COMMODORE 128

BLITZ!™

BASIC COMPILER

Skyles Electric Works

BLITZ! 128

A BASIC Compiler

for

Commodore 128

from

Skyles Electric Works

Instruction Manual

by

Bob Skyles

Table of Contents

(blank)

Congratulations on the purchase of your BLITZ! 128 Compiler. It is an easy to use, versatile BASIC compiler, designed to speed up the operation of all your BASIC programs.

BLITZ! 128 is fully compatible with the Commodore 128 computer, BASIC 7.0. BLITZ! 128 has the following features:

High speed P-Code.

Small P-Code execution and run- time routines (11K Bytes).

Extensions to standard BASIC 7.0 can be used in programs compiled with BLITZ! 128.

Program overlays are possible, and variables can be passed between chained programs.

Compiled programs can be more easily protected.

compiled programs are handled in the same manner as BASIC programs. They may be RENAMED, LOADed, AUTO LOADed, SAVEd and protected as if they were BASIC programs.

High Speed P-Code

BLITZ! 128 translates BASIC program files into a special P-Code. This P-Code is executed much faster than the original program. Moreover, very large BASIC programs become smaller and need less memory and disk capacity, after you BLITZ! 128 them.

Faster program execution is a result of the following improvements over the standard Commodore BASIC interpreter:

Standard BASIC searches through memory for the location of variables and arrays, and for the destination lines referenced by GOTO, GOSUB, IF-THEN, and other statements. BLITZ! 128 stores these locations, so no searches are necessary.

Numerical constants are converted to floating point or integer constants, as required, during compilation, saving run time conversion.

Syntax checking, is done during compilation, not during run time.

Integer expressions are calculated using true integer arithmetic. The standard BASIC interpreter would convert them to floating point, do the arithmetic, then convert them back to integer.

Expressions are evaluated using Reverse Polish Notation (RPN), thus avoiding unnecessary intermediate data storage.

Small P-Code and Run-time Routines

Generally the P-Code produced by BLITZ! 128 will be approximately 65% of the size of the original program. This will vary depending on the type of BASIC commands used, the number of statements per line, and the amount of "remark" statements in the original BASIC program. The BLITZ! 128 compiler appends to the program about 11k bytes of interpretation and run time routines. Programs that are originally 35 Kbytes (140 blocks of disk storage) or larger will usually be smaller after compilation.

EXTENSIONS

BLITZ! 128 allows you to use extensions to standard BASIC in your programs. These commands must be preceded by a double colon (::) so they are not compiled, but passed on to the built-in BASIC 7.0 interpreter during run-time.

Overlay

BLITZ! 128 is the first compiler which can translate chained programs and even pass variables between them. Unlike standard BASIC, it allows the calling program to be longer or shorter than the called program. Also, the run-time routines are only added to the starting program, saving disk space.

DONGLE PROTECTION

BLITZ 128 only runs with a "dongle" plugged into the user port. You may make backups of your BLITZ! 128 program disk. The user port is located at the left rear of the 128.

Restrictions

BLITZ! 128 compiles all commands of BASIC 7.0. Restrictions apply only to the commands TRAP, RESUME, COLLISION, GRAPHIC CLR, and CONT.

TRAP: TRAP may be used alone (TRAP turnoff) or with a line number (without Variable). All active TRAP commands will be turned off by program end commands (END or STOP), or following a program load with LOAD or DLOAD command.

RESUME RESUME is restricted to the line number option only.

COLLISION: With COLLISION only a line number (without Variable) is allowed. A program end command (END or STOP). Following a load of a program with a LOAD or DLOAD command all active COLLISIONS are cancelled.

GRAPHIC CLR This command will clear the 9 Kbyte HI-RES graphic screen except if it refers to GRAPHIC 0 command. If a GRAPHIC command is discovered during the compiling process, the program is automatically relocated to start at hexadecimal address 4000 (see also special instruction REM ** NG).

CONT: The CONT command may not be used with programs that have been compiled with BLITZ! 128. A SYS command is not available to substitute for the CONT command.

