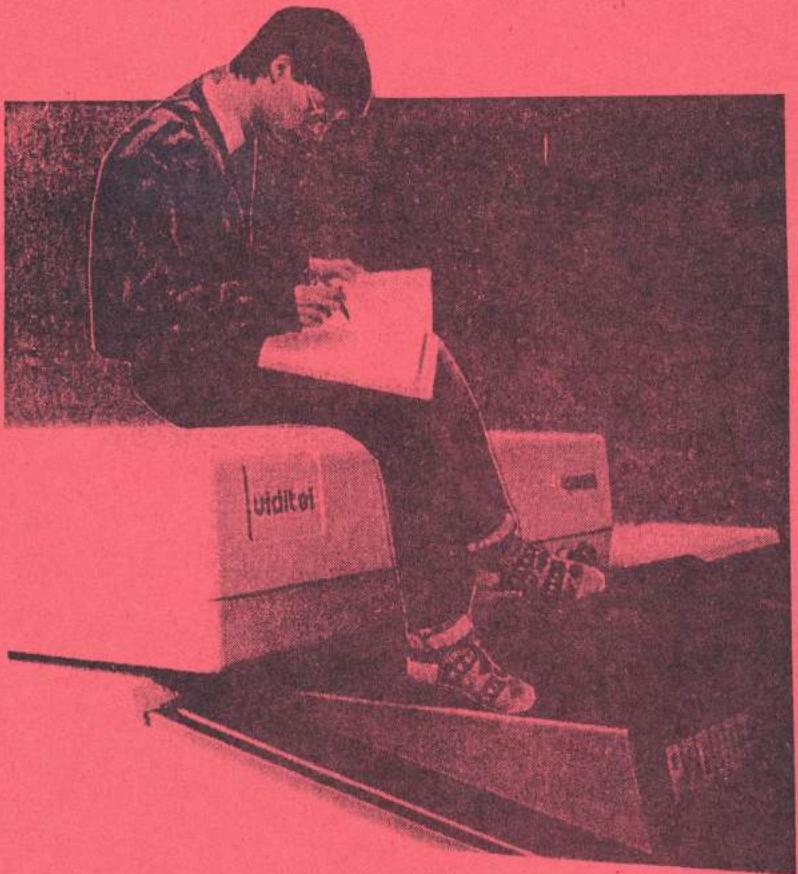


R D O S 3.1

R A M D I S K

64 K U I T B R E I D I N G



J E R O E N

HOPPENBROUWERS

## De banken van de 64K-print als RAM disc

Er is voor de P2000 een grote geheugenprint te koop zodat u uw P2000 uit kunt breiden tot 64K geheugen. Meer kan de computer gewoon niet bevatten. Toch zit er -door technische redenen- meer geheugen op die print. Hoe kan dat dan? Wel, 48K van het printje is direct voor BASIC bruikbaar. Er gaat ongeveer 9K vanaf omdat BASIC zelf ook wat geheugen nodig heeft. Blijft over 39K. Langs die 39K is nu 40K extra gebouwd. Die 40K is verstoppt, zodat de P2000 niet eens weet dat hij meer geheugen heeft! Laat staan dat u het kunt gebruiken.

Nu is er een programma ontwikkeld dat deze extra veertigduizend geheugenplaatsen wel kan benutten. Dat gaat door ze om te bouwen tot *RAM disc*. Een RAM disc kunt u opvatten als een extra recorder: u kunt er programma's of gegevens in opbergen en later weer terughalen. Hoe dat opbergen en terughalen precies in zijn werk gaat is voor u niet van belang: het programma stuurt alles en er kan niets verkeerd gaan.

Om het RAM Disc Operating System (RDOS) te kunnen gebruiken moet u dus de beschikking hebben over een 64K-kaart met ZES geheugenbanken (0 t/m 5) (dat is de gewone standaard 64K-kaart). Een floppycontroller heeft maar twee banken, dat gaat dus niet zomaar, er moet wat omgebouwd worden. Het multifunctiebord daarentegen heeft zowel een floppycontroller als geheugenbanken, dat werkt dus weer wel. Bij twijfel even informeren!

De RAM disc gedraagt zich bijna helemaal als de ingebouwde recorder. Alleen, in plaats van de cassette (C) stuurt u de RAM disc (R) aan. U typt dus niet CLOAD maar RLOAD en niet CSAVE maar RSAVE. Verder blijft alles hetzelfde! Met deze opdrachten kunt u programma's en (string-)arrays direct lezen en schrijven uit/in de RAM disc. Ondanks de banken (van 8K) worden langere of kortere programma's normaal geaccepteerd, de RAM disc is volledig continu. Wanneer een naam bij een RSAVE-opdracht al bekend is, volgt netjes de vraag *Hier overheen (j/n)?*, net zoals bij de recorder. Een bijzonderheid van RLOAD en RSAVE is, dat ook blokken geheugen zondermeer gelezen en geschreven kunnen worden, bijvoorbeeld beeldschermen. Dat geeft een aantal erg interessante mogelijkheden.

Op het toetsenbord van de P2000 zit een knop "ZOEK". Die geeft de inhoud van de cassette. Net zo kunt u de inhoud van de RAM disc te zien krijgen met RZOEK. Maar met RZOEK kunt u ook de inhoudsopgave vanuit een programma bekijken (geen ge-USR!) en zelfs door het programma laten lezen. Hierna kan de inhoud bijvoorbeeld gesorteerd worden.

De RAM disc is uitmuntend geschikt om programma's achter elkaar te laten lopen. Als het ene programma "klaar" is, wordt automatisch het volgende ingelezen en opgestart. Dat kan met RCALL. U zult niet eens merken dat er van programma gewisseld wordt, want daarvoor is de RAM disc te snel (ca. 100K per seconde). Met RCALL kunt u ook gegevens doorspelen naar het aangeroepen programma, zodat u werkelijk twee programma's in elkaar kunt laten overgaan. U kunt zelfs het ene programma oproepen als subroutine in een ander! Dat schept heel veel mogelijkheden, bijv. bij informatieprogramma's. Ook is het mogelijk "losse" opdrachten te gebruiken. RDOS past namelijk maar net in het geheugen en toch moesten er nog wat dingen bij. Dergelijke stukjes machinetaal kunt u onder een naam opbergen en volledig automatisch op laten roepen wanneer ze nodig zijn. "Rename" bijv. is zo'n extra functie.

De cassettereader is niet in staat een los programma op de cassette te wissen, alleen een hele kant tegelijk. In de RAM disc kunt u echter wel een bepaalde file uitwissen. Dat gaat met RKILL. De hele RAM disc in een keer kan

overigens ook, met RKILL ALL. Na een RKILL-opdracht wordt de RAM disc automatisch ge-CRUNCH-ed, er vallen zo nooit gaten. Dat CRUNCH-en duurt in het ongunstigste geval een seconde of twee. Het kost u GEEN werkgeheugen.

Om tijdverlies te voorkomen kunt u een hele RAM disc (van maximaal 40K) in een keer op een cassette zetten of van een cassette inlezen. Dat gaat met RDUMP resp. RTAKE. RDOS comprimeert de hele inhoud en zet hem in zo weinig mogelijk blokken op cassette. U moet zo'n cassette dan wel weer met RTAKE inlezen, want de gewone CLOAD begrijpt van zo'n gecomprimeerde RAM disc niks.

Onder RDOS kunt u op een heel gemakkelijke manier met gegevensbestanden werken. Zowel seriele- als random files zijn mogelijk. De spelling van de nodige opdrachten verschilt nauwelijks bij random- of seriele files zodat u niet voor allebei andere opdrachten hoeft te leren.

Voor deze opdrachten is bewust afgeweken van de MicroSoft-standaard met OPEN en CLOSE, omdat die nu niet bepaald uitblinkt door duidelijkheid. Hoewel het werken met files niet moeilijk is, gaat het toch te ver om alles hier even uit te leggen. Voor meer informatie kunt contact opnemen met de auteur.

RDOS is een erg complex programma. Voor de gebruiker werkt het daarentegen (of dus?) heel simpel. Zolang u niets anders doet dan programma's lezen en schrijven heeft u aan deze korte inleiding genoeg. Maar RDOS kan veel, veel meer dan ooit in zo'n kort verhaaltje te vertellen is. Daarom is er een zeer uitgebreide handleiding uitgebracht van 21 pagina's A4. Hierin staan alle mogelijkheden, zisksprongen, achterdeurtjes, opdrachten en systeemgegevens netjes gerangschikt. Ook voor de mensen die liever in machinetaal tegen RDOS praten is er genoeg voorhanden. Verder wordt er over RDOS regelmatig iets gepubliceerd (zie bijv. TRON) en verschijnen er ook vaak nieuwe systeemopdrachten in de vorm van zogenaamde systemfiles. Die kunt u zelf aankoppelen aan uw RDOS, zodat het systeem weer meer kan. RDOS heeft, en u kunt zelf nieuwe toepassingen en mogelijkheden uitdenken! Al deze nieuwtjes worden bijgehouden in het P2000-datanet, vooral in Simpeltel-Rotterdam. Telefoon 010-4379696, 1200-75 Baud, 24 uur per dag bereikbaar. Als u een (viditel-)modem heeft, beslist eens bellen!

Wanneer u RDOS aanschaft, ontvangt u de handleiding, het hoofdsysteem en een heel stel demo- en hulpprogramma's zoals een speciaal indexprogramma. Verder een aantal systemfiles die de mogelijkheden aardig opkrikken. En u heeft altijd recht op nieuwe versies (tegen portokosten) en service bij problemen. RDOS kost inclusief verpaknings- en verzendkosten F 35,- zonder cassette. Die zult u dus zelf op moeten sturen of af moeten geven. Nogmaals: eenmaal aangeschaft blijft u vrijwel gratis op de hoogte van alle ontwikkelingen!

Voor verdere informatie en bestellingen kunt u terecht bij:

Jeroen Hoppenbrouwers - Wilhelminapark 8 - 5554 JE VALKENSWAARD  
Telefoon (04902)-13808 (19-21 uur of in het weekend) Vidibus 400021237

## Handleiding bij het programma RDOS versie 3.1

### **Wat is RDOS?**

Een van de meest aansprekende dingen bij het werken met de P2000T is de ingebouwde mini-cassettorecorder. In vergelijking met andere "thuis-computers" heeft de P2000T daarmee namelijk een goedkoop en toch redelijk snel opslagmedium voor te bewaren gegevens. Het enige wat er op aan te merken valt is die snelheid: niet zozeer het inladen of wegschrijven van programma's, maar het zoeken naar de juiste plaats op de cassette vergt nogal wat tijd. Het enige alternatief is de aanschaf van een floppydisc-drive, maar dat is nu niet bepaald een goedkope oplossing.

Nu is er sinds enige tijd een extra-grote uitbreidingsprint voor de P2000 op de markt, waarmee het geheugen maar liefst 102 kilobytes groot wordt. Dat klinkt natuurlijk leuk, maar een groot gedeelte van dat geheugen is onbruikbaar voor die mensen die geen machinetaal beheersen, en dat zijn er nogal wat. Hoe komt het dat dat geheugen niet te benutten is? Wel, de ontwerpers van de print hebben het idee gekregen de geheugencapaciteit flink op te voeren door het aanbrengen van zes geheugenbanken van acht kilobytes elk. Van die zes banken (0 tot en met 5) kan er telkens maar een ingeschakeld zijn, en zodoende is het onmogelijk om zondermeer gegevens daarin op te slaan: niet alleen de plaats van de gegevens, maar ook het banknummer moet opgegeven worden, en dat extra nummer zit niet standaard in de BASIC-insteekmodule.

Het programma RDOS 3.1 is nu in staat deze bankkeuze wel voor U uit te voeren, zodat BASIC de beschikking krijgt over  $32+40=72$  kilobyte vrij geheugen. Dat is natuurlijk aardig, maar wat voor programma's hebben nou zoveel geheugen nodig? Dat blijken er niet veel te zijn en daarom doet RDOS iets speciaals met al dat geheugen.

