Z BASIC

Version 2.2
BASIC COMPILER

JUNE 1982

Written by
Andrew Gariepy

Technical questions
Call (602) 323-9391

SIMUTEK
Computer Products Inc.

4897 E. Speedway Blvd.
Tucson, Arizona 85712
(602) 323-9391

ZBASIC 2.2

PLEASE READ THIS

#A. No LINE #0 is allowed in ZBASIC 2.2

#B. Memory size must be set before loading ZBASIC 2.2 cassette Version else unusual problems will result

Version	Memory Size Limit
ZXC16 & ZXCM16 --------	21000
ZXC32 & ZXCM32 --------	29000
ZXC48 & ZXCM48 --------	37000

#C. TIME$ will NOT function on the MODEL III computer.

#D. INPUT#1,var(,var...) in the DISK versions is allowed.

#E. All ZBASIC 2.2 DISK functions do work with NEWDOS80 2.0 except 'LOF(n)' and 'LOC(n)'.

#F. It is O.K. to reload the compiler when chaining programs when relocating but you must use the same memory size limit when re-entering the compiler.

#G. For advanced programers to find the address of the 256 byte disk buffer just use the VARPTR(var$) where var$ is the first variable fielded into the disk buffer. The FILE DCB is located 64 bytes below the DISK buffer.

#H. Note: DO NOT use variables used in a FIELD statement on a lower line # than the FIELD statement itself or strange results may result.

#I. Just a note you may not have any high-memory drivers in the computer when running the ZBASIC 2.2 compiler and you MUST set the memory size when entering BASIC or the ZBASIC 2.2 COMPILER will not run correctly.

#J. Use CAUTION not to leave GOSUB's without RETURN's as if more than about 20 of these are in your program the stack will overflow causing the computer to crash.

#K. Remember the ZBASIC 2.2 created program does not store variables the same way BASIC does so the following cautions should be used when converting programs.

a. VARPTR(var$) points to the first character in a string the string is terminated with a zero byte.
b. VARPTR(var) points to the LOW byte of an integer value
c. SPECIAL BASIC SORT machine language routines will not work with ZBASIC 2.2 strings.
d. If your program puts a string into a variable longer than the variable length is set for at the parameter section of ZBASIC 2.2 then other variables will be altered causing strange results.
e. When using @math values the shortest string length should be set to 36 characters.

#L. THE AUTO ADDITION OF THE "/CMD" HAS BEEN REMOVED TO MAKE ZBASIC MORE COMPATIBLE WITH SOME MODEL III DOSES.

STAMLER PRICES

TRS-80 MODEL-III
48K 2 DRIVES+CONTROLLER.....1899.00

APPLE II PLUS.....1249.95
1ST DRIVE (W/CONTROLLER).....589.95
2ND DRIVE.....499.95
APPLE MONITOR.....249.95
MONITOR STAND.....29.95
MICROSCI DRIVE.....W/CONTROLLER....499.95
MICROSCI DRIVE.....WO/CONTROLLER....399.95
R.H. SUPER FAN-II.....69.95
APPLE WRITER II.....149.95
GRAPPLER plus.....169.95

SMITH CORONA TP-1 12 CPS.....699.95
DAISY WHEELS.....7.95
FILM RIBBONS.....3.95
MULTISTRIKE RIBBONS.....7.95

BYTEWRITER.....795.00
BROTHER HR-1.....999.95
DAISY WHEELS.....7.95
FILM RIBBONS.....3.95

OKIDATA ML-80.....80cps.....369.95
OKIDATA ML-82.....120cps.....499.95
OKIDATA ML-83A....120cps.....799.99
OKIDATA ML-84.....200cps.....1299.99

EPSON MX/80FT.....W/GRAFTRAX.....559.95
EPSON MX/100.....799.95
EPSON GRAFTRAX.....89.95
EPSON RIBBONS.....14.95

TEC C-ITOH STARWRITER...40CPS.PRLL..1649.95
TEC C-ITOH STARWRITER...40CPS.SRLL..1799.95
TEC C-ITOH PROWRITER..100CPS.SRLL.DM.649.95
TEC C-ITOH PROWRITER..100CPS.PRLL.DM.499.95
RIBBONS.....PROWRITER.....12.95
RIBBONS.....STARWRITER.....19.95
DAISY WHEELS.....12.95

KAYCOMP II.....1789.00

SANYO.....COLOR.....499.95
BMC.....B/W.....109.95
BMC.....COLOR.....299.95
ZENITH.....GREEN.....149.95
APPLE-III.....GREEN.....149.95
USI.....AMBER.....189.95
AMDEK VIDEO 100.....GREEN.....97.50
BMC (RED GREEN BLUE) "COLOR".....599.95

SIMUTEK
Computer Products Inc.

April 17, 1982

To Whom it may concern,

Simutek Computer Products Inc. and Andrew Gariepy, the sole owners of the copyrighted program ZBASIC, authored by Andrew Gariepy, hereby state publicly that all or any programs created with ZBASIC shall be sold or bought without royalty payment to Simutek Computer Products Inc. or it's author. However, any programs compiled using ZBASIC shall state at the start of the program for at least ONE second , and in the documentation:

THIS PROGRAM WAS COMPILED USING THE ZBASIC COMPILER. COPYRIGHT SIMUTEK COMPUTER PRODUCTS INC. and ANDREW GARIEPY.

The sale, distribution or giving away of ZBASIC documentation or programs is strictly forbidden and unlawful and does great damage to both Simutek Computer Products Inc. and the author. Such action or actions shall be subject to immediate criminal prosecution. The originally purchased ZBASIC is for one user only and all backups and copies shall remain the property of that person. All ZBASIC programs contain the serial number of the user. Found unauthorized copies shall dictate immediate and harsh legal action.

Simutek Computer Products Inc. and Andrew Gariepy shall have no responsibility for any program or programs created using ZBASIC. Simutek Computer Products Inc. and Andrew Gariepy shall have no liability to persons or companys that buy or sell programs made with ZBASIC. Simutek Computer Products Inc. and Andrew Gariepy shall have no responsibility or liability to anyone for anything whatsoever concerning ZBASIC compiled programs or ZBASIC.

Signed _____
Michael A. Gariepy
President
Simutek Computer Products Inc.
4897 E. Speedway Blvd.
Tucson, Arizona 85712
(602) 323-9391

TABLE OF CONTENTS

INTRODUCTION

ZBasic 2.2 is the latest version of the ever-popular ZBasic compiler. This version is the most extensive to date, with random access file handling, improved high-precision math and numerous other additional features.

However, regular Basic and ZBasic are different in some respects, and you should be aware of these differences before you begin to use ZBasic. This manual will detail all the differences between the two and will offer you aid in converting programs.

The ZBasic Diskette

The ZBasic diskette is a formatted single density diskette with six files on it. The diskette does not have a DOS on it. However, the diskette does have a file transfer utility. This utility will enable Mod I users that have only 1 drive to transfer ZBasic onto a system diskette.

These are the files that are on the diskette:

ZXC32/CMD	32K ZBasic
ZXCD32/CMD	32K ZBasic with disk I/O supported
ZXCDM32/CMD	32K ZBasic with disk I/O and high-precision math supported
ZXC48/CMD	48K ZBasic.
ZXCD48/CMD	48K ZBasic with disk I/O supported
ZXCDM48/CMD	48K ZBasic with disk I/O and high-precision math supported
CMDFILE/CMD	Machine language merge utility

Model I Loading Procedure

The ZBasic diskette is a formatted diskette with the ZBasic files but no DOS. To transfer the files using a two-drive system, put a DOS diskette in drive Ø, the ZBasic diskette in drive 1, and copy the files over.

Since this method won't work on a one-drive system, we have included a file transfer utility. If you have only one drive, put the ZBasic diskette into your drive and boot the system. The file loader program will come up and you'll see a directory listing and transfer information. The utility will then ask for the destination drive number. Since you are transferring using only one drive, hit 'Ø'. The program will then ask you to mount the destination diskette. Any disk with a DOS and plenty of free space will do. Just stick this diskette into the drive and go from there. The utility will continue to prompt you to mount either the destination or the source diskette until all the files have been copied over.

Model III Loading Procedure

To transfer ZBasic to a Model III diskette you have to use the CONVERT utility; to use the convert utility you must have two drives. If you don't have access to a two drive Model III, return the ZBasic diskette along with a diskette with DOS on it and we'll convert it for you.

If you have access to a two drive Model III, insert your ZBasic diskette into drive 1 and a diskette with DOS and plenty of free space into drive Ø. Then run the CONVERT utility. The source drive is 1 and the destination drive is Ø. The CONVERT utility will then copy all the versions of ZBasic down to your Model III diskette (provided you have enough room on the diskette).

Loading from Tape

There are six separate ZBasic tape versions.

Tape Filename	Version Description
Side One	
ZXC16	16K ZBasic.
ZXC32	32K ZBasic.
ZXC48	48K ZBasic.
Side Two	
ZXCM16	16K ZBasic with high-precision math.
ZXCM32	32K ZBasic with high-precision math.
ZXCM48	48K ZBasic with high-precision math.

(Model III users should note that the ZBasic tape is recorded at low cassette speed.)

All the versions are loaded by typing 'SYSTEM' from Basic and then typing the name of the version. After the tape has loaded, type a slash and ZBasic will come up.

Running ZBasic

To load ZBasic into memory on a disk system, simply type the name of the version you wish to use. Loading on a tape system was explained above.

After ZBasic has loaded into your machine, the screen will look somewhat like this:

```
*******************************
* ZBASIC 2.2 BASIC COMPILER   *
* Read manual before changing *
* any of the parameters!!     *
* Copyright 1982 SIMUTEK INC. *
*******************************
Memory size limit 41ØØØ or less
Change parameters (Y/N) ?
```

(The memory size limit will vary from version to version.)

ZBasic is asking you if you wish to change any of the parameters. These parameters are the various values associated with ZBasic, such as string length etc. Hit 'Y' so that you can get a glimpse at the parameters and their current values. Simply hit the return key to go on to each new parameter.

The first parameter you can change is the array string length. If you change this parameter, each element in a string array will take up the number of bytes that you indicated. The normal string size can also be changed. Similarly, if you change the normal string size parameter, all non-array strings will take up the number of bytes that you indicated. These two commands are especially useful when memory is at a premium. Reducing these two parameters can buy the programmer extra memory.

The 'Top of Variable RAM' and the 'Base Address of Object' parameters will be explained in the relocation section of the manual. Ignore them for now.

ZBasic also allows you to specify the maximum number of characters that can be typed into a string when the INPUT statement is used. For example, if you specify a value of 5Ø for this parameter, ZBasic will not allow any more than 5Ø characters to be typed into an INPUT string. The 51st character will not echo to the screen nor will it be put into the string.

The 'Warm or Cold' start parameter is used for chaining programs. It will be explained later.

The memory size limit (41ØØØ in the above example) is important; remember it for later reference.

After you have looked at all the parameters, the 'Change Parameters' message will pop up again. If you wish to change any parameters hit 'Y'. To replace a parameter value, simply type in the new value. To leave a value as it is, just hit return.

If you are using a disk system, you should now load disk Basic. **Remember to set the memory size to the limit discussed earlier.** (For example, some Basics ask for a 'Mem Size' before you enter Basic. So, instead of just hitting the enter key, you would type in the memory

size limit. Other Basics will require that you type the size limit on the same line as the command to enter Basic. Check your DOS manual for specifics). It can't be emphasized enough: you must set the memory size limit or Basic will clobber the memory where ZBasic resides.

Now you are ready to use ZBasic.

ZBasic Theory

ZBasic and Basic are co-resident; they are both in memory at the same time. This is the ultimate in convenience because it eliminates the need to be continually loading programs.

To use ZBasic, first load or type a program into regular Basic. You might then edit this program some. When you're ready to compile the program, hit the 'Z','X' and 'C' keys simultaneously. Provided there were no errors in the program, you'll see a display like this:

```
Compiled by ZBasic 2.2
SIMUTEK INC. TUCSON,AZ
(C) 1982 Andy Gariepy
Pass 1,Pass 2,Pass 3
Compiled Size   Basic size     Variable size
 Ø2688            ØØ352           ØØ288
(R)un (B)asic (S)ave :
```

At this point you have three choices: you can hit the 'B' key to return to Basic, you can hit the 'S' key to save the machine language version of your program, or you can type 'R' to run the machine language version of your program.

