Cursus

ZX-Spectrum

versie: maart, 1985

Hoofdstuk 1: Beeldscherm I/O

H. 1 Beeldscherm I/O.

Inleiding.

Het beeldscherm is verdeeld in twee stukken. Het bovenste deel, regels 0 t/m 21, is voor de eigenlijke uitvoer en het onderste deel, regels 22 en 23, is voor berichtgeving en invoer. U kunt zelf ook berichten naar deze regels sturen.

Verder is het beeldscherm vertikaal in 32 kolommen, 0 t/m 31, verdeeld en in twee subkolommen, die beginnen op kolom 0 en kolom 16 en elk 16 kolommen breed zijn.

Elk karakter is opgebouwd uit pixels (= puntjes). Dit zijn bits die 1 (zwart) of 0 zijn. Het beeldscherm bevat 176 pixels vertikaal (0 t/m 175) en 256 pixels horizontaal (0 t/m 255). Zie verder bij §5.

Indien u elk voorbeeld met een schoon scherm wilt beginnen, moet u eerst CLS (K,v) intoetsen. Dit betekent CLear Screen.

§1. Uitvoer naar het bovenste deel.

Voor de uitvoer naar dit deel beschikken we over het statement PRINT (K,p). Hierbij geldt dat het regelnummer r niet groter mag zijn dan 21. Grafische uitvoer word in een later stadium besproken.

Gebruiken we PRINT alleen, dan heeft dit tot resultaat dat er een Line Feed (LF) en een Carriage Return (CR) plaatsvind. Er wordt dus NIETS geprint.

Willen we nu de uitvoer sturen, dwz. daar laten beginnen waar wij het willen, dan beschikken we over de volgende besturings functies:

- AT r,k (SS+i)
- TAB k (E,p)
- de komma (SS+n)
- de puntkomma (SS+o)
- het apostrof (SS+7)

Hier staat r voor regel en k voor kolom.

AT r,k

Met AT (=op) kunnen we de uitvoer op een zelf gekozen regel en een zelfgekozen kolom van die regel laten beginnen.
Na AT r,k MOET een puntkomma volgen.

Voorbeeld: PRINT AT 2,3;"3e regel, 4e kolom"

Stel dat we een regel willen printen op regelnummer 21, 22e regel, en deze zou doorlopen op regel 22. Dat zal niet gebeuren, maar er zal 'Scroll ?' worden gevraagd. Deze vraag kunt u beantwoorden met 'nee' door 'n', 'BREAK' of 'STOP' (SS+a) in te toetsen. Alle andere toetsen zijn 'ja'.
Dit gegeven geldt voor alle stuur functies en LIST.
U kunt deze boodschap onderdrukken door: POKE 23692,255.

TAB k

Met TAB (van tabulator) kunnen we de uitvoer op een bepaalde kolom laten beginnen. De regel waarop de uitvoer komt te staan kunnen we hiermee dus niet kiezen. Dit zal altijd de volgende regel zijn. Na TAB k MOET een puntkomma volgen.

```
Voorbeeld:  PRINT TAB 3;"?e regel, 4e kolom"
```

De komma.

Met de komma springen we naar het begin van een subkolom, dus naar kolom 0 of kolom 16, afhankelijk van het laatste PRINT statement.

```
Voorbeeld: PRINT ,"Start in het",,"midden."

Dit is hetzelfde als:  PRINT TAB 16;"Start in het"
                       PRINT TAB 16;"midden"
```

Het is alleen veel goedkoper, want een komma verbruikt slechts 1 byte.

De puntkomma.

Hiermee onderdrukken we een LF en een CR. Dat wil zeggen dat de volgende regel direct achter de laatst geprinte regel begint. Hiermee kunnen we dus mooi variabelen tussen strings printen.

```
Voorbeeld:  LET a=10: PRINT "a heeft de waarde ";a

            PRINT "een regel";
            PRINT " en nog steeds die regel"
```

Het apostrof.

Met het apostrof (') vervangen we PRINT. Dus we gaan, net als met PRINT naar het begin van de volgende regel. Een LF en een CR dus. Het voordeel is dat u niet steeds een PRINT statement hoeft te geven.

```
Voorbeeld:  PRINT "een regel"'"en de volgende regel"

Dit is hetzelfde als:  PRINT "een regel"
                       PRINT "en de volgende regel"
```

§2. Uitvoer naar het onderste deel.

We kunnen zelf, in een programma, berichten sturen naar het onderste deel van het beeldscherm. Dit heeft als voordeel dat u het bovenste deel geheel kunt gebruiken voor schermen, plaatjes, en dergelijke.

Om naar dit deel te kunnen printen, laten we het PRINT statement direct volgen door #1 of #0 (# = SS+3) en een van de volgende stuur functies: de komma, de puntkomma of het apostrof. Hierna kunt alle stuur functies normaal gebruiken.

```
Voorbeeld:  PRINT #1;  of
            PRINT #1,  of
            PRINT #1'
```

U kunt hier NIET zeggen: PRINT #1;AT 22,? want u kan deze regels niet op deze manier benaderen. U kunt AT wel gebruiken.

```
Voorbeeld:  PRINT #1;AT 0,0;"tekst"
```

De computer stelt in het onderste deel allereerst slechts twee regels tot beschikking en zal dit aantal vergroten, indien nodig, zonder eerst 'Scroll ?' te vragen. Verder neemt de computer aan dat regel 22 de eerste regel is, dus regelnummer 0. In bovenstaand voorbeeld zal de string 'tekst' dus op regel 22 worden geprint. Alle andere regelnummers die u ingeeft zullen geinterpreteerd worden als 1. Regel 23 is voor de computer dus hetzelfde als 1.

```
Voorbeeld:  PRINT #1;AT 1,0;"tekst"
            PRINT #1;AT 18,0;"tekst"
```

Kolommen en subkolommen blijven hetzelfde. Ook kunt alle kleur functies (flash, inverse, enz.) normaal blijven gebruiken.

Een volgend PRINT statement zal de vorige tekst NIET wissen, maar omhoog doen scrollen. Om dit te voorkomen zet u voor elk PRINT statement een INPUT statement, gevolgd door een puntkomma en verder niets.

```
Voorbeeld:  INPUT;:PRINT #1;"tekst"
```

Het INPUT statement 'restaureert' het onderste deel, het brengt het terug in de oorspronkelijke toestand: leeg.

In plaats van INPUT; gevolgd door PRINT #1 kunt u ook alleen INPUT gebruiken.

```
Voorbeeld:  INPUT "tekst";
```

Ook met INPUT kunt alle stuur- en kleur functies gebruiken.

```
Voorbeeld:  INPUT,"tekst"     (begint op kolom 16 = subkolom 2)
            INPUT'"tekst"     (scrolled naar regel 22)
            INPUT AT 0,0;"tekst"
            INPUT FLASH 1;"tekst"
            enz.
```

Om variabelen te printen, kunt u natuurlijk PRINT gebruiken, maar het is soms noodzakelijk om bij invoer een variabele te laten zien. Deze plaatst u dan tussen haakjes.

```
Voorbeeld:  LET a=10: PRINT #1;"a heeft de waarde ";a
            LET a=12: INPUT "Voer het ";(a);"e getal in: ";b
```

De computer zal (a) niet zien als de variabele waar het 12e getal in moet komen te staan, maar zal a printen en doorgaan naar b. De variabele b staat niet tussen haakjes en dus is dat de variabele waar het 12e getal in komt te staan.

§3. Lezen van het beeldscherm.

We kunnen alleen lezen van het bovenste deel van het beeldscherm. Dit doen we met de functie SCREEN$. Hiermee kunnen we ALLEEN tekst lezen en geen grafische karakters. We kunnen ook geen hele regel in een keer lezen, slechts enkele karakters. De functie heeft r en k als variabelen en geeft het karakter terug, welke op regel r en kolom k van die regel op het beeldscherm staat.

```
Voorbeeld:  PRINT AT 2,3;"a"
            PRINT SCREEN$ (2,3)
```

Aan het begin van de volgende regel zal een a worden geprint. Ook kunnen we het gelezen karakter aan een variabele toekennen.

```
Voorbeeld:  PRINT AT 5,11;"H"
            LET a$= SCREEN$ (5,11)
            PRINT "Op regel 5, kolom 11 staat een ";a$
```

Dit is handig als het karakter dezelfde kleur heeft als de achtergrond, als u het niet ziet.
Zo kunt u ook een hele regel lezen. Dit moet dan wel in een lus gebeuren.

```
Voorbeeld:  PRINT AT 2,0;"Deze regel is 32 karakters lang."
            LET a$=""
            FOR i=0 TO 31
            LET a$=a$+ SCREEN$ (2,i)
            NEXT i
            PRINT a$
```

Het statement LET a$="" is noodzakelijk, omdat de computer deze variabele niet eerder heeft gezien en, ook al is dat wel het geval, maakt a$ leeg.

§4. Het gebruik van de attributen.

De functies kunnen alleen in extended mode (E) worden verkregen. We kennen de volgende functies:

```
INK 0 t/m 7       (E,SS+x)
PAPER 0 t/m 7     (E,SS+c)
BORDER 0 t/m 7    (K,b)

FLASH 1/0         (E,SS+v)
BRIGHT 1/0        (E,SS+b)
OVER 1/0          (E,SS+n)
INVERSE 1/0       (E,SS+m)
```

De kleuren welke we mee willen geven, moeten gekozen worden uit de cijfers 0 t/m 7. Zie de bovenste rij van het toetsenbord. Boven de toetsen staat de kleur aangegeven. Deze worden alleen gebruikt met INK, PAPER en BORDER.

De andere vier functies kunt u alleen aan (1) of uit (0) zetten. Al deze functies, behalve BORDER (=kommando), dienen, binnen een PRINT statement, gevolgd te worden door een komma, puntkomma of het apostrof. Daarna mag u AT en TAB gebruiken en er uitvoer achter plaatsen. U kunt ze ook alleen gebruiken. Als kommando. Dan geldt het kommando voor alles.

Voorbeeld:
```
PRINT INK 1; PAPER 6;"de inktkleur is blauw en";
PRINT " de 'papier'kleur is geel"

PRINT BRIGHT 1; FLASH 1;"dit is helder en gaat";
PRINT " aan en uit."

PRINT AT 0,0;"dit is leesbaar en"
PRINT AT 0,0; OVER 1;"deze maar half."
PRINT AT 2,3;"onderstreepte tekst."
PRINT AT 2,2; OVER 1;"_____"
```

In deze voorbeelden worden de functies binnen een PRINT statement gebruikt. In het eerste en tweede deel ziet u dat door het gebruik van een puntkomma de attributen mee worden genomen naar het volgende PRINT statement. Dit gebeurt ook als u de puntkomma zou vervangen door een komma of het apostrof. Toets de volgende regels maar eens in.

Voorbeeld:
```
PRINT PAPER 1; INK 7; BRIGHT 1;"Ziet u?",,
PRINT BRIGHT 1; FLASH 1;"En deze dan...."'''
```

Wanneer u alle uitvoer in een bepaalde inktkleur of papierkleur (of 'flitsen' of helder, enz.) wilt hebben, voert u de functies als kommando's in. Dat kan vanuit een programma, maar ook direct, maar dan moet u wel, behalve bij BORDER en INK, twee keer op ENTER drukken. Doet u het slechts 1x, dan zal alleen dat wat wordt geprint in de gewenste kleuren verschijnen. Doet het vanuit een programma, geef dan als laatste een CLS in de regel.

Voorbeeld:
```
PAPER 6 (enter)
PRINT "alleen deze tekst is geel" (enter)
PAPER 6 (enter, enter)

Nu is het hele scherm geel, behalve de rand.
Dit doen we met BORDER.

BORDER 6 (enter)            Nu is de rand ook geel.

PAPER 7: BORDER 7: CLS      Nu is alles weer normaal.
```

U kunt al deze functies door elkaar gebruiken. Als u ze als kommando binnen een programma gebruikt, doet u precies hetzelfde als bij alle andere kommando's.

Nu volgt er iets dat moeilijk? te onthouden is. U kunt namelijk de uitvoer de attributen meegeven zonder FLASH, INK, enz. in te toetsen.
Hier volgt de uitleg van de truc. Tik alles in zoals het er staat, met die uitzondering dat dat wat tussen () staat, moet worden gelezen als zijnde opdrachten.

Voorbeeld:
```
PRINT "(E,1)(E,CS+7) Nu is de inkt wit en het papier
blauw",,

PRINT "(E,9)(E,CS+9) Dit is helder en het flitst en
(E,8)(E,CS+8)dit is weer gewoon."
```

Leuk, nietwaar? En het bespaart bytes. Houdt er wel rekening mee dat om dit 'aan' te zetten, u het tussen aanhalingstekens moet plaatsen, anders gebeurt er niets. Uit zetten hoeft niet tussen aanhalingstekens.

Voorbeeld:
```
LET a$="Dit wordt wit met een rode achtergrond."
PRINT "(E,2)(E,CS+7)";a$
LET b$="(E,2)(E,CS+7)": PRINT b$;a$
```

Zo kunt u deze codes dus ook in een variabele plaatsen. Om de oorspronkelijke kleuren weer terug te krijgen, zet u de kleuren op dezelfde manier.

Voorbeeld:
```
LET a$="Een regel met tekst."
LET b$="Een volgende regel."
LET c$="(E,1)(E,CS+7)": LET d$="(E,7)(E,CS+1)"

PRINT c$;a$'d$;b$
hier is de eerste regel blauw met witte inkt en de
tweede wit met blauwe inkt.
```

De variabelen c$ en d$ kan u door het hele programma gebruiken. Ze werken alleen met een ; en een , en niet met de '.

§5. Tekenen.