De zes "loze" banken worden namelijk omgebouwd tot RAM-disc. U kunt dit opvatten als een floppy-disc-drive in de P2000, die niet met een magnetische schijf werkt maar met "gewoon" geheugen. Zodoende hoeft er nooit meer gezocht te worden naar bepaalde gegevens, en de tijd van inlezen of wegschrijven is vrijwel tot nul gereduceerd. Er komen helemaal geen mechanische handelingen meer aan te pas, alles is zuiver machinetaal en dus verschrikkelijk snel. Het enige nadeel van een RAM-disc is de vluchtigheid: als U de P2000 uitschakelt zijn alle opgeslagen gegevens weg. De RAM-disc vervangt dus de ingebouwde recorder niet, hij vult hem aan op de momenten dat die recorder door snelheidsgebrek verstek laat gaan.

Bovendien zijn met RDOS programmeer-grapjes mogelijk die normaliter alleen met een dure discdrive kunnen of zelfs daar niet mee.

Al met al is RDOS versie 3.1 dus een waarde aanvulling op de BASIC-module!

### Het werken met RDOS.

Om op een redelijke manier alle mogelijkheden van het programma te kunnen gebruiken volstond het niet om RDOS op de bekende manier onder een USR of een functietoets te zetten: de aanroep met ?USR(0) bijvoorbeeld (standaard voor machinetaalprogramma's) is wel simpel maar biedt gewoon te weinig mogelijkheden, terwijl het bijmaken van functietoetsen problemen geeft met allerlei andere programma's (zoals bijvoorbeeld Gereedschapskist).

Daarom is gekozen voor het toevoegen van extra BASIC-commando's aan de al bestaande 100. Deze opzet biedt maximale flexibiliteit bij het maken van een nieuw programma (de commando's kunnen gewoon in de LIST-ing worden opgenomen) en geven vrijwel nooit problemen met andere programma's die ze niet gebruiken.

Om deze extra BASIC-woorden bij Uw P2000 in te bouwen volstaat het om het programma "RDOS 3.1" in te laden en te starten. Het programma wordt dan vanzelf opgeborgen in een van de zes banken, de resterende vijf banken worden geformatteerd (geschikt gemaakt voor het opnemen van gegevens) en de koppeling van RDOS aan BASIC wordt aangebracht. Het enige "speciale" dat U hiervan merkt is het vrije geheugen: de P2000 wordt teruggebracht van de 48K naar de 32K-uitvoering. Dit is een noodzakelijk kwaad om met de banken te kunnen schuiven. Verder is bij het standaard geleverde programma de "bliksemtoets" op het kleine toetsenbord als extra functietoets gedefinieerd om snel de inhoud van de RAM-disc op het scherm te kunnen zetten. Deze toets wordt weer uitgeschaakt als er andere functietoetsen bijkomen.

U bent vrij om het standaardprogramma aan Uw eigen wensen aan te passen: zo kunt U bijvoorbeeld meer functietoetsen toevoegen, extra RDOS-opdrachten bijplaatsen etcetera. Juist om deze reden wordt RDOS geleverd zonder copieerbeveiliging: bij gebleken misbruik wordt het weer ouderwets star en onaanpassbaar. Het is dus in Uw eigen belang RDOS niet verder te verspreiden. Verknal het niet voor iedereen en copieer alleen voor Uzelf!

### De basismogelijkheden van RDOS 3.1.

In de nu volgende paragrafen zullen de verschillende RDOS-commando's in een opbouwende volgorde en aan de hand van voorbeelden worden uitgelegd. Als U alleen geïnteresseerd bent in programma-opslag kunt U heel wat overslaan, maar bepaalde dingen moet U toch beslist weten. Misschien is het dan raadzaam om even te informeren naar RDOS versie 2.0, dit is een eenvoudiger programma dat werkt met een keuze-menu, zodat er bijna niets mis kan gaan.

Iedereen die langer dan twee minuten achter de P2000 heeft gezeten weet hoe een programma van cassette ingeladen moet worden: U typt CLOAD "Naam" en de rest gaat vanzelf. In principe is dit al een hele vooruitgang vergeleken bij andere computers! De "C" in CLOAD komt van "Cassette". Wat ligt er dan meer voor de hand om iets uit de RAM-disc te laden dan die "C" te vervangen door de "R" van RAM? Dus RLOAD "Naam" werkt precies zoals CLOAD, alleen wordt het programma nu niet van de cassette gehaald maar van de RAM-disc. Het is wel even wennen, want de snelheid waarmee dit gebeurt grenst aan het ongelooflijke: een programma van 20 blokken op cassette zit in minder dan tweetiende seconde in het geheugen!

Het logische vervolg is dan de opdracht **RSAVE "Naam"**, waarbij "Naam" de naam is waaronder het programma in de RAM-disc moet worden opgeslagen. Ook hier weer een akelige snelheid.

De inhoud van de RAM-disc wordt op het scherm gezet met de opdracht **RZOEK** (ook in een programma mogelijk), in precies dezelfde opmaak als de toets ZOEK op het kleine toetsenbord produceert: alleen de resterende vrije geheugenruimte en het aantal opgeslagen file's worden extra vermeld (een file is een blok gegevens, dat kan ook een programma zijn). Als U de standaardversie van RDOS gebruikt werkt de "Bliksemtoets" hetzelfde als RZOEK.

Dus **<SHIFT>-<ZOEK>** geeft de inhoud van de cassette en  
**<SHIFT>-<BLIKSEM>** geeft de inhoud van de RAM-disc.

Als U zoveel file's opgeslagen hebt dat de lijst te lang wordt om in een keer op het scherm te kunnen, wordt hij gesplitst op de manier die U van LIST-ings gewend bent: U kunt hem regel-voor-regel, per pagina of geheel doordraaien, en zelfs (met de printertoets) op papier zetten.

Het aantal vrije bytes wordt normaal in geel gegeven, op bepaalde momenten kunnen ze echter in magenta verschijnen. Hierover later meer.

De ingebouwde cassettereorder is niet in staat om een programma op de cassette te wissen: alleen met behulp van een hulpprogramma (of door zelf gebruik te maken van machinetaal) is hier en daar wat te bereiken.

Een in de RAM-disc opgeslagen file kan echter zonder meer uitgewist worden, terwijl de vrijgekomen ruimte automatisch weer benut wordt voor de opslag van de volgende programma's! Dit gaat met de opdracht **RKILL "Naam"**. De opgegeven file wordt uit de RAM-disc verwijderd, U bent hem dus definitief kwijt (tenzij hij op cassette of in het BASIC-geheugen staat, natuurlijk). Met "verwijderd" bedoel ik ook "verwijderd": er valt geen gat, maar RDOS schuift alles netjes aan. Om snel de gehele RAM-disc te wissen kan ook de variant **RKILL ALL** gebruikt worden. Dit commando is overigens ook bedoeld om de RAM-disc te formatteren als het RDOS-programma net is opgestart. Doet U dit niet, door bijvoorbeeld in het standaardprogramma RKILL ALL te schrappen, dan is er een kans dat het systeem vastloopt of de mededeling "0 bytes vrij- 0 file's opgeslagen" geeft. Als er al een geformateerde RAM-disc in het geheugen aanwezig is, dan hoeft de opdracht RKILL ALL natuurlijk niet gegeven te worden: zo spaart U volgeladen banken als het RDOS-systeemprogramma ooit eens mocht sneuvelen (die kans is zeker niet nul). Het opstarten van het RDOS-programma zonder de RKILL ALL-opdracht heeft namelijk geen enkele invloed op reeds geladen programma's.

Wanneer U alleen belangstelling hebt voor de "opgevoerde recorder" die U met RDOS verkrijgt en alleen programma's wilt lezen en schrijven, dan weet U nu voldoende om met het systeem om te kunnen gaan. Voor de programmeerverslaafden is er met RDOS nog een heleboel meer mogelijk. Hierover gaat het volgende hoofdstuk.

### *Heer mogelijkheden met RLOAD en RSAVE, array's en blokken geheugen.*

De standaard RDOS lees- en schrijfcommando's kunnen met kleine wijzigingen ook gebruikt worden voor het opnemen en inlezen van gegevensbestanden in of van de RAM-disc. In de meeste gevallen bestaat een bestand bij de P2000 uit een numeriek array, dat in zijn geheel naar cassette wordt geschreven (bij de standaard-cassette-BASIC is dat zelfs de enige mogelijkheid). RDOS is in staat om ook dergelijke file's te verwerken: op de bekende manier wordt met *RSAVE\** een array naar de RAM-disc geschreven. Alles is weer precies zoals bij de standaard CLOAD\*- en CSAVE\*-opdrachten. Een voorbeeld:

met *RSAVE\* A* wordt het numerieke array A gekopieerd in de RAM-disc onder de naam "A". Er mag ook weer een naam opgegeven worden, dat gaat met *RSAVE\* A @"Naam"*. Voor bijzonderheden: zie de BASIC-handleiding.

Zo is het mogelijk zeer snel van bestand te wisselen, dat kan erg veel tijd sparen (zeker bij programma's als bijvoorbeeld Minitext).

Het terug inlezen van een in de RAM-disc opgeslagen array gaat met bijv. *RLOAD\* A @"Naam"*. Voordat deze opdracht wordt uitgevoerd moet eerst genoeg ruimte ge-DIM-d zijn, anders volgt *Out of memory*. Is het te laden array helemaal niet ge-DIM-d, dan verschijnt *Illegal function call*.

De standaard-BASIC is niet in staat om string-array's weg te schrijven of in te lezen, omdat dat nogal wat werk is (de strings staan helemaal verspreid door het geheugen en moeten eerst bij elkaar geveegd worden). Met het hulpprogramma "Stringsave" kunt U de BASIC dit wel laten doen. Omdat stringarray's een prachtige opslagmogelijkheid voor gegevens vormen is RDOS hier ook mee uitgerust. De opdracht *RSAVE\* A\$ @"Naam"* werkt dus precies zoals dat moet. Omdat de standaard stringsaveroutines ongewijzigd in RDOS zijn opgenomen was het meteen mogelijk ze "los" aan de BASIC te hangen. Daarom werken CLOAD\* en CSAVE\* ook meteen op een stringarray als RDOS opgestart is!

Deze routines zijn echter nogal traag, zodat het wel enige tijd kan duren voordat een groot array ge-SAVE-d is (laden gaat wel snel). Maak daarom geen stringarray's van meer dan ongeveer vijfhonderd strings! Bovendien kost het U een boel geheugen, want de stringsaveroutines hebben een zeer grote stringruimte nodig. Wilt U echt veel gegevens opslaan? Lees dan maar rustig verder.

Op dit punt aangekomen verlaten we de mogelijkheden van BASIC en komen we terecht in de extra commando's van RDOS.

Om een blok geheugen even te bewaren was tot nu toe altijd een stukje machine-taal nodig, omdat BASIC daarvoor veel te traag is. Snelle schermwisselingen bijvoorbeeld moesten altijd in machinetaal geprogrammeerd worden. RDOS doet het voor U! Wilt U een beeldscherm bewaren? Typ *RSAVE "Scherm", &H5000, &H5757* en het is gebeurd (bij de P2000 zit het videogeheugen tussen deze adressen). Na de naam moet het beginadres van het te SAVE-n blok geheugen volgen, gevolgd door het eindadres. Het eindadres wordt ook nog meege-SAVE-d. Geeft U geen eindadres op, dan neemt RDOS aan dat U tot en met &HFFFF wilt wegschrijven. In de meeste gevallen levert dat *Out of memory* op omdat er geen plaats voor is.

