sinclair

ZX SPECTRUM

En regnbue i en datamat

– et kursus i BASIC-programmering

ZX SPECTRUM

En regnbue i en datamat

af Steven Vickers
i dansk bearbejdelse

– et kursus i BASIC-programmering

Tryk: Hellbrandt Offset/Bogtryk A/S

ISBN 87-88267-06-7

Forfatter: Steven Vickers
Oversat til dansk af Claus Dissing
Redigeret af Svend Garbarsch

Forsideillustration: John Harris

Til Spectrum-købere:

Kast endelig ikke den originale transportkasse til ZX Spectrum bort. Den er en god transportkasse, hvis du skal rejse omkring med din datamat og er *absolut påkrævet*, hvis du af en eller anden grund skal sende maskinen til leverandøren.

Alle henvendelser om teknisk service i henhold til garantien skal rettes til serviceafdelingen i Karup, hvis fulde adresse er:

ZX-data
Serviceafdelingen
7470 Karup

telefon: 07 - 10 15 09 (telefontid mandag-fredag kl. 17–18). Varer, der ikke er købt hos en forhandler med autorisation fra ZX-data, er ikke omfattet af nogen garanti og repareres kun »på regning«.

Indhold

Forord

Denne instruktionsbog er beregnet på den britiske mikrodatamat ZX Spectrum, men da bogen er et komplet kursus i BASIC-programmering, kan den også anvendes til selvstudium eller undervisning i dette datamatsprog – udbyttet ved brug af en anden datamat eller slet ingen bliver dog mere begrænset, end hvis bogen anvendes sammen med en Spectrum.

Navnet Spectrum har datamaten fået eller den mangefarvede stribe, der opstår, hvis lys ledes gennem et prisme eller en anden form for lysbrydende materiale. Vi kender alle prismevirkningen fra dagligdagen. Den skabes f.eks. hyppig af en regnbyge i luften. Dråberne bryder sollyset, og der dannes en regnbue – Naturens egen Spectrum ...

ZX Spectrum, der er fremstillet af den britiske datamatpioner Clive Sinclair, bringer så at sige regnbuen ind i dataverdenen. Lad os håbe, at Spectrum også gør dine oplevelser med datamaten til en festlig og farverig fornøjelse.

Den instruktionsbog, du nu går i gang med, er opbyggel, så den er anvendelig for såvel begyndere som viderekomne.

De første ni kapitler er en introduktion, holdt i et forhåbentlig letforståeligt sprog. Disse kapitler indeholder nogle simple anvisninger på, hvordan datamaten sluttes til og anvendes.

Øvede datamatbrugere kan muligvis nøjes med en hastig gennemløbning af disse kapitler og evt. et lidt mere indgående studium af de tekniske afsnit bagest i bogen, mens begyndere nok bør tyre det hele igennem.

Folk, der i forvejen gennem brug af f.eks. ZX81 kender til Sinclair BASIC, kan springe de første ni kapitler over og starte ved kapitel 10. Her begynder den egentlige og mere detaljerede gennemgang af ZX Spectrum, dens sprog, dens faciliteter og dens brug.

Skønt der kan synes at være ret megen matematik i bogen, bør du absolut ikke lade dig afskrække, hvis du måske ikke er helt stiv i matematik.

Spring endelig ikke de matematiske kapitler over, selv om du ikke forstår dem. Flere af dem indeholder vigtige principper, der mere har med programmering end med matematik at gøre. Læs dem igennem og nøjes med at forstå det, du kan. Du skal se, du fatter nok alligevel mere, end du oprindeligt vilde tro.

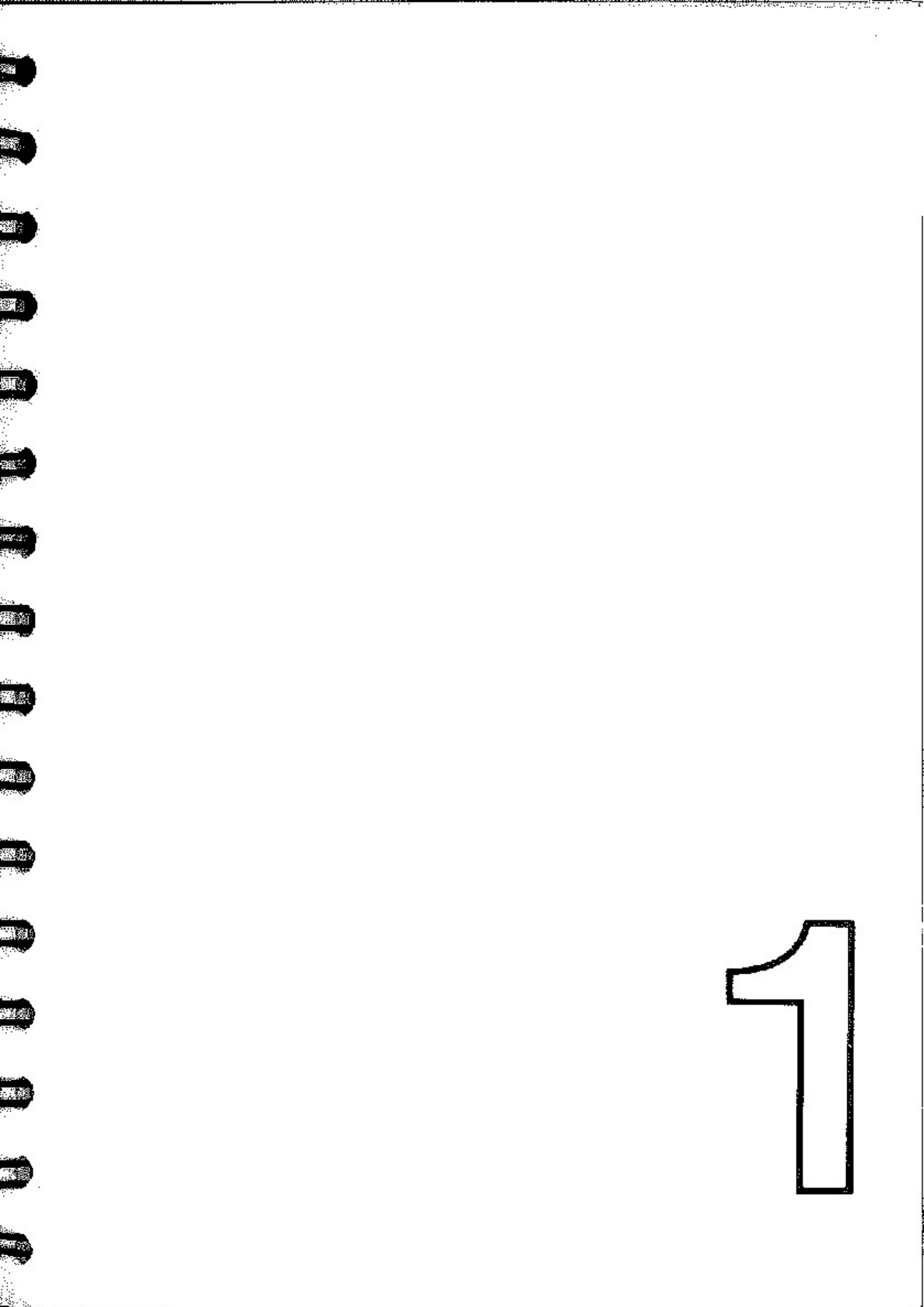
Du må heller ikke forbigå de opgaver, der står i slutningen af en del af kapitlerne. Mange af dem demonstrerer punkter, det er nødvendigt at kende til for at forstå de efterfølgende kapitler.

Men uanset, hvad du gør, så brug først og fremmest din datamat hele tiden. Hvis du spørger dig selv, hvad der mon sker, hvis du gør sådan eller sådan , så er svaret let: Prøv at taste det ind og se, hvad der sker; Hver gang denne instruktionsbog foreslår, at du skal indtaste et eller andet, så spørg altid dig selv: "Hvad kunne jeg skrive i stedet for?" og prøv så at skrive det. På den måde lærer du lidt efter lidt at programmere, og jo mere du selv skriver egne programmer, des bedre kommer du til at forstå datamaten.

Bagest i instruktionsbogen er en række afsnit om opbygningen af datamatens lagre, om datamatens talbehandling og en række programeksempler til belysning af ZX Spectrums anvendelighed!

Og med disse ord: Velkommen til Spectrumland og de regnbuefarvede oplevelser!

Red.



1. Du og din datamat

Din nye datamat er testet og tjækket flere gange fra fabrikken og skulle være klar til brug, lige så snart du har fået sluttet den til et fjernsynsapparat og fået sat strøm til den.

Ved udpakningen fandt du følgende ting i pakken:

1. Denne bog, der er et komplet kursus i datamaten og BASIC.

2. Din Spectrum.

3. En netdel, der transformerer strømmen til de 9 volt, Spectrum kræver.

4. En ca. to meter lang ledning, der forbinder datamaten med dit fjernsyns antennestik.

5. Et ca. 75 cm langt ledningspar med 3,5 mm jackstik i begge ender. De skal forbinde din datamat med din kassettebåndoptager.

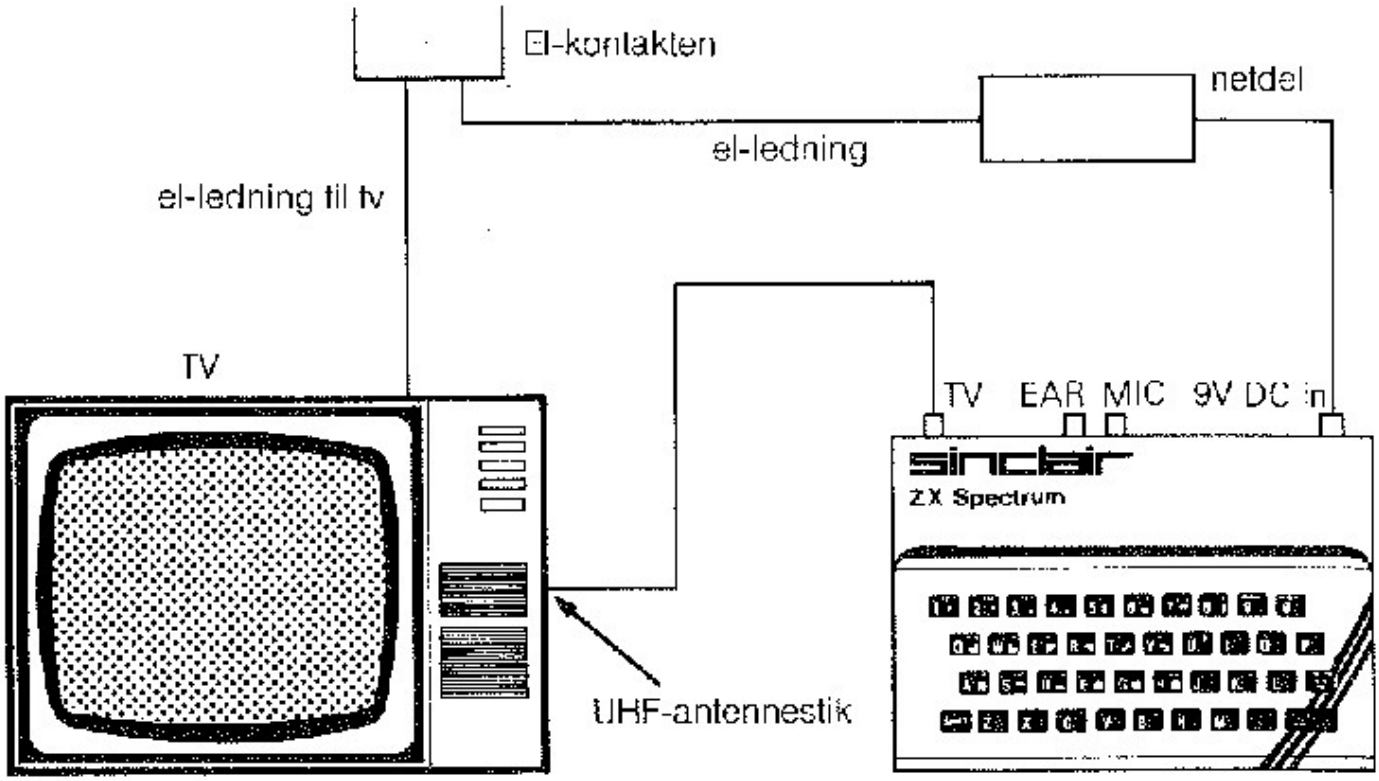
6. Et prøvebånd, der præsenterer Spectrum, dens sprog og dens funktioner.

Inden vi nu går videre, må du have fat i et tv-apparat, der har UHF, så du kan finde kanal 36 – den kanal, Spectrum sender på. Du kan teoretisk godt anvende Spectrum uden fjernsyn, men så vil du arbejde helt i blinde, idet du ikke kan se, hvad datamaten foretager sig.

Det tv-apparat, du skal bruge, skal helst være et farvefjernsyn, idet Spectrum jo er en farvedatamat. Men du kan udmærket anvende et sort-hvidt. Så vil du blot opleve Spectrums farver som sort, hvidt og seks forskellige afskygninger af gråt.

Se nu nærmere på siden af din Spectrum. Der er tre jackstik. De er mærket "9V DC IN", "EAR" og "MIC". Der er også et noget større tv-stik, og desuden er der et datastik bagi. Det bruges til at sætte ekstraudstyr som f.eks. en printer til datamaten.

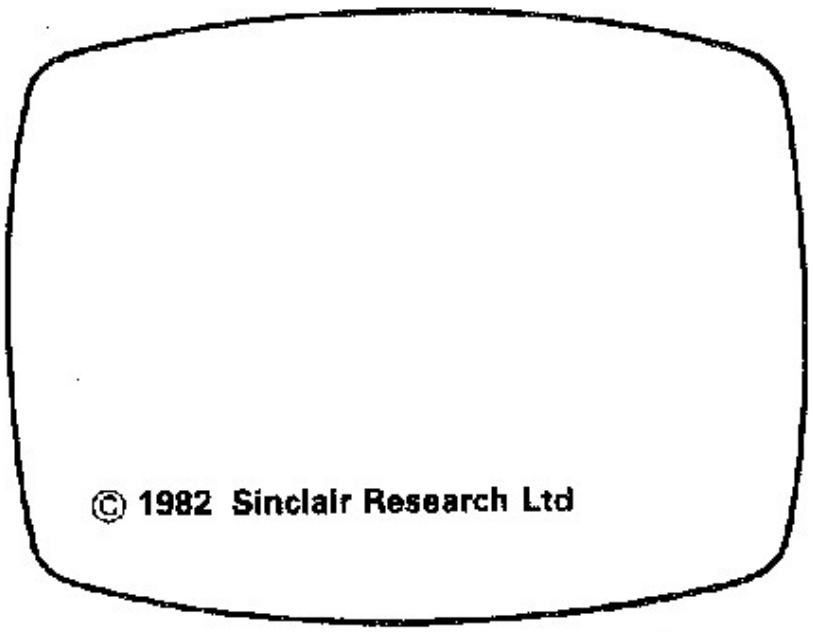
Du skal forbinde datamaten, som vist på tegningen næste side.



Hvis dit fjernsyn har to antennestik, mærket UHF og VHF, skal du bruge UHF-stikket.

Nu skal du tænde for strømmen til din datamat og dit fjernsyn. Du vil sandsynligvis se et gråligt flimmer på skærmen, fordi dit fjernsyn skal indstilles. Prøv at finde frem til kanal 36.

Når datamaten er tændt og fjernsynet rigtigt indstillet, skal du se dette billede på skærmen:



Hvis dit fjernsyn nu står og hvæser, så skru helt ned for lyden.

Mange tv-apparater har en række trykknapper, der stiller ind på forudvalgte tv-

kanaler. Hvis du har sådan et, så prøv at finde en knap, du normalt ikke bruger, og indstil den permanent på kanal 36. Så skal du sandsynligvis aldrig mer igennem denne procedure igen – altså medmindre du bytter tv.
Hvis du bytter din datamat – f.eks. med en kammerats – vil du ofte opdage, at dit fjernsyn må finindstilles, for at den anden datamats billede skal stå skarpt. Det skyldes, at der kan være små variationer fra datamat til datamat i nøjagtigheden af tv-signalet. Det er muligt, at du skal stille lidt på kontrast, lysstyrke og farveintensitet for at få det helt perfekte Spektrumbillede frem på skærmen.
De data, du kan lasle ind i din Spectrum og lade den forarbejde, går tabt, når du slukker for strømmen. Du kan opbevare dem til senere brug ved at spille dem over på en båndoptager. De fleste kassettebåndoptagere vil passe fint sammen med din Spectrum. Det kan du læse mere om i kapitel 6 og 29.
Du kan også via kassettebåndoptageren forsyne din Spectrum med dataprogrammer, som andre mennesker har lavet. Feks. de programmer, der ligger på prøvebåndet i Spectrumpakken.
Godt. Nok om den sag. Nu har du fået din Spectrum »op at stå«. Så skulle du gerne igang med at bruge den. Hvis du allerede i din utålmodighed har trykket på et par taster, vil du have set, at det fjerner copyright-meldingen fra skærmen. Det er helt i orden. Prøv bare at trykke lidt på må og få på tasterne. **Du kan ikke skade din datamat ved det.**

Hvis du ikke kan komme videre, så husk, at der altid er en nødbremse: Hvis du trækker 9V-stikket ud af datamaten og sætter det til igen, kommer det oprindelige billede med copyright-meldingen tilbage. Men det er og bliver den sidste udvej, for bruger du den, taber du jo alle dine indtastede eller overspillede data.

ADVARSEL: Brug ikke en 16K RAM fra en ZX81 på din ZX Spectrum uden et særligt tilpasningsled, købt hos din autoriserede Spectrum-forhandler specielt til dette formål.

Du har et års garanti fra købsdatoen, hvis datamaten er købt hos en autoriseret forhandler, men garantien bortfalder, dersom du tilslutter uautoriseret tilbehør, der forvolder skade på den, hvis du øver vold mod din datamat f.eks. ved at sætte den strømførende ledning ind i antennestikket, eller hvis du åbner din datamat.

Sæt aldrig din ZX-printer i datastikket, mens der er strøm på datamaten, og træk aldrig printerstikket ud, før du har taget strømmen fra datamaten!
Normal brug af din datamat kan ikke skade den! Normal brug er alt, hvad du kan skrive ind i den ved hjælp af tastaturet eller indspille i den med normal styrke fra en almindelig kassettebåndoptager!

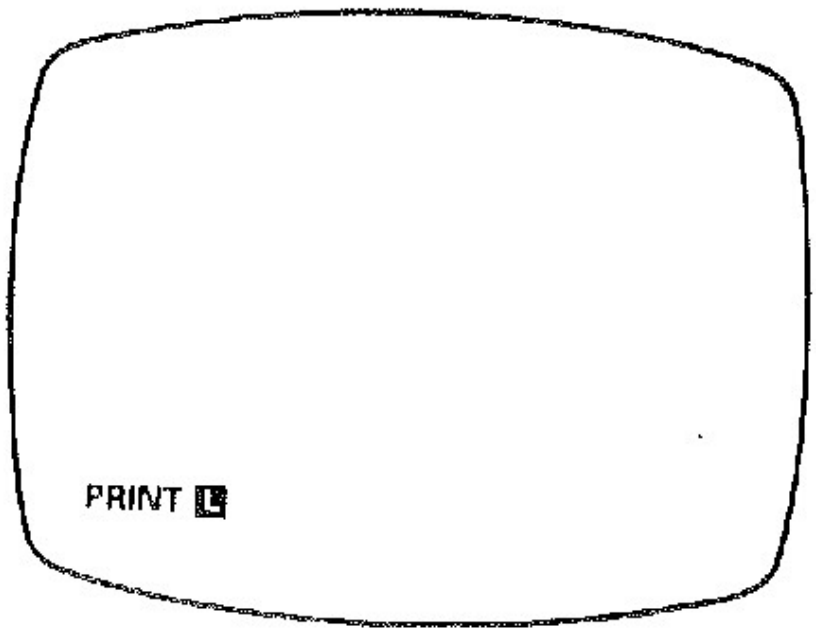
Hvis du skulle få brug for service, så kontakt ZX-datas serviceafdeling. Se side 3. Datamater, købt hos en autoriseret forhandler, bærer ZX-datas garantimærke på undersiden.

2. Tastaturet

Man lærer bedst noget ved at prøve sig frem. I det følgende beskriver vi tastaturets forskellige funktioner. Hvis du er nybegynder eller helt ukendt med Sinclair-datamater, er det en god idé at prøve de forskellige ting, vi beskriver, efterhånden som vi når frem til dem.

Spectrums tastatur ligner meget det, der er på en almindelig skrivemaskine. Tal og bogstaver er placeret på nøjagtigt samme steder. Der er dog én vigtig forskel mellem Spectrum og skrivemaskinen: Spectrums taster har mere end én funktion. Som på en skrivemaskine kan du ved hjælp af en skiftelast få små eller store bogstaver frem efter behov.

Men skiftetasterne på Spectrum kan meget mere. De kan indstille datamaten på forskellige brugstilstande. På tv-skærmen kan du hele tiden se, hvilken tilstand datamaten i øjeblikket er indstillet på. Det afsløres af den blinkende markør nederst til venstre på skærmen. Hvis den er et **K**, vil du ved et tryk på en tast få et BASIC-ord, der står på den. Trykker du f.eks. på **P**-tasten, skriver datamaten **PRINT**. (Den copyright-melding, du får, når du tænder datamaten, virker også som en K-markør).



K'et i markøren står for Keyword eller Nøgleord. Spectrums taster indeholder disse nøgleord. Dette system er stærkt pladsbesparende i Spectrums arbejdslager, for hvert af dem fylder kun én byte i "hukommelsen", som man også med et lidt forkert udtryk kalder dette lager. Nøgleiast systemet er som regel også meget arbejdsbesparende, vil du efterhånden erfare.

Hvis du ikke allerede har skrevet **PRINT** på skærmen, så gør det nu ved at trykke på **P**- tasten. Hvis der i forvejen står en anden tekst end copyright-meldingen på skærmen, så træk datamatens stik ud og start forfra.

På **P**-tasten står også "-tegnet. Det kan du få frem ved at holde på skiftetasten med påskriften **SYMBOL SHIFT** nederst til højre, mens du samtidig trykker på **P** tasten.

Prøv det. Så vil du se, at markøren skifter til et **L**. Det står for Letter eller Bogstav, d.v.s. at datamaten nu er skiftet til en anden tilstand og forventer, at du indskriver bogstaver eller tal.

Prøv at indtaste dit fornavn. Det første store bogstav får du frem ved at bruge skiftetasten **CAPS SHIFT** helt ud til venstre. Denne tast bruges generelt, hvis du skal anvende noget, der er skrevet med hvidt over tasterne. Hvis du skal have fat i noget, der står med rødt på tasterne, bruger du **SYMBOL SHIFT**-tasten.

Hvis du har fulgt anvisningerne, ser du nu på skærmen en tekst, der består af **PRINT**, efterfulgt af et citationstegn " og dit fornavn.

Når datamaten får en melding bestående af en **PRINT**-ordre, fulgt af et eller flere tegn skrevet mellem citationstegn, så skriver den teksten mellem citationstegnene på skærmen. Prøv derfor nu at skrive endnu et "-tegn efter dit navn.

Endnu er den tekst, du har skrevet, ikke behandlet af datamaten. Det bliver den, når du trykker på tasten **ENTER**. Du skal opfatte denne tast som en kontakt, der igangsætter datamatens behandling af dine data.

Når du trykker på **ENTER**, vil dit fornavn stå på skærmen sammen med nogle andre tegn nederst til venstre. Et blinkende spørgsmålstegn viser f.eks., at du har begået en eller anden fejl og må finde den og rette den, inden du kan gå videre.

Den kode, du fik i bunden af skærmen, hvis alt gik, som det skulle, er datamatens melding om, at alt fungerede godt. Denne kode er vigtig, når du kører programmer, men i øjeblikket kan vi glemme den.

Læg mærke til noget andet: Bogstavet O og tallet nul skrives med to forskellige tegn. Nullet er et Ø og kan udmærket anvendes til dette bogstav, hvis du har brug for et Ø i et ord, men må *aldrig* anvendes på steder, hvor datamaten kan forveksle det med et tal.

Datamaten vil altid betragte bogstavet O som et bogstav, så husk endelig, hvis du er vant til at skrive på skrivemaskine, at du ikke må bruge det som et nul, sådan som du måske plejer. På samme måde må du endelig ikke bruge lille L i stedet for et 1-tal.

Datamater er – som du allerede kan se – rent dumme. De gør kun det, de får besked på, og skal have 100 pct. nøjagtig besked på det, ellers forstår de dig slet ikke.

Det er menneskeligt at fejle – men ikke, hvis man vil have et dataprogram til at køre...

Da tastaturets forskellige tilstande er så vigtige, lager vi lige en opsummering:

Det blinkende hvide **L** kaldes markøren. Den viser, hvor på skærmen datamaten vil sætte det næste, du skriver. Det er ikke altid et **L**. Hvis du slukker for datamaten, tænder igen og trykker **ENTER**, når copyright-meldingen kommer, så får du en **K**-markør.

Det bogstav, der står i markøren, viser dig, hvordan datamaten vil udlægge det, du indtaster.

Ved begyndelsen af en programlinje får du en **K**-markør. K'et står for Nøgleord, og det er de ord, der kommer i starten af en ordre til datamaten. De skal fortælle datamaten, hvilken ordre du vil give den. Når datamaten har fået et nøgleord, forventer den

ikke flere. Derfor skifter markøren til [L], der betyder bogstav eller tal.

Vi taler om datamatens tilstande, og vil i det følgende tale om K-tilstand og L-tilstand.

Hvis du ønsker at skrive en masse med store bogstaver, behøver du ikke at holde tasten **CAPS SHIFT** nede. Du kan låse den ved at holde **CAPS SHIFT** nede, mens du trykker på 2-tasten. Så får du det, der står med hvidt over 2-tasten **CAPS LOCK**, hvilket betyder 'låste' store bogstaver.

For at vise dig, at du har låst tasten, ændres markøren til et blinkende [C]. Hvis du vil tilbage til [L]-markøren og altså små bogstaver, så trykker du påny på **CAPS LOCK**-tasten.

(Hvis du trykker på **CAPS LOCK**, mens datamaten er i K-tilstanden, mærker du ikke umiddelbart nogen forskel, mens så snart du har skrevet et nøgleord, får du en [C]- markør i stedet for [L]-markøren).

Udover nøgleord, bogstaver, tal og forskellige programmeringsord og videnskabelige udtryk rummer tastaturet otte grafiske tegn. De ligger på tasterne 1 til 8 og kan skrives på skærmen på samme måde som bogstaver og tal. For at få fat i dem må du skifte til grafiktilstand. Det gør du ved at trykke på **CAPS SHIFT**-tasten og på 9-tasten. Nu står der en [G]-markør på skærmen. Hvis du vil tilbage til L-tilstanden, må du trykke 9 påny.

Så er der kun én tilstand tilbage, du kan få frem. Det hedder E-tilstanden, og E står her for extended eller udvidet. Du får fat i [E] markøren ved at trykke tasterne **CAPS SHIFT** og **SYMBOL SHIFT** *samtidig*. Nu kan du anvende programmeringsordene og de videnskabelige udtryk, der står over eller under tasterne.

E-tilstanden varer kun ét tasttryk. Hvis du nu med [E]-markøren på skærmen trykker på en af skiftetasterne, så får du de ord, der er skrevet med rødt under tasterne. Hvis du ikke trykker på nogen skiftetast, får du de ord, der er skrevet med grønt over tasterne, og hvis du trykker på et tal uden at trykke på **SHIFT**, får du en farve. Hvis du vil have et af de ord, der står med rødt under taltasterne, må du samtidig holde på **SYMBOL SHIFT**.

Hvis du står i E-tilstanden og gerne vil tilbage til L-tilstanden, så må du igen presse de to skiftetaster samtidigt.

Selv ikke den dygtigste og mest trænede maskinskriver eller programmør kan undgå af og til at trykke forkert. Hidtil har vores eneste udvej været at trække stikket ud. Det er jo udmærket, hvis der kun står én ordre i datamaten, men ikke særligt praktisk, hvis den rummer et helt program, man møjsommeligt har indtastet.

Heldigvis rummer vort tastatur en løsning. Det er ordren **DELETE** på 0-tasten helt oppe til højre.

Lad os fx antage, at du vil skrive

PRINT "Hej"

men at du glemte at bruge **SYMBOL SHIFT**-tasten ved det første citationstegn. Nu står der i stedet

PRINT PHej"

Vor datamat forstår ikke længer ordren, når du trykker **ENTER,** for der er jo ikke det ventede citationstegn. Den viser sin forvirring ved at blinke med et spørgsmålstegn for enden af linjen.

Heldigvis behøver du ikke at indtaste det hele igen. På de øverste taster ser du fire piletaster over 5, 6, 7 og 8. Dem kan du bruge til at flytte markøren med, hvis du holder på **CAPS SHIFT,** mens du trykker på dem. Det er kun venstre- og højrepilen, der virker i dette tilfælde. Nu kan du flytte markøren hen til fejlen, og hvis du trykker **CAPS SHIFT** og **0**, fjerner datamaten det tegn, der står umiddelbart før markøren. Nu kan du bare indtaste et nyt korrekt tegn i stedet. Prøv det et par gange for at indøve denne rutine!

Hvis du holder på piletasterne eller **DELETE** i et eller to sekunder, vil du se, at de ikke blot flytter sig ét skridt, men bare cykler derudaf. Samtidig hører du en klikkende lyd. Det samme sker faktisk med enhver tast, der bliver holdt nedtrykket i ca. tre sekunder, fordi din datamat er forsynet med automatisk gentagelse.

Husk, at hvis du har skrevet noget forkert, eller du har ændret mening, så kan du ikke blot som ved en skrivemaskine skrive noget nyt oveni det, du har skrevet. Du må slette ved hjælp af **DELETE**-tasten og skrive det korrekte i stedet.

3. Tal, bogstaver og datamaten som regnemaskine

Vi har allerede set, hvordan man får datamaten til at skrive bogstaver og grafik på skærmen ved hjælp af ordren **PRINT**. Vi har også set, hvordan **ENTER** anvendes for at få datamaten til at udføre en ordre. Herefter skriver vi ikke længer **ENTER** i denne instruktionsbog, hver gang vi har brugt en ordre. Vi går ud fra, at du selv automatisk trykker på den tast, når du har fuldført en linje.

Datamaten har lettere ved at behandle tal end bogstaver.

Hvis vi derfor indtaster

PRINT 2

så dukker tallet 2 op på skærmen.

Man kan også blande tal og bostaver:

PRINT 2, "ABC"

I læg mærke til, at kommaet i denne PRINT-sætning gav et mellemrum mellem 2 og ABC.

Prøv så at skrive

PRINT 2; "ABC"

og derefter

PRINT 2 "ABC"

Som du ser, giver et komma mellem to ting en *tabulering* af en tekst eller nogle tegn hen over 16 kolonner, mens et semikolon ikke giver mellemrum, og hvis der slet ikke er noget tegn mellem to ting, får man en fejlmelding.
PRINT kan også bruges med tastaturets matematiske funktioner. Man kan faktisk bruge en Spectrum som en regnemaskine.

Hvis du fx skriver:

PRINT 2+2

så får du svaret i skærmens øverste del. Prøv at sammenligne dette med:

PRINT "2+2"

Man kan også kombinere det til noget mere nyttigt:

PRINT "2+2=";2+2

Du kan også forsøge dig med andre former for regnestykker:

PRINT 3–2
PRINT 4/5
PRINT 12*2

Hvis du prøvede de tre ting, så opdagede du, at skråstregen i regnestykke 2, som du fandt på **V**-tasten, faktisk er et divisionstegn, mens stjernen, der står på **B**-tasten, er et gangetegn. De to tegn bruges i stedet for x og : (eller ÷) for at undgå at forvirre datamaten. Den skal jo, som vi tidligere antydede, have alting ind med skeer.

Du kan også eksperimentere med forskellige udregninger. Hvis du har lyst, kan du også prøve negative tal eller tal med decimaler. Det negative fortegn står på **L**-tasten, og til decimaler skal du bruge punktum på **M**-tasten – endelig ikke komma, for det får jo, som vi så tidligere, datamaten til at tabulere.

Som du nu har set, bruges de nederste linjer på skærmen til indskrivning, og det indskrevne dukker ved brug af tasten **ENTER** op på skærmens øverste del, hvor der er plads til 22 linjer. Hvis du skriver tilstrækkelig meget til at udnytte alle 22 linjer, vil du opdage, at teksten automatisk ruller en linje opad, så den øverste linje tilsyneladende går tabt. Det kaldes SCROLL. Denne særlige finesse skal vi senere vende tilbage til.

Kalkulationer udføres ikke altid i den rækkefølge, du ville forvente. Prøv fx at skrive:

PRINT 2+3*5

Det var vist et lidt overraskende resultat for den, der ikke tidligere har haft fingre i en datamat!

Du troede måske, at datamaten ville sige: '2 og 3 er 5, og 5 gange 5 er 25'. Som du ser, er det ikke tilfældet. Datamaten har i sit indbyggede regelsæt en *prioritering* af regnefunktionerne. Først udfører den multiplikationer og divisioner, derefter additioner og subtraktioner. Derfor betyder udtrykket '2+3*5' faktisk: 'tag 3 og gang det med 5, som giver 15, og læg derefter 2 til, så resultatet bliver 17'.

Da multiplikationer og divisioner har samme prioritet, udfører datamaten dem i den orden, den finder dem i, når den går frem fra venstre mod højre. Det samme gælder additioner og subtraktioner. Hvis du skal bruge matematik på din Spectrum, må du altså opstille dine regnestykker udfra denne forudsætning, ellers får du forkerte resultater.

Lad os nu prøve at se, hvordan datamaten behandler:

PRINT 20–2*9+4/2*3

I 20–2*9+4/2*3
II 20–18+4/2*3
III 20–18+2*3

} først udføres multiplikationer og divisioner i rækkefølge fra venstre mod højre

IV 20—18+6
V 2+6
VI 8

} og derefter additioner og subtraktioner

Du behøver blot at huske, hvilke operationer der har lavere eller højere prioritet, men for at tilfredsstille din evt. nysgerrighed skal vi oplyse, at datamaten ordner prioriteringen ved at lade de forskellige operationers prioritet udtrykke med et tal, der ligger mellem 1 og 16. * og / har prioritet 8, mens + og – har prioritet 6.
Denne rangfølge overholder datamaten strengt. Den skal jo have det hele bøjet i neon for at forstå det. Men vi mennesker har jo lov til at være mere snedige end datamater. Du kan faktisk snyde datamaten ved brug af parentester. Alt det, der står i parentes udregnes først og behandles derefter som et enkelt tal. – Derfor vil

PRINT 3*2+2

give svaret 8, mens

PRINT 3*(2+2)

vil give svaret 12 – nemlig resultatet af 3*4.

Det er undertiden en god måde at servere et udtryk for datamaten, idet du kan give den et udtryk, hver gang den forventer at få et tal af dig. Så vil den udregne svaret. Undtagelserne fra denne regel er så få, at vi vil kunne nævne dem specielt, hver gang de dukker op.

Du kan skrive tal med decimaler, idet du bruger punktum til skilletegn, men du kan også bruge eksponentiel notation – noget, der er rel almindelig på lommeregnere. Her skriver du efter et almindeligt tal – med eller uden decimaler – en eksponentdel, der består af bogstavet **e**, derefter måske **–** og derefter igen et tal. Eksponentdelen flytter decimalkommaet til højre ved positive tal og til venstre ved negativ eksponent. Herved bliver altså det oprindelige tal multipliceret (eller divideret) et antal gange **10**. Fx:

2.34e0=2.34
2.34e3=2340
2.34e–3=0.0234 o.s.v.

Prøv engang at skrive disse tal på datamaten.

Her er et af de få tilfælde, hvor du ikke kan erstatte et tal med et udtryk. Fx kan du ikke skrive

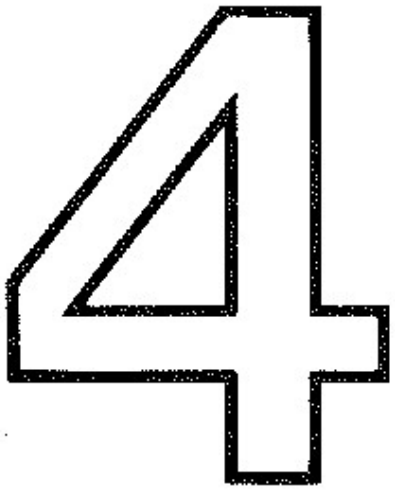
(1.34+1)e(6/2).

Man kan også bruge udtryk, hvis værdi ikke er tal, men rækker af bogstaver. Du har allerede set dette i den enkleste form: En række bogstaver omgivet af citationstegn.

Dette kaldes en *streng*. Det svarer stort set til den enkleste form for numerisk udtryk, der blot er et tal, indskrevet uden nogen form for kalkulationer. Endnu har du ikke set brugen af + med strenge (men ikke med −, * eller /, så derfor opstår ingen prioritets-problemer her). Hvis man adderer strenge, opnår man blot, at de lænkes sammen den ene efter den anden. Prøv derfor

PRINT "jers"+"eyko"

Du kan sammenføje så mange strenge, du lyster, i et enkelt udtryk, og hvis du har lyst, kan du endog bruge parenteser.



4. Nogle enkle ordrer

Datamatens 'hukommelse' kan bruges til at gemme alt muligt. Vi har hidtil set, hvordan **PRINT**-ordren tillader os at vise bogstaver, tal og resultater – både med tal og bogstaver – på skærmen.

Hvis vi vil have datamaten til at huske et tal eller en række bogstaver, så må vi afsætte plads i dens arbejdslager til det.

De fleste lommeregnere har en tast, der hedder 'memory', som man bruger, når man vil have apparatet til at huske et tal til senere brug. Din datamat kan gøre det endnu bedre: Den kan have en hel række tænkte hylder at gemme ting på, og du kan navngive hver eneste af dem.

Lad os fx forestille os, at du gerne vil huske en persons alder. Så bruger du **LET**-ordren, der står på L-tasten. Lad os sige, at det er 34 år, du skal huske. Så skriver du

LET aar=34

Det, der sker ved **LET**-ordren, er, at du hæfter etiketten 'aar' på et område (en hylde) i arbejdslageret og derefter gemmer tallet 34 i det. Hvis du nu vil have den opbevarede information ud igen, så skriver du

PRINT aar

og vupti kommer svaret 34. Det er let nok at ændre det, hylden 'aar' rummer. Skriv blot

LET aar=56

og derefter

PRINT aar

svaret på skærmen bliver selvfølgelig 56. Aar er et eksempel på det, der kaldes *en variabel.* Navnet kommer af, at værdien i en variabel kan variere, altså veksle. Det er muligt at sammenknytte en meddelelse på skærmen med værdien af en variabel. Skriv

PRINT "du er ";aar

og læg mærke til mellemrummet efter 'er'. Hvis det ikke er der, så vil 'er' blive klinet op ad tallet.

Der er dog meget nyttigere ting, datamaten kan gøre, end blot at huske tal med tilknyttede navne. Den kan også huske bogstavssstrenge. For at datamaten kan skelne mellem talvariabler og strengvariabler – som de kaldes – bruger man et dollartegn **$** i enden af variabelnavnet. Hvis vi fx ville gemme bogstavsrækken

"du er"

så kunne vi kalde den

a$

(strengvariablers navne kan kun bestå af ét bogstav, udover **$**). Prøv derfor nu at skrive

LET a$="du er "

Hvis du nu indlaster

PRINT a$

får du som svar hele denne række af bogstaver: Strengvariablen **a$.** Hvis du nu ikke har haft slukket for din datamat siden begyndelsen af dette kapitel, kan du nu taste

PRINT a$;aar

og se, hvad der sker.

Der er andre måder, hvorpå man kan få lagt oplysninger ind i datamatens lager uden at bruge **LET.**

Hvis man fx bruger **INPUT**-ordren i dens enkleste form, fortæller man datamaten, at den må forvente, at der kommer oplysninger fra tastaturet. I stedet for at taste **LET** o.s.v. hver gang kan du i stedet skrive

INPUT aar

Så snart **ENTER**-tasten er blevet rørt, vil en blinkende ■-markør dukke op på skærmen. Det betyder, at datamaten ønsker at få nogle oplysninger af dig. Skriv din alder og tryk **ENTER.** Skønt intet synes at ske, har du faktisk tildelt variablen den værdi, du indtastede. Det kan du bevise ved at taste

PRINT aar

Lad os nu prøve at sammenføje all dette i en hel serie ordrer. Skriv

```
LET b$="Hvor gammel er du?"
LET a$="Du er "
INPUT (b$);aar:PRINT a$;aar
```

Bemærk, at sidste linje består af to ordrer, adskilt ved kolon.

INPUT (b$);aar

er en anden måde at skrive

INPUT "Hvor gammel er du?";aar

5. Simpel programmering

Hidtil har vi givet datamaten ordrer direkte gennem tatastaturet, men skønt det er muligt at sammenknytte kommandoer, er det kun muligt at anvende datamaten i begrænset omfang, hvis man anvender denne metode.

Det strålende ved datamater er, at de er programmerbare. Det betyder, at man kan give dem en serie instrukser, der får dem til at udføre ting og sager i en bestemt rækkefølge. En sådan sammenstilling af instrukser kaldes *et program*.

Enhver datamat har sit eget sprog, der tillader os at kommunikere med den, når vi skal programmere datamaten. Nogle sprog er meget enkle, således at datamaten let kan forstå dem. Uheldigvis er sprog, der er lette for en datamat, temmelig svære for mennesker. På sin vis gælder det modsatte også – altså , at sprog, der er lette for os, er relativt svære for en datamat. De må oversættes og fortolkes.

ZX Spectrum anvender et højniveausprog, der hedder **BASIC.** Navnet skrives med lutter store bogstaver, fordi det er en forkortelse for ordene **B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode, hvilket i oversættelse betyder noget i retning af en "symbolsk instruktionskode, som begyndere kan anvende til ethvert formål". Dette datasprog blev udviklet i 1964 på Dartmouth College i den amerikanske delstat New Hampshire.

Sproget var, som navnet siger, oprindeligt tiltænkt begyndere, men har i tidens løb udviklet sig så meget og fået så vid udbredelse, at det i dag er verdens mest anvendte datasprog.

Men skønt sproget er så vidt udbredt, især blandt mikrodatamater, betyder det ikke, at BASIC-datamaterne umiddelbart kan kommunikere med hinanden. Der er væsentlige forskelle fra datamat til datamat. Derfor er denne instruktionsbog skrevet specielt til ZX Spectrum.

Spectrums BASIC ligger dog ikke langt fra de mest udbredte BASIC-former, så du vil nok ikke få store vanskeligheder med at tilpasse andre datamaters BASIC-programmer, så de kan køre på din Spectrum. her er én vigtig forskel. Modsat de fleste andre BASIC-datamater tillader Spectrums BASIC *ikke*, at du udelader **LET**-ordren, når du skal tildele en variabel en værdi.

Der er grænser for, hvor mange ordrer du kan opbevare i en datamat. Spectrum melder om denne grænse ved summen.

Der er grænser for, hvor mange ordre du kan opbevare i en datamat. Spectrum melder om denne grænse ved en summen.

Når man programmerer i BASIC, er det nødvendigt at fortælle datamaten den rækkefølge, hvori instrukserne skal udføres. Derfor må hver linje begynde med et nummer. Det er normalt at begynde med 10, fortsætte med 20, 30 o.s.v. Når man på denne måde laver "mellemrum" mellem linjerne, kan man altid indsætte linjer, man har glemt, eller ændre på programmet. F.eks. indsætte en linje med nummeret 21 efter 20, 5 før 10 o.s.v.

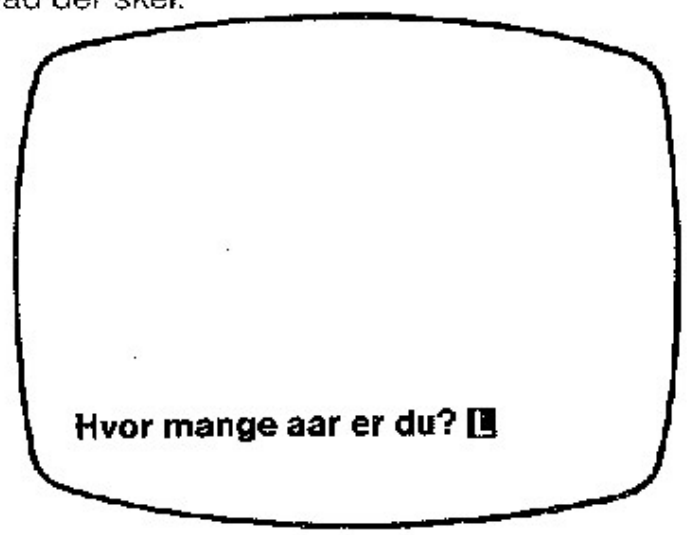
Lad os prøve at se på et ret enkelt program. Se på den serie ordrer, vi brugte i slutningen af sidste kapitel. Hvis vi ønskede at gentage denne serie, ville det være nødvendigt for os at indtaste dem igen hver gang. Formålet med et *program* er at spare os for denne møje.

Prøv at indtaste det følgende og husk at skrive **ENTER** efter hver linje:

```
10 LET b$="Hvor mange aar er du?"
20 LET a$="Du er ";" aar"
30 INPUT (b$) ;aar
40 PRINT a$;aar
```

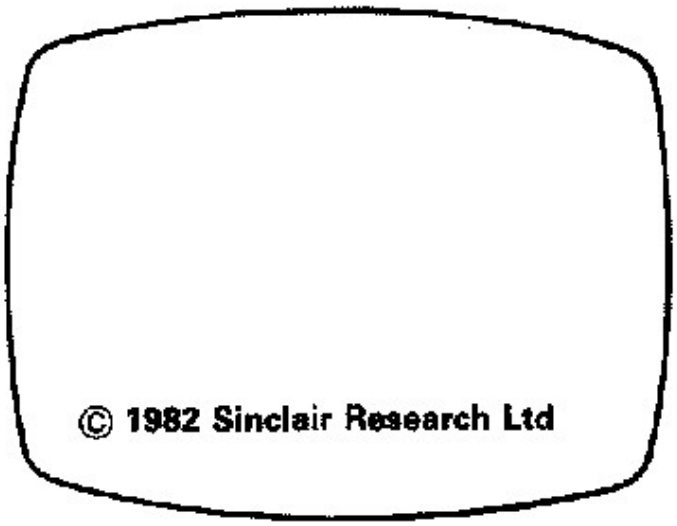
Læg mærke til, at det ikke er nødvendigt at indskrive mellemrum, undtagen på passende steder mellem citationstegnene.

Nu har du skrevet et program, men datamaten kan ikke gøre det mindste ved det, før du giver den en arbejdsordre. Det gør du ved at bruge ordren **RUN** (kør det), der ligger som nøgleord på **R**-tasten.
Prøv det og se, hvad der sker.



Bemærkede du, at der var en højrepil udfor hver linje, du indtastede. Denne pil peger altid på den sidst indskrevne linje. Hvis du gerne vil se programmet igen, kan du kalde det frem fra datamatens lager ved at trykke **ENTER** eller **LIST.** Når du gør det, får du det, man kalder *programlistningen.* Du kan også trykke **RUN** og køre programmet lige så mange gange, du måtte lyste.

Når du er blevet træt af det, kan du fjerne det ved at bruge ordren **NEW** (**A**-tasten). Denne ordre sletter et program fra hukommelsen, og lige meget hvor mange gange du trykker **LIST,** får du det ikke tilbage igen. Med **NEW** har du fået tavlen vasket ren og er klar til et nyt program. Prøv engang med **NEW** og **LIST** og se, hvad der sker!



For lige at repetere:

Når du indtaster et nummer og en ordre, forstår datamaten, at det ikke blot er en ordre, men en programlinje. Datamaten udfører den ikke, men gemmer den til senere brug.

ZX Spectrum viser dig hjælpsomt på skærmen alle de programlinjer, du har indtastet – den *lister* dem, og der er et pilemærke udfor den sidst indtastede.

Datamaten vil ikke øjeblikkeligt udføre programlinjerne (således som tilfældet ville være med mange af ordrerne, hvis de ikke havde linjenummer). Den gemmer dem i sit arbejdslager.

For at få datamaten til at udføre linjerne må du bruge kommandoen **RUN.**

Hvis du trykker på **ENTER** og intet andet, får du programlistningen igen.

Lad os nu se på endnu et enkelt program. Dette bliver en smule mere matematisk. Det viser kvadratet på alle tal mellem 1 og 10. (For ikke-matematikeren: Kvadratet af et tal er bare tallet ganget med sig selv).

Her får vi en ny side af programmering i BASIC, nemlig den metode, datamaten anvender for at tælle. Vi så tidligere i bogen, at datamaten kan gemme tal i sit arbejdslager, hvis man hænger en etikette på det – eller med andre ord: Hvis man tildeler en variabel en værdi. Lad os tage variablen **x** og tildele den startværdien 1 og derefter øge dens værdi i trin fra 1 til 10. Det gør vi meget let med ordren **FOR...TO...STEP.**

Vi starter med at trykke **NEW** for at rense datamaten for tidligere data og indtaster følgende:

```
10 FOR x=1 TO 10 STEP 1
```

(Normalt kan man udelade **STEP 1,** hvis tællingen går et trin op ad gangen).

Næste linje må nu fortælle datamaten, hvad den skal gøre med **x**, uanset dens værdi. Derfor skriver du:

```
20 PRINT x,x*x
```

Til slut skal vi bruge en linje, der fortæller datamaten, at den skal gå videre til **x**'s næste værdi. Derfor skriver du:

```
30 NEXT x
```

Når datamaten når frem til denne instruks, iler den tilbage til linje 10 og gentager det hele. Når værdien af **x** overstiger 10, skal datamaten vide, at programmet er færdigt. Derfor indtaster vi:

```
40 STOP
```

Når du nu trykker **RUN,** vil skærmen vise to kolonner. I den første har du værdien af **x**, i den anden har du kvadratet på **x** (altså **x** gange **x**). Det er også muligt at navngive disse kolonner. Tryk **LIST** eller **ENTER** og tilføj linjen:

```
5 PRINT "x","x*x"
```

Læg mærke til, at skønt linje 5 blev indføjet efter alle de øvrige linjer, så sørgede datamaten for, at den kom ind på rette plads - øverste linje, fordi den havde det laveste linjenummer.

Du behøver ikke at indskrive linjer i den rigtige rækkefølge. Datamaten skal nok sortere dem efter nummer. Men linjerne skal nummereres i den rækkefølge, du ønsker dem udført!

Prøv at skrive programmer, der bruger andre matematiske funktioner. Hvis du er i tvivl om deres rette brug, så find dem i stikordsregistret bagi bogen. Det henviser til de sider, hvor de står omtalt.

6

6. Brugen af kassettebåndoptageren

Det er ret trættende at skulle indtaste programmer i din datamat, hver gang du bruger den. Heldigvis kan du overspille dine programmer fra ZX Spectrum til en almindelig kassettebåndoptager og senere spille dem tilbage i datamaten, når du har brug for dem. Hvis du på dette tidspunkt har et program i datamatens arbejdslager (f.eks. kvadratprogrammet i forrige kapitel), kan du prøve at lagre det i din båndoptager ved den fremgangsmåde, vi beskriver om lidt.

De fleste kassettebåndoptagere vil samarbejde med Spectrum. Datamaten er ikke kræsen. Den foretrækker ofte en billig, bærbar monooptager fremfor et dyrt stereoanlæg, ikke mindst fordi den billige hyppigere har de rigtige stik.

