EN DATA BECKER BOG

LIESERT

AMSTRAD

464/664 & 6128

Peeks & Pokes

Uundværlig for programmører

DANSK/NORSK udgave

Forlag: [logo] box 105 DK-6950 Ringkøbing

PEEKS & POKES

EN DATA BECKER BOG

LIESERT

AMSTRAD

464/664 & 6128

Peeks & Pokes

Uundværlig for programmører

DANSK/NORSK udgave

NORDIC COMPUTER SOFTWARE
SMEDEGADE 7 · POSTBOX 105 · DK-6950 RINGKØBING
1986

Copyright 1986 Data Becker & Nordic Computer Software
Postbox 105
Smedegade 7
DK-6950 Ringkøbing

Distribution i Norge:
Computer Equipment A/S
Gyldenløves gt. 42
N-4600 Kristiansand S.
Tlf. (042) 70 294

Sats & tryk: Tarm Bogtryk & Offset A/S

Dansk oversættelse & bearbejdning: Peter Friis

ISBN 87-7283-008-5

Alle rettigheder til den danske version tilhører Nordic Computer Software.
Bogen må under ingen form (fotokopi, aftryk el. lign) reproduceres uden skriftlig tilladelse fra udgiveren. Ligeledes må bogen, eller dele heraf, ikke udbredes via elektroniske medier.

VIGTIGT

Programmer, programeksempler mv. er omfattet af lov om copyright. Disse må kun anvendes til personlige- eller undervisningsformål og må ikke anvendes kommercielt.

Alle programeksempler, tekniske anvisninger osv. i denne bog er omhyggeligt gennemgået for fejl. Trods dette kan der optræde fejl i reproduktionsfasen.
Skulle nogle læsere opdage fejl, beder vi Dem rette henvendelse til os, så vi har mulighed for at foretage rettelse.

FORORD

Når der i bogen refereres til CPC 464, menes alle tre 3 typer hvor den eneste forskel er RAM størrelsen.

Mon ikke man som ejer af en CPC-446 eller 664/6128 på et eller andet tidspunkt har spekuleret lidt over, hvad der egentlig foregår inde i maskinen?

Har man ikke også brændt efter at finde ud af, hvad et operativsystem er, og hvad det bestiller?

Den slags spørgsmål, hverken kan eller skal en medfølgende manual kunne svare på. Mange af de tricks og små hjemmefuskede kneb, der gør livet en smule lettere for en programmør, kan kun tilegnes ved at man kigger i kulissen.

Denne bog skal hjælpe til med at løse nogle af mysterierne ved programmering. Vi vil forklare, hvordan computeren arbejder; hvad der sker, når man trykker på return-tasten.

Selv om bogens titel antyder det, så drejer det sig ikke alene om de to kommandoer PEEK og POKE, (det ville blive alt for kedeligt).

PEEK og POKE er BASIC's bagdør ind i maskinsprogets spændende verden. Kender man en smule til Commodore 64, så ved man også, at mange ting på denne computer er vanskelige at have med at gøre, fordi de kræver flittig brug af PEEK- og POKE-kommandoerne. Denne bogs forgænger er skrevet til Commodore 64, så deraf titlen.

Vi vil via PEEK og POKE bevæge os over i maskinsprog, men hele tiden med en sikker fod i BASIC. Desuden gennemgåes operativsystemet og dets funktioner.

De mange tricks, der dukker op bogen igennem, kan frit benyttes i egne programmer. Det er vort håb, at det korte kursus i maskinkodeprogrammering sidst i bogen vil give læseren så meget blod på tanden, at han/hun får lyst til at fortsætte på egen hånd.

INDHOLDSFORTEGNELSE

1. HVORDAN COMPUTEREN ARBEJDER 9

1.1.	Busser er også for microcomputere	9
1.2.	CPC's Hardware-konfiguration	10
1.3.	Hukommelsens inddeling	11
1.4.	Pointer og stacks	13

2. OPERATIVSYSTEM OG INTERPRETER 15

2.1.	Den uundværlige koordinator	15
2.2.	Interpreteren	16
2.3.	Afbrydelser sat i system	16
2.4.	Ej kun for dem, der kan: specialkommandoer i Basic	18
2.4.1.	Peek og Poke	18
2.4.2.	Call	19
2.4.3.	En udflugt i binær-aritmetikken	19
2.4.4.	Forbindelse med resten af verden: Port-kommandoer	21
2.5.	Kommandoer der ikke står i manualen	22

3. HUKOMMELSEN 24

3.1.	Beskyttelse af hukommelse	24
3.2.	Hvordan fungerer Bank-switching?	25
3.3.	Udlæsning af Rom	26
3.4.	Hukommelsesudvidelser	27

4. TRICKS TIL SKÆRMBILLEDET 28

4.1.	Styring via CHR$-kommandoer	28
4.2.	Video-Ram set indefra	31
4.3.	Grafik i det skjulte	33
4.4.	Lagring af skærmbilleder	35
4.5.	Scroll	36
4.6.	Scroll på en anden måde	37
4.7.	Cursorstyring endnu en gang	38

5. GRAFIK 40

5.1.	Grafik-styretegnet	40
5.2.	Kasser og rektangler	41
5.3.	Rundt om Sinus og Cosinus	42
5.4.	Hvorfor pixeltest	45
5.5.	Koordinatsystemer	48
5.6.	3-D Grafik	49

6. NYTTIG GRAFIK 54

6.1. Diverse diagrammer 54
6.2. Programmet for kunstnere 56

7. INTERRUPT-PROGRAMMERING 59

7.1. Hvordan fungerer en Basic-interrupt? 59
7.2. Interrupt-kommandoer 60
7.3. Ideer til interrupt-programmering 62

8. SOUND 63

8.1. Mini-synthesizer 63
8.2. Hvordan planlægges en lyd 66

9. BASIC OG OPERATIVSYSTEMET 68

9.1. Hvordan lagres Basic-liner 68
9.2. Garbage Collection 69
9.3. Error! Error! 70
9.4. Ubekendte sider 70
9.5. Endnu et par tricks 71

10. EKSTRAUDSTYR OG DERES FUNKTIONER 73

10.1. Diskettedrevet 73
10.2. Printeren 73
10.3. Joystick eller styrepind 74

11. LIDT OM INTERFACING 76

11.1. Lille interface-kursus 76
11.2. Hvordan fungerer et interface? 76
11.3. Det personlige interface 77
11.4. Tastaturaflæsning 77

12. KASSETTEBÅNDOPTAGER OG TASTATUR 78

12.1. Hvordan opbygges en fil? 78
12.2. Inkey i et andet lys 80

13. INDFØRING I MASKINSPROG 82

13.1. Hvad er maskinsprog i det hele taget for noget 82
13.2. Clock-frekvensen 83
13.3. En microprocessors opbygning 83
13.4. En microprocessors arbejdsgang 85

13.5. HEX-tal eller sekstentalssystemet 86
13.6. Binær-aritmetik 87
13.6.1. Addition 87
13.6.2. Subtraktion 88
13.6.3. Multiplikation 90
13.6.4. Division 90
13.7. Hvordan fungerer sammenligninger (relationer) 91
13.8. Det første Z80-program 92
13.9. Hvordan programmeres en løkke? 94
13.10. Yderligere aritmetikrutiner 95
13.10.1. 16-bits-addition 95
13.10.2. Multiplikation 96
13.11. Nyttige maskinkoderutiner 98
13.12. Adresseringsmulighederne 100
13.13. Z80-kommandoer 101
13.14. Z80-processorens OPCODES 109

TILLÆG 126

Hukommelsens belægning 126

ORDLISTE 129

1. HVORDAN COMPUTEREN ARBEJDER

I de følgende afsnit vil vi kigge på CPC-464's interne arbejdsgang. Skulle der blandt læserne være nogle, der er fortrolige med computerteknik, er det tilladt for disse at springe let hen over siderne. Skulle der ligeledes være nogle super-crackere blandt læserne, bedes man have os undskyldt at vi til tider måske forenkler sagerne lidt for meget. Vi gør det for forståelighedens skyld.

1.1. BUSSER ER OGSÅ FOR MICROCOMPUTERE

Først skal vi se på det grundlæggende. Enhver mikroprocessor kan adressere et bestemt adresse/hukommelsesområde. Det vil sige: Kommunikere med et nærmere bestemt område af hukommelsen. Størrelsen af et sådant adresseområde afhænger af antallet af processorens "adresseringskanaler". Hver af disse kanaler repræsenterer en bit (hvad en bit er, kender man sikkert fra manualen), der kan antage to tilstande:

0 og 1.

Mikroprocessoren Z80 (der er hjernen i CPC 464), er i besiddelse af 16 af sådanne adresseringskanaler. Dette kaldes bussen. Bussen = Antallet af adresseringskanaler. Adressebussen kan ialt kommunikere med 2↑16 = 65536 hukommelsesceller.

Måske, undrer man sig over den kendsgerning at CPC'en råder over 64 K RAM og 32 K ROM, ialt 96 K RAM, d.v.s. mere end den kan adressere. Hvordan det trods alt er muligt, vil vi fortælle senere. Foruden adressebussen, findes der endnu 2 busser i Z80'eren. Den ene er STYRE-BUSSEN, der egentlig kun er et fællesnavn for det samlede antal af styreledninger, der f.eks. har til opgave at skifte mellem hukommelsens input og output. Den anden er databussen. Hver hukommelsescelle (adresse) består af 8 bits (1 byte). Derfor har processoren brug for en databus med 8 adresseringskanaler, så den kan læse/skrive på den adresse, der er blevet adresseret. Da databussen er nøjagtig halvt så bred som adressebussen, skal der bruges 2 bytes til hver adresse.

Kun i processoren kan der adderes og subtraheres eller regnes på anden måde. Alle andre dele i en computer gemmer "husker" informationer eller konverterer dem til andre former for data (f.eks. toner eller signaler til skærmbilledet).

Databussen står i forbindelse med alle chips i computeren, der skal kommunikere med processoren. Hver gang processoren udfører en kommando eksternt, skal den adressere den rigtige adresse i hukommelsen. Den eller de chips, der indeholder hukommelsen, ved så, hvilken hukommelsescelle, der står for tur. På den måde kan der transporteres information enten til eller fra Z80-processoren.

Alt dette er gældende for alle 8-bits processorer. Men fra nu af vil vi koncentrere os om CPC-464.

1.2. CPC'S HARDWARE-KONFIGURATION

For at kunne forstå de følgende kapitler til bunds, er det nødvendigt at kende lidt til, hvordan computerens indre er opbygget. I slutningen af dette afsnit er der vist et stærkt forenklet blokdiagram over computeren (fig. 1). Som det fremgår, ligger RAM og ROM delvis placeret ved siden af hinanden, d.v.s., at de benytter de samme adresser. Et eller andet sted må det således blive bestemt, hvilken af de to dele, der menes: RAM eller ROM. Alt efter formål, kan Z80 skifte mellem disse afsnit.

Fra BASIC, kan man iøvrigt kun kommunikere med RAM via PEEK og POKE - Og dog! Senere skal vi se på, hvordan man kan snyde lidt og alligevel bruge ROM via BASIC.

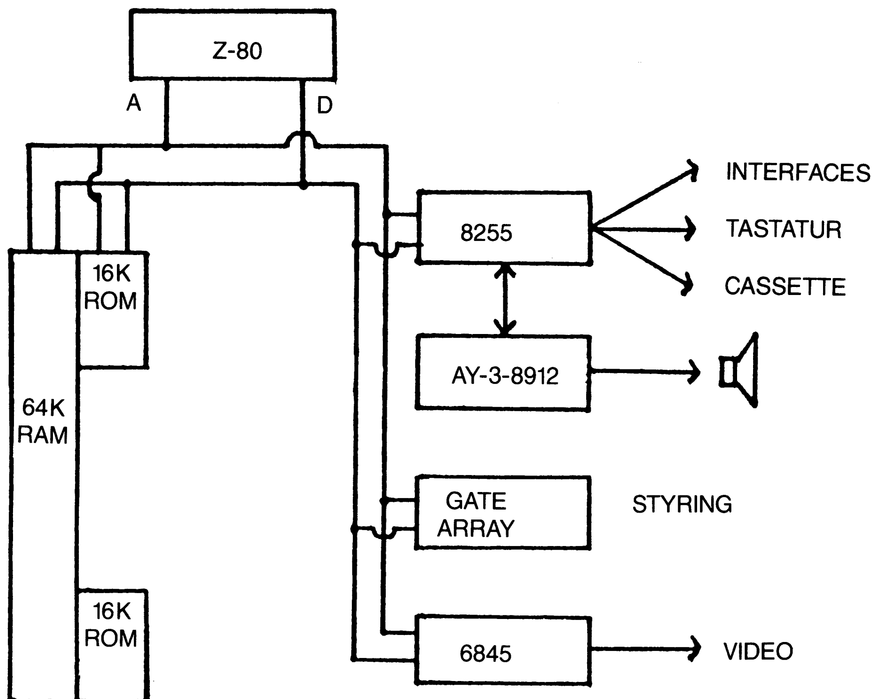
En del af RAM optages af skærmbilledets hukommelse. Det er herfra den såkaldte 6845 chip får sine informationer. Denne integrerede kreds har til opgave at opbygge dataene fra Video-RAM til et færdigt video-signal til monitoren.

Der findes flere chips, f.eks. AY-3-8912, der er ansvarlig for al lyd, d.v.s. at den omsætter SOUND-kommandoen til hørbare toner. Af tekniske grunde er AY-3-8912 ikke koblet direkte på Z80.

Dataene overføres i stedet for via interface-chip'en 8255 (se fig. 1). Denne kreds er mellemled for data fra processoren til perifert udstyr, så som kassettestation, interfaces og tastatur (sidstnævnte betragtes som hørende under perifert udstyr, da dette kun af hensigtsmæssige grunde er koblet sammen med centralenheden i samme "kasse".

Når Z80 skal aflæse tastaturet, så gives der besked til interface-chip'pen, der så kigger efter, hvilken tast, der blev trykket på. Det meddeles så via databussen. Skal en byte sendes ud via et interface, så sendes den til 8255, der

"forsinker" byten indtil udstyret, der kommunikeres med, har meldt klar bane. En anden chip's opgave er at styre bestemte interne operationer. Denne har ikke så stor betydning for os.



Figur 1: Opbygningen af CPC-464.

1.3. HUKOMMELSENS INDDELING

Som det er fremgået af det forrige kapitel, så har CPC'en 64 K RAM. I BASIC har vi imidlertid kun 42,5 K RAM til rådighed. Så det er med rette, man kan stille spørgsmålet:

HVOR ER RESTEN AF HUKOMMELSEN BLEVET AF?

Ved det første blik, kan det se ud som om, at det ville være muligt at reservere et meget større område til BASIC. Men desværre, skal en computer også bruge plads i hukommelsen til de interne funktioner. En del af denne plads optages af video-RAM'en, der har til opgave at gemme skærmens udseende. Hver gang et punkt på skærmen ændres, så ændrer processoren en tilsva-

rende værdi i video-RAM. Den chip, der har skal styre signalet til monitoren, må i regelmæssige intervaller, finde ud af, hvilke punkter på skærmen, der skal lyse op. Chip'en aflaster Z80 for dette "kedsommelige" job, idet processoren har vigtigere opgaver.

Video-RAM fylder 16 K bytes, der selvfølgelig ikke kan benyttes af BASIC, og ligger i området fra adresse 49152 til 65536. Det er de samme adresser som BASIC-fortolkeren. Trækker vi 16 K fra de 64 K, så er der 48 K tilbage. Ifølge vore beregninger, mangler der så 5,5 Kb.

Alle processorer har brug for et område i hukommelsen, hvori de kan mellemlagre deres "personlige" data. Dette område kaldes STACK (stakken). Hvergang Z80 skal udføre en underrutine, skal den kunne gemme oplysninger om, hvor rutinen blev kaldt fra, for at kunne vende tilbage igen efter udført arbejde. Det sker ved hjælp af STACK. Den ligger i området fra 48896 til 49151 og fylder dermed nøjagtig 256 bytes. Det område skal man afholde sig fra at POKE i, da det højst sansynligt vil føre til at computeren hænger op, og kun kan bringes til at fungere igen efter at have været afbrudt.

For at kunne arbejde i både RAM og ROM angiver området 0 til 63 i RAM en kopi af de parallelle bytes (tilsvarende). Arbejder Z80 i RAM og får brug for data fra ROM, kan den via dette område skifte mellem RAM og ROM. I RAM findes også rutiner mellem BASIC-hukommelsen og video-RAM. Heller ikke her må man ændre nogen værdier med POKE-kommandoen.

Operativsystem og interpreter (fortolker) har brug for et vist antal hukommelsesadresser for at kunne gemme mellemresultater ved udførelse af aritmetiske opgaver, udveksle data, mellemlagre tastatur-input o.s.v.

I denne sammenhæng, er tastatur-bufferen af særlig interesse. Den gør det muligt, at indtaste tegn inden BASIC overhovedet er klar til at modtage disse. Det kan man selv teste. Prøv at indtaste følgende linie, og kør den:

```
10 FOR I=1 TO 10000:NEXT I
```

Når programmet kører, kan man indtaste nogle vilkårlige tal. Læg mærke til at ikke et eneste af disse tegn vises på skærmen. Men så snart de titusinde gennemløb er fuldført, kommer tegnene til syne. Mens programmet kørte, havde operativsystemet gemt op til 20 tegn. På den måde kan man f.eks. ved lange beregninger forberede indtastningen til den næste INPUT-kommando. Husk, at hvis man BREAK'er sit BASIC-program, så mister man alle evt. tegn i tastaturbufferen.

Hvis man ønsker et større overblik over hukommelsen, kan man kigge i denne bogs tillæg. Der finder man en oversigt over hukommelsens belægning med adresseangivelser for de enkelte områder.

1.4. POINTER OG STACKS

To fagord, man uvilkårligt støder på om og om igen, er POINTER og STACK. En pointer peger på et bestemt sted i hukommelsen og kaldes ligeledes en VEKTOR. Der kan stå enten informationer eller underrutiner. Cursorpointeren peger f.eks. på det sted i skærmbilledets hukommelse, hvori cursoren netop befinder sig, og dermed på bogstavkoden for det blinkende tegn. Pointere på underrutiner blev indført, for at give interpreteren muligheder for udvidelser. Ændrer vi f.eks. vektoren på en rutine for udskrivning af et tegn, så er det muligt at ændre PRINT-kommandoen med et maskinkodeprogram, således at alle tegn udskrives samtidigt på skærmen og på printeren.

Pointere har altid et bestemt format. De består i almindelighed af 2 bytes, hvoraf den første kaldes Lo-byte og den sidste kaldes Hi-byte. For at finde cursorens position eller den byte, der peges på, benytter man følgende formel.

ADRESSE = LO-BYTE + 256 * HI-BYTE

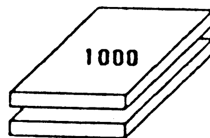
Arbejder man i hexadecimalsystemet, sættes Lo-byte på den højre side og Hi-byte på den venstre, og danner på den måde den fire-positioners hexadresse.

Det hører til computerens egenhed, at Lo-byte altid står før Hi-byte.

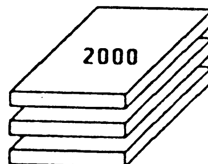
1-byte-pointere peger på et sted indenfor et bestemt område, f.eks. tastaturbuffer eller stack, og adderes til en BASISADRESSE.

En stack (engelsk: stak eller søjle) har til opgave at mellemlagre data, og aflevere dem igen i omvendt rækkefølge når de skal benyttes. Som ved en rigtig stak, kan man kun tage fra oven, og kun lægge nye ting oveni. (Se figur). Dette benyttes af alle underprogrammer. Ved kald af et underprogram, mellemlagres den forladende adresse i stack, og hentes tilbage når underprogrammet er udført (ved RETURN), således at hovedrutinen kan fortsættes på det sted, hvor den blev afbrudt.

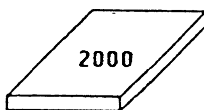
1000 GOSUB 2000

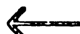


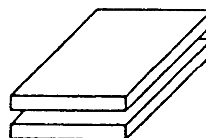
2000 GOSUB 3000



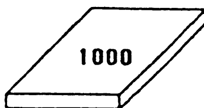
3010 RETURN

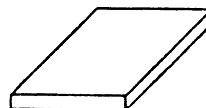






2010 RETURN





POINTERE

ADRESSE = LO-BYTE + 256 * HI-BYTE
LO-BYTE = ADRESSE - INT(ADRESSE/256)*256
HI-BYTE = INT(ADRESSE/255)

Pointere består sædvanligvis af 2 bytes, der altid optræder i rækkefølgen Lo/Hi.

---

2. OPERATIVSYSTEM OG INTERPRETER

Før vi går nærmere ind på CPC-464's muligheder, så er der nogle fagudtryk, der skal uddybes. Det er bestemt ingen skam, hvis man indtil nu ikke ved, hvad et operativsystem er og hvad det bestiller. Det er operativsystemet, vi vil beskæftige os med i dette kapitel.

2.1. DEN UUNDVÆRLIGE KOORDINATOR

Alle computere har det, men det er kun de færreste begyndere i computerverdenen, der ved hvad der ligger bag udtrykket operativsystem. Som navnet antyder, er operativsystemet uundværligt for computeren. For f.eks. at flytte en byte fra processoren til printeren, skal der bruges små rutiner i operativsystemet. Selv om selve overførslen foregår af sig selv, skal interface-chip'en programmeres og kontrolleres, så evt. opståede fejl kan opdages i tide. Generelt styrer operativsystemet således sammenarbejdet mellem computer og periferi (også tastatur og printer). Hvert apparat har dermed sine egne rutiner i operativsystemet, der kan anvendes af BASIC-fortolkeren eller andre programmer. For at kunne sende bogstaver og tal til periferien, behøver BASIC kun at klargøre tegnene, for derefter at kalde de rigtige rutiner i operativsystemet.

Læseren har formodentlig hørt udtryk som CP/M og MS-DOS. Det er navnene på de mest kendte og anvendte operativsystemer. De kaldes forøvrigt standardoperativsystemer. Systemerne er opbygget, så der ikke længere sker kald af underrutiner, men overførsel af kommandonumre og data til specielle registre. På den måde kan et operativsystem tilpasses forskellige computere; koderne forbliver de samme. Man må i den forbindelse være klar over, at maskinkodeprogrammering har nogle bivirkninger. Når rutinerne tilpasses en anden computer, forandres tilgangsadresserne for de efterfølgende underprogrammer, idet antallet af bytes næsten aldrig kommer til at stemme. Hver computers operativsystem har sine egne adresser og dette afstedkommer at alle programmer, der skal gøre brug af rutinerne må skrives om. Når man i stedet for anvender kommandokoder bliver maskinkodeprogrammering et godt værktøj til at skabe nye operationer (ens for computerne). For softwarehusene betyder det, at man ikke skal skrive programmerne om, hver gang der kommer en ny computer på markedet.

2.2. INTERPRETEREN

Navnet siger meget praktisk, hvad dette maskinkodeprogram bosiddende i ROM laver. Fortolkeren (det danske udtryk for samme) oversætter BASIC-kommandoerne til noget, processoren kan forstå. Z80-processoren kan kun forstå sine egne kommandoer (fælles for alle processorer). BASIC-kommandoer er for processoren kun sammmenhængende tegn. Følger der efter en sådan tegnrække, en kode for ENTER-tasten, starter fortolkeren sin første fase i oversættelsen.

Nu har processoren pludselig fået travlt. Bogstaver og tal skal genkendes som kommandoer, data og linienumre. Kommandoer får tildelt et specielt kodenummer, så interpreteren kan finde dem i en fart, ved den endelige kørsel af underprogrammer.

Indtil nu er tegnfølgen kun blevet kodet, derimod står der intet at læse i programhukommelsen. Der skal måske endda først skaffes plads til den. Derefter overføres programmet. De indtil nu forløbne processor sker i løbet af det korte øjblik, der opstår mellem aktiveringen af ENTER-tasten og cursorens tilbagekomst på skærmen; altså "lynhurtigt". Anden del af fortolkningen sker når RUN-kommandoen udføres. Fortolkeren henter de enkelte kommando-koder og data fra programhukommelsen og bringer dem til udførelse. Står den aktuelle kode for PRINT-kommandoen, startes underprogrammet betegnet PRINT.

Da kommandoerne har specifikke koder, behøves PRINT-tegnkoden ikke længere at blive genkendt efter RUN; derimod skal den rigtige pointer på et underprogram findes, hvilket sparer tid.

2.3. AFBRYDELSER SAT I SYSTEM

Det, der mellem mennesker anses for uhøfligt, er god tone blandt computere. Der er mange funktioner, der kun er mulige på grund af interrupt (dansk: afbrydelse). Både BASIC-fortolker og operativsystem er maksinkodeprogrammer, der startes op når computeren tændes. De er aktive indtil et andet program i maskinsprog kaldes. Hvis det sker ved hjælp af en CALL-kommando, så vender computeren tilbage til BASIC, efter at have udført maskinsprogsrutinen.

Som man af erfaring ved fra BASIC-programmering, kan en computer kun udføre en enkelt ting af gangen (multiprocessor-systemer undtaget). Interpreter og operativsystem er imidlertid to adskilte programmer, der aktiveres alt efter opgave. Hvordan sker dette?

Den nemmeste måde at få to programmer afviklet på næsten samtidigt, er metoden med at lade programmerne kalde hinanden efter tur. Hver gang BASIC har afsluttet en del af sit arbejde, kobler den operativsystemmet ind, og omvendt. Dette sker f.eks. når forskelligt periferiudstyr anvendes. BASIC'en bearbejder således løbende de data, som operativsystemet skal sende til andre enheder. Det medfører dog, at tastaturet kun aflæses, når det er operativsystemets tur til at arbejde. Under udførelse af et program er det nødvendigt at lade mindst ESC-tasten (BREAK) fungere. Ellers ville den eneste mulighed for at standse et program, være at slukke for computeren! Dette problem førte til opfindelsen af den såkaldte interrupt. Hver 1/50 sekund afbrydes det netop afviklende program for en aflæsning af tastaturet og lignende ting. Hvis der ved sådan en interrupt "opdages" et tryk på ESC-tasten, så afbrydes udførelsen af BASIC-programmet. Hvis andre taster er aktiveret, vil deres værdier blive flyttet til tastatur-bufferen, der forøvrigt er en meget nyttig indretning.

For brugeren ser det ud som om tastaturet hele tiden aflæses. Selv den hurtigste tastatur-hacker kan vel næppe klare mere end 15 tegn i sekundet. For microprocessoren derimod synes tiden mellem to interrupts uendelig lang, da den arbejder med en frekvens på fra ca. 4 millioner skift i sekundet og en maskinkodekommando i gennemsnit afvikles på fra 8 til 10 af sådanne skift. Sagt på en anden måde så kan processoren i tidsrummet mellem to interrupts udføre tusinder af instruktioner. Efter hver tastaturaflæsning fortsætter processoren samme sted i programmet, den forlod.

Under udførelse af enkelte kommandoer kobles interrupt fra, f.eks. ved kassettebrug. Farveskift styres af interrupt. Også i BASIC kan man gøre brug af interrupt. Anvender man AFTER og EVERY, kan man opbygge egne interruptrutiner i BASIC. Møder fortolkeren en AFTER- eller en EVERY-kommando, startes en såkaldt TIMER, der virker som et vækkeur. Når det ringer udløses et interrupt. Fortolkeren smider så alt hvad den har i hænderne, og udfører interrupt-rutinen.

Interrupt-signalet kan også afgives af andre af computerens chip's. Det kan f.eks. være for at gøre opmærksom på, at der skal overføres data gennem et interface. BASIC anerkender dog kun TIMER-signalerne.

En speciel form for interrupt, er RESET-funktionen. Her afbrydes alle løbende processer, og Z80 får besked på at starte forfra ved adresse 0, som ved computerens opstart. I adresse 0, ligger starten på initialiseringsrutinen, der sletter hukommelsen og foretager andre vigtige opstartsoperationer. Fra BASIC kan man kalde denne funktion med CALL 0. Men husk at gemme evt. vigtige data på kassettebånd inden!

2.4. EJ KUN FOR DEM, DER KAN: SPECIALKOMMANDOER I BASIC

Lad os forestille os den følgende situation:

I et computerblad finder man et super-duper-program lige til at taste ind. Efter at de 20 K er tastet ind får man en ERROR meddelelse; eller endnu værre, computeren har låst sig fast. Det eneste, man kan gøre, er at slukke for computeren!

For at finde fejlen, må man enten forstå, hvad der sker i programmet, eller også sætte sig til at sammenligne hvert enkelt bogstav i hele udlistningen. Et udbrud, man jævnligt støder på lyder:

- Hvis bare der ikke var alle de dumme POKE kommandoer! Der er da ingen normale programmører, der bruger dem!

Altså er det på tide, at se på, hvad POKE og PEEK egentlig står for og hvorfor de er så nødvendige.

2.4.1. PEEK og POKE

Lad os først tage POKE kommandoen. Syntaksen er vel bekendt:

POKE adresse,byte.

Adressen må ligge mellem 0 og 65535, byten mellem 0 og 255. Kommandoens opgave er at gemme byten på den angivne adresse. Dette kan have mange formål. Alt efter adresse, kan man fylde skærmen, bestemme en farve eller noget helt tredie. Vi kan koste rundt med computeren, som vi ønsker.

På samme måde kan man undersøge, hvad der står i en hvilken som helst adresse i hukommelsen. Dette gøres med kombination af PRINT PEEK(adresse), der skriver byten ud af adressen.

PEEK er en funktion, og kan kun stå i forbindelse med en tilskrivelse (A=PEEK...) eller et andet udtryk. Fælles for de to kommandoer er det, at deres formål afhænger af hvilke adresser, de benyttes på. Derfor er det en god ide, at undersøge i hvilken del af computerens hukommelse der arbejdes. Det er ofte muligt direkte at læse ud fra dette, hvad funktionen er.

2.4.2. CALL

Nu kommer vi til en af de kommandoer, der specielt er af interesse for maskinsprogsprogrammører:

CALL adresse

Kommandoen tjener til kald af programmer skrevet i maskinsprog. Ved SYS kommandoen angiver adressen, den byte hvormed programmet skal starte. Efter afslutning af rutinen, vender interpreteren tilbage fra underprogrammet og fortsætter BASIC programmet.

2.4.3. EN UDFLUGT I BINÆR-ARITMETIKKEN

Blandt de i det følgende beskrevne kommandoer, skal der nok være nogle stykker, der kendes i forvejen; dog er deres mangesidede anvendelsesmuligheder nok ikke kendt af enhver.

De første, der skal nævnes er:

AND, OR, XOR og NOT

De har indtil nu kun været anvendt på følgende måde:

IF A=0 AND B=0 THEN 100

Det er den såkaldte IF-THEN-instruktion. Men de er egentlig tænkt anvendt til brugen af udsagnslogik, dvs. logisk sammenligning af variabler (herunder tekst) og tal. Alle sammenligninger sker numerisk. Prøv følgende kommandoer:

PRINT (1=2)

PRINT (1=1)

En sammenligning, der som resultat bliver SAND udtrykkes med et -1, og et FALSKt udtrykkes med et 0. I det binære talsystem ser SAND således ud:

11111111

Hvis man ikke betragter den mestbetydende bit som fortegnsbit, vil det til decimalsystemet konverterede tal lyde:

Hvad har så dette med BASIC-kommandoer at gøre? En IF-THEN instruktion forlades altid (ignoreres) når resultatet bliver 0. For at vise de enkelte sammenligninger, foretager vi nu en lille udflugt til binær-aritmetikken.

AND, OR, XOR og NOT er såkaldte BOOLESKE OPERATIONER, der tjener til sammenligning af logiske tilstande. Og som det er fremgået, kan logiske tilstande på simpel måde angives med 0 for FALSK og 1 for SAND.

Det er altid kun to tilstande, der kan sammenlignes af gangen. Hvad der kommer ud af det, viser nedenstående skema:

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

XOR	0	1
0	0	1
1	1	0

Det ses at resultatet altid er 1, hvis begge bits er 1.

Funktionerne kan også siges med talte ord:

Ved AND er resultatet 1, når bit 1 OG bit 2 er 1.

Ved OR er resultatet 1, hvis bit 1 ELLER bit 2 er 1.

Ved XOR er resultatet 1, hvis en af operanderne er 1.

Helt anderledes forholder det sig med NOT. Denne funktion INVERTERER ganske enkelt bit'en:

NOT	0	1
	1	0

Så langt, så godt. Desværre er der endnu et problem for os ubemidlede BASIC-programmører! De enkelte bits er jo ikke så tilgængelige i BASIC. Vi kan kun arbejde med decimaltal. For at beregne, hvad udtrykket 45 AND 123 giver som resultat må vi gå frem på følgende måde:

1. Tallene skal konverteres til to-talssytemet. (Se i CPC-manualen).

    45 ==› 00101101
    123 ==› 01111011

2. Bitvis sammenligning i to-talssystemet:

    45 AND 123 giver:

```
      00101101
AND   01111011
--------------
      00101001
--------------
```

    00101001 ==› decimaltalssystemet = 41

Det kunne selvfølgelig lettere regnes ud med:

PRINT 45 AND 123

Men det lærer man nok ikke ret meget af!

