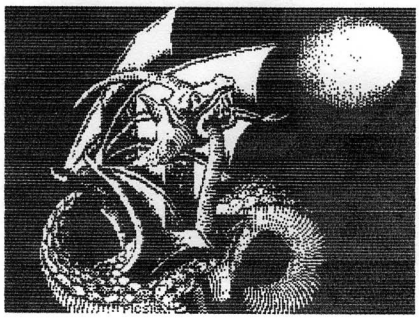
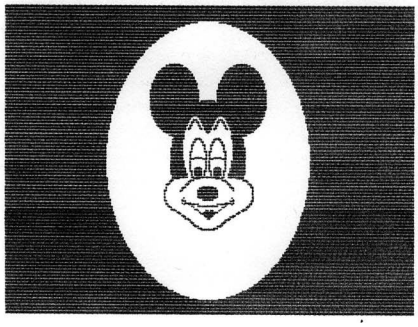
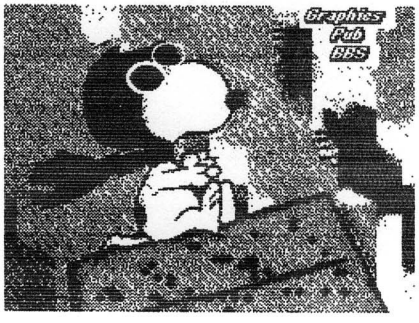
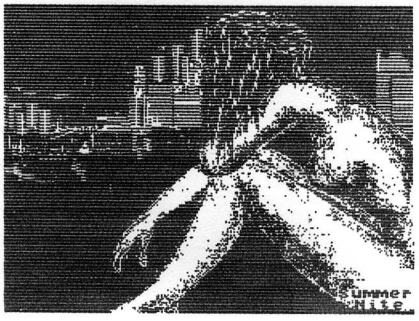
PROGRAM BITEN 91-3









ISSN 0281-1146

COLOR EDITOR

(från PB 84-5.18)
av H.Reitinger, Österrike

```
1 ! COLOR EDITOR foer att bl
anda tva valfria faerger med
 joyst eller tangenter
2 ! Med haelsningar fraan E.
H.REITINGER, WIEN, OESTERRIK
E
3 ! TI99-Journal-Klub, A-115
0 Wein Felberstrasse 24/26
4 ! Justerat av Craig Miller
, Millers Graphics
10 CALL SCREEN(16):: CALL CL
EAR
20 M$="55AA55AA55AA55AA" ::
A=122
30 CALL MAGNIFY(2):: CALL CH
AR(64,RPT$("F",16),34,"FF818
1FFFFFF",128,"FFFFFFFFFFFFFF
",73,M$):: CALL COLOR(3,16,2
,4,16,2,6,1,1,5,2,1)
40 CALL VCHAR(1,27,64,192)::
 CALL HCHAR(23,1,64,162):: H
=1
50 G=-2 :: FOR I=3 TO 16 ::
CALL SPRITE(#I,64,I,(G+I)*12
,230):: NEXT I :: CALL SPRIT
E(#2,34,16,5,230)!#16,128,16
,17,230)
60 CALL SPRITE(#1,42,2,A,231
)
70 FOR S=4 TO 22 :: CALL HCH
AR(S,3,73,24):: NEXT S
80 CALL JOYST(1,X,Y):: ON (S
GN(Y)+2)GOTO 90,130,110
90 A=A+12 :: IF A>170 THEN A
=2
100 CALL LOCATE(#1,A,231)::
GOTO 130
110 A=A-12 :: IF A<0 THEN A=
170
120 CALL LOCATE(#1,A,231)
130 CALL KEY(1,K,S):: IF S=0
 THEN 80
140 IF K=5 THEN 110 :: IF K+
1=1 THEN 90
145 F=INT(A/12+2):: CALL SOU
ND(200,660,2):: GOTO 180 ! D
ENNA RAD ERSAETTER RADERNA 1
50, 160 OCH 170/Miller
150 !FOR F=2 TO 16 :: CALL C
OINC(#1,#F<3,C):: IF C THEN
CALL SOUND(200,660,2):: GOTO
 180
160 !NEXT F
170 !GOTO 80
180 CALL COLOR(6,F,H):: DISP
LAY AT(24,9)SIZE(7):USING "#
# ## ":F,H :: H=F :: GOTO 80
```

■

Redaktör: Jan Alexandersson
Utmanarredaktör: Anders Persson
TI-74 redaktör:Lars Herold Andersen
Programbankir: Börje Häll

Föreningens adress:
Föreningen Programbiten
c/o Schibler
Wahlbergsgatan 9 NB
S-121 46 JOHANNESHOV
Sverige

Postgiro 19 83 00-6
Medlemsavgiften för 1991 är 120:-

Datainspektionens licensnummer:
82100488

Annonser, insatta av enskild medlem (ej företag), som gäller försäljning av moduler eller andra tillbehör i enstaka exemplar är gratis.

Övriga annonser kostar 200 kr för hel sida. För lösblad (kopieras av annonsören) som skickas med tidningen gäller 200 kr per blad. Föreningen förbehåller sig rätten att avböja annonser som ej hör ihop med föreningens verksamhet eller ej på ett seriöst sätt gäller försäljning av originalexemplar av program.

Föreningens tillbehörsförsäljning:
Följande tillbehör finns att köpa genom att motsvarande belopp insätts på postgiro 19 83 00-6 (porto ingår)

Användartips med Mini Memory	20:-
Nittinian T-tröja	40:-
99er mag. 12/82, 1-5,7-9/83(st)	40:-
Nittinian årgång 1983	50:-
Programbiten årgång 84-89(styck)	50:-
1990	80:-
TI-Forth manual	100:-
Hel diskett ur programbanken(st)	30:-

Enstaka program 5:- st + startkostnad 15 kr per skiva eller kassett (1 program=20kr, 3 program=30 kr). Se listor i PB89-3 och PB90-4.

PROGRAM CONVERTER

```
100 ! PROGRAM CONVERTER
110 ! BASED ON A PROGRAM BY
      JIM PETERSON
      TIGERCLUB SOFTWARE
120 ! REVISED BY JIM SWEDLOW
130 ! VERSION XB.1.1
140 ! 01 NOV 86
150 !
160 DATA @,{,&,},^,~,*,|
170 @=1 :: DIM C$(8):: FOR I
=@ TO 8 :: READ C$(I):: NEXT
 I :: O$="OLD" :: N$="NEW" :
: GOTO 180 :: W,A,B,D,J,K=L
:: A$=T$ :: !@P-
180 DISPLAY AT(@,6)ERASE ALL
:"PROGRAM CONVERTER 1.1":"Th
is takes a program saved  as
 LIST ""DSK1.OLD"" and sets
it up for use by TI Writer"
190 DISPLAY AT(6,@):"or for
listing on a printer in comp
ressed type.": :"Output can
be various widthsand can be
used by the"
200 DISPLAY AT(11,@):"Text F
ormatter.": : :"Use with Tex
t <F>ormatter   or <E>ditor
or <P>rinter?  F"
210 ACCEPT AT(15,28)SIZE(-@)
VALIDATE("EFP")BEEP:T$ :: I=
T$="P" :: IF W=0 THEN W=28-1
08*I ELSE IF W>80 AND I=0 TH
EN W=28
220 DISPLAY AT(17,@):"Column
 width:";TAB(26+I);W: :"Old
file:  DSK1.";O$ :: IF I THE
N DISPLAY AT(20,@)ELSE DISPL
AY AT(20,@):"New file:  DSK1
230 IF A=0 THEN 250 ELSE ACC
EPT AT(17,27+I)SIZE(-2+I)VAL
IDATE(DIGIT)BEEP:W :: IF W<1
0 OR W>80-56*I THEN 230
240 O$="" :: A=19 :: GOSUB 4
20 :: O$=A$ :: IF I=0 THEN A
=20 :: GOSUB 420 :: N$=A$
250 A=@ :: DISPLAY AT(22,@):
"Information Correct?
Y" :: ACCEPT AT(22,28)SIZE(-
@)VALIDATE("YN")BEEP:A$ :: I
F A$="N" THEN DISPLAY AT(22,
@):: GOTO 210
260 DISPLAY AT(22,@):"Initia
lizing . . ." :: OPEN #@:"DS
K1."&O$,DISPLAY ,VARIABLE 80
,INPUT :: B=-@ :: O$="" :: I
F I THEN A=W>80 :: OPEN #2:"
PIO",VARIABLE 80-56*A :: IF
A THEN PRINT #2:CHR$(15):: G
OTO 290 ELSE 290
270 A$=N$ :: B=B+@ :: IF B T
HEN A$=N$&STR$(B)
280 OPEN #2:"DSK1."&A$,DISPL
AY ,VARIABLE 80,OUTPUT :: A=
0 :: A$="" :: IF T$="F" THEN
 A$=CHR$(13):: PRINT #2:A$:"
.NF;NA";A$:".TL 126:94";A$:"
.TL 123:64";A$:".TL 125:38";
A$:".TL 124:42";A$:A$ :: A$=
""
290 IF O$="" THEN LINPUT #@:
A$ :: IF A$="" THEN IF EOF(@
)THEN 390 ELSE 290
300 A$=A$&O$ :: O$="" :: D=L
EN(A$):: IF EOF(@)OR(D<>80 A
ND D<>160)THEN 340
310 LINPUT #@:O$ :: IF O$=""
 THEN IF EOF(@)THEN 340 ELSE
 310 ELSE J=POS(O$," ",@)-@
:: IF J<@ OR J>5 THEN 300
320 FOR I=@ TO J :: K=ASC(SE
G$(O$,I,@)):: IF K<49+(I>@)O
R K>57 THEN 300
330 NEXT I :: J=VAL(SEG$(O$,
@,J)):: IF J<=L OR J>32767 T
HEN 300
340 A=A+D :: I=POS(A$," ",@)
:: L=VAL(SEG$(A$,@,I-@)):: D
ISPLAY AT(22,@):"Converting
line";L :: IF T$<>"F" THEN 3
80
350 FOR J=@ TO 7 STEP 2 :: I
=@
360 I=POS(A$,C$(J),I):: IF I
 THEN A$=SEG$(A$,@,I-@)&C$(J
+@)&SEG$(A$,I+@,255):: GOTO
360
380 FOR I=@ TO D STEP W :: P
RINT #2:"   ";SEG$(A$,I,W)::
 NEXT I :: IF A>27000 AND T$
<>"P" THEN CLOSE #2 :: GOTO
270 ELSE IF EOF(@)=0 OR O$<>
"" THEN A$="" :: GOTO 290
390 IF T$="P" THEN PRINT #2:
CHR$(18)
400 CLOSE #@ :: CLOSE #2 ::
DISPLAY AT(22,@):"CONVERSION
 COMPLETED" :: IF B=0 OR T$=
"P" THEN STOP ELSE DISPLAY A
T(23,@):"NEW FILE NAMES:":N$
410 FOR I=@ TO B :: PRINT N$
&STR$(I):: NEXT I :: STOP
420 ACCEPT AT(A,17)SIZE(-9)B
EEP:A$ :: IF A$="" OR A$=O$
OR A$=SEG$(O$,@,LEN(O$)+(O$<
>""))THEN 420 ELSE RETURN
```

■

TI-99/4A FRÅN GRUNDEN - 2

*av Peter Walker, TI*MES, England*

PRIMTAL

I datorernas barndom utförde de stora rördrivna maskinerna ofta långa matematiska beräkningar, som att räkna ut pi med hundratals decimaler eller att söka efter primtal. I själva verket används beräkning av primtal fortfarande som ett sätt att jämföra snabbheten hos olika datorer. Som vi alla vet är TI-99/4A Basic långt ifrån att vara snabb, så jag var intresserad av att se hur lång tid det skulle ta vår maskin att visa alla primtal under 1000.

Primtal är de som inte har några andra faktorer än talet självt och 1. De är 2,3,5,7,11,13,17.... Det första program jag försökte finns nedan. För varje tal mellan 2 och 1000, dividerar vi det med varje tal mindre än kvadratroten ur det ursprungliga talet. Om någon division ger upphov till ett exakt heltal så är det inte ett primtal. Programmet tar i Extended Basic 2 min 53 sek att köra (även i grund-Basic).

```
100 FOR N=2 TO 1000
110 FOR J=2 TO SQR(N)
120 IF INT(N/J)*J=N THEN 150
130 NEXT J
140 PRINT N;
150 NEXT N
```

I ett försök att snabba upp programmet, kan vi bestämma att det är slöseri med tid att prova alla tal som en möjlig divisor: det är endast nödvändigt att prova primtal som divisor. Så när ett primtal har hittats, skapar vi en fältvariabel (array) för att lagra dessa och endast välja divisor från denna lista. Programmet skulle se ut så här:

```
100 DIM P(1000):: P(1)=2 ::
    L,J,N,M,Q=1
110 !@P-
120 FOR N=2 TO 1000 :: M=SQR(N)
130 FOR J=1 TO L :: Q=P(J):: IF Q>M
    THEN 160
140 IF INT(N/Q)*Q=N THEN 170
150 NEXT J
160 PRINT N;:: P(L+1)=N :: L=L+1
170 NEXT N
```

Det enda tråkiga är att detta program tar 3 min 3 sek att köra - så mycket för att snabba upp rutinen. T.o.m. att ta bort FOR-NEXT slingan på rad 130 (eftersom den alltid hoppas ur via IF-kommandot i slutet av raden) kommer ytterligare att slöa ned programmet. Observera beräkningen av kvadratroten utanför FOR-NEXT slingan för att undvika upprepade uträkning. Vad kan vi göra för att få ett snabbare program?

