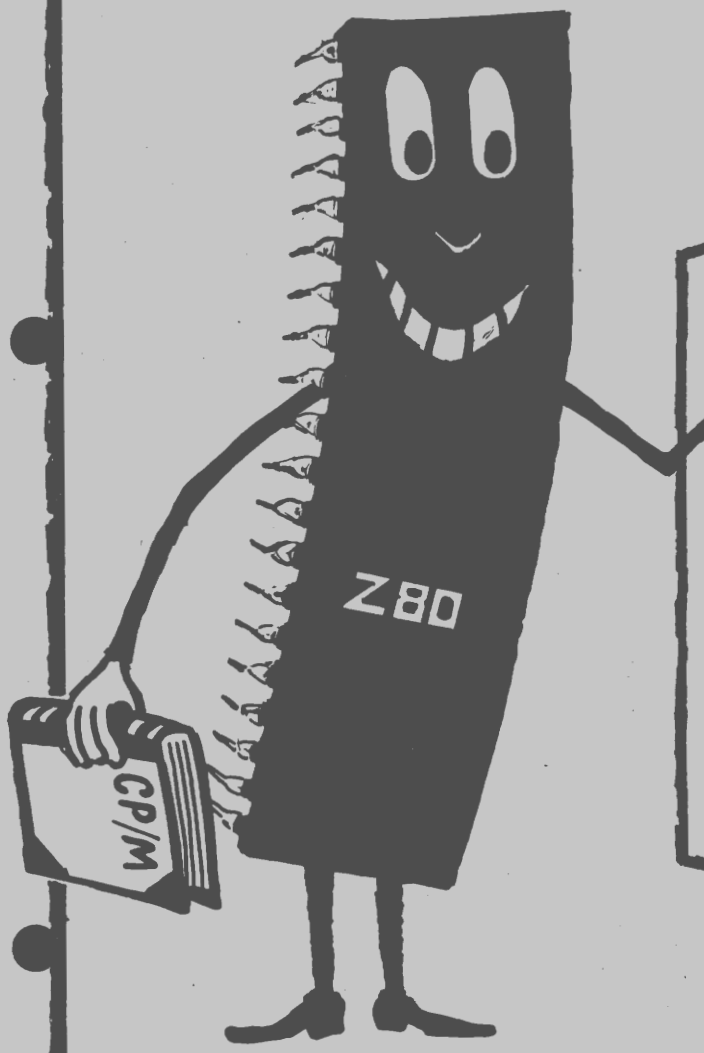
NAS BUS Z80 NYT

UDGIVET AF Z80 BRUGERGRUPPEN 5. ÅRGANG NR. 8

O K T O B E R 1 9 8 4

Så er vi igang igen for alvor. I medlemundersøgelsen er der givet udtryk for, at der er for få konstruktioner i bladet ! Dette har jeg forsøgt at rode bod på i dette nummer. Men der kan godt komme flere i kommende numre, så prøv at skrive ind til bladet med dine egne konstruktioner. I dette nummer har jeg også anbragt indholdsfortegnelsen for 1983 af Z80 NYT!! Den skulle havde været bragt i januar, men den daværende redaktør havde glemt den.



INDHOLD:

ALMINDELIGE OPLYSNINGER OM FORENINGEN

HENVENDELSE TIL FORENINGEN TIL FORRETNINGSFØREREN:

I. SKAVIN
Broholms alle 3
2920 Charlottenlund
Telefon 01 - 64 03 14

Hertil skal rettes henvendelse om indmeldelse, adresseforandring, salg af foreningens materialer (bånd, blade og programmer). Øvrige henvendelser af generel art til formanden. Stof og annoncer til foreningens blad sendes til Asbjørn Lind.

Indmeldelsesgebyr:	25.00 kr.
Kontingent 1.7.84 - 1.7.85.	150.00 kr.

Annoncering for medlemmer er gratis i Z80 NYT. For andre 250 kr. pr. A4 side.

Bestyrelsesmedlemmer:

Formand:
René Hansen
Bispevangen 6,13,th
2750 Ballerup
Tlf. 02 65 59 76.
Kl. 18.30 - 21.00

Næstformand:
Jesper Skavin
Træffes ikke før 1.12.84

Teknisk redatør:
Ole Hasselbalch
Vibeskrænten 9
2750 Ballerup
Tlf. 02 97 70 13.

Frank Damgaard
Kastebjergvej 26A
2750 Ballerup

Per Thomsen
Ulspilager 75
2791 Dragør

Redaktør for Z80 NYT:
Asbjørn Lind
Sidevolden 23
2730 Herlev
Tlf. 02 91 71 82. (20.00 - 21.00)
(man. til torsdag)

Sidste frist for indlevering af stoft til næste nummer:
20.10.1984

God contra dårlig programmering.

Samt gode råd fra Peter Villadsen

Denne artikel handler om, hvordan man bærer sig ad med at skrive gode programmer, hvad gode programmer egentlig er, og hvad det er, der adskiller et godt program fra et dårligt.

Et godt program er ikke bare et program der virker, selvom der desværre er mange der tror det. Det er jo nemlig sådan, at programmer der skal bruges mere end een gang (og det gælder vel for langt de fleste programmer), altid bliver ændret undervejs, bliver udvidet til at kunne mere og mere, og bliver filet mere og mere til. Derfor kan man sige at et godt program er et program der virker, og som bliver ved med at virke, selvom man skal lave ændringer i programmet et stykke tid efter man har skrevet det. Der findes en række måder man kan skrive programmer på, således at man selv (og andre) kan hitte rede i programmet og "forstå" det. Det er faktisk ikke særlig svært at skrive programmer hvor dette er tilfældet - det er faktisk kun en vanesag.

Når man skal skrive et program, er det som regel fordi man har en eller anden ide til hvad programmet skal kunne. En af de metoder man kan bruge til at skrive et godt program er den metode som ofte kaldes "Top Down" planlægning. Den består (groft sagt) i følgende :

Man starter med at skrive programmets hovedtræk i almindeligt sprog, uden at tage nogen uvæsentlige detaljer med. Denne "opskrift" består af en række trin, som så efterhånden hver især gøres lidt mere detaljeret etc. etc. Når man så har opnået et passende niveau af detalje, begynder man at kode sit program i et eller andet programmeringssprog, f.eks. BASIC, Pascal eller Assembler.

Hvad er så fordelen med denne metode? Det lyder jo som om man skal gøre arbejdet flere gange, ved at sidde og detaljere programmet mere og mere? Det må jo tage hundrede år at skrive et program på denne måde. Hvorfor kan man ikke klø på og gå igang? Der er selvfølgelig flere grunde til dette.

1) Når man følger denne opskrift er det lettere for en selv og for andre at overskue hvad det pågældende program egentlig kan, og hvordan det gør det.

2) Det er den eneste måde man kan skrive store programmer på, fordi man skriver en lille del af programmet ad gangen. Hver del kan så testes, uafhængigt af de andre dele.

3) Det er lettere at finde fejl i et program man har skrevet på denne måde.

4) Det tager faktisk ikke længere at udvikle et program på denne måde, fordi man sparer en masse tid når man skal finde og rette fejl i sit program.

5) Programmet er ikke bundet til et specielt programmeringssprog. Hvis man vil skrive programmet i et andet programmeringssprog, er det bare nødvendigt at se på sin planlægning, og kode forfra. Bemærk, at der er meget stor forskel på at programmere og at kode. At programmere er selve planlægningen af programmet, medens at kode bare er det at omsætte planlægningen til et eller andet programmeringssprog.

Der er nogle programmeringssprog der er lettere at have med at gøre end andre. Højniveausprog som Pascal ligger tæt på den metode man skal løse problemet i, mens man i assembler og Basic er nødt til at være mere kringlet i opbygningen af koden, og dermed gå dybere ned i detaljer før man kan kode programmet.

Vi må vist hellere tage et eksempel:

Vi skal lave et program der skal finde alle de primtal der er mindre end et eller andet givent tal T. Vi starter med at opskrive den metode vi vil løse opgaven efter, uden nogle uvæsentlige detaljer (der findes væsentlig bedre metoder end denne!)

(1) Indtast tallet som primtallene skal være mindre end (vi kalder dette tal for T)
(2) for alle tal x imellem 2 og T udføres:
(3) Hvis x er et primtal udskrives x.

Vi specificerer nu dette lidt nærmere: Det ligger i definitionen af et primtal at det ikke har andre divisorer end 1 og sig selv. Næste udvikling bliver følgende:

(1) Udskriv en rykker til brugeren om at han skal indtaste T
(2) Indtast T
(3) for alle tal X imellem 2 og T udføres:
(4) Hvis X har divisorer forskellige fra 1 og X
udskrives X.

(1)-(3) er der jo ikke meget kød på, så vi koncentrerer os om at finde ud af hvordan vi finder ud af om et tal har nogen divisorer der er forskellige fra 1 og tallet selv, eller ej.

(1) For at finde ud af om et tal X har nogen divisor forskellig fra 1 og fra tallet selv, undersøges det om der er noget tal (vi kan kalde det I) imellem 2 og tallet selv således at I går op i X. Hvis dette er tilfældet har X en divisor (nemlig tallet I).

Efter at have tænkt os lidt om, indser vi at X går op i Y hvis X/Y giver en rest der er 0. Udover dette ses det at X/I altid giver rest forskellig fra 0 hvis I er større end X/2. Vi har derfor:

(1) For I imellem 2 og X/2 udføres:
(2) Hvis X/I har en rest = 0 har X divisoren I.

Vi har altså sammenlagt:

(1) Udskriv en rykker til brugeren om at han skal indtaste T
(2) Indtast T
(3) for alle tal X imellem 2 og T udføres:
(4) for I imellem 2 og X/2 udføres:
(5) Hvis X/I har en rest = 0 er I divisor og X er ikke et primtal.
(6) Hvis vi ikke fandt nogle divisorer udskrives X.

