BASIC
KORTE INFORMATIE

INHOUD

LEVEL II BASIC

Indien u de ASTER CT-80 kocht met de TRS-80 Level II BASIC ROM's geinstalleerd dan kunt u de BASIC exact zo behandelen als bij de TRS-80 het geval is.
Indien u de ASTER CT-80 kocht met CP/M als Operating Systeem en daarbij MBASIC als hogere programmeertaal bestelde, dan kunt u ook de volgende besprekingen en voorbeelden van BASIC hanteren. MBASIC en TRS-80 BASIC zijn vrijwel identiek. Raadpleeg eerst het hoofdstuk VERSCHILLEN TUSSEN MBASIC EN TRS-80 BASIC. Beide BASIC's zijn namelijk afkomstig van Microsoft. Deze software-fabrikant maakt voor vrijwel alle merken microcomputers de BASIC taal.

Korte beschrijving kommando's

In de volgende hoofdstukken worden alle kommando's uitgebreid beschreven; toch is het gemakkelijk alle kommando's met een korte verklaring bij elkaar te hebben. Als u snel de betekenis van een kommando wilt weten dan kunt u op deze pagina's snel het gewenste vinden.

DIREKTE KOMMANDO's:

Uitdrukking (Statement)	Betekenis
AUTO 10	Automatisch regelnummeren bij programmeren
AUTO 100	Idem, echter met 100 tegelijk omhoog
CLEAR 1000	Reserveer 1000 bytes voor string ruimte, zet alle variabelen op '0'
CLOAD "A"	Laad cassetteprogramma met naam "A"
CLOAD ?"A"	Verifieer geladen programma met cassette
CONT	Ga verder met programma na BREAK of STOP
CSAVE "A"	Schrijf programma met naam "A" naar cassette
DELETE 30-80	Wis BASIC regels 30 t/m 80 uit geheugen
EDIT 40	Ga in EDIT mode en display begin van regel 40
LIST 100	Display regel 100 op het beeldscherm
LIST 50-100	Display regel 50 t/m 100 op het beeldscherm
LIST 300-	Display alle programmaregels vanaf regel 300
LLIST 100	Druk regel 100 af op de printer
	Alle voorbeelden van LIST zijn van toepassing op LLIST
NEW	Wis programma uit geheugen
RUN 100	Start programma op regel 100
	Indien geen regelnummer, start bij laagste regel
SYSTEM	Schakel over op monitor om Assembler programma te laden
TROFF	Schakel trace uit
TRON	Schakel trace in

BASIC FUNKTIES

Funktie	Resultaat
ABS(getal)	Absolute waarde van getal
ASC(karakter)	Decimale waarde van karakter
ATN(getal)	Arctangens van getal in radialen
CDBL(getal)	Dubbel precisie getal
CHRS(getal)	Karakter wiens decimale waarde tussen haakjes staat, bijv. 65=A
CINT(getal)	Heel getal. 2.5=2, -2.5=3. Bereik: -32768 tot +32767
COS(getal)	Cosinus van getal (getal is in radialen)
CSNG(getal)	Enkele precisie getal
ERL	Regelnummer van ontstane fout
FIX(getal)	Verwijder decimalen van getal, bijv. 5.25=5
FRE(karakter)	Beschikbare hoeveelheid String geheugen
FRE(getal)	Beschikbare hoeveelheid niet-String geheugen
INKEY$	Geeft letterwaarde van toets welke ingedrukt werd
INP(poortnummer)	Geeft decimale waarde van gelezen byte op input-poort
INT(getal)	Heel getal (2.5=2, -2.5=-3 etc.), bereik onbegrensd
LEFT$(string,getal)	Geeft substring van karakters met lengte van getal, vanuit linkerzijde string
LEN(string)	Geeft lengte van string
LOG(getal)	Geeft logaritme van getal
MEM	Geeft het ongebruikte aantal geheugenbytes van het niet-beschermd geheugen
MID(string,start,lengte)	Substring van karakters met start en aantal karakters uit hoofdstring
PEEK(adres)	Geeft decimale waarde van geheugenlokatie
POINT(horizontaal,vertikaal)	Grafisch blok, aan=TRUE(-1), uit=FALSE(0)
POS(0)	Geeft cursor positie op een regel (0-63)
RIGHT$(string,getal)	Geeft substring uit string met lengte van getal, vanaf rechterkant van string
RND(getal)	Geeft willekeurig getal tussen nul(0) en getal
RND(0)	Geeft willekeurig getal tussen nul (0) en een (1)
SGN(getal)	Geeft positieve of negatieve waarde van getal, negatief=-1, nul=0, positiev=+1
SIN(getal)	Geeft sinus van getal, getal is in radialen
SQR(getal)	Geeft vierkantswortel van getal
STR$(getal)	Geeft karakter string van getal
STRING$(getal,variabele)	Geeft string van karakters, lengte van getal
TAN(getal)	Geeft tangens van getal, getal is in radialen
USR(0)	Voert machinetaal routine uit
VAL(string)	Geeft getal uit een karakterstring
VARPTR(variabele)	Geeft het adres van een variabele in het geheugen

BASIC Statements

Uitdrukking (Statement)	**Betekenis**
CLEAR bytes	Reserveer geheugen voor string ruimte
CLS	Maakt scherm schoon, zet formaat eventueel terug naar 16 x 64, zet cursor linksboven
DATA getal,letter	Maak tabel in geheugen van letters of cijfers
DEFDBL karakter	Definieer dubbel precisie variabele(n)
DEFINT karakter	Definieer heel-getal variabele(n)
DEFSTR karakter	Definieer karakter variabele(n)
DIM variabele(getal,getal,...)	Definieer een array met zijn dimensies
END	Beeindig werking van programma
ERROR code	Simuleer een fout-conditie (zie foutcodes)
FOR var=getal TO getal	Herhaal tot dmv. NEXT het tweede getal bereikt is
STEP getal	Grotere stappen (getal) door FOR-NEXT herhaling
GOSUB regel	Tijdelijke sprong naar subroutine tot aan RETURN
GOTO regel	Sprong naar ander gedeelte van programma
IF conditie THEN aktie ELSE tweede aktie	Voorwaardelijke aktie gebaseerd op waar of niet waar zijn van konditie
INPUT variabele of string	vraag om getal of string, beeindig met ENTER
INPUT -..,variabele	Lees data van tape of disk
LET variabele=waarde	Geeft variabele een waarde, bijv. LET A=5, LET is optioneel
LPRINT variabele,string	Print data
LPRINT TAB(getal) variabele,string	Print met tabulators
LPRINT USING specifikatie;variabele,string	Print met gebruikmaking van vastgesteld formaat
NEXT variabele	Einde van FOR NEXT herhaling
ON ERROR GOTO regelnr.	Indien fout ontstaat ga naar ander gedeelte van programma
ON getal GOSUB regel,regel regel is eerste regel van subroutine	Ga op getal naar eerste, tweede enz. regel
ON getal GOTO regel,regel	Ga op getal naar eerste, tweede enz. regel
OUT poort,getal	Schrijf getal naar output poort, max. 255 decimaal
POKE adres,getal	Plaatst getal, max. 255 decimaal, rechtstreeks in geheugenplaats
PRINT variabele,string	Geeft variabele of string op beeldscherm weer
PRINT TAB (getal);variabele,string	Als print, doch met tabulator funktie
PRINT -..,variabele	Schrijf data naar tape of disk via buffers
PRINT USING specifikatie;variabele,string	Geef weer op beeldbuis met gebruikmaking van vastgesteld formaat
PRINT a getal, variabele of string	Druk op scherm af op positie zoals aangegeven in eerste getal, in TRS80 formaat max. 1023
READ waarde, waarde	Lees data van tabel uit geheugen welke werd opgezet met DATA statement
REM of '	Na dit teken of woord kan tekst worden geplaatst die niet van direkte invloed is op programma, bijv. voor commentaar

RESET (horizontaal,vertikaal)	Schakel grafisch blok op scherm uit op gegeven positie
RESTORE	Zet teller welke DATA uit geheugentabel inleest op nul, nieuwe READ begint op nul
RESUME regel NEXT	Einde van foutbehandelings-routine en sprong naar gewenste regel
RETURN	Terugkeer uit sub-routine na een GOSUB statement
SET (horizont.,vertik.)	Schakel grafisch blokje op scherm aan op gewenste positie
STOP	Geprogrammeerde stilstand van programma

BASIC foutmeldingen en codes

/0	11	Delen door nul, of getal te klein
BS	09	Aangeroepen array met ongeldige toevoeging
CN	17	Continueren van programma onmogelijk
DD	10	Poging om array opnieuw te dimensioneren zonder te CLEARen
FC	05	Ongeldige parameter voor een funktie
FD	22	Foutieve of buiten-tabel DATA werd gelezen
ID	12	Onjuist gegeven INPUT als direkt kommando
LS	15	String langer dan 255 tekens
MO	21	Instruktie mist toevoeging
NF	01	Poging NEXT voor FOR uit te voeren
NR	18	RESUME niet mogelijk daar programma eerder werd beeindigd
OD	04	Data niet vindbaar bij READ of INPUT, te weinig data
OM	07	Onvoldoende geheugen beschikbaar, alles gebruikt of gereserveerd
OS	14	Onvoldoende string ruimte beschikbaar, indien nodig meer reserveren
OV	06	Getal is te groot voor de computer, overflow
RG	03	Poging RETURN eerder uit te voeren dan GOSUB
RW	19	poging RESUME eerder uit te voeren dan ON ERROR GOTO
SN	02	Syntax fout (benaming, spelling, formaat etc.) 16 String behandeling te complex,
ST		Onderverdelen in kleinere eenheden
TM	13	Verkeerde soort data in statement gebruikt, letter of cijfer
UE	20	Het ERROR statement heeft een foute code
UL	08	Verwijzing naar niet bestaande programmaregel

DIVERSE BASIC FEITEN

Integer	-32768 tot 32767
Enkele-precisie	-1.701411E+-38 tot 1.701411+-38
Dubbele-precisie	-1.701411834544556D+-38
	tot 1.701411834544556D+-38
Karakterstring	0 tot 255 bytes
Regel-nummer	0 tot 65529
Gekodeerde regellengte	0 tot 255 bytes

MINIMAAL AANTAL GEHEUGEN PLAATSEN

Programma regel	5 bytes
Gereserveerd woord	1 bytes
Operator	1 bytes
Variabele naam	1 bytes
Speciaal karakter	1 bytes
Constante karakter	1 bytes

MINIMAAL AANTAL GEHEUGENPLAATSEN VOOR HET UITVOEREN VAN:

Integere variabele	5 bytes
Enkele precisie variabele	7 bytes
Dubbele precisie variabele	11 bytes
String variabele	6 bytes
Array variabele	12 bytes
FOR-NEXT herhaling	16 bytes
GOSUB	6 bytes
Waarde tussen haakjes	16 bytes

GERESERVEERDE WOORDEN IN BASIC

@	FOR	POKE
ABS	FORMAT	POS
AND	FRE	POSN
ASC	FREE	PRINT
ATN	GET	PUT
AUTO	GOSUB	RANDOM
CDBL	GOTO	READ
CHR$	IF	REM
CINT	INKEY$	RENAME
CLEAR	INP	RESET
CLOCK	INPUT	RESTORE
CLOSE	INSTR	RESUME
CLS	INT	RETURN
CMD	KILL	RIGHT$
CONT	LEFT$	RND
COS	LET	RSET
CSNG	LSET	SAVE
CVD	LEN	SET
CVI	LINE	SGN
CVS	LIST	SIN
DATA	LOAD	SQR
DEFDBL	LOC	STEP
DEFFN	LOF	STOP
DEFINT	LOG	STRING$
DEFSNG	MEM	STR$
DEFUSR	MERGE	TAB
DEFSTR	MID$	TAN
DELETE	MKD$	THEN
DIM	MKI$	TIME$
EDIT	MKS$	TO
ELSE	NAME	TROFF
END	NEW	TRON
EOF	NEXT	USING
ERL	NOT	USR
ERR	ON	VAL
ERROR	OPEN	VARPTR
EXP	OR	VERIFY
FIELD	OUT	
FIX	PEEK	
FN	POINT	

CONTROLE CODES

Het uitvoeren of op het beeldscherm printen door middel van bijvoorbeeld het BASIC Statement CHR$(..) van bepaalde codes kan soms effekten tot gevolg hebben die misschien niet direkt duidelijk zijn. Onderstaand een tabel met de zgn. Controle Codes en hun betekenis.

Decimaal	Hexadecimaal	Funktie
0-7	00-07	Geen funktie
8	08	Cursor een spatie terug, karakter wordt gewist
9	09	Geen funktie
10	0A	Regelopschuiving en wagenterugloop (LF+CR)
11	0B	Begin van blad code
12	0C	Begin van blad code
13	0D	Regelopschuiving en wagenterugloop (LF+CR)
14	0E	Zet cursor aan
15	0F	Zet cursor uit
16-22	10-16	Geen funktie
23	17	Schakel over naar 32 karakters per regel op videoscherm
24	18	Cursor een plaats terug
25	19	Cursor een plaats vooruit
26	1A	Cursor een regel naar beneden
27	1B	Cursor een regel naar boven
28	1C	Schakel over naar 64 karakters per regel, plaats cursor links bovenaan videoscherm op positie 0,0
29	1D	Breng cursor naar begin van regel waarop cursor staat
30	1E	Wis scherm vanaf cursor positie tot einde regel
31	1F	Wis scherm vanaf cursor positie tot einde scherm

BASIC
DISK-BASIC
DOS

ABS(A)

Geeft de absolute waarde van het argument (x). ABS(A)=A als A groter dan of gelijk aan nul (0) is en ABS(A)=-A indien A kleiner is dan nul(0).

AND

Uitdrukking uit Boleaanse Algebra. Wordt gebruikt voor vergelijkingen. Kan in BASIC worden toegepast om een voorwaardelijke programmasprong te maken.
Voorbeeld:

```
10 IF B=D AND F=C THEN GOTO 100
```

Slechts als B gelijk is aan D en F is gelijk aan C dan wordt naar regel 100 gegaan. In alle andere gevallen zal het programma regel 10 verlaten en gewoon doorgaan met de eerstvolgende regel.

ASC(string)

Geeft de ASCII waarde van het eerste karakter van de tussen haakjes geplaatste string in decimale vorm. Indien de string geen waarde heeft, dus nul is, zal een foutmelding gegeven worden.
Voorbeeld:

```
50 PRINT ASC(B)
```

Op het beeldscherm zal de decimale waarde van B worden gegeven.
Voorbeeld:

```
60 PRINT ASC('X')
70 PRINT ASC('XY')
```

Regel 60 en 70 zullen beide dezelfde waarden geven.
Het argument mag uitgebreider zijn dan bovenstaande voorbeelden. Complexe bewerkingen zijn hierdoor mogelijk.
Voorbeeld:

```
80 PRINT ASC(LEFTS(X$,1))
```

Regel 80 zal de decimale waarde weergeven van de eerste letter van de string X$.

Het ASC statement kan gebruikt worden voor het omzetten van kleine letters in hoofdletters onder basic. Zoals u misschien uit de ASCII tabel heeft kunnen opmaken is het verschil in decimale waarde tussen hoofd- en kleine letters exact 32. Indien u dus een waarde uitleest d.m.v. ASC welke kleine letters vertegenwoordigen dan kunt u door het bijtellen van 32 en het gebruiken van het statement CHR$(..) karakters naar het hoofdletter formaat omzetten. Zie hiervoor de uitleg bij CHR$.

ATN(A)

Dit statement geeft de absolute waarde van het getal A. ABS(A)=A als A gelijk of groter is dan nul(0). ABS(A)=-A als A kleiner is dan nul(0).
Voorbeeld:

10 IF ABS(A) 1E-5 THEN PRINT 'GETAL TE KLEIN'

AUTO
Het kommando AUTO is bijzonder gemakkelijk voor het intypen van BASIC programma's. Nadat het kommando wordt ingetypt wacht BASIC totdat u aangeeft met welke hoeveelheid u de regels wilt aanvangen en met welke waarde u wilt ophogen.
Voorbeeld:

AUTO

De BASIC zal nu automatisch starten met regelnummer 10. Iedere ENTER zal een regelnummer genereren dat 10 hoger is dan het voorgaande.
Voorbeeld:

AUTO 10,5

De BASIC zal nu starten met regelnummer 10. Ieder ENTER zal een regelnummer genereren dat 5 hoger is dan het voorgaande.
Voorbeeld:

AUTO 1000,50

De BASIC zal nu starten met regelnummer 1000. Ieder ENTER zal een regelnummer genereren dat 50 hoger is dan het voorgaande.

Indien u dit automatisme wenst te stoppen dan is het indrukken van de BREAK-toets voldoende.
Als regelnummers reeds werden toegepast (bijvoorbeeld bij het toevoegen van regels aan een bestaand BASIC-programma) dan zal het gebruik van het AUTO kommando een asterix (ster) veroorzaken direkt na ieder automatisch afgedrukt regelnummer dat reeds BASIC kommando's bevat.

CDBL(A)

Geeft een dubbel precisie representatie van het argument tussen haakjes. De ontstane waarde is maximaal de 17 significante cijfers.
CDBL is goed toepasbaar indien u normalerwijze enkele precisie of integere getallen gebruikt maar toch voor een enkele berekening dubbel precisie wilt toepassen.
Voorbeeld:

10 FOR A%=1 TO 10:PRINT1/CDBL(A%);:NEXT

CHR$(getal)

Dit statement geeft het karakter weer dat volgens de ASCII tabel hoort bij het getal tussen haakjes. Het getal mag liggen tussen 0 en 255. Ook mogen variabelen worden toegepast.
Voorbeeld:

```
10 PRINT ASC(65)
```

```
10 A=65
20 PRINT ASC(A)
```

Het eerste en het tweede voorbeeld zullen beide de letter 'A' als resultaat geven.

Aanhalingstekens zijn normalerwijze niet toepasbaar in Strings. Door middel van het ASC statement kunnen toch aanhalingstekens worden toegepast. Bij het raadplegen van de ASCII-tabel zien we dat de aanhalingstekens (") de ASCII waarde 34 vertegenwoordigen. B$=ASC(34) geeft B$ de waarde ".
Voorbeeld:

```
10 B$=CHR$(34) 20 PRINT"Hij zei,";B$;"Hallo";A$
```

Deze regels geven als resultaat:

Hij zei,"Hallo"

CINT(A)

Geeft de grootste integer, niet groter dan het getal of variabele tussen haakjes.
Voorbeeld:

```
10 PRINT CINT(3.5)
```

Geeft als resultaat 3.

```
20 print cint(-10.5)
```

Geeft als resultaat -10.

Het maximale bereik is +32767 tot -32768.

CLEAR getal

Indien direkt gebruikt, zonder getal, worden 50 bytes gereserveerd als string-geheugen. Bij het opstarten van een programma is het belangrijk te weten hoeveel string-geheugen u nodig hebt. Als te weinig string geheugen wordt geCLEARed resulteert dit in een OUT OF STRING SPACE foutmelding.
Ieder getal tussen 0 en +32767 mag worden toegepast. Negatieve getallen geven een foutmelding. Als geen string geheugen wordt toegepast in een programma mag CLEAR 0 worden gebruikt.
Voorbeeld:

```
10 CLEAR 2000
```

Geeft 2000 bytes ruimte voor string geheugen.

CLOCK — DISK

Bij gebruikt van Disk Operating Systeem (DOS) zorgt het statement CLOCK ervoor dat rechts bovenaan het scherm de tijd wordt weergegeven. Doordat de klok-funktie interrupt gestuurd is onder het DOS en geen hardware funktie van de ASTER CT-80 is, zal iedere andere interrupt een vertraging kunnen veroorzaken in het bijhouden van de tijd.
Het schrijven en lezen van diskettes bijvoorbeeld zal een onjuiste tijdmelding tot gevolg hebben. Voor de ASTER CT-80 is een hardware klok board verkrijgbaar dat de juiste tijd tot op honderdsten van seconden bijhoudt. Het in- of uitschakelen van de netspanning is dan niet van invloed op de eenmaal geprogrammeerde tijd of datum.

Normaal wordt onder DOS opgestart onder de konditie CLOCK OFF.

Indien de klok moet worden aangezet dan is het volgend kommando onder DOS voldoende:

CLOCK

Het uitschakelen onder DOS wordt gedaan door middel van:

CLOCK (OFF)

CLOSE — DISK

Dit kommando beeindigt het gebruik van een open file d.m.v. een reeds eerder gespecificeerde buffer. Indien de buffer voor een file niet eerder werd geopend heeft het CLOSE statement geen effekt. Tegelijkertijd kunnen meerdere buffers worden gesloten. Voorbeeld:

100 CLOSE 2,3,7

Bovenstaande programmaregel sluit de buffers 2, 3 en 7. Eventueel kunnen de gesloten buffernummers aan andere files worden toegekend.

Het niet sluiten van openstaande buffers/files tijdens disk gebruik kan enorme gevolgen hebben voor de opgeslagen informatie. Indien een diskette wordt verwijderd tijdens het niet gesloten zijn van een buffer dan zal over het algemeen de gehele file onleesbaar geworden zijn. Data opgeslagen in een buffer welke nog niet geheel werd gevuld wordt pas naar disk weggeschreven na het CLOSE'n van een buffer/file.

Onder sommige omstandigheden worden files/buffers automatisch geCLOSE'd. Deze omstandigheden zijn:

DISK Fouten
EDITen van een file
CLEAR
NEW
RUN
MERGE
Wissen, corrigeren of toevoegen van programma regels

CLS

Dit kommando betekent 'maak het scherm schoon' oftewel 'clear Screen'. Alle informatie op het video scherm wordt gewist, de cursor wordt naar de meest lage positie (0,0) gebracht, links bovenaan het videoscherm. Uiteraard is het mogelijk om PRINT statements te blijven herhalen. Het scherm "scrollt' automatisch omhoog als nieuwe informatie onderaan het scherm wordt bijgevoegd. Veel netter kan uw video scherm eruit zien als regelmatig van het CLS kommando gebruik wordt gemaakt.

CMD"..." **DISK**

Via het CMD kommando kunnen diverse kommando's uit het DOS vanuit BASIC worden toegepast. Het DOS van de TRS-80 TRS-DOS 2.3 kent dan via BASIC de kommando's CMD"D", CMD"R", CMD"S", CMD"T". Het Disk Operating Systeem NEWDOS 80 2.0 herkent onder BASIC via het CMD kommando vrijwel alle DOS kommando's.
Voorbeelden van TRS-DOS CMD kommando's:

CMD"D" **DISK**

Indien gegeven tijdens BASIC zal het DEBUG programma geladen worden en direkt toegepast kunnen worden. Doordat het DEBUG programma zich in het adres-bereik direkt onder de DISK BASIC bevindt, kan met G en daarna ENTER de BASIC weer worden ingeschakeld, zonder dat het BASIC programma dat aanwezig was wordt gewist.
Iedere keer dat tijdens een BASIC programma op BREAK wordt gedrukt nadat CMD"D" eenmaal was aangeroepen, zal het DEBUG programma automatisch worden gestart.
Indien vanuit DEBUG opnieuw opgestart moet worden, is het voldoende om G5200 te typen. In het geheugen aanwezige BASIC programma's en hun variabelen worden gewist.

CMD"I",doskommando **DISKBASIC**

Dit kommando is gelijk aan CMD"S", er kan echter direkt een DOS kommando mee worden gegeven. Het DOS kommando mag echter geen verdere toevoegingen hebben, zoals drive-nummer etc.
Voorbeeld:

CMD"I",'TEKST'

Vanuit BASIC wordt nu overgegaan naar het Disk Operating Systeem en direkt het programma TEKST opgestart.

