NASCOM Z80 NYT

NASCOM BRUGERGRUPPE
2730 Herlev

Sidevolden 23
Giro 6742602

NR: 1
3. årgang

Januar 1982

Jeg vil minde om fristen for indlevering af forslag til forårets generalforsamling alle relevante tider er angivet i vedtægterne.

Der har været nogen usikkerhed omkring leveringen af sidste nummer af Nascom Nyt, men jeg håber, at alle der har betalt for medlemsskab har modtaget sidste nummer. Vi har forsøgt at sende flere ud end måske nødvendigt, men det er vel undskyldeligt ? I samme forbindelse vil jeg minde om, at I husker at skrive afsenderadresse på girokortet, og at I kontrollerer at I har fået et medlemnummer ved næste udsendelse af Nascom Nyt. Medlemnumret står oven over jeres navn på kuverten. (Vi har et blankt girokort med datoen 29.12.81, hvem mon det er ??)

Venlig hilsen

si'r

ASBJØRN

INDHOLD

SIDE	2	SOFTWARE TIL KARAKTERGENERATOR
SIDE	9	PROGRAMMERING AF 2532 OG 2732
SIDE	11	STRUKTURERET PROGRAMMERING (1.del)
SIDE	17	FIRE SORTERINGSRUTINER
SIDE	20	INDHOLD AF 2. ARGANG
SIDE	21	KEYBOARD RUTINER
SIDE	23	NYE MEDLEMMMER

Software til Programmerbar Karaktergenerator

Af J. Haigh
fra "Micropower" nr.1 1981.

En karaktergenerator er normalt en hukommelsesenhed der gemmer data, som definerer hvilke punkter i en karakter, der er "on". I NASCOM'en består hver karakter af 16 linier x 8 prikker og dataene gemmes som 16 efter hinanden følgende bytes. Når VDU kredsløbet viser en bestemt karakter, servdet på dataene som er gemt i generatoren på den adresse som er sat op. Denne byte bliver sendt til et kredsløb, som ser på hvert enkelt bit i rækkefølge. Hvis det er en "1'er" bliver strålen på videodisplayet intensiveret, og en prik fremkommer. På NASCOM 2 bliver dataene fra karaktergeneratoren vist med det mindst betydende bit på højre side af karakteren, mens det på min NASCOM 1 monterede system, som er en kombination af Steven Hope's programmerbare karaktergenerator og et gammelt Bits and P.C.'s garfikkort, viser bit'ene i modsat rækkefølge.
I den efterfølgende softwarebeskrivelse vil jeg holde mig til NASCOM 2 display systemet, men fortælle hvordan programmerne ændres til andre maskiner. En yderligere inkompabilitet skyldes det, at NASCOM 2 kun viser 14 linier pr. karakter. Der har dog været vist en simpel modifikation i "Liverpool Software Gazette" som ændrer formatet til 16 linier.

Den simpleste måde at bruge en programmerbar karaktergenerator på, er at gemme karaktersæt på bånd og load'e hvert sæt efter som det skal bruges. - Men dette bliver hurtigt uinteressant. Special karakterer kan inkluderes i programmer som anvender dem og skrives ned i PKG'ens RAM enten med en kopieringsrutine, for et maskinkodeprogram, eller med en READ, POKE løkke i BASIC. Der er dog lige to ting som kræver speciel interesse :
Karakteren AØH og punktgrafikken i CØH - FFH.
Zeap assembleren anvender AØH som End Of Line (EOL) - mærke. I en standard NASCOM 2 grafikROM er denne karakter magen til et space, og kan derfor ikke ses på skærmen; Hvis AØH ikke er blank, hvilket den ikke vil være hvis du lige har tændt eller lige har brugt specialkarakterer, vil Zeap'en få et ret utydeligt display. Et lignende problem forekommer ved anvendelse af Nas-debug, som anvender karakteren CØH som separator. Man kan selvfølgelig altid slette disse adresser med en modify kommando, men en bedre løsning er at have en kort rutine i EPROM, som sletter AØH og skriver standard grafikken ind fra CØH til FFH. Listningen til en sådan rutine findes længere omme i bladet. Den efterfølges af et endnu simplere program som skriver TRS 80's karaktersæt ind i PKG'en. Dette er ret anvendeligt hvis man prøver at indtaste et TRS 80 program, som anvender dennes grafik, på NASCOM'en. Hvis din maskine viser generatorens bits i omvendt orden skal linie 270 ændres til
LD A 15 og linie 320 til ADD A FØH;
Værdierne i linierne 540 og 580 ombyttes på samme måde.

På en NASCOM 1 består de øverste 4 prikker i punktgrafikken af 5 linier x 4 prikker, mens de nederste indeholder 6 linier. Da en umodificeret NASCOM 2 udelader de 2 bundlinier, bliver de i dette tilfælde på 4 x 4.

Du vil hurtigt finde ud af at det er morsommere at opfinde sine egne karakterer, enten til brug enkeltvis eller i blokke - der kan laves meget imponerende high-resolution billeder med 128 grafikkarakterer i en 16 x 8 blok. Besværet kommer når man skal til at finde ud af, hvilke data der skal lægges i PKG RAM'en. En metode er at tegne karaktererne på millimeterpapir og så konvertere dem til bytes. Det tredie program kan anvendes til at tegne karakterer direkte fra tastaturet. Programmet eksekveres ved at skrive : E 1ØØØ aaaa, hvor aaaa er adressen på din PKG RAM eller hvis din PKG adresse er sammenfaldende med et EPROM område som i Steven Hope's design, et område med fri RAM hvor en kopi af PKG dataene kan ligge. Karaktererne er vist på højre side af skærmen i stor skala, hvert bit repræsenteret af 2 grafikbrikker. Bit'ene tændes og slukkes af de 8 mest venstreliggende taster i tastaturets 4 rækker (1-8,Q-I,A-K og Z-,). En pil indikerer hvilket sæt på 4 linier som er valgt i øjeblikket; Tasterne O,P,; og / flytter pilen så alle 16 linier kan modificeres. Karakteren som defineres dukker op ved cursoren, som kan flyttes rundt på den venstre del af skærmen v.h.a. cursor tasterne. Denne karakter kan efterlades på enhver position ved at trykke på - tasten, og fjernes med =. Du kan på denne måde opbygge blokke af karakterer og derved tegne et komplekst billede. På bundlinien viser programmet Hex værdien af den definerede karakter og de 16 bytes med data i karakteren i Hex (til brug i maskinkodeprogrammer) og i decimal (til BASIC). Du kan steppe frem eller tilbage gennem karaktersættet med N/L og BS tasterne. Når du første gang vælger en karakter er den sikkert fyldt med uønskede data; disse kan slettes med CTRL + C.
Da programmet anvender punktgrafikken til at lave de forstørrede karakterer kan man kun definere 64 karakterer ad gangen; programmet vil derfor virke med et system som kun har 1k RAM og gemmer resten af karaktererne (blokgrafikken) i EPROM.
Modifikationerne for inverse display er :
1Ø55 ændres til ØFH, 1Ø5F ændres til FØH, 124A ændres til ØlH, 125Ø til Ø2 og 12A9,1283,128D,12D9 til ØE.
Hvis din PKG RAM er sammenhørende med en blok EPROM skal dens startadresse placeres i 12EC-12EDH.
Det fjerde program er inkluderet som et demonstrations program. Det viser et billede af rumfærgen som bevæger sig langsomt over skærmen. Det meste af programmet består af datatabeller som kopieresover i PKG RAM'en for at producere startbilledet. Dette bevæges så ved at rotere hver byte i grafikkarakterernе en bit ad gangen. Hvis du ønsker at konvertere dette program til inverse video skal adresse 135FH ændres til 16H og datatabellen skal inverteres. Jeg har inkluderet en rutine i 13CØH som udfører denne invertering.

Til sidst er der 2 korte programmer som demonstrerer punkt plotning fra BASIC. Det første plotter en satellits bane rundt om 2 legemer, et synligt og et usynligt. Det andet viser hvordan simple grafer kan produceres i BASIC. Begge rutiner sætter punkterne direkte fra BASIC og kan selvfølgelig gøres hurtigere ved anvendelse af maskinkoderutiner som kaldes med USR. Med en 128 karakterers PKG har du lige nok karakterer til at plotte en sinuskurve med akser i fuld skærmstørrelse. Hvis du forsøger at plotte mange kurver vil du se at der ikke er karakterer nok - til sådanne billeder skal du have bit-mapped grafik. Dog er det overraskende hvor effektive billeder man kan opnå med en simpel PKG; et LIFE program er meget imponerende på et 384 x 240 array. Nu da mange kommercielle grafikenheder er tilgængelige håber jeg at der vil blive lavet software som gør fuld anvendelse af high-resolution mulig.

-0- -0-

Jeg skal lige tilføje, at den omtalte ændring fra the Liverpool Software Gazette ikke har været til at opdrive for mig. Hvis nogen ligger med bladet, vil jeg være taknemmelig for en kopi.

Go' fornøjelse

Jesper Nielsen.

→

Hvis nogle medlemmer har et Gemini disksystem ned den oprindelige D-DOS kan de henvende sig til Jesper Skavin og bestille en DOS-udvider !!