Først nu er vi nået så langt at vi umiddelbart kan kode programmet i et eller andet programmeringssprog. Vi viser eksemplet i Basic, Comal-80 og Pascal; det er måske en god øvelse at finpudse disse programmer og få dem til at køre hurtigere. Som sagt før er der andre metoder der er adskilligt mere effektive, men disse metoder vil være for omfattende at planlægge her.

```
10 REM Dette er et primtals program.
20 REM Programmet udskriver primtallene imellem 2 og et
30 REM tal som brugeren indtaster.
35 REM Sidst opdateret d. 30/7/1984 af Ellen Danielsen.
37 TRUE=(1=1): FALSE=(1=2)
40 PRINT " **** PRIMTALS PROGRAM ****"
50 INPUT "Indtast den øvre grænse for primtallene"; T
60 FOR X=2 TO T
70 INGENDIVISOR=TRUE
80 FOR I=2 TO X/2
90 IF X/I=INT(X/I) THEN INGENDIVISOR=FALSE
100 NEXT I
110 IF INGENDIVISOR THEN PRINT "  PRIMTAL :";X
120 NEXT X
130 END
```

fig. 1. Primtals program i Basic.

```
0010 // Dette er et primtals program.
0020 // programmet udskriver primtallene imellem 2 og
0030 // et tal som brugeren indtaster.
0040 // Sidst opdateret d. 30/7/1984 af Zebedæus Zachariasen.
0045 //
0047 INPUT " INDTAST DEN ØVRE GRAENSE FOR PRIMTALLENE" : T
0050 FOR X:=2 TO T DO
0060   INGENDIVISOR:=TRUE
0070   FOR I:=2 TO X/2 DO
0080     IF X MOD I = 0 THEN INGENDIVISOR:=FALSE
0090   NEXT I
0100   IF INGENDIVISOR THEN PRINT "  PRIMTAL :";X
0110 NEXT X
0120 END
```

fig. 2. Primtals program i Comal-80.

```
Program Primtalsprogram;
     (*
        Dette er et primtals program.
        programmet udskriver primtallene imellem 2 og
        et tal som brugeren indtaster.

        Sidst opdateret d. 30/7/1984 af  Hanne V. Nielsen.
     *)
     var
       ingendivisor : boolean;
       X,I,T        : integer;
     begin
       write (' Indtast den øvre grænse for primtallene  ');
       read (T);
       for X:=2 to T do
       begin
         ingendivisor:=true;
         for I:=2 to trunc (X/2) do
            if X mod I = 0 then ingendivisor:=false;
         if ingendivisor then writeln (' Primtal : ',X)
       end
     end.
```

fig. 3. Primtals program i Pascal.

Ved lidt større programmer end det vi har lavet her, vil det være naturligt at lave hver større del af programmet til et underprogram. Dette fremmer yderligere overskueligheden, og desuden gør dette det muligt at teste hver underprogram for sig. Pascal og Comal (og mange andre sprog iøvrigt) har mulighed for at have rigtige procedurer og funktioner, og det er nok allermest dette der udmærker disse sprog frem for det primitive Basic. Men der er ingen grund til at fortvivle fordi man programmerer i Basic. Hvis man følger visse regler er det faktisk muligt at lave noget der ligner rigtige strukturerede programmer. Vi vil skrive nogle af disse regler op, samtidigt med at vi vil give nogle gode råd om programmering i Basic i særdeleshed.

Opbyg programmet med et hovedprogram og en serie underprogrammer eller subrutiner, som f.eks.:

```
 10 REM HOVEDPROGRAM
  :
  :
  :
100 REM HER SLUTTER HOVEDPROGRAMMET.
110 REM
120 REM UNDERPROGRAM DATOTEST.
125 REM Checker om en dato er gyldig.
130 REM Indgangs variable : Måned : Månedens nummer (Dec. = 12)
140 REM                     År    : Årstallet (f.eks. 1978)
150 REM                     Dag   : Dag i måned (1..31)
160 REM
170 REM Udgangs variable  :  Gyldigdato :
```

```
180 REM                          Hvis den dato der er an-
190 REM                          givet er gyldig er denne
200 REM                          variabel = TRUE og ellers
210 REM                          FALSE.
220 .......
230 :
240 :
250 :
260 RETURN
270 REM
280 REM UNDERPROGRAM HEX_TO_DEC
290 REM Konverterer et tal fra hexadecimalt til decimalt.
290 REM Indgangs variable :  STR$ :  Indeholder de  hexadecimale
300                                  ciffre
310 REM
310 REM Udgangs variable  :   TAL :  Indeholder det konverterede
320 REM                              tal.
330 REM Rutinen ændrer variablerne : TEMP1 og TEMP2.
340 REM
350 .........
360 :
370 :
380 RETURN.
```

Her opnår man at man har helt klarhed over hvilke variable der bruges til hvad, og hvilke der ændres under vejs. Det er nemlig en meget almindelig kilde til fejl : man er ikke opmærksom på at en eller anden variabel ikke har den samme værdi før og efter kaldet af en subrutine, hvis denne variabel bruges i subrutinen! Dette gøres selvfølgelig endnu værre af at der er nogle Basic fortolkere der kun skelner imellem variable på de første 2 bogstaver i variabelnavnet. Ved at bruge den opstilling vi har vist her, opnår man at kunne genbruge de underprogrammer man engang har lavet, uden at skulle bekymre sig om hvordan de virker, hvis man ved at man en gang har testet dem.

Dette indebærer også at man kun må bruge underprogrammet ved at kalde f.eks:

```
GOSUB 100
```

Ikke noget med at hoppe ind i koden fordi man herved lige kan spare et par bytes! Iøvrigt skal underprogrammet forlades med RETURN og ALDRIG med et GOTO! Hvis man har en Basic fortolker der kan hoppe til linienumre der er givet ved udtryk, kan man starte sit program ved at tildele en række variable værdier svarende til linienumrene:

```
10 TESTDATO = 100
20 HEX_TO_DEC = 270
```

og så bruge

```
67 GOSUB TESTDATO
```

Dette siger mere end GOSUB 100, men man risikerer at skulle ændre disse variables værdi når man redigerer i programmet.

Brug dog variabel navne der siger noget om den funktion som variablen har i programmet! Det er den lille ting med den store virkning. Det giver meget mere mening at skrive:

```
10 SALGSPRIS = INDKØBSPRIS*MOMS + AVANCE

end

10 A = B*C + T
```

Det koster kun så lidt i tid foran tastaturet og gør programmet 10 gange mere forståeligt.

FOR sætninger er meget nyttige, men desværre også lette at misbruge. Vi slår fast: En forløkke har formen:

```
FOR variabel = udtryk1 to udtryk2
:
:
NEXT variabel
```

eller

```
FOR variabel = udtryk1 to udtryk2 STEP udtryk3
:
:
NEXT variabel
```

I koden imellem FOR og NEXT antager variabel succesivt værdierne imellem udtryk1 og udtryk2. Hvis der ikke er en tilknyttet STEP del, tælles variablen 1 op hvergang man når NEXT, ellers tælles der op med ved værdien af udtryk3 (som selvfølgelig kan være negativ, hvorved der tælles ned). Når NEXT mødes, tælles variablen op (eller ned), og der hoppes til linien med FOR sætningen. Hvis variabel nu er større end udtryk2, hoppes der til linien efter NEXT.

Problemet er nu: Det er ikke defineret om koden imellem FOR og NEXT skal udføres 0 eller 1 gange hvis udtryk1 >= udtryk2, med positivt udtryk3, eller udtryk1 <= udtryk2 med negativt udtryk3. Nogle Basic fortolkere gør det ene, andre det andet. Hvis man skal skrive et program som andre også skal have glæde af, skal man undgå at lave programmer hvor man risikerer denne situation.

Alle ved at man ikke må hoppe ind i koden imellem FOR og NEXT (det resulterer uvægerligt i "NEXT error"). Hvad mange ikke ved er at man heller ikke bør hoppe ud af en FOR-løkke med en GOTO sætning. Dette er fordi fortolkeren husker værdien af løkke styre variablen indtil det NEXT hvor der hoppes ud af løkken. Hvis man så forlader løkken før dette er tilfældet, risikerer man at disse værdier hober sig op, indtil der ikke er plads til flere.

Det betragtes iøvrigt som dårlig programmering at benytte løkkestyre variablens værdi efter NEXT.

Til en FOR sætning hører EN og KUN EN NEXT sætning. Man bør aldrig lave følgende:

```
10 FOR I=A TO B
20 :
30 :
40 IF I>3 THEN NEXT I : REM Næste værdi
50 :
60 NEXT I
```

Dette skyldes linie 50 udføres både hvis I<=3 og hvis I>B. Hvis man skal realisere dette, skal det gøres således:

```
10 FOR I=A TO B
20 :
30 :
40 IF I>3 THEN GOTO 60
50 :
60 NEXT I
```

Man må aldrig lave programmer der benytter FOR-løkker der ligger skævt i forhold til hinanden.

```
10 FOR I=1 TO 3
20 :
30  FOR J=1 TO 4
40 NEXT I
50 NEXT J
```

I en datamat kan man i virkeligheden ikke repræsentere reelle tal, dvs tal der ikke er heltal. Man gør det, at man laver noget der ligner det ønskede tal, men det er kun en tilnærmelse, i de fleste tilfælde. Dette kan godt give uønskede resultater, hvis man ikke er opmærksom på det.

Tallet 0.75 decimalt kan udtrykkes i det binære talsystem, som jo er det eneste talsystem en datamat i virkeligheden forstår, som

0.11

fordi 1*2↑-1 + 1*2↑-2 = 0.75. 0.75 kan altså udtrykkes præcist i computeren. Hvis vi derimod prøver at konvertere tallet 0.1 decimalt til binært stiller situationen sig noget anderledes:

0.00011001100110011.....

Dette tal kan ikke omsættes til det binære talsystem med et endeligt antal ciffre, og dermed heller ikke i en computer med et endeligt antal bits afsat til at rumme et tal. Dette betyder at vi må forvente fejl ved brug af reelle tal.

```
10 SUM=0
20 FOR I=1 TO 1000
30 SUM = SUM + 0.1
40 NEXT I
50 PRINT SUM
```

giver langtfra resultatet 100.000 som man skulle forvente. På en almindelig MicroSoft Basic med 6 1/2 ciffre giver resultatet 99.9905. Hvis man istedet for 0.1 havde brugt et tal der kan omsættes præcist til binært inden for det antal bits man har til rådighed til et tal (f.eks. 0.75 eller 0.5), ville man have fået det rigtige resultat.

