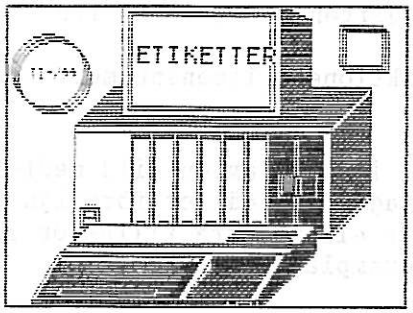
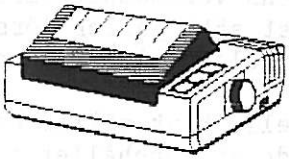
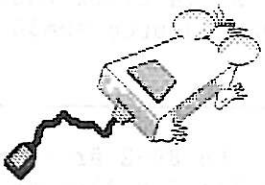
PROGRAM BITEN 20-1







INNEHÅLL

ISSN 0281-1146

KALLELSE TILL ÅRSMÖTE

Medlemmar i föreningen Programbiten kallas härmed till Årmöte Lördagen den 10 mars 1990 kl.13.00:
Albatrossvägen 76, 1 tr.ned, Haninge
(pendeltåg söder om Stockholm)

Förslag till dagordning:

1. Mötet öppnas

2. Val för mötet av ordförande, sekreterare och två justeringsmän

3. Mötets behörighet
   - utlysning
   - röstberättigade närvarande

4. Beslut om dagordning

5. Styrelsens årsredovisning för 1989. Verksamhetsberättelse och kassarapport delas ut på mötet.

6. Revisorernas berättelse för 1989

7. Om ansvarsfrihet för styrelsen för 1989

8. Val av styrelse för 1990. Årmötet väljer enligt stadgarna en styrelse bestående av ordförande, sekreterare, kassör och redaktör, vilka samtliga utses till befattning på årsmötet, samt därtill en och högst fem övriga ledamöter.

9. Val av revisorer för 1990. Mötet väljer två revisorer och en suppleant.

10. Val av valberedning. Mötet väljer två ledamöter varav en sammankallande.

11. Om årsavgift 1990.

12. Ändring av stadgarna (se PB 89-1 sid 13)

13. Årmötets avslutas.

Därefter samlas den nyvalda styrelsen till styrelsemöte.

Redaktör: Jan Alexandersson

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

Postgiro 19 83 00-6
Medlemsavgiften för 1990 är 120:-

Datainspektionens licensnummer:
82100488

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:-
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).

Kraftig jordbävning i Newcastle, Australien i slutet av dec 1989 (Tony och Will McGoverns hemstad).

Manusstopp för PB 89-2 är 22 mars.
Sänd direkt till: Jan Alexandersson, Springarvägen 5,3tr, 14261 TRÅNGSUND
Telefon 08-771 05 69.

SAMPLING AV LJUD

av Lars Herold Andersen, Danmark

Sampler diskette indeholder følgende filer:

EXPMEM/S
EXPMEM/O
MINIMEM/S
MINIMEM/O
CLEAN/S

Mange har vel prøvet det svenske program Perfect Push, og måske fået lyst til selv at eksperimentere med lyd-sampling. Jeg forstår at det går at købe sourcekoden til Push, så man kan sikkert lure teknikken dær. Samplerprogrammet på denne disk er imidlertid 'klart til brug' !

Inspireret af Anders Perssons kassette-program "tidsfordriv" i et gammelt nummer af PB (85-2), har jeg skrevet en assembler rutine, som sampler lyd fra kassette indgangen. Den lagrer lydens binære repræsentation i expansions-minnet, og kværner den ud igen på befaling.

Min første version var til MINI MEM, så jeg havde alle 32 k fri og kunne sample så lange lydsekvenser som muligt. Versionen findes som sourcefil og objectfil på disketten, under navnene MINIMEM/S og /O. I sourcefilen har jeg beskrevet programmets virkemåde, og det er iøvrigt kommenteret hele vejen igennem. (På dansk - men det går vel at forstå, tænker jeg !)

Versionen EXPMEM/O loades til low expmem, og optager således lidt af pladsen til samplede data. Det drejer sig om ca. 2100 bytes, eftersom alt EDIT/ASSM utility også ligger dær. Det er klart, at programmet kan ændres, så det frilægger al RAM pånær sin egen længde, men jeg fandt ingen grund til at gøre det.

CLEAN/S er MINI MEM versionen uden kommentarer. (Kortere og hurtigere at arbejde med.)

Kvaliteten af lyden, som programmet leverer, er ret ringe. Men som demonstration af, "hvad man kan gøre" kan det vel være roligt endda. Mine brødre har en AMIGA 500+programmer, som benytter sig af sound sampling på overbevisende måde, så programmet var nødvendigt : Hør brødre, hør hvad min Texas kan !

Måske kan programmet være til inspiration i Programbiten : En elektronik-kyndig laver en A/D konverter til PIO porten, som sampler i f.eks 4 bit's bredde. Selvfølgelig laver han også en D/A konverter til PIO porten, som fodres med de samplede nybbles ; musikken spiller ! En anden skriver et program, som umiddelbart flytter de samplede data - ikke til exp minnet - men hurtigt udenom disk DSR, direkte til disken. En tredje skriver en sound editor, "DIGITAL COMPOSER" (!), så man kan slette, flytte, kopiere og elektronisk filtrere blokke af samplet data. Den kan naturligvis også konvertere til LPC til tale synt'en. (Jeg har hørt, at talen på PARSEC og STAR-TREK er samplet.) Nå, skæmt til side ! Hvis du kan bruge programmerne til et eller andet, er du velkommen.

Tak for et udmærket og helt nødvendigt Programbiten.
Med venlig hilsen: Lars Herold Andersen, Ellehammervej 9, DK-8260 Viby J, Danmark
(SAMPLER kan beställas från programbanken, MODCOL och BOOT ingår).