Systemmet er beskrevet på engelsk på 6 A4 ark med tilhørende 9 ark maskinkode skrevet på ZEAP - assembler. Denne udvidede DOS tilbyder følgende muligheder: 1) USER INPUT, som modtager, dekoder og gemmer filnavne og andre parametre, hvorefter kontrollen overlades til en af de følgende rutiner, og som også tillader enkontrolleret udgang fra DOS'en. 2) FUNCTION 1, en rutine der viser alle filnavne og restplads på disken. 3) FUNCTION 2, en rutine der finder ønsket fil og placerer den i RAM på en adresse der findes på disken. 4) FUNCTION 3, en rutine der gemmer et program på disken med parametre, der kan vælges frit eller efter programart bliver sat. Opdaterer indeks på disken. 5) FUNCTION 4, sletter fil og forøger den ledige plads på disken. 6) FUNCTION 5, der giver en fil et nyt navn. 7) FUNCTION 6, en rutine der læser en navngivet fil til RAM og derefter hopper til forudsat adresse og udfører programmet herfra.

```
1000            0010          ORG  £1000    ; Nascom 2 pixel set
1000 210000     0020          LD   HL 0     ; Put your P.C.G. address here
1003 110002     0030          LD   DE £200  ; Offset to character £A0
1006 19         0040          ADD  HL DE
1007 E5         0050          PUSH HL       ; Save HL
1008 0610       0060          LD   B 16     ; Sixteen bytes to be cleared
100A 3600       0070 CLRA0    LD   (HL) 0   ; Set all bytes to zero
100C 23         0080          INC  HL
100D 10FB       0090          DJNZ CLRA0
100F E1         0100          POP  HL       ; Recover HL
1010 19         0110          ADD  HL DE    ; Now go to character £C0
1011 0EC0       0120          LD   C £C0      ;Character is kept in C
1013 C5         0130 SBC      PUSH BC       ; Save BC
1014 D70B       0140          RCAL PIX5     ; Set 5 top bytes as necessa
1016 D709       0150          RCAL PIX5     ; Set next 5 bytes
1018 0606       0160          LD   B 6      ; Set bottom 6 bytes
101A D707       0170          RCAL PIXEL
101C C1         0180          POP  BC       ; Recover BC
101D 0C         0190          INC  C        ; Next character
101E 20F3       0200          JR   NZ SBC   ; Continue until zero
1020 C9         0210          RET
1021 0605       0220 PIX5     LD   B 5      ; Routine to set 5 bytes
1023 AF         0230 PIXEL    XOR  A        ; Clear A
1024 CB09       0240          RRC  C        ; Test bits 0,1 or 2
1026 C5         0250          PUSH BC
1027 3002       0260          JR   NC RRC3  ; Jump if bit = 0
1029 3EF0       0270          LD   A £F0    ; Draw pixel if bit = 1
102B CB09       0280 RRC3     RRC  C        ; Now test bits 3,4 or 5
102D CB09       0290          RRC  C
102F CB09       0300          RRC  C
1031 3002       0310          JR   NC LDHLA; Jump if bit = 0
1033 C60F       0320          ADD  A 15     ; Draw pixel if bit = 1
1035 77         0330 LDHLA    LD   (HL) A   ; Set number of lines
1036 23         0340          INC  HL       ; stored in B
1037 10FC       0350          DJNZ LDHLA
1039 C1         0360          POP  BC       ; Recover BC
103A C9         0370          RET
```

```
1000            0380          ORG  £1000    ; TRS80 pixel set
1000 0EC0       0390          LD   C £C0
1002 210000     0400          LD   HL 0     ; Put your P.C.G. address here
1005 C5         0410 TPUSH    PUSH BC
1006 D70B       0420          RCAL TRSP5    ; Two sets of 5 lines
1008 D709       0430          RCAL TRSP5
100A 0606       0440          LD   B 6
100C D707       0450          RCAL TRSPIX   ; One set of six lines
100E C1         0460          POP  BC
100F 0C         0470          INC  C
1010 20F3       0480          JR   NZ TPUSH
1012 C9         0490          RET
1013 0605       0500 TRSP5    LD   B 5
1015 AF         0510 TRSPIX   XOR  A
1016 CB09       0520          RRC  C        ; Test bits 0, 2, or 4
1018 3002       0530          JR   NC TRRC
101A 3EF0       0540          LD   A £F0    ; Draw pixel if bit = 1
101C CB09       0550 TRRC     RRC  C        ; Test bits 1, 3, or 5
101E C5         0560          PUSH BC
101F 3002       0570          JR   NC TRR2
1021 C60F       0580          ADD  A 15     ; Draw pixel if bit = 1
1023 77         0590 TRR2     LD   (HL) A   ; Set number of lines
1024 23         0600          INC  HL       ; stored in B
1025 10FC       0610          DJNZ TRR2
1027 C1         0620          POP  BC
1028 C9         0630          RET
```

```
T1000 1300 0 8 1
  1000 C3 83 10 31 32 33 34 35 36 37 38 51 57 45 52 54
  1010 59 55 49 41 53 44 46 47 48 4A 4B 5A 58 43 56 42
  1020 4E 4D 2C 74 65 64 20 11 10 27 CD 40 10 11 E8 03
  1030 CD 40 10 11 64 00 CD 40 10 1E 0A CD 40 10 1E 01
  1040 AF 3C ED 52 30 FB 3D 19 F6 30 F7 C9 06 05 AF CB
  1050 09 C5 30 02 3E F0 CB 09 CB 09 CB 09 30 02 F6 0F
  1060 77 FD 77 00 FD 23 23 10 F7 C1 C9 3A 23 10 26 00
  1070 CB BF 07 17 CB 14 17 CB 14 17 CB 14 6F ED 5B 25
  1080 10 19 C9 EF 0C 00 3A 0B 0C FE 02 28 32 EF 0D 59
  1090 6F 75 20 68 61 76 65 20 6E 6F 74 20 65 6E 74 65
  10A0 72 65 64 20 74 68 65 20 50 2E 43 2E 47 2E 20 52
  10B0 41 4D 20 61 64 64 72 65 73 73 2E 0D 00 DF 5B 2A
  10C0 0E 0C 22 25 10 3E 20 32 24 10 3E 80 32 23 10 DD
  10D0 21 00 00 FD 2A 25 10 11 00 04 FD 19 EB 0E 40 C5
  10E0 CD 4C 10 04 CD 4E 10 CD 4C 10 C1 0C F2 DF 10 EF
  10F0 0C 00 21 6A 08 36 E0 06 08 2C 36 E4 10 FB 2C 36
  1100 C4 06 05 11 40 00 19 36 C7 10 FB 19 36 C3 06 08
  1110 2D 36 D2 10 FB 2D 36 D8 06 05 ED 52 36 F8 10 FA
  1120 3E 0D 32 B4 08 3A 23 10 2A 29 0C 77 47 E5 21 6A
  1130 0A 22 29 0C EF 43 68 61 72 2E 20 00 78 DF 68 CD
  1140 6B 10 EB 21 CA 0A 22 29 0C 21 10 00 19 EB E5 DF
  1150 54 21 4A 0B 22 29 0C E1 06 04 CD 66 11 DF 6A 06
  1160 04 CD 66 11 18 27 EF 20 00 5E 23 56 23 E5 21 00
  1170 80 A7 ED 52 30 0A EF 2D 00 67 6F A7 ED 52 18 04
  1180 EB EF 20 00 CD 27 10 DF 69 E1 10 DD C9 E1 22 29
  1190 0C DF 7B 11 FF FF FE 11 28 15 11 01 00 FE 12 28
  11A0 0E 11 C0 FF FE 13 28 07 11 40 00 FE 14 20 34 2A
  11B0 29 0C 3A 24 10 77 19 7C FE 07 20 04 26 09 18 19
  11C0 FE 0A 20 04 26 08 18 11 7D E6 3F FE 2A 20 06 7D
  11D0 EE 20 6F 18 04 FE 09 28 F6 7E 32 24 10 3A 23 10
  11E0 77 18 AB FE 03 20 0D CD 6B 10 06 10 36 00 23 10
  11F0 FB C3 94 12 FE 30 20 23 DD 21 00 00 1E 00 16 00
  1200 21 B4 08 01 40 00 36 20 09 09 36 20 09 36 20 09