Hitting the 'B' key will dump you right back into Basic. Your program will still be there. At this point you could edit your program and then jump back to ZBasic and compile again.

Hitting the 'S' key will allow you to save the machine language version of your file. Disk users need not specify the '/CMD'; ZBasic will automatically add it on. If the file already exists on disk, ZBasic, as a precaution, will ask you if you want to replace it. If the file doesn't exist ZBasic will ask you if you want to create it. Tape users need only specify a valid filename.

Watch out for error messages when using the compiler. One error message that may pop up is the 'Variable/Code Conflict' message. This message means that either the program or the variables are too large. You'll either have to reduce the size of the program or the variables (and sometimes both). Another solution might be to relocate the program. Both the error messages and the relocation feature are explained later on in the manual.

Variables in ZBasic

ZBasic allows two types of variables: string and integer. (ZBasic's high-precision math variables are stored as strings. This will be covered later.)

Unlabeled variables or variables followed by a '%' sign are integers. ALL other variables (variables followed by '#','$' or '!' signs) are strings.

Definitions

Below are some terms that are used to explain statement syntax throughout this manual. Make sure that you are familiar with the difference between simple and complex strings.

'expr'	Integer Expression. Examples include: 100,A+B*(4*6+4/4) ABS(INT(RND(1)))
'numb'	Integer number from 0 to 65535 or -32678 to 32767 (or from hex %0000 to %FFFF.)
'byte'	Integer number from 0 to 255 or from %00 hex to %FF hex. If the number to be evaluated as a byte is larger than 255, the lower 8 bits will be used.
'sstr'	Simple string. Examples include: "HELLO",CHR$(43),A$ A$(8),INKEY$ A complex string is a string that contains functions or a string expression in general. All the following strings are complex strings: RIGHT$(A$,3),LEFT$(B$,5),A$+B$
'var'	Variable name. Examples include: A$,B#,D$,E!
'fnum'	Disk file number (1-8)

Note: The phrase 'physically before' means the statement must appear on a lower line number or earlier in a program line than the parameter being defined.

KEYWORDS

The following pages are an alphabetically ordered list of ZBasic keywords and the functions they perform. Some statements will be exactly or nearly exactly the same as their Basic counterparts. These statements will be touched upon only lightly. Special ZBasic keywords will be explained at more length.

A (D) means the keyword is available in disk ZBasic only.
An (M) means the keyword is available in @Math ZBasic only.

ABS (expr)

Removes the sign (if negative) of the expression in the parentheses. Note that the absolute value of -32768 is Ø.

AND

Logical AND of two 16 bit expressions.

ASC (sstr)

Returns the ASCII code of the first character of the simple string. A null string returns Ø.

ATN ( )

Not supported by ZBasic. See page 53.

AUTO (bits,dur,freq)

A tone generation statement. 'Bits' is a tone parameter (usually 256), 'dur' is the duration (relative from 1 to 1ØØ, no relation to any specific time unit), and 'freq' is the frequency (from 1ØØHz to 15ØØØHz).

Example: 1Ø AUTO (256,2Ø,1ØØØ)

This line will play a short 1KHz 'beep.'

CDBL (#bytes,key,start)

A search function. '#bytes' is the number of bytes to search, 'key' is the particular byte being searched for, and 'start' is the starting memory location of the search. The search will begin at 'start' and will count backward (start, start-1, start-2, etc.) until '#bytes' have been searched. The argument returned will be the last location searched minus one. The returned value will either be the memory

location before the located key or it will be the end of the list, indicating the key was not found. (This is the same as the CPDR Z-8Ø instruction)

Example: 1Ø X=CDBL(16ØØØ,ASC("A"),16ØØØ)

This line searches backwards through the first 16K for the letter 'A'.

CHR$ (expr)

Will convert an integer expression to the matching ASCII character.

CINT (#bytes,key,start)

Same as CDBL except the search counts forward (start, start+1, start+2, etc.) until '#bytes' have been searched. The argument returned will be the last location searched plus one. The memory location returned will either be the byte after the located key or the end of the list, indicating the key was not found. (This is the same as the CPIR Z-8Ø instruction).

Example: 1Ø A=CINT(16ØØØ,ASC("A"),Ø)

This line searches forward through the first 16K for the letter 'A'.

CLEAR expr

Clears all variables to either nulls or Ø as appropriate. ZBasic ignores the expression after the CLEAR statement.

CLOAD #bytes,start

Cassette load statement. Loads '#bytes' from tape into memory starting at location 'start'. '#bytes' must be less than or equal to 256.

Example: 1Ø CLOAD 255,VARPTR(A$)

The above line loads 255 bytes from tape into A$.

CLOSE fnum(,fnum...)

Close disk files. The file numbers must be included.

CLS

Clears the video display screen and resets the 32 character mode.

CMD sstr

Executes the DOS command in the simple string. Available to Newdos 8Ø Version 2 users exclusively.

Example: 1Ø CMD "DIR"

This line will execute the Newdos 8Ø Ver 2.Ø directory command. DOSPLUS users: see the NAME command.

CONT (expr1,expr2)

Returns the actual video MEMORY address of the screen at that point; 'expr1' and 'expr2' are graphics point coordinates.

Example: Z = CONT(X,Y)-15360

This line will set Z equal to the PRINT @ position of the graphic point at (X,Y).

COS

Not supported in ZBasic. See pages 52 and 53.

CSAVE #byte,start

Cassette save statement. Saves '#bytes' starting at memory location 'start' to tape. '#bytes' cannot be larger than 255.

Example: 1Ø CSAVE 25,VARPTR(A$)

This line will dump 25 characters from A$ to tape.

CSNG (expr1,expr2,expr3)

Machine language call function. 'Expr1' is the address to call, 'expr2' will be passed to both the DE register pair and to the A register, and 'expr3' will be passed to HL. This function returns the HL register pair.

Example: 1Ø X=CSNG(%33,65,Ø)

Call ROM location 33 hex to print the letter 'A.'

CVD (sstr) (D,M)

Takes the simple string (usually associated with a FIELD statement) and converts it into a ZBasic @Math string.

Example: @X$=CVD(B$)

Convert B$ (normally a FIELDed variable) to an @Math string.

CVI (sstr) (D,M)

Takes the simple string (usually associated with a FIELD statement) and converts it into a ZBasic integer variable.

Example: 1Ø X=CVI(B$)

Convert B$ (normally a FIELDed variable) to an integer.

CVS (sstr) (D,M)

Takes the simple string (usually associated with a FIELD statement) and converts it into a ZBasic @Math string. See CVD for an example. CVD and CVS perform identical functions. Both are supported to aid in converting Basic syntax to ZBasic syntax.

DATA numb,"string",numb,numb,....

Holds data to be read by READ statements. Strings must have quotes around them. Be careful not to read data in the wrong order.

DEFUSR

Not supported by ZBasic.

DEFFN

Function definitions are not supported by ZBasic.

DEFB

Hex byte format specifying statement. All subsequent output statements (PRINT, PRINT # etc.) will output the low byte of an integer in hex form. If the integer is greater than 255, only the low order byte will be output.

Example: 1Ø DEFB:PRINT 255

The above line will print 'FF' on the screen.

DEFN

Standard integer format specifying statement. All subsequent output statements (PRINT, PRINT #, etc.) will output the integer as a signed number between -32768 and 32767.

DEFU

Decimal integer format specifying statement. All subsequent output statements (PRINT, PRINT #, etc.) will output the integer as an unsigned number between ØØØØØ and 65536.

DEFW

Hex format specifying statement. All output statements (PRINT, PRINT #, etc.) will output the integer as a hex word (ØØØØ to FFFF).

Example: 1Ø DEFW:PRINT 65535

This line will print the hex number 'FFFF' on the screen.

DEFDBL var or variable range

Define specified variable(s) as STRINGS. (This is done because ZBasic does all its floating point math in strings). Must appear physically before any of the variables it defines.

Example: 1Ø DEFDBL A-M:@F="1ØØ"+"35Ø.9Ø"

This line will add 1ØØ and 35Ø.9Ø into the @Math string F.

DEFINT var or variable range

Define specified variable(s) as integers. Must appear physically before any of the variables it defines.

DEFSNG var or variable range

Define specified variable(s) as strings. (See DEFDBL). Must appear physically before any of the variables it defines. See DEFDBL for an example.

DEFSTR var or variable range

Define specified variable(s) as strings. Must appear physically before any of the variables it defines. See DEFDBL for an example. DEFSNG and DEFSTR perform functions identical to DEFDBL. They are supported only as an aid in converting Basic to ZBasic.

DELETE expr

Time delay statement. Expression evaluates to the number of milliseconds to be delayed. (1000 milliseconds = 1 second).

Example: 10 DELETE 1000

This line will delay the program for one second.

DIM var(num(,numb..))

Dimension variable(s) for use as an array. The number of elements has to be a number, NOT an expression. The DIM statement must appear physically before any of the variables it dimensions.

Example: 10 DIM A$(25),B$(3,30)

EDIT (bits,cycles,period)

Tone generation statement. 'Bits' is a tone parameter (between 0 and 65535), 'cycles' is the number of cycles sounded (between 1 and 65535) and 'period' is the relative time between each cycle (between 1 and 65535).

Example: 10 EDIT (256,100,50)

ELSE

Keyword to be used in an IF statement for evaluation if the statement is false. ELSE can specify a line number or just continue on with another statement. ELSE cannot be in the last line of a program.

END

This statement ends the program and returns control to DOS.

EOF (fnum) (D)

DOS error function. This function returns the number of the last DOS error. Can be used as and end-of-file detector by checking each time a file is read whether or not a 'Read Past End of File' error has been detected.

Example: 1Ø IF EOF(1)=27 THEN PRINT "Disk Full"

This line checks for a DOS error. If the error is number 27, 'Disk Full' will be printed on the screen.

ERL (#bytes,dest,source)

Memory move statement. Move '#bytes' starting at 'source' to memory location starting at 'dest.' This routine copies memory location 'source' to location 'dest,' 'source+1' to 'dest+1' and so on until '#bytes' have been copied. (Same as the LDIR Z-8Ø instruction).

Example: 1Ø POKE 15360,191:ERL(1Ø23,15361,15360)

The above line will white out the screen.

ERR (#bytes,dest,source)

Memory move statement. Same as ERL except this statement copies from 'source' to 'dest,' 'source-1' to 'dest-1' until '#bytes' have been copied. The progression is down through memory instead of up through memory. (Same as LDDR)

Example: 1Ø POKE 16383,191:ERR(1Ø23,16382,16383)

The above line will white out the screen.

ERROR expr (D)

DOS error statement. Prints the full error message for the DOS error which 'expr' evaluates to.

Example: 1Ø ERROR 27

Prints 'Disk Space Full' on the screen.

EXP ( )

Not supported by ZBasic. See page 53.

FIELD fnum, byte AS var, ... (D)

Disk file FIELDing statement. 'Fnum' cannot be an expression and the 'var'iables have to be strings. (String arrays are not permitted). This statement must appear physically before any of the variables used in the statement.

FIX (#bytes,bits,start)

Memory 'XOR' function. This function XOR's the bit pattern 'bits' against memory starting at 'start' until '#bytes' have been XOR'ed. Using 63 as 'bits' is a good way to invert the graphics screen.

Example: 1Ø X=FIX(1Ø24,63,15360)

The above line will 'invert' the graphics on the screen.

FN

Remainder function. For example, the statement 'A = 1Ø FN 3' will result in setting A to 1. Three goes into ten 3 times with a remainder of **one.**

FOR var = expr TO expr (STEP numb)

Iteration statement. The largest value that can be counted to is 32767 minus the amount of the step. FOR loops use integer values only.

FRE (expr)

Argument passing function. This function allows a ZBasic program to be called as a USR routine from a Basic program. Here is an example to explain the use of this function:

```
1Ø REM   BASIC PROGRAM
2Ø REM
3Ø DEFUSRØ=&HFØØØ  <-- Address of ZBasic routine
4Ø X=USR(9999)     <-- Jump to routine and pass '9999'
5Ø PRINT "Value from ZBasic routine is ";X
6Ø END

1Ø REM   ZBASIC PROGRAM
2Ø REM
3Ø BV=FRE(12345)   <-- Get '9999' from Basic and pass '12345'
4Ø PRINT "Value passed from Basic :";BV  <-- Will be '9999'
5Ø RETURN
```

The ZBasic program can only read the passed value with the FRE function one time. If the statement '35 P=FRE(12345)' were inserted into the program, P would be equal to 12345 and not 9999.