ETIKETTER

ETIKETTER av Alexander Hulpke finns på en skiva som kan beställas från programbanken. Den kan användas för att skriva ut TI-Artist bilder i mindre format än vad du normalt får med andra program. Proportionerna mellan längd och bredd är riktiga vilket inte gäller andra program som t.ex. skivan till Brashears Home Publishing. Du kan välja mellan NORMAL (med bredd 2) och BIG (med bredd 4) som ger bilder på 40x54 mm respektive 77x108 mm. Detta program har använts för bilder på första sidan i PB 89-3 och PB 89-4.

```
****                    ********************************
*                       * 'SAMPLER' update 14.Juli 89 *
*                       * MINI MEMORY VERSION     lha *
*                       ********************************
*
*
*     Dette program fungerer som en simpel lyd-sampler. Det registrerer
* audio-input fra kassettekablets hvide ledning, også kaldet CRU >0000 bit 27.
*     Et stort antal gange i sekundet ser programmet om denne bit er høj eller
* lav. Det resulterer i en lang række bits, høje og lave, som lagres i 32 kRAM
* delen. Variationen i bit-strengens værdi, altså mellem 1 og 0, er udtryk for
* amplitude-variationen i den samplede sekvens.  Bit-strengen afspilles ved at
* sætte og resætte AUDIO-gate bit'en (24) i hurtig sekvens, afhængig af de
* samplede bits' værdier. Man får nu en gengivelse af den samplede lydsekvens på
* monitorens højttaler. Lyden er ret ringe, men sikkert genkendelig. Man skal
* ikke forvente det store af en båndbredde på to ! I virkeligheden bliver AUDIO
* gaten ikke høj og lav, den bliver disabled og enabled. Så det man hører er
* faktisk skiftet mellem høj og lav. Det er mere korrekt at manipulere med bit
* 25 -magnetisk output- i stedet for AUDIO-gaten, men resultatet kommer så ikke
* direkte i højttaleren; det ligger på den røde ledning, og skal forstærkes for
* at blive hørbart.
*     Man kan eksperimentere med forstærkning og filtrering af det oprindelige
* signal for at optimere gengivelsen. Det kan også være morsomt at ændre
* sampling- og playback-rate ved at justere de to delay-løkkers varighed. Som
* programmet er her, sampler det 261904 bits i løbet af ca. 12.5 sekunder. Det
* svarer til en sampling-rate på ca. 21 kHz. ( Kold computer ! )
*
*     Denne version af programmet er beregnet til at loades i MINI MEMORY, men
* det er uproblematisk at ændre det for placering i 32 kRAM delen: Utility-
* vektorerne og variablerne MIN1,MAX1,MIN2,MAX2 mA justeres - men husk at disse
* sidste fire værdier skal være LIGE, for at programmet kan fungere.
*
*     Kørsels-instruktioner for MINI MEM versionen :
*             VÆLG MINI-MEM, LOAD & RUN
*             FILENAME:     DSKn.MINIMEM/O   ( Programmet auto-starter )
*             Sæt musik på kassette-afspilleren, sæt volume højt, og sørg for,
*             at kassette-kablet er tilsluttet. Følg nu instruktionerne på
*             skærmen.
*
*     Med MINI MEM kan man QUIT'e, og ændre f.eks delay-konstanterne med EASYBUG
* Programmet kan herefter startes med EASY BUG kommandoen E 7118 .
****


       DEF  SAMPLR

USRWSP EQU  >8300        VÆLG ET KVIKT WORKSPACE
MAGIN  EQU  >1B          HER BOR KASSETTEKABLETS HVIDE LEDNING
AUDIO  EQU  >18          HER SLÅS LYD-UDGANGEN TIL OG FRA
STATUS EQU  >837C        ADRESSEN FOR GPL STATUS BYTE. UNDERSØGES EFTER KEYSCANS
ISR    EQU  >83C4        HER LOADES STARTADRESSEN FOR USER-DEFINED INTERRUPT
VSBW   EQU  >6024        UTILITY-VEKTORER FOR MINI-MEMORY
VMBW   EQU  >6028
VWTR   EQU  >6034
KSCAN  EQU  >6020

       AORG >7118        FØRSTE FRIE ADRESSE I MEDIUM MEMORY

SAMPLR LWPI USRWSP
```

```
       LI   R0,>01F0      VÆLG TEXT MODE
       BLWP @VWTR
       SWPB R0
       MOVB R0,@>83D4     HUSK - EN KOPI AF VDP-reg 1 HERTIL !

       CLR  R12           SØRG FOR DEN RETTE CRU-BASE

       MOV  @LINK,@ISR    FORTÆL @>83C4 OM INTERRUPT-RUTINENS PLACERING

**
* 'SAMPLER' BENYTTER SIG AF TO SKÆRME;
* EN FRA OG MED >0000 TIL OG MED >03BF, OG
* EN FRA OG MED >0C00 TIL OG MED >0FBF.
**

       LI   R0,>3BF       SKÆRMEN SLETTES FRA BUNDEN; POSITION #959 FØRST
       LI   R1,>2000      ASCII-KODEN FOR >SPACE< ER >20
CLRLOP BLWP @VSBW         >SPACE< SKRIVES TIL VDP-ADRESSEN I R0 ( SKÆRM 1 )
       SOC  @ADDER,R0     LÆG >0C00 TIL R0
       BLWP @VSBW         >SPACE< SKRIVES TIL VDP-ADRESSEN I R0 ( SKÆRM 2 )
       SZC  @ADDER,R0     TRÆK >0C00 FRA R0
       DEC  R0
       JOC  CLRLOP        FYLD MED >SPACE< INDTIL R0 GÅR FRA >0000 TIL >FFFF
*                         ( ALTSÅ NÅR POSITION #0 OG #>C00 ER OVERSKREVET )

**
* LÆG TEKST PÅ DE TO SKÆRME
**

       LI   R0,>0030      HER SKAL TEKSTEN PÅ SKÆRM 1 STARTE
       LI   R1,TEXT1      HER FINDES DEN,
       LI   R2,103        SÅ MANGE BYTES FYLDER DEN,
       BLWP @VMBW         OG NU SKRIVES DEN...
       LI   R0,>0C32      HER SKAL TEKSTEN PÅ SKÆRM 1 STARTE
       LI   R1,TEXT2      O.S.V...
       LI   R2,20
       BLWP @VMBW

**
*           HER STARTER DET EGENTLIGE PROGRAM.
* PROGRAMMET RESTARTER OGSÅ HER EFTER ET BRUGER-INTERRUPT
**

RESTRT
       SBZ  AUDIO         ENABLE AUDIO-GATE ( KAN VÆRE RESAT I INTRPT-ØJEBLIKKET)
       LIMI 0             INTERRUPT FORBUDT
       LI   R0,>07FC      SKÆRMEN SKAL VÆRE GRØN ( C ) MED HVID FORGRUND ( F )
       BLWP @VWTR
       LI   R0,>0200      SKÆRM 1 SKAL BRUGES; DEN STARTER VED >0000, ALTSÅ
       BLWP @VWTR         >00 TIL VDP-reg 2

**
* PROGRAMMET HAR LAGT TEKST 1 PÅ EN GRØN SKÆRM,
*       OG NU MÅ BRUGEREN BESLUTTE SIG
**

       CLR  @>8374        DET ALMINDELIGE KEYBOARD SKAL SKANNES
SCAN   BLWP @KSCAN
       MOVB @STATUS,R4    KIG PÅ RESULTATET AF SKANNINGEN
       COC  @KEYBIT,R4    ER DER TRYKKET PÅ EN TAST ?
       JNE  SCAN          HVIS IKKE, SÅ SKANNES DER ENDNU ENGANG
```

```
**
* NÅR EN TAST ER RAMT, MÅ PROGRAMMET VIDERE
**

       LI   R0,>07F6      SKÆRMEN SKAL VÆRE MØRK-RØD ( 6 ), FORGRUNDEN HVID
       BLWP @VWTR
       LI   R0,>0203      SKÆRM 2 SKAL BRUGES; DEN STARTER VED 3 * >400 = >C00
       BLWP @VWTR         ALTSÅ >03 TIL VDP-reg 2

**
* PROGRAMMET HAR LAGT TEKST 2 PÅ EN RØD SKÆRM -
*     SKAL DER INDSPILLES ELLER AFSPILLES ?
**

       MOV  @>8374,R4     HENT ASCII-KODEN FOR DEN TAST SOM RAMTES
       COC  @ENTKEY,R4    BLEV DER TRYKKET PÅ >ENTER< ?
       JEQ  SAMPLE        HVIS JA, SÅ SKAL DER SAMPLES. ELLERS ER DET PLAYBACK..

************************
**   PLAYBACK AFSNIT   **
*                      *

PLYBCK
       MOV  @MIN1,R0      AFSPILNINGEN STARTER FRA MIN1
LOOP1  C    R0,@MAX1      ER R0 NÅET FREM TIL SLUTNINGEN AF LOW MEMORY ?
       JNE  $+6           HVIS NEJ, SÅ SKIP NÆSTE INSTRUKTION
       MOV  @MIN2,R0      HVIS JA, SÅ FORTSÆTTER AFSPILNINGEN FRA START AF HI-MEM
       C    R0,@MAX2      ER R0 NÅET FREM TIL SLUTNINGEN AF HIGH MEMORY ?
       JHE  PLYBCK        HVIS JA, SÅ PLAYBACK IGEN
       LI   R2,>F         DER ER 16 BITS I ET WORD. DET SKAL SHIFTES 15 GANGE
       MOV  *R0+,R1       HENT ET WORD IND I R1 FOR DISSEKTION. FORØG R0 MED 2
LOOP2  COC  @LSBIT,R1     ER BIT 15 SAT ?
       JNE  SILNC1        HVIS NEJ, SÅ ER DET TYST
       SBO  AUDIO         HVIS JA, SÅ BLIVER AUDIO-BIT'EN HØJ
       JMP  $+4           SPRING VIDERE
SILNC1 SBZ  AUDIO         AUDIO-BIT'EN BLIVER LAV

       LI   R3,1          EN LILLE FORSINKELSE, OM MAN VIL.
DELAY1 DEC  R3            JO HØJERE VÆRDI ( >= 1 ) I R3, DES LAVERE
       JNE  DELAY1        AFSPILNINGS-HASTIGHED

       SRL  R1,1          SHIFT R1 TIL HØJRE; KLAR TIL AT UNDERSØGE NÆSTE BIT
       DEC  R2            FORMINDSK SHIFT-TÆLLEREN
       JNE  LOOP2         HVIS IKKE ALLE BITS I WORD'ET ER AFSPILLET SÅ GØR DET
       LIMI 2             GIV BRUGEREN LIDT INDFLYDELSE; TILLAD VDP-INTERRUPTS
       LIMI 0             SLUT MED DET !
       JMP  LOOP1         VIDERE TIL NÆSTE WORD I RÆKKEN
************************

************************
**   SAMPLER AFSNIT    **
*                      *

SAMPLE
       MOV  @MIN1,R0      INDSPILNINGEN STARTER I MIN1
LOOP3  C    R0,@MAX1      ER R0 NÅET FREM TIL SLUTNINGEN AF LOW MEMORY ?
       JNE  $+6           HVIS NEJ, SÅ SKIP NÆSTE INSTRUKTION
       MOV  @MIN2,R0      HVIS JA, SÅ FORTSÆTTER INDSPILNINGEN I START AF HI-MEM
       C    R0,@MAX2      ER R0 NÅET FREM TIL SLUTNINGEN AF HIGH MEMORY ?
       JHE  RESTRT        HVIS JA, ER INDSPILNINGEN SLUT; TILBAGE TIL RESTART
       LI   R2,>F         ELLERS INITIALISERES SHIFT-TÆLLEREN
LOOP4  SRL  R1,1          SHIFT TIL HØJRE; BIT 0 ER KLAR TIL AT MODTAGE EN SAMPLE
```

```
       LI   R3,1          JO HØJERE VÆRDI ( >= 1 ) I R3, DES LAVERE SAMPLING-RATE
DELAY2 DEC  R3
       JNE  DELAY2

       TB   MAGIN         ER DEN HVIDE KASSETTELEDNING LOGISK HØJ ?
       JNE  SILNC2        HVIS IKKE ER DET TYST; BIT 0 FORBLIVER 0
       SOC  @MSBIT,R1     INDGANGEN VAR HØJ; BIT 0 BLIVER SAT
SILNC2 DEC  R2            FORMINDSK SHIFT-TÆLLEREN
       JOC  LOOP4         HVIS IKKE ALLE BITS I WORD'ET ER INDSPILLET, SÅ GØR DET
       MOV  R1,*R0+       FLYT DET INDSPILLEDE WORD UD I MEMORY
       LIMI 2             GIV BRUGEREN LIDT INDFLYDELSE; TILLAD VDP-INTERRUPTS
       LIMI 0             SLUT MED DET !
       JMP  LOOP3         VIDERE TIL NÆSTE WORD I RÆKKEN
*                        *
*                        *
**                      **
***                    ***
************************


**
* HER FINDES INTERRUPT-RUTINEN, SOM UDFØRES 50 GANGE I SEKUNDET
**


INTRPT
       LWPI USRWSP        LOAD EGET WORKSPACE. INTERRUPT-RUTINEN HAR VALGT GPLWS
       BLWP @KSCAN        KEYBOARD'ET SKANNES...
       MOVB @STATUS,R4
       COC  @KEYBIT,R4
       JNE  $+8           HVIS INGEN TAST ER RAMT, SÅ SKIP NÆSTE INSTRUKTION
       MOV  @REST,@>83DC HVIS EN TAST ER RAMT, SKAL DER RETURNERES TIL RESTRT.
*                         ROM-INTERRUPT-RUTINEN RETURNERER FRA INTWS, SÅ VED AT
*                         LÆGGE ADRESSE FOR RESTRT I INTWS' R14 ( @>83DC ) BÆRES
*                         PROGRAM-COUNTEREN TIL DET RETTE STED AF RTWP !
       LWPI >83E0         RESTORE GPLWS
       RT                 RETURNER

**
* NOGLE NYTTIGE DATA
**

MIN1   DATA >2000         FØRSTE FRIE ADRESSE I LOW MEMORY
MAX1   DATA >3FFE         SIDSTE FRIE ADRESSE I LOW MEMORY
MIN2   DATA >A000         FØRSTE FRIE ADRESSE I HIGH MEMORY
MAX2   DATA >FFE0         SIDSTE FRIE ADRESSE I HIGH MEMORY
ADDER  DATA >0C00         BRUGES VED SLETNING AF SKÆRMENE
ENTKEY DATA >000D         ASCII-KODEN FOR >ENTER< I MSBYTE
KEYBIT DATA >2000         BRUGES VED KSCAN. MASKE FOR BIT 2
MSBIT  DATA >8000         BIT 0
LSBIT  DATA >0001         BIT 15
LINK   DATA INTRPT        HER FINDES BRUGER-INTERRUPTETS START-ADRESSE
REST   DATA RESTRT        HER FINDES RESTART-RUTINENS START-ADRESSE
TEXT1  TEXT 'PRESS >ENTER< TO SAMPLE'
       TEXT '                       '
       TEXT '                       '
       TEXT '-ANY OTHER TO PLAY BACK'
TEXT2  TEXT 'HIT ANY KEY TO ABORT'

       END  SAMPLR
```

MYARC OCH TI RS232/PIO

av Jan Alexandersson

Ett RS232/PIO kort monteras i expansionsboxen och har två RS232-utgångar och en parallellutgång PIO. Du använder dessa för att ansluta printer eller modem. Observera att både PIO och RS232 är dubbelriktade så att de kan både sända och ta emot data (således även PIO !!).

Det finns tre fabrikat av sådana kort från TI, Corcomp och Myarc. Jag känner inte till Corcomp men jag ska försöka beskriva skillnaderna mellan TI och Myarc. Jag tror inte att Corcomp har problem med vissa printrar som TI har. Du som har Corcomp, skriv och berätta. Jag vore mycket tacksam om någon kunde sända en kopia av Corcomps manual till mig.

RS232-KONTAKT

Anslut RS232 via en 25-polig D-SUB:

pinne	RS232/1	RS232/2
1	skyddsjord	skyddsjord
2	RD in	-
3	TX ut	-
5	fri bit ut**	-
6	1.8kohm +12V*	1.8kohm +12V*
7	signaljord	signaljord
8	DCD ut	-
12	-	DCD ut
13	-	fri bit ut**
14	-	RD in
16	-	TX ut
19	-	DTR in
20	DTR in	-

* = DSR endast fast nivå
**= CTS som använder CRU >130A resp CRU >130C. Finns ej hos Myarc.

Det kan vara mycket praktiskt att ha en Y-kabel för RS232 som släpper ut de två RS232-portarna på två separata 25-poliga D-SUB-kontakter kablade som RS232/1 ovan.

Anslut modem till RS232/1:

Dator pinne	Modem pinne		CCITT V.24
2	3	RDX	104
3	2	TDX	103
5 *	20	DTR	108.2
6	-		
7	7	signaljord	102
8 *	4	RTS	105
20	5	CTS	106
-	6	DSR	107
-	8	DCD	109

* = är ej nödvändig.
- = anslutes ej.

Modem pinne 20 DTR kan eventuellt anslutas till Dator pinne 6.

Anslut printer enligt följande:

Dator pinne	Printer pinne
2	2
3	3
20	20 eller 11
7	7

Myarc har följande tillägg som ej finns hos TI:

- RS232/1 kan även anropas med P så du kan t.ex. skriva P.BA=4800.LF.

- även hastigheten BA=19200.

PIO-KONTAKT

pinne	PIO/1
1	handshake ut
2	data, LSB
3	data
4	data
5	data
6	data
7	data
8	data
9	data, MSB
10	handshake in
11	signaljord
12	10 ohm + 5V
13	fri bit in *
14	fri bit ut *
15	1 kohm +5V
16	signaljord

* saknas hos Myarc som saknar ingången och har jord på utgången.

Anslut en printer enligt följande:

Dator	Printer	
1	1	Strobe
2	2	
3	3	
4	4	

5	5	
6	6	
7	7	
8	8	
9	9	
10	11	Busy
11	19	Signaljord
16	16	Signaljord

Det kan finnas problem att använda TI RS232 med vissa printrar t.ex. Tandy, Okidata och Star NX-1000 (endast vissa versioner av PROM). Parallellsnitt kan finnas i två olika utföranden:

- Centronix (som Myarc har)
- Epson (som TI använder)

Skillnaden består i olika tidsvillkor för handshake (strobe). TI har ej den vanliga standarden som Myarc har, så Myarc fungerar alltid utan problem med alla printrar.

