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

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

MC SYLLABUS

7.2

MC SYLLABUS

7.2

COLLOQUIUM MODERNE REKENMACHINES

DEEL 2

MATHEMATISCH CENTRUM AMSTERDAM

1969

A. van Wijngaarden
The state of computer circuits containing memory elements 115

J. Berghuis
Enige beschouwingen over iteratieve processen en niet lineaire transformaties 132

J. Berghuis
Enige beschouwingen over iteratieve processen en niet lineaire transformaties II 142

E.W. Dijkstra
Multi-lengte programmering 160

FERRIETKERNEN

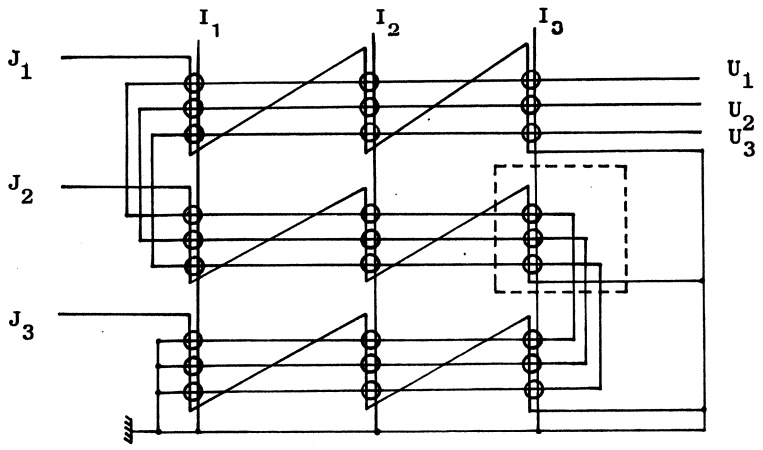
C.S. Scholten

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 30 oktober 1954 te Amsterdam.

Nu gedurende de laatste jaren de productie op grote schaal van ferrietkernen met rechthoekige hystereselus mogelijk is gebleken, is de belangstelling ervoor in de kringen van hen, die zich met constructie van electronische rekenmachines bezig houden sterk toegenomen, te meer daar deze kernen tot op vrij hoge frequenties bedreven kunnen worden.

Als toepassingsmogelijkheid van deze kernen komt uiteraard in de eerste plaats in aanmerking het macroscopisch geheugen. Inderdaad steekt een ferrietkerngeheugen met een (parallel) access-tijd van ca. 1 μsec. zeer gunstig af bij alle andere tot dus verre in gebruik zijnde geheugentypen. Een nadeel is, dat door het uitlezen van een kern de daarin aanwezige informatie verloren gaat, zodat voorzorgen getroffen zullen moeten worden deze informatie na het uitlezen weer terug te schrijven.

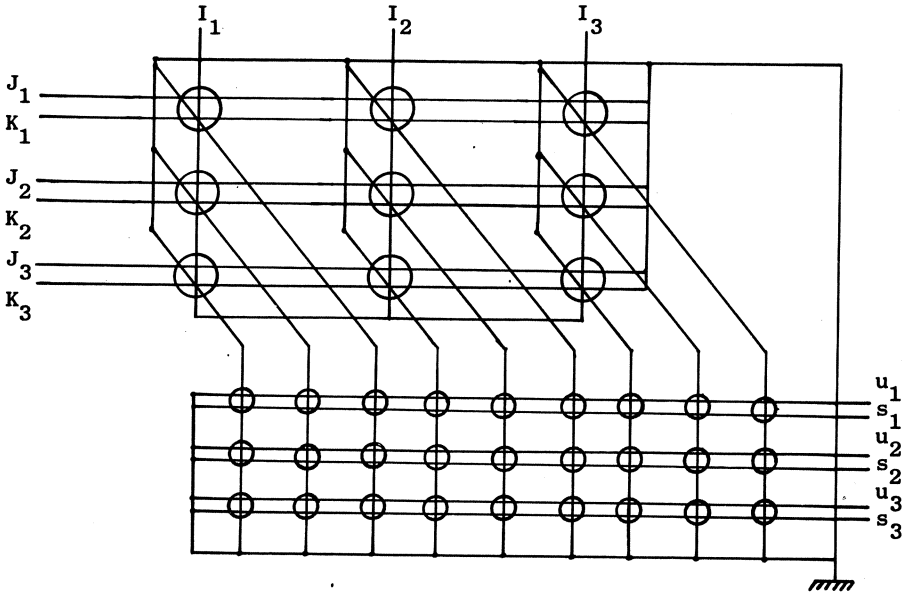
Een tweede moeilijkheid vormt de selectie. Om niet in een te groot aantal buizen te vervallen lijkt matrixselectie aangewezen. Deze zou voor een geheugen van 9 woorden van 3 digits er als volgt uitzien.



Selectie van het omhokte woord zou dan plaats vinden door bekrachtiging van I_3 en J_2, de inhoud zou parallel verschijnen op U_1, U_2 en U_3. Dit systeem heeft echter het grote bezwaar, dat blijkbaar verondersteld wordt, dat de geselecteerde kernen overgaan op de combinatie van stromen via een I-lijn en een J-lijn, doch niet op elk dezer stromen afzonderlijk.

Het inlezen zou zelfs nog critischer zijn, daar dan nog een derde stel ingangslijnen (wat vlechtwerk door de kernen betreft slechts een duplicering van de lijnen U_1, U_2, U_3) zou moeten worden beschouwd om aan te geven op welke plaats in het geselecteerde woord een één moet worden geschreven. Dan hoort een kern dus te reageren op de som van 3 stromen en niet op een willekeurige combinatie van 2 daarvan. Bovendien heeft dit systeem het nadeel, dat de storing van alle niet geselecteerde kernen mede op de uitgang verschijnt.

Het volgende schema, waarbij elk woord geselecteerd wordt door een selectiekern is veel minder critisch.



Door alle I-wikkelingen loopt normaal stroom, zodat alle selectiekernen in rust in dezelfde toestand, zeggen we toestand 1, verkeren. Selectie van een woord geschiedt nu door van de bijbehorende selectiekern eerst de bias-stroom op de betreffende I-lijn weg te nemen en vervolgens de bijbehorende J-lijn te bekrachtigen, waardoor de bedoelde selectiekern overgaat naar toestand 0.

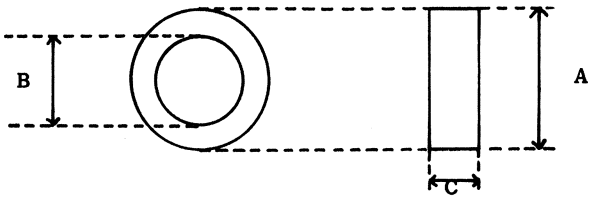
De eisen aan de bias-stroom zijn nu slechts dat deze zo groot is, dat de niet geselecteerde kernen in toestand 1 blijven en geen last hebben van eventuele bekrachtiging van hun J-lijn.

De omslag 1-0 van de selectiekern drijft nu alle digitkernen van het bijbehorende woord in een bepaalde toestand (0) en de inhoud verschijnt op de uitgangen U_1, U_2 en U_3. De informatie, die vervat was in de uitgelezen kernen is nu verdwenen en dient te worden teruggeschreven. Veronderstellen we dat deze informatie is opgevangen in enige flipflops, dan worden de terugschrijflijnen S_1, S_2, S_3 al of niet bekrachtigd, al naar gelang van de stand van de bijbehorende flipflops. De stroom door S_1, S_2, S_3 heeft de polariteit om de kernen in de toestand 1 te drijven, zonder nochtans daartoe groot genoeg te

zijn. Via een der lijnen K_1, K_2 en K_3 wordt nu de uitgekozen selectiekern weer in de toestand 1 gebracht, echter met een snelheid, die zoveel lager is dan die waarmee hij tevoren in de toestand 0 werd gedreven, dat de resulterende stroom in de doorkoppelwikkelingen naar de digitkernen slechts in combinatie met bekrachtiging van een S-lijn voldoende is om de bijbehorende digitkern in toestand 1 te brengen, waarmee dan de schrijfoperatie voltooid is. Het enige onderdeel van de bewerking, dat hier nog critisch is, is het schrijven waar dus een kern moet omslaan op een combinatie van twee stromen en niet op ieder daarvan afzonderlijk. Aan het slot zal nog een suggestie worden gedaan om ook dit schoonheidsfoutje te elimineren.

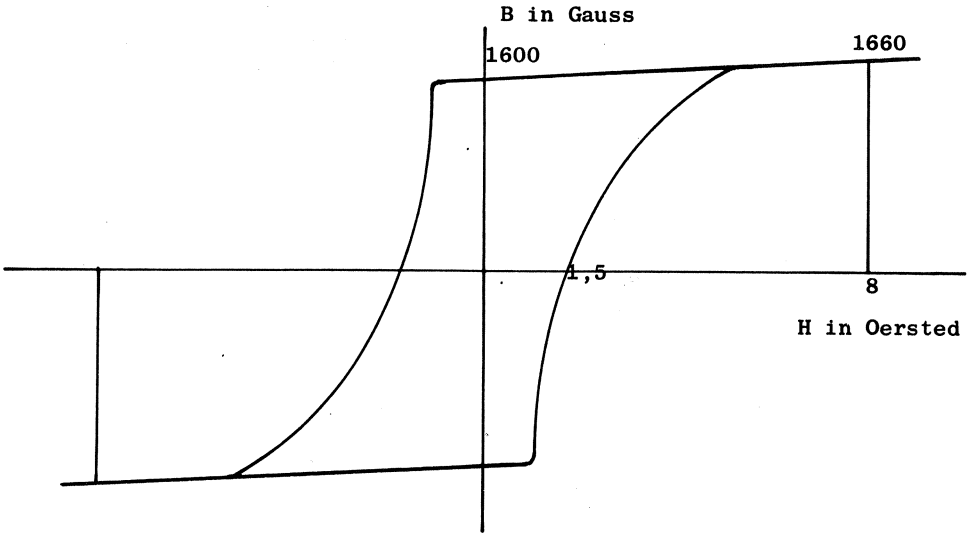
Al deze beschouwingen hebben er toe geleid, dat o.a. in Cambridge, Engeland, en de laatste tijd ook op het Mathematisch Centrum te Amsterdam proeven met ferrietkernen worden genomen en wel hoofdzakelijk gebaseerd op het laatste principeschema.

De toroidale ferrietkernen die hiertoe gebruikt worden, hebben de volgende afmetingen:

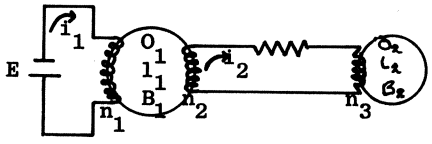


	A	B	C	
Selectiekernen	10.5	7.3	3.3	mm
Digitkernen	2.0	1.4	0.6	mm

De volgende grafiek geeft een indruk van de hysteresselus van het gebruikte materiaal.



Tekenen we nu het schema van een selectiekern met een digitkern en veronderstellen we, dat op de primaire winding van de selectiekern een spanning E wordt aangesloten dan luiden de vergelijkingen voor de primaire resp. sec. keten:



$$E = n_1 O_1 \frac{dB_1}{dt} = 4\pi \mu_0 \mu_r \frac{O_1}{l_1} \left(n_1^2 \frac{di_1}{dt} - n_1 n_2 \frac{di_2}{dt}\right)$$

$$O = i_2 R - n_2 O_1 \frac{dB_1}{dt} + n_3 O_2 \frac{dB_2}{dt} = i_2 R + 4\pi \mu_0 \mu_r \left\{\left(n_2^2 \frac{O_1}{l_1} + n_3^2 \frac{O_2}{l_2}\right)\frac{di_2}{dt} - n_1 n_2 \frac{O_1}{l_1} \frac{di_1}{dt}\right\}$$

aannemende dat we in beide kernen met dezelfde waarde van μ_r te maken hebben.

Introductie van een weerstand in de secundaire keten is voor onze doeleinden onontbeerlijk. Voor R = 0 nemen de vergelijkingen n.l. de volgende zeer eenvoudige vorm aan

$$E = n_1 O_1 \frac{dB_1}{dt} \qquad (1)$$

$$n_2 O_1 \frac{dB_1}{dt} = n_3 O_2 \frac{dB_2}{dt}$$

waaruit o.a. volgt $B_2(t) - B_2(0) = \frac{n_2 O_1}{n_3 O_2} \{B_1(t) - B_1(0)\}$.

Zijn de constanten zodanig gekozen, dat de digitkern van toestand is veranderd tegen de tijd dat dit met de selectiekern het geval is, hetgeen vereist $n_2 O_1 \geq n_3 O_2$ dan blijkt dus dat we de selectiekern niet zo kalm kunnen terugdrijven dat de digitkern niet mee terug wordt genomen, een eerste vereiste i.v.m. onze schrijftechniek.

Overheerst daarentegen in de tweede vergelijking de resistieve term de inductieve term t.g.v. de digitkern, dan krijgen we bij goede benadering:

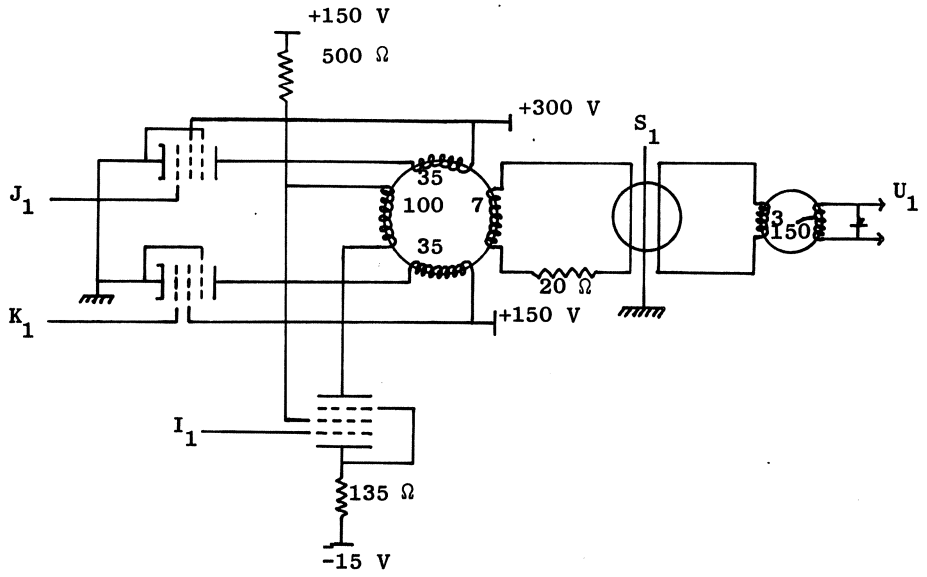
$$E = n_1 O_1 \frac{dB_1}{dt}$$

$$n_2 O_1 \frac{dB_1}{dt} = i_2 R$$

$$i_2 = \frac{E n_2}{R n_1} .$$

Nu is het derhalve mogelijk door voor uit- en inlezen van een verschillende E gebruik te maken, te bereiken, dat de digitkern bij het uitlezen wel, bij het inlezen niet uitsluitend door de selectiekern wordt meegenomen.

In de practijk blijkt de volgende opstelling te voldoen



De bedoeling is, dat tijdens het uitlezen en teruglezen de bijbehorende pentodes bottomen zodat gedurende zekere tijd de aanname van een constante spanning over de primaire van de selectiekern zo goed mogelijk vervuld is.

Aannemende dat er ongeveer 250 V over de spoel komt te staan vinden we uit (1) bij het uitlezen

$$\frac{dB_1}{dt} = \frac{250}{35 \times 5{,}3 \times 10^{-6}} = 13 \times 10^5 \text{ pra. Gauss/sec.}$$

Een fluxverandering van $2 \times 1800 = 3600$ Gauss = 0,36 pra. Gauss vindt dan plaats in $\frac{36}{13} \times 10^{-7}$ sec. = 0,28 μsec. terwijl de secundaire stroom $\frac{250 \times 7}{20 \times 35} = 2{,}5$ A is, een waarde, die in de goede orde van grootte ligt als we bedenken, dat met een coercitiefkracht van 1,5 Ø bij de gegeven waarden van de parameter een stroom van 0,64 A correspondeert.

Het uit de digitkern verkregen signaal heeft een open spanning van ca. 0,5 V en wordt opgetransformeerd met een toroidale ferroxcube transformator met de volgende afmetingen

A	B	C	
26,5	20,0	5,0	mm

Het ingangssignaal zakt door deze belasting tot ca. 0,2 V terwijl aan de uitgang waar de slingeringen uitgedempt worden met een germaniumkristal ca. 10 V verkregen wordt. Om deze spanning te bereiken bleek het noodzakelijk de 150 windingen van de secundaire wikkeling zeer zorgvuldig te leggen.

Het beschreven proefje wordt uitgevoerd met twee selectiekernen, die ieder 10 digitkernen moeten drijven. De uitleeswikkelingen van beide staan in serie evenals de inleeswikkelingen. Selectie geschiedt dus door omschakeling van de bias-wikkelingen.

Het van de kern afkomstige signaal wordt versterkt en onthouden in 10 flipflops en vervolgens via een relaisschakeling teruggevoerd naar de invoerlijnen S, waardoor in geval van een 1 een stroom van 300 mA loopt. Tijdens de terugleesperiode wordt aldus het juist uitgelezen getal weer op zijn oorspronkelijke plaats opgeborgen. Een en ander wordt bestuurd door een cijfertijdring met een basisfrequentie van 500 KH en wel zodanig, dat elke 24 µsec. een keer uit- en weer ingelezen wordt.

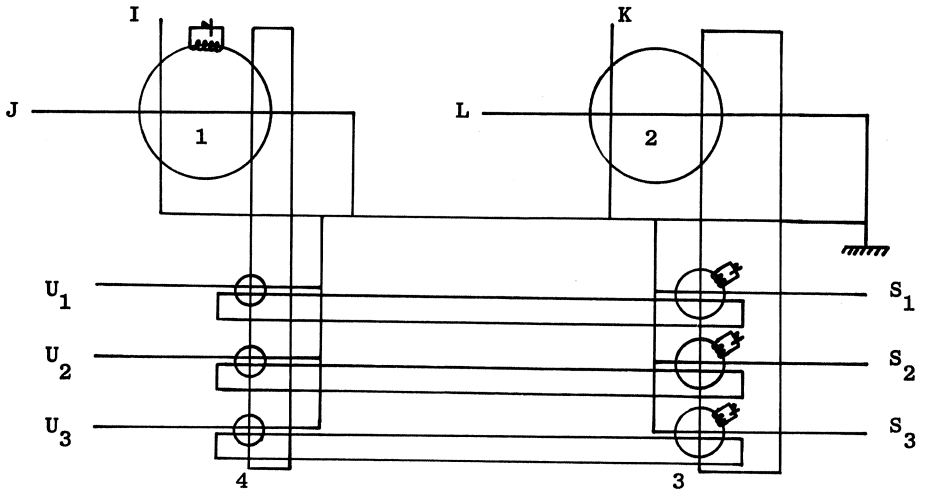
Tevens is het mogelijk om een getal in een van de twee adresplaatsen in te lezen van 10 handschakelaars uit.

Grote zorg moest worden besteed aan het vermijden van gemeenschappelijke gedeelten van de uitleeswikkelingen U van de verschillende kernen. Vandaar dat al deze wikkelingen dubbelzijdig zijn uitgevoerd en geheel los van de aarde zijn gehouden.

De frequentie, waarop de kernen bedreven worden is dus ca. 40 KH, de bovengrens ligt naar gebleken is, in elk geval beneden 500 KH, hetgeen betekent, dat zij voor toepassing in circuits, die met klokpulsfrequentie kunnen wisselen niet in aanmerking komen. Er zijn toepassingen denkbaar van deze kernen, waarbij men van een selectiekern slechts een bekrachtiging in één richting van een digitkern zou wensen. Zulks zou mogelijk zijn door in de doorkoppelwikkeling een gelijkrichter op te nemen. Bedenkt men echter, dat het

hier gaat om stromen van enige ampères dan moeten we deze oplossing laten vervallen, daar de voor dit doel geschikte gelijkrichters ten enenmale ontbreken. Een betere oplossing is dan het leggen van een extra wikkeling op de selectiekern met een passend aantal windingen, waaroverheen men een germaniumkristal schakelt om de ongewenste polariteit weg te nemen.

Een voorbeeld van een dergelijke schakeling is de volgende:



In de kernen 4 is de informatie bewaard. Kernen 1 en 2 maken deel uit van twee matrices. Kernen van het type 3 zijn er evenveel als van type 4. Zij verkeren alle in dezelfde toestand (1). De schrijflijnen S lopen door alle kernen van type 3 heen.

Bij het uitlezen wordt de bias-stroom door I opgeheven en wordt J bekrachtigd ten gevolge waarvan kern 1 van 1 naar 0 gaat. Dit drijft kernen 4 in een nul en de inhoud verschijnt op de uitgangen U_1, U_2, U_3. Bij wederom bekrachtigen van I gaat 1 naar toestand 1 terug zonder kernen 4 daarbij te storen. Het terug te schrijven signaal verschijnt op de lijnen S_1, S_2 en S_3 en wel in die zin, dat geen stroom loopt daar, waar een 1 geschreven moet worden. De eventueel lopende stroom moet voldoende zijn om de kernen 3 in de 1-toestand te houden onafhankelijk wat er met kern 2 gebeurt.

Bij het terugschrijven wordt 2 geselecteerd via lijnen K en L. De overgang 1-0 van 2 drijft al die kernen 3 in toestand 0, waar geen stroom via een S-lijn loopt. De omslag van kernen 3 drijft tenslotte kernen 4 in toestand 1. Bij terugdrijven van 1 in toestand 1 gaan kernen 3 weer mee zonder daarbij kernen 4 te storen. In dit systeem zijn de stromen in het geheel niet critisch meer, doch wordt alleen van sommigen geëist, dat zij groot genoeg zijn.

Hoewel de prijs van de gebruikte kernen nog niet geheel vaststaat, moeten we wel verwachten, dat deze voor de kleinste maat kernen enige dubbeltjes per stuk zal bedragen.

Aannemende een prijs van *f* 0,30 komen we voor een geheugen van 1000 woorden van 30 digits tot een bedrag van ca. *f* 10.000,- of rondweg 10 × zoveel als men betaalt voor een trommel met een dergelijke capaciteit.De access-tijd daarentegen is bij een kerngeheugen gemiddeld een factor 10^4 kleiner.

ONTWERP VAN EEN INVOERPROGRAMMA VOOR EEN ELECTRONISCHE REKENMACHINE

H.J. Heyn

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 27 november 1954 te Amsterdam.

De machine, die als een proefmodel op het Natuurkundig Laboratorium bij Philips in aanbouw is, zal een binaire machine worden met een capaciteit van 20 cijfers per woord. Het rekenkundig orgaan werkt parallel en bevat 3 registers, A, M en S. Getallen, x, in de machine zijn ware breuken, $-1 < x < +1$, met de komma achter het tekencijfer en met 1-complementen als voorstelling voor negatieve x. De machine kent +0 en -0.

Bij optelling ontvangt M de opteller uit het geheugen; in de A(ccumulator) ontstaat de som. Bij vermenigvuldigingen wordt een factor in S gebracht, de andere in M; het product ontstaat in A en S en wel als $(M) \times (S) + (A) \cdot 2^{-19}$. Het geheugen is voorlopig alleen een magnetische trommel.

Bij het ontwerpen van het invoerprogramma werd in ruime mate partij getrokken van methoden bij de ARRA en de PTERA. Instructies voor de machine zijn van het een-adres systeem. Gedacht is aan 11 digits voor het adres, 16 soorten instructies en mogelijk enkele B-digits en digits met bepaalde functie.

Correctie van instructies tijdens het inlezen zal nodig zijn, er is echter getracht het inlezen zo snel mogelijk te maken door te zorgen dat geen tussentijds opbergen nodig is. Daarom komt de "sluitletter" voorop op de ponsband, daarna volgt het adres, decimaal voor decimaal in binaire vorm en tenslotte de operatie.

Door voor al deze symbolen slechts 4 gaten op de band te ge-

bruiken ontstaat de mogelijkheid om met het 5e gat nog iets speciaals te doen. Zo werd de leesopdracht gecombineerd met een sprong afhankelijk van de aanwezigheid van het 5e gat op de band (gestreept, ongestreept, als bij de PTERA).

De opdrachten

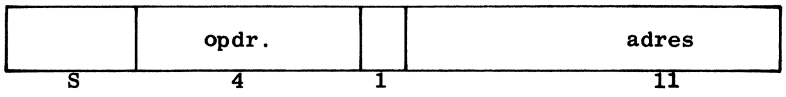
In het invoerprogramma kwam het handig uit de conditionele sprong bij positieve accumulator met "0" te coderen, de sprong met negatieve accumulator heeft de codering "8". Conditionele sprongen zijn dus kenbaar aan de drie achterste nullen in de opdracht-tetrade.

De opdrachten zijn te verdelen in die met en die zonder adres. Opdrachten met adresdeel zijn Optellen, Aftrekken, Vermenigvuldigen, Delen, Wegbergen van A met schoonmaken, Id. zonder schoonmaken, Vullen van S, Wegbergen uit S en Collationeren.

Van deze opdrachten hebben de eerste 8 een oneven codering. De eerste 4 opdrachten gebruiken het M register (collationeren wordt even buiten beschouwing gelaten) en hebben "1" op de een na achterste cijferplaats; van de overige heeft het tweetal dat de Accumulator gebruikt een "1" op de 3e cijferplaats. "Complementaire" opdrachten verschillen steeds 8.

De opdrachten zonder adres zijn Links- en Rechtsschuiven over n plaatsen, twee conditionele sprongopdrachten, een Invoer- en een Uitvoeropdracht. Een mogelijkheid is nog ongebruikt (zie tabel).

De rangschikking in een instructie is



De voorste digits waren oorspronkelijk gereserveerd voor B-digits. Deze zullen echter bij een snel geheugen gemist kunnen worden. De digit tussen adres en opdracht is een Stop-digit: een "1" doet de machine na uitvoering van de opdracht stoppen.

Het invoerprogramma

Zoals met de gehele machines enigszins het geval is, is ook het invoerprogramma onder het bouwen gegroeid. De hier weergegeven vorm is nog een voorlopige, er zullen waarschijnlijk nog wel fouten in schuilen en ook het nut van de verschillende faciliteiten zal wel discutabel zijn. Wij hebben getracht de volgende functies onder te brengen.

1. Normale invoer van een ponsband.
2. Een Check direct na 1. Getallen en Instructies worden opnieuw gevormd en vergeleken met wat op de trommel staat. Bij verschil stopt de machine.
3. Na-Check onder de berekening in geval van twijfel. De machine vergelijkt instructies en getallen met de ponsband. Bij een verschil typt de machine de aanwezige versie en het bijbehorend adres.

Standaard Subroutines (SSR)

Om te maken dat een zelfde bandje voor een SSR zowel tijdens het normaal inlezen als voor het checken bruikbaar is, moet elke SSR beginnen en eindigen met een speciale ponsing, die het invoerprogramma tijdelijk verandert resp. herstelt. De band vangt aan met een voorponsing, die de adrescorrecties verzorgt.

SSR (K)	Einde v. SSR	Blank	Voorponsing	Blank	Begin v. SSR	SSR (K+1)
	2/		4/5,2,96/5/		4/7,2,33/5/10/	
	4/5,2,33/5/		3/V_1		4/7,2,31/5/11/	
	2,0/8/		3/V_2		4/7,2,29/5/0.	
	4/5,2,31/5/		3/V_3		4/5,2,35/0/	
	2,0/8/		etc.		4/8,3,0/8/	

Het Checken

Om de inhoud van het geheugen na een berekening enigszins te kunnen controleren moet men weten welke opdrachten en getallen wel

veranderd mogen zijn na het inlezen en welke niet. Daarom zijn de woorden verdeeld in

vaste instructies, voorafgegaan door 4/0
variabele instructies, voorafgegaan door 4/1
vaste getallen, voorafgegaan door 4/2
variabele getallen, voorafgegaan door 4/3

We kunnen nu met behulp van een voorbereidingsbandje het invoerprogramma zo omvormen, dat vaste instructies en getallen van de band af opnieuw worden opgebouwd en vergeleken worden met de geheugeninhoud. Is hun verschil niet nul dan stopt de machine. Bij variabele instructies en getallen wordt de geheugeninhoud uitgetypt.

Voorbereidingsbandjes

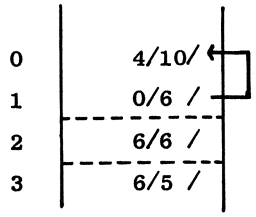
Gemeenschappelijk:	4/5	2	33/5/	
		2	0/8/	0/8/ → 33
	4/5	2	31/5/	
		2	0/8/	0/8/ → 31
	4/6	2	0/2/	0/2/ → 68
	4/5	2	60/5/	60/5/ → 30

Geheugenplaats	Invoer		Check na inlezen		Check na berekening	
60	2	0/2/	2	0/2/	2	0/2/
61	2	0/2/	2	0/2/	2	v.v./5/
62	2	0/2/	2	0/2/	2	0/2/
63	2	0/2/	2	0/2/	2	v.v./5/
64	2	88/3/	2	88/3/	2	88/3/
65	2	35/0/	2	35/0/	2	35/0/
66	2	72/0/	2	72/0/	2	72/0/
67	2	53/0/	2	53/0/	2	53/0/
68	2	89/11/	2	89/3/	2	89/3/
69	2	0/2/	2	0/2/	2	0/2/
70	2	0/8/	2	0/8/	2	β/8/
71	2	0/8/	2	0/8/	2	α/0/

α = beginadres van typeroutine, die inhoud van A negatief uittypt.
$\beta = \alpha - 1$, in β staat $[2^{-19}]/11$

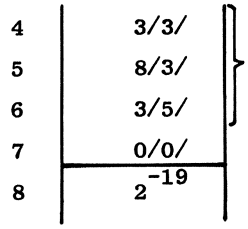
Het inlezen van het invoerprogramma

Alvorens de volgende 4 instructies met de hand in te lezen, worden de twee tracks, die het invoerprogramma beslaat, vol met nullen geschreven.



Opberginstructie

Een speciaal bandje breidt dit programma uit met de volgende instructies



ophogen van opberginstructie

constante voor ophogen

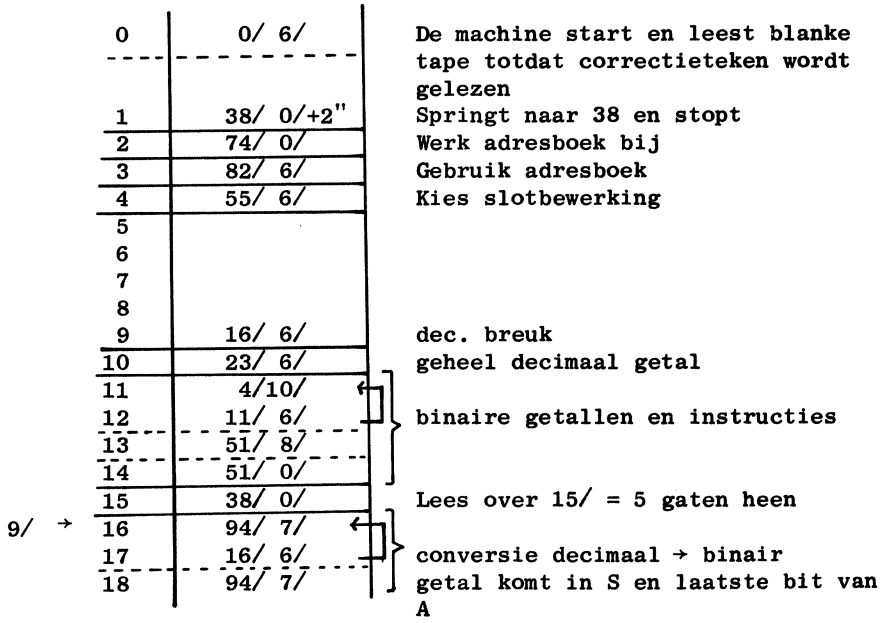
Het uitbreidingsbandje bevat de volgende ponsingen:

0	500	3/3/	(6) = 3/5/	
	500	8/8	(3) = 8/5/	
	1/1/		(8) = 2^{-19}	
	500	5/5	(3) = 5/5/	
	300	8/8/	(5) = 8/3/	
	500	4/4	(3) = 4/5/	
	300	3/3/	(4) = 3/3/	
	501	13/13	(3) = 29/5/	Begin op te bergen in 29.

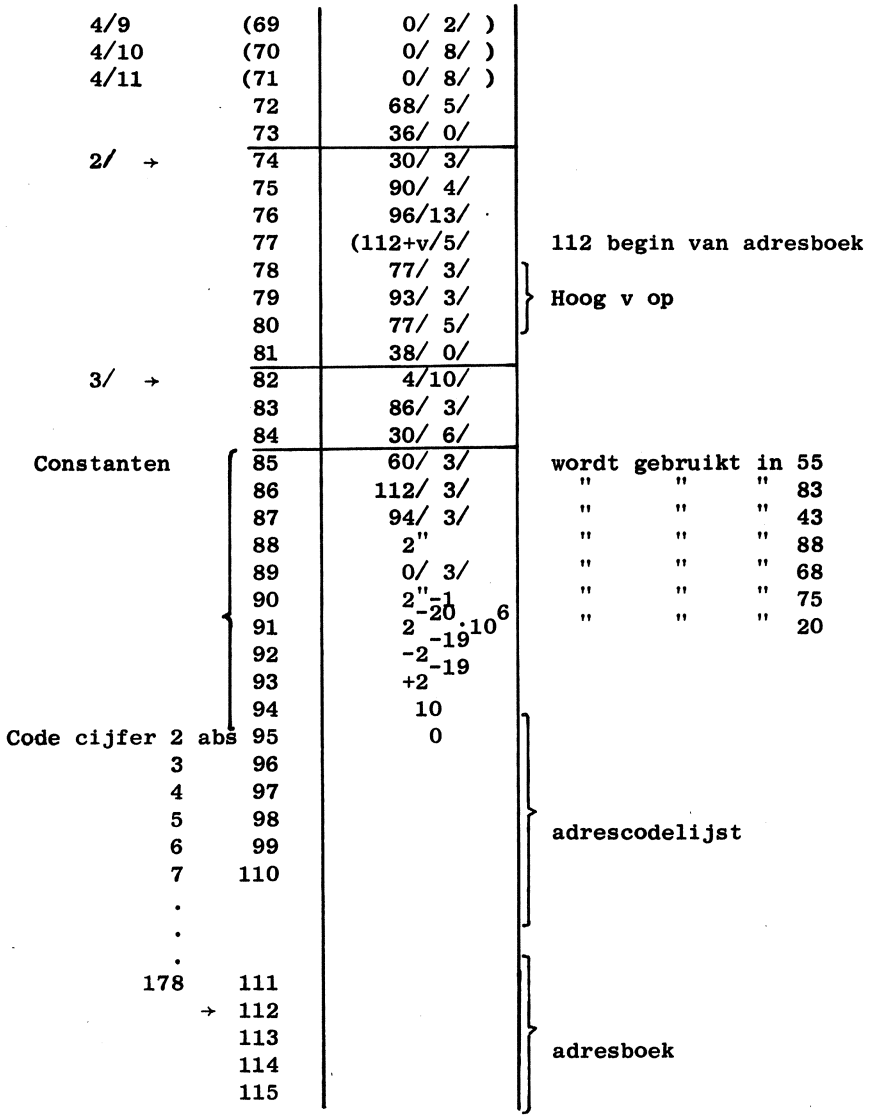
Het invoerprogramma kan nu ingelezen worden naar keuze geheel, of slechts de opdrachten 29 tot 53 in hexadecimale cijfers.

Tabel van Instructies

n/0/	Spring naar n als (A) ≥ +0
n/8/	Spring naar n als (A) ≤ -0
n/1/	(n) naar S
n/9/	(S) naar n, S schoonmaken
n/2/	(A,S) schuif n plaatsen naar rechts
n/10/	(A,S) schuif n plaatsen naar links
n/3/	(A) + (n) → A
n/11/	(A) - (n) → A
n/4/	Collationeer (A) met (n), uitkomst in A
n/12/	.....
n/5/	(A) n met schoonmaken van A
n/13/	(A) n zonder schoonmaken van A
n/6/	lees band en spring naar n indien het symbool ongestreept was
../14/	type tetrade uit Accumulator
n/7/	(S) × (n) → A,S
n/15/	(A,S) : (n) → S rest in A



	19	18/10/	naar A en 1e bit van S
	20	20/15/	quotient in S
	21	vv/5 /	A schoonmaken
	22	26/ 0/	
10/ →	23	94/ 7/	conversie decimaal → binair
	24	23/ 6/	getal in S
	25	94/ 7/	
	26	28/ 6/	leesteken
	27	92/ 7/	
	28	19/10/	getal naar A
	(29	0/ 2/)	Slotbewerking
Checken: (n/11/)	(30	n/ 5/)	Opberginstructie
(1/0/)	(31	1/ 8/)	
Check op 0	32	93/ 3/	
	(33	1/ 8/)	opberginstructie ophogen
	34	30/ 3/	
	35	30/ 5/	
	36	68/ 3/	Heersende slotbehandeling
	37	29/ 5/	herstellen
	38	41/ 6/	
	39	40/ 5/	gestreept code cijfer
	(40	c/0)	
	41	93/11/	Blanke tape wordt genegeerd
	42	38/ 8/	
	43	87/ 3/	ongestreept code cijfer
	44	51/ 5/	
	45	94/ 7/	
	46	45/ 6/	adres converteren
	47	94/ 7/	
	48	7/10/	operatie lezen
	49	11/ 6/	
	50	12/10	
	(51	93+c/ 3/)	adrescorrectie
	52	29/10/	
	52a	29/ 8/	
	53	58/ 5/	
	54	38/ 6/	
4/v →	55	85/ 3/	vorm en plaats 57
	56	57/ 5/	
	(57	60+v/ 3/)	
(y/5/)	(58	29/ 5/)	
	59	38/ 0/	
4/0	(60	0/ 2/)	vaste instructie
4/1	(61	0/ 2/)	variabele instructie
4/2	62	0/ 2/	constante
4/3	63	0/ 2/	variabele constante
4/4	64	88/ 3/	plaatst stopteken
4/5	65	35/ 0/	vervangt opberginstructie
4/6	66	72/ 0/	" heersende slotbehandeling
4/7	67	53/ 0/	
4/8	68	89/11/	


4/9 (69 0/ 2/)
4/10 (70 0/ 8/)
4/11 (71 0/ 8/)
72 68/ 5/
73 36/ 0/
2/ → 74 30/ 3/
75 90/ 4/
76 96/13/
77 (112+v/5/
112 begin van adresboek
78 77/ 3/
79 93/ 3/
Hoog v op
80 77/ 5/
81 38/ 0/
3/ → 82 4/10/
83 86/ 3/
84 30/ 6/
Constanten 85 60/ 3/
wordt gebruikt in 55
86 112/ 3/ " " " 83
87 94/ 3/ " " " 43
88 2" " " " 88
89 0/ 3/ " " " 68
90 2"-1 " " " 75
91 2^-20.10^6 " " " 20
92 2^-19
93 -2^-19
94 +2^10
Code cijfer 2 abs 95 0
3 96
4 97
5 98
6 99
7 110
178 111
→ 112
113
114
115
adrescodelijst
adresboek

FLOATING ADDRESSES IN AN AUTOMATIC DIGITAL COMPUTER

W.L. van der Poel

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 29 januari 1955 te Amsterdam.

