INHOUD

INLEIDING

Deze manual beschrijft DOS65 Basic. Momenteel zijn er twee versies van DOS65 leverbaar: versie 1.01 en versie 2.01. Voor beide versies van DOS is een Basic verkrijgbaar, respektievelijk versie 1.00 en versie 2.00. Op een paar kleine details na zijn deze Basic's identiek. Programma uitwisseling tussen de beide Basic's is mogelijk, mits de programma's in ASCII vorm op schijf worden opgeslagen en u het volgende in het hoofd houdt:

- De workspace van Basic 2.00 is ca. 8 Kbytes groter. Zeer grote programma's die wel onder Basic 2.00 draaien doen dat mogelijk niet onder versie 1.00
- Basic 2.00 kan met maximaal zes geopende data-files samenwerken, Basic 1.00 met maximaal vier.

Er zijn nog een aantal kleine verschillen, en waar nodig zal hier aandacht aan besteed worden in deze manual.

Wij gaan ervan uit dat er in de toekomst nieuwe (betere) releases van DOS65 Basic uitgebracht zullen worden. Het ligt echter voor de hand dat alleen versie 2.00 verder ontwikkeld zal worden. Een reden te meer om uw DOS65 systeem aan te passen aan de geheugenindeling van DOS versie 2.01.

Wageningen, 6-aug-1986

Gert Klein

HOOFDSTUK 1

OVEREENKOMSTEN EN VERSCHILLEN MET KB9 BASIC

1.1 OVEREENKOMSTEN MET KB9 BASIC

DOS65 Basic is ontwikkeld uit de in onze club alom bekende KB9 Basic van Microsoft. De volgende sleutelwoorden, functies en operators zijn in werking gelijk gebleven aan KB9 Basic. Voor een beschrijving hiervan zij verwezen naar de officiele documentatie van Microsoft, of een van de vele publicaties over de taal Basic.

Sleutelwoorden

END	FOR	NEXT	DATA	DIM	READ
LET	GOTO	IF	RESTORE	GOSUB	RETURN
REM	STOP	ON	DEF	POKE	PRINT
CONT	LIST	CLEAR	NEW	TO	THEN
STEP	FN				

Functies

SGN	INT	ABS	FRE	POS	SQR
RND	LOG	EXP	COS	SIN	TAN
ATN	PEEK	LEN	STR$	ASC	CHR$
LEFT$	RIGHT$	MID$	TAB(	SPC(	

Operators

+	-	*	/	^	AND
OR	NOT	<	=	>	

1.2 VERSCHILLEN MET KB9 BASIC

1.2.1 Geheugenindeling DOS65 Basic

Basic 1.00 neemt het geheugengebied van adres $761E tot en met adres $9FFF in beslag. De workspace begint op adres $2000. Basic 2.00 neemt het geheugengebied van adres $951A tot en met $9FFF in beslag. De workspace begint op adres $0400. De zero page wordt door Basic 1.00 van adres $00 tot en met adres $9B gebruikt. Bij Basic 2.00 is dit adres $00 tot en met adres $9D Zie de appendix 1 voor een overzicht van de zeropage onder Basic. Het gebied van adres $9C respektievelijk $9E tot en met $FF is vrij voor de gebruiker. Basic 1.00 maakt gebruik van de adressen $0200 tot en met $02D1 , Basic 2.00 maakt gebruik van de adressen $0200 tot en met $0312

1.2.2 Algemene wijzigingen.

- DOS65 Basic accepteert input in lower case. Sleutelwoorden, functies en namen van variabelen worden door Basic intern tot uppercase gekonverteerd. Lowercase karakters in stringvariabelen blijven echter lower case.

- De stop toets in DOS65 Basic is de control-C toets. Basic kontroleert elke keer wanneer een nieuw statement wordt gexecuteerd of de ctrl-C toets ingedrukt is geweest. Is dat het geval dan keert Basic met een Break in ... boodschap terug op commando niveau. Met CONT kan de executie van het programma hervat worden.

- Er zijn een aantal wijzigingen in de invoer routines. Bij KB9 Basic leidde het intoetsen van een '@' tot het cancelen van het ingetypte commando of de invoer string bij een INPUT statement. In DOS65 Basic is dit 'feature' verwijderd.

- Bij KB9 Basic was de underline ($5F) het delete character. Bij DOS65 Basic is dat de rub-out.

- Werd in KB9 Basic een ascii shift in ($0F) karakter geprint, dan leidde dit tot een onderdrukking van alle output. Dit is in DOS65 Basic verwijderd.

- Control characters worden bij invoer geweigerd, met uitzondering van het BELL ($07) character.

- De inputbuffer is van de zero page naar page 2 verplaatst. De inputbuffer is 80 karakters lang en loopt van adres $0200 tot en met adres $024F. Door het verplaatsen van de inputbuffer is er veel meer ruimte gekomen in de zero page. De zero page adressen die boven de inputbuffer lagen, zijn naar onder verplaatst.

- De USR functie is verwijderd. In plaats daarvan kent DOS65 Basic het CALL sleutelwoord.

- In de allereerste versies van de 6502 processor was de ROR instruktie nog niet aanwezig. Deze instruktie is echter wel nodig bij het werken met floating point getallen. De KB9 Basic emuleert deze instructie met reeksen van shift en or instrukties. Uiteraard gaat dit ten koste van de rekensnelheid. Is DOS65 wordt de ROR instructie wel in de floating point routines gebruikt, met als resultaat een hogere rekensnelheid.

HOOFDSTUK 2

NIEUWE COMMANDO'S IN DOS65 BASIC

2.1 OPSTARTEN VAN BASIC

Alvorens we DOS65 Basic opstarten kopieren we hem op de systeemdisk. Er zijn twee manieren om Basic op te starten:

- Door vanuit DOS65 het commando BASIC gevolgd door een return in te tikken. Basic meldt zich dan met het aantal vrije bytes in de workspace en het nummer van de betreffende release.

- Door vanuit DOS65 het commando BASIC "filenaam" in te tikken, waarbij de file een Basic programma bevat. Basic wordt dan in het geheugen geladen, en direkt daarna wordt het Basic programma opgestart. In dat geval verschijnt de opstartboodschap niet op het beeldscherm. Wordt het programma beeindigd, bijvoorbeeld na een END statement of als er een fout optreedt, dan keren we terug op DOS niveau. Op deze wijze kunt u Basic programma's opstarten vanuit een DOS65 commando file.

