BASIC 64

complete compiler for the Commodore 64

By T. Helbig

25933

A Data Becker product

Published by:

Abacus Software

Third Printing, July 1985
Printed in U.S.A. Translated by Gene Traas, Heidi Sumner
Copyright (C)1984 Data Becker, GmbH
Merowingerstr. 30
4000 Dusseldorf, W.Germany
Copyright (C)1984 Abacus Software, Inc.
P.O. Box 7211
Grand Rapids, MI 49510

ISBN # 0-916439-17-8

TABLE OF CONTENTS

0. INTRODUCTION

Congratulations on your purchase of BASIC-64, our highly versatile compiler for the Commodore 64! BASIC-64 allows you to compile programs in speedcode (pseudo-code), machine language, or a mix of both. You can merge and compile a series of programs using the overlay feature. You can change parameters using the advanced development features. You can compile programs written using BASIC extensions. You can compile programs that work in conjunction with Assembler/Monitor-64 and much more.

BASIC-64 is compatible with the BASIC interpreter, together they form a program development system that lets you write fast, efficient programs in BASIC.

1. GETTING STARTED

SAVE your BASIC program onto a work diskette. Make sure that there is enough extra space on this diskette since larger compiled programs can require up to to 300 blocks disk space. Remove the work diskette from the drive. Now, carefully place the BASIC-64 distribution diskette in the disk drive and type:

```
LOAD "BASIC 64",8 <RETURN>
```

After it has loaded, type:

```
RUN <RETURN>
```

The drive will run for a short time and the MAIN MENU is displayed on the screen which looks like this:

```
          BASIC 64 COMPILER   V1.03
         (C)1984 DATA BECKER, T.HELBIG

      1  = COMPILER/OPTIMIZER I
      2  = COMPILER/OPTIMIZER II
      3  = ADVANCED DEVELOPMENT PACKAGE
      4  = OVERLAY
```

When the disk drive has stopped, remove the distribution diskette, and reinsert the work diskette containing the program to be compiled.

Press 1 or <RETURN> (<RETURN> defaults the compiler to option 1), which selects Compiler/Optimizer I.

Now enter the program name and press <RETURN>.

The compiler translates your program into a speedcode program. The line number of the line being compiled is displayed on the screen. If errors are detected, the compiler displays specific error messages.

Once the compiler is finished, it displays READY. At this point, pressing N tells the compiler that no other programs are to be compiled; pressing any other key restarts the compiler, returning you to the MAIN MENU.

If errors were detected, you should correct the original program, and reRUN the compiler again to recompile. If no errors were detected, BASIC-64 saves the compiled program on the work diskette with a name P-progname. If your program name is "TEST", then the compiled program is named "P-TEST".

To run the compiled program type:

```
LOAD "P-progname",8 <RETURN>
RUN <RETURN>
```

In a compiled program, errors may be detected during runtime. The same error messages as in BASIC are output to the screen. However, the address in memory at which the error occurred is displayed instead of a statement's line number. Using the address listing, you can easily find the error in the original program (see D in Advanced Development Features described later).

Any of the other capabilities of BASIC-64 , covered later in this text, can be used to decrease execution time.

2. THE COMPILER/OPTIMIZERS

The compiler defaults to Compiler/Optimizer I when you press either <RETURN> or 1. You can also run Compiler/Optimizer II (by pressing 2 from the MAIN MENU). These two selections are different only in the manner in which the program is optimized.

Optimizer I is totally compatible with the BASIC interpreter. Calculations are performed as whole number operations, so long as the whole number falls within the integer value range (-32768 to +32767). Otherwise, they are automatically changed to floating point. Since Optimizer I is completely compatible with BASIC 2.0, the optimizing procedures do not affect program behavior, and serve only to increase program speed. Optimizer I uses integer calculations for speed, so it helps if you change all variables to type integer (by adding a '%' to each variable name).

Optimizer II has functions different from Optimizer I and the BASIC interpreter:

> All variables are normally treated as integers (by default), except for string variables, i.e. the compiler places a '%' after each variable.
>
> The division of two integers is performed in whole number operations, unlike the normal floating point division.

Optimizer II ignores decimal places and converts numbers directly into integer data.

Optimizer II is best suited for programs which require mixed variables or which normally do not allow the use of integer variables in BASIC 2.0.

A typical application for Optimizer II:

```
10 A=INT(RND(1)*1000)
```

Variable A is assigned a whole number, even though it is associated with a floating point calculation (RND(1)*1000). Optimizer II will easily compile this program.

Programs to be compiled by Optimizer II should use as few floating point variables as possible. Since Optimizer II doesn't normally handle floating point variables, it is necessary to use the compiler directive REM @ R = so that variables are correctly handled by the compiler. This is discussed later.

Optimizer II has no effect on arrays. Arrays should be treated as whole numbers with a '%' suffix added. (NOTE: this will save a good deal of memory space.)

WARNING!

Use Optimizer II only with programs that you either have written yourself or that you understand completely in terms of operation and logic. This applies to all compiler features that don't rely on the BASIC interpreter (e.g. compiler directives).