Man kan i stedet med rette stille spørgsmålet: Hvad skal det gøre godt for alt sammen? Bortset fra sammenligningen,bruges disse kommandoer tit til at gøre indflydelse på enkelte bits. Med logisk sammenligning med AND, bliver 254 den mindstbetydende bit altid slettet (0), og ved OR sættes den altid til (1). Prøv selv med andre tal!

2.4.4. FORBINDELSE MED RESTEN AF VERDEN: PORT-KOMMANDOER

Via stikforbindelsen på bagsiden af CPC'en, kan man kommunikere med perifere enheder. For at det også kan lade sig gøre fra BASIC, findes der specialkommandoer til dette. INP(port) henter en byte fra porten, hvis nummer er angivet i parentes. Portnummeret har således ikke noget at gøre med adresseangivelser. Der findes også et særligt adresseringssystem, der først i forbindelse med maskinsprogsprogrammering bliver forståelig. Alligevel vil vi forklare de tilhørende kommandoer for at fortælle, hvad der ligger bag. OUT port. Man kan sende en byte ud via en port. Som det ses, svarer syntaxen så nogenlunde til PEEK og POKE. Tilbage er den lidt hemmelighedsfulde kommando:

WAIT port,X,Y

Den har en opgave, der resulterer i at bit'ene i enhver processors hukommelse roterer. Der skal nemlig ventes; og det er slet ikke noget, en computer bryder sig om. Det sker ved kontinuerlig logisk sammenligning af bytes. Møder fortolkeren en WAIT-kommando, så læses der først en byte fra porten. Dette tal EXCLUSIV OR'es (XOR) med tallet Y.

Resultatet af den første sammenligning AND'es nu med tallet X. Bliver resultatet et 0 (nul), så gentages hele proceduren, ellers fortsættes med den næste kommando i rækken.

Der findes dog også en anden variant af WAIT-kommandoen, hvor Y-argumentet ikke angives. Her afventer fortolkeren, at byten fra den angivne port AND X bliver forskellig fra 0.

2.5. KOMMANDOER, DER IKKE STÅR I MANUALEN

Hvad man kender fra andre computerfabrikater, er også sket ved konstruktionen af CPC 464. Efter at en programmør har brugt al sin viden og resourcer på opbygningen af en BASIC-fortolker og efter at han har tilbragt oceaner af tid med at debugge sit program, så glemmer forfatteren af håndbogen at beskrive en af kommandoerne.

Kommandoen hedder MOD, der kommer af ordet MODulo. Udtrykket er kendt af de fleste matematikere. Modulo-funktionen giver REST-tallet efter en division. Her et par eksempler:

10 / 4 = 2.5 ===› 2 REST 2

10 MOD 4 = 2

11 MOD 4 = 3 (11/4=2 REST 3)

Kommandoen MOD kan således erstatte divisionstegnet (/), hvis man ønsker at finde RESTen af en division. Dog fungerer MOD kun ved tal og variabler af typen INTEGER (-32768 til +32767). Med MOD-kommandoen er det således enkelt at programmere EUKLID's ALGORITME (se kapitel 14).

Modstykket til MOD-kommandoen er INTEGER-divisionen. Den giver det modsatte tal, nemlig det nærmeste hele tal af en division. 11 delt med 4 giver således resultatet 2. INTeger-divisionen er ligeledes kun beregnet for heltalsværdier.

Det har hele tiden været et salgsargument, at CPC kan forsynes med ekstra ROM-chips, der kan indeholde alt fra spil til tekstbehandlingsprogrammer og alternative programmeringssprog, og at disse kan startes fra BASIC med en dertil reserveret kommando. Denne kommando er imidlertid ikke kommet med i manualen. Kommandoen eksisterer dog i bedste velgående. I modsætning til alle andre kommandoer, så består den kun af kun eet tegn, der fremkommer ved at trykke alfakrølle-tasten sammentidig med SHIFT.

På skærmen vises der en lodret streg, der forresten også kan bruges til grafikformål. Denne streg fortæller computeren, at nu gælder det et andet område af hukommelsen. Dog ved den endnu ikke hvilken ROM. Derefter skal man angive en såkaldt LABEL, dvs. et navn, som kan genkendes af den aktuelle ROM-kreds. Da CPC'en i grundversionen kun er forsynet med BASIC-ROM, er det også kun BASIC-labelen, man må angive. Hvis man prøver kommandoen, vil man se at BASIC startes op forfra med meldingen BASIC 1.0. Alle data slettes iøvrigt. Et ROM-område kan godt indeholde flere LABELS, der kan arbejde som ekstra-kommandoer. Floppy-controller (se kapitel 10.1) indeholder en sådan kommandoudvidelse.

CPC's BASIC er envidere meget meddelsom. Der findes nemlig en funktion, der angiver adressen hvorfra en variabel er lagret i hukommelsen. Det er naturligvis ikke så interessant for en BASIC-programmør, men hvis man ønsker at "hoppe" på maskinsprogs-vognen, kan den være til stor nytte. Funktionen opnåes ved hjælp af alfakrøllen. PRINT &a giver altså adressen for variablen A (hvis den eksisterer, ellers fejlmeddelelsen IMPROPER ARGUMENT)!

En anden sær-kommando fremkommer ved DEC$(X,Y). Den nævnes kun en enkelt gang og kun som værende i famillie med BIN$. Den findes som selvstændig kommando i ROM, som det ses af kommando-tabellen i denne bog's tillæg. Dog findes der tilsyneladende ikke en tilhørende rutine i interpreteren. Måske har det været programmørens mening at indbygge kommando som modstykke til BIN$ og HEX$. Af en eller anden grund kom vedkommende på ideen med at anvende præfix'erne $ og &X.

3. HUKOMMELSEN

CPC-464 hører hjemme i gruppen af sværvægtere blandt hjemmecomputere, når det drejer sig om lagerkapacitet. Foruden de 64 K RAM er der 32 K ROM til rådighed. Yderligere kan ekstraudstyr udvide hukommelsen betydeligt. Hvordan det kan lade sig gøre, og hvordan man gør, skal det følgende kapitel belyse.

3.1. BESKYTTELSE AF HUKOMMELSE

I BASIC er der som allerede nævnt, 42.5 K RAM til fri benyttelse. Maskinkodeprogrammørerne (og de, der ønsker at blive det), har derimod ikke særlig meget plads at boltre sig på. BASIC-fortolker bryder sig faktisk ikke ret meget om maksinkodeprogrammer. Den bruger hukommelsen, som det passer den. Indtastes en ny variabel eller en ny BASIC-linie, så overskrives det frie område efter programmet af de nye data. Hukommelsen fyldes dog ikke ud nede fra og opefter. I starten ligger program-linierne, strengene lægges i bunden og imellem lægges de andre variabler på må og få. På et eller andet tidspunkt, må de forskellige områder nødvendigvis støde sammen midt i hukommelsen. Herved fremkommer fejlmeddelelsen:

MEMORY-FULL-ERROR.

Det er altså ikke på noget tidspunkt muligt at sige, hvilke bytes der er fri i hukommelsen.

Derfor må man reservere et antal bytes til maskinkodeprogrammer. Dette klares med kommandoen MEMORY. Slutningen på BASIC-hukommelsen er angivet med en pointer, med hvilken fortolkeren kan orientere sig under sit arbejde. MEMORY-kommandoen ændrer ganske enkelt denne pointer. Sættes den til en lavere adresse, så bliver BASIC-hukommelsen tilsvarende mindre.

Pointeren findes i adresserne &AE7B og &AE7C. Man kan alternativt ændre den med POKE-kommandoer, hvis man er med i gruppen: ”Hvorfor gøre dette lettere, når det kan gøres vanskeligere?”.

Kommandoen ”? HIMEM” opfylder den modsatte funktion. Den hiver os den modsatte retning på pointeren. Efter computerens opstart er den sat til 43903. Dette tal markerer slutningen på BASIC-hukommelsesområdet, der starter ved adresse 368.

MEMORY-kommandoen kan kun udføres ved værdier mellem 368 og 43903, ellers udskrives fejlmeddelelsen:

MEMORY-FULL-ERROR

Denne forhindring kan dog undgåes med nogle POKE-kommandoer, der ændrer pointeren.

Hvis man ønsker at reservere 256 bytes til et maskinkodeprogram, skal man ganske enkelt indtaste:

MEMORY 43647

Nu kan man lægge sine maskinsprogskommandoer fra adresse 43648. (MEMORY og HIMEM benytter sig af den sidste byte i BASIC-hukommelsen)!

BESKYTTELSE AF HUKOMMELSE

MED MEMORY-KOMMANDOEN KAN DEN ØVRE GRÆNSE AF BASIC-HUKOMMELSEN FLYTTES NEDEFTER OG FLYTTES TILBAGE IGEN.

BASIC-HUKOMMELSEN KAN BEVÆGES MELLEM ADRESSERNE 368 OG 43903.

MED HIMEM KAN DEN MODSATTE (NEDRE) GRÆNSE FLYTTES.

3.2. HVORDAN FUNGERER BANK-SWITCHING?

Som det er fremgået af de foregående kapitler, ligger RAM og ROM i nogle områder ved siden af hinanden (paralelt). Processoren kan kun kommunikere med et af disse områder af gangen. Der skal således være en mulighed for at kunne skifte mellem områderne. De adresser, der ankommer til adressebussen bliver analyseret i en slags LOGISK OMSKIFTER, hvorefter de flyttes til den aktuelle hukommelseskreds.

Denne omskifter kan påvirkes udefra, d.v.s. at processoren via IN- og OUTPUT-kommandoer (svarende til INP og OUT) kan bestemme hvorledes adresserne skal dekodes (læses). Nogen gange er det kun RAM-kredsene, der kommunikeres med, andre gange kobles RAM-kredsen fra adresse-bussen, der kobles til ROM. Yderligere, kan ROM-kredsene kobles til hver for sig. På den måde kan man nøjes med at holde operativsystemet aktivt, mens der kommunikeres med video-RAM.

Man kan også skifte til ROM fra BASIC. De fleste forsøg derpå resulterer dog i at computeren hænger op. Prøv det aligevel. OUT &7F82,&82 indkobler begge ROM-områder.

Som det ses, er omskiftning mellem hukommelsesområder ikke særlig anbefalelsesværdig fra BASIC. Det er faktisk ikke noget under, idet man ved denne BANK-switching "prakker" Z80-processoren en helt ændret hukommelse på. I mange tilfælde arbejder fortolkeren i RAM. Skifter man nu om til ROM, kan processoren ikke længere finde sit egentlige program. Følgerne er fatale for computeren.

3.3. UDLÆSNING AF ROM

Ofte kan det være nyttigt eller sågar nødvendigt at hente data fra ROM, f.eks. et karaktersæt (der ligger i området &3800 til &3FFF). Det er ikke muligt fra BASIC. PEEK-kommandoen kan kun bruges til RAM-udlæsning (omskiftning mellem RAM og ROM er jo knap så heldigt). I teorien ville det være muligt at lave et program i BASIC til udlæsning af ROM, men CPC's reaktion er næsten allergisk. Her følger under alle omstændigheder et program dertil fordi.....

1. princippet skal fremhæves....

2. det til delvis erstatning, medleverede maskinkodeprogram ikke mere skal forklares, da det næsten er en komplet oversættelse.

Her er BASIC-programmet:

```
10 OUT &7F82,&82
20 A=PEEK(X):REM X = DEN ØNSKEDE ADRESSE
```

I maskinsprog ser programmet således ud:

```
10 DATA &01,&82,&7F,&ED,&49
20 DATA &1A,&32,&7F,&AB,&C9
30 MEMORY &AB6F:FOR I=&AB70 TO &AB79
40 READ A:POKE I,A: NEXT:END
50 REM X = ØNSKEDE ADRESSE
60 CALL &AB70,X
70 A=PEEK(&AB7F):RETURN
```

Linierne 10 til 40 initialiserer programmet, dvs. her flyttes maskinkodekommandoerne til hukommelsen (via POKE-løkken). Denne del gennemkøres kun en gang i starten. Derefter er maskinkodeprogrammet blevet overflyttet til hukommelsen og kan udføres. Det er værdierne i DATA-linierne, der er det egentlige maskinkodeprogram (10 bytes).

Hvis man ønsker at læse en byte ud fra ROM, så skal programmet have adressen at vide. Adressen ligger i variabelen X. Via GOSUB 60 startes selve programmet op. Linie 60 overtager kaldet med CALL &AB70,X. Da et maskinkodeprogram ikke, sådan uden videre, kan overføre data til et BASIC-program, hjælper vi til med dette ved at lægge den søgte værdi over i en del af hukommelsen, der kan læses med PEEK (&AB7F) fra BASIC. Man kan naturligvis ændre både linienumre og variabelnavne, som man ønsker det.

3.4. HUKOMMELSESUDVIDELSER

I reklamerne/annoncerne gøres der ofte opmærksom på, at CPC's hukommelse kan udvides via indsættelse af ekstra kredse. Det kunne f.eks. være ROM's med lagrede programmer: Tekstbehandling, spil og sprog. Det er også muligt at udvide selve RAM.

Man skal huske, at hukommelseskortene skal tilsluttes via adressebussen. Man kan så udnytte den ekstra RAM ved BANK-switching, svarende til omskiftningen mellem RAM og ROM. For Z80 er der ikke den store forskel. Det er kun tallene i OUT-kommandoen, der skal ændres tilsvarende.

Floppy-drevet har en tilsvarende udvidelses-ROM, der indholder de for betjeningen nødvendige kommandoer.

4. TRICKS TIL SKÆRMBILLEDET

Når det gælder grafik, hører CPC 464 ligeledes til blandt eliten. De indbyggede BASIC-kommandoer tillader brug af særdeles avancerede skærmbilleder. Der findes nogle skjulte muligheder, som vi vil afsløre her:

4.1. STYRING VIA CHR$-KOMMANDOER

I manualens kapitel 9 findes en tabel over styretegn. Disse tegn kan anvendes som ekstra kommandoer til styring af skærmbilledet, idet man kan kalde nogle af systemets underprogrammer direkte. Selv BASIC har ingen underprogrammer for den slags funktioner, men overlader dem til operativsystemet og nøjes med at "bestille" kørsler via bestemte styretegn.

Nedenfor er kun vist tegn, der kan bruges til nyttige funktioner. Der er f.eks. ingen grund til at skabe en LOCATE-kommando nr. 2, der endda er vanskeligere. Fælles for alle styretegnene er, at de kaldes via PRINT CHR$(X). PRINT-kommandoen sørger for, at tegnene lander ved den rigtige rutine i operativsystemet. CHR$-funktionen gør, at vi kan udtrykke koden/funktionen som et tal (og behandle den som et tal). Ellers ville der stå anført et grafiktegn her. Det sidste ville ikke fungere hver gang, da operativsystemet behandler grafik- og styretegn forskelligt!

Ved hjælp af styretegn nr. 1 (SOH) kan man gøre andre styretegn synlige. Som man sikkert er klar over, så bliver tegn (ASCII-koder), hvis numre er lavere end 32 ikke behandlet som tegn, der skal udskrives på skærmen, men derimod som styretegn til operativsystemet. Sætter man derfor CHR$(1) (ASCII-kode nr. 1) udskrives der et tegnmønster; f.eks. LF (NY LINIE, Line Feed) giver en pil, der peger nedad. Prøv selv med PRINT CHR$(1)CHR$(10). SOH kan kun påvirke den efterfølgende kode. For hvert tegn, der skal udskrives, må man altså sætte en ny SOH inden.

Styretegn nr. 2 har også en interessant effekt. Med STX (navnet på tegnet), kan man "slukke for cursoren". Dette lader sig dog kun gøre under afvikling af et program. Hvis BASIC arbejder i direkte mode, vil cursoren vende tilbage ved hver READY-meddelelse. Det følgende lille program viser brugen af ASCII-kode nr. 2:

```
10 PRINT CHR$(2)
20 INPUT "TEKST"; A$
30 PRINT CHR$(3)
```

Som det ses, forsvinder den ellers så trofaste cursor ved INPUT-kommandoen. PRINT-kommandoen i linie 30 bevirker præcis det modsatte. Her synliggøres cursoren igen (hvilket vi kunne spare os at gøre; det sker jo under alle omstændigheder, når programmet finder sin afslutning).

Nu kommer vi til de tegn, der får cursoren til at bevæge sig rundt på skærmen. CHR$(8) virker på samme måde som pil til venstre. CHR$(9) flytter cursoren en position mod højre, CHR$(10) nedefter, og CHR$(11) flytter cursoren opefter. Disse 4 koder kan i mange tilfælde være nyttige. På den beskrevne måde, kan man flytte cursoren en linie op og angive potens-tallet, og finde tilbage til sin egen linie igen. Ved første øjekast synes ASCII 12 at være overflødig. Med den kan man slette skærmbilledet som med CLS. Der åbner sig en hel ny verden af muligheder, når man kombinerer tekststrenge med disse koder. Prøv f.eks. denne:

```
A$=CHR$(12)+"OVERSKRIFT"
```

Hver gang man lader A$ udskrive med PRINT-kommandoen, slettes først skærmen, hvorefter teksten bliver udskrevet. Man kan kombinere alle styretegn med sådanne strenge.

Der er endnu et styretegn (for cursoren), der kan bruges på samme måde. Det er CHR$(13) (CR). Med den kan man flytte cursoren til næste linies første position. CR betyder Carriage Return, der svarer til vognretur på dansk. Den stammer fra dengang, hvor computere blev betjent via fjernskriver-terminaler. Med CR kunne computeren flytte fjernskriverens skrivehoved til en linies første position (kombineret med LF, blev papiret skubbet en linie frem samtidigt).

Funktionen af CHR$(16) simulerer CLR-tasten. D.v.s. at tegnet under cursoren slettes. Prøv selv:

```
10 CLS
20 LOCATE 20,10
30 PRINT"LOOP-TEST"
40 WHILE INKEY$="":WEND
50 PRINT CHR$(8)CHR$(8)CHR$(8)CHR$(8)CHR$(8)
   CHR$(16)
```

Start programmet op og iagttag virkningen af tegnene i linie 50 på programmets titel. Der er flere anvendelsmuligheder, end man umiddelbart tror.

Til at slette hele blokke af skærmbillededer, kan bruges styretegnene 17 til 20. Som eksempel kan man skifte de sidste tegn i linie 50 ud med CHR$(17). Når programmet startes op, vil alle tegn fra liniens start og frem til cursoren blive slettet. Dette ses ved at teksten forsvinder. Bruger man CHR$(18) får man samme virkning, men fra den modsatte side.

Koderne 19 og 20 virker på samme vis, blot sletter CHR$(19) alle tegn fra starten af skærmen og hen til cursoren. CHR$(20) sletter alle tegn fra cursoren og frem til slutningen på skærmen (nederste højre hjørne).

Den næste kode er en rigtig "godte". Med CHR$(21) kan man nemlig "slukke" for tekstskærmen, hvilket betyder at alle udskrivninger på skærmen, der ikke stammer fra grafikkommandoer undertrykkes. Det kan bruges i et program, der indlæser et PASSWORD:

```
  10 PRINT"INDTAST PASSWORD"CHR$(21)
  20 INPUT A$
  30 IF A$="PASSWORD" THEN GOTO 1000 ELSE NEW
1000 PRINT CHR$(6)
...
..
```

Efter udførelsen af linie 10 undertrykkes tekstudlæsninger, hvilket også indbefatter de tegn, der indtastes via tastaturet. Bortset fra det, forløber programmet normalt. Linie 1000 ophæver ASCII kode 21's virkning.

Koden SYN (ASCII 22) har en interessant effekt. Efterfølges den af kode 1, aktiveres TRANSPARENT-MODE. Denne funktion tillader at nye tegn skrives oven i gamle, uden at disse slettes. Det kan eksempelvis bruges til understregning af ord eller sætninger (noget, der ellers ikke er daglig kost for computere):

1. Først skriver man teksten, der skal understreges.

2. Cursoren bevæges tilbage til udgangsposition.

3. Kommandoen PRINT CHR$(22)CHR$(1); indtastes.

4. Nu kan man understrege (SHIFT + 8).

5. PRINT CHR$(22)CHR$(0) genetablerer normal mode.

Kode 24 er af særlig stor nytte. Den bytter ganske enkelt om på PAPER- og PEN-farve. Resultatet er INVERSE (negativ) tegnfremstilling. Bogstaver, der før var gule på blå baggrund, bliver nu blå på gul baggrund.

Den sidste kode, der skal nævnes, gælder igen cursoren. Med PRINT CHR$(30) flyttes cursoren "hjem", d.v.s. op i venstre hjørne.

En lille hage er der dog ved brugen af styretegn. Flyttes PRINT-kommandoen til grafikmode via TAG, er det kun grafiktegn, der kan udskrives. Kodernes funktioner er inaktive indtil TAGOFF.

CHR$-KODER (ASCII-KODER)

ASCII	FUNKTION
1	STYRETEGN UDTRYKT GRAFISK.
2/3	TEKST-CURSOR ON/OFF
7	BEEP
8	CURSOR EN POSITION MOD VENSTRE.
9	CURSOR EN POSITION MOD HØJRE.
10	CURSOR EN LINIE OPEFTER.
11	CURSOR EN LINIE NEDEFTER.
12	SLET SKÆRM (CLS)
13	CURSOR TIL START AF LINIE.
16	SOM CLR-TASTEN.
17	SLET LINIE INDTIL CURSOR.
18	SLET LINIE FRA CURSOR.
19	SLET SKÆRM INDTIL CURSOR.
20	SLET SKÆRM FRA CURSOR.
21/6	TEKSTSKÆRM ON/OFF.
22	TRANSPARENT-MODE ON/OFF.
24	INVERSE (NEGATIV).
30	HOME (CURSOR TIL ØVERSTE VENSTRE HJØRNE).

4.2. VIDEO-RAM SET INDEFRA

Hvad ved man egentlig om video-RAM. Vel ikke så meget endnu. Lad os lave om på det. Vi ved allerede, at punkterne på skærmen repræsenteres af bytes i video-RAM. Derfor må det første spørgsmål lyde:

HVORDAN ER PUNKTERNE ORDNET I HUKOMMELSEN?

For at kunne svare på det, må vi foretage et lille eksperiment. Inden bør vi for en "sikkerheds" skyld slukke og tænde computeren; så ved vi nemlig hvordan hukommelsen ser ud, når vi starter. Indtast derefter:

```
MODE 2

FOR I = 49152 TO 65535:POKE I,255:NEXT
```

Kommandoerne sætter alle video-RAM's bits til 1, og skærmen fyldes ud med de tilsvarende punkter. Efter at RETURN-tasten er blevet aktiveret, kan man se, at det først er de øverste punkter i alle tekstlinier, der fyldes ud. Derefter følger næste linie og så fremdeles, indtil hele skærmen er fyldt ud med punkter og danner en ubrudt flade. Forestiller man sig at alle 25 tekstlinier lægges efter hinanden til en enkelt lang linie, så vil den bestå af 8 vandrette linier af punkter (hvert tegn dannes i en matrix af 8 * 8 punkter i MODE 2). Hvis vi deler 16 Kbytes (skærmhukommelsen) med 8, så får vi resultatet 2 Kbytes eller 2048 bytes for hver linie. Lægges de 8 linier i forlængelse af hinanden, har man repræsentationen af bytes i hukommelsen. Endnu har vi ikke forklaret, hvordan det enkelte punkt fremstilles.

Når man, som i MODE 2, kun har to farver til rådighed (punkt tændt eller punkt slukket), kan vi nøjes med 1 bit pr. punkt (0 for slukket og 1 for tændt). Ved 640 * 200 punkter bliver det ialt 128000 bits svarende til 16000 bytes. 16 Kbytes er 16384 bytes, d.v.s. 384 bytes mere end nødvendigt. Det er der 2 forklaringer på. For det første, er det nemmere for processoren og TV-kredsen, at regne med det totale antal Kbytes end mellemværdier (vi vil jo også selv helst arbejde med hele tal fremfor brøker!). Desuden har processoren brug for nogle ekstra bytes under udførelse af SCROLL (kommer vi til senere). Derfor er der i slutningen af hver 2 K-blokke 48 bytes i reserve.

I MODE 1 er der 4 farver til rådighed. Det er derfor et enkelt punkt skal bruge 2 bits. Deres mulige kombinationer er:

00, 01, 10, 11

Da der nu skal bruges dobbelt så mange bits, er der kun kapacitet til at "tænde" halvdelen af punkterne; de bliver så dobbelt så brede. På den måde bruger 320 * 200 punkter stadigvæk 16000 bytes.

I MODE 0 er der hele 16 farver. For at kunne skelne mellem 16 forskellige punkt-typer, skal vi tilsvarende bruge en halv byte. Da antallet af bits igen fordobles, skal punktantallet således endnu en gang halveres. Vi har nu "kun" 160 * 200 punkter.

Selv nu har vi ikke løst hemmeligheden med video-RAM helt. I MODE 2 siger det sig selv, hvilke punkter, der svarer til hvilke bits. Den første byte i video-RAM bestemmer de første 8 punkters udseende. Med POKE 49152,X kan man "præge" et bestemt bit-mønster i hukommelsen (og på skærmen, i hvert fald så længe vi bliver i MODE 2). Prøv med værdierne 1, 2, 4, 8, 16, 32, 64, 128, 240, 15, 170 idet man prøver at forestille sig den binære fremstilling. Man vil se, at punkternes placering nøje svarer til den binære værdi.

I MODE 1 skal hvert punkt bruge 2 bits. Det ville være nærliggende at antage at hver punkt-bit's nabo-bit angiver farven, men desværre, er det ikke sådan. Derimod skelner Tv-kredsen mellem den venstre og den højre halvdel af en byte i video-RAM (en halv byte kaldes også en NIBBLE). Skriv den højre nibble under den venstre, som vist i eksemplet herunder (her er kun bit-numrene angivet):

```
7 6 5 4
3 2 1 0
```

Farverne angives ved, at man læser bits'ene parvis nedefra og opefter. Byten 11110000 vil give de 4 punkter farven 01. Ved 11111010 får punkterne 1 og 3 farven 3 (binært 11). 2. og 4. punkt får farven 1. Efterprøv med POKE-kommandoer.

Også MODE 0 er organiseret på samme måde. Bits'ene skrives under hinanden som vist her:

```
7 6
3 2
5 4
1 0
```

Som vi viste før, så svarer en byte i denne mode til 2 punkter på skærmen. Bit-kombinationerne skal igen læses nedefra og opefter, hvorved man får punkternes farver. Hvilke farver, kan man se i manualen (såfremt farverne ikke er fremkommet med INK-kommandoen).

4.3. GRAFIK I DET SKJULTE

I "professionel" programmering, vil man oftest tilstræbe at opbygge færdige skærmbilleder. Det vil sige, at man ikke viser tekst eller grafik på skærmen, før hele skærmen ligger "færdig" i hukommelsen. Eller man vil vise to uafhængige grafik-billeder. Det kan opnås på to måder. Hvis de to "billeder" ikke

skal skære hinanden, er det nok at lade det ene fremtræde lys på mørk baggrund (INK 2,0), og når det er komplet, så skifte til lys baggrund og mørk tegning (INK 2,24).

Som sagt så går denne metode kun an, hvis billederne er fuldstændigt adskilte fra hinanden og man ikke befinder sig i MODE 2.

Det går nemmere, hvis man lader CPC opbygge en ekstra skærm-hukommelse. Det koster desværre meget BASIC-hukommelse. Den ekstra skærmhukommelse må selvfølgelig ikke ligge et sted, hvor der skal foregå andet. Desuden kan video-RAM kun forskydes i intervaller af 16 K, hvilket vil sige at de mulige startadresser indskrænker sig til 0, 16384, 32768 og 49152. Den sidste mulighed er opbrugt allerede ved opstart af computeren. I området fra 0 til 16383 og 32768 til 49151 har såvel operativsystem som BASIC-fortolker lagret "livsvigtige" data. Der bliver således kun området fra 16384 tilbage.

Det betyder desværre også at BASIC-hukommelsen må "krympes" med mere end 16 K via MEMORY 16383. Metoden er god, men den egner sig kun til forholdsvis små programmer, hvilket igen er et spørgsmål om ydre lagringsmuligheder. Benytter man diskettestation kan man klare situationen ved at "skære" et stort program op i mindre dele, der så kan kalde hinanden efter behov.

Med CALL &BC06,&40 forskydes skærmhukommelsen til 16384. CALL &BC06,&C0 genetablerer udgangspositionen.

Alle grafik- og andre udlæsningskommandoer benytter sig kun af den skærmbilledhukommelse, der netop er aktiv. Man kan altså arbejde helt normalt.

Men det kan man også lave om på. I nogle tilfælde er det nødvendigt at bearbejde et skjult skærmbillede, mens det andet er synligt. Heldigvis, så findes der en adresse i RAM, hvori operativsystemet noterer sig hvilken skærmadresse, der er aktiv. Skal en tekst eller grafik udlæses, vil computeren ikke gå til de aktuelle adresser, men til den noterede. Dermed kan vi "narre" computeren til at arbejde med et skærmbillede, der ikke eksisterer. Adressen er &B1CB. Sålænge der står den samme værdi, der også er hægtet på CALL-kommandoen, forløber alt normalt.

CALL &BC06,&40:POKE &B1CB,&C0 kobler altså video-RAM om til start fra adresse 16384, hvorimod skærmudlæsningerne fortsættes i den gamle hukommelse.

POKE-kommandoen kan naturligvis også indtastes alene. Det er imidlertid vigtigt at værdierne &40 og &C0 bliver POKEd ind, idet computeren ellers kan hænge sig op.

Desværre nødsages man til at tage hensyn til en speciel ting. Når skærmen skal scroll'es, forskydes indholdet i video-RAM ikke byte for byte. Derimod ændres starten af video-RAM for tv-kredsen. I stedet for start i adresse 49152 f.eks. 53248 (for at nævne en vilkårlig værdi). Via en adresse-logik hægtes de agterudsejlede bytes igen på ved adresse 49152, så der fra et brugermæssigt synspunkt tilsyneladende ikke er sket nogen forandringer.

POKEr man i hukommelsen, vil det fremgå ved at punkterne ikke kommer til syne foroven til venstre i billedet, men et andet sted. For computeren (og brugeren) har det den fordel at scroll'ningen foregår hurtigere. P.g.a. dette særlige forhold anbefales det, at man i starten af BASIC-programmet angiver en MODE-kommando i hver skærmhukommelse. På den måde reset'es pointerne. Det er ikke nok at anvende CLS eller CLG. Endvidere må man passe på, at der ikke bliver brug for scroll-funktionen i programmet.

FORSKYDNING AF SKÆRMBILLEDE	
CALL &BC06,&40	SKIFTER VIDEO-RAM FRA OMRÅDET 16384 TIL 32767. (HUSK AT BESKYTTE MED MEMORY)!
CALL &BC06,&C0	SKIFTER TIL NORMAL TILSTAND.
I ADRESSE &B1CB ANGIVER OPERATIVSYSTEMET, HVILKEN VIDEO-RAM, DER ER AKTIV.	

4.4. LAGRING AF SKÆRMBILLEDER

I CPC-BASIC findes der en speciel kommando, til at SAVE nærmere definerede områder af hukommelsen. På den måde kan man også gemme skærmbilledets hukommelse på bånd, og hente den tilbage, når der er brug for den. Kommandoerne til dette lyder:

SAVE "!TV",B,49152,16384

LOAD "!TV",49152

Filnavnene kan man jo ændre, hvis man har lyst; husk at udråbstegnet skal blive stående ligegyldigt hvilket filnavn, der anvendes. Ellers vil operativsystemets meddelelser forstyrre skærmens indhold.

Det er også muligt at gemme dele af hukommelsen. Hertil skal man beregne start- og slutadresser for hver punkt-linie i hukommelsen. Kapitel 4.2 burde have udstyret os med den nødvendige viden om dette, således at det kan lade sig gøre uden de store vanskeligheder. Hver linie skal så gemmes med hver sin SAVE-kommando. Husk at det kun fungerer, hvis der ikke scrolles (jfr. problemet fra kapitel 4.3).

4.5. SCROLL

Fænomenet scrolling, kendes vel af enhver computerejer. Når cursoren kommer til skærmens nederste linie, forskydes indholdet opefter for at give plads til de nye linier. Her er der to ting, der er af betydning for BASIC-programmører. Først og fremmest måden, hvorpå der scrolles. For det andet, at scrolling ikke nødvendigvis behøver at foregå opefter. Hermed er bolden allerede givet op for et par interessante effekter.

Der kan scrolles som normalt, ved at styretegnet CHR$(11) (cursor op) udskrives. Den derved opståede linie kan så fyldes ud med PRINT-kommandoer. Men horisontal-scroll er heller ikke lukket land. Vi skal til denne effekt bruge to OUT-kommandoer, så vi kan ændre starten af video-RAM's OFFSET. Som vi så i forrige kapitel, er det ikke selve indholdet i hukommelsen, der ændres, men startadressen, der forskydes med en skærmlinie. Denne forskydning kaldes for OFFSET. Den kan heldigvis foregå i mindre skridt. Mindste skridt er 2 bytes, hvilket alt efter mode, svarer til et halvt eller helt tegn. På den måde kan man lave "sideværts scroll".