1. Introduction

In a recent article Wilkes has described a system of floating addresses for an automatic computer in which numbers as well as instructions are held in the same store. During investigations in the Central Laboratory it became clear that, although floating numbering relieves the programmer of much administrative actions, the routine for handling these floating addresses had many disadvantages.

First, an exposition of the general system of programming for PTERA (Post. Tel. Automatische Reken Automaat) will be given. Especially the points of difference between PTERA and EDSAC will be stressed.

Second, the system of floating addresses as used in PTERA will be given. No special list of instructions which must be altered afterwards is needed nor a process has to be applied after the programme has been read into the store. The programme is immediately adjusted during input.

2. The system of relative addressing in PTERA

PTERA is a one-address code machine. The store consists of a magnetic drum, containing 1024 words of 31 bits each. The access time plus operation time of an instruction is always 25 ms for all storage locations. (In PTERA taking in a new instruction is also an instruction and is treated in the same way as a calculating instruction. There is no essential difference between instruction cycle and operation cycle. Both take 25 ms.) The inputmedium is normal teletype tape (20 symbols/s), the output can be on an electric typewriter and/or on a tape punch (10 symbols/s).

To the instruction as it stands in the memory a code digit is added on the tape. The structure of an instruction on the tape is

c a / o p

in which c is a single code digit, a is an address consisting of one or more digits and op are two octal digits giving the operational part. The stroke / is associated with the last digit of the address and serves as a separation mark between address and operation.

An instruction c a/op on the tape is put in as a + (35 + c)/op or in words: put in the instruction with address a, augmented by the contents of storage location 35 + c. ((x) will be shorthand writing for "the contents of storage location x"). Thus with the help of the code digits a system of relative numbering is possible. This is much the same as the system for the EDSAC as described by Wilkes c.q., so it will not be gone into in detail.

A point of difference which has some importance for floating addresses is the controlcombinations on the tape. To begin input at c a the controlcombination c a/10/ is used. Here the last digit is marked with a stroke thus distinguishing between an instruction to be put in and a control combination. Remark that every controlcombination also has a code digit.

The second type of controlcombinations is c a/20/. The meaning of this is: begin to execute programme at c a.
Controlcombinations of this type must always be followed by a dummy instruction (e.g. 0 0/00).

3. The addressbook programma. Stroked codedigits

For special manipulations during input it is an advantage to be able to leave the input programme temporarily without destroying the store instruction. (When it is done by c a/20/ the store instruction is lost.) A possibility to do this exists in PTERA by stroking the code digits. When a code digit is followed by a stroke the normal sequence of address and operation is not followed. Several of these stroked codedigits are relevant for the next paragraphs.

e/: Do not fill in this instruction but proceed to input of next location

2/: Place address of present store instruction in 36, thus making the end of a subroutine the reference point from where the next subroutine begins to number with 0. At the same time this beginning address is also filed in the addressbook, a list in which the beginnings of all subroutines are stored consecutively. The addressbook begins at 74. All subroutines on the tape end in 2/, whether special or library routines.

1/: The structure of an instruction using 1/ is always:

1/r/a/op

in which r is called the rank number and a and op are address and operation. Under control of this punching a + (74 + r)/op is actually put in. 1/ is used when a routine refers to another subroutine previously put in. Although the actual location of this subroutine is not known by the programmer, he knows in which order the subroutines have been put in. Hence he can refer to the beginning of this subroutine by using its appropriate rank number.

Of other stroked codedigits in use, the following will only be mentioned very briefly, 8/, a/ and b/ are used for putting in fractions, f/ stops the tape, 0/, 7/ and 9/ govern the use of floating addresses and will be extensively dealt with in the next paragraph.

4. Floating addresses

In drawing up a programme on paper most of the instructions need not be numbered, as they are never referred to. Only a few instructions require numbering. Two cases can be distinguished:

1^{o}) An instruction refers backward to a location, already put in. This location can be known to the machine.

2^{o}) An instruction refers forward to a location which has not been filled in yet.

The first case is easiest to deal with. The only action that need be done by the programmer is assigning a floating address to the location, where afterwards has to be referred to. In the machine the true location associated with the floating address must be retained in a list.

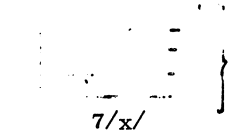
The second case is more difficult. Now the location to where must be referred is not known in advance. Wilkes proposed the use of a second list of locations which must afterwards be adjusted or giving these instructions a special mark. In both cases a special programme has to work through a routine for adjusting these orders.

Now it can be shown that it is possible to make efficient use of a floating address system by using a list of addresses associated to floating addresses only. Therefore it is necessary to describe first the method of listing the floating addresses. When looking for an appropriate list, the parameter registers from 35 to 43 are most naturally suited. Furthermore the addressbook (from 74 onward as far as necessary) provides a second list as an extension when 35 to 43 is not sufficient. Listing is done by a stroked codedigit 7/. The meaning of 7/x/ is:

7/x/: Store the actual address of the present store instruction of the input programme in 35 + x.

To a floating address x can be referred as x 0/op. It is even possible to use a local numbering beginning at x 0 by just adding an address to the codedigit: x a/op.

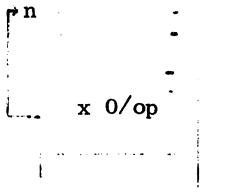
A short example will look as follows:



Preceding instructions on the tape, all having the form c a/op

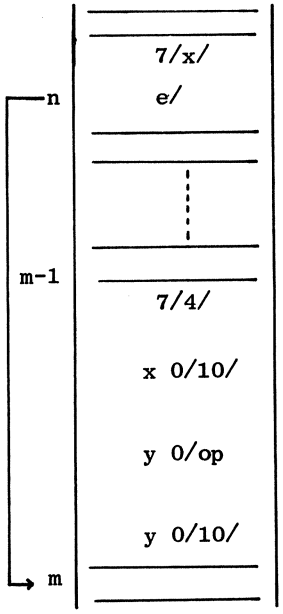
7/x/

With this controlcombination n is put in 35 + x



This instruction refers back to true address n

When a programmer is making a programme in the conventional numbered way and he must refer forward, then he does not fill in the exact address at that moment but he waits till he knows the address.
The same method can be applied to floating addresses for referring forward. A short example may elucidate this point:



This instruction has to refer to location m which is not known in advance. Hence by e/ location n is temporarily kept unfilled.

After putting further instructions the programme has come to m - 1.

At this moment the true location of m is fixed with floating address y.

With this controlcombination input is begun at floating address x

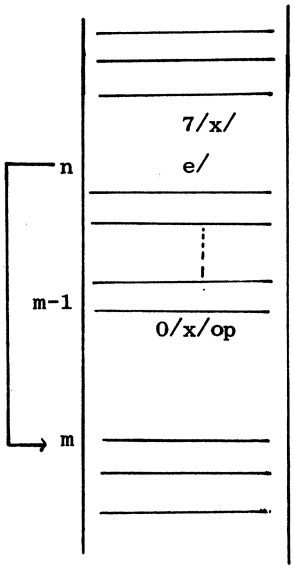
The referred input of the instruction on n takes place

Input is resumed at m

The only trick of this procedure is simply not putting in in the order in which the instructions are executed, but in the order in which they would be written down with normal programming.

Several drawbacks of the above process are obvious. A floating address must be attached to m only for resuming input at m. Two controlcombinations are needed for beginning input at n and m.

These drawbacks can be easily overcome by a new stroked codedigit 0/. Here the property of stroked codedigits not to destroy the store instruction is indispensable. The structure of a combination with 0/ is: 0/x/op meaning: store in floating address x the instruction m/op where m is the actual address of the present store-instruction. The last example can be redrawn with 0/ as follows:



Temporarily unfilled

At the moment that m has to be filled (and m is known to the machine) m/op is stored in n.

The routines for the actions of 7/ and 0/ consist of only 7 and 9 instructions resp.

5. The use of the addressbook for floating addresses

Although the locations 35 - 43 can be used over and over again in different routines, their number is rather limited, as also parameters make use of 35 - 43. Therefore an extension to 7/ is made by 9/.
It us used in the following way:

9/y: Store the actual address of the present store instruction in 74 + y.

The addressbook is then used as a second list. To this list can be referred with instructions of the structure 1/y/0/op, the same as used for subroutines. The codedigit 0/ cannot be applied to this list.
9/ is restricted to fix the addresses for more permanent sections of a programme. Whenever possible the 7/x/ combinations have preference, especially for temporary use.

On this spot it is perhaps curious to remark that even without 7/, 0/ or 9/ a system of floating addresses is possible.

With 2/ floating addresses can be assigned, but only in numerical order. The references can be made with 1/y/0/op or also to the last floating address assigned with 1 0/op. The numerical order in which the floating addresses must be given is the main reason for using this system only very occasionaly.

Although floating programmes have the very agreeable property that any number of instructions can be inserted without renumbering, the complete unacquaintance of the programmer with the actual addresses makes debugging of a programme on the machine not so easy. E.g. when it is found that only one instruction is wrong it is often corrected by making a supplementary correction tape which goes into the machine after the programme. To be able to know the actual addresses belonging to the floating addresses, a special edition of the 7/ and 9/ routine has been made which does not only perform the actions explained before, but also gives a typewritten account of every 7/ and 9/ combination, writing down the floating address (one digit for 7/, two for 9/) followed by the actual address. With this record it is very easy to make alterations in a programme without correcting first the tape and putting it again in the machine. The possibility of local numbering from a floating address by x a/op or 1/y/a/op is very useful for these corrections.

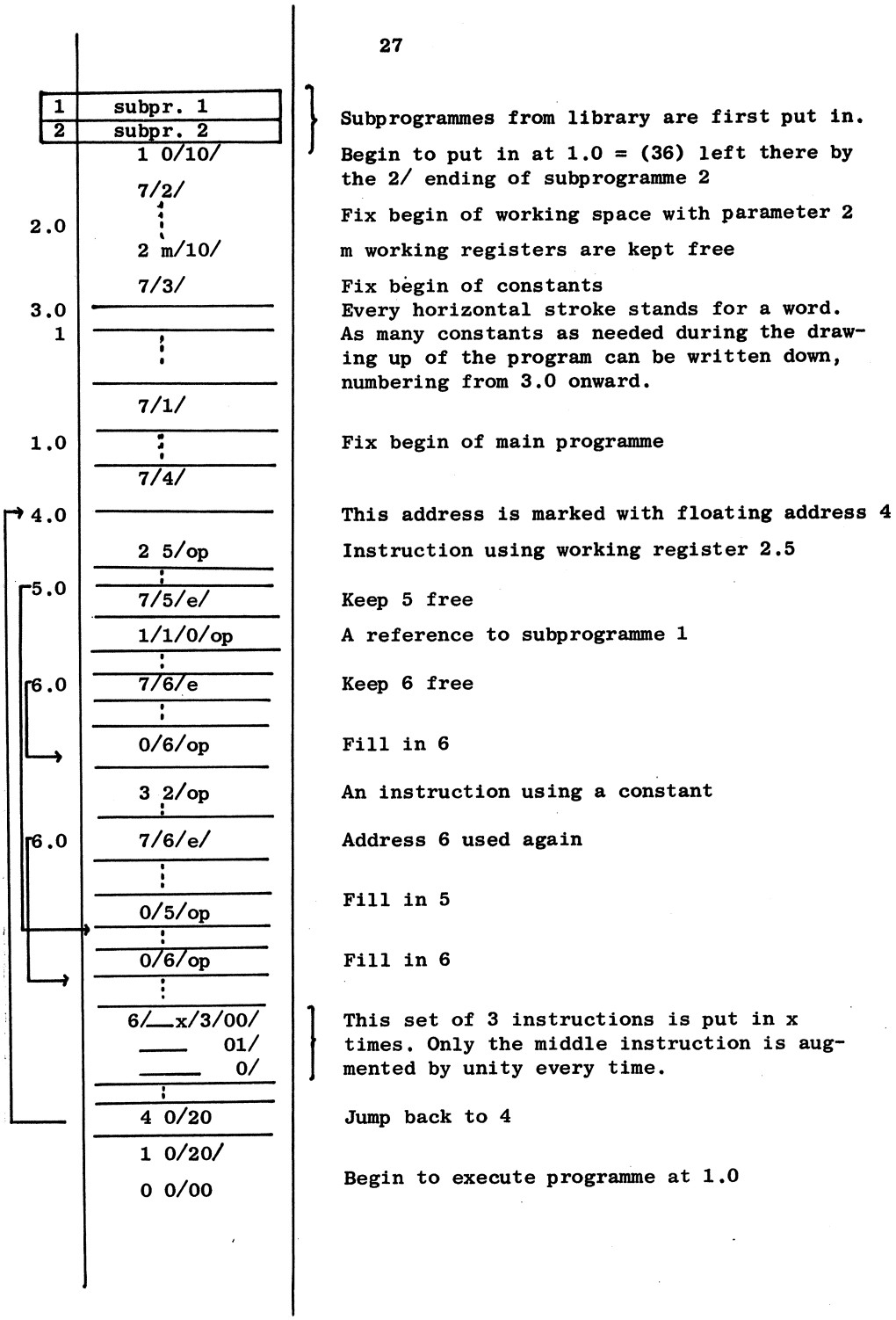
6. The use of floating addresses with serial input

PTERA is not a very quick machine. Therefore the tendency is to make no cycles but to stretch a programme whenever possible.
No storing of partial results and no counting need be done then.
Such stretched programmes are very tedious to draw up and to punch.
For putting in long sequences of rather similar instructions a so called serial input programme has been made. On the tape an instruction of the form 6/c a/op x/y/z/ can be written. 6/ starts serial input, c a/op is the instruction which has to be put in, x is the number of times it has to be put in, y/ is the number of locations which must be gone further before the next instruction of the series must be placed and

z/ is the increment with which the instruction must be augmented every time it is put in. When x is preceded by a 0 then the number of times is taken as (35 + x).
This enables the programmer to give the number of times by a preset parameter. Especially in this case a floating address system is indispensable as the length of the series is not fixed and not known in advance. Input is resumed normally after the last instruction of the series, but when a 0 preceeds z/, then input is resumed after the first instruction of the former series, enabling interlaced series to be put in. In the last case 6/, x/ and y/ need not be repeated as they are the same.

7. A general example

In this paragraph a general example will be given of the normal usage of floating addressing. As the code of the machine is not given only the structure of the flow diagram can be discussed here. (This structure is made clear by arrows before the margin.) It serves further the purpose of compiling several possibilities of the use of address-book and serial input.


1 subpr. 1
2 subpr. 2
1 0/10/
7/2/
2.0
2 m/10/
7/3/
3.0
1
7/1/
1.0
7/4/
4.0
2 5/op
5.0
7/5/e/
1/1/0/op
6.0
7/6/e
0/6/op
3 2/op
6.0
7/6/e/
0/5/op
0/6/op
6/__x/3/00/
____ 01/
_____ 0/
4 0/20
1 0/20/
0 0/00
Subprogrammes from library are first put in.
Begin to put in at 1.0 = (36) left there by the 2/ ending of subprogramme 2
Fix begin of working space with parameter 2
m working registers are kept free
Fix begin of constants
Every horizontal stroke stands for a word.
As many constants as needed during the drawing up of the program can be written down, numbering from 3.0 onward.
Fix begin of main programme
This address is marked with floating address 4
Instruction using working register 2.5
Keep 5 free
A reference to subprogramme 1
Keep 6 free
Fill in 6
An instruction using a constant
Address 6 used again
Fill in 5
Fill in 6
This set of 3 instructions is put in x times. Only the middle instruction is augmented by unity every time.
Jump back to 4
Begin to execute programme at 1.0

HET UNIVAC SYSTEEM

G.C.J.F. Nielen

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 26 februari 1955 te Amsterdam.

De Universal Automatic Computer UNIVAC is een groot "general purpose" digital computer systeem, dat speciaal voor de administratie is ontworpen. Dit blijkt uit de bijzondere invoermogelijkheden, de volledige - automatische - interne controle en de eenvoudige bediening.

De invoer in het systeem wordt - evenals bij het ponskaartensysteem - verzorgd door losse machines. Men onderscheidt:

1. Unityper I een apparaat, dat aanslagen op een gewoon schrijfmachine toetsenbord omzet in magnetische code op metalen tape (veel interne controlemogelijkheden op tape layout).
2. Unityper II een normale electrische schrijfmachine, maar met een gesynchroniseerde tape recorder.
3. Verifytyper een Unityper II met gesynchroniseerde tape reader, een vergelijkingsgedeelte en met de mogelijkheid van correctie van de tape.
4. Card to Tape Converter een ponskaart reader met gekoppelde tape recorder. Snelheid: 200 kaarten/min. Foutenmogelijkheid verwaarloosbaar klein.
5. Teletype to tape converter.

Het geheugen bestaat uit metalen magnetische tape met een 7 kanaalscode plus een "sprocketchannel".

4 bits voor een cijfercode (excess three semi binary), 2 voor het onderscheiden van alfabet en 1 bit voor het oneven maken van het aantal impulsen per code. Tot 120 tekens per inch; lengte tot 1500'.

De bewerkingen gebeuren door de eigenlijke computer.

De uitvoer wordt verzorgd door:

1. Uniprinter een tape reader met aangekoppelde electrische schrijfmachine.
2. High Speed Printer een drumprinter, die de code op tape leest en met regels van 130 tekens tegelijk afdrukt. Snelheid: 10 regels per seconde. Foutenmogelijkheid verwaarloosbaar klein.
3. Tape to card converter.
4. Tape to teletype converter.

De voornaamste karakteristieken van de UNIVAC computer zijn:

Invoer: tot 10 UNISERVO's. Dit zijn niet autonome tape reader-recorders, die vooruit of achteruit lezen met een snelheid van ± 100" per seconde; dus ± 10.000 tekens per seconde. Alleen de linkerspoel is verwisselbaar en er bestaat dus zoiets als een "rewind".
Uitvoer: Dezelfde UNISERVO's. De schrijfsnelheid is gelijk aan de leessnelheid. Achterwaarts schrijven is niet mogelijk.
Geheugen: 100 acoustische delaylines, elk van 10 woorden (12 tekens per woord). Maximale accesstime 400 musec. Verder nog een input- en outputbuffer, elk van 6 delaylines. Het overbrengen van tape naar buffer houdt rekening met de leesrichting; de buffer bevat dus altijd het gelezen blok (van 60 woorden) in de "vooruit" volgorde. Transfers tussen de buffers en het geheugen vragen 3500 musec.
Rekenkundig orgaan: Alle bewerkingen gebeuren in serie. Naast de normale registers is er een register voor het verplaatsen van 2 woorden en nog een voor het verplaatsen van een hele delayline-inhoud. Doordat de computer zowel alfabetische als numerieke tekens verwerkt, is het rekenkundig orgaan aan conventies gebonden:

A + 2 = A A + A = "error" A × A = onzin A kleiner dan Z
9 kleiner dan A

In verband met de vaste woordlengte is het mogelijk, uit een woord tekens te selecteren.

Controle: De interne automatische controle resulteert in een gedeeltelijk automatische interne diagnose.
1. Bij het lezen van de tape wordt elk teken onderzocht op een oneven aantal impulsen. Bij het optreden van een even aantal wordt het hele

blok teruggelezen en herlezen. Na 5 repetities stopt de computer.
2. Ook elke interne transfer gaat over de hoofdversterker (High Speed Bus) waar telkens opnieuw het aantal impulsen oneven moet zijn. Een even aantal stopt de computer en geeft de indicatie HSB.
3. Elke 30 seconden wordt het hele geheugen uitgelezen over de HSB en opnieuw ingelezen. (In de praktijk wordt deze controle altijd uitgeschakeld).
4. Met uitzondering van het geheugen en de buffers bestaat de hele computer in duplo. Twee registers van elke soort, twee volledige rekenmechanismen, twee HSB's, twee Control Counters etc. De twee HSB's staan in voortdurend contact. Bij gebrek aan overeenstemming stopt de machine en geeft de indicatie, waar de fout optrad.
5. In het geval van een "closed loop" stopt de computer na twee seconden en geeft de indicatie "stall". Bovendien geeft de aangesloten loudspeaker geen geruis meer, doch een (vaak determineerbare) toon.

Besturing: Univac wordt bestuurd door een "stored Program" van 1-adresinstructies, waarvan er twee in een woord voorkomen. De machine kent dus vier phasen:

1. Halen van het instructiewoord.
2. Verhogen van de control counter met 1.
3. Uitvoering van instructie 1.
4. Uitvoering van instructie 2.

Spronginstructies moeten dus in het algemeen rechts in een woord staan. Niet-automatische besturing is mogelijk met behulp van het besturingspaneel (Supervisory Control), dat o.m. voorzien is van een toetsenbord en een schrijfmachine.

Uit het arsenaal van 43 instructies noemen we enkele interessante.

3-0120 Transfer de inhoud van de inputbuffer naar het geheugen (adressen 120-179) en lees een blok (60 woorden) van de tape op Uniservo 10 naar de inputbuffer.

860000 Wind de tape op Uniservo 6 terug en verhinder elk volgend gebruik van deze tape totdat het klapdeurtje van de servo open is geweest.

100620 Stop; geef "input ready" indicatie. Als een woord op het toetsenbord getypt is, zet het dan op adres 620 en vervolg de bewerking.

Q30130 Stop, wanneer de "conditional transfer breakpoint stop selector button" nummer 3 ingedrukt is. Zo niet, spring naar adres 130, mits (rA) = (rL).

R00223 Zet de inhoud van de Control Counter als het adres van een spronginstructie (U) op adres 223. Deze instructie is waardevol in toepassing van subroutines.

Een invoerprogramma is overbodig; de eerste instructie (lees programma-tape) wordt gegeven door het indrukken van een knop op het Supervisory Control.

Overflow: Een woord in Univac bestaat uit 12 tekens. Wanneer we met getallen werken, is de eerste positie gereserveerd voor het teken (0 is plus; - is min), terwijl de 11 volgende posities de cijfers bevatten (alle getallen tussen +1 en -1). Een decimale overdracht naar de tekenpositie doet de computer reageren als volgt:

a. In register A wordt de som zonder deze overdracht afgeleverd.
b. In de eerstvolgende "1-phase" is het adres van de op te halen instructielijn niet de inhoud van de Control Counter, maar zonder meer 000.
c. De inhoud van de Control Counter wordt niet veranderd.

Automatische programma's: In de loop der jaren zijn - behalve de nodige automatische "service routines" - ook vele generatieve programma's ontwikkeld, die - met als invoer de parameters van het specifieke probleem - het specifieke programma en een werkinstructie opleveren.

We noemen: General 2, 3, 4, way sorting routine
General 2, 3, 4, way merging routine
General output edit routine.

DE ELECTRONISCHE REKENEENHEID GAMMA

J. Berghuis

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 26 maart 1955 te Amsterdam.

De Gamma is gebouwd voor boekhoudkundige en administratieve doeleinden en wel voor het gebruik tezamen met andere ponskaartenmachines (de Bull tabulateurs B.S., B.S.M. of de reproduceermachine P.R.D.). Het is de bedoeling om voorkomende arithmetische bewerkingen tijdens het doorvoeren van de kaarten in de genoemde ponskaartenmachines uit te voeren op de electronische rekeneenheid Gamma. De Gamma is dan ook geen zelfstandige machine, de in- en uitvoer moet worden verzorgd door de machine, waaraan hij gekoppeld is. Wij zullen nu dus de twee volgende punten dienen te behandelen:

1e. de organisatie evt. opbouw van de electronische rekeneenheid zelf;

2e. de verbinding met de gekoppelde ponskaartenmachine.

1e. de rekeneenheid zelve:

De Gamma werkt in het binaire-decimale stelsel en is een seriemachine. De geheugens zowel als de accumulator, hier Operateur genaamd, zijn vertragingslijnen. Elk geheugen kan 12 decimale cijfers bevatten, dat betekent dus 48 pulsen. De repetitiefrequentie is 280.000/sec, zodat een volledige omwenteling van de geheugeninhoud 170 µsec duurt. Schakelelementen in de Gamma zijn de germaniumdioden.

Voor de operateur bevindt zich een coderingsapparaat additionneur-soustracteur genaamd, hetwelk de optelling en/of aftrekking van getallen uit andere geheugens bij de inhoud van de operateur verzorgt.

Verder bezit de operateur nog de mogelijkheid van cadrage: dit betekent het gehele getal een decimale plaats naar rechts doen verschuiven; hierbij wordt de plaats 12 gelijkgesteld aan de plaats rechts van 1. Dit geschiedt als volgt: de inhoud van elke plaats wordt 1 plaats

naar rechts geschoven, zodat de inhoud van 1 op plaats 12 staat. De inhoud van 12 blijft nu 11 maal rondlopen op plaats 12, de inhoud der andere 11 plaatsen loopt nu cyclisch rond. Na 170 µsec is dus het gehele getal 1 plaats naar rechts verschoven.

Naar links schuiven is niet mogelijk, maar hier ook niet nodig. Het aantal vertragingslijnen is naar wetenschappelijke begrippen niet groot, n.l. maximaal 32, minimaal 4, al dient hierbij bedacht te worden, dat de ponskaarten een immens geheugen betekenen.

De opdrachten worden gegeven via een schakelbord, de Gamma is dus niet een machine, welke kan rekenen met zijn opdrachten. Wel kan een opdracht geselecteerd worden via ponsingen in de kaart of via het programma.

De operateur bevat altijd de modulus van het getal en de machine kan alleen een teken onthouden, indien hij voorzien is van het dispositief operateur de signe.

Ter onderscheiding van de operateur heten de andere geheugens mémoires banales.

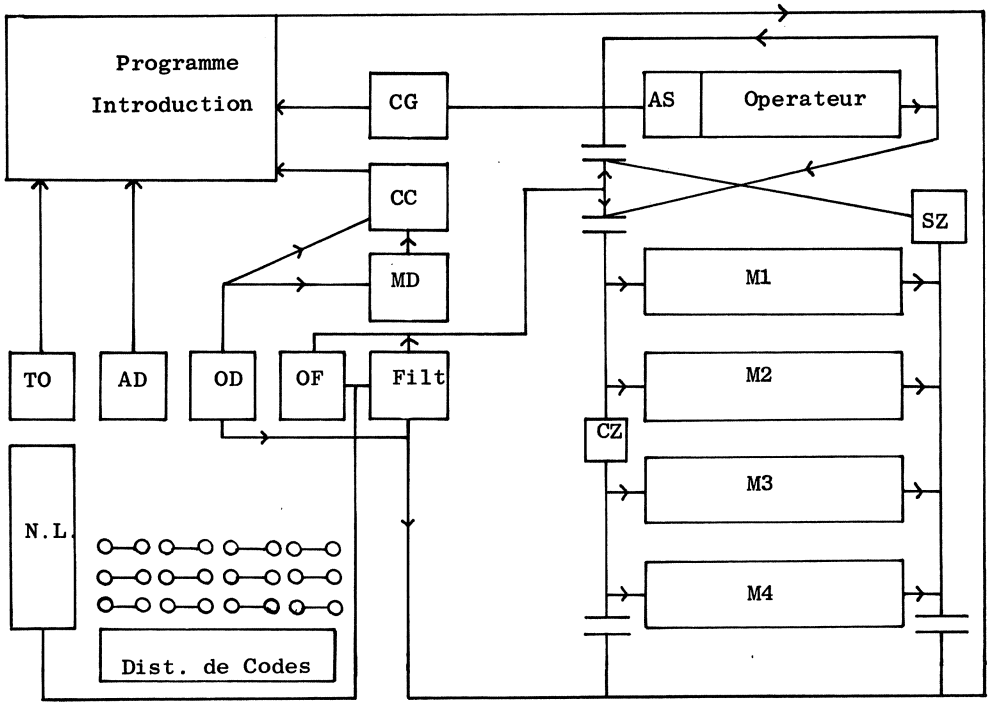
Nu komen bij boekhouden weinig bewerkingen voor met 12 cijfers. Om zoveel mogelijk profijt van de capaciteit der geheugens te trekken, kan men de opdrachten doen uitvoeren voor een gedeelte van de inhoud der geheugens. Bij de opdracht dient men dus de minst en de meest significante plaats (ordre debut en ordre fin) van het banale geheugen aan te geven, waarop de opdracht betrekking moet hebben. Er bevindt zich in de machine een register MD (mémoire de décalages) hetwelk steeds de minst significante plaats van het getal uit de operateur aangeeft. De machine voert de opdracht pas uit, indien MD = OD d.w.z. hij voert zoveel malen de cadrage uit, totdat de inhoud van het register MD gelijk is aan de OD van de uit te voeren opdracht.

De Gamma is een één-adres machine. Een opdracht wordt dus gekarakteriseerd door de volgende getallen TO, AD, OD, OF, elk binair voorgesteld met ten hoogste 4 binalen (max. 15), hierbij is

TO = type d'operation
AD = adress
OD = ordre début
OF = ordre fin.

Het aantal opdrachten op het schakelbord is 32, 64 of 128. Als adres geldt het nummer van het geheugen, de operateur draagt no. 1 en het register OF no. 0. Met behulp van OF is dus het optellen van een constante kleiner dan 10 mogelijk of vermenigvuldigen met een dergelijke constante. Onder bijzondere omstandigheden mag deze constante boven de 10 zijn. In de geheugens met een adres hoger dan twee worden de nullen als tien geschreven, welke codes onderdrukt worden op het canal de données.

Het programma wordt verkregen door door middel van snoertjes een code uit de distributeur de codes naar de ingangen der registers TO, AD, OD en OF te brengen. De nullen moeten opengelaten worden. Een code kan via speciale relais geselecteerd worden, waarbij deze relais bestuurd worden door de gekoppelde machine. Alle TO uitgangen gaan via een coderingsboom van germaniumdiodes naar dezelfde ingang van de besturing of Introduction Programma. Welke uitgang van TO bekrachtigd wordt, wordt bestuurd door de teller Numéro de Ligne.



figuur 1

TABLEAU DE CODE GAMMA-M-

OF ↘	AD ⟶				0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	⟵ AD	
OD ↓	0	4	8	12	jamais	>	=	$\geq$	$\overline{Ms}$				9,14	15,14	1,14	0,14	11,14	12,14	13,14	14,14	O	V
	1	5	9	13	toujours	$\leq$	$\neq$	<	Ms^{+}				8,3	9,3	0,3	11,3	12,3	13,3	14,3	15,3		
	2	6	10	14	V0	V1	V2	V3	V4	V5	V6	V7		2,14	3,14	4,14	5,14	6,14	7,14	8,14		
	3	7	11	15	$\overline{V0}$	$\overline{V1}$	$\overline{V2}$	$\overline{V3}$	$\overline{V4}$	$\overline{V5}$	$\overline{V6}$	$\overline{V7}$	$P^{t}.3$	1,3	2,3	3,3	4,3	5,3	6,3	7,3		
0	0	1	2	3	Condition de transfert en NL																	
1	4	5	6	7																	1	
2	8	9	10	11																	2	AMD
3	12	13	14	15					R à Z				R à Z				Maintien MD				3	ZB
4	16	17	18	19					Filt.		OF→M1		Filt.		OF→MB						4	KB
5	20	21	22	23					M1		$Int.^{r}$→M1 OD=MD		MB		$Int.^{r}$→MB						5	IS
6	24	25	26	27	OF ⟶ M1				Maintien f.MD				MB ⟶ M1				R à Z tot. M1 OD ⟶ MD				6	BO
7	28	29	30	31													M2 ⟶ MD				7	BD
8	32	33	34	35					R à Z f. M1				R à Z f. MB M1 ⟶ MB				Opérations avec cadrage préalable				8	OB
9	36	37	38	39	Sup.	M1 $\overset{>}{<}$ OF			M1 t. $\overset{>}{<}$ M1 f.				M1 $\overset{>}{<}$ MB								9	CN
10	40	41	42	43	cad. préal	M1+OF			M1 × 2				M1 + MB				→ MD=OD				10	AN
11	44	45	46	47	Maint MD	M1-OF			R à Z f. M1				M1 - MB								11	SN
12	48	49	50	51	M1 × OF = M1								M1 × MB = M1				Nombre de décalages de M1=OD				12	MR
13	52	53	54	55	M1 ÷ OF = M1								M1 ÷ MB = M1								13	DR
14	56	57	58	59	M2 × OF = M1M2									M2×MB= M1×M2			Nombre de décalages de M1M2=MD				14	MC
15	60	61	62	63	M1M2÷OF=M2									M1M2÷ MB=M2							15	DC
↑ OD	↑ NL				0				1				2	3 à 15			⟵ AD				↑ TO	

Voor de bespreking van de opdrachten volgt hieronder het tableau de Code voor de Gamma M.

De opdrachten BD (TO = 7) en D.C.C. (TO = 15, AD = 0, OD = 10, OF = 1) zijn speciaal voor het werk met drijvende komma.

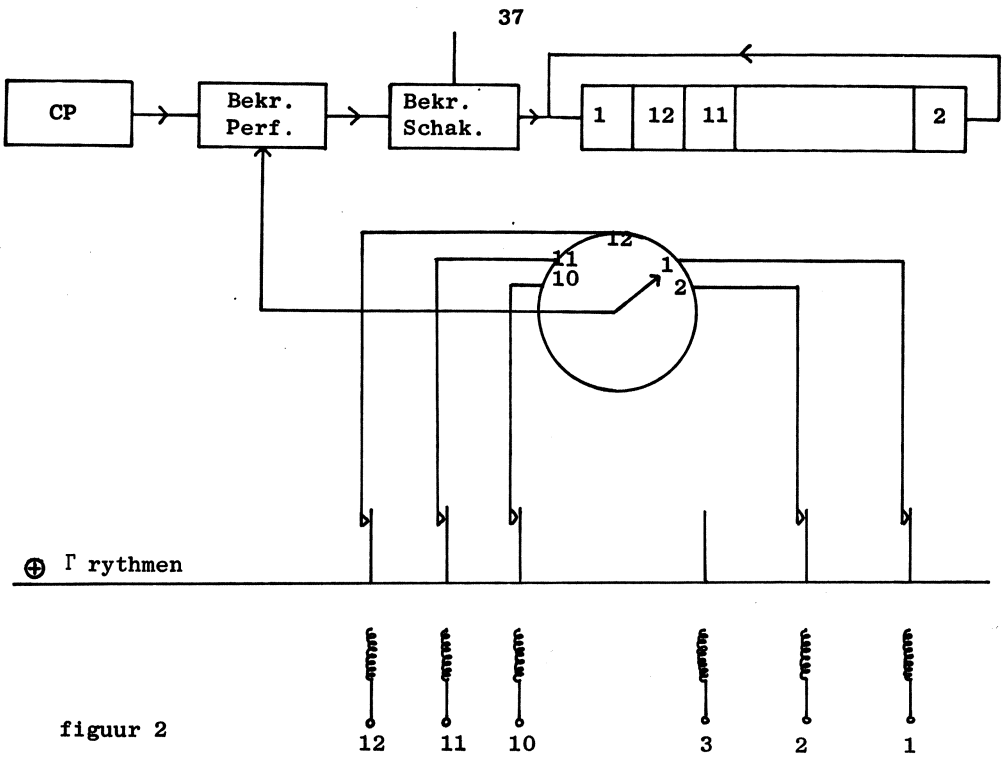
De opdracht 1 heeft alleen zin bij een machine met meerdere schakelborden. Indien de Gamma is uitgerust met een dispositief P.P.C. (Programme Par Cartes) dan is het mogelijk het schakelbord voor subroutines te gebruiken. De opdrachten worden dan in binaire code op de kaart geponst door 4 opdrachten per punt op de kaart naast elkaar (dus 4 × 4 × 4 = 64 kolommen) en door de Gamma in relais vastgehouden. Deze relais worden afgelezen door z.g. e-pulsen (een onderharmonische van de normale frequentie) en uitgevoerd. Het is mogelijk naar het schakelbord en terug te komen via de Varianten.

2e. Verbinding met gekoppelde machine.

Deze verbinding omvat de invoer van getallen eventueel van opdrachten en de uitvoer van getallen.

Het invoeren van opdrachten is reeds behandeld bij P.P.C. Het invoeren van getallen kan op 2 wijzen geschieden n.l. de Introduction Cinématique en Introduction Statique, de uitvoer slechts op één wijze. Als voorbeeld zullen wij de Introduction Cinématique beschouwen en wel vanaf de B.S.

Gedurende de eerste veertien graden van elk punt der machine stuurt de Rupteur van de B.S. een puls uit; het al of niet doorgeven van deze puls door het ponsgat in de kaart geeft een cijfer aan. Deze electrische pulsen dienen omgezet te worden in pulsen bruikbaar op de Gamma. Dit wordt gedaan door de unifieur. Dit apparaat verzorgt tevens het mémoire C.P. (Code Point). Het mémoire C.P. bevat op elk punt van de gekoppelde machine steeds de waarde van dat punt. De introduction cinématique verloopt nu als in de tekening geschetst.



figuur 2

De verbinding tussen de Gamma en de gekoppelde machine geschiedt weer door relais. De uitvoer geschiedt bijna op dezelfde wijze, er is n.l. een thyratron toegevoegd per decimaal cijfer om de relais te bekrachtigen.

ZELF-CONTROLERENDE CODES

C.S. Scholten

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 26 november 1955 te Amsterdam.

De titel van deze voordracht behelst de opgave om, met gebruikmaking van n digits het maximum aantal (k) getallen (= configuraties van deze n digits) te zoeken, zodanig dat elke configuratie van elke andere in tenminste δ digits verschilt ("afstand" δ). Gevraagd derhalve $k = k(n,\delta)$. Een dergelijke code met l getallen duiden we aan met $C(n,\delta,l)$, of als het aantal getallen in het midden gelaten wordt $C(n,\delta)$.

Een voorbeeld vormt de volgende code met $n = 5$, $\delta = 3$.

0 0 0 0 0

0 0 1 1 1

1 1 1 0 0

1 1 0 1 1

Inderdaad verschillen deze vier getallen allen onderling in tenminste 3 digits. Voordat we kunnen besluiten tot $k(3,5) = 4$ moeten we echter nog laten zien dat we, met 5 digits ter beschikking, niet méér getallen kunnen vinden, die allen onderling een afstand δ of meer hebben. Men kan dit natuurlijk proberenderwijs aantonen, doch wij stellen het bewijs liever even uit.

Tot nu toe is geen expliciete formule voor k als functie van n en δ bekend, noch een bruikbare methode om voor één waardenset (n,δ) k te berekenen en een (de) bijbehorende code te construeren.

Niettemin zijn er wel enige gelijkheden en ongelijkheden bekend waaraan k, n en δ voldoen. De voornaamste hiervan zijn de volgende:

a. $\frac{2^n}{\sum_{r=0}^{\delta-1} \binom{n}{r}} \leq k(n,\delta) \leq \frac{2^n}{\sum_{r=0}^{\left[\frac{\delta-1}{2}\right]} \binom{n}{r}}$

b. $k(n,2\Delta) = k(n-1, 2\Delta-1)$

c. $\frac{1}{2} k(n+1,\delta) \leq k(n,\delta) \leq k(n+1,\delta)$

d. $k(n,1) = 2^n$

e. $k(n,2) = 2^{n-1}$

f. $2^{n-a} \leq k(n,3)$ waarbij a het kleinste gehele getal is $\geq \lg_2(n+1)$

g. $k(n,\delta) = 2$ als $\frac{2}{3} n < \delta$

h. $k(n,\delta) = 4$ als $\frac{2}{3} n = \delta$

Betrekking a bewijst men op de volgende wijze.
Allereerst de ondergrens voor $k(n,\delta)$. Voor het eerste getal van onze code hebben we een keuze uit 2^n elementen. Het tweede getal moet tot dit eerste getal een afstand δ hebben. Nu zijn er juist $\sum_{r=0}^{\delta-1} \binom{n}{r}$ getallen, die tot het eerste getal een afstand $\delta-1$ of minder hebben, zodat we voor het tweede getal de keuze hebben uit $2^n - \sum_{r=0}^{\delta-1} \binom{n}{r}$ getallen.