3. ADVANCED DEVELOPMENT FEATURES

BASIC-64 offers special options for program development, which can be reached by pressing 3 from the MAIN MENU. All choices appear on the screen in alphabetical order:

A- Gives you the option of producing 6502/6510 machine code, speedcode, or no code whatsoever. If you choose machine language, compile the program using Optimizer II (the compiled program will have the prefix "M-filename").

B- Allows you to input the name of a symbol table to be loaded before compiling. A symbol table retains all variables and memory addresses. This is needed when multiple program require the same variables (e.g. the Overlay Feature).

C- Allows you to save a symbol table. You can list a symbol table to the screen or printer with the program SYMBOL which is found on the distribution diskette. You can also create a symbol table which is compatible with our Assembler/Monitor using SYMBOL.

D- Generates an address list. After compiling a program, an optional address list is written to diskette. It can be loaded by typing:

```
LOAD "Z-filename",8 <RETURN>
```

and listed with:

```
LIST
```

Memory addresses are listed on the left side of the screen and the BASIC line numbers appears on the right (used for finding errors starting a program section with SYS).

E- Lets you change the end-of-memory address (normal end address for the compiler is 65536, for BASIC = 40960).

F- Allows you to raise or lower the starting address of a compiled program. The compiler gives you the option of removing the runtime module (see below) and loading it as a separate program. To start the program, you will have to change the starting address and SYS to the starting address (location 16384 is a good location).

G- Controls the connection of the runtime module to the compiled program. Under some circumstances (for instance, when merging a series of programs with the Overlay Feature) a runtime module separate from the main program(s) may be necessary, which can save disk space and execution time.

H- Gives you the ability to compile programs written using BASIC extensions. Options are SIMON'S BASIC, VICTREE (EXBASIC II), BASIC 4.0 (as found in MASTER-64) and OTHERS. When choosing OTHERS, you will have to input the starting address, and a few other items, check your manual for the extension. NOTE: Toolkit commands are not compilable. SUPERGRAPHICS extensions are not applicable.

I- Lets you input the number of bytes per extended BASIC command (normally one byte, except SIMON'S BASIC).

J - Locates the ELSE command and adjusts the compiler accordingly.

K - Gives you the option of switching runtime error handling off or on (in other words, the compiler will not halt when an error is encountered). Putting a line 0 in your BASIC program means "error system on".

L- Suppresses the Overlay Feature, which can disturb character string constants.

M- Lets you send commands to the disk drive, such as scratching a program that has already been compiled (i.e. the original) to save disk space. After the drive performs its task, disk status is displayed (press <RETURN> to return to the menu).

N- Displays the directory of the disk on the screen.

4. THE OVERLAY FEATURE

The Overlay Feature is used to compile a number of successive programs which can share the same set of variables.

Press 4 (Overlay Feature) from the MAIN MENU and then press 1 (Overlay Pass 1). The compiler will return to the MAIN MENU. Now compile the first program with Optimizer I, the compiler is just setting up a symbol table on these two passes, so it will not behave "normally". When it displays "READY.", press <RETURN>. Press 4 and then 2 (Overlay Pass 2), and do a "recompiling" of the same program. This second time around the program is compiled and you are returned to the MAIN MENU. Repeat this process with all remaining programs (the compiler will set up a separate variable table called "S-OVERLAY").

REMINDER:

- Since these programs are supposedly layered, be sure to include internal LOAD commands with the proper code prefix for the program name (e.g. 999 LOAD "P-NEXTPART",8).

- The first program must be longer than all subsequent program in the overlay group (do NOT alter the start-of-BASIC by POKEing locations 45 and 46).

- Character strings (e.g. A$="DATA") are lost during setup of an overlay.

5. COMPILER DIRECTIVES

It is often necessary to inform the compiler of changes during compilation. To do this, you may use Compiler Directives. These directives are inserted into a program as a extra program line as a REM statement, followed by an @ symbol. Thus a compiler directive always being like this:

```
        REM@ directive
```

TYPES OF DIRECTIVES:

- Arranging variable addresses:

```
REM@ A variable = address
```

The variable is placed at the address given. For example, you can put an integer variable into a sprite control register and the sprite can be quickly and easily controlled. NOTE: Addresses below 768 are not permitted.

- Switching error handling:

```
        REM@ E line number
```

This can be changed manually by pressing K in the submenu. The error system defaults to 'ON' (placing a line 0 in your program also switches it on). Some BASIC extensions offer this, as does BASIC-64.

- Declaring integer variables:

```
REM@ I = variable, variable, ...
```

This is the most frequently used directive.

All named floating point (or other) variables are changed into integers by the compiler for faster execution. Additionally, integers can be used in FOR- NEXT loops (something which the BASIC interpreter normally doesn't allow). This command should only be used with Optimizer II.

- Switching Machine Code Generator:

```
REM@ M
```

From this line onward, the compiler produces machine language. Also, this instruction tells the program to switch to machine language during runtime.

- Switching Speedcode Generator:

```
REM@ P
```

From this line onward, the compiler produces speedcode and makes the proper runtime command. NOTE: @M and @P should be used carefully--you can change between machine code and speedcode only if the individual program sections do not have any GOTOs and GOSUBs to locations outside of that section.

- Switching Optimizers:

```
REM@ 01         (for Optimizer I)
REM@ 02         (for Optimizer II)
```

This directive switches optimizers in mid-program, which affects only variables not previously used before these directives.