HP Ink Jet har dessutom inverterad strobe.

TENEX säljer en aktiv specialkabel som passar till TI-kort med olämplig strobe-signal. Micropendium sept 1989 har ett schema för en kabel med en aktiv 74121 IC-krets.

Myarc fungerar enligt Centronix om du anropar med PIO men det finns två nya mjukvaruswitchar som ändrar handshake till TI-typ med "PIO.HS" och till "inverted Busy" med "PIO.IB". Med enbart "PIO" har Myarc normal strobe och inte handshake.

MANUAL FÖR RS232

TI RS232-manualen är mycket dålig med flera grova sakfel blandat med tryckfel. Myarcs manual är en kopia av TI-manualen med alla fel rättade, kompletterad med Myarcs egna ändringar. Vid programöverföring mellan två datorer måste Myarc anropas med SAVE "PIO.HS" för att parallellporten skall efterlikna TI-kortet.

Rättelser till TI-manualen:

sid 6 radnummer 200 skall avslutas med VARIABLE 255 och inget annat.

sid 7 exempel 1 skall vara:
(alla tak fattades)

```
100 OPEN #1:"PIO"
200 PRINT #1:"X","X^2","X^3"
300 FOR I=1 TO 10
400 PRINT #1:I,I^2,I^3
500 NEXT I
600 CLOSE #1
700 END
```

DEVICE NAME

RS232-kortet finns på CRU >1300 och har då följande "device name":
PIO som även kan anropas med PIO/1
RS232 som även anropas med RS232/1
(Myarc har även P för detta)
RS232/2

Du kan ha ett andra kort samtidigt om du ändrar till CRU >1500 på detta och får då:
PIO/2
RS232/3
RS232/4

TEST AV RS232/PIO

För dig som vill prova RS232 finns en testrutin från Mack McCormick som aktiverar en särskild testmode i RS232 så att det går att sända och ta emot samtidigt utan yttre kabel.

TI TEST som finns i programbanken har testrutiner för att prova både PIO och RS232. I detta fall måste du förbinda in- och utgångarna med varandra. Du behöver inte ansluta lysdioderna eftersom testprogrammet fungerar ändå. Denna test fungerar troligen ej med Myarc.

REFERENSER

Myarc RSIC-1 manual till RS232-kort
TI PHP 1220 manual till RS232-kort
TI Schematic RS-232 prod.360
TI TEST (finns i programbanken)
TELCO 2.1 manual p.23(programbanken)
TI*MES 20 p.49: RS232 and all that
Mack McCormick: Direct Access to the TMS9902 UART.
Michael Bunyard: Hardware Manual p 9-1, RS232 Card.
PB 86-5: Hur man styr och ställer
PB 88-4: RS232 till TI-99/4A
PB 89-2: DSR-LINK och RS232/PIO
Micropendium:
juli 87: Avatex modem cable
sept 89: TI RS232 PIO, Star NX-1000.

QUAD DENSITY FLEXSKIVOR

av Jan Alexandersson

En flexskiva lagrar data i form av sektorer som vardera innehåller 256 bytes. En sådan sektor är den minsta mängd data som du kan skriva till en skiva, så datorn skriver alltid en hel sektor på en gång. Du skriver ibland mot ett datafil med PRINT #1: A,B,C som kanske består av mycket mindre än 256 bytes. Datorn lagrar då i en buffert i RAM-minnet tills den kan skriva hela sektorn. Glöm bara inte att stänga filen med CLOSE #1 eftersom det kan finnas data kvar i bufferten.

SKIVHUVUD

För att hålla reda på alla filer på en skiva finns två speciella sektorer nummer 0 och 1. Sektor 1 består av pekare sorterade i filernas bokstavsordning som pekar ut vilken sektor som innehåller filhuvudet. Detta filhuvud visar filnamn, filstruktur (DIS/VAR 80, INT/FIX 128, PROGRAM m.m.) och vilka sektorer som innehåller data som tillhör just denna fil. Sektor 0 har övergripande information om skivan som antalet sidor, spår/sida, sektorer/spår och en tabell över vilka sektorer som är upptagna så att datorn vet vilka sektorer den kan använda för en ny fil. TI gjorde denna tabell så att den rymmer högst 1600 sektorer. En SS/SD 90 kbytes skiva använder 360 sektorer medan en 360 kbytes DS/DD skiva har 1440 sektorer.

Om man har en större skiva än 400 kbytes (1600 sektorer) så kommer det inte att finnas plats för alla sektorer. Detta är skälet till att en 512 kbytes Myarc RAM-disk inte kan använda mer än 400 kbytes. De resterande 112 kbyten används som expansionsminne 32 kB, printbuffert och arbetsminne för Myarc Extended Basic II. Horizon RAM-disk har löst problemet genom att man samtidigt kan ha två disknummer på vardera 360 kbytes (1440 sektorer) på varje kort. Corcomp 512 kbytes RAM-disk har 32 kB expansionsminne men de resterande 480 kB kan användas som en RAM-disk. Detta ordnas genom att ytterligare en sektor används för att markera upptagna sektorer. En tom RAM-disk har då 3 upptagna sektorer (0-2).

QUAD DENSITY (5,25 tums)

Det finns ett Myarc diskkontrollkort (med DS/QD PROM) och ett Myarc hårddiskkort (HFDC) som båda kan använda skivor med 720 kbytes DS/QD (double sided/quad density). Detta betyder att det finns 2880 sektorer. Myarc har löst problemet med sektor 0 genom att låta varje bit markera två sektorer samtidigt (1 allocation unit = 2 sektorer) vilket betyder att du inte kan använda mindre än två sektorer för filhuvud och två sektorer för data. Den kortaste fil tar då fyra sektorer. Det program du använder skriver och läser trots detta enstaka sektorer på 256 bytes. En flexskiveenhet för quad density kan även läsa single density och double density men endast skriva QD.

För att snabba upp sökningen av filer så lagras filhuvudena i första hand på sektor 2-33. För en vanlig SS/SD eller DS/DD betyder detta att 32 filhuvuden kan placeras med snabb åtkomst. Vid mer än 32 filer så kommer sektorer med höga nummer att användas. Om mindre än 32 filer finns så kommer datorn att lagra data även på lediga sektorer 2-33 men först när alla andra sektorer är upptagna.

För en DS/QD skiva som använder 2 sektorer för filhuvud betyder detta att endast 16 filhuvuden kan placeras på sektor 2-33 med snabb åtkomst. Med mitt Myarc HFDC-kort kan jag inte lagra datasektorer på sektor 2-33 om jag har få men långa filer. Om du använder Archiver så kan man lätt få ett större antal mycket långa filer.

DISK MANAGER FÖR 720 KB DS/QD

Det finns några disk manager som även fungerar för 720 kbytes:

- Funnelweb Show Directory 80 COL
- Funnelweb Disk Review
- Myarc CALL DIR
- Myarc DM III
- Myarc DM V för HFDC
- Disk Utilities

Övriga disk manager som jag har sett fungerar ofta bra utom det att de anger antalet lediga och upptagna sektorer felaktigt eftersom 1 allocation unit = 2 sektorer. Detta gäller DM1000, Hard Master, Funnelweb Quick Directory och Show Directory 40 COL.

De disk manager som fungerar bra visar alla samma antal lediga och upptagna sektorer. Dessa motsvarar de allocation unit som är markerade i sektor 0. När det gäller filens längd så anges den på olika sätt:

DS/QD	Huvud	Datasektorer	Minimum
FW SD	1	använda	2
CALL DIR	2	använda	3
DM III	2	jämt antal	4
DM V	2	jämt antal	4
DSKU	2	jämt antal	4

Funnelweb anger den längd som visar vilket utrymme som behövs vid kopiering till en mindre skiva. CALL DIR anger det antal sektorer som ej kan disponeras för att öka den egna filens längd. Vid udda antal datasektorer finns ju en ledig sektor som endast får användas av den egna filen. DM V och DSKU visar det antal sektorer som ej kan disponeras av andra filer. Ingen av dessa disk manager har kläm på att eventuella lediga sektorer 2-33 ej kan användas för data utan endast för filhuvud.

TI-Writer uppför sig annorlunda än andra program eftersom den kortaste fil som kan lagras på en DS/QD-skiva har två datasektorer. Trots detta kan tre datasektorer lagras.

Även 3,5 tums DS/DD 720 kB kan användas på samma sätt som 5¼ tums DS/QD.

HIGH DENSITY (3,5 tums)

Myarc HFDC är förberedd för DS/HD 1.44 MB 3,5 tums skivenheter men mjukvara finns ej klar. Jag har ej sett några uppgifter om hur den kommer att vara organiserad när det gäller allocation unit. Jag misstänker att den får allocation unit = 4 sektorer. Detta skulle medföra att den minsta fil blir 8 sektorer lång.

HÅRDDISK

Min 20 MB hårddisk har en allocation unit = 2 sektorer så att filer upptar alltid 2 sektorer för filhuvud och ett jämt antal sektorer för data. Den minsta filen blir då 4 sektorer. I detta fall visar båda CALL DIR och DM V 3 sektorer för minsta filen. Detta är något underligt eftersom en 20 MB hårddisk helt motsvarar en DS/QD 720 kB flexskiva. En hårddisk kan ha allocation nummer på max >FFFF och en 20 MB hårddisk har >99C0. Detta betyder att en 40 MB hårddisk måste ha mer än 2 sektorer per allocation unit.

REFERENSER

PB 86-2: Att hitta på en skiva
PB 86-4: Advanced Diagnostics
PB 86-5: Disk Manager
PB 88-1: Myarc DS/QD-kort
PB 88-4: Myarc HFDC, DM V 1.21
PB 89-1: Myarc HFDC, DM V 1.29
PB 89-1: Nyheter, sid 2
PB 89-4: Sektor-Editor
MG Advanced Diagnostics
Asgard Hard Master

FUNNELWEB 4.21 DEC/20/89

Funnelweb 4.21 kan beställas från programbanken. FW för 99/4A (40 kolumn) ryms på 3 skivor (45 kr) och FW för 9938 (40+80 kolumn) ryms på 4 skivor (60 kr). En nyhet är DISK-REVIEW som tidigare fanns som ett fristående program QDAV(se PB 89-2), men som nu även finns för 99/4A i en 40-kolumnsversion (saknas i tidigare vers. 4.20). Konfigureringen av menyerna är radikalt ändrad så att "pathname" används så att hela Funnelweb kan lagras på en egen subdirectory på hårddisk så det går att ladda med RUN "WDS1.FWB.LOAD". Även DM1000 klarar reload från hårddisk. BACK från centrala menyn medger byte av printernamn och CHARA-fil (döpta till C1 och C2).

MCGOVERNS XB-SKOLA, DEL 2

av Tony McGovern, Australien

Detta är en fortsättning på tidigare avsnitt av Extended Basic-skolan som har publicerats i tidigare nummer av Programbiten:

del 1A se PB 87-4 sid 23
del 1B se PB 88-1 sid 4

III. PARAMETERLISTA FÖR UNDERPROGRAM

I föregående avsnitt såg vi hur underprogram passade in i det övergripande arbetet hos Extended Basic. I detta avsnitt ska vi gå in på detaljerna när det gäller att skriva underprogram (SUB-program). De flesta tidskrävande detaljer handlar här om konstruktionen av parameterlistor samhörande med CALL- och SUB-kommandona, och något om de små fallgropar du kan råka ut för.

Alla sorters information kan överföras från det anropande CALL-programmet till det anropade SUB-programmet via parameterlistan, och allt som inte överförs på detta sätt kommer att förbli unikt för varje program, med undantag för DATA-satser som är lika åtkomliga för alla. Om något anges i parameterlistan så är det en dubbelriktad kanal om inte särskilda åtgärder vidtages, som finns tillgängliga i XB. I detta fall kan det anropande programmet informera underprogrammet om variabelvärden i en parameterlista, men inte tillåta underprogrammet att ändra värdet av den variabel som finns i CALL-programmet. Variabler i form av fält t.ex. A(10) och B(11,14)(array), numeriska eller av strängtyp, kan inte skyddas från att påverkas av SUB-programmet när de en gång har blivit överförda via parameterlistan.

