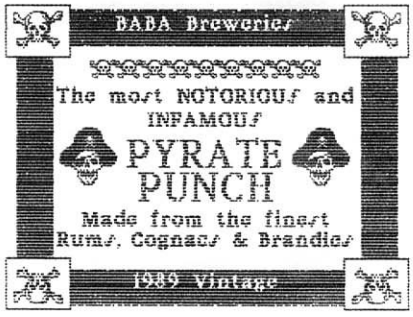
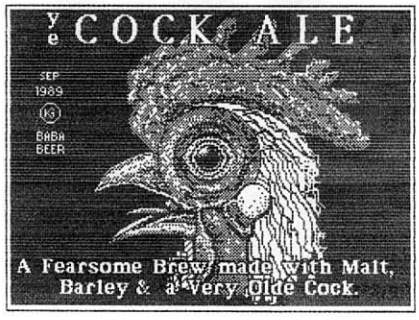
PROGRAM BITEN 91-6





ISSN 0281-1146

REDAKTÖREN

INIT av tangentbord i PB 91-2.14 3:e raden efter label INIT ska ändras:

```
CLR  R0
MOVB R0,@>8374
MOVB R0,@>837A
```

TI*MES, TI99/4A User's Group (UK), har fått ny kontakt för medlemsskap (15 GBP/år): Philip Trotter, 80 Martonburn Rd., Grovehill, Middelsborough, TS4 2TH, Storbritannien.

I PB har vi publicerat program och artiklar från BUG NEWS utgiven av BREA 99'ers. Tidning på 10 sidor ges ut en gång/månaden (15 USD/år inom USA). Artiklar har kommit via: Frank Aylstock, 4336 Eureka Avenue, YORBA LINDA, CA 92686, USA.

Junction Softworks har upphört med all verksamhet för TI-99/4A, PB 91-4.28 gäller ej.

Asgard har upphört med sitt 80-kolumnskort EGI, PB 91-4.23 gäller ej. Man har slutat att ta emot nya prenumeranter på Asgard Reflections. Asgard fortsätter att sälja program.

Texaments, 53 Center Street, Patchogue, NY 11772, USA säljer:

Myarc HFDC	USD 200+porto	USD 45
Myarc dsk-contr	USD 140+porto	USD 20
Myarc Geneve	USD 420+porto	USD100
Myarc RS232/PIO	USD 100+porto	USD 20

Du kan få en kopia av FW 4.40 genom att sända skivor och frankerat svarskuvert till redaktören. 40-kolumnsversionen ryms på 3 st SS/SD medan extrafiler för 80-kolumn ryms på 2 st extra skivor SS/SD.

Jan Alexandersson, Springarvägen 5, 3 tr, S-142 61 TRANGSUND(08-7710569)

FW 4.40 (continued from page 3)

Oct/30/91 ... Back to programming with renewed vision !! Read/Write failure indicators added for DR file copies. DSKU notes handling revised and corrected. FWDOC files /DR40,1, /DR80,2, /PSRV updated. Dual 80/40 Editor, files ED80 & ED81, now windows correctly in 40 col mode. Cause of error recovery bug in DR narrowed to Myarc 80-Tk FDCs. ■

Redaktör: Jan Alexandersson
Medlemsregister: Claes Schibler
Tryckning av tidning: Åke Olsson
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 84-89 (per årgång)	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.

FUNNELWEB VERSION 4.40

by Tony McGovern, Australia

New Files (-READ-ME)

The FUNNELWEB package contains a number of files (50 files in all). The following is new compared to FW 4.31.

ML - a sample Multi-List file

FWDOC/SCLL - details of Low-Loader, Script-Loader, Assembly MAKE, and User and Multi-List files

ML80 - the 80-column Multi List program

FWDOC/PSRV - details of useful program services, pointers, and data available in the FW interface block.

Update Notes (-READ-ME)

Vn 4.40 is a further refinement and extension of the FUNNELWEB system, and retains full external compatibility with prior versions. Significant changes, apart from minor bug-fixes, from Vn 4.31 are

(i) A flashing cursor with auto accelerating repeat has replaced the static sprite underline cursor, with timing delays compensated for processor speed. GROM address setting now should support Module Library devices. More externally accessible program services and new documentation are provided.

(ii) DM-1000 files MG/MH and Disk-Patch file DP were dropped after Vn 4.30. They still can be used with Vn 4.40. Use of the LOAD only reload path from DM-1000 is no longer advisable.

(iii) Script-Load and the Assembler now support a multiple file MAKE function.

(iv) Error indication for Duplicate DEFs and Unresolved REFs has been improved in the object loaders, and Script-Load now has a full error handler with extended Unresolved REF display.

(v) The 80 column Editor now supports a 40 column Edit mode. Program file checking has been dropped for extra speed from the Editor SDs as superfluous now DR is here, and the original TI RE bug is fixed. This has made room for --

(vi) Double-View in the 80-col Editor which allows page scrolled access to parts of one or more files from SD without further disk activity.

(vii) DiskReview now has some support for DSKU file comments, and file read in for View is faster.

(viii) In UL files the ‹esc› path has been modified to suit better the revised ‹esc› handling in the main program. It would be a good idea to transfer your existing lists on to the new template by Fetching the old file, Making Reserve of it, Fetching the new UL, eXchanging data, and then Saving back under the original filename.

(ix) A new class of Multi User List files has been introduced.

(x) Various other auxiliary system programs have been revised, in particular CF/CG, LDFW, CT8K/O, LL, SL, AS/AT, and ED/EE. Replace all files to be on the safe side.

Bugs and Mods (FWDOC/REPT)

May/30/91 ... First issue of Vn 4.40.

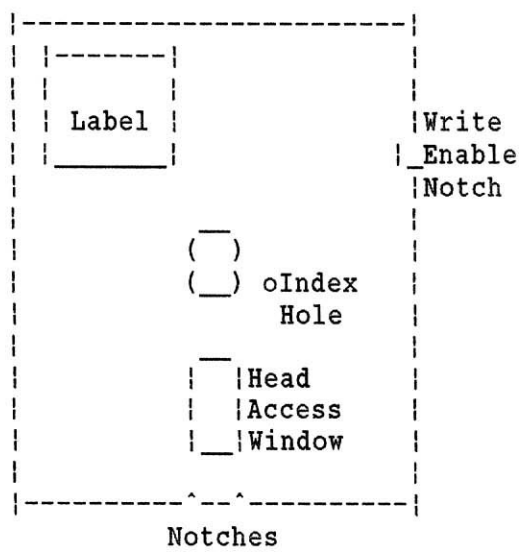
Jul/26/91 ... Fixes for LOAD's XB user list program loading, and for ED80 ‹P›Dir function from SD. FILENT routine in FW/LOAD now clears the last char in the window with ‹fn-3›. ML40 repaired. (continued on page 2)

SWEDLOW TI BITS * 11-13 *

by Jim Swedlow, USA

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

THE PARTS OF A DISK



QUOTES OF THE MONTH

"The game isn't over till it's over."
---Yogi (Lawrence Peter) Berra

"The computer 'doth make fools of us all."
---Weinberg

BELL COMPATIBLE?

Ever noticed that modem adds include a statement about Bell compatibility? This will give you an idea of what that means.

BELL 103A is the standard format for transmitting data by telephone at speeds of 300 baud or less.

BELL 202 is a standard format for transmitting data by telephone at 1,200 baud. Bell 202 format is half duplex only and has now largely been replaced by Bell 212A.

BELL 212A is the standard format for transmitting data by telephone at 1,200 baud.

WORD OF THE MONTH

BAUD: a unit that measures the speed of data transmission. One baud equals one bit per second. Or, 300 baud is 300 bits per second. That is why 1200 baud is four times faster than 300 baud. (Use baud only for symbole rate of transmission and use bit/s for information rate. The symbole rate could sometimes be less than the bitrate in bit/s. Ed. note)

MORE ACRONYMNS

Here are two more acronyms:

CD-ROM - "Compact Disk Read Only Memory" - using the same CD's that are becomming the rage for music to store data. One CD can hold an entire encyclopedia with space left over. The current price of the drives is high ($1000 or so) as are the few CD-ROM's on the market ($300 to $1000+). Prices will come down, however.

WORM - "Write Once Read Many" - the next generation for CD-ROM, these allow the user to write on CD disks once and then read many, many times. The drive has two laser devices, one to write and one to read.

MORE QUOTES

"PRICE is what you pay NOW;
COST is what you pay LATER."
---A Pilgrims' Pride Catalog

"You may not always get what you pay for but you never get what you don't pay for!"
---Ibid.

MAILING LABELS

As the United States Postal Service (USPS) uses more and more Optical Character Readers (OCR's), a problem is developing with computer generated mailing labels.

According to an article in a recent issue of InfoWorld, the USPS has purchased some 400 OCR's since 1984 and it recently signed a contract for over 400 more. These units can process mail at speeds up to 35,000 pieces an hour (the hand rate is about 900).

What's the mailing label problem? Seems that the OCR's have trouble reading mailing labels in compressed print or when the dots don't touch. The solution is to print your labels in Near Letter Quality or Emphasized Print and to use 10 to 12 characters per inch. If in doubt, check with your local post office.

TI LIVES!

I was looking thru the current issue of MICROTIMES and spotted the listing of user groups. Most orphan computers have one or two user groups in the greater Southern California area. A few have three groups.

One computer stood out with ten (count them, ten) user groups. None other than our 4A.

EXPANDING YOUR SYSTEM

I have had a couple of conversations recently about system expansion. There are so many products available today -- the Rave keyboard, RAM disks, the GENEVE -- that it is difficult to make decisions.

It is not my intent to tell you what to do or buy, rather I hope to give you some food for thought.

The first step in any expansion plan is to do a needs assessment. Does your current system meet your needs? Where does it fall short? TI Writer and Multiplan are quite powerful for some applications and inadequate for others. The 48 key keyboard that TI gave us has limitations but is workable. Perhaps you just want to be able to walk into a store again and buy some NEW software.

The next step is much harder. How much can you invest? As my boss once said, this is where the rubber meets the road. Your system may fall very short of your needs and wants, but if the dollars aren't there, they aren't there.

Just for discussion purposes, lets assume that you have a P box with RS232, 32K and one or more drives and that you can afford to spend $700 to $1000 on upgrading your system. How should you spend the money? Here are two different routes.

A TI SUPER SYSTEM: You could invest you money in upgrading your current TI system:

Item	Low	High
Print Spooler	$100	$100
RAM Disk	150	200
RAVE 99 Keyboard	185	200
Upgraded Disk Drives	100	300
Composite Color Monitor	150	200
Total	$685	$1000

PRINT SPOOLER: There are many options, including the CORCOMP Triple Tech Card. You will find a print spooler invaluable. Luckily, TI opted for a standard Centronics parallel interface. You plug your printer cable into the spooler and the spooler's cable into your printer. Then all those long delays during printing are gone and you have real multi-tasking! When you print something with the Formatter, for example, about a quarter of the way thru, your TI is done and ready for other things while your printer just clacks away. You will love it.

RAM DISK: If you have not seen one of these demonstrated, it is an electronic disk. To your system it looks just like a disk drive but many times faster. For example, you press the key for the Formatter and in one or two SECONDS you have the Formatter. You may have passed the point where computers seem like magic but a RAM disk will.

KEYBOARD: My two major complaints about my TI are the lack of an 80 column screen and the keyboard. Although we have yet to see a fully compatible 80 column card, RAVE 99

does sell a full size keyboard for about $200 and a device for attaching an IBM style keyboard for about $150 (comes out to around $200).

Look for other devices in the near future.

DISK DRIVES: If you have single sided drives, you can increase your storage capacity. One way is to upgrade to double sided drives. All three Disk Controller Cards (TI, MYRAC and CORCOMP) support double sided drives. Twice the space is a joy. If you have trouble with installing them, you may need help from an experienced user group member.

Another approach is to replace your TI Disk Controller card with one from MYRAC or CORCOMP. This will turn your drives into double density drives (ALL of the drives TI sold were single sided double density drives - the TI disk controller card didn't support double density). You may have limited compatibility problems as a few programs that access disk drives don't work with non-TI cards.

Jumping from SSSD to DSDD quadruples your disk storage space!

COLOR MONITOR: If you are still using a TV, you will be amazed at the improvement that you will get from a monitor. Crisper colors. Clarity from edge to edge and minimal distortion.

TWO WORDS ON PRICE: Shop around. As I was preparing for this article, I found quite a range in prices for the same item and items for the task at hand. Prices quoted here are middle of the range.

So, folks, there you have it - two ways to spend your $700 - $1000. Now, if only I could get my hands on the bucks!

WORD OF THE MONTH

ERGONOMICS: The science of tailoring an environment to suit the worker. It has become more than a mere catch phrase - it is now a major player in the struggle to win a profit.

MILLER GRAPHICS' NIGHT MISSION

I should have bought this long ago, but the money and the opportunity never came together.

As a game, it is a good one. The speed is amazing when you consider that it is 100% Extended Basic. This alone justifies the $20 price tag.

The real value, in my opinion, is the documentation. Sure, it has the game instructions, but it also includes three other sections.

First, it explains the program, line by line. You WILL learn new Extended BASIC programming techniques by reading this.

Second, it has a list of CALL LOAD's and CALL PEEK's, some of which are new.

Third, it has a very interesting section on 'The Power of AND'. This is a completely new programming device.

As a game, as a tutorial or as both, this is a wise purchase.

XB PROGRAMMING TIP

I was working on a program and I ran into something interesting.

I had a variable (A) that could only be either 9 or 7. I needed to separate the beginning of a string. If A was 9, I needed the first 5 characters in the string, otherwise I needed the first 4 characters.

My first approach was:

```
IF A=9 THEN N$=SEG$(A$,1,5)
ELSE N$=SEG$(A$,1,4)
```

That seemed bulky so I tried this:

```
N$=SEG$(A$,1,5+(A=7))
```

[This works because if A is 7, the expression (A=7) returns -1. Otherwise it returns 0. If this is not

clear, try this loop:

```
FOR A=6 TO 8 ::
PRINT A;A=7;5+(A=7) ::
NEXT A
```

Note that the parenthesis are only needed for 5+(A=7). Without them, your TI would compare 5+A to 7.]

Back to my tip. That was better, but I did a bit of fiddling and realized that 0.6 times 9 is 5.4 and 0.6 times 7 is 4.2. I tried this:

```
N$=SEG$(A$,1,INT(A*.6))
```

Then I wondered if the INT was needed. It turns out that SEG$ (like many TI functions) automatically rounds off.

How can you tell if a function rounds? Try using it in a loop:

```
FOR I=1 TO 2 STEP .1
PRINT I;SEG$("ABC",1,I)
NEXT I
```

My final expressions was:

```
N$=SEG$(A$,1,A*.6)
```

XMODEM

You may have heard of a transfer protocol called XMODEM and wondered what it is. If you use FAST-TERM or 4A TALK, you probably use it. The following should give you some idea of how it works.

When you communicate with another computer on phone lines thru modems, your data must travel thru the same voice phone lines that we use every day. Some connections are better than others. Most have noticeable static.

You brain, a computer whose power has never been equaled, can usually distinguish the 'data' (voice) from the 'noise' (static). It is almost impossible for your computer to make this judgement.

In the early days of data transfer, data was simply sent and the receiving computer had to do as good a job as it could to distinguish between data and noise. In a text, or DV80 file, this was not a major problem. If one character was bad you could easily find the problem and edit it.

With a memory image or Program file, however, one bad byte could render the entire file useless. Although editing is possible, it is very tricky.

In August 1977, Ward Christensen developed an error detection method he called MODEM2. It was also dubbed "Christensen" protocol or XMODEM.

It was very simple. Data is sent in blocks of 128 bytes. XMODEM adds up the values of all the characters in each block and compares that number with a total that is sent by the sending computer. If they don't agree, the receiving computer sends a code to the sending computer and the block is transmitted again.

In 1982, Ward Christensen and Chuck Forsberg released an enhancement called Cyclic Redundancy Checking (CRC). CRC does sequential division on each character in the block resulting is a significant improvement in error detection.

Both protocols continue to be called XMODEM. Although others have been developed, XMODEM is used by all major systems, including Compuserve. (Source: an article in FOGLIGHT)

TI WRITER TIP

Find String (FS) is a powerful tool for finding something in a document. Just hit FCTN 8 and then enter FS. Your TI Writer gives you this prompt:

FIND enter /string/ :

You enter your string and use the slash as limiters. If you want to find the word "John", you would enter /John/. If you wanted to find John only when it is used as the last word in a sentence, you would enter /John./.

Should the "John" you find not be the one you wanted, you would go

back to command mode and enter FS again. You will find /John./ still there. You just press enter and the search resumes.

Lets say, however, that now you want to find the word "Mo". But /John./ is on your screen. You could delete /John./. You could type in Mo but then you would have this:
/Mo/n./

Need you worry about the text after the second slash? No. Your TI Writer only searches for the information between the first and second slash. It ignores everything to the right of the second slash.

You will have a problem with that if you use Replace String, but that is another story.

THE PAPERLESS OFFICE

One of the things that futurists often project is the paperless office. Everything would be done on computers so paper would virtually disappear.

Not necessarily so. According to an article in a recent issue of 'The Office', the demand for paper has been increasing at the rate of 5% to 8% a year. Growth is expected to continue at that rate.

Cited reasons include the continuing shift from a production to a service economy and the fact that computers generate reams of paper. Also noted were the need to generate hard copies for filing and the proliferation of photocopy machines.

Enjoy. ■

BATTLE STAR – SPEL FÖR XB

Använd tangenterna E-X respektive S-D för att skjuta ned de fientliga vita rymdskeppen och de röda robotarna. Du måste ha fyra fingar samtidigt på de fyra tangenterna. Använd ena handen för upp-ned och den andra för vänster-höger. Robotarna får ej träffa dig.

```
100 REM *************
110 REM *BATTLE STAR*
120 REM *************
130 REM BY W.K.BALTHROP
140 REM 99'ER 82-06 XB
180 RANDOMIZE :: DIR=1 :: CA
LL CLEAR
190 CALL COLOR(9,7,1,10,6,1)
:: CALL SCREEN(2)
200 CALL CHAR(96,"0000000000
0707071818183C7EFFDB99000000
0000E0E0E0070E1CFFFF1C0E07")
220 CALL CHAR(104,"18423C999
93C4218",101,"E07038FFFF3870
E0070707")
230 CALL CHAR(107,"104628240
A923044",103,"99DBFF7E3C1818
18",100,"E0E0E")
250 CALL CHAR(112,"30787C477
C7830001010386CEEEE7C000C1E3
EE23E1E0C00007CEEEE6C38101")
270 CALL CHAR(116,"101038FE3
81010000000183CFF7E2442",105
,"1818181818181818000000FFFF
")
290 FOR COL=1 TO 12 :: CALL
COLOR(COL,16,1):: NEXT COL
300 L=100 :: S=5 :: SC,SA1,S
B1,SA2,SB2,SA3,SB3,SA4,SB4,T
=0
310 GOSUB 350
320 GOSUB 390 :: GOSUB 650
330 L=L-.5 :: IF L<1 THEN L=
1
340 DISPLAY AT(24,3):SC :: G
OTO 320
350 CALL SPRITE(#10,96,16,81
,113,0,0,#11,97,16,81,121,0,
0,#12,98,16,81,129,0,0)
360 CALL SPRITE(#13,99,16,89
,113,0,0,#14,104,7,89,121,0,
0,#15,101,16,89,129,0,0)
370 CALL SPRITE(#16,102,16,9
7,113,0,0,#17,103,16,97,121,
0,0,#18,100,16,97,129,0,0)
380 RETURN
390 CALL KEY(3,K,S):: IF S=0
 THEN RETURN
400 IF K=69 THEN 450 ELSE IF
 K=83 THEN 500 ELSE IF K=88
THEN 550 ELSE IF K=68 THEN 6
00
440 RETURN
450 IF SA1=0 AND SB1=0 THEN
```

```
CALL VCHAR(1,16,105,10):: CA
LL SOUND(10,800,0):: CALL VC
HAR(1,16,32,10):: SC=SC-10 :
: RETURN
460 IF SB1=0 THEN CALL VCHAR
(2,16,105,9):: CALL SOUND(50
0,110,2,-5,2):: CALL VCHAR(2
,16,32,9):: SC=SC+50 :: SA1=
0 :: RETURN
470 CALL POSITION(#1,P1,P2):
: IF P1>76 THEN 840
480 P1=INT(P1/8)+1 :: CALL V
CHAR(P1,16,105,10-P1):: CALL
 SOUND(200,110,10,-5,8):: CA
LL VCHAR(P1,16,32,10-P1)
490 CALL DELSPRITE(#1):: SC=
SC+20 :: SB1=0 :: RETURN
500 IF SA2=0 AND SB2=0 THEN
CALL HCHAR(12,1,106,14):: CA
LL SOUND(10,800,0):: CALL HC
HAR(12,1,32,14):: SC=SC-10 :
: RETURN
510 IF SB2=0 THEN CALL HCHAR
(12,3,106,12):: CALL SOUND(5
00,110,2,-5,2):: CALL HCHAR(
12,3,32,12):: SC=SC+50 :: SA
2=0 :: RETURN
520 CALL POSITION(#2,P1,P2):
: IF P2>86 THEN 840
530 P2=INT(P2/8)+1 :: CALL H
CHAR(12,P2,106,15-P2):: CALL
 SOUND(200,110,10,-5,8):: CA
LL HCHAR(12,P2,32,15-P2)
540 CALL DELSPRITE(#2):: SC=
SC+20 :: SB2=0 :: RETURN
550 IF SA3=0 AND SB3=0 THEN
CALL VCHAR(14,16,105,10):: C
ALL SOUND(10,800,0):: CALL V
CHAR(14,16,32,10):: SC=SC-10
 :: RETURN
560 IF SB3=0 THEN CALL VCHAR
(14,16,105,10):: CALL SOUND(
500,110,2,-5,2):: CALL VCHAR
(14,16,32,10):: SC=SC+50 ::
SA3=0 :: RETURN
570 CALL POSITION(#3,P1,P2):
: IF P1<110 AND P1>0 THEN 84
0
580 P1=INT(P1/8)+1 :: CALL V
CHAR(14,16,105,P1-14):: CALL
 SOUND(200,110,10,-5,8):: CA
LL VCHAR(14,16,32,P1-14)
590 CALL DELSPRITE(#3):: SC=
SC+20 :: SB3=0 :: RETURN
600 IF SA4=0 AND SB4=0 THEN
CALL HCHAR(12,18,106,14):: C
ALL SOUND(10,800,0):: CALL H
CHAR(12,18,32,14):: SC=SC-10
 :: RETURN
610 IF SB4=0 THEN CALL HCHAR
(12,18,106,13):: CALL SOUND(
500,110,2,-5,2):: CALL HCHAR
(12,18,32,13):: SC=SC+50 ::
SA4=0 :: RETURN
620 CALL POSITION(#4,P1,P2):
: IF P8<142 AND P8>0 THEN 84
0
630 P2=INT(P2/8):: CALL HCHA
R(12,18,106,P2-15):: CALL SO
UND(200,110,10,-5,8):: CALL
HCHAR(12,18,32,P2-15)
640 CALL DELSPRITE(#4):: SC=
SC+20 :: SB4=0 :: RETURN
650 IF SB1=0 THEN P1,P2=0 EL
SE CALL POSITION(#1,P1,P2)
660 IF SB2=0 THEN P3,P4=0 EL
SE CALL POSITION(#2,P3,P4)
670 IF SB3=0 THEN P5,P6=0 EL
SE CALL POSITION(#3,P5,P6)
680 IF SB4=0 THEN P7,P8=0 EL
SE CALL POSITION(#4,P7,P8)
690 IF P1>76 OR P4>86 OR(P5<
110 AND P5>0)OR(P8<142 AND P
8>0)THEN 840
700 NS=INT(RND*L):: IF NS>10
 THEN RETURN
710 NS=INT(RND*4)+1 :: ON NS
 GOTO 730,760,790,820
720 IF SA1=1 AND SB1=1 THEN
RETURN
730 CALL HCHAR(2,16,115):: S
A1=1 :: IF L<80 AND SB1=0 TH
EN CALL SPRITE(#1,116,7,17,1
20,11-(L/10),0):: SB1=1
740 RETURN
750 IF SA2=1 AND SB2=1 THEN
RETURN
760 CALL HCHAR(12,3,112):: S
A2=1 :: IF L<80 AND SB2=0 TH
EN CALL SPRITE(#2,116,7,88,1
7,0,11-(L/10)):: SB2=1
770 RETURN
780 IF SA3=1 AND SB3=1 THEN
RETURN
790 CALL HCHAR(23,16,113)::
SA3=1 :: IF L<80 AND SB3=0 T
HEN CALL SPRITE(#3,116,7,175
,120,-11+(L/10),0):: SB3=1
800 RETURN
810 IF SA4=1 AND SB4=1 THEN
RETURN
820 CALL HCHAR(12,30,114)::
SA4=1 :: IF L<80 AND SB4=0 T
HEN CALL SPRITE(#4,116,7,88,
216,0,-11+(L/10)):: SB4=1
830 RETURN
840 CALL DELSPRITE(#1,#2,#3,
#4):: CALL SOUND(2000,110,2,
220,2,1000,30,-4,2)
850 FOR BUB=10 TO 18 :: CALL
 MOTION(#BUB,INT(RND*40)-20,
INT(RND*40)-20):: CALL PATTE
RN(#BUB,107):: NEXT BUB
860 CALL SOUND(1000,110,2,22
```

```
0,2,110,2,-5,2):: CALL SOUND
(1,40000,30)
870 CALL DELSPRITE(ALL):: CA
LL CLEAR
880 DISPLAY AT(12,7):"YOUR S
CORE IS": :TAB(10);SC
900 DISPLAY AT(22,1):"DO YOU
 WISH TO PLAY AGAIN?  (Y/N)"
910 ACCEPT AT(23,8)VALIDATE(
"YN"):ANS$ :: IF ANS$="N" TH
EN 950
920 CALL CLEAR :: GOSUB 350
:: SC=0 :: L=100
930 SB1,SB2,SB3,SB4,P1,P2,P3
,P4,P5,P6,P7,P8=0
940 RETURN
950 END
```

■

DEBUGGING EXTENDED BASIC

by Jim Peterson, Tigercub, USA

When you have finished writing a program, the next thing you should do is to run it. And, very probably, it will crash!

Don't be discouraged. It happens to the very best of programmers, very often.

So, the next thing to do is to debug it. And you are lucky that you are using a computer that helps you to debug better than some that cost ten times as much.

There are really three types of bugs. The first type will prevent the program from running at all - it will crash with an error message. The second type will allow the program to run, but will give the wrong results.

And the third type, which is not really a bug but might be mistaken for one, results from trying to run a perfectly good program with the wrong hardware, or with faulty hardware. As for instance, trying to run a Basic program, which uses character sets 15 and 16, in Extended Basic.

First, let's consider the first type. The smart little TI computer makes three separate checks to be sure your program is correct. First, when you key in a program line and hit the Enter key, it looks to see if there is anything it can't understand - such as a misspelled command or an unmatched quotation mark. If so, it will tell you so, most likely by SYNTAX ERROR, and refuse to accept the line.

Next, when you tell it to RUN the program, it first takes a quick look through the entire program, to find any combination of commands that it will not be able to perform. This is when it may crash with an error message telling you, for instance, that you have a NEXT without a matching FOR, or vice versa.

And finally, while it is actually running and comes to something that it just can't do, it will crash and give you an error message - probably because a variable has been given a value that cannot be used, such as a CALL HCHAR(R,C,32) when R happens to equal 0.

The TI has a wide variety of error messages to tell you when you did something wrong, what you did wrong, and where you did it wrong. But, it can be fooled! For instance, try to enter this program line (note the missing quotation mark).

```
100 PRINT "Program must be s
aved in:"merge format."
```

And, sometimes you may be told that you have a STRING-NUMBER MISMATCH when there is no string involved, because the computer has

tried to read a garbled statement as a string.

Also, the line number given in the error message is the line where the computer found it impossible to run the program; that line may actually be correct but the variables at that point may contain bad values due to an error in some previous line.

If the error occurs in a program line which consists of several statements, and you cannot spot the error, you may have to break the line into individual single-statement lines. This is the easiest way to do that - Be sure the line numbers are sequenced far enough apart. Bring the problem line to the screen, put a ! just before the first ::, and enter it. Bring it back to the screen with FCTN 8, retype the line number 1 higher, use FCTN 1 to delete the first statement and the ! and ::, put a ! before the first ::, and continue. Then, when you have solved the bug, just delete the ! from the original line and delete all the temporary lines.

Pages 212-215 of your Extended Basic manual list almost all the error codes, and almost all the causes of each one - it will pay you to consult these pages rather than guessing what is wrong.

You may create some really bad bugs when you try to modify a program that was written by someone else - especially if you add any new variable names or CALLs to the program. Your new variable might be one that is already being used in the program for something else, perhaps in a subscripted array. I have noticed that programmers rarely use @ in a variable name, so I always tack it onto the end of any variable that I add to a program.

Also, the program that you are modifying may have ON ERROR routines, or a prescan, already built in. The ON ERROR routine was intended to take care of a different problem than the one you create, so it could lead you far astray - you had better delete that ON ERROR statement until you are through modifying.

The prescan had better be the subject of another lesson, but if the program has an odd-looking command !@P- up near the front somewhere, it has a prescan built in. And if so, if you add a new variable name or use a CALL that isn't in the program, you will get a SYNTAX ERROR even though there is no error. One way to solve this is to insert a line with !@P+ just before the problem line, and another with !@P- right after it.

When a program runs, even though it crashes or is stopped by FCTN 4 or a BREAK, the values assigned by the program to variables up to that point will remain in memory until you RUN again, or make a change to the program, or clear the memory with NEW. This can be very useful. For instance, if the program crashes with BAD VALUE IN 680, and you bring line 680 to the screen and find it reads
CALL HCHAR(R,C,CH)
just type PRINT R;C;CH and you will get the values of R, C and CH at the time of the crash. You will find that R is less than 1 or more than 24, or C is less than 1 or more than 32, or CH is out of range.

In Extended Basic, you can even enter and run a multi-statement line in immediate mode (that is, without a line number), if no refer-

ence is made to a line number. So, you can dump the curent contents of an array to the screen by
FOR J=1 TO 100::PRINT A(J);::
: NEXT J - or you can even open a disk file or a printer to dump it to.

You can also test a program by assigning a value to a variable from the immediate mode. If you BREAK a program, enter A=100 and then enter CON, the program will continue from where it stopped but A will have a value of 100.

You can temporarily stop a program at any time with FCTN 4, of course (the manual says SHIFT C, but it was written for the old 99/4), and restart it from that point with CON. Or you can insert a temporary line at any point, such as 971 BREAK if you want a break after line 970. Or, you can put a line at the beginning of the program listing the line numbers before which you want breaks to occur, such as 1 BREAK 960,970,980
Note that in this case the program breaks just BEFORE those listed line numbers.
You can also use BREAK followed by one or more line numbers as a command in the immediate mode.
The problem with using BREAK and CON is that BREAK upsets your screen display format, resets redefined characters and colors to the default, and deletes sprites. So, it is sometimes better to trace the assignment of values to your variables by adding a temporary line to DISPLAY AT their values on some unused part of the screen. If you want to trace them through several statements, it will be better to GOSUB to a DISPLAY AT. And if you need to slow up the resulting display, just add a CALL KEY routine to the subroutine.

Sometimes, your program will appear to be not flowing through the sequence of lines you intended (perhaps because it dropped out of an IF statement to the next line!) and you will want to trace the line number flow. This can be done with TRACE, either as a command from the immediate mode or as a program statement, which will cause each line number to print to the screen as it is executed. If used as a command, it will trace everything from the beginning of the program, so it is usually better to insert a temporary line with TRACE at the point where you really want to start. Once you have implemented TRACE, the only way to get rid of it is with UNTRACE.
TRACE has its limitations because it can't tell you what is going on within a multi-statement line, and it will certainly mess up any screen display. Sometimes it is better to insert temporary program lines to display line numbers. I use CALL TRACE( ) with the line number between the parentheses, and a subprogram after everything else
30000 SUB TRACE(X)::DISPLAY
AT(24,1):X :: SUBEND

Some programmers use ON ERROR combined with CALL ERR as a debugging tool, but I can't tell you much about that because I have never used it. ON ERROR can give more trouble than help if not used very carefully, and I cannot see that CALL ERR gives any information not available by other means.

Sometimes you can debug a line by simply retyping it. It is only very rarely that the computer is actually interpreting a line differently than it appears on the screen, but retyping may result in correcting a typo error that you just could

not see. In fact, most bugs turn out to be very simple errors.

When you are debugging a string-handling routine, don't take it for granted that a string is really as it appears on the screen - it may have invisible characters at one or both ends. Try PRINT LEN(M$) to see if it contains more characters than are showing; or PRINT "*"&M$&"*" to see if any blanks appear between the asterisks and the string.

There is no standard way to debug a program. Each problem presents a challenge to figure out what is going wrong, to devise a test to find out what is really happening.

Don't debug by experimenting, by changing variable values just to see what will happen, etc. Even if you succeed, you will not have learned what was wrong so you will not have learned anything - and if your program contains lines that you didn't understand when you wrote them, you will have real problems if you ever try to modify the program. (I speak from experience!)■

```
100 ! SILENTNITE, BUG NEWS
103 CALL CLEAR
104 DISPLAY AT(4,8):"SILENT
NIGHT"
105 DISPLAY AT(8,0):"Graphic
s: B August BUG 12/87": :"Mu
sic : D Weaver TIRUG 10/87"
106 DISPLAY AT(11,2):"'Orkes
tration' and Doctoring ":"
    Earl Raguse 2/88"
107 DISPLAY AT(18,3):"PRESS
ANY KEY TO PROCEED"
108 CALL KEY(3,K,S):: IF S=0
 THEN 108
109 CALL CHARSET
110 RANDOMIZE :: CALL CLEAR
:: CALL SCREEN(1):: CALL CHA
R(48,"80C0E0F0F8FCFEFF",49,"
0103070F1F3F7FFF",56,"000000
001C46FF42")
120 FOR CH=71 TO 82 :: READ
D$ :: CALL CHAR(CH,D$):: NEX
T CH :: FOR SET=2 TO 7 :: RE
AD FC,BC :: CALL COLOR(SET,F
C,BC):: NEXT SET
130 CALL HCHAR(20,4,71,2)::
CALL HCHAR(21,4,71,3):: CALL
 HCHAR(22,1,40,96):: FOR D=1
 TO 45 :: READ R,C,CH :: CAL
L HCHAR(R,C,CH):: NEXT D
140 COL=COL+1 :: CH=INT(RND*
7)+72 :: CALL GCHAR(21,COL,C
OI):: IF COI<>32 THEN 140 EL
SE CALL HCHAR(21,COL,CH):: I
F COL<>32 THEN 140 ELSE 500
150 END
160 DATA FFFFE7E7E7E7E7FFFF,
103038383C7C7E18,000018183C3
C7E18,0040387C3C181091,00000
000001818BB,00001018383C7D13
170 DATA 183C7E3C181818,0000
00000044EEEF,1,1,071E3C7C78F
8F0F0,F0F0F8787C3C1E07,16,16
,15,1,7,1,5,11,13,1,16,1
180 DATA 5,9,81,6,9,82,19,4,
49,19,5,48,20,6,48,20,20,49,
20,21,48,21,8,56,21,20,71,21
,21,71,21,23,56
190 DATA 2,5,80,2,32,80,3,12
,80,4,10,80,4,15,80,4,17,80,
4,21,80,5,3,80,5,7,80,5,17,8
0
200 DATA 6,16,80,6,27,80,7,4
,80,7,7,80,7,21,80,7,28,80,8
,12,80,8,25,80,9,15,80,10,18
,80,10,31,80
210 DATA 11,12,80,12,5,80,12
,20,80,14,7,80,15,25,80,15,2
,80,16,1,80,16,10,80
220 DATA 17,7,80,17,16,80,17
,20,80,18,4,80,18,30,80
500 ! From TIRUG, by David W
eaver, doctored by E Raguse
510 RESTORE 590
540 FOR I=1 TO 46
550 READ X,Y
560 DUR=200
570 CALL SOUND(X*DUR,Y/2,0,Y
/1.99,8,Y/2.01,6)
580 NEXT I
585 FOR T=1 TO 700 :: NEXT T
 :: CALL CLEAR :: END
590 DATA 3,784,1,880,2,784,6
,659,3,784,1,880,2,784,6,659
600 DATA 4,1175,2,1175,6,988
,4,1047,2,1047,6,784,4,880,2
,880,3,1047,1,988,2,880
610 DATA 3,784,1,880,2,784,6
,659,4,880,2,880,3,1047,1,98
8,2,880
620 DATA 3,784,1,880,2,784,6
,659,4,1175,2,1175,2,1397,2,
1175,2,988
630 DATA 6,1047,6,1319,2,104
7,2,784,2,659,2,784,2,698,2,
587,12,523
```
■

GRAPHIC1 FÖR ASSEMBLER

av Jan Alexandersson

9918A/9929A FÖR TI-99/4A

TI-99/4A har en videoprocessor 9918A/9929A som har fyra olika grafikmoder:

	MAX	MINNE	---TECKEN---	
MODE	FÄRG	KBYTES	FÄRG	PUNKTER
Graphic1	16	4	2	8x8
Graphic2	16	16	16	8x8
Text1	2	4	2	6x8
Multicolor	16	4	16	2x2

Graphic1 delar upp skärmen i rutor med 24 rader och 32 kolumner. Varje sådan ruta kan fyllas med ett av 256 möjliga grafiska tecken. Varje teckenruta består av 8x8 punkter. Ett sådant tecken kan ha två färger av 16 möjliga. Åtta tecken med närliggande ASCII-nummer måste ha samma färger. Dessutom kan man ha upp till 32 olika sprites som lägges ovanpå grafikskärmen.

Du måste initiera videoprocessorns 8 olika register (#0-#7) för att kunna använda Graphic1 på följande sätt:

```
#0  >00         binärt 0000 0000
#1  >E0         binärt 1110 0000
#2  värdet x >400 = Screen Table
#3  värdet x >40  = Color Table
#4  värdet x >800 = Pattern Table
#5  värdet x >80  = Sprite Attribute
#6  värdet x >800 = Sprite Pattern
#7  högra hexdelen= Screen Color
```

Läs mera om VDP-register i PB 89-4.

TABELL 99/4A	REG	BYTES	PLATSER
SCREEN	2	768	16
COLOR	3	32	256
PATTERN	4	2048	8
SPRITE ATTRIBUTE	5	128	128
SPRITE PATTERN	6	1024	8
TOTALT		4000	4

Skärmtabellen kan placeras på 16 olika ställen genom att >00 till >0F skrivs till VDP-register #2. Antalet möjliga placeringar minskas eftersom de övriga tabellerna måste finnas någonstans samt av att VDP-RAM även används som diskbuffert m.m.

```
************************
* Graphic1 DEMO PROGRAM 1
* for 99/4A
* by Jan Alexandersson
* 1991-08-25
************************

       DEF  START

WS     BSS  >20
RTN    DATA 0

START  MOV  R11,@RTN     save return
       LWPI WS
       BL   @G14A        graphic1

****** print message on the screen
       LI   R0,5*32+8+SCRTAB
       LI   R1,TEXT1
       LI   R2,16
       BLWP @VMBW

       LI   R0,23*32+6+SCRTAB
       LI   R1,TEXT2
       LI   R2,19
       BLWP @VMBW

****** wait for key before return
       LI   R0,>2000
KEYLP  BLWP @KSCAN       key scan
       MOVB @STATUS,R1
       COC  R0,R1        test EQ bit
       JNE  KEYLP

       CLR  R1
       MOVB R1,@STATUS
       LWPI GPLWS
       MOV  @RTN,R11
       RT                return

TEXT1  TEXT 'Test of GRAPHIC1'
TEXT2  TEXT 'PRESS ENTER TO EXIT'
       EVEN

       COPY "DSK2.G1-TAB-4A"
       COPY "DSK2.G14A"
       COPY "DSK2.BLKVDP"
       COPY "DSK2.VDP-UTIL"   PB91-2
       COPY "DSK2.KSCAN"
       COPY "DSK2.GPLLNK"     PB91-2

       END

****************************
* G1-TAB-4A tables for 99/4A
* in Graphic1 mode
****************************
```

```
SCREEN EQU  0
SCREND EQU  24*32
PATTER EQU  1
PATEND EQU  >800
COLOR  EQU  >E
COLEND EQU  >20
ATTRIB EQU  6
SCRCOL EQU  3          green screen
COLCHR EQU  >1300      black on green

SCRTAB EQU  SCREEN*>400
PATTAB EQU  PATTER*>800
COLTAB EQU  COLOR*>40
ATTADR EQU  ATTRIB*>80


***********************
* G14A initialize
* Graphic1 for 99/4A
* by Jan Alexandersson
* 1991-08-25
***********************

FAC    EQU  >834A
STATUS EQU  >837C
KEYBRD EQU  >8374

CRIGHT DATA >3C42,>99A1,>A199,>423C
CURSOR DATA >7070,>7070,>7070,>7070
       DATA >007E,>4242,>4242,>7E00

G14A   MOV  R11,R10     save return
       LI   R0,>01A0    screen off
       BLWP @VWTR
       CLR  R0          VDP-reg #0
       BLWP @VWTR
****** screen table
       LI   R0,>0200+SCREEN
       BLWP @VWTR
****** colour table
       LI   R0,>0300+COLOR
       BLWP @VWTR
****** pattern table
       LI   R0,>0400+PATTER
       BLWP @VWTR
****** sprite attribute table
       LI   R0,>0500+ATTRIB
       BLWP @VWTR
****** screen colour
       LI   R0,>0700+SCRCOL
       BLWP @VWTR
****** standard keyboard = 0
       CLR  R0
       MOVB R0,@KEYBRD
****** number of moving sprites = 0
       MOVB R0,@>837A

****** clear screen table
       BL   @BLKVDP
       DATA SCRTAB,>2000,SCREND

****** D0 removes standing sprites
       LI   R0,ATTADR
       LI   R1,>D000
       BLWP @VSBW

***** colour table initialize
       BL   @BLKVDP
       DATA COLTAB,COLCHR,COLEND

****** clear pattern table
       BL   @BLKVDP
       DATA PATTAB,0,PATEND

****** normal char 32-95
       LI   R0,PATTAB+>100
       MOV  R0,@FAC
       CLR  R1
       MOVB R1,@STATUS
       BLWP @GPLLNK
       DATA >18

****** lower char 96-127
       LI   R0,PATTAB+>300
       MOV  R0,@FAC
       MOVB R1,@STATUS
       BLWP @GPLLNK
       DATA >4A

****** char 10 = copyright
       LI   R0,PATTAB+>50
       LI   R1,CRIGHT
       LI   R2,8
       BLWP @VMBW

****** char 30-31 = cursor
       LI   R0,PATTAB+>F0
       LI   R1,CURSOR
       LI   R2,16
       BLWP @VMBW

       LI   R0,>01E0    screen on
       BLWP @VWTR
       SWPB R0
       MOVB R0,@>83D4   for KSCAN

       B    *R10        return


***********************************
* BLKVDP by Mack McCormick
* revised by Jan Alexandersson
* writes data to a block of VDP RAM
* see PB 91-3 page 18
***********************************
VDPWA  EQU  >8C02
VDPWD  EQU  >8C00

BLKVDP MOV  *R11+,R0
       ORI  R0,>4000
       MOV  *R11+,R1
       MOV  *R11+,R2
       SWPB R0
```

```
       MOVB R0,@VDPWA
       SWPB R0
       MOVB R0,@VDPWA
BLKLP  MOVB R1,@VDPWD
       DEC  R2
       JNE  BLKLP
       RT


*******************************
* KSCAN utility disassembled *
* by J.Peter Hoddie 1987     *
* use it together with       *
* VDP-UTIL from PB 91-2 p.15 *
*******************************

KSCAN  DATA UTILWS,KSCAN0

KSCAN0 LWPI >83E0
       MOV  R11,@AK
       BL   @>000E
       LWPI UTILWS
       MOV  R11,@>83F6
       RTWP


*******************************
* Graphic1 DEMO PROGRAM 2
* BIG CHAR from power up screen
* by Jan Alexandersson
* 1991-08-25
*******************************

       DEF  START

       REF  VMBW,VSBW,VWTR,GPLLNK
       REF  KSCAN,GPLWS

       COPY "DSK2.G1-TAB-4A"
       COPY "DSK2.G14A"
       COPY "DSK2.BLKVDP"

WS     BSS  >20
RTN    DATA 0

START  MOV  R11,@RTN     save return
       LWPI WS
       BL   @G14A        graphic1

****** big char 160-223 initialize
       LI   R0,PATTAB+>500
       MOV  R0,@FAC
       CLR  R1
       MOVB R1,@STATUS
       BLWP @GPLLNK
       DATA >16

****** print character set 0-255
       CLR  R1
       LI   R0,5*32+SCRTAB
LOOP   BLWP @VSBW
       INC  R0
       AI   R1,>0100
       JNE  LOOP

****** print message on the screen
       LI   R0,23*32+6+SCRTAB
       LI   R1,TEXT
       LI   R2,19
       BLWP @VMBW

****** wait for key before return
       LI   R0,>2000
KEYLP  BLWP @KSCAN      key scan
       MOVB @STATUS,R1
       COC  R0,R1       test EQ bit
       JNE  KEYLP

       CLR  R1
       MOVB R1,@STATUS
       LWPI GPLWS
       MOV  @RTN,R11
       RT               return

TEXT   TEXT 'PRESS ENTER TO EXIT'

       END
```

9938 FÖR 80-KOLUMNSKORT

Med videoprocessorn 9938 ska alla VDP-register #0 - #7 initieras som för TI-99/4A ovan. Även VDP-register #1 bör initieras med >E0 även om manualen till 9938 anger >06. Detta är nödvändigt om programmet även ska kunna köras med 9918A/9929A. Jag har inte upptäckt några problem med mitt AVPC-kort när jag använder >E0. Dessutom bör följande register initieras:

#8 >88 aktiverar musen

1	0	0	0	1	0	0	0
MS	0	TP	0	VR	0	SPD	0

MS = Mouse enable
TP = Color code 0 palette enable
VR = 64 kbit Video RAM
SPD= Sprite disabled

#9 >00 ger 60 Hz skärmväxling

0	0	0	0	0	0	0	0
LN	0	0	0	IL	0	NT	0

LN = Linjer 212 (26,5 rader)
IL = Interlace
NT = 50 Hz skärmväxling

#10 Extended Color Table
#11 Extended Sprite Attribute Table
#14 värdet x >4000 är Base Address

VDP-register #10 och #11 innehåller de mest signifikanta bitarna för

adressen till respektive tabell eftersom det behövs 2 bytes för att skriva önskat värde.

TABELL 9938	REG	BYTES	PLATSER
SCREEN	2	864	128
COLOR	10+3	32	2048
PATTERN	4	2048	64
SPRITE ATTRIBUTE	11+5	128	1024
SPRITE PATTERN	6	1024	64
TOTALT		4096	32
BASE	14	16384	8

I vårt exempel för 9938 används VDP-minne 65536 och uppåt. Värdet för Color Table är >40E (decimalt 1038). Detta skrivs som >4 till #10 och >E till #3. På motsvarande sätt skrivs värdet >208 för Sprite Attribute (>2 till #11 och >8 till #5). Man kan endast adressera 0 till >3FFF via de normala VDP-portarna. Högre adresser kommer man åt genom att skriva Base Address till VDP-register #14. Det finns totalt 8 olika bankar som kan kopplas in när du skriver eller läser adressen. Det finns inget som hindrar att du sprider ut dina tabeller på olika Base Address. Kom bara ihåg att skifta värdet i VDP-register #14 före anropet av VDP-minne.

Man kan även använda VDP-register #17 (hex >11) om man behöver skriva värden till ett större antal VDP-register som ligger i nummerföljd med hjälp av VDPID (VDP Indirect Data) på adress >8C06 (kallas port #3 i manualen). Skriv första registernumret till VDP-register #17 på normalt sätt med BLWP @VWTR. Du kan sedan skriva data till VDPID. Registernumret stegas fram med auto-increment vid upprepade skrivningar.

```
*************************
* Graphic1 DEMO PROGRAM 3
* for 9938
* use VDP RAM at 65536
* by Jan Alexandersson
* 1991-08-25
*************************

       DEF  START

       REF  VMBW,VSBW,VWTR,GPLLNK
       REF  KSCAN,GPLWS

       COPY "DSK2.G1-TAB-38"
       COPY "DSK2.G138"
       COPY "DSK2.G14A"
       COPY "DSK2.BLKVDP"

WS     BSS  >20
RTN    DATA 0

START  MOV  R11,@RTN    save return
       LWPI WS

       BL   @G138       graphic1

****** fill the screen with @
       BL   @BLKVDP
       DATA SCRTAB,>4000,SCREND

****** print message on the screen
       LI   R0,5*32+6+SCRTAB
       LI   R1,TEXT
       LI   R2,20
       BLWP @VMBW

****** wait for key before return
       LI   R0,>2000
KEYLP  BLWP @KSCAN      key scan
       MOVB @STATUS,R1
       COC  R0,R1       test EQ bit
       JNE  KEYLP

****** restore to VDP RAM 0-16383
VDPID  EQU  >8C06       indirect data
       LI   R0,>0E00    base address
       BLWP @VWTR
       LI   R0,>1102    #17 at #2
       BLWP @VWTR

       LI   R0,RSTORE
       LI   R1,10
RSTLP  MOVB *R0+,@VDPID restore reg
       DEC  R1
       JNE  RSTLP

       CLR  R1
       MOVB R1,@STATUS
       LWPI GPLWS
       MOV  @RTN,R11
       RT               return

TEXT   TEXT 'Graphic1 - 26.5 rows'

RSTORE DATA >000E,>0106,>00F5 #2-#7
       DATA >8800,>0000      #8-#11

       END

*****************************
* G1-TAB-38 tables for 9938
* in Graphic1 with 212 lines
* at VDP address >10000 = dec 65536
*****************************
```

```
SCREEN EQU  64
SCREND EQU  27*32
PATTER EQU  33
PATEND EQU  >800
COLOR  EQU  >E
COLEXT EQU  4
COLEND EQU  >20
ATTRIB EQU  8
ATTEXT EQU  2
SCRCOL EQU  3          green screen
COLCHR EQU  >1300      black on green
BASE   EQU  4

SCRTAB EQU  SCREEN-64*>400
PATTAB EQU  PATTER-32*>800
COLTAB EQU  COLOR*>40
ATTADR EQU  ATTRIB*>80


****************************
* G138 initialize for 9938
* of Graphic1
* by Jan Alexandersson
* 1991-08-11
****************************

G138   MOV  R11,R9      save return
****** base address
       LI   R0,>0E00+BASE
       BLWP @VWTR
       BL   @G14A       99/4A init
       LI   R0,>01A0    screen off
       BLWP @VWTR
       LI   R0,>0888    mouse enable
       BLWP @VWTR
       LI   R0,>0980    212pix,60Hz
       BLWP @VWTR
****** extended color table
       LI   R0,>0A00+COLEXT
       BLWP @VWTR
****** extended attribute table
       LI   R0,>0B00+ATTEXT
       BLWP @VWTR
       LI   R0,>01E0    screen on
       BLWP @VWTR
       B    *R9         return
```

■

RUMORS — V9978

by Don Jones, Chicago UG, USA

The following "rumor" comes from our fine SysOp, Mike ("Wetsuit Warrior") Maksimik: " Here are some new rumours on the airwaves: In the works at Yamaha is the new V9978 video chip, for the advanced MSX2 standard. Mostly pin compatible with the 9938 and 9958 it is said to provide 1024 by 848 pixel resolution, at 16 colors. This setup will require more video RAM, (at least twice as much) and can be accomplished by adding another address bit to the existing 7 bits, thereby allowing a larger chip size, or by adding more *CAS lines to the existing 3 bits, or both. Naturally, more features can be added to the chip in modes already existing, like a true 512 X 424 X 512 color mode, and other modes of display. Note that any final specifications are up to Yamaha. It is also rumoured that the color bus will be used for the extra banks of memory, thereby making the Myarc mouse and all bus mice unusable with the Geneve, or any of the video cards out there. (This is the same as with the 9958.) It is hoped that the chip will be released some time this year, if not already. Some sources indicate that prototypes are already being used in Japan for development of a new video game standard (seems like overkill!!) to be used in the higher end game market. What this means for us is TRUE VGA compatibility (since the current systems fall short of the 1024 X 800 X 256 color standard). Perhaps the new chip will even address a meg of memory! Keep your ears open and stay tuned!" And thanks for the juicy "rumor" oh mighty ROBOFROG!!!

■

JUSTIFYING DECIMAL

by Earl Raguse, Bug News, USA

Console BASIC does not provide a command for aligning decimal points as does XBASIC with PRINT USING, but here is a routine that does it well. Its so easy, that I often use it in XB, instead of PRINT USING. Lines 5 thru 8 do all the work, the rest are for demonstration. Delete what you wish, RESequence and SAVE in MERGE format for MERGEing with you number programs.

```
  2 CALL CLEAR
  3 INPUT "COLUMN FOR DECIMAL
    POINT ":C
  4 INPUT "INPUT A NUMBER ":X
  5 X$=STR$(X)
  6 IF POS(X$,".",1)=0 THEN 7
    ELSE 8
  7 X$=X$&".00"
  8 PRINT TAB(C-POS(X$,".",1))
    ;X$
  9 GOTO 4
```

■

BEGINNER ASSEMBLER - 6
BITSPRITE

by Mack McCormick, USA

```
****************************************
**  SPRITES AND TEXT FROM BIT MAP     **
**   BY MACK MCCORMICK                **
**     ENTRY POINT START              **
**     WORKSPACE AT >8300             **
**     R0,R1,R2 - GEN VDP             **
**     NO RESERVED REGISTERS ON EXIT  **
**                                    **
**FOR MORE INFO SEE MY SPRITE TUTORIAL**
** SET UP FOR A MEMORY IMAGE SAVE     **
****************************************

       DEF  START,SFIRST,SLAST,SLOAD
       REF  VSBW,VMBW,VSBR,VMBR,VWTR
SLOAD
SFIRST B    @START
PDT    EQU  >0000        PATTERN DESCRIPTOR TABLE
SVVDP1 EQU  >83D4        SAVE LOC FOR VDP R1
SIT    EQU  >1800        SCREEN IMAGE TABLE
CT     EQU  >2000        COLOR TABLE
SATAB  EQU  >1B00        SPRITE ATTRIBUTE LIST
SDTAB  EQU  >1C00        SPRITE DESCRIPTOR TABLE
STATUS EQU  >837C

WS     EQU  >8300        MY WORKSPACE IN HIGH SPEED CPU RAM

*  DATA STATEMENTS  *

VDPREG DATA >02A0,>06FF,>0336,>030F VDP REGS BIT MAP MODE

SPRITE DATA >0001,>0307,>0F0F,>3FFF,>3F0F,>0F07,>0301,0  SPRITE DESCRIPTOR
       DATA >F0E0,>E0C0,>8286,>FEFF,>FE86,>82C0,>E0E0,>F000
SALINT DATA >5080,>8005,>D000

CHRTAB BSS  >2F8

STRING TEXT 'THIS IS THE TEXT TO BE DISPLAYED'
       EVEN

*  BEGIN CODE  *

START  LWPI WS

*  SAVE THE CHAR SET FROM E/A LOADER  *

       LI   R0,>900      DASE ADDRESS E/A (START CHAR >20)
NOTEA  LI   R1,CHRTAB    READ THE CHAR SET INTO CHRTAB
       LI   R2,>2F8
       BLWP @VMBR

*  SET VDP REGS TO BIT MAP  *

       CLR  R0           VDP REG VALUE
       LI   R1,VDPREG    VDP REG POINTER
       LI   R2,8         REGISTERS TO WRITE TO
```

```
VDPL    MOVB *R1+,@WS+1   MOV TO R0 LSB
        BLWP @VWTR
        AI   R0,>0100     NEXT VDP REG
        DEC  R2
        JNE  VDPL
        MOVB @VDPREG+1,@SVVDP1 BLANK DISABLED
* YOU MUST SAVE A COPY OF VDP R1 AT >83D4 OR KEYSCAN WILL INTERFERE

*  FORMAT SCREEN >00->FF THREE TIMES  *

        LI   R0,SIT
        CLR  R1           BEGIN WITH 0
        LI   R2,>300      LEN OF SIT
THIRDS  BLWP @VSBW
        INC  R0           NEXT LOCATION
        AI   R1,>0100     WILL WRAP AFTER >FF00
        DEC  R2
        JNE  THIRDS

*  CLEAR THE PATTERN AREA BLANK AND MAKE COLOR TABLE RED ON WHITE*

        BL   @CLRVDP      CLEAR PATTERN AREA
        DATA PDT,>1800,0  DATA FOR SUBROUTINE
        BL   @CLRVDP      RED ON WHITE COLOR AREA
        DATA CT,>1800,>9F9F

*  ENABLE THE SCREEN SO YOU CAN SEE  *

        LI   R0,>01E3     WRITE >E3 TO VDP REG 1
        BLWP @VWTR        DOUBLE/MAGNIFIED SPRITES
        SWPB R0
        MOVB R0,@SVVDP1   SAVE THE VALUE

*  PUT THE TEXT ON THE SCREEN  *

        BL   @DISTEX      SUBROUTINE TO PLACE TEXT ON THE SCREEN
        DATA >B00,STRING,32

*  SET UP SPRITES *

        LI   R0,SDTAB     SPRITE DESCRIPTOR TABLE ADDRESS
        LI   R1,SPRITE    SPRITE CHARACTER DATA
        LI   R2,32        DOUBLE SIZE SPRITE
        BLWP @VMBW

        LI   R0,SATAB     SPRITE ATTRIBUTE TABLE
        LI   R1,SALINT    INIT DATA FOR SPRITE ATTRIBUTE LIST
        LI   R2,5         FIVE BYTES TO WRITE
        BLWP @VMBW        >D0 REQUIRED TO DISABLE REMAINING SPRITES

* THE SPRITE IS ON THE SCREEN. NOW TO MOVE IT.

LOOP    LI   R0,SATAB+1   X POSITION OF THE SPRITE. Y AT SATAB
        BLWP @VSBR        READ THE CURRENT X POSITION
        SRL  R1,8         PUT VALUE IN RIGHT BYTE
        DEC  R1           SUBTRACT 1 FROM THE X VALUE
        JNE  MOVE         IF X=0 THEN
        LI   R1,>FF       X=>FF

MOVE    SLA  R1,8         MOVE TO MSBYTE
        BLWP @VSBW        WRITE THE NEW X POSITION UP
```

```
       LI   R2,800        DELAY TO SLOW DOWN THE SPEED OF THE SPRITE
DELAY  NOP                WASTE TIME
       DEC  R2
       JNE  DELAY

       JMP  LOOP          THIS IS WHERE YOU COULD PLACE THE REMAINING PROGRAM

* THIS IS A CONTINUOUS LOOP NO PROVISION FOR EXITING GIVEN *

************************************
*  SUBROUTINE TO FILL VDP WITH DATA  *
*    RT IS R12                       *
*    DATA PDT,>1800,>0000            *
*    (DEST,LEN,VALUE TO FILL)        *
************************************

CLRVDP MOV  *R11+,R0      DEST
CLRLV  MOV  *R11+,R3      LEN
CLRV   MOV  *R11+,R4      VALUE(S)
CLR    MOV  R11,R12       SAVRTN

       SRL  R3,1          DIVIDE BY 2
       LI   R2,2          BYTES OF DATA
       LI   R1,WS+8       FROM R4
CLRVL  BLWP @VMBW
       INCT R0            NEXT 2 LOCATIONS
       DEC  R3            DONE?
       JNE  CLRVL
       B    *R12          RT

************************************
*      DISPLAY TEXT SUBROUTINE     *
*         (LOC,TEXT,LENGTH)        *
*           >388,MMENU,9           *
************************************

DISTEX MOV  *R11+,R0      LOCATION
*                         LOC=((ROW-1)*256)+((COLUMN-1)*8)
       MOV  *R11+,R3      ADDRESS OF TEXT TO WRITE
       MOV  *R11+,R5      LENGTH OF TEXT STRING
       LI   R2,8          8 BYTES IN A CHAR DESCRIPTION
PUTDIS MOVB *R3+,R1       BYTE TO WRITE
       SRL  R1,8          RIGHT JUSTIFY
       AI   R1,-32        STRIP THE ASCII OFFSET
       SLA  R1,3          MULTIPLY BY 8 FOR OFFSET
       AI   R1,CHRTAB     ADD THE BASE ADDRESS OF THE CHAR TABLE
       BLWP @VMBW         PUT IT ON THE SCREEN
       AI   R0,8          NEXT SCREEN LOCATION
       DEC  R5            DECREMENT LENGTH COUNTER
       JNE  PUTDIS        ANY MORE TO WRITE?
       RT

SLAST  END                                                          ■
```

```
1 REM SVENSKA BASIC
2 REM PB 85-4.23
8000 CALL CHAR(91,"002800384
47C4444")
8010 CALL CHAR(92,"0028007C4
444447C")
8020 CALL CHAR(93,"003828384
47C4444")
8030 CALL CHAR(123,"00002800
38447C44")
8040 CALL CHAR(124,"00002800
7C44447C")
8050 CALL CHAR(125,"00003828
38447C44")
```

FAST EXTENDED BASIC!
87/12 - LOVESTORY

My Christmas gift to you is a real program, one that you can keep and play with during the holiday season... a cute game full of SPRITES. In a winter landscape, a boy and girl walk or run at random speeds; you can change their direction and speed by pressing any key. Sometimes boy chases girl, sometimes the opposite (she's a woman of the 80's). See what happens when they meet!

After the now familiar PRE-SCAN (note that it is put back ON in line 280, long before the end, since a lot of CALLs and variables are used only in the last portion of the program), the title and landscape are displayed while Tex lazily redifines our sprites; we cannot use our fast DISPLAY AT routine for the landscape (see Sept. issue), since we want to use all 32 columns.

All our sprites are four-char. long, and their numbers are all multiples of 4. Char. 36 defined in line 140, takes up only one char., so we can use it as a single character in the title (line 150); to use it as a sprite, we make the following three char. blank, hence the RPT$ (repetition) of "0"s. In line 240, we encounter a CALL MAGNIFY: this special sprite function controls their size. The factor of 4 will make them double size, i.e. 16 char. (4 ch.*4). Unfortunately, it applies to all the sprites in the program, that is why we made the surprise so small: when magnified, it will be only 4 (4*1) char. in size.

Line 250 plants the trees in the landscape; these are stationery sprites, so we could have used standard CALL HCHARs, but can you imagine the length of the code to define and display three trees of 16 char. each? Using sprites simplifies our work. CALL SPRITE is one of those "multiple parameters" statement: you can use it for as many sprites as you want, three here; I will explain below how the sprite numbers were chosen.

Let's study the left tree:
```
CALL SPRITE(#9,120,13,89,26,#7,...
            S# CH#,CL,PR,PC
```

Sprite number S# is the number we assign to the tree on the left, and CH# is its character number. CL is the color of the sprite foreground: since sprites are designed to be seen over each other, their background is always transparent, and does not have to be specified. To make them move smoothly, XB controls the sprites pixel by pixel, instead of row by row and column by column. Since there are eight pixels in a row or column, to get the right p-row and p-column (PR and PC) you multiply the normal value by eight and adjust it as you want. Here, p-row 89 is somewhere "into" normal row 11, p-column 26 "into" column 4.

The moon is dealt with on line 260, in a single statement, but we have two additional parameters after its position (25 and 180), for the "velocity" of the sprite. The first parameter is vertical vel.; since it is "0", our moon will stay up in the air. The second one is for horizontal vel.; to make the program more animated, the moon moves very slowly towards the left, hence the "-1" (negative value for left, positive for right). Change the "-1" to anything up to "-127", or a number between 1 and 127, and see what happens!

Boy and girl come on stage as motionless sprites (line 270), their velocity randomly determined on line 290; it will be any value from -25 to +24, and each figure gets its own motion, as in real life. Their velocity is then passed on to the CALL MOTION statement in line 300; it has only two parameters, vertical ("0" in this case: we don't want our couple to wander around and hide behind the trees!) and horizontal,

which are our random values. Each time you press a key, the CALL KEY statement in line 330 will take the program counter back to these lines, otherwise it will skip them.

Line 310 controls the ligthing of the trees by two user-defined subroutines; I chose that approach so you can play with the lights if you wish (no, I had no tutorial intent! this program is for fun!). And since we expect some action from our couple, line 320 checks for the coincidence of their sprites. The "10" is a coincidence factor: the last parameter, "COIN", will return a value of -1 when the coincidence is 10 pixels or less, or of 0 when there is no coincidence. Note that this does not work all the time: this is because the program is not on that particular line when they meet... bad luck! If there is a coincidence (COIN=-1), well, read the name of the sub to find what happens....

To light up the trees, the SUB ON puts three more sprites right on top of the trees (same position), and the SUB OFF deletes them. Readers who know about sprites might object that CALL PATTERNs would have been simpler: CALL SPRITE the lights somewhere before the main game routine, and you define another sprite, e.g. 96, as a blank one, i.e. with a foreground color of 1, transparent: no need to redefine the character, since we will never see it! We would then define SUB ON as CALL PATTERN(#8,124,#6,124,#1,124), and SUB OFF as CALL PATTERN(#8,96, #6,96,#1,96).

BUT: XB does not allow more than four sprites on the same row at the same time. We already have a boy, a girl, a tree and a light cluster, on or off. When the couple meets, the heat of its passion causes a fifth sprite to appear, so one has to disappear. Since the girl's sprite has the highest number (#5), she loses her body! Then, as the surprise sprite moves upwards, the trees loose part of their branches... Try it just to see! So, we avoid that problem by deleting the light-sprites in the SUB OFF (our heroes' passion is hot enough to keep the scene lighted anyway), and put them back in the SUB ON.

You see now that careful planning is needed when you number your sprites: here, we want the couple to pass beyond the right tree, but in front of the middle one. And we want the surprise to appear between the couple and the right tree in case their encounter happens right behind it. The sprite with the lesser number passes in front of a sprite with a higher number; I therefore planned the sprites as follows ("ltr" means left tree, "l.ltr" lights on left tree, and so on):

#10 moon	# 9 ltr	# 8 l.ltr	# 7 mtr	# 6 l.mtr	# 5 grl	# 4 boy	# 3 ???	# 2 rtr	# 1 l.rtr

A word on the SUB KISS: each time boy meets girl, or vice-versa, we freeze them in embrace with a motion of "0,0" and we sneak on them with a CALL POSITION that gives us the p-row and p-column of the top left pixel of the boy-sprite (by that time, the girl is close enough!). If PC is too far out on the right, we correct it (there are 256 p-columns, and the surprise sprite will be six pixels to the right of boy, so 250 is the highest PC we want). We now use those values to call a new sprite slightly right and below the couple's upper left position, then give it a vertical velocity of -30 (negative value for upward motion, positive for downward) and an horizontal one of 0.

A short sound confirms the happy ending, as the moon quietly continues to pace the sky. If you don't change the velocity of a sprite, it moves forever, until you stop it with a CALL MOTION(#n,0,0) or delete it with a CALL DELSPRITE (see line 290). After the brief kiss, our couple can resume its

course in line 300.

Before I let you play, a last, COLORful note: if you study the program, you may have noted that we have defined two sprites in one set (trees+lights, boy+girl). But their colors, in the CALL SPRITE statements, are not the same: boy is blue, girl is magenta. Even better, in line 350, the lights have three colors (12, 8 and 16), but are all the same character, i.e. same set! And the surprise, which was left its default black in the title, becomes a vivid red when used as a sprite. This is because the color of a sprite is totally independant from the color of the char. sets. The VDP RAM keeps that information in two distinct areas: sprite color is part of the SPRITE ATTRIBUTE TABLE, while character/set colors are kept in the COLOR TABLE.

TO DESIGN SPRITES, you can use any good Sprite Definition program, such as Compute!'s SUPERFONT, or you can use a program which has a ready-to-use collection of sprites. Since I am lazy, this is exactly what I have done: tree (PINE), MOON and GIRL come straight from SPRITE BUILDER program, and boy is just its FARMER without a hat. Good news: SPRITE BUILDER is freeware, and part of the Ottawa Users' Group Library!

```
100 REM ** LOVE STORY / by L
. Dorais / Nov. 1987
120 RANDOMIZE :: GOTO 140 ::
 CALL CHAR :: CALL HCHAR ::
CALL SCREEN
130 CALL COLOR :: CALL MAGNI
FY :: !@P-
140 CALL CHAR(36,"44EEFEFEFE
7C3810"&RPT$("0",48))! surpr
ise
150 DISPLAY AT(7,4)ERASE ALL
:"$ LOVE STORY $" :: RANDOMI
ZE
160 CALL CHAR(48,"FFFFFFFFFF
FFFFFFF",136,"FFFFFFFFFFFFFF
FF"):: CALL COLOR(3,15,1,14,
16,1)
170 CALL HCHAR(16,1,136,128)
:: CALL HCHAR(20,1,48,64)::
CALL HCHAR(22,1,136,96)
180 CALL CHAR(120,"0103070B0
70F170F1F2F1F3F5F0101010080C
0A0C0E0D0E0F0E8F0F8F4")! tre
es
190 CALL CHAR(124,"010001080
0021000042100044000000000000
02000401080002800A004")! lig
hts
200 CALL CHAR(104,"030303070
1070F1B33070F1F06060E0EC0C0C
0E080E0F0D8CCE0F0F860607070"
)! girl
210 CALL CHAR(108,"030303030
1070F1B3303070706060E0EC0C0C
0C080E0F0D8CCC0E0E060607070"
)! boy
220 CALL CHAR(116,"030F1F070
3010103070F0300031F0F03C0F0F
8FCFC9CBEFEFEBE3C7CFCF8F0C0"
)! moon
230 REM
240 DISPLAY AT(7,4):"" :: CA
LL SCREEN(5):: CALL MAGNIFY(
4)
250 CALL SPRITE(#9,120,13,89
,26,#7,120,13,110,106,#2,120
,13,144,200)! put trees
260 CALL SPRITE(#10,116,11,2
5,180,0,-1)! put moon
270 CALL SPRITE(#4,108,5,132
,64,#5,104,14,132,145)! put
couple
280 !@P+
290 BV=(RND*50)-25 :: GV=(RN
D*50)-25 ! velocity
300 CALL MOTION(#4,0,BV,#5,0
,GV)! move couple
310 CALL ON :: CALL OFF ! li
ght trees
320 CALL COINC(#4,#5,10,COIN
):: IF COIN=-1 THEN CALL KIS
S :: GOTO 300
330 CALL KEY(0,K,S):: IF S=0
 THEN 310 ELSE 290 ! press a
ny key
340 REM
350 SUB ON :: CALL SPRITE(#8
,124,12,89,26,#6,124,8,110,1
06,#1,124,16,144,200):: SUBE
ND
360 SUB OFF :: CALL DELSPRIT
E(#8,#6,#1):: SUBEND
370 SUB KISS :: CALL MOTION(
#4,0,0,#5,0,0):: CALL POSITI
ON(#4,PR,PC):: IF PC>250 THE
N PC=250
380 CALL SPRITE(#3,36,9,PR+6
,PC+6,-30,0):: CALL SOUND(30
0,1900,0)
390 FOR DEL=1 TO 300 :: NEXT
 DEL :: CALL DELSPRITE(#3)::
 SUBEND
```

■

TIPS FROM TIGERCUB #45

TIGERCUB SOFTWARE
156 Collingwood Ave.
Columbus, OH 43213

Distributed by Tigercub Software to TI-99/4A Users Groups for promotion.

Here is a versatile printer utility which will accept all printer control codes, print in 1 to 5 columns with choice of column separation and margin width, allow alternate margins and pause at end of page to turn paper over, and will load and print a diskfull of files one after another. It is set up for the Gemini 10X and may require modification for other printers.

```
100 DIM M$(400),F$(50)
110 GOTO 150
120 K,ST,SET,S,P$,P,CL,DW$,S
S$,I$,D$,E$,NC,CW,TC,TA,TX,A
V,CS,S$,LT,A$,LSP,LP,RM,OK$,
QQ$,X,F$(),SL,F,IP,M$(),T$,F
LAG,J,PP,LT$
130 CALL CLEAR :: CALL KEY :
: CALL COLOR :: CALL SCREEN
:: CALL SOUND
140 !@P-
150 CALL CLEAR :: CALL KEY(3
,K,ST):: ON WARNING NEXT
160 FOR SET=0 TO 14 :: CALL
COLOR(SET,2,8):: NEXT SET ::
 CALL SCREEN(5)
170 DISPLAY AT(3,6):"TIGERCU
B PRINTALL": :TAB(7);"Copyri
ght 1987":TAB(6);"Tigercub S
oftware" !programmed by Jim
 Peterson
180 DISPLAY AT(12,1):"May be
 distributed without":"restr
iction providing that":"no p
rice or copying fee is":"cha
rged."
190 DISPLAY AT(18,7):"TURN P
RINTER ON!"
200 DISPLAY AT(20,8):"PRESS
ANY KEY" :: DISPLAY AT(20,8)
:"press any key" :: CALL KEY
(0,K,S):: IF S=0 THEN 200 EL
SE CALL CLEAR
210 DISPLAY AT(12,1):"PRINTE
R DESIGNATION?" :: ACCEPT AT
(14,1)BEEP:P$ :: IF POS(P$,"
.LF",1)=0 THEN P$=P$&".LF"
220 ON ERROR 230 :: OPEN #1:
P$,VARIABLE 255 :: ON ERROR
STOP :: PRINT #1:CHR$(27);"@
" :: CALL CLEAR :: GOTO 240
230 DISPLAY AT(20,1):"CANNOT
 OPEN PRINTER!" :: RETURN 21
0
240 DISPLAY AT(12,1):"PRINT
SIZE?": :"  (1) PICA":"  (2)
 ELITE":"  (3) CONDENSED"
250 ACCEPT AT(12,13)VALIDATE
("123")SIZE(1):P :: PRINT #1
:CHR$(27);"B";CHR$(P);
260 !The values 80, 96 and 1
36 in the next line are the
maximum number of pica, elit
e and condensed characters p
er line on Gemini 10X
270 !Change as necessary for
 your printer!
280 CL=(P=1)*80+(P=2)*96+(P=
3)*136 :: CL=ABS(CL)
290 DISPLAY AT(12,1)ERASE AL
L:"DOUBLE-WIDTH? (Y/N) N" ::
 ACCEPT AT(12,21)SIZE(-1)VAL
IDATE("YN")BEEP:DW$ :: IF DW
$="Y" THEN PRINT #1:CHR$(27)
;"W";CHR$(1);:: CL=CL/2
300 DISPLAY AT(12,1)ERASE AL
L:"SUPERSCRIPT? (Y/N) N" ::
ACCEPT AT(12,20)SIZE(-1)VALI
DATE("YN")BEEP:SS$ :: IF SS$
="Y" THEN PRINT #1:CHR$(27);
"S";CHR$(0);
310 DISPLAY AT(12,1)ERASE AL
L:"ITALICS? (Y/N) N" :: ACCE
PT AT(12,16)VALIDATE("YN")SI
ZE(-1)BEEP:I$ :: IF I$="Y" T
HEN PRINT #1:CHR$(27);"4";
320 DISPLAY AT(12,1)ERASE AL
L:"DOUBLE-STRIKE? (Y/N) Y" :
: ACCEPT AT(12,22)VALIDATE("
YN")SIZE(-1)BEEP:D$ :: IF D$
="Y" THEN PRINT #1:CHR$(27);
"G";
330 IF P<>3 AND P<>4 THEN DI
SPLAY AT(12,1):"EMPHASIZED?
(Y/N) Y" :: ACCEPT AT(12,19)
VALIDATE("YN")SIZE(-1)BEEP:E
$ :: IF E$="Y" THEN PRINT #1
:CHR$(27);"E";
340 DISPLAY AT(12,1)ERASE AL
L:"NUMBER OF COLUMNS? (1-5)"
 :: ACCEPT AT(12,26)VALIDATE
("12345")SIZE(1)BEEP:NC
350 DISPLAY AT(12,1):"COLUMN
```

```
 WIDTH (NUMBER OF": :"CHARAC
TERS?" :: ACCEPT AT(14,13)VA
LIDATE(DIGIT)BEEP:CW
360 TC=NC*CW :: TA=CL-TC ::
TX=TC+NC*2-2
370 IF TX<=CL THEN 390 :: DI
SPLAY AT(18,1):STR$(NC)&" co
lumns of "&STR$(CW)&" charac
ters":"plus 2-column spacing
 equals"
380 DISPLAY AT(20,1):STR$(TC
)&" characters; maximum":"av
ailable in print size":"sele
cted is "&STR$(CL)&".":"****
Please reselect****" :: GOTO
 240
390 IF NC=1 THEN 410 :: AV=I
NT(TA/(NC-1)):: DISPLAY AT(1
2,1)ERASE ALL:"COLUMN SEPARA
TION?":"MINIMUM 2":"MAXIMUM
"&STR$(AV)&" AVAILABLE ":"2"
400 ACCEPT AT(15,1)VALIDATE(
DIGIT)SIZE(-2)BEEP:CS :: IF
CS<2 OR CS>AV THEN 400 ELSE
S$=RPT$(" ",CS)
410 TA=TA-CS*(NC-1):: IF TA<
2 THEN 450
420 DISPLAY AT(12,1)ERASE AL
L:"LEFT MARGIN WIDTH?": :"MA
XIMUM "&STR$(TA)&" AVAILABLE
" :: ACCEPT AT(12,20)VALIDAT
E(DIGIT)BEEP:LT :: IF LT>TA
THEN 420
430 DISPLAY AT(12,1):"ALTERN
ATING LEFT/RIGHT": :"MARGIN?
 (for pages to be":"later re
produced on both":"sides) (Y
/N) N"
440 ACCEPT AT(16,14)VALIDATE
("YN")SIZE(-1):A$
450 LSP=12 :: DISPLAY AT(10,
1):" ":" ":"LINES PER PAGE?
60":" ":" ":" ":" " :: ACCEP
T AT(12,17)VALIDATE(DIGIT)SI
ZE(-3):LP :: IF LP<70 THEN 4
90
460 DISPLAY AT(12,1):"LINE S
PACING - 72 INCH" :: DISPLAY
 AT(11,16):"__" :: ACCEPT AT
(10,16)VALIDATE(DIGIT)BEEP:L
SP
470 IF LP/(INT(72/LSP))>11.5
 THEN DISPLAY AT(20,1):"WON'
T FIT!" :: GOTO 450
480 PRINT #1:CHR$(27);"A";CH
R$(LSP);
490 RM=TA-LT
500 DISPLAY AT(12,1)ERASE AL
L:STR$(NC)&" columns of":STR
$(CW)&"-character width":"le
ft margin of "&STR$(LT)&" sp
aces"
510 DISPLAY AT(15,1):STR$(LP
)&" lines per page":"with "&
STR$(LSP)&"/72 line spacing"
520 DISPLAY AT(17,1):STR$(CS
)&" spaces between columns":
"right margin of "&STR$(RM)&
" spaces": :"OK? (Y/N) Y"
530 ACCEPT AT(20,11)VALIDATE
("YN")SIZE(-1)BEEP:OK$ :: IF
 OK$="N" THEN 240
540 DISPLAY AT(12,1)ERASE AL
L:"PAUSE AT END OF PAGE? N"
:: ACCEPT AT(12,23)VALIDATE(
"YN")SIZE(-1):QQ$
550 DISPLAY AT(1,1)ERASE ALL
:"INPUT FILENAMES TO BE":"PR
INTED.":"PRESS ENTER WHEN DO
NE"
560 X=X+1 :: DISPLAY AT(X+3,
1):"FILENAME? DSK" :: ACCEPT
 AT(X+3,14)SIZE(-12)BEEP:F$(
X)
570 IF F$(X)="" THEN X=X-1 :
: GOTO 600 ELSE F$(X)="DSK"&
F$(X)
580 ON ERROR 590 :: OPEN #2:
F$(X),INPUT ::: CLOSE #2 ::
GOTO 560
590 ON ERROR STOP :: CALL SO
UND(1000,110,0,-4,0):: DISPL
AY AT(20,1):"CANNOT OPEN "&F
$(X):: X=X-1 :: RETURN 560
600 SL=1
610 F=F+1 :: IF F>X THEN 700
 :: ON ERROR 620 :: OPEN #2:
F$(F),INPUT :: DISPLAY AT(22
,1):"READING ";F$(F):: ON ER
ROR STOP :: GOTO 630
620 CALL SOUND(1000,110,0,-4
,0):: DISPLAY AT(20,1):"COUL
D NOT OPEN "&F$(F):: STOP
630 FOR IP=SL TO LP*NC :: LI
NPUT #2:M$(IP):: IF LEN(M$(I
P))=0 THEN 670 :: IF NC>1 AN
D POS(M$(IP),CHR$(13),1)<>0
THEN M$(IP)=SEG$(M$(IP),1,LE
N(M$(IP))-1)
640 !CORRECTED VERSION -OMIT
THIS LINE - IT DOES MORE
HARM THAN GOOD!
650 IF LEN(M$(IP))<=CW THEN
670 :: T$=SEG$(M$(IP),1,CW):
: CALL SOUND(1000,110,0,-4,0
):: DISPLAY AT(12,1):M$(IP);
" OVER";CW;"CHARACTERS":"TRU
NCATED TO ";T$:"OK?"
660 CALL KEY(3,K,S):: IF S=0
 THEN 660 ELSE IF K<>89 THEN
 STOP ELSE M$(IP)=T$
670 M$(IP)=M$(IP)&RPT$(" ",C
W-LEN(M$(IP)))
680 IF EOF(2)=1 THEN CLOSE #
```

```
2 :: SL=IP+1 :: GOTO 610
690 NEXT IP :: IF EOF(2)=1 T
HEN CLOSE #2 :: GOTO 720 ELS
E GOTO 720
700 ON ERROR 710 :: FLAG=1 :
: FOR J=IP+1 TO NC*LP :: M$(
J)="" :: NEXT J :: GOTO 720
710 STOP
720 PP=PP+1 :: IF PP/2=INT(P
P/2)AND A$="Y" THEN LT$=RPT$
(" ",RM)ELSE LT$=RPT$(" ",LT
)
730 FOR J=1 TO LP :: ON NC G
OSUB 750,760,770,780,790 ::
NEXT J :: PRINT #1:CHR$(12):
: SL=1 :: IF F>X THEN STOP E
LSE IF QQ$="N" THEN 630
740 DISPLAY AT(24,1)BEEP:"PR
ESS ANY KEY TO CONTINUE" ::
CALL KEY(0,K,S):: IF S=0 THE
N 740 ELSE DISPLAY AT(24,1):
"" :: GOTO 630
750 PRINT #1:LT$&M$(J)&CHR$(
10):: RETURN
760 PRINT #1:LT$&M$(J)&S$&M$
(J+LP)&CHR$(10):: RETURN
770 PRINT #1:LT$&M$(J)&S$&M$
(J+LP)&S$&M$(J+LP*2)&CHR$(10
):: RETURN
780 PRINT #1:LT$&M$(J)&S$&M$
(J+LP)&S$&M$(J+LP*2)&S$&M$(J
+LP*3)&CHR$(10):: RETURN
790 PRINT #1:LT$&M$(J)&S$&M$
(J+LP)&S$&M$(J+LP*2)&S$&M$(J
+LP*3)&S$&M$(J+LP*4)&CHR$(10
):: RETURN
```

This is an improved version of the math program in Tips #36.

```
100 CALL CLEAR :: RANDOMIZE
110 B=INT(5*RND+2):: IF B=B2
 THEN 110 ELSE B2=B
120 F=INT(5*RND+2):: IF F=F2
 THEN 120 ELSE F2=F
130 D=INT(5*RND+2):: IF D=D2
 THEN 130 ELSE D2=D
140 X=F*B*D
150 BB=INT(5*RND+2):: IF BB=
BB2 OR BB=B THEN 150 ELSE BB
2=BB
160 DD=INT(5*RND+2):: IF DD=
DD2 OR DD=D THEN 160 ELSE DD
2=DD
170 F=F*BB*DD
180 DISPLAY AT(3,1)ERASE ALL
:"IF";B;"BOYS CAN CATCH";X;"
FROGS IN";D;"DAYS,"
190 DISPLAY AT(6,1):"HOW MAN
Y FROGS CAN";BB;"BOYS":"CATC
H IN";DD;"DAYS?"
210 ACCEPT AT(7,19):Q
220 IF Q=F THEN DISPLAY AT(9
,1):"THAT'S RIGHT!" :: GOTO
110
230 DISPLAY AT(9,1):"NO, THA
T'S WRONG."
240 DISPLAY AT(11,1):"IF";B;
"BOYS CAN CATCH";X;"FROGS IN
";D;"DAYS"
250 DISPLAY AT(13,1):"THEN O
NE BOY CAN CATCH";X/B;"FROGS
 IN";D;"DAYS"
260 DISPLAY AT(15,1):"AND ON
E BOY CAN CATCH";X/B/D;"FROG
S IN ONE DAY."
270 DISPLAY AT(17,1):"SO, IF
 ONE BOY CAN CATCH";X/B/D;"F
ROGS IN ONE DAY,"
280 DISPLAY AT(19,1):"THEN";
BB;"BOYS CAN CATCH";X/B/D*BB
;"FROGS IN ONE DAY"
290 DISPLAY AT(21,1):"AND";B
B;"BOYS CAN CATCH";X/B/D*BB*
DD;"FROGS IN";DD;"DAYS."
300 DISPLAY AT(24,1):"PRESS
ANY KEY" :: CALL KEY(0,K,S):
: IF S=0 THEN 300 ELSE 110
```

Here's an idea for an unusual title screen -

```
100 CALL CLEAR :: FOR SET=1
TO 8 :: CALL COLOR(SET,1,1):
: NEXT SET :: CALL CHAR(100,
"0",101,"0")
110 X$(0)="4043241818244202"
 :: X$(1)="4021261818648402"
 :: X$(2)="2020131C38C80404"
 :: X$(3)="1010101FF8080808"
 :: X$(4)="081010907E111020"
120 X$(5)="080808F81F101010"
 :: X$(6)="0404C8381C132020"
 :: X$(7)="0284641818262140"
130 A$=RPT$(CHR$(100)&CHR$(1
01),13):: FOR R=1 TO 24 :: C
=C+1+(C=2)*2 :: DISPLAY AT(R
,C):A$ :: NEXT R
140 CALL VCHAR(1,29,1,168)
150 CALL SCREEN(2):: CALL CO
LOR(9,5,16):: FOR S=1 TO 8 :
: CALL COLOR(S,16,2):: NEXT
S
160 DISPLAY AT(5,5):" TIGERC
UB SOFTWARE ";:: DISPLAY AT(
8,6):" SQUIRMY SCREEN ";
170 FOR J=0 TO 7 :: CALL CHA
R(100,X$(J)):: CALL CHAR(101
,X$(7-J)):: NEXT J
180 CALL KEY(0,K,S):: IF S=0
 THEN 170
```

MEMORY FULL Jim Peterson ■

FROM BASIC TO ASSEMBLY
TEXT 1 MODE

by Bob August, Bug News, USA

Up to now, we have been working in what is called the Graphics mode. This gives you 32 columns of print on the screen (0 to 767 total characters). This month we will show you how to use what is called TEXT mode. This will give you forty columns of print on the screen (0 to 959 total characters). In Extended Basic there is not an equivalent to 40 columns unless you play games with the program, so we will not show you an Extended Basic example this month.

We used a new utility in our program this month which is VWTR. This stands for single byte write to VDP register. Also a new Mnemonic SWPB which stands for swap bytes. This moves the high byte to the low byte and the low byte to the high byte. In our program we swap the bytes in register zero (>01F0 to >F001) and then move the >F0 to >83D4. To go to the text mode you only need these four commands:

```
LI   R0,>01F0
BLWP @VWTR
SWPB R0
MOVB R0,@>83D4
```

Put this in your program and you are in text mode. It's as simple as that. In the text mode you have a total of 960 screen locations instead of 768 as in graphics mode. Because of this we had to change our clear screen routine so we would clear the entire screen. We added the gauge of 1 to 40 on the screen so you could see that you had forty columns. There are no other changes in the program. Your Display at and Key scan routines are the same as we used before. Next month we will show you how to put color into you life.

HAPPY ASSEMBLING! ■

BASIC DISPLAY AT
(favorit i repris)

```
60 REM DISPLAY AT I BASIC
70 REM NITTINIAN 83-1.13
80 CALL CLEAR
90 RAD=3
100 KOL=10
110 Z$="TEXT SOM SKRIVS UT"
120 GOSUB 1000
130 GOTO 130
1000 REM DISPLAY AT I BASIC
1010 FOR Z=1 TO LEN(Z$)
1020 CALL VCHAR(RAD,KOL,ASC(
SEG$(Z$,Z,1)))
1030 KOL=1-KOL*(KOL<>32)
1040 RAD=RAD-(KOL=1)
1050 NEXT Z
1060 RETURN
```

```
**********************
* BASIC TO ASSEMBLY *
*  Lesson Number 8  *
**********************
*
       DEF  START                    Entry point of program
       REF  VSBW,VMBW,KSCAN,VWTR     Utilities used in program
*
WRKSP  BSS  32                       Workspace buffer
SAV11  BSS  2                        Save return address buffer
*
GAUGE  TEXT '0000000001111111111222222222233333333334'   40 column gauge
       TEXT '1234567890123456789012345678901234567890'
MESAGE TEXT 'This is in text mode and we have forty  '    Message in
       TEXT 'columns on the screen instead of thirty '    40 columns
       TEXT 'two columns like we have in the graphics'
       TEXT 'mode.'
```

```
QUIT   TEXT 'Press the Enter key to quit'              Prompt to quit
*
       EVEN                  Make sure we start on even byte
*
* Start of program
*
START  MOV  R11,@SAV11       Save return address
       LWPI WRKSP            Load our workspace
       BL   @CLEAR           GOSUB CLEAR to clear the screen
*
* Put into text form
*
       LI   R0,>01F0         Load VDP R1 with >F0
       BLWP @VWTR            Write if to VDP register
       SWPB R0               Switch left and right bytes
       MOVB R0,@>83D4        Store result in >83D4 for key scan
*
       BL   @DISPLY          GOSUB to DISPLY routine
       DATA 160,GAUGE,80     Screen location, Text, Length
*
       BL   @DISPLY          GOSUB to DISPLY routine
       DATA 280,MESAGE,125   Screen location, Text, Length
*
       BL   @DISPLY          GOSUB to DISPLY
       DATA 726,QUIT,27      Screen location, Text, Length
*
*      Call Key routine
*
       CLR  @>8374           Clear to zero for CALL KEY(0,K,S)
       CLR  @>837C           Clear status to zero
KLOOP  BLWP @KSCAN           CALL KEY(0,K,S)
       CB   @>837C,@>20      Check status for key press (may not work ed. note)
       JEQ  KLOOP            If S=0 THEN KLOOP          (>0020 is >0460 ed. note)
       MOV  @>8375,R0        Move key press to register zero
       CI   R0,>0D           Compare to 13 or Enter key
       JNE  KLOOP            If not enter key, GOTO KLOOP
       CLR  @>837C           Clear status to zero
       MOV  @SAV11,R11       Put return address in register 11
       BLWP @0               Quit ( FCTN = )
*
* Clear screen routine
*
CLEAR  LI   R0,0             Load R0 with zero ( Row 1, Column 1 )
       CLR  R1               Clear Register one
       LI   R2,1             Load R2 with one ( Length byte )
CLOOP  BLWP @VSBW            Write a space to the screen
       INC  R0               Add one to R0
       CI   R0,959           Compare R0 to 959 ( 24 X 40, 0 to 959 )
       JLE  CLOOP            Jump to CLOOP if less then 959
       RT
*
* Display at routine
*
DISPLY MOV  *R11+,R0         Put screen location in R0
       MOV  *R11+,R1         Put Text in R1
       MOV  *R11+,R2         Put length of text in R2
       BLWP @VMBW            Write it to the screen
       RT                    Return to calling area
*
* End program with auto start
*
       END  START                                                    ■
```

PROGRAM BITEN 79:2

TEXAS INSTRUMENTS INTRODUCERAR HEMDATOR

Söndagen den 3:e juni visades för första gången Texas Instruments hemdator. Tillsammans med hemdatorn, som fått beteckningen TI-99/4, visades även en rad program som lagras i Texas Instruments unika halvledarmodul (Solid State Software). TI-99/4 består av en tangentbordsenhet som innehåller 16 K bytes RAM användarminne, och 26 K bytes ROM. Den har en kraftfull utökad BASIC, fullständig färggrafik, samt ljudgenerator. Den version som visades är avsedd att kopplas till en färgmonitor eller färg-TV med NTSC videoingång, eller valfri svart-vit monitor eller TV.

Tack vare Texas Instruments unika teknik med program lagrade i halvledarmoduler, blir användningen av hemdatorn mycket enkel. Var och en kan redan från början utnyttja TI-99/4:s omfattande möjligheter, och större delen av datorns kapacitet. Allt som behövs för att köra ett program, är att plugga in önskad programmodul och följa instruktionerna som omedelbart kommer på bildskärmen. Fortlöpande ges sedan nya anvisningar som utföres på tangentbordet.

På Chicago-mässan "US-Consumer Electronics Show", som öppnade den 3:e juni, visades tillsammans med TI-99/4 följande programmoduler: privatekonomi, hushållsbudget, pedagogiska spel, grundläggande engelsk stavning och grammatik, fysisk träning, sifferlekar, amerikansk fotboll, datorkonst, demonstration och diagnostik.

Texas Instruments programmoduler för TI-99/4 skiljer sig markant från den vanliga tidsödande metoden med program lagrade på bandkassetter. Programmodulerna, som var och en består av upp till 5 st ROM-kretsar med fast lagrade program, ger användaren omedelbar tillgång till de lagrade programmen, utan att man behöver vänta på att informationen skall läsas in i datorns minne.
En enda programmodul kan innehålla program på 30 K bytes. Med bandkassetter måste man vid längre program vänta åtskilliga minuter, innan programmet är inläst och kan användas.

Förutom hemdatorn och programmodulerna visades också en fjärrkontrollenhet, med en så kallad "Joy Stick" (kulledad styrspak) och en aktiveringstangent på sidan. Två sådana fjärrkontrollenheter kan samtidigt anslutas till datorn.

Ytterligare tillbehör som kommer att presenteras inom en snar framtid är skrivare, flexskivminne (Floppy Disc) RS/232 (V 24)-interface samt en enhet för syntetiskt tal.

Samma avancerade teknologi som i Texas Instruments Speak & Spell utnyttjades i enheten för syntetiskt tal. Den har en inbyggd kapacitet av 200 ord. Ordförrådet kan utnyttjas i program för att producera verbala meddelanden och resultat! Kommande programmoduler kommer att utnyttja denna möjlighet.

Det kraftfulla programspråket som är inbyggt i TI-99/4 - TI BASIC - är helt ASCII-kompatibelt och i överensstämmelse med specifikationerna från American National Standards Institute (ANSI). TI-BASIC omfattar 13 siffrors flyttalsaritmetik, 24 BASIC-satser, 14 kommandon, färggrafik (16 färger) samt ljud och musik över fyra hela oktaver.

Enligt Texas Instruments kommer TI-99/4 att introduceras i Sverige i början av 1980. Hemdatorn är ännu inte prissatt för den svenska marknaden, men en rimlig gissning är omkring 5.000:-- exkl moms.