

microsoft basic compiler documentation

The Microsoft BASIC Compiler and associated software are accompanied by the following documents:

1. **BASIC-80 REFERENCE MANUAL**
provides syntax and detailed explanations of all Microsoft BASIC statements and functions.
2. **BASIC COMPILER USER'S MANUAL**
gives the BASIC compiler command format, error messages, and general directions for the use of the compiler.
3. **MICROSOFT UTILITY SOFTWARE MANUAL**
describes the command formats for the MACRO-80 Assembler and LINK-80 Linking Loader, and provides a reference for MACRO-80 pseudo-operations.

BASIC-80 Reference Manual

This manual is a reference for Microsoft's BASIC-80 language, release 5.0 and later.

There are significant differences between the 5.0 release of BASIC-80 and the previous releases (release 4.51 and earlier). If you have programs written under a previous release of BASIC-80, check Appendix A for new features in 5.0 that may affect execution.

Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement.

©Microsoft, 1979

To report software bugs or errors in the documentation, please complete and return the Problem Report at the back of this manual.

8101-510-05

The following changes should be noted in the BASIC-80 Reference Manual and BASIC Compiler User's Manual.

BASIC-80 Reference Manual Version 5.1

2.53 RANDOMIZE

The prompt string has been changed from:

Random Number Seed (0-65529)?
to
Random Number Seed (-32768 to 32767) ?

3.14a INKEY\$

Format: INKEY\$

Action: Returns either a one character string containing a character read from the terminal or a null string if no character is pending at the terminal. No characters will be echoed and all characters are passed through to the program except for Control-C which terminates the program. (In the compiler version Control-C is also passed through to the program.)

Example:

```
1000 'Timed Input Subroutine
1010 RESPONSE$=""
1020 FOR I%=1 TO TIMELIMIT%
1030 A$=INKEY$ : IF LEN(A$)=0 THEN 1060
1040 IF ASC(A$)=13 THEN TIMEOUT%=0 : RETURN
1050 RESPONSE$=RESPONSE$+A$
1060 NEXT I%
1070 TIMEOUT%=1 : RETURN
```

3.41 VAL

The VAL function now strips leading blanks, tabs, and linefeeds from the argument string. For example,

VAL("- -3")

now returns -3 instead of 0.

3.42 VARPTR(#<file number>)

For random files VARPTR returns the address of the FIELD buffer instead of the disk I/O buffer.

L.1 OPERATIONAL DIFFERENCES

The following statements and commands are not implemented and will generate an error message:

```
AUTO CLEAR CLOAD CSAVE CONT DELETE EDIT  
LIST LLIST RENUM SAVE LOAD MERGE NEW  
COMMON
```

L.2 LANGUAGE DIFFERENCES

The COMMON statement will be implemented in a future release of the BASIC compiler; however, its implementation will be different from the BASIC interpreter's version. The COMMON statement will be similar to FORTRAN's COMMON statement.

The USRn functions are significantly different from the interpreter versions. The argument to the USR function is ignored and an integer result is returned in the HL registers. It is recommended that USR functions be replaced by the CALL statement.

The CHAIN and RUN statements have been implemented in their simplest form only; i.e., CHAIN filename\$. For CP/M, the default extension is .COM. BASCOM programs can chain to any COM file; however, the command line information is not automatically passed. Command line information can be passed by POKEing the appropriate information into the command line area.

Currently, the ERR and ERL functions always return integer values; therefore, programs having errors in lines numbered between 32768 and 65535 will return a negative ERL value. This will be changed in a future release.

PRINT <list>, USING "format"; <list>
is not a valid print statement format. The only allowed formats are:

```
[L]PRINT [#<file number> ,] USING <string expr>; <list>
```

BASIC Compiler User's Manual

1.1.1 BASIC Compilation Switches

The /Z switch tells the compiler to use Z80 opcodes whenever possible. The generated code is listed using 8080 opcodes except in those cases where Z80 opcodes have been used.

The /T switch tell the compiler to use BASIC-80 Version 4.51 execution conventions in the following cases:

1. FOR/NEXT loops are always executed at least one time.
2. TAB, SPC, POS, and LPOS preform according to 4.51 conventions.
3. Automatic floating point to integer conversions use truncation instead of rounding except in the case where a floating point number is being converted to an integer in an INPUT statement.

