Commodore BASIC

Commodore BASIC, also known as **PET BASIC** or **CBM-BASIC**, is the dialect of the BASIC programming language used in Commodore International's 8-bit home computer line, stretching from the PET (1977) to the Commodore 128 (1985).

Commodore BASIC	
Designed by	Microsoft
Developer	Microsoft
First appeared	1977
Stable release	V7.0 / 1985
Preview release	V10.0 / 1991
Implementation language	Assembly (6502)
Platform	PET to the Commodore 128

The core is based on 6502 Microsoft BASIC, and as such it shares many characteristics with other 6502 BASICs of the time, such as Applesoft BASIC. Commodore licensed BASIC from Microsoft in 1977 on a "pay once, no royalties" basis after Jack Tramiel turned down Bill Gates' offer of a $3 per unit fee, stating, "I'm already married," and would pay no more than $25,000 for a perpetual license.[1]

The original PET version was very similar to the original Microsoft implementation with few modifications. BASIC 2.0 on the C64 was also similar, and was also seen on C128s (in C64 mode) and other models. Later PETs featured BASIC 4.0, similar to the original but adding a number of commands for working with floppy disks.

BASIC 3.5 was the first to really deviate, adding a number of commands for graphics and sound support on the C16 and Plus/4. BASIC 7.0 was included with the Commodore 128, and included structured programming commands from the Plus/4's BASIC 3.5, as well as keywords designed specifically to take advantage of the machine's new capabilities. A sprite editor and machine language monitor were added. The last, BASIC 10.0, was part of the unreleased Commodore 65.

History

Commodore took the source code of the flat-fee BASIC and further developed it internally for all their other 8-bit home computers. It was not until the Commodore 128 (with V7.0) that a Microsoft copyright notice was displayed. However, Microsoft had built an easter egg into the version 2 or "upgrade" Commodore Basic that proved its provenance: typing the (obscure) command `WAIT 6502, 1` would result in `Microsoft!` appearing on the screen. (The easter egg was well-obfuscated—the message did not show up in any disassembly of the interpreter.)[2]

The popular Commodore 64 came with BASIC v2.0 in ROM even though the computer was released after the PET/CBM series that had version 4.0 because the 64 was intended as a home computer, while the PET/CBM series were targeted at business and educational use where their built-in programming language was presumed to be more heavily used. This saved manufacturing costs, as the V2 fit into smaller ROMs.

Technical details

Program editing

A convenient feature of Commodore's ROM-resident BASIC interpreter and KERNAL was the full-screen editor.[3][4] Although Commodore keyboards only have two cursor keys which alternated direction when the shift key was held, the screen editor allowed users to enter direct commands or to input and edit program lines from anywhere on the screen. If a line was prefixed with a line number, it was tokenized and stored in program memory. Lines not beginning with a number were executed by pressing the RETURN key whenever the cursor happened to be on the line. This marked a significant upgrade in program entry interfaces compared to other common home computer BASICs at the time, which typically used line editors, invoked by a separate `EDIT` command, or a "copy cursor" that truncated the line at the cursor's position.

It also had the capability of saving named files to any device, including the cassette – a popular storage device in the days of the PET, and one that remained in use throughout the lifespan of the 8-bit Commodores as an inexpensive form of mass storage. Most systems only supported filenames on diskette, which made saving multiple files on other devices more difficult. The user of one of these other systems had to note the recorder's counter display at the location of the file, but this was inaccurate and prone to error. With the PET (and BASIC 2.0), files from cassettes could be requested by name. The device would search for the filename by reading data sequentially, ignoring any non-matching filenames. The file system was also supported by a powerful record structure that could be loaded or saved to files. Commodore cassette data was recorded digitally, rather than less expensive (and less reliable) analog methods used by other manufacturers. Therefore, the specialized Datasette was required rather than a standard tape recorder. Adapters were available that used an analog-to-digital converter to allow use of a standard recorder, but these cost only a little less than the Datasette.

The `LOAD` command may be used with the optional parameter `,1` which will load a program into the memory address contained in the first two bytes of the file (these bytes are discarded and not retained in memory). If the `,1` parameter is not used, the program will load into the start of the BASIC program area, which widely differs between machines. Some Commodore BASIC variants supplied `BLOAD` and `BSAVE` commands that worked like their counterparts in Applesoft BASIC, loading or saving bitmaps from specified memory locations.

The PET does not support relocatable programs and the `LOAD` command will always load at the first two bytes contained in the program file. This created a problem when trying to load BASIC programs saved on other Commodore machines as they would load at a higher address than where the PET's BASIC expected the program to be, there were workarounds to "move" programs to the proper location. If a program was saved on a CBM-II machine, the only way to load it on a PET was by modifying the first two bytes with a disk sector editor as the CBM-II series had their BASIC program area at $0, which would result in a PET attempting to load into the zero page and locking up.

Commodore BASIC keywords could be abbreviated by entering first an unshifted keypress, and then a shifted keypress of the next letter. This set the high bit, causing the interpreter to stop reading and parse the statement according to a lookup table. This meant that the statement up to where the high bit was set was accepted as a substitute for typing the entire command out. However, since all BASIC keywords were stored in memory as single byte tokens, this was a convenience for statement entry rather than an optimization.

In the default uppercase-only character set, shifted characters appear as a graphics symbol; e.g. the command, `GOTO`, could be abbreviated `G{Shift-O}` (which resembled `GΓ` onscreen). Most such commands were two letters long, but in some cases they were longer. In cases like this, there was an ambiguity, so more unshifted letters of the command were needed, such as `GO{Shift-S}` (`GO♥`) being required for `GOSUB`. Some commands had no abbreviated form, either due to brevity or ambiguity with other commands. For example, the command, `INPUT` had no abbreviation because its spelling collided with the separate `INPUT#` keyword, which was located nearer to the beginning

of the keyword lookup table. The heavily used `PRINT` command had a single `?` shortcut, as was common in most Microsoft BASIC dialects. Abbreviating commands with shifted letters is unique to Commodore BASIC.

This tokenizing method had a glitch such that if one included a `REM` (BASIC statement to add a comment to the code) followed by a `{Shift-L}`, when trying to view the program listing, the BASIC interpreter would immediately abort the listing, display a `?SYNTAX ERROR` and return to the `READY.` prompt. This glitch was used to some effect by programmers who wanted to try and protect their work, although it was fairly easy to circumvent.

By abbreviating keywords, it was possible to fit more code on a single program line (which could take up two screen lines on 40-column displays - i.e., C64 or PET, or four lines on the VIC-20's 22-column display). This allowed for a slight saving on the overhead to store otherwise necessary extra program lines, but nothing more. All BASIC commands were tokenized and took up 1 byte (or two, in the case of several commands of BASIC 7 or BASIC 10) in memory no matter which way they were entered. Such long lines were a nuisance to edit. The `LIST` command displayed the entire command keyword - extending the program line beyond the 2 or 4 screen lines which could be entered into program memory.

Performance

Like the original Microsoft BASIC interpreter, Commodore BASIC is slower than native machine code. Test results have shown that copying 16 kilobytes from ROM to RAM takes less than a second in machine code, compared to over a minute in BASIC. To execute faster than the interpreter, programmers started using various techniques to speed up execution. One was to store often-used floating point values in variables rather than using literal values, as interpreting a variable name was faster than interpreting a literal number. Since floating point is default type for all commands, it's faster to use floating point numbers as arguments, rather than integers. When speed was important, some programmers converted sections of BASIC programs to 6502 or 6510 assembly language that was loaded separately from a file or POKEd into memory from DATA statements at the end of the BASIC program, and executed from BASIC using the `SYS` command, either from direct mode or from the program itself. When the execution speed of machine language was too great, such as for a game or when waiting for user input, programmers could poll by reading selected memory locations (such as `$C6`[5] for the 64, or `$D0`[6] for the 128, denoting size of the keyboard queue) to delay or halt execution.

A unique feature of Commodore BASIC is the use of control codes to perform tasks such as clearing the screen or positioning the cursor within a program; these can be invoked either by issuing a `PRINT CHR$(X)` command where X corresponds to the control code to be issued (for example, `PRINT CHR$(147)` is the control code to clear the screen) or by pressing the key in question between quote marks, thus pressing ⇧ Shift + CLR HOME following a quote mark will cause BASIC to display the visual representation of the control code (in this case, a reversed heart) which is then acted upon at program execution (directly printing out the control codes uses less memory and executes faster than invoking a `CHR$` function). This is in comparison to other implementations of BASIC which typically have dedicated commands to clear the screen or move the cursor.

BASIC 3.5 and up have proper commands for clearing the screen and moving the cursor.

Program lines in Commodore BASIC do not require spaces anywhere (but the `LIST` command will always display one between the line number and the statement), e.g., `100 IFA=5THENPRINT"YES":GOTO160`, and it was common to write programs with no spacing. This feature was added to conserve memory since the tokenizer never removes any space inserted

between keywords: the presence of spaces results in extra `0x20` bytes in the tokenized program which are merely skipped during execution. Spaces between the line number and program statement are removed by the tokenizer.

Program lines can be 80 characters total on most machines, but machines with 40 column text would cause the line to wrap around to the next line on the screen, and on the VIC-20, which had a 22 column display, program lines could occupy as many as four. BASIC 7.0 on the Commodore 128 increased the limit of a program line to 160 characters (four 40-column lines or two 80-column lines). By using abbreviations such as `?` instead of `PRINT`, it is possible to fit even more on a line. BASIC 7.0 displays a `?STRING TOO LONG` error if the user enters a program line over 160 characters in length. Earlier versions do not produced an error and simply display the READY prompt two lines down if the line length is exceeded. The line number is counted in the number of characters in the program line, so a five digit line number will result in four fewer characters allowed than a one digit number.

The order of execution of Commodore BASIC lines was not determined by line numbering; instead, it followed the order in which the lines were linked in memory.[7] Program lines were stored in memory as a singly linked list with a pointer (containing the address of the beginning of the next program line), a line number, and then the tokenized code for the line. While a program was being entered, BASIC would constantly reorder program lines in memory so that the line numbers and pointers were all in ascending order. However, after a program was entered, manually altering the line numbers and pointers with the POKE commands could allow for out-of-order execution or even give each line the same line number. In the early days, when BASIC was used commercially, this was a software protection technique to discourage casual modification of the program.

Line numbers can range from 0 to 65520 and take five bytes to store regardless of how many digits are in the line number, although execution is faster the fewer digits there are. Putting multiple statements on a line will use less memory and execute faster.

`GOTO` and `GOSUB` statements will search downward from the current line to find a line number if a forward jump is performed, in case of a backwards jump, they will return to the start of the program to begin searching. This will slow down larger programs, so it is preferable to put commonly used subroutines near the start of a program.

Variable names are only significant to 2 characters; thus the variable names `VARIABLE1`, `VARIABLE2`, and `VA` all refer to the same variable.

Commodore BASIC also supports bitwise operators — `AND`, `OR`, and `XOR`, although this feature was part of the core Microsoft 6502 BASIC code, it was usually omitted in other implementations such as Applesoft BASIC.

The native number format of Commodore BASIC, like that of its parent MS BASIC, was floating point. Most contemporary BASIC implementations used one byte for the characteristic (exponent) and three bytes for the mantissa. The accuracy of a floating point number using a three-byte mantissa is only about 6.5 decimal digits, and round-off error is common. 6502 implementations of Microsoft BASIC utilized 40-bit floating point arithmetic, meaning that variables took five bytes to store (four byte mantissa and one byte for the exponent) unlike the 32-bit floating point found in BASIC-80.

While 8080/Z80 implementations of Microsoft BASIC supported integer and double precision variables, 6502 implementations were floating point only.

Although Commodore BASIC supports signed integer variables (denoted with a percent sign) in the range -32768 to 32767, in practice they are only used for array variables and serve the function of conserving memory by limiting array elements to two bytes each (an array of 2000 elements will occupy 10,000 bytes if declared as a floating point array, but only 4000 if declared as an integer

array). Denoting any variable as integer simply causes BASIC to convert it back to floating point, slowing down program execution and wasting memory as each percent sign takes one additional byte to store (since this also applies to integer arrays, the programmer should avoid using them unless very large arrays are used that would exceed available memory if stored as floating point). Also, it is not possible to `POKE` or `PEEK` memory locations above 32767 with address defined as a signed integer.

A period (.) can be used in place of the number 0 (thus `10 A=.` instead of `10 A=0` or `10 FOR A=. TO 100` instead of `10 FOR A=0 to 100`), this will execute slightly faster.

The `SYS` statement, used to start machine language programs, was added by Commodore and was not in the original Microsoft BASIC code, which featured only the USR function for invoking machine language routines. It automatically loads the CPU's registers with the values in `$30C-$30F` (C64, varies on other machines)--this can be used to pass data to machine language routines or as a means of calling kernal functions from BASIC (as an example, `POKE 780,147:SYS 65490` clears the screen).

Since Commodore 8-bit machines other than the C128 cannot automatically boot disk software, the usual technique is to include a BASIC stub like `10 SYS 2048` to begin program execution. It is possible to automatically start software after loading and not require the user to type a `RUN` statement, this is done by having a piece of code that hooks the BASIC "ready" vector at `$0302`.

As with most other versions of Microsoft BASIC, if an array is not declared with a `DIM` statement, it is automatically set to ten elements (in practice 11 since array elements are counted from 0). Larger arrays must be declared or BASIC will display an error when the program is run and an array cannot be re-dimensioned in a program unless all variables are wiped via a CLR statement. Numeric arrays are automatically filled with zeros when they are created, there may be a momentary delay in program execution if a large array is dimensioned.

String variables are represented by tagging the variable name with a dollar sign. Thus, the variables `AA$`, `AA`, and `AA%` would each be understood as distinct. Array variables are also considered distinct from simple variables, thus `A` and `A(1)` do not refer to the same variable. The size of a string array merely refers to how many strings are stored in the array, not the size of each element, which is allocated dynamically. Unlike some other implementations of Microsoft BASIC, Commodore BASIC does not require string space to be reserved at the start of a program.

Unlike other 8-bit machines such as the Apple II, Commodore's machines all have a built-in clock that is initialized to 0 at power on and updated with every tick of the PIA/VIA/TED/CIA timer, thus 60 times per second. It is assigned two system variables in BASIC, `TI` and `TI$`, which both contain the current time. TI is read-only and cannot be modified; doing so will result in a Syntax Error message. `TI$` may be used to set the time via a six number string (an error results from using a string other than six numbers). The clock is not a very reliable method of timekeeping since it stops whenever interrupts are turned off (done by some kernal routines) and accessing the IEC (or IEEE port on the PET) port will slow the clock update by a few ticks.

The `RND` function in Commodore BASIC can use the clock to generate random numbers; this is accomplished by `RND(0)`, however it is of relatively limited use as only numbers between 0 and 255 are returned. Otherwise, `RND` works the same as other implementations of Microsoft BASIC in that a pseudo-random sequence is used via a fixed 5-byte seed value stored at power on in memory locations `$8B-$8F` on the C64 (the location differs on other machines). `RND` with any number higher than 0 will generate a random number amalgamated from the value included with the `RND` function and the seed value, which is updated by 1 each time an RND function is executed. `RND` with a negative number goes to a point in the sequence of the current seed value specified by the number.

Since true random number generation is impossible with the `RND` statement, it is more typical on the C64 and C128 to utilize the SID chip's white noise channel for random numbers.

BASIC 2.0 notoriously suffered from extremely slow garbage collection of strings. Garbage collection is automatically invoked any time a `FRE` function is executed and if there are many string variables and arrays that have been manipulated over the course of a program, clearing them can take more than an hour under the worst conditions. It is also not possible to abort garbage collection as BASIC does not scan the RUN/STOP key while performing this routine. BASIC 4.0 introduced an improved garbage collection system with back pointers and all later implementations of Commodore BASIC also have it.

The `FRE` function in BASIC 2.0 suffered from another technical flaw in that it cannot handle signed numbers over 32768, thus if the function is invoked on a C64 (38k BASIC memory), a negative amount of free BASIC memory will be displayed (adding 65535 to the reported number will obtain the correct amount of free memory). The PET and VIC-20 never had more than 32k of total memory available to BASIC, so this limitation did not become apparent until the C64 was developed. The `FRE` function on BASIC 3.5 and 7.0 corrected this problem and `FRE` on BASIC 7.0 was also "split" into two functions, one to display free BASIC program text memory and the other to display free variable memory.

Alternatives

The Simons' BASIC start-up screen

Many BASIC extensions were released for the Commodore 64, due to the relatively limited capabilities of its native BASIC 2.0. One of the most popular extensions was the DOS Wedge, which was included on the Commodore 1541 Test/Demo Disk. This 1 KB extension to BASIC added a number of disk-related commands, including the ability to read a disk directory without destroying the program in memory. Its features were subsequently incorporated in various third-party extensions, such as the popular Epyx FastLoad cartridge. Other BASIC extensions added additional keywords to make it easier to code sprites, sound, and high-resolution graphics like Simons' BASIC (1983) and Vision BASIC (2022).

Although BASIC 2.0's lack of sound or graphics features was frustrating to many users, some critics argued that it was ultimately beneficial since it forced the user to learn machine language.

The limitations of BASIC 2.0 on the C64 led to use of built-in ROM machine language from BASIC. To load a file to a designated memory location, the filename, drive, and device number would be read by a call: `SYS57812"filename",8`;[8] the location would be specified in the X and Y registers: `POKE780,0:POKE781,0:POKE782,192`;[9] and the load routine would be called: `SYS65493`.[10]

A disk magazine for the C64, *Loadstar*, was a venue for hobbyist programmers, who shared collections of proto-commands for BASIC, called with the `SYS address + offset` command.

From a modern programming point of view, the earlier versions of Commodore BASIC presented a host of bad programming traps for the programmer. As most of these issues derived from Microsoft BASIC, virtually every home computer BASIC of the era suffered from similar deficiencies.[11] Every line of a Microsoft BASIC program was assigned a line number by the programmer. It was common practice to increment numbers by some value (5, 10 or 100) to make inserting lines during program editing or debugging easier, but bad planning meant that inserting large sections into a program often required restructuring the entire code. A common technique was to start a program at some low line number with an `ON...GOSUB` jump table, with the body of

the program structured into sections starting at a designated line number like 1000, 2000, and so on. If a large section needed to be added, it could just be assigned the next available major line number and inserted to the jump table.

In addition, all variables are treated as global variables. Clearly defined loops beyond the FOR...NEXT construct are hard to create, often causing the programmer to rely on the `GOTO` command (this was later rectified in BASIC 3.5 with the addition of the `DO, LOOP, WHILE, UNTIL`, and `EXIT` commands). Flag variables often needed to be created to perform certain tasks.

Later BASIC versions on Commodore and other platforms included a `DELETE` and `RENUMBER` command, as well as an AUTO line numbering command that would automatically select and insert line numbers according to a selected increment. Earlier BASICs from Commodore also lack debugging commands, meaning that bugs and unused variables are hard to trap. `IF...THEN...ELSE` structures, a standard part of Z80 Microsoft BASICs, were added to BASIC 3.5 after being unavailable in earlier versions of Commodore BASIC.

Use as user interface

In common with other home computers, Commodore's machines booted directly into the BASIC interpreter. BASIC's file and programming commands could be entered in direct mode to load and execute software. If program execution was halted using the RUN/STOP key, variable values would be preserved in RAM and could be PRINTed for debugging. The 128 even dedicated its second 64k bank to variable storage, allowing values to persist until a `NEW` or `RUN` command was issued. This, along with the advanced screen editor included with Commodore BASIC gave the programming environment a REPL-like feel; programmers could insert and edit program lines at any screen location, interactively building the program.[12] This is in contrast to business-oriented operating systems of the time like CP/M or MS-DOS, which typically booted into a command line interface. If a programming language was required on these platforms, it had to be loaded separately.

While some versions of Commodore BASIC included disk-specific `DLOAD` and `DSAVE` commands, the version built into the Commodore 64 lacked these, requiring the user to specify the disk drive's device number (typically 8 or 9) to the standard `LOAD` command, which otherwise defaulted to tape. Another omission from the Commodore 64s BASIC 2.0 was a `DIRECTORY` command to display a disk's contents without clearing main memory. On the 64, viewing files on a disk was implemented as loading a "program" which when listed showed the directory as a pseudo BASIC program, with the file's block size as the line number. This had the effect of overwriting the currently loaded program. Addons like the DOS Wedge overcame this by rendering the directory listing direct to screen memory.

Versions and features

A list of CBM BASIC versions in chronological order, with successively added features:

Released versions

- V1.0: PET 2001 with chiclet keyboard and built-in Datassette (original PET)
  - arrays limited to 256 elements
  - PEEK command explicitly disabled over BASIC ROM locations above $C000
- V2.0 (first release): PET 2001 with full-travel keyboard & upgrade ROMs
  - add IEEE-488 support
  - improved the garbage collection[13]

- fix array bug
- Easter egg – entering `WAIT6502, [number]` displays `MICROSOFT!` an arbitrary number of times

- V4.0: PET/CBM 4000/8000 series (and late version PET 2001s)
  - disk operations: `DLOAD,DSAVE,COPY,SCRATCH,` etc. (15 in all)
  - disk error-channel variables: `DS,DS$`
  - greatly improved garbage-collection performance[13]
- V2.0 (second release, after 4.0): VIC-20; C64
- V2.2 C64GS (1990)
- V4+ : CBM-II series (aka B, P range)
  - memory management: `BANK`
  - more disk operations: `BLOAD, BSAVE,DCLEAR`
  - formatted printing: `PRINT USING,PUDEF`
  - error trapping: `DISPOSE`
  - alternative branching: `ELSE`
  - dynamic error handling: `TRAP,RESUME,ERR$()`
  - flexible `DATA` read: `RESTORE [line number]`
  - string search function: `INSTR`
- V3.5: C16/116, Plus/4
  - sound and graphics commands
  - joystick input: `JOY`
  - decimal ↔ hexadecimal conversion: `DEC(),HEX$()`
  - structured looping: `DO,LOOP,WHILE,UNTIL,EXIT`
  - function key assignment: `KEY` (also direct mode)
  - program entry/editing: `AUTO,DELETE,RENUMBER`
  - debugging (tracing): `TRON, TROFF`
  - MLM entry command: `MONITOR`
  - C(1)16, Plus/4 Easter egg – enter `SYS 52650`
- V7.0: C128
  - more sound and graphics commands, including sprite handling
  - built-in sprite editor: `SPRDEF`
  - multi-statement blocks for `IF THEN ELSE` structures: `BEGIN,BEND`
  - paddle, lightpen input: `POT,PEN`
  - exclusive or function: `XOR`
  - get variable address: `POINTER`
  - text mode windowing: `WINDOW`
  - controlled time delay: `SLEEP`
  - memory management: `SWAP,FETCH,STASH,FRE(1)`
  - used the 128's bank switching to store program code separately from variables. Variable values would be preserved across program executions if the program was started with the GOTO command.
  - more disk operations: `BOOT,DVERIFY`
  - CPU speed adjustment: `FAST,SLOW` (2 vs 1 MHz)
  - enter C64 mode: `GO64`

- undocumented, working: `RREG` (read CPU registers after a `SYS`)
- unimplemented commands: `OFF,QUIT`
- C128 Easter egg – enter `SYS 32800,123,45,6`

Unreleased versions

- V3.6 : Commodore LCD (unreleased prototype). Almost identical to V7.0, with the following differences:[14]
  - `VOLUME` instead of `VOL`
  - `EXIT` instead of `QUIT`
  - `FAST,SLOW` commands not present
  - Additional command: `POPUPS`
- V10 : Commodore 65 (unreleased prototype)
  - graphics/video commands: `PALETTE,GENLOCK`
  - mouse input: `MOUSE,RMOUSE`
  - text file (`SEQ`) utility: `TYPE`
  - program editing: `FIND,CHANGE`
  - memory management: `DMA,FRE(2)`
  - unimplemented commands: `PAINT,LOCATE,SCALE,WIDTH,SET,VIEWPORT,PASTE,CUT`

Notable extension packages

- Super Expander (VIC-20; delivered on ROM cartridge) (Commodore)
- Super Expander 64 (C64; cartridge) (Commodore)
- Simons' BASIC (C64; cartridge) (Commodore)
- Graphics BASIC (C64; floppy disk) (Hesware)
- BASIC 8 (C128; floppy disk and optional internal ROM chip) (Walrusoft)
- Vision BASIC (https://visionbasic.net/) (C64; floppy disk) (Dennis Osborn)

References

1. Stated by Jack Tramiel at the Commodore 64 25th Anniversary Celebration at the Computer History Museum December 10, 2007 [1] (http://www.computerhistory.org/events/index.php?id=1193702785) Archived (https://web.archive.org/web/20081211123347/http://www.computerhistory.org/events/index.php?id=1193702785) 2008-12-11 at the Wayback Machine[2] (http://www.computerhistory.org/events/video/75/) Archived (https://web.archive.org/web/20171003111509/http://www.computerhistory.org/events/video/75/) 2017-10-03 at the Wayback Machine[3] (https://www.youtube.com/watch?v=NBvbsPNBIyk).
2. "Bill Gates' Personal Easter Eggs in 8 Bit BASIC - pagetable.com" (http://www.pagetable.com/?p=43). *www.pagetable.com*. Retrieved 8 August 2018.
3. "Keyboarding and the Screen Editor" (https://archive.org/stream/Commodore_Microcomputers_Volume_6_Number_4_Issue_36_1985-07_Commodore_US#page/n41/mode/2up). July 1985.
4. "Byte July 1983" (https://web.archive.org/web/20170824214757/http://www.classiccmp.org/dunfield/c64/bytejl83.pdf) (PDF). Archived from the original (http://www.classiccmp.org/dunfield/c64/bytejl83.pdf) (PDF) on 2017-08-24. Retrieved 2015-05-01.
5. Leemon, Sheldon (1987). *Mapping the Commodore 64 & 64C* (https://archive.org/stream/Compute_s_Mapping_the_64_and_64C#page/n49/mode/2up). COMPUTE! Publications. p. 37. ISBN 9780874550825. Retrieved 2018-03-25.

6. Cowper, Ottis R. (1986). *Mapping the Commodore 128* (https://archive.org/details/Compute_s_Mapping_the_Commodore_128). COMPUTE! Publications. p. 66 (https://archive.org/details/Compute_s_Mapping_the_Commodore_128/page/n70). ISBN 9780874550603.
7. "Mapping The C64" (http://www.unusedino.de/ec64/technical/project64/mapping_c64.html). *www.unusedino.de*. Retrieved 21 August 2023.
8. Leemon, Sheldon (1987). *Mapping the Commodore 64 & 64C* (https://archive.org/stream/Compute_s_Mapping_the_64_and_64C#page/n221/mode/2up). COMPUTE! Publications. p. 209. ISBN 9780874550825. Retrieved 2018-03-25.
9. Leemon, Sheldon (1987). *Mapping the Commodore 64 & 64C* (https://archive.org/stream/Compute_s_Mapping_the_64_and_64C#page/n83/mode/2up). COMPUTE! Publications. p. 71. ISBN 9780874550825. Retrieved 2018-03-25.
10. Leemon, Sheldon (1987). *Mapping the Commodore 64 & 64C* (https://archive.org/stream/Compute_s_Mapping_the_64_and_64C#page/n243/mode/2up). COMPUTE! Publications. p. 231. ISBN 9780874550825. Retrieved 2018-03-25.
11. "Compute!'s First Book of Atari" (https://www.atariarchives.org/c1ba/page007.php). *www.atariarchives.org*. Retrieved 21 August 2023.
12. Scrimshaw, N. B. (11 November 2013). *An Introduction to the Commodore 64: Adventures in Programming* (https://books.google.com/books?id=UbvzBwAAQBAJ&pg=PA9). Birkhäuser. ISBN 9781489967879.
13. "BASIC 4.0 Memory Map" (http://www.zimmers.net/anonftp/pub/cbm/firmware/README.txt). *zimmers.net*. Retrieved 21 August 2023.
14. "Mike Naberezny – Commodore LCD Firmware" (http://mikenaberezny.com/2008/10/04/commodore-lcd-firmware/). *mikenaberezny.com*. Retrieved 8 August 2018.

Sources

- Commodore/Microsoft Basic version timeline (http://www.weihenstephan.org/~michaste/pagetable/wait6502/msbasic_timeline.pdf)
- Bill Gates' Personal Easter Eggs in 8 Bit BASIC (http://www.pagetable.com/?p=43), pagetable.com

BASIC 2.0

- Angerhausen et al. (1983). *The Anatomy of the Commodore 64* (for the full reference, see the C64 article).

BASIC 3.5

- Gerrard, Peter; Bergin, Kevin (1985). *The Complete COMMODORE 16 ROM Disassembly*. Gerald Duckworth & Co. Ltd. ISBN 0-7156-2004-5.

BASIC 7.0

- Jarvis, Dennis; Springer, Jim D. (1987). *BASIC 7.0 Internals*. Grand Rapids, Michigan: Abacus Software, Inc. ISBN 0-916439-71-2.

BASIC 10.0

- c65manual.txt Commodore 65 preliminary documentation (March 1991), with addendum for ROM version 910501. (http://www.zimmers.net/cbmpics/cbm/c65/c65manual.txt)

- ■