STOP key: The STOP key is always disabled when a program is BLITZ! 128ed. The RESET key is available for stopping a program at any time.

Special Instructions

BLITZ! 128 contain 5 special instructions that may be included in the BASIC source code program. Four begin with a REM ** statement. This allows the BASIC program to run without interference of the special instructions.

1. REM ** NG = NO GRAPHIC-ARRAY
   If a GRAPHIC command is discovered during the compiling process BLITZ! 128 reserves 9 Kbytes for the HI-RES screen. However GRAPHIC 5 (80 column text) the 9 Kbyte area is not needed. With the use of this special instruction, the 9 Kbyte area is not reserved. However it is only necessary to use this special command when extremely long programs (greater than 220 blocks) are compiled. The instruction is need only once in the first program when a multiple program compiling option is chosen (sub options 2, 3, and 4).

2. REM ** RI = RESTORE INPUT BUFFER
   BLITZ! 128 has its own input buffer. In BASIC 7.0 the input buffer is located at hexadecimal 0200 to 0240. BLITZ! 128 decides when it is important to switch it with the run-time routine. The input buffer should not under any circumstances be changed. You should expect that EXTENSION or SYS routines might use this space. If you are not familiar with the design of an EXTENSION or SYS command, use this instruction following them. Do not conclude the EXTENSION or SYS commands with the input buffer changed.

Special Instructions

3. REM ** FI = FLOATING PT. to INTEGER Var.
   With this instruction floating point variables are changed to integer variables. Please note that the changed variables are not passed to EXTENSION commands. This instruction is very useful for FOR-NEXT loops, With integer variables a substantial speed up occurs.
   SYNTAX: REM ** fi a,b,c etc.

4. REM ** NE = No EXTENSION Listing
   This instruction cancels the listing of the EXTENSION commands on the first pass of the compiling process.

5. :: = EXTENSION
   When BLITZ! 128 comes across an EXTENSION command it does not recognize (Compiler message ?SYNTAX ERROR) it will skip them if they are proceeded by two colons (::). The entire expression between the "::" and the next ":" is skipped by BLITZ! 128 and passed to the BASIC interpreter during program execution. If your program has intentional delay loops that you do not wish changed use the double colon to cause BLITZ! 128 to leave them uncompiled. Note that if a BASIC command is treated as an extension all related commands must also be treated as extensions. For example; FOR - NEXT or OPEN - PRINT, INPUT, - CLOSE.

Compiling

Compiling programs with Blitz! 128 is a very simple procedure. With your computer turned OFF plug the BLITZ! 128 dongle into the user port located at the left rear corner. Place the BLITZ! 128 program diskette into your 1541 or 1571 disk drive. Turn on your disk drive and your C-128. The BLITZ! 128 program is automatically loaded and run.

The first question presented on the screen is :

e = english

d = deutsch

It is suggested that the "E" key be selected.

After striking "E" the main menu is presented. They are six menu options presented.

1 = SINGLE DRIVE - no COLLISION command and RESUME with line number only

2 = DUAL DRIVE -

3 = 2 SINGLE DRIVES -

4 = SINGLE DRIVE - RESUME and COLLISION with full syntax

5 = DUAL DRIVE -

6 = 2 SINGLE DRIVES -

Menu items 1, 2, and 3 compile programs without COLLISION commands, and RESUME commands with a line number only. If you select menu options 1, 2, or 3 the compiling occurs 15% faster and the P code (compiled program) is 20% shorter than the same program compiled with menu options 4, 5, or6.

Compiling option 1 and 4, SINGLE DRIVE

This is the most common selection. A BASIC program in drive 0 of device 8 is compiled and recorded back onto the diskette with the prefix of "c/" followed by the original program name. An additional cross reference file prefixed by "z/" is also recorded on the diskette.

To proceed with this option type "1" or "4". The screen prompt then asks:

programname:?

Place the compiling data disk with a copy of the source program into your disk drive. Their should be unused space of 2 times the size of the source program available on the diskette. Now type the name of the source program and strike "RETURN". Note the maximum length of the source program name is 14 characters. Now relax and watch BLITZ! 128 compile your program. A discussion of the compiling program is discussed on the following pages under the heading "COMPILING". If you have not reviewed these notes please do it now.

