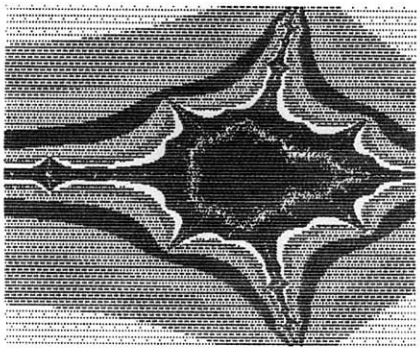
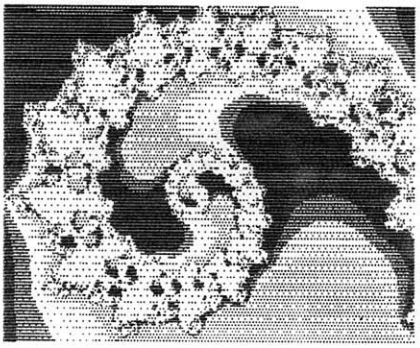
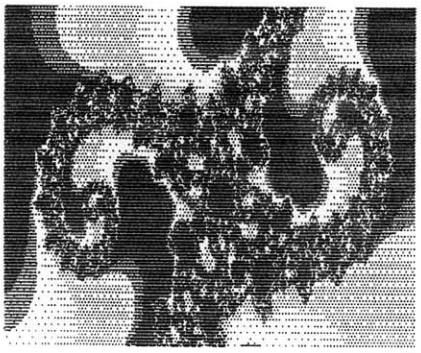
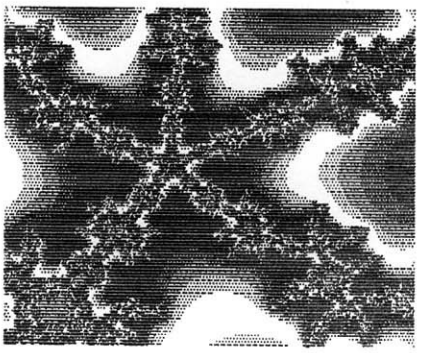
PROGRAM BITEN 91-1









ISSN 0281-1146

KALLELSE TILL ÅRSMÖTE 16 MARS 1991

Medlemmar i föreningen Programbiten kallas härmed till Årsmöte Lördagen den 16 mars 1991 kl.13.00:
Albatrossvägen 76, 1 tr.ned, Haninge
Pendeltåg + buss söder om Stockholm.
(kontakta Kent Edgardh 08-777 29 44)
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 1990. Verksamhetsberättelse och kassarapport delas ut på mötet.

6. Revisorernas berättelse för 1990

7. Om ansvarsfrihet för styrelsen för 1990

8. Val av styrelse för 1991. Årmötet väljer enligt stadgarna en styrelse bestående av ordförande och kassör, vilka samtliga utses till befattning på årsmötet, samt därtill tre och högst sju övriga ledamöter.

9. Val av revisorer för 1991. Mötet väljer två revisorer och en suppleant.

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

11. Om årsavgift 1991.

12. Årmötets avslutas.

Därefter samlas den nyvalda styrelsen till styrelsemöte. ■

Sänd in dina program och artiklar till redaktören (tel. 08-771 05 69):

Jan Alexandersson
Springarvägen 5, 3 tr
S-142 61 TRÅNGSUND
Sverige

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.

SAMLINGSSKIVA FRÅN PROGRAMBITEN 1990

Du kan beställa en samlingsskiva från PB-90 för 30 kr som betalas till föreningens postgirokonto 19 83 00-6.

Du kan även beställa samlingsskivor från NN-83, PB-84, PB-85, PB-86, PB-87, PB-88 och PB-89.

```
PROGBIT-90
FIL=42 LED=5 ANV=355
filnamn     sekt typ       längd

BASKONVERT     4 PROGRAM     583
CHARFIX       19 DIS/VAR     163
COINC/XB       2 PROGRAM     161
DATAFIL-A      2 PROGRAM     162
DATAFIL-B      2 PROGRAM     194
DATAFIL-C      2 PROGRAM     102
DATAFIL-D      4 PROGRAM     657
DATAFIL-E      5 PROGRAM     866
DATAFIL-F      2 PROGRAM     145
DATAFIL-G      2 PROGRAM     102
DATAFIL-H      2 PROGRAM     102
DATAFIL-I      2 PROGRAM     129
DATAFIL-J      3 PROGRAM     276
DATAFIL-K      3 PROGRAM     400
DATAWRITER     7 PROGRAM    1390
DEMO74         6 DIS/VAR      80
DISKMENU       5 PROGRAM     823
FIX            2 PROGRAM     204
FIX/DOC        9 DIS/VAR      80
LOADMAKER     12 PROGRAM    2712
NUTRITION      4 PROGRAM     733
ORACLE         7 PROGRAM    1524
OTHELLO       27 PROGRAM    6433
PERISCOPE     28 PROGRAM    6823
REMOVER        2 PROGRAM     214
SAMPL-EA/O     6 DIS/FIX      80
SAMPL-EA/S    41 DIS/VAR      80
SAMPL-MM/O     6 DIS/FIX      80
SAMPL-MM/S    48 DIS/VAR      80
SCHEMA_P      25 PROGRAM    6144
SEDUCTION     14 PROGRAM    3232
SPELLING2      4 PROGRAM     704
TML-AUTO       4 PROGRAM     555
TML-RITA       5 PROGRAM     825
TML-SPRITE     4 PROGRAM     561
TRACK/DOC      3 DIS/VAR      80
TRACK/O        3 DIS/FIX      80
TRACK/S        3 DIS/VAR      80
TRACK/USE      2 PROGRAM     247
UCSD/ARC      16 INT/FIX     128
UCSD/O         2 DIS/FIX      80
UCSD/S         4 DIS/VAR      80
```

■

Manusstopp 15 mars PB 91-2
15 maj PB 91-3

MINI MEMORY EXPMEM

*av Stephen Shaw, TI*MES, England*

Mini Memory tillåter att du använder MINIMEM och EXPMEM2 som filer för data - och om du först öppnar till MINIMEM, så kan du sedan använda EXPMEM1 som datafil. Du kan emellertid välja EXPMEM1 utan att först använda MINIMEM, genom att trixa med datorn:

```
100 CALL PEEK(28672,S1,S2)
110 CALL LOAD(28672,90,165)
120 OPEN #1:"EXPMEM1",OUTPUT,
    INTERNAL,FIXED 60
130 CALL LOAD(28672,S1,S2)
```

Ett falskt värde placeras i 28672, filen öppnas och sedan återställes 28672.

(Med Mini Memory och 32 kbytes expansionsminne har du en RAM-disk med 3 filer MINIMEM, EXPMEM1 och EXPMEM2. Dessa rymmer 4, 24 respektive 8 kbytes ö.a.) ■

```
70 REM LEDTEXT MED INPUT
80 REM AV LARS HEDLUND
90 REM NITTINIAN 83-4.25
100 CALL CHAR(91,"0028003844
7C4444")
110 CALL CHAR(92,"0028003844
444438")
120 CALL CHAR(93,"0010003844
7C4444")
130 INPUT "MINSTA VÄRDE PÅ A
: ":AMIN
140 INPUT "STÖRSTA VÄRDE PÅ
A: ":AMAX
150 INPUT "MINSTA VÄRDE PÅ B
: ":BMIN
160 INPUT "STÖRSTA VÄRDE PÅ
B: ":BMAX
170 ADIFF=AMAX-AMIN+1
180 BDIFF=BMAX-BMIN+1
190 RANDOMIZE
200 A=INT(ADIFF*RND)+AMIN
210 B=INT(BDIFF*RND)+BMIN
220 PRINT
230 PRINT
240 INPUT "HUR MYCKET BLIR "
&STR$(A)&"X"&STR$(B)&"? ":C
250 IF C=A*B THEN 280
260 PRINT "DET VAR FEL!"
270 GOTO 240
280 PRINT "DET VAR RÄTT!"
290 GOTO 200
```

■

DISK MANAGER FÖR TI-99/4A

av Jan Alexandersson

Det finns nu många bra disk manager till 99:an. Jag har försökt jämföra de sex mest använda programmen:

- TI Disk Manager 2 modul
- Ottawa DM 1000 version 3.5
- Myarc DM V version 1.30 för HFDC
- John Birdwell Disk Utilities 4.2
- Funnelweb Disk Review 40 kolumn
- Funnelweb Disk Review 80 kolumn

Det finns buggar i DM1000 som gör att du inte bör använda tidigare eller senare versioner än 3.5. Funnelweb har en förbättrad version av 3.5 som är det bästa som finns för den vanlige användaren.

Om du endast har en skivenhet är det viktigt att du kan arbeta med samma enhet för original och kopia. Alla klarar detta utom FW DR80 som kräver två olika skivenheter. FW DR40 kan kopiera enstaka filer (FCTN-C) mellan skivor i samma skivenhet, men "Tag" av många filer samtidigt kräver två skivenheter.

	TI DM 2	OTTAWA DM1000 3.5	MYARC DM V 1.30	DISK UTIL 4.2	FWEB DR40 4.31	FWEB DR80 4.31
Antalet DSK-nummer	1-3	1-8	1-9,WDS	1-9,A-Z	1-9	1-9
En skivenhet in/ut	JA	JA	JA	JA	(JA)-	-
Density	S,D	S,D	S,D,Q	S,D,Q	S,D,Q	S,D,Q
FIL kopiera fil	JA	JA	JA	JA	JA	JA
huvud före data	JA	-	JA	JA	JA	JA
läs/skriv	fil??	sektor	sektor	sektor?	fil	fil
sektorer per pass	45	40	58	40	?	59
kopia med nytt namn	JA	-	-	JA	JA	JA
kopia överskriv test	-	-	JA	-	-	-
flytta fil	-	JA	JA	JA	-	-
radera fil	JA	JA	JA	JA	JA	JA
rädda fil	-	JA	-	JA	JA	JA
se D/V80 + D/F80	-	JA*	JA	JA	JA	JA
se alla D/V + D/F	-	-	JA	JA	JA	JA
se INT + PROGRAM	-	-	-	-	JA	JA
se Myart G6 + G7	-	-	-	-	-	JA
ändra skrivskydd	JA	JA	JA	JA	JA	JA
ändra filnamn	JA	JA	JA	JA	JA	JA
DISK katalog	JA	JA	JA	JA	JA	JA
printer kontrollkod	-	JA	-	JA	-	-
ändra skivnamn	JA	JA	JA	JA	JA	JA
multifilkopiering	JA	JA	JA	JA	JA	JA
filer per pass	en	en	flera	en	en	en
destination antal	1	1	1	1	8	8
kopia använda sekt	-	JA	-	JA	-	-
kopia alla sektorer	-	JA	JA	JA	-	-
sektorer per pass	-	104	57	39	-	-
formatera	JA	JA	JA	JA	JA	JA
formatera många	-	JA	-	JA	-	-
sweep disk	-	JA	-	-	JA	JA
TEST lästest	JA	-	JA	JA	JA	JA
skrivtest	JA	-	JA	-	-	-
SEKTOREDITOR	-	-	-	JA	JA	JA

* = FW-versionen av DM1000 kan inte visa DIS/FIX 80

Density är single, double eller quad. Observera att DM1000 ej kan användas för DS/QD.

En fil består av ett filhuvud samt ett antal datasektorer. De 32 första filerna lagrar alltid filhuvudet på sektor 2-33, men vid flera filer kommer även högre sektornummer att användas. Alla disk manager (utom DM1000) kommer att först skriva filhuvudet och sedan datasektorerna. Om du använder DM1000 med filkopiering för att kopiera TI-FORTH kommer du att upptäcka att kopian inte kan användas eftersom en mycket stor fil behövde låga sektorer för data när alla höga sektorer var upptagna. Filhuvudet kommer då efter datasektorerna trots att filhuvudet hamnar på låga sektornummer.

Programmen läser och skriver filer genom att anropa filrutiner eller sektorrutiner i diskkontrollkortet. Filskrivning fungerar felaktigt med Myarc HFDC så använd endast disk manager med sektorläsning för hårddiskkontrollkortet.

Om du har raderat en fil kan du rädda den med funktionen "recover file" under förutsättning att dessa lediga sektorer ej blivit överskrivna av andra filer.

FW Disk Review har möjlighet att från en originalskiva samtidigt kopiera till 8 olika skivenheter.

Sweep disk gör att sektor 0 och 1 skrivs som om det var en nyformaterad skiva. Någon test av eventuellt felaktiga sektorer görs inte. Använd inte detta om du är osäker på om skivan kan ha felaktiga sektorer.

REFERENSER

PB 86-3: Disk Manager 99
PB 86-4: MG Advanced Diagnostics
PB 86-5: Disk Manager
PB 87-2: DM 1000 version 3.5
PB 87-3: DM 1000 version 3.5
PB 88-3: Rapid Copy
PB 88-4: Mike Dodd MCOPY
PB 88-4: Myarc DM V version 1.21
PB 89-1: Myarc DM V version 1.29
PB 89-1: Disk Utilities 4.12
PB 89-3: Styrprogram för DM99

NYHETER TILL 99:AN

Vid en stor mässa i Chicago nyligen presenterades flera intressanta nyheter. Anmärkningsvärt är att Myarc och JP Software inte deltog. Slutsatsen måste bli att Myarc håller på att minska sitt intresse för 99-utrustning som Myarc Geneve och Myarc HFDC. Om du köper Myarc-kort får du räkna med att eventuella buggar och ofullständiga programdelar inte kommer att åtgärdas av Myarc. Utrustningen säljs i befintligt skick men rena reparationer kan sannolikt fortfarande utföras.

Rave presenterade en ny expansionsbox Rave 99 PE/2 med 200 W för TI-99/4A eller Geneve. Pris USD 309 för Geneve och USD 369 för TI-99/4A.

Asgard presenterade ett MIDI-interface för musik. Pris USD 44.95.

OPA presenterade TI-Image Maker (T.I.M.) med en 9958 videoprocessor (vidareutvecklad 9938). 80-kolumnskortet monteras inne i konsolen och kostar USD 179. Kontakta:
OPA, 432 Jarvis St. Suite 502, Toronto, Ontario, Canada.

OPA visade även Gismo som är en cartridge expander som kan ha 8 moduler samtidigt på ett sådant sätt att ett program kan anropa rutiner i olika moduler under samma körning.