6845-videocontrolleren er i besiddelse af bestemte registre, hvori denne offset kan lagres. Man kan skrive i registrene med OUT-kommandoen. Læsning med INP er ikke muligt.

```
OUT &BC00,13:OUT &BD00, offset
```

Så længe der ikke er sket nogen scrollning vil værdien i denne offset være 0. Ved forøgelse af værdien, forskydes til venstre og omvendt til højre. Vil man forskyde fra midten mod både højre og venstre, anbefales det at man sletter skærmbilledet inden den første PRINT-kommando og ved hjælp af:

```
OUT &BC00,13:OUT &BD00,20
```

at scrolle alt til midten. Forskydning af en hel linie sker med en løkke, som den nedenfor viste:

```
OUT &BC00,13:FOR I=0 TO 40:OUT &BD00,I:NEXT
```

Den første OUT-kommando tjener til at udvælge det rigtige 6845-register. Den skal derfor ikke gentages inde i løkken. Da operativsystemet også gør brug af registre i videocontrolleren, kan det forekomme, at et registernummer ikke længere stemmer. Man kan altså ikke undvære OUT &BC00,13 inden hver DO-IT-YOURSELF-scroll!

SCROLLING

SCROLLING NEDEFTER KAN OPNÅES MED CHR$(11) I ØVERSTE LINIE AF SKÆRMEN.

SIDEVÆRTS SCROLLING FREMKOMMER MED:

OUT &BC00,13 : OUT &BD00, OFFSET

HVOR OFFSET ER BYTEANTALLET DELT MED 2.

4.6. SCROLL PÅ EN ANDEN MÅDE

Der findes en speciel effekt på CPC, der til forskel fra normal scrolling, hvor skærmhukommelsen ændres, tillader forskydning af billedet (inklusive rammen) i alle retninger. Også her spiller 6845-registeret en vigtig rolle. Det er ikke skærmhukommelsen, der påvirkes, men derimod video-signalet til monitoren. Dette video-signal indeholder foruden billedinformation til monitoren også de såkaldte synkroniseringssignaler, der angiver en linies (billedets) afslutning. Afsendes dette signal en smule forsinket, så kan monitoren ikke længere placere billedinformationerne det rigtige sted på skærmen. Resultatet bliver et forskudt billede.

6845 giver mulighed for at bestemme tidspunktet for dette synkronisations-signals afsendelse via OUT-kommandoer. Inden forskydning i horisontal retning skal kommandoen:

```
OUT &BC00,2
```

udføres. Med et andet signal:

OUT &BD00,X

indstilles synkronisationstidspunktet. X'es normale værdi er 46. X kan antage en hvilken som helst værdi mellem 0 og 63. Den komplette kommando-følge for en forskydning på en position mod venstre lyder således:

OUT &BC00,2:OUT &BD00,47

Ved vertikal forskydning lyder kommandoerne:

OUT &BC00,7:OUT &BD00,X

X må ligge i intervallet 0 til 38 og normalværdien er 30. Bliver X mindre forskydes billedet nedefter.

Disse tricks kan bruges til mange flotte effekter. Det ville være en smal sag at konstruere et spil, hvor man kunne kæmpe om at forhindre skærmbilledet i at vandre. Husk endvidere at man kan kombinere retningerne, der forskydes i.

BILLEDEFORSKYDNING	
HORISONTAL FORSKYDNING:	OUT &BC00,2:OUT &BD00,X
VERTIKAL FORSKYDNING:	OUT &BC00,7:OUT &BD00,Y

4.7. CURSORSTYRING ENDNU EN GANG

Kommandoen INKEY$ har den ulempe i forhold til INPUT-kommandoen, at der ikke optræder nogen cursor i forbindelse med udførelsen. Til alt held, kan man via BASIC selv bestemme, om cursoren skal være med eller ej. Det drejer sig om kald af to rutiner i operativsystemet. Her følger et lille programeksempel:

```
10 CALL &BB81:REM CURSOR ON
20 WHILE INKEY$="" :WEND:AFVENT TAST
30 CALL &BB84:REM CURSOR OFF
40 ...
....
```

Det tilsvarende kan ikke foregå i direkte mode, da operativsystemet overtager cursorkontrollen for hvert "READY".

CURSOR ON/OFF

CURSOR SYNLIG:	CALL &BB81
CURSOR USYNLIG:	CALL &BB84

---

5. GRAFIK

Vil man have andet end linier og punkter frem på skærmen, så ser det umiddelbart sort ud for CPC-BASIC. Der findes nemlig ikke nogen kommandoer for cirkler, firkanter og lignende. Heldigvis kan de manglende kommandoer simuleres med nogle meget simple underprogrammer.

5.1. GRAFIK-STYRETEGNET

I kapitel 4.1 har vi omtalt et styretegn, der har med højopløselig grafik at gøre. CHR$(23) aktiverer de forskellige pen-farver. I normal tilstand (inden CHR$(23) er blevet anvendt) sættes punkterne på skærmen, uanset hvordan denne i forvejen så ud på samme sted. CPC kan andet. Hvis man inden punktet sættes, indtaster kommandoen PRINT CHR$(23)CHR$(1), så XOR'es de nye punkter med de gamle. XOR sammenligningen har den fordel, at resultatet kun bliver 1, hvis udgangsbits'ene er forskellige. Således sættes kun et nyt punkt, hvis det tidligere punkt er forskelligt fra det nye.

Har man lavet en linie fra koordinatet 0,0 til 100,100 og gør det samme anden gang, dog med XOR, så slettes den samme linies punkter. Her er et eksempel:

```
10 MODE 2:PRINT CHR$(23)CHR$(1):REM XOR-MODE
20 FOR I=100 TO 200 STEP 2
30 MOVE 10,I:DRAW 100,I:NEXT:REM TEGNING AF KASSE
   NR. 1
40 FOR J=1 TO 2
50 WHILE INKEY$="":WEND
60 FOR I=130 TO 180 STEP 2
70 MOVE 60,I:DRAW 150,I:NEXT I,J:
   REM TEGNING AF KASSE NR. 2
```

Programmet er ligeud ad landevejen, og kræver derfor ikke nogen særlig forklaring. Der tegnes ialt 3 kasser: en stor og to mindre, hvoraf den sidste sletter sin forgænger. Prøv selv.

Man kan også anvende AND og OR. Begge fungerer som XOR-mode, men sammenligningen er anderledes. Ved AND sættes kun et punkt, hvis der tidligere var et punkt samme sted.Det kan udnyttes til brug af flere farver. Sætter man punkter med en anden farve oveni gamle punkter, så vil farverne kun

komme frem, hvor andre farver var sat til 0. Det kan gøres med PRINT CHR$(23)CHR$(2).

OR-mode, PRINT CHR$(23)CHR$(3) svarer til transparent-mode for tekster. Nye punkter sættes ganske enkelt oveni gamle. Selv hvis en ny farve har koden 0, slettes det gamle ikke.

GRAFIK-MODES	
NORMAL:	CHR$(23)CHR$(0)
XOR:	CHR$(23)CHR$(1)
AND:	CHR$(23)CHR$(2)
OR:	CHR$(23)CHR$(3)

5.2. KASSER OG REKTANGLER

De følgende afsnit vil indeholde en række underprogrammer, der kan anvendes som ekstra grafikkommandoer. Vi begynder med de nemmeste figurer: Kasser og rektangler.

En kasse skal, efter vores egen definition, være en flade omsluttet af fire linier, der sammen danner en firkant. Det, der skal til, er således fire linier tegnet med DRAW. Det gøres med følgende underprogram:

```
65500 REM KASSER: X1,Y1,X2,Y2,F
65501 MOVE X1,Y1:DRAW X1,Y2,F
65502 DRAW X2,Y2,F:DRAW X2,Y1,F
65503 DRAW X1,Y1,F:RETURN
```

Dette og de følgende underprogrammer er lavet således, at de kan hægtes i enden på egne programmer med MERGE. Yderligere benyttes der variabler som udtryk for de ønskede værdier.

For at kunne fastlægge en kasses udseende, er det kun nødvendigt at angive et punkt i øverste venstre hjørne og et punkt i nederste højre hjørne. Disse koordinater, skal inden underprogrammet kaldes, lægges ind i variablerne X1 og Y1 (øverste venstre punkt) og X2 og Y2 (nederste højre punkt). Tallet, der angiver farven lagres i variablen F.

Første linie i underprogrammet flytter grafikcursoren til kassens startpunkt og trækker den første streg. De nsæte 3 DRAW-kommandoer tegner de resterende strækninger. RETURN overgiver igen kommandoen til hovedprogrammet. Et typisk kald af rutinen kunne lyde:

```
X1=100:Y1=200:X2=200:Y2=100:F=1:GOSUB 65500
```

Det næste program skal tegne en rektangel, der egentlig kun adskiller sig fra kassen, ved at fladen mellem linierne også fyldes ud. Det gøres ved at tegne mange linier under hinanden indtil rektanglen har den ønskede størrelse. Her er programmet:

```
65505 REM REKTANGEL: X1,Y1,X2,Y2,F
65506 FOR II=Y1 TO Y2 STEP 2*SGN(Y2-Y1)
65507 MOVE X1,II:DRAW X2,II,F
65508 NEXT II:RETURN
```

Her er det ligeledes kun nødvendigt at angive 2 af punkterne. I FOR-kommandoen findes igen STEP 2, der er vigtig for XOR-mode. Yderligere skal STEP-værdien multipliceres med -1, hvis y2 er mindre end y1. Det klares af SGN(Y2-Y1).

I linie 65507 flyttes cursoren tilbage til den venstre side af rektangelen, og der tegnes en linie mod højre. For hvert gennemløb af FOR-NEXT løkken forskydes denne linie et punkt nedefter, så man efterhånden får tegnet en sammenhængende flade.

5.3. RUNDT OM SINUS OG COSINUS

De to næste rutiner kendes fra en lidt mere simpel version i manualen til computeren. Men de i længden temmeligt kedelige cirkler og skiver, kan faktisk udnyttes til lidt mere avanceret grafik.

Lad os holde lidt fast ved circle-rutinerne. I begge tilfælle skal centrum (x1,y1), radius (r1) og farven (f) angives. Efter kald af rutinen løber en FOR-NEXT-løkke cirkelen rundt (0-360 grader) og ved hjælp af sinus- og cosinusfunktionen beregnes afstanden til omkredsen, hvor punkterne skal sættes. En skive fremkommer ved at undlade at bruge PLOTR, og i stedet for lade radius indtegne cirkelen rundt.

Vælges radius meget stor, kan man komme ud for, at nogle punkter ikke fremstår i den valgte farve, men forbliver mørke. Det kan forhindres ved at

tilføje STEP 0.5 til FOR-NEXT-kommandoen (i enkelte tilfælde en endnu lavere værdi). Her er de to rutiner:

```
65510 REM CIRKEL
65511 DEG:R2=R1:FOR II=0 TO 359
65512 XX=COS(II)*R1:YY=SIN(II)*R2
65513 MOVE X1,Y1:PLOTR XX,YY,F
65514 NEXT II:RETURN
```

```
65515 REM SKIVE
65516 DEG:R2=R1:FOR I=0 TO 359
65517 XX=COS(II)*R1:YY=SIN(II)*R2
65518 MOVE X1,Y1:DRAWR XX,YY,F
65519 NEXT II:RETURN
```

Typiske kald er:

```
X1=320:Y1=200:R1=100:F=1:GOSUB 65510
```

og

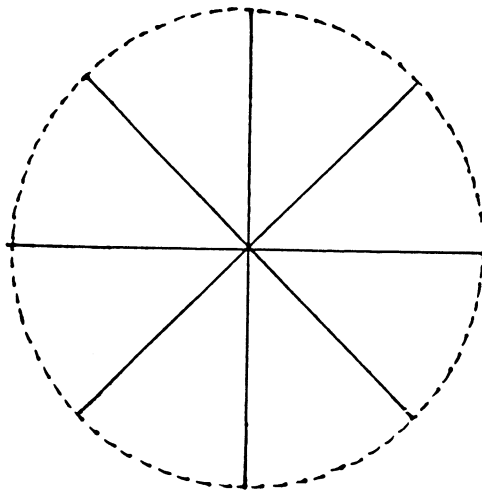
```
X1=100:Y1=100:R1=30:F=1:GOSUB 65515
```

Med kun få ændringer, kan man lave elipser. Tilsyneladende er de to radiusvariabler fra linie 65512 unødvendige, men hvis de indeholder forskellige værdier, tegnes der i stedet en elipse. Prøv en gang! R1 er radius i X-retning og R2 er radius i Y-retning. Linierne kommer til at se således ud:

```
65520 REM ELIPSE:X1,Y1,R1,R2
65521 DEG:FOR II=0 TO 359
65522 GOTO 65512
```

"GOTO 65512" bevirker at cirkelen tegnes med, da den indtil de to første linier er indentisk. På den måde sparer man lidt tasteri!

En lille ændring i skive-programmet gør os i stand til at tegne stjerner med variabel stråleantal. Teoretisk set er også skiven en slags stjerne, idet der til hvert af punkterne på omkredsen knytter sig en stråle til centrum (se billedet).



Figur 2: Stjerne.

Da strålerne ligger så tæt som de gør, vil de danne en tæt overflade. Det der skal til, er at formindske antallet af linier, hvilket kan gøres med en STEP-kommando. Man kan f.eks. nøjes med at tegne hver 10. stråle. For at fordele strålerne regelmæssigt (finde STEP-værdien), må vi dividere antallet op i gradtallet (360).

I linie 65527 finder vi igen en SPARE-GOTO. For at være sikker på, at alle strålerne optegnes, skal FOR-NEXT-løkken forlænges til 360.

Til opbygningen af stjernen, skal centrum, radius og farvekode, lægges ind i de kendte variabler. Variablen EC indeholder det ønskede antal stråler. Her følger igen en listning og et kald af rutinen for eksemplets skyld:

```
65525 REM STJERNE: X1,Y1,R1,EC,F
65526 DEG:R2=R1:FOR II=0 TO 360 STEP 360/EC
65527 GOTO 65517
```

Eksempel på kald af rutinen:

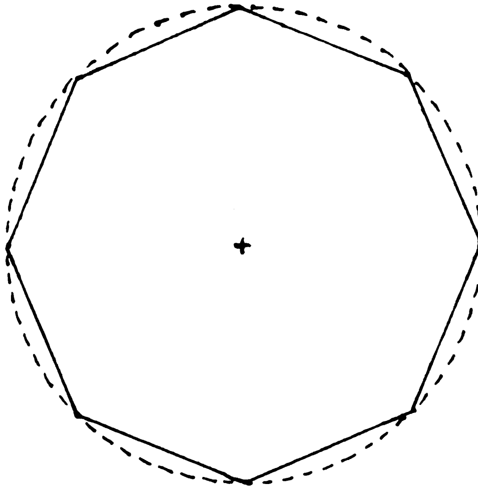
```
X1=100:Y1=100:R1=30:EC=12:F=1:GOSUB 65525
```

Den sidste underrutine tegner forskellige polygoner (flerkanter). For at forstå princippet i funktionen, kan man iagttage stjernen fra forrige program. Forbindes alle punkterne i omkredsen (det yderste punkt på strålerne) med hinanden via rette linier, så opstår der en polygon. Her kan vi drage nytte af vores rutine.

Som ved stjernen beregnes nogle punkter i en indbyrdes større eller mindre afstand. Fra hvert punkt tegnes der en ret linie til det næste punkt. Desuden skal cursoren, så at sige "sættes på sporet", da der ellers vil blive tegnet nogle højst udekorative ekstra linier. Her skal løkken også forlænges; denne gang til 370.

```
65530 REM POLYGON: X1,Y1,R1,EC,F
65531 DEG:MOVE X+R1,Y:FOR II=0 TO 370 STEP 360/EC
65532 XX=COS(II)*R1:YY=SIN(II)*R1
65533 DRAW X1+XX,Y1+YY,F:NEXT II:RETURN
```

Hvis man har lyst, kan man også lave æggeformede stjerner og polygoner, hvori der tilføjes en 2. radius. Der kan også tegnes halvkredse (buer) eller andre cirkeludsnit, hvis FOR-NEXT-løkken f.eks. "kun" løber fra 180 til 360. Lad bare fantasien få frit løb.



Figur 3: Polygon.

5.4. HVORFOR PIXELTEST

Umiddelbart er det svært at få øje på den dybere mening med kommandoerne TEST og TESTR. Hvis vi ønsker at kende et bestemt punkts farve, kan vi jo bare se på skærmen. Men der findes altså også anvendelser, der ville være utænkelige uden TEST.

I denne sammenhæng er det nærliggende at tænke på Commodore's SPRITE og Apple's SHAPE. For begynderen, der endnu ikke er så velbevandret i computerverdenen, skal det siges, at disse kommandoer omhandler figurer, der kan defineres på samme måde som bogstaver og tegn, hvorefter de ved en enkelt kommando kan vises på skærmen. Noget lignende kan vi lave med CPC-BASIC.

Vort program skal indeholde to funktioner. For det første skal en del af skærmen kunne kopieres til et andet sted. Princippet i dette underprogram er egentlig temmelig simpelt. Som ved kasse- og rektangel-programmet i kapitel 5.1, angives der to punkter, som skal afgrænse området, der skal kopieres. Derefter skal alle punkterne i "rektangelen" gennemkøres af to FOR-NEXT-løkker.

Resultatet af TEST-kommandoerne skal gemmes som PEN-farve-koder til de punkter, der bagefter skal kunne sættes. Var TEST-resultatet f.eks. et 0, skal der sættes et punkt med farvekoden 0, hvilket igen betyder, at punktet forbliver "slukket". På tilsvarende måde aflæses hele området pixel for pixel.

Den anden rutine skal kunne anbringe en figur på skærmen, der før var usynlig. Den kopieres altså ikke, men opbygges påny. Et eller andet sted, skal alle oplysninger om figuren være optegnet. I BASIC-programmer kan man nemmest gøre det i DATA-linier. Hver linie af punkter, der skal bringes på skærmen af underrutinen, skal ligge i en streng, hvori farverne til punkterne skal stå anført. Strengens længde angiver ligeledes, hvor mange punkter der skal ligge i en linie. Hvert punkt har sit eget tegn i strengen. Tegnet skal angive et hex-tal, der står for den tilsvarende farvekode. Rutinen skal også vide, hvornår figuren er færdigopbygget. Dette angives med en anden streng med længden 0. Møder underprogrammet denne streng, skal udførelsen bringes til standsning.

Til slut skal vi vide, hvor på skærmen figuren skal opbygges. De nødvendige koordinater overføres til rutinen via variabler.

```
 999 REM KOPIERING AF OMRÅDE:X1,Y1,X2,Y2,XS,YS
1000 FOR II=Y1 TO Y2 STEP -2:FOR JJ=X1 TO X2
1010 PLOT XS+JJ-X1,YS+II-Y1,TEST (JJ,II)
1020 NEXT JJ,II:RETURN

1049 REM PLOT OMRÅDE PÅ SKÆRM:XS,YS
1050 ZZ=0
1060 READ AA$:11=LEN(AA$):IF LL=0 THEN RETURN
1070 FOR II=0 TO LL-1:AA=VAL("&"+MID$(AA$,II,1))
```

```
1080 PLOT XS,II*M,YS-ZZ*2,AA:NEXT II:ZZ=ZZ+1:
     GOTO 1060
```

Det første underprogram skal kende koordinaterne til det område, der skal kopieres: x1,y1,x2,y2.

xs og ys angiver øverste venstre punkt i området.

Ved rutine 2, skal der ikke angives så mange værdier. Xs og ys skal indeholde kordinaterne til det øverste venstre punkt på skærmen. I PLOT-kommandoen bør man iagttage fordoblingen af ZZ. Denne variabel indeholder nummeret på den netop gennemløbne linie. Var dette ikke tilfældet, ville to af DATA-linierne blive plottet oveni hinanden (CPC fordobler Y-koordinaterne). Det skal der også tages hensyn til ved angivelsen af figurens højde. Altså: For hver linie, der skal plottes, skal der tælles to punkter videre i Y-retning. Det samme gælder for X-retningen. Her multipliceres med variablen M, hvis indhold (værdi) afhænger af MODE. Ved Mode 0, skal M indeholde værdien 4, da der her skal 4 X-koordinater til at fremstille et punkt. Ved Mode 1 er der tale om 2, og for Mode 2's vedkommende er det et 1-tal.

For at understrege programmets brug, har vi lavet et eksempel:

```
100 MODE 2
110 XS=320:YS=200:M=1:GOSUB 1050:END
120 DATA "100001000011000100000100000111111"
130 DATA "100001000100100100000100000100001"
140 DATA "100001001000010100000100000100001"
150 DATA "100001001000010100000100000100001"
160 DATA "111111001111110100000100000100001"
170 DATA "110001001100010100000110000110001"
180 DATA "110001001100010110000110000110001"
190 DATA "110001001100010110000110000110001"
200 DATA "110001001100010111110111110111111"
210 DATA ""
```

Disse linier skal tilføjes listningen. I DATA-linierne er angivet punkt-mønsteret for ordet HALLO. Som sagt, angiver et 1-tal et tændt punkt. 0 er en tom pixel. Hvis man arbejder i andre modes, kan man gøre brug af flere farver samtidigt.

5.5. KOORDINATSYSTEMER

Dette afsnit er først og fremmest interessant for matematikere og skoleelever blandt læserne. Dog vil jeg først bede læserne undskylde, at jeg forklarer noget så enkelt som et koordinatsystem. Skoleleverne har måske netop brug for "en anden måde", at se tingene på.

Hver gang man ønsker at vise en funktion grafisk, skal funktionsværdierne konverteres til skærmkoordinater. En del af dette arbejde kan overtages af ORIGIN-kommandoen. Som eksempel, vil vi gennemgå en sinuskurve.

Som bekendt ligger funktionsværdierne i en sinuskurve mellem -1 og +1. Derfor skal X-aksen skære skærmbilledet nøjagtig midt på.

Y-aksen sættes ved trigonometriske funktioner, for det meste i venstre side af skærmen (papiret). På den måde ligger nulpunktet i koordinatsystemet i skærmens koordinater (0,199). Indtaster vi nu kommandoen ORIGIN 0,199, så vil alle følgende koordinater rette sig efter det nye nulpunkt skabt af kommandoen. Plot 100,-1 sætter dermed et punkt umiddelbart under X-aksen.

Hvis man har eksperimenteret lidt med ORIGIN-kommandoen, og har glemt de tilhørende værdier, så kan disse hentes tilbage med følgende kommandolinie:

```
PRINT UNT(PEEK(&B328)+256*PEEK(&B329))
```

På den måde får man værdien for X-retningen. For at få den tilsvarende Y-retning, skal man ændre adresse &B32A og &B32B.

Hvis vi indsætter værdierne fra sinus-funktionen i grafikkommandoerne (f.eks. PLOT X,SIN(X)), så ville sinuskurven kun blive 1 pixel høj. Derfor ser den rigtige kommando således ud:

```
PLOT X,SIN(X)*199
```

Man kan selvfølgelig også multiplicere X-koordinatet med en sådan faktor, hvis området ikke stemmer oversens med grafikkoordinaterne.

Nedenfor findes et underprogram, der på baggrund af indtastede X- og Y-værdier, tegner en kurve på skærmen:

```
1000 INPUT "X-MINIMUM";XA
1010 INPUT "X-MAXIMUM";XE
1020 INPUT "Y-MINIMUM";YA
1030 INPUT "Y-MAXIMUM";YE
1040 CLG
1050 MX=(XE-XA)/639:X=-XA/MX
1060 MY=(YE-YA)/399:Y=-YA/MY
1070 ORIGIN X,Y
1080 MOVE XA/MX,0:DRAW XE,MX,0:REM Y-AKSEN
1090 MOVE 0,YA,MY:DRAW 0,YE/MY:REM X-AKSEN
1100 RETURN
```

De første 5 liniers opgaver burde være klare. I de to næste linier, beregnes målestok-forholdet (MX,MY) for X- og Y-aksen og koordinaterne til nulpunktet (X,Y).

Målestokken findes ved forskellen mellem minimum og maximum af området delt med antallet af mulige punkter. Forskellen angiver, hvor mange punkter der ønskes. Disse projiceres til de mulige punkter ved division.

Hver gang en funktionsværdi eller en X-værdi skal beregnes ud fra en skærmbilledekoordinat, skal dette kun angives via målestoksforholdet. Det sker således også ved beregning af de nye koordinater til nulpunktet.

Linie 1080 trækker Y-aksen og linie 1090 står for X-aksen. Her findes igen målestoksforholdet.

Programmet kan gøres "finere" ved, at man anvender TAG og PRINT til at angive betegnelser og akse-intervaller.

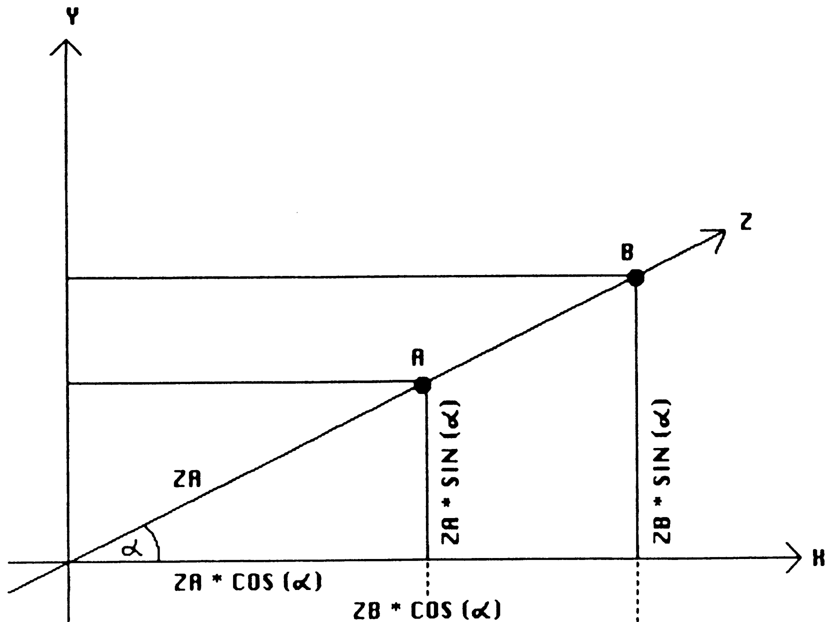
5.6. 3-D GRAFIK

Fremstilling af tredimensionale funktioner hører uden tvivl til den mere interessante del af grafikken; desværre også den vanskeligste. Alligevel, vil vi vise en "apertif" indenfor denne teknik.

Lad os kigge på koordinatsystemet. I modsætning til den sædvanlige funktionsfremstilling, er der nu 3 akser: X,Y og Z.

Z-aksen må nødvendigvis kunne vises "skråt" i rummet (se fig. 4). Vinklen kan vi selv bestemme.

Da vi ikke kan angive 3 dimensioner i PLOT-kommandoen, må vi omregne til 2 dimensioner. Det er det, der kaldes projektion. Vi anvender den enkle parallelle projektion, hvor der ikke findes et brændpunkt (linierne er jo, som navnet siger, parallelle).



Figur 4: Koordinatsystem til 3-D grafik.

Som det ses på billedet, skal et punkt, alt efter Z-koordinat forskydes mere eller mindre opad mod højre. På Z-aksen er der indtegnet to punkter med koordinaterne (0,0,ZA) og (0,0,ZB). Man kan se at projektionskoordinaterne kan beregnes ved addition af ZA*COS(A) og ZB*SIN(A). Formlerne ser således ud:

X=0 + ZA * COS(A)　　　Y=0 + ZA * SIN(A)

eller i standardform:

X=XA + ZA * COS(A)　　　Y=YA + ZA * SIN(A)

Da disse 2-D koordinater ikke altid stemmer overens med skærmkoordinaterne, indfører vi nogle såkaldte afstandsfaktorer SX og SY (se linie 100). Koordinatet bestemmes af ORIGIN 320,200 således at billedet vises i midten af skærmen.

Idet kun en af de tre koordinater beregnes, er der brug for to nestede FOR-NEXT løkker. Af grunde, vi ikke vil komme ind på her, er det en god ide at plotte bagfra.

I det nedenfor viste program bruges INPUT ved de værdier, der ofte ændres ved eksperimentering. Programmet arbejder i grafikmode 1, da det her kommer an på den størst mulige opløsning; og ikke flest kulører. Tast programmet ind og indsæt de følgende værdier:

w=45
sx=0.5
sy=0.5
xs=2
zs=20

```
  1 REM ++++++++++++++++++
  2 REM 3D-Grafik-Plotter
  3 REM ++++++++++++++++++
 10 INPUT "VINKEL";w:DEG: z1=cos(w): z2=sin(w)
 20 INPUT "X-LÆNGDE";sx
 30 INPUT "Y-LÆNGDE";sy
 40 INPUT "X-STEP";xs
 50 INPUT "Z-STEP";zs
 60 MODE 2:ORIGIN 320,200
 70 FOR z= 180 TO -180 STEP -zs
 80 FOR x= -180 TO 180 STEP xs
 90 y= sin(x)*cos(z)*100: REM Funktionsterm
100 bx=sx*(x+z*z1): by=sy*(y+z*z2)
110 PLOT bx,by,1
120 NEXT x,z
```

Ved kørsel af programmet vil man opdage, at der er nogle punkter, der burde være optisk skjulte af foranliggende billedelementer. Det kaldes PASTE. Problemet kan løses ved at slette alle punkter under de netop satte. Linie 110 skal se således ud:

```
110 PLOT bx,by,1: MOVE bx,by-2:DRAW BX,-200,0
```

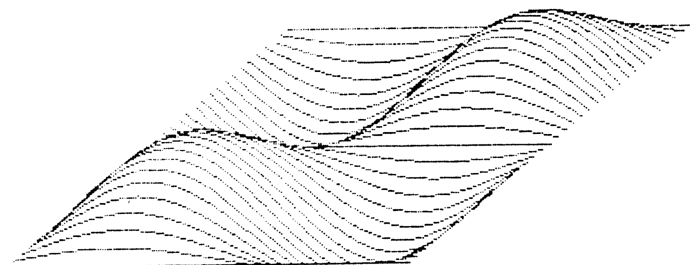
Vi vil forklare det med et eksempel. Vi går ud fra, at vores funktion angiver en fuldstændig jævn flade. Denne flade vil være smallest for oven; det forreste punkt på figuren vil ligge under det bageste punkt, og kan dermed ikke dække andre af figurens punkter. Lad os antage at Y-koordinatet til den for-

reste linie af punkter forhøjes med 1. Så vil den netop dække den ovenfor liggende linie. Forhøjer vi koordinaterne endnu en gang, vil tilsvarende flere linier dækkes. Deraf ses det, at alle punkter, på den til enhver tid nederst liggende linie, skal slettes. Det gælder også, hvis figuren viser en kurve eller lignende. Den bedste måde at forstå det på, er nok at iagttage sletningen, idet grafikken opbygges.

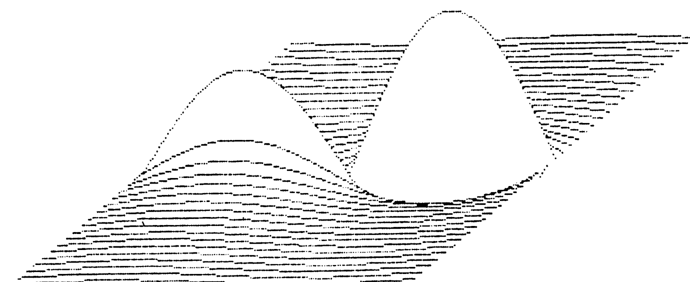
Endnu et par forklaringer til belysning af programlinierne. Linie 10 beregner vinklen W's cosinus- og sinusværdier. Da disse værdier altid er de samme, kan de lægges ind som konstanter, hvilket sparer meget processortid. (det er de langsommeste beregninger på computeren). Tallene i FOR-NEXT løkken (linie 70/80) kan ændres alt afhængig af, hvor stor en del af funktionen, man ønsker tegnet.

Multiplikationen i linie 90 (*100) skyldes, at funktionsværdierne altid bliver mindre end 1. Denne værdi skal strækkes; men graden svinger fra funktion til funktion.

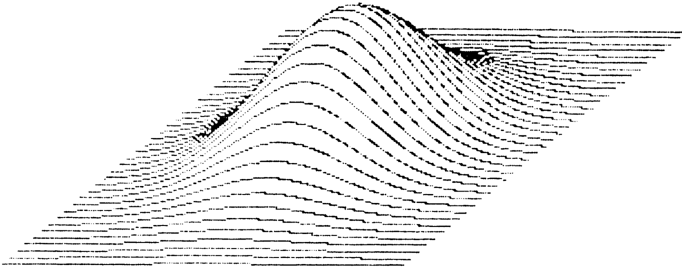
Prøv at anvende forskellige STEP-værdier, strækningsfaktorer og funktioner.