Kassettebåndoptageren skal have de rigtige stik (input) til mikrofon og et stik (output) til hovedtelefoner (earphone). Hvis sidstnævnte ikke findes, så forsøg dig med stikket til ekstrahøjttaler. Stikkene skal være 3,5 mm jackstik – andre stik giver meget ofte ikke tilstrækkelig kraftigt signal.

I teorien duer ethvert kassettebånd, men i praksis bør du vælge bånd af en god kvalitet og ikke tilbudsbånd af typen "ti for en tier".

Lad os nu forudsætte, at du står med en passende båndoptager. Nu forbinder du den til Spectrum ved hjælp af de medfølgende dobbeltledninger. Sæt den ene farve i MIC på Spectrum og det tilsvarende stik på båndoptageren, og den anden farve i EAR på din Spectrum og EAR på båndoptageren.

Der sker ikke mindste skade på din Spectrum, hvis du forbinder dem forkert.

Nu skal vi til at gemme et program ved ordren *SAVE*. Men først skal du trække det ene EAR-stik ud (vælg selv hvilket), ellers optager du kun et brum på båndoptageren, fordi lyden løber lige fra det ene mikrostik til det andet og tilbage gennem EAR-stikkene. Så opstår der svingninger, der udsletter det signal, du forsøger at optage. **HUSK ALTID AT UNDGÅ DETTE VED OPTAGELSE!**

Hvis du nu har kvadratprogrammet fra sidste kapitel inde i datamaten, trykker du bare:

SAVE "KVADRAT"

Kvadrat er navnet på dit program. Ethvert program, der overspilles til båndoptageren, skal have et navn, ellers kan du ikke finde det igen, og iøvrigt vil datamaten ejheller acceptere overspilningen. Navnet, der blot er en etikette, du hæfter på programmet, kan bestå af indtil ti tegn: Kun tal eller bogstaver accepteres.

Når du har skrevet **SAVE "Kvadrat"**, svarer datamaten med tilbagemeldingen **Start tape then press any key** eller oversat: Start båndoptageren og tryk på en hvilken som helst tast.

Nu laver vi først en prøvekørsel uden at starte båndoptageren. Tryk på en tast og se på tv-skærmens kant. Her løber et mønster af farvede vandrette striber. Først kommer fem sekunders rød og lyseblå. Striberne er ca. en cm brede og bevæger sig langsomt opad. Så kommer et kort udbrud af blå og gule striber. Et sekund er alt igen normalt. To sekunder igen af rød og lyseblå striber, og igen et sekund af det blå-gule mønster.

Prøv det – eventuelt nogle gange, indtil du kender mønstret. Dine data gemmes i to

blokke, og begge blokke har en indledning, der svarer til det rød-lyseblå mønster, og dine data, der svarer til det blå-gule mønster. Den første blok er en "forridder", der indeholder navnet og forskellige små oplysninger om programmet. Den anden er selve programmet med de evt. variabler. Det hvide afsnit ind imellem er blot et hul.

Lad os forsøge at fange signalet på båndoptageren.

1. Sørg for at båndet er kørt så langt frem, at du ikke "optager" på den farvede plaststrimmel.

2. Skriv

SAVE "Kvadrat" (og **ENTER**)

3. Start båndoptageren

4. Tryk en eller anden tast

5. Se på fjernsynsmønstret, og når det slutter med meldingen **0 OK**, stopper du din båndoptager.

For at sikre dig, at det hele gik som ventet, kan du nu tjække signalet på din båndoptager ved hjælp af **VERIFY**-ordren under **R**-tasten (E-tilstand og skift **R**). Når ordren **VERIFY** gives, sammenligner datamaten signalet på båndet med programmet for at se, om alt er kommet med på båndet.

1. Skru styrkeknappen på din båndoptager omtrent halvt op og sæt EAR-stikket i igen.

2. Spol båndet tilbage før optagelsen.

3. Tast

VERIFY "KVADRAT"

4. Afspil båndet

Kanten at fjernsynsbilledet vil veksle mellem rød og lyseblå, indtil båndet når din optagelse. Så ser du det samme mønster som før, da du optog programmet, men med én forskel: I det hvide hul i midten står der nu **Program Kvadrat**. Når datamaten leder efter noget på et bånd, skriver den altid navnene på de programmer, den passerer undervejs til det eftersøgte.

Hvis du ser hele dette mønster, og datamaten standser med meldingen **0 OK**, så er dit program gemt godt og sikkert på båndet, og du kan herefter springe det næstfølgende i rammen over, dersom du har lyst. Men hvis noget gik galt, så gennemlæs "rammefortællingen" for at se, hvad der skete.

Hvad der kan være gået galt ved SAVE

Viste navnet sig på skærmen?

Hvis ikke, så er programmet enten ikke ordentligt optaget, eller ikke ordentligt afspillet. Du må undersøge, om det var det ene eller det andet. Spol båndet tilbage til lige før optagelsens start, træk EAR-stikket ud af båndoptageren og afspil gennem båndoptagerens højttaler. Det rød-lyseblå mønster giver en klar høj tone, og de blå-gule data giver en mere ubehagelig lyd, der nærmest kan sammenlignes med morsesignaler i en brandstorm. Begge disse signaler er meget kraftige. Hvis du spiller med fuld lydstyrke, kan de let overdøve en samtale.

Hvis du nu ikke hører disse lyde, så blev programmet sandsynligvis ikke optaget. Tjæk nu, om du har sat de rigtige ledninger i de rigtige stik: Altså MIC til MIC, men ingen forbindelse fra EAR til EAR. Ved nogle båndoptagere er der ikke forbindelse, hvis jack-stikket er skubbet helt ind. Prøv at trække det ca. en millimeter ud. Somme tider kan du ligefrem mærke, at det så sidder i en mere naturlig stilling. Tjæk også, om du i virkeligheden optog på den indledende farvede plaststrimmel. Når du har tjækket disse ting, så prøv en ny optagelse.

Hvis du kan høre de lyde, vi beskriver, så gik din **SAVE** sikkert godt, og problemet ligger ved tilbagespilningen til datamaten.

Tjæk igen, om forbindelserne er rigtige. Tjæk også dit afspilningsniveau. Hvis du afspiller med for lav styrke,kan datamaten ikke høre signalet, og så ser du ikke det rigtige mønster på skærmen. Hvis du afspiller for højt, får du et forvrænget signal. Du kan måske endog høre signalet ud gennem datamatens egen højttaler. Der er en række acceptable styrkeniveauer mellem disse to yderligtgående. Prøv dig frem.

Datamaten finder programmet og skriver navnet, men ingenting sker. Her er nogle muligheder:

Du stavede navnet forkert enten ved **SAVE** (så skriver datamaten stavefejlen på skærmen), eller ved **VERIFY**. Så ignorerer datamaten programmet og fortsætter med sine røde og lyseblå striber.

Der er fejl ved optagelsen. Så vil datamaten skrive **R Tape loading error** (altså fejl ved afspilningen), og det betyder, at datamaten ikke kan gennemføre **VERIFY**-proceduren. Tilbage og lav en ny **SAVE**.

Det er også muligt, at styrkeindstillingen på båndoptageren ikke er helt rigtig, men helt forkert kan det ikke have været, når det lykkedes datamaten at læse den første blok.

Lad os nu gå ud fra, at du har **SAVE**d programmet og fået en god **VERIFY**. Genafspilningen til datamaten foregår nøjagtigt som **VERIFY**, bortset fra, at du denne gang skriver

LOAD "Kvadrat"

LOAD sidder på **j**-tasten. Eftersom dit program klarede **VERIFY**, skulle du ikke have problemer med **LOAD**ning.
LOAD virker som **NEW**, hvis du i forvejen har et program i datamaten. Det gamle program forsvinder, og du behøver altså ikke at slette det, før du **LOAD**er et nyt.

Når du har **LOAD**et et program, kan du "køre" det med ordren **RUN.**

Du kan købe forudindspillede programmer på bånd. Disse programmer skal være lavet specielt til Spectrum. Det kan således ikke nytte at prøve med kassetter fra ZX81 eller andre datamater.

Du behøver ikke at skrive **LOAD** og et programnavn i gåseøjne, hvis du kun har et program indspillet på et bånd, eller hvis du ved, at det ønskede program er det første, der vil komme, når afspilningen af båndet starter. Så kan du nøjes med at taste **LOAD** "" – og datamaten vil så snuppe det første program, den møder.

7

7. Farver

Én af grundene til, at du købte en ZX Spectrum, er sikkert, at den giver dig mulighed for at bruge farver på fjernsynsskærmen. Derfor vil vi ikke længer spænde dig på pinebænken i denne sag.
Skærmen er delt i to områder. Det ene er en ramme i kanten. Den kaldes **BORDER.** Området inden for rammen kaldes **PAPER.** Du kan frit skifte farverne i disse to områder, direkte fra tastaturet eller indirekte via et program.

ZX Spectrum har otte farver, der bærer numrene 0 til 7. Det er muligt, at du synes, de er valgt lidt tilfældigt, men just disse farver er udvalgt, fordi de giver hver sin grå-tone på et sort-hvidt fjernsyn

Her er listen over farverne og de taltaster, de sidder ved:

0 sort
1 blå
2 rød
3 purpur eller magentarød
4 grøn
5 lyseblå eller cyan
6 gul
7 hvid

Når man tænder for datamaten, arbejder den først i sort-hvid, og både **BORDER** og **PAPER** har værdien 7, d.v.s. hvid. Farven på ethvert tegn, der dukker op på skærmen, bestemmes af **INK**-ordren. Her er farven normalt ved start 0, d.v.s. sort. Det er i første omgang datamaten, der bestemmer de tre farveordrer, men du kan ændre deres værdi, f.eks. ved at indtaste

BORDER 2

Hvis du huskede at trykke på ENTER efter dette, vil rammen nu skifte fra hvid til rød. Det gælder også det rammeområde forneden, hvor dine indtastninger registreres. Hvis du nu forsøger at indtaste andre tal, kan du se, hvordan farverne ændres.

Prøv nu at ændre farven på midterfeltet ved at indtaste

PAPER 5

PAPER-ordren er en af ordrene i E-tilstand, som vi tidligere nævnte. Du får den frem ved *samtidigt* at trykke på tasterne **CAPS SHIFT** og **SYMBOLS SHIFT.** Derefter skal du trykke på **C** og en af skiftetasterne. Når du derefter trykker to gange **ENTER**, skulle midterfeltet skifte farve til lyseblåt. Det første **ENTER** annullerer den **PAPER**- ordre, der allerede ligger i datamaten, men først det andet **ENTER** skifter farven. (Dette andet **ENTER** får datamaten til at **LIST**e et program og derved genopbygge sine skærmoplysninger).

Hvis du anvender et farvetv, og det ikke skifter farver sådan, som det skulle, må du forsøge at ændre på farvekontrolknapperne på dit fjernsyn, eller måske skal du dreje på finindstillingsknappen.

INK-ordren ligner **PAPER**-ordren. Den bestemmer farven på de tegn, du skriver på den baggrundsfarve, du giver skærmen, idet **INK** (blæk) skriver på **PAPER** (papir). Hvis du giver **INK** og **PAPER** samme farve, skriver du med usynlig skrift, men det usynlige kan fremkaldes ved at skifte farve enten på **INK** eller **PAPER**.

Du kan naturligvis bruge de tre ordrer **BORDER**, **INK** og **PAPER** i programmer. Her er et enkelt program, der viser dig farveskalaen, du har til rådighed:

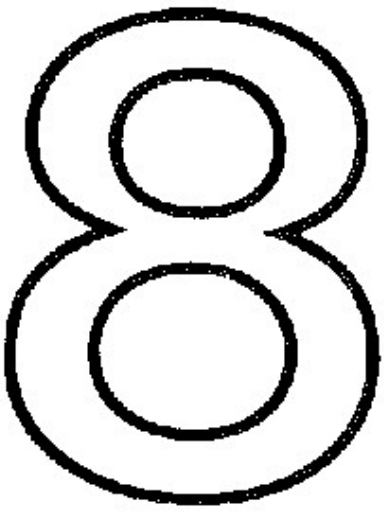
```
10 FOR x=0 TO 7
20 BORDER x
30 PAPER 7-x:CLS
40 PAUSE 50
50 NEXT x
```

Dette program gennemløber de otte farver og modstiller **PAPER** og **BORDER** farverne. Ordren **CLS** efter **PAPER** tvinger datamaten til at genopbygge skærmbilledet og anvende den nye **PAPER** farve. **PAUSE**-ordren standser programmet i et sekund, så vi kan se, hvad der sker (prøv engang at udelade **PAUSE**-linjen).

For at se, hvordan **INK**-ordren virker, kan du indtaste følgende program – efter en **NEW**-ordre.

```
10 BORDER 7
20 PAPER 1
30 INK 4
40 PRINT "grønne tegn på blå baggrund"
```

Der er andre ordrer knyttet til din Spectrums farvemuligheder. De omtales senere i instruktionsbogen.



8. Lyd

ZX Spectrum kan lave lyde i uendelige variationer. En tones frekvens og varighed bestemmes af brugeren. Ordren **BEEP** bruges til at fortælle datamaten, at den skal lave lyd. **BEEP** er en ordre i E-tilstanden og hentes på **Z**-tasten.
BEEP-ordrens "centrale" frekvens er det midterste C. Den kan varieres med **BEEP**-ordren, og man kan skabe enhver tone gennem at udtrykke den som halvtoner eller dele af halvtoner over eller under denne centrale frekvens. Hvis man skriver ordren

```
BEEP 2,0
```

vil datamaten frembringe en to sekunder lang C-tone.

De to tal styrer den tone, der udsendes. Det første tal angiver tonens længde i sekunder. Det andet viser tonens højde i halvtoner over det midterste C. Dette C udtrykkes altså med **0**, mens C# hedder 1, D er 2 o.s.v. opad til det næste C, der hedder 12, fordi der går 12 halvtoner på en oktav. Du kan fortsætte med 13 og videre, hvis du ønsker, således at jo højere tallet er, des højere er tonen på skalaen.

Prøv at skrive dette:

```
BEEP 1,4:BEEP 1,2:BEEP 2,0
```

det skulle være den første takt af den engelske melodi Three Blind Mice. Eftersom du kan sammenkæde rel mange **BEEP**er på denne måde med kolon'er imellem, skulle du med lidt tålmodighed kunne indtaste en hel melodi.

(Kolon'er bruges ikke blot til at sammenkæde **BEEP**er. Man kan bruge dem til at bygge sammensatte ordrer, og man kan benytte alle de elementære ordrer til dette).

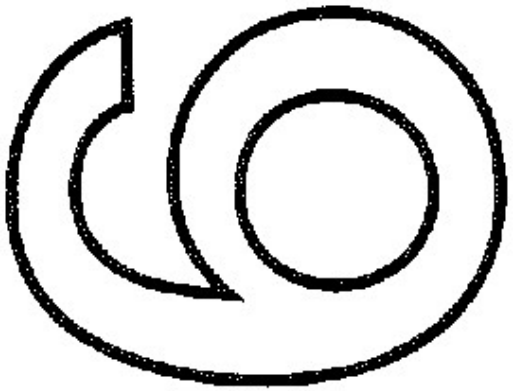
Du kan også bygge **BEEP**er sammen med andre ordrer, f.eks. til noget, man kunne kalde en-syngende-kamælon-ordre:

```
BORDER 1:BEEP 1,14:BORDER 3:BEEP 1,16:BORDER
4:BEEP 1,12:BORDER 6:BEEP 1,0:BORDER 5:BEEP
4,7:BORDER 1
```

Du behøver ikke blive forvirret over, at denne serie ordrer strækker sig fra den ene linje til den anden. Det bliver datamaten ikke.
Her har du så et kort program, der spiller en hel serie toner:

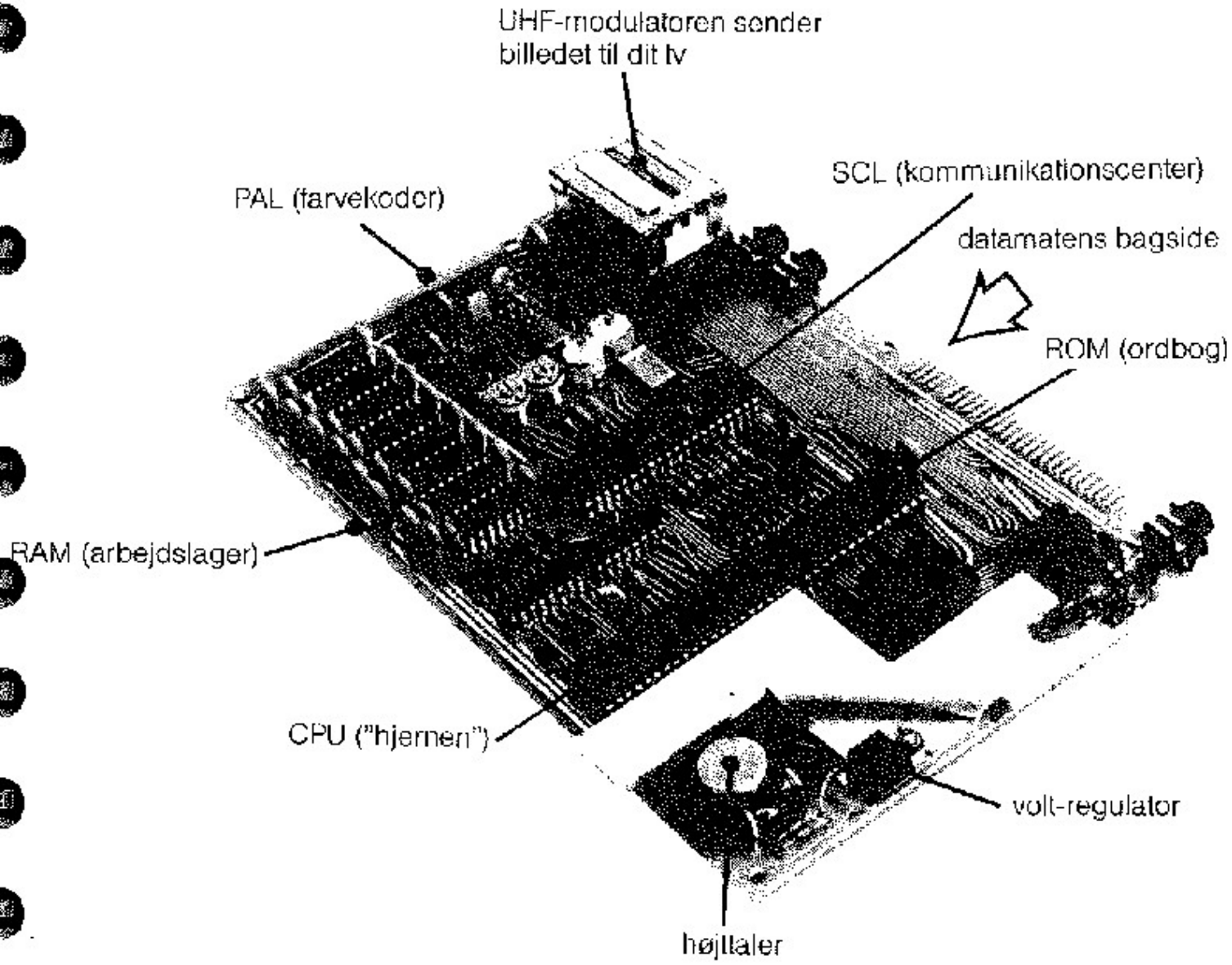
```
10 FOR x=0 TO 24
20 BEEP 2,x
30 NEXT x
```

Man kan udføre mange andre ting med denne ordre. Det kan du se senere i instruktionsbogen.
Hvis du vil spille toner, der er lavere end det midterste C, angiver du halvtonerne ved negative tal.



9. Hvad er der indeni?

Her er den nøgne sandhed om ZX Spectrum.



Som du kan se, bærer det meste inden i en Spectrum et navn på tre bogstaver. De sorte rektangulære plaststykker med en masse metalben er de integrerede kredse, der gør hele arbejdet i datamaten. Inden i hver af dem ligger et knapt en halv kvadratcentimeter stort stykke kisel, der er forsynet med tusinder af transistorer, som sammen med ledningsforbindelser udgør datamatens elektroniske kredsløb – mekanikken, der får det hele til at fungere.

"Hjernen" bag det altsammen er den integrerede kreds, der bærer navnet CPU. det er en mikroprocessor, der ikke blot udfører regneopgaverne, men også styrer datamaten, holder øje med hvilke taster, der er trykket på, afgør, hvad dette skal føre til, og iøvrigt beslutter, hvad datamaten skal gøre. i Spectrum er CPU'en en mikroprocessor, der kaldes Z80A.

Men uanset, hvor snild mikroprocessoren skulle være, så kan den ikke gøre arbejdet

alene. Eks. kender den ikke spor til datamatens sprog BASIC.
Reglerne for det arbejde, den skal gøre, må den hente i en anden kreds kaldet ROM. Navnet står for **R**ead **O**nly **M**emory, hvilket betyder et lager, der kun kan læses. En datamats ROM indeholder en lang liste instrukser for et dataprogram og for Spectrums BASIC. Disse instrukser er skrevet i et sprog, der kaldes Z80-maskinkode og består af en lang række tal. Der er ialt 16384 (16*1024) tal. De 1024 udgør det, man i datasproget kalder en kilobyte, og man siger derfor, at Spectrum har en 16 kilobyte eller 16K ROM. Denne ROM kan man populært kalde for datamatens ordbog.

Der er andre lignende kredse i andre datamater, men den rækkefølge, hvori instrukserne er skrevet, er specielt lavet til Spectrum.

De otte kredse øverst til venstre er datamatens RAM, hvilket står for **R**andom **A**ccess **M**emory eller arbejdslager. I RAMen opbevarer mikroprocessoren sine data, BASIC-programmer, variabler, fjernsynsbilledet og en masse andre ting. Mens en ROM er permanent kodet med sine egne oplysninger, lagres RAMens oplysninger kun midlertidigt. De forsvinder, hvis du slukker for strømmen eller gennem programmering tømmer lageret.

Den store kreds, du ser i midten, hedder SCL, og det står for **S**inclair **C**omputer **L**ogic. Denne kreds er kodet specielt til Sinclair Research, Spectrums producent. Den fungerer som datamatens kommunikationscenter – en slags telefoncentral, der sikrer, at alt går, som det skal. Den aflæser også lageret for at se, hvad der skal være på fjernsynsbilledet, og den sender så de rigtige signaler til UHF-modulatoren.

PAL-farvekoderen er en hel gruppe komponenter, hvis opgave det er at omdanne SCL-kredsens signaler til en form, der gør dem egnede til farvefjernsyn.

Regulatoren sørger for konstant at holde den lidt svingende strømstyrke på de nødvendige fem volt.

10

10. Over til det mere alvorlige

Hvad enten du har læst de første ni kapitler eller er kommet direkte til dette, skulle du nu være i besiddelse af en grundlæggende viden om, hvordan en datamat fungerer. Du ved således, at ordrer adlydes øjeblikkeligt, at instruktioner begynder med et linjenummer, og at disse linjer lagres til senere brug. Du skulle også kende kommandoerne **PRINT**, **LET** og **INPUT**, der bruges på alle datamater ved BASIC-programmering, og desuden de specielle Spectrum-ordrer **BORDER**, **PAPER** og **BEEP**.

Inden vi nu går over til det mere tekniske, skal vi repetere nogle ting, der er blevet beskrevet i den "populære" del af denne instruktionsbog. Det sker dels mere summarisk og dels med flere detaljer end i de foregående ni kapitler.

Fra kapitel 11 finder du også øvelser i slutningen af hvert kapitel. Spring endelig ikke disse opgaver over. Mange af dem belyser ting, du senere må vide, hvis du skal forstå de efterfølgende kapitler.

Brug endelig datamaten så meget som muligt. Kun derved lærer du den så godt at kende, at du virkelig kan få nytte af den. Sid ikke og spekulér over, hvad der kan ske, hvis du gør en eller anden ting. Prøv at indtaste det i maskinen og se, hvad der sker!

Prøv selv at finde på andre eksempler end dem, instruktionsbogen giver dig. Glem ikke, at jo flere programmer du selv laver, des bedre vil du komme til at forstå din Spectrum.

Tastaturet

ZX Spectrums karaktersæt er mere omfattende end de fleste andre mikrodatamaters. Det består ikke blot af enkeltsymboler (bogstaver, tal, tegn o.s.v.), men også af sammensatte symboler, der indskrives fra tastaturet ved tryk på blot én tast (plus evt. en eller to skiftetaster). Disse symboler skal altså ikke staves bogstav for bogstav. Dette er både tids- og lagerbesparende, idet hvert af dem kun optager én byte i datamatens "hukommelse".

For at dække alle disse ordrer eller funktioner har hver af datamatens taster flere muligheder. Nogle taster har hele seks anvendelsesområder.

Hvornår tasterne skal virke på den ene, anden, tredje, fjerde, femte eller sjette måde, afgøres enten automatisk af datamaten eller af brugeren ved hjælp af skiftetasterne **CAPS SHIFT** og **SYMBOL SHIFT**.

De forskellige "indstillinger", der kaldes for "tilstande", vises af markøren i form af et blinkende bogstav, der står dér, hvor næste karakter fra tastaturet skal indskrives. **K**-markøren viser, at datamaten forventer et "keyword" eller "nøgleord". **K**-markøren kommer automatisk, når datamaten venter et sådant ord eller en programlinje. Den kommer frem, 1. når en programlinje skal indledes, 2. efter THEN eller 3. efter tegnet : (undtagen i strenge).

Hvis man derefter ikke trykker på en skiftetast, vil den næste tast blive udlagt som enten et af de nøgleord, der står med hvidt på tasterne, eller et tal.

L-tilstanden dukker op, efter at et nøgleord er indskrevet. Det er datamatens normaltilstand. Hvis der ikke trykkes på en skiftetast, vil datamaten ved en **L**-markør udlægge et tasttryk som det hovedsymbol, der står i hvidt på tasten. Hvis det er et bogstav, vil det blive skrevet som "lille".

Ved både **K**- og **L**-tilstand vil et tryk *samtidigt* på **SYMBOL SHIFT** og en tast af

datamaten blive udlagt som det røde tegn, der står på selve tasten. Mens der ved *samtidigt* tryk på en taltast og **CAPS SHIFT** vil blive udlagt som den kontrolfunktion, der med hvid skrift er trykt over tasten. I **L**-tilstand vil et samtidigt tryk på **CAPS SHIFT** og en bogstavtast medføre, at der skrives med STORE bogstaver. Hvis der er **K**-tilstand, sker der ingenting ved brug af denne skiftetast og bogstavtaster.

Skiftetasten til STORE bogstaver kan "låses". Det sker ved *samtidig* nedtrykning af tasterne **CAPS SHIFT** og **CAPS LOCK**. Herved skifter markøren til **E** og bliver i denne tilstand, indtil der påny trykkes på to taster.

E-markøren anvendes, når man skal bruge nogle af de øvrige karakterer. Man frembringer **E**-tilstand ved at trykke *samtidigt* på begge skiftetaster. **E**-tilstanden varer kun ét tasttryk. I **E**-tilstand får man valget mellem to slags symboler. Hvis man trykker på en af skiftetasterne, får man det,der er skrevet med rødt under tasterne, trykker man ikke skift, får man de grønne symboler over tasterne, Trykker man **SYMBOL SHIFT** og en taltast, får man tegnet, men uden **SYMBOL SHIFT** giver taltasten i denne tilstand den farve, der står over tasten.

G-tilstanden giver grafiktegn. Den skabes ved samtidigt tryk på **CAPS SHIFT** og **9**-tasten. Den vedvarer, indtil der igen trykkes på begge taster eller blot **9**-tasten. I denne tilstand vil en taltast give et af maskinens indbyggede grafiktegn, tasten **0** vil virke som sletletast (**DELETE**), mens alle bogstavtasterne – med undtagelse af **V**, **W**, **X**, **Y** og **Z** – nu giver dig brugerdefinérbare grafiktegn.

Hvis nogen tast holdes nede i mere end to-tre sekunder, bliver den automatisk repeterende.

Indskrivning fra tastaturet vises i skærmens nederste del, hvor hver karakter skrives umiddelbart før markøren.

Markøren kan flyttes til venstre ved *samtidigt* tryk på **CAPS SHIFT** og **5**, til højre med **CAPS SHIFT** og **8**.

Karakteren før markøren kan slettes med **DELETE** (Ø-tasten) plus **CAPS SHIFT**.

En *hel* linje kan slettes ved at trykke **EDIT** (**CAPS SHIFT** plus **1**) og **ENTER**.

Når der tastes **ENTER**, bliver en linje udført, indsat i programmet eller brugt som **INPUT**-data, medmindre den indeholder syntaksfejl. Sådanne fejl markeres ved et blinkende **?** ved siden af fejlen.

Efterhånden som programlinjerne bliver indsat i progammet, vises der en programlistning i skærmens øverste del. Den sidst indtastede linje markeres med denne pil ▶, og denne markør kan flyttes op og ned ved hjælp af piletasterne på **6** og **7**. Hvis man trykker **EDIT**, får man den linje, ▶-markøren peger på, ned i den nederste del af skærmen, hvor man kan redigere den.

Når en kommando er udført, eller et program kørt, vises resultatet i øverste skærm del. Det forbliver her, indtil en ny linje indtastes,eller der trykkes på **ENTER** med en tom linje, eller der trykkes på piletasterne **6** og **7**.

I skærmens nederste venstre hjørne fremkommer en rapportkode, der beskrives i Appendiks B. Koden forbliver på skærmen, indtil man trykker på en ny tast, og koden fungerer som **K**- markør.

Under visse omstændigheder vil **CAPS SHIFT** og **SPACE** virke som **BREAK** og stoppe et kørende program med rapport **D** eller **L**. Disse omstændigheder indtræffer:

1. Ved slutningen af en sætning, mens programmet kører
2. Hvis datamaten bruger båndoptageren eller en anden printer.

Skærmen
Skærmen har 24 linjer hver med 32 karakterer. Den er delt op i to:

Øverst et område på højst 22 linjer til fremvisning af en programlistning eller et programresultat.

Nederst et område til indtastning af kommandoer, programlister eller **INPUT**-data samt til udskrivning af rapportkoder. Dette område kan udvides på bekostning af det øverste område. Hvis der indtastes så mange linjer, at de når programlistningen i skærmens øverste del, ruller (scroller) linjerne i øverste skærmdel opad.

Samme scroll-virkning indtræffer, hvis der indsættes mere end 22 linjer i et program. Hvis denne oprulning medfører, at du ikke kan se teksten, før den ruller ovenud af skærmen, vil datamaten automatisk bede dig om lov med meldingen **scroll**? Hvis du så trykker på tasterne **N**, **SPACE** eller **STOP**, vil programmet standse scroll'ningen med rapportkoden **D BREAK - CONT repeats**, mens alle de øvrige taster får skærmen til at scroll'e videre.

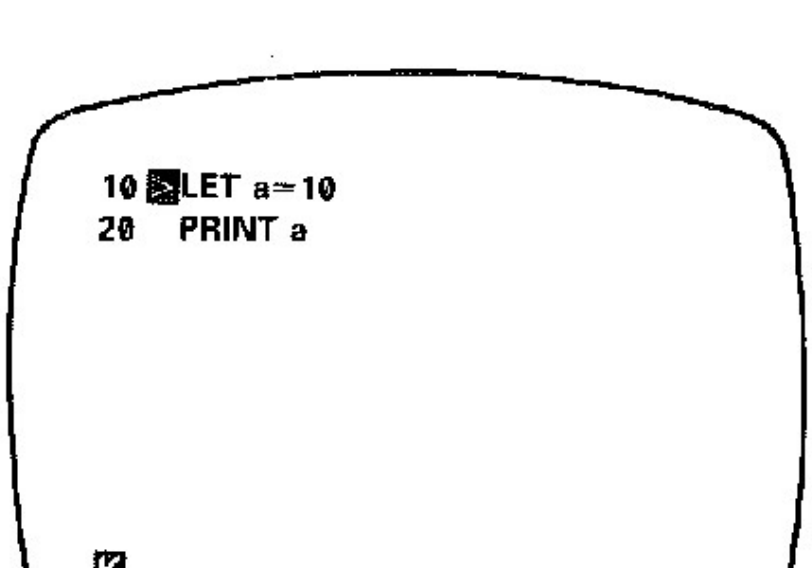
11

11. Begreber i BASIC-programmering

Indtast dette program, der beregner summen af to tal:

```
20 PRINT a
10 LET a=10
```

så ser skærmen således ud:



Som du allerede ved, gemmer maskinen programlinjerne, når de begynder med et tal. Du ser, hvordan maskinen selv opstiller programlinjerne i nummerorden: Dette er vigtigt, når programmet køres, og desuden kan du bedre overskue dit program. Du har endnu kun indtastet et tal, så tast:

```
15 LET b=15
```

Bemærk, hvorledes linjen af sig selv smutter ind mellem de to foregående. Det kunne ikke have ladet sig gøre, hvis du havde nummereret dine linjer **1** og **2** o.s.v. Linjenumre skal forøvrigt altid være hele tal mellem **1** og **9999**. Allerede nu bør du gøre det til en vane at lade huller stå åbne i din nummerering af programlinjer. **10**, **20**, **30** o.s.v. er et godt system.

Det er nu nødvendigt at ændre linie 20 til:

```
20 PRINT a+b
```

Du kunne selvfølgelig skrive det hele om igen, men det er lettere at bruge **EDIT**. Den lille pil ▶ i linie 15 kaldes programmarkøren, og den linje, den peger på, er "**den aktuelle linje**". Programmarkøren peger altid på den sidste indtastede linje, men du

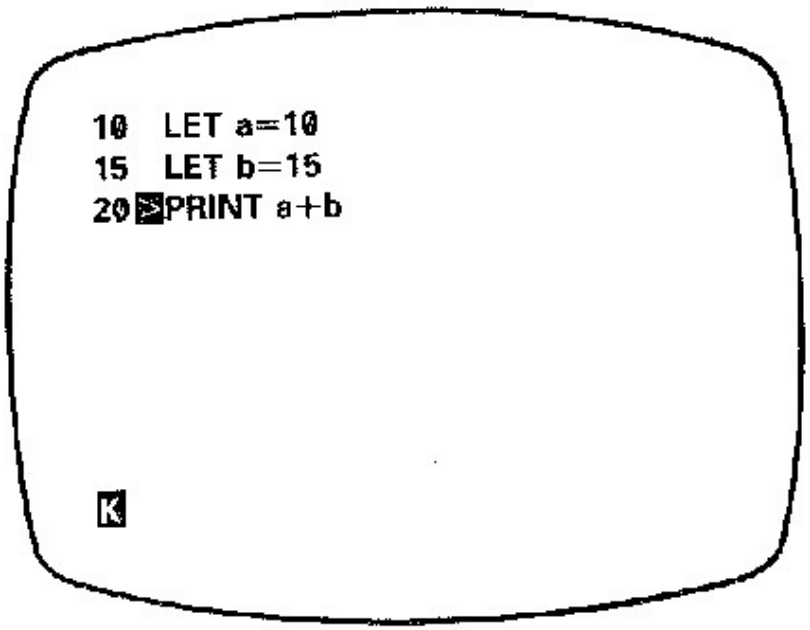
kan bruge "pilene" ▲ ▼ til at flytte programpilen op og ned. Prøv dette, så programpilen peger på linje 20. Tryk nu **EDIT**, og en kopi af linje 20 vil vise sig nederst på skærmen. Tast højrepilen ▶ og hold tasten nede, indtil ▌-markøren kommer hen til slutningen af linjen og tast så:

+b (uden at taste **ENTER**)

Den nederste linje skulle nu gerne indeholde:

20 PRINT a+b

Tryk på **ENTER**, og den gamle linje 20 erstattes af den nye, så skærmen nu ser sådan ud:



Kør dette program ved at taste **RUN** og **ENTER**, og summen af de to tal vil komme frem på skærmen. Kør programmet igen og skriv:

PRINT a,b

Variablerne er der stadig, selv om programmet er slut.

Et lille trick, såfremt du ønsker at slette alt det, der står nederst på skærmen: Tast **EDIT**, så den aktuelle linje kommer frem nederst. Det sletter det, der står forneden. Tast så **ENTER**, og linjen føres tilbage til programmet uden ændringer, mens skærmens nederste del nu kun indeholder markøren.

Lad som om du er lidt fraværende og skriv:

12 LET b=8 (og **ENTER**)

Da der er tale om en fejltagelse, vil du gerne have linien slettet. Det gøres sådan:

12 (og **ENTER**)

Du vil da se, at programmarkøren er væk. Det er den nu ikke helt, den står bare mellem linie 10 og 15, der hvor linje 12 var, så hvis du trykker på ↑, dukker den op i linje 10, og trykker du ↓, flytter den ned til linje 15.

Tast:

12 (og **ENTER**)

Igen vil programmarkørenblivegemt mellem linje 10 og 15. Tryk så på **EDIT**, og linje 15 vil komme ned i skærmens nederste del.

Når programmarkøren er gemt mellem to linjer, bringer **EDIT** *den sidste af de to ned.*

Tast **ENTER** for at rense den nederste del af skærmen. Skriv så:

30 (og **ENTER**)

Denne gang vil programmarkøren være gemt efter den sidste linje i programmet, og **EDIT** vil hente den sidste linje i programmet, nemlig linje 20.

Skriv til sidst:

LIST 15

På skærmen vil du se to sidste linjer: 15 og 20 med programmarkøren i linje 15. Linje 10 er forsvundet fra skærmen, men den findes stadigvæk i dit program.

Det kan du bevise ved at trykke **ENTER**. **LIST 15** bevirker kun, at programlisten begynder med linje 15, og at programmarkøren flyttes til denne linje.

Hvis du har et meget langt program, kan du med fordel bruge **LIST** i stedet for ↑ ↓ piletasterne til at flytte programmarkøren med. Dette illustrerer, hvoriedes linjenumrene kan bruges: de fungerer som navne på programlinjerne, på samme måde som variabler har navne.

LIST

skrevet alene, udskriver programmet helt fra begyndelsen.
I de indledende kapitler så vi kommandoen: **NEW.**

NEW sletter alle gamle variabler og programmer. Indtast nu omhyggeligt dette program, som omregner fra Fahrenheit grader til Celcius grader.

```
10 REM enhedsomregning af temperaturer
20 PRINT "grader i F", "grader i C"
30 PRINT
40 INPUT "Indtast grader i F", F
50 PRINT F, (F-32)*5/9
60 GO TO 40
```

De fleste af bogstaverne i linje 10 må du selv taste ind. Mellemrummet i **GO TO** skriver maskinen selv (**GO TO** er et nøgleord på **G**-tasten).

Kør så programmet. Du vil se, at teksten i linje 20 kommer frem på skærmen, men hvad skete der egentlig med linje 10? Det ser ud, som om maskinen bare har sprunget den over.

Det har den egentlig også. **REM** i linie 10 betyder på engelsk: "remark" (på dansk: bemærkning) og bruges til tider til at minde dig om, hvad programmet kan gøre. En **REM** kommando efterfulgt af hvad som helst, vil blive ignoreret af datamaten ved **RUN**. Maskinen venter nu på, at du skal indtaste en værdi for F, p.g.a. **INPUT** kommandoen i linje 40.

Det kan du se på, at der i stedet for en K-markør nu er en L-markør. Indtast et tal – husk **ENTER**. Datamaten viser resultatet på skærmen og venter på et nyt tal. Det skyldes linje 60: **GO TO 40**, der betyder: gå tilbage til linje 40. I stedet for at løbe tør for program og stoppe, hopper maskinen tilbage til linje 40. Så indtast en ny temperatur.

Efter et par omgange til, begynder du måske at tænke på, om maskinen da aldrig bliver træt af det program. Det gør den ikke. Næste gang den spørger dig om et tal, så tast **STOP**. Maskinen giver dig rapporten: **D STOP in INPUT in line 30:1**, som fortæller dig, hvorfor og hvor den stoppede (den første kommando i linie 30).

Hvis du ønsker at fortsætte, taster du:

CONTINUE

og maskinen vil bede dig om et nyt tal. Maskinen husker nemlig, hvilken rapport den skrev til dig, og i hvilken linje du stoppede den. Så længe rapporten ikke er **0 OK**, hopper maskinen tilbage til linien, hvor den stoppede: Her linje 40.

Udskift linje 60 med **GO TO 31**. Det gør ingen forskel. Når der i en **GO TO**-sætning henvises til et ikke-eksisterende linjenummer, vil hoppet ske til den næste linje derefter. Det samme gælder for **RUN. RUN** alene betyder faktisk **RUN 0**.

Tast nu tal ind, indtil skærmen er fuld. Næste gang du så indtaster et tal, vil maskinen flytte den øverste linje op, så der bliver plads i bunden af skærmen til det indtastede tal. Det kaldes scroll'ning. Når du bliver træt af at indtaste tal, så stop maskinen ved at taste **STOP** og tast derefter **ENTER** for at se programlisten.

Se på kommaet i linje 50. Det er vigtigt, og du skal huske, at maskinen meget præcist skelner mellem f.eks. et komma og et punktum.

Komma bruges, når der skal skrives i venstre margin eller midt på skærmen, afhængig af, hvad der følger efter. I linje 50 bevirker kommaet, at "Celcius grader" bliver skrevet til højre for midten af skærmen.

Hvis du udskifter kommaet med et semikolon, vil det næste tal eller den næste tekst blive udskrevet umiddelbart efter det foregående. Du kan selv prøve at udskifte kommaet med et semikolon i linje 50. Du kan også anvende apostrof (') sammen med **PRINT**-kommandoer. Når maskinen møder en apostrof, skrives de efterfølgende tegn i starten af næste linje. Du får dog ikke megen brug for apostroftegnet, da hver **PRINT**-kommando indeholder en "usynlig" apostrof. Det er derfor, **PRINT** kommandoen i linje 50 altid skriver på en ny linje, og det er også derfor, **PRINT** kommandoen i linje 30 skriver en tom linje.

Prøv at udskifte linje 50 med de følgende linjer, den ene efter den anden og kør hver gang programmet for at se, hvorledes "skilletegnene" virker:

```
50 PRINT F,
50 PRINT F;
50 PRINT F
50 PRINT F'
```

Linjen med kommaet deler op i to kolonner, linien med semikolon skriver det hele efter hinanden, og linjen med apostrof skriver ligesom linien uden tegn på en ny linje. (Bemærk, at **PRINT**, efterfulgt af en apostrof, virker som **PRINT**).

Husk forskellen mellem komma og semikolon i **PRINT** kommandoer. Du må endelig forveksle dem med kolon (:), som bruges til at adskille flere kommandoer i samme linje.

Tast så nogle ekstra linjer ind:

```
100 REM navneprogram
110 INPUT n$
120 PRINT "Hej";n$"!"
130 GO TO 110
```

Dette er et andet program end det sidste, du skrev, men du kan opbevare dem begge i arbejdslageret. Hvis du vil køre det nye program, så skriv:

```
RUN 100
```

Fordi dette program forventer en tekst og ikke et tal, skriver det to anførselstegn. De fortæller dig,at du skal indtaste en streng og sparer dig for noget indtastning. Prøv at skrive et navn, og maskinen svarer dig.

Men du behøver ikke begrænse dig til strengkonstanter (d.v.s. anførselstegn med tegn imellem). Maskinen vil regne på ethvert strengudtryk, også et med strengvariabler i. Hvis du ønsker at bruge strengvariabler, må du slette de to anførselstegn. Fjern dem med højrepilen ▶, og tast derefter **DELETE** to gange og skriv så:

```
n$
```

Da der ikke er nogen anførselstegn, véd maskinen, at den skal regne. I det her tilfælde beregner maskinen værdien af strengvariablen n$, som er lig det, du indtastede første gang. Selvfølgelig virker **INPUT**-kommandoen som **LET n$=n$**, så værdien af **n$** er uændret. Så skriv igen:

```
n$
```

Denne gang skal du ikke slette anførselstegnene. For nu at forvirre dig fuldstændig, har variablen **n$** nu værdien "**n$**".

Såfremt du ønsker at bruge **STOP** ved strong-"input", skal du først have flyttet markøren tilbage til begyndelsen af linjen ved at bruge venstrepilen ◄.

Lad os kigge på det, vi skrev tidligere, nemlig **RUN 100**. Det gør blot, at maskinen hopper til linje 100. Så kunne vi ikke lige så godt have skrevet **GO TO 100?** Det kunne vi udmærket i dette tilfælde. Men der er forskel: **RUN 100** "renser" allerførst maskinens variabler, og bagefter virker det ligesom **GO TO 100**. **GO TO 100** fjerner ikke noget. Til tider vil du sikkert gerne køre et program uden at slette variablerne. I det tilfælde er **GO TO** nødvendigt, mens **RUN** ville være katastrofalt. Så gør det ikke til en vane automatisk at skrive **RUN**, når du vil køre et program.

En anden forskel ligger deri, at du kan skrive **RUN** uden linienummer. Så starter datamaten med programmets første linje. **GO TO** skal altid efterfølges af et linjenummer.

Begge disse programmer stoppede, fordi du skrev **STOP** i **INPUT**-sætningen. Men en gang imellem kan man ved en fejltagelse komme til at skrive programmer, der ikke kan stoppes, og som heller ikke stopper sig selv. Prøv:

```
200 GO TO 200
RUN 200
```

Det ser næsten ud, som om det kunne fortsætte i det uendelige – medmindre du hiver stikket ud. Der er dog en lidt mindre drastisk metode: Tryk på **CAPS SHIFT** og **SPACE**-tasten. Det giver **BREAK**. Programmet stopper med rapporten: **L BREAK into program**.

Ved slutningen af hver linje vil maskinen undersøge, om disse taster er trykket ned. Hvis de er, stopper den. **BREAK** kan også bruges, såfremt du ønsker at stoppe overførsel af data til eller fra båndoptager, printer eller noget andet, der ikke udfører det, du ønsker. Så kommer en anden **BREAK**-rapport, nemlig: **D BREAK-CONT repeats**. **CONTINUE** vil i det tilfælde (og de fleste andre) gentage linjen, hvor den stoppede. Men med **CONTINUE** efter **L BREAK into program** fortsætter datamaten med næste linje.

Kør "navneprogrammet" igen, og når det spørger efter **INPUT**, så fjern anførselstegnene og skriv:

```
n$
```

Selvfølgelig vil **n$** være en udefineret variabel, og du får fejlrapporten: **2 Variable not found** (på dansk: udefineret variabel). Hvis du skriver:

```
LET n$="noget sludder"
```

der giver rapporten **0 OK,0:1**, og derefter

```
CONTINUE
```

vil du se, at du sagtens kan bruge **n$** som **INPUT**-data.

I det her tilfælde virker **CONTINUE** som **GO TO 110**. Kommandoen overser rap-

porten **0 OK**, fordi den ikke indeholder nogen fejl, og hopper tilbage til linjen, hvor den fik fejlrapport: **2 Variable not found.** Det er faktisk ret nyttigt, for hvis maskinen stopper af den ene eller grund, så kan du lave de nødvendige rettelser og derefter fortsætte ved brug af **CONTINUE**.

Den funktion, der fremkalder automatisk listning, har måske forvirret dig noget. Hvis du taster et program på 50 linier, alle **REM**-sætninger, kan du eksperimentere lidt:

```
1 REM
2 REM
3 REM
: :
: :
49 REM
50 REM
```

Det første, du må huske, er, at den aktuelle linje med ▶ altid vises på skærmen, helst tæt ved midten.

Skriv:

LIST (og **ENTER**)

og når maskinen spørger: **scroll**? (fordi skærmen er fyldt), så tast **n** for "nej". Maskinen giver dig rapporten: **D BREAK-CONT repeats**, som når du taster **BREAK**. Prøv i stedet at skrive "**y**", når maskinen spørger: **scroll?**. "**n**", **SPACE og STOP** forstår maskinen som "nej", hvorimod samtlige andre forstås som "ja". Skriv nu **ENTER**, så du får linierne 1-22 frem på skærmen.

Tast så:

23 REM

så får du linierne 2-23 frem.
Skriv:

28 REM

så får du linjerne 27-28 frem på skærmen. I begge tilfælde har du, ved at indtaste en ny linje, flyttet programmarkøren, så maskinen skal lave en ny listning.

Ser det en smule tilfældigt ud? Jamen, datamaten prøver faktisk at give dig det, den tror, du ønsker. Men mennesker er nogle underlige, uforudsigelige væsener, og derfor gætter datamaten somme tider forkert.

Den holder regnskab ikke alene med den aktuelle linje, men også med den øverste linje på skærmen. Når den prøver at at lave en programliste, prøver den først at sammenligne den aktuelle linje med den øverste linje.

Hvis den øverste linje ligger efter den aktuelle, er der ingen idé i at starte der. Så bruger datamaten i stedet den aktuelle linje til ny øverste linje.

Ellers prøver den at udskrive en programliste fra den øverste linje, og det bliver den ved med, indtil den aktuelle linje er med på skærmen – om nødvendigt ved at rykke nogle linjer op!

Men først beregner datamaten, hvor lang tid dette vil tage, og hvis den finder svaret: "for lang ting", så flytter maskinen den øverste linje ned, så den ligger tættere på den aktuelle linje.

Nu har maskinen altså "udarbejdet" en "øverste linje", og den starter udskrivning af programlisten derfra. Hvis den aktuelle linje kommer med på skærmen, når datamaten når til slutningen af programmet eller til det nederste af skærmen, så stopper den. Ellers rykker den linjerne enkeltvis op, sådan at den "øverste linje" bliver rykket én ned for hver gang, og det bliver den med ved, indtil den aktuelle linje er med på skærmen.

Eksperimentér lidt med det her ved at skrive nogle kommandoer på formen:

linjenummer **REM**

LIST flytter den aktuelle linje, men ikke den øverste linje, så en række **LIST**'ninger af det samme program kan godt se noget forskellige ud.

Prøv f.eks. at skrive:

LIST

og tast så **ENTER** igen, så får du linje Ø til at være den øverste linje. Nu skulle du gerne have linjerne 1-22 på skærmen.

Skriv:

LIST 22

som giver dig linjerne 22-43. Når du igen taster **ENTER**, får du linjerne 1-22 tilbage igen. Dette er mere nyttigt ved korte programmer.

Brug dit **REM**-program og tast:

LIST

og når maskinen spørger **scroll?**, så tast **n**. Skriv derefter:

CONTINUE

Her opfører **CONTINUE** sig en smule ejendommeligt, idet den nederste del af skærmen bliver blank. Dette skyldes, at **LIST** var den første kommando i linjen, så den blanke skærm fremkommer ved, at datamaten repeterer **LIST**, når den får kommandoen **CONTINUE**. Maskinen kommer derved til at "køre i ring", og det bliver den ved med, indtil du taster **BREAK**.

Prøv at erstatte **LIST** med:

:LIST

hvor maskinen skriver rapporten: **0 OK**, når du igen taster **CONTINUE**. (**CONTINUE** hopper til den anden kommando i linjen, som er slutningen).

Prøv til sidst at taste:

:: LIST

hvor datamaten nu skriver rapporten: **N Statement lost**. (på dansk: N ingen kommando.). (**CONTINUE** hopper her til den tredie kommando i linjen, hvor der ikke længer er nogen kommando).

Du har nu set kommandoerne: **PRINT, LET, INPUT, RUN, LIST, GO TO, CONTINUE, NEW** og **REM.** De kan alle bruges som enten kommandoer eller programsætninger – det er faktisk tilfældet med de fleste kommandoer i ZX Spectrum BASIC. **RUN, LIST, CONTINUE** og **NEW** er som regel ikke til megen nytte i et program, men de kan anvendes.

Øvelser:

1. Anbring en **LIST**-kommando i et program, så programmet under kørsel **lister** sig selv.

2. Skriv et program, der på basis af **INPUT** om priser, fortæller dig momsbeløbet (22%). Indfør nogle **PRINT**-sætninger, således at programmet fortæller dig, hvad det laver. Endelig skal du lave programmet om, så du kan tage højde for, at moms'en kan stige (ved at indføre moms-procenten i en **INPUT**-sætning).