Voor het derde getal moeten we wederom een keuze maken uit deze $2^n - \sum_{r=0}^{\delta-1} \binom{n}{r}$ getallen, doch nu met de bijconditie , dat ook de afstand tot het tweede getal minstens δ moet zijn. We hebben nu dus minstens de keus uit $2^n - 2 \sum_{r=0}^{\delta-1} \binom{n}{r}$ getallen. Voor het m^e getal hebben we dus de keuze uit $2^n - (m-1) \sum_{r=0}^{\delta-1} \binom{n}{r}$ getallen. Zolang derhalve

$2^n - (m-1) \sum_{r=0}^{\delta-1} \binom{n}{r} > 0$ is, kunnen we nog een m^e getal toevoegen.

Voor ket $(k+1)^e$ getal is geen keus meer, derhalve $2^n - k \sum_{r=0}^{\delta-1} \binom{n}{r} \leq 0$

of $\frac{2^n}{\sum_{r=0}^{\delta-1} \binom{n}{r}} \leq k$.

Voor het tweede gedeelte van betrekking a overwegen we het volgende: Veronderstel $\delta = 2\Delta + 1$ dan luidt dus de bewering $k(n,2\Delta+1) \leq \frac{2^n}{\sum_{r=0}^{\Delta} \binom{n}{r}}$. Hebben we een code van k getallen geconstrueerd, dan inverteren we b.v. eens een van de kolommen van deze code. Dit geeft dan weer een bruikbaar stel van k getallen. Vervolgens inverteren we successievelijk alle andere kolommen. We hebben dan $\binom{n}{0} + \binom{n}{1}$ systemen gekregen. Daarna inverteren we uitgaande van de eerste code twee kolommen op alle mogelijke manieren, vervolgens drie kolommen, enz. t/m Δ kolommen. We hebben dan $\sum_{r=0}^{\Delta} \binom{n}{r}$ systemen van elk k getallen, dus $k \sum_{r=0}^{\Delta} \binom{n}{r}$ getallen, gekregen. We beweren nu, dat al deze getallen verschillend zijn. Voor zover twee getallen A en B uit eenzelfde systeem afkomstig zijn is dit duidelijk. Ze verschillen dan immers in minstens $2\Delta + 1$ digits. Behoren A en B tot verschillende systemen, dan zijn zij ontstaan uit de getallen A' en B' van het oorspronkelijke systeem. Er zijn nu twee mogelijkheden: A' = B' of A' $\neq$ B', in welk laatste geval A' en B' in minstens $2\Delta + 1$ digits verschillen. Ingeval A' = B' is direct duidelijk dat A $\neq$ B, daar beiden door verschillende inversies uit A' ontstaan zijn. Is A' $\neq$ B', dan bedenken we, dat het onmogelijk is twee getallen die in $2\Delta + 1$ digits verschillen gelijk te maken door van elk hoogstens Δ digits te veranderen.

Derhalve $k \sum_{r=0}^{\Delta} \binom{n}{r} \leq 2^n$ $\qquad$ $k \leq \frac{2^n}{\sum_{r=0}^{\Delta} \binom{n}{r}}$

We stellen ons nu voor $C(n-1,2\Delta-1,k)$ te hebben geconstrueerd. Alle getallen met een oneven aantal énen ("oneven" getallen) verschillen

onderling in minstens 2Δ digits, evenzo de "even" getallen. Door aan de oneven getallen een nul toe te voegen en aan de even getallen een één, ontstaat een code in n digits van getallen die onderling in minstens 2Δ digits verschillen; dus $k(n,2\Delta) \geq k(n-1,2\Delta-1)$. Omgekeerd kunnen we uit $C(n,2\Delta,k)$ door weglating van een willekeurige kolom een $C(n-1,2\Delta-1)$ construeren; derhalve $k(n,2\Delta) \leq k(n-1,2\Delta-1)$, dus $k(n,2\Delta) = k(n-1,2\Delta-1)$ (betrekking b).

Van betrekking c is het tweede lid triviaal en het eerste lid bewijzen we als volgt. Beschouw een willekeurige kolom van een $C(n+1,\delta)$. Neem aan dat zich in iedere kolom minstens evenveel nullen als énen bevinden. We schrijven alle getallen op, die in deze kolom een nul hebben en laten deze kolom vervolgens weg, waarmee we een $C(n,\delta)$ geconstrueerd hebben. Derhalve
$\frac{1}{2}\, k(n+1,\delta) \leq k(n,\delta)$.

Betrekking d is triviaal en e volgt daarna met behulp van b.

Betrekking f leiden we op de volgende wijze af. We kunnen de eis $\delta = 3$ aldus interpreteren, dat ieder getal herkenbaar moet blijven ook als één van de digits (correctiedigits incluis) gewijzigd wordt. We kunnen dit bereiken door de digits van onze code aan bepaalde betrekkingen (somcijferbetrekkingen) te laten voldoen.

Onder de aanname, dat maximaal slechts één fout gemaakt wordt, zal uit het al of niet voldaan zijn aan deze betrekkingen moeten blijken, welke digit gewijzigd is. Stellen we m betrekkingen tussen de digits vast, dan kunnen we ons doel bereiken door de digits d_0 t/m d_{m-1} elk in één der betrekkingen te laten optreden, de volgende $\binom{m}{2}$ digits in 2 der vergelijkingen, enz., de laatste digit in alle vergelijkingen. Op deze wijze kunnen we $\sum_{r=1}^{m} \binom{m}{r} = 2^m-1$ digits controleren. Hiervan zijn door de overige $2^m - (m + 1)$ bepaald. Deze laatsten zijn echter geheel vrij en genereren een code van $2^{2^m-(m+1)}$ getallen. Resumerende kunnen we dus met $2^m-1 = n$ digits een code van $2^{n-\lg_2(n+1)}$ getallen construeren. Uitbreiding tot $n \neq 2^m - 1$ ligt voor de hand. Van de juistheid van g en h overtuigt men zich gemakkelijk door inspectie. Op practische gronden zijn deze betrekkingen trouwens oninteressant.

Tenslotte kunnen we gemakkelijk laten zien dat voor geen enkele n en δ k = 3 kan zijn. We nemen nl. als eerste getal de configuratie bestaande uit louter nullen. Vinden we hierbij nog twee getallen a en b dan bevatten a en b beiden minstens δ énen evenals de carryloze som a + b. Dit getal kan dan als vierde element dienst doen. Op deze wijze redenerend komt men licht tot de overtuiging dat k een macht van twee moet zijn, immers, kunnen we bij de set 0, a, b, a + b nog een vijfde getal c vinden, dan vinden we in totaal 8 elementen nl. behalve de reeds genoemde nog a + c, b + c en a + b + c. Deze redenering is in zoverre onjuist dat zij gebaseerd is op de aaname van a + b als vierde element. Inderdaad komen ook waarden van k voor die geen machten van 2 zijn.

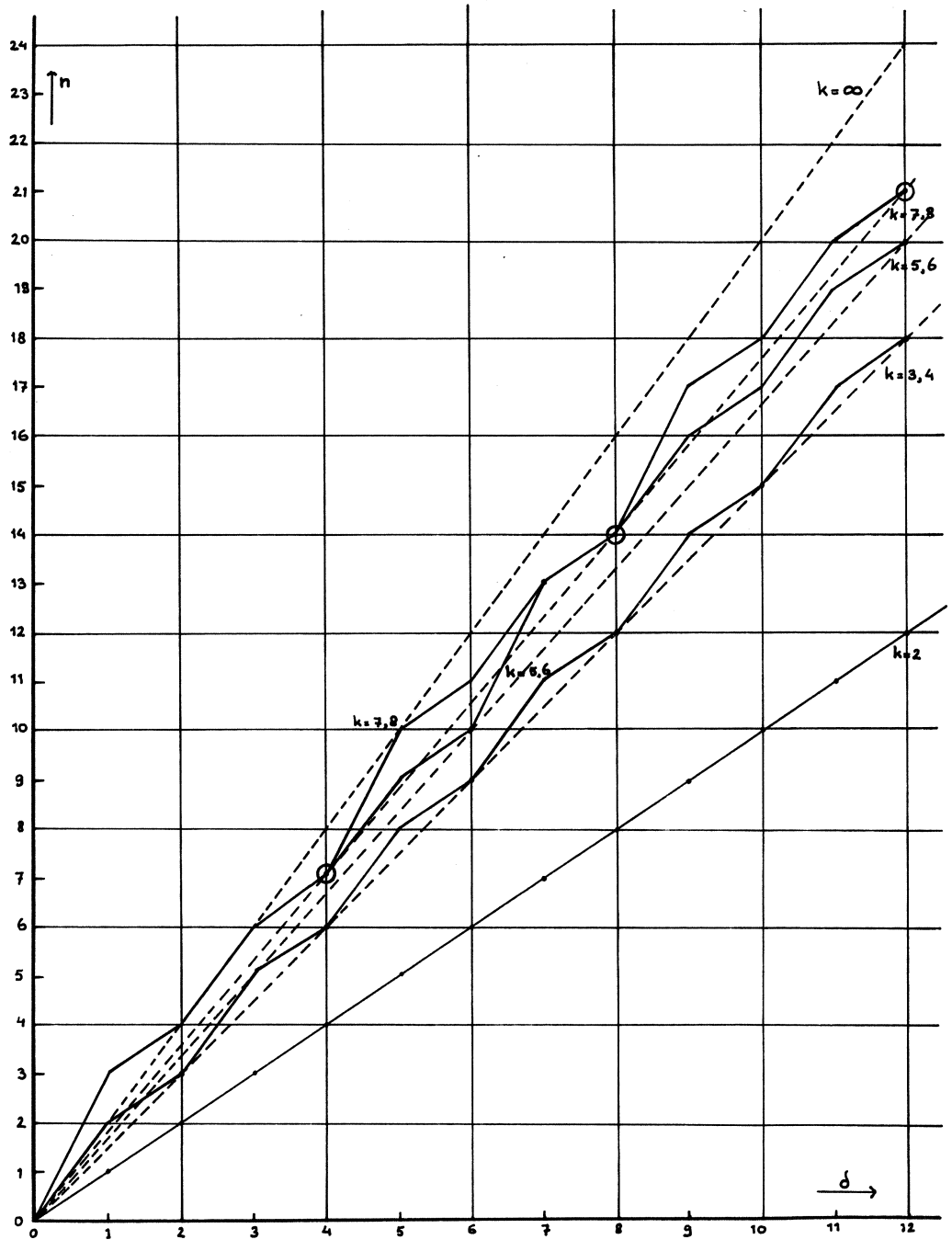
Een beter overzicht over het algemeen verband wordt verkregen wanneer we de vraag omdraaien en ons gaan interesseren voor n = n(k,δ), d.w.z. hoeveel digits zijn minimaal vereist om k getallen allen onderling een afstand δ of meer te geven. Te dien einde beschouwen we een kolom van de code. Deze bevat dus in totaal k nullen en énen. Het is duidelijk dat bv. een kolom met één nul en zeven énen (k = 8) al zeer ongeschikt is om de acht getallen van elkaar te onderscheiden. Een betere keus zou zijn vier nullen en vier énen. We preciseren dit nog wat nader. We moeten k getallen allen onderling in δ digits (of meer) laten verschillen, d.w.z. dat we op de een of andere wijze $\binom{k}{2}\delta$ inversies (= verschillen) moeten opbrengen. Is k = 21, dan kan een kolom l^2 dezer inversies leveren en het minimum aantal digits om ons doel te bereiken is dus minstens gelijk aan $\frac{\binom{k}{2}\,\delta}{l^2} = \frac{2l - 1}{l}\,\delta$ of $\frac{2l - 1}{l} \leq \frac{n}{\delta}$. Is k=2l-1 dan vindt men evenzo $n \geq \frac{\binom{k}{2}\,\delta}{l(l-1)}$ of eveneens $\frac{2l - 1}{l} \leq \frac{n}{\delta}$. We denken ons nu een grafiek van n als functie van δ van verschillende waarden van k. De "grensrechten" voor k = 2l en k = 2l - 1 vallen dan dus samen. We kunnen echter een reeks punten berekenen waar de actuele kromme voor een bepaalde k-waarde coincideert met de bijbehorende grensrechte. Voor k = 2l - 1 behoort het eerste punt van deze

reeks bij het geval waarin men alle mogelijke kolommen van 1 - 1 nullen en 1 énen (ten getale van $\binom{21-1}{1-1}$) naast elkaar opschrijft. We krijgen dan het punt $n = \binom{21-1}{1-1}$, $\delta = \binom{21-2}{1-1}$.

Voor k = 21 vindt men hetzelfde punt in verband met het feit dat men b.v. de kolommen bestaande uit 1 nullen gevolgd door 1 énen resp. 1 énen gevolgd door 1 nullen in dit verband als identiek mag beschouwen. Aangezien we het hier geschetste procédé in horizontale richting (uitbreiding naar hogere n bij constante k) kunnen voortzetten, vinden we de puntenreeks

$$n = i\binom{21-1}{1-1}, \quad \delta = i\binom{21-2}{1-1} \qquad i = 0, 1, 2, \ldots .$$

Deze grenspunten verdelen zowel de grensrechte als de gebroken lijn die n als functie van δ voorstelt in stukken. Het laat zich gemakkelijk inzien, dat elk volgend stuk minstens zo "goed" is als het voorafgaande. Immers bevat de grafiek het punt n,δ dan is in elk geval het punt $n + \binom{21-1}{1-1}, \delta + \binom{21-2}{1-1}$ te realiseren door de kolommen die het punt $\binom{21-1}{1-1}, \binom{21-2}{1-1}$ opleverden er eenvoudigweg naast te zetten. De opvolgende stukken naderen dus tot een limietvorm. Hulpmiddelen bij het construeren van de grafieken zijn behalve de grensrechten ook de gedeelten tussen $\delta = 2\Delta - 1$ en $\delta = 2\Delta$ die onder een hoek van 45^{o} met de assen verlopen en de betrekking $n(k,\delta_1 + \delta_2) \leq n(k,\delta_1) + n(k,\delta_2)$. In de grafiek zijn de grensrechten en de werkelijke krommen getekend voor k = 2 t/m 8. Het eerste punt op de kromme voor k = 7,8 waarvan we zeker zijn dat het op de grensrechte ligt is het punt n = 35, δ = 20. De kromme blijkt echter reeds bij n = 7,δ = 4 met de grensrechte te coincideren. Van daar af zijn alle stukken congruent met het eerste stuk.


n
δ
k=∞
k=7,8
k=5,6
k=3,4
k=2
k=7,8
k=5,6

BEREKENING VAN EIGENWAARDEN EIGENVECTOREN

J. Kruizinga

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 28 januari 1956 te Amsterdam.

Eigenwaarden van het probleem van het type

$$\vec{Ax} = \lambda\vec{x}$$

komen veel voor bij allerlei problemen in de physica, de chemie en de technologie (bij voorbeeld atoomtrillingen in moleculen, quantumchemie, trillingen van mechanische systemen).

Het probleem is, voor een gegeven matrix A die waarden van λ te bepalen, waarvoor het bovengenoemde stelsel een oplossing bezit en vervolgens voor ieder van deze eigenwaarden λ_i ook de bijbehorende eigenvectoren $\vec{x}_i$ te berekenen.

Gebruikmakend van een door Lanczos gepubliceerde iteratieve methode is voor de computer een programma ontwikkeld, dat de gebruiker der machine de eigenwaarden en eigenvectoren van een gegeven matrix A levert. Dit programma kan voor symmetrische matrices tot de orde 20 gebruikt worden.

De oorspronkelijk door Lanczos gegeven formulering voor enkelvoudige eigenwaarden kan gemakkelijke uitgebreid worden tot het geval van meervoudige eigenwaarden. De belangrijkste relaties van de Lanczos methode zijn de volgende:

a) $\vec{b}_{k+1} = A\vec{b}_k - \alpha_k\vec{b}_k - \beta_{k-1}\vec{b}_{k-1}$

b) $\alpha_k = \dfrac{\vec{b}_k \cdot (A\vec{b}_k)}{\vec{b}_k^{\,2}}$

c) $\beta_{k-1} = \dfrac{\vec{b}_{k-1} \cdot (A\vec{b}_k)}{\vec{b}^2_{k-1}}$

d) $P_0(\lambda) = 1$

e) $P_{m+1}(\lambda) = (\lambda - \alpha_m)P_m(\lambda) - \beta_{m-1}P_{m-1}(\lambda)$

f) $P_n(\lambda) = 0$

g) wortels van f): $\lambda_1, \ldots, \lambda_n$

h) $\vec{u}_j(\lambda_j) = \sum_{e=0}^{n-1} \dfrac{P_e(\lambda_j)}{\vec{b}^2_e} \vec{b}_e$.

Bij meervoudige wortels moet een nieuwe startvector geconstrueerd worden.

i) $\vec{c}_0 = \vec{e}_e - \dfrac{b_{0e}}{\vec{b}^2_0}\vec{b}_0 - \dfrac{b_{1e}}{\vec{b}^2_1}\vec{b}_1 - \ldots - \dfrac{b_{m-1e}}{\vec{b}^2_{m-1}}\vec{b}_{m-1}$.

De uitdrukkingen a) ... i) zijn zeer geschikt voor berekening met electrische computers. Voor een gegeven matrix A en een willekeurig gekozen startvector $\vec{b}_0$ kunnen dan achtereenvolgens berekend worden:

$\alpha_0,$ $\vec{b}_1$

$\alpha_1, \beta_0,$ $\vec{b}_2$

$\alpha_2, \beta_1,$ $\vec{b}_3$

.....

$\alpha_{m-1}, \beta_{m-2},$ $\vec{b}_m$

totdat voor een bepaalde waarde van $m \leq n$ blijkt dat $\vec{b}^2_m = 0$. Vervolgens worden de coëfficiënten van de polynomen e) berekend met behulp van de recursieve betrekking

c') $g_k = q_{k-1} - \alpha_e q_k - \beta_{e-1}P_k$.

Het laatste polynoom waarvoor de coëfficiënten op deze manier berekend worden is dan de karakt. vgl. $P_n(\lambda) = 0$ (of een deler ervan). Hiervan worden dan de wortels berekend.

De berekende grootheden worden tenslotte gesubstitueerd in h), hetgeen dan voor enkelvoudige wortels de volledige oplossing geeft. Heeft men nog niet de volledige oplossing, dan moet het iteratieprocédé herhaald worden met de nieuwe startvector i), die dan de ontbrekende eigenvectoren levert, of deel ervan in geval men nog hoge multipliciteit van de wortels heeft.

LOGICAL STRUCTURE OF THE ZEBRA

W.L. van der Poel

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 27 februari 1956 te Amsterdam.

A system for a simple computer

The logical design of a very simple computer is described.
This computer uses all the bits of the operational part of an instruction in a practical way, thus achieving an enormous flexibility. There is not built in a multiplier nor a divider. Repetition of an instruction is possible. The amount of hardware in the arithmetic unit and the control is reduced as much as possible to enable the machine to locate a great deal of its own troubles. The speed compares very favourably with that of computers in the same class.

In the research laboratory of the Netherlands Postal and Telecommunications Services we have an automatic computer, called PTERA, which has been in operation for three years. Its speed, however, fell short of most of the problems encountered lately. Therefore, we had to look for a design for a new machine. In the light of our experience this machine had to fulfil several requirements.

1. It had to be very simple in technical respect. This makes serial working almost a necessity. For storage a magnetic drum is the obvious solution.
2. It had to be rather quick. This can be achieved by including a few immediate access registers to the store.
3. The coding had to be extremely flexible. Indeed, for attaining the greatest simplification, most of the complicated operations are programmed instead of being built in. For example, even multiplication

and division are not built in.

This new machine is called ZEBRA (Zeer Eenvoudige Binaire RekenAutomaat = very simple binary automatic computer) and will be built for us by Standard Telephones and Cables Limited.

To begin with a few pertinent facts:

The storage capacity of the drum will be:

2^{13} = 8192 words located in 256 tracks of 32 words each. Each word contains 35 bits.

The speed of revolution of the drum is 6,000 rpm. That is equivalent to 10 ms/rev. or 312 μs/word. This 312 μs/word time is the basic cycle for all the operation.

15 short registers are provided as immediate access registers.

These 15 registers include the two accumulators.

The arithmetic unit consists of two accumulators of 33 bits each. They are called A and B. The contents of A and B can be read off from short registers 2 and 3 respectively.

Overflows from B can be brought into A, so that A and B together can serve as a double-length accumulator.

The control consists of two registers, C and D. C is the instruction staticizer for the instruction to be carried out next. D serves as an address-counter. This address-counter also contains a complete instruction, not only an address. The drum addresspart is augmented by 2 to allow for one word time between two successive instructions.

The structure of the instruction word is one of the characteristic and peculiar things of the machine. There are two addresses a drum address of 12 bits, a short address of 5 bits for determining the short register. The remaining 15 bits form the operation. All the bits of the operation part have a separate function. Thus no operation decoder is required. Learning the code consists of learning the functions of the separate bits, thus 2^{15} different instructions can be combined with them. The functional bits are denoted by letters. When the letter occurs in an instruction as it is written down on paper, the corresponding bit is 1 otherwise it is 0. An exception is made for the first bit, the A bit, when the bit is 0, it is written down as X instead of nothing.

The operation bits are denoted in sequence by A,K,Q,L,R,I,B,C,D,E,V,V_4, V_2,V_1,W. The function of the operation bits is:

The A-bit - When the A-bit is 0, the drum store is used for the control. This means that the drum is used as a source of a new instruction. Otherwise said: an X-instruction is a jump instruction.

When the A-bit is 1, the drum store is used for the arithmetic unit. This can be for a transport from the drum to accumulator or back; that depends on other functional digits.

The K-bit determines whether the short store is used for the arithmetic unit or for the control when the K-bit is 0. The selected short register is used in conjunction with the arithmetic unit. This can be for a transfer to as well as from the accumulator.

When the K-bit is 1, the selected short register is used for the control.

It is clear that there are four combinations of the bits A and K.

They have the following names:

00: Adding jump

01: Double jump

10: Double addition

11: Jumping addition

In the adding jump the drum store is giving the next instruction. At the same time the short store is used for an addition in the accumulator or possibly for a transfer back from accumulator to store.

The double jump uses both adresses as new instruction source. This is equivalent to modifying the instruction as it stands on the drum by the amount mentioned in the short store.

In the same way the double addition uses both stores for the addition, so two numbers can be added at the same time. A more important application is taking in a number from one of the stores and storing at the same time the number which was in the accumulator before in the other store.

The jumping addition can modify the address count by the contents of the short store while doing at the same time one addition from the drum store. Because the address counter not only contains an address but an

operation part as well, this is a very powerful method to make an active use of the cycle on which the next instruction is taken in. In fact there is a complete duality between adding jumps and jumping additions. The Q-digit, when present, adds unity in the B-accumulator independently of whether another addition is going on at the same time. In fact unity is introduced as if it were a carry from the "34th bit" of B.
The L-bit, when present, shifts the double-length contents of A and B together one place to the left. At the same time a number from the store could be added to the shifted contents. The R-bit, when present, shifts the double-length accumulators one place to the right in the same way. Shifting over from A to B or back does not take place when one of the accumulators is cleared.
The I-bit determines whether the arithmetic operations must be done positively or negatively. The I-bit only relates to additions in the accumulators.
The B-bit determines whether the A or the B-accumulator must be used. When the B-bit is 0, the A-accumulator is used and the B-accumulator is left undisturbed. Otherwise the B-accumulator is used.
The C-bit controls the clearing of the accumulator determined by the B-bit. The clearing takes place after storing, but before shifting. This is a consequence of a delay-type serial adder. The D-bit determines whether reading or writing takes place on the drum store. When B = 0 the contents of the drum store is read off and its contents are used according to the other functional digits. When the D-bit is 1, something is stored to the drum store from the accumulator determined again by the other functional digits.
The E-bit has the same function in relation to the short store as the B-bit has to the drum store.
The V, V_4, V_2 and V_1 bits serve for testing. Every instruction can be a test instruction. Testing is possible on the sign digits of both the accumulators, on the rightmost bit of the B-accumulator and on the position of several external selection switches. The instruction on which the test bits are present is only executed when the criterion to be tested is fulfilled. Otherwise the instruction is skipped completely. The W-digit

determines whether the drum is used or not. When W = 0 the drum is used and the instruction is always waiting until the selected drum location is reached. When W = 0 no waiting takes place and the drum is completely disregarded.
This last feature offers the possibility to repeat an instruction.
In the instruction word the short address is adjacent to the drum address. The last one is occupying the rightmost position of a word. When a jump is given to an instruction in a short register and W = 1, then the address counter is still augmenting the drum address by 2 every time. But this address is not active as such, so only the contents of the short register are obeyed. By placing an appropriate address in the jump instruction it takes a definite number of times until the down address flows over into the short address in the address counter. In this way the repetition of the instruction in the selected short location is ended and the next instruction is taken from the next short register. No special hardware is needed for it. A few examples of possible instructions are:

X200KE7: Jump to 200 and place the contents of the address counter in 7. This is useful for jumping to a sub programme and retaining the return instruction.

A200QBCE5R: Store the contents of the B-accumulator to 5. Take into B the contents of 200 and add unity. At the same time shift the A-accumulator one place to the right.

Repetition of ALQBC3V1: Shift the A-accumulator to the left, count the number of places shifted in B and test whether A has become positive. This is just the action called normalisation. It is very useful in conjunction with floating point operations.
In the same way multiplication, division, sideways addition, block transport from drum to short store and back, etc., can all be done with one single repeated instruction.
The main object of the whole design has been to simplify the inhomogeneous part of the logical design as much as possible. The homogeneous part of the hardware is mainly connected with the store and this is forming the largest portion of the material. For locating a trouble in

the homogeneous part of the machine only a few sections need to operate correctly and the machine can diagnose its own mistakes. Furthermore all the sections of the store are identical. For the sake of locating troubles in the inhomogeneous part which must be done manually, this part must be kept as small and as simple as possible.

The simplification of hardware has also been maintained in the input and output equipment. For input a Ferranti tapereader will be used. The holes of the tape are read off separately and they are assembled into pentads by programming. For output a teleprinter and a highspeed punch will be available. The 7-unit code teleprinter signal will be generated by a programme. The only thing which is built-in is a flip-flop which can be put high or low on selecting one of the spare short storage locations.

A few specific operation times are:

Optimum Multiplication	11 ms
Division	22 ms
Subprogramme for square rooting:	80 ms
Subprogramme for cosine rooting:	80 ms
Interpretative floating point operation:	30 ms per operation average.
Floating matrix inversion:	$50\ n^3$ ms where n is degree of matrix.

DE ARMAC
(AUTOMATISCHE REKENMACHINE MATHEMATISCH CENTRUM)

B.J. Loopstra

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 26 mei 1956 te Amsterdam.

Het is de bedoeling in deze lezing een korte beschrijving te geven van de structuur van deze nieuwe machine.
Als verklaring van een aantal van haar eigenschappen vermelden we allereerst de overwegingen welke bij de opzet van het ontwerp toonaangevend zijn geweest.
Gewenst werd een machine welke ca. 30-50 maal sneller zou zijn dan de ARRA met een opdrachtencode welke in principe van die van de ARRA (en de FERTA) niet veel zou verschillen zodat een zekere mate van continuiteit van de zijde der programmeurs bereikt zou worden. De machine diende voorts zo snel mogelijk beschikbaar te zijn, zodat in hoofdzaak gebruik gemaakt moest worden van constructiemethodes welke hun betrouwbaarheid reeds hadden bewezen. Daarnaast werd het met het oog op nieuwere technische ontwikkelingen gewenst geacht op kleine schaal voor geheugendoeleinden gebruik te maken van magnetische kernen.
Aan deze eisen werd voldaan door de huidige constructie welke in principe vrij klassiek kan worden geacht en welke tengevolge van haar deels experimenteel karakter geenszins de pretentie heeft als optimaal ontwerp in welke zin dan ook te gelden. Het feit dat de machine in iets meer dan een jaar tijds kon worden ontworpen, gebouwd en in bedrijf gesteld moge in dit opzicht echter als verontschuldiging gelden.
Ter orientatie laten we nu enkele specifieke gegevens volgen.

1. De machine werkt in het tweetallig stelsel met vaste komma
2. Woordlengte 34 bits (33 + teken)
3. 2 instructies van 17 bits (opdrachtgedeelte 5 bits, adres 12 bits) per woord

4. Geheugen
   a. Magnetische trommel met 128-16 = 112 tracks van 32 woorden
   b. Snel geheugen in de vorm van magnetische kernen: 16 tracks = 512 woorden + instructie buffer van 32 woorden
5. Instructiecode
   Min of meer in overeenstemming met die van FERTA. Geen deelopdracht. Wel transportopdrachten met behulp waarvan een gehele track uit langzaam naar snel geheugen kan worden gebracht of omgekeerd.
6. In- en uitvoer: Electrische schrijfmachine, telexponser (25 pentades/sec) en foto-electrische bandlezer.
7. Snelheid
   a. Korte opdrachten (optelling enz.): 416 µsec
   b. Vermenigvuldiging 5,4 msec.
   c. Transporten 14,6 msec.
   d. Aantal uitgevoerde instructies per seconde afhankelijk van de structuur van het programma tussen ca. 2500 en 1000.
8. Gebruikte apparatuur ca. 1200 buizen en 9000 germaniumdiodes.
9. Energieverbruik ca. 10kW.

Evenals de ARRA en de FERTA is de ARMAC een seriemachine. De grondfrequentie is bijna 100kHz. De tijdsduur der korte opdrachten is te verklaren door de omstandigheid dat voor elk woord op de trommel 40 bits beschikbaar zijn waarvan er slechts 36 worden gebruikt. Het feit dat we met een seriemachine te doen hebben heeft tot gevolg dat de mogelijkheden van het parallel werkende snelle geheugen lang niet ten volle worden benut.

Het meest opvallende en centrale punt der machine is de instructie-buffer waarvan we nu een nadere verklaring zullen geven.

Het is bekend dat de instructie-informatie bij vrijwel elk probleem een typisch geordende structuur vertoont, zulks in tegenstelling tot wat men bij de zuiver numerieke informatie dikwijls aantreft. Dit betekent dat wanneer men de instructie op adres p gebruikt de kans zeer groot is dat vervolgens de instructies op adressen p+1, p+2 enz. aan de beurt zullen komen. Indien we ons de instructies in eerste instantie op een magnetische trommel geplaatst denken en we willen het

nadeel van de lange accestijden welke met dit medium verbonden zijn zoveel mogelijk beperken zonder gebruik te maken van een of andere vorm van optimum-programmeren, dan kunnen we dit doen door een beperkte hoeveelheid snel geheugen voor opdrachten beschikbaar te stellen. Hierbij kunnen we dan nog twee verschillende wegen inslaan. We kunnen met behulp van een of andere transportinstructie de gewenste instructies in dit snelle geheugen plaatsen en deze vervolgens uitvoeren of we kunnen het zo trachten in te richten dat dit transport automatisch plaatsvindt wanneer de controle-organen van de machine een instructie op een bepaald adres op de trommel specificeren. Bij de keuze tussen deze twee mogelijkheden gelden de volgende overwegingen. Wanneer men een geprogrammeerd transport van instructies kiest vormen deze transporten, welke niets met de feitelijke berekening te maken hebben een tamelijk ernstige extra belasting van de programmeur: zijn instructiegeheugen is inhomogeen geworden en de samenwerking tussen de samenstellende delen komt geheel voor zijn rekening. Het bezwaar wordt groter naarmate het beschikbare snelle geheugen kleiner is daar de transporten dan veelvuldiger worden. Aan de andere kant zal bij een machine met snel zowel als langzaam geheugen waarschijnlijk toch een of andere vorm van transportopdracht aanwezig zijn zodat deze oplossing geen extra apparatuur kost.
Dit laatste kan niet gezegd worden van het automatisch transport. Het kwam ons evenwel voor dat de voordelen ervan zo groot waren dat deze tegen de kosten opwogen. De programmeur kan nu op enkele onbelangrijke uitzonderingen na zijn programma's zo inrichten dat hij net doet alsof al zijn instructies op de trommel staan en van daar uit worden gehoorzaamd: voor zijn doeleinden is het instructiegeheugen homogeen geworden. Wat in feite gebeurt is nu het volgende. Wanneer de instructie op adres p aan de beurt is inspecteert de machine of deze instructie wellicht in de instructiebuffer aanwezig is. Dit laatste zal het geval zijn wanneer de track van de trommel welke het adres p bevat de laatste track is geweest die (automatisch) naar deze buffer is gebracht. Dit kan de machine ontdekken doordat bij elk dergelijk transport het nummer van deze track intern bewaard blijft. Wanneer het tracknummer van het

instructieadres (inhoud opdrachtteller) overeenstemt met dit nummer dan staat vast dat de bewuste instructie in de buffer aanwezig is en deze wordt dan daaruit geextraheerd: het trommelgeheugen wordt niet geraadpleegd.
Er kunnen echter verschillende redenen zijn welke er toe leiden dat de genoemde overeenstemming niet bestaat. We noemen:

1. Men start de machine met de hand op een geheel nieuw punt.
2. Bij het uitvoeren der instructies is de laatste instructie van een track aan de beurt geweest en wordt de eerste van de volgende track verlangd (overloop) (Hierbij wordt dan dus aangenomen dat deze laatste instructie niet een gehoorzaamde sprong was naar een adres dat wel in de buffer aanwezig is).
3. De machine heeft een sprongopdracht uitgevoerd naar een adres dat niet in de buffer aanwezig is (z.g. langzame sprong).

In deze en dergelijke gevallen wordt dan automatisch de buffer gevuld met de track waarin de nieuwe instructie voorkomt, waarna de gewenste instructie uit de buffer wordt geextraheerd en de machine verder rekent. De oude bufferinhoud wordt nooit naar de oorspronkelijke track teruggebracht. Technisch gezien komt het automatische transport neer op het uitvoeren van een normale transportopdracht waarbij de gegevens voor het transport aan de opdrachtteller worden ontleend in plaats van aan het opdrachtregister. Een paar moeilijkheden verdienen nadere bespreking. Het komt dikwijls voor dat men wijzigingen wil aanbrengen op adressen welke zich op dat moment in de buffer bevinden en ook dat men getallen wil lezen uit dergelijke adressen.
Het laatste biedt geen serieuze moeilijkheden wanneer we voor getallen de conventie invoeren dat ze uit de buffer gelezen zullen worden wanneer ze zich daarin bevinden. Als ze er zich niet in bevinden worden ze hetzij uit het normale snelle geheugen, hetzij gewoon van de trommel gehaald. In geval van schrijven op een dergelijk adres ligt de zaak even anders. Wanneer de wijziging van zodanige aard is dat er geen bezwaar tegen bestaat dat deze bij overgang naar een andere track weer verloren gaat wil men deze om tijdsverlies te voorkomen liefst alleen in de

buffer aanbrengen. Het kan echter ook zijn dat de wijziging van meer permanente aard is en dan zal ook de oorspronkelijke informatie op de trommel gewijzigd moeten worden ook al kost dit extra tijd. Er zijn daarom twee soorten schrijfopdrachten. De normale schrijfopdracht wijzigt zowel de buffer als het overeenkomstige adres op de trommel. De buffer-schrijfopdracht wijzigt alleen de buffer. In vele gevallen kan men door van deze laatste soort gebruik te maken een zekere besparing bereiken op het gebied van "herzet" opdrachten: wanneer het stuk programma in de bewuste track opnieuw wordt gebruikt nadat eerst andere tracks aan bod zijn geweest wordt hij weer in de oorspronkelijke staat van de trommel of in de buffer gelezen.

De 12 beschikbare cijfers voor het adres kunnen 4096 verschillende geheugenplaatsen aangeven. De adressen 0 t/m 511 hebben hierbij betrekking op het snelle geheugen, de nummers 512 t/m 4095 op het trommelgeheugen. (De instructie-buffer is niet apart adresseerbaar, behalve in zekere zin in het geval van de bufferschrijfinstructies). In het voorgaande is steeds aangenomen dat we spraken over de adressen 512 e.v. Wanneer hetzij de instructie, hetzij het getal krachtens de betreffende aanduiding in opdrachtteller of opdrachtregister uit een der adressen 0-511 moet worden gehaald, gebeurt dit uiteraard direct: hierbij spelen noch trommel, noch instructie-buffer een rol (de laatste blijft ook geheel ongewijzigd). Hoewel het snel geheugen in de eerste plaats voor numerieke informatie bestemd is kan men er ook bijzonder vaak optredende programma-onderdelen in bergen: we denken hierbij bijvoorbeeld aan drijvende kommaroutines.

Opgemerkt kan nog worden dat door het gebruik van hetzelfde adresgedeelte der instructie voor langzaam en snel geheugen (het onderscheid ligt slechts in de numerieke waarde van het adres) alsmede van een ongeadresseerde instructie-buffer de combinatie der beide geheugentypes door de programmeur desnoods als één homogeen geheugen kan worden opgevat. Dit betekent alleen dat wanneer men dit doet het aldus geconstrueerde programma geheel correct uitgevoerd zal worden. Uit snelheidsoverwegingen verdient deze methode overigens natuurlijk in het geheel geen aanbeveling.