- Declaring floating point variables:

```
REM@ R = variable, variable, ...
```

All variables named are converted into floating point variables. NOTE: This step can only be used by Optimizer II.

EXAMPLE:

```
10 FOR I=1 TO 1000
20 A = SQR(I):PRINT A;
30 NEXT
```

4634
2029

You would have to insert the following line to compile this program with Optimizer II:

```
5  REM@ R=A
```

- Freeing cassette buffer memory:

```
REM@ S address
```

The compiler normally places all variables in the cassette buffer for easy access. You can use this range for you own purposes by supplying an address where you feel the buffer should end. For example:

```
REM@ S 1024
```

6. HINTS FOR PROGRAMMING

Here are a few items to help your programming before compiling:

- A GOTO/GOSUB executes slower than a RETURN.

- An IF-THEN executes slower than a FOR-NEXT loop.

- The speed of GOTO/GOSUB depends on the size of the jump.

- Spaces make the program easier to read, but take more time to execute.

- Structured BASIC programs execute more slowly than unstructured ones.

- When the limits of an array are unknown, such as:

```
10 INPUT X: DIM A(X)
```

the compiler displays the field name, a single parenthesis and a question mark; you respond with the maximum value and <RETURN>. This will occur only with unknown arrays.

- Try to dimension arrays in the first line whenever possible.

Programmer's Appendix

Compiler Details

Pass 1: Interprets and optimizes the program, then produces the corresponding code (SPEED-CODE or M/L). The computer displays the current line number and every end-of-line character (colon). Compiler instructions (`REM@`) will display an "R", commands not recognized by Commodore Basic (Basic extensions) are displayed with an "E".

PASS2: Generates the code, the runtime module is merged and the DATA inserted. The screen displays:

DATA - code start: The starting DATA line address in the compiled program.

OBJECT - code start: The starting address of the program.

STRINGS: This is the area for STRING storage. All variables and arrays are stored above the end of STRING range to the end of memory.

EXTENSIONS: If the program uses commands from a BASIC extension the compiler displays the number used.

ERRORS: Lists lines in which errors occurred.

Error Messages:

Normal BASIC errors are identical to interpreted BASIC
RUNTIME: Example is division by zero (A/0)
SYSTEM ERROR: Example: Disk Drive not turned on.

THE ADDRESS LIST:

When RUNTIME errors occur, error messages are displayed. The number presented is a memory location. The line in question can be determined by using the address list generated by OPTION D in the DEVELOPMENT PACKAGE.

To use the address list, note the memory address of the error, LOAD "Z-name",8, and LIST until you reach the proper location. The right side of the list contains the line number which matches the memory address on the left side. This line is normally the incorrect one, but in rare cases, the error may be at the end of the preceding line.

ARRAY Dimensioning:

Arrays must be explicitly dimensioned during compilation. Arrays can exist in memory from $A000-$FFFF. Arrays should be dimensioned in the first program line when possible. Sometimes the limits of an array are not known when compiling, in these cases the compiler displays the array name and a question mark. Enter the maximum size of the array and press return.

Example: 10 INPUT x: DIMA(x)

INTEGER LOOPS:

Integer variables are normally not allowed in BASIC loops (eg. FOR I%=1 TO 10: NEXT I%) , it is possible to do so with the corresponding compiler instruction. Integer loops are not only faster, they use fewer stack locations and can be nested much deeper than in BASIC. There is a limitation in the use of INTEGER loops. No STEP-value is stored for the sake of speed. The increase is always by 1: STEP is not permitted in INTEGER loops. Faster loops can be made to imitate STEP values, examples:

```
10 I%=1
20 REM LOOP INTERIOR
30 I%=I%+2:IF I%<=1000 THEN 20: REM STEP 2
```

```
 5 REM@ I=I
10 FOR I=1 TO 1000
20 REM LOOP INTERIOR
30 I=I+2:NEXT
```

BASIC EXTENSIONS:

BASIC 64 compiles most extensions using a common procedure. No "standard" extension exists for the '64, and commands exist in some extensions that can not be compiled. To compile most BASIC extensions please observe the following rules. A compilable command has the following format:

COMMAND value,value,value

Each value can be a constant, variable or formula of any sort. Here are some commands that comply:

```
PLOT 1,3*SIN(X) TO 4,A*B
GMODE 0,1
```

Many extensions have additional functions, which are used in the following format:

```
VARIABLE = FUNCTION(VALUE,VALUE,...)
```

The parenthesis are left off if no value is needed for the function. The compiler doesn't always understand the data resulting from a function. The best thing to do is pre-assign a variable to a function result, so that the data type can be recorded. Example:

```
X = EVAL(A$)
A$ = INSERT(B$,C$,2)
```

Some DIRECT MODE extension commands are not compilable. Examples: RENUM, TRACE, etc.

Commands that directly change the execution of a program can't be compiled. Examples include (CALL, EXEC, CGOTO,etc.) and program structures (REPEAT, UNTIL, etc.)

Commands that alter memory management can't be compiled. These commands include GLOBAL, LOCAL, etc.

Commands which assign variables values can't be compiled. Examples: INPUTLINE, etc.