Lösningen är att återvända till det grundläggande. Ett annat vanligt sätt att hitta primtal är att skriva ned alla tal mellan 1 och 1000. Du stryker sedan alla multiplar av 2, d.v.s. 4,6,8,10... Sedan strykes alla multiplar av 3, d.v.s. 6,9,12, 15... Talet 4 har redan strukits så du kan hoppa över det eftersom det tydligen inte är ett primtal. När du har nått 1000 så är alla tal som ej har strukits primtal. Nedanstående program förverkligar detta. Elementet hos fältvariabeln P() sätts till -1 (SANT) när de strukits och primtal kan visas vartefter vi framskrider.

```
100 DIM P(1000):: J,K=0
110 !@P-
120 FOR J=2 TO 999 :: IF P(J)THEN
    140 ELSE PRINT J;
130 FOR K=J+J TO 1000 STEP J ::
    P(K)=-1 :: NEXT K
140 NEXT J
```

Den goda nyheten är att detta program körs på endast 37,6 sekunder. I själva verket tycks det begränsas i hastighet, inte av beräkningsslingan utan av fördröjningen när talen visas på själva skärmen. Sensmoralen är klar: utforska flera metoder för att uppnå dina mål och räkna inte med att smarta lösningar också ger största snabbhet. Kan någon slå 37,6 sekunder? Du kan prova att göra om detta program till grund-Basic. Det körs på 52,8 sekunder - kan detta förbättras?

PARAMETERÖVERFÖRING

Jag ska här beskriva en teknik som löser två snarlika problem. När ett program startas, eller fortsättes med CONTINUE efter BREAK, blir alla variabler nollställda. Man kan inte heller överföra variabler från ett program till ett annat efter RUN. För att komma runt detta kan vi använda teckendefinitioner (CALL CHAR) som inte återställs vid RUN och i vissa fall inte heller efter BREAK.

Betrakta de två nedanstående programmen. Vi önskar överföra en teckensträng från PROG1 till PROG2. Om A$ är mindre eller lika med 8 teckens längd, kan varje tecken kodas som ett par hexadecimala tecken inom en teckendefinierande sträng DEF$. Detta beror på att varje tecken kan ha ett värde av 256 möjliga (0-255) ASCII-värden och två hexadecimala tecken (0 till F) ger 16 x 16 = 256 kombinationer. DEF$ är alltid 16 hexadecimala tecken lång. Slingan 120-200 tar varje tecken och omvandlar det till ett par hexadecimala värden B1 och B2. Underprogrammet HEX1 omvandlar varje värde till dess hexadecimala motsvarighet. Rad 220 lagrar definitionen som tecken 127.

När programmet PROG1 kör PROG2 bestämmer CHARPAT DEF$. Slingan 120-180 omvandlar vart och ett av de åtta paren till ett enstaka tecken. Underprogrammet HEX2 omvandlar varje hexadecimalt tecken tillbaka till dess decimala värde. Längre strängar kan hanteras genom att ta upp plats för två tecken. Tal kan överföras genom att de först omvandlas till en textsträng, även om du kanske kan hitta på effektivare sätt att att koda in tal i den hexadecimala strängen. Exempelvis kunde de 16 hexadecimala tecknen i DEF$ användas för att koda 13 signifikanta siffror och två exponenter.

Denna teknik kan användas för att skydda viktiga data från förlust p.g.a. oförutsedda BREAK. När du startar programmet på nytt kan du ha ett val att hämta tillbaka data från teckendefinitionerna.

(Den här beskrivna metoden för parameteröverföring passar bäst om du har kassettbandspelare. Om du har flexskiveenhet kan du istället öppna en temporär fil på flexskivan för lagring av viktiga data ö.a.)

```
90 REM PROG1
100 A$="STRING!"
110 PRINT A$
120 FOR A=1 TO LEN(A$)
130 B$=SEG$(A$,A,1)
140 B=ASC(B$)
150 B1=INT(B/16)
160 CALL HEX1(B1,C$)
170 B2=B-16*B1
180 CALL HEX1(B2,D$)
190 DEF$=DEF$&C$&D$
200 NEXT A
210 DEF$=SEG$(DEF$&RPT$("20",8),
    1,16)
220 CALL CHAR(127,DEF$)
230 PRINT DEF$
240 RUN "DSK1.PROG2"
250 SUB HEX1(A,B$)
260 H$="0123456789ABCDEF"
270 B$=SEG$(H$,A+1,1)
280 SUBEND
```

```
90 REM PROG2
100 CALL CHARPAT(127,DEF$)
110 PRINT DEF$
120 FOR A=0 TO 7
130 B1$=SEG$(DEF$,A+A+1,1)
140 CALL HEX2(B1$,B1)
150 B2$=SEG$(DEF$,A+A+2,1)
160 CALL HEX2(B2$,B2)
170 A$=A$&CHR$(16*B1+B2)
180 NEXT A
190 PRINT A$
200 STOP
210 SUB HEX2(B$,A)
220 H$="0123456789ABCDEF"
230 A=POS(H$,B$,1)-1
240 SUBEND
```

■

STUDSANDE SPRITE

(från Nittinian 83-2)

```
80 REM STUDSANDE SPRITE
90 REM CRAIG MILLER, USA
100 CALL CLEAR :: CALL COLOR
(2,6,6):: PRINT RPT$("+",252
);RPT$("+",252);RPT$("+",56)
:: R=40 :: C=30 :: CALL SPRI
TE(#1,42,2,25,17,R,C)
110 FOR K=1 TO 900 :: CALL P
OSITION(#1,Y,X):: R=R+80*((Y
+R>200)-(Y+R<-1)):: C=C+60*(
(X+C>250)-(X+C<-1)):: CALL M
OTION(#1,R,C):: NEXT K
```

■

DISK MANAGER 1000 VN 3.5 FW 4.30

dropped from FW 4.31 in favour of DR

by Tony McGovern, Australia

This has been reassembled from the Vn 3.5 source code from the Ottawa UG to improve its interface to FUNNELWEB. Various bugs have been fixed and some changes made. Now all DD operations with Myarc disk controllers are at 18 sectors per track in either SS or DS. The program has been shortened and <ctrl-C> and <ctrl-A> added as alternatives to <BACK> and <PROCD>. Some other features have been made over more to our idea of convenience too. The XB program unprotect has been removed as taking up more space than it was worth. Just CALL LOAD(-31931,0) instead.

The option to reload FUNNELWEB will return directly to a set pathname using filename FW, else it first asks for primary and secondary drive numbers with those at entry as default, and then tries to load the FW file or else UTIL1 from the secondary drive. If neither of these files is found then it will use XB LOAD in the primary drive as its source of the FUNNELWEB code. This allows owners of single SSSD to use XB with standard TI-Writer functions and DM-1000 without FW/UTIL1 on the disk.

The program files have been renamed MG/MH to avoid confusion with the originals. Saving details to disk from DM-1000 will go to the MG/MH boot drive, or to the default drive #1 if MG/MH are loaded other than from FUNNELWEB and boot tracking fails. This default may be modified with DPatch in the filename just after the start of the MG file.

** WARNING ** With a Myarc disk controller present and Myarc sense on, any operation involving formatting at double density (Initialize or DiskCopy) will go directly to the physical disk 1 to 4 and format that, ignoring any RAMdisk emulations, but the disk header sectors will be written to the emulating disk if any. Myarc sensing can be turned off as a <fn-3> option for ROMs which default to 18 s/t.

** CAUTION ** DM-1000 does NOT work with 80 track disks. ■

DISK MANAGER 1000

av Jan Alexandersson

Det finns en felaktig artikel om DM 1000 i Micropendium jan 91 sid 36.

Kopiering av skiva med bitmap betyder att endast de sektorer som är markerade som använda på sektor 0 kommer att kopieras till exakt samma sektorer på kopian. Det du vinner är att kopieringen går mycket fort om skivan endast har ett mindre antal använda sektorer. Om du har samma skivenhet för original och kopia så slipper du onödiga skivbyten. Fraktunerade filer kommer inte att bli ofrakturerade. En frakturerad fil har inte sina datasektorer i nummerordning efter varandra utan de finns uppdelade i två eller flera block (detta kan slöa ned laddningen av filen). Du måste istället välja multifilkopiering (sektor 1 har pekare till alla filhuvuden) för att ta bort frakturingen. Markera alla filer med C eller tryck på A om du har FW-versionen som då automatiskt markerar alla filer med C.

Kopiering av skiva med sektorkopiering betyder att alla sektorer kopieras upp till det antal som anges på sektor 0 byte >A->B:

>0168	motsvarar	360 sektorer
>02D0	motsvarar	720 sektorer
>05A0	motsvarar	1440 sektorer

Om du inte själv har skapat filerna på skivan är det alltid säkrast att använda sektorkopiering som ger en exakt kopia oberoende av hur sektorerna har markerats på sektor 0-1.

Filhuvuden lagras i första hand på sektor 2-33 (hex >02 - >21) och först om dessa är upptagna kommer högre sektornummer att användas. Detta gäller oberoende av skivformatering SS/SD, DS/SD, DS/DD.

Ny adress till Ottawa UG:
Ottawa TI-99/4A Users' Group
c/o Bill Gard, 3489 Paul Anka Drive
Ottawa, Ontario K1V 9K6, Canada ■

ASSEMBLER I PROGRAMBITEN

av Jan Alexandersson

83-1.12 Mini Memory, recension
Björn Gustavsson
83-2.04 Editor/Assembler manual
errors (rättelser av EA)
84-1.10 LOADASMBAS, omvandlar
DIS/FIX 80 assembler till
CALL LOAD(adr,data,...).
Börje Häll
84-1.24 Assembler-skolan I:
talsystem, minne, processor.
Tony Rogvall
84-2.20 Assembler-skolan II:
skriva text på skärmen(VSBW)
Tony Rogvall
84-3.24 (ersättes av PB 86-2.22)
84-4.08 Assembler-skolan III:
Dis-Assembler(programbanken)
Långt exempel på AL-program.
Tony Rogvall
85-2.05 Närkontakt med kassett (CS)
Anders Persson
85-2.07 Magnetic Tape Test
Anders Persson
85-2.07 Mini-Assembler för XB+EM
Lennart Thelander
85-2.17 LT-Support, beskrivning
Anders Persson
85-3.23 LT-Support, källkod
Anders Persson
85-4.04 Assembler-skola: program,
register, subrutin, work-
space, context switching,
subrutiner från Basic/XB
Anders Persson
85-4.19 Print logotype on printer 1
DIS/FIX 80 för EA med REF
Sören Bernle
85-4.21 Alpha Lock Check
Computer Kontakt
86-2.06 Lagring på kassettband
Anders Persson
86-2.19 Mini-ASMBAS för Mini Memory
omvandlar maskinkod till
CALL LOAD(adr,data,...).
Björn Landsberg
86-2.22 INPUT för Assembler
Börje Häll
86-3.07 MG Explorer, recension
Anders Persson
86-4.09 Screendump från XB
Sören Bernle
86-5.09 Ladda assembler i PROGRAM-
format och initiera register
och tabeller i XB.
Jan Alexandersson
87-1.20 Styrning av PIO från XB/MM
Anders Persson
87-2.18 Snabbare HCHAR med CALL LINK
Bengt Fahlgren
87-2.22 Print logotype on printer 2
PROGRAM för TIWR m.fl.
reviderat jfr med PB 85-4
Sören Bernle
87-3.04 X-Basic Character Reset
Sören Bernle
87-4.04 Checklist (programbanken)
Lars Thomasson
88-1.05 Superbug II, recension
Jan Alexandersson
88-3.09 Tips och tricks: POKEV,
PEEKV, Listskydd, Sprites
Fredrik Nilsson
89-1.07 Krympa assembler, del 1
Tony McGovern
89-2.06 DSRLNK och RS232/PIO
Tony McGovern
89-2.09 Krympa assembler, del 2
Tony McGovern
89-4.08 VDP-register för 99/4A
Jan Alexandersson
89-4.09 Krympa assembler, del 3
Tony McGovern
89-4.13 Anrop av subrutiner i AL
R.A.Green
89-4.14 Subrutiner i assembler
R.A.Green
90-1.03 Sampling av ljud - MM
L-H.Andersen
90-1.28 Krympa assembler, del 4
Tony McGovern
90-5.11 Assembler Tutorial - 1
Execute at other address.
Tony McGovern
90-5.29 Sampling av ljud - EA
L-H.Andersen
90-6.09 Assembler Tutorial - 2
Position independant code.
Tony McGovern
90-6.21 Boot Tracking
Adrian Robinson
91-1.08 7 sätt att lagra program
PROGRAM, INT/VAR 254, MERGE,
DIS/FIX 80, COMPRESSED.
R.A.Green
91-1.12 Beginner Assembler - 1
The first program
Mack McCormick
91-2.12 BUBBLE - assemblerprogram
INIT av VDP, VDP utility,
GPLLNK, DEF, REF, COPY.
Jan Alexandersson
91-2.16 Beginner Assembler - 2
Using SBUG to debug AL
Mack McCormick ■

CALL LINK FRÅN BASIC & XB

av Jan Alexandersson

Du kan använda CALL LINK för att anropa assemblerrutiner från Basic eller Extended Basic så att man hoppar tillbaka till Basic efter det att maskinkodsrutinen har utförts. Följande korta AL-program har tre olika startadresser som sedan kan anropas med CALL LINK("BEEP"), CALL LINK("HONK") respektive CALL LINK ("POWER"). BEEP ger en ljus ton, HONK ger en negativ ton medan POWER hoppar tillbaka till färgbalkarna (power up).

