MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

REKENAFDELING

Programmering voor de ARMAC

DEEL I

door

E.W. Dijkstra

MR 25

1956

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

VOORWOORD

Dit rapport bevat de noodzakelijke gegevens, waarover men moet beschikken om een programma op te kunnen stellen voor de ARMAC (Automatische Rekenmachine van het MAthematisch Centrum).

Na de functionele beschrijving van de machine worden de conventies uiteengezet, die voortvloeien uit de permanent in het geheugen aanwezige standaardprogramma's.

Voor een uitvoeriger uiteenzetting over programmeringstechnieken in het algemeen verwijzen wij naar de syllabus van de "Cursus 1955-'56 Programmeren voor automatische Rekenmachines" onder leiding van Prof. Dr. Ir. A. van Wijngaarden en E.W. Dijkstra.

1. Inleiding

De ARMAC is een automatische digitale machine.

De ARMAC is een digitale machine, omdat hij in staat is op getallen de rekenkundige grondbewerkingen uit te voeren, n.l. optellen, aftrekken, vermenigvuldigen en delen. (Omdat er geen "deler" in de ARMAC is ingebouwd, worden quotienten in feite min of meer langs een omweg bepaald; in dit eerste overzicht in vogelvlucht is dat echter van geen belang.) Tengevolge hiervan kan de ARMAC slechts dan een probleem oplossen, indien het in een geschikte numerieke vorm is gegoten. Bij het z.g. "handrekenen" (d.w.z. met tafelmachines) doet zich tot op zekere hoogte dezelfde situatie voor (b.v. de integratie van differentiaalvergelijkingen). Dit aspect van de "aanpassing" van een probleem aan de ARMAC is daarom niet het onderwerp van dit rapport.

De ARMAC is een automatische rekenmachine, omdat hij in staat is lange berekeningen uit te voeren zonder tussentijds menselijk ingrijpen. Om de machine hiertoe in staat te stellen, moet men hem eerst de daartoe benodigde gegevens verschaffen. Deze gegevens zijn tweeërlei: een gedeelte is numerieke informatie, b.v. de coëfficiënten van een vergelijking of de waarde van één of meer parameters (om b.v. gebied en interval van een onafhankelijk variabele vast te leggen) of de waarde van constantes (als $\frac{1}{4}\pi$, etc.); een ander gedeelte is logische informatie, n.l. een volledige beschrijving van de uit te voeren berekening.

Wij veronderstellen, dat een dergelijke gedetailleerde beschrijving van de berekening opgesteld is in termen van de rekenkundige grondbewerkingen, elk aangegeven door wat genoemd wordt een opdracht (of ook wel een instructie). In deze reeks arithmetische opdrachten moeten nog andere soorten opdrachten tussengevoegd worden, n.l. opdrachten die getallen "van de ene plaats naar de andere sturen" en opdrachten, waarmee men aan kan geven, dat een bepaald stuk van de berekening een aantal keren herhaald moet worden, voordat de machine door mag gaan.

Aan enige voorbeelden kan de noodzaak van dergelijke opdrachten reeds nu toegelicht worden.

Vermenigvuldigen kan de ARMAC alleen maar, als, voordat de vermenigvuldiging begint, een van beide factoren is gezet in een speciaal register van het arithmetisch orgaan, een register, dat hier dus funotioneert als een soort vermenigvuldiger-

register. Indien geen der factoren toevallig al in dit register staat, moet de vermenigvuldigopdracht vooraf worden gegaan door een niet-arithmetische instructie, die een der factoren naar dit register stuurt.

Dit soort opdrachten vormt min of meer een uitbreiding van de groep der arithmetische instructies, dit in tegenstelling tot de tweede soort opdrachten, dat nog ingelast moet worden. Zij treden b.v. op als het onmogelijk is, de reeks uit te voeren arithmetische bewerkingen exact van te voren aan te geven: soms hangt deze reeks n.l. af van door de berekening gevormde tussenresultaten. Laten wij de iteratieve processen beschouwen: hier wordt een resultaat gevormd als limiet van opeenvolgende benaderingen. Het grondpatroon van een dergelijke berekening is de verbetering van een benadering, d.w.z. als een benadering van het gewenste eindresultaat gegeven is, wordt met behulp hiervan een betere benadering gevormd, dan wordt deze benadering verbeterd, en zo voort, totdat de limiet bereikt is, d.w.z. totdat de correctie op de vorige benadering nul blijkt te zijn. Het aantal malen, dat de "verbetering" uitgevoerd moet worden, kan van geval tot geval aanzienlijk variëren: hoe beter de eerste schatting, des te sneller is de limiet bereikt. Om dit te kunnen benutten, wordt aan de machine overgelaten, om vast te stellen, of de limiet bereikt is of niet: aan het einde van elke iteratiestap wordt de nieuwe benadering vergeleken met de vorige: als het verschil nog niet verwaarloosbaar klein is, moet er nog meer "verbeterd" worden ("de machine moet in de iteratieve cyclus blijven"), is echter het verschil gelijk aan nul (of althans klein genoeg), dan is een voldoend goede benadering van het gewenste antwoord gevonden en de machine kan doorgaan naar de volgende fase van de berekening ("de machine verlaat de iteratieve cyclus"). Om dit mogelijk te maken, worden de arithmetische opdrachten, die de berekening van de nieuwe benadering uit de oude omschrijven, gevolgd door een paar "decisie-opdrachten", dankzij welke de machine uitvindt of hij terug moet gaan om nog een keer te itereren, of dat hij door moet gaan. Het zijn deze decisie-opdrachten, waaraan de moderne automatische rekenmachines hun enorme flexibiliteit ontlenen; wij hebben hier vanzelfsprekend slechts één van de vele toepassingen genoemd.

Een reeks opdrachten, die tesamen een rekenschema volledig beschrijven, wordt een programma genoemd.

Samen met de numerieke begingegevens wordt het programma in een speciale code op telexband (vijf-gats-papierstrook) geponst. Alle informatie, die de ARMAC nodig heeft voor het uitvoeren van de betroffen berekening, is nu op een of meer banden geponst. Deze banden worden door de fotoelectrische bandlezer van de ARMAC gelezen; op deze wijze wordt de benodigde informatie in de machine ingebracht. Als regel worden alle banden eerst gelezen, voordat de berekening begint.

De fotoelectrische bandlezer is het belangrijkste invoer-mechanisme van de ARMAC. Om de resultaten aan de buitenwereld prijs te geven, heeft de ARMAC de keuze tussen twee uitvoermechanisme's: de bandponser en de electrische schrijfmachine. De ponser wordt gebruikt voor de uitvoer van gegevens, die later weer door de machine opgenomen moeten worden (of algemener, die bedoeld zijn voor verdere automatische verwerking); voor eindresultaten, die van interesse zijn voor de menselijke gebruiker, wordt de schrijfmachine gebruikt. Beide methodes van uitvoer gebruiken bewegende mechanische delen en kosten daarom relatief veel tijd. Ook om deze reden moet de uitvoer van onbelangrijke nevenresultaten niet worden aangemoedigd.

Het laatste onderdeel van de ARMAC, dat wij hier vermelden, is het geheugen. Het geheugen wordt gebruikt om alle informatie, die van de banden gelezen is (dus het programma en de numerieke begingegevens) te onthouden; naast deze informatie bergt het geheugen de tussenresultaten van de berekening, d.w.z. uitkomsten, die misschien niet uitgetypt hoeven te worden, maar die nodig zijn voor een later stadium van de berekening.

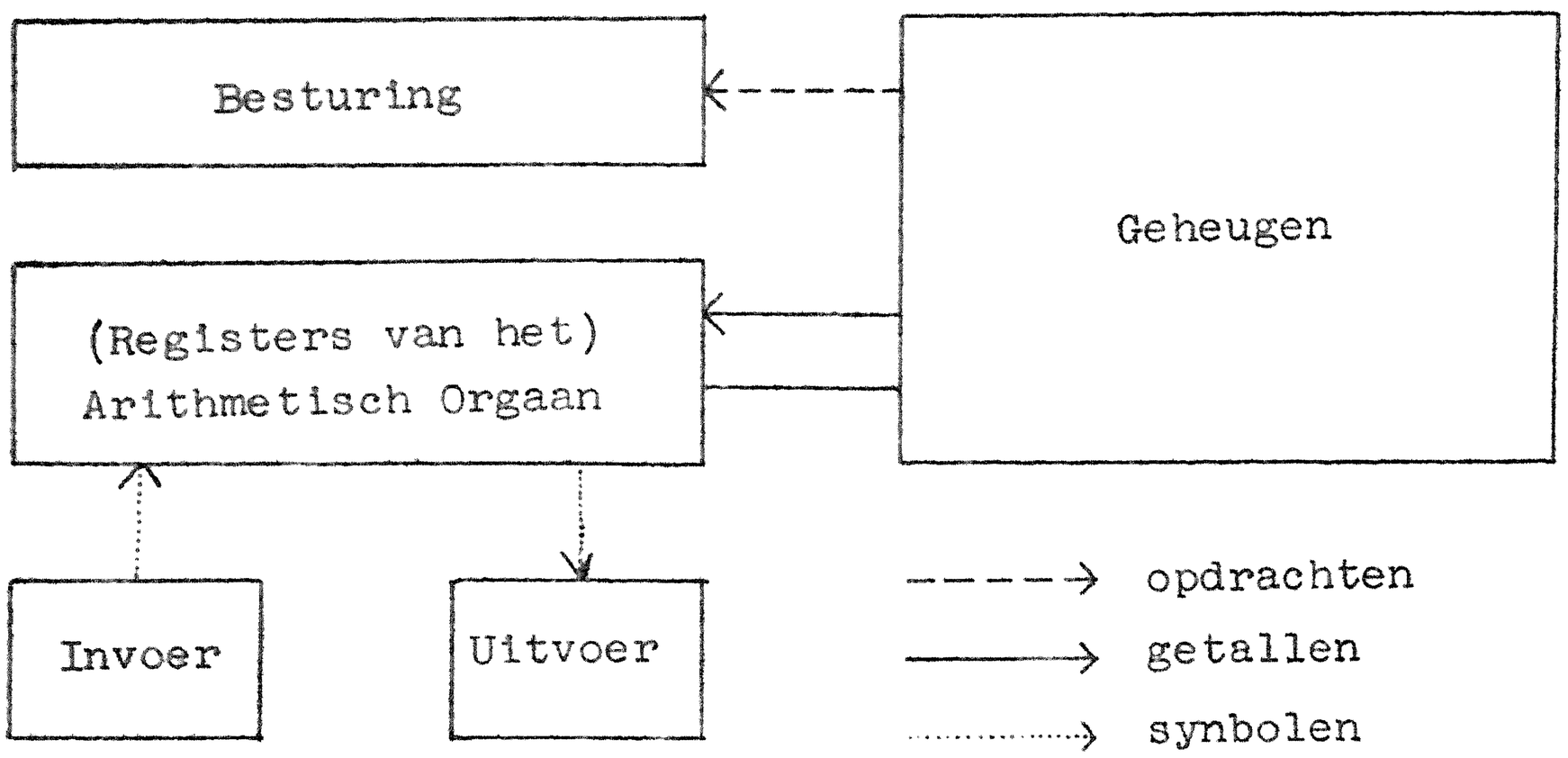
Het volgende schema toont de informatiestroom tussen de vijf belangrijkste onderdelen van de ARMAC.

Het arithmetisch orgaan is in staat die rekenkundige bewerkingen uit te voeren.

Het geheugen is in staat om cijfers te herbergen; om in dit geheugen ook het programma vast te kunnen leggen, is voor de opdrachten een speciale cijfercode gekozen.

De besturing nu vormt de schakel tussen de opdracht, zoals deze het geheugen verlaat (een rij cijfers) en zijn werkelijke uitvoering. Als b.v. een optelling moet worden uitgevoerd, zorgt de besturing er voor, dat het addendum aan het arithmetisch orgaan wordt toegevoerd, het geeft eveneens de nodige signalen dat het arithmetisch orgaan, dat kàn optellen, nu ook inderdaad gáát optellen. Zodra een opdracht is uitgevoerd,

haalt de besturing de volgende opdracht uit het geheugen, etc.



Het schema laat zien, dat slechts vanuit de registers van het arithmetisch orgaan informatie naar het geheugen kan worden gezonden. Dit "schrijven in het geheugen", zoals wij zeggen, is om redenen, die later duidelijk zullen worden, aangeduid als transport van getallen, hoewel gedurende de invoer het programma (een reeks opdrachten dus) ook langs deze weg naar het geheugen gaat. Het meest voorkomende gebruik is echter het opbergen van numerieke tussenresultaten.

Invoer van informatie van de band naar het geheugen loopt via de registers van het arithmetisch orgaan. Van de band wordt steeds een rijtje van vijf (al of niet) gaatjes, een pentade, zoals wij zeggen, gelezen. Een aantal opeenvolgende pentades - elk b.v. een cijfer van een decimaal getal specificerend - wordt tot één geheel geassembleerd. Deze assemblage wordt in de registers verricht; het resultaat wordt vandaar naar de gewenste plaats in het geheugen gestuurd. Om soortgelijke redenen zijn ook de uitvoermechanismes aan de registers verbonden.

Het schema toont twee routes, waarlangs informatie het geheugen verlaat. De onderste pijl naar links duidt het transport aan van getallen naar het arithmetisch orgaan (om daar b.v. afgetrokken of vermenigvuldigd te worden). Langs de bovenste pijl naar links worden opdrachten naar de besturing getransporteerd; hier reageert de machine dus op ("luistert naar") de instructies van het programma in zijn geheugen. Hoe de besturing deze informatie na ontrafeling doorgeeft aan alle andere onderdelen van de machine, is in het schema niet aangegeven.

2. De binaire voorstelling

In het voorgaande hebben we van de term "informatie" een tamelijk los gebruik gemaakt, soms getallen, soms opdrachten bedoelend en soms wat we symbolen hebben genoemd. Zoals de lezer misschien vermoedt, is het onderscheid tussen deze verschillende soorten informatie niet zo scherp, als men op het eerste gezicht zou kunnen veronderstellen. We zullen zien, dat al deze verschillende soorten informatie voorgesteld worden door cijferrijen en dat het verschil hoofdzakelijk een kwestie is van (de meest gebruikelijke) interpretatie.

Om technische redenen gebruikt de ARMAC het tweetallig stelsel. Hier hebben de opeenvolgende cijfers betrekking op de successieve machten van 2, in plaats van van 10, zoals we gewend zijn. Een kleine nadere beschouwing van hoe het tientallig stelsel eigenlijk in elkaar zit, zal ons wijzen, hoe het tweetallige stelsel werkt.

We weten, hoe de waarde van een decimaal getal bepaald is door de opeenvolgende cijfers, b.v.

$$7384 = 7 \times 1000 + 3 \times 100 + 8 \times 10 + 4 \times 1$$

Of iets systematischer: vermenigvuldig het eerste cijfer (= 7) met 10 en tel het volgende cijfer (= 3) bij dit product (d.w.z. 70 + 3 = 73); vermenigvuldig deze som met 10 en tel het volgende cijfer (= 8) op (d.w.z. 730 + 8 = 738); vermenigvuldig deze som weer met 10 en tel het volgende (en laatste!) cijfer (= 4) op (d.w.z. 7380 + 4 = 7384). Nu is het proces voltooid.

Omgekeerd weten we, hoe we de opeenvolgende decimale cijfers kunnen extraheren: deel het gegeven getal door de hoogste macht van 10, die niet groter is dan het gegeven getal (hier 1000); het quotient is gelijk aan het eerste cijfer, de rest wordt op dezelfde wijze behandeld, d.w.z. wordt gedeeld door de één lagere macht, etc. Een alternatief biedt de volgende methode: deel het gegeven getal (b.v. weer 7384) door 10; de rest (= 4) is gelijk aan het laatste cijfer - of "het minst significante cijfer", zoals wij zeggen -; het quotient (= 738) wordt weer door 10 gedeeld, waarbij de rest (= 8) nu het op één na laatste cijfer levert, etc.