Låt oss till att börja med ta ett mycket enkelt men användbart exempel, där ett program behöver åstadkomma en fördröjning på olika ställen. Några BASIC-dialekter (och TI LOGO) har en inbyggd WAIT-funktion. XB har inte detta kommando (även om en syniker kan tycka att GPL ger det hela tiden vare sig du vill eller inte) så du måste programmera det. Det kan göras med ett antal CALL SOUND eller med en FOR-NEXT-slinga. Låt oss använda en tom loop för att alstra fördröjningen, omkring 4 millisekunder per varv och placera loopen i ett SUB-program.

```
...
230 CALL DELAY(200)
...
670 CALL DELAY(200/D)
...
990 CALL DELAY(T)
...
...
3000 SUB DELAY(A):: FOR I=1
TO A :: NEXT I ::SUBEND
```

Detta är enklare att följa när du editerar ditt program än att använda GOSUB, och du måste skriva in subrutinen i varje SUB-program eftersom GOSUB eller GOTO ut ur ett SUB-program är förbjudet. Det blir även mindre rörigt än att skriva ut fördröjningsslingan varje gång. Exemplet visar flera olika CALL av DELAY. Den första har ett numeriskt värde och när DELAY anropas så kommer motsvarande variabel i SUB-listan, A, att sättas till 200. Det är en speciell sorts CALL från rad 670 där uttrycket 200/D först beräknas innan det överförs till DELAY och tilldelas A. Variabel D kan t.ex. representera svårighetsgraden i ett spel. CALL från rad 990 använder variabel T så att A i SUB-programmet sätts till värdet av T i det anropande programmet vid tidpunkten för anropet.

Inget oönskat händer med T i detta exempel, eftersom DELAY inte ändrar A. Nu kanske det inte hade någon betydelse i detta exempel om T behåller sitt värde efter CALL. Antag istället att fördröjningen skulle anropas i sekunder. Då kan ett SUB-program DELAYSEC efter samma principer se ut så här

```
...
230 CALL DELAYSEC(2)
...
990 CALL DELAYSEC(T)
```

```
...
4000 SUB DELAYSEC(A):: A=A*2
50
4010 FOR I= 1 TO A :: NEXT I
:: SUBEND
```

Nu efter det att DELAYSEC har blivit utförd på rad 990, kommer T att ha ett värde 250 gånger större än före anropet. Detta är inte något problem om du inte använder T igen med sitt ursprungliga värde. Om det anropande programmet anger en numerisk konstant som på rad 230, eller ett numeriskt uttryck, så kommer ändringar av A i underprogrammet inte att påverka huvudprogrammet. Antag att du inte kan tolerera att T ändras på rad 990 (och detta kan vara orsaken till programbuggar). Du finner att XB tillåter att T kan tvingas att behandlas som om det var ett matematiskt uttryck, och således hindra T från att ändras av SUB-programmet, om T sätts inom parentes i CALL-listan (ej SUB-listan). Antag att även DELAYSEC anropas från rad

```
970 CALL DELAYSEC((T))
```

Om detta CALL på rad 970 följs av ett CALL från rad 990, och om inte T ändrats under tiden, kommer samma fördröjning att erhållas, men om ordningen skiftas så kommer den andra fördröjningen att bli ungefär 250 gånger längre. I XB:s språkbruk är detta känt som överföring av värde till skillnad mot överföring av variabel (reference). Detta kan endast göras för enkla variabler eller enstaka element i fält (matriser), som uppför sig som enkla variabler i CALL-listan. Hela fält (matriser) kan inte överföras med värde utan endast som variabel. Matematiska uttryck och konstanter kan endast överföras med värde, och det är svårt att se vad annat som skulle kunna göras med dem. I det skrivna exemplet har ett annat variabelnamn använts i SUB, men om du kommer ihåg det lilla experimentet i föregående avsnitt så ser du att det inte har någon betydelse om T hade använts i SUB-listan istället för A.

Låt oss nu krångla till det hela litet genom ett blinkande meddelande på den understa skärmraden under fördröjningstiden.

```
...
200 CALL MESSAGE(300," YOUR
TURN NOW")
...
270 CALL MESSAGE(T,A$)
...
...
3000 SUB MESSAGE(A,A$):: DIS
PLAY AT(24,1):A$
3010 FOR I=1 TO A :: NEXT I
:: DISPLAY AT(24,1):""
3020 SUBEND
```

SUB-parameterlistan innehåller nu en numerisk och en sträng-variabel i den ordningen. Varje CALL till detta underprogram måste vara försett med ett numeriskt värde eller numerisk variabel och ett strängvärde eller en strängvariabel, i exakt samma ordning som de uppträder i SUB-listan. I den lilla programsnutten ovan, kommer rad 200 att överföra konstantens värde och rad 270 överför hela variabeln. Det är inget som hindrar att den ena överförs med värde och den andra överförs med variabel om så önskas.

Denna process kan utsträckas till godtyckligt antal poster i parameterlistan, under förutsättning att motsvarande poster i SUB- och CALL-listan motsvarar varandra för varje post, numerisk för numerisk och sträng för sträng. XB-manualen säger inte detta uttryckligen men det verkar som det inte finns någon gräns, utöver det vanliga problemet med maximal radlängd, för antalet poster i listan. Detta är den enda uppenbara skillnaden mellan parameterlistan i XB-underprogram och argumentlistan för CALL LINK ("xxxxxx", , ...) för maskinkodsrutiner med Mini Memory och E/A-Basic (CALL LINK kan även fungera med varierande antal poster i argumentlistan ungefär som CALL CHAR m.fl. i Basic ö.a.)

En liten frihet som är förknippad med de inbyggda underprogrammen finns inte tillgänglig med användardefinierade SUB-program. Några inbyggda som t.ex. CALL SPRITE tillåter ett varierande antal poster i CALL-listan. Parameterlistan i användardefinierade underprogram måste exakt motsvara den etablerade SUB-listan eller det ges ett felmeddelande "INCORRECT ARGUMENT LIST in

...". Användardefinierade CALL tillåter att hela fält (matriser), numeriska eller av strängtyp, överförs till underprogram. Hela fält (matriser) kan endast överföras som variabel. Enstaka element i fält (matriser) kan användas som om de var enkla variabler och kan skyddas från att ändras med extra parenteser i CALL-listan. Ett fält (matris) anges i parameterlistan med parenteser omkring indexpositionerna. Endast närvaron av varje index behöver anges som i A(). MATCH(,,) anger ett tredimensionellt fält MATCH som tidigare uttryckligen eller indirekt har dimensionerats som en sådan. Lämna inte mellanslag i listan. Om underprogrammet behöver veta dimensionen av fältet så måste detta överföras separat (eller som förutbestämda element hos fältet). TI Basic är svagare än vissa andra dialekter eftersom den inte tillåter direkta operationer på fältet som en helhet, en mycket irriterande begränsning.

Fält (matriser) kan dimensioneras med DIM inom ett SUB-program. Detta kommer att skapa ett nytt fältnamn i programmet, och ett fält- eller variabelnamn från SUB-parameterlistan kan inte användas eftersom det resulterar i ett felmeddelande. I den följande snutten överför huvudprogrammet bl.a. ett fält SC till SUB-programmet BOARD (kanske en rutin för att skriva en resultatlista i ett spel).

```
100 DIM SC(2,5) :: ....
...
450 CALL BOARD(P,A$(),SC(,))
...
4000 SUB BOARD(P,A$(),S(,))
:: DIM AY(5):: ......... ::
CALL REF(P,AY(),S(,)))
...
4080 SUBEND
5000 SUB REF(V,A(),B(,))::
... :: SUBEND
```

BOARD skapar internt en vektor AY() vilken överförs till ett annat underprogram REF tillsammans med SC(,), som BOARD känner som S(,), och REF i sin tur som B(,) -- samma namn skulle ha kunnat användas på alla ställena. Det finns emellertid ingen möjlighet för huvudprogrammet eller något SUB-program vars kedja av CALL inte kommer från BOARD att känna till vektorn AY(). Detta gäller på samma sätt för varje variabel eller fält (matris), sträng eller numerisk, som först definieras inom BOARD och vars värde inte har transmitterats tillbaka till det anropande programmet genom någon annan variabel som finns i BOARD:s parameterlista.

Genom att följa detta resonemang kan du kontrollera slutsatsen att det inte finns någon möjlighet för underprogrammet vars kedja av CALL inte går genom BOARD att känna till vektorn AY(). Den enda lösningen är att AY() DIMensioneras i huvudprogrammet (även om det är dess enda förekomst där) och att informationen överförs genom alla nödvändiga CALL-SUB-kedjor.

Denna lösning med att DIMensionera ett fält (matris) enbart inom ett SUB-program är särskilt användbart om ett fält (matris) skall läsas från DATA-satser med READ och ska användas av underprogrammet. Samma sak kan göras från andra underprogram som behöver samma data, utan att man behöver överföra dess namn upp och ned längs CALL-SUB-kedjor. Kom ihåg att DATA-satser fungerar som en gemensam pool från vilken alla underprogram kan läsa med READ. Om fältets värde är resultatet av beräkningar så måste dessa värden överförs genom CALL-parameterlistor.

För fullständighets skull skall sägas att IMAGE-satser, trots att XB-manualen inte säger något om detta, som används för att formatera PRINT kan kommas åt från godtycklig del av ett program på samma sätt som DATA-satser och är inte begränsade till de underprogram i vilka de befinner sig som när det gäller DEF-kommandon.

Det är inte nödvändigt att ha några parametrar i listan överhuvudtaget. Underprogram som används på detta sätt kan vara mycket nyttiga för att dela upp ett långt program i mera hanterbara segment för att underlätta editeringen. Vi kommer att se i senare avsnitt att det även kan finnas andra fördelar.

Ytterligare ett kommando för underprogram återstår nämligen SUBEXIT. Detta är inte helt nödvändigt eftersom det alltid är möjligt att skriva SUBEND på en separat rad och använda GOTO till denna rad om ett villkor uppfylls som kräver ett omedelbart uthopp. Liksom många små bekvämligheter i livet är detta mycket trevligt att ha och gör program mycket lättare att läsa och editera. Det ersätter inte SUBEND som är signalen till XB pre-scan att markera slutet av ett underprogram. SUBEXIT erbjuder ett snyggt och lättfattligt uthopp från underprogrammet (klumpigt t.ex. i vissa Pascal-dialekter). Nästa avsnitt kommer att visa typiska exempel på dess användning.

IV. ANVÄNDBARA UNDERPROGRAMSEXEMPEL

I föregående avsnitt använde vi DELAY som ett exempel på ett underprogram som, med små ändringar, kan användas för att ersätta WAIT-kommandot som finns i vissa andra programmeringsspråk. Du kan fortsätta på denna väg genom att själv bygga upp ett bibliotek av praktiska underprogram som du kan använda i program för att få dina egna tillägg till samlingen av underprogram som XB erbjuder. MERGE-möjligheten med flexskivor gör detta särskilt enkelt. Se Jim Peterson Tigercub Tips för många flera exempel.

Som vårt första exempel tar vi en av de mera frustrerande saker som TI gjorde genom att välja uppsättningen av inbyggda underprogram. Om du har Mini Memory eller E/A så känner du till att systemets tangentavkänningsrutin, SCAN, som finns i datorns ROM kommer att lämna information om tangentbord och joystick samtidigt, medan XB tvingar dig att göra olika CALL-anrop nämligen KEY och JOYST för att gräva fram detta. Eftersom dessa GPL-rutiner är långsamma så är det svårt att skriva snabba spel i XB som behandlar tangentbord och joystick på ett likvärdigt sätt som när det gäller många spelmoduler. Men det finns å andra sidan inget skäl varför spel som kräver tänkande och inte arkadspelens snabba reaktion skall tvinga spelaren att göra valet en-gång-för-alla och inte kunna använda tangenter eller joystick valfritt under hela spelets gång.

SUB-programmets lösning på detta är, när man väl insett att det är möjligt (och vi har sett kommersiella XB-spel där författarna ej har) är att skriva ett spel som använder joystick, men ersätter JOYST med det användardefinierade SUB-programmet JOY som returnerar samma värden som JOYST också när tangenter används.