Använd filen GPLLNK som fanns publicerad i PB 91-2 genom ladda in den i assembleraren med COPY. GPLLNK anropar inbyggda rutiner i GROM (se EA-manualen sid 252-256). Eftersom GPLLNK saknas i XB så måste vi använda den självständiga rutinen som tas in med COPY "DSK2.GPLLNK". Både EA och MM kan lösa ut denna med REF GPLLNK.

```
       DEF  BEEP,HONK,POWER

STATUS EQU  >837C

       COPY "DSK2.GPLLNK"

WS     BSS  >20

RTN    DATA 0

BEEP   MOV  R11,@RTN
       LWPI WS
       CLR  R1
       MOVB R1,@STATUS
       BLWP @GPLLNK
       DATA >34
       LWPI GPLWS
       MOV  @RTN,R11
       RT

HONK   MOV  R11,@RTN
       LWPI WS
       CLR  R1
       MOVB R1,@STATUS
       BLWP @GPLLNK
       DATA >36
       LWPI GPLWS
       MOV  @RTN,R11
       RT

POWER  CLR  R1
       MOVB R1,@STATUS
       BLWP @GPLLNK
       DATA >20

       END
```

Detta program fungerar från Basic (EA/MM) eller XB. Med XB kan du gömma CALL LINK i SUB-program så att du endast behöver anropa med CALL BEEP, CALL HONK eller CALL POWER:

```
100 CALL INIT
110 CALL LOAD("DSK2.LINK/O")
120 REM PROGRAM
130 SUB BEEP
140 CALL LINK("BEEP")
150 SUBEND
160 SUB HONK
170 CALL LINK("HONK")
180 SUBEND
190 SUB POWER
200 CALL LINK("POWER")
210 SUBEND
```

Nästa korta assemblerprogram skriver ett värde till önskat VDP-register: CALL LINK("VDPREG",register,värde) med Extended Basic.

NUMREF (XB EQU >200C, som är felskrivet i EA-manualen) används för att överföra ett tal från Basic till assembler:
R0 = 0 för elementnummer (ej array)
R1 = 1,2,... 15 för variabelnummer

XMLLNK anropar inbyggda rutiner i ROM. CFI omvandlar från flyttal till heltal.

ANDI R2,>00FF maskar bort allt utom högra byten så att max 255 tas emot. Om du endast har 99/4A kan du ändra första stället till ANDI R2,>0007 så att 0 - 7 register kan anropas men för 9938 behövs även högre nummer.

```
       DEF  VDPREG

CFI    EQU  >12B8
FAC    EQU  >834A
```

```
STATUS EQU  >837C
NUMREF EQU  >200C
VWTR   EQU  >2030
XMLLNK EQU  >2018
GPLWS  EQU  >83E0

WS     BSS  >20

VDPREG LWPI WS
       CLR  R0
       LI   R1,1
       BLWP @NUMREF
       BLWP @XMLLNK
       DATA CFI
       MOV  @FAC,R2
       ANDI R2,>00FF
       SWPB R2

       CLR  R0
       LI   R1,2
       BLWP @NUMREF
       BLWP @XMLLNK
       DATA CFI
       MOV  @FAC,R0
       ANDI R0,>00FF
       A    R2,R0

       BLWP @VWTR

       CLR  R1
       MOVB R1,@STATUS
       LWPI GPLWS
       RT

       END
```

Du kan använda detta i XB för att sätta ljusblå skärm vilket motsvarar CALL SCREEN(6):

```
100 CALL INIT
110 CALL LOAD("DSK2.VDPREG/O")
130 CALL LINK("VDPREG",7,5)
140 GOTO 140
```

Du kan skapa nya rutiner som stänger av (CALL SCROFF) och aktiverar (CALL SCRON) skärmen. KSCAN-rutinen uppdaterar VDP-register 1 genom att kopiera värdet från >83D4 till VDP-register 1. CALL LOAD till -31788 skriver värdet till >83D4.

```
100 CALL INIT
110 CALL LOAD("DSK2.VDPREG/O")
120 REM PROGRAM
130 SUB SCRON
140 CALL LINK("VDPREG",1,224)
145 CALL LOAD(-31788,224)
150 SUBEND
160 SUB SCROFF
170 CALL LINK("VDPREG",1,160)
175 CALL LOAD(-31788,160)
180 SUBEND
```

Med videoprocessorn 9938 kan du ordna NTSC(60 Hz), PAL(50 Hz), INTERLACE och 212 linjer.

```
100 CALL INIT
110 CALL LOAD("DSK2.VDPREG/O")
120 REM PROGRAM
130 SUB NTSC
140 CALL LINK("VDPREG",9,0)
150 SUBEND
160 SUB PAL
170 CALL LINK("VDPREG",9,2)
180 SUBEND
190 SUB INTERLACE
200 CALL LINK("VDPREG",9,8)
210 SUBEND
220 SUB PUNKT212
230 CALL LINK("VDPREG",9,128)
240 SUBEND
```

Om du istället vill överföra värden från assembler till Basic så ska du använda NUMASG (XB EQU >2008) samt CIF (XB EQU >0020) som omvandlar från heltal till flyttal.

Strängar överförs på motsvarande sätt med STRREF (XB EQU >2014) och STRASG (XB EQU >2010). I detta fall måste R2 innehålla den CPU-adress som strängen ska flyttas till. Samtidigt måste denna CPU-adress förberedas så att dess första byte innehåller den maximalt tillåtna längden av strängen.

Om du vill göra motsvarande överföringar med EA-Basic, måste du först ladda in BSCSUP (finns på en av skivorna till Editor/Assembler) med CALL LOAD("DSK2.BSCSUP") innan du laddar huvudprogrammet. Du kan sedan REF NUMREF,NUMASG,STRREF,STRASG,ERR. MM-Basic har motsvarande färdigt i ROM så du kan REF utan att ladda BSCSUP. Följande EQU användes:

```
Editor/Assembler:
   CFI EQU >1200
   CIF EQU >2300
```

YAPP VERSION 1.10

YAPP av Alexander Hulpke finns nu i version 1.10. Hardcopy har integrerats med YAPP så att du kan välja den från YAPP:s interna meny. Det krävs dock RAM-minne på >6000 för att denna nya Hardcopy ska kunna användas. Det räcker således inte med Mini Memory utan Supercart eller GRAM krävs. Laddning av GIF-filer fungerar dock fortfarande med Mini Memory.

Default-värdena för svärtning av färger har ändrats så att de är proportionella mot roten ur summan av kvadraterna för röd, grön och blå. Den ursprungliga metoden kan laddas in via filen PATTERN256.

Det finns en ny möjlighet att öka svärtningen genom att datorn kan skriva varje rad två gånger (eller en gång som tidigare).

Den ursprungliga Hardcopy finns troligen kvar som en fristående fil om du beställer YAPP. YAPP som är ett ritprogram för 9938-processorn finns beskrivet i PB 91-1. Se även redaktören i PB 91-2. Du kan beställa YAPP från (disk G15):
Asgard Software, P.O.Box 10306,
Rockville, MD 20849, USA
för USD 29.95 + porto USD 7.50. ■

Mini Memory:
CFI EQU >1200
CIF EQU >7200 (fel i MM-manualen)

Om du ska visa text på skärmen i ett assemblerprogram som anropas från Basic/XB, måste du tänka på att teckentabellen börjar på >0400 i VDP-RAM medan teckennummer räknas från >0000. Detta gör att du måste lägga till 96 till det tecken som ska visas. A som har ASCII-kod 65 måste visas som 161. Du kan här se skälet till att högsta tecken i Basic är 159 eftersom 159+96=255.

Med CALL LINK("NAMN",var1,var2,... var15) kan högst 15 variabler överföras mellan Basic och Assembler. Antalet variabler som överföres kan variera från anrop till anrop ungefär som vissa inbyggda CALL i Basic. Detta kan inte göras med användardefinierade SUB-program som anropas med CALL. ■

FUNNELWEB 4.31 MAR/30/91

by Tony McGovern, Australia

Update Notes (-READ-ME)

(add this to PB 91-1 page 20)

(viii) Boot disk tracking is improved.

(ix) Error handling in the various object file loaders has been revised and improved.

Experience with the Myarc HFDC has been so bad that plans to support this device in DiskReview have been abandoned.

Bugs and Mods (FWDOC/REPT)

Vn 4.31 Oct/20/90 ... Original issue

Nov/16/90 ... Various bug-fixes, minor updates, repairs (some going back to Vn 4.30) in DR40/41, DR80/81, ED/EE, LDFW, CT8K/O, FW/LOAD, and FWDOCs /LOAD /TIWR /DR40 /DR41. The DR-80 document now stretches over 3 files FWDOC/DR80 to /DR82.

Nov/22/90 ... Boot disk tracking now handles DSKn.xx loads from any floppy or RAMDisk under any filename. FW, LOAD, -READ-ME, and FWDOC/LOAD & /EASM revised.

Dec/06/90 ... File Recovery in DiskReview now forces you to do the right thing, and Myarc FDC direct format now works again. Various other little bugs fixed in DR40/41, DR80/81.

Mar/30/91 ... Error handling in the various D/F-80 file loaders (4 Object, 5 ScriptLoad, 6 LowLoader) has been extensively reworked. Title screen added to FW. New files for FW, LOAD, SL, LL. Document files -READ-ME, FWDOC/LOAD, FWDOC/EASM, FWDOC/UTIL are revised. ■

BEGINNER ASSEMBLER – 3
KEYBOARD AND JOYSTICK

by Mack McCormick, USA

INTERACTING WITH YOUR COMPUTER

This assembly language tutorial describes how to use the keyscan (KSCAN) utility contained in the computer for keyboard and joystick input. After you have studied this tutorial you should understand the key addresses required, how to set up for the routine, and how to envoke the utility.

A somewhat different approach has been taken with this tutorial. I have included a few advanced programming concepts to reduce the learning curve required to go from the beginning books on the market to the advanced techniques used in commercial software. If you don't understand some of these concepts then don't worry, they will come to you later. Simply disregard them for now. Concentrate on the objective of learning the keyscan routine.

Several folks have asked questions about addressing modes for opcodes. I have attempted to use all of the modes and will explain them as we go through the program. Let's get started!

If you think about it Extended BASIC uses the KSCAN routine a lot. It's used when you are in the command mode and when in the program execution mode it is used for INPUT, CALL KEY, CALL JOYST, and ACCEPT AT. In fact it's the only way we have to interface to the computer. The KSCAN routine itself is contained in the console in the ROM chip beginning at address >20. It's a large routine and it talks to the keyboard using the Communications Register Unit (CRU) thru a TMS9901 Peripheral Systems Interface chip (just background stuff).

It is not difficult to use. Just REFerence KSCAN in your program to tell the computer you will be using its KSCAN routine. The X/B equate is >201C. Just do a BLWP @KSCAN to use the routine. There are four single byte addresses you need to know in order to use KSCAN. >8374 tells the computer how to map out the keyboard for scanning. Place one of the following bytes at this address to change the keyboard mapping.

>00 Standard TI Keyboard.
>01 Left Side of keyboard and JOYST 1
>02 Right side and JOYST 2
>04 PASCAL keyboard

See your BASIC reference guide for detailed information on the keycodes returned in each mode for each key.

>8375 contains the ASCII hex value of the last key pressed or >FF if no key was pressed. We use the >FF feature to allow us to detect if a key is held down for auto-repeating. >8376 contains the value of the Y (up/down) position of the JOYST and >8377 the X (right/left) position. There are three values returned >00 for no movement, >04, and >FC (-4). These values actually are looked up in a GROM chip by the KSCAN routine based on the position of the controller. There is another way to check if a key has been pressed. The GPL status byte at >837C is bit mapped (each bit in the byte provides different information). Here's the way the STATUS byte is mapped.

0	1	2	3	4	5	6	7
L>	A>	EQ	C	OV	OP	X	-

This status byte is set as a result of the execution of an opcode. For a detailed explanation see your E/A manual. The bit we are interested in for the KSCAN routine is the EQual or bit number 2. This bit is set whenever the result from an opcode is equal. Many opcodes affect this bit and other opcodes use it to make a branching decision. For example, jump equal (JEQ) and jump not equal (JNE) check this bit to see if it is

on when deciding if a jump is to be taken. There is a somewhat unique feature that is frequently used in programming. If you execute MOV R1, R1 or move any byte or word to itself the equal status bit will be set if the value there is equal to zero. We use this feature to check for a key press. By MOVB @STATUS, @STATUS we can see if the equal status bit is set. If set by the KSCAN routine no key has been pressed. If reset, there has been a key pressed. If a key was pressed we can get its ASCII value from >8375. OK, there's most of the background let's go to the program and look at how KSCAN is used in practice. Remember that the computer is actually dumb but very fast. Consider for a moment what is required to have a full function cursor in BASIC. You even have to tell the computer to put the cursor on the screen.