Het is natuurlijk volslagen zinloos, om met potlood en papier de decimale cijfers van een getal als boven te "berekenen", omdat we ze al gebruiken, om het gegeven getal te noteren. Desondanks is het nuttig, zich de deugdelijkheid van deze

rekenschema's te realiseren, omdat zij de weg zullen wijzen naar andere talstelsels, in ons geval naar het tweetallige.

We trekken uit de rekenschema's twee (welbekende) conclusies.

Als aan de minst-significante kant van een decimaal getal een extra nul wordt gezet, wordt het getal met tien vermenigvuldigd; geheel analoog wordt een tweetallig getal met twee vermenigvuldigd, als er een extra nul achter geplaatst wordt.

Voorts merken we op, dat elk decimaal cijfer - berekend als rest na deling door tien - kleiner dan tien (de deler) moet zijn; analoog zijn de cijfers van een tweetallig getal kleiner dan twee, dus 0 of 1. Als voorbeeld zullen we berekenen de binaire voorstelling van (het decimale getal) 25:

```
25:2 = quotient 12, rest 1
12:2 =    "      6,  "   0
 6:2 =    "      3,  "   0
 3:2 =    "      1,  "   1
 1:2 =    "      0,  "   1
```

Nu is het quotient = 0 en ons antwoord is gevonden: de binaire voorstelling van 25 luidt: 11001. Omgekeerd correspondeert met het binair getal 11001 het getal:

$$1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 25$$

(Hier zijn de machten van twee gebruikt, dus $2^4 = 16$, $2^3 = 8$, $2^2 = 4$, $2^1 = 2$ en $2^0 = 1$.) Weten wij een keer, dat het binaire getal 11001 decimaal = 25 is, dan zien wij onmiddellijk, dat binair 1100100 = decimaal 100 (n.l. 4 x 25).

In de ARMAC is de "natuurlijke" eenheid van informatie een reeks van 34 binaire cijfers. Een dergelijke cijferrij wordt een woord genoemd. Om een getal voor te stellen (te onthouden) wordt altijd een volledig woord gebruikt.

Laat ons ons voor het ogenblik beperken tot positieve gehele getallen: zij worden voorgesteld door een nul - het tekencijfer genaamd - gevolgd door 33 binaire cijfers. Als de binaire representatie van het getal minder dan 33 binaire cijfers gebruikt, wordt het tekencijfer gevolgd door een aantal "non-significante" nullen. Als bijvoorbeeld de ARMAC het positieve getal 25 moet hanteren, manipuleert hij

met het woord

+25 = 0 000 00000 00000 00000 00000 00000 11001

Als de binaire representatie van een geheel getal meer dan 33 cijfers nodig heeft, dan overschrijdt het getal de capaciteit en het kan het niet in een woord worden opgeborgen. Hieruit volgt, dat het grootste positieve getal, dat in een woord kan worden vastgelegd, gelijk is aan
85899 34591 = 2^{33} -1; hiermede correspondeert het woord

+85899 34591 = 0 111 11111 11111 11111 11111 11111 11111

Om een getal van teken te veranderen (met -1 te vermenigvuldigen), worden alle cijfers van het overeenkomstige woord (inclusief het tekencijfer) geinverteerd, d.w.z. nullen worden door enen vervangen en enen door nullen. Enige voorbeelden:

-25 = 1 111 11111 11111 11111 11111 11111 00110
+ 3 = 0 000 00000 00000 00000 00000 00000 00011
- 3 = 1 111 11111 11111 11111 11111 11111 11100

Wij willen niet nalaten er de nadruk op te vestigen, dat deze wijze van voorstelling van negatieve getallen niet inherent is aan het tweetallig stelsel: het is een extra conventie betreffende de voorstelling van getallen in de ARMAC. Wel zien we, dat het voor de ARMAC gekozen systeem aan de evidente eis voldoet, dat twee keer achter elkaar van teken wisselen het oorspronkelijke woord weer teruggeeft.

Wij vermelden enige consequenties.

De capaciteitsbeperking kan nu als volgt geformuleerd worden: gehele getallen moeten in absolute waarde kleiner zijn dan 2^{33}.

Het getal nul heeft twee representaties, namelijk de woorden:

+0 = 0 000 00000 00000 00000 00000 00000 00000
-0 = 1 111 11111 11111 11111 11111 11111 11111

Wij laten de lezer de verificatie van de volgende beweringen:

a. Boven omschreven methode van voorstelling van negatieve getallen is in overeenstemming met de volgende regel voor de optelling: de tekencijfers van de twee addenda worden behandeld als "de meest significante cijfers", d.w.z. ook zij worden (samen met de eventuele overdracht van de plaats

er rechts naast) gewoon opgeteld; als op de plaats van de tekencijfers een overdracht ontstaat, dan moet deze aan de minst significante zijde opgeteld worden. Deze overdracht wordt met de sprekende naam "end-around carry" aangeduid.

b. Optelling van twee getallen met gelijke absolute waarde, maar tegengesteld teken levert de som -0 af. Aangezien de ARMAC aftrekt door optelling van het geinverteerde addendum, geldt hetzelfde resultaat voor de aftrekking van twee gelijke nullen. Slechts in exceptionele gevallen levert het arithmetisch orgaan +0 af als uitkomst van een additieve bewerking, n.l. na optelling van +0 bij (of aftrekking van -0 van) +0.

We hebben eerder vermeld, dat het geheugen tweeërlei informatie herbergt, n.l. numerieke gegevens (getallen) en het programma (opdrachten). De inhoud van het geheugen bestaat nu louter uit woorden, die in overeenstemming met deze twee soorten informatie, twee interpretaties toestaan: behalve getallen - als boven beschreven - kunnen woorden ook opdrachten voorstellen.

Slechts 17 binaire cijfers zijn nodig om een opdracht te karakteriseren. Twee opdrachten staan samen in een woord, een in de minst significante helft en een in de andere. Zolang het woord alleen beschouwd wordt als opdrachtenpaar, heeft het "tekencijfer van het woord" niets te maken met positief of negatief: het is alleen maar het meest significante cijfer van een opdracht, evenals het cijfer onmiddellijk rechts van het midden van het woord.

De volledige beschrijving, hoe twee maal zeventien cijfers twee opdrachten specificeren, stellen we echter uit, totdat we voldoende aandacht hebben geschonken aan de organisatorische aspecten van het geheugen.

Het geheugen is n.l. niet alleen een onmisbaar onderdeel van een automatische rekenmachine, het is vaak een van zijn meest karakteristieke onderdelen. De reden hiervoor is evident: de eigenschappen van de arithmetische operaties zijn elders exact gedefinieerd (2 + 5 moet altijd = 7 zijn, onafhankelijk van door welke rekenmachine deze som gevormd wordt), physische verschillende geheugenelementen echter hebben verschillende eigenschappen en deze eigenschappen zijn vaak in hoge mate bepalend voor de snelheid van de machine. Aangezien daarenboven het geheugen nogal kostbaar pleegt te zijn, hebben machineontwerpers altijd veel aan-

dacht en vernuft gespendeerd aan dit onderdeel van hun geesteskind. Het resultaat van deze inspanning is een grote hoeveelheid verschillende geheugens.

3. Het geheugen van de ARMAC

Als informatie naar het geheugen van de ARMAC wordt gestuurd, om daar te worden onthouden, zeggen we, dat deze informatie in het geheugen wordt geschreven.

Omgekeerd, als het geheugen informatie aflevert aan een ander onderdeel van de machine, dan zeggen we, dat er in het geheugen wordt gelezen.

Het volledige geheugen zal in eerste instantie bestaan uit 4096 vakjes, adressen genaamd, elk in staat om één woord te bergen ("Woord" hier natuurlijk in de speciale betekenis van een cijferrij van 34 tweetallige cijfers!)

Als een woord op een bepaald adres wordt geschreven, wordt daardoor de vorige inhoud van dit adres uitgewist. Van nu af aan kan de nieuwe inhoud van dit adres zo vaak gelezen worden, als men wil, totdat er een nieuw woord in wordt geschreven. Zo kan dus eenzelfde adres tijdens een berekening vele verschillende getallen achter elkaar onthouden; het kan vanzelfsprekend niet twee woorden tegelijkertijd onthouden.

Als de woorden in het geheugen geborgen zouden worden als knikkers in een zak, zou het moeilijk zijn, om aan te geven met wèlk getal een bepaalde operatie uitgevoerd moet worden. Om dit mogelijk te maken, zijn de 4096 adressen genummerd van 0 t/m 4095. (In de wandeling wordt ook het nummer van het adres met de naam "adres" aangeduid. Als verwarring dreigt, bestaat er nog de term "geheugenplaats".)

Voor elk getal, dat het geheugen moet onthouden, wordt tijdens de opstelling van het programma een - in die fase van de berekening beschikbaar - adres gereserveerd. Iedere keer, dat dit getal voor de berekening moet worden aangehaald, wordt in het programma een (lezende) opdracht geplaatst, die naar dit adres refereert. Als een dergelijk getal een tussenresultaat van de berekening is, is het dus eerder berekend: op dat ogenblik is het naar het geheugen op de correcte plaats geschreven door een schrijfopdracht die refereerde naar ditzelfde adres. Het adres fungeert dus als "naam" van het getal, ongeacht of het een bij de invoer meegegeven constante, dan wel een variabele is.

Zo zien we, dat we steeds kunnen specificeren, welk getal in elke operatie betrokken is, dankzij het feit, dat de geheugenplaatsen genummerd zijn. Dit genummerd zijn impliceert echter meer: het verschaft de geheugenplaatsen met een "natuurlijke" volgorde. Als we één bij het adres van een geheugenplaats optellen, krijgen we het adres van de "volgende". Deze ordening is juist, wat we nodig hebben, om een programma in het geheugen vast te kunnen leggen, want een programma is niet zo maar een aantal opdrachten, maar een reeks opdrachten, uit te voeren de een na de ander, in een heel bepaalde volgorde.

Dat een woord twee opdrachten herbergt, is slechts een kleine complicatie; hier geldt, dat de opdracht die staat in de laagste (= minst significante) helft geacht wordt te staan vóór de opdracht in de andere helft van hetzelfde woord. De opdracht in de laagste helft, wordt de "a-opdracht van dit adres" genoemd, die in de hoogste helft de "b-opdracht".

De "volgende opdracht", wat betreft de plaatsing is dus als volgt gedefinieerd: op een a-opdracht volgt de b-opdracht uit hetzelfde woord; op een b-opdracht volgt de a-opdracht uit het volgende adres. Tenzij anders gespecificeerd, worden de opdrachten in deze volgorde uitgevoerd. De mogelijkheid, om deze volgorde te onderbreken, is gegeven door speciale opdrachten, die om begrijpelijke redenen sprongopdrachten worden genoemd. Als, zoals wij zeggen, een sprong wordt uitgevoerd, is de in tijd volgende opdracht niet de naar plaatsing volgende; met welke (halve) plaats uit het geheugen de in tijd volgende opdracht dan wel wordt aangehaald, wordt door de sprongopdracht gespecificeerd. Beginnend op deze (halve) plaats gaat dan de machine de opdrachten in volgorde lezen en uitvoeren, totdat hier weer een sprongopdracht aangehaald en uitgevoerd wordt.

Het merendeel der andere opdrachten verwijst naar één adres: bij de lezende opdrachten is dit het adres, waar het getal vandaan gehaald moet worden (om b.v. opgeteld of vermenigvuldigd te worden), bij schrijvende opdrachten is dit het adres, waarvan de inhoud vervangen moet worden. (De nieuwe inhoud van dit adres wordt geacht al aanwezig te zijn in een van de registers, waarin het rekenkundig orgaan de resultaten achterlaat.) Omdat het geheugen $4096 = 2^{12}$ adressen omvat, zijn er twaalf binaire cijfers nodig, om een

willekeurig adres te specificeren. In al deze opdrachten zijn dan ook de twaalf laagste cijfers, die "apart beschouwd" een getal van 0 (= twaalf nullen) t/m 4095 (= twaalf enen) kunnen voorstellen, gereserveerd, om het adres te specificeren, waarop de opdracht betrekking heeft. We hebben gezegd, dat voor elke opdracht zeventien cijfers ter beschikking stonden: de resterende vijf (die aan de hoge kant van de twaalf adrescijfers staan) vormen het z.g. functiegedeelte: hier wordt in een nummercode de aard van de bewerking omschreven: of het een lees-opdracht, dan wel een schrijfopdracht is; als het een schrijfopdracht is uit welk register van het arithmetisch orgaan (dat er twee heeft) de inhoud gecopieerd moet worden, als het een leesopdracht is, hoe het gelezen getal door het arithmetisch orgaan verwerkt moet worden, etc.

Ook kan het functiegedeelte omschrijven, dat de opdracht nòch een lees- nòch een schrijfopdracht is, maar b.v. een sprongopdracht. In dit geval worden de twaalf adrescijfers gebruikt om het adres van de volgende opdracht vast te leggen. Omdat met het adres alleen de plaats van een opdracht in het geheugen niet vastgelegd is, kent de ARMAC twee soorten sprongopdrachten: sprongen naar de a-opdracht en naar de b-opdracht. (M.a.w. de dertiende bit is hier in het functiegedeelte ondergebracht.)

Voor het functiegedeelte zijn dus vijf binaire cijfers gereserveerd. Apart beschouwd vormen deze het opdrachtnummer f, dat in de ARMAC loopt van 0 t/m 29. Twee van de twee en dertig (= 2^5) mogelijkheden n.l. f = 30 en f = 31 worden in de ARMAC niet gebruikt. De ARMAC kent dus in eerste instantie dertig verschillende opdrachten elk gekarakteriseerd door (gecodeerd met behulp van) hun eigen functiegedeelte f. Omdat elke opdracht door de programmeur genoteerd wordt als functiegdeelte f (of te wel opdrachtnummer f) gevolgd door het adres, moet hij de betekenis van de dertig waarden voor f uit zijn hoofd kennen. De praktijk heeft geleerd, dat men, mede dankzij het feit, dat er een vrij duidelijk systeem in zit, zich deze kennis heel snel en in de letterlijke zin des woords spelenderwijs eigen maakt. De exacte omschrijving hiervan stellen we echter uit, (zie "De opdrachtencode") en keren terug tot het adresgedeelte van de opdracht; op de notatie hiervan heeft de indeling van het geheugen een onuitwisbaar stempel gedrukt.

De adressen van het geheugen zijn ingedeeld in 128 kanalen van 32 opeenvolgende adressen elk. Adressen 0 t/m 31 staan in kanaal 0, adressen 32 t/m 63 in kanaal 1 etc., t/m kanaal 127, dat de adressen 4064 t/m 4095 bevat. Zoals de lezer zich realiseert, is het kanaalnummer gelijk aan het getal, gevormd door de apart beschouwde hoogste zeven binaire cijfers van het adres, de laagste vijf cijfers van het adres bepalen, zoals wij zeggen, de plaats in het kanaal. Van nu af aan wordt een adres niet meer als decimaal getal genoteerd, maar "semi-binair": in plaats van adres 1025 spreken we liever van adres 1 van kanaal 32. In zijn programma noteert de programmeur voor elk adres eerst de plaats in het kanaal, gevolgd door de naam van het kanaal. De naam van het kanaal bestaat uit twee gedeelten: een letter, de z.g. sluitletter genaamd, gevolgd door een getal ≤ 31, de z.g. kanaalcorrectie. Er zijn totaal 16 sluitletters. Een hiervan (de X) heeft een vaste betekenis, welke kanaalnummers met de overige 15 sluitletters overeenkomen, mag de programmeur zelf kiezen. Als hij gekozen heeft, dat "A0" de naam van kan. 32 zal zijn, dan is "A1" die van kan. 33, "A2" die van 34, etc. De kanaalcorrectie heeft dus de betekenis van "zoveel kanalen verder" en geeft dus aan, hoeveel malen 32 er bij het adres, zoals dit door plaats en sluitletter bepaald is, opgeteld moet worden, om het bedoelde adres te verkrijgen. (Opm.: Bij bovenstaande keuze zou adres 1025 dus aangeduid worden als "1A0".)