We behandelen nu in het kort nog enkele technische bijzonderheden van de ARMAC.
In de logische schakelingen van de machine wordt gebruik gemaakt van dezelfde diode-versterkertechniek welke in voorgaande machines werd toegepast. De betrouwbaarheid hiervan is ook bij de hier gebruikte hogere frequentie ten volle bevestigd geworden. Getracht wordt zoveel mogelijk, rekening te houden met de beperkingen welke aan deze techniek inhaerent zijn: de vertragingen welke in de circuits van nature optreden zijn geenszins verwaarloosbaar en dienen van meet af aan in de logische structuur der schakelingen hun rol te spelen. De verschillende tijdsignalen worden over het algemeen niet onafhankelijk van elkaar uit de direct van het klokpulsrad verkregen impulsen afgeleid. De signalen met een lagere herhalingsfrequentie worden door tellertrappen afgeleid uit signalen met hogere herhalingsfrequentie. Het zou dus onjuist zijn er op te rekenen dat de beginflanken van verschillende tijdsignalen welke logisch gezien zouden moeten samenvallen dit ook in feite zullen doen. Tijdmarkering dient dus steeds te geschieden door een signaal van hoge frequentie te combineren met een signaal van lagere frequentie dat logisch gezien eerder begint (er dus geen beginflank mee gemeen heeft). Als voorbeeld noemen we de z.g. cijfertijdsignalen. Elk der 32 getaltijden is verdeeld in 40 cijfertijden, genummerd (0-19)a en (0-19)b. Binnen elke cijfertijd (ca 10 µsec) vallen 3 impulsen welke D_1, D_2 en D_3 genoemd worden (elk ca. 3 µsec lang). Wanneer men nu het punt dat logisch gezien als 12ct D_1 aangegeven zou kunnen worden moet uitcoderen kan dus geen gebruik gemaakt worden van de signalen 12ct en D_1, omdat 12 ct door het optreden van vertragingen iets later begint dan D_1 zodat combinatie een signaal zou opleveren dat korter was dan 3 µsec hetgeen we niet willen tolereren. De uitweg uit deze moeilijkheid is gevonden door niet de cijfertijdsignalen zelf te maken doch alle (overlappende) combinaties van 2 opeenvolgende; zo is b.v. het signaal 112 ct beschikbaar dat te beschouwen is als combinatie van 11ct en 12ct. Bovendien zijn de D-impulsen in twee versies beschikbaar, n.l. de even en de oneven D-impulsen (bijv. D_{10} en D_{11}). De markering van 12ct D_1 gebeurt nu door te combineren 112ct met de D_{10} waardoor met alle moeilijkheden

rekening wordt gehouden. Door deze en dergelijke maatregelen zijn de eisen aan flanksteilheden der impulsen bijzonder klein geworden en is het zelfs bij de frequentie van ca. 100 kHz welke hier wordt gebruikt nog mogelijk van een normale kabeltechniek voor de bedrading gebruik te maken.
De vermenigvuldigtijd van 5,4 msec. wordt bereikt door bij elke slag van de vermenigvuldiging 3 cijfers van de vermenigvuldiger gelijktijdig te beschouwen. Hiertoe worden gedurende de vermenigvuldiging de drie beschikbare serieoptellers van A-register, S-register en opdrachtteller in serie geschakeld, zodat hiervoor betrekkelijk weinig extra apparatuur nodig is.
A- en S-register zelf bestaan in de ARMAC uit flip-flops. De reden hiervoor ligt in het feit dat pogingen welke destijds ondernomen zijn om kern-registers op deze frequentie te bedrijven niet volledig succesvol waren. Een bijkomend belangrijk voordeel van het flip-flop-arrangement is dat de inhoud der beide registers nu direct op neon-lampjes zichtbaar gemaakt kan worden, wat van groot belang is voor een snel overzicht over de gang van zaken bij het testen van nieuwe programma's.
De selectie-apparatuur voor de trommel is geheel electronisch en bestaat in hoofdzaak uit een matrix van schrijftransformatoren en (hoogvacuum)diodes. Er zijn geen bijzondere maatregelen getroffen om de schakeltijden van deze apparatuur bijzonder kort te houden: voor elke omschakeling wordt 1 getaltijd beschikbaar gesteld. Dit is bij ons systeem niet bezwaarlijk omdat de meeste trommelreferenties bestaan uit transporten van een volledige track waarbij de hierdoor verloren tijd slechts enkele procenten bedraagt.
De selectie van het snelle geheugen (inclusief de buffer) geschiedt door middel van selectiekernen. Aan elk woord is een dergelijke selectiekern toegevoegd: de uitleesoperatie berust daardoor niet op een coincidentie van twee stromen van precies gedefinieerde amplitude. Dit is wel het geval bij schrijven (schrijven omvat natuurlijk zowel het inbrengen van nieuwe informatie als het herstellen van uitgelezen informatie).

Het uitlezen en herstellen van een woord kost ongeveer 2 cijfertijden (20 μsec).
Zowel in het snelle geheugen als ook op de trommel wordt gebruik gemaakt van controlecijfers. Elk woord heeft twee controlecijfers: één voor iedere helft van het woord. Bij het uitvoeren van een instructie worden zowel de instructie zelf als het erdoor aangehaalde woord gecontroleerd en de machine stopt wanneer één van beide incorrect bevonden wordt, onder aangave van de aard van de fout. Daar de controle zelf als serie-operatie wordt uitgevoerd heeft de foutieve opdracht, of de bewerking met het foutieve getal dan inmiddels al plaats gevonden. Aan dit nadeel is zonder een dure parallel-controle-apparatuur helaas niet te ontkomen. De controles zijn ook vaak van toepassing op de normale transport-operaties: de machine stopt zodra een foutief getal gevonden wordt zodat men ook dan kan zien waar de fout is opgetreden.
De tijdsduur van de transportoperaties (ca. 14,5 msec) moet worden geinterpreteerd als één trommelomwenteling (13,3 msec) + 3 getaltijden. Deze tijd is constant omdat een transport van een track op elke willekeurige getaltijd kan beginnen (er is geen wachttijd). Wanneer een transport moet plaatsvinden leest de machine destand van de klokpulsteller af, start het transport en wacht tot deze klokpulstellerstand opnieuw bereikt wordt. Van de drie extra getaltijden is er één nodig voor het schakelen van de trommelselectie, één voor voorbereiding van het transport en één voor enkele nabewerkingen.
Van een nadere bespreking der opdrachtencode zien we hier af aangezien deze in een toekomstige lezing nog uitgebreid ter sprake zal komen.
Vgl. pag. 9 e.v. Wel willen we nog iets zeggen over enkele uitbreidingen welke de handbedieningsorganen hebben ondergaan in vergelijking met de vroegere machines. In de eerste plaats is toegevoegd een schakelaar welke bepaalt of alvorens de "gekozen" of "volgende" opdracht (van een trommeladres) wordt uitgevoerd of gelezen, de track waarin deze zich bevindt in elk geval in de buffer zal worden ingelezen van de trommel af of alleen indien ze er nog niet in aanwezig was. Een of andere faciliteit hiervoor zal immers nodig zijn omdat dezelfde track op de trommel en in de buffer niet identiek behoeven te zijn: Verder zijn

enkele knoppen ingevoerd waarmee men het in de "beginadres-schakelaars" of in de opdrachtteller gespecificeerde adres in het op lampjes zichtbare M-register kan halen zonder dat hierdoor de inhoud van enig machine-register verandert.
Bovendien is toegevoegd een 17-tal driestanden schakelaars genaamd "stop-opdracht-schakelaars" alsmede één tweestandenschakelaar getiteld "stop gekozen opdracht". Wanneer deze laatste schakelaar in de aan-stand is zal de machine stoppen zodra een opdracht in het opdrachtregister verschijnt welke met de 17 eerder genoemde schakelaars in elk cijfer overeenstemt. Overeenstemming wordt geacht te bestaan wanneer de opdracht énen (nullen) heeft waar ook de schakelaars énen (nullen) aanwijzen, terwijl bij cijfers waarvan de bijbehorende schakelaar in de middenstand (indifferent) staat, de overeenstemming altijd aanwezig wordt geacht. Met behulp van deze schakelaars kan men de machine bijv. laten stoppen op elke opdracht welke op een bepaald adres betrekking heeft of op opdrachten van bepaald type onafhankelijk van het adres enz.
Tenslotte rest ons nog te vermelden dat de afmetingen van de ARMAC ca. 3x2x2 meter zijn (afgezien van de voedingskast), binnen welk volume men dan de beschikking heeft over een electronische rekenmachine, ca. 4 m^2 nuttig tafeloppervlak, een tweeplanksboekenkast alsmede een kastje voor het opbergen van ponsbanden.

0/n (A) + (n) ≥ (A)
1/n (A) - (n) ≥ (A)
2/n + (n) ≥ (A)
3/n - (n) ≥ (A)
4/n + (A) ≥ (n); (n) $\geq$ + 0? of (A) $\geq$ + 0?
5/n - (A) ≥ (n); (n) $\geq$ + 0? of (A) $\leq$ - 0?
6/n spring naar n, a-opdracht
7/n spring naar n, b-opdracht
8/n (S) + (n) ≥ (S)
9/n (S) - (n) ≥ (S)
10/n + (n) ≥ (S)

11/n - (n) ≯ (S)

12/n + (S) ≥ (n); (n) ≥ + 0? of (S) ≥ + 0?

13/n - (S) ≥ (n); (n) ≥ + 0? of (S) ≤ - 0?

14/n als conditie positief, spring naar n, a-opdracht

15/n als conditie positief, spring naar n, b-opdracht

16/n (A) + (n).(S) ≥ (AS)

17/n (A) - (n).(S) ≯ (AS)

18/n + (n).(S) ≯ (AS)

19/n - (n).(S) ≯ (AS)

20/"n" Tracktransport Trommel → Matrix

21/"n" Tracktransport Matrix → Trommel

22/n link ≯ (A), spring naar n, a-opdracht

23/n link ≯ (A), spring naar n, b-opdracht

24/n
⋮
29/n } Schuif- en communicatieopdrachten.

ad "20" en "21": de opdrachten voor de tracktransport gebruiken de registers A en S niet, edoch steeds een kanaal (matrix) uit het snelle geheugen en een kanaal (spoor) op de trommel. De inhoud van het ene kanaal wordt in het andere gecopieerd, het functiegedeelte bepaalt de richting van het transport, het numeriek gedeelte als volgt, op welke kanalen het transport betrekking heeft: de hoogste 7 cijfers van het "adres" bepalen het spoor op de trommel (het getal, door deze gevormd, moet dus minstens 16 zijn) de laagste 4 bepalen het nummer van het snelle kanaal. Hiertussen staat een ongebruikt cijfer.

ad "22" en "23": dit zijn de zg. "subroutineaanroepen": de opdrachtteller wordt uitgelezen: in de a-helft van het A-register komt de 6- of 7-sprong naar de onmiddellijk volgende opdracht, in de b-helft worden 17 nullen geplaatst. Vervolgens wordt de sprong naar de a- resp. b-opdracht van adres n uitgevoerd.

ad "24" t/m "29": bij de schuif- en communicatie-opdrachten is het numeriek gedeelte nooit een adres, maar omlijnt het anderszins de

functie. Wij noteren hier de opdrachten in overeenstemming met de standaardponsconventies.

Nultest op ongetekende nul

28	0	XO: A≠0?	28	0	X8: S≠0?
29	0	XO: A=0?	29	0	X8: S=0?

Handregister H (komt alleen in het invoerprogramma voor)

24	1	XO: (A) + (H) ≽ (A)	24	1	X8: (S) + (H) ≽ (S)
25	1	XO: (A) - (H) ≽ (A)	25	1	X8: (S) - (H) ≽ (S)
26	1	XO: + (H) ≽ (A)	26	1	X8: + (H) ≽ (S)
27	1	XO: - (H) ≽ (A)	27	1	X8: - (H) ≽ (S)

Getalschakelaar G

24	2	XO: (A) + (G) ≽ (A)	24	2	X8: (S) + (G) ≽ (S)
25	2	XO: (A) - (G) ≽ (A)	25	2	X8: (S) - (G) ≽ (S)
26	2	XO: + (G) ≽ (A)	26	2	X8: + (G) ≽ (S)
27	2	XO: - (G) ≽ (A)	27	2	X8 - (G) ≽ (S)

Bandlees en -ponsopdracht

24	4	XO: (A) + (B) ≽ (A)	24	4	X8: (S) + (B) ≽ (S)
25	4	XO: (A) - (B) ≽ (A)	25	4	X8: (S) - (B) ≽ (S)
26	4	XO: + (B) ≽ (A)	26	4	X8: + (B) ≽ (S)
27	4	XO: - (B) ≽ (A)	27	4	X8: - (B) ≽ (S)
28	4	XO: + (A) ≽ (U) ≽ (B); (A) ≥ + 0?	28	4	X8: + (S) ≽ (U) ≽ (B); (S) ≥ + 0?
29	4	XO: - (A) ≽ (B) ≽ (B); (A) ≤ - 0?	29	4	X8: - (S) ≽ (U) ≽ (B); (S) ≤ - 0?

Opm.: De bovenste vier regels zijn bandleesopdrachten: het "symbolische" register B bevat aan de hoge kant 29 nullen, de laagste vijf cijfers worden van de telexband gelezen, die daarna een pentade opschuift. De onderste twee zijn band-pons-opdrachten: al of niet met tekenwisseling (0-1 inversie) worden de laagste vijf cijfers van A of S geponst. Dit gaat via een tussenregister U, dat de laagste zes cijfers van A of S

overneemt.

Deze opdrachten zetten tevens de conditie.

Typ- en terugleesopdrachten

24	8	XO:	(A) + (U) ≥ (A)	24	8	X8:	(S) + (U) ≥ (S)
25	8	XO:	(A) - (U) ≥ (A)	25	8	X8:	(S) - (U) ≥ (S)
26	8	XO:	+ (U) ≥ (A)	26	8	X8:	+ (U) ≥ (S)
27	8	XO:	- (U) ≥ (A)	27	8	X8:	- (U) ≥ (S)
28	8	XO:	+ (A) ≥ (U); (A) ≥ + 0?	28	8	X8:	+ (S) ≥ (U) (S) ≥ + 0?
29	8	XO:	- (A) ≥ (U) (A) ≤ - 0?	29	8	X8:	- (S) ≥ (U) (S) ≤ - 0?

Opm.: De onderste twee regels zijn de typopdrachten: er wordt een symbool getypt, gespecificeerd door de laagste 6 cijfers van ± (A) of ± (S). Zij zijn weer conditiezettend. De eerste vier regels beschrijven de mogelijkheid om het laatst getypte (of geponste) symbool voor controledoeleinden terug te lezen.

Conditionele stopopdrachten

26 16 XO (en 26 16 X8): stop, als de conditie positief is.

27 16 XO (en 27 16 X8): stop, als de conditie negatief is.

Bufferschrijfopdrachten op plaats n in de buffer (0 ≤ n ≤ 31)

28	n	X2:	+ (A) ≥ buffer; (A) ≥ + 0?	28	n	X10:	+ (S) ≥ buffer; (S) ≥ + 0?
29	n	X2:	- (A) ≥ buffer; (A) ≤ - 0?	29	n	X10:	- (S) ≥ buffer (S) ≤ - 0?

De adresloze optelling (0 ≤ n ≤ 127)

24	n	X4:	(A) + n ≥ (A)	24	n	X12:	(S) + n ≥ (S)
25	n	X4:	(A) - n ≥ (A)	25	n	X12:	(S) - n ≥ (S)
26	n	X4:	+ n ≥ (A)	26	n	X12:	+ n ≥ (S)
27	n	X4:	- n ≥ (A)	27	n	X12:	- n ≥ (S)

De schuifopdrachten

De cijfers in een bepaald "schuifcircuit" (zie onder) worden een bepaald aantal plaatsen naar rechts geschoven; dit kan op vier wijzen, bepaald door het functiecijfer:

f = 24, de additieve schuif: de cijfers, die aan de lage kant het circuit verlaten, komen aan de hoge kant in het tekencijfer weer binnen;

f = 26, de schone schuif: de cijfers, die aan de lage kant het circuit verlaten, gaan verloren; aan de hoge kant wordt met het tekencijfer aangevuld.

f = 28, de pos. uit-schuif: arithmetisch als bij f = 24, maar tevens wordt de conditie gezet op het nieuwe tekencijfer van het circuit.

f = 29, de neg. uit-schuif: arithmetisch als bij f = 24, maar tevens wordt de conditie gezet op de inverse van het nieuwe teken van het circuit.

Het aantal cijferposities, waarover geschoven wordt, geeft men aan in het plaatsgedeelte van de opdracht; het aantal is maximaal 35. Het einde van de opdracht, (de sluitletter X en) de kanaalcorrectie, bepaalt het circuit; er zijn vier mogelijkheden:

X20: A → A: het circuit begint bij het tekencijfer van A en eindigt aan de lage kant van A.

X22: S → A: het circuit begint bij het tekencijfer van S; aan de lage kant van S wordt het voortgezet met het hoogste (= teken-) cijfer van A en het eindigt aan de lage kant van A. ("Schuif van S naar A door het teken van A heen".) Het tekencijfer van S fungeert als tekencijfer van het circuit.

X28: A → S: het circuit begint bij het tekencijfer van A, aan de lage kant van A wordt het voortgezet met het op een na hoogste cijfer van S en het eindigt aan de lage kant van S. ("Schuif van A naar S, onder het teken van S door") Het tekencijfer van A fungeert als tekencijfer van het circuit. Het tekencijfer van S blijft ongewijzigd.

X30: S → S: het circuit begint bij het tekencijfer van S en eindigt aan de lage kant van S.

HET COMMUNICATIEPROGRAMMA VAN DE ARMAC

E.W. Dijkstra

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 30 juni 1956 te Amsterdam.

Om invoer en uitvoer bij de ARMAC te verzorgen, bevindt zich een complex van programma's en subroutines permanent in het geheugen; zij worden tezamen aangeduid als "het communicatieprogramma". Het ligt niet in de bedoeling, in dit colloquium het communicatieprogramma - dat circa 750 opdrachten omvat - in detail te behandelen. Wij moeten volstaan met enkele grepen.

Om informatie in grote hoeveelheden in de ARMAC in te voeren, ponst men deze in geschikte code op een telexband. De fotoelectrische bandlezer is dan ook het belangrijkste invoermechanisme van de machine, gelukkig evenwel niet het enige. Als men een enkel "woord" informatie wil inbrengen, zou het wat omslachtig zijn, om daarvoor eerst helemaal een bandje te moeten gaan ponsen. Als het wijziging van een opdracht betreft, staan de operateur 34 twee-standen-schakelaars ter beschikking. De ARMAC is een zuiver binaire machine met een woordlengte van 34 cijfers; om de gewijzigde opdracht op deze schakelaars op te zetten, vereist slechts weinig oefening. Om met de hand een (decimaal gegeven) getal in te brengen, zijn deze 34 schakelaars, hoewel in principe bruikbaar, natuurlijk minder geschikt. Daarom bevindt zich op het bedieningspaneel tevens het zg. handregister. Dit bestaat uit 10 + 4 toetsen: de 10 "cijfertoetsen" dragen de opschriften 0 t/m 9, de 4 "tekentoetsen" de opschriften +, -, +. en -. (Zoals bekend hangt de correlatie tussen een decimale en een binaire cijferrij af van de plaats, waar de komma geinterpreteerd wordt. In overeenstemming met de twee gangbare interpretaties kan men gehele getallen of echte breuken inbrengen; vandaar

de vier tekentoetsen.) Het handregister stelt de operateur in staat, een decimaal getal geconverteerd in een van de registers van het arithmetisch orgaan te brengen. De decimaal-binale conversie is geprogrammeerd: het "handregister-programma" vormt een onderdeel van het communicatieprogramma.

Ingebouwd is, dat zodra een van de toetsen van het handregister is ingedrukt, (en weer losgelaten), de machine start op een vast punt (nl. de eerste opdracht van het handregisterprogramma); tevens is voor dit doel ingebouwd een opdracht, die uitleest, welke toets van het handregister is ingedrukt (voor de cijfertoetsen het binair aequivalent van de opschriften, voor de tekentoetsen resp. 10, 11, 12 en 13).

Om een getal met het handregister in te brengen, drukt men eerst op een tekentoets. De machine start, verzorgt de nodige voorbereidingen en stopt (binnen een fractie van een seconde) weer. Dan drukt men de successieve decimale cijfertoetsen in. Steeds stopt de machine, en wordt deze weer gestart door het indrukken van het volgende decimale cijfer. Als conventie hebben wij gekozen, dat na het stoppen na het laatste cijfer de machine nog een keer moet worden "doorgestart", om de tekentoets in rekening te brengen; als de machine dan weer gestopt is, staat het bedoelde getal correct geconverteerd en met het goede teken in het register. Als het handregister uitsluitend voor invoer van getallen gebruikt werd, was deze conventie natuurlijk nodeloos omslachtig: men kan immers het handregisterprogramma zo maken, dat na het stoppen steeds het getal (goede teken etc.) in het betroffen register was geplaatst "voor zover aangeslagen". Het "afmaken" is dan niet meer nodig. Nu echter hebben we bereikt, dat het proces der decimale conversie bij het handregister steeds door een tekentoets wordt voorafgegaan (ingeluid) en door het afmaken wordt gevolgd (afgesloten). Dit stelt ons in staat om ingedrukte cijfertoetsen, niet door een teken voorafgegaan, op geheel andere wijze te verwerken. Een geprogrammeerde twee-standen wissel wordt door een tekentoets de ene kant uitgezet - naar de decimale conversie -, bij het afmaken wordt deze wissel teruggezet naar de zg. "autostart" -. Voor de andere verwerkingsmogelijkheid der cijfertoetsen hebben wij nl. gekozen het starten van de machine op vaste(gekozen) punten in het

communicatieprogramma. Standaardverrichtingen worden nu door de machine uitgevoerd na het indrukken van een enkele cijfertoets. Van de 10 mogelijkheden zijn er inmiddels 9 benut. Wij noemen: het uittypen van een registerinhoud als breuk of geheel getal, het testen van het trommelgeheugen, het binair uitponsen van een (gekozen) stuk geheugen, en het starten van het invoerprogramma om band te gaan lezen.

Resumerend: het handregisterprogramma reageert tweeerlei: op de cijfertoetsen, al naar gelang van de voorgeschiedenis. Deze mogelijkheid is hier verkregen ten koste van de verplichting het getal "af te maken". Een andere mogelijkheid zou geweest zijn, het aantal cijfers van getallen vast te leggen, zodat het handregisterprogramma tellenderwijs kan constateren, wanneer het laatste cijfer gelezen is. De beperkingen, die voor de operateur daaruit voortvloeien, leken ons hinderlijker.

De inrichting van het handregisterprogramma is hier vrij uitvoerig beschreven, omdat hier in eenvoudige vorm geillustreerd zijn de technieken, waarvan ook het bandleesprogramma veelvuldig gebruik maakt. In plaats van 14 staan ons hier 32 symbolen ter beschikking, anderzijds zijn de functies, die van het bandleesprogramma verlangd worden gevarieerder. Het is o.a. de taak van het bandleesprogramma groepjes pentades (rijtjes van vijf al of niet gaatjes) te assembleren tot woorden. Evenals het handregister, zoals wij gezien hebben, de informatie, welke cijfertoets is ingedrukt, verschillend kan verwerken, kan het bandleesprogramma het woord volgens verschillende regels uit de successieve pentades opbouwen. Deze regels corresponderen met de ponsconventies, die voor de verschillende "soorten" woorden gelden. Als verschillende soorten woorden kent het bandleesprogramma o.a. opdrachten, getallen en (door de machine) binair geponste woorden. De ponsconventies voor elk soort afzonderlijk konden vrij efficient gekozen worden, dankzij het feit, dat een dergelijk woord pas gelezen wordt als al bekend is, van wat voor soort het zal zijn. Dit is bereikt ten koste van de verplichting elke keer als de soort wisselt dit expliciet op de band te vermelden. In plaats van te specificeren dat de nu volgende zoveel woorden (een aantal) als getallen uit de pentades opgebouwd

moeten worden, wordt aangegeven dat er tot nader aankondiging getallen worden gelezen. (Dit geheel analoog aan de situatie bij het handregister waardoor de tekentoets de nu volgende cijfertoetsen tot nader aankondiging (nl. het "afmaken") in de decimale opbouw verwerkt zullen worden).

Over de verschillende soorten woorden enkele opmerkingen. Het functiegedeelte van opdrachten in de ARMAC loopt van 0 t/m 29. Het was - deels op historische gronden, die bestaan, omdat de ARMAC een soort snelle ARRA is - gewenst, elke opdracht te beginnen met het functiecijfer en dit te ponsen in één pentade. Als de soort gespecificeerd is als opdrachten zijn dus slechts twee pentades (nl. 30 en 31) niet een openingspentade van een opdracht. De 31 (= vijf gaatjes) wordt als erase geskipt: de 30 blijft over om aan te kondigen "dat er iets aan de hand is". De toetsen van de bandlezer dragen de opschriften 0 t/m 31; vanaf 10 tevens nog een nevenopschrift. Wij achtten het nl. een vereiste dat banden zonder meer van de vellen van de programmeur geponst zouden kunnen worden. Het dubbelbenoemd zijn van vele toetsen was een goedkope methode om een grote winst te boeken wat betreft de overzichtelijkheid van de programma's (en de uitspreekbaarheid van standaardcombinaties). De toetssymbolen zijn de volgende (het nevensymbool is tussen haakjes aangegeven)

0	8	16 (A)	24 (J)
1	9	17 (B)	25 (K)
2	10 (OO)	18 (C)	26 (L)
3	11 (OOO)	19 (D)	27 (T)
4	12 (+)	20 (E)	28 (P)
5	13 (-)	21 (F)	29 (S)
6	14 (+.)	22 (G)	30 (R)
7	15 (-.)	23 (H)	31 (X)

Wat wij zojuist over de openingspentades van opdrachten opmerkten wordt nu gereformuleerd als: "aan het begin van elke opdracht, worden eventuele extra pentades X geskipt", en "elke controlecombinatie begint met een R". Als regel wordt deze R nog door één letter gevolgd.

De technische constructie van de ARMAC (met name de buffer) dwingt

de programmeur - tenzij deze geheel zonder scrupules is - tot enige trackbewustheid en maken technieken voor drijvende addressering minder aantrekkelijk. Vandaar dat het functiecijfer door een "klassiek" adres gevolgd wordt.
Daartegenover staan gelukkig voordelen: de standaard programmabladen bieden juist ruimte aan één track programma (= 32 woorden = 64 opdrachten). De semi-binaire schrijfwijze voor adressen (nummer van de track en plaats in de track), die zoals bekend "peeping" aanzienlijk vergemakkelijkt, kon met enkele verfraaiingen zonder meer overgenomen worden uit de ARRA-conventies (die dateren uit de tijd dat de tracks via relais geselecteerd werden en trackwisseling dus tijd kostte!)

Het adres begint met de plaats in de track, gevolgd door een sluitletter (A t/m X, dus bandwaarde 16 en hoger). Hoewel in het merendeel der gevallen de plaats maximaal 31 is (en deze dus met één pentade geponst moet kunnen worden), is voor het inbrengen van schuifopdrachten, adresloze optellingen etc. gewenst, dat de "plaats", (die hier groter dan 31 kan zijn), een decimaal getal van variabel aantal cijfers is. Hier vloeit de ponsconventie uit voort dat de plaats, indien = 0, als 0 geponst moet worden, dat verder nonsignificante nullen aan het begin weggelaten mogen worden. Als de pentades voor de plaats aan de beurt zijn, leest het bandleesprogramma de eerste pentade ongetest, leest dan decimaal bij, totdat de sluitletter gelezen is.

Het tweede soort woorden, dat door het bandleesprogramma verwerkt kan worden, zijn getallen. Zij hebben een variabel aantal cijfers (breuken worden in maximaal 9 cijfers achter de komma ingevoerd), het einde van het vorige getal wordt aangegeven door het teken van het volgende (of door een X, die aan het begin van elk getal getypt wordt, of door de R van een controlecombinatie).

Een enkele programmatechnische bijzonderheid moge volgen. De interne "lees getal subroutine" neemt aan, dat het teken al gelezen is en op een daarvoor gereserveerde geheugenplaats staat. Dit is echter niet het geval voor het eerste getal na controlecombinaties. Bij het verwerken van deze controlecombinaties wordt daarom 31(=X) op de voor

het teken gereserveerde plaats gezet; wordt de controlecombinatie nu door een getal gevolgd, dan wordt door de "lees getal subroutine" deze 31 als (mogelijk) teken aangehaald en na analyse als X geskipt; het teken wordt alsnog gelezen.

Een tweede complicatie is het variabel aantal cijfers van de breuk. Omdat de ARMAC niet kan delen, is het zinloos om, met de decimale cijfers, een 10-macht op te bouwen; het is beter, de cijfers te tellen, en na het laatste cijfer een complementerend aantal malen met 10 te vermenigvuldigen; daarna wordt door 10^9 gedeeld (met 10^{-9} vermenigvuldigd). Dit wordt in het programma bereikt door bij breuken de decimale opbouw niet met nul maar met een vast klein getalletje te beginnen; dit wordt dan bij elk decimaal cijfer met 10 vermenigvuldigd en is zo gekozen, dat het met 10^9 vermenigvuldigd, juist capaciteitsoverschrijding geeft. Op deze capaciteitsoverschrijding test het programma, dat met factoren 10 bijvermenigvuldigt.

Naast de soortspecificatie houdt het bandleesprogramma de plaats van wegbergen bij, d.w.z. tenzij anders gespecificeerd wordt elk volgend woord op een volgend adres geborgen. Dit kan onderbroken worden door de zg. adresindicatie (begin daar en daar te schrijven) of de skipcombinatie (laat zoveel plaatsen open en ga door). Beide laten de soortspecificatie onveranderd, terwijl ook omgekeerd de soortspecificatie de plaats van wegbergen onbeinvloed laat.
Tenslotte kan het bandleesprogramma de geassembleerde woorden schrijven of vergelijken met wat er al staat. In geval van verschil stopt de machine. Of het bandleesprogramma in de schrijf- dan wel in de controlestand staat, wordt door soortspecificaties en adresindicaties niet beinvloed.

Er is voorzien in twee methoden van uitvoer: binair uitgeponste band, en decimaal uitgetypte getallen. Het typprogramma is uit de aard der zaak het gecompliceerdste.

Het typen van een getal wordt beheerst door een codewoord ter lengte van maximaal 34 bits. Het codewoord specificeert:

a. Waar in het uit te typen binaire woord de komma geinterpreteerd wordt; hier is voorzien in de twee standaardmogelijkheden: als

geheel getal en als (echte) breuk.

b. Of het teken getypt wordt, of uitsluitend de absolute waarde.

c. Welke decimale cijfers facultatief, imperatief dan wel niet getypt moeten worden (facultatief= nonsignificante nullen door spaties vervangen).

d. Waar punten of spaties ingelast moeten worden.

e. Of het getal na afloop misschien door een spatie of een tabulatorsignaal gevolgd moet worden.

Het communicatieprogramma kent 10 aanroepen van de typsubroutine: voor elke te gebruiken aanroep wordt bij de invoer een typcodewoord ingebracht. Deze aanroepen halen hun eigen codewoord aan en gaan daarmee naar de centrale typroutine, die het getal typt volgens dit codewoord. Alle naar de schrijfmachine gezonden signalen worden teruggelezen; met behulp hiervan wordt een controleresultaat opgebouwd. Als de controle faalt wordt op de volgende regel het mislukte getal overgetypt. Hieruit volgt, dat de typroutine gebonden is aan de regel-lay-out. Dit is nu uitgebreid tot pagina-layout. De faciliteiten, die het layout-programma bieden, lijken aan de meeste behoeften op een vanzelfsprekende wijze te voldoen. Om codewoorden in te kunnen voeren is een nieuw "soort" ingevoerd. Dit wordt aangekondigd door zijn eigen controlecombinatie; dit stuk van het bandleesprogramma bouwt het codewoord op uit een symbolenrij, die gemakkelijk door de programmeur opgesteld kan worden.

Door een van de controlecombinaties (RJ), gevolgd door een sprongopdracht wordt deze sprongopdracht uitgevoerd zonder dat aan een van de heersende specificaties iets veranderd is.
Er zijn bij de controlecombinaties nog ettelijke ongebruikte. Het bandleesprogramma stopt op alle ongebruikte combinaties. Door deze stopopdrachten door sprongen te vervangen, kunnen er gemakkelijk faciliteiten aan toegevoegd worden. (B.v. het lezen van floating getallen!) Ook in andere opzichten kijkt het leesprogramma na, of niet tegen de ponsconventies gezondigd is. Als van tevoren de specificatie van de soort, of de indicatie, waar her opgebouwde woord thuishoort, achterwege mocht blijven, stopt de machine. Het zwaarst is de controle bij de subroutine

die de typcodewoorden leest. Omdat de programmeur dit maar een paar keer per probleem zal doen, is de kans op vergissingen hier immers het grootst.

We hebben geprobeerd, een communicatieprogramma te construeren, dat niet meer gewijzigd hoeft te worden, dat echter wel gemakkelijk kan worden uitgebreid.

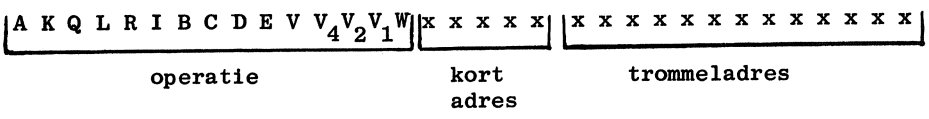
DE INVOERORGANISATIE VAN DE ZEBRA

W.L. v.d. Poel

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 29 september 1956 te Amsterdam.

1. Inleiding

De belangrijkste eigenschappen van de ZEBRA zijn de volgende: Het geheugen bestaat uit een magnetische trommel met een geheugencapaciteit van 8192 woorden verdeeld over 256 sporen van ieder 32 woorden. Elk woord bevat 33 binaire cijfers. De trommel maakt 6000 omw/min., d.i. 10 ms/omw. Elk woord vereist dus 1/32 × 10 ms = 312 μs. Dit heet de woordtijd.
Naast het trommelgeheugen bevindt zich een snel geheugen waarvan de registers alle in een woordtijd bereikbaar zijn. Physisch is dit geheugen uitgevoerd als vertragingslijnen op de trommel. We noemen het snel geheugen. De registers zijn genummerd van 0 tot 31. Alleen 4 tot 15 zijn werkelijk als registers aangebracht, de andere adressen dienen voor de bediening van enkele bijzondere functies en voor invoer en uitvoer. (O.a. (0) = 0, (1) = 1, 2 en 3 zijn de beide accumulatoren enz.)
Het rekenorgaan bestaat uit twee accumulatoren, A en B genaamd. Deze zijn bruikbaar als aparte accumulatoren maar zij kunnen ook gekoppeld worden bij schuiven en optellen tot een dubbele lengte accumulator.
De besturing bestaat uit twee registers C en D. Ruwweg is C het register waar steeds de volgende uit te voeren instructie komt en D de adresteller.
De structuur van de instructie is als volgt:



Hierin zijn de letters A tot W de aanduidingen voor de operatie. De letter wordt alleen dan geschreven als het bit in het woord op die plaats 1 is; anders wordt hij weggelaten. Alleen voor het A-cijfer bestaat de uitzondering dat hiervoor X geschreven wordt als dit cijfer 0 is.
De bits van de operatie hebben alle een functionele betekenis. Zo is bijv. L: schuiven naar links over een plaats, R: schuiven naar rechts, Q: tel een eenheid in de B-accu op, I: doe de bewerking negatief, D: berg op naar de trommel en lees niet, enz.

2. De code op de band

Een instructie in de ZEBRA is normaal samengesteld uit functionele letters en twee adressen. Het trommeladres wordt steeds geschreven als een getal van 3 of meer cijfers of als een getal groter of gelijk aan 32.
Adressen kleiner dan 32 zijn korte adressen. Verder komt het trommeladres eerder dan het kortadres en de functionele letters mogen op elke gewenste manier hiertussen, voor of achter staan. Bijv.:

A300BCE4	Functionele letters A, B, C en E, Adressen: 300 en 4
X8100QBC	Functionele letters géén, A, K en E. Kort adres ontbreekt dus 0
A5	Functionele letter A. Trommeladres ontbreekt.

Om deze instructies in te kunnen zetten bedient de band zich van de volgende 32 symbolen:

0	0	16	B
1	1	17	C
2	2	18	D
3	3	19	E
4	4	20	T
5	5	21	U
6	6	22	V
7	7	23	N
8	8	24	A
9	9	25	X
10	K	26	+
11	Q	27	-

12	.	28	Y
13	L	29	Z
14	R	30	P
15	I	31	correctie

Hierin komen enkele tekens voor die niet in het woord voorkomen.
Bijv. wordt de punt gebruikt voor getallen, maar ook voor scheiding tussen adressen als geen andere functionele letters beschikbaar zijn: X200.4.
De symbolen A, X, N, +, - en T heten openingssymbolen. Zij dienen om het begin van een instructie aan te geven en daarmee tegelijk ook het einde van de vorige instructie. Vandaar dat in het voorbeeld X200.4 de X ook voorop moest gaan, daar anders

A300	zou betekenen	A300200
200X4		X4

Een openingssymbool is dus tevens sluitsymbool van de vorige regel.
N is een openingssymbool die het invoerprogramma vertaalt door X met een trommeladres gelijk aan het adres waar deze instructie geplaatst wordt plus 1. N betekent dus: volgend, Next adres.
De meeste andere symbolen worden aanvullingssymbolen genoemd.
De + en - kondigen getallen aan. Deze worden verder op geheel natuurlijke manier gecodeerd. Bijv.:
Gehele getallen: +356 of -2675. Hierbij mogen dus niet-significante nullen links worden weggelaten.
Breuken: +.256 -0.6 -1.0. Hierbij mogen niet-significante nullen rechts worden weggelaten. Het dichtstbijzijnde binaire getal wordt ingezet.
Ook hier is het eind van een getal dus weer door het openingssymbool van het volgende woord aangegeven. Er mag dus geen blank tussen het eind van een getal en het begin van een volgend getal gelaten worden. Anders wordt: +300 blank blank + enz. gelijk aan +30000.
Het openingssymbool T wordt gebruikt voor drijvende adressen. We zullen hier niet verder op ingaan.
Een correctie heeft de eigenschap dat het over elk ander symbool heengeponst kan worden. Het corrigeert dan het hele woord dat daaraan voorafgaat. Het is dus niet nodig om een geheel woord door correctie te overponsen maar een correctie erachter is al voldoende.

3. Inzetaanwijzingen

Om het inzetten op een gegeven plaats in het geheugen te beginnen moet de instructie die het opbergen van de achtereenvolgens opgenomen instructies in het geheugen verzorgt, de z.g. opberginstructie, vervangen worden door een andere. De opberginstructie is meestal van de vorm ADn, waarin n een trommeladres is. Door een Y achter dit woord te zetten wordt de opberginstructie vervangen door dit woord. Dus:

```
     AD200Y   Vervang opberginstr. door AD200: ga opbergen op 200
200  AKE4     Normale instructies
201  NQBC11   enz.
```

De Y heeft nog een tweede functie n.l. als sluitsymbool. Na Y is geen aanvullingssymbool meer te verwachten. Alle symbolen op de band die anders alleen maar aanvullingssymbool zijn worden nu bijzondere openingssymbolen. Deze worden gebruikt voor bijzondere functies. Bijv. betekent het bijzondere openingssymbool blank: sla over tot het eerste normale openingssymbool. Met Y kan men ook de opberginstructie vervangen door elke willekeurige instructie. Het invoerprogram zou dus verlaten kunnen worden door de opberginstructie te vervangen door een sprong. Beter is evenwel van een ander sluitsymbool gebruik te maken, n.l.:
instr. Z: voer de instructies tijdens het inzetten meteen uit.
Als we hiermee een sprong uitvoeren dan wordt het invoerprogram dus verlaten. Maar we kunnen ook elke andere instructie uitvoeren, bijv. een optelling. Omdat de normale accu in gebruik is voor het inzetten zelf, wordt voor dat optellen een z.g. schijnaccu gebruikt. Het is mogelijk om met deze Z programs te maken die in het geheel niet in het geheugen komen maar meteen worden uitgevoerd. We noemen dit bandprograms. Een paar voorbeelden hiervan zijn: een bandprogram om de inhoud van het geheugen vanaf een bepaald punt uit te typen; een bandprogram om de som van alle instructies van een program te bepalen, enz.

4. Parameters

Natuurlijk moet een invoercode de mogelijkheid bieden om relatief te coderen, zodat standaard subprograms in een invariante vorm geschreven kun-

nen worden. Hiervoor dienen de parameters. De inhoud van elk register kan als referentiepunt gebruikt worden. De codering is als volgt:
X5P200 = X200 + (5). Het adres 200 is dus te rekenen vanaf het vaste bedrag dat in 5 vermeld is.
X300P200 = X200 + (300)
X5P2 = X002 + (5) Trommeladressen achter de P behoeven niet meer met drie of meer cijfers geschreven te worden. Het getal voor de P geeft wel aan of het een kort of een trommeladres is, doordat het met minder of meer dan 3 cijfers geschreven wordt.
Parameters kunnen verder nog op twee manieren gebruikt worden:
1e Accumulatieve parameters:
A5P.200P.7P.300.15 = A300.15 + (5) + (200) + (7)
Er is dus een willekeurig aantal parameters toe te voegen aan een instructie.
2e Cumulatieve parameters:
A7P8P5 = A ((7) + 008) + 005 Hierbij is dus 7P8 het adres dat voor de P5 staat. De parameters worden dus niet naast elkaar gezet maar de ene parameter zegt waar de volgende staat. Indien dus (34) = X2000 en (2006) = = X3116 dan stelt dus X34P6P5 voor het adres X3121.

