STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

ZW 1952 - 023

Voordracht in de serie Actualiteiten

# Een of meer adrescodes in rekenmachines?

20 december1952

C.S. Scholten

Voordracht door C.S. Scholten in de serie

Actualiteiten op 20 December 1952.

# EEN OF MEER ADRESCODES IN REKENMACHINES?

De bedoeling van deze voordracht is, een beknopt overzicht te geven van de mogelijke codes en waar mogelijk, aan de hand van een voorbeeld de voor- en nadelen van de verschillende mogelijkheden tegen elkaar af te wegen. Vooropgesteld zij, dat het geenszins de bedoeling is in dezen naar volledigheid te streven, al was het alleen maar wegens het feit dat het probleem van het "gemiddelde programma" zelfs nog niet bij benadering is opgelost, zo het al ooit serieus is aangepakt.

Om de gedachten te bepalen zullen we ons voorlopig voorstellen dat we te doen hebben met een seriemachine met een of andere vorm van cyclisch geheugen (b.v. een magnetische trommel), waaruit we onze woorden eveneens in serie zullen uitlezen.

Nemen we aan, dat we in een track 1024 bits kunnen bergen en dat een volledig woord uit 32 bits bestaat, dan kunnen we dus 32 hele woorden op een track bergen. Deze zullen we gewoon achter elkaar op de omtrek neerzetten, zodat, wanneer we afzien van een gap, elk woord $\frac{1}{32}$ van de omtrek beslaat.

Het is duidelijk dat de gemiddelde aanhaaltijd voor een woord (gerekend tot en met het binnenlopen van de laatste digit) ongeveer een halve omwenteling (cyclus) bedraagt. Iets nauwkeuriger bedraagt deze aanhaaltijd als we even N woorden per track aannemen i.p.v. 32

$\tau_{1N} = \frac{1}{2} + \frac{1}{N}$ (in omwentelingen).

De tweede term verdisconteert het feit, dat de woorden een van nul verschillende fractie van de omtrek in beslag nemen.

Willen we meer woorden tegelijk aanhalen, dan wordt de aanhaaltijd langer, de aanhaaltijd per woord echter korter. Willen we b.v. twee woorden aanhalen, onafhankelijk van hun volgorde, dan is de aanhaaltijd $\tau_{2N} = \frac{2}{3} + \frac{7}{6N}$, waarbij tevens gemiddeld is over de mogelijke posities van de twee woorden op de omtrek. De aanhaaltijd per woord is dus $\frac{1}{3} + \frac{7}{12N}$.

Volledigheidshalve vermelden we nog dat onder dezelfde voorwaarden geldt

$$\tau_{mN} = \frac{m}{m+1} + \frac{3m+1}{2(m+1)N} \qquad (m \leqslant N)$$

en de aanhaaltijd per woord dus

$$\tau_{\frac{mN}{m}} = \frac{1}{m+1} + \frac{3m+1}{2m(m+1)N} .$$

We onderscheiden nu de volgende codes:

1. 1-adrescode. Bij dit systeem bevat de opdracht behalve het functiegedeelte (f) slechts een adres (a). Afgezien van enkele speciale opdrachten is de gang van zaken zo, dat de arithmetische bewerking, aangegeven in het functiegedeelte wordt uitgevoerd op het getal op het aangegeven adres en op dat getal dat zich op een vast adres, zeg adres 0, bevindt. Het resultaat van de bewerking kunnen we naar verkiezing weer op adres a of op adres 0 terugschrijven. Kiezen we de laatste mogelijkheid en bedenken we dat we adres 0 tweemaal moeten passeren, hetgeen ons minstens een omwenteling kost en dat we bovendien een mogelijkheid moeten hebben, het gevonden resultaat te bewaren tot we weer op adres 0 zijn, waar we het resultaat weer kwijt kunnen, dan ligt de bouw van een accumulator zeer voor de hand. Deze fungeert dan als snelle geheugenplaats. De opdracht f/a heeft dan betrekking op het getal op adres a en het getal in de accumulator (A), terwijl het resultaat weer in A wordt afgeleverd. (Van technische complicaties bij vermenigvuldiging en deling wordt hier afgezien).