CMD"R" **DISKBASIC**

Na gebruik van cassette I/O dient dit kommando gegeven te worden om de klok routine weer te starten. Interrupts worden weer herkend.

CMD"S" **DISKBASIC**

Met dit kommando verlaat u Disk BASIC en wordt overgeschakeld op het Disk Operating Systeem. U verliest het BASIC programma en alle variabelen. SAVE eerst naar disk voor het gebruik van dit kommando.

CMD"T" **DISKBASIC**

Stop de klok en stop alle interrupts. Dit kommando moet gegeven worden voordat u cassette I/O wilt plegen. De cassette routines zijn tijd-gebaseerd. Interrupts - dus ook de klok-routine - verbreken de nauwkeurige tijdberekening nodig voor het herkennen van de informatie van de cassetterecorder.
Voorbeeld:

```
100 OPEN"I",1,"FILE/BAS"
110 CMD"T":INPUT//1,A,B
120 CMD"R"
130 PRINT A,B
```

Onderstaande kommando's moeten worden voorafgegaan door het CMD"T" kommando:
CLOAD
CLOAD?
CSAVE
INPUT//-1
INPUT//-2
PRINT//-1
PRINT//-2
SYSTEM

CONT

Na het stoppen van een programma door middel van de BREAK-toets of doordat een STOP kommando in het programma werd toegepast kan door het CONT kommando gekontinueerd worden zonder dat variabelen verloren gaan.
Het gebruik van STOP en CONT tesamen kan handig zijn om tussentjds variabelen te kontroleren op hun waarde.
Voorbeeld:

```
10 FOR A=1 TO 10
20 IF A=5 THEN STOP
30 NEXT A
```

In regel 20 zal gestopt worden zodra de FOR NEXT herhaling voor de vijfde maal gedaan is. Indien u dan tijpt: PRINT A dan krijgt u de waarde 5 op het beeldscherm te zien. U kunt het programma verder laten lopen door nu het kommando CONT in te tijpen.

COS(getal)

Dit statement geeft de cosinus van het tussen haakjes geplaatste getal. Om bijvoorbeeld de cosinus van Y te verkrijgen als Y in graden wordt uitgedrukt dan is de volgende programmaregel toepasbaar:

10 A=COS(Y+3.3)

CSNG(getal)

Geeft een enkele precisie van het getal tussen haakjes. Als het originele getal een dubbel-precisie getal is dan worden de 6 meest signifikante cijfers weergegeven met een 4/5 afronding in het minst signifikante cijfer. Voorbeeld:

10 PRINT CSNG(.5555555555556) geeft .555556

20 PRINT CSNG(.4444444444444) geeft .444444

CVD **DISKBASIC**

Het tegenovergestelde van MKD$. Met CVD verandert u data naar numerieke vorm. Toepasbaar na het lezen van informatie van Disk. Voorbeeld:

Stel dat de naam WINST$ refereert aan een 8-byte field in een random-acces file buffer. Nadat een GET uitgevoerd is op het record, bevat de string WINST$ een MKDS representatie van het getal 12345.67. Als we vaststellen dat KOSTEN een getal vertegenwoordigt groot 2340, dan zal de volgende programmaregel: .
10 PRINT CVD(WINST$)-KOSTEN

als resultaat het getal 10005.67 geven.

Indien we de programmaregel als volgt toepassen:

10 PRINT WINST$-KOSTEN

dan wordt een TYPE MISMATCH foutkode gegeven, omdat strings niet als gelijke van variabelen in rekenkundige bewerkingen kunnen worden beschouwd.

CVI **DISKBASIC**

Zelfde statement als CVD, echter nu wordt een 2-karakter string gedefinieerd. CVI is het omgekeerde van MKI$.

CVS **DISKBASIC**

Zelfde statement als CVD, echter nu wordt een 4-karakter string gedefinieerd. CVS is het omgekeerde van MKS$.

DATA getal,getal, of string, string, etc.

Met het DATA statement kunt u een tabel in het geheugen maken waar de computer gebruik van maakt tijdens het draaien van een programma. Ieder READ statement leest een volgend gegeven uit de tabel. De tabel kan bestaan uit numerieke of alphanumerieke informatie. Indien alphanumerieke informatie in een DATA tabel wordt opgenomen, dan moet deze informatie tussen aanhalingstekens worden geplaatst. In principe mogen zowel numeriek als string DATA door elkaar staan. Belangrijk is echter dat het juiste type READ statement wordt gebruikt, daar anders foutmeldingen zullen ontstaan.
Het DATA statement wordt gebruikt tesamen met de andere BASIC statements READ en eventueel RESTORE. Raadpleeg de beschrijving van deze statements voor verdere informatie.
Voorbeeld:

```
10 DATA "Jan","Piet","Klaas"
20 DATA 1,2,3
30 READ A1$,A2$,A3$,A1,A2,A3
40 PRINT A1,A1$
50 PRINT A2,A2$
60 PRINT A3,A3$
```

geeft:

```
1  Jan
2  Piet
3  Klaas
```

Het is niet noodzakelijk dat de DATA regels vooraan in het programma staan, zij mogen willekeurig voor of na de READ statements worden ingevoerd.

DEFDBL variabele(n)

De variabele achter DEFDBL genoemd zal behandeld worden als een dubbel precisie, tenzij een uitroepteken wordt toegevoegd. Dubbel precisie getallen bestaan uit maximaal 17 cijfers, 16 cijfers worden afgebeeld.
Het statement maakt het mogelijk om bepaalde reeksen van variabelen te benoemen als dubbel precisie. Voorbeeld:

```
10 DEFDBL A-G, U-Z
```

Alle variabelen beginnend met een letter tussen A en G en tussen U en Z worden beschouwd als dubbel precisie.

DEFFN **DISKBASIC**

De afkorting van Definieer Funktie. Hiermee is het mogelijk uw eigen impliciete funkties samen te stellen. Als de funktie gedefinieerd is behoeft u deze slechts aan te roepen d.m.v. FN met erachter de funktienaam. Voorbeeld:

```
50 DEFFNXY(P,Q)= P x Q
60 INPUT "Geef twee getallen '; R, S
70 PRINT "De uitkomst is ';FNXY(R,S)
```

Let op, u ziet dat FNXY gedefinieerd werd met de variabelen P en Q. Bij het gebruik van de funktie echter werden de variabelen R en S toegepast. Iedere geldige variabelen benoeming mag worden toegepast om getallen via de funktie te verwerken.

DEFINT variabele(n)

De achter het statement genoemde variabele zal behandeld worden als een integer getal. Meerdere variabelen kunnen tegelijkertijd benoemd worden. Indien de berekeningen binnen een programma het toelaten integere getallen te gebruiken dan is het toepassen van DEFINT aanbevelenswaardig. Hiervoor zijn twee redenen aan te voeren:
- Het rekenen met integers gaat sneller
- Integers nemen veel minder plaats in het geheugen in.

Let op: een integer kan geen waarde hebben buiten +32767 en -32768.

Voorbeeld:

```
10 DEFINT A
```

Alle variabelen beginnend met A worden beschouwd als integere getallen.

```
20 DEFINT A-K, M-Z
```

Alle variabelen behalve variabele L worden behandeld als integere getallen.

DEFSNG variabele(n)

De via dit statement benoemde variabelen worden beschouwd als enkel precisie, behalve wanneer een uitroepteken werd toegevoegd aan de variabele. Enkel precisie variabelen worden in het geheugen bewaard als getallen met 7 cijfers precisie en worden afgebeeld als getallen met 6 cijfers precisie. Verdere behandeling van variabelen als bij DEFDBL.

DEFUSR **DISKBASIC**

Hiermee geeft u het entry-adres aan voor een subroutine in machinetaal, een zgn. USR-routine. Deze routines kunnen gemaakt worden met bijvoorbeeld de Editor Assembler of via POKE statements in het geheugen worden gebracht vanuit een BASIC programma. In beide gevallen dient u de Memory Size te zetten, zodanig dat het BASIC programma niet de gebruikte geheugenbytes overschrijft.
Voorbeeld:

10 DEFUSR5=H7D20

Tijdens uw BASIC programma kan nu een subroutine worden aangeroepen die zich bevindt op adres 7D20 (Hexadecimaal) ofwel 32032 decimaal.
Zie ook USR.

DEFSTR variabele

Na het uitvoeren van dit statement worden de benoemde variabelen als strings behandeld, tenzij een uitroepteken aan de variabele wordt toegevoegd. Iedere string-variabele kan tot 255 karakters lang zijn, indien voldoende string-ruimte gereserveerd werd d.m.v. CLEAR.
Voorbeeld:

10 A$="ASTER CT-80"

is hetzelfde als:

10 DEFSTR A
20 A="ASTER CT-80"

DELETE regelnummer-regelnummer

Het uitwissen van programmaregels kan in ROM-BASIC door het intijpen van het regelnummer met erachter de ENTER toets. In bijvoorbeeld NEWDOS-80 met DISKBASIC is dit niet mogelijk. Indien u echter meerdere regels van het programma uit het geheugen wilt wissen, dan is het DELETE statement gemakkelijk. U wist hiermee een aantal regels tegelijkertijd uit, die regels die u achter het statement benoemt.
Voorbeeld:

DELETE 10

wist regel 10 uit het geheugen.

DELETE 10-100

wist alle regels vanaf 10 t/m 100 uit het geheugen.

Een zojuist ingetijpte programma-regel kan op de volgende wijze geDELETE worden:

DELETE.

Dus het DELETE statement met een punt (".") erachter.

DIM naam(getal,getal,getal)

Hiermee geeft u de grootte van een ARRAY aan. Indien het DIM statement niet wordt gebruikt neemt de BASIC een ruimte van 11 voor iedere dimensie van ieder ARRAY aan.
Voorbeeld:

10 DIM B(20)

geeft een een-dimensionaal ARRAY van 20 elementen aan.

20 DIM C(30,30,30)

geeft een drie-dimensionaal ARRAY van ieder 30 elementen aan.
Tenzij anders wordt vastgelegd zullen de variabelen in de ARRAYS enkele precisie waarden zijn.

30 DIM D$(15,15)

geeft een twee-dimensionaal string-ARRAY aan met ieder 15 elementen.

Indien een benoemd ARRAY vergroot of verkleind wordt in een programma, dient een CLEAR statement te worden gebruikt, anders ontstaat een foutmelding.

DE EDITOR, INGEBOUWDE KORREKTIEMOGELIJKHEDEN IN DE BASIC

EDIT regelnummer

De ingebouwde EDITOR van de BASIC is bijzonder krachtig. Met eevoudige kommando's kunt u fouten korrigeren, regels veranderen of regelgedeelten wissen. Ook kan met de EDITOR gemakkelijk een bepaald gedeelte uit een te korrigeren regel snel worden opgezocht.

Het geven van het kommando EDIT met erachter het te korrigeren regelnummer schakelt de ASTER CT-80 over naar de EDIT-mode.
Voorbeeld:

EDIT 10

zal ervoor zorgen dat het regelnummer 10 op het scherm verschijnt. Door nu op de spatiebalk te drukken zal letter voor letter de gehele regel op het scherm verschijnen.
Als voorbeeld zullen we een programmaregel nemen. Tijp deze regel op uw ASTER CT-80 en laten we dan eens met de EDITOR gaan experimenteren:

```
10 FOR A=1 TO 20:PRINT"ASTER CT-80":PRINTA:NEXTA
```

Tijp nu EDIT 10. Uw scherm zal het volgende afbeelden:

```
10
```

U kunt nu beginnen met het EDITten (korrigeren) van regel 10. Druk nu 6 x op de spatiebalk, op uw scherm staat nu:

```
10 FOR A=
```

Druk nu op de ENTER toets. Het scherm zal het volgende afbeelden:

```
10 FOR A=1 TO 20:PRINT"ASTER CT-80":PRINTA:NEXTA
```

Het kan ook gemakkelijker. Tijp nu EDIT. (Dus met punt erachter). Het scherm zal weer het regelnummer aangeven. Druk nu twee keer op de cijfertoetsen 1. U tijpt dus het getal 11 in. Druk direkt erna op de spatiebalk en zie het wonder geschiedt, op uw beeldscherm staat de programmaregel tot en met de spatie achter TO afgedrukt. Als we nu het getal 20 van de FOR NEXT herhaling willen wijzigen in bijvoorbeeld 30 dan handelen we als volgt:
Tijp de letter C
Tijp het cijfer 3
Druk op de ENTER toets.

Nu is de regel gekorrigeerd. De herhaling 20 is veranderd in 30. U kunt dit gemakkelijk nu kontroleren. Toets in: LIST 10, en zie wat er gebeurt. Regel 10 zal nu op uw scherm worden afgebeeld. Kijk naar het getal, inderdaad het werd gewijzigd in 30.
Iedere letter kan op deze wijze worden gekorrigeerd. Nogmaals de juiste volgorde:
1 - Ga naar het te korrigeren karakter
2 - Plaats de cursor er juist voor
3 - Druk op de toets C
4 - Tijp het nieuwe karakter

Herhaal handelingen 1 t/m 4 voor ieder karakter in de programma-regel dat u wenst te wijzigen.

In het voorgaande stukje tekst werden al twee mogelijkheden van de EDITOR behandeld: Het x-plaatsen verderspringen door middel van het tijpen van een getal en het aanraken van de spatiebalk en het veranderen van een karakter in een programmaregel.
Evenzo als u x-posities vooruit kunt stappen, kunt u x-posities achteruit stappen.
Voorbeeld, u tijpt:

EDIT 10

Tijp nu 20 gevolgd door het indrukken van de spatiebalk.
Het beeldscherm toont u:

10 FOR A=1 TO 20:PRINT"

U bent in de EDIT-mode, juist voor de letter A. Als u nu het getal 14 tijpt (dus een 1 en een 4) en u drukt dan op de "pijl-naar-links" toets, dan ziet u plotseling de cursor 14 plaatsen naar links schuiven. De tekst verdwijnt tevens voor een groot gedeelte (14 karakters) van uw beeldscherm.
DIT WIL NIET ZEGGEN DAT DE PROGRAMMAREGEL KORTER ZOU ZIJN! Slechts wordt aangegeven op welk punt van de programmaregel u kunt korrigeren.
Het beeldscherm toont u:

10 FOR A=1

Druk nu op ENTER en de regel staat weer in zijn geheel op uw beeldscherm.

Het invoegen en verwijderen

Het kan noodzakelijk zijn dat u uit uw programma regels tekst wil verwijderen. Ook kunt u bij het intijpen karakters vergeten zijn. Met de EDITOR kunt u dit allemaal eenvoudig korrigeren.

Type EDIT 10

Laat nu met de spatiebalk teken voor teken tevoorschijn komen totdat u de onderstaande tekst op uw beeldscherm heeft staan:

10 FOR A=1 TO 2

Druk nu op de lettertoets D. U ziet op uw beeldscherm de nul verschijnen tussen twee uitroeptekens. Dit wil zeggen dat de nul nu uit de regel verwijderd is.
De programmaregel luidt nu:

10 FOR A=1 TO 2:PRINT"ASTER CT-80":PRINTA:NEXTA

Eenvoudig, nietwaar? Uiteraard moet u met hetzelfde gemak waarmee u karakters laat verdwijnen ook karakters kunnen tussenvoegen. U tijpt weer EDIT 10. En u gaat met de spatiebalk indrukkend naar de plaats waar u zo juist de nul heeft gewist. Uw beeldscherm toont:

10 FOR A=1 TO 2
Nu drukt u de letter I in. LET OP alles wat u nu tijpt wordt in de programmaregel ingevoegd direkt rechts van de twee (2). Tijp als voorbeeld een vier (4). Druk nu op de ENTER-toets. Uw beeldscherm moet nu tonen:

10 FOR A=1 TO 24:PRINT"ASTER CT-80":PRINTA:NEXTA

Het (D)eleten van karakters was gemakkelijk, iedere keer op de D drukken laat een karakter verdwijnen. Het (I)nserten van karakters is evenzo gemakkelijk. U moet alleen onthouden dat bij het eenmaal indrukken van de I-toets u in de (I)nsert-mode gaat. Dus alles wat u tijpt wordt in de programmaregel ingevoegd. U kunt hieruit slechts op twee manieren ontsnappen.
- Door op de ENTER-toets te drukken.
- Door de SHIFT-toets tegelijk met de "Pijl-omhoog-toets" in te drukken. Ogenschijnlijk gebeurt er niets, doch u heeft de (I)nsert-mode verlaten en kan nu met de spatie-balk de regel letter voor letter verder bekijken en desnoods corrigeren.

Als u een wijziging heeft aangebracht die u niet bevalt, en u bent niet in de (I)nsert-mode, dan kunt u door het indrukken van de A-toets alle korrekties ongedaan maken en weer opnieuw de korrektieprocedure doen starten.

Als u vindt dat alle korrekties goed zijn uitgevoerd, en u heeft geen behoefte om de regel verder te bekijken dan kunt u de regel ook verlaten door het indrukken van de E-toets. MITS U NIET IN DE (I)NSERT-MODE BEZIG BENT.

Als u een regel gaat EDITen en u heeft de regel niet voor u op het scherm staan, dan kan dit lastig zijn. Zodra u EDIT 10 getijpt heeft, kunt u de regel even LISTen door op de L-toets te drukken voordat u iets anders doet. Uw scherm toont het volgende:

EDIT 10

10

Druk op L (zonder ENTER)

10 FOR A=1 TO 24:PRINT"ASTER CT-80":PRINTA:NEXTA
10

Bijzonder gemakkelijk om eerst de regel nog even te bekijken voordat u gaat EDITen.

Als u aan het einde van een programmaregel nog een stukje wilt toevoegen, dan is er een bijzonder gemakkelijk kommando. Door het indrukken van de X-toets, nadat u EDIT 10 heeft getijpt, wordt de regel direkt op uw scherm afgedrukt, terwijl de cursor zich aan het einde van die regel bevindt. U kunt nu toevoegen wat u wilt. Als u daarna de aktie beeindigt met het indrukken van de ENTER-toets, wordt alles direkt achter de bestaande regel toegevoegd.

Evenzo als u met het vorige kommando tekst achteraan een programmaregel kunt toevoegen, kan het ook nodig zijn om een gedeelte van een regel te laten vervallen. Dit kan met de D-toets. Is het echter een laatste deel van een programmaregel dan is het H-kommando gemakkelijk. U gaat op de normale wijze naar het karakter waar vanaf u het resterende gedeelte wilt laten vervallen. Druk nu op de H-toets, het resterend gedeelte vervalt. U kunt nu eventueel tegelijkertijd een nieuwe toevoeging achter aan deze regel tijpen. Wilt u gewoon niets meer toevoegen dan drukt u op de ENTER-toets en de regel is ingekort zoals u wilde.

Als u tijdens het EDITen in het geheel geen wijzigingen wilt doorvoeren - dit kan zijn omdat u bijvoorbeeld op een verkeerde regel bezig bent, terwijl u dit dan pas merkt - dan kunt u met een kommando de aktie stoppen en alle eventuele wijzigingen ongedaan maken. Druk - mits niet in (I)nsert-mode - op de Q-toets.

Bij het korrigeren van lange programma-regels is het nogal eens lastig om alle karakters door middel van het indrukken van de spatie-balk door te lopen. Ook het opgeven van grotere sprongen behoeft niet altijd het gewenste resultaat te geven.
De EDITOR kent gelukkig ook een zoek-kommando. Dit is erg eenvoudig. Nadat u de te EDITen regel heeft aangeroepen handelt u als volgt: Stel, u wilt in de volgende regel de letter C wijzigen in bijvoorbeeld een S.

10 FOR A=1 TO 24:PRINT"ASTER CT-80":PRINTA:NEXTA

U kunt nu de EDITOR zelf laten zoeken naar de letter C door als volgt te handelen.
- U drukt de S-toets in
- U tijpt de gezochte letter, in dit geval de C

Het beeldscherm toont nu:

10 FOR A=1 TO 24:PRINT"ASTER

U drukt nu de C-toets in, gevolgd door de S-toets en drukt daarna op ENTER. Op uw beeldscherm staat nu:

10 FOR A=1 TO 24:PRINT"ASTER ST-80":PRINTA:NEXTA

In het geval dat u naar een karakter zoekt dat meerdere malen in een regel aanwezig is, dan kunt u een andere methode toepassen. Als voorbeeld dezelfde regel, doch u wilt de A corrigeren uit het statement PRINT A. Dit is de 3e A in die regel. U tijpt dan na het aanroepen van de regel door het EDIT kommando eerst het cijfer 3, direkt gevolgd door de letter S en de letter A.
Uw beeldscherm toont het volgende:

10 FOR A=1 TO 24:PRINT"ASTER CT-80":PRINT

U staat nu op het korrektie punt voor de gewenste letter A.

LET OP, alle enkele letter-toets aanslagen behoeven niet te worden gevolgd door het indrukken van de ENTER-toets. Alleen, daar waar in de handleiding uitdrukkelijk vermeld.

ELSE

Wordt gebruikt in kombinatie met het IF statement. Indien het IF statement niet waar is dan kan men met ELSE toch een andere opdracht uitvoeren zonder dat het programma bij een onwaar beantwoordde IF naar de volgende regel doorvalt.
De toepassing is:

IF .... THEN .... ELSE

Oftewel: ALS .... DAN .... ANDERS
Voorbeeld:

```
10 B=22:INPUT A
20 IF A=B THEN 100 ELSE 10
100 pRINT"A=22"
```

In regel 10 wordt aan B de waarde 10 gegeven. Tevens wordt om de waarde van variabele A gevraagd. Indien een ander getal dan 22 ingegeven wordt zal op regel 20 het programma weer naar regel 10 springen. Is A inderdaad 20 dan springt het programma naar regel 100.

END

Maakt een einde aan uw programma's. Is optioneel als laatste programmaregel het einde van uw programma is. Wordt toegepast indien - zoals gebruikelijk - de laatste regels van uw programma's uit subroutines bestaan. Het beeindigen van uw programma door het runnen van de laatste programmaregels kan dan tot een foutmelding leiden.
Voorbeeld:

```
10 FOR A=1 TO 15
20 PRINT"ASTER CT-80 MICROCOMPUTER"
30 IF A=10 THEN GOSUB 60
40 NEXT A
50 END
60 PRINT"ER IS NU 10 MAAL TEKST AFGEDRUKT"
70 RETURN
```

Het programma eindigt nu bij regel 50 als de FOR NEXT herhaling 15 keer is gedaan. Als er geen END statement in regel 50 zou staan, dan zou het programma doorlopen in de subroutines vanaf regel 60 en het RETURN statement van regel 70 zou een foutmelding geven.

EOF **DISKBASIC**

De einde-file detector. Dit statement kunt u gebruiken bij het inlezen van data vanaf diskette. Om te voorkomen dat u de foutmelding INPUT PAST END krijgt, kunt u tijdens de inleesroutine het EOF statement toepassen en het einde van de file "zien aankomen".
Voorbeeld:

```
70 IF EOF(3) THEN PRINT"GEEN DATA MEER OP DISKETTE":CLOSE 3
```

Regel 70 wordt juist achter het INPUT statement geplaatst waarmee via buffer 3 (zoals achter het

EOF statement genoemd) de data wordt ingelezen. Indien inderdaad het einde van de file wordt gezien dan zal de tekst van regel 70 worden afgedrukt en buffer 3 wordt door het CLOSE statement gesloten.