Fortunately, most of these limitations are rare ones; useful commands, such as graphic commands, are compilable. The compiler accepts most of the popular BASIC extensions.

SIMON'S BASIC

All of the important commands in Simon's BASIC are acceptable to the compiler. This applies to the graphic commands and multiple functions. The commands listed below do not meet the above requirements and can't be compiled.

-Programmer's Aids (AUTO, RENUMBER, RESET, MERGE)
-INPUT Control (FETCH)
-Number Conversions (%=, $=)
-Design Commands and PRINT AT
-Program Structures (ELSE is acceptable)
-Error Handling (ON ERROR, NO ERROR, OUT)

DESIGN:

There are several ways to replace this command; the simplest method is shown below:

```
10 A=64*13: REM BASE ADDRESS
20 FOR I=A TO A+62 STEP3
30 READ A$; FOR J =0 TO 2
40 W=0: FOR K=1 TO B
50 W=W*2: IF MID$(A$,J*8+K,1() ="B" THEN W=W+1
60 NEXT: POKE I+J,W:NEXT:NEXT
```

Now you can construct SPRITES as if you had the DESIGN command (NOTE: the parenthesis replace the command DATA)

It is possible to use DESIGN as a SPRITE-EDITOR: after designing a sprite, the SPRITE can be read from its memory location, saved on disk, and loaded as needed. The advantage to this method is that a program using SPRITES will not have to be recompiled to change the SPRITE.

PRINT AT:

To replace the `PRINT AT(X,Y);Z` command use:

```
POKE211,X: POKE214,Y:SYS 58732:PRINT Z
```

PROGRAM STRUCTURES:

The programming structures of Simon's BASIC are easily replaced with the usual commands GOTO, GOSUB and IF.

BASIC 4.0 (MASTER-64)

Some BASIC extensions have syntax that can lead to errors after compiling:

```
COPY D0 TO  D1
```

D0 and D1 represent disk drives, the compiler will view these as variables and attempt calculations in the program. A way to rework the syntax so the compiler can use these commands is to put the variables into parenthesis. Here is a BASIC 4.0 example:

```
   10 INPUT "DRIVE 1 OR 0";X
   20 CATALOG D(X)
```

Some of the remaining commands in MASTER-64 are also compilable, as long as variables, formulas are set in parenthesis. Many MASTER commands interfere directly with memory management, and are not compilable (such as those that define screen-zones, data buffers, etc.)

COMPILING BASIC EXTENSIONS:

Remember, your program must not contain any non-compilable commands. BASIC extensions are constantly being updated, so whether commands can be compiled may depend on the version being used. When in doubt, experiment!

To compile the program, remove the BASIC extension package from memory (by removing the cartridge or resetting your '64) after saving your program.

Start the compiler and choose Option 3 from the main menu. Next choose the BASIC extension using option "H".

During PASS 1, when the compiler finds an extension command it will display an "E". The compiler does not recognize these extended commands from the codes stored in the standard BASIC interpreter.

After PASS 2 is completed, the compiler displays the number of BASIC extension commands used.

Before running the compiled program, load the BASIC extension (insert cartridge or load the extensions from diskette). If the program has uncompiled errors, either the extension will display an ERROR message or the program will not run. If you follow the rules above, regarding command compatibility, there should be no problems.

Memory locations 704 (sprite 11) to 767 should NOT be changed with POKE commands, mainly because Simon's BASIC uses these registers for purposes other than sprites.

OPERATION OF COMPILED COMMANDS FROM AN EXTENSION

To execute a compiled extension command, the program gives the command to the extended interpreter, which then executes the command. Extended compiled commands do not execute any faster, only the standard Commodore BASIC commands are speeded up. By using the rules below you can compile extensions and reach higher speeds.

Use as few extended commands as possible.

Use extended commands in program sections where time is not a critical factor.

When employing graphics for calculation displays, use them after and not during the calculations.

Use one complex command instead of several short ones.

When an extended command can be replaced by a standard BASIC command, do so. This is especially valuable for character string functions; complex character string formulas in BASIC work faster than corresponding single extended commands.

Graphic commands should use only whole numbers for coordinates. Calculating coordinates and other graphics should work best with integers variables.

OTHER EXTENSIONS

BASIC extension not listed in option "H", can also be compiled if the meet the criteria listed above. The best way to find out is to try and compile an extended program using each "H" option. You must enter the values for end- of-memory, number of bytes per command and the code for ELSE. These must be determined before running the compiler. If they are not given in the extension documentation use the following procedures.