Het weer terug inladen van een blok geheugen gaat natuurlijk met RLOAD. Hier zijn een paar verschillende mogelijkheden: *RLOAD "Scherm"*, (let op de komma!) laadt het blok geheugen in op de plaats waar het vandaan kwam, hier &H5000. *RLOAD "Scherm", &H7000* zet het blok geheugen neer op de gewenste plek. Bij het weer terugladen van een blok geheugen moet U natuurlijk erg goed uitkijken waar het terecht komt! Een verkeerd adres kan de P2000 volkomen op tilt zetten. Om de kans op dergelijke missers wat te verkleinen kunt U ook bij een RLOAD-opdracht een eindadres meegeven. Dit dient alleen als grenswacht: bij dreigende overschrijding wordt de opdracht niet uitgevoerd en komt er *Out of memory* op het scherm. Dit geeft al wat beveiliging tegen fouten. RLOAD "Scherm", &H5000 en RLOAD "Scherm", &H5000, &H5757 doen dus precies hetzelfde, maar de tweede opdracht weigert files die langer zijn dan &H757.

### *Filetypen en extensies.*

RDOS geeft aan elke opgeslagen file automatisch een zogenaamde *extensie* mee, een uitbreiding van de filenaam. Deze extensie geeft aan wat voor type file het is. Een BASIC-programma staat nl. geregistreerd als BAS, een enkele-precisie-array als SNG (single) etc. Dit zijn de standaard extensies die BASIC ook gebruikt. Losse blokken geheugen worden opgenomen onder MEM(ory).

Het is gebleken dat veel mensen het nut van die extensies onderschatten. Dat komt vermoedelijk omdat vrijwel alle file's op cassette de extensie BAS meekrijgen (het zijn tenslotte BASIC-programma's). De gelukkigen onder ons die met een floppydisc-systeem kunnen spelen zullen het verschijnsel extensie ongetwijfeld beter kennen en gebruiken. Het zou namelijk niet de eerste keer zijn dat mensen een array inladen als programma en dan natuurlijk prompt tegen een syntax- of andere fout aanlopen, zonder ooit te weten waarom. De kans dat dit met een disc-systeem gebeurt is veel groter, omdat daar veel meer filetypen mogelijk zijn (BASIC, ASCII, machinetaal, bestanden, bootstrap-programma's enzovoort), en omdat RDOS nogal veel van een disc-systeem weg heeft is het zeer aan te bevelen die extensies goed in de gaten te houden. Ook bij RDOS kunnen namelijk veel verschillende filetypen voorkomen.

Om zelf enige grip te hebben op die extensies kunt U een eigen uitbreiding aan een ingegeven naam hangen. Dat doet U net zoals bij CSAVE door achter de naam een CHR\$(0) toe te voegen en daarachter drie willekeurige tekens te typen. Bijvoorbeeld RSAVE "Programma"+CHR\$(0)+"PRG"; het BASIC-programma "Programma" krijgt dan niet de standaard-extensie BAS maar de aanduiding PRG. Ga hierin ook weer niet te ver, het moet een file-type blijven, anders ziet U straks door de bomen het bos niet meer en worden de problemen alleen maar groter. Maar kijk altijd eerst of U wel de juiste RLOAD-opdracht geeft voor de gewenste file. RDOS trekt zich namelijk nooit iets aan van extensies om U zoveel mogelijk programmeervrijheid te geven.

Ook wordt van elke file een zgn. applicatieteken onthouden, meestal is dat een "B". Bepaalde machinetaalprogramma's kunnen hun eigen teken meegeven, maar voor RDOS maakt dat allemaal niets uit.

### *Filenamen en afkortingen.*

De ingebouwde cassettereorder kijkt bij het zoeken van een programma of bestand alleen naar de eerste letter van de naam (waarschijnlijk uit gemakzucht) en dat levert nog wel eens problemen op.

RDOS doet het wat anders. Wanneer U een naam opgeeft worden namelijk alle ingegeven tekens gecontroleerd. U kunt dus rustig de files "Prog1" en "Prog2" tegelijk in de RAM-disc zetten, het zijn verschillende namen. Typt U RLOAD "P" dan verschijnt *Niet gevonden*, want "P" komt niet voor in de RAM-disc. Dus altijd de volledige naam intypen! Omdat dat lastig is kunt U een naam ook afkorten. Dat gaat met een "\*", bijvoorbeeld RLOAD "Pro\*". RDOS bekijkt dan van elke naam alleen de eerste drie tekens. De file met een overeenstemmende naam die het hoogst in de inhoudsopgave staat (kijk met RZOEK) wordt dan geladen.

Wordt bij RSAVE een al bekende naam opgegeven, dan vraagt RDOS *Hier overheen?* en wordt de oude file al dan niet overschreven. Was de naam die opgegeven werd afgekort, dan wordt hij vervangen door de naam zoals die in de RAM-disc staat. Er ontstaan overigens geen problemen als de nieuwe file langer of koper is dan de oude; RDOS schuift net zolang totdat alles weer precies past.

Met POKE &H60AC,1 (de "cassette-POKE") wordt deze vraag eenmalig onderdrukt.

## Het automatisch verwisselen van programma's

Omdat U RDOS-commando's in de LIST-ing kunt opnemen is het natuurlijk ook heel goed mogelijk programma's zichzelf te laten verwisselen. Dat is bijvoorbeeld nuttig als U een hoofdprogramma heeft gemaakt waar een informatieprogramma bijhoort, of wat niet in zijn geheel in het geheugen past. Zonder RDOS bent U dan verplicht op de recorder te wachten, met de RAM-disc gaat het verwisselen ongemerkt!

**RCALL "Naam"** haalt het gevraagde programma uit de RAM-disc en voert dat direct uit. Ook de cassette-BASIC kent zo'n commando: RUN "Naam" doet hetzelfde, alleen de bron verschilt. U kunt op een willekeurige plaats in het programma een RCALL-opdracht opnemen, zodat een programma in feite een lengte van  $40+32=72$  kilobytes mag hebben.

Het kan wel eens voorkomen dat bepaalde gegevens vanuit het eerste programma doorgegeven moeten worden aan het tweede. Als die gegevens in een array staan is het simpel genoeg: RSAVE het array, RCALL het tweede programma en RLOAD het array daar weer. Maar met losse variabelen gaat het zo niet.

Daarom kunt U met RDOS 3.1 bij elke RCALL-opdracht variabelen meegeven. Bijv. **RCALL "Ander prg",A,B,C\$**. Direct na de opstart van het andere programma zijn de variabelen A, B en C\$ onder dezelfde naam beschikbaar. Dan mag het aangeroepen programma natuurlijk niet met een CLEAR beginnen! Maximaal kunt U 1 K aan variabelen doorgeven. Het moeten wel "normale" variabelen zijn, dus geen array-elementen! Door het andere programma te beginnen met een ON ... GOTO-opdracht kunt U zelfs opgeven waar in het andere programma begonnen moet worden.

Het nadeel van de bovengenoemde werkwijze is de onomkeerbaarheid: U kunt wel een ander programma aanroepen, maar weer terugkeren in Uw oude programma op de plaats waar U het verliet is (bijna) onmogelijk. Toch zou dat erg veel extra programmeergemak geven: U zou dan bijvoorbeeld op willekeurige momenten een informatieprogramma kunnen aanroepen, zonder dat dat in het werkgeheugen van de P2000 staat en dus het hoofdprogramma minder ruimte laat! Nu, met RDOS kan het: **RCALL SUB "Naam"** roept het betreffende programma aan als subroutine. Het aanroepende programma wordt met al zijn variabelen en array's ge-SAVE-d in de RAM-disc en daarna wordt het sub-programma geladen en opgestart. Om het oude programma op te bergen moet er natuurlijk wel genoeg ruimte zijn. Zoals al verteld mag U ook bij RCALL SUB variabelen opgeven die doorgelust worden naar het aangeroepen programma.

Het enige dat door RCALL SUB wordt aangetast is de zgn. *BASIC stack*, waar de P2000 o.a. terugspringadressen van FOR-NEXT en GOSUB bewaart. Zorg er dus voor dat er geen subroutines of lussen meer "openstaan" als U RCALL SUB laat uitvoeren. Alle normale programmagegevens zoals variabelentypen (DEFSTR etc.) blijven onaangeroerd.

Er moet natuurlijk ook weer teruggekeerd kunnen worden uit een SUB-programma. Dat gaat met de opdracht **RCALL RET** (zonder naam). De P2000 zoekt dan zijn oude programma weer op, laadt dat in, plakt alle variabelen en andere gegevens op hun plaats en gaat vrolijk verder na de RCALL SUB-instructie. En ook bij RCALL RET mogen weer variabelen opgegeven worden.

Wanneer de computer een RCALL RET-opdracht tegenkomt terwijl er geen returnfile in de inhoudsopgave staat, m.a.w. als er geen RCALL SUB aan voorafgegaan is, volgt de foutmelding *RETURN without GOSUB*. Het is niet mogelijk RCALL SUB-opdrachten te "nesten", dat wil zeggen een RCALL SUB binnen een met RCALL SUB aangeroepen programma. Alleen de laatste SUB-opdracht is geldig.

### Systemfiles en "losse" BASIC-commando's

U heeft beslist wel eens kleine machinetaalroutines opgeborgen in de beschermde ruimte boven het BASIC-geheugen. Programma's als BASICODE, Flexkist, Audio-cassette enzovoort worden in negen van de tien gevallen door een BASIC-programma opgestart en "los" achtergelaten in de top van het geheugen. Met een ?USR(0)-opdracht zijn ze dan aan te roepen. Dit geeft aanleiding tot vragen als *Wilt U het programma in de top van het geheugen?* of *Welk nummer wilt U voor de USR-functie gebruiken?* En het over elkaar heen zetten van programma's -zeg maar "stapelen"- leidt al gauw tot chaos.

Natuurlijk is er een hoop op te lossen door die BASIC-opstartprogramma's in de RAM-disc te laden en als U ze nodig heeft even te RCALL-en. Maar dat kost een hele hoop onnodige geheugenruimte; er zit praktisch altijd informatie bij en tekst vreet geheugen. Het zou veel efficienter zijn alleen de machinetaal op te slaan. En het zou nog mooier zijn die machinetaal niet permanent boven in het geheugen te plakken, maar ergens anders, zodat de verschillende programma's elkaar nooit in de weg zitten en een geladen stukje machinetaal niet door een CLEAR-commando weggepoetst kan worden. Maar waar vinden we zo'n stuk "veilig" geheugen? Het antwoord is eigenlijk erg simpel: in de RAM-disc! Bij disc-aanhangers gaat er nu een lampje branden. Een beetje disc operating system kent de zgn. *system file* of "*C0H*"-file. Dat is een stuk machinetaal, dat "kaal" op de schijf staat onder een naam die wat zegt over de functie. Heeft U die speciale functie nodig, dan typt U de naam ervan in en het DOS zorgt ervoor dat de machinetaal op de juiste plaats in het geheugen terechtkomt en meteen wordt uitgevoerd. Zo kunnen nuttige, maar weinig gebruikte instructies uit het hoofdgeheugen blijven (zuinig!) en toch bereikbaar zijn.

Iets dergelijks kent RDOS ook. Hoog in het geheugen, ver boven de BASIC-ruimte en de gereserveerde ruimte, is een buffer vrijgehouden voor o.a. systemfiles (hij zit van &HFC00 tot &HFFFF). Na de opdracht *RCALL SYS "Naam"* wordt de file "Naam" in de sysfileruimte geladen en als machinetaal uitgevoerd. U moet er zelf op letten of het ook inderdaad machinetaal is! Daarom heeft een sysfile altijd de extensie *SYS*. Let daar heel goed op!

Voor sommige sysfiles is het nodig dat U een bepaald gegeven aan die sysfile doorgeeft, bijvoorbeeld een naam. Dat kan op twee verschillende manieren:

- De sysfile voert een bewerking op het gegeven uit, dat dus veranderd terugkomt. In dat geval moet het gegeven natuurlijk losse variabele zijn omdat RDOS anders niet weet waar het moet blijven. De aanroep van een dergelijke "functie-sysfile" is bijvoorbeeld *RCALL SYS "Naam", A*.
- De sysfile heeft een gegeven nodig, maar laat dat ongemoeid. Een voorbeeld is de naam van een te behandelen file. In dat geval hoeft er niets teruggegeven te worden en mag dus ook een expressie of constante ingetypt worden. De aanroep verschilt slechts door het gebruik van een ":" i.p.v. een ",": *RCALL SYS "Naam"; "Filenaam"* of iets dergelijks.