Det första steget för att bestämma om tangenter eller joystick har använts är att testa tangenterna, och om ingen har tryckts så undersökes joystick. Om en tangent har tryckts så måste dess svar, K, bearbetas så att piltangenterna gömda i det delade tangentbordet alstrar motsvarande JOYST-värden. Ett underprogram enligt samma riktlinjer som det som används i TEX-BOUNCE gör just detta.

```
900 SUB JOY(PL,X,Y):: CALL
KEY(PL,K,ST):: IF ST=0 THEN
CALL JOYST(PL,X,Y):: SUBEXIT
910 X=4*((K=4 OR K=2 OR K=1
5)-(K=6 OR K=3 OR K=14))
920 Y=4*((K=15 OR K=14 OR K=
0)-(K=4 OR K=5 OR K=6))
930 SUBEND
```

PL är spelarens (vänster eller höger joystick eller sida av det delade tangentbordet) nummer och lämnas orörd av proceduren. Den enkla lösningen att omvandla K till (X,Y)-värden genom att använda XB:s logiska operatorer (en av de mera irriterande funktioner som saknas i grund-BASIC) tycks fungera lika väl som något annat. Underprogrammet som det är skrivet kommer att först testa tangenterna men balanserar detta genom att lägga jobbet med bearbetningen beroende på tangentsvaret.

Detta är ett bra tillfälle som något att vässa dina egna talanger genom att utarbeta alternativa versioner av denna procedur, och också genom att skriva en passande ersättning för CALL KEY som svarar med piltangenternas värden även när joystick användes.

UCSD PASCAL MODCOL & BOOT

av Lars Herold Andersen, Danmark

```
(*

Programmet 'modcol' på UCSD disket-
ten ændrer farver i UCSD kommando
mode.

Takket være Anders Persson ved vi,
at P-systemet holder en kopi af VDP
registrene i 8-kram delen, nærmere
bestemt i >2cb0 - >2cbc. Det er ty-
deligvis herfra de ægte VDP registre
opdateres, og ved at skrive en ny
værdi til den adresse som opdaterer
VDP register 7, ( som jo styrer for-
og bggrundsfarve i text-mode ) kan
man få andre farver på skærmen.

Pseudo-VDP register 7 holder til i
>2cbc og>2cbd. Den første byte er
>87 og angiver, at den anden byte
skrives til VDP-reg 7. ( bit 0 sat,
bits 4-7 giver registernummeret.)
Den anden byte styrer for- og bag-
grundsfarven på skærmen, og har
normalt værdien >17, altså sort på
cyan.

Om man til eksempel vil ændre far-
verne til hvid på rød, må man skrive
værdien >87f6 til adresssen >2cbc. I
2's complement bliver det -30730 til
11452.
   ( Regel; hex -> 2'sC : konverter
   fra hex til decimal. Om decimal
   er større end 32767 så subtraher
   65536 )

Programmet her giver hvidt på sort,
men det står frit at ændre.

Jeg har lagt programmet ind som del
af min SYSTEM.STARTUP. Her bruger
jeg i forvejen MODRS232, hvor vari-
ablerne cpuaddr.int og .ptr allerede
eksisterer, og jeg har således blot
tilføjet konstanten psuedo_vdp_reg_7
og programlinierne til den originale
MODRS232.

                        lha Nov-89   *)

program modcol;

  const
    pseudo_vdp_reg_7 = 11452;
                (* hex 2cbc *)

  type
    window = record
             case boolean of
               true: (int:  integer);
               false:(ptr:  integer);
             end;
  var
    cpuaddr : window;

  begin  (* main *)

    cpuaddr.int := pseudo_vdp_reg_7;

    cpuaddr.ptr := -30735;
       (* white on black *)

    writeln; writeln;
    writeln(' color modified ');

  end.
```

```
* Dette program booter UCSD pascal.
* Hvis man anvender både UCSD og
* ED/ASSM, er det praktisk at kunne
* resette til pascal uden det tradi-
* tionelle 'power down 10 sec.-
* power up'. Programmet kan evt
* lægges i Funnelweb menuen.  For at
* programmet kan fungere, må P-code
* kortet naturligvis være 'ON'.
* Programmet loades som DSKn.UCSD .
* Skift til UCSD system disk og an-
* giv programname : UCSD .
* Tak til ANDERS PERSSON for hans
* P-code memory map.      lha Nov-89

       DEF  UCSD
       AORG >A000
WORD1  TEXT '99'
WORD2  TEXT '/4'

UCSD   LWPI >8300
       MOV  @WORD1,@>38FA
       MOV  @WORD2,@>38FC
       BLWP @>0000
       END
```

ARCHIVER 3.03 FIX

Om du använder Archiver från april 89 eller tidigare tillsammans med Myarc Geneve så behöver du ändra enligt följande. Leta efter 04E08C00 och ersätt med D8018C00.

TIPS FROM THE TIGERCUB

#51

TIGERCUB SOFTWARE
156 Collingwood Ave.
Columbus, OH 43213
USA

I believe this word game is totally different from anything you have ever seen, and very challenging if you don't use the AID key. The first time you run it, pick option 3 to create a file of phrases and give it the file name COMPUTE. This will then become the computer's file, option 1, and you can create as many of your own files as you want. Recommend phrases of several to as many as 20 words - short ones are too difficult.

```
100 DIM W$(20):: DIM D$(20)
110 GOTO 150
120 Q$,K,S,Q,F$,E,FLAG,X,J,X
$,Y$,A,B,M$,DY$,V,A$(),C$,CH
,CH$,Y,W$(),L,M,D$(),F,Z,C,R
,H
130 CALL CHAR :: CALL KEY ::
 CALL SOUND :: CALL CLEAR ::
 CALL CHARPAT :: CALL COLOR
:: CALL SCREEN :: CALL VCHAR
 :: CALL SPRITE :: CALL LOCA
TE :: CALL DELSPRITE
140 !@P-
150 CALL CHAR(94,"3C4299A1A1
99423C"):: DISPLAY AT(2,1)ER
ASE ALL:"TIGERCUB SHUTTLESEA
RCH V.1.1":"":"^ Tigercub So
ftware for free":"distributi
on but no price"
160 DISPLAY AT(6,1):"or copy
ing fee to be charged":"":"I
f you should feel moved to":
"send me a few bucks for my"
:"work, I won't be offended!
"
170 DISPLAY AT(12,1):"Jim Pe
terson":"156 Collingwood Ave
.":"Columbus, OH 43213"
180 DISPLAY AT(16,5):"Instru
ctions? (Y/N) N" :: ACCEPT A
T(16,25)SIZE(-1)VALIDATE("YN
"):Q$ :: IF Q$="N" THEN 260
190 DISPLAY AT(2,1)ERASE ALL
:" The computer will display
 a":"phrase or saying concea
led":"within a grid of rando
m":"letters."
200 DISPLAY AT(6,1):" The wo
rds will be horizon-":"tal,
one word per line and":"on c
onsecutive lines, but":"not
necessarily beginning on"
210 DISPLAY AT(10,1):"the to
p line, and the phrase":"may
 `wrap around' from the":"bo
ttom row to the top."
220 DISPLAY AT(13,1):" You c
an find the phrase by":"shut
tling columns of letters":"u
p and down, looking for":"co
nsecutive rows with letter"
230 DISPLAY AT(17,1):"combin
ations that could be":"parts
 of words.":" A cheat key is
 available,":"if you are rea
lly stuck, but"
240 DISPLAY AT(21,1):"try no
t to use it!"
250 DISPLAY AT(23,8):"PRESS
ANY KEY" :: DISPLAY AT(23,8)
:"press any key" :: CALL KEY
(0,K,S):: IF S=0 THEN 250
260 DISPLAY AT(3,2)ERASE ALL
:"Do you want to - 1":"":" (
1) Solve a saving from my
   file?":"":" (2) Solve a p
hrase from       your file
?"
270 DISPLAY AT(11,2):"(3) Cr
eate a file of":"    phrase
s?":"":" (4) Have someone ty
pe in a      phrase to solve
?"
280 ACCEPT AT(3,19)SIZE(-1)V
ALIDATE(DIGIT):Q :: IF Q<1 O
R Q>4 THEN 280
290 ON Q GOTO 300,310,410,47
0
300 F$="1.COMPUTE" :: E=1 ::
 GOTO 320
310 DISPLAY AT(18,1):"Filena
me? DSK" :: ACCEPT AT(18,14)
:F$ :: E=2
320 ON ERROR 370
330 IF FLAG=1 THEN 350 :: FL
AG=1 :: OPEN #1:"DSK"&F$,FIX
```

```
ED,RELATIVE,INPUT :: ON ERRO
R STOP
340 INPUT #1,REC 0:X :: CLOS
E #1 :: FOR J=1 TO X :: X$=X
$&CHR$(J):: NEXT J :: Y$=X$
350 RANDOMIZE :: A=INT(RND*L
EN(Y$)+1):: B=ASC(SEG$(Y$,A,
1)):: Y$=SEG$(Y$,1,A-1)&SEG$
(Y$,A+1,255):: IF LEN(Y$)=0
THEN Y$=X$
360 OPEN #1:"DSK"&F$,FIXED,R
ELATIVE,INPUT :: ON ERROR ST
OP :: INPUT #1,REC B:M$ :: C
LOSE #1 :: GOTO 490
370 FOR J=1 TO 10 :: DISPLAY
 AT(20,1):"" :: DISPLAY AT(2
0,1):"CANNOT OPEN FILE!" ::
CALL SOUND(-99,110,5,-4,5)::
 NEXT J
380 ON ERROR 390 :: CLOSE #1
390 FLAG=0 :: INPUT "CHECK D
ISK AND DRIVE, PRESS ANY KEY
":DY$
400 IF E=1 THEN RETURN 260 E
LSE IF E=2 THEN RETURN 310 E
LSE RETURN 410
410 DISPLAY AT(8,1)ERASE ALL
:"Filename? DSK" :: ACCEPT A
T(8,14):F$
420 E=3 :: ON ERROR 370 :: O
PEN #1:"DSK"&F$,FIXED 124,RE
LATIVE,OUTPUT :: ON ERROR ST
OP :: X=0
430 DISPLAY AT(12,1):"Enter
END when finished":"":"":"Ty
pe phrases, not more than 20
 words and 124 characters"
440 X=X+1 :: ACCEPT M$ :: IF
 LEN(M$)>124 THEN PRINT "TOO
 LONG!" :: X=X-1 :: GOTO 440
450 IF M$<>"END" THEN PRINT
#1,REC X:M$ :: GOTO 440
460 PRINT #1,REC 0:X :: CLOS
E #1 :: GOTO 260
470 CALL KEY(3,K,S):: DISPLA
Y AT(12,1)ERASE ALL:"Type a
phrase of less than  20 word
s and press Enter"
480 ACCEPT M$ :: CALL CLEAR
490 DISPLAY AT(3,2)ERASE ALL
:"Choose skill level - 1":""
:" (1) All words begin in":"
     first column"
500 DISPLAY AT(8,2):"(2)`All
 words begin in same":"
column":"":" (3) Each word m
ay appear in":"     a differ
ent column"
510 DISPLAY AT(14,2):"(4) As
 No. 3 but AID key is":"
 disabled":"":" (5) Quit"
520 ACCEPT AT(3,23)SIZE(-1)V
ALIDATE(DIGIT):V :: IF V<1 O
R V>5 THEN 520 :: IF V=5 THE
N CALL CLEAR :: STOP
530 DISPLAY AT(12,6)ERASE AL
L:"SCRAMBLING....."
540 A$(1)="jkzae klmpr vgaho
 nceci sdufy bqijw astrf urd
sa nvjxe blbig trakv nobth w
ehey vnijo oherq umbmi rtika
 opleg nosve tarkh zeski "
550 A$(2)="!boiu m.fgt krac,
 pjip? tn-un osheg kar,q ibl
.o tons! idrix ?uhig ebarf u
ks,k ,,jhge vifyt kibrn taga
, .!ry lakle ilf.! inst"
560 C$=A$(1)&A$(2)
570 FOR CH=65 TO 90 :: CALL
CHARPAT(CH,CH$):: CALL CHAR(
CH+32,CH$):: NEXT CH :: CALL
 CHAR(42,"82444428281010")
580 CALL CHAR(143,"18243C4A4
A3C2418"):: CALL COLOR(14,16
,1)
590 M$=M$&" " :: Y=1
600 X=POS(M$," ",1):: W$(Y)=
SEG$(M$,1,X):: L=LEN(W$(Y)):
: M=MAX(M,L):: RANDOMIZE ::
W$(Y)=W$(Y)&SEG$(C$,INT(230*
RND+1),20-L)
610 Y=Y+1 :: IF Y=21 THEN 62
0 :: M$=SEG$(M$,X+1,255):: I
F LEN(M$)>0 THEN 600
620 FOR J=Y TO 20 :: W$(J)=S
EG$(C$,INT(230*RND+1),20)::
NEXT J
630 ON V GOTO 670,640,650,65
0
640 X=INT(RND*(20-M))+M+1 ::
 FOR J=1 TO Y :: W$(J)=SEG$(
W$(J),X,255)&SEG$(W$(J),1,X-
1):: NEXT J :: GOTO 670
650 FOR J=1 TO Y :: X=INT(RN
D*(20-M))+M+1 :: W$(J)=SEG$(
W$(J),X,255)&SEG$(W$(J),1,X-
1):: NEXT J :: GOTO 670
660 ! the string
670 FOR J=1 TO 20 :: FOR L=1
 TO 20 :: D$(J)=D$(J)&SEG$(W
$(L),J,1):: NEXT L :: NEXT J
680 IF V=1 THEN F=M ELSE F=2
0
690 FOR J=1 TO F :: Z=INT(20
*RND+1):: D$(J)=SEG$(D$(J),Z
,255)&SEG$(D$(J),1,Z-1):: NE
XT J
700 CALL CLEAR :: CALL SCREE
N(5):: FOR S=1 TO 13 :: CALL
 COLOR(S,5,16):: NEXT S :: C
ALL VCHAR(1,31,1,96)
710 CALL VCHAR(4,5,143,20)::
 CALL VCHAR(4,28,143,20)
720 FOR C=1 TO 20 :: FOR R=1
```

```
 TO 20 :: CALL VCHAR(R+3,C+6
,ASC(SEG$(D$(C),R,1))):: NEX
T R :: NEXT C
730 DISPLAY AT(1,1):"s&d to
select, e&x to scrollfctn 7
aid, fctn 8 restart"
740 H=1 :: C=48 :: CALL SPRI
TE(#1,42,7,18,C)
750 CALL KEY(3,K,S):: IF S=0
 THEN 750 ELSE ON POS("EXSD"
&CHR$(1)&CHR$(6),CHR$(K),1)+
1 GOTO 750,800,810,820,830,7
60,840
760 IF V=4 THEN 750
770 FOR S=5 TO 8 :: CALL COL
OR(S,16,5):: NEXT S
780 CALL KEY(3,K,S):: IF S=-
1 THEN 780
790 FOR S=5 TO 8 :: CALL COL
OR(S,5,16):: NEXT S :: GOTO
750
800 D$(H)=SEG$(D$(H),2,19)&S
EG$(D$(H),1,1):: FOR R=1 TO
20 :: CALL VCHAR(R+3,H+6,ASC
(SEG$(D$(H),R,1))):: NEXT R
:: GOTO 750
810 D$(H)=SEG$(D$(H),20,1)&S
EG$(D$(H),1,19):: FOR R=1 TO
 20 :: CALL VCHAR(R+3,H+6,AS
C(SEG$(D$(H),R,1))):: NEXT R
 :: GOTO 750
820 C=C-8-(C=48)*8 :: H=C/8-
5 :: CALL LOCATE(#1,18,C)::
GOTO 750
830 C=C+8+(C=200)*8 :: H=C/8
-5 :: CALL LOCATE(#1,18,C)::
 GOTO 750
840 CALL CLEAR :: FOR J=1 TO
 20 :: D$(J)="" :: NEXT J ::
 M=0 :: CALL DELSPRITE(#1)::
 IF Q=1 OR Q=2 THEN 350 ELSE
 470
```