Compiling option 2 and 5, DUAL DRIVE

This selection places the compiled program or programs onto the diskette in drive 1. There are five different selections presented on the sub-menu when this option is selected.

To proceed with this option type "2" or "5" and the following menu screen will appear.

Compiling option 2 and 5, DUAL DRIVE

```
            Compile...

   1. = single program to drive 1
        (with runtime routines)

   2. = all programs on drive 0 to empty
        disk in drive 1 with
        runtime routines

   3. = all programs on drive 0 to empty
        disk in drive 1, only first
        module will contain the
        runtime routines


   4. = all programs on drive 0 to
        emty disk in drive 1 with
        OVERLAY feature

   5. = single program to drive 1
        without runtime routines

   please select!
```

If you select option 1 the screen prompt then asks:

```
   programname:?
```

Place the compiling source data disk with a copy of the source program into disk drive 0. Place the compiling destination data disk into drive 1. Their should be unused space of 2 times the size of the source program available on the destination diskette. Now type the name of the source program and strike "RETURN". Note the maximum length of the source program name is 14 characters. Now relax and watch BLITZ! 128 compile your program. A discussion of the compiling program is presented on the following pages under the heading "DETAILS".

Compiling option 2 and 5, DUAL DRIVE

If you select option 2 the screen prompt then asks:

```
   pleas insert source disk in drive 0

   hit y when ready
?
```

Place the compiling source data disk with a copy of the source program into disk drive 0. the maximum length of the source program names is 14 characters. Place the compiling destination data disk into drive 1. This should be a formatted blank disk.

Now strike the "Y" key and the screen will respond with:

```
   insert empty disk into drive 1

   do you want to format this disk yes/no
```

Now type "n" for no. Now relax and watch BLITZ! 128 compile your program. A discussion of the compiling program is presented on the following pages under the heading "DETAILS". If you have not reviewed these notes please do it now.

If you select option 3 the screen prompts are identical to option 2. Proceed exactly as option 2. Everything will be the same except the P-code program module will only be attach to the first program. This will reduce the total amount disk space. The loading time of the programs will be correspondingly reduced. Note that variables will not be passed between the programs. If you wish to pass variables between related programs please select option 4 (with OVERLAY).

Compiling option 2 and 5, DUAL DRIVE

If you select option 4 the screen prompts are the same as option 2. Note that variables will be passed between the programs. **BLITZ! 128** will make an extra pass to read the variables. Proceed exactly as option 2. Everything will be the same as option 3. The P-code module is attached to the first program only. This will reduce the total amount of disk space. The loading time of the programs will be correspondingly reduced.

If you select option 5 the screen prompt then asks:

```
programname:?
```

Place the compiling source data disk, with a copy of the source program, into disk drive 0. Place the compiling destination data disk into drive 1. Their should be unused space of 2 times the size of the source program available on the destination diskette. Now type the name of the source program and strike "RETURN". Note the maximum length of the source program name is 14 characters.

Option 5 allows corrections and modifications to a program within a set of programs previously compiled with options 3.

Compiling option 3 and 6, 2 DISK DRIVES

This selection places the compiled program in the device number specified by the user. There are five different selections presented on the sub-menu when this option is selected. The sub-menu selections are identical with the previously discussed options available with a dual disk drive.

To proceed with this option type "3" or "6" and the following menu screen will appear.

```
device no. source program ?8

device no. object program :9
```

If you type return twice the following screen appears:

```
                Compile...

1. = single program to device 9
     (with runtime routines)

2. = all programs on device 8 to empty
     disk in device 9 with
     runtime routines

3. = all programs on device 8 to empty
     disk in device 9, only first
     module will contain the
     runtime routines


4. = all programs on device 8 to
     emty disk in device 9 with
     OVERLAY feature

5. = single program to device 9
     without runtime routines

please select!
```

Compiling option 3 and 6, 2 DISK DRIVES

If you select option 1 the screen prompt then asks:

```
programname:?
```

Place the compiling source data disk with a copy of the source program into disk device 8. Place the compiling destination data disk into device 9. Their should be unused space of 2 times the size of the source program available on the destination diskette. Now type the name of the source program and strike "RETURN". Note the maximum length of the source program name is 14 characters. Now relax and watch BLITZ! 128 compile your program. A discussion of the compiling program is presented on the following pages under the heading "DETAILS". If you have not reviewed these notes please do it now.