Omdat *RCALL SYS* vrij lang is en toch vaak gebruikt wordt kunnen deze twee woorden samen worden vervangen door "\$". \$"Naam";"Filenaam" is dus hetzelfde als *RCALL SYS "Naam";"Filenaam"*. Het enige verschil tussen een "vaste" RDOS-instructie en een "losse" is dus dat \$-tekentje en de " "'s om de naam. Overigens is niet bij alle sysfiles een gegeven nodig en kan ook het aantal gegevens per sysfile verschillen. Dat hangt helemaal van de functie af.

Omdat een sysfile machinetaal is en geen BASIC, mag U hem niet met CLOAD-RSAVE of RLOAD-CSAVE van of naar de cassette sturen. U dient een speciale cassette-opdracht te gebruiken. Dat vindt U verderop bij de cassettereorder-instructies.

### Het zelf maken van systemfiles

Wanneer U overweg kunt met machinetaal is het erg goed mogelijk zelf aanvullende commando's bij te maken. Daarvoor moet U het volgende weten: De sysfile moet kunnen werken op adres &HFC00, maximaal 1 kilobyte lang. Als er aan de systemfile begonnen wordt staat het eventueel meegegeven argument in de Floating Point Accumulator, net zoals bij de aanroep met USR. De registers A en HL zijn ook op dezelfde manier gevuld: A=type, HL=adres FAC. Werd er geen argument meegegeven, dan zijn zowel A als HL gelijk aan nul. Bij de terugkeer kijkt RDOS naar de inhoud van A: is die nul, dan was alles oke, maar is die niet gelijk aan nul dan verschijnt de normale BASIC-foutmelding met het nummer dat in A staat. Hierbij maakt het niets uit of de fout groter of kleiner is dan 64 (STOP), RDOS verwijst de melding automatisch juist door. Is A gelijk aan nul (geen fout dus), dan wordt de inhoud van de FAC op dat moment al- of niet teruggeplaatst in de meegegeven variabele en de controle terugegeven aan RDOS, dat op zijn beurt BASIC in gang zet.

Wilt U meer dan 1 gegeven aan de sysfile doorgeven, dan moet U zelf BASIC gaan uitdecoderen. Dat gaat als volgt: POP eerst het returnadres van de systemfile (dat mag weg). Vervolgens POP HL. In HL staat dan het adres van de eerste letter van de eerste parameter. Hierna kunt U zelf aan de slag. U moet nu wel alles zelf doen, zoals variabelen opzoeken e.d. Ook foutmeldingen moeten weer "ouderwets" worden aangeroepen met een JP. Na het beeindigen van de systemfile moet in HL het adres van het scheidingsteken tussen twee statements staan (nul of : dus), de plaats waar BASIC nu verdergaat. De uitvoering van het BASIC-programma wordt dan vervolgd met JP E41B (dus niet met RET!).

De sysfiles zijn vooral bedoeld voor aanvullende commando's en functies en niet om er bijvoorbeeld een spelletje in te zetten (daar zijn ze trouwens te kort voor). In de loop van een paar jaar (?) zullen er beslist vele het licht zien, en daarom ben ik van plan ze te gaan verzamelen of in ieder geval bij te houden welke er verkrijgbaar zijn en bij wie. Heeft U een nuttige of leuke systemfile gemaakt, stuur hem dan op of geef door dat hij er is. Regelmäßig kan er dan een lijst verschijnen met nieuwertjes.

### Werken met de cassettereccorder

U kunt een hele RAM-disc in een keer op cassette zetten of van cassette halen. Het wegschrijven naar cassette gaat met het commando RDUMP. De gehele inhoud van de RAM-disc wordt gecomprimeerd en in zo weinig mogelijk blokken op cassette gezet. Een dump komt vanzelf achter het laatste programma op de cassette. Wordt er op de cassette een oude dumpfile gevonden, dan komt de vraag *Hier overheen?*, net zoals bij CSAVE of RSAVE. De vraag *Hag de rest weg?* wordt echter nooit gesteld, omdat het teveel tijd kost "even" naar het einde van de oude dump te spoelen. Uit besparingsoverwegingen wist een dumpfile altijd de programma's die verder op de cassette staan: U mag dus wel iets achter zo'n dumpfile zetten, maar het verdwijnt weer als een nieuwe dump op de cassette wordt geschreven, ook al is die dump even lang of korter dan de oude!

Met RTAKE wordt een complete inhoud van cassette gehaald en in de RAM-disc opgeslagen. Daarbij gaat een eventuele oude disc natuurlijk verloren! RTAKE slikt alleen "voorgebakken" dumpfiles, een normaal BASIC-bandje werkt zo niet (hier is wel een sysfile voor). Treedt er bij het laden een cassettefout op, dan kunnen gedeeltes van de RAM-disc vernietigd zijn maar de inhoudsopgave geeft dat niet aan! Door even proberen kunt U de schade snel vaststellen: files die voor de fout zijn ingeladen blijven onbeschadigd.

Als U zeker weet dat een in te laden file verderop de cassette staat, kunt U het terugspoelen onderdrukken door POKE &H60AC,1. Dit werkt maar een keer, U moet deze POKE voor elke RTAKE herhalen. Om veiligheidsredenen werkt dit grapje niet met RDUMP (U zou per ongeluk midden op de band kunnen gaan schrijven, de dump is dan niet meer terug te vinden). Toch heeft POKE &H60AC,1 bij RDUMP wel effect: net zoals bij RSAVE kunt U daarmee de vraag *Hier overheen?* overslaan, zodat een eventuele oude dumpfile direct overschreven wordt. Ook hier moet de POKE voor elke nieuwe opdracht herhaald worden.

Normaliter moeten, om een losse file van cassette in de RAM-disc te krijgen, twee opdrachten gegeven worden, namelijk een CLOAD en een RSAVE. CLOAD heeft echter als nadeel, dat de BASIC-interpreter zich met de file gaat bemoeien: hij neemt aan dat de ingeladen file een programma is en wijzigt iets in de regelinidelijng. Als de file geen BASIC-programma is maar bijvoorbeeld een kale systemfile wordt hij zo grondig verknoeid. Bovendien maakt CLOAD natuurlijk gebruik van het programmagheugen van de P2000 en bent U een eventueel al geladen programma kwijt.

Met de twee opdrachten RDUMP "Naam" en RTAKE "Naam" kunt U nu een file rechtstreeks van de RAM-disc naar de cassette schrijven of vice versa. Hierbij wordt de filenaam niet veranderd, ook extensie en applicatieteken blijven ongewijzigd. Beide opdrachten maken gebruik van de vrije geheugenruimte na het BASIC-programma (incl. variabelen en array's), Uw programma wordt dus in het geheel niet aangetast. Het kan echter voorkomen dat er niet genoeg vrije ruimte meer over is. In dat geval zit er niets anders op dan het aanwezige BASIC-programma te wissen (of even in de RAM-disc te zetten). RDUMP "Naam" en RTAKE "Naam" maken gebruik van de BASIC-load- en save-routines, een applicatieteken moet bij RTAKE "Naam" dus "B" zijn. Het applicatieteken van een met RDUMP (zonder naam) gemaakte file is "R", vergissing is daarom niet mogelijk. En ook hier kan weer met POKE &H60AC,1 het terugspoelen bij RTAKE "Naam" tegengehouden worden. Bij RDUMP "Naam" heeft deze POKE geen effect.

Zowel RTAKE als RDUMP geven de normale cassettefouten af als daar aanleiding toe is. Deze fouten kunnen op de standaardmanier met ON ERROR GOTO worden afgevangen.

#### *Zelf uitlezen van de inhoudsopgave*

Met RZOEK komt de inhoud van de RAM-disc op het scherm, dat wist U al. Maar het is zo niet mogelijk om vanuit een programma te kijken welke files er in de RAM-disc staan, of hoe lang ze zijn, of hoeveel ruimte er nog vrij is. Daarom is het mogelijk met de opdracht RZOEK de directory (inhoudsopgave) van RDOS af te tasten.

Het aantal vrije bytes in de RAM-disc kunt U te weten komen door bijvoorbeeld RZOEK FRE A. Na deze instructie staat in de variabele A het resterende aantal bytes. Omdat de P2000 ook negatieve getallen weer moet kunnen geven, klopt deze waarde niet altijd: boven de 32768 vrije bytes komt er ineens een negatief getal uit. Door daar 65536 bij op te tellen verkrijgt U de juiste waarde. Met een kleine formule wordt deze correctie automatisch uitgevoerd wanneer hij nodig is: RZOEK FRE A: A=A-65536\*(A<0).

Alle gegevens over een bepaalde file (volledige naam, extensie, lengte) zijn als volgt op te vragen: stop de naam (evt. afgekort en gevolgd door \*) in een stringvariabele, bijv. A\$, en dan RZOEK A\$. Wordt de naam niet gevonden, dan komt A\$ leeg terug (A\$=""). Wordt hij wel gevonden, dan heeft de stringvariabele altijd de volgende opbouw:

- 16 tekens naam (evt. aangevuld met spaties), gevolgd door een spatie;
- 3 tekens extensie, gevolgd door een spatie;
- 5 tekens lengte in ASCII, rechts uitgelijnd.

Zo'n string zou er bijvoorbeeld als volgt uit kunnen zien:

Parameterkrommen BAS 512. Elk gegeven heeft altijd zijn vaste positie in de string, met MID\$ en evt. VAL kunt U dus alles te weten komen.

Als laatste is er de mogelijkheid de hele directory in strings te vangen. Dat gaat zo: maak eerst een lege stringvariabele (bijv. LET A\$=""). Dat is voor RDOS het teken om vooraan in de directory te beginnen met lezen. Typ dan: RZOEK NEXT A\$. De eerste filenaam komt dan in A\$ te staan, op de hierboven beschreven manier. U kunt met deze string nu doen wat nodig is. Voor de volgende file laat U A\$ intact (het gaat overigens alleen om de naam) en typt U nogmaals RZOEK NEXT A\$. RDOS geeft dan in A\$ de filegegevens terug van de file die volgt op die met de opgegeven naam. Zo kunt U de hele directory afgaan. Wordt het einde bereikt, dan komt de string leeg terug; nogmaals RZOEK NEXT begint dan weer vooraan.

#### *Het werken met sequentiele- en random files*

Hoewel stringarray's een mooie opslagmogelijkheid voor allerlei gegevens vormen, leveren ze regelmatig problemen op. Het duurt erg lang om ze weg te schrijven, ze nemen relatief veel geheugenruimte in beslag en ze moeten ook altijd in het werk- of programmageheugen staan. Dat bekort zowel hun eigen lengte als die van het bijbehorende programma. Voor de opslag van grote hoeveelheden gegevens, zoals een ledenbestand, zijn ze daarom ongeschikt.

Wat U dan nodig hebt zijn *files*. Normaal is dat alleen mogelijk met een floppy-disc-systeem, met RDOS 3.1 gaat het ook.

Onder RDOS bestaan twee soorten files, de *random*- of willekeurig toegankelijke file en de *sequentiele*- of seriele file. Het technisch verschil tussen die twee is nogal groot en ze hebben allebei een eigen toepassingsgebied. Maar door de bijzondere programmeertaal die ik er voor ontworpen heb worden ze bijna op dezelfde manier geprogrammeerd!

#### *Sequentiele files*

Voor grote bestanden die vaak en veel veranderen en ook voor hele kleintjes is dit filetype het meest geschikt, omdat het erg zuinig is met geheugen en dergelijke bestanden toch meestal in hun geheel worden gelezen, aangepast en weer worden weggeschreven. Het belangrijkste kenmerk van de sequentiele file is namelijk het definitieve karakter: eenmaal geschreven blijft geschreven, veranderingen zijn niet meer mogelijk; verlengingen wel. Als U steeds het hele bestand in een keer inleest is dat natuurlijk geen bezwaar, omdat het dan ook weer in zijn geheel opnieuw weggeschreven kan worden!