3. Lav et lille program, der hele tiden fortæller dig om summen af de tal, du kontinuerligt indtaster. (Forslag: Opstil en variabel, der hedder **TOTAL**, og en, der hedder **TAL**, lad **TOTAL** være 0 fra starten, og læg hele tiden **TAL** til, **PRINT** begge og begynd forfra).

4. Hvad kan kommandoerne **CONTINUE** og **NEW** gøre i et program? Kan du finde nogle anvendelser for dem?

12

12. Hvis nu... Hvad så?

De programmer, vi hidtil har beskæftiget os med, har ikke givet os mange overraskelser. Maskinen har udført instruktionerne en for en og er måske til sidst gået tilbage til udgangspunktet. Den slags er der jo ikke meget ved. En ægte datamat skal kunne træffe nogle beslutninger; vælge imellem nogle muligheder.

Det gør ZX Spectrum ved at bruge **IF-THEN**-sætninger. Datamaten bruger **IF-THEN**, sådan: **IF** (hvis) et eller andet sandt eller usandt, **THEN** (så) gør et eller andet.

Rens maskinen ved brug af **NEW** og indtast dette program, der er et spil for to personer:

```
10 REM Gaet et tal
20 INPUT a:CLS
30 INPUT "Gaet et tal", b
40 IF b=a THEN PRINT "Du har fundet tallet": STOP
50 IF b<a THEN PRINT "For lille tal, prøv igen"
60 IF b>a THEN PRINT "For stort tal, prøv igen"
70 GO TO 30
```

Som du kan se, har **IF – THEN**-sætningerne følgende opbygning:

IF betingelse **THEN** ...

hvor "...", står for en række kommandoer adskilt med kolon. Betingelsen er et udtryk, der enten skal være **sandt** eller **falsk**. Hvis det viser sig, betingelsen er sand, så bliver kommandoerne efter **THEN** udført, hvis ikke, springer datamaten videre til næste linje.

De oftest benyttede betingelser er sammenligninger af to tal eller to strenge. Betingelserne afprøver om to tal er ens, eller om det ene er større end det andet. Ligeledes kan to strenge sammenlignes.

De kan være identiske, eller den ene kan komme før den anden alfabetisk.

Vi bruger tegnene:

=	(er lig med)
<	(mindre end)
>	(større end)
<=	(mindre end eller lig med)
>=	(større end eller lig med)
<>	(forskellig fra)

Læg mærke til, at tegnet = her ikke betyder det samme som i **LET**-sætninger.

Det, der her er kaldt betingelse, kaldes også ofte for "udsagn". Eks.:

12. Hvis nu ... Hvad så?

1<2
-2<-1
-3<1

er alle sande udsagn, men

1<0
0<-2

er begge falske, fordi én ikke kan være mindre end nul eller nul mindre end minus to.

Linje 40 sammenligner a og b. Hvis de er lig med hinanden, så stopper programmet p.g.a. **STOP**-kommandoen. Rapporten i bunden af skærmen: **9 STOP, statement, 30:3**, fortæller, at den tredje kommando i linje 30 er årsag til, at datamaten stoppede.

I linje 50 afgøres, om b er mindre end a, og ligeledes afgøres det i linje 60, om b er større end a. Hvis en af betingelserne er opfyldt, udskrives teksten i den linje, hvor betingelsen er opfyldt, og når maskinen kommer til linje 70, hopper den tilbage til linje 30 og begynder forfra.

CLS i linje 20 sletter skærmbilledet, for at den anden person, som skal gætte tallet, ikke kan se det. (CLS betyder Clear Screen, d.v.s. rens skærmen). Matematikere skriver til tider: ≦,≧ og ≠, i stedet for: <=,>= og <>. De skriver f.eks. også: "2<3<4", som matematisk betyder: "2<3 og 3<4", men dette er ikke muligt i BASIC.

Bemærk: i nogle BASIC-versioner, men ikke i ZX Spectrum-BASIC, kan **IF-THEN**-sætningen tage formen:

IF betingelse **THEN** linjenummer

Ved ZX Spectrum skriver du i stedet:

IF betingelse **THEN GO TO** linjenummer.

Øvelser:

1. Prøv dette program:

10 PRINT "x":STOP:PRINT"y"

Når du kører det, vil datamaten skrive **x** på skærmen og derefter stoppe med rapporten: **9 STOP statement, 10:2**. Tast nu **CONTINUE**.

Det kan være, at du forventede, at **CONTINUE** ville have samme virkning som tidligere, nemlig et hop tilbage til **STOP**-kommandoen, men det ville jo ikke være særlig nyttigt. Derfor er det ordnet således, at når maskinen er stoppet med fejlrapport **9** (**STOP**-udført), så hopper maskinen til kommandoen efter **STOP**. Derfor skriver datamaten **y** og kommer til slutningen af programmet.

13

13. Løkker

Lad os antage, at du ønsker at indtaste fire tal og gerne vil have dem lagt sammen. En måde at gøre det på, er at skrive dette lille program (der er ingen grund til at indtaste det, medmindre du er meget pligtopfyldende):

```
10 LET total=0
20 INPUT a
30 LET total=total+a
40 INPUT a
50 LET total=total+a
60 INPUT a
70 LET total=total+a
80 INPUT a
90 LET total=total+a
100 PRINT total
```

Dette er ikke nogen særlig god programmeringsmetode. Den kan sårnænd være god nok til kun fire tal, men tænk, hvor besværligt det ville være, hvis der var 100 tal.

Det er meget bedre at opstille en variabel, der kan tælle til fire og derefter stoppe programmet. Prøv engang at skrive dette program:

```
10 LET total=0
20 LET taelling=1
30 INPUT a
40 REM taelling=antal gange a er blevet indtastet.
50 LET total=total+a
60 LET taelling=taelling+1
70 IF TAELLING <=4 THEN GO TO 30
80 PRINT total
```

Læg godt mærke til, hvor nemt det vil være at ændre i linje 70, sådan at dette program lægger f.eks. 50 eller 100 tal sammen.

At kunne tælle på denne måde er så nyttigt, at der faktisk er to specielle kommandoer, der kan gøre det lettere endnu. De bruges altid sammen, og med dem ser programmet således ud:

```
10 LET total =0
20 FOR t=1 TO 4
30 INPUT a
40 REM t=antal gange a er tastet ind
50 LET total=total+a
60 NEXT t
80 PRINT total
```

Når du vil lave dit tidligere program om, skal du blot ændre linjerne 20, 40, 60, 70. (**TO** er **SYMBOL SHIFT** og **F**).

Læg mærke til, at vi har ændret "taelling" til "t". Tællevariablen, eller *kontrolvariablen* i en **FOR-NEXT**-løkke, må kun bestå af et enkelt bogstav.

I dette program gennemløber t værdierne 1 *(startværdien)*, 2, 3, og 4 *(slutværdien)*. For hver af disse udføres linjerne 30, 40, 50. Når t er nået til at skulle ændre værdi til 5 (som den jo ikke kan), udføres linie 80.

Ydermere behøver kontrolvariablen ikke at vokse med 1 hver gang. Du kan ændre trin- eller **STEP**-værdien, ved at tilføje et **STEP** til **FOR**-kommandoen. Syntaksen for en **FOR-STEP**kommando ser således ud:
FOR kontrolvariabel=startværdi **TO** slutværdi **STEP** stepværdi. Her er kontrolvariablen et enkelt bogstav, og startværdi, slutværdi og stepværdi er alle numeriske udtryk.

Hvis du udskifter linje 20 med:

```
20 FOR t=1 TO 4 STEP 3/2
```

vil t gennemløbe værdierne 1, 2.5 og 4. Læg mærke til, at du ikke er begrænset til hele tal. Ligeledes behøver kontrolvariablen ikke nøjagtigt at ramme slutværdien. Programmet bliver ved at køre, så længe den er mindre end eller lig med slutværdien.

Prøv dette program, som udskriver tal fra 10 til 1:

```
10 FOR n=10 TO 1 STEP -1
20 PRINT n
30 NEXT n
```

Her skal du tænke på, at STEP opfører sig anderledes end tidligere: når stepværdien er negativ, kører **FOR-NEXT**-løkken, så længe kontrolvariablen er større end eller lig med slutværdien.

Pas på, hvis du kører to **FOR-NEXT**-løkker inden i hinanden. Prøv dette program, som laver dominobrikker (et fuldstændigt sæt) på skærmen:

```
10 FOR m=0 TO 6
20 FOR n=0 TO m
30 PRINT m;":";n;" ";
40 NEXT n
50 PRINT
60 NEXT m
```

n-løkke (linjerne 20–40) og **m-løkke** (linjerne 10–60)

n-løkken er helt omsluttet af **m**-løkken – men det er godt nok. Du skal undgå at konstruere løkker, der overlapper hinanden og ikke er ordentligt inden i hinanden og heller ikke er ordentligt adskilt.

Se på følgende program, som ikke virker:

```
10 FOR m=0 TO 6
20 FOR n=0 TO m
30 PRINT m;":";n;" ";
40 NEXT m
50 PRINT
60 NEXT n
```

m-løkke (linjerne 10–40) og **n-løkke** (linjerne 20–60)

En anden ting at undgå er et hop ind midt i en løkke. Kontrolvariablen sættes nemlig først op, når **FOR**-sætningen udføres. Hvis den sætning er sprunget over, vil **NEXT**-sætningen forvirre maskinen. Hvis du er heldig, får du rapporten: "**NEXT without FOR**", (**NEXT** uden **FOR**), eller: "**variable not found**".

Der er ikke noget, der forhindrer dig i at anvende **FOR-NEXT**-kommandoer direkte uden linjenummer.

Prøv f.eks. dette:

```
FOR m=0 TO 10:PRINT m:NEXT m
```

På den måde kan du få datamaten til at udføre en "serie" udregninger uden program. Du kan somme tider bruge dette i stedet for **GO TO**, som du ikke kan bruge i en direkte ordre, da **GO TO** skal efterfølges af et linjenummer. Eks. således:

```
FOR m=0 TO 1 STEP 0:INPUT a: NEXT m
```

Her får 0 programmet til at gentage sig selv.

Hvis der på en eller anden måde opstår en fejl, så vil du miste kommandoen, og **CONTINUE** vil ikke virke.

Øvelser:

1. En kontrolvariabel har ikke bare et navn og en værdi sådan som en simpel variabel, men også en grænse, en trinværdi, og et linjenummer, som løkken skal gå tilbage til (d.v.s. linjenummeret efter **FOR**-sætningen).

Prøv at overbevise dig selv om, at

```
NEXT c
```

faktisk er det samme som i almindelig ZX Spectrum-BASIC at sige:

```
LET c=c+1
IF c <=5 THEN GO TO (linjen med INPUT)
```

2. Kør det tredie "TOTAL"-program og skriv:

```
PRINT t
```

Hvorfor er svaret 5 og ikke 4? (Svar: **NEXT** i linje 60 udføres 5 gange, og t forøges hver gang med 1). Hvad sker der, hvis du tilføjer **STEP 2** i linje 20?.

3. Lav en ændring i programmet, så det nu ikke bare adderer 4 tal, men spørger dig, hvor mange tal du ønsker adderet.

Skaber det problemer, hvis du nu svarer 0, d.v.s. at du ikke ønsker nogen tal adderet?

Du kan sikkert ændre programmet, så det ikke sker.

4. Prøv at ændre **n** fra **10** til **100** i linje 10 i programmet, der tæller "baglæns". Så bliver tallene fra 100 til 89 udskrevet på skærmen, og maskinen spørger **"scroll?"** – forneden. Det gør den, for at du skal se alle tallene, så de ikke bare "rykker ud" af skærmen.

Hvis du nu taster: "**n**", **STOP** eller **BREAK,** stopper maskinen med rapporten: **D BREAK-CONT repeats.**
Trykker du i stedet på en af de andre taster, vil datamaten skrive de næste 22 tal og spørge: **"scroll?"**.

Derefter kan du prøve at slette linje 30. Når du så kører dette forkortede program, vil datamaten skrive det første tal på skærmen og stoppe med: **0 OK**

Taster du:

NEXT n

vil programmet køre en runde til og skrive det næste tal.

14

14. Underprogrammer

Engang imellem vil forskellige dele af dit program have næsten den samme opgave. Du sidder og skriver de samme linjer flere gange i det samme program. Og det er slet ikke nødvendigt. Du kan nemlig lave et *underprogram* (en subrutine). Disse underprogrammer kan så bruges (eller kaldes) fra ethvert sted i dit program, uden du behøver at skrive dem igen.

Du skal bruge kommandoerne: **GO SUB** (**GO** to **SUB**rutine) og **RETURN** (**RETURN**'er).

```
GO SUB n
```

betyder: gå til underprogrammet, hvis første linje har nummer n. Virkningen er den samme som, hvis du skriver **GO TO n**, men maskinen husker **GO SUB**-sætningens linjenummer (returadressen), som den vender tilbage til efter at have udført underprogrammet. Returadressen husker den ved at lægge den øverst i en stak adresser kaldet: **GO SUB**-stakken.

Ordren **RETURN** fjerner det øverste linjenummer i stakken, og datamaten går videre med den næste kommando.

Lad os prøve at se på "tal-gætte"-programmet. Tast programmet ind igen, men indtast det på følgende måde:

```
10 REM En anden version af talprogrammet
20 INPUT a: CLS
30 INPUT "Gaet tallet",b
40 IF a=b THEN PRINT "Du har fundet tallet": STOP
50 IF a<b THEN GO SUB 100
60 IF a>b THEN GO SUB 100
70 GO TO 30
100 PRINT "Prøv igen"
110 RETURN
```

GO TO-kommandoen i linje 70 er meget vigtig, da programmet uden den ville fortsætte ind i underprogrammet og give fejlrapport: "**7 RETURN without GO SUB**" - (**RETURN** uden **GO SUB**), når det når til **RETURN**- kommandoen.

Her er et andet program, som illustrerer brugen af **GO SUB**:

```
100 LET x=10
110 GO SUB 500
120 PRINT s
130 LET x=x+4
140 GO SUB 500
150 PRINT s
160 LET x=x+2
170 GO SUB 500
180 PRINT s
190 STOP
500 LET s=0
510 FOR y=1 TO x
520 LET s=s+y
530 NEXT y
540 RETURN
```

Kan du se, hvad der foregår? (Underprogrammet starter i linje 500).

Et underprogram kan kalde andre underprogrammer, eller sig selv, så vær ikke bange for at have flere underprogrammer i flere "lag".

15

15. Read, Data, Restore

I nogle af de tidligere programmer så vi, at informationer eller data kan køres direkte ind i datamaten ved brug af **INPUT**-kommandoen. Til tider kan det være upraktisk at indtaste de samme informationer, hver gang programmet køres. Du kan spare tid ved at bruge kommandoerne: **READ**, **DATA**, **RESTORE**.

Skriv f.eks.:

```
10 READ a,b,c
20 PRINT a,b,c
30 DATA 10,20,30
40 STOP
```

En **READ**-kommando skal altid efterfølges af variabler adskilt med kommaer. **READ** virker på en måde som **INPUT**, bortset fra, at maskinen finder variablerne i **DATA**-sætningerne.

Hver **DATA**-sætning indeholder en række udtryk – numeriske størrelser (tal) eller strenge – som skal være adskilt med kommaer. Du kan skrive **DATA**-sætninger alle steder i et program, fordi maskinen ignorerer sætningerne, medmindre den møder en **READ**-kommando. Du skal forestille dig, at udtrykkene i alle **DATA**-sætninger i et program, er én lang "**DATA**-liste". Første gang datamaten "læser" **READ**, tager den det første udtryk fra **DATA**-listen, næste gang tager den det andet udtryk, og således vil maskinen arbejde sig gennem **DATA**-listen. (Hvis en **READ**-kommando evt. forsøger at læse data efter slutningen af **DATA**-listen, vil datamaten skrive en fejlrapport).

DATA kan ikke, som andre kommandoer, bruges direkte, fordi **READ** ikke kan finde **DATA**-informationerne uden linjenummer.

Lad os prøve at se, hvorledes **READ-DATA**-kommandoerne virker i det program, du har indtastet.

Linje 10 fortæller maskinen, at den skal læse tre "stykker data", og den giver dem variabelnavnene **a,b,c**. Linje 20 udskriver disse variabler. **DATA** i linje 30 tildeler hver af variablerne **a,b** og **c** en værdi, og i linje 40 stopper programmet.

Hvis du ændrer linje 20 til:

```
20 PRINT b,c,a
```

kan du se, hvordan maskinen arbejder.

DATA-informationer kan sagtens være en del af f.eks. en **FOR-NEXT**-løkke.

Indtast:

```
10 FOR n=1 TO 6
20 READ d
30 DATA 2,4,6,8,10,12
40 PRINT d
50 NEXT n
60 STOP
```

Når du kører programmet, kan du se, hvordan **READ** systematisk arbejder sig gennem **DATA**-listen.

DATA-sætninger kan også indeholde strengvariabler:

Prøv f.eks. dette program:

```
10 READ a$
20 PRINT "Vi har d.",a$
30 DATA "14. December, 1983"
40 STOP
```

Det er den simple måde at hente informationer fra **DATA**-listen: start fra begyndelsen og gå systematisk igennem informationerne, indtil slutningen af **DATA**-listen.

Du kan imidlertid også få datamaten til at hoppe rundt til forskellige steder i **DATA**-listerne ved at bruge kommandoen **RESTORE**. Hvis du bruger **RESTORE** alene, vil den "nulstille" "pegepinden", og **READ**-sætningerne vil starte udlæsningen af data forfra i den første **DATA**-sætnning.

Skriver du derimod et linjenummer efter **RESTORE**, vil **READ**s "pegepind" starte forfra ved eller efter den pågældende linje.

Prøv dette program:

```
10 READ a,b
20 PRINT a,b
30 RESTORE 10
40 READ x,y,z
60 DATA 1,2,3
70 STOP
```

I linje 10 tildeles variablerne **a** og **b** værdierne 1 henholdsvis 2. **RESTORE 10** i linje 20 "nulstiller" variablerne og tillader **READ** at udlæse **x,y,z** fra starten af **DATA**- linjen.

Prøv at køre programmet uden linje 30 for at se **RESTORE**s virkning.

16

16. Udtryk og variabler

Du har allerede set flere af de metoder, som ZX Spectrum anvender, når den regner med tal. Den regner med de "fire" regnearter: +,–,*,/, (husk at * betyder "gange" eller multiplikation, og at / betyder "dele" eller division), og den kan beregne værdien af en variabel.

Lad os f.eks. se på:

```
LET TAKST=sum*15/100
```

som illustrerer, hvorledes operationer og variabler kan kombineres. En kombination som f.eks.: **sum*15/100**, kaldes et "udtryk". Så et "udtryk" er bare en kort måde at fortælle datamaten, at den skal udføre flere udregninger, én for én. I vort eksempel betyder "udtrykket": **sum*15/100**: undersøg værdien af variablen med navnet "sum", gang den med 15 og dividér med 100.

Har du endnu ikke den fulde forståelse af, hvorledes ZX Spectrum opererer med tal, og i hvilken rækkefølge den prioriterer matematiske udtryk, vil vi anbefale dig at læse de første kapitler i bogen. Lad os kort resumére:

Multiplikation og division udføres først. De har højere prioritet end addition (+) og subtraktion (–). Multiplikation og division har samme prioritet, og det betyder, at de bliver udført i rækkefølgen venstre mod højre.

Efter multiplikation og division kommer addition og subtraktion, og de prioriteres også ens, så de bliver udført i rækkefølgen venstre mod højre.

For at gøre dette lidt tydeligere kan vi forklare det ved at give hver operation en prioritering og hæfte et tal på fra 1 til 16:

* og / har prioriteret 8
\+ og – har prioriteret 6

Operationernes rækkefølge er således helt fast, men du kan godt omgå den, nemlig ved at bruge parenteser. Det, der står i parentes, bliver udregnet først og bliver derefter betragtet som et tal.

Normalt kan du, nårsomhelst maskinen forventer, du giver den et tal, skrive et "udtryk" i stedet, og den vil så regne resultatet ud. Du kan også sammenlægge flere strenge (eller strengvariabler) i et "udtryk", og du kan endda anvende parenteser, hvis du ønsker det.

Vi vil nu fortælle dig, hvad du må (og hvad du ikke må) anvende som navne for variabler. Som vi fortalte tidligere, skal navnet for en strengvariabel bestå af et bogstav efterfulgt af et dollar-tegn **$**, og navnet på kontrolvariablen i en **FOR- NEXT**-løkke skal være et enkelt bogstav.

For numeriske variabler er reglerne anderledes friere: Du kan bruge alle sammensætninger af bogstaver og tal, når blot det første tegn er et bogstav. Der kan også være mellemrum i et numerisk variabelnavn, hvis navnet ønskes lettere læseligt, men de tæller ikke med som en del af navnet. Ligeledes gør det ingen forskel, om navnet er skrevet med store eller små bogstaver.

Her er nogle eksempler på tilladte variabelnavne:

x
t42
dette navn er så langt, at det kan være svært at huske, når det skal bruges igen

nu er vi seks
nU Er VI sekS
} Disse to navne henviser til den samme variabel.

Eksempler på variabel-navne, der ikke er tilladt:

2001	(begynder med et tal)
3 øl	(begynder med et tal)
M*A*S*H	(* er hverken et tal eller et bogstav)
Hans-Kristian	(- er heller ikke et tal eller bogstav)

Numeriske udtryk kan skrives som et tal efterfulgt af an eksponent (**e**). Prøv at skrive følgende:

PRINT 2.34e0
PRINT 2.34e1
PRINT 2.34e2
PRINT 2.34e3

o.s.v. op til

PRINT 2.34e15

Du vil se, at ZX Spectrum af sig selv bruger eksponentiel notation. Det skyldes, at den aldrig kan bruge mere end 14 pladser til at skrive et tal.
Prøv ligeledes at skrive:

PRINT 2.34e-1
PRINT 2.34e-2

o.s.v.

PRINT-kommandoen kan højest udskrive 8 betydende cifre.

Skriv

PRINT 4294967295,4294967295-429e7

Dette viser, at ZX Spectrum gemmer alle cifrene, selv om den ikke kan vise dem alle på en gang.

ZX Spectrum regner med "flydende komma", hvilket betyder, at den holder separat regnskab med cifrene i et tal og placeringen af kommaet. Det betyder, at den ikke altid husker et tal helt korrekt – selv ikke, hvis det er et helt tal.

Skriv:

PRINT 1e10+1-1e10,1e10-1e10+1

Tal opbevares med ca. 9 1/2 ciffers præcision, så 1e10 er allerede for stort et tal til, at det kan huskes helt nøjagtigt. Unøjagtigheden er større end 1, her faktisk ca. 2, så tallene 1e10 og 1e10+1 for maskinen synes lige store.

Et mere bemærkelsesværdigt eksempel ser du, hvis du skriver:

PRINT 5e9+1−5e9

Her er unøjagtigheden kun ca. 1, og med 1-tallet, der lægges til, "runder" datamaten op til 2. Tallene 5e9+1 og 5e9+2 er for maskinen ens. Det største heltal, som maskinen kan gemme helt præcist er 2^{32}-1 eller (4.294.967.295), men det rækker nok også ret langt. Strengen "", der ikke indeholder karakterer (heller ikke mellemrum), kaldes en *tom streng.*

Prøv:

PRINT "Har du set"Borte med blaesten"fornylig?"

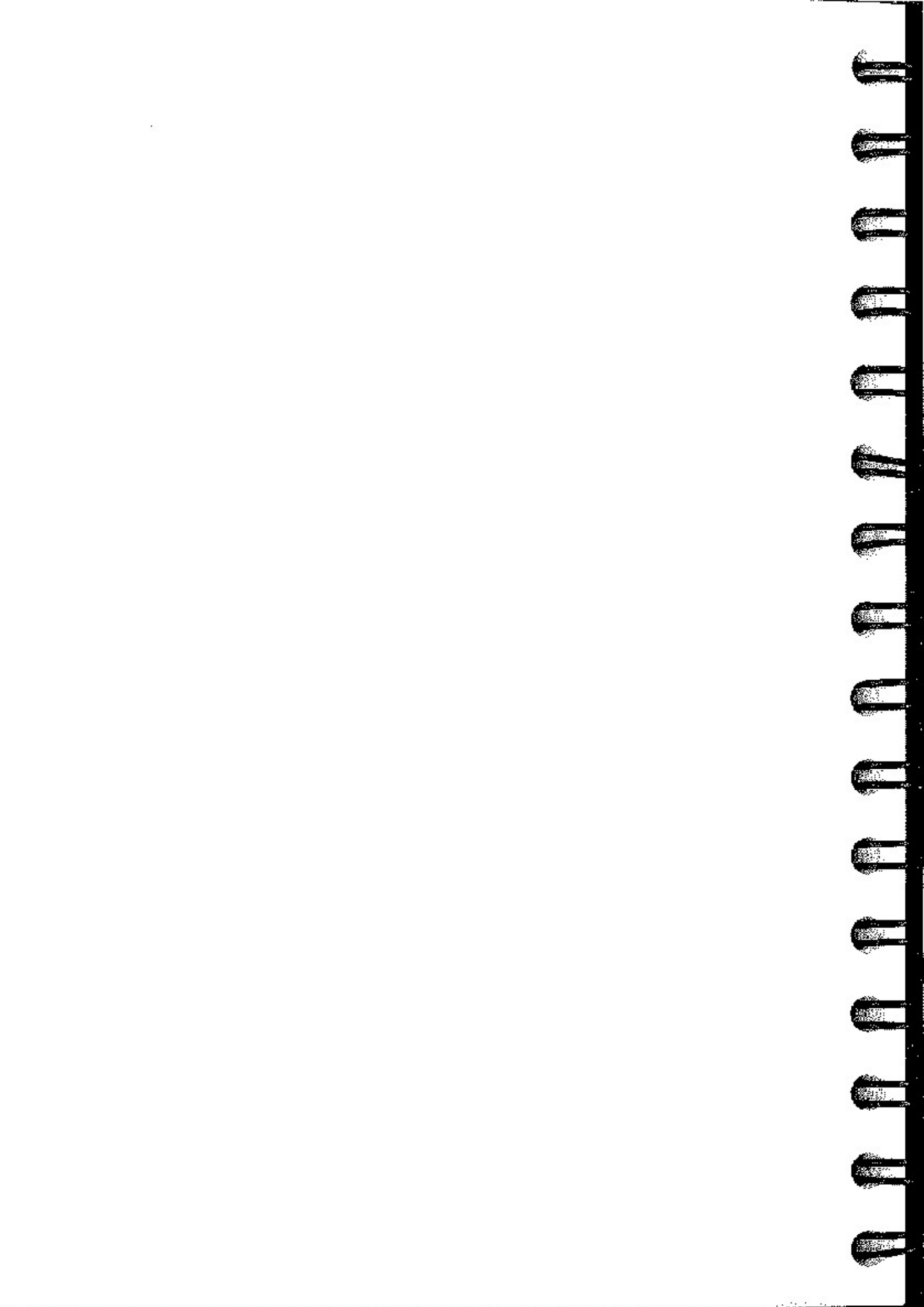
Når du trykker **ENTER**, vil du se et blinkende spørgsmåltegn, som fortæller, at der er noget galt. Dette skyldes, at maskinen tror, at det første anførselstegn i *Borte med blaesten* er slutningen på strengen, og derfor kan den ikke finde ud af, hvad der står bagefter.

Men den kan løse problemet ved at skrive to anførselstegn efter hinanden, hvor du skrev et!

Prøv at ændre **PRINT**-linjen til:

PRINT "Har du set""Borte med blaesten"" ?"

Som du kan se på skærmen, bliver "dobbelt-anførselstegnene" kun udskrevet som et tegn.



17

17. Strenge

Bemærk: Strengbehandlingen på ZX Spectrum er ikke standard.

Hvis vi har en tekststreng, så er en *delstreng* en del af denne streng med karakterne ordnet i den samme rækkefølge. Hvis vi holder os til denne definition, så er "**STRENG**" en delstreng af "**TEKSTSTRENG**", mens "**- TESTSTRENG**" og "**KSSTRENG**" ikke er det. Der findes et begreb, som vel kan oversættes til "skiveskæring" eller "salamimetoden", der på det datamatiske fagsprog hedder "**slicing**". En "**slicing**" kan illustreres på denne måde:

et strengudtryk (start **TO** slut)

eller f.eks.

"abcdef"(2 TO 5)= "bcde"

ZX Spectrum "skærer" altså et stykke ud af tekststrengen mellem 2 og 5.

Såfremt det første tal udelades, antager maskinen, at der menes 1 – hvis sidste tal udelades, antages det, at der menes til slutningen af strengen.

Derfor gælder det:

"abcdef"(TO 5)="abcdef"(1 TO 5)="abcde"
"abcdef"(2 TO)="abcdef"(2 TO 6)="bcdef"
"abcdef"(TO)="abcdef"(1 TO 6)="abcdef"

En lidt anderledes form udelader **TO** og har bare et tal:

"abcdef"(3)="abcdef"(3 TO 3)="C"

Selv om det sædvanligvis kræves, at tallet både for begyndelse og slutning af en "**slicing**" skal indeholde en del af strengen, så overskygges denne regel af en anden, der siger, at hvis begyndelsen af "**slicingen**" ligger længere ude end strengen, så er indholdet af delstrengen en "tom streng", d.v.s., når du skriver

"abcdef"(5 TO 7)

får du fejlrapport: **3 subscript wrong**, (3 forkert indeks), fordi strengen kun indeholder 6 karakterer, og slutningen (7) ligger længere ude end strengen.

Men du kan skrive:

"abcdef"(8 TO 7)=""

og

"abcdef"(1 TO 0)=""

Hvis start eller slutning er negativ, får du fejlrapport: **B integer out of range,** (heltal uden for definitionsområdet).

Det næste program illustrerer nogle af reglerne:

```
10 LET a$="abcdef"
20 FOR n=1 TO 6
30 PRINT a$(n TO 6)
40 NEXT n
50 STOP
```

Når du har kørt programmet, så tast **NEW** og indtast dette program:

```
10 LET a$="1 2 3 4 5 6"
20 FOR n=1 TO 10
30 PRINT a$(n TO 10),a$((10-n) TO 10)
40 NEXT n
50 STOP
```

For strengvariabler gælder det, at ikke alene kan vi lave "**slicing**", men vi kan også henvise til de opståede delstrenge – prøv f.eks. at skrive:

```
LET a$="Jeg er ZX Spectrum"
```

og derefter:

```
LET a$(5 TO 8)="********"
```

og

```
PRINT a$
```

Da delstrengen **a$(5 TO 8)** kun er fire karakterer, vil maskinen selv skære de overflødige *'er væk. En delstreng skal altid være samme længde ved henvisning som før. Derfor forlænges delstrengen med mellemrum eller forkortes til den rigtige længde. Dette kaldes en *procrusteansk tilpasning* efter kroværten Procrustes, der enten strakte sine gæster på en pinebænk eller skar deres fødder af, fordi han ønskede, at gæsterne skulle passe til sengens længde.

Hvis du prøver at skrive:

```
LET a$()="hallo"
```

og

```
PRINT a$;"."
```

vil du se, at der sker det samme igen (denne gang med mellemrum) da **a$** tæller som en delstreng.

Hvis du skriver:

```
LET a$="hallo"
```

vil den skrive det rigtigt.

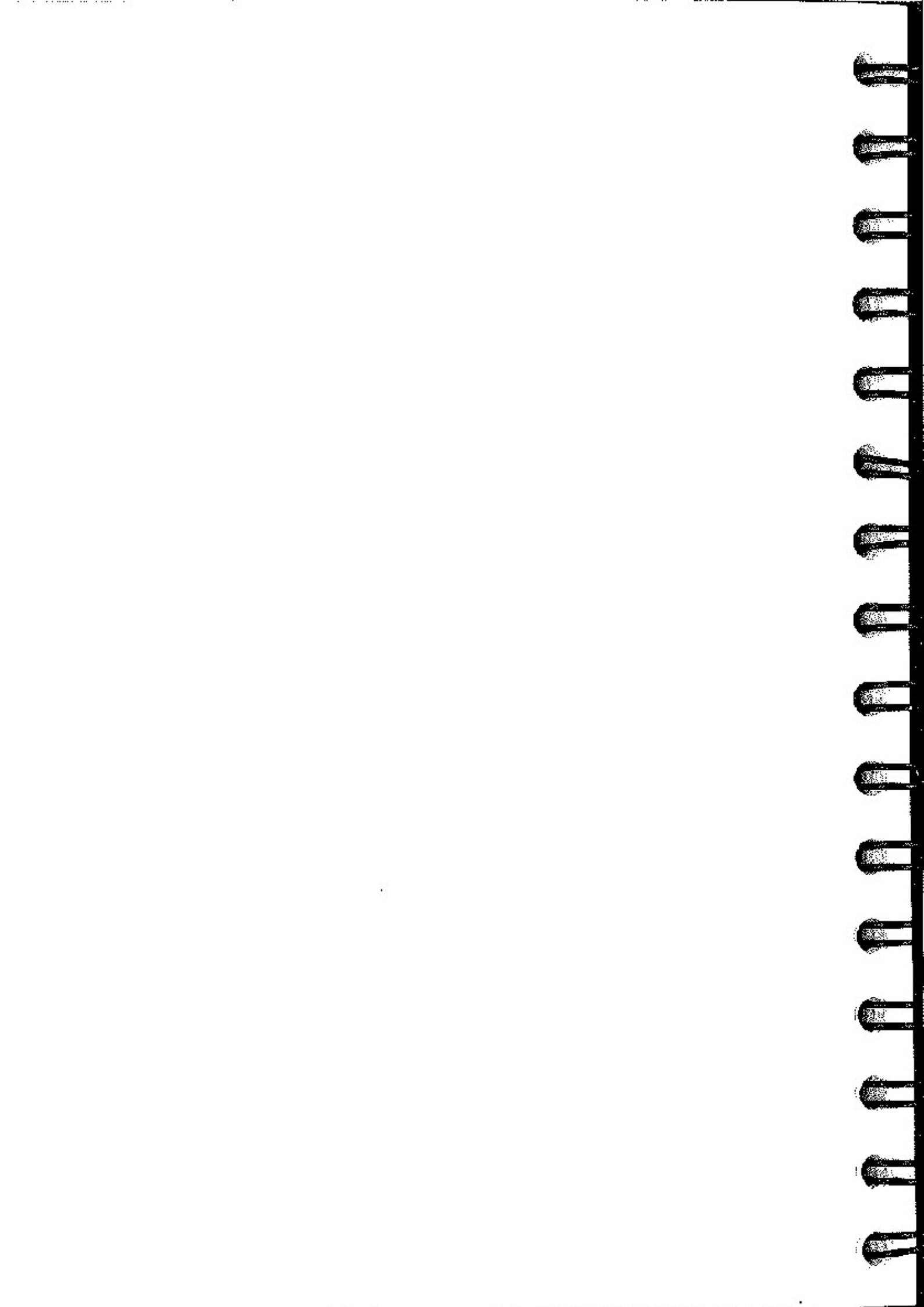
Komplicerede strenge eller sætninger behøver en hel del parentesor, før de kan deles.

Eeks.:

```
"abc"+"def"(1 TO 2)="abcde"
("abc"+"def")(1 TO 2)="ab"
```

Øvelse:

1. Prøv at skrive et program, som skal udskrive dagene i en uge ved at bruge streng-**"slicing"**. Forslag: lad strengen være: ManTirOnsTorsFreLørSøn.




18

18. Funktioner

Forestil dig en valsemaskine til korn. Hvis du putter hvedekorn i maskinen og tænder den, så kommer der valset hvede ud i den anden ende. Havrekorn bliver til valset havre, bygkorn bliver til valset byg o.s.v. Efter samme metode virker funktioner. Du tilfører maskinen en værdi (kaldet: "argumentet"), og den regner på det og når til sidst frem til resultatet.
Princip:

Korn ind → Valsemaskine → Valset korn ud
Argument ind → Funktion → Resultat ud

Forskellige argumenter giver forskellige resultater, og hvis argumentet er forkert, vil funktionen stoppe og skrive en fejlrapport.

På samme måde, som du kan have forskellige maskiner til at lave forskellige produkter, vil forskellige funktioner udføre forskellige udregninger. Hver funktion får sin egen værdi, for at den kan skelnes fra andre.

Du bruger en funktion ved at skrive navnet for den og dernæst argumentet. Når maskinen har vurderet funktionsudtrykket, vil den udarbejde resultatet af funktionen.

Eeks. er **LEN** en funktion, som udregner længden af en streng. Her er argumentet den pågældende streng, du ønsker at finde længden af, og resultatet er længden af den.

Hvis du skriver:

PRINT LEN "Sinclair"

vil datamaten skrive svaret 8 (antal karakterer i "Sinclair"). Du får fat i **LEN** og i de fleste andre funktioner ved at trykke på **CAPS SHIFT** og **SYMBOL SHIFT** på samme tid, sådan at K-markøren skifter til E-markør. Tryk nu på **K**-tasten for at få **LEN.**

Hvis du blander funktioner og regneoperationer sammen i et udtryk, så vil funktionerne blive behandlet *før* regneoperationerne. Men også her kan disse regler omgås ved at anvende parenteser.

Prøv at se på dette eksempel:

1)
LEN "Per"+ LEN "Jensen"
4+ LEN "Jensen"
4 + 6
10

2)
LEN("Per"+"Jensen")
LEN("PerJensen")
LEN "PerJensen"
10

De to udtryk adskiller sig fra hinanden p.g.a. parenteserne. I dette her eksempel bliver resultatet det samme, selv om udregningerne bliver udført i hver sin rækkefølge.

Her er nogle flere funktioner:
STR$** omdanner tal til strenge: Argumentet er et tal, og resultatet er en streng, som vil komme frem på skærmen, hvis tallet skrives med en **PRINT**-kommando. **$ efter **STR** viser, at resultatet skal være en streng.

Prøv at skrive:

LET a$ = STR$ 1e2

der betyder det samme som:

LET a$ ="100"

Du kan også skrive:

PRINT LEN STR$ 100.0000

hvor du får svaret 3, fordi **STR$ 100.0000 = "100"**

VAL minder lidt om **STR$**, bare modsat: **VAL omdanner strenge til tal.** Eks.

VAL "3.5"=3.5

På en måde er **VAL** det modsatte af **STR$**, for hvis du bruger **STR$** på et tal og derefter bruger **VAL** på det, så vil du komme tilbage til det oprindelige tal. Hvis du bruger **VAL** på en streng og dernæst bruger **STR$** på det, så er det ikke altid, du kommer tilbage til den oprindelige streng.

VAL er en meget værdifuld funktion, fordi strengen, som er argumentet, ikke er begrænset til at være et præcist og "klart" udtryk – det kan være et hvilket som helst numerisk udtryk.

VAL opfører sig f.eks. således:

VAL "2*3" = 6

og

VAL ("2"+"*3") = 6

I det første eksempel bliver **VAL**'s argument behandlet som en streng. Strengudtrykket **"2"+"*3"** bliver sammenregnet til strengen: **"2*3"**, og derefter bliver udtrykket behandlet som et tal. Så **2*3** beregnes til værdien 6.

Dette kan være meget forvirrende, hvis du ikke passer på!

Prøv at skrive:

PRINT VAL"VAL""VAL""""2""""""""

(Husk at anførselstegnene i en streng skal skrives to gange. Hvis du går videre med endnu mere komplicerede strenge , vil du blive nødt til at firedoble eller ottedoble anførselstegnene).

Der findes en anden funktion, som minder lidt om **VAL**, nemlig **VAL$**. Argumentet for **VAL$** er en streng, men resultatet er også en streng. For at se, hvorledes dette virker, så prøv at huske, hvordan **VAL** regner i to trin: Først bliver argumentet behandlet som en streng, og dernæst fjernes anførselstegnene, og det, der er tilbage, bliver

behandlet som et tal. **VAL$** gør det samme i første trin, men efter at anførselstegnene er fjernet i andet trin, bliver det resterende behandlet som en *anden* streng.

Se her, hvordan **VAL$** omdanner en streng til en anden:

VAL$""""Is-kager"""" = "Is-kager"

Skriv:

LET a$="99"

og prøv derefter, ved at bruge en **PRINT**-kommando, at udskrive følgende: **VAL a$, VAL "a$", VAL""""a$"""", VAL$ a$, VAL$ "a$"** og **VAL$ """"a$""""**. Nogle af udtrykkene vil virke, andre vil ikke. Prøv at forklare, hvorfor maskinen gør, som den gør.

SGN er en fortegnsfunktion (kaldet signum, for ikke at forveksle den med sinus, **SIN**). Det er den første funktion, du har set, som ikke regner på strenge, da både argument og resultat er et tal. Resultatet vil altid være +1,0 eller -1, såfremt argumentet er henholdsvis positivt, nul eller negativt.

ABS er en anden funktion, hvor både argument og resultat er tal. **ABS** omdanner argumentet til et positivt tal, som er resultatet, således at:

ABS -3.2 = ABS 3.2 = 3.2

INT betyder på engelsk "integer part" (på dansk: "hel del"). Funktionen uddrager heltalsdelen af argumentet og runder altid ned:

INT 3.9 = 3

Pas på, når argumentet er negativt, da f.eks.

INT -3.9 = -4.

SQR udregner kvadratroden af argumentet. (Du kan beregne argumentet ved at gange resultatet med sig selv). F.eks.

SQR 4 = 2, da 2*2 = 4
SQR 0.25 = 0.5, da 0.5*0.5 = 0.25
SQR 2 = 1.4142136(tilnærmet), da
1.4142136*1.4142136 = 2.0000001

Hvis du ganger et positivt eller negativt tal med sig selv, så vil resultatet altid være positivt. Det betyder, at der til negative tal ikke findes en kvadratrod. Hvis du evt. prøver at bruge **SQR** på et negativt argument, vil du få fejlrapport: **An Invalid Argument** (på dansk: Ugyldigt argument).

Du har også mulighed for at definere dine egne funktioner. Navnene for disse funktioner er **FN** efterfulgt af et bogstav (hvis resultatet er et tal) eller **FN** efterfulgt af et bogstav efterfulgt af et **$**, (hvis resultatet er en streng).

Her skal du være mere påpasselig med parenteser: argumentet skal altid være omsluttet af parenteser.

Du definerer en funktion ved at putte en **DEF FN** ind et eller andet sted i dit program. Her er defineret en funktion, som beregner kvadratet af argumentet:

```
10 DEF FN s(x) = x*x:REM kvadratet af x
```

Markøren skal ændres til en **E**-markør, og derefter tastes **SYMBOL SHIFT** og **1** for at få fat i **DEF**. Du vil se, at **FN** automatisk kommer med, fordi en **DEF**-kommando altid efterfølges af **FN**. Herefter angiver **s** navnet på funktionen. **x** inde i parenteserne er navnet på det argument, som funktionen skal bruge. Navnet skal være et enkelt bogstav (eller et enkelt bogstav efterfulgt af et **$**, hvis argumentet er en streng).

Efter lighedstegnet kommer den egentlige definition af funktionen. Det kan være et hvilket som helst udtryk, og du kan henvise til argumentet ved at anvende det navn, som du giver det (her **x**), som om det var en almindelig variabel.

Når du har indtastet denne linje, kan du bruge funktionen som en af maskinens egne funktioner, ved at taste navnet **FN s**, efterfulgt af argumentet. (**FN** ligger på **2**-tasten). Husk, at når du selv definerer funktioner, så skal argumentet være omsluttet af parenteser.

Prøv det nogle gange – f.eks.:

```
PRINT FN s(2)
PRINT FN s(3+4)
PRINT 1+INT FN s (LEN "kylling"/2+3)
```

Når du har puttet **DEF**-kommandoen ind i dit program, kan du bruge dine egne funktioner i udtryk, lige så frit som datamatens egne.

I nogle BASIC-dialekter skal du have parenteser om argumentet, når du bruger maskinens egne funktioner. Dette er ikke nødvendigt ved ZX Spectrum.

INT runder altid ned. Du kan lave din egen funktion, som afrunder til den nærmeste hele del ved at lægge 0.5 til:

```
20 DEF FN r(x) = INT(x+0.5) : REM afrunder til nærmeste hele tal
```

Du vil så få:

```
FN r(2.9) = 3        FN r(2.4) = 2
FN r(-2.9) = -3      FN r(-2.4) = -2
```

Sammenlign dette med de resultater, du får, hvis du anvender **INT** i stedet for **FN r**.

Indtast og kør følgende program:

```
10 LET x=0 : LET y=0 : LET a=10
20 DEF FN p(x,y) = a+x*y
30 DEF FN q() = a+x*y
40 PRINT FN p(2,3),FN q()
```

Der er flere spidsfindigheder i programmet.

For det første: En funktion er ikke begrænset til ét argument. Der kan være flere eller slet ingen, men du skal stadig huske parenteserne.

For det andet: Det er ligegyldigt, hvor du placerer en **DEF**-kommando, blot den findes i programmet, d.v.s. at når datamaten er færdig med linje 10, hopper den over linje 20 og 30 og fortsætter med linje 40.

DEF kan ikke bruges som kommando uden linjenummer.

For det tredie: **x** og **y** er ikke begge navne på variabler i programmet og navne på argumenter for funktionen **FN p**. Efter **FN p** har anvendt variablerne **x** og **y**, glemmes de midlertidigt, men da der ikke er noget argument kaldet **a**, husker funktionen stadig variablen **a**. Når **FN p(2,3)** er behandlet, har variablen **a** værdien 10, **x** har værdien 2, da **x** er det første argument, og **y** har værdien 3, da **y** er det andet argument. Resultatet er derfor: 10+ 2*3=16.

Derimod er der ingen argumenter i **FNq()**, så her bruger maskinen variablerne i linje 10, som har værdierne: 10, 0, 0. Resultatet er nu: 10+0*10=10.

Prøv at ændre linje 20 til:

```
20 DEF FN p(x,y) = FN q()
```

Nu vil **FN q(2,3)** have værdien 10, da **FN q** stadig vil gå tilbage og bruge variablerne **x** og **y** og fremfor at bruge argumentet **FN p**.

Nogle BASIC-dialekter, men ikke ZX Spectrums, har nogle funktioner, der hedder LEFT$, RIGHT$, MID$ og TL$.

LEFT$ (a$,n) giver den delstreng af a$, der består af de første n-karakterer.

RIGHT$ (a$,n) giver den delstreng af a$, der indeholder karaktererne fra nr. n og videre.

MID$ (a$,n1,n2) giver den delstreng, der består af n2 karakterer, begyndende med nr. n1.

TL$ (a$) giver til slut den delstreng, der, med undtagelse af den første karakter, indeholder hele a$.

Du kan lave nogle brugerdefinérbare funktioner, der kan det samme:

```
10 DEF FN t$(a$) = a$(2 TO) : REM TL$
20 DEF FN i$(a$,n) = a$( TO n ) : REM LEFT$
```

Kontrollér, at de også kan bruges på strenge, der kun indeholder 0 eller 1 karakter. Bemærk, at **FN i$** har to argumenter, hvor det ene er et tal, og det andet er en streng.

En funktion kan have indtil 26 numeriske argumenter (hvorfor 26?), og samtidig indtil 26 streng-argumenter.

Øvelse

1. Anvend funktionen **FN s(x) = x*x**, til at teste **SQR**-funktionen.

Du skulle gerne se, at

```
FN s(SQR x) = x
```

hvis du anvender ethvert positivt tal for **x**, og at:

```
SQR FN s(x) = ABS x
```

hvad enten **x** er positiv eller negativ (hvorfor **ABS**?).

19

19. Matematiske funktioner

Dette kapitel beskriver den matematik, som ZX Spectrum kan anvende. måske får du aldrig brug for nogen matematikfunktioner, så hvis du finder det for tungt, så løb det rask igennem.

Matematikfunktionerne omhandler operationen ↑ (opløftning til potens), funktionerne **EXP** og **LN** og de trigonometriske funktioner **SIN**, **COS**, **TAN** og deres inverse funktioner **ASN**, **ACS**, **ATN**.

Først ↑ og **EXP**:
Du kan opløfte et tal (a) til et andet tal (b). Normalt skrives a opløftet til b sådan: a^b, men dette vil være vanskeligt på en datamat, så vi bruger i stedet symbolet ↑. Eks. opløftet 2 til forskellige tal således:

2↑1 = 2
2↑2 = 2*2 = 4
2↑3 = 2*2*2 = 8
2↑4 = 2*2*2*2 = 16

Vi kan beskrive det elementært med bogstaver: a^b betyder det samme som 'a ganget med sig selv b gange', men dette vil imidlertid kun være fornuftigt, hvis b er et helt positivt tal. For at finde en definition, som kan arbejde med andre værdier af b, må vi se på følgende:

a↑(b+c) = a↑b*a↑c

Bemærk, at vi giver ↑ en højere prioritet end * og /, så når der er flere regneoperationer i et udtryk, bliver ↑ udført før * og /.

Det vil sikkert ikke undre dig, at dette virker, hvis b og c begge er hele positive tal, men hvis vi ønsker, at det skal virke, når b og c ikke er hele positive tal, må vi tage følgende forbehold:

a^0 = 1
a↑(–b) = 1/a↑b
a↑(1/b) = den b'nde rod af a
og a↑(b*c) = (a↑b)^c

Hvis du aldrig før har set noget af det beskrevne, så prøv ikke at huske det præcist. Hvis du kan huske:

a↑(–1) = 1/a
og a↑(1/2) = **SQR** a

og evt. forstår dette, så vil du sikkert senere kunne forstå det andet. Eksperimentér med det ved at prøve dette program:

```
10 INPUT a,b,c
20 PRINT a↑(b+c),a↑b*a↑c
30 GO TO 10
```

Hvis reglerne fra før er sande, så vil maskinen naturligvis udskrive to ens tal, når programmet køres.

Bemærk: p.g.a den metode, datamaten anvender til at udarbejde ↑, må tallet til venstre (**a** i det her tilfælde) aldrig være negativt.

Du kan f.eks. bruge funktionen til at udføre rentesregning med. Grundformlen for beregning af renter over en periode ser således ud: $k_n = k(1+r)^n$, hvor k_n er kapitalen efter n-perioder, r er den aktuelle rente og k er startkapitalen. Lad os antage, vi har 1000 kr. i banken, og at bankrenten er 10 %. Vi ønsker at se, hvor stor kapitalen er f.eks. 5 år senere.

Prøv at skrive:

```
PRINT 1000 (1+0.1)↑5
```

hvor du får resultatet 1610.51

Hvis du prøver at udskifte 5 med 10, vil du opdage, at kapitalen ikke er fordoblet, men vokset betydeligt mere. Dette kaldes eksponentiel vækst.