Note we REFerenced KSCAN as a system utility we will be using. Next we establish the EQUates needed for the program. All these do is equate a label with a value and do not occupy any memory. The label may now be used instead of that value. We EQUated MYWS with >8300 which is the bottom of high speed RAM which is actually contained in the computer. I say high speed because the CPU can address 16 bits at a time instead of 8 bits if we placed our workspace in expansion RAM. This increases the speed of program execution as long as we keep the most frequently used values in our registers. This is generally a good place for your workspace even when LINKing from BASIC or X/B but there are exceptions so consult your manual first if you are using system ROM utilities. Next we have the five addresses >8374 - >837C needed for the KSCAN utility. GPLWS EQU >83E0 is the address for the Graphics Programming Utility (GPL) workspace which is used in BASIC and elsewhere as we will see in a moment. The next address is >8C02 which is the address where the write address for the VDP chip is memory mapped in CPU RAM. More in a moment. VWD EQU >8C00-VDPWA will be used as an addressing offset later in the program. This EQUates the label VWD to >FFFE or (-2).

Next are the values we will use for comparison with the ASCII value returned by KSCAN at >8375 in the program.

The statement LI VDP,VDPWA uses the EQUates we established previously to load R15 (the R is not required to be present) with the value of >8C02 (the VDP write address).

Next we set up the screen by setting the background to white and then using a subroutine to set a block of VDP RAM (the color table in this case) equal to a value (dark blue on white characters in this case). Look at the BLKVDP subroutine for a moment. The BL branch and link opcode branches to a subroutine which uses the same workspace registers as the main program. The computer knows where to return because the return address (next statement after the BL) is placed in R11. We can pass data to a subroutine by following the BL with a DATA statement. In this case we are passing three pieces of data. The VDP RAM location to write to, The data value to write, and the number of times to write it. We use an addressing mode called indirect auto incrementing to help us out. The statement MOV *R11+,R0 does the following. R11 is pointing to the address of the word of memory following the BL at this time (>380 in this case). This statement says "Move the value of the data at the address contained in R11 to R0 and then increment the address in R11 to the address of the next word of memory (>4F00 in this case). We just passed >380 to R0 for use by the subroutine. ORI R0,>4000 sets the first two bits of >380 to binary 01 or >4380 without changing the other bits. This is necessary because the first two bits set to 01 tell the VDP chip that we are going to alter the address where it writes or reads data in VDP RAM. Once the new address is set the VDP chip will automatically increment the address to the next byte for consecutive byte writes or reads. This really saves time because a VSBW or VSBR would alter the address every time the routine is used. Data is then passed to R1 and R2 using the same technique. Now we are ready to alter the VDP address. We *always* write

the least significant byte first so we execute a SWAp Byte command and MOVe >80 in this case to the VDPWriteAddress. Now we swap the bytes back and write the most significant byte (>43 in this case). We can now move consecutive data to or from VDP RAM very quickly using the statement MOVB R1,@VWD(VDP). This moves >4F in this case to the VDP write data address at >8C00. How did we come up with >8C00? By using indexed memory addressing of course. Remember VDP is really R15 which we loaded previously with the VDPWA (>8C02). The parenthesis always contains a register. Remember that VWD contains >FFFE or -2. This statement adds the value contained in VWD to the value contained in VDP (R15 ,>8C02) and that becomes the address the data to moved to or >8C00. Easy huh? <grin>. Because we incremented the address in R11 to the next word in the previous statement MOV *R11+,R2 it is now pointing to the next instruction in the program or the following BL @BLKVDP. Next we clear the screen by writing spaces (>20) to all positions and put up the prompts.

Now for the first keyscan. We previously cleared the word of memory at KEYADR or >8374 to tell the computer to scan the entire standard keyboard. Next we MOVeByte @STATUS, @STATUS to see if the equal bit is set (on). We check with a JumpEQual opcode and if the STATUS bit is set we loop back to the KSCAN routine until a key is pressed. When a key is pressed we fall out of the loop and compare the value at KEYVAL (>8375) to a constant to see which key was pressed. We only are checking for one, two, or quit being pressed in this case. If there had been multiple keys we were checking (say more than 5) then the most efficient means would have been to set up a table and use indexed memory addressing to Branch to the correct routine. When the correct key is pressed we Jump to that routine and execute. Note the EOJQ routine. This causes the computer to return to the power up title screen. First interrupts are enabled using a LoadInteruptMaskImmediate command because the GPL interpreter (system monitor) runs with interrupts enabled. Next we tell the computer to use the GPL Workspace. Finally, we Branch and Link Workspace Pointer to the power up reset vector contained in console ROM at address >0000.

Let's look now at the VDWTR routine. First we BranchandLink to another subroutine to clear the screen and place the MSG3 prompt at the bottom of the screen. No data (parameters) are passed to this subroutine. Note that the return address in R11 is saved in R10 because we BL to the BLKVDP routine and the original return address would be lost. Rule: When nesting subroutines always save the return address. This is one method when you only are going one or two levels deep. In some programs I write I go up to six levels deep. In that case I implement a stack in CPU RAM and PUSH and POP values to the stack. Finally we B *R10. We Branch to the address contained in R10 which is our original return address. This is known as indirect addressing. Note in the next keyscan we do not check the STATUS byte to detect a keypress but rather check KEYVAL (>8375) for >FF (no key pressed). This enables the key to auto-repeat. We use some code to slow it down. You should try experimenting with these values. Next we see if Clear, Redo, or the Left Arrow was pressed. If not we print the character on the screen. This routine is fairly straight forward and well documented so you should have no problem following it.

Next is the JOYST routine. Again we clear the screen. Next we scan the entire keyboard. One note on addressing here. The statement CLR @KEYADR tells the computer to clear "whats at (@)" the address of the KEYADR label (>8374 in this case). This is called symbolic memory addressing. MOVB @JOYVAL,@KEYVAL tells the computer to scan the left half of the keyboard and JOYST one by moving a value of 1 to >8374. Then we execute a KSCAN but don't wait for a key. We then divide the JOYST routine to check for three basic conditions; up, down, and if neither of these then left/right. We only check for left/right movement in the up and down routines if the JOYST was moved either up or down.

This routine is sufficiently documented for you to follow from here.