  1210 36 20 21 B4 08 19 36 0D C3 91 11 FE 50 20 08 DD
  1220 21 04 00 1E 80 18 D7 FE 3B 20 08 1E C0 DD 21 08
  1230 00 18 CB FE 2F 20 09 DD 21 0C 00 11 00 01 18 C0
  1240 21 03 10 0E 04 1E 00 06 08 16 80 BE 28 37 23 CB
  1250 0A 10 F8 1C 0D 20 F0 FE 2D 20 09 3A 23 10 32 24
  1260 10 C3 25 11 FE 3D 20 04 3E 20 18 F2 FE 08 20 0B
  1270 3A 23 10 3D CB FF 32 23 10 18 19 FE 0D 20 E2 3A
  1280 23 10 3C 18 EF D5 CD 6B 10 D1 7A 16 00 19 DD E5
  1290 D1 19 AE 77 CD 6B 10 E5 FD E1 21 AB 08 11 38 00
  12A0 0E 05 06 08 3E C0 FD CB 00 06 30 04 CB C7 CB DF
  12B0 FD CB 01 06 30 04 CB CF CB E7 FD CB 02 06 30 04
  12C0 CB D7 CB EF 77 23 10 DC FD 23 FD 23 FD 23 19 0D
  12D0 20 D0 06 08 3E C0 FD CB 00 06 30 04 CB C7 CB DF
  12E0 CB CF CB E7 77 23 10 EC 2A 25 10 11 00 00 01 00
  12F0 04 ED B0 C3 25 11 6E 20 74 68 61 74 20 63 68 61
```

```
T1000 13D8 0 8 1
  1000 C3 06 13 4C 44 01 03 00 03 03 03 03 03 03 03 78
  1010 7F FF FF FF 7F F8 FC 0E EF EF EF EF EF EF EF 0F
  1020 C0 FF FF FF FF 00 00 00 00 80 C0 E0 F0 F8 FC FE
  1030 FF 3F CF F7 F7 00 00 00 00 00 00 00 00 00 00 00
  1040 00 80 C0 E0 F8 00 00 00 00 00 00 00 00 00 00 00
  1050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  1060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  1070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  1080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  1090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  10A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  10B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  10C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  10D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  10E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  10F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  1100 00 00 00 00 00 FF FF FF 7F FF FF FF 7F 78 02 1E
  1110 7E 1E 00 1E 7E FF FF FF FF FF FF FC C3 3F FF FF
  1120 FF FF FF FF FF F8 FF FF FF FF FF FF FF FF FF FF
  1130 FF FF FF FF FF 00 FF FF FF FF FF FF FF FF FF FF
  1140 FF FF FF FF FF 00 FF FF FF FF FF DD CD D5 D5 D9
  1150 FF FF FF FF FF 00 FC FF FF FF FF F3 ED E1 ED ED
  1160 FF FF FF FF FF 00 00 FF FF FF FF F1 F7 F3 FD F1
  1170 FF FF FF FF FF 00 00 C0 FF FF FF F9 F6 F0 F6 F6
  1180 FF FF FF FF FF 00 00 00 FF FF FF FF FF FF FF FF
  1190 FF FF FF FF FF 00 00 00 FF FF FF FF FF FF FF FF
  11A0 FF FF FF FF FF 00 00 00 00 FF FF FF FF FF FF FF
  11B0 FF FF FF FF FF 00 00 00 00 FF FF FC FC FE FF FF
  11C0 FF FF FF FF FF 00 00 00 00 F0 FC 07 31 79 31 FF
  11D0 FF FF FF FF FF 00 00 00 00 00 00 00 00 00 80 F0
  11E0 FE FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00
  11F0 00 70 7C 7F 7F 00 00 00 00 00 00 00 00 00 00 00
  1200 00 00 00 00 00 1E 00 00 00 00 00 00 00 00 00 00
  1210 00 00 00 00 00 FF FF 00 00 00 00 00 00 00 00 00
  1220 00 00 00 00 00 FF FF 00 00 00 00 00 00 00 00 00
  1230 00 00 00 00 00 FF FF 00 00 00 00 00 00 00 01 03
  1240 07 0F 1F 3F 00 FF FF 00 03 07 0F 1F 3F 7F FF FF
  1250 FF FF FF FF 00 FF FF 55 FF FF FF FF FF FF FF FF
  1260 FF FF FF F0 00 FF D5 FF FF FF FF FF FF FF FF FF
  1270 FF FE E0 00 00 55 FF FF FF FF FF FF FF FF FF FE
  1280 E0 00 00 00 00 AA FF FF FF FF FF FF FF FF E0 00
  1290 00 00 00 00 00 AA FF FF FF FF FF FF FC 00 00 00
  12A0 00 00 00 00 00 54 FF FF FF FF FF E0 00 00 00 00
  12B0 00 00 00 00 00 28 FF FF FF F8 00 00 00 00 00 00
  12C0 00 00 00 00 00 7F 9F DF 80 00 00 00 00 00 00 00
  12D0 00 00 00 00 00 FF FF FF 00 00 00 00 00 00 00 00
  12E0 00 00 00 00 00 7F 7E 70 00 00 00 00 00 00 00 00
  12F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  1300 00 00 00 00 00 00 EF 0C 00 3A 0B 0C FE 02 28 0B
  1310 EF 41 72 67 2E 20 00 DF 6B DF 5B 21 0A 09 22 03
  1320 10 36 8F 11 40 00 19 36 9F 19 36 AF 19 36 BF 21
  1330 05 10 ED 5B 0E 0C 01 00 03 ED B0 EB 36 00 23 10
  1340 FB 2A 0E 0C 11 00 00 01 00 04 ED B0 DD 2E 08 DD
  1350 26 04 2A 0E 0C 11 10 00 0E 10 E5 06 10 A7 CB 1E
  1360 F5 FF 19 F1 10 F8 E1 23 0D 20 EF 11 F0 00 19 DD
  1370 25 20 E2 2A 0E 0C 11 00 00 01 00 04 ED B0 DD 2D
  1380 20 CD 21 38 09 11 40 00 E5 CD A2 13 E1 19 CB 4C
  1390 28 F6 ED 5B 0E 0C 21 10 00 19 01 00 03 ED B0 C3
  13A0 3B 13 D5 E5 D1 1C 01 2E 00 ED B8 7E 12 D1 E6 0F
  13B0 20 02 36 21 35 C9 20 61 6E 64 20 72 65 6D 6F 76
  13C0 21 05 10 11 05 13 06 08 CB 06 1F 10 FB 77 23 A7
  13D0 ED 52 19 38 F1 DF 5B 20 63 61 6E 20 74 68 75 73
```

```
LIST

1 REM: "ORBITS" BY S. HOPE.
5 LT=1:G=500:CLS:SC=3:CH=129
10 DIMS(LT),V(LT),A(LT),SN(LT),VN(LT),D(7),CH(2047)
20 FORI=0TOLT:INPUT"CO-ORDS FOR START (S,V)";S(I),V(I):NEXT
30 INPUT"GIVE 2ND MASS COORDS";M(0),M(1)
40 CLS:FORI=0TO7:READD(I):NEXT:DATA128,64,32,16,8,4,2,1
45 FORI=1TO40:X=48*RND(1):Y=15*RND(1):PRINT@X,Y,".";:NEXT
50 FORI=0TO31:DOKEI,0:NEXT:FORI=13TO17:READA:POKEI,A:CH(I)=A:NEXT
60 DATA28,62,62,62,28
70 PRINT@22,9,CHR$(128);@22,10,CHR$(129);
100 GOSUB1000:X=S(0)/SC:Y=S(1)/SC
102 J=2592+X/8-INT(Y/16)*64:IFPEEK(J)>100THENK=(PEEK(J)-128)*16:GOTO110
104 CH=CH+1:IFCH>255THENPRINT@0,0,"NO MORE CHARS":STOP
105 K=16*(CH-128):FORI=KTO14+KSTEP2:DOKEI,0:NEXT
107 IFPEEK(J)=46THENDOKEK+7,6168
108 POKEJ,CH:PRINT@42,0,255-CH;
110 A=KBD:IFA<>0THENTL=A-48
115 L=K+15-(YAND15):CH(L)=CH(L)ORD(XAND7):POKEL,CH(L):GOTO100
1000 R=0:R1=0:FORI=0TOLT:VN(I)=V(I)+A(I):SN(I)=S(I)+V(I)+A(I)/2
1002 X=(SN(I)+S(I))/2:R=X*X+R:Y=X-M(I):R1=R1+Y*Y
1003 NEXT:IFR<1ORR1<1THENPRINT@0,0,"BANG";:STOP
1010 A=-G/R/SQR(R):A1=-G/R1/SQR(R1)
1015 FORI=0TOLT:A(I)=A*(S(I)+SN(I))/2+A1*((S(I)+SN(I))/2-M(I))
1020 V(I)=(VN(I)+V(I)+A(I))/2:S(I)=(SN(I)+S(I)+V(I)+A(I)/2)/2:NEXT:RETURN
Ok



LIST

5 REM: A PROGRAM TO PLOT SIMPLE GRAPHS
10 CLEAR:DIMCH(2047),DT(7)
20 DATA 128,64,32,16,8,4,2,1
30 FOR N=0 TO 7:READ DT(N):NEXT
40 GOSUB 900
45 REM: THE SINE CURVE
50 FORX=0 TO 383:Y=120+100*SIN((X-191)/60)
60 GOSUB 1000:NEXT
70 FORT=1TO2000:NEXT
80 GOSUB900:FORX=0TO383:XS=(X-191.5)/30
85 REM: THE GRAPH OF SIN(X)/X
90 Y=120+100*SIN(XS)/XS
100 GOSUB 1000:NEXT
800 END
900 CLS:CR=127:FORX=0TO383:Y=120:GOSUB1000:NEXT
910 FORY=0TO239:X=191:GOSUB1000:NEXT:RETURN
1000 X1=INT(X/8):X2=X-8*X1
1005 Y1=INT(Y/16):Y2=Y-16*Y1
1010 M=2954+X1-64*Y1:C=(PEEK(M)-128)*16
1020 IF C>0 THEN 1060
1030 CR=CR+1:IFCR>255THEN RETURN
1040 C=16*(CR-128):POKEM,CR
1050 FORK=CTOC+15:POKEK,0:CH(K)=0:NEXT
1060 N=C+15-Y2:CH(N)=CH(N)ORDT(X2):POKEN,CH(N)
1070 RETURN
Ok
```