ZX Spectrum har en funktion som hedder **EXP** (eksponentiel funktion), der defineres således:

EXP x = e↑x

hvor **e** er grundtallet. **e** er ikke et helt tal, men et tilnærmet decimaltal. Du kan se nogle af decimalerne ved at skrive:

```
PRINT EXP 1
```

da **EXP 1 = e↑1 = e**

LN:
(Logaritmefunktion) er det modsatte af **EXP**, d.v.s., at hvis du bruger den ene og derefter, på resultatet, bruger den anden, så får du dit oprindelige tal tilbage. F.eks.:

LN EXP e = EXP LN e = e

eller

LN EXP 2 = EXP LN 2 = 2

Maskinen regner med den naturlige logaritme med grundtallet **e**, men den kan også regne med f.eks. 10-talslogaritme. Når du vil finde den, skal du dividere den naturlige logaritme til tallet x med den naturlige logaritme til 10:

log x = **LN x/ LN 10**

eller

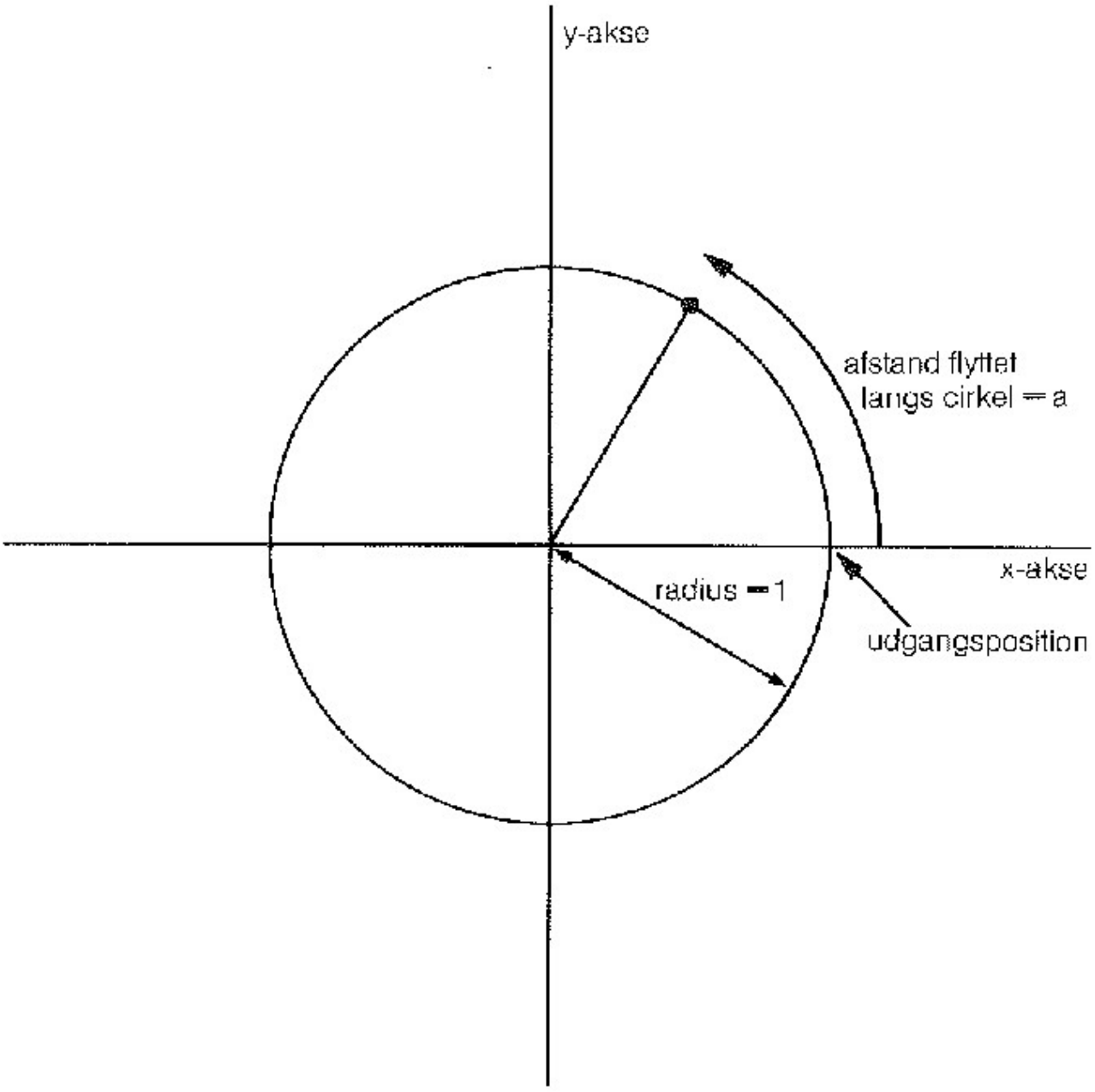
$\log_a x$ = **LN x/ LN a**, hvis **a** er grundtallet.

PI:
Hvis du har en given cirkel, kan du finde omkredsen således: Du ganger diameteren eller 2radius med tallet π, (π er et græsk p og udtales "pi"). Ligesom e, er **PI** også et tilnærmet decimaltal: **PI** = 3.141592653589...

Prøv at taste:

```
PRINT PI
```

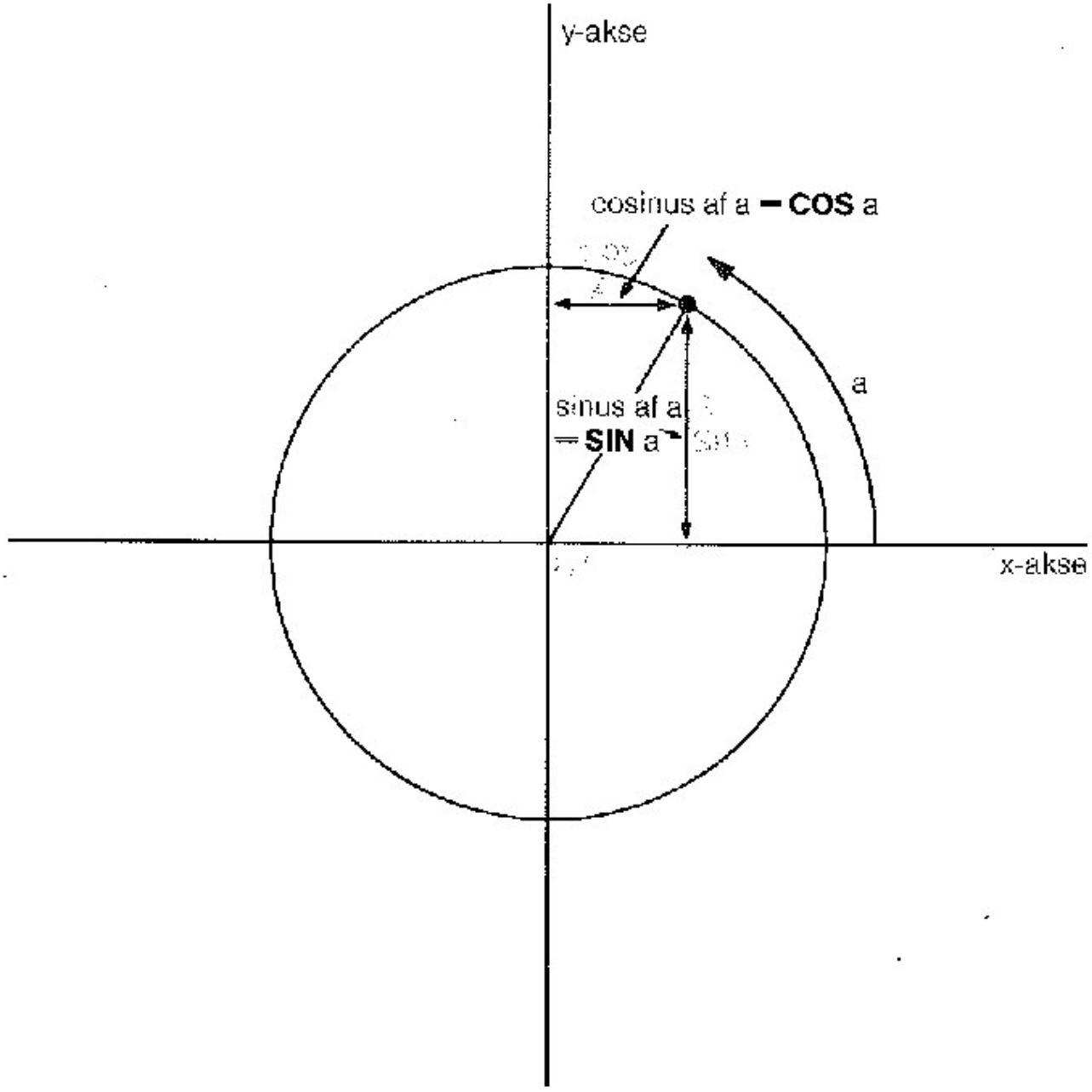
SIN, COS og **TAN; ASN, ACS** og **ATN**
De trigonometriske funktioner kan bruges til at se, hvordan et punkt flytter sig rundt på en cirkel. Lad os antage, at vi har en cirkel med radius 1, (du kan godt regne på cirkler med andre radiusser, hvis du ganger med den samme koeficient alle steder). Startpunktet ligger i klokken 3, og punktet flytter sig rundt på cirklen, modsat uret.



Se på tegningen: linjen fra 9 til kl. 3 kaldes x-aksen, og ligeledes kaldes linjen fra kl. 6 til kl. 12 for y-aksen. For at bestemme, hvor punktet er, undersøger vi, hvor langt punktet har flyttet sig fra udgangspositionen (kl. 3). Lad os kalde denne længde for a. Vi ved, at omkredsen af cirklen er 2PI (fordi radius er 1, og derfor er diameteren 2). Så når punktet har flyttet sig fra kl. 3 til kl. 6, så er a=PI/2. Når punktet er nået halvvejs rundt (fra kl. 3 til kl. 9), så er a=PI, og når punktet har kørt en omgang, så er a=2PI.

Lad os antage, at a er et eller andet tal. Så er afstanden fra y-aksen til punktet matematisk udtrykt: sinus a, og ligeledes er afstanden fra x-aksen til punktet: cosinus a. Funktionerne **SIN** og **COS** på maskinen kan beregne dette.

Der er dog visse forbehold, når du anvender **SIN** og **COS**: Hvis punktet ligger til venstre for y-aksen, så bliver cosinus negativ, og hvis punktet ligger under x-aksen, så bliver sinus negativ.



Endvidere starter sinus og cosinus forfra, når punktet har kørt en omgang (a=2PI):

SIN (a+2*PI) = SINa
COS (a+2*PI) = COS a

Tangens til a er defineret som sinus divideret med cosinus. Denne funktion hedder **TAN** på maskinen.

Til tider kan vi få brug for de modsatte funktioner til sinus, cosinus og tangens, hvis vi ønsker at finde værdien for a. Det er funktionerne **ASN**, **ACS**, **ATN**.

Prøv at se på den anden illustration, hvor punktet flytter rundt på cirklen, og hvor radius er linjen fra centret til punktet. Du bør kunne se, at den længde, som vi kalder a, bruges til at måle vinklen fra x-aksen til radius. Når a er PI/2, så er vinklen 90, når a er PI, er vinklen 180, o.s.v. indtil a er lig 2PI, hvor vinklen så er 360.

ZX Spectrum regner ikke i grader, men i radianer, (radianer er længden af a). Hvis du får brug for at regne i grader, kan du gøre dette ved at dividere med 180 og gange med PI. Hvis du skal gå fra grader til radianer, dividerer du med PI og ganger med 180.

20

20. Tilfældige tal

Dette kapitel omhandler funktionen **RND** og nøgleordet **RANDOMIZE**. De bruges begge, når der skal opereres med tilfældige tal, så pas på, at du ikke forveksler dem. De findes begge på **T**-tasten (på tastaturet er **RANDOMIZE** forkortet til **RAND**).

På en måde er **RND** en funktion: den beregner og producerer et resultat, men den er ualmindelig, fordi den ikke behøver et argument. Hver gang du bruger **RND**, er resultatet et nyt tilfældigt tal mellem **0** og 1. (**RND** kan antage værdien **0**, men aldrig 1.)

Prøv:

```
10 PRINT RND
20 GO TO 10
```

og læg mærke til, hvordan resultaterne varierer. Det er meget usandsynligt, at du kan se noget mønster i svarene. **RND** giver faktisk slet ikke et rigtig tilfældigt tal, men et tal i en række på 65535 forskellige tal, stillet op på en sådan måde, at de synes tilfældige -- de er altså fingeret tilfældige.

RND giver et tilfældigt tal mellem **0** og 1, men du kan let få tilfældige tal i andre områder. F.eks. **5*RND** giver tal mellem **0** og 5 og **1.3 + 0.7*RND** mellem 1.3 og 2. Hvis du ønsker hele tal, kan **INT**-funktionen anvendes (husk at **INT** altid runder ned). Hvis vi f.eks. ønsker at lave hele tal mellem 1 og 6, som ved et terningprogram, kan vi gøre det sådan: **1+INT (RND*6).**

RND*6 giver tal mellem **0** og 6, men da 6 aldrig kommer frem, giver **INT (RND*6)** tallene: **0**,1,2,3,4,5. Prøv at indlaste programmet:

```
10 REM terningprogram
20 CLS
30 FOR n = 1 TO 2
40 PRINT 1+INT (RND*6);" ";
50 NEXT n
60 INPUT a$: GO TO 20
```

Tast **ENTER**, hver gang du vil "kaste" terningerne.

RANDOMIZE bruges til at bestemme, hvor i talrækken **RND** skal starte den "tilfældige" talrække. Det kan du se i dette program:

```
10 RANDOMIZE 1
20 FOR n=1 TO 5: PRINT RND,: NEXT n
30 PRINT : GO TO 10
```

Hver gang maskinen har udført **RANDOMIZE 1**, vil **RND** starte med **0.0022735596**.

Du kan bruge **RANDOMIZE** efterfulgt af ethvert tal mellem 1 og 65535 og starte **RND** forskellige steder.

RANDOMIZE alene (og **RANDOMIZE 0**) er anderledes. Den **RANDOMIZE**rer RND. Se selv:

```
10 RANDOMIZE 0
20 PRINT RND : GO TO 10
```

Den rækkefølge, du får, er ikke særlig tilfældig, fordi **RANDOMIZE** nu bruger tiden, som datamaten har været lændt, til at bestemme, hvor **RND** skal starte.

Du vil få mere tilfældige tal, hvis du udskifter **GO TO 10** med **GO TO 20**.

De fleste BASIC-dialekter bruger **RANDOMIZE** og **RND** til at producere tilfældige tal, men ikke alle bruger dem på samme måde.

Her er et talgætte-program, hvor maskinen selv udvælger det hemmelige tal:

```
10 REM talprogram
20 LET a=1+INT (RND*6)
30 INPUT "Indtast et tal mellem 1 og 100 ";b
40 IF b=a THEN PRINT "Det rigtige tal":STOP
50 IF b<a THEN PRINT "For lille"
60 IF b>a THEN PRINT "For stort"
70 GO TO 30
```

Øvelser:

1. Afprøv denne regel:
Lad os sige, at du vælger et tal mellem 1 og 872 og skriver:

RANDOMIZE og dit tal

Så vil den næste værdi af **RND** blive:

(75*(dit tal+1)−1)/65536

2. *For matematikere:*

Lad p være et (stort) primtal og lad a være en primitiv rod *modulo* p. Hvis b_i så er resten ved a_i *modulo* p($1 \leq b_i \leq p-1$), så er rækken:

$$\frac{b_i - 1}{p-1}$$

cyklisk og består af p−1 bestemte tal i området fra 0 til 1 (aldrig 1). Ved at vælge en hensigtsmæssig værdi for a kan rækken fås til at se ret tilfældig ud. 65537 er et såkaldt mersennsk primtal lig med $2^{16}-1$. Brug dette og Gauss' lov om kvadratisk reciprocitet til at vise, at 75 er en primitiv rod *modulo* 65537.

ZX Spectrum bruger p=65537 og a=75 og gemmer nogle værdier af b_i-1 i hukommelsen.

Funktionen **RND** erstatter b_i-1 med $b_{i+1}-1$ i hukommelsen og giver resultatet $(b_{i+1}-1),(p-1)$. **RANDOMIZE** n ($1 \leq n \leq 65535$) gør $b_i = n$ i 1.

RDN er omtrent ensartet fordelt over området fra 0 til 1.

21

21. Talsæt eller indicerede variabler

(Den måde, som ZX Spectrum behandler indicerede variabler på, er ikke helt standard).

Lad os forestille os, at du har en række tal f.eks. karaktererne fra 10 personer i en klasse. Variablerne, der skal indeholde selve tallene for personerne, kunne du så kalde pers 1, pers 2, pers 3 o.s.v., men programmet ville være langsomt at skrive. Det ville være meget nemmere, hvis du bare kunne skrive f.eks.:

```
10 REM Dette program virker ikke
20 FOR n=1 TO 10
30 READ pers n
40 NEXT n
50 DATA 10,2,5,19, 16,3,11,1,0,6
```

Desværre, det kan du *ikke*.

Men der er en metode, hvorved du kan anvende denne idé. Den opererer med *talrækker* – eller *indicerede variabler*. En indiceret variabel er en række af variabler, alle med det samme navn, men hver med sit tal som eget "efternavn". Dette tal skrives i parentes efter navnet. I vort eksempel kunne navnet være p (en indiceret variabel må kun have et bogstav i navnet). De ti variabler ville så komme til at hedde p(1), p(2), o.s.v. op til p(10). Før du kan bruge indicerede variabler, skal du reservere lidt plads til dem inde i datamaten, og dette gøres ved hjælp af en **DIM**-kommando, kommer af ordet: **dim**ensioner).

```
DIM p(10)
```

reserverer plads til den indicerede variabel p med dimensionen 10. Samtidig sættes værdien af de 10 variabler til 0. Kommandoen sletter også en eventuel foregående indiceret variabel med navnet p.

Derimod har disse variabler ingen indflydelse på simple talvariabler (dem kender du fra tidligere), og en simpel variabel ved navn p kan altså sagtens eksistere samtidig uden forvekslingsmulighed – for den har jo intet "efternavn".

Nu kan du altså lave det umulige program fra før om til:

```
10 FOR n=1 TO 10
20 READ p(n)
30 NEXT n
40 DATA 10,2,5,19,16,3,11,1,0,6
```

En linje har en dimension, et punkt har ingen, et rektangel har to, og en kasse har tre. I ZX Spectrum er vi ikke begrænset til kun at have indicerede variabler i en dimension, men kan sagtens have dem i flere. Tænk f.eks. på biblen, hvor man sædvanligvis refererer til en bestemt tekst ved at sige: 1. Mosebog, kapitel 2, vers 3. Det er en reference med tre dimensioner. Vi kunne kalde denne specielle tekst for **T(1,2,3)**. Dimensionér en variabel b, giv den to dimensioner og lad disse være 3 og 4, sådan:

DIM b(3,4)

Det giver dig så 3*4=12 indicerede variabler, som vi kunne skrive på denne måde:

b(1,1), b(1,2), b(1,3), b(1,4)
b(2,1), b(2,2), b(2,3), b(2,4)
b(3,1), b(3,2), b(3,3), b(3,4)

Det samme princip kan bruges til ethvert antal dimensioner.
Bemærk, som før nævnt, at du ikke kan have to indicerede variabelsæt med det samme navn. Det gælder også, selv om de har forskellige dimensioner.

Der findes også indicerede strengvariabler. De er dog lidt anderledes end simple strengvariabler, idet de altid er af konstant længde, og tildelingen af værdier til dem sker efter det procrusteanske princip. Navnet på et indiceret strengvariabelsæt skal være et enkelt bogstav efterfulgt af **$**. En simpel strengvariabel med det samme navn kan ikke eksistere samtidig. Du ønsker f.eks. at danne et sæt af strengvariabler bestående af 5 strenge, der hver på 10 karakterer. Så skriver du:

DIM a$(5,10)

Såfremt du nu henviser til variablen med det samme antal tal, som var til stede ved dimensioneringen (to i dette her tilfælde) får du et enkelt bogstav, men hvis du udela der det sidste tal, får du en streng af den konstante længde (10). **a$(2,7)** er således den syvende karakter i strengen **a$(2)**. Ved at bruge notationen fra kapitel 17 kunne vi også skrive dette som **a$(2)(7)**. Skriv så:

LET a$(2)="1234567890"

og

PRINT a$(2),a$(2,7)

så får du udskriften:

1234567890 7

Du kan også lave **"slicing"** på indicerede strengvariabler.

a$(2,4 TO 8)=a$(2)(4 TO 8)="45678"

Du kan godt danne indicerede strengvariabelsæt uden dimension. Skriv:

DIM a$(10)

og du vil se, at **a$** opfører sig ligesom en simpel strengvariabel, men den har altid længden 10.

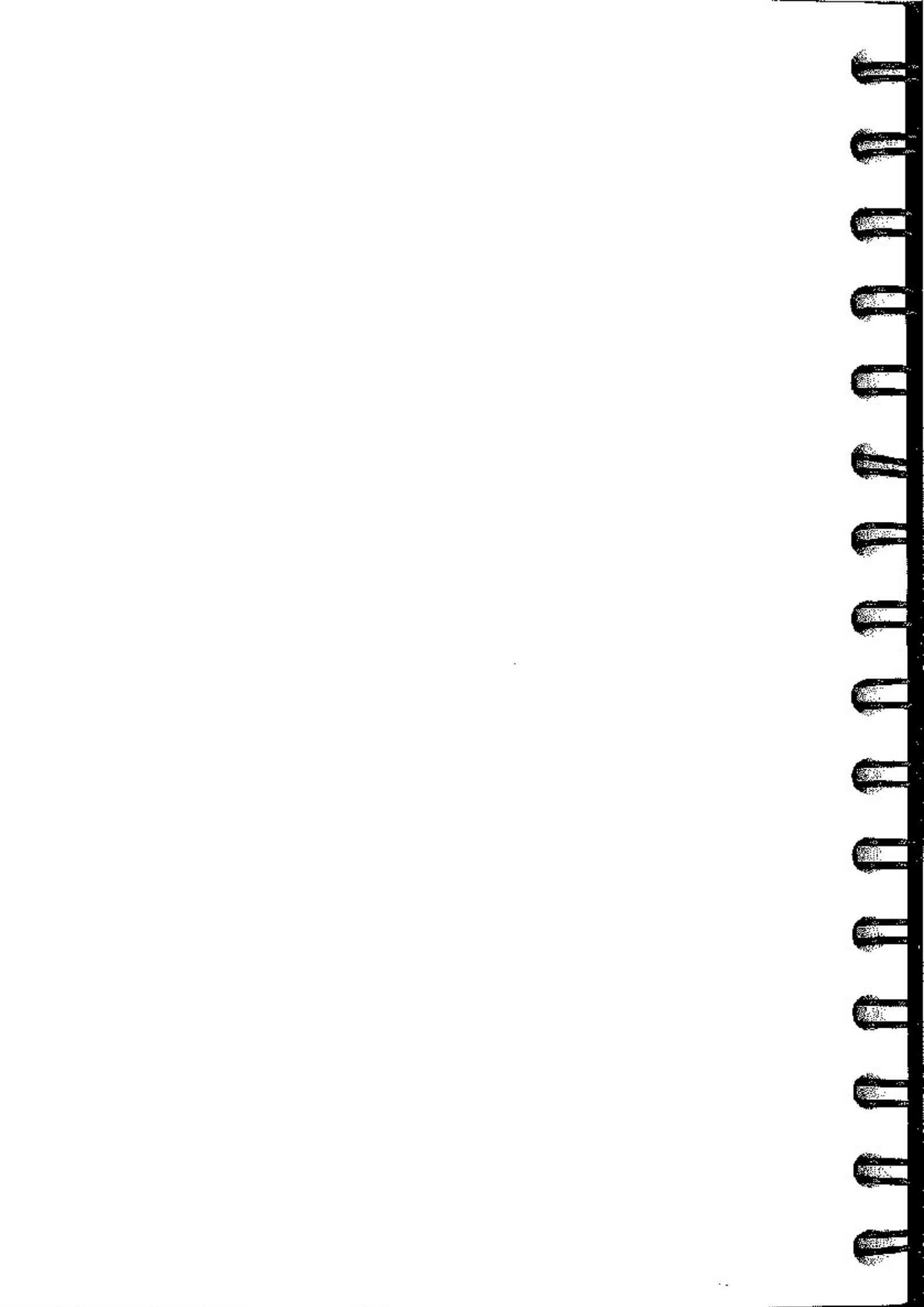
Øvelse:
1. Opstil en indiceret strengvariabel **m$** med 12 strenge, hvor **m$(n)** er navnet på den **n**'te måned, ved hjælp af **READ** og **DATA**. Forslag: Brug **DIM** kommandoen sådan:

DIM m$(12,9)

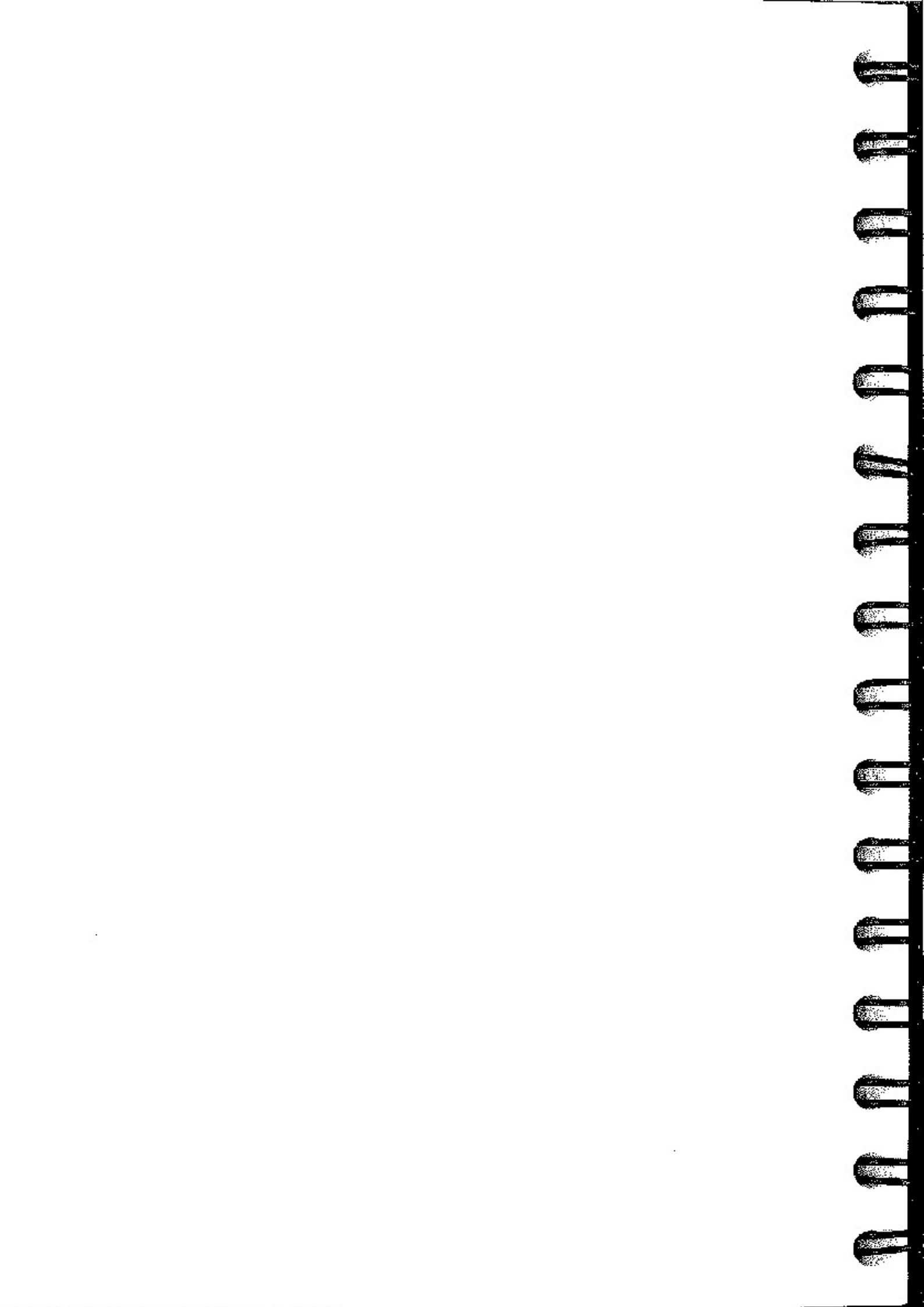
Test at det virker ved at udskrive alle **m$(n)**. (Brug en løkke). Skriv:

PRINT "Nu har vi ";m$(5);" ";"og det er den bedste aarstid"

Hvad kan du gøre ved alle mellemrummene?




22



22. Betingelser:

Vi så i kapitel 12, hvordan **IF**-kommandoen tog formen:

IF betingelse **THEN**...

Vi brugte relationstegnene (=,<,>,>=,<= **og**<>), som sammenligner to tal eller to strenge. Du kan imidlertid kombinere disse ved at anvende de logiske udtryk: **AND**, **OR** og **NOT**.

Hvis udtrykkene på begge sider af **AND** er sande, så er hele udtrykket sandt. Du kan f.eks. have en linje som denne:

IF a$="ja" AND x>0 THEN PRINT x

hvor **x** *kun* bliver skrevet, hvis a$ er lig "ja", og hvis **x** er større end **0**.

OR virker således:

1. relation **OR** 2. relation

Udtrykket er sandt, hvis blot en relationerne er sand.

NOT-relationen er sand, når relationen er falsk, og falsk, når relationen er sand.

Logiske udtryk kan altså dannes ved hjælp af relationerne og **AND**, **OR** og **NOT**, ligesom numeriske udtryk kan dannes med tal samt +,– o.s.v. Det kan endda lade sig gøre at bruge parenteser om nødvendigt.

De logiske udtryk **AND**, **OR** og **NOT** prioriteres på samme vis som de matematiske regneudtryk +,–,*,/ og ↑. De prioriteres i rækkefølgen: **OR** har lavesle prioritet, så kommer **AND**, så **NOT**, så relationerne og så de sædvanlige regneudtryk.

NOT er en rigtig funktion med et argument og et resultat, men den prioriteres lavere end de andre funktioner.

Da **NOT** har en lav prioritet, behøver dens argument (d.v.s. det, der følger efter **NOT**) ikke at være i parentes, medmindre det indeholder **AND** eller **OR** eller begge. **NOT a=b** betyder det samme som **NOT (a=b)**, (og det samme som **a<>b**). <> er det modsatte af =, eller:

a<>b er det samme som **NOT a=b**
og **NOT a<>b** er det samme som **a=b**

Overbevis dig selv om, at >= er det modsatte af <, og <= er det modsatte af >.

I relationer behøver du altså aldrig bruge **NOT**, du kan bare bruge relationens modsætning. Ligeledes er:

NOT (første udtryk **AND** andet udtryk)
det samme som
NOT (første udtryk) **OR NOT** (andet udtryk)

NOT (første udtryk **OR** andet udtryk)
er det samme udtryk som
NOT (første udtryk) **AND NOT** (andet udtryk)

Faktisk vil du efterhånden finde ud af, at **NOT** ikke er særlig nødvendig. Der kan sagtens findes på andre måder at fortælle maskinen det samme. Men - **NOT** kan i visse tilfælde gøre et program mere overskueligt. Prøv at skrive:

PRINT 1=2.1<>2

som du måske forventer, vil resultere i en syntaksfejl. Men der er faktisk ikke noget, maskinen holder mere af end *logiske udtryk og værdier.*

a: =,<,>, <=,>= **og** <> giver numeriske resultater.
Resultatet af dem er 1 for sand og Ø for falsk.
b: I sætningen:

IF betingelse **THEN** kommando

kan betingelserne faktisk være enhver numerisk værdi.
Hvis den er Ø, tæller den som falsk, og alle andre værdier tæller sande (inkl. 1). Derfor er **IF**-sætningen faktisk det samme som:

IF betingelse <> Ø **THEN** kommando

c: **AND**, **OR** og **NOT** er også matematiske relationer:

x **AND** y har værdien { x hvis y ikke er Ø (sand)
Ø hvis y er Ø (falsk)

x **OR** y har værdien { 1 hvis y ikke er Ø
x hvis y er Ø

NOT x har værdien { Ø hvis x ikke er Ø
1 hvis x er Ø

Prøv lige at læse kapitlet igennem igen i lyset af denne afsløring, så du er sikker på, at det hele virker.
Udtrykkene **x AND y**, **x OR y**, og **NOT x** vil antage værdien Ø eller 1 for falsk og sandt. Udarbejd de ti forskellige kombinationer (fire for **AND**, fire for **OR** og to for **NOT**) og undersøg, om de gør som beskrevet i kapitlet.
Prøv dette program:

```
10 INPUT a
20 INPUT b
30 PRINT (a AND a>=b)+(b AND a<b)
40 GO TO 10
```

Hver gang skrives den største værdi af variablerne. Prøv at blive helt sikker i denne logik, så du ved, at **x AND y** betyder **x** hvis **y** er sand, ellers Ø. Og lige sådan med de andre relationer.

Udtryk, som indeholder **AND** eller **OR**, kaldes "betingede udtryk".

Du kan også lave relationer mellem strenge, men kun med **AND**, hvis det er betingede relationer, du skal bruge. Prøv dette program, som tager imod de to strenge som **INPUT** og sætter dem i alfabetisk orden.

```
10 INPUT "Indtast to strenge"a$,b$
20 IF a$>b$ THEN LET c$=a$:LET a$=b$:LET b$=c$
30 PRINT a$;" ";("<"AND a$<b$)+("="AND a$=b$);" ";b$
40 GO TO 10
```

Øvelse:

1. BASIC er ikke noget almindeligt sprog. Faktisk er det sådan, at et hvilket som helst programmeringssprog er mere logisk end selv det mest logiske menneskesprog. Derfor kan det være nødvendigt at tænke sig lidt om, inden man kaster sig ud i programmering. Eks. hvorledes ville du skrive denne sætning i BASIC:

"Hvis **a** ikke er lig **b** eller **c**, så skriv **17**"

23

23. ZX Spectrums tegnsæt

De bogstaver, tal, grammatiske tegn og grafiske symboler, som kan optræde i strenge, kaldes tegn eller *karakterer*. De udgør ZX Spectrums *karakter-sæt*. De fleste af disse karakterer er enkelttegn, men nogle få er hele "ord", f.eks. **PRINT, STOP, GO TO** o.s.v.

Der er ialt 256 karakterer, og hver af dem repræsenteres af en talkode mellem **0** og 255. Du kan finde en liste over dem i Appendiks A. Hvis man vil skifte fra kode til karakter eller omvendt, kan man bruge funktionerne **CODE** og **CHR$**.

CODE bruges i forbindelse med strenge og giver som resultat talkoden for den første karakter i strengen (eller **0**, hvis strengen er tom)

CHR$ bruges i forbindelse med et tal og giver som resultat en streng med en enkelt karakter, hvis kode var givet i tallet.

Dette program udskriver hele karaktersættet:

```
10 FOR a=32 TO 255: PRINT CHR$ a;: NEXT a
```

Øverst kan du se et mellemrum, 15 karakterer **" ! %** o.s.v. de ti tal, yderligere syv karakterer, de store bogstaver, seks karakterer til, de små bogstaver og endnu fire karakterer. De er alle (undtagen **£** og ©) ASCII-karakterer. ASCII står for: American Standard Codes for Information Interchange. ASCII bruger talkoder for karaktererne, og det er disse, ZX Spectrum også bruger.

Resten af karaktererne er ikke ASCII-karakterer, men specielle for ZX Spectrum. De første af disse karakterer er et mellemrum og 15 sort-hvide tegn. De kaldes "grafik"-symboler og kan bruges til at tegne forskellige billeder og symboler.

Du kan indtaste dem fra tastaturet ved at trykke **GRAPHICS** (**CAPS SHIFT** og **9**), så markøren skifter til en **G**-markør. Når du nu trykker på tasterne fra 1 til 8, vil datamaten skrive symbolerne på tasterne. Hvis du trykker på en af **SHIFT**-tasterne, skrives de inverse tegn – d.v.s., hvad der før var hvidt, er nu sort og omvendt. **9**-tasten ændrer til **L**-markør, uanset om du trykker på skiftetasterne, og **0**-tasten virker som slettetast (**DELETE**).

Her er de 16 grafik-symboler:

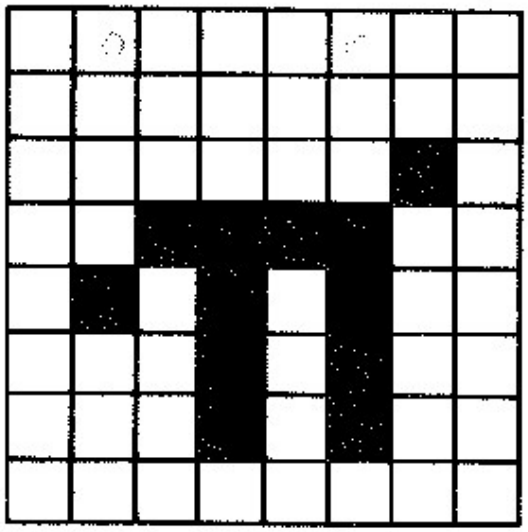
Symbol	*Kode*	*Fås ved*
	128	G 8
	129	G 1
	130	G 2
	131	G 3
	132	G 4
	133	G 5
	134	G 6
	135	G 7

Symbol	*Kode*	*Fås ved*
	143	G Shift + 8
	142	G Shift + 1
	141	G Shift + 2
	140	G Shift + 3
	139	G Shift + 4
	138	G Shift + 5
	137	G Shift + 6
	136	G Shift + 7

Efter grafiksymbolerne kommer der igen en række store bogstaver i alfabetisk orden fra A til U. Disse karakterer kan du selv lave om på, men når maskinen tændes, de bogstaverne fra A til U, indtil du ændrer på dem. De kaldes for "brugerdefinérbare" grafiktegn. Du kan indtaste dem fra tastaturet ved at omstille maskinen til grafik (**G**-

markør) og bruge tasterne fra A til U. Hvis du selv vil definere karakterer, så prøv følgende fremgangsmåde – den definerer en karakter, som viser tegnet PI (π):
1. Udarbejd, hvordan karakteren ser ud. Hver karakter er opdelt i 8×8 punkter, hvor hvert punkt kan vise enten "paper"-farven eller "ink"-farven, (se i kapitel 7).

Du opstiller et diagram, som vist her, hvor de sorte punkter står for "ink"-farven:



Som du kan se, har vi lavet en 1-punkt's margin rundt om tegnet, fordi alle de andre bogstaver i datamaten ser sådan ud (undtagen nogle få af de små bogstaver).
2. Bestem dernæst, hvilken tast, du vil bruge til grafik-tegnet. I dette tilfælde vælger vi P, så hvis du taster P, når maskinen er omstillet til grafik, får du π.
3. Nu skal vi til at lave selve grafiktegnet. Hvert grafiktegn opbevares som otte tal, ét for hver række. Du kan skrive hvert af tallene ved at anvende **BIN** (**B**-tasten) efterfulgt af otte **0**'er eller 1'ere,– **0** for "paper" og 1 for "ink".

Så de otte tal for π er:

```
BIN 00000000
BIN 00000000
BIN 00000010
BIN 00111100
BIN 01010100
BIN 00010100
BIN 00010100
BIN 00000000
```

(Hvis du kender til binære tal, så vil det være en hjælp at vide, at **BIN** bruges til at skrive et tal binært).

De otte tal gemmes i lageret på otte pladser, hver med sin adresse. Adressen på den første "byte", eller gruppe på otte tal, er **USR "P"**, (P fordi vi valgte dette bogstav i pkt. 2). Den næste adresse er **USR "P"+1** o.s.v.

USR er her en funktion, som omdanner et strengargument til adresser i de første "byte" af hukommelsen. Strengargumentet skal være en enkelt karakter, som kan være et selv-defineret grafiktegn eller det tilhørende bgstav (stort eller lille). **USR** bruges på en anden måde, når argumentet er et tal.

Hvis du ikke forstår dette, så vil følgende program hjælpe dig:

```
10 FOR n=0 TO 7
20 INPUT raekke: POKE USR "P"+n,RAEKKE
30 NEXT n
```

Programmet vil spørge efter **INPUT** otte gange, hvor du så hver gang skal svare med **BIN** efterfulgt af et binært tal.

POKE-kommandoen lagrer et tal direkte i hukommelsen, uden om den mekanisme, som normalt bruges af BASICen. Det modsatte af **POKE** er **PEEK**, og denne kommando giver os mulighed for at se indholdet i en bestemt adresse uden at ændre i den. Dette bliver nøjere beskrevet senere.

Efter de "brugerdefinérbare" grafiktegn, kommer ZX Spectrums kommando-ord.

Vi udskrev ikke de første 32 karakterer (koderne fra 0 til 31). Disse er kontrolkarakterer. De producerer ikke noget på "skrift", men nogle har stor indvirkning på skærmen. Andre bruges til at kontrollere andre ting end skærmen, og skærmen skriver så ? for at vise, at den ikke forstår dem. Disse er beskrevet i Appendiks A. Skærmen bruger de tre karakterer med koderne 6, 8 og 13, hvoraf **CHR$ 8** er den nyttigste.

CHR$ 6 skriver mellemrum på nøjagtig samme måde, som et komma laver mellemrum i en **PRINT**-sætning. Eks.

```
PRINT 1;CHR$6;2
```

er det samme som

```
PRINT 1,2
```

Dette er dog ikke en særlig smart måde at bruge det på. En mere praktisk anvendelig måde er f.eks.:

```
LET a$= "1"+CHR$ 6+"2"
PRINT a$
```

CHR$ 8 er "tilbageflytning". Den flytter printpositionen en plads baglæns på linjen. Prøv.

```
PRINT "1234";CHR$8;"5"
```

som skriver **1235**.

CHR$ 13 er "newline" (på dansk: ny linje). Den flytter printpositionen til begyndelsen af næste linje.

Skærmen bruger også karaktererne med koderne 16 og 23. Disse bliver forklaret i kapitel 24 og 25. Alle kontrolkaraktererne står i Appendiks A.

Ved at bruge koderne for karaktererne kan vi udvide begrebet "alfabetisk orden" til alle ZX Spectrums 256 karakterer. I stedet for, som normalt, kun at se på alfabetets bogstaver, bruger vi koderne for karaktererne i den rækkefølge, de er opstillet.

(Bemærk, at de små bogstaver kommer efter de store bogstaver, så "a" kommer efter "Z".

Følgende strenge står i "ZX Spectrums alfabetiske orden":

```
CHR$ 3+"ZOOLOGISK HAVE"
CHR$ 8+"OLFERT SPISER"
"  AAAARGH"
"100"
"129.95 inkl Moms"
"PRINT"
"zoo"
"zoologi"
```

Her er reglerne for, hvilken af to strenge der kommer først: Først sammenlignes karaktererne. Hvis de er forskellige, så har en af dem en talkode mindre end den anden, og den med mindst kodetal kommer først i rækkefølgen. Hvis de første karakterer er ens, sammenlignes de næste karakterer. Hvis en af strengene er kortere end den anden, så er det den korteste streng, der kommer først. Først til sidst kommer muligheden for, at strengene er ens.
Relationerne: =,<,>,<=,>= og <> bruges på strenge på samme måde som på tal: < betyder "kommer før"- og > betyder "kommer efter".

Så:

```
"abcd"<"abcdef"
"abcdef">"abcd"
```

er begge sande

<=, og >= virker på samme måde, som de virker på tal, så:

```
"ENS STRENGE" <= "ENS STRENGE"
```

er sandt, men:

```
"ENS STRENGE"<"ENS STRENGE"
```

er falsk.

Eksperimentér med dette ved at bruge følgende program, som modtager to strenge og stiller dem op i tal-rækkefølge:

```
10 INPUT "Indtast to strenge:",a$,b$
20 IF a$>b$ THEN LET c$=a$:LET a$=b$:LET b$=c$
30 PRINT a$;" ";
40 IF a$<b$ THEN PRINT "<";:GOTO 60
50 PRINT "="
60 PRINT " ";b$
70 GO TO 10
```

I linje 20 anvender vi en ny variabel **c$** for at kunne bytte om på **a$** og **b$**,

```
LET a$=b$: LET b$=a$
```

vil ikke have den ønskede effekt.

Følgende program anvender "brugerdefinérbar" grafik til at lave spillebrikker til skak:

```
  5 LET b=BIN 01111100:LET c=BIN 00111000:
    LET d=BIN 00010000
 10 FOR n=1 TO 6:READ p$:REM 6 brikker
 20 FOR f=0 TO 7:REM indlæser punkter til 8"byte"
 30 READ a:POKE USR p$+f,a
 40 NEXT f
 50 NEXT n
100 REM "løber"
110 DATA "b",0,d,BIN 00101000, BIN 01000100
120 DATA BIN 01101100,c,b,0
130 REM"konge"
140 DATA "k",0,d,c,d
150 DATA c, BIN 01000100,c,0
160 REM "tårn"
170 DATA "r",0,BIN 01010100,b,c
180 DATA c,b,b,0
190 REM "dronning"
200 DATA "q",0,BIN 01010100, BIN 00101000,d
210 DATA BIN 01101100,b,b,0
220 REM "bonde"
230 DATA "p",0,0,d,c
240 DATA c,d,b,0
250 REM "springer"
260 DATA "n",0,d,c, BIN 01111000
270 DATA BIN 00011000,c,b,0
```

(Du kan bruge 0 istedet for **BIN 00000000**).

Når du har kørt programmet, kan du se skakbrikkerne ved at omstille maskinen til grafik.

Her er de taster, som er anvendt:

P for bonde
R for tårn
N for springer
B for løber
K for konge
Q for dronning

ØVELSER

1. Forestil dig, at et tastsymbol optager en plads, der kan deles i fire lige store dele. Hvis derfor hver fjerdedel kan være enten sort eller hvid, er der 2*2*2*2=16 muligheder eller kombinationer af sorte og hvide punkter.
Find dem alle i karaktersættet.

2. Kør dette program:

```
10 INPUT a
20 PRINT CHR$ a;
30 GO TO 10
```

Hvis du eksperimenterer lidt med dette program, opdager du, at **CHR$** *runder* a af til det *nærmeste* hele tal. Hvis a *ikke* befinder sig i området 0 til 255, stopper programmet med rapport: **B integer out of range**.

3. Hvilken af disse to strenge "kommer først"?

```
"ZX Spectrum"
"zx spectrum"
```

24

24. Mere om PRINT og INPUT

Du har nu mange gange set, hvordan **PRINT**-kommandoen bruges. En **PRINT**-sætning arbejder med et eller flere udtryk, adskilt med semikolon eller komma. Disse udtryk kaldes **PRINT**-elementer, og vi kalder tegnene komma og semikolon for "skilletegn". Et **PRINT**-element behøver ikke nødvendigvis at indeholde noget. **PRINT**, efterfulgt af to kommaer, viser dette.

Der findes endnu to slags **PRINT**, der bruges til at fortælle datamaten ikke, *hvad* den skal skrive, men *hvor* den skal skrive det. Disse to er **PRINT**-formaterne:

PRINT AT linje, kolonne

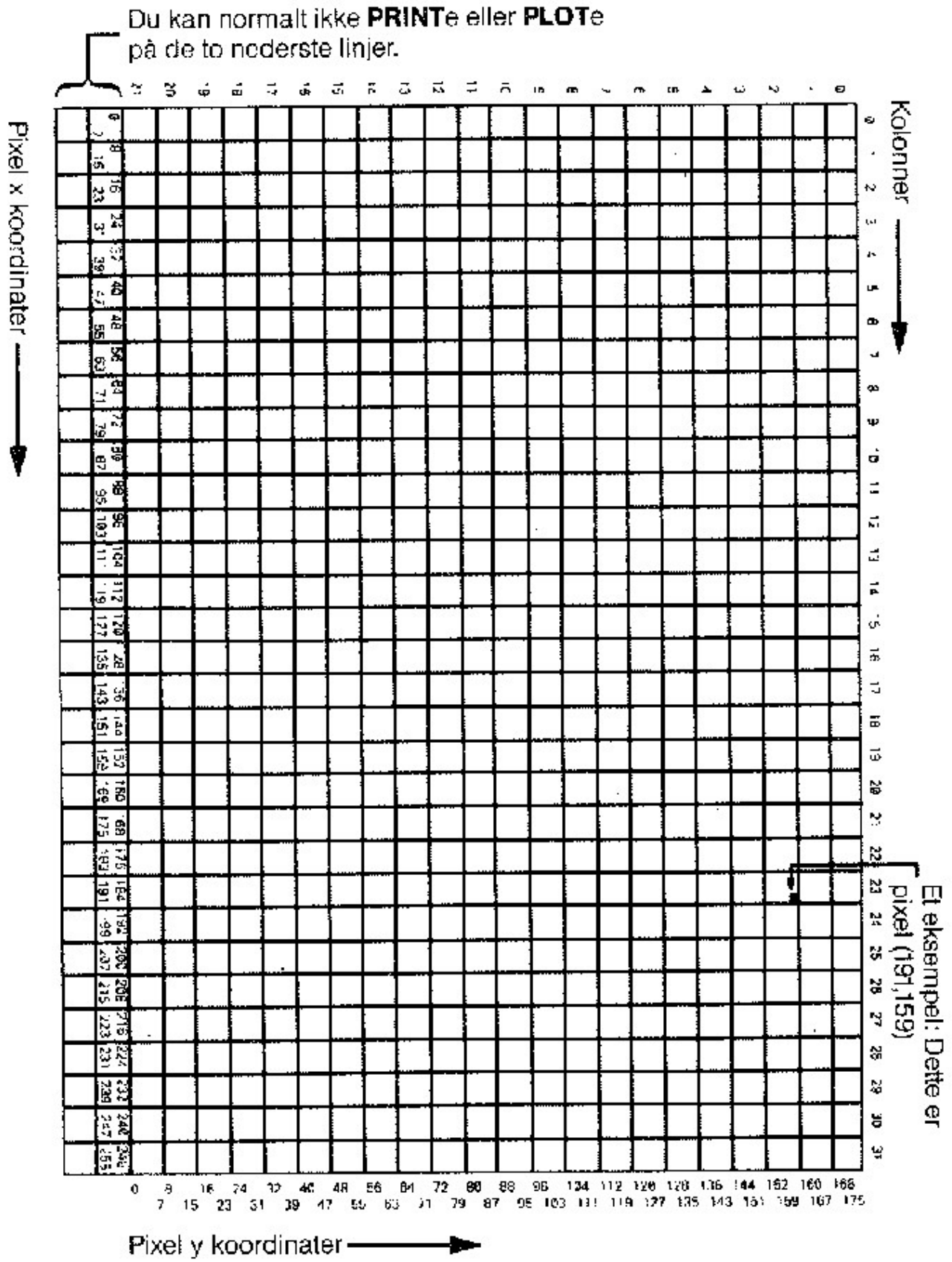
og

PRINT TAB kolonne

Den første flytter PRINT-positionen (d.v.s. der, hvor det næste tegn skrives) til den angivne linje og kolonne. Linjerne er nummeret fra 0 (øverst) til 21, og kolonnerne fra 0 (yderst til venstre) til 31.
Hvis du skriver:

PRINT AT 11,16;"*"

får du anbragt en stjerne midt på skærmen.


Du kan normalt ikke **PRINT**e eller **PLOT**e
på de to nederste linjer.
Kolonner
Pixel x koordinater
Et eksempel: Dette er
pixel (191,159)
Pixel y koordinater

Den anden – **PRINT TAB** – flytter en **PRINT**-position til den angivne kolonne. Når det ønskede er udskrevet, bliver maskinen på samme linje eller, hvis den næste ordre medfører et flyt mod venstre, på næste linje. Det skal bemærkes, at kolonnenummeret styres af modulus (MOD 32) (d.v.s., maskinen dividerer med 32 og beholder resten), således at **TAB 33** er det samme som **TAB 1**.

Prøv at skrive:

```
PRINT TAB 30;1;TAB 12;"Indhold";AT 3,1;"Kapitel";TAB 24;"Side"
```

Det viser, hvordan du f.eks. kan opstille en oversigt over en bog.
Prøv at køre dette:

```
10 FOR n=0 TO 20
20 PRINT TAB 8*n;n;
30 NEXT n
```

Programmet viser, at TABs tal bliver reduceret MOD 32. Hvis du ønsker at få et lidt mere elegant eksempel, så udskift 8 i linje 20 med 6.
Et par små vink:

1. Det er bedst at afslutte bestandelene i **PRINT**-sætningerne med semikolon som ovenfor. Du kan bruge komma eller ingenting, men det betyder, at den omhyggeligt udvalgte **PRINT**-position straks flytter igen. Det er ikke ret nyttigt.
2. Du kan ikke skrive på de nederste to linjer af skærmen (22 og 23), fordi de er reserveret til kommandoer.
3. Du kan bruge **AT** til at skrive noget i en position, hvor der allerede står noget. Det gamle vil bare blive slettet.