Op alle beweegredenen, die ons er toe geleid hebben, het kanaalnummer niet direct te noteren, maar met behulp van een "naam" als zojuist omschreven, gaan wij nu niet nader in. Het mag de lezer een omslachtige notatie lijken, nochtans hopen wij, dat het systeem zich gaandeweg in zijn ogen zal weten te rechtvaardigen.

Het invoeren van het begrip "kanaal" is niet alleen maar een notatie-technische gril geweest; integendeel: de indeling in kanalen beantwoordt direct aan de physische constructie van het geheugen.

Het grootste gedeelte van het geheugen van de ARMAC bestaat uit een z.g. magnetische trommel, d.w.z. een draaiende cylinder met een magnetiseerbaar oppervlak, langs beschrijvende waarvan 112 lees- en schrijfkopjes zijn. Deze kopjes vervullen beide functies: bij het schrijven wordt door de wikkeling van een kopje een "stroombeeld" gestuurd, ten gevolge waarvan bij het trommeloppervlak een wisselend magnetisch veld ontstaat, karakteristiek voor het te schrijven getal. Omdat in

die tijd de trommel onder het kopje doordraait, wordt een magnetisch patroon langs een streepje op het oppervlak vastgelegd. Draait later dit stukje van de trommel weer onder dat kopje door, dan ontstaat bij het kopje een wisselend magnetisch veld, dat in de wikkelingen van het kopje een wisselende spanning induceert, karakteristiek voor het op de trommel geschreven patroon. Door dit wisselende spanninkje te versterken en te analyseren, kan op de trommel gemagnetiseerde informatie weer gelezen worden.

Elk kopje bestrijkt nu juist een 32-tal adressen, dat samen een kanaal vormt; een kanaal staat langs een cirkelomtrek rondom de trommel. De localisatie van een adres op de trommel geschiedt nu in twee "richtingen": ten eerste wordt het goede kanaal geselecteerd; om in het kanaal het juiste adres te vinden, moet van de goede plaats op de trommelomtrek gelezen worden, d.w.z. gedurende de periode, dat de cijfers van het betroffen adres onder het kopje doordraaien. Deze periode is iets kleiner dan een twee en dertigste deel van de omwentelingstijd van de trommel. De twee en dertig adressen van het kanaal staan n.l. achter elkaar (en wel in volgorde 0,1, ... , 30, 31, 0, 1, ... etc.) op de trommelomtrek, maar zijn door ongebruikte stukjes van elkaar gescheiden. Omdat deze laatste fase van de adreslocalisatie in feite door tijdstippen bepaald wordt, ("begin nu te lezen" en "houdt nu op te lezen") is het interne rythme van de ARMAC afgestemd op de trommel: draait deze, die 75 omwentelingen per seconde hoort te maken, iets te langzaam, dan werkt ook de ARMAC iets te langzaam.

Uit het bovenstaande volgt, dat een adres op de trommel slechts eens per omwenteling, d.w.z. ongeveer om de 14 msec bereikbaar is. Om een willekeurig adres te lezen of te beschrijven, moet dus gemiddeld een halve omwenteling = $\pm$ 7 msec gewacht worden: m.a.w. de trommel is een betrekkelijk "langzaam" geheugen.

De hoge rekensnelheid van het arithmetisch orgaan van de ARMAC zou niet tot zijn recht komen, als de trommel de enige vorm van geheugen was. Het grootste deel van de tijd zou de machine dan werkloos moeten wachten, zowel voor de aanvoer van getallen uit, als ook de afvoer naar het geheugen. Daarom is een gedeelte van het geheugen van de ARMAC "snel".

Het snelle geheugen is geconstrueerd volgens geheel andere

physische principes: kleine ferriet-kerntjes (ringetjes) worden of in de ene, of in de andere richting gemagnetiseerd; deze twee magnetisatie-richtingen stellen de cijfers 0 of 1 voor. Op deze wijze heeft men voor elk binair cijfer een kerntje nodig. De kerntjes worden geassembleerd in een rechthoekig patroon om de kruispunten van horizontaal en verticaal gespannen draden. Deze vorm heeft aan dit soort geheugen de naam "matrix-geheugen" gegeven.

Dankzij de rechthoekige hystereselus van ferriet is het nu mogelijk om in een matrix een bepaald woord (een rij kerntjes) te selecteren om er te lezen of te schrijven, zonder de magnetische toestand van de andere kerntjes noemenswaard te beinvloeden. In tegenstelling tot de adressen op de trommel, die slechts periodiek beschikbaar zijn, zijn de adressen van het matrix-geheugen instantaan bereikbaar.

In het volledig uitgevoerde project zal de ARMAC beschikken over 16 snelle kanalen, n.l. de kanalen 0 t/m 15; de kanalen 16 t/m 127 staan op de trommel. Het trommelgeheugen is volledig, het matrix-geheugen bestaat voorlopig uit slechts één kanaal, n.l. kanaal 0. De kanalen van 1 t/m 15 ontbreken. Hoewel in de code dus ruimte is voor een geheugen van 4096 woorden, omvat het ARMAC geheugen in feite voorlopig slechts 3616 woorden.

Het snelle geheugen is hier beschreven, als was het bestemd om er getallen in te bergen. In de praktijk wordt het dan ook hoofdzakelijk hiervoor gebruikt (n.l. voor de getallen, die het meest in de berekening voorkomen.) De lezer realisere zich, dat het echter volwaardige adressen zijn, waarvan de woorden ook best als opdrachten(paren) geinterpreteerd kunnen worden, m.a.w. we mogen in het snelle geheugen ook best een stuk programma zetten. Als het snelle geheugen uitgebreid wordt, zal dit steeds belangrijker worden.

4. De buffer

De apparatuur, die een bepaald adres in het geheugen opspoort, wordt om begrijpelijke redenen de selectie genoemd.

Bij de doorsnee-opdrachten treedt de selectie twee maal in werking: eerst als opdrachtselectie, om de opdracht, die uitgevoerd moet worden, te lezen, daarna als getalselectie, omdat tijdens de uitvoering van de opdracht doorgaans een getal van of naar het geheugen getransporteerd moet worden.

Bij getalselectie worden altijd hele woorden getransporteerd, maar beide richtingen zijn mogelijk; bij de opdrachtselectie worden opdrachten, dus halve woorden, getransporteerd, en wel altijd van het geheugen naar dè besturing.

De getalselectie werkt onder controle van het adres in de opdracht, die wordt uitgevoerd; hier immers is aangegeven, tussen welk adres van het geheugen en het arithmetisch orgaan er contact gelegd moet worden. De opdrachtselectie werkt onder controle van de z.g. opdrachtteller. Hierin staat steeds de plaats (d.w.z. het adres benevens de a-b-indicatie), waarvandaan de volgende opdracht aangehaald moet worden. De opdrachtteller bestaat dus uit 13 tweetallige cijfers; de a-b-indicatie (met 0 voor een a-opdracht en 1 voor een b-opdracht) kan het beste opgevat worden als "de eerste binaal achter de komma". Na het lezen van een opdracht wordt $\frac{1}{2}$ bij de inhoud van de opdrachtteller opgeteld. Om de plaats van de volgende opdracht te bepalen (waarbij het geval van de sprongopdracht, die de normale volgorde verstoort even buitengesloten wordt).

Als we de inhoud van de opdrachtteller aanduiden met (T), kunnen we dit dus formuleren als volgt: (T) is achtereenvolgens = n, = n + $\frac{1}{2}$, = n + 1 etc. De opdrachten, die onder controle van deze (T) worden aangehaald, zijn dus de a-opdracht van adres n als (T) = n, de b-opdracht van adres n als (T) = n + $\frac{1}{2}$, de a-opdracht van adres n + 1 als (T) = n + 1, etc.

We zien dus, hoe organisatorisch verwezenlijkt wordt, dat onder voorbehoud van sprongopdrachten de opdrachten in het geheugen in volgorde worden afgewerkt. Het uitvoeren van een sprong wordt door de machine geeffectueerd door (T) te vervangen door de plaats van de nieuwe opdracht.

Ook omdat de inhoud van de opdrachtteller in lichtjes op het bedieningspaneel van de ARMAC zichtbaar is en voor de operateur vaak de belangrijkste brom is van informatie hoever de berekening al gevorderd is, hebben we dit onderdeel iets uitvoeriger beschreven, dan voor de programmeur strikt noodzakelijk is.

Om opdrachten sneller van de trommel te kunnen lezen, is de ARMAC voorzien van een additionele matrix, genaamd de buffer, die, evenals de andere matrices, een capaciteit van één kanaal heeft. Als een opdracht (!) uit het trommelgeheugen

moet worden gelezen, gaat deze selectie niet direct, maar via de buffer. De buffer zal n.l. steeds de copie van het betroffen kanaal op de trommel bevatten: als de opdracht-selectie overschakelt naar een ander kanaal op de trommel, wordt dit nieuwe kanaal automatisch in een omwentelingstijd van de trommel gecopieerd in de buffer en de berekening wordt weer voortgezet.

De tijdswinst die hierdoor op relatief goedkope wijze geboekt wordt, is enorm. Immers zodra het betroffen kanaal in de buffer gecopieerd is, zijn de opdrachten zonder verdere wachttijd beschikbaar. Door bij de opstelling van het programma er een beetje op te letten, dat er voor de opdrachten geen onnodige trommelkanaalwisselingen optreden, kan een aanzienlijke tijdswinst geboekt worden. Als een of ander cyclusje vele malen herhaald moet worden, scheelt het aanzienlijk of de opdrachten van dit cyclusje alle in hetzelfde kanaal staan of b.v. juist over de scheiding van twee kanalen. In het laatste geval duurt elke doorgang door de cyclus twee omwentelingen langer.

De buffer is het beste te vergelijken met de pagina, waarop een boek open ligt. Als wij een boek zouden lezen maar na elk gelezen woord het boek even zouden dichtslaan, om vervolgens voor het volgende woord van de zin de betroffen bladzijde eerst weer op te zoeken, zouden wij even inefficient werken als een ARMAC zonder buffer: voor elke opdracht zouden we het volledige selectiewerk weer moeten verrichten. Omdat het in volgorde afwerken wel regel is voor opdrachten, maar de getallen in de opeenvolging van gebruik vaak tamelijk kris-kras in het geheugen staan, wordt het inlezen van de buffer (het "definitief omslaan van de pagina") beheerst door de opdrachtselectie. De getalselectie beinvloed dus nimmer, welk trommelkanaal in de buffer staat. Moet een getal van een trommeladres gelezen worden, dan impliceert dit een gemiddelde wachttijd van een halve omwenteling. Is het trommeladres echter een uit het kanaal, dat tevens in de buffer gecopieerd staat, dan kiest de machine de onmiddellijk beschikbare overeenkomstige plaats uit de buffer. (Bij het inlezen van de buffer worden 32 woorden gecopieerd: hier kunnen best ook wat getallen meekomen, b.v. de coefficienten van een machtreeks ter berekening van de sinus.)

Het copieren van een nieuw trommelkanaal in de buffer gebeurt alleen als dit nodig is. M.a.w. gaat na een sprongop-

dracht de opdrachtselectie opdrachten uit het snelle geheugen aanhalen, dan blijft de bufferinhoud ongewijzigd. Het is immers zinloos een snel kanaal eerst in de even snelle buffer over te nemen. Springt daarna de opdrachtselectie weer terug naar een opdracht in het oorspronkelijke trommelkanaal, dan wordt de buffer niet opnieuw ingelezen: het betroffen kanaal stond er immers nog in.

Het inlezen van de buffer geschiedt na een sprong naar een nieuw trommelkanaal of na de uitvoering van de laatste opdracht van het vorige kanaal.

Zoals wij reeds terloops vermeldden, geschiedt dit overnemen van het nieuwe trommelkanaal in de buffer automatisch, d.w.z. er bestaat geen aparte "buffer-inleesopdracht", die ingelast zou moeten worden. Dit is een van de elegantste eigenschappen van de ARMAC; wij laten in dit stadium van ons betoog de volle appreciatie hiervan aan de meer gevorderde programmeurs.

Als door een schrijfopdracht een woord op de trommel wordt ingevuld op een adres, dat toevallig behoort tot het kanaal dat zich op dat moment in de buffer bevindt, dan wordt dit woord ook ingevuld op de overeenkomstige plaats in de buffer, m.a.w. een wijziging van het origineel wordt hier in de copie bijgehouden.

Tot zover hoeft de programmeur niet beslist van het bestaan van de buffer af te weten. Wie echter, zich van de buffer onbewust, voor de ARMAC programmeert, zal wel merken, dat zijn programma's veel minder efficient zijn, dan die van anderen.

Er bestaan echter speciale bufferschrijfopdrachten. Zij vullen een woord in op een plaats in de buffer, maar niet op de overeenkomstige plaats op de trommel. Zij kunnen dus zonder eventuele wachttijd onmiddellijk uitgevoerd worden en zijn dus door hun snelheid niet zonder charme. Wie echter van deze bufferschrijfopdrachten gebruik maakt, doet dit "voor eigen risico"! Hij realisere zich het volgende goed:

a. Van nu af aan kan de inhoud van de buffer afwijken van die van het overeenkomstige kanaal op de trommel. Nu is het dus van het hoogste belang zich bewust te zijn, dat getallen op trommel-adressen "zo mogelijk" uit de overeenkomstige bufferplaats gelezen worden. Het oorspronkelijke woord op de trommel is voorlopig onbereikbaar!

De informatie, die in de buffer wordt geschreven, gaat onherroepelijk verloren, zodra opdrachten uit een ander trommelkanaal geselecteerd worden. De buffer is hier dus een tijdelijke uitbreiding van het snelle geheugen.

Deze waarschuwingen zijn met opzet in nogal krasse bewoordingen vervat, opdat de programmeur bij het gebruik hiervan (zo mogelijk) nog meer op zijn qui vive is dan bij de opstelling van de rest van het programma. Anderzijds willen wij niet suggereren, dat het "nare" opdrachten zouden zijn. Integendeel, de faciliteit om alleen in de buffer te kunnen schrijven, is doelbewust in het ontwerp opgenomen. De geboden voorzichtigheid is de prijs, die men moet betalen voor de winst, die deze faciliteit in andere situaties oplevert. Welke de situaties zijn, waarvoor de bufferschrijfopdrachten gemaakt zijn, zullen wij later tegenkomen.

Het arithmetisch orgaan

De antwoorden van rekenkundige bewerkingen worden in de ARMAC achtergelaten in een van de registers van het arithmetisch orgaan. Als dit antwoord als tussenresultaat onthouden moet worden, moet de arithmetische opdracht gevolgd worden door een schrijfopdracht.

Het arithmetisch orgaan omvat twee registers, ter onderscheiding het A-register en het S-register genaamd, die elk de capaciteit van één woord hebben. Met betrekking tot operaties, die slechts één register betreffen, zijn deze registers volledig symmetrisch: "wat in A kan, kan ook in S".

Voordat enkele kleine stukjes programma toelichten, hoe het arithmetisch orgaan alzo gebruikt wordt, zullen wij enige notaties afspreken. De inhoud van registers of adressen wordt met haakjes aangegeven, dus b.v. (A), (S), (2 X 0) of (10 A 7), voor resp. de inhoud van het A-register, van het S-register, van adres 2 X 0 en van adres 10 A 7. (Opm.: Voor het laatste adres nemen we aan, dat gedefinieerd is, welk kanaal met A 0 overeenkomt; A 7 ligt dan 7 kanalen verder. Adres 2 X 0 is plaats 2 van kanaal 0; de betekenis van de sluitletter X is altijd dezelfde.) Voorts zullen we gebruik maken van een nieuw symbool "het gerichte gelijkteken" $\Rightarrow$ (lees "vervangt") waarmede wat rechts ervan staat, gedefinieerd wordt in termen van wat links staat.

