aan de slag en komt uiteindelijk terug als de integraal is berekend, die wordt afgeleverd op 11F3 (I') en 9F3 (I"). Het kan zijn, dat b zelf een berekeninsresultaat was dat al eerder berekend was en in eens op de goede plaats, nl. 4F3 was opgeborgen. In dit geval hoeft men b natuurlijk niet opnieuw te specificeren en kan men met de openingsvariant "Start zonder b" beginnen. In dit laatste geval kan het gebeuren, dat men van te voren weet, dat h veel kleiner moet zijn dan (b - a)/4, wat eigenlijk een inconsequentie is, omdat de programmeur zich daarover geen zorgen behoeft te maken. Om tijdsverlies tengevolge van het vele malen afwijzen van h en en halveren te beperken kan men dan met een suggestie voor h in S aanroepen op de ingang "Suggestie h".

Als de subroutine klaar is met haar werk heeft zij alle te gebruiken gegevens zorgvuldig behouden. Wenst men dan verder te integreren (bijv. bij de berekening van een lopende integraal) dan roept men haar met de nieuwe b in S aan op de hoofdingang "Doorgaan met b" (of op de variant "Doorgaan zonder b"). Het blokschema verduidelijkt de werking van de subroutine, speciaal wat betreft het gemanipuleer met de h.



TABELLATIE

Tenslotte geven wij als voorbeeld van een administratieve subroutine van in hoge mate overkoepelend karakter één, die tot taak heeft een tabel te maken van één of meer functies van één variabele. Het is er een, die volkomen in staat is te werken zonder echt hoofdprogramma en dus met het ideale hoofdprogramma bediend kan worden.

De opgave luidt precieser een functie f(x) te tabelleren voor x = a(h)b, met de restrictie, dat a en b exacte veelvouden van h zijn, wat praktisch altijd het geval is. Overigens is door een kleine modificatie deze restrictie te vermijden. De functie f(x) mag een vector zijn, d.w.z. een aantal functies eventueel met differenties, enz. De paginaindeling moet flexibel zijn maar aan hoge eisen voldoen. Daartoe is verondersteld, dat een aantal andere administratieve subroutines ter beschikking staan. Deze zullen wij eerst de revue laten passeren.

Subroutine "Functie". Deze is in staat om bij gegeven x, die op een vaste plaats in tabellatie te vinden is f(x) te berekenen. Eventuele differenties behoeven niet te worden berekend; daar wordt elders zorg voor gedragen. De subroutine kan anders moeten ageren dan normaal als zij voor de eerste keer moet rekenen, bijv. als zij lopende integralen met behulp van "Integratie" berekent. Daarom zullen wij de mogelijkheid open laten, dat zij een speciale ingang heeft voor dat geval. Wij noemen haar dan "Nulfunctie". De twee inloopadressen moeten natuurlijk aan "Tabellatie" worden meegegeven. Als er geen bijzondere Nulfunctie-ingang is wordt daarvoor dan de gewone ingang meegegeven.

Subroutine "Regeltype".

Deze is in staat om bij gegeven x en f(x), die op medegedeelde plaatsen in "Functie" te vinden zijn een nette regel uit te typen eventueel voor differenties, enz. zorgdragend. Een voorbeeld daarvan hebben wij behandeld. Bijv. in het geval dat differenties door "Regeltype" berekend worden moet zij speciale ingangen hebben, nl. "Nultype" voor de eerste regel en "Herhaaltype" om een regel te herhalen zonder opnieuw differenties te vormen. De drie inloopadressen moeten aan "Tabellatie" worden meegegeven. Als er geen bijzondere inlopen zijn wordt daarvoor de gewone ingang meegegeven.

Subroutine "Kop".

Deze verzorgt de bovenzijde van de pagina boven de eigenlijke

tabel. Bijvoorbeeld typt zij een streep over de breedte van het papier om het later precies te kunnen afsnijden, voert dan het papier een voorgeschreven aantal regels op en typt vervolgens indicaties boven de verschillende kolommen. Als men het paginanummer boven de pagina wenst, dan is daarvoor in "Tabellatie" op een vaste plaats de grootheid p te vinden die de pagina's telt. Onder de indicaties wordt weer het papier enkele regels opgevoerd.

Subroutine "Staart".

Deze verzorgt de onderzijde van de pagina onder de eigenlijke tabel, ook bijv. met indicaties of een paginanummer. Het is verstandig om haar ook tenslotte een streep onderaan te laten typen, opdat men de bovenzijde en onderzijde van de pagina's kan afsnijden. Als het papier wat scheefgetrokken is, kan men dan de machine stoppen na "Staart" en het papier weer rechtzetten. Natuurlijk kunnen "Kop" en "Staart" weer gemeenschappelijke subroutines gebruiken, bijv. de subroutine "Streep", die een streep over het papier typt!

De gehele rest van de administratie wordt door "Tabellatie" verzorgd. Zij hoogt het argument x op van a tot en met b en ziet toe op de paginaindeling. Allereerst zorgt zij ervoor, dat de pagina in Q blokjes van R regels wordt getypt. Van het Q + 1ste blokje wordt dan nog de eerste regel getypt, de pagina wordt met "Staart" afgewerkt en op de nieuwe pagina, die met "Kop" wordt voorbewerkt, wordt deze extra regel herhaald. Zij doet echter nog meer. Zij zorgt er nl. voor dat het argument x = 0 in de tabel op de eerste regel van een pagina getypt wordt of, als het in werkelijkheid niet voorkomt, getypt zou worden als men het interval voor x zou vergroten. Bij de eerste pagina zorgt zij er nl. voor dat na "Kop" het papier zover wordt opgevoerd, dat dit voor elkaar komt. Evenzo, zorgt zij ervoor dat de laatste pagina van de tabel van dezelfde lengte wordt door zo nodig voor "Staart" het papier een passend aantal regels op te voeren. Dit zijn schijnbaar futiliteiten, maar zij zijn in werkelijkheid absoluut noodzakelijk als men goedverzorgde tabellen wil afleveren.

Het bijgaande blokschema spreekt nu voor zichzelf.

Wij sluiten hiermede de behandeling van de administratieve subroutines af. Om zich goed te realiseren welke macht de subroutines voor speciale functies en de administratieve subroutines hebben, is het aardig om het volgende vraagstuk te