END-OF-MEMORY: `PRINT PEEK(55) +256 * PEEK(56)`

BYTES PER TOKEN: `TYPE NEW <RETURN>` then enter `10 (EXTENSION COMMAND)<RETURN>`
`PRINT PEEK(2054)` If this value is not 0 then enter 2 in BYTES PER TOKEN

ELSE CODE: Enter NEW <RETURN> then enter `10 ELSE <RETURN>`
`PRINT PEEK(2053)` This value gives the first byte of the command. The second byte is only needed by extensions with two bytes per command
`PRINT PEEK(2054)`

Remember a colon (:) must proceed ELSE when using the `IF...THEN...ELSE` command

ERRORS

The number of an error in a compiled programs is stored in location 700 and can be read by PEEK(700). Below is the list of messages used to check for certain errors (PEEK (700) = 5; Device not present--i.e. Printer not turned on)

1 too many files	16 out of memory
2 file open	17 undef'd statement
3 file not open	18 bad subscript
4 file not found	19 redim'd array
5 device not present	20 division by zero
6 not input file	21 illegal direct
7 not output file	22 type mismatch
8 missing filename	23 string too long
9 illegal device number	24 file data
10 next without for	25 formula too complex
11 syntax	26 can't continue
12 return without gosub	27 undef'd function
13 out of data	28 verify
14 illegal quantity	29 load
15 overflow	

CODE-START: Option "F" allows you to raise the start of the program. If the RUN-TIME module is loaded separately you must start the program with a SYS address command. A good location to set this to is 16384, this frees a graphic screen at 8192-16191 and 16192-16383 may be used for sprites. Programs using the graphic screen are also affected by the interpreter. You must enter POKE44,64:POKE 16384,0:NEW; now you can develop and compile graphic programs.

STOP KEY: POKE 788,PEEK(788)+3 to disable the <STOP> key.

MEMORY LAYOUT

MEMORY MAP for compiled program:

0-1024	system memory
1024-2048	screen memory
2048-code start	Runtime module
code start-start of strings	Program code
start of strings-end of strings	Character strings
end of strings-top of memory	variables and arrays

Compiled program variables:

Integer: 2 bytes - low byte, high byte
Floating-point: 5 bytes - exponent, 4-byte mantissa
String variables: 3 bytes - length, low byte, high byte

Strings: 2 bytes plus 1 byte per character

Variable addresses are found in the symbol table produced in OPTION 3. The locations or length of the strings can't be altered after compiling the program.

Locations 144-828 are used by the operating system.

REM@ S 1024 will free the cassette buffer.

The range from $2C0 to $2FF is reserved for BASIC extensions; usually this memory is free.

To free the memory above $C000, the top of memory must be set at 49152 in OPTION 3, selection "E".

LOAD "NAME",8,128 will LOAD data into the locations from which it was saved. Unlike the LOAD "NAME",8,1 the program will continue onward. This is useful for graphics and assembler routines.

MEMORY SWITCHING:

`POKE 1,52`: allows access to $A000-$FFFF

`POKE 1,51`: character generator in $D000-$DFFF.

`POKE 1,55`: normal

Compiled programs can run without the BASIC ROM's but there are some limitations.

The interrupt must be switched off when changing location 1.
Use:

`POKE 56334,PEEK(56334) AND 254` to switch off the interrupt.

Use:

`POKE 566334,PEEK(56334) OR 1` to turn on the interrupt.

When the interrupt is off input/output commands and floating point operations should not be attempted.

FLOATING-POINT FUNCTIONS:

Complex floating-point functions (SIN, COS, TAN, LOG, etc.) work very slowly. Although these functions are accurate to 9 decimal places after compiling the program will run much faster if simpler equations are used. Below are a few examples:

SIN (X) = X-X*X*X/6 (for: -pi/2 < x < pi/2)
COS (X) = 1-X*X/2
TAN (X) = X+X*X*X/3

A small increase is possible by switching off the video processor with:

```
POKE 53265,PEEK(53265) AND 239: REM SCREEN OFF
POKE 53265,PEEK(53265) OR 16: REM SCREEN ON
```

RS-232 OPERATIONS

To use the RS-232 port on the Commodore 64 you must remember to lower memory to protect the RS-232 buffer area. Use Option 3 item E and lower to 39936. The cassette buffer can not be used for storage when using the RS-232 port with the compiler. You must remember to lower memory to protect the buffer area.

TECHNICAL NOTES

There are actually four different options:

1. Optimizer 1 - P code

2. Optimizer 1 - Machine code

3. Optimizer 2 - P code

4. Optimizer 2 - Machine code

The code output (P code or Machine code) is selected from the Advanced Development menu, item A.

The difference between the two optimizers is primarily that optimizer 1 assumes the ordinary variables (floating point) are to be handled as floating point variables and are allocated five bytes each. Optimizer 2 assumes that these variables should be treated as integer variables and the only way you can have a floating point variable in optimizer 2 is by telling the compiler about it through use of the REM@ statements.

To view it another way, suppose you have a program with both normal floating point variables (such as A) and integer variables (A%). In optimizer 1, A will be assigned as a floating point (five bytes internally) and treated as a floating point variable, and A% will be treated as an integer variable (two byte assignment). Note that this is better than the five bytes allocated in normal basic for a fixed point variable.

In optimizer 2, both will be treated as integer variables. The only way you can have A treated as a floating point is by doing a REM@ R=A.

The compiler starts assigning space to variables in the cassette buffer and then assigns them down from your top of memory address. You can change the start point in cassette buffer memory by a REM@ S command and give the starting address (838-1023). If the address is 1024 or above, the compiler assumes that you do not want any variables in buffer memory and therefore will only assign them down from the top of memory.

Note that some programs and extensions to BASIC use the cassette buffer memory as work space and therefore, with these programs, you will want to make sure that you use a REM@ S=1024, or your variables will be assigned in the work space with disastrous results.