GET (#)fnum(,expr) (D)

Get the record number 'expr' from file number 'fnum.'

GOSUB numb

Branch to line number 'numb' and return to the next statement upon encountering a RETURN statement. 'Numb' may not be an expression.

GOTO numb

Branch to line 'numb.' 'Numb' may not be an expression.

IF expr THEN <numb or statements>

Condition statement. Identical to Basic except that only simple strings are allowed to be used in IF statements. (Simple strings are defined at the beginning of the manual)

INKEY$

Keyboard scan function. INKEY$ returns the character string of the key currently being pressed. If no key is pressed, INKEY$ returns a null.

INP (expr)

Get the value of the input port specified by 'expr.'

INPUT ("<message>";)var(,var)

Input data from the keyboard. If a string is being input, it must be the only variable input.

```
Example: 1Ø INPUT A$,A
         2Ø INPUT A$:INPUT A
```

Line 1Ø is illegal because A$ is not the only variable. Line 2Ø is legal.

INPUT @expr,("<message>";var(,var)

Same as INPUT except the prompt starts at the screen location 'expr.'

INSTR (expr,sstr,sstr)

String search function. Search the first string to see if it contains the second string. Start the search 'expr' characters from the left. The 'expr' is MANDATORY. This function returns a zero if the second string is not contained within the first.

Example: 1Ø A$=LEFT$(X$,INSTR(X1,",",IN$))

Their must be a variable or expression where X1 is.

INT (expr)

Ignored by ZBasic. See page 41.

KILL sstr

Kill the disk file specified in the simple string.

LEFT$ (sstr,expr)

Take the amount of characters specified in expr from the left side of the string.. The string argument must be a simple string.

LEN (sstr)

Function to return the length of the simple string.

LET

Optional variable assignment statement.

LINEINPUT #fnum,var (D)

Input a string from file 'fnum' until a cr/lf sequence is input or 255 characters are read from the file. This is a disk I/O statement only. Use the regular INPUT statement for keyboard line input.

LIST

Not supported by ZBasic. (LIST from Basic only)

LLIST

Not supported by ZBasic. (LLIST from Basic only)

LOAD sstr (D)

Load the machine language file specified by the simple string into memory. The loaded file must not conflict with the program that loads it.

Example: 1Ø LOAD "DATA/CIM"

The above line loads the core image file 'DATA/CIM' into memory.

LOC (fnum) (D)

This function returns the present record number. Note that this function may not work with some DOSes (specifically Newdos 8Ø Ver 2.Ø).

LOF (fnum) (D)

This function returns the last record in a file. Note that this function may not work with some DOSes (specifically NEWDOS Ver. 2.Ø).

LOG ( )

Not supported by ZBasic. See page 52.

LPRINT <prompts, messages and/or variable(s)>

Statement to output information to the line printer.

LSET sstr = sstr (D)

Load a string into a fielded buffer. The string on the left must have been FIELDed earlier and the string on the right has to be a simple string.

Example: LSET A$=B$

Load B$ into the FIELDed string A$.

MEM

This function returns the address where the currently compiled code will be. For more information see the special MERGE section.

Example: 1Ø DEFW:PRINT MEM:DEFN

The above line (placed at the start of a ZBasic program) will print the present address of the program in hex.

MERGE

See the special MERGE section on page 24.

MID$(sstr,start,#char)

Function to take a portion of a string. The function will return the piece of the simple string that begin at 'start' and is '#char' characters long.

MID$(sstr,start,#char) = sstr

MID$ assignment statement. The sstr on the right side of the equals sign (or as much as can fit) will be inserted into the left hand simple string starting at the portion specified by the MID$ argument.

MKD$ (sstr) (M,D)

This function converts an @Math string into the format used by FIELDed variables. The function can only be used with FIELDed variables.

Example: @B$=MKD$(A$)

The FIELDed variable B$ is loaded with the @Math string A$.

MKI$ (expr) (M,D)

This function convert an integer expression into the format used by FIELDed variables. This function should be used with FIELDed variables only.

Example: @B$=MKI$(A+2)

The FIELDed variable B$ is packed with the expression A+2.

MKS$ (sstr) (M,D)

This function converts an @Math string into the format used by FIELDed variables. The function should only be used with FIELDed variables. See MKD$ for an example.

NAME sstr (D)

This statement exits the ZBasic program and executes the DOS command in the simple string. This function will not return control to the ZBasic program after the command is executed except in DOSPLUS.

Example: 1Ø NAME "DIR"

NEW

Not supported by ZBasic.

NEXT (var,(var,...))

End of a loop started with a FOR statement. Each loop in a ZBasic program must consist of one FOR statement and one NEXT statement. There cannot be multiple NEXT statements for one FOR statement.

NOT (expr)

Logical NOT operator. This operator requires that the expression have parentheses around it.

ON expr (GOTO or GOSUB) numb,numb,numb

Branch control to the indicated line depending upon the value of the expr. A 1 will go to the first number, 2 to the second, and so on.

OPEN "I or O or R",fnum,sstr (D)

Open a disk file for I/O. If the file is non-existent, the 'O' option will create it, the 'I' and 'R' options will not create a new file.

OR

Logical integer OR operator.

OUT expr1,expr2

Send to output port 'expr1' the ASCII code 'expr2.'

PEEK (expr)

This function retrieves the value of the memory location defined by 'expr.'

PEEKW (expr)

This function retrieves the 16 bit word starting (in low-high order) at the memory location defined by 'expr.'

```
Example: 10 X1=PEEK(19001)*256+PEEK(19000)
         20 X2=PEEKW(19000)
```

The variables X1 and X2 will be indentical. Line 20 executes 50 times faster.

POINT (expr1,expr2)

This function returns a -1 if the graphics point at coordinate (expr1,expr2) is on; 0 if it is not.

POKE expr1,expr2

This statement puts 'expr2' into memory at memory location 'expr1.'

POKEW expr1,expr2

This statement puts word 'expr2' into memory (in low-high order) at location 'expr1.'

```
Example: 10 POKEW 15360,-32767
```

This line will poke FF into address 15360 and 7F into address 15361.

POS(0 or 1)

Print or screen position function. Used with a 0 argument, the position of the print head is returned. Used with a 1, the PRINT @ position (0-1023) is returned.

```
Example: 10 LPRINT TAB(POS(0)+10);"HELLO"
```

The above line will print 'HELLO' ten characters from the current printer carriage position.

PRINT (@expr,)<prompts, messages, @exprs and/or variable(s)>

Output information to the screen.

PRINT #fnum,<prompts, messages and/or variable(s)> (D)

Output information to disk file number 'fnum.'

PUT (#)fnum (,expr) (D)

Put the contents of the file buffer for file 'fnum' into the disk file at record 'expr.'

RANDOM

This statement reseeds the random number generator.

READ var(,var)

Read the information from a DATA statement into the variables indicated. Make sure that your ZBasic program reads the data in the correct order and that you have quotes around all strings that are in DATA statements.

REM remarks

This statement leaves the rest of the line open for program remarks.

RESET (expr1,expr2)

This statement turns the graphics dot at coordinate (expr1,expr2) off.

RESTORE

This statement will reset internal pointers so that any subsequent READ statements will start with the first DATA statement. This statement should be used to reset the internal pointer if you READ data in a program using ZBasic's warm start feature.

RESUME (D)

This statement will call the system debugger (on a disk based system). Hit 'G' in DEBUG to return to the ZBasic program.

RETURN

This statement returns from a previous GOSUB statement. This statement is also used as a return to a Basic program from a ZBasic/machine language subroutine.

RIGHT$ (sstr,expr)

This function returns the rightmost 'expr' characters of the simple string.

RND (expr)

This functions returns a random number between 1 and 'expr'.

RSET var1 = var2 (D)

This functions converts the FIELDed string 'var2' into a normal string 'var1.' FIELDed strings cannot be used until they have been converted in this manner.

Example: 1Ø RSET A$=B$

The string is loaded with the FIELDed variable B$.

RUN sstr (D)

Run the program whose name is in the simple string. The program has to be a /CMD file; and ZBasic does not assume the /CMD.

Example: 1Ø RUN "PROGRAM/CMD"

This line will load and run the machine language file 'PROGRAM/CMD.'

SAVE

See page 3 for instructions on saving ZBasic programs.

SET (expr1,expr2)

Turn the graphics dot specified by coordinate (expr1,expr2) on.

SGN (expr)

This function returns a -1 if 'expr' is negative, Ø if it is zero and 1 if it is positive.

SIN ( )

Not supported by ZBasic. See page 52.

SQR (expr)

Returns the square root of the integer expression.

STEP numb

This keyword sets the iteration amount in a FOR loop. The amount cannot be an expression or a variable.

STOP

This statement ends execution of the ZBasic program and returns control to Basic.

STR$ (expr)

This function creates a string out of the integer number that the expression evaluates to.

STRING$ (expr1,(sstr or expr2))

This function creates a string 'expr1' characters long composed of either the simple string or the ASCII code defined by 'expr2.'

SYSTEM (DOS or BASIC or numb)

This statement scans the keyboard and branches to DOS, Basic, or line number 'numb' if the break key has been hit.

Example: 1Ø SYSTEM BASIC
2Ø GOTO 1Ø

This program will loop until the break key is hit, at which time it will return to Basic.

TAB (expr)

This statement tabs to the column indicated by the expression. Ø-63 on the screen, Ø-255 on the printer.

TAN ( )

Not supported by ZBasic. See page 52.

TIME$ (D)

This function returns the date and time on a disk-based system.

TRON

Turn the tracing feature on (from this point physically downward).

TROFF

Turn the tracing feature off.

USING sstr;sstr (M)

See the PRINT USING section on page 27.

USR (expr)

This calls a machine language subroutine and passes the expression to the HL register pair. This function returns the value of HL. The address to call is poked into locations 16526 (the low byte) and 16527 (the high byte.)

VAL (sstr)

This function converts the ASCII number in the simple string into a regular integer number. The ASCII number may be hex, decimal, or unsigned.

```
Example: 1Ø X=VAL("1Ø"):Y=VAL("%ØA"):IF X=Y THEN PRINT"="
```

This line will print an equals sign on the screen.

VARPTR (var)

This function returns the memory location of the VALUE of the variable. It does not return the location of the variable name.

```
Example: 1Ø X=VARPTR(A$):Y=VARPTR(Z)
```

This line will put the address of the first character of A$ into the variable X and the address of the low byte of integer variable Z into the variable Y.

INSERTING MACHINE LANGUAGE CODE INTO A ZBASIC PROGRAM

The MERGE statement allows the insertion of a machine language program or routine within a ZBasic program. The code or bytes are inserted directly inline with the normal Z-80 code created by ZBasic. This function also allows access to ZBasic variables and a simple address relocator for direct calls and jumps within the created machine language program. Just remember that this function can also cause crashes if used incorrectly. Carefully test your machine language programs before using them. If you don't have experience with machine language, it might be best to leave the MERGE command alone.

As an example, this MERGE statement ....

```
MERGE %CD,MEM+n,%21,65535,%2A,XX$,%21,XY$.%01,1000,%C9
```

Makes the following machine language program.

```
        CALL    MEM+n           ;n MUST BE A number
MEM:    LD      HL,65535        ;if n=0
        LD      (VARXX$),HL     ;if n=3
        LD      HL,VARXY$       ;if n=6
        LD      BC,1000         ;if n=9
        RET
```

Using MERGE with the TRON statement

The TRON statement when active adds 6 bytes of code at each line in the program. Make sure this will not conflict with merge programs before using this function. You can disable TRON before and re-enable TRON after each MERGEed machine language program to get by this problem.