Voor het tekenen van lijnen, cirkels en krommen maken we gebruik van de volgende kommando's:

- PLOT (K,q)
- DRAW (K,w)
- CIRCLE (E,SS+h)

PLOT x,y

Bij het tekenen maken we gebruik van de pixels. Dat wil zeggen dat de kolommen niet meer gelden. Zoals in het begin al is gezegd, is het beeldscherm opgebouwd uit pixels. Dit is de basis. Zonder deze pixels kunnen er geen karakters op het beeldscherm verschijnen, daar deze ook uit pixels bestaan. Er zijn 256 x 176 pixels. Horinzontaal van 0 t/m 255 en verticaal van 0 t/m 175.

Het benaderen van een pixel werkt hetzelfde als in twee-dimensionaal assenstelsel. Een X-as (hor) en een Y-as (ver) met de oorsprong in de linker benedenhoek (altijd coord. 0,0). We gebruiken hier het kommando PLOT x,y voor, waarbij x een punt is van de horizontale as (0 - 255) en y een punt van de verticale as (0 - 175).

Voorbeeld: PLOT 10,10

Na dit kommando ziet u dat de pixel op coordinaat 10,10 zwart is geworden. Voor de computer is dit de nieuwe oorsprong (zie DRAW). Het middelpunt van het scherm is coordinaat 128,88.

DRAW x,y
DRAW x,y,r

Met het kommando DRAW kunnen we een rechte of een kromme tekenen. Het beginpunt is 0,0 en is met PLOT te verplaatsen. De richting van een rechte wordt NIET in graden meegegeven, maar in het aantal pixels naar rechts (x) en dan het aantal pixels omhoog (y). Indien dit aantal gelijk is, zal de lijn onder een hoek van 45 graden lopen. We kunnen x en y ook negatief maken, zodat de lijn de andere kant op gaat.

Voorbeeld: DRAW 10,10

Indien u vooraf geen pixel gePLOT heeft, zal deze lijn starten in de linker benedenhoek, anders vanaf het gePLOTte punt. Aangezien in dit voorbeeld x en y gelijk zijn, maakt deze lijn een hoek van 45 graden met beide assen.

Voorbeeld: PLOT 128,88: DRAW 10,0

In dit voorbeeld trekken we een horizontale lijn (y=0) naar rechts vanuit het middelpunt van het scherm.

Voorbeeld: DRAW 0,20

Deze lijn zal starten aan het eind van de vorige lijn, in verticale richting (x=0) omhoog. Hieruit blijkt dat wanneer u een lijn trekt vanuit een bepaald punt, het einde van die lijn altijd de nieuwe oorsprong wordt, van waaruit de volgende lijn zal beginnen. Als u zou willen dat de volgende lijn weer vanuit het bepaalde punt moet starten, moet u de oorsprong met een PLOT statement weer terug zetten.

```
Voorbeeld:  PLOT 128,88: DRAW 0,-20
            DRAW 20,0

            PLOT 128,88: DRAW 0,-20
            PLOT 128,88: DRAW 20,0
```

Indien de lijn die u wilt trekken buiten het scherm zal vallen, krijgt u vanzelf een foutboodschap.

We kunnen met het DRAW kommando ook een kromme tekenen. Hierbij zijn x en y een punt van de cirkel waar de kromme een deel van is en r het aantal radialen dat moet worden afgelegd. Een radiaal is een lengte eenheid van de cirkel. Een hele cirkel bestaat uit 2*PI radialen. Dus als r=PI dan zal er een halve cirkel worden getekend.

De kromme begint altijd op een gePLOT punt en eindigt op x,y. Zo geeft u dus de straal van de cirkel weer.

Voorbeeld: PLOT 100,100: DRAW 128,128,PI

In dit voorbeeld zal een halve cirkel worden getekend, die start op 100,100 en eindigt op 128,128. De diameter van deze cirkel is:

x=100, y=100, x'=128, y'=128

Deze punten zijn BEIDE punten van de cirkel.

```
(x'-x)^2 + (y'-y)^2 = (diameter)^2
(128-100)^2 + (128-100)^2 = d^2
(28)^2 + (28)^2 = d^2
784 + 784 = d^2
1568 = d^2
d = V1568 = 39,6
```

Dus de diameter van de cirkel is 39,6 PIXELS lang.
Ook hier geldt dat het punt 128,128 de oorsprong zal zijn van de volgende lijn of kromme.

CIRCLE x,y,s

Met het kommando CIRCLE tekenen we een cirkel met als middelpunt (x,y) en een straal s. De x en y in CIRCLE werken als PLOT. U hoeft dus niet eerst een punt te PLOTten.
Ook hier geldt weer dat als de cirkel is getekend, het laatste punt van die cirkel de oorsprong is voor de volgende lijn of kromme.

Voorbeeld: CIRCLE 128,88,50

In dit voorbeeld is het middelpunt van de cirkel coordinaat 128,88 en de straal is 50 PIXELS lang. De diameter is dus 100 PIXELS lang.

U mag bij bovenstaande kommando's de attribuut functies gewoon gebruiken. Let wel: FLASH, PAPER en BRIGHT gelden ALTIJD voor 8*8 pixels. En de functies dienen ALTIJD door een puntkomma gevolgd te worden.

```
Voorbeeld:  DRAW BRIGHT 1;50,50
            DRAW FLASH 1;0,50
            DRAW PAPER 2;50,0
            DRAW INK 2; BRIGHT 1;0,-50
```

§6. Lezen attributen en pixels.

Over het algemeen kunt u op het scherm zien in welke kleur de tekst op het beeldscherm staat en welke pixels INK of PAPER zijn. Maar een BASIC programma weet dit niet. Stel u schrijft een spelletje in BASIC, dan moet u weten wanneer het vliegtuig in een verboden zone komt. Als u slim bent, geeft u deze zone een andere kleur. U kunt dan de waarde lezen met de functie

ATTR (r,k)

waarbij r een regelnummer (0-21) en k een kolomnummer (0-32) is. De functie is een broertje van SCREEN$ (r,k).

```
Voorbeeld:  PRINT AT 10,11; PAPER 2;" "
            PRINT AT 21,0; ATTR (10,11);TAB 10; ATTR (10,12)
```

Om in een programma te weten of een bepaalde pixel INK of PAPER is, hebben we functie

POINT (x,y)

waarbij x een pixelnummer is van de horizontale as (0-255) en y een pixelnummer is van de verticale as (0-175). Deze functie is in weze een vergelijking die 0 geeft als de pixel PAPER is en 1 als de pixel INK is.

```
Voorbeeld:  PLOT 128,88              Pixel wordt INK.
            PRINT POINT (128,88)     Er wordt 1 geprint.
            PLOT OVER 1;128,88       Pixel wordt weer PAPER.
            PRINT POINT (128,88)     Er wordt 0 geprint.
```

§7. Overige kommando's.

CLS

Met dit kommando (CLear Screen) maken we het scherm leeg, zetten beginposities voor PRINT en DRAW weer op 0,0.

COPY

Met COPY sturen we het hele beeldscherm naar printer. Dus tekeningen kunt u zo printen.

LIST, LLIST en LPRINT

Met LIST stuurt u een listing van een BASIC programma naar het beeldscherm.
Met LLIST stuurt u de listing naar printer.

Voor LPRINT geldt hetzelfde als PRINT, maar nu naar printer.
Let wel: AT werkt als TAB.
Van de attribuut functies werkt alleen INVERSE.

§8. Beeldschermgeheugen.

Het geheugen van een computer bestaat uit bits. Elke acht bits vormen een byte. Om deze bytes individueel te kunnen benaderen hebben we ze genummerd. Zo'n nummer noemen we een adres. De ZX-Spectrum heeft een 64K RAM chip (=65536 bytes, nl. 0 t/m 65535) waarvan circa 24K wordt ingenomen door de monitor. U heeft ongeveer 41K tot uw beschikking.

Het beeldschermgeheugen maakt deel uit van de monitor en bestaat uit twee delen. Een deel voor de patronen, bijvoorbeeld tekst, en een deel voor de kleuren.

Het deel voor de patronen begint op adres 16384 en eindigt op adres 22527. Het deel voor de kleuren volgt hier direct op en begint derhalve op adres 22528 en eindigt op adres 23295.

We zullen steeds de linker bovenhoek bespreken en deze voorstellen met nullen en sterretjes ipv. enen. Dit voor de duidelijkheid.

De bytes liggen horixontaal. Dus u heeft 32 bytes horizontaal bij 192 bytes vertikaal (incl. regels 22 en 23) en dat zijn 6144 bytes voor het patroongeheugen. Voor de kleuren is er per acht bytes in vertikale richting 1 byte gereserveerd. Dus dit zijn er 32 kolommen x 24 regels = 768 bytes.

U kunt hier door middel van POKE waarden in zetten.

Voorbeeld: CLS : POKE 16384,255

U ziet links bovenin een streepje. Dit is de eerste byte van het beeldschermgeheugen en bevat de maximale waarde. Een byte kan slechts waarden bevatten van 0 <= n <= 255.

Voorbeeld: POKE 16385,1

Wat nu? U ziet dat in de tweede byte 1 bit (pixel) zwart is. Maar ook dat deze niet aansluit aan de eerste acht pixels. De oorzaak hiervan is dat men bij bytes de waarde, overigens in elk talstelsel, van rechts naar links representeert.

Voorbeeld: adres 16384 bevat: 11111111 = 255
adres 16385 bevat: 00000001 = 1

na elkaar geplaatst ziet dit er als volgt uit:

16384 16385
1111111100000001

POKE 16386,128 : 10000000

16384 16385 16383
111111110000000110000000

een byte zit als volgt in elkaar:

```
 1   1   1   1   1   1   1   1
 x   x   x   x   x   x   x   x
2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
 =   =   =   =   =   =   =   =
128+ 64+ 32+ 16+ 8 + 4 + 2 + 1 = 255.
```

Nu is dit tamelijk eenvoudig. In vertikale richting wordt het moeilijker. De bytes volgen elkaar niet netjes op. Dit kunt u goed zien bij het laden van een scherm, bijvoorbeeld bij een spel. Dan ziet u dat eerst de eerste rij bytes van de eerste 7 regels worden geladen. Vervolgens de tweede rij, enz. totdat de regels 0 t/m 7 vol zijn. Dan begint de computer aan de regels 8 t/m 15 en als deze vol zijn aan de laatste 7 regels, 16 t/m 23. Kleuren bespreken we nog.

Dus de byte op de vlogende rij is NIET 16384 + 32 = 16416, want dit is de eerste byte van de volgende REGEL.

Voorbeeld:

```
                          +1       +2
 REGEL 0.   16384 :  111111110000000110000000
                     0
                     0
                     0
                     0
                     0
                     0
                     0
 REGEL 1.   16416 :  000000000000000000000000
```

De eerste byte direct onder adres 16384 is : 16384 + 32*8 = 16384 + 256 = 16640

```
Voorbeeld:                      +1       +2
           16384 :  11111111 00000001 10000000
           16640 :  00000000 00000000 00000000

           POKE 16640,254

           16384 :  11111111 00000001 10000000
           16640 :  11111110 00000000 00000000

           POKE 16640+32x8,252

           16384 :  11111111 00000001 10000000
           16640 :  11111110 00000000 00000000
           16896 :  11111100 00000000 00000000
```

Wat betreft de kleuren laten we het bij het volgende schema, wat de waarden aangeeft voor 1 byte en waar deze voor geldt: Het adres n moet zijn: 22528 <= n <= 23295.

```
byte n  : bits 0,1 en 2 voor INK.
          bits 3,4 en 5 voor PAPER.
          bit  6        voor BRIGHT.
          bit  7        voor FLASH.
```

```
Voorbeeld:  POKE 22528,8+64  en
            PRINT AT 0,0;PAPER 1;BRIGHT 1;OVER 1;" "
            hebben hetzelfde resultaat.
```

Als u wel eens naar het nieuws voor doven en slechthorenden kijkt, weet u dat u nu ook zo'n klok kunt maken die zo naast het beeld loopt.

```
Voorbeeld:  FOR i=0 TO 21: PRINT AT i,0;"(G,CS+8)": NEXT i
            PRINT #1;AT 0,0;"(G,CS+8)"'"(G,CS+8)"
            FOR i=16384 TO 20480 STEP (8*256)
            FOR j=i TO (i+7*32) STEP 32
            FOR k=j TO (j+7*256) STEP 256
            POKE k,0: PAUSE 9: NEXT k: NEXT j: NEXT i
```

In de eerste r regels wordt op positie 0,0 van ELKE regel een zwart blokje geprint. De bits van de bytes zijn dan allemaal 1, dit wil zeggen dat de bytes de waarde 255 hebben. En in de laatste regel krijgen ze de waarde 0 (bits allemaal 0).

Hoofdstuk 2: Randapparatuur

- Inleiding.

§1. Toetsenbord.

§2. Cassettes.

§3. Microdrives.

§4. Streams.

H.2 Randapparatuur.

Inleiding.

Onder randappartuur verstaan we printers, beeldschermen, externe geheugens, toetsenborden, enz.
Voor de Spectrum zullen we magneetbandgeheugens, cassettes en microdrives, het toetsenbord en de printers bespreken.

§1. Het toetsenbord.

Invoer van data geschiedt normaal gesproken met INPUT. Hier wacht de computer tot u ENTER intoetst, alleen ENTER wordt niet geaccepteerd, en vervolgens plaatst de computer de data bij de correcte variabele.

```
Voorbeeld:  INPUT "Geef getal..";a
            INPUT "Geef tekst..";a$
```

In dit voorbeeld is 'a' een numerieke variabele, dwz. ALLEEN voor getallen, en 'a$' een alfanumerieke variabele, dwz. voor TEKST. U kunt dan wel getallen invoeren, maar u kan er niet mee rekenen. Zie verder het hoofdstuk BASIC.