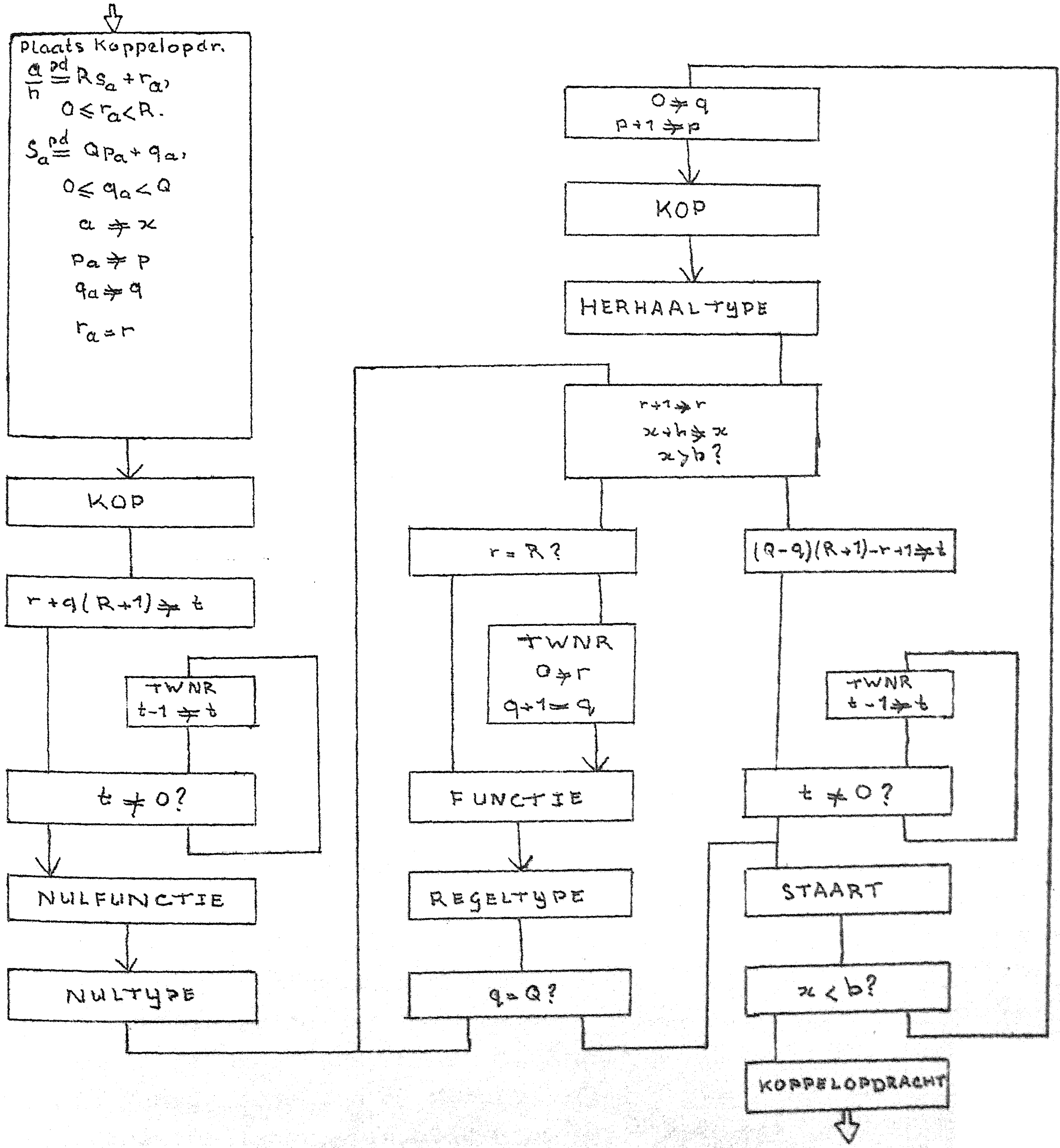
analyseren:

Maak een tabel in 6D van de Fresnelintegralen

$$C(x) = \int_0^x \cos\frac{\pi}{2}t^2.dt \quad \text{en} \quad S(x) = \int_0^x \sin\frac{\pi}{2}t^2.dt$$

voor x = 0(0.001)25 met differenties.

Men zal dan zien, dat men praktisch niets behoeft te programmeren!



Syllabus No. 16 van de cursus 1955-'56:
Programmeren voor automatische rekenmachines
Onder- leiding van
Prof. Dr. Ir. A. van Wijngaarden en de Heer E.W. Dijkstra

SUBROUTINE-AGGREGATEN VOOR SEMI-INTERPRATIEF WERK

Wij hebben gezien, hoe de opdrachtencode van de ARRA door de subroutines als het ware werd uitgebreid. Enerzijds betrof het hier de "functie-subroutines" (als wortel, sinus, logarithme etc.), anderzijds de administratieve subroutines (als typen, typcontrole, regeltypen, tabulatie en integratie.) Zij laten zich gemakkelijk met de normale machine-code combineren, omdat zij manipuleren met getallen in de normale representatie: zij verrichten andere operaties dan de ingebouwde opdrachten, maar sluiten daarbij aan, doordat zij opereren op "dezelfde soort" getallen.
Een duidelijk ander aspect krijgen subroutines, zodra zij met getallen in andere representaties werken. Van mogelijke andere representaties laten wij enige aspecten de revue passeren.

Multilengte-rekentechniek

Er kunnen berekeningen voorkomen, waarin een precisie van 29 binalen (d.w.z. circa "$8\frac{1}{2}$ decimaal") ontoereikend is. Er zijn slechts $2^{30}(-1)$ verschillende woorden; als wij meer verschillende getallen willen kunnen onderscheiden, is het duidelijk dat wij niet kunnen volstaan met elk getal voor te stellen door een enkel woord. Stellen wij per getal bijv. twee woorden ter beschikking - men spreekt dan van dubbellengte-rekentechniek - dan is het aantal mogelijke verschillende getallen gestegen tot zo iets als 2^{60}. (Het is niet gezegd, dat in feite ook zoveel verschillende getallen gerepresenteerd zullen worden: òf sommige mogelijkheden blijven onbenut, of verschillende cijferconfiguraties zijn arithmetisch aequivalent, d.w.z. representeren hetzelfde getal.) In de keuze van de representatie hebben wij enige vrijheid: men zal de keuze zo maken, dat de vereiste operaties op de getallen zo gemakkelijk mogelijk verlopen. Deze keuze hangt dus, naast de fantasie van de programmeur, nogal van de code van de machine af. Als wij in de ARRA bijv. driedubbellengte-rekentechniek bedrijven, met breuken, kleiner dan 1, en stellen we voor elk getal drie opeenvolgende adresplaatsen ter beschikking, dan liggen de volgende mogelijkheden voor de hand:

$$[n].2^{-29} + [n+1].2^{-58} + [n+2].\ 2^{-87} \quad \text{of}$$
$$[n].10^{-8} + [n+1].10^{-16} + [n+2].10^{-24}.$$

M.a.w.: we schrijven het getal in het 2^{29}-tallig- of in het 10^8 tallig stelsel en plaatsen de successieve "cijfers" in successieve adressen. De invoer en uitvoer is bij de laatste keuze kennelijk gemakkelijker, de rekentijd wordt in een binaire machine echter langer. Bovendien is de eerste methode zuiniger met geheugenruimte. Een tweede punt is, wat we doen met het tekencijfer: over het algemeen worden de conventies zo gemaakt, dat (n), (n + 1) en (n + 2) altijd hetzelfde teken hebben. Men realisere zich, dat dit niet noodzakelijk is! (Er is geen wiskundig bezwaar tegen, om 40,7 voor te stellen als +41 -.3.)

Drijvende komma rekentechniek

Er zijn ook berekeningen, waar op zichzelf de relatieve precisie der woordlengte (in ons geval dus $8\frac{1}{2}$ decimaal) voor de getallen wel voldoende is, maar waar de orde van grootte zo varieert, dat wij ze niet met een vaste schalingsfactor binnen de capaciteitsgrenzen -1 en +1 van de machine kunnen brengen, zonder daarbij ongepermitteerd precisie te verliezen. In dat geval moet de machine zelf een schalingsfactor construeren; is deze exponentieel bepaald en heeft elk getal zijn eigen schalingsfactor, dan spreekt men van drijvende komma rekentechniek. Dit komt er in wezen op neer, dat men elk getal x representeert door twee getallen, nl. de breuk b en het gehele getal m, zodat

$$x = b.g^m, \text{ waarbij } g^{-1} \leq |b| < 1.$$

Als g gekozen is leggen b en m enerzijds, x anderzijds elkaar eenduidig vast. Voor g komen 2 en 10 in aanmerking. De opmerkingen, die bij de dubbellengte-getallen over de keuze tussen 2^{-29} en 10^{-8} gemaakt zijn, zijn hier voor 2 en 10 met dezelfde argumentatie van toepassing. Een moeilijkheid is, dat het getal 0 niet kan worden voorgesteld.