Examples of MERGE usage

ZBasic line	Machine language created
MERGE %FB	EI ;Enable interrupts
MERGE %F3	DI ;Disable interrupts
MERGE %31,%C000	LD SP,0C000H ;Load stack pointer
MERGE %ED,%7B,SP	LD (VARSP),SP ;Load SP with stack pointer
MERGE %2A,SP,%F9	LD SP,(VARSP) ;Set stack pointer to SP

```
MERGE %21,%3CØØ                 LD   HL,3CØØH   ;Start address of screen
MERGE %11,15361                 LD   DE,15361   ;Start +1 address
MERGE %Ø1,%3FF                  LD   BC,3FFH    ;# of bytes
MERGE %36,191                   LD   (HL),191   ;Byte to fill screen with
MERGE %ED,%BØ                   LDIR            ;Block move/fill
-------------------------------------------------------------------------

MERGE %ED,%5F                   LD   A,R        ;Read refresh reg
MERGE %32,A                     LD   (VAR A),A  ;Set variable A to it
-------------------------------------------------------------------------

MERGE %2A,X                     LD   HL,(VAR X) ;Get value of variable X
MERGE %CB,%2C                   SRA  H          ;Shift HL right 1 bit
MERGE %CB,%1D                   RR   L          ;Very fast divide by 2
MERGE %22,X                     LD   (VAR X),HL ;Set new value of X
-------------------------------------------------------------------------

MERGE %76                       HALT            ;Hardware reset Model I
-------------------------------------------------------------------------
```

The following example is a simple clock program for the Model I.

```
1ØØØ  MERGE %F3,%21,MEM+1Ø          ;On interrupt goto function
1Ø1Ø  MERGE %2A,%4Ø13,%3E,%C3       ;Make JP
1Ø2Ø  MERGE %32,%4Ø12,%FB,%C9       ;And turn on interrupts
1Ø3Ø  MERGE %F5,%C5,%D5,%E5         ;Interrupt service start
1Ø4Ø  MERGE %DD,%E5,%FD,%E5         ;Save all registers
1Ø45  IF PEEK(%37EØ)AND64THEN1Ø9Ø   ;FDC interrupt
1Ø5Ø  T=T+1:IFT>4ØTHENT=ØELSE11ØØ   ;T = tick counter
1Ø6Ø  S=S+1:IFS>59THENS=ØELSE11ØØ   ;S = seconds
1Ø7Ø  M=M+1:IFM>59THENM=ØELSE11ØØ   ;M = minutes
1Ø8Ø  H=H+1:IFH>23THENH=ØELSE11ØØ   ;H = hours
1Ø85  GOTO11ØØ
1Ø9Ø  XX=PEEK(%37EC)                ;Clear FDC interrupt
11ØØ  XX=PEEK(%37EØ)                ;Reset interrupt latch
1111  MERGE %FD,E1,%DD,%E1          ;POP IY,IX
1121  MERGE %E1,%D1,%C1,%F1          ;POP HL,DE,BC,AF
1131  RETURN                        ;RET
```

The following example is a small delay routine using the MERGE feature.

```
10 PRINT"Example delay routine"
20 FOR I = 1 TO 100
30 D=100 : GOSUB 1000
40 NEXT I
50 STOP
1000' Remark statements do not add any bytes or delays to
1010' ZBasic programs use them freely to document
1020' your ZBasic programs.
1030 MERGE %2A,I'          LD     HL,(VARD)
1040 MERGE %2B   ' DECHL:  DEC    HL
1050 MERGE %7C   '         LD     A,H
1060 MERGE %B5   '         OR     L
1070 MERGE %20,%FB'        JR     NZ,DECHL
1080 MERGE %C9'            RET
```

The following is a ZBasic MERGE program with several subroutines.

```
10 CLS : '---- Merge example program with many subroutines ---
20 PRINT"This is just a dumb example program"
30 INPUT"Input me HL,DE,BC,A ";HL,DE,BC,A
40 GOSUB 1000'----- Move ZBasic variables to Z80 reg's -----
50 MERGE %CD,%33,0'-- Call your routine here --
60 GOSUB 2000'----- Move Z80 reg's back to ZBasic Variables --
70 DEFW : PRINT"HL REG AFTER ROUTINE IS ";HL
80        PRINT"DE REG AFTER ROUTINE IS ";DE
90        PRINT"BC REG AFTER ROUTINE IS ";BC
100       PRINT"A  REG AFTER ROUTINE IS ";A
110 GOTO 10              'JP    LINE 10
1000'
1010'Machine language program to set main Z80 reg's with
1020'ZBasic variables.
1030'
1040 MERGE %2A,HL        'LD    HL,(VAR HL)
1050 MERGE %ED,%5B,DE    'LD    DE,(VAR DE)
1060 MERGE %ED,%4B,BC    'LD    BC,(VAR BC)
1070 MERGE %3A,A         'LD     A,(VAR A)
1080 RETURN              'RET
2000'
2010'Machine language to read present Z80 reg's and put then
2020'into ZBasic program variables.
2030'
2040 MERGE %32,A         'LD    (VAR A),A
2050 MERGE %ED,%43,BC    'LD    (VARBC),BC
2060 MERGE %ED,%53,DE    'LD    (VARDE),DE
2070 MERGE %22,HL        'LD    (VARHL),HL
2080 MERGE %C9           'RET
```

Turn to page 56 for another sample MERGE program.

ZBASIC PRINT USING

The ZBasic PRINT USING function has been added for more versatile use of the ZBasic high-precision math package. This function was designed to be very similar to Basic's syntax while making it even more powerful in that ZBasic allows more than one USING clause per PRINT statement. Its major limitation as compared to Basic's PRINT USING is that it allows only one value per USING and no commas. Also, no floating '$' or '%' symbols or trailing signs are allowed in ZBasic. ZBasic PRINT USING is approximately 50 times faster than a typical Basic PRINT USING statement when using double precision. ZBasic USING does not work with integers. To use USING with integers, you must first convert them to @Math numbers.

ZBasic USING Syntax

```
PRINT USING sstr;sstr(,)(;)(USING sstr;sstr)..etc.etc..
```

More than one USING may appear in the same PRINT statement. Any other normal print function may be used in the same statement. (This is valid with the @MATH package only)

Note: X! is SINGLE PRECISION in BASIC but it is a STRING in ZBasic

Example Basic TO ZBasic Conversions

```
Basic --- 20 PRINT USING" ####.## ####.##";X!;Y!
ZBasic -- 20 PRINT USING" ####.##";X!;" ####.##";Y!

Basic --- 30 PRINT USING" ####.##";X!/100
ZBasic -- 30 @Z!=X!/"100":PRINT USING" ####.##";Z!

Basic --- 30 PRINT USING"YOUR TAX IS ####.##";X!
ZBasic -- 30 PRINT USING"YOUR TAX IS ####.##";X!

Basic --- 40 PRINT USING" ###,###.###,###";X!
ZBasic -- 40 PRINT USING" ######.######";X! <---------- No commas

Basic --- 50 PRINTX,:PRINT USING" ###.##";X!
ZBasic -- 50 PRINTX,USING" ###.##";X!

Basic --- 60 PRINTX,:PRINTUSING" ##.##";X!,:PRINTZ,A$
ZBasic -- 60 PRINTX,USING" ##.##";X!,Z,A$

Basic --- 70 U$="###.##":PRINT USING U$;X!
ZBasic -- 70 U$="###.##":PRINT USING U$;X!

Basic --- 80 U$=" ##.## ##.##":PRINT USING U$;X!;Y!
ZBasic -- 80 U$=" ##.##":PRINT USINGU$;X!;USINGU$;Y!
```

RELOCATING ZBASIC PROGRAMS

ZBasic allows you to relocate a program to any part of memory you choose. Relocation is important since the code created by ZBasic is true machine language. Many times authors want their programs to run on a 16K tape TRS-80, but create the program on a 48K disk system. Using the relocation ability of ZBasic, you may save the program to run in that area of memory.

We'll give you a few examples to show you how simple program relocation is with ZBasic.

Relocating a program will require some thinking on the part of the programmer. For one, you will need to watch the size of both the object code and the variables after compiling. With these figures to work with, you can calculate the position in memory for relocating the object code and the variables.

It is important for you to understand that the object code and the variables occupy different areas in memory.

For those unfamiliar with the term "object code," it is what we call the Basic program after it is compiled. The actual Basic program is called the "source code" since it is the source of the object code.

Variables, as referred to above, are the area in memory where ZBasic stores the values for the variables of the object code. For instance, if the string A$ had the value of "HELLO," these characters would be stored in the location for the variable A$ in the variables area. The same goes for all the other variables.

The more variables your program uses, the more memory the variables area will take up.

After ZBasic compiles a program for you, it gives you some very important information for relocation.

For example:

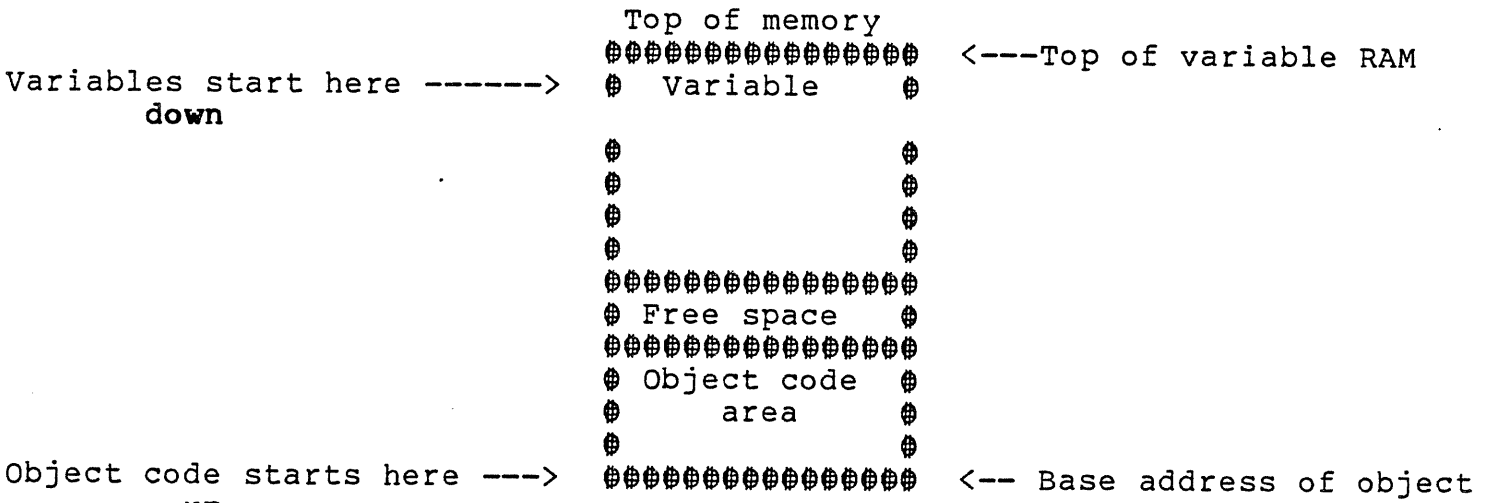
```
Compiled by ZBasic
SIMUTEK INC. TUCSON,AZ
(C) 1982 Andy Gariepy
PASS 1, PASS 2, PASS 3
Compiled size     Basic size     Variable size
Ø65ØØ               Ø4ØØØ          125ØØ
(R)un  (B)asic  (S)ave :
```

Under "Compiled size," ZBasic tells you the actual size of the object code. In the example above it is 65ØØ bytes long. Under "Basic size" it tells you the size of the Basic source code. Under "Variable size" it tells you the amount of memory required to hold the variables (125ØØ in the example above)

In most instances, the "Compiled size" will be larger than the "Basic

size" since ZBasic immediately adds a 1900 byte subroutine package to the object code.

The way the object code and variables are situated in memory is shown in the following memory map.