If you select option 2 the screen prompt then asks:

```
pleas insert source disk in device 8

hit y when ready
?
```

Place the compiling source data disk with a copy of the source program into disk device 8. the maximum length of the source program name is 14 characters. Place the compiling destination data disk into device 9. This should be a formatted blank disk.

Compiling option 3 and 6, 2 DISK DRIVES

Now strike the "Y" key and the screen will respond with:

insert empty disk into device 9

do you want to format this disk yes/no

Now type "n" for no. Now relax and watch BLITZ! 128 compile your program.

If you select option 3 the screen prompts are identical to option 2. Proceed exactly as option 2. Everything will be the same except the P-code program module will only be attach to the first program. This will reduce the total amount of disk space. The loading time of the programs will be correspondingly reduced. Note that variables will not be passed between the programs. If you wish to pass variables between related programs please select option 4 (with OVERLAY).

If you select option 4 the screen prompt are the same as option 2. Proceed exactly as option 3. Everything will be the same except, that variables will be passed between the programs. **BLITZ! 128** will make an extra pass to read the variables. The P-code program module will only be attach to the first program. This will reduce the total amount of disk space. The loading time of the programs will be correspondingly reduced.

Compiling option 3 and 6, 2 DISK DRIVES

If you select option 5 the screen prompt then asks:

```
programname:?
```

Place the compiling source data disk with a copy of the source program into disk device 8. Place the compiling destination data disk into device 9. Their should be unused space of 2 times the size of the source program available on the destination diskette. Now type the name of the source program and strike "RETURN". Note the maximum length of the source program name is 14 characters.

Option 5 allows corrections and modifications to a program within a set of programs previously compiled with options 3.

blank

Compilation, First Pass

The program is translated into P-Code. BLITZ! 128 also checks for errors.

SYNTAX and **TYPE MISMATCH** errors. The line numbers of the erroneous lines are displayed as they are found. The compiler will complete compilation, but any lines in error must be corrected since they are not translated.

A BAD SUBSCRIPT ERROR will be reported if you change the number of dimensions in an array (i.e. x$(4,2) and later x$(5)). Since it is legal to clear (CLR) the array and redimension it later in the program, this message should be considered a warning only.

OVERFLOW ERROR is displayed whenever a number greater than 1 e38 is found in the program. In this case, a wrong number would be placed in the memory location for that variable.

EXTENSIONS are not considered errors, but are shown as an '?EXTENSION' message together with the corresponding line number.

In pass 1 the line numbers pointed to by GOSUB, GOTO, and THEN statements are stored. In addition the variable and array names are evaluated and stored. This is in contrast to the standard BASIC 7.0 interpreter that repeatedly evaluates each variable as it appears during the execution of the program.

At the end of pass 1, all multi-dimensional arrays are checked. If any of these are not declared by a DIM statement, the message:

BAD SUBSCRIPT ERROR OF <array name> is printed. A 1-dimensional array with no corresponding DIM statement defaults to 11 elements as they would in standard BASIC 7.0.

Compilation, Second Pass

At this point, BLITZ! 128 replaces all variable and line references with their exact locations in memory. If at this time a previously referenced line is still unknown, the error **UNDEFINED STATEMENT IN <line number>** is displayed.

The compiled program is now built up from the following parts.

Run-time routines.
Data statements as found in the whole program.
P-Code (your actual program).
Table of variables.

BLITZ! 128 Compiling Options

BLITZ! 128 handles 3 different system configurations; a C-128 and a single disk drive, a C-128 and a dual disk drive, and a C-128 and two individual disk drives. For each computer disk configuration there are two main compiling methods. One, without COLLISION commands and RESUME with line number only, compiles faster and has 20% smaller P-code. The second, with COLLISION and RESUME, handles all BASIC 7.0 commands except as noted on page 4 of this manual (Restrictions).