Compleze getallen

In wetenschappelijke berekeningen wordt vaak verlangd te rekenen met zg. complexe getallen van de gedaante

$$z = a + bi$$

waar a en b normale getallen zijn (in dit verband ook wel reële getallen genoemd), terwijl i de imaginaire eenheid is, met de eigenschap $i^2 = -1$. Men noemt dan a het reële deel van z, en b (soms bi) het imaginaire deel van z. De reële

getallen zijn op te vatten als complexe getallen, met imaginair deel = 0. Als

$$z = a + bi \text{ en } w = c + di,$$

dan gelden de volgende rekenregels:

1. $z + w = a + bi + c + di = (a + c) + (b + d)i$
   Dus reëel, resp. imaginair, deel van de som van twee complexe getallen is gelijk aan de som van de reële, resp. imaginaire delen der termen.
2. $zw = (a + bi)(c + di) = ac + adi + bci + bdi^2 = (ac - bd) + (ad + bc)i$
3. $\frac{1}{z} = \frac{1}{a + bi} = \frac{1}{a + bi}.\frac{a - bi}{a - bi} = \frac{a - bi}{a^2 + b^2} = (\frac{a}{a^2 + b^2}) - (\frac{b}{a^2 + b^2})i,$
   waaruit volgt:

$$\frac{w}{z} = \frac{c + di}{a + bi} = \frac{ac + bd}{a^2 + b^2} + \frac{ad - bc}{a^2 + b^2}i$$

Om in een machine het complexe getal $z = a + bi$ te bergen, bergt men in het geheugen niet de imaginaire eenheid i, maar wel a en b, bijv. op twee successieve geheugenplaatsen en spreekt af, dat de eerste het reële deel a, de tweede het imaginaire deel b zal bevatten.

Hier hebben wij drie aspecten van getal-representaties genoemd; zij kunnen alle gecombineerd worden: men kan - desnoods! - rekenen met dubbellengte drijvende komma getallen, of met complexe drijvende komma getallen - bij deze laatste combinatie mag men kiezen, of reëel en imaginair deel elk hun afzonderlijke macht m hebben, of dat men schrijft:

$$z = (a + bi).g^m \text{ met } g^{-1} \leq \sqrt{a^2 + b^2} < 1.$$

Het is duidelijk, dat men in een programma, dat met zulke getallen moet manipuleren gebruik gaat maken van subroutines voor de arithmetische operaties. Overwegingen van de beschikbare programmaruimte zullen ons hier doorgaans toe dwingen, en is het niet de programmaruimte, dan is het wel de anders vereiste denkarbeid - wij zullen zien dat er bij deze subroutines nog wel een en ander komt kijken! - , die ons van het idee afbrengt, een dergelijk programma ab initio en subroutineloos op te stellen. Aan de andere kant vraagt de aard van het probleem om subroutines voor de afzonderlijke algebraische bewerkingen: het stuk programma, dat bv. twee drijvende komma getallen met elkander vermenigvuldigt, is een in zijn

logische functie isoleerbaar stuk, van zulk een algemeenheid, een zozeer afgerond geheel, dat er zelfs één wiskundig symbool voor bestaat, nl. het maalteken!

In twee opzichten onderscheiden deze subroutines zich van wat we tot nog toe gezien hebben.

Ten eerste komen ze praktisch nooit in hun eentje in een programma voor. (De dubbellengte optelling is m.i. het enig denkbare geval, dat hierop misschien een uitzondering maakt.) Vandaar dat in de titel van deze syllabus gesproken wordt van subroutine aggregaten. Zo is er een dergelijk aggregaat van subroutines die de afzonderlijke arithmetische bewerkingen (en nog meer) bewerkstelligen op enkellengte complexe getallen in de representatie

$$z = (a + bi)2^m \text{ met } \tfrac{1}{2} \leqslant \sqrt{a^2 + b^2} < 1;$$

elke z beslaat hier in het geheugen dus drie adressen.

Ten tweede plegen de subroutines uit één aggregaat in reeksen achter elkaar aangeroepen te worden: de berekening van een of andere algebraische vorm eist toch bijna altijd meer dan een arithmetische bewerking; dit is geheel analoog aan de opbouw van een normaal programma, waar rijen opdrachten voor de "werkelijke" berekening (aanhalen, rekenen en wegschrijven) door stukjes administratie gescheiden worden.

Dat deze subroutines samen een aggregaat vormen, komt o.a. tot uiting in gemeenschappelijke werkruimtes, speciaal zij, die samen de zg. quasi-accumulator vormen. Het volgende arrangement heeft nl. zijn deugdelijkheid bewezen: geheel analoog aan de een-adrescode, waaraan wij allen gewend zijn, opereren de arithmetische subroutines van een bepaald aggregaat op een getal, dat - van een paar successieve adressen - uit het geheugen komt, en op de "inhoud van de quasi-accumulator" (d.w.z. het getal, dat door de inhouden van de betroffen gemeenschappelijke werkruimtes is bepaald); het resultaat laten zij weer in de quasi-accumulator achter. Een eerste consequentie is, dat het subroutine-aggregaat naast arithmetische subroutines ook transportsubroutines kent (van het geheugen "schoon" in de quasi-accumulator en omgekeerd). Dat dit transport werkelijk door subroutine geschiedde, houdt o.a. verband met de - zich inderdaad voordoende en niet onbenut gelaten - mogelijkheid, dat de rekensnelheid wel eens opgevoerd kon worden, indien de inhoud der adressen van de quasi-accumulator niet exact volgens dezelfde conventies geïnterpreteerd wordt, als in een bij elkaar behorend groepje adressen in het geheugen. (Er is bv. niet de minste reden, waarom de adressen van de quasi-accu-

mulator successieve adressen zouden zijn.)

Nu ook het transport van en naar de quasi-accumulator per subroutine gaat, is de kans op langere ondoorbroken reeksen aanroepen groter geworden; ook dit kan benut worden. In een dergelijke reeks is het nl. niet nodig, om steeds met behulp van de 24 6 XO de subroutines aan te roepen. Immers de opdracht 24 6 XO maakt het mogelijk, om een subroutine aan te roepen van een willekeurig punt in de machine: als wij een ondoorbroken reeks aangeroepen hebben, is deze faciliteit wat overbodig: zodra de spatiering van de inspringpunten in het hoofdprogramma constant is, hoeven we alleen aan te geven, waar de besturing de eerste keer terug moet komen: de volgende keer telt het programma deze vaste spatiering bij de koppelopdracht van de vorige subroutine op. (De situatie is geheel analoog aan die bij de invoer, waar we met de geponste adresindicatie alleen de plaats van het eerste molecuul specificeren: volgende moleculen worden op volgende plaatsen opgeborgen. Om de koppelopdracht te kunnen berekenen uit die van de vorige subroutine, is het nodig, te weten, waar deze staat: omdat de vorige subroutine (praktisch) elke subroutine uit het aggregaat kan zijn, wordt het volgende arrangement gesuggereerd: behalve de gemeenschappelijke quasi-accumulator hebben de subroutines van een aggregaat nog meer gemeenschappelijk: nl. de zich ophogende koppelopdracht: na afloop van elke subroutine springt de besturing naar een vast punt, waar de gemeenschappelijke koppelopdracht ontmoet wordt, nadat het adres hiervan met het increment is opgehoogd. Een consequentie is, dat voor de eerste subroutine van een reeks iets aparts gedaan moet worden. Dit is de zg. inloopscombinatie, die op de plaats van de gemeenschappelijke koppelopdracht invult, wat er gestaan zou hebben, als de eerste subroutine door een "nulde" zou zijn voorafgegaan.