Gezien het feit, dat er geen extra adres gebruikt wordt om de plaat van de volgende instructie aan te geven, volgen we in dit geval de conventie, dat we de opdrachten in die volgorde afhandelen, waarin ze in het geheugen zijn neergezet.

2. 1+1 -adrescode. Het tweede adres wordt hier gebruikt om aan te ge op welke plaats in het geheugen de volgende opdracht gehaald moet worden. Overigens als boven.

3. 2-adrescode. Hierbij wordt op beide aangegeven adressen gelezen en/ geschreven. Op het eerste gezicht lijkt het aangewezen b.v. de opt opdracht a 0 b aldus te interpreteren: tel de getallen op de adressen a en b op. Rest nog slechts de vraag, wat we met het resultaat zullen doen. Opbergen in A heeft weinig zin, tenzij we A ook werkelijk als geheugenplaats beschouwen en een adres geven. Anders kunnen we n.l. niet meer op de som opereren, tenzij we deze eerst met een andere opdracht in het geheugen terugschrijven.

Een andere interpretatie is: tel (a) op bij A en berg het resultaat in b. Dit heeft het nadeel, dat we in vele gevallen niet de bedoeling hebben het resultaat in het geheugen terug te schrijven, doch er eers nog andere bewerkingen op willen uitvoeren.

4. 2+1 -adrescode. Zie verschil tussen 1- en 1+1 -adrescode.

5. 3-adrescode. De meest voor de hand liggende interpretatie van de opdracht $f/_{abc}$ luidt nu: Voer de operatie f uit op (a) en (b) en schrijf het resultaat op c. Alle drie adressen zijn nu variabel en het nut van een accumulator is nu slechts gelegen in het snel bereikbaar zijn als geheugenplaats.

6. 3+1-adrescode. Duidelijk na het voorgaande.

We zullen nu de verdiensten van deze verschillende codes even nagaan aan de hand van een subroutine voor worteltrekking. Precieser gezegd: De subroutine vormt uit de (positieve) breuk $\alpha$ de opvolgende benaderingen $\alpha_0$ $\alpha_1$ ... voor $\sqrt{\alpha}$ volgens $\alpha_{j+1} = \frac{1}{2}(\alpha_j + \frac{\alpha}{\alpha_j})$ en stopt wanneer $\alpha_j - \alpha_{j+1} < \delta$ is geworden. Als beginschatting nemen we de "bijna 1".

In de een-adrescode zoals b.v. bij de ARRA in gebruik, zou deze routine als volgt luiden. We komen de routine binnen met het adres c waar we vandaan gekomen zijn in A en $\alpha$ b.v. in S.

```
Aanroep:        c     0/c
              c+1     7/d
              c+2     ...

Subroutine:   d+0       0   )7/2(
                1      10   d+17
                2      11   d+18
                3       0   )<1(
   d+15  -->    4      10   d+19   1
                5       0   d+18   1
                6      13   d+19   2
                7       3   d+20   1
                8       8   d+19   1
                9      14   1      1
               10       0   d+21   1
               11       7   d+15   0  -->
               12       8   d+21   1
               13       0   d+19   1
               14       7   d+4    1  ⇒
   d+11  -->   15       8   d+21
               16       0   d+19
               17     (7    c+2)
               18     (α)
               19     (α_j)
               20       v
               21       δ
```

De hierin gebruikte opdrachten hebben de volgende betekenis:

| | | |
|---|---|---|
| 0 | a | Vormt (a)+(A) en zet dit in A |
| 3 | a | Zet (S) in A en a. |
| 7 | a | Verplaatst besturing naar a als (A) $\geqslant$ 0 is. Neemt anders volgende opdracht. |
| 8 | a | Vervangt (A) door (A)-(a). |
| 10 | a | Zet (A) in a en maakt A schoon. |
| 11 | a | Zet (S) in a en maakt A schoon. |
| 13 | a | Deelt (A) afgerond door (a) en plaatst quotient in S. Maakt A niet schoon. |
| 14 | a | Schuift (A) a plaatsen naar rechts. Links wordt het teken van (A) aangevuld. |

Achter elke opdracht van de cyclus in deze subroutine is aangegeven hoeveel omwentelingen deze opdracht kost. Hierbij is aangenomen dat de deling ongeveer een hele omwenteling kost, hetgeen in overeenstemming is met onze afspraak 32 woorden van 32 digits op de omtrek te plaatsen.