2.2 HET WERKEN MET HEXADECIMALE GETALLEN

DOS65 Basic ondersteunt het werken met hexadecimale getallen. Een hexadecimaal getal moet altijd vooraf worden gegaan door een '&'. Verder moet het hexadecimale getal uit vier digits bestaan. 01FA is dus een legaal getal, 1FA niet:

```
10 A = &E000
20 PRINT A
30 END
RUN
 57344
```

De VAL functie is aangepast om strings die een hexadecimaal getal representeren korrekt om te zetten. De string moet daarbij ook met een '&' beginnen:

```
10 A$ = "&E000"
20 C = VAL(A$)
30 PRINT C
40 END
RUN
 57344
```

Een nieuwe functie HEX$ zorgt voor konversie van decimale getallen naar een string in hex formaat. De string heeft altijd een lengte van vier karakters. HEX$ gooit bij een gebroken getal de cijfers achter de decimale punt weg (geen afronding dus maar een 'truncate').

```
10 A = 65000
15 B = 65000.9
20 C$ = HEX$(A)
25 D$ = HEX$(B)
30 PRINT C$, D$
40 END
RUN
FDE8        FDE8
```

2.3 CURSORSTURING EN VIDEO SLEUTELWOORDEN

De volgende sleutelwoorden zijn aan DOS65 Basic toegevoegd om een optimaal gebruik te kunnen maken van de video faciliteiten van het systeem.

HOME - Wist het beeldscherm en zet de cursor links boven in het beeldscherm

INVERSE - Schakelt over op inverse video. Alle output naar het beeldscherm wordt invers afgedrukt.

NORMAL - Schakelt terug van inverse video naar normale video.

Voor cursorsturing is het sleutelwoord CURSOR toegevoegd. De syntax is:

```
CURSOR TO row,column
```

Het regelnummer moet liggen tussen 1 en 24, het kolomnummer tussen 1 en 80. Andere waarden leveren een 'Illegal Quantity' error op.

Een voorbeeld met bovengenoemde sleutelwoorden:

```
10 HOME : REM Wis het beeldscherm
20 CURSOR TO 10,30 : REM Zet cursor op regel 10, kolom 30
30 INVERSE : REM Schakel over op inverse video
40 PRINT "Dit is inverse video"
50 NORMAL : REM Schakel terug naar normale video
60 CURSOR TO 11,30 : REM Zet cursor op regel 11, kolom 30
70 PRINT "Dit is normale video"
80 END
```

2.4 EXIT EN MONITOR SLEUTELWOORDEN

Met het commando EXIT keren we vanuit Basic terug naar DOS niveau. Met het DOS commando

```
$GO 0
```

keren we terug in Basic (mits u geen puinhoop van de zero page heeft gemaakt).

In Basic 1.00 bestaat het sleutelwoord MONITOR. Met het commando MONITOR wordt naar de MON65 monitor gesprongen. De monitor wordt als subroutine aangeroepen, derhalve zorgt het monitor commando Q voor een terugkeer naar Basic. MONITOR kan in een programma gebruikt worden, na Q wordt dus de executie van het programma hervat. Met GO 0 vanuit DOS is dit niet mogelijk. We komen dan altijd in Basic commando mode terecht. Het Monitor sleutelwoord bestaat niet in versie 2.00 aangezien er geen monitor in ROM aanwezig is.

2.5 HET CALL SLEUTELWOORD

Met het CALL sleutelwoord kan een machinetaal subroutine worden aangeroepen. De syntax is:

CALL <adres>

waarbij het adres moet liggen tussen 0 en 65535 oftewel $0000 en $FFFF.

Een voorbeeld (alleen voor versie 1.00):

```
10 A=&E033 : REM Monitor als subroutine
20 CALL A
30 ? "Terug in Basic"
40 END
RUN

MON65 1.01
HaViSoft

Q<return>

Terug in Basic

Ok
```

2.6 DEBUGGEN MET TRACE

Het TRACE sleutelwoord is bedoeld als hulp bij het debuggen van programma's. De syntax is:

```
TRACE [ON]
      [OFF]
```

Na TRACE ON zal Basic de regelnummers van alle geexecuteerde statements uitprinten. De regelnummers staan tussen vierkante haakjes. Met TRACE OFF wordt het trace mechanisme uitgezet. Met behulp van TRACE kunt u de executie van een programma dus gemakkelijk volgen.

```
10 FOR X=1 TO 3
20 LET A=A+X
30 NEXT X
40 END
TRACE ON
RUN

[ 10][ 20][ 30][ 20][ 30][ 20][ 30][ 40]
```

2.7 HET GET SLEUTELWOORD

Het GET sleutelwoord is in DOS65 Basic als volgt geimplementeerd. Na een GET statement kijkt Basic of er een toets werd ingedrukt. Is dat niet het geval dan wordt een lege string teruggegeven. werd er wel een toets ingedrukt dan krijgt de stringvariabale de waarde van de toets toegekend. Control characters worden ook aan de variabele toegekend, dit in tegenstelling tot een INPUT statement dat control characters negeert.

```
10 GET A$
20 IF A$ = "" THEN GOTO 10
30 IF A$ = CHR$(13) THEN PRINT "Dit is een <return>"
40 IF A$ = "A" THEN PRINT "Dit is een 'A'"
.
.
```

2.8 WIJZIGINGEN IN HET INPUT STATEMENT

In DOS65 Basic werkt het INPUT statement iets anders dan in KB9 Basic. Wordt bij KB9 Basic alleen een return gegeven na een INPUT, dan wordt de executie van het programma afgebroken en keert Basic op commando niveau terug. Bij DOS65 Basic is dat niet het geval. Wordt bij een INPUT alleen een return gegeven dan keert INPUT met een lege string terug of het getal 0 in het geval van een numerieke variabele.

In DOS65 Basic bestaat de mogelijkheid om het vraagteken dat afgedrukt wordt na een INPUT te onderdrukken. Dit doen we door in een statement direkt na het INPUT sleutelwoord een punt-komma op te nemen:

```
10 INPUT "Input met vraagteken" ; A$
20 IF A$ = "" THEN PRINT "Dit was een lege string"
30 IF A$ <> "" THE PRINT A$
40 REM
50 INPUT; "Input zonder vraagteken" ; A$
60 IF A$ = "" THEN PRINT " Dit was een lege string"
70 IF A$ <> "" THEN PRINT A$
80 INPUT; "Geef getal :" ; B
90 IF B = 0 THEN PRINT "Zeker alleen een return gegeven"
100 IF B<> 0 THEN PRINT B
.
.
```

HOOFDSTUK 3

INPUT EN OUTPUT MET DOS65 BASIC

3.1 ALGEMEEN

Een van de belangrijkste factoren die de bruikbaarheid van een computer systeem bepalen is de flexibiliteit waarmee I/O gedaan kan worden. Wat dat betreft scoort het DOS65 systeem hoog, want vooral op het gebied van I/O zijn er uitgebreide mogelijkheden. Bij het schrijven van DOS65 Basic is er vanuit gegaan dat Basic zo goed mogelijk aan de I/O faciliteiten moest worden aangepast.

In DOS65 Basic bestaan de volgende I/O mogelijkheden:

- Mogelijkheden om alle acht monitor I/O devices aan te sturen.
- Opslaan van Basic programma's op disk, zowel in binaire als in ASCII vorm.
- Mogelijkheden om DOS commando's via Basic uit te voeren.
- Sequential acces data files.
- Random acces data files.
- I/O redirect.

In de nu volgende paragrafen zullen we hier uitgebreid op in gaan.

3.2 MONITOR I/O DEVICES

Zoals bekend zal zijn kunnen vanuit de monitor 8 input en 8 output devices aangestuurd worden. Ook via Basic is dit mogelijk. In principe kan dit op twee manieren:

- Via een control^I of een control^O commando, waarmee u in de statusregel het gewenste device kunt aan- of uitzetten.