BASIC COMPILER User's Manual

BASIC Compiler Command Format and Switches
Procedures for Using the BASIC Compiler
Sample Compilation
Error Messages

© Microsoft, 1979

8102-510-01

Microsoft
BASIC Compiler User's Manual

CONTENTS

CHAPTER 1 BASIC Compiler Command Scanner

- 1.1 Command Format
- 1.1.1 BASIC Compilation Switches

CHAPTER 2 Using the BASIC Compiler

- 2.1 Procedure
- 2.2 Sample Compilation

CHAPTER 3 Error Messages

- 3.1 BASIC Compiler Error Messages
- 3.2 BASIC Runtime Error Messages

CHAPTER 1

BASIC COMPILER COMMAND SCANNER

1.1 COMMAND FORMAT

To run the BASIC Compiler, type BASCOM followed by a carriage return. (For users with 32K CP/M systems, type BASCOM32 instead of BASCOM. BASCOM32 is a small loader program which loads BASCOM into the user TPA.) BASIC will return the prompt "*", indicating it is ready to accept commands. To tell the BASIC compiler what to compile and with which options, it is necessary to input a "command string," which is read by the compiler's command scanner. The general format of a BASIC compiler command string is:

objprog-dev:filename.ext, list-dev:filename.ext=
source-dev:filename.ext

objprog-dev:

The device on which the object program is to be written.

list-dev:

The device on which the program listing is written.

source-dev:

The device from which the source-program input to BASIC is obtained. If a device name is omitted, it defaults to the currently selected drive.

The available device names with CP/M are:

A:, B:, C:, D: Disk drives
HSR: High speed reader
LST: Line printer
TTY: Teletype or CRT

filename.ext

The filename and filename extension of the object program file, the listing file, and the source file. Filename extensions may be omitted. The default filename extensions with CP/M are:

BAS	BASIC source file
MAC	MACRO-80 source file
REL	Relocatable object file
PRN	Listing file
COM	Absolute file
FOR	FORTRAN-80 source file
COB	COBOL-80 source file

Either the object file or the listing file or both may be omitted. If neither a listing file nor an object file is desired, place only a comma to the left of the equal sign. If the names of the object file and the listing file are omitted, the default is the name of the source file.

Examples:

*=TEST	Compile the program TEST.BAS and place the object in TEST.REL
* ,TTY:=TEST	Compile the program TEST.BAS and list program on the terminal. No object is generated.
*TESTOBJ=TEST.BAS	Compile the program TEST.BAS and put object in TESTOBJ.REL
*TEST,TEST=TEST	Compile TEST.BAS, put object in TEST.REL and listing in TEST.PRN
* ,=TEST.BAS	Compile TEST.BAS but produce no object or listing file. Useful for checking for errors.

1.1.1 BASIC Compilation Switches

A switch on the end of a compiler command string specifies a special parameter to be used during compilation. Switches are always preceded by a slash (/). More than one switch may be used in the same command. The available switches are:

<u>Switch</u>	<u>Action</u>
/E	The /E switch tells the compiler that the program contains the ON ERROR GOTO statement. If a RESUME statement other than RESUME <line number> is used with the ON ERROR GOTO statement, use /X instead (see below). To handle ON ERROR GOTO properly in a compiled environment, BASIC must generate some extra code for the GOSUB and RETURN statements. Therefore, do not use this switch unless your program contains the ON ERROR GOTO statement. The

/E switch also causes line numbers to be included in the binary file, so runtime error messages will include the number of the line in error.

/X The /X switch tells the BASIC compiler that the program contains one or more RESUME, RESUME NEXT, or RESUME 0 statements. The /E switch is assumed when the /X switch is specified. To handle RESUME statements properly in a compiled environment, the compiler must relinquish certain optimizations. Therefore, do not use this switch unless your program contains RESUME statements other than RESUME <line number>. The /X switch also causes line numbers to be included in the binary file, so runtime error messages will include the kumber of the line in error.

/N The /N switch prevents listing of the generated code in symbolic notation. If this switch is not set, the source listing produced by the compiler will contain the object code generated by each statement.

/D The /D switch causes debug/checking code to be generated at runtime. This switch must be set if you want to use TRON/TROFF. The BASIC compiler generates somewhat larger and slower code in order to perform the following checks:

1. Arithmetic overflow. All arithmetic operations, integer and floating point, are checked for overflow and underflow.
2. Array bounds. All array references are checked to see if the subscripts are within the bounds specified in the DIM statement.
3. Line numbers are included in the generated binary so that runtime errors can indicate the statement which contains the error.
4. RETURN is checked for a prior GOSUB.

/Z The /Z switch tells the compiler to use Z80 opcodes whenever possible. The generated code is listed using 8080 opcodes except in those cases where Z80 opcodes have been used.

/S The /S switch forces the compiler to write long quoted strings (i.e., more than 4 characters) to the binary file as they are encountered. This allows large programs with many quoted strings to compile in less memory. However, there are two disadvantages:

1. Memory space is wasted if identical, long quoted strings appear in the program.
2. Code generated while the /S switch is set cannot be placed in ROM.