You can see the assignment of variables in memory by using the symbol table output option. You do this by selecting option 3 from the Advanced Development feature. Then select option C. It will ask for a file name where you want the symbol table. This file name will be prefixed with an "S-", and stored on the disk during the compile process. When the compile process is completed, you must reset the 64 and then load the program "SYMBOL" form the BASIC 64 disk. You will be asked for the file name (do not key the S-), just the name you entered under option C). Then select option 2 for a listing to device 4 (the printer). The listing will show string variables and functions first, then arrays, then ordinary variables. They are sorted with the major sort on the second letter (blank in the case of single letter variables) and then the first letter. Each is followed by the decimal address of its assignment. If you want to see how the compiler assigns them, you must remember that addresses are generally assigned in encounter sequence (the sequence in which they are first used in your program). If you hunt for the first variable in the list you will see that it is either in the cassette buffer or starting down from the top of memory you allocated, then the next one down, and so forth.

You will notice that all strings are assigned a 3 byte space allocation. This is sufficient space for a pointer to the beginning of the string in the strings work area (2 bytes) and 1 byte for the length of the string. At the end of the compile, it prints out the amount of space allocated for strings. This is a string storage pool where all string variables are stored. Since all numeric variables are pre-assigned, the FRE() command only shows the amount of space remaining in this string storage pool.

Although this compiler works with almost any type of BASIC program a rule which will result in more free space is that all DIM statements should appear at the beginning of your program. This is because the compiler allocates space as it encounters variables. The first time it sees an array variable, it will allocate (just like BASIC) room for eleven entries (0-10). If later in your program you have a DIM statement for 21 entries (DIM(20)), the compiler will then allocate space for 21 entries of the variable. It will change the address pointer to this new 21 element space, but the 11 entry space previously allocated will be unused and wasted. Therefore all DIM statements should be moved up to the beginning of your program (it is not sufficient to have them as the first executed statements, even though they may be at the back of the program and executed with a GOTO).

The compiler stores the runtime routines starting at address 1025 (just like BASIC). There is no way to relocate this loader, it must always be at 1025.

The stack used for FOR/NEXT and GOSUB/RETURN is quite different from BASIC. First, it is larger, you can do about 70 GOSUBS without RETURNs before the stack will overflow. The system does not use the 6502 stack. Instead, it uses its own stack with the pointer located at address 64 (decimal). If you need to do the equivalent of a "pop" command to clear the stack, you simply poke the value 206 in location 64 [POKE 64,206]. This resets the stack pointer. This is useful in programs where you exit from GOSUB routines by the GOTO command, and this can be issued back at the main menu to clean up the stack.

If the program uses RS-232 operations, you must allocate 512 bytes at the top of memory for RS-232 buffer space. If the normal starting point was 40959 as the top of memory, and you are using RS-232, simply specify the top of memory as 40447 (40959-512) from the Advanced Development menu.

You can use BASIC extensions (such as VIDEO BASIC) with BASIC 64. Simply change any "pop" commands as mentioned above, and compile the program. As the system encounters VIDEO BASIC tokens, it will show you this with an E. It appears to operate all of these well except for the ones which require seeking a line number (such as sprite definitions and tone command setups). You have to change these by pre-compiling the sprites and loading them into memory yourself, since line numbers don't exist after the compilation is finished.

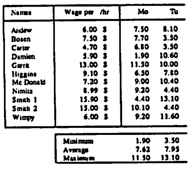
There is a difference in the way the compiler handles a "FOR/NEXT" loop which should not be executed at all. For example, for a factorial routine, you would normally do "FOR I=1 to FT: V=V*I:NEXT". This is fine as long as FT is 1 or greater. If you go in with FT = 0, it results in a FOR I=1 to 0. In standard interpreted BASIC, the expression will be executed once and then at the NEXT command it will finish. In BASIC 64, it does not execute at all.

Note that the top of memory means top of available memory, not top of memory plus 1 as we are used to specifying with BASIC program loading and saving.

A REM statement takes no space at all in the compiled version. Therefore, you can make your programs much clearer by including REMs in the source code. The runtime routines take about 3K of memory, so you can have about about 3K of REMs in your program, and still compile it and it should fit into memory.

SUPER PRODUCTIVITY

POWER PLAN

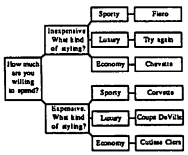


Names	Wage per /hr	Mo	Tu
Andrew	6.00 $	7.50	8.10
Bosen	7.50 $	7.70	3.50
Carter	4.70 $	6.80	3.50
Damien	5.90 $	1.90	10.60
Gerrk	13.00 $	11.50	10.00
Higgins	9.10 $	6.50	7.80
Mc Donald	7.20 $	9.00	10.40
Nimitz	8.99 $	9.20	4.40
Smith 1	15.90 $	4.40	13.10
Smith 2	15.00 $	10.10	4.40
Wimpy	6.00 $	9.20	11.60

