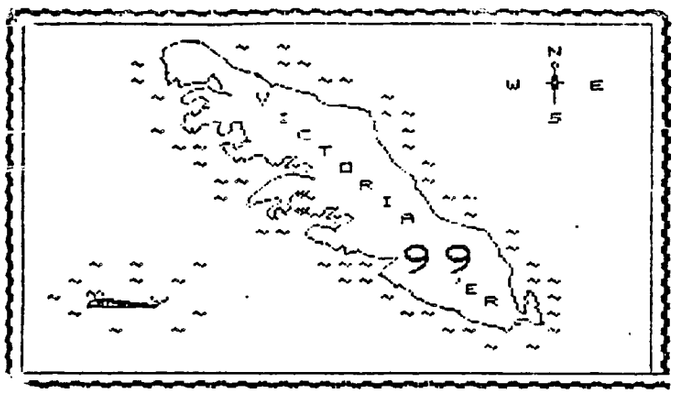

OCTOBER 1984

STRING FUNCTIONS IN BASIC AND X-BASIC:
A string is a variable consisting of a bunch af characters 'strung' together . I23 is a numeric value (one hundred twenty three) "123" is a string value character 49 "1" and character 50 "2" and character 51 "3" . The character numbers mentioned are ASCII characters a standard used by computers so the can exchange information these character codes are in appendix III-1 of your TI 99/4A USERS REFERENCE GUIDE .
Strings can be defined simply by a statement A$="A"
They can also be defined like A$=CHR$(65) ! 65 is the ASCII code for capital A
Here is a break down of the various string 'functions' thes are in the USERS REFERENCE GUIDE between pages II-99 and II-103

ASC(string-expression) This converts a single character into its ASCII code and puts that code into a numeric variable . For instance CODE=ASC("A") would make CODE=65 .

CHR$ (numeric expression) does the reverse of ASC , it takes a number between 0 and 255 and converts it to a single character with the code of the numeric expression . For example STRING$=CHR$(65) makes STRING$="A"

LEN(string-expression) This function puts the number of characters in the string into a variable . LENGTHSTRING=LEN("123456789") would make LENGTHSTRING=9 ! You can use a variable name interchangably with a literial string statement . You can substitute "123456789" for ONETONINE$ and vice-versa .

POS(string1,string2,num-expression) Finds the first occurence of string2 in string1 . It starts looking at the location specified in the numeric expression and looks to the end of the string. If it dosent find a match it puts a 0 into your variable else it puts the location of the start of the match into your variable .

```
STR1$="123456789"
STR2$="456"
POS2IN1=POS(STR1$,STR2$,1) would make POS2IN1=4
POS2IN1=POS(STR1$,STR2$,5) would make POS2IN1=0 ! no match
```

SEG$(string,num1,num2) This makes the specified string up from a segment of the original string . The segment starts a the character num1 characters into the string and end at the character num2 characters into the string . PART$=SEG$("123456789",3,4) would make PART$="3456" it starts at the third character and is made of the next 4 characters including the third one . PART$=SEG$("123456789",5,2) would make PART$="56" .

STR$(numeric-expression) This converts the number specified into a string
A$=STR$(1984) makes A$="1984" .

VAL(string-expression) This converts a character string into a numeric value the character string must be one that can be converted or an error will be produced . For example A=ASC("123")

There is another important symbol used with strings this is the & sign this means CONCATINATION just think of it meaning AND or + A$="123456789" or A$="12345"&"6789" are equivalint .

Here is a short program that takes your name as an imput (firstname/last-t name) and converts it from one string into two .

```
100 INPUT "  firstname (space) lastname  ":FULLNAME$
110 SPACELOCATION=POS(FULLNAME$,CHR$(32),1)
120 FIRSTNAME$=SEG$(FULLNAME$,1,SPACELOCATION-1)
130 LASTNAME$=SEG$(FULLNAME$,SPACELOCATION+1,20)
140 PRINT "First name is  "&""""&FIRSTNAME$&""""
150 PRINT "Last  name is  "&""""&LASTNAME$&""""
160 PRINT
170 GOTO 100
```

ASSEMBLY LANGUAGE BASICS :

First a basic program

```
100 FOR I=1 TO 10000      * Starts loop sets counter to 1 .
110 CALL CLEAR            * Goes to clear rountine .
120 NEXT I                * increments counter & checks for end of loop .
130 END                   * End of program .
```

In Ed/Assm 9900 Assembly Language the program would be .

```
0001          CLR  R3             * Starts loop and
0002  ILOOP    AI   R3,1           * Increments counter by one .
0003          BL   @CLEAR         * Goes to clear routine .
0004          CI   R3,10000       * Checks for end of loop
0005          JNE  ILOOP          * Next loop if not done .
0006          END                 * End of program .
```

The BASIC program can be typed in and run . The Assembly Language program while correct will not run without additional supporting segments . The sub-program CLEAR has not been defined and no method for getting into or out of this program has been provided .

The R3 refered to in line 1,2 and 4 of the program is REGISTER 3 . The computer has 16 active registers form R0 to R15 . A register is a location in memory in which is contained a value . The value can be a number or a character or another location in memory . It is not necessarry to know where the registers are located in order to use them . The computer keeps track of where they are .

Line 0001 CLR R3 (clear register 3) is equivalent to LI R3,0 (load immediate register 3 with 0) Both of these put a 0 into register 3 . We don't know where register 3 is but we know it has a 0 in it .

Line 0002 ILOOP AI R3,1 ILOOP is a label (when the program is loaded into memory the location that the instruction AI R3,1 is put into will be substituted for the label whenever it is refered to . If AI R3,1 where loaded into >D000 then line 0005 JNE ILOOP whould in effect become JNE >D000 the assembler uses labels to make your program easier to understand .) AI R3,1 (add immediate to register 3 1 . If register 3 contained a 0 before this instruction it will now contain a 1 .

Line 0003 BL @CLEAR (Branch and link at label CLEAR) This is a jump to a subprogram . Again CLEAR is a label which is the location of the start of a subprogram which clears the screen [it dosent exist yet we have to write it]) BL @CLEAR can be thought of as CALL CLEAR or GOSUB line XXXX in BASIC . The program will do the subprogram CLEAR return and continue with the next pro-gram statement.

Line 0004 CI R3,10000 (Compare immediate value in register 3 with 10000) This instruction sets various pointers in another register called the STATUS register depending if the value in R3=10000 or not . (Again you don't need to know where the status register is , the computer knows .)

Line 0005 JNE ILOOP (Jump not equal to ILOOP) This tests the status register and if the equal test bit has not been set (If R3<>10000) the program jumps back to location ILOOP else it continues to the next instruction . In the case of this non working program this was the last instruction , the assembly line END is something called an ASSEMBLER DIRECTIVE . It is used to let the assembler know it is at the end of the SOURCE CODE listing and to stop making OBJECT CODE , but the directive does not itself produce any object code . SOURCE CODE is the assembly language program that you type in the ASSEMBLER reads it and produces a file of OBJECT CODE which the LOADER loads into the computer for running.

Because this program is not made to return to the program that called it the computer will look at the locations after the program and try to do the instructions that are there > Whatever is in memory after our program is just garbage from our viewpoint but nobody told the computer so it wallows in the garbage and pigs out into never-never-land . It locks up (You would probably have to shut it off)

You will find this to be a popular past time during program development !

I will now try to explain the following REF DEF EQU .

The ED/ASSM loader will support external REFerences . The X-BASIC loader will not . That is why you go VMBR EQU >2114 in X-BASIC and REF VMBR with the ED/ASSM .

When your program is loaded into memory all REFerences to label VMBR in X-BASIC are replaced with >2114 . With MINIMEM and ED/ASSM you can use external REFerences . You go REF VMBR in your program and when your program is loaded into memory the LOADER look in a TABLE for the EQU addresses , the loader in effect does VMBR EQU >xxxx and that value replaces every VMBR label as your program is loaded . All that this does is put the start point of a subprogram called VMBR into your program whereever you had the label VMBR.

when you write a program you need to give it a name so that you can link to it to run it. This is done with the DEFine directive in the manner DEF START and a the start of your program line 0001 in this case you would add your program name as a label

```
0001  START CLR  R3
```

This tells the LOADER to put the label START into the REFerence table and to put the entry point of the program there as well .

Now to stay out of never-never-land after the program is done we want to return to where we were before we linked to the program. When we first enter a program the computer saves our return address in register 11 . It also saves some other important stuff in other registers . It is a good idea when you enter a program to save the registers in the condition that they are in . You do this by switching to your own set of registers and using them until you are ready to return. When ready to return you restore the original registers and branch to the location in register 11 .

Here is a complete program for use with ED/ASSM

```
0001           REF  VSBW            * Makes avaible a utility loaded by ED/ASSM .
0002           DEF  CLLOOP          * Enter from basic with CALL LINK("CLLOOP")
0003 SAVRTN    DATA >0000           * Return address buffer .
0004 MYREGS    BSS  >20             * My registers use these to preserve environment
0005 STATUS    EQU  >837C           * Address of status bytes .
0006
0007 CLLOOP    MOV  R11,@SAVRTN     * Save return address .
0008           LWPI MYREGS          * Use my own registers .
0009 *  Environment is set now entering main program segment .
0010           CLR  R3              * Starts loop and
0011 ILOOP     AI   R3,1            * Increments counter by one .
0012           BL   @CLEAR          * Goes to clear routine .
0013           CI   R3,10000        * Checks for end of loop
0014           JNE  ILOOP           * Next loop if not done .
0015 *  End of main program prepare for return to calling program .
0016           CLR  R0              * Prepare to return .
0017           MOV  R0,@STATUS      * Indicate no errors in status .
0018           RT                   * Same as  B *R11   Return to calling program .
0019
0020 *  This is the end of the main program .
0021 *  Here is the subprogram CLEAR .
0022 CLEAR     CLR  R0              * Holds VDP address to be written to .
0023           LI   R1,>2000        * Left byte HEX 20 (Blank) is value to write .
0024 JLOOP     BLWP @VSBW           * GOSUB VDP single byte write . This routine is
0025 *                                created and loaded by ED/ASSM cartridge .
0026           AI   R0,1            * Add 1 to VDP address value .
0027           CI   R0,768          * Check if the screen has been cleared ( there
0028 *                                are 768 locations to clear on the screen )
0029           JLT  JLOOP           * If not done do JLOOP again
0030           RT                   * Return to main program same as   B  *R11
0031           END                  * This is an assembler directive to stop assembly

0014           END                  * End of program .
```

SCREEN DUMP FROM EXTENDED BASIC .

After your program draws its screen enter the line

xxxx RUN "DSK1.XBSCNDMP2"

Naturalv you call the program XBSCNDMP2

```
100 REM LAZYMANS SIDEWAYS SCREEN DUMP X-BASIC TONY BIGRAS OCT 6 1984 ! GEMINI 10
110 DIM PAT$(145)    ! Holds printer ready character codes .
120 A$="84C2A6E195D3B7F"
130 OPEN #1:"PIO.CR"
140 PRINT #1:CHR$(27)&"A"&CHR$(7)! set linefeed to 7/72
150 FOR I=32 TO 1 STEP -1   ! Start to read screen send 32 lines to printer .
160 PRINT #1:CHR$(27)&"K"&CHR$(192)&CHR$(0);! Graphics mode 192 bytes to come.
170 FOR J=1 TO 24
180 CALL GCHAR(J,I,A)     ! Read values from the screen into A .
190 IF LEN(PAT$(A))>1 THEN 260 ! Dont get patern again.
200 IF (A<32)+(A>143)THEN PAT$(A)="00000000" :: GOTO 260
210 CALL CHARPAT(A,AA$)
220 FOR K=2 TO 16 STEP 2  ! Calculate the printer codes .
230 C(K/2)=POS(A$,SEG$(AA$,K-1,1),1)+POS(A$,SEG$(AA$,K,1),1)*16
240 NEXT K
250 PAT$(A)=CHR$(C(1))&CHR$(C(2))&CHR$(C(3))&CHR$(C(4))&CHR$(C(5))&CHR$(C(6))&CH
R$(C(7))&CHR$(C(8))
260 OUT$=OUT$&PAT$(A)
270 NEXT J
280 PRINT #1:OUT$ :: OUT$=""   ! Output to printer and reset OUT$
290 PRINT #1:CHR$(13)&CHR$(10)  ! Carriage return and linefeed .
300 NEXT I  ! Do next line
310 PRINT #1:CHR$(27)&"@" ! Reset print to nornal modes
320 CLOSE #1
330 END
```