Als b.v. de som van de getallen op 10 B 0 en 11 B 0 op 12 B 0 geplaatst moet worden, noteren wij dit

(10 B 0) + (11 B 0) ⇒ (12 B 0)

(Lees: " de inhoud van 10 B 0 + de inhoud van 11 B 0 vervangt de inhoud van 12 B 0").

Om dit te programmeren, hebben wij drie opdrachten nodig. Immers elke opdracht refereert slechts naar één adres. Het programma, dat deze bewerking met gebruikmaking van het A-register uitvoert, luidt b.v.:

```
2 10 B 0:          (10 B 0) ⇒ (A)
0 11 B 0: (A) + (11 B 0) ⇒ (A)
4 12 B 0:          (A) ⇒ (12 B 0)
```

Wij hebben hier van de volgende drie opdrachten gebruik gemaakt.

De 2-opdracht "leest schoon in A". d.w.z. met vernietiging van de vorige inhoud van A wordt de inhoud van de aangegeven geheugenplaats in A overgenomen.

De 0-opdracht "leest additief in A" d.w.z. de inhoud van de aangegeven geheugenplaats wordt bij de inhoud van A opgeteld, de som wordt in A achtergelaten.

De 4-opdracht is de schrijf-opdracht, waardoor de inhoud van A gecopieerd wordt op de aangegeven geheugenplaats.

(Opm.: In de explicatie naast het stukje programma staat steeds, welk woord vervangen wordt. "Verzwegen" woorden blijven gelijk. Zo laat de schrijfopdracht de inhoud van A ongewijzigd in A achter; eveneens blijft door deze drie opdrachten de inhoud van S onveranderd.)

Het hier gestelde doel had op vele andere wijzen bereikt kunnen worden, b.v. men had eerst (11 B 0) in A kunnen zetten en daarna (10 B 0) in A optellen; men had ook, door andere functie-cijfers te gebruiken, de berekening in het S-register kunnen uitvoeren, daarbij (A) onaangetast latend.

Als voorbeeld van een operatie, waarin beide registers gebruikt worden, bekijken we nu een vermenigvuldiging en wel in zijn eenvoudigste vorm, n.l. de 18-opdracht. Om te kunnen vermenigvuldigen, moet eerst een van beide factoren in S staan. Daarna komt de 18-opdracht, waarvan het adres bepaalt, waar de andere factor gelezen wordt. Aangezien beide factoren bestaan uit teken, gevolgd door maximaal 33 significante binaire cijfers, bestaat het product uit een teken en maximaal 66 significante cijfers. Deze cijfers van het product worden nu verdeeld over de beide registers: de hoogste 33

productcijfers voorafgegaan door het teken van het product, worden in A afgeleverd, de laagste 33 productcijfers, eveneens door het teken van het product voorafgegaan, komen in S terecht. Voor dit dubbellengte woord gebruiken we het symbool (AS); dit is dus alleen van toepassing, als A en S hetzelfde teken hebben.

In het volgende duiden we met n een willekeurig adres aan. De voorlopige formulering van de vermenigvuldiging luidt dan

18 n: (n) x (S) ⇒ (AS)

Nu is echter het ogenblik gekomen, om opnieuw de interpretatie van het woord als getal onder de loupe te nemen.

Tot nog toe hebben we getallen beschouwd als gehele getallen, d.w.z. de komma wordt geacht te staan na het laagste (= minst significante) cijfer. Het woord bestaat echter uitsluitend uit een cijferrij met teken, waarbij de komma als zodanig niet geborgen wordt. Waar de komma gedacht wordt is een kwestie van interpretatie. Willen we specificeren, dat het woord (n) als geheel getal beschouwd wordt, dan noteren we [n] ; een nauwkeuriger definitie van de vermenigvuldiging in de ARMAC luidt dan ook

18 n: [n] x [S] ⇒ [AS],

waar $[AS] = [A].2^{33} + [S]$. Immers A bevat de 33 "kopcijfers" van het product, waar achter nog de laatste 33 laagste cijfers van het product in S komen, voordat de komma komt. Dat het eenhedencijfer van [A] overeenkomt met het aantal 2^{33}-vouden in het product [AS] wordt in bovenstaande definitie van [AS] juist uitgedrukt.

De andere gebruikelijke interpretatie van een woord is als breuk, d.w.z. hier wordt de komma geacht onmiddellijk op het tekencijfer te volgen. Deze interpretatie van het woord (n) wordt met {n} aangegeven. De samenhang met de interpretatie als geheel getal is kennelijk $\{n\} = [n].2^{-33}$

Analoog kent het dubbellengte-woord (AS) drie interpretaties. We zagen reeds

$$[AS] = [A].2^{33} + [S],$$

met de komma aan de lage kant van S. Interpreteren we de komma tussen de beide registers, dan hebben we dus door 2^{33} gedeeld; we noteren dit

$$\{AS\} = 2^{-33}[AS] = [A] + \{S\}.$$

Schuiven we de komma nog eens 33 plaatsen naar links, dan noteren we

$$\{AS\} = 2^{-33}[AS] = \{A\} + 2^{-33}\{S\},$$

waar de komma dus aan de hoge kant van A gedacht wordt. De drie aequivalente omschrijvingen van de vermenigvuldiging luiden nu

$$[n] \times \{S\} = \begin{matrix} [n] \times [S] \Rightarrow [AS] \\ \{n\} \times [S] \Rightarrow [AS\} \\ \{n\} \times \{S\} \Rightarrow \{AS\} \end{matrix}$$

Of in woorden:
(geheel getal) x (geheel getal) geeft een antwoord in S: als dit in absolute waarde minstens gelijk aan 2^{33} is, staan de kopcijfers in A, (anders komt +0 of -0 in A!);
(breuk) x (geheel getal) geeft van het antwoord geheel gedeelte in A, breukgedeelte in S;
(breuk) x (breuk) geeft het antwoord in A; in S staan de door afronding doorgaans grotendeels onbetrouwbaar volgende 33 cijfers, die nog voor de afronding van $\{AS\}$ op 33 binalen achter de komma gebruikt kunnen worden.

Dat deze uitweiding over de plaats van de komma pas ter sprake is gekomen bij de vermenigvuldiging, heeft een aanwijsbare mathematische achtergrond. De komma geeft aan, welke binaal het eenhedencijfer is. Nu is de eenheid wiskundig m.b.v. de vermenigvuldiging gedefinieerd, n.l. doordat vermenigvuldigen met 1 elk getal onveranderd laat. Het getal 0, dat elk willekeurig getal na optelling onveranderd laat, is dus al gedefinieerd met betrekking tot de additieve algebraische bewerkingen. Nu we echter de komma hebben ingevoerd, moet de omschrijving van de optelopdracht gecomplementeerd worden met de evidente restrictie, dat in beide addenda de komma op dezelfde plaats staat: de komma's in beide getallen moeten bij de optellingen immers onder elkaar komen. Wij blijven hierbij in de beschrijving van de opdrachtencode van ronde haakjes gebruik maken. Met (A) + (n) wordt dus b.v. $[A] + [n]$ of $\{A\} + \{n\}$ bedoeld; in het antwoord wordt de komma op dezelfde plaats geinterpreteerd als in de addenda.

Het derde register van het arithmetisch orgaan is maar een heel kleintje, n.l. het z.g. conditie-register C, dat plaats biedt aan een tekencijfer. Als (C) = 0, resp. = 1 is zeggen we, dat de conditie positief, resp. negatief is. De meeste opdrachten doen niets met de conditie. Sommige opdrachten, genaamd de "uit-opdrachten", omdat de schrijfopdrachten er onder vallen, zetten de conditie, d.w.z. een bepaald tekencijfer wordt (eventueel invers) in het conditie-register over-

genomen. Sommige opdrachten reageren op de conditie: zij laten hem daarbij onveranderd. Deze opdrachten worden afhankelijk van de conditie uitgevoerd of geskipt.
Het zijn:

de "+conditionele stopopdracht": als de conditie positief is, stopt de machine, anders wordt de opdracht geskipt, d.w.z. de machine gaat "gewoon door" (de opdrachtteller wordt met $\frac{1}{2}$ opgehoogd, de volgende opdracht wordt aangehaald, etc.)

de "-conditionele stopopdracht": als de conditie negatief is, stopt de machine, anders wordt de opdracht als skip geinterpreteerd.

de "+ conditionele sprongopdrachten": als de conditie positief is, wordt de sprong uitgevoerd, anders wordt de opdracht geskipt.

De conditionele sprongopdrachten zijn verreweg de belangrijkste. Omdat deze altijd "+ conditioneel zijn" wordt het zetten van de conditie genoteerd door een vraag. Als het antwoord op deze vraag bevestigend is, wordt de conditie positief gezet (er wordt wel gesprongen), is het antwoord ontkennend, dan wordt de conditie negatief gezet. Bij de formulering van deze vraag wordt +0 geacht groter te zijn dan -0.

Opm.: Alle schrijfopdrachten zetten de conditie. In de explicatie naast het programma wordt de vraag dan alleen genoteerd als deze conditie inderdaad gebruikt wordt. Tot aan de betroffen conditionele opdrachten mogen dus geen andere conditiezettende opdrachten voorkomen!

6. De opdrachtencode

De opdrachten, waarin de laagste twaalf cijfers een echt adres voorstellen, worden in het onderstaande genoteerd als f/n.

Waar een uitdrukking met een vraagteken staat, wordt de conditie positief gezet, als het antwoord op de vraag bevestigend is en negatief, als het antwoord ontkennend is. De inhoud van een register of geheugenplaats is hier de inhoud na de operatie.

Voor de omschrijvingen der arithmetische verrichtingen wordt weer gebruik gemaakt van het vervangingssymbool $\Rightarrow$.

0/n (A) + (n) ⇒ (A)
1/n (A) - (n) ⇒ (A)
2/n + (n) ⇒ (A)
3/n - (n) ⇒ (A)
4/n + (A) ⇒ (n); (n) ≥ + 0? of (A) ≥ + 0?
5/n - (A) ⇒ (n); (n) ≥ + 0? of (A) ≤ - 0?
6/n spring naar de a-opdracht van adres n
7/n spring naar de b-opdracht van adres n

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 de conditie positief is, spring dan naar de a-opdracht van adres n; anders skip
15/n als de conditie positief is, spring dan naar de b-opdracht van adres n; anders skip.

16/n [A] + [n].[S] ⇒ [AS]
17/n [A] - [n].[S] ⇒ [AS]
18/n + [n].[S] ⇒ [AS]
19/n - [n].[S] ⇒ [AS]
20/"n" transporteer track van langzaam geheugen naar snel.
21/"n" transporteer track van snel geheugen naar langzaam.
22/n plaats link in A en spring naar de a-opdracht van adres n
23/n plaats link in A en spring naar de b-opdracht van adres n

24/...
25/...
26/...
27/...
28/...
29/...
} Schuif- en communicatieopdrachten.

ad "20" en "21".

Dit zijn de z.g. tracktransporten. Zij laten de conditie onbeinvloed en gebruiken de registers A en S niet. Er zijn steeds twee kanalen bij betrokken, n.l. een kanaal (matrix) van het snelle geheugen en een kanaal (trommelspoor) van het langzame. De inhoud van het ene kanaal wordt in het andere gecopieerd, waarbij het functiegedeelte f = 20, resp. f = 21 de richting van het transport bepaalt. Het numeriek gedeelte "n" van de opdracht bepaalt als volgt op welke kanalen

het transport betrekking heeft: de hoogste zeven cijfers van het "adres" bepalen het spoor op de trommel (het getal, dat door deze zeven cijfers apart beschouwd gevormd wordt, is dus minstens 16), terwijl de laagste vier cijfers het nummer van het snelle kanaal bepalen. Hier tussen staat een ongebruikt cijfer. Omdat voorlopig kanaal X 0 het enige kanaal van het snelle geheugen is, eindigt het numeriek gedeelte der opdracht dus op vijf nullen. Om b.v. de inhoud van het kanaal A 7 in het snelle kanaal X 0 te copieren, geeft men dus de opdracht:

20 0 A 7.

Opm.: De buffer wordt bij deze opdracht noch gebruikt, noch beschreven. Het transport is dus altijd van resp. naar de trommel. Naast de buffer-schrijfopdracht biedt ook deze opdracht de mogelijkheid om de buffercopie van het origineel op de trommel te doen verschillen: men wijzigt met één opdracht het origineel (waarschijnlijk) volslagen. Een en ander is zo drastisch, dat deze eigenschap van de tracktransport wel van puur academisch belang zal zijn.

ad "22" en "23"

Dit zijn de z.g. subroutineaanroepen. Het zijn sprongopdrachten, die dus n (benevens de aan het functiegedeelte ontleende a-b-indicatie) in de opdrachtteller plaatsen. Voordat dit echter gebeurt - dus voordat de sprong werkelijk wordt uitgevoerd - wordt de oude inhoud van de opdrachtteller gelezen ("gered"); met behulp hiervan wordt in de a-helft van het A-register de 6-of 7-sprong naar de onmiddellijk volgende opdracht geplaatst, terwijl in de b-helft van het A-register 17 nullen worden gezet. Als de subroutinesprong uitgevoerd wordt, bevindt zich dus in het A-register de sprongopdracht naar de opdracht, die aan de beurt geweest zou zijn, ware de subroutineaanroep geen sprong geweest. Deze sprongopdracht in het A-register wordt de koppelopdracht of de link genoemd; hoe deze benut wordt om later de hoofdberekening voort te zetten op het punt, waar hij door de subroutinesprong werd onderbroken, zullen wij later zien.

ad "24" t/m "29"

Bij de schuif- en communicatieopdrachten is het numeriek gedeelte nooit een adres, maar specificeert het anderszins de verrichting van de opdracht. Onder deze groep zijn alle bewerkingen ondergebracht, die niet naar geheugenplaatsen refereren. Wij noteren hier de opdrachten volgens de standaard

ponsconventies.

Nultest op de ongetekende nul

28 0 X 0 : A ≠ 0 ?
29 0 X 0 : A = 0 ?

28 0 X 8 : S ≠ 0 ?
29 0 X 8 : S = 0 ?

Deze opdrachten zetten de conditie op het al of niet gelijk aan nul zijn van één van beide registers. Zij maken geen onderscheid tussen + 0 en - 0.

Handregisteropdrachten

Deze opdrachten komen uitsluitend in het communicatieprogramma voor; wij noemen ze hier om der wille van de volledigheid. Voor het woord (H) bestaan slechts veertien mogelijkheden 0,1, ... 9,10,11,12 of 13, de laatste vier met de speciale toetsbenamingen +, -, +. en -. .

24 1 X 0	(A) + (H) ⇒ (A)
25 1 X 0	(A) - (H) ⇒ (A)
26 1 X 0	+ (H) ⇒ (A)
27 1 X 0	- (H) ⇒ (A)
24 1 X 8	(S) + (H) ⇒ (S)
25 1 X 8	(S) - (H) ⇒ (S)
26 1 X 8	+ (H) ⇒ (S)
27 1 X 8	- (H) ⇒ (S)

f = 28 en f = 29 komt bij de handregisteropdrachten niet voor.