U legt een sequentiele file aan (dat heet "openen") met *RFILE "Naam"*. Duidelijker kan bijna niet! Om technische redenen wordt RDOS nu geblokkeerd voor wis- en schrijfopdrachten, maar dat is bijna nooit een bezwaar. Hoe deze toestand weer op te heffen is volgt nog.

Nu kunt U de file gaan vullen. Dat gaat weer met *RFILE*, maar U moet er bij opgeven dat U wilt schrijven. Om bijvoorbeeld het getal 123 in de file te plaatsen typt U *RFILE PUT 123*. Simpel, nietwaar? Dit heet het schrijven van een *record*. Een file bestaat namelijk altijd uit een (groot) aantal onafhankelijke records die ieder een bepaald gegeven bevatten. In dit geval hebt U record 1 met 123 gevuld. Op precies dezelfde manier gaat het met strings: *RFILE PUT "Teststring"*. Alle soorten variabelen of expressies kunnen zo in de file worden opgeslagen, ongeacht hun lengte.

Bij elke volgende RFILE PUT-opdracht wordt het meegegeven record achter aan de rij toegevoegd; daarom heet het een sequentiële- of seriele file! Dit kan net zolang doorgaan totdat het geheugen vol is (40 Kbyte).

U kunt ook verschillende records tegelijk opgeven. Dat gaat met bijvoorbeeld RFILE PUT 123, 1+4, "Test twee", A\$+B\$+C\$. I.t.t. andere systemen hebben de komma's alleen een scheidingsfunctie, ze worden niet mee opgeslagen.

Goed, U heeft alle gegevens weggeschreven naar de file "Naam" en nu wilt U wel eens zien wat er allemaal inzit, of U wilt iets nieuws in de RAM-disc zetten. Maar dat gaat niet zomaar: de P2000 weet nog niet dat U klaar bent met schrijven! U dient de file eerst af te sluiten. Dat gaat met RFILE RES (reset). Op dat moment worden wis- en schrijfopdrachten weer vrijgegeven. RDOS zorgt er voor dat op het einde van een file altijd een afsluitteken staat, geeft U dus de opdracht RDUMP zonder eerst RFILE RES getypt te hebben, dan gaat er niets verkeerd. Bij schrijven heeft RFILE RES alleen de betekenis: "Ik ben klaar, je mag het systeem weer helemaal vrijgeven". Staat er namelijk een file open voor schrijven, dan resulteert een RSAVE- of RKILL-opdracht in de foutmelding *File still open*, omdat deze opdrachten de plaats van de file in de RAM-disc kunnen wijzigen, en dat mag niet. RKILL ALL en RTAKE trekken zich van eventueel openstaande files niets aan, deze zeer destructieve commando's worden natuurlijk niet voor niets gebruikt.

Na een RFILE RES is de sequentiële file automatisch geopend om er in te lezen. Heeft U de file al eerder aangelegd, dan dient U ook de filenaam nogmaals op te geven, door weer RFILE "Naam" in te typen. Als deze naam al voorkomt in de inhoudsopgave "weet" RDOS namelijk dat de file gelezen moet worden, want overschrijven gaat niet!

Het eerste record van de file leest U nu met RFILE GET <variabele>. Let er wel op dat het type van die variabele (INT, SNG, DBL of STR) overeen moet stemmen met het type van het record! Zo niet, dan volgt *Type mismatch*. Dit dient om te voorkomen dat "per ongeluk" een fout over het hoofd wordt gezien. Ook hier kunnen weer meerdere records tegelijk gelezen worden: RFILE GET A,B,C\$,D% etc. De records worden gelezen in dezelfde volgorde waarin ze geschreven zijn, om bijv. het zesde record te lezen moet U dus zes keer RFILE GET laten uitvoeren.

Op een gegeven moment zult U het einde van de rij records bereiken. Meer inlezen gaat dan niet. Om nu te kijken wanneer U moet stoppen is er een speciale hulpvariabele ingevoerd, namelijk EOF (End Of File). Bij elke GET-opdracht wordt deze variabele aangepast: is het einde van de file nog niet bereikt, dan is hij niet waar (0), is het einde wel bereikt, dan wordt hij waar (-1). Controle is dus heel simpel: IF EOF THEN is het einde bereikt (ELSE niet)! EOF is eigenlijk de gewone enkele precisie-variabele EO. Het is duidelijk niet de bedoeling hiermee zelf te gaan werken, omdat de variabele bij elke GET-opdracht overschreven wordt met 0 of -1. Wanneer U met DEFtype het standaard-type (SNG) aanpast, moet U er op letten dan EOF altijd SNG blijft! Dus eventueel "EOF!" gebruiken.

Zodra RDOS gemerkt heeft dat een file ten einde is en dus EOF "waar", wordt de file vanzelf afgesloten. Voert U dan halsstarrig nog een RFILE GET <var>-opdracht uit, dan verschijnt *File not open*.

Om tijdens het lezen, dus voordat het einde bereikt is, opnieuw te beginnen kan ook weer RFILE RES gebruikt worden. Was de file al afgesloten, dan moet hij heropend worden met RFILE "Naam".

### *Verlengen van sequentiele files.*

Een reeds aangelegde en weer afgesloten seriele file kan nog verlengd worden. Hier is wel een beperking aan verbonden: *hij moet als laatste file in de RAM-disc staan*. Met wat kunst-en-vliegwerk of een speciale systemfile kunt U dat voor elkaar krijgen. "Verlengen" of "aankoppelen" is in het Engels "append", om de file te verlengen typt U daarom RFILE "Naam" FOR APPEND. Staat de file toch niet achteraan, dan komt Can't extend als waarschuwing. Staat hij wel achteraan, dan wordt hij gewoon geopend voor schrijven, waarbij het eerste nieuwe record netjes achter het laatste oude record geplakt wordt. Verder is alles precies zoals bij het "normale" schrijven.

### *Het gelijktijdig openen van verschillende files.*

Met RDOS 3.1 kunt U meerdere files naast elkaar en onafhankelijk van elkaar aanleggen. Zo wordt het mogelijk een file rechtstreeks over te copieren, of gegevens uit een file in te lezen en bewerkt op te slaan in een andere file. U kunt files tegelijkertijd open houden door bij de RFILE-opdrachten een filenummer op te geven. Geeft U niets op, dan neemt RDOS aan dat file nummer nul bedoeld wordt. Om bijvoorbeeld de file "Naam" nummer twee te geven typt U RFILE #2, "Naam". Merk op dat een filenummer altijd voorafgegaan wordt door een "#". Om nu uit deze file te lezen moet ook weer het juiste filenummer gebruikt worden: RFILE GET #2, <var>. Laat U #2 weg, dan wordt uit file nul gelezen en het is nog maar de vraag of dat mogelijk is. Bij alle file-opdrachten kunt U een filenummer opgeven: RFILE #3,"Naam" of RFILE PUT #0,"Test" of RFILE RES #2 etcetera. Onder RDOS kunt U vier files tegelijk openen (nummers 0 t/m 3). Voor vrijwel alle toepassingen is dat meer dan genoeg.

Wanneer een oude file weg mag, moet dat altijd met RKILL, omdat RDOS nooit weet wanneer *Hier overheen?* gevraagd mag worden (U zou nooit een file kunnen heropenen!). Een sequentiele file krijgt altijd de extensie SEQ mee, zodat fouten tot een minimum beperkt kunnen blijven. Ook hier geldt weer, dat RDOS bij het her-openen van een file niet naar de extensie kijkt!

### *Random files.*

In sommige gevallen kan het heel handig zijn willekeurig door alle records heen te kunnen lezen en schrijven, in plaats van achter elkaar zoals bij een sequentiele file. Met een random acces file is dat mogelijk. Hieraan zijn wel bepaalde voorwaarden verbonden: elk record moet dezelfde, vooraf vastgelegde lengte hebben. Alleen dan is RDOS in staat uit te rekenen waar een bepaald record te vinden is (recordnummer \* lengte = plaats). En omdat bij het schrijven in een random file niet noodgedwongen vooraan begonnen hoeft te worden, wil RDOS ook graag het hoogste recordnummer van U weten, zodat er genoeg ruimte vrijgemaakt kan worden.

Een random file wordt geopend met bijvoorbeeld RFILE 99, 20, "Naam". In dit geval wordt er ruimte gereserveerd voor 100 records van ieder 20 tekens lang. Record nul doet namelijk ook mee! Omdat nu alles van de file bekend is blijft RDOS volledig functioneren: na een RKILL-opdracht (of als bij RSAVE een oude file overschreven is) is de file echter verplaatst en moet hij even opnieuw geopend worden met RFILE 99,20,"Naam". Het heropenen van een random file heeft geen enkel gevolg voor de inhoud, en er kan dan meteen weer geschreven en gelezen worden. Als een random file voor het eerst geopend wordt zijn alle records gevuld met spaties.

Een andere specifieke eigenschap van de random file is de opbouw van een los record. Dat bestaat namelijk altijd uit een string. Binnen die string worden verschillende velden aangewezen, die ieder een eigen taak krijgen. Om bijvoorbeeld een ledenbestand aan te leggen krijgt ieder record (1 lid) de volgende velden: 25 tekens naam - 20 tekens adres - 6 tekens postcode - 15 tekens woonplaats. Het recordnummer is het lidmaatschapsnummer. Totaal dus  $25+20+6+15=66$  tekens per record. Met MID\$ worden eerst alle gegevens achter elkaar in een string van 66 spatie's geplaatst, en die string wordt dan in zijn geheel in de RAM-disc gezet op de juiste plaats, aangegeven door het recordnummer.

Het schrijven van zo'n record gaat natuurlijk weer met RFILE PUT. Om bijv. op plaats 6 een record neer te zetten typt U RFILE PUT 6, A\$ als A\$ het record bevat. Het is niet zo dat weglaten van het recordnummer record nul inhoudt, dat heeft gewoon nummer 0. Wanneer de lengte van de string niet gelijk is aan het van tevoren opgegeven aantal tekens krijgt U de melding *Bad record length*, ook als de string korter is dan de recordlengte. Er zit dan toch iets fout!

Het weer terug lezen van een record gaat nu wel heel gemakkelijk: gewoon RFILE GET 23, A\$ en record 23 staat in A\$. Daarna kunt U met MID\$ aan de slag om alle gegevens te scheiden.

Na elke random file-opdracht kunt U opvragen hoeveel records de file heeft. Dat gaat met de variabele LOF (Last Of File), die altijd het hoogste recordnummer bevat. Voor EOF gelden precies dezelfde regels als voor EOF. Ook EOF heeft trouwens een functie bij random files: als U met RFILE GET X,A\$ het laatste record van de file heeft gelezen (dus als X=LOF) dan wordt EOF "waar", anders "niet waar". Dit dient alleen als waarschuwing, het gebruik is beslist niet noodzakelijk.

Zowel bij PUT als bij GET mag U weer meerdere gegevens opgeven: RFILE PUT 0,A\$, 1,B\$, 2,C\$, 3,D\$ is toegestaan. Of het programma er duidelijker op wordt...?

Natuurlijk kunnen ook bij random file-opdrachten filenummers gebruikt worden. Geheel volgens het normale patroon krijgt U dan bijvoorbeeld RFILE PUT #2,4,A\$ of RFILE #1,99,20,"Naam".

Zoals al verteld wordt een random file heropend met RFILE (#X), Y, Z, "Naam". Als de recordlengte Z niet klopt met die van de file komt er een foutmelding. Wanneer het aantal records Y niet overeenstemt zijn er twee mogelijkheden:

- Er worden minder records opgegeven dan de file bevat. In dat geval zet RDOS de filelengte gewoon op het "oude" aantal en doet verder net alsof U dat zelf ingetypt heeft. LOF is natuurlijk wel netjes bijgewerkt.
- U geeft meer records op dan de file heeft, m.a.w. U wilt de file verlengen. Net zoals bij sequentiële files kan dat alleen als de random file achteraan in de RAM-disc staat. Zo niet, dan breekt het programma af met een fout. Als de file wel achteraan staat maakt RDOS het benodigde aantal records bij en vult ze met spaties. Verder gebeurt er niets bijzonders.