In verband met de benodigde programma-ruimte is het gewenst het increment zo klein mogelijk te kiezen. Het is echter wel prettig, als het increment een even aantal opdrachten is, omdat dan het verhogen van de gemeenschappelijke koppelopdracht door een optelling in het adresgedeelte bewerkstelligd kan worden. (Zou het increment een oneven aantal opdrachten zijn, dan zou op de plaats van de gemeenschappelijke koppelopdracht 7- en 15-opdracht elkaar af moeten wisselen. Om dit te effectueren kost tijd!) Met inachtneming van deze overwegingen wordt ons als ideaal gesuggereerd een increment van 1 in het adresnummer, dus twee opdrachten. Is dit niet

te weinig? De tweede van deze twee opdrachten zal een sprong zijn naar een van de subroutines (het adres van de sprongopdracht bepaalt dus, welke operatie wordt uitgevoerd; de adrescijfers fungeren dus als functiecijfers). De eerste opdracht zal op een of andere manier aan de subroutine moeten meegeven op welk getal in het geheugen de operatie betrekking heeft (dus bijv. welk getal moet worden aangehaald, om bijv. als addendum te fungeren.) Wij zullen deze informatie in S zetten. Wij kunnen niet het getal zelf in S plaatsen, daar dit als regel enige woorden zal beslaan; dus moeten wij enige informatie in S plaatsen, waaruit de betroffen subroutine destilleren kan, waar in het geheugen (de rest van) het getal te vinden is. Deze informatie kan alleen uit het geheugen komen. Deze overwegingen hebben geleid tot het volgende arrangement. (Om de gedachten te bepalen, beschrijven wij het hier voor enkellengte reële getallen met drijvende binaire komma). Het getal G worde voorgesteld door

$$G = b.2^m \quad \tfrac{1}{2} \leqslant |b| < 1;$$

(tevens geldt $|m| < 8192 = 2^{13}$, een beperking, die opgelegd is, omdat voor m niet meer cijferposities ter beschikking zijn gesteld.) Elk getal wordt in het geheugen geborgen op twee opeenvolgende adressen en wel, als G staat op h en h + 1, dan is

$$\{h\} = b \text{ en } [h + 1] = (m + 2^{13}).\ 2^{15} + h.$$

Het eerste adres bevat dus de breuk, het tweede adres bevat een assemblage van de macht (waarvoor de volle woordlengte echt niet nodig is) en de plaats in het geheugen! De subroutine-aanroep wordt nu voorafgegaan door de opdracht 10/h + 1: deze plaatst de assemblage in S; in de subroutine wordt nu de assemblage ontrafeld, de macht m - het ene gedeelte van het getal - wordt geisoleerd van de rest, die de aanduiding bevat (nl. h) waar de breuk - het nog ontbrekende gedeelte - gevonden kan worden. Alle "schrijf"-subroutines (d.w.z. uit de quasi-accumulator naar het geheugen) vormen weer het complete assemblage-resultaat en schrijven dat weg; een consequentie van dit systeem is, dat alle werkruimtes moeten worden "voorbereid": op elke plaats h + 1 moet in de a-positie h staan, willen wij aan de schrijfsubroutine mee kunnen geven, dat op h en h + 1 geschoven moet worden. De quasi-accumulator bestaat uit twee adressen: één voor de breuk b, en één voor de macht m ($+2^{13}$); deze macht is hier niet in assemblage 15 plaatsen naar links geschreven. Hier zien we een voorbeeld, hoe de representatie in de quasi-accumulator van die in de rest van het geheugen,

wat kan afwijken.
Uit de beschrijving van het betroffen subroutine-aggregaat (code Rd3) nemen wij het volgende lijstje over. (Hier is de inhoud van de quasi-accumulator met (R) aangeduid; (h, h + 1) is het getal op adressen h en h + 1).

	24	6	X0	Inloopscombinatie
	4	159	X24	
=)	10	"h+1"		Functie: (R)+(h,h+1) ⇒ (R)
	7		X6 =)	
=)	10	"h+1"		Functie: (R)-(h,h+1) ⇒ (R)
	7	1	X6 =)	
=)	10	"h+1"		Functie: (h,h+1) ⇒ (R)
	7	2	X6 =)	
=)	10	"h+1"		Functie: -(h,h+1) ⇒ (R)
	7	3	X6 =)	
=)	10	"h+1"		Functie: (R) ⇒ (h,h+1)
	7	4	X6 =)	
=)	10	"h+1"		Functie: -(R) ⇒ (h,h+1)
	7	5	X6 =)	

Tot zover uit de beschrijving van Rd3. Verdere subroutines verzorgen het vermenigvuldigen en delen van (R), het verheffen van |(R)| tot een willekeurige (gebroken) macht, speciaal - uit tijdsoverweging - het trekken van de tweedemachtswortel uit |(R)|, en verder subroutines voor het invoeren en het typen van drijvende komma getallen. D.w.z. bij de invoer wordt niet alleen de constructie van de assemblage verzorgd, maar ook de overgang van "decimaal drijvend" naar "binair drijvend"; tijdens het uittypen vindt het omgekeerde proces plaats. Omtrent de invoer staat in de beschrijving:

"Het bandlezen

Drijvende komma getallen kunnen decimaal geponst worden in de representatie $T.10^t$. Aan de ongelijkheid $0,1 \leq |T| < 1$ moet voldaan zijn! Men ponst eerst T (als breuk), dan t (als geheel getal). Alle omrekeningen en assemblage wordt door Rd3 verzorgd, mits het reeksje (dat uit een enkel tweetal mag

bestaan!)

1. voorafgegaan wordt door de "Aankondigingscombinatie"
   F 19 X9
2. gevolgd wordt door de "Afsluitingscombinatie"
   F 16 X9

Tussen deze twee controlecombinaties mogen <u>uitsluitend</u> decimaal drijvende getallen, als boven omschreven, geponst worden, en <u>niets anders</u> (dus geen normale moleculen, geen adres-indicaties etc.). Tussen getallen wordt wel X geskipt.