Minimum	1.90	3.50
Average	7.62	7.95
Maximum	11.50	13.10

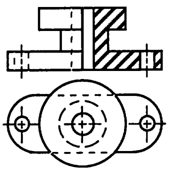
Powerful spreadsheet *plus* builtin graphics - display your important data visually as well as numerically. You'll learn fast with the 90+ HELP screens. Advanced users can use the short-cut commands. For complex spreadsheets, you can use POWER PLAN's impressive features: cell formatting, text formatting, cell protection, windowing, math functions, row and column sort, more. Then quickly display your results in graphics format in a variety of 2D and 3D charts. Includes system diskette and user's handbook. **$49.95**

XPER - expert system


How much are you willing to spend?
Inexpensive What kind of styling?
Expensive. What kind of styling?
Sporty
Fiero
Luxury
Try again
Economy
Chevette
Sporty
Corvette
Luxury
Coupe DeVille
Economy
Cutlass Ciera

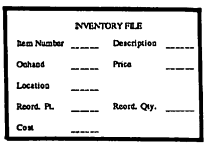
XPER is the first *expert system* - a new breed of intelligent software for the C-64 & C-128. While ordinary data base systems are good at reproducing facts, **XPER** can help you make decisions. Using its simple entry editor, you build the information into a *knowledge base.* XPER's very efficient searching techniques then guide you through even the most complex decision making criteria. Full reporting and data editing. Currently used by doctors, scientists and research professionals. **$59.95**

CADPAK Revised Version



CADPAK is a superb design and drawing tool. You can draw directly on the screen from keyboard or using optional lightpen. POINTs, LINEs, BOXes, CIRCLEs, and ELLIPSEs; fill with solids or patterns; free-hand DRAW; ZOOM-in for intricate design of small section. Mesuring and scaling aids. Exact positioning using our *AccuPoint* cursor positioning. Using the powerful **OBJECT EDITOR** you can define new fonts, furniture, circuitry, etc. Hardcopy to most printers. **$39.95**
McPen lightpen, optional **$49.95**

DATAMAT - data management



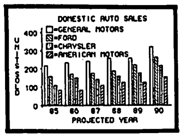
INVENTORY FILE

Item Number	____	Description	_____
Onhand	____	Price	____
Location	____		
Reord. Pt.	____	Reord. Qty.	_____
Cost	_____		

"Best data base manager under $50"
RUN Magazine

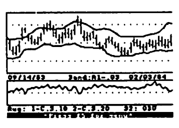
Easy-to-use, yet versatile and powerful features. Clear menus guide you from function to function. Free-form design of data base with up to 50 fields and 2000 records per diskette (space dependent). Simple data base design. Convenient and quick data entry. Full data editing capabilities. Complete reporting: sort on multiple fields and select records for printing in your specific format. **$39.95**

CHARTPAK



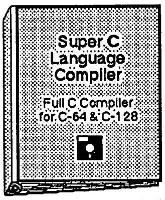
Make professional quality charts from your data in minutes. Quickly enter, edit, save and recall your data. Then interactively build pie, bar, line or scatter graph. You can specify scaling, labeling and positioning and watch **CHARTPAK** instantly draw the chart in any of 8 different formats. Change the format immediately and draw another chart. Incudes statistical routines for average, deviation, least squares and forecasting. Hardcopy to most printers. **$39.95**
CHARTPLOT-64 for 1520 plotter **$39.95**

TAS - technical analysis



Technical analysis charting package to help the serious investor. Enter your data at keyboard or capture it through **DJN/RS** or Warner Services. Track high, low, close, volume, bid and ask. Place up to 300 periods of information for 10 different stocks on each data diskette. Build a variety of charts on the split screen combining information from 7 types of moving averages, 3 types of oscillators, trading bands, least squares, 5 different volume indicators, relative charts, much more. Hardcopy to most printers. **$84.95**

SUPER LANGUAGES

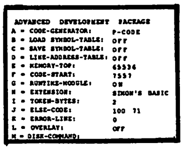


The most advanced C development package available for the C-64 or C-128 with very complete source editor; full K&R compiler (w/o bit fields); linker (binds up to 7 separate modules); and set of disk utilities. Very complete editor handles search/replace, 80 column display with horizontal scrolling and 41K source files. The I/O library supports standard functions like printf and fprintf. Free runtime package included. For C-64/C-128 with 1541/1571 drive. Includes system diskette and user's handbook. **$79.95**

Super Pascal Development Package
Complete Pascal Compiler and Tools for C-64 & C-128

Not just a compiler, but a complete development system. Rivals Turbo Pascal© in both speed and features. Produces fast 6510 machine code. Includes advanced source file editor; full Jensen & Wirth compiler with system programming extensions, new high speed DOS (3 times faster); builtin assembler for specialized requirements. Overlays, 11-digit arithmetic, debugging tools, graphics routines, much more. Free runtime package. Includes system diskette and complete user's handbook. **$59.95**

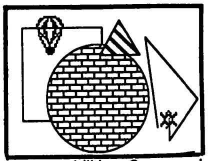
BASIC-64 full compiler



```
ADVANCED  DEVELOPMENT  PACKAGE
A = CODE-GENERATOR:      P-CODE
B = LOAD SYMBOL-TABLE:   OFF
C = SAVE SYMBOL-TABLE:   OFF
D = LINE-ADDRESS-TABLE:  OFF
E = MEMORY-TOP:          65536
F = CODE-START:          7537
G = RUNTIME-MODULE:      ON
H = EXTENSION:           SIMON'S BASIC
I = TOKEN-BYTES:         2
J = ELSE-CODE:           100  71
K = ERROR-LINE:          0
L = OVERLAY:             OFF
M = DISK-COMMAND:
```