Programmering af 2532 og 2732 v.h.a. EPROM brænderen fra NN 5/81.

Af Ole Brandt.

Programmering af 2532 kan lade sig gøre alene v.h.a. software ændringer, men da et lille indgreb er nødvendigt for 2732, har jeg udnyttet dette til 2532 også. (Det giver også mindre softwareændring).

På selve brænderen har jeg monteret en 2mm bananbøsning til den sidste adressetællers (7493) ben 11 (QD). Denne bliver så = A11.

Til 2532 har jeg lavet en 'extendersokkel' hvor alle ben, undtagen ben 18, går direkte i programmerens sokkel. Ben 18 er forbundet til et lille stykke ledning der skal forbindes til A11 udtaget.

Til 2732 bruger jeg en anden 'extendersokkel'. Her er ben 18 og -2o byttet (se fig. 1), og ben 21 er forbundet til en ledning der skal forbindes til A11 udtaget.

Når dette er gjort, er der kun tilbage at anvende et nyt program. Dette program er 27o8/2716 programmet med følgende ændringer:

- Hvis omskifteren på brænderen står på 27o8 udskrives fejl og returneres til Nassys.
- Alle kommandoer der gennemløber alle adresser skal ændres til at gennemløbe 1oooH.
- Overskrift ændres til 2532/2732.
- 'PRGM' rutinen ændres, således at spændingerne bliver sat rigtigt op. (Her er den eneste forskel på 2532 og 2732 i programmet). For 2732 skal det være AND ØADH, og for 2532 skal det være AND ØF7H.

Jeg har valgt, blot at ændre det oprindelig program, sådan at jeg nu har ét program til 27o8/2716 og et andet til 2532/2732. 2532/2732 programmet kan kun køre i RAM, da omskiftningen mellem de to typer består i at ændre 2 bytes i programmet. (Overskrift 25/27 samt AD/F7 i 'PRGM' modulet - ikke særligt smart, men det var hurtigt, og det virker).

Sourceliste af programmet kan fås ved fremsendelse af frankeret svarkurvert hertil* . Denne liste indeholder også et par småændringer i forhold til Morten Kølbæk's oprindelige program. Det gælder især, at programmering af FF springes over ved 2716 (27o8 kan ikke tåle at man springer noget over), samt udlæsning af hver 16 adresse ved programmering i stedet for blot et punktum.

For de der står og skal have programmeret en '32er NU, har jeg vedlagt et hexdump af programmet. *) Box 45, 354o Lynge.

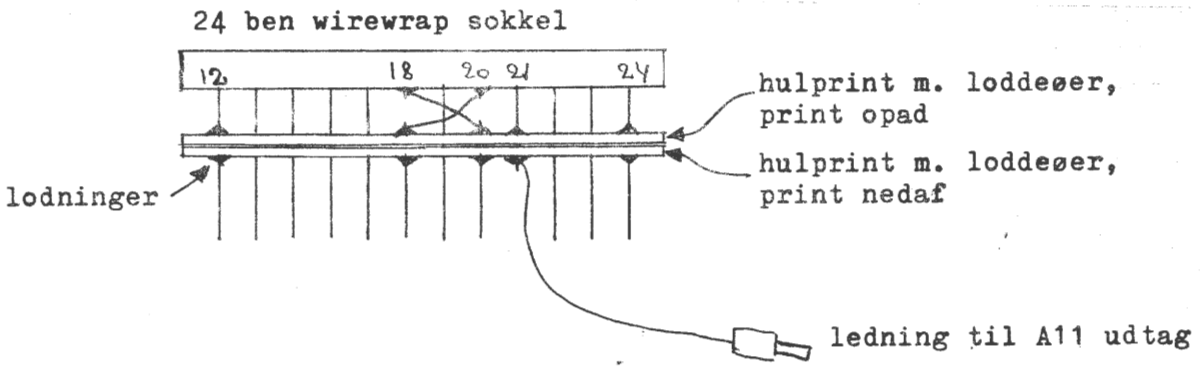


Fig. 1. 2732 extendersokkel til Eprom brænderen.

```
EPROM PROGRAMMER 2532/2732. OMB 811225.                    SIDE 01


T 2000 229A 0 8
  2000 EF 0C A0 11 D5 0B 21 10 20 01 1B 00 ED B0 18 1B   ▫...U.!. ...m0..
  2010 45 50 52 4F 4D 20 50 52 4F 47 52 41 4D 4D 45 52   EPROM PROGRAMMER
  2020 20 54 49 4C 20 32 20 4D 48 7A 2E 31 00 10 3E FF    TIL 2 MHz.1..>.
  2030 D3 06 D3 06 3E CF D3 07 3E 80 D3 07 3E FF D3 05   S.S.>OS.>.S.>.S.
  2040 CD 4C 22 21 32 27 28 1A EF 4F 4D 53 4B 2E 20 53   ML"!2'(.▫OMSK. S
  2050 54 41 41 52 20 54 49 4C 20 32 37 30 38 21 0D 00   TAAR TIL 2708!..
  2060 DF 5B EF 0D 45 50 52 4F 4D 20 54 59 50 45 20 00   ←[▫.EPROM TYPE .
  2070 DF 66 EF 0D 0D 50 20 3A 20 50 52 4F 47 52 41 4D   ←FD..P : PROGRAM
  2080 4D 45 52 20 45 50 52 4F 4D 0D 56 20 3A 20 56 45   MER EPROM.V : VE
  2090 52 49 46 49 43 45 52 20 45 50 52 4F 4D 0D 54 20   RIFICER EPROM.T
  20A0 3A 20 54 52 41 4E 53 46 45 52 20 45 50 52 4F 4D   : TRANSFER EPROM
  20B0 20 54 49 4C 20 52 41 4D 0D 4E 20 3A 20 4E 41 53    TIL RAM.N : NAS
  20C0 2D 53 59 53 20 52 45 54 55 52 4E 0D 4B 20 3A 20   -SYS RETURN.K :
  20D0 4B 4F 4E 54 52 4F 4C 20 41 46 20 49 4E 44 48 4F   KONTROL AF INDHO
  20E0 4C 44 20 28 52 3D 52 65 74 75 72 6E 29 0D 46 20   LD (R=Retur).F :
  20F0 20 46 46 20 49 20 52 41 4D 42 55 46 46 45 52 0D    FF I RAMBUFFER.
  2100 4D 20 3A 20 4D 4F 44 49 46 59 20 4D 4F 44 45 0D   M : MODIFY MODE.
  2110 32 37 33 32 2F 32 35 33 32 3A 20 32 30 34 35 20   2732/2532: 2045
  2120 32 37 2F 32 35 20 32 32 31 46 20 41 44 2F 46 37   27/25 221F AD/F7
  2130 2E 0D 00 DF 63 1A FE 41 38 24 FE 5B 30 20 FE 50   ...←c.~A8$~[0 ~P
  2140 CA C8 21 FE 56 28 37 FE 54 28 1A FE 4E 28 14 FE   JH!~V(7~T(.~N(.~
  2150 4B CA 51 22 FE 46 CA 6E 22 FE 4D CA 82 22 DF 6B   KJQ"~FJn"~MJ."←k
  2160 C3 2B 20 DF 5B 21 00 10 CD 06 22 16 10 0E 04 06   C+ ←[!..M.".....
  2170 00 ED A2 20 04 15 CA 2B 20 CD 11 22 18 F3 CD 06   .m" ..J+ M.".≤M.
  2180 22 21 00 10 01 00 10 00 DB 04 ED A1 C4 97 21 E2   "!.....[.m!D.!b
  2190 B7 21 CD 11 22 18 F1 F5 C5 F5 2B DF 66 7E DF 68   7!M.".▫u▫Eu+←F~←H
  21A0 DF 69 23 F1 DF 68 DF 69 DF 69 DF 62 30 06 CF FE   ←i#▫←H←i←i←b0.O~
  21B0 1B CA 2B 20 C1 F1 C9 EF 0D 0A 54 52 59 4B 20 54   .J+ A▫I▫..TRYK T
  21C0 41 53 54 00 CF C3 2B 20 DD 21 00 10 CD 1C 22 3E   AST.OC+ ]!..M.">
  21D0 FF D3 06 AF D3 06 21 00 10 DD E5 C1 CD 06 22 7E   .S./S.!..]eAM."~
  21E0 FE FF 28 05 D3 04 CD 23 22 CD 11 22 79 E6 0F 20   ~.(.S.M#"M."yF.
  21F0 0C E5 C5 01 00 F0 09 DF 66 DF 69 C1 E1 23 0B 78   .eE..p.←F←iAA#.x
  2200 B1 20 DC C3 2B 20 DB 05 E6 DF D3 05 F6 20 D3 05   1 \C+ [.F←S.v S.
  2210 C9 DB 05 E6 FE D3 05 F6 01 D3 05 C9 DB 05 E6 AD   I[.F~S.v.S.I[.F-
  2220 D3 05 C9 C5 DB 05 F6 02 D3 05 CD 44 22 DB 05 E6   S.IE[.v.S.MD"[.F
  2230 FD D3 05 C1 C9 C5 06 21 10 FE C1 C9 06 03 CD 35   }S.AIE.!.~AI..M5
  2240 22 10 FB C9 06 C8 CD 35 22 10 FB C9 DB 05 E6 80   ".<I.HM5".<I[.F.
  2250 C9 CD 06 22 18 0A CD 11 22 DF 7B FE 52 CA 68 22   IM."..M."←{~RJh"
  2260 DB 04 DF 68 DF 69 18 EE CD 06 22 C3 2B 20 21 00   [.←H←i.nM."C+ !.
  2270 10 16 10 06 00 3E FF 77 23 05 20 FB 15 20 F8 C3   .....>.w#. ←. xC
  2280 2B 20 13 DF 79 DF 4D C3 2B 20 00 00 00 00 00 00   + .←y←MC+ ......
```

Ved 4 MHz ændres ADR 2025H til 34 samt ADR 2237H til 40H (med wait) eller 42H (uden wait).

Lidt om STRUKTUREret PROGRAMMERING

Det er vist efterhånden rygtedes, at der findes et programmeringssprog, der hedder "struktureret", og at det er noget for sig. Den sidste påstand er god nok, der er bare ikke tale om et computersprog, men om en metode, der hjælper (og tvinger) os til at skrive vel tilrettelagte programmer.

Først et par ord om det, der gør, at de fleste af vores programmer ender med at være u-strukturerede:

ASSEMBLER (og udpræget Z80's) giver adgang til et snedigt roderi af memory og en endnu mere fascinerende samling funktioner. MEMORY: RAM, registre, dobbelte registre, alternative registre, B-tæller, BC-tæller, index registre, stakken og C-portpointer. Og FUNKTIONER: 8/16 bit aritmetik, måske biner, 2'-complement eller decimal, byte- eller bitlogik samt ikke at forglemme en variation af flytninger, der står mål memorystrukturen.