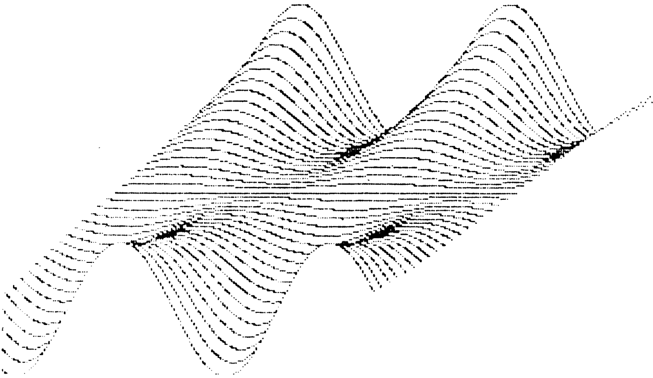
*Figur 5: y = sin(x) * sin(z) * 140*



*Figur 6: y = sin(x) * 2 / (z / 8 + 0.5) * 80*



Figur 7: $y = exp(-(x * x + z * z) / 2) / sqr(2 * \pi) * 300$



Figur 8: $y = sin(x/30) * (exp(z/90)-1/exp(z/90)) * 10$

6. NYTTIG GRAFIK

Med hjælp fra forrige kapitel, kan man lave mange smukke grafikbilleder. Men nogen nytteværdi kan man vel ikke sige, at de har. Det vil vi opveje med et par anvendelser fra den professionelle verden. Hører man til den del af computerejere, der vil bruge maskinen i erhvervsmæssig sammenhæng, så vil kapitel 6.1 være af særlig interesse. Maleprogrammet i kapitel 6.2 er beregnet for hobbyfolk og kunstnere.

6.1. DIVERSE DIAGRAMMER

Næsten alle computerejere har, på et eller andet tidspunkt, set brugen af diagrammer til bl.a. visning af valgresultater eller landets betalingsbalance. Man kan også bruge bjælkediagrammer til at vise salgstal, eller den gennemsnitlige nedbør i juni måned. Kan vi ikke også lave noget sådant?

Det kan vi, og på CPC er det en temmelig overkommelig sag. Faktisk er sådan en bjælke ikke andet end en rektangel, som den vi allerede har brugt i kapitel 5.1. Vi skal altså bare have lavet vore værdier om til koordinater. Omregning af de faktiske værdier er gennemgået i kapitel 5.5. Vi vil anvende den samme metode her.

Til X-området skal der ikke længere angives minimum og maximum, men kun antallet af bjælker, der skal udskrives (tegnes) i variablen XE. Dette tal skal multipliceres med 10 for at reservere punkterne for bjælkebredden (en bjælke skal jo være bredere end en enkelt pixel; se linie 60040).

I array W (0...XE) skal de enkelte værdier anføres. Det størst mulige tal beregnes i den første FOR-NEXT-løkke. Da nummereringen af array-elementerne starter ved 0, og vi mennesker har for vane at starte med 1, skal der trækkes 1 fra antallet i XE.

Ved målestoks-formlerne er de forringede punktkonstanter og den deraf følgende formindskelse af diagramfladen iøjnefaldende. De er nødvendige for at skaffe plads til værdi-beskrivelser.

Idet vi går ud fra, at salgstal og lign. aldrig bliver negative, kan vi forlade os på minimum i begge intervaller (XA og YA).

Linie 60040 sletter skærmen og sætter koordinatens nulpunkt til 50,30. På den måde har vi plads fri til kurvebeskrivelser. I linie 60050 tegnes afgrænsning.

I linierne 60060 og 60070 finder man rektangel-programmet i en noget ændret form. Da vi gerne vil kunne tegne lodrette bjælker, er det både hurtigere og mere fordelagtigt også at tegne linierne lodrette med DRAW.

Nogle variabler er blevet erstattet med aritmetiske udtryk. Konstanten 10 angiver antallet af enheder. Den egentlige bjælkebredde er +8. Hvis man ønsker mindre eller større mellemrum, behøver man kun at ændre denne konstants værdi.

```
60000 REM BJÆLKEDIAGRAM:XE,W(0...XE-1),F
60010 XE=XE-1:YE=0
60020 FOR II=0 TO XE:YE=MAX(YE,W(II)):NEXT
60030 MX=(XE+1)*10/584:MY=ZE/364
60040 CLG:ORIGIN 50,30
60050 MOVE -5,0:DRAW -5,YE/MY+5:MOVE -5,0:
      DRAW (XE+1)*10/MX+5,0
60060 FOR II=0 TO XE:FOR JJ=II*10/MX TO (II*10+8)/MX
60070 MOVE JJ,0:DRAW JJ,W(II)/MY,F:NEXT JJ,II
60080 RETURN
```

Den anden diagrammeringsmetode ser tilsyneladende noget mere kompliceret ud, men også her kan vi vende tilbage til de tidligere underprogrammer. Vi er her nået frem til den såkaldte LAGKAGEGRAFIK, hvor værdierne, der skal vises grafisk, er angivet som større eller mindre skiver af en lagkage (cirkeludsnit).

Fremgangsmåden ved opbygningen af et sådant diagram er meget enkel. En efter en tegnes cirkeludsnit, hvis størrelse afhænger af den værdi, der skal repræsenteres. For at kunne adskille de enkelte segmenter, tildeles de forskellige farver.

I forbindelse med beskrivelsen af programmering af et segment, bedes man blade tilbage til kapitel 5.3. I listningens linie 65516 møder man kommandoen FOR I=0 TO 359. Som det blev nævnt, aftegnes der her en omkreds på 360 grader. Hvis vi i stedet giver kommandoen FOR I= 0 TO 90, så får vi en kvart cirkel. Som det ses, kan start og slutpunkterne frit vælges efter behov.

Der optræder dog et andet problem. Vores cirkel tegnes modsat uret. Det kan imidlertid ændres ved at man bytter om på SIN- og COS-funktionerne til koordinaterne. På den måde starter "omløbet" ved klokken 12.00 og bevæger sig med uret rundt. Her er listningen:

```
60100 REM LAGKAGEGRAFIK : X1,Y1,R,XE,W(0...XE-1),
      F(0...XE-1)
60110 WB=0:DEG:FOR II= 0 TO XE-1
60120 WA=WB:WB=WB+W(II)*3.6:S(II)=(WB-WA)/2
60130 FOR JJ=WA TO WB
60140 XX=SIN(JJ)*R:YY=COS(JJ)*R
60150 MOVE X1,Y1:DRAWR XX,YY,F(II):PLOT R0,0,1
60160 NEXT JJ,II:RETURN
```

Dette underprogram forventer ligeledes at få nogle parametre at arbejde med (se liste i 60100). X1 og X2 angiver cirkelens centrum. R er radius. Funktionen af XE er også forblevet den samme. De enkelte værdier, der angiver udsnittene, er alle lagret som procenttal i array W(0...XE-1). Skal et felt fylde 20% af kredsen, skal det tilsvarende tal i array'en være 20. Der kan yderligere angives en farve for hvert udsnit (F(II)).

Dette program består ligeledes af to nestede løkker. Den yderste sørger for, at der tages hensyn til alle værdier. Den inderste gennemgår alle segmenternes vinkler.

I linie 60120 beregnes start- og slutvinkler. Desuden har vi flottet os lidt. I array S(0...XE-1), der skal DIMensioneres inden underprogramet kaldes, lagres vinklen, der halverer segmentet. Hvis man vil lægge bogstaver/tegn/tal på de enkelte udsnit, kan det lade sig gøre ved hjælp af TAG. Her skal koordinaterne også beregnes med COS og SIN, dog skal radius forhøjes således at skriften ikke rammer kredsen.

I skriverutinen er det kun PLOTR-kommandoen, der er ny. Den tegner en kant rundt om diagrammet i tilfælde af at segmentfarven er sort. Gøres dette ikke, kan man nemt få det indtryk at grafikken forestiller en stjerne. Resten af programmet er gennemgået tidligere.

Pas på at summen af værdier i array'en W bliver 100, ellers kan det ske at der kun tegnes et cirkeludsnit.

6.2. PROGRAMMET FOR KUNSTNERE

Det følgende program er tænkt anvendt af mennesker, der kan lide at eksperimentere med grafik og evt. ønsker at designe grafikbilleder til brug i egne programmer. Det er alt for besværligt at "tegne" hele tegninger med PLOT og DRAW kommandoerne. Noget nemmere er det at flytte cursoren rundt på skærmen og lade den sætte eller slette punkter. Det kan lade sig gøre med det følgende program:

```
 10 MEMORY 16383:CALL &BC06,&40:MODE 2:X=320:
    Y=200:M=1
 20 A$="":WHILE A$="":A$=INKEY$:WEND:PLOT X,Y,F
 30 IF A$=CHR$(240) AND Y<398 THEN Y=Y+2
 40 IF A$=CHR$(243) AND X<639 THEN X=X+1
 50 IF A$=CHR$(241) AND Y>1 THEN Y=Y-2
 60 IF A$=CHR$(242) AND X>0 THEN X=X-1
 70 IF A$=CHR$(224) THEN F=1:BORDER 24:M=0
 80 IF A$=CHR$(13) THEN F=0:BORDER 24,0:M=0
 90 IF A$=" " THEN M=1:BORDER=0
100 IF A$="S" THEN CALL &BC06,&C0:MODE 2:INPUT
    "FIL  NAVN";F$:SAVE F$,B,16384, 16384:CALL
    &BC06,&40:GOTO 20
110 IF A$="l" THEN CALL &BC06,&C0:MODE 2:INPUT
    "FILNAVN";F$:LOAD F$,16384: CALL &BC06,&40:
    GOTO 20
120 IF A$="C" THEN CLG:GOTO 20
130 IF A$="X" THEN CALL &BC06,&C0:END
140 IF M=1 THEN F=TEST(X,Y)
150 PLOT X,Y,1:GOTO 20
```

Til grafikken anvendes den anden skærmhukommelse fra 16384. På den måde forhindres det, at det langsommeligt opbyggede skærmbillede forstyrres af tekstindtastninger.

Ved tegn er der 3 modes, der angives ved forskellig kantfarve. Den første mode kunne passende kaldes for cursormode, da man her kan flytte en cursor rundt på skærmen uden at ændre evt. underliggende punkter. Her er kanten sort. Er rammen lys, sættes punktet under cursoren. Blinker rammen slettes punktet. Hver mode er valgbar med sin egen tast. I cursor-mode er det mellemrumstasten. Punkter sættes ved tryk på COPY, og ENTER sætter programmet til slette-mode.

Men det er ikke alt. Med S-tasten kan man gemme billedet på tape. Herved skiftes der til den normale skærmhukommelse, hvori man så kan angive et filnavn. Så saves der (se linie 100). På samme måde fungerer LOAD-mode, der kaldes ved tryk på L (linie 110). I begge linier findes der MODE-kommandoer, hvormed skærmen slettes og scroll-effekten omgåes under hukommelsesskift.

C-tasten sletter skærmen. Til slut har vi X, der afslutter programmet uden at

forstyrre skærmen. Finder man ud af, at man vil foretage en ændring, kan man indtaste følgende:

CALL &BC06,&40:GOTO 20

Linie 10 initialiserer skærmen og sætter startkoordinaterne. I næste linie hentes et tegn via tastaturet og det nye punkt sættes (resultatet af løkke-gennemløb). Ved tryk på tastaturet vælges en af flere modes.

Variablen F angiver farven for PLOT-kommandoen i linie 20. Er cursor-mode aktiv (M=1), så fastlægges F af punktets tidligere farve. På den måde kan skærmbilledet holdes uforandret. PLOT-kommandoen er lagt efter INEKY$-kommandoen for at holde grafik-cursoren på skærmen så længe som muligt. Cursoren selv skyldes PLOT-kommandoen i linie 150.

Dette korte program er slet ikke tænkt som en slags super-software. Det er meningen, at man, efter at have forstået princippet i programmet, kan fortsætte med større programmer på egen hånd. Forøvrigt kan billederne, der laves med programmet via LOAD "!navn",49152 hentes over i den normale skærmhukommelse.

7. INTERRUPT-PROGRAMMERING

Interruptprogrammering er en af de mest fremragende egenskaber hos CPC-BASIC. Manualen har ikke ofret nogen særlig omtale af denne særlige programmeringsmetode. Det vil vi råde bod på her.

7.1. HVORDAN FUNGERER EN BASIC-INTERRUPT?

I princippet er der ikke den store forskel på maskinkode- og BASIC-interrupt. I begge tilfælde leverer en integreret kreds (for det meste TIMEREN) et såkaldt interruptsignal, der afbryder det netop kørende program (dog færdigafvikles en evt. kommando) og der udløses et hop til en underrutine.

Ved Z80 forløber alt dette på hardware-basis, hvilket betyder at det ikke er programmel, men derimod specielle kredse, der klarer dette arbejde. I BASIC er det spidsfindige maskinkodeprogrammer, der står for udførelsen. Desuden kan et BASIC-interrupt kun udløses af en af de 4 timere (0-3), og ikke af en kreds.

Står et sådant signal for tur, undersøges det først om en evt. påbegyndt kommando skal færdigudføres. Således er det ikke muligt, at foretage interrupt under udførelse af en INPUT-kommando eller anden tidsafhængig kommando. Det kan også forekomme at interruptrutinen er spærret ved hjælp af en DI-kommando (Disable Interrupt). I begge de to sidste tilfælde vil BASIC dog mellemlagre interrupt-anmodningen og hente den til udførelse efter frigivelse.

Endelig undersøger den, hvilken rutine, der er ansvarlig for den netop aktuelle timer, og starter den som et normalt underprogram.

Men dermed er det ikke slut med på antallet af opgaver, som CPC står for i forbindelse med interrupt. Har man prøvet at skifte farve med INK-kommandoen, så udløses farveskifte af en speciel timer. Timeren kan findes ved hjælp af kommandoen SPEED INK. Hver gang der skal skiftes farve, ændrer operativsystemet en byte i kredsen, der styrer farverne.

Foruden dette, skal også tastaturet aflæses. Det sker tilsvarende ved brug ef et interrupt. Dette interrupt udløses hver 1/50 sekund, hvorved den netop trykkede tast's kode sendes til processoren. Der sker således hele tiden en mængde forskelligt inde i en computer.

7.2. INTERRUPT-KOMMANDOER

Efter denne "snak" vil vi kigge lidt på de enkelte interrupt-kommandoer. Her kommer først de to kommandoer, der udløser BASIC-interrupts:

AFTER

og

EVERY

After står for "EFTER" og every for "HVER". Kommandoerne bevirker næsten det samme; endog syntaksen er fælles:

AFTER X,Y, GOSUB Z

EVERY X,Y, GOSUB Z

Kommandoerne starter en timer, næsten på samme måde, som man stiller et vækkeur. I modsætning til det larmende vække-værktøj så skal der kun angives hvor mange 50. dele sekunder, der skal gå inden interruptet udløses. Denne værdi kan angives i variablen X.

AFTER 200....

udløser således et interrupt i løbet af 200/50 = 4 sekunder. Således er vi fremme ved den eneste forskel mellem AFTER og EVERY. En AFTER-kommando resulterer kun i et enkelt interrupt, mens EVERY sætter timeren til repetition, dvs. interruptet udløses igen og igen i faste intervaller. Således kan man f.eks. hver 10. sekund aflæse en måleværdi, der så kan bearbejdes i samme åndedrag.

Den andet parameter (Y) angiver hvilken af de 4 timere, der benyttes. Jo højere nummer, des større betydning har interruptet. Timer 3 kan afbryde timer 2, men det modsatte kan ikke lade sig gøre. Kommandoens sidste del, angiver hvilken linie, der skal hoppes til ved udløsning af interrupt.

Kommandoerne DI og EI er hurtigt forklaret. DI er en forkortelse for DISABLE INTERRUPT, der betyder, at der ikke længere kan udløses interrupt. Selv en timer med højere prioritet kan intet stille op. Først når enten RETURN eller EI gives, er interrupt tilladt igen. EI står for ENABLE INTERRUPT, dvs. tilladelse til udførelse af interrupt. Di og EI bruges mest i rutiner, der arbejder med grafik-kommandoer, for at forhindre at grafikcursoren ændres uønsket ved et andet interrupt.

Til uddybelse af forståelsen følger her et lille programeksempel med prioriteter og DI. Det indeholder to interruptrutiner; hovedprogrammet (linie 30) består af en endeløs løkke.

```
  1 CLS:PRINT T;" SEKUNDER"
  2 LOCATE 6,5:PRINT "SEKUNDER"
 10 EVERY 50,0 GOSUB 100
 20 EVERY 50,1 GOSUB 200
 30 GOTO 30
100 T=T+1:LOCATE 1,1:PRINT T
101 WHILE INKEY$=" ":WEND
102 RETURN
200 X=X+1:LOCATE 1,5:PRINT X
201 RETURN
```

Begge interruptrutiner forhøjer hvert sekund en tæller med 1. Rutinen fra linie 200 har højere prioritet, da den udløses af timer 1. Underprogrammet fra linie 100 behøver derimod længere tid til udførelse, da der afventes et tastatur-tryk (linie 101). Prøv at lade programmet køre, og læg mærke til, at den første rutine bestandig afbrydes af den anden.

Indføjer man nu kommandoen DI i linie 100 inden "t=t+1", så afbrydes alle andre interrupts indtil RETURN udføres. Når programmet startes op, vil også den anden sekundtæller først aktualiseres efter tastatur-tryk, da den anden interruptrutine må afvente frigivelse, via RETURN, i slutningen af første rutine.

Den sidste rutine i denne sammenhæng hedder REMAIN. Den fører et dobbeltliv, da den har to mulige former:

1. REMAIN Y

2. PRINT REMAIN (Y) eller A=REMAIN (Y) o.s.v.

I alle tilfælde bevirker den, at timeren Y reset'es; der kan ikke længere udløses interrupt. Indsættes REMAIN som funktion (vist i 2. form), så angiver den desuden hvor mange 1/50. sekunder der er tilbage indtil næste afbrydelse.

7.3. IDEER TIL INTERRUPT-PROGRAMMERING

Det hyppigst nævnte eksempel for et interruptprogram er et ur, der fortløbende ajourføres på skærmen. Hvert sekund (EVERY 50...) kaldes en underrutine, der forhøjer sekundtallet med 1. Den eneste ulempe er, at uret går i stå ved brug af kassettebånd, da kassettekommandoer standser timeren.

Hvad med en gættekonkurrence, hvor det ikke kun gælder om at finde det rigtige svar, men også at gøre det på den kortest mulige tid? Forløbet kunne se således ud:

1. Deltageren ved tastaturet stilles et spørgsmål.

2. Der gives flere svarmuligheder, hvor hver er angivet med kendetal. Tallet kan aflæses via INKEY$.

3. Via en AFTER-kommando afbrydes INKEY$-løkken. Der gives alt efter resultat og tid et passende antal point.

Punkt 2 kan ikke programmeres med INPUT, da interrupt som sagt ikke kan udløses under udførelse af en kommando.

En anden mulighed er at udskifte to karakterer på skærmen periodisk, og således simulere en bevægelse. Det første tegn kunne f.eks. vise en PAC-MAN med åben mund, og det andet en med lukket mund. Når der via en interruptrutine skiftes regelmæssigt, ser det ud som om figuren laver tyggebevægelser.

Eller en bil kunne køre over skærmen. Der findes i alle tilfælde rige muligheder for udnyttelse af interrupts. Brug fantasien.

8. SOUND

Mange fagfolk betegner CPC som en supercomputer. En af grundene er uden tvivl muligheden for brug af hvid støj. Her er nogle anvendelser:

8.1. MINI-SYNTHESIZER

ENT- og ENV-instruktionerne tilbyder lydprogrammøren et utal af muligheder for at ændre en tone eller simulere musikinstrumenter. Alle faktorer undtagen klangfarve kan forandres. I det følgende vises et program, hvormed alle parametre kan ændres med tryk på taster. Det giver mulighed for effektive eksperimenter med indhylningskurver (ENVelope).

```
10 MODE 2:WINDOW #1,2,46,2,7:REM MENU- OG
   ARBEJDS-VINDUE
20 WINDOW #2,1,40,9,23:REM ENV-VINDUE
30 WINDOW #3,42,80,9,23:REM ENT-VINDUE
40 MOVE 0,272:DRAW 639,272
50 MOVE 0,32:DRAW 639,32
60 MOVE 320,32:DRAW 320,272
70 MOVE 368,272:DRAW 368,399
80 LOCATE 49,2:PRINT"MINI SYNTHESIZER VERSION
   1.0"
90 LOCATE 49,4:PRINT CHR$(164)" 1985 DATA-BECKER
   GMBH"
100 LOCATE 49,6:PRINT"SKREVET AF: HANS J. LIESERT"
110 LOCATE 2,25:PRINT"1 = STEPVÆRDI
    2 = STEP-STØRRELSE 3 = PAUSELÆNGDE."
120 LOCATE #2,3,2: PRINT #2, "ENvelope Volume" CHR$ (10)
130 PRINT #2,"   1 2 3"CHR$ (10)
140 PRINT #2," 1 0 0 0"CHR$ (10)
150 PRINT #2," 2 0 0 0"CHR$ (10)
160 PRINT #2," 3 0 0 0"CHR$ (10)
170 PRINT #2," 4 0 0 0"CHR$ (10)
180 PRINT #2," 5 0 0 0"
190 LOCATE #3,3,2: PRINT #3, "ENvelope Tone" CHR$ (10)
200 PRINT #3,"   1 2 3"CHR$ (10)
210 PRINT #3," 1 0 0 0"CHR$ (10)
```

```
 220 PRINT #3," 2 0 0 0"CHR$ (10)
 230 PRINT #3," 3 0 0 0"CHR$ (10)
 240 PRINT #3," 4 0 0 0"CHR$ (10)
 250 PRINT #3," 5 0 0 0"
 260 DATA "q", "2", "w", "3", "e", "r", "5", "t", "6", "y", "7", "u",
     "i"
 270 DIM T$ (12): FOR i = 0 TO 12: READ t$ (i): NEXT
 280 DIM V (5,3), T(5,3)
 290 SPEED KEY 255,10: ENV 1: ENT 1
 300 LOCATE 1,1: C=1
1000 CLS #1: PRINT #1, " Hovedmenu "CHR$ (10)
1010 PRINT #1, "Mellemrumstaste = Spil Melodi"
1020 PRINT #1, "XV                = ENV ændre"
1030 PRINT #1, "T                 = ENT ændre"
1040 a$="" :WHILE a$="": a$=INKEY$:WEND
1050 IF a$=" " THEN 1100
1060 IF a$="v" THEN 1200
1070 IF a$="t" THEN 1300
1080 GOTO 1040
1100 CLS #1: PRINT #1, "Klaviatur" CHR$ (10)
1110 PRINT#1,"2 3 4 5 6 7"
1120 PRINT#1,"q w e r t y u i"CHR$(10)
1125 PRINT#1, "Mellemrumstast = Menu"
1130 a$="":WHILE a$="":a$=INKEY$:WEND
1140 IF a$= " " THEN 1000
1150 FOR i = 0 TO12 : IF a$<>t$(i) THEN NEXT i :
     GOTO 1130
1160 per = ROUND (125000/440/2↑(I/12))
1165 c=c*2: IF c=8 THEN c=1
1170 SOUND c,per,0,0,1,1
1180 GOTO 1130
1200 CLS#1: PRINT#1, "ENV ændre" CHR$((10)
1210 PRINT#1, " Slut=0"CHR$(10)
1220 INPUT #1 ,"Hvilket element (Side,Spalte)";z,s
1230 IF z*s = 0 THEN 1000
1240 INPUT #1,"Værdi";V(z,s)
1250 LOCATE #2,s*10,4+2*z: PRINT #2,V(z,s);"    "
1260 ENV 1,v(1,1),v(1,2),v(1,3),v(2,1),v(2,2),v(2,3),v(3,1),v(3,2),
     v(3,3),v(4,1),v(4,2),v(4,3),v(5,1),v(5,2),v(5,3)
1270 GOTO 1200
```

```
1300 CLS#1:PRINT#1,"ENT ændre" CHR$(10)
1310 PRINT#1,"Slut=0"CHR$(10)
1320 INPUT #1, "HVilket element (side,spalte)";z,s
1330 IF z*s = 0 THEN 1000
1340 INPUT #1, "Værdi"; t(z,s)
1350 LOCATE #3,s*10,4+2*z: PRINT #3,t(z,s);"  "
1360 ENT -1,t(1,1),t(1,2),t(1,3),t(2,1),t(2,2),t(2,3),t(3,1),t(3,2),
     t(3,3),t(4,1),t(4,2),t(4,3),t(5,1),t(5,2),t(5,3)
1370 GOTO 1300
```

Efter programmets opstart fremkommer en skærmmaske med en hovedmenu (øverst til venstre). Den ønskede programdel kan vælges ved tryk på en tast. Så længe alle parametre er 0, giver en spillende melodi ikke meget lyd fra sig. Der kræves mindst en angivelse af volumen. Til det skal der først angives, hvilket element og den nye værdi. Øjeblikkelig ændres i den nedenfor stående matrix. Alle parametre overgives til indhylningskommandoerne. De er iøvrigt nemme at flytte til egne programmer.

Der skal også knyttes et par bemærkninger til dette ret omfattende program. Linie 10 til 270 indretter en skærmmaske og de 3 vinduer. To af vinduerne bruges til udlæsning af ENV- og ENT-parameter. I det faste vindue afvikles alle indtastninger og ledetekster. I linie 260 indlæses tastaturbelægningen for de enkelte toner i strengen T$. Ved sammenligning, kan en enhver tast i feltindexet anvendes som "node-nummer". Nummeret bruges som basis ved beregning af nodeperioden.

I linie 280 dimensioneres de to parameter-arrays for ENV og ENT. Linie 290 kobler repetitionsfunktionen fra og sletter tidligere indhylningskurver (envelopes).

I linie 1000 starter det egentlige program. Først udskrives menuen. I den efterfølgende linie aflæses et tegn fra tastaturet (A$), hvorfra en relevant underrutine vælges og kaldes.

Linierne 1100 til 1180 optages af delprogrammet: "SPIL MELODIEN". Det er først og fremmest FOR-NEXT-løkken i linie 1150, der er interessant. Den fortsættes kun, hvis der ikke er overensstemmelse mellem A$ og T$(I). Ellers beregnes nodeperiode fra I (tast og node-tal) til SOUND-kommandoen.

IK linie 1200 starter ændringsdelen for ENVelope-volume. Heri er der egentlig ikke noget usædvanligt. Takket være WINDOW-teknikken, kan man indlæse de nødvendige data via INPUT. Den sidste del (fra 1300) tjener til ændring af ENT-parameter og er næsten identisk med ENV-delen.

8.2. HVORDAN PLANLÆGGES EN LYD?

Som det kendes fra manualen, kan man ved hjælp af en indhylningskurve "efterabe" forskellige musikinstrumenter. For at kunne gøre dette, må man bortset fra at være i besiddelse af en vidtrækkende tålmodighed, også kende lidt til de "ægte" lyde. Man kan således tænke sig til, at lyden af en trompet klinger af lige så hurtigt, som den starter, imens en klokke har et hurtigt anslag og en lang udklingningstid. Et klaver har et meget abrupt anslag og efterklang. I det følgende vil vi prøve at simulere forskellige musikinstrumenters lyde ved at ændre på indhylningskurvens parameter.

En glasklokke har en meget ren, dvs. regelemæssig frekvens. Derfor ændres tonehøjden ikke med ENT-parameteren. Volumen skal derimod, så hurtigt som muligt, sættes til højeste værdi, hvorefter den uhyre langsomt skal dale til 0. De to parametermatrix kommer til at se således ud:

1	15	1	1	0	1
1	0	1	1	0	1
1	0	1	1	0	1
12	-1	8	1	0	1
2	-1	20	1	0	1

eller i ENV-kommando således:

```
ENV 1, 1, 15, 1, 1, 0, 1, 1, 0, 1, 12, -1, 8, 2, -1, 20
```

En metalklokke klinger lidt mere skurrende. Det kan vi også få den til på CPC ved, at vi lader frekvensen køre lidt ujævnt (afvige opefter og nedefter). Kommandoen ser således ud:

```
ENT -1, 1, 1, 3, 1, -1, 3, 1, 0, 1, 1, 1, 3, 1, -1, 3
```

Desværre er det ikke muligt at ændre på klangfarven. I modsætning til andre computere, kan CPC ikke lave "lukkede" blæselyde, men kun høje lyse klange. Der er derfor ingen mulighed for at efterligne træblæseinstrumenter. Den typiske metalliske klang fra en trompet kan heller ikke fralokkes CPC (i hvert fald ikke fra BASIC). Alligevel vil vi beskrive nogle indhylningskurver, hvis ligheder med de ægte instrumenter anes.

En harmonika har alt efter bygning og spillemetode en relativ langsom opbygning af volumen og en tilsvarende langsom udklingning. Tonen er desuden ikke særlig ren:

ENV 1,7,2,1,1,1,1,1,0,1,1,0,1,15,-1,1

ENT -1,1,0,3,1,-1,1,1,0,2,1,0,1,1,1,1

Alle instrumenter, hvis toner stammer fra en streng besidder en karakteristisk indhylningskurve. Efter det meget hurtige anslag klinger tonen lidt af, hvorefter den kun langsomt bringes til ophør. Kommandoen til denne lyd er:

ENV 1,1,15,1,1,-3,2,1,0,1,1,0,1,12,-1,4

Med forskellige kurver kan man opnå "strenge" effekter. Et klavers klang fås ved:

ENT -1,1,1,3,1,-1,3,1,0,3,1,1,3,1,1,3,1,-1,3

Den lidt anstrengte klang hos en banjo fås med:

ENT -1,1,2,1,1,0,2,1,-2,1,1,0,4

Det var kun et lille udsnit af de muligheder, der findes. Ja faktisk, er spekteret uudtømmeligt. Hvordan med et stykke slagtøj eller fantasiklange?

INDHYLNINGSKURVER	
KLOKKE:	ENV 1,1,15,1,1,0,1,1,0,1,12,-1,8,2,-1,20
METALKLOKKE:	ENT -1,1,1,3,1,-1,3,1,0,1,1,1,3,1,-1,3
HARMONIKA:	ENV 1,7,2,1,1,1,1,1,0,1,1,0,1,15,-1,1 ENT -1,1,0,3,1,-1,1,1,0,2,1,0,1,1,1,1
STRENGINSTRUMENT:	ENV 1,1,15,1,1,-3,2,1,0,1,1,0,1,12,-1,4
KLAVER:	ENT -1,1,1,3,1,-1,3,1,0,3,1,1,3,1,1,3,1,-1,3
BANJO:	ENT -1,1,2,1,1,0,2,1,-2,1,1,0,4

9. BASIC OG OPERATIVSYSTEMET

Fortolkerens og operativsystemets rutiner kan være af stor nytte.Det har vi allerede fået bevis for.

9.1. HVORDAN LAGRES BASIC-LINIER?

Hvordan gemmes en BASIC-linie egentlig internt? Som det fremgik af kapitel 2, så tildeles alle kommandoer, så som PRINT, SIN, SAVE o.s.v. en speciel kode. Denne kode kaldes også for TOKEN. Metoden sparer enorm megen hukommelse (i stedet for PRINT, der fylder 5 bytes, kun 1 byte til TOKEN). Desuden skal fortolkeren ikke spilde tid med at dekode bogstaver under programkørsel.

Variabelnavne og tekster gemmes i form af ASCII-koder. Operatorer som *, / og AND o.s.v. har ligeledes en TOKEN-kode. Tal lagres i en noget mere kompliceret form, alt efter type (INTEGER, REAL) og størrelse.

I hukommelsen kan en TOKEN kendes på, at den består af en byte med værdi større end 127. Variabelnavne og strenge defineres alle med tegn, der ligger under værdien 128. På den måde kan fortolkeren nøjes med en kodetabel.

Lad os lave en lille test. Slet et evt. program i hukommelsen med NEW og indtast denne linie:

```
100 PRINT "test"
```

Dette "program" vil vi finde i hukommelsen. I direkte mode indtastes:

```
FOR I=368 TO 383:PRINT PEEK(I),:NEXT
```

Ved byte 368 starter vores BASIC-hukommelse, der indeholder den foroven viste tegnfølge og fylder de første 16 bytes. På skærmen finder vi følgende værdier:

```
13  0  100  0  191  32  34  116  101  115  116  34  0  0  0  0
```

De første 4 bytes angiver to 16 bits tal i pointerformat. De to første angiver liniens længde (13 bytes). Hvis fortolkeren leder efter en bestemt linie f.eks. ved GOTO, så tester den først om det første linienummer er det rigtige. Er

målet ikke fundet, så adderes linielængden til adressen; på den måde finder fortolkeren den næste linies start, og kan teste forfra.

I de næste to bytes står linienummeret. Så følger den første TOKEN; 191 betyder PRINT. 32 og 34 er ASCII-koderne for mellemrum og anførselstegn. Så er det ikke længere vanskeligt at forestille sig, at de næste fem koder danner ordet TEST. Med 34 for sidste anførselstegn har vi afsluttet linien. Da der ikke er lagret flere linier, vil de sidste bytes være på 0.