ERL

Bij het afhandelen van fouten kan het gemakkelijk zijn om fouten van dezelfde soort toch onderscheidenlijk te behandelen, vooral als ze uit verschillende programma regels ontstaan. Het ERL statement geeft u namelijk het nummer van de BASIC regel waarin een fout is ontstaan. Onderstaand een voorbeeld hoe de afhandelingsroutine zou kunnen zijn:

```
10 CLEAR 8
20 ON ERROR GOTO 100
30 INPUT"GEEF UW NAAM";A$
40 PRINT A$:A="
50 INPUT"GEEF UW WOONPLAATS";B$
60 PRINT B$:B="
70 END
100 IF ERL=30 THEN PRINT"Sorry, uw naam niet langer dan 8 tekens":ELSE IF ERL=50 THEN
GOTO 120
110 RESUME 30
120 PRINT"Kort de woonplaats af tot 8 letters"
130 RESUME 50
```

Zoals u ziet wordt in regel 100 bepaald in welke regel de fout is opgetreden. Afhankelijk van het regelnummer wordt nu of de tekst uit regel 100 afgedrukt of gesprongen naar regel 120. In beide gevallen wordt het RESUME statement gebruikt om weer dezelfde regel waarin de fout werd gekonstateerd opnieuw te starten.

Indien een fout niet direkt met een regelnummer te maken heeft dan zal het ERL statement het getal 65535 geven, hetgeen het grootste getal is dat in twee bytes kan worden weergegeven.

ERR

Het ERR statement dient eigenlijk tegelijk met het ERL statement te worden toegepast. Zoals ERL de regel aangeeft waarin de fout ontstond, zo geeft ERR de soort fout aan. Zie voor foutnummers en hun betekenis de tabel FOUTCODES.
Als in het begin van een programma het ON ERROR GOTO statement werd opgenomen dan zal de foutbehandelings routine zowel ERL al ERR kunnen bevatten om de fout zowel als soort en als afkomstig uit een programma deel afzonderlijk te kunnen behandelen.
LET OP: BASIC geeft echter voor het ERR statement een getal op dat tweemaal zo hoog is als de echte foutkode zelf plus een afwijking van 1.
Om de fout juist te kunnen beoordelen dient het foutnummer gedeeld te worden door twee, waarna er het getal 1 bij wordt opgeteld.
Voorbeeld:

```
10 ON ERROR GOTO 100
20 FOR A=1 TO 20:PRINT A:NEXT A 40 RETURN
100 IF ERR/2+1=3 THEN RESUME 20
```

Bovenstaand is een programma dat altijd een foutmelding zal blijven geven. Maar het illustreerT de wijze waarop ERR werkt.

In regel 40 wordt een RETURN statement gebruikt, terwijl nog geen GOSUB werd toegepast. In regel 100 wordt de fout gekontroleerd en het programma zal weer bij regel 20 starten. Zoals u kunt zien zal het programma eindeloos door blijven "draaien" omdat de fout kontrole een programma onderbreking opvangt.
Het gebruik van het ERL en ERR statement is dan ook bedoeld om programma's een stukje "gebruikersvriendelijker" te maken. Indien u zelf programmeert dan is het stoppen van een programma bij een onverwachte fout niet zo erg. U kunt de fout opsporen en het programma eventueel iets wijzigen en weer doorwerken.
Als u echter een programma maakt dat zgn. "Turn-Key" moet zijn, dus voor een gebruiker die zich niet in de programmatuur zal verdiepen, dan zijn deze routines uiterst handig om bijvoorbeeld een melding op het scherm te geven hoe de gebruiker op dat moment moet handelen om te kunnen doorwerken. Of dat hij bijvoorbeeld met u - de maker van het programma - kontakt op moet nemen.

EXP(variabele)

Geeft het natuurlijk exponent van de variabele tussen haakjes. Dit is de tegenovergestelde funktie van LOG.

FIELD **DISKBASIC**

Statement voor het definieren van een Random Acces Buffer. Zie ook het OPEN statement.
In een random file buffer kunnen maximaal 256 bytes worden opgeborgen. Vanuit BASIC kunt u de buffer uitlezen of invullen met nieuwe informatie. De buffer moet worden ingedeeld in FIELDs op een zodanige wijze dat uw BASIC programma daar ook data in kan plaatsen zoals het voor het programma juist is. Iedere buffer kan tijdens het uitvoeren van een programma eventueel opnieuw worden ingedeeld. U moet echter bedenken dat de buffer hierdoor niet wordt gewist, enkel de wijze van verdeling onderling in de buffer van de FIELDS (velden) kan worden gewijzigd. Alle informatie in de buffer moet bestaan uit STRINGs. Voor het konverteren van data van en naar STRINGs zijn enkele kommando's beschikbaar. Deze kommando's zijn:
- MKI$ en CVI
- MKS$ en CVS
- MKD$ en CVD

Raadpleeg de informatie verder in deze handleiding over het gebruik van deze kommando's.
Voorbeelden:

```
10 FIELD 1,256 AS A$
```

De buffer met nummer 1 is nu in zijn geheel ingedeeld als FIELD door de stringvariabele A$.

```
10 FIELD 1,128 AS A$,128 AS B$
```

De buffer met nummer 1 is nu onderverdeeld in 2 fields. De eerste 128 bytes met de naam A$, de tweede 128 bytes met de naam B$.

```
10 FIELD 1,64 AS A$,64 AS B$,64 AS C$,64 AS D$
```

De buffer met nummer 1 is nu onderverdeeld in 4 fields. De eerste 64 bytes met de naam A$, de tweede 64 bytes met de naam B$ enz. enz.

FIELden behoeft niet altijd zo te geschieden dat de lengtes van de FIELDs even groot zijn. Voor een ledenbestands programma zou het FIELD statement bijvoorbeeld zo kunnen worden toegepast:

10 FIELD 2,20 AS NM$, 20 AS AD$, 7 AS PC$, 20 AS WP$, 12 AS VN$, 12 AS PR$, 8 AS CB$

In deze verdeling zijn 99 bytes van de buffer gebruikt. Er blijven er dus nog 155 over. Deze behoeven niet noodzakelijk te worden benoemd. Uiteraard kost dit wel extra ruimte op de diskette maar verder in deze beschrijving zult u zien hoe dat wat eleganter kan worden opgelost.
Om bovenstaande verdeling wat te verduidelijken: de eerste 20 posities zijn benoemd als NAAMstring, de volgende 20 als ADRESstring, de volgende 7 als POSTCODEstring, 20 als WOONPLAATSstring, 12 als VOORNAAMstring, 12 als voornaam PARTNER, 8 als CONTRIBUTIEstring.
Een indeling die u in een programma zou kunnen tegenkomen voor bijvoorbeeld een Tennisclub.

De FIELD namen zoals NM$, AD$, WP$, etc., zijn geen STRINGs in de zin zoals besproken onder STRINGS. Zij nemen in het STRINGgeheugen dan ook geen extra plaats in. Zij geven slechts aan hoe de verdeling van een BUFFER is georganiseerd.

Zoals eerder beloofd zullen we ook ingaan op het beter benutten van de buffer bij kleinere fieldlengtes. Als u een adressenbestand wilt maken en u heeft alleen Naam, Adres, Woonplaats nodig, dan is een maximaal aantal posities van 64 ruim voldoende.
U kunt dan een buffer als volgt indelen:

```
10 FIELD 1, 21 AS NAAM$(1), 18 AS ADRES$(1), 7 AS POST$(1), 18 AS PLAATS$(1)
20 FIELD 1, 64 AS VUL$, 21 AS NAAM$(2), 18 AS ADRES$(2), 7 AS POST$(2), 18 AS PLAATS$(2)
30 FIELD 1, 128 AS VUL$, 21 AS NAAM$(3), 18 AS ADRES$(3), 7 AS POST$(3), 18 AS PLAATS$(3)
40 FIELD 1, 192 AS VUL$, 21 AS NAAM$(4), 18 AS ADRES$(4), 7 AS POST$(4), 18 AS PLAATS$(4)
```

Misschien is het voorbeeld wat complex, bij nadere bestudering valt het best mee. In regel 10 worden de eerste 64 bytes van BUFFER 1 ingedeeld. U ziet nu achter de BUFFER strings een cijfer staan tussen haakjes. Dit slaat op het eerste gedeelte van de buffer. Zoals u ziet zijn de overige 192 bytes van de buffer in het geheel niet benoemd. Dat gebeurt in regel 20. Nu wordt opnieuw de buffer ingedeeld, echter de eerste 64 bytes worden overgeslagen door middel van de benoeming VUL$. De FIELDnamen krijgen ook de extentie (2). Regel 30 wordt op dezelfde wijze opgebouwd. Door middel van VUL$ wordt over de eerste 128, reeds ingedeelde, bytes heengesprongen. Regel 40 geeft de indeling van de laatste 64 bytes van de buffer.
In een RECORD (256 bytes) kunnen nu 4 adressen worden geplaatst.
Zie ook GET, PUT, LSET, RSET, EOF

FIX

Geeft een afbreking van een getal op een integer. Alle cijfers achter de komma worden simpelweg eraf gehaald (dus niet afgerond). Voorbeeld:

```
10 A=2.4
20 B=-2,4
30 PRINT FIX(A)
30 PRINT FIX(B)
```

Geeft het resultaat A=2 en B=2.

FOR TO NEXT - STEP

Als een computer ergens sterk in is, dan is het in het eindeloos herhalen van handelingen. U kunt in een programma iets een aantal malen laten herhalen, kijken hoeveel maal dat is gedaan en dan het doorgaan met het volgende deel van het programma. De FOR NEXT herhaling is een gekontroleerde hoeveelheid van herhalingen. Er is vrijwel geen BASIC programma waar de FOR NEXT niet in voorkomt.
De opbouw is als volgt:

1 - FOR variabele = getal(a) TO getal(b)
2 - behandeling van programmadelen
3 - NEXT variabele

Iedere keer dat het programma het NEXT statement tegenkomt, wordt getal(a) opgeteld bij de variabele en gekontroleerd of de variabele al gelijk is aan getal (b). Zo niet dan worden de programma regels vanaf het FOR statement herhaald.
Voorbeeld:

```
10 FOR A=1 TO 10
20 PRINT A;
30 NEXT A

RUN
1 2 3 4 5 6 7 8 9 10
READY
```

Bovenstaand voorbeeld werkt optellend. Ook kan het statement aftrekkend werken. Voorbeeld:

```
10 FOR A=1 TO 10 STEP-1

RUN
10 9 8 7 6 5 4 3 2 1
READY
```

De FOR NEXT herhaling stopt zodra de waarde bereikt is of overschreden dreigt te worden.
Voorbeeld:

```
10 FOR A=0 TO 2 STEP .6
20 PRINT A;
30 NEXT A

RUN
.6  1.2  1.8
```

Uiteraard kunnen in alle FOR NEXT herhalingen op de plaats van de getallen ook variabelen geplaatst worden die eerder werden benoemd.
Voorbeeld:

```
10 A=1:B=10:C=2
20 FOR X=A TO B STEP C
30 NEXT X
```

FOR NEXT herhalingen kunnen worden "genesteld". D.w.z. meerdere van deze herhalingen kunnen gewoon in elkaar worden opgenomen. Zoals misschien eerder opgemerkt werd zelden de variabele bij het NEXT statement opgenomen. Ook bij genestelde herhalingen is dit niet noodzakelijk. Voor de programmeur is het wel gemakkelijk omdat dan sneller kan worden gekonstateerd in welke herhaling dat men bezig is.
Oplettendheid is echter wel geboden. De juiste variabele moet bij de juiste NEXT worden gezet. Het programma geeft anders een foutmelding.
Voorbeeld van een genestelde herhaling:

```
10 FOR A=1 TO 4
20 PRINT"Buitenste herhaling"
30 FOR B=1 TO 3
40 PRINT"Binnenste herhaling"
50 NEXT B
60 NEXT A
```

In plaats van de regels 50 en 60 kan ook de volgende wijze van programmeren toegepast worden:

```
10 FOR A=1 TO 4
20 PRINT"Buitenste herhaling"
30 FOR B=1 TO 3
40 PRINT"Binnenste herhaling"
50 NEXT B,A
```

Het gebruik van genestelde herhalingen is in principe onbeperkt. Slechts de hoeveelheid geheugen maakt uit hoeveel maal u deze bewerking kunt toepassen.
Zie hiervoor ook bij het overzicht hoeveel bytes gebruikt worden door (genestelde) herhalingen.

FORMAT **DOS**

Met deze mogelijkheid kunt u Data Diskettes maken waarop zoveel mogelijk Data kan en zo weinig mogelijk plaats wordt ingenomen door de zgn. Systeem Informatie.
Op een data-diskette die volgens het TRS-DOS systeem gemaakt is kan voor data 83.75 Kilobyte worden benut. Wij raden echter de gebruiker die volgens het TRS-80 Disk Formaat programma's wil verwerken het gebruik van NEWDOS-80.2 aan. De specifiek andere mogelijkheden van NEWDOS 80.2 staan vermeld in het bijbehorend manual dat zeer uitgebreid is. Het manual is geschreven in de engelse taal.
Enkele voordelen van NEWDOS-80.2 zijn:
- meer gebruikers vriendelijk
- sneller Disk I/O
- grotere capaciteit diskettes (20%)
- alle DOS kommando's via BASIC toegankelijk d.m.v. CMD"..."

Voor de juiste toepassing NEWDOS-80.2 dient de gebruiker echter wel eerst de TRS-80 BASIC te kennen die in de ASTER CT-80 wordt toegepast. NEWDOS-80.2 is namelijk een aanvulling op de bestaande BASIC. Reeds bestaande kommando's worden in NEWDOS-80.2 niet opnieuw besproken.

Maar nu terug naar het FORMAT-programma. U kunt DATA diskettes slechts gebruiken vanaf de tweede drive. Deze drive heeft volgens de DOS nummer 1, omdat het getal nul (0) in computertaal ook een geldig getal is.
Een nieuwe, ongebruikte diskette kan niet zonder meer worden gebruikt. Het FORMAT-proces deelt de diskette in. Deze indeling, in sporen en sektoren, wordt zodanig gedaan dat de DIRECTORY (de inhoud) later precies weet in welke sector de diverse soorten data bewaard wordt. Deze gang van zaken lijkt wat ingewikkeld, doch het staat de gebruiker later toe op een diskette waarop al een aantal files geschreven en gewist zijn de open ruimten zo te benutten dat een file desnoods in tientallen kleine stukjes over de diskette verdeeld wordt. De DOS kan dan via de DIRectory toch de files in zijn originele volgorde en lengte netjes terughalen en in het geheugen plaatsen.
Het FORMAT programma dient niet alleen om de diskette in te delen, ook dient het programma om de diskette te controleren op fouten zodat u later geen problemen krijgt als u data wilt wegschrijven.

Voor het FORMATTEN van een diskette handelt u als volgt.
Als de boodschap DOS READY wordt afgebeeld, dan stop u in de rechter diskdrive van de ASTER CT-80 een nieuwe, ongebruikte diskette. Nu tijpt u FORMAT, gevolgd door ENTER.
Het scherm toont u:

```
DISK FORMATTER UTILITY 2.1

WHICH DRIVE IS TO BE USED ? 1 (ENTER)
DISKETTE NAME ? TESTDATA (ENTER)
CREATION DATE (MM/DD/YY) ? 01/01/83
MASTER PASSWORD ? ASTER
DO YOU WANT TO LOCK OUT ANY TRACKS? N (ENTER)

FORMATTING

READY
```

Bovenstaand een vraag en antwoordspel met het FORMAT-programma. Allereerst vraagt het programma welke drive moet worden gebruikt, daarna de naam van de diskette (maximaal 8 letters), dan de datum (in het maand/dag/jaar formaat), daarna een wachtwoord (password) en de vraag of eventuele tracks (sporen) moeten worden overgeslagen. Dit kan handig zijn als bij een vorige FORMAT poging een spoor van de diskette geweigerd werd bij het formatten. U kunt dan deze diskette toch benutten, hetzij met wat minder data opslag capaciteit.

FRE(string)

Dit statement vertelt u hoeveel STRING geheugen nog beschikbaar is. De tussen haakjes geplaatste string mag een willekeurige zijn. Het FRE kommando zoekt in het geheugen naar de vrije ruimte in het string geheugen. Afhankelijk van de opgebruikte string-geheugen ruimte komt het kommando direkt of iets later terug met de informatie. Indien u het FRE kommando gebruikt met inplaats van string een getal tussen de haakjes dan krijgt u de beschikbare hoeveelheid normale geheugenruimte terug. Het FRE kommando werkt dan als het MEM kommando.

FREE **DOS**

Dit DOS kommando vertelt u de vrije ruimte op de diskettes.

GET **DISKBASIC**

Dit statement leest een record van de diskette. Zie het FIELD statement ook voor verdere beschrijving.
Nadat een file geopend is zal via de RANDOM ACCES methode een willekeurig record vanuit deze file in een buffer kunnen worden geplaatst.
Voorbeeld:

50 GET 1

Deze regel leest record nr. 1 in buffer nr. 1.

60 GET 2,45

Deze regel leest record nr. 45 in buffer nr. 2.

LET OP! Bij een poging een hoger recordnummer in te lezen dan de file bevat, krijgt u geen foutmelding. De buffer wordt dan echter gevuld met nullen. Indien u wilt weten hoelang de file is, dan kunt u het laatste recordnummer te weten komen door middel van de LOF funktie.

GOSUB(regelnummer)

Het GOSUB kommando is een sprong naar een subroutine, meestal op een aantal hogere regelnummers. De sprong wordt na het afhandelen van de subroutine gevolgd door een RETURN, die het programma terugbrengt een regel verder dan waarvanaf de sprong gemaakt werd.

GOTO(regelnummer)

Met het GOTO kommando maakt u een sprong in het programma naar een andere programma regel. Meestal wordt de sprong gemaakt aan de hand van een IF statement, of om eerst een variabele waarde op te halen.

IF

Het IF kommando wordt gebruikt om een voorwaardelijke konditie te beoordelen. Voorbeeld:

```
10 INPUT A
20 IF A=10 THEN 100
30 PRINT"GEEF ALS ANTWOORD HET GETAL 10":GOTO 10
100 PRINT"DE WAARDE VAN A IS 10"
```

In regel 20 wordt de waarde van A vergeleken met 10. Als A 10 is dan wordt de sprong gemaakt naar 100. In alle andere gevallen valt het programma door naar regel 30, waar gemeld wordt dat A tien moet zijn, waarna teruggesprongen wordt naar regel 10.

Zie ook het ELSE statement.

INKEY$

Dit statement geeft een 1-letterige string zodra een toets wordt ingedrukt. Het statement "kijkt" maar een keer naar het toetsenbord en gaat dan verder met de volgende delen van het programma. Dit in tegenstelling tot een INPUT. Hierbij wordt gewacht totdat een enkel- of meervoudige variabele of string wordt ingevoerd, afgesloten door een ENTER.

Het INKEY$ statement is bijzonder krachtig. Doordat het toegepast kan worden tijdens het "draaien" van het programma, kunt u er bijzondere dingen mee doen.

Als geen toets wordt ingedrukt, geeft INKEY$ een nul-string terug. Een voorbeeld om een enkelvoudige string binnen te halen die niet door een ENTER wordt gevolgd ziet u hieronder:

```
10 CLS
20 A$=INKEY$:IFA$=""THEN GOTO 20
30 PRINT A$;
40 GOTO 20
```

Als u dit programma RUNt zult u zien dat het scherm leeg blijft totdat u een toets aanraakt. Het teken dat u indrukte komt nu links bovenaan op het scherm. Iedere volgende letter zal erachter geplaatst worden. Alsof u met een tekstverwerker bezig bent!
Hoe werkt het?
Regel 10 maakt eerst het scherm schoon.
Regel 20 maakt A$ gelijk aan INKEY$. Na de dubbele punt staat het volgende statement op regel 20. Hier wordt gekeken of A$ eventueel een nul-string is. Dit is het geval als er geen toets werd ingedrukt. Als er inderdaad een nul-string gekonstateerd wordt, dan zal het programma terugspringen naar het begin van regel 20. Dit wordt net zo lang herhaald totdat A$ geen nul-string is. Het programma springt dan direkt door naar Regel 30.
Op regel 30 wordt de inhoud van A$ op het scherm afgedrukt. Normalerwijze zou dan direkt erachter een RETURN (regelopschuiving) worden geplaatst door het programma. De punt-komma (;) zorgt er echter voor dat een tweede karakter direkt achter het eerder afgedrukte wordt geplaatst.
Regel 40 zorgt ervoor dat na het doorlopen van het programma weer teruggesprongen wordt naar regel 10.

U ziet, een bijzonder sterk statement. Echter, bovenstaand programma "vergat" de string zo gauw hij werd afgedrukt. Het volgende voorbeeld laat u een 4-karakter lange string intypen zonder dat daarvoor de ENTER toets behoeft te worden ingedrukt.

```
10 CLS
20 A1$=INKEY$:IFA1$=""THEN 20 ELSE PRINT A1$;
30 A2$=INKEY$:IFA2$=""THEN 20 ELSE PRINT A2$;
40 A3$=INKEY$:IFA3$=""THEN 20 ELSE PRINT A3$;
50 A4$=INKEY$:IFA4$=""THEN 20 ELSE PRINT A4$
60 A5$=A1$+A2$+A3$+A4$
70 PRINT A5$
```

Regels 20 t/m 50 zorgen voor het toekennen aan de A(nummer)$. Ook wordt door het einde van iedere regel de string afgedrukt. De punt-komma zorgt ervoor dat geen regelopschuiving wordt gegeven. Behalve bij het afdrukken van de laatste enkele string, A5$.
In regel 60 worden nu alle strings "aan elkaar geknoopt" door ze simpelweg bij elkaar te voegen. Regel 70 tenslotte drukt de komplete string nog eens af.

INP(poortnummer)

De ASTER CT-80 kent een maximum van 256 INPUT-poorten. Dat wil zeggen, de software houdt rekening met het aansluiten in hardware van 256 verschillende 8-bit aansluitingen waarover informatie uit de "buitenwereld" (in het engels is de standaard computer uitdrukking: The Outside World) naar de ASTER CT-80 kan worden toegebracht.

Aktueel zijn niet alle poorten aangesloten. De ASTER CT-80 kent er 5.

Video Controller Poort 1
Het OUTPUT register 252 decimaal (FC HEX), gebruikt als adres register voor de Video Controller.

Video Controller Poort 2
Het INPUT/OUTPUT register 253 decimaal (FD HEX), gebruikt als data register voor de Video Controller.

Systeem Kontrole Poort
Het OUTPUT register 254 decimaal (FE HEX), voor een aantal taken betreffende o.a. de MEMORY MAP instelling, de CLOCK-snelheid, de ROM-selecties en de HALT-mogelijkheid voor de Z80 Microprocessor.

Cassette Interface Poort
Het INPUT/OUTPUT register 255 decimaal (FF HEX), gebruikt voor o.a. de cassette I/O en het schakelen van 32/64 karakters of 40/80 karakters per video-regel.

Centronics Parallel Printer Poort
Het INPUT/OUTPUT register xxx decimaal (XX HEX), gebruikt voor de tweede aanspreek mogelijkheid van de Printer Interface. In de TRS-80 Model I wordt de printer interface via het geheugen gebied aangesproken. In de TRS-80 Model III is de printer interface I/O georienteerd. De ASTER CT-80 Printer Interface kan via beide systemen worden aangesproken. Vandaar dat u de Printer Interface hier ook in het I/O gebied aantreft.
De Printer Interface is zelfs in het I/O gebied door middel van zgn. dipswitches over een gebied van 14 I/O adressen verplaatsbaar.