/4

The /4 switch allows the compiler to use the lexical conventions of the Microsoft 4.51 BASIC interpreter. That is, spaces are insignificant, variables with embedded reserved words are illegal, variable names are restricted to two significant characters, etc. This feature is useful if you wish to compile a source program that was coded without spaces, and contains lines such as

```
FORI=ATOBSTEPC
```

Without the /4 switch, the compiler would assign the variable "ATOBSTEPC" to the variable FORI. With the /4 switch, it would recognize it as a FOR statement. It is recommended that such programs be edited to the 5.0 lexical standards, rather than using the /4 switch. Delimiting reserved words with spaces causes no increase in the generated code and greatly improves readability.

/C

The /C switch tells the compiler to relax line numbering constraints. When /C is specified, line numbers may be in any order, or they may be eliminated entirely. Lines are compiled normally, but of course cannot be targets for GOTOS, GOSUBs, etc. While /C is set, the underline character causes the remainder of the physical line to be ignored, and the next physical line is considered to be a continuation of the current logical line.
NOTE: /C and /4 may not be used together.

Examples:

* ,TTY:=MYPRG/N Compile MYPRG.BAS and list the source program on the terminal but without the generated code. Put the object file in MYPRG.REL.

*=TEST/E Compile TEST.BAS. The source file contains an ON ERROR GOTO statement. Put the object file in TEST.REL.

*=BIGGONE/D Compile BIGGONE.BAS and put the object file in BIGGONE.REL. Check for overflow and out-of-bound array subscripts, and include line numbers in the object file.

CHAPTER 2

USING THE BASIC COMPILER

2.1 PROCEDURE

The following steps give the procedure for creating, compiling, and saving BASIC programs using the BASIC compiler and LINK-80 loader on the CP/M operating system.

1. Create a source file
Create a BASIC source file using the CP/M editor or Microsoft's EDIT-80 Text Editor or Microsoft's BASIC-80 interpreter. Filenames are up to eight characters long, with 3-character extensions. BASIC source filenames should have the extension BAS. (MACRO-80 source filenames should have the extension MAC.)
2. Error check
Before attempting to compile the program and produce object code for the first time, it is advisable to do a simple syntax check. This will help eliminate the necessity of recompiling later due to syntax errors or other easy-to-fix errors. One way to check for errors is to run the program on Microsoft's BASIC-80 interpreter.

Another way to perform the error check is to do a compilation without generating an object or listing file. For example, if your BASIC source file is called MAX1.BAS, type the following:

```
A>BASCOM ,=MAX1/N
```

This command compiles the source file MAX1.BAS without producing an object or listing file. (For users with 32K CP/M systems, type BASCOM32 instead of BASCOM. BASCOM32 is a small loader program which loads BASCOM into the user TPA.)

If necessary, return to the editor (or interpreter) and correct any errors.

3. Compile the source file
To compile the edited source file and produce an object and listing file, type

A>BASCOM MAX1,MAX1=MAX1

The compiler will create a REL (relocatable) file called MAX1.REL and a listing file called MAX1.PRN.

4. Load, Execute and Save the Program
To load the program MAX1.REL into memory and execute it, type

A>L80 MAX1/G

To exit LINK-80 and save a memory image of the object code, type

A>L80 MAX1/E

When LINK-80 exits, three numbers will be printed: the starting address for execution of the program, the end address of the program and the number of 256-byte pages used. For example

[210C 301A 48]

Use the CP/M SAVE command to save a memory image. The number of pages used is the argument for SAVE. For example

A>SAVE 48 MAX1.COM

NOTE

CP/M always saves memory starting at 100H and jumps to 100H to begin execution. Do not use /P or /D to set the origin of the program or data area to 100H, unless program execution will actually begin at 100H.

The CP/M version of LINK-80 is capable of creating COM files by using the /N switch, (See LINK-80 Switches, Utility Software Manual). In our example,

A>L80 MAX1,MAX1/N/E

loads and links MAX1.REL, creates the file MAX1.COM for direct execution, and exits to CP/M.