Denne struktur kan vi manipulere lidt med. Hvis fortolkeren finder en linie med 0 i starten af programhukommelsen, så bliver denne ikke LISTet, selvom den udføres på normal vis. Der kan heller ikke hoppes til linien, og den kan ikke slettes, idet en linie med nummeret 0 ikke kan eksistere. Ønsker man altså at beskytte den første linie mod LIST, så skal bytene 370 og 371 sættes til 0 via POKE-kommandoen. Prøv selv.

Det er ikke muligt at opnå samme effekt med linier, der ikke ligger i starten af hukommelsen. Linienummeret kan, selvom det sættes til 0, listes på normal vis.

Måske kender man kommandoen OLD eller RENEW fra andre computere. Den tjener til at genetablere et program, der ellers er blevet slettet med NEW-kommandoen. Det fungerer, fordi nogle computere ved NEW ikke fylder hukommelsen op med nuller, men derimod nøjes med at reset'e pointere, så fortolkeren tror, at hukommelsen er tom. Desværre hører CPC ikke til i denne kategori af computere. Der findes ikke nogen GØR-DET-SELV opskrift til denne ekstrakommando.

9.2. GARBAGE COLLECTION

Har læseren nogensinde hørt om en affalds-indsamling? Det er i hvert fald den direkte oversættelse af garbage-collection. I computersammenhæng er det en snu indretning, som vi vil gå lidt nærmere ind på.

Når fortolkeren arbejder med strengvariabler, så bi-produceres der affald i store mængder! Hver gang en streng forandres et eller andet sted i hukommelsen (om det så kun drejer sig om et enkelt bogstav), så defineres og anlægges den fuldstændig forfra, uden at den gamle slettes eller overskrives (man kan næsten sige, at de ligger side om side). Der kan altså ligge mange af den slags variabel-dubletter rundt omkring i hukommelsen til ingen nytte. På et eller andet tidspunkt, er alle bytes i hukommelsen fyldt op, selv om der måske kun er gemt nogle få data. Resten er affald. Skal der nu anlægges en

helt ny variabel, skal der ryddes op inden. Denne proces kaldes for GARBAGE COLLECTION. Og da det er en rigtig tidsforsinker, er den meget berygtet blandt computerfolk. Her er et eksempel:

```
DIM A$(8000)
FOR I=0 TO 8000:A$(I)=CHR$(1):NEXT I
```

Disse få kommandoer, har nu fyldt hukommelsen godt op. Med PRINT FRE("") (endelig ikke PRINT FRE(0)!) kan vi nu udløse en garbage collection. Udførelsen af den tilsyneladende simple proces tager op til flere minutter. Fortolkeren skal nemlig slette 8000 ens strenge ud af et antal på ialt 8001.

For at forhindre at denne tidskrævende proces bliver udført fra tid til anden, er det en god ide, at strø omkring sig med PRINT FRE("") , så vil skraldindsamlingen foregå ofte, men ikke så omfattende.

9.3. ERROR! ERROR!

Efter devisen: intet program uden fejl, har der også indsneget sig en fejl i BASIC-ROM, der dog ikke bemærkes uden videre. Uheldigvis gemmer den sig i REM-linier, så man nemt overser den i et program.

Hvis der i en REM-linie er anført enten PIL MOD HØJRE (TAB-tast) eller LODRET STREG (SHIFT +alfakrølle), så kollapser fortolkeren og der opstår særdeles uberegnelige symptomer. Hvis man har heldet med sig, består forstyrrelsen i, at der hoppes til linie 32511. Eksisterer linien ikke, afbrydes programkørselen. Sletning af hele programblokke kan også forekomme.

Den uberegnelige virkning skyldes sikkert resten af programmets opbygning. Der kan endda ske skift mellem skærm-modes. Det kan være, at man ved fornyet efterforskning i sagen kan drage nytte af fænomenet, hvem ved?

9.4. UBEKENDTE SIDER

Her vil vi præsentere læseren for nogle ubekendte sider af BASIC og operativsystem. Det er egenskaber, der ikke er beskrevet i computerens manual.

Den første egenskab angår editoren, der f.eks. er ansvarlig for styringen af cursor,COPY-tasten og indlæsning af linier. Når man under arbejdet har indtastet nogle tegn i en linie, og vil korrigere en opstået fejl, så sker dette via

EDIT-kommandoen. Her skal cursoren først flyttes til den linie, der ønskes rettet i. Alle tegn der tastes, vil blive indføjet. Det kan være irriterende, hvis man ønsker at slette gamle tegn. INSERT-mode kan kobles fra ved samtidigt tryk på CTRL og TAB. Fornyet tryk kobler INSERT ind igen.

Den anden egenskab berører BASIC-fortolkeren. Hvis man kommunikerer med en array (f.eks. A(1) for første gang i et program uden at have dimensioneret den, så foretger fortolkeren en automatisk dimensionering med plads til 11 elementer. Det erstatter kommandoen DIM A(10).

Dertil skal siges, at det kun lønner sig at udelade DIM-kommandoen, hvis tabellen virkelig skal have 11 elementer. For hver dimensioneret variabel skal der bruges hukommelsesplads, der ikke kan udnyttes af data. Desuden gør det ikke et program nemmere at læse, at der pludselig dukker en ny tabel op, der ikke er blevet dimensioneret.

VI NARRER BASIC

&AE45 TJENER SOM NOTATBYTE VED PROGRAMBESKYTTELSE.

POKE &AC00,1 KOBLER KOMPRESSIONS-MODE IND, DVS. AT ALLE OVERFLØDIGE MELLEMRUM SLETTES. (SLÅS FRA MED POKE &AC00,0).

BYTENE &AE81 OG &AE82 PEGER PÅ BYTEN FØR PROGRAMSTART OG &AE83 OG &AE84 PÅ AFSLUTNINGEN.

9.5. ENDNU ET PAR TRICKS

Værdier som SPEED xxxx eller lignende glemmes hurtigt igen, når man eksperimenterer. Så er gode råd dyre. Ved SPEED INK-kommandoer er der en løsning på problemet. De to parametre for denne kommando gemmes af BASIC i adresserne &B1D7 og &B1D8, hvor de hentes af operativsystemet ved farveskift. Og det, operativsystemet kan, kan vi faktisk også.....

Har man gjort brug af selvdefinerede tegn, så vil man sikkert også have ærgret sig over, at de 8 bytes i en tegnmatrix hele tiden skal beregnes pany, også selvom ændringen kun drejer sig om et enkelt punkt. Her er der også hjælp at hente. De selvdefinerede tegn fra CHR$*240) til CHR$(255) er lagret i RAM i området &AB80 til &ABFF. Til hvert tegn er der reserveret 8 bytes, der nemt kan udlæses med PEEK. Her kan vi altså nøjes med at beregne de enkelte punkter.

Adressen for et tegn bestemmes med:

ADRESSE = 43904 + (X-240)*8

X er nummeret på det ønskede tegn i området mellem 240 og 255.

10. EKSTRAUDSTYR OG DERES FUNKTIONER

Alle computere har brug for ekstraudstyr, for at kunne arbejde fornuftigt. Ved nogle fabrikater må man, bogstavlig talt, opsøge dette udstyr.Heldigvis er det anderledes hos AMSTRAD.

10.1. DISKETTEDREVET

Tilhører læseren også den hektiske og utålmodige type? Tager det for lang tid at lagre og hente programmer med bånd? Så må man anskaffe sig et diskettedrev. For at loade 22 K bytes skal kassettebåndoptageren bruge 2 minutter og 27 sekunder med speed-load. Diskettedrevet bruger 9 sekunder til samme mængde.

Men det er ikke den eneste fordel ved et diskettedrev. Da de enkelte magnetspor (40 ialt) er ordnet som de enkelte numre på en lp-plade, så kan man som med en pick-up, gå direkte til det ønskede sted på "båndet" læs: disketten. Det kaldes direkte tilgang. Computeren skal ikke læse alle filerne igennem inden den når den rigtige, men kan gå direkte til den ønskede fil med det samme.

En diskettestation kan bruges næsten som en forstørret hukommelse.

Et diskettedrev har også andre fordele. F.eks. findes der på hver diskette en indholdsfortegnelse over de filer, der befinder sig på den. Een enkelt ulempe er der dog også; den er dyr. Den skal bruge et specielt styrekredsløb, der kaldes en CONTROLLER. En sådan følger med det første drev, man anskaffer sig. Kapaciteten på en controller rækker til 2 diskettedrev. Uden controller går det altså ikke.

Yderligere ligger der en ROM i drevet, der indeholder en udvidelse til CPC's egen BASIC. Det er de, til betjening af drevet, nødvendige kommandoer.

10.2. PRINTEREN

En af CPC'erens mange fordele, er det indbyggede interface til en printer. I modsætning til et diskettedrev, skal en printer ikke have særligt udvidelsessoftware for at kunne fungere. Alt er indbygget.

Det drejer sig om et såkaldt centronics interface, hvilket betyder at interfacet er tilpasset printerfirmaet Centronics. Det er blevet en standard for mange fabrikanter af printere og bevirker således, at man kan koble næsten enhver printer, på markedet, til sin computer. Lidt malurt er der dog alligevel, idet CPC kun overfører de nederste 7 bits af 8 bits ASCII-koden og således "stjæler" halvdelen af de mulige tegn. Ulempen er dog ikke så stor. Koderne fra 0 til 127 indeholder de vigtige tegn. Resten er reserveret til grafik o.l.

Det kan forekomme, at printerens ASCII-koder ikke svarer helt til computerens. Hvis det er tilfældet, må man lave en konverterings/tilpasnings-rutine. Det lyder vanskeligere end det er. Ved hjælp af en ASCII-tabel, er det ikke svært at finde de ønskede tegn og derefter overføre dem med PRINT#8,CHR$(X). X er udtryk for ASCII-værdien.

10.3. JOYSTICK ELLER STYREPIND

På et eller andet tidspunkt, vil man få brug for at aflæse et joystick's bevægelser i et program. Har man allerede prøvet det, er det næsten sikkert, at man har anvendt IF-THEN versionen:

```
IF JOY(0)=1 THEN 100
IF JOY(0)=2 THEN 200
IF JOY(0)=3 THEN 300
...
...
```

Metoden er særdeles langsom og sammenligningsformen er omstændelig. Det går langt hurtigere med:

```
10 A=LOG (JOY(0))/LOG(2)
20 ON A GOTO 100,200,300....
```

Linie 20 forgrener sig, alt efter indholdet i variablen A, til forskellige programafsnit. Hvis ON-kommandoen blev koblet direkte på JOY-værdien ville vi ikke opnå det ønskede resultat, da JOY antager værdier af 2'er potenser (1,2,4,8,16,32). Derfor må vi beregne logaritmen til basis 2.

Et joystick er egentlig kun et antal (5-6) kontakter (taster), hvor f.eks. affyringstasten har værdien 2. Resten sidder i en rundkreds under styrestangen i hvert sit verdenshjørne. Kontakterne aflæses på samme måde som de øvrige taster.

Der er naturligvis mange forskellige slags joysticks på markedet. De billigste arbejder med foliekontakter (ZX81 typen). De mere håndfaste benytter rigtige microswitches. Der laves nu næsten udelukkende standardstik til joysticks.

11. LIDT OM INTERFACING

På bagsiden af Amstraden findes diverse bøsninger og stikforbindelser. Dette kapitel vil oplyse om deres funktioner.

11.1. LILLE INTERFACE-KURSUS

Inden vi kaster os over diverse udredninger, så lad os først slå fast, hvad et interface egentlig er. Når man "snakker" interface, drejer det sig nemlig ikke altid om en stikforbindelse til et eller andet ekstraudstyr. I de fleste tilfælde, er det en direkte forbindelse til det inderste i en computer. Et godt eksempel herpå er den såkaldte EKSPANSION CONNECTOR (udvidelses-forbindelse), der også er tilslutningssted for diskettestationen. Her kan man også koble ekstra ROM-kredse til. En ROM-kreds er et hukommelseselement, der kommunikeres direkte af processoren. Dette udvidelsesstik består af adresse-, data- og styrebus + nogle ekstra styreimpulser.

Ved siden af findes printertilslutningen, der følger centronicsstandarden. Denne PORT er også forbundet direkte med processoren. Yderligere, er der forbindelse til 8255 kredsen.

Joystick-stikket er en direkte udvidelse af tastaturet. Det er forbundet med forskellige kredse i computeren.

Tilslutningerne for monitor og stereoforstærker er også interfaces, men de arbejder ikke direkte sammen med processoren.

11.2. HVORDAN FUNGERER ET INTERFACE?

Et interface fungerer, stort set, på samme måde på alle computere. De data, der skal kommunikeres afleveres i interface-kredsen af processoren, der kommunikerer med det perifere udstyr via samme interface og sørger for overførselen af data. Hertil benyttes særlige synkroniseringsimpulser, der tjener til at forhindre de kommunikerende parter i at "tale i munden" på hinanden.

Inden overførselen, skal processoren have at vide, om data skal sendes eller modtages. Alt efter valg, sættes porten til output eller input.

I nogle tilfælde indbygges der ikke en særlig interface-chip (som ved CPC). Her overtager processoren alle disse opgaver, hvilket betyder at hastigheden formindskes.

Programmerne, der skal styre et interface, ligger lagret i ROM, og kan anvendes af BASIC-kommandoer. En effektiv programmering af interfaces kan dog kun foregå i maskinsprog (med enkelte undtagelser).

11.3. DET PERSONLIGE INTERFACE

Joystick-porten kan bruges til mere seriøse ting end lige netop spilleprogrammer. Den såkaldte USER-PORT kan bruges i BASIC med kommandoerne JOY og INKEY. Således er alle forudsætninger for kontakt med omverdenen til stede.

Pin 1 til 7 (se manualens tillæg 5) kan kobles til 2 kontakter. Den ene kontakt kobles til COMMON (pin 8), den anden til COM 2 (pin 9). Med de to sidstnævnte ben skelner computeren mellem joystick 0 og 1. Ved aktivering af en kontakt lægges en indgang på (1-7) på et commonsignal. For CPC er det det samme som en tast- eller joystickbevægelse. Joystick-portene kan forsynes med både relæer og transistorer. Hvordan de bruges, vil vi overlade til læserens egen fantasi.

11.4. TASTATURAFLÆSNING

Ved en hurtig optælling kan man fastslå at CPC har 73 taster (SHIFT er kun talt med 1 gang). Alle disse taster aflæses regelmæssigt i intervaller af 1/50 sekund. Til det formål er tastaturet delt op elektronisk i 10 koloner, som Z80 kan koble ind enkeltvis. Dette gøres ved at overføre det ønskede kolonnenummer til 8255-kredsen.

Er en tast i den valgte kolonne nede, så sættes en bit til 0, ellers 1. Hver spalte råder over 8 bits, der således tillader sammensætning til en hel byte. Processoren kan således ved at aflæse de nulstillede bits, finde ud af hvilken tast, der er blevet aktiveret.

12. KASSETTEBÅNDOPTAGER OG TASTATUR

Betjeningen af kassettebåndoptageren har lidt en krank skæbne i computerens manual. Det vil vi afhjælpe i dette afsnit. Også tastaturaflæsningen gemmer uventede muligheder.

12.1. HVORDAN OPBYGGES EN FIL?

I CPC-manualen er man temmelig sikkert stødt på betegnelsen "ASCII-FIL". Desværre, har man glemt, at gøre opmærksom på, at også aritmetik- og strengvariabler kan gemmes på bånd. Vi vil se på, hvordan man kan indlæse ASCII-listninger i BASIC-programmer.

Navnet ASCII-FIL kommer af, at alle data lagres som simple kæder af ASCII-koder. Den eneste forskel mellem en ASCII- og en program-fil består i, at en ASCII-fil ikke er en kopi af et adresseområde, men tegnfølger, der afsluttes med CHR$(13). Der skelnes ikke mellem data, variabler o.s.v.

Skrivningen og aflæsningen af den slags filer minder meget om output på skærmen; PRINT#9 og INPUT#9. Også ved udskrift på skærmen bruges CHR$(13) som afslutning på de enkelte informationer. Dog har tegnet yderligere den funktion, at cursoren flyttes ned i den efterfølgende linies første position.

Lad os nu forestille os, at vi, grebet af ægte computer-iver, har fået samlet en hel del data sammen i nogle variabler, og ønsker at gemme dataene til imorgen. Hvad så? Lad os se på et programeksempel:

```
10 REM SKRIVNING AF DATA
20 OPENOUT "TEST"
30 PRINT#9,A$
40 PRINT#9,B
50 CLOSEOUT
```

```
10 REM INDLÆSNING AF DATA
20 OPENIN "TEST"
30 INPUT#9,A$
40 INPUT#9,B
```

```
50 PRINT A$,B
60 CLOSEIN
```

Det første program skriver indholdet af variablerne A$ og B$ på bånd. Det andet program henter dataene tilbage til computerens hukommelse igen. Variabelnavnene er ikke vigtige i denne sammenhæng. Der kunne bruges Y$ og H$ med samme resultat. Men derimod skal variabeltypen stemme, idet man ellers vil få udskrevet fejlen TYPE-MISMATCH.

Ved lagring af ASCII-listninger, skabes der på lignende vis rækker af tegn, der kan læses som strenge i BASIC. Her et program som eksempel:

```
10 INPUT "HVOR MANGE LINIER ØNSKES ";A:A=A-1
20 DIM A$(A)
30 OPENIN "FILNAVN"
40 FOR I= 0 TO A
50 INPUT #9,A$(I)
60 NEXT
70 CLOSEIN
```

Efter gennemkørsel af dette program, vil en ASCII-listning være blevet overført til en streng-tabel (array), hvor man kan manipulere sine data. Desværre, er der et lille men. BASIC anvender også komma til adskillelse af strenge. Resultatet bliver at alle linier (der er som regel mange), der indeholder et komma, vil blive skilt på dette sted og fylde to linier (strenge). Det kan undgåes på en smart måde. Kommandoen INPUT skal ganske enkelt skiftes ud med kommandoen LINE INPUT. Denne kommando godtager kun et CHR$(13) som skilletegn.

I det ovenstående programeksempel skal man på forhånd angive antallet af linier i tabellen (antallet af tekstlinier). Har man taget fejl af dette antal, vil beskeden EOF MET blive udskrevet, når programmet gør forsøg på at indlæse flere data end der er i listningen (filen). Der findes en tilstand i BASIC, hvorpå man kan teste for EOF (End Of File). PRINT EOF giver resultatet 0, hvis der endnu er data, der skal læses. Ved slutningen af en fil er resultatet -1. Det giver følgende ændring:

10, 20, 30 og 70 skal ikke ændres.

```
40 WHILE NOT(EOF)
50 LINE INPUT#9,A$(I):I=I+1
60 WEND
```

WHILE-WEND rutinen aflæser kun fra båndet, så længe der er data i filen. Dog kan man møde fejlen SUBSCRIPT OUT OF RANGE, hvis listningen omfatter flere linier, end der er blevet dimensioneret.

Ved opbygning af en datafil, kan man gøre sit til at sikre overførselen, ved at lagre antallet af linier i en variabel i starten af hver fil. D.v.s. at man skrive denne variabel på båndet umiddelbart inden selve data. Når man så, på et senere tidspunkt, skal indlæses sine data igen, vil det første der læses være antallet af linier i filen (dataantallet). Dette tal kan dimensionere tabellen til den rigtige kapacitet.

En særlig lækkerbidsken kan man finde i de 32 adresser fra &B807 TIL &B816 OG &B84C TIL &B85B. Her lagres navnet på INPUT-fil (&B087) og OUTPUT-fil(&B84C) af operativsystemet som ASCII-strenge. Ved hjælp af den følgende linie, kan man udlæse det sidste output-filnavn:

```
FOR I=&B84C TO &B85B:PRINT CHR$(PEEK(I));:NEXT
```

Hvad med en rutine, der kan implementeres i et program til sikring af, at et filnavn ikke kan ændres. Det er kun i starten af et program, at filnavnet skal aflæses. Er navnet forkert, kan man lade hele programmet forsvinde med NEW (måske ledsaget af et par kommentarer!).

Og hvis man ønsker at erfare, hvilken skrivehastighed der er aktiv, så hedder det:

PRINT PEEK(&B8D1). Er resultatet 6, så lagres der ved lav hastighed. Ved 12 er det SPEED-WRITE 1.

12.2. INKEY I ET ANDET LYS

Senest under opbygning af det første BASIC-spilleprogram får man brug for at kunne aflæse flere taster på een gang. Heldigvis er der en BASIC-kommando til dette job:

INKEY(X) (Bemærk at dollar-tegnet mangler!!).

Denne kommando oplyser om, at tasten X netop aktiveres. Da denne funktion arbejder uafhængigt af tastaturbufferen, vil man ikke få aflæst næste tegn i køen, men kun den aktive elektroniske registrering. Her er et eksempel på anvendelsen af kommandoen:

```
10 CLS
20 IF INKEY(69) = 0 THEN LOCATE 1,5:PRINT"TAST A
   TRYKKET"
30 IF INKEY(36) = 0 THEN LOCATE 1,10:PRINT"TAST L
   TRYKKET"
40 LOCATE 1,5:PRINT SPACE$(20):REM SLET LINIE 5
50 LOCATE 1,10:PRINT SPACE$(20):REM SLET LINIE 10
60 GOTO 20
```

Start programmet op og tryk A og L samtidigt. Begge meddelelser udskrives på skærmen. Det kunne ikke lade sig gøre med INKEY$, idet denne kun kan aflæse eet tegn af gangen.

I mange programmer er der indbygget en funktion, der afventer et vilkårlig tryk på en tast. En sådan løkke ser som regel sådan ud:

```
WHILE INKEY$="":WEND
```

Den samme funktion kan udføres af en rutine i ROM, der kaldes med:

```
CALL &BB18
```

13. INDFØRING I MASKINSPROG

Der er mange publikationer med programmer lige til at taste af. Tit er programmerne skrevet i maskinsprog, et sprog der for begyndere tilsyneladende er forbudt område. Det skal indrømmes, at maskinsprog ikke er så nemt at lære som BASIC, men på den anden side er det væsentlig hurtigere, og byder på mange flere muligheder, for en kreativ og fantasifuld programmør, end BASIC gør.

I dette kapitel vil vi vise grundlæggende skridt til programmering i maskinsprog på processoren Z80.

13.1. HVAD ER MASKINSPROG I DET HELE TAGET FOR NOGET?

Som man sikkert ved, så er maskinsprog den eneste måde at kommunikere direkte med processoren på uden at skulle benytte hverken fortolker eller compiler. Dette gør, at man kan opnå utrolige hastigheder for udførelse af et program.

Maskinsproget omfatter forskellige kommandoer, hvoraf alle de komplexe operationer i BASIC eller et andet programmeringssprog er sammensat. Groft kan maskinsprogskommandoerne inddeles i 3 grupper. For BASIC-programmører er de nemmeste at forstå sikkert HOP-kommandoerne, hvormed programmet kan springe rundt i hukommelsen på samme måde som med GOTO og GOSUB. Andre kommandoer tjener til data-manipulationer, f.eks. addition, boolsk algebra etc. Den sidste gruppe omfatter operationer, der flytter informationer fra et sted i hukommelsen til et andet.

Der findes ingen variabler for mikroprocessorer. Den kender kun de normale hukommelsesceller og de interne registre. Programmøren må selv sørge for at skelne mellem data og programbytes. I almindelighed kan data-manipulationer kun foregå i de interne registre.

En maskinkodekommando består altid af en såkaldt operationskode (OP-CODE), der så at sige angiver kommandoens nummer. Denne OPCODE kan for Z80 processorens vedkommende bestå af op til 3 bytes. Yderligere kan kommandoen følges af indtil 2 bytes bestående af data. I teorien har Z80-kommandoer dermed en længde på indtil 5 bytes. I praksis er det imidlertid kun 4, da 3-byte-OPCODES kun optræder i følge med 1 databyte.

13.2. CLOCK-FREKVENSEN

Alle chips i en computer arbejder efter et lille uanseeligt stykke quarts, der bestemmer en takt (frekvens) på 4, 2 eller 1 Megahertz (millioner svingninger i sekundet). Det er nødvendigt fordi de forskellige IC (Intergrated Circuits) skal synkroniseres. Manglede denne synkronisering kunne det f.eks. ske, at en chip sendte data til processoren, selvom den ikke var klar til at modtage. Selv en nok så hurtig processor skal bruge tid til at forarbejde sine data.

13.3. EN MICROPROCESSORS OPBYGNING

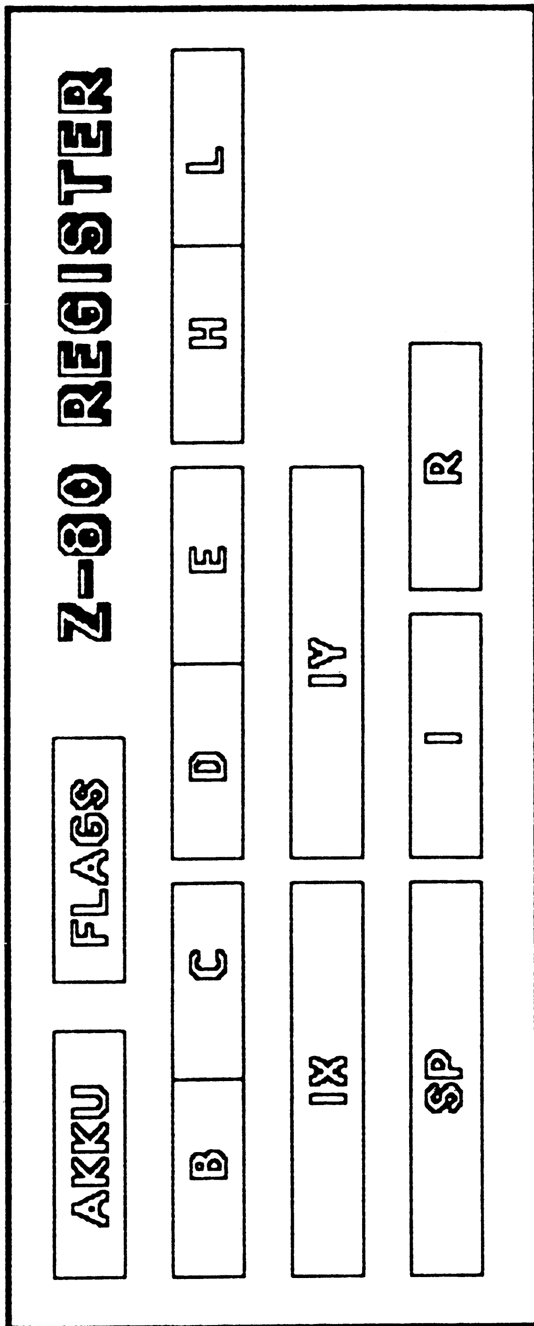
Enhver mikroprocessor har interne registre, hvori dens operationer udføres. Det vigtigste register er den såkaldte akkumulator. I denne sker de fleste aritmetiske og logiske beregninger. Akkumulatoren (AKK. eller A) er et 8 bits register, hvilket vil sige, at den kan optage 1 byte til bearbejdning (egentlig laver akkumulatoren ikke selv nogen bearbejdninger, her lægges kun resultater). De fleste aritmetik-kommandoer behøver 2 operander (f.eks. addition af 2 tal). Den første operand befinder sig allerede i akkumulatoren inden beregningen. Den anden stammer fra et andet register i processoren eller fra hukommelsen. Efter additionen gemmes resultatet igen i akkumulatoren. Sådan foregår det i alle processorer.

Et andet register kaldet F gemmer forskellige flag, der viser forskellige tilstande i processoren. På baggrund af et sådant flag kan man f.eks. bestemme om indholdet af akkumulatoren er 0.

Hos Z80 er der yderligere 6 registre med specielle egenskaber. Disse hukommelsesceller danner parvis et 16-bits-register. Parret HL er dermed næsten en 16-bits akkumulator, dvs. at den overtager de samme opgaver, som den rigtige akkumulator, men med 16 bits i stedet for 8. Derved gøres det lettere at bearbejde større tal.

Yderligere register-par er BC og DE. Men ikke nok med det. IX og IY er 2 indexregistre (med hver 16 bits), der virker som pointere for bestemte hukommelsesceller. V.h.a. denne pointer kan man uden videre bearbejde hele grupper af data. Hvordan det foregår, følger senere.

16 bits registeret SP har en specialopgave. Den peger altid på det øverste element i STAKken (SP betyder også STACK-Pointer). Hvergang der lægges noget i STAKken eller der tages noget derfra, aktualiseres Z80 pointeren, så den peger på den nye position.



Figur 9.

Til slut er der registrene I og R, der er forbeholdt specielle opgaver (hardware-styring).

Som om det ikke kunne være nok, så har hvert register A til L et ekstraregister, der kan byttes ud med originalregisteret. Man kan dog kun arbejde med eet register af gangen. Derfor tjener ekstraregistrene mest som mellemlagre. I kommandoerne angives reserveregistrene med en efterstillet apostrof (i denne bog af tekniske grunde med anførselstegn).

Hermed er de til maskinsprog beregnede registre i Z80 blevet præsenteret. (Se figur 1). I det næste afsnit beskrives en enkelt kommandos gang i processoren.

13.4. EN MICROPROCESSORS ARBEJDSGANG

Lad os antage, at der ligger et maskinkodeprogram i computerens hukommelse. Det venter kun på at blive udført. Et eller andet sted må processoren vide, hvor programmet befinder sig. Dertil findes der et særligt 16-bits-register, kaldet PC eller Program Counter (program-tæller). I den er adressen på den kommando, der skal udføres næste gang lagret. Skal kommandoen udføres nu, henter processoren byten fra den angivne adresse. Byten fastholdes i processoren og PC (program-tælleren) forhøjes med 1, for at vi kan finde den næste byte. Samtidig bliver OPCODEn (det er den byten omhandler) dekodet, dvs. at microprocessoren fastslår, hvilken af de mange kommandoer, der egentlig står i hukommelsen. Nogle af Z80's kommandoer har en OPCODE, der er flere bytes lang (ellers kunne der jo kun genkendes 256 kommandoer). I dette tilfælde hentes de nødvendige bytes efter hinanden, på nøjagtig samme måde, som den første blev det. (PC'en peger hele tiden på den aktuelle adresse, fordi den forhøjes efter hver byte. 1 og 2 bytes data kan også forekomme; også disse indlæses, men skal ikke dekodes. De hører hjemme i bestemte registre, og holdes parat til forarbejdning et eller andet sted i processoren.

Skal dataene ændres (f.eks. ved addition e.l.), så foregår denne operation nu og resultatet gemmes igen i (f.eks. akkumulatoren). Dermed er en kommando afsluttet, og den næste kan påbegyndes.

Disse ting foregår selvfølgelig ikke i løbet af nulkommafem, selv strøm behøver en vis tid til at løbe rundt i. Under normale forhold varer hver af Z80's arbejdstrin såsom "hentning af opcode", "dekodning af opcode", "udførelse af kommando" og "lagring af resultat" en clock-cyklus. Mere komplekse kommandoer kan strække sig over flere cykler.

Samler vi trådende, kan man sige at PC'en hele tiden peger på den adresse, hvori den byte, der skal bearbejdes næste gang befinder sig. OPCODES og data indlæses een efter een. OPCODEn bliver dekodet og kommandoen udføres.

13.5. HEX-TAL ELLER SEKSTENTALS-SYSTEMET

Når man arbejder med maskinsprog, kan man ikke undgå at stifte bekendtskab med det hexadecimale talsystem. I modsætning til det normale talsystem, består dette af 16 cifre (0 til 9 og bogstaverne A til F for værdierne 10 til 15). Systemet benyttes fordi omregningen fra binær til hexadecimale tal er nem. Desuden repræsenterer et HEX-CIFFER netop en halv byte. Det største to-positioners hextal FF svarer til det binære tal 1111 1111, hvilket er det størst mulige indhold i en byte. Derfor tager man altid en halv byte og omregner den til et hextal. Tabellen viser de tilsvarende decimale- og binære tal:

DECIMAL	HEX-TAL	BINÆR
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Byten 1010 1011_2 svarer således til det hexadecimale tal AB_{16} idet 1010_2 er A_{16} og 1011_2 er B_{16}. Det modsatte er selvfølgelig også gældende.

For omregning fra hextal til decimaltal, bliver nu hver enkelt ciffer omregnet til decimaltal. Tallene regnet fra positionen længst mod højre multipliceres med grundtallet efter følgende mønster:

$ABCD_{16}$ ===› 10 11 12 13

```
13 * 16↑0 = 13 * 1    =    13
12 * 16↑1 = 12 * 16   =   192
11 * 16↑2 = 11 * 256  =  2816
10 * 16↑3 = 10 * 4096 = 40960
                        -----
DECIMAL                 43981
                        -----
```

Man kan også gå den modsatte vej (fra decimal til hex) hertil skal divisionsmetoden bruges, hvor decimaltallet hele tiden deles med 16. Resttallet bruges som hextal. Man fortsætter divisionen indtil værdien 0 er nået. Her følger et eksempel:

```
53000 / 16 = 3312 REST 8  ==› 8
 3312 / 16 =  207 REST 0  ==› 0
  207 / 16 =   12 REST 15 ==› F
   12 / 16 =    0 REST 12 ==› C
```

$53000_{10} = CF08_{16}$

Der er dog kommet lommeregnere, der understøtter både omregning og almindelig talbehandling i decimal-, oktal-, hex- og binære talsystemer. Iøvrigt bør gode ASSEMBLERE også være udstyret med disse funktioner.