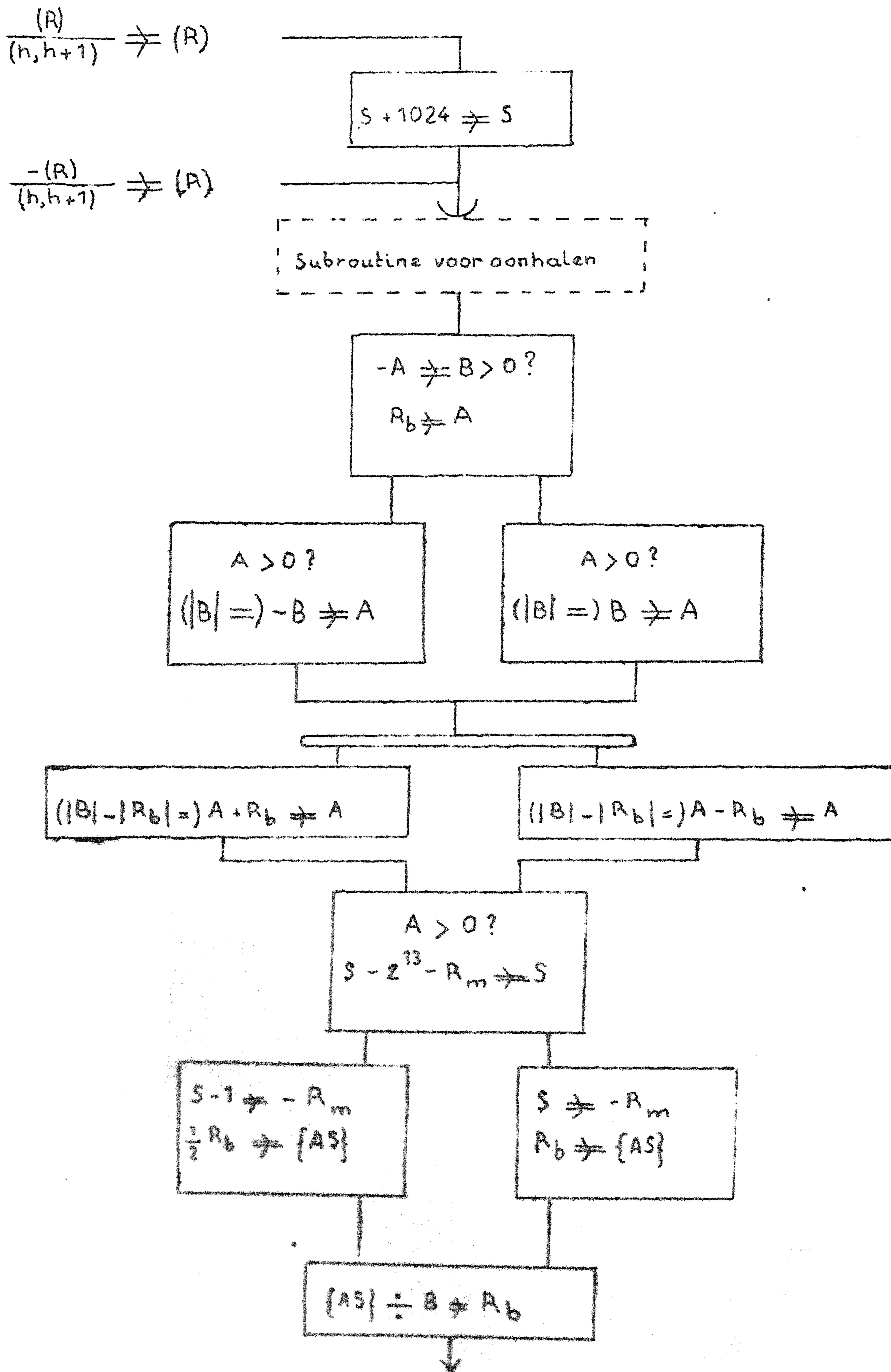
Het effect van de twee controlecombinaties is een tijdelijke wijziging van de pentaden-assemblage. Zij veranderen niets aan de stand (schrijvend of controlerend) van het invoerprogramma. Het ophogen van de wegbergopdracht gaat gewoon door. Wel moet erop gelet worden, dat slechts complete tweetallen worden ingevoerd: de tijdelijk gewijzigde interpretatie van de band strekt zich uit over een even aantal adressen." Tot zover de beschrijving van Rd3.

Ter illustratie een stukje programma, dat de som a+b-c (die staan in het begin van kanaal A0) op het volgend tweetal adressen plaatst. We nemen aan, dat vlak van te voren het programma direct gewerkt heeft, en beginnen dus met de inloopscombinatie. De lezer realisere zich, dat het onderstaande stukje programma onafhankelijk luidt t.o.v. a-b-positie!

```
     24    6  X0                  Inloopscombinatie.
      4  159  X24
     10    1  A0                  (0A0,1A0) = a
      7    2  X6 =) . . . . .  .a ⇒ (R)
=)   10    3  A0                  (2A0,3A0) = b
      7       X6 =) . . . . .  .a+b = (R) + b ⇒ (R)
=)   10    5  A0                  (4A0,5A0) = c
      7    1  X6 =) . . . . .  .a+b-c = (R) - c ⇒ (R)
=)   10    7  A0                  (6A0,7A0) = d
      7    4  X6 =) . . . . .  .a+b-c = (R) ⇒ d
=)   ..........
```

De inloopscombinatie zet op de plaats van de gemeenschappelijke koppelopdracht (159 X 24) de sprong, die verwijst naar de opdracht 10 1 A0; als zodanig wordt deze opdracht niet gehoorzaamd: voordien is er 1 bij het adres opgeteld, en de eerste keer springt de besturing naar de opdracht 10 3 A0, etc. De zichzelf ophogende koppelopdracht heeft de functie van de opdrachtteller overgenomen.

Het aanhalen van een getal uit het geheugen is een element, dat in vele subroutines van het aggregaat voorkomt. Dit wordt verwezenlijkt door een interne subroutine - om plaatsruimte te besparen - die terugkomt met de macht $m + 2^{13}$ in S en in A de breuk met hetzelfde teken als in het geheugen, indien de "sub-subroutine" de ongewijzigde assemblage heeft meegekregen, edoch van teken gewisseld, als deze assemblage met $1024 = 2^{10}$ was vermeerderd.
Dit ter explicatie van het blokschema van de subroutine voor de delingen die wij ter illustratie laten volgen. De werkruimtes van de quasi-accumulator zijn R_b en R_m genoemd; voorts komt er een werkruimte B bij te pas.



naar zelfophogende koppelopdracht

Het blokschema voor de drijvende komma deling.

Syllabus No. 17 van de cursus 1955-'56:
Programmeren voor automatische rekenmachines
Onder leiding van
Prof. Dr. Ir. A. van Wijngaarden en de Heer E.W. Dijkstra

SUPERPROGRAMMAS

Tenslotte komen wij dan tot de behandeling van superprogrammas, wier taak het is om andere programmas te onderzoeken, te interpreteren, enz. Deze taak kan op allerlei wijzen worden verwezenlijkt. Een van de allereenvoudigste taken, welke zo'n programma kan opknappen is een bepaald gedeelte van de geheugeninhoud uit te typen of te ponsen. Op deze wijze kan een nette copie van een programma gemaakt worden, een programma, dat overigens zelf in het geheel niet aan bod komt. Het meest praktische nut van een dergelijk uitponsprogramma is, dat het een ponsband kan afleveren die heel eenvoudig door het invoerprogramma kan worden gelezen. Een programma inclusief constanten e.d. in het geheugen is immers niets anders dan een aantal cijferrijen, woorden, die wat het inbrengen betreft gerust allen als getal kunnen worden beschouwd. Pas wanneer het programma moet werken wordt het van belang te weten, wat opdrachten en wat getallen zijn, De programmeur moet het natuurlijk ook weten of beter, het is het enige wat hij weet. Hij heeft er weet noch zorg van hoe bijv. een opdrachtenkoppel er binair geschreven uitziet. Hij schrijft en ponst zijn programma volgens een code, die voor hem gemakkelijk is en heeft een invoerprogramma ter beschikking, dat deze code ontrafelt en de juiste cijferrijen in het geheugen plaatst. De taak van dit invoerprogramma is dan echter niet eenvoudig en hoe gemakkelijker en flexibeler de code is, des te ingewikkelder en des te trager is het invoerprogramma. Is het programma echter eenmaal ingelezen en op zijn deugdelijkheid getest, dan kan het uitponsprogramma een band maken, waarop de door het programma met toebehoren beslagen geheugeninhoud als rij van woorden in het tweetallig stelsel wordt gegeven. Voor de ARRA met 30 bits per woord worden dus 6 pentades van 5 cijfers per woord geponst. Heeft men later dat programma weer nodig, dan kan een speciaal hoekje van het invoerprogramma zo'n biband zeer snel weer inlezen. Het hoeft nl. alleen telkens zes pentades van de band te lezen, die aan elkaar te rijgen en weg te bergen. Let er wel op, dat het programma noodzakelijkerwijs op precies dezelfde plaats in het geheugen dient te worden geborgen. Bij niet zeer snelle machines is deze wijze van werken zeer aanbevelenswaardig. Het is overigens wel zielig, dat de met zorg geponste banden van het

programma slechts één maal behoeven te worden gebruikt en dan zodra de biband wordt gemaakt, weggegooid kunnen worden.