An object code file has now been saved on the disk under the name specified with the LINK-80 /N switch or the CP/M SAVE command (in this case MAX1). To execute the program simply type the program name

A>MAX1

5. CP/M Command Lines

CP/M command lines and files are supported; i.e., a BASIC, COBOL-80, FORTRAN-80, MACRO-80 or LINK-80 command line may be placed in the same line with the CP/M run command. For example, the command

```
A>BASCOM =TEST
```

causes CP/M to load and run the BASIC compiler, which then compiles the program TEST.BAS and creates the file TEST.REL. This is equivalent to the following series of commands:

```
A>BASCOM  
*=TEST  
A>
```

2.2 SAMPLE COMPIRATION

BASCOM Y5.0 - Copyright 1979 (C) by MICROSOFT - 11776 Bytes Free
0014 0007 00100 ' SAMPLE BASIC COMPIRATION
 ** 0014'L00100:
0014 0007 00200 '
 ** 0014'L00200:
0014 0007 00300 DEFINT I-N,S
 ** 0014'L00300:
0014 0007 00400 DIM S(50)
 ** 0014'L00400:
0014 006D 00500 S(0) = 1 : S(1) = 1
 ** 0014'L00500: LXI H,0001
 ** 0017' SHLD S%
 ** 001A' SHLD S%+0002
001D 006D 00600 FOR I=0 TO 24
 ** 001D'L00600: LXI H,0000
 ** 0020' SHLD I%
 ** 0023' JMP I00000
 ** 0026'I00001:
0026 006F 00700 S(2*(I+1))=S(2*(I+1)-1)+S(2*(I+1)-2)+3
 ** 0026'L00700: LHLD I%
 ** 0029' DAD H
 ** 002A' DAD H
 ** 002B' PUSH H
 ** 002C' LXI D,S%+0002
 ** 002F' DAD D
 ** 0030' MOV E,M
 ** 0031' INX H
 ** 0032' MOV D,M
 ** 0033' XCHG
 ** 0034' SHLD T:01
 ** 0037' POP H
 ** 0038' PUSH H
 ** 0039' LXI D,S%
 ** 003C' DAD D
 ** 003D' MOV E,M
 ** 003E' INX H
 ** 003F' MOV D,M
 ** 0040' LHLD T:01
 ** 0043' DAD D
 ** 0044' INX H
 ** 0045' INX H
 ** 0046' INX H
 ** 0047' SHLD T:02
 ** 004A' POP H
 ** 004B' LXI D,S%+0004
 ** 004E' DAD D
 ** 004F' PUSH H
 ** 0050' LHLD T:02
 ** 0053' XCHG
 ** 0054' POP H
 ** 0055' MOV M,E
 ** 0056' INX H
 ** 0057' MOV M,D

```
0058 006F      00800  NEXT I
    ** 0058'L00800: LHLD   I%
    ** 005B'        INX    H
    ** 005C'        SHLD   I%
    ** 005F'I00000:
    ** 005F'        LHLD   I%
    ** 0062'        LXI    D,FFE7
    ** 0065'        MOV    A,H
    ** 0066'        RAL
    ** 0067'        JC     I00002
    ** 006A'        DAD    D
    ** 006B'        DAD    H
    ** 006C'I00002: JC     I00001
006F 006F      00900  PRINT "ANSWER =";S(50)
    ** 006F'L00900: CALL   $PROA
    ** 0072'        LXI    H,<const>
    ** 0075'        CALL   $PV1D
    ** 0078'        LHLD   S%+0064
    ** 007B'        CALL   $PV2C
007E 006F
    ** 007E'        CALL   $END
```

00000 Fatal Errors
11151 Bytes Free

The address in the left-hand column is the current program address. The address in the next column is the current data address.

Note the examples of common subexpression elimination in lines 500 and 700, and constant folding and peephole optimization in line 700.

CHAPTER 3

ERROR MESSAGES

3.1 BASIC COMPILER ERROR MESSAGES

The following errors may occur while a program is compiling. The BASIC compiler outputs the two-character code for the error, along with an arrow. The arrow indicates where in the line the error occurred. In those cases where the compiler has read ahead before it discovered the error, the arrow points a few characters beyond the error, or at the end of the line.

The error codes are as follows:

FATAL ERRORS

<u>Code</u>	<u>Error</u>
SN	Syntax Error. Caused by one of the following: Illegal argument name Illegal assignment target Illegal constant format Illegal debug request Illegal DEFxxx character specification Illegal expression syntax Illegal function argument list Illegal function name Illegal function formal parameter Illegal separator Illegal format for statement number Illegal subroutine syntax Invalid character Missing AS Missing equal sign Missing GOTO or GOSUB Missing comma Missing INPUT Missing line number Missing left parenthesis Missing minus sign Missing operand in expression Missing right parenthesis

Missing semicolon
Name too long
Expected GOTO or GOSUB
String assignment required
String expression required
String variable required here
Illegal syntax
Variable required here
Wrong number of arguments
Formal parameters must be unique
Single variable only allowed
Missing TO
Illegal FOR loop index variable
Missing THEN
Missing BASE
Illegal subroutine name