Here are three screen display subprograms of the type you will find on my Nuts and Bolts disks. Note that subprograms can read DATA from the main program. The double colons in the DATA statement cause input of null strings of data for spacing between the lines. The M$() in the subprogram parameter' lists is necessary, even though the array is not passed from the main program, in order to DIMension the array in the subprogram - unless you prefer to place the DIM in the subprogram itself. T is the number of DATA items to be read.

```
100 CALL CLEAR
110 DATA THIS IS A DEMO,,OF
THREE SCREEN PRINTING,,SUBPR
OGRAMS PUBLISHED IN,,TIPS FR
OM THE TIGERCUB,,No. 51,,BY
TIGERCUB SOFTWARE
120 DIM M$(11):: CALL DOWNPR
INT(M$(),11):: FOR D=1 TO 10
00 :: NEXT D :: CALL CLEAR :
: RESTORE 110 :: CALL DIAGPR
INT(M$(),11)
130 FOR D=1 TO 1000 :: NEXT
D :: CALL CLEAR :: RESTORE 1
10 :: CALL INWARD(M$(),11)
1000 SUB DOWNPRINT(M$(),T)
1001 FOR J=1 TO T :: READ M$
(J):: L=INT(LEN(M$(J))+.5)::
 M$(J)=RPT$(" ",14-INT(L/2))
&M$(J):: M$(J)=M$(J)&RPT$("
",28-LEN(M$(J))):: NEXT J
1002 FOR J=1 TO 28 :: FOR L=
1 TO T
1003 DISPLAY AT(L,1):SEG$(M$
(L),1,J):: NEXT L
1004 NEXT J :: SUBEND
2000 SUB INWARD(M$(),T):: FO
R J=1 TO T :: READ M$(J):: N
EXT J :: R=1 :: FOR A=1 TO T
2001 L=INT(LEN(M$(A))):: F=1
3-L/2 :: G=L+F
2002 FOR J=1 TO INT(L/2+.5):
: DISPLAY AT(R,F+1):SEG$(M$(
A),J,1);:: DISPLAY AT(R,G):S
EG$(M$(A),L-J+1,1);:: F=F+1
:: G=G-1 :: NEXT J :: R=R+1
:: NEXT A :: SUBEND
3000 SUB DIAGPRINT(M$(),T)::
 FOR J=1 TO T :: READ M$(J):
: L=INT(LEN(M$(J))+.5):: M$(
J)=RPT$(" ",14-(L/2))&M$(J):
: M$(J)=M$(J)&RPT$(" ",28-LE
N(M$(J))):: NEXT J
3001 FOR J=1 TO 28+L :: FOR
L=1 TO T
3002 IF J<L THEN 3007
3003 DISPLAY AT(L,1):SEG$(M$
(L),1,J-L):: NEXT L
3004 NEXT J :: SUBEND
```

Just in case you didn't know - to jump directly to the first or last line in a TI-Writer file, use FCTN 9 and S(earch) and 1 for the first line or E for the last.

MEMORY ALMOST FULL...

Jim Peterson

SWEDLOW EXTENDED BASIC

```
X   X   BBBB      # 1 to 6
 X X    B   B
  X     BBBB      By
 X X    B   B     Jim
X   X   BBBB      Swedlow
```

(This article is a summary of six XB columns that originally appeared in the User Group of Orange County, California ROM)

IF THEN

The incompatibility between FOR NEXT and IF THEN statments can cause your program to be awkard. The MAX and MIN statements will often work. For example, instead of this:

```
IF A<6 THEN A=6
```

Try:

```
A=MAX(A,6)
```

TOO MANY GO TO's

I was looking at a text only adventure game and found the following code:

```
370 FOR B=1 TO 58 ::
    IF A(B)=L THEN C=C+1 ::
    GOTO 372
371 NEXT B ::
    GOTO 374
372 IF C=1 then PRINT "You See:"
373 PRINT A$(B) ::
    GOTO 371
374 ! Program Continues
```

Note that in line 370 the program leaves the loop to print the message and then returns. Then it skips over the print instructions to continue.

This is simpler:

```
370 FOR B=1 TO 58 ::
    IF A(B)=L THEN C=C+1 :: IF
    C=1 THEN PRINT "You See:":
    A$(B) ELSE PRINT A$(B)
371 NEXT B ! Program Continues
```

This saves 25 bytes of memory

MORE IF THEN

Suppose that C$ can be only "Y" or "N" and that you want to write a routine to change C$ without knowing its current value.

You might do this:

```
100 IF C$="Y" THEN C$="N" ELSE
    C$="Y"
```

You don't need to use IF THEN:

```
100 C$=CHR$(167-ASC(C$))
```

TEACH YOURSELF EXTENDED BASIC: This was released by TI to support the XB cartridge.

A working knowledge of BASIC is necessary to understand this material. If you are new to XB or if you have not explored all of XB's features, this a a good tutorial. Even if you are an old hand you might learn something new (see next item!).

The material is clear and presents some information not in the XB book (although most is a repetition). It is primarily text but there are examples, expecially for sprites.

LISTING TO DISK: In the XB book it suggests that you can list a program to a device but the material points you toward a printer. TEACH YOURSELF XB adds that you can list a program to disk. The command is:

```
LIST "DSK1.TEST"
```

The program is now saved on disk exactly as you last saw it on the screen. The file paramaters are DISPLAY, VARIABLE 80.

Since those are the paramaters for a TI WRITER file you can load the file onto TI WRITER. Why? Well, it could

be helpfull when doing a newsletter. Also, the FIND STRING command could help you locate something in a long program. Mainly, however, just to see what you could do.

I have not found a way to get the file back to program status. If you could do that there might be some interesting possibilities.

NB: This also works in BASIC. Also, some symbols may cause strange things to happen when you run it thru the Text Formatter.

DISK MENU PROGRAM: This month's program will read your disk and display a menu on the screen. After you choose a program, it is loaded and ran. If you save this on your disk as LOAD, it will auto boot when you opt for XB.

This program requires one disk drive and the memory expansion. In a month or so, I'll do a disk menu program that does not require memory expansion.

When you enter this program, save it to disk BEFORE running it. If you make an error in line 220 or 230, the system could lock up and the program would be lost.

LINES 100-150 comprise the header. This program is based on one published in the Pomona Users Group newsletter.

LINES 160-190 read the programs on the disk and display them on the screen.

LINES 200-210 wait for the user to select a program and then validates the user's selection.

LINES 220-240 change line 240 to have the selected program name rather than "1234567890" and then run that program.

After you get this working, try entering BREAK 240 before running it. When the program stops, LIST 240 to see the change.

```
100 ! DISK MENU PROGRAM
110 ! VERSION XB.1.2
120 ! 29 DEC 84
130 ! FROM THE POMONA (CA)
      99 UG
140 ! MODIFIED BY J. SWEDLOW
150 !
160 DIM A$(18):: OPEN #1:"DS
K1.",RELATIVE,INPUT ,INTERNA
L :: INPUT #1:D$,A,B,C :: DI
SPLAY AT(1,1)ERASE ALL:"DISK
 ";D$;" * FREE";C: :"Press F
or"
170 INPUT #1:D$,A,B,C :: IF
D$="" THEN 190 ELSE IF ABS(A
)<>5 OR D$="LOAD" THEN 170
180 S=S+1 :: A$(S)=D$ :: IF
S<18 THEN DISPLAY AT(S+4,3):
CHR$(S+64);"   ";D$ :: GOTO
170 ELSE DISPLAY AT(22,3):"R
 To Continue"
190 DISPLAY AT(24,1)BEEP:"Pr
ess <ERASE> to stop"
200 CALL KEY(3,A,B):: IF A=7
 THEN CLOSE #1 :: STOP ELSE
IF A<65 OR A>64+S THEN 200 E
LSE A=A-64
210 IF A=18 AND D$<>"" THEN
CALL HCHAR(5,1,32,32*20):: S
=0 :: GOTO 180 ELSE D$="DSK1
."&A$(A):: DISPLAY AT(24,1)B
EEP:"Loading ";A$(A):: CLOSE
 #1
220 CALL INIT :: CALL PEEK(-
31952,A,B):: CALL PEEK(A*256
+B-65534,A,B):: C=A*256+B-65
534 :: CALL LOAD(C,LEN(D$))
230 FOR I=1 TO LEN(D$):: CAL
L LOAD(C+I,ASC(SEG$(D$,I,1))
):: NEXT I :: CALL LOAD(C+I,
0)
240 RUN "DSKX.1234567890"
```

IF THEN ELSE - AGAIN: Another thing I found in the 'Teach Yourself XB' tutorial is how XB matches ELSE's with IF's.