Deze allereerste taak van de nu beschouwde programmas is wel zeer summier. Immers het feit, dat het uitgeponste een programma is, komt in het geheel niet ter sprake. Het binaire uitponsprogramma kan inderdaad ook heel geschikt gebruikt worden om grote hoeveelheden getallenmateriaal uit het geheugen te redden voor later gebruik. Een speciale rol wordt vervuld, als men bij een programma, dat zeer lang werk heeft, bijv. bij het oplossen van partiele differentiaalvergelijkingen zo nu en dan de bereikte toestand laat uitponsen om, wanneer onverhoopt een fout zou ontstaan, niet gedwongen te zijn geheel opnieuw te beginnen doch slechts bij de toestand welke het laatste was vastgelegd.

Het genoemde invoerprogramma, dat wij trouwens al lang kenden is ook zo'n programma, dat iets met een ander programma "doet" behalve het alleen inlezen. Het vertaalt het nl. uit de programmeurscode in de machinecode, die een beetje of heel veel er van afwijkt. Wat dat betreft is het al een stuk intelligenter dan het binaire uitvoerprogramma, dat alleen amorphe cijferrijen kent. Wij hebben indertijd al een en ander over het invoerprogramma verteld, doch zullen nu over gaan tot enkele andere mogelijkheden om aan te tonen, dat men veel meer kan doen dan tot nu toe behandeld was. Voorop staat, dat in de eerste plaats het doel is de programmeur het leven gemakkelijk te maken, zowel wat betreft het programmeren als ook wat betreft het ponsen.

Wat dit laatste betreft kan men al of niet gebruik maken van speciale apparatuur bij het ponsen. Hoe slim het invoerprogramma ook is, het heeft toch in ieder geval de minimaal benodigde hoeveelheid informatie nodig en daarvoor is een zeker aantal pentades op de band nodig. Dit impliceert een zekere hoeveelheid aanslagen op een gewone ponsmachine, waarop alleen de pentades geponst kunnen worden. Heeft men echter een speciaal gemaakte ponsmachine ter beschikking, die op indrukken van speciale toetsen reageert met het ponsen van meerdere pentades, dan kan men ponsarbeid sparen, iets wat met programmeren niet valt te bereiken. Men heeft dan als het ware een steno ponsmachine. De ARRA heeft een dergelijke speciale ponsmachine niet, maar toetsen van veelvoorkomende combinaties als bijv. 24XO (skip) zouden wel enig nut hebben. Bij de ARMAC is een speciale ponsmachine voorzien, die toetsen heeft op zo geschikt mogelijk gekozen plaatsen en waarbij ook deze "roffeltoetsen" zijn gedacht. Een en ander resulteert in een enorme daling van de ponsarbeid, bijv.

tot 50%.

Een heel andere bron van gemak is daarentegen te vinden in de vorm van het invoerprogramma zelf, eventueel dan nog in samenwerking met speciale roffels via de bandponser geintroduceerd. Voorbeelden van speciale faciliteiten zijn het zg. drijvend programmeren, het ladderen, Utility-programmas.

Het drijvend programmeren (M.V. WILKES, The use of a "floating address" system for orders in an automatic digital computer, Proc. Camb. Phil. Soc. 49 (1953), Part 1, 84) is een methode om het vervelende administratieve werk aan programmas drastisch te beperken. Bij de behandelde programmas werden aan de constanten, variabelen, en variabele opdrachten steeds bepaalde adressen toegewezen, bijv. x = (13 P6) of zo, terwijl wij er, als optimaal programmeren niet nagestreefd wordt, in werkelijkheid helemaal niet in geinteresseerd zijn, waar die x nu precies in het geheugen staat. Wel moet de x op een of andere wijze gekarakteriseerd worden, bijv. door het getal .6. Alle opdrachten die met x opereren, dus het adres van x als numeriek gedeelte in de opdracht hebben kunnen dus gekarakteriseerd worden door daar 6 voor het adres te schrijven. Waar de opdrachten zelf staan kan ons dat helemaal niet schelen, alleen de adressen van de variabele opdrachten en die van de plaatsen in het programma waarnaar wordt gesprongen zijn essentieel, maar deze kunnen ook worden gekarakteriseerd door een beperkt aantal getallen. Wij zullen ons verder een machine voorstellen, waarin slechts één opdracht in een woord gaat, want bij het systeem van de ARRA met zijn opdrachtenkoppels rijzen een aantal complicaties. Het programma wordt aan één stuk achter elkaar geschreven zonder dat men zich behoeft te bekommeren over het nummeren van de opdrachten. Achteraf mag men hier en daar gewone opdrachten inlassen. De adressen van de constanten enz. geeft men nummers, niet noodzakelijkerwijs in volgorde. Wel moet men hier en daar natuurlijk die constanten etc. vermelden. Er is allereerst een openingsletter, bijv. P te gebruiken met de volgende betekenis. Als ergens staat P7 dan is dit een aanwijzing voor het invoerprogramma, dat het adres waarop het nu gaat invullen (dat de programmeur niet weet, maar het invoerprogramma wel!) verderop zal aangeduid worden met het cijfer 7. Het invoerprogramma kan daarvan een notitie maken in een drijvend-adresboekje en als verder als drijvend adres 7 wordt gebruikt daarvoor het echt adres substitueren. Tot zover is alles eenvoudig. Als een grootheid niet meer nodig is, dan kan zijn drijvend adres opnieuw gebruikt worden. Er ontstaat een moeilijkheid als het adres behorend bij het drijvend adres nog