13.6. BINÆR-ARITMETIK

13.6.1. ADDITION

Det skal være sagt med det samme, at den eneste forskel mellem binær- og decimal addition er talsystemerne. Summen af to nuller eller summen af et nul og et ettal byder ikke nogen vanskeligheder. Men vil vi beregne 1 + 1 får vi problemer. Decimalt er resultatet = 2. Men dette tal findes ikke i det binære talsystem. Derfor må vi (ligesom ved overskridelse af 9 i decimaltalssystemet) indføre en mente (CARRY) til næste position:

```
  0     0     1     1
+ 0   + 1   + 0   + 1
---   ---   ---   ---
  0     1     1    10
---   ---   ---   ---
```

Hele bytes kan adderes på samme måde:

```
  01101101 =   109
+ 00001001 = +   9
------------------
      1  1 (menter)
------------------
  01110110 =   118
------------------
```

For at lette oversigten, er menterne (CARRY) anført for sig. Skulle det forekomme, at hele 3 ettaller skal adderes (1+1+1 = 3), så bliver resultatet 1 1 (selvfølgelig!).

Nu består vores resultat pludselig af 9 bits! Den niende bit kaldes CARRY- eller overflows-bit. Den angiver, at additionen af 2 8-bits-tal overskrider den højeste værdi for en byte (255), hvorved vi befinder os i 16-bits-additionerne. Der er ikke nogen computer, der kan klare sig med 8 bits til repræsentation af tal. Men det er stadigvæk en umiskendelig kendsgerning, at en 8-bits-processor kun kan bearbejde 8 bits på samme tid. Består f.eks. et tal af 2 bytes, må additionen foregå i 2 omgange. Et eksempel:

```
  00110101 10010011
+ 10011011 11011111
-------------------
   1111111    11111 (CARRY)
-------------------
  11010001 01110010
-------------------
```

Additionens højre del kender vi allerede.

13.6.2. SUBTRAKTION

Når en computer skal trække to tal fra hinanden, så finder den komplementet til dette tal (dvs. multiplicerer med -1) og adderer derefter tallene. Det

hænger sammen med at additioner og negationer kan sammensættes elektronisk via AND, OR, XOR, NOT, men ikke subtraktioner.

For at kunne frembringe negative tal, flyttes en bytes talområde fra 0 - 255 til -127 - +127. Den mestbetydende bit (bit 7) tjener som fortegn. Er den 1, så har vi med et negativt tal at gøre. Er den 0 er tallet positivt. Men det er således ikke nok blot, at gøre et tal negativt, ved at sætte bit 7 til 1. Et eksempel tydeliggør vanskelighederne:

```
  0000001
+ 1000001
---------
  1000010
---------
```

Omregnet til decimaltal, ville resultatet blive at 1 + (-1) skulle være = -2, hvilket jo ikke er tilfældet! Vi må altså gå en anden vej. En byte kan ved opbygningen af et såkaldt toer-komplement på en enkel måde multipliceres med -1. På den måde inverteres alle bits og sluttelig adderes med 1.

```
Eksempel:     01011011
Inverteret:   10100100
            +        1
            ----------
              10100101
            ----------
```

Hvis vi benytter denne metode til at beregne 1 - 1 i det binære talsystem, så får vi det rigtige resultat:

```
  00000001
+ 11111111
----------
  11111111   (CARRY)
----------
  10000000
----------
```

Som det ses, har vi fået et overflow. Men i dette tilfælde forholder subtraktionen sig anderledes. Vi kan ganske enkelt ignorere denne bit. Subtraheredes 16 bits, vil den overflødige CARRY sørge for at positionen i den anden byte, ville blive sat til 0. Dette er vigtigt, idet alle neagtive 16-bits-tal inverteres i alle positioner. Som 2-byte tal vil -1 se således ud:

11111111 11111111

Manglede vi den ekstra carry-bit, vil resultatet være:

11111111 00000000

Og det er forkert!

Til alt held, er programmering af subtraktioner ikke så kompliceret i virkeligheden. Subtraktionskommandoerne i begge microprocessorer har indbygget toerkomplement-funktioner.

13.6.3. MULTIPLIKATION

Selvom man ikke skulle tro det: Maskinsprog indeholder kun to regningsarter, og det er addition og subtraktion. Alle andre regnearter er opbygget af disse to grundkommandoer, som regel som underprogrammer.

Da vi ikke vil gennemgå alle enkelthederne i maskinsprog (dertil findes der både bedre og mere udførlig litteratur), så vises her kun den enkleste algoritme for multiplikation. Den bruges ikke så gerne af professionelle programmører, da den ikke er så effektiv. Men nu til sagen.

For at kunne beregne produktet X * N er det nok at addere X N-gange. Det fungerer naturligvis, kun ved hele tal. Decimaltal bearbejdes på en meget mere kompliceret måde. Til belysning af multiplikation følger et eksempel:

4 * 3 ===› 4 + 4 + 4 = 12

13.6.4. DIVISION

Også til division findes der en simpel fremgangsmåde. For at dele X med N, trækker man kontinuerligt værdien N fra X. Antallet af disse subtraktioner, indtil N bliver større end X, angiver resultatet af divisionen. Her følger et eksempel:

10 / 3 = ?

10 - 3 = 7 ===› TÆLLEREGISTER = 1

7 - 3 = 4 ===› TÆLLEREGISTER = 2

4 - 3 = 1 ===› TÆLLEREGISTER = 3 ===› 10 / 3 = 3 REST 1

Disse metoder er kun mulige, fordi maskinsprogskommandoer udføres med meget store hastigheder. Iøvrigt er det samme princip, en lommeregner anvender. For hver gang en regnetast trykkes udføres en maskinsprogsrutine (naturligvis med de tilsvarende algoritmer).

De 4 grundliggende regnearter kan kombineret skabe højere funktioner (f.eks. potenser, sinus, kvadratrødder o.a.). Enhver matematisk operation kan udtrykkes via AND-, OR-, XOR- og NOT-operationer.

13.7. HVORDAN FUNGERER SAMMENLIGNINGER (RELATIONER)

I BASIC er sammenligninger ikke noget usædvanligt. Men hvordan klares de i maskinsprog? Lad os se på et eksempel:

A = B <===> A - B = 0

Som det ses, er en sammenligning mellem to tal ret så simpel at omdanne. For computeren har denne form den fordel, at der på den højre side af lighedstegnet står et nul. 0 (nullet) er den eneste måde, hvorpå microprocessoren kan finde ud af, om der er et tal i akkumulatoren eller ej. Dertil kombineres alle bits med hinanden ved hjælp af OR-relationen. Således :

BIT 7 OR BIT 6 OR BIT 5 OR BIT 4 OR BIT 3 OR BIT 2 OR BIT 1 OR BIT 0

Hvis alle 8 bits i akkumulatoren er 0, så vil resultatet altid blive 0; i alle andre tilfælde (hvor mindst 1 bit er 1) vil resultatet blive 1. På den måde kan microprocessoren angive om regneregisteret, hvori resultatet af den sidste operation næsten altid står anført, er 0 eller forskellig fra 0. Således er de to første ligheder blevet udført. For at kunne sammenligne A = B eller A <> B behøver vi således kun at trække de to tal fra hinanden, og derefter undersøge om indholdet i akkumulatoren er 0. Det kan man se på Zero-flag (Zero = nul). Er dette flag 1, så resultatet af den sidste beregning = 0. Det er ikke kun akkumulatoren, der bruger Zero-flag, men heller ikke alle kommandoerne. Om en kommando bruger Zero-flag, kan man se i kommandooversigten i denne bogs tillæg.

Ved "større end" og "mindre end" er proceduren næsten magen til den med "lig med". Efter subtraktionen undersøger man, om tallet i akkumulatoren er mindre end 0 eller større end 0, hvilket jo ses af fortegnsbit'en:

A › B ‹===› A - B › 0 SAND HVIS BIT 7 = 0

A ‹ B ‹===› A - B ‹ 0 SAND HVIS BIT 7 = 1

Mange kommandoer flytter fortegnsbit'en til S-flag (Z80), S betyder SIGN = fortegn. Der kan den aflæses af specielle kommandoer.

Der findes også et Z80 flag, der hedder P/V (da den faktisk ikke benyttes i denne bog, kalder vi den for nemheds skyld P). Den har to funktioner. For det første kan den angive en paritet (lige/ulige, hvilket er uinteressant i øjeblikket) og for det andet fortæller den efter udførelse af aritmetiske operationer, om en fortegnsbit er blevet ændret.

13.8. DET FØRSTE Z80-PROGRAM

Nu da grundlaget for maskinkodeprogrammering er i orden, kan vi gå igang med det første program. Vi vil lave et additionsprogram for 8-bits-tal

Først skal vi meddele programmet hvilke tal, vi ønsker adderet. Der findes ikke nogen INPUT-kommando i maskinsprog; derfor er vi nødt til at være så flinke, at vi selv lægger disse to tal ind i computerens hukommelse. Derfra kan programmet selv hente tallene, når de skal bruges. Det er den første opgave, programmet skal løse. Kommandoen "LD A,(nnnn)" minder om BASIC'ens kendte PEEK-kommando. Den henter en byte fra adressen nnnn og transporterer den til akkumulatoren.

Vi ved hvor den anden byte ligger gemt, men vi kan ikke hente den med LD B,(nnnn) og aflevere den til processoren. Kommandoen findes desværre ikke i Z80-maskinkode. Det er imidlertid muligt, at bruge registerparret HL som pointer på vores byte. Dertil bringer vi adressen til det ønskede register med LD HL,nnnn. Bemærk at udtrykket "nnnn" nu ikke længere skal være indeholdt i en parentes. Det viser, at dette udtryk skal flyttes direkte til HL, og ikke skal stå som adresse for den egentlige værdi.

Med den næste kommando bliver de to bytes endelig adderet. Den lyder "ADD A,(HL)" og resulterer i, at værdien fra akkumulatoren og byten, hvis adresse er gemt i HL bliver adderet. Resultatet lægges tilbage i akkumulatoren. Bliver summen af de to tal højere end 1 bytes maksimale værdi, vil Z80 vise dette ved at sætte sin overflows-bit til 1.

Da det ikke er særligt oplysende at have resultatet liggende i akkumulatoren, hvor vi ikke kan se det, har vi brug for endnu en kommando, der kan bringe

det til et sted i hukommelsen, hvor vi kan læse det med PEEK. Det klares af "LD (nnnn),A". Kommandoen fungerer på samme måde som "LD A,(nnnn); blot i modsat retning.

Til sidst afsluttes programmet med JP FFE0. Dette hop kobler 8502 ind igen, der så overlader roret til BASIC-fortolkeren.

Indtil nu har vi blot programmeret løs, uden at bekymre os om hukommelses-adresserne, hvori det hele foregår. I vores eksempel, kan man næsten selv bestemme, hvor i hukommelsen man vil arbejde; det eneste, man skal passe på er, at computeren ikke bruger det valgte område til noget andet. Det bedste sted at placere sit maskinkodeprogram er i RAM-området fra 1300 til 17FF. Data-byte'ene følger så i slutningen af dette område.

Vores program kommer dermed til at se således ud:

Adresse	Kommando		Kommentar
AB00	LD	A,(AB7F)	hent det første tal i AB7F
AB03	LD	HL,AB7E	det andet tal står i AB7E
AB06	ADD	A,(HL)	adder akk. og det andet tal
AB07	LD	(AB7D),A	gem resultatet
AB0A	RET		Program-afslutning

Adresserne viser, at kommandoerne bruger et varieret antal bytes. Kommandoer, der har indbygget en adresseangivelse, har brug for mindst 3 bytes. En kommando som f.eks. "ADD A,(HL)" er en 1-byte-kommando.

Inden man kan POKE byte'ene ind i hukommelsen, må man vide, hvorledes de ser ud. I tabellen med Z80-kommandoer, kan man slå de enkelte kommandoer efter, og se hvilken opcode, de har. For "LD A,(nnnn)" er det 3A plus to adressebytes. Vær opmærksom på, at Lo-bytes altid angives før Hi-bytes i en adresse. Hele kommandoen udtrykt i tal, lyder således:

3A 7F AB

Her er koderne for resten af programmet:

```
21  7E  AB  (LD HL,AB7E)
86          (ADD A,(HL))
32  7D  AB  (LD (AB7D),A)
C9          (RET)
```

Disse værdier skal POKE's ind i de tilsvarende adresser. Denne opgave varetages af dette korte BASIC-program:

```
10 MEMORY &AAFF
20 FOR I = &AB00 TO &AB0A: READ A: POKE I,A:NEXT
30 DATA &3A, &7F, &AB, &21, &7E, &AB, &86, &32, &7D,
   &AB, &C9
```

Maskinkodeprogrammet kan startes med CALL &AB00. Inden skal tallene, der skal adderes lægges ind i hukommelsen med POKE &AB7F,21 og POKE &AB7E,22. Efter CALL vil PRINT PEEK(&AB70) give resultatet af additionen. De følgende programlinier skal hægtes på det ovenfor viste program, således at det bliver muligt at teste med forskellige værdier.

```
40 INPUT "TAL 1,TAL2"; T1,T2
50 POKE &AB7F,T1:POKE &AB7E,T2
60 CALL &AB00:PRINT PEEK(&AB7D):GOTO 40
```

En subtraktion programmeres på tilsvarende måde. ADD-kommandoen skal dog erstattes af SUB (HL). I DATA-linien skal den 7. værdi ændres fra 86 til 96.

13.9. HVORDAN PROGRAMMERES EN LØKKE?

Hvis man vil gentage en sekvens et antal gange, har man i BASIC to muligheder: FOR-NEXT eller WHILE-WEND. Der er en tredie mulighed, der ikke er nær så komfortabel, og af samme grund heller ikke meget benyttet. Den er nem at lave. I slutningen af en sekvens kan man forhøje en tæller-variabel med 1 og via IF-THEN hoppe til sekvensens start, hvis endnu et gennemløb ønskes.

Uanset hvor ukomfortabel denne metode er, så er det den eneste måde, hvorpå en løkke kan programmeres i maskinkode. I stedet for en variabel benytter vi et processorregister og løkkens længde tælles ned ovenfra. Det nedenfor viste program bestiller ikke andet end at afvente 255 gennemløb (da løkken er tom). Der er ikke vist nogen BASIC-loader, da programmet ikke har nogen synlig virkning. Maskinkodeprogrammet arbejder så hurtigt, at forløbet kun varer en brøkdel af et sekund. Her er programmet:

```
AB00  LD   B,FF      Load løkkens længde
AB02  DEC  B         Formindsk B med 1
AB03  JP   NZ,AB02   Hop, hvis forskellig fra 0
AB06  RET            ellers slut
```

Den første kommando flytter løkkens længde til registeret B. Tallet er indbygget i programmet, og står således anført direkte efter opcoden. Man vil måske spørge, hvorfor det netop er register B, der bruges som tæller. Jo, akkumulatoren er reserveret for regneoperationer, og på lignende vis, findes der bestemte tælle-kommandoer for B-registeret.

DEC-kommandoen (decrement = formindsk med 1) bevirker, at der trækkes 1 fra B. Bliver resultatet af denne subraktion 0, sættes Zero-flag til 1, ellers forbliver den 0. Dette forhold udnyttes af næste kommando. JP NZ hopper kun til den angivne adresse, hvis Zero-flag er 0. Retur-hoppet udføres dermed kun så længe løkkens afslutning endnu ikke er nået. Er Z = 1, så fortsætter Z80 med den næste kommando. Den hedder RET og standser programudførelsen.

13.10. YDERLIGERE ARITMETIKRUTINER

13.10.1. 16-BITS-ADDITION

Med 8-bits-tal kan, som allerede nævnt, kun bearbejdes tal med største værdi 255. Ved 16 bits er det tal med højeste værdi 65535. Z80 processoren er en af de få 8-bits processorer, der kan udføre kommandoer for 16-bits-aritmetik, og desuden er i besiddelse af de nødvendige 16-bits-registre.

For at udføre en 16-bits-addition, skal det første tal lægges i registerparret DE. Med "LD DE,(nnnn)" flyttes byten nnnn til register D. E indeholder værdien af adresse nnnn+1. Her benyttes også pointerformatet Lo-Hi. Den samme kommando findes til HL.

Additionskommandoen for 16 bits lyder "ADD HL,DE". Som man sikkert har gættet, adderes derved de to tal fra DE og HL, og resultatet lægges i HL. Herfra kan det flyttes til hukommelsen med "LD (nnnn),HL".

Hele programmet ser således ud:

```
AB00  LD   DE,(AB7E)
AB04  LD   HL,(AB7C)
AB07  ADD  HL,DE
AB08  LD   (AB7A),HL
AB0B  RET
```

Vær opmærksom på, at hver LD-kommando samtidig bearbejder to bytes. Loaderen ser således ud:

```
10 DATA &ED,&5B,&7E,&AB
11 DATA &2A,&7C,&AB"
12 DATA &19
13 DATA &22,&7A,&AB
14 DATA &C9
20 FOR I = &AB00 TO &AB0B:READ A:POKE I,A:NEXT
30 INPUT "TAL1, TAL2 "; T1,T2
40 POKE &AB7E,T1 AND 255:POKE &AB7F,INT(T1/256)
50 POKE &AB7C,T2 AND 255:POKE &AB7D,INT(T2/256)
60 CALL &AB00:PRINT PEEK(&AB7A)+ 256 * PEEK(&AB7B)
70 GOTO 50
```

Inden dette program udføres, skal man sætte de to initialiseringer fra additionsprogrammet.

Også her kan man eksperimentere med forskellige værdier.

13.10.2. MULTIPLIKATION

For at multiplicere to tal, må man addere et antal gange, hvilket vi allerede har beskæftiget os med i kapitel 14.6.3. I BASIC ville denne algoritme blive omsat til:

```
10 INPUT "TAL 1, TAL 2"; T1,T2: e=0
20 FOR i = 1 TO T1
30 e = e + T2: NEXT
40 PRINT "RESULTAT ";e
```

Som det ses, består programmet stort set kun af en løkke og en addition. Vi har allerede programmeret begge dele i maskinsprog. Multiplikation af to 8-bits-tal resulterer altid i et 16-bits-tal. Derfor bør man tage højde for dette på samme måde i 16-bits-additions-rutinen.

Inde i løkken må der kun være additions-kommandoen. Alle andre kommandoer tjener til klargøring af registre og lagring af resultater. Med den første kommando (se listningen) loades det første tal til akkumulatoren. Det skulle egentlig loades i register B, men da dette register ikke kan loades direkte fra hukommelsen, må det mellemlagres i akkumulatoren, inden det med kommando nummer to flyttes til register B. Således er løkkens længde bestemt. De andre 8-bits-tal adderes hele tiden til registerparret HL. På den måde når de til register E. Da ADD HL,DE altid adderer register D med, skal dette

sættes til 0 først. HL skal også være sat til 0, inden løkken startes. Det klares af de to næste kommandoer.

Så startes løkken med med ADD-kommandoen. Da registerparret HL kun ændres af additionskommandoen, indeholder den altid det sidste mellemresultat.

Resten af maskinkodeprogrammet kender vi. DEC B og JP NZ,130D markerer løkkens afslutning. LD(137C),HL sørger for at gemme resultatet og JP FFE0 afslutter programmet. Her er listningen:

```
AB00  LD   A,(AB7F)   Tal 1
AB03  LD   B,A        til B
AB04  LD   A,(AB7E)   Tal 2
AB07  LD   E,A        til E
AB08  LD   D,00       Slet D
AB0A  LD   HL,0000    Slet HL
AB0D  ADD  HL,DE      adder
AB0E  DEC  B          decrementer (formindsk)
AB0F  JP   NZ,AB0D    Hop tilbage, hvis ikke 0
AB12  LD   (AB7C),HL  Gem resulatet
AB15  RET             Returner til BASIC
```

Selvfølgelig findes der også en BASIC-loader til dette program:

```
10 DATA &3A,&7F,&AB
11 DATA &47
12 DATA &3A,&7E,&AB
13 DATA &5F
14 DATA &16,&00
15 DATA &21,&00,&00
16 DATA &19,&05
17 DATA &C2,&0D,&AB
18 DATA &22,&7C,&AB
19 DATA &C9
20 FOR I= &AB00 TO &AB15:READ A:POKE I,A:NEXT
30 INPUT ”TAL1, TAL2”; T1,T2
40 POKE &AB7F,T1:POKE &AB7E,T2
50 CALL &AB00:PRINT PEEK(&AB7C)+256*PEEK(&AB7D)
60 GOTO 30
```

Her skal man ligeledes indsætte initialiseringslinierne.

13.11. NYTTIGE MASKINKODERUTINER

I dette afsnit vil vi forene det behagelige med det nyttige. Derfor vises nogle rutiner, der ikke kun tjener som eksempler, men også kan bruges til noget nyttigt.

Lad os begynde med noget destruktivt; sletning af hele hukommelsesområder. I BASIC kan det gøres med en FOR-NEXT løkke og nogle POKE-kommandoer, men det er relativt langsomt at bruge den metode. Det er en ideel opgave for et maskinkodeprogram. Her gælder der dog et lille minus; der kan kun slettes 256 bytes i een arbejdsgang, idet programmet ellers ville være for indviklet at gå i gang med.

Tilsvarende SAVE-kommandoen for bestemte adresseområder, er det kun nødvendigt at angive startadressen og antallet af bytes, der skal slettes.

HL tjener som pointer på de aktuelle bytes, og bliver forhøjet for hvert gennemløb i løkken. Register B indeholder som sædvanligt løkkens længde (antallet af bytes for sletning). I løkken loades byten, der angives af HL, med 0. Begge registre ajourføres og der returneres med JP NZ,(xxxx) til løkkens start.

Dermed burde vort program være klart, men vi tillader os at "indbygge" lidt luksus. I løkkens slutning indeholder HL adressen på den næste byte, der skal slettes (nulstilles). Denne adresse flyttes tilbage i hukommelsen til det sted, hvor kommandoen hentede sin første adresse. Ønsker man at fortsætte sletningen, er det ikke nødvendigt at POKE sig frem. Man kan i stedet for bruge sin CALL-kommando.

En tilsvarende fremgangsmåde kan anvendes til løkkens længde. Da denne værdis adresse ikke ændres i løbet af kørselen, bortfalder ligeledes POKE-kommandoen efter første gennemløb. Ønsker man således at slette 100 bytes i skridt af 10 bytes, kan man bruge følgende kommandorække:

```
POKE &AB7F,10:POKE &AB7D,LO-BYTE:POKE &AB7E,HI-BYTE
FOR I=1 TO 10:CALL &AB00:NEXT
```

Der er endnu et punkt med hensyn til løkke-længde, der skal bemærkes. Hvis en rutine med løkke-længde 0 kaldes, slettes 256 bytes, idet der før den første JP (betingelse) sker en formindskelse af B. 0 minus 1 svarer til tallet 256 for alle mikroprocessorer (kig på afsnittet om binær aritmetik og læg 1 til 255). Programmet ser således ud:

```
AB00  LD   HL,(AB7D)
AB03  LD   A,(AB7F)
AB06  LD   B,A
AB07  LD   (HL),0
AB09  INC  HL
AB0A  DEC  B
AB0B  JP   NZ,AB07
AB0E  LD   (AB7D),HL
AB11  RET
```

```
10 DATA &2A,&7D,&AB
11 DATA &3A,&7F,&AB
12 DATA &47
13 DATA &36,&00
14 DATA &23
15 DATA &05
16 DATA &C2,&07,&AB
17 DATA &22,&7D,&AB
18 DATA &C9
20 FOR I=&AB00 TO &AB11:READ A:POKE I,A:NEXT
30 INPUT"STARTADRESSE ";A
40 POKE &AB7D,A-INT(A/256):POKE &AB7E,INT(A/256)
50 INPUT"LÆNGDE ";B
60 POKE &AB7F,B
70 CALL &AB00:GOTO 20
```

Den anden rutine, vi vil anvende, kopierer hukommelsesblokke. Da vi kan gøre brug af en særlig Z80 kommando, er rutinen kortere og grænsen på de 256 bytes bortfalder. Den særlige kommando hedder LDIR. Den kopierer ganske enkelt blokke af hukommelse uanset størrelse. Denne størrelse skal angives i registerparret BC. Z80 skal yderligere hente startadressen på området, der skal kopieres i HL og målområdet i DE. Derefter kan LDIR gå i aktion. Den kopierer byte'en, hvis adresse er anført i HL til adressen angivet i DE. Så forhøjes HL og DE; de peger på de næste bytes. Z80 formindsker BC. Så længe BC er forskellig fra nul (0) gentages LDIR automatisk. På denne måde kan een enkelt kommando erstatte en løkke!

```
AB20  LD   HL,(AB6E)
AB23  LD   DE,(AB6C)
AB27  LD   BC,(AB6A)
AB2B  LDIR
AB2D  RET
```

```
10 DATA &2A,&6E,&AB
11 DATA &ED,&5B,&6C,&AB
12 DATA &ED,&4B,&6A,&AB
13 DATA &ED,&B0
14 DATA &C9
20 FOR I=&AB20 TO &AB2D:READ A:POKE I,A:NEXT
30 INPUT"KILDEOMRÅDE";A
40 POKE &AB6E,A-INT(A/256):POKE &AB6F,INT(A/256)
50 INPUT"MÅLOMRÅDE";B
60 POKE &AB6C,B-INT(B/256):POKE &AB6D,INT(B/256)
70 INPUT"LÆNGDE";C
80 POKE &AB6A,C-INT(C/256):POKE &AB6B,INT(C/256)
90 CALL &AB20:GOTO 30
```

13.12. ADRESSERINGSMULIGHEDERNE

Hvis man kigger på kommandolisten, vil man opdage, at operanden ikke kun består af registre, så som A, B, C o.s.v., men også af parenteser og lignende. Ved en kommando som ADD A,B er det let at se, hvad der sker; her adderes registrene A (Akkumulator) og B. Men hvad betyder ADD A,(HL) ?

Et udtryk i parentes angiver adressen på en byte i hukommelsen og ikke operanden selv. "(xx)" betyder altså "indholdet af adresse xxxx" (et bogstav = en byte; xx er således 2 bytes = 16 bits = 4 hextal). Denne metode kaldes også for direkte adressering. En anden metode er indirekte adressering, hvor operanden ikke angiver adressen på de data, der skal berøres, men stedet hvor denne adresse kan findes. Kommandoen ADD A,(HL) arbejder således på den måde, at procesoren først "kigger efter", hvilken værdi, der ligger gemt i register HL. Denne værdi er pointer for adressen, hvori byten, der skal adderes med AKK. befinder sig. Særpræget måde at klare tingene på, mener læseren måske, men konstruktørerne siger derimod "ganske smart". Denne indirekte adressering kan jo anvendes til at manipulere hele blokke af bytes, på samme måde som en tabel.

En anden version af den netop omtalte, er indirekte-indiceret adressering. (IX+D) betyder i dette tilfælde, at den egentlige adresse udgøres af resultatet af additionen af register IX og konstanten D.

Den sidste i rækken er den relative adressering, der kun kendes fra bestemte hop. Tilsvarende den indicerede adressering følges opcoden af en konstant. Denne adderes til program-counter. Er bytens bit 7 på 1, behandles den som

et negativt tal (hvilket iøvrigt også er gældende for den indirekte indicerede adressering). Resultatet bliver et baglæns hop. Da PC efter indlæsning af konstanten allerede peger på næste kommando, er den maksimale skridtlængde fremad 129 og baglæns 127.

13.13. Z80-KOMMANDOER

Hvis man betragter tillæget med Z80-OPCODES (i dette er alle kommandoerne opført med deres koder og berørte flag), så vil man kunne fastslå, at der findes flere grupper af kommandoer, der kun adskiller sig fra hinanden ved deres operander (adresseringsmåde). Det er disse grupper, vi vil beskrive i det følgende. Det er ikke nødvendigt at lære alle betydningerne udenad; de er tænkt anvendt som en slags opslagsliste og til at give et overblik over de mange muligheder, der gives med programmering i maskinkode.

Da kommandoer til forskellige operander altid fungerer ens, vil disse blive angivet via ubekendte (x, n, o.s.v.).

ADC A,x

Operanden x adderes med Akk. (akkumulatoren). Der tages højde for evt. overflow (carry-bit). Resultatet lægges tilbage i Akk., og nyt overflow angives i carry-bit. Efter udførelse er de tilhørende flag blevet sat.

ADC HL,x

Kommandoen svarer til den forrige, men foretager 16-bits-addition. Operanden og Carry-bit adderes til HL registerparret. Resultatet gemmes i HL, og flagene sættes tilsvarende.

ADD A,x

Operanden adderes til Akk. og resultatet lægges tilbage. Der tages ikke hensyn til Carry-bit, men er som ved alle andre additioner sat, svarende til resultatet.

ADD HL,x

Som ADC HL,x, men Carry-bit medtages ikke ved additionen.

ADD IX,x

Operanden adderes til indexregisteret IX og resultatet gemmes igen. Carry-bit aktualiseres. Alle andre flag berøres ikke.

ADD IY,x

Som ADD IX,x, men indexregister IY benyttes.

AND x

Operanden og Akk. AND'es, og resultatet lagres i Akk. Flagene sættes alt efter resultatet. Carry-bit sættes til 0 (AND giver aldrig overflow).

BIT n,x

Operanden x's bit n testes og flyttet til Zero-flag. Er den valgte bit 1, så er Zero-flag = 0, ellers omvendt. Flagene S og P indeholder tilfældige værdier.

CALL nn

Kald af underprogram med start i adresse nnnn (svarer nogenlunde til GOSUB i BASIC).

CALL betingelse,nn

Underprogrammet i adresse nnnn kaldes kun, hvis betingelsen er sand (opfyldt). Betingelsen kan testes af flagene S, Z. P og (Carry). Alt efter tilstanden kan der hoppes til udførelse af et underprogram eller fortsættes på normal vis.

CCF

Komplementerer Carry-bit (-1)0, 0-)1).

CP x

Operanden x sammenlignes med Akk. D.v.s., at den subtraheres, men resultatet gemmes ikke. Indholdet i Akk. bibeholdes. Er operanden mindre end Akk., så bliver S=1, er den større eller lig med, så bliver S=0. Er værdierne ens, så er Z=1, forskellige så Z=0.

CPD

HL bruges som pointer på en adresse, der skal sammenlignes med Akk. Hvis værdierne er ens, så er Z=1, hvis Akk. er større eller lig med, så er S=0. Er S=1 er Akk. mindre end byten fra adressen. Så decrementeres (formindskes med 1) HL og BC. Er indholdet af BC=0, så sættes P-flaget til 0, ellers 1. BC kan således anvendes som byte-tæller.

CPDR

Ligesom CPD, men operationen gentages, indtil der er lighed (Z=1) eller BC=0 (P-flag=0).

CPI

Ligesom CPD. HL decrementeres dog ikke, men incrementeres i stedet for (forhøjes i stedet for formindskes).

CPIR

Ligesom CPDR. HL incrementeres dog.

CPL

Alle bits i Akk. inverteres (0 ændres til 1, og omvendt).

DAA

Efter en aritmetisk operation (ADC, ADD, DEC, INC, NEG, SBC, SUB) korrigeres forkerte BCD-cifre og flag.

DEC x

Den viste operand formindskes med 1 og genlagres (genskrives). Foruden ved registrene BC, DE, HL, IX, IY og SP sættes de tilhørende flag alt efter resultatet.

DI

Efter udførelse af kommandoen ignoreres alle interrupts af Z80 (maskerbare interrupts).

DJNZ e

B formindskes med 1. Er indholdet i B ikke 0, adderes den relative hopafstand e til adressen på den efterfølgende kommando. Programudførelsen fortsætter fra den nye adresse.

EI

Efter denne kommando frigives de maske'rbare interrupts igen, d.v.s. at processoren udfører den særlige rutine ved interruptanmodning.

EX,x,y

De to angivne operander byttes (SWAP, EXCHANGE). X kan også stå for SP (Stack-pointer). Er dette tilfældet byttes anden operand med det øverstplacerede element i STACK.

EXX

Registrene BC, DE og HL ombyttes med andenregistre.

HALT

Z80-processoren standser alt arbejde, indtil der anmodes om interrupt.

IM n

Valg af interrupt-mode.
Mode 0: Chippen, der ønsker interrupt, skal have en kommando klar ved databus'en.
Mode 1: Z80-processoren hopper til adresse $0038 og udfører interruptrutine herfra.
Mode 2: Chippen leverer den mindstbetydende del af den adresse, hvis mestbetydende del er lagret i register I. Under denne adresse ligger en pointer på starten af interruptrutinen.