Læren af alt dette er, at man aldrig bør stole for meget på de relle tal som maskinen afleverer. Derfor er det ikke klogt at sammenligne to reelle tal direkte, som i

```
10 IF A=B THEN ...
```

Hvis A og B ligger tæt på hinanden, kan testen falde ud til hvilken side det skal være. Det er langt bedre at bruge en test af formen:

```
10 IF ABS (A-B) < TOLERANCE THEN ...
```

hvor TOLERANCE er et tilpas lille tal.

Basic har een stærk facilitet, nemlig det der kaldes dynaminske strenge. Dette betyder at strenge kan fylde mere eller mindre i lageret, alt efter hvad programmet gør ved dem. Men alt har jo sin pris her i verden, og det har dette altså også. Ulempen er at denne flexibilitet gør at fortolkeren skal rokere rundt med sine variable i lageret, alt efter hvordan strengene ændrer længde. Dette gøres ved at bruge løs af pladsen indtil der ikke er mere, og så "rydde op" og fortsætte med kørslen af programmet. Denne oprydning (som ofte kaldes "Garbage Collection", Affalds opsamling) tager sin tid, ofte flere sekunder. Lad os forestille os at vi skulle bruge en streng der er 100 tegn lang. Vi kan ikke umiddelbart tildele en sådan streng til en variabel, så vi prøver en anden (og temmelig ufiks!) metode.

```
10 A$=""
20 FOR I=1 TO 100
30 A$=A$+" "
40 NEXT
```

Programmet er i og for sig godt nok, men det sluger lige 5050 bytes når det køres! Dette kommer af at maskinen laver plads til hver ny version af A$. Den totale plads der bruges under udførelsen bliver altså:

1+2+3+.....+99+100 = 5050 bytes.

Hvis ikke der er plads til dette, må maskinen udføre Garbage Collection (mens brugeren henter en kop kaffe). Hvis vi istedet brugte:

```
10 A$=""
20 FOR I=1 TO 10
30 A$=A$+"          "
40 NEXT
```

ville programmet kun bruge

10+20+30+...+90+100 = 550 bytes.

Garbage collection mekanismen sættes iøvrigt også igang når man bruger FRE(x) kommandoen.

Hvis man har et program som skal køre hurtigt er visse triks man kan bruge for at presse hastigheden lidt op.

1. Man fjerner alle udregninger hvis værdi ikke ændres fra inden i en løkke. Eksempelvis:

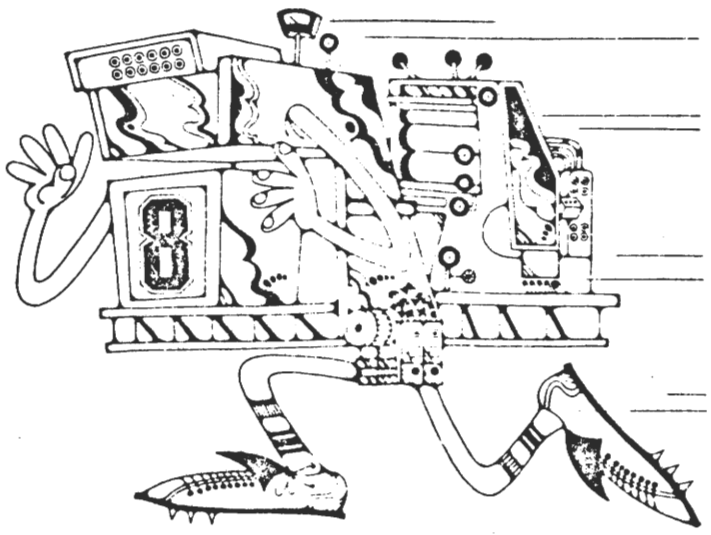
```
10 FOR I=1 TO 100
20  A(I) = SIN(B)*2*I
30 NEXT I
```

kan med fordel laves om til:

```
10 TSINB = 2*SIN (B)
20 FOR I=1 TO 100
30  A(I)=TSINB*I
40 NEXT I
```

2. Når et basic program refererer til en variabel, skal fortolkeren slå værdien op i en tabel over variable og deres værdier. Denne tabel er udformet som en liste, dvs. det tager længst tid at finde de variable som mødes sidst i programmets forløb. (Dette gælder ikke for BBC computeren). Derfor kan det betale sig at anføre de variable som bruges mest, i begyndelsen af programmet.

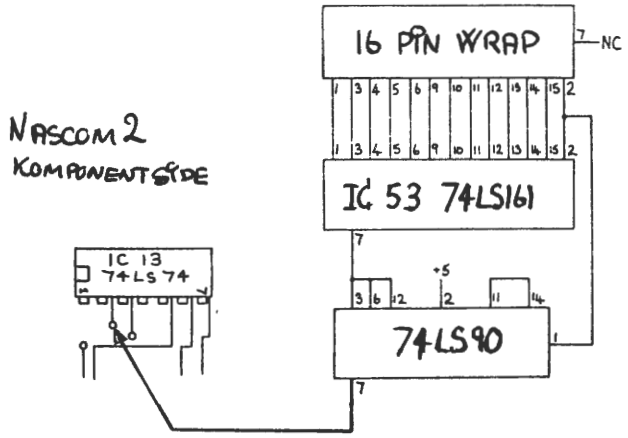
3. Når fortolkeren møder et GOTO LINIENR (eller GOSUB LINIENR), findes linien med nummeret LINIENR ved at starte fra det laveste linienummer og søge opad indtil den finder det rette linienummer. Derfor kan det betale sig at skrive de underprogrammer der bruges hyppigst først i programmet, dvs. med laveste linienummre.



Slut med gyngesygen.

Denne gyngesygeafhjælper vil totalt fjerne enhver gyngesyge på Nascom 2 skærmen. Den er usynlig for brugeren og den berører ikke afviklingen af programmer på nogen måde.

Den ekstra 74LS90 kreds deler præcis i det rette forhold i modsætning til de på Nascom 2 værende delere (11 og 13 delere) i forhold til frekvensen. Den nemmeste måde er at optage IC 53 og isætte en wrire-wrap sokkel, hvor 7 ikke føres op til 74LS161, men tages fra 74LS90's ben 3. Indgangsfrekvensen tages fra ben 1 på IC 59. Som kan findes nær IC 13. Men se diagram for yderligere detaljer.



Slut med screen flash.

Dette kredsløb indsætter et waite state, hvis elektronstrålen befinder sig midt på den synllige del af skærmen. Det vil sige, at der kun sker en opdatering, når der er returstråle. Resultatet er, at der ikke kommer nogle hvide striber på skærmen under opdatering. F.eks. vil man ikke kunne se en tabulering af bare ens byte. Kun ved at iagttage de skiftende memoadresser afsløres, at der skrives til skærmen.

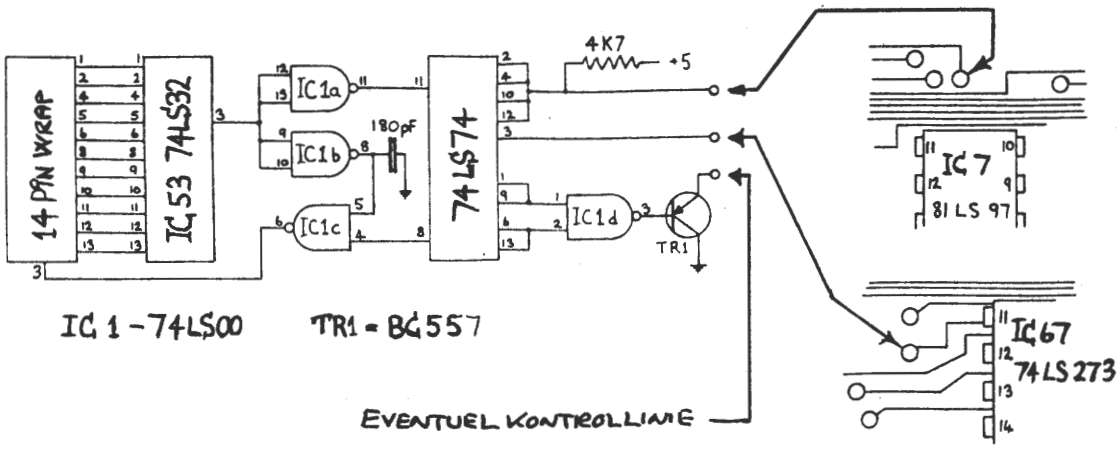
Denne gang er det IC 69 (74LS32), der skal op af soklen og ben 3 skal brydes og køres gennem to ekstra kredse (74LS74 og 74LS00). Se godt på konstruktionstegningen inden du går i gang. Der er også en tegning, hvor du nemmest kan finde de berørte signaler.

Inden du sætter spænding på **skal du fjerne IC 58 (74LS123).** Denne kreds danner en blanking puls, hver gangen skærmen accesses. Denne blanking puls skal fjernes og det gøres nemmest ved at fjerne hele kredsen!

På diagrammerne er **ikke** vist forsyningsspændingerne. Husk dem nu.

Hvis du har brug for uhyggelig præcis timing samtidig med at du skriver til skærmen, kan du forbinde kontrollinien til 0 volt (evt. gennem en port), hvorved enheden sættes ud af kraft. En tidsforsinkelse på ca 1% kan forventes ved afvikling af programmer.

Hvis du kører med 2 MHz, vil det være klogt at skifte IC 74LS74 ud med en 74S74, samtidig kan du fjerne kondensatoren på 180 pF, skønt dette nok ikke vil være nødvendigt på alle maskiner - men så vil den også køre med alle mulige hastigheder.



Jeg har monteret begge dele - det kører bare strålende. Det er kun små, men meget kraftfulde forbedringer. Asbjørn Lind.

INDHOLD 1983 af Z80 NYT

PROGRAMABEL BAUD RATE GENERATOR

for Nascom 2

af René Hansen

Som lovet i forrige nummer af Z80 NYT, bringer jeg denne konstruktion, en Programabel Baud Rate Generator for Nascom 2.

Baud Rate Generatoren er en Weston Digital kreds som hedder WD8116M-00 eller WD1943M-00. Dette er en krystalstyret Dual Baud Rate Generator.

Generatoren har 2 stk, 4 bit indgange, som kan programmeres uafhængigt af hinanden, til at levere Clock signalerne til UART'ens RCLK og TCLK indgang. Dette giver mulighed for at køre splidt speed i alle de nedenstående baud hastighedder.