We komen zodoende tot 11 omwentelingen per cyclus. Men ziet dat het feit, dat deling (en vermenigvuldiging) c.a. 32 woordlengten in beslag nemen nog slechts 10% in tijd uitmaakt. Nemen we in plaats van adres d+20 een algemeen "vuilnis" adres op een andere track, dan moeten we nog de schakeltijd in rekening brengen. Nemen we aan dat we een magnetische trommel hebben met 20 msec omwentelingstijd, dan kunnen we nog met relais schakelen met een operatietijd, die we conservatief op 10 msec zullen stellen. Dit scheelt in dit geval 1 omwenteling. Hebben we daarentegen een megahertz kwiktank met een cyclus van 1 msec dan is dit uiteraard ontoelaatbaar. De schakeltijd zal dan van de orde van 500 $\mu$sec of minder moeten zijn.

De geheugenruimte die dit programma inneemt is vrij critisch op het toaal aantal geheugenplaatsen. Nemen we aan dat een opdracht een halve woordlengte beslaat (b.v. 4 functiedigits en 11 of 12 adresdigits), dan komen we in totaal tot 13 woorden van 32 digits. Hiervan neemt de cyclus met bijbehorende getaladressen er $9\frac{1}{2}$ voor zijn rekening.

Laat ons zien wat er van dit alles terecht komt in systeem no.2, dat ons de mogelijkheid van optimaal programmeren biedt, d.w.z. de mogelijkheid onze opdrachten en getallen zodanig in het geheugen te plaatsen dat zodra we een opdracht hebben ingelezen, juist het adres waar de opdracht betrekking op heeft, gelezen kan worden en omgekeerd, zodra een opdracht geheel is uitgevoerd, juist de volgende opdracht gelezen kan worden.

We kunnen al dadelijk zeggen, dat het minimum aantal omwentelingen per cyclus 5 zal zijn zolang we het getal $\alpha_j$ slechts op een plaats op de omtrek neerzetten, aangezien deze plaats 4 x wordt aangeroepen en we voor de deling ook nog een omwenteling nodig hebben. Dit minimum is inderdaad bereikbaar mits we de opdrachten d+12 en d+13 uit ons vorige programma laten stuivertje wisselen, (adres van $\delta$ wordt ook 2 x aangeroepen!). De cyclus zou er dan b.v. als volgt uitzien (beginnende op plaats 0). De (relatieve) verwijzingsadressen staan tussen haakjes.

| | | |
|---|---|---|
| 0 | 0 | 1(+2) |
| 1 | $\alpha$ | |
| 2 | 13 | 3(+2) |
| 3 | $\alpha_j$ | |
| 4 | 3 | 5(+2) |
| 5 | v | |
| 6 | 8 | 3(+1) |
| 7 | 14 | 1(+2) |
| 8 | 8 | 10(+5) |
| 9 | 0 | 10(+2) |
| 10 | $\delta$ | |
| 11 | 7 | 15(+1) → |
| 12 | 0 | 3(+28) |
| 13 | 10 | 3(+1) |
| 14 | 7 | 0 ⟹ |

De benodigde geheugenruimte is nu echter gestegen tot 15 hele woorden voor de cyclus.