Wat moeilijker maar interessanter is het LEZEN van het toetsenbord. De computer doet dit bij INPUT ed. Het lezen gaat per karakter. Dus geen hele regels in 1 keer, maar steeds 1 karakter.

We kunnen op vele manieren het toetsenbord lezen. In BASIC hebben we de functie INKEY$ (E,n). Deze functie leest wat u voor toets indrukt, maar er wordt NIET op u gewacht. Het programma gaat gewoon door. Het '$' teken betekent dat wat u intoetst als tekst wordt verwerkt.

```
Voorbeeld:  PRINT INKEY$
```

Voordat u wat hebt ingetoetst is het programma al gestopt. Om nu toch wat in te kunnen toetsen, kunnen twee dingen doen:

a/ we kunnen het PAUSE (K,m) kommando gebruiken.
b/ we kunnen een lus maken, waarin INKEY$ steeds het toetsenbord leest totdat er wat wordt ingetoetst.

```
Voorbeeld:  PAUSE 0: PRINT INKEY$

        10 PRINT INKEY$: IF INKEY$="" THEN GOTO 10
```

U kunt wat u intoetst ook aan een variabele toekennen.

```
Voorbeeld:  PAUSE 0: LET a$=INKEY$
            PRINT a$
```

Een wat ingewikkelder voorbeeld met IF (K,u) en THEN (K,SS+g) omdat we zo de invoer kunnen controleren op correctheid. Dit noemen we screening. In een spelletje kunnen we zo controleren of we wel dat intoetsen wat relevant is.

Voorbeeld:
```
 5 REM    lezen toetsenbord
10 IF INKEY$="5" THEN GOTO (links)
20 IF INKEY$="6" THEN GOTO (beneden)
30 IF INKEY$="7" THEN GOTO (omhoog)
40 IF INKEY$="8" THEN GOTO (rechts)
50 IF INKEY$="i" THEN GOTO (instructies)
60 GOTO 10
```

Wat u intoetst wordt bijgehouden in 2 systeemvariabelen. De een bewaart de ASCII code van het karakter en de ander de ASCII code van de hoofdletter welke op de TOETS staat, om te weten welke toets is ingedrukt. Dit zijn de variabelen op de adressen 23560 en 23556. Als u bijvoorbeeld PRINT intoetst, zal in adres 23560 de waarde 245 staan (ASCII code van PRINT) en in adres 23556 de waarde 80 (ASCII code hoofdletter P).

Via deze systeemvariabelen kunnen we ook het toetsenbord lezen. In het vorige voorbeeld controleren we of u een 'i' intoetst (regel 50). Stel dat u in hoofdletter mode zit. Dan kan u op 'I' blijven drukken tot u een ons weegt, maar de computer zal niet naar 'instructies' springen omdat de ASCII code van de 'i' een andere is dan van de 'I'. Wat u had kunnen doen is het volgende:

```
50 IF INKEY$="i" OR INKEY$="I" THEN GOTO (instructies)
```

Maar dit kost extra geheugen. Omdat in adres 23556 toch de ASCII code van de hoofdletters op de toets wordt bewaart, of u nu kleine letters of hoofdletters gebruikt, is het het simpelst om de inhoud van dit adres te lezen.

Voorbeeld:
```
 5 REM de ASCII code van 5=53, 6=54, 7=55, 8=56 en
van I=73
10 LET a=PEEK 23556
15 IF a=53 THEN GOTO (links)
20 IF a=54 THEN GOTO (beneden)
30 IF a=55 THEN GOTO (omhoog)
40 IF a=56 THEN GOTO (rechts)
50 IF a=73 THEN GOTO (instructies)
60 GOTO 10
```

U kunt nu geen fout maken. Als u CAPS SHIFT en 7 indrukt of SYMBOL SHIFT en 7 zal dit NIETS uitmaken. In 23556 staat dan altijd de waarde 55. In 23560 daarentegen zal NIET de waarde 55 staan, maar de ASCII code van CS+7 of SS+7.

§2. Cassettes.

Alles wat in het geheugen voorkomt kunnen we in principe op band bewaren en later weer van de band teruglezen (laden). Achter op de computer zit hiervoor een ingang en een uitgang, de EAR ingang en de MIC uitgang. EAR hoort bij EAR op uw cassette deck of recorder en MIC evenzo.

Om gegevens naar de band te schrijven gebruiken we het kommando SAVE (K,s). Dit kommando MOET gevolgd worden door een naam, zodat de computer later de gegevens terug kan vinden. Verder kent het SAVE kommando nog diverse opties, die voor de soort gegevens van belang zijn.

Opties: SAVE "naam" .....
Op de plaats van de stippellijn kan komen:
(sadr=startadres en rnr=regelnummer)

Niets	: BASIC programma, niet zelfstartend.
LINE rnr	: BASIC programma, zelstartend op rnr.
DATA a$()	: Blok data, dus geen programma.
CODE sadr,l	: Blok bytes, dat start op sadr en waarvan l de lengte is in bytes.
SCREEN$	: Het beeldschermgeheugen.

Voorbeeld:
```
SAVE "progr"
SAVE "progr" LINE 10
SAVE "tekst" DATA c$()
SAVE "bytes" CODE 16384,6913
SAVE "scherm" SCREEN$
```

In de eerste regel wordt het BASIC programma "progr" geSAVEd. Als u dit programma weer laad, moet u zelf RUN geven.
In de tweede regel wordt het programma geSAVEd met opgaaf van regelnummer 10. Wanneer u dit programma laad, zal het uit zichzelf starten op regel 10.
In de derde regel wordt er geen programma, maar data, naar de band geschreven. In dit geval alfanumerieke data, maar als u getallen wil bewaren, zet u uw variabele achter DATA. WEL met de haakjes erna, dit is verplicht.
En in de vierde regel worden er bytes weggeschreven. Dit kan alles zijn. In bovenstaand voorbeeld wordt het beeldscherm naar de band geschreven. In dit voorbeeld is dit hetzelfde als de laatste regel.

Om te controleren of het SAVEn correct is verlopen, hebben we het kommando VERIFY (E,SS+r). Spoel de band terug tot aan het begin van het programma. Doe:

a/ BASIC programma: VERIFY "naam"
U moet, indien u dat hebt gedaan, LINE weglaten.

b/ Data: VERIFY "naam" DATA c$()
Dit moet wel volledig.

c/ Code (bytes): VERIFY "naam" CODE
Hier mag u de gegevens weglaten.

d/ Beeldscherm: VERIFY "naam" SCREEN$ of zoals in c.

Zorg dat de EAR is aangesloten en de MIC uit de computer is verwijderd. Start dan de band. De gegevens worden niet geladen, hoewel het er wel op lijkt.

Nu wilt u de gegevens terug gaan laden. Hier hebben we het LOAD (K,j) kommando voor. Hiermee gaat u te werk als met VERIFY. De gegevens worden nu wel geladen.

Dan is er nog het kommando MERGE (E,SS+t). Dit kommando is ALLEEN voor BASIC programma's en NIET voor bytes en data. U hebt met dit kommando de mogelijkheid om BASIC programma's aan elkaar te koppelen. U kunt dan modulair programmeren, dat wil zeggen, dat u niet 1 heel groot programma schrijft, maar allemaal stukjes. Het hoofdprogramma houdt u in het geheugen en afhankelijk van de keuze laad u een module.

Een paar punten waar u ten zeerste rekening mee dient te houden zijn:

a/ Let op uw regelnummering. Met MERGE worden regels met nog niet voorkomende regelnummers tussen de wel voorkomende regels gevoegd. Indien er een reeds bestaand nummer wordt geladen, vervalt de bestaande. Dat wil zeggen dat de bestaande regel wordt overschreven door de nieuwe.

b/ Het programma(deel) wat u gaat MERGEn moet nog wel in het geheugen passen.

c/ Indien u een klein programma in het geheugen hebt en u wilt daar een groot programma in MERGEn, zal dat verschrikkelijk lang duren. Beter is het om het kleine deel eerst op band te zetten, dan het grote programma te LADEN en vervolgens het kleine programma te MERGEn.

Voorbeeld (a): U heeft het volgende programmaatje in het geheugen staan:

```
10 PRINT AT 0,14;"MENU"
20 PRINT AT 3,5;"Invoeren = 1"
30 PRINT TAB 5;"Opvragen = 2"
40 PRINT TAB 5;"Wijzigen = 3"
50 INPUT "Uw keuze? ";a
```

en u gaat de volgende regels MERGEn:

```
30 PRINT TAB 5;"Sorteren = 4"
45 PRINT TAB 5;"Printen  = 5"
```

U bent regelnummer van regel 30 vergeten aan te passen en het zal er zo uitzien:

```
   10 PRINT AT 0,14;"MENU"
   20 PRINT AT 3,5;"Invoeren = 1"
-> 30 PRINT TAB 5;"Sorteren = 4"
   40 PRINT TAB 5;"Wijzigen = 3"
-> 45 PRINT TAB 5;"Pinten  = 5"
   50 INPUT "Uw keuze? ";a
```

Een paar tips:

1/ U moet, als u gaat saven, de EAR plug uit de computer verwijderen. Om dit niet te vergeten is het makkelijk om altijd maar 1 plug in de computer te hebben. Dus als u gaat laden haalt u de MIC plug eruit. Zo wordt het een gewoonte.

2/ Lees voor deze tip eerst hoofdstuk 3. U kunt de namen van programma's, data en bytes die u gaat SAVEn, ook kleuren meegeven binnen de aanhalingstekens. Deze hoeft u NIET te gebruiken als u het programma weer gaat laden.

Voorbeeld: SAVE "(E,9)naam"
LOAD "naam"

3/ U kunt normaal een naam van minimaal 1 en maximaal 10 karakters meegeven. Dit mogen ook tekens en graphics zijn. Indien u zoals in 2 een kleur hebt meegegeven, heeft u nog maar een maximum van 9 tekens.
Geeft u twee kleuren mee, bijvoorbeeld BRIGHT en FLASH dan daalt het maximum tot 6 tekens.

§3. Microdrives.

De microdrive is een computergestuurd extern geheugen, dat werkt met een eindeloze band. Het is sneller en compacter dan een cassette en een cassettrecorder, maar in langzamer dan schijven en biedt ook minder mogelijkheden dan schijven. U hebt INTERFACE 1 nodig.

Om van de band te lezen en naar de band te schrijven gebruiken we dezelfde kommando's als met cassettes, maar met een uitbreiding. Dit is om de computer te laten weten dat het NIET om een cassette gaat.

Voorbeeld: SAVEn naar cassette weet u nu. De volgende SAVE opdracht geldt voor drive 1:

```
SAVE *"m";1;"naam"
```

Het deel '*"m"' is de aanduiding dat er een microdrive gebruikt gaat worden. De 1 is welke er gebruikt gaat worden. U kunt aan de Spectrum maximaal acht microdrives hangen. De meest linkse is altijd de laatste en daarom is de 1 hier de meest rechtse. De ; wordt als scheidingsteken gebruikt. Dit mag geen ander teken zijn. Verder gelden alle opties zoals ze in §2 bij SAVE zijn beschreven. Voor de 1 mag u ook een numerieke variabele gebruiken.

Voorbeeld:
```
LET md=1
SAVE *"m";md;"naam"
```

De kommando's VERIFY, LOAD en MERGE werken ook met het stuk '*"m";md;'. Voor de rest blijft alles hetzelfde, behalve bij SAVE, waar 1 optie is bijgekomen.

Voorbeeld: Cassette:

```
SAVE "naam" LINE 10
VERIFY "naam"
NEW
LOAD "naam"
```

Microdrive:

```
SAVE *"m";1;"naam" LINE 10
VERIFY *"m";1;"naam"
NEW
LOAD *"m";1;"naam"
(idem met MERGE)

u kan als naam nu 'run' geven.

SAVE *"m";1;"run" LINE 10
VERIFY *"m";1;"run"
NEW
RUN
```

Het programma met de naam 'run' (kleine letters) wordt nu automatisch opgezocht en geladen en zal uitzichzelf starten op regel 10, omdat we LINE 10 hebben meegegeven.

Er zijn wel extra kommando's bijgekomen. Een cartridge moet namelijk, voordat deze gebruikt wordt, een naam krijgen. Hiervoor is het kommando FORMAT (E,SS+0) wat u al op uw computer had gezien.

Verder wilt u natuurlijk graag weten wat er op de band staat. Bij de cassettes moest u dit met de hand bijhouden, maar nu doet de computer het voor u door middel van het kommando CAT md (E,SS+9) waarin md voor een drivenummer staat. CAT is een afkorting van CATalogus. U krijgt op uw beeldscherm de naam van de band, de namen van de files, op ASCII code gesorteerd, en de hoeveelheid vrij geheugen in KB (Kilo Bytes).

Nog een extra kommando is ERASE (E,SS+7) wat 'wis' betekent. Met uw cassettes was het simpel om een nieuw programma over het oude heen te SAVEn. Dat kan met de microdrive niet. U SAVEd het programma (als er nog voldoende geheugen is) met een andere naam en vervolgens controleert u met VERIFY of het transport correct heeft plaatsgevonden. DAN pas ERASEd u het oude programma.

Voorbeeld: ERASE "m";1;"naam"

Bij ERASE hoeft u niets anders dan de naam te geven. Of dit nu een BASIC programma, bytes of DATA is. De computer zoekt naar de naam en wist de file van de band.

Deze extra kommando's kon u niet eerder gebruiken, vandaar dat u bij deze geen * hoeft te gebruiken.

Het is raadzaam om het FORMATeren een paar keer te herhalen, zodat het inderdaad goed gebeurt. Of het gelukt is kunt u zien door het FORMAT kommando direct door CAT te laten volgen.

```
Voorbeeld:  FORMAT "m";1;"naam"
            CAT 1
```

Op het beeldscherm verschijnt de naam van de cartridge en het aantal KB. Dit moet minstens 87 KB zijn en is hoogstens 94 KB. 100 KB is een onmogelijkheid.