BASIC skulle som højniveausprog hjælpe på forståelsen, det hele foregår på engelsk. - Bortset fra, at man selvfølgelig altid skal forkorte korrekt, og husk at bruge paranteser om parametre, bare ikke altid.

Det uheldigste er dog (programmeringsmæssigt set), at der optræder et helt uvedkommende element: linienumre. Man kan rette linienumre, henvise til dem og notere dem , men forstå dem ? - umuligt. Gode programmer skal kunne forstås. Først, fordi man kan se at de virker, senere fordi man kan ændre og opfylde nye specificationer.

BASIC kan gøres struktureret og næsten læseligt, men det sker ved, at man tager sig i nakken med en grundig planlægning, der senere afspejles via lidt REMarks.

Hvad er det, der gør galt når vi arbejder med NASCOM'men? For ex. har man brug for 3 adressepointere, det passer lige: BC, DE og HL. Desuden en tæller - nå, BC pushes indtil videre, BC er et naturligt valg til netop en tæller; men desværre kan pointeren BC ikke poppes uden at ødelægge noget andet.

Altså: vælg registre, når programmet er færdigt, til den tid har man et temmelig godt overblik over pladsbehovet.

Herefter det mere positive: nogle af de programmeringsværktøjer, der er brug for ved den egentlige programskrivning, d. v. s. ved planlægningen:

1. PROBLEMOPSTILLING / beskrivelse
2. STRUKTUR - diagrammer
3. MODULER, f. ex. til et bibliotek
4. WARNIER - ORR - diagrammer
5. STATE - diagrammer
6. FLOW CHART (som bør undgås !)

Dette skema dækker en ordentlig mundfuld, som ikke kan uddybes tilstrækkeligt på denne plads, men en begyndelse er et forsøg værd. Først en oversigt for dem, der ikke er er helt dus med udtrykkene ovenfor.

FLOW CHART's var det almindeligeste værktøj i mange år. Det hedder på dansk meget malende "flydediagrammer", og det viser sig, at ved komplekse programmer kommer alting til at flyde. Diagrammet kommer til at indeholde meget lange hop, hvert enkelt problem løses ved at følge en linie, der zig-zag'ger sig på på kryds og tværs. Jeg har set diagrammer på ca. en kvadratmeter, ret uhandy at sidde med, når der skal findes fejl. Denne type programmer kaldes også "spagetti": find et, tegn alle hop som strøger, så ligner det.

MODULARISERING er det første skridt på på vejen til bedre programmer. Der skal være meget få forbindelser mellem modulerne, som

derfor kan behandles hver for sig. Meddelelser mellem modulerne trækker et nyt behov med sig: en DATAstruktur. Denne er i moderne programskrivning uhyre vigtig, for alle de uoverskuelige hop er erstattet af "logiske data", som styrer det job, modulerne udfører.

Modulariseringen bliver for større programmer gennemført i flere lag, der opstår et HIERAKI: et overordnet modul anvender underordnede moduler. Der er opfundet flere måder at anskueliggøre sådanne hierakier, hvoraf WARNIER-ORR diagrammet vist er det kendteste. Det giver på een gang jobudførelse (ovenfra og ned) og hieraki (fra venstre mod højre). W-O kendes på lang afstand på, at man tegner store venstre tuborg-paranteser. De er svære at lave med en skrivemaskine - trist, at man ikke kan få alle fordele på een gang.

Tilbage til flow chart, som er delt op i moduler. Det er fristende at finde nogle få helt enkle bidder, der kunne danne basis for f. eks. moduler. Det blev til bare fem typer, som fik navnet STRUKTURER. Strukturerne kan formuleres med en skrivemaskine. De ligner i så fald et forenklet PASCAL og går under betegnelsen PSEUDOKODE. Strukturerne kan også tegnes som flow chart, så vil man kunne nyde, at flow chartet for en gangs skyld er til at forstå.

STATE - diagrammet er egentlig blot een af disse strukturer, men det har nogle særlige fordele. Det er en grafisk fremstilling, som ofte benyttes direkte til brugsanvisning for brugeren. Når vi "er i BASIC" eller "er i monitoren" er det to NASCOM states, som vi kan skifte mellem med simple ordrer.

....

Bogen MICROCOMPUTERE / PROGRAMMERINGSTEKNIK af David Higgins (ved Henning Mejer) giver en grundig forklaring af WARNIER - ORR programmering. Også begyndere kan følge med et nyttigt stykke vej. (teknisk forlag)

::

: 1. Prøv først at lave en beskrivelse :

::

Et sted skal man begynde, det kan være ved tastaturet, men det kan lige så godt være alle andre steder, papir og blyant (og måske et lille viskelæder) er nok.

En beskrivelse på dette tidspunkt vil givetvis blive mangelfuld, men skidt være med det. Vil du lave et spil, så find et sæt spilleregler, skriv nogenlunde dansk med lidt forsøg på disposition: indledning, generelle regler, undtagelser og en værdig slutning.

Tegn skitser af skærmbilleder, eller tegn diagrammer af ydre "isenkram", hvis det er et styringsprogram du vil starte på.

```
###     Et eksempel
#
# COMPARE..DE..og..HL  sådan,  at flagene bliver
# justeret fornuftigt,  men uden at registrene
# af den grund bliver ændrede.
#
```

Tænk lidt over hvad dette er for noget vrøvl. Alligevel er det en meget normal specifikation, fordi den opgave der almindeligvis skal løses er en del mere kringlet, og man kan ikke forlange et overblik endnu. Senere vil jeg vende tilbage efter at have brugt mit viskelæder.

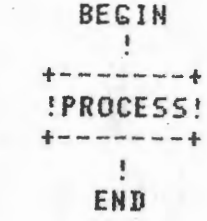
NB: "mangelfuld" er ment bogstaveligt: det, der er skrevet, skal tages alvorligt, er pr. definition altså korrekt. Derimod kan der forventes uddybninger af alt det, som vi mennesker anser for selvfølgeligheder, men som en computer aldrig har lært. Heller ikke eksemplet indeholder noget, der er forkert, der mangler bare lidt fastere holdepunkter, som kan tilføjes; de SKAL endda tilføjes inden programmet skrives.

::

2. PROGRAMSTRUKTURER

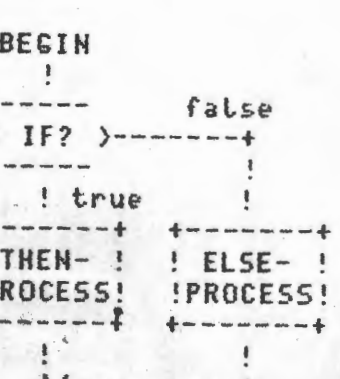
Den vigtigste fordel ved strukturer er, at der er så få typer:

PROCEDURE



Det er da enkelt, det er faktisk for enkelt til at det er nok. Men bemærk, at der kun er en indgang og en udgang på strukturen. I praksis må man altså ikke hoppe ind i processen og bruge sidste halvdel. Enten skulle programmet være delt i to procedurer, eller du bliver nødt til at kopiere den sidste halvdel som en ny procedure.

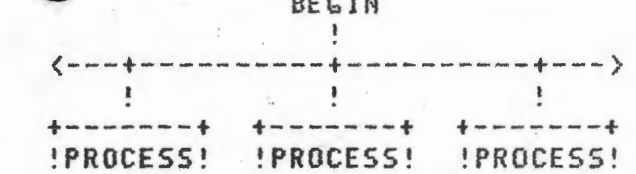
IF - THEN - ELSE



Når denne struktur er tilladt er det muligt at vælge , ikke alene mellem to muligheder, men flere, ELSE-processen kan være en ny IF - THEN. Tre-fire " nestede " strukturer er temmelig normalt, skønt der findes en udvidet valgstruktur, CASE.
MORALE skal ikke glemmes: 1 indgang, 1 udgang

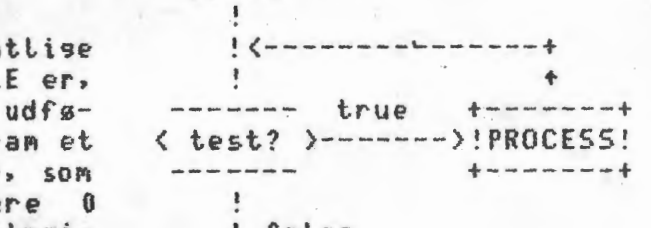
Ordforklaring: "nestede" betyder, at en ramme (her PROCESS) indeholder endnu en struktur, et kinesisk - æske - system.

CASE



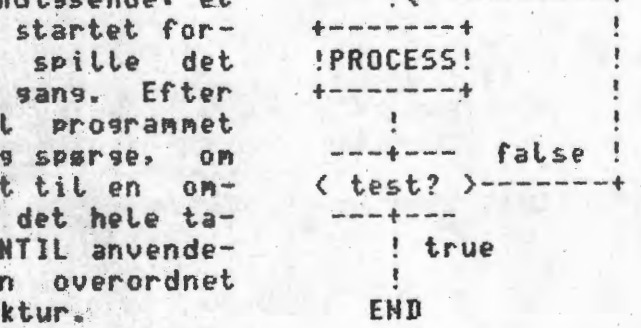
Så kan vi igen bruge tabelstyrede programmer. Og det er den sidste af de strukturer, der kun går fremad. For også at få lidt tilknytning til det endelige program: en CASE i BASIC hedder ON xx GOTO nn,mm...; men MORALE: nn, mm og alle de andre skal ende med GOTO ssss, strukturslutadressen, hvad ellers?