Het verlengen van een random file gaat alleen met een RFILE-opdracht, wanneer U bij een RFILE PUT- of RFILE GET-instructie buiten de grenzen gaat geeft de P2000 een *Subscript out of range*-melding.

Bij het openen krijgt een random file altijd de extensie RND mee. Weer zelf uitkijken of het inderdaad RND is bij het heropenen!

Misschien ten overvloede: een RND-file hoeft nooit afgesloten te worden. Zolang er maar geen andere file met hetzelfde filenummer geopend wordt kunt U altijd meteen lezen en schrijven.

Soms kan het makkelijk zijn een file (SEQ of RND) tijdelijk in een stringarray te stoppen en dat stringarray bijvoorbeeld te sorteren, omdat dat ietsje sneller gaat dan het rechtstreeks sorteren van een RND-file en omdat het bij SEQ-files de enige mogelijkheid tot sorteren is. Er is geen enkel bezwaar tegen het op deze manier gebruiken van stringarray's, want ze hoeven niet weggeschreven te worden! Wanneer U het makkelijk vindt om een file eerst in een array in te lezen, alle bewerkingen op dat array uit te voeren en daarna de file weer record-voor-record te vullen, dan kan dat alleen maar aangemoeid worden. Zo kunt U optimaal gebruik maken van de "random"-mogelijkheden van stringarray's en de zuinigheid en efficientie van de SEQ-files.

#### **Samengevat:**

- Stringarray's zijn bijzonder geschikt voor flexibele opslag van een niet al te groot aantal records (bijvoorbeeld 100).
- Sequentiële files vormen een mooie opslagmogelijkheid voor bijvoorbeeld ledenbestanden. Ze moeten wel telkens in hun geheel worden behandeld.
- Randomfiles kunnen met succes worden toegepast als U zo iets als een telefoonboekje wilt samenstellen: U kunt snel sorteren, uitzoeken of kleine wijzigingen aanbrengen.

Een combinatie van verschillende filetypen kan soms heel handig zijn! Verder is er over het gebruik van files niet al te veel te vertellen, het grootste deel leert U zelf door het gewoon eens te proberen. Wilt U er liever wat over lezen, dan zijn er verschillende prima boeken over het gebruiken van bestanden. Hoewel RDOS een zeer afwijkende syntax heeft en het rechtstreeks overlopen gegarandeerd spaak loopt blijft de algemene lijn natuurlijk hetzelfde.

#### **Compressie van numerieke variabelen.**

Zoals U weet mag een record bij een random file alleen een string zijn. Ook getallen kunnen in een string worden geplaatst met de STR\$(...)-functie. Maar dat is in de meeste gevallen niet erg efficient: er moet altijd ruimte gereserveerd worden voor het "slechtste geval" en dat kan aardig oplopen.

In het geheugen van de P2000 worden getallen daarom op een andere manier bewaard (zie de handleiding op pag. 54 e.v.). Heel in het kort komt het hierop neer: een geheel getal (INT) kost twee bytes, een enkele-precisie getal (SNG) kost vier bytes en een dubbele-precisie getal acht. Dat is dus ongeacht de grootte van het getal. Voor een integer hoeft dan in principe maar voor twee tekens ruimte in de recordstring gereserveerd te worden!

Omdat het verwerken van de losse bytes in een string niet echt makkelijk is heeft RDOS een speciale opdracht om dergelijke getallen om te zetten in een string en omgekeerd: RTURN. De syntax is heel eenvoudig: bijv. RTURN A TO A\$. Als A van het type SNG is, komt er een string uit van vier tekens lengte die rechtstreeks in de recordstring in te voegen is. En later wordt het "echte" getal weer teruggehaald met RTURN A\$ TO A o.i.d. Belangrijk is, dat een zo verkregen string niet gevuld is met ASCII-tekens! PRINT A\$ levert dus verrassingen op. Het is ook niet mogelijk een getal dat oorspronkelijk een integer was later te vertalen naar enkele precisie of zo.

**Foutmeldingen van RDOS 3.1.**

Om de zaak niet nodeloos ingewikkeld te maken gebruikt RDOS bijna alleen maar standaard BASIC-foutmeldingen. Alle meldingen zijn echter geheel met de BASIC verweven zodat ON ERROR GOTO, ERR en ERL en de RESUME-opdrachten normaal werken. Bijvoorbeeld de melding *Niet gevonden* is prima af te vangen. Enige BASIC-fouten hebben overigens een groter "werkgebied" gekregen. Hierna worden alle mogelijke foutmeldingen van RDOS opgesomd en van uitleg voorzien.

**Syntax error** Er klopt iets niet in de spelling van een opdracht. Er is bijv. een komma of een "#" vergeten. Sla er deze handleiding eens op na.

**RETURN without GOSUB** Er moet een RCALL RET uitgevoerd worden, terwijl er geen RCALL SUB aan voorafgegaan is. Deze fout kan ook duiden op een paging tot nesten van RCALL SUB-opdrachten, wat niet toegestaan is.

**Illegal function call** Ergens is een onjuiste parameter opgegeven, bijvoorbeeld een filenummer groter dan drie of een lege string als naam. Deze fout ontstaat ook als U over RAMTOP heen probeert te SAVE-n. Kijk bij halsstarrige gevallen in de handleiding.

**Out of memory** Er is voor de gevraagde file niet genoeg plaats meer, hetzij in de RAM-disc, hetzij in het werkgeheugen. Een file RKILL-en, meer geheugen CLEAR-en of een groter array DIM-men kan de oplossing zijn.

**Subscript out of range** U geeft bij een random file een negatief of te groot recordnummer op. Eventueel kan de file vergroot worden.

**Type mismatch** Het type van de meegegeven variabele(n) is onjuist, bijvoorbeeld string i.p.v. integer.

**Out of string space** Bij het SAVE-n van een stringarray bleek de gereserveerde stringruimte te klein. U dient meer ruimte te CLEAR-en.

**Hissing operand** Er ontbreekt een parameter, bijv. een naam.

**Geen stopje** Vreemd misschien, maar ook de RAM-disc heeft een stopje! Dat zit op adres &HE43C. 1=Wel, 0=geen stopje. Als er geen stopje inzit zijn lees- en wisopdrachten onmogelijk. Sommige toepassingsprogramma's werken hiermee om de RAM-disc te vrijwaren voor veranderingen in de filevolgorde.

**Niet gevonden** Overbekend, de gevraagde filenaam bestaat niet.

**Nee** U heeft de vraag "Hier overheen?" met "Nee" beantwoord.

Dan zijn er ook nog een heel stel nieuwe foutmeldingen:

**File not open (nr. 82)** Er is nog geen file geopend met het gevraagde filenummer, of deze file is zojuist afgesloten (bij het lezen uit een seq. file).

**Bad file mode (nr. 83)** De file is niet op de juiste manier geopend voor de gegeven opdracht, bijv. RFILE PUT A\$ voor een random file, of RFILE PUT als het een file voor lezen is.

**Can't extend (nr. 84)** De te verlengen file staat niet als laatste in de RAM-disc. Zorg ervoor dat hij daar wel komt te staan.

**File still open (nr. 85)** U probeert een file te wissen of te schrijven terwijl er nog een seq. file openstaat om te schrijven.

**Bad record length (nr. 86)** De recordlengte bij een random file-opdracht komt niet overeen met de recordlengte van de file.

**Directory full (nr. 87)** Het maximale aantal files dat in de RAM-disc past is overschreden (dat is 37). Meer kunnen er gewoon niet in. RDUMP de oude RAM-disc of wis een aantal files om plaats te maken.

RDOS 3.1 zit ergens tussen de zgn. run time-lus van BASIC-NL. Daardoor is het mogelijk de extra BASIC-commando's te gebruiken. Met een truukje is de direct mode-lus ook hierlangs geleid, zodat er geen verschil bestaat tussen directe stand of programmastand. Er is maar een geval waar RDOS "zijn" opdracht niet meteen herkend, en dat is direct na een IF...THEN-instructie. Wilt U daar toch een RDOS-commando neerzetten, voeg dan een dubbele punt (:) tussen en het leed is geleden. Dus: IF A\$="Ja" THEN : RCALL SUB "Info" of zoets.

Verder slikt RDOS 3.1 overal constanten, variabelen of expressies, en hoeft U niet op het wel- of niet gebruiken van spatie's te letten. Dit houdt in, dat RDOS 3.1 volledig aansluit op de BASIC-NL, zonder beperkingen in syntax of parametergebruik. Het wijzigen van de "C" van CLOAD of CSAVE in een "R" kan ongestraft, altijd en overal (behalve wanneer er een zgn. "star" stuk machine-taal met het programma is samengevoegd. Dat wordt gelukkig steeds zeldzamer).

#### *De systeemopbouw van RDOS 3.1.*

Het hele programma in machinetaal staat, samen met de inhoudsopgave (1 K) en een bufferblok van 1 K voor o.a. systemfiles, in bank 5. Om nu de gegevens in de andere banken aan te kunnen moeten er af en toe ook kleine stukjes machine-taal in het "normale" geheugen staan. Daarvoor is de kladblokruimte vanaf &H6150 gekozen. De kladblok wordt daarbij echter niet aangetast, zodat hij rustig door Uw eigen programma's gebruikt mag worden. De functietoets van RDOS staat er wel permanent in, maar die is dan ook in de BASIC-listing opgenomen zodat hij in ieder geval te schrappen of te verplaatsen is.

Ook de BASIC-buffer vanaf &H6260 wordt door RDOS regelmatig gebruikt, maar net als de kladblok zet RDOS het eerst even netjes weg voordat er in geschreven wordt. Verder zijn in het geheugen hier en daar wat pointers omgegooid, zodat BASIC de aanwezigheid van RDOS toelaat. U zult hier vrijwel nooit last van hebben.

De zes geheugenbanken vormen een rare manier van geheugenuitbreiding, dat wisten we al. Maar de grootste grap komt nog: toen de P2000 op de tekenplank verscheen dachten de ontwerpers natuurlijk nog helemaal niet aan banken of iets dergelijks. Ja, er was een floppycontroller waar 1 extra bank opzat om het DOS kwijt te kunnen, maar dat was dan ook alles. Dit had tot gevolg dat het machinetaalprogramma dat zorgt voor de eerste opstart van de P2000 na een RESET alleen bank 0 en bank 1 wist! De banken 2 t/m 5 blijven volledig behouden (tenzij U te lang op de knop drukt, want dan "vervagen" ze langzaam). RDOS zit dan ook niet voor niets in bank 5. Wanneer er "per ongeluk" eens een RESET optreedt is het hele systeem dan namelijk nog aanwezig. U hoeft het alleen maar opnieuw aan de BASIC te koppelen. Omdat dat niet zo simpel is kan RDOS het voor U doen.

Meestal is het na een RESET voldoende om OUT &H94,5 in te typen. Met een truukje van M&A Soft werd dat mogelijk. Soms lukt dat echter niet en krijgt U opnieuw een RESET of blijft de P2000 "hangen". Druk in dat geval nogmaals op de knop en typ dan het volgende:

*CLEAR 50,&HFFFF: OUT &H94,5: DEF USR=&HE400: ?USR(0).*

Werkt het dan nog niet, dan is RDOS op de een of andere manier beschadigd. Een RESET komt namelijk niet vanzelf, er moet een oorzaak voor zijn, en een heel goede reden is bijvoorbeeld een machinetaalprogramma dat op hol slaat. Voordat de P2000 het echt niet meer weet en zichzelf RESET kan zo'n "programma" heel wat verknoeien in RDOS. Drukt U daarentegen zelf (kort) op de knop, dan zal er niets ernstigs gebeuren. Blijkt RDOS beschadigd, dan kunt U proberen het RDOS-programma nog een keer te RUN-nen waarbij de RKILL ALL-opdracht uit het programma geschrapt is. U moet wat proberen om te kijken hoe het met de inhoudsopgave staat, bijvoorbeeld RZOEK. Blijkt de boel gesloopt, dan rest slechts RKILL ALL...