§4. Het werken via streams.

In hoofdstuk 1 hebben we al kennis gemaakt met het # om naar het onderste deel van het beeldscherm te printen. Dit teken houdt in dat u een stream opent tussen de computer en een randapparaat.

Een overzicht van de streams:

0 en 1: Uitvoer naar het onderste deel van het beeldscherm of invoer van het toetsenbord.

2 : Uitvoer naar het bovenste deel van het beeldscherm.

3 : Uitvoer naar de printer.

4 t/m 15: Zelf te definieren.

Voor de streams 0, 1, 2 en 3 hoeven we niet op te geven welk randapparaat er bedoeld wordt. Dit is al in de computer vastgelegd.

```
Voorbeeld:  PRINT #1   =  PRINT #0  ;onderste deel beeldscherm.
            PRINT      =  PRINT #2  =  LPRINT #2
            PRINT #3   =  LPRINT
            LIST       =  LIST #2   =  LLIST #2
            LIST #3    =  LLIST
```

Zo kunnen we de catalogus van een cartridge naar de printer sturen:

```
CAT #3;1  (1=drive 1)
```

U kan het streamnummer ook aan een variabele toekennen:

```
10 INPUT "CAT naar beeldscherm of printer? 2/3";a
20 IF a<>2 AND a<>3 THEN GOTO 10
30 CAT #a;1
```

Idem met een listing:

```
10 INPUT "LIST naar beeldscherm of printer? 2/3";a
20 IF a<>2 AND a<>3 THEN GOTO 10
30 LIST #a
```

Normaal gesproken moeten we, voordat we via een stream data kunnen verplaatsen, een channel openen. De channels die de computer kent zijn:

```
K - toetsenbord
S - beeldscherm
P - ZX-printer of de Seikosha's
m - microdrive
n - netwerk
t - RS232: tekst
b - RS232:
```

De channels m, n, t en b werken alleen met INTERFACE I. De streams 0, 1, 2 en 3 behoeven niet geopend te worden, hier zorgt de computer voor. Bij gebruik van de streams 4 t/m 15 moet u wel een channel openen. Na het gebruik van een channel, moet we de stream weer sluiten.
Voor deze handelingen gebruiken we het OPEN # (E,SS+4) en het CLOSE # (E,SS+5) statement.

Voorbeeld:
```
OPEN #4;"P"        ;stream 4 gekoppeld aan channel P.
PRINT #4;"Naar printer"
LIST #4
CLOSE #4

OPEN #4;"K"        ;stream 4 gekoppeld aan channel K.
INPUT #4;a$
CLOSE #4
```

Voor het wegschrijven van data naar de microdrive, kunnen we ook een stream openen.

Voorbeeld:
```
OPEN #4;"m";1;"naam"
```

Hier is stream 4 gekoppeld aan channel m. De 1 is het drivenummer en 'naam' is de naam van de file. Geen BASIC programma. U kunt nu data naar de band PRINTen.

Voorbeeld:
```
DIM a(10)
FOR i=1 TO 10
INPUT a(i)
NEXT i
OPEN #4;"m";1;"nummers"
FOR i=1 TO 10
PRINT #4;a(i)
NEXT i
CLOSE #4
```

Teruglezen gaat als volgt:

```
DIM a(10)
OPEN #4;"m";1;"nummers"
FOR i=1 TO 10
INPUT #4;a(i)
NEXT i
CLOSE #4
```

Dit is nog wel te doen. Maar stel nou dat u 1000 getallen wil lezen. Dat duurt heel lang. In dat geval kan u beter de data SAVEn.
Stel dat u de inhoud van de file wilt weten. U kunt als bovenstaand voorbeeld te werk gaan, maar het kan sneller met het MOVE (E,SS+6) statement. MOVE opent de file uit zichzelf en geeft de data door aan de stream die u hebt gekozen.
Bijvoorbeeld de inhoud van file 'nummers' naar de printer sturen:

```
MOVE "m";1;"nummers" TO #3
```

Zie verder de bijgeleverde handleiding. Hiermee is ook de uitvoer naar de printer besproken.

Hoofdstuk 3: Grapics

- Inleiding.

§1. User defined graphics (UDG's).

§2. Het veranderen van de karakterset.

Inleiding.

De Spectrum beschikt zelf over 16 gedefinieerde grafische karakters en 21 zelf te definieren karakters.
U kunt ze benaderen door in graphische mode te gaan. (CS+9)
In deze mode kunt u DELETEn door gewoon op 0 te drukken. Zonder CAPS SHIFT dus. Om uit deze mode te raken drukt u op 9.

In graphische mode zijn de cijfertoetsen 1 t/m 8 voor de standaard karakters en de letters A t/m U voor de zelf te definieren karakters.
Op de cijfertoetsen wordt het witte deel van het karakter zwart als u CAPS SHIFT gebruikt, anders wordt het PAPER.
Graphics is tekst. Om ze te printen moet u ze tussen " " zetten of ze in een stringvariabele plaatsen, bijvoorbeeld a$.

We zullen in dit hoofdstuk alleen ingaan op het zelf maken van graphics. Ook besteden we aandacht aan het zelf maken van een nieuwe karakterset.

§1. User defined graphics (UDG's).

Het geheugen is te benaderen door het gebruik van adressen. Het UDG deel staat aan het einde van het geheugen en begint op adres 65368. Dit is de eerste byte van UDG "a". In plaats van het adres te zoeken, kunt u elk adres van de UDG's ook aanroepen door middel van USR (E,1).

```
Voorbeeld:  PRINT PEEK 65368  of
            PRINT PEEK USR "a"
```

Om een waarde van een geheugenplaats te lezen gebruiken de functie PEEK (E,o). Deze functie geeft de decimale waarde van 1 byte. Om een waarde in een geheugenplaats te schrijven gebruiken we het kommando POKE (K,o). De waarde mag niet kleiner zijn dan 0 en niet groter zijn dan 255.
Om onze UDG's te maken gebruiken we deze statements.

We zullen onder de A een andere A zetten. Zodra u uw computer aanzet, worden de hoofdletters A t/m U naar de UDG's gecopieerd. Om een andere letter onder de A de zetten moeten we de eerste 8 bytes veranderen. Dit zijn de adressen:

```
USR "a"+0  -  65368
USR "a"+1  -  65369
USR "a"+2  -  65370
USR "a"+3  -  65371
USR "a"+4  -  65372
USR "a"+5  -  65373
USR "a"+6  -  65374
USR "a"+7  -  65375
```

U ziet al dat we straks een lus gaan gebruiken:

```
FOR i=0 to 7
..... USR "a"+i
NEXT i
```

We gaan eerst kijken hoe de nieuwe A er uit gaat zien. Dit doen we door acht rijen van nullen en enen te tekenen. De enen vormen samen de letter.

We zetten voor elke rij gelijk het adres waar de waarde van de byte (=1 rij) gePOKEd moet worden. Achter elke rij zetten we de decimale waarde van de byte.

```
USR "a"+0 - 00000000 - 0
USR "a"+1 - 01111110 - 126
USR "a"+2 - 01100010 - 98
USR "a"+3 - 01111110 - 126
USR "a"+4 - 01100010 - 98
USR "a"+5 - 01100010 - 98
USR "a"+6 - 01100010 - 98
USR "a"+7 - 00000000 - 0
```

In plaats van USR "a" kunt u natuurlijk USR "b" gebruiken. Dan komt de nieuwe letter onder de B.

We zijn tevreden over deze A en gaan nu de waarden POKEn. Dit kan op vele manieren. De volgende voorbeelden illustreren ze:

1/ Decimale waarden, direct.

```
POKE USR "a"+0,0
POKE USR "a"+1,126
POKE USR "a"+2,98
POKE USR "a"+3,126
POKE USR "a"+4,98
POKE USR "a"+5,98
POKE USR "a"+6,98
POKE USR "a"+7,0
```

2/ Decimale waarden, indirect.

```
DATA 0,126,98,126,98,98,98,0
FOR i=0 TO 7
READ a
POKE USR "a"+i,a
NEXT i
```

3/ Binaire waarden, direct.

```
POKE USR "a"+0,BIN 0
POKE USR "a"+1,BIN 01111110
POKE USR "a"+2,BIN 01100010
POKE USR "a"+3,BIN 01111110
POKE USR "a"+4,BIN 01100010
POKE USR "a"+5,BIN 01100010
POKE USR "a"+6,BIN 01100010
POKE USR "a"+7,BIN 0
```

4/ Binaire waarden, indirect.

```
DATA BIN 0,BIN 01111110,BIN 01100010,BIN 01111110,
     BIN 01100010,BIN 01100010,BIN 01100010,BIN 0
FOR i=0 TO 7
READ a
POKE USR "a"+i,a
NEXT i
```

5/ Met INPUT.

```
INPUT "Onder welke letter? ";a$
FOR i=0 TO 7
INPUT "Waarde voor USR ";(a$);"+";(i);"? ";a
IF a<0 or a>255 THEN GOTO FOUT
POKE USR a$+i,a
PRINT "USR ";a$;"+";i;" - ";a
NEXT i
```

In de voorbeelden wordt van BIN gebruik gemaakt. Dit is een afkoring van BINair. In het laatste voorbeeld is FOUT niet verder uitgewerkt. FOUT is een variabele waarvan de waarde een regelnummer moet zijn.

U kunt zo bijvoorbeeld een duikboot maken, door het voorstuk onder de A, het middelstuk onder de B en het achterstuk onder de C te zetten. Om de boot dan als geheel te printen doet u het volgende:

```
PRINT "(CS+9)ABC"
```

Indien u de waarden nog eens terug wilt lezen doet u het zoals de volgende voorbeelden:

1/ Met variabele.

```
INPUT "Van welke UDG? ";a$
FOR i=0 TO 7
LET a=PEEK (USR a$+i)
PRINT a
NEXT i
```

U kan nooit LET a=USR a$ zeggen, omdat dit een RUN kommando is voor een machinetaalprogramma. U MOET dus PEEK gebruiken.

2/ Direct.

```
INPUT "Van welke UDG? ";a$
FOR i=0 TO 7
PRINT PEEK (USR a$+i)
NEXT i
```

Omdat PEEK een functie is MOET u haakjes om de operand plaatsen zolang deze een berekening is. Als u alleen USR a$ zou gebruiken hoeven er geen haakjes omheen. Bij POKE hoeven er geen haakjes omheen.

§2. Het veranderen van de karakterset.

We kunnen de Spectrum een geheel andere karakterset geven. Dit is uiteraard niet voor altijd, maar zolang de computer aanstaat werkt u dan met de nieuwe karakters. Een andere mogelijkheid is om te werken met twee karaktersets, waarvan er een door u is gemaakt. We zullen met de eerste mogelijkheid beginnen.

vervanging.

Voor het gemak zullen we ons bij alleen bij letters houden. We zullen eerst moeten weten op welk adres de eerste byte staat van de kleine letter a en de hoofdletter A.
Nu heeft het monitorprogramma z'n eigen variabelen. Net zoals een BASIC-programma variabelen heeft. Nu is er een variabele welke het beginadres bevat van de karakterset. Deze variabele beslaat twee bytes, omdat de waarde in 1 byte niet groter kan zijn dan 255. Om nu deze twee bytes te lezen als 1 variabele gebruiken we de volgende formule:

```
LET a=PEEK (adres)+256*PEEK (adres+1)
```

De variabele a bevat nu de correcte waarde. De adressen van de systeemvariabele zijn 23606 en 23607. We passen de formule op deze adressen toe:

```
LET a=PEEK 23606+256*PEEK 23607
```

De variable a bevat nu 15360. Dit is het beginadres van de karakterset. Als u hier 256 bij optelt heeft u het eerste adres van de spatie.
De eerste a is een hoofdletter. Deze heeft de ASCII code 65. Om nu het adres van de eerste byte te vinden doet u het volgende:

```
LET b=65*8=520
LET a=a+b=15880
```

Noteer dit adres op een kladje. De kleine letter a heeft ASCII code 97. Op dezelfde manier als boven berekenen we het adres van de eerste byte:

```
LET b=97*8=776
LET a=a+b=16136
```

Noteer dit ook op het kladje. Nu gaat u, zoals in paragraaf 1 is uitgelegd, nieuwe karakters maken in de UDG's. Eerst de hoofdletters A t/m U. Deze copieert u vervolgens naar het monitorprogramma. Er zijn 21 UDG's. Dit is bijelkaar 21*8=168 bytes.

```
FOR i=0 TO 167
POKE 15880+i,PEEK (USR "a"+i)
NEXT i
```

U heeft nu de UDG's a t/m u gecopieerd onder de letters A t/m U. Nu moet u nog zorgen voor de letters V t/m Z. U maakt weer nieuwe karakters onder de UDG's. Onder de a komt dus de nieuwe V te staan en onder de e de nieuwe Z. Dit beslaat slechts vijf UDG's wat 5*8=40 bytes is. Deze copieert u nu onder de V t/m Z.

U bent in het vorige voorbeeld geeindigd bij de U. Dit waren 168 bytes. Het b beginadres van de V is dan 15880+168=16048.

```
FOR i=0 TO 39
POKE 16048+i,PEEK (USR "a"+i)
NEXT i
```

Nu zijn alle oude hoofdletters vervangen door nieuwe. U kunt deze nieuwe karakterset op band bewaren. Het totale aantal bytes is 168+40=208.

```
SAVE "naam"CODE startadres,lengte
SAVE "karakters"CODE 15880,208
```

Om de kleine letters te vervangen doet u hetzelfde als boven, maar dan niet met adres 15880 (hoofdletters) maar met 16136.

