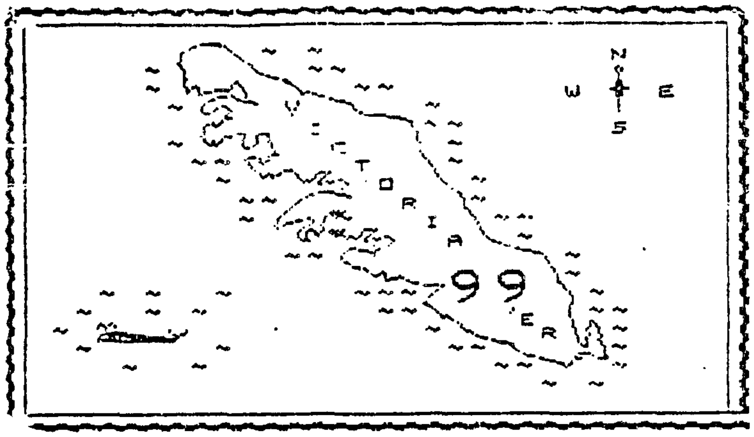

MARCH 1985

Joe Macmurchie made the comment to me that the 99/4A doesn't use assembly language but really uses utility language. This was in reference to the constant use of utility routines in assembly language. Becouse of this high use of utilities and because the modules that run user written assembly language programs have their own utilities and loaders, it is often confusing and difficult for a person learning assembly language to figure out the differing requirements of the modules.

Most utilities are loaded into directly addressable memory from memory mapped areas in GROM. When you go CALL INIT with XBASIC the utilites are loaded at the bottom of low memory $2000 up and the pointers are set. With E/A the utilities are loaded at $2000 up and a reference definition table is loaded at the top of low memory $3FFF down. With MINI MEMORY the utilities are in rom between $6000 and $7000 and when initalized a reference definition table is loaded at $7FFF down. With both E/A and MINI MEMORY pointers are reset upon initalization. Both E/A and MINI MEMORY have a REF DEF table upon initalization. XBASIC has only a DEF table and it is empty on initalization.

In all 3 modules when CALL LOAD("Filename") is used a subprogram is used to load the file program and put any defined routine names and entry points into the DEF table for later use by CALL LINK. The pointer to the table is adjusted for the additional entries. With CALL LINK a search is made from the begining of the DEF table to the pointer position. If the routine name is found control is passed to it, if it is not found an error message is issued. After a program is loaded and it's name is entered in the DEF table if CALL INIT is used you will not be able to link to the program. The program is still in memory and the name is still in the DEF table but the pointer to the end of the DEF table has been reset. If you poke the original value of the pointer back in the program can be linked to again.

The main problems with writing assembly language programs to run with the 3 modules is that their loaders are not compatible, and that some of the utilities avaible for one module are not avaible for another one. The loader in all modules can be accessed with a CALL LOAD. In the E/A and MINI MEMORY modules, LOADER is also a utility which can be run from your assembly language program. These differ form the XBASIC loader in their relocatible code space, their speed, their ability to use a REF DEF table, and to be able to load compressed object code.

The relocatable space for the 3 modules is approx. :
XBASIC $2000 to $3FFF
E/A $2000 to $3FFF and $A000 to $FFFF
MINI MEMORY $2000 to $3FFF and $A000 to $FFF and $7000 to $7FFF

MINIMMEMORY and E/A loaders are fast. XBASIC'S loader is slow. In each module the utility VMBW is stored in a different place. With E/A and MINI MEMORY the loader looks up that place in the REF table . The program just says REF VMBW and the loader substitutes the real address of VMBW when it loads the program. With XBASIC you must specify where the utility is loeated with VMBW EQU $XXXX . This means that all programs written for XBASIC that use utilitys must be converted to run with E/A or MINI MEMORY and viceaversa.

The fact that XBASIC is "missing" DSRLNK and GPLLNK utilities makes for a little less fun as well. Not that long ago (Im still learning) I wrote a fairly large assembly language routine that did string handling and search and modify for a crossword program written in XBASIC. I developed and tested the program with the E/A . When I went to substitute XBASIC EQU's for E/A DEF's I found there was not a DSRLNK utility in XBASIC.

TI probably did these things to XBASIC for good reasons but it sure puts an added burden on the programer. The best way around these hassels is to not use any of the TI utilities or the REF feature of the loaders. You write (Or borrow) your own library of utilities and include their source code with your application's source code. This is what was done with TI DEBUG, and one version will load in all three modules.

The other day I was modifing the above mentioned crossword program to include clues to the words. The program was written over a year ago and at the time I was not aware of loop counter naming conventions. I used loop counters with descriptive names such as DELAY, COUNT, ROW, COL or i used temporary variable names like T1,T2.T3. The advantage of using standard loop counters like I, J, K, L is that anytime you see one of these you know it is in a loop and if used anywhere else it is in a different loop. If you see COL it could be a loop counter or maybe it refers to a sprite position or some other variable. It isn't immediatly apparent that it is a loop counter. Anyway today i always use I, J, K, L etc.

So Im modifing this program and all is going well until I add this loop to input the clues. I use a multi statement line that starts with FOR I=1 TO 150 . When run I get SYNTAX ERROR IN 592 (the line in question). I don,t see anything wrong so I do my usual trick of making a single multi statement line into a multi line group. The idea is to find the exact cause of the syntax error. The error still occurs on 592 FOR I=1 TO 150 . Im using a widget. I save off plug XBASIC directly into the consol and reload, the problem remains. I figure maybe I typed some invisible characters so I type in a line 1 FOR I=1 TO 150 :: NEXT I . I run it and line 1 works but I still get the error in 592 . Time to put it on the printer and find out how I messed up the original program. I saw my problem on the first page. To speed up the start of the program i had declared all my variables and call routines and turned the prescan off. No where in the original program had I used I as a variable. I added it to the prescan list and it now works fine. This all took two hours to find.

A historical note for those interested. Back in the old days 1960's BASIC allowed only a single character for a variable name. No DELAY just D . By convention it was decided to use I, J, K, L for the loop counters. I for the outer loop J for next in K for next again etc.

So I have just suggested that you accomidate your advanced 1980 computer to 1960's hardware and software technoligy.

Here is a little comparison of the capabilities of FORTH and XBASIC while both are in graphics mode.

	FORTH	XBASIC
USER DEFINABLE CHARACTERS	256	112
TOTAL NUMBER OF SPRITES	32	28
DEFINABLE CHARACTERS WITH ALL SPRITES 4 MAGNIFIED AND UNIQUE	256	0

So XBASIC gives you 112 characters OR 28 4 magnified sprites. Forth offers 256 characters AND 32 4 magnified sprites.

In FORTH you can three times the graphics twenty times as fast. What language will you use for your next game?