Getalschakelaarsopdrachten

Op het bedieningspaneel van de ARMAC is een rij van 3^4 schakelaars met twee standen: naar beneden stelt het cijfer 0 voor, naar boven het cijfer 1.
Het woord door deze 3^4 schakelaars voorgesteld, wordt met (G) aangeduid. Dit wordt door de machine het arithmetisch orgaan ingevoerd door de volgende opdrachten:

24 2 X 0	(A) + (G) ⇒ (A)
25 2 X 0	(A) - (G) ⇒ (A)
26 2 X 0	+ (G) ⇒ (A)
27 2 X 0	- (G) ⇒ (A)

24 2 X 0	(S) + (G) ⇒ (S)
25 2 X 0	(S) - (G) ⇒ (S)
26 2 X 0	+ (G) ⇒ (S)
27 2 X 0	- (G) ⇒ (S)

f = 28 en f = 29 komt bij de getalschakelaars-opdrachten niet voor.

Bandlees- en -ponsopdrachten

Deze opdrachten zullen buiten het standaard-communicatiepro-gramma niet veel voorkomen. Wie echter aan de faciliteiten van dit programma niet genoeg heeft, zal ze wel moeten gebruiken. Deze groep omvat de opdrachten:

24 4 X 0	(A) + (B) ⇒ (A)
25 4 X 0	(A) - (B) ⇒ (A)
26 4 X 0	+ (B) ⇒ (A)
27 4 X 0	- (B) ⇒ (A)
28 4 X 0	+ (A) ⇒ (U) ⇒ (B); (A) ≥ + 0 ?
29 4 X 0	- (A) ⇒ (U) ⇒ (B); (A) ≤ - 0 ?
24 4 X 8	(S) + (B) ⇒ (S)
25 4 X 8	(S) - (B) ⇒ (S)
26 4 X 8	+ (B) ⇒ (S)
27 4 X 8	- (B) ⇒ (S)
28 4 X 8	+ (S) ⇒ (U) ⇒ (B); (S) ≥ + 0 ?
29 4 X 8	- (S) ⇒ (U) ⇒ (B); (S) ≤ - 0 ?

In de beide eerste vier regels staan de bandleesopdrachten; het symbolische register B bevat aan de hoge kant 29 nullen, de laagste 5 cijfers worden van de telexband gelezen, die daarna een pentade opschuift.

In de beide onderste regels staan de bandponsopdrachten: al of niet met tekenwisseling (d.w.z. 0-1 inversie) worden de laagste zes cijfers van A of S in een tussenregister U overgenomen; de laagste vijf cijfers van U gaan door naar de ponser. (Het "uit-register U" heeft zes cijfers, omdat ditzelfde register ook voor het typen gebruikt wordt en er meer dan 32 verschillende signalen naar de schrijfmachine gestuurd kunnen worden.) Als "uit-opdracht" zetten de ponsopdrachten tevens de conditie.

De typ- en terugleesopdrachten

Ook deze opdrachten zullen voornamelijk in het standaard-communicatieprogramma voorkomen.

24 8 X 0	(A) + (U) ⇛ (A)
25 8 X 0	(A) - (U) ⇛ (A)
26 8 X 0	+ (U) ⇛ (A)
27 8 X 0	- (U) ⇛ (A)
28 8 X 0	+ (A) ⇛ (U); (A) ⩾ + 0 ?
29 8 X 0	- (A) ⇛ (U); (A) ⩽ - 0 ?
24 8 X 8	(S) + (U) ⇛ (S)
25 8 X 8	(S) - (U) ⇛ (S)
26 8 X 8	+ (U) ⇛ (S)
27 8 X 8	- (U) ⇛ (S)
28 8 X 8	+ (S) ⇛ (U); (S) ⩾ + 0 ?
29 8 X 8	- (S) ⇛ (U); (S) ⩽ - 0 ?

In de beide onderste twee regels staan de typopdrachten: de al of niet geinverteerde laagste zes cijfers van A of S worden in hetzelfde register U gecopieerd, dat ook bij het ponsen wordt gebruikt. De inhoud van U bepaalt dan welk symbool door de electrische schrijfmachine getypt wordt.

Onderstaande tabel geeft aan de correlatie tussen (U) en het getypte symbool. Hier zijn steeds twee mogelijkheden, afhankelijk of de schrijfmachine ingesteld staat op hoofdletters dan wel kleine letters. Deze stand kan door speciale typopdrachten, n.l. met (U) = 18 en (U) = 19 gezet worden. Normaal staat de schrijfmachine op kleine letter! Elk programma, dat de schrijfmachine in hoofdletterstand zet, zette hem weer zo snel mogelijk terug op kleine letters.

In beide eerste vier regels staan terugleesopdrachten, ingebouwd om controle op het typen mogelijk te maken. (Ook bij het ponsen zouden deze opdrachten gebruikt kunnen worden; daar is deze controle echter onvoldoende.)

(U)	kl.	HL	(U)	kl.	HL
0	0	½	24	d	D
1	1	"	25	e	E
2	2	½	26	f	F
3	3	£	27	g	G
4	4	$	28	h	H
5	5	%	29	i	I
6	6	ƒ	30	j	J
7	7	&	31	k	K
8	8	(	32	l	L
9	9	)	33	m	M
10	Tab		34	n	N
11	TWNR		35	o	O
12	-	_	36	p	P
13	+	=	37	q	Q
14	.	;	38	r	R
15	,	?	39	s	S
16	/	:	40	t	T
17	'	"	41	u	U
18	HL		42	v	V
19	kl		43	w	W
20			44	x	X
21	a	A	45	y	Y
22	b	B	46	z	Z
23	c	C	56 t/m 63	Spatie	

(U) = 20 en (U) = 47 t/m 55 zijn niet aangesloten. (U) = 56 t/m 63 geven alle een spatie. Deze spatie wordt gegeven, ongeacht de stand van de schrijfmachine, evenals Tab (Tabuleer) en TWNR (Terug Wagen en Nieuwe Regel) en HL en kl.

De conditionele stopopdrachten

26 16 X 0 of } stop, als de conditie positief is; anders
26 16 X 8 } skip.

27 16 X 0 of } stop, als de conditie negatief is; anders
27 16 X 8 } skip.

Beide + conditionele stopopdrachten zijn aequivalent; hetzelfde geldt voor de - conditionele stopopdrachten.

De bufferschrijfopdrachten

28 n X 2 + (A) ⇒ plaats n v.d. buffer; (A) ≥ + 0 ?
29 n X 2 - (A) ⇒ plaats n v.d. nuffer; (A) ≤ - 0 ?

28 n X 0 + (S) ⇒ plaats n v.d. buffer; (S) ⩾ + 0 ?
29 n X 0 - (S) ⇒ plaats n v.d. buffer; (S) ⩽ - 0 ?

In overeenstemming met de capaciteit van de buffer geldt hier voor 0 ⩽ n ⩽ 31. f = 24 t/m 27 komt hier niet voor, omdat speciale bufferleesopdrachten niet bestaan: leesopdrachten met het overeenkomstig trommeladres halen immers het woord al uit de buffer:

De adresloze optelling

Om kleine gehele getallen n (0⩽n⩽127) in een van de registers op te tellen etc. is het niet noodzakelijk, voor dit getal een adres te reserveren. Er bestaan n.l. speciale opdrachten voor kleine addenda, waar het addendum direct in de laagste zeven plaatsen van het "adresgedeelte" van de opdracht is aangegeven. Vandaar dat deze addenda kleiner moeten zijn dan $128 = 2^7$.

24 n X 4	(A)	+ n ⇒	(A)	0 ⩽ n ⩽ 127
25 n X 4	(A)	- n ⇒	(A)	"
26 n X 4		+ n ⇒	(A)	"
27 n X 4		- n ⇒	(A)	"
24 n X 2	(S)	+ n ⇒	(S)	0 ⩽ n ⩽ 127
25 n X 2	(S)	- n ⇒	(S)	"
26 n X 2		+ n ⇒	(S)	"
27 n X 2		- n ⇒	(S)	"

f = 28 en f = 29 komen bij deze groep niet voor.

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 van het circuit weer binnen.

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

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

f = 29, de negatieve uit-schuif: de cijfers worden rondgeschoven als bij f = 24, maar tevens wordt de conditie gezet op het inverse van het nieuwe tekencijfer van het circuit.

Het aantal cijferposities, waarover geschoven wordt, geeft men aan in het plaatsgedeelte van de opdracht; dit aantal is maximaal = 35, minimaal = 0, behalve voor de uitschuiven, die minimaal over 1 plaats schuiven.

Het einde van de opdracht, dus (de sluitletter X en) de kaalcorrectie - zo heet het getal, dat op de sluitletter volgt - bepaalt het circuit. Het begin, dat steeds het tekencijfer van een register is, fungeert tevens als tekencijfer van het circuit. Er zijn vier mogelijkheden:

X 20: A→A: het circuit begint bij het tekencijfer van A en eindigt aan de lage kant van A.

X 22: 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; aan de lage kant van A eindigt het circuit. M.a.w. "schuif S naar A door het teken van A heen".

X 28: A→S: het circuit begint bij het tekencijfer van A; aan de lage kant van A wordt het voortgezet met het op één na hoogste cijfer van S; aan de lage kant van S eindigt het circuit. Het tekencijfer van S blijft ongewijzigd. M.a.w. "schuif A naar S onder het tekencijfer van S door".

X 30: S→S: het circuit begint bij het tekencijfer van S en eindigt aan de lage kant van S.

Enige karakteristieke voorbeelden mogen het gebruik van de schuifopdrachten toelichten.

24 34 X 22: schuif S add. naar A over 34 plaatsen. De cijfers van S komen juist in A; omdat de schuif additief is, komen de cijfers van A juist in S, m.a.w. A en S wisselen van inhoud.

26 1 X 30: schuif S schoon over één plaats, dus $\frac{1}{2}(S) \neq (S)$; de halvering is niet afgerond. Waarom niet 24 1 X 30?

24 31 X 20: schuif A over 31 plaatsen rond. Als we 34 plaatsen hadden geschoven, kwam elk cijfer weer terug in zijn oorspronkelijke positie. Deze schuif, die drie plaatsen minder naar rechts schuift, is dus ook te interpreteren als: schuif drie plaatsen naar links. Onder voorbehoud van capaci-

teitsoverschrijding luidt dus de functie: $2^3.(A) \nrightarrow (A)$.

26 29 X 28: $2^4\{AS\} \nrightarrow [AS]$ (N.B. $2^4\{AS\}$ en niet $2^5\{AS\}$!)

Ook het testen van het teken doen we met schuifopdrachten: b.v.

28 34 X 30 (S) $\geq$ + 0 ?
29 34 X 20 (A) $\leq$ - 0 ?

7. Illustraties van het gebruik van de opdrachtencode

Zoals wij hebben gezien, komt in de opdrachtencode geen deelopdracht voor. De bespreking, hoe quotienten dan wel berekend worden, stellen wij nog even uit en in de volgende voorbeeldjes ter illustratie van de opdrachtencode zullen dus geen delingen voorkomen. In het volgende voorbeeldje zijn de adressen voor de getallen gekozen in het snelle kanaal.

Gevr. (3 X 0) + (4 X 0) - (5 X 0) $\nrightarrow$ (8 X 0)

2	3	X	0
0	4	X	0
1	5	X	0
4	8	X	0

(3 X 0) $\nrightarrow$ (A)
(A) + (4 X 0) $\nrightarrow$ (A)
(A) - (5 X 0) $\nrightarrow$ (A)
(A) $\nrightarrow$ (8 X 0).

Een praktisch aequivalent programma is:

11	3	X	0
9	4	X	0
8	5	X	0
13	8	X	0

Wat is het verschil, behalve dat hier de berekening in het andere register is uitgevoerd?

Voor de verandering zullen wij in het volgende voorbeeld opereren op getallen op de trommel. Laat het kanaal waarin zij staan, de naam H3 gekregen hebben. De gestelde opgave kan dan luiden:
$[3\ H\ 3].[8\ H\ 3] \nrightarrow [25\ H\ 2]$
Het gaat hier dus om de vermenigvuldiging van gehele getallen; blijkens de vraag het antwoord op een adres in te vullen, moet uit anderen hoofde bekend zijn, dat het product in absolute waarde kleiner is dan 2^{33}.

10	3	H	3
18	8	H	3
12	25	H	2

Welke hiermede arithmetisch aequivalente stukjes programma zijn er al zo nog meer?

Ging het er om, het product van de breuken $\{5\ H\ 2\}$ en $\{7\ H\ 3\}$ te schrijven op adres 2 X 0, dan wordt dit verricht door

(niet afgerond)

10	5	H	2
18	7	H	3
4	2	X	0

(wel afgerond)

10	5	H	2
18	7	H	3
4	2	X	0
26	32	X	30
8	2	X	0
12	2	X	0

De bovenstaande stukjes programma moeten ergens in het geheugen staan; dit hebben we niet expliciet aangegeven. Anders is dit bij het volgende voorbeeld, waar gevraagd wordt $\{S\}$ zovaak te verdubbelen, totdat de inhoud van S minstens $= \frac{1}{2}$ is. Dus in formule: gegeven $q > 0$; nu is $q' = 2^n q$; door $\frac{1}{2} \leqslant q' < 1$ is n bepaald.
Gevraagd het programma, dat als $q = \{2X0\}$ uitvoert:
$q' \Rightarrow\!\!\!| \{S\}$ en $n \Rightarrow\!\!\!| [A]$.
We nemen aan, dat het programma begint op de a-opdracht van OA1

kan. A1:

	0	10	2	X	0	$q \Rightarrow\!\!\!| \{S\}$
		26	0	X	4	$0 \Rightarrow\!\!\!| [A]$
	1	6	2	A	1	⇒ inconditioneel naar a2A1
b2 ⇒		24	1	X	4	$[A] + 1 \Rightarrow\!\!\!| [A]$
a1 →	2	28	33	X	30	"$2\{S\}$" $\Rightarrow\!\!\!| \{S\} \geqslant +0$?
		15	1	A	1	→ conditioneel naar b1A1
	3	24	1	X	30	"$\frac{1}{2}\{S\}$" $\Rightarrow\!\!\!| \{S\}$
		..	..	..	..	

Over de "opmaak" van dit stukje programma het volgende: rechts van de vier kolommen staan dubbele pijltjes achter inconditionele sprongen, enkele achter conditionele; links van de vier kolommen worden dubbele pijltjes gezet bij opdrachten waar de besturing alleen maar via sprongopdrachten kan komen, enkele pijltjes bij opdrachten, die of na een sprong, of na de vorige opdracht aan de beurt zijn. Bij deze pijltjes zetten wij tevens waarvandaan de sprongen uitgevoerd worden. Is dit een (halve) plaats uit hetzelfde kanaal, dan wordt met de plaatsaanduiding alleen volstaan.

8. Subroutines

Een subroutine is een reeks opdrachten, die tezamen een of andere duidelijk isoleerbare bewerking verrichten, annex de faciliteit om deze bewerking op elk gewenst punt van het programma in te lassen. Hiertoe zijn de subroutinesprongen (f = 22 en f = 23, zie aldaar) speciaal ingebouwd. Op het moment, dat de sprong naar de subroutine wordt uitgevoerd staat in de a-helft van het A-register immers de link, d.w.z. een cijferrij, die als instructie beschouwd de sprongopdracht voorstelt naar de op de subroutinesprong volgende opdracht. De subroutine begint nu met (A) te schrijven op een voor de link gereserveerd adres aan het einde van de subroutine. Als aan de hand van de opdrachten in de subroutine de betroffen standaardbewerking is uitgevoerd, ontmoet de besturing de opdracht, de link n.l., die op de gereserveerde plaats achteraan was neergeschreven. De link is een sprongopdracht en de berekening wordt dus weer voortgezet, te beginnen op het punt, waar hij door de subroutinesprong was onderbroken.

Een aantal subroutines is permanent in het geheugen van de ARMAC aanwezig. Wij noemen er op het ogenblik vier.

De vierkantswortel uit een breuk. (netto tijdsduur*) ± 80 msec)

Aanroep: 22 0 X 27 =) : $\sqrt{|\{S\}|} \Rightarrow \{S\}$

N.B. De subroutine maakt gebruik van de adressen 0 X 0 en 1 X 0 uit het snelle geheugen.