Twee karaktersets.

In plaats van de karakterset te vervangen, kunnen we de computer ook wijs maken dat de karakterset op een geheel ander adres begint. We POKEn dan het adres, waar de nieuwe set begint, in de systeemvariabele CHARS (adressen 23606 en 23607).
Als voorbeeld nemen we adres 15360. Dit is het beginadres van de karakterset.

```
LET a=INT (15360/256)=60
LET b=15360-256*a=0
POKE 23606,b
POKE 23607,a
```

Wat u nu gaat doen is het volgende: U kopieert de gehele karakterset naar een andere plaats. U begint bijvoorbeeld op USR "a"-256*8=63320. Dan komt de set precies onder de UDG's te liggen. Nu POKEd u 63320 in CHARS op bovenstaande manier.

```
LET a=INT (63320/256)=247
LET b=63320-256*a=88
POKE 23606,b
POKE 23607,a
```

Nu gaat u in de gecopieerde set de karakters vervangen zoals in **vervanging.** is besproken. U werkt dan met andere adressen dan die we eerst hebben gebruikt.

U kunt zo ook andere cijfers en tekens maken. Wat u beter NIET kan doen is het veranderen van de eerste 256 bytes en de laatste 1024 bytes. De eerste 256 zijn namelijk besturingstekens voor PRINT, DELETE, ENTER, AT, TAB, de cursor en de attribuut functies en de laatste 1024 zijn de codes van de UDG's, functies en kommando's zelf.

Hoofdstuk 4: BASIC en BETA BASIC

- Inleiding.

§1. Kommando's.

§2. Functies.

§3. Beta BASIC variabelen.

§4. Interresante systeem variabelen.

Inleiding.

In dit hoofdstuk zullen we het BASIC van de Spectrum en Beta BASIC bespreken. De Beta BASIC kommando's en functies die overeen komen met Spectrum BASIC kommando's en functies zullen na elkaar worden genoemd en worden aangeduid met 'BB'. De rest staat op alfabetische volgorde tussen de standaard kommando's en functies in. De meeste kommando's en functies zijn van BASIC programma's voorzien ter verduidelijking.

§1. Kommando's.

ALTER (BB) (G,a)> Om de attributen te veranderen.

```
ALTER TO PAPER 1, INK 0, BRIGHT 1

is hetzelfde als:
PAPER 1: INK 0: BRIGHT 1: CLS
```

AUTO (BB) (G,6)> Om tijdens het programmeren gebruik te maken van automatische regelnummering. Na ENTER geeft Beta BASIC het volgende regelnummer. Met DELETE kunt u dit regelnummer veranderen.

BEEP d,t (E,SS+z)> Brengt een toon voort via de luidspreker en de EAR-uitgang.
d = duur van de toon in seconden.
t = toonhoogte. Mag ook negatief zijn.

BORDER k (K,b)> Om de randen om het scherm een kleur te geven.
k = kleur. 0 <= k <= 7

BREAK (BB) > Geen keyword. Beta BASIC heeft er een tweede BREAK aangekoppeld, zodat u nu zelfs uit een zelfgeschreven machine-taal programma kan komen.

BRIGHT s (E,SS+b)> Beeldscherm of uitvoer helder of normaal.
s=1 : helder
s=0 : normaal

```
BRIGHT 1: CLS   hele scherm helder.
PRINT BRIGHT 1;"Tekst helder"
```

CAT md (E,SS+9)>

Om de inhoud van een microdrive cartridge op te vragen.
md = drivenummer. 0 <= md <= 8

```
CAT 1      inhoud van cartridge in drive
           1 naar het beeldscherm.

CAT #3;1   idem, maar nu naar de ZX-
           of de Seikosha printer.
```

CIRCLE x,y,r (E,SS+h)>

Tekent een cirkel op het beeldscherm met (x,y) als middelpunt.
x = horizontale coordinaat.
y = verticale coordinaat.
r = straal van de cirkel in pixels.

CLEAR adres (K,x)>

Veegt het variabelen geheugen schoon, geeft een RESTORE en CLS. Het zet de PLOT en PRINT posities weer op (0,0) en maakt de GOSUB stack leeg.
Indien u een adres meegeeft, wordt dat de nieuwe RAMTOP. Dat wil zeggen dat u ruimte heeft voor BASIC tot 'adres'.

CLOCK (BB) (G,c)>

Geeft u een digitale klok die u op het beeldscherm zichtbaar kan maken en welke op een bepaald tijdstip een subroutine kan starten.
Tijdens SAVE, LOAD of LPRINT staat de klok stil.

```
CLOCK "13:30:00"  zet de tijd op 13.30
CLOCK 1    laat de klok op het scherm
           zien.
CLOCK "A14:30"    zet alarm op 14.30
CLOCK 100         GOSUB 100
CLOCK 4 t/m 7     Aktiveer alarm.
CLOCK "AXXXX"     Wis alarmtijd.
```

CLOSE #s (E,SS+5)>

Sluit een stream. Zie OPEN #.
s = streamnummer. 4 <= s <= 15

CLS (K,v)>

Maakt het scherm leeg, zet de PLOT en PRINT posities weer op (0,0).

CONTINUE (K,c)>

Gaat verder waar het programma door een error is gestopt. Indien de error niet is verholpen stopt het programma weer.
Dit gebeurt niet indien u regels hebt moeten toevoegen of wijzigen.

COPY (K,c)> Maakt een afdruk van het beeldscherm. Tekeningen kunnen alleen op de ZX- en de Seikosha printer worden afgedrukt. Voor het printen naar printers via een interface zal u eerst de printer volgens de handleiding aan moeten passen.

DATA d1,d2,d3, (E,d)> Hoort bij READ. Heeft niets te maken met DATA in een SAVE, maar is wel hetzelfde woord. DATA in een programmaregel zijn waarden die een programma altijd nodig heeft en daarom ook niet veranderen.
U mag strings en numerieke waarden door elkaar in een DATA statement zetten.

```
10 DATA 1,0,4,"t","e",8,"k","s",5,"t"
20 READ a,b,c,a$,b$,d,c$,d$,e,e$
```

Deze manier is vooral handig en geheugen besparend als u veel variabelen moet initialiseren aan het begin van een programma.
DATA mag overal in een programma voorkomen.

DEF FN (E,SS+1)> Om een formule die meerdere keren in een programma gebruikt moet worden slechts 1 keer in te tikken. Dit voorkomt fouten en is geheugen besparend. Om de diverse formules te onderscheiden krijgen ze een naam die uit 1 letter mag bestaan.

```
10 DEF FN a(x)=x^x
20 DEF FN b(x,y)=x^y
```

DEF FN mag overal in een programma staan.

DEF KEY (BB) (G,SS+1)> Om een programma regel of een direct uit te voeren statement onder een toets kwijt te kunnen.
Let goed op : en ;

```
DEF KEY "1": PRINT "Tot ziens."
```

Onder toets '1' staat nu het uit te voeren PRINT statement. Als u nu SS+SPACE indrukt verandert de cursor in een sterretje. Druk vervolgens op '1' en 'Tot ziens' wordt gePRINT.

```
DEF KEY "a";"tekst:"
```

In dit geval wordt 'tekst' als een onderdeel van een basic regel beschouwd en zal ook onderin het beeld verschijnen en wachten op ENTER of meer tekst.

```
10 INPUT "(SS+SPACE,a)";a$
```

Zie verder de handleiding van BB.

DEF PROC naam (BB) (G,1)> Dit is een soort GOSUB met dit verschil dat u GEEN regelnummer hoeft te onthouden. De procedure (een stuk BASIC) moet wel een naam krijgen.
De procedure moet eindigen met END PROC, een soort RETURN.

```
10 DEF PROC initialisatie
20 DATA 0,0,"","",10,22,"tekst"
30 READ a,b,a$,b$,c,d,c$
40 END PROC

90 ........
100 PROC initialisatie
110 .......
```

U mag niet uit een procedure springen. U MOET dan naar END PROC springen. Een procedure mag overal in het programma voorkomen. Het programma zal niet langzamer zijn.

DELETE v TO t (BB) (G,7)> Om een blok regels uit een programma te verwijderen.
v = regelnummer vanaf
t = regelnummer t/m.

U kunt BB ook als toolkit gebruiken, bijvoorbeeld een gewoon BASIC programma hernummeren.
U doet het volgende:
- laad Beta BASIC;
- MERGE het BASIC programma;
- toets RENUM (G,4) in;
- vervolgens DELETE 0 TO 0
- SAVE het BASIC programma.

DIM n(i,b,d,..) (K,d)> Dimensioneert een tabel van 1 x b x d x n elementen en wist een reeds bestaande van met dezelfde naam.
n = naam van de tabel.
Indien n niet gevolgd wordt door een '$' is de tabel voor floating-point getallen. Met het $ teken voor alfa-numieke tekens. (vaste lengte)

DIM a(3,4) = tabel van 3 x 4 = 12 elementen.

DIM a$(3,4) = tabel van 12 elementen voor tekst.

U kunt een tabel DIMensioneren zo groot als er geheugen vrij is.

DO (BB) (G,d)> Om een lus te vormen, zodat een stuk BASIC een bepaald aantal maal wordt uitgevoerd. Achter DO kunt u de condities plaatsen die bepalen hoe vaak de lus moet worden uitgevoerd.
WHILE (G,j) = zolang
UNTIL (G,k) = totdat

10 LET gev=0: DO WHILE NOT gev

Hier staat: DOE ZOLANG NIET gev =
DOE ZOLANG gev=0

10 LET gev=0: DO UNTIL gev

Hier staat: DOE TOTDAT gev =
DOE TOTDAT gev<>0

De lus wordt afgesloten met LOOP (G,l) (soort NEXT)

U mag NOOIT met IF .. THEN GO TO uit een DO ... LOOP springen. Bij FOR ... NEXT mag dit wel. (zie EXIT IF)

DRAW x,y,z (K,w)> Om rechten en krommen te tekenen. Als u een rechte lijn wilt is z=0 en mag u z weglaten.
Wanneer z<>0 dan tekent u een kromme (stuk van een cirkel) naar (x,y). Dit punt is dus een deel van die cirkel. (zie H.3)

EDIT rnr (BB) (K,O)> EDITs de regel met nummer rnr. Indien het woord EDIT niet verschijnt, maar een O, toets dan na de O een spatie en vervolens het regelnummer.

EDIT 10 en O 10 zijn dan identiek.

ELSE (BB) (G,e)> Aanvulling op IF ... THEN.
Normaal wordt verder gegaan met de volgende BASIC regel als de conditie niet waar is. Nu wordt verder gegaan na ELSE.

```
10 IF a=1 THEN LET a=0:GO TO 100
20 GO TO 200    wordt:

10 IF a=1 THEN LET a=0: GO TO 100:ELSE
   GO TO 200
```

END PROC (BB) (G,3)> Einde van een procedure. Zie DEF PROC.

ERASE "m";md;n (E,SS+7)> Om een file van microdrive cartridge te wissen. U hoeft slechts de naam van de file mee te geven, ongeacht wat de soort is.
n = "naam" of een stringvariabele die op dat moment de naam bewaart.

EXIT IF (BB) (G,i)> Om op de correcte manier uit een lus (DO ... LOOP) te springen. Achter IF volgt de conditie waarop uit de lus moet worden gesprongen.

```
10 DO ....
20 EXIT IF INKEY$="E" OR INKEY$="e"
.....
70 LOOP
```

FILL x,y (BB) (G,f)> Vult een gebied met INK als FILL of FILL INK k is gebruikt en met PAPER als FILL PAPER k is gebruikt.

```
10 PLOT 79,95: DRAW 0,41: DRAW 41,0:
   DRAW 0,-41: DRAW -41,0: PAUSE 0
20 FILL INK 2; BRIGHT 1;110,110
30 PAUSE 0: FILL PAPER 6;FLASH 1;79,95
40 PLOT 127,7: DRAW 0,41: DRAW 41,0:
   DRAW 0,-41: DRAW -41,0: PAUSE 0
50 FILL INK 4; FLASH 1;129,8
```

FLASH 1/0 (E,SS+v)> Maakt een gebied flistend. (zie H.3)
1 = aan, 0 = uit.

FOR v=x TO y STEP z > FOR (K,f), TO (K,SS+f), STEP (K,SS+d)

Om een stuk BASIC een bepaald aantal maal uit te voeren.
Indien z=1 mag STEP worden weggelaten.
v = lusvariabele. (numeriek)
x = startwaarde
y = eindwaarde
z = stapgrootte
Indien v al eerder is gebruikt, wordt deze hier weer op 0 gezet.
De lus dient afgesloten te worden met NEXT v (K,n).
U mag met IF ... THEN GO TO uit een FOR .. NEXT lus springen.
De formule om uit te rekenen hoevaak de lus doorlopen wordt is:

aantal = 1+ INT ((y-x)/z)

De lus wordt beeindigd als v>y.

```
10 FOR i=-10 TO 10
20 PRINT i
30 NEXT i
40 PRINT i
```

FORMAT "m";md;n (E,SS+7)> Om een microdrive cartridge te initialiseren en een naam (n) te geven.
Alle reeds aanwezige files worden hierdoor gewist. (zie H.2)

GET (BB) (G,g)> Zelfde als INKEY$, met dit verschil dat er wordt gewacht tot er een toets wordt ingetoetst en dat deze aan de variabele achter GET wordt toegekend.

```
10 GET a$   is identiek met
20 PAUSE 0: LET a$=INKEY$
```

GOSUB rnr (K,h)> Er wordt naar de regel met rnr als nummer gesprongen en teruggesprongen naar het volgende statement achter de GOSUB als er een RETURN statement wordt tegengekomen.

GO TO rnr (K,g)> Er wordt naar de regel met rnr als nummer gesprongen.