- Vanuit een programma met het DEVICE sleutelwoord.

De syntax van het DEVICE statement is:

```
DEVICE n [INPUT]  [ON]
         [OUTPUT] [OFF]
```

Daarbij is n het devicenummer. Het devicenummer moet liggen tussen 1 en 8. Andere waarden leveren een 'Illegal quantity' error op. INPUT geeft aan dat een input device gestuurd moet worden, OUTPUT een output device. ON geeft aan dat het betreffende device ingeschakeld moet worden, OFF dat het device uitgeschakeld moet worden. Voor een uitvoerige beschrijving van alle I/O devices, zie de MON65/IO65 manual.

Hieronder een korte opsomming van alle beschikbare devices:

Input	Output
1 Toetsenbord	1 Beeldscherm
2 Niet beschikbaar	2 Printer
3 RS232 interface	3 RS232 interface
4 VIACOM input	4 VIACOM output
5 Memory input	5 Memory output
6 vrij	6 vrij
7 vrij	7 vrij
8 vrij	8 vrij

Nu een voorbeeld van een programma met uitvoer naar de printer:

```
10 REM Zet de printer aan
20 DEVICE 2 OUTPUT ON
30 REM Zet het beeldscherm uit
40 DEVICE 1 OUTPUT OFF
50 PRINT "Deze regel wordt op de printer geprint"
60 REM Zet de printer uit
70 DEVICE 2 OUTPUT OFF
80 REM Niet nodig device 1 weer aan te zetten, dit is
90 REM de default waarde als alle devices uit worden
100 REM gezet.
110 END
```

Een opmerking over het aanzetten van device 3, de RS232 interface. Elke keer dat een device wordt aangezet, wordt het geinitialiseerd. In het geval van Basic 1.00 betekent dit dat de ACIA wordt geladen met de bytes in de adressen $CE50 (ACIA control register) en $CE51 (ACIA command register). Voor versie 2.00 is dit respektievelijk adres $E734 en $E735. Wanneer u de RS232 interface wilt gebruiken met een afwijkende baudrate of iets dergelijks dan dient u de juiste waarden vooraf in deze adressen te poken.

De initialisatie van de Memory I/O is anders bij de verschillende Basic versies. Bij versie 1.00 kunt u het begin en eindadres in de statusregel intikken. Bij Basic versie 2.00 dient u deze adressen in het geheugen te poken:

adres			functie
WRBEG	-	$E776	Write begin adress
WREND	-	$E778	Write end adress
REBEG	-	$E77A	Read begin adress
REEND	-	$E77C	Read end adress

Een klein verschil merkt u ook wanneer het intialiseren van de printer mislukt. Bij Basic versie 1.00 ziet de melding 'Device not ready' in de statusregel verschijnen. Bij Basic versie 2.00 volgt de foutmelding 'Device initialization error'.

3.3 HET SAVE SLEUTELWOORD.

Met het SAVE sleutelwoord kunnen we een Basic programma in een file wegeschrijven. De syntax is:

```
SAVE "filenaam" [,A]
```

De filenaam krijgt automatisch de extensie .BAS toegekend. Het is niet nodig deze filenaam extensie op te geven. We kunnen een programma op twee manieren wegschrijven. Met het commando:

```
SAVE "filenaam"
```

wordt het programma in 'getokenisede' vorm weggeschreven. Feitelijk betekent dat het stuk geheugen waarin het programma staat in de file gered wordt.

Met het commando:

```
SAVE "filenaam",A
```

wordt het programma in de vorm van een ascii file weggeschreven. Een dergelijke file kunt u met de editor wijzigen, of u kunt hem TYPEn of PRINTen. Het saven van een programma is heel handig als u een programma met de editor hebt aangemaakt en er een fout in ontdekt als u het geladen hebt. U kunt het programma nadat u de fout hersteld hebt als ascii file saven, zonder dat u eerst Basic moet verlaten en de editor opstarten om de fout te corrigeren. Een belangrijk voordeel van het Saven in ASCII vorm is tevens dat u programma's kunt uitwisselen met iemand die een andere versie van Basic heeft dan u.

3.4 HET LOAD SLEUTELWOORD

Met het LOAD sleutelwoord kunnen een we een Basic programma in het geheugen laden. De syntax is:

```
LOAD "filenaam"
```

Het maakt niet uit of het programma in getokenisede vorm, dan wel in ascii vorm in de file staat. Basic kijkt zelf hoe het programma in de file staat en zorgt er voor dat het correct in het geheugen komt te staan. Het laden van een ascii file duurt overigens wel stukken langer dan het laden van een getokenised programma. Het load commando mag niet in een programma gebruikt worden. Doet u dit toch dan volgt er een 'Syntax' error.

3.5 HET MERGE SLEUTELWOORD

Met het MERGE commando kunnen we een Basic programma dat in ascii vorm in een file staat toevoegen aan een programma dat op dat moment al in het geheugen staat. De syntax is:

```
MERGE "filenaam"
```

Zoals gezegd moet de te mergen file een ascii file zijn, een ander filetype levert een 'Incorrect format' error op. Het merge commando moet u wel met enige voorzichtigheid gebruiken. Staan er namelijk regels in de ascii file die hetzelfde regelnummer hebben als statements die in het geheugen staan, dan worden deze overschreven. Het merge statement mag niet in een programma gebruikt worden, dat levert een 'Syntax' error op.

3.6 HET RUN SLEUTELWOORD

Het sleutelwoord RUN is aangepast aan DOS65. Het kent nu drie vormen:

```
RUN

RUN regelnummer

RUN "filenaam"
```

- RUN zorgt voor het opstarten van het programma dat in het geheugen staat. Er wordt gestart op het laagste regelnummer.
- RUN regelnummer zorgt ervoor dat het programma gestart wordt op het opgegeven regelnummer.
- RUN "filenaam" laadt het programma van disk in het geheugen en begint executie op het laagste regelnummer. Net als bij een LOAD mag het programma zowel als ascii file als in getokenisede vorm in de file staan.

```
.
.
100 INPUT "Uw keuze"; A$
110 IF A$="1" THEN FI$="FILE1"
130 IF A$="2" THEN FI$="FILE2"
140 IF A$="3" THEN FI$="FILE3"
150 REM Start nu het programma op
160 RUN FI$
```

Nog een opmerking over het opstarten van een programma via het commando BASIC "filenaam" vanuit DOS65. Ook hierbij maakt het geen verschil uit of het programma in ascii vorm dan wel in getokenisede vorm in de file staat.

3.7 HET DOS SLEUTELWOORD

Met het sleutelwoord DOS kunnen we een DOS commando executeren. De syntax is:

DOS "doscommando"

Opgelet! Niet alle DOS commando's kunnen aangeroepen worden, met name commando's op schijf kunnen de Basic workspace overschrijven. Een aantal commando's die wel gebruikt mogen worden:

- Alle 'ingebouwde' commando's, zoals DIR, SEE, ASN, enz.
- De commando's CAT, DELETE, TYPE, PRINT, TYPE, CREATE, DUMP, LIST, RENAME, SETBEGIN, SETDRIVES, SETMODE, TIME, UNEXPAND, ECHO

NIET gebruikt mogen worden: APPEND, COPY, FORMAT, MEMFILL, MEMMOVE, PLIST, SDIR, MONITOR.