DO WHILE



Det væsentlige ved DO WHILE er, at man kan udføre et program et antal gange, som også kan være 0 gange. Den logiske test sker inden "process" eventuelt udføres.

DO UNTIL



Det er da indlysende: et spil bliver startet fordi man vil spille det mindst een gang. Efter spillet skal programmet selvfølgelig spørge, om man har lyst til en omgang til. I det hele taget er DO UNTIL anvendelig som en overordnet programstruktur.
Noter lige igen forskel-

lighederne på de to sløjfestrukturer :
DO WHILE udføres muligvis slet ikke og afsluttes, hvis testresultat = false.
DO UNTIL udføres altid een gang og afsluttes, hvis testresultat = true.

PASCAL - fans har på dette tidspunkt en fornemmelse af, at de hører til de frelste, for PASCAL er et struktueret sprog. Bortset fra DO UNTIL , som kaldes REPEAT UNTIL er alle strukturerne med i PASCAL.

I BLS PASCAL er der en mulighed for at hoppe til en label, men det er aldrig nødvendigt. Det er en slags urent trav, det kan man se af, at funktionen f. ex. er udeladt i visse andre mini-PASCALS, brug REPEAT UNTIL.

Normalt afstedkommer højniveau-oversættelser en kode (maskinsprog), som kunne være kortere og hurtigere hvis den var skrevet direkte i maskinskode. Det er imidlertid min erfaring, at et assemblerprogram bliver mere kompakt, når det skrives med et struktueret forløg, end hvis man starter på lurenkig og fortsætter med at lave en subrutine, når to bidder program er ens.

Det skyldes, at håndoversættelsen fra struktureret pseudokode til assembler er mindre rodet end lurenkig, mens man stadig har mulighed for at udnytte registre og instruktioner mere effektivt end computercompilering ville gøre.

Lad mig på dette planlægningstidspunkt lige nævne, at det er klogt, sideløbende at lave en liste over variable og konstanter inclusive en umisforståelig navngivning. Ikke fordi det hører til i selve pseudokoden, men fordi denne oversigt alligevel skal laves, når pseudokoden skal oversættes til et eller andet sprog.
::


```
::::::::::::::::::::::::::::::::::::::::::::::::::::;
  3.  MODULER  til et programbibliotek              ;
::::::::::::::::::::::::::::::::::::::::::::::::::::;
```

Nu kan jeg demonstrere med nogle eksempler, hvordan strukturerne kan anvendes i praksis. Samtidig er det forhåbentligt inspiration til at samle på små (bittesmå), gode, sikre, veldokumenterede programmer. Læg ved samme lejlighed mærke til, at ind-data og ud-data optræder foran hvert program, en enkel datastruktur. For større programkomplekser starter man faktisk med at udvikle ind-datastrukturen og ud-datastrukturen inden man begynder at programstrukturere.

```
###     Beskrivelser
#
# CLEAR..ACC   skal sætte accumulatoren => 0
# og justere flagene Cy og Z. I øvrigt så
# få sideeffekter som muligt.
#
# CLEAR..CARRY   skal sætte Cy - flaget => 0,
# helst ikke mere.
#
# COMPARE..HL..DE  bevarer indholdet af ordregi-
# strene DE og HL,  men  Cy og Z  flagene skal
# sættes som ved subtraktionen HL - DE.
# (NB:  nu ved vi,  hvad der trækkes fra hvad)
#
#       Strukturerne bliver: een "PROCESS"
#
#       Modulerne bliver:
#.....................................................
#
# CLEAR..ACCUMULATOR
# ud:   Acc = 0 (byte)
#       Cy=> 0, Z => 1, P => 1 (flag)
#       S => 0, N => 0, H => 0 (flag)
        ORG   00
CLRACC: XOR   A
#.....................................................
::
```

```
#.....................................................................
#
# CLEAR..CARRY
# ud:   Cy=> 0, N => 0, H => 0 (flag)
#       Z, P og S justeres jfr. Acc. (flag)
        ORG   00
CLRCY:  OR    A
#.....................................................................
#
# COMPARE..HL..DE
# ind:  DE og HL : unsigned 16-bit ord
# ud:   flag justeres som ved HL - DE
        ORG   00
CPHLDE: OR    A             ; fjern Cy
        SBC   HL,DE         ; justerer flag
        ADD   HL,DE         ; reparer  HL, kun Cy
#                           ;   "ændres", dog til
#                           ;   samme værdi !
#.....................................................................
```

Sådan som vi laver programmer vil det være fornuftigt at dele sin samling moduler i rutiner og subrutiner. Hvis vi havde rådighed over macroassemblere, betinget linkning og andre finesser ville man kun samle på rutiner. Med kun een udgang på hvert modul er det nemt at sætte et RET efter en rutine.

Det var en forklaring på nødvendigheden af "kun een udgang", en anden er, at modulerne gerne skulle kunne indbygges i en (hierakisk) overordnet struktur.

Eksemplerne ovenfor er på bare 1, 1 og 4 bytes. Det giver faktisk en tvivlsom fordel at bruge selv CPHLDE som subrutine-call. Er rutinen brugt 4 gange bliver forbruget :

som rutiner (macros)..... 4*4=16 bytes
som subrut. (calls)..... 4*3+4+1=17 bytes

Med een relative call (RCAL) står det lige, bortset fra en elendig timing.

Et BASIC-program kan få kommentarer, der forsøger at vise en struktur. Læg mærke til, at jeg har undgået kommentaren

```
4050 SK(I)=I: REM kort SK(I) sættes lig med I
```

- det eneste "oplysende" er ordet "kort", der er en datatypeoplysning, som hører hjemme i en samlet dataoversigt. Netop BASIC er meget sårbar, når man ikke har skrevet sine variable ned samlet. Man kommer nemt til at ændre en værdi, som en anden del af programmet har gemt til senere brug ("I" i eksemplet får en værdi 52, som næppe kan bruges til noget).

```
#.....................................................................
#
# DAN..SORTERET..KORTBUNKE  på  eet  spil kort.
#
4000 REM >> DAN..SORTERET..KORTBUNKE  SK
4010 REM hvert kort har et nr. 0 ... 51
4020 REM ind: DIM SK(51) (hovedprogr. init.)
4030 REM ud:  SK(n) = n ,     I = 52
4040 FOR I=0 TO 51 STEP 1   : REM DO ..
4050 SK(I) = I
4060 NEXT I :   REM I=I+1   : REM .. UNTIL I>51
4070 RETURN
#.....................................................................
```

Strukturen findes altså, men oversættelsen fra pseudokode kan kun genfindes, hvis man får hjælp fra REMarks. Hvis man har problemer med lagerplads, er det ikke nogen dårlig ide blot at arkivere sine pseudokoder efter at have noteret indgangslinienummeret ved hver rutines begyndelse. Kombinationen af noter og program er fantastisk, hvis man ikke har snydt og skrevet lidt frihåndsBASIC.

Rutinens linienummer 4000 kan rutinen måske få lov til at beholde i det fremtidige program, hvis hvert modul får sit nummer.

Til sidst en lidt mere normal rutine. Mit private mål for, at en rutine ikke er for stor, er, at den alt inclusive kan rummes let på en enkelt side.

```
##      Beskrivelse
# Start med en kode og en  tabel.  Find et ord
# i tabellen, der svarer til koden.  Tabellens
```

```
# sidste opslag er koden NULL og giver svaret
# "ellers".
#     Parametrene er nøglekoden, en tabel,
# en tabelpointer og et resultat. Tabellen
# består af opslag, hver med en byte kode og
# et dobbeltbyte ord: sidste bytekode er NULL.
#     Koden er en byte i Acc., HL er pointer,
# og HL returnerer desuden resultatet. Acc
# og HL skal initialiseres før rutinen.
#.....
#
# TABLE.SEARCH
# ind:  (HL) => TABELSTART - 1 (adresse)
#       Acc = nøgle (byte)
#       TABEL: Cn=kode (8 bit), Wn=ord(16 bit)
#              C1,W1,C2,W2, ....,0,Ww
# ud:   HL = ord jævnfør tabelopslag
        ORG  00
TBLSCH: PUSH DE
        LD   D,A          ; nøgle
; DO UNTIL ((nøgle=kode) OR (kode=0))
TSI:    INC  HL
        LD   A,(HL)       ; tabel-kode
        INC  HL
        LD   E,(HL)
        INC  HL
        OR   A            ; kode=0?
        JR   Z,TSEND
        CP   D            ; OR  kode=nøgle?
        JR   Z,TSEND
        JR   TSI          ; en gang til
        ;
TSEND:  LD   H,(HL)       ; dan resultat
        LD   L,E
        LD   A,D          ; ryd op
        POP  DE
#.....
```

Selvfølgelig kunne jeg spare to bytes ved at vende det sidste betingede hop, men det kan jeg sikkert også se, når jeg engang vil bruge rutinen. Ellers er skaden da til at bære. Som programmet fremtræder nu er det til gengæld nemt at få øje på løsningen af "logisk ELLER" - problemet til trods for, at det er klaret med betingede hop.

FORTSÆTTES I NN 2.

AF IB LEMCHE.