5. Het adresboekprogram

Het invoerprogram stopt en begint op 000 en de nulde track 000 tot 031 is permanent geblokkeerd. Als een z.g. keuzeschakelaar U6 uit wordt gezet en de machine wordt gestart dan begint hij program te lezen.
Wordt hij gestart met U6 aan dan gaat hij naar de instructie op 34.
Als een program in de machine wordt gezet dan komt dit program met zijn eerste instructie op een zekere plaats die het sleuteladres van het program wordt genoemd. Dit sleuteladres wordt in 34 geplaatst. Gebruiker A kan het geheugen dus in gebruik hebben van 200 tot 4700 en zegt tegen gebruiker B dat hij het geheugen mag gebruiken van 4700 af. 4700 is dan het sleuteladres van het program van B.
Een program zal in de regel bestaan uit een aantal subprograms en uit een hoofdprogram. Eerst worden altijd de subprograms ingezet en als laatste

het hoofdprogram. Al deze programs komen kop aan staart in het geheugen en elk program legt zijn eigen eind als begin van het volgende program vast in het z.g. adresboek dat op het sleuteladres begint en waarvoor normaal zestien plaatsen worden opengehouden. Een program met 4 subprograms ziet er dus als volgt uit:

4700	beginadres van het hoofdprogram	(34) = 4700
4701	beginadres van eerste subprogram	Dit adresboek wordt
4702	beginadres van tweede subprogram	automatisch tijdens
4703	beginadres van derde subprogram	het inzetten gemaakt.
4704	beginadres van vierde subprogram	
4705	beginadres van het hoofdprogram	
4706	adres volgend op het eind van het hoofdprogram	
	eerste subprogram	
	tweede subprogram	
	derde subprogram	
	vierde subprogram	
	hoofdprogram.	

Aan het eind van het hoofdprogram wordt het begin adres van het hoofdprogram in het sleuteladres gezet. Met U6 aan wordt dus via 34, via het sleuteladres naar het begin van het hoofdprogram gesprongen.
Elke instructie van elk subprogram is terug te vinden, zonder dat de programmeur de werkelijke plaatsen van de subprograms hoeft te kennen. Bijv. het 7e adres van het 5e subprogram wordt geschreven als X34P5P7 want (34) = X4700 dus 34P5 = 4705 dus 34P5P7 = 4705P7 = 7e adres vanaf (4705) = 7e adres vanaf begin van 5e subprogram.
Elk subprogram heeft voorts zijn aanroepcombinatie op zijn nulde adres staan. In het hoofdprogram roepen we dus het m^e subprogram aan met X34PmPP = X4700+mPP = X beginadres m^e subprogram P = inhoud van beginadres van m^e subprogram = aanroepcombinatie van m^e subprogram.
Uit het bovenstaande voorbeeld van de opmaak van een band blijkt dat het beginadres van het hoofdprogram tweemaal in het adresboek voorkomt. Dit is dienstig voor het automatisch laten opzoeken van het eind van een program door een bandprogram. Deze begint dus het beginadres van een

program op te zoeken op het sleuteladres. Daarna zoekt hij de plaats in het adresboek waar dit beginadres nog eens voor de tweede maal voorkomt. Het daaropvolgende adres is dan het eind van het program. Een bandprogram om een ander program in binaire code uit te ponsen hoeft dus alleen het sleuteladres opgegeven te krijgen en kan dan verder geheel automatisch werken.

PHILIPS' EXPERIMENTELE TWEETALLIGE ELECTRONISCHE REKENMACHINE

H.J. Heyn

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 27 oktober 1956 te Amsterdam.

In 1951 begon een kleine groep in het Natuurkundig Laboratorium in Eindhoven met de bestudering van rekenmachines. Het doel was de fabriek eventueel van advies te kunnen dienen bij de fabricage van onderdelen en units voor rekenmachines en zelf in het bezit te geraken van een hulpmiddel voor het oplossen van de rekenproblemen die uit het laboratorium naar voren komen.

In 1953 werd met de bouw van de uiteindelijke machine begonnen en deze machine groeide gelijkelijk op met de kennis van de bouwers. Dit verklaart dat er vele schakelingen eleganter opgelost hadden kunnen worden.
Nu, een kleine vijf jaar na het eerste begin, is de machine klaar.

PETER is een machine, die geheel in het tweetallige systeem werkt. In het rekenorgaan worden de woorden parallel, in de besturing en het geheugen in serie behandeld. De woordlengte bedraagt 20 binaire digits.

Het geheugen bestaat uit een trommel die om een horizontale as 100 omw./sec. maakt. Het cylinderoppervlak van de trommel is bedekt met een ijzeroxyde. Diametraal tegenover elkaar zijn twee kophouders opgesteld, die ieder een pakket van 8 kopjes bevatten. De kopjes zijn zo geplaatst, dat een kopje van het éne pakket tussen twee sporen die door kopjes uit het andere pakket beschreven worden. De spoorbreedte bedraagt 1,0 mm, de afstand tussen 2 sporen 0,2 mm, de afstand van kop tot trommel $\approx 20\mu$ en de spleetbreedte in het kopje 10μ. Het kopje bevat twee stel wikkelingen, ieder van 40 windingen. De schrijfstroom bedraagt ongeveer 100 mA.

Het clocktrack, de pulsgenerator van de machine, wordt geleverd door een gegraveerde aluminium schijf. De krasjes, het zijn er 64 × 24 = 1536, zijn gevuld met ijzeroxyde, dat permanent in één richting gemagnetiseerd is. Daar niet de gehele omtrek van de schijf gegraveerd is, maar deze een z.g. gap vertoont, is de pulsherhalingsfrequentie van het clocktrack niet $100 \times 1536 \cdot 10^{-3}$ kHz maar $\approx$ 180 kHz.

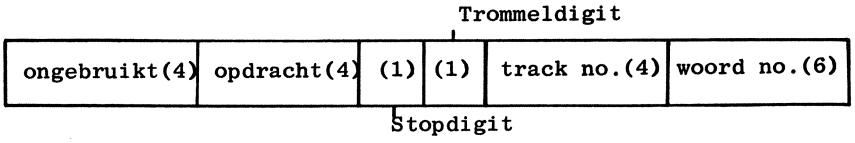
De diameter van de trommel is ongeveer 15 cm, zodat de schrijfdichtheid op de trommel ca. 4 digits/mm bedraagt.
Voor ieder woord zijn 24 digitplaatsen beschikbaar, waardoor dus tussen de woorden 4 onbeschreven digitplaatsen voorkomen. Deze tijd tussen de woorden wordt gebruikt o.a. voor het schakelen in selectie en leesversterker. De dode tijd van de leesversterker bij het overschakelen van schrijven op lezen of bij het overschakelen van het ene kopje op het andere bedraagt 5 - 10μ sec.

Op een spoor kunnen 64 woorden worden geschreven, zodat de geheugencapaciteit op het ogenblik slechts 64 × 16 = 1024 woorden bedraagt.

Naast de trommel kennen we nog een plugbord geheugen, dat wij het kunsthoofd hebben gedoopt en waarin door middel van stekerpennen 32 woorden kunnen worden ingesteld. Dit kunsthoofd is wel zeer nuttig gebleken bij het testen van de machine en het inlezen van het invoerprogramma en variabele gegevens.

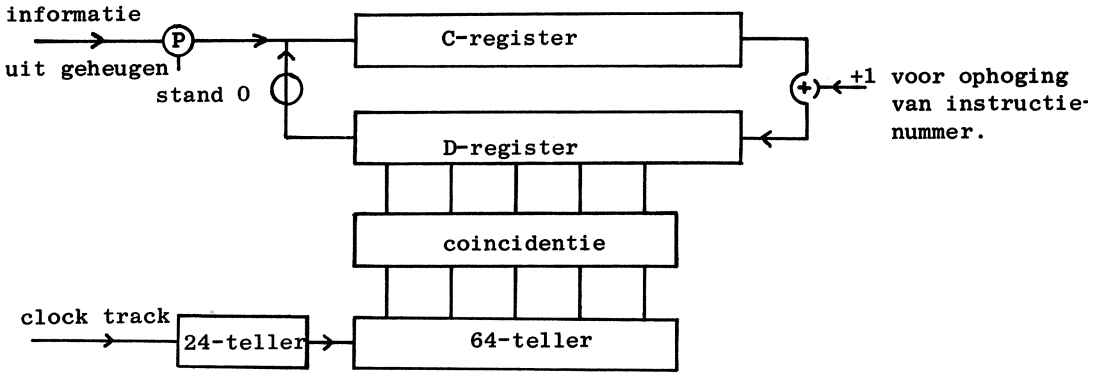
Een speciale digit in de instructie is gereserveerd voor de keuze van kunsthoofd (0) of trommel (1).

Verder zijn er in de instructie 4 digits die het spoor op de trommel bepalen en 6 digits die de plaats in het spoor aangeven. De machine kent 16 opdrachten, die op de laatste bladzijde zijn vermeld. Van de resterende digits (20 - 15 = 5) zijn er nog vier niet gebruikt, terwijl de vijfde, indien aanwezig, de machine kan doen stoppen indien dit gewenst wordt (stopdigit).
In de machine ziet een opdracht er als volgt uit:



Besturing

De belangrijkste delen voor de besturing worden gevormd door het C- en D-register en de micro-besturing. In het D-register wordt de opdracht gedecodeerd en coincidentie met de adressenteller (64-teller) geconstateerd. Het C-register fungeert in die tijd als een Control-counter om het nummer van de uit te voeren instructie vast te houden.



De microbesturing bestaat uit een teller die 6 standen heeft. In deze standen worden de volgende handelingen verricht:

STAND 0: Na coincidentie van het D-register waarin zich in deze fase het instructienummer bevindt met de 64-teller, wordt de van de trommel gelezen informatie (= uit te voeren instructie) in het C-register geschreven.

1: De registers C en D wisselen van inhoud.

2: Het opdrachtgedeelte van het D-register wordt onderzocht.

3: Na coincidentie van D-register en 64-teller wordt een getal van de trommel naar één van de registers van het rekenorgaan geschreven.

4: De uitvoering van de opdracht wordt gestart.

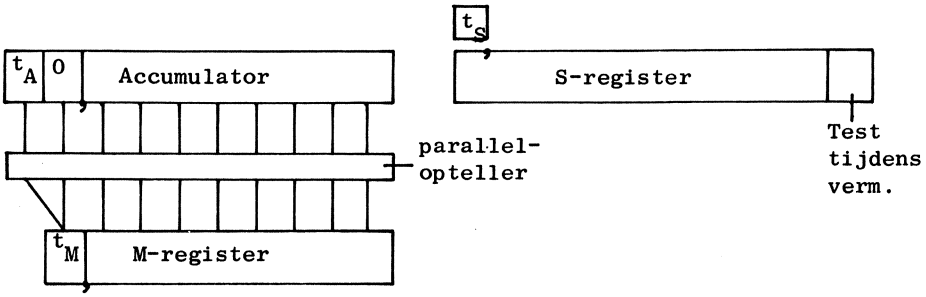
5: De inhoud van register C wordt in het D-register geschreven en daarbij met de eenheid opgehoogd (instructie-nummer wordt opgehoogd).

6: De besturing wacht tot van het rekenorgaan of van de in- en uitvoer, welke zelfstandig zijn, een klaar-signaal ontvangen wordt en springt dan terug in stand 0.

Vindt de besturing in stand 2 een conditionele opdracht, waaraan voldaan is, dan springt de teller in stand 0. Is aan de voorwaarde van de sprongopdracht niet voldaan dan wordt de teller in stand 5 gezet. Is de opdracht een leesopdracht dan wordt afhankelijk van het "vijfde gat" bij het ingelezen symbool van stand 4 naar stand 5 of 6 gesprongen.

Het rekenorgaan bestaat o.a. uit 3 registers, de A(ccumulator) 21 digits lang, het M(emory)-register en het S(hift)-register, beiden 20 digits lang. Tussen A en M is een parallel optelorgaan aanwezig. M en S kunnen worden geinverteerd. A en S kunnen tezamen een geheel vormen en de inhoud van A en S tezamen kan zowel naar links als naar rechts worden geschoven over maximaal 31 plaatsen.
We werken met negatieve getallen die door invertering, digit voor digit, uit de positieve getallen kunnen worden gevonden, dus in het vals-complement systeem. De machine heeft dus een end-around carry en kent twee vormen van het getal nul. In de accumulator is tussen tekendigit en meest belangrijke digit nog een z.g. overflowdigit gebouwd o.a. om een tijdelijke overflow tijdens de vermenigvuldiging te kunnen verwerken.

Bij de schuifopdrachten moeten we het volgende opmerken. Bij het schuiven naar rechts wordt het teken van de accumulator overgenomen in de tekendigit van S, t_S en de acc. wordt aangevuld met de tekendigit t_A. Steeds wordt buiten het teken van S omgeschoven. Bij het naar links schuiven wordt t_S overgenomen in t_A en in de overflowdigit en het S-register wordt aan de achterzijde aangevuld met t_S.

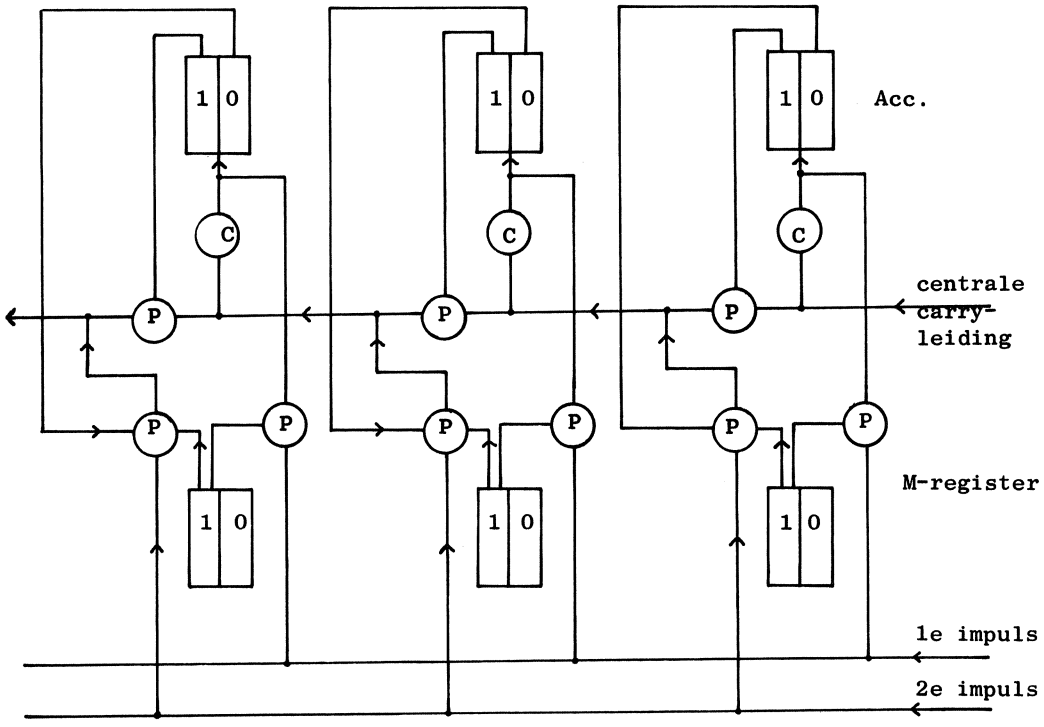


Bij het opbergen van de Acc. wordt de overflow bit niet mee opgeborgen. Opdracht 5 laat de gehele accumulator schoon achter (in + nul). De vermenigvuldiging is vanwege de valse-complement methode niet additief. Als het teken van het product negatief zal worden, wordt de accumulatorinhoud gelijk aan - nul gemaakt. Alleen, indien de getallen geheel en positief zijn is een additieve vermenigvuldiging mogelijk. De deling geeft voor alle tekencombinaties een correct quotiënt (in S) en rest (in Acc.). De deler wordt in het M-register gebracht. De deling wordt uitgevoerd volgens de z.g. non-restoring methode en rekenend in de warecomplement methode ontstaat het quotiënt in de valse complement vorm. De opeenvolgende microhandelingen, waaruit een opdracht is opgebouwd worden bestuurd door de z.g. microbesturing van het rekenorgaan.
Bij een vermenigvuldiging zijn de handelingen bijvoorbeeld:

1. Indien het teken van S (t_S) negatief is worden M en S geinverteerd. Zal het teken van het product negatief worden, plaats dan - nul in de Acc.
2. Test de inhoud van de minst belangrijke digit van S. Is dit een één, ga dan naar punt 3; is het een nul spring dan naar punt 5.
   Test het einde van de vermenigvuldiging (inhoud van de slagen teller).
3. Geef een optelimpuls.
4. Geef een carry-impuls.
5. Geef een schuifimpuls en spring terug naar punt 2.

In de punten 3 en 4 wordt de optelling verricht.

De optelling wordt verricht in twee stappen. Eerst tellen we twee overeenkomstige digits van het A- en M-register bij elkaar op zonder te letten op de overdrachten. Daarna maken we de overdrachten in orde. Een overdracht begint daarbij op de plaats waar na de eerste stap in M een één en in de Acc. een nul staat. Het recept luidt nu dat deze overdracht langs de volgende enen in de Acc. moet lopen tot een nul wordt bereikt. De gepasseerde enen en de gevonden nul moeten worden geinverteerd. Een blokschema ziet er als volgt uit.



In- en uitvoer

De 4 minst belangrijke digits van de Acc. zijn zowel met de uitgang van de bandlezer als met de uitvoerapparatuur verbonden. De gegevens worden via een Ferranti bandlezer van een 5-kanalen telexband gelezen. De informatie van het vijfde kanaal wordt naar de besturing gevoerd.

Bij uitvoer kunnen symbolen in genoemde 4 plaatsen van A zowel getypt als geponst worden. Ten behoeve van de ponsband is ook nog een vijfde digit van de Acc. uitgevoerd.

Enige tijden

Vanwege de vorm van de microbesturing van het rekenorgaan zijn de tijden van de rekenoperaties iets groter dan men zo zou verwachten. De opteltijd, indien de getallen in de registers al klaar staan, bedraagt 15μ sec, de aftrektijd 20μ sec. De vermenigvuldiging duurt afhankelijk van het aantal enen in de vermenigvuldiger 0,5 - 0,9 msec, terwijl op het resultaat van de deling steeds 0,8 msec gewacht moet worden.
Het schuiven kost per plaats ≈5μ sec.
Een nadeel van de machine in de tegenwoordige vorm ligt in de lange opzoektijden waardoor we meestal slechts ca. 100 opdrachten per sec kunnen uitvoeren.
De bandlezer kan tot 100 karakters per sec lezen, de schrijfmachine tot 10 cijfers per sec typen.
De ponsmachine ponst ≈6 pentades per sec.

Enkele technische gegevens

De machine is opgebouwd uit units. De schakeling is daarbij op een plaatje gemonteerd, dat, staande op een normale buisbodem, is aangebracht. Aan de top er van is een normale buishouder aangebracht, die de bijbehorende buis draagt. Er zijn 6 hoofdtypen n.l. de flip-flop, de kathode-volger, de schuifunit, 2 units voor het maken van poorten en een diode-unit.
Het vermogen en het aantal buizen en dioden volgt uit bijgaande tabel.

	ECC81	E90CC	EL81/83	diodes(OA85)
Rekenorgaan	330	270	18	670
Besturing	220	155	5	300
Geheugen	50	32	53	1200
In- en uitvoer	48	32	20	84
Totaal	650	490	105	2250

	Gloeistr. vermogen	Gelijkstr. vermogen
Rekenorgaan	1,8 KW	280 Watt
Besturing	1,2 KW	220 Watt
Geheugen	0,6 KW	100 Watt
In- en uitvoer	0,3 KW	55 Watt
Totaal	3,9 KW	650 Watt

Opdrachten van PETER

0/n Spring naar n indien $t_A = 0$

8/n Spring naar n indien $t_A = 1$

1/n (n) → S

9/n (S) → n; (S) → S

2/n (SA) schuift n plaatsen naar rechts n ≤ 31

10/n (SA) schuift n plaatsen naar links n ≤ 31

3/n (A) + (n) → A

10/n (A) - (n) → A

4/n Collationeer (A) met (n); $(A_k)(M_k) \rightarrow A_k$

12/n Nog ongebruikt

5/n (A) → n; + 0 → A

13/n (A) → n; (A) → A

6/n Lees een tetrade van de band. Indien geen "vijfde gat" aanwezig is wordt de volgende opdracht van n gehaald.

14/.. Type

7/n (A) × (n) → AS

15/n (AS) : (n) → S rest in A.

DE BEREKENING VAN EIGENWAARDEN EN -VECTOREN VAN MATRICES OP DE FERTA

H.J. Dannenburg

Voordracht in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 24 november 1956 te Amsterdam.

De eigenwaarde met de grootste modulus, λ_1, van een matrix $\overline{M}_1$, kan tezamen met de bijbehorende eigenkolom, $\overline{K}_1$, worden berekend met een iteratieve methode. Een iteratie-stap bestaat uit de matrixbewerkingen:

$$\overline{Q}_j = \overline{M}_1 \cdot \overline{P}_{j-1}$$

$$\overline{P}_j = \frac{1}{N_j} \cdot \overline{Q}_j .$$

$\overline{P}_j$ en $\overline{Q}_j$ zijn kolommen. De beginkolom $\overline{P}_0$ kan willekeurig gekozen worden. De normeringsfactor, N_j, wordt zodanig gekozen, dat het grootste element, $p_{\alpha,j}$, van $\overline{P}_j$ de waarde $2^{-\sigma}$ aanneemt. Dus

$$\overline{N}_j = 2^{\sigma} \cdot q_{\alpha,j}$$

als $q_{\alpha,j}$ het grootste element van $\overline{Q}_j$ is. Door een geschikte keuze van de "schaalfactor" σ kan ervoor gezorgd worden, dat de elementen van $\overline{P}_j$ en van $\overline{Q}_j$ allen kleiner dan 1 blijven. (Dus $\sigma \geq 1$.)

Als $|\lambda_1| > |\lambda_2|$ is, dan volgen eigenwaarde en - kolom uit:

$$\lambda_1 = \lim_{j\to\infty} 2^{\sigma} q_{\alpha,j}$$

$$\overline{K}_1 = \lim_{j\to\infty} \overline{P}_j .$$

Als dus in de vereiste nauwkeurigheid $q_{\alpha,j} = q_{\alpha,j-1}$ en $\overline{P}_j = \overline{P}_{j-1}$ is, dan zijn λ_1 en $\overline{K}_1$ door $2^{\sigma} q_{\alpha,j}$ en $\overline{P}_j$ bepaald.

Wordt het iteratieproces toegepast op de gespiegelde, $\bar{M}_1'$, van de matrix $\bar{M}_1$, dan convergeert $\bar{P}_j$ naar de eigenrij $\bar{R}_1$. Hierbij wordt uiteraard dezelfde eigenwaarde λ_1 gevonden.

De tweede eigenwaarde, - kolom en - rij worden berekend uit de "gereduceerde" matrix:

$$\bar{M}_2 = \bar{M}_1 - \frac{\lambda_1 \bar{K}_1 \bar{R}_1}{\bar{R}_1 \bar{K}_1} .$$

$\bar{M}_2$ heeft dezelfde eigenwaarden en - vectoren als $\bar{M}_1$, met uitzondering van de eerste. In de plaats van λ_1 heeft $\bar{M}_2$ een eigenwaarde nul. De tweede eigenwaarde van $\bar{M}_1$, λ_2, is dus de grootste eigenwaarde (in absolute waarde) van $\bar{M}_2$. Nadat deze 2^e eigenwaarde met - kolom en - rij door iteratie is bepaald, kan ook deze uit de matrix "geëlimineerd" worden. Zo kunnen dus successievelijk alle eigenwaarden en - vectoren worden berekend.

Zoals vermeld, is voorwaarde voor convergentie, dat $|\lambda_1| > |\lambda_2|$ is. Is $|\lambda_1| = |\lambda_2|$, dan convergeert $\bar{P}_j$ niet. Is $|\lambda_1| \approx |\lambda_2|$, dan is de convergentie slecht. In dit geval kan echter na een voldoend aantal iteratiestappen gesteld worden:

$$\bar{P}_j = \alpha_{1,j} \bar{K}_1 + \alpha_{2,j} \bar{K}_2 .$$

Ook de ongenormeerde iteratiekolom $\bar{Q}_j$ is dan dus een lineaire combinatie van 2 eigenkolommen. Het is nu mogelijk, de 2 eigenwaarden λ_1 en λ_2 beide te berekenen uit een vierkantsvergelijking:

$$\bar{\lambda}^2 - a\bar{\lambda} + b = 0 \quad (\bar{\lambda} = 2^{-\sigma}\lambda).$$

Voor het bepalen van a en b zijn de waarden van 2 elementen, het k-de en het l-de, van 3 opeenvolgende iteratiekolommen nodig. Voor die kolommen zijn gekozen de ongenormeerde $\bar{Q}_{j-2}$, $\bar{Q}_{j-1}$ en $\bar{Q}_j$. De elementen, die in elke iteratiestap van de iteratiekolom naar een vaste plaats in het geheugen worden gecopieerd, zijn dus

$$q_{k,j-2},\ q_{l,j-2},\ q_{k,j-1},\ q_{l,j-1},\ q_{k,j} \text{ en } q_{l,j}.$$

De coëfficiënten van de vierkantsvergelijking zijn:

$$a = q_{\alpha,j-1} \cdot \frac{q_{k,j-2} \cdot q_{l,j} - q_{l,j-2} \cdot q_{k,j}}{q_{k,j-2} \cdot q_{l,j-1} - q_{l,j-2} \cdot q_{k,j-1}}$$

$$b = q_{\alpha,j-2} \cdot q_{\alpha,j-1} \frac{q_{k,j-2} \cdot q_{l.j} - q_{l,j-1} \cdot q_{k,j}}{q_{k,j-2} \cdot q_{l,j-1} - q_{l,j-2} \cdot q_{k,j-1}} .$$

De bijbehorende eigenkolommen volgen uit

$$\bar{K}_1 = \bar{Q}_j - \lambda_2 \cdot \bar{P}_{j-1}$$

$$\bar{K}_2 = \bar{Q}_j - \lambda_1 \cdot \bar{P}_{j-1}.$$

De hiervoor benodigde vectoren $\bar{Q}_j$ en $\bar{P}_{j-1}$ staan tegelijk in het geheugen na de bewerking $\bar{Q}_j = \bar{M}_1 \cdot \bar{P}_{j-1}$. Op dit moment kan dus de vierkantsvergelijking opgelost worden.

Teller en noemer van de in a en b voorkomende breuken zijn in het algemeen zo klein, dat overgegaan moet worden op een "drijvende komma"-rekentechniek. Voordat a en b berekend worden, worden de daartoe benodigde getallen omgezet in getallen met drijvende komma. Na berekening van de wortels van de vierkantsvergelijking worden deze weer teruggebracht tot getallen met vaste komma.

De coëfficiënten a en b kunnen ook uitgedrukt worden in de elementen $p_{k,i}$ en $p_{l,i}$ (i = j-2, j-1 en j) van de genormeerde iteratiekolommen. Deze zijn in het algemeen groter dan de gebruikte elementen van de ongenormeerde kolommen. Toch zal ook in dit geval met drijvende komma gerekend moeten worden om verlies van nauwkeurigheid te voorkomen. Bij gebruik van de elementen p, moet echter $\bar{Q}_j$ genormeerd worden, voordat de vierkantsvergelijking kan worden opgelost, omdat in a en b $p_{k,j}$ en $p_{l,j}$ voorkomen. Dan wordt $\bar{P}_{j-1}$ overschreven door $\bar{P}_j$, doch $\bar{P}_{j-1}$ moet bewaard worden voor de berekening van de kolommen $\bar{K}_1$ en $\bar{K}_2$. Door gebruik van de elementen q wordt deze moeilijkheid vermeden.

Voor het oplossen van λ_1 en λ_2 uit de vierkantsvergelijking wordt gesteld:

$$\lambda_1 = 2^{\sigma}\bar{\lambda}_1 = 2^{\sigma}(\mu + \omega)$$

$$\lambda_2 = 2^{\sigma}\bar{\lambda}_2 = 2^{\sigma}(\mu - \omega).$$

Dus:

$$\mu = \frac{a}{2}$$

$$\omega^2 = \frac{a^2}{4} - b.$$

Een matrix met reële elementen kan eigenwaarden hebben, die paarsgewijze toegevoegd complex zijn. De toegevoegd complexe eigenwaarden hebben gelijke moduli, zodat de berekening ervan slechts mogelijk is door middel van de vierkantsvergelijking. In dit geval is dus $\omega^2 < 0$.
Het programma voor iteratie van een reële matrix is dan ook zodanig gemaakt, dat gediscrimineerd wordt op het teken van ω^2.

Is $\omega^2 < 0$, dan wordt een van beide toegevoegd complexe eigenwaarden met de bijbehorende complexe eigenvector (wat hierboven voor de kolom is beschreven geldt ook voor de berekening van de eigenrij) berekend en getypt. Er wordt verder geïtereerd met de reële iteratievector. Als in 2 opeenvolgende iteratiestappen de vierkantsvergelijking wordt opgelost en de uitgetypte complexe eigenwaarden en -vectoren stemmen voldoende overeen, dan zijn daarmee dus 2 eigenwaarden en -vectoren bekend.

De 2 eigenwaarden kunnen na elkaar uit $\bar{M}_1$ "geëlimineerd" worden. Dan wordt $\bar{M}_2$ complex, terwijl $\bar{M}_3$ weer reëel wordt. Het is eenvoudiger om beide toegevoegde complexe eigenwaarden gelijktijdig te "elimineren".

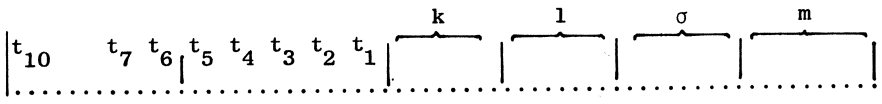
Als $\omega^2 > 0$, dan worden de eigenwaarden met de grootste modulus en de bijbehorende vector uitgetypt. De vector, behorende bij de "kleinste" eigenwaarde wordt in het geheugen bewaard, om bij berekening van de volgende eigenwaarde als beginvector dienst te doen. Als de vierkantsvergelijking wordt opgelost, voordat de bijdragen van eigenvectoren, behorend bij "kleinere" eigenwaarden dan λ_1 en λ_2, verwaarloosd kunnen worden, levert de vierkantsvergelijking niet de gezochte eigenwaarden en -vectoren op. Toch zullen de grootste wortel en de bijbehorende vector

veelal een betere benadering zijn voor eigenwaarde en -vector, dan de iteratievectoren, waaruit zij zijn berekend. De iteratievector $\overline{P}_j$ wordt daarom vervangen door de uit de vierkantsvergelijking berekende vector $\overline{K}_1$, waarna wordt verdergegaan met itereren. Het iteratieproces kan er aanzienlijk door versneld worden.

De beschreven methode, een iteratief proces, dat hoofdzakelijk bestaat uit elementaire matrixbewerkingen, is bijzonder geschikt voor uitvoering op een electronische rekenautomaat. Moeilijkheden met de "scaling" zijn er bijna niet. De grootte van de elementen $\overline{P}_j$ en $\overline{Q}_j$ kan door de schaalfactor σ geregeld worden. Slechts bij het oplossen van de vierkantsvergelijking en bij de normering van een complexe vector moet gebruik gemaakt worden van een "drijvende-komma"-rekentechniek om verlies van nauwkeurigheid te voorkomen.

Het is niet eenvoudig om deze berekening vol-automatisch uit te laten voeren door de machine. De machine zou zelf moeten beslissen of een eigenvector nauwkeurig genoeg is bepaald door de iteratievector. Een criterium is wel aan te geven, doch vooraf is moeilijk te overzien of er, met de afrondingsfouten, die de machine maakt, aan kan worden voldaan. Een criterium, aan de hand waarvan besloten wordt tot het al dan niet oplossen van de vierkantsvergelijking zou al zeer gecompliceerd worden. Dergelijke beslissingen worden dan ook overgelaten aan de operateur van de machine. Deze kan het iteratieproces volgen door van tijd tot tijd de iteratievector uit te laten typen. Hij kan de machine besturen met behulp van de getalschakelaar (GS). Deze GS bestaat uit een serie van 30 schakelaars, waarmee een binaal getal van 30 cijfers kan worden voorgesteld. Door de opdracht: "lees getalschakelaar", wordt het getal, dat in de GS is aangegeven, in één van de registers gelezen. De operateur kan zijn beslissingen in een bepaalde vorm in de GS aangeven en deze zo aan de machine meedelen. Tijdens de iteratie wordt de getalschakelaar slechts gelezen, nadat een normeringsfactor en iteratievector zijn uitgetypt. De inhoud van de GS wordt dan in het geheugen geborgen en later, waar nodig, in het programma verwerkt.

De indeling van de GS is hieronder schematisch weergegeven:



De functies van de verschillende delen zijn:

m is het aantal iteratiestappen, dat moet worden uitgevoerd voordat weer getypt wordt, en dus ook voordat de GS weer gelezen wordt. Het is overbodig om de iteratiekolom en de normeringsfactor na elke iteratiestap uit te typen. Het typen kan hiermee dus m-1 maal onderdrukt worden. Door de beschikbare ruimte in de GS moet $m \leq 31$ zijn. Het minimaal aantal uit te voeren iteratiestappen is m = 1. m = 0 wordt door de machine geïnterpreteerd als het sein, dat eigenwaarde en vector voldoende nauwkeurig bekend zijn. Was de machine bezig met het berekenen van de eigenkolom dan wordt na m = 0 de kolom $\bar{P}_j = \bar{K}_1$ geponst en naar een andere plaats in het geheugen gecopieerd. $\bar{K}_1$ moet nl. bewaard worden voor de reductie. Vervolgens wordt overgegaan op de berekening van de eigenrij, door in de routine matrix $(\bar{M}_1)$ maal vector $(\bar{P}_j)$ de spatieringen van de adressen van de elementen in rij en kolom van de matrix te verwisselen. Als de eigenrij is bepaald wordt dit eveneens door m = 0 aangegeven. De rij wordt uitgeponst, gevolgd door de reductie van de matrix. Daarna wordt begonnen met de berekening van de volgende kolom.

σ is de schaalfactor bij de normering ($1 \leq \sigma \leq 31$).

k en l zijn de nummers van de elementen, die gebruikt moeten worden bij het opstellen van de vierkantsvergelijking. Voor het bereiken van een zo groot mogelijke nauwkeurigheid moeten hiervoor 2 van de grootste elementen gekozen worden.

Met t_1 wordt aangegeven of na de reductie de nieuwe matrix al dan niet moet worden uitgetypt.
($t_1 = 0$ typt niet, $t_1 = 1$ typt wel)

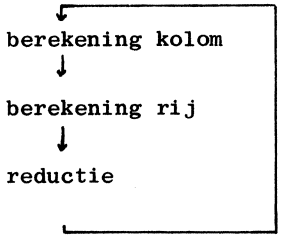
Met t_2 kan bij de normering het bepalen van het grootste element onderdrukt worden. Als de eigenvector zover benaderd is, dat het nummer

van het grootste element niet meer verandert, kan de moot, waarin dat nummer berekend wordt, worden overgeslagen door $t_2 = 1$ te maken.

Door $t_3 = 1$ wordt de opdracht tot het oplossen van de vierkantsvergelijking gegeven.

Met $t_4 = 1$ wordt het typen van de iteratievector onderdrukt. In het algemeen convergeert $2^{\sigma}q_{\alpha,j}$ sneller naar de eigenwaarde λ_1, dan $\overline{P}_j$ naar de vector $\overline{K}_1$. Voordat λ_1 voldoende nauwkeurig bekend is, is het dus overbodig om $\overline{P}_j$ te typen (tenzij voor het oplossen van de vierkantsvergelijking de nummers k en l van het grootste element moeten worden bepaald).

Door t_5 wordt aangegeven of de eigenkolom ($t_5 = 0$), dan wel de eigenrij ($t_5 = 1$) berekend wordt. De operateur is echter niet vrij in de keuze van t_5. Hij is gebonden aan de voorgeschreven volgorde:



Met t_6 en t_7 wordt een keuze gemaakt voor de beginvector van de iteratie en wel als volgt:

$t_7t_6 = 0\ 0$ Begonnen wordt met de vector, die al staat op de plaats van de iteratievector $\overline{P}_j$.

$0\ 1$ Begonnen wordt met een vector, waarvan alle elementen $2^{-\sigma}$ (of $2^{-\sigma} + 2^{-\sigma} \cdot j$) zijn.

$1\ 0$ De beginvector wordt van een ponsband ingelezen.

$1\ 1$ Als beginvector wordt gekozen een benadering, die berekend is bij het oplossen van een vierkantsvergelijking bij het bepalen van de vorige eigenwaarde.

Gezien het grote aantal gegevens, dat door de operateur in de GS verwerkt moet worden, kan het voorkomen, dat er vóór het lezen van GS

niet voldoende tijd meer is voor het aanbrengen van de gewenste veranderingen. Door nu $t_{10} = 1$ te maken, wordt de machine gestopt "voordat" de GS gelezen wordt. De wijzigingen kunnen dan alsnog aangebracht worden en de kans op vergissingen wordt kleiner.

Overigens is het programma zo gemaakt, dat bepaalde fouten, bij het instellen van de GS door de machine worden gesignaleerd of gecorrigeerd. De vierkantsvergelijking kan bij voorbeeld pas worden opgelost, als k, l en σ gedurende drie opeenvolgende iteraties dezelfde waarden hebben gehad. Wordt in de GS k, l of σ veranderd, $m = 1$ en $t_3 = 1$ gemaakt (1 iteratie, daarna vierkantsvergelijking), dan voert de machine toch 3 iteratiestappen uit en lost dan pas de vierkantsvergelijking op.

CRYOTRONS

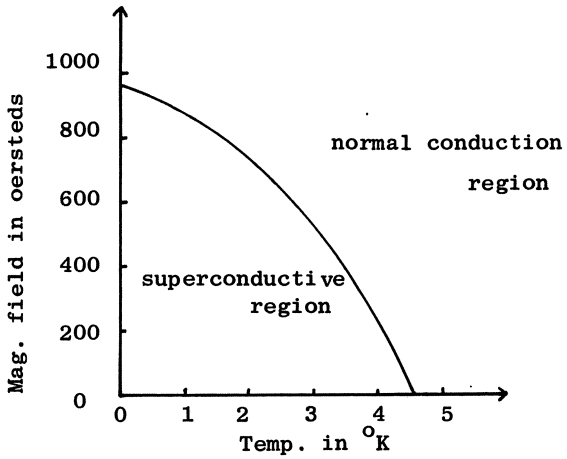
W.V. Wright

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 26 januari 1957 te Amsterdam.

Before describing the cryotron,a few of the experiments in superconductivity will be discussed.

Kamerlingh Onnes discovered superconductivity in 1911 while trying to measure the low-temperature resistivity of mercury (1). He found that the resistivity of mercury suddenly disappeared at about 4.2° K, the transition temperature. Subsequently a number of other metals and alloys have been found to exhibit superconductivity with transition temperatures ranging up to 17° K. Two of these, tantalum and niobium (columbium), which seem best suited for the construction of cryotrons,have transition temperatures of 4.4° K and 8° K respectively. That the resistance of a superconductor is really zero or at least immeasurably small was demonstrated by Onnes and Tuyn in their "persistent" current experiment (2). A current induced in a superconducting lead ring was found to remain constant over a period of several hours.