Ett nytt tyskt diskkontrollkort BWG-Disk Controller för SS/SD - DS/DD visades. kortet har även en klocka med batteri-backup. Pris USD 240. Kontakta:
System 99 User Group, Attn: Michael Becker, Sankt Ingberter Strasse 5, D-6800 Mannesheim 31, Tyskland.

Denna information har tagits från en lång artikel av John Koloen i Micropendium. Du kan prenumerera på denna tidning för USD 42.00 med flygpost till Europa. Sänd beloppet (postgirots utlandsblankett utan att ange gironummer, money order eller kreditkort Master Card/Visa) till Micropendium, P.O. Box 1343, Round Rock, TX 78680, USA. Micropendium kommer ut 12 gånger per år och är den utan jämförelse bästa tidningen för TI-99/4A. ■

YET ANOTHER PAINT PROGRAM YAPP - RITA MED 9938

av Jan Alexandersson

Det har kommit ett nytt ritprogram för videoprocessorn 9938 som finns i Myarc Geneve, DIJIT AVPC och Mechatronic 80 kolumnskort. Du kan själv skapa bilder i Myart-format och även ladda in andras bilder i Myart- eller GIF-format. Programmet som är skrivet av Alexander Hulpke kan beställas från Asgard Software, P.O. Box 10306, Rockville, MD 20849, USA för USD 29.95 + porto USD 5.00 till Europa. YAPP är i assembler PROGRAM-format och kan laddas av EA, TW eller Funnelweb (oberoende av modul).

De flesta funktioner fungerar utan ytterligare hårdvara men det finns några som kräver 192 kbytes VDP-RAM och CPU-RAM på >7000. Mechatronic och DIJIT har vanligen allt VDP-RAM från början medan Geneve måste byggas ut (se PB 90-4 sid 19). Geneve har alltid CPU-RAM på >7000 medan TI-99/4A kräver Super-Cart, GRAM eller Mini Memory för att kunna ladda GIF-filer.

GRAPHIC6 OCH GRAPHIC7

Programmet kan användas med fyra olika grafikmoder:

grafik	punkter	färger
G6	512x212	16 av 512
G6 interlace	512x424	16 av 512
G7	256x212	256
G7 interlace	256x424	256

Alla dessa arbetar med äkta bitmap så att varje punkt kan ha en färg helt oberoende av alla övriga punkter. YAPP har till skillnad mot Myarc Myart dubbelt så många punkter när interlace är inkopplat. Min Philips CM 8833 monitor får med interlace en bild som ej står helt stilla. Du kan inifrån YAPP välja 50 eller 60 Hz bildfrekvens. Jag använder alltid 60 Hz som tar bort det vanliga flimret som du brukar se när du tittar på TV.

MUS ELLER JOYSTICK

Drivrutinerna för pennan finns i en särskild YAPPDSR-fil som ska laddas som DSK1.YAPPDSR. Det är möjligt att ändra med en sektoreditor (t.ex. FW Disk Review) till DSK2.YAPPDSR eller WDS1.YAPP.YAPPDSR.

Du kan rita med Joystick, Myarc mus (3 tangenter 9995/9938), DIJIT AVPC mus (2 tangenter 9938), Mechatronic mus (2 tangenter joystick-port interrupt) eller Asgard mus (3 tangenter RS232-port interrupt). Det är mycket smidigare att rita med mus än med joystick. Programmet har möjlighet att utnyttja alla tre mustangenterna. De två första tangenterna används för att sätta ned pennan respektive för att visa färgpaletten och kommandosymbolerna. Den tredje tangenten (som saknas på DIJIT) används för att ångra det du ritat senast (ungefär som OOPS! i TI-Writer). Det finns dock en alternativ tangenttryckning CTRL-U som utför samma kommando.

RITKOMMANDON

Följande möjligheter finns när det gäller att rita:

- välja färg ur paletten. I G7 förstoras en del av paletten så att färgen enkelt kan väljas.

- rita linje på fri hand som kan väljas mellan fyra olika storlekar på penseln.

- sprayflaska

- rita rak linje mellan två punkter

- rita öppen ram (rektangel)

- rita fylld box (rektangel)

- rita cirkel/ellips

- fylla en sluten yta av godtycklig form med färg

- kopiera del av bilden till annan plats

- flytta del av bilden till annan plats

- klippa ut del av en bild som kan klistras in i en annan bild

- peka på en punkt så att dess färg används av pennan i fortsättningen vilket är särskilt bra i G7 som har 256 olika färger

- zooma in och rita med en liten del av bilden uppförstorad (om interlace är inkopplat behövs 192 kbytes VDP-RAM)

- skriva text som kan tas från font-filer som används i TI-Artist d.v.s. filnamn som slutar på _F

- omdefiniera 16 färger från en palett på 512 färger i G6-mode

- reset till normala färger i G6-mode

- logiska operationer IMP, AND, OR, XOR, NOT som verkar när du laddar bild eller ritar

- utskrift till Epson-kompatibel skrivare

FLEXSKIVA OCH HÅRDDISK

YAPP kan lista katalogen för en flexskiva eller godtycklig sub-directory för hårddisk. Du kan leta dig steg för steg till underliggande subdirectory tills du hittar dina filer med bilder eller ange hela path-namnet från början beroende på vad som är enklast för dig. Alla 127 filnamn i en subdirectory kan visas samtidigt på 80-kolumnsskärmen i Text2-mode. Allt detta är mycket bättre än motsvarande i TI-Artist Plus som har flera begränsningar när det gäller anrop av hårddisk.

GIF-FILER

YAPP kan alltid ladda och spara filer i Myart-format. Om du har RAM-minne på >7000 samtidigt som det finns 192 kbytes VDP-RAM så kan du även ladda filer i GIF-format (Graphics Interchange Format). GIF-filer krymps horisontellt så att hela bilden syns vilket är mycket bättre än andra kända GIF-laddare. Exempel på hur YAPP laddar GIF:

punkter	färger	bild	mode
320x200	32	27x18cm	G7
320x200	16	17x18cm	G6
(borde vara		27x18cm	G7)
640x200	16	27x18cm	G6
640x400	16	27x18cm	G6 interlace
640x350	16	27x16cm	G6 interlace

GIF bibehåller all information som finns i bilder som hämtats från andra datorer som Amiga eller IBM PC. Andra program för GIF är GIF99 av Achim Liese för TI-99/4A och GIF2 av Paul Charlton för Geneve.

Observera att Texaments GIF Mania endast omvandlar till TI-Artist som arbetar i Graphic2 som en vanlig TI-99/4A även om 9938 finns ansluten.

REFERENSER

PB 89-1 80-kolumnskort
PB 89-2 XHI 3.4
PB 89-4 XHI 3.6
PB 90-2 Program för 9938
PB 90-4 Geneve 192k VDP RAM
PB 90-6 Fractals 2.1 för 9938

Micropendium:

feb 87: Mechatronic Mouse, review
nov 87: Mouse DSR
dec 87: Myarc MY-Art, review
sep 88: Myarc, DIJIT rely on 9938
feb 90: XHI, review
maj 90: Asgard/Mechatronic 80 column
nov 90: Asgard Mouse, review
dec 90: YAPP, review ■

FUNNELWEB 4.31 NOV/16/90

Funnelweb 4.31 finns nu i en ny version från NOV/16/90. Disk Review 80 kan nu även visa YAPP-bilder med 424 linjer. Eftersom FW använder en del av VDP-RAM för internt bruk, måste dessa "interlace"-bilder visas i två delar. Först visas övre halvan av bilden och sedan visas resten. Dokumentationen till DR80 har delats på tre filer FWDOC/DR80, FWDOC/DR81 och FWDOC/DR82. ■

7 SÄTT ATT LAGRA PROGRAM

av R.A.Green, Ottawa UG, Canada

Det finns sju olika sätt att lagra program med en TI-99/4A. I denna artikel kommer vi att titta på alla dessa sju former och hur de används.

Alla är bekanta med det sätt som används av TI Basic för att lagra program på kassett eller skiva. Det betecknas som PROGRAM i skivkatalogen. Det har skapats eller sparats med Basics SAVE-kommando och laddats av Basics OLD-kommando. Detta är det enda sätt som TI Basic använder för att lagra dina program.

Extended Basic kan använda samma sätt som TI Basic för att lagra program. I själva verket kan Extended Basic använda TI Basics program. Det finns emellertid två andra former som XB använder. Båda dessa former kan endast användas för att lagra program på skiva.

Om du har 32 kbytes expansionsminne kan du skriva ett XB-program som är för stort för att lagras i det vanliga formatet. XB kommer att spara dessa stora program i en INTERNAL VARIABLE 254 fil. De vanliga SAVE- och OLD-kommandona används för att spara och ladda dessa program.

Det tredje sättet som används av XB är MERGE-formatet som lagras i en DISPLAY VARIABLE 163 fil. Denna form skapas när MERGE väljs som tillägg till SAVE (SAVE DSK1.FILNAMN,MERGE), och som laddas med XB:s MERGE-kommando (MERGE DSK1.FILNAMN). Det fina med MERGE är att när det laddar så bibehålles tidigare programrader i minnet. MERGE-kommandot sammansmälter det nya programmet med det program som tidigare finns i minnet med hänsyn till radnummer.

Nu kommer vi till godbitarna nämligen assemblerprogram. Det finns tre former av assemblerprogram: "tagged object", "compressed tagged object" och "memory image".

Tagged object lagras i en DISPLAY FIXED 80 fil endast på skiva. Alla programdata finns i hexadecimal form så att det kan editeras med E/A-editorn. Tagged object kan laddas med CALL LOAD i XB, val 3 i E/A-menyn, val 1 i MM-menyn eller med CALL LOAD i TI Basic när antingen E/A- eller MM-modulen används. Programmet kan vara "absolute" eller "relocatable". Ett "absolute" program måste alltid laddas till samma minnesutrymme. Ett "relocatable" kan laddas till valfri plats i minnet. Ett "tagged object" program kan ha referenser till andra program eller subrutiner. Laddaren kommer att lösa upp dessa externa referenser, utom när det gäller XB-laddaren.

Compressed tagged object är nästan samma sak som tagged object bortsett från att programdata lagras som bytes istället för som hexadecimala siffror. Compressed tagged object laddar snabbare än vanlig tagged object, som väntat. XB-laddaren kan inte ladda compressed object.

Tagged object av båda sorter, kan alstras av assembleraren när den assemblerar ett källkodsprogram.

PROGRAM-formatet (memory image) är den form av assemblerprogram som är mest kompakt och som laddas snabbast. Det kan lagras på kassett eller skiva. Det visas som PROGRAM i skivkatalogen (precis som ett Basic-program). PROGRAM-format kan laddas med val 5 i E/A-menyn eller val 3 i TI-Writer-menyn. Det bör påpekas att det finns en liten men viktig skillnad mellan hur E/A anropar en PROGRAM-fil och hur TI-Writer gör. TI-Writer suddar skärmen alldeles före anropet vilket E/A inte gör. Detta betyder att programmet måste aktivera skärmen eftersom annars inget händer. PROGRAM-filer skapas av ett nyttoprogram som finns på skivan till E/A (Du bör inte ha REF till unika adresser i E/A som t.ex. VMBW, utan dessa måste ersättas med en subrutin med egen källkod, så att övriga moduler och programladdare kan starta detta program ö.a.).

En PROGRAM-fil, som innehåller assembler memory image eller ett Basic-program, kan läsas eller

skrivas till godtycklig in- eller utenhet med en enstaka I/O-operation. Detta är ett av skälen till att den laddas så snabbt.

Det finns en begränsning i storleken av en assembler PROGRAM-fil på >2400 bytes (9216 decimalt). Emellertid kan E/A och TI-Writer modulerna ladda flera sammanhörande PROGRAM-filer så att programmet kan vara av godtycklig storlek. Det använder principen att namnet på den andra och följande filer erhålles genom att stega fram sista bokstaven eller siffran hos föregående filnamn. Exempelvis består TI-Writers editor av två PROGRAM-filer: EDITA1 & EDITA2.

Även Adventure, Tunnels of Doom, Personal Record Keeping, Statistics och Personal Report Generator modulerna använder PROGRAM-filer för sina databaser. Det faktum att PROGRAM-filer kan sparas och laddas med en enstaka I/O-operation gör dem attraktiva för sådana användare.

En hel del spel skrivna i assembler finns i PROGRAM-format så låt oss titta närmare på dessa. PROGRAM-filer för assembler börjar med tre ord som följs av de data som ska placeras i minnet. De tre inledande orden (ord = 2 bytes) är:

(1) Detta ord är en flagga som om det inte är noll (t.ex. >FFFF) så är detta inte den sista filen av de sammanhängande PROGRAM-filerna. Exempelvis är flaggan för EDITA1 >FFFF vilket betyder att det finns ytterligare en fil kallad EDITA2. Flaggan i EDITA2 är >0000 vilket betyder att det är sista filen och att någon EDITA3 inte finns.

(2) Detta ord är längden av PROGRAM-filen i bytes inklusive huvudets 6 bytes (3 ord).

(3) Detta ord är CPU-adressen där PROGRAM-filen skall laddas.

PROGRAM-filer startar alltid exekveringen med den första byten i det första programsegmentet som har laddats.

Till slut, den sjunde formen av program. Denna skapas och laddas av EASY BUG med Mini Memory modulen. Den kan endast skrivas till kassett och är i PROGRAM-format, men något annorlunda än E/A PROGRAM-filer. EASY BUG:s PROGRAM-filer kan endast bestå av ett enda segment. Inledningen med EASY BUG-formatet är två ord enligt följande:

(1) Detta ord är den CPU-minnesadress till vilken programmet skall laddas.

(2) Detta ord är längden av data exklusive de fyra inledande byten.