Program til generering af 100 tilfældige tal,
der kan læses af sorteringsprogrammerne, så-
ledes at de sorterer de samme tal alle fire.

```
100 PO=100                              antal poster max 100
110 SETNEW(0),"TILFÆ:1",R,"I",101       åbner randomfil med 101 heltal
120 SETOUT(0),PO                        læser antal ind på første plads
130 FOR I=1 TO 100                      danner tilfældige tal
150 SETOUT(0),S                         læser til fil efter tur
160 NEXT
170 SETCLS(0)                           lukker for filen "(0)"
```

Program Bubblesort

```
100 INPUT"ANTAL FRA TILFÆLDIGHEDSTABEL";AN
110 SETNEW(0),"TILFÆ:1"                 åbner for (0) med navn TILFÆ på disk 1
120 SETINP(0),PO                        læser max antal ind
130 IF AN>PO THEN AN=PO                 max antal=po
140 PO=AN
150 DIM A(PO)
160 FOR I=1 TO PO                       indlæs ønskes antal
170 SETINP(0),A(I)                      fra (0)
180 NEXT
190 SETCLS(0)                           luk for fil (0)
200 PRINT"BUBBLESORT"                   Sorteringen bygger på en sammenligning
210 I=0                                 mellem to tal, der skifter plads, hvis
220 FOR J=PO TO 2 STEP -1               sammenligningen angiver dette
230 IF A(J)=>A(J-1) THEN 250            linie 230-240
240 T=A(J):A(J)=A(J-1):A(J-1)=T:I=-1I er et flag, der sættes hvis rokering
250 NEXT                                er foretaget.
260 IF I=-1 THEN 210                    Fortsætter til det er sandt!
280 FOR I=1 TO PO                       Udlæsning.
290 PRINTA(I);                          Gennemsnitlig antal flytninger:
300 NEXT                                0.75*(PO↑2-PO)
```

Program Quicksort

```
100 INPUT"ANTAL FRA TILFÆLDIGHEDSTABEL";AN
120 SETINP(0),PO
130 IF AN>PO THEN AN=PO                 Indlæsning af data
140 PO=AN
150 DIM Q(PO)                           Sorteringen bygger på samme princip som
160 FOR I=1 TO PO                       bubblesort, men det foregår over store
170 SETINP(0),Q(I)                      afstande. Man sætter X lig med et til-
180 NEXT                                fældigt tal (f.eks. det midterste), her
190 SETCLS(0)                           efter deles tallene ud på to grupper,
200 ST=LOG(PO)/LOG(2)                   hvor elementer er større end X og en,
210 DIM ST(ST,1)                        der er mindre end X.
220 PRINT"QUICKSORT"                    Denne procedure fortsættes, indtil der
230 L=0:R=1                             ikke kan deles mere. Alle disse oplys-
240 S=1:ST(1,L)=1:ST(R,1)=PO            ninger, må man opbevare i en stack (ST)
250 L1=ST(S,L):R1=ST(S,R):S=S-1         hvis størrelse er LOG(PO)/LOG(2)
260 I=L1:J=R1:X=Q(INT((L1+R1)/2))       linie 200
270 IF Q(I)<X THEN I=I+1:GOTO 270       Antallet af flytninger er ca.:
280 IF X<Q(J) THEN J=J-1:GOTO 280       PO/LOG(PO)
290 IF I>J THEN 320                     Hvis man ikke lige rammer ned i den
300 W=Q(I):Q(I)=Q(J):Q(J)=W             største eller mindste værdi som X !!
310 I=I+1:J=J-1                         hvor der så kræves PO↑2 flytninger
320 IF J>=I THEN 270
330 IF J-L1>R1-I THEN 370
340 IF I<R1 THEN S=S+1:ST(S,L)=I:ST(S,R)=R1
350 R1=J
360 GOTO 390
370 IF L1<J THEN S=S+1:ST(S,L)=L1:ST(S,R)=J
380 L1=I
```

```
390 IF L1<R1 THEN 260
400 IF S<>0 THEN 250
410 PRINT"SORTERING SLUT"
420 FOR I=1 TO PO
430 PRINTQ(I);
440 NEXT
```

Programmet Heapsort

```
100 INPUT"ANTAL FRA TILFÆLDIGHEDSTABEL";AN
110 SETNEW(0),"TILFÆ:1"
120 SETINP(0),PO                        Indlæs data
130 IF AN>PO THEN AN=PO
140 PO=AN                               Metoden foregår ved udvælgelse, men ikke
150 DIM H(PO)                           bare af et element i forholdt til de øv-
160 FOR I=1 TO PO                       rige. Men man finder det mindste sæt be-
170 SETINP(0),H(I)                      stående af først 2 - siden 4 tal osv.
180 NEXT
190 SETCLS(0)                           Langsom for små talmængder
200 PRINT"HEAPSORT"
210 L=INT(PO/2)+1                       Men max antal flytninger er
220 N1=PO                               PO*LOG(PO)
230 IF L=1 THEN 270                     Gennemsnitlig flytninger er
240 L=L-1                               0.5*PO*LOG(PO)
250 A=H(L)
260 GOTO 310
270 A=H(N1)
280 H(N1)=H(1)
290 N1=N1-1
300 IF N1=1 THEN 420
310 J=L
320 I=J
330 J=2*J
340 IF J=N1 THEN 370
350 IF J>N1 THEN 400
360 IF H(J)<H(J+1) THEN J=J+1
370 IF A>H(J) THEN 400
380 H(I)=H(J)
390 GOTO 320
400 H(I)=A
410 GOTO 230
420 H(1)=A
430 FOR I=1 TO PO
440 PRINTH(I);
450 NEXT
```

Programmet Shell-Metzner

```
100 INPUT"ANTAL FRA TILFÆLDIGHEDSTABEL";AN
110 SETNEW(0),"TILFÆ:1"
120 SETINP(0),PO
130 IF AN>PO THEN AN=PO                 Indlæs data
140 PO=AN
150 DIM S(PO)
160 FOR I=1 TO PO
170 SETINP(0),S(I)
180 NEXT
190 SETCLS(0)
200 PRINT"SHELL-METZNER"                Metoden der benyttes er indsætning
210 K=PO                                sammenligningen foregår over store
220 K=INT(K/2)                          afstande. Hvis der er 100 tal, så
230 IF K=0 THEN 340                     foregår 1. sammenligning mellem nr.
240 V=PO-K:J=1                          2 og 52, hvorefter afstanden mellem
250 I=J                                 tallene bliver mindre og mindre, ind-
260 L=I+K                               til de er sorterede
```

```
270 IF S(I)<=S(L) THEN 310
280 T=S(I):S(I)=S(L):S(L)=T
290 I=I-K
300 IF I>0 THEN 260
310 J=J+1
320 IF J>V THEN 220
330 GOTO 250
340 FOR I=1 TO PO
350 PRINTS(I);
360 NEXT
```

Gennemsnitlig flytninger er PO*1.2

Her følger en tidstagning på de fire programmer.
Tilfældene 1 til 3 er 100 tilfældige tal, men de samme for alle sorteringsrutiner.
I tilfælde 4 er tallene anbragt i omvendt rækkefølge.
I sidste tilfælde er tallene næsten sorterede, kun få står forkert.

	1.	2.	3.	4.	5.
Shell	12.6 s	12.0 s	12.9 s	9.7 s	6.5 s
Heap	12.4	12.3	12.5	11.4	13.3
Quick	9.7	8.8	9.7	6.8	7.0
Bubble	73.4	72.4	63.1	98.4	1.4 (!)

Asbjørn.

SÆLGES:

brugt blæser 220 V.(110 mm Ø)				50.00 kr.
25 stk. 24 V relæ 2skift a'				5.00 kr.
IC sokler	8 ben,	14 ben,	16 ben,	24 ben
til	1.00	2.00	2.50	4.50 kr.
DIP stik				2.50 kr.
Lysdioder rød/grøn				1.00/1.50 kr.
2N 3055				5.00 kr.
BC 547 / BC 557				1.00 kr.
IC 555				6.50 kr.
IC 741				5.00 kr.
MANGE andre typer transistorer				
Diode 1N 4148				0.25 kr.
Diode 4005				1.00 kr.
Kredse i 74xx				
Kredse i 4xxx				
24 leder fladkabel pr. meter				25.00 kr.
Printspyd med hunstik				0.15 kr.

Komponetpakke til M.K. EPROM-brænder uden 24 bens sokkel 60.00 kr.

Regulator 78xx og 79xx 11.00 kr.

KØBES

Netdel til NASCOM

HUSK

Jeg laver print efter opgave og har print til sneplov og EPROMbrænder.

Henvendelse til
Hans Ole Groth
09 58 16 03

Her følger en oversigt over 2. ÅRGANG (1981):