In tegenstelling tot andere computers heeft de ASTER CT-80 nogal wat INPUT en OUTPUT poorten voor het verrichten van extra funkties. Voor een meer gedetailleerde beschrijving van wat deze poorten voor u kunnen doen wordt u verwezen naar de hoofdstukken over de VIDEO CONTROLLER en over de MEMORY MAP.

WAARSCHUWING: Mocht u zelf apparatuur willen aansluiten via de I/O mogelijkheid van de ASTER CT-80 en u wilt zelf ook de daarvoor benodigde hardware ontwerpen - zie hiervoor het Technisch Manual - laat dan alle poorten vanaf decimaal 128 vrij voor latere fabrieks uitbreidingen van de ASTER CT-80 microcomputer. Een speciale afdeling werkt aan het uitbrengen van diverse interfaces. Zij zullen alle via de bovenste helft van het I/O adresgebied worden aangesproken.

Enkele voorbeelden:

```
10 A=INP(255)
20 PRINT A
```

Regel 10 vraagt het byte van input poort 255. Het adres van deze poort wordt altijd in het decimaal stelsel gegeven. Regel 20 zal het byte dat op die input poort aanwezig is op uw scherm afdrukken, ook in decimale waarde.

INSTR **DISKBASIC**

Een bijzonder krachtig statement. Het zoekt een string die verborgen is in een andere string en geeft tevens op welke karakter positie de substring begint in de hoofdstring.
Indien de substring niet- of niet in zijn geheel aanwezig is in de hoofdstring, dan wordt een nul als startpositie teruggegeven. Indien u verwacht dat een te zoeken string meerdere malen voorkomt in de hoofdstring, dan kan een startpositie worden opgegeven waarvanaf het zoeken in de hoofdstring dient te beginnen.
Voorbeeld:

```
10 A$="ASTER CT-80 MICROCOMPUTER EEN UNIVERSELE COMPUTER"
20 B$="COMPUTER"
30 B=INSTR(1,A$,B$)
40 PRINT B
50 C=INSTR(B+9,A$,B$)
60 PRINT C
```

Regel 10 geeft A$ zijn inhoud. Regel 20 geeft B$ zijn inhoud. In regel 30 wordt door middel van INSTR gezocht naar B$ in A$, beginnend vanaf de eerste positie van A$. Regel 40 geeft de startpositie van B$ in A$. Regel 50 maakt gebruik van de eerste INSTR routine om de tweede B$ in A$ te zoeken. Het zoeken start nu vanaf de 22e positie (B=13, 13+9 = 21). De uitkomst C zal dus 42 zijn.
Regel 50 kan ook anders worden geprogrammeerd:

```
50 C=INSTR(B+LEN(B$),A$,B$).
```

INT(getal of variabele)

Geeft een integer van het getal tussen haakjes.
Voorbeeld:

```
10 A=INT(B x 100+.5)/100
```

In regel 10 krijgt A de waarde van B maar dan afgerond tot twee decimalen.
Bovenstaande formule is bijv. van toepassing bij het maken van administratieve programma's.

KILL **DOS**

Op een niet write-protected (dus zonder plakker) diskette kan met dit kommando een FILE worden gewist. De DOS zoekt naar de eerst voorkomende file met de gegeven naam, tenzij een drive nummer wordt aangegeven.
Voorbeeld:

```
KILL TESTFILE/BAS
```

Indien de file werd voorzien van een wachtwoord (PASSWORD) dan dient ook dit wachtwoord te worden meegegeven. De DOS zal anders het wissen weigeren. Voorbeeld:

```
KILL TESTFILE/BAS.PASSWORD
```

KILL **DISKBASIC**

Eigenlijk is dit kommando een DOS funktie. Echter vanuit BASIC kan deze funktie direkt worden toegepast. Op dezelfde wijze als een FILE wordt geSAVEd kan een FILE worden geKILLed.
Voorbeeld:

```
KILL "TESTFILE/BAS"
```

De filenaam moet tussen aanhalingstekens staan, anders zal een foutmelding worden gegenereerd. Zie ook KILL als DOS kommando.

LEFT$(string,getal)

Geeft het eerste (getal) aantal karakters van de string tussen haakjes. Het getal mag een direkt getal zijn of een variabele. Voorbeeld:

```
10 A$="ASTER CT-80 MICROCOMPUTER"
20 B$=LEFT$(A$,11)
30 PRINT B$
40 C$=LEFT$(A$,LEN(B$)+6)
50 PRINT C$
```

In regel 10 wordt A$ ingevuld. Regel 20 laat B$ de eerste 11 letters uit A$ zijn. Regel 30 drukt de gevonden string af. In regel 40 wordt wat verder gegaan. Hier wordt de lengte van C$ bepaald door de lengte van de eerste string berekend door het LEN statement plus 6 letters.

LET

Optioneel statement, oudere BASICs hebben dit voorvoegsel nodig om een foutmelding te voorkomen. In de BASIC voor de ASTER CT-80 kan het statement worden weggelaten of worden toegevoegd zonder konsekwenties.
Voorbeeld:

```
10 LET A=10
```

is hetzelfde als:

```
10 A=10
```

LSET

Bij het gebruik van RANDOM ACCES files zijn eerst de BUFFERS gedefinieerd d.m.v. het FIELD statement. Ieder field heeft zijn eigen lengte. Als we een adres file als voorbeeld nemen wordt een NAAM-, ADRES-, POSTKODE- en WOONPLAATSstring in het field gedefinieerd. Niet iedere naam of adres is van gelijke lengte. We moeten dus de computer laten weten hoe de informatie in het FIELD geplaatst moet worden.
Stel de lengte van het NAAMfield is 20 letters. De in te voeren naam is J.Jansen. Een lengte van 8 karakters. De toepassing van het kommando LSET nu is het plaatsen van deze 8 karakters in het NAAMfield. Bij het kommando LSET worden de karakters vanaf links in het FIELD geplaatst, terwijl rechts met spaties wordt opgevuld.
Voorbeeld:

```
10 NAAM$="J.Jansen"
20 LSET NAAM$
```

of:

```
10 LSET NAAM$="J.Jansen"
```

Bovenstaande programma regels zorgen ervoor dat de data er in de buffer als volgt uitziet:

J.Jansen␣␣␣␣␣␣␣␣␣␣␣␣

waarbij de ␣ een spatie voorstelt.

RSET

Bij het gebruik van RANDOM ACCES files zijn eerst de BUFFERS gedefinieerd d.m.v. het FIELD statement. Ieder field heeft zijn eigen lengte. Als we een adres file als voorbeeld nemen wordt een NAAM-, ADRES-, POSTKODE- en WOONPLAATSstring in het field gedefinieerd. Niet iedere naam of adres is van gelijke lengte. We moeten dus de computer laten weten hoe de informatie in het FIELD geplaatst moet worden.
Stel de lengte van het NAAMfield is 20 letters. De in te voeren naam is J.Jansen. Een lengte van 8 karakters. De toepassing van het kommando RSET nu is het plaatsen van deze 8 karakters in het NAAMfield. Bij het kommando RSET worden de karakters vanaf rechts in het FIELD geplaatst, terwijl links met spaties wordt opgevuld.
Voorbeeld:

```
10 NAAM$="J.Jansen"
20 RSET NAAM$
```

of:

```
10 RSET NAAM$="J.Jansen"
```

Bovenstaande programma regels zorgen ervoor dat de data er in de buffer als volgt uitziet:

␣␣␣␣␣␣␣␣␣␣␣␣J.Jansen

waarbij de ␣ een spatie voorstelt.

LSET en RSET

Zoals u kunt lezen uit de bespreking zijn beide kommando's exact aan elkaar gelijk met dien verstande dat LSET de data aan de linkerzijde in het field plaatst en rechts spaties bijvoegt, terwijl RSET het tegenovergestelde doet.

LEN(string)

Dit kommando geeft de lengte van een string weer. De string moet tussen haakjes achter het kommando staan.
Voorbeeld:

```
10 A$="ASTER CT-80 MICROCOMPUTER"
20 A=LEN(A$) 30 PRINT A
```

Het resultaat van bovenstaand programma is 25.

LINE

Een (nog) niet gebruikt BASIC kommando is de ASTER CT-80.

LIST

Het LIST kommando wordt veel gebruikt tijdens het programmeren in BASIC. Het eenvoudigweg intoetsen van LIST laat alle regels van het BASIC programma zien.
Ook het selektief LISTen van programma delen is mogelijk. Voorbeeld:

LIST 100

Alleen regel 100 wordt op het beeldscherm getoond.

LIST 10-100

Alleen de regels 10 tot 100 worden op het beeldscherm getoond.

LIST 200-

Alle regels vanaf programma regel 200 worden op het beeldscherm getoond.

LIST -200

Alle regels tot regel 100 worden op het beeldscherm getoond.

LIST .

De regel die men zojuist heeft geLIST of geEDIT wordt op het beeldscherm getoond.

Het listen kan worden onderbroken door de BREAK-toets in te drukken.

Het listen kan tijdelijk worden gestopt door de SHIFT en de ALPHA-toets @ tegelijkertijd in te drukken. Het wederom indrukken van deze toets-kombinatie laat het listen vervolgen.

LLIST

Het LLIST kommando wordt veel gebruikt tijdens het programmeren in BASIC. Het eenvoudigweg intoetsen van LLIST laat alle regels van het BASIC programma op de printer afdrukken. Ook het selektief LLISTen van programma delen is mogelijk. Voorbeeld:

LLIST 100

Alleen regel 100 wordt op de printer afgedrukt.

LLIST 10-100

Alleen de regels 10 tot 100 worden op de printer afgedrukt.

LLIST 200-

Alle regels vanaf programma regel 200 worden op de printer afgedrukt.

LLIST -200

Alle regels tot regel 100 worden op de printer afgedrukt.

LLIST .

De regel die men zojuist heeft ge(L)LIST of geEDIT wordt op de printer afgedrukt.

Het llisten kan worden onderbroken door de BREAK-toets in te drukken.

Het llisten kan tijdelijk worden gestopt door de SHIFT en de ALPHA-toets (@) tegelijkertijd in te drukken. Het wederom indrukken van deze toets-kombinatie laat het llisten vervolgen.

CASSETTE LOAD en SAVE handelingen

CSAVE

Met dit kommando kunt u een BASIC programma vanuit het geheugen naar een magneetband cassette wegschrijven. Het wegschrijven moet eigenlijk kopieren heten, want het programma zelf verdwijnt niet uit het geheugen. Het wordt slechts naar een "device" gekopieerd.
U kunt het programma een naam geven van maximaal 1 karakter. Het karakter mag zowel numeriek als alphanumeriek zijn.
Het is een goede gewoonte op de magneetband cassette aan te tekenen welke programma's erop weggeschreven zijn om de programma's later gemakkelijk te kunnen terugvinden.
De programma naam dient ook om later het programma gemakkelijker te kunnen inlezen met CLOAD. U kunt dan de computer zelf het programma op de cassette laten terugvinden.
Voorbeeld:

CSAVE"T"

CSAVE"5"

In het eerste geval krijgt het programma de naam T. In het tweede geval de naam 1.

Als u twee cassetterecorders heeft aangesloten dan kunt u met het CSAVE kommando tegelijkertijd kiezen op welke recorder dat u het programma wegschrijft. Om op recorder 1 te schrijven geeft u het volgende kommando:

CSAVE #-2,"T"

is gelijk aan het eerste voorbeeld maar nu specifiek naar de tweede cassetterecorder. Onnodig te zeggen dat de desbetreffende cassetterecorder in de opname-stand geschakeld dient te zijn.

Als u de juiste aansluitsnoeren gebruikt (niet standaard meegeleverd, evenmin als de recorder(s)), dan zal de ASTER CT-80 ook de in- en uitschakeling van de motoren van de beide cassette recorders verzorgen.

Tijdens het bijwerken van FILES is het gebruikelijk om een cassetterecorder in de weergavestand en de andere in de opnamestand te schakelen. De informatie wordt dan van de eerste cassetterecorder gelezen, bewerkt, en dan op de tweede cassetterecorder weggeschreven.
Zie voor uitgebreidere beschrijving van cassette FILE behandeling het PRINT #-(getal)- en INPUT #-(getal)-statement.

CLOAD

Het CLOAD kommando laat u BASIC programma's van cassette inlezen. Net zoals bij het CSAVE kommando tijpt u ook het CLOAD kommando als volgt in:

CLOAD"T"

Het programma met de naam T wordt van cassette ingelezen.
Tijdens het inlezen wordt in de rechter bovenhoek van het scherm een knipperende ster zichtbaar, tesamen met de files die niet beantwoorden aan de toevoeging die u het CLOAD kommando gaf. Deze mogelijkheid is ook toepasbaar indien u niet meer weet welke files op een cassette staan. U

CLOADt gewoon met een file-naam waarvan u zeker weet dat die niet op de cassette weggeschreven werden. Tijdens deze aktie kunt u de file-namen noteren als ze in de rechterbovenhoek verschijnen.

Als u de programma naam weglaat, dus als volgt:

CLOAD

dan zal de computer de eerste file die hij tegenkomt in het geheugen laden.

CLOAD?

Als u er zeker van wilt zijn dat het programma goed van de cassetterecorder werd geladen, dan is er een mogelijkheid om dit te controleren. Met het CLOAD? programma kontroleert u een in het geheugen ingelezen programma tegenover het origineel op de cassette.
De toepassing ervan is het sterkst als controlemiddel op het goed wegschrijven van programma's NAAR de cassetterecorder.

Als er verschil is tussen de informatie op de tape en de informatie in het geheugen, dan zal de boodschap BAD op het scherm worden afgebeeld.
Een waarschuwing is op zijn plaats, let goed op het vermelden van het vraagteken achter het CLOAD kommando. Als u het vergeet, dan wordt het programma in het geheugen gewist en het programma van de cassette ervoor in de plaats gezet. Mocht het laden eerder fout zijn gegaan dan heeft u geen foutloos programma meer over.

LOC
Gereserveerd BASIC kommando. Nog geen funktie.

LOF **DISKBASIC**

Bij het gebruik van sequentiele en random files kunt u door middel van het LOF kommando te weten komen wat het hoogste record nummer van de file is.
Tijdens random-acces bij files die reeds eerder bestonden dient het programma te weten wanneer het laatste record gelezen wordt. Met een FOR NEXT herhaling waarbij LOF gebruikt wordt als hoogste FOR NEXT herhaling kunt u veilig record-handeling toepassen.
LOF wordt toegepast bij het inlezen, maar ook bij het toevoegen van informatie aan een bestaande file.
Voorbeeld:

```
100 DEFINT A
110 OPEN "R",2,"ADRFILE"
120 FIELD 2,20 AS NAAM$, 18 AS ADRES$, 218 AS REST$
130 FOR A=1 TO LOF(2)
140 GET 2,A
150 PRINT NAAM$
160 PRINT ADRES$
170 PRINT REST$
180 NEXT A
190 FOR B=1 TO 10:INPUT NAAM$
200 REM verdere programmaregels zorgen voor
    het plaatsen in de buffer van 10 nieuwe, op disk
    weg te schrijven, adres-informaties.
210 ...
220 ...
230 ...
240 PUT 2,LOF(2)+1
250 NEXT B
```

In regel 100 wordt A als integer benoemd. De file met de naam ADRFILE wordt geopend in regel 110. Bij regel 120 wordt de buffer in fields ingedeeld. In regel 130 wordt het aantal mogelijke records in de file bepaald door het LOF statement. Als LOF dus 23 is, dan staat er eigenlijk: FOR A=1 TO 23.
Regel 140 haalt het record van de diskette en plaatst het in buffer nummer 2.
Regels 150 tot 170 plaatsen de informatie op het beeldscherm.
Regel 180 zorgt ervoor dat het programma zoveel maal doorlopen wordt als LOF heeft aangegeven.

LOG(getal of variabele)

Geeft het natuurlijk logarithme van het getal tussen haakjes. Het LOG statement is het omgekeerde van het EXP statement.
Om het logarithme van een getal tegenover een ander getal te vinden, kunt u de volgende formule toepassen:

```
10 LOG B(A)=LOGC(A)/LOGC(A)
```

Een ander voorbeeld:

```
20 B=LOG(32767)/LOG(2)
30 PRINT B
```

MEM

In sommige gevallen kan het tijdens het uitvoeren van een programma nodig zijn om te weten of de geheugen grootte nog toereikend is voor het opzetten van bijvoorbeeld arrays.
Ook is het handig om na het maken van een programma te weten hoe lang het programma is, door als direkt statement te tijpen:

```
RINTMEM
```

De ASTER CT-80 zal dan in het decimaal stelsel de beschikbare hoeveelheid geheugenruimte op het beeldscherm afdrukken.

Als u de ASTER CT-80 opstart en u vraagt direkt de beschikbare hoeveelheid geheugen dan krijgt u bij een NON-Disk systeem de melding:

```
48338
```

Voor de gebruiker is dus 48Kbyte geheugen beschikbaar.
De snelle rekenaar zal misschien zeggen dat dit getal niet exact 48Kbyte vertegenwoordigt. Inderdaad, de computer zelf neemt van het geheugen een aantal bytes (814) af voor eigen gebruik. Hierin wordt de STACK, DCBDRIVER-adressen en diverse andere interne informatie geplaatst.

Afhankelijk van welk soort DISK-operating systeem gebruikt wordt en hoe de MEM-SIZE werd gezet zal een gebruiker met Diskdrives bij PRINTMEM andere meldingen krijgen.

Hoewel de ASTER CT-80 IN DE TRS80-mode 48Kbyte over 48Kbyte aan geheugen kan beschikken kan toch de overige 16Kbyte RAM worden benut. De wat meer in MACHINETAAL thuis zijnde programmeur kan door middel van speciale instrukties bijvoorbeeld subroutines tijdelijk in dit gedeelte opslaan. Aanwijzingen hoe dit te doen treft u aan in het hoofdstuk MEMORY MAP.

MERGE **DISKBASIC**

De programmeur die veel programma's maakt, heeft meestal programmadelen die veelvuldig in verschillende programma's worden toegepast. Het is een goede zaak, om veel toegepaste subroutines altijd op dezelfde regelnummers te plaatsen. Het voordeel is onder meer, dat de programma's overzichtelijker worden. U behoeft bij het zoeken naar een eventuele fout in dat ene programma niet te zoeken waar de gebruikte sub-routine bepaalde zaken verricht als u steeds de subroutine op dezelfde programma-regels laat staan.
Als u die verschillende Sub-routines op diskette wegschrijft en daar een stukje administratie over bijhoudt, dan kunt u die diskette als een soort programmeer-hulp gebruiken.
Heeft u een bepaalde subroutine nodig, dan kunt u met het MERGE kommando zeer simpel een stukje programma "bij-laden" bij het reeds in het geheugen staande programma. Voorwaarde is wel dat het programma met de ASCII-optie naar disk weggeschreven werd.
Belangrijk is tevoren te weten of in de te MERGEn file niet dezelfde regelnummers voorkomen als het programma dat in het geheugen staat. De regelnummers van de diskette prevaleren namelijk boven die in het geheugen.
Komen dus zowel in het geheugen als in de te MERGEn diskette-file de regels 100, 110 en 120 voor, dan zullen de regels uit het geheugen worden gewist en die van de diskette ervoor in de plaats komen.

Het MERGE kommando is een der krachtigste programmeerhulpen die voor de ASTER CT-80 gebruiker beschikbaar zijn.

OPMERKING: Het toepassen van MERGE sluit d.m.v. CLOSE alle openstaande files en CLEARed alle variabelen.

MID$ **DISKBASIC**

Het manipuleren van strings is een zeer krachtige eigenschap van de BASIC in de ASTER CT-80. Met MID$ kunt u een gedeelte uit een andere string vervangen door een nieuw gedeelte.
Zie voor een juist begrip ook de andere string-behandeling kommando's. Het kommando wordt als volgt ingegeven:

MID$(A$,4,2)

Waarbij A$ de te behandelen string is, het getal 4 het start karakter aangeeft in de te behandelen string, terwijl de 2 de lengte aangeeft van het aantal karakters dat van de in te brengen string in de te behandelen string ingebracht wordt.
Voorbeeld:

```
10 A$="MICROCOMPUTER "
20 B$="ASTER CT-80 TEKSTVERWERKING"
30 C$=MID$(B$,13,15)
40 PRINT C$
```

In regel 10 wordt A$ ingevuld.
Regel 20 bevat de te behandelen string.
Het MID$ kommando wordt in regel 30 toegepast. Het resultaat wordt door regel 40 afgedrukt. De letters "TEKSTVERWERKING" worden vervangen door het woord "MICROCOMPUTER"

MKD$ **DISKBASIC**

Dit kommando verandert een getal in een string. De lengte van de string is 8-bytes. Er wordt vanuitgegaan dat het getal een dubbel-precisie getal is.

MKS$ **DISKBASIC**

Dit kommando verandert een getal in een string. De lengte van de string is 4-bytes. Er wordt vanuitgegaan dat het getal een enkel-precisie getal is.

MKI$ **DISKBASIC**

Dit kommando verandert een getal in een string. De lengte van de string is 2-bytes. Er wordt vanuitgegaan dat het getal een integer getal is.

NAME

Dit kommando is gereserveerd voor toekomstige toepassing.

NEW

Dit kommando wist het gehele programma uit het geheugen. Toepasbaar als u een nieuw programma wilt intijpen en het oude zou willen wissen.
Ook toepasbaar om door middel van een wachtwoord uw programma te beschermen. Als een programma in het geheugen "draait" en u wilt alleen dat bevoegde gebruikers het programma kunnen benutten, dan kunt u de volgende programma regels toevoegen:

```
10 INPUT X$:IF X$="WACHTWOORD"THEN 20 ELSE 65000
20 uw programma
30 uw programma
40 etc.
65000 NEW
```

Regel 10 vraagt om een wachtwoord. Is de vergelijking niet juist dan wordt naar regelnummer 65000 gesprongen. Op deze regel staat het NEW statement, dat direkt het gehele programma uitwist.
Uiteraard is bovenstaand voorbeeld zeer eenvoudig en doorzichtig. De ervaren programmeur kan echter met wat meer moeite de opslag van het wachtwoord en het uitvoeren van het NEW statement in het programma zo moeilijk naspeurbaar maken dat het erg moeilijk wordt om deze funktie uit te schakelen.

NEXT

Het NEXT statement is reeds besproken bij het FOR statement. Zie de inhoudsopgave.

NOT

In de BOLEAANSE algebra wordt het NOT statement toegepast. Het is een negativering van een positief, ofwel een positivering van een negatief.
Misschien wat moeilijk uitgedrukt, maar het resultaat is simpel. Als het statement op bits wordt toegepast dan is het gevolg:

Indien het argument 0 is, is het resultaat 1
Indien het argument 1 is, is het resultaat 0

Bij het toepassen van het NOT statement wordt het getal waarmee wordt gewerkt omgezet in een 16-bits twee-complements getal in het gebied van -32768 tot +32767. Daarna wordt de de boleaanse algebra toegepast. Het resultaat wordt zonodig afgerond zodat het in hetzelfde gebied blijft (tussen -32768 en +32767). Mocht het getal niet binnen deze grenzen blijven, dan resulteert een FC foutmelding.

ON variabele GOTO regel,regel,regel,etc.