Följande tabell visar möjligheterna:

INNEHÅLL	FILTYP	MODUL	DSK	CS
BASIC-program	PROGRAM	konsol	Ja	Ja
BASIC-program	PROGRAM	XB	Ja	Ja
BASIC-program	INT/VAR254	XB	Ja	Nej
MERGE-program	DIS/VAR163	XB	Ja	Nej
Tagged Object	DIS/FIX 80	XB	Ja	Nej
Tagged Object	DIS/FIX 80	EA	Ja	Nej
Tagged Object	DIS/FIX 80	MM	Ja	Nej
CompressedObj	DIS/FIX 80	EA	Ja	Nej
CompressedObj	DIS/FIX 80	MM	Ja	Nej
EA PROGRAM	PROGRAM	EA	Ja	Ja
EA PROGRAM	PROGRAM	TW	Ja	Ja
MM PROGRAM	PROGRAM	MM	Nej	Ja ■

MCGOVERNS XB-SKOLA

av Tony McGovern, Australien

XB-skolans 7 delar har publicerats i följande nummer av Programbiten:

1A	PB 87-4 sid 23
1B	PB 88-1 sid 4
2	PB 90-1 sid 12
3	PB 90-2 sid 12
4	PB 90-3 sid 10
5	PB 90-4 sid 4
6	PB 90-5 sid 4
7	PB 90-6 sid 4

Efter det att del 1 hade översatts men före del 2 har Tony omarbetat denna artikelserie. Här kommer en reviderad inledning som ersätter den som har publicerats i PB 87-4.

I. INTRODUKTION

Denna artikelserie skrevs ursprungligen för många år sedan för att försöka att förbättra den avgrundslika programmeringsstandard som visat sig i tidskrifter och medlems-

tidningar, och även i kommersiella program vid denna tidpunkt. De första avsnitten publicerades i Sydney TISHUG NewsDigest medan senare avsnitt kom i HV99 Newsletter efter det att denna förening bildats som en fristående företeelse. Standarden på publicerade program skrivna i XB har i allmänhet förbättrats sedan dess, men inte så mycket som det borde, och vartefter avancerade användare går vidare, kommer nytillkomna nybörjare att ta deras platser med grund-konsolen och XB och behöver lära sig använda dem. XB är inte längre det viktigaste språket för en expanderad TI-99/4a, men det är fortfarande ett mycket uttrycksfullt och kraftfullt språk, med egenskaper som först nyligen har kommit i nya och mera anspråksfulla datorer. Det tycks finnas ett kontinuerligt behov av undervisande artiklar om XB, så jag hoppas att denna serie kommer att vara av värde för nya och gamla programmerare.

Artikelserien är varken avsedd som en elementär grundkurs för fullständiga nybörjare eller som en referenslista. Den är avsedd för den intresserade användaren som är villig att lägga ned en viss möda på att förstå hur Extended Basic arbetar för att använda det på bästa sätt, och som önskar att utveckla en känsla för hur maskinen verkligen uträttar sitt arbete. Denna utgåva av XB-skolan representerar en försiktig bearbetning av de ursprungliga filerna, huvudsakligen för att ta bort material som blivit omodernt.

Avsikten med denna serie om TI Extended Basic har hela tiden varit koncentrerat på de egenskaper som inte har fått nödvändig uppmärksamhet i användarföreningarnas tidningar och kommersiella tidskrifter. I själva verket hade de flesta program som publicerats i dessa källor fram till XB-skolans första utgåva i mycket liten utsträckning användt den mest kraftfulla egenskapen hos XB, användardefinierade underprogram, eller av andra funktioner hos XB. Ett av skälen till detta var att vissa författare publicerade samma material för flera olika datorer i samma tidning, påtvingande användaren den lägsta gemensamma nämnaren i Basic. Ännu värre var de många program som visade hur man skriver invecklad och oläslig programkod. Ett mycket försummat hjälpmedel är TI:s Extended Basic Tutorial på kassett eller skiva. Programmen i denna samling är ej skyddade så de är öppna för inspektion och det är värt att titta på dess programlistning för att se ett exempel på hur underprogram kan ge en enkel och lättförstålig struktur för ett program.

Nå vad kommer vi att behandla? Ämnen som kommer att tas med

(1) Användardefinierade underprogram
(2) Pre-scan switch-kommandot
(3) Buggar i Extended Basic
(4) Krympning av programlängd

Inledningsvis kommer diskussionen att inskränkas till saker som kan göras med enbart grunddatorn. Ursprungligen var avsikten att täcka länkning av assemblerrutiner (med CALL LINK) men serien tog slut innan dess. I själva verket kommer närvaron av expansionsminne för de flesta spelprogrammerare med XB inte att snabba upp XB så mycket att inte snabbheten tycks begränsas av de inbyggda underprogrammen (CALL COINC, osv) vilka körs från GROM genom GPL-interpretatorn. Det verkliga lyftet med expansionsminne för spelprogram, utöver möjligheten till längre program, är att GPL kan läggas åt sidan till förmån för maskinkodsrutiner i de hastighetskritiska delarna av spelet, vilka vanligen endast är en mycket liten del av spelets programkod. Trots detta kan ofta omsorgsfull XB-programmering ge den nödvändiga snabbheten. Som exempel, är snabbheten hos pucken i TEX-BOUNCE, vår första större övning i XB-kodning, en faktor 10 snabbare i den slutliga utgivna versionen än i den första kodningen av spelet. Lustigt nog visade det sig att kodningen av detta spel är så väl avvägd till det sätt som TI XB hanterar saker att puckens hastighet måste minskas för Myarc XBII så att den kan hanteras utan att kontakten tappas med den, trots att den är avsedd att vara mycket snabbare.

(fortsätt här att läsa avsnitt II i PB 87-4) ■

9938 UTVECKLAD TILL 9958

av Jan Alexandersson

Yamaha har vidareutvecklat sin videoprocessor V9938 till V9958 för användning i nästa generation av spelmaskinen Nintendo. Den tidigare 9938 finns i Myarc Geneve, DIJIT AVPC och Mechatronic 80-kolumnskort. OPA:s nya T.I.M. använder den nya 9958.

Ändringarna i 9958 är ganska måttliga jämfört med 9938. Båda videoprocessorerna är helt mjukvarukompatibla med TMS 9918A/9929A som sitter i TI-99/4A. Den nya 9958 är nästan kompatibel med 9938 men man har tagits bort kompositvideoutgång och gränssnitt för mus. 9958 är dock inte pinkompatibel med 9938.

9958 har tre nya VDP-register # 25, 26 och 27.

9958 har möjlighet att mjukt skrolla bilden horisontellt med en punkt åt gången. En eller två skärmars bredd kan visas under skrollningen.

9958 har WAIT-funktion som snabbar upp åtkomsten av VDP-RAM eftersom alla övriga portar görs inaktiva under anropet (där försvann musporten).

9958 kan använda kommandofunktionerna (HMMC, LINE, SRCH, PSET, POINT m.fl.) i samtliga grafikmoder d.v.s. även de som ej är äkta bitmap. Den tidigare 9938 kan endast använda dessa i G4-G7.

YJK-grafik möjliggör användning av 19 268 olika färger samtidigt. Man använder 17 bitar för att bestämma så många färger. Den enda inskränkning som finns är att fyra närliggande punkter har 12 bit gemensamt medan 5 bit är unika för varje punkt:

```
byte  C7 C6 C5 C4 C3 C2 C1 C0
1:a   ******Y1****** ***KL***
2:a   ******Y2****** ***KH***
3:e   ******Y3****** ***JL***
4:e   ******Y4****** ***JH***
```

De fyra punkterna delar på 4 bytes = 32 bit enligt följande:

```
1:a   Y1 + KL + KH + JL + JH
2:a   Y2 + KL + KH + JL + JH
3:e   Y3 + KL + KH + JL + JH
4:e   Y4 + KL + KH + JL + JH

Y = B/2 + R/4 + G/8
J = R - Y
K = G - Y
```

YAE-grafik möjliggör att varje grupp om fyra punkter antingen kan välja bland 9634 färger med samma inskränkning som YJK. Varje punkt kan aktivera YJK eller färgpaletten med 16 av 512 färger:

```
byte  C7 C6 C5 C4 C3 C2 C1 C0
1:a   *****Y1****  A ***KL***
2:a   *****Y2****  A ***KH***
3:e   *****Y3****  A ***JL***
4:e   *****Y4****  A ***JH***
```

Om A=0 kommer de fyra punkterna att dela på knappt 4 bytes = 28 bit enligt följande:

```
1:a   Y1 + KL + KH + JL + JH
2:a   Y2 + KL + KH + JL + JH
3:e   Y3 + KL + KH + JL + JH
4:e   Y4 + KL + KH + JL + JH
```

Om A=1 kommer endast Y1, Y2, Y3 och Y4 att användas men färgnumret översättes till verklig färg med hjälp av färgpalettregistren. Dessa kan visa 16 av 512 färger.

Genom att läsa statusregister S#1 bit 1-5 kan programmet självt avgöra om det sitter en 9938 eller 9958. Dessa visar 0 för 9938 respektive 2 för 9958.

REFERENSER

V9938 MSX-Video Technical Data Book, aug 1985, 152 sidor.

V9958 MSX-Video Technical Data Book, dec 1988, 30 sidor, endast ändringar jämfört med V9938.

Micropendium:

aug 88: The 9918A and earlier chips (TMS 9918A/28A/29A, TMS 9118/28/29, TMS 9927/37)

sep 88: Myarc, DIJIT rely on Yamaha 9938.

■

BEGINNER ASSEMBLER - 1
THE FIRST PROGRAM

by Mack McCormick, USA

Here are the objectives of this first tutorial:

1. To introduce you to the Hexadecimal and Binary Numbering systems.

2. To introduce you to the assembler instruction format.

3. To introduce you to addressing modes.

4. First program. Adding two numbers and displaying them on the screen.

5. How to assemble.

Just a few words on Assembly language before we begin. It's not as difficult as you may believe. You will be communicating with the microprocessor at the first level above machine language, assembler. As you know, the machine actually communicates in binary 0's and 1's, on or off. Assembler allows us to talk to the machine in a language we can understand (Although I'm sure the uninformed would disagree). With these tutorials I will assume no prior knowledge of assembler or other number systems. Please bear with me, I won't insult your intelligence and things will become more complex soon. Stick with the tutorials. Read every book about assembler you can get your hands on.

Numbering Systems

Hexadecimal (HEX) and binary are merely different base numbering systems for counting. It's important we understand both of these systems in addition to base 10 or the decimal system because assembler uses all three. I will tell you up front that I use a calculator designed for these numbering systems usually but we need to understand the principles also. If you want to get a calculator, and I recommend that you do, there are several inexpensive models on the market.

Binary Number System

As already mentioned, binary is the native language for your computer. Everything eventually gets converted to binary. Let's look at a decimal number first. As you know decimal means powers of 10. Each number represents a power of ten. For example 4175:

```
10^3=1000   4 X 1000=4000
10^2= 100   1 X  100= 100
10^1=  10   7 X   10=  70
10^0=   1   5 X    1=   5
                      ----
                      4175
```

Binary numbers can be 1 or 0 only, hence base 2. The individual number is called a bit. A group of eight of these is called a byte. To convert the binary number 00001011 to decimal follow the same procedures you used with the decimal number:

Ignore any leading zeros.

```
2^3=8       1 X 8=8
2^2=4       0 X 4=0
2^1=2       1 X 2=2
2^0=1       1 X 1=1
                  --
                  11
```

To make it easier to communicate with the computer we most often use HEX. From now on I will use a > to indicate a number is in hex. Hex is base 16. That is a number may be 0 thru 15. To represent numbers greater than 9 we use letters of the alphabet. 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Just remember to use >A for 10 and count to 15 ending with >F. Let's convert >394F to decimal:

```
16^3=4096   3 X 4096=12,288
16^2= 256   9 X  256= 2,304
16^1=  16   4 X   16=    64
16^0=   1   F X    1=    15

                     14,671
```

The largest number you may represent in one byte is >FF or decimal 255. The largest value in a word (two bytes) is >FFFF or 65,535. Enough on numbering systems for now, we'll cover minus numbers (twos compliment) and additional points as we encounter them in programs.

Assembler Instruction Syntax.

Like every computer language there are certain rules we must follow for inputting instructions. Unlike BASIC, assembler will not give you a warning or error until you assemble the program. Here's the general syntax:

LABEL OPCODE OPERAND COMMENT

labels must begin in the 1st column and may be up to 6 characters long. One or more spaces follow. Next is the OPCODE. This is the actual instruction to be performed followed by one or more spaces. Next are one or more operands or data for the instruction to operate on followed by one or more spaces. Finally is an optional comment which may extend to column 80. Each time we use a new instruction I will fully explain it.

Addressing Modes.

There are five general addressing modes and one special addressing mode used for assembler instructions. We will examine each one in detail as we encounter them in a program. There is one type of addressing we need to look at now. We are going to be operating on individual bits, bytes, and words of memory. Think of the computers memory as a series of consecutive small pieces of memory laid out end to end. We can address any single byte of memory by hanging a label on it but frequently we must address a byte of memory some distance from that label. Think of it like an array. To get to the 5th byte from the label we could say LABEL+4. We used 4 instead of 5 because we must start counting from 0. Think of it like OPTION BASE 0 in BASIC. Lot's more on this later.

First Program

I strongly recommend you enter the program manually by typing it in instead of just cleaning it up using TI-Writer or Editor/ Assembler. The only way to gain experience programming is to practice.

Program explanation. These comments supplement the comments contained in the program itself. Any statement with an * in column 1 is a comment and you may enter anything else on that line. One fairly unique thing about the 9900 microprocessor in the TI-99 is the ability to designate your own workspace registers anyplace in memory or more than one set at a time. Think of registers as 32 consecutive bytes of memory that are used as your scratch paper for calculations. Thirty-two bytes is of course 16 words of memory. Because this is a 16 bit (1 word) machine (something many of your friends can't brag about) that gives us 16 registers to use for our computations. We place an R in front of the number to designate that we are refering to a register. For example, R0 is register zero and R15 is register fifteen (really the 16th word of memory because we started counting with zero.) Here's the detailed explanation of the program:

```
       DEF START
```

DEFines the entry point of the program so the computer may find it. Places the name START in the Reference/Definition table. More on this next time.

```
       REF VSBW,VMBW
```

REFerence refers to module routine the program will use. In the advanced tutorials we'll create our own utilities.

```
WSREG  BSS >20
```

WSREG is the label we decided on for our workspace registers. Could have been any label 1 to 6 characters long. Block Starting Symbol (BSS) sets aside a block of memory, unititialized for use as our workspace. >20 means set aside >20 or 32 bytes (16 words) for R0 to R15.

```
X      DATA 10
```

Initializes one word of memory to 10 (0010). Hangs the label X on that word.

```
START  LWPI WSREG
```

Load Workspace Register Immediate (LWPI). Tells the computer to use the 32 bytes of memory for our workspace which begin at WSREG. START is the entry point (beginning) of our program.