If you are really interested in learning assembler there is only one way. Hit the books and practice programming.

```
****************************************
****************************************
***                                  ***
***        TUTORIAL 3 (NOVICE)       ***
***   Keyboard and Joystick Access   ***
***   By Mack McCormick              ***
***      Use Load and Run from       ***
***           M/M or E/A             ***
***    Entry point is START          ***
***   Active Fctns:Clear,Redo,Quit   ***
***    and Left Arrow                ***
***                                  ***
****************************************
****************************************

       DEF  START
       REF  KSCAN,VSBW,VMBW,VWTR

       TITL 'JOYSTICK AND KEYSCAN DEMO PROGRAM 10 MAY 1985'
****************************************
*      CPU RAM EQUATES                 *
****************************************
MYWS   EQU  >8300        DEFINE MY WORKSPACE (HIGH SPEED RAM)
KEYADR EQU  >8374        KEYBOARD TO SCAN (>00 TO >04)
KEYVAL EQU  >8375        ADDRESS WHERE THE KEY VALUE (ASCII) IS RETURNED
YPOS   EQU  >8376        Y POSITION FOR THE JOYSTICK
XPOS   EQU  >8377        X POSITION FOR THE JOYSTICK
STATUS EQU  >837C        GPL STATUS BYTE
GPLWS  EQU  >83E0        ADDRESS OF GPL WORKSPACE
VDPWA  EQU  >8C02        VDP WRITE ADDRESS
VWD    EQU  >8C00-VDPWA  WRITE DATA ADDRESS OFFSET
VDP    EQU  15           REGISTER FOR THE VDP WRITE ADDRESS

****************************************
* KEYSTROKE VALUES STANDARD KEYBOARD   *
****************************************
ONE    BYTE >31          ASCII VALUE FOR 1
TWO    BYTE >32          ASCII VALUE FOR 2
CLEAR  BYTE 2            FCTN 4
QUIT   BYTE 5            FCTN =
REDO   BYTE 6            FCTN 8
LARROW BYTE 8            FCTN S
H04    BYTE 4            JOYST UP
H00    BYTE 0            JOYST NO MOVEMENT
HFC    BYTE >FC          JOYST DOWN (-4)
HFF    BYTE >FF          VALUE FOR NO KEY PRESSED

****************************************
*           MISC VALUES                *
****************************************
JOYVAL BYTE 1            VALUE FOR JOYST 1
MSG    TEXT 'PLEASE SELECT'
MSG1   TEXT '1 - VIDWRITER'
MSG2   TEXT '2 - JOYST FUN'
MSG3   TEXT 'FCTN REDO-RETURN TO MAIN MENU'
       EVEN              FORCE THE PROGRAM COUNTER TO AN EVEN ADDRESS
```

```
*-- PROGRAM STARTS HERE --*
START  LWPI MYWS          POINT TO OUR WORKSPACE
       LI   VDP,VDPWA     LOAD THE VDP WRITE ADDRESS IN R15

*-- SET UP THE SCREEN --*
       LI   R0,>070F      SET THE BACKGROUND COLOR TO WHITE (>F). VDP REG 7.
       BLWP @VWTR         WRITE IT TO VDP REGISTER 7

       BL   @BLKVDP       SET THE CHARACTERS TO DK BLU ON WHITE
       DATA >380,>4F00,>1E COLOR TABLE, DK BLU ON WHT, 30 BYTES

ENTER  BL   @BLKVDP       NOW CLEAR THE SCREEN
       DATA 0,>2000,767   PLACE >20 IN ALL POSITIONS OF THE SCREEN IMAGE TABLE

       CLR  @KEYADR       SET TO STD TI KEYBOARD

       LI   R0,201        PUT UP 1ST PROMPT
       LI   R1,MSG        ADDRESS OF DATA TO WRITE
       LI   R2,13         NUMBER OF BYTES TO WRITE
       BLWP @VMBW         ON VDP ROUTINES eg. VMBW,VMBR, ETC.
       LI   R0,265        R0 IS ALWAYS THE VDP RAM ADDRESS
       LI   R1,MSG1       R1 IS ALWAYS THE CPU RAM ADDRESS OR CHAR TO WRITE
       BLWP @VMBW             IF SINGLE BYTE ROUTINE (VSBW,VSBR)
       LI   R0,297        R2 IS ALWAYS THE NUMBER OF BYTES TO MOVE
       LI   R1,MSG2
       BLWP @VMBW

*-- TITLE SCREEN KEYSCAN --*
SCAN1  BLWP @KSCAN        SCAN THE KEYBOARD
       MOVB @STATUS,@STATUS IS THE EQUAL STATUS BIT SET?
       JEQ  SCAN1         YEP. KEEP WAITING FOR A KEY PRESS
       CB   @KEYVAL,@ONE  WAS ONE PRESSED?
       JEQ  VDWTR         YEP. GO EXECUTE THAT SUBPROGRAM
       CB   @KEYVAL,@TWO  WAS TWO PRESSED?
       JEQ  JOYST         YEP. GO EXECUTE THAT SUBPROGRAM
       CB   @KEYVAL,@QUIT WAS QUIT PRESSED?
       JEQ  EOJQ          YEP. RETURN TO TITLE SCREEN
       JMP  SCAN1         ONLY ACCEPT 1,2, OR QUIT. KEEP SCANNING.

*-- RETURN TO THE POWER UP TITLE SCREEN --*
EOJQ   LIMI 2             ENABLE INTERRUPTS
       LWPI GPLWS         LOAD GPL WORKSPACE
       BLWP @>0000        BRANCH TO THE RESET VECTOR (TITLE SCREEN)

*-- VIDEO TYPEWRITER ROUTINE --*
VDWTR  BL   @PRMT         CLEAR THE SCREEN AND PUT UP THE PROMPT

       CLR  R3            SCREEN LOCATION COUNTER
SCAN2  BLWP @KSCAN        SCAN THE KEYBOARD
       CB   @KEYVAL,@HFF  HAS A KEY BEEN PRESSED?
       JEQ  SCAN2         KEEP CHECKING

       CB   @KEYVAL,R1    SAME KEY AS LAST TIME?
       JEQ  YEP           SAME KEY

* EXPERIMENT WITH THE VALUES FOR R4 UNTIL YOU OPTIMIZE THE SPEED
       LI   R4,>3000      DELAY TO SLOW DOWN PRINTING FOR USER REACTION (DIFF KEY
       JMP  LOOP
YEP    LI   R4,>5000      DELAY FOR AUTO REPEAT CURSOR
LOOP   DEC  R4            SUBTRACT 1 20,480 TIMES (THATS FAST!)
       JNE  LOOP          PLACE AN * IN FRONT OF THIS LINE TO SEE THE SPEED
```

```
NOPE   CB   @KEYVAL,@CLEAR    CLEAR THE SCREEN?
       JEQ  VDWTR         RESTART THE PROGRAM AND CLEAR THE SCREEN
BYPASS CB   @KEYVAL,@REDO EXECUTE A WARM START TO TITLE SCREEN
       JEQ  ENTER         RETURN TO TITLE SCREEN
       CB   @KEYVAL,@LARROW LEFT ARROW PRESSED?
       JEQ  ARROW         YEP

       MOV  R3,R0         MOVE THE COUNT TO R0
       MOVB @KEYVAL,R1    MOVE THE KEYVALUE TO R1 MSBYTE
       BLWP @VSBW         WRITE IT TO THE SCREEN

       INC  R3            NEXT SCREEN POSITION
       CI   R3,735        ONLY ALLOW TO WRITE TO SCREEN POSN 735
       JLT  SCAN2         CONTINUE TO NEXT CHARACTER
       JMP  VDWTR         CLEAR THE SCREEN AND RESET THE CHAR COUNTER

ARROW  DEC  R3
       CI   R3,0          CHECK FOR LOWER SCREEN LIMIT
       JLT  VDWTR         CLR R0 AND START OVER
       MOV  R3,R0         MOVE THE COUNT TO R0
       LI   R1,>2000      ACSII SPACE CHAR IN MSBYTE OF R1
       BLWP @VSBW         BLANK OUT THE CHAR
       LI   R1,>0800      LOAD FOR AUTO REPEAT DELAY
       JMP  SCAN2         GET THE NEXT CHAR

*-- JOYSTICK ROUTINE --*
JOYST  BL   @PRMT         CLEAR THE SCREEN AND PUT UP THE PROMPT
       LI   R0,400        START PATTERN IN CENTER OF THE SCREEN

JOYLP  CLR  @KEYADR       SET FOR KEYBOARD SCAN
SCAN3  BLWP @KSCAN        SCAN FOR FCTN REDO
       CB   @KEYVAL,@REDO REDO PRESSED?
       JEQ  ENTER         RETURN TO TITLE SCREEN
       CB   @KEYVAL,@CLEAR CLEAR PRESSED?
       JEQ  JOYST         START OVER

       MOVB @JOYVAL,@KEYADR SCAN JOYST 1
       BLWP @KSCAN        SCAN THE JOYSTICK

       MOV  R0,R14
       LI   R0,16         * ADDED TO TEST FIRE BUTTON
       MOVB @KEYVAL,R1
       AI   R1,>2000
       BLWP @VSBW
       MOV  R14,R0

       CB   @YPOS,@H04    MOVED UP?
       JNE  CKDWN         NOPE. MUST BE DOWN

       CB   @XPOS,@H00    MOVED IN THE X DIRECTION?
       JNE  XDET          YEP GO FIND OUT IF LEFT OR RIGHT
       AI   R0,-32        MOVED UP ONLY SO -32 TO GET TO ROW ABOVE
       JMP  DRAW          DRAW THE PATTERN

XDET   CB   @XPOS,@H04    MOVED UP AND RIGHT?
       JNE  XDET1         NOPE
       AI   R0,-31        PRINT POS UP AND TO RIGHT ONE POSITION
       JMP  DRAW          DRAW IT

XDET1  AI   R0,-33        UP AND LEFT IS ALL THAT IS LEFT
       JMP  DRAW          DRAW IT
```

```
CKDWN  CB   @YPOS,@HFC   WAS JOYST MOVED DOWN?
       JNE  LAT          NOPE

       CB   @XPOS,@H00   MOVED STRAIGHT DOWN?
       JNE  XDET2        NOPE
       AI   R0,32        NEXT ROW DOWN
       JMP  DRAW         DRAW IT

XDET2  CB   @XPOS,@HFC   DOWN AND LEFT?
       JNE  XDET3        NOPE
       AI   R0,31        ONE LINE DOWN AND ONE SPACE LEFT
       JMP  DRAW         DRAW IT

XDET3  AI   R0,33        ONE LINE DOWN AND ONE POSITION RIGHT
       JMP  DRAW

LAT    CB   @XPOS,@HFC   NOT MOVED UP OR DOWN. WAS IT LEFT?
       JNE  LAT1         NOPE
       DEC  R0           IT WAS SO MOSVE LEFT ONE POSITION
       JMP  DRAW         DRAW IT

LAT1   CB   @XPOS,@H04   WAS IT MOVED RIGHT?
       JNE  JOYLP        START ALL OVER AT BEGINNING
       INC  R0           YEP MOVED RIGHT.

DRAW   CI   R0,0         CHECK FOR LESS THAN 0
       JLT  OUTBDS       YEP OUT OF BOUNDS
       CI   R0,735       CHECK FOR GREATER THAN 735
       JGT  OUTBDS       YEP OUT OF BOUNDS
       JMP  INBDS        JUMP TO INBOUNDS
OUTBDS MOV  R2,R0        RESTORE OLD SCREEN LOCATION IN R0

INBDS  LI   R1,>4000     LOAD THE @ SYMBOL TO DISPLAY
       MOV  R0,R2        SAVE THE OLD VALUE IN CASE NEW IS OUT OF BOUNDS
       BLWP @VSBW        WRITE IT TO THE SCREEN

       LI   R4,>1000     DELAY FOR A FEW MSECS.
DELAY  DEC  R4
       JNE  DELAY        PLACE * IN FRONT TO SEE REAL SPEED
       JMP  JOYLP        START ALL OVER

*****************************************
* This subroutine clears the screen    *
*   and places the MSG3 on the screen  *
*    No inputs required.               *
*  Alters R0,R1,R2,R10.                *
*   No reserved registers on exit      *
*     Levels 2                         *
*****************************************
PRMT   MOV  R11,R10      SAVE THE RT ADDRESS SINCE WE WILL NEST ANOTHER SUB

       BL   @BLKVDP      CLEAR 732 SCREEN POSITIONS
       DATA 0,>2000,737  PLACE >20 IN THE SIT FOR 732 POSITIONS

       LI   R0,738       LINE 23 OF SCREEN (BASE 0)
       LI   R1,MSG3      ADDRESS OF MSG3
       LI   R2,29
       BLWP @VMBW

       B    *R10         RETURN TO MAIN PROGRAM
```

```
**************************************
* This subroutine writes data to a     *
*        block of VDP RAM              *
*   by directly accessing the VDP      *
* Advantage: Very Fast and efficient   *
*  Inputs: ADDRESS,DATA,# OF BYTES     *
*  Alters:R0,R1,R2,VDPWA               *
*    Levels 1                          *
**************************************
BLKVDP MOV  *R11+,R0     VDP ADDRESS TO WRITE TO
       ORI  R0,>4000     OR ADDRESS WITH BINARY 0100 (>4) TO INDICATE A WRITE
       MOV  *R11+,R1     DATA TO WRITE
       MOV  *R11+,R2     NUMBER OF BYTES TO WRITE
* R11 (RT ADDR) NOW POINTS TO THE INST FOLLOWING THE BL IN THE MAIN PROGRAM

       SWPB R0           ALWAYS WRITE THE LSBYTE FIRST
       MOVB R0,@VDPWA    MOVE THE LSBYTE TO THE VDP ADDRESS REGISTER
       SWPB R0           SWAP THE BYTES BACK
       MOVB R0,@VDPWA    WRITE THE MSBYTE NOW
* THE VDP CHIP ADDRESS REGISTER IS NOW POINTING TO THE ADDRESS IN R0

BLKLP  MOVB R1,@VWD(VDP) MOVES DATA TO EACH SCREEN POSITION
* VDP CHIP ADDRESS REGISTER AUTOMATICALLY INCREMENTS TO NEXT WRITE POSITION
       DEC  R2           DECREMENT THE COUNT
       JNE  BLKLP        FINISHED? (GPL STATUS EQ BIT SET?){=0?} NOPE...CLRLP

       RT

       END                                                              ■
```

FAST EXTENDED BASIC!
87/09 – ERASEDEMO

(c) 1989 Lucie Dorais, Ottawa TI-99/4A Users' Group, Canada

Everybody knows that TI BASIC is slow... but what about its big brother, Extended Basic? Do we use it to its full capacity? After four years of using it, I am still finding ways to make it work faster, in some cases almost as fast as assembly: the time lost in running the program is more than compensated for by the time saved in programming!

Over those years, two of which were spent with a cassette system, I also developed quite a few tricks to make my programs more compact, so that they would load faster from CS1. Many people have asked me to share some of my knowledge of XB, so here is the first installment. But I don't want to write a whole manual. Since I suppose that people dabbling in XB already have a good knowledge of TI BASIC, this will be a practical course: you write (and RUN) the program, and I will explain its main function, but other XB goodies will be thrown in as well.

Study the ERASEDEMO program, which demonstrates five ways to erase part of a screen (20 lines) almost as fast as a CALL CLEAR or an assembly routine, from the slowest to the fastest.

TECHNIQUE #1 (line 160) fills all the screen positions with a space, row by row, column by column... Pretty slow!

TECHNIQUE #2 (line 180) is much simpler and faster, provided you don't use borders: a unique CALL HCHAR that puts 28*20 spaces all at once: much faster! Highly recommended to TI BASIC users!

TECHNIQUE #3 (line 200), unique to XB, is almost same speed, and I used it for years: display 20 empty

lines. The DISPLAY AT could be replaced by CALL HCHAR(X,3,32,28), you would get the same speed.

But why not use Extended Basic DISPLAY AT function to its fullest, since it displays a line instant-aneously, whatever its length? Unfortunately, it has a limit of 255 characters, but that is more than nine lines erased all at once. For exactly nine lines, you need 252 characters, and a full screen would need at the most three strings. To erase less than a full screen, you use the long string N times 9 lines, and you pad the rest with a shorter string.

To build this string, we use XB's RPT$ statement, intended just for that. In our example, we display a 252 long string twice (2 * 9 lines = 18 lines), then pad with 2 more lines, i.e. 56 characters (see TECHNIQUE #4, line 220). Now, our three strings are displayed very fast, but... Tex takes its time to build each string, using a loop. As every manual on any programming language will tell you, the loop is the computer's biggest slowing factor... and all this stop and go is not very agreeable to watch.

The solution of course is TECHNIQUE #5: you build the string only once, at the beginning of the program, and you give it the maximum length (line 130). Each time you need it, you DISPLAY it completely, or just a segment of it. To make your program short and sweet, use the colon se-parator to display the lines, since they are consecutive (line 240).

Now, who said Extended Basic was slow???

P.S.: Did you appreciate my goodies? They are: PRESCANNING (lines 120 and 300), and a USER-DEFINED SUBROUTINE (lines 310-340). If you don't know these macro-tricks, just study the listing and try to figure them out until I write about them in later columns.

```
100 REM ** ERASE DEMO ** Aug
. 1987
110 REM
120 GOTO 130 :: X,Y,K,S,END$
,ER$,LI$,TXT$ :: CALL CLEAR
:: CALL HCHAR :: CALL KEY ::
 CALL MSG :: !@P-
130 TXT$=RPT$("*",252):: ER$
=RPT$(" ",252):: LI$=RPT$("-
",28)
140 DISPLAY AT(1,9)ERASE ALL
:"ERASING DEMO":LI$ :: DISPL
AY AT(23,1):LI$
150 GOSUB 290 :: CALL MSG("C
ALL HCHARs")
160 FOR X=3 TO 22 :: FOR Y=3
 TO 30 :: CALL HCHAR(X,Y,32)
:: NEXT Y :: NEXT X
170 GOSUB 290 :: CALL MSG("O
NE CALL CHAR")
180 CALL HCHAR(3,1,32,640)
190 GOSUB 290 :: CALL MSG("2
8 DISPLAY AT")
200 FOR X=3 TO 22 :: DISPLAY
 AT(X,1):"" :: NEXT X
210 GOSUB 290 :: CALL MSG("R
PT$(' ',252)")
220 DISPLAY AT(3,1):RPT$(" "
,252):: DISPLAY AT(12,1):RPT
$(" ",252):: DISPLAY AT(21,1
):RPT$(" ",56)
230 GOSUB 290 :: CALL MSG("F
AST ERASE")
240 DISPLAY AT(3,1):ER$:ER$:
SEG$(ER$,1,56)
250 DISPLAY AT(24,1)BEEP:"
(1) RUN AGAIN   (2) END"
260 CALL KEY(0,K,S):: IF S=0
 OR K<49 OR K>50 THEN 260 EL
SE IF K=49 THEN 150
270 END$=RPT$("END ",63):: D
ISPLAY AT(3,1):END$:END$:SEG
$(END$,1,56)
280 DISPLAY AT(24,1)BEEP:"YO
U CAN USE IT FOR TEXT TOO!"
:: END
290 DISPLAY AT(3,1):TXT$:TXT
$:SEG$(TXT$,1,56):: RETURN
300 !@P+
310 SUB MSG(TXT$)
320 DISPLAY AT(24,1)BEEP:"Pr
. a key for "&TXT$
330 CALL KEY(0,K,S):: IF S=0
 THEN 330
340 SUBEND
```

■

STYRELSEN ÖNSKAR ALLA EN GOD SOMMAR. Manusstopp för PB 91-4 är 1 aug 1991.

SWEDLOW TI BITS * 1 - 4 *

by Jim Swedlow, USA

(This article originally appeared in the User Group of Orange County, California ROM)

WELCOME TO TI BITS

TI BITS will bring you product reviews, programming notes, TI tidbits, comments, quotes and miscellanea.