IF cond THEN doe > IF (K,u), THEN (K,SS+g)

Om bepaalde handelingen onder bepaalde voorwaarden uit te voeren.
De conditie kan uit meerdere vergelijkingen bestaan die dmv AND (K,SS+y) of OR (K,SS+u) aan elkaar gekoppeld zijn.
Indien aan 'cond' wordt voldaan, worden de opdrachten 'doe' uitgevoerd.
Wanneer niet aan 'cond' wordt voldaan wordt verder gegaan met de volgende regel.

```
10 INPUT a$
20 IF a$<"A" OR a$>"Z" THEN GO TO 10
```

Achter THEN mag weer een IF ... THEN volgen:

```
10 INPUT a$;" en ";b$
20 IF a$="i" THEN IF b$="t" THEN GO TO
   ....
```

Dit is hetzelfde als:

```
10 INPUT a$;" en ";b$
20 IF a$="i" AND b$="t" THEN GO TO ..
```

maar dit is langzamer omdat hier twee vergelijkingen worden uitgevoerd en in de vorige slechts 1 als a$<>"i".

INK k (E,SS+x)> Om de inktkleur van de alle tekst te veranderen of, gebruikt binnen een PRINT statement, de te PRINTen tekst een inktkleur mee te geven.
0 >= k >= 7. (zie H.3)

NPUT (K,i)> Om de gebruiker data te laten invoeren welke nooit hetzelfde is.

```
10 INPUT a          is voor getallen.
20 INPUT a$         is voor tekst.
```

Als u de " " niet wilt gebruikt u LINE (E,SS+3):

```
30 INPUT LINE a$
```

Als u tekst wilt printen voor de variabele:

```
40 INPUT "Geef tekst: "; LINE a$    of
50 LET a$="Geef tekst: "
60 INPUT (a$);b$
```

(zie ook de voorgaande hoofdstukken)

INVERSE 1/0 (E,SS+m)> Om de PAPER en INK te wisselen. Dus PAPER wordt INK en andersom. (zie H.3) 1 = aan, 0 = uit.

JOIN rnr (BB) (G,SS+6)> Koppelt BASIC regel met nummer rnr aan de regel waar de cursor zich bevindt. Indien rnr wordt weggelaten, wordt de eerstvolgende regel genomen.

KEYIN (BB) (G,SS+4)> Hiermee kan u een BASIC regel, die als tekst in een stringvariabele is opgeslagen, in het programma invoeren. U kan stukken BASIC als tekst opslaan, deze LOADen en met KEYIN aan het programma toevoegen.

```
10 LET a$="10 DATA "
20 INPUT "Data: ";b$
```

U toetst in: 1,2,3,4,5,6,7,8,9,10

```
30 LET a$=a$+b$: SAVE "r10" DATA a$()
```

Vervolgens typt u NEW.

```
10 LET a$="": LOAD "r10" DATA a$()
20 KEYIN a$
```

Het ziet er dan zo uit:

```
10 DATA 1,2,3,4,5,6,7,8,9,10
20 KEYIN a$
```

KEYWORDS 1/0 (BB) (G,8)> Aangezien BB de graphics gebruikt voor BASIC keywords, kunt u deze niet zomaar gebruiken. Als u dat zou willen moet u eerst KEYWORDS 0 intypen.

```
10 KEYWORDS 0: LET a$="(G)1234567"
30 KEYWORDS 1: PRINT a$
40 KEYWORDS 0: PRINT a$
```

LET (K,1)> Met LET geven we variabelen een waarde (ook: READ, INPUT).

```
10 LET a=1: LET b=a: LET c=a+b
```

LIST rnr (K,k)> LIST het programma vanaf regel met nummer rnr. Als rnr wordt weggelaten is rnr=1.

LIST (BB) > Hetzelfde keyword, maar uitgebreid. U kan nu LISTen vanaf een rnr t/m een rnr.

```
                        LIST TO 100   1 t/m 100
                        LIST 50 TO 100
```

LLIST (E,v)> Idem als LIST normaal en met BB, maar nu naar de printer.
(zie voor LIST en LLIST ook H.2)

LOAD n wat (K,j)> Om van tape of disk gegevens te laden. LOAD overschrijft de reeds bestaande gegevens. (zie H.2)

```
n = "naam";
wat = niets : BASIC programma.
wat = DATA a$() of DATA a() : gegevens
wat = CODE st,l : l bytes vanaf adres
                  st. (st & l mogen
                  worden weggelaten).
wat = SCREEN$ : beeldscherm.
```

LOOP (BB) (G,l)> Zie DO. De condities welke achter DO kunnen, kunnen ook achter LOOP. Als ze achter DO staan, draait de lus minimaal 0 keer, achter LOOP minimaal 1 keer. Ze kunnen niet beide tegelijk achter DO en LOOP voorkomen. Dus of de een of de ander.

MERGE n (E,SS+t)> Zoals bij LOAD, maar alleen voor BASIC programma's. MERGE wist het aanwezige programma niet, doch voegt een ander programma bij/tussen het aanwezige. Identieke regelnummers worden door het te MERGEn programma overschreven.
(zie H.2)

MOVE s1 TO s2 (E,SS+6)> MOVE zorgt voor transport van gegevens via stream s1 naar stream s2. Stream s1 is altijd de stream waar de gegevens binnen komen en stream s2 de stream waar ze naar toe gaan.
MOVE heeft alleen zin met microdrives. U kan dan de inhoud van een file, die met OPEN # en CLOSE # naar de band is geschreven, direct naar een andere file schrijven of naar de printer sturen.

```
10 OPEN #4;"m";1;"listing"
20 LIST #4: CLOSE #4: NEW
```

```
10 MOVE "m";1;"listing" TO #3
```

stuurt de listing direct naar de printer. (zie H.2)

NEW (K,a)> Wist het BASIC programma en alle variabelen uit het geheugen. Dit is in feite een 'warme start' voor de computer. Een 'koude start', bijvoorbeeld na een machinetaal spel, is of de steker uit de computer of, bijvoorbeeld Beta BASIC wissen, GO TO USR 0.

NEXT v (K,n)> Zie FOR.

ON (BB) (G,o)> Uitbreiding op GO TO en GOSUB. In plaats van een reeks IF ... THEN statements kan u nu het volgende doen:

```
10 INPUT a: GO TO ON a;90,175,600
20 PRINT "FOUT, 1,2 of 3!!": GO TO 10
```

Als a=1 dan wordt naar de eerst genoemde regel (90) gesprongen, als a=2 naar de tweede (175) in de rij en als a=3 naar de derde (600), enz..

Hetzelfde geldt voor GOSUB.

```
10 INPUT "Welke routine? 1,2 of 3: ";a
20 GOSUB ON a;90,175,600
30 PRINT #1;"Terug van routine ";a
40 PAUSE 80: GO TO 10
```

ON ERROR rnr (BB) (G,n)> Door ON ERROR te gebruiken stopt het BASIC programma niet, maar er wordt naar regel met nummer rnr gesprongen, waar u de fout, dmv BASIC, af kan handelen. Met RETURN wordt terug gesprongen. Bij ON ERROR horen twee BB variabelen die u NIET voor andere doeleinden mag gebruiken.

- error = het nummer van de foutmelding.
- line = het regelnummer waar de fout optrad.

U moet ON ERROR altijd voor de handeling die u gaat uitvoeren plaatsen.

```
10 ON ERROR 100
20 INPUT "Van ";a;" t/m ";b
30 PLOT 100,100: DRAW a,b
40 GO TO 20
100 IF error=5 THEN PRINT #1;"De lijn
is te lang!!": PAUSE 80: RETURN
```

OPEN #s (E,SS+4)> Om een stream aan een channel te koppelen en via deze stream gegevens te transporteren. (zie CLOSE en H.2)

OUT p,w (E,SS+o)> Om een waarde w via poort p naar een randapparaat te sturen.
In de handleiding vindt u de diverse poortadressen.

OUT 254,2 maakt de rand rood. Dit duurt net zolang tot u een toets indrukt, dan neemt de rand weer de vastgestelde kleur aan.

OVER 1/0 (E,SS+n)> Om over tekst heen te printen in plaats van erachter. Als direct kommando geldt het voor alle tekst en binnen een PRINT statement voor de te printen gegevens. 1 = aan, 0 = uit

PAPER k (E,SS+c)> Om de achtergrondkleur van de tekst te veranderen. 0 >= k >= 7

PAUSE t (K,m)> Wacht tx(50/60) seconden als t<>0. Als t=0 wordt er gewacht tot er een toets wordt ingedrukt.

PLOT x,y (K,q)> Om het punt waar vandaan begonnen moet worden een lijn te trekken te bepalen. In het begin is dit (0,0). (zie H.3)

PLOT x,y;$ (BB) > Zelfde keyword, maar u kan nu OVERAL op het scherm een stuk tekst PLOTten.

```
10 LET i=175: LET a$="Plot a$"
20 DO WHILE i>1
30 PLOT i,i;a$: LET i=i-1
40 LOOP
```

POKE a,w (K,o)> Om in een byte op adres a een waarde w te plaatsen. Zo kunt u bijvoorbeeld de 'Scroll ?' boodschap uitschakelen door POKE 23692,255. (zie H.3)
0 >= w >= 255

POKE a,w$ (BB) > Hetzelfde, maar uitgebreid. We kunnen vanaf de byte op adres a een reeks bytes met 1 POKE statement een waarde geven, door alle waarden in een string te plaatsen en ze slechts 1 byte groot te laten zijn. We kunnen niet gewoon getallen invoeren, maar we moeten de ASCII code invoeren. (zie CHR$ bij functies)

```
10 LET a$="": FOR i=1 TO 32
20 LET a$=a$+CHR$ 255: NEXT i
30 PRINT a$
```

U ziet 6 rijen ' COPY ' staan.

```
40 FOR i=16384 TO 18431 STEP 32
50 POKE i,a$: NEXT i
```

U ziet het bovenste deel gevuld worden met lijnen. Elke byte wordt met 255 gevuld (11111111) want de ASCII code van COPY is 255.

DPOKE a,w (BB) (G,p)> Om waarden van 0 >= w >= 65535 op de correcte wijze over twee adressen te verspreiden. (D = dubbel)

```
10 DPOKE 16384,65535   is identiek aan

20 POKE 16384,65535-INT(65535/256)x256
30 POKE 16385,INT(65535/256)
```

POP (BB) (G,q)> Om een RETURN adres van de GOSUB/DO-LOOP/PROC stack te wissen, zodat er geen RETURN meer mogelijk is.
Indien u achter POP een numerieke variabele plaatst, wordt het nummer van de regel waar naar teruggesprongen zou worden in de variabele geplaatst.

```
50 GOSUB 100
.....
100 POP reg
110 PRINT "Deze subr. werd van regel";
    reg;" aangeroepen."
120 GO TO reg+1
```

PRINT (K,p)> Hiermee sturen we de gegevens naar een randapparaat, bijvoorbeeld het beeld-scherm of een printer.
Zonder PRINT kan u niets beginnen.
Achter PRINT kan u eigenlijk alles zetten, van een stukje tekst tot complexe berekeningen, waarvan het antwoord dan wordt gePRINT.
(zie H.1, H.2 en H.3)

PROC n (G,2)> Hiermee roepen we een procedure aan die ergens in het programma met DEF PROC n is gedefineerd. PROC is als GOSUB.
n = naam. Geen aanhalingstekens.

RANDOMIZE w (K,t)> Om de RND funcie van een bepaald punt af getallen te laten produceren.

```
10 RANDOMIZE 7
20 FOR i=1 TO 10
30 PRINT INT (RND x 10)
40 NEXT i: PRINT''
50 RANDOMIZE 7
60 FOR i=1 TO 10
70 PRINT INT (RND x 10)
80 NEXT i
```

U ziet twee rijen met dezelfde getallen. Dit omdat we bij de tweede lus ook RND vanaf 7 hebben laten beginnen dmv RANDOMIZE 7.
Als u regel 50 verwijdert ziet de tweede rij getallen er anders uit.
Op deze manier zijn de getallen nog willekeuriger, vooral als u w door RND laat berekenen. 0 >= w >= 65535

```
10 RANDOMIZE INT (1+RND x 200)
```

READ (E,a)> Om een DATA lijst te lezen. Zie DATA.

REM (K,e)> Commentaarregel. Alles achter REM, op dezelfde regel, wordt als commentaar beschouwd en niet als BASIC.

RENUM (BB) (G,4)> Hiermee kan u op alle mogelijke manieren uw BASIC regels hernummeren.

```
RENUM (100 TO 1000) LINE 1200
RENUM ( TO 1000) STEP 5
RENUM (100 TO ) LINE 1500 STEP 20
```

De eerste RENUM hernummert regels 100 t/m 1000 en laat ze beginnen op 1200.
De tweede hernummert de regels 1 t/m 1000 en telkens 5 verhogend, ipv 10.
De laatste hernummert alle regels vanaf regel 100 en plaatst ze op regel 1500 en verder, steeds 20 verhogend.