In attempting to make an electromagnet from his resistanceless conductors, Onnes discovered that superconductivity can be destroyed by a magnetic field (3). The magnetic field intensity necessary for this effect is called the critical field and is a function of temperature. A plot of this relationship for a typical metal, tantalum, is shown in Fig. 1.
From this plot it can be seen that the critical field for tantalum at 4.2° K, the temperature of an open tank of liquid helium, is between 50 and 100 oersteds.



The simplest and most obvious interpretation of the early experiments in superconductivity was the disappearance of electrical resistance. Later experiments by Meissner showed that this interpretation is in general unsatisfactory (4). The best theory at present seems to be the one advanced by F. London and H. London in 1935 (5). The operation of the cryotron is however adequately explained by the assumption of zero resistance, and therefore the more recent theories will not be taken up here.

The Cryotron

To date the only application of superconductivity to the computer field is in an experimental device, the cryotron, developed by D.A. Buck at M.I.T. (6). The cryotron consists of a tantalum wire around which is wound a single layer solenoid of niobium. The central wire is known as the gate circuit and the niobium coil as the control circuit. In operation the device is submerged in a tank of liquid helium where both wires become superconductive. The resistance of the tantalum can then be restored by passing a sufficiently large current through the niobium coil.

Tantalum was selected for the central wire because it has a transition temperature just above the boiling point of helium at atmospheric pressure. Thus a small magnetic field is sufficient to restore the resistance of the central wire, and consequently the device can be controlled by a small current. The control winding is made of niobium

so that it will remain superconductive at all times. Thus the only impedance of the control circuit is its inductance.

Static Characteristics

A current passing through the gate circuit of a cryotron will cause a magnetic field which is directed around the wire. The strength of this field at the surface of the wire is

$$h_g = \frac{4\ i_g}{10\ d} \text{ oersteds} \tag{1}$$

where i_g is the gate current in amperes and d the diameter of the wire in centimeters. From this formula it is clear that the greatest current that can flow in the gate circuit without destroying its superconductivity is

$$I_g = \frac{10}{4} H_c d \text{ amperes} \tag{2}$$

where H_c is the critical field for tantalum in oersteds.

Current flowing in the control winding, on the other hand, will create a magnetic field directed along the central wire and whose strength is

$$h_c = \frac{4\pi}{10} \frac{N}{\ell} i_c \text{ oersteds.} \tag{3}$$

In this expression N is the number of turns in the control winding, ℓ its length in centimeters, and i_c the control current in amperes. Thus the control current necessary to destroy the superconductivity of the gate circuit is

$$I_c = \frac{10}{4\pi} \frac{\ell}{N} H_c. \tag{4}$$

The current gain of a cryotron is the ratio of the maximum gate current that can be controlled to the minimum control current that will maintain the central wire in the normal conducting state.
This is given by

$$K = \frac{I_g}{I_c} = \pi \, d \, \frac{N}{\ell} \quad . \quad . \tag{5}$$

Since the two fields above are at right angles to each other they will add in quadrature. Therefore the total magnetic field is given by

$$h_t = (h_g^2 + h_c^2)^{\frac{1}{2}} \quad \text{or} \tag{6}$$

$$h_t = \left[\frac{4}{10}\left(\frac{i_g}{d}\right)^2 + \left(\frac{\pi N}{\ell} i_c\right)^2\right]^{\frac{1}{2}}. \tag{7}$$

It is clear from this last expression that the curve for a given total field in the i_g -- i_c plane is an ellipse. In particular the region in which the gate circuit of a cryotron is superconductive is separated from the region of normal conductivity by the ellipse corresponding to the critical field of tantalum as shown in Fig. 2.

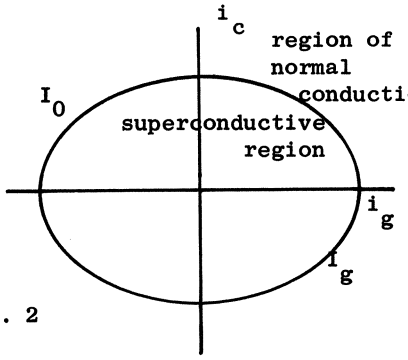


fig. 2

Furthermore the current gain of a cryotron is given by the ratio of the horizontal axis to the vertical axis of an ellipse such as the one shown in Fig. 2.

Speed

As will be seen later, cryotrons are connected into circuits with their central wires in series with the control windings of one or more other cryotrons. The operating speed of these circuits is determined by the time required for a normally conducting cryotron to dissipate the energy stored in the control windings with which it is connected. Then some idea of the operation time of these devices can be gained from the quotient $\frac{L}{R}$ where L is the inductance of a control winding and R is the resistance of a normally conducting gate circuit. The values of L and R are given in terms of the design parameters of a cryotron by the

following formulas:

$$L = 4\pi^2 d^2 \left(\frac{N}{\ell}\right)^2 \ell \, 10^{-9} \text{ henries, and} \tag{8}$$

$$R = \frac{4p\ell}{\pi d^2} \text{ ohms.} \tag{9}$$

In the latter formula p is the resistivity of tantalum in ohm-centimeters (about 15.5×10^{-6}). Thus the time constant is

$$\frac{L}{R} = \frac{\pi^3}{p} \left(\frac{N}{\ell}\right)^2 d^4 \, 10^{-9} \text{ sec.} \tag{10}$$

If one cryotron is to be used to control another it must have a current gain of unity or greater. Then equation 5 may be used to simplify the last formula to

$$\frac{L}{R} = \frac{\pi}{p} d^2 \, 10^{-9} \text{ sec.} \tag{11}$$

As an example, the time constant for a cryotron with a control wire 2 millimeters in diameter is

$$\frac{L}{R} = \frac{3.14 \times 0.2 \times 0.2}{15.5 \times 10^{-6}} 10^{-9} = 8.1 \times 10^{-6} \text{ sec.}$$

The speed of cryotron circuits can be improved by increasing the resistance of the gate circuit. This can be done by using a hollow central wire. Such a cryotron will still have no resistance when in the superconductive state, but will have a much higher resistance in the normal conducting state. One way to fabricate the hollow central wire is to coat an insulating core with tantalum, using a metallic evaporation process. The thickness of the layer of tantalum produced in this way ranges between 10^{-4} and 10^{-6} centimeters. The normal conducting resistance of a gate circuit made in this way is

$$R = \frac{p\ell}{\pi d\lambda} \text{ ohms} \tag{12}$$

where λ is the thickness of the coating in centimeters. The time constant then becomes

$$\frac{L}{R} = \frac{4\pi^3}{p} \lambda\, d^3 \left(\frac{N}{\ell}\right) 10^{-9} \text{ sec.} \tag{13}$$

and for a current gain of unity

$$\frac{L}{R} = \frac{4\pi}{p} \lambda\, d\, 10^{-9}. \tag{14}$$

As an illustration of the speeds possible with cryotrons of this design, the time constant for d = 0.2 centimeters and $\lambda = 10^{-5}$ centimeters is

$$\frac{L}{R} = \frac{4 \times 3.14 \times 0.2 \times 10^{-5}}{15.5 \times 10^{-6}} \times 10^{-9} = 1.6 \times 10^{-9} \text{ sec.}$$

Circuits

In many ways, cryotrons are similar to electromechanical relays. They share the property that both the control circuits and the gate circuit (the contacts of a relay) are bilateral in action. That is they operate in a manner independent of the direction of current flow. On the other hand, the resistance of a cryotron gate circuit is switched between a small value and zero, while the relay contact operates between a small and an almost infinite resistance. Furthermore, the control circuit of a cryotron has only inductance while a relay coil has both inductance and resistance. These differences limit cryotrons to constant-current circuit applications. Relays can, and in fact, are more commonly used in constant-voltage circuits.

A number of systems are possible for arranging cryotrons into circuits for computer application. One of the simplest systems will be described. Cryotrons will be represented in the circuit diagrams to be given as shown in Fig. 3a or Fig. 3b.

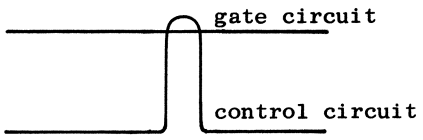


fig. 3a

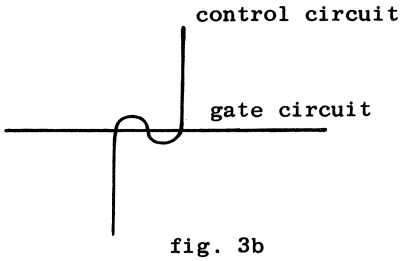


fig. 3b

All of the cryotrons in each circuit are identical and have current gains somewhat greater than one. The power supply current I is chosen to be greater than I_c for the cryotrons but less than I_g.

The general form of the circuits to be described is shown in Fig. 4. It consists of several parallel paths, each containing a number of cryotrons in series. This circuit operates when all of the cryotrons in one path are superconductive and all of the other paths contain a normal conducting element. Therefore all of the supply current is switched to the superconducting path. Once it has become established, this current encounters no back voltage, thus no energy is dissipated. Only while the current is being switched is power required.

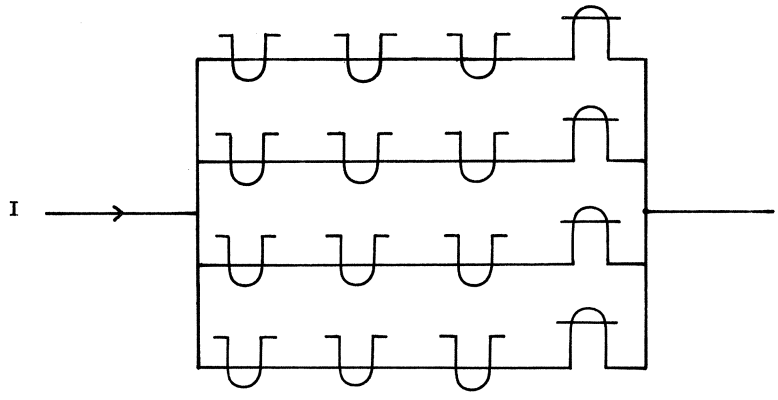


fig. 4

These cryotron circuits exhibit a form of memory. Once current has been established in one of the paths, no back voltage is produced. Therefore, making any or all of the remaining paths superconducting cannot alter the current pattern. The only way to operate the circuit is to introduce a resistance in the conducting path by means of one of its cryotrons.

A very simple static register can be constructed from cryotrons. Fig. 5 shows three binary columns of such a register. The circuit requires two cryotrons per bit. Reading into the register is accomplished by pulsing the appropriate cryotron associated with each bit.

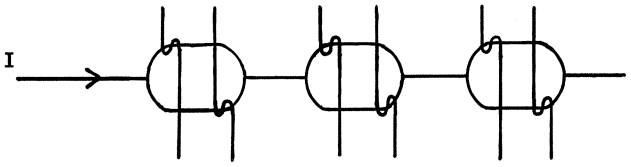


fig. 5

Any switching function can be expressed in the canonical, or "sum of products" forms. Therefore any switching function can be realized by using two cryotron circuits of the type being discussed. This is best explained by an example. Fig. 6 gives a circuit which realizes the function

$$f = xy' \cup x'y.$$

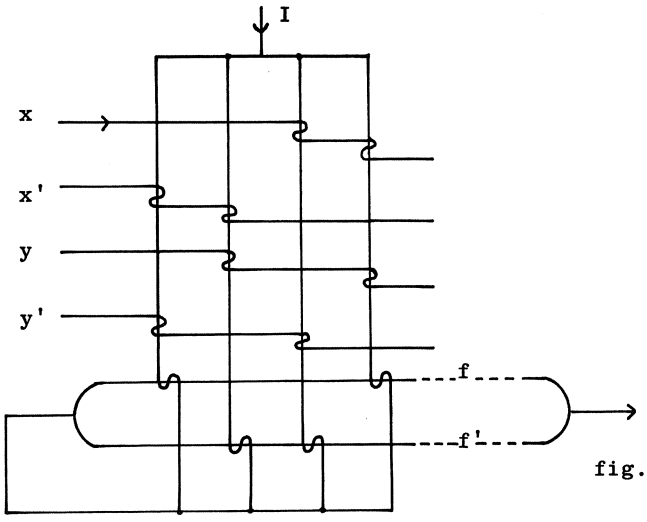


fig. 6

Currents in the wires x, x', y and y' will operate at least one cryotron in each of three of the vertical wires. The supply current is thus forced along the remaining wire. This current will then operate one of the cryotrons in the wires f and f' and thereby cause the proper output.

Conclusions

The greatest disadvantage of cryotron circuitry is, of course, that it must be submerged in liquid helium. While helium liquifiers are commercially available, they are rather expensive. Therefore cryotrons can only be justified for the largest of installations. With regard to speed, the cryotrons constructed by the evaporation process are able to compete with any known computer device. The size, reliability, and noise level of these devices is favorable. Cryotrons are constructed from cheap materials, and the fabrication does not seem particularly difficult. Therefore they should be cheaper than conventional components, but not drastically so.

1. H. Kamerlingh Onnes, Leiden Comm., 122b, 124c (1911).
2. H. Kamerlingh Onnes and W. Tuyn, Proc. Acad. Sci. Amsterdam, 25, 443 (1923).
3. H. Kamerlingh Onnes, Leiden Comm., Supplement 35 (1913).
4. W. Meissner and R. Ochsenfeld, Naturwissenschaften, 21, 787 (1933).
5. F. London and H. London, Physica, 2, 341 (1935).
6. D.A. Buck, Proc. I.R.E., 44, 482 (april 1956).

ENIGE ORDENINGSPROBLEMEN

E.W. Dijkstra

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 23 februari 1957 te Amsterdam.

Zonder ons nu uitgebreid bezig te gaan houden met algemene rangschikkings- en sorteerproblemen, willen wij vandaag vier ordeningsproblemen behandelen, die (als onderdeel van grotere vraagstukken) op de ARMAC inderdaad zijn opgelost.

Sorteerproblemen zijn voor machines als de ARMAC n.l. niet uitgesproken aantrekkelijk. Dit mede heeft tot gevolg gehad, dat wij niet voor de opgave van "groot sorteerwerk" met de ARMAC zijn geplaatst, zodat wij daarmede tot nog toe geen ervaring hebben opgedaan. Dit feit zij de ene rechtvaardiging voor beperkte en specifieke behandeling van dit onderwerp; de andere rechtvaardiging is daarin gelegen, dat bij het wel uitgevoerde rangschikkingswerk dankzij analoge technieken het aantal benodigde handelingen binnen de perken gehouden kon worden.

1. Lijnbezettingsgegevens bij simulatie van een telefooncentrale

De opgave was, om na te gaan welke bezetting in een model van een telefooncentrale gemiddeld zou optreden onder een aangenomen stochastische verdeling van oproeptijden (= tijdsintervallen tussen twee opeenvolgende oproepmomenten) en houdtijden (= gespreksduren).

Om dit na te kunnen gaan was het noodzakelijk, dat nauwgezet werd bijgehouden, welke abbonnées in gesprek waren, over welke lijn en tot hoelang. Op het moment, dat een abbonnée verbonden werd - een vrije lijn gevonden had etc. - werd n.l. de gespreksduur geloot en stond dus het z.g. afvalmoment (= moment van verbreking van het gesprek) vast.

De lijnen waren genummerd van 1 t/m N; hun aantal N was een gegeven van de centrale. De lijnbezettings-gegevens per lijn bevatten, of via deze lijn gesproken werd en zo ja (door welke abbonnée en) tot welk afvalmoment.

Voor diverse administratieve handelingen was het gewenst, de lijnbezettings-gegevens in het geheugen te rangschikken naar oplopend lijnnummer. Een tweede gewenste ordening was echter naar oplopend afvalmoment.

Immers: zodra een oproepmoment voor een nieuw gesprek in de nabije toekomst geloot is, moeten alle gesprekken met een afvalmoment vroeger dan dit oproepmoment verbroken worden, voordat onderzocht kan worden, of voor dit nieuwe gesprek een lijn beschikbaar is.

Om in staat te zijn, zonder tijdrovend gezoek de lijnbezettingsgegevens af te kunnen werken in volgorde van oplopend afvalmoment, (te beginnen bij het vroegste afvalmoment tot aan het eerste afvalmoment dat na het bewuste oproepmoment valt) is aan de lijnbezettingsgegevens van elke lijn een z.g. "loodsmannetje" toegevoegd, dat verwijst naar het nummer van de lijn (d.w.z. de adressen der bezettings-gegevens), met het vroegste latere afvalmoment.Zet men bovendien op een <u>vaste</u> plaats een eenzaam loodsmannetje, dat verwijst naar de lijn met het vroegste afvalmoment, dan is het duidelijk, dat men, beginnend bij dit eenzame loodsmannetje, de lijnbezettings-gegevens kan afwerken in volgorde van oplopend afvalmoment.

Na dit aantal (mag = 0 zijn!) verbrekingen kan het nieuwe gesprek tot stand gebracht worden, mits er nu een lijn beschikbaar is. Als dit - zoals gelukkig meestal! - het geval is, moeten de bezettings-gegevens van deze lijn t.g.v. het nieuwe gesprek genoteerd worden èn opgenomen worden in de ordening naar afvalmoment. Aangezien het hier het afvalmoment van het laatste tot stand gekomen gesprek betreft, zal de plaats zijn in de buurt van de laatste afvalmomenten. (Immers: duurden alle gesprekken even lang, dan kwam het nieuwe gesprek gegarandeerd helemaal achteraan.) M.a.w. om de lijnbezettings-gegevens t.g.v. een nieuw gesprek zo efficiënt mogelijk in de ordening hun plaats te kunnen geven, wil men

de lijnbezettingsgegevens af kunnen tasten te beginnen bij die met het laatste afvalmoment en dan verder in de volgorde van teruglopend afvalmoment. Daartoe is aan de bezettingsgegevens voor elke lijn een tweede loodsmannetje toegevoegd, dat verwijst naar de lijn met het laatste vroegere afvalmoment, terwijl weer op een vaste plaats een eenzaam loodsmannetje verwijst naar de lijn met het allerlaatste afvalmoment om het aftasten te kunnen beginnen.

Om aan de rangschikking der elementen een element te ontnemen of toe te voegen, hoeft men alleen wijzigingen aan te brengen in loodsmannetjes der naburige elementen.

De details van het programma laten wij hier gaarne buiten beschouwing; hoewel het proces recht toe recht aan loopt, is het programma, naar onze maatstaf althans, tamelijk ingewikkeld. Wij volstaan met de vermelding dat het programma voor het wegnemen en het toevoegen van een element beduidend vereenvoudigd wordt, zodra, zoals in bovengenoemd geval, elk element zowel van het "voorwaartse" als ook het "achterwaartse" loodsmannetje is voorzien.

Dat het proces nogal veel geheugen gebruikt, was in de zojuist geschetste toepassing geen bezwaar. Aangezien het hier ging om het maken en verbreken van vele duizenden gesprekken, was echter snelheid een absolute vereiste.

Tot slot zij de aandacht er op gevestigd, dat het met deze techniek mogelijk is, om elementen simultaan te rangschikken naar verschillende criteria; elementen hoeven niet in alle rangschikkingen opgenomen te zijn.

2. De kortste route via een wegennet

Als een aantal steden, benevens een wegennet, dat deze steden onderling verbindt, gegeven is, wordt gevraagd tussen twee willekeurig gekozen steden, de kortste route vast te stellen. De vorm, waarin de onderlinge samenhang van het wegennet gegeven is, is de volgende: in een trajectentabel is van elke stad gegeven, welke haar "buursteden" zijn en op welke afstand zij liggen, waar onder "buursteden" verstaan wordt twee steden die met elkaar verbonden zijn via een weg, die geen andere steden passeert.

Afgezien van één-richting verkeer komt elk wegstuk dus twee-maal in de trajectentabel voor, n.l. bij zijn ene en bij zijn andere einde.

Van het overstelpend aantal mogelijkheden om van het vertrekpunt A de bestemming B te bereiken, kan het overgrote deel geëcarteerd worden op grond van de volgende overweging: als de kortste route van A naar B loopt via een stad C, dan bestaat deze route uit de kortste route van A naar C, gevolgd door de kortste route van C naar B. Het is evident dat de "tussenstad" C dichter bij B ligt, dan A. Als bij de uitgangsstad A bekend is, langs welke harer buursteden de korste route naar B loopt, dan is dit gegeven voldoende, om de kortste route van A naar B af te leggen, mits ditzelfde gegeven bekend is voor alle steden, die dichter bij B liggen dan A. Dan kan men de eerst bereikte tussenstad n.l. opnieuw als uitgangspunt beschouwen en vandaar de kortste route naar B inslaan. Dat dit gegeven voor alle steden, dichter bij B dan A bekend is, is kennelijk een misschien niet nodige, maar wel voldoende voorwaarde.

Deze overwegingen suggereren, om de omgeving van B te verkennen, en van alle steden te noteren, via welke buurstad de kortste weg naar B loopt, en wel in volgorde van opklimmend kortste afstand naar B. Zodra in deze lijst de stad A voorkomt, is het probleem opgelost.

Om in te zien, dat hiermede het zoekproces tot redelijke proporties is teruggebracht, verdelen we onze steden in drie groepen.

De eerste groep omvat alle steden, waarvan op dat moment al bekend is, via welke buurstad de kortste route naar B begint, en zal zo samengesteld zijn, dat als een stad in de eerste groep voorkomt, ook alle "dichter" bij B gelegen steden reeds in de eerste groep voorkomen. Van elke stad in de eerste groep wordt genoteerd, via welke harer buursteden de kortste route naar B loopt èn hoeveel de totale (kortste) afstand naar B bedraagt. De steden in de eerste groep zijn er aan toegevoegd in volgorde van opklimmende kortste afstand naar B.

De tweede groep omvat alle steden die (als uitgangspunt beschouwd) een buurstad (als bestemming) in de eerste groep hebben; als dit er verscheidene zijn, wordt die stad uit de eerste groep gekozen, die aanleiding geeft tot de minimum afstand tot B. Ook van elke stad uit de

tweede groep wordt de (voorlopig beste) buurstad op weg naar B genoteerd, benevens de afstand naar B via deze buurstad.

De derde groep omvat alle andere steden.

Het proces, dat deze groepen opbouwt, bestaat uit twee stappen, die elkaar afwisselen.

Bij de ene stap wordt de dichtstbijgelegen stad uit de tweede groep overgeplaatst naar de eerste groep, laat dit de stad C zijn; bij de andere stap worden alle steden vanwaar een directe weg naar C leidt (deze worden in de trajectentabel opgezocht) aan een onderzoek onderworpen: als zo'n stad nog niet in de eerste of tweede groep voorkomt, wordt ze aan de tweede groep toegevoegd; als ze al in de eerste groep voorkomt, wijzigen we niets in de tweede groep, als ze al in de tweede groep voorkomt, wordt vergeleken of de daar genoteerde afstand groter dan wel kleiner is dan de afstand via C: de kortste wordt gekozen en zo nodig wordt de notitie in de tweede groep gewijzigd.

Als alle steden, die blijkens de trajectentabel een route naar C hebben, zo zijn afgewerkt, wordt weer de eerste stap uitgevoerd, etc. totdat de stad A aan de eerste groep wordt toegevoegd.

Het proces begint met de eerste en tweede groep leeg: B wordt (met een afstand = 0) aan de eerste groep toegevoegd.

Omdat in de eerste en tweede groep samen elke stad hoogstens éénmaal voorkomt, zijn deze groepen samen nooit groter dan het totale aantal steden op de kaart. Omdat bovendien bij elke tweede stap een nieuwe stad aan de eerste groep wordt toegevoegd, is voor een bescheiden aantal steden het einde van het proces wel te zien.

Behoefte aan ervaring en nieuwsgierigheid naar de uiteindelijke macht van een dergelijke methode zijn aanleiding geweest, om voor een kaart met 64 steden het zojuist beschreven systeem te programmeren. Na enige "aankleding" was het directe resultaat van deze inspanning een demonstratieprogramma; de opgedane ervaring zou ons sneller dan verwacht van pas komen.

3. De kortste boom, die n punten onderling verbindt

Dit probleem is een bekabelingsprobleem: n gegeven soldeerpunten moeten onderling verbonden worden door de boom - opgebouwd uit n - 1 verbindingen van soldeerpunt tot soldeerpunt - die de minimale hoeveelheid draad vereist.

De punten zijn gegeven in een soort coördinaten; verder is gegeven, hoe men uit deze coördinaten van twee punten de lengte van de draad berekent, die deze punten direct met elkaar verbindt.

Het is onmogelijk om al deze bomen te beschouwen, die deze n punten onderling verbinden, en dan de kortste te kiezen. Sinds Cayley is n.l. bekend, dat dit aantal n^{n-2} bedraagt, terwijl n in de gegeven opgave variëert van 10 tot 30. Het is gelukkig ook niet nodig.

Voor het zoek-proces verdelen we de punten in twee groepen, de eerste groep (die aanvankelijk leeg is) bevat m punten, die door m - 1 verbindingen uit de kortste boom onderling zijn verbonden, de tweede groep bevat de resterende punten. Punt A zij een punt uit de tweede groep; onder de "afstand van A tot de eerste groep" verstaan wij de afstand van A tot het dichtst bij A gelegen punt uit de eerste groep.

Het proces bestaat daaruit, dat in successie punten aan de eerste groep worden toegevoegd; is m = n, dan is de oplossing gevonden.

Stel dat de vorming van de eerste groep gedeeltelijk gevorderd is: in de uiteindelijk kortste boom is minstens één punt uit de eerste groep verbonden met minstens één van de zich nog in de tweede groep bevindende punten. Uit de tweede groep is hierbij altijd het punt, dat de kortste afstand tot de eerste groep bezit. Als wij hebben bijgehouden, hoelang de afstanden van de punten uit de tweede groep tot de eerste groep zijn, komt deze selectie neer op het minimum bepalen van n - m afstanden. Als wij bovendien voor de punten uit de tweede groep hebben bijgehouden via welk punt van de eerste groep de betroffen afstand gerealiseerd werd, dan hebben wij niet alleen een punt aan de eerste groep toegevoegd, maar eveneens aan de kortsteboom een verbinding, gekarakteriseerd door zijn beide eindpunten.

In het geheugen worden directe verbindingen - door begin- en eindpunt gekarakteriseerd - onthouden; tevens wordt van elke onthouden verbinding de lengte bewaard.

Er zijn m - 1 verbindingen, die samen alle gegevens omtrent de eerste groep samenvatten. De tweede groep wordt geborgen aan de hand van de n - m verbindingen, die de afstanden van de n - m punten uit de tweede groep tot de eerste groep realiseren; deze verbindingen hebben dus alle één eindpunt in de eerste groep en één in de tweede groep, terwijl elk punt uit de tweede groep precies eenmaal voorkomt.

Het proces bestaat uit twee stappen, die elkaar afwisselen. Bij de ene stap wordt het punt (en tevens zijn verbinding) uit de tweede groep, dat de kortste afstand tot de eerste groep heeft, aan de eerste groep toegevoegd. Laat dit punt C zijn. Bij de andere stap wordt voor de resterende punten in de tweede groep nagegaan of hun afstand tot de eerste groep ook is afgenomen doordat punt C hieraan is toegevoegd. Dit impliceert dat de afstand van punt C tot elk der resterende punten in de tweede groep wordt berekend en wordt vergeleken met de genoteerde. Is de nieuwe berekende afstand kleiner, dan laat men de verbinding met C de oorspronkelijk genoteerde vervangen.

Om het proces te starten, plaatst men een willekeurig punt in de eerste groep, en tabelleert de afstanden van dit punt tot de andere, en men begint met de stap, die het dichtstbijgelegen punt aan de eerste groep toevoegt.

Men verifieert gemakkelijk, dat in dit proces elk der $\frac{1}{2}(n - 1)$ afstanden éénmaal berekend wordt, en dat de hoeveelheid handelingen evenredig is met n^2. De voor de tussenresultaten benodigde geheugenruimte is evenredig met n.

4. Een langste "sliert" in een boom tussen n punten

Vervolgens is gevraagd om in een boom, zoals deze bij het derde vraagstuk is gevonden, die onvertakte draad aan te wijzen, die het maximum aantal punten verbindt. Dit noemen we een sliert.(Als - wat veel voorkomt - meer dan een sliert dit maximum aantal punten verbindt, mag een willekeurige van deze langsten gekozen worden.) Omdat wij alleen het aantal punten

in onze beschouwing opnemen, kunnen wij over de afstand van twee punten praten, als wij aan elke directe verbinding nu dezelfde lengte - de eenheid - toekennen.

Een punt, dat het verste van een willekeurig punt A verwijderd is, is een eindpunt van een langste sliert; kiest men vervolgens dit eindpunt als punt A en zoekt daarbij het verst verwijderde punt, dan zijn twee eindpunten van een langste sliert bekend; hierdoor is de sliert kennelijk bepaald.

Het bepalen van het verst van A verwijderde punt kan recht toe recht aan geschieden, dankzij het feit dat de beschikbare verbindingen door hun eindpunten zijn gegeven. Men rangschikt - vindt! - de punten naar opklimmende - althans niet dalende - afstand van A. Het laatste punt is het verst verwijderd.

Deze voorbeelden mogen illustreren, dat bij ordeningsproblemen, waar men de elementen niet volslagen chaotisch tot zijn beschikking krijgt, dankzij de flexibiliteit van de automatische rekenmachine van hun onderlinge relaties met vrucht gebruik kan maken. De efficiency van bovengenoemde processen is in hoge mate ervan afhankelijk in welke vorm een en ander in het geheugen onthouden wordt.

THE STATE OF COMPUTER CIRCUITS CONTAINING MEMORY ELEMENTS

A. van Wijngaarden

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 30 maart 1957 te Amsterdam.

Introduction

The object of this paper is to investigate the behaviour of digital switching circuits which contain memory elements.

Let us consider a box B_{kl} with $\underline{k}$ input lines and $\underline{l}$ output lines. On a certain "moment" or "step" $\underline{n}$ on each input line an input u_n^i, $i = 1(1)k$, in the form of a digit in the scale of $m \geq 2$ is presented to the box which reacts by emitting on the output lines again digits in the scale of $\underline{m}$, v_n^j, say $j = 1(1)l$. If $\underline{k}$ resp. $l = 1$, the superscript i resp. j and $\underline{k}$ resp. $\underline{l}$ will be omitted. Physically, there are, of course, time delays involved and the switching processes may be more involved than is assumed in the schematisation followed below, but this is of no importance for our purpose.

What is important, is that the v_n^j not only depend on the u_n^i since the box is supposed to contain memory elements that influence its coding function. Moreover the contents of these memory elements will also be changed after the step so that the state in which the box is left is different from what it was before, and hence also its coding function is changed. This state of the box, A_n, defined in some way or another by the digits in the memory elements together with the internal construction of the box can therefore be considered as part of the input as well as part of the output. It forms part of the input in so far as it governs together with the external input u_n^i the output, but it forms part of the output since the new state A_{n+1} forms with the external output v_n^j the total output of the box.

If the digits a contained in the memory elements are numbered in a prescribed manner a_r, one can form a number written in the scale of m, $\sum a_r m^r$, and this number fully determines the state of the box, since inversely the value of this number determines its digits when written in the scale of m. The state A_n is, therefore, representable by a number, but the form given is on the one hand too vague and on the other hand perhaps too clumsy. Indeed, one wants to have an ordering scheme and also, in general, many of the states so defined might be equivalent for the operation of the box. Only a certain function of the a_r's is important, in general, and even some a_r's might be fully superfluous, so that a number with smaller range of variation may do perhaps. A tentative approach for simple boxes is dealt with below.

Basic units

A very simple box is the one digit delay line, D_1, which stores a digit d_n, and, when a digit u_n is presented at its only input, emits this digit d_n as only external output v_n but stores u_n as new contents d_{n+1}. It is reasonable to identify the state A_n with d_n, whence $v_n = A_n$, $A_{n+1} = u_n$. Since all digits d in the scale of m, like d_n, u_n and v_n, satisfy the condition $0 \leq d \leq m-1$, the two equations can also be written as one equation

$$mA_{n+1} + v_n = A_n + mu_n . \qquad (1)$$

Indeed, from A_n and u_n a complete input is made as a number written in the scale of m and the equation (1) states that A_{n+1} resp. v_n are the quotient resp. the remainder of this number when divided by m. Since only two quantities, A_{n+1} and v_n have to be determined equation (1) which is a special case of

$$m'A_{n+1} + v_n = A_n + m'u_n , \quad m' \geq m, \qquad (1')$$

is sufficient, but it is useful to remember that the form (1') would also do. It does involve, however, larger numbers, but its use will appear later on.

Another simple box is the two digit serial adder, C_2, which remembers

a carry c_n, result of a former addition. When on its two input lines digits u_n^1 and u_n^2 are presented a sum $u_n^1 + u_n^2 + c_n = mc_{n+1} + v_n$ is formed, where the sum digit v_n is emitted and the new carry c_{n+1} is stored instead of c_n. If the stored carry c_n is identified with the state A_n of the adder one can again describe its operation by a single equation, viz.

$$mA_{n+1} + v_n = A_n + u_n^1 + u_n^2. \tag{2}$$

It should be realised that, even when $m > 3$, c_n is a digit in the scale of m that cannot reach with increasing n values > 2. Even the value $c_n = 2$ can only be maintained as long as it had that value once and from then all $u_n^1 = u_n^2 = m - 1$. Even if the construction of the adder was so silly that it could ever contain a carry > 1 then still the carry would degenerate to at most 2. This is an example of a general property of boxes with memory, which will be discussed later on. If in examples below a C_2 is used, it is understood that it cannot contain a carry $c_n > 1$. Of course in the most important case $m = 2$, this question does not arise at all.

Equations (1) and (2) are special cases of the following one

$$mA_{n+1} + Qv_n = A + P_i u_n^i \quad , \quad (m,Q) = 1, \tag{3}$$

where the P_i and Q are integers and $P_i u^i$ stands for $P_1 u^1 + P_2 u^2 + \ldots + P_k u^k$. No restrictions are put to the P_i, but the requirement that the greatest common divisor of m and Q, $(m,Q) = 1$, guarantees that (3) determines both A_{n+1} and v_n. It is, therefore, of interest to see what other boxes than the two simple ones mentioned above satisfy (3) with suitably chosen P_i and Q.

Linear boxes

A box is called a linear box with one output, B_k if it satisfies equation (3) with suitably chosen integers P_i resp. Q, called input moduli resp. output modulus Q. Examples are:

One digit delay line, D_1: $k = 1$, $P = m$, $Q = 1$,
A_n is the digit d_n stored in the delay line.

Two digit serial adder, C_2: $k = 2$, $P_1 = P_2 = Q = 1$,
A_n is the carry c_n stored in the adder.

Other simple examples are furnished by some circuits that do not contain any memory elements, but that can, of course, by definition said to remember a constant, i.e. a number independent of n. Three of those circuits with one input and one output exist, viz.:

Identity, I: $k = 1$, $P = Q \neq 0$, e.g. $P = Q = 1$, $A_n = 0$.
Function: $v_n = u_n$, physical realisations: copper wire, cathod follower, etc.

Invertor, N: $k = 1$, $P = 1$, $Q = -1$, $A_n = 1$.
Function: $v_n = m - 1 - u_n$, physical realisations: triode, "the other polarity", etc.

Emitter, E: $k = 1$, $P = 0$, $Q = 1 - m$, $A_n = d$ $(0 \leq d \leq m-1)$.
Function: $v_n = d$, physical realisations: digit emitter, clock pulse, etc.

The last circuit could of course just as well be considered as a box with no input at all, $k = 0$, but it has advantages to credit any box, formally at least one input. Linear boxes with two or more proper inputs and one output but without memory function, with constant A_n therefore, do not exist if u_n^1 and u_n^2 are uncorrelated since otherwise v_n would not always be a digit in the scale of $\underline{m}$.

That there are many other, more complicated linear boxes, follows from the following theorem:

If the output of a linear box $B'_{k'}$, is used as j-th input of a linear box $B''_{k''}$, there results a linear box $B_{k'+k''-1}$.
The trivial proof that yields also the moduli of the new box as well as the definition of its state runs as follows. By definition one has

$$mA'_{n+1} + Q'v'_n = A'_n + P'_1u'^1_n + \ldots + P'_{k'}u'^{k'}_n,$$

$$mA''_{n+1} + Q''v''_n = A''_n + P''_1u''^1_n + \ldots + P''_{j-1}u''^{j-1}_n + P''_jv'_n + P''_{j+1}u''^{j+1}_n + \ldots + P''_{k''}u''^{k''}_n.$$

If one eliminates v_n' from these equations and if one puts

$$A_n = P''_j A'_n + Q' A''_n,$$

$$u_n^1 = u_n'^1; \ \ldots \ u_n^{k'} = u_n'^{k'},$$

$$u_n^{k'+1} = u_n''^1; \ \ldots \ u_n^{k'+j-1} = u_n''^{j-1}; \ u_n^{k'+j} = u_n''^{j+1}; \ \ldots \ u_n^{k'+k''-1} = u_n''^{k''};$$

$$v_n = v_n''$$

$$P_1 = P_1' P_j''; \ \ldots \ P_{k'} = P_{k'}' P_j'';$$

$$P_{k'+1} = Q' P_1''; \ \ldots \ P_{k'+j-1} = Q' P_{j-1}''; \ P_{k'+j} = Q' P_{j+1}''; \ \ldots \ P_{k'+k''-1} = Q' P_{k''}'';$$

$$Q = Q' Q'',$$

then one finds

$$mA_{n+1} + Qv_n = A_n + P_i u_n^i$$

and, since moreover by definition $(Q',m) = (Q'',m) = 1$, also $(Q,m) = 1$. Hence the assembly of B' and B'' is a new box that satisfies (3) and is therefore a linear box.

This theorem learns how to construct and analyze arbitrarily complicated linear boxes from elementary building stones.

As first application let us take the case of a B_1' in series with another B_1'', what yields therefore a B_1, and

$$\begin{aligned} A_n &= P'' A'_n + Q' A''_n, \\ P &= P' P'', \\ Q &= Q' Q''. \end{aligned} \tag{5}$$

This yields e.g. the result that the s-digit delay line, D_s, which stores a number $S_n = d_n^0 + d_n^1 m + \ldots + d_n^{s-1} m^{s-1}$ is a linear box. Here the least significant digit d_n^0 is at the output side and the most significant digit d_n^{s-1} at the input side of the delay line. If one guesses that this is true and that $P = m^s$, $Q = 1$, what is correct for $s = 1$, then by

considering a D_{s+1} as a D'_s followed by a D''_1, one has $A_n = 2A'_n + A''_n$, $P = m^s \cdot m^s = m^{s+1}$ and $Q = 1 \cdot 1 = 1$, whence the truth of the assumption follows by induction.

Another simple case is afforded by a linear box B' with input u_n^1, used as input of an adder C_2 the other input of which is u_n^2. Then

$$A_n = A'_n + Q'c_n;\ P_1 = P';\ P_2 = Q';\ Q = Q'. \tag{6}$$

Again if two linear boxes B' resp. B'' with inputs u_n^1 resp. u_n^2 feed the inputs of an adder C_2 then

$$A_n = Q''A'_n + Q'A''_n + Q'Q''c_n;\ P_1 = P'Q'';\ P_2 = P''Q';\ Q = Q'Q''. \tag{6'}$$

The equations (6) are a special case of (6') if one chooses the box B'' to be an identity with $A'' = 0$, $P'' = Q'' = 1$. One sees here an example of the use of the identity as a switching element since it diminishes the number of formulas to be at hand.