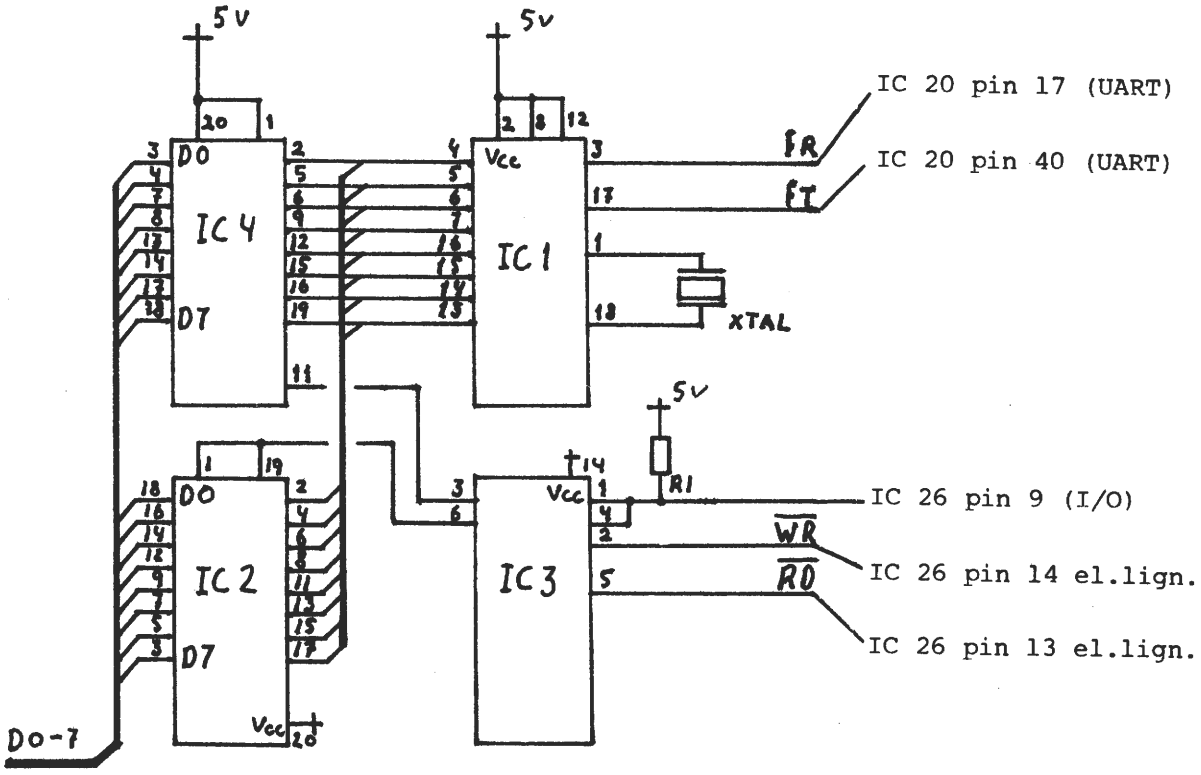
Til foremålet har jeg valgt port 3, da denne dels ikke er anvendt og dels ligger i behalig nærhed af UART'ens porte.

Portens 4 mindst betydende bit (LSB), er til at sætte modtage hastigheden, og de 4 mest betydende bit (MSB), er til at sætte sende hastigheden.

Tr/Re byte	Baud rate	Output freq.	% error	Duty cycle
0 0 0 0	50	0.8 Khz	0	50/50
0 0 0 1	75	1.2	0	50/50
0 0 1 0	110	1.76	- .006	50/50
0 0 1 1	134.5	2.152	- .019	50/50
0 1 0 0	150	2.4	0	50/50
0 1 0 1	300	4.8	0	50/50
0 1 1 0	600	9.6	0	50/50
0 1 1 1	1200	19.2	0	50/50
1 0 0 0	1800	28.8	0	50/50
1 0 0 1	2000	32.149	+ .465	50/50
1 0 1 0	2400	38.4	0	50/50
1 0 1 1	3600	57.6	0	50/50
1 1 0 0	4800	76.8	0	50/50
1 1 0 1	7200	115.2	0	50/50
1 1 1 0	9600	153.6	0	50/50
1 1 1 1	19200	307.2	0	44/56

Eks. 55 hex = 300 baud, 66 hex = 600 baud, 77 hex = 1200 baud,
AA hex = 2400 baud, CC hex = 4800 baud, etc.

71 hex giver 1200 baud transmit med 75 baud handshake.



IC 1 intern Data bus,
kan også hentes på UART'ens
pin 5 til 12.

0 volt = IC 1 pin 11, IC 2 pin 10, IC 3 pin 7, IC 4 pin 10

IC 2, en 74LS244 kan undværes hvis man ikke skal læse fra port 3, det vil sige hvis man ikke har brug for at kunne læse hvilken baud rate generatoren er sat op til at køre. Det vil samtidig medføre at OR gaten (1/4 74LS32), som er forbundet til, R_D ikke skal anvendes.

Ellers skulle opstillingen være så simpel som det nu er muligt.

Komponent liste.

IC 1 = WD8116M-00, WD1943M-00, COM5016-00
IC 2 = 74LS244
IC 3 = 74LS32
IC 4 = 74LS273
XTAL = 5.0688 Mhz
R 1 = 1 K

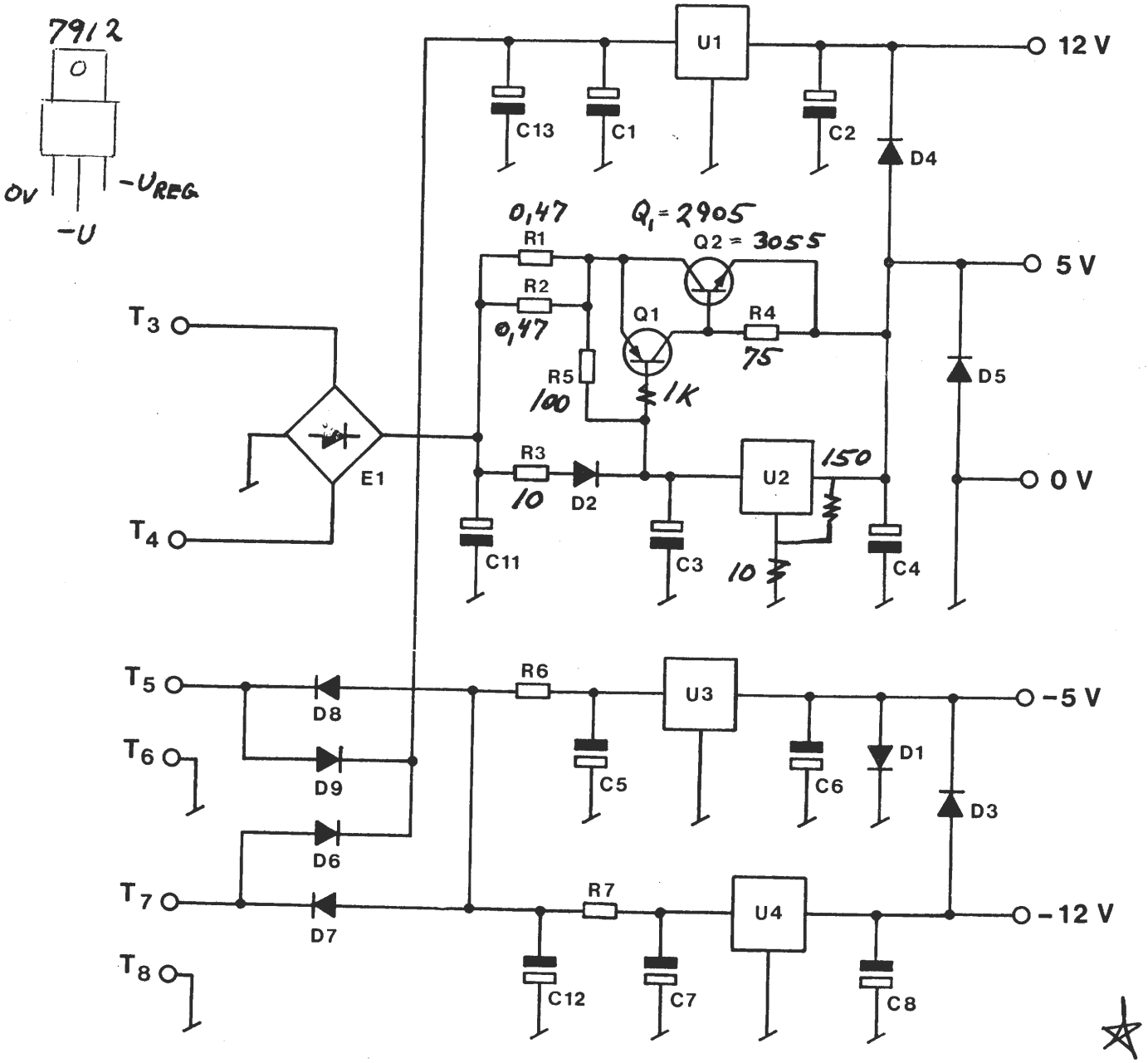
Forbedring af Customs Strømforsyning

Den meget udbredte og populære Strømforsyning fra Custom, som er leveret fra Piezodan i stort antal kan forbedres på to væsentlige punkter.

Udgangspændingen kan hæves fra 4,9 til 5,2v ved at hæve regulatorens nul-ben med 10 ohm samt forbinde det til udgangen med 150 ohm.

Kortslutnings sikkerheden er ikke som den burde være. Ved kortslutning af de 5V vil Q1=2905 ofte stå af. Ved at begrænse basis-strømmen til 10mA med en 1k modstand vil collektorstrømmen ikke overstige de max 1A som den kan tåle.

Lohse



REALCLOCK til NASCOM.