In feite kunt u elke utility gebruiken die op adres $0800 (DOS 1.01) of adres $A000 (DOS 2.01) wordt geladen. U kunt dit zelf nagaan met het MAP commando.

3.8 SEQUENTIAL DATA FILES IN DOS65 BASIC

In de voorgaande paragrafen is al gesproken over de samenwerking tussen DOS65 en Basic met betrekking tot de opslag van programma files. We willen echter ook variabelen kunnen opslaan in files om later weer te kunnen gebruiken. Een van de filetypen waarin we variabelen kunnen opslaan is die van de sequentiele file. Een sequentiele file is een file waar gegevens in een opeenvolgende reeks worden opgeslagen. De lengte en het type van de variabelen in de file zijn van te voren niet bekend. Omdat alle variabelen in de vorm van ascii characters worden opgeslagen kunnen numerieke en stringvariabelen door elkaar worden weggeschreven. Een sequentiele file kan niet tegelijkertijd worden gelezen en beschreven. Een file is ofwel geopend om te worden gelezen ofwel om te worden

beschreven. Verder is het niet mogelijk om na het openen van een file een willekeurige variabele in te lezen. We moeten de variabelen een voor een in lezen, te beginnen met de eerste. Andersom, als we de waarde van een variabele willen wijzigen of een variabele willen toevoegen, dan moeten we de file eerst openen om te lezen, vervolgens alle variabelen inlezen, de file sluiten, opnieuw openen om te beschrijven en dan alle variabelen al dan niet gewijzigd weer wegschrijven.

Voor het openen en sluiten van een file dienen de sleutelwoorden OPEN en CLOSE:

```
OPEN "filenaam", [I]
                 [O]

CLOSE "filenaam"
```

Om een file te openen om te beschrijven gebruiken we bv het statement:

```
.
.
100 OPEN "TEST", O
.
```

Er wordt nu een file aangemaakt door DOS65 met de file TEST.VAR. De filenaamextensie .VAR wordt automatisch toegevoegd. Als we een file openen om te beschrijven dan kijkt DOS65 eerst of een file met dezelfde naam al aanwezig is. Is dat inderdaad het geval, dan wordt deze file eerst gewist alvorens de nieuwe file wordt aangemaakt. Het zal u duidelijk zijn waarom het nodig is eerst alle variabelen uit deze file te lezen voordat u de file kunt wijzigen. De file TEST.VAR openen we om te lezen met:

```
.
.
100 OPEN "TEST", I
.
```

Nadat de file geopend is kunnen we de variabelen een voor een inlezen. Denk eraan dat het niet mogelijk is terug te lezen in de file. Hebt u bijvoorbeeld de 10e variabele ingelezen uit de file, en wilt u bijvoorbeeld de 1e nog eens inlezen, dan moet u eerst de file sluiten, vervolgens weer openen en de variabele inlezen.

Een file die geopend is moet ook weer gesloten worden. Als de file geopend was om te beschrijven, dan zorgt het sluiten ervoor dat data die nog intern in buffers staat ook daadwerkelijk worden weggeschreven. Om de file TEST.VAR te sluiten gebruiken we het statement:

```
.
.
100 CLOSE "TEST"
.
```

Voor het lezen uit en schrijven in een sequentiele file gebruiken we de sleutelwoorden FREAD (file read) en FWRITE (file write). De syntax is:

```
FREAD "filenaam", variabele(n)

FWRITE "filenaam", variabele(n)
```

De werking van FREAD en FWRITE is in feite hetzelfde als van INPUT en PRINT, alleen komt de input en gaat de output naar een file. Elke variabele die wordt weggeschreven moet afgesloten worden met een return, om later bij het inlezen herkend te worden (bij een INPUT sluiten we het ingetypte immers ook af met een return). Een ouput file heeft hetzelfde formaat als een normale DOS65 text file. We kunnen een dergelijke file dus TYPEn en PRINTen of desnoods met de editor wijzigen. Willen we in een FREAD statement meerdere variabelen in dezelfde regel inlezen, dan moeten deze door een komma gescheiden zijn. Vergelijk dit met INPUT waar twee variabelen ook door een komma gescheiden moeten worden als we ze met hetzelfde INPUT statement inlezen.

Een voorbeeld om het bovenstaande te illustreren:

```
10 REM Open file TEST.VAR om te beschrijven
20 OPEN "TEST", O
30 FOR X=1 TO 10
40 REM Schrijf de variabele X weg
50 FWRITE "TEST", X
60 NEXT X
70 REM Sluit de file
80 CLOSE "TEST"
90 REM Nu gaan we de file weer inlezen
100 DIM A(10)
110 OPEN "TEST", I
120 FOR X=1 TO 10
130 REM Lees een variabele uit de file
140 FREAD "TEST", A(X)
150 NEXT X
160 REM Sluit nu de file
170 CLOSE "TEST"
180 REM Print het array
190 FOR X=1 TO 10
200 PRINT A(X);
210 NEXT X
220 END
```

Toelichting:

- In regel 20 openen we de file TEST.VAR om te beschrijven. Een oude file TEST.VAR wordt eerst gewist.

- In regel 50 schrijven we de waarde van X weg. Omdat het statement niet afgesloten wordt met een ';' wordt er een afsluitende return weggeschreven. Vergelijk dit met een PRINT statement!

- In regel 80 sluiten we de file. De inhoud van interne buffers wordt nu naar floppy geschreven.

- In regel 110 openen we de file om er uit te lezen.

- In regel 140 lezen we de variabelen in. Omdat FREAD zich net zo gedraagt als een INPUT gedraagt, zorgt elke return in de input file ervoor dat de variabele wordt ingelezen.

- In regel 170 tenslotte sluiten we de file weer. Openen we de file daarna weer om te lezen, dan beginnen we weer met de eerste variabele in de file.

Het zal u duidelijk zijn dat u als programmeur zult moeten weten hoeveel variabelen er in de file staan om niet een variabele te veel in te lezen. Dat leidt namelijk tot een fout waarna de executie van het programma wordt afgebroken. Een goede manier om het aantal variabelen bij te houden en het inlezen van een file eenvoudig te houden, is ervoor te zorgen dat er slechts variabelen van een en het zelfde type in de file staan. Dus alleen string of numerieke variabelen. Verder zorgt u ervoor dat de eerste variabele in de file een getal is dat aangeeft hoeveel variabelen in de file staan.