RESTORE rnr (E,s)> Om de pointer, welke bij het lezen van een DATA lijst steeds 1 opschuift, weer terug te zetten aan het begin van de DATA lijst op regel rnr, of van alle lijsten indien rnr weggelaten is.

```
10 DATA 1,2,3,4,5,6,7,8,9,10
20 DATA 11,12,13,14,15,16,17,18,19,20
30 FOR i=1 TO 20
40 READ a: PRINT a;", ";
50 NEXT i
60 READ a: PRINT a
```

Nu zal u in regel een 'Out of DATA' error krijgen, omdat er maar 20 gegevens opgenomen zijn in de lijst en u wilt de 21e printen. Wanneer u de hele lijst nog eens wilt printen kan u RESTORE of RESTORE 10 geven, en als u slechts de tweede wilt RESTORE 20. De computer zet de pointer dan aan het begin van de genoemde regel.
(zie READ en DATA)

RETURN (K,y)> Hoort bij GOSUB. Na een GOSUB wordt het terugkeeradres op de GOSUB stack bewaard en zodra er een RETURN wordt tegengekomen, gaat de computer verder op het adres wat bovenin de stack staat en wist daarna het adres van de stack. Er mogen meerdere RETURN's in een subroutine voorkomen tegenover 1 GOSUB aanroep. De subroutine zelf mag natuurlijk n maal worden aangeroepen, maar mag NOOIT zichzelf aanroepen.
(zie GOSUB)

ROLL (BB) (G,r)> Hiermee kan u een scherm of een deel van het scherm over het scherm laten bewegen. Met ROLL zal wat aan de rand verdwijnt aan de tegenovergestelde rand weer verschijnen.
Wat verplicht is achter ROLL is de richting waarin het moet bewegen.
(zie BB handleiding)
Daarachter kan u het aantal pixels geven wat per ROLL moet worden verplaatst. Daarachter kan u een 'window' definieren waarbinnen geROLLed moet worden.

```
10 PRINT AT 1,1;"BOODSCHAPPEN"
20 PLOT 7,169: DRAW 97,0: DRAW 0,-9:
   DRAW -97,0: DRAW 0,9
30 DO
40 ROLL 8;8,170;12,8: PAUSE 40
50 LOOP
```

RUN rnr (K,r)> RUN is identiek aan CLEAR: GO TO rnr. Als rnr de eerste regel van het programma is mag ze weggelaten worden.

SAVE n wat (K,s)> Om gegevens naar tape of disk te schrijven. (zie H.2)

```
n = "naam";
wat = niets:     BASIC programma.
wat = LINE rnr: BASIC programma wat
      uitzichzelf start op rnr als het
      geladen wordt.
wat = DATA a$() of DATA a(): gegevens
wat = CODE st,l: l bytes, vanaf adres
      st. St en l zijn verplicht.
wat = SCREEN$ : het beeldscherm.
```

SCROLL (BB) (G,s)> Zelfde als ROLL maar nu verdwijnt de data wel. SCROLL kan gewoon worden gebruikt, dan schuift het hele beeld-scherm 1 regel omhoog.

SORT (BB) (G,m)> Om strings of tabellen te sorteren.

```
10 LET a$="gjasgshahgsgsahtegswakl"
20 SORT a$: PRINT a$
```

A$ zal er dan zo uitzien:

```
aaaaeggggghhhjklssssstw
```

```
10 DIM a$(10,10)
20 LET i=1: DO WHILE i<11
30 INPUT a$(i): LOOP
40 SORT a$: PROC lijst
50 SORT a$()(5 TO ): PROC lijst
60 SORT a$()( TO 5): PROC lijst
70 SORT a$( TO 5)(5 TO ): PROC lijst
....
200 DEF PROC lijst
210 CLS: LET i=1: DO WHILE i<11
220 PRINT a$(i): LOOP
230 END PROC
```

SPLIT (BB) (K,SS+w)> SPLIT is het tegenovergestelde van JOIN. Het woord SPLIT bestaat niet, maar het teken <> wordt gebruikt. U plaatst dit teken NA een : in een BASIC regel en de regel wordt gesplitst.

STOP (K,SS+a)> STOPt het programma. Kan niet als direct kommando worden gegeven, maar moet onderdeel uitmaken van een BASIC regel. Met CONTINUE gaat het programma weer verder.

TRACE rnr (BB) (G,t)> Hiermee kan u uw BASIC programma debuggen. TRACE gosubt regel rnr. Bij TRACE horen twee variabelen die u niet anders dan met TRACE mag gebruiken.
line = bewaart het regelnummer waar de computer mee bezig is.
stat = bewaart het statementnr binnen de regel.

VERIFY n wat (K,SS+r)> Zie LOAD. VERIFY is om te controleren of de gegevens correct zijn weggeschreven naar tape of disk. (zie H.2)

§2. Functies.

Functies zijn NIET direct in te geven en worden ALTIJD samen met een kommando en RECHTS van het '=' teken gebruikt. U kan dus niet zeggen:

```
INT a
```

maar wel:

```
PRINT INT a
LET b=INT a
```

De gegevens hoeven niet tussen haakjes te worden geplaatst als het constanten zijn, WEL als het berekeningen zijn.
Net zoals in §1 worden ook hier de Beta BASIC functies besproken. De meeste Beta BASIC functies worden met FN (E,SS+2) aangeroepen. We zullen dit als een extra mode weergeven met een hoofdletter F en daarachter wat u vervolgens in moet toetsen.
AND ziet er dan als volgt uit: .F,a(.
Het haakje hoort bij de functie.

ABS w (E,g)> Geeft de absolute waarde van w. Absolute waarde wil zeggen: de zuivere waarde van het getal ongeacht het + of - teken.

ACS w (E,SS+w)> Geeft de waarde x terug in radialen waarmee de COS van een hoek is gevonden.

```
x=ACS (COS x)
```

AND (K,y)> Logische operand om vergelijkingen uit te voeren. AND kan op diverse manieren worden gebruikt, in IF ... THEN, LET, PRINT, enz.

```
10 LET c=0: INPUT a;" en ";b
20 IF a=1 AND b=1 THEN ........
30 LET c=(1 AND a<0 AND b<0): GO TO 40
   +(-30 AND c)
40 LET a$=("000" AND a<10)+("00" AND (
   a>9 AND a<100)+("0" AND a>99 AND a<
   1000)+STR$ a
```

Regel 20 lijkt me duidelijk. Maar nu regel 30 en 40. In regel 30 staat eigenlijk:

```
IF a<0 AND b<0 THEN LET c=1: GO TO 10
```

Wat er gebeurt is dat de computer de vergelijking uitvoert en in het vlagregister de SIGN bit zet als de vergelijking WAAR is. Indien de SIGN bit 1 is zal c de waarde krijgen welke binnen de haakjes is genoemd, in dit geval 1. Hetzelfde gebeurt in het GO TO deel. Als c=1 dan zal de waarde 30 van 40 worden afgetrokken en zal er naar regel 10 worden gesprongen. Hiermee is regel 40 ook uitgelegd.

AND (x,y) .F,a(.> Logische AND op de bits van de getallen x en y, waarbij y het masker is waarmee de AND wordt uitgevoerd. Een AND operatie is om bits te bewaren (1 laten) en te schonen (0 maken). Zodra in x EN y dezelfde bits 1 zijn, is het resultaat een 1. In alle andere gevallen een 0. Zo kunnen we van een byte bits 0 maken en de rest hetzelfde laten.

```
     10110111
AND  11101110
-------------
     10100110
```

(0 <= x <= 65535 en 0 <= y <= 65535)

ASN w (E,SS+q)> Geeft de waarde x terug in radialen waarmee de SIN van een hoek is gevonden.

x=ASN (SIN x)

ATN w (E,SS+e)> Geeft de waarde x terug in radialen waarmee de TAN van een hoek is gevonden.

x=ATN (TAN x)

ATTR (y,x) (E,SS+1)> Geeft de inhoud van de byte van het attributengeheugen van PRINT positie (y,x), waarbij y het regelnummer is en x het kolomnummer. (zie H.3)

PRINT ATTR (0,0) en
PRINT PEEK 22528 geven hetzelfde.

BIN (E,b)> Hiermee kan men een binair getal omzetten naar decimaal. NIET andersom. Men MOET de voorloop nullen en enen stuk voor stuk intikken achter BIN.

```
                    PRINT BIN 0011111
                    PRINT BIN 10101010
                    LET a=BIN 11111
                    INPUT a
                    U typt: BIN 11011011
```

BIN$(w) .F,b+$.> Geeft de binaire representatie van een decimaal getal w terug in een string.

```
                    PRINT BIN$(255)
                    en er zal 11111111 verschijnen.

                    LET a$=BIN$(3): PRINT a$
                    en er zal 00000011 verschijnen.
```

CHAR$(w) .F,c+$.> Deze functie converteert het decimale getal w in een string van twee karakters lang volgens de formule welke bij DPOKE is besproken.

```
                    DPOKE adr,16384  en
                    POKE adr,CHAR$(16384) geven hetzelfde.
```

Deze functie maakt het mogelijk om snel en geheugen besparend grote getallen op te slaan.
(0 <= w <= 65535)

CHR$ w (E,u)> Deze functie geeft het karakter terug in een string waarvan w de ASCII code is. (0 <= w <= 255)

```
                    PRINT CHR$ 65    geeft de 'A'
                    LET a$=CHR$ 48   zet de '0' in a$.
```

CODE "k" (E,i)> Het omgekeerde van CHR$. Deze functie geeft de ASCII code welke bij k hoort.

```
                    PRINT CODE "A"              geeft 65.
                    LET a$="A":LET c=CODE a$  zet 65 in c.
```

COS r (E,w)> Geeft de COS van r radialen.
PI radialen = halve cirkel.
PI/2 radialen = kwart cirkel.

COSE(r) .F,c+(.> Idem als COS, maar minder precies en 6x sneller.

DEC($) .F,d+(.> Geeft als resultaat de decimale waarde van een hexadecimaal getal, welke in een string $ bewaart wordt.

```
                    PRINT DEC("FF")
                    LET a$="0FAC": PRINT DEC(a$)
```

EXP w (E,x)> Geeft als resultaat het grondtal van de natuurlijke logarithme e tot de macht w: e^w

FILLED() .F,f+(.> Geeft het aantal pixels dat door het laatste FILL kommando is gevuld.

FN n(w) (E,SS+2)> Hiermee roept men een met DEF FN gedefinieerde functie aan welke de met w wordt geevalueerd. (zie DEF FN)

HEX$(w) .F,h+$.> Hiermee wordt een decimaal getal omgezet in een hexadecimaal getal wat in een string wordt geplaatst.
(0 <= w <= 65535)

PRINT HEX$(255) geeft 'FF'.

IN adr (E,SS+i)> Leest de poort op adres adr op processor nivo. IN is zoals PEEK, maar IN leest een poort waar gegevens van de randapparatuur binnenkomen.

INKEY$ (E,n)> Leest het toetsenbord slechts 1 keer. Dat wil zeggen dat INKEY$ NIET wacht tot u een toets indrukt en nooit meer dan 1 karakter accepteert. (zie H.2)

INSTRING() .F,i+(.> Hiermee kan u in een string zoeken naar een teken of een groep tekens. Als eerste geeft u de positie op waar het zoeken moet beginnen, vervolgens de string waarin gezocht moet worden en daarna het zoekargument.
Het resultaat is de positie van de eerste letter van het te zoeken item binnen de string. Dan wordt er ook gelijk gestopt. Als u verder wilt zoeken moet u bij het resultaat 1 optellen en dat de startpositie maken.

```
10 LET a$="nbnbmbmbmbmbmnbnhghggrtrtre
     sdwautwnjTEKSTbnvngghnTOKSThfd"
20 LET st=1: REM start bij 1e letter
30 LET resul=INSTRING(st,a$,T#KST)
40 PRINT resul
50 IF NOT resul THEN GO TO 70
60 LET st=resul+1: GO TO 30
```

U ziet in het zoekargument het #. Dit betekent 'zoek naar T KST' en de 2e letter doet er niet toe.

INT w (E,r)> Geeft het getal zonder zijn decimalen. De functie rond NIET af.

```
INT 4.67        is 4
4.67-INT 4.67   is .67
```

LEN $ (E,k)> Geeft de lengte van string $.

```
10 LET a$="gdhetuqgaxpooetfwgyejsgetr"
20 PRINT LEN a$     is 26.
```

LN w (E,z)> Geeft de natuurlijke logarithme van w. Het grondtal van de natuurlijke logarithmen is e = 2.7182818 . We vinden de waarde e door EXP 1, want e^1 = e.

MEM() .F,m+(.> Geeft het aantal bytes vrij geheugen. Indien Beta BASIC niet in het geheugen zit kan met PRINT 65535-USR 7962 het aantal vrije bytes worden opgevraagd.

MEMORY$() .F,m+$.> Geeft de inhoud van het hele geheugen van de computer als een string, welke met POKE adr,$ weer in het geheugen kan worden geladen. Om een bepaald deel van het geheugen in een string te bewaren, bijvoorbeeld het beeldscherm, geeft men achter MEMORY$() het begin en eindadres van het blok.

```
10 LOAD "" SCREEN$
20 LET a$=MEMORY$()(16384 TO 23295)
30 CLS : POKE 16384,a$
```

Op deze manier kan u ook stukjes zelfgeschreven machinetaal opslaan in strings en naar gelang het gebruik op een bepaald adres POKEn.
De strings kan u apart, als data, laden en saven.

MOD(x,y) .F,v+(.> Geeft de rest van de deling x/y.

```
PRINT MOD(10,3)
```

De uitkomst is 1 want 10/3=3, rest 1.

NOT w (K,SS+s)> Een NOT operatie vergelijkt of w=0 of dat w<>0. De vergelijking is WAAR als w=0 en NIET WAAR als w<>0.

NUMBER($) .F,n+(.> Converteert de string welke gemaakt is met CHAR$() weer naar een decimaal getal.

OR (K,SS+u)> Logische operatie. Bij OR kan 1 van de twee operanden WAAR zijn, in tegenstelling tot AND.

```
10 INPUT a;" en ";b
20 IF a=1 OR b=1 THEN GO TO ...
30 LET c=(1 AND (a=0 OR b=0))
...
```

OR (x,y) .F,o+(.> Logische operatie op x en y op binair nivo. Een bit is 1 als een bit in x OF in y 1 is. Zijn ze beide 0, dan is het resultaat ook een 0.

```
PRINT OR(181,79)    is 255.

   10110101 (181)
OR 01001111 (79)
------------
   11111111 (255)
```

PEEK adr (E,o)> Leest de inhoud van een byte op adres adr. Met POKE zet u een waarde in een byte en met PEEK leest u deze waarde.