Een mogelijkheid om op de waarde van een variabele, voorwaardelijk te springen naar subroutines of delen van een programma.
Het getal achter ON mag maximaal de waarde 255 hebben. In principe is het aantal regels achter GOTO invulbaar tot maximaal 255 regelnummers. De inputbuffer van BASIC laat echter niet meer dan 255 karakters toe. Indien dus een zeer uitgebreid ON .. GOTO kommando gebruikt moet worden, dan dient dit in meerdere programmaregels te worden opgevangen.
Voorbeeld:

```
10 INPUT A
20 ON A GOTO 40,50,60,70
30 GOTO 10
40 PRINT"Het ingegeven cijfer is 1":GOTO10
50 PRINT"Het ingegeven cijfer is 2":GOTO10
60 PRINT"Het ingegeven cijfer is 3":GOTO10
70 PRINT"Het ingegeven cijfer is 4":GOTO10
80 END
```

Er mag geen hoger getal worden ingevoerd dan 4. In regel 20 wordt afhankelijk van de variabele A gesprongen naar de regelnummers 40 tot 70.

Als er het aantal regelnummers teveel is om in een programma regel te worden opgenomen dan moeten de ON GOT statements verdeeld worden over verschillende regels. Voorbeeld:

```
10 INPUT A
20 ON A GOTO 60,70,80,90,100,110,120,130,140,150
30 ON A+10 GOTO 160,170,180,190,200,210,220,230,240,250
40 ON A+20 GOTO 260,270,280,290,300,310,320,330,340,350
50 GOTO 10
60 PRINT"Het ingegeven cijfer is 1":GOTO10
70 PRINT"Het ingegeven cijfer is 2":GOTO10
80 PRINT"Het ingegeven cijfer is 3":GOTO10
90 PRINT"Het ingegeven cijfer is 4":GOTO10
enzovoorts
350 PRINT"Het ingegeven cijfer is 30":GOTO10
360 GOTO 10
```

ON variabele GOSUB regel,regel,regel, ect.

Een mogelijkheid om op de waarde van een variabele, voorwaardelijk te springen naar subroutines of delen van een programma.
Het getal achter ON mag maximaal de waarde 255 hebben. In principe is het aantal regels achter GOSUB invulbaar tot maximaal 255 regelnummers. De inputbuffer van BASIC laat echter niet meer dan 255 karakters toe. Indien dus een zeer uitgebreid ON .. GOSUB kommando gebruikt moet worden, dan dient dit in meerdere programmaregels te worden opgevangen.
Voorbeeld:

```
10 INPUT A
20 ON A GOSUB 40,50,60,70
30 END
40 PRINT"Het ingegeven cijfer is 1":RETURN
50 PRINT"Het ingegeven cijfer is 2":RETURN
60 PRINT"Het ingegeven cijfer is 3":RETURN
70 PRINT"Het ingegeven cijfer is 4":RETURN
80 END
```

Er mag geen hoger getal worden ingevoerd dan 4. In regel 20 wordt afhankelijk van de variabele A gesprongen naar de regelnummers 40 tot 70.

Als er het aantal regelnummers teveel is om in een programma regel te worden opgenomen dat moeten de ON GOSUB statements verdeeld worden over verschillende regels. Voorbeeld:

```
10 INPUT A
20 ON A GOSUB 60,70,80,90,100,110,120,130,140,150
30 ON A+10 GOSUB 160,170,180,190,200,210,220,230,240,250
40 ON A+20 GOSUB 260,270,280,290,300,310,320,330,340,350
50 END
60 PRINT"Het ingegeven cijfer is 1":RETURN
70 PRINT"Het ingegeven cijfer is 2":RETURN
80 PRINT"Het ingegeven cijfer is 3":RETURN
90 PRINT"Het ingegeven cijfer is 4":RETURN
enzovoorts
350 PRINT"Het ingegeven cijfer is 30":RETURN
360 END
```

OPEN **DISKBASIC**

Teneinde het in- en uitvoeren van data naar en van diskette goed te doen verlopen gebruikt de BASIC een BUFFER.
Er zijn twee typen van files te gebruiken. De RANDOM file en de SEQUENTIAL file. Het eerste type laat toe dat data willekeurig uit de file wordt gelezen. Het tweede type file kan alleen worden gelezen vanaf zijn begin. Heeft men bij een SEQUENTIAL file dus data uit het midden ervan nodig dan dient toch alle daarvoor staande data te worden doorgelezen.

Het benoemen van een BUFFER doet men met het OPEN statement.
Voorbeeld:

```
10 OPEN "I",1,"TESTFILE:0"
```

Regel 10 opent een SEQUENTIAL file buffer voor het inlezen van data vanaf diskette. De file staat op drive 0. Mocht de file niet bestaan, dan wordt een foutmelding gegeven.

```
20 OPEN "O",1,"TESTFILE:0"
```

Regel 20 opent een SEQUENTIAL file buffer voor het wegschrijven van data naar een diskettefile. De file staat op drive 0. Mocht de file niet bestaan, dan wordt deze ook gelijktijdig aangemaakt. Mocht de file wel reeds bestaan, dan wordt over de bestaande data heengeschreven. De bestaande file is dus vervangen door de nieuwe file met dezelfde naam. Dit houdt dus automatisch in, dat aan een sequentiele file geen informatie kan worden toegevoegd. Indien dit noodzakelijk is, dan is de te volgen methodiek:

- open de file
- lees de inhoud bijv. in een array in het geheugen
- voeg nieuwe informatie toe aan het array
- schrijf de nieuwe informatie als gehele file naar disk.

LET OP: Laat een file nooit langer OPEN dan nodig is. Mocht tijdens het OPEN staan van een file iets gebeuren waardoor u zou moeten resetten, of dat het programma opnieuw moet worden opgestart, dan is de file niet meer toegankelijk.
Het is verstandig om een file na toegang te hebben gehad zo snel mogelijk te CLOSEn.
De "gulden regel": open de file als u hem nodig heeft, sluit hem direkt erna.

Voorbeeld van het openen van een RANDOM ACCES file:

```
10 OPEN "R",2,"ADRFILE"
```

In regel 10 wordt een RANDOM ACCES file BUFFER geopend voor zowel invoer als outvoer.
Mocht een file met deze naam nog niet bestaan, dan wordt ze alsnog gecreeerd.
Bij het schrijven naar-, of lezen uit een RANDOM ACCES file dient het record nummer te worden opgegeven. Zie de kommando's LOF en EOF voor een beschrijving hoe de hoeveelheid records van een file op te vragen.

OR

Een statement uit de Boleaanse Algebra. In de logika een eenvoudige voorwaarde. De waarheidstabel behorende bij het OR statement.
Gewerkt wordt met twee grootheden. Bijvoorbeeld A en B. Het resultaat is C:

A	B	C
1	1	1
1	0	1
0	1	1
0	0	0

Hieruit volgt dus dat C alleen nul is als A en B nul zijn.
Voorbeeld:

```
10 A=4:B=2
20 C=B OR A
30 PRINT C
```

Het resultaat is 6. A wordt eerst omgezet naar een binair getal, 100. B wordt ook omgezet naar een binair getal, 10.
Als nu deze twee binaire getallen ge-ORred worden ontstaat het binair getal 110 oftewel 6 decimaal.

OUT poortnummer,waarde

Met de OUT-instruktie schrijft u een bepaalde waarde naar een der 256 OUTPUT poorten. De notatie is als volgt:

```
10 OUT 253,2
```

In regel 10 wordt op poort 252 (het adres-register van de Video Controller) het byte decimaal 10 geschreven. De video-controller zal nu ingesteld zijn om via poort 253 de data te ontvangen die in adres-register 10 van de Video-Controller moet worden geplaatst.

Zie ook voor uitgebreidere beschrijving van de mogelijkheden met het OUT kommando de hoofdstukken VIDEO CONTROLLER en MEMORY MAP.

PEEK(adres)

Gebruik bij het hanteren van dit statement het overzicht van de geheugen indeling van de ASTER CT-80, de Memory Map.
Met het PEEK statement "kijkt" u namelijk in het geheugen. U past het statement als volgt toe:

```
10 A=PEEK(14312)
20 PRINT A
```

In regel 10 wordt gekeken op geheugenplaats 14312. Het resultaat van dit onderzoek wordt afgedrukt op uw beeldscherm door regel 20.
Om te PEEKen boven adres 32767 moet de volgende formule toegepast worden:

-1 maal (65536 min adres) is het juiste PEEK adres.

POINT(a,b)

Met dit statement is het mogelijk om het programma te laten konstateren of een bepaald grafisch blok op het videoscherm "aan" of "uit" staat.
Als het blokje aan staat dan wordt de waarde -1 geretourneerd. Als het blokje uit staat dan wordt de waarde 0 geretourneerd.
Achter het POINT statement wordt tussen haakjes het regelnummer en de karakterpositie weergegeven.

Zie ook de statements SET en RESET, waar het aan- en uitschakelen van de grafische blokjes wordt besproken.

POKE

Direkte mogelijkheid tot het plaatsen van informatie op een specifiek adres.
Gebruikte formaat:

POKE adres,waarde

Het POKE statement kan gebruikt worden om bepaalde informatie direkt in het geheugen te bewaren, bijvoorbeeld in zgn. HIGH MEMORY. Bij het toepassen van verschillende BASIC programma's die bepaalde informatie nodig hebben kan men die informatie boven de grens van het zgn. beschermd geheugen inPOKEn. Het RUNnen van een ander programmadeel heeft geen invloed op dit bechermd stuk geheugen. (Zie in dit verband ook VARPTR)
Ook wordt het POKE statement gebruikt om in HIGH MEM machinetaal routines te plaatsen vanuit het BASIC programma. De eigenlijk uit HEX codes bestaande routine wordt in decimalen in het geheugen gePOKEd en later via een USR aangeroepen.
Om te POKEn boven adres 32767 moet de volgende formule toegepast worden:

-1 maal (65536 min adres) is het juiste POKE adres.

Ook kan het gebruik van POKE bij het videoscherm uw programma's er professioneel laten uitzien.
Het videoscherm van de ASTER CT-80 is namelijk zgn. MEMORY MAPPED. Dat wil zeggen dat het videoscherm eigenlijk bestaat uit een aantal geheugenlokaties die via een beeldscherm zichtbaar worden.
Hetgeen in dat geheugengebied staat, ziet u ook aktueel op uw beeldscherm.

Om het videoscherm snel te laten "wit-verven" kan men dit dan ook met POKE statements via een kort programma realiseren.
Voorbeeld:

```
10 CLS
20 B=191
30 FOR A= 15360 TO 15360+1023
40 POKE A,B
50 NEXT A
60 GOTO 60
```

Daar het videoscherm 1023 bytes "lang" is (het laatste byte is 16383) wordt de FOR NEXT herhaling 1023 keer gedaan, daarna blijft het programma eindeloos "hangen" in regel 60 om het beeldscherm wit te laten, zonder dat de READY melding afgebeeld wordt.
BREAK het programma eens en verander in regel 20 de waarde van B in 65. Of in 42.

Ziet u dat het videogeheugen zonder meer gevuld kan worden met ieder soort waarde en dat dan ook het bijbehorend teken uit de ASCII tabel wordt afgedrukt?

Om het merendeel der tekens uit de karakter generator te zien kunt u het volgende programma eens proberen:

```
10 CLS
20 FOR A=15360 TO 15360+1023
30 B=B+1
40 POKE A,B
50 IF B=255 THEN GOTO 70
60 NEXT A
70 GOTO 70
```

ERNSTIGE WAARSCHUWING!! *POKEn zonder dat u precies weet waar u dit doet, kan desastreuze gevolgen hebben voor informatie in het geheugen of informatie die op de diskette staat. U kunt per ongeluk op een adres een bepaalde waarde zetten die bijvoorbeeld een ongewilde schrijfaktie op de diskette veroorzaakt. Let dus goed op.*

POS(willekeurige getal)

Om te weten op welke positie (van vorenaf gerekend) dat de cursor op een video regel staat, kunt u het POS statement gebruiken.
Voorbeeld:

```
10 PRINT"ASTER CT-80";
20 PRINT TAB(POS(0)+1)"MICROCOMPUTER SYSTEEM"
```

Bovenstaand programma zal op uw beeldbuis de volgende regel plaatsen:

ASTER CT-80 MICROCOMPUTER SYSTEEM

In regel 10 worden de woorden ASTER CT-80 op het scherm geplaatst. Door de punt-komma erachter wordt geen regel-opschuiving gegenereerd. In regel 20 wordt via het TAB statement en het POS statement de woorden "MICROCOMPUTER SYSTEEM" geplaatst. POS geeft de cursor-positie na het weergeven van de eerste woorden. Er wordt 1 positie bij opgeteld, dit wordt ingegeven aan het TAB statement.

De waarde 0, geplaatst tussen haakjes achter het POS statement is een zgn. DUMMY ARGUMENT en mag een willekeurig cijfer zijn.

POSN

Gereserveerd BASIC statement voor latere toepassing.

PRINT, EEN UITGEBREID STATEMENT

De meeste computergebruikers zijn er inmiddels aan gewend, printen is niet alleen het afdrukken op een regeldrukker oftewel line-printer maar het PRINT statement wordt ook gebruikt om informatie op het beeldscherm te "printen".

Tevens wordt het PRINT statement toegepast in het wegschrijven van informatie naar een cassette recorder of naar een Disk file.
In onderstaande informatie zal het PRINT statement in al zijn vormen worden behandeld.

PRINT - Videoscherm toepassingen

Deze vorm van het statement PRINT wordt toegepast met de volgende toevoegingen:
- PRINT
- PRINT@
- PRINT TAB
- PRINT USING

Het toepassen van het PRINT statement zonder meer gaat als volgt:

```
10 PRINT "ASTER CT-80"
```

Direkt na de cursor positie zullen de woorden ASTER CT-80 op het beeldscherm worden geplaatst. Nadat dit is gedaan zal een RETURN worden gegenereerd. Dit zorgt ervoor dat de cursor naar het begin van de volgende regel wordt geplaatst.
Het volgende programma geeft dan ook 3 regels tekst:

```
10 PRINT "ASTER"
20 PRINT "CT-80"
30 PRINT "MICROCOMPUTER"
```

Om op eenzelfde regel tekst te plaatsen uit meerdere PRINT statements dient direkt achter een PRINT aktie het punt-komma teken te worden opgenomen.
Voorbeeld:

```
10 PRINT "ASTER ";
20 PRINT "CT-80 ";
30 PRINT "MICROCOMPUTER"
```

Bovenstaand programma geeft 1 regel tekst. Achter ASTER en achter CT-80 staan "punt-komma" tekens. Deze zorgen ervoor dat de cursor direkt achter de afgedrukte tekst blijft staan.

PRINT@positie, informatie

Met het PRINT @ statement kunt u informatie op een willekeurige plaats op het video scherm afbeelden. Het videoscherm kent 1023 posities. (Zie ook de hulp-bladen voor werken op het video-scherm).
Voorbeeld:

```
10 INPUT A$
20 PRINT a 896,A$;
30 GOTO 30
```

Dit programma print de tekst die u in regel 10 invoerde op de voorlaatste schermregel af.
Om te voorkomen dat bij het plaatsen van informatie op de onderste regel het beeld "scrollt' dient u achter het print statement een punt-komma teken op te nemen.

PRINT TAB(positie)

Zo als bij een schrijfmachine tabulator stops mogelijk zijn, zo kunt u bij de ASTER CT-80 tabulator posities in uw PRINT statements programmeren. Het TAB statement met daarachter tussen haakjes de positie op de videoregel wordt eenvoudigweg direkt na het PRINT statement en voor de te printen informatie opgenomen.
Voorbeeld:

```
10 FOR A=1 TO 15
20 PRINT TAB(A)"ASTER CT-80 MICROCOMPUTER"
30 NEXT A
```

Bovenstaand programma zal de tekst 15 keer laten afdrukken, iedere afdruk 1 positie verder naar rechts verschoven.

Zoals u ziet mag de positienotering van de TAB ook een variabele zijn.

PRINT USING string,variabele

Dit statement vereenvoudigt en verfraait uw programma's drastisch. Als u cijferreeksen moet afdrukken dan is het meestal vereist om alle cijfers van gelijke waarde onder elkaar te zetten.
Door middel van PRINT USING kunt u een formaat vaststellen waardoor data in dat specifieke formaat zal worden afgedrukt.

De onderstaande tekens geven een specifiek formaat aan:

#
Dit teken wordt gebruikt om een numeriek formaat aan te geven. Het aantal van deze tekens dat gebruikt wordt bepaalt het aantal cijfers dat in het numerieke veld afgedrukt zal worden. .
Indien ergens in een rij van deze tekens een punt geplaatst wordt, zal de BASIC dit als de decimale komma beschouwen en een getal dat cijfers achter de komma kent dan ook zo afdrukken dat het in het aangegeven formaat past. Eventuele meerdere cijfers achter de komma dan in het formaat gespecificeerd, worden afgerond.

In de amerikaans/engelse rekentechniek zijn de komma en de punt ten opzichte van de nederlandse toepassing verwisseld. Daar waar de BASIC een komma in een cijferveld zou toepassen, zouden wij een punt plaatsen. Ook het omgekeerde is het geval. De zgn. "cijfers achter de komma" kent de BASIC als "cijfers achter de punt".
Voorbeeld:

Dit getal in nederlandse notering:

f. 12.345,67

wordt in het amerikaans formaat (BASIC):

f. 12,345.67

Het is even wennen, maar ja, BASIC is van origine een Amerikaanse Hogere Programmeertaal. Indien u dus perse een komma wilt plaatsen in een cijfernotering dan is dit uiteraard mogelijk. Het zal alleen een stukje subroutine kosten om alles om te zetten. Van PRINT USING kunt u dan voor deze toepassing geen gebruik meer maken.

Voorbeelden:

```
10 A$="##########"
20 INPUT A,B
30 PRINT USING A$;A
40 PRINT USING A$;B
```

Zoals u ziet zijn bij A$ een aantal van 8 cijfer-formaat-aangevers geplaatst. Vul nu bij de vraag van regel 20 een getal in, zonder cijfers achter de komma van 4 cijfers. Vul bij de tweede vraag een getal in van zes cijfers. Ziet u dat ze keurig onder elkaar komen te staan? Een gemakkelijk en nuttig statement.

Nu een voorbeeld met cijfers achter de decimale punt (onze komma, weet u nog wel?)

```
10 A$="##########.###"
20 INPUT A,B
30 PRINT USING A$;A
40 PRINT USING A$;B
```

Zoals u ziet zijn bij A$ een aantal van 8 cijfer-formaat-aangevers geplaatst. Daarna een decimale punt en dan weer 2 cijfer-formaat-aangevers. Vul nu bij de vraag van regel 20 een getal in, zonder cijfers achter de komma van vier cijfers en twee cijfers achter de decimale punt. Vul bij de tweede vraag een getal in van zes cijfers met drie cijfers achter de decimale punt. Ziet u dat ze keurig onder elkaar komen te staan?

**

Twee sterren (in computertaal "asterix" genoemd) aan het begin van een afdruk-formaat geplaatst zorgen ervoor dat alle ongebruikte posities van dat veld opgevuld worden met sterren. Een der toepassingen is het invullen van cheques of waardebonnen. Als u een veld specificeert waar bijvoorbeeld zes cijfers voor de decimale punt kunnen en u drukt een bedrag af met 2 cijfers voor de decimale punt dan worden de overblijvende 4 voorafgaande posities ingevuld met sterren.

Voorbeeld:

```
10 A$="**########.###"
20 INPUT A,B
30 PRINT USING A$;A
40 PRINT USING A$;B
```

Bij A$ zijn eerst twee sterren geplaatst, daarna een aantal van 8 cijfer-formaat-aangevers. Dan een decimale punt en weer 2 cijfer-formaat-aangevers. Vul nu bij de vraag van regel 20 een getal in, zonder cijfers achter de komma van vier cijfers en twee cijfers achter de decimale punt. Vul bij de tweede vraag een getal in van zes cijfers met drie cijfers achter de decimale punt.
U ziet dat de niet benutte ruimte voor de cijfers opgevuld wordt met sterren.

$$
Misschien niet voor nederlandse programma's, maar het is inbegrepen in de BASIC, dus toch de moeite waard om verteld te worden.
Twee dollar tekens aan het begin van een formaat string zorgen ervoor dat het dollar teken direkt voor het bedrag zal worden afgedrukt.
Voorbeeld:

```
10 A$="$$##########.###"
20 INPUT A,B
30 PRINT USING A$;A
40 PRINT USING A$;B
```

****$**
Als de sterren en het dollarteken gekombineerd worden geeft dit het effekt dat het dollarteken direkt voor het bedrag wordt afgedrukt. De overgebleven posities naar links worden dan ingevuld met sterren.
Voorbeeld:

```
10 A$="**$##########.##"
20 INPUT A,B
30 PRINT USING A$;A
40 PRINT USING A$;B
```

↑↑↑↑
Deze tekens aan het begin van de formaat string zorgen ervoor dat het getal exponentieel (E of D) afgedrukt zal worden.

+
Een plus-teken aan het begin of aan het einde van een formaat string zorgt ervoor dat op de plaats waar het teken wordt geplaatst (vooraan of achteraan) een plus of min zal worden afgedrukt. Dit uiteraard afhankelijk van het positief of negatief zijn van het getal zelf.
Voorbeeld:

```
10 A$="+##########.###"
20 INPUT A,B
30 PRINT USING A$;A
40 PRINT USING A$;B
```

Voer nu bij de eerste vraag een positief getal in en bij de tweede vraag een getal voorafgegaan door een min-teken.

Het min-teken geplaatst als laatste positie in een formaat string, zorgt ervoor dat bij negatieve getallen een min achter het bedrag wordt afgedrukt. Bij positieve getallen blijft de plaats open.
Voorbeeld:

```
10 A$="##########.##-"
20 INPUT A,B
30 PRINT USING A$;A
40 PRINT USING A$;B
```

Voer bij de eerste vraag een negatief getal in en bij de tweede vraag een positief getal.

%spaties%
Om een letter formaat string aan te geven van meer dan een karakter, wordt de formaat aangever: procent-spaties-procent gebruikt. De formaat string procent-spatie-procent geeft een aantal van 3 posities. Het aantal spaties dat tussen de procenten staat, vermeerderd met 2 geeft dus de lengte aan van de letter formaat string.
Voorbeeld:

```
10 A$="%    %"
20 INPUT B$
30 PRINT USING A$;B$
40 END
```

In regel 10 wordt A$ gedefinieerd als een letter string formaat met 5 spaties. Voer nu in B$ (regel 20) een tekst in van 3 letters. En daarna een tekst van 7 letters. Ziet u wat er gebeurt? Oefen zelf een aantal keer met andere letter string formaat lengten.

!
Dit teken in de formaat string geplaatst, geeft aan dat slechts het eerste karakter van af te drukken letterstring op de beeldbuis (of printer) zal worden geplaatst.
Voorbeeld:

```
10 A$="!"
20 B$="ASTER"
30 PRINT USING A$;B$
```

Geeft het resultaat:

```
A
```

Het eerste karakter uit B$ wordt dus slechts afgedrukt.

LPRINT - LINEPRINTER TOEPASSINGEN

Vrijwel alle normen die gelden voor het PRINT statement kunnen toegepast worden voor het LPRINT statement. Onderstaand treft u alle mogelijkheden met het LPRINT statement voor de regeldrukker oftewel lineprinter aan.
LET OP: Indien u een Daisy-Wheel (letterkwaliteit) printer heeft, kan het zijn dat de karakters van uw toetsenbord niet geheel overeenkomen met die op uw printer. Een ALPHA-NUMERIEK Daisy-Wheel voor computertoepassing geeft dezelfde letters als een matrix printer. Als u veel programma's maakt, is deze uitgave (over het algemeen minder dan f. 100,-) ruimschoots verantwoord.