Het herstarten van RDOS veroorzaakt normaal een complete re-initialisatie van het systeem en van BASIC, vergelijkbaar met een RESET. De gevolgen van zo'n herstart kunnen beperkt blijven door POKE &H60AC,1. RDOS koppelt zichzelf dan alleen aan en laat de re-initialisatie achterwege. Het herstarten van RDOS heeft overigens geen enkele invloed op de al geladen files.

Om zoveel mogelijk profijt te hebben van de niet-gewiste banken vult RDOS ze "van boven naar beneden". Eerst wordt bank 4 volgeladen, dan bank 3 etcetera. De banken 2 en hoger worden niet gewist bij een RESET, dus U heeft gegarandeerd  $3 * 8 \text{ Kbyte} = 24 \text{ Kbyte}$  aan gegevens terug. Alles wat in de banken 0 en 1 stond is wel definitief verdwenen.

Om U een beetje op weg te helpen geeft RZOEK het aantal vrije bytes aan in twee kleuren. Is het geel, dan zit U nog boven bank 1 en is een RESET niet erg gevaarlijk. Wordt het getal echter magenta, dan zitten er gegevens in de onderste banken en loopt U dus wat risico. Deze kleuromschakeling vindt plaats bij 16384 bytes vrij geheugen. Het kan de moeite lopen op dat moment een oude file te wissen.

Wanneer U liever hebt dat de banken 0 en 1 helemaal niet gewist worden zult U met een EPROM-programmer aan de slag moeten gaan om de MONITOR- of BASIC-ROM om te programmeren. Dat lijkt mij niet zo verstandig, maar het kan wel.

### *RDOS 3.1 vanuit machinetaal.*

De volgende systeemadressen zijn rechtstreeks vanuit machinetaal toegankelijk:

*E400 Herstel pointers van BASIC-NL; systeemininitialisatie.*

*E403 Rload-routine.*

*E406 Rsave-routine.*

*E409 Rkill-routine.*

*E40C Rkill All-routine.*

*E40F Rzoek-routine (directory naar scherm).*

*E412 Zoek-file-routine (NZ=Niet gevonden, anders HL=adres juiste directory).*

*E415 Rdump-routine (alles naar cassette).*

*E418 Rtake-routine (alles van cassette).*

*E41B Bekijk volgend BASIC-statement (HL=adres scheidingsteken).*

Op adres E43C zit het stopje van RDOS, 1=wel, 0=geen.stopje. Alle andere communicatie met het systeem verloopt via de cassetteheaders op &H6030 en &H6130 en het cassetteadres &H60AC. Rload, Rsave en Rkill vragen om een header op &H6030, Zoek-file om een header op &H6130. Bijzonderheden: bij Rload moet het beginadres op &H6030 staan en de maximale lengte op &H6034, bij Rsave het beginadres op &H6030 en de lengte op &H6034. Rkill en Zoekfile hebben alleen een naam nodig. Ook de namen in de headers (2\*8 tekens) mogen afgekort zijn met een sterretje (geen token maar ASCII). Bij de return heeft Z (A=0) de betekenis "succesvol verlopen", NZ=fout, met het foutnummer in A. Na Rload: HL=adres na het laatste geladen byte. Na Rsave: HL=adres directory. Na Rkill: HL=eerste vrije directory-adres.

De directory zit als volgt in elkaar: op &HE000 staat het aantal vrije bytes (LSB en MSB). Vanaf &HE002 begint de directory: steeds 16 bytes naam (evt. met spaties), 3 bytes extensie, 1 byte applicatie, 2 bytes oorsprong, 2 bytes lengte, 1 byte banknummer, 2 bytes adres in de bank. Na de laatste file volgen nullen tot aan &HE400. Het buffergedeelte vanaf &HFC00 is -buiten systemfiles- vrij te gebruiken maar het wordt regelmatig overschreven!

Zet de stack niet in de kladblok of de BASIC-buffer (dat doen de meeste assembler/monitors namelijk!) want deze stukken worden soms even weggezet. De niet vermelde registers kunnen allemaal aangetast zijn.

Ondanks de respectabele hoeveelheid documentatie die U nu doorgenomen heeft kan het best gebeuren dat er iets nog niet geheel duidelijk is. In dat geval kunt U contact opnemen met de auteur, die zijn uiterste best zal doen een en ander uit te leggen. Ook wanneer U een systeemfout ontdekt wordt U vriendelijk verzocht even een briefje te schrijven of op te bellen (liefst tussen 19:00 en 21:00 uur of in het weekend).

Met RDOS 3.1 zijn nog veel meer dingen mogelijk dan uit deze handleiding blijkt, maar iedereen zal zeker zijn eigen toepassingen verzinnen. Ik hoop dat RDOS er voor zal zorgen dat de zes geheugenbanken nu eindelijk eens volledig benut worden en dat het geheel leidt tot wezenlijk nieuwe programmeermogelijkheden en dus tot nieuwe programma's.

Aan mij zal het niet liggen!

Valkenswaard, 22-07-1986

Jeroen Hoppenbrouwers

Wilhelminapark 8

5554 JE VALKENSWAARD

Telefoon: (04902)-13808

Vidibus: 400021237

## Beschikbare systemfiles d.d. 01-12-1986

### Rename

Hiermee wordt de naam van een al in de RAM disc aanwezige file veranderd.  
Syntax: \$"Rename";"Oude naam";"Nieuwe naam"

"Oude naam" mag worden afgekort, "Nieuwe naam" niet. Wordt er bij "Nieuwe naam" geen extensie opgegeven (+CHR\$(0)+"EXT") dan blijft die hetzelfde als bij de oude naam.

Overigens, ook systemfiles mogen van een andere naam voorzien worden, zolang de duidelijkheid niet verloren gaat. Maar laat de extensie altijd SYS blijven!

### Status

Met deze sysfile verkrijgt u een zeer uitgebreide inhoudsopgave. Niet alleen naam, extensie, applicatie en lengte van de opgeslagen file worden getoond, maar ook alle overige bekende systeemgegevens: oorsprong, banknummer en bankadres waar de file nu opgeslagen is.

Syntax: \$"Status"

### Dirconv

Dient om de directory van een RDOS 3.0-dumpfile om te zetten (te converteren) zodat de dumpfile voor RDOS 3.1 leesbaar wordt.

Syntax: \$"Dirconv"

De systemfile spoelt de in de recorder geplaatste cassette terug, zoekt een RDOS 3.0-dumpfile en past die aan. Wordt geen 3.0-file gevonden, dan verschijnt *Niet gevonden*.

Let op: sequentiële files (SEQ) van RDOS 3.0 kunnen niet direct door RDOS 3.1 worden gelezen, ook al is de directory aangepast!

### Take

Om snel een "normaal" cassettebandje in zijn geheel in de RAM disc te zetten.  
Syntax: \$"Take"

Spoelt de cassette terug en laadt dan een voor een alle daarop staande files in de RAM disc. Hierbij wordt alleen het vrije geheugen gebruikt, een eventueel BASIC-programma wordt dus niet aangetast. Het kan zo natuurlijk wel gebeuren dat er niet genoeg ruimte over is om de file te laden, in dat geval komt *Out of memory*. Geef dan CLFAR of zelfs NEW en begin opnieuw. De vorderingen worden op het scherm aangegeven.

### Zet laatst

Moet gebruikt worden om een file als laatste in de RAM disc te zetten. De sysfile verandert dus de filevolgorde. Het in de laatste positie zetten van een SEQ- of RND-file is nodig als deze verlengd moet worden.

Syntax: \$"Zet laatst";"Filenaam"

Bij het verwisselen wordt de vrije ruimte gebruikt, is die te klein dan volgt *Out of memory* en dient u CLEAR of NEW te geven.

### Zoek

Bij een op cassette opgenomen RDOS-dumpfile is het nogal onduidelijk wat er nu precies allemaal instaat. Met deze systemfile kunt u dat te weten komen.  
Syntax: \$"Zoek"

De inhoud van de cassette wordt op de normale manier (zoals bij SHIFT-ZOEK) op het scherm gezet, maar zodra er een RDOS-file wordt gevonden leest de sysfile zijn inhoudsopgave en stuurt die door naar het scherm. Dat gaat trouwens erg snel.

## Vervolg systemfiles

### Maakfile

Deze sysfile moet u gebruiken in samenwerking met de volgende. Hiermee reserveert u namelijk een stuk in de RAM disc als printerfile. Alle tekens die ge-LPRINT worden gaan niet rechtstreeks naar de printer, maar worden in de buffer opgeslagen. Dat gaat natuurlijk veel sneller. De hele printerfile kan achteraf in 1 keer uitgeprint worden met de systemfile "Uitprinten" (hieronder). Maar omdat de printerfile een "gewone" RDOS-file is kunt u dat uitprinten ook uitstellen tot de volgende dag: schrijf de file met RDUMP naar cassette en laadt hem de volgende dag (of bij uw buurman) weer in!

Syntax: \$"Maakfile" : reserveert een file van 1K = 1024 tekens  
      \$"Maakfile": N : reserveert een file van N tekens

Een eventuele oude printerfile wordt -na waarschuwing- overschreven. Tijdens het gebruik van de file wordt het stopje uit de RAM disc gehaald, dat is technisch noodzakelijk. U kunt dan dus niet schrijven of wissen. Raakt de buffer vol, dan verschijnt *Geen printer*. Reserveer dan in het vervolg meer ruimte. De systemfile laat noodgedwongen wat machinetaal achter in de kladblokruimte, na gebruik mag dat weer overschreven worden. Tijdens het gebruik van de file hier echter afbliven!

### Uitprinten

Stuurt de met "Maakfile" aangelegde en door LPRINT- of LLIST-opdrachten (gedeeltelijk) gevulde printerfile naar de printer. Hierbij wordt ter controle ieder teken op het scherm gezet, ook als de printer off line staat (of er helemaal niet is). Na gebruik wordt de file niet gewist, u kunt dus vooraf even kijken wat er in zit en ook meerdere printouts maken. De sysfile herstelt de oude printeradressen weer, koppelt de printerfile af en stopt het stopje weer in de RAM disc. Ook hier wordt weer even van de kladblok gebruikgemaakt.

Syntax: \$"Uitprinten"

Het bestand in de RAM disc bestaat uit ASCII-tekens, beeindigd door &HFF.

### Interruptgestuurde printerbuffer

Het is mogelijk ook een "echte" interrupt driven printer spooler tegelijk met RDOS te laten werken. Hierbij wordt (een gedeelte van) de RAM disc gebruikt als buffer tussen de P2000 en de printer. Alles wat u naar de printer stuurt wordt razendsnel opgeslagen in de buffer en dan op "printersnelheid" aan de printer gevoerd. Het tijdrovende wachten tot de printer klaar is behoort zo tot het verleden. De printer spooler gebruikt de gewone RS-232-uitgang van de P2000, er zijn dus geen speciale voorzieningen aan de computer nodig.

Wanneer u de printer spooler opstart als RDOS al in de computer zit kunt u opgeven hoeveel banken (van 8192 tekens elk) u voor de spooler wilt reserveren. Door 0 te typen wordt de buffer gewist (of niet aangelegd). Het aantal gebruikte banken gaat natuurlijk van de vrije ruimte in de RAM disc af. Dat kan een gevoelige aanslag op de opslagcapaciteit van RDOS betekenen! Verder weet RDOS nu niet beter of het heeft "gewoon" minder banken ter beschikking, dus u ziet ook niet dat er een buffer aanwezig is. RKILL ALL mag zonder meer. RUN-t u de printer spooler terwijl er al een buffer aangelegd is, dan waarschuwt het programma en op commando wordt de oude buffer in zijn geheel gewist zodat RDOS dan weer de volledige geheugenkapaciteit heeft. *Dit is de enige juiste manier om de buffer te verwijderen!*

Gebruikt u de spooler zonder RDOS, dan worden alle banken gereserveerd. U houdt dan een 32K-P2000 over met ruimte in de buffer voor 49152 tekens, die ook weer werkt op de normale RS-232-uitgang.