IN x,(C)

Register C angiver hvilken port, byten i register x skal læses fra.

IN A,(n)

Akk. loades med en byte fra port n.

INC x

Ligesom DEC, men operanden forhøjes med 1.

IND

Register HL anvendes som pointer på en adresse, hvori der læses en byte fra en port. Register C angiver portens nummer. Desuden decrementeres HL og B. Er B=0, så sættes Zero-flag til 1. Alle øvrige flag har tilfældige værdier, undtagen Carry-flag.

INDR

Ligesom IND, men operationen gentages indtil B=0.

INI

Ligesom IND. Dog incrementeres HL.

INIR

Ligesom INDR. Dog incrementeres HL.

JP nn

Programmet hopper til adresse nnnn (svarer til GOTO i BASIC).

JP betingelse,nn

Hopper, hvis betingelsen opfyldes (sand). Se også CALL.

JP x

Hopper til adressen, der er lagret i operand x (HL, IX, IY).

JR n og JR betingelse,n

Som JP-kommandoen. Der er tale om relativt hop, dvs. at den følgende byte(n) adderes til PC.

LD x,y

Denne kommando hører til den mest udbredte gruppe. Alle har et punkt til fælles. Indholdet af den anden operand overføres til den første operand. Det kan således lade sig gøre, at indholdet af Akk. flyttes til en adresse nnnn, og omvendt. Ligesådan er det med 16-bits-register kommandoer. Skal et 16-bits-register flyttes til en adresse, lægges Lo-byte i nnnn og Hi-byte i nnnn+1. Denne metode anvendes altid i computerverdenen (læg mærke til det specielle: først Low, så High!!).

LDD

HL og DE bruges som pointer på en byte i hukommelsen. Indholdet af den af HL adresserede location flyttes til den af DE adresserede. Så decrementeres HL, DE, BC. Er BC=0 så sættes P-flag til 0. På denne måde kan BC anvendes som tæller.

LDDR

Ligesom LDD, men kommandoen gentages indtil BC <> 0.

LDI

Ligesom LDD, dog incrementeres registrene HL og DE.

LDIR

Ligesom LDDR, HL og DE incrementeres dog.

HNEG

Byten i Akk. negateres (multipliceres med -1). Resultatet gemmes i Akk. og flagene aktualiseres. P er 1, hvis Akk. tidligere var $80 og C er 1, hvis Akk var 0.

NOP

Kommandoen udfører intet, men berøver processortid (bruges som forsinkelsesenhed).

OR x

Akk. og operanden OR'es. Resultatet gemmes i Akk. Flag aktualiseres. Carry-bit sættes til 0, da der ikke kan forekomme overflow ved OR.

OTDR

Hl bruges som pointer på en adresse, hvis indhold skal udlæses via port (C). HL og B decrementeres. Dette gentages indtil B=0. Foruden Carry-bit og Z (der er sat til 1), modtager alle flag tilfældige værdier.

OTIR

Som OTDR. Hl incrementeres dog i stedet.

OUT (C),x

Operanden x udlæses via en port, der angives af register C.

OUT (n),A

Akk. udlæses via port n.

OUTD

HL er pointer på en adresse, hvis indhold skal udlæses via port (C). HL og B decrementeres. Er B=0, så sættes Zero-flag til 1. De øvrige flag, bortset fra Carry), modtager tilfældige værdier.

OUTI

Som OUTD. Dog incrementeres HL.

POP x

Registerparret i operanden loades fra STACK og SP aktualiseres. AF står for Akk. & flag.

PUSH x

Registerparret i operanden flyttes til STACK. SP aktualiseres. AF står for Akk. & flag.

RES n,x

Bit n i operanden x slettes.

RET og RET-betingelse.

Denne kommando resulterer i et retur-hop fra en underrutine (svarer til RETURN i BASIC). Der kan angives en betingelse, der udløser retur-hop, hvis en betingelse opfyldes (er sand); et flag indeholder en bestemt værdi.

RETI

Udløser retur-hop fra interrupt-rutine.

RETN

Udløser retur-hop fra NMI-rutine.

RL x og RLA

Operanden roteres venstre-om, hvorved Carry-bit forskydes til bit 0 og bit 7 flyttes til Carry-bit. RL-kommandoer ændrer alle flag. RLA-kommandoen (ikke RL A) er en undtagelse. Den virker som RL A, men påvirker kun Carry-flag.

RLC x og RLCA

Operanden roteres venstre-om. Bit 7 flyttes til Carry. Bortset fra RLCA, forandrer RLC-kommandoer alle flag.

RLD

Denne kommando udfører en BCD-rotation. Den venstre halvdel af en adresse, hvis nummer ligger i HL, flyttes til den højre halvdel af Akk. Den højre halvdel af adressen flyttes til den venstre, og den oprindelige højre halvdel af Akk. flyttes til den højre halvdel af adressen. Flagene S, Z og P påvirkes.

RR x og RRA

Disse kommandoer fungerer som RL x og RLA, dog roteres højre-om.

RRC x og RRCA

Som RLD, dog højre-rotation. Den nederste Akk.-nibble (halve byte) flyttes til den øvre, forskudt af den HL adresserede location. Den øverste adresse-nibble flyttes til den nedre, og den nedre flyttes til den nedre Akk.-halvdel.

RST n

Ved adresse n startes et underprogram.

SBC A,x

Operanden x subtraheres fra Akk. Der tages hensyn til overflow i Carry. Resultatet lægges igen i Akk. og en ny overflow gemmes i Carry. Efter udførelse er de tilsvarende flag blevet sat.

SBC HL,x

Denne kommando svarer til den forrige, blot er det en 16-bits-addition. Nu subtraheres operanden og Carry fra HL. Resultatet lægges igen i HL, og flagene sættes.

SFC

Carry-bit sættes til 1.

SET n,x

Bit n i operanden x sættes til 1.

SLA x

Operanden forskydes mod venstre. Bit 0 udfyldes med et 0. Bit 7 flyttes til Carry. Flagene sættes tilsvarende.

SRA x

Operanden forskydes mod højre, bit 7 (fortegn) bibeholdes. Bit 0 flyttes til Carry-bit. Alle flag berøres.

SRL x

Operanden forskydes mod højre. Bit 7 fyldes med 0. Bit 0 flyttes til Carry. Alle flag påvirkes.

SUB x

Operanden trækkes fra Akk. Flagene sættes og resultatet gemmes igen i Akk.

XOR x

Operanden XOR'es (eXclusiv OR'es) med Akk. Resultatet lægges igen i Akk. Flagene aktualiseres og Carry-bit slettes. (XOR giver aldrig overflow).

13.14. Z80-PROCESSORENS OPCODES

Mnemonic		Opcode	Flags	Beskrivelse
ADC	A,A	8F	S Z P C	Adderer Carry og Akk. til Akk.
ADC	A,B	88	S Z P C	... til B
ADC	A,C	89	S Z P C	... til C
ADC	A,D	8A	S Z P C	... til D
ADC	A,E	8B	S Z P C	... til E
ADC	A,H	8C	S Z P C	... til H
ADC	A,L	8D	S Z P C	... til L
ADC	A,n	CEnn	S Z P C	... til følgende byte
ADC	A,(HL)	8E	S Z P C	... til adresse i HL
ADC	A,(IX+d)	DD8Edd	S Z P C	... til adresse IX+d
ADC	A,(IY+d)	FD8Edd	S Z P C	... til adresse IY+d
ADC	HL,BC	ED4A	S Z P C	Adderer Carry og HL til BC
ADC	HL,DE	ED5A	S Z P C	... til DE
ADC	HL,HL	ED6A	S Z P C	... til HL
ADC	HL,SP	ED7A	S Z P C	... til SP
ADD	A,A	87	S Z P C	Adderer Akk. til Akk.
ADD	A,B	80	S Z P C	... til B
ADD	A,C	81	S Z P C	... til C
ADD	A,D	82	S Z P C	... til D
ADD	A,E	83	S Z P C	... til E
ADD	A,H	84	S Z P C	... til H
ADD	A,L	85	S Z P C	... til L
ADD	A,n	C6nn	S Z P C	... til følgende byte
ADD	A,(HL)	86	S Z P C	... til adresse i HL
ADD	A,(IX+d)	DD86dd	S Z P C	... til adresse IX+d
ADD	A,(IY+d)	FD86dd	S Z P C	... til adresse IY+d
ADD	HL,BC	09	C	Adderer HL til BC
ADD	HL,DE	19	C	... til DE
ADD	HL,HL	29	C	... til HL
ADD	HL,SP	39	C	... til SP
ADD	IX,BC	DD09	C	Adderer IX til BC
ADD	IX,DE	DD19	C	... til DE
ADD	IX,HL	DD29	C	... til HL
ADD	IX,SP	DD39	C	... til SP
ADD	IY,BC	FD09	C	Adderer IY til BC
ADD	IY,DE	FD19	C	... til DE
ADD	IY,HL	FD29	C	... til HL
ADD	IY,SP	FD39	C	... til SP
AND	A	A7	S Z P C=0	AND Akk. med Akk.

AND	B	A0	S Z P C=0	... med B
AND	C	A1	S Z P C=0	... med C
AND	D	A2	S Z P C=0	... med D
AND	E	A3	S Z P C=0	... med E
AND	H	A4	S Z P C=0	... med H
AND	L	A5	S Z P C=0	... med L
AND	n	E6nn	S Z P C=0	... med følgende byte
AND	(HL)	96	S Z P C=0	... med adresse i HL
AND	(IX+d)	DD96dd	S Z P C=0	... med adresse IX+d
AND	(IY+d)	FD96dd	S Z P C=0	... med adresse IY+d
BIT	0,A	CB47	Z	Tester Akk.'s Bit 0
BIT	0,B	CB40	Z	Tester B's Bit 0
BIT	0,C	CB41	Z	... C's
BIT	0,D	CB42	Z	... D's
BIT	0,E	CB43	Z	... E's
BIT	0,H	CB44	Z	... H's
BIT	0,L	CB45	Z	... L's
BIT	0,(HL)	CB46	Z	... adresse i HL
BIT	0,(IX+d)	DDCBdd46	Z	... adresse IX+d
BIT	0,(IY+d)	FDCBdd46	Z	... adresse IY+d
BIT	1,A	CB4F	Z	Tester Akk.'s Bit 1
BIT	1,B	CB48	Z	Tester B's Bit 1
BIT	1,C	CB49	Z	... C's
BIT	1,D	CB4A	Z	... D's
BIT	1,E	CB4B	Z	... E's
BIT	1,H	CB4C	Z	... H's
BIT	1,L	CB4D	Z	... L's
BIT	1,(HL)	CB4E	Z	... adresse i HL
BIT	1,(IX+d)	DDCBdd4E	Z	... adresse IX+d
BIT	1,(IY+d)	FDCBdd4E	Z	... adresse IY+d
BIT	2,A	CB57	Z	Tester Akk.'s Bit 2
BIT	2,B	CB50	Z	Tester B's Bit 2
BIT	2,C	CB51	Z	... C's
BIT	2,D	CB52	Z	... D's
BIT	2,E	CB53	Z	... E's
BIT	2,H	CB54	Z	... H's
BIT	2,L	CB55	Z	... L's
BIT	2,(HL)	CB56	Z	... adresse i HL
BIT	2,(IX+d)	DDCBdd56	Z	... adresse IX+d
BIT	2,(IY+d)	FDCBdd56	Z	... adresse IY+d 2
BIT	3,A	CB5F	Z	Tester Akk.'s Bit 3
BIT	3,B	CB58	Z	Testet B's Bit 3
BIT	3,C	CB59	Z	... C's
BIT	3,D	CB5A	Z	... D's

BIT	3,E	CB5B	Z	... E's
BIT	3,H	CB5C	Z	... H's
BIT	3,L	CB5D	Z	... L's
BIT	3,(HL)	CB5E	Z	... adresse i HL
BIT	3,(IX+d)	DDCBdd5E	Z	... adresse IX+d
BIT	3,(IY+d)	FDCBdd5E	Z	... adresse IY+d
BIT	4,A	CB67	Z	Tester Akk.'s Bit 4
BIT	4,B	CB60	Z	Tester B's Bit 4
BIT	4,C	CB61	Z	... C's
BIT	4,D	CB62	Z	... D's
BIT	4,E	CB63	Z	... E's
BIT	4,H	CB64	Z	... H's
BIT	4,L	CB65	Z	... L's
BIT	4,(HL)	CB66	Z	... adresse i HL
BIT	4,(IX+d)	DDCBdd66	Z	... adresse IX+d
BIT	4,(IY+d)	FDCBdd66	Z	... adresse IY+d
BIT	5,A	CB6F	Z	Tester Akk.'s Bit 5
BIT	5,B	CB68	Z	Tester B's Bit 5
BIT	5,C	CB69	Z	... C's
BIT	5,D	CB6A	Z	... D's
BIT	5,E	CB6B	Z	... E's
BIT	5,H	CB6C	Z	... H's
BIT	5,L	CB6D	Z	... L's
BIT	5,(HL)	CB6E	Z	... adresse i HL
BIT	5,(IX+d)	DDCBdd6E	Z	... adresse IX+d
BIT	5,(IY+d)	FDCBdd6E	Z	... adresse IY+d
BIT	6,A	CB77	Z	Tester Akk.'s Bit 6
BIT	6,B	CB70	Z	Tester B's Bit 6
BIT	6,C	CB71	Z	... C's
BIT	6,D	CB72	Z	... D's
BIT	6,E	CB73	Z	... E's
BIT	6,H	CB74	Z	... H's
BIT	6,L	CB75	Z	... L's
BIT	6,(HL)	CB76	Z	... adresse i HL
BIT	6,(IX+d)	DDCBdd76	Z	... adresse IX+d
BIT	6,(IY+d)	FDCBdd76	Z	... adresse IY+d
BIT	7,A	CB7F	Z	Tester Akk.'s Bit 7
BIT	7,B	CB78	Z	Tester B's Bit 7
BIT	7,C	CB79	Z	... C's
BIT	7,D	CB7A	Z	... D's
BIT	7,E	CB7B	Z	... E's
BIT	7,H	CB7C	Z	... H's
BIT	7,L	CB7D	Z	... L's
BIT	7,(HL)	CB7E	Z	... adresse i HL

BIT	7,(IX+d)	DDCBdd7E	Z	... adresse IX+d
BIT	7,(IY+d)	FDCBdd7E	Z	... adresse IY+d
CALL	nn	CDnnnn		Kalder underrutine nnnn
CALL	C,nn	DCnnnn		... når Carry=1
CALL	M,nn	FCnnnn		... når S=1 (minus)
CALL	NC,nn	D4nnnn		... når Carry=0
CALL	NZ,nn	C4nnnn		... når Z=0
CALL	P,nn	F4nnnn		... når S=0 (plus)
CALL	PE,nn	ECnnnn		... når P=1
CALL	PO,nn	E4nnnn		... når P=0
CALL	Z,nn	CCnnnn		... når Z=1
CCF	3F		C	Komplementerer Carry-Flag
CP	A	BF	S Z P C	Sammenligner A med A
CP	B	B8	S Z P C	... med B
CP	C	B9	S Z P C	... med C
CP	D	BA	S Z P C	... med D
CP	E	BB	S Z P C	... med E
CP	H	BC	S Z P C	... med H
CP	L	BD	S Z P C	... med L
CP	n	FEnn	S Z P C	... med følgende byte
CP	(HL)	BE	S Z P C	... med adresse i HL
CP	(IX+d)	DDBEdd	S Z P C	... med adresse IX+d
CP	(IY+d)	FDBEdd	S Z P C	... med adresse IY+d
CPD		EDA9	S Z P	sammenligner og decrement
CPDR		EDB9	S Z P	bloksammenligning og decrement
CPI		EDA1	S Z P	sammenligning og increment
CPIR		EDB1	S Z P	bloksammenligning og increment
CPL		2F		Komplementerer Akk.
DAA		27	S Z P C	Akk.'s BCD-tilpasning
DEC	A	3D	S Z P	Decrementerer Akk.
DEC	B	05	S Z P	... B
DEC	BC	0B		... BC
DEC	C	0D	S Z P	... C
DEC	D	15	S Z P	... D
DEC	DE	1B		... DE
DEC	E	1D	S Z P	... E
DEC	H	25	S Z P	... H
DEC	HL	2B		... HL
DEC	IX	DD2B		... IX
DEC	IY	FD2B		... IY
DEC	L	2D	S Z P	... L
DEC	SP	3B		... SP

DEC	(HL)	35	S Z P	... adresse i HL
DEC	(IX+d)	DD35dd	S Z P	... adresse IX+d
DEC	(IY+d)	FD35dd	S Z P	... adresse IY+d
DI		F3		Forhindrer Interrupt
DJNZ	e	10ee		Decrement & hop ved ‹› 0
EI		FB		Interrupt frigives
EX	AF,AF”	08		bytter Akk./Flags med anden.
EX	DE,HL	EB		bytter DE og HL
EX	(SP),HL	E3		bytter HL med STACK-byte
EX	(SP),IX	DDE3		... IX
EX	(SP),IY	FDE3		... IY
EXX		D9		... BC,DE,HL med andet register
HALT		76		Standser Z80 indtil næste Interrupt
IM	0	ED46		Interruptmode 0
IM	1	ED56		Interruptmode 1
IM	2	ED5E		Interruptmode 2
IN	A,(C)	ED78	S Z P	Læser byte fra Port C i A
IN	A,(n)	DBnn		... fra Port n
IN	B,(C)	ED40	S Z P	... i B
IN	C,(C)	ED48	S Z P	... i C
IN	D,(C)	ED50	S Z P	... i D
IN	E,(C)	ED58	S Z P	... i E
IN	H,(C)	ED60	S Z P	... i H
IN	L,(C)	ED68	S Z P	... i L
INC	A	3C	S Z P	Incrementerer A
INC	B	04	S Z P	... B
INC	BC	03		... BC
INC	C	0C	S Z P	... C
INC	D	14	S Z P	... D
INC	DE	13		... DE
INC	E	1C	S Z P	... E
INC	H	24	S Z P	... H
INC	HL	23		... HL
INC	IX	DD23		... IX
INC	IY	FD23		... IY
INC	L	2C	S Z P	... L
INC	SP	33		... SP
INC	(HL)	34	S Z P	... adresse i HL
INC	(IX+d)	DD34dd	S Z P	... adresse IX+d
INC	(IY+d)	FD34dd	S Z P	... adresse IY+d
IND		EDAA	Z	indlæs med decrement
INDR		EDBA	Z=1	indlæs blok med decrement

INI		EDA2	Z	indlæs med increment
INIR		EDB2	Z=1	indlæs blok med increment.
JP	C,nn	DAnnnn		hop til nn, hvis C=1
JP	M,nn	FAnnnn		... til nn, hvis S=1
JP	NC,nn	D2nnnn		... til nn, hvis C=0
JP	nn	C3nnnn		... til nn
JP	NZ,nn	C2nnnn		... til nn, hvis Z=0
JP	P,nn	F2nnnn		... til nn, hvis S=0
JP	PE,nn	EAnnnn		... til nn, hvis P=1
JP	PO,nn	E2nnnn		... til nn, hvis P=0
JP	Z,nn	CAnnnn		... til nn, hvis Z=1
JP	(HL)	E9		... til adresse i HL
JP	(IX)	DDE9		... til adresse i IX
JP	(IY)	FDE9		... til adresse i IY
JR	C,e	38ee		betinget hop, hvis C=1
JR	e	18ee		... til PC+e
JR	NC,e	30ee		... hvis C=0
JR	NZ,e	20ee		... hvis Z=0
JR	Z,e	28ee		... hvis Z=1
LD	A,A	7F		Load Akk. med Akk.
LD	A,B	78		... med B
LD	A,C	79		... med C
LD	A,D	7A		... med D
LD	A,E	7B		... med E
LD	A,H	7C		... med H
LD	A,I	ED57	S Z P	... med I
LD	A,L	7D		... med L
LD	A,n	3Enn		... med efterfølgende byte
LD	A,R	ED5F	S Z P	... med R
LD	A,(BC)	0A		... fra adresse til BC
LD	A,(DE)	1A		... fra adresse til DE
LD	A,(HL)	7E		... fra adresse til HL
LD	A,(IX+d)	DD7Edd		... fra adresse IX+d
LD	A,(IY+d)	FD7Edd		... fra adresse IY+d
LD	A,(nn)	3Annnn		... fra adresse nn
LD	B,A	47		Load B med Akk.
LD	B,B	40		... med B
LD	B,C	41		... med C
LD	B,D	42		... med D
LD	B,E	43		... med E
LD	B,H	44		... med H
LD	B,L	45		... med L
LD	B,n	06nn		... med efterfølgende byte
LD	B,(HL)	46		... fra adresse til HL

LD	B,(IX+d)	DD46dd	... fra adresse IX+d
LD	B,(IY+d)	FD46dd	... fra adresse IY+d
LD	BC,nn	01nnnn	Load BC med følgende bytes
LD	BC,(nn)	ED4Bnnnn	... med nnnn og nnnn+1
LD	C,A	4F	Load C med Akk.
LD	C,B	48	... med B
LD	C,C	49	... med C
LD	C,D	4A	... med D
LD	C,E	4B	... med E
LD	C,H	4C	... med H
LD	L,L	4D	... med L
LD	C,n	0Enn	... med efterfølgende byte
LD	C,(HL)	4E	... fra adresse til HL
LD	C,(IX+d)	DD4Edd	... fra adresse IX+d
LD	C,(IY+d)	FD4Edd	... fra adresse IY+d
LD	D,A	57	Loader D med Akk.
LD	D,B	50	... med B
LD	D,C	51	... med C
LD	D,D	52	... med D
LD	D,E	53	... med E
LD	D,H	54	... med H
LD	D,L	55	... med L
LD	D,n	16nn	... med efterfølgende byte
LD	D,(HL)	56	... fra adresse i HL
LD	D,(IX+d)	DD56dd	... fra adresse IX+d
LD	D,(IY+d)	FD56dd	... fra adresse IY+d
LD	DE,nn	11nnnn	Loader DE med følgende bytes
LD	DE,(nn)	ED5Bnnnn	... med nnnn og nnnn+1
LD	E,A	5F	Loader E med Akk.
LD	E,B	58	... med B
LD	E,C	59	... med C
LD	E,D	5A	... med D
LD	E,E	5B	... med E
LD	E,H	5C	... med H
LD	E,L	5D	... med L
LD	E,n	1Enn	... med efterfølgende byte
LD	E,(HL)	5E	... fra adresse i HL
LD	E,(IX+d)	DD5Edd	... fra adresse IX+d
LD	E,(IY+d)	FD5Edd	... fra adresse IY+d
LD	H,A	67	Loader H med Akk.
LD	H,B	60	... med B
LD	H,C	61	... med C
LD	H,D	62	... med D
LD	H,E	63	... med E

LD	H,H	64	... med H
LD	H,L	65	... med L
LD	H,n	26nn	... med efterfølgende byte
LD	H,(HL)	66	... fra adresse i HL
LD	H,(IX+d)	DD66dd	... fra adresse IX+d
LD	H,(IY+d)	FD66dd	... fra adresse IY+d
LD	HL,nn	21nnnn	Loader HL med følgende bytes
LD	HL,(nn)	2Annnn	... fra nnnn og nnnn+1
LD	I,A	ED47	Loader I med Akk.
LD	IX,nn	DD21nnnn	Loader IX med følgende bytes
LD	IX,(nn)	DD2Annnn	... fra nnnn og nnnn+1
LD	IY,nn	FD21nnnn	Loader IY med følgende bytes
LD	IY,(nn)	FD2Annnn	... fra nnnn og nnnn+1
LD	L,A	6F	Loader L med følgende bytes
LD	L,B	68	... med B
LD	L,C	69	... med C
LD	L,D	6A	... med D
LD	L,E	6B	... med E
LD	L,H	6C	... med H
LD	L,L	6D	... med L
LD	L,n	2Enn	... med efterfølgende byte
LD	L,(HL)	6E	... fra adresse i HL
LD	L,(IX+d)	DD6Edd	... fra adresse IX+d
LD	L,(IY+d)	FD6Edd	... fra adresse IY+d
LD	R,A	ED4F	Loader R med Akk.
LD	SP,HL	F9	Loader SP med HL
LD	SP,IX	DDF9	... med IX
LD	SP,IY	FDF9	... med IY
LD	SP,nn	31nnnn	... med følgende bytes
LD	SP,(nn)	ED7Bnnnn	... fra nnnn og nnnn+1
LD	(BC),A	02	Loader adresse fra BC med Akk.
LD	(DE),A	12	Loader adresse fra DE med Akk.
LD	(HL),A	77	Loader adresse fra HL med Akk.
LD	(HL),B	70	... med B
LD	(HL),C	71	... med C
LD	(HL),D	72	... med D
LD	(HL),E	73	... med E
LD	(HL),H	74	... med H
LD	(HL),L	75	... med L
LD	(HL),n	36nn	... med efterfølgende byte

LD	(IX+d),A	DD77dd		Loader adresse IX+d med Akk.
LD	(IX+d),B	DD70dd		... med B
LD	(IX+d),C	DD71dd		... med C
LD	(IX+d),D	DD72dd		... med D
LD	(IX+d),E	DD73dd		... med E
LD	(IX+d),H	DD74dd		... med H
LD	(IX+d),L	DD75dd		... med L
LD	(IX+d),n	DD36ddnn		... med efterfølgende byte
LD	(IY+d),A	FD77dd		Loader adresse IY+d med Akk.
LD	(IY+d),B	FD70dd		... med B
LD	(IY+d),C	FD71dd		... med C
LD	(IY+d),D	FD72dd		... med D
LD	(IY+d),E	FD73dd		... med E
LD	(IY+d),H	FD74dd		... med H
LD	(IY+d),L	FD75dd		... med L
LD	(IY+d),n	FD36ddnn		... med efterfølgende byte
LD	(nn),A	32nnnn		Loader adresse nnnn med Akk.
LD	(nn),BC	ED43nnnn		... med C og nnnn+1 med B
LD	(nn),DE	ED53nnnn		... med E og nnnn+1 med D
LD	(nn),HL	22nnnn		... med L og nnnn+1 med H
LD	(nn),IX	DD22nnnn		... og nnnn+1 med IX
LD	(nn),IY	FD22nnnn		... og nnnn+1 med IY
LD	(nn),SP	ED73nnnn		... og nnnn+1 med SP
LDD		EDA8	P	Load og decrementer (formindsk med 1)
LDDR		EDB8	P=0	Load blok og decrementer
LDI		EDA0	P	Load og incrementer
LDIR		EDB0	P=0	Load blok og incrementer
NEG		ED44	S Z P C	Negater Akk.
NOP		00		WAIT (Nuloperation)
OR	A	B7	S Z P C	OR Akk. med Akk.
OR	B	B0	S Z P C	... med B
OR	C	B1	S Z P C	... med C
OR	D	B2	S Z P C	... med D
OR	E	B3	S Z P C	... med E
OR	H	B4	S Z P C	... med H
OR	L	B5	S Z P C	... med L
OR	n	F6nn	S Z P C	... efterføgende byte
OR	(HL)	B6	S Z P C	... med adresse i HL
OR	(IX+d)	DDB6dd	S Z P C	... med adresse IX+d
OR	(IY+d)	FDB6dd	S Z P C	... med adresse IY+d
OTDR		EDBB	S Z P	Udlæs blok og formindsk

OTIR		EDB3	S Z P	Udlæse blok og forhøj
OUT	(C),A	ED79		Udlæs Akk. via Port C
OUT	(C),B	ED41		... B
OUT	(C),C	ED49		... C
OUT	(C),D	ED51		... D
OUT	(C),E	ED59		... E
OUT	(C),H	ED61		... H
OUT	(C),L	ED69		... L
OUT	(n),A	D3nn		Udlæs Akk. via Port n
OUTD		EDAB	S Z P	Udlæs og formindsk
OUTI		EDA3	S Z P	Udlæs og forhøj
POP	AF	F1		Hent Akk. og flag fra STACK
POP	BC	C1		Hent BC fra STACK
POP	DE	D1		Hent DE fra STACK
POP	HL	E1		Hent HL fra STACK
POP	IX	DDE1		Hent IX fra STACK
POP	IY	FDE1		Hent IY fra STACK
PUSH	AF	F5		Læg Akk. & Flags i STACK
PUSH	BC	C5		Læg BC i STACK
PUSH	DE	D5		Læg DE i STACK
PUSH	HL	E5		Læg HL i STACK
PUSH	IX	DDE5		Læg IX i STACK
PUSH	IY	FDE5		Læg IY i STACK
RES	0,A	CB87		Slet Bit 0 i Akk.
RES	0,B	CB80		Slet Bit 0 i B
RES	0,C	CB81		... C
RES	0,D	CB82		... D
RES	0,E	CB83		... E
RES	0,H	CB84		... H
RES	0,L	CB85		... L
RES	0,(HL)	CB86		... adressen i HL
RES	0,(IX+d)	DDCBdd86		... adresse IX+d
RES	0,(IY+d)	FDCBdd86		... adresse IY+d
RES	1,A	CB8F		Sletter Bit 1 i Akk.
RES	1,B	CB88		Sletter Bit 1 i B
RES	1,C	CB89		... C
RES	1,D	CB8A		... D
RES	1,E	CB8B		... E
RES	1,H	CB8C		... H
RES	1,L	CB8D		... L
RES	1,(HL)	CB8E		... adressen i HL
RES	1,(IX+d)	DDCBdd8E		... adresse IX+d
RES	1,(IY+d)	FDCBdd8E		... adresse IY+d
RES	2,A	CB97		Sletter Bit 2 i Akk.

RES	2,B	CB90	Sletter Bit 2 i B
RES	2,C	CB91	... C
RES	2,D	CB92	... D
RES	2,E	CB93	... E
RES	2,H	CB94	... H
RES	2,L	CB95	... L
RES	2,(HL)	CB96	... adressen i HL
RES	2,(IX+d)	DDCBdd96	... adresse IX+d
RES	2,(IY+d)	FDCBdd96	... adresse IY+d
RES	3,A	CB9F	Sletter Bit 3 i Akk.
RES	3,B	CB98	Sletter Bit 3 i B
RES	3,C	CB99	... C
RES	3,D	CB9A	... D
RES	3,E	CB9B	... E
RES	3,H	CB9C	... H
RES	3,L	CB9D	... L
RES	3,(HL)	CB9E	... den første adresse i HL
RES	3,(IX+d)	DDCBdd9E	... adresse IX+d
RES	3,(IY+d)	FDCBdd9E	... adresse IY+d
RES	4,A	CBA7	Sletter Bit 4 i Akk.
RES	4,B	CBA0	Sletter Bit 4 i B
RES	4,C	CBA1	... C
RES	4,D	CBA2	... D
RES	4,E	CBA3	... E
RES	4,H	CBA4	... H
RES	4,L	CBA5	... L
RES	4,(HL)	CBA6	... adressen i HL
RES	4,(IX+d)	DDCBddA6	... adresse IX+d
RES	4,(IY+d)	FDCBddA6	... adresse IY+d
RES	5,A	CBAF	Sletter Bit 5 i Akk.
RES	5,B	CBA8	Sletter Bit 5 i B
RES	5,C	CBA9	... C
RES	5,D	CBAA	... D
RES	5,E	CBAB	... E
RES	5,H	CBAC	... H
RES	5,L	CBAD	... L
RES	5,(HL)	CBAE	... adressen i HL
RES	5,(IX+d)	DDCBddAE	... adresse IX+d
RES	5,(IY+d)	FDCBddAE	... adresse IY+d
RES	6,A	CBB7	Sletter Bit 6 i Akk.
RES	6,B	CBB0	Sletter Bit 6 i B
RES	6,C	CBB1	... C
RES	6,D	CBB2	... D
RES	6,E	CBB3	... E

RES	6,H	CBB4		... H
RES	6,L	CBB5		... L
RES	6,(HL)	CBB6		... adresse i HL
RES	6,(IX+d)	DDCBddB6		... adresse IX+d
RES	6,(IY+d)	FDCBddB6		... adresse IY+d
RES	7,A	CBBF		Sletter Bit 7 i Akk.
RES	7,B	CBB8		Sletter Bit 7 i B
RES	7,C	CBB9		... C
RES	7,D	CBBA		... D
RES	7,E	CBBB		... E
RES	7,H	CBBC		... H
RES	7,L	CBBD		... L
RES	7,(HL)	CBBE		... adressen i HL
RES	7,(IX+d)	DDCBddBE		... adresse IX+d
RES	7,(IY+d)	FDCBddBE		... adresse IY+d
RET		C9		Returner fra underrutine.
RET	C	D8		Returner, hvis C=1
RET	M	F8		... hvis S=1
RET	NC	D0		... hvis C=0
RET	NZ	C0		... hvis Z=0
RET	P	F0		... hvis S=0
RET	PE	E8		... hvis P=1
RET	PO	E0		... hvis P=0
RET	Z	C8		... hvis Z=1
RETI		ED4D		Returner fra interrupt
RETN		ED45		... fra NMI-Routine
RL	A	CB17	S Z P C	Venstrerotation af Carry & Akk.
RL	B	CB10	S Z P C	... B
RL	C	CB11	S Z P C	... C
RL	D	CB12	S Z P C	... D
RL	E	CB13	S Z P C	... E
RL	H	CB14	S Z P C	... H
RL	L	CB15	S Z P C	... L
RL	(HL)	CB16	S Z P C	... og adresse i HL
RL	(IX+d)	DDCBdd16	S Z P C	... adresse IX+d
RL	(IY+d)	FDCBdd16	S Z P C	... adresse IY+d
RLA		17	C	... og Akk.
RLC	A	CB07	S Z P C	Venstrerotation af Akk.
RLC	B	CB00	S Z P C	... B
RLC	C	CB01	S Z P C	... C
RLC	D	CB02	S Z P C	... D
RLC	E	CB03	S Z P C	... E
RLC	H	CB04	S Z P C	... H

RLC	L	CB05	S Z P C	... L
RLC	(HL)	CB06	S Z P C	... adresseb i HL
RLC	(IX+d)	DDCBdd06	S Z P C	... adresse IX+d
RLC	(IY+d)	FDCBdd06	S Z P C	... adresse IY+d
RLCA		07	C	... i Akk.
RLD		ED6F	S Z P	Decimalrotation mod venstre.
RR	A	CB1F	S Z P C	Højrerotation af Carry & A
RR	B	CB18	S Z P C	... B
RR	C	CB19	S Z P C	... C
RR	D	CB1A	S Z P C	... D
RR	E	CB1B	S Z P C	... E
RR	H	CB1C	S Z P C	... H
RR	L	CB1D	S Z P C	... L
RR	(HL)	CB1E	S Z P C	... og adresse i HL
RR	(IX+d)	DDCBdd1E	S Z P C	... adresse IX+d
RR	(IY+d)	FDCBdd1E	S Z P C	... adresse IY+d
RRA		1F	C	... og Akk.
RRC	A	CB0F	S Z P C	Højrerotation af Akk.
RRC	B	CB08	S Z P C	... B
RRC	C	CB09	S Z P C	... C
RRC	D	CB0A	S Z P C	... D
RRC	E	CB0B	S Z P C	... E
RRC	H	CB0C	S Z P C	... H
RRC	L	CB0D	S Z P C	... L
RRC	(HL)	CB0E	S Z P C	... adressen i HL
RRC	(IX+d)	DDCBdd0E	S Z P C	... adresse IX+d
RRC	(IY+d)	FDCBdd0E	S Z P C	... adresse IY+d
RRCA		0F	C	... Akk.
RRD		ED67	S Z P	Decimalrotation mod højre
RST	00	C7		Kald af underrutine ved 00
RST	08	CF		... ved 08
RST	10	D7		... ved 10
RST	18	DF		... ved 18
RST	20	E7		... ved 20
RST	28	EF		... ved 28
RST	30	F7		... ved 30
RST	38	FF		... ved 38
SBC	A,A	9F	S Z P C	Subtraher Carry & Akk. fra Akk.
SBC	A,B	98	S Z P C	... B fra Akk.
SBC	A,C	99	S Z P C	... C fra Akk.
SBC	A,D	9A	S Z P C	... D fra Akk.
SBC	A,E	9B	S Z P C	... E fra Akk.
SBC	A,H	9C	S Z P C	... H fra Akk.