Het statement LPRINT wordt toegepast met de volgende toevoegingen:
- LPRINT
- LPRINT@
- LPRINT TAB
- LPRINT USING

Het toepassen van het LPRINT statement zonder meer gaat als volgt:

```
10 LPRINT "ASTER CT-80"
```

Direkt aan het begin van de regel zullen de woorden ASTER CT-80 op de printer worden geplaatst. Nadat dit is gedaan zal een RETURN worden gegenereerd. Dit zorgt ervoor dat de printkop naar het begin van de volgende regel wordt geplaatst.
Het volgende programma geeft dan ook 3 regels tekst:

```
10 LPRINT "ASTER"
20 LPRINT "CT-80"
30 LPRINT "MICROCOMPUTER"
```

Om op eenzelfde regel tekst te plaatsen uit meerdere LPRINT statements dient direkt achter een LPRINT aktie het punt-komma teken te worden opgenomen.
Voorbeeld:

```
10 LPRINT "ASTER ";
20 LPRINT "CT-80 ";
30 LPRINT "MICROCOMPUTER"
```

Bovenstaand programma geeft 1 regel tekst. Achter ASTER en achter CT-80 staan "punt-komma" tekens. Deze zorgen ervoor dat de printkop direkt achter de afgedrukte tekst blijft staan.

LPRINT TAB(positie)

Zo als bij een schrijfmachine tabulator stops mogelijk zijn, zo kunt u bij de ASTER CT-80 tabulator posities in uw LPRINT statements programmeren. Het TAB statement met daarachter tussen haakjes de positie op de regel wordt eenvoudigweg direkt na het LPRINT statement en voor de te printen informatie opgenomen.
Voorbeeld:

```
10 FOR A=1 TO 15
20 LPRINT TAB(A)"ASTER CT-80 MICROCOMPUTER"
30 NEXT A
```

Bovenstaand programma zal de tekst 15 keer laten afdrukken, iedere afdruk 1 positie verder naar rechts verschoven.
Zoals u ziet mag de positienotering van de TAB ook een variabele zijn.

LPRINT USING string,variabele

Dit statement vereenvoudigt en verfraait uw programma's drastisch. Als u cijferreeksen moet afdrukken dan is het meestal vereist om alle cijfers van gelijke waarde onder elkaar te zetten.
Door middel van LPRINT USING kunt u een formaat vaststellen waardoor data in dat specifieke formaat zal worden afgedrukt.

De onderstaande tekens geven een specifiek formaat aan:

#
Dit teken wordt gebruikt om een numeriek formaat aan te geven. Het aantal van deze tekens dat gebruikt wordt bepaalt het aantal cijfers dat in het numerieke veld afgedrukt zal worden. .
Indien ergens in een rij van deze tekens een punt geplaatst wordt, zal de BASIC dit als de decimale komma beschouwen en een getal dat cijfers achter de komma kent dan ook zo afdrukken dat het in het aangegeven formaat past. Eventuele meerdere cijfers achter de komma dan in het formaat gespecificeerd, worden afgerond.

In de amerikaans/engelse rekentechniek zijn de komma en de punt ten opzichte van de nederlandse toepassing verwisseld. Daar waar de BASIC een komma in een cijferveld zou toepassen, zouden wij een punt plaatsen. Ook het omgekeerde is het geval. De zgn. "cijfers achter de komma" kent de BASIC als "cijfers achter de punt".
Voorbeeld:

Dit getal in nederlandse notering:

f. 12.345,67

wordt in het amerikaans formaat (BASIC):

f. 12,345.67

Het is even wennen, maar ja, BASIC is van origine een Amerikaanse Hogere Programmeertaal. Indien u dus perse een komma wilt plaatsen in een cijfernotering dan is dit uiteraard mogelijk. Het zal alleen een stukje subroutine kosten om alles om te zetten. Van LPRINT USING kunt u dan voor deze toepassing geen gebruik meer maken.

Voorbeelden:

```
10 A$="##########"
20 INPUT A,B
30 LPRINT USING A$;A
40 LPRINT USING A$;B
```

Zoals u ziet is bij A$ een aantal van 8 cijfer-formaat-aangevers geplaatst. Vul nu bij de vraag van regel 20 een getal in, zonder cijfers achter de komma van 4 cijfers. Vul bij de tweede vraag een getal in van zes cijfers. Ziet u dat ze keurig onder elkaar komen te staan? Een gemakkelijk en hulpzaam statement.

Nu een voorbeeld met cijfers achter de decimale punt (onze komma, weet u nog wel?)

```
10 A$="##########.##"
20 INPUT A,B
30 LPRINT USING A$;A
40 LPRINT USING A$;B
```

Zoals u ziet is bij A$ een aantal van 8 cijfer-formaat-aangevers geplaatst. Daarna een decimale punt en dan weer 2 cijfer-formaat-aangevers. Vul nu bij de vraag van regel 20 een getal in, zonder cijfers achter de komma van vier cijfers en twee cijfers achter de decimale punt. Vul bij de tweede vraag een getal in van zes cijfers met drie cijfers achter de decimale punt. Ziet u dat ze keurig onder elkaar komen te staan?

******

Twee sterren (in computertaal "asterix" genoemd) aan het begin van een afdruk-formaat geplaatst zorgen ervoor dat alle ongebruikte posities van dat veld opgevuld worden met sterren. Een der toepassingen is het invullen van cheques of waardebonnen. Als u een veld specificeert waar bijvoorbeeld zes cijfers voor de decimale punt kunnen en u drukt een bedrag af met 2 cijfers voor de decimale punt dan worden de overblijvende 4 voorafgaande posities ingevuld met sterren.
Voorbeeld:

```
10 A$="++##########.##"
20 INPUT A,B
30 LPRINT USING A$;A
40 LPRINT USING A$;B
```

Bij A$ zijn eerst twee sterren geplaatst, daarna een aantal van 8 cijfer-formaat-aangevers. Dan een decimale punt en weer 2 cijfer-formaat-aangevers. Vul nu bij de vraag van regel 20 een getal in, zonder cijfers achter de komma van vier cijfers en twee cijfers achter de decimale punt. Vul bij de tweede vraag een getal in van zes cijfers met drie cijfers achter de decimale punt.
U ziet dat de niet benutte ruimte voor de cijfers opgevuld wordt door sterren.

$$

Misschien niet voor nederlandse programma's, maar het is inbegrepen in de BASIC, dus toch de moeite waard om verteld te worden.
Twee dollar tekens aan het begin van een formaat string zorgen ervoor dat het dollar teken direkt voor het bedrag zal worden afgedrukt.
Voorbeeld:

```
10 A$="$$##########.###"
20 INPUT A,B
30 LPRINT USING A$;A
40 LPRINT USING A$;B
```

****$**

Als de sterren en het dollarteken gekombineerd worden geeft dit het effekt dat het dollarteken direkt voor het bedrag wordt afgedrukt. De overgebleven posities naar links worden dan ingevuld met sterren.
Voorbeeld:

```
10 A$="**$##########.###"
20 INPUT A,B
30 LPRINT USING A$;A
40 LPRINT USING A$;B
```

^^^^

Deze tekens aan het begin van de formaat string zorgen ervoor dat het getal exponentieel (E of D) afgedrukt zal worden.

+

Een plus-teken aan het begin of aan het einde van een formaat string zorgt ervoor dat op de plaats waar het teken wordt geplaatst (vooraan of achteraan) een plus of min zal worden afgedrukt. Dit uiteraard afhankelijk van het positief of negatief zijn van het getal zelf.
Voorbeeld:

```
10 A$="+##########.##"
20 INPUT A,B
30 LPRINT USING A$;A
40 LPRINT USING A$;B
```

Voer nu bij de eerste vraag een positief getal in en bij de tweede vraag een getal voorafgegaan door een min-teken.

-

Het min-teken geplaatst als laatste positie in een formaat string, zorgt ervoor dat bij negatieve getallen een min achter het bedrag wordt afgedrukt. Bij positieve getallen blijft de plaats open.
Voorbeeld:

```
10 A$="##########.##-"
20 INPUT A,B
30 LPRINT USING A$;A
40 LPRINT USING A$;B
```

Voer bij de eerste vraag een negatief getal in en bij de tweede vraag een positief getal.

% spaties %

Om een letter formaat string aan te geven van meer dan een karakter, wordt de formaat aangever: procent-spaties-procent gebruikt. De formaat string procent-spatie-procent geeft een aantal van 3 posities. Het aantal spaties dat tussen de procenten staat, vermeerderd met 2 geeft dus de lengte aan van de letter formaat string.
Voorbeeld:

```
10 A$="%    %"
20 INPUT B$
30 LPRINT USING A$;B$
40 END
```

In regel 10 wordt A$ gedefinieerd als een letter string formaat met 5 spaties. Voer nu in B$ (regel 20) een tekst in van 3 letters. En daarna een tekst van 7 letters. Ziet u wat er gebeurt? Oefen zelf een aantal keer met andere letter string formaat lengten.

!

Dit teken in de formaat string geplaatst, geeft aan dat slechts het eerste karakter van af te drukken letterstring op de beeldbuis (of printer) zal worden geplaatst.
Voorbeeld:

```
10 A$="!"
20 B$="ASTER"
30 LPRINT USING A$;B$
```

Geeft het resultaat:

A

Alleen het eerste karakter uit B$ wordt dus slechts afgedrukt.

PUT **DISKBASIC**

In toepassingen met RANDOM ACCES files wordt het PUT statement gebruikt. Het PUT statement is het instrument om data uit een BUFFER te halen en in een diskfile te plaatsen.
De toevoeging aan het PUT statement bepaald uit welke buffer en naar welk disk-record geschreven zal worden. Welke file zal worden ge-updatet wordt aangegeven door het OPEN statement.
Voorbeeld:

```
10 C$="ASTER CT-80 MICROCOMPUTER SYSTEEM"
20 OPEN "R",2,"TESTFILE"
30 FIELD 2,256AS B$
40 LSET B$=C$
50 PUT 2,27
```

In regel 10 wordt de string gedefinieerd. In regel 20 wordt de RANDOM ACCES file met de naam TESTFILE geopend. De buffer wordt gedefinieerd in regel 30, terwijl de string in de buffer geplaatst wordt in regel 40 met LSET.
Regel 50 tenslotte schrijft (PUT) de buffer naar record nummer 27. Gemakshalve wordt er vanuitgegaan dat record 1 t/m 26 reeds geschreven zijn.

RANDOM

Random is een soort CLEAR statement voor de RANDOM NUMBER GENERATOR, oftewel de willekeurig getal generator.
Bij het toepassen van een programma is het goed dit statement een keer bij het begin aan te roepen. Op deze wijze bent u er zeker van dat de getal generator echt "schoon" begint.

READ

Het READ statement "leest" informatie uit de geheugentabel die met het DATA statement werd gemaakt.
Voorbeeld:

```
10 DATA 1,2,3,4,5,6
20 FOR A=1 TO 6
30 READ B
40 PRINT B;
50 NEXT B
```

In regel 10 wordt een tabel gegeven van 6 getallen.
De FOR NEXT herhaling van regel 20 loopt 6 maal. Daardoor wordt in regel 30 het READ statement ook 6 maal herhaald. Iedere keer dat het READ statement gebruikt wordt houdt een interne teller bij welk getal achtereenvolgens uit de tabel moet worden gelezen door het READ statement. In regel 40 wordt het gelezen getal afgedrukt.

REM

Het REM statement wordt toegepast om kommentaar te leveren bij uw programma's. Ook voor eigen gebruik is het handig om bij bepaalde programmadelen een stukje kommentaar te plaatsen. De helpt u dan later herinneren hoe het programma werkte.
Voorbeeld:

```
10 FOR A=1 TO 10:REM Zet de herhaling 10 keer
20 INPUT B$:REM Geef de te herhalen tekst
30 PRINT B$:REM Druk de tekst af
40 NEXT A:REM Herhaal totdat A 10 is geworden
50 REM Dit is een programma voor de ASTER CT-80
```

Ook kunt u uw naam en/of adres op deze manier in het programma vermelden. Sommige programmeurs passen dit statement zeer veelvuldig toe. Een waarschuwing is wel op zijn plaats: REM statements en de tekst erachter nemen een stuk van het geheugen in beslag.
Onder NEWDOS-80.2 is het mogelijk alle REM-statements en alle spaties uit een programma te verwijderen. Dit geeft natuurlijk een aanzienlijke geheugenruimte winst. U kunt dan bijvoorbeeld twee versies van een programma hebben. De eerste versie voor het later controleren van het programma, de tweede (met de verwijderde REM's en spaties) als RUN programma.

Inplaats van het woord REM mag ook de apostrof worden toegepast (het teken boven de 7 op uw toetsenbord).

RENAME **DOS**

Dit kommando laat u een filenaam wijzigen. Als u bijvoorbeeld een file (data of programma) maakte en u wilt de naam wijzigen of er iets aan toe voegen dan kunt u dat met dit statement eenvoudig doen.
Voorbeeld:

```
RENAME TESTFILE TO EINDFILE
```

Of als u aan een file een toevoeging wilt geven:

```
RENAME TESTFILE TO TESTFILE/BAS
```

Als u meerdere diskdrives heeft aangesloten dan kunt u ook selektief naar een bepaalde drive verwijzen. U tijpt dan:

```
RENAME TESTFILE:2 TO EINDFILE
```

De file op diskdrive 2 wordt dan gewijzigd. Eventuele files met dezelfde naam op andere diskdrives blijven ongewijzigd.

Het RENAME kommando neemt de gehele filenaam, wist deze en kent een totaal nieuwe filenaam toe. **Dus ook een eventueel PASSWORD wordt verwijderd en dient aan de hernaamde file opnieuw te worden meegegeven.**

RESET(h,v)

Schakelt een grafisch blok op het beeldscherm uit. Het punt waar dit zal gebeuren wordt gedefinieerd door de getallen tussen haakjes. Deze getallen vertegenwoordigen de horizontale en vertikale positie op het scherm.
De getallen mogen ook worden benoemd door variabelen.

RESTORE

Het RESTORE kommando zet de interne teller die bijhoudt welk DATA statement gelezen werd op nul. Het volgende READ statement zal de allereerste DATA waarde inlezen.
Zie ook de DATA em READ statement beschrijvingen.

RESUME regelnummer

Na het afhandelen van een fout die door middel van de ON ERROR GOTO routine werd opgevangen wordt door het RESUME statement opnieuw gestart op de regel aangegeven achter het RESUME statement.

```
10 ON ERROR GOTO 70
20 INPUT "GEEF EEN WILLEKEURIG GETAL";A
30 B=2
40 C=A/B
50 GOTO 20
60 END
70 PRINT"NUL DELEN DOOR TWEE KAN NIET !"
80 RESUME 20
```

Probeer bovenstaand programma en geef bij regel 20 een nul(0) in.

RETURN

Na een "GOSUB regelnummer" zal het programma naar een subroutine springen. Aan het einde van die subroutine is het voldoende om het RETURN statement te plaatsen. De BASIC houdt namelijk bij vanaf welke plaats in het programma naar een subroutine wordt gesprongen. Bij een RETURN statement wordt teruggekeerd naar een positie direkt na die waarvandaan naar de subroutine werd gegaan.

RIGHT$(string,getal)

Geeft van een string het aantal karakters gedefinieerd in "getal", vanaf de rechterzijde van de tussen haakjes genoemde string. Dus RIGHTS(A$,7) geeft de laatste 7 letters van A$. Mocht A$ korter zijn dan het getal aangeeft, dan zal geen foutmelding worden gegeven. De gehele string zal dan worden afgedrukt.

RND(getal)

De "willekeurige getal" generator van de ASTER CT-80 kan getallen opwekken die volkomen willekeurig zijn.
Als het getal tussen haakjes een nul(0) is, wordt een integer getal tussen nul(0) en een(1) opgewekt.
De maximale grootte van het getal dat tussen haakjes mag worden geplaatst is 32768.
Voorbeeld:

```
10 FOR B=1 TO 10
20 A=RND(10)
30 PRINT A
40 NEXT B
```

Dit programma wekt 10 willekeurige getallen op tussen nul(0) en tien(10).
U kunt er natuurlijk de Lotto mee proberen in te vullen, of uw voetbal toto formulier. Er zijn tientallen toepassingen voor dit statement.

SAVE **DISKBASIC**

Na het maken van een BASIC programma wilt u dit ook graag bewaren. Met het SAVE kommando plaatst u een BASIC programma op diskette. U kunt het programma gewoon een naam geven. De naam mag maximaal 8 karakters lang zijn en mag alleen letters of cijfers bevatten. **Het eerste karakter moet altijd een letter zijn.** Punten, komma's en andere tekens mogen niet worden gebruikt, tenzij met speciale bedoelingen (PASSWORDS enz.)
Voorbeeld:

```
SAVE"TESTFILE"
```

Zal de file met de naam TESTFILE op de diskette plaatsen. Als u een toevoeging wilt meegeven dan kunt u dit doen door na de filenaam een schuine streep te tijpen (de streep op de vraagteken toets) en een drie-letterige toevoeging te tijpen.
Niet alle toevoegingen zijn geldige koderingen. Een goed gebruik is om voor Tekst-Files de toevoeging TXT te geven. Als het BASIC programma's zijn kunt u de toevoeging BAS meegeven.

Om files te kunnen MERGEn (mengen) met programma's in het geheugen dient een file in het ASCII formaat te worden weggeschreven naar diskette. Het normale SAVE kommando schrijft de file namelijk in een gekomprimeerd formaat naar de diskette.
De file die u later wilt MERGEn kunt u op de volgende manier SAVEn:

```
SAVE "TESTFILE/BAS",A
```

De toevoeging ,A zorgt ervoor dat de file in het ASCII formaat wordt weggeschreven.
Uiteraard kunt u een per ongeluk niet in ASCII formaat weggeschreven file ook weer laden en dan alsnog in ASCII wegschrijven.

SET(h,v)

Schakel het grafisch blok in op de positie gedefinieerd door (h,v). Waarin h=horizontaal (0-127) en v=vertikaal (0-48). Hiermee is het grafisch scherm van de ASTER CT-80 te besturen. Het totale grafische scherm kent een hoeveelheid van 6.144 blokjes die afzonderlijk aan- en uitgeschakeld kunnen worden door de kommando's SET en RESET. De coordinaten mogen gegeven worden in direkte getallen, variabelen of konstanten en behoeven niet perse integeren te zijn, daar het SET en RESET statement slechts het integer deel van een getal gebruikt. De linker bovenhoek van het grafisch scherm is SET(0,0). De rechter benedenhoek van het scherm is SET(127,47).

Als u het op uw scherm eens wilt laten "sneeuwen" dan voldoet onderstaand programma direkt aan uw wensen.

```
10 SET(RND(128)-1,RND(48)-1)
20 GOTO 10
```

Zie ook RESET

SGN(variabele)

De Vlag-funktie geeft -1 als de variabele negatief is, nul als de variabele nul is en +1 als de variabele positief is.
Voorbeeld:

```
10 INPUT A
20 ON SGN(A)+2 GOSUB 110,150,200
30 GOTO 10
110 REM SUBROUTINE 1
150 REM SUBROUTINE 2
200 REM SUBROUTINE 3
```

Afhankelijk van het negatief, nul of positief zijn van een getal kunt u op deze wijze naar een subroutine springen.

SIN(getal

Geeft de sinus van het getal. Het getal moet in radialen zijn. Om de sinus van A te verkrijgen als A in graden is kunt u de onderstaande formule in uw programma opnemen.
Voorbeeld:

```
10 B=SIN(A+.01745329)
20 PRINT B
```

SQR(getal)

Geeft de vierkantswortel van het getal.
Voorbeeld:

```
10 A=SQR(X 2-H 2)
20 PRINT A
```

STOP

Dit statement genereert een BREAK in uw programma. U kunt het gebruiken tijdens het programmeren om in uw programma tijdelijke STOPs te zetten en dan bijvoorbeeld variabelen op hun waarde te controleren. Als u het programma niet d.m.v. het EDIT kommando wijzigt, dan kunt u met het CONT kommando het programma weer laten vervolgen. Is het programma uiteindelijk naar uw zin, dan verwijdert u deze statements.
Voorbeeld:

```
10 FOR A=1 TO 10
20 IF A=7 THEN STOP
30 NEXT
```

STRING$(aantal,getal of "karakter")

Een kommando voor het verfraaien van videoscherm of te printen formulieren.
Het STRING$ kommando laat u een aantal karakters definieren dat u als een lange reeks van tekens op videoscherm of printer afdrukt. Iedere variabele tussen 0 en 255 mag worden gebruikt. Ook kunt u direkt een karakter in dit kommando toepassen.
Voorbeeld:

```
10 PRINT STRING$(60,191)
```

Geeft een aaneengesloten rij van 60 grafische blokjes ter grootte van een karakter.

Voorbeeld:

```
10 PRINT STRING$(60,"+")
```

Geeft een aaneengesloten rij van 60 plusjes.

STR$(variabele)

Dit statement verandert de waarde van de variabele in string-karakter(s). Voor de string-karakter(s) wordt een spatie afgedrukt. De variabele is ook een echt string-karakter geworden. Rekenkundige bewerkingen kunnen niet meer erop worden toegepast.
Voorbeeld:

```
10 B=20
20 F$=STR$(B)
30 PRINT F$
```

Het resultaat van bovenstaand programma is het afdrukken van het getal 20, maar nu als string-variabele, voorafgegaan door een spatie.

TAN(getal)

Geeft de tangens van het getal. Het getal moet in radialen zijn.

THEN

Gedeelte van totaal statement IF ....THEN.
Voorwaardelijke sprong. **Zie ook IF.**
Voorbeeld:

```
10 INPUT A
20 IF A=10 THEN PRINT"ASTER CT-80"
30 GOTO 10
```

TIME$ **DISKBASIC**

De software-matig opgewekte Real Time Clock in de ASTER CT-80 houdt een aantal zaken bij. Als eerste de datum in het Maand/Dag/Jaar formaat en daarbij ook de tijd in het Uur/Minuut/Sekonde formaat. De notering van de tijd is in het 24-uur formaat zoals u dat ook van een digitaal horloge kent. Nog een bijzonderheid die u ook van digitale horloges zult herkennen: de datum wordt niet zoals bij ons in dag, maand, jaar weergegeven. Nee, de maand wordt eerst gegeven, daarna de dag en het jaar. Dit wordt veroorzaakt door het feit dat ook de klok-routine afkomstig is uit de amerikaanse tijdberekening.

De klok-routine is zodanig opgezet dat hij op de laatste van de maand precies weet of de 30e als laatste dag moet worden aangemerkt of de 31e. Ook in februari zal de 28e als laatste dag worden gezien. Mocht u de computer echter dag en nacht laten aanstaan, dan moet u eens in de vier jaar (schrikkeljaar) de datum even bijwerken.

Overigens, er kleeft een nadeel aan de software-matige klokroutine. De routine is interrupt gestuurd. Dat wil zeggen dat bij het gebruik van andersoortige INPUT/OUTPUT die ook de interrupt nodig heeft, de computer de klok even "vergeet". Dit resulteert in het "achterlopen" van de klok bij Disk I/O en bij Cassette I/O. Tevens zal bij het uitschakelen van de computer ook de tijd-registratie uitgeschakeld worden. .
Bij aanschaf van het Real Time Clock Board, worden deze nadelen ondervangen. Het Real Time Clock Board heeft een geheel eigen, microprocessor gestuurde klokroutine. Of de computer nu op het lichtnet aangesloten is of niet, deze klok gaat gewoon door, een oplaadbare batterij zorgt ervoor dat ondanks het afwezig zijn van netspanning de klok gewoon gedurende bepaalde tijd doorgaat. Raadpleeg hiervoor de handleiding van het Real Time Clock Board. (Leverbaar in de loop van 1983).