Another element in the construction of linear boxes is tying together some of the inputs of a box or the output to some of the inputs. The first procedure is very simple. If the input u_n^i is also fed into the input for u_n^j, thus $u_n^j = u_n^i$, then k is diminished by one and P_i is replaced by $P_i + P_j$. A little more care must be taken when the output v_n is used as input, e.g. $u_n^i = v_n$. This should only then be done if u_n^i does not influence v_n, but only v_{n+r} with $r > 0$, since otherwise logical difficulties may arise. This means that P_i is divisible by m. If so, then k is diminished by 1 and Q is replaced by $Q - P_i$. Since $m \mid P_i$ one sees that anew $(m, Q-P_i) = 1$.

The condition $m \mid P_i$ could actually be relaxed to the condition $(m, Q-P_i) = 1$, but whether a digital circuit would work on this basis depends too much on the physical background than that it could be discussed here. Moreover, in the important case $m = 2$ both conditions are apparently equivalent.

Degeneration

If one writes

$$x^+ = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}, \quad \text{and} \quad x^- = \begin{cases} 0 & \text{if } x \geq 0 \\ x & \text{if } x < 0 \end{cases},$$

then from (3) one has

$$mA_{n+1} \leq A_n + (m-1)(P_1^+ + \ldots P_k^+ - Q^-)$$

or if one puts $M_{max} = P_1^+ + \ldots P_k^+ - Q^-$

$$mA_{n+1} \leq A_n + (m-1)M_{max}.$$

The equality sign can be realised under certain circumstances. First of all one should choose $u_n^i = 0$ for those i for which $P_i > 0$, and if it then so happens that $v_n = 0$ if $Q > 0$ or $v_n = m-1$ if $Q < 0$ then the equality holds. Anyhow, if the construction of the box permits to start with $A_0 > M_{max}$ then it follows $mA_1 < A_0 + (m-1)A_0 = mA_0$, whence $A_1 \leq A_0 - 1$. If $A_1 > M_{max}$ then the same reasoning gives $A_2 \leq A_1 - 1$, and so on, and after a finite number of steps one has $A_n \leq M_{max}$.

If $A_n < M_{max}$ then

$$mA_{n+1} < M_{max} + (m-1)M_{max} = mM_{max},$$

whence also $A_{n+1} < M_{max}$.

If $A_n = M_{max}$ then it may happen that also $A_{n+1} = M_{max}$, and it is necessary and sufficient to choose $u_n^i = 0$ resp. m-1 corresponding to $P_i < 0$ resp. > 0. Indeed one has

$$mA_{n+1} = M_{max} + (m-1)(P_1^+ + \ldots + P_k^+) - Qv =$$

$$= M_{max} + (m-1)M_{max} + (m-1)Q^- - Qv,$$

whence

$$(m-1)Q^- \equiv Qv \pmod{m}.$$

If $Q > 0$ then $Q^- = 0$ and in view of $(m,Q) = 1$ one has $v \equiv 0 \pmod m$ or in view of $0 \leq v \leq m-1$, $v = 0$, whence $A_{n+1} = M_{max}$. Again, if $Q < 0$ then $Q^- = Q$, whence $v \equiv m-1 \pmod m$, whence $v = m-1$ and again $A_{n+1} = M_{max}$.

In the same way, with $M_{min} = P_1^- + \ldots + P_k^- - Q^+$ one finds that if $A_n < M_{min}$ then $A_{n+1} \geq A_n + 1$ so that after at most a finite number of steps one has $A_n \geq M_{min}$. Also, if $A_n > M_{min}$ then also $A_{n+1} > M_{min}$, and if $A_n = M_{min}$ then $A_{n+1} \geq M_{min}$ where the equality can be realised by proper choice of the input alone.

The quantities M_{max} resp. M_{min}, the maximum resp. minimum modulus of the box follow from its construction, and although this construction may be such that values $A_n > M_{max}$ or $A_n < M_{min}$ are not excluded a priori, nevertheless there is for such a box a finite positive number N independent of the sequence u_n^i such that for all $n > N$ holds

$$M_{min} \leq A_n \leq M_{max},$$

and if for a value of $n > N$ one knows that instead of $\leq$ holds $<$ on one of the sides resp. on both sides then for all greater values of n the $<$ sign holds on that side resp. on both sides.

This degeneration of the memory contents of such a box is a typical example of the effects of poor internal structure of an organization which misses the capability of dealing with a high amount of information available originally. A simple example was found in the adder C_2. Here $P_1 = P_2 = Q = 1$, whence $M_{max} = 2$ and $M_{min} = -1$. Even if a carry $c_n > 1$ is not excluded it soon holds $c_n \leq 2$, and as soon as once $c_n \leq 1$ has occurred this will hold forever. The construction of the adder is such that a negative $A_n = c_n$ cannot occur, so that the bound $-1 \leq c_n$ is automatically refined to $0 \leq c_n$.

Coders and decoders

A linear box B_1' translates a sequence of digits u_n into another sequence v_n, it acts as a coder therefore. It follows from (3) that

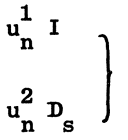
$$mA'_{n+1} - P'u_n = A'_n - Q'v_n,$$

so that when A'_n and the v_n are known and if $(m,P') = 1$, the new state A'_{n+1} and the unknown input u_n can be found. Hence if $(m,P') = 1$ there exist a decoding scheme which can be realised by a conjugate box B''_1 which is such that

$$A''_n = A'_n,\ P'' = -Q' \text{ and } Q'' = -P'.$$

If one puts the two boxes B'_1 and B''_1 in series they act as an identity. Indeed one finds with the aid of (5) for the box $B = B'B''$ that $A_n = P''A'_n + Q'A''_n = -Q'A'_n + Q'A'_n = 0$; $P = P'P'' = -P'Q'$, and $Q = Q'Q'' = -P'Q'$ whence $A_n = 0$ and $P = Q$ which define the identity.

An example of such a box B'_1 and its conjugate B''_1 is furnished by means of the following box B_2:



By means of a switch the auxiliary input u_n^2 can be either identified with $u_n^1 = u_n$, what yields the box B', or with the output v_n, what yields the box B''. This last feedback is permitted, since D_s is a delay line with $s > 0$ digits, so that its input modulus is divisible by $\underline{m}$. In order to show the validity of this assertion one computes the characteristic quantities of B' and B'' in a scheme as below. The number in the delay line D_s is designated by S_n.

	I	D_s	$\begin{matrix}I\\D_s\end{matrix}\}C_2$	N	B_2	B'	B''	$B \equiv B'B''$
A_n	0	S_n	S_n+c_n	1	S_n+c_n+1	$S'_n+c'_n+1$	$S''_n+c''_n+1$	$S'_n-S''_n+c'_n-c''_n$
P_1	1	m^s	1	1	1	m^s+1	1	m^s+1
P_2			m^s		m^s			
Q	1	1	1	-1	-1	-1	$-(m^s+1)$	m^s+1

Indeed P = Q. There is still the difficulty, however, that not necessarily $A_n = 0$, since the contents of the two adders C_2' and C_2'' and those of the delay lines D_s' and D_s'' do not need to be equal at the start n = 0. But here the degeneration comes in. Since P = Q, one has $M_{min} = M_{max} = 0$ and hence after a finite number of steps $A_n = 0$. This means that the coding decoding scheme is self correcting, and it holds for all boxes B' and B'', not only for this special example.

Boxes with complete feedback

An interesting special problem arises when the output v_n of a box B_k is fed back to all inputs u_n^i. Then $u_n^1 = \ldots = u_n^k = v_n$ and when the modulus M of the box is defined as

$$M = P_1 + P_k - Q$$

equation (3) becomes

$$mA_{n+1} = A_n + Mv_n. \tag{7}$$

Since the feedback required $m \mid P_i$, here for all i = 1, ..., k, and (m,Q) = 1, it follows that (m,M) = 1 so that A_n defines both A_{n+1} and v_n. A_n can only take a finite set of values and is therefore periodic. A systematic treatment of the theory of periods for this and similar equations has been given by DUPARC [1]. Here a simple treatment for the specific case (7) is given.

First of all the question of degeneration. Since $M_{min} = 0$ and $M_{max} = M$ after a finite number of steps holds

$$0 \leq A_n \leq M. \tag{8}$$

Moreover, since there is now only one variable v_n instead of u_n^i and v_n more can be said. If $A_n = 0$ then $Mv_n \equiv 0 \pmod m$ whence $v_n \equiv 0 \pmod m$, whence $v_n = 0$ and $A_{n+1} = 0$. Also if $A_n = M$ then $0 \equiv M(v_n + 1) \pmod m$ whence $v_n + 1 \equiv 0 \pmod m$ whence $v_n = m - 1$ and $A_{n+1} = M$. Moreover $mA_{n+1} \equiv A_n \pmod M$. Therefore, if $A_n \equiv 0 \pmod M$ then also $A_{n+1} \equiv 0 \pmod M$.

---

1) H.J.A. DUPARC, Periodicity properties of certain sets of integers, Proc. Kon. Ned. Akad. Wet., A, LVIII, p. 449-458, 1955.

There are two cases to be distinguished therefore:

i) The trivial case $A_0 = aM$, where $\underline{a}$ is an integer. Then for all $n > N$ one has $A_n = M$ and $v_n = m - 1$ if $a > 0$ or $A_n = 0$ and $v_n = 0$ if $a \leq 0$.

ii) The non trivial case $A_0 \neq aM$. Then for all $n > N$ holds

$$0 < A_n < M. \tag{8'}$$

If it is assumed that the initial steps have been performed one can ask for the periods of A_n and v_n. In the trivial case i these periods are clearly 1. In the case ii A_n is periodic with period C_A and v_n therefore also with period C_v. Since A_n determines v_n, C_A is also a period of v_n, whence $C_v | C_A$. However,

$$mA_{C_v+n+1} - A_{C_v+n} = Mv_{C_v+n} = Mv_n = mA_{n+1} - A_n,$$

whence
$$\begin{aligned} A_{C_v+n} - A_n &= m(A_{C_v+n+1} - A_{n+1}) \\ &= m^2(A_{C_v+n+2} - A_{n+2}) \\ &= \ldots \\ &= m^j(A_{C_v+n+j} - A_{n+j}). \end{aligned}$$

Hence $m^j | (A_{C_v+n} - A_n)$ with arbitrary large j, whence $A_{C_v+n} - A_n = 0$, so that C_v is a period of A_n, thus $C_A | C_v$. Together with $C_v | C_A$ one has $C_v = C_A$. There is therefore only one period to be found, and we can get away with v_n by weakening (7) to a congruence:

$$mA_{n+1} \equiv A_n \pmod{M}.$$

However, it is still awkward that (A_n, M) may differ from 1 although it is clearly independent of n. If one defines $M' = M/(A_0, M)$, $A' = A/(A_0, M)$, then $(A'_n, M') = 1$ and one can weaken the congruence to

$$mA'_{n+1} \equiv A'_n \pmod{M'}.$$

By iterating this congruence one has

$$m^j A'_j \equiv A'_0 \pmod{M'},$$

and in particular for $j = C_A$ what is also the period of A'

$$m^{C_A} A'_{C_A} = m^{C_A} A'_0 \equiv A'_0 \pmod{M'}.$$

If by definition $C_m(M')$ is the exponent of m modulo M', i.e. the smallest positive exponent for which holds

$$m^{C_m(M')} \equiv 1 \pmod{M'},$$

then apparently $C_m(M') | C_A$. Now let be $j = C_m(M')$. Then

$$m^{C_m(M')} A'_{C_m(M')} \equiv A'_{C_m(M')} \equiv A'_0 \pmod{M'},$$

but since $0 < A' < M'$, also

$$A'_{C_m(M')} = A'_0.$$

Thus $C_m(M')$ is a period of A' and hence of A, whence $C_A | C_m(M')$ and since one had already $C_m(M') | C_A$ it follows

$$C_v = C_A = C_m(M'). \tag{9}$$

The determination of the period of A_n and of v_n is thus reduced to a well known problem in number theory and can therefore be considered as solved.

Applications are found for instance in the construction of ring counters where only moderately large periods are wanted, and where moreover A_0 is well defined, so that also M' and thus $C_m(M')$ is determined.

Other applications can be made in the construction of random digit generators. Here very large periods C_v are wanted, and moreover it is useful if the period is independent of A_0, the state of the generator at the start. This can be realised by constructing a box for which (8') holds by its construction only and moreover M is a prime number. Then

$(A_0,M) = 1$ for every choice of A_0, and hence $M' = M$, whence $C_v = C_m(M)$. That it is possible to construct such a box is shown by the following box B:

$$u_n D_t \{_I^{D_s}\} C_2 N v_n,$$

which is turned into a generator by putting u = v. The characteristic quantities are computed according to the scheme below.

	D_t	D_s	$D_t D_s$	$D_t\{_I^{D_s}\}C_2$	N	B
A_n	T_n	S_n	$T_n 2^s+S_n$	$T_n(2^s+1)+S_n+c_n$	1	$T_n(2^s+1)+S_n+c_n+1$
P_1	m^t	m^s	m^{s+t}	$m^{s+t}+m^t$	1	$m^{s+t}+m^t$
Q	1	1	1	1	-1	-1

The modulus $M = m^{s+t} + m^t + 1$. Since $0 \leq T_n \leq m^t-1$ and $0 \leq S_n \leq m^s-1$ one has $1 \leq A_n \leq m^{s+t} + m^t = M - 1$. Hence (8') holds always in virtue of the construction. Whether there exist prime numbers of the form $m^{s+t} + m^t + 1$ is for arbitrary m a difficult question. For m = 2 a lot of examples are known. Since s+t is a measure of the required equipment it should be kept moderate, but on the other hand $C_v = C_m(M)$ should be as high as possible, and since for a prime number M according to Fermat's theorem holds $m^{M-1} \equiv 1 \pmod M$ one has $C_m(M) \mid M - 1$ so that M should be large so that s+t should be large. A reasonable compromise for m = 2 is furnished by choosing s = 10, t = 17, $M = 2^{27} + 2^{17} + 1 = 134348801$ (prime) and $C_v = C_2(M) = \frac{1}{2}(M - 1) = 67174400$.

Extensions

In the foregoing sections only linear boxes with one output were investigated. As soon as one of these two restrictions is released great difficulties arise. No systematic treatment is given, but only some loose remarks are made concerning this subject. For simplicity also m = 2 is taken.

First non linearity may be introduced in the box by switching elements with a non linear character. It is no restriction logically to deal only with the case of switching elements with two inputs and one output. If the inputs are as before u_n^1 and u_n^2 and the output is v_n then as well known all these elements perform a function described by

$$v_n = p_0 + p_1 u_n^1 + p_2 u_n^2 + p_{12} u_n^1 u_n^2.$$

For instance for an and-gate holds $p_0 = p_1 = p_2 = 0$, $p_{12} = 1$ and for a non exclusive or-gate holds $p_0 = 0$, $p_1 = p_2 = 1$, $p_{12} = -1$.

If there are no memory elements in the box then the theory is now well known. If memory elements are present that store digits a_n^0, a_n^1, ..., a_n^r then one can form a state $A_n = a_n^0 + 2a_n^1 + \ldots + 2^r a_n^r$. Moreover one can set up all the equations that govern the switchings performed within the box. For instance in the extremely simple box

$$u_n D_1^1 \left\{ \begin{matrix} D_1^0 \\ I \end{matrix} \right\} \times v_n$$

with two delay lines containing each one digit, and an and-gate, one has

$$d_{n+1}^1 = u_n;\ d_{n+1}^0 = d_n^1;\ v_n = d_n^0 d_n^1.$$

One could define $A_n = d_n^0 + 2d_n^1$. In order to compute $A_{n+1} = d_{n+1}^0 + 2d_{n+1}^1$ and $v_n = d_n^0 d_n^1$ one needs the digits of A_n again separately. This is not impossible, since the digits and each function of the digits of a number A of known finite variation are expressible as functions, e.g. polynomials in A. Thus for our A_n that can only take the values 0, 1, 2 and 3 one has

$$d_n^0 = \frac{1}{3} A_n (A_n - 2)(2A_n - 5),$$

$$d_n^1 = \frac{1}{6} A_n (A_n - 1)(7 - 2A_n),$$

$$d_n^0 d_n^1 = \frac{1}{6} A_n (A_n - 1)(A_n - 2).$$

Hence

$$v_n = \frac{1}{6} A_n(A_n - 1)(A_n - 2),$$

$$A_{n+1} = \frac{1}{6} A_n(A_n - 1)(7 - 2A_n) + 2u_n \tag{10}$$

and the single equation corresponding to (3) that determines both A_{n+1} and v_n becomes

$$2A_{n+1} + v_n = \frac{1}{2} A_n(A_n - 1)(4 - A_n) + 4u_n.$$

One can do better, however, by observing that d_n^0 is not interesting at all but only the product $d_n^0 d_n^1$, and also the simpler definition $A_n = d_n^1(1 + d_n^0)$ will do. This new A_n takes only the values 0, 1 and 2 and one has

$$d_n^1 = \frac{1}{2} A_n(3 - A_n),$$

$$d_n^0 d_n^1 = \frac{1}{2} A(A_n - 1).$$

Hence

$$v_n = \frac{1}{2} A_n(A_n - 1)$$

$$A_{n+1} = \frac{1}{2} (2 + 3A_n - A_n^2)u_n \tag{11}$$

and the equation corresponding to (3) becomes

$$2A_{n+1} + v_n = \frac{1}{2} A_n(A_n - 1) + (2 + 3A_n - A_n^2)u_n.$$

In practice one would write the coefficients depending on A_n in (10) or (11) as columnvectors where the successive elements of the vector denote the values of the coefficients for $A_n = 0, 1, \ldots$. These representations would then run for (10).

$$v_n = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \quad A_{n+1} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} + 2u_n, \tag{10'}$$

whereas for (11) one would obtain

$$v_n = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad , \quad A_{n+1} = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix} u_n . \tag{11'}$$

Also in more complicated cases one can proceed in this way, where especially the vector notation is useful. If the box is such that it contains complicated linear subboxes together with some non linear elements it is more practical to cut the whole structure into pieces to be treated separately than to try to describe the box as a whole.

Another difficulty arises when elements with more than one output are involved. A typical example of such an element is the half adder, for which holds $mv_n^2 + v_{n_2}^1 = u_n^1 + u_n^2$. This is a linear relation, which determines both v_n^1 and v_n^2 but it could, of course, also be written as two non linear relations like e.g., if m = 2, in the form: $v_n^2 = u_n^1 u_n^2$, and $v_n^1 = u_n^1 + u_n^2 - u_n^1 u_n^2$, so that there is a close relationship between multiple output and non linearity.

The occurrence of a term like mv_n^2 would cause difficulty in equation (3) since the contributions of mA_{n+1} and mv_n^2 would be partly undistinguishable. This difficulty can be overcome by remembering that the factor m in front of A_{n+1} was rather arbitrary. Indeed if one returns to the one-digit delay line D_1, then the equation (1') defines also A_{n+1} and v_n, and the choice m' = m was only made in order to keep the most natural representation of the state of the delay line D_s, viz. the number S_n contained in it, when one reads the sequence of digits as a number in the scale of $\underline{m}$. Nothing prohibits, however, to take a larger value of m'. For instance if one takes $m' = n^2$ one can accomodate the term $mv_n^2 + v_n^1$ without interfering with A_{n+1}.

With little changes here and there the theory of the linear boxes can be rewritten with m' instead of m. The only thing that one cannot do, in general, is to recombine the two outputs v_n^1 and v_n^2 again by means of a linear box, since the v_n^1 and v_n^2 are provided with certain output moduli Q_1 and Q_2 and unless these two have the same ratio as that of the input moduli P_1 and P_2 of the recombination box, the usual elimination

procedure will not work. One can, of course, cut the box here in two what solves the difficulty.

ENIGE BESCHOUWINGEN OVER ITERATIEVE PROCESSEN EN NIET LINEAIRE TRANSFORMATIES

J. Berghuis

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 27 april 1957 te Amsterdam.

Zoals bekend kunnen iteratieve processen voor het vinden van (een) wortel(s) van een vergelijking $f(x) = 0$ gegeven worden door

$$x_{n+1} = x_n - c_n f(x_n) \tag{1}$$

waarbij x_n de n^e iterant is en de coëfficiënt c_n mag afhangen van x_n.

Wij veronderstellen voor het gemak dat de gezochte wortel 0 is; dit kan zonder bezwaar gedaan worden: Zij de wortel n.l. a en $\varepsilon_n = x_n - a$, dan gelden de volgende beschouwingen voor ε_n i.p.v. voor x_n.

De orde α van het iteratieve proces is de exponent voorkomende in de relatie

$$x_{n+1} = A\, x_n^{\alpha} + O(x_n^{\alpha+\beta}) \tag{2}$$

waarin A een van x_n onafhankelijke constante is en $\beta > 0$.

Hierbij is gebruik gemaakt van het symbool O ook voorkomende bij asymptotische ontwikkelingen, hetwelk aangeeft dat er een vast van x_n onafhankelijk getal C bestaat zodanig dat de restterm in absolute waarde kleiner is dan $C\,|x_n^{\alpha+\beta}|$.

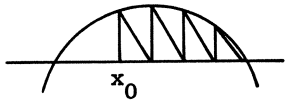
Met behulp van (1) kan indien $f(x)$ en c_n een asymptotische ontwikkeling naar x_n bezitten de relatie (2) worden afgeleid; zelfs merken wij op dat al voldoende is een relatie van de vorm

$$g(x) = \sum_{h=0}^{H-1} g_h\, x^{\phi h} + O(x^{\phi H}) \tag{3}$$

voor één waarde van H.

Om maar eenvoudig te beginnen zullen wij $c_n = c$ een constante voor elke n kiezen. Dit proces als beschreven door von Mises (1929) heeft enkele voor rekenmachines plezierige eigenschappen, ook nadelen.

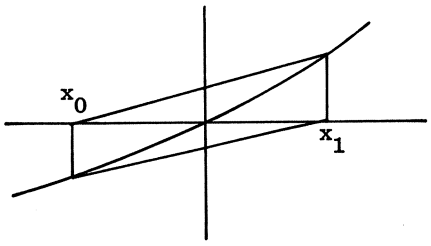
Zij te bepalen alle reële nulpunten van f(x). Indien het mogelijk is $|c|$ te bepalen kleiner dan het omgekeerde van Max $|f'(x)|$ dan vindt men startend in een bepaald punt x_0 een voorgeschreven nulpunt. De waarde van x_n nadert monotoon tot de wortel. Heeft men een keer een wortel gevonden, dan wijzigt men het teken van c, verhoogt of verlaagt de waarde der wortel een klein beetje en itereert door. Een wortel van even multipliciteit wordt twee keer gevonden, één van oneven multipliciteit slechts één keer.



Kiest men c te groot in absolute waarde, dan bestaat het gevaar dat men nulpunten overslaat, hoewel er ook cycli kunnen ontstaan. Voorwaarde hiertoe is $f(x_0) + f(x_1) = 0$

$f(x_0) + f[x_0 - c\, f(x_0)] = 0.$

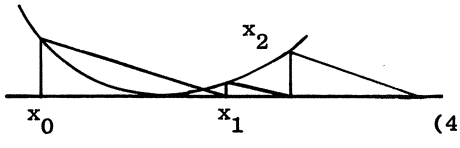
Zelfs meervoudige cycli kunnen ontstaaan en al deze cycli kunnen stabiel of instabiel zijn.



Er dient bedacht te worden dat de grootte van c afhankelijk is van f(x) en dus niet voor elke f(x) hetzelfde genomen kan worden.

We stellen nu dat voor f(x) geldt:

$$f(x) = a_1x + a_2x^2 + O(x^3)$$



(4)

met $a_1 \neq 0$ dus

$$x_{n+1} = (1 - ca_1)x_n + ca_2x_n^2 + O(x_n^3).$$

Dit is dus een lineair proces tenzij $ca_1 = 1$. In dit laatste geval is het proces kwadratisch tenzij $a_2 = 0$ enz.

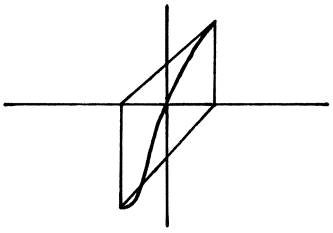
Stel nu

$$f(x) = a_1 x^P + O(x^Q) \quad \text{met} \quad Q > 1,$$

dus

$$x_{n+1} = x_n - c\, f(x_n) = x_n - ca_1 x_n^P + O(x_n^Q).$$

Is nu $P > 1$ dan is de convergentie lineair maar slecht en indien $P < 1$ dan convergeert het proces niet weer; men krijgt meer cycli e.d. Er is echter niet meer voldaan aan $|c| < 1/\text{Max}\,|f'(x)|$.



Een tweede voorbeeld is het proces van Newton

$$c_n = 1/f'(x_n).$$

Dit proces is kwadratisch indien $f(x)$ voldoet aan (4), in alle andere gevallen is het lineair. Voldoet $f(x)$ aan

$$f(x) = a_1 x^p + O(x^{p+1}) \quad p > 0 \tag{5}$$

dan is het proces

$$x_{n+1} = x_n - p\, f(x_n)/f'(x_n)$$

steeds kwadratisch. Een andere manier om een altijd kwadratisch proces af te leiden is de volgende: De functie $\frac{f(x)}{f'(x)}$ bezit slechts enkelvoudige nulpunten en dus geeft het proces van Newton hierop toegepast een kwadratisch proces:

$$x_{n+1} = x_n - \frac{f(x_n) \cdot f'(x_n)}{\{f'(x_n)\}^2 - f(x_n) f''(x_n)}\,. \tag{6}$$

Een kwadratisch proces heeft altijd een convergentiegebied; trouwens de volgende stelling geldt:

Indien de orde α van een iteratieproces groter is dan 1, d.w.z. er geldt:

$$x_{n+1} = Ax_n^{\alpha} + O(x_n^{\alpha+\beta}) \quad \text{met} \quad \alpha > 1 \text{ en } \beta > 0$$

dan is er een omgeving aan te geven zodanig dat het proces convergent is:

Er geldt

$$x_{n+1} = x_n^{\alpha}(A + R_n x_n^{\beta})$$

dus

$$\frac{x_{n+1}}{x_n} = x_n^{\alpha-1}(A + R_n x_n^{\beta}).$$

Volgens gegeven blijft R_n beneden een bepaalde waarde en dus is x_n zo klein te bepalen dat $|A + R_n x_n^{\beta}| < 2|A|$ en dan weer is x_n zo klein te bepalen dat

$$|x_n^{\alpha-1}| < \frac{1}{3A} \text{ , wegens } \alpha - 1 > 0.$$

Hieruit volgt $\left|\frac{x_{n+1}}{x_n}\right| < \frac{2}{3}$ en dus convergentie.

Een lineair proces kan convergent, divergent of oscillatorisch zijn. Het proces van Newton is zeker divergent in die punten waarin geldt $f'(x) = 0$. Ook kunnen hierbij cycli optreden waarbij moet gelden

$$\frac{f(x_0)}{f'(x_0)} + \frac{f(x_1)}{f'(x_1)} = 0.$$

Nog een voorbeeld is Lin's methode van "synthetic division" voor het bepalen van wortels van polynomen.

Zij het polynoom $f(x) = \sum_{i=0}^{k} a_i x^i$ en bepaal de coëfficiënten b_l zodanig dat $b_0 = 0$ en

$$b_l = a_l + xb_{l-1} \quad (l = 1(1)k).$$

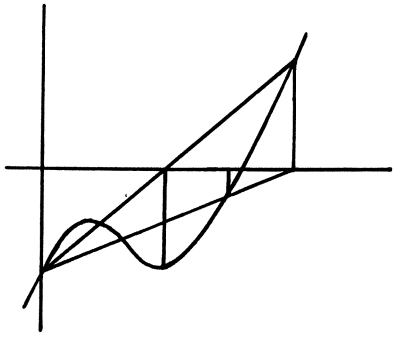
De iteratie volgens Lin is dan

$$x_{n+1} = x_n - \frac{b_k}{b_{k-1}}$$

of anders geschreven

$$x_{n+1} = x_n - \frac{x_n \cdot f(x_n)}{f(x_n) - f(0)} \quad . \tag{7}$$

Divergentie zeker indien $f(x_n) = f(0)$.



Voor de ordebeschouwing van het proces kan men beter de wortel a stellen, $x_n = a + \varepsilon_n$ en dan (4) aannemen.
Er volgt

$$\varepsilon_{n+1} = \left[1 + a\,\frac{f'(a)}{f(0)}\right] \cdot \varepsilon_n + O(\varepsilon_n^2)$$

zodat de divergentie optreedt indien

$$\left|1 + a\,\frac{f'(a)}{f(0)}\right| > 1 \text{ en}$$

convergentie als die factor < 1 is. Het proces is lineair tenzij

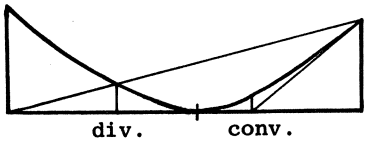
$$a = -\frac{f(0)}{f'(a)} \quad .$$

We schrijven even het (divergente) voorbeeld van Hildebrand (p. 454) over:

x	1.3	1.45	0.91	-5.8
1	1	1	1	1
0	1.3	1.45	0.91	
-1	0.69	1.102	-0.172	
-1	-0.103	0.598	-1.157	
Δx	0.15	-0.54	-6.7	

Treedt divergentie op, dan doet men verstandig het punt x = 0 te verleggen. Neemt men voor f(x) niet (4) maar een andere ontwikkeling aan, dan is de orde van het proces steeds weer te bepalen. De factor $1 + a\,\frac{f'(a)}{f(0)} = 1$ in het geval dat

$f(x) = (x - a)^P + O((x - a)^Q)$ met $p > 1$ en convergentie hangt af van hogere termen.



De methode van Lin is echter een speciaal geval van "Regula Falsi" die wij nu gaan behandelen.

Wij kiezen eerst twee punten x_0 en x_1 en bepalen dan x_2 enz. volgens

$$x_{n+1} = x_n - \frac{x_{n-1} - x_n}{f(x_{n-1}) - f(x_n)} \cdot f(x_n). \tag{8}$$

Nu geldt asymptotisch

$$c_n = \frac{x_{n-1} - x_n}{f(x_{n-1}) - f(x_n)} \sim \frac{1}{f'(\alpha)}$$

dus indien $f(x)$ voldoet aan (4) dan is de orde zeker hoger dan 1. Regula Falsi bezit een convergentiegebied, d.w.z. mits men niet willekeurig twee punten x_0 en x_1 in dat gebied kiest, want $f(x_0) = f(x_1)$ betekent steeds divergentie.

Uit (4) en (8) volgt:

$$x_{n+1} = \frac{a_2}{2a_1} x_n x_{n-1} + \text{hogere termen.}$$

Stelt men nu $x_{n+1} \sim Ax^\alpha$ dan volgt

$$\alpha^2 - \alpha - 1 = 0 \quad \text{of} \quad \alpha = \frac{1 + \sqrt{5}}{2} = 1.618 \quad \text{enz.} \tag{9}$$

Newton en Regula Falsi hangen samen: wat bij de een differentiaalquotiënt is, is bij de ander differentie quotiënt. Indien het rekenwerk vereist voor bepaling van $f(x_n)$ groot is, werkt Regula Falsi sneller dan Newton.

De afleiding van hogere orde iteratieprocessen is gegeven door E. Schröder.

Stelt men $y = f(x)$, dan is x als functie van y te ontwikkelen in de omgeving van x_n:

$$x = x_n + \sum_{h=1}^{k-1} \frac{(y - y_n)^h}{h!} \left(\frac{d}{dy}\right)^h x_n + O((y - y_n)^k).$$

De gevraagde wortel x voldoet aan y = 0 dus

$$x_{n+1} = x_n + \sum_{h=1}^{k-1} \frac{(-1)^h y_n^h}{h!} \left(\frac{d}{dy}\right)^h x_n$$

is een hogere orde iteratieproces.
Schrijf nu weer $f(x_n)$ i.p.v. y_n en

$$\frac{d}{dy} = \frac{d}{df} = \frac{1}{f'} \frac{d}{dx}$$

verder

$$\frac{d}{dy} x_n = \left(\frac{dx}{dy}\right)_{x=x_n} = \frac{1}{f'(x_n)} ,$$

zodat

$$x_{n+1} = x_n + \sum_{h=1}^{k-1} \frac{(-1)^h \{f(x_n)\}^h}{h!} \left[\left\{ \frac{1}{f'(x_n)} \cdot \frac{d}{dx} \right\}^{h-1} \frac{1}{f'(x_n)} \right] \tag{10}$$

het gevraagde proces is.

Een derde proces (als (4) geldt) is bijv.:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{f''(x_n)\{f(x_n)\}^2}{2\{f'(x_n)\}^3} \tag{11}$$

maar ook

$$x_{n+1} = x_n - \frac{2f(x_n)\, f'(x_n)}{2[f'(x_n)]^2 - f(x_n) f''(x_n)} . \tag{12}$$

Dit laatste proces door P. Wynn toegeschreven aan Richmond wordt gebruikt voor een aardige toepassing (MTAC, April 1956). Hij elimineert n.l. de tweede afgeleide indien de functie f(x) voldoet aan een differentiaalvergelijking van de tweede orde.

Keren we nu weer terug naar Regula Falsi en het proces van Newton. Met behulp van differentie-rekening hebben we uit Newton Regula Falsi gekregen. Ditzelfde proberen wij bij het proces gedefinieerd door (11).

Zij gegeven de drie punten x_n, x_{n-1} en x_{n-2}, daarbij berekend $f(x)$. We leggen een parabool door deze drie waarden en berekenen met behulp daarvan $f'(x_n)$ en $f''(x_n)$.

$$x_{n+1} = x_n - \frac{Ax_n\left[B^2 - AC\, f(x_n)\right]}{B^3} \tag{13}$$

met

$$A = (x_{n-2} - x_{n-1})(x_{n-1} - x_n)(x_n - x_{n-2})$$

$$B = (x_{n-1} - x_n)^2 f(x_{n-2}) - (x_{n-2} - x_n)^2 f(x_{n-1}) + (x_{n-2} - x_{n-1})(x_{n-3} - 2x_{n-2} + x_{n-1}) f(x_n)$$

$$C = (x_n - x_{n-1})f(x_{n-2}) - (x_{n-2} - x_n)f(x_{n-1}) + (x_{n-2} - x_{n-1}) f(x_n).$$

Substitueert men (4) teneinde de orde van het proces te bepalen dan vindt men

$$x_{n+1} = -\frac{a_3}{6a_1} x_n x_{n-1} x_{n-2} + \frac{1}{2}\left(\frac{a_2}{a_1}\right)^2 x_n^3 + \text{hogere termen} \tag{14}$$

en de orde α is de wortel van de vergelijking

$$\alpha^3 - \alpha^2 - \alpha - 1 = 0$$

of $\alpha \sim 1.840$.

Bij het proces (12) vindt men dezelfde orde. Het resultaat is niet veelbelovend: dit extrapolerend vindt men bij een proces steunend op k punten

$$x_{n+1} = A\, x_n\, x_{n-1} \cdots x_{n-k+1}$$

en

$$\alpha^k - \alpha^{k-1} - \alpha^{k-2} - \ldots - 1 = 0$$

zodat $\alpha < 2$ maar $\lim_{k\to\infty} \alpha = 2$.

Wil men de orde van het proces (13) verbeteren dan zou men de formule zodanig moeten wijzigen dat in (14) de term $x_n\, x_{n-1}\, x_{n-2}$ verdwijnt. We zouden dan bij $x_n\, x_{n-1}^2$ als laagste term $\alpha = 2$ vinden, bij $x_n^2\, x_{n-2}$ als laatste geldt $\alpha^3 - 2\alpha^2 - 1 = 0$ of $\alpha \sim 2.2056$.

Tenslotte nog iets over het δ^2 proces van Aitken.

Uit de oorspronkelijke reeks geitereerde waarden x_n vindt men de nieuwe reeks u_n

$$u_n = \frac{x_n\, x_{n-2} - x_{n-1}^2}{x_n - 2x_{n-1} + x_{n-2}}$$

of

$$u_n = x_{n-2} - (\Delta x_{n-2})^2/\Delta^2 x_{n-2}$$

of

$$u_n = x_n - (\Delta x_n)^2/\Delta^2 x_n \qquad \text{(Steffensen)}.$$

Is de orde van het oude proces gelijk aan 1, dan is de orde van de nieuwe reeks 2. Is de orde van de oude reeks gelijk aan $r \neq 1$, dan is die der nieuwe $2r - 1$.

Er volgt dus uit dat u_n slechts in het geval $r = 1$ een betere benadering is dan x_n.

Maar van een lineair proces als beschreven door F.B. Hildebrand maakt het δ^2 proces een kwadratisch.

En dat betekent dat het nieuw verworven proces een convergentiegebied heeft:

Lin's methode $x_0 = 1.3 \rightarrow x_1 = 1.45$ en $x_2 = 0.91$

$\delta^2 \rightarrow 1.33$ $\quad x_0 = 1.33 \rightarrow x_1 = 1.3006$ en $x_2 = 1.4460$

$\delta^2 \rightarrow 1.3251$; werkelijke wortel 1.3247180.

Er kan nog opgemerkt, dat men algemener ontwikkelingen dan (4) kan toelaten; dat als $r > 1$ men iteratieprocessen van willekeurig hoge orde kan bereiken, maar dat men daarmede niet dichter bij het doel komt.

Beschouwt men het proces

$$x_{n+1} = x_n + c\ f(x_n),$$

waarbij voor $f(x_n)$ de ontwikkeling (4) geldt, dan krijgt men door toepassing van Aitken een kwadratisch proces, behalve in de gevallen $c = -1/a_1$ en $c = -2/a_1$.

In het eerste geval $c = 1/a_1$ is het proces zelf kwadratisch en Aitken maakt het van de derde orde, in het tweede is het proces zelf lineair en Aitken geeft weer een derde orde.

ENIGE BESCHOUWINGEN OVER ITERATIEVE PROCESSEN EN NIET LINEAIRE TRANSFORMATIES (II)

J. Berghuis

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 25 mei 1957 te Amsterdam.

Meer dan een variabele

Onder $\bar{x}$ versta ik een vector met componenten x^i, $i = 1(1)n$. Zij de op te lossen vergelijkingen

$$\bar{f}(\bar{x}) = 0 \text{ of } f^i(x^1, \ldots, x^n) = 0 \text{ voor } i = 1(1)n.$$

We veronderstellen dat er een dergelijke oplossing is.

De iteratieve methoden worden geschreven in een van de twee vormen

I
$$\left\{\begin{array}{l} \bar{x}(m+1) = \bar{x}(m) + C(m) \cdot \bar{f}(\bar{x}(m)) \\ \text{of} \\ x^i(m+1) = x^i(m) + c^i_k(m) \cdot f^k(x^1(m), \ldots, x^n(m)) \quad i = 1(1)n \end{array}\right.$$

II
$$\left\{\begin{array}{l} \bar{x}(m+1) = \bar{x}(m) + C(m) \cdot \bar{f}(\bar{x}) \\ \text{met} \\ x^i(m+1) = x^i(m) + c^i_k(m)\, f^k(x^1(m+1), \ldots, x^{i-1}(m+1), \\ \qquad\qquad x^{i+1}(m), \ldots, x^n(m)) \quad i = 1(1)n. \end{array}\right.$$

Indien onder- en bovenindex dezelfde zijn, dient men hierover te sommeren.

De methoden onder I vallende heten "iterations by total steps" die onder II "iterations by single steps".