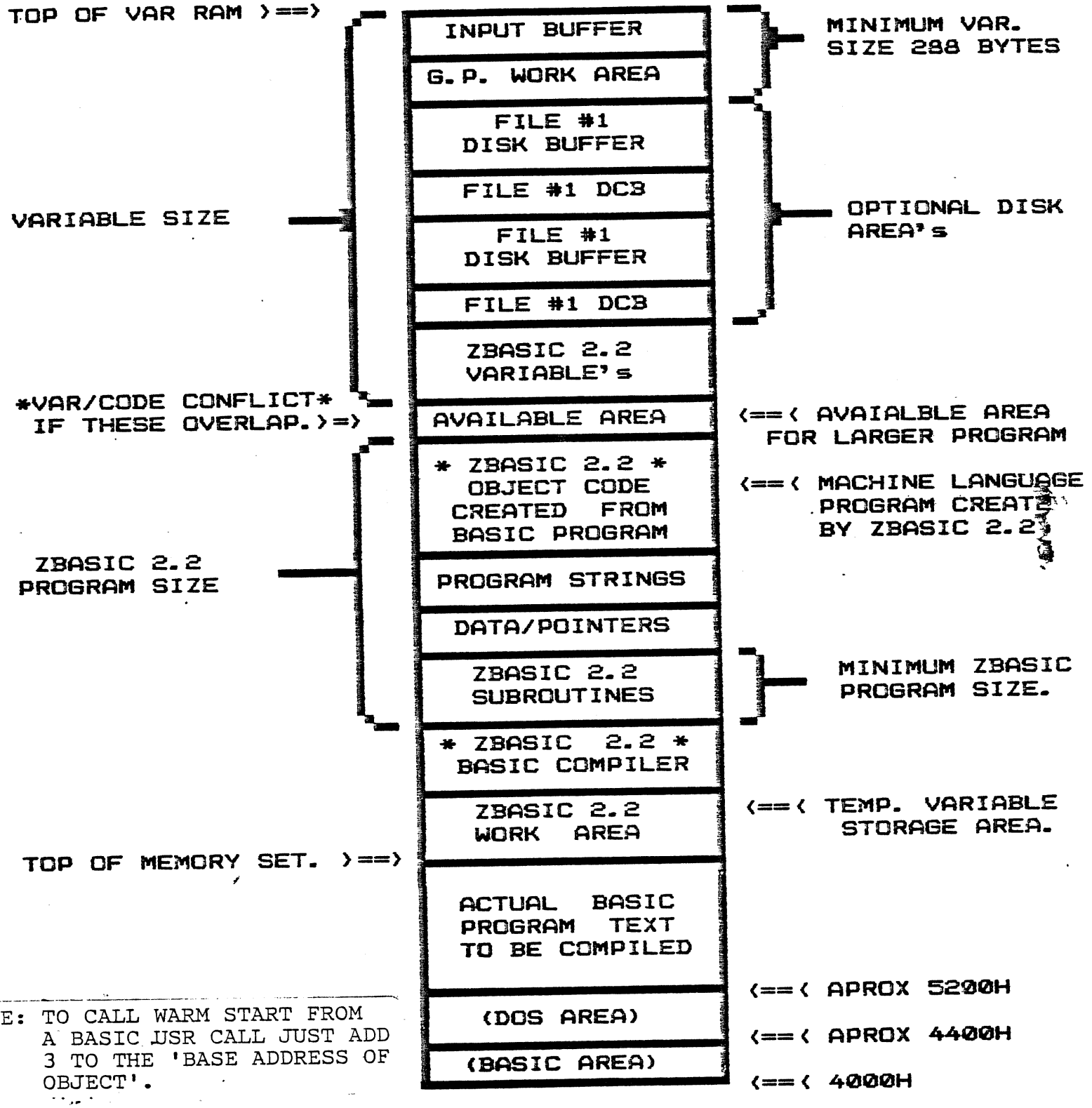
To better explain the meanings above, let us show you the configuration message that ZBasic displays upon entering the program. (48K version)

```
*******************************
*  ZBASIC BASIC COMPILER      *
* Read manual before changing *   E X P L A N A T I O N S
* any of the parameters!!     *          B E L O W
* Copyright 1982 SIMUTEK INC. *
*******************************
Memory size limit 40000 or less        <-- What to set memory size to
Change parameters  (Y/N)?         Y    <---- We typed 'Y'es
Max string array len    00032 <ENTER>  <-- Maximum string array length
Max regular string len 00255 <ENTER>  <-- Maximum regular string length
Top of Variable RAM     65535 <ENTER>  <-- Address variables start at
Base address of object 46038 <ENTER>  <-- Address object code starts at
(W)arm or (C)old start Cold  <ENTER>  <-- See CHAIN
```

The variables start at address 65536 (48K shown above) and go down through memory. The object code above starts at 46038 and goes up through memory. If ever the two should meet, you will encounter an error in ZBasic called "VAR/CODE CONFLICT ERROR." This means that the variables going down through memory and the object code going up through memory have overlapped.

You can see that this would cause some problems in program execution, that is, if the program would run at all!

ZBASIC MEMORY MAP



NOTE: TO CALL WARM START FROM A BASIC USR CALL JUST ADD 3 TO THE 'BASE ADDRESS OF OBJECT'.

Relocation example

Follow the instructions carefully, and all of your relocation attempts will be successful.

First, let's boot the system and load in ZBasic without changing any parameters. Enter BASIC, set memory size, and type in the following program:

```
10 'This is a test of the relocate feature of ZBasic
20 DIM A(200),A$(300),B$(50)
30 PRINT"Hello, we have successfully relocated!"
40 PRINT
50 PRINT
60 INPUT"Hit enter to discontinue";A$
70 END
```

When you are done, save the Basic program above. Call it "TEST."

Now compile it. ZBasic will give you the neccessary information for relocating the program properly. For this example, program sizes were:

```
Compiled by  ZBasic
SIMUTEK INC. TUCSON, AZ
Pass 1, Pass 2, Pass 3
Compiled size     Basic size      Variable size
02001              00184            12562
(R)un    (B)asic   (S)ave
```

Don't save the compiled program this time. Instead, go back to Basic. Since we would like to relocate the program to run in the lower 16K of memory, we first must calculate if it will fit. We add the variable size and the compiled size and find out the total amount of memory needed is 14563. We'll have a little room to spare.

Now we must calculate where to put the object code. Since object code starts at a specified address and goes up through memory we need to find the lowest place in the 16K memory where we can locate it at. This is at memory location 17500. For those interested in the lowest memory location for use with DOS, it is location 21000. Although this address may vary depending on the disk operating system you use, 21000 is generally considered safe.

To calculate where the variables should go, we usually look for the top of memory in the system. The top of memory in a 16K computer is 32767. Therefore, we must determine that the variables and object code do not overlap.

First we add 2001 to the address 17500. This gives us 19501. Then we subtract 12562 from 32767 to get 20205. This confirms that the object code and variables do not overlap. Look at figure below for a better description.

```
ROM/VIDEO    !--OBJECT CODE---!      !-------VARIABLES-----------!
-------------17500--------19501------20205--------------------32767
```

It is imperative that the object code and the variables do not overlap. This will cause a definite system crash.

OK, now we have the figures we need to relocate. Make sure you have saved a copy of the Basic program above. Call it "TEST." The next thing to do is reload ZBasic. If you have a tape system, turn the system off, and reload ZBasic. For disk systems, hit the reset button, and then reload ZBasic.

Answer the configuration questions as stated below.

```
*******************************
*  ZBASIC BASIC COMPILER      *
* Read manual before changing *    A N S W E R S
* any of the parameters!!     *      B E L O W
* Copyright 1982 SIMUTEK INC. *
*******************************
Memory size limit 41ØØØ or less
Change parameters  (Y/N)?          Y
Max string array len   ØØØ32     <ENTER>
Max regular string len ØØ255     <ENTER>
Top of Variable RAM    65535      32767
Base address of object 46Ø38      17500
Maximum input length   ØØ255     <ENTER>
(W)arm or (C)old start Cold      <ENTER>
```

When ZBasic asks to set parameters again, just press "N."

Now we go back to Basic, set memory size, and reload the Basic test program. When you change the configuration parameters, ZBasic will allow you two options after compiling. Either (S)ave the program or return to (B)asic. You cannot run the program when you set the parameters to other than the default settings.

We will now compile the program by pressing the "ZXC" keys. Answer with (S)ave to save your program to tape or disk.

The saved program is now relocated. It will reside in the areas you wanted. You will no longer be able to compile programs after relocating. You must reload ZBasic to continue.

Note: Remember that ZBasic destroys itself after you save a relocated program. You'll have to reload ZBasic if you wish to use it again.

Review of ZBasic Relocation Procedures

1. Write your program in BASIC. DEBUG it under the default modes when possible so that you can use the interactive part of the compiler. This will save a great deal of time.

2. Compile the program and write down the object code size and variable size

3. Calculate where you wish to put the program. Lowest addresses for object: 17500 tape, 21000 disk.

4. Be sure sure variables and object code do not overlap. To do this, add the "Compiled size" to the address you chose for the "Base address of object." Subtract the "Variable size" from the address you chose for "Top of variable RAM." If this number, variables minus the "Top of Variable RAM," is less then the number you obtained for base address PLUS object code size, then the program will relocate improperly.

5. Save the Basic program.

6. Reload ZBasic and set the default parameters. The object address should be set to the address where your program is to start and the variable address should be set to where your program is to end.

7. Go into Basic.

8. Set memory size.

9. Load the Basic program.

10. Compile and save.

CHAINING ZBASIC PROGRAMS

ZBasic has a powerful chaining facility which allows you to go from one program to another and retain variable values!

Chaining is very important because it gives you the power to create programs larger than the available memory in the computer. For instance, if you have an array that takes up 30K of memory, this leaves you with only 8K or so for your main program. Unfortunately, in regular Basic, you would have to save the array (along with any other pertinent information) to disk, load the new program and the array back in from disk, all of which is very time consuming and inefficient.

ZBasic gives you a better way. Chaining leaves variable values intact while allowing different programs to be loaded; variables can thus be shared by each overlay program! This saves a great deal of time in program execution, especially in business programs which typically require more memory and larger data requirements. Since most TRS-80's have only 48K of user RAM, chaining makes this memory seem larger by giving you the ability to overlay programs and still retain the variables; chaining allows you to emulate systems with 128K or more!

Example ZBasic Chaining

We're going to take you through an actual chaining sequence to aquaint you with the ZBasic chaining commands. Be sure to follow each step carefully; missing a step will cause the chaining feature to fail.

The next few pages will give you a step-by-step example of chaining three programs in ZBasic. Remember, even though we are only demonstrating the chaining of 3 programs, many more may be chained.

Go into Disk Basic and type in the following programs and save them to the disk as files "PROG1," "PROG2," "PROG3." Notice that the programs are almost the same so you won't need to type all three completely. Just type in "PROG1," save it, then edit it to be the same as "PROG2," save it, then do the same for "PROG3." Be careful to type the programs in exactly as shown.

This is "PROG1"

```
10 '*********************************************
20 '******    P R O G R A M--O N E      *********
30 '*********************************************
35 DIM A(5),A$(5)
50 IFZ<>999THENCLEAR:Z=999:'IF PROGRAM IS NEW THEN CLEAR VARIABLES
90 CLS:PRINT"This is Program #1"
100 PRINT"The test variables contain this data:"
110 PRINT
120 PRINT"A="A,"B="B,"C="C
130 PRINT"A(1)="A(1),"A$(1)="A$(1)
135 INPUT"Change variables? Y/N";A$:IFA$="Y"ORA$="y"THENGOSUB1000:GOTO90
140 PRINT:INPUT"Which program do you want to run: 2 or 3";PR
150 IFPR<2ORPR>3THEN90
160 IFPR=1THENRUN"PRG2/CMD"
170 RUN"PRG3/CMD"
1000 INPUT"A,B,C";A,B,C
1010 INPUT"A(1)=";A(1)
1020 INPUT"A$(1)=";A$(1)
1030 RETURN
```

This is "PROG2"

```
10 '*********************************************
20 '******    P R O G R A M--T W O      *********
30 '*********************************************
90 CLS:PRINT"This is Program #2"
100 PRINT"The test variables contain this data:"
110 PRINT
120 PRINT"A="A,"B="B,"C="C
130 PRINT"A(1)="A(1),"A$(1)="A$(1)
135 INPUT"Change variables? Y/N";A$:IFA$="Y"ORA$="y"THENGOSUB1000:GOTO90
140 PRINT:INPUT"Which program do you want to run: 1 or 3";PR
150 IFPR<1ORPR>3ORPR=2THEN90
160 IFPR=1THENRUN"PRG1/CMD"
170 RUN"PRG3/CMD"
1000 INPUT"A,B,C";A,B,C
1010 INPUT"A(1)=";A(1)
1020 INPUT"A$(1)=";A$(1)
1030 RETURN
```

This is "PROG3"

```
10 '*******************************************
20 '******    P R O G R A M--THREE     *********
30 '*******************************************
90 CLS:PRINT"This is Program #3"
100 PRINT"The test variables contain this data:"
110 PRINT
120 PRINT"A="A,"B="B,"C="C
130 PRINT"A(1)="A(1),"A$(1)="A$(1)
135 INPUT"Change variables? Y/N";A$:IFA$="Y"ORA$="y"THENGOSUB1000:GOTO90
140 PRINT:INPUT"Which program do you want to run: 1 or 2";PR
150 IFPR<1ORPR>2THEN90
160 IFPR=1THENRUN"PRG1/CMD"
170 RUN"PRG2/CMD"
1000 INPUT"A,B,C";A,B,C
1010 INPUT"A(1)=";A(1)
1020 INPUT"A$(1)=";A$(1)
1030 RETURN
```

Ok, Now that all the programs have been typed in as above, and SAVED, go back to DOS READY, (CMD"S"). Load in ZBasic.