Der er en kommando mere, som hører til **PRINT**-sætningerne. Det er **CLS**. Den visker skærmen ren på samme måde som **CLEAR** og **RUN**.

Når maskinen har fyldt skærmen (d.v.s. er nået til den nederste linje), begynder den at flytte linjerne opad som en skrivemaskine. Du kan se dette, hvis du skriver:

```
CLS:FOR n=1 TO 22: PRINT n: NEXT n
```

og derefter skriver:

```
PRINT 99
```

et par gange.

Hvis maskinen skal skrive mange linjer på skærmen, så rykker den ikke linjerne op, uden at du får chancne for at se dem. Du kan se dette, hvis du skriver:

```
CLS: FOR n=1 TO 100:PRINT n:NEXT n
```

Når maskinen har fyldt skærmen, vil den stoppe og skrive: **scroll?** forneden på skærmen. Du kan så se på de første 22 linjer. Når du er færdig med dem, så tast **y** (for "ja") og datamaten vil så fylde skærmen med 22 nye linjer.

Du kan bruge **INPUT**-kommandoen til meget mere, end vi har fortalt indtil nu. Du har dog allerede set **INPUT**-kommandoer som f.eks.:

```
INPUT "Hvor gammel er du?", aar
```

hvor maskinen skriver: **"Hvor gammel er du"** forneden på skærmen, og du indtaster din alder.

På en måde er **INPUT**sætninger opbygget som **PRINT**-sætninger med komma og semikolon som "skilletegn". Der er dog visse vigtige forskelle.

For det første indeholder **INPUT**-sætninger altid variabler, i vores tilfælde **"aar"**, som skal indtastes fra tastaturet. Reglen er, at hvis variablen starter med et bogstav, så skal **INPUT**-svaret være en tal-værdi.

For det andet kan du ikke umiddelbart skrive værdien af en variabel som en del af "teksten" i **INPUT**-sætningen. Du kan dog komme om ved dette ved at skrive parenteser om variablen. Ethvert udtryk, som starter med et bogstav, skal være omsluttet af parenteser, hvis det skal i en **INPUT** "tekst".

Her er et eksempel, som illustrerer, hvorledes det virker:

```
LET min alder=INT(RND*100):INPUT("Jeg er";min alder;
".");"Hvor gammel er du?",din alder
```

"min alder" er omsluttet af parenteser, så værdien bliver udskrevet. **"din alder"** er ikke omsluttet af parenteser, så her skal du indtaste en værdi.

INPUT-sætningen skriver altid i den nederste del af skærmen, der fungerer noget uafhængigt af den øverste del af skærmen. Specielt gælder det, at dens linjer er nummereret i forhold til den øverste linje i den nederste halvdel, selv om denne er rykket op i selve skærmen (hvilket den gør, hvis du indtaster store mængder **INPUT**-data.

For at se, hvordan **AT** virker i en **INPUT**-sætning, så prøv at køre:

```
10 INPUT "Dette er linie 1.",a$;AT 0,0;"Dette er linie 0.",a$;
AT 2,0;"Dette er linie 2.",a$;AT 1,0; "Dette er stadig linie 1."a$
```

(Tryk **ENTER**, hver gang programmet stopper). Når der skrives: **"Dette er linie 2"**, så vil den nederste del af skærmen flytte op for at gøre plads for denne, men nummeringen flytter også op, så linjerne med tekst beholder de samme numre.

Prøv dette:

```
10 FOR n=0 TO 19: PRINT AT n,0;n;:NEXT n
20 INPUT AT 0,0;a$;AT 1,0;a$;AT 2,0;a$;AT 3,0;a$; AT 4,0;a$;
   AT 5,0;a$;
```

Mens den nederste del af skærmen rykker op, forbliver den øverste del uforstyrret, indtil den nederste del forsøger at skrive på samme linje som **PRINT**-positionen. Så begynder den øvre del at **scroll**'e for at forhindre dette.

En detalje, som vi endnu ikke har set, er **LINE-INPUT**, som tager imod streng-**INPUT** på en anden måde. Hvis du skriver **LINE** før navnet på strengvariablen i en **INPUT**-sætning, som f.eks.:

```
INPUT LINE a$
```

vil datamaten ikke skrive anførselstegnene som sædvanligt, selv om den lader, som om de er der. Så hvis du skriver:

kat

som **INPUT**-data, så får variablen **a$** værdien **kat**. Da anførselstegnene ikke vises på skærmen, kan du ikke slette dem og indtaste andre slags udtryk som **INPUT**-data. Husk at du ikke kan bruge **LINE** til numeriske variabler.

Kontrolkaraktererne **CHR$ 22** og **CHR$ 23** virker på samme måde som **AT** og **TAB**. De er ikke meget bevendt som kontrolkarakterer, for når en af dem skrives på skærmen, skal denne efterfølges af endnu to karakterer, som ikke har deres oprindelige virkning: de bliver af maskinen opfattet som tal (deres koder) til bestemmelse af linje og kolonne (for **AT**) eller tab-position (for **TAB**). Du vil næsten altid finde det lettere at bruge **AT** og **TAB** på normal vis, men under visse specielle omstændigheder vil du muligvis finde kontrolkaraktererne brugbare. Kontrolkarakteren for **AT** er **CHR$ 22**. Den første karakter efter denne bestemmer linjen og den anden kolonnen.

F.eks.:

PRINT CHR$ 22+CHR$ 1+CHR$ c;

er præcis det samme som

PRINT AT 1,c;

Sådan er det, når **CHR$ 22** står forrest, selv om **CHR$ 1** og **CHR$ c** normalt har en anden betydning (f.eks. hvis **c=13**). Kontrolkarakteren for **TAB** er **CHR$ 23**, og de to efterfølgende karakterer bruges til at give et tal mellem 0 og 65535, som bestemmer **TAB**-positionen.

PRINT CHR$ 23+CHR$ a+CHR$ b;

virker på samme måde som

PRINT TAB a+256*b;

Du kan anvende **POKE** for at forhindre maskinen i at spørge **'scroll?'** Det sker ved at skrive

POKE 23692,255

Efter dette vil den **scroll**'e skærmen 255 gange, før den igen skriver **'scroll?'** Prøv

```
10 for n=0 TO 10000
20 PRINT n: POKE 23692,255
30 NEXT n
```

og se alting suse op over skærmen.

Øvelse
1. Prøv dette multiplikationsprogram på nogle børn:

```
10 LET m$=""
20 LET a=INT(RND*12)+1:LET b=INT(RND*12)+1
30 INPUT (m$)""hvad er";(a);"*";(b);"?";c
100 IF c=a*b THEN LET m$="rigtigt": GO TO 20
110 LET m$="forkert, prøv igen": GO TO 30
```

Hvis de er meget kvikke, vil de kunne gætte, at de ikke behøver selv at gøre udregningerne. Hvis datamaten f.eks. spørger efter svaret **2*3**, så behøver de kun at indtaste **2*3**.

En måde at løse problemet på er at lade dem indtaste strenge i stedet for tal. Erstat **c** i linje 30 med **c$** og **c** i linje 100 med **VAL c$** og indsæt denne linje:

```
40 IF c$<>STR$ VAL c$ THEN LET m$="Indtast det
   som et tal":GO TO
```

Dette vil forvirre dem. Efter et par dage vil en af dem måske opdage, at de kan snyde maskinen ved at slette anførselstegnene og skrive **STR$ (2*3)**. - *MEN* hvis du erstatter **c$** i linje 30 med **LINE c$**, så kan de ikke snyde.

25

25. Kulør på tilværelsen

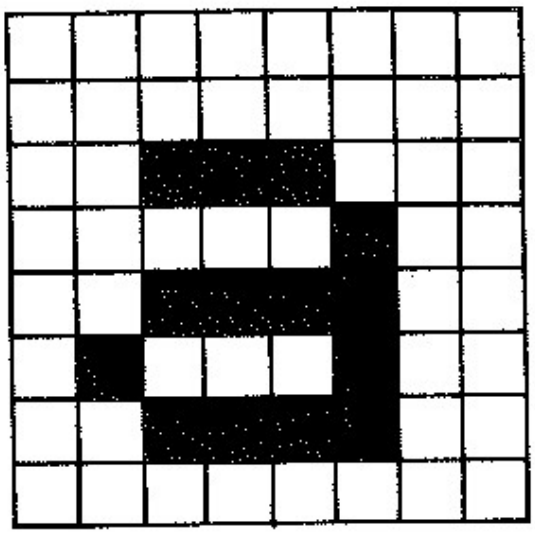
Kør dette program:

```
10 FOR m=0 TO 1: BRIGHT m
20 FOR n=1 TO 10
30 FOR c=0 TO 7
40 PAPER c:PRINT"    ";:REM fire farvede mellemrum
50 NEXT c: NEXT n: NEXT m
60 FOR m=0 TO 1: BRIGHT m: PAPER 7
70 FOR c=0 TO 3
80 INK c: PRINT c;"  ";
90 NEXT c: PAPER 0
100 FOR c=4 TO 7
110 INK c: PRINT c;"  ";
120 NEXT c: NEXT m
130 PAPER 7:INK 0 :BRIGHT 0
```

Dette viser de otte farver (inkl. hvid og sort) og de to lysstyrker, som ZX Spectrum kan producere på et farvefjernsyn. (Hvis du har et sort-hvidt tv, så vil du i stedet se forskellige nuancer af gråt). Her er en liste over farverne med deres tilhørende talkoder:

0-sort
1-blå
2-rød
3-purpur
4-grøn
5-lyseblå
6-gul
7-hvid

På et sort-hvidt tv angiver tallene rækkefølgen i lysstyrke.
For at kunne bruge farverne rigtigt er du nødt til at vide noget mere om, hvorledes billedet er opbygget.
Billedet eller skærmen er opdelt i 768 (24 linjer af 32) positioner, hvor karakterer kan skrives.



Hver karakter skrives i et kvadrat med 8×8 punkter, som f.eks. **a.** Dette minder dig måske om 'brugerdefinérbar' grafik i kapitel 23, hvor Ø'er stod for hvide punkter, og 1'ere stod for sorte punkter.

Til hver karakterposition hører der to farver, nemlig 'ink'-farven eller forgrundsfarven, som i vort tilfælde er farven på de sorte punkter, og 'paper'-farven eller baggrundsfarven, som er farven på de hvide punkter. Når maskinen tændes, har hver karakterposition sort 'ink' og hvid 'paper', så der skrives med sort på hvidt.

Hver karakter har to lysstyrker (normal eller ekstra lys), og ligeledes kan hver karakterposition blinke. Datamaten får hver enkelt karakterposition til at blinke ved at skifte mellem 'ink' og 'paper' farverne.

De forskellige styremuligheder af hver karakterposition bestemmes af talkoder.

Der gælder følgende:

1. Et kvadrat på 8×8 punkter af Ø'er og 1'ere danner en karakter, hvor Ø står for 'paper' og 1 for 'ink'.

2. 'ink' og 'paper' -farverne har talkoder mellem Ø og 7.

3. Lysstyrkekontrollen (**BRIGHT**), Ø for normal og 1 for ekstra lys.

4. 'Blinkekontrollen' (**FLASH**), Ø for normal og 1 for blink.

Bemærk, at du kan ikke have mere end to farver inden for et kvadrat med 64 punkter (8×8). Disse to farver er 'ink' og 'paper' -farverne. Det samme gælder blink og lysstyrke. Det er nødvendigt at operere med 64 punkter hele tiden. Farverne, lysstyrkekontrollen, blinkekontrollen på en given position, kaldes for dens attributter.

Når du skriver noget på skærmen, ændrer du mønstret af punkter i den pågældende position, og du har mulighed for at ændre dens attributter.

Ved starten vil du ikke bemærke noget af alt dette, da der som sagt skrives med sort på hvidt, (og normal lysstyrke og ingen blink), men du kan ændre dette ved at anvende kommandoerne: **INK, PAPER, BRIGHT** og **FLASH**.

Prøv:

```
PAPER 5
```

og skriv et eller andet. Det vil optræde på lyseblå baggrund, fordi vi har bestemt 'paper'-farven til dette (kode 5).

De andre kommandoer virker på samme måde, hvis du bruger de tilhørende talkoder:

PAPER, tal mellem 0 og 7
INK, tal mellem 0 og 7
BRIGHT 0 eller 1
FLASH 0 eller 1
} Tænk på 0 som slukket og 1 som tændt

Prøv de forskellige farve-kommandoer. Du skulle nu være i stand til at kunne se, hvordan programmet i starten virker (husk at et mellemrum er en karakter med samme 'ink' og 'paper'-farve).
Der findes flere tal, som du kan bruge i forbindelse med de omtalte kommandoer, men de har en mindre direkte virkning.

8 kan bruges i alle fire kommandoer og betyder så 'gennemsigtig'. D.v.s. at den oprindelige farve 'skinner' igennem (også lysstyrke og blink vil 'skinne' igennem).

Skriv f.eks.:

```
PAPER 8
```

Ingen karakterposition kan have en 'paper'-farve med kodetal 8, fordi denne farve ikke findes. Der sker det, at når en karakterposition skrives, så forbliver 'paper'-farven den samme som før. **INK 8, BRIGHT 8** og **FLASH 8** virker på samme måde.

Kodetallet 9 kan kun bruges sammen med **PAPER** og **INK** og betyder 'kontrast'. Farven ('ink' eller 'paper'), som du bruger det med, danner 'kontrast' til de andre farver. Den bliver hvid, hvis de andre er mørke (sort, blå, rød eller purpur) og sort, hvis de andre er lyse (grøn, lyseblå, gul eller hvid).

Prøv dette ved at skrive:

```
INK 9: FOR c=0 TO 7: PAPER c: PRINT c: NEXT c
```

Prøv også at køre det første program med farvestriberne, som giver et bedre indtryk, hvis du derefter skriver dette:

```
INK 9: PAPER 8: PRINT AT 0,0;: FOR n=1 TO 1000:
PRINT n;: NEXT n
```

Her danner 'ink'-farven altid kontrast til den gamle 'paper'-farve i hver position.

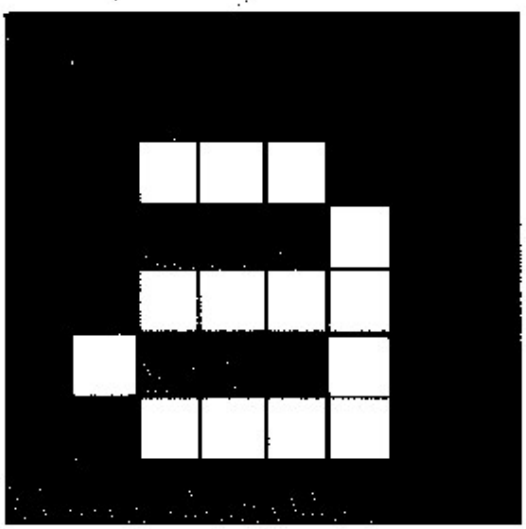
Farve-tv bygger på den kendsgerning, at mennesket i realiteten kun kan se tre farver: blå, rød og grøn. De andre farver er kombinationer af disse. F.eks. er purpur en kombination af blå og rød, det er derfor purpur har koden 3, som er summen af koderne for blå og rød.

For at se, hvordan de otte farver hører sammen, så prøv at forestille dig tre firkantede spotlys, med farverne blå, rød og grøn, som lyser forskudt for hinanden på et hvidt lærred i mørke. Dér, hvor de overlapper hinanden, vil du kunne se de forskellige farvekombinationer, som i dette program (bemærk, at mellemrummene med 'ink'-farve fås ved at bruge **SHIFT** og 8 i grafik-tilstand).

```
 10 BORDER 0: PAPER 0: INK 7: CLS
 20 FOR a=1 TO 6
 30 PRINT TAB 6; INK 2;"                  ":
    REM 18 'ink'-kvadrater
 40 NEXT a
 50 LET datalinje=200
 60 GO SUB 1000
 70 LET datalinje=210
 80 GO SUB 1000
 90 STOP
200 DATA 2,3,7,5,4
210 DATA 2,2,6,4,4
1000 FOR a=1 TO 6
1010 RESTORE datalinje
1020 FOR b=1 TO 5
1030 READ c: PRINT INK c;"      ";: REM 6 "ink"-kvadrater
1040 NEXT b: PRINT : NEXT a
1050 RETURN
```

En funktion, der kaldes **ATTR**, kan finde ud af, hvad attributterne er på en given position. Det er en meget kompliceret funktion, så den beskrives først til sidst i dette kapitel.

Der er to andre kommandoer, nemlig **INVERSE** og **OVER**. De styrer punktmønstret, som skrives på skærmen. De bruger tallene 0 for slukket og 1 for tændt, på samme måde som for **FLASH** og **BRIGHT**, og det er de eneste muligheder. Hvis du bruger **INVERSE 1**, så vil punktmønstret blive skrevet 'inverst' eller omvendt: d.v.s. at 'paper' punkterne erstattes af 'ink'-punkter og omvendt. Således vil et **a** blive skrevet således:



Hvis vi har sort 'ink'-farve og hvid 'paper'-farve, så vil **a** fremtræde hvid på sort – men vi har stadig sort 'ink'-farve og hvid 'paper'-farve på **a**'s karakterposition. Det er punkterne, der er ændret.

Kommandoen:

OVER 1

giver mulighed for at skrive 'oveni' en tidligere karakter. Når der på normal vis skrives noget i en karakterposition, så slettes det, der var der tidligere, men ved brug af **OVER** vil den nye karakter simpelt hen blive lagt oven på den gamle (men se i øvelse 3). Dette kan være specielt værdifuldt ved sammensatte karakterer som f.eks. bogstaver med accent som i dette program, der skriver et tysk 'ö' med 'umlaut'.

(Tast **NEW** først).

```
10 OVER 1
20 FOR n=1 TO 32
30 PRINT "o"; CHR$ 8;"""";
40 NEXT n
```

(Læg mærke til kontrol karakteren **CHR$ 8**, som rykker en plads baglæns).

Der er en anden måde at bruge **INK, PAPER** o.s.v., som du sikkert vil finde mere nyttig end direkte kommandoer. Du kan anvende dem sammen med **PRINT**-kommandoer (efterfulgt af ;), og de vil så gøre nøjagtig det samme, som de ville have gjort, hvis de var blevet anvendt som kommandoer alene, bortset fra, at deres effekt kun er midlertidig. Den varer så længe som den **PRINT**-kommando, der rummer dem.

Så hvis du skriver:

PRINT PAPER 6;"x";: PRINT "y"

er det kun **x**, som vil blive skrevet på gul baggrund.

INK og de øvrige har brugt som ordrer ingen indvirkning på farverne i den nederste del af skærmen, hvor kommandoer og **INPUT**-data indtastes. Den nederste del af skærmen anvender kantfarven som 'paper'-farve og kode 9 som kontrast til 'ink'-farven, der ikke blinker eller har ekstra lysstyrke. Du kan ændre kantfarven til en af de otte farver (ikke 8 eller 9) ved at bruge kommandoen:

BORDER farve

Når du indtaster **INPUT**-data, så følges reglerne om 'ink'-farvens kontrast til 'paper'-farven, men du kan ændre farven på **INPUT**-tasten ved at bruge **INK** og **PAPER** (o.s.v.) i **INPUT**-kommandoen på samme måde som ved **PRINT**-kommandoer.

Effekten varer også her kun til afslutningen af kommandoen, eller indtil nogle **INPUT**-data indtastes.

Prøv:

INPUT FLASH 1; INK 1;"Hvad er dit tal ?";n

Der findes en anden måde at ændre farverne ved at bruge kontrolkarakterer – på tilsvarende måde som kontrolkaraktererne for **AT** og **TAB** i kapitel 24.

CHR$ 16 svarer til **INK**
CHR$ 17 svarer til **PAPER**
CHR$ 18 svarer til **FLASH**
CHR$ 19 svarer til **BRIGHT**
CHR$ 20 svarer til **INVERSE**
CHR$ 21 svarer til **OVER**

Disse skal efterfølges af en karakter, som bestemmer farven (kodetallet), så f.eks.:

PRINT CHR$ 16+CHR$ 9; . . .

har samme virkning som

PRINT INK 9; . . .

Normalt vil du sikkert ikke få brug for kontrolkaraktererne, da du lige så godt kan bruge farverne direkte. Imidlertid kan du med fordel anvende dem i programmer for at gøre dem mere overskuelige. Ved at anvende kontrolkaraktererne kan du liste programlinjerne med forskellige farver, så du kan opdele et program i farve-blokke, eller du kan bruge dem til at sætte lidt 'kulør' på dit program.

Du må sætte dem ind efter linjenummeret, ellers mister du dem.

For at få disse farver med i programmet skal du indtaste dem fra tastaturet på taltasterne med **E**-markør.

Tallene **0** til 7 bestemmer den tilsvarende farve – ink, hvis **CAPS SHIFT** trykkes, og paper, hvis ikke.

Mere præcist: Hvis du har **E**-markør og taster et tal, f.eks. **6** (det skal være tal mellem **0** og 7, ikke 8 og 9) for gul, så indsættes to karakterer: først **CHR$ 17** for paper og så **CHR$ 6**, som betyder: 'farven skal være gul'. Hvis du havde trykket **CAPS SHIFT**, da du trykkede på tallet, så ville du i stedet for **CHR$ 17** have fået **CHR$ 16**, som betyder ink i stedet for paper.

Da du taster to karakterer ind på samme tid, er det også nødvendigt at taste **DELETE** to gange, hvis du ønsker at slette dem igen. Første gang du taster **DELETE**, vil datamaten skrive et spørgsmålstegn eller noget andet, men fortvivl ikke, tast bare **DELETE** en gang til.

Pilmarkørerne ◂, ▸ kan også opføre sig mærkeligt, når markøren flyttes forbi kontrolkaraktererne.

Med **E**-markør:

8 giver **CHR$ 12** og **CHR$ 0** for normal lysstyrke.
9 giver **CHR$ 12** og **CHR$ 1** for ekstra lysstyrke.
CAPS SHIFT og **8** giver **CHR$ 18** og **CHR$ 0** for 'ingen blink'.
CAPS SHIFT og **9** giver **CHR$ 9** og **CHR$ 1** for 'blink'.

Der er et par til med normal markør (**L**-markør):

CAPS SHIFT og **3** giver **CHR$ 20** og **CHR$ 0** for normale karakterer.
CAPS SHIFT og **4** giver **CHR$ 20** og **CHR$ 1** for inverse karakterer.

Her er en komplet oversigt over den øverste række på tastaturet:

MODE	SHIFT										
E	SYMBOL	**DEF FN**	**FN**	**LINE**	**OPEN #**	**CLOSE #**	**MOVE**	**ERASE**	**POINT**	**CAT**	**FORMAT**
	CAPS	Ink blue	Ink red	Ink magenta	Ink green	Ink cyan	Ink yellow	Ink white	Flash off	Flash on	Ink black
	NONE	Paper blue	Paper red	Paper magenta	Paper green	Paper cyan	Paper yellow	Paper white	Normal brightness	Extra bright	Paper black
G	EITHER									EXIT GRAPHICS	**DELETE**
	NONE									EXIT GRAPHICS	**DELETE**
	CAPS	**EDIT**	CAPS LOCK	TRUE VIDEO	INVERSE VIDEO	⇧	⇧	⇧	⇩	GRAPHICS MODE	**DELETE**
K, L or C	SYMBOL	!	@	#	$	%	&	'	(	)	—
	NONE	1	2	3	4	5	6	7	8	9	0

25. Kulør på tilværelsen

Funktionen **ATTR** har formen:

ATTR (linje, kolonne)

Funktionens to argumenter er linje- og kolonnetallene, som du også bruger sammen med **AT**, og resultatet er et tal, som viser farven og lign. på den tilsvarende karakterposition på skærmen. Du kan bruge dette i udtryk lige så frit som andre funktioner. Tallet, som er resultatet, er summen af fire følgende tal:

128, hvis karakteren blinker, og 0 hvis den ikke gør.

64, hvis karakteren har ekstra lysstyrke, og 0 hvis den ikke har.

8* koden for paper-farven.

koden for ink-farven.

Eks. hvis karakteren blinker og har gul paper-farve og blå ink-farve, så vil de fire tal, som vi skal lægge sammen, være: 128, 0, 8*6=48, og 1, som giver resultatet 177.

Test dette ved at skrive:

PRINT AT 0,0; FLASH 1;PAPER 6;INK 1;" ";ATTR (0,0)

Øvelser

1. Prøv:

PRINT "B"; CHR$ 8; OVER 1;"/";

Der, hvor **/** har skåret gennem B, har den efterladt et hvidt punkt. Det er den måde, som ZX Spectrum skriver 'over' en tekst på: to paper eller to ink giver et paper; en af hver giver en ink. Det har den interessante virkning, at hvis du skriver 'over' en tekst to gange med den samme karakter, så vil du få den oprindelige karakter igen. Hvis du nu taster:

PRINT CHR$ 8; OVER 1;"/"

hvorfor får du så et pletfrit **B** tilbage?

2. Tast:

PAPER 0: INK 0

Er det ikke godt, at det ikke virker på den nederste del af skærmen?

Tast nu:

BORDER 0

og se så, hvor godt datamaten holder øje med dig!

3. Kør dette program:

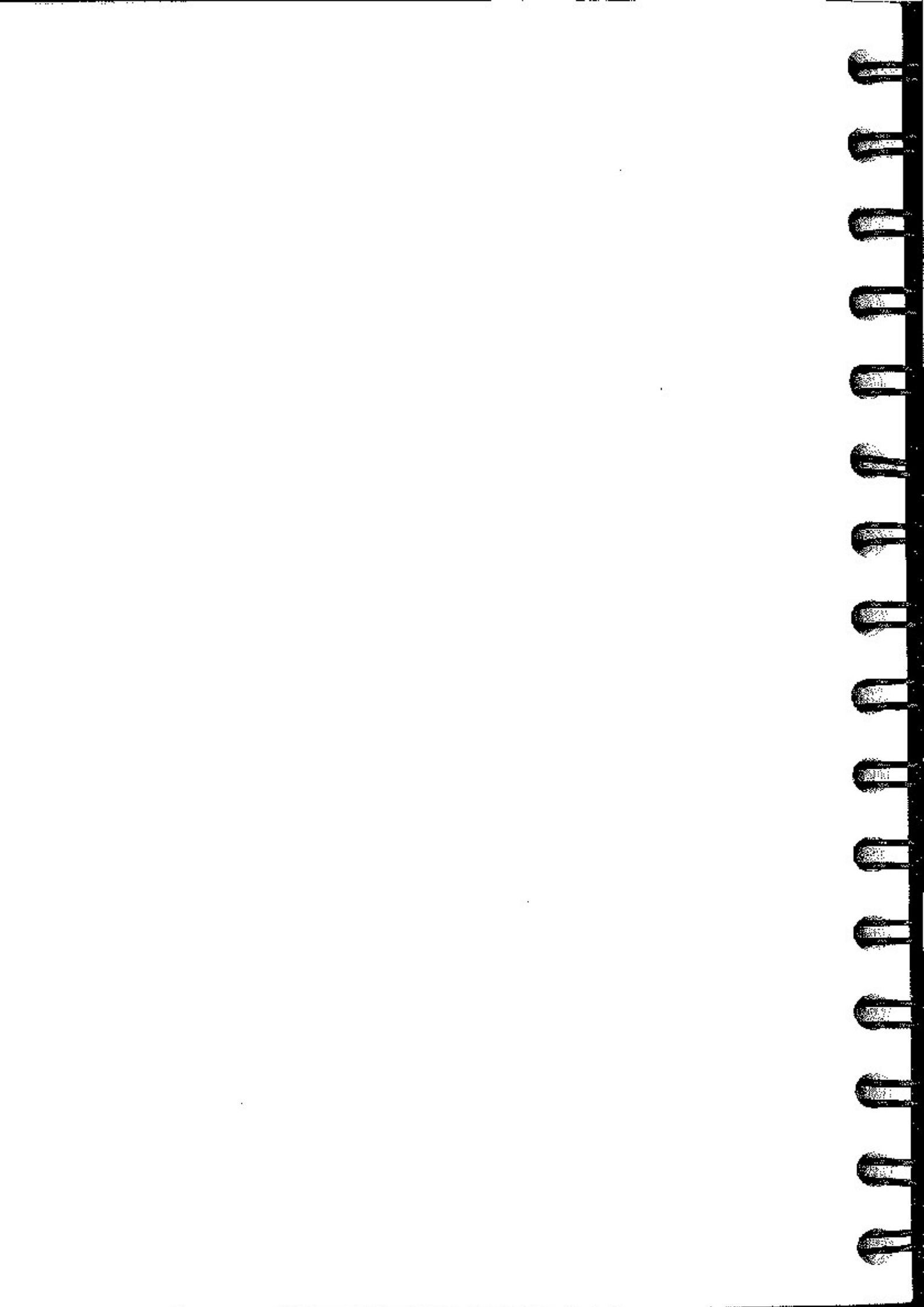
```
10 POKE 22527+RND*704,RND*127
20 GO TO 10
```

Du skal ikke bryde hjernen med, hvordan det virker. Det ændrer farverne på kvadrater på skærmen, og **RND** skal sikre, at det sker tilfældigt. Hvis du evt. ser diagonale striber, så skyldes det, at **RND** (som tidligere beskrevet) ikke er rigtigt tilfældig.

4. Indtast eller **LOAD** skak-brik-programmet fra kapitel 23 og indtast derefter dette program, som tegner et diagram med dem:

```
  5 REM braet tegnes
 10 LET bb=1: LET bw=2: REM rødt og blaat braet
 15 PAPER bw: INK bb: CLS
 20 PLOT 79,128: REM kanten
 30 DRAW 65,0: DRAW 0,-65
 40 DRAW -65,0: DRAW 0,65
 50 PAPER bb
 60 REM braet
 70 FOR n=0 TO 3: FOR m=0 TO 3
 80 PRINT AT 6+2*n,11+2*m;" "
 90 PRINT AT 7+2*n;10+2*m;" "
100 NEXT m: NEXT n
110 PAPER 8
120 LET pw=6: LET pb=5: REM farverne på hvide og sorte brikker
200 DIM b$(8,8): REM brikkernes position
205 REM udgangsposition
210 LET b$(1)="rnbqkbnr"
220 LET b$(2)="pppppppp"
230 LET b$(7)="PPPPPPPP"
240 LET b$(8)="RNBQKBNR"
300 REM viser braet
310 FOR n=1 TO 8: FOR m=1 TO 8
320 LET bc=CODE b$(n,m): INK pw
325 IF bc=CODE" " THEN GO TO 350: REM mellemrum
330 IF bc>CODE "Z" THEN INK pb: LET bc=bc-32:
    REM små sorte bogstaver
340 LET bc=bc+79: REM omformer til grafik
350 PRINT AT 5+n, 9+m; CHR$ bc
360 NEXT m: NEXT n
400 PAPER 7: INK 0
```

26



26. Grafik

I dette kapitel skal vi se, hvorledes der tegnes på ZX Spectrum. Den skærm, du bruger til at skrive på, har 22 linjer og 32 kolonner, d.v.s.22*32=704 karakterpositioner. Som du måske husker fra kapitel 25, er hver af disse karakterpositioner opbygget af et 8*8 kvadrat af punkter, og disse kaldes for billedpunkter eller pixels.

Et billedpunkt er defineret ved to tal, dets koordinater. Det første, dets x-koordinat, fortæller, hvor langt punktet ligger fra venstre side af skærmen. Det andet, dets y-koordinat, hvor langt punktet er fra bunden af skærmen. Disse koordinater skrives sædvanligvis som et par i parentes, så (0,0), (255,0), (0,175) og (255,175) betegner henholdsvis nederste venstre, nederste højre, øverste venstre og øverste højre hjørne.

Kommandoen:

PLOT x-koordinat, y-koordinat

tegner billedpunktet ved disse koordinater med ink-farven, så dette "mæslinge"-program:

```
10 PLOT INT (RND*256), INT (RND*176): INPUT a$: GO TO 10
```

'plotter' et vilkårligt punkt på skærmen, hver gang du trykker **ENTER.**

Her er et mere anvendeligt program. Det tegner det grafiske billede af sinusfunktionen (en sinuskurve) for værdier mellem 0 og 2π (PI).

```
10 FOR n=0 TO 255
20 PLOT n,88+80*SIN (n/128*PI)
30 NEXT n
```

Det næste program tegner en 'graf' af **SQR**-funktionen mellem 0 og 4 (der tegnes faktisk en del af en parabel):

```
10 FOR n=0 TO 255
20 PLOT n, 80*SQR (n/64)
30 NEXT n
```

Bemærk, at disse punktkoordinater er noget forskellige fra de linje- og kolonnekoordinater, vi benyttede i forbindelse med **PRINT AT**-kommandoer. I kapitel 24 fandt du et diagram, der kan være meget nyttigt, når du arbejder med koordinater.

For at gøre det lettere for dig at tegne, vil datamaten kunne tegne lige linjer og cirkler og 'dele' af cirkler ved hjælp af kommandoerne **DRAW** og **CIRCLE**.

Kommandoen **DRAW** tegner en linje og lager formen:

DRAW x,y

Startpunktet for linjen er punktet, hvor den sidste **PLOT, DRAW** eller **CIRCLE**-kommando stoppede (dette punkt kaldes for **PLOT**-positionen, og **RUN, CLEAR, CLS**

og **NEW** nulstiller dette punkt og giver det koordinaterne (0,0)), mens slutpunktet er de koordinater, som skrives efter **DRAW**-kommandoen.

Eksperimentér med **PLOT** og **DRAW**-kommandoerne f.eks.:

```
PLOT 0,100: DRAW 80, -35
PLOT 90,150: DRAW 80, -35
```

Bemærk, at tallene i en **DRAW**-kommando kan være negative i modsætning til tallene i en **PLOT**-kommando, som ikke må være negative.

Du kan også bruge **PLOT** og **DRAW** med farver, men du skal huske, at du kun kan bestemme farven på en hel karakter og ikke på hvert billedpunkt. Når et punkt 'plottes', tegnes det med ink-farven, og samtidigt får hele karakterpositionen den aktuelle ink-farve.

Dette program viser dette:

```
10 BORDER 0: PAPER 0: INK 7: CLS: REM sort skærm
20 LET x1=0: LET y1=0: REM linjestart
30 LET c=1: REM ink farven starter med blå
40 LET x2=INT (RND*256): LET y2=INT (RND*176):
   REM slut på tilfældig linje
50 DRAW INK c;x2-x1,y2-y1
60 LET x1=x2: LET y1=y2: REM næste linje starter,
   hvor den sidste sluttede.
70 LET c=c+1: IF c=8 THEN LET c=1: REM ny farve
80 GO TO 40
```

Jo længere tid programmet kører, des bredere bliver linjerne, og dette skyldes, at en linje ændrer alle ink-farverne i alle karakterpositioner, den passerer. Bemærk, at du kan bruge **PAPER, INK, FLASH, BRIGHT, INVERSE** og **OVER** kommandoer sammen med kommandoerne **PLOT** og **DRAW** på samme måde, som når **PRINT** og **INPUT**-kommandoerne bruges. Kommandoerne skal stå mellem nøgleordet og koordinaterne og skal afsluttes med et semikolon eller et komma.

Du kan også bruge **DRAW** til at tegne dele af cirkler med, i stedet for lige linjer. Dette gøres ved at tilføje en ekstra koordinat, der viser vinklen:

```
Draw x,y,a
```

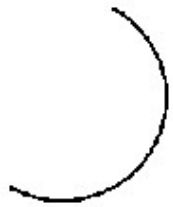
x og **y** bestemmer, som før, slutpunktet, og **a** er den længde, som cirkelbuen skal have (i radianer). Hvis **a** er positiv, så drejer cirklen venstre om, og hvis **a** er negativ, så drejer den højre om. Du kan f.eks. prøve at tegne en cirkelbue med længden 2pi, hvor du så naturligvis får tegnet en hel cirkel. Hvis **a** f.eks. er pi så tegnes en halvcirkel, og hvis **a** er 0.5*pi, så tegnes en fjerdedel af en cirkel o.s.v.

Antag at **a** er lig pi. Uanset hvilke værdier **x** og **y** har, vil der blive tegnet en halvcirkel. Kør:

```
10 PLOT 100,100: DRAW 50,50,PI
```

som vil tegne dette:

slutter ved (150,150)



starter ved (100,100)

Tegningen starter mod syd-vestlig retning, men slutter i nord-vest. I mellemtiden er den drejet 180 grader eller pi-radianer (værdien af **a**). Kør programmet flere gange med forskellige værdier for **a**, f.eks.: **—PI, PI/2, 3*PI/2, PI/4,1,0,** o.s.v.

Den sidste kommando, som vi skal se på i dette kapitel, er kommandoen **CIRCLE**, som tegner en hel cirkel. Du skal opgive koordinaterne for centrum og for radius, sådan her:

CIRCLE x-koordinat, y-koordinat, radius

Ligesom når du anvender **PLOT** og **DRAW,** kan du også med **CIRCLE** anvende forskellige farver.

Funktionen **POINT** bruges til at oplyse dig, om et billedpunkt er skrevet med ink eller med paper-farve. Funktionen har to argumenter, nemlig koordinaterne for punktet, som skal være omsluttet af parenteser.

Resultatet er 0, hvis punktet har paper-farve, og 1, hvis punktet har ink-farve.

Prøv:

CLS: PRINT POINT (0,0): PLOT 0,0: PRINT POINT (0,0)

Tast:

Paper 7: INK 0

og lad os undersøge, hvordan **INVERSE** og **OVER** virker sammen med **PLOT**-kommandoen. Disse to påvirker kun de relevante billedpunkter og ikke resten af karakterpositionen.

Normalt er punkterne slukket (0) i en **PLOT**-kommando, så du behøver kun at nævne dem for at tænde dem (1).

Her er en liste over de forskellige muligheder:

PLOT; – Dette er den almindelige version. Her 'plottes' et punkt med ink-farve, d.v.s. billedpunkterne viser ink-farven.

PLOT INVERSE 1; – Nu 'plottes' der omvendt, d.v.s. at punkterne viser 'paper'-farven.

PLOT OVER 1; – Dette ændrer billedpunktet fra, hvad det end var før, så hvis punktet før havde ink-farve, så får det nu i stedet paper-farve og omvendt.

PLOT INVERSE 1; OVER 1; – Dette bevarer billedpunktet præcis, som det var før, men bemærk, at det ændrer PLOT-positionen, så du kan simpelthen bruge det til dette.

Prøv at fylde skærmen med karakterer, hvor der skrives med sort på hvidt og tast derefter:

```
PLOT 0,0: DRAW OVER 1;275,175
```

Dette vil tegne en rimeligt pæn linje, selv om der er huller, hver gang den rammer noget tidligere skrevet. Prøv at give datamaten den samme kommando igen. Linjen vil nu forsvinde uden at efterlade noget spor. Dette er den store fordel ved **OVER 1**. Hvis du havde tegnet linjen ved at bruge:

```
PLOT 0,0: DRAW 255,175
```

og slettet den ved at bruge:

```
PLOT 0,0: DRAW INVERSE 1; 255,175
```

så vil du også have slettet noget af det skrevne.

Prøv nu:

```
PLOT 0,0: DRAW OVER 1;250,175
```

og visk det ud ved at skrive:

```
DRAW OVER 1;–250,–175
```

Dette virker ikke helt efter hensigten, da det ikke er helt de samme punkter, linjen bruger på tilbagevejen. Det er nødvendig at slette en linje i nøjagtig samme retning, som den er skrevet.

En måde at lave unormale farver på er at blande to normale sammen i et enkelt kvadrat, ved hjælp af brugerdefinérbar grafik. Kør dette program:

```
1000 FOR n=0 TO 6 STEP 2
1010 POKE USR "a"+n, BIN 01010101: POKE USR "a"+n+1,
     BIN 10101010
1020 NEXT n
```

der giver brugerdefinérbar grafik som et skakbræt. Hvis du skriver denne karakter, (grafik-indstilling og **a**) med rød ink-farve på gul paper-farve, så vil du få en rimelig god orange-farve.

Øvelser

1. Brug kommandoerne **PAPER, INK, FLASH** og **BRIGHT** sammen med **PLOT**-kommandoen. Det er disse, der påvirker karakterpositionen og billedpunktet. Normalt er det, som om **PLOT**-kommandoen var startet med:

PLOT PAPER 8; FLASH 8; BRIGHT; . . .

og kun ink-farven på en karakterposition forandres, hvis der 'plottes' noget, men du kan ændre dette, hvis du vil.

Vær særligt omhyggelig, når du bruger **INVERSE 1**, fordi dette sætter billedpunktet til at vise paper-farven, men ændrer ink-farven, og det var måske ikke, hvad du havde forventet.

2. Prøv at tegne cirkler ved at bruge **SIN** og **COS** (hvis du har læst kapitel 19, kan du sikkert gøre det).

Kør dette:

```
10 FOR n=0 TO 2*PI STEP PI/180
20 PLOT 100+80*COS n,87+80*SIN n
30 NEXT n
40 CIRCLE 150,87,80
```

Du kan se, at **CIRCLE**-kommandoen er langt hurtigere omend ikke så præcis.

3. Prøv:

```
CIRCLE 100,87,80: DRAW 50,50
```

Ud fra dette kan du se, at **CIRCLE**-kommandoen placerer **PLOT**-positionen et ubestemt sted – det findes altid ca. halvvejs oppe på den højre side af cirkelbunden. Så hver gang du bruger en **CIRCLE**-kommando, bør du efterfølge den af en **PLOT**-kommando, før du fortsætter med at tegne.

4. Her er et program, som kan tegne en graf for de fleste funktioner. Først spørger det efter et tal **n**. Grafen vil blive 'plottet' for værdien fra **–n** til **+n**. Dernæst spørger det efter selve funktionen som en streng. Strengen skal være et udtryk, der bruger **x** som argument for funktionen.

```
 10 PLOT 0,87: DRAW 255,0
 20 PLOT 127,0: DRAW 0,175
 30 INPUT s,e$
 40 LET t=0
 50 FOR f=0 TO 255
 60 LET x=(f–128)*s/128: LET y=VAL e$
 70 IF ABS y>87 THEN LET t=0: GO TO 100
 80 IF NOT t THEN PLOT f,y+88: LET t=1: GO TO 100
 90 DRAW 1,y–gammel y
100 LET gammel y=INT (y+.5)
110 NEXT f
```

Kør det og prøv f. eks. at bruge **10** som **n** og **10*TAN x** som funktion. Programmet vil så 'plotte' grafen for tan **x** i intervallet –10 til 10.

27

27. Om tid og bevægelse

Ganske ofte vil du ønske, at dit program varer et bestemt stykke tid, og her vil du finde **PAUSE**-kommandoen meget anvendelig:

PAUSE n

stopper beregningerne og viser skærmbilledet i n perioder (hvor 1 periode er et tv-billede. Der er 50 billeder i sekundet, d.v.s. 50 perioder pr. sekund). n kan have en værdi op til 65535 eller lidt under 22 minutter. Hvis n er nul, betyder det: **PAUSE** i det uendelige. En **PAUSE** kan afbrydes ved at trykke på en tast (**CAPS SHIFT** og mellemrumstasten betyder også en afbrydelse af programmet). Tasten skal aktiveres, efter at **PAUSE**n er begyndt.

Dette program viser sekunderne på et ur:

```
10 REM Først tegnes skiven
20 FOR n=1 TO 12
30 PRINT AT 10-10*COS (n/6*PI),16+10*SIN (n/6*PI);n
40 NEXT n
50 REM Uret startes
60 FOR t=0 TO 200000: REM t er tiden i sekunder
70 LET a=t/30*PI: REM a er et stykke af cirkelbuen, i radianer
80 LET sx=80*SIN a: LET sy=80*COS a
200 PLOT 128,88: DRAW OVER 1;sx,sy: REM viseren tegnes
210 PAUSE 42
220 PLOT 128,88: DRAW OVER 1;sx,sy: REM viseren slettes
400 NEXT t
```

Uret vil stoppe efter ca. 55.5 timer, fordi vi har begrænset dets virketid i linje 60, men du kan nemt få det til at virke længere. Læg også mærke til, hvordan takten er styret af linje 210. Du havde nok forventet, at **PAUSE 50** havde fået uret til at 'tikke' én gang i sekundet, men datamatens beregning af programmet tager også nogen tid, og det må der kompenseres for. Graden af denne kompensation finder man bedst ved at prøve sig frem og kontrollere 'datamaturet' med et rigtigt ur. En justering af linje 210 kan imidlertid ikke blive helt nøjagtig, for en ændring på 1 periode betyder en ændring på 2% eller ca. ½ time pr. døgn.

Der findes imidlertid en meget mere præcis måde at måle tid. Det gøres ved at bruge kommandoen **PEEK**, som kan udlæse indholdet af bestemte områder i maskinens hukommelse. **PEEK** forklares i kapitel 34.

Vi bruger udtrykket:

(65536*PEEK 23674+256*PEEK 23673+PEEK 23672)/50

Resultatet af udtrykket er et tal, som skifter hvert sekund. Tallet starter med 0, når maskinen tændes, og tæller op til 3 dage og 21 timer, hvorefter tallet igen bliver 0.

Her er et revideret ur-program, som bruger det nye udtryk:

```
 10 REM Først tegnes skiven
 20 FOR n=1 TO 12
 30 PRINT AT 10-10*COS (n/6*PI),16+10*SIN (n/6*PI);n
 40 NEXT n
 50 DEF FN t()=INT ((65536*PEEK 23674+256*PEEK
    23673+PEEK 23672)/50): REM sekunder siden start
100 REM Nu startes uret
110 LET t1=FN t()
120 LET a=t1/30*PI: REM a er et stykke af cirkelbuen i radianer
130 LET sx=72*SIN a: LET sy=72*COS a
140 PLOT 131,91: DRAW OVER 1;sx,sy: REM viser tegnes
200 LET t=FN t()
210 IF t<=t1 THEN GO TO 200: REM Venter indtil næste viser skal
    tegnes
220 PLOT 131,91: DRAW OVER 1;sx,sy: REM Den gamle viser
    viskes ud
230 LET t1=t: GO TO 120
```

Maskinens indvendige ur har en præcision på 0,01% (Eller ca. 10 sekunder pr. dag), så længe datamaten kører dette program, men det indre ur stoppes, hvis du giver en **BEEP**-kommando, eller hvis båndoptager, printer eller andre evt. tilsluttede enheder til ZX Spectrum bruges. Maskinen vil så tabe tid.

Tallene **PEEK 23674, PEEK 23673** og **PEEK 23672** er indbygget i datamaten og bruges til at tælle 1/50 sekund. Hvert af tallene kan anlage værdier mellem 0 og 255. De starter med at være 0 og øges derefter gradvist til 255, hvorefter de igen bliver 0. **PEEK 23672** er det tal, som oftest forøges. For hvert 1/50 sekund forøges det med 1. Når det når 255, er næste trin 0, og samtidigt forøges **PEEK 23673** med 1. Når **PEEK 23673** er forøget til 255, så skifter den over til 0 igen efter (255/50) sekunder, og skubber samtidigt **PEEK 23674** op med 1. Dette skulle være forklaring nok på, hvordan ovenstående udtryk virker. Prøv omhyggeligt at tænke over følgende:

Antag at de tre tal er 0 for **PEEK 23674**, 255 for **PEEK 23673** og 255 for **PEEK 23672**. Det betyder, at det er ca. 21 minutter, siden datamaten blev tændt – så vores udtryk burde være:

(65536*0+256*255+255)/50=1310.7

Men her skal du passe på! Næste gang maskinen tæller 1/50 sekund, vil de tre tal ændres til 1, 0, 0, som jo er rigtig nok. Men undertiden sker dette, mens maskinen er nået halvvejs gennem beregningen af udtrykket: datamaten behandler **PEEK 23674** som 0, men ændrer samtidigt de to andre tal til 0, før den kan nå at udlæse **PEEK**-værdierne, og svaret vil så være:

(65536*0+256*0+0)/50=0

som er fuldstændigt forkert.

En simpel måde at løse problemet på er at behandle udtrykket to gange efter hinanden og så tage det største resultat.

Hvis du ser godt på ovenstående program, så vil du se, at det gør det af sig selv. Her er en smart måde at anvende det på. Definér funktionerne:

```
10 DEF FN m(x,y)=(x+y+ABS(x-y))/2: REM det største af x og y
20 DEF FN u()=(65536*PEEK 23674+256*PEEK 23673+PEEK
   23672)/50: REM tiden, maaske forkert
30 DEF FN t()=FN m(FN u()): REM tiden, rigtig
```

Du kan ændre de tre tal, således at de giver tiden på et normalt ur, i stedet for tiden fra maskinen blev tændt. Hvis du f.eks. ønsker at 'stille uret' til kl. 10.00, så kan du udregne, at det er 10*60*60*50=1800000 halvtredsindstyvendedele af et sekund og at:

1800000=65536*27+256*119+64

For at stille de tre tal skriver du:

```
POKE 23674,27: POKE 23673,119: POKE 23672,64
```

Funktionen **INKEY$**, der ikke har noget argument, aflæser tastaturet. Hvis du trykker på en enkelt tast (eller på en af **SHIFT**-tasterne og en enkelt tast), så skrives karakteren på denne tast, ganske som hvis **L**-markøren var fremme. Hvis ingen tast nedtrykkes, vil resultatet være en tom tekststreng ("").

Prøv at lave dette program, der fungerer som en skrivemaskine:

```
10 IF INKEY$ <>"" THEN GO TO 10
20 IF INKEY$="" THEN GO TO 20
30 PRINT INKEY$;
40 GO TO 10
```

Her venter linje 10 på, at du skal løfte din finger fra 'den gamle tast', og linje 20 venter på, at du skal taste noget nyt.

Husk at modsat **INPUT** venter **INKEY$** ikke på dig. Så du skal ikke trykke **ENTER**. På den anden side, hvis du intet trykker, så taber du din chance.