OM Out of Memory
Array too big
Data memory overflow
Too many statement numbers
Program memory overflow

SQ Sequence Error
Duplicate statement number
Statement out of sequence

TM Type Mismatch
Data type conflict
Variables must be of same type

TC Too Complex
Expression too complex
Too many arguments in function call
Too many dimensions
Too many variables for LINE INPUT
Too many variables for INPUT

BS Bad Subscript
Illegal dimension value
Wrong number of subscripts

LL Line Too Long

UC Unrecognizable Command
Statement unrecognizable
Command not implemented

OV Math Overflow

/0 Division by Zero

DD Array Already Dimensioned

FN FOR/NEXT Error
FOR loop index variable already in use
FOR without NEXT
NEXT without FOR

FD Function Already Defined

UF Function Not Defined

WE WHILE/WEND Error
WHILE without WEND
WEND without WHILE

/E Missing "/E" Switch

/X Missing "/X" Switch

WARNING ERRORS

ND Array Not Dimensioned

SI Statement Ignored
Statement ignored
Unimplemented command

3.2 BASIC RUNTIME ERROR MESSAGES

The following errors may occur while a compiled program is executing. The error numbers match those issued by the BASIC-80 interpreter. The compiler runtime system prints long error messages followed by an address, unless /D, /E, or /X is specified. In those cases the error message is followed by the number of the line in which the error occurred.

<u>Number</u>	<u>Message</u>
2	Syntax error A line is encountered that contains an incorrect sequence of characters in a DATA statement.
3	RETURN without GOSUB A RETURN statement is encountered for which there is no previous, unmatched GOSUB statement
4	Out of data A READ statement is executed when there are no DATA statements with unread data remaining in the program.
5	Illegal function call A parameter that is out of range is passed to a math or string function. An FC error may also occur as the result of: <ol style="list-style-type: none">1. a negative or unreasonably large subscript2. a negative or zero argument with LOG3. a negative argument to SQR4. a negative mantissa with a non-integer exponent5. a call to a USR function for which the starting address has not yet been given6. an improper argument to ASC, CHR\$, MID\$, LEFT\$, RIGHT\$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING\$, SPACE\$, INSTR, or ON...GOTO7. a string concatenation that is longer than 255 characters
6	Floating overflow or integer overflow The result of a calculation is too large to be represented in BASIC-80's number format. If underflow occurs, the result is zero and execution continues without an error.

- 9 Subscript out of range
An array element is referenced with a subscript that is outside the dimensions of the array.
- 11 Division by zero
A division by zero is encountered in an expression, or the operation of involution results in zero being raised to a negative power. Machine infinity with the sign of the numerator is supplied as the result of the division, or positive machine infinity is supplied as the result of the involution, and execution continues.
- 14 Out of string space
String variables exceed the allocated amount of string space.
- 20 RESUME without error
A RESUME statement is encountered before an error trapping routine is entered.
- 21 Unprintable error
An error message is not available for the error condition which exists. This is usually caused by an ERROR with an undefined error code.
- 50 Field overflow
A FIELD statement is attempting to allocate more bytes than were specified for the record length of a random file.
- 51 Internal error
An internal malfunction has occurred in Disk BASIC-80. Report to Microsoft the conditions under which the message appeared.
- 52 Bad file number
A statement or command references a file with a file number that is not OPEN or is out of the range of file numbers specified at initialization.
- 53 File not found
A LOAD, KILL or OPEN statement references a file that does not exist on the current disk.
- 54 Bad file mode
An attempt is made to use PUT, GET, or LOF with a sequential file, to LOAD a random file or to execute an OPEN with a file mode other than I, O, or R.

- 55 File already open
A sequential output mode OPEN is issued for a file that is already open; or a KILL is given for a file that is open.
- 57 Disk I/O error
An I/O error occurred on a disk I/O operation. It is a fatal error, i.e., the operating system cannot recover from the error.
- 58 File already exists
The filename specified in a NAME statement is identical to a filename already in use on the disk.
- 61 Disk full
All disk storage space is in use.
- 62 Input past end
An INPUT statement is executed after all the data in the file has been INPUT, or for a null (empty) file. To avoid this error, use the EOF function to detect the end of file.
- 63 Bad record number
In a PUT or GET statement, the record number is either greater than the maximum allowed (32767) or equal to zero.
- 64 Bad file name
An illegal form is used for the filename with LOAD, SAVE, KILL, or OPEN (e.g., a filename with too many characters).
- 67 Too many files
An attempt is made to create a new file (using SAVE or OPEN) when all 255 directory entries are full.