Betroffen subroutine vervangt de inhoud van S dus door de wortel uit de (als breuk opgevatte) absolute waarde.

Om dus a = $\sqrt{b}$ te berekenen, als b.v. b al staat op 12 C 0 en a ingevuld moet worden op 13 C 0 zet men in zijn programma:

10	12	C	0
22	0	X	27
12	13	C	0

=)

Het stompe pijltje wordt altijd gezet achter de subroutinesprong. Links van de volgende opdracht (hier de 12-opdracht) wordt geen dubbel pijltje meer gezet. De opdracht 22 0X 27 fungeert immers in het programma als de opdracht "trek de wortel uit S", en het is voor de programmeur, alsof de code met deze opdracht is uitgebreid.

*) Netto tijdsduur wil zeggen: exclusief het inlezen van de buffer, dat van het aanroepen en terugkomen het gevolg kan zijn.

De sinus en de cosinus. (netto tijdsduur: 47 msec)
1ste Aanroep 22 0 X 29 =) : sin $\{S\}\pi \Rightarrow \{S\}$
2de Aanroep 23 0 X 29 =) : cos $\{S\}\pi \Rightarrow \{S\}$.
Het argument wordt in S dus meegegeven in eenheden π radiaal; het bestrijkt dus alle vier de quadranten (met uitzondering van $\pm\pi$). Het antwoord is maximaal 2p ($p = 2^{-33}$) onnauwkeurig. Dankzij deze onnauwkeurigheid blijven de antwoorden in absolute waarden onder de 1.

De exacte deling (netto tijdsduur $\pm$ 80 msec)
Aanroep 22 0 X 30 =)
Functie: $[0X0] \cdot 2^{33} + [1X0]$ wordt door $[S]$ gedeeld. Het quotient wordt in S, de rest wordt in A geplaatst; de rest is gedefinieerd als de rest met minimum absolute waarde en het teken van het deeltal. Beide helften van het deeltal moeten hetzelfde teken hebben. Het quotient moet in absolute waarde kleiner zijn dan 2^{33}.
N.B. (0X0) en (1X0) worden door de subroutine gebruikt en gewijzigd.

De breukendeling (netto tijdsduur: $\pm$ 65 msec)
Aanroep: 22 0 X 31 =) : $\{0X0\}/\{S\} \Rightarrow \{S\}$.
Evenals de exacte deling gebruikt en wijzigt deze subroutine (0X0) en (1X0); evenzo wordt weer de deler in S meegegeven en het quotient weer in S afgeleverd. Omdat het quotient als breuk wordt beschouwd, wordt hier geen rest bepaald. Het quotient moet in absolute waarde kleiner zijn dan 1 zijn.

Opm.: De naam "breukendeling" is ontleend aan de meest gebruikelijke interpretatie. De subroutine is echter ook bruikbaar om een geheel getal door een (in absolute waarde groter) geheel getal te delen, dus $[0X0] / [S] = \{S\}$.

De inhoud van de kanalen 16 t/m 31 is immers dezelfde. Het is normaal niet mogelijk, om op deze kanalen te schrijven. Hierboven hebben wij vier standaardsubroutines beschreven, die in dit geblokkeerde stuk geheugen zijn ondergebracht. De rest hiervan is gevuld met programma's en subroutines ten dienste van de communicatie (bandlezen, -ponsen en typen.) Zij komen later aan de orde.

9. De ponsconventies

Het standaardcommunicatieprogramma verzorgt de invoer en de uitvoer van gegevens. Thans is de invoer aan de orde, speciaal wat betreft de met de hand geponste telexbanden. Omdat deze banden onmiddellijk geponst worden van de programmavellen, moeten deze opgesteld zijn volgens de daarvoor geldende regels; deze regels heten de ponsconventies.

De 32 toetsen van de telexbandponsers zijn voorzien van de vol-

gende etiketten (in de 1e kolom staat het hoofdsymbool = de bandwaarde van de betroffen pentade; in de 2e kolom staat het nevensymbool, indien aanwezig):

1e	2e	1e	2e	1e	2e	1e	2e
0		8		16	A	24	J
1		9		17	B	25	K
2		10	00	18	C	26	L
3		11	000	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

De eerste 10 zijn dus enkel benoemd, de laatste 22 dubbel. Voordat het invoerprogramma in staat is woorden te lezen d.w.z. successieve pentades tot woorden te assembleren) moet van te voren het volgende gespecificeerd zijn:

1. De wisselstand: als het woord uit de gelezen pentades is opgebouwd, kan het n.l.
   a. in het geheugen geschreven worden
   b. met de inhoud van het geheugen vergeleken worden. We noemen a) de "schrijfstand": hier vindt de werkelijke invoer (van band naar geheugen) plaats. We noemen b) de "controlestand": als bij vergelijking een verschil optreedt, stopt de machine. Zodoende is enige controle op de invoer mogelijk.
2. De plaats, d.w.z. het adres in het geheugen, waarop het van de band opgebouwde woord moet (komen) te staan. Opeenvolgende woorden hebben, tenzij anders aangegeven, op successieve adressen betrekking; de plaats hoeft dus steeds alleen voor het eerste woord van een rijtje expliciet aangegeven te worden.
3. De soort, d.w.z. de regels, die de pentade assemblage tot gehele woorden beheersen. Deze regels zijn n.l. voor verschillende soorten verschillend. Thans kent het invoerprogramma vier soorten moleculen, n.l. getallen, opdrachten(paren), binaire woorden en typcodes.
   Naast de woorden komen op de band de z.g. controlecombinaties voor: dit zijn o.a. speciale pentadegroepen, die de drie zojuist genoemde specificaties omschrijven en wijzigen.
   Tenslotte vindt men op de band:
   pentades X (= 31, 5 gaatjes): deze worden voor elke controlecombinatie en elk (niet binair)woord geskipt. Deze faciliteit is ingelast, om tijdens het ponsen ontdekte fouten te kunnen corrigeren;
   pentades blank (= 0, geen gaatjes): deze mogen aan het begin van een band en na bepaalde controlecombinaties in een willekeurig aantal voorkomen. Deze faciliteit is ingelast, om het inleggen van de band in de bandlezer te vergemakkelijken en om de verschillende stukken van de band te kunnen scheiden. Het ontbreken van ponsingen maakt hier de band gemakkelijk beschrijfbaar.

Omdat het de bedoeling is, binaire woorden niet met de hand

te ponsen, maar dit alleen door het ponsprogramma van de ARMAC te laten doen, beperken wij ons in de nu volgende beschrijving tot getallen en opdrachten. De ponsconventies betreffende de typcodes wordt uitgesteld tot de uitvoer aan de orde is.

Om getallen te kunnen lezen, moet de soort gespecificeerd zijn door de controlecombinatie RG (zie later). Het invoerprogramma kan dan gehele getallen en echte breuken lezen; zij bestaan uit een teken (+ of - voor gehele getallen, +. of -. voor echte breuken), gevolgd door het decimale gedeelte. De decimale cijfers worden in de normale volgorde van links naar rechts geponst, men mag hier alleen gebruik maken van de toetsen 0 t/m 9, 00 (dubbel nul) en 000 (tripelnul); deze twee laatste toetsen mogen niet gebruikt worden als eerste cijferponsing bij gehele getallen. Omdat nonsignificante nullen aan het begin van gehele getallen weggelaten mogen worden, is er ook niet de minste reden, na + of - de dubbelnul of tripelnul te willen ponsen.
Gehele getallen bestaan uit teken en minstens een decimaal cijfer; breuken bestaan uit teken (annex punt) en maximaal negen cijfers achter de komma, desgewenst minder; nonsignificante nullen aan het einde van breuken hoeven dus niet geponst te worden.
N.B. Breuken worden niet exact ingevoerd!
De meeste decimale breuken repeteren in het tweetallig stelsel, maar ook een breuk, die door de machine exact kan worden voorgesteld, is soms 2^{-33} mis. (B.v. +.75 komt als +.75 - 2^{-33} in de machine).

Om opdrachten te kunnen lezen, moet de soort gespecificeerd zijn door de controle-combinatie RD (zie later). Het is onmogelijk één losse opdracht in te lezen: bij elkaar behorende a- en b-opdracht worden altijd samen ingevoerd. Van een opdrachtenpaar wordt eerst de a-opdracht, dan de b-opdracht geponst, beide volgens dezelfde conventies. Zo worden b.v. ook aan het begin van de b-opdracht - dus midden in een woord - eventuele extra pentades X geskipt.

Elke opdracht wordt in tweeën geponst: functiegedeelte, gevolgd door adres (of algemener: numeriek gedeelte; omdat het echter voor de ponsconventies geen verschil maakt, zullen wij dit gedeelte van de opdracht met de kortste naam "adres" blijven aanduiden). Omdat op zijn beurt het adres in drieën wordt geponst, bestaat elke opdracht in volgorde uit de volgende vier onderdelen:

f = functie
p = plaats
s = sluitletter
k = kanaalcorrectie
} adres.

ad f: het functiegedeelte loopt van 0 t/m 29 en wordt altijd in 1 pentade geponst.

ad p: de plaats wordt geponst in minstens 1 pentade. Als $p \leq 31$ (dit is regel) mag p in 1 pentade geponst worden; als $p \geq 32$, moet men de successieve decimale cijfers ponsen, waarbij ook dubbel- en tripelnul, mits niet aan het begin - gebruikt mogen worden.

ad s: de sluitletter is een van de symbolen A,B,C,D,E,F,G,H,J, K,L,T,P,S,R of X en wordt dus altijd in 1 pentade geponst.

ad k: de kanaalcorrectie loopt van 0 t/m 31 en wordt altijd in 1 pentade geponst.

De sluitletters markeren beginpunten (adressen), relatief ten opzichte waarvan men de adressen nummert; het moet afgeraden worden voor deze beginpunten iets anders dan nulde opdrachten van kanalen te kiezen, in verband met de bufferschrijfopdracht etc. Het effect van een sluitletter is, dat (het adres van) het bijbehorende beginpunt opgeteld wordt bij het door p en k in eerste instantie omschreven adres = p + 32k. De kanaalcorrectie geeft dus aan het aantal 32-vouden, dat bij p moet worden opgeteld: dit is ook de reden, waarom doorgaans $p \leq 31$ is. De sluitletter X markeert altijd adres 0, dus het begin van het geheugen; X laat dus p + 32k onveranderd. De vijftien andere additieve parameters mag de programmeur zelf kiezen: zij worden aan de machine meededeeld door speciale controlecombinaties (de vulindicaties, zie later) op de z.g. voorponsing, die ingelezen moet worden vòòr de band, waarop de sluitletters gebruikt worden.

Voordat de ARMAC in het invoerprogramma gestart wordt, moet de band met het stuk blank onder de bandlezer worden gelegd. De wijze van starten bepaalt de wisselstand: men start het invoerprogramma met toets 0 van het handregister in de schrijfstand, met toets 1 in de controlestand (zie "De autostarts"). Wat betreft de soortspecificatie is het invoerprogramma nog in het ongewisse; deze is daarom door de start ingesteld op "skip blank, X tot R", d.w.z. de machine gaat bandlezen, skipt blank (indien aanwezig) tot aan X of R; als X gelezen mocht zijn, dan skipt de machine alsnog deze en eventuele verdere

pentades X tot R. Komt er nu een pentade ≠ R, dan stopt de machine: bij doorstarten wordt deze pentade evenwel als R geinterpreteerd. R is n.l. de aankondigingspentade voor controlecombinaties. Voordat er echte woorden op de band verschijnen, moet immers nog plaats en soort gespecificeerd worden. Omdat deze specificeringen door controlecombinaties worden beschreven, "eist" het invoerprogramma eerst de pentade R. Dit is een van de voorbeelden, waar het invoerprogramma de band "nakijkt". Zo stopt de ARMAC b.v. eveneens, als de plaats niet gespecificeerd wordt.

Tot nu toe zijn de volgende controlecombinaties toegestaan. Voor elke R wordt een willekeurig aantal pentades X geskipt.

RA (Adresindicatie)
De plaats van het eerstvolgende echte woord wordt gespecificeerd door onmiddellijk achter RA het adres (in drieën, als bij de opdracht) te ponsen, waar het eerstvolgende echte molecuul moet (komen te) staan. Afhankelijk van de wisselstand zal er worden geschreven of vergeleken. Evenals de wisselstand blijft ook de soortspecificatie bij RA.... onveranderd, d.w.z. was de soort nog ongespecificeerd, dan komt de besturing na afloop terug in "Skip blank, X tot R"; was de soort al gespecificeerd als b.v. getallen, dan is het programma klaar om nu de getallen te lezen, evenzo met opdrachten en typcodes. Voor binaire moleculen geldt dit niet. (zie onder).

RB (Biband)
De controlecombinatie RB stelt de soortspecificatie in op binaire moleculen; de uitzonderingspositie van RB bestaat hierin, dat bij het lezen van bibanden de ARMAC "blind" is voor alle controlecombinaties op één na: de indicatie dat het einde bereikt is. (Omdat een binair woord nooit met een pentade = blank begint, is voor deze aanduiding gekozen een pentade blank na de laatste pentade van het laatste binaire woord.) Deze blindheid impliceert, dat voor het speciale geval van de biband de soort (RB) pas gespecificeerd mag worden, als de plaats vastligt. Na afloop komt de besturing terug bij "Skip blank, X tot R", wel nieuwe soort-specificatie eisend, maar niet nieuwe plaats; tenzij anders aangegeven op de band, heeft het volgende echte woord normaal betrekking op het volgende adres. RB werkt bij beide wisselstanden en laat deze onveranderd.

RC (Controlewissel)
Door de controlecombinatie RC wordt de stand van de controlewissel omgezet; deze omzetting wordt pas effectief bij de

eerstvolgende RA-combinatie. Na RC komt de besturing in "Skip blank, X tot R" terug, zowel plaats- als soortspecificatie eisend.

RD (Lees opdrachten)

De controlecombinatie RD stelt de soort-specificatie in op opdrachten. Het invoerprogramma leest daarna tot nader aankondiging alleen opdrachtenparen en controlecombinaties. RD laat wisselstand en plaatsbepaling onbeinvloed.

RE (Einde band)

Door de controlecombinatie RE gaat de besturing over naar "Skip blank, X tot R", wel nieuwe soortspecificatie eisend, wisselstand en plaatsbepaling onbeinvloed latend. Deze controlecombinatie is meer de aankondiging van het start blank, dat volgt. Als men de band af wil scheuren, ponst men achter het laatste stukje blank altijd EE, de stoppende versie van RE; maakt men met behulp van de reproducer van vele kleine banden een lange, dan laat men deze EE's, behalve de laatste steeds weg.

RF (Vulindicatie adres)

Om voor de sluitletters met variabele betekenis (dus: A,B,C, D,E,F,G,H,J,K,L,T,P,S, en R) de bijbehorende additieve parameters gelijk te maken aan een of ander adres, gebruikt men de zg. vulindicatie RF; hierna ponst men de letter, waaraan de nieuwe waarde toegekend wordt, daarna wordt de nieuwe parameter als adres geponst (dus in drieën: plaats, sluitletter, kanaalcorrectie). De betekenis van de in dit adres gebruikte sluitletter, indien $\neq$ X, moet door een eerdere voorponsing zijn ingevuld. De vulindicatie RF verandert niets aan de wisselstand, de plaats van wegbergen of de soortspecificatie. Om aan een sluitletter-parameter een waarde, anders dan een adres toe te kennen, gebruikt men de vulindicatie RH (zie onder).

RG (Lees getallen)

De controlecombinatie RG stelt de soort-specificatie in op getallen. Het invoerprogramma leest tot nader aankondiging alleen getallen en controlecombinaties. RG laat wisselstand en plaatsbepaling onbeinvloed.

