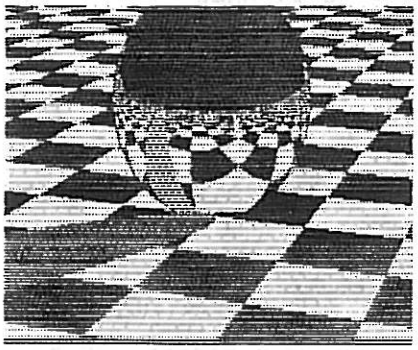
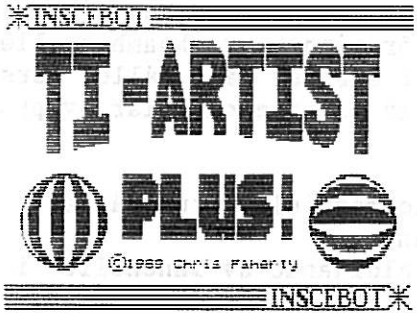
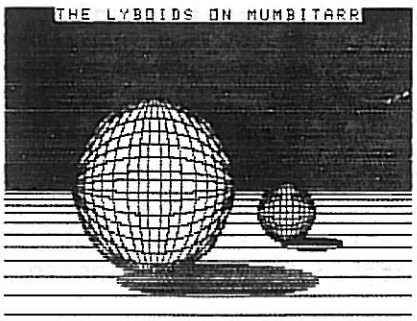
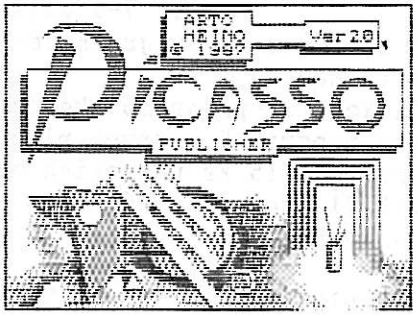
PROGRAM BITEN 89-4









INNEHALL

ISSN 0281-1146

REDAKTÖREN

Det råder fortfarande stor brist på skriftliga bidrag till tidningen från dig som medlem. Du behöver inte skriva långt och komplicerat. Även korta notiser om små nyheter eller eller programmeringstrix är välkomna. Sedan jag övertog uppdraget som redaktör i juni 1989 har det endast kommit in fem nya bidrag (inkl. översättning) som resulterade i sammanlagt 11 sidor i tidningen. Bidragen har kommit från Mikael Nordlin, Åke Jönsson, Claes Schibler och Börje Häll men vad har alla ni andra hållit på med under sista halvåret. Detta är på tok för litet eftersom tidningen har 24 sidor per nummer. Sidan 1 och 2 kan räknas bort men det återstår 4x22 = 88 sidor att fylla. Resten har fyllts med äldre material som tidigare kommit från medlemmar eller plockats från skivor som kommit från utlandet. I övrigt har jag själv personligen skrivit eller översatt resten. Detta är något som jag inte kan göra i fortsättningen. Som mest kanske jag kan skriva egna sidor och plocka från utländska skivor så att det täcker 10 sidor per nummer. Resterande 12 sidor per nummer måste fyllas av dig som medlem. Du kan skriva själv eller översätta från utländska skivor eller User Group-tidningar. Föreningen måste av skattetekniska skäl ge ut minst fyra nummer per år så detta är vad medlemmarna minst kan vänta sig. Vår ekonomi tillåter troligen att vi ger ut sex nummer per år, men då måste det finnas något att fylla dessa med. Har du synpunkter på föreningens framtid så hör gärna av dig per telefon eller brev. Tag även kontakt med valberedningen Kent Edgardh och Sten Gunnarsson när det gäller förslag om funktionärer för nästa år. Du kanske själv kan tänka dig att vara med i styrelsen.

John Guion, 22 år, dog i en bilolycka i september. Han var känd för Multi-Mod och modifierade PROM till TI RS232 och Kontrollkort. Han har även konstruerat P-GRAM Card, förbättrat Horizon RAM-disk och ombyggnad av konsolen med 32 KB exp-minne.

Amnion Helplines ägare Guy-Stefan Romano, 57 år, dog i augusti.

Redaktör:
Jan Alexandersson

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

Postgiro 19 83 00-6
Medlemsavgiften för 1989 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 m.m.

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 magazine nr 12/82, 1-5, 7-9/83(st)	40:-
Gamla årgångar av Programbiten	50:-
TI-Forth manual	100:-
Hel diskett ur programbanken(st)	30:-

Enstaka program 5:- styck plus startkostnad 15 kr per skiva eller kassett (1 program = 20 kr, 3 program = 30 kr).

SÄND IN DINA PROGRAM OCH ARTIKLAR.

SEKTOR-EDITOR FÖR SKIVOR

av Jan Alexandersson

Det finns ett flertal program till TI-99/4A med flexskiva som ger dig möjlighet att direkt läsa och skriva en sektor oberoende av filerna. Följande är de vanligaste:

- Disk Patch (följer med Funnelweb)
- Disk Utilities av John Birdwell
- Disk Aid av Col Christensen
- MG Advanced Diagnostics
- Sector One av Randy Moore
- Hard Master från Asgard

De tre första kan beställas från Programbanken. Sector One kan fås från: Randall W. Moore, 3931 Navaho Dr., N.Highlands, CA 95660, USA för USD 10 + porto USD 3. Ange om du vill ha 40-(99/4A) eller 80-(9938) kolumnsversionen. Hard Master finns hos Asgard för USD 15 + porto USD 5.

En flexskiva består normalt av 40 spår per sida och har 9 eller 18 sektorer per spår. En sektoreditor hoppar över alla skrivskydd för filer så du måste veta vad du gör. I annat fall kan hela skivan förstöras. Den viktigaste användningen är att ändra till svenska bokstäver, ändra RS232 till PIO och ändra filnamn DSK1.UTIL1 till DSK1.FW o.dyl.

En sektoreditor kan även användas för att reparera skadade skivor där t.ex. sektor 0 eller 1 är förstörda vilket hindrar åtkomsten av filerna trots att de kanske finns kvar oskadade. Ett sätt kan vara att kopiera sektor 0 och 1 från en oanvänd skiva och sedan använda Recover File i DM 1000. Enklast är att byta skiva mellan läs och skriv av sektorn.

	DISK PATCH	DISK UTIL	SECTOR ONE	HARD MASTER
SEKTOR				
Läsa	JA	JA	JA	JA
Editera	JA	JA	JA	JA
ASCII/HEX	JA	JA	JA	JA
Stega	JA	JA	(JA)-	JA
Skriva	JA	JA	(JA)-	JA
Printer	-	JA	JA	JA
Print DSK	-	JA	JA	JA
Jämföra	-	-	JA	-
Kopiera	-	(JA)-	JA	(JA)-
Söka Text	-	JA	JA	JA
Sekt0/Mark	-	JA	-	-
Sekt0/Free	-	JA	-	-
FIL				
Sökning	JA	JA	-	JA
DISK				
DSK-katalog	JA	JA	-	JA
directory	-	-	-	JA
WDS-katalog	-	-	-	JA
FORTH				
visa scrn	-	-	-	JA
print scrn	-	-	-	JA
MEDIUM				
flexskiva	JA	JA	JA	JA
hårddisk	-	-	JA	JA
DISPLAY				
40 kolumn	JA	JA	JA	JA
80 kolumn	-	-	JA	JA

DISK-PATCH har även följande originaltangenter:

FCTN 1	Hex
FCTN 2	ASCII
FCTN 4	Nästa sektor
FCTN 5	Hoppa till annan sektor
FCTN 6	Föregående sektor
FCTN 8	Skriva sektor
FCTN 9	Meny

MG Advanced Diagnostics är det enda program som kan läsa informationen mellan sektorerna. En skiva på 90 kB har egentligen 125 kB eftersom mycket används för adress, synkronisering och checksumma (CRC).

Disk Utilities kan endast kopiera mellan två skivor om samma sektornummer önskas.

Sector One kan ej stega till andra sektorer direkt i editorn utan man måste hoppa till menyn och använda READ. Sector One skriver en sektor utan att fråga en gång till om du verkligen vill skriva. Detta är en potentiell risk för fel eftersom minnesbuffert och sektornummer ej automatisk hänger ihop. Varning för detta eftersom BACK och FORWARD ej läser utan endast ändrar sektornummer och inte minnesbufferten. DS/QD 720 kB-skivor visar felaktig Map Sector trots att sektoreditorn i övrigt fungerar. Sector Used visas felaktigt (den sista saknas ibland) trots att antalet sektorer visas

riktigt. Tecknet = har kommit fel på 80-kolumnsversionen. Manualen är på 6 sidor och är ganska slarvigt skriven men innehåller trots detta läsvärd information.

Hard Master kommer med två versioner på samma flexskiva, version 2.14 (40 kolumner) och version 2.18 (80 kolumner). COPY av sektorer kan endast ske inom samma enhet, så WDS1 till DSK1 går ej. Den visar map över använda sektorer både för flexskiva och hårddisk. Hard Master kan läsa och visa FORTH-skärmar men ej editera dessa. Det finns även möjlighet att skriva ut dessa mot printer eller disk så att endast normala ASCII-tecken visas (övriga blir blanktecken). Detta kan vara mycket användbart för en redaktör som vill ta in Companion- eller Pascal-texter. Alla kontrolltecken tvättas automatiskt bort. Manualen på 24 sidor är mycket välskriven med mycket läsvärt utöver handhavandet av programmets kommandon. Författaren har troligen missuppfattat sektor >20 - >3F på hårddisk. Jag tror att dessa är avsedda att visa felaktiga sektorer så låt bli att göra back-up med CO 0 20 20 enligt sidan 8 i manualen. Hard Master visar alltid ett jämt antal datasektorer även om den sista ej används. Den är ju upptagen eftersom en allocation unit = 2 sektorer för en 20 MB hårddisk.