Logic for clear routine. Writes the space character >20 or 32 to the screen 768 times to the screen to clear it. R0 is the counter. R1 contains the space character. We increment R0 to point to the next screen location and check it against 767 to insure we haven't gone too far.

```
       CLR  R0
```

CLeaRs the contents of R0 to zero.

```
       LI   R1,>2000
```

Load Immediate R1 with >2000.

```
LOOP   BLWP @VSBW
```

Branch and Link Workspace Pointer to the Video Single Byte Write Routine. Branches to the module routine for writing single bytes of information to the screen. R0 always contains the address on the screen to write to. Briefly, there are 768 screen positions 24 rows X 32 Columns=768. This routine writes to VDP RAM in the screen image table (SIT) which is 768 bytes long. Any ASCII value you write to the SIT is displayed on the screen. For example to display the number 3 at row one column one, R0 would have 0 in it (because we begin counting at 0) and R1 would contain >3300 in it. Note the number to be written is in the left byte of the word. The VSBW routine always writes on the Most Significant Byte and disregards the LSB. Here's the easy way to remember it. R0 always contains the address in VDP RAM. R1 always contains the address or data in CPU RAM.

```
       INC  R0
```

INCrement R0 by one. Add one to the contents of R0.

```
       CI   R0,767
```

Compare Immediate whats in R0 to 767.

```
       JLE  LOOP
```

Jump Less than or EQUAL to the label LOOP. IF R0<=767 THEN GOTO LOOP.

Logic for the addition routine. We add the two number together. Because only ASCII numbers may be displayed on the screen we must add >30 to each byte before we display it. In this case our number is 37. We must place a 3 and 7 on the screen. To do this we divide 37 by 10 resulting in a quotient of 3 and remainder of 7. We add >30 to the 7 to make ASCII >37. We move this value to the right most byte of our ANS word. We then divide 3 (old quotient) by 10 resulting in 0 quotient and 3 remainder. We again mask up by >30 and shift it left 8 bits (1 byte) in the register. We then move this byte to the left (MSB) of ANS. ANS looks like >3337 when we're finished.

```
       A @Y,@X
```

Adds two words of memory. Places the sum in the second operand. May also use registers (eg. A R0,R1). Adds whats at the word of memory with label Y to whats at the word of memory label X.

```
       DIV  @TEN,R5
```

DIVides uses two registers. In this case R5 and R6. Divides whats in R6 by whats at TEN or 10. The quotient is placed in R5 and the remainder at R6. That is why we clear R5 before we divide.

```
MOV   R6,@ANS
```

MOVe the contents of R6 to whats at the label ANS.

```
SLA   R6,8
```

Shift Left Arithmetic. Shift the contents of R6 left 8 bits (1 byte) to the MSB. Fills the shifted out positions with 0.

```
MOVB  R6,@ANS
```

MOVe Byte moves the Most Significant Byte (MSB) or leftmost to the word at ANS without disturbing the LSB of ANS.

Logic for display on the screen routine. R0 contains the position on the screen to display the answer. Found by SCREEN ADDRESS=((ROW-1)* 32)+(COLUMN-1). In this case 366. R1 contains the address of the beginning of the data to write to the screen. In this case R1 contains the address of ANS. R2 contains the number of bytes to write beginning at the VDP address in R0 and the CPU address in R1.

```
JMP  $
```

Instructs the computer to JuMP to the current location of the program counter. Same as 100 GOTO 100. This locks up the computer so you may see the result. If you want to see how quickly the computer executes place an * in column 1 in front of this instruction and reassemble.

Logic for the Return to the Calling Routine. Clears the GPL status byte at >837C. Much more on this important byte later. Loads the workspace pointer back to the GPL workspace and branches to the routine at >0070. END is a directive to inform the assembler there are no more instructions.

How to Assemble.

If you have the Molesworth book refer to page 42 or page 30-36 in the Editor/Assembler manual. Here's a brief step by step.

1. Select EDITOR/ASSEMBLER from the main menu. Place your editor assembler disk A in drive 1.

2. Select 1-EDIT from the E/A menu.

3. Select 2-EDIT from the EDIT menu.

4. Enter your program just as shown. You may omit any comments if you desire.

5. Press FCTN ESCAPE twice to return to the EDIT title screen.

6. Select 3-SAVE. Answer Y to the VAR/80 prompt. Enter your source file name such as DSK2.SOURCE. If you only have one drive place another disk in drive one first and use DSK1 instead of DSK2.

7. Press FCTN ESCAPE to return to E/A title screen.

8. Select 2-ASSEMBLE. Answer Y to the load assembler question. Insure the E/A disk A is in drive 1.

9. At the SOURCE FILE NAME enter the same file name you used in 6 a bove. ie. DSK2.SOURCE, press enter.

10. At the OBJECT FILE NAME enter DSK2.OBJECT and press enter.

11. Press enter at the LIST FILE NAME. More on this feature next time.

12. At the OPTIONS prompt enter R. Press enter. R means you used R in front of your register numbers in the source code. You may also enter CLST. C is compressed object code (will not load from X/B loader). L is a source listing if you entered a LIST FILE NAME at the prompt. S prints the symbols and registers used in the program on your source list. T prints the full text string in your source listing. More on these features later. Assembler executing will appear followed by 0000 errors (you hope). Press enter.

13. Press 3-LOAD AND RUN.

14. At the FILE NAME prompt enter your OBJECT file name ie. DSK2. OBJECT. Press enter. Press enter again to get to the PROGRAM NAME

prompt.

15. This is the name we DEF in our program in this case START.

16. Your program will execute.

SUMMARY.

I realize this has been long but there has been much to cover to get started. Don't get discouraged. We'll go at this together. I strongly recommend you study these references in your Editor-Assembler manual and experiment on your own. Until next time, "ASSEMBLER EXECUTING".

Page
20-36 Using the Editor-Assembler Cartridge.
39 Sec 3.1 Registers
46-48 Source Statement Format
53 Predefined Symbols. ($)
57-62 Sec 4.1.1, 4.1.4, 4.2, 4.4 Addressing.
80 Add Instruction
85 Add Immediate
88 Divide
90 Increment
107 Branch
115 Jump Less Than or Equal.
143 Compare Immediate
163 Load Immediate
165 Load Workspace Pointer Immediate
166 Move Word
168 Move Byte
200 Shift Left Arithmetic
212 Block Starting Symbol
225 Data
227 DEF
228 REF
248 VSBW,VMBW
329-330 Graphics Mode Tables
394-396 Numbering Systems
428-429 ASCII character set
442 Other Returns

```
****************************************
*                                      *
* THIS IS THE FIRST PROGRAM FOR THE    *
* BEGINNER ASSEMBLER TUTORIAL.         *
* IT CLEARS THE SCREEN, ADDS TWO NUMBERS *
* TOGETHER AND DISPLAYS THE SUM IN THE *
* CENTER OF THE SCREEN.                *
* HERE'S WHAT IT WOULD LOOK LIKE IN    *
* EXTENDED BASIC:                      *
* 10 CALL CLEAR                        *
* 20 X=10                              *
* 30 Y=27                              *
* 40 X=X+Y                             *
* 50 DISPLAY AT(12,15):X               *
* 60 END                               *
*    BY MACK MCCORMICK                 *
****************************************

* THIS PART OF THE PROGRAM IS THE INITIALIZATION *

       DEF  START  THE PROGRAM NAME IS START
       REF  VSBW,VMBW MODULE ROUTINES WE ARE GOING TO USE
WSREG  BSS  >20    SETS ASIDE A BLOCK OF 32 BYTES FOR USE AS WORKSPACE REGISTERS
X      DATA 10     COULD HAVE USED >A INSTEAD OF 10 (LIKE X=10)
Y      DATA 27     (Y=27) COULD HAVE ALSO SAID Y EQU 0027 (not with @Y in code)
TEN    DATA 10
ANS    DATA 0        WORD TO PUT ANSWER IN. INIT TO 0.

* PROGRAM BEGINS HERE
*------------------------------*
START  LWPI WSREG    LOAD WORKSPACE POINTER IMMEDIATE. POINT TO OUR WORKSPACE.

* CLEAR THE SCREEN   (CLR R0 is short for LI R0,>0000)
       CLR  R0       CLEARS R0 TO ZERO (BEGINNING OF SCREEN IMAGE TABLE)
       LI   R1,>2000 LOAD IMMMEDIATE R1 WITH >2000. VSBW ROUTINE WRITES THE LEFT
```

```
LOOP    BLWP @VSBW     BYTE IN R1 ALWAYS. IN THIS CASE >20 OR 32 OR SPACE CHR
        INC  R0        ADD 1 TO R0
        CI   R0,767    COMPARE IMMEDIATE R0 TO 767
        JLE  LOOP      IF IT'S LESS THAN OR EQUAL JMP (GOTO) LOOP

* ADD THE NUMBERS TOGETHER AND CONVERT TO ASCII
*------------------------------*
        A    @Y,@X     ADDS X TO Y AND PLACES RESULT IN X
        MOV  @X,R6     MOVE WHATS AT X TO R6
        CLR  R5        CLEAR R5
        DIV  @TEN,R5   DIVIDES 10 INTO 37. QUOTIENT IN R5. REMAINDER R6.
        AI   R6,>30    ADD IMMEDIATE ASCII OFFSET TO R6
        MOV  R6,@ANS   MOVE CONTENTS OF R6 TO THE WORD ANS
        MOV  R5,R6     MOVE CONTENTS OF R5 TO R6
        CLR  R5        CLEAR R5
        DIV  @TEN,R5   DIVIDE 10 INTO R5, R6
        AI   R6,>30    ADD IMMEDIATE ASCII OFFSET >30 TO R6
        SLA  R6,8      SHIFT LEFT ARITHMETIC R6 8 BITS.
        MOVB R6,@ANS   MOVE MSBYTE R6 TO R1

* DISPLAY ON THE SCREEN AT ROW 12 COLUMN 15
*------------------------------*
        LI   R0,366    POSITION ON THE SCREEN IS 366
        LI   R1,ANS    LOAD R1 WITH THE ADDRESS OF ANS
        LI   R2,2      TWO BYTES TO WRITE
        BLWP @VMBW

        JMP  $         PREVENTS THE PROGRAM FORM ENDING SO YOU MAY SEE THE RESULT

* RETURN TO THE CALLING PROGRAM
*------------------------------*
        CLR  @>837C    CLEAR THE STATUS BYTE
        LWPI >83E0     LOAD GPL WORKSPACE REGISTERS
        B    @>0070    BRANCH TO THE CALLING PROGRAM

        END                                                          ■
```

TRANSPUTER

by Tony McGovern, Australia

Not so directly TI related but still one for the curious, was on the Inmos Transputer. You may have heard of this device as a very fast micro especially well adapted to use in multiple sets for parallel processing. with communication between processors over a very fast single bit bus. This device has not really made anywhere near the impact it should have because of high prices and the idiosyncratic policies of its British makers on software. As I understood it they wouldn't even let anyone know what its assembly language was like, and insisted on the use of the special Occam language. This has its virtues, but was incompatible with anything else that programmers ever used, so generally they passed it by, and now mainstream RISC developments may well be doing so to the chips too. Anyhow the device was recently taken over by SGS-Thompson who are much more into mass marketing, prices are coming down to more reasonable levels, and what is more articles have started to appear on its internal architecture and assembly language. The designers explicitly said that the two initial inspirations were the HP calculator stack and the TMS-9900 workspace pointer concept. So there is a 32-bit screamer out there that develops a idea familiar to TI-99 assembly programmers. It really is a pity that TI got stuck in the 99000 dead-end that never seemed to go anywhere. ■

FUNNELWEB 4.31 -READ-ME

by Tony McGovern, Australia

(1) General Notes

FUNNELWEB Vn 4.31 is a powerful operating and utility system for the TI-99/4a computer. It will run on a minimal disk-based system with 32K expansion as an Extended Basic program (TI or Myarc XBII), as an assembly program file (E/A or TI-Wr), or with a Horizon style auto-booting RAMdisk with any or no cartridge at all. It supports in a single seamless system a complete and improved range of standard utilities (E/A, TI-Wr, Disk Manager) and menu/loading facilities for a wide range of other programs. Cartridge swapping from XB is now largely unnecessary. The basic TI-99/4a system was always and still is the easiest of computers to use, and FUNNELWEB makes it just as easy for the advanced facilities too. The more powerful the system the better it goes. An extra set of files is available to exploit the extra power of V9938 based systems, the DIJIT AVPC card, the Mechatronics 80 column unit and the Geneve 9640 in GPL mode. This latest version gives some support for hard disk systems using the Myarc HFDC so that the FUNNELWEB system can be run from its own directory on the hard disk, leaving the "DSK1 emulation" file free for use as work-disk.

The program has been written entirely at Funnelweb Farm and is distributed as "FAIRWARE". It is not to be sold nor distributed with excessive copy fees, nor ADVERTISED as part of ANY commercial sale, nor placed on copy-protected disks. Placing of these files on any electronic network or BBS without explicit permission (to be renewed for each new version) of the authors is expressly forbidden.

The Vn 3.0 and later programs are NOT IN THE PUBLIC DOMAIN, but "fairware", with all rights reserved by the authors. No responsibility is accepted for consequences of its use. Please refer to the Fairware Notes at the end of this file. The FUNNELWEB package is issued with no commercial copyrighted utility programs on it, and we request that the package be passed on in its original and complete form only, complete with ALL document files.

The equipment that you will need to run FUNNELWEB is the same as needed for TI-Writer or E/A except of course that the specific module is not necessary.