RH (Vulindicatie, woord)

Om voor de sluitletters met variabele betekenis de bijbehorende additieve parameters gelijk te maken aan iets anders dan een adres - dus b.v. een getal of een opdrachtenpaar, wat

voor specifieke toepassingen soms gewenst is - gebruikt men de vulindicatie RH; hierna ponst men de letter, waaraan de nieuwe waarde toegekend wordt, daarna (indien nodig) een soort-specificatie, gevolgd door het woord. De vulindicatie RH verandert niets aan de wisselstand, de plaats van wegbergen of de soortspecificatie. De tussen RH en het betroffen woord geponste soort-specificatie is dus alleen op dit volgende woord van toepassing. Ontbreekt deze soortspecificatie, dan wordt een woord volgens de "heersende" soort gelezen. N.B. Getallen in de RH-combinatie moeten door een pentade X worden afgesloten!

RJ (Jump)

Na RJ wordt ongeacht de gespecificeerde soort één opdracht gelezen; deze wordt vervolgens uitgevoerd. Aan het feit, dat deze opdracht altijd een sprongopdracht zal zijn, ontleent deze controlecombinatie zijn naam. Wisselstand, soortspecificatie en plaatsbepaling blijven onbeinvloed. Tussen RJ en de opdracht mogen geen extra pentades X ingelast worden!

RT (Lees typcode)

Door de controlecombinatie RT wordt de soortspecificatie ingesteld op typcodes. Tot nader aankondiging leest het invoerprogramma alleen typcodes en controlecombinaties. RT laat wisselstand en plaatsbepaling onbeinvloed.

RX (Skip)

De controlecombinatie RX wordt altijd door 1 pentade gevolgd, die gelijk is aan het aantal adressen, dat worde overgeslagen bij het wegbergen of vergelijken: als in een programma een adres opengelaten moet worden, omdat deze door het programma zelf ingevuld wordt, ponst men daarvoor in de plaats RX1. Deze faciliteit is overbodig; men zou met RA, gevolgd door een adres hetzelfde hebben kunnen bereiken. RX wijzigt de plaatsbepaling, maar verandert niets aan wisselstand en soortspecificatie.

Tot zover de controlecombinaties. Hoe sommige gebruikt worden, illustreren we bij de opmaak van z.g. zelfcontrolerende banden. De band begint met een roffel blank, plaats- en soortspecificatie van het eerste woord, dan de woorden (en eventuele controlecombinaties, als woorden van verschillende soort geassembleerd moeten worden etc.). Na het laatste woord ponst men RC, gevolgd door een roffeltje blank; dan herhaalt men de band totzover (d.w.z. van het roffeltje blank aan het begin tot en met de roffel blank na RC); na deze pentades blank

ponst men de stopcombinatie EE, daarna weer roffel blank, waarin de band afgescheurd kan worden. Wordt met het begin van deze band in de bandlezer het invoerprogramma schrijvend gestart, dan wordt het tweede stuk controlerend gelezen, door de laatste RC-combinatie wordt het invoerprogramma weer in de schrijfstand achtergelaten. Dat is van belang, als men de banden tot een geheel assembleert en alternerend schrijvend en controlerend gelezen wordt.

Met de kanaalcorrectie bestrijkt men 32 kanalen, dus slechts een kwart van het geheugen: om een adres in het laatste driekwart van het geheugen aan te geven, moeten we dus iets van de tafel van 32 kennen. Dit worde beperkt tot de voorponsing, en wel alleen de tafel van 1024 = 32.32.
Om aan te geven, dat kanaal B 0 kanaal 73 zal zijn ponst men

RFB 2048 X9

De operateur leest hier onmiddellijk het kanaalnummer 64 + 9 uit af.

Tot slot over de invoer een enkel woord over de voorponsingen.
De functie van de sluitletter is tweeërlei.
Ten eerste is op deze wijze een overzichtelijke opbouw en kortere notatie van het programma mogelijk. De hiervoor gebruikte sluitletters hebben in het hele probleem dezelfde betekenis en worden aan de machine meegedeeld in de z.g. generale voorponsing van het programma.
Ten tweede vervullen de sluitletters een belangrijke functie bij de invoer van standaardsubroutines. Behalve de standaardsubroutines, die permanent in de kanalen 16 t/m 31 aanwezig zijn, zijn er meer subroutines, waarover de programmeur beschikken kan. Hij mag zelf kiezen, op welke kanalen hij deze subroutines zetten wil. Om hem toch in staat te stellen, deze subroutine met een standaardband in te voeren, zijn deze subroutines in termen van variabele sluitletters geponst. De programmeur hoeft er slechts een z.g. specifieke voorponsing aan toe te voegen. Zodra twee subroutines in termen van dezelfde sluitletter geponst zijn, wisselt de betekenis van deze sluitletter dus. Om niet "klem" te komen zitten, is de afspraak gemaakt, dat de subroutines in termen van de achterste sluitletters geponst worden. De eerste kan de programmeur in zijn generale voorponsing gebruiken. Het komt de overzichtelijkheid ten goede, als men consequent alle verwijzingen naar subroutines evenals de specifieke voorponsingen, ponst in termen van de generale voorponsing.

N.B. Het programma, dat banden in overeenstemming met boven beschreven ponsconventies leest, gebruikt het hele kanaal XO.

10. Subroutines "lees binair woord" (gebruiken 29 XO)

Elk binair woord begint met een pentade ≥ 16. Er zijn twee subroutines voor het lezen van binaire woorden; beide testen of de eerstgelezen pentade ≥ 16 is. De ene skipt alle eventuele pentades < 16, totdat een pentade ≥ 16 gevonden is en leest dan alsnog een binair woord. De ander komt onmiddellijk terug, zodra een pentade ≤ 16 gelezen is. De ene aanroep is

23 2 X 17 =) "lees binair woord ⇒ (S)"

Eventuele pentades < 16 (dus a fortiori pentades blank) aan het begin worden geskipt.

De andere aanroep luidt

23 5 X 17 =) "lees, zo mogelijk, binair woord ⇒ (S)"

Met "zo mogelijk" wordt bedoeld, als de eerste pentade ≥ 16 is; de besturing komt dan met negatieve conditie in het hoofdprogramma terug. Als de eerste pentade < 16 is, wordt er niets meer gelezen en de besturing komt onmiddellijk in het hoofdprogramma terug, maar nu met positieve conditie.

11. De ponsprogramma's

Thans is de uitvoer van gegevens aan de orde. Dit geschiedt door ponsprogramma's of typprogramma's. In aansluiting op de bovenbeschreven leessubroutines van binaire woorden beschrijven wij eerst de ponsprogramma's.

22 24 X 19 =) "pons roffel blank" (± 15 cm)

Hierbij wordt dus nog geen informatie geponst.

Om één woord uit te ponsen, plaatst men dit in S en geeft de aanroep

22 27 X 20 =) "pons (S) binair"

Het is tevens mogelijk de woorden op een rij opeenvolgende woorden op een rij opeenvolgende adressen uit te ponsen. Om deze rij vast te leggen, specificeert men begin en lengte, d.w.z. het adres van het eerste uit te ponsen woord en het aantal adressen. Deze specificaties worden tot een codewoord gecombineerd en wel: het beginadres in de a-helft, de lengte in de b-helft. Als opdrachtenpaar beschouwd, staat op de a-helft dus altijd een 0-opdracht; op de b-helft kan behalve een 0-opdracht ook een 16-opdracht staan (dus een 1 op de

plaats van het tekencijfer van het codewoord). In dat geval worden de controlecombinaties onderdrukt. Men plaatst het codewoord in S en geeft de aanroep

22 28 X 19 =) met codewoord in S.

Als ook "de b-opdracht in het codewoord" een O-opdracht is, gebeurt het volgende:

1. Er wordt een roffel blank gegeven
2. De adresindicatie voor het begin wordt geponst: RA....XO.
3. De controleconbinatie RB wordt geponst.
4. De successieve woorden worden binair uitgeponst; achter het laatste woord komt een pentade blank.

Was "de b-opdracht in het codewoord" een 16-opdracht geweest, dan was 2. en 3. overgeslagen.

Dit laatste ponsprogramma is ook als zelfstandig programma te gebruiken. Daartoe zet men het oodewoord in de getalschakelaars en start de machine met de handregistertoets 9 (zie "De autostarts"). Na de pentade blank achter het laatste woord stopt de machine.
Start men door, dan ponst de machine RCO en stopt.
Start men weer door, dan ponst de machine een roffel blank, gevolgd door EEO en stopt.

12. Het typprogramma

Na de ponsprogramma's is thans de typroutine aan de orde, of liever het aggregaat der typroutines. In verband met de gevarieerde eisen, die men aan een dergelijk programma stellen kan, is de flexibiliteit van dit standaardprogramma zo hoog mogelijk gehouden. Het resultaat is geworden een aggregaat van in elkaar grijpende subroutines.

De standaardtyproutines typen de inhoud van S als getal.

Een van de functies van het typprogramma is, uit de in S meegegeven binaire representatie de opeenvolgende decimale cijfers uit te rekenen; deze zijn afhankelijk van de plaats, waar de binaire komma in het woord (S) wordt geinterpreteerd. De typroutine voorziet hier in de twee meest gebruikelijke mogelijkheden: òf achter het laagste cijfer (d.w.z. typ [S], dus als geheel getal) of onmiddellijk achter het tekencijfer (d.w.z. typ {S}, dus als breuk).

De onderscheiding in "geheel getal" en "breuk" slaat op de interpretatie van het binaire woord (S), niet op het beeld op de pagina! Het is n.l. aan de programmeur om te bepalen,

òf, en zo ja, voor welk cijfer, ~~een~~ punt getypt moet worden.

Voorts kan òf de absolute waarde, òf het getal met teken getypt worden; ook het aantal cijfers is variabel. Deze en nog enige andere vrijheden worden voor elk geval vastgelegd in een speciaal codewoord, typ-code genaamd

Het standaard-typprogramma bevat ruimte voor tien typ-codes, genummerd van 0 t/m 9; in het geheugen zijn hiervoor in volgorde gereserveerd de adressen 4080 t/m 4089. Hiermede corresponderen in volgorde de aanroepen van de typroutines
22 0 X 28, 22 1 X 28 .... 22 9 X 28. Zo typt b.v. de aanroep

10	13	B	2
22	5	X	28

=)

de inhoud van 13 B 2 volgens de specificatie van typcode 5 in adres 4085.

Normaal zal tien verschillende typcodes voor één programma wel genoeg zijn. In dat geval worden zij bij de invoer meteen op de adressen 4080 e.v. ingevuld. Mocht men aan tien verschillende typcodes niet genoeg hebben, dan kan b.v. het programma de betekenis van de aanroep 22 9 X 28 wijzigen door op adres 4089 een nieuwe typcode te schrijven.

Opm. 1: Het is essentieel, dat, zoals in bovenstaand voorbeeld, de 10-opdracht, die het te typen getal in S zet, onmiddellijk gevolgd wordt door de sprong naar de typroutine. Deze restrictie is een gevolg van de manier, waarop het typen gecontroleerd wordt (zie later).
Opm. 2: Alle typroutines voor getallen werken, mits de schrijfmachine op "kleine letter" is ingesteld; zij laten de schrijfmachine in deze stand achter.

Het opstellen van de typcode

Als het teken getypt moet worden, begint het codewoord met T; als het teken onderdrukt moet worden, laat men deze T weg. De typroutine typt dan de absolute waarde; er wordt geen spatie in de plaats van het onderdrukte teken gegeven, er gaat dus één signaal minder naar de schrijfmachine.

Hierna komt een S, als na (al of niet) het teken een spatie ingelast moet worden; ontbreekt deze S in de typcode, dan wordt er geen spatie ingelast.

Hierna komt Gn of Bn.
In het geval Gn ($1 \leqslant n \leqslant 10$) wordt (S) opgevat als geheel getal van n cijfers; het nde cijfer komt overeen met het eenhedencijfer. Als $n \leqslant 9$, moet [S] in absolute waarde kleiner zijn dan 10^n; als n = 10 is automatisch het eerste cijfer maximaal 8 (het grootste positieve getal is 85899 34591 = $2^{33} - 1$.)
In het geval Bn ($2 \leqslant n \leqslant 10$) wordt (S) opgevat als breuk van n decimale cijfers; het eerste cijfer is het eenhedencijfer, de breuk wordt exact afgerond op het laatste cijfer, dus de n-1ste decimaal achter de komma. Door deze afronding kan de breuk = $\pm$ 1 worden; daarom is van de n cijfers 1 cijfer voor, en de rest achter de komma gekozen. Als regel is het eerste cijfer voor de komma natuurlijk = 0. Omdat $n \geqslant 2$, komt dus minstens één cijfer achter de komma.

Elk dezer n cijfers kan op drie wijzen verwerkt worden.

a) "Loos": als van te voren bekend is, dat een of meer cijfers = 0 zijn en wij deze nullen niet op het papier wensen te zien, kunnen ze geheel onderdrukt worden. Als een geheel getal b.v. een exact 1000-voud is, maar we in deze nullen niet geinteresseerd zijn (omdat ze b.v. een gevolg zijn van om programma-technische redenen ingelaste factor 1000) eindigt het typproces met drie slagen in de z.g. loze cyclus.
b) "imperatief", d.w.z. het verwerkte cijfer wordt zonder meer getypt (zie ad c).
c) "Facultatief", d.w.z. als het verwerkte cijfer $\neq$ 0 is, wordt het, benevens alle volgende cijfers, zonder meer getypt; als het verwerkte cijfer = 0 is (en vorige cijfers $\neq$ 0 zijn nog niet opgetreden), dan wordt het door een spatie vervangen.

Opm.1: De typroutine heeft twee standen, de facultatieve en de imperatieve; in de eerstgenoemde worden nullen door spaties vervangen. Tenzij anders gespecificeerd begint het typprogramma bij elke nieuwe aanroep in de facultatieve stand; het kan, eventueel onmiddellijk, naar de imperatieve stand overgaan òf doordat een cijfer $\neq$ 0 verwerkt wordt, òf door expliciete aanduiding in de typcode. De overgang terug naar weer facultatief typen is in hetzelfde getal niet meer mogelijk.
Opm.2: Men realisere zich de verschillen tussen loos en facultatief: bij facultatief typen wordt het cijfer, dat = 0 blijkt

te zijn, door een spatie vervangen, bij loze verwerking wordt er geen enkel signaal naar de typmachine gezonden. Verder betreft de loze verwerking cijfers, die gegarandeerd = 0 zijn, de facultatieve echter cijfers, die misschien = 0 zijn.

Bovendien kan men desgewenst punten of spaties inlassen. De punt bewerkstelligt, als nog facultatief getypt werd, de overgang naar imperatief voor de volgende cijfers; de spatie laat evenals een aantal loze verwerkingen deze stand onbeinvloed.

Preciezer: na Gn of Bn kunnen wij kiezen uit de volgende specificaties.
Fm ($m \geq 1$): typ m facultatieve cijfers
Sm ($m \geq 1$): typ spatie, gevolgd door m cijfers
Jm ($m \geq 1$): typ m imperatieve cijfers
Pm ($m \geq 1$): typ punt, gevolgd door m imperatieve cijfers
Lm ($m \geq 1$): verwerkt m cijfers loos.

Hieruit mogen voor een typcode maximaal zes keuzen gedaan worden; deze keuzen specificeren in volgorde van typen, dus van links naar rechts, hoe de n "typbare" cijfers verwerkt moeten worden.

Opm.: Omdat steeds $m \geq 1$ is, kan men b.v. niet twee punten, een punt en een spatie, etc. achter elkaar inlassen.

Na afloop wordt de typcode afgesloten door
XK, als hier het getal klaar is,
Xs, als het getal door een spatie gevolgd moet worden
XT, als het door een tabulatie gevolgd moet worden.

Enige voorbeelden mogen dit toelichten:

T G10 S9 J1 XK
TS G10 F9 J1 XK
TS G10 F4 F5 J1 XK

} typ geheel getal van 10 cijfers na teken en spatie; de eerste negen cijfers facultatief, het laatste imperatief. Het onderste voorbeeld illustreert, hoe het ook kan, maar niet moet.