Answer the "Change configuration questions" as follows:

```
*******************************
*  ZBasic 2.2 Basic COMPILER  *
* Read manual before changing *
* any of the parameters!!     *
* Copyright 1982 SIMUTEK INC. *
*******************************
Memory size limit 41000 or less
Change parameters (Y/N) ?
Max string array len      00032
Max regular string len    00255
Top of variable RAM       65535
Base address of object    47040
Maximum input length      00255
(W)arm or (C)old start    Cold
```

ANSWER AS BELOW

```
Y
<ENTER>
<ENTER>
<ENTER>
<ENTER>   (These addresses
<ENTER>    may differ)
W
```

The only question we answer beside "Y" for Change parameters is "W" for WARM START. This means we want the programs to start running without destroying the values in the variables. Cold start zeroes out all the variables when the program starts. Warm start runs the program but uses whatever values were left in the variables. We are chaining, so the whole idea is to retain the values in the variables so that the programs loaded from disk can use them. By answering "W" for Warm start, we make sure the program will start running with the variables intact. Note in line 50 of program one that we use the CLEAR command to clear variables manually. Since the variables need to be zeroed out when the program is first run, we check the variable "Z" to see if it has a value of 999. If it does, we leave the variables alone; otherwise, they are cleared out just as in a Cold start.

Now type in Basic and set memory size. (41ØØØ for 48K, 29ØØØ for 32K) After you are back in Basic, load "PROG1."

When "PROG1" is back in memory, we will compile it by holding in the "ZXC" keys together. "PROG1" is the only program we compile in chaining without holding in the shift key. All programs after the first one require that the shift key be held in when the "ZXC" keys are pressed.

Now SAVE "PROG1" as "PRG1." Answer the "Create it" question with "Y." Answer the "Save again" question with "N."

After you have saved the compiled "PROG1," load "PROG2." Chaining programs to the first is accomplished by compiling with the shift key held in. Normally, when we compile, we press "ZXC" together. This time we will do it a little differently. Hit the shift key and "Z," "X," and "C" keys simultaneously.

The shift key must be held in or the variable pointers from the previous compilation of PROG1 will be lost. The idea is to retain the old variable values and make them common to all. If you accidentally compile without holding in the shift key, the chaining feature will have been lost. You will need to re-compile and re-chain programs to insure success.

After compiling, answer "S" to save the compiled program under the filespec "PRG2." ZBasic will append it with "/CMD" automatically.

Follow exactly the same procedure for "PROG3." Save the compiled program as "PRG3."

Now we are ready to test our chained programs.

Go back to DOS READY (CMD"S")

Now type in "PRG1 <ENTER>."

Program one will ask if you want to change variable values. Try giving them different values. Now transfer control to one of the other programs. Notice the variables still retain the values from the old program? Each program will allow changing the values. And you will see for yourself that the variables retain values between programs!

Keep in mind that chaining may be done from as many programs as you like. Just be careful to keep the size of the programs about the same and make sure variables and program don't overlap in memory.

Note: Remember that ZBasic is destroyed any time you use the relocation feature. To chain relocated programs, you'll have to reload ZBasic for every program you wish to chain.

Additional Chaining information

1. If you make changes in any of the chained programs, they all must be re-compiled and re-chained to assure you of stable variable structure. Failure to re-compile and re-chain all the programs may cause faulty variables and possible system crashes.

2. When using DATA statements in chained programs, be sure to use the RESTORE command when entering programs so ZBasic can keep its pointers correct. Failure to do so will cause faulty DATA reads.

3. Caution should be taken when exiting from one program to another when data files are left OPEN. Running a program does not close all open files as in Basic. In fact, different programs can access open data files opened from another program! This can save significant disk access time but ZBasic will not warn you when files are already open if you attempt to open them again! We suggest users have some error checking in their programs.

4. Use caution when chaining relocated programs. Variable addresses must be the same with all programs. Allowing object code and variable code to overlap will cause a system crash or faulty variable values.

5. If you compile without the shift key, variables are lost from previously compiled programs. It will be neccessary to go back to the beginning and re-compile and re-chain the programs.

6. Do not re-DIM in consecutive programs. ZBasic already knows that the array variables exist from the DIM in the first program. Arrays may be DIMensioned one time only when the chain feature is being used.

7. If errors occur when chaining programs, it is best to fix the error in the program and then re-compile and re-chain all the programs together.

8. When writing programs to be chained, compile and check each program thoroughly for errors before chaining. This will save you much time in program creation.

CONVERTING BASIC TO ZBASIC

Converting Basic programs to run in ZBasic is usually a simple matter. The ZBasic compiler allows trial compilations in seconds to find program problems quickly and efficiently. The main difference between ZBasic and Basic programs is the floating point math package and disk I/O. The way string functions are written is also important.

The major difference between Basic and ZBasic is that ZBasic uses integer as a default math package, and Basic uses single precision as a default math package. Single precision would lead to very slow compiled program execution and a very large compiled program size. With these facts in mind, let's go through a program example showing both the original Basic program and the resulting ZBasic program.

Definitions

Integer: Any number which does not require a decimal point and does not exceed the range of -32768 to 32767.

Floating point: Any number which requires a decimal point or which may exceed the range of -32768 to 32767.

Complex string: A string which contains one of the following functions (MID$,RIGHT$,LEFT$,STRING$,TIME$,+)

Simple string: A string which may be operated on directly by ZBasic, such as A$,"HELLO",INKEY$,CHR$(7) etc.

@Math: This is a name for the floating point package used in ZBasic. It allows addition, subtraction, multiplication, and division with up to 32 digits of accuracy. It uses a BCD internal format for significantly reduced rounding errors.

Largest @Math number (+/-9999999999999999.9999999999999999)
Smallest @Math number (+/- .0000000000000001)
Zero is stored as "0.00" or "-0.00"

Example ZBasic Conversions

The following are hypothetical lines of Basic code and their conversions into ZBasic compatible code.

```
 Basic:  10 INPUT X : DIM A(X) <-- ZBasic cannot DIM by variable
ZBasic:  10 X=100 : DIM A(100) <-- Simply use the largest possible amount
```

Note: All arrays must be dimensioned. See Chaining section for an exception.

```
 Basic:  20 IF LEFT$(A$,1)="X" THEN 100 <--- Complex string illegal
ZBasic:  20 ZZ$=LEFT$(A$,1):IF ZZ$="X" THEN 100 <--- Same as above
```

Note: Complex strings only allowed in assignment statements and they can never be nested.

```
 Basic:  25 X$=MID$(RIGHT$(A$,5),2,2) <----- No nested string functions
ZBasic:  25 ZZ$=RIGHT$(A$,5):X$=MID$(ZZ$,2,2) <--- Split functions apart
```

```
 Basic:  30 X=(N/64)*3 <---- Works when calculated in floating point
ZBasic:  30 X=(N*3)/64 <---- Works in integer AND floating point
```

Note: Check all division for accuracy problems.

```
 Basic:  35 IF 1/2 THEN 100 <---- Always TRUE in Basic (.5)
ZBasic:  35 IF 1/2 THEN 100 <---- Always FALSE in ZBasic (0)
```

```
 Basic:  40 INPUT A$,B$ <--- ZBasic allows only one string per INPUT
ZBasic:  40 INPUT A$ : INPUT B$ <---- Split strings into two INPUTs
```

Note: ZBasic's INPUT reads strings just as Basic's LINEINPUT does

```
 Basic:  75 X$="HELLO":X!=".1"   <---- X$ and X! are identical in ZBasic
ZBasic:  75 X1$="HELLO":X2!=".1" <---- Rename the variables to fix
```

```
 Basic:  70 IF INT(X/2)-X/2 THEN 100 <--- Both expr's are ZBasic integers
ZBasic:  70 IF X FN 2 THEN 100       <---- Identical ZBasic function
```

```
 Basic:  80 X=NOT Z
ZBasic:  80 X=NOT(Z) <-- ZBasic NOT requires parentheses
```

```
 Basic:  90 DATA FRED,HERMAN <------- ZBasic string data requires quotes
ZBasic:  90 DATA "FRED","HERMAN" <--- Add quotes to fix
```

```
 Basic: 100 FOR I=0TO1000:NEXT <------- Basic delay. ZBasic much too fas
ZBasic: 100 DELETE 2000 <-------------- ZBasic 2 second delay


 Basic: 120 X=5:FOR I=1 TO 100 STEP X <----- Step variable not allowed
ZBasic: 120 FORI=1TO100 STEP 5 <------------ Make step a number


 Basic: 140 INPUT"STRING";A$ <------ ZBasic doesn't print '?'
ZBasic: 140 INPUT"STRING ? ";A$


 Basic: 150 PRINT@512,"";:INPUT X <--------- Works in ZBasic
ZBasic: 150 INPUT@512,X <------------------- Better ZBasic solution


 Basic: 160 X=&H1234 <---------------- Basic's hex conversion
ZBasic: 160 X=%1234 <---------------- ZBasic's hex conversion


 Basic: 180 AD=VARPTR(X$) <----------------- Points to variable name
ZBasic: 180 AD=VARPTR(X$) <----------------- Points to variable value


 Basic: 190 X=INSTR(X$,Y$)
ZBasic: 190 X=INSTR(1,X$,Y$) <----- Leading expr required in ZBasic

         Note: INSTR is an expression, not a string function
```

@Math Conversion

```
 Basic: 191 PRINT USING"###.## ###.##";A!,B!  <--- Change USING format to
ZBasic: 191 PRINT USING"###.##";A!,USING"###.##";B$ <-- convert


 Basic: 192 PRINT USING"###,###.###,###";A! <-- Works in Basic
ZBasic: 192 PRINT USING"######.######";A!   <-- No commas allowed


 Basic: 193 X=X+.1 <------- Floating point operations must use @Math
ZBasic: 193 @X!=X!+".1" <-- New ZBasic format
```

Note: All floating point math must be converted to @Math

```
 Basic: 194 X!=ABS(X!) <--- Floating point ABS not supported
ZBasic: 194 @X!=X!+"Ø":X$=RIGHT$(X!,LEN(X$)-1)) <--- Identical in ZBasic


 Basic: 195 X!=INT(X!) <------- Floating point INT not supported
ZBasic: 195 @X!=X!+"Ø":X!=LEFT$(X!,INSTR(1,X!,".")-1) <-- Same in ZBasic
   or
ZBasic: 195 X=VAL(X!)  <----- Works if @Math number is -32768 to 32767


 Basic: 196 X!=(1ØØ+Y!)*4 <---- Change to @Math for ZBasic
ZBasic: 196 @X!="1ØØ"+Y!*"4" <- ZBasic's format


 Basic: 197 X!=1ØØØ+Y!*4 <------- Change to @Math for ZBasic
ZBasic: 197 @X!=Y!*"4"+"1ØØØ" <-- ZBasic's format
```

Note: ZBasic executes @Math in the order it is presented

```
 Basic: 198 IF X!<Y! THEN 1ØØ  <------ In ZBasic, subtract the numbers
ZBasic: 198 @Z!=X!-Y!:IF ASC(Z!)=ASC("-") THEN 1ØØ  <-- and check for a
                                                        negative number
```

Disk I/O Conversions

ZBasic disk I/O may use up to 8 open files at one time. These files may be any combination of random or sequential disk files the must be explicitly numbered 1-8. Random files are limited to a record length of 256 and must use both RSET and LSET functions to take and put strings into a random access fielded variable. Use caution when using a DOS other than TRSDOS to avoid disk I/O related problems. (DOSPLUS, MULTIDOS, NEWDOS PLUS and NEWDOS 80 Ver. 1.0 all appear to work correctly. NEWDOS 80 Ver. 2.0 will not work properly).

```
 Basic: 200 OPEN"R",X,"FILENAME" <----- ZBasic requires a number
ZBasic: 200 OPEN"R",1,"FILENAME" <----- Substitute a number for X


 Basic: 210 OPEN"R",1,"FILENAME" <----- Opens a new file in Basic
ZBasic: 210 OPEN"O",1,"FILENAME" <----- Opens a new file in ZBasic


 Basic: 220 IF EOF(1)=-1 THEN 1000 <------- ZBasic returns the error #
ZBasic: 220 IF EOF(1) THEN 1000 <---------- not just a -1


 Basic: 230 IF EOF(X) THEN 1000 <------- ZBasic requires a number
ZBasic: 230 IF EOF(1) THEN 1000 <------- Change X to 1


 Basic: 240 FIELD #1, X AS A$ <------- ZBasic requires a number
ZBasic: 240 FIELD #1, 8 AS A$ <------- Change X to 8


 Basic: 250 GET X,Y <---------- ZBasic requires a number
ZBasic: 250 GET 1,Y <---------- Change X to 1

        Note: The same applies to the PUT statement


 Basic: 260 CLOSE <--------- ZBasic requires file numbers
ZBasic: 260 CLOSE 1,2,3

        Note: Variables can't be used for file numbers in ZBasic


 Basic: 270 FIELD#1,25 AS A$(X) <-- Arrays not allowed as FIELDed var's
ZBasic: 270 FIELD#1,25 AS A$ <----- Substitute normal variable
```

```
 Basic: 28Ø FIELD#1,2ASA$:X$=A$ <------- ZBasic requires RSET to move
ZBasic: 28Ø FIELD#1,2ASA$:RSETX$=A$ <--- FIELDed var's to normal var's


 Basic: 29Ø X$=MKI$(999) <------- Specify @Math for ZBasic
ZBasic: 29Ø @X$=MKI$(999) <------ Use @ to specify @Math

        Note: Same for MKS$ and MKD$


 Basic: 31Ø X=LOC(1):Y=LOF(1) <---- Use caution. Some DOSes
ZBasic: 32Ø X=LOC(1):Y=LOF(1) <---- handle files differently


 Basic: 34Ø OPEN"R",9,"XXX"
ZBasic: 34Ø OPEN"R",8,"XXX" <----- ZBasic allows only 8 files
```

Additional Random I/O Notes

1. To detect disk errors, use the ZBasic EOF function.

2. The FIELD statement must appear before any fielded variables are used.

3. When using the MKI$,MKS$,MKD$ the result must go directly to the FIELDed variable and must not be moved using normal string functions.