Absolutely necessary

(1) TI-99/4a console
(2) 32K memory expansion
(3) Disk drive + controller
(4) Module/RAMdisk to load it

Highly desirable

(4) Two or more disk drives
(5) RS232/PIO and printer
(6) A RAMdisk or several
(7) V9938 80 column system

Other items that you will need to make full use of FUNNELWEB Vn 4.31 are

(1) E/A, TI-Wr manuals
(2) c99-REL4 package
(3) Your utility files

(2) Files and Documentation

The FUNNELWEB package contains a number of files besides this one (46 files in all). These may be wholly or partially archived on the disk, using Barry Boone's Archiver III program - also "fairware",

LOAD, FW - the main program in XB and E/A program formats

ED,EE - files for both Text Editor and Programmer's Editor

FO,FP - Formatter program files

AS,AT - Assembler program files

CF,CG - Configuration program to set preferences in FW and LOAD and to set up User Lists

SYSCON - Sample configuration file for use with CF/CG

C1,C2 - character definition files

QD,QF - Quick Disk Directory called up by <AID> from main program or by the Assembler and Formatter

DR,DS (or perhaps on disk as DR40,DR41) - Disk Review program called from Central Menu Option #8 or as initial autoload

EA,LL,SL - system auxiliary loader files for EA Program and Load/Run, LowLoader, and ScriptLoader

UL,D1 - sample User List files

SCRIPT - sample load script

LH - LineHunter assembly programmers' search utility

FSAVE - improved Save utility for generating E/A program files

CP,C99PFIO;O - auxiliary files for use with Clint Pulley's c99 Rel 4

CT8K/O - installs FUNNELWEB as menu item in E/A + 8K RAM cartridges

LDFW - Dis/Fix 80 loader for FW from Minimem, E/A and Myarc XBII

XB4THLD - loader for TI-Forth from XB cartridge.

Detailed documentation of the various aspects of the program is to be found in the FWDOC files,

FWDOC/LOAD - general information on system and disk organization, the XB user's list, and program-wide facilities

FWDOC/TIWR - use of the package as a substitute for the TI-Writer module

FWDOC/EASM - programmer's editor and assembler operation, and program loading functions

FWDOC/UTIL - notes on utility programs in the package used with FUNNELWEB

FWDOC/REPT - chronicle of bugs, fixes, updates, problems, and background information. Make sure you read this file

FWDOC/DR40 and FWDOC/DR41 - use of the DiskReview menu option in the 40 column version.

The supplementary file collection for 80 column use with the DIJIT AVPC and other V9938 based systems contains

ED,EE (may be found as ED80, ED81 on the distribution disk) - 80 column Editor files

DR,DS (may be as DR80, DR81) - 80 column Disk Review program

FWDOC/EDAV - editor and general docs for 80 column operation with AVPC, Geneve, Mechatronics systems

FWDOC/DR80, /DR81, and /DR82 - docs for 80 col DiskReview.

In order to fit the files on a DSSD disk they have been partially archived. Enough files (-READ-ME, LOAD, FW, ED, EE, DR, DS, QD) are left unarchived to get started on any system. Barry Boone's FAIRWARE Archiver can be used to unpack the other files.

(3) Update Notes

Vn 4.31 is a further refinement and extension of the FUNNELWEB system, and retains full external compatibility with prior versions. Significant changes from Vn 4.30 are

(i) The 40 column DiskReview has been enhanced to match most capabilities of its 80-col big brother, including disk manager and sector editor, and supports single drive file copy as well.

(ii) DM-1000 files MG/MH and Disk-Patch file DP have been dropped as DiskReview fills all essential functions of these. The program files

from Vn 4.30 remain compatible with Vn 4.31 if continued use is desired.

(iii) Improved support is provided for multi-lingual use of the Editors, as the character file names may be edited in the main program as needed. The fives marker in the ruler line has been changed from <fn-A> to ":" so that it is less likely to be redefined.

(iv) The 40-col Editor SD now allows selection of another drive directly by number. Cursor bar motion has been slowed down.

(v) A separate Quick Directory file QF now allows file marking from within the Assembler or Formatter.

(vi) The 80 column DiskReview now loads Myart RLE image files automatically in <V>iew mode, as well as having improved sector editor and disk manager functions.

(vii) Loading of E/A program files from cassette is no longer directly supported, and color changes with <0> no longer occur while LOAD is still in XB mode.

(4) Fairware Notes

Your letters and contributions in appreciation of this program will be welcome. Many suggestions from correspondents have found their way into the system already. If you wish to interface to FUNNELWEB at the assembly language level, consult the FWDOC/REPT file and articles (Living with Spiders) that have been published (to be updated for this version) and contact us for further details if necessary.

The fundamental design policy remains that all capabilities be accessible with the XB module in place, and to create a system which makes the most of the strengths of the TI-99 system, without imitating any other OS. The FUNNELWEB system is naturally designed to enhance the hardware that we have here and developed on that hardware. Extensions to exploit fully new hardware really require hands-on development, but even so wherever possible system extensions are made for significant new or other hardware that comes to our notice, using whatever information is available on device capabilities and foibles, and just flying blind. The system in use has 2 Horizon 192K RAMdisks, Myarc 512K RAMdisk, TI RS232 card, Myarc FDC, 2 Chinon DSDD and 2 National 5" 80 track drives, various E/A + RAM cartridges, and the DIJIT AVPC card. Specific extra program capabilities or bug work-arounds have been made, in the flying blind category, for the Geneve 9640, and Corcomp FDC. GRAM devices have not been explicitly considered in their absence. A Myarc HFDC, 10 Mbyte hard disk and Mechatronics 80-Z unit are now available for development work.

No tangible assistance or direct cooperation for the FUNNELWEB development has come from any hardware maker, with the exception of DIJIT Systems, whose AVPC card has given a major new direction to the programming effort and reason for staying with the TI-99/4a. We have no interest of any kind in DIJIT's sales other than to wish them the highest success. Also Myarc Inc returned the XB-II files to us on our own disk with no other communication ever, following repeated queries on XB-II and advice to them of disk DSR bugs. Gary Bowser of OPA provided the Vn 8.14 HRD ROS. All support other than these items has come from individuals and User Groups.

Any commercial software producer wishing to claim compatibility of a product with Funnelweb in advertising is advised to have us check this first.

As for each "fairware" user's obligations, we can suggest only that you judge the program on its intrinsic merits, best measuring its worth by how much you use it as compared to other "fairware" or commercial programs that you have. Even individual components of the package are as substantial programs as many that are offered as commercial items at $10-20 or more apiece. If contributions are made

SWEDLOW EXTENDED BASIC

```
X   X   BBBB      # 19 to 20
 X X    B   B
  X     BBBB      By
 X X    B   B     Jim
X   X   BBBB      Swedlow
```

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

ON FREEWARE

Last year I wrote an Extended Basic program that prints a Multiplan spreadsheet sideways. I wrote it because I was tired of seeing new products for those other computers but not the 99/4A.

After a great deal of thought, I decided to release it as freeware. As I write this (about a month before you read it), I have shipped over 100 copies of my program and, based on user input, revised it twice.

The experience exceeded my expectations. I have had requests from over 30 states and Canada and have received some very nice letters. It's nice to know that there are others out there who still use the 4A.

I'd like to pass on some suggestions should you write for freeware.

IF YOU SEND MONEY: First and foremost, please print clearly. I agonized over deciphering two addresses. Better yet, if you have one, send a printed return address label. You could use this month's program to make some.

IF YOU SEND A DISK: Initialize the disk. This makes sure that it is OK. Sweeping the disk is not sufficient, format it and verify the sectors.

While you are at it, why not put a few public domain programs on the disk? Three folks sent me some programs that were really nice. Do include a note so that the recipient will check the disk, however.

WRITE A NOTE: I released my program to share something with the TI community. The notes I received were worth their weight in gold. On the other hand, the guy who sent me a check without any kind of note did not make any points!

GIVE FEEDBACK: Take the time to let the programmer know what you thought of the program. If you liked it, say so. If it didn't work, let him know. If you have a suggestion, pass it on.

PAY FOR THE PROGRAM: Some freeware/shareware programmers ask you to pay $10 or so if you like the program. Considering that most commercial software starts at $20, this is a bargain. Some of the best new items come from freeware/shareware. Without your support, this source just might dry up.

---

by check or IMO (the best way from the USA) they should be made payable to Tony or Will, or Will alone (rather than "and").

All letters needing an answer will be responded to sooner or later, but we just can't afford the time for or cost of routine mailing of updates and don't volunteer to do so. If and when updates are issued they will be placed in distribution through our regular, and/or recent contacts, and we do not offer to provide distribution from Funnelweb Farm at near nominal cost or less. Our normal practice is to release improvements as soon as they are made. If you write new utilities for FUNNELWEB please to let us know the details.

Nov 16th / 1990
Tony and Will McGOVERN
215 Grinsell St.,
Kotara, NSW 2289
AUSTRALIA

■

ON PRE-SCAN AND DIM

If you are not familiar with pre-scan, read the supplement to the XB manual and my column in the June, 1985 ROM.

Your TI does not execute a DIM statement. The only time it reads the DIM is during pre-scan. Therefore, DIM can be part of the pre-scan list.

For those arrays that have ten values, you do not need to DIMention them. You only need to put A() in the pre-scan list. The parenthesis can be completely empty (assuming a one dimention array, that is).

This, then, is a legal, functional pre-scan list:

```
100 GOTO 110 :: A,B,C$,C(),
    D$ :: DIM Q(12) :: !@P-

110 ! Program continues
```

This program, of course, uses no subroutines. For another example, see lines 150 to 180 of this months program.

99-CALC

I just received a copy of 99-CALC, a freeware spreadsheet by Phil Barnes. I will let you know how it works as soon as I test it out.

Much to my surprise , I discovered that Phil authored the original disk label program that came from our user group library and was the basis of this month's program and one I published here previously. I didn't note the source before because I didn't know it. This should correct that error.

Should you want 99-CALC right away send an initialized disk, a self-addressed mailer and return postage to:

Phil Barnes
24631 Via San Fernando
Mission Viejo, CA 92692

LABEL

I haven't published a program in a few months -- my entry into the freeware arena took up more time than I expected.

I have previously published this program as two separate ones: one that prints disk labels and one that did mailing labels. It got to be a chore changing programs so I combined them.

CUSTOMIZING: Substitute your name and address in the DATA statements in line 170. Be sure and keep the leading and trailing commas (unless you use a four or five line address). Your address lines should be no more than 27 characters long.

Lines 190 to 260 are the printer control codes. They work for most Epson and compatible printers. If you printer uses different codes, change them as necessary.

If your printer name is not PIO, modify line 280 as needed.

DEBUGGING: If you run into problems, delete the <:: !@P-> at the end of line 180 to keep pre-scan on for the entire program and change <ON ERROR 550> to <ON ERROR STOP> in line 360. This should help you find any errors.

PROGRAM NOTES: Lines 520 and 530 do some interesting things:

ON POS is used to validate the correct key return. The value of I then is used, after returning from the subroutine, as the value in <ON I GOTO . . .>.

The second CALL KEY in line 530 remaps the keyboard as a 99/4A so that you can input lower case for your labels.

By changing 13 and 32 to 80 (or <ENTER> and <SPACE BAR> to <P>) in line 520, you can start printing by pushing P or ENTER or the SPACE BAR.

```
********************************
* Programlistning för LABEL 3.0 *
* finns i PB 89-1 sid 18        *
********************************
```

ERROR TRAPPING

We have talked here before about making your programs 'user proof'. No matter what the user does, your program should have a defense. A while back I covered one area of vulnerability - when the user inputs something from the keyboard. This month the subject is error trapping.

Say, for example, that the program must access a disk file to run. Fred Klutz, your program's user, puts the wrong disk in the drive (or doesn't put any disk in). What happens? Well, your program opens a disk file and the Disk Controller goes to the specified drive to look for the file. When it doesn't find it, program execution stops, an error message appears on the screen and any data held in memory is virtually lost.

There is a way around this. Two XB commands can let you decide what happens when an error occurs: ON ERROR nnn and CALL ERR().

The default condition for ON ERROR is ON ERROR STOP. This means that if an error occurs, program execution stops and an error message is displayed. The alternative is ON ERROR nnn, where 'nnn' is a line number. With this, when an error occurs, program execution transfers to the specified line number.

I do not fully understand error trapping. I can use it but I don't understand it. Once you get the hang of error trapping, try intentionally causing an error with TRACE active. You will see that the computer does not exactly go to the error instructions even though it follows them.

Here is an example of how ON ERROR works:

```
200 INPUT "File Name:  ":A$
210 ON ERROR 500 ::
    OPEN #1,"DSK1."&A$

Program Continues

500 ! Error Instructions
510 PRINT "Could not find DSK1.";A$
520 ON ERROR 540 :: CLOSE #1
530 ON ERROR STOP :: RETURN 200
540 RETURN 530
```

Fearless Fred inputs a bogus file name in line 200. We set the error trap in line 210. Our 4A tries to open a file in line 210 but can't find it. Control transfers to line 500.

First we tell Fleckless Fred that the file name was bad. Then we try and close the file. The code may seem odd, but it works. Sometimes, if you don't close the file an error will occur when you re-OPEN it but closing the file will also cause an error. So we put in an ON ERROR before closing the file just in case.

You have three options with RETURN in an ON ERROR routine. RETURN by itself will send you back to the instruction that caused the error. RETURN NEXT will return you to the very next instruction. And RETURN nnn will return you to line number nnn. These RETURN's do not work with GOSUB.

ON ERROR executes like a GOSUB. You could end the error language with a GOTO but you would create a pending RETURN that eats memory just as it does if GOSUB is not followed by RETURN. It could cause a problem if you use GOSUB later in the program!

Why the ON ERROR STOP in line 530, you ask. Well, once an ON ERROR nnn is triggered by an error, error control reverts back to ON ERROR STOP. However, I never know if the CLOSE #1 will cause an error condition and I don't want the ON ERROR 540 to be active after the file is closed, so I override it just to be safe.

Back to program flow. We had an error when opening the file, we told the user, we closed the file and we returned back to asking for a file name. The process starts over. If Fleabit Fred inputs a good file name, our program can continue. Anticipating a problem, we reset the error trap in line 210.

There is another tool you can use after ON ERROR has transferred control to error trapping language. It is CALL ERR(A,B,C,D). Look it up in

your XB manual. It can tell you the error type, the line number in which the error occured and the file number associated with the error if it is an I/O error. This information can be quite valuable in deciding what to do with an error.

A couple words of caution. First, do not add error trapping language to your program until you have completely debugged it. Otherwise, other errors in the program will be very difficult to locate.

Second, your TI executes ON ERROR STOP until you give it other instructions. In our sample program above, an error before line 210 would not trigger the error trapping language in line 500. Also, the ON ERROR 500 remains in effect until an error occurs or you execute another ON ERROR statement. This means that if an error occurs any where after line 210, the error message, "Could not find . . ." will appear even if it is not appropriate.