A/D CONVERTER(A)	2	18
AKUSTISK MELDER	9	2
ANAGRAM (P)	5	19
AUTOLIST,-START	2	15
BASIC BEGYNDER	7	16
BLS's PASCAL	4	15
BUDGET (B)	4	2
BUFFERKORT N1	6	2
CIRKELTEGNING(B)	1	16
DEC TIL BIN (B)	1	17
EHRENFEST MODEL	10	7
EPROMBRÆNDER	5	9
EPROMBRÆNDER	5.2	2
EPROMBRÆNDER	6	23
EPROMSLETNING	6	25
EPSON NYE TEGN	10	2
EPSON PRINTEREN	4	10
FØR DATAMATKØB	7	9
GRAFIK I ASSEM.	10	10
HEX ADD/MUL	7	11
HEX-DEC DEC-HEX	9	4
HØJTIDERNE	3	16
I/O MULIGHEDER	9	6
INDLÆSNING 1MHZ	9	5
INTELLIGENTE MA.	8	4
INTERRUPT	3	2
KALENDER	9	10
KANSAS CITY INT	5	2
KANSAS CITY INT	5.2	7
KARAKTERPAKKER	3	14
KASSETTEINTERF.	10	3
KONSTANTSTRØMG.	7	8
KRYDSFERANCE(B)	1	12
LÆSERBREVE	1	2
MASKINKODEPGR.	2	17
MASKINSPROG	2	2
MASKINSPROG IV	1	8
MATH 48	1	20
MATRIXREGNING(B)	7	15
MATRIXREGNING(B)	8	2
MEMORYADR.N2	9	12
MEMORYSAMMENLIGN	1	7
MEMORYSAMMENLIGN	2	5
MEMORYTEST(ASS)	1	5
MIKROCOMPUTEREN	2	3
MØNSTER JUL81	10	11
MØNSTERTEGNING	5	5
NASCOM 1 MOD.	4	12
NASCOM DÆMON	5	13
NASPEN	10	8
NETFILTER	3	21
NYHEDER NASCOM	4	14
OMBYGNING AF N1	9	19
ORDLISTE	1	10
ORDLISTE	2	14
ORDLISTEN	3	13
ORDLISTEN	4	9
OTHELLO	6	24
OVERSKRIFTPGR	6	8
PASCALGRUPPEN	10	22
PRG. KARAKTERG.	10	15
REFERAT GENERAL.	5.2	8
REGNSKAB 1980	5.2	13
RENUMERERING(A)	4	4
RS232	3	12
RUBIK PÅ VDU	7	7
RUBIKS TERNING	6	18
SAMMENLIGNING(A)	5	17
SNEPLOV	5	17
SOFTWARESPALTEN	6	9
SPROGSTAMTRÆ	3	11
STANDARDSPROG	4	7
STOCK CAR (B)	2	17
STRØMFORS.MOD.	10	9
TABULERING (A)	4	6
TEGNEPRG.(B)	10	9
TEGNING	3	17
TEKNISK BREVK.	6	6
TIPSPRÆMIER(B)	1	14
TONER I BASIC	6	14
TOOLKIT(BITS&PC)	1	18
TV-->MONITOR	3	19
TÆLLER BÅNDOPTA.	9	8
UDV.FEJLM. BASIC	7	5
VEDTÆGTER	5.2	12
Z80 EKSTRA KODER	2	6
ZAMBIELAND (B)	5	16
ZEAP	4	8
ZILOG NYHEDER	10	5

I et brev modtaget fra Lars Rugård Jensen, Ørnevej 12 st.tv 2400 NV (01 14 18 55 (8-16)/01 85 31 51 (efter 17)) tilbydes vejledning til selvbyggere af disksystem til ca. 3300 kr. incl drive, controllerkort og DOS-system. Lars har selv bygget et system og vil gerne give sine erfaringer videre til interesserede. Hvis der er stor interesse for sagen, vil der komme en byggeartikel her i bladet. DOS-systemet har 13 kommandoer med mulighed for kald fra pascal, basic og maskinkodeprogrammer.

KEYBOARD - RUTINER

Det har ofte irriteret mig, at monitorens keyboardrutine kun kan checke en tast ad gangen, samtidig med at den er ret langsom. Den er faktisk dårlig, hvis man vil lave hurtige maskinkode-spil. Derfor kiggede jeg lidt på den, før jeg gik igang med det sidste spil, jeg har skrevet, det er et der hedder "Falling stones", et spil i invader-stilen, hvor netop fart og samtidigt virkende taster er en forudsætning. Først en kort forklaring på hvordan dimsen egentlig fungerer.

Keyboardet er af polling matrix typen, dvs at det ikke danner ASCII koden direkte, istedet skal det gennemsøges, det vil da returnere nogen bits der angiver hvilke taster der har været nedtrykkede. Først nulstilles tælleren med en puls på port 0, bit 1 ved hjælp af den rutine der hedder FFLP, derefter gemmes de første 8 bit i keyboard-map. Nu tælles keybord counteren 1 op igen med FFLP, port 0 bit 0 denne gang. Dette gøres nu ialt 9 gange, således at den 1. og den 9. addresse i map'en indeholder samme data. Hvis der undervejs var en tast nedtrykt, vil den tilhørende bit nu være 1 (sat) i map'en.

Tasterne har følgende positioner :

bit	7	6	5	4	3	2	1	0
0C01:	ctr	lf	@	shift		-	enter	back
0C02:		↑	T	X	F	5	B	H
0C03:		vp	Y	Z	D	6	N	J
0C04:		np	U	S	E	7	M	K
0C05:		hp	I	A	W	8	,	L
0C06:		graf	O	Q	3	9	.	;
0C07:		Æ	P	1	2	0	/	:
0C08:		Å	R	space	C	4	V	G
0C09: = 0C01								

Monitoren laver nu en masse undersøgelser, der skal ordne problemer med debounce, shift,control og grafik tast, om en nedtrykket tast er sluppet osv. Alt dette er unødvendigt ved anvendelser i spil, her ønsker vi blot at vide kort og godt, er en tast nede eller ej! Dette kan gøres ved at kalde følgende rutine der blot loader keyboard map'en, senere laves en helt enkel bit-test, på positionen af den tast man vil checke.

```
        KMAP:   EQU   0C01H
        FFLP:   EQU   45H

        LKM:                        ;LOAD MAP
F5              PUSH  AF
E5              PUSH  HL
C5              PUSH  BC
21010C          LD    HL,KMAP
3E02            LD    A,2
CD4500          CALL  FFLP          ;RESET COUNTER
0608            LD    B,8
DB00    LKZ:    IN    A,(0)         ;LÆS PORT
2F              CPL
77              LD    (HL),A
23              INC   HL
3E01            LD    A,1
CD4500          CALL  FFLP          ;TÆLLE
10F4            DJNZ  LKZ
C1              POP   BC
E1              POP   HL
F1              POP   AF
C9              RET
```

Koden er fuldt relokerbar, ved anvendelse fra basic kan den ligge fra 0C80 og opad. Den kaldes med DOKE 4100,3200:A=USR(0). Senere kan du poke fra 3073 til 3081 og and'e med 1 2 4 8 16 32 osv. alt efter hvilke taster du vil checke.

Denne rutine fandt jeg udmærket til formålet, dermed være ikke sagt at den er perfekt, man kunne f. eks. sagtens resette tælleren, tælle op, læse port og teste bits altsammen i gennemløbet af programmet, hvilket nok ville spare et par micro-sec, men også gøre en afhængig af den rækkefølge tasterne aflæses i. Nå, men efter dette lille indlæg håber jeg, at der kommer nogen meget hurtige maskinkode spil....

Hilsen fra

Erling Thorup Madsen
Vejlesøvej 10
2840 Holte
(02) 42 49 97

RUSTENBORGVEJ 1, 2800 LYNGBY

MEDLEMSMØDE. Sønd.d.14.Februar.
Kl. 12.30.

Emne: Sammenligning af højniveau-sprog.
v. Frank Damgaard.
samt arbejdende anlæg.
c/o Pæd. Central

VELKOMMEN TIL NYE MEDLEMMER:

224.
Howeni Aps.
Aboulewarden 58.
2200 kbh.N.

225.
Aksel Glyming.
Fredbovej 2.
4571 Grevinge.

226.
Erling Simonsen.
Markvænget 80.
7700 Thisted.

227.
Ole Poulsen.
Østerdalsgade 14.
2300 Kbh.S.

228.
Lars R. Jacobsen.
Hedeparken 137.
2750 Ballerup.

229.
Georg Landbo.
Fasanvej 7.
7190 Billund.

230.
Jacob Brix Nielsen.
Grønnegade 36.
9362 Gandrup.

231.
Niels Jørgen Kofod.
Højbjerg Byvej 6.
8840 Rødkjærsbro.

232.
Peter Meisner.
Nr.Farimagsgade 67.2.
1364 Kbh.K.

233.
Troels Jørgensen.
Srandvej 6.
4800 Nykøbing F.

234.
A.Augustenborg.
Halskovvej 52.2.
Postbox 55.
4220 Korsør.

235.

236.
Ivan Brønnum.
Lundemosen. 41.
2670 Greve Strand.

237.
Peter Jørgensen.
Damagervej 19.2.tv.
2450 S.V.

238.
Eli Schrøder.
Kjellerupsgade 13.
9560 Hadsund.

ALMINDELIGE OPLYSNINGER

OM FORENINGEN :

Bestyrelsens sammensætning:

Formand	Asbjørn Lind Sidevolden 23 2730 Herlev 02 91 71 82
Næstformand	Jesper Skavin Broholms Alle 3 2920 Charlottenlund 01 64 03 14
Kasserer	Erik Hansen Lyngby Kirkestræde 6.1 2800 Lyngby helst skriftlig henv. 02 88 60 55 (Torsdag kl.17 til 18)
Sekretær	Carsten Senholt Blommevangen 6 2760 Måløv 02 66 19 65
	Ole Hasselbalch Vibeskrænten 9 2750 Ballerup 02 97 70 13
	Søren Sørensen Højlundvej 13 3500 Værløse 02 48 31 01
	Frank Damgård Kastebjergvej 26A 2750 Ballerup 02 97 10 20

Henvendelse til foreningen:

Indmeldelse, adresseændringer o.l. til **kasseren**
Programbibliotek til **næstformanden**

Øvrige henvendelser til **formanden**
(herunder annoncer/stof til NASCOM NYT)

Indmeldelsesgebyr:	25,00 kr.
Kontingent 1.1.82 - 1.7.82:	40,00 kr.

Oplag: 300

Redaktionen sluttet den 11.01.82
Husk at gamle numre kan købes hos Ole for 10 kr./stk +porto
Printerservice hos formanden
Bånd og bokse kan købes hos Carsten til følgende priser:
10 bånd 45 kr., ekstra etiketter 0,25 kr./stk og bokse 1,50 kr./stk + porto.
Annoncepris 0.75 kr. pr. A4 side (siderne 4 - n-2)*oplag
Indlevering foreningens adresse.