Zijn we bereid 5 geheugenplaatsen extra ter beschikking te stellen dan kunnen we het aantal omwentelingen nog drukken tot 3, als volgt:

| | | |
|---|---|---|
| 0 | 0 | 1(+2) |
| 1 | $\alpha$ | |
| 2 | 13 | 3(+2) |
| 3 | $\alpha_j$ | |
| 4 | 3 | 5(+2) |
| 5 | v | |
| 6 | 8 | 8(+3) |
| 7 | 2 | 8(+3) |
| 8 | $\alpha_j$ | |
| 9 | 14 | 1(+2) |
| 10 | 10 | 17(+9) |
| 11 | 0 | 12(+2) |
| 12 | $\delta$ | |
| 13 | 7 | 20(+1) → |

| | | |
|---|---|---|
| 14 | 8 | 15(+2) |
| 15 | $\delta$ | |
| 16 | 0 | 17(+2) |
| 17 | $\alpha_j$ | |
| 18 | 2 | 3(+21) |
| 19 | 7 | 0 $\Longrightarrow$ |

Willen we deze subroutine nu ook in een 2-adrescode gaan bekijken, dan moeten we ons eerst even realiseren hoe de opdrachten er ongeveer zullen gaan uitzien. Uitgaande van de opmerking die dienaangaande reeds is gemaakt, zullen we veronderstellen een accumulator (A) en nog een hulpregister (S) te bezitten, beiden als geheugenplaats aanroepbaar.

We stellen nu de volgende opdrachten voor:

| | | | |
|---|---|---|---|
| 0 | a | b | (a) + (b) $\rightarrow$ A |
| 1 | a | b | (a) - (b) $\rightarrow$ A |
| 2 | a | b | $\frac{(a)}{(b)} \rightarrow$ S (afgerond) |
| 3 | a | b | (A) a plaatsen naar rechts en schrijf het resultaat op b. |
| 4 | a | b | (a) $\rightarrow$ b |
| 5 | a | b | Vorm (a) + (A) en indien dit $\geq$ 0 is, verplaats dan de besturing naar b. Anders gewoon volgende opdracht. |

De cyclus uit de subroutine zou nu de volgende vorm krijgen:

| | | | | |
|---|---|---|---|---|
| 0 | 2 | 7 | 8 | 2. |
| 1 | 1 | S | 8 | 1 |
| 2 | 3 | 1 | S | 1 |
| 3 | 5 | 9 | 10 | 1 $\rightarrow$ |
| 4 | 0 | 8 | S | 1 |
| 5 | 4 | A | 8 | 1 |
| 6 | 5 | S | 0 | 1 $\Longrightarrow$ |
| 7 | $\alpha$ | | | |
| 8 | $\alpha_j$ | | | |
| 9 | $\delta$ | | | |

Tijd: 8 omwentelingen.
Geheugenruimte: 10 hele woorden.

De optimaal geprogrammeerde 2-adrescode met of zonder ter beschikkingstellen van extra geheugenruimte, biedt weinig interessante gezichtspunten meer, daar de overwegingen die ons bij de 1+1 adrescode tot een minimum van 5 resp. 3 omwentelingen brachten, ook nu gelden. Wel moeten

we opmerken dat we langzamerhand met een ruimteprobleem te kampen krijgen. Bij 2048 woorden en 16 opdrachten blijven er nog 6 digits over voor ons verwijzingsadres, dat we nu wel relatief moeten nemen.

Volledigheidshalve geven we de programma's nog even:

```
0     2     1     2     (+3)
1     ∝
2     ∝j
3     1     S     2     (+1)
4     3     1     S     (+2)
5     5     S     0           ⟹
6     5     7     10    (+2)  ⟶
7     δ
8     0     2     S     (+1)
9     4     A     2     (+28)
```

Tijd: 5 omwentelingen.
Geheugenruimte: 10 hele woorden.

```
0     2     1     2     (+3)
1     ∝
2     ∝j
3     1     S     5     (+3)
4     4     A     5     (+3)
5           ∝j
6     3     1     S     (+2)
7     4     A     11    (+6)
8     5     9     14    (+2)  ⟶
9     δ
10    0     11    S     (+2)
11          ∝j
12    4     A     2     (+24)
13    5     S     0           ⟹
14    ....
```

Tijd: 3 omwentelingen.
Geheugenruimte: 14 hele woorden.