Øvelser

1. Hvad sker der, hvis du udelader linje 10 i skrivemaskineprogrammet?

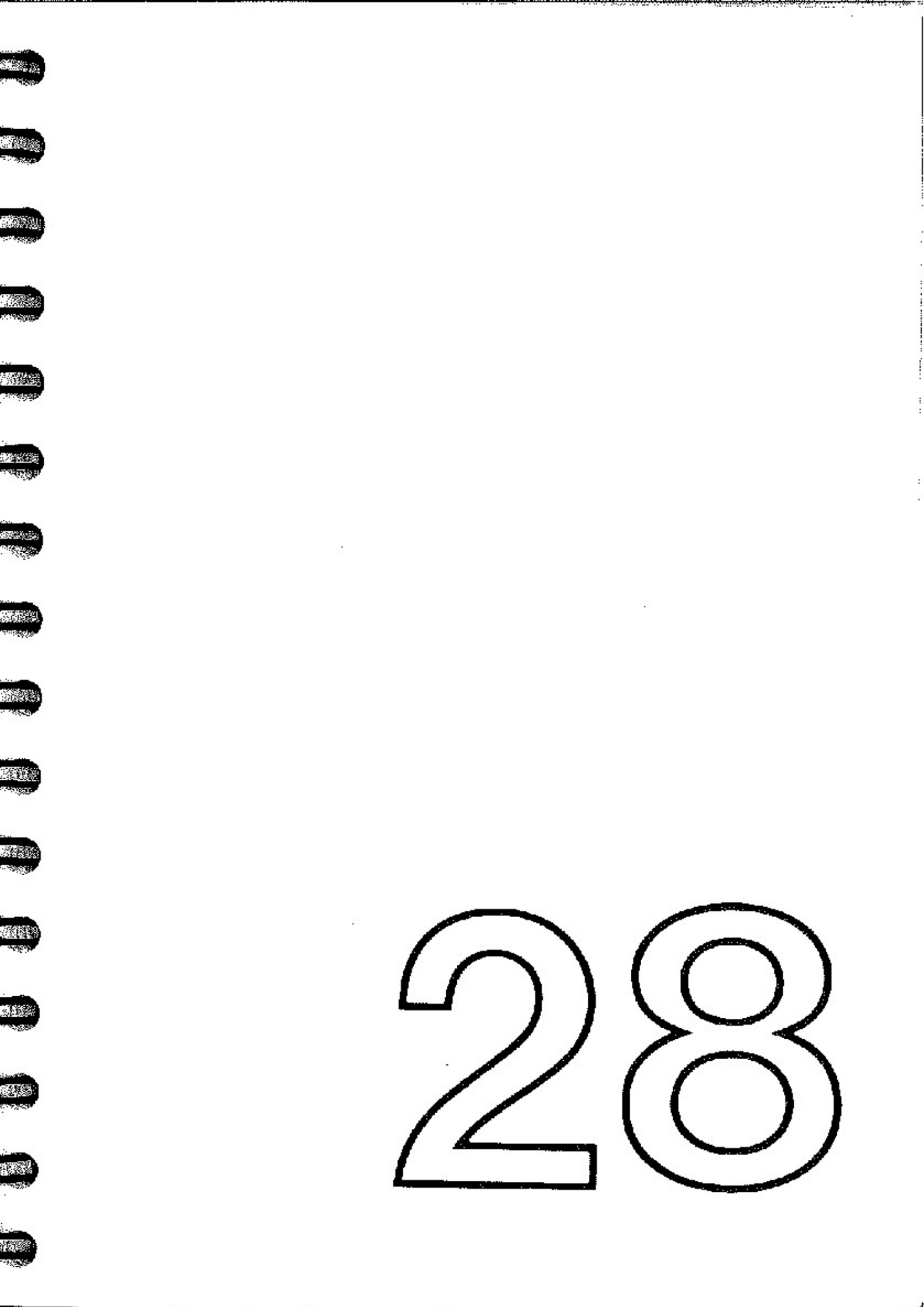
2. En anden anvendelse for **INKEY$** er i forbindelse med **PAUSE** -- som f.eks. i dette alternative skrivemaskineprogram:

```
10 PAUSE 0
20 PRINT INKEY$;
30 GO TO 10
```

3. Lav det sidste ur-program om, så det også viser minutter og timer, og så disse visere tegnes hvert minut. Såfremt du ønsker at gøre noget ekstra ud af det, så gør det sådan, at der hvert kvarter vises noget 'specielt' på skærmen – du kan f.eks. lave 'Big Ben' som kimer ved hjælp af **BEEP**. (Se næste kapitel).

4. (For sadister). Prøv dette:

```
10 IF INKEY$="" THEN GO TO 10
20 PRINT AT 11,14;"AV!!"
30 IF INKEY$<>"" THEN GO TO 30
40 PRINT AT 11,14;" "
50 GO TO 10
```


28

28. BEEP

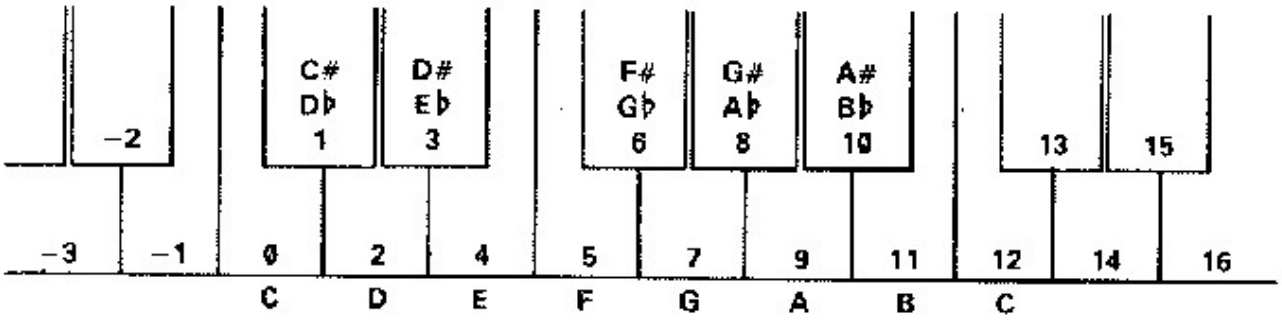
Hvis du ikke allerede har opdaget, at ZX Spectrum har en indbygget højttaler, så bør du læse kapitel 8, før du fortsætter her.

Højttaleren giver en lyd fra sig ved brug af kommandoen **BEEP**, således:

BEEP tonelængde, tonehøjde

hvor tonelængde og tonehøjde repræsenteres af numeriske udtryk. Tonelængden angives i sekunder og tonehøjden i halvtoner omkring det midterste C (negative tal for halvtoner under det midterste C).

Her er et diagram, som viser tonehøjde-værdierne for alle noder i en oktav på et klaver:



For at nå højere eller lavere noder må du lægge 12 til eller trække 12 fra for hver oktav, du går op eller ned.

Hvis du har et klaver ved siden af dig, når du programmerer en melodi, så vil dette diagram sikkert være nok til, at du kan finde de forskellige tonehøjder. Ønsker du imidlertid at overføre et skrevet musikstykke direkte fra noderne til ZX Spectrum, anbefaler vi dig at opstille et diagram med tonehøjde-værdier for hver nodelinje, og med tonearten taget i betragtning.

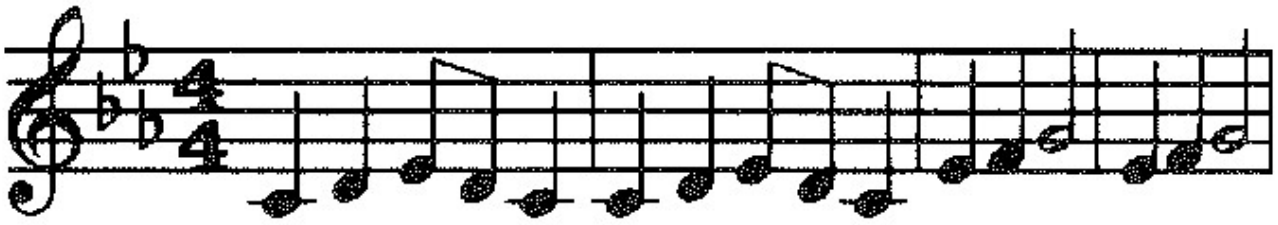
Prøv at indtaste dette:

```
10 PRINT "Frere Gustav"
20 BEEP 1,0: BEEP 1,2: BEEP .5,3: BEEP .5,2: BEEP,1,0
30 BEEP 1,0: BEEP 1,2: BEEP .5,3: BEEP .5,2: BEEP 1,0
40 BEEP 1,3: BEEP 1,5: BEEP 2,7
50 BEEP 1,3: BEEP 1,5: BEEP 2,7
60 BEEP .75,7: BEEP .25,8: BEEP .5,7: BEEP .5,5: BEEP .5,3:
   BEEP .5,2: BEEP 1,0
70 BEEP .75,7: BEEP .25,8: BEEP .5,7: BEEP .5,5: BEEP .5,3:
   BEEP .5,2: BEEP 1,0
80 BEEP 1,0: BEEP 1,-5: BEEP 2,0
90 BEEP 1,0: BEEP 1,-5: BEEP 2,0
```

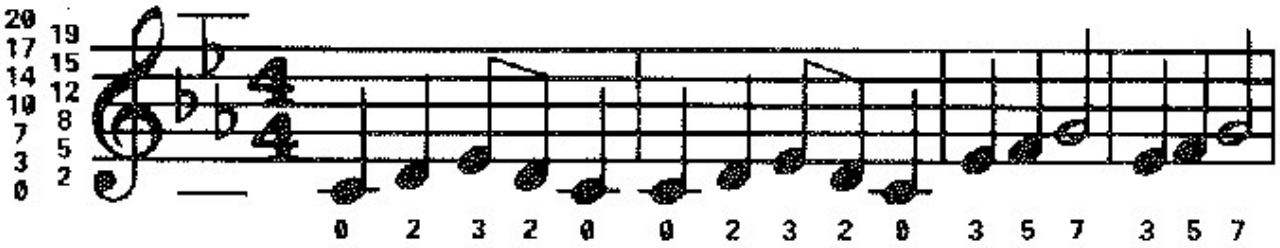
Når du kører dette, vil du kunne høre begravelsesmarchen fra Mahlers første symfoni.

28. Beep

Antag, at din melodi er skrevet i C-mol som musikstykket ovenfor. Begyndelsen vil se sådan ud:



Du kan skrive tonehøjde-værdierne for noderne sådan:



Vi har for en god ordens skyld indsat to styrelinjer. Bemærk, at b for E (ES) imellem vore styrelinjer ikke blot påvirker det øverste F, så tonen sænkes fra 16 til 15, men også E på nederste linje, så den sænkes fra 4 til 3. Det vil nu være ret nemt at finde højdeværdien for enhver node på linjen.

Hvis du ønsker at ændre stykkets toneart, så er det bedst at opstille en variabel **t** og at indsætte **t+** lige før hver tonehøjdeværdi. Så kommer anden linje til at se sådan ud:

```
20 BEEP 1,t+0: BEEP 1,t+2: BEEP .5,t+3:
   BEEP .5,t+2: BEEP 1,t+0
```

Før du kører programmet, må du nødvendigvis give variablen **t** en værdi: – 0 for C-mol, 2 for D-mol, 12 for C-mol en oktav højere, o.s.v. Du kan stemme datamaten efter andre instrumenter ved at justere på variablen **t** med små værdier.

Du må også udarbejde varighed for alle noderne. Vi har her givet en fjerdedelsnode længden ét sekund. Det er mere fleksibelt at opstille en variabel **v** at definere længden ud fra det.

Linje 20 skal så se sådan ud:

```
20 BEEP v, t+0: BEEP v, t+2: BEEP v/2, t+3:
   BEEP v/2, t+2: BEEP v, t+0
```

Ved at give variablen **v** passende værdier kan du nemt variere hastigheden.

Da der kun er indbygget én højttaler i datamaten, kan du kun spille én tone ad gangen, så hvis du vil spille flerstemmig musik, må du selv synge til.

Prøv selv at lave nogle melodier. Start med noget let, f.eks. "Lille Peter Edderkop",

og benyt den fremgangsmåde, som tidligere er beskrevet. Et enkelt instrument som en blikfløjte vil muligvis være til hjælp, hvis du ikke har et klaver. Tast følgende:

FOR n=0 TO 1000: BEEP .5,n: NEXT n

Dette program vil spille toner så højt, det kan, og derefter stoppe med fejlrapport **B integer out of range.** Du kan udskrive værdien af n for at se, hvor "højt" det kom.

Prøv det samme med negative værdier for n. De laveste toner vil lyde som klik. De høje toner består faktisk også af klik, men de er så hurtige, at øret ikke kan adskille dem.

Kun de midterste toner er helt egnede til musik; de laveste toner lyder som klik, og de høje toner er "tynde" og vibrerende. Indtast denne programlinje:

10 BEEP .5,0: BEEP .5,2: BEEP .5,4: BEEP .5,5: BEEP .5,7: BEEP .7,9: BEEP .5,11: BEEP .5,12: STOP

Dette spiller C-durskalaen, der bruger alle de hvide tangenter på et klaver fra det midterste C til C en oktav højere oppe. Denne skala er afstemt nøjagtigt som på et klaver og kaldes "veltemporeret", fordi halvtone-intervallet er det samme hele vejen op ad skalaen.

En violinist vil imidlertid spille skalaen på en helt anden måde, så den lyder mere behagelig. Han kan gøre dette ved at flytte sine fingre en smule op og ned ad strengene, noget der ikke er muligt på et klaver. For en violinist vil den naturlige skala lyde:

20 BEEP .5,0: BEEP .5,2.039: BEEP .5,3.86: BEEP .5,4.98: BEEP .5,7.02: BEEP .5,8.84: BEEP .5,10.88: BEEP .5,12: STOP

Måske kan du, og måske kan du ikke høre nogen forskel på disse to – nogle mennesker kan. Den første mærkbare forskel er, at den tredje tone er lavere på den naturlige skala. Hvis du er rigtig perfektionist, så vil du muligvis bruge denne ved programmering. Ulempen er, at selv om den virker perfekt i tonearten C, går det mindre godt med andre tonearter, da hver har sin egen naturlige skala – og i nogle tonearter virker den ualmindelig dårligt. Den veltemporerede skala er kun lidt ved siden af og virker lige godt i alle tonearter.

Dette er kun et lille problem på datamaten, da du kan ændre tonehøjden med variablen t.

Noget musik – f.eks. indisk – bruger en afstand i tonehøjde, der er mindre end en halvtone. Du kan uden vanskeligheder programmere disse ind i **BEEP**-kommandoen. F.eks. har kvarttonen over det midterste C en værdi på .5.

Hvis du vil have tastaturet til at "bippe" i stedet for at "klikke", når du indtaster noget i datamaten, så skriv:

POKE 23609,255

Det andet tal bestemmer længden af lyden (prøv forskellige værdier mellem 0 og 255). Når værdien er 0, er bippet så kort, at det lyder som et blødt klik.

Hvis du er interesseret i at bruge lyden yderligere f.eks. på en anden højttaler, så kan du finde signalet på **MIC** og **EAR**-stikket (størst er styrken på **EAR**-stikket). Du kan tilslutte et par hovedtelefoner eller en forstærker (her er **MIC** nok det bedste). Du kan også tilslutte en båndoptager, så Spectrum kan spille med sig selv.

Du vil ikke kunne ødelægge din Spectrum – selv ikke, hvis du kortslutter **MIC** og **EAR** stikkene, så eksperimentér bare, indtil du finder frem til det bedste udgangssignal til dit formål.

Øvelse

1. Omskriv Mahler-programmet, så det i stedet anvender **FOR-NEXT**-løkker til taktterne.

Programmér datamaten, så den spiller hele resten af Mahlers første symfoni.

29

29. Arkivering på bånd

De grundlæggende meloder for opbevaring at data på en båndoptager med kommandoerne **SAVE**, **LOAD** og **VERIFY** er beskrevet i kapitel 6, og dette bør gennemlæses, og de forskellige procedurer gennemprøves, før du fortsætter her.

Vi har set, at **LOAD** sletter de gamle programmer og variabler i datamaten, inden et nyt program indkøres fra båndet.

Der findes en anden kommando, **MERGE**, som ikke gør dette. **MERGE** kan **LOAD**e to programmer, og den sletter kun en gammel programlinje eller variabel, hvis der er en ny med samme linjenummer eller samme variabelnavn i det andet program.

Indtast terningprogrammet fra kapitel 20, **SAVE** det under navnet "terning". Dernæst indtaster du og kører følgende:

```
 1 PRINT 1
 2 PRINT 2
10 PRINT 10
20 LET x=20
```

Fortsæt derefter som ved kontrol af et program for evt. fejl, men erstat **VERIFY** "terning", med:

```
MERGE "terning"
```

Hvis du lister programmet, kan du se, at linje 1 og 2 bevares, men linje 10 og 20 er erstattet af linjerne fra terningprogrammet. Ligeledes eksisterer variablen **x** stadig. Prøv at skrive **PRINT x.**

Du har nu set nogle simple eksempler på de fire kommandoer, der bruges ved båndoptagelser.

SAVE, gemmer et program og variabler på bånd.
VERIFY, undersøger og sammenligner program og variabler på et bånd med det, der er i datamaten.
LOAD sletter hele programmet og alle variablerne i maskinen og indsætter nye fra bånd.
MERGE virker som **LOAD**, men sletter ikke program og variabler, medmindre et nyt program har samme linjenummer eller variabelnavn.

Hver af disse kommandoer er efterfulgt af en streng: Ved **SAVE** giver den et navn til programmet på båndet, mens den ved de tre andre kommandoer giver datamaten et navn at søge efter.

Mens datamaten søger, skriver den navnet på hvert program, den passerer.

Der er dog et par afvigelser fra reglerne:

Når du bruger **VERIFY**, **LOAD** og **MERGE**, kan du lade datamaten få en tom streng som navn at søge efter (f.eks. **LOAD " "**), så tager den det første program, der kommer.

En variant ved brug af **SAVE** er:

SAVE streng **LINE** linjenummer

Et program, som gemmes på denne måde, hopper automatisk til det linjenummer, som skrives efter **LINE**, når programmet indkøres med kommandoen **LOAD** (ikke **MERGE**) og **RUN**er sig selv.

Hidtil har vi kun gemt programmer sammen med variabler. Der er to andre metoder, kaldet "arrays" (på dansk: variabelsæt, som vi forkorter til v. sæt) og "bytes".

V. sæt kan behandles på flere måder:

Du kan gemme v. sæt på bånd ved at bruge **DATA** sammen med **SAVE**-kommandoen på denne måde:

SAVE streng **DATA** v. sæt navn ()

"Streng" er navnet på de informationer, som gemmes på bånd, og virker på nøjagtig samme måde, som når almindelige programmer eller bytes lagres.

V. sæt-navnet specificerer de v. sæt, som du ønsker at gemme, så navnet skal bare være et enkelt bogstav eller et enkelt bogstav efterfulgt af et **$**-tegn. Husk parenteserne efter v. sæt navnet: du finder dem måske unødvendige, men du er stadig nødt til at indtaste dem for at gøre det lettere for datamaten. Vær helt sikker på forskellen mellem strengnavne og v. sæt-navne.

Hvis du f.eks. skriver:

SAVE "Blokke" DATA b()

så vil **SAVE** tage v. sættet **b** fra datamaten og gemme det på bånd under navnet "Blokke".

Hvis du taster:

VERIFY "Blokke" DATA b()

så vil datamaten søge efter et v. sæt, gemt under navnet "Blokke" (når den finder det, vil den skrive: "Number array Blokke", (på dansk: "tal-v. sæt")) og sammenligne det med v. sæt **b** i datamaten.

LOAD "Blokke" DATA b ()

finde v. sættet på bånd og – hvis der er plads i datamaten – sletter den et evt. tidligere v. sæt med navnet **b** og indsætter det nye v. sæt fra båndet med navnet **b.**

Du kan ikke bruge **MERGE**, når du gemmer v. sæt.

Du kan gemme karakter (strenge)-v. sæt på nøjagtig samme måde. Når datamaten søger på båndet efter et karakter v. sæt og finder det, skriver den: "Character array" (på dansk: "Karakter v. sæt), efterfulgt af navnet.

Når du indkører et karakter-v. sæt, vil den slette ikke blot det evt. tidligere karakter-v. sæt med samme navn, men også enhver streng med det samme navn.

Byte-lagring bruges for informationsstumper uden henvisning til, hvad informationen bruges til. Det kan være et tv-billede, brugerdefinérbar grafik eller noget andet, som du har lavet.

Her bruger du **CODE** således:

SAVE "billede" CODE 16384,6912

Lagringsenheden i datamaten er *byten* (et tal mellem 0 og 255), og hver byte har en *adresse* (som er tal mellem 0 og 65535).

Det første tal efter **CODE** er adressen på den første byte, som skal gemmes på båndet, og det andet er antallet af bytes, der skal lagres. I vort tilfælde er 16384 adressen på den første byte i skærm-filen (som indeholder tv-billedet), og 6912 er antal bytes, der skal gemmes for at få en kopi af tv-billedet – prøv det.

Navnet "billede" virker på samme måde som navne på programmer.

For at få det tilbage i datamaten, skriv:

LOAD "billede" CODE

Du kan også skrive tal efter CODE, således:

LOAD navn **CODE** start, længde

Her bruges "længde" som sikkerhed. Når datamaten har fundet de pågældende bytes med det rigtige navn, så vil den ikke tage imod dem, hvis der er flere end den forventede længde. Eftersom der er flere data, end du ventede, risikerer du jo at slette et område i hukommelsen, mener datamaten. Derfor skriver den:
R Tape loading error (på dansk: R Fejl ved indkørsel af data fra bånd). Du kan undlade at skrive længde efter **CODE**, og datamaten vil så indkøre alle bytes, uanset hvor mange der er.

"Start" er den første adresse, som den første byte skal **LOAD**es i. Denne adresse kan være forskellig fra den adresse, hvorfra den blev **SAVE**t.

Hvis det er de samme adresser, så kan du undvære "start" efter **CODE.**

CODE 16384,6912 er så værdifuld til at **SAVE** og **LOAD**e et billede med, at du kan udskifte det med **SCREEN$** – f.eks.:

SAVE "billede" SCREEN$
LOAD "billede" SCREEN$

Dette er et af de sjældne tilfælde, hvor **VERIFY** ikke virker. Du kan ikke bruge **VERIFY** til at kontrollere et skærmbillede, idet billedet ændres, mens **VERIFY** tjækker det. I alle andre tilfælde bør du bruge **VERIFY** efter hvert **SAVE.**

Følgende er en oversigt over de fire kommandoer i dette kapitel.

Navn står for et ethvert strengudtryk og henviser til det navn, som informationerne er gemt under på bånd. Det skal bestå af ASCII-karaktere, hvoraf kun de 10 første bruges.

Der er fire slags informationer, der kan gemmes på bånd: programmer og variabler sammen, tal-v. sæt, karakter-v. sæt og bytes alene.

Når **VERIFY, LOAD** og **MERGE** søger efter informationer på båndet med et givet navn og af en given art, så skriver datamaten art og navn på at det, den finder på skærmen.

Maskinen skriver "Program:", "Number array", "Character array" eller "Bytes". Hvis *navn* er en tom streng (d.v.s. " "), så tages de første informationer på båndet uanset navnet.

SAVE:
SAVEr informationer på bånd under et givet navn. Fejl F kommer, hvis navnet er en tom streng eller indeholder 11 eller flere karakterer.

Når maskinen modtager en **SAVE**-kommando, skriver den altid: **Start tape, then presse any key** (Start tape og pres en eller anden tast) og venter, indtil en af tasterne nedtrykkes, før den **SAVE**r.

1. Programmer og variabler:

SAVE navn

er den normale metode.

SAVE navn **LINE** linje nummer

gemmer programmet og variabler, så **LOAD** automatisk virker som

LOAD og **GO TO** linjenummer.

SAVE navn **LINE** uden "nummer" betyder det samme som **SAVE** navn **LINE 1.** Så når programmet er indkørt i maskinen, kører det sig selv fra starten.

2. Bytes:

SAVE navn **CODE** start, længde

gemmer "længde" (antal) bytes fra adressen "start".

SAVE navn **SCREEN$**

betyder det samme som:

SAVE navn **CODE 16384,6912**

og gemmer tv-billedet.

3. V. sæt:

SAVE navn **DATA** bogstav ()

eller

SAVE navn **DATA** bogstav$ ()

gemmer v. sættet, hvis navn er "bogstav" eller "bogstav$" (dette behøver ikke at have nogen relation til "navn").

VERIFY:
Undersøger informationerne på båndet og sammenligner dem med informationerne i datamatens hukommelse. Hvis disse informationer ikke er de samme, så skrives fejlrapport: **R Tape loading error.**

1. Programmer og variabler:

VERIFY navn

2. Bytes:

VERIFY navn **CODE** start, længde

Hvis de pågældende bytes på båndet indeholder mere end "længden" så skrives fejlrapport R.
Ellers undersøges de aktuelle bytes fra adresse "start".

VERIFY navn **CODE** start

undersøger de pågældende bytes fra adresse "start".

VERIFY navn **CODE**

undersøger de pågældende bytes fra den første adresse, som blev optaget på båndet.

VERIFY navn **SCREEN$**

betyder det samme som:

VERIFY navn **CODE 16384,6912**

men vil næsten altid fejle.

3. V. sæt:

VERIFY navn **DATA** bogstav ()

eller

VERIFY navn **DATA** bogstav $ ()

undersøger v. sæt navnet på båndet og sammenligner det med "bogstav" eller "bogstav $" i hukommelsen.

LOAD
Indkører nye informationer fra bånd og sletter gamle informationer i hukommelsen.

1. Programmer og variabler:

LOAD navn

sletter det gamle program og de gamle variabler og **LOAD**er program og variabler fra bånd. Hvis programmet er gemt på bånd ved brug af **SAVE** navn **LINE**, så hopper det automatisk til linjenummeret efter **LINE**.

Du får fejlrapport **"4 Out of memory"** (på dansk: ikke mere plads i hukommelsen), hvis der ikke er plads til det nye program og variabler. I så fald slettes de gamle informationer ikke.

2. Bytes:

LOAD navn **CODE** start, længde

Hvis der er flere bytes (navn) på bånd end "længde" – bytes, så får du fejlrapport R. Ellers indkøres de i hukommelsen med første adresse i "start", og de tidligere slettes.

LOAD navn **CODE** start

indkører bytene "navn" fra adresse "start" og sletter det tidligere.

LOAD navn **CODE**

indkører bytene "navn" fra den første adresse på båndet og sletter alle tidligere bytes i hukommelsen på disse adresser.

3. V. sæt:

LOAD navn **DATA** bogstav ()

eller

LOAD navn **CODE** bogstav $ ()

sletter alle v. sæt kaldet "bogstav" eller "bogstav $" og indsætter et ny v. sæt fra bånd.

Fejlrapport **"4 Out of memory"** skrives, hvis der ikke er plads til nye v. sæt. Gamle v. sæt i så fald slettes ikke.

MERGE

Er som **LOAD**, men sletter ikke informationer i hukommelsen.

1. Programmer og variabler:

MERGE navn

indkører programmet "navn" oveni det gamle program og sletter kun programlinjer og variabler fra det gamle program, hvis disse har samme linjenummer eller navn. Fejlrapport **"4 Out of memory"** hvis der ikke er plads til både det nye og gamle program og variablerne.

2. V. sæt:

Ikke muligt

3. Bytes:

Ikke muligt

Øvelser

1. Lav et bånd, hvor det første program skriver en "menu" (en liste over de andre programmer på båndet), beder dig vælge et program og derefter **LOAD**er det, du ønsker.

2. Indtast eller **LOAD** skakbrik-programmet fra kapitel 23 og tast så **NEW**. Skakbrikkerne vil stadig ligge i datamatens hukommelse. Det gør de ikke, hvis du slukker for maskinen.

Hvis du ønsker at gemme dem, må du optage dem på bånd ved brug **SAVE** og **CODE**. Det nemmeste er at gemme alle 21 brugerdefinérbare grafiktegn sådan:

SAVE "skak" CODE USR "a",21*8

eftertulgt af

VERIFY "skak" CODE

Det er den metode, der blev brugt til at gemme et skærmbillede med.

Adressen på den første byte, der skal gemmes, er **USR "a"** – adressen på den første af de otte bytes, som bestemmer mønstret for de første brugerdefinérbare grafiktegn – og antal bytes, der skal gemmes, er 21*8 – otte bytes for hvert af de 21 grafiktegn.

For at køre grafiktegnene ind i maskinen igen kan du bruge dette:

LOAD "SKAK" CODE

Hvis du evt. **LOAD**er det tilbage til en Spectrum med en anden størrelse hukommelse, eller hvis du har flyttet den brugerdefinérbare grafik til en anden adresse (det kræver rot avanceret viden), skal du være mere forsigtig og bruge:

LOAD "skak" CODE USR "a"

USR tillader, at grafikken flyttes til en anden adresse.

30

30. ZX Skriveren

Hvis du har anskaffet en skriver til din ZX Spectrum, har du en betjeningsvejledning med. Derfor er dette kapitel ikke nogen brugsanvisning til skriveren, men fortæller om de BASIC-kommandoer, man skal bruge, hvis man vil have lidt glæde af den.

De første to, **LPRINT** og **LLIST**, er nøjagtig ligesom **PRINT** og **LIST** – selvfølgelig bortset fra, at de bruger skriveren i stedet for skærmen.

Dengang man opfandt BASIC (ca. 20 år efter, at man havde opfundet datamaten), fik man ikke resultaterne skrevet ud på en tv-skærm, men på en slags skrivemaskine. Derfor betyder **PRINT**: at trykke – ligesom på engelsk. L i **LPRINT** kommer af, at den skrivemaskine man brugte, undertiden kaldtes en *linjeskriver.*

Prøv dette her:

```
10 LPRINT "Dette program:".
20 LLIST
30 LPRINT "skriver karaktersaettet."
40 FOR n=32 TO 255
50 LPRINT CHR$ n;
60 NEXT n
```

En tredje kommando, **COPY**, udskriver en kopi af skærmen på skriveren. **LIST** f.eks. det foregående program på skærmen og skriv

```
COPY
```

Bemærk, at **COPY** ikke virker sammen med datamatens automatiske listninger, fordi disse bliver slettet eller nulstillet, når en kommando udføres.

Det er altid muligt at stoppe skriveren, mens den kører, ved at taste **BREAK** (**CAPS SHIFT** og **SPACE**).

Såfremt du udfører disse kommandoer *uden* skriveren, vil datamaten nok 'labe' udprintningen og gå videre til næste udtryk.

Prøv dette:

```
10 FOR n=31 TO 0 STEP -1
20 PRINT AT 31-n,n; CHR$ (CODE"0"+n);
30 NEXT n
```

Du vil se et mønster af bogstaver, der fra højre bevæger sig diagonalt ned over skærmen, indtil det når underkanten af skærmen. Da spørger programmet **scroll**?

Prøv nu at ændre **AT 31-n,n** i linje 20 til **TAB n**. Programmet vil gøre nøjagtig det samme som før.

Udskift så **PRINT** i linje 20 med **LPRINT**. Denne gang vil maskinen ikke spørge **scroll**? (der ikke bruges ved printer), og programmet vil fortsætte med bogstaverne F og O.

Til sidst vil vi foreslå, at du ændrer **TAB n** til **AT 31-n,n**, mens du stadig benytter **LPRINT**. Denne gang får du kun en enkelt linje fyldt med karakterer. Årsagen til dette

fænomen er, at det, der skal skrives fra **LPRINT**, ikke skrives med det samme, men gemmes væk i et stødpudelager, også kaldet en *buffer*.

Udskrivning af indholdet i denne buffer sker først

1. når bufferen er fuld,
2. efter en **LPRINT**-sætning, der ikke ender med komma eller semikolon,
3. når et komma, en apostrof eller **TAB** kræver en ny linje,
4. ved slutningen af et program, hvis der stadig er ting tilbage i bufferen, der skal udskrives.

Punkt 3. forklarer, hvorfor vort program med **TAB** opfører sig, som det gør. Med hensyn til **AT**, så ignoreres linjenummeret, og **LPRINT**- positionen ændres til kun at omfatte kolonnenummeret. En **PRINT AT**- sætning kan aldrig sende en linje til skriveren.

Øvelse

1. Lav en **SIN**usgraf på skriveren ved at bruge programmet i kapitel 28 og **COPY**-kommandoen.

31

31. Andre tilslutningsenheder

Der er andre enheder, som du har mulighed for at slutte til din ZX Spectrum.

ZX Microdrive er en slags hurtig storlagerenhed, som er langt mere fleksibel i sin brug end en båndoptager. Den opererer ikke alene med kommandoerne **SAVE, VERIFY, LOAD** og **MERGE**, men også med **PRINT, LIST, INPUT** og **INKEY$**.

Du kan sammenkoble flere Spectrummer, så de kan snakke med hinanden – en af fordelene er, at de så kun behøver ét fælles Microdrive.

RS232 interfacet er en standardenhed, som giver dig mulighed for at koble ZX Spectrum til andre tastaturer, skrivere, datamater og forskellige andre maskinenheder, selv om de ikke er specielt beregnet til Spectrum.

Dertil skal der bruges nogle ekstra kommandoer, som også findes på ZX Spectrums tastatur, men ikke kan bruges uden de omtalte enheder. Kommandoerne er: **OPEN #, CLOSE #, MOVE, ERASE, CAT** og **FORMAT**.

32

32. **Ind** og **ud**

Mikroprocessoren i ZX Spectrum kan læse fra og (i det mindste med RAMen) skrive til hukommelsen ved at bruge **PEEK** og **POKE**. Selve mikroprocessoren er ligeglad med om hukommelsen er ROM, RAM eller ingenting. Den véd bare, at der er 65536 lageradresser, og den kan udlæse en byte fra hver af dem (selv hvis det er noget vrøvl) og indlæse en byte i hver af dem (selv hvis den går tabt).

På lignende måde er der 65536 af noget, der kalder I/O porte (ind/ud-porte). Disse bruges af mikroprocessoren til at kommunikere med ting som tastaturet eller skriveren, og de kan styres fra BASICen ved hjælp af **IN**-funktionen og **OUT**-kommandoen.

IN er en funktion som **PEEK**

IN adresse

Den har ét argument, nemlig port-adressen, og dens resultat er en byte læst fra den pågældende port. **OUT** er en kommando som **POKE**

OUT adresse, værdi

skriver en given værdi til en port med en given adresse. Hvordan adressen bliver fortolket, afhænger meget af resten af datamaten: ofte vil mange forskellige adresser betyde det samme. På ZX Spectrum er det mest fornuftigt at forestille sig adresserne skrevet binært, fordi de enkelte bits har en tendens til at arbejde uafhængigt af hinanden. Der er 16 bits, som vi kalder (A for adresse):

A15, A14, A13, A12,.....A2, A1, A**Ø**

Her er A**Ø** enerbiten, A1 toerbiten, A2 firerbiten, o.s.v.

Bitsene A**Ø**, A1, A2, A3 og A4 er de vigtigste. De er normalt 1, men hvis en af dem er **Ø**, fortæller det datamaten, at den skal gøre noget bestemt. Maskinen kan ikke klare mere end én ting af gangen, så kun en af de fem bits bør være **Ø**. Bitsene A5, A6 og A7 ignoreres, så hvis du er dygtig til elektronik, kan du selv anvende disse adresser. Det er bedst at anvende de adresser, som er 1 mindre end et multipla af 32, så adresserne A**Ø**,...A4 er alle 1. Bits A8, A9 o.s.v. bruges sommetider til at give ekstra informationer.

En byte, der ind- eller ud-læses, består af 8 bits, og disse kaldes ofte for D7, D6,...D1, D**Ø** (D for data). Her er en liste over de anvendte port-adresser.

Der findes et sæt "input"-adresser, som aflæser tastaturet og EAR-stikket.

Tastaturet er opdelt i 8 halve rækker med 5 taster hver:

IN 65278 aflæser den halve række **CAPS SHIFT** til **V**
IN 65Ø22 aflæser den halve række **A** til **G**
IN 64510 aflæser den halve række **Q** til **T**
IN 63486 aflæser den halve række **1** til **5**
IN 61438 aflæser den halve række **Ø** til **6**
IN 57342 aflæser den halve række **P** til **7**
IN 49150 aflæser den halve række **ENTER** til **H**
IN 32766 aflæser den halve række **SPACE** til **B**

(Disse adresser er 254+256*(255—2↑n), hvor n går fra Ø til 7).

I den indlæste byte står bits DØ til D4 for de fem taster i en given række – DØ for den yderste tast og D4 for tasten nærmest midten. Biten er Ø, hvis tasten nedtrykkes, 1 hvis ikke. D6 er værdien for EAR-stikket.

Portadresse 254 driver højttaleren (D4) og MIC-stikket (D3) og bestemmer også kantfarven (D2, D1 og DØ).

Portadresse 251 kører skriveren både i indlæsning og udlæsning: indlæsning finder ud af, om skriveren er klar til at modtage mere, og udlæsning sender prikker, som skal skrives.

Portadresserne 254, 247 og 239 bruges til de ekstra enheder, nævnt i kapitel 31.

Kør dette program:

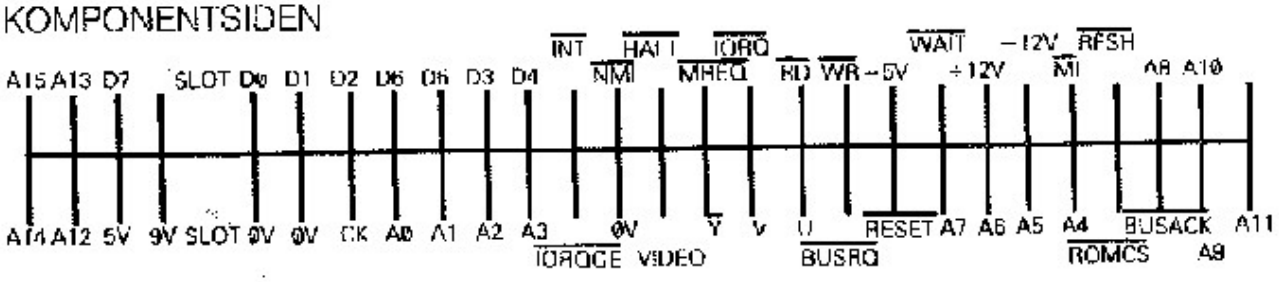
```
10 FOR n=0 TO 7: REM halvraekke tal
20 LET a=254+256*(255-2↑n)
30 PRINT AT 0,0; IN a: GO TO 30
```

og tryk på forskellige taster. Når du keder dig, så pres **BREAK** og tast

```
NEXT n
```

Kontrol-, data- og adressebusserne er alle ført ud på Spectrums bagside, så du kan gøre næsten alt det med en Spectrum, som du kan gøre med en Z80. Undertiden kommer Spectrums udstyr dog i vejen for dette. Her er et diagram over forbindelserne på bagsiden:

KOMPONENTSIDEN



UNDERSIDEN

33

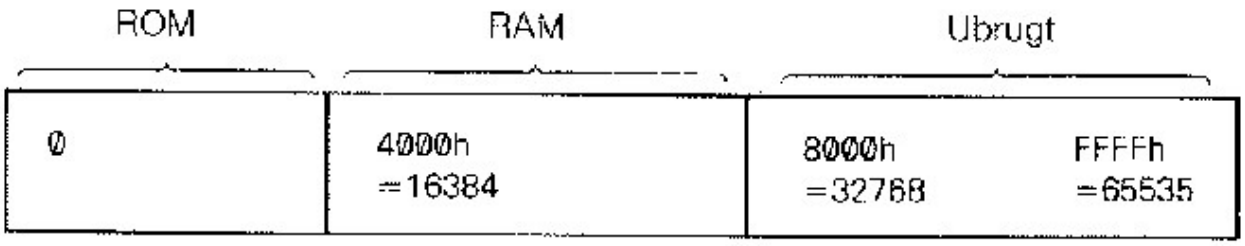
33. Hukommelsen

Inde i datamaten er alle data lagret som bytes, d.v.s. som tal mellem 0 og 255. Du troede måske, at du havde lagret prisen på benzin, din læges telefonnummer eller lign., men det er altsammen omdannet til serier af bytes, og bytes er alt, hvad datamaten ser.

Hver plads, hvor en byte kan lagres, har en adresse, som er tal mellem 0 og FFFFh (så en adresse kan gemmes som to bytes). Du kan forestille dig datamatens lager som en lang række af nummererede kasser, hver med plads til en byte.

Der er forskel på kasserne, nogle er åbne, nogle er låste.

I 16K standard-udgaven er der ingen kasser mellem 8000h og FFFFh. Kasserne fra 4000h til 7FFFh er RAM-kasser, hvilket betyder, at du kan åbne låget og både se og ændre indholdet. Kasserne fra 0 til 3FFFh er ROM-kasser, låste, men med glaslåg, som du blot kan se ned i, men ikke røre. Indholdet kan læses, men har været uforandret, siden din Spectrum blev skabt.



For at se indholdet i en kasse bruger vi **PEEK**-funktionen. Argumentet er en adresse på en kasse, og resultatet er dens indhold. Hvis du f.eks. ønsker at se indholdet i de 21 første bytes i ROMen (og deres adresser), så indlast dette program:

```
10 PRINT "Adresse"; TAB 8; "Byte"
20 FOR a=0 TO 20
30 PRINT a; TAB 8; PEEK a
40 NEXT a
```

Alle disse bytes vil sandsynligvis være meningsløse for dig, men mikroprocessoren opfatter dem som instruktioner, der fortæller den, hvad den skal gøre.

For at ændre indholdet i en kasse (i RAMen) bruger vi **POKE**-kommandoen. Den tager formen:

POKE adresse, ny værdi

hvor "adresse" og "ny værdi" står for numeriske udtryk.

Hvis du f.eks. taster:

POKE 31000,57

vil byten på adressen 31000 antage værdien 57.

Du kan kontrollere dette ved at skrive:

PRINT PEEK 31000

Prøv at **POKE**e andre værdier mellem −255 og +255, (hvis værdien er negativ, bliver der lagt 256 til). **POKE**-kommandoen giver dig kolossal magt over maskinen, hvis du véd, hvordan du skal bruge den; og store ødelæggelsesevner, hvis du ikke véd det. Det er let nok! Du kan f.eks. **POKE** en forkert værdi ind i en forkert adresse, så taber du store programmer og mange timers arbejde, men maskinen tager ikke skade.

Vi vil nu prøve at beskrive RAMen i detaljer, men hvis det ikke interesserer dig, så spring det over.

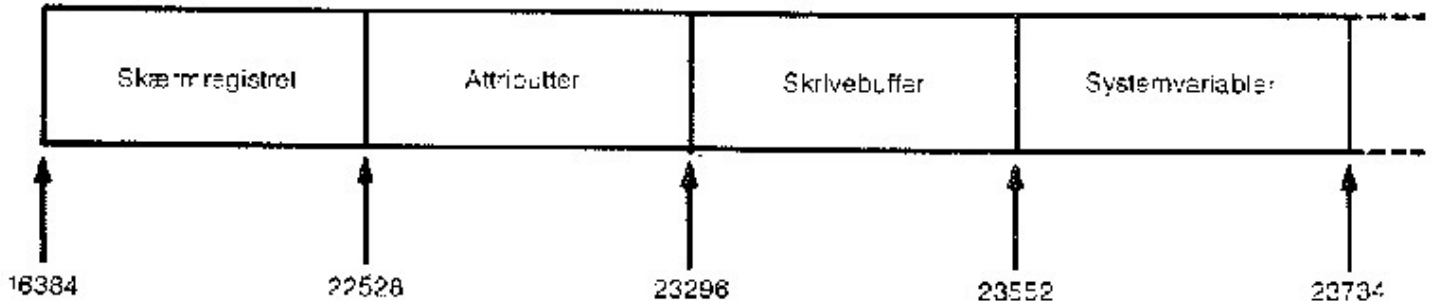
Hukommelsen er opdelt i forskellige områder (vist på det store diagram) til lagring af forskellige slags informationer. Områderne er kun så store, at de netop kan rumme den aktuelle datamængde. Hvis du tilføjer ekstra datamængder i et givet område, f.eks. ved at skrive en ekstra linje eller tilføje nogle variabler, så gøres der plads til dem ved at flytte alle områder opad fra den blok, der skal bruges. Omvendt flyttes alt nedad, hvis data slettes.

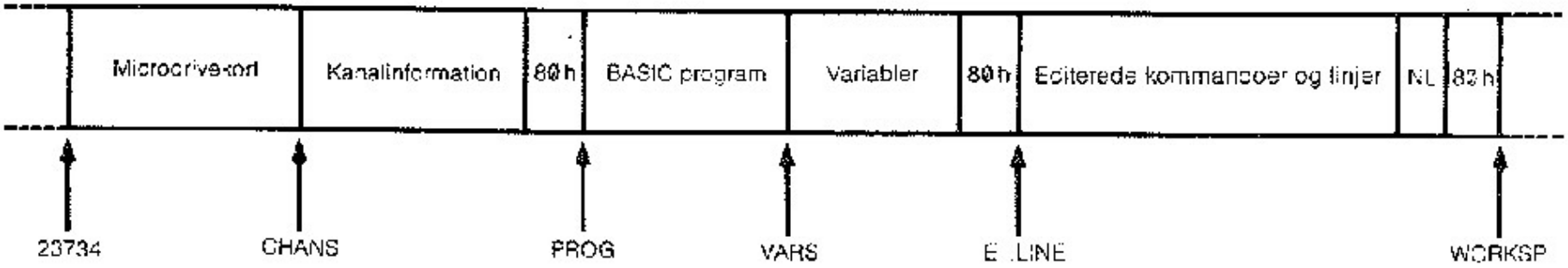
Skærmregistret gemmer skærmbilledet. Det er temmelig specielt opbygget, så du bør ikke **PEEK**e eller **POKE** i registret. Hver karakterposition på skærmen består af et 8*8 kvadrat af punkter, og hvert punkt kan være 0 (paper) eller 1 (ink). Ved at bruge binær notation kan vi lagre mønstret som 8 bytes, en for hver række. Disse 8 bytes lagres ikke sammen. Hver række punkter i en linje med 32 karakterer lagres som 32 bytes i serie, fordi elektronstrålen i fjernsynet behøver det, når den "scanner" fra venstre side af skærmen til den anden. Da hele billedet har 24 linjer, hver af 8 "scan"-rækker, forventer du sikkert, at det hele lagres som 172 "scan"-rækker i rækkefølgen, den ene efter den anden, men her tager du fejl. Først kommer den øverste "scan"-række fra linjerne 0 til 7 og derefter den næste "scan"-række fra linjerne 0 til 7, o.s.v. til den nederste "scan"-række fra linjerne 0 til 7. Dernæst det samme fra linjerne 8 til 15 og det samme fra linjerne 16 til 23. Følgen af alt dette er, at hvis du er vant til en datamat, som bruger **PEEK** og **POKE** på skærmen, så må du her lære at anvende **SCREEN$** og **PRINT AT** eller **PLOT** og **POINT** i stedet.

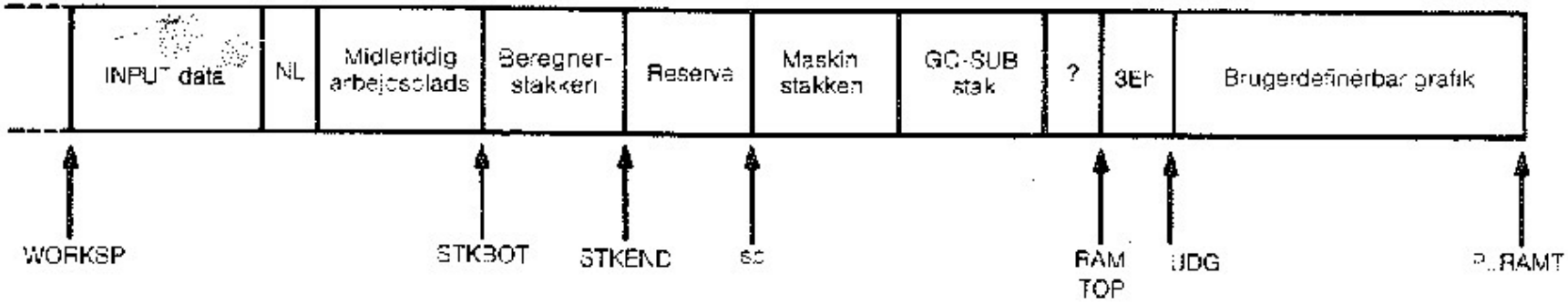
Attributterne er farverne o.s.v. for hver karakterposition, som bruger formatet **ATTR**. Disse er lagret linje efter linje i den rækkefølge, du forventer.

Skrive-bufferen lagrer de karakterer, som er bestemt for skriveren.

System-variablerne indeholder forskellige informationer, som fortæller maskinen noget om den tilstand, den befinder sig i. I det næste kapitel er der en fuldstændig oversigt over systemvariablerne, men i øjeblikket skal du bare bemærke, at der er nogle (kaldet CHANS, PROG, VARS, E_LINE, o.s.v.), som indeholder adresserne på skellene mellem de forskellige områder i hukommelsen. Det er *ikke* BASIC-variabler, og deres navne kan ikke genkendes af maskinen.


Skærmregistret
Attributter
Skrivebuffer
Systemvariabler
16384
22528
23296
23552
23734

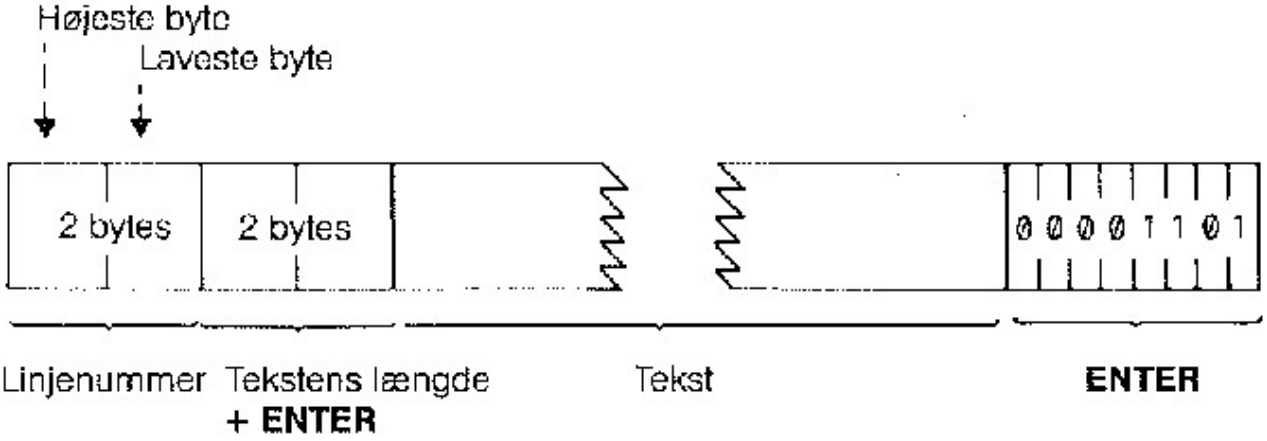

Microdrivekort
Kanalinformation
80h
BASIC program
Variabler
80h
Editerede kommandoer og linjer
NL
82h
23734
CHANS
PROG
VARS
E_LINE
WORKSP


INPUT data
NL
Midlertidig arbejdsplads
Beregner-stakken
Reserve
Maskin stakken
GO-SUB stak
?
3Eh
Brugerdefinerbar grafik
WORKSP
STKBOT
STKEND
SP
RAM TOP
UDG
P_RAMT

Microdrive-afsnittet kan kun bruges sammen med et Microdrive, og normalt er der ikke noget der.

Kanal-informationen indeholder informationer om "input" og "output"-enheder, specielt tastaturet, (med den nederste del af skærmen), den øverste del af skærmen og skriveren.

Hver linje i et BASIC-program tager formen:

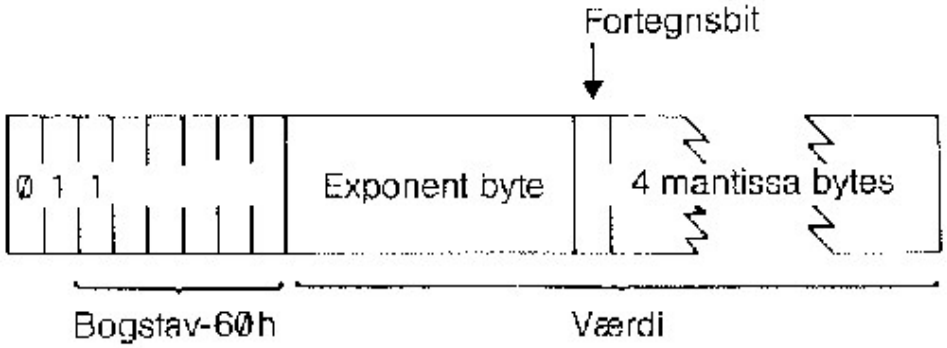


Ligesom i ti-talssystemet, hvor det ciffer, der har mest betydning for tallets størrelse, står til venstre, er den "højeste" byte her til venstre for den "laveste". Dette gælder kun for linjenumre og ikke for alle andre to-bytes tal i Z80.