A single drive system has only one compiling sub option. A dual drive system and a 2 drive system have 5 sub options. The sub options are identical for systems that consist of a dual disk drive and 2 disk drives. Drive 0, of a dual drive system, is replaced by the device number of the source drive, in a 2 drive system. Drive 1, of a dual drive system, is replaced by the device number of the destination drive of a 2 drive system.

Compiling Option 1 and 4, Single Drive

Sub Option 1

Compiles one program from the diskette located in your single drive onto the same diskette. It is recommended that the source program be placed on new formatted blank diskette. If not there should be at least twice as many blocks free as the length of the source program. The compiler prefixes the file name with 'c/' to mark the compiled program. Another file, with the prefix 'z/', contains the cross reference to the line numbers in the original program. (see ERROR CORRECTION)

During compilation BLITZ! 128 uses two other files (prefix 'p/' and 'd/') which hold the pure P-Code and DATA. These files will be scratched before the end of pass 2.

Example:	Drive 0	test (source file, in standard BASIC 7.0)
results in:	Drive 0	c/test (compiled program, running version)
		z/test (cross reference)
		p/test (pure P-code, scratched during pass 2)
		d/test (data, scratched during pass 2)

The c/test program should now be run and tested. If problems occur please see the following section of this manual labeled ERROR CORRECTION. Eventually the cross reference (z/test) may be scratched.

Compiling Option 2 and 5, Dual Drive

Since the sub options under compiling options 2 and 5 are identical to the sub options available under compiling options 3 and 6 the details will not be covered separately. All sub options will be discussed under compiling option 3 and 6. We will use device number 8 as the source disk and device number 9 as the destination disk. If you have a dual disk drive, substitute drive 0 for device 8 and drive 1 for device 9.

Compiling Option 3 and 6, 2 Single Drives

In the following detailed description disk drive device number 8 is the source disk drive and disk drive device number 9 is the destination drive. When you run BLITZ! 128 you may specify other device numbers for the source and destination drive.

Sub Option 1

Compiles one program, from the diskette located in device number 8, onto the diskette located in device number 9. It is recommended that the destination diskette in device number 9 be a new formatted blank diskette. If not there should be at least twice as many blocks free as the length of the source program. The compiler prefixes the file name with 'c/' to mark the compiled program. Another file with the prefix 'z/' contains the cross reference to the line numbers in the original program. (see ERROR CORRECTION)

During compilation BLITZ! 128 uses two other files (prefix 'p/' and 'd/') which hold the pure P-code and data. These files will be scratched before the end of pass 2.

Compiling Option 3 and 6, 2 Single Drives

Sub Option 1

Example:

Source in:	Device 8	test (source file, in standard BASIC 7.0)
Results in:	Device 9	c/test (compiled program, running version)
		z/test (cross reference)
		p/test (pure P-code, scratched during pass 2)
		d/test (data, scratched during pass 2)

The c/test program should now be run and tested. If problems occur please see the following section of this manual labeled ERROR CORRECTION. Eventually the cross reference (z/test) may be scratched.

Sub Option 2

BLITZ! 128 compiles all files from the source diskette (device 8) onto the destination diskette (device 9). The run-time routines are added to all programs compiled. No cross reference files are created when using this option. If an error occurs during run-time, the program should be compiled using option 1 to produce the cross reference.

Compiling Option 3 and 6, 2 Single Drives

Sub Option 2

Example

```
Source in:     Device 8   program a
                          program b
                          program c

Results in:    Device 9   program a
                          program b
                          program c
```

Sub Option 3

Compiles all programs from the source diskette (device 8) to the destination diskette (device 9). Run-time routines are added to the first program only. No variable passing between the modules is possible. No cross reference files are created. If an error occurs during run-time, the program should be compiled using option 1 to produce the cross reference.

This option has the following advantages:

* Shorter loading time for chained programs.
* Less disk storage needed.
* Only the first program can be loaded and run. This can be used as a form of program protection.