QUOTES OF THE MONTH

The computer is no better than its program.
--Elting Elmore Morison, 1966

And for the support of this Declaration, with a firm reliance on the protection of Divine Providence, we mutually pledge to each other our Lives, our Fortunes and our sacred Honor.
--Last Sentence of the Declaration of Independence

SHAREWARE REVIEW: FAS-TRAN

FAS-TRAN is a shareware item offered by Bill Harms. This program is a checkbook recapper/planner. You enter your checks by category and FAS-TRAN helps you keep records for taxes and budgeting. Requirements are Extended Basic, 32K and at one disk drive. A printer makes the program more usefull.

The program comes with 70 preselected categories but you can change them or add new ones. It will accept a total of 99 categories. It will print out reports on your transactions including a very usefull year to date recap. You can sort your file by date, transaction number (check number) or category as needed. You can save your files in a special format that can be read by MULTIPLAN!

It contains a spread sheet option as well as a calculator you can call into a window when working on the spreadsheet.

FAS-TRAN takes up two SS/SD disks and comes complete with sample files and documentation. There are simply too many features to discuss here. If the program has a weakness, it is the documentation. It takes a bit of reading to get thru. Otherwise, this is a fine program. If you need a home budget tool, give FAS-TRAN serious consideration.

To get FAS-TRAN send $5 or 2 initialized disks and a postage paid return mailer to Bill Harms, 6527 Hayes Court, Chino CA 91710.

DID YOU KNOW?

When Disk Manager 2 formats a disk it verifies each sector (you knew that). What was new to me was that if it finds a bad sector, DM2 locks it off. This allows you to use the disk, with slightly less storage space. (Source: a letter in MICROpendium - personally verified)

THE BEST FREEWARE

COMPUTE did a survey on Compuserve of the best free programs. The five TI winners were:

FAST-TERM (terminal emulator)
DISK MANAGER 1000
FUNNELWEB
NEATLIST
MASS-COPY

BOOK REVIEW

The Orphan Chronicles by Ronald G. Albright, Jr. M.D., 172 pages, published by Millers Graphics

I planned to buy this book but never got around to it. Then I won a copy in the May raffle. I started reading it that night and finished it two days later. I just couldn't put it down.

Dr. Albright chronicles the TI Home Computer from the 99/4 (no A) thru TI's self-destructive marketing techniques to Black Friday (October 28, 1983 - the day TI announced that they were dropping the 99/4A) and beyond. Along the way he covers the International Users Group (a "commercial" user group - an oxymoron unique to the 4A world) and publications including Home Computer Magazine (and tells us who Regina really is).

Fully half of the book is devoted to the current status of our orphan. There are chapters on telecommunications, freeware, the future and on current 4A supporters.

An interesting chapter is the survival guide. In ten steps, Dr. Albright suggests practical strategies that we can take to help ensure our computers continued existence.

This informative book is clearly written. The material is comprehensive and logically organized. It will help you understand the history that led to today and will give you food for thought about tomorrow.

All in all, this is a "must have" for the serious 4A owner. It will help put the joys and sorrows of having an orphan into perspective.

QUOTE OF THE MONTH

When you have eliminated the impossible, whatever remains, however improbable, must be the truth.
---Sir Arthur Conan Doyle, The Sign of the Four (1890)

MORE QUOTES OF THE MONTH

Well, as you can tell, life in the Texas Instruments 99/4A orphanage is active and never dull.
---G. Albright in The Orphan Chronicles (1985)

The sellouts were a sight to behold. Rivaled only by the crunch to buy Cabbage Patch dolls the following year, when a J. C. Penney's or Sears or Montgomery Wards sold out TI, the buyers were lined up before the doors opened . . . . Needless to say, stocks were easily sold out. Quickly and efficiently.
---Ibid

FIRST LOOK: THE LOST HITS

Tenex is selling a disk with three games: Computer War, Submarine Commander and River Rescue. According to their catalog, Thorn EMI wrote these for TI just before Black Friday and then they disappeared. Requirements are a disk drive, 32K and XB, EA or Mini-Memory.

Arcade games for $30? Well, my 14 year old twisted my arm until I sent for them. I have not had a chance to try them because I can't get the disk away from him. From over his shoulder I can tell you that they have excellent graphics, changing scenery or multi-screens (no Munch Man here) and require more than just a good hand with the joy stick. You have to think.

My son gives them his highest rating ("not bad" -- which is much better than "OK").

SOME THOUGHTS ON BLACK FRIDAY PLUS 3

As the third anniversary of Black Friday nears, it may be meet to ponder the future of our computer. You probably noticed that this column deals with this issue from a number of perspectives. Reading The Orphan Chronicles does make you think.

When TI opted out of the home computer market, I figured that the 4A would last for another two or three years. I decided to keep my system as I had too much time and money invested to dump it and move to another computer.

I never expected that, three years later, the support for the 4A would be as diverse, extensive, and strong as it is. The products available today far exceed those that TI gave us. Compare DM1000 to DM2, FAST-TERM and 4A TALK to TEII, TI ARTIST and GRAPHX to VIDEO GRAPHICS.

Look at the products that Millers Graphics, CorComp and Myarc (to name a few) continue to put on the market. Look at the quality fairware that keeps coming out.

The 4A will not live forever, but I believe that its useful lifespan is far from over.

There remains one key question: Now that we have done so well, what can we do to keep the 4A viable?

I do not think that there is any one answer. I do think, however, that it is time to stop and think about this issue and to think deeply. Time indeed.

THE INPUT-OUTPUT BUFFER

When you send data to your printer or to disk, your TI stores information in the input-output buffer. Generally it will keep data until it sees the end of a record.

To illustrate, consider this program that demonstrates the graphics abilities of Epson and compatible printers:

```
10 OPEN #1:"PIO.CR"
20 PRINT #1:CHR$(27);"L";CHR$(127);
           CHR$(0)
30 FOR I=1 TO 127
40 PRINT #1:CHR$(I)
50 NEXT I
60 PRINT #1:CHR$(13)
70 CLOSE #1
```

Line 10 opens your printer and tells your 4A NOT to send a carriage return and a line feed every 80 characters. Line 20 puts your printer in graphics mode and tells it to expect 127 graphics characters. The loop in lines 30 thru 50 send the entire range of graphics characters. Line 60 sends a carriage return to clear the printers buffer.

Since there is no print separator after the CHR$(I) in line 40, each character is taken as a record and sent to the printer. If you added a semi-colon after the CHR$(I), all 127 graphics codes would be held in the input-output buffer until line 60 executed.

The difference is speed. Without the semi-colon, it took about 10.2 seconds for this program to run. When the print separator is added, run time dropped to 7.5 seconds.

QUOTES OF THE MONTH

The best laid schemes o' mice and men. Gang aft a-gley
---Robert Burns 1759-1796

But it does move!
---Attributed to Galileo Galilei
1564-1642

ON SUBPROGRAMS

SUBprograms, if you remember, use different variables from the main program. As a refresher, consider this:

```
10 A=3 :: CALL TEST :: PRINT A
20 SUB TEST :: A=10 :: SUBEND
```

If you run this, you will get the number 3 on your screen because the variable A in line 10 is a different variable from that in SUB TEST.

I wondered about how DATA strings and DEFinitions worked in SUBprograms. So I ran some experiments.

It turns out that a XB program can read a DATA statement anywhere. It works if the DATA statement is inside a SUBprogram and the READ command is in the main program or vice-versa. In other words, for purposes of READing DATA, the special rules about SUBprograms do not apply.

No so with DEFinitions. If you DEFine A in the main program, it does NOT carry over into the SUB program. For example:

```
10 DEF A=10 :: C=A :: PRINT C
20 CALL TEST(C) :: PRINT C
30 SUB TEST(C) :: C=A :: SUBEND
```

This little program will first print 10 from line 10 where C is set equal to A, which is DEFined to be 10. It will then print 0 as inside SUBprogram TEST, A is not DEFined so it is zero. In the same manner, if you DEFine something inside a SUBpro-

gram, that DEFinition does not operate outside of that SUB.

QUOTES OF THE MONTH

"The car of the future must be a car for the people . . . the market for a low priced car is unlimited."
---Henry Ford (1863-1947)

"When you call me that, smile!"
---Owen Wister (1860-1938)
from "The Virginian"

PROGRAMMING TIP

Suppose you are writing a program that does a great deal of printing. There is a bug somewhere in the middle of the printing instructions. Every time you try and find it, however, you must wait while your printer wastes a lot of paper getting to the problem. What to do? If your printer is PIO, try substituting RS232.BA=9600. Unlike the parallel port, the serial port does not wait for a ready signal to return from your printer. So all of your print instructions will go out thru the RS232 port into thin air until you find your problem. Setting the baud rate at 9600 speeds things up (if you don't specify a rate, your TI will use 300 - much slower).

MORE QUOTES

"Only those who attempt the absurd achieve the impossible."
---Anon

"The technique is wonderful. I didn't even dream it would be so good. But I would never let my children to come close to the thing. It's awful what they are doing."
---Vladimir Kosma Zworykin (1889)
Developer of television
on his 92nd birthday.

AN INTRODUCTION TO PRINTERS

If you are thinking about buying a printer, beware. Your choices are many as are the pitfalls.

First, you will need some things other than a printer. You need an RS232 card (stand alone or one for your P Box) and a cable. Most printers with a Centronics parallel port that will work with a standard cable (available from the houses that still support the 4A - Tex-Comp etc).

But which printer to buy? Epson? Star? Gorilla? Tandy? What kind? Dot matrix? Daisy Wheel?

First, lets look at the two basic types: daisy wheel and dot matrix (the others are probably out of your price range). A dot matrix printer is five to ten times faster and much more versatile. A daisy wheel gives you letter quality print while the dot matrix gives draft (poor) and 'near' letter quality (better). A tractor feed usually comes with a dot matrix printer but can be an extra cost item with a daisy wheel printer.

If 90% of your work is correspondence and you need top quality in its visual presentation, a daisy wheel is probably for you. Otherwise, for listing programs and all the other things that a printer can do, a dot matrix printer is the better choice.

Having narrowed the field, you still have to pick between the many models on the market. There are no standards in the world of printers for command structures (the codes your computer sends to the printer to tell it what to do). About the only codes two that are close to universal are Carriage Return and Line Feed. After that, anything can mean anything.

There are two 'de-facto' standards. The first is IBM. When big blue made a printer for its PC, it used a character set and command structure completely different than ASCII and just about every printer on the market. Alas, what will work with an IBM PC will NOT work on the 4A, so IBM compatibility is useless (unless you plan to defect).

The other quasi-standard is Epson. These folks developed a rather comprehensive instruction set (includ-

ing graphics protocols) that some other manufactures and many software manufacturer followed. The TI impact printer is actually a bottom of the line Epson MX80. Most of the graphics programs for the TI will work with Epsons. Some of them support other printers, others do not.

A number of manufacturers make printers that follow Epson commands. Most Star (Gemini 10X, SG10, etc) and Panasonics do while the Axiom, Tandy and Banana printers do not.

Here are some suggestions to help you choose. First, see what your friends have and what they think of it. Then, in the store, have the salesman show you the draft and near letter quality print fonts. Note how long it takes to print a page (200 cps - characters per second - means different things depending on who is writing the advertising, I mean specs). Look for true decenders (is the loop below the 'g' below the line?) and the difference between the zero (0) and the letter (O). Make sure you can return it if it doesn't workout.

Plan to spend at least $200 (if you are buying a new printer). Any of the bargains below that normally do not have the features you will need.

My printer? A Star Gemini 10X. Its about 85% Epson compatible and has been a faithful companion.

ANOTHER PROGRAMMING HINT

When working on a program, you save it to disk often just in case your system locks up, etc. To save time, use a working name of <A> for these frequent saves. This saves up to nine key strokes. Also, if you have a load program that reads the disk directory, your working program will be at the top of the list.

Enjoy! ■

Du kan köpa äldre nummer av Programbiten/Nittinian. Passa på och beställ medan de ännu finns i lager.

MODIFIED ARCHIVER FOR HARDDISK(WDS)

```
**********************
* Ralph T.Johnson    *
* P.O. Box 563       *
* St. Bonifacius,    *
* MN 55375, USA      *
* ------------------ *
* 8/21/89            *
**********************
```

I hope this program has NOT violated the rights of Barry Boone, but I had made this modification to his Archiver program shortly after I received it. I know he is in the process of creating an HFDC version, but I felt this would help the TI community until a version is out that has complete support.

The program is just patches to Barry Boone's Archiver program to allow the low level reads, and IF-128 file writes (packed files) to go to or from hard drives. The archive to WDS program will also work with a WDS100 personality card, but the unpack files will not, due to no low-level routines. (I'm working on it!)

Now for the instructions....

ARC-NORMAL: Currently Archiver 3.03g (8/5/89). Archiving files (Support Barry with a donation, his new one may appear sooner). Disk-to-disk.

ARC-TO/WDS: Archive files from the floppy to the ROOT DIR of the hard drive. (This only works to the hard drive ONLY if you compress the files).

ARC-FR/WDS: De-Archive from hard drive to a floppy disk.
IMPORTANT:This program needs RAM at >7000->7FFF to work (MM,SC,GK) due to some workspace shifting.

This program is public domain, but The Archiver programs ARE NOT! Please support Barry and his efforts, so he will continue his work in the TI community.