Dette lille letbyggede ur har nu været i brug hos mig, siden vi var i Silkeborg i maj måned. Det har kun tabt nogle få sekunder, og min computer har været slukket ret så mange gange. Årsagen er, at jeg bruger min JET-80 noget mere nu. Uret er opbygget over en 18 bens kreds fra Oeki. Den er ganske billig og overmåde nem at få til at virke. Uret har en standbye forbindelse indbygget, som gør at forbruget falder stærkt, når der bliver slukket for computeren. Det er blevet mig fortalt, at man kan erstatte akkumolatoren, der er af NICA typen, med en stort lyt i Farad størrelsen, men det er dog ikke forsøgt. Ideen med sænkningen af stand-bye strømmen laves ved hjælp af en transistor, der lukker, når man slukker for computeren. Jeg bruger en lille Deac akku, som dukkede op ved køb af noget surplus. Navnet på kredsen er: MSM 5832 RS. Den kan købes hos firmaet Peter Petersen i Silkeborg. Der skal bruges 13 modstande, een fast kondensator, en trimmer, samt et krystal på 32768 Hz. Krystallet kan fås hos Piezodan. Uret skal tilsluttes en Pio, men der er nok een eller anden, der laver dette om, så det kan arbejde direkte på bussen. Det er jo en stor fordel, hvis man bruger sin PIO som printerudgang. Jeg selv har ikke dette problem, da jeg har et I/O kort med 3 PIO'er. Jeg finder ikke, at det er nødvendigt med et print, idet der kan laves en nem lille opstilling på et lille stykke Veroboard.

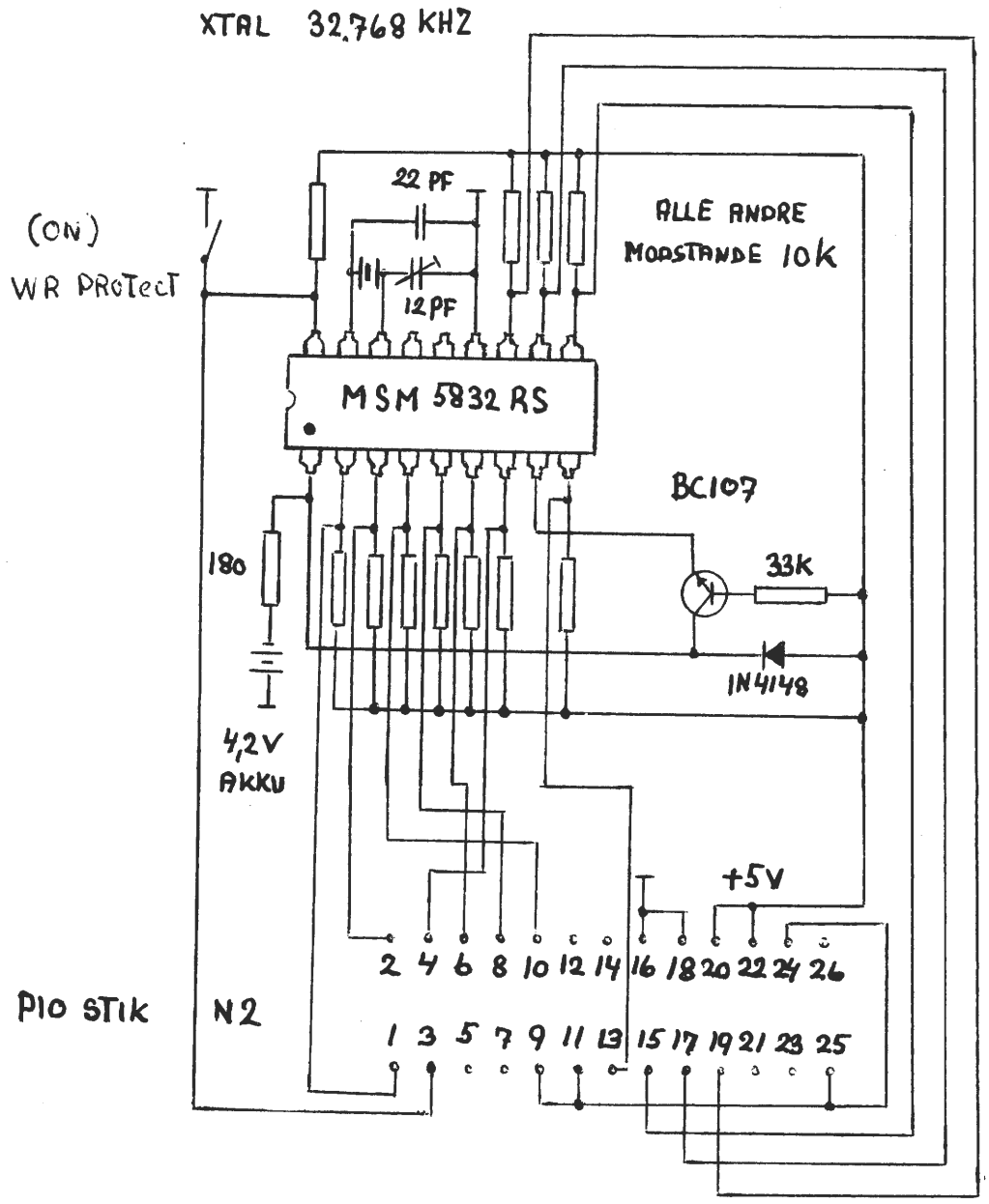
Det er et af vore medlemmer. Per Thomsen, der har lavet uret, efter jeg sendte ham kredsen, og det er også hans fortjeneste, at der er kommet en sourcetekst. Den er så stor, at den kommer i bladet delt over flere numre. Er en eller anden helt vild efter at komme igang, kan jeg godt lave en objektkode, men husk da at sende mig et bånd med angivelse af adressen, den skal ligge i.

Jeg har ventet lidt med artikler, idet jeg først skulle have printeren igang, men nu er det lykkedes, og jeg håber at alle nu er tilfredse ?

Artiklen er skrevet i WS, men jeg har ikke gjort så meget ud af skriftstørrelser og andre sjove ting. Det kommer, når jeg finder ud af mere.

O.H.

(Red. bemærkning: Da Ole og jeg nu er kompatible i diskformat, er dette stykke udskrevet på typehjulsprinter og ikke på Ole's Epson matrixprinter. Selvom det var mange gange bedre end på Ole's gamle brandudsalgs skrivemaskine!!)



1. DEL AF LISTNING PÅ SIDE 28.

De fleste cifre klarer opgaven ret problemfrit. Men nogle volder knuder:

$(1+1+1)! = 6$
$4+4-\sqrt{4} = 6$
$\sqrt[3]{8} + \sqrt[3]{8} + \sqrt[3]{8} = 6$

og så den værste:
$(\log 10 + \log 10 + \log 10)! = 6$

```
;                   IOfh.OV
;                   -------
;
;Overlay til filbehandling fra maskinkodeprog.
;herunder også Pascal.
;
;Overlayet kan oprette en fil, loade en fil
;og save nye data i en allerede eksisterende
;fil, samt verifisere en diskette.
;
;Skal der oprettes en ny fil med et navn der
;allerede eksisterer, vil den gamle fil blive
;slettet.
;Opstår der en fejl under disk access er der
;i overlayet en speciel fejlmelding der melder
;fejl og venter på ENTER herefter retuneres
;der til det kaldende program.
;
;Programmet har to entry point, en til Pascal
;og en til assembler programmer.
;
;
;Der er i overlayet indlagt mulighed for en
;blokflytning af data.
;En file skal her være opbygget af max 256
;datablokke på 4k.
;Den ønskede bloknr. skal ved kald overføres
;som den høje byte i (option) integeren.
;En blokfile der bliver udvidet skal altid
;være den sidste på disken.
;Oprettelse af filen foregår som ved en helt
;normal file, men længden skal være lidt
;under 4k.
;-----------------------------------------
;Eksempel på normalkald fra Pascal:
;
;CONST
;     N='Datafil';     (* Navn på fil 7 tegn *)
;     D=0;             (* Drive nummer *)
;VAR
;    FILEST:INTEGER;
;     DATA:ARRAY(.1..100.) OF REAL;
;     TEXT:ARRAY(.1..100.) OF STRING(.15.);
;    FILESL:INTEGER;
; .
; .
; PROCEDUTE disk(N:(STRING(.7.);R,D,B,E,O:INT);
; EXTERNAL $C900;        (* Overlay kald adr*)
; .
; B:=ADDR(FILEST);    (* Filstart *)
; E:=ADDR(FILESL);    (* Fileslut *)
; O:=0;                  (* Option:  *)
;                        (* 0=save ny *)
;                        (* 1=load    *)
;                        (* 2=save gl *)
;                        (* 4=save sp *)
;                        (* 5=load prog.DF*)
;                        (*13=load prog.IO*)
;                        (* 8=verify file*)
;                        (* 9=sidste file*)
;
; R:=1;                  (* Filenummer *)
; DISK(N,R,D,B,E,O);     (* Disk kald  *)
; .
; .
;Dette vil være et typisk kald fra Pascal
;Efter dette vil der på disken ligge en fil
;med navn:   Datafil1.DF:0  .
;Indholdet vil være: Data(.1..100.)
;                    Text(.1..100.)
;
;-----------------------------------------
;Kald fra assemblerprogram:
;
; BUFF: EQU     0CA00H          ;buf start
; DISK: EQU     BUFF+20         ;rutine start
;       .
;       .
;       LD      HL,STAK
;       LD      DE,BUFF
;       LD      BC,19
;       LDIR
;       CALL    DISK
;       .
;       .
; STAK: DB      'Datafil'       ;Filnavn
;       DS      1               ;Filnummer
;       DB      '.DF:0',0       ;ext.dr.fyld
; BEG:  DS      2               ;Filstart
; SLUT: DS      2               ;Filslut
; OPT:  DS      1               ;Option:
;                               ;0=save
;                               ;1=load
;                               ;2=save gl  ;4=save sp
;                               ;5=load prog.
;                               ;13=load pr.IO
;                               ;8=verify file
;                               ;9=sidste file
;Dette er kun een måde på kald.
;-----------------------------------------
;Option 0/1/2:
;
;Normal vil filen blive gemt med "LOAD"
;som startadr. og "EXEC" som filens længde.
;Dette har betydning når filen loades til et
;dataområde, idet der kun vil blive lagt data
;på plads i rigtig længde. Der bruges et buf-
;feområde i IOfh til den sidste sector der
;loades.
;-----------------------------------------
;Option 4:
;Er bit 2 sat i option vil "LOAD" stadig være
;startadr. men "EXEC" vil her også være start
;adr. Dette er nødvendigt når man f.eks. saver
;et program, istedet for et dataområde.
;-----------------------------------------
;Option 5:
;Er bit 0 og bit 2 sat vil der blive loadet
;den kaldte file, herefter vil der foregå et
;jump til "EXEC" adressen. Dette kan anvendes
;som chain fra et program til et andet. Exten-
;sion skal her ved pascal være ".DF".
;
;-----------------------------------------
;Option 13:
;Er bit 0,2,3 sat vil der foregå det samme
;som i option 5, men den kaldte file skal her
;altid være med extension ".IO".
;Anvendes til kald tilbage til master program.
;-----------------------------------------
;Option 8:
;Er bit 3 sat vil der blive foretaget en
```

```
;test om den spurgte filenr. befinder sig på
;disken. Er filen til stede, vil overlayet
;placere filenr. i (FLAG) ellers vil der i FLAG
;stå 00. (Addr.C8FFH)
;-----------------------------------------
;Option 9:
;Er bit 0 og 3 sat vil der blive undersøgt om
;den spurgte file er den sidste på disken.
;(FLAG) vil efter kald indeholde:
;  0 = findes ikke.
;  1 = er den sidste.
;  2 = er ikke den sidste.
;-----------------------------------------



        REFS    SYSEQU
        REF
WPS     EQU     0C92H
SEBUF   EQU     0CC00H
        ORG     OVAREA
        IDNT    $,0

;IO kald kommer her.
        DB      'IOfh'
        LD      HL,(S1FCB+FLDA)
        LD      DE,(S1FCB+FSEC)
        LD      A,(S1FCB+FNSC)
        LD      B,A
        LD      A,(MDRV)
        LD      C,A
        SCAL    ZDRD
        SCAL    ZCKER
        LD      HL,(S1FCB+FEXA)
LDADR   JP      (HL)


        ORG     OVAREA+100H-1
FLAG    DB      00H
;Her (C900H) kommer kald fra pascal.
;Rutinen henter overførte data fra procedure-
;kaldet.
;
```

```
PASINI  LD      HL,(WPS)
        DEC     HL
        DEC     HL
        LD      BC,6
        LD      DE,OPTION
        LDDR
        LD      DE,NAVN
        LD      A,(HL)
        SET     4,A
        SET     5,A
        LD      (DRIVE),A
        DEC     HL
        DEC     HL
        LD      A,(HL)
        LD      (NR),A
        LD      B,7
LOOP    DEC     HL
        LD      A,(HL)
        LD      (DE),A
        INC     DE
        DJNZ    LOOP
        JP      CONV

;Her er buffer for overførte data.
        ORG     OVAREA+200H

NAVN    DB      '_______'
NR      DB      '_.DF:'
DRIVE   DB      '0 '
START   DB      0,10H
SLUT    DB      0,11H
OPTION  DB      0,0

;Her omsættes og analyseres overførte data.

CONV    LD      DE,NAVN         ;Converter
        LD      HL,S1FCB        ;til S1FCB
        LD      B,0             ;
        SCAL    ZCFS            ;
        CALL    FEJL            ;
        RET     NZ              ;
        SCAL    ZRDIR           ;Læs lib.
        CALL    FEJL            ;Ret hvis fejl.
        RET     NZ              ;
```

```
        LD      A,(OPTION)      ;
        CP      8               ;
        JP      Z,VERIFY        ;Option 8
        CP      9               ;
        JP      Z,LAST          ;Option 9
        AND     00000011B       ;Afmask
        OR      A               ;Er bit 0,1=1?
        JR      NZ,LOAD         ;så load.

;Save så en ny file eller læg en blok til
;En blokfile skal være den sidste file på
;disken hvis den ikke er helt udbygget.
SAVE    LD      HL,0            ;Ny save.
        LD      (S1FCB+FSFL),HL ;Flag=0
        LD      HL,(NXTSEC)     ;Første frie
        LD      (S1FCB+FSEC),HL ;sec=start
        EX      DE,HL           ;
        PUSH    DE              ;Gem start
        LD      DE,(START)      ;Beregn files
        LD      HL,(SLUT)       ;længde
        SBC     HL,DE           ;
        PUSH    HL              ;HL=længe
        LD      L,H             ;
        INC     L               ;L=antal sec
        LD      H,0             ;
        LD      B,L             ;B=antal sect.
        LD      (S1FCB+FNSC),HL ;Antal sect.
        POP     HL              ;Hent længe
        LD      A,(OPTION)      ;
        BIT     2,A             ;
        JR      Z,SAVE0         ;
        LD      HL,(START)      ;FEXA=FLDA
SAVE0   LD      (S1FCB+FEXA),HL ;FEXA=længde
        LD      HL,(START)      ;
        LD      (S1FCB+FLDA),HL ;LOAD=start
        POP     DE              ;Start sect.

;       HL = ram adr.
;       DE = disk adr.
;        B = antal sect.
        SCAL    ZDWR            ;skriv til disk
        CALL    FEJL            ;
        RET     NZ              ;Ret ved fejl.
        LD      HL,S1FCB        ;
```

```
        CALL    ENTR            ;
        RET                     ;
ENTR    SCAL    ZENTER          ;Indskriv i bib
        RET     Z               ;Ret / ejfejl
        CP      31H             ;Er det dubb.
        RET     NZ              ;Nej så retur
        PUSH    HL              ;Det er ikke
        LD      HL,FSFL         ;blokmode, så
        ADD     HL,DE           ;slet gammel
        BIT     0,(HL)          ;file.
        LD      A,33H           ;
        JR      NZ,SKIP         ;
        SET     1,(HL)          ;
        POP     HL              ;
        JR      ENTR            ;
SKIP    POP     HL              ;
        RET                     ;

;Load en file eller en blok
;     HL = S1FCB
;     DE = Øverf. buffer +1
;      B = 0
;     OPTION = XXXXXXYY  en af Y=1
LOAD    LD      B,00110000B     ;
        PUSH    HL              ;
        LD      A,(OPTION)      ;
        BIT     3,A             ;Ret til IO
        JR      Z,LOAD01        ;file ved opt.
        LD      HL,'I'+'O'*256  ;13.
        LD      (S1FCB+FEXT),HL ;
LOAD01  POP     HL              ;
        SCAL    ZLOOK           ;Søg i lib
        CALL    FEJL            ;og kopier til
        RET     NZ              ;til S1FCB.
                                ;ret hvis file
                                ;ikke er der.
        LD      DE,(S1FCB+FSEC) ;Start sect.
        LD      A,(S1FCB+FNSC)  ;Antal sect.
        LD      HL,(START)      ;RAM start
        LD      B,A             ;Antal til B
        LD      A,(OPTION)      ;
        AND     00000011B       ;
        CP      1               ;Er bit0=1,så
        JR      Z,LOAD1         ;option 1 load
        CP      2               ;Er bit1=1,så
        JR      Z,GLSAVE        ;opt. 2 glsave
        RET                     ;Fejl.........
LOAD1   LD      A,(OPTION)      ;
        BIT     2,A             ;Er bit2=1,så
        JR      NZ,LOAD5        ;opt. 5

;Normal load eller blokload

        CALL    BLOK            ;Jurster blokmo

;       HL = ram adr.
;       DE = disk adr.
;       B  = antal sect.

        DEC     B               ;-Er der mere
        JR      Z,LOAD2         ;end en sect.?
        SCAL    ZDRD            ;-Ja, så load
        CALL    FEJL            ;dem minus 1.
        RET     NZ              ;
LOAD11  INC     H               ;Find sidste
        INC     DE              ;sect. og tilh.
        DJNZ    LOAD11          ;RAM adr.
LOAD2   PUSH    HL              ;Gem RAM adr.
        LD      HL,SEBUF        ;HL=buff.adr.
        LD      B,1             ;B= 1 sect.
        SCAL    ZDRD            ;Indlæs til
        CALL    FEJL            ;buffer.
        JR      NZ,LOAD4        ;Ret ved fejl.
        LD      HL,(S1FCB+FEXA) ;Hent længde.
        LD      B,0             ;
        LD      A,L             ;A= rest <100H
        OR      A               ;Er det 0
        JR      NZ,LOAD3        ;Nej
        INC     B               ;Ja, B=1
LOAD3   LD      C,L             ;C= rest.
        POP     HL              ;Hent RAM adr.
        LD      DE,SEBUF        ;DE=buff adr.
        EX      DE,HL           ;
        LDIR                    ;Flyt til RAM.
        RET                     ;Retur
LOAD4   POP     HL              ;Jurster stak
        RET                     ;Retur (fejl)

;Load ved option 5 og 13
LOAD5   LD      HL,(S1FCB+FLDA) ;Hent load adr.
        SCAL    ZDRD            ;
        CALL    FEJL            ;
        LD      HL,(S1FCB+FEXA) ;Hent exec. adr
        JP      (HL)            ;Hop til exec.
        RET                     ;
;Save i gammel file option 2 eller bloksave,blot
;ikke i første blok
;       HL = ram adr.
;       DE = disk adr.
;        B = antal sect.
GLSAVE  CALL    BLOK            ;Jurster blokmo
        SCAL    ZDWR            ;Skriv i gl.
        CALL    FEJL            ;file.
        LD      A,(OPTION)      ;Er det blokmo
        AND     11110000B       ;
        OR      A               ;
        RET     Z               ;Nej så ret.
        LD      HL,10H          ;
        ADD     HL,DE           ;
        EX      DE,HL           ;DE=slutadr.
        OR      A               ;Reset C
        LD      HL,(NXTSEC)     ;

        SBC     HL,DE           ;Er ny blok >
        RET     NC              ;nej, så ret.
;jurster S1FCB og lib
        LD      (NXTSEC),DE     ;jurster lib
        LD      HL,S1FCB        ;
        LD      B,0             ;Flyt ikke!
        SCAL    ZLOOK           ;Find FCB adr
        CALL    FEJL            ;
        RET     NZ              ;
        LD      HL,FNSC         ;Peg på FNSC
        ADD     HL,DE           ;i FCB.
        LD      A,(OPTION)      ;A=files sect.
        AND     11110000B       ;antal,gem det
        LD      (HL),A          ;i FCB
        SCAL    ZWDIR           ;
        RET                     ;Retur

;Undersøger om det er blokmode,og beregner
;den ønskede disk adr. til DE
```

```
;Verify vil teste om filen er til stede på
;disken. Er den det vil (NR) blive skrevet i
;FLAG så den senere kan læses af det kalden-
;de program.
;
VERIFY  LD      B,00110000B     ;Set flag
        SCAL    ZLOOK           ;Søg i lib
        OR      A               ;Er filen der
        LD      A,0             ;
        LD      (FLAG),A        ;0 i FLAG
        RET     NZ              ;Nej så retur
        LD      A,(NR)          ;Ja, så skriv
        LD      (FLAG),A        ;nummer i FLAG
        RET                     ;Retur

;Option 9

LAST    LD      B,00110000B     ;
        SCAL    ZLOOK           ;Søg i lib
        OR      A               ;
        LD      A,0             ;
        LD      (FLAG),A        ;0=findes ej
        RET     NZ              ;
        LD      HL,(S1FCB+FSEC) ;
        LD      DE,(S1FCB+FNSC) ;
        ADD     HL,DE           ;fileslut
        LD      DE,(NXTSEC)     ;lib sidste
        OR      A               ;res C
        SBC     HL,DE           ;Er det sidste
        LD      A,1             ;file.
        LD      (FLAG),A        ;1=ok
        RET     Z               ;Det er sidste
        INC     A               ;
        LD      (FLAG),A        ;2=ej sidste.
        RET                     ;

END
```

```
;Og sætter B=16 sect. 4 Kbyte
BLOK    LD      A,(OPTION)      ;Hent kode
        AND     11110000B       ;Afmask
        OR      A               ;Er det blokmo.
        RET     Z               ;Nej, så retur
        PUSH    HL              ;Gem RAM adr.
        LD      H,0             ;
        LD      L,A             ;
        ADD     HL,DE           ;
        LD      DE,10H          ;
        OR      A               ;
        SBC     HL,DE           ;
        EX      DE,HL           ;
        POP     HL              ;
        LD      B,10H           ;
        RET


;Denne rutine er i stedet for PolySys'es egen
;rutine, idet man herved undgår at komme til-
;bage til monitoren ved fejl.
FEJL    OR      A
        RET     Z
        PUSH    AF
        PUSH    HL
        PUSH    DE
        PUSH    BC
        PUSH    AF
        RST     PRS
        DB      CR
        DB      'Disk access error =',0
        POP     AF
        SCAL    ZB2HEX
        RST     PRS
        DB      CR,'<Tast  ENTER>',0
        SCAL    ZBLINK
        POP     BC
        POP     DE
        POP     HL
        POP     AF
        RET

;Option 8
```