Compiling Option 3 and 6, 2 Single Drives

Sub Option 4

Compiles all programs, from the source diskette, onto the destination diskette. Full variable passing between chained programs is possible. No cross reference files are created. If an error occurs during run-time, the program should be compiled using option 1 to produce the cross reference.

In this option BLITZ! 128 collects all variables and arrays during an additional pass. The starting program, if shorter than the others, is lengthened to the size of the longest program. The starting program is stored with the table of variables of all programs as well as the run-time routines and the P-Code.

You should notice the following restrictions:

BLITZ! 128 capacity applies to total variables, etc.

Max. Length of BASIC source line:	255	bytes
Number of BASIC source lines: . .	5000	maximum
Number of jumps in source program (includes GOTO, GOSUB, and NEXT)	6000	maximum
Number of variables:	1000	maximum
Number of arrays	255	maximum
Number of REM **FI variables . .	200	maximum

All arrays should be declared (with DIM statements) at least in the program that references them first. If this is not done, BLITZ! 128 will tell you it has found a BAD SUBSCRIPT ERROR.

No reloading of the starting program is possible or all variables will be lost. If it is required to restart the first program during run-time, save that program twice on the source disk with different names and call the second one of these during run-time.

Compiling Option 3 and 6, 2 Single Drives

Sub Option 4

Example:

source disk	start
	program a
	program b
	program c
	start2

In this example, it is necessary to be able to call all programs beginning with 'program' from the program named 'start'. It is also necessary to call the start program from one of the called programs.

destination disk	start
	program a
	program b
	program c
	start2

'start' must be loaded and run first, it can then call program a, program b, or program c. Also, program a, program b, or program c can call start2. Start2 is the original program without the run-time routines, or table of variables added.

Sub Option 5

Identical to sub option 1 except no P-code is attached to the compiled program. This is a very handy option to use in conjunction with option 3.

Foreward

BLITZ! 128 will not make you a better programmer. BASIC was developed to introduce people to computer programming. It was intentionally made as an "interpretive" program with maximum of error massages to assist program correction. Correcting errors in a compiled program is much more difficult than an interpretive program. The error must be located in the source code, the source must be corrected, and then the program must be recompiled.

BLITZ! 128 will not make a better program from a poorly written public domain or commercial program. Additionally, all compiled programs require explicit array dimension statements and a different method of attaching machine language sub routines to the original BASIC program. BLITZ! 128 is not recommended for converting "protected" programs to "unprotected" programs. A well written "protected" program will at best become a faster "protected" program.

BLITZ! 128, during compilation, checks for the following errors:

SYNTAX ERROR
TYPE MISMATCH ERROR
UNDEFINED STATEMENT ERROR

All other errors are logic errors (bugs) which occur during run-time. Since the BLITZ! 128 discards all line numbers, the run-time routines show any error together with the contents of its own program counter. The cross reference file (Z/test) created by compiling with option 1 may be used to find the corresponding line number within the original BASIC program. Please see the following example on the next page.

Foreward

Example:

You have just run the program 'c/test' and the computer has replied with 'ILLEGAL QUANTITY ERROR IN 9912'. You should type:

```
LOAD "z/test",8     <RETURN>
```

After the file loads, type:

```
LIST -9912          <RETURN>
```

It will list to counter number 9912. The last line printed is:

```
9909=520
```

You now know that line 520 in the source program caused the error. Make the appropriate changes and compile the program again. This may seem complicated, but keep in mind that, compilation should be the last step in program development. Since the standard BASIC program and the compiled program produce the same results, you should use the built in BASIC interpreter to write and correct the program.

In the above example you may use the standard OPEN and CMD commands to print out a hard copy of the complete cross reference file.

Capacity

Max. Length of BASIC source line:	255 bytes
Number of BASIC source lines: . .	5000 maximum
Number of jumps in source program (includes GOTO, GOSUB, and NEXT)	6000 maximum
Number of variables:	1000 maximum
Number of arrays	255 maximum
Number of REM **FI variables . .	200 maximum

Stop Key:

BLITZ! 128 disables the stop key. The Reset switch, located next to the power switch, on the right side of the Commodore 128, is available to stop the programing

Continue

The use of CONT after a program break is not allowed.