De 3-adrescode zonder accumulator blijkt in dit geval geen nuttig effect te sorteren. Willen we onze opdrachten niet al te veel ad hoc kiezen dan zouden we b.v. kunnen hebben:

| | | |
|---|---|---|
| 0 | a b c | (a)+(b) → c |
| 1 | a b c | (a)-(b) → c |
| 2 | a b c | (a)x(b) → c (afgerond) |
| 3 | a b c | (a):(b) → c (afgerond) |
| 4 | a b c | $2^b$(a) → c |
| 5 | a b c | $2^{-b}$(a) → c |
| 6 | a b c | (a)÷(b) ≥ 0 dan besturing naar c. |

Weliswaar bevat onderstaand programma slechts 6 opdrachten, doch daar staat tegenover dat we nu in geen geval onze woorden van 32 binalen zullen kunnen handhaven. Houden we vast aan een geheugencapaciteit van 2048 plaatsen en de mogelijkheid van 16 verschillende opdrachten, dan komen we tot een minimum van 37 binalen

| | | | | |
|---|---|---|---|---|
| 0 | 3 | 6 | 7 | 8 |
| 1 | 1 | 8 | 7 | 9 |
| 2 | 5 | 9 | 1 | 10 |
| 3 | 6 | 10 | 11 | 12 → |
| 4 | 0 | 7 | 10 | 7 |
| 5 | 6 | 7 | 10 | 0 ⟹ |
| 6 | $\alpha$ | | | |
| 7 | $\alpha_j$ | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |
| 11 | $\delta$ | | | |

Tijd: 8 omwentelingen.

Er is geen sprake van dat deze routine zodanig geprogrammeerd kan worden, dat hij ook afgewerkt kan worden in drie omwentelingen. Daarvoor is het aantal adressen dat twee- of meermalen aangeroepen wordt te groot. Hierbij komt nog dat het invoeren van nog een adres weer een uitbreiding van het aantal binalen vereist. Natuurlijk is het mogelijk door wederinvoering van A onze opdrachten nog wat werkzamer te maken, door twee operaties op drie getallen uit te voeren en het resultaat in A op te bergen (dus b.v. de opdracht (a)x(b)+(c) → A). Wellicht is het dan mogelijk ook in deze code optimaal te programmeren, doch veel schieten we hier niet mee op. Van de tot nu toe verkregen resultaten geven we nog even een overzicht. De gegevens slaan op de cyclus in de subroutine.

| | Tijdsduur (in omw.) | Geheugenruimte | Opmerkingen |
|---|---|---|---|
| 1 adres | 11 | $9\frac{1}{2}$ | |
| 1+1 | 5 | 15 | |
| | 3 | 20 | |
| 2 | 8 | 10 | |
| 2+1 | 5 | 10 | Slechts 6 |
| | 3 | 14 | digits voor verwijzingsadres |
| 3 | 8 | 12 | minimaal 37 digits per woord |

We merken op dat bij de optimaal geprogrammeerde gevallen van 3 omwentelingen de deling een aanzienlijke rol gaat spelen in de tijdsduur.

Nu is het inderdaad mogelijk de tijdsduur van de deling (en vermenigvuldiging) te bekorten. Wanneer we de beide getallen die we willen delen eenmaal hebben ingelezen, kunnen we deling op een b.v. 10 x hogere frequentie uitvoeren. Dit heeft echter het nadeel dat de optelapparatuur 10 x sneller moet kunnen werken dan voor een optelopdracht vereist is.

Een andere methode is de volgende:

We splitsen onze 32 binalen in 8 groepen van 4 digits en behandelen onze getallen verder alsof ze in het 16-tallig stelsel waren neergeschreven. Een woord schrijven we nu op 4 tracks en wel zo, dat we telkens de 4 binalen van 1 sedecimaal tegelijk inlezen. Dit vereist natuurlijk 4 x zoveel lees- en schrijfapparatuur als vroeger. In een omwenteling kunnen we dan 128 woorden lezen i.p.v. 32. Leren we nu onze machine de tafels van vermenigvuldiging van 0 x 0 = 0 tot en met 15 x 15 = 1 met een carry van 14, dan behoeft een vermenigvuldiging slechts 8 woordlengten te duren.