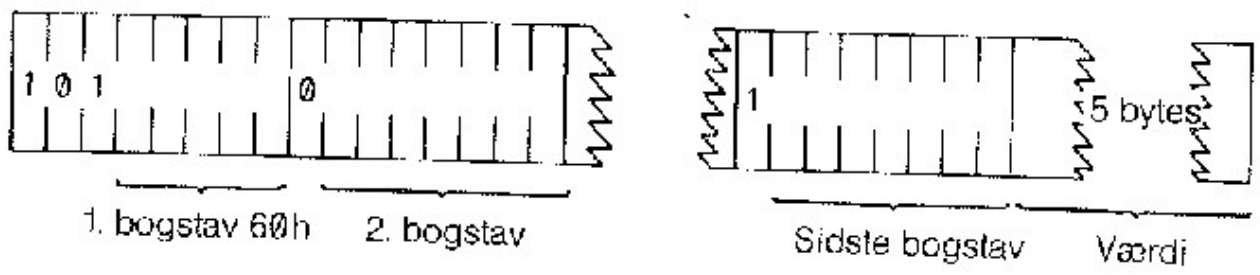
En tal-konstant er lagret på den måde, at først kommer **CHR$ 14** og derefter selve værdien i fire bytes.

Variabler har forskellige formater alt efter deres natur. Bogstaver i navne bliver opfattet som små bogstaver.

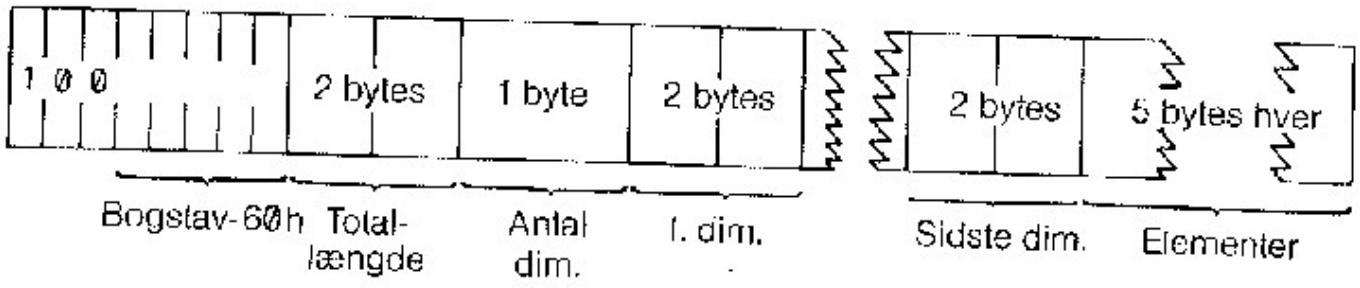
Variabel med kun et bogstav i navnet:



Variabel med flere bogstaver i navnet:



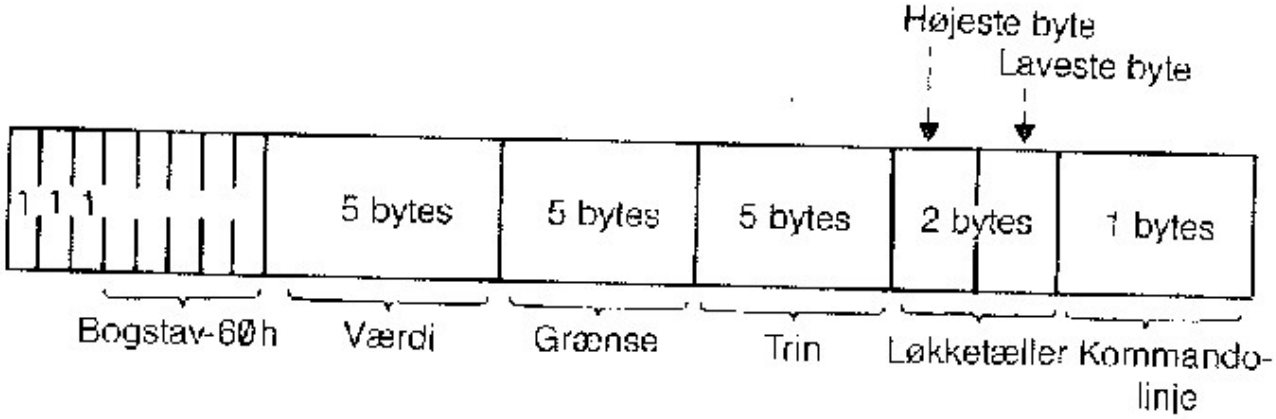
Indiceret talvariabelsæt:



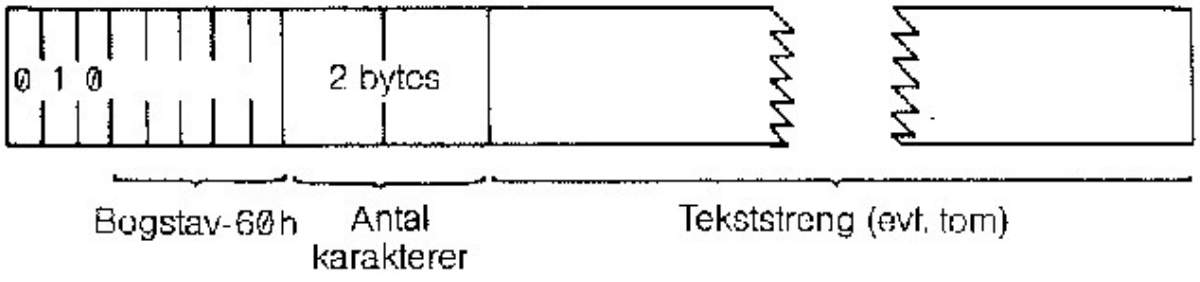
Elementerne i et variabelsæt er ordnet efter dette system:
Først elementer med første indices 1,
så elementer med første indices 2, o.s.v.
Dernæst ordnes efter andet indices efter samme mønster o.s.v.

Som eksempel er her systemet for b(2,3):
b(1,1), b(1,2), b(1,3), b(2,1), b(2,2), b(2,3).

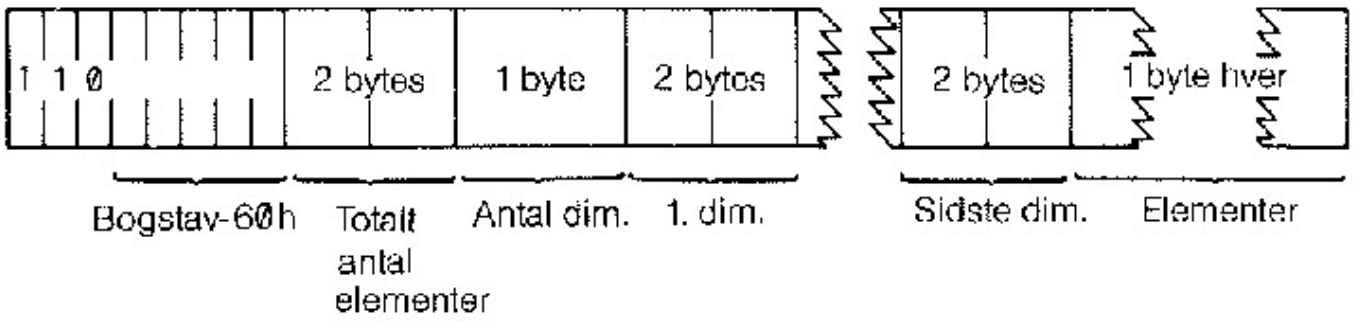
Kontrolvariablen for en **FOR-NEXT** løkke:



Strengvariabel:



Strengvariabelsæt:



Beregneren er den del af BASIC-systemet, som har at gøre med aritmetik, og de tal, den behandler, lagres hovedsagelig i beregnerstakken.

Reserven indeholder hukommelse, som endnu ikke er udnyttet.

Maskinstakken er den lagerplads, der bruges af Z80 processoren til opbevaring af returadresser m.v.

GO SUB-stakken blev nævnt i kapitel 14.

Den højeste adresse, BASIC-systemet bruger, er den, som RAMTOP "peger" på. Selv **NEW**, som renser RAMen, går kun op til RAMTOP, så kommandoen ændrer ikke den brugerdefinérbare grafik. Du kan ændre RAMTOP ved at efterfølge **CLEAR**-kommandoen med et tal:

CLEAR ny RAMTOP

Dette:
1. Sletter alle variabler,
2. renser skærmregistret (som **CLS**),
3. nulstiller **PLOT**-positionen til at starte i nederste venstre hjørne,
4. udfører kommandoen **RESTORE**,
5. renser **GO SUB**-stakken og placerer den lige før den nye RAMTOP – forudsat at dette ligger mellem beregnerstakken og den fysiske ende af RAMen. Ellers lader den RAMTOP være, hvor den var.

RUN udfører også kommandoen **CLEAR**, men ændrer ikke RAMTOP.

Ved at bruge **CLEAR** på denne måde kan du flytte RAMTOP op, så der bliver mere

plads til BASICen. Derved slettes den brugerdefinérbare grafik. Du kan også flytte RAMTOP ned, så der er en ekstra RAM-reserve, der ikke slettes af **NEW**.

Tast **NEW** og dernæst **CLEAR 23800** for at få en fornemmelse af, hvad der sker, når maskinen bliver fyldt.

En af de første ting, du vil bemærke, når du indtaster et program, er, at efter et stykke tid vil datamaten ikke acceptere mere og "knurrer" ad dig. Det betyder, at maskinen er fyldt op, og det er nødvendigt at lænse (tømme) den lidt. Der er også to fejlmeddelelser med nogenlunde samme betydning, nemlig **4 Memory full** (på dansk: hukommelsen fyldt), og **G No room for line** (på dansk: Ikke plads til ny linje).

Datamaten knurrer også, hvis du forsøger at indtaste en linje, der er længere end 23 linjer. Det indtastede ignoreres ikke, men du kan bare ikke se det. Datamaten knurrer for at advare dig mod at indtaste mere.

Du kan ændre den tid, datamaten knurrer, ved at **POKE** adresse 23608. Når maskinen tændes, er værdien 64.

Ethvert tal (undtagen 0) kan skrives sådan:

$\pm m \times 2^e$

hvor ± er fortegnet.

m er mantissen og ligger mellem ½ og 1 (kan ikke være 1),

og

e er eksponenten, et helt tal (evt. negativt).

Antag, at du vil skrive m binært. Da det er en brøkdel, skal det have et binært komma (som et komma i et decimaltal), og så en binær brøk (som en decimal brøk).

Tal skrevet binært:

En halv skrives som .1

En kvart skrives som .01

Trekvart skrives som .11

En tiendedel skrives som .0001100110011001100110011.. o.s.v.

Ved vores tal m, fordi det er mindre end 1, er der ingen bits før det binære komma, og fordi det mindst er en ½, er biten umiddelbart efter det binære komma et 1.

For at lagre et tal i datamaten bruger vi fem bytes, på følgende måde:

1. Skriver de første otte bits i mantissen i den anden byte (vi ved at den første bit er 1), de næste otte bits i den tredje byte, de næste igen i den fjerde byte, og de sidste otte i byte fem.
2. Udskifter den første bit i den anden byte - som vi ved er 1 – med fortegnet: 0 for plus, 1 for minus.
3. Skriver eksponenten +128 i den første byte.

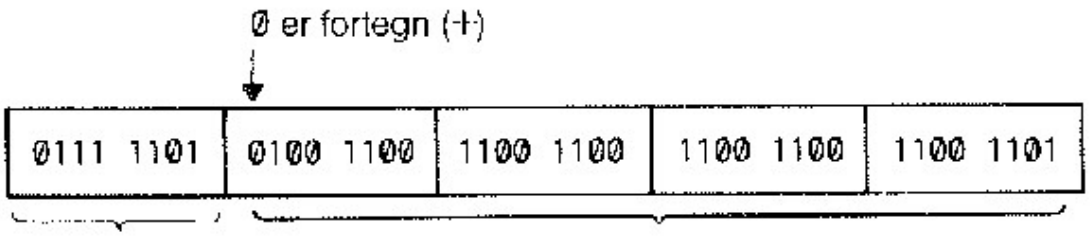
Hvis vi antager, at vort tal f.eks. er 1/10

$$1/10 = 4/5 \times 2^{-3}$$

Derfor er mantissen m:

.11001100110011001100110011001100 i binært (da den 33. bit er 1, skal vi runde den 32. op fra 0 til 1), og eksponenten e er −3.

Ved at lægge vores tre regler sammen, får vi:



Der er en anden måde at lagre tal mellem —65535 og +65535:
1. Den første byte er Ø.
2. Den anden byte er Ø for et positivt tal, FFh for et negativt.
3. Den tredje og fjerde byte er den mindst og den mest betydende byte (eller tallet + 131072, hvis det er negativt).
4. Den femte byte er Ø.

34

34. Systemvariablerne

Hukommelsesområdet fra 23552 til 23733 er reserveret til brug for systemet. Du kan **PEEK**e dem og få rede på forskellige ting om systemets tilstand, og nogle af dem kan det være nyttigt at ændre med **POKE**.

Systemvariablerne, som de kaldes, er vist i det følgende sammen med en forklaring på deres brug og indhold. Du må endelig ikke forveksle dem med de ord og variabler, der bruges i BASIC. Datamaten kender dem ikke ved disse navne. De har kun fået navne, for at vi mennesker bedre skal kunne huske dem.

Forkortelserne i kolonne 1 har denne betydning:

X Variablen bør ikke ændres, idet systemet så kan bryde sammen.
N En ændring af variablen vil ikke have nogen blivende virkning.

Tallet i kolonne 1 er antallet af bytes i variablen. Når der er to bytes, er den første den laveste, måske det omvendte af, hvad du ville forvente. Så hvis du ønsker at ændre værdien i en 2 bytes variabel på adressen **n** til værdien **v**, skal du benytte denne fremgangsmåde:

```
POKE n,v-256*INT (v/256)
POKE n+1,INT (v/256)
```

og hvis du vil udlæse værdien, så brug udtrykket:

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

Bemærkn.	*Adresse*	*Navn*	*Indhold*
N8	23552	KSTATE	Bruges til at aflæse tastaturet.
N1	23560	LAST K	Fortæller hvilken tast, der sidst er påvirket.
1	23561	REPDEL	Tiden (1/50 sekund) som en tast skal holdes nede, inden den repeterer. Starter med at være 35, men kan **POKE**s til andre værdier.
1	23562	REPPER	Pause (1/50 sekund) mellem hver repetition af en tast: normalt 5.
N2	23563	DEFADD	Adressen på argumenter for brugerdefinérbare funktioner, som bliver behandlet; ellers 0
N1	23565	K DATA	Gemmer den anden byte fra farvekontrollen indtastet fra tastaturet.
N2	23566	TVDATA	Gemmer farve-bytene, **AT** og **TAB** kontrollerne går til skærmen
X38	23568	STRMS	Adresser på kanaler, der er tilsluttet strømmen.
2	23606	CHARS	256 mindre end adressen på karaktersættet (som starter med mellemrum og fortsætter indtil copyright symbolet). Normalt i ROM, men du kan lave dit eget i RAM og lade CHARS pege på det.

Bemærkn.	*Adresse*	*Navn*	*Indhold*
1	23608	RASP	Varighed af advarselstone.
1	23609	PIP	Varighed af tastaturklik.
1	23610	ERR NR	1 mindre end rapportkoden. Starter som 255 (for –1), så **PEEK 23610** giver 255.
X1	23611	FLAGS	Diverse tegngivere til BASIC-systemet.
X1	23612	TV FLAG	Tegngivere til skærmen
X2	23613	ERR SP	Adressen på værdien i maskinstakken, som bruges som "tejlretur".
N2	23615	LIST SP	Returadresse fra automatiske listninger.
N1	23617	MODE	Indholdet specificerer K, L, C, F eller G-markør.
2	23618	NEWPPC	Linje, der skal hoppes til.
1	23620	NSPPC	Kommandonummer i linje, som maskinen skal hoppe til. Ved POKE først NEWPPC og så NSPPC tvinges maskinen til at hoppe til en bestemt kommando i en linje.
2	23621	PPC	Linjenummeret på det linjeudtryk, der i øjeblikket udføres.
1	23623	SUBPPC	Nummeret på den kommando i en linje, der udføres.
1	23624	BORDCR	Kantfarve * 8; indeholder også attributterne, som normalt bruges til den nederste halvdel af skærmen.
2	23625	E PPC	Linjenummeret på den aktuelle linje (med pilemarkør).
X2	23627	VARS	Adresse på variabler.
N2	23629	DEST	Adresse på den arbejdende variabel.
X2	23631	CHANS	Adresse på kanaldata.
X2	23633	CURCHL	Adresse på informationer, som i øjeblikket bruges som "input" og "output".
X2	23635	PROG	Adresse på BASIC-program.
X2	23637	NXTLIN	Adresse på næste linje i programmet.
X2	23639	DATADD	Adresse på sidste **DATA**-element.
X2	23641	E LINE	Adresse på indtastet kommando.
2	23643	K CUR	Adresse på markøren.
X2	23645	CH ADD	Adresse på den næste karakter, der skal oversættes: karakteren efter argumentet til **PEEK** eller **NEWLINE** efter en **POKE**-kommando.
2	23647	X PTR	Adressen på karakteren efter ?-markøren.
X2	23649	WORKSP	Adresse på en midlertidig arbejdsplads.
X2	23651	STKBOT	Adresse på det nederste af beregnerstakken.
X2	23653	STKEND	Adresse på starten af reserveplads.
N1	23655	BREG	Beregnerens b-register.

Bemærkn.	*Adresse*	*Navn*	*Indhold*
N2	23656	MEM	Adresse på det område, der benyttes af beregneren. (Sædvanligvis MEMBOT, men ikke altid).
1	23658	FLAGS2	Flere tegngivere.
X1	23659	DF SZ	Antallet af linjer i skærmens nederste del (inkl. en tom linje).
2	23660	S TOP	Nummeret på den øverste linje i automatiske listninger.
2	23662	OLDPPC	Linjenummeret, **CONTINUE** springer til.
1	23664	OSPCC	Kommandonummeret i en linje, **CONTINUE** springer til.
N1	23665	FLAGX	Forskellige tegngivere.
N2	23666	STRLEN	Længde af den arbejdende streng.
N2	23668	T ADDR	Adresse på den næste enhed i syntakstabellen (kan vist ikke være særlig nyttig).
2	23670	SEED	**RND**s start. Det er denne variabel, der styres af **RANDOMIZE**.
3	23672	FRAMES	3 byte (den mindst betydende først), billedtæller. Forøges med en hvert 20ms. Se kap 27.
2	23675	UDG	Adresse på 1'ste brugerdefinérbare grafiktegn. Du kan ændre den, hvis du f.eks. ønsker at spare plads ved at have færre af disse grafiktegn.
1	23677	COORDS	x-koordinat for det sidste punkt, der er "plottet".
1	23678		y-koordinat for det sidste punkt, der er "plottet".
1	23679	P POSN	33-kolonnetal for skriverposition.
1	23680	PR CC	Den laveste byte i adressen til det næste punkt, der skal **LPRINT**es (i skriverbufferen).
1	23681		Ubrugt.
2	23682	ECHO E	33-kolonnetal og 24 linjetal (i den nederste del) i slutningen af "input-bufferen".
2	23684	DF CC	Adresse i skærmfilen på **PRINT** positionen.
2	23686	DFCCL	Som DF CC for nederste del af skærmen.
X1	23688	S POSN	33-kolonnetal for **PRINT**-positionen.
X1	23689		24-linjetal for **PRINT**-positionen.
X2	23690	SPOSNL	Som S POSN for nederste del.
1	23692	SCR CT	Styrer "scroll": Adressen er altid én mere, end det antal gange, datamaten "scroller", når den spørger **scroll?** Hvis du **POKE**r adressen med et tal større end 1 (f.eks. 255), så vil skærmen scrolle og scrolle uden at spørge dig.
1	23693	ATTR P	Permanente farver (bestemt af farvekommandoer).

Bemærkn.	*Adresse*	*Navn*	*Indhold*
1	23694	MASK P	Bruges til "transparente" farver. Enhver bit, der er 1, viser, at den tilsvarende attribut-bit ikke er taget fra ATTR P, men fra det, der allerede står på skærmen.
N1	23695	ATTR T	Midlertidig aktuelle farver, (bestemt af farve-elementer).
N1	23696	MASK T	Som MASK P, men midlertidig.
1	23697	P FLAG	Flere tegngivere.
N3Ø	23698	MEMBOT	Beregnerens hukommelsesområde. Bliver brugt til at lagre de tal, der ikke på en hensigts-mæssig måde kan lagres i beregnerstakken.
2	23728		Ubrugt.
2	23730	RAMTOP	Adresse på den sidste byte i BASIC-systemets område.
2	23732	P-RAMT	Adresse på den sidste byte i den fysiske RAM.

Dette program udskriver værdien af de første 22 bytes i variabelområdet:

```
10 FOR n=0 TO 21
20 PRINT PEEK (PEEK 16400+256*PEEK 16401+n)
30 NEXT n
```

Prøv om du kan få det til at passe med det, der blev beskrevet ovenfor.
I det program, du netop har lavet, skal du ændre linje 20 til:

```
20 PRINT PEEK (16509+n)
```

Nu fortæller det dig indholdet i de første 22 bytes i programområdet. Prøv at sammenholde det med selve programmet.

35

35. Brug af maskinkode

Dette kapitel er skrevet for dem, der forstår *Z80 maskinkode*, det sæt af instruktioner, som Z80 processoren bruger. Hvis du ikke kender noget til dette sprog, men kunne tænke dig at gøre det, er der mange bøger om det.

Programmerne er normalt skrevet i *assembler sprog*, som umiddelbart kan forekomme gådefuldt, men i praksis ikke er så svært at forstå. (Du kan se Z80 assembler-sprog i Appendiks A.).

For at køre dem på datamaten, må du kode en række sammenhørende bytes. Dette kaldes for *maskinkode*. Du kan køre maskinsprog direkte på ZX Spectrum, men hvis du anvender en assembler (ikke indbygget i Spectrum), oversætter ZX Spectrum informationerne direkte og indsætter dem i hukommelsen.

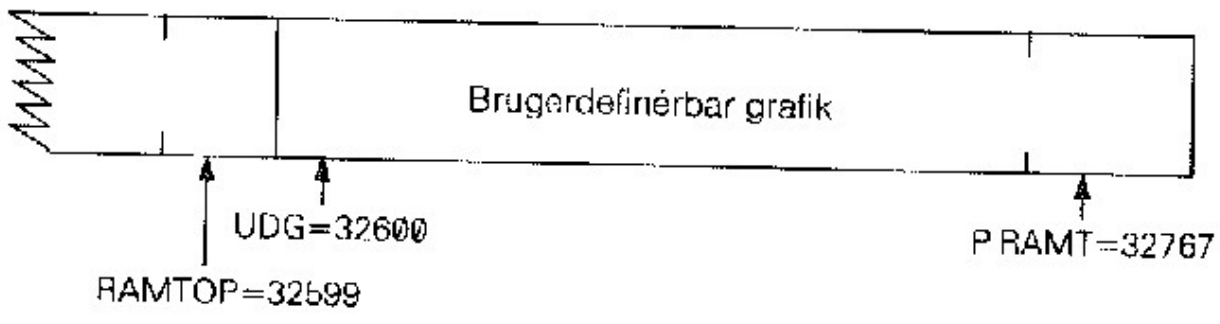
Lad os prøve dette eksempel:

```
ld bc, 99
ret
```

som loader bc registerparret med 99. Dette oversættes til fire maskinkode bytes 1, 99, 0 (for ld bc, 99) og 201 (for ret). (Hvis du ser på 1 og 201 i Appendiks A, vil du finde ld bc,NN – hvor NN står for ethvert to-byte tal – og ret).

Nu har du sammensat et maskinkodeprogram, og det skal nu ind i datamaten. (En assembler ville kunne gøre dette automatisk). Du skal bestemme, hvor i hukommelsen det skal placeres, og det bedste er at lave plads mellem BASIC-området og den brugerdefinérbare grafik.

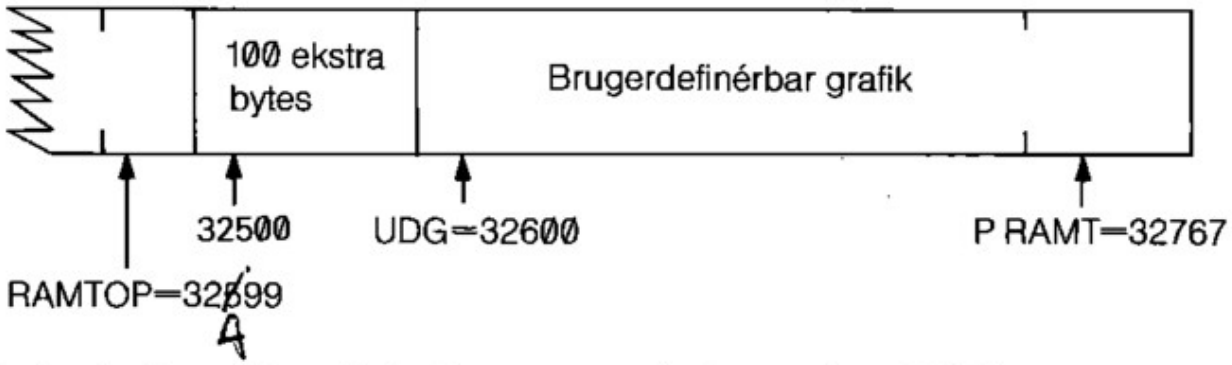
Antag f.eks., at du har en 16K Spectrum. Når datamaten tændes, så ser den øverste ende af RAMen sådan ud:



Hvis du taster:

CLEAR 32499

vil du få 100 byte ekstra plads, med første adresse på 32500.



Du kan indkøre dit maskinkodeprogram ved at anvende et BASIC-program som

```
10 LET a=32500
20 READ n: POKE a,n
30 LET a=a+1: GO TO 20
40 DATA 1,99,0,201
```

Programmet vil stoppe med rapport **E Out of DATA**, når de fire bytes er fyldt.

For at køre maskinkode skal du bruge funktionen **USR,** men denne gang med et numerisk argument, nemlig startadressen. Resultatet er værdien af bc registret, så hvis du taster:

```
PRINT USR 32500
```

får du svaret 99.

Returadressen til BASIC bliver stakket som sædvanlig, så returnering foregår ved en Z80 "ret" instruktion. Du bør ikke bruge iy og i registrene i maskinkoderutiner.

Du kan nemt gemme dine maskinkodeprogrammer ved at skrive:

SAVE "et navn" **CODE 32500,4**

Umiddelbart kan du ikke gemme et maskinkodeprogram, så det automatisk kører sig selv, når det loades tilbage i maskinen, men ved at lave et BASIC-program på følgende måde kan du løse problemet:

```
10 LOAD "" CODE 32500,4
20 PRINT USR 32500
```

Først skal du:

SAVE "et navn" **LINE**

og dernæst:

```
SAVE "xxxx" CODE 32500,4
LOAD "et navn"
```

Dette vil køre BASIC-programmet automatisk, når det loades, og dette program kører så maskinkodeprogrammet.

Karaktersættet

Her viser vi det komplette ZX Spectrum karaktersæt med koder i decimale og hexdecimale tal. Koderne er Z80 maskinkoder og kolonnerne til højre giver "huskeordene" (de mnemotekniske ord), som de forekommer i Z80 Assembler-sproget.

Mange assemblerinstruktioner til Z80 er sammensatte og starter med CBh eller EDh (h for hex). De to kolonner yderst til højre angiver disse instruktioner.

Kode	*Karakter*	*Hex*	*Z80 assembler*	*Efter CBh*	*Efter EDh*
0	ubrugt	00	nop	rlc b	
1		01	ld bc,NN	rlc c	
2		02	ld (bc),a	rlc d	
3		03	inc bc	rlc e	
4		04	inc b	rlc h	
5		05	dec b	rlc l	
6	**PRINT** komma	06	ld b,N	rlc (hl)	
7	**EDIT**	07	rlca	rlc a	
8	markør t.v.	08	ex af,af'	rrc b	
9	markør t.h.	09	add hl,bc	rrc c	
10	markør ned	0A	ld a,(bc)	rrc d	
11	markør op	0B	dec bc	rrc e	
12	**DELETE**	0C	inc c	rrc h	
13	**ENTER**	0D	dec c	rrc l	
14	tal	0E	ld c,N	rrc (hl)	
15	ubrugt	0F	rrca	rrc a	
16	**INK** kontrol	10	djnz DIS	rl b	
17	**PAPER** kontrol	11	ld de, NN	rl c	
18	**FLASH** kontrol	12	ld (de),a	rl d	
19	**BRIGHT** kontrol	13	inc de	rl e	
20	**INVERSE** kontrol	14	inc d	rl h	
21	**OVER** kontrol	15	dec d	rl l	
22	**AT** kontrol	16	ld d,N	rl (hl)	
23	**TAB** kontrol	17	rla	rl a	
24	ubrugt	18	jr DIS	rr b	
25		19	add hl,de	rr c	
26		1A	ld a,(de)	rr d	
27		1B	dec de	rr e	
28		1C	inc e	rr h	
29		1D	dec e	rr l	
30		1E	ld e,N	rr (hl)	
31		1F	rra	rr a	
32	mellemrum	20	jr nz,DIS	sla b	
33	!	21	ld hl,NN	sla c	
34	"	22	ld (NN),hl	sla d	
35	#	23	inc hl	sla e	
36	$	24	inc h	sla h	

37	%	25	dec h	sla l	
38	&	26	ld h,N	sla (hl)	
39	'	27	daa	sla a	
40	(	28	jr z,DIS	sra b	
41	)	29	add hl,hl	sra c	
42	*	2A	ld hl,(NN)	sra d	
43	+	2B	dec hl	sra e	
44	,	2C	inc l	sra h	
45	-	2D	dec l	sra l	
46	.	2E	ld l,N	sra (hl)	
47	/	2F	cpl	sra a	
48	0	30	jr nc,DIS		
49	1	31	ld sp,NN		
50	2	32	ld (NN),a		
51	3	33	inc sp		
52	4	34	inc (hl)		
53	5	35	dec (hl)		
54	6	36	ld (hl),N		
55	7	37	scf		
56	8	38	jr c,DIS	srl b	
57	9	39	add hl,sp	srl c	
58	:	3A	ld a,(NN)	srl d	
59	;	3B	dec sp	srl e	
60	<	3C	inc a	srl h	
61	=	3D	dec a	srl l	
62	>	3E	ld a,N	srl (hl)	
63	?	3F	ccf	srl a	
64	@	40	ld b,b	bit 0,b	in b,(c)
65	A	41	ld b,c	bit 0,c	out (c),b
66	B	42	ld b,d	bit 0,d	sbc hl,bc
67	C	43	ld b,e	bit 0,e	ld (NN),bc
68	D	44	ld b,h	bit 0,h	neg
69	E	45	ld b,l	bit 0,l	retn
70	F	46	ld b,(hl)	bit 0,(hl)	im 0
71	G	47	ld b,a	bit 0,a	ld i,a
72	H	48	ld c,b	bit 1,b	in c,(c)
73	I	49	ld c,c	bit 1,c	out (c),c
74	J	4A	ld c,d	bit 1,d	adc hl,bc
75	K	4B	ld c,e	bit 1,e	ld bc,(NN)
76	L	4C	ld c,h	bit 1,h	
77	M	4D	ld c,l	bit 1,l	reti
78	N	4E	ld c,(hl)	bit 1,(hl)	
79	O	4F	ld c,a	bit 1,a	ld r,a
80	P	50	ld d,b	bit 2,b	in d,(c)
81	Q	51	ld d,c	bit 2,c	out (c),d
82	R	52	ld d,d	bit 2,d	sbc hl,de

83	S	53	ld d,e	bit 2,e	ld (NN),de
84	T	54	ld d,h	bit 2,h	
85	U	55	ld d,l	bit 2,l	
86	V	56	ld d,(hl)	bit 2,(hl)	im 1
87	W	57	ld d,a	bit 2,a	ld a,i
88	X	58	ld e,b	bit 3,b	in e,(c)
89	Y	59	ld e,c	bit 3,c	out (c),e
90	Z	5A	ld e,d	bit 3,d	adc hl,de
91	[	5B	ld e,e	bit 3,e	ld de,(NN)
92	/	5C	ld e,h	bit 3,h	
93	]	5D	ld e,l	bit 3,l	
94	↑	5F	ld e,(hl)	bit 3,(hl)	im 2
95	_	5F	ld e,a	bit 3,a	ld a,r
96	£	60	ld h,b	bit 4,b	in h,(c)
97	a	61	ld h,c	bit 4,c	out (c),h
98	b	62	ld h,d	bit 4,d	sbc hl,hl
99	c	63	ld h,e	bit 4,e	ld (NN),hl
100	d	64	ld h,h	bit 4,h	
101	e	65	ld h,l	bit 4,l	
102	f	66	ld h,(hl)	bit 4,(hl)	
103	g	67	ld h,a	bit 4,a	rrd
104	h	68	ld l,b	bit 5,b	in l,(c)
105	i	69	ld l,c	bit 5,c	out (c),l
106	j	6A	ld l,d	bit 5,d	adc hl,hl
107	k	6B	ld l,e	bit 5,e	ld hl,(NN)
108	l	6C	ld l,h	bit 5,h	
109	m	6D	ld l,l	bit 5,l	
110	n	6E	ld l,(hl)	bit 5,(hl)	
111	o	6F	ld l,a	bit 5,a	rld
112	p	70	ld (hl),b	bit 6,b	in f,(c)
113	q	71	ld (hl),c	bit 6,c	
114	r	72	ld (hl),d	bit 6,d	sbc hl,sp
115	s	73	ld (hl),e	bit 6,e	ld (NN),sp
116	t	74	ld (hl),h	bit 6,h	
117	u	75	ld (hl),l	bit 6,l	
118	v	76	halt	bit 6,(hl)	
119	w	77	ld (hl),a	bit 6,a	
120	x	78	ld a,b	bit 7,b	in a,(c)
121	y	79	ld a,c	bit 7,c	out (c),a
122	z	7A	ld a,d	bit 7,d	adc hl,sp
123	{	7B	ld a,e	bit 7,e	ld sp,(NN)
124	|	7C	ld a,h	bit 7,h	
125	}	7D	ld a,l	bit 7,l	
126	~	7E	ld a,(hl)	bit 7,(hl)	
127	©	7F	ld a,a	bit 7,a	

Kode	*Karakter*		*Hex*	*Z80 assembler*	*efter CBh*	*efter EDh*
128	▯		80	add a,b	res 0,b	
129	▗		81	add a,c	res 0,c	
130	▖		82	add a,d	res 0,d	
131	▄		83	add a,e	res 0,e	
132	▝		84	add a,h	res 0,h	
133	▐		85	add a,l	res 0,l	
134	▘		86	add a,(hl)	res 0,(hl)	
135	▚		87	add a,a	res 0,a	
136	▙		88	adc a,b	res 1,b	
137	▀		89	adc a,c	res 1,c	
138	▜		8A	adc a,d	res 1,d	
139	▟		8B	adc a,e	res 1,e	
140	▛		8C	adc a,h	res 1,h	
141	▞		8D	adc a,l	res 1,l	
142	▌		8E	adc a,(hl)	res 1,(hl)	
143	■		8F	adc a,a	res 1,a	
144	(a)		90	sub b	res 2,b	
145	(b)		91	sub c	res 2,c	
146	(c)		92	sub d	res 2,d	
147	(d)		93	sub e	res 2,e	
148	(e)		94	sub h	res 2,h	
149	(f)		95	sub l	res 2,l	
150	(g)		96	sub (hl)	res 2,(hl)	
151	(h)		97	sub a	res 2,a	
152	(i)		98	sbc a,b	res 3,b	
153	(j)		99	sbc a,c	res 3,c	
154	(k)	user graphics	9A	sbc a,d	res 3,d	
155	(l)		9B	sbc a,e	res 3,e	
156	(m)		9C	sbc a,h	res 3,h	
157	(n)		9D	sbc a,l	res 3,l	
158	(o)		9E	sbc a,(hl)	res 3,(hl)	
159	(p)		9F	sbc a,a	res 3,a	
160	(q)		A0	and b	res 4,b	ldi
161	(r)		A1	and c	res 4,c	cpi
162	(s)		A2	and d	res 4,d	ini
163	(t)		A3	and e	res 4,e	outi
164	(u)		A4	and h	res 4,h	
165	**RND**		A5	and l	res 4,l	
166	**INKEY$**		A6	and (hl)	res 4,(hl)	
167	**PI**		A7	and a	res 4,a	
168	**FN**		A8	xor b	res 5,b	ldd
169	**POINT**		A9	xor c	res 5,c	cpd
170	**SCREEN$**		AA	xor d	res 5,d	ind
171	**ATTR**		AB	xor e	res 5,e	outd
172	**AT**		AC	xor h	res 5,h	

Kode	*Karakter*	*Hex*	*Z80 assembler*	*efter CBh*	*efter EDh*
173	**TAB**	AD	xor l	res 5,l	
174	**VAL$**	AE	xor (hl)	res 5,(hl)	
175	**CODE**	AF	xor a	res 5,a	
176	**VAL**	B0	or b	res 6,b	ldir
177	**LEN**	B1	or c	res 6,c	cpir
178	**SIN**	B2	or d	res 6,d	inir
179	**COS**	B3	or e	res 6,e	otir
180	**TAN**	B4	or h	res 6,h	
181	**ASN**	B5	or l	res 6,l	
182	**ACS**	B6	or (hl)	res 6,(hl)	
183	**ATN**	B7	or a	res 6,a	
184	**LN**	B8	cp b	res 7,b	lddr
185	**EXP**	B9	cp c	res 7,c	cpdr
186	**INT**	BA	cp d	res 7,d	indr
187	**SQR**	BB	cp e	res 7,e	otdr
188	**SGN**	BC	cp h	res 7,h	
189	**ABS**	BD	cp l	res 7,l	
190	**PEEK**	BE	cp (hl)	res 7,(hl)	
191	**IN**	BF	cp a	res 7,a	
192	**USR**	C0	rct nz	set 0,b	
193	**STR$**	C1	pop bc	set 0,c	
194	**CHR$**	C2	jp nz,NN	set 0,d	
195	**NOT**	C3	jp NN	set 0,e	
196	**BIN**	C4	call nz,NN	set 0,h	
197	**OR**	C5	push bc	set 0,l	
198	**AND**	C6	add a,N	set 0,(hl)	
199	<=	C7	rst 0	set 0,a	
200	>=	C8	ret z	set 1,b	
201	<>	C9	rct	set 1,c	
202	**LINE**	CA	jp z,NN	set 1,d	
203	**THEN**	CB		set 1,e	
204	**TO**	CC	call z,NN	set 1,h	
205	**STEP**	CD	calf NN	set 1,l	
206	**DEF FN**	CE	adc a,N	set 1,(hl)	
207	**CAT**	CF	rst 8	set 1,a	
208	**FORMAT**	D0	ret nc	set 2,b	
209	**MOVE**	D1	pop de	set 2,c	
210	**ERASE**	D2	jp nc,NN	set 2,d	
211	**OPEN #**	D3	out (N),a	set 2,e	
212	**CLOSE #**	D4	call nc,NN	set 2,h	
213	**MERGE**	D5	push de	set 2,l	
214	**VERIFY**	D6	sub N	set 2,(hl)	
215	**BEEP**	D7	rst 16	set 2,a	
216	**CIRCLE**	D8	ret c	set 3,b	
217	**INK**	D9	exx	set 3,c	

Kode	*Karakter*	*Hex*	*Z80 assembler*	*efter CBh*	*efter EDh*
218	**PAPER**	DA	jp c,NN	set 3,d	
219	**FLASH**	DB	in a,(N)	set 3,e	
220	**BRIGHT**	DC	call c,NN	set 3,h	
221	**INVERSE**	DD	prefixes instructions using ix	set 3,l	
222	**OVER**	DE	sbc a,N	set 3,(hl)	
223	**OUT**	DF	rst 24	set 3,a	
224	**LPRINT**	E0	ret po	set 4,b	
225	**LLIST**	E1	pop hl	set 4,c	
226	**STOP**	E2	jp po,NN	set 4,d	
227	**READ**	E3	ex (sp),hl	set 4,e	
228	**DATA**	E4	call po,NN	set 4,h	
229	**RESTORE**	E5	push hl	set 4,l	
230	**NEW**	E6	and N	set 4,(hl)	
231	**BORDER**	E7	rst 32	set 4,a	
232	**CONTINUE**	E8	ret pe	set 5,b	
233	**DIM**	E9	jp (hl)	set 5,c	
234	**REM**	EA	jp pe,NN	set 5,d	
235	**FOR**	EB	ex de,hl	set 5,e	
236	**GO TO**	EC	call pe,NN	set 5,h	
237	**GO SUB**	ED		set 5,l	
238	**INPUT**	EE	xor N	set 5,(hl)	
239	**LOAD**	EF	rst 40	set 5,a	
240	**LIST**	F0	ret p	set 6,b	
241	**LET**	F1	pop af	set 6,c	
242	**PAUSE**	F2	jp p,NN	set 6,d	
243	**NEXT**	F3	di	set 6,e	
244	**POKE**	F4	call p,NN	set 6,h	
245	**PRINT**	F5	push af	set 6,l	
246	**PLOT**	F6	or N	set 6,(hl)	
247	**RUN**	F7	rst 48	set 6,a	
248	**SAVE**	F8	ret m	set 7,b	
249	**RANDOMIZE**	F9	ld sp,hl	set 7,c	
250	**IF**	FA	jp m,NN	set 7,d	
251	**CLS**	FB	ei	set 7,e	
252	**DRAW**	FC	call m,NN	set 7,h	
253	**CLEAR**	FD	prefixes instructions using iy	set 7,l	
254	**RETURN**	FE	cp N	set 7,(hl)	
255	**COPY**	FF	rst 56	set 7,a	

Rapporter

Rapporter bliver udskrevet i den nederste del af skærmen, når maskinen stopper af en eller anden årsag. Rapporten forklarer kort, hvorfor datamaten stoppede.

Rapporten har en tal- eller bogstavkode som henviser til dette appendiks.

CONTINUEs funktion er afhængig af, hvilken rapport datamaten har udskrevet. Normalt hopper **CONTINUE** til den linje og kommando, hvor den stoppede, men hvis rapport-koderne er 0, 9 eller D, så fungerer **CONTINUE** anderledes (se appendiks C).

Hvis du ønsker en mere udførlig forklaring på maskinens udskrevne rapport, bør du se i appendiks C.

Kode	*Betydning*	*Situationer*
0	OK Heldig afslutning eller et forsøg på at hoppe til et linienummer, der er større end noget, der findes.	Alle
1	NEXT without FOR. Kontrolvariablen er ikke blevet skabt af en FOR sætning (findes ikke) men der er en almindelig variabel med det samme navn.	**NEXT**
2	Variabel not found. En udefineret variabel er blevet brugt. Det vil ske for en simpel variabel, hvis den benyttes, inden den har optrådt i en **LET, READ** eller **INPUT**-kommando, eller er loadet fra bånd. For en indiceret variabel vil det ske, hvis den benyttes, inden den er dimensioneret i en **DIM** kommando eller loadet fra bånd.	Enhver
3	Subscript wrong Indices større end dimensionering, eller er forkert. Hvis indices er større end 65535 eller er negativ, så fejl B.	Indicerede variabler delstrenge.
4	Out of memory Ikke plads i hukommelsen. Hvis maskinen er kørt fast, er du nødsaget til at slette en eller flere linjer og/eller bruge kommandoen **CLEAR**, så variablerne ikke optager plads i hukommelsen.	**LET**, **INPUT**, **FOR**, **DIM**, **GO SUB**, **LOAD**, **MERGE**, til tider ved beregninger.
5	Out of screen Ikke mere plads på skærmen! En **INPUT**-kommando har forsøgt at skrive mere end 23 linjer i den nederste del af skærmen. Også fejl 5 hvis **PRINT AT 22**,...	**INPUT**, **PRINT AT**

Kode	*Betydning*	*Situationer*
6	Number too big For stort tal. Beregninger har ført til tal større end 10^{38}.	Al aritmetik.
7	RETURN without GO SUB **RETURN** uden **GO SUB**. Flere **RETURN** end **GO SUB**er.	**RETURN**
8	End of file	Microdrive og lignende
9	STOP statement **STOP** kommando udført. Efter dette vil **CONTINUE** ikke gentage **STOP**, men fortsætte med næste kommando.	**STOP**
A	Invalid argument Forkert argument. Af en eller anden årsag er argumentet forkert.	**SQR**, **LN**, **ASN**, **ACS**, **USR**, (med streng-argument).
B	Integer out of range Heltal uden for definitionsområdet. Når BASIC behøver et helt tal, men får en variabel, der ikke er hel, vil tallet blive afrundet til nærmeste tal. Hvis dette tal er uden for definitionsområdet, så fejl B.	**RUN**, **RANDOMIZE**, **POKE**, **DIM**, **GO TO**, **GO SUB**, **LIST**, **LLIST**, **PAUSE**, **PLOT**, **CHR$**, **PEEK**, **USR** (med et numerisk argument)
	For indicerede variabler, fejl 3	Indicerede variabler.
C	Nonsense in BASIC Forkert i BASIC. Teksten i (streng) argumentet er ikke gyldig.	**VAL**, **VAL$**
D	BREAK-CONT repeats **BREAK** er tastet, mens maskinen udførte et eller andet. **CONTINUE** gentager udtrykket. Se rapport L.	**LOAD**, **SAVE**, **VERIFY**, **MERGE**, **LPRINT**, **LLIST**, **COPY**. Også når maskinen spørger **"scroll?"** og du taster **N**, **SPACE** eller **STOP**
E	Out of DATA Du har anvendt **READ** efter enden af DATA-**listen.**	**READ**
F	Invalid file name **SAVE**s navn er tomt eller længere end 10 karakterer.	**SAVE**
G	No room for line Ikke plads til ny linje, pga. af manglende hukommelsesplads.	Indtastning af en linje i et program

Kode	*Betydning*	*Situationer*
H	STOP in INPUT En **INPUT**-kommando har fået en **STOP**-kommando (hvis **INPUT LINE**, **STOP** er tastet). **CONTINUE** vil repetere **INPUT**-kommandoen.	**INPUT**
I	FOR without NEXT **FOR** uden **NEXT**. Der findes ikke en **NEXT**-kommando til **FOR**-sætningen.	**FOR**
J	Invalid I/O	Microdrive og lign.
K	Invalid colour Tallet er ikke anvendeligt.	**INK**, **PAPER**, **BORDER**, **FLASH**, **BRIGHT**, **INVERSE**, **OVER**. også ved brug af de tilsvarende kontrolkarakterer.
L	BREAK into program **BREAK**-tasten er aktiveret mellem to kommandoer. Linje- og kommando-tallene i rapporten, refererer til kommandoen, før **BREAK** blev aktiveret, men **CONTINUE** fortsætter med kommandoen efter, så ingen kommandoer repeteres.	Alle
M	RAMTOP no good Det tal, som er specificeret for RAMTOP, er for stort eller for lille.	**CLEAR** (undertiden i **RUN**).
N	Statement lost Ingen kommando. Hopper til en kommando, som ikke længer eksisterer.	**RETURN**, **NEXT**, **CONTINUE**.
O	Invalid stream	Microdrive og lign.
P	FN without DEF Brugerdefinérbar funktion.	**FN**
Q	Parameter error Parameterfejl. Forkert antal argumenter eller forkert type (streng i stedet for tal og omvendt).	**FN**
R	Tape loading error Bånd-loadningsfejl. Informationer er fundet på bånd, men af en eller anden grund kan de ikke indlæses eller undersøges.	**VERIFY**, **LOAD** eller **MERGE**.

Beskrivelse af ZX Spectrum

Den første del af dette appendiks repeterer introduktionen til tastaturet og skærmen.

Tastaturet:
ZX Spectrums karakterer omfatter ikke bare enkelte symboler (bogstaver, tal og lign.), men også hele "ord" (kommandoer, funktionsnavne m.m.) og alle disse kan kun indtastes fra tastaturet. For at få plads til det, har visse taster helt op til fem eller flere betydninger, der kan være "den gældende" alt efter, hvilken tast man trykker samtidig med en skiftetast (**CAPS SHIFT** eller **SYMBOL SHIFT** eller dem begge på samme tid), eller hvilken "tilstand", datamaten befinder sig i.

Tilstanden angives af en *markør*, et blinkende bogstav, der viser, hvor den næste karakter indtastet fra tastaturet vil blive placeret.

K-tilstand (for Keyword eller nøgleord) fremkommer automatisk, når maskinen forventer en kommando (og ikke **INPUT**-data) eller et linjenummer.

D.v.s. at der er **K**-markør ved linjestart eller lige efter **THEN** og lige efter **:** (undtagen i en streng).

L-tilstand (for letters eller bogstaver) er som regel til stede på alle andre tidspunkter. Såfremt der ikke skiftes (**CAPS SHIFT** eller **SYMBOL SHIFT**), vil hovedkarakteren på den pågældende tast være den karakter, der bliver skrevet på skærmen (små bogstaver, hvis det er en bogstav-tast).

Både ved **K**- og **L**-tilstand vil **SYMBOL SHIFT** plus en eller anden tast blive udlagt som det røde tegn på tasten, og **CAPS SHIFT** og en af tal-tasterne vil blive udlagt som kontrolfunktionen, der er skrevet med hvidt over tal-tasterne. **CAPS SHIFT** i **L**-tilstand ændrer små bogstaver til store bogstaver.

C-tilstand (for Capitals eller "store bogstaver") er en variant af **L**-tilstand, hvor alle bogstaver ændres til store bogstaver. Du kan "låse" **C**-markøren ved at taste **CAPS LOCK** (det samme, hvis du vil tilbage til **L**-tilstand).

E-tilstand anvendes, når du skal have fat i de røde og de grønne "ord" over og under tasterne. **E**-tilstand får du ved at trykke på begge "shift"-tasterne samtidigt. Når **E**-markøren viser sig på skærmen, kan du gøre to ting: hvis du trykker på en af tasterne, får du det grønne "ord" på tastaturet, og hvis du holder en af "shift"-tasterne nede samtidig med, at du trykker ned på en af tasterne, får du det røde "ord" under den pågældende tast.

En taltast giver et rødt "ord", hvis **SYMBOL SHIFT** holdes nede samtidig med, at tasten aktiveres. Ellers er taltasterne farvestyring.

G-tilstand (for Graphics eller grafik) kommer frem ved at trykke **CAPS SHIFT** og **9**. Den vedvarer, indtil der tastes **9** igen. En taltast vil give grafiktegnene på tasterne og tast **0** vil virke som **DELETE**-tast.

Bogstav-tasterne (dog ikke V, W, X, Y og Z) giver et brugerdefinérbart grafiktegn.

Hvis en tast holdes nede 2 eller 3 sekunder, begynder den at repetere.

"Input" fra tastaturet sker i den nederste del af skærmen, enten som enkelte tegn eller som kommandoord m.v. og skrives lige før markøren. Den kan flyttes til venstre ved hjælp af venstre-pilen (**CAPS SHIFT** og **5**) og til højre med højre-pilen (**CAPS SHIFT** og **8**).

Karakteren foran markøren kan slettes med **DELETE** (**CAPS SHIFT** og **0**). Hele

linjen kan slettes ved at taste **EDIT** (**CAPS SHIFT** og **1**) efterfulgt af **ENTER**. Når der tastes **ENTER**, udføres linjen, indføres i programmet eller bruges som **INPUT**-data – medmindre den indeholder en syntaksfejl. I dette tilfælde vil maskinen melde sig med et blinkende **K**, umiddelbart foran fejlen.

Efterhånden som programlinjer indføres i programmet, vil der blive dannet en programliste på den øverste del af skærmen. Den måde, hvorpå denne liste dannes, er ret kompliceret og forklares noget nøjere i kapitel 2. Den sidste linje, der er indført, kaldes den *aktuelle linje* og markeres med tegnet ▶, men dette ændres ved at benytte op-og nedpilene (**CAPS SHIFT 7** og **6**). Hvis der tastes **EDIT** (CAPS SHIFT og **1**), bliver den aktuelle linje bragt ned i skærmens nederste halvdel, hvor den kan rettes.