De lengte van de TIME$ is 17 karakters lang en als volgt opgebouwd:

MM/DD/JJ UU:MM:SS

Maand, dag, jaar, uur, minuut en sekonde vermelding is altijd in een twee-cijferig formaat. Bij getallen onder 10 gaat aan het cijfer een nul(0) vooraf.
Bij gebruik van TRS-DOS moet u de klok vanuit de DOS gelijkzetten. Bij het gebruik van NEWDOS-80.2 kunt u dit vanuit BASIC doen door middel van het CMD kommando.
Zie voor het instellen van de klok de DOS-kommando's TIME en DATE.

Om het voor u wat handiger te maken zullen we ook in dit hoofdstuk het instellen van de klok bespreken:
Tijdens het in werking zijn van TRS-DOS (of vanuit BASIC onder NEWDOS-80.2 d.m.v. CMD" etc) tijpt u:

DATE 01/03/83

TIME 11:30:00

U heeft nu de datum ingesteld op 3 januari 1983 en de tijd op half twaalf 's-ochtends precies.
Om dit te controleren tijpt u:

PRINT TIME$

Als u alles goed heeft gedaan, zal de computer de volgende tekst op het scherm plaatsen:

01/03/83 11:30:00

U kunt het TIME$ statement bijzonder goed toepassen in programma's die op een bepaald moment een handeling dienen te verrichten.
Voorbeeld:

```
10 A$="14:45:00"
20 IF RIGHT$(TIME$,8)=A$ THEN 30 ELSE 20
30 PRINT"Het is kwart voor twee precies."
40 INPUT"Geef een nieuwe tijd: ";A$
50 GOTO 20
```

Regel 10 geeft de tijd waarmee vergeleken moet worden. In regel 20 wordt de tijd van de klok vergeleken met de string van regel 10. Als TIME$ niet gelijk is aan A$ dan gaat via het ELSE statement het programma terug naar het begin van regel 20. Dit gaat door totdat de tijd bereikt is. Via het THEN statement valt het programma door naar regel 30 en geeft op uw beeldscherm de boodschap dat de gewenste tijd bereikt is. Direkt daarna vraagt het programma of u op een nieuw tijdstip gewaarschuwd wilt worden.
Na invoer van het nieuwe tijdstip (A$) gaat het programma weer naar regel 20.

TRON en TROFF
Bij het maken en korrigeren van een programma zijn de TRON en TROFF funkties een bijzonder sterk hulpmiddel. Indien u voordat het programma gaat "runnen" het kommando TRON intypt, dan zal de computer bij het uitvoeren van iedere programmaregel het regelnummer tussen twee haakjes op het beeldscherm afdrukken.
Voorbeeld:

```
10 PRINT"ASTER CT-80"
20 PRINT"MICROCOMPUTER"
30 PRINT"SYSTEEM"
```

Als u TRON tijpt voordat u het bovenstaande programma runt dan zal het resultaat zijn:

```
(10) ASTER CT-80
(20) MICROCOMPUTER
(30) SYSTEEM
```

Met TROFF schakelt u deze funktie uit.

Uiteraard kunt u dit kommando in uw programma's opnemen. Het kan zijn dat een bepaald gedeelte van uw programma niet helemaal doet wat u wilt. U wilt graag weten wat er precies gebeurt en welk regelnummer een gebeurtenis veroorzaakt.
U doet dan als volgt:

```
10 PRINT"ASTER CT-80"
20 TRON
30 PRINT"MICROCOMPUTER"
40 TROFF
50 PRINT"SYSTEEM"
```

Het resultaat ziet er zo uit:

```
ASTER CT-80
(30) MICROCOMPUTER
SYSTEEM
```

Als het programma naar uw zin is dan kunt u de regels met de TRON en TROFF statements verwijderen.
Bij een wat ingewikkelder programma kan het zijn dat het u wat te vlug gaat. Hier geldt weer dezelfde regel als bij het LIST kommando: het indrukken van SHIFT en de a -toets tegelijkertijd doet het programma tijdelijk stoppen. Het aanraken van een volgende toets - dat mag een willekeurige zijn - laat het programma weer verder lopen.

USR(getal)

BASIC is een hogere programmeertaal. Dat wil zeggen dat de taal zelf zodanig geschreven is dat in eenvoudig Engels een aantal opdrachten gegeven kunnen worden. De taal is niet "machine-georienteerd" dat wil zeggen: u behoeft eigenlijk geen volleerd technicus te zijn om met de ASTER CT-80 programma's te gaan maken. Eigenlijk iedereen, die wat logisch nadenkt, weet wat hij wil en er de tijd voor neemt, kan een BASIC programma maken. De "moeilijkheden" worden door de BASIC voor u opgeknapt. BASIC heeft echter net zoals alle hogere programmeertalen een nadeel. Het is relatief langzaam. Dit wordt veroorzaakt door het feit dat de taal (BASIC) uw programma moet omzetten in voor de microprocessor begrijpelijke cijferreeksen. Deze cijferreeksen zijn instrukties voor de microprocessor. Zij kan ze bijzonder snel uitvoeren en geeft dan de resultaten weer "terug" aan de BASIC, die ze voor ons weer in begrijpelijke taal omzet. BASIC is dan ook een zgn. "INTERPRETER". BASIC interpreteert de kommando's voor de microprocessor en interpreteert ze voor de gebruiker.
Uiteraard is een programma dat rechtstreeks geschreven is in machine-codes (begrijpelijke instrukties voor de Z80 microprocessor) velen malen sneller. Een nadeel is echter dat het programma ook wat minder overzichtelijk is. Ook wordt veel meer inzicht van de programmeur gevraagd in de inwendige konstruktie van de microprocessor en de microcomputer zelf.

Nu is het echter niet altijd noodzakelijk om programma's in machinetaal te schrijven als bepaalde zaken zeer snel moeten gebeuren. Meestal is het zo, dat enkele delen van het programma om grote snelheid vragen. Andere delen kunnen dan weer rustig op het tempo dat BASIC geeft worden uitgevoerd. In die gevallen past men subroutines toe die in machinetaal geschreven zijn. Vanuit BASIC worden ze dan aangeroepen. De machinetaal-routine eindigt weer met een sprong (JUMP) terug naar het punt waar hij het BASIC programma uitgesprongen is.

In LEVEL II BASIC kan slechts een machinetaalroutine worden aangeroepen. Dat wil zeggen, men kan 1 routine tegelijkertijd toepassen. Met DISKBASIC kan men 10 USR-routines tegelijkertijd in het geheugen hebben en aanroepen wanneer nodig.

Een USR-routine kan in het geheugen worden gePOKEd door middel van DATA regels en READ statements vanuit het BASIC programma.
Tevoren dient men de MEMORY SIZE? te zetten, zodanig dat het gedeelte waarin de USR-routine staat niet door de BASIC overschreven kan worden.
Na het inPOKEn van de routine in het geheugen dient men - voordat het USR statement gebruikt kan worden - de BASIC te vertellen wat het ingangspunt (entry-point) van de USR-routine is. Dit gebeurt door het adres door middel van POKE statements in de geheugen-lokaties 16526 en 16527 te brengen. Het minst signifikante byte in lokatie 16526, het meest signifikante byte in lokatie 16527.
Bij het inbrengen van de informatie in deze geheugenplaatsen dient men er wel aan te denken dat het adres eerst in hexadecimaal formaat moet worden omgezet, daarna in twee delen moet worden gesplitst en deze twee hexadecimalen getallen dienen dan weer in decimale vorm in de twee adressen te worden gePOKEd. Misschien wat omslachtig, maar uiteindelijk geeft het aan bepaalde programma's later meer snelheid.
Voorbeeld:
Als het ingangs-punt decimaal 327062 is dan is dit hexadecimaal het getal 7FFA. Dit wordt gesplitst in 7F en FA. 7F is het meest signifikante byte, FA het minst signifikante. Het getal 7F is decimaal 127, terwijl FA in decimaal 250 is.
Het ingangspunt voor de USR-routine wordt dus als volgt ingebracht:

```
10 POKE 16526,250:POKE 16527,127
```

Als u de routine in het geheugen heeft geplaatst, beginnend bij het gegeven entry-point, dan kunt u deze aanroepen door in uw programma het statement:

```
120 A=USR(X)
```

aan te roepen.

Waarbij X een willekeurige getal mag zijn, dienend als vulmiddel. Het is ook mogelijk om via X een getal aan de subroutine mee te geven, maar hierover later meer.

Na het uitvoeren van de sub-routine zal het BASIC programma weer vervolgen.

Indien u variabelen mee wilt geven aan de USR-routine dan is dit mogelijk door aan het begin van de routine zelf de instruktie CALL 0A7FH te plaatsen. Deze instruktie zorgt ervoor dat de waarde X (zie regel 120 hierboven) in het HL-register van de Z80 microprocessor wordt geplaatst. Ook het omgekeerde funktioneert. Na het toepassen van de machinetaal routine kunt u een resultaat (integer) in het 16-bit brede HL register plaatsen. In plaats van een C9 instruktie (JUMP terug naar BASIC) geeft u dan een JP 0A9AH instruktie. Deze zorgt er dan voor, dat het in de HL-register aanwezig getal in de BASIC toegekend wordt aan de variabele A uit het A=USR(X) statement.

Een bijzonder goed boek over Z80 machinetaal routines is het Z80 Programming Manual van Rodney Zacks. In honderden pagina's wordt zeer begrijpelijk uitgelegd hoe de Z80 processor funktionert en hoe zijn set van 158 instrukties te gebruiken is. Het zou te ver gaan om daarover in dit boek veel te vertellen. Het is heel wat complexer dan BASIC, doch er zijn ook instrukties die in enkele kommando's bijvoorbeeld duizend karakters in het geheugen kunnen verschuiven binnen microseconden in tijd.
Veel computerspellen maken gebruik van dit soort instrukties om "bewegende" beelden te maken. Dit is niets meer (of minder) dan het zeer snel achterenvolgend verplaatsen van blokken informatie uit het werkgeheugen naar het video-geheugen (desnoods tientallen komplete beeldschermen per sekonde).
In BASIC zou dit enkele sekonden duren, het beoogde effekt zou dan verloren zijn.

Voor het toepassen van kleine routines vanuit BASIC, terwijl u toch geen machine-taal-programmeur wilt worden is het boekje TRS-80 Assembly-Programming door William Barden Jr. en het EDITOR-ASSEMBLER manual van Radio Shack (bestelnummer 26-2002) bijzonder geschikt. In het boekje van William Barden Jr. vindt men komplete, zo toepasbare routines. In het Editor-Assembler manual treft men aanwijzingen hoe zelf in machinetaal te programmeren door middel van dit programma, dat bij NEWDOS-80.2 reeds op de diskette staat.

VAL(string)

Als u een karakter-string heeft, bestaande uit cijfers, dan kunt u er met het VAL statement een variabele van maken. BASIC werkt bijzonder logisch. Alle bewerkingen die u met BASIC op variabelen kunt toepassen kunnen direkt op de omgezette string van toepassing zijn.
Voorbeeld:

```
10 A$="80"
20 B=VAL(A$) 30 PRINT B
```

Geeft als resultaat het getal 80. Maar ook rekenkundige bewerkingen kunnen in een keer worden toegepast.
Voorbeeld:

```
10 X$="10"
20 Y$="E"
30 Z$="17"
40 A=VAL(X$+Y$+Z$) 50 PRINT A
```

Geeft als resultaat 10E17. Dit is aktueel 10 x 10 tot de 17e.

Alleen de cijfers die in de string zijn opgenomen worden omgezet naar variabelen.
Voorbeeld:

```
10 A$="ASTER CT-80"
20 A=VAL(A$) 30 PRINT A
```

Geeft als resultaat het getal 80.
Als in het geheel geen getal in de string aanwezig is, ofdat het getal nul(0) is, dan wordt een nul(0) gegeven.

VARPTR(variabele)

Als u een programma in verschillende delen van diskette wilt "draaien" en daarbij verschillende variabelen wilt bewaren, dan kunt u met het VARPTR statement te weten komen waar de BASIC de variabelen heeft "opgeborgen".
Voorbeeld:

```
10 X=VARPTR(Y)
20 PRINT X
```

Waarbij Y een integer variabele is, geeft voor X een getal dat het minst signifikante byte van de integer bevat. X-plus-1 is dan het adres waar het meest signifikante deel van het byte opgeslagen is.

Indien Y een enkel precisie getal is, dan geeft X het minst signifikante byte, X+1 het op-een-na meest signifikante byte, X+2 het meest signifikante byte, terwijl X+3 het eventuele exponent van het getal bevat.

Indien Y een dubbel precisie getal is, dan geeft X het minst signifikant byte, X+1 een volgend byte, enzovoort door tot X+6 het meest signifikante byte aangeeft. X+7 bevat dan het exponent van het getal.

Bij enkel en dubbel precisie is het hoogste bit van X+6 het vlag-bit. Indien dit 0 is dan is het getal positief. Indien het 1 is, is het getal negatief.

Ook kan met VARPTR de adressen worden opgevraagd van string variabelen.
Voorbeeld:

```
10 A$="ASTER CT-80"
20 X=VARPTR(A$)
30 PRINT X
```

Dan geeft X de lengte van de string, X+1 het minst signifikante byte van het string-start-adres, terwijl X+2 het meest signifikante byte van het string-start-adres aangeeft.
Meest waarschijnlijk is, dat de adressen zich in het hoogste deel van het RAM geheugen zullen bevinden, daar BASIC bij het CLEAR statement in dat deel ruimte reserveert voor strings.

Indien u door middel van VARPTR de plaats van ARRAYs wilt weten dan is de handelwijze als volgt:

```
10 DIM AB(5,10)
20 A=VAL(AB)
```

Dan geeft A het adres van het eerste byte van het element in het array AB. Afhankelijk van het array: integer, enkel-, dubbel-precisie of string-array, zal de lengte van dat element 2 bytes, 4 bytes, 8 bytes of 3 bytes zijn.

02-68

VERIFY DOS

Met dit kommando laat u het Disk Operating Systeem alle files die naar disk worden weggeschreven kontroleren. Dit kontroleren gebeurt direkt na het wegschrijven. Er moet wel rekening gehouden worden met een vertraging van meer dan de helft van de normale tijd als dit kommando in werking is.
Het inschakelen van het kommando is eenvoudig. Als u "in" het Disk Operating Systeem bent - de melding DOS READY staat op uw scherm - dan tijpt u VERIFY.
Alle disk-schrijf akties worden dan gekontroleerd. Het uitschakelen gebeurt ook onder DOS. U tijpt dan VERIFY (OFF).

APPEND DOS

Als u meerdere data-files "aan elkaar wilt knopen" dan is daarvoor het APPEND kommando.
Voorbeeld:

APPEND TESTFILE TO DATAFILE

Geeft het resultaat dat DATAFILE tevens - aan het einde - de informatie van TESTFILE zal bevatten. TESTFILE zelf blijft ongewijzigd.
Ook BASIC programma's kunnen geAPPEND worden. Deze programma's dienen dan beide met de A-optie (ASCII) weggeschreven te zijn, terwijl de regelnummers in het eerst genoemde programma hoger dienen te zijn dan in die van het tweede programma. Logisch natuurlijk, want de bedoeling van het APPEND kommando op een BASIC programma is, dat het ook als een programma kan "draaien".

AUTO DOS

Soms is het gemakkelijk om een programma van diskette automatisch te laten opstarten als de computer gestart of aangezet wordt. Het AUTO kommando verzorgt het opstarten van een programma. (In TRS-DOS alleen een machine-taal programma, in NEWDOS-80.2 kan ook een BASIC programma worden opgestart).
De toepassing is eenvoudig. Zorg dat het "write-protect-label" van uw diskette verwijderd is. Tijpe nu:

AUTO TEKST

En u krijgt de melding DOS READY. Iedere keer als u nu de reset-toetsen indrukt, terwijl deze diskette in uw systeem in drive "0" geplaatst is, wordt het programma TEKST automatisch opgestart.
LET OP: een AUTO kommando op een diskette waar niet het programma op staat dat gestart moet worden geeft een foutmelding.

Het terugzetten van het kommando geschiedt als volgt:

AUTO

U tijpt dus het woord AUTO, zonder filenaam.

ATTRIB **DOS**

Met dit kommando geeft u aan een file een bepaalde status. Deze status kan zijn:

- onzichtbaar (I)
- toegangs-wachtwoord (PASSWORD)
- verander-wachtwoord
- bescherming tegen div. akties

Met toevoeging van de I optie maakt u een file voor de normale DIRECTORY (inhouds-opgave) onzichtbaar. Dit kan niet ongedaan worden gemaakt, behalve met het RENAME kommando.

De wachtwoord (PASSWORD) mogelijkheden onder het Disk Operating Systeem hebben een goede toepassingsmogelijkheid.
Zo kunt u verschillende "klassen" van toegang toekennen. Als we als voorbeeld een voorraad programma nemen, waar meerdere mensen toegang in hebben dan kunnen we als volgt wachtwoorden toevoegen:

ATTRIB VOORRAAD/DATA (ACC=JANSEN,UPD=DEGROOT,PROT=READ)

In bovenstaande regel worden de volgende protecties gezet: DEGROOT mag in de file veranderen, zijn mogelijkheden zijn onbegrensd. JANSEN mag bij het ACCESSEN (de toegang tot de file) alleen lezen en niets veranderen. Dit werd vastgelegd in het PROT=READ statement.
De notering is zodanig dat eerst een wachtwoord wordt gegeven voor ACCES, dus toegang. Dan een wachtwoord voor UPDATE, dus wijziging, daarna wordt de wijze van bescherming voor het ACCESSEN (toegang) vastgelegd in PROT=READ. De PROTectie van ACC is dus READ (alleen lezen, bekijken).

De verschillende klassen waarin protectie kan worden gezet onder het "ACC=" zijn:

KILL - alles toegestaan
RENAME - andere naam geven, aanvullen, lezen, uitvoeren
WRITE - aanvullen, lezen, uitvoeren
READ - lezen, uitvoeren
EXEC - alleen uitvoeren

Met uitvoeren wordt gedoeld op de mogelijkheid dat een file een programma is dat "uitgevoerd" wordt. Een READ klassement mag dus het programma bijvoorbeeld LISTen, terwijl een EXEC klasse alleen maar het programma mag laten draaien (onder BASIC werkt LOAD"PROGRAMMA" niet).

Het wijzigen of veranderen van wachtwoorden kan alleen gebeuren door personen die een toegangs-status hebben dat onder UPD staat of ACC onder PROT=KILL.
Om wachtwoorden te verwijderen of een toegangs-status te veranderen handelt men als volgt:

ATTRIB VOORRAAD/BAS.DEGROOT (ACC=)

In bovenstaande situatie wordt de status van het ACC wachtwoord op nul gezet, terwijl de UPD status gehandhaafd blijft.

BASIC **DOS**

Hiermee roept u de hogere programmeertaal aan.

BASIC2 **DOS**

Vanuit de DOS kunt u met dit kommando naar de ROM georienteerde BASIC LEVEL II springen. Als u vanuit LEVEL II BASIC weer naar DOS wilt tijpt u:

SYSTEM

+? /0

BASICR **DOS**

Hetzelfde kommando als BASIC, met het verschil dat u de Disk Basic aanroept die ook een hernummer (RENUMBER) kommando heeft. In NEWDOS-80.2 is dit een aanroepbare routine onder normaal Disk Basic.

CLOCK **DOS**

Het intijpen van het kommando CLOCK laat in de rechter bovenhoek van uw scherm de tijd afdrukken. Zie het kommando DATE en TIME voor het op-tijd-zetten van de CLOCK.

COPY **DOS**

Het kopieren van een file van de ene diskette naar de andere doet u met het COPY kommando. U kunt tegelijk de file RENAMEn als u dat wilt.
Voorbeeld:

COPY OUDFILE/BAS:1 TO NWFILE/BAS:0

Hiermee kopieert u de file OUDFILE/BAS op drive 1 naar de nieuwe file NWFILE/BAS op drive 0. De file met de naam OUDFILE/BAS blijft bestaan. Een eventuele file met de naam NWFILE/BAS op drive 0 wordt met dit kommando overschreven.

DATE **DOS**

Het zetten van de datum is een eenvoudige handeling:

DATE 01/03/83

Bovenstaand kommando zet de datum op 3 januari 1983.

DIR **DOS**

Hiermee vraagt u de inhoudsopgave van een diskette.
Als u DIR zonder meer tijpt krijgt u altijd de inhoudsopage van drive 0. .
DIR :1 geeft u de inhoud van drive 1, enz.

DIR :1 S,I,A

Geeft u alle files van drive 1. Waarbij de toevoeging I u alle onzichtbare files laat zien (zie ATTRIB). De toevoeging S geeft u een overzicht van alle SYSTEM files. SYSTEM files zijn files die gebruikt worden door het Disk Operating Systeem. U ziet ze genummerd vanaf SYS0/SYS en hoger.
De A-toevoeging geeft u een overzicht hoeveel ruimte er door de diverse files op de diskette wordt ingenomen.

DUMP **DOS**

Met het DUMP kommando kunt u een stuk uit het geheugen naar diskette wegschrijven. Als u met een programma informatie op een specifieke plaats in het geheugen heeft gezet, dan kunt u met het DUMP kommando exact aangeven vanaf welk adres tot welk adres u naar disk wilt schrijven.
Als u een machinetaal programma in het geheugen heeft gemaakt, dan kunt u ook het entry-point van het programma meegeven. Als het programma dan later van disk wordt aangeroepen zal het automatisch vanaf dit entry-point opstarten.
Voorbeeld:

DUMP TEKST/CMD (START=X'6C00,END=X'8000,TRA=X'6B00)

Bovenstaande regel schrijft vanuit het geheugen een programma naar de diskette onder de naam TEKST met de toevoeging CMD. Het startadres is 6C00, hexadecimaal, het eindadres is 8000 hexadecimaal en het entry-point is 6B00.
Als geen entry-point wordt meegegeven dan zal na het laden van de file vanaf disk, de DOS terugspringen naar adres 402D. Dit is het normale re-entry adres van het Disk Operating Systeem na het uitvoeren van een bepaalde routine.

Voor het DUMPen van een tabel bijvoorbeeld behoeft u geen "entry-adres" mee te geven. U schrijft de tabel dan als volgt weg:

DUMP TABEL/CIM (START=X'5000,END=X'5FFF)

Als u vanuit DOS dan deze file LOAD (dus naar het geheugen laat brengen) dan zal na het laden de melding DOS READY weer op het scherm verschijnen. Het ingeven van het start en eind adres bij het DUMPen van een gedeelte van het geheugen zorgt er later bij het LOADen automatisch voor dat deze informatie weer op dezelfde plaats in het geheugen komt.

KILL **DOS**

Het kommando voor het uitwissen van files. Bij een beschermde (protected) file dient het password te worden meegegeven. Voorbeeld:

KILL TESTFILE/BAS.PASSWORD

FREE **DOS**

Geeft de vrije ruimte op alle aanwezige diskdrives. Het kommando wordt gegeven zonder verdere toevoegingen en kontroleert alle aanwezige diskdrives. Vrije ruimte op diskettes wordt onder TRS-DOS-achtige Disk Operating Systemen uitgedrukt in GRANULES. Een GRAN of GRANULE is 1.25 Kilobyte.
Een TRS-DOS Diskette heeft nog 25 GRANS vrij. Een TRS-DOS Datadiskette heeft nog 67 granules vrij. Een diskette kan tot 48 verschillende filenamen bevatten. De minimale ruimte door een file ingenomen bedraagt 1 GRANULE.
Een GRAN of GRANULE is 1.25 Kilobyte.
Een NEWDOS-80.2 Datadiskette heeft 77 GRANULES oftewel 96.25 Kilobyte aan netto opslagkapaciteit.
Alle capaciteiten worden opgegeven bij toepassing van single side (een-zijdig) single-density (enkele densiteit), single-track-density (40-track) diskdrives.
Zie voor de capaciteit van de diverse diskdrive configuraties het hoofdstuk Diskdrives.