niet is gespecificeerd als het wordt gebruikt. Dit hoeft niet voor te komen bij werkruimten, constanten en variabelen daar men daarvoor altijd tijdig adressen met P-combinatie kan reserveren, maar is nu en dan praktisch onvermijdelijk bij variabele opdrachten en bij sprongopdrachten die naar voren springen. Een manier om deze moeilijkheid op te lossen werkt als volgt. Als men verwijst naar een nog niet vastgelegd drijvend adres geeft men dit aan met een openingsletter, bijv. Q. Verder kan in het drijvend-adresboekje, dat het invoerprogramma bijhoudt bij ieder adres natuurlijk ook nog een functiegedeelte F worden onthouden. Dit zal een functiegedeelte zijn, dat nooit door de programmeur met een drijvend adres voorzien kan of in elk geval mag worden gebruikt. Zulk een functiegedeelte is altijd wel te vinden, bijv. een niet bestaande opdracht , een schuifopdracht, een stopopdracht of iets dergelijks. Bij de aanvang van het invoerprogramma worden in het adresboekje deze functiegedeelten F vast ingevuld. Een P-combinatie vult er dan het adres bij in. Een Q-comninatie zal dan als volgt werken. Hij zal de combinatie, die in het adresboekje bij dat drijvend adres staat plaatsen op het in te vullen adres en in het boekje noteren het functiegedeelte dat op de band is gespecificeerd plus het echte adres waar geborgen moest worden. Dit kan zo een aantal malen gebeuren. De functie van de P-combinatie wordt nu nader zó gedefinieerd, dat hij eerst nakijkt of in het adresboekje het "onbestaanbare" functiegedeelte F is gespecificeerd. Zo ja, dan wordt het vast te leggen adres toegevoegd en de zaak is klaar. Zo neen, dan was het drijvende adres al een of meer keren met een Q-opening gebruikt en moet op die plaats(en) het echte adres nog worden ingevuld. Waar dat de laatste keer gebeurd is, is te zien aan het adres in het boekje. De inhoud van dat adres wordt in het boekje geplaatst na het functiegedeelte dat in het boekje stond te hebben gered. Dit functiegedeelte wordt geassembleerd met het echte adres en op zijn goede plaats opgeborgen. Hierna begint de analyse weer opnieuw. Men zal zien als men nagaat wat er gebeurt, dat alle adressen, waar de Q-combinatie is gebruikt bijgewerkt worden alvorens het programma verder gaat.

Een schema van de organisatie luidt bijv.:

Op band	In adresboekje op drijvend adres 9:	Werkelijke adressen	Ingevuld in geheugen
-	F...	-	-
-		-	-
-		-	-
f_1Q9	f_1a_1	a_1	F...
-		-	-
-		-	-
-		-	-
-		-	-
f_2Q9	f_2a_2	a_2	f_1a_1
-		-	-
-		-	-
-		-	-
f_3Q9	f_3a_3	a_3	f_2a_2
-		-	-
-		-	-
-		-	-
P9x		a	x
	f_2a_2	a_3	f_3a
	f_1a_1	a_2	f_2a
	F...	a_1	f_1a
-	Fa	a+1	-
-		-	-
-		-	-
-		-	-

Kennelijke voordelen van het drijvend adresseren zijn gelegen in kortheid en flexibiliteit. Omdat het aantal drijvende adressen tamelijk klein is, en de drijvende adressen dus zeer kort zijn, wordt ook nog aanzienlijke ponsarbeid bespaard. Veranderingen aan het programma zijn zeer eenvoudig aan te brengen, men last eenvoudig in of schrapt opdrachten weg zonder dat omnummering noodzakelijk is. Een nadeel is, dat optimum programmering illusoir wordt en voorts dat men niet weet, waar het programma in het geheugen precies staat. Hieraan is evenwel gemakkelijk tegemoet te komen. Het drijvend invoerprogramma kan bijv. al bij iedere P-indicatie het werkelijke adres uittypen en eventueel kan men na de invoer even het programma-uittypprogramma alles laten uittypen. Men heeft dan een nette copie.

Het is gemakkelijk kleine accessoires in te bouwen. Bijv. komt het nogal eens voor, dat een sprong gemaakt moet worden naar een adres dat een zeer klein aantal plaatsen ver-

der op staat. In plaats van de Q-faciliteit te gebruiken kan men dan bijv. een R-faciliteit benutten, waarbij R3 betekent 3 plaatsen verder. Dit spaart arbeid en tijd.

Nauw verwant met het drijvend adresseren en al of niet in samenhang ermee te gebruiken is het adresboekprogramma. Laat het programma bestaan uit een groot aantal subroutines, die elkaar gedeeltelijk kunnen gebruiken en een hoofdprogramma. Iedere subroutineband sluit met een controlecombinatie, welke zorgt, dat het adres dat volgt op het zo juist ingevulde, in volgorde genoteerd wordt in een adresboek. De ingangen van de subroutines kunnen dan worden gekenmerkt door het nummer van invoer te vermelden. Alleen dient er zorg voor gedragen te worden, dat een subroutine A, die gebruikt wordt door subroutine B ook voor B wordt ingelezen. In het bijzonder wordt het hoofdprogramma het allerlaatste ingelezen. Het aanduiden van zo'n adres wordt weer door een speciale letter, bijv. S onderscheiden van de aanduiding van een normaal adres.

Een geheel andere faciliteit is bijv. het ladderen. Bij langzame machines zelfs, als geen optimum programmering wordt of kan worden toegepast, kan, zoals gezien, desniettemin de snelheid van een programma enorm worden opgevoerd door het zg. strekken, d.w.z. het volledig uitschrijven van het programma in plaats van het gebruik van cycli. Dit heeft dan ten gevolge, dat gedeelten van het programma bestaan uit opdrachtengroepen die herhaald worden en slechts een regelmatige verandering in hun adressen vertonen. Een voorbeeld hiervan hebben wij gezien in syllabus No. 12. In dat geval kan een ladder-invoerprogramma benut worden, dat uit de indicaties: "hier is een opdracht, schrijf deze <u>k</u> maal neer om de <u>m</u> plaatsen telkenmale het adresgedeelte met <u>n</u> verhogend of iets algemener met de volgende adresgedeelten......., en sta dan weer klaar om op de eerste vrije plaats weer in te vullen" voldoende weet om de opgedragen taak te vervullen. Niet alleen is dit veel eenvoudiger voor de prgrammeur, maar het spaart bovendien veel ponstijd en invoertijd.

Een andere mogelijkheid tenslotte om tijd te sparen ligt in het gebruik van utilityprogrammas. Het gewone invoerprogramma kan bijv. gehele getallen en breuken inbrengen en moet rekening houden met verschillend aantal geponste cijfers door onderdrukking van niet significante nullen. Dit maakt enerzijds nodig, dat sluit- of openingsletters worden geponst ter inleiding van de getallen, anderzijds veroorzaakt het, dat het in-

voerprogramma noodzakelijkerwijs traag is, omdat het op allerlei eventualiteiten bedacht moet zijn. Moet men nu een grote hoeveelheid getallenmateriaal (bijv. een matrix) invoeren, bestaande uit louter breuken van 6 decimalen bijv., dan kan een utilityprogramma voor banden van 6 decimalen een band begrijpen als bijv.

+.123456 +.324500 -. 881123 +. 710247 -. 056643 +. 100000, blank.

Een dergelijk programma kan gestrekt zijn nadat het aantal cijfers vastligt en aanzienlijk sneller zijn dan het normale invoerprogramma en de ponsarbeid is ook aanzienlijk geringer.