If you have more questions, just as when all else fails, read the manual -- it does give good information about XB.

ON PRE-SCAN AND USER SUBS

The XB book says that the first CALL for ALL subprograms must be within the active pre-scan area. It also states that all SUB and SUBEND statements must be pre-scaned. In debugging a program, I found that the first CALL of a user-defined sub does not need to be pre-scaned. Only CALLs of BUILT-IN subs must be pre-scaned.

This program will run without a hitch:

```
100 !@P-
110 CALL TEST :: END
120 !@P+
130 SUB TEST :: PRINT "OK" ::
    SUBEND
```

FREEWARE REVIEW: 99-CALC

99-CALC is a spreadsheet written for the 4A. What makes this program valuable is that it will run on a bare bones system (no memory expansion and no disk drive). It requires only a cassette player and Extended Basic. A printer, disk drive, and memory expansion are optional.

Given the small size of the requirements, 99-CALC compares well with full featured programs. You can do arithmetical functions (add, divide, subtract , multiply and per-cent) and you can total and average columns and rows.

You have full screen editing, can move from one cell to an adjacent one or jump to any active cell. The spreadsheet can be printed on a 80 column printer. You can opt to print formulas.

Phil Barnes, who wrote 99-CALC has truly come up with an ingenious program. He uses every bit of memory possible. For example, you can hide numbers under column and row titles. The program is filled with nice touches. 99-CALC comes with on disk documentation (that prints 7 pages) and a sample file.

RECOMMENDATION: If you do not have memory expansion and have need of a spread sheet program, 99-CALC would be a valuable addition to your library.

For those of you with 32K and a disk drive, Multiplan has more features. 99-CALC is worth getting if for no other reason than to see what can be done with the 4A's 16K of VDP RAM. 99-CALC is also much easier to master.

HOW TO ORDER:

Phil Barnes
24631 Via San Fernando
Mission Viejo, CA 92692

Phil asks for a donation of $10 (plus cost of disk and postage) if you like the program. It is well worth it!

Enjoy! ■

TIPS FROM TIGERCUB #40

TIGERCUB SOFTWARE
156 Collingwood Ave.
Columbus, OH 43213, USA

When Texas Instruments developed Extended Basic, they took away the ability of Basic to redefine or color the characters in sets 15 and 16, ASCII 144 to 159, in order to make room in memory for sprites (they did let us have color set 0 instead. That is why Basic programs which use sets 15 and 16 will crash if you try to run them in XBasic.

Finally, John Behnke published in the Chicago Times newsletter an amazing routine which gave us back those missing sets. His routine was 13 sectors long. Recently, Richard Heath published in the L.A. newsletter a shortened version. And, without having any idea how it works, I have managed to scrunch it down to only 4 sectors -

```
1 CALL BXB
29999 !BXB by Jim Peterson, adapted from VDPUTIL2 by John Behnke/Richard Heath
30000 SUB BXB :: CALL INIT :: CALL LOAD(8194,37,194,63,240)
30001 CALL LOAD(16368,80,79,67,72,65,82,37,58,80,79,75,69,86,32,37,168)
30002 !
30003 FOR J=1 TO 136 :: CALL LOAD(9529+J,ASC(SEG$(][\[]$,J,1))):: NEXT J :: SUBEND
30004 SUB CHAR(A,A$):: CALL LOAD(9500,A):: CALL LINK("POCHAR",A$):: SUBEND
30005 SUB COLOR(A,B,C):: CALL LOAD(9492,8,15+A,(B-1)*16+C-1)
30006 CALL LINK("POKEV"):: SUBEND
```

Note than line 30002 is missing. That's because there is no way to key it in. Once again we need a program that writes a program -

```
100 FOR J=1 TO 136 :: READ A :: M$=M$&CHR$(A):: NEXT J
110 OPEN #1:"DSK1.BXBDATA",VARIABLE 163,OUTPUT :: PRINT #1:CHR$(117)&CHR$(50)&"][\[]$"&CHR$(190)&CHR$(199)&CHR$(136)&M$&CHR$(0)
120 PRINT #1:CHR$(255)&CHR$(255):: CLOSE #1
130 DATA 2,224,37,20,3,0,0,0,2,5,48,48,2,6,37,2,205,133,2,134,37,17
140 DATA 17,252,4,192,2,1,0,1,2,2,37,1,2,3,18,0,212,131,4,32,32,20
150 DATA 208,4,9,80,2,32,3,0,2,1,37,2,2,2,0,8,2,7,11,0,2,8,7,0,193
160 DATA 1,192,193,193,180,97,133,145,135,21,1,113,136,6,198,145
170 DATA 135,21,1,113,136,210,70,10,198,177,137,220,198,2,131,37,10
180 DATA 17,240,4,32,32,36,16,6,2,224,37,20,3,0,0,0,4,32,32,32,4
190 DATA 192,216,0,131,124,2,224,131,224,4,96,0,112
```

RUN that to create a file BXBDATA on the disk. Then load the BXB program, and enter MERGE DSK1.BXBDATA. The unprintable line will pop into place. SAVE this completed BXB routine in MERGE format, and merge it into any Basic-only program. If you want, the result can be run through a Compactor program and turned into multi-statement program lines for more speed.

Or, you can write an Extended Basic program using all 16 character sets for graphics and color - actually 17, because set 0 is also available. Even the characters 24 through 31 can be redefined! Craig Miller has warned against fooling around in that area of memory, but there seems to be no problem with redefining the cursor (30) or the edge character (31).

Sprites can only use

characters between 32 and 143 and their color cannot be changed with CALL COLOR(#_,_). I have not found any other bugs, but have not had time for much experimenting.

Here's an easy Tigercub challenge - run this one in Basic, not Extended Basic.

```
>LIST
100 DISPLAY AT(1,1):0
>RUN
  0
  0
```

Why did it print the zero twice?

I wrote this next one primarily for blind users. It converts each PRINT or DISPLAY directly to speech output and also provides a speech prompt for INPUTs.

```
100 !PRINT SPEAKER by Jim Peterson - to add OPEN #1:"SPEECH",OUTPUT and convert PRINT and DISPLAY statements to PRINT #1
110 !Also writes a PRINT #1 for INPUT prompts
120 !Program to be converted must first be SAVEd in MERGE format. Recommend it be RESequenced before SAVEing, to make room for INPUT lines
130 PS$=CHR$(156)&CHR$(253)&CHR$(200)&CHR$(1)&"1"&CHR$(181)
140 DISPLAY AT(3,1)ERASE ALL:"INPUT FILENAME?":"DSK" :: ACCEPT AT(4,4):IF$ :: OPEN #1:"DSK"&IF$,INPUT ,VARIABLE 163
150 DISPLAY AT(5,1):"OUTPUT FILENAME?":"DSK" :: ACCEPT AT(6,4):OF$ :: OPEN #2:"DSK"&OF$,OUTPUT,VARIABLE 163
160 PRINT #2:CHR$(0)&CHR$(1)&CHR$(159)&CHR$(253)&CHR$(200)&CHR$(1)&"1"&CHR$(181)&CHR$(199)&CHR$(6)&"SPEECH"&CHR$(179)&CHR$(247)&CHR$(0)
170 LINPUT #1:M$ :: P=POS(M$,CHR$(156),3):: A=POS(M$,CHR$(162),3):: Z=POS(M$,CHR$(181),3)
180 I=POS(M$,CHR$(146),1):: IF I=0 THEN 210 :: IF Z=0 OR Z<I THEN PRINT #2:M$ :: GOTO 240
190 M2$=SEG$(M$,1,1)&SEG$(M$,2,1)&PS$&SEG$(M$,I+1,Z-I-1)&CHR$(0):: PRINT #2:M2$
200 PRINT #2:SEG$(M$,1,1)&CHR$(ASC(SEG$(M$,2,1))+1)&SEG$(M$,3,255):: GOTO 240
210 IF P+A=0 THEN PRINT #2:M$ :: GOTO 240
220 M=MAX(P,A)
230 M$=SEG$(M$,1,2)&PS$&SEG$(M$,M+1,255):: PRINT #2:M$
240 IF EOF(1)<>1 THEN 170 ELSE CLOSE #1 :: CLOSE #2
250 DISPLAY AT(12,1)ERASE ALL:"Type NEW and Enter" :: DISPLAY AT(15,1):"Type MERGE DSK";OF$ :: END
```

```
*****************************
      MOLLY DARLING
100 CALL CLEAR :: CALL SCREEN(5):: FOR SE=1 TO 12 :: CALL COLOR(SE,16,5):: NEXT SE
110 DISPLAY AT(3,8):"MOLLY DARLING": :" Written and performed by": :TAB(9);"Eddy Arnold" :: DISPLAY AT(24,1):"Programmed by Jim Peterson"
120 FOR D=1 TO 200 :: NEXT D :: DISPLAY AT(12,1):"Just a moment.....": :".....looking for my music..."
130 DIM N(100),N2(100),A(250),B(250),C(250):: F=110 :: FOR J=1 TO 80 :: N(J)=INT(F*1.059463094^(J-1)+.5):: NEXT J
140 DATA 16,11,8,16,8,11,16,4,11,18,11,8
150 DATA 20,16,11,23,11,16,25,21,16,28,16,21
160 DATA 23,20,16,23,16,20,23,11,16,23,16,11
170 DATA 20,11,16,20,16,11,20,8,11,20,11,8
180 DATA 20,11,16,25,16,11,23,11,16,20,8,4
190 DATA 18,16,10,18,10,16,18,16,10,18,11,16
200 DATA 18,15,11,18,9,15,18,11,9,18,9,3
210 DATA 28,8,1,28,13,8,28,8,13,28,13,4
220 DATA 27,20,18,27,18,20,20,18,12,20,12,18
230 DATA 25,21,16,25,16,21,25,13,16,25,16,13
240 DATA 27,23,21,27,21,23,27,23,18,27,18,21
250 DATA 28,23,20,28,20,23,28,20,16,27,16,20
260 DATA 30,21,13,28,13,21,2
```

```
7,21,13,25,13,21
270 DATA 23,20,16,23,16,20,2
0,11,16,20,16,11
280 DATA 30,23,13,28,13,23,2
3,20,13,20,13,16
290 DATA 25,21,16,25,16,21,2
5,21,16,27,16,21
300 DATA 28,23,20,20,16,11,1
8,15,11,20,11,15
310 DATA 16,11,8,16,8,11,16,
9,1,16,1,9
320 DATA 16,11,8,16,8,11,16,
1,8,16,13,1
330 DATA 25,21,16,25,16,13,2
5,13,9,25,9,4
340 DATA 23,20,16,23,16,11,2
3,11,8,23,8,4
350 DATA 21,18,11,21,11,9,21
,9,6,20,6,3
360 DATA 21,16,11,20,16,11,2
0,11,8,20,8,4
370 DATA 18,13,10,18,10,6,18
,6,1,20,13,10
380 DATA 22,18,13,28,22,18,2
7,18,22,25,22,18
390 DATA 23,18,15,23,15,11,2
3,11,6,23,6,3
400 DATA 23,21,15,23,15,11,2
3,11,9,23,9,6
410 DATA 16,13,8,16,8,13,16,
13,8,18,13,9
420 DATA 20,11,8,21,8,11,20,
11,8,18,11,6
430 RESTORE 140 :: T=16 :: G
OSUB 480 :: RESTORE 140 :: T
=4 :: GOSUB 480 :: RESTORE 1
80 :: T=12 :: GOSUB 480 :: R
ESTORE 140 :: T=16 :: GOSUB
480
440 RESTORE 210 :: T=28 :: G
OSUB 480 :: RESTORE 170 :: T
=4 :: GOSUB 480 :: RESTORE 2
50 :: T=4 :: GOSUB 480 :: RE
STORE 280 :: T=4 :: GOSUB 48
0 :: RESTORE 190 :: T=8
450 GOSUB 480 :: RESTORE 140
 :: T=16 :: GOSUB 480 :: RES
TORE 290 :: T=48 :: GOSUB 48
0 :: RESTORE 140 :: T=16 ::
GOSUB 480 :: RESTORE 410 ::
T=8 :: GOSUB 480
460 RESTORE 310 :: T=8 :: GO
SUB 480 :: GOTO 490
470 GOTO 490
480 FOR J=1 TO T :: X=X+1 ::
 READ A(X),B(X),C(X):: A(X)=
A(X)+12 :: B(X)=B(X)+12 :: C
(X)=C(X)+12 :: NEXT J :: RET
URN
490 DISPLAY AT(10,1):"Contro
l volume of 3 voices":"using
 1, 2 and 3 keys for":"loude
r and Q, W and E for":"softe
r.":""
500 DISPLAY AT(15,1):"Contro
l speed using `F' for":"fast
er and `S' for slower."
510 DISPLAY AT(18,1):"Change
 key using `A' for":"higher
and `D' for lower."
520 DISPLAY AT(21,1):"Press
`Z' for minor key, `X'":"for
 major key." :: V1,V2,V3=10
:: F,P,Y=0 :: X=200
530 FOR J=1 TO 192 :: CALL S
OUND(-999,N(A(J)-Y),V1,N(B(J
)-Y),V2,N(C(J)-Y),V3):: FOR
T=1 TO X/50 :: P=1^X :: NEXT
 T
540 CALL KEY(0,K,S):: IF S<1
 THEN 710 :: ON POS("123QWEF
SADZX",CHR$(K),1)+1 GOTO 710
,550,560,570,580,590,600,610
,620,630,650,670,690
550 V1=V1-1-(V1=0):: GOTO 71
0
560 V2=V2-2-(V2=0)*2 :: GOTO
 710
570 V3=V3-2-(V3=0)*2 :: GOTO
 710
580 V1=V1+2+(V1=30)*2 :: GOT
O 710
590 V2=V2+2+(V2=30)*2 :: GOT
O 710
600 V3=V3+2+(V3=30)*2 :: GOT
O 710
610 X=X-20-(X<2)*20 :: GOTO
710
620 X=X+20 :: GOTO 710
630 IF F=1 THEN GOSUB 700
640 Y=Y-1-(Y=-20):: GOTO 710
650 IF F=1 THEN GOSUB 700
660 Y=Y+1+(Y=6):: GOTO 710
670 IF F=1 THEN 710 :: GOSUB
 680 :: GOTO 710
680 F=1 :: Y=0 :: FOR W=3 TO
 27 STEP 12 :: N2(W)=N(W)::
N(W)=N(W-1):: N2(W+5)=N(W+5)
:: N(W+5)=N(W+4):: N2(W+10)=
N(W+10):: N(W+10)=N(W+9):: N
EXT W :: RETURN
690 IF F=0 THEN 710 :: GOSUB
 700 :: GOTO 710
700 F=0 :: FOR W=3 TO 27 STE
P 12 :: N(W)=N2(W):: N(W+5)=
N2(W+5):: N(W+10)=N2(W+10)::
 NEXT W :: RETURN
710 NEXT J :: J=192 :: FOR V
=10 TO 30 :: CALL SOUND(-999
,N(A(J)-Y),V,N(B(J)-Y),V,N(C
(J)-Y),V):: NEXT V :: FOR D=
1 TO 500 :: NEXT D :: GOTO 5
30
```

Jim Peterson■

MELTDOWN – SPEL FÖR XB

av Steve Langguht, USA

MED JOYSTICK (ALPHA LOCK uppe):
Flytta med upp och ned.
Hämta syre genom att passera.
Skjut med vänster eller höger.