HFDC-ägare kan kontakta Jan Alexandersson för en kopia, skiva + returporto. Observera att NORMAL-versionen är omgjord för att även passa Geneve (ny assemblering). ■

TIPS FROM TIGERCUB #42

TIGERCUB SOFTWARE
156 Collingwood Ave.
Columbus, OH 43213

I'm very sorry about the error in the BXB routine in Tips #40. The "program to write a program" generated line number 32000 instead of 30002. Here is the correct line -

```
110 OPEN #1:"DSK1.BXBDATA",V
ARIABLE 163,OUTPUT :: PRINT
#1:CHR$(117)&CHR$(50)&"][\[]
$"&CHR$(190)&CHR$(199)&CHR$(
136)&M$&CHR$(0)
```

The Hyphenated Fill and Adjust in Tips #41 will crash if the file contains a line with one character too many, which may be only an unnecessary control character. This fix will help -

```
300 IF LEN(M$)<=L THEN 310 :
: CALL SOUND(200,110,0,-4,0)
:: PRINT M$;" is";LEN(M$);"c
haracters long":"Truncated t
o ";SEG$(M$,1,L):"OK? (Y/N)"
305 CALL KEY(3,K,S):: IF S=0
 THEN 305 ELSE IF K<>89 THEN
 STOP ELSE M$=SEG$(M$,1,L)
310 PRINT #2:M$ :: IF EOF(1)
<>1 THEN 220 ELSE CLOSE #1 :
: CLOSE #2
```

I know that this line is wrong, but key it in just as it's printed, and see what kind of error message you get -

```
100 !DISPLAY AT(3,1):"Progra
m must be SAVEd in:"MERGE fo
rmat."
```

A friend asked me for a program to help him solve the Scram-Lets puzzles in our local newspaper, so I rewrote the Anagrammer that was published way back in Tips #12. It will print out all possible combinations of any 3- to 6-letter word, or only those which have one or two letters in specified positions.

```
100 CALL CLEAR :: DISPLAY AT
(3,5)ERASE ALL:"SCRAM-LETS S
OLVER": :! by Jim Peterson
110 DISPLAY AT(8,1):"OUTPUT
TO? 1":"  (1) SCREEN":"  (2)
 PRINTER" :: ACCEPT AT(8,12)
VALIDATE("12")SIZE(-1):P ::
P=P-1
120 IF P=1 THEN DISPLAY AT(1
2,1):"PRINTER? PIO" :: ACCEP
T AT(12,10)SIZE(-18):P$ :: O
PEN #1:P$
130 PL(1),PL(2)=0 :: L$(1),L
$(2)="" :: DISPLAY AT(5,1)ER
ASE ALL:"TYPE A 3-,4-,5- OR
6-LETTER WORD  " :: ACCEPT A
T(6,6):A$ :: W=LEN(A$):: IF
(W<3)+(W>6)THEN 130
140 DISPLAY AT(14,1):"SEARCH
 FOR COMBINATION WITH":"LETT
ER IN KNOWN POSITION? N" ::
ACCEPT AT(15,27)VALIDATE("YN
")SIZE(-1):Q$ :: IF Q$="N" T
HEN 180
150 DISPLAY AT(17,1):"LETTER
?" :: ACCEPT AT(17,9):L$(1):
: DISPLAY AT(19,1):"POSITION
?" :: ACCEPT AT(19,11):PL(1)
160 DISPLAY AT(21,1):"ANOTHE
R LETTER/POSITION? N" :: ACC
EPT AT(21,26)VALIDATE("YN")S
IZE(-1):X$ :: IF X$="N" THEN
 180
170 DISPLAY AT(21,1):"LETTER
?" :: ACCEPT AT(21,9):L$(2):
: DISPLAY AT(23,1):"POSITION
?" :: ACCEPT AT(23,11):PL(2)
180 PRINT #P :: FOR J=1 TO W
 :: B$(J)=SEG$(A$,J,1):: NEX
T J :: FOR J=2 TO W :: IF B$
(J)>=B$(J-1)THEN 220
190 T$=B$(J):: FOR L=J-1 TO
1 STEP -1 :: B$(L+1)=B$(L)
200 IF B$(L-1)>=T$ THEN 210
:: B$(L)=T$ :: GOTO 220
210 NEXT L
220 NEXT J
230 FOR A=1 TO W :: FOR B=1
TO W :: IF B=A THEN 440
240 FOR C=1 TO W :: IF (C=A)
+(C=B)THEN 430
250 IF W=3 THEN 310
260 FOR D=1 TO W :: IF (D=A)
+(D=B)+(D=C)THEN 420
270 IF W=4 THEN 320
280 FOR E=1 TO W :: IF (E=A)
+(E=B)+(E=C)+(E=D)THEN 410
290 IF W=5 THEN 330
300 FOR F=1 TO W :: IF (F=A)
```

```
+(F=B)+(F=C)+(F=D)+(F=E)THEN
 400 ELSE 340
310 W$=B$(A)&B$(B)&B$(C):: I
F W$<=V$ THEN 430 ELSE 350
320 W$=B$(A)&B$(B)&B$(C)&B$(
D):: IF W$<=V$ THEN 420 ELSE
 350
330 W$=B$(A)&B$(B)&B$(C)&B$(
D)&B$(E):: IF W$<=V$ THEN 41
0 ELSE 350
340 W$=B$(A)&B$(B)&B$(C)&B$(
D)&B$(E)&B$(F):: IF W$<=V$ T
HEN 410
350 IF Q$="N" THEN 380
360 IF SEG$(W$,PL(1),1)<>L$(
1)THEN 390
370 IF X$="N" THEN 380 ELSE
IF SEG$(W$,PL(2),1)<>L$(2)TH
EN 390
380 PRINT #P:W$&" ";:: G=G+1
390 V$=W$ :: ON W-2 GOTO 430
,420,410,400
400 NEXT F
410 NEXT E
420 NEXT D
430 NEXT C
440 NEXT B
450 NEXT A
460 PRINT #P: :"  ";G;"TOTAL
 COMBINATIONS.": : :: G=0 ::
 V$="" :: PRINT "PRESS ANY K
EY"
470 CALL KEY(0,K,S):: IF S=0
 THEN 470 ELSE 130
```

And here is a much-improved XBasic version of the Adder-Upper which first appeared in Tips #13. I find it very useful in adding up several categories of figures in one pass.

```
100 CALL CLEAR :: CALL SCREE
N(16):: FOR SET=1 TO 14 :: C
ALL COLOR(SET,5,1):: NEXT SE
T
110 DISPLAY AT(3,4)ERASE ALL
:"TIGERCUB ADDER-UPPER": :"T
o add up several categories"
:"at one time.": :"Input cat
egories - END when":"finishe
d"
120 CALL KEY(3,K,S):: DIM C$
(22),T(22)
130 X=X+1 :: DISPLAY AT(12,1
):"Category #";STR$(X):: ACC
EPT AT(12,13):C$(X):: IF C$(
X)="END" THEN X=X-1 :: GOTO
170
140 A$=SEG$(C$(X),1,1):: IF
POS(F$,A$,1)=0 THEN F$=F$&A$
 :: IF X<17 THEN 130 ELSE 17
0
150 DISPLAY AT(15,1):"Code l
etter ";A$;" already":"used.
":"Pick another code letter"
 :: ACCEPT AT(17,26)SIZE(1):
A$
160 IF POS(F$,A$,1)<>0 THEN
DISPLAY AT(15,1):;:;:;:;:;::
GOTO 150 ELSE F$=F$&A$ :: C$
(X)=A$&C$(X):: DISPLAY AT(15
,1):;:;:;:;:;:: IF X<17 THEN 1
30 ELSE 170
170 CALL CLEAR :: R=2+(X>8):
: FOR J=1 TO X :: DISPLAY AT
(R,1):"(";SEG$(C$(J),1,1);")
";SEG$(C$(J),2,255):: R=R+2+
(X>8):: NEXT J
180 DISPLAY AT(R+2,1):"Categ
ory ";F$ :: DISPLAY AT(R+4,1
):"Amount"
190 DISPLAY AT(24,1):"Use mi
nus value to subtract"
200 ACCEPT AT(R+2,11+LEN(F$)
)SIZE(1)VALIDATE(F$):Z$ :: Y
=POS(F$,Z$,1)
210 ACCEPT AT(R+4,8)VALIDATE
(NUMERIC):A :: T(Y)=T(Y)+A :
: DISPLAY AT(Y*(2+(X>8)),20)
:T(Y):: GOTO 200
```

Can you figure this one out? (I can't!) -

```
100 DISPLAY AT(3,4)ERASE ALL
:"ILLOGICAL COMPUTER!!": :"
    by Tigercub"
110 DISPLAY AT(7,1):"100 IF
A=2 THEN IF B=2 THEN C=4 ELS
E IF A=2 THEN IF B=3 THEN C=
6 ELSE IF A=3 THEN IF B=3 TH
EN C=9 ELSE IF A=3 THEN IF B
=4 THEN C=12 ELSE C=9"
120 DISPLAY AT(14,1):"Why ca
n't you get C to ":"equal 9
or 12 or 99?"
130 DISPLAY AT(18,1):"A? " :
: ACCEPT AT(18,4):A :: DISPL
AY AT(20,1):"B? " :: ACCEPT
AT(20,4):B
140 IF A=2 THEN IF B=2 THEN
C=4 ELSE IF A=2 THEN IF B=3
THEN C=6 ELSE IF A=3 THEN IF
 B=3 THEN C=9 ELSE IF A=3 TH
EN IF B=4 THEN C=12 ELSE C=9
9
150 DISPLAY AT(22,1):"C=";C
:: GOTO 130
```

This might come in handy to dress up a program -

```
100 CALL CLEAR :: CALL COLOR
(2,5,16):: CALL HCHAR(1,1,42
,768)
```

```
110 X=X+1 :: DISPLAY AT(X,9)
:"**************";:: DISPLAY
AT(X+1,9):"PRESS ANY KEY";::
 DISPLAY AT(X+2,10):"TO CONT
INUE";
120 CALL KEY(0,K,S):: ON S+1
 GOTO 110,130
130 !continue program here
```

Or, if you'd rather do it backwards -

```
100 CALL CLEAR :: CALL COLOR
(2,5,16):: CALL HCHAR(1,1,42
,768)
110 FOR X=10000 TO 1 STEP -1
 :: DISPLAY AT(X+2,9):"*****
********";:: DISPLAY AT(X+1,
9):"*TO CONTINUE*";:: DISPLA
Y AT(X,9):"PRESS ANY KEY";
120 CALL KEY(0,K,S):: ON S+1
 GOTO 130,140
130 NEXT X
140 !continue program here
```

You might find this one useful -

```
100 ! PAINT CALCULATOR by Ji
m Peterson
110 CALL CLEAR :: FOR SET=1
TO 12 :: CALL COLOR(SET,2,8)
:: NEXT SET :: CALL SCREEN(5
):: CALL KEY(3,K,S):: ON WAR
NING NEXT
120 DISPLAY AT(3,7)ERASE ALL
:"PAINT CALCULATOR": :"To de
termine the amount of":"pain
t needed for a room."
130 DISPLAY AT(8,1):"Is the
room a regular square or rec
tangle? Y" :: ACCEPT AT(9,16
)SIZE(-1)VALIDATE("YN")BEEP:
Q$ :: IF Q$="Y" THEN 160
140 DISPLAY AT(11,1):"How ma
ny rectangular areas":"does
the room contain?" :: CALL A
CCEPTER(12,24,A):: IF A=1 TH
EN 160
150 FOR B=1 TO A :: DISPLAY
AT(3,10)ERASE ALL:"AREA #";B
 :: GOTO 170
160 CALL CLEAR
170 DISPLAY AT(5,1):"How hig
h is the ceiling?":"    ft.
   in." :: CALL ACCEPTER(6,2
,HF)
180 CALL ACCEPTER(6,9,HI)::
HI=HI/12 :: H=HF+HI
190 DISPLAY AT(8,1):"How man
y walls?" :: CALL ACCEPTER(8
,17,W):: CALL HCHAR(5,1,32,6
40)
200 FOR J=1 TO W :: DISPLAY
AT(5,10):"WALL #";J: :"Width
    ft    in" :: CALL ACCEPT
ER(7,7,WF)
210 CALL ACCEPTER(7,13,WI)::
 WI=WI/12 :: WW=WF+WI :: SQ=
SQ+H*WW
220 DISPLAY AT(11,1):"How ma
ny doors, windows or":"other
 areas not to be":"painted i
n wall #";J;"?"
230 CALL ACCEPTER(13,19,D)::
 IF D=0 THEN 280
240 FOR L=1 TO D :: DISPLAY
AT(15,1):"AREA NOT TO PAINT
#";L: :;"Width ft    in" ::
CALL ACCEPTER(17,10,WDF)
250 CALL ACCEPTER(17,16,WDI)
:: WDI=WDI/12 :: WD=WDF+WDI
260 DISPLAY AT(19,1):"Height
 ft    in" :: CALL ACCEPTER(
19,11,HDF)
270 CALL ACCEPTER(19,17,HDI)
:: HDI=HDI/12 :: HD=HDF+HDI
:: SQ=SQ-WD*HD :: NEXT L
280 NEXT J :: DISPLAY AT(21,
1):"Paint the ceiling?" :: A
CCEPT AT(21,20)SIZE(1)VALIDA
TE("YN"):QQ$ :: IF QQ$="N" T
HEN 320
290 CALL HCHAR(5,1,32,640)::
 DISPLAY AT(5,1):"Ceiling di
mensions": :"    ft    in by
    ft    in" :: CALL ACCEPT
ER(7,2,CWF)
300 CALL ACCEPTER(7,8,CWI)::
 CWI=CWI/12 :: CW=CWF+CWI
310 CALL ACCEPTER(7,17,CLF):
: CALL ACCEPTER(7,23,CLI)::
CLI=CLI/12 :: CL=CLF+CLI ::
SQ=SQ+CW*CL
320 CALL HCHAR(5,1,32,640)::
 IF Q$="Y" THEN 340
330 NEXT B
340 DISPLAY AT(3,1)ERASE ALL
:"Total of";INT(SQ+.5);"squa
re feet."
350 DISPLAY AT(5,1):"How man
y square feet will":"one gal
lon of your paint":"cover?"
360 ACCEPT AT(7,8)SIZE(3)VAL
IDATE(DIGIT)BEEP:SF :: DISPL
AY AT(9,1):"How many coats?"
 :: CALL ACCEPTER(9,17,C)::
G=SQ/SF*C :: G=INT(G+.5)
370 DISPLAY AT(15,1):"You wi
ll need";G;"gallons or":G*4;
"quarts of paint."
380 CALL KEY(0,K,S):: IF S=0
 THEN 380 ELSE STOP
390 SUB ACCEPTER(R,C,Q):: AC
CEPT AT(R,C)SIZE(2)VALIDATE(
DIGIT)BEEP:Q :: SUBEND
```

■

BASIC & XB TIPS(HTA) - 3

from Bill Sponchia, Canada

20. Did you know that you could delete a file when you close it. The statement is: CLOSE #1:DELETE

21. When programming in XB it pays in two ways to squeeze as many statements as you can into each program line. The first reason is that it saves memory by eliminating line numbers; the second is that it speeds up execution by eliminating the need for the program to process extra lines of code.