SBC	A,L	9D	S Z P C	... L fra Akk.
SBC	A,n	DEnn	S Z P C	... følgende bytes fra Akk.
SBC	A,(HL)	9E	S Z P C	... adresse i HL fra A
SBC	A,(IX+d)	DD9Edd	S Z P C	... adresse IX+d fra Akk.
SBC	A,(IY+d)	FD9Edd	S Z P C	... adresse IY+d fra Akk.
SBC	HL,BC	ED42	S Z P C	Subtraher C & BC fra HL
SBC	HL,DE	ED52	S Z P C	... DE fra HL
SBC	HL,HL	ED62	S Z P C	... HL fra HL
SBC	HL,SP	ED72	S Z P C	... SP fra HL
SCF		37	C=1	Sæt Carry-Bit til 1
SET	0,A	CBC7		Sæt Bit 0 i Akk.
SET	0,B	CBC0		Sæt Bit 0 fra B
SET	0,C	CBC1		... fra C
SET	0,D	CBC2		... fra D
SET	0,E	CBC3		... fra E
SET	0,H	CBC4		... fra H
SET	0,L	CBC5		... fra L
SET	0,(HL)	CBC6		... med adresse i HL
SET	0,(IX+d)	DDCBddC6		... med adresse i (IX+d)
SET	0,(IY+d)	FDCBddC6		... med adresse i (IY+d)
SET	1,A	CBCF		Sæt Bit 1 i Akk.
SET	1,B	CBC8		Sæt Bit 1 fra B
SET	1,C	CBC9		... fra C
SET	1,D	CBCA		... fra D
SET	1,E	CBCB		... fra E
SET	1,H	CBCC		... fra H
SET	1,L	CBCD		... fra L
SET	1,(HL)	CBCE		... med adresse i HL
SET	1,(IX+d)	DDCBddCE		... med adresse i (IX+d)
SET	1,(IY+d)	FDCBddCE		... med adresse i (IY+d)
SET	2,A	CBD7		Sæt Bit 2 i Akk.
SET	2,B	CBD0		Sæt Bit 2 fra B
SET	2,C	CBD1		... fra C
SET	2,D	CBD2		... fra D
SET	2,E	CBD3		... fra E
SET	2,H	CBD4		... fra H
SET	2,L	CBD5		... fra L
SET	2,(HL)	CBD6		... med adresse i HL
SET	2,(IX+d)	DDCBddD6		... med adresse i IX+d
SET	2,(IY+d)	FDCBddD6		... med adresse i IY+d
SET	3,A	CBDF		Sæt Bit 3 i Akk.
SET	3,B	CBD8		Sæt Bit 3 fra B
SET	3,C	CBD9		... fra C
SET	3,D	CBDA		... fra D

SET 3,E	CBDB	... fra E
SET 3,H	CBDC	... fra H
SET 3,L	CBDD	... fra L
SET 3,(HL)	CBDE	... med adresse i HL
SET 3,(IX+d)	DDCBddDE	... med adresse i (IX+d)
SET 3,(IY+d)	FDCBddDE	... med adresse i (IY+d)
SET 4,A	CBE7	Sæt Bit 4 i Akk.
SET 4,B	CBE0	Sæt Bit 4 fra B
SET 4,C	CBE1	... fra C
SET 4,D	CBE2	... fra D
SET 4,E	CBE3	... fra E
SET 4,H	CBE4	... fra H
SET 4,L	CBE5	... fra L
SET 4,(HL)	CBE6	... med adresse i HL
SET 4,(IX+d)	DDCBddE6	... med adresse i (IX+d)
SET 4,(IY+d)	FDCBddE6	... med adresse i (IY+d)
SET 5,A	CBEF	Sæt Bit 5 i Akk.
SET 5,B	CBE8	Sæt Bit 5 fra B
SET 5,C	CBE9	... fra C
SET 5,D	CBEA	... fra D
SET 5,E	CBEB	... fra E
SET 5,H	CBEC	... fra H
SET 5,L	CBED	... fra L
SET 5,(HL)	CBEE	... med adresse i HL
SET 5,(IX+d)	DDCBddEE	... med adresse i (IX+d)
SET 5,(IY+d)	FDCBddEE	... med adresse i (IY+d)
SET 6,A	CBF7	Sæt Bit 6 i Akk.
SET 6,B	CBF0	Sæt Bit 6 fra B
SET 6,C	CBF1	... fra C
SET 6,D	CBF2	... fra D
SET 6,E	CBF3	... fra E
SET 6,H	CBF4	... fra H
SET 6,L	CBF5	... fra L
SET 6,(HL)	CBF6	... med adresse i HL
SET 6,(IX+d)	DDCBddF6	... med adresse i (IX+d)
SET 6,(IY+d)	FDCBddF&	... med adresse i (IY+d)
SET 7,A	CBFF	Sæt Bit 7 i Akk.
SET 7,B	CBF8	Sæt Bit 7 fra B
SET 7,C	CBF9	... fra C
SET 7,D	CBFA	... fra D
SET 7,E	CBFB	... fra E
SET 7,H	CBFC	... fra H
SET 7,L	CBFD	... fra L
SET 7,(HL)	CBFE	... med adresse i HL

SET	7,(IX+d)	DDCBddFE		... med adresse i (IX+d)
SET	7,(IY+d)	FDCBddFE		... med adresse i (IY+d)
SLA	A	CB27	S Z P C	Skriv Carry & Akk. til venstre
SLA	B	CB20	S Z P C	... og B venstre
SLA	C	CB21	S Z P C	... og C venstre
SLA	D	CB22	S Z P C	... og D venstre
SLA	E	CB23	S Z P C	... og E venstre
SLA	H	CB24	S Z P C	... og H venstre
SLA	L	CB25	S Z P C	... og L venstre
SLA	(HL)	CB26	S Z P C	... & adresse i HL venstre
SLA	(IX+d)	DDCBdd26	S Z P C	... & adresse i IX+d venstre
SLA	(IY+d)	FDCBdd26	S Z P C	... & adresse i IY+d venstre
SRA	A	CB2F	S Z P C	... og A højre
SRA	B	CB28	S Z P C	... og B højre
SRA	C	CB29	S Z P C	... og C højre
SRA	D	CB2A	S Z P C	... og D højre
SRA	E	CB2B	S Z P C	... og E højre
SRA	H	CB2C	S Z P C	... og H højre
SRA	L	CB2D	S Z P C	... og L højre
SRA	(HL)	CB2E	S Z P C	... & adresse i HL højre
SRA	(IX+d)	DDCBdd2E	S Z P C	... & adresse i IX+d højre
SRA	(IY+d)	FDCBdd2E	S Z P C	... & adresse i IY+d højre
SRL	A	CB3F	S Z P C	... og Akk. højre
SRL	B	CB38	S Z P C	... og B højre
SRL	C	CB39	S Z P C	... og C højre
SRL	D	CB3A	S Z P C	... og D højre
SRL	E	CB3B	S Z P C	... og E højre
SRL	H	CB3C	S Z P C	... og H højre
SRL	L	CB3D	S Z P C	... og L højre
SRL	(HL)	CB3E	S Z P C	... & adresse HL højre
SRL	(IX+d)	DDCBdd3E	S Z P C	... & adresse (IX+d) højre
SRL	(IY+d)	FDCBdd3E	S Z P C	... & adresse (IY+d) højre
SUB	A	97	S Z P C	Subtraher Akk. fra Akk.
SUB	B	90	S Z P C	... B fra Akk.
SUB	C	91	S Z P C	... C fra Akk.
SUB	D	92	S Z P C	... D fra Akk.
SUB	E	93	S Z P C	... E fra Akk.
SUB	H	94	S Z P C	... H fra Akk.
SUB	L	95	S Z P C	... L fra Akk.
SUB	n	D6nn	S Z P C	... næstfølgende byte
SUB	(HL)	96	S Z P C	... adresse i HL
SUB	(IX+d)	DD95dd	S Z P C	... adresse i (IX+d) fra Akk.
SUB	(IY+d)	FD96dd	S Z P C	... adresse i (IY+d) fra Akk.
XOR	A	AF	S Z P C=0	Eksklusiv eller OR A

XOR	B	A8	S Z P C=0	... og B og A
XOR	C	A9	S Z P C=0	... og C
XOR	D	AA	S Z P C=0	... og D
XOR	E	AB	S Z P C=0	... og E
XOR	H	AC	S Z P C=0	... og H
XOR	L	AD	S Z P C=0	... og L
XOR	n	EEnn	S Z P C=0	... med efterfølgende byte
XOR	(HL)	AE	S Z P C=0	... & Adresse Hl
XOR	(IX+d)	DDAEdd	S Z P C=0	... & Adresse IX+d
XOR	(IY+d)	FDAEdd	S Z P C=0	... & Adresse IY+d

TILLÆG

HUKOMMELSENS BELÆGNING

De følgende sider indeholder adresser på kendte funktioner i hukommelsen. Der kan have indsneget sig enkelte fejl, derfor skal man udvise forsigtighed ved PEEK og POKE. Man skal dog ikke være bange for at eksperimentere lidt; husk at gemme eventuelle programmer på bånd eller diskette inden der POKES!

0000	-3FFF	OPERATIVSYSTEM
0000	-003F	KOPI AF ROM FOR BANKSWITCHING
0040	-013F	INPUT-BUFFER OG ARBEJDSOMRÅDE
0170	-AB7F	BASIC-HUKOMMELSE
3800	-3FFF	KARAKTERGENERATOR I ROM
AB80	-ABFF	KARAKTERGENRATOR FOR SELVDEFINEREDE TEGN
AC00		SLETNING AF FLAG FOR MELLEMRUM
AC01	-AC03	UDVIDELSESHOP FOR READY-MODE
AC04	-AC06	UDVIDELSESHOP FOR ERROR-HANDLING
AC07	-AC09	UDVIDELSESHOP FOR UDFØRELSE AF KOMMANDO
AC0A	-AC0C	UDVIDELSESHOP FOR FUNKTIONSBEREGNING
AC0D	-AC0F	UDVIDELSESHOP FOR HENT KONSTANTER
AC10	-AC12	UDVIDELSESHOP FOR INPUT AF BASICLINIER
AC13	-AC15	UDVIDELSESHOP FOR LIST
AC16	-AC18	UDVIDELSESHOP FOR TALKONVERTERING
AC19	-AC1B	UDVIDELSESHOP FOR OPERATORER
AC1C		FLAG FOR AUTO
AC1D	-AC1E	AUTO-LINIENUMMERERING
AC1F	-AC20	STEPVÆRDI FOR AUTO
AC21		SIDSTE STREAMNUMMER
AC22		INPUTFIL
AC23		PRINTERHOVEDES POSITION
AC24		WIDTH
AC25		SIDSTE POSITION PÅ BÅND
AC26		FLAG FOR FOR-NEXT
AC27	-AC28	MELLEMLAGRING AF FOR
AC2C	-AC2D	ADRESSE FOR NEXT
AC2E	-AC2F	ADRESSE FOR WEND
AC34	-AC35	ADRESSE FOR ON-BREAK
AC38	-AC43	NODE-KØ 0
AC44	-AC4F	NODE-KØ [illegible]
AC80	-AC91	NODE-KØ 2

AC92 - ACA3	NODE-KØ 3
ACA4 - ADA3	INPUT-BUFFER
ADA6 - ADA7	ERROR-ADRESSER
ADA8 - ADA9	BASIC-PROGRAM-POINTER EFTER ERROR
ADAA	ERROR-NUMMER
ADAB - ADAC	BASIC-PROGRAM-POINTER EFTER INTERRUPT
ADAD - ADAE	ADRESSE FOR AFBRUDT LINIEUDFØRELSE
ADAF - ADB0	ADRESSE FOR ON-ERROR
ADB1	FLAG FOR ON-ERROR AKTIV
ADB2	FIL-STATUS
ADB3	ENT
ADB4	ENV
ADB5 - ADB6	PERIODE
ADB7	STØJ-PERIODE
ADB8	VOLUMEN
ADB9 - ADBA	LÆNGDE
ADBB - ADBC	ENV OG ENT
ADCB - ADCF	MELLEMLAGRING FOR FLYDENDE KOMMA BEREGNINGER
ADD0 - AE03	TABEL FOR SKALAR-VARIABLER
AE04 - AE05	FN-TABEL
AE06 - AE0B	ARRAY-TABEL
AE0C - AE25	FORDEFINEREDE VARIABELTYPER A - Z
AE2D	SKILLETEGN FOR INPUT
AE2E - AE2F	ADRESSE FOR READ
AE30 - AE31	ADRESSE FOR DATA
AE32 - AE33	HUKOMMELSE FOR POINTER I BASIC-STACK
AE34 - AE35	ADRESSE FOR AKTUEL KOMMANDO
AE36 - AE37	ADRESSE FOR AKTUEL PROGRAMLINIE
AE38	FLAG FOR TRACE
AE3F - AE40	STARTADRESSE FOR LOAD-KOMMANDO
AE41	FLAG FOR CHAIN/MERGE
AE42	FILTYPE
AE43 - AE44	FILLÆNGDE
AE45	FLAG FOR PROGRAMBESKYTTELSE
AE46 - AE78	ARBEJDSOMRÅDE FOR ASCII-KONVERTERING
AE72 - AE73	CALL-ADRESSE
AE74	BANK-SWICTH MODE FOR CALL
AE75 - AE76	HL-REGISTER FOR CALL
AE77 - AE78	SP-REGISTER FOR CALL
AE79	TABULATOR-AFSTAND
AE7B - AE7C	HIMEM-POINTER
AE7D - AE7E	POINTER PÅ SLUTNING AF FRI RAM
AE7F - AE80	POINTER : START PÅ FRI RAM

AE81	-AE82	POINTER : BASIC-PROGRAM START
AE83	-AE84	POINTER : PROGRAM-SLUTNING
AE85	-AE86	POINTER : START PÅ VARIABLER
AE87	-AE88	POINTER : START PÅ ARRAY
AE89	-AE8A	POINTER : SLUT PÅ ARRAY
AE8B	-B08A	BASIC STACK (FOR, GOSUB...)
B08B	-B08C	BASIC STACK-POINTER
B08D	-B08E	POINTER PÅ STRENG-START
B08F	-B090	POINTER PÅ STRENG-SLUT
B09A	-B09B	POINTER PÅ STRENG-STACK
B0BA	-B0BC	STRENG-ANGIVELSE
B0C1		VARIABELTYPE
B0C2	-B0C3	DIVERSE ADRESSER
B100	-B1AB	ARBEJDSOMRÅDE FOR STYRING AF OPERATIVSYSTEM
B1C8		LØBENDE SKÆRM-MODE
B1CA		SKÆRM-OFFSET
B1CB		SKÆRMADRESSE
B1CC	-B1D6	DIVERSE
B1D7	-B1D8	BLINK-TID
B1D9	-B20B	DIVERSE REGISTER FOR BLINK-FARVE
B20C		AKTUELT VINDUE
B20D	-B276	PARAMETER FOR VINDUE
B285	-B286	CURSOR-POSITION (LINIE,KOLONNE)
B287	-B28B	DIVERSE REGISTRE FOR VINDUER
B28C	-B327	DIVERSE REGISTRE FOR SKÆRM
B328	-B329	ORIGIN-X
B321	-B32B	ORIGIN-Y
B32C	-B4DD	DIVERSE REGISTRE FOR GRAFIK
B4DE	-B550	DIVERSE REGISTRE FOR TASTATUR-SCAN
B551	-B7FF	DIVERSE REGISTRE FOR SOUND
B800	-B8DC	DIVERSE REGISTRE FOR KASSETTE
B807	-B816	NAVN PÅ INPUT-FIL
B84C	-B85B	NAVN PÅ OUTPUT-FIL
B8D1		SKRIVE-HASTIGHED
B8DD		INSERT-FLAG
B8E4	-B8E7	RANDOM
B8E8	-B8F6	MELLEMLAGER FOR FLYDENDE KOMMA BEREGNINGER
B8F7		FLAG FOR DEG / RAD
BF00	-BFFF	PROCESSOR-STACK
C000	-FFFF	VIDEO-RAM
C000	-FFFF	INTERPRETER-ROM (BASIC-FORTOLKER)
C000	-FFFF	UDVIDELSES-ROM

ORDLISTE (ikke komplet)

A

Acces

Den tid, der går fra data kaldes og indtil de er tilgængelige. Kaldes også tilgangstid.

Adresse

Talkode, der betegner et bestemt sted i hukommelsen.

Algoritme

En fuldstændig beskrevet fremgangsmåde til løsning af et givet program.

Arbejdslager

Det område af hukommelsen, der bruges til lagring og bearbejdning af data. Kaldes også RAM-området.

ASCII

En amerikansk standard for udveksling af bogstaver og symboler. ASCII er en kodetabel, hvori alle bogstaver, tal og specialtegn har fået en talkode. Denne talkode er international. Oprindelig udviklet til brug for telex-kommunikation.

B

BASIC

Beginners All-purpose Symbolic Instruction Code. Sikkert det mest udbredte og mest anvendte programmeringssprog på hjemmmecomputere og minicomputere (PC-ere).

BCD-format

Tallene 0-9 noteret i total-system. Hvert tal udgøres af 4 cifre (bits).

Bloksøgning

En gruppe af data søges.

Bloktransfer

En gruppe af data overføres.

Brugerinterface

Kobling mellem bruger og datamat.

Buffer

Mellemlager i en datamat.

Bug (lus)

Programfejl.

C

Cartridge

Et modul, der indeholder en ROM-kreds hvori et program er lagret. De fleste hjemmecomputere er forsynet med en såkaldt port, hvori modulet (cartridge) forbindes med computeren. Fordelen ved at bruge en cartridge som lagermedium i stedet for bånd eller diskette, er den ultrakorte tid, det tager at udføre et program fra ROM-kredsen i modulet til computerens arbejdslager. Det er især spil og alternative programmeringssprog, der forhandles i cartridge-form.

Compiler

Er betegnelsen for en "oversætter". Et højniveausprog som BASIC skal før det kan "køre" omdannes til maskinsprog som computeren kan forstå. I normal BASIC sker dette løbende efterhånden som programmet afvikles. Denne form for oversættelse kaldes fortolkning, men i stedet for at anvende denne relativt langsomme metode, kan man oversætte sit program een gang for alle. Det er det, man skal bruge en compiler til.

Cursor

Den lille markør på skærmen, der viser hvor "man" befinder sig. Markøren kan have flere former, og kan enten være blinkende eller fast. Cursoren svarer til spidsen af en blyant.

D

Database

Indholdet i alle filer.

Datatransmission

Data sendes ad elektronisk vej fra et sted til et andet.

Diskette

En rund magnetiserbar polyesterskive indkapslet i et plasthylster. Ydre lagringsmedie for en computer. Kapaciteten kan variere fra under 100 Kbytes til over 1 Mbytes.

F

Fil

En ordnet samling af sammenhørende data. I daglig tale refereres til den fysiske tilstedeværelse af et EDB-program/dataoplysninger på en diskette eller et bånd.

Formattering

Den proces, der inddeler en diskette i et antal spor og sektorer for skrivning og læsning af data.

H

Hexadecimal

Talsystem, der har 16 som grundtal.

I

Implementere

At udføre det arbejde der kræves for at gøre teori til praksis.

Indexerede rækker

Rækker der har fået index.

Initialisere

At tildele en variabel en værdi på forhånd.

Inversere

Omvende.

K

Kompatible

To datamater siges at være kompatible, hvis de kan afvikle det samme program uden ændringer, kommunikere med hinanden eller bruge de samme data.

Konfiguration

Betegnelse for indholdet af de enkelte enheder i et samlet computer-anlæg. Enhederne omfatter både hukommelse, interfaces og tilslutningen af printere, diskettestationer, monitorer.

Krympe

Formindske.

Konverteres

Omvende eller omforme.

L

Label

Identifikation, mærke eller etiket.

Load

Kommando der bevirker at et bestemt program indlæses fra baggrundslager.

Løkke

Programdel, hvis instruktioner skal gentages, indtil en bestemt betingelse er opfyldt.

M

Markør

Se under CURSOR.

Matrix

Rektangulær opstilling af elementer i række inden for matematikken.

O

Operand

Data, som kan behandles ved hjælp af en operator.

Overflow (stakoverløb)

Datamatens kapacitet (lager) er opbrugt.

P

Parametre

En variabel, hvis værdi er fast i en given anvendelse.

Pixel

Et enkelt element i et billede/tegn.

Port

Dataindgang til centralenheden (CPU). Bruges ofte som betegnelse for alle ud- og indgange på en computer.

Printer

En slags elektrisk skrivemaskine uden tastatur. Computeren kan aktivere printerens skrivehoved, således at de enkelte bogstaver og tegn kan udskrives på papir.

Processor

Del af central enheden i en datamat.

Pseude-akkumulator

Kunstig-akkumulator.

R

Rutiner

Program eller dele af et program, der anvendes almindeligt eller hyppigt.

S

Scrollning

Rulning (skærmbilledet).

Sekventielt

Data umiddelbart efter hinanden i en fil.

Simulering

Efterligning af virkelige forhold ved hjælp af en model.

Sound-chippen

Lyd-chippen.

Streng-variabler

Navnet på en streng (følge af bogstaver, tal og/eller tegn).

String (alfanumeriske data)

Følge af tal, tegn eller bogstaver.

Syntax

Regler for dannelse af tilladelige konstruktioner i et sprog.

V

Valid tegn

Lovligt tegn.

Vektor

Numerisk variabel med en dimension.

3-D-grafik

Tre-dimentionalt grafik.

Ovenstående ordliste er, som angivet i overskriften, ikke komplet, men indholdet er et skønsomt antal af de fremmedord, som optræder i bogen. Skulle der, efter din opfattelse være glemt nogle der burde have været med, hører vi gerne herom SKRIFTLIGT.

BUDGET MANAGER AMSTRAD

Med BUDGET MANAGER behøver De ikke længere tænke på forvaltningen af Deres husholdningskasse. BUDGET MANAGER kontrollerer alle udgifter og indtægter, kredit- og opsparingskonti på øre. Alle data kan læses på skærm eller udskrives på printer.

Da BUDGET MANAGER arbejder terminsorienteret kan alle data udskrives til forfaldsdato. Naturligvis råder BUDGET MANAGER over mange nyttige kalenderfunktioner, som hjælper dem i overvågningen af de vigtige terminer.

Endvidere er der funktioner til beregning af lånetilbud og renteberegning.

BUDGET MANAGER i stikord

- ***Overvågning af indtægt, udgift, kredit, opsparing og variable omkostninger.***
- ***Kalenderfunktioner.***
- ***Terminsovervågning.***
- ***Bedømmelse af lånetilbud.***
- ***Automatisk aktuallisering af konti.***

Kun kr. 498,-

DATAMAT
AMSTRAD

DATAMAT er et komfortabelt dataforvaltningsprogram til AMSTRAD.
DATAMAT kan indeholde op til 512 tegn pr. »kort« og bruger ethvert felt som index (søge) felt. De kan søge, sortere og udvælge data efter alle kriterier og på alle felter, efter Deres ønske.
DATAMAT kan overføre data til TEXTOMAT således at disse bruges til f.eks. personlige serie-breve m.v.
DATAMAT er extrem hurtig da den er skrevet i 100% maskinkode.

DATAMAT i stikord

- ***Fuld menustyrring giver en hurtig og nemmere betjening.***
- ***Fri definerbar indgangsmaske.***
- ***512 tegn pr. datablok, max. 50 felter pr. blok.***
- ***Arbejder sammen med TEXTOMAT.***
- ***Arbejder sammen med 1 eller 2 diskettestationer.***
- ***Udprintning af lister og labels i fri format.***
- ***Data kan sorteres og selekteres.***
- ***Printer kan også tilsluttes gennem RS 232 porten.***

Kun kr. 498,-

PROFIMAT

PROFIMAT er en pakke med 2 programmer. Til den ene monitor »Profi-Mon«, til den anden assembler »Profi-Ass«. »Profi-Ass« er en assembler, som kan assemble de maskinkodeprogrammer som CPC's hukommelse kan rumme. »Profi-Ass« tillader brugen af de såkaldte »makros« ligesom de professionelle assemblere. Symboltabeller kan trykfremstilles, også i alfabetisk rækkefølge, derved bliver brugen af »makros« talt op. Med »Profi-Ass« kan udarbejdes assembler-lister i valgfri format, som kan fremstilles med BASIC-editor. Sammenkædning af flere assembler-kartoteker er mulig. Symboltabeller kan indkodes.

PROFIMAT i stikord

- ***Profi-Ass, den professionelle assembler til Z80.***
- ***Profi-Mon, en maskinkodemonitor med disassembler.***
- ***Valgfri format.***
- ***Udarbejdelse af komplet assembler-liste.***
- ***Makros er mulig.***
- ***Omfangsrig pseudo-opcodes.***
- ***Sammenkædning af kildeprogrammer.***
- ***Objekt-kode går på disketten eller direkte i hukommelsen.***

Kun kr. 498,-

REGNEARKET KALKUMAT

KALKUMAT klarer tabelkalkulationsproblemer af enhver art. Formelindkodning gør det muligt at foretage hurtige beregninger med store talmængder. Det arbejdsområde som KALKUMAT administrerer har et omfang af 255 linier og 63 spalter, det vil sige, at der teoretisk står ca. 16000 felter til rådighed, der i reel arbejdsindsats kun begrænses af lagerkapaciteten. KALKUMAT tillader brugeren at omsætte de udregnede værdier til grafik. KALKUMATEN giver således et hurtigt og nemt overblik over de udregnede værdier. Både de med KALKUMAT frembragte resultater og fremstillede grafiske figurer kan opbevares på diskette eller overføres til papir via printer. Alle slags resultater kan, for oversigtens skyld, forsynes med skriftlige kommentarer.

KALKUMAT i stikord

- *Tabel kalkulationsprogram til CPC 464, 664 og 6128.*
- *Fuldt menustyret for hurtigt arbejde.*
- *255 linier og 63 spalter giver ca. 16.000 indkodningsfelter.*
- *Fremstilling af kage-, bjælke- samt minimum- og maximumgrafik.*
- *Både lydmæssig og optisk alarm ved fejlindkodning.*
- *Arbejdstema kan struktureres med farver.*
- *Såvel arbejdstema som grafik kan der skrives på.*
- *Interface til TEXTOMAT og DATAMAT.*

Kun kr. 498,-

EN DATA BECKER BOG

DULLIN · RETZLAFF
SCHNEIDER · STRASSENBURG

AMSTRAD 464/664 & 6128 Tips & Tricks

En sand guldgrube for brugere af 464/664 & 6128

DANSK/NORSK udgave

Forlag: , box 105, DK-6950 Ringkøbing

Kr. 248,-

BØGER PÅ TYSK:

Den store

JOYCE

bog er kommet

Kr. 298,-

JOYCE

for begyndere

Kr. 149,-

Bogen om

Z80
processoren

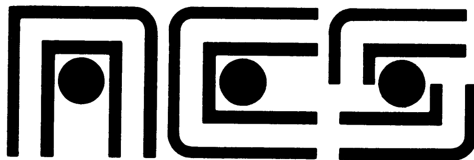
Kr. 298,-

KATALOG

med illustrationer
og korte indholdsforklaringer på 100 bøger og
programmer fås hos din

FORHANDLER

eller send kr. 5,00 i frimærker og få det direkte fra



NORDIC COMPUTER SOFTWARE
POSTBOX 105 · DK 6950 RINGKØBING
DANMARK

ADMINISTRATIONS-SYSTEM TIL 6128

BUSYPACK

Det mest solgte administrative
program til
AMSTRAD BUSYPACK TOTALSYSTEM
med finansmodul, lager,
debitor, kreditor, fakturering.
FULDT integreret.

Vejl. udsalgspris kr. 2995,-
incl. moms

Spørg DIN forhandler.

TIL NOTATER:

AMSTRAD-bladet...

- dit brugerblad

Med alt det spændende, nye til en af de allerstærkeste computere på markedet.

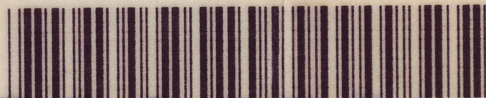
Amstrad-bladet er det eneste officielle brugerblad i Danmark.
Dette betyder at DU som bruger bliver holdt orienteret om alle de mange spændende projekter der sker omkring Amstrad computeren, både i Danmark og i udlandet, via direkte telexforbindelse til producenten i England samt vort samarbejde med den danske importør, Dinamicro.
Foruden nyheder og reportager bringer Amstrad-bladet også masser af tests (så du undgår at købe »katten i sækken«), oplysende artikler om programmering (så din investering måske kunne give en lille ekstraindtægt), 16 sider med programmer, som er lige til at taste ind, annoncer, tips og tricks, læser til læser – kontakt, oplysninger om brugerklubber og meget, meget mere.
Bladet er trykt i 4-farve offset og udkommer hver anden måned. Hvert nummer indeholder minimum 40 sider.
Amstrads computere har over hele verden vist sig at være virkelige 'top – chart – runner's når det gælder de internationale salgslister. Men selv en så stærk computer som Amstrad'en har brug for opbakning. Dinamicro og Twilights forhandlere er indstillede på at yde dig den bedst mulige hjælp vedr. dit køb og igangsætningen, resten af vejen kan Amstrad-bladet være en god og værdifuld kilde til oplysninger og interessante opgaver.

EN YDERLIGERE FORDEL FOR DIG:

Få vort nye blad «Input», fyldt med programlistninger til din Amstrad - Helt Gratis...

Vi vil gerne have lov at præsentere vort nye programblad for alle nye ejere af Amstrad computere. Derfor får du gratis det første nummer tilsendt, når du tegner abonnement, og så håber vi naturligvis, at du bliver SÅ begejstret at du henter **INPUT** hos din bladhandler, hver gang det udkommer

FORLAGET MICROTECH


117458997 Nordjyske Landsbibliotek

Gødvad Bakke 4 8600 Silkeborg 06 82 24 55