MED TANGENTER (ALPHA LOCK nere):
Flytta med E och X.
Hämta syre genom att passera.
Skjut med L och P.

```
100 REM ************
110 REM * MELTDOWN *
120 REM *          *
130 REM ************
140 REM REM BY STEVE LANGGUH
T
150 REM HCM
160 REM VERSION 4.1.1
170 REM X-BASIC
180 REM
190 REM **TITLE SCREEN**
200 REM
210 CALL CLEAR :: DISPLAY AT
(11,11):"MELTDOWN"
220 REM
230 CALL CHAR(43,"0000FF8181
FF0000",37,"007E7E42427E7E00
")
240 CALL CHAR(96,"0306060301
3F34353437030303030FC06060
C080FC2CAC2CE0606060780000")
250 CALL CHAR(100,"030606030
13F3435340706061E0000C0606
0C080FC2CAC2CECC0C0C0C0C0F0"
)
260 CALL CHAR(104,"000000000
00038EE0300000000000000000
000000038EE8300000000000000"
)
270 CALL CHAR(108,"0103060C1
93366CCCC6633190C06030180C06
03098CC66333366CC983060C080"
)
280 CALL CHAR(132,"66",116,"
66",124,"66")
290 CALL CHAR(112,"001F20404
04E5B51515B4E4040201F0000F80
402020202324A12227A0204F800"
)
300 CALL CHAR(120,"0000103B1
2021E10101E02123B10000000000
8DC484078080878404 8DC080000"
)
310 CALL CHAR(140,"C1E171381
C0E07E3E3070E1C3871E1C183878
E1C3870E0C7C7E070381C8E8783"
)
320 CALL CHAR(128,"012110080
204011BC309041121400000000000
284882090C3D880204010088480"
)
330 CALL CHAR(136,"000012121
23F7FD5D57F3F121212000000004
84848FCFEABABFEFC4848480000"
)
340 REM INITIALIZATION
350 RESTORE :: OPTION BASE 1
 :: DIM ROW(12):: DIM COL(12
):: DIM VEL(12):: DIM MP(12)
:: DIM PPR(6):: DIM PPC(6)
360 FOR X=1 TO 12 :: READ RO
W(X),COL(X),VEL(X):: NEXT X
370 FOR X=1 TO 12 :: READ MP
(X):: NEXT X
380 FOR X=1 TO 6 :: READ PPR
(X),PPC(X):: NEXT X
390 REM KEY OR JOYST
400 CALL CLEAR :: DISPLAY AT
(7,6):"DO YOU WISH TO USE" :
: DISPLAY AT(9,10):"1) JOYST
ICK "
410 DISPLAY AT(11,3):"-NOTE-
RELEASE ALPHA LOCK-"
420 DISPLAY AT(14,10):"2) KE
YBOARD ?" :: DISPLAY AT(16,6
):"PRESS 1 OR 2"
430 CALL KEY(0,K,S):: IF K<>
49 AND K<>50 THEN 430
440 IF K=49 THEN K1=1 ELSE K
1=0
450 REM DRAW PLAY FIELD
460 CALL COLOR(1,2,2,2,2,2,1
1,2,2,12,2,2,13,2,2)
470 CALL CLEAR :: CALL SCREE
N(2):: CALL MAGNIFY(3)
480 FOR T=1 TO 4 :: READ R1,
C1 :: CALL HCHAR(R1,C1,37,14
):: NEXT T
490 FOR T=1 TO 10 :: READ R2
,C2 :: CALL HCHAR(R2,C2,43,1
4):: NEXT T
500 FOR T=1 TO 4 :: READ R3,
C3 :: CALL VCHAR(R3,C3,37,2)
:: NEXT T
510 FOR T=1 TO 6 :: READ R4,
C4,CH4 :: CALL HCHAR(R4,C4,C
H4,15):: NEXT T
520 FOR T=1 TO 6 :: READ R5,
C5 :: CALL SPRITE(#(T+5),140
,2,R5,C5):: NEXT T
530 CALL COLOR(1,5,2,2,5,2)
540 FOR X=3 TO 8 :: CALL COL
```

```
OR(X,16,2):: NEXT X
550 ARMOR=8 :: OXYGEN=200 ::
 SCORE=0 :: LEVEL=1 :: PN=0
:: OF=0 :: AM=0
560 DISPLAY AT(23,19)SIZE(5)
:"SCORE" :: DISPLAY AT(22,1)
SIZE(6):"OXYGEN"
570 DISPLAY AT(24,2)SIZE(5):
"ARMOR"
580 DISPLAY AT(22,7)SIZE(4):
OXYGEN :: DISPLAY AT(24,8)SI
ZE(3):ARMOR
590 DISPLAY AT(12,14)SIZE(2)
:LEVEL :: CALL SOUND(100,(33
0*LEVEL),2,(392*LEVEL),2,(52
3*LEVEL),2)
600 GOSUB 1950
610 CALL SOUND(25,262,30)::
CALL SOUND(500,(330*LEVEL),2
,(392*LEVEL),2,(523*LEVEL),2
)
620 FOR T=1 TO 250 :: NEXT T
630 DISPLAY AT(12,14)SIZE(2)
:"  "
640 REM CREATE HERO
650 CALL SPRITE(#1,96,3,165,
121):: S=0 :: SH=0
660 PN=PN+1 :: IF PN>(15*LEV
EL)THEN 1620
670 REM CREATE OXYGEN
680 IF OF=1 THEN 730
690 RANDOMIZE :: Q=INT(RND*3
)+1 :: P=INT(RND*6)+1
700 IF Q<>2 THEN 730
710 CALL SPRITE(#4,112,12,PP
R(P),PPC(P)):: OF=1 :: CALL
SOUND(50,3000,3,4000,5)
720 REM GAMMA RAYS
730 RANDOMIZE :: X=INT(RND*2
)+1 :: IF X<>2 THEN 910
740 RANDOMIZE :: Y=INT(RND*3
)+1 :: ON Y GOTO 750,800,850
750 CALL COLOR(#6,16):: CALL
 SOUND(200,2500,2):: CALL CO
LOR(#7,16):: CALL SOUND(200,
2600,2)
760 GOSUB 1770
770 CALL COLOR(11,16,2):: CA
LL SOUND(100,1000,2,1200,4):
: CALL POSITION(#1,DR,DC)
780 IF (DR<20)+(DR>30)=0 OR(
DR<92)+(DR>102)=0 THEN CALL
COLOR(11,2,2):: CALL COLOR(#
6,2,#7,2):: GOTO 1340
790 CALL COLOR(11,2,2):: CAL
L COLOR(#6,2,#7,2):: GOTO 91
0
800 CALL COLOR(#8,16):: CALL
 SOUND(200,2700,2):: CALL CO
LOR(#9,16):: CALL SOUND(200,
2800,2)
810 GOSUB 1770
820 CALL COLOR(12,16,2):: CA
LL SOUND(100,1000,2,1200,4):
: CALL POSITION(#1,DR,DC)
830 IF (DR<44)+(DR>54)=0 OR(
DR<116)+(DR>126)=0 THEN CALL
 COLOR(12,2,2):: CALL COLOR(
#8,2,#9,2):: GOTO 1340
840 CALL COLOR(12,2,2):: CAL
L COLOR(#8,2,#9,2):: GOTO 91
0
850 CALL COLOR(#10,16):: CAL
L SOUND(200,2900,2):: CALL C
OLOR(#11,16):: CALL SOUND(20
0,3000,2)
860 GOSUB 1770
870 CALL COLOR(13,16,2):: CA
LL SOUND(100,1000,2,1200,4):
: CALL POSITION(#1,DR,DC)
880 IF (DR<68)+(DR>78)=0 OR(
DR<140)+(DR>150)=0 THEN CALL
 COLOR(13,2,2):: CALL COLOR(
#10,2,#11,2):: GOTO 1340
890 CALL COLOR(13,2,2):: CAL
L COLOR(#10,2,#11,2):: GOTO
910
900 REM ANTIMATTER
910 IF LEVEL<2 THEN 970
920 RANDOMIZE :: X=INT(RND*2
)+1 :: Y=INT(RND*(8-LEVEL))+
1 :: Z=INT(RND*6)+1
930 IF AM<>1 THEN 950
940 IF X=2 THEN CALL DELSPRI
TE(#5):: CALL SOUND(50,3333,
2,4444,3):: AM=0 :: GOTO 970
 ELSE GOTO 970
950 IF Y=1 THEN CALL SPRITE(
#5,120,6,PPR(Z),PPC(Z)):: AM
=1 :: CALL SOUND(50,2222,2,3
333,3)
960 REM PARTICLE CHOOSE
970 RANDOMIZE :: X=INT(RND*(
LEVEL*2))+1 :: ON X GOTO 980
,980,980,980,990,980,990,990
,990,990
980 CN=108 :: GOTO 1010
990 CN=136 :: GOTO 1010
1000 REM MAIN GAME LOOP
1010 GOSUB 1770
1020 RANDOMIZE :: X=INT(RND*
12)+1 :: Y=INT(RND*13)+3 ::
CALL SPRITE(#3,CN,Y,ROW(X),C
OL(X),0,VEL(X)*(5+LEVEL))
1030 IF CN=108 THEN 1040 ELS
E 1050
1040 FOR A=0 TO 30 STEP 15 :
: CALL SOUND(-99,250+Y*10,A,
760+Y*10,A,1512+Y*10,A):: NE
XT A :: GOTO 1060
1050 FOR T=1 TO 3 :: CALL SO
UND(25,5000,2,5555,3,6000,2)
:: NEXT T
1060 IF SH<>0 THEN CALL COIN
```

```
C(#2,#3,15+LEVEL,HIT1):: IF
HIT1=-1 THEN 1250
1070 GOSUB 1850
1080 IF OF<>1 THEN 1090 ELSE
 CALL COINC(#1,#4,15,HIT3)::
 IF HIT3=-1 THEN 1410
1090 IF AM<>1 THEN 1100 ELSE
 CALL COINC(#1,#5,10,HIT5)::
 IF HIT5=-1 THEN AM=0 :: GOT
O 1440
1100 GOSUB 1770
1110 CALL COINC(#1,#3,15,HIT
2):: IF HIT2=-1 THEN 1340
1120 IF SH<>0 THEN CALL COIN
C(#2,#3,15+LEVEL,HIT1):: IF
HIT1=-1 THEN 1250
1130 GOSUB 1850
1140 CALL POSITION(#2,DR2,DC
2):: IF DC2>40 AND DC2<210 T
HEN 1160
1150 CALL DELSPRITE(#2):: SH
=0
1160 GOSUB 1770
1170 CALL POSITION(#3,DR3,DC
3):: IF DC3>25 AND DC3<225 T
HEN 1220
1180 CALL DELSPRITE(#3):: IF
 SCORE<5 THEN SCORE=0 ELSE S
CORE=SCORE-5
1190 OXYGEN=OXYGEN-1 :: DISP
LAY AT(23,24):SCORE
1200 DISPLAY AT(22,7)SIZE(4)
:OXYGEN :: RH=0
1210 IF OXYGEN<1 THEN 1440 E
LSE 660
1220 OXYGEN=OXYGEN-1 :: DISP
LAY AT(22,7)SIZE(4):OXYGEN :
: IF OXYGEN<1 THEN 1440
1230 CALL SOUND(40-(LEVEL*3)
,1000+(LEVEL*200),2,500+(LEV
EL*150),2)
1240 CALL SOUND(40-(LEVEL*3)
,500+(LEVEL*200),2,250+(LEVE
L*150),2):: GOTO 1060
1250 IF CN=108 THEN 1280
1260 IF RH=1 THEN 1280 ELSE
CALL DELSPRITE(#2):: RH=1 ::
 SCORE=SCORE+10 :: CALL COLO
R(#3,16)
1270 CALL MOTION(#3,0,VEL(X)
*(7+LEVEL)):: GOTO 1060
1280 REM DESTROY PARTICLE
1290 CALL DELSPRITE(#2):: CA
LL MOTION(#3,0,0,#1,0,0):: R
H=0
1300 FOR T=1 TO 5 :: CALL CO
LOR(#3,2):: CALL SOUND(-25,1
111,2)
1310 CALL COLOR(#3,10):: CAL
L SOUND(-25,1122,4):: NEXT T
1320 CALL PATTERN(#3,128)::
FOR T=1 TO 5 :: CALL SOUND(-
99,500*T,2+T):: NEXT T
1330 CALL SOUND(50,4000,2)::
 CALL DELSPRITE(#3):: SH=0 :
: SCORE=SCORE+10 :: DISPLAY
AT(23,24):SCORE :: GOTO 660
1340 REM DESTROY HERO
1350 CALL DELSPRITE(#3):: CA
LL MOTION(#1,0,0)
1360 FOR T=1 TO 5 :: CALL CO
LOR(#1,2):: CALL SOUND(-99,7
50,2)
1370 CALL COLOR(#1,16):: CAL
L SOUND(-99,775,4):: NEXT T
1380 CALL PATTERN(#1,128)::
FOR T=5 TO 1 STEP -1 :: CALL
 SOUND(-99,450*T,1+T):: NEXT
 T
1390 CALL DELSPRITE(#1):: AR
MOR=ARMOR-1 :: DISPLAY AT(24
,8)SIZE(3):ARMOR :: IF ARMOR
<1 THEN 1440 ELSE 650
1400 REM OBTAIN OXYGEN
1410 CALL DELSPRITE(#4):: CA
LL SOUND(100,200,3,300,4)::
OF=0 :: OXYGEN=OXYGEN+10
1420 DISPLAY AT(22,7)SIZE(4)
:OXYGEN :: GOTO 1060
1430 REM END GAME
1440 CALL DELSPRITE(ALL)
1450 FOR T=10 TO 2 STEP -1
1460 CALL SOUND(-1000,-7,T):
: CALL SCREEN(12):: CALL COL
OR(1,12,12,2,12,12,11,12,12,
12,12,12,13,12,12)
1470 CALL SOUND(-1000,-7,T-1
):: CALL SCREEN(9):: CALL CO
LOR(1,9,9,2,9,9,11,9,9,12,9,
9,13,9,9)
1480 NEXT T
1490 CALL SCREEN(2):: CALL C
OLOR(1,2,2,2,2,2,11,2,2,12,2
,2,13,2,2)
1500 FOR T=1 TO 5 :: CALL SO
UND(-1000,1000+(200*T),T)::
NEXT T
1510 CALL SOUND(100,125,1)
1520 IF OXYGEN<1 THEN DISPLA
Y AT(8,10)ERASE ALL:"OUT OF
OXYGEN"
1530 IF ARMOR<1 THEN DISPLAY
 AT(8,10)ERASE ALL:"OUT OF A
RMOR"
1540 IF HIT5=-1 THEN DISPLAY
 AT(8,10)ERASE ALL:"ANTIMATT
ER HIT!"
1550 DISPLAY AT(12,10)SIZE(5
):"SCORE" :: DISPLAY AT(12,1
7):SCORE
1560 IF SCORE>HIGH THEN HIGH
=SCORE
1570 DISPLAY AT(14,10)SIZE(4
):"HIGH" :: DISPLAY AT(14,17
```