Integer Arithmetic

BLITZ! 128 uses true integer arithmetic whenever requested. This provides improvements in execution time, but doesn't change the results of these calculations.

Extensions

BLITZ! 128 allows you to use extensions to standard BASIC 7.0. These commands must be preceded by a double colon (::) so they are not compiled, but passed on to the C-128's built-in BASIC 7.0 interpreter, during run-time. If you want to suppress the '?EXTENSION' message while compiling, there is a special remark statement which will do this: REM ** NE .

Functions

Try this example, type:

PRINT PEEK(145) <RET>	This prints the contents of location 145.
PRINT PEEK(145/3*3) <RET>	This prints the contents of location 144!
A=145/3*3 <RET> PRINT PEEK(A) <RET>	This prints the contents of location 145.

This problem is generated by the BASIC 7.0 interpreter using different rounding procedures in assigning variables and during parameter evaluation. These sorts of errors occur in conjunction with the BASIC functions PEEK(X), INT(X), and with statements POKE, WAIT, SYS, and with array index calculations. BLITZ! 128 does all rounding using the same procedure, so in the example above all lines would print the contents of location 145.

Arrays

Arrays with more than one dimension must be defined explicitly (in BASIC they would default to 11 elements in each dimension). Arrays with only one dimension will default to 11 elements if they are not defined explicitly, just as in standard BASIC. Since problems could arise with the use of implicit 'DIM' statements, repeated 'DIM' and 'CLR' statements, it is recommended that all arrays be defined explicitly. BLITZ! 128 builds a special array with the normally illegal name 'z*%'. This array contains a 5 byte header, and a series of 2 byte pointers to each array found in the original BASIC program.

FOR-NEXT

It is possible with programs compiled with BLITZ! 128 to write FOR-NEXT loops using integer variables, to speed up your programs even further. Since this is an illegal programming practice in standard BASIC, such programs would cause a SYNTAX ERROR when testing. To alleviate this problem it is recommended you use the REM ** FI instruction

Example:

source program statement

```
FOR I=1 TO 100:I%=I:A(I%)=0:NEXT
```

insert REM ** FI I
(this will run in BASIC)

Overhead

Programs compiled with BLITZ! 128 use less overhead on variables than the standard BASIC interpreter:

COMMAND	BASIC	BLITZ! 128
GOSUB	5 bytes	3 bytes
COLLISION	5 byte	3 bytes
FOR	18 bytes	16 bytes
FOR with integer	illegal	9 bytes

This enables you to construct programs with more complicated nesting structures. However they may cause an 'OUT OF MEMORY ERROR' when executed with standard BASIC.

blank

Using Machine Language With BLITZ! 128

With BLITZ! 128 compiled programs it is not possible to predict the location of the end of the program. Therefore it is not possible to attach machine language programs to the end of the BASIC program.

With BASIC 7.0 it is best to use the BLOAD or BOOT commands for loading machine language programs.

BLITZ! 128 takes over, after a SYS command, the parameters for the a, x, y, and s registers (if available). The machine language program should restore the BASIC text pointer at locations 61/62 decimal.

Machine language programs may be loaded from programs compiled with BLITZ! 128, but the usual loading procedures must be changed. In Commodore BASIC, to load a machine language program, usually a flag is set in the BASIC program that indicates when the machine language program has been loaded:

```
10 IF F=0 THEN F=1:BLOAD "MLP",B1,P3584
```

This sets the variable to 1 and loads "MLP" only once (i.e. on the first pass through that line). This will not work with a BLITZ! 128-compiled program. Instead a method I have found easier to use is to check the first byte of the machine language program to see if it has been loaded, i.e.:

loading a machine language program that starts at 3584 decimal (bank 1) whose first byte is 4c hexadecimal (76 decimal):

```
10 BANK1:IF PEEK(3584)<>76 THEN BLOAD
"MLP",B1,P3584
```