4. The RSET statement must be used to take a FIELDed string from the random buffer and put into a normal string before using the string in any ZBasic string functions.

5. The LSET function must be used when putting strings into the random buffer to avoid overflow into other fields and leaving a Ø terminator in the buffer.

6. The MKI$,MKS$,MKD$,CVS, and CVD functions are all @Math functions and as such must be used in an @Math statement line. (Eg; @X$=MKI$(A+Z) or @X$=CVD(A$).

ZBASIC ERROR STATEMENTS

The next few pages describe all of the ZBasic error messages. This section also contains a useful list of situations that can cause ZBasic programs to crash.

Syntax Error in line #nnnnn

This is a general-purpose error that occurs whenever ZBasic encounters an unknown (misspelled) or out of order statement in the Basic program. The line number is always the line in which the SYNTAX error occured. Another cause may be a control character inserted in the Basic line.

Line # Error in line #nnnnn

This error says there is a GOTO/GOSUB/THEN/ELSE/ON to nnnnn where nnnnn does not exist in the program. Line number Ø is not allowed.

For/Next Error in line #nnnnn

This Error may be caused by several conditions. If nnnnn = 65535, there are more FORs than there are NEXTs in the program. If nnnnn is less than 65535, the problem may be a bad STEP value (STEP must be a number). Other problems include bad loop variable types (array, string, etc.) and extra NEXTs with no matching FORs.

Illegal-Stmt Error in line #nnnnn

This error means ZBasic has encountered a Basic instruction which is not supported or available in that particular version of ZBasic.

Variable Error in line #nnnnn

This error means ZBasic encountered an array variable which has not yet been dimensioned by the program. It can also mean a DIMension statement has an illegal value or the wrong number of parameters in the array parentheses.

Var Overflow Error in line #nnnnn

This error means ZBasic has run out of room to store variable names while compiling. The only way to correct this error is to limit memory to a smaller value when entering Basic or remove extra variable names from the program.

Out of Memory Error in line #nnnnn

This error says that ZBasic does not have enough room to compile the program. The program must be shortened or segmented into more parts to allow compiling.

Non-fatal error codes

Below are two error codes that are mere warnings to the programmer. They point to possible problem spots, but in some cases these codes are actually supposed to appear.

*** VAR/CODE CONFLICT ***

This is a non-fatal Error that warns a programmer that the program has been compiled correctly; but, when run, the variables will over-write the program and will crash. To correct this error, you must shrink the program or use the relocation options to move the variables and code farther apart until this message is not displayed. See the memory map and the relocation section to better understand the effect of this error message.

*** Extra RETURN ***

This is a message you may receive after running a ZBasic created program while in the interactive compile mode. This message is to warn that ZBasic encountered a RETURN which was not matched with a GOSUB. This happens when ZBasic fell through all the program lines and hit the automatic return at the end of a created machine language program. This could cause a crash in the created program unless it is going to be used as a machine language program (USR routine, etc.).

Probable causes of crashes

Below is a list of situations that can cause ZBasic programs to crash. Keep this list handy when debugging your ZBasic programs.

1. 'END' Statement encountered when executing from Basic
2. 'STOP' Statement encountered when executing from DOS
3. 'RETURN' Statement encountered when executing from DOS
4. Machine language routine crashed
5. Infinite loop in program.(Example 10 GOTO 10)
6. High memory driver overlaps ZBasic's RAM area
7. Relocated to area of RAM already being used
8. String variable length set has been exceeded
9. 'POKE' into wrong location
10. 'LPRINT' when the line printer is OFF or BUSY
11. 'FOR/NEXT' loop to 32767 or to 32767-step
12. Caught in 'IF' type loop (100 IF X/2=0 THEN 100)
13. No checking for DOS errors (100 IF EOF(1) THEN ...)
14. Warm start with DATA and no RESTORE statement.

Warm start cautions

A warm start address allows entry into a ZBasic created program without clearing variables and without resetting the DATA pointer to the start of the DATA table created by ZBasic. If DATA/READ statements are used, a RESTORE statement must be used before any data may be read. Variables may contain anything on entry to a warm started program but if they need to be cleared, a CLEAR must appear in the program to clear all the variables to null strings or Ø values. See the chaining section of the manual for further reference.

ZBASIC INDEX

```
100 '==================== ZBASIC 2.2 SORT ====================
150 CLEAR 5000
200 DIMS$(100)'*** STRING ARRAY TO BE SORTED *****
300 N=96'********** NUMBER OF ELEMENTS TO SORT ***
400 FOR I = 1 TO N : FOR J= 1 TO 8
500 S$(I)=S$(I)+CHR$(RND(26)+64)
600 NEXTJ,I
700 PRINT"START SORT: "; : GOSUB63000 : PRINT"END SORT"
800 FORI=1TON : PRINT S$(I), : NEXT
900 STOP : '*** END IF FROM DOS ***
63000 '***** ACTUAL SORT SUBROUTINE ******
63010 '******** SORTS ARRAY S$(X) **********
63020 '**** SORTS N ELLEMENTS IN S$() ****
63030 E=1
63040 E=E+E:IFE<N THEN 63040
63050 E=INT((E-1)/2)
63060 IF E=0 THEN RETURN
63070 IT=N-E
63080 FOR I= 1 TO IT
63090 J=I
63100 L=J+E
63110 IF S$(L)<=S$(J) THEN 63170
63120 XX$=S$(J)
63130 S$(J)=S$(L)
63140 S$(L)=XX$
63150 J=J-E
63160 IF J>0 THEN 63100
63170 NEXTI
63180 GOTO63050
```

```
110 A#="1.00001":B#="1.00002":GOSUB120:STOP
120 '***************** @MATH COMPARE VALUES ***********************
130 @CP#=A#-B#:IF ASC(CP#)=45 THEN              PRINT"A#<=B#"
140 IF ASC(CP#)<>45 THEN IF CP#<>" 0.00"       PRINT"A#> B#"
150 @CP#=B#-A#:IF ASC(CP#)=45 THEN              PRINT"A#>=B#"
160 IF ASC(CP#)<>45 THEN IF CP#<>" 0.00"       PRINT"A#< B#"
170 CP#=RIGHT$(CP#,LEN(CP#)-1)'--- REMOVE SIGN POSITION ---
180 IF CP#="0.00" THEN PRINT "A# =B#" ELSE     PRINT"A#<>B#"
190 RETURN
200 '********************** FUNCTIONS ****************************
210 @L#=L#+"0":L#=LEFT$(L#,INSTR(1,L#,".")-1)'---- L=INT(L) ---
220 @L#=L#+"0":L#=RIGHT$(L#,LEN(L#)-1)'------------ L=ABS(L) ---
230 @L#=L#*L#'---------------------------------- L=L^2    ---
240 @L#=L#*L#*L#'------------------------------- L=L^3    ---
```

NOTE: SEE HOW AT MATH NUMBERS
ARE COMPARED.