Syllabus No. 18 van de cursus 1955-'56:
Programmeren voor automatische rekenmachines
onder leiding van
Prof.Dr Ir A. van Wijngaarden en de heer E.W. Dijkstra

SUPERPROGRAMMA'S

De functies van de superprogramma's, die tot nu toe behandeld waren, missen nog het meest fundamentele facet, dat ze kunnen hebben. Zij hanteren nl. nog steeds het objectprogramma als een stuk dood materiaal, maar begrijpen nog niet het wezenlijk levende karakter van het objectprogramma, nl. dat dit zelf, als het aan bod kwam, zou gaan manipuleren met getallen of zelfs met opdrachten, dat de opdrachten ervan niet in volgorde zouden worden afgehandeld, maar dat er al of niet conditionele sprongen in voorkomen, dat er cycli aanwezig zijn, enz. Wel waren de behandelde superprogramma's eventueel in staat een vertaling te leveren van een willekeurige code, de programmeurscode, in de machinecode, waaraan nu eenmaal niet valt te tornen, omdat zij de axiomatiek van de gegeven machine is. De vraag rijst - en wordt bevestigend beantwoord - of er superprogramma's kunnen worden ontwikkeld, die het objectprogramma wezenlijk kunnen interpreteren, d.w.z. met inachtneming van de werking van het objectprogramma. Zulke superprogramma's heten interpreterende programma's. Het objectprogramma kan in de machinecode in de machine aanwezig zijn. In dat geval zou het dus zonder meer aan het werk gezet kunnen worden. Het nut van het interpreterende programma zou dus dubieus kunnen lijken en ook inderdaad zijn, als het zich alleen beperkte tot het nabootsen van wat het objectprogramma zou doen, als het aanbod was. Het interpreterende programma kan echter behalve dit meer doen, bijv. uittypen welke opdrachten van het objectprogramma gehoorzaamd worden, bijv. het adres in de opdracht zelf, en voorts welke getallen gemanipuleerd worden, wat de gewijzigde inhouden van de registers van het rekenorgaan zijn, wat de conditie is, enz. Een dergelijk programma is een prachtig hulpmiddel om fouten in een gemaakt programma op te sporen. Het is noodzakelijkerwijs langzaam, zelfs zeer langzaam, maar kan op allerlei wijzen geweldig worden versneld. Het zou nl. ook alle gebruikte subroutines gaan interpreteren, wat kennelijk niet de bedoeling is, ook de werking van een semi-interpreterende subroutine, zoals in syllabus 16 behandeld is, terwijl men daar met veel minder informatie zou kunnen volstaan. Door bijv. van te voren te vertellen aan het interpreterende programma, in welke gedeelten van het geheugen het werkelijk te interpreteren objectprogramma staat, kan

het volstaan met het interpreteren van dat gedeelte, terwijl het de subroutines vrij laat lopen (bij de koppelopdracht pikt het dan de draad weer op). Door de subroutines dan in klassen te verdelen, bijv. zij die hun resultaat in S afleveren en zij die het in een pseudo-accumulator afleveren (zoals de semi-interpreterende subroutines), kan het interpreterende programma zich na het beëindigen van de subroutines dan beperken tot mededelingen aangaande de inhoud van de werkelijk interessante registers (bijv. S of de pseudo-accumulator).

Het samenstel, machine plus interpreterend programma, van deze soort is dan in wezen te beschouwen als een nieuwe machine, een pseudomachine, die handelt als de echte machine, maar daarbij nog voortdurend verslag uitbrengt van zijn handelingen als een neurotische patiënt aan de psychiater (nl. de programmeur).

Dat een dergelijk programma te maken is, zal wel zonder meer duidelijk zijn op grond van wat tot nu toe behandeld is. Het interpreterende programma moet zelf een pseudomachine bijhouden in de vorm van een aantal adressen gebruikt als pseudo-opdrachtteller, pseudorekenregisters, pseudocondities, enz. Er rijzen geen principiële moeilijkheden.

Naar aanleiding van de analogie patiënt-psychiater merken wij nog op, dat wij ook het objectprogramma alleen als patiënt kunnen zien. Het interpreterende programma is dan psychiater en het uitgetypte diens aantekeningen. Een hoogst vermakelijk experiment, dat wij jaren geleden op de ARRA pleegden, is het interpreterende programma in duplo in de machine brengen, daarbij elk specimen als objectprogramma aan het andere aanwijzend. Laat men dan een van beide werken, dan verkrijgt men uitgetypt een verslag van psychiater A, die psychiater B's handelingen analyseert, als deze (B) bezig is A's handelingen te onderzoeken. Het bleek daarbij, dat het interpreteren van dit verslag de programmeur ook spoedig rijp maakt voor de psychiater.

Een tweede soort interpreterend programma, dat veel nuttiger is, werkt op een objectprogramma, dat in de machine aanwezig is, maar in een andere code dan de machinecode is gesteld. Het samenstel machine plus interpreterend programma van deze soort is te beschouwen als een pseudomachine van een geheel ander soort dan de oorspronkelijke, bijv. een machine die zonder meer met complexe getallen kan werken of met drijvende komma. Het stelt ons allereerst in staat om een programma met complexe getallen net zo eenvoudig te programmeren als een gewoon programma. Daarbij dient te worden bedacht, dat zelfs in een programma, dat veelvuldig

werkt met drijvende complexe getallen toch nog een aanzienlijk gedeelte, nl. de gehele administratie gewoon kan geschieden. Het is een groot tijdverlies ook de administratie te laten interpreteren. Daarom is het van belang en mogelijk om gestadig om te schakelen tussen interpreteren en niet-interpreteren. De ene omschakeling, nl. die tussen niet-interpreteren en interpreteren is eenvoudig. Het objectprogramma, dat als subject werkt, heeft dan eenvoudig een passende sprong naar het interpreterende programma, te interpreteren als: " ga me nu verder maar interpreteren". De ongekeerde overgang is principieel moeilijker. Het vereist, dat één bepaalde pseudo-opdracht in het objectprogramma door het interpreterende programma herkend en gehoorzaamd wordt, te interpreteren als: "ik wil nu weer zelf werken". Zulk een pseudo-opdracht mag dan natuurlijk niet tevens toegelaten zijn als opdracht in de te interpreteren code. Deze overgang is principieel dezelfde als optreedt bij de invoer als een bepaalde controlecombinatie op de band het invoerprogramma, dat overigens de informatie op de band manipuleert, dwingt om de gelezen opdracht uit te voeren, zodat bijv. het ingelegen programma zelf kan starten.

Een aardige toepassing van dit soort interpreterende programma's rijst bij het ontwerpen van een nieuwe machine als men zelf beschikt over een andere machine, die in een andere code werkt. Men kan deze nieuwe machine nabootsen door een interpreterend programma op de bestaande machine en daarmede dan de ontworpen programma's en subroutines voor de nieuwe machine vast controleren.

Een derde soort interpreterende programma's werkt op programma's die zich nog niet in de machine bevinden, maar werkt op programma's die zich op de band bevinden. Er zijn weer twee typen. Het eerste type maakt uit de informatie op de band een machineprogramma en plaatst dat in de machine. Het gewone invoerprogramma is het eenvoudigste voorbeeld van deze genererende programma's. Door uitbreiding van het invoerprogramma (ladderen, drijvende adressen hebben wij al behandeld) kunnen wij de code op de band, dus de programmeercode willekeurig compact maken. Bijvoorbeeld kunnen wij het algebraïsche formules laten uitrafelen, recursiebetrekkingen laten vertalen, enz. Deze gevormde programma's zullen in het algemeen niet bijzonder elegant, doch eerder wat houterig zijn, maar het voornaamste is, dat ze snel programmeren met kleine kans op fouten. Het aldus gemaakte programma kan uitgeponst worden en staat dus verder op normale wijze ter beschikking.

Het tweede type van de derde soort, het compilerende programma, maakt niet van de bandinformatie een programma in de machine, maar interpreteert direct de bandinformatie en voert de gevraagde handelingen uit.

In het algemeen kan men zeggen, dat een schier eindeloze hoeveelheid "kennis" in de superprogramma's kan worden opgeslagen, zodat het converseren met de machine meer het karakter krijgt van het praten tot een collega in plaats van tegen een imbeciele slaaf. Hun toepassing vereist echter grote snelheid en geheugencapaciteit van de machine.