Een voorbeeld:

```
10 INPUT "Hoeveel variabelen"; Y
20 OPEN "TEST2", O
25 REM Schrijf het aantal variabelen in de file
30 FWRITE "TEST2", Y
40 REM INPUT nu de variabelen
50 FOR X=1 TO Y
60 INPUT A
70 FWRITE "TEST2", A
80 NEXT X
90 CLOSE "TEST2"
100 REM Nu de file openen en aantal variabelen inlezen
110 OPEN "TEST2", I
120 FREAD "TEST2", Y
130 REM Y bevat nu het aantal variabelen in de file
140 REM Daarmee geven we het array de juiste grootte
150 DIM A(Y)
160 REM Lees nu het array in
170 FOR X=1 TO Y
180 FREAD "TEST2", A(X)
190 NEXT X
200 FOR X=1 TO Y
210 PRINT A(X)
220 NEXT X
230 END
```

Het is ook mogelijk om meerdere variabelen in een FREAD of FWRITE statement op te nemen, mits er gezorgd wordt voor scheidende komma's:

```
10 OPEN "TEST3", O
20 FOR X=1 TO 20 STEP 2
30 FWRITE "TEST3", X; ","; 3*X
40 NEXT X
50 CLOSE "TEST3"
60 DIM A(20)
70 OPEN "TEST3", I
80 FOR X=1 TO 20 STEP 2
90 FREAD "TEST3", A(X), A(X+1)
100 NEXT X
110 CLOSE "TEST3"
120 .
130 .
```

Let op regel 30 en de wijze waarop de komma tussen de twee variabelen wordt weggeschreven. U kunt de file TEST3.VAR nog eens met een TYPE bekijken om te zien hoe het er uitziet:

```
$TYPE TEST3.VAR

 1 , 3
 2 , 6
 3 , 9
.
.
```

3.9 RANDOM ACCES FILES

In de voorgaande paragraaf zijn we al uitgebreid ingegaan op het gebruik van sequentiele files. Zoals we gezien hebben is er aan het gebruik van een sequentiele file een specifiek nadeel verbonden. Het is erg lastig om wijzigingen aan te brengen in een sequentiele file. Daarvoor moeten we eerst de file openen, hem in zijn geheel inlezen, de file vervolgens sluiten, weer openen en vervolgens de gewijzigde variabelen wegschrijven. DOS65 Basic kan met een type file samenwerken dat er op gericht is dit probleem op te lossen. Dit type file heet random acces file. In een random file kunnen we zowel lezen als schrijven nadat we de file geopend hebben. Verder is het mogelijk om in een willekeurige volgorde variabelen te lezen of te schrijven in de file. We kunnen een file dus random openen, vervolgens een variabele in het midden wegschrijven, daarna een variabele aan het begin inlezen en bijvoorbeeld aan het eind van de file weer wegschrijven. Na het sluiten van de file zorgt DOS65 ervoor dat alle wijzigingen correct op de floppy worden weggeschreven. Om dit alles goed te laten verlopen is het wel nodig om ordening in de file aan te brengen. Dit wordt gedaan door de file in stukjes van gelijke lengte te verdelen. Een dergelijke eenheid wordt een record genoemd. Feitelijk vertoont een op deze manier georganiseerde file een grote overeenkomst met een kaartenbak. De kaartenbak als geheel stelt de file voor, de kaarten in de bak zijn de records. Elke kaart in de bak heeft dezelfde indeling. In het geval van een kaartenbak met adressen zou elke kaart (record) er bijvoorbeeld als volgt uitzien:

1. Naam
2. Voorletters
3. Straat
4. Postcode
5. PLaats
6. Telefoon

In elk record in de file staat derhalve dezelfde informatie, en deze informatie staat steeds in dezelfde volgorde. Het bovenstaande, een uniforme indeling van elk record is in de praktijk een voorwaarde voor het doelmatig werken met een random file. Bij het werken met een random file zullen we tevens moeten aangeven hoelang een record in de file is. Elk record in de file is even lang. Elk record in de file heeft een uniek nummer, het recordnummer. Het eerste record in de file heeft recordnummer 0, het tweede recordnummer 1 enz. Records opzoeken in de file gebeurt aan de hand van het recordnummer. Pas dan kunnen we in het record lezen of beschrijven.

De maximale lengte van een DOS65 file is 16 Mbyte. Een record in een random file moet minimaal 1 en mag maximaal 65535 bytes lang zijn. Een random file kan maximaal 65535 records bevatten. In de praktijk zullen deze uiterste waarden maar zelden bereikt worden. Gewoonlijk zal de omvang van een random file beperkt worden door de maximale capaciteit van de floppy-drive.

Het openen van een random file gebeurt met het OPEN sleutelwoord. De syntax is echter anders dan bij een sequentiele file:

OPEN "filenaam", R, <recordlengte>

DOS65 Basic herkent een random file aan de 'R' optie in het OPEN statement. De recordlengte is een integer die moet liggen tussen 1 en 65535. Bij het openen van een sequentiele file wordt een eventueel al bestaande file van de zelfde naam eerst verwijderd. Bij een random file is dat niet het geval. Een al bestaande file wordt intakt gelaten, maar u kunt er behalve uit lezen ook in schrijven. Bestaat de random file nog niet, dan wordt hij eerst gecreeerd. Na het openen heeft u dan een lege file waar u eerst in moet schrijven alvorens u hem kunt lezen. DOS65 Basic weet overigens zelf niet hoe lang een record in de random file is. (In feite weet DOS65 niet eens het verschil tussen een random file en een sequentiele file. Het is Basic die het random karakter aan de file geeft) U zult dat dus zelf moeten bijhouden. Geeft u de recordlengte op in de vorm van een gebroken getal dan wordt de fractie van het getal weggegooid (truncate). Een recordlengte van meer dan 65535 levert een 'Illegal quantity error' op.

Voor het selekteren van een record in de random file gebruiken we het RECORD sleutelwoord:

RECORD "filenaam", <recordnummer>

Het recordnummer moet liggen tussen 0 en 65535. Andere waarden leveren een 'Illegal quantity error' op. Het is alleen mogelijk om het RECORD sleutelwoord met een random file te gebruiken. Probeert u het RECORD sleutelwoord te gebruiken bij een file die sequentieel geopend is, dan volgt een 'Type mismatch error'. Datzelfde gebeurt ook wanneer u een random file met een recordlengte van 0 hebt geopend. Een gebroken getal als recordnummer wordt afgerond met een 'truncate'.