Sector One och Hard Master är de enda som även fungerar med en hårddisk och detta är naturligtvis den viktigaste användningen för dessa båda. Av dessa två är Hard Master den bästa. Det är inget som hindrar att du använder dessa även om du inte har hårddisk.

```
***  Sector One/40 v1.1  ***

Diskname.....FW*4*13
Number of Sectors...0720
Sectors/Track.......09
Initialization code.DSK
Disk protection.....No
Tracks/Side.........40
Sides...............Double
Density.............Single
Sectors used........0705
Sectors free........0015


Diskname.....HARDDISK
Number of Sectors..>013380
Write Precomp Track>001D
Bufferd Head Step...No
Number of Heads.....4
Sectors/Track.......32
Number of Files.....019
Number of Subdirs...017
Root Dir Map Sect..>0040
Sub Dir Map Sect...>0000DA


Filename.............LOAD
Directory Sector.....>001D
File Type............Program
Protected............No
Record Length........000
Records/Sector.......000
Total Records........0000
Sector Offset........>ED
Number of Data Sectors.0030
Sectors Used.........22E-24B


Filename.............MICRO/D
Directory Sector.....>0005
File Type............Int/Fix
Protected............No
Record Length........085
Records/Sector.......003
Total Records........0221
Sector Offset........>00
Number of Data Sectors.0074
Sectors Used.........02C-073
                     0C1-0C2


Filename.............FWDOC/REPT
Directory Sector.....>0016
File Type............Dis/Var
Protected............No
Record Length........080
Records/Sector.......003
Total Records........0054
Sector Offset........>1C
Number of Data Sectors.0054
Sectors Used.........17D-1B2
```

INTERFACE STANDARD

The Interface Standard and Design Guide for TI99/4A Peripherals finns nu färdig. Den består av 100 sidor samt en dubbelsidig flexskiva (eller 2 enkelsidiga). Priset är USD 22 för DS/SD och USD 23 för SS/SD. Räkna med att extra porto tillkommer. USD 8 kanske kan vara lagom. Adressen är: Tony Lewis, 409 Drolmond, Raleigh, NC 27615, USA.

```
WDS1 HARDDISK   Sze>13380  Sec>0                    (HARD MASTER)
Addr 0 1  2 3  4 5  6 7  8 9  A B  C D  E F
 00  4841 5244 4449 534B 2020 99C0 2020 013A HARDDISK  ..  .:
 10  131D 8A4A B2AE 1911 0020 0000 006D 0021 ...J..... ...m.!
 20  0023 0073 0075 0077 0079 007B 0025 007D .#.s.u.w.y.{.%.}
 30  007F 0092 009C 013A 009E 00A0 00A8 0000 .....:........
 40  0000 0000 0000 0000 0000 0000 0000 0000 ................
 50  0000 0000 0000 0000 0000 0000 0000 0000 ................
 60  0000 0000 0000 0000 0000 0000 0000 0000 ................
 70  0000 0000 0000 0000 0000 0000 0000 0000 ................
 80  0000 0000 0000 0000 0000 0000 0000 0000 ................
 90  0000 0000 0000 0000 0000 0000 0000 0000 ................
 A0  0000 0000 0000 0000 0000 0000 0000 0000 ................
 B0  0000 0000 0000 0000 0000 0000 0000 0000 ................
 C0  0000 0000 0000 0000 0000 0000 0000 0000 ................
 D0  0000 0000 0000 0000 0000 0000 0000 0000 ................
 E0  0000 0000 0000 0000 0000 0000 0000 0000 ................
 F0  0000 0000 0000 0000 0000 0000 0000 0000 ................
```

ROOT (HARD MASTER)

Filename	Sze	Typ	Len	FdrNo	Clusters	
ALEX-FORTH	16A	PGM		100	1764-18CB	
CAT/DEL	8	PGM		102	18CC-18D1	
CAT/MYARC	7	PGM		104	18D2-18D7	
HM	22	PGM		368	3376-3395	
HM80	22	PGM		36A	3396-33B5	
HM81	11	PGM		36C	33B6-33C5	
HN	10	PGM		36E	33C6-33D3	
LOAD	20	PGM		116	1902-191F	
MICRO/D	4C	I/F	85	360	32DA-32DD	32E4-3329
MICRO/S	5	I/F	255	30A	2F42-2F45	
SECTOR140	22	PGM		318	2FE2-3001	
SECTOR141	12	PGM		31A	3002-3011	
SECTORONE	22	PGM		370	33D4-33F3	
SECTORONF	14	PGM		372	33F4-3405	
TIME	3	PGM		120	192A-192B	

Diskname=HARDMASTER Free=105 Used=63

Filename	Sze	Typ	Len	FdrNo	Clusters
HM	21	PGM		2	22-41
HM80	21	PGM		3	42-61
HM81	10	PGM		4	62-70
HN	F	PGM		5	71-7E

```
  Drv1 HARDMASTER Sze>168    Sec>0
DISK-SECTOR-BIT-MAP                    X = Used   - = Free
           SEC 00  02  04  06  08  0A  0C  0E  10  12  14  16  18  1A  1C  1E
           TOR +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+--
38=3F000000 000|X X X X X X - -|- - - - - - - -|- - - - - - - -|- - - - - - - -
3C=FCFFFFFF 020|- - X X X X X X|X X X X X X X X|X X X X X X X X|X X X X X X X X
40=FFFFFFFF 040|X X X X X X X X|X X X X X X X X|X X X X X X X X|X X X X X X X X
44=FFFFFF7F 060|X X X X X X X X|X X X X X X X X|X X X X X X X X|X X X X X X X -
48=00000000 080|- - - - - - - -|- - - - - - - -|- - - - - - - -|- - - - - - - -
4C=00000000 0A0|- - - - - - - -|- - - - - - - -|- - - - - - - -|- - - - - - - -
50=00000000 0C0|- - - - - - - -|- - - - - - - -|- - - - - - - -|- - - - - - - -
54=00000000 0E0|- - - - - - - -|- - - - - - - -|- - - - - - - -|- - - - - - - -
58=00000000 100|- - - - - - - -|- - - - - - - -|- - - - - - - -|- - - - - - - -
5C=00000000 120|- - - - - - - -|- - - - - - - -|- - - - - - - -|- - - - - - - -
60=00000000 140|- - - - - - - -|- - - - - - - -|- - - - - - - -|- - - - - - - -
64=00FFFFFF 160|- - - - - - - -|X X X X X X X X|X X X X X X X X|X X X X X X X X
```

TI-BASE 2.02 AUG 1989

av Jan Alexandersson

NY VERSION

TI-Base som är ett databasprogram finns nu i version 2.02 från aug 1989 och kan beställas från Texaments, 53 Center Street, Patchogue, NY 11772, USA för USD 24.95 + porto USD 8. Ägare av TI-Base 2.0 kan sända in programskivan och porto USD 8 men du som har TI-Base 1.0 måste sända in både program- och tutorialskivorna samt USD 8 + USD 8 porto.

De viktigaste nyheterna är:

- Kan nu köras både från hårddisk och RAM-disk vilket ej går utan problem med tidigare versioner. Naturligtvis fungerar även flexskiveenheter. Program-, kommando- och datafiler kan användas på alla dessa lagringsmedia.

- Sortering med lika fält fungerar riktigt (fr.o.m. version 2.01).

- Variabler av X-typ med efterföljande nollor fungerar.

- Editering av fält med bredden 25 fungerar riktigt så att extra tecken ej kan skrivas.

- SUM med ;FOR fungerar.

- PRINT och DISPLAY med ;FOR fungerar riktigt med temporära lokala variabler.

- CATALOG fungerar även med flexskiva utan namn.

- REPLACE med ;FOR fungerar med sortering och literals.

LADDNING

TI-Base kan nu laddas från hårddisk med WDS1.TIB.TIBASEW (16 tecken). I Funnelweb User List måste du krympa till 15 tecken. Denna "path" kan ändras godtyckligt men högst 20 tecken får användas. Jag kopierade filen TIBASEW till WDS1.DSK1 och döpte om den till DB så att Funnelweb kan ladda den som DSK1.DB från den centrala menyskärmen. Hela TI-Base lagrade jag dock på WDS1.TIB.

Det finns även laddningsprogram från flexskiva för DIS/FIX 80, PROGRAM och Extended Basic.

KVARSTAENDE PROBLEM

Om markören sätts på sista tecknet i input-fältet och man sedan gör delete av ett tecken och Execute för att lagra detta så kommer första bokstaven i nästa fält att försvinna.

Sortering kan ej göras på fältnamn med längden 10 utan max 9 gäller.

SVENSKA BOKSTÄVER

Kopiera filen SCRN till en ny tom skiva. Använd en sektoreditor som bl.a. finns i Funnelweb och ändra enligt följande (OBS olika placering beroende på version):

VERS	2.01	2.02		
SEKT	BYTE	BYTE	HEX-STRÄNG	ASC
>27	>4D	>75	08107C4078407C00	64
>28	>26	>4E	280038447C444400	91
>28	>2E	>56	28007C4444447C00	92
>28	>36	>5E	382838447C444400	93
>28	>3E	>66	2800444444443800	94
>28	>4E	>76	081038447C403800	96
>29	>27	>4F	2800384848483400	123
>29	>2F	>57	2800384444443800	124
>29	>37	>5F	1000384848483400	125
>29	>3F	>67	2800484848483400	126

MICRODEX I OCH II

Microdex innehåller datafiler för TI-Base. Microdex I har innehållsregister till följande tidningar:

- 99'er/Home Computer Magazine
- Compute! 1983-1986
- Micropendium 1984-1988 (2 skivor)

Microdex II har detta för följande:

- Computer Shopper 1985-1988
- Enthusiast 99

- Super 99 Monthly
- The Smart Programmer
- Byte
- Creative Computing
- Family Computing
- Mini-Mag 99
- Popular Computing
- R/D Computing

Följande filstruktur har använts:

```
FIELD DESCRIPTOR TYPE WIDTH

  1   SUBJECT     C    29
  2   SOURCE      C    22
  3   TYPE        C    19
  4   DATE        C     6
  5   PAGE        C     4
```

Microdex I består av fyra skivor och Microdex II har två skivor. Dessa säljs av Texaments för USD 14.95 respektive USD 9.95. Varje skiva har en stor datafil och ett antal hjälpfiler (kommandofiler) för utskrift och sökning m.m.

Alla hjälpfiler till Microdex I är lika utom MENU/C och REPORT/C . Du kan dock använda samma MENU-fil om du skriver USE DSKx.BASE innan du skriver DO MENU. Du kan även editera MENU så att texten blir generell. Du kan använda samma REPORT-fil om du ersätter "Micropendium 1984-88-A-L" med:

```
LOCAL HEAD C 20
READSTRING 23,1 HEAD
```

Du kan sedan göra PRINT av HEAD var du vill.

Microdex II är annorlunda eftersom kommandofilerna har skapats med TI-BASE och inte TI-Writer som när det gäller Microdex I. TI-Writer ger DIS/VAR 80-filer med variabel postlängd medan TI-BASE ger DIS/VAR 80-filer med postlängden 40 (!!!) oberoende av innehåll. Detta betyder att TI-BASE filer behöver mera skivutrymme. Om du laddar en sådan fil till TI-Writer och sedan sparar den med PF (Print File) till en skiva så har du sparat många sektorer. Microdex II hjälpfiler är något annorlunda jämfört med Microdex I eftersom utskriften görs i annan ordning. SNAP och tangent "S" finns i MENU1-filen på alla Microdex-skivor trots att någon SNAP/C-fil inte finns. Microdex är inte så bra strukturerad som man skulle kunna önska sig. Det går inte att sortera på DATE som JAN86. Det är bättre att använda 86-01.24 för år-månad.sida. Du bör alltid se till att det viktigaste kommer först i varje fält eftersom det går bra att söka på avkortad text.

FIND och PRINT i Microdex meny kan endast finna identiska fält. Om du ändrar till .OR. istället för .AND. så fungerar det bättre. Du kan även använda FOR efter PRINT ALL så att även avkortad text kan sökas.

EXEMPEL KOMMANDOFIL

```
* Microdex II utskrift
SET PAGE 62
SET HEADING ON
SET RECNUM ON
LOCAL LEFT X 20
REPLACE LEFT WITH "0A0A0A1B671B6C12"
CLOSE ALL
USE DSK2.AL
PRINT LEFT
PRINT ALL ;FOR SOURCE="R/D"
CLOSE
SET HEADING OFF
USE DSK2.MZ
PRINT ALL ;FOR SOURCE="R/D"
CLOSE
SET HEADING ON
RETURN
```

```
* Egen utskrift Micropendium
SET PAGE 62
SET HEADING ON
SET RECNUM OFF
LOCAL LEFT X 20
REPLACE LEFT WITH "0A0A0A1B501B6C0A"
CLOSE ALL
USE DSK2.MICRO
PRINT LEFT
PRINT ALL MANUFACT NAME MODULE ;
TYPE PAGE ;FOR TYPE="GAME"
CLOSE
RETURN
```

REFERENSER

PB 88-4: TI-BASE 2.0
PB 89-1: Programbiten, register

Micropendium:
jun 88: TI-BASE 1.0 prel. look
nov 88: TI-BASE 2.0 review
dec 88: TI-BASE 2.0 newsbytes
maj 89: TI-BASE 2.0 tutorial

VIDEOPROCESSOR FÖR 99/4A

av Jan Alexandersson

1.Minnesuppbyggnad

TI-99/4A har en centralprocessor (CPU) 9900 som är en 16-bitars processor som kan adressera 64 kbytes. För att kunna hantera all data finns en databuss, en adressbuss och en kontrollbuss. Databussen behöver 16 ledare vilket betyder att två bytes kan överföras samtidigt. Adressbussen har 15 ledare för att klara 64 kbytes beroende på att två bytes alltid tas samtidigt.

Minnestabell för CPU:

```
>0000->1FFF Operativsystem ROM 8 kB
>2000->3FFF Låga delen(8 kB) av EM
>4000->5FFF Periferienheter 8 kB
>6000->7FFF Modul-ROM 8 kB
>8300->83FF PAD-RAM 256 bytes
            SOUND   >8400
            VDP     >8800, >8802,
                    >8C00, >8C02
            SPEECH  >9000, >9400
            GROM    >9800, >9802,
                    >9C00, >9C02
>A000->FFFF Höga delen(24 kB) av EM
```

Inom området >8000 ->9FFF är adresserna ofullständigt avkodade vilket betyder att flera olika adresser uppfattas som samma sak av datorn.

2.Bankswitchning

För att komma åt ytterligare minne används ofta bankswitchning som kopplar in flera olika minnesbankar på samma adress. Detta används för alla tilläggskort som aktiveras av sin egen CRU-adress på kontrollbussen. Upp till 16 olika kort kan adresseras på CPU-adress >4000->5FFF.

Modul-ROM för Extended Basic switchar 4 kB i två bankar samt en fast 4 kB. Det betyder att 12 kB används där. Spelmoduler från ATARI switchar två kompletta 8 kbytes-bankar. I det fallet finns totalt 16 kB. Super Space använder RAM på denna adress som kan bestå av upp till 4 bankar med vardera 8 kbytes. Det finns en sådan modul med 32 kbytes.

3.Memory mapping

Ett annat sätt att adressera mycket minne är genom memory mapping som används för att komma åt ljud, tal, bildskärm(VDP) och ett speciellt GROM-minne. Minnet kommer datorn åt genom speciella in- och utportar som finns inom minnesutrymmet >8000->9FFF. Normalt finns en adressport och en dataport.

GROM-minnet har följande uppdelning:

```
>0000->17FF Operativsystem 6 kB GROM
>2000->57FF TI-BASIC 12 kB GROM
>6000->FFFF Modul-GROM 30 kB
```

VDP-minnet för BASIC 16 kB:

```
>0000->02FF Skärmen
>0300->031F Färgtabellen
>0320->036F Crunch buffer
>0370->07FF Teckendefinitioner m.m.
>0800->35D7 BASIC-program m.m.
>35D8->3FFF Disk buffert
```

4.VDP-register

I 99:an används en 9929A som videoprocessor (9918A endast för NTSC). Den har 8 st 8-bitars register för att skriva till och ett statusregister att läsa från. De 8 bitarna är numrerade 0-7 där 0 har den största tyngden.

VDP-register 0:
Bit 0-5 skall alltid vara 0.
Bit6 = Grafikmode2 (bitmap).
Bit7 = Extern video (ej 99/4A).

VDP-register 1:
Bit0 = 16kB RAM. Bit1 = Aktivera skärm. Bit2 = Interrupt. Bit3 = Textmode1. Bit4 = Multicolor mode. Bit5 = 0. Bit6 = Sprite storlek. Bit7 = Sprite magnify.

VDP-register 2:
Värdet(av bit 4-7) gånger hex >400 visar början av Skärmtabellen.

VDP-register 3:
Värdet(av bit 0-7) gånger hex >40 visar Färgtabellen.

KRYMPA ASSEMBLER, DEL 3

av Tony McGovern, Australien

Block som skall flyttas har inte alltid känd längd. En datatyp som finns i många språk är sträng, som är en samling tecken tillsammans med en längdindikator. Den mest bekanta formen är den i TI-Basic där en längdbyte som kommer före teckensträngen anger längden, upp till max 255. Andra strängkonventioner används också i olika datorspråk, men låt oss titta på just denna ett tag.

Ditt assemblerprogram skall flytta en sträng från en känd adress STR1 till en annan STR2. Två saker är klara från början. För det första måste antalet bytes som skall flyttas läsas från den första strängen (om den redan var känd så skulle en enkel blockflyttning duga) och för det andra måste byteflyttning användas eftersom byte är den enhet som bygger upp strängar.

Ett närliggande sätt att koda med hänsyn till detta är

```
        LI   R0,STR1
        LI   R1,STR2
        MOVB *R0,R2
        SRL  R2,8
        INC  R2
LOOP    MOVB *R0+,*R1+
        DEC  R2
        JGT  LOOP
```

Längdbyten läses till ett register, omvandlas till nedräkningsord (ord = 2 bytes), ökat med ett för att även täcka själva längdbyten. Denna kod tar 10 ord och använder 3 register. Detta är ganska påkostat så låt oss göra det kortare.

```
        MOVB @STR1,R1
        SRL  R1,8
        INC  R1
LOOP    MOVB @STR1-1(R1),@STR2-1(R1)
        DEC  R1
        JGT  LOOP
```

Detta använder samma indexerade adressering i loopen som vi gjorde för blockflyttningen tidigare. Det tar nu 9 ord och använder endast 1 register. Det är möjligt att korta ned det med 1 ord till genom att använda en speciell egenskap hos DEC-instruktionen. DEC sätter alltid CARRY STATUS-bit utom när den ändras från noll till minus ett (se E/A-manualen sid 100).

```
        MOVB @STR1,R1
        SRL  R1,8
LOOP    MOVB @STR1(R1),@STR2(R1)
        DEC  R1
        JOC  LOOP
```

Loopen genomlöpes nu en extra gång när R1 är noll, så INC-instruktionen är inte längre nödvändig och den indexerade adress-offseten blir också ändrad. Så jobbet kan göras med en kod på 8 ord (16 bytes) och ett register.

VDP-register 4:
Värdet(av bit 5-7) gånger hex >800 visar Teckendefinitionstabellen.

VDP-register 5:
Värdet(av bit 1-7) gånger hex >80 visar Sprite Attribute-tabellen.

VDP-register 6:
Värdet(av bit 5-7) gånger hex >800 visar Sprite Teckentabellen.

VDP-register 7:
Bit 0-3 = Förgrundsfärg.
Bit 4-7 = Bakgrundsfärg.

De 8 registren har följande värden:

```
 0  1  2  3  4  5  6  7
00 E0 F0 0C F8 86 F8 07 för BASIC
00 E0 00 20 00 06 00 07 för XB
00 E0 00 0E 01 06 00 F5 för EA
02 E2 06 FF 03 36 03 00 för Parsec
```

ZENO BOARD

är nytt kort för anslutning till TI-99/4A utan andra tillbehör. På detta kort monterar du: 32 kB expansionsminne, klocka med batteri-backup, Speech, Extended Basic, tre extra GROM, Reset switch. Kraft tas från datorn. Priset är USD 17.50 för tomt kort + USD 1 för manual + porto USD 8. Adress: Eric Zeno, 414 Highland RD., Pittsburgh, PA 15235, USA.

XHI FÖR 9938 & DIJIT AVPC

av Jan Alexandersson

XHI VERSION 3.6 FÖR 9938

XHI finns nu i version 3.6 från 29 sep 1989. Du kan få den för fairware donation + 7 DM (disk+porto) från: Alexander Hulpke, Sadowastrasse 68, D-5600 Wuppertal 1, Västtyskland. Nödvändigt: Geneve 9640 eller TI-99/4A med 32kbytes EM och 9938 VDP, färgmonitor, flexskiveenhet. HARDCOPY kan dock även användas utan 9938.

Skivans innehåll:

*README	Denna fil
ARC1	För uppackning av DOC-fil
XHI	Programmet XHi
XHI/T	Källkod
XHIDOX	Arkivpackade manualer
XHIDOC	Engelsk manual (i XHIDOX)
XHIDEU	Tysk manual (i XHIDOX)
COLDEF	Definiera om färger
LOAD	XHI-laddare med SysTex
CHARA1	Teckenset för HARDCOPY
HCSETUP	Printer Setup för HARDCOPY
HCLOAD	XB-laddare för HARDCOPY
YLOAD	XBASIC EA5 Laddare
HARDCOPY	Version 1.1, EA5
HIRESDEMO	Demo-program med grafik
UHR	Demo, analog klocka
FIXSTERNE	Demo, stjärnbilder
KUGEL	Enkelt Ray-tracing program
KUGEL;PAR	Exempel på startvärden
KUGEL;PIC	Färdig bild

NYHETER TILL XHI

FILL är ett nytt kommando som fyller hela ytan runt en punkt med färg.

LINE är ändrad så att den även ritar sista punkten.

MOTION är ändrad så att den även fungerar med DIJIT AVPC. Du måste dock alltid se till att den första rad du knappar in är REM med minst 16 tecken (med AVPC). Jag använder:
100 !123456789ABCDEF
Raden måste ligga högst upp i CPU-RAM vilket säkrats fås om man:
SAVE DSK1.MERGE,MERGE, NEW,
MERGE DSK1.MERGE

HARDCOPY v.1.1 från 19 sep 1989 finns som ett fristående program.

X80 för Text2 finns nu i version 0.91 från 11 aug 1989 med engelsk manual (ingår ej i XHI-skivan).

DIJIT AVPC

Jag har nu fått ett tillägg till manualen från DIJIT:
DIJIT SYSTEMS AVPC TECH NOTE 1, by David Allen, march 3,1989, Rev.A. Denna beskriver EPROM version 1.00 med 2 kbytes. Denna har Powerup-rutin, Interrupt-rutin och plats för framtida DSR-rutin.

Powerup-rutinen initierar VDP-register där TI felaktigt har satt reserverade bitar som saknar betydelse för 9918A/9929A men som har stor betydelse för 9938. Videoprocessorn stegar alltid fram adressen efter varje läsning av data vilket gör att när 9938 läser >3FFF så görs en automatisk bankswitchning för att koppla in nästa VDP-bank. De ursprungliga fyra grafikmoderna som även finns i 99/4A arbetar utan denna bankswitchning även med 9938, men Text2 och Graphic3-7 har bankswitchning som gör att diskbuffertarna ej kan läsas med TI- eller Corcomp-kontrollkort (Myarc använder aldrig VDP). Lösningen på detta är att powerup måste sätta högsta adress LFAVDP (PAD >8370) till ett lägre värde så att diskbufferten ej når >3FFF.

Interruptrutinen (ISR) har en egen interrupt-manager för att hålla reda på alla interna och externa interruptrutiner. När ett ERROR inträffar i datorns grund-BASIC kommer 99/4A att felaktigt återställa VDP-REG 2-6. Detta korrigeras av ISR-rutinen men du kommer under en bråkdel av en sekund se färgade band över skärmen vid ERROR. Extended BASIC har inga sådana problem eftersom TI för en gångs skull lyckades göra rätt med VDP-register i XB. När en extern interrupt-rutin startas kommer AVPC att lägga ut sin kod i

MIAMI BOOT OCH MENU

av Jan Alexandersson

Det finns en helt ny version av Miami Boot från feb 1989. Den finns dels som BOOT för Extended Basic med 32 kB EM och dels som MENU för Horizon RAM-disk. Båda dessa versioner finns nu i programbanken men på olika skivor. Adress: The Miami Users Group, 6755 Tamiami Canal Road, Miami Florida 33126, USA.

Miami Boot är ett menyprogram som du kan lagra på varje skiva med Extended Basic-program (PROGRAM och INT/VAR 254) och Assemblerprogram (PROGRAM). Du kan själv lägga in dina program i menyn och ange vilken fil som skall laddas. Menyn har nu tre olika skärmar som du stegar mellan med mellanslag. Detta gör att du kan lägga in 24 olika program i samma meny. Följande kommandon finns:

1	Katalog
2	Titta på fil
3	Starta XB- eller AL-program
C	Startar GROM-modul
D	Suddar fil från skiva
G	Stegar mellan olika GROM
I	Ändrar XB-skärmfärgen
K	CALL
P	Printer-namn
S	ROM-modul >6000
T	Track Disk
V	Version
X	Startar XB(INT/VAR 254)
SPACE	Stegar mellan tre menyer
FCTN5	Editerar din egen meny

Du kan skriva ut både katalog och filer till din skrivare genom att trycka på SHIFT1 respektive SHIFT2.

CPU-RAM från >FFD8. Detta är samma plats som Extended Basic har sina programrader. Detta är förklaringen till varför XHI med MOTION förstör första XB-programraden. Enligt ett brev från Alexander Hulpke så ska det finnas en ny version av EPROM som tar bort detta problem. Jag har ännu ej tagit kontakt med DIJIT om detta eftersom det inte är något stort problem. I ett tidigare brev berättar Thomas Spillane att en ny EPROM som håller på att tas fram kommer att fungera med TELCO och en ny p-Code editor.

Bengt Jönsson i Trelleborg har problem med att använda DIJIT AVPC med Pascal P-Code-kort tillsammans med Corcomps diskkontrollkort. Med TI-kontrollkort och Pascal så fungerar allt liksom AVPC med Corcomp utan Pascal.

PagePro från Asgard funger endast med AVPC om du först kör SET99/4A.

VDP-ANROP MED 9938

>8800 VDPRD VDP Read Data
>8802 VDPSTA VDP Status Register
>8C00 VDPWD VDP Write Data
>8C02 VDPWA VDP Write Address
>8C04 VDPCD VDP Color Data
>8C06 VDPID VDP Indirect Data Write

YAMAHA V9938 MANUAL

Manualen som är på 152 sidor säljs av Asgard för USD 30 + porto USD 7. Denna är helt nödvändig om du själv vill skriva program som utnyttjar nyheterna i 9938. Observera att Asgard levererar originalmanualen till skillnad mot vissa andra säljare som har kopierat den med hänvisning till att den inte kan fås från Yamaha.

```
100 !123456789ABCDEF
110 REM XHI TEST MOTION
120 CALL SPRITE(#1,64,1,50,5
0,10,10)
130 CALL LINK("HICLR")
140 ON ERROR 210
150 CALL SCREEN(16)
160 ATTRIB$="080808080808080
808080808080808"
170 CALL LINK("SPRITE",1,70,
ATTRIB$,50,50)
180 CALL MAGNIFY(2)
190 CALL LINK("MOTION",1)
200 CALL KEY(5,K,S):: IF S<1
 THEN 200
210 CALL LINK("NORMAL")
```

PROGRAM-LADDARE FÖR XB

Detta är ett enklare menyprogram än Miami Boot. Det publicerades ursprungligen i 99'er Magazine men har modifierats så att 40 program får plats i menyn och även så att långa XB-program i INT/VAR 254-format kan användas. Det kan användas utan expansionsminne och tar mindre utrymme på skivan. I övrigt är det underläget Miami Boot.

Du placerar din skiva med XB-program i DSK1 och skriver RUN. Programmet läser då skivkatalogen och skapar ett nytt program i MERGE-format med namnet CAT. Skriv sedan NEW och MERGE DSK1.CAT. Spara sedan detta på skivan med SAVE DSK1.LOAD.

Varje gång du använder skivan kan du sedan bara välja Extended Basic så startas automatiskt menyn. Du kan sedan välja godtyckligt program på skivan.

```
100 REM LOAD XB-PROG&I/V 254
110 REM 1987-07-11
120 REM 99'ER MAGAZINE
130 REM MODIFIERAD AV JAN AL
EXANDERSSON (40 rader + INT/
VAR 254)
140 ! RUN
150 ! NEW
160 ! MERGE DSK1.CAT
170 ! SAVE DSK1.LOAD
180 CALL CLEAR :: PRINT "PRO
GRAM STATUS.......WORKING" :
: CL$="CLEAR" :: DIM A$(40):
: OPEN #1:"DSK1.",INPUT ,REL
ATIVE,INTERNAL
190 DEF LN$(N)=CHR$(0)&CHR$(
N)
200 DEF DI$(R)=CHR$(162)&CHR
$(240)&CHR$(183)&CHR$(200)&C
HR$(LEN(STR$(R)))&STR$(R)&CH
R$(179)&CHR$(200)&CHR$(1-1*(
COL>9))&STR$(COL)&CHR$(182)&
CHR$(181)
210 DEF IF$(N)=CHR$(132)&"K@
"&CHR$(190)&CHR$(200)&CHR$(2
)&STR$(N)&CHR$(176)&CHR$(169
)&CHR$(199)&CHR$(LEN(A$(I-64
))+5)&"DSK1."&A$(I-64)
220 FOR I=0 TO 40
230 J=J+1 :: INPUT #1:A$(I),
B,C,D :: IF I=0 THEN 240 ELS
E IF J>=127 OR LEN(A$(I))=0
THEN 250 ELSE IF ABS(B)<>5 A
ND ABS(B)*2+D<262 OR A$(I)="
LOAD" THEN 230
240 NEXT I
250 CLOSE #1 :: EN$=CHR$(181
)&CHR$(199)&CHR$(28)&"PRESS
<ERASE> TO END PROGRAM"&CHR$
(0):: COL=1 :: L=I-1 :: OPEN
 #2:"DSK1.CAT",VARIABLE 163
260 PRINT #2:LN$(1)&CHR$(157
)&CHR$(200)&CHR$(5)&CL$&CHR$
(0)
270 PRINT #2:LN$(2)&DI$(1)&C
HR$(199)&CHR$(28)&"CATALOG"&
RPT$(" ",12-LEN(A$(0)))&"DIS
KNAME-"&A$(0)&CHR$(0)
280 FOR I=1 TO L :: COL=1-15
*(I>20):: RAD=12+I+20*(I>20)
-(MIN(INT(L/2),10)):: PRINT
#2:LN$(I+2)&DI$(RAD)&CHR$(19
9)&CHR$(3+LEN(A$(I)))&CHR$(I
+64)&"--"&A$(I)&CHR$(0):: NE
XT I
290 PRINT #2:LN$(L+3)&CHR$(1
62)&CHR$(240)&CHR$(183)&CHR$
(200)&CHR$(2)&"24"&CHR$(179)
&CHR$(200)&CHR$(1)&"1"&CHR$(
182)&CHR$(238)&EN$
300 PRINT #2:LN$(L+4)&CHR$(1
57)&CHR$(200)&CHR$(3)&"KEY"&
CHR$(183)&CHR$(200)&CHR$(1)&
"0"&CHR$(179)&"K@"&CHR$(179)
&"S@"&CHR$(182)&CHR$(0)
310 PRINT #2:LN$(L+5)&CHR$(1
32)&"S@"&CHR$(190)&CHR$(200)
&CHR$(1)&"0"&CHR$(176)&CHR$(
201)&LN$(L+4)&CHR$(0)
320 FOR I=65 TO L+64 :: PRIN
T #2:LN$(L+I-59)&IF$(I)&CHR$
(0):: NEXT I
330 PRINT #2:LN$(2*L+6)&CHR$
(132)&"K@"&CHR$(190)&CHR$(20
0)&CHR$(1)&"7"&CHR$(176)&CHR
$(157)&CHR$(200)&CHR$(5)&CL$
&CHR$(130)&CHR$(139)&CHR$(0)
340 PRINT #2:LN$(2*L+7)&CHR$
(134)&CHR$(201)&LN$(L+4)&CHR
$(0):CHR$(255)&CHR$(255):: C
LOSE #2 :: DISPLAY AT(23,21)
BEEP:"COMPLETE" :: END
```

PROTOTYPING BOARD

Detta finns nu med ny och utförligare dokumentation. Kortet kan beställas från Bud Mills Services eller L.L.Conner Enterprises. Priset är USD 35+Porto. Se PB 88-3, 88-4.

ANROP AV SUBRUTINER I AL

av R.A.Green, Canada

Översättning från Ottawa T.I.99-4A Users' Group Newsletter volume 8 number 5 May 1989.

Strukturerad programmering, dvs användning av subrutiner, är viktigt vid assemblerprogrammering såväl som vid programmering i något högnivåspråk. I assembler är det möjligt och vanligt att använda kortare subrutiner och att använda dem oftare. Detta medför att det är viktigt att använda en effektiv anropssekvens.

Det finns två instruktioner tillgängliga för anrop av subrutiner. Branch and Link (BL) och Branch and Load Workspace Pointer (BLWP) och deras motsvarande returer, Return (RT) och Return with Workspace Pointer (RTWP).

Generellt så används BL för anrop av interna subrutiner. Interna subrutiner assembleras som en del av det anropande programmet. BL-instruktionen sparar returadressen i register R11 och hopar till subrutinen och använder samma arbetsregister (workspace) som den anropande rutinen. Notera att om subrutinen anropar andra subrutiner så måste R11 sparas på något sätt.

Generellt så används BLWP för externa anrop. Externa subrutiner assembleras separat från den anropande rutinen. BLWP-anropet är elegantare än BL-anropet. En ny workspace sätts upp och så väl som att spara returadressen så sparas workspace pointer (WP) och statusregistret (ST). Notera att ett anrop med BLWP kan anropa andra subrutiner, antingen med BL eller BLWP, utan att spara något register. Vid återhopp återställer BLWP anropande rutins workspace pointer och statusregister. Vi bör komma ihåg att återställning av statusregister återställer interrupt mask.

Låt oss titta på "kostnaden" för de här två sätten att anropa en subrutin.

Ett exempel på BL-anrop är:

```
      BL @ISUB Anropa rutinen
      ...
      ...
ISUB  ...      Subrutin
      ...
      ...
      RT       Återvänd till anro-
*              pande rutin
```

Minneskostnaden är 3 ord (2 för BL, 1 för RT) och tidskostnaden är 5 ord (3 för att hämta instruktionerna, 2 ord för att spara och återställa returadressen).

Ett exempel på BLWP-anrop är:

```
      REF  XSUB     Definiera extern
*                   XSUB
      ...
      BLWP @XSUB
      ...
      ...
      ...
      END
      DEF  XSUB     Definiera namnet
XSUB  DATA XWSP,$+2 Transfervektor
      ...
      ...
      ...
      RTWP          Återvänd
XWSP  BSS  32
      END
```

Minneskostnaden är 21 ord (3 för instruktionerna, 2 för transfervektorn och 16 för workspace) och tidskostnaden är 11 ord (3 för att hämta instruktionerna, 2 för att hämta transfervektorn, 3 för att spara WP, PC och ST, 3 för att återställa WP, PC och ST).

Den här kostnadsanalysen tyder på att BLWP-anrop är mycket dyrare än BL-anrop. Varför då använda BLWP? En anledning är att reducera den tredje kostnaden för ett program, dvs att reducera det arbete Du lägger ner på att skriva ett program. Eftersom en rutin, som anropas med BL, använder samma workspace som den anropande rutinen så måste de två delarna samordna sitt användande av registren. Att ha gemensamma register är lätt med små subrutiner men blir mycket svårt om subrutinerna är stora eller

komplexa. Delning av register medför också att programmet och subrutinen "vet" vad den andra gör. Det medför att subrutinen är för ett speciellt ändamål för just det här programmet.

När BLWP används vid anrop så sker ingen delning av arbetsregistren. Register som används av subrutinen tillhör den subrutinen. Den situationen passar för externa generella subrutiner som är skrivna oberoende av den anropande rutinen och för att användas i flera program.

Eftersom anrop av subrutiner hänger ihop så intimt med överföring av parametrar och används så ofta är det värt ordentlig eftertanke för att reducera de tre kostnader som hänger ihop med skrivandet av dem. Du bör läsa dokumentationen för CALL och RCALL makro (och deras makrodefinitioner) som medföljer Din favoritmakroassambler.

Vi kommer att fortsätta den här diskussionen nästa gång där vi kommer att se att, i en del fall, den högre kostnaden för BLWP-anrop kan kringgås.

SUBRUTINER I ASSEMBLER

av R.A.Green, Canada

Översättning från Ottawa T.I.99-4A Users' Group Newsletter volume 8 number 6 June 1989.

Som vi antydde i förra artikeln så kanske anropet av en subrutin med BLWP inte är så kostsam som det först verkar. Eftersom workspace till anropad subrutin är känd så kan några användbara värden redan ha "laddats" i registren. Du måste komma ihåg att registren inte är något speciellt, de är bara ord i minnet: iden med register är bara ett adresseringssätt som tillåter en 4-bit adress (0 till 15) för att göra instruktioner kortare.

Så väl som att ha värden laddade i dessa register så behöver inte subrutinen reservera utrymme för alla 16 registren om de inte kommer att användas.

För att visa dessa koncept ska vi skriva en bekant rutin, VMBW. Först genom att använda internt anrop och sedan externt anrop. Vi ska sedan summera kostnaderna för varje metod. Notera att ingen av rutinerna är skriven på det sätt som VMBW är skriven i E/A-paketet - båda är bättre. Dessutom är den externa rutinen gjord "korrekt" så att interrupt kan sättas på i huvuddelen av programmet och kan stängas av när VDP används, som man måste göra.

```
*
*VMBW Internt anrop
*     Inget av anropande rutins
*     register ändras
*     Interrupt måste stängas av
*
*     LI   R0,Vaddr
*     LI   R1,Caddr
*     LI   R2,count
*     BL   @VMBW        Kostnad
*                       Minne Tid
* Spar R3
VMBW  MOV  R3,@S3         2    4
* Spar R2
      MOV  R2,@S2         2    4
* R3 = VDP adress
      MOV  R0,R3          1    3
* Sätt VDP skriv bit
      ORI  R3,>4000       2    3
* Sätt VDP adress
      MOVB R3,@>8C02      2    4
      SWPB R3             1    2
      MOVB R3,@>8C02      2    4
* R1 = CPU adress
      MOV  R1,R3          1    3
* Byte till VDP
VMBW2 MOVB *R3+,@>8C00    2    5*
      DEC  R2             1    2*
* Slinga för alla byte
      JGT  VMBW2          1    1*
* Aterställ R3
      MOV  @S3,R3         2    4
* Aterställ R2
      MOV  @S2,R2         2    4
* Atervänd till anropande rutin
      RT                  1    2
*
```

```
* För att spara R3
S3    BSS  2            1     -
* För att spara R2
S2    BSS  2            1     -
*                      ----  ----
* Total kostnad         24    37+8 per
*                                byte

*
*VMBW Externt anrop
*     Inget av anropande rutins
*     register ändras
*     Interrupt kan sättas på och
*     stängas av
*
*     LI   R0,Vaddr
*     LI   R1,Caddr
*     LI   R2,count
*     BLWP @VMBW       Kostnad
                       Minne Tid
* Definiera extern rutin
      DEF  VMBW
* Kort WSP
VMBW  DATA SWSP-18,$+2   2     -
* Interrupt av
      LIMI 0             2     2
* R12 till anropande rutins R0
      MOV  R13,R12       1     3
* R11 = VDP adress
      MOV  *R12+,R11     1     4
* Sätt VDP skriv bit
      ORI  R11,>4000     2     3
* Sätt VDP adress
      MOVB R11,*R10      1     4
      SWPB R11           1     2
      MOVB R11,*R10      1     4
* R11 = CPU adress
      MOV  *R12+,R11     1     4
* R12 = räknare
      MOV  *R12,R12      1     4
* Byte till VDP
VMBW2 MOVB *R11+,*R9     1     5*
      DEC  R2            1     2*
* Slinga för alla byte
      JGT  VMBW2         1     1*
* Återvänd till anropande rutin
      RTWP               1     4
*
* R9, R10 laddas i förväg
SWSP  DATA >8C00,8C02    2     -
* R11, R12 arbetsregister
      BSS  4             2     -
* R12, R15 länkning
      BSS  6             3     -
*                      ----  ----
* Total kostnad         24    34+8 per
*                                byte
```

Som Du kan se av denna inte speciellt komplexa rutin så kunde vi kringgå nackdelen med minne och tidskostnad vid användning av BLWP-anrop; och samtidigt lägga till möjligheten med interrupt till/från.

TIPS VOL. 4 has 48 more from issues No. 33 through 41. NOW JUST $10 EACH, POSTPAID.

TIGERCUB CARE DISKS #1,#2,#3 and #4. Full disks of text files (printer required). No. 1 contains the Tips news letters #42 thru #45, etc. Nos. 2 and 3 have articles mostly on Extended Basic programming. No. 4 contains Tips newsletters Nos. 46-52. These were prepared for user group newsletter editors but are available to anyone else for $5 each postpaid.

This educational program is a much expanded version of a routine I published before.

```
100 DIM M$(100)
110 GOTO 150
120 S,K,A$(),J,M$(),Y$,Z$,Z,
X,ING$,A,AN$
130 CALL CLEAR :: CALL COLOR
 :: CALL SCREEN :: CALL CHAR
 :: CALL KEY :: CALL ING ::
CALL HCHAR
140 !@P-
150 CALL CLEAR :: FOR S=0 TO
 12 :: CALL COLOR(S,2,8):: N
EXT S :: CALL SCREEN(5):: DI
SPLAY AT(3,1):"LEARNING TO "
"ING"" IT  V.1.1"
160 CALL CHAR(64,"3C4299A1A1
99423C"):: DISPLAY AT(5,1):"
@ Tigercub Software 1987 for
free distribution - no price
or copying fee to be charged
"
170 CALL KEY(3,K,S)
180 A$(1)="No, if the word d
oes not end in B, D, G, M, N
, P, R or T you always just
add ING"
190 A$(2)="No,if the last le
tter is not E and the next-t
o-last     letter is not a v
owel, just add ING"
200 A$(3)="No, if the word h
as two     vowels just befor
e the last letter, just add
ING"
210 A$(4)="No, if a word end
s in B, D, G, M, N, P, R or
T with one vowel (but not tw
o vowels!) just before it, y
ou must     double the last
letter and add ING"
220 A$(5)="No, if the word e
nds in IE, change the IE to
Y and add  ING"
230 A$(6)="No, BE is an exce
ption to   the rules,"
240 A$(7)="Some dictionaries
 give EYING but EYEING is be
tter"
250 A$(8)="No, if a word end
s in E (ex-cept BE and words
  ending in IE,OE,UE AND YE)
 you must drop the E and add
 ING"
260 A$(9)="No, if the word e
nds in EE, or OE or UE, just
 add ING"
270 A$(10)="No, QUIP, QUIT a
nd QUIZ are exceptions to th
e rule.     Double the last
letter and  add ING."
280 FOR J=1 TO 100 :: READ M
$(J):: NEXT J
290 FOR J=1 TO 100 :: Y$=Y$&
CHR$(J):: NEXT J :: Z$=Y$
300 DISPLAY AT(3,1):"":"":""
:" Type the word with the
  correct ING suffix"
310 RANDOMIZE :: Z=INT(RND*L
EN(Z$)+1):: X=ASC(SEG$(Z$,Z,
1)):: Z$=SEG$(Z$,1,Z-1)&SEG$
(Z$,Z+1,255):: IF LEN(Z$)=0
THEN Z$=Y$
320 CALL ING(M$(X),ING$,A)
330 DISPLAY AT(12,1):M$(X)::
 ACCEPT AT(12,15):AN$
340 CALL HCHAR(15,1,32,280):
: DISPLAY AT(10,1):"" :: IF
AN$=ING$ THEN DISPLAY AT(10,
10):"CORRECT!" :: GOTO 310
350 DISPLAY AT(15,1):A$(A):"
":"The word is ";ING$ :: GOT
O 310
360 !@P+
370 DATA LODGE,BUY,HOPE,QUIP
,TITHE,WISH,CUT,DRIVE,SEE,EY
E,GO,CRY,TRY,AGREE,QUIT
380 !@P-
390 DATA BOIL,COOL,HURT,BUTT
,CAGE,BE,ROVE,PITY,SAVE,COOL
,RULE,MEASURE,TUNE,RAVE
400 DATA RUN,BEG,STOP,THINK,
ERR,BORE,TEAR,BAR,CARE,BARE,
BEAR,LET,QUIZ,HOOT,HEAT,COME
410 DATA DREAM,TAKE,FRY,CADD
Y,FLEE,HOE,SEW,TRIP,HOPE,RIG
,DRAG,SUE,KNEE,BOO,BABY,NURS
E,CRUISE
420 DATA LIE,TIE,DIE,BELIE,V
IE,DODGE,LIVE,DRIVE,LOVE,LEA
VE,HUM,HOP,BEG,BEGIN,BOMB,BO
B
430 DATA ADD,AID,BAT,BOAT,PR
AY,LAY,QUOTE,SNORE,STARE,HIR
```

```
E,FIRE,LINE,CRY,SAY
440 DATA BOOGIE,RAGE,RATTLE,
GRATE,LEAVE,STRIVE,DRAW,WRIT
E
450 !@P+
460 SUB ING(M$,ING$,A):: E$=
SEG$(M$,LEN(M$),1):: F$=SEG$
(M$,LEN(M$)-1,1):: A$="ING"
:: C$="BDEGMNPRT" :: V$="AEI
OU"
470 GOTO 500
480 C$,E$,ING$,M$,A$,A,V$,F$
490 !@P-
500 IF LEN(M$)=4 AND SEG$(M$
,1,3)="QUI" THEN ING$=M$&E$&
A$ :: A=10 :: SUBEXIT
510 IF POS(C$,E$,1)=0 THEN I
NG$=M$&A$ :: A=1 :: SUBEXIT
520 IF E$="E" THEN 550
530 IF POS(V$,F$,1)=0 THEN I
NG$=M$&A$ :: A=2 :: SUBEXIT
540 IF POS(V$,SEG$(M$,LEN(M$
)-2,1),1)<>0 THEN ING$=M$&A$
 :: A=3 :: SUBEXIT ELSE ING$
=M$&E$&A$ :: A=4 :: SUBEXIT
550 IF F$="I" THEN ING$=SEG$
(M$,1,LEN(M$)-2)&"YING" :: A
=5 :: SUBEXIT ELSE IF F$="E"
 OR F$="O" OR F$="U" THEN IN
G$=M$&A$ :: A=9 :: SUBEXIT
560 IF M$="BE" THEN ING$="BE
ING" :: A=6 :: SUBEXIT
570 IF M$="EYE" THEN ING$="E
YEING" :: A=7 :: SUBEXIT
580 ING$=SEG$(M$,1,LEN(M$)-1
)&A$ :: A=8
590 !@P+
600 SUBEND
```

I still have a sort of an old-fashioned idea that the computer can be a useful educational tool -

```
100 CALL CLEAR :: FOR SET=0
TO 12 :: CALL COLOR(SET,2,8)
:: NEXT SET :: CALL SCREEN(5
):: DISPLAY AT(3,6):"NOUN TO
 ADJECTIVE" :: CALL KEY(3,K,
S)
110 CALL CHAR(64,"3C4299A1A1
99423C"):: DISPLAY AT(5,5):"
@ Tigercub Software":"":" Fo
r free distribution - no  pr
ice or copying fee to be  ch
arged."
120 DISPLAY AT(12,1):" One m
oment...loading memory"
130 DATA ROGUE,ROGUISH,HOG,H
OGGISH,PIG,PIGGISH,SWINE,SWI
NISH,THIEF,THIEVISH,KNAVE,KN
AVISH,BRUTE,BRUTISH or BRUTA
L
140 !@P-
150 DATA FAME,FAMOUS,TUMULT,
TUMULTUOUS,RIOT,RIOTOUS,SCAN
DAL,SCANDALOUS,MOUNTAIN,MOUN
TAINOUS,ODOR,ODOROUS or ODOR
IFEROUS
160 DATA CAVERN,CAVERNOUS,VI
LLAIN,VILLAINOUS,DANGER,DANG
EROUS,PERIL,PERILOUS,ADVANTA
GE,ADVANTAGEOUS
170 DATA BARB,BARBED,FORK,FO
RKED,BORDER,BORDERED,WHEEL,W
HEELED,HUNGER,HUNGRY,ANGER,A
NGRY
180 DATA PARLIAMENT,PARLIAME
NTARY,PLANET,PLANETARY,LEGIS
LATURE,LEGISLATIVE,PARISH,PA
ROCHIAL
190 DATA CONGRESS,CONGRESSIO
NAL,ELEPHANT,ELEPHANTINE,FAN
TASY,FANTASTIC,BULL,BULLISH
200 DATA GIRL,GIRLISH,BOY,BO
YISH,BABY,BABYISH,AMATEUR,AM
ATEURISH,FEVER,FEVERISH,DEVI
L,DEVILISH,FOOL,FOOLISH
210 DATA OAF,OAFISH,SHEEP,SH
EEPISH,CHILD,CHILDISH or CHI
LDLIKE,VIRTUE,VIRTUOUS,PRIDE
,PROUD or PRIDEFUL
220 DATA HATE,HATEFUL,DOUBT,
DOUBTFUL,THOUGHT,THOUGHTFUL,
SHAME,SHAMEFUL,FEAR,FEARFUL,
SORROW,SORROWFUL
230 DATA WISH,WISHFUL,PEACE,
PEACEFUL,EVENT,EVENTFUL,TRUT
H,TRUTHFUL,SKILL,SKILLFUL,MA
N,MANLY
240 DATA WOMAN,WOMANLY,FATHE
R,FATHERLY,MOTHER,MOTHERLY,B
ROTHER,BROTHERLY,SISTER,SIST
ERLY
250 DATA NIGHT,NIGHTLY,HOUR,
HOURLY,MONTH,MONTHLY,ORDER,O
RDERLY,SERIES,SERIAL
260 DATA TIME,TIMELY,GRAVEL,
GRAVELLY,FRIEND,FRIENDLY,WOO
L,WOOLLY,YEAR,YEARLY,SOUTH,S
OUTHERN or SOUTHERLY
270 DATA NORTH,NORTHERN or N
ORTHERLY,WEST,WESTERN or WES
TERLY,EAST,EASTERN or EASTER
LY
280 DATA CHARITY,CHARITABLE,
TERROR,TERRIFIED or TERRIBLE
,HORROR,HORRIFIED or HORRIBL
E or HORRIFIC
290 DATA RAG,RAGGED,MILITARY
,MILITARISTIC,ART,ARTISTIC,C
AT,CATTY,DOG,DOGGY,FOG,FOGGY
,SUN,SUNNY
300 DATA BAG,BAGGY,LEG,LEGGY
```

```
,BOG,BOGGY,STUB,STUBBY,FUN,F
UNNY,FUR,FURRY,GUM,GUMMY,AVA
RICE,AVARICIOUS
310 DATA CLOUD,CLOUDY,RAIN,R
AINY,FLOWER,FLOWERY or FLORA
L,GREED,GREEDY,THIRST,THIRST
Y,AIR,AIRY,BUSH,BUSHY,FISH,F
ISHY
320 DATA SOUP,SOUPY,BLOOD,BL
OODY,FOAM,FOAMY,BEAD,BEADY,S
WAMP,SWAMPY,SILVER,SILVERY,C
OPPER,COPPERY,DUST,DUSTY
330 DATA DIRT,DIRTY,GUILT,GU
ILTY,SALT,SALTY,GRAIN,GRAINY
,OIL,OILY,TRICK,TRICKY,HILL,
HILLY,ROCK,ROCKY
340 DATA SAND,SANDY,SOAP,SOA
PY,SUDS,SUDSY,SILK,SILKY,WOO
D,WOODY,MODESTY,MODEST,PIETY
,PIOUS,DAY,DAILY
350 DATA TREE,TREELIKE,TOY,T
OYLIKE,FINGER,FINGERLIKE,SWA
N,SWANLIKE,WAR,WARLIKE,DISH,
DISHLIKE,PLATE,PLATELIKE
360 DATA SPOON,SPOONLIKE,BIR
D,BIRDLIKE,SNAKE,SNAKY,WIRE,
WIRY,BONE,BONY,SMOKE,SMOKY,F
LAKE,FLAKY
370 DATA NOISE,NOISY,BRINE,B
RINY,TASTE,TASTY,STONE,STONY
,WAVE,WAVY,GORE,GORY,PASTE,P
ASTY,BUBBLE,BUBBLY
380 DATA LABOR,LABORIOUS,ORN
AMENT,ORNAMENTAL,GOVERNMENT,
GOVERNMENTAL,CONTINENT,CONTI
NENTAL,MUSIC,MUSICAL
390 DATA MAGIC,MAGICAL,TOPIC
,TOPICAL,SENSATION,SENSATION
AL,LOGIC,LOGICAL,ALARM,ALARM
ING,ARTERY,ARTERIAL
400 DATA GOLD,GOLDEN,EARTH,E
ARTHEN,GLAMOUR,GLAMOURIZED,D
EPUTY,DEPUTIZED,ENERGY,ENERG
IZED,PART,PARTIAL,FIRE,FIERY
410 DATA ANGEL,ANGELIC,CHERU
B,CHERUBIC,BURDEN,BURDENSOME
,TROUBLE,TROUBLESOME,BEAST,B
ESTIAL
420 DATA HISTORY,HISTORICAL,
GEOGRAPHY,GEOGRAPHICAL,BOTAN
Y,BOTANICAL,BIOLOGY,BIOLOGIC
AL,LITURGY,LITURGICAL
430 !@P+
440 DIM A$(175),B$(175):: FO
R J=1 TO 174 :: READ A$(J),B
$(J):: Z$=Z$&CHR$(J):: NEXT
J :: Y$=Z$ :: RANDOMIZE
450 DISPLAY AT(7,1):"":"Type
 the adjective form of -":""
460 X=INT(RND*LEN(Y$)+1):: Y
=ASC(SEG$(Y$,X,1)):: Y$=SEG$
(Y$,1,X-1)&SEG$(Y$,X+1,255):
: IF LEN(Y$)=0 THEN Y$=Z$
470 DISPLAY AT(12,1):A$(Y)::
 ACCEPT AT(12,14):Q$ :: IF P
OS(B$(Y),Q$,1)=0 THEN 490
480 DISPLAY AT(18,1):"":"" :
: FOR D=1 TO 100 :: NEXT D :
: DISPLAY AT(18,1):" That is
 the word in my      memory b
anks.":"" :: GOTO 460
490 DISPLAY AT(18,1):" The a
djective in my memory banks
is ";B$(Y):: GOTO 460
```

When one program is run from from another by RUN DSK.., the screen is not cleared, sprites are not deleted, and screen color, character definitions and sprite magnification are not returned to the default values. This can cause some strange results, which can be prevented by CALLing CLEARALL just before the RUN.

```
1000 SUB CLEARALL :: CALL CL
EAR :: CALL DELSPRITE(ALL)::
 CALL SCREEN(8):: CALL CHARS
ET :: CALL MAGNIFY(1)
1001 FOR CH=65 TO 90 :: CALL
 CHARPAT(CH,CH$):: CALL CHAR
(CH+32,"00"&SEG$(CH$,1,12)&S
EG$(CH$,15,2)):: NEXT CH
1002 CALL CHAR(96,"000201008
",123,"0018202040202018",124
,"00101010001010100030080804
0808300000205408")
1003 FOR CH=127 TO 143 :: CA
LL CHAR(CH,"0"):: NEXT CH ::
 SUBEND
```

The routine in line 1001 can be used, by deleting the +32 if necessary, to modify some of the character sets on my Nuts & Bolts disks.

From an idea in a program by Ed Machonis, here is an improvement to my 28-Column Converter published in Tips #18. After line 160, insert

```
165 DISPLAY AT(20,1):"Tab se
tting? 1" :: ACCEPT AT(20,14
)SIZE(-2)BEEP:T
```

And change line 290 to -

```
290 PRINT #2:TAB(T);L$ :: S=
S+28 :: GOTO 410
```

MEMORY FULL! - Jim P.

EXTENDED BASIC

av okänd amerikansk författare

This is the first of a series of tutorials on extended basic programming. It is designed for the user who has a basic familiarity with the TI computer and with extended basic; that is to say, if you have ever written a program and got it running, you will be able to follow these lessons. We will start at the very beginning and end by writing two programs - one graphic and music intensive, the other menu and disk intensive. The programs will use advanced techniques, so will have value for the serious advanced programmer, as well as the "advanced novice".

This first lesson will discuss programming philosophy in general, define the outline of the series, and end by writing a module to set up a title screen.

Way back in the dark ages I found myself temporarily finantially embarrassed (that means flat broke). I, of course, did what all good computer buffs do - used my computer to get myself out of it. Accordingly, I took a job teaching programming at a local commercial computer school.

This was a typical South Florida institution - take their money and keep them happy, and damn if they learn anything. For the first week, I sat in on classes. I observed scared, intimidated, housewives, being taught extremely technical material by knowledgeable teachers. No one was learning anything.

My first day of teaching, I presented the following problem. You want to make dinner. You want to make steak, but can only afford it if steak costs less than $4.00 a pound. If it is more than that, the family must settle on hamburger. You are going to work, and must leave a note to your kid to do the shopping. Can you write it? Of course everyone did very easily. then I asked them to write it in Russian. "We can't, they complained, we don't know Russian!" Then I sprang it on them. You have just written a program. Trouble is, the computer doesn't understand English, and you don't understand basic. The point is, I am not going to teach you how to program - you already know that, and have been doing it for years. What I am going to do is teach you a language -basic- that the computer understands. You will then be able to communicate with the computer.

Make no mistake about it, programming is an art. You can be taught English, or Russian, or Basic. You can be taught the techniques to manipulate words in the most pleasing and efficient fashon - but to write King Lear or Doctor Zhivago or Fastterm is an art. You can't be taught art.

The TI is blessed with the most powerful basic I have ever seen. I am referring of course to Extended Basic. Console Basic, for the purposes of serious programming, does not exist. Basic is much maligned, but oddly enough it is one of the most powerful and useful languages ever written. It is easy to understand, is interactive, has powerful I/O capabilities, extensive string handling routines, and in short, can do anything that any other language can do. It has two problems; one, it can, because of its flexibility, be used to write totally unstructured code, which we will learn to avoid, and two, especially on the TI, it is slower than molasses in a hailstorm, which unfortunately we are stuck with.

As I stated before, this series will result in writing two programs. The first will be a game, ZAPA, the second a personal record kepping program. Both will be relatively simple and not extremely useful compared to others in these databases, but will be used to illustrate the techniques.

For our game we first define the rules. This will be a two player game. We will set up a screen with

a number of randomly placed blocks. Each player can move in all eight directions. Hitting a block will loose you points (if you have any) but turn the block into your emblem. The object is to hit the OTHER person's emblem, which gets you points - but when you do, it turns into your emblem. Hitting the border immediately looses - and hitting the other player's man immediately wins.

The design of the graphics will be as follows: the screen will turn blue, then we will print "ZAPA" in large black block letters. As soon as this is printed, we will cause a red outline to appear around the title, turn the screen inside the red border to yellow, and turn the letters to light blue. This will all happen apparently instaneously. The music will start, and when any key is pressed, the title screen will be erased, and the playing screen will appear. It will have a light blue border, a dark blue outline, and a yellow backround inside the outline. There will be a grey scoring area at the top. The blocks will be square magenta and the players a rotated square with a circle in the center. the players will be dark red and dark green with the insignias light red and light green of the same design, we hope. We may run out of character sets. See, I havn't programmed this yet - we are doing it together. This gives an opportunity for everyone to review the code as written, and make comments and improvements.

The most time consuming, boring, and important part comes now. Sit down with graph paper and design the elements described. Next, assign the verious characters to character sets, according to the colors to be used. (read the description of CALL CHAR, CALL COLOR, and CALL SCREEN.)

We will start at the last character set, and work backwards until we run into a character we need (such as a number). Some juggling may have to be done later. For the title screen, we will assign the background to set 1 (so the screen may be initially filled with spaces and a simple CALL CLEAR will do), the red outline to set 14, the yellow background to set 13, and the letters "ZAPA to set 12. For simplicity, we will use the first character in each set. this makes the red border CHR$(136), the background CHR$(128), and the letters CHR$(120)

After graphing out the title screen, we find that the block needs to be 27 by 11. Since the screen is 32 by 24, this leaves 5 to be distributed at the sides and 13 at the top and bottom. We will divide this into 3 on the left, 2 on the right, 6 on the top, and 5 on the bottom. This defines the position of all the characters of the title screen.

The boring part is over for the moment. We have done our TOP DOWN design. We will now do some BOTTOM UP programming. This is often considered impossible in BASIC (read Brodie: STARTING FORTH for more information) but in TI Extended basic it is not only possible, or easy, it is the simplest way because of one statement: CALL XXX(perameter).

The CALL statement is often considered an advanced technique, but really, you have been using it often. Every time you use CALL, as in CALL CLEAR, CALL CHAR, etc., the basic interpeter CALLS a GPL (Graphics Programming Language) subroutine. You may write a basic subroutine which the basic interpeter handles exactly like a GPL routine. Read CALL on page 55 of the EXB manual, SUB on page 180, and SUBEND and SUBEXIT on page 184. You will then be totally confused, but not to worry.

Why use CALL rather than GOSUB - RETURN? Several reasons. Call doesn't need line numbers, so you don't have to remember where you stuck a subprogram, or find it after RES. The variables are local to the program, so you can use M$ in the main program and M$ in the subprogram, and they are actually different variables. You can write code that explains itself. example, what does this code do?

```
100 CALL TITLE
110 CALL MUSIC
120 GOTO 100
1000 SUB MUSIC
```

```
1010 CALL NOTE(DONE)
1020 CALL CHECKKEY(PRESSED)
1030 IF PRESSED=1 THEN DONE=1::CALL
     GAME
1040 IF DONE=1 THEN SUBEXIT ELSE
     GOTO 1010
1050 SUBEND
2000 SUB TITLE
2010 REM CODE GOES HERE
2020 SUBEND
3000 SUB NOTE(FINISHED)
3010 REM CODE GOES HERE
3020 SUBEND
4000 SUB CHECKKEY(PUSHED)
4010 REM CODE GOES HERE
4020 SUBEND
5000 SUB GAME
5010 REM CODE GOES HERE
5020 CALL PRINTSCORE(SCORE)
5030 SUBEND
6000 SUB PRINTSCORE(TOTAL)
6010 REM CODE GOES HERE
6020 SUBEND
```

The above code contains some non-trivial control logic. Without any explanation or documentation, you should be able to completely follow the program logic and see exactly what the program does and how it does it. The only help you may need is to explain that what is in the () in the CALL is assigned to what is in the () in the SUB: that is, PRESSED is equated to PUSHED, etc. When it comes time to debug, see how simple it is!

But the real subtle, dramatic, and awe-inspiring power is this....

YOU HAVE JUST WRITTEN YOUR OWN EXTENSION TO EXTENDED BASIC!!!

Never again will you have to spend your valuable time writing a title program, or a music program, or a checkkey program. For the first two, you must merely change the data. The checkey module, needing no data, is finished for all time! Sooner or later, you will write a total, complete, program, without ever writing a line of code - you will just reassemble parts from your master disk!

You really do this now, you know. When you write CALL CLEAR, you essentially take a subprogram out of ROM and put it in your program. What's the difference in taking it off the disk?

Don't worry if the actual method does not seem clear at this point. We have learned that this language has some advantages over English in that if there is no word that has the exact meaning we want, we can just make one up....

T'was brillig, and the slithy toves
Did gyre and Gimbol in the wabe.....

Now to get down to the nitty gritty and make up our new word. We will call our word TITLE and it will print the title screen.

For this subprogram we will use CALL HCHAR, CALL VCHAR, READ, and DATA. Read the descriptions of these statements now.

Our data statement will give a start row, start column, character code, and repetition for each design to be printed. How will we know when we are done? simple - just put a "0" in the data for start row! so we can see the code we need.....

```
100 READ ROW
110 IF ROW=0 THEN 200
120 READ COL,CHARA,REP
130 CALL HCHAR(ROW,COL,CHARA,REP)
140 GOTO 100
200 READ ROW
210 IF ROW=0 THEN STOP
220 READ COL,CHARA,REP
230 CALL VCHAR(ROW,COL,CHARA,REP)
240 GOTO 200
```

Time now to fire up the trusty TI. Key in this code.

Now let's work out the data.

We previously defined that the title block will start at row 7 col 4, and the character is 136. So we write...

```
10 DATA 7,3,136
```

Now our graph paper tells us that the block is 27 characters wide, so we add.....

```
10 DATA 7,4,137,27
```

Next, the bottom red line, which starts 10 down, at 17

```
10 DATA 7,4,137,27,17,4,137,27,0
```

Why the "0"? Remember, that tells the ccomputer that we are done with the horizontal lines.

Now we will put in the vertical lines of the outline. We could start at the same place and end at the same place, but it will save a very small amount of time if we don't overprint the starting and ending characters, and small amounts add up. So we will add.....

```
10 DATA 7,4,137,27,17,4,137,27,0,8,
   4,137,9,8,30,9,0 !TITLE BLOCK
   OUTLINE
```

Now we can run this.....of couse, all we will see is some characters disappear.....because we never defined character 137! So, just to check (although we NEVER make a mistake) we can stick in a CALL CLEAR at line 90 and change the 137's to 042. Try this, and run it. If you made no typos, you will see a square box on the screen.

The next step is to fill in the outline we have just drawn with the yellow background. Remember, we said we would use CHR$(128) for the background in the box. How shall we do that? Pause a minute and think of how.

Maybe you realized that all we have to do is to add 9 horizontal lines in our data statement. If you did, you are right, but think a minute. That will be 9 identical data statements. That takes a lot of memory (not to mention typing) and data uses a lot of memory - it can't be tokenized! Also you might have realized that we can't just add it to the end of the data statements, because the computer is done with the horizontal lines, and at present, has no way of going back. So we must be careful how we do this.

I hope by this time you are thinking ahead of me. We will just figure out a way to put in a repeat function, and at the same time, make a loop that allows the computer to start over until we are done with all the graphics.

How shall we tell the computer to repeat a line?

Well, we told it to repeat a character by putting a repeat character field in the data. We can do the same for repeat line. But think a minute! That will mean that we must go back and put a "0" in all the lines which don't repeat. Is this a good idea?

Maybe yes, maybe no. Let's think about it. Most lines repeat characters, so we will waste little memory by putting in the repeat character field in all statements. But so far, it doesn't look like many lines will repeat. Do we have to waste the memory?

NO!

Look at the code. The first thing we do is check if row=0 or a start position. But we don't have to put a positive number here. We can just as easily make it a negative number. Of course, by the time it gets to the CALL CHAR statement it must be positive or an error will occur, but no problem!

Same with the repeat. Up to now, a "0" in the row of the vertical print meant end the routine. It could just as easily mean loop to the beginning. So let's write the code!

But wait! Did we forget something? How do we end? Well, we don't have to use "0" for the final vertical stop. Any number will do - as long as it is bigger than 24. We can't have a row over 24 now, can we?

So let's pick a stop number. Let's make it distinctive, so we can easily spot it in a large data table. like 999.

So now we can write the code.

Filling in the new lines we can write.....

```
100 READ ROW
110 IF ROW=0 THEN 200
120 READ COL,CHARA,REP
122 IF ROW>0 THEN 130 !IF WE DON'T
    REPEAT A ROW, THIS LOOPS
    TO THE SAME PLACE AS BEFORE
124 ROW=-ROW !IF WE GET HERE, WE
    WILL BE REPEATING A ROW, SO WE
    BETTER MAKE ROW POSITIVE BEFORE
```

```
    WE CALL CHAR
126 READ HREP !HERE WE READ THE
    ADDITIONAL DATA OF HOW MANY TIMES
    TO REPEAT THE ROW
128 HREP=HREP-1 !WHY DID WE DO THIS?
    FOR THE ANSWER, DELETE THIS LINE
    AFTER THE PROGRAM WORKS AND SEE
130 CALL HCHAR(ROW,COL,CHARA,REP)
132 IF HREP=0 THEN 100 ELSE HREP=
    HREP-1::ROW=ROW+1::GOTO 130 !WE
    CANNOT MAKE THIS 3 LINES AS IT
    IS ALL PART OF THE IF THEN ELSE
200 READ ROW
210 IF ROW=0 THEN 100 ELSE IF ROW=999
    THEN STOP !HERE WE ADDED LOOP
    BACK AND LOOK FOR MORE DATA
220 READ ROW,COL,CHARA,REP
230 CALL VCHAR(ROW,COL,CHARA,REP)
240 GOTO 100
```

Now we can add the data statement to fill in the border with one simple data sstatement. For the time being, however, let's use CHR$(046) insted of CHR$(128). You'll see why later.

```
20 DATA -8,5,046,25,9,0,999 !BORDER
```

Notice the "0" - that is because we aren't doing any vertical printing here, only horizontal. And the "999" tells the computer that we are done.

Now we can put in the "ZAPA". On my graph paper the screen looks:

```
----------------------------------
[                                ] 1
[                                ] 2
[                                ] 3
[                                ] 4
[                                ] 5
[12345678901234567890123456789012] 6
[   ***************************  ] 7
[   *.........................*  ] 8
[   *.xxxxx.xxxxx.xxxxx.xxxxx.*  ] 9
[   *.....x.x...x.x...x.x...x.*  ] 10
[   *....x..x...x.x...x.x...x.*  ] 11
[   *...x...x...x.x...x.x...x.*  ] 12
[   *..x....xxxxx.xxxxx.xxxxx.*  ] 13
[   *.x.....x...x.x.....x...x.*  ] 14
[   *.xxxxx.x...x.x.....x...x.*  ] 15
[   *.............x...........*  ] 16
[   ***************************  ] 17
[                                ] 18
[                                ] 19
[                                ] 20
[                                ] 21
[                                ] 22
[                                ] 23
[                                ] 24
----------------------------------
```

Now you can see that printing the letters is just a matter of counting the positions of the X's and putting them into data statements. I get....

```
30 DATA 9,6,088,5,15,6,088,5,0,0
   !REM Z
40 DATA 9,12,088,5,13,13,088,3,0,10,
   12,088,6,10,16,088,6,0 !REM A
50 DATA 9,18,088,5,13,19,088,3,0,10,
   18,088,7,10,22,088,4,0 !REM P
60 DATA 9,24,088,5,13,25,088,3,0,
   10,24,088,6,10,28,088,6,999
```

Notice that I have used 088 for the character instead of 120 which we had previously agreed upon. Why don't I just define the characters and color sets now?

Well, I'll bet the light is dawning. We will write another program to take the characters and colors out of data statements, of course. For now it is easier to debug by just substituting the character. We have used 088 instead of 88 so that later we can just type over.

Also, some of the more astute programmers might have noticed that there is a lot of repeated data. We could have saved memory by only keying in the character once. You are right!

But wait! There's more! You also get.....

It will become clear. Have faith!

Now run the program. What do we see?

Oh, No! there is no diagonal on the "Z"!

Do we have to put in all that data just for 5 louzy "Z" blocks? I can see the lights flashing and the bells ringing! By now, most of you are totally ignoring my question and instead asking yourselves.....

How in the hell to we program a diagonal line?

That's the right question, of course.

Up to now, we have been writing statements which were as simple as possible. I'm sure that you are

champing at the bit to start writing some of those lovely convoluted statements which do 749 things in one line. OK... I give in! we will change line 210.....

```
210 IF ROW=0 THEN 100 ELSE IF ROW=-1
    THEN THEN GOSUB 300 :: GOTO 100
    ELSE IF ROW=999 THEN STOP
```

And add the following lines.....

```
300 READ ROW :: IF ROW=0 THEN RETURN
    ELSE READ COL,CHARA,REP
310 FOR I=1 TO REP
320 CALL HCHAR(ROW,COL,CHARA)
330 ROW=ROW+1
340 COL=COL-1
350 NEXT I
360 GOTO 300
```

Now add the data in line 30.....

```
30 DATA 9,6,088,5,15,6,088,5,0,
   -1,10,10,088,5,0
```

Run the program Does it work? It does? Glory be! Neither of us made any typo's!

Now we will do a little fancy disk work. Save the program to DSK1.Q. Then delete all the data statements. Then add the following statememts...

```
10 SUB PRINTSCREEN
1000 SUBEND
```

Now type.....

```
RES 10000,1
```

Yes, that's ten thousand comma one. Then save that to DSK1.PRNTSCRN0, MERGE. Now recall the program on DSK1.Q. Delete all the program steps, but not the data! Now type...

```
RES 1,1
```

And save this to DSK1.SCRNDAT0,MERGE

Why did we do this?

We are building a convention for simple programming. We will decide to put all our data between 1 and 1000 in steps of 100. Since we end our file in "0" we know the line numbers are in the 0 hundred block, or 0 to 99. Similarily, the sub-programs all start with 10,000 and go up in blocks of 100. PRINTSCREEN then is numbered from 10,000 to 44,099.

Now we start writing our program

We type.....

```
1000 CALL CLEAR
1010 CALL PRINTSCREEN
```

That's our program!

Of course, we have to add the data..

```
MERGE DSK1.SCRNDAT0
```

And we have to inform the computer to the extension to Extended Basic..

```
MERGE DSK1.PRNTSCRN0
```

Now we type "RUN" and it merrily churns out the screen!

So far.....

You see what we did. We wrote a powerful segment. We will never write it again. From now on, you can spend all your time thinking up those fancy, dramatic screens and not waste time figuring out how to program them, because you have added the CALL PRINTSCREEN statememnt to your own personal version of Extended basic. You have all the controls in the DATA statements. 0 ends both horizontal and vertical print, -1 starts a diagonal, 999 ends the screen. It is totally complete!

What? it isn't complete? You're right! We can print a diagonal from right to left, but not one from left to right!

SO.....

We have a homework project! Modify the CALL PRINTSCREEN statement to also print a diagonal from left to right.

THINK CALL!! Bongiovonni