In woordlengten is de winst een factor 4, in tijd zelfs een factor 16. Mede in verband met het feit dat we nu de beschikking hebben over 128 woorden per omtrek. kost het nu niet de geringste moeite de gehele subroutine in twee omwentelingen te programmeren in een van de optimaal programmeerbare codes.

Willen we de subroutine nog sneller maken, dan kunnen we twee dingen doen:

a. De cyclus uit de subroutine meermalen langs de omtrek uitschrijven. In het laatste beschouwde geval zouden we dit b.v. 4 maal kunnen doen. Dit helpt echter niet zo bar veel. Elke keer dat ik een nieuwe benadering $\alpha_j$ heb gevonden, zal ik deze ergens op de omtrek neer moeten zetten.

Tijdens de volgende cyclus wil ik die waarde weer gebruiken. Hoe slim ik de opdrachten ook neerzet, dit kost minstens een omwenteling.

I.h.a. bedraagt in dit geval de tijd voor 1 cyclus bij m x uitschrijven $\frac{m+1}{m}$ omwenteling.

b. Een of meer snelle geheugenplaatsen invoeren. Hebben we 1 snelle geheugenplaats, waar we onze $\alpha_j$ in opbergen, dan kunnen we b.v. in de 1+1 adrescode zonder meer 1 omwenteling halen en bovendien nog geheugenruimte sparen, daar de extra schrijfopdrachten en opbergplaatsen van de $\alpha_j$ vervallen. Het programma zou er dan b.v. als volgt uitzien: (de snelle geheugenplaats is met G aangegeven).

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | (+2) |
| 1 | $\alpha$ | | |
| 2 | 13 | G | (+8) |
| ⋮ | | | |
| 10 | 3 | 11 | (+2) |
| 11 | v | | |
| 12 | 8 | G | (+1) |
| 13 | 14 | 1 | (+2) |
| 14 | ... | | |
| 15 | 0 | 16 | |
| 16 | ζ | | |
| 17 | 7 | ... | (+1) ⟶ |
| 18 | 0 | 19 | |
| 19 | ζ | | |
| 20 | 0 | G | |
| 21 | 10 | G | |
| 22 | 7 | 0 | ⟹ |

Wanneer we nu nog eens de methode genoemd onder a toepassen, blijkt deze veel meer effect te sorteren. Bij m keer uitschrijven blijkt een cyclus slechts $\frac{1}{m}$ omwenteling te kosten.

Van de andere mogelijke organisatievormen van onze machine wil ik nog slechts één geval even aanstippen, en wel een extreem. Gesteld dat we een programma hebben zonder besturingsverplaatsingen. We lezen nu onze opdrachten parallel van de trommel (d.w.z. alle 32 digits tegelijk), nemen voor al onze getallen snelle geheugenplaatsen, maken voorts een instantane opteller en vermenigvuldiger (deler) en zijn nu in staat onze opdrachten met klokpuls-frequentie uit te voeren, d.w.z. 20 sec per opdracht.

Geen slecht resultaat voor een magnetische trommel! Over de kosten van een instantane vermenigvuldiger zullen we het echter maar niet hebben, evenmin als over het voorkomen van programma's zonder besturingsverplaatsingen.

Zoals reeds in de inleiding werd opgemerkt, is het voorgaande geen uitputtend onderzoek geweest. Verre van dien. De bedoeling was slechts te onderzoeken wat we, bij niet al te onredelijke programma's, van de verschillende organisatievormen mogen verwachten. Tevens moge duidelijk geworden zijn, dat een en ander sterk afhangt van de aard van het programma, i.h.b. van het aantal vermenigvuldigingen en delingen en, bij cyclische geheugen, van het aantal cycli in het programma en hun gemiddelde lengte.