Er zij opgemerkt dat de matrix $C(m) = ((C^i_k(m)))$ de rang n moet hebben, anders behoeft men de oplossing niet te vinden. Deze eis komt overeen met de (natuurlijke) eis $c_n \neq 0$ in het enkelvoudige geval.

Bij een veelgebruikte vorm van iteratie bevat de matrix C(m) alleen maar van nul verschillende elementen op de hoofddiagonaal; daarbij moet wel op de volgorde gelet worden van de vergelijkingen f^x.
Bij deze vorm van iteratie is wel door deze van nul verschillende elementen kleiner dan $1/\text{Max} \left|\frac{\partial f^k}{\partial x^l}\right|$ te kiezen te zorgen dat het proces altijd convergent is. De monotonie blijft echter niet gehandhaafd.

Men kan nu weer veronderstellingen maken over de functies $f^i(\bar{x})$ in de omgeving van de oplossing. Zonder de algemeenheid te schaden mag men weer $\bar{x} = \bar{0}$ als oplossing aannemen. Een mogelijke veronderstelling is dan

$$f^i(\bar{x}) = f^i_k x^k + \frac{1}{2} f^i_{kl} \cdot x^k x^l + O(\bar{x}^3) \qquad (3)$$

waarbij de ordeterm aangeeft dat de fout kleiner is dan een constante onafhankelijk van $\bar{x}$ vermenigvuldigd met een homogeen polynoom van de graad 3 in de variabelen

$$x^i \ (i = 1, 2, \ldots, n).$$

Een iteratief proces is van de orde α indien

$$\bar{x}(m + 1) = O(\bar{x}^{\alpha}(m)). \qquad (4)$$

Uit I en (3) volgt

$$x^i(m+1) = x^i(m) + c^i_h f^h_k \cdot x^k(m) + \frac{1}{2} c^i_h f^h_{kl} x^k(m) \cdot x^l(m) + O(\bar{x}^3(m)).$$

Een proces dat in het algemeen lineair is. Het is kwadratisch, indien

$$1 + c^i_h \cdot f^h_k = 0$$

of

$$c^i_h \cdot f^h_k = - \delta^i_k$$

dus als $C(m) = - ((F'))^{-1}$.

Dit laatste proces is beschreven door Milne; als eis geldt dat $|F'| \neq 0$. Het vereist echter de kennis van de coëfficiënten f^h_k, welke onbekend zijn.

Zij kunnen echter evenals bij het proces van Newton benaderd worden.

Het analogon van Regula Falsi is ook hier te geven, de differentiaalquotiënten moeten dan vervangen worden door differentiequotiënten. Het aantal malen dat de functie berekend dient te worden is echter groot bij "total step"-iteratie. Bij de "single step" procedure is een aardige variant, waarvan de orde nog onbekend is.

Frequentie

Laten wij eens het geval van een iteratief proces van eerste orde van een vergelijking met een onbekende beschouwen; de fout x_{n+1} voldoet dan aan de relatie

$$x_{n+1} = a\, x_n + \ldots . \qquad (1)$$

Al sinds 1870 kent men het begrip geitereerde functies: Passen wij de operator (1) n-maal op x_0 toe, dan krijgen wij

$$x_n = a^n x_0 + \ldots$$

waarin x_0 als een soort amplitude kan worden opgevat en a als een frequentie. De andere termen voorkomende in de operator (1) verstoren dit eenvoudige beeld wel, maar wellicht is iets te bereiken met het toevoegen van andere frequenties. Iets dergelijks is ons trouwens bekend uit de theorie van het iteratieproces ter bepaling van eigenwaarden en eigenvectoren bij een symmetrische matrix: Is deze matrix van de orde K, dan krijgen wij iets als

$$x_n = B + \sum_{k=1}^{K} A_k q_k^n . \qquad (A)$$

Bij het itereren van de oplossing van meerdere vergelijkingen voldoen de geitereerde waarden in het algemeen niet aan de voorwaarde (A), omdat wij in feite te maken hebben met matrixvermenigvuldigingen van de foutvector.

Met uitzondering van bepaalde nog nader aan te duiden rij, voldoet elke rij x_n aan voorwaarde A op een bepaald moment.

Men kan n.l. bij x_r voor r = ((n - K) (1) (n + K))in het algemeen 2K + 1 getallen $B_{K,n}$, $A_{k,n}$, $q_{k,n}$, k = 1(1)K aangeven zodanig dat

$$x_r = B_{K,n} + \sum_{k=1}^{K} A_{k,n}\, q_{k,n}^{r},\quad r = ((n - K)\ (1)\ (n + K)). \tag{2}$$

Het getal $B_{K,n}$ kunnen wij de momentane limiet(basis) noemen van de K^e orde.

Gaan wij even terug naar rijen, welke voldoen aan voorwaarde (A): zij kunnen voldoen aan de voorwaarde $|q_k| < 1$ en dus convergent zijn en een limietwaarde hebben n.l. B of zij kunnen divergent zijn en een antilimiet n.l. B hebben. Bovendien kan de convergentie of divergentie nog oscillerend zijn.

Laat nu Δx_n de gewone voorwaartse differentie van x_n zijn, dan voeren wij in:

$$B_{K,n} = \frac{\begin{vmatrix} x_{n-K} & \cdots & x_n \\ \Delta x_{n-K} & \cdots & \Delta x_n \\ \Delta x_{n-K+1} & \cdots & \Delta x_{n+1} \\ \hline \Delta x_{n-1} & \cdots & \Delta x_{n+K-1} \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ \Delta x_{n-K} & \cdots & \Delta x_n \\ \hline \Delta x_{n-1} & \cdots & \Delta x_{n+K-1} \end{vmatrix}} \tag{3}$$

met als voorwaarden: $B_{0,n} = x_n$

als de noemer gelijk nul is en de teller niet: $B_{K,n} = \infty$

als de noemer gelijk nul is en de teller ook : $B_{K,n} = B_{K-1,n}$.

Opgemerkt dient te worden dat

$$B_{1,n} = \frac{x_{n+1}x_{n-1} - x_n^2}{x_{n+1} + x_{n-1} - 2x_n}\ . \tag{4}$$

Geschreven in de vorm $B_{K,n} = e_K(x_n)$, zien wij dat de operator e_K geen lineaire is, alhoewel men direct door ontwikkeling van de determinanten naar hun eerste rijen ziet uit (3) dat

$$e_K(x_n) = \frac{c_{n-K}x_{n-K} + \ldots + c_n x_r}{c_{n-K} + \ldots + c_n} \tag{5}$$

waarin c_i de bijbehorende minoren voorstellen.

Als C een constante is gelden wel de volgende regels

$$e_K(Cx_n) = Ce_K(x_n)$$

en

$$e_K(x_n + C) = e_K(x_n) + e_K(C) = e_K(x_n) + C.$$

Door optelling van de eerste, tweede tot en met r + eerste rij bij de eerste rij in de determinant van de teller van (3) vindt men

$$e_K(x_n) = \frac{c_{n-K}x_{n-K+r} + \ldots + c_n x_{n+r}}{c_{n-K} + \ldots + c_n}.$$

Tot nu toe hebben wij stilzwijgend de $B_{K,n}$ gebruikt in de formule (2) en (3); wij gaan bewijzen dat dit toegestaan is.

Stel eens

$$\varepsilon_n = \sum_{k=1}^{K} A_k q_k^n \quad (n = 0(1)(2K - 1)).$$

Nu zijn de q_k's de wortels van de vergelijking

$$\begin{vmatrix} 1 & q & \ldots & q^K \\ \varepsilon_0 & \varepsilon_1 & \ldots & \varepsilon_K \\ \ldots & \ldots & \ldots & \ldots \\ \varepsilon_{K-1} & \varepsilon_K & \ldots & \varepsilon_{2K-1} \end{vmatrix} = 0.$$

Substitueren wij nu voor ε de waarde Δx_{n-K+r} en voor A_k de waarde $A_{k,n}\left[q_{k,n}^{n-K+1} - q_{k,n}^{n-K}\right]$ dan zijn de $q_{k,n}$ de wortels van

$$\begin{vmatrix} 1 & q & \dots & q^K \\ \Delta x_{n-K} & \Delta x_{n-K+1} & \dots & \Delta x_n \\ \dots & \dots & \dots & \dots \\ \Delta x_{n-1} & \dots & & \Delta x_{n+K-1} \end{vmatrix} = 0$$

of in de notatie van (5)

$$c_{n-K} + \dots + c_n q^K = 0. \tag{6}$$

Substitueert men (2) in (5) en maakt men gebruik van bovenstaande vergelijking dan krijgt men

$$e_K(x_n) = B_{K,n}, \tag{7}$$

indien althans niet aan de vergelijking

$$c_{n-K} + c_{n-K+1} + \dots + c_n = 0$$

voldaan is. In dit laatste geval voldoet q = 1 aan de vergelijking (6) en dan is de propositie (2) niet mogelijk.

Met behulp van de operator e_K zijn weer nieuwe operatoren te definieren, b.v. door herhaald toepassen van e_K, uit de verkregen waarden een bepaalde rij te nemen en daarop weer een e_K toe te passen. Voor hun eigenschappen nader bestudeerd worden geven wij eens een voorbeeld.
Zie Shanks „Journal Math. & Phys. XXXIV no. 1, April, 1955.

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \dots$$

n	x_n	e_1	$(e_1)^2$	$(e_1)^3$	$(e_1)^4$
0	4.0000000				
1	2.6666667	3.1666667			
2	3.4666667	3.1333333	3.1421053		
3	2.8952381	3.1452381	3.1414502	3.1415993	
4	3.3396825	3.1396825	3.1416433	3.1415909	3.1415928
5	2.9760462	3.1427129	3.1415713	3.1415933	3.1415927

n	x_n	e_1	$(e_1)^2$	$(e_1)^3$
6	3.2837385	3.1408814	3.1416029	3.1415925
7	3.0170718	3.1420718	3.1415873	
8	3.2523659	3.1412548		
9	3.0418396			

Nemen wij hetzelfde voorbeeld en passen hierop de lineaire middelings-operator toe:

n	A_n	1	2	3	4	5	6	7	8	9
0	4.0000000									
		3.3333333								
1	2.6666667		3.2							
		3.0666667		3.1619048						
2	3.4666667		3.1238096		3.1492064					
		3.1809524		3.1365080		3.1445888				
3	2.8952381		3.1492064		3.1399712		3.1428128			
		3.1174603		3.1434344		3.1410368		3.1421024		
4	3.3396825		3.1376624		3.1421024		3.1413920		3.1418099	
		3.1578644		3.1407704		3.1417472		3.1415174		3.1416868
5	2.9760462		3.1438784		3.1413920		3.1416428		3.1415636	
		3.1298924		3.1420136		3.1415383		3.1416098		
6	3.2837385		3.1401488		3.1416845		3.1415768			
		3.1504052		3.1413554		3.1416152				
7	3.0170718		3.1425620		3.1415459					
		3.1347188		3.1417364						
8	3.2523659		3.1409108							
		3.1471028								
9	3.0418396									

Bij het bestuderen der eigenschappen van de niet lineaire operatoren $B_{k,n}$ zij allereerst opgemerkt dat zij niet regulier zijn:

Wij noemen een operator T regulier indien als de rij x_n convergeert ook $T(x_n)$ convergeert en tevens

$$\lim_{n\to\infty} T(x_n) = \lim_{n\to\infty} x_n .$$

De operator e_1 is niet regulier

$$3 = 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \ldots .$$

De oneven $B_{1,n}$ zijn alle oneindig, de even $B_{1,n}$ alle gelijk aan 3 dus $\lim_{n\to\infty} B_{1,n}$ bestaat niet.

Wel geldt de stelling van S. Lubkin:

Als x_n convergeert en $e_1(x_n)$ convergeert dan

$$\lim_{n\to\infty} e_1(x_n) = \lim_{n\to\infty} x_n .$$

Bewijs: Stel $e_1(x_\infty) \neq x_\infty$. Nu gelden

$$e_1(x_n) = x_{n-1} + \frac{\Delta x_{n-1}}{1 - R_{n+1}} = x_n + \frac{\Delta x_n}{1 - R_n} \qquad (8)$$

met

$$R_n = \frac{\Delta x_{n-1}}{\Delta x_n} \quad \text{en} \quad n > 0.$$

Dus $\dfrac{\Delta x_n}{1 - R_n} = e_1(x_n) - x_{n-1} \to e_1(x_\infty) - x_\infty \neq 0.$

Wegens x_n convergent volgt $\Delta x_n \to 0$ dus $1 - R_n \to 0$ of $R_n \to 1$. Er bestaat dus een N zodanig dat voor $n > N$ alle Δx_n hetzelfde teken hebben. Verder houdt ook $1 - R_n$ hetzelfde teken voor voldoend grote n wegens $e_1(x_\infty) - x_\infty \neq 0$.

$R_n > 1$ is niet mogelijk wegens de convergentie van x_n. Als $R_n < 1$ dan bestaat of Δx_n of $-\Delta x_n$ vanaf $n > N$ uit positieve afnemende termen en convergentie vereist $n \cdot \Delta x_n \to 0$. Maar

$$\frac{n\Delta x_n}{n(1 - R_n)} \to e_1(x_\infty) - x_\infty \neq 0$$

dus ook $n \cdot (1 - R_n) \to 0$.

Volgens een criterium uit K. Knopp, Theorie und Anwendung der Unendliche Reihen, p. 286, no. 170 kan dan x_n niet convergeren.

Er volgt dus $e_1(x_\infty) = x_\infty$.

Enige voorbeelden:

I $\begin{cases} x_n = 1, 0, 1, 0, 1, 0, \text{enz.} \\ e_1(x_n) = \frac{1}{2} \end{cases}$

II $\begin{cases} x_n = 1, 1\text{-}2, 1\text{-}2\text{+}4, \text{enz. of } \Delta x_n = 1, -2, 4, -8, 16, -32, \text{enz.} \\ e_1(x_n) = \frac{1}{3} \end{cases}$

III $\begin{cases} \Delta x_n = 1, 2, 4, 8, 16 \\ e_1(x_n) = -1 \end{cases}$

IV $\begin{cases} \Delta x_n = 1, -\frac{1}{2}, +\frac{1}{3}, \ldots \frac{(-1)^{n+1}}{n} \\ \Delta e_1(x_n) = \frac{(-1)^n}{n(4x^2 - 1)} \quad \text{eerste term } \frac{2}{3} \end{cases}$

D. Shanks, Theorema I. Zij $f(m) = \sum_{i=0}^{M_1} f_i m^i$ en $g(m) = \sum_{i=0}^{M_2} g_i m^i$ met $f_{M_1} \neq 0$, $g_{M_2} \neq 0$ en $g(m) \neq 0$ $(m = 0, 1, 2, \ldots)$ en zij

$$A_n = \sum_{m=0}^{n} (-1)^m f(m)/g(m)$$

dan geldt:

a: als $M_2 > M_1$, A_n convergeert en elke afgeleide rij $B_n = e_1(A)$, $C_n = e_1(B_n)$ convergeert sneller dan zijn voorganger naar dezelfde limiet.

b: als $M_1 > M_2$, A_n divergeert en elke afgeleide rij divergeert minder sterk dan zijn voorganger tot $e_1^M(A_n)$ met $M = \frac{1}{2}\left[(M_1 - M_2)\right]$. De rij $e_1^{M+1}(A_n)$ convergeert en elke verdeelde rij convergeert sneller naar dezelfde limiet.

Bewijs: Uit (8) volgt

$$\Delta B_n = \Delta A_n \frac{\Delta A_{n+1}\Delta A_{n-1} - (\Delta A_n)^2}{(\Delta A_n - \Delta A_{n+1})(\Delta A_{n-1} - \Delta A_n)} = \tag{10}$$

$$= \Delta A_n \frac{f^2(n+1)g(n)g(n+2) - g^2(n+1)f(n)f(n+2)}{\left[f(n)g(n+1) - f(n+1)g(n)\right]\left[f(n+1)g(n+2) - f(n+2)g(n+1)\right]} .$$

Voor voldoend grote waarden van n geldt dan

$$\Delta B_n = \Delta A_n \left\{\frac{M_1 - M_2}{4n^2} + O(\frac{1}{n^3})\right\} \tag{9}$$

en wegens

$$\frac{\Delta A_n}{\Delta A_{n-1}} = - \frac{f(n+1)g(n)}{g(n+1)f(n)}$$

is $\frac{\Delta A_n}{\Delta A_{n-1}}$ negatief.

De reeks $A_0 + \sum_{n=0}^{\infty} \Delta A_n$ is alternerend en indien a), geldt $\Delta A_n \to 0$ dus convergeert de reeks. Wegens (9) convergeert B_n ook en zelfs convergeert B_n sneller. Uit (10) volgt dat geschreven mag worden $\Delta B_n = (-1)^n \frac{f'(n)}{g'(n)}$, waarbij de graad van f' gelijk aan M_1' en die van g' gelijk aan M_2' wordt gesteld. Wegens (9) geldt echter

$$M_2' - M_1' = M_2 - M_1 + 2 > 0$$

en dus is a) bewezen.

Het bewijs van b) is nu duidelijk.

S. Lubkin is verder gegaan dan deze stelling: Hij bewees de stelling voor het geval

$$\frac{\Delta A_n}{\Delta A_{n-1}} = \alpha_0 + \frac{\alpha_1}{n} + \frac{\alpha_2}{n^2} + \ldots$$

met $|\alpha_0| < 1$ of $\alpha_0 = -1$.

Zij x_n de rij voortgebracht door de kettingbreuk

$$1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + 1}}}$$

n	x_n	$e_1(x_n)$	$e_1^2(x_n)$
1	1		
2	3/2	17/12	
3	7/5	99/70	19601/13860
4	17/12	577/408	
5	41/29		

N.B. 19601/13860 = 1.414213564.

Het blijkt dat e_1^m toegepast op x_n identiek is met het itereren van $\sqrt{2}$ met behulp van Newton.

Schrijft men $x_n = \frac{T_n}{N_n}$ dan blijkt uit de recursievergelijkingen

$$T_n = 2T_{n-1} + T_{n-2} \ , \quad N_n = 2N_{n-1} + N_{n-2}$$

$$T_1 = N_1 = 1 \ , \quad T_2 = 3 \ , \quad N_2 = 2$$

dat $$x_n = \sqrt{2}\left[\frac{1 + (2\sqrt{2} - 3)^n}{1 - (2\sqrt{2} - 3)^n}\right],$$

maar uit

$$x_n = \sqrt{2}\,\frac{[1 + x^n]}{[1 - x^n]}$$

volgt $e_1(x_n) = \sqrt{2}\,\frac{1 + (x^2)n}{1 - (x^2)n} = x_{2n}$

en eveneens $\frac{1}{2}\,(x_n + \frac{2}{x_n}) = x_{2n} = e_1(x_n)$

zodat bij herhaald toepassen het gevraagde volgt.
Zij x_n de partiële som der reeks

$$c + cx + cx^2 + cx^3 + \ldots \qquad (11)$$

dus

$$x_n = \frac{c}{1 - x} - \frac{cx^n}{1 - x}$$

dan is dus $e_1(x_n) = \frac{c}{1 - x}$, x_n bevat n.l. één frequentie.
Dit antwoord is onafhankelijk van de grootte van x, zelfs mag de reeks (11) divergent zijn. In het geval x = 1 krijgt men $e_1(x_n) = \infty$ maar dit is ook het goede antwoord.

Als volgende voorbeeld beschouwen wij

$$f(z) = \frac{2}{(1 - z)(2 - z)} = 1 + \frac{3}{2}\,z + \frac{7}{4}\,z^2 + \frac{15}{8}\,z^3 + \frac{31}{16}\,z^4 + \ldots .$$

De partiële sommen der laatste reeks zijn te schrijven in de vorm:

$$x_n = f(z) - \frac{2z}{1 - z}\,z^n + \frac{z}{z - 2}\,(\frac{z}{2})^n \qquad (12)$$

zoals direct blijkt door f(z) te splitsen in partiële breuken. Uit (12) volgt nu dat x_n twee frequenties bevat dus

$$e_2(x_n) = f(z),$$

behalve de waarden z = 1 of z = 2.
De vraag rijst wat er gebeurt indien wij e_1^m op de rij (12) toepassen.

Bij het bewezen verband tussen e_1 toegepast op de afgebroken kettingbreuk van $\sqrt{2}$ en het proces van Newton zij nog opgemerkt dat e_1 slechts uit de rationale getallen der kettingbreuk weer rationale getallen levert. Aangezien $\sqrt{2}$ irrationaal is volgt daaruit dat een eindig aantal malen e_1

toepassen slechts een rationale benadering voor $\sqrt{2}$ levert.

Vroeger hebben wij al gezien dat toepassing van e_1 op een kwadratisch convergerende rij geen verbetering meer levert, zodat we e_1 niet meer kunnen toepassen op A_1, B_2, C_3 enz.
Hoe het resultaat is wanneer men e_2, e_3 enz. toepast op een kwadratisch convergerende reeks is niet bekend.

Toepassing van e_K met $K > 1$ levert ook slechts rationale getallen, waaruit weer volgt dat de rij der benaderende kettingbreuken tenminste ∞ veel frequenties bevat.

In het nu volgende gaan wij e_1^m toepassen op de partiële sommen der reeksontwikkeling van de functie

$$f(z) = 1/(z - z_0)(z - z_1)$$

met

$$0 < |z_0| < |z_1|.$$

Door $f(z)$ als verschil van twee breuken $C_1(z - z_0)^{-1}$ en $C_2(z - z_1)^{-1}$ te schrijven vindt men voor de te behandelen rij partiaalsommen

$$x_n = f(z) + f(z)\,\frac{z - z_1}{z_1 - z_0}\,\left(\frac{z}{z_0}\right)^{n+1} - f(z)\,\frac{z - z_0}{z_1 - z_0}\,\left(\frac{z}{z_1}\right)^{n+1} \qquad (13)$$

waaruit blijkt dat x_n precies twee frequenties bevat. Toepassing van $B_{2,n}$ geeft dus direct het goede resultaat.

Maar nu e_1:
We gebruiken het Theorema IV van D. Shanks (loc. cit.). Uit (13) kan door middel van rekenen worden afgeleid dat

$$B_n = e_1(x_n) = f(z) - \left(\frac{z}{z_1}\right)^{n+2} \frac{f(z)(z_1 - z_0)}{(z-z_0) - (z-z_1)(z_0/z_1)^{n+2}}\,. \qquad (14)$$

D.w.z. dat B_n convergeert binnen de cirkel $|z| = |z_1|$, terwijl x_n convergeert binnen de cirkel $|z| = |z_0|$. Verder is de grootste frequentie $\left(\frac{z}{z_0}\right)$ geëlimineerd; hiervoor zijn er oneindig veel in de plaats gekomen, zoals men ziet door de noemer in (14) te ontwikkelen

$$B_n = f(z) - f(z)\,\frac{z_1 - z_0}{z - z_0}\sum_{i=0}^{\infty}\left[\frac{z-z_1}{z-z_0}\right]^i\left[\frac{z}{z_1}\,(\frac{z_0}{z_1})^i\right]^{n+2}. \qquad (15)$$

Door de operator e_1 op (14) toe te passen krijgen we

$$C_n = f(z) - \frac{f(z)(z - z_1)(z_1 - z_0)^3 z_0^{-1} z_1^{-1}(zz_0/z_1^2)^{n+3}}{(z - z_0)^2(z - z_1)^2 z_0 z_1^{-3} + \sum_{i=1}^{3} a_i (z_0/z_1)^{in}} \qquad (16)$$

met a_i bepaalde coëfficiënten afhankelijk van z_0, z_1.

Volgens gegeven is $\left|\frac{z_0}{z_1}\right| < 1$ en C_n is dus convergent indien

$$\left|\frac{zz_0}{z_1^2}\right| < 1.$$

En dus weer is het convergentiegebied vergroot tot $\left|\frac{z_1^2}{z_0}\right|$.

Maar als $z = \frac{z_1^2}{z_0}$ dan wordt de frequentie $\frac{z_0 z}{z_1^2} = 1$ en C_n convergeert naar

$$f(z)\left\{1 - \frac{z_0 z_1}{(z_0 + z_1)^2}\right\} \qquad (17)$$

een foutief antwoord.

Voor verdere toepassing van e_1 op C_n in het punt $z = z_1^2/z_0$ kan men C_n opvatten als een limiet (17) met frequenties en dat betekent dat bij verdere toepassing van e_1 men steeds het foutieve antwoord krijgt.

Verder krijgt men een foutief antwoord in de punten $z = (\frac{z_1^2}{z_0})^r \cdot z_0$, want deze leveren bij r maal toepassen van e_1 een frequentie gelijk aan 1.

Indien wij uitgaan van meer dan twee frequenties, dan blijft gelden dat de grootste frequentie geëlimineerd wordt door toepassing van e_1; er worden echter oneindig veel andere ingevoerd. Het convergentiegebied wordt wel vergroot door eliminatie van de grootste frequentie, maar het

$$\begin{array}{ccccc} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} & B_{0,4} \\ & B_{1,1} & B_{1,2} & B_{1,3} & B_{1,4} \\ & & B_{2,2} & B_{2,3} & B_{2,4} \\ & & & B_{3,3} & B_{3,4} \end{array}$$

Voeren wij verder in $B_{k,n} = e_k(x_n)$

$$C_{k,n} = e_k(B_{k,n})$$

$$D_{k,n} = e_k(C_{k,n})$$

dan wordt $\tilde{e}_k$ gedefiniëerd door

$$\tilde{e}_k(x_n) = x_0;\ B_{k,k};\ C_{k,2k};\ D_{k,3k} \quad \text{enz.}$$

Het is begrijpelijk, dat men ook kan toepassen $e_d^2 = e_d \cdot e_d$ of $\tilde{e}_k \cdot e_d$ of $\tilde{e}_k^2$ enz. De operator e_1 blijkt voor de sommatie van een divergente reeks zeer plezierig: D. Shanks geeft de volgende voorbeelden

$$\ln 3 = 0 + 2 - \frac{1}{2}2^2 + \frac{1}{3}2^3 - \frac{1}{4}2^4 + \ldots$$

$$\int_0^\infty \frac{e^{-t}}{1+t}\,dt = 0{,}596347 = 0! - 1! + 2! - 3! + 4! - 5!$$

$$C = \frac{1}{2} + \frac{1}{2}B_2 + \frac{1}{4}B_4 + \frac{1}{6}B_6$$

$$G = \frac{\pi}{2}(3 \ln 3 - 5 \ln 5 + 7 \ln 7 \ldots)$$

$$f(x) = \frac{x^2}{2!} - \frac{x^5}{5!} + \frac{11x^8}{8!} - \frac{375x^{11}}{11!} + \ldots$$

de laatste voor $x > 3.12735$.

Direct kan weer opgemerkt worden dat $\tilde{e}_k$ en e_d slechts rationale getallen leveren wanneer de input x_n rationaal is.

Kiezen wij dus de reeks voor de logarithme

blijft zeer de vraag of herhaalde toepassing van e_1 het resultaat aanzienlijk zal verbeteren.

Een ieder is bekend met het itereren van de grootste eigenwaarde van symmetrische matrices en weet ook dat toepassing van het δ^2 proces vaak eer een verslechtering dan een verbetering geeft. Het zij opgemerkt dat de rij geïtereerde waarden in dit geval convergent is en dat herhaald toepassen van e_1 geen foutief antwoord kan geven. Versnelt nl. e_1 het iteratieproces, dus bestaat $e_1(x_\infty)$ dan is $e_1(x_\infty) = x_\infty$ volgens Lubkin. Maar het δ^2 proces kan minder snel convergeren, zoals al gebleken is in het geval dat het iteratieve proces niet van de eerste orde is. Lubkin heeft criteria aangegeven, waarvoor toepassing van e_1 minder snel convergente rijen geeft: o.a. het geval $\frac{\Delta x_n}{\Delta x_{n+1}} \to 1$ is zeer gevaarlijk. In het laatste geval dient e_1 gewijzigd te worden.

Men kan gemakkelijk nagaan dat toepassing van e_n bij itereren van eigenwaarden van een matrix van de orde n direct het antwoord geeft; dat bij het gelijk zijn van twee eigenwaarden e_n hetzelfde antwoord geeft als langdurige iteratie op de gewone wijze. In dit verband is het van belang het gedrag van bijv. e_2 en e_3 te onderzoeken op rijen, die meer dan drie frequenties bevatten. Waarschijnlijk worden de grootste twee of drie frequenties geëlimineerd en zouden deze operatoren van groot belang bij matrixiteraties zijn.

Andere operatoren

Wij hebben gedefiniëerd de waarden $B_{k,n}$.
Allereerst zij gedefiniëerd de diagonale transformatie,

$$e_d(x_n) = B_{n,n}.$$

Deze is nl. de diagonaal van de getallen

$$\log(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} \ldots$$

en stellen wij $x = -2$, dan kunnen wij nooit de goede waarde $\log(-1) = \pi i$ of $-\pi i$ vinden (naar gelang het Riemannse oppervlak waarop wij ons bevinden). De operator e_d blijkt goede waarden te leveren mits x zich niet bevindt op de snede lopende langs de negatief reële as. Men noemt dit convergentie behalve in de schaduw van de vertakkingspunten (d.w.z. de willekeurig aan te brengen snede).

MULTI-LENGTE PROGRAMMERING

E.W. Dijkstra

Lezing in het colloquium Moderne Rekenmachines van het Mathematisch Centrum op 29 juni 1957 te Amsterdam.

De wijze, waarop multi-lengte rekenwerk door een automatische rekenmachine het meest efficiënt verricht kan worden, zal in hoge mate beïnvloed worden door de specifieke eigenschappen van de betroffen machine.

Zonder te streven naar volledigheid laten wij ter inleiding enkele mogelijkheden volgen.

Wij zullen ons beperken tot machines met vaste komma. (Bij een machine met ingebouwde drijvende komma zal men van de drijvende komma waarschijnlijk geen enkel voordeel hebben, mogelijkerwijs zelfs nadeel, b.v. als er in de code niet voorzien is in een soepele methode ter isolatie van het minst significante gedeelte van een product.)

Het multi-lengte getal wordt opgesplitst in "blokken"; voor elk blok wordt een woord van de standaardlengte ter beschikking gesteld.

Bij een binaire machine rijst allereerst de vraag, of de eenheid van blok 10^n of 2^m maal zo significant moet zijn als de eenheid van het volgende blok.

Om het lange getal in decimale blokken te splitsen, biedt eigenlijk alleen voordeel bij de invoer en de uitvoer; bij administratieve processen via ponskaarten valt het te overwegen, mits de deling door de machine voldoende snel kan worden uitgevoerd. Dit laatste is minder urgent, als het administratieve proces hoofdzakelijk uit additieve bewerkingen bestaat; zij worden vergemakkelijkt als de woordcapaciteit meer dan twee maal zo groot is dan de betroffen tienmacht, die met de bloklengte overeenkomt.

Voor de ARMAC, zonder ingebouwde deling, en voor wetenschappelijke berekeningen, was het opsplitsen in decimale blokken kennelijk niet het overwegen waard.

Van vrij veel invloed is de in de machine gangbare representatie van negatieve getallen (annex de werking van het arithmetisch orgaan).

1. Het "complementen-systeem"

Is de woordlengte m+1 bits, (tekencijfer, gevolgd door m binaire cijfers) en denken wij de komma onmiddellijk achter het tekencijfer, dan zijn in dit systeem de capaciteitsgrenzen

$$-1 \leq x \leq 1 - 2^{-m}.$$

0 is hier "het kleinste positieve getal". Een getal wordt van teken gewisseld, door alle cijfers te inverteren (0 door 1 vervangen en omgekeerd) en bovendien 2^{-m} (een 1 op de minst significante plaats) op te tellen. Tekenwisseling geschiedt dan wel m.b.v. de opteller; additieve bewerkingen worden zonder end around carry uitgevoerd.

(Een andere versie van het complementensysteem zou zijn met de capaciteitsgrenzen

$$-(1 - 2^{-m}) \leq x \leq 1;$$

0 zou dan "het grootste negatieve getal" zijn. De voorstelling van positieve getallen wordt dan minder vanzelfsprekend; m.i. wegen de - aanvechtbare! - bezwaren van een negatieve nul ruimschoots op tegen het bezit van de +1.)

2. Het inversen-systeem

Hier zijn de capaciteitsgrenzen

$$-(1 - 2^{m}) \leq x \leq 1 - 2^{-m}.$$

Er zijn twee representaties voor het getal nul: +0 (allemaal nullen) en -0 (allemaal enen). Een getal wordt van teken gewisseld door alle cijfers te inverteren; dit kan dus geschieden zonder gebruik te maken van de op-

teller. Additieve bewerkingen moeten echter uitgevoerd worden met end around carry.

Voor de representatie van een getal van q-voudige lengte nemen we aan dat het tekencijfer van elk woord als tekencijfer van het blok fungeert, dat dus met een woordlengte van m+1 bits de opeenvolgende blokken een factor 2^m in significantie verschillen.
Als we de blokken als geheel getal opgevat met B_i aanduiden, is het multi-lengte getal gelijk aan

$$B_1 \cdot 2^{-m} + B_2 \cdot 2^{-2m} + \ldots + B_q \cdot 2^{-qm}.$$

In het complementensysteem nu is het aantrekkelijkste om het meest-significante blok het teken van het getal mee te geven, en alle volgende blokken het +-teken (resp. het teken van de nul) te laten hebben. (Analoog aan de voorstelling: log 0.2 = - 1 + .30103.)

Dat dit voordelen heeft wordt het beste geïllustreerd met het feit, dat met deze conventies de capaciteitsrestricties (voor positieve nul) luiden:

$$-1 \leq x \leq 1 - 2^{-mq}.$$

In het inversensysteem verdient het de voorkeur, om elk blok het teken van het lange getal te geven. Dan luiden n.l. de capaciteitsrestricties analoog aan het enkele woord

$$-(1 - 2^{-mq}) \leq x \leq 1 - 2^{-mq}.$$

Met betrekking tot de additieve bewerkingen valt - ongeacht of de machine uitgerust is met een echte dubbel-lengte accumulator - de vergelijking tussen beide systemen uit ten gunste van het complementensysteem. Weinig verschil maakt het bij additie van twee getallen van hetzelfde teken: de optelling wordt uitgevoerd te beginnen bij het minst significante blok, de overdracht, die ontstaat wordt bij de optelling der volgende blokken in rekening gebracht. Bij het complementen-systeem is deze overdracht 0 of 1, bij het inversen-systeem is deze over-

dracht +1 of +0 bij additie van positieve getallen, -1 of -0 bij die van negatieve getallen. Bij het complementen-systeem gaat de additie van twee getallen van verschillend teken even gemakkelijk; bij het inversen-systeem echter is de inhoud van het minst significante blok van het antwoord afhankelijk van het uiteindelijke teken van de som, dus van iets wat pas na afloop van de optelling bekend is. De optelling van twee getallen van verschillend teken geschiedt nu in twee fasen: eerst worden de getallen bloksgewijs opgeteld (overdracht is uitgesloten!); als aan het meest significante blok $\neq 0$ het teken van het antwoord gedetecteerd is, wordt het "teken consistent" gemaakt.

De ARMAC heeft geen dubbel-lengte accumulator; evenwel is dankzij de schuifopdrachten, welker verrichtingen, zoals in een machine werkend met het inversen-systeem voor de hand ligt, volslagen symmetrisch zijn in de enen en de nullen, een simpele mogelijkheid geboden om de "getekende overdracht" te isoleren, zoals deze voorkomt bij de additie van getallen van hetzelfde teken.

Bij een vermenigvuldiging van twee teken consistente getallen kan hiervan uiteraard alle vruchten geplukt worden.

De multi-lengte routines van de ARMAC zijn bestemd voor het rekenen met breuken, en <u>niet</u> met heel grote gehele getallen. Wat gewoonlijk als deling wordt aangeduid is hier dus vanwege de veronachtzaming van de rest een quotiëntberekening. Hiervoor is een proces gekozen, waarbij multi-lengte optellingen slechts hoeven worden toegepast op getallen van hetzelfde teken. Er is gebruik gemaakt van een variant van de relatie:

$$\frac{a}{1 - c} = (1 + c)(1 + c^2)(1 + c^4)(1 + c^8) \ldots . \quad |c| < 1.$$

Evenals bij toepassing van bovenstaande formule nodig, resp. gewenst is, worden teller en noemer van teken gewisseld, als de noemer negatief is en worden ze beide met 2^n vermenigvuldigd, n zo gekozen, dat de nieuwe noemer b voldoet aan de ongelijkheden

$$\frac{1}{2} \leq b < 1.$$

Als we de zo verkregen teller en noemer met a resp. b aanduiden zou het boven gegeven proces aanleiding geven tot het rekenschema:

Inleiding: $(q_0 = a)$ en $c_0 = 1 - b$

Repetitie: $q_{i+1} = q_i + q_i c_i$ en $c_{i+1} = c_i^2$

Resultaat: "als $c_i = 0$, dan geldt $q_i = \frac{a}{b}$".

Omdat dit proces per stap twee vermenigvuldigingen van multi-lengte getallen vergt is de volgende variant gekozen:

Inleiding: $(q_0 = a$ en $b_0 = b)$

Repetitie: $c_i \approx 1 - b_i$ $q_{i+1} = q_i + q_i c_i$ en $b_{i+1} = b_i + b_i c_i$

Resultaat: "als $b_i = c$, dan geldt $q_i = \frac{a}{b}$".

Als in de repetitie exact $c_i = 1 - b_i$ gekozen wordt, zijn de processen identiek; we kiezen nu voor c_i een getal, dat slechts in één blok van nul afwijkt (en wel zó, dat $c_i \leq 1 - b_i$, m.a.w. de q_i blijft het quotiënt van onderen naderen, de c_i blijven dus positief).

De twee vermenigvuldigingen per stap zijn nu teruggebracht tot vermenigvuldigingen van een multi-lengte getal met het enkele-lengte getal c_i. Een en ander gaat - tenslotte - ten koste van de quadratische convergentie. Desondanks is het proces in deze vorm in tijd, en zeker in programmaruimte, aanmerkelijk voordeliger.

Hoewel we het niet hebben toegepast, zie ik geen enkel bezwaar om de quadraat-wortel op een analoge wijze te berekenen: nl.

Inleiding: $w_0 = b \cdot 2^{\frac{1}{2}n}$ en $b_0 = b \cdot 2^n$ zodat $\frac{1}{2} \leq b_0 \leq 1$.

(Ter versnelling van de convergentie):

Repetitie: $c_i \approx \frac{1}{2}(1 - b_i)$, $w_{i+1} = w_i(1 + c_i)$ en $b_{i+1} = b_i(1 + c_i)^2$.

Resultaat: als $b_i = 1$, dan is $w_i = b^{\frac{1}{2}}$.

De dubbel-lengte worteltrekking in de ARMAC b.v. maakt er expliciet gebruik van, dat het getal maar in dubbele lengte gegeven is: het argument wordt tussen $\frac{1}{4}$ en 1 genormeerd: dan wordt de (enkele lengte) wortel met de normale wortelsubroutine getrokken, deze wordt met één naslag m.b.v. Newton's iteratie tot tweevoudige lengte verbeterd, en over het halve aantal binaire plaatsen teruggeschreven.

Resumerend moet worden opgemerkt, dat de multi-lengte routines voor de ARMAC nog al ernstig zijn beïnvloed door de beperkingen van de machine. Het inversen-systeem is hiervan de minste. Meer sporen hebben achtergelaten de afwezigheid van de ingebouwde deling van het grotere snelle geheugen en wellicht van de "volle" dubbel-lengte accumulator.