Scientific ROM-CALLS

```
10 ' ************************************************************
20  '**          ZBASIC 2.2 SCIENTIFIC ----SUBROUTINES          **
30  '**          USING THE   TRS-80 BASIC ROM ROUTINES.         **
40  '**          WARNING: IF ILLEGAL VALUES ARE SENT TO         **
50  '**          TO THE ROM ROUTINES IT MAY CRASH SO BE         **
60  '**          CAREFULL WHEN USING THESE ROUTINES.            **
70  '**          THESE ROUTINES WILL RETURN THE SINGLE          **
80  '**          PRECISION RESULT.  (ABOUT 6 DIGIT  PREC.)      **
90  '**          TO USE, JUST SET THE VALUE OF THE NUMBER       **
100 '**          TO BE USED IN VARIABLE 'XV!' AND CALL          **
110 '**          THE LINE WITH THE FUNCTION REQUIRED.           **
120 '**          UPON RETURN, THE VALUE OF THE NUMBER           **
130 '**          WILL BE CONTAINED IN @MATH VAR  "XV!"          **
135 '**   NOTE:  PLEASE USE @MATH ONLY. (NO INTEGER'S)          **
140 ' ************************************************************
160 INPUT"NUMBER TO USE ";NM!
170 @NM!=NM!+"0" : NM!=NM!+"!" :' MAKE A SINGLE PREC. NUMBER
180 XV!=NM!:GOSUB360:PRINT"THE SIN(";NM!;")=";XV!
190 XV!=NM!:GOSUB370:PRINT"THE COS(";NM!;")=";XV!
200 XV!=NM!:GOSUB380:PRINT"THE TAN(";NM!;")=";XV!
210 XV!=NM!:GOSUB390:PRINT"THE ATN(";NM!;")=";XV!
220 XV!=NM!:GOSUB400:PRINT"THE SQR(";NM!;")=";XV!
230 XV!=NM!:GOSUB410:PRINT"THE LOG(";NM!;")=";XV!
240 XV!=NM!:GOSUB420:PRINT"THE EXP(";NM!;")=";XV!
250 XV!=NM!:GOSUB430:PRINT"THE ABS(";NM!;")=";XV!
260 XV!=NM!:GOSUB440:PRINT"THE INT(";NM!;")=";XV!
270 GOTO160
271' ************************************************************
272' **** The following MERGE machine language lines conv- ****
273' **** ert the variable XV! to BINARY and back again.    ****
274' **** Be sure to type the code exactly as below in      ****
275' **** your programs when using these scientific func-   ****
276' **** tions in your programs.                           ****
277' ************************************************************
290 '          V  --- THE 'V' POINTS TO THE INPUT VARIABLE.
300 MERGE %21,XV!,%CD,%E6C:POKE%40AF,4:RETURN' - MAKE BINARY -
310 MERGE %21,XV!,%3E,128,%32,%40D8,%01,%0909,%36,%20,%CD,%FC1
320 '          A  --- THE 'A' POINTS TO THE OUTPUT VARIABLE.
330 '
340 @XV!=XV!+"0":RETURN :'--- CLEAR LEADING ZERO'S ---
350 '
360 GOSUB300:MERGE%CD,%1547:GOTO310:'----- SIN -----
370 GOSUB300:MERGE%CD,%1541:GOTO310:'----- COS -----
380 GOSUB300:MERGE%CD,%15A8:GOTO310:'----- TAN -----
390 GOSUB300:MERGE%CD,%15BD:GOTO310:'----- ATN -----
400 GOSUB300:MERGE%CD,%13E7:GOTO310:'----- SQR -----
410 GOSUB300:MERGE%CD,%0809:GOTO310:'----- LOG -----
420 GOSUB300:MERGE%CD,%1439:GOTO310:'----- EXP -----
430 GOSUB300:MERGE%CD,%0977:GOTO310:'----- ABS -----
440 GOSUB300:MERGE%CD,%0B37:GOTO310:'----- INT -----
```

```
10 '================= ZBASIC 2.2 SIN/COS/TAN ==================
20 PRINT"INPUT AN ANGLE FROM 0 TO 360 DEGREE'S"
30 INPUT"ANGLE TO COMPUTE SIN/COS/TAN FROM ";A!
40 SYSTEM BASIC
50 @A!=A!+"0" '--- STANDARD BCD FORMAT ---
60 X!=A! : GOSUB 1000 : PRINT"SIN(";A!;")=";Y!
70 X!=A! : GOSUB 2000 : PRINT"COS(";A!;")=";Y!
80 X!=A! : GOSUB 3000 : PRINT"TAN(";A!;")=";Y!
90 GOTO 10
1000 '================ X!=SIN(X!) in DEGREE's ================
1010 ZZ!=RIGHT$(X!,LEN(X!)-1) : '----- ZZ!=ABS(X!) -----
1020 @Z!=ZZ!/X! : @X!=Z! * X! : '----- SET UP SIGN -----
1030 @CP!="360"-X! : IF ASC(CP!)=45 THEN X!=CP! : GOTO 1030
1040 @CP!="90"-X!  : IF ASC(CP!)=45 THEN @X!="180"-X!
1050 @X!=X!/"57.29578" : '--- CONVERT TO RADIANS -------
1060 ZZ!=RIGHT$(X!,LEN(X!)-1) : '--- X!=ABS(X!) --------
1070 T!=X!
1080 GOSUB 1170 : @ZZ!=T!/"6"              : @Y!=X!-ZZ!
1090 GOSUB 1170 : @ZZ!=T!/"120"            : @Y!=Y!+ZZ!
1100 GOSUB 1170 : @ZZ!=T!/"5040"           : @Y!=Y!-ZZ!
1110 GOSUB 1170 : @ZZ!=T!/"362880"         : @Y!=Y!+ZZ!
1120 '---------- FOR ADDED ACCURACY ADD THE FOLLOWING ---------
1130 'GOSUB 1170 : @ZZ!=T!/"39916800"      : @Y!=Y!-ZZ!
1140 'GOSUB 1170 : @ZZ!=T!/"6227020800"    : @Y!=Y!+ZZ!
1150 'GOSUB 1170 : @ZZ!=T!/"1307700000000" : @Y!=Y!-ZZ!
1160 RETURN
1170 @T!=T!*X!*X! : RETURN : '---- NEXT CUBE SUBROUTINE ----
2000 '================ X!=COS(X!) in DEGREE's ================
2010 @X!=X!+"90":@CP!="360"-X!
2020 IF ASC(CP!)=45 THEN @X!=X!-"360"
2030 GOTO 1000
3000 '================ X!=TAN(X!) in DEGREE's ================
3010 T1!=X! : GOSUB 2000 : T2!=X! : X!=T1! : GOSUB 1000
3020 @Y!=Y!/T2! : RETURN
```

```
100 CLS'----------------- HIGH SPEED GRAPHIC CIRCLE ----------
110 FOR I=0 TO 359
120 X=I : GOSUB 170 : XP=X*32/550+64
130 X=I : GOSUB 180 : YP=X*23/1000+24
140 SET(XP,YP)
150 IF PEEK(14400)=4 THEN STOP
160 NEXT I : GOTO 100
170 X=X+90'----------------- COS(X)*1000 ENTRY POINT ----------
180 IF X>359 THEN X=X-360'-- SIN(X)*1000 ENTRY POINT ----------
190 S=0 : IF X>179 THEN X=X-180 : S=1
200 IF X>89 THEN X=180-X
210 X=174*X/10 : R=X/10 : Z=R*R/200*R
220 X=X-Z/30+Z/100*R/250*R/240 : IF S THEN X=0-X
230 RETURN
```

```
100 CLS'-------------------- ZBASIC 2.2 PONG GAME ---------------
110 PRINT@14,"---- SIMUTEK ZBASIC 2.2 PONG-PING ----";CHR$(15);
120 FOR I=0 TO 127 : SET(I,3) : SET(I,47) : NEXT I
130 FOR I=3 TO 47
140 SET(1,I) : SET(126,I) : SET(0,I) : SET(127,I)
150 NEXTI
160 FOR I=60 TO 68 : FOR J=22 TO 26 : SET(I,J) : NEXT J,I
170 P1=15878 : P2=15929' --- PLAYER 1 & 2 PADDLE POSITIONS ---
180 X=54 : Y=20 : XD=2 : YD=1' BALL POSITION & DIRECTION FLAGS
190 KB=PEEK(14400)' ----- READ FULL KEYBROD ROW FOR ARROWS----
200 IF KB AND 4  THEN STOP' --------- RETURN TO BASIC --------
210 IF KB AND 8  THEN GOSUB 330' (UP ARROW)  PLAYER #1 UP
220 IF KB AND 16 THEN GOSUB 360' (DN ARROW)  PLAYER #1 DOWN
230 IF KB AND 2  THEN GOSUB 420' (CLEAR)     PLAYER #2 DOWN
240 IF KB AND 64 THEN GOSUB 390' (RT ARROW)  PLAYER #2 UP
250 RESET (X,Y)'-------------- TURN OFF OLD BALL --------------
260 X=X+XD : IF POINT (X,Y) THEN X=X-XD : XD=0-XD : GOSUB 450
270 Y=Y+YD : IF POINT (X,Y) THEN Y=Y-YD : YD=0-YD : GOSUB 450
280 SET (X,Y)'----------------- TURN ON NEW BALL --------------
290 IF 1/2=0 THEN FOR I=0TO200 : NEXT I'------ TIME DELAY ----
300 IF X<5 THEN S2=S2+1 : PRINT @56,S2; : GOSUB 480
310 IF X>122 THEN S1=S1+1 : PRINT@0,S1; : GOSUB 480
320 GOTO 190'------------ MAIN GAME LOOP ----------------------
330 '-------------------- PLAYER #1 UP ------------------------
340 IF P1 < 15552 THEN RETURN
350 POKE P1+64,128 : P1=P1-64 : POKE P1,191 : RETURN
360 '-------------------- PLAYER #1 DOWN ----------------------
370 IF P1 > 16192 THEN RETURN
380 POKE P1,128 : P1=P1+64 : POKE P1+64,191 : RETURN
390 '-------------------- PLAYTER #2 UP -----------------------
400 IF P2<15552 THEN RETURN
410 POKE P2+64,128 : P2=P2-64 : POKE P2,191 : RETURN
420 '-------------------- PLAYER #2 DOWN ----------------------
430 IF P2>16192 THEN RETURN
440 POKE P2,128 : P2=P2+64 : POKE P2+64,191 : RETURN
450 '-------------------- BEEP OFF WALL -----------------------
460 IF 1/2=0 THEN EDIT(256,50,20) ' TRUE ONLY IN ZBASIC
470 RETURN
480 '-------------------- SCORE BEEP --------------------------
490 IF 1/2=0 THEN EDIT(256,10,50) ' TRUE ONLY IN ZBASIC
500 RETURN
```

```
100 '---------- MERGING MACHINE LANGUAGE INTO 2.2 ----------
110 CLS:PRINT"PRESS ANY KEY"
120 X$=INKEY$:IF X$=""THEN 120:'  DELAY TILL A KEY IS PRESSED.
130 POKE 15360,191
140 '                             NOW MERGE MACH. LANG WITH BASIC
150 '
160 MERGE 33,%3C00'               LD      HL,3C00H
170 MERGE 17,%3C01'               LD      DE,3C01H
180 MERGE 1,%03FF'                LD      BC,03FFH
190 MERGE 237,176'                LDIR
200 GOSUB 260'                    CALL    LINE200
210 '                             END OF MACH. LANG. PROGRAM
220 '
230 DELETE 2000'                  DELAY 2000 MS (APROX 2 SECONDS)
240 PRINT@0,"DONE."'              FINISHED!
250 STOP
260 '
270 '             SUBROUTINE
280 '
290 POKE 15360,ASC("*")
300 ERL(63,15361,15360)'           LDIR TOP LINE
310 RETURN'                        RET
```

```
100 INPUT"SQUARE ROOT OF ";SQ! : SYSTEM BASIC
200 @SQ#=SQ#*SQ# : '** THE RESULT SHOULD EQUAL THE INPUT! **
300 GOSUB 64000 : PRINT"SQR(";SQ!;")=";Y# : GOTO 100
64000 '*****************************************
64010 '**  ZBASIC 2.2 SQUARE ROOT ROUTINE  **
64020 '**  ACCURATE TO APROX 20 DIGITS!!!  **
64030 '**  ON ENTRY SQ!, ON EXIT Y#        **
64035 '**  X#,W#,Z#,CP#  ARE ALTERED       **
64040 '*****************************************
64050 @X#=SQ#+"0" : IF X#=" 0.00" THEN Y#=X# : RETURN
64060 IF ASC(SQ#)=45 THEN Y#="*ERROR*" : RETURN
64070 @Y#=X#*".5" : Z#="0"
64080 @W#=X#/Y#-Y#*".5"
64090 @CP#=W#-Z# : IF CP#=" 0.00" OR CP#="-0.00" THEN RETURN
64110 @Y#=Y#+W# : Z#=W# : GOTO 64080
```

```
100 '-------  HIGH SPEED SCREEN MOVE ZBASIC 2.2 EXAMPLE -------
110 PRINTCHR$(15)'                                    TURN OFF CURSOR
120 POKE 15360,128 : ERL(1023,15361,15360)'           * GRAPHIC CLS *
130 FOR I=1TO127 : SET (I,1) : SET(I,24) : SET(I,47) : NEXT
140 FOR I=1TO47 : SET(127,I) : SET(1,I)  : SET(64,I) : NEXT
150 PRINT@132,"ZBASIC 2.2";
160 KB=PEEK(14400)'                                   KEYBOARD SCAN
170 IF KB AND 4  THEN STOP'                           <BREAK>
180 IF KB AND 8  THEN GOSUB 240'                      <UP ARROW>
190 IF KB AND 16 THEN GOSUB 290'                      <DOWN ARROW>
200 IF KB AND 32 THEN GOSUB 340'                      <LEFT ARROW>
210 IF KB AND 64 THEN GOSUB 400'                      <RIGHT ARROW>
220 IF KB AND 2  THEN X=FIX(1024,63,15360)'           <CLEAR>
230 GOTO 160
240 '------------- UP ARROW KEY SCROLL UP ----------------------
250 ERL(64,VARPTR(A$),15360)'                         SAVE TOP LINE
260 ERL(960,15360,15424)'                             MOVE UP
270 ERL(64,16320,VARPTR(A$))'                         TOP TO BOTTOM
280 RETURN
290 '--------------- DOWN ARROW KEY SCROLL DOWN ----------------
300 ERL(64,VARPTR(A$),16320)'                         SAVE BOT. LINE
310 ERR(960,16383,16319)'                             MOVE LINES DOWN
320 ERL(64,15360,VARPTR(A$))'                         BOTTOM TO TOP
330 RETURN
340 '--------------- LEFT ARROW KEY SCROLL LEFT ----------------
350 FOR I=15360 TO 16320 STEP 64'                     DO EACH LINE
360 A=PEEK(I)'                                        SAVE LEFT SIDE
370 ERL(63,I,I+1)'                                    MOVE OVER ONE.
380 POKE I+63,A'                                      END TO END.
390 NEXT I:RETURN
400 '-------------- RIGHT ARROW KEY SCROLL RIGHT ---------------
410 FOR I=15360 TO 16320 STEP 64'                     DO EACH LINE
420 A=PEEK(I+63)'                                     GET END CHAR.
430 ERR(63,I+63,I+62)'                                MOVE LINE OVER
440 POKE I,A'                                         NEW END CHAR.
450 NEXT I : RETURN
```

```
100 PRINT "PACKED STRINGS ARE ALLOWED IN ZBASIC ..."
110 PRINT "                                        "
120 PRINT "  TTTTTTT  EEEEEEE  XX    XX  TTTTTTT   "
130 PRINT "     TT    EE       XX    XX     TT     "
140 PRINT "     TT    EEEEEE    XX  XX      TT     "
150 PRINT "     TT    EE       XX    XX     TT     "
160 PRINT "     TT    EEEEEEE  XX    XX     TT     "
170 ' -- THE ONLY TWO CHARACTERS WHICH MAY NOT APPEAR --
180 ' -- IN A QUOATED STRING ARE CHR$(0) AND CHR$(34) --
```