Each ELSE is paired with the last unmatched IF. For example:

```
IF A THEN B :: IF C THEN D
ELSE E ELSE F
```

In words: If A is true, do B and then test C. If C is true, do D. If C is false, do E. If A is false, do F.

When I write a program, I generally start with something simple and then

modify it until it does all that I want it to do. I'm working on a revised program that will delete files, print or display a disk catalog and run programs. Look for Version 2.0 soon.

LOADMAKER: This month's program is a Disk Menu program for those who do not have memory expansion.

A note about the origin of this program. It is similar in concept to POOR MAN's LOADER, which appeared in 99'er Magazine. Both read a disk's directory and then create a LOAD program on the disk in merge format. The difference between the two programs is excution. PML uses DEF statements while LOADMAKER uses subroutines.

Save this program as LOADMAKER. Initially and whenever programs are added or deleted, run LOADMAKER and then follow the on screen instructions to recapture your LOAD program.

As before, save your program on disk before running as some errors can cause your system to lock up. If the program that LOADMAKER writes is defective, use the following description to find the area of probable error and compare character by character to find the problem.

LINES 100-140 are the header.

LINE 150 sets up the video screen and does necessary intializing.

LINES 160-210 read the disk for programs. If no programs are found, line 210 stops excution. If there are more than 26 programs, the user is given a choice in lines 190-200 of either aborting or continuing.

LINE 220 closes the disk reading file and opens the file for writing the LOAD program (under the merge file name 'XXX').

LINES 230-240 create LOAD line 100 which is the sole header line.

LINES 250-260 write LOAD line 110 which clears the CRT and displays the top line of the screen.

LINES 270-280 make the LOAD lines that display the menu on the CRT. In line 270, the display format is set: 1 column in the center if less than 19 programs, 2 columns if more.

LINES 290-310 create a LOAD line that displays 'Press your choice' on CRT line 24.

LINES 320-350 make the CALL KEY line for selecting a program.

LINES 360-390 do a LOAD line that DISPLAY's 'Loading' followed by a GOTO statement that sends the program to the correct RUN line.

LINES 400-410 write the LOAD lines that display the name of the selected program and then RUN it.

LINE 420 saves the final line on disk as well as the end of file marker and then closes the disk file.

LINE 430 displays instructions for retrieving the LOAD program.

LINE 440 is a subroutine that increases the line number counter (B), prints the previous line on disk and displays the new line number on your CRT.

LINE 450 adds the line number in base 256 to the line.

LINE 460 is a subroutine that starts a new line and and then adds 'DISPLAY AT (F,G)' to that line.

LINE 470 is a subroutine that adds the number F to a line.

Good luck!

```
100 ! LOADMAKER
110 ! VERSION XB.1.1
120 ! 29 DEC 84
130 ! BY JIM SWEDLOW
140 !
150 DISPLAY AT(10,10)ERASE A
LL:"LOADMAKER": : : : : :"In
itializing . . ." :: @=1 ::
DIM A$(26),A(26):: B$=CHR$(1
82)&CHR$(181)&CHR$(199):: B=
100 :: C$=CHR$(179)
160 OPEN #@:"DSK1.",INPUT ,I
```

```
NTERNAL,RELATIVE :: INPUT #@
:A$(C),D,E,F :: DISPLAY AT(1
6,@):"Disk ";A$(C);" * Free"
;F :: A(C)=LEN(A$(C))
170 INPUT #@:D$,D,E,F :: IF
D$="" THEN 210 ELSE IF ABS(D
)<>5 OR D$="LOAD" THEN 170
180 C=C+@ :: IF C<27 THEN A$
(C)=D$ :: A(C)=LEN(D$):: DIS
PLAY AT(17,@):"Reading:  ";D
$ :: GOTO 170
190 DISPLAY AT(16,@)BEEP:"Yo
ur disk has more than 26  pr
ograms.  Do you want to  pr
oceede with the first 26  pr
ograms?  Press Y or N."
200 CALL KEY(3,F,D):: IF F=7
8 THEN CLOSE #@ :: STOP ELSE
 IF F<>89 THEN 200
210 IF C=0 THEN DISPLAY AT(1
6,@)BEEP:"No programs were f
ound on  this disk." :: CLO
SE #@ :: STOP
220 CLOSE #@ :: DISPLAY AT(1
6,@):"Disk reading completed
": : : : : :: OPEN #@:"DSK1.
XXX",VARIABLE 163,DISPLAY ,O
UTPUT
230 DISPLAY AT(16,@):"  Mak
ing LOAD line 100" :: D$=CHR
$(0)&CHR$(100)&CHR$(131)&CHR
$(32)&CHR$(80)&CHR$(82)&CHR$
(79)&CHR$(71)&CHR$(82)&CHR$(
65)&CHR$(77)
240 D$=D$&CHR$(32)&CHR$(76)&
CHR$(79)&CHR$(65)&CHR$(68)&C
HR$(69)&CHR$(82)
250 G,F=@ :: GOSUB 460 :: D$
=D$&CHR$(182)&CHR$(239)&CHR$
(236)
260 D$=D$&CHR$(181)&CHR$(199
)&CHR$(7+A(0))&CHR$(68)&CHR$
(105)&CHR$(115)&CHR$(107)&CH
R$(32)&CHR$(42)&CHR$(32)&A$(
0)
270 G=@-7*(C<19):: E=2 :: FO
R D=@ TO C :: E=E-(D>C/2)*(G
=@)*INT(C/2):: G=G+(D>C/2)*(
G=@)*14 :: IF D=C AND G=15 A
ND C/2<>INT(C/2)THEN G=8
280 F=E+D :: GOSUB 460 :: D$
=D$&B$&CHR$(A(D)+3)&CHR$(64+
D)&CHR$(32)&CHR$(32)&A$(D)::
 NEXT D
290 F=24 :: G=@ :: GOSUB 460
 :: D$=D$&CHR$(182)&CHR$(238
)&CHR$(181)&CHR$(199)&CHR$(1
7)
300 D$=D$&CHR$(80)&CHR$(114)
&CHR$(101)&CHR$(115)&CHR$(11
5)&CHR$(32)&CHR$(121)&CHR$(1
11)&CHR$(117)
310 D$=D$&CHR$(114)&CHR$(32)
&CHR$(99)&CHR$(104)&CHR$(111
)&CHR$(105)&CHR$(99)&CHR$(10
1)
320 GOSUB 440 :: D$=D$&CHR$(
157)&CHR$(200)&CHR$(3)&CHR$(
75)&CHR$(69)&CHR$(89)&CHR$(1
83)
330 D$=D$&CHR$(200)&CHR$(@)&
CHR$(51)&C$&CHR$(75)&C$&CHR$
(83)&CHR$(182)&CHR$(130)
340 D$=D$&CHR$(132)&CHR$(75)
&CHR$(191)&CHR$(200)&CHR$(2)
&CHR$(54)&CHR$(53)&CHR$(186)
&CHR$(75)&CHR$(192)
350 F=64+C :: GOSUB 470 :: D
$=D$&CHR$(176)&CHR$(201):: G
OSUB 450
360 F=24 :: GOSUB 460 :: D$=
D$&B$&CHR$(7)&CHR$(76)&CHR$(
111)&CHR$(97)&CHR$(100)&CHR$
(105)&CHR$(110)&CHR$(103)
370 D$=D$&CHR$(130)&CHR$(155
)&CHR$(75)&CHR$(194)&CHR$(20
0)&CHR$(2)&CHR$(54)&CHR$(52)
&CHR$(134)
380 D=B :: FOR B=B+10 TO B+1
0*C STEP 10 :: IF B>D+10 THE
N D$=D$&C$
390 D$=D$&CHR$(201):: GOSUB
450 :: NEXT B :: B=D
400 FOR D=@ TO C :: F=24 ::
G=9 :: GOSUB 460 :: D$=D$&B$
&CHR$(A(D))&A$(D)&CHR$(130)&
CHR$(169)&CHR$(199)
410 D$=D$&CHR$(A(D)+5)&CHR$(
68)&CHR$(83)&CHR$(75)&CHR$(4
9)&CHR$(46)&A$(D):: NEXT D
420 PRINT #@:D$&CHR$(0):CHR$
(255)&CHR$(255):: CLOSE #@
430 DISPLAY AT(16,@)BEEP:"LO
AD program made!": :"Enter t
he following commands": :"
>NEW":"  >MERGE DSK1.XXX":"
 >SAVE DSK1.LOAD":"  >RUN" :
: STOP
440 B=B+10 :: PRINT #@:D$&CH
R$(0):: D$="" :: DISPLAY AT(
16,20):B
450 D$=D$&CHR$(INT(B/256))&C
HR$(B-256*INT(B/256)):: RETU
RN
460 GOSUB 440 :: D$=D$&CHR$(
162)&CHR$(240)&CHR$(183):: G
OSUB 470 :: D$=D$&C$ :: F=G
470 IF F<10 THEN D$=D$&CHR$(
200)&CHR$(@)&CHR$(48+F):: RE
TURN ELSE D$=D$&CHR$(200)&CH
R$(2)&CHR$(48+INT(F/10))&CHR
$(48+F-10*INT(F/10)):: RETUR
N
```

TI-WRITER ONCE AGAIN

Tony McGovern - HV 99 - Australia

Several years ago I wrote a HV99 Newsletter article entitled "TI-Writer under the Hood". As far as I know, this general overview of its Editor's internal operation, as distinct from how to use it articles, remains the only published discussion seen here of the Editor's internal workings. This is not a very good reflection on the state of information sharing in the TI-99 community, as quite a few people must have studied its workings over the years, even if only a few have ever seriously attacked it.

Recently I resumed work on a substantial rewrite of the Editor in preparation for updating Funnelweb to Vn 4.20, and while the details are fresh in mind I will write about the operation of the buffer manager in the Editor. For a gentle introduction dig out your old HV99 and read the original article again. This time we will go into a lot more detail, at the level of talking about the routines used, even if we still stay clear of detailed coding.

To remind you of how it goes, the hi-mem segment of CPU RAM contains some buffers, the buffer manager code itself, and starting at >A410 the pile of squished lines growing upwards and the line table growing downwards from >FFC8. When these meet up you have the dreaded "Text Buffer Full" condition. Each entry in the line table is a pointer to the squished text record in the Edit buffer pile corresponding to that line number used as an index into the line table. Records are usually added to the top of the Edit buffer, and if a line is deleted it is removed and the pile collapsed down so there are no gaps left.

The buffer manager code as bequeathed to us by TI contains 4 major functions called as BLWP procedures and a small cast of supporting routines. The main procedures are, to give them TI's labelling, GETLN, INSTLN, UPDTLN, and DELTLN. As you can see from the names these are line oriented. The minor routines are

BINDEC - converts a line number in binary form to the 4 digit decimal you see down the side of your screen.

CKLIN - given the line number it returns the line table and edit buffer pointers and the length of the record.

CKROOM - this is the one that returns the 'Text Buffer Full' if it isn't satisfied.

MOVMEM - removes a record from the edit buffer pile by dropping down all the records above it in a mass move of the Edit buffer contents.

SQUISH - this is the one we talked about in last month's Assembly programming contest, that does run length encoding of text lines before they are placed on the top of the Edit buffer pile. SQUISH, and the unsquisher in GETLN, have had a functional update in that the encoding has been improved slightly to give greater apparent buffer capacity. The improvement varies depending on the document file from none at all, through about 3% for a file like this one , to the 5-10% range for assembly source files. Buffer capacity is never decreased. As before buffer encoding is purely internal to the Editor.

These services are called on by the main routines as needed. I will start from the simplest function (not necessarily the simplest code) in looking at how these work. All the buffer management functions in the standard TI-Writer are performed by these routines, and as we shall see, in very inefficient fashion because the basic services are all line oriented.

GETLN - this procedure fetches a specified line from the Edit buffer for purposes such as refreshing the screen buffer, for Find String etc,

and for Saving to disk. It uses CKLIN to find the record in the Edit buffer and then unsquishes the record as part of the service. At the level of detail coding this was the sloppiest effort by TI on any of the routines. I have left the function alone but have refined the code extensively. At the intermediate level the original code is inefficient in that it copies the line from the Edit buffer into the squish buffer before unsquishing to the destination buffer, where it could have been done directly from buffer to destination and now is of course.