Zowel met- als zonder RDOS geldt, dat het bufferstuurprogramma bovenin het geheugen komt te staan (vanaf &HDD00). Het werkgeheugen wordt dus iets kleiner dan zonder de spooler (768 bytes).

## Kort overzicht van RDOS-instructies.

### Algemeen:

- <naam> is een string of stringexpressie die een filenaam weergeeft.  
Een evt. niet-standaard extensie kan als volgt opgegeven worden:  
<naam> +CHR\$(0)+"EXT". Dat is normaal volgens BASIC-NL.
- <getal> is een getal of numerieke expressie. Elk type (INT, SNG, DBL) mag gebruikt worden. Evt. getallen in hexadecimale notatie moeten voorafgegaan worden door "&H", bijv. &H5000. Ook dat is standaard BASIC-NL.
- <var> is een numerieke- of stringvariabele, of de "hoofdernaam" van een array.
- <expr> is een numerieke- of stringexpressie.

RLOAD <naam>	: laadt een BASIC-programma in.
RLOAD * <var>	: laadt een array in met naam <var>.
RLOAD * <var> @ <naam>	: laadt een array in met naam <naam>.
RLOAD <naam>, <getal>	: laadt een blok geheugen in vanaf adres <getal>.
RLOAD <naam>, <get.1>, <get.2>	: laadt een blok geheugen in vanaf <getal1>, met als maximaal eindadres <getal2>.
RLOAD <naam>,	: laadt een blok geheugen in vanaf de plaats waar het vandaankwam.
RSAVE <naam>	: schrijft een BASIC-programma weg.
RSAVE * <var>	: schrijft een array weg met naam <var>.
RSAVE * <var> @ <naam>	: schrijft een array weg met naam <naam>.
RSAVE <naam>, <getal>	: schrijft het geheugen weg vanaf adres <getal>.
RSAVE <naam>, <get.1>, <get.2>	: schrijft geheugen weg tussen <get.1> en <get.2>
RCALL <naam>	: laadt een programma in en start dat direct op.
RCALL <naam>, <var>, <var>, ...	: idem met doorgave van variabelen.
RCALL SUB <naam>	: roept een programma op als subroutine.
RCALL SUB <naam>, <var>, ...	: idem met doorgave van variabelen.
RCALL RET	: keert weer terug in het aanroepende programma.
RCALL RET, <var>, <var>, ...	: idem met doorgave van variabelen.
RCALL SYS <naam>	: roept een externe functie op.
RCALL SYS <naam>, <var>, ...	: externe procedure met parameters.
RCALL SYS <naam>; <var>; ...	: externe functie met gegevens.
<i>Opm. "\$" = "RCALL SY"</i>	
RKILL <naam>	: wist de opgegeven file en cruncht de RAM disc.
RKILL ALL	: wist de hele RAM disc.
RZOEK	: zet de inhoud van de RAM disc op het scherm.
RZOEK <var\$>	: -als de naam <var\$> niet in de inhoud voorkomt, komt <var\$> leeg terug (<var\$>="" ). -komt de naam wel voor, dan bevat <var\$> de volledige naam, extensie en lengte.
RZOEK NEXT <var\$>	: geeft de naam, extensie en lengte van de file die volgt op de file met naam <var\$>. Is <var\$> leeg, dan wordt vooraan in de inhoud begonnen. Is het einde bereikt dan wordt <var\$> leeg.
RZOEK FRE <num.var>	: geeft de vrije ruimte in de RAM disc terug in <num.var>.

## Vervolg overzicht RDOS-instructies

RFILE <naam> : opent of heropent sequentiële file <naam> met als filenummer nul.

RFILE #<getal>,<naam> : idem met filenummer <getal>.

RFILE #<get.1>,<naam1>,#<get.2>,<naam2>,... : idem voor meerdere files.

RFILE <naam> FOR APPEND : opent file nul om te verlengen.

RFILE #<getal>,<naam> FOR APPEND : idem voor file <getal>.

RFILE #<get.1>,<naam1>,#<get.2>,<naam2> FOR APPEND : combinatie.

RFILE PUT <expr>,<expr>,... : schrijft <expr> naar sequentiële file nul.

RFILE PUT #<getal>,<expr>,... : idem naar file <getal>.

RFILE GET <var>,<var>,... : haalt het volgende record van seq. file 0 op.

RFILE GET #<getal>,<var>,... : idem uit seq. file <getal>.

RFILE RES : reset file 0: schrijven wordt lezen of lezen begint weer vooraan.

RFILE RES #<get.1>,#<get.2>,... : idem voor file <get.1>, <get.2>, ...

RFILE <get.1>,<get.2>,<naam> : opent of heropent randomfile <naam> met als filenummer nul. <get.1> = aantal records, <get.2> = recordlengte. Aantal records mag evt. veranderd worden, recordlengte niet. Bij <get.2> = 0 zet RDOS zelf de recordlengte goed.

RFILE #<get.1>,<get.2>,<get.3>,<naam> : idem met filenummer <getal1>.

RFILE #<get.1>,<get.2>,<get.3>,<naam>,#<get.4>,<get.5>,<get.6>,<naam>,... : combinatie.

RFILE PUT <getal>,<expr\$> : schrijft <expr\$> op record <getal> in file nul.

RFILE PUT #<get.1>,<get.2>,<expr\$> : idem voor file <getal1>.

RFILE PUT <get.1>,<expr\$>,<get.2>,<expr\$> : combinatie.

RFILE GET <getal>,<var\$> : leest record <getal> uit file nul in <var\$>.

RFILE GET #<get.1>,<get.2>,<var\$> : idem uit file <getal1>.

RFILE GET <get.1>,<var\$>,<get.2>,<var\$> : combinatie.

EOF : SNG-variabele, "waar" (=-1) als het einde van de laatst gebruikte random- of sequentiële file bereikt is, anders "niet waar" (=0).

LOF : SNG-variabele, geeft het hoogste recordnummer van de laatst gebruikte random file.

RDUMP : schrijft de hele RAM disc naar cassette.

RDUMP <naam> : schrijft de opgegeven file naar cassette.

RTAKE : leest de hele RAM disc in van cassette.

RTAKE <naam> : leest de opgegeven file in van cassette.

## De systeemcassette

Op de meegeleverde cassette staan verschillende programma's en hulpbestanden die u naar believen kunt gebruiken. Voor uw gemak is het een en ander zo in elkaar gezet dat u kunt volstaan met het in de recorder plaatsen van de cassette en een druk op de RESET-toets naast de recorder.

Eerst wordt dan het hoofdprogramma RDOS van de cassette gehaald en opgestart. U ziet dat door de mededeling *RAM Disc Operating System versie 3.1* die bovenaan het scherm verschijnt. Vanaf dat moment is het systeemprogramma RDOS in uw P2000 aanwezig en kunt u de zes geheugenbanken van de 64K-print gebruiken als RAM disc. Wat een RAM disc precies is, vindt u in de handleiding (pag. 1).

In het programma RDOS zijn vanaf regel 100 een aantal BASIC-opdrachten opgenomen die ervoor zorgen dat de andere hulpprogramma's op de cassette na het opstarten van RDOS automatisch worden geladen. Voor een eerste kennismaking met RDOS kunt u het beste de computer "zijn eigen gang laten gaan" en rustig wachten totdat het index-programma verschijnt. Wilt u echter het programma RDOS "los" gaan gebruiken (dus zonder alle demoprogramma's etc.), dan moet u de extra opdrachten vanaf regel 100 even verwijderen. Laat echter RKILL ALL staan, dit is een broodnodige instructie als de P2000 uitgeschakeld is geweest (handleiding pagina 3).

Na ongeveer een minuut heeft de P2000 de volledige cassettekant gelezen en verschijnt het index-programma (overigens geheel geschreven in BASIC, met commentaar). Met de pijltoetsen en ENTER kunt u het gewenste programma kiezen. Hier loopt u al meteen tegen een bijzonderheid van RDOS aan: lang niet alle namen in de index horen bij een BASIC-programma! Alleen de namen waar "BAS" achter staat zijn BASIC-programma's, de rest hoort ergens bij als gegevensbestand of is een zgn. *systemfile* (zie verder). Wanneer u toch een naam uitkiest waar geen "BAS" achterstaat, weigert het indexprogramma de opdracht. U wordt aangeraden als eerste "Demo van RDOS" aan te roepen. "Demo deel twee" volgt daar vanzelf op. "File demo" is niet zozeer een gebruikersprogramma, maar laat zien hoe er onder RDOS met "random files" gewerkt kan worden. De LIST-ing is hier belangrijker dan de werking.

De vele "SYS"-bestanden vertegenwoordigen evenveel extra RDOS-instructies. In de P2000 is namelijk niet genoeg ruimte om alle mogelijke opdrachten die RDOS kent een plaatsje te geven. Noodgedwongen moeten sommige commando's daarom in de RAM disc blijven staan. Wanneer ze nodig zijn zorgt RDOS er zelf voor dat de losse instructie tijdelijk ingekoppeld wordt in RDOS. U hoeft er alleen maar voor te zorgen dat RDOS het commando ook kan vinden, dus dat het inderdaad in de RAM disc staat! Wanneer u een "losse" opdracht wilt gebruiken dus eerst even kijken of hij er wel is, zo niet, dan bijladen. Instructies die u niet (meer) nodig heeft hoeven natuurlijk niet in de RAM disc te blijven staan: die kunnen gewist worden (maar gooï ze niet weg, je weet nooit). Wilt u daarentegen een systemfile in zijn eentje op een cassette zetten (dus zonder bijv. de demoprogramma's) dan kan dat ook d.m.v. een RDUMP-opdracht. Hoe dat precies moet vindt u in de handleiding op pag. 9.

Op de volgende pagina's volgt een korte samenvatting van de zgn. *systemfiles*. Nogmaals, het zijn dus "gewone" RDOS'opdrachten, alleen staan ze niet permanent in het geheugen van de computer. Voor meer informatie: zie de handleiding, pagina 7 e.v.

## Trefwoorden index

Trefwoord	Pagina
Adressen	17
Afkortingen van namen	5
Applicatietaken	5
Array's lezen en schrijven	4, 14
Bank	1, 16, 17
Buffer	Bijlage
Cassettewriter	8
Dirconv	Bijlage
EOF	11, 13
Entry points	17
Extensie	5
File	3, 5, 10
Filenummer	12
Foutmeldingen	15
Geheugenbank	1, 6, 17
Heropenen random file	13
Herstellen van RDOS	16
Hier overheen ?	5, 8
Kleuren bij RZOEK	17
LOF	13
Makfile	Bijlage
Machinetaal	17
Mag de rest weg ?	8
Meer files tegelijk	12
Printer spooler	Bijlage
RAM disc	1
RCALL BASIC-programma	6
RCALL RET	6
RCALL SUB	6
RCALL SYS	7
RDOS vanuit machinetaal	17
RDUMP	8
RDUMP "Naam"	9
RFILE	10
RFILE FOR APPEND	12
RFILE GET	11, 13
RFILE PUT	10, 13
RFILE RES	11
RKILL	3
RKILL ALL	3
RLOAD	2
RLOAD geheugen	4
RLOAD *	4
RSAVE	3
RSAVE geheugen	4
RSAVE *	4
RTAKE	8
RTAKE "Naam"	9
RTURN	14
RZOEK	3
RZOEK FRE	9
RZOEK NEXT	10
RZOEK string	9
Random files	12
Recorder	8
Rename	Bijlage
Reset van de computer	16
SYS	7
Sequentiële files	10
Spooler	Bijlage
Status	Bijlage
Stringarray's	4, 14
Systeemopbouw	16
Systemfile	7
Take	Bijlage
Uitprinten	Bijlage
Variabelen-omzetting	14
Zelf maken van systemfiles	8
Zet laatst	Bijlage
Zoek	Bijlage