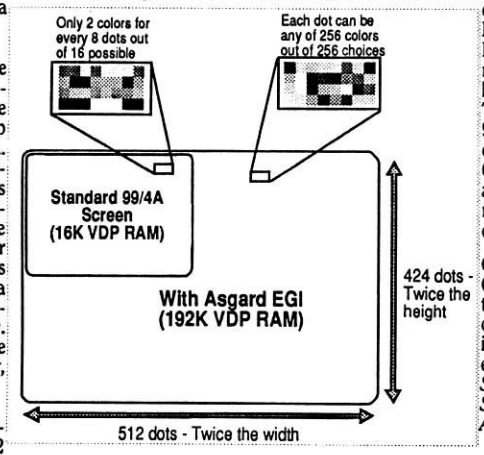
```
):HIGH
1580 DISPLAY AT(19,10):"GAME
 OVER"
1590 DISPLAY AT(21,5):"PLAY
AGAIN? Y OR N"
1600 CALL KEY(0,K,S):: IF K=
89 OR K=121 THEN CALL CLEAR
:: GOTO 350
1610 IF K=78 OR K=110 THEN 1
750 ELSE 1600
1620 LEVEL=LEVEL+1 :: PN=0 :
: IF LEVEL>5 THEN 1630 ELSE
590
1630 CALL CLEAR :: CALL SCRE
EN(2):: CALL DELSPRITE(ALL)
1640 CALL SOUND(200,262,4)::
 CALL SOUND(200,330,4):: CAL
L SOUND(200,392,4):: CALL SO
UND(100,523,3)
1650 CALL SOUND(150,523,30):
: CALL SOUND(200,392,3):: CA
LL SOUND(500,330,2,392,2,523
,2)
1660 CALL SOUND(200,131,2,16
5,2,196,2)
1670 DISPLAY AT(8,7):"CONGRA
TULATIONS !" :: DISPLAY AT(1
1,6):"YOU HAVE JUST SAVED"
1680 DISPLAY AT(12,6):"THE R
EACTOR AND THE" :: DISPLAY A
T(13,7):"ENTIRE WORLD FROM"
1690 DISPLAY AT(14,6):"COMPL
ETE DESTRUCTION."
1700 DISPLAY AT(16,6):"YOUR
MOTHER WOULD BE" :: DISPLAY
AT(17,11):"PROUD !!!" :: DIS
PLAY AT(20,3):"PLAY AGAIN?"
1710 DISPLAY AT(21,4):"PRESS
 Y OR N"
1720 DISPLAY AT(5,3):"SCORE
";SCORE
1730 CALL KEY(0,K,S):: IF K=
89 THEN CALL CLEAR :: GOTO 3
50
1740 IF K=78 THEN 1750 ELSE
1730
1750 END
1760 REM HERO MOVE SUB
1770 IF K1<>1 THEN 1800
1780 CALL JOYST(2,XR,YR):: I
F XR=0 AND YR<>0 THEN CALL M
OTION(#1,-YR*4,0):: GOTO 182
0
1790 CALL MOTION(#1,0,0):: R
ETURN
1800 CALL KEY(0,K,S):: IF K=
69 THEN CALL MOTION(#1,-16,0
):: GOTO 1820
1810 IF K=88 THEN CALL MOTIO
N(#1,16,0):: GOTO 1820 ELSE
CALL MOTION(#1,0,0):: RETURN
1820 IF HP=0 THEN CALL PATTE
RN(#1,100):: CALL SOUND(3,-3
,2):: CALL SOUND(20,-7,10)::
 HP=1 :: RETURN
1830 CALL PATTERN(#1,96):: C
ALL SOUND(3,-3,2):: CALL SOU
ND(20,-6,10):: HP=0 :: RETUR
N
1840 REM SHOOT SUB
1850 IF SH=1 THEN RETURN
1860 IF K1<>1 THEN 1880
1870 CALL JOYST(2,XR,YR):: I
F XR=0 THEN RETURN ELSE 1890
1880 CALL KEY(0,K,S):: IF K<
>76 AND K<>80 THEN RETURN
1890 CALL POSITION(#1,DR,DC)
:: Z=INT(((DR-1)/12)-.05)::
IF Z<1 OR Z>12 THEN RETURN
1900 IF K1<>1 THEN 1920
1910 IF XR=4 THEN 1930 ELSE
1940
1920 IF K=76 THEN 1940
1930 CALL SPRITE(#2,104,16,M
P(Z),122,0,16):: CALL SOUND(
100,500,2):: SH=1 :: RETURN
1940 CALL SPRITE(#2,104,16,M
P(Z),107,0,-16):: CALL SOUND
(100,500,2):: SH=1 :: RETURN
1950 REM CALL SCREEN SUB
1960 ON LEVEL GOTO 1970,1980
,1990,2000,2010
1970 RETURN
1980 CALL SCREEN(13):: RETUR
N
1990 CALL SCREEN(5):: RETURN
2000 CALL SCREEN(14):: RETUR
N
2010 CALL SCREEN(10):: RETUR
N
2020 REM DATA
2030 DATA 25,14,1,49,14,1,73
,14,1,97,14,1,121,14,1,145,1
4,1
2040 DATA 25,237,-1,49,237,-
1,73,237,-1,97,237,-1,121,23
7,-1,145,237,-1
2050 DATA 25,25,49,49,73,73,
97,97,121,121,145,145
2060 DATA 25,120,49,120,73,1
20,97,120,121,120,145,120
2070 DATA 3,1,3,19,21,1,21,1
9
2080 DATA 6,1,6,19,9,1,9,19,
12,1,12,19,15,1,15,19,18,1,1
8,19
2090 DATA 1,14,1,19,22,14,22
,19
2100 DATA 5,3,116,14,16,116,
8,16,124,17,3,124,11,3,132,2
0,16,132
2110 DATA 25,14,97,237,49,23
7,121,14,73,14,145,237
```

■

ASGARD
Extended Graphics Interface

The *Asgard EGI* is an easy-to-install "sidecar" like device for the TI-99/4A that radically expands the graphics capabilities of even a console-only TI-99/4A computer to the level of a VGA display on an IBM compatible computer, at a comparable cost.

This device features the 9938 video processor - which is 100% compatible with the video chip installed in the TI-99/4A. When installed it is "invisible" to the 99/4A unless you run a program specifically designed to take advantage of it. All other TI-99/4A software views the *Asgard EGI* as simply a standard 99/4A display - and runs flawlessly. Software designed to take advantage of it, however, can be astounding!



The *Asgard EGI* can generate a display of up to 512 dots wide by 424 dots high - a higher resolution then a standard Apple Macintosh - with up to 16 colors per dot (where every dot can be one of 16 colors). It also supports 7 other new graphics modes including a lower resolution mode that offers 256 dots by 424 dots where every dot can be one of 256 different colors! Several 80-column text modes are also offered. The *Asgard EGI* also expands the video memory of TI-99/4A from 16K to 192K, and offers such features as multi-color sprites, built-in commands for bitmap graphics, etc.

Extending the graphics capabilities would seem to be pointless unless there is software to take advantage of it. Recognizing that, the *Asgard EGI* comes with several disks full of 80-column software, including an 80-column word processor and disk manager, version of Multiplan, and terminal emulator. Additionally, the card functions with dozens of commercially available products: including Asgard Software's *PrEditor, HardMaster, Paint Pro, Spell It!, Typewriter, Page Pro Banner Maker,and* dozens of other programs in the works. Coupons offering discounts for these items are also included.

The *Asgard EGI* works out of the box with a standard composite monitor, and will also function with Analog RGB monitors. The interface is attached much like a Speech Synthesizer, and has a port for plugging in other devices (such as a Corcomp Micro-Expansion box or a PE-Box). Extensive documentation is included for both users and programmers. The device carries a standard 90-day warranty and service contracts are available. Compatible with both NTSC and PAL displays. The only requirement is a TI-99/4A console.

Originally manufactured in Germany by Mechatronics, the *Asgard EGI* is a proven device with almost 1000 installed. Shipping date expected to be Feb.1, 1991. *Suggested retail $249.95. S&H: U.S., Can. add $15.00, Airmail $20.00.*

Item No. P03

CABLES

EGI Monitor Cable: Custom monitor cables for the EGI . *Suggested retail price $24.00.*

Item No. C01

"Y" Cable: Useful for both the Asgard Mouse and MIDI Master. *Suggested retail $18.00.*

Item No. C02

Hayes Modem Cable: Attach any Hayes compatible modem to your TI-99/4A, or any PC to your 99/4A for direct transfers. *Suggested retail $18.00.*

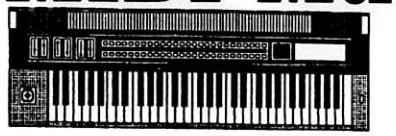
Item No. C03

Epson Printer Cable: Attach an Epson or compatible to your 99/4A. *Suggested retail $22.00.*

Item No. C04

Shipping on Cables $2.50 U.S./Canada - $5.00 Airmail

MIDI Master



For the last ten years TI, engineers and programmers have said it couldn't be done. According to the experts there was no simple, inexpensive way to interface MIDI compatible devices to a TI-99/4A or Myarc Geneve 9640. However, Mike Maksimik, armed with enough knowledge to figure it out but perhaps not enough to know it couldn't be done has produced *MIDI Master,* a complete MIDI interface with software for the TI-99/4A and the Myarc Geneve 9640.

Functionally identical to MIDI interfaces for IBM PC's, Atari ST's and Mac's, *MIDI Master* provides most of the functionality for as little as 10% of the cost of MIDI systems based on other computers. While *MIDI Master* is really only the beginning of a whole line of MIDI products for all levels of users, it provides enough capabilities to allow even the professional musician full access to the wide variety of MIDI keyboards, synthesizers, and other equipment available - yet it is simple enough where even the novice musician can readily use it. *MIDI Master* is a must for all levels of musical interest - from novices and students to the professional performer. It is the only device that allows the musically inclined to talk to a computer in a language you understand.

MIDI Master allows you to control up to 16 channels on multiple MIDI devices. The basic software included allows you to do "sequencing" (recording music that you play on a keyboard, for instance); "patching" (changing and storing the internal settings of a synthesizer); to type in music from sheet music, compile it and play it on your MIDI devices; and even to convert music stored in other MIDI formats. The software in this package can be used effectively and easily by musicians, composers, students and even performers. *MIDI Master* also provides a toolbox of routines for future add-on programs by Asgard and other publishers - sheet music printers, graphical note editors, multi-track recorders and such. Additional utilities will be produced for *MIDI Master* as our customers let us know what they are doing with it.

MIDI Master is distributed in two versions: a TI-99/4A version on cartridge with the interface and several disks of example songs and utilities; and an M-DOS version for the Geneve that takes advantage of the 9640's memory capabilities, and also includes examples. Both versions include an extensive manual detailing the MIDI language, the *MIDI Master* software, and outline of the MIDI standard, and so on. Requires a TI-99/4A or a Geneve with a disk system and an RS232 card. Cassette version for 4A users available shortly. *Suggested retail $44.95.* **Item No. P01**

Mastercard ☐ Visa ☐
Check/Money Order ☐

Credit cards add 7%

Credit Card Number:

Expiration Date:_____

Shipping & Handling: Unless stated-
Software: add $2.00 U.S./Can., $3.00 Airmail
Hardware: add $5.00 U.S./Can., $7.00 Airmail

Asgard Peripherals
P.O. Box 10697
Rockville, MD 20849
(703)255-3085

V9938 Technical Data Book

The essential reference for anyone trying to write programs for a Geneve 9640 or a 80-column card! The V9938 Technical Data Book is produced by Yamaha, and is an extensive text on this exciting processor. Not for the beginner. 152 Pages, Soft-bound.

T04 - Text - *$34.95 + $2.00 S&H*

Rock Runner

This arcade-quality game by Eric Lafortune combines fascinating graphics, excellent game-play, wonderful music and sound effects to be perhaps the game of the year for TI-99/4A arcade game fans. Utilizing a new graphics mode found in all 99/4A's, Rock Runner displays graphics that put a Nintendo to shame and is as exciting as anything available for any other computer. Each of the 16 screens requires a different strategy - each of the more then a dozen colorful monsters will make it as difficult for you as possible. Requires a TI-99/4A computer, 32K, a joystick, a disk system and an Editor/Assembler module. Not compatible with the Myarc Geneve 9640.

E05 - Disk - *$12.95*