T B10 F1 P3 S3 S3 XT: typ teken, gevolgd door breuk met 1 facultatief cijfer voor de komma - dit zal haast altijd een spatie geven - gevolgd door een punt en negen cijfers, door twee spaties in drie groepjes van drie gescheiden. Na afloop Tab.

Gevraagd een in centen gegeven bedrag uit te typen in guldens: het bedrag wordt zonder teken getypt en is minder dan

ƒ 1.000.000,--; de zes cijfers voor het gulden aantal worden door een spatie in twee maal drie verdeeld. Als het getal door een spatie afgesloten moet worden luidt het antwoord: G8 F3 S2 J1 P2 XS. Tussen gulden- en centenaantal wordt een punt getypt. Als we weten, dat het bedrag een geheel aantal guldens is, en als de punt en de twee dan volgende nullen niet getypt moeten worden, zou de typcode luiden:
G8 F3 S2 J1 L2 XS.

Als we van een breuk het duizendvoud uit willen typen, maar weten, dat dit maximaal 10 is, en nog drie cijfers achter de komma willen zien, dan luidt de typcode:
B7 L2 F1 J1 P3 XT. Na afloop hebben we een tabulatie laten geven.

Het inbrengen van de typcode

De lezer zal gemerkt hebben, dat voor de opstelling van de typcode slechts gebruik is gemaakt van symbolen die op het toetsenbord voorkomen. Men ponst het codewoord van links naar rechts, alle aantallen met één pentade ponsend. Aan het begin van elke typcode wordt een willekeurig aantal (mag = 0 zijn) pentades X geskipt. De "soort" moet het laatst gespecificeerd zijn door de controlecombinatie RT: "Lees typcode".

- Het Layout-programma

  Ook het "verdelen" van de getallen over de pagina wordt door het standaardtypprogramma verzorgd.

  Regelindeling

  Aan het begin van elk getal moet een tabulatorstop staan, ook aan het begin van het eerste; er komt dus een tabulatorstop enige plaatsen rechts van de kantlijn te staan. Als regel worden er links van de stop van het meest rechtse getal niet meer stoppen gezet.

  Normaal wordt een vast aantal (zeg i) getallen per regel getypt. In dit geval wordt de z.g. regelparameter, waar adres 4077 in het geheugen voor gereserveerd is, tijdens de invoer = i gemaakt. De typroutine geeft dan na elk i^{de} getal het signaal TWNR en na een vertraging om de wagen de tijd te geven, terug te lopen tot de kantlijn, een Tab-signaal, zodat dan de wagen weer klaar staat om het eerste getal van de volgende regel te typen.

Als we in een van de kolommen een plaats open willen laten, doen wij dit met de aanroep
22 6 X 26 =) "acht getal geteld getypt".
Door een Tab-signaal loopt de wagen door tot aan de volgende tabulatorstop; de telling van het aantal nog te typen getallen op diezelfde regel wordt met 1 verlaagd. Dit Tab-signaal komt in dit geval altijd, ook als men het laatste getal op de regel wil overslaan. Men moet daarom rechts op de wagen nog minstens een extra tabulatorstop zetten.

De telling van het aantal getallen per regel vervult een dubbele functie. Ten eerste om het einde van de regel te detecteren, ten tweede om bij te houden, bij de hoeveelste tabulatorstop van de regel de wagen staat. Het typen is een door de machine gecontroleerde operatie: als een fout gedetecteerd wordt, typt de machine op de volgende regel het mislukte getal in dezelfde kolom over. Daartoe moet na het signaal TWNR het goede aantal tabulatorsignalen gegeven worden. Dit is de reden, waarom er aan het begin van elk getal een tabulatorstop moet staan en ook waarom men liever niet meer tabulatorstoppen in moet zetten. Als het toch gewenst is dat ergens een extra tabulatorstop staat, die bij het typen van de cijfers van een getal gepasseerd wordt, moet de typaanroep van dit getal gevolgd worden door
22 24 X 26 =) "tel gepasseerde Tab.".
Voor de vaststelling van de regelparameter moet zo'n tabulatorstop natuurlijk meegeteld worden.

Blokindeling

Vaak is het gewenst, dat de regels in z.g. blokken van een vast aantal (zeg j) onder elkaar getypt worden, terwijl de afzonderlijke blokken door een extra regel blank gescheiden worden. Indien dit gewenst is, maakt men bij de invoer de z.g. blokparameter, waarvoor adres 4078 gereserveerd is, gelijk aan j, het aantal regels per blok; als men de blok-indeling wil onderdrukken, zet men de blokparameter j = 0.

Blokindeling onderdrukken wil zeggen, dat de regels niet geteld worden. In dit geval is er geen verschil tussen de beide aanroepen:
23 1 X 26 =) "acht regel geteld voltooid"
en
22 3 X 26 =) "acht regel ongeteld voltooid"

In beide gevallen wordt TWNR- en Tab-signaal gescheiden door een vertraging, gegeven en wordt de telling van de getallen - preciezer het aantal gepasseerde tabulator-stoppen - ingesteld op het begin van de regel.

Als j ≠ 0, dan telt de op deze manier afgemaakte regel in het eerste geval wel, in het tweede geval niet mee in de blokindeling.

Paginaindeling

Indien blokken bestaan uit een vast aantal regels, kan het gewenst zijn, een vast aantal (zeg n) blokken op een pagina te scheiden. Voor het maken van een tafel wil men b.v. de pagina's scheiden door een aantal extra regels blank; aan de bovenkant van de pagina wil men misschien opschriften boven de kolommen typen etc. Dit regelt men met de z.g. paginaparameter k, waar in het geheugen adres 4079 gereserveerd is.

Er zijn voor deze parameter drie gevallen:

k = 0 : de paginaindeling treedt niet in werking en blokken worden dus niet geteld.

k = 2n - 1 : na n blokken wordt nog een extra regel blank ingelast (dus totaal komen hier twee regels blank, n.l. ook al een, omdat het einde van een blok bereikt was). Dit treedt uiteraard alleen in werking, als j ≠ 0.

k = 2n : na n blokken wordt nog geen extra (tweede) regel blank ingelast, want de layout-routine handelt, als of er nog een n + 1ste blok onderaan de pagina komt, bestaande uit slechts één regel. Dit arrangement stelt ons in staat, om, wat bij sommige tabellen gewenst is, de eerste regel van elke pagina ook onderaan de vorige bladzijde te typen. Dit werkt weer alleen als j ≠ 0.

Alleen aan het einde van de pagina komt de besturing met de conditie positief terug in het hoofdprogramma. Hierdoor is de programmeur in staat, aam het begin van de nieuwe pagina zijn maatregelen te treffen, als daar zijn het geven van extra regels blank, het typen van het pagina-nummer, hoofden boven de kolommen en, als k = 2n, het herhalen van de onderaan de vorige pagina reeds getypte regel.

Analoog aan twee eerder gegeven aanroepen voor voltooide regels kent het communicatieprogramma de aanroepen:

22 4 X 26 =) "acht blok geteld voltooid"

en

22 5 X 26 =) "acht blok ongeteld voltooid"

Beide sturen naar de schrijfmachine twee TWNR-signalen, gevolgd door één tabulatie-signaal. Tellingen worden ingesteld op het begin van een blok.

De aanroep

22 0 X 26 =) "start nieuwe pagina"

is aan het begin van elk programma verplicht. Hier worden de volgende verrichtingen uitgevoerd:

1. Twee TWNR- en een Tab-signaal worden naar de schrijfmachine gezonden.
2. Alle tellingen worden ingesteld op het begin van de pagina.
3. Voor de controle op het typen noodzakelijke voorbereiding wordt getroffen.

Voorts staan de programmeur ter beschikking de aanroepen:

22 0 X 24 =) geef TWNR, waarna vertraging

en

22 3 X 24 =) geef Tab, waarna vertraging.

De vertraging na TWNR is aanmerkelijk langer dan die na Tab.

De controle op het typen

Het is gezegd, dat de 10-opdracht, die het te typen getal in S plaatst, onmiddellijk aan de subroutinesprong vooraf moet gaan.

De reden hiervoor is, dat na het typen de besturing niet in het hoofdprogramma bij de volgende, maar bij de vorige opdracht terug komt. Hier wordt dan weer het getal in S geplaatst; in de typroutine wordt nu de besturing naar een controleprogramma gestuurd, waar het ten tweeden male aangehaalde getal vergeleken wordt met een uit de getypte symbolen opgebouwd controleresultaat. Bij deze controle wordt ook de typcode opnieuw aangehaald en ontrafeld.

Als de controle klopt, komt de besturing bij de volgende opdracht terug; als hij faalt wordt de wagen in de positie gebracht, waarin hij stond, toen aan het mislukte getal begonnen werd, en de besturing komt weer terug bij de vorige opdracht, zodat het getal in de volgende regel weer wordt

getypt. De hier gegeven extra TWNR telt niet mee in de blokindeling.

Typen en controleren wisselen elkaar dus af. De wissel, die dit beheerst, wordt elke keer omgezet. Door "start nieuwe pagina" wordt deze zo gezet, dat de eerste keer getypt, en niet gecontroleerd wordt.

Het tab-bandje

Achteraan het bandje dat de z.g. typconstantes (d.w.z. lay-out-parameters en typcodes) inbrengt (dit stuk make men zelfcontrolerend) ponst men het z.g. tabbandje.

Als a,b,c,....n de afstanden zijn van elke tabulatorstop tot de linkerkantlijn, resp. de voorafgaande tabulatorstop, ponst men het tabbandje als volgt (RG hier overbodig):

```
     RJ  6  22  X  25
           + a
           + b
           + c
            .
            .
            .
           + n
     RJ  6  27  X  31
```

Men verwijdert voor het inlezen alle tabulatorstoppen. Als het boven aangegeven bandje gelezen wordt, wordt eerst TWNR gegeven. Dan geeft de machine a spaties en stopt. Men zet een tabulatorstop en start door; de machine geeft b spaties en stopt; men zet de tweede tabulatorstop, etc.

Résumé der geheugenverwijzingen wat betreft het typen

Voor invoer:

```
RJ  6  22  X  25: aankondiging van Tabbandje

4077  X  0: adres voor regelparameter
4078  X  0:   "      "  blokparameter
4079  X  0:   "      "  paginaparameter
4080  X  0:   "      "  typcode 0
  .                        .
  .                        .
  .                        .
4089  X  0:   "      "  typcode 9
```

Voor programma

```
22  0  X  28  =): typ volgens typcode 0
     .
     .
     .
22  9  X  28  =): typ volgens typcode 9
22  0  X  24  =): TWNR, waarna vertraging
22  3  X  24  =): Tab,     "         "
22  0  X  26  =): start nieuwe pagina
22  6  X  26  =): acht getal geteld getypt
22 24  X  26  =): tel gepasseerde tab.
23  1  X  26  =): acht regel geteld voltooid
22  3  X  26  =): acht regel ongeteld voltooid
22  4  X  26  =): acht blok geteld voltooid
22  5  X  26  =): acht blok ongeteld voltooid.
```

14. Het handregister

Op het bedieningspaneel van de ARMAC bevinden zich veertien toetsen met de opschriften 0 t/m 9, +, -, +. en -. . Als men op een van deze toetsen drukt, wordt de machine gestart op de a-opdracht van adres 0 X 16 in het communicatieprogramma. De besturing ontmoet dan de handregisteropdrachten (zie "De opdrachtencode") waarmede gelezen wordt, welke toets is ingedrukt. Het handregister wordt gebruikt voor incidentele invoer van decimale getallen en voor de z.g. autostarts.

De invoer van getallen met het handregister

Decimaal gegeven gehele getallen of breuken worden met het handregister in S gebracht door:

1. de betroffen tekentoets in te drukken
2. de opeenvolgende decimale cijfers in te drukken
3. na het indrukken van het laatste cijfer de machine door te starten.

Door het indrukken van de toetsen werkt de machine steeds eventjes, en stopt dan weer. Als na het doorstarten de machine gestopt is, staat het bedoelde getal in S.

Bij gehele getallen hoeven non-significante nullen aan het begin, bij breuken non-significante nullen aan het einde niet aangeslagen te worden. Het doorstarten na het laatste cijfer is verplicht, ook als men halverwege tot de ontdekking komt, dat het helemaal niet de bedoeling is, om een of ander getal in S te brengen.

Als men een fout heeft gemaakt begint men bij het teken op-

nieuw.

De autostarts

De cijfertoetsen van het handregister worden alleen in de decimale opbouw verwerkt, als na de laatste indrukking van een tekentoets nog niet is doorgestart voor afmaking. Anders - normaal - starten zij de machine in standaardfuncties. Van de tien mogelijkheden voor de autostarts zijn er negen gebruikt.

"0": start invoerprogramma schrijvend
"1": start invoerprogramma lezend
"2": typ (A) als geheel getal
"3": typ (S) als geheel getal
"4": typ (A) als breuk
"5": typ (S) als breuk
"6": test trommelgeheugen
"7": lees getalschakelaars: (G) ≠ (S)
"8": herstel vrij kanaal (= kanaal 126 bij invoer)
"9": pons biband

15. De geheugenbezetting der standaardprogramma's

De kanalen 16 t/m 31 zijn door de hierboven genoemde standaardprogramma's bezet. Als speciale voorzorg is het normaal onmogelijk gemaakt, nog op deze kanalen te schrijven.

De beide delingen en de worteltrekking gebruiken uit het snelle kanaal tevens de adressen 0 X 0 en 1 X 0.

Kanaal 127 - het laatste kanaal op de trommel - wordt grotendeels in beslag genomen door het aggregaat der typroutines. Hier worden typcodes, layout-parameters, tellingen en wisselstanden onthouden. Ook kanaal 127 staat dus niet ter beschikking van de programmeur.

Bij het typen van getallen wordt uit het snelle kanaal de adressen 0 X 0 t/m 5 X 0 gebruikt als werkruimtes. De layout-aanroepen gebruiken uit het snelle kanaal adres 4 X 0.

Bij het bandlezen wordt gebruik gemaakt van het snelle kanaal en kanaal 126. Kanaal 126 heet op historische gronden "het vrije kanaal". Wat hier moet staan, wordt ingevuld door autostart "8", met de functie: "Herstel vrij kanaal". De door de voorponsing ingebrachte parameters der sluitletters worden in het vrije kanaal geborgen. Men lette er dus op, dat "Herstel vrij kanaal" de voorponsing bederft.

Wat in het snelle kanaal moet staan, wordt door de autostarts "0" en "1" (de starttoetsen van het bandleesprogramma) ingevuld.

16. De tijdsduur van de opdrachten

De basis-tijdseenheid van de ARMAC is voor de programmeur de getaltijd; deze is 1/2400 sec = 0.417 msec.

Inlezen van de buffer, evenals tracktransport van of naar de trommel duurt altijd 35 getaltijd = 14.6 msec.

De rekentijd voor de vermenigvuldiging is 13 getaltijd = 5.42 msec, alle andere opdrachten duren 1 getaltijd = 0.417 msec. Deze tijden zijn exclusief eventuele wachttijd voor het getal: de lees- en schrijfopdrachten die naar een adres op de trommel verwijzen, duren n.l. langer. In dit geval stellen gedurende de eerste getaltijd de versterkers zich in, daarna wordt het getal zo gauw mogelijk gelezen. Elk adres op de trommel is n.l. eens per omwentelingstijd (eens in de 32 getaltijden) bereikbaar, omdat de adressen van een kanaal in volgorde onder de kop doordraaien. De wachttijd is dus minimaal 1, maximaal 32 getaltijden, dus gemiddeld 16.5 getaltijd = 6.88 msec.

Opm.: In de meeste programma's worden sommige stukjes veel vaker doorlopen dan andere. Om "weinig intensieve stukken" te versnellen, loont dan amper de moeite.