424 tegn på Nascom tastaturet.

Nascom'en har i alt 256 forskellige tegn, og de fleste regner vel med, at man fra tastaturet kan indtaste netop disse 256 forskellige tegn. Hverken mere eller mindre.

Der er imidlertid mange flere muligheder, og det kræver ikke den store datakraft at regne ud, hvor mange det i virkeligheden drejer sig om. Der er nemlig 53 taster foruden SHIFT-, CTRL- og GRAPH-tasterne. Hvis man kombinere disse 53 taster med SHIFT, CTRL eller GRAPH, giver det 8*53=424 indtastningsmuligheder.

At der således er mange flere indtastningsmuligheder, end der er tegn, viser sig ved, at mange tegn kan indtastes på flere måder - en normal måde og en mere utraditionel. F.eks. vil CTRL/2 give tegnet 'r'.

De mange ekstra indtastningsmuligheder kan man udnytte til specielle kontrolfunktioner i datamaten. Jeg har f.eks. udnyttet to af dem til at styre de ledige bits i output port 0 (bit 2 og bit 5) hvorved jeg kan skifte mellem to forskellige tegnsæt (anbring en 2732 i tegnsoklen og forbind det ekstra adresseben med en af disse bits).

For at kunne gøre dette har det været nødvendigt at udvide KBD-rutinen. Umiddelbart før KSE kaldes, indsættes denne stump kode til at fange specialindtastningen (i dette tilfælde CTRL/CH for bit 2 og CTRL/SHIFT/CH for bit 5):

```
        LD   C,A
        AND  7FH          ;fjern SHIFT
        CP   0EH          ;CH-tast nedtrykket?
        JR   NZ,KSC4B     ;hop hvis ikke
        BIT  3,(HL)       ;CRTL nedtrykket?
        JR   Z,KSC4B      ;hop hvis ikke
        LD   A,00000100B;sæt bit 2
        BIT  7,C          ;SHIFT nedtrykket?
        JR   Z,KSC4A
        LD   A,00100000B;så sæt i stedet bit 5
KSC4A:  DEC  HL           ;HL=PORT0
        XOR  (HL)         ;flip bit i PORT0
        LD   (HL),A
        OUT  (0),A        ;og skriv den ud
        RET

KSC4B:  LD   A,C
        RCAL  KSE
```

Den ekstra plads, som denne kode kræver, kan skaffes på flere måder (se min artikel i Z80-nyt nr.8 1983).
Hvis man ønsker at ændre på dekodningen af tastaturet, f.eks. hvis man har danske tegn og ønsker ÆØÅ anbragt korrekt på tastaturet (se en udmærket artikel af Søren H. Nielsen i NASCOM-NYT nr. 9 1982), så løber man ind i de samme vanskeligheder som Nascom-folkene og tilsyneladende også Søren har været ude i.

På trods af de mange indtastningsmuligheder er der ikke nok taster på tastaturet! Hvis man ønsker at kunne indtaste samtlige skrivbare ASCII-tegn ved tryk på en enkelt tast eventuelt i kombination med SHIFT-tasten, har man brug for 96/2=48 taster og da 7 af tasterne bruges til kontrol-tegn (og dette bør nok ikke ændres), har man kun 46 til disposition.

Man er altså nødt til at udelade nogle tegn, hvilket sker ved at fjerne dem fra KBD-tabellen. Det er dog ikke lige meget hvilke tegn, man fjerner, for de manglende skal helst kunne indtastes v.hj.a. CRTL. Som hovedregel betyder dette, at man kun bør fjerne tegn med ASCII-værdier mellem 20H og 3DH. Hvilken en af disse man vælger er delvis en smagssag. Selv har jeg valgt at lade det være £-tegnet, som så kan indtastes som CTRL/C.

For at få den fulde forståelse af KBD-rutinen i NASSYS, bør man læse en artikel i INMC80 NEWS 1981 nr.5 (forløberen for 80-BUS NEWS).

Svend Daugaard Pedersen (333)

'MICROMOUSE' København 1984.

Mikromus-konkurrencen, omtalt i Z80 NYT, nr. 5, Juni 1984, er som man vil huske en international konkurrence i hvilken "små" robot-mus skal prøve, på hurtigste tid, at finde ind til centrum i en ca. 3x3m stor labyrint.

Prøverne blev afholdt i Lyngby/DTH bygn. 116, med de indledende runder i forhallen mandag 27/8, der var tilmeldt 15 mus, men kun 11 klarede at komme til finalen om tirsdagen i auditoriet.

Det var et meget interesseret publikum, der overværede de mange, meget forskellige mus's forsøg på at trænge ind i labyrinten, nogle opgav straks p.gr.af 'maskinskade', andre havde fået 'bare pletter' i memory, -det kan jo ske for den bedste!-, én ville absolut igennem en væg, én over en væg og nogle var bare for lang-somme. De 'MORSOMME' mus såvel som de hurtigste blev belønnet af publikum med kraftige bifald!

Den helt suveræne vinder blev "Enterprise" fra England med en tid på kun 27 sek. (den accelererede på de lange, lige strækninger op til ca. 3m/sek. Det gav en præmie: 1 fly-billet, VIP med ophold i en uge betalt, til Japan. Præmieoverrækkelse foregik ved en recep-tion den næste dag på Kbh's Rådhus, præmien var givet af Japan Science Foundation. Der fulgte også en messing-"ost"/trofæ med og naturligvis et certificat.

Nr. 2 blev "Manu" fra Finland med 41 sek. nr. 4 blev "Tellu" også fra Finland, det gav det finske hold 2 fly-billetter til Japan.

Tredjepladsen gik til "T5" (56 sek.) England, det blev belønnet med 1 fly-billet til Japan + en 'lomme-computer'.

Yderligere 1 fly-billet blev vundet af Ralf Hinkel, Tyskland.

De yngste deltagere var en gruppe skoleelever fra England, deres 'mus' nægtede at gå længere end halvt ind i labyrinten, men som et plaster på såret over skuffelsen fik de en lomme-computer og lommeregnere.

Af det hidtil fremførte kunne man få det indtryk at det hele nok er noget "japansk" noget, det er dog ikke tilfældet, men i 1985 skal der afholdes en teknologisk 'verdens-udstilling' EXPO 1985 i den japanske by Tsukuba, her vil 'Japan Micromouse Association' afholde verdens-finale i august 1985 - det er grunden til at de to organisationer, JSF og JMA har givet rejserne.

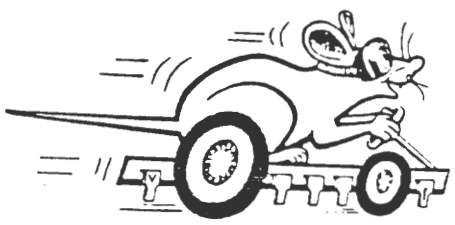
Lomme-computerne og lommeregnerne er givet af EUROMICRO.

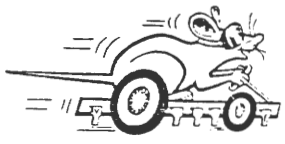
I forbindelse med EUROMICRO 85 i Bruxelles vil der også være en konkurrence for mikro-mus, samt en helt ny type, robotter der spiller bordtennis!

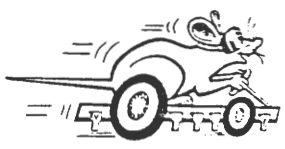
- Nej, desværre, der var ingen danske deltagere denne gang!

I Z80-Brugergruppen må der da være nogen, som har konstruktive talenter 'bløde & hårde'- det kan nås endnu til 1985.

Evt. interesserede medl. kan kontakte Viggo Jørgensen / 536.







Programbiblioteket under CP/M

De fejl, jeg havde med min BIOS, er nu rettet. Det betyder, at jeg nu kan læse/skrive bl.a. NCR DM 5, James og Galaxy på begge sider, det vil sige op til henholdsvis 312K, 788K og 788K. Yderligere et format, som nok bliver meget populært, er kommet til nemlig JET-80 (800K).