INSTLN -- whenever a new line has to be added to the buffer, the INSert-LiNe procedure is called. This uses SQUISH to encode the line into the squish buffer, and it is then copied at the top of the Edit buffer pile, with CKROOM testing for "Buffer Full" before this is done. Then the new pointer is inserted into the line table and all pointers above it shuffled along to make room. Nothing too much needed updating in this routine.

UPDTLN -- when you edit a line and changes are made, it has to be replaced in the Edit buffer by the UP-DateLiNe procedure. This fetches the data for the existing line with CKLIN, SQUISHes the new line, and then checks the length against that of the existing line. If this is unchanged the record is written back over the earlier version without further ado. If the lengths don't match then it has to delete the existing line and insert the new line. A subtle improvement on the original is that it now uses only the increase in length when it calls CKROOM.

DELTLN -- when you use ‹fctn-3› or ‹ctrl-N› to delete a single line DELeTeLiNe does the job. For normal documents this is the most expensive operation of all, because when a line is deleted from the Edit buffer the whole buffer above where that line was stored must be moved down. The reference in the line table must also be deleted and the rest of the table moved along, with pointers to any record that has been moved adjusted appropriately. Yes, all that goes on every time you delete or even just edit a line. Since the line table has only a 2-byte word aligned entry for each line (to give the CPU memory address of the actual record) it is usually much shorter than the the Edit buffer. However just how much of the Edit buffer has to be moved depends on past history. A well ordered buffer is one in which the records start from the bottom at >A410 in order of their line numbers. A file freshly loaded from disk is in this state. Editing any line enough to change its length will, as we have seen already, end up with it pulled out of sequence and placed at the top of the pile.

TI-Writer uses these buffer manager routines in the form that we have described them, always in a line at a time fashion. The most annoyingly slow function of all in TI-Writer is Delete Lines and this was the one I decided to attack first as it is also called by other functions. Why is it so slow ? The reason is that each time a line is deleted, it is done as a separate task and a complete mass move of the Edit buffer and update of the line table is performed for deletion of that line. Code polishing improved this noticeably, but a new approach is clearly needed. So a new procedure was added.

DLBLOC -- when given a range of line numbers to delete this keeps checking successive lines to see if they form a contiguous block in the Edit buffer. Each such block is deleted as a single unit from both Edit buffer and line table. So with a well ordered buffer even large blocks are now deleted almost instantaneously. You could imagine a large buffer in the opposite of a well ordered state - then the process would proceed a line at a time, but usually there is some degree of block ordering. What you can see is that TI's original method is guaranteed to give the worst possible result every time. Oh well, they invented GPL too! DLBLOC is now also called from Reformat when it has finished making a reformatted copy of a paragraph to get rid of the original paragraph, and so Reformat is now speedier.

Move Lines and Copy Lines are also very much slower in TI-Writer than one might reasonably expect them to be. Once again this is because line by line services are used to construct block related functions. Apart from speed a bad failing of Move Lines is that it can give rise to a "Text Buffer Full" condition. If you think about it this should not occur as no text is being added. It happens because TI-Writer implements Move Lines by first doing a Copy Lines - which is where the full buffer disaster may strike, and then deletss the original lines, adding insult to injury by doing it one at a time always. At this stage DLBLOC had already speeded things up very noticeably, but clearly something much better is needed. In principle all that is necessary is to shuffle around blocks of the line table, and the Edit buffer need not be altered at all. So a new general purpose support routine, MVLINS, was written that could be called as whole or part of buffer functions.

MVLINS -- this is a routine that does the line table block shuffle. It is written as a generic routine, and the block limits are pre-ordered by the calling procedures, William came off his Amiga for long enough to make an initial suggestion on how to do it in linear time in block size, and I think the resulting code is the most elegant piece of 9900 code I have yet written.

TI-Writer did Copy Lines by using GETLN to fetch each line and INSTLN to add a copy to the Edit buffer and line table. Once again this is about as inefficient a method as could be dreamed up, and made worse by all the unsquishing and resquishing involved. A further bad feature is that if the buffer fills during the process you are left with a partially copied mess.

CPBLOC -- the obvious solution here is to do direct copying of the lines from their existing position in the Edit buffer to append them to the end of the buffer, and to add new pointers to the end of the line table. In this process a new entry point CKCOPY to CKROOM is invoked which checks for room in the buffer before copying each line. At this stage the copy process can still be abandoned leaving everything as it was since nothing has yet been done inside the previous Edit buffer and line table limits - only additions to the ends. If all is satisfactory then the end pointers for the buffer and line table are updated. The Edit buffer is now in fine shape but we have a whole block of line pointers out of position, unless they were originally meant to be added at the end. But all it takes now is to call MVLINS with the right pointers and the job is all done.

An aspect of Copy Lines that bears a little thinking about is that the new approach would make it possible to do a Copy Lines with the "After" line number in between the "From" and "To" line numbers. This could be done because the copy of the block is made before any reinsertions are made, but it is not implemented because the Copy Lines and Move Lines call a common routine to take down the line numbers and do range checking. The conditions for Move Lines are more restrictive of course.

A further modification to the buffer manager code which has not yet been incorporated, is to abandon the fixed squish buffer and float it at the top of the Edit buffer, with appropriate modifications to the code. There is no net gain in buffer capacity, but since lines are always added to the top of the Edit buffer (except for the direct replacement in UPDTLN) it will mean that the squished line is already in place without having to be copied into position again. Strictly speaking it is only the direct replacement path in UPDTLN that forces provision of an explicit squish buffer, since you cannot tell that a replacement line is the same length until it has been squished. The other major area besides the buffer manager that will bear looking at is the Load/Save module. The change just discussed will help this along by reducing overhead in the file reading process. Another way to help it along was discussed as an aside in the Living with Spiders series on interfacing to Funnelweb published in the

HV99 newsletter many months ago. This is the technique used in the E/A Utilities object loader, and extensively exploited in the LINE-HUNTER utility program that has been part of the Funnelweb package for what seems like very long time now. The idea is to use the full DSRLNK only for the initial access to a device, usually a disk DSR, saving the CRU base address and the entry address, and using these directly for future accesses to the same file. Linehunter gets its speed by keeping track of these separately for both MAIN and COPY files.

The revised buffer manager has been incorporated in the 40 column version also, and the speed-up is such that it now seems impossible to lose a keystroke on end of line automatic wordwrap. It is still just possible to lose a repeated character in the AVPC version, but remember that this has to write up twice as much on the screen in between each keystroke, and do it via the 8-bit bus to the PE-Box as well. I must compare this against the Mechatronics 80-col unit sometime. Another upgrade to the 40 column version is that <ctrl-Y> now functions as a full margin release as has been the case in the AVPC version.

Another new feature which is proving its worth right now in writing this document is the provision of dual tab settings. The command line entry "DF" is now obsolete as SD does the job better, and has been replaced by "ST" for Swap Tabs, which swaps the current tab set with an alternate set. So when I want to write inset paragraphs all I have to do is go to command mode, ST, and Enter. Both tab sets are now saved with the document and if an old document with only one tab set is read in this is made the initial alternate set also. In the AVPC version the current tabs show on the line 26 ruler when called for editing by T or ST. Pressing <ctrl-=> cleans the ruler line to numbers only. I have slso added a NTSC/PAL toggle from <ctrl-N> in the command mode. The control key may change but it seems a worthwhile inclusion (assuming your monitor will handle it) as I find it easier on the eyes to read the less elongated screen characters with less "lininess" to the display. Interlace mode becomes pretty horrendous though at 50 Hz frame rate. I still have to sort out why the output from the 9938 seems so much worse than standard TV.

Users of the Programmer's Editor will find a whole new mode of operation. In the Word Processor pressing <ctrl-0> toggles the cursor between the solid word-wrap shape and the hollow cursor to indicate fixed mode. Previously the Pgm/Ed allowed only a modified form of fixed mode to give E/A compatibility of saved files and to make sure no reformat could ever be done on program source files. Now <ctrl-0> toggles in a diamond cursor to indicate that you are in assembly source mode (ASMode from now on). It has always been a nuisance switching between upper case for the assembly code itself and to lower case for the comments. In ASMode you may leave the alpha lock off, and type both code and comments on the same line in lower case, with <shift> needed only for capitals in comments as in normal typing. It doesn't matter that the assembly instructions are in lower case because ASMode converts them to upper case before updating the line in the Edit buffer. I recall from the brief look we had at My-Word for the Geneve a year or more ago that a "C" could be added to the tab line to control conversion to upper case. Atrax Robustus would never be satisfied with anything so inflexible. What ASMode does is to parse partially each line as an assembler source line. Comment lines are ignored, and the line parsed for the first three fields (two if no label starts in the first column) separated by blanks. Some error checking is done to catch a couple of common typing errors that can be difficult to find by eye. It indicates these by giving a little audible error bloop as the line is converted, and ceases case conversion at that point to make it more obvious. In the Opcode field only alpha characters are allowed as all legal opcodes and directives contain only these - it dislikes things like L1 or SOCB. In the Operand field ASMode leaves any entry between single quotes unalter-

ed - it is not allowed to touch your text data strings. It also checks for an uneven number of single quotes. Further it notices any "." not part of a text string and lets you know - as in LI R8.6 for instance. There are a couple of minor bits of anomalous behavior, but they are only incidental and very occasional, and a fix is not a high priority in assigning code space. It bloops on the filenames in COPY directives, which in standard TI Assembler syntax are indicated by normal double quote marks and usually contain one or more "."s not isolated within single quotes. Some opcodes do not have an operand field, and if these also carry comments the first word of the comment, if any, will be treated as if it were the operand field and converted to upper case. Most commonly these are the RTWP opcode and the RT pseudo-instruction. Usually the function of these is so limited and obvious that they don't deserve a specific comment, but no harm is done anyway. In practice neither problem is a real bother, but if they do grate on any sensitivity just turn ASMode off, or never turn it on in the first place.

This new ASMode code is written over the Reformat code on the way in so that it will no longer be reasonable to have immediate re-entry to the Editor interchangable between Word Processor and Program Editor. In fact instant reentry is a very marginal feature at best, and may end up being omitted altogether if it gets in the road of more desirable features.

KRYMPA ASSEMBLER DEL 4

av Tony McGovern, Australien

En annan sorts sträng som ibland uppträder, använder inte en byte eller ett ord som längdindikator i början, utan använder en noll-byte för att markera slutet av strängen. Detta betyder att strängen kan vara av godtycklig längd men kan inte innehålla några noll-bytes som en del av strängen. Åter igen tittar vi på problemet med att flytta en sträng från en känd adress till en annan.

```
      LI   R0,STR1
      LI   R1,STR2
LOOP  MOVB *R0+,*R1+
      JNE  LOOP
```

Du kan se varför denna sorts sträng används vartefter bytes flyttas och sätter slutvillkoret när de har flyttats, och en separat nedräkning behövs inte. Koden ovan tar bara 6 ord och använder 2 register, men det är möjligt att göra det ännu bättre genom att använda indexerad adressering igen.

```
      SETO R1
LOOP  INC  R1
      MOVB @STR1(R1),@STR2(R1)
      JNE  LOOP
```

SETO initierar index så att under första varvet av loopen flyttas den första byten. Detta använder fortfarande 6 ord men endast ett register. I båda fallen är noll-byten den sista som skall flyttas och behöver inte särbehandlas.

Medan vi håller på, antag att vi läser ett bestämt antal bytes, t.ex. från GROM och vi vill avsluta dem med en noll-byte för att åstadkomma denna typ av sträng i CPU-RAM.

```
LOOP  MOVB @GRMRD,*R1+
      DEC  R2
      JGT  LOOP
      MOVB @NULL,*R1
```

NULL är label för en noll-databyte. Du kan spara en byte genom att ersätta den sista in struktionen med

```
      SB   *R1,*R1
```

Subtraktion av en byte från sig själv är ett bekvämt sätt att sudda en enstaka byte eftersom CLR-instruktionen endast arbetar med hela ord. I detta fall skulle

```
      MOVB R2,*R1
```

också göra jobbet eftersom villkoret för avbrytande av loopen lämnar R2 med ett noll-ord. Om någon försöker att läsa koden vid ett senare tillfälle, och detta gäller även dig själv, så är den föregående formen att föredra eftersom den gör det som skall göras på ett tydligare sätt.