DPEEK(adr) .F,p+(.> Leest waarden die, omdat ze groter dan 255 zijn, over twee adressen zijn verspreid, nl. (adr) en (adr+1). DPEEK is hetzelfde als, in BASIC:

```
PEEK adr + 256 * PEEK (adr+1)
```

PI (E,m)> Levert de waarde 3.14159265.

POINT (x,y) (E,SS+8)> De x en y zijn pixelcoordinaten. POINT betekent zoveel als 'Is die pixel aan of uit?'. Indien de pixel INK is, dan is het resultaat van POINT 1, anders is het 0.

```
10 PRINT POINT (0,10)
20 PLOT 0,10
30 PRINT POINT (0,10)
```

RND (E,t)> Levert een op een willekeurig moment berekend getal g. (0 <= g <= 1)
De berekening welke wordt uitgevoerd is:

```
((75 mod 65537)-1)/65535
```

RNDM(w) .F,r+(.> Werkt hetzelfde als RND, maar is 6x sneller en biedt de faciliteit om willekeurige getallen te geven tussen 0 en (w+1).

```
PRINT RNDM(1000)     levert een getal
                     tussen 0 en 1001.
```

SCREEN$ (x,y) (E,SS+k)> De x is een regelnummer en de y een kolomnummer. Het is een functie met twee gebruiksmogelijk heden. De eerste is bij een SAVE opdracht om het beeld-scherm op band te zetten.
De tweede werkt ongeveer als POINT, maar nu om een karakter van het beeld-scherm te lezen. (zie H.1)

```
PRINT AT 11,11;"AB"
PRINT SCREEN$ (11,11)
PRINT SCREEN$ (11,12)
```

Wat met deze functie NIET kan is het herkennen van graphics. In plaats van een graphic wordt er een spatie geprint.

SCRN$(x,y) .F,k+$.> Deze functie werkt als SCREEN$ met dit verschil dat deze WEL graphics herkent en dus geen spatie print.

SGN w (E,f)> Deze functie geeft aan of de variabele w, of de constante w, negatief, nul of positief is. Het resultaat is respec-tievelijk -1, 0 en 1.

```
LET a=-1*(RND *10)+5
PRINT a,SGN a
```

SIN w (E,q)> Geeft de SINus van w radialen.

SINE(w) .F,s+(.> Geeft de SINus van w radialen tot op 4 cijfers achter de komma correct, maar 6x sneller dan SIN.

SQR w (E,h)> Geeft de vierkantswortel van w.

```
PRINT SQR 4    geeft 2, want 2*2=4.
```

STR$ w (E,y)> Geeft het getal w terug als een string met w als inhoud.

```
10 LET a=3: LET b=4
20 PRINT a+b                    geeft 7.
30 PRINT STR$ a+STR$ b          geeft 34.
```

Vrij vertaald:
```
20 PRINT 3+4
30 PRINT "3"+"4"
```

STRING$(a,$) .F,s+$.> Geeft a*$.

```
PRINT STRING$(64,"*")
```

Dit print 64 sterretjes achterelkaar. De BASIC vertaling van deze functie ziet er zo uit:

```
10 LET a=64: LET a$="*"
20 FOR i=1 TO a
30 PRINT a$;
40 NEXT i
```

TAN w (E,e)> Geeft de TANgens van w radialen.

TIME$() .F,t+$.> Deze functie hoort bij CLOCK. De tijd die de klok op een bepaald moment heeft kan met TIME$ in een stringvariabele worden geplaatst.

```
LET a$=TIME$()
```

UNTIL cond .G,k.> Zie DO.

USING $ .G,u.> Met USING kunnen we van getallen met cijfers achter de komma, de komma op dezelfde positie printen door het uitvoerformaat te specificeren. Dit doen we met het # (spaties) of met de nul (nullen).

```
10 LET a=RND *10
20 PRINT USING "##.##";a
30 PRINT USING "00.00";a
```

De variabele a heeft bijvoorbeeld de waarde 5.3046758 .
In regel 20 zal a er zo uitkomen:

```
 5.30
```
en in regel 30 zo:
```
05.30
```

USING$($,w) .F,u+$.> Geeft het resultaat dat met USING zou zijn bereikt terug als een string.

```
10 LET a=RND *10
20 LET a$=USING$("##.##",a)
30 LET b$=USING$("OO.OO",a)
40 PRINT a$,b$
```

USR w/$ (E,l)> De waarde w stelt een adres voor. De $ is kan de letter "a" t/m "u" zijn. In het eerste geval hebben we het over het aanroepen van een machinetaal programma en in het tweede geval over user defined graphics. (zie H.3)
U kan op diverse manieren een machine-taal programma starten:

```
LET a=USR w;
PRINT USR w;
GO TO USR w;
RANDOMIZE USR w.
```

VAL $ (E,j)> Maakt van de string $ een numerieke waarde als de string uit cijfers en maximaal 1 punt bestaat.

```
LET a=VAL "1495"
LET a$="1495": LET a=VAL a$
```

Het gebruik van getallen in strings is veel goedkoper dan het getal als numerieke waarde te bewaren. Zo kan u bijvoorbeeld:
GO TO VAL "3000" zeggen, of RANDOMIZE USR VAL "32000".

VAL$ $ (E,SS+j)> Eigenlijk hetzelfde als VAL, maar het resultaat is weer een string en geen numerieke variabele. Waar deze functie voor nodig is mag u zelf ontdekken, misschien dat u er een toepassing voor kan vinden.

WHILE cond .G,j.> Zie DO.

XOR(x,y) .F,x+(.> Dit is een EXCLUSIVE OR. Dat wil zeggen dat als dezelfde bits, in x en y, dezelfde waarde hebben, dus beide 1 of O, dan is het resultaat een O, anders een 1. (zie ook AND en OR)

PRINT XOR(181,79) is 250.

```
    10110101 (181)
XOR 01001111 (79)
-------------
    11111010 (250)
```

§3. Beta BASIC variabelen.

Inleidng.

Er zijn diverse variabelen die met Beta BASIC samengaan en in BASIC programma's zijn te gebruiken. Er zijn vier variabelen voor de x-as en y-as, welke van belang zijn voor het maken van tekeningen op het beeldscherm en er zijn drie variabelen welke aangemaakt worden door TRACE en ON ERROR. Ik heb gezegd dat u deze niet voor andere doeleinden mag gebruiken. Dit is niet helemaal waar, maar het is verstandiger ze enkel met de functies te gebruiken. We beginnen met de drie variabelen voort TRACE en ON ERROR.

- ERROR of error > Deze variabele bewaart de code van de laatst opgetreden error. Als u in de handleiding kijkt, appendix B, dan ziet u dat elke foutboodschap een code heeft die loopt van 0 t/m R. Beta BASIC leest de systeem variabele op adres 23610 en telt er 1 bij op zodat de correcte code in ERROR komt te staan.
  Deze variabele wordt gebruikt als u met ON ERROR werkt.

- LINE of line > Niet te verwarren met de functie LINE. LINE wordt gebruikt bij ON ERROR en bij TRACE. De variabele houdt bij ON ERROR het regelnummer vast waar de fout in optrad. Met TRACE bewaart LINE keer op keer de regel waar het programma op dat moment mee bezig is.

- STAT of stat > Deze variabele houdt het nummer van het statement vast van de regel welke in LINE staat. Een programmaregel kan meer dan 1 opdracht bevatten en de computer houdt met een teller bij met welke opdracht van die regel hij bezig is. Bij een nieuwe regel wordt weer opnieuw geteld.
  Deze variabele wordt ook bij beide functies gebruikt.

Dan nu de variabelen welke voor het gebruik van CICLE, PLOT, DRAW en FILL van belang 4ijn. Zoals u in H.1 en H.3 hebt kunnen is het beeldscherm opgebouwd uit 256 punten in horizontale richting, de x-as, en 176 punten in verticale richting, de y-as. De oorsprong van dit assenstelsel is het punt (0,0) en ligt in de linker hoek van het beeldscherm. Het is dus niet mogelijk om links van de y-as of onder de x-as te werken. Met de volgende variabelen is dat wel mogelijk.

- XOS of xos > Met deze variabele kan u de oorsprong van de x-as verleggen naar een punt tussen 0 en 256.

```
LET xos=128
```

Hiermee is de oorsprong van de x-as 129 punten naar rechts verlegd. Normaal kon u 256 punten naar rechts benutten, nu is dat veranderd in 127 punten naar rechts en 128 punten naar links.

- YOS of yos > Hetzelfde verhaal als voor de x-as, maar nu voor de y-as.

```
LET yos=88
```

Hiermee is de oorsprong van de y-as 89 punten naar boven verlegd. Normaal kon u 176 punten naar boven benutten, nu is dat veranderd in 87 punten naar boven en 88 punten naar beneden.

- XRG of xrg > Hiermee kan u de lengte van de x-as instellen. De berekening blijft over het oorspronkelijke aantal pixels gelden. Zo kan u tekeningen welke normaal het hele scherm in beslag nemen verkleinen of uit proporties trekken door de y-as een geheel andere lengte te geven.

```
10 INPUT "Nieuwe oorsprong x-as:";ox
20 INPUT "Nieuwe oorsprong y-as:";oy
30 LET xos=ox: LET yos=oy
40 INPUT "Lengte x-as:";lx;", y-as:";ly
50 LET xrg=lx: LET yrg=ly
60 PLOT 0,0: DRAW 255,0: DRAW 0,175: DRA
   W -255,0: DRAW 0,-175: DRAW 255,175
70 GO TO 10
```

De straal bij CIRCLE blijft normaal.

- YRG of yrg > Hetzelfde verhaal als voor XRG, maar nu voor de y-as.

§4. Systeem variabelen.

Zoals een BASIC programma werkt met variabelen, zo werkt het monitorprogramma ook met variabelen. We hebben in H.3 al de variabele besproken die het begin van de karakterset bewaart en in H.1 de variabele die bepaald of er 'Scroll?' gevraagd moet worden. Er zijn er nog meer en welke effecten die hebben gaan we nu bespreken.

We zullen eerst nog even de formules herhalen welke nodig zijn om waarden groter dan 255 uit twee adressen te lezen:

```
PEEK adr+256*PEEK (adr+1)
```

en om waarden groter dan 255 over twee adressen te verdelen:

```
POKE adr,(w-256*INT(w/256))
POKE (adr+1),INT(w/256)
```

Daar waar 1 adres is genoemd kunnen geen waarden groter dan 255 worden gebruikt.

Adres:

23556 > Bewaart de ASCII code van de hoofdletter die op de toets staat die wordt ingedrukt.

23560 > Bewaart de ASCII code van het teken wat het laatst is ingetoetst.

23561 > Bewaart de duur, in 1/50e seconden, die een toets moet zijn ingedrukt voordat het teken wordt herhaald. Normaal staat hier 35 in.

23562 > Bewaart de tijd tussen de herhalingen van de toets die is ingedrukt.

23606/7 > Bewaart het begin van de karakterset. Het 1e karakter, de spatie, begint 256 adressen verderop.

23608 > Bewaart de duur van de waarschuwingstoon.

23609 > Bewaart de duur en tevens toonhoogte van de klik als u een toets indrukt.

23610 > Bewaart de code-1 van de foutboodschap.

23617 > Deze variabele bewaart de mode waarin het systeem verkeert. Dit kan L,E of G mode zijn.

0 = L mode
1 = E mode
2 = G mode

Met hogere waarden veranderd ook de cursor.

23624 > Bewaart de BORDER kleur en de attributen voor het onderste deel van het scherm.

23625/6	> Bewaart het regelnummer waar de cursor zich bevindt. In plaats van met de pijltoetsen van bijvoorbeeld regel 10 naar regel 1000 te lopen, poked u 1000, volgens bovenstaande formule in deze adressen.
23658	> Vlagregister. Bit 3 is wel/geen hoofdletters. POKE 23658,8 en u bent in hoofdlettermode.
23675/6	> Bewaart het adres van de 1e UDG.
23677	> Bewaart de x coordinaat van het laatst geplotte punt.
23678	> Bewaart de y coordinaat van het laatst geplotte punt.
23684/5	> Bewaart een adres van het beeldschermgeheugen waar het volgende geprint moet worden. (zie H.3)
23686/7	> Idem als bij 23684/5 maar nu voor het onderste deel van het beeldscherm.
23692	> Telt scrolls. Met waarden groter dan 1 wordt er niet gestopt, maar doorgescrolled.
23693	> Bewaart de kleuren zoals die met PAPER, INK, BRIGHT, enz. gezet zijn.
23697	> Hier wordt OVER, INVERSE, INK 9 en PAPER 9 in bewaart. Bits 1,3,5 en 7 zijn voor permanent en bits 0,2,4 en 6 voor tijdelijk.

Dit is het einde. Deze cursus is bedoeld als ondersteuning en niet om te zorgen dat u de computer uit uw hoofd kent terwijl u 'm nog nooit hebt gezien. U moet veel zelf doen en ontdekken. Wat u met wat u weet gaat doen kunnen wij u niet voorschrijven en deze syllabus is bedoeld als hulpmiddel om in het begin even snel uw geheugen op te frissen. Voor meer details en als u diep in de computer wilt duiken volgt hier een lijstje van aanbevolen literatuur.

- de handleiding zelf.

- Zakboekje voor de ZX-Spectrum, Wessel Akkermans, Kluwer.

- Understanding your Spectrum, Dr. Ian Logan, Melbourne House.

- Cursus Z-80 assembleertaal, Roger Hutty, Acadamic Service.