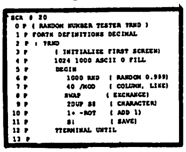
The most advanced **BASIC** compiler available for the C-64. Our bestselling software product. Compiles to superfast 6510 machine code or very compact speed-code. You can even mix the two in one program. Compiles the complete BASIC language. Flexible memory management and overlay options make it perfect for all program development needs. **BASIC 64** increases the speed of your programs from 3 to 20 times. Free runtime package. Includes system diskette and user's handbook. **$39.95**

VIDEO BASIC development



The most advanced graphics development package available for the C-64. Adds dozens of powerful commands to standard BASIC so that you can use the hidden graphics and sound capabilities. Commands for hires, multicolor, sprite and turtle graphics, simple and complex music and sound, hardcopy to most printers, memory management, more. Used by professional programmers for commerical software development. Free runtime package. Includes system diskette and user's handbook. **$39.95**

FORTH Language



```
SCR # 20
 0 P ( RANDOM NUMBER TESTER TRND )
 1 P FORTH DEFINITIONS DECIMAL
 2 P : TRND
 3 P      ( INITIALIZE FIRST SCREEN)
 4 P      1024 1000 ASCII 0 FILL
 5 P      BEGIN
 6 P         1000 RND   ( RANDOM 0.999)
 7 P         40 /MOD    ( COLUMN, LINE)
 8 P        SWAP       ( EXCHANGE)
 9 P         2DUP S@    ( CHARACTER)
10 P         1+ -ROT    ( ADD 1)
11 P         S!         ( SAVE)
12 P      ?TERMINAL UNTIL
13 P
```

Our **FORTH** language is based on the Forth 79 standard, but also includes much of the 83 level to give you 3 times vocabulary of fig-Forth. Includes full-screen editor, complete Forth-style assembler, set of programming tools and numerous sample programs to get you deeply involved in the FORTH language. Our enhanced vocabulary supports both hires and lores graphics and the sound synthesizer. Includes system diskette with sample programs and user's handbook. **$39.95**

Other software also available!
Call now for free catalog and the name of your nearest dealer. Phone: **616/241-5510.**

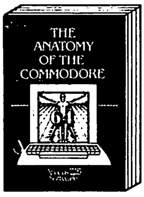
Abacus Software

P.O. Box 7211 Grand Rapids, MI 49510 616/241-5510

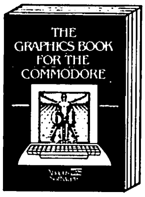
For fast service call **616/241-5510.** For postage and handling, include $4.00 per order. Foreign orders include $8.00 per item. Money orders and checks in U.S. dollars only. Mastercard, Visa and Amex accepted.

Dealer Inquiries Welcome
More than 1200 dealers nationwide

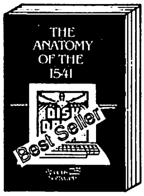
SUPER BOOKS



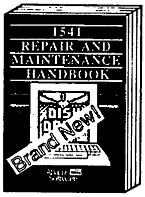
300 pages, $19.95



350 pages, $19.95

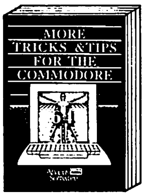


320 pages, $19.95

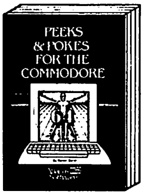


200 pages, $19.95

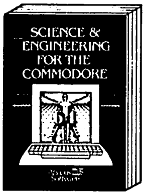
275 pages, $19.95



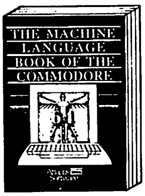
250 pages, $19.95



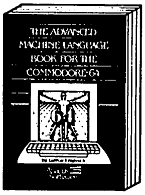
200 pages, $14.95



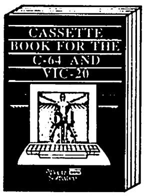
340 pages, $19.95



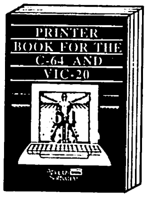
215 pages, $14.95



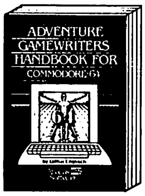
210 pages, $14.95



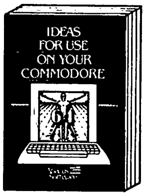
210 pages, $14.95



330 pages, $19.95

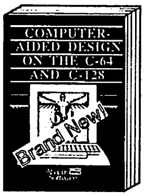


225 pages, $14.95



220 pages, $12.95

250 pages, $19.95



250 pages, $19.95

For fast service call **616/241-5510**. For postage and handling, include $4.00 per order. Foreign orders include $8.00 per item. Money orders and checks in U.S. dollars only. Mastercard, Visa and Amex accepted.

Dealer Inquiries Welcome
More than 1200 dealers nationwide

Abacus Software

P.O. Box 7211 Grand Rapids, MI 49510 616/241-5510

Other software also available!
Call now for free catalog and the name of your nearest dealer. Phone: **616/241-5510.**