LIB **DOS**

Het LIB kommando geeft alle beschikbare DOS kommandos.
Voorbeeld:

LIB

Alle kommandos worden nu op het beeldscherm afgebeeld.

LIST **DOS**

Met het LIST kommando kunt u onder DOS een der aanwezige files op het beeldscherm laten afdrukken. De te LISTen files die niet in ASCII werden weggeschreven kunnen moeilijk leesbare cijer en letterreeksen teweeg brengen.
Om het LISTen tijdelijk te stoppen moet u de SHIFT toets tesamen met de a -toets indrukken. Het indrukken van een willekeurige andere toets doet het LISTen vervolgen.

PRINT **DOS**

Werkt precies eender als LIST, doch de output gaat niet naar het videoscherm maar naar de printer.

LOAD **DOS**

Met LOAD brengt u machinetaal routines of tabellen in het geheugen die eerder met DUMP naar diskette werden geschreven. Ook kunt u hiermee files die met het TAPEDISK kommando werden gemaakt in het geheugen plaatsen.

PROT **DISK**

Het PROT kommando kent drie toevoegingen:
- PW
- LOCK
- UNLOCK

PW geeft de mogelijkheid het wachtwoord (PASSWORD) te wijzigen.
LOCK plaatst het hoofd wachtwoord (MASTER PASSWORD) bij alle files.
UNLOCK verwijdert alle wachtwoorden van alle files op de diskette.

TRS-DOS en NEWDOS-80.2 diskettes hebben het wachtwoord PASSWORD bij de produktie meegekregen.
Voorbeeld:

PROT :2 (PW)

Na het ingeven van dit kommando vraagt de DOS u om een nieuw wachtwoord te geven.

PROT :2 (UNLOCK)

Na het ingeven van dit kommando vraagt de DOS om het hoofd wachtwoord voor de diskette van drive 2.
Na ingave van dit wachtwoord zullen alle wachtwoorden van de diskette worden verwijderd.

PROT :2 (LOCK)

U krijgt de vraag tot het ingeven van een nieuw wachtwoord. Dit wachtwoord zal worden toegevoegd aan alle zgn. USERFILES. Een USERFILE is een niet bij de SYSTEM files behorende file.

Indien een file voorzien is van een wachtwoord, dan zal dit in de DIRECTORY worden aangegeven door middel van een P (Protected) achter de naam.

RENAME **DOS**

Met het RENAME kommando geeft u een nieuwe naam aan een bestaande file. Als een file een wachtwoord bevat, wordt dit niet meegekopieerd naar de nieuwe file.
Voorbeeld:

RENAME TESTFILE/BAS TO NWFILE/BAS

Dit verandert de naam TESTFILE/BAS in NWFILE/BAS.

TRACE **DISK**

Na het uitvoeren van dit kommando zal de inhoud van het PC-register (de Program-Counter) afgebeeld worden in de rechter bovenhoek van het beeldscherm. De informatie wordt iedere 8 milliseconden vernieuwd.
Voorbeeld:

TRACE

TRACE (OFF)

BACKUP **DOS**

Een der **belangrijkste,** goede gewoonten bij het gebruik van een computersysteem is **het maken van kopieen van uw diskettes.** Als u een programma gemaakt heeft, dan moet u **minimaal 3 diskettes** reserveren voor dit programma.

1 Diskette om - desnoods in een kluis - veilig op te bergen.
1 Diskette om te gebruiken.
1 Diskette als reserve.

Mocht om een of andere reden uw gebruiks-diskette niet meer funktioneren, dan kunt u de reserve diskette proberen. Mocht deze ook niet werken, dan zou het kunnen zijn dat uw diskdrive niet meer funktioneert. Of dat uw DOS "kapot" is. Een niet goed funktionerende DOS kan beide diskettes "vernield" hebben.Maar gelukkig, u bent een goed programmeur. U heeft nog een diskette ergens veilig opgeborgen.
Na reparatie van DOS of Diskdrive kopieert u weer lachend de twee kopieen die u nodig heeft voor gebruik.

Bovenstaand "verhaal" is niet overdreven. Het is gebeurd dat gebruikers 3 tot 4 maanden werk verloren zagen gaan door onzorgvuldig te werk te gaan. Door geen extra kopieen te hebben. Uit "zuinigheids overwegingen".
Maak dus altijd voldoende kopieen van uw programma's op aparte diskettes.

Voor eigen gebruik mag u programma's, DOS'sen en andere informatie die op een gekochte diskette staat zoveel kopieren als u wilt.
Iedere kopie is weer een op zichzelf staand origineel, niet te onderscheiden van het echte origineel.

Het kopieren gaat als volgt:

Doe een diskette met TRS-DOS in de eerste diskdrive (0), links.

Plaats een blanko - nieuwe - diskette in drive 1, rechts.

Type nu:

BACKUP

Het BACKUP programma vraagt u nu:

BACKUP DATE (MM/DD/YY) ? u tijpt 03/01/83

(INSERT SOURCE DISK)

Plaats nu de te kopieren diskette in drive nul(0), of, als u de TRS-DOS diskette wilt kopieren druk dan alleen op ENTER.

Het kopieren gaat nu geheel automatisch.

LET OP - Gebruikers van het NEWDOS-80.2 programma dienen niet het BACKUP kommando aan te roepen maar inplaats daarvan de volgende tekst te tijpen:

COPY 0 1

Newdos vraagt dan:

FORMAT DISKETTE Y/N u antwoordt Y

ARE SYSTEM DISKETTE AND SOURCE DISKETTE THE SAME Y/N

Als u de Systeem diskette wilt kopieren drukt u hier de Y-toets in. Anders drukt u de N-toets in. Newdos vraagt u dan om de SOURCE DISKETTE (de te kopieren diskette).

Nu vraagt Newdos:

PRESS ENTER IF DESTINATION DISKETTE IS IN DRIVE 1

U deed reeds een nieuwe diskette in drive 1, dus u drukt op ENTER.

Als Newdos gereed is volgt, zoals bij TRS-DOS de boodschap

DOS READY.

Zie voor de zeer uitgebreide mogelijkheden van het COPY kommando van NEWDOS-80.2 het bijbehorend manual in de engelse taal.

FORMAT **DOS**

Met het FORMAT programma initialiseert u een diskette in sporen en sektoren. Tevens wordt de DIRECTORY aangebracht en enkele SYSTEM files, nodig voor het funktioneren van de schijf als data-diskette.

Een standaard TRS-DOS diskette (dus met BASIC etc. erop) heeft aan vrije ruimte 44 GRANULES oftewel 55 Kbytes.
Een Datadiskette heeft aan vrije ruimte 67 granules oftewel 83.75 Kbytes.
De volgens TRS-DOS geFORMATte diskettes zijn ingedeeld in 35 sporen. De volgens NEWDOS-80.2 geFORMATte diskettes zijn ingedeeld in 40 sporen. NEWDOS-80.2 is compatible met TRS-DOS diskettes. Dat wil zeggen, met NEWDOS-80.2 kunt u wel met TRS-DOS gemaakte (data)diskettes gebruiken, andersom niet.
De vijf extra sporen die NEWDOS-80.2 gebruikt worden namelijk meer naar het middelpunt van de diskette geschreven. De eerste 35 sporen zijn op beide formaten op exact dezelfde afstand van het middelpunt aangebracht. Tegenwoordig kan iedere diskdrive en iedere diskette deze extra 5 sporen ook fysiek aan. Bij Radio Shack heeft men destijds gekozen voor 35-track (sporen) vanwege de toen beschikbare soort diskdrives. Inmiddels zijn vrijwel alle TRS-DOS-achtige Disk Operating Systemen overgegaan op 40-track Disk Systemen met de mogelijkheid om 35-track te lezen.

FORMAT is dus de allereerste schrijfaktie op een blanko diskette. Het kommando is inbegrepen als optie in het BACKUP kommando van TRS-DOS en in het COPY kommando van NEWDOS-80.2

De kommando volgorde voor TRS-DOS

Zoals reeds vermeld zijn de handelingen voor TRS-DOS en NEWDOS-80.2 iets afwijkend. De volledige TRS-DOS handeling wordt hieronder weergegeven, de NEWDOS-80.2 handeling voorzover zij overeenkomt met TRS-DOS.
Voor het FORMATten doet u een TRS-DOS diskette (waar BASIC enz. op staat) in drive 0 (de linker drive bij de ASTER CT-80) en een nieuwe, blanko diskette in drive 1 (de rechter diskdrive). **LET OP:** De diskette wordt met het etiket naar boven en de langwerpige sleuf naar achteren in de diskdrive gedaan.
Nu tijpt u:

FORMAT ENTER

De ASTER CT-80 antwoordt met:

WHICH DRIVE IS TO BE USED ? **1** ENTER
DISKETTE NAME ? **ASTERDAT** ENTER
CREATION DATE (MM/DD/YY) ? **03/01/83** ENTER
MASTER PASSWORD ? **PASSWORD** ENTER
DO YOU WANT TO LOCK OUT ANY TRACKS? **Y** ENTER
WHICH TRACKS (0-34)? **2,4-6** ENTER
FORMAT THE LOCKED-OUT TRACKS? **N** ENTER

Bovenstaand zijn alle mogelijkheden weergegeven die u met het formatten bij TRS-DOS kunt tegenkomen.
De vet-gedrukte tekst is het antwoord dat u geeft, gevolgd door ENTER.
We zullen vraag voor vraag doorlopen.

WHICH DRIVE vraagt om welke diskdrive. Ingegeven werd nummer 1, want 0 was immers de linker drive, 1 de rechter drive.

DISKETTE NAME vraagt om de naam van de diskette, maximaal 8 letters.

CREATION DATE vraagt voor invoer van de datum in het bekende maand, dag, jaar, formaat.

Bij **MASTER PASSWORD** voert u het hoofd wachtwoord in.

DO YOU WANT TO LOCK OUT ANY TRACKS is een vraag die u in staat stelt diskettes met een foutje erop toch toe te passen. Als tijdens formatten een fout wordt ontdekt, dan zal het FORMAT programma dit aangeven. Bij een nieuwe poging kunt u dan die sporen uitsluiten. Dit resulteert in een wat mindere kapaciteit. Dit is alleen mogelijk bij Data-diskettes en dan nog alleen op andere tracks dan track 0 en track 17. Hierop staat namelijk de SYSTEM informatie en de DIRECTORY.

Het beste is eigenlijk deze diskettes niet te gebruiken doch te retourneren aan de fabrikant indien ze fouten vertonen direkt bij het uit de doos halen.

Als u de vraag over het uitsluiten positief beantwoordt, krijgt u de vraag WHICH TRACKS. Bij het negatief beantwoorden van de vorige vraag wordt deze vraag overgeslagen.

Tevens wordt bij het positief beantwoorden van de LOCK OUT vraag invoer verlangd of de uitgesloten tracks toch geFORMAT moeten worden. Deze vraag moet u altijd met N beantwoorden. U kunt die tracks toch niet gebruiken.

Als alle vragen beantwoord zijn zal het TRS-DOS FORMAT programma zijn werk doen en een geslaagde FORMAT beantwoorden met:

DOS READY

Indien de diskette reeds DATA bevat, geeft het programma de melding:

DISKETTE CONTAINS DATA, FORMAT OR NOT?

Als u de vraag positief beantwoordt (met Y) dan zal de diskette alsnog geFORMAT worden.
Als u dit niet wilt, dan drukt u gewoon op de ENTER toets.

De kommando volgorde voor NEWDOS-80.2

NEWDOS-80.2 is in een aantal zaken veel uitgebreider. De wijze van het invoeren van kommando's is echter veelal eenvoudiger. Als we bovenstaand verhaal beschouwen voor TRS-DOS dan kunnen we bij NEWDOS-80.2 zeer kort zijn.
Stel u wilt een diskette in drive 1 formatten (de rechter diskdrive). U zorgt dat in de linker diskdrive (0) een NEWDOS-80.2 diskette met het FORMAT programma aanwezig is.
U tijpt nu:

FORMAT 1

Dit is eigenlijk alles.
De meest eenvoudige vorm van het formatten. Er worden geen vragen gesteld. De diskettenaam zal worden NONAME, de datum zal worden overgenomen uit de datum die u bij het opstarten van de ASTER CT-80 heeft ingegeven. Mocht u geen datum hebben ingevoerd, dan zal de datum 00/00/00 worden ingevoerd.
NEWDOS-80.2 zal over eventueel bestaande data op de diskette "heenFORMATten". Let dus wel op

wat u doet. Eventueel defekte tracks zullen worden opgemerkt. U krijgt dan de vraag of u:

- het nog eens wilt proberen,
- of u gewoon door wilt gaan,
- of u wilt stoppen (cancelen)

Als alles goed verloopt krijgt u de melding:

DOS READY

OPMERKING: *Mocht bij het formatten iets niet goed gaan, zeker bij het gebruik van TRS-DOS dan kan het nodig zijn dat u de diskette buiten de computer moet schoonwissen. Dit schoonwissen kunt u doen met een magneet. Een goede magneet hiervoor is het type dat u in een aquariumhandel koopt om het glas van een aquarium schoon te wissen.*

WEES UITERST VOORZICHTIG MET MAGNETEN EN DISKETTES. EEN MAGNEET IN DE BUURT VAN UW DISKETTE HEEFT ONHERSTELBARE GEVOLGEN.
Een goede tip: bevestig de magneet ergens aan de muur, ver van de plaats waar u eventueel ooit een diskette zou kunnen neerleggen. Bevestig hem zo, dat u echt moet opstaan en de diskette in uw hand moet nemen en er naar toe moet brengen.

Het wissen zelf is eenvoudig. U beweegt de achterzijde van de diskette (de voorzijde bevat het etiket) over de magneet, er net vrij van, zodanig dat de magneet het gehele oppervlak van de diskette eenmaal "gezien" heeft. U doet dit met rustige snelheid. De benodigde tijd is ongeveer 4 tot 6 sekonden. Daarna is de diskette volledig schoon. Werkelijk alle informatie is verdwenen. Mocht daarna een FORMAT of COPY aktie niet succesvol zijn, dan kunt u er vanuitgaan dat de diskette niet meer bruikbaar is.

TAPEDISK DOS

Met dit kommando schrijft u SYSTEM tapes over naar diskette:
Niet alle tapes kunnen worden overgezet naar Disk. De meeste Radio Shack TRS-80 Tapes voor LEVEL II zullen niet funktioneren, terwijl een programma dat laadt beneden adres 21748 decimaal (54F4 HEX) ook niet funktioneert. Het LMOFFSET programma van NEWDOS-80.2 geeft wat meer mogelijkheden. Raadpleeg hiervoor het NEWDOS-80.2 Manual.
U start als volgt:

TAPEDISK

Het programma meldt zich met een vraagteken.

?

Als de cassetterecorder met de tape gereed staat tijpt u:

C

Het programma wordt nu geladen.
Als het vraagteken weer komt is de tape geladen. Nu moet u weten welke naam en wat de adressen zijn waarop het begin, eind en ingangspunt (entrypoint) zich bevinden.

U heeft bijvoorbeeld een programma STARWARS,U tijpt dan:

F STARWARS/CMD:1 6000 6FFF 6203

Het programma STARWARS met als beginlokatie in het geheugen 6000 (HEX) en als eindlokatie 6FFF (HEX) en als entrypoint 6203 (HEX), wordt nu op disk gezet. Bij het later "draaien" van het programma weet de DOS automatisch waar het programma geladen moet worden in het geheugen en wat het startpunt is.

Nu geeft u als kommando:

E

Dat zoveel betekent als Exit naar TRS-DOS.

DISKDUMP/BAS **DISKBASIC**

Met dit programma kunt u diskfiles kontroleren door ze te laten afdrukken op de beeldbuis of op de printer.
Het programma draait onder BASIC en wordt normaal als een BASIC programma geladen.
Als u geen afdruk op de printer wilt hebben, maar op de beeldbuis dan moet u de volgende regels wijzigen.
Regels 170, 240 en 250 alle LPRINT's in PRINT. En regel 160 in:

160 GET1,SN

Na het opstarten vraagt het programma welke file u wilt zien, u tijpt dan de volledige filenaam in. Dan wordt gevraagd welke sector van de file. Als u alles wilt zien geeft u gewoon ENTER en voor iedere volgende sector ook ENTER.

De afdruk is als volgt opgebouwd:
Vooraan het byte-nummer van de file, dan een groep van 8 Hexadecimale bytes, daarna weer 8 Hexadecimale bytes, dan - tussen twee uitroeptekens - de 16 bytes in ASCII vorm. Daar het programma geen kleine letters kent zullen de kleine letters en niet-afdrukbare codes als punten worden afgedrukt.

GETTAPE/BAS en GETDISK/BAS **DISKBASIC**

Deze twee routines maken het mogelijk files, in wat voor vorm ook van TAPE naar DISK en vice versa te kopieren. U moet alleen oppassen als u een sequentiele file van tape naar disk overzet dat de EOF (End Of File pointer) niet altijd juist overkomt. Kontroleer daarom na het overschrijven van sequentiele files eerst of u een goed EOF kunt uitvoeren. Is dit niet mogelijk dan behoeft u slechts de file eenmaal onder Disk Basic in te lezen en weer opnieuw weg te schrijven. Is dit niet mogelijk dan zult u in het programma dat de file leest een beveiliging moeten inbouwen tegen dit euvel.
Het programma is interaktief en vraagt de gebruiker de nodige handelingen te verrichten.

Onder DISK BASIC tijpt u:

RUN "GETDISK/BAS"

of

RUN "GETTAPE/BAS"

TRS-DOS FOUTCODES

Foutnummer	Mogelijke oorzaak (zie uitleg)	Fout beschrijving
00	-	Geen fout
01	MD	Pariteits fout bij lezen van tape-aanvangsdata
02	D	Zoekfout bij lezen
03	XK	Data onjuist bij lezen
04	MB	Pariteits fout bij lezen
05 06	FMD	Data record niet gevonden bij lezen
07	P	Poging tot lezen van SYSTEM data record
08	P	Poging tot lezen van SYSTEM data record
09	UP	Apparatuur niet aanwezig (bijv. diskdrive nr. 2)
10	MD	Pariteitsfout gedurende schrijven op tape van aanvangsdata
11	D	Zoekfout tijdens schrijven
12	XC	Data onjuist tijdens schrijven
13	MD	Pariteitsfout tijdens schrijven
14	FMD	Data record niet gevonden tijdens schrijven
15	XD	Schrijffout op diskette
16	UXD	Diskette beschermd (met sticker afgeplakt hoekje)
17	PS	Onjuist logisch file getal (slecht dcb)
18	MPDS	DIRectory leesfout
19	MPDS	DIRectory schrijffout
20	UP	Onjuiste file naam (slecht dcb)
21	MPDS	GAT leesfout (Granule Allocation Table)
22	MPDS	GAT schrijffout
23	MPDS	HIT leesfout (Has Index Table)
24	MPDS	HIT schrijffout
25	UP	File niet in DIRectory
26	UP	File toegang geweigerd (onjuist wachtwoord)
27	UP	DIRectory ruimte vol (maximaal 48 files)
28	UP	Disk ruimte vol (70 granules max. bij TRS-DOS)
29	P	EOF aangetroffen (End Of File - Einde file)
30	P	NRF geen record gevonden, buiten file bereik (No Record Found)
31	UP	Volle DIRectory, file kan niet verlengd worden.
32	UP	Programma niet gevonden
33	UP	Onjuist drive nummer aangegeven.
34	UP	Onvoldoende ruimte voor nieuw apparaat aanwezig
35	MPUS	Load file formaatfout, geen programma
36	XCS	Geheugenfout
37	PUXC	Poging tot laden (schrijven) in ROM geheugen
38	P	Toegang tot beschermde file wordt geweigerd
39-62	UP	De file is niet geOPENd
63	P	Nog niet gedefinieerde foutcodes Onbekende fout opgetreden.

Uitleg van mogelijke fout oorzaken

C=Fout in CPU (Z80) van ASTER CT-80
D=Diskdrive fout
F=Diskette niet geFORMAT
M=Foutieve diskette
P=Fout in uw programma
S=TRS-DOS fout, opnieuw starten
U=U volgde een onjuiste procedure
X=Interne fout.

Bij veel soorten programmatuur is het gemakkelijk om direkt Hexadecimaal of Octaal gebaseerde getallen toe te passen. Dit is vooral toepasselijk bij het aanroepen van geheugenadressen, USR-routines enz. enz.
Voorbeeld:

```
10 POKE&H51FE,41
20 POKE 20990,65
```

Exact dezelfde waarden worden in dezelfde adressen gePOKEd.

```
30 PRINT&H41
```

Regel 30 zal als resultaat het getal 65 geven.

```
40 PRINT&O DB
```

Regel 40 zal als resultaat 333 geven, de decimale waarde van het octale getal DB.

LINEINPUT **DISKBASIC**

In DISKBASIC zijn bepaalde tekens "verboden" toe te passen bij een INPUT statement. Hieronder vallen de volgende tekens:
, - de komma
; - de punt-komma
: - de dubbele punt
" - de aanhalingstekens oftewel quotes.

Al deze tekens zijn zgn. "scheidings tekens" oftewel "delimiters".
Ze worden in BASIC gebruikt om bepaalde funkties aan te geven of om bepaalde statements van elkaar te scheiden. Toch kan het noodzakelijk zijn deze tekens in tekst te gebruiken. Met het LINEINPUT statement kunt u een string invoeren waar al deze tekens als normale karakters worden beschouwd. Ook kunt u eenzelfde string naar disk schrijven en later van disk teruglezen met het LINEINPUT statement. Slechts de ENTER wordt als "delimiter" beschouwd. Ook verschilt het invoeren met de LINEINPUT iets van het invoeren van een string met INPUT. INPUT geeft een zgn. prompt, het vraagteken ('?'). Een LINEINPUT statement geeft geen prompt, is dus uitstekend toe te passen als in een programma tekst ingevoerd moet worden.
Voorbeeld:

```
10 CLEAR 300:CLS:PRINT"ASTER CT-80 ETIKETTEN PROGRAMMA"
20 PRINT STRING$(60,191)
30 A$="":B$=A$:C$=A$:PRINT
40 PRINT"Geef maximaal 3 regels tekst, eindig met de ENTER-toets'
50 PRINT
60 LINEINPUT"Eerste regel ";A$
70 PRINT
80 LINEINPUT"Tweede regel ";B$
90 PRINT
100 LINEINPUT"Derde regel ";C$
110 INPUT"Hoeveel regels van rand etiket tot rand etiket ";R
120 INPUT"Hoeveel etiketten wenst u ";N
130 INPUT"Staat de printer klaar ? Druk dan op ENTER";A
```

```
140 FOR X=1 TO N
150 FOR A=1 TO 3:LPRINT":NEXT A
160 LPRINT A$
170 LPRINT B$
180 LPRINT C$
190 FOR B=1 TO R-6:LPRINT":NEXT B
200 NEXT X:GOTO 10
```

Met bovenstaand programma kunt u op een met een doos enkelrij zelfklevende etiketten eenvoudig meerdere etiketten maken.
Toegepast wordt het LINEINPUT statement.
Mocht uw etiket 4 of meer regels moeten bevatten, verander dan de FOR NEXT herhaling die de extra regel-opschuivingen maakt en voeg na C$ meerdere strings toe.