Når en kommando er udført eller et program kørt, vises udskriften i skærmens øverste del, hvor den forbliver, indtil en ny programlinje tastes ind, eller der trykkes på **ENTER** med en "tom linje". Endelig forsvinder den også, hvis der trykkes på op- eller nedpilene. I skærmens nederste del udskrives en rapport-kode (tal eller bogstav), som henviser til Appendiks B. Dette efterfølges at en kort tekst og to tal adskilt at kolon: det første tal er den linje, hvor programmet stoppede, og det andet fortæller, i hvilken kommando det stoppede.

Rapporten står på skærmen, indtil der foretages en ny indtastning (og viser **K**-tilstand).

Under visse omstændigheder virker mellemrumstangenten (**CAPS SHIFT** og **SPACE**) som **BREAK**, d.v.s. stopper maskinen og udskriver rapport **D** eller **L**. Det sker:

1. Når programmet kører og når til afslutningen af en sætning.
2. Når maskinen bruger skriveren eller båndoptageren.

Skærmen:

Skærmen har 24 linjer, hver 32 karakterer lang, og deles i to dele. Den øverste del er højst 22 linjer og viser enten en programlistning eller udskrift fra et program. Når dette har nået den nederste del af skærmen, vil den **scroll**e (rulle) linjerne op. Såfremt dette indebærer, at du ikke kan nå at se linjerne, før de forsvinder "overnud" af skærmen, vil datamaten automatisk spørge: **scroll?** Tasterne: **N, SPACE** eller **STOP** vil standse programmet med rapport: **D BREAK-CONT repeats**, alle de øvrige taster får maskinen til at **scroll**e videre.

Den nederste del at skærmen bruges til at indtaste kommandoer, programlinjer og **INPUT**-data og også til rapportkoder.

Hvis der indtastes mere end to linjer i den nederste del, rykker linjerne opad, indtil de når den "aktuelle" print-position i den øverste del. Indtastes yderligere, begynder de øverste linjer i den øverste del, at **scroll**e ud af skærmen.

Hver karakter-position har attributter, der specificerer dens farver – paper (baggrund) og ink (forgrund), en to-niveaus lysstyrke og en blinkekontrol. De mulige farver er: sort, blå, rød, lilla, grøn, lyseblå, gul og hvid.

Du kan også bestemme kantfarven med kommandoen **BORDER**.

En karakter-position er opdelt i 8X8 felter (billedpunkter) og højopløsningsgrafik

opnås ved at fastsætte ink- og paper-farverne for hvert billedpunkt på en bestemt karakterposition.

Attributterne til en given karakterposition tilpasses, hver gang en karakter skrives eller et billedpunkt "plottes". Tilpasningen bestemmes af "skriveparametre", hvoraf der er to sæt kaldet "permanente" og "midlertidige" hver på seks parametre: **PAPER, INK, FLASH, BRIGHT, INVERSE** og **OVER.**

Permanente parametre for den øverste del kan altså styres med kommandoerne **PAPER, INK,** o.s.v., og de forbliver de samme, indtil der gives en ny kommando (når datamaten tændes, skrives der med sort på hvidt, der er normal lysstyrke, og der er ingen blinkende karakterer, o.s.v.). Den nederste del bruger "kant"-farven som paper-farve, og ink-farven er enten sort eller hvid afhængig af, hvad paper-farven er. Ligeledes er der normal billedstyrke, ingen blink, "normal video", og der overprintes ikke.

Midlertidige parametre styres også af **PAPER, INK,** o.s.v., men her optræder de i kommandoerne **PRINT, LPRINT, INPUT, PLOT, DRAW** og **CIRCLE** og også i **PAPER, INK,** o.s.v. sammen med kontrolkarakterer, som bruges på skærmen - disse er efterfulgt af en byte, som specificerer parameter-værdien.

Virkningen af disse farvekommandoer varer kun til slutningen af **PRINT**- kommandoen, eller i **INPUT**-kommandoer, indtil der er indtastet noget. Derefter kommer de permanente farveparametre igen.

PAPER- og **INK**-parametrene kan antage værdier mellem 0 og 9. Værdierne 0 til 7 er farverne, som kan anvendes, når en karakter skrives:

0 sort
1 blå
2 rød
3 lilla
4 grøn
5 lyseblå
6 gul
7 hvid

Parameter 8 ("transparent") specificerer, at farven på skærmen forbliver uændret, når en karakter skrives.

Parameter 9 ("kontrast") specificerer, at farven skal være enten sort eller hvid afhængig af, hvad de øvrige farver er.

FLASH og **BRIGHT** kan antage værdierne 0, 1 eller 8. 1 betyder at "blink" og "lysstyrke" er "tændt", 0 at de er slukket, og 8 ("transparent") lader det, der står på en given karakterposition, forblive uændret.

OVER og **INVERSE** kan antage værdierne 0 og 1.

OVER 0 nye karakterer sletter de gamle.

OVER 1 gamle og nye karakterer "forenes", d.v.s. der skrives oveni (overprintning).

INVERS 0 nye karakterer skrives med ink-farve på paper-farve (normal video).

INVERS 1 nye karakterer skrives med paper-farve på ink-farve (invers video).

rot MOD 32 til n_{00}, og derefter skrives tilstrækkeligt med mellemrum, så **PRINT**-positionen flyttes til kolonne n_{00}.

Når en komma-kontrolkarakter modtages, så skrives de nødvendige mellemrum (i det mindste et), så **PRINT**-positionen flyttes til kolonne 0 eller 16.

Når en **ENTER**-kontrolkarakter modtages, flyttes **PRINT**-positionen til den næste linje.

Skriveren:
"Output" til ZX skriveren kommer via en "buffer" på én linje (32 karakterer), og en linje sendes til skriveren:

1. når en linje er fyldt, og en ny påbegyndes.
2. når en **ENTER**-karakter er modtaget.
3. hvis ved slutningen af et program, der er noget uprintet tilbage.
4. når en **TAB**-kontrol eller komma-kontrol flytter **PRINT**-positionen til en ny linje.

TAB-kontrollen og komma-kontrollen laver mellemrum på samme måde som på tv'et.

AT-kontrollen ignorerer linjenummeret og anvender kolonnenummeret til at ændre print-positionen.

Du kan anvende **INVERSE** og **OVER** på samme måde som på skærmen, men ikke kommandoerne **PAPER, INK, FLASH** og **BRIGHT.**

Skriveren vil stoppe med fejlrapport **B**, hvis **BREAK** tastes.

Hvis skriveren ikke tilsluttes, går alt det udlæste tabt.

Basic-sproget

Tal lagres med en præcision på 9 eller 10 cifre. Det største tal, maskinen kan arbejde med, er ca. 10^{38} og det mindste positive tal er ca. $4*10^{-39}$. ZX Spectrum arbejder med aritmetik med flydende komma, og tal lagres på denne form med en byte til eksponenten e ($1<=e<=255$) og fire mantisse bytes m ($½<=m<1$). Det repræsenterer tallet $m*2^{e-128}$.

Da $½<=m<1$, er den højeste bit i mantissen, er m altid 1. Derfor kan det lade sig gøre at skifte denne bit ud med en bit, der viser fortegnet, nemlig – 0 for positive tal, 1 for negative tal.

Små heltal repræsenteres specielt. Den første byte er 0, den anden bestemmer fortegn (0 eller FFh) og den tredje og fjerde er heldelen i to komplement former, den laveste byte først.

Numeriske variabler har navne af vilkårlig længde, der begynder med et bogstav og fortsætter med bogstaver eller tal. Mellemrum og farver ignoreres, og alle bogstaver bliver opfattet som små bogstaver.

Kontrolvariabler i **FOR-NEXT** løkker har navne på ét bogstav.

Numeriske, indicerede variabler skal have navne på ét bogstav, men dette navn på en indiceret variabel må gerne forekomme samtidigt med en simpel variabel af samme navn. De må gerne have uendeligt mange dimensioner, og dimensionerne er uden begrænsninger i størrelsen. Indices starter med 1 som det laveste.

Strengvariabler er fleksible i længden, og navnet på en sådan variabel består af et enkelt bogstav efterfulgt af **$**.

Indicerede strengvariabler kan have uendeligt mange dimensioner og uendelig størrelse, og navnet skal bestå af et enkelt bogstav efterfulgt af et **$**. Navnet må ikke være det samme som navnet på en simpel strengvariabel. Alle variabler i et sæt har den samme konstante længde, som specificeres i form af en ekstra afsluttende dimension i **DIM**-sætningen. Minimumsværdien af indices er 1.

Delstrenge kan specificeres ved at bruge *"slicers"*.

En *"slicer"* kan være:

1. Tom.
2. Et numerisk udtryk.
3. Et numerisk udtryk **TO** et numerisk udtryk,

og bruges til at angive en delstreng ved en af følgende metoder:

a. Strengudtryk ("slicer").
b. Indiceret strengvariabelsæt (indices, indices..., indices, "slicer")

det sidste vil være det samme som:

Indiceret strengvariabelsæt (indices, indices..., indices) ("slicer").

Se i øvrigt kapitel 21.

Funktioner
Argumentet til en funktion behøver ikke parenteser, hvis det er en konstant eller en variabel.

Funktion f(op)	*Operandtype (op)*	*Resultat*
ABS	tal	Absolutte størrelser.
ACS	tal	Arccosinus i radianer.
AND	binær operation, højre operand altid et tal.	
	Numerisk venstre op.:	A **AND** B= { A hvis B <> =Ø ; Ø hvis B=Ø
	Streng venstre op.:	A$ **AND** B= { A$ hvis B <> Ø ; "" hvis B=Ø
ASN	tal	Arcsinus i radianer. Fejlraport A, hvis operand ikke i området –1 til +1.
ATN	tal	Arctangens i radianer.
ATTR	to argumenter, x og y begge tal i parenteser.	Et tal, hvis binære form koder attributterne i linje x og kolonne y på skærmen. Bit 7 (mest betydende) er 1 for blink, Ø for ikke-blink. Bit 6 er 1 for ekstra lysstyrke, Ø for normal. Bit 5 til 3 er paperfarve. Bit 2 til Ø er ink-farve. Fejl B, med mindre Ø<=x<=23 og Ø<=y<=31.
BIN		**BIN** er ikke en rigtig funktion, men en anden måde at skrive et tal på; **BIN** efterfulgt af en række Øer og 1ere er et tal med sådan en fremtrædon i binært.
CHR$	tal	Den karakter, hvis kode er operanden (evt. afrundet til nærmeste heltal.
CODE	streng	Koden for den første karakter i op. (Ø, hvis der er tale om en tom streng).
COS	tal i radianer	Cosinus til op.
EXP	tal	e^{op}
FN		**FN** efterfulgt af et bogstav kalder en brugerdefineret funktion (se **DEF**). Argumentet skal være omsluttet af parenteser, som skal være der, også når der ikke er noget argument.

Funktion f(op)	*Operandtype (op)*	*Resultat*
IN	tal	Resultatet er en byte læst fra port op. (0<=op<=FFFFh) (loader bc-register-parret med op. og udfører maskinsprogsinstruktionen in a (C).
INKEY$	ingen	Aflæser tastaturet. Resultatet er karakteren for den berørte tast i **L**-tilstand eller **C**-tilstand eller 0, hvis ingen eller mere end en tast aktiveres.
INT	tal	Heltalsdelen (runder altid ned).
LEN	streng	Længden af strengen.
LN	tal	Naturlig logaritme (e som grundtal). Fejl A, hvis op. <=0.
NOT	tal	0 hvis op. <> 0, 1 hvis op.=0, **NOT** har prioritet 4.
OR	binær operation begge op. tal	a **OR** b= 1 hvis b <> 0 a hvis b=0 **OR** har prioritet 2.
PEEK	tal	Værdien af den byte i hukommelsen, der har adressen op. (afrundet til nærmeste heltal). Fejl B, hvis op. ikke er i området 0 til 65535.
PI	ingen	π (3,14159265)
POINT	to argumenter, x og y, begge tal i parenteser.	1, hvis billedpunktet i (x,y) har ink-farve. 0 hvis paper-farve. fejl B, medmindre 0<=op<=255 og 0<=175.
RND	ingen	Det næste pseudo-tilfældige tal i en række, der er fremkommet ved at tage forskellige potenser af 75 MOD 65537, trække 1 fra og dividere med 65536. 0<y<1.
SCREEN$	to argumenter, x og y, begge tal i parenteser.	Den karakter i (x,y), normal eller invers, på skærmens x-linje og y-kolonne. Giver en tom streng, hvis karakteren ikke anerkendes. Fejl B, med mindre 0 <= x<=23 og 0<=y<=31.
SGN	tal	Fortegnet til op. –1,0 eller 1.
SIN	tal i radianer	Sinus.
SQR	tal	Kvadratroden. Fejl A, hvis op. er mindre end 0.
STR$	tal	Den karakter, der ville blive vist på skærmen, hvis op. blev udskrevet.

Funktion	*Operandtype*	*Resultat*
TAN	tal i radianer	Tangens
USR	tal	Kalder maskinkodesubrutine, hvis startadresse er op. Ved retur er resultatet indholdet af bc registerparret.
USR	streng	Adresse på bit-mønstret for brugerdefinérbar grafik, svarende til op. Fejl A, hvis op. ikke er et enkelt bogstav mellem a og u eller et brugerdefinérbart grafiktegn.
VAL	streng	Beregner op. (uden anførselstegn) som et numerisk udtryk. Fejl C, hvis op. indeholder syntaksfejl eller er en ikke-numerisk streng. Andre fejl mulige afhængigt af udtrykket.
VAL$	streng	Beregner op. (uden anførselstegn) som et numerisk udtryk. Fejl C, hvis op. indeholder syntaksfejl eller giver en numerisk værdi. Andre fejlmuligheder som ved **VAL**.
–	tal	Negation.

De følgende operationer er binære:

+	addition eller sammenkædning af strenge.	
–	subtraktion.	
*	multiplikation.	
/	division	
↑	opløftning i potens, fejl B, hvis venstre op. er negativ.	
=	lig med	Begge operander skal være af samme type og resultatet er 1, hvis udtrykket er sandt, 0, hvis det er falsk.
>	større end	
<	mindre end	
<=	mindre end eller lig med	
>=	større end eller lig med	
<>	forskellig fra	

Funktioner og operationer har disse prioriteter:

Operation	*Prioritet*
Indikering og **"slicing"**	12
Alle funktioner undt. **NOT** og fortegnsminus	11
↑	10
Fortegnsminus	9
*/	8
+,– (regnetegn)	6

Operation	*Prioritet*
=,>,<,>=,<=,<>	5
NOT	4
AND	3
OR	2

Kommandoer:
I denne liste vil

a	repræsentere et enkelt bogstav.
v	repræsentere en variabel
x,y,z	repræsentere numeriske udtryk
m,n	repræsentere numeriske udtryk afrundet til nærmeste heltal.
u	repræsentere et udtryk
s	repræsentere et strengudtryk
k	repræsentere en række kommandoer, adskilt med kolon
t	repræsentere en række farveelementer, afsluttet af kommaer eller semikoloner. Et farveelement har formen: **PAPER, INK, FLASH, BRIGHT, INVERSE** eller **OVER**.

Alle kommandoer, undtagen **INPUT**, **DEF** og **DATA** kan bruges enten direkte eller i programmer. En kommando eller programlinje kan indeholde flere ordrer, adskilt ved kolon (:) Bortset fra **IF** og **REM** kan kommandoen bruges hvor som helst på en linje.

BEEP x,y
Sender en lyd ud af højttaleren i x sekunder med tonehøjden y omkring det midterste C på toneskalaen (y skal være negativ for toner under C).

BORDER m
Bestemmer kantfarven på skærmen og paper-farven for den nederste del af skærmen.
Fejl K, hvis m ikke ligger mellem 0 og 7.

BRIGHTn
Bestemmer lysstyrken for efterfølgende karakterer, n=0 for normal, 1 for ekstra lysstyrke og 8 for "transparent".
Fejl K hvis n ikke er 0, 1 eller 8.

CAT
Virker ikke uden Microdrive.

CIRCLE x,y,z
Tegner en cirkel med center (x,y) og radius z.

CLEAR
Sletter alle variabler og frigør deres plads. Kommandoen udfører også **RESTORE** og **CLS**, nulstiller **PLOT**-positionen og "renser" **GO SUB**-stakken.

CLEAR n
Som **CLEAR**, men giver mulighed for at ændre RAMTOP til n og placerer den nye **GO SUB**-stak der.

CLOSE #
Virker ikke uden Microdrive.

CLS (Clear Screen). Renser skærmregisteret.

CONTINUE Fortsætter programmet derfra, hvor det stoppede, hvis rapporten er forskellig fra Ø. Hvis rapporten var 9 eller L, så fortsættes med den næste kommando ellers repeteres den kommando, hvori fejlen opstod.

Hvis sidste rapport var i kommandolinjen, så vil **CONTINUE** forsøge at fortsætte kommandoen også ind i en løkke, hvis fejlkoden var Ø:1, give fejlkode Ø, hvis den var Ø:2 eller give fejl N, hvis den var Ø:3 eller større.

CONTINUE hedder **CONT** på tastaturet.

COPY Hvis en skriver er tilsluttet, sender **COPY** en kopi af skærmen til skriveren, ellers gør ordren ingenting. Bemærk, at **COPY** ikke kan bruges til at udskrive automatiske listninger med.

DATA $u_1,u_2,...$ Del af en **DATA**-liste. Skal være i et program.

DEF FN$a(a_1,...a_k)=u$ Brugerdefinérbar funktion: skal være i et program. hver af a og a_1 til a_k, er enten et enkelt bogstav eller et enkelt bogstav efterfulgt af $ for strengargumenter eller resultater

Tager formen: **DEF FN** a()=u, hvis der ikke er noget argument.

DELETE f Virker ikke uden Microdrive.

DIM $a(n_1,...n_k)$ Sletter alle variabelsæt med det samme navn **a** og danner et nyt variabelsæt **a**, med k dimensioner $n_1,...n_k$. Sætter variablernes værdi til Ø.

DIM $a\$(n_1,...n_k)$ Sletter alle sæt eller strengvariabler med navnet **a$** og danner et nyt variabelsæt, med k dimensioner $n_1,...n_k$. Sætter værdien af alle variablerne til "". Sættet består faktisk af en række strengvariabler, alle af længden n_k med k-1 dimension.

Fejl 4, hvis der ikke er plads til dannelse af sættet. Et sæt er ikke defineret, før det har optrådt i en **DIM**-sætning.

DRAW x,y Tegner (**DRAW**) x,y,Ø.

DRAW x,y,z Tegner en linje fra **PLOT**-positionen, ved at flytte x vandret og y lodret og drejer med vinklen z.

Fejl B, hvis det kører udenfor skærmen.

ERASE Virker ikke uden Microdrive.

FLASH Definerer om karakterer skal blinke eller ikke. n=Ø for ikke-blink, n=1 for blink, n=8 for uændret.

FOR a=x **TO** y

FOR a=x **TO** y **STEP** z

FOR a=x **TO** y **STEP** 1.

Sletter enhver simpel variabel **a** og etablerer en kontrolvariabel med værdien x, begrænsning y, trin z. Kontrollerer, at startværdi er større (hvis trin >=Ø) eller mindre (hvis trin < Ø) end grænseværdien. Er den det, springes til den førstkommende linje, hvor **NEXT** a befinder sig først på linjen. Se **NEXT** a.

Fejl 1, hvis der ikke findes **NEXT** a.

Fejl 4, hvis der ikke er plads til kontrolvariablen.

FORMAT s

Virker ikke uden Microdrive.

GO SUB n

Putter linjenummeret fra **GO SUB**-sætningen ind i en stak, ellers som **GO TO** n.

Fejl 4, hvis der ikke er nok **RETURN**-kommandoer.

GO TO n

Hopper til linje n (eller hvis der ikke er en sådan, da til den første linie efter).

IF x **THEN** k

Hvis x er sand (ulig Ø), blive k udført. k er alle kommandoer i linjen. Du kan ikke skrive: **IF** x **THEN** linjenummer.

INK n

Sætter ink-farven (forgrundsfarven) for efterfølgende karakterer. n skal være mellem Ø og 7 for farver, og 8 for "transparent" eller 9 for kontrast. Se "tv-skærm" – Appendiks C (1).

Fejl K, hvis n ikke er i intervallet Ø til 9.

INPUT ...

"..." er en sekvens af **INPUT**-enheder, adskilt som **PRINT**-kommandoer med kommaer, semikoloner eller apostroffer. En **INPUT**-enhed kan være:

1. Enhver **PRINT**-enhed, som ikke begynder med et bogstav.
2. Et variabelnavn, eller
3. **LINE**, og en strengvariabel.

PRINT-enhederne og skilletegnene i (1.) bliver behandlet på samme måde som i en **PRINT**-sætning bortset fra, at det foregår i den nederste del af skærmen. I punkt (2.) stopper datamaten og venter på input af et udtryk fra tastaturet. Variablen bliver tildelt denne værdi. Syntaksfejl i en **INPUT**-sætning giver et blinkende spørgsmålstegn. For strengudtryk er input-bufferen "født" med to anførselstegn (som kan slette om nødvendigt). Hvis den første karakter er **STOP**, stopper programmet med fejlrapport H. (3.) er som (2.), bortset fra, at inputtet bliver opfattet som en streng uden anførselstegn, og **STOP** kommandoen virker ikke. For at stoppe skal du taste ↵ i stedet.

INVERSE n

Kontrollerer, hvordan en række karakterer skrives. Hvis n=Ø, skrives karaktererne normalt med ink-farve på paper-farve.
Hvis n=1 skrives karaktererne "inverst" dvs. paper-farve på ink-farve. Se tv-skærm i Appendiks C (1).
Fejl K, hvis n ikke er Ø eller 1.

LET v=u

Tildeler v værdien u. **LET** kan ikke udelades, og en simpel variabel er udefineret, indtil den indgår i en **LET**, **READ** eller **INPUT**-sætning. Såfremt v er en indiceret strengvariabel eller en delstreng, sker tildelingen procrusteansk for at få u til at passe i v.

LIST

LIST Ø

LIST n

Lister programmet på skærmen og starter med linjen n og gør samtidig denne linje til den aktuelle.

LLIST

LLIST Ø

LLIST n

Som **LIST** n, men lister på skriveren.

LOAD s

Loader program og variabler.

LOAD s **DATA** ()

Loader numeriske variabelsæt.

LOAD s **DATA** $ ()

Loader karakter-variabelsæt.

LOAD s **CODE** m,n

Loader n bytes fra adresse m og frem.

LOAD s **CODE** m

Loader bytes fra startadresse m.

LOAD s **CODE**

Loader bytes tilbage til den adresse, hvor fra de blev gemt. (**SAVE**).

LOAD s **SCREEN$**

LOAD s **CODE** 16384,6912.
Søger efter det rigtige på bånd og loader det, idet de tidligere versioner, samtidig slettes fra hukommelsen. Se kapitel 29.

LPRINT

Som **PRINT**, men bruger skriveren.

MERGE s

Som **LOAD** s men **MERGE** sletter ikke gamle variabler og programlinjer, medmindre det er for at gøre plads for nye med samme linjenummer eller samme variabelnavn.

MOVE s_1,s_2

Virker ikke uden Microdrive.

NEW

Genstarter BASIC-systemet, sletter programmer og variabler op til variablen RAMTOP og lader variablerne UDG, P RAMT, RASP og PIP være.

NEXT a

1. Finder kontrolvariablen a

2. Lægger a til trin-værdien.
3. Hvis værdien har overskredet grænseværdien (evt. negativt), hoppes til næste linje efter **NEXT** a.

Fejl 2, hvis der ikke er en variabel a.

Fejl 1, hvis der er en simpel variabel i stedet for kontrolvariablen.

OPEN #

Virker ikke uden Microdrive.

OUT m,n

OUT tildeler port m byte n (loader bc registerparret med m, a registeret med n og udfører assembliersprogsinstruktionen: out (c),a).

0<=m<=65535, -255<=n<=255, ellers fejl B.

OVER n

Kontrollerer, hvordan en række karakterer skrives oveni andre karakterer. Hvis n=0 slettes tidligere karakterer på den pågældende karakterposition. Hvis n=1 bliver den nye karakter blandet med den gamle og giver ink-farve, hvis én af dem (men ikke begge) havde ink-farve, og paper-farve, hvis de begge er paper eller ink farvede. Se tv-skærm Appendiks C (1). Fejl K, hvis n ikke 0 eller 1.

PAPER n

Som **INK**, men for paper (baggrundsfarve).

PAUSE n

Stopper beregningerne og udskriver skærmregistret i en periode, der svarer til n skærmbilleder (50/sek.), eller indtil en tast trykkes ned.
0<=n<=65535, ellers fejl B. Hvis n=0 er pausen ikke tidsbestemt, men varer, indtil en tast trykkes ned.

PLOT f;m,n

Skriver en ink prik (underkastet **OVER** og **INVERSE**) i billedpunktet m,n, flytter **PLOT**-positionen.

Medmindre farvekommandoen f fortæller andet, vil punktet blive plottet med den permanente ink-farve og de andre (paper-farve, blink og styrke) forbliver uændret.

0<=m<=255, 0<=n<=175, ellers fejl B.

POKE m,n

POKEr adresse m med værdien n (adresse m tildeles værdien n).

0<=m<=65535, -255<=n<=255, ellers fejl B.

PRINT ...

"..." er en sekvens af **PRINT**-enheder, adskilt af kommaer, semikolon eller af apostrof, og de udskrives til skærmregisteret for at blive udskrevet på skærmen. Et semikolon ; mellem to enheder har ingen effekt, det bruges kun til at adskille med. Et komma udlæser komma-kontrolkarakteren og en apostrof udlæser

ENTER-karakteren. Hvis en **PRINT**-sætning ikke slutter med , eller ; følger af sig selv en **ENTER**-karakter.

En **PRINT** enhed kan være:

1. tom.

2. et numerisk udtryk.

Hvis værdien x er mindre end eller lig med 10^{-5} eller større end eller lig med 10^{13}, bliver der ved udskriften benyttet eksponentiel notation. Eksponentdelen begynder med et E efterfulgt af et + eller – og dernæst et eller to cifre. Ellers udskrives x i normal decimal notation med op til 8 betydende cifre og uden overflødige nuller efter kommaet.

0 skrives som et enkelt tal 0.

3. Et strengudtryk.

Kommandoord tastet ind strækkes evt. med et par mellemrum.

Kontrolkarakterer har deres kontroleffekt.

Ukendte karakterer udskrives som ?

4. **AT** m,n.

Udskriver en **AT**-kontrolkarakter efterfulgt af en byte for m (linienummer) og en byte for n (kolonnenummer).

5. **TAB** n.

Udskriver en **TAB**-kontrolkarakter efterfulgt af to bytes for n (mindste betydende først). TABulatorstoppel.

6. Et farve-element, der tager form som en **PAPER**, **INK**, **FLASH**, **BRIGHT**, **INVERSE**, eller **OVER** kommando.

RANDOMIZE

RANDOMIZE 0.

RANDOMIZE n

Fastsætter den systemvariabel, der kaldes SEED og som bruges til at generere den næste værdi af **RND**. Hvis n<>0, så tildeles SEED værdien af n; hvis n=0, så får SEED værdien af en anden systemvariabel, der kaldes FRAMES (den tæller antallet af tv-billeder siden datamaten blev tændt). Herved skulle **RND** kunne blive ret så tilfældig. På tastaturet betyder **RAND**, **RANDOMIZE**.

Fejl B, hvis n ikke er i området 0 til 65535.

READ $v_1,v_2,..v_k$

Udlæser variablerne i **DATA**-listerne, og giver dem navnene $v_1,v_2,..v_k$.

Fejl C, hvis et udtryk er af forkert type.

Fejl E, hvis **READ** "læser" flere variabler end **DATA**-listen indeholder.

REM

Har ingen virkning og kan optræde med alle tegn undtagen **ENTER**. Kolon (:) kan også optræde. Derfor er ingen kommandoer mulige efter en **REM**-kommando.

RESTORE

RESTORE 0.

RESTORE n

Nulstiller **DATA**-"pegepinden", så den peger på det første **DATA**-element i linje n; den næste **READ**-kommando begynder at udlæse elementer fra dette punkt.

RETURN

Tager et linjenummer fra **GO SUB**-stakken, lægger 1 til og hopper til det derved fremkomne linjenummer.

Fejl 7, hvis der ikke er noget linjenummer i stakken.

RUN

RUN 0

RUN n

CLEAR og så **GO TO** n.

SAVE s

Gemmer program og variabler.

SAVE s **LINE**

Gemmer program og variabler, så når de igen loades ind i maskinen, starter det et automatisk hop til linje 1.

SAVE s **LINE** m

Som ovenfor, men starter i stedet automatisk i linje m.

SAVE s **DATA** ()

Gemmer numerisk variabelsæt.

SAVE s **DATA$** ()

Gemmer karakter-variabelsættet $.

SAVE s **CODE** m,n

Gemmer n bytes fra adresse m.

SAVE s **SCREEN$**

SAVE s **CODE** 16384,6912.

Gemmer informationer på bånd under navnet s.

Fejl F, hvis s er tom eller har en længde over 11 (se kapitel 29.)

STOP

Stopper programmet med rapport 9.
CONTINUE vil fortsætte med den efterfølgende kommando.

VERIFY

Som **LOAD** bortset fra, at informationerne ikke loades ind i RAM'en men sammenlignes med dem, der allerede er der.

Fejl R, hvis en eller flere af de sammenlignede bytes ikke har samme værdi.

Programeksempler

Dette appendiks indeholder programmer, som demonstrerer ZX Spectrums evner.

Det første skal have en dato som "input", og leverer derefter ugedagen for denne dato:

```
 10 REM omformer dato til dag
 20 DIM d$(7,6): REM ugedage
 30 FOR n=1 TO 7: READ d$(n): NEXT n
 40 DIM m(12): REM maaneders længde
 50 FOR n=1 TO 12: READ m(n): NEXT n
100 REM indtast dato
110 INPUT "dag?";dag
120 INPUT "måned?";maan
130 INPUT "aar (kun 20. aarhundrede)?";aar
140 IF aar<1901 THEN PRINT "20. aarhundrede begynder i 1901":
    GO TO 100
150 IF aar>2000 THEN PRINT "20. aarhundrede slutter I 2000":
    GO TO 100
160 IF maan<1 THEN GO TO 210
170 IF maan>12 THEN GO TO 210
180 IF aar/4-INT(aar/4)=0 THEN LET maan(2)=29: REM skudaar
190 IF dag>m(maan) THEN PRINT "Denne maaned har kun";
    m(maan);"dage": GO TO 500
200 IF d>0 THEN GO TO 300
210 PRINT "Sludder og vrøvl. Giv mig en rigtig dato"
220 GO TO 500
300 REM omformer dato til antal dage siden aarhundredets
    begyndelse
310 LET y=aar-1901
320 LET I=365*y+INT (y/4): REM antal dage til aarets
    begyndelse
330 FOR n=1 TO maan-1: REM adder tidligere maaneder
340 LET I=I+m(n): NEXT n
350 LET I=I+dag
400 REM omform til ugedag
410 LET I=I-7*INT (I/7)+1
420 PRINT dag;"/";maan;"/";aar
430 FOR n=6 TO 3 STEP -1: REM
440 IF d$(I,n)<>" " THEN GO TO 460
450 NEXT n
460 LET e$=d$(I, TO n)
470 PRINT "er en";d$;"dag"
500 LET m(2)=28: REM Februar
510 INPUT "igen?",a$
```

```
520 IF a$="n" THEN GO TO 540
530 IF a$<>"N" THEN GO TO 100
1000 REM ugedag
1010 DATA "Man", "Tirs", "Ons"
1020 DATA "Tors", "Fre", "Lør", "Søn"
1100 REM maanedslaengde
1110 DATA 31, 28, 31, 30, 31, 30
1120 DATA 31, 31, 30, 31, 30, 31
```

Her er et program, som hedder (pangolins eller "skældyr").

Du tænker på et dyr, og datamaten prøver så at gætte, hvilket dyr det er, ved at stille dig nogle spørgsmål, som kan være enten "ja" eller "nej". Hvis den ikke tidligere har "hørt" om dit dyr, beder den dig om flere spørgsmål, som den kan bruge næste gang, nogen giver den et "nyt" dyr.

```
  5 REM "skaeldyr"
 10 LET nq=100: REM antal spørgsmål og dyr
 15 DIM q$(nq,50): DIM a(nq,2): DIM r$(1)
 20 LET qf=8
 30 FOR n=1 TO qf/2-1
 40 READ q$(n): READ a(n,1): READ a(n,2)
 50 NEXT n
 60 FOR n=n TO qf-1
 70 READ q$(n): NEXT n
100 REM start
110 PRINT "Taenk paa et dyr.", "Tryk på en tast for at fortsaette"
120 PAUSE 0
130 LET c=1: REM første spørgsmål
140 IF a(c,1)=0 THEN GO TO 300
150 LET p$=q$(c): GO SUB 910
160 PRINT "?": GO SUB 1000
170 LET in=1: IF r$="j" THEN GO TO 210
180 IF r$="J" THEN GO TO 210
190 LET in=2: IF r$="n" THEN GO TO 150
200 IF r$<>"N" THEN GO TO 150
210 LET c=a(c,in): GO TO 140
300 REM dyr
310 PRINT "Taenker du paa";
320 LET p$=q$(c): GO SUB 900: PRINT "?"
330 GO SUB 1000
340 IF r$="j" THEN GO TO 400
350 IF r$="J" THEN GO TO 400
360 IF r$="n" THEN GO TO 500
370 IF r$="N" THEN GO TO 500
380 PRINT "Svar mig ordentligt, naar jeg", "taler til dig":
    GO TO 300
```

```
400 REM gaettet det
410 PRINT "Det taenkte jeg nok"
500 REM nyt dyr
510 IF qf>nq-1 THEN PRINT "Jeg er sikker på, at dit dyr
    er meget interessant, men jeg har ikke plads til det nu":
    GO TO 800
520 LET q$(qf)=q$(c): REM flyt gammelt dyr
530 PRINT "Hvad er det så?": INPUT q$(qf=1)
540 PRINT "Giv mig et spørgsmaal, som skelner mellem"
550 LET p$=q$(qf): GO SUB 900: PRINT "og"
560 LET p$=q$(qf+1): GO SUB 900: PRINT ""
570 INPUT s$: LET l=LEN s$
580 IF s$(l)="?" THEN LET l=l-1
590 LET q$(c)=s$(TO l): REM indsaet spørgsmaal
600 PRINT "Hvad er svaret paa"
610 LET p$=q$(qf+1): GO SUB 900: PRINT "?"
620 GO SUB 1000
630 LET in=1: LET io=2: REM svar for nyt og gammelt dyr
640 IF r$="j" THEN GO TO 700
650 IF r$="J" THEN GO TO 700
660 LET in=2: LET io=1
670 IF r$="n" THEN GO TO 700
680 IF r$="N" THEN GO TO 700
690 PRINT "Det nytter ikke noget": GO TO 600
700 REM opdater svar
710 LET a(c,in)=qf+1: LET a(c,io)=qf
720 LET qf=qf+2: REM næste frie dyreplads
730 PRINT "Der narrede du mig"
800 REM igen?
810 PRINT "Vil du have en ny omgang?": GO SUB 1000
820 IF r$="j" THEN GO TO 100
830 IF r$="J" THEN GO TO 100
840 STOP
900 REM ikke for mange mellemrum i print
910 PRINT " ";
915 FOR n=50 TO 1 STEP -1
920 IF p$(n)<>" " THEN GO TO 940
930 NEXT n
940 PRINT p$(TO n);: RETURN
1000 REM faa svar
1010 INPUT r$: IF r$=" " THEN RETURN
1020 LET r$=r$(1): RETURN
2000 REM begynd dyr
2010 DATA "Lever dyret i havet",4,2
2020 DATA "Har det skæl",3,5
2030 DATA "Spiser det myrer",6,7
```

```
2040 DATA "en hval", "en dessert", "et skaldyr", "en myre"
```

Her er et program, der tegner flaget fra ZX Spectrums føderland:

```
  5 REM britisk flag
 10 LET r=2: LET W=7: LET b=1
 20 BORDER 0: PAPER b: INK w: CLS
 30 REM sort forneden
 40 INVERSE 1
 50 FOR n=40 TO 0 STEP -8
 60 PLOT PAPER 0;7,n: DRAW PAPER 0;241,0
 70 NEXT n: INVERSE 0
100 REM de hvide felter tegnes
105 REM St. Georgskorset
110 FOR n=0 TO 7
120 PLOT 104+n,175: DRAW 0,-35
130 PLOT 151-n,175: DRAW 0,-35
140 PLOT 151-n,48: DRAW 0,35
150 PLOT 104+n,48: DRAW 0,35
160 NEXT n
200 FOR n=0 TO 11
210 PLOT 0,139-n: DRAW 111,0
220 PLOT 255,139-n: DRAW -111,0
230 PLOT 255,84+n: DRAW -111,0
240 PLOT 0,84+n: DRAW 111,0
250 NEXT n
300 REM St. Andreaskorset
310 FOR n=1 TO 35
320 PLOT 1+2*n,175-n: DRAW 32,0
330 PLOT 224-2*n,175-n: DRAW 16,0
340 PLOT 254-2*n,48+n: DRAW -32,0
350 PLOT 17+2*n,48+n: DRAW 16,0
360 NEXT n
370 FOR n=0 TO 19
380 PLOT 185+2*n,140+n: DRAW 32,0
390 PLOT 200+2*n,83-n: DRAW 16,0
400 PLOT 39-2*n,83-n: DRAW 32,0
410 PLOT 54-2*n,140+n: DRAW -16,0
420 NEXT n
425 REM de andre smaabidder
430 FOR n+0 TO 15
440 PLOT 255,160+n: DRAW 2*n-30,0
450 PLOT 0,63-n: DRAW 31-2*n,0
460 NEXT n
470 FOR n=0 TO 7
480 PLOT 0,160+n: DRAW 14-2*n,0
```

```
485 PLOT 255,63-n: DRAW 2*n-15,0
490 NEXT n
500 REM røde striber
510 INVERSE 1
520 REM St. Georg
530 FOR n=96 TO 120 STEP 8
540 PLOT PAPER r;7,n: DRAW PAPER r;241,0
550 NEXT n
560 FOR n=112 TO 136 STEP 8
570 PLOT PAPER r;n,168: DRAW PAPER r;0,-113
580 NEXT n
600 REM St. Patrick
610 PLOT PAPER r;170,140: DRAW PAPER r;70,35
620 PLOT PAPER r;179,140: DRAW PAPER r;70,35
630 PLOT PAPER r;199,83: DRAW PAPER r;56,-28
640 PLOT PAPER r;184,83: DRAW PAPER r;70,-35
650 PLOT PAPER r;86,83: DRAW PAPER r;-70,-35
660 PLOT PAPER r;72,83: DRAW PAPER r;-70,-35
670 PLOT PAPER r;56,140: DRAW PAPER r;-56,28
680 PLOT PAPER r;71,140: DRAW PAPER r;-70,35
690 INVERSE 0: PAPER 0: INK 7
```

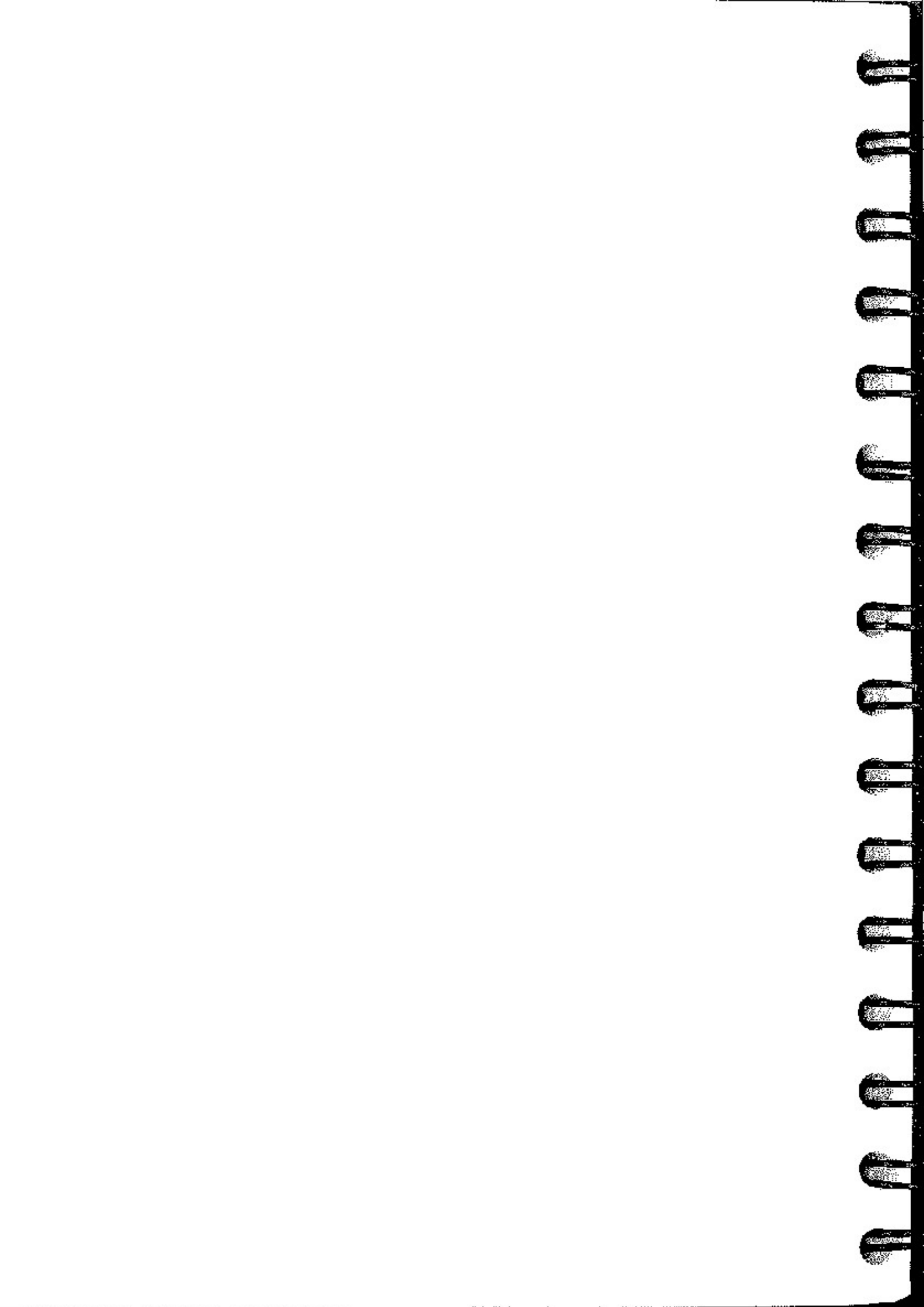
Hvis du ønsker at tegne f.eks. det danske flag, så prøv at se, hvordan ovenstående program er sat sammen, så kan du nemt lave et dannebrogsflag.

Her er et program, der hedder "Hangman" eller "Hæng en mand". En spiller skal indtaste et ord, og en anden skal gætte det.

```
  5 REM Hangman
 10 REM skaermbillede
 20 INK 0: PAPER 7: CLS
 30 LET x=240: GO SUB 1000: REM mand tegnes
 40 PLOT 238,128: DRAW 4,0: REM mund
100 REM ord
110 INPUT w$: REM det ord, der skal gaettes
120 LET l=LENw$: LET v$=" "
130 FOR n=2 TO l: LET v$=v$+" "
140 NEXT n: REM v$ det gaettede ord
150 LET c=0: LET d=0: REM gaet- og fejltaeller
160 FOR n=0 TO l-1
170 PRINT AT 20,n;"-";
180 NEXT n: REM skriver -'er i stedet for bogstaver
200 INPUT "Gaet et bogstav: ";g$
210 IF g$=" " THEN GO TO 200
220 LET g$=g$(1): REM kun første bogstav
```

```
230 PRINT AT 0,c;g$
240 LET c=c+1: LET u$=v$
250 FOR n=1 TO l: REM opdaterer gaettet ord
260 IF w$(n)=g$ THEN LET v$(n)=g$
270 NEXT n
280 PRINT AT 19,0;v$
290 IF v$=w$ THEN GO TO 500: REM det gaettede ord
300 IF w$<>u$ THEN GO TO 200: REM Rigtigt gaet
400 REM tegner næste del af galgen
410 IF d=8 THEN GO TO 600: REM haengt
420 LET d=d+1
430 READ x0,y0,x,y
440 PLOT x0,y0: DRAW x,y
450 GO TO 200
500 REM fri mand
510 OVER 1: REM visker manden ud
520 LET x=240: GO SUB 1000
530 PLOT 238,128: DRAW 4,0: REM mund
540 OVER 0: REM tegner manden igen
550 LET x=146: GO SUB 1000
560 PLOT 143,129: DRAW 6,0,PI/2: REM smil
570 GO TO 800
600 REM haeng mand
610 OVER 1: REM visker gulv ud
620 PLOT 255,65: DRAW-48,0
630 DRAW 0,-48: REM aaben lem
640 PLOT 238,128: DRAW 4,0: REM fjerner mund
650 REM flytter lemmer
660 REM arme
665 PLOT 255,117: DRAW-15,-15: DRAW-15,15
670 OVER 0
680 PLOT 236,81: DRAW 4,21: DRAW 4,-21
690 OVER 1: REM ben
700 PLOT 255,66: DRAW-15,15: DRAW-15,-15
710 OVER 0
720 PLOT 236,60: DRAW 4,21: DRAW 4,-21
730 PLOT 237,127: DRAW 6,0,-PI/2: REM rynken
740 PRINT AT 19,0;w$
800 INPUT "Igen?";a$
810 IF a$=" "THEN GO TO 850
820 LET a$=a$(1)
830 IF a$="n" THEN STOP
840 IF a$(1)="N" THEN STOP
850 RESTORE: GO TO 5
1000 REM tegn mand i kolonne x
1010 REM hoved
```

```
1020 CIRCLE x,143,8
1030 PLOT x+4,134: PLOT x-4,134: PLOT x,131
1040 REM krop
1050 PLOT x,123: DRAW 0,-20
1060 PLOT x,101: DRAW 0,-19
1065 REM ben
1070 PLOT x-15,66: DRAW 15,15: DRAW 15,-15
1080 REM arme
1090 PLOT x-15,117: DRAW 15,-15: DRAW 15,15
1100 RETURN
2000 DATA 120,65,135,0,184,65,0,91
2010 DATA 168,65,16,16,184,81,16,-16
2020 DATA 184,156,68,0,184,140,16,16
2030 DATA 204,156,-20,-20,240,156,0,-16
```



Binær og hexdecimal

Dette appendiks beskriver, hvordan datamater tæller ved brug af det binære system.

I de fleste vestlige (indoeuropæiske) sprog bruges en eller anden form for ti-talssystem. Vore talfegn, de arabiske tal, gør det endda endnu mere klart. Den eneste grund til, at vi bruger et ti-talssystem, er, at vi har ti fingre og ti tæer.

I stedet for at bruge decimaltal, hvor 10 er grundtallet, bruger datamaten hexadecimal (forkortes til hex), hvor 16 er grundtal. Så behøves selvfølgelig seks ekstra cifre og de skrives A, B, C, D, E, F. Men hvad kommer så efter F? – Ja prøv at se:

Begge talsystemer starter på samme måde:

Hex	*Tifalssystem*
0	nul
1	et
2	to
3	tre
4	fire
5	fem
6	seks
7	syv
8	otte
9	ni

Så kommer:

A	ti
B	elleve
C	tolv
D	tretten
E	fjorten
F	femten
10	seksten
11	sytten
..	..
..	..
19	femogtyve
1A	seksogtyve
1B	syvogtyve
..	..
..	..
1F	enogtredive
20	toogtredive
21	treogtredive
..	..
..	..
9E	ethundrede og otteoghalvtreds

9F	ethundrede og nioghalvtreds
AØ	ethundrede og tres
..	..
..	..
B4	ethundrede og firs
..	..
..	..
FE	tohundrede og firoghalvtreds
FF	tohundrede og femoghalvtreds
1ØØ	tohundrede og seksoghalvtreds

Såfremt du skal bruge hexdecimal notation, og du ønsker at gøre det helt klart, så tilføj et "h" efter tallet, f.eks. for enogtredive skriv 1Fh og udtal det: en F hex.

Nu undrer du dig sikkert over, hvad dette dog kan have at gøre med datamater. Jo se, datamater har kun to cifre at arbejde med, nemlig Ø og 1, svarende til tændt og slukket. Dette kaldes det binære talsystem og de to binære cifre kaldes bits. En bit er altså enten Ø eller 1.

I de tre forskellige talsystemer begynder talrækkerne sådan her:

Dansk	*ti-tal*	*hex*	*Binært*
nul	Ø	Ø	ØØØØ
et	1	1	ØØØ1
to	2	2	ØØ1Ø
tre	3	3	ØØ11
fire	4	4	Ø1ØØ
fem	5	5	Ø1Ø1
seks	6	6	Ø11Ø
syv	7	7	Ø111
otte	8	8	1ØØØ
ni	9	9	1ØØ1
ti	1Ø	A	1Ø1Ø
elleve	11	B	1Ø11
tolv	12	C	11ØØ
tretten	13	D	11Ø1
fjorten	14	E	111Ø
femten	15	F	1111
seksten	16	1Ø	1ØØØØ

Det vigtigste punkt i det her er, at seksten er 2^4, hvilket gør omregning mellem binær og hex meget enkel. Når et hex-tal skal laves om til et binært, så brug tabellen og lav hvert hex-ciffer om til fire bits.

Når binære tal skal laves om til hex-tal, så del det binære op i grupper på fire bits og brug igen tabellen til at lave disse om til hex-cifre.

Selv om datamaskinen udelukkende bruger det binære system, skriver mennesker ofte tal fra datamaten i hex-form.

Bitsene i datamaten er som regel ordnet i grupper på otte, og en sådan gruppe kal-

des en *byte*. En enkelt byte kan indeholde alle værdier fra Ø til 255 (d.v.s. 11111111 binært eller hex FF) eller med andre ord: et hvilket som helst tegn i ZX Spectrum-karaktersættet. Dets værdi kan skrives med to hex-decimaler.

To bytes kan kobles sammen, og så har man det, der i tagsprogot kaldet et *ord*. Et ord kan rumme alle værdier fra Ø til 65535=2^{16}−1.

En byte er altid otte bits, men der er nogen forskel på længden af et ord fra maskine til maskine.

BIN-notationen i kapitel 23 giver en metode til at skrive binært på ZX Spectrum: **BIN Ø** repræsenterer "intet" eller nul og **BIN 1** repræsenterer en, **BIN 1Ø** repræsenterer to, o.s.v.

Du kan kun bruge Ø og 1 til dette, så tallet må nødvendigvis være et ikke-negativt heltal. Du kan f.eks. ikke skrive: **BIN −11** for minus tre. I stedet skal du skrive **−BIN 11**. Tallet må heller ikke være større end 65535 i decimal, d.v.s. ikke flere end seksten bits.

Funktionen **ATTR** er i virkeligheden binær. Hvis du omregner resultatet fra **ATTR** til et binært resultat, så vil det kunne skrives som otte bits:

Den første er 1 for "blink" og Ø for normal.

Den anden er 1 for ekstra lysstyrke og Ø for normal.

De næste tre er koder for paper-farven, skrevet binært.

De sidste tre er koder for ink-farven, skrevet binært.

Farvekoderne bruger også binær. Hver kode, skrevet binært, kan skrives som tre bits, den første for grøn, den anden for rød og den tredje for blå. Sort har ingen "lys"-styrke, så alle bits er Ø (slukket). Koden for sort er derfor ØØØ, binært eller nul (decimal).

De rene farver grøn, rød og blå har kun en bit (1) af de tre bits tændt. Deres koder er 1ØØ, Ø1Ø og ØØ1, binært eller fire, to og en.

De andre farver er kombinationer af disse, så deres binære koder indeholder to eller flere "bits 1".

Stikordsregister

R

S

ISBN 87-88267-06-7