22. Another method to save memory by reducing the size of a program is to replace a constant used with a variable. This is assuming that that constant is used a number of different times in the program.

23. When you are editing a program and accidently erase a line by pressing FCTN 3 to get the line back simply type in a single quote mark and then press ENTER. This gives a syntax error and the erased line is back because the change was not syntactically correct and thus not acceptable. The putting in of the quote mark must be done before moving from the line that was erased.

24. If you need to know if CALL INIT has already been executed in your program put in the following lines:
```
10 CALL PEEK(8198,A,B)::IF A=170 AND
B=85 THEN xxx ELSE CALL INIT !!xxx =
```
line number to go to if CALL INIT already executed.

25. Here's a use of the MIN and MAX statements:
MIN - If a variable is restricted to being no higher than 6 you would normally say IF A>6 THEN A=6 however you can say A=MIN(A,6)
MAX - If a variable is restricted to being no lower than 6 you would normally say IF A<6 THEN A=6 however you can say A=MAX(A,6)

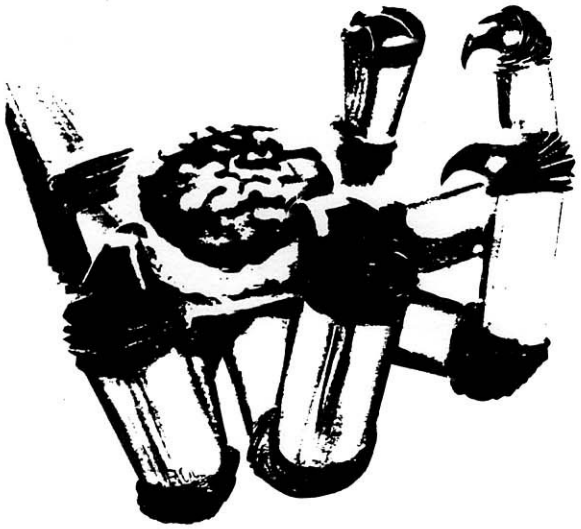
26. You can LIST a program to disk by stating LIST "DSKn.program". This gives a D/V 80 file which is then readable by TI-Writer. This can be helpful for putting program listings in documents but another benefit is the ability to use the FIND STRING command to help locate something in a long program.

27. Here's a short program to write DATA lines which can then be merged into another program.

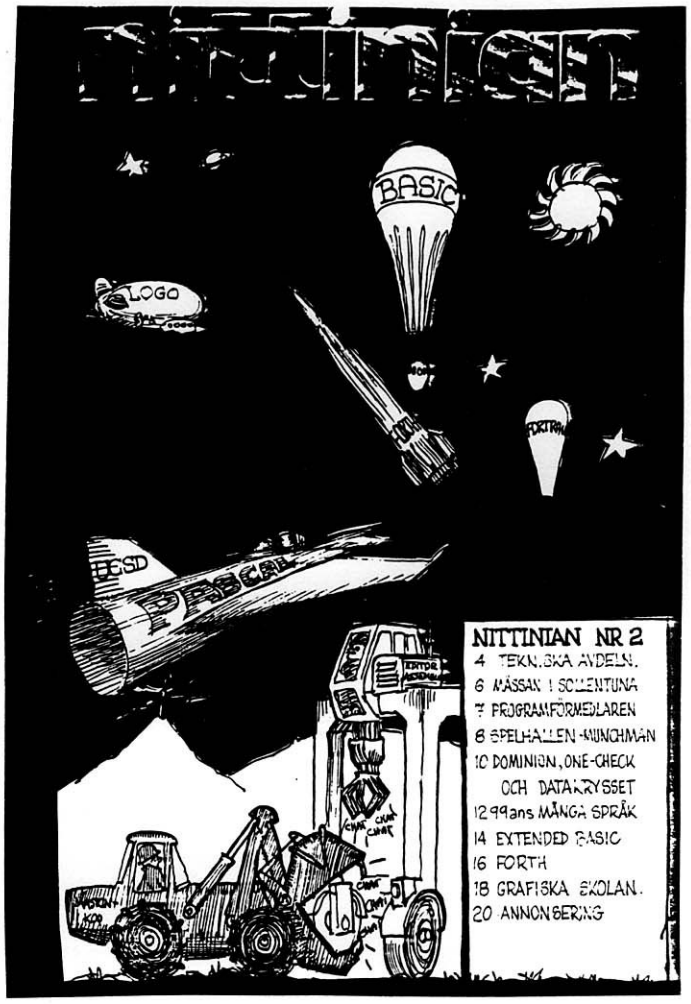
```
100 ON WARNING NEXT
110 DISPLAY AT(10,1)ERASE ALL:"ENTER
FIRST LINE NUMBER:"::
ACCEPT AT(10,25)BEEP
VALIDATE(DIGIT)SIZE(4):LN
120 DISPLAY AT(12,1):"ENTER
INCREMENT"::
ACCEPT AT(12,17)BEEP
SIZE(3)VALIDATE(DIGIT):I
130 DISPLAY AT(14,1):"ENTER
FILENAME:"::
ACCEPT AT(14,16)BEEP
VALIDATE(UALPHA,DIGIT)SIZE(10):FN$
140 OPEN #1:"DSK1."&FN$,VARIABLE 163
150 DISPLAY AT(2,6)ERASE ALL:"PRESS
ENTER TO END"::
DISPLAY AT(22,1):"ENTER A LINE OF
DATA:"::
LINPUT D$
160 IF D$="" THEN 190
170 PRINT #1:CHR$(INT(LN/256)&
CHR$(LN-256*INT(LN/256))
&CHR$(147)&D$&CHR$(0)
180 LN=LN+I::
GOTO 150
190 PRINT #1:CHR$(255)&CHR$(255)
200 CLOSE #1::
END
```

This will save your DATA lines in a Merge format almost ready to be merged into your program. Before this can be done you must do the following:
i) type NEW and press ENTER to clear memory
ii) MERGE in the saved DATA lines. (ie - MERGE DSK1.filename)
iii) EDIT each DATA line by retyping (typing over) the word DATA
iv) SAVE the edited DATA lines in the MERGE format (ie - SAVE DSK1.filename, MERGE)
It is now ready to be put into your program. ■

nittinian



Innehåll nr 1



NITTINIAN NR 2

nittinian

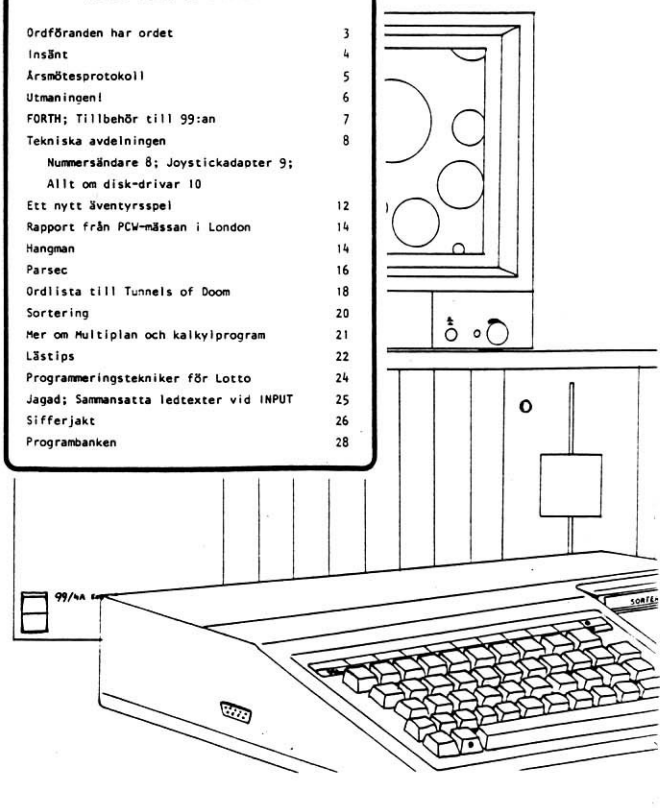
HEMDATORÄGARTIDNINGEN

99AN 83-3

nittinian

83-4/5

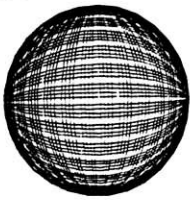
INNEHÅLL



PROGRAM

nittinian BITEN

84-1



Innehåll

PROGRAM

nittinian BITEN

84-2

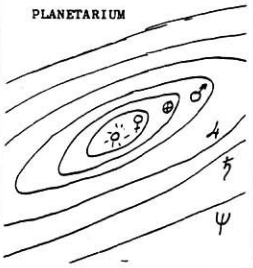
TI59

PÅ

TI-99 4A

I

samt



Innehåll

ISSN 0281-1146

PROGRAM

nittinian BITEN

84-3

Spara skärmen på kassett
Konsten att skriva spel i EX-basic
Dataöverföring i praktiken
Testbild f justering
PåskSöndagen och
ETT SUBPROGRAM
HURRA vad vi är bra !!!
DITT
TILLBEHÖRS-
FÖRSÄLJNING
BIDRAG
ÄR TIONDEN TUNN
SAKNAR VI JUST

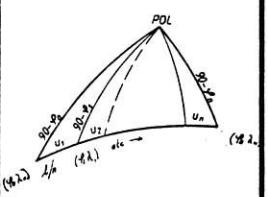
Innehåll

ISSN 0281-1146

PROGRAM

nittinian BITEN

84-4



Innehåll

ISSN 0281-1146

PROGRAM nittinian BITEN 84:5

Årsmöte

Medlemsträff

Årsmöte

Årsmöte

Årsmöte

Årsmöte

Årsmöte

Innehåll

ISSN 0281-1146

PROGRAM nittinian BITEN 85-1

FORTH spalten

STADGAR

LT-WRITER

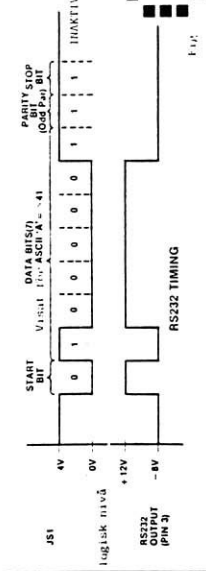
Trippelpunkten

PASCAL

Innehåll

ISSN 0281-1146

PROGRAM nittinian BITEN 85-1.5



Innehåll

ISSN 0281-1146

PROGRAM nittinian BITEN 85-2

FORTH spalten

LT-SUPPORT

Innehåll

ISSN 0281-1146

PROGRAM nittinian BITEN

FORTH spalten

89-3

FORIT

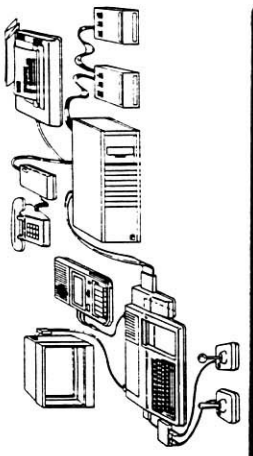
Innehåll

ISSN 0281-1146

PROGRAM nittinian BITEN

FORTH spalten

89-4



Innehåll

ISSN 0281-1146

PROGRAM nittinian BITEN

FORTH spalten

89-1

Årsmöte

Medlemsträff

STAFFANSTORP

Innehåll

ISSN 0281-1164

PROGRAM nittinian BITEN

FORTH spalten

89-2

HELI-KOP-TER

STJÄRN KIKERI

Innehåll

ISSN 0281-1146