# Create your own Version of Microsoft BASIC for 6502

2008-10-20 by Michael Steil

*Update*: The source is available at github.com/mist64/msbasic

If you disassemble a single binary, you can never tell why something was done in a certain way. If you have eight different versions, you can tell a lot. This episode of "Computer Archeology" is about reverse engineering eight different versions of Microsoft BASIC 6502 (Commodore, AppleSoft etc.), reconstructing the family tree, and understanding when bugs were fixed and when new bugs, features and easter eggs were introduced.

This article also presents a set of assembly source files that can be made to compile into a byte exact copy of seven different versions of Microsoft BASIC, and lets you even create your own version.

## Microsoft BASIC for MOS 6502

Twitter:
@pagetable

Mastodon:
@pagetable@mastodon.social

Categories

6502 Amiga Apple
Archeology
BASIC Bildschirmtext
C64 C128

First written in 1976, Microsoft BASIC for the 8 bit MOS 6502 has been available for virtually every 6502-based computer including the Commodore series (PET, C64), the Apple II series, Atari 8 bit machines, and many more.

These are the first eight versions of Microsoft BASIC:

| Name | Release | VER | ROM | FP | ROR | Buffer | Extensions | Version |
|---|---|---|---|---|---|---|---|---|
| Commodore BASIC 1 | 1977 | | Y | 9 | Y | ZP | CBM | 1.0 |
| OSI BASIC | 1977 | 1.0rev3.2 | Y | 6 | Y | | – | 1.0a |
| AppleSoft I | 1977 | 1.1 | N | 9 | Y | 0200 | Apple | 1.1 |
| KIM BASIC | 1977 | 1.1 | N | 9 | N | ZP | – | 1.1a |
| AppleSoft II | 1978 | | Y | 9 | Y | 0200 | Apple | 2 |
| Commodore BASIC 2 | 1979 | | Y | 9 | Y | 0200 | CBM | 2a |
| KBD BASIC | 1982 | | Y | 6 | Y | 0700 | KBD | 2b |
| MicroTAN | 1980 | | Y | 9 | N | ZP | – | 2c |

**Name**: Name of the computer system or BASIC interpreter

**Release**: Release date of this version – not necessarily the date when the source code was forked from Microsoft's

Commodore
Commodore Peripheral Bus Digital Video DOS Final Cartridge III Floppy Disks Game Boy GEOS GitHub Hacks Hardware KERNAL Literature Operating Systems PET Presentation Puzzle SCUMM Security Tapes Teardown TED Tricks Trivia Uncategorized VIC-20 Virtualization Whines X16 x86 Xbox

github

**VER**: Version string inside the interpreter itself

**ROM**: Whether the software shipped in ROM, or was a program supposed to be loaded into RAM

**FP**: Whether the 6 digit or 9 digit floating point library was included. 9 digit als means that long error messages were included instead of two character codes, and the GET statement was supported.

**ROR**: Whether the ROR assembly instruction was used or whether the code worked around it

**Buffer**: Location of the direct mode input buffer; either zero page or above

**Extensions**: What BASIC extensions were added by the OEM, of any.

**Version**: My private version number used in this article and in my combined source

## The Microsoft BASIC 6502 Combined Source Code

Download the assembly source code here: msbasic.zip

In order to assemble if, you will need the CC65 compiler/assembler/linker package.

The source can be assembled into byte-exact versions of the following seven BASICs:

- Commodore BASIC 1
- OSI BASIC

**mist64**

Worked on
x16-rom

Archives

Select Month

Meta

Log in
Entries RSS
Comments RSS
WordPress.org

- AppleSoft I
- KIM-1 BASIC
- Commodore BASIC 2 (PET)
- Intellivision Keyboard Component BASIC
- MicroTAN BASIC

You can build the source by running the shell script make.sh. This will create the seven files cbmbasic1.bin, osi.bin, applesoft.bin, kb9.bin, cbmbasic2.bin, kbd.bin and microtan.bin in the "tmp" directory, which are identical to the original ROMs.

You are welcome to help clean up the source more, to make it more readable and to break features out into CONFIG_* defines, so that the source base can be made more customizable.

Make sure to read on to the end of the article, as it explains more about the source and what you can do with it.

## Microsoft BASIC 1

Ric Weiland, Bill Gates and Monte Davidoff at Microsoft wrote MOS 6502 BASIC in the summer of 1976 by converting the Intel 8080 version. While the former could fit well into 8 KB, so that a computer manufacturer could add some machine-specific I/O code and ship a single 8 KB ROM, code density was less on the 6502, and they could not fit it significantly below 8 KB – it was around 7900 bytes – so that computers with BASIC in ROM would require more than a single 8 KB ROM chip.

Spilling over 8 KB anyway, they decided to also offer an improved version with extra features in a little under 9 KB: This version had a 40 bit floating point library ("9 digits")

instead of the 32 bit one (“6 digits”), and the two-character error codes were replaced with actual error messages:

| 6 digit BASIC | 9 digit BASIC |
| --- | --- |
| ?NF ERROR<br>OK | ?NEXT WITHOUT FOR ERROR<br>OK |

9 digit BASIC also added support for the GET statement to read single keystrokes from the keyboard.

On startup, Microsoft BASIC 6502 asks for the size of memory:

```
MEMORY SIZE?
```

If the user just presses return, BASIC detects the size of memory itself. If, on the other hand, the user enters “A”, it prints:

```
WRITTEN BY RICHARD W. WEILAND.
```

Versions since 1.1 print:

```
WRITTEN BY WEILAND & GATES
```

Then it asks:

```
TERMINAL WIDTH?
```

Microsoft's codebase could also be assembled either for use in ROM or in RAM: The RAM version additionally asks:

```
WANT SIN-COS-TAN-ATN?
```

These four statements are located at the very end of the interpreter image (actually, the init code is at the very end, but that gets overwritten anyway), so that up to 250 more bytes are available for the BASIC program if the start of BASIC RAM was set to the beginning of the SIN/COS/TAN/ATN code ("N"), or to overwrite ATN only ("A") – in this case, the user would gain about 100 bytes extra.

All these questions were very similar to the ones presented on an Intel 8080 BASIC system – after all, BASIC 6502 was a direct port.

The start message looks something like this:

```
MOS TECH 6502 BASIC V1.0
COPYRIGHT 1977 BY MICROSOFT CO.
n BYTES FREE
OK
```

Microsoft's codebase was very generic and didn't make any assumptions on the machine it was running on. A single binary image could run on any 6502 system, if the start of RAM was set correctly, the calls to "MONRDKEY", "MONCOUT", "LOAD" and "SAVE" were filled with pointers to the machine-specific I/O code, and the "ISCNTC" function was filled with code to test for Ctrl+C.

Microsoft maintained this source tree internally and, at different points in time, handed their current version of the source to OEMs, which adapted and/or extended it for their machines. While most OEM versions were heavily modified in its user interaction (startup screen, line editing…), most of the code was very similar; some functions were even never changed for any version of BASIC. No OEM ever came back to Microsoft for updates, except for Apple and Commodore, which both synced once each, up to the bugfixed version 2.

## Commodore BASIC 1 (1.0)

The BASIC that shipped with the first Commodore PET in 1977 is the oldest known version of Microsoft BASIC for 6502. It does not say "Microsoft" anywhere, and memory size detection and screen width were hardcoded, so on startup, it just prints `*** COMMODORE BASIC ***`, followed by the number of bytes available for BASIC.

Commodore added the "OPEN", "CLOSE", "PRINT#", "INPUT#" and "CMD" statements for file I/O and added VERIFY to compare a program in memory to a file on a storage device. They also added "SYS" to call into assembly code – Microsoft's code had only provided the "USR" function with a similar purpose. It seems Commodore didn't like the "OK" prompt, so they renamed it to "READY.".

All machine-specifics were properly abstracted by calls into the KERNAL jump table, the upper 7 KB of the 16 KB ROM – except for one call out into the screen editor part of the PET ROM:

```
        iny
        lda     (INDEX),y
.ifdef CONFIG_CBM1_PATCHES
        jsr     LE7F3   ; patch
.else
        ldy     #$00
        asl     a
        adc     #$05
.endif
        adc     INDEX
        sta     INDEX
        bcc     L33A7
        inc     INDEX+1
```

This code fixes the garbage collector by doing the missing ldy/asl/adc in the patch code.

Speaking of patches: Commodore BASIC 1 has been binary patched a lot: There are six patch functions appended to the very end of the interpreter image that work around miscellaneous fixes. This is what one of these calls into a patch function looks like:

```
.ifdef CONFIG_CBM1_PATCHES
      jmp     PATCH1
.else
      clc
      jmp     CONTROL_C_TYPED
.endif
```

Here is the patch function – someone indeed forget to clear the carry flag:

```
PATCH1:
        clc
        jmp     CONTROL_C_TYPED
```

Some of these patches are in generic code, and some in Microsoft-specific code. Later fixes in generic code are not necessarily identical to these patches. So this indicates that Commodore wrote the fixes. But it is unknown why these additions were done in the binary as opposed to the source: Commodore had the source and made lots of additions to it. Maybe it was just more convenient to patch the binary for debugging at some point.

## Ohio Scientific (1.0a)

Ohio Scientific sold a wide series of 6502-based machines for several years, but they all shipped with the same version of 6 digit BASIC bought from Microsoft in 1977.

6 digit vs. 9 digit was probably a compile time option, because the differences are pretty straightforward, as can be seen in this example:

```
; ----------------------------------------------------------------------------
; ADD MANTISSAS OF FAC AND ARG INTO FAC
; ----------------------------------------------------------------------------
FADD4:
        adc     ARGEXTENSION
        sta     FACEXTENSION
.ifndef CONFIG_SMALL
        lda     FAC+4
        adc     ARG+4
        sta     FAC+4
.endif
```

```
        lda     FAC+3
        adc     ARG+3
        sta     FAC+3
        lda     FAC+2
        adc     ARG+2
        sta     FAC+2
        lda     FAC+1
        adc     ARG+1
        sta     FAC+1
        jmp     NORMALIZE_FAC5
```

Ohio Scientific only made minimal adaptions for their computers, and added no extensions. It asks for memory size and terminal width, and then prints `OSI 6502 BASIC VERSION 1.0 REV 3.2"`.

One quirk on the Ohio Scientific is the inclusion of the `WANT SIN-COS-TAN-ATN` string, although BASIC ran in ROM. The code to print this string and adjust memory layout accordingly is not included. OSI BASIC is 7906 bytes in size. Without the extra string, they could have saved 21 bytes.

The string Garbage Collector was horribly broken in OSI BASIC, effectively destroying all string data – in Commodore BASIC 1, it had been binary patched for fix the problem.

## AppleSoft I (1.1)

Apple shipped the first Apple II systems with Integer BASIC in ROM, Microsoft BASIC was only available as an option loaded from disk or tape. AppleSoft BASIC, as it was named, had only minor adaptions and extensions. On startup, it printed:

```
APPLE BASIC V1.1
COPYRIGHT 1977 BY MICROSOFT CO.
```

In order to make AppleSoft feel more like Integer BASIC, it showed a ‘]’ character instead of “OK” and said “ERR” instead of ERROR.

The memory size easter egg was modified in this version, it printed `COPYRIGHT 1977 BY MICROSOFT CO` instead of Weiland’s and Gates’ names. Since the Apple II character output code ignored the uppermost bit, this text could be hidden in ROM by setting the MSBs of every character:

```
.;287F C3 CF D0 D9 D2 C9 C7 C8 "COPYRIGH"
.;2887 D4 A0 B1 B9 B7 B7 A0 C2 "T 1977 B"
.;288F D9 A0 CD C9 C3 D2 CF D3 "Y MICROS"
.;2897 CF C6 D4 A0 C3 CF 0D 00 "OFT CO."
```

This version introduced another easter egg present in all later versions: BASIC 1.1 was the first version to include the “MICROSOFT!” easter egg text, as described in a previous article. The encoded (XOR 0x87) text was hidden in some floating point constants and never addressed.

AppleSoft I is the oldest known BASIC 1.1. Compared to 1.0, version 1.1 included minor bugfixes in GET/INPUT/READ, TAB() and LIST, as well as the fix in the Garbage Collector present in the Ohio Scientific machines and binary patched in Commodore BASIC 1.

BASIC 1.0 also had a bug where lines in direct mode that started with a colon were ignored:

```
        jsr     CHRGET
.ifdef CONFIG_11
        tax
```

```
.endif
        beq     L2351
```

CHRGET is supposed to set the zero flag on the end of an instruction, which can be end of line (0 character) or a colon. The original code wanted to check for an empty line and got the first character, and went on reading another line of it was empty – but a colon as the first character had the same effect. 1.1 fixed this by setting the flags on the value again.

Version 1.1 also contained various tiny speed optimizations: BEQs and BNEs were changed so that a cycle could be saved on the more likely case.

Here is another optimization in LEFT$/RIGHT$/MID$:

```
.ifndef CONFIG_11
        sta     JMPADRS+1
        pla
        sta     JMPADRS+2
.else
        tay
        pla
        sta     Z52
.endif
[...]
.ifdef CONFIG_11
        lda     Z52
        pha
        tya
        pha
.endif
        ldy     #$00
        txa
.ifndef CONFIG_11
        inc     JMPADRS+1
        jmp     (JMPADRS+1)
```

```
.else
        rts
.endif
```

The original code isn't only suboptimal, it's even dangerous, because it only increments the low byte of the address it wants to jump to and assumes it doesn't roll over to $00.

For some reason, the random number seed was changed slightly:

```
    .ifdef CONFIG_11
        .byte   $80,$4F,$C7,$52,$58
    .else
        .byte   $80,$4F,$C7,$52,$59
    .endif
```

But this doesn't make a difference, due to a bug present in all 9 digit versions of BASIC: The value is copied into the zero page together with the CHRGET routine:

```
.ifdef CONFIG_SMALL
        ldx     #GENERIC_CHRGET_END-GENERIC_CHRGET
.else
        ldx     #GENERIC_CHRGET_END-GENERIC_CHRGET-1
.endif
L4098:
        lda     GENERIC_CHRGET-1,x
        sta     CHRGET-1,x
        dex
        bne     L4098
```

On 9 digit BASIC, one extra byte had to be copied, but the start index was not changed, so the last digit was omitted. This bug exists in every known version of Microsoft BASIC.

Another bug was introduced on the Apple II: All previous versions of BASIC had the input buffer for instructions in direct mode in the zero page. On the Apple II, it was at $0200 in RAM, which broke some code that made assumptions on the address:

```
NEWSTT:
        jsr     ISCNTC   ; check for Ctrl + C
        lda     TXTPTR
        ldy     TXTPTR+1 ; high-byte of instruction pointer
        beq     L2683    ; 0 -> direct mode
        sta     OLDTEXT
        sty     OLDTEXT+1
```

Subsequent versions of BASIC compared the high-address of the text pointer:

```
        cpy     #>INPUTBUFFER
```

## KIM-1 (1.1a)

The KIM-1 is a computer kit based around the MOS 6502, which was sold by the makers of the 6502 to show off the capabilities of this CPU. A 6 digit and a 9 digit version of Microsoft BASIC was available on tape, but the 6 digit version seems to be very rare. BASIC for the KIM-1 is the most authentic version of Microsoft BASIC, because it has only been minimally modified, it contains all questions about memory size, screen width, and the trigonometric functions, as well as the memory width easter egg. The encoded "MICROSOFT!" string can be found at the same spot as on the Apple II.

Although this is based on BASIC 1.1, just like AppleSoft I, there are a few fixes in array handling and the PRINT statement.

But they also introduced another bug: In input handling, again concerning the location of the input buffer, there is the following code:

```
ldx     #<(INPUTBUFFER-1)
ldy     #>(INPUTBUFFER-1)
bne     L2AF8   ; always
```

This code has been in place since 1.0 and assumes that INPUTBUFFER is above $0100. On the CBM1, which had the input buffer in the zero page, this had been hotfixed by Commodore by swapping the ldx and the ldy. On the OSI, this code didn't exist, as it is only included in versions that have the GET statement, i.e. 9 digit versions. AppleSoft I was not affected either, because it had the input buffer at $0200. And versions after the KIM fixed this by replacing the BNE with a BEQ in case the input buffer is in the zero page. It is obviously hard to maintain a single codebase with many compile time options that still does optimizations like these.

Since the first KIM-1 systems shipped in late 1975, their CPUs had the 6502 ROR bug, so KIM-1 BASIC had to work around this: Every ROR instruction is replaced by a corresponding sequence using LSR instead.

## AppleSoft II (2.0)

AppleSoft II is the oldest version of Microsoft BASIC 2. It was available on tape or disk, and also in ROM in later Apple II models. It is the first BASIC from an OEM that had extended

BASIC which was re-sync'ed with Microsoft's codebase. In other words: Apple licensed an improved and bugfixed version of BASIC, and merged their old changes into it.

BASIC 2 contains mostly bugfixes (all input buffer location bugs have finally been eliminated), small optimizations (reuse two adjacent zeros inside the floating point constant of 1/2 as the 16 bit constant of zero instead of laying it down separately), better error handling for DEF FN, and support for "GO TO" with a space in between as a synonym for GOTO. Also, the memory test pattern has been changed from $92/$24 to the more standard $55/$AA.

In AppleSoft II, Apple also eliminated the "memory size" and "terminal width" questