CSI Design Group Presents:

9900BASIC

Assembly Language Programming Aid

—Enhanced Version—

New Features Include:

File Processing

Extended Basic Support

SnapStart Preprocessor

Now it's even easier to convert your TI BASIC or TI Extended BASIC programs into assembly language using the 9900BASIC System

Use SnapStart to help convert existing programs, or start from scratch. Either way it's much easier to program assembly using 9900BASIC.

9900BASIC requires a TI 99/4A with:

One or more disk drives
32K Memory Expansion
Editor/Assembler Module

to use the SnapStart preprocessor you will also need TI Extended BASIC

Version 1.10

9900BASIC QUICK REFERENCE GUIDE

ACCEPT	row,col,size,label
CHAR	char,>data,>data,>data,>data
CLEAR	no param
CLRBUF	label,size
COIN1	sprite number,tolerance,return variable
COLOR	charset,foreground color,background color
DELSPR	sprite number
DISP$	row,column,label,size
FCLOSE	file number
FOPEN	number,mode,logical record length, name length,variable(name)
FOR	variable,beginning value,termination value,step
FREAD	number,buffer,return variable
FWRITE	number,buffer,character count
GCHAR	row,column, return variable
GPOSIT	sprite number,row return variable, column return variable
HCHAR	row,column,character,repetitions
IF	variable,'relation',value,label,label
JOYST	key-unit,x-return,y-return
KEY	key-unit,key-return,status-return
LOCATE	sprite number,dotrow,dotcolumn
MAGNFY	magnification factor
MOTION	sprite number,y-velocity,x-velocity
NEXT	variable
ONGOTO	variable,label,label,label. . . .
PATTER	sprite number,pattern
POSIT	sprite number,y-return,x-return
RETURN	no parameters
RND	tolerance,return variable
SCOLOR	sprite number,color
SCORE	row,column,value,destination variable (must be four bytes)
SCREEN	color
SCRLDN	no parameters
SCRLLT	no parameters
SCRLRT	no parameters
SCRLUP	no parameters
STRLEN	label,return variable
SOUND	please see sound section
SPRITE	number,char,color,dot-row,dot-col,row-velocity,col-velocity
VCHAR	row,column,character,repetitions

NOTE TO USERS:

The routines comprising **9900BASIC** are intended to simulate the Basic language as closely as possible while simultaneously allowing the user the inherent flexibility and speed of assembly language.

This set of routines was designed to be used on the Texas Instruments Home Computer in conjunction with the Texas Instruments Editor-Assembler Package although it will work in conjunction with any editor-assembler which supports all of the functions allowed in the Texas Instruments Package.

These routines may be used and resold as a part of any program providing the following conditions are met: 1) they may be resold as a part of any **OBJECT** code program for resale; 2) the source code is not supplied or included with the program; 3) it is stated on the packaging and documentation of the program '**PARTIAL CONTENTS INCLUDE 9900BASIC tm copyright 1984 CSI Design Group**.

Because these routines are sold as source code as a programmer's aid it is impossible for us to prevent piracy (**THEFT !**) of the source code. These routines were written to help YOU, the programmer, so please treat them as you would like your programs to be treated by the general public.

ABOUT YOUR PROGRAM

It is the opinion of the author of these routines that the the easiest way to develop a program using them is to write the said program in Basic using structured programming techniques, list the source code as a disk file using the default parameters through the command LIST "DSK1.SOURCECODE" and translate this into **9900BASIC** coding one line at a time using your editor. The last statement in your program should be an assembler copy directive COPY "DSK1.9900BASIC". Note that the **9900BASIC** file must be available to the assembler in order to assemble your program. **See SNAP-START, COMPRESSED and EQUATES sections.**

All variables must be defined as labels (PX DATA 0 OR PX BSS 2) with the exception of any variable to be used with the SCORE routine which requires variables of 4 bytes(PLAYER1 DATA 0,0 or PLAYER1 BSS 4). String variables must be allocted the maximum amount of space that they will require with either a BSS or TEXT directive. Arrays may not be directly addressed through the routines but may be indirectly addressed by reference or indirectly addressed by value in one of two ways. MOV @ARRAY(R2),@TEMP will allow the use of the contents of the array element found by dividing the contents of register 2 by 2 and adding one and will place the contents of that array element in the variable TEMP. Using MOV @ARRAY+6,@TEMP will move the contents of the fourth element of the array ARRAY to temp (6/2+1) .Because the **9900BASIC** routines are called with a BLWP instruction you need not worry about them altering your workspace registers.

Always start your program with a LWPI instruction (32 bytes at >20BA are reserved) followed by a BLWP @INIT which performs duties corresponding to sprite usage and VDP access.

There is no error trapping in running programs so make sure that it is not necessary or that you have provided it within your program. Absolute parameters for use with **9900BASIC** may be in the range 0 to >24FF with the exception of the parameters passed to SOUND and CHAR routines. Any other values that you wish to use such as -1 must be defined in the program as constants(MINUS1 DATA -1). It is advisable to make sure that your program terminates in some fashion.

The most common errors usually involve forgetting a parameter which has a default value in basic i.e. a step or repetition parameter. Note that errors of this type will not be caught by the assembler as it assumes that you know how many parameters to pass. **See SNAP-START section.**

All **9900BASIC** functions are called with BLWP instructions and passed parameters in data statements following them.

As with all disk based programs it is advisable to use a work disk and leave the master in a safe place. It may be advantageous from a time and space standpoint for the user to delete some of the functions of **9900BASIC** off of his work disk and leave only the ones he will use. It is important however not to delete any of the subroutines or functions called by any of the other functions the user is leaving on the disk. **See SNAP-START and COMPRESSED sections.**

The mathematics of the the user's program are left to his own design and the user is reminded that expression evaluation is not honored except as is dictated by the assembler.

If any routines which requires interrupts to function properly are enabled (SOUND,MOTION) it is suggested that the KEY routine be used as part of the main loop as it enables interrupts, if this is not practical it becomes a burden on the user to enable interrupts for the system to avoid routine failure or console lockup.

Also included with **9900BASIC** are several routines normally enabled only through a REF directive in your program. These routines are VSBW,VMBW, VSBR,VMBR,VWTR,KSCAN, and DSRLNK. These are included in the interest of completeness and speed. If you want to use them do not include a REF directive in your program, simply use them by name and the assembler will resolve the references. **See COMPRESSED and EQUATES sections.**

****

ROUTINE NAME: ACCEPT

BASIC EQUIVALENT: ACCEPT AT

FORMAT: BLWP @ACCEPT
DATA ROW,COLUMN,SIZE,LABEL

DESCRIPTION: Accepts into a reserved area of memory which starts at LABEL. The number of characters of data accepted into this area is specified by SIZE. The data is accepted beginning at graphics row ROW and Graphics column COLUMN . This routine does not blank the input area before beginning . Data is accepted as a string. If numeric input is desired it must be converted from string data (two examples of this are in the demonstration program).

EXAMPLE:

9900BASIC: BLWP @ACCEPT
DATA 10,3,10,NAME$
BASIC: ACCEPT AT(10,1)SIZE(-10):NAME$

This example will accept a string of up to ten characters at row 10 column 3 on the screen. Note the difference in the column argument and the difference in the size argument. **See CLRBUF as well as SNAP-START section.**

****

****

ROUTINE NAME: CHAR

BASIC EQUIVALENT: CALL CHAR

FORMAT: BLWP @CHAR
DATA CHARACTER NUMBER,>DATA,>DATA,>DATA,>DATA

DESCRIPTION: Defines the specified character number with the 8 bytes of hexadecimal data specified. This routine allows the use of a variable for character number but not for the 8 bytes of data.

EXAMPLE:

9900BASIC: BLWP @CHAR
DATA 42,>FFFF,>FFFF,>FFFF,>FFFF
BASIC: CALL CHAR(42,"FFFFFFFFFFFFFFFF")

This example will define the character 42 (normally an asterisk) as a solid block.

****

****

ROUTINE NAME: CLEAR

BASIC EQUIVALENT: CALL CLEAR

FORMAT: BLWP @CLEAR
(**NO PARAMETERS**)

DESCRIPTION: Fills the screen with the ASCII character 32.

EXAMPLE:

9900BASIC: BLWP @CLEAR

BASIC: CALL CLEAR

This example will clear the screen.

****

****

ROUTINE NAME: CLRBUF

BASIC EQUIVALENT: NONE

FORMAT BLWP @CLRBUF
DATA LABEL,COUNT

DESCRIPTION: LABEL is a a label which represents the beginning of a series of memory locations that are to be initialized to zero COUNT is the number of bytes to be initialized.

EXAMPLE:

9900BASIC: BLWP @CLRBUF
DATA A$,COUNT

BASIC : NONE

This example sets COUNT number of bytes beginning at A$ to zero.

****

****

ROUTINE NAME: COIN1

BASIC EQUIVALENT: NO DIRECT EQUIVALENT

FORMAT: BLWP @COIN1
DATA SPRITE NUMBER,. TOLERANCE, RETURN VARIABLE

DESCRIPTION: Returns in RETURN VARIABLE the lowest numbered sprite within TOLERANCE pixels of sprite SPRITE NUMBER or returns 0 if no coincidence.

EXAMPLE:

9900BASIC: BLWP @COIN1

BASIC: 1,16,C

This example will place in variable C the sprite number of the lowest numbered sprited within 16 pixels of sprite 1.

****

****

ROUTINE NAME: COLOR

BASIC EQUIVALENT: CALL COLOR

FORMAT: BLWP @COLOR
        DATA CHARACTER SET NUMBER,FORE COLOR,BACK COLOR

DESCRIPTION: Sets the foreground and background colors of the character set specified according to the values FORE COLOR and BACK COLOR.

EXAMPLE:

9900BASIC: BLWP @COLOR
               DATA 9,16,2

BASIC: CALL COLOR(9,16,2)

This example will set the characters of character set 9 to white foreground and black background.

****

****

ROUTINE NAME: DELSPR

BASIC EQUIVALENT: CALL DELSPRITE

FORMAT: BLWP @DELSPR
        DATA SPRITE NUMBER

DESCRIPTION: Moves sprite number SPRITE NUMBER out of the visible screen and sets its' motion attributes to 0.

EXAMPLE:

9900BASIC: BLWP @DELSPR
               DATA 1

BASIC: CALL DELSPRITE(*1)

This example will place sprite number 1 off the visible screen and stop its' motion.

****

****

ROUTINE NAME: DISP$

BASIC EQUIVALENT: DISPLAY AT

FORMAT: BLWP @DISP$
        DATA ROW, COLUMN, LABEL, SIZE

DESCRIPTION: Displays SIZE number of characters at Graphics Row ROW and Graphics Column COLUMN. The characters are written from CPU memory marked beginning at LABEL.

EXAMPLE:

9900BASIC: BLWP @DISP$
               DATA 1,3,TITLE$,20

BASIC: DISPLAY AT(1,1)SIZE(20):TITLE$

This example will print the 20 characters of TITLE$ on the screen beginnng at Graphics Row 1 and Graphics Column 3. Note the difference in the column arguments and the difference in argument lists. Note also that this is essentially a VMBW using row and column parameters.

****

****

ROUTINE NAME: FCLOSE

BASIC EQUIVALENT: CLOSE

FORMAT BLWP @FCLOSE
DATA FILE#

DESCRIPTION: FILE# is a value between 0 and 6. All errors are returned in ERFLAG.

EXAMPLE:

9900BASIC: BLWP @FCLOSE
DATA 1

BASIC : CLOSE #1

This example closes the file number 1 that was previously open.

****

****

ROUTINE NAME: FOPEN

BASIC EQUIVALENT: OPEN

FORMAT BLWP @FOPEN
DATA FILE#,MODE,LOGICAL RECORD LENGTH,NAMELENGTH,LABEL

DESCRIPTION: FILE# is a value between 0 and 6,MODE is a value corresponding to a mode taken from the file-mode table,LOGICAL RECORD LENGTH is the length of record to be using during file access,NAMELENGTH is the length of the file descriptor name, LABEL is a label which signifies the beginning of the file descriptor. All errors are returned in ERFLAG.

EXAMPLE:

9900BASIC: BLWP @FOPEN
DATA 1,20,80,9,A$

BASIC : OPEN #1,A$,DISPLAY,VARIABLE 80,INPUT

This example assumes that A$ holds a filename e.g DSK1.DATA and that this file is indeed a dis-var 80 file. Under these conditions the file is opened for input.

****

****

ROUTINE NAME: FOR

BASIC EQUIVALENT: FOR

FORMAT: BLWP @FOR
        DATA VARIABLE,INITIAL,TERMINATE,STEP

DESCRIPTION: Initiates a closed loop setting VARIABLE to INITIAL . The loop will be terminated when VARIABLE is greater than TERMINATE if STEP is positive or when VARIABLE is less than TERMINATE is STEP is negative. Refer to ABOUT YOUR PROGRAM for information concerning negative numbers

EXAMPLE:

9900BASIC: BLWP @FOR
               DATA A,1,10,1

BASIC: FOR A=1 TO 10 STEP 1

This example initiates a for-next loop.

****

****

ROUTINE NAME: FREAD

BASIC EQUIVALENT: INPUT #

FORMAT BLWP @FREAD
DATA FILE#,LABEL,COUNT

DESCRIPTION: FILE# is a file that has been opened for input, LABEL is a label which signifies the beginning of a cpu input buffer,count is a return variable whose value after the read will be the number of characters actually read. All errors are returned in ERFLAG.

EXAMPLE:

9900BASIC: BLWP @FREAD
DATA 1,A$,COUNT

BASIC : INPUT#1,A$

This example assumes that file number 1 has been opened for input. If this is true then FREAD will read the next record from the file and place it in memory beginning at A$ and place the number of bytes actually read in count.

****

****

ROUTINE NAME: FWRITE

BASIC EQUIVALENT: PRINT #

FORMAT BLWP @FWRITE
DATA FILE#,LABEL,COUNT

DESCRIPTION: FILE# is a file that has been opened for output, LABEL is a label which signifies the beginning of a cpu output buffer,count is the number of characters to be written. All errors are returned in ERFLAG.

EXAMPLE:

9900BASIC: BLWP @FWRITE
DATA 1,A$,COUNT

BASIC : PRINT #1,A$

This example assumes that file number 1 has been opened for output. If this is true then FWRITE will write the next record to the file. The record is read beginning at A$ and its' length is COUNT.

****

****

ROUTINE NAME: GCHAR

BASIC EQUIVALENT: GCHAR

FORMAT: BLWP @GCHAR
        DATA ROW,COLUMN,RETURN VARIABLE

DESCRIPTION: Returns in RETURN VARIABLE the ASCII value of the character at row ROW and column COLUMN.

EXAMPLE:

9900BASIC: BLWP @GCHAR
               DATA 10,5,G

BASIC: CALL GCHAR(10,5,G)

This example will place in variable G the ASCII value of the character at row 10 and column 5.

****

****

ROUTINE NAME: GPOSIT

BASIC EQUIVALENT: NO DIRECT EQUIVALENT

FORMAT: BLWP @GPOSIT
DATA SPRITE NUMBER, Y-RETURN,X-RETURN

DESCRIPTION: Returns in Y-RETURN and X-RETURN the graphics row and column presently occupied by sprite number SPRITE NUMBER

EXAMPLE:

9900BASIC: BLWP @GPOSIT
DATA 1,Y,X

BASIC: No direct equivalent

This example will place the row and column position of sprite number 1 in variables Y and X.

****

****

ROUTINE NAME: HCHAR

BASIC EQUIVALENT: HCHAR

FORMAT: BLWP @GPOSIT
DATA ROW,COLUMN,CHARACTER NUMBER,REPITITIONS

DESCRIPTION: Places character CHARACTER NUMBER in consecutive horizontal screen positions beginning at row ROW and column COLUMN repeating the character REPITITIONS times.

EXAMPLE:

9900BASIC: BLWP @HCHAR
DATA 1,2,42,5

BASIC: CALL HCHAR(1,2,42,5)

This example will place 5 consecutive horizontal characters described as character 42 (normally an asterisk) beginning at row 1 column 2 and ending at row 1 column 6.

****

****

ROUTINE NAME: IF

BASIC EQUIVALENT: IF THEN ELSE

FORMAT: BLWP @IF
DATA VALUE1,'RELATION',VALUE2,LABEL1,LABEL2

DESCRIPTION: Conditionally branches to either LABEL1 or LABEL2 depending on whether "VALUE1 RELATION VALUE2" is true. Note that the relation must be in single quotes and must be one of '>' , '<' , '=' . Note also that the else parameter must be included.

EXAMPLE:

9900BASIC: BLWP @IF
DATA A,'>',B,LN100,LN200

BASIC: IF A>B THEN 100 ELSE 200

This example will redirect program execution to the code beginning at LN100 if the contents of memory location(variable) A are arithmetically greater than the contents of memory location (variable) B. Note that if A<=B then program execution will continue at the coding marked with LN200.

****

****

ROUTINE NAME: JOYST

BASIC EQUIVALENT: CALL JOYST

FORMAT: BLWP @JOYST
        DATA KEY-UNIT,X-RETURN,Y-RETURN

DESCRIPTION: Inputs data into the variables X-RETURN and Y-RETURN concerning the status of the remote handheld unit number KEY-UNIT.

EXAMPLE:

9900BASIC: BLWP @JOYST
               DATA 1,JX,JY

BASIC: CALL JOYST(1,JX,JY)

This example will check the position of joystick number 1 and place its' positions in JX and JY.

****

****

ROUTINE NAME: KEY

BASIC EQUIVALENT: CALL KEY

FORMAT: BLWP @KEY
        DATA KEY-UNIT,KEY-RETURN,STATUS-RETURN

DESCRIPTION: Inputs data into KEY-RETURN and STATUS-RETURN based on the value in KEY-UNIT and the keyboard status when checked.

EXAMPLE:

9900BASIC: BLWP @KEY
               DATA 0,K,S

BASIC: CALL KEY(0,K,S)

This example will check the entire keyboard and place the ASCII value of the key depressed into variable K and place the status of the keyboard in variable S. See the Basic or Extended Basic user's manual for more information concerning keyboard I/O.

****

****

ROUTINE NAME: LOCATE

BASIC EQUIVALENT: CALL LOCATE

FORMAT: BLWP @LOCATE
        DATA SPRITE NUMBER,Y-POSITION,X-POSITION

DESCRIPTION: Moves SPRITE NUMBER to the given X and Y coordinates.

EXAMPLE:

9900BASIC: BLWP @LOCATE
               DATA 1,50,45

BASIC: CALL LOCATE(1,50,45)

This example will place sprite number 1 at the coordinates y=50 and x=45.

****

****

ROUTINE NAME: MAGNFY

BASIC EQUIVALENT: CALL MAGNIFY

FORMAT: BLWP @MAGNFY
DATA MAGNIFICATION FACTOR

DESCRIPTION: Sets the sprite magnification the the value FACTOR.

EXAMPLE:

9900BASIC: BLWP @MAGNFY
DATA 3

BASIC: CALL MAGNIFY(3)

This example will set all sprites to double size unmagnified.

****

****

ROUTINE NAME: MOTION

BASIC EQUIVALENT: CALL MOTION

FORMAT: BLWP @MOTION
        DATA SPRITE-NUMBER,Y-VELOCITY,X-VELOCITY

DESCRIPTION: Sets the motion attributes of sprite SPRITE-NUMBER to y=Y-VELOCITY, x=X-VELOCITY.

EXAMPLE:

9900BASIC: BLWP @MOTION
               DATA 1,15,20

BASIC: CALL MOTION(#1,15,20)

This example will give sprite number 1 a y-velocity of 15 and a x velocity of 20.

****

****

ROUTINE NAME: NEXT

BASIC EQUIVALENT: NEXT

FORMAT: BLWP @NEXT
        DATA VARIABLE

DESCRIPTION: Checks for termination of FOR loop by adding STEP to VARIABLE and comparing the result to the TERMINATION value in matching FOR statement.

EXAMPLE:

9900BASIC: BLWP @NEXT
               DATA A

BASIC: NEXT A

This example will add the step value of the matching FOR statement to A and conditionally returns to the statement following the FOR statement.

****

****

ROUTINE NAME: ONGOTO

BASIC EQUIVALENT: ON GOTO

FORMAT: BLWP @ONGOTO
DATA VALUE,LABEL1,LABEL2,.....

DESCRIPTION: Unconditonally branches to the VALUEth label in the list.
EXAMPLE:

9900BASIC: BLWP @ONGOTO
DATA A,L1,L2,L3,L4,L5

BASIC: ON A GOTO 100,200,300,400,500

This example will continue program execution at then Ath label (line number) in the list. Note that 9900BASIC uses labels instead of line numbers.

****

****

ROUTINE NAME: PATTER

BASIC EQUIVALENT: CALL PATTERN

FORMAT: BLWP @PATTER
        DATA NUMBER,PAT

DESCRIPTION: Changes the pattern descriptor number of sprite number NUMBER to PAT.
EXAMPLE:

9900BASIC: BLWP @PATTER
               DATA 1,116

BASIC: CALL PATTERN(#1,116)

This example will change the pattern of sprite number one to the pattern described by character number 116.

****

****

ROUTINE NAME: POSIT

BASIC EQUIVALENT: CALL POSITION

FORMAT: BLWP @POSIT
        DATA SPRITE NUMBER,Y-RETURN,X-RETURN

DESCRIPTION: Returns in Y-RETURN and X-RETURN the current y and x attributes (positions) of sprite number SPRITE NUMBER.

EXAMPLE:

9900BASIC: BLWP @POSIT
               DATA 1,PY,PX

BASIC: CALL POSITION(1,PY,PX)

This example will place the y and x positions of sprite number 1 in variables py and px.

****

****

ROUTINE NAME: RETURN

BASIC EQUIVALENT: NO DIRECT EQUIVALENT

FORMAT: BLWP @RETURN
(NO PARAMETERS)

DESCRIPTION: Not to be mistaken for a return from a subroutine (RT or B *11) this routine branches unconditionally to the program it was called from whether it was TI BASIC, TI EXTENDED BASIC, or the TI EDITOR-ASSEMBLER.

EXAMPLE:

9900BASIC: BLWP @RETURN
(NO PARAMETERS)

BASIC:

This example will return control to the ROM/GROM program that called it.

****

****

ROUTINE NAME: RND

BASIC EQUIVALENT: RND (**CONVERSION NOT DIRECT**)

FORMAT: BLWP @RND
        DATA TOLERANCE,RETURN-VARIABLE

DESCRIPTION: Places in return variable a randomly selected number between 0 and TOLERANCE (inclusive)

EXAMPLE:

9900BASIC: BLWP @RND
               DATA 10,A

BASIC: A=11*RND

This example will place a random number between 0 and 10 inclusive in the variable A.

****

****

ROUTINE NAME: SCOLOR

BASIC EQUIVALENT: CALL COLOR (#

FORMAT: BLWP @SCOLOR
        DATA SPRITE-NUMBER,COLOR

DESCRIPTION: Sets sprite number SPRITE-NUMBER to the specified color.

EXAMPLE:

9900BASIC: BLWP @SCOLOR
               DATA 1,16

BASIC: CALL COLOR (#1,16)

This example will set the color of sprite number 1 to white.
****

****

ROUTINE NAME: SCREEN

BASIC EQUIVALENT: CALL SCREEN

FORMAT: BLWP @SCREEN
        DATA COLOR

DESCRIPTION: Sets the default background color to COLOR. NOTE: the background colors of character set 1 are not automatically set to transparent as in Basic so if you want anything besides the border color to change you must set the background color of character set 1 to transparent.

EXAMPLE:

9900BASIC: BLWP @SCREEN
               DATA 15

BASIC: CALL SCREN(15)

This example will set the default screen color as well as the border color to gray.

****

****

ROUTINE NAME: SCORE

BASIC EQUIVALENT: NO DIRECT EQUIVALENT

FORMAT: BLWP @SCORE
DATA ROW,COLUMN,INCREMENT,VARIABLE

DESCRIPTION: Adds INCREMENT to the 4 byte variable VARIABLE and displays the result at Graphics row ROW and Graphics column COLUMN . Before it displays the decimal value on the screen it blanks an 8 column area. Using this routine either before row 1 column 4 or after row 24 column 26 will cause problems for the VIDEO DISPLAY PROCESSOR. Remember to define a 4 byte variable for use with this routine.
EXAMPLE:

9900BASIC: BLWP @SCORE
DATA 24,5,100,A

BASIC:

This example will add 100 to the contents of the four byte variable A and display the result at row 24 column 5.

****

****

ROUTINE NAME: SCRLDN

BASIC EQUIVALENT: NO EQUIVALENT

FORMAT: BLWP @SCRLDN
(**NO PARAMETERS**)

DESCRIPTION: Scrolls the screen down 1 row and fills the vacated area with ASCII character 32.

EXAMPLE:

9900BASIC: BLWP @SCRLDN

BASIC:

This example will scroll the screen 1 row down.

****

****

ROUTINE NAME: SCRLLT

BASIC EQUIVALENT: NO EQUIVALENT

FORMAT: BLWP @SCRLLT
(**NO PARAMETERS**)

DESCRIPTION: Scrolls the screen to the left 1 column and fills the vacated area with ASCII character 32.

EXAMPLE:

9900BASIC: BLWP @SCRLLT

BASIC:

This example will scroll the screen 1 row to the left.

****

****

ROUTINE NAME: SCRLRT

BASIC EQUIVALENT: NO EQUIVALENT

FORMAT: BLWP @SCRLRT
(**NO PARAMETERS**)

DESCRIPTION: Scrolls the screen to the right 1 column and fills the vacated area with ASCII character 32.

EXAMPLE:

9900BASIC: BLWP @SCRLRT

BASIC:

This example will scroll the screen 1 column to the right.

****

****

ROUTINE NAME: SCRLUP

BASIC EQUIVALENT: PRINT (NOT A DIRECT EQUIVALENT)

FORMAT: BLWP @SCRLUP
(**NO PARAMETERS**)

DESCRIPTION: Scrolls the screen up 1 row and fills the vacated area with ASCII character 32.

EXAMPLE:

9900BASIC: BLWP @SCRLUP

BASIC:

This example will scroll the screen up 1 row.

****

****

ROUTINE NAME: STRLEN

BASIC EQUIVALENT: LEN

FORMAT BLWP @STRLEN
DATA LABEL,RETURN VARIABLE

DESCRIPTION: LABEL is a a label which represents the beginning of a series of memory locations that hold a zero terminated string. This routine returns in RETURN VARIABLE the length of the string.

EXAMPLE:

9900BASIC: BLWP @STRLEN
DATA A$,COUNT

BASIC : COUNT=LEN(A$)

This example returns in count the number of characters beginning at label A$ and terminated by a null.

****

****

ROUTINE NAME: SOUND

BASIC EQUIVALENT: CALL SOUND

FORMAT: BLWP @SOUND
DATA DURATION,FREQ1,VOL1,FREQ2,VOL2,FREQ3,VOL3,NOISE,VOL4

DESCRIPTION: Immediately updates the sound generator with the specifications given. Any parameter except sound may be omitted with DATA 0. Variables may not be used as sound data. Duration is limited to the values 20 through 4000 and all values given must be positive numbers. To halt program execution until the sound is completely processed (i.e. emulate positive duration sound processing in Basic) the following routine is recommended and may be called with BL @SDWAIT:

```
SDWAIT BLWP @KEY
       DATA 0,K,S
       MOVB @>83C3,@>83CE
       JNE SDWAIT
       RT
```

Please see the Editor-Assembler manual for more information on sound processing.

EXAMPLE:

9900BASIC: BLWP @SOUND
DATA 100,110,5,111,5,112,5,5,30

BASIC: CALL SOUND(100,110,5,111,5,112,5,-5,30)

This example will make a short deep tone.

****

****

ROUTINE NAME: SPRITE

BASIC EQUIVALENT: CALL SPRITE

FORMAT: BLWP @SPRITE
DATA NUMBER,CHARACTER,COLOR,Y-POS,X-POS,Y-VEL,X-VEL

DESCRIPTION: Creates a sprite with the attributes given. Please see the Extended Basic and Editor-Assembler manuals for more information on sprites.

EXAMPLE:

9900BASIC: BLWP @SPRITE
DATA 1,42,7,50,45,0,10

BASIC: CALL SPRITE(#1,42,7,50,45,0,10)

This example will create a sprite in the shape of a red asterisk at y=50, x=45 moving to the right across the screen

****

****

ROUTINE NAME: VCHAR

BASIC EQUIVALENT: CALL VCHAR

FORMAT: BLWP @VCHAR
DATA ROW,COLUMN,CHARACTER,REPETITIONS

DESCRIPTION: Places REPETITIONS number of the character CHAR vertically on the screen beginning at row ROW and column COLUMN.
EXAMPLE:

9900BASIC: BLWP @VCHAR
DATA 3,5,42,1

BASIC: CALL VCHAR(3,5,42,1)

This example will place one character described as 42 (normally an asterisk) at row 3 column 5.
****

COMPRESSED 9900BASIC OBJECT CODE

Included on your program diskette is a file named COMPRESSED. It consists of pre-assembled 9900BASIC code for use in development and with the save utility. This code is in compressed dis-fix 80 format and may not be loaded through the TI Extended Basic Loader. When using this file (i.e. during development) assemble your program with the assembler using the EQUATES file. This may be done with the directive COPY "DSK1.EQUATES" and **do not** use the copy directive to copy the 9900BASIC source file. To execute your program through the editor-assembler load and run use the following steps: load the COMPRESSED file first, followed by your program and then enter the program through your entry point (i.e. START).

EQUATES

Included on your program diskette is a file named EQUATES. This file is for use in development and with the save utility. Instead of assembling the 9900BASIC file on every assembly you may use the copy directive to copy the equates instead. When doing this make sure that you load the COMPRESSED file before your program file or your program will not execute and the console will lock up. Please note that the EQUATES file contains equates for more that just the 9900BASIC functions, it also includes the VDP access utilities and DSRLNK as well as ERFLAG.

EXTENDED BASIC SUPPORT

Included on your program diskette are the following files pertaining to support for TI Extended Basic: LOAD, LOADER, LOADS, SAVES, SAVEUTIL, COMPRESSED, and EQUATES. Please see SAVEUTIL, COMPRESSED and EQUATES sections for more information. LOAD is a boot program for use with Extended Basic and the files created with the save utility. LOAD loads the object file LOADER through the TI Extended Basic loader and links to it (via a BL instruction). LOADS is the source code for LOADER. LOADER sets up the character table in VDP ram and creates its' own utility DSRLNK in order to load the files created by the save utility. It then loads the files P1 and P2 into the high memory expansion and branches to the first instruction after the COMPRESSED file. Note that the COMPRESSED file need not be on the diskette and that this is the reason for the loading order in the save utility.

SAVEUTILITY

Included on your program diskette are two files named SAVEUTIL and SAVES. SAVES is the source code for the save utility SAVEUTIL. The entry point for SAVEUTIL is SAVE. To use the save utility (to create Extended Basic executable code) first load the COMPRESSED file, then your program file, and then the save utility. **Do not** load any files before the COMPRESSED file. Insert the diskette that your wish the object files to be written on into disk drive number one. Enter the program name as SAVE. The save utility will create two mirror image files named P1 and P2 for use with the loader on the diskette. Note that the save utility always saves the first 16K bytes of the high area of the memory expansion. Note also that the 9900BASIC COMPRESSED file is slightly less than 4K (leaving 12K of program space without modifying the save utility).

DEMONSTRATION PROGRAM

Included on your program diskette is a demostration program. The source file is named DEMO and the object file is named OBJECT. The program demonstrates many of the 9900BASIC functions and is executable through the editor-assembler load and run with the program name START. Note that the demonstration program was assembled without some of the 9900BASIC functions (to save space on the diskette) and re-assembly of it may produce a longer file.

SnapStart
A 9900BASIC Preprocessor

Introduction: SnapStart is a tool that makes it easier to convert an existing TI BASIC or Extended BASIC program into code compatible with 9900BASIC. SnapStart takes each statement of BASIC code that has a 9900BASIC equivalent and converts that line into source code that can be used with 9900BASIC. SnapStart also begins the job of placing variables in the required variable table within your source code and defining the variable type. SnapStart cannot evaluate arithmatic calculations, LET statements [including implied LET statements] or other assignment statements.

Preparing Your Program for SnapStart: Before you can run SnapStart on your program it must be in the proper format. This is best accomplished by following these steps:

1) Load existing program from disk

2) Resequence program using a starting value of one hundred (100) and an increment of ten (10).

3) Save your program to the disk as a list file {for example by typing LIST "DSK1.TEST"}, remember this file name, it is the input file used by SnapStart.

Using SnapStart: Before running SnapStart make sure you have space on the disk(s) for the output file and debug file which SnapStart generates. Together these two files will probably be about three (3) times the size of the input file generated by listing your program to disk. If SnapStart runs out of room on the disk processing will stop, however any source code generated up to that point will not be lost. You will recieve an error message to indicate any I/O error. To use SnapStart follow these steps:

1) Load SnapStart from disk [on the original disk it's file name is SNAPSTART, but it may be renamed on your own disks.]

2) Type 'RUN' to start the program.

3) The cursor will appear next to INPUT FILE:. Type the name of the file you created by listing your program and hit ENTER.

4) The cursor will now move to the OUPUT FILE: position. Type the name of the output file to be used with 9900BASIC.

5) The cursor will now move to the DEBUG FILE: line. Type the name of the Debug listing file. {Note: the debug file can be sent to a printer as well as a disk.}

6) SnapStart is now operating. It may take ten or more minutes for a twelve kilobyte (12K) program to be processed, so be patient.

Editing your source code: After SnapStart has completed processing your code, leave Extended BASIC and go to the Editor/Assembler. Using the Load option from the Editor load your output file. Now you can see the results of SnapStart at work on your program. Each line number has been converted into a label by adding an 'L' to the front. Each CALL, FOR-TO, NEXT, IF-THEN-ELSE, DISPLAY AT, and ACCEPT AT have been converted into the format required by 9900BASIC. At the bottom of the file you will find a list of the variables that SnapStart found in your program- divided into numeric variables and string variables. Your code may look almost finished, but it's probably not. The following are things to look for in your source code before attempting to compile.

Uncompiled Lines: You'll find these lines in the source code tagged by an *. These lines can also be found in the debug file. Usually this occurs because the statement was unsupported by 9900BASIC. In the debug file these lines are indicated by the message ' ? STATEMENT'. (Note: REM statements are also tagged with a *, this is the correct way to indicate a remark to the Assembler. No changes need to be made to these statments and they will not be found in the debug file)

Multi-Statement Lines: While SnapStart supports many of the commands of Extended BASIC, it does not support multi-statement lines. The best way to handle this is by eliminating multi-statement lines before saving the listing (input) file.

Multi-Parameter Statements: SnapStart does not allow the use of more than one set of parameters in a statement. For example:

```
100 CALL COLOR(2,3,12,4,4,12) would not be processed correctly, but
100 CALL COLOR(2,3,12)
110 CALL COLOR(4,4,12) would be processed correctly.
```

Formulae Within Statements: may not be used, these must be eliminated before compiling.

Repeated Variables: in the Variable table are not a fatal error, but should be deleted to prevent confusing error messages at the time of compilation.

Using the Debug File: The debug file will prove to be a great aid in finding and eliminating problems in the code. Uncompiled lines, and lines with any type of problem will be sent to the debug file in their original BASIC format along with an error message to show what SnapStart didn't like about them. In some cases you may find SnapStart overly cautious (such as throwing out an FOR-TO for lack of a STEP). In many cases SnapStart will attempt to fix the problem for you by inserting a default value, but it's still a good idea to look at the processed code and make sure it says what you want.

Note to Users: This is the first version of SnapStart and it's obviously lacking in several areas. SnapStart is being provided free with 9900BASIC in the hope that you will find it a useful tool in the 9900BASIC System. Since SnapStart is unprotected you can modify it to fit you needs. You will find the code to be in a very loosely structured arrangement, but if you follow the examples of how CALL and other statments are handled, you should be able to accomplish what you need. Please let us know whether or not you've found a use for this program and especially if you've made improvements that would be of value to others.

INPUT AND OUTPUT MODES FOR USE WITH 9900BASIC FILE HANDLING ROUTINES

9900BASIC MODE	FILE-TYPE	E-A MODE	TYPE OF FILE
0	DIS-FIX	UPDATE	SEQUENTIAL
1	DIS-FIX	UPDATE	RELATIVE
2	DIS-FIX	WRITE	SEQUENTIAL
3	DIS-FIX	WRITE	RELATIVE
4	DIS-FIX	READ	SEQUENTIAL
5	DIS-FIX	READ	RELATIVE
6	DIS-FIX	APPEND	SEQUENTIAL
7	DIS-FIX	APPEND	RELATIVE
8	INT-FIX	UPDATE	SEQUENTIAL
9	INT-FIX	UPDATE	RELATIVE
10	INT-FIX	WRITE	SEQUENTIAL
11	INT-FIX	WRITE	RELATIVE
12	INT-FIX	READ	SEQUENTIAL
13	INT-FIX	READ	RELATIVE
14	INT-FIX	APPEND	SEQUENTIAL
15	INT-FIX	APPEND	RELATIVE
16	DIS-VAR	UPDATE	SEQUENTIAL
18	DIS-VAR	WRITE	SEQUENTIAL
20	DIS-VAR	READ	SEQUENTIAL
22	DIS-VAR	APPEND	SEQUENTIAL
24	INT-VAR	UPDATE	SEQUENTIAL
26	INT-VAR	WRITE	SEQUENTIAL
28	INT-VAR	READ	SEQUENTIAL
30	INT-VAR	APPEND	SEQUENTIAL

The user should refer to the Texas Instruments Editor Assembler manual for information pertaining to file i/o as well as i/o modes and error information. Specifically sections 18.2xx .

IMPORTANT INFORMATION FOR THE USER

The **CSI DESIGN GROUP** does not warrant that the programs and data included with this package are error free or will meet the requirements of the consumer. The consumer assumes full responsibility for any decisions made or actions taken based on information obtaining while using the programs and/or documentation.

The **CSI DESIGN GROUP** makes no claims as to the suitability of this product for any purpose.

MORE ABOUT THE PROGRAMS AND DATA

We have made every effort to provide complete and accurate information and documentation, but errors do occur. In the event of a problem be it programming, lack of information, or misinformation the **CSI DESIGN GROUP** will make a reasonable endeavor to aid the user.

As this is the first release of SNAP-START it more than likely will not be as friendly as the documentation implies. It is important that you complete and return your warranty card so that updates will be made available to you.

The 9900BASIC package is not a compiler and is not to be sold or otherwise implied to be such. The 9900BASIC package is an assembly language program development aid.