Hvis jeg skal lave dit format, behøver jeg følgende oplysninger, hvis du ikke følger brugsanvisningen fra et tidligere Z80 NYT:

Spor pr. disk	Sectorer pr. spor	Bytes pr. sector
Blok størrelsen	Antal dir. entry	Antal systemspor
Sector screw +)	Density (S eller D)	Kompl. data på disk +)
Antal sider	Skrivemåde på disk +)	1. sectornummer

Track pr. inch (48 / 96 / 100) og gerne GAB 3 (formateringsøjemed)

+)

Sector screw — Denne faktor kan have to tilstande: Enten screw'es der ved formateringen, så skal screw = 0 eller også screw'es der i hardware så kan screw = 0 til sector-1.

Kompl. data — Normalt gemmes data i samme form på disk og i ram, men nogle maskiner (bl.a SuperBrain) gemmer data på disk i komplementeret form!

Skrivemåde — Dette er kun aktuelt, hvis drevet kan skrive på begge sider. Man kan gøre det på tre forskellige måder: 1) Cylinder mode, hvor BIOS'en skifter side, når halvdelen af antallet af sectorer er skrevet/læst på den ene side. 2) Odd/even, hvor ulige spor skrives på side 0 og ulige spor på side 1 - eller omvendt! Herved fås der det dobbelte antal spor af diskdrevets. 3) Disk side, hvor hele den ene side skrives først. Når det ønskede spor er over sporantallet subtraheres sporantallet og der skiftes side.

Eksempler på disse tre diskformater er: 1) Nascom med Gemini BIOS. 2) NewBrain 80 spor og JET-80. 3) Gemini Galaxy (800 K), med dette format bliver single side og dobbelt side kompatible med de første 400 K.

Asbjørn Lind

NYHED

Nye dansk fremstillede Winchester disk-system har set dagens lys. De er specielt fremstillet til brug sammen med IBM PC/TX eller andre 16-bits maskiner. Men skulle det mon ikke være muligt, at tilpasse dem til 8-bits maskiner også ? Drevene har egen strømforsyning (switch-mode) og kan lagre enten 10 Mb eller 20 Mb. Yderligere oplysninger fra IBIS (01 87 38 84) Startpris er opgivet til 16.000 kr. + moms.

Annonce Annonce Annonce Annonce Annonce Annonce Annonce

1 stk. MIDICOS med 6 bånd sælges. Drive-interfacet er indbygget i en lækker MK-kasse. På båndene ligger diverse programmer, som Pascal og Polysys 4.x Pris kun **2000.00 kr.** Henvendelse til Michael Brouer, Ørnevang 62 2.th, 3450 Allerød. 02 275031 (18.30 - 19.30).

Nascom 3 (Nascom 2 i integreret kasse med strømforsyning): 5000 kr. Floppy kontroler kort (GM809): 2.000 kr. Kasse til 2 alm. / 4 slim-line diskettestationer med strømforsyning: 500.00 kr. Et Shugart slimline drive til 800 Kb: 2.500 Kr. PolyDos 2.0 sælges for 1200.00 kr. Et stk. DP 8480 (Star printer) matrixprinter sælges for 2.800 kr. Rabat gives ved samlet salg. Harry Timm (06) 86 73 99

2 stk. CODEX 9600 BPS modems (smalbånds 300-3.000 Hz) ønskes byttet med et IVC kort samt en CP/M til Nascom 2. Memorex 1377 skærmterminal 24x80 karakterer, 1.1 MBPS seriel I/F. sælges for 3.500 kr. (er som nyt) Henvendelse til P. Munch Andersen Hegnstoften 8, 2630 Taastrup (02) 52 80 10

****

Div. tilbud (priser inkl. moms, ekskl. forsendelse)

Nascom-1 tastatur + 10 løse taster m. top kr. 450
Ny Ball Inc. 12" monitor wire frame, til 15 Volt, kr. 500
Skinner for Nascom/Gemini kortholder, pr. par kr. 10
10 stk. Agfa digital cassette PC15 kr. 150
TTL IC 74LS245 kr. 25
Power variabel regulator IC (4-pin TO-3):
8 Amp. 4 - 16 Volt kr. 435
5 Amp. 4 - 30 Volt kr. 200
3 Amp. 4 - 35 Volt kr. 175
1,5 Amp. 4 - 30 Volt kr. 80
Ellyt 10.000 uF aksial 16 V kr. 15

Custom Electronics, tlf. 02-872282

****

Efterlysninger Efterlysninger Efterlysninger Efterlysninger

Er der nogle kloge medlemmer med forstand på Midicos, som kan fortælle, hvad man gør, når en fejl på båndet bevirker, at Midicos'en ikke kan komme videre?

Hvis program 2 længere fremme på båndet fungerer, men der er læsefejl undervejs i program 1, stopper Midicos og udskriver "read error" - brandirriterende, og umuligt at få den til at gå videre - i hvert fald uden et eller andet trick, som jeg udbeder mig!

Hvis fejlen ligger i det ønskede program, ville det ofte også være en fordel bare at få en fejlagtig version ind - så kan man da altid forsøge at redde stumperne - kan det problem løses på eventuelle samme vis ?

Venlig hilsen Henrik Dyhr.

☆

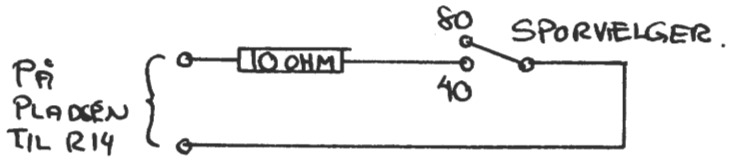
Alteration Guide til Hisoft PASCAL 4T. I implementation notes til NewBrain udgaven af HP4T er nævnt, at NewBrainens skærmeditor ikke understøttes, men det er muligt ved hjælp af Alteration Guide at læse hvordan andre end HP4T's standard editor kan anvendes. Skulle nogen være i besidelse af en sådan guide eller iøvrigt vide hvordan man gør, er jeg meget interesseret i at høre nærmere. Firmaet leverer nemlig ikke mere den omtalte guide!

Niels Jensen, Klintøs Allé 5, 9700 Brønderslev, tlf.: (08) 82 43 72.

☆

Tip Tip Tip Tip Tip Tip Tip

Til Teac 80 spor drive 55F, kan man ved at anbringe en 10 ohms modstand i serie med en kontakt på stedet, hvor der står "R14", få drevet til at læse både 40 spor og 80 spor (På 55E er der en rytter, der skal/kan flyttes). Se skitsen:



☆

Medlemsmøde Medlemsmøde Medlemsmøde

På medlemsmødet den 11/10 1984 (torsdag) fortæller René Hansen om 16-bits maskiner. Der fremvises bl.a. IBM PC og Stearns maskiner. Stedet er det sædvanlige: RECU, Vermundsgade 5 (audi. 18 1.sal) 2100 Kbh. Ø

☆

```
;TEST:
        ORG 08000H
;       MEM$
TEST:   CALL WARM
        JR   TEST
CR:     EQU  000DH
RIN:    EQU  00CFH
CRT:    EQU  0BDFH
CRLF:   EQU  006AH
DISP:   EQU  0028H
DATA:   EQU  0018H      ;DATAPORTE TIL UR
DATB:   EQU  0019H      ;ADD OG KONTROLPORT
CONA:   EQU  001AH      ;TIL UR
CONB:   EQU  001BH

        ORG  0A000H
;       MEM$

;       INITRUTINE
STAR:   JP   BEG
VARM:   JR   CAL1
BEG:    XOR  A
        LD   (CAL),A
        JR   INIT
CAL1:   LD   A,"C"      ;HUSKER OM DET
        LD   (CAL),A    ;    ER ET CALL
INIT:   LD   A,4FH          ;OUTPUT MODE
        OUT  (CONA),A
        LD   A,0F
        OUT  (CONB),A
        LD   A,3AH
        LD   (TIM+2),A
        LD   (MIN+2),A
        LD   (SEC+2),A
        LD   A,2DH
        LD   (DAG+2),A
        LD   (MAA+2),A
        LD   A,2FH
        LD   (AAR)+2),A
        LD   A,(CAL)        ;HUSKER OM DET
        CP   "C"            ;  ER ET CALL
        JR   Z,READ
START:  RST  28H
        DB   "VIL DU JUSTERE URET?."
        DB   CR,00H
        RST  8H
        CP   "J"
        JP   NZ,READ
        JP   INDP
;DENNE RUTINE LÆSER URET TIL RAMBUFFER

READ:   LD   A,10H
        OUT  (DATB),A
        CALL VENT
        IN   A,(DATA)
        ADD  A,30H
        LD   (SEC+1),A
        LD   A,11H
        OUT  (DATB),A
        CALL VENT
        IN   A,(DATA)
        ADD  A,30H
        LD   (SEC),A
        LD   A,12H
        OUT  (DATB),A
        CALL VENT
        IN   A,(DATA)
        ADD  A,30H
        LD   (MIN+1),A
        LD   A,13H
        OUT  (DATB),A
        IN   A,(DATA)
        ADD  A,30H
        LD   (MIN),A
        LD   A,14H
        OUT  (DATB),A
        CALL VENT
        IN   A,(DATA)
        ADD  A,30H
        LD   (TIM+1),A
        LD   A,15H
        OUT  (DATB),A
        CALL VENT
        IN   A,(DATA)
        RES  2,A        ;CLEAR 12 TIMER
        RES  2,A
        RES  3,A
        ADD  A,30H
        LD   (TIM),A
        LD   A,16H
        OUT  (DATB),A
        CALL VENT
        IN   A,(DATA)
        ADD  A,30H
        LD   (UGD),A    ;UGEDAG
        LD   A,17H
        OUT  (DATB),A
        CALL VENT
        IN   A,(DATA)
        ADD  A,30H
        LD   (DAG+1),A ;DATO 10
        LD   A,18H
        OUT  (DATB),A
        CALL VENT
        IN   A,(DATA)
        ADD  A,30H
        LD   (DAG),A    ;DAG 1
        LD   A,19H
        OUT  (DATB),A
        CALL VENT
        IN   A,(DATA)
        ADD  A,30H
        LD   (MAA+1),A ;MAANED 1
        LD   A,1AH
```

FORTSETTES INR 9.

```
**************************
*                        *
*   UR PROGRAM TIL       *
*   CHIP MSM5832RS       *
*   AF PER THOMSEN       *
*   FOR PIEZODAN.        *
*                        *
**************************
```