Om een record te lezen of te beschrijven gebruikt u dezelfde sleutelwoorden als bij een sequentiele file:

```
FREAD "filenaam", variabele(n)

FWRITE "filenaam", variabele(n)
```

Opgelet! Een record zelf is wel sequentieel opgebouwd. Net als bij een sequentiele file staan de variabelen achter elkaar in het record gescheiden door een return. Wilt u dus de laatste variabele uit het record inlezen dan moet u ze allemaal inlezen. Een record is dus in feite een miniatuur sequentiele file. Omdat het aantal variabelen in een record over het algemeen klein zal zijn, omdat elk record in de file er hetzelfde uitziet en omdat u de file niet hoeft te openen en te sluiten levert dit geen problemen op. Omgekeerd geldt natuurlijk ook dat u alle variabelen in een record moet inlezen en allemaal moet wegschrijven om een variabele te wijzigen in een record. In de praktijk zal het er meestal op neer komen dat een subroutine schrijft die een geheel record inleest en een subroutine die een geheel record wegschrijft. Het zal vaak voorkomen dat een record niet geheel gevuld is. In dat geval zullen bij het creeeren van het record de resterende bytes met nullen ($00) gevuld worden. Bij een volgende schrijfoperatie worden de nieuwe variabelen over de oude heen geschreven. Wat er dan in de ongebruikte bytes staat is afhankelijk van de voorgaande inhoud van het record. Indien u een record selekteert dat nog niet bestaat in de file, dan wordt het door DOS65 aangemaakt, inclusief alle tussenliggende records. Die tussenliggende records zijn allemaal leeg, d.w.z. met nullen gevuld. Stel dat u een random file "TEST" hebt met 10 records. Geeft u dan het statement:

```
RECORD "TEST", 15
```

Dan worden de records 11, 12, 13 en 14 aan de file toegevoegd. Al deze records zijn leeg. De lees/schrijf pointer staat na afloop van het statement op het begin van record 15 gericht, dat u dan kunt beschrijven (niet lezen natuurlijk). Probeert u toch uit zo'n leeg record te lezen, dan volgt er een 'Empty record' foutmelding. Datzelfde geldt feitelijk altijd als u een ascii null ($00) uit een file leest. Probeert u te lezen uit een record dat u met het RECORD statement aan een file hebt toegevoegd en waarin nog niet geschreven is dan volgt een 'End file' foutmelding.

Een geopende random file wordt tenslotte met het close statement gesloten:

```
CLOSE "filenaam"
```

DOS65 zorgt er dan voor dat alle updates in de random file correct worden weggeschreven op de floppy. In tegenstelling tot andere Basic's is het niet nodig om een gewijzigd record met een apart statement weg te schrijven. Dit wordt allemaal volautomatisch door DOS65 geregeld.

Tenslotte nog een voorbeeld van een eenvoudig programma met het gebruik van random files:

```
10 HOME
20 PRINT "Toevoegen, Inlezen, Einde   Uw keuze:";
25 GET KZ$ : IF KZ$ = "" THEN GOTO 25
30 IF KZ$ = "T" THEN GOSUB 1000
40 IF KZ$ = "I" THEN GOSUB 2000
50 IF KZ$ = "E" THEN END
60 GOTO 10


1000 INPUT; "Filenaam:"; FI$
1010 INPUT; "Recordlengte:"; RL
1020 OPEN FI$, R, RL
1030 HOME
1040 INPUT; "Recordnummer (100 is einde):"; RN
1045 IF RN = 100 THEN GOTO 1300
1050 RECORD FI$, RN
1060 CURSOR TO 10,25
1070 INPUT; "Naam        :"; NA$
1080 CURSOR TO 11,25
1090 INPUT; "Voorletters :"; VL$
1100 CURSOR TO 12,25
1110 INPUT; "Straat      :"; ST$
1120 CURSOR TO 13,25
1130 INPUT; "Postcode    :"; PC$
1140 CURSOR TO 14,25
1150 INPUT; "Woonplaats  :"; PL$
1160 REM schrijf nu het record weg
1170 FWRITE FI$, NA$
1180 FWRITE FI$, VL$
1190 FWRITE FI$, ST$
1200 FWRITE FI$, PC$
1210 FWRITE FI$, PL$
1220 GOTO 1030

1300 CLOSE FI$
1310 GOTO 10
```

```
2000 INPUT; "Filenaam:"; FI$
2010 INPUT; "Recordlengte:"; RL
2020 OPEN FI$, R, RL
2030 HOME
2040 INPUT; "Recordnummer (100 is einde):"; RN
2050 IF RN = 100 THEN GOTO 2300
2060 RECORD FI$, RN
2070 FREAD  FI$, NA$
2080 FREAD  FI$, VL$
2090 FREAD  FI$, ST$
2100 FREAD  FI$, PC$
2110 FREAD  FI$, PL$
2120 CURSOR TO 10,25
2130 PRINT "Naam        :"; NA$
2140 CURSOR TO 11,25
2150 PRINT  "Voorletters :"; VL$
2160 CURSOR TO 12,25
2170 PRINT  "Straat      :"; ST$
2180 CURSOR TO 13,25
2190 PRINT  "Postcode    :"; PC$
2200 CURSOR TO 14,25
2210 PRINT  "Woonplaats  :"; PL$
2220 CURSOR TO 17,25
2230 PRINT "Druk een willekeurige toets";
2240 GET TO$ : IF TO$ = "" THEN GOTO 2240
2220 GOTO 2030

2300 CLOSE FI$
2310 GOTO 10
```

Intern bestaan er nogal wat verschillen in de wijze waarop Basic 1.00 en 2.00 met Random files omgaat. Dat heeft voornamelijk te maken met verschillen tussen DOS 1.01 en 2.01. Bij Basic 1.00 zult u merken dat het achterwaarts zoeken van een record ongeveer vier tot vijf keer langzamer gaat dan het voorwaarts zoeken. Bij Basic 2.00 is dat probleem opgelost. Een typische zoektijd in een file van 64 Kbytes is 4 seconden van begin tot einde van de file. Daar staat tegenover dat het opvullen van lege records met nullen aan het eind van een file onder Basic 1.00 aanzienlijk sneller gaat dan onder Basic 2.00. In de praktijk is dat echter nauwelijks een bezwaar omdat het zoeken in een file veel vaker voorkomt.

3.10 FOUTAFHANDELING IN DOS65 BASIC

Wordt er tijdens het lezen of schrijven van een file een DOS fout ontdekt, dan worden alle geopende files gesloten. Datzelfde is het geval wanneer de executie van een programma door een andere Basic fout wordt afgebroken. Wordt de executie van het programma afgebroken door een END of STOP statement of na een control-C dan blijven geopende files open staan. Wanneer u met EXIT Basic verlaat worden overigens ook alle geopende files gesloten.

3.11 MAXIMALE AANTAL GEOPENDE FILES

Onder Basic 1.00 kunnen we maximaal vier files tegelijkertijd geopend hebben, onder Basic 2.00 zijn dat er zes. Hierbij dient u te bedenken dat een I/O redirect en de printer spooler elk voor een file tellen. Geeft u dus in een programma een DOS"PRINT ..." commando dan kunnen er nog maar vijf files geopend worden.

3.12 INPUT/OUTPUT REDIRECT

Input/output redirect is een belangrijk beginsel in DOS65. Het is niet de bedoeling van deze manual om dit beginsel uitgebreid uit te leggen, hiervoor zij verwezen naar de DOS65 manual. Bij I/O redirect komt de invoer van een programma als Basic uit een file of gaat de uitvoer naar een file. De invoerfile neemt daarbij de rol over van het toetsenbord, de uitvoerfile die van het beeldscherm. In zo'n invoerfile kunnen zowel direkte Basic kommando's staan als programmaregels. De kommando's worden geexecuteerd, de programmaregels in het geheugen gezet. Het is dus mogelijk een Basic programma met de editor te schrijven en als het klaar is met een I/O redirect in het werkgeheugen te laden. Ook is het mogelijk een stuurfile aan te maken waarin Basic commando's staan om die bijvoorbeeld een aantal programma's opstarten. Die programma's kunnen hun invoer ook uit de stuurfile krijgen. Op deze wijze is het mogelijk een applicatie te schrijven die zonder tussenkomst van de programmeur gerund wordt.

Een voorbeeld verduidelijkt een en ander.

```
$ CREATE TEST
&10 HOME
&20 FOR X=1 TO 10
&30 PRINT X,
&40 NEXT X
&50 END
&RUN
&EXIT
&^D
$
```

Nu typen we in:

```
$< TEST BASIC
.
.                       (Basic start op)
.
.
10 ........             (Het programma wordt gelist als werd
20 ........              het ingetypt)
30 ........
.
.
RUN                     (Het programma wordt gestart)
.
.                       (Uitvoer komt op het scherm)
.
Ok
EXIT                    (Terugkeer naar DOS)
$
```

Het is ook mogelijk om de uitvoer naar een file te sturen:

```
$< TEST > TEST.LOG BASIC
$                       (Er lijkt niets te gebeuren)
```

List nu de uitvoer door in te typen:

```
$ TYPE TEST.LOG
```

HOOFDSTUK 4

HET EXECUTE STATEMENT

4.1 HET EXECUTE SLEUTELWOORD

Een geheel nieuw sleutelwoord in DOS65 Basic is EXECUTE. EXECUTE is een zeer bijzonder commando. Voor zover ons bekend is er geen andere Basic die dit sleutelwoord kent. EXECUTE voert een Basic commando uit dat in een stringvariabele staat of in een quoted string. In beginsel kan elk legaal Basic statement geexecuteerd worden. De syntax is:

```
EXECUTE "string"
```

De string mag een maximale lengte hebben van 80 karakters. Een voorbeeld om een en ander duidelijk te maken:

```
A$ = "A=0.5 : B=SIN(A) : PRINT B"
```

Wanneer we deze string printen zien we:

```
PRINT A$
A=0.5 : B=SIN(A) : PRINT B
```

In feite staat in de string dus een Basic statement. Tik nu in:

```
EXECUTE A$
 .479425539
```

Het Basic statement dat in de stringvariabele A$ staat is nu geexecuteerd. Zoals u ziet biedt het EXECUTE statement allerlei mogelijkheden tot slim programmeren. Stel dat we een programma hebben waarin we gebruik willen maken van de verschillende mogelijkheden van een matrixprinter. Gewoonlijk zal dat betekenen dat we allerlei escape sequences naar de printer moeten sturen. Natuurlijk kunnen we dat door middel van print statements in het programma doen, maar het vervelende is dat elke fabrikant weer andere sequences voor zijn printer bedenkt. Dat

betekent dat iedereen die het programma overneemt het moet gaan bestuderen en regels aanpassen voor zijn eigen printer. Met behulp van het EXECUTE statement is een heel andere benadering van dit probleem mogelijk. Als voorbeeld nemen we hier een Epson MX80 printer. Deze printer heeft (onder andere) de volgende mogelijkheden:

1. Condensed characters -> SI
2. Cancel condensed mode -> DC 2
3. Enlarged characters -> SO
4. Cancel enlarged mode -> DC 4
5. Emphasized characters -> ESC E
6. Cancel emphasized mode -> ESC F
7. Underline on -> ESC - $01
8. Underline off -> ESC - $00

De daarbij behorende print statements zien er als volgt uit:

```
PRINT CHR$(15)
PRINT CHR$(18)
PRINT CHR$(14)
PRINT CHR$(20)
PRINT CHR$(27) ; "E"
PRINT CHR$(27) ; "F"
PRINT CHR$(27) ; "-"; CHR$(1)
PRINT CHR$(27) ; "-"; CHR$(0)
```

De bedoeling is nu dat deze statements niet in het Basic programma worden gebruikt maar dat ze in een aparte tekst file worden gezet. Dat doen we gewoon met de editor. Tevens zorgen we er voor dat de file de extensie .VAR krijgt. Laten we in dit geval die file PRINT.VAR noemen. U kunt de file gewoon uittypen met een TYPE:

```
$TYPE PRINT.VAR
PRINT CHR$(15)
PRINT CHR$(18)
PRINT CHR$(14)
PRINT CHR$(20)
PRINT CHR$(27) ; "E"
PRINT CHR$(27) ; "F"
PRINT CHR$(27) ; "-"; CHR$(1)
PRINT CHR$(27) ; "-"; CHR$(0)
$
```

Een dergelijke text file is vanuit DOS65 Basic als sequentiele file te benaderen. Vanuit een programma

laden we de print statements nu in een stringarray:

```
10 OPEN "PRINT", I
20 FOR X=1 TO 8
30 FREAD "PRINT", EX$(X)
40 NEXT X
50 CLOSE "PRINT"
.
.
```

Het array bevat nu acht print statements. Om bijvoorbeeld de printer in condensed mode te zetten doen we het volgende:

```
1000 REM zet de printer aan
1010 DEVICE 2 OUTPUT ON
1020 REM printer optie = 1 (condensed on)
1030 OP = 1
1040 EXECUTE EX$(OP)
.
.
```

Om de printer in enlarged mode met underline te zetten doen we het volgende:

```
2000 REM printer optie = 2 (cancel condensed mode)
2010 OP = 2
2020 EXECUTE EX$(OP)
2030 REM printer optie = 3 (enlarged mode)
2040 OP = 3
2050 EXECUTE EX$(OP)
2060 REM printer optie = 7 (underline on)
2070 OP = 7
2080 EXECUTE EX$(OP)
.
.
```

Het grote voordeel van de bovenstaande oplossing is dat iemand die uw programma overneemt alleen maar de file PRINT.VAR hoeft aan te passen. Kent zijn printer een bepaalde optie niet dan kan hij het betreffende print statement door een REM vervangen. Een REM statement kan ook geEXECUTEerd worden, het doet namelijk niets.

Een ander voordeel is dat de file PRINT.VAR vanuit elk Basic programma gebruikt kan worden. U hoeft dus de control statements voor uw printer maar een keer te schrijven. Daarna bent u voor eens en voor altijd van dat routinematige programmeerwerk verlost.

Het bovenstaande voorbeeld is maar een van de vele mogelijkheden die het EXECUTE statement biedt. Het EXECUTE statement maakt van DOS65 Basic een zeer flexibel stuk gereedschap. Wanneer u er verstandig gebruik van maakt kunt u er eenvoudige en compacte programma's mee schrijven.

APPENDIX 1

```
;*********************************
;*                               *
;*     Zero page adresses        *
;*                               *
;*********************************
nljmp   equ     $00     ;new line jump, points to READY
usrjmp  equ     $03     ;USR jump
flpfix  equ     $06     ;vector to 'floating to fixed' subr.
fixflp  equ     $08     ;vector to 'fixed to floating' subr.
srcchr  equ     $0a     ;search character
endchr  equ     $0b     ;end delimiter in LIST, CRUNCH
count   equ     $0c     ;general counter for BASIC
valtyp  equ     $0e     ;variable type: numeric or string
intflg  equ     $0f     ;numeric type: integer or floating
inpflg  equ     $10
subflg  equ     $11     ;subscript flag: FNX flag
sign    equ     $12     ;flag: $00=input, $40=GET, $98=READ
domask  equ     $13     ;mask used by <,=,>
trflg   equ     $14     ;suppress input flag
poslin  equ     $16     ;position on print line
width   equ     $17     ;maximum print line width
collim  equ     $18     ;input column limit
linnum  equ     $19     ;(line)number read by LINGET
temppt  equ     $1b     ;current descriptor stack pointer
tempst  equ     $1e     ;top of descriptor stack
index   equ     $27     ;indirect index pointer
index1  equ     $27     ;indirect index pointer #1
index2  equ     $29     ;indirect index pointer #2
txttab  equ     $30     ;pointer: start of program text
vartab  equ     $32     ;pointer: start of variables
arytab  equ     $34     ;pointer: start of arrays
strend  equ     $36     ;pointer: end of arrays
fretop  equ     $38     ;pointer: start of real strings
frespc  equ     $3a     ;pointer: top of free string space
memsiz  equ     $3c     ;pointer: top of RAM
curlin  equ     $3e     ;current line number
oldlin  equ     $40     ;last line number (for CONT)
oldtxt  equ     $42     ;old txtptr (for CONT)
datlin  equ     $44     ;current data line
datptr  equ     $46     ;pointer to current data statement
inpptr  equ     $48     ;pointer to inputted value
varnam  equ     $4a     ;current variable name
forpnt  equ     $4e     ;current FOR variable
lstpnt  equ     $4e     ;pointer for LIST
opptr   equ     $50     ;pointer in table
vartxt  equ     $50     ;index to current operator
opmask  equ     $52     ;mask created by current operator
defpnt  equ     $53     ;pointer to function definition
dscpnt  equ     $55     ;pointer to descriptor
tempf1  equ     $5c     ;temporary FLP accu
highds  equ     $5d     ;destination pointer in BLT
hightr  equ     $5f     ;source pointer in BLT
deccnt  equ     $62     ;# of digits before decimal point
tenexp  equ     $63     ;exponent base ten in FOUT
lowtr   equ     $64     ;pointer for LIST
fac     equ     $66     ;floating point accu #1
facexp  equ     $66     ;exponent of flp accu #1
facho   equ     $67     ;MSB mantissa of FLP accu #1
facmoh  equ     $68     ;mantissa, upper middle, of FLP accu #1
facmo   equ     $69     ;mantissa, lower middle, of FLP accu #1
```

```
faclo   equ     $6a     ;LSB mantissa of FLP accu #1
facsgn  equ     $6b     ;sign of FLP accu #1
argexp  equ     $6e     ;exponent of FLP accu #2
argho   equ     $6f     ;MSB mantissa of FLP accu #2
argmoh  equ     $70     ;mantissa, upper middle, of FLP accu #2
argmo   equ     $71     ;mantissa, lower middle, of FLP accu #2
arglo   equ     $72     ;LSB mantissa of FLP accu #2
argsgn  equ     $73     ;sign of FLP accu #2
arisgn  equ     $74     ;sign comparison of FLP accus
strng1  equ     $74     ;pointer to a string
facov   equ     $75     ;overflow of FLP accu #1
bufptr  equ     $76     ;index save in CRUNCH
strng2  equ     $76     ;2nd pointer to a string
fbufpt  equ     $76     ;index in output in FOUT
chrget  equ     $78     ;gets next char. from program text
chrgot  equ     chrget+6  ;gets current char. from prog. text
txtptr  equ     chrget+7  ;pointer to current char.
qnum    equ     chrget+13 ;label in chrget
iodev   equ     $95     ;temporary for ON/OFF token
devnr   equ     $96     ;temporary for device #
strlen  equ     $97     ;temporary for string length
inqflg  equ     $98     ;flag for question mark with INPUT
rndpnt  equ     $99     ;pointer used by RECORD
runmod  equ     $9b     ;flag for running mode
rndp2   equ     $9c     ;pointer used by RECORD (v2.00 only)
;*********************************
;*                               *
;* Zero page locations used by   *
;* functions.                    *
;*                               *
;*********************************
integr  equ     $0a     ;one byte integer from QINT
dimflg  equ     $0d     ;flag: subscripts and integers allowed
garbfl  equ     $10     ;garbage collect counter
tansgn  equ     $13     ;flag sign of TAN
poker   equ     $19     ;adress to be PEEKed or POKEd
tempf3  equ     $53     ;extra FLP accu used by power
lastpt  equ     $1c     ;last descriptor on descr. stack
resho   equ     $2b     ;MSB mantissa of extra FLP accu
resmoh  equ     $2c     ;upper middle mantissa of same
addend  equ     $2d     ;stepsize in array size calc
resmo   equ     $2d     ;lower middle mantissa of FLP accu
reslo   equ     $2e     ;LSB mantissa of same
fdecpt  equ     $4c     ;index in FLP constants OUT
varpnt  equ     $4c     ;pointer to current variable
andmsk  equ     $4e     ;AND mask in WAIT
eormsk  equ     $4f     ;EOR mask in WAIT
jmper   equ     $59     ;indirect pointer for function JMP
size    equ     $55
oldov   equ     $5b     ;old overflow of FLP accu1
arypnt  equ     $5d     ;pointer in array building
tempf2  equ     $61     ;temporary FLP accu2
lowds   equ     $62     ;pointer to last byte dest. in BLT.
dptflg  equ     $64     ;flag: decimal point found in FIN
expsgn  equ     $65     ;sign of exponent base 10 in FOUT
dsctmp  equ     fac     ;temporary descriptor area
indice  equ     $69     ;index for DIM is left here
sgnflg  equ     $6c     ;sign of mantissa base 10 in FOUT
degree  equ     $6c     ;grade of polynome
bits    equ     $6d     ;bit counter in FLP accu normalize
curtol  equ     $76     ;absolute linear index
polypt  equ     $76     ;pointer to polynome constants
rndx    equ     $90     ;current random number seed
```

APPENDIX 2

Overzicht van alle sleutelwoorden, operators en functies in DOS65 Basic

Sleutelwoorden

END	FOR	NEXT	DATA	INPUT	DIM
READ	LET	GOTO	RUN	IF	RESTORE
GOSUB	RETURN	REM	STOP	ON	EXIT
WAIT	LOAD	SAVE	MERGE	DEVICE	INVERSE
NORMAL	HOME	CURSOR	TRACE	OPEN	CLOSE
FREAD	FWRITE	CALL	RECORD	EXECUTE	DEF
POKE	PRINT	CONT	LIST	CLEAR	GET
NEW	DOS	MONITOR	TAB(	TO	FN
SPC(	THEN	ELSE	NOT	STEP	OUTPUT
OFF					

Operators

+	-	*	/	^	AND
OR	>	=	<		

Functies

SGN	INT	ABS	FRE	POS	SQR
RND	LOG	EXP	COS	SIN	TAN
ATN	PEEK	LEN	STR$	HEX$	VAL
ASC	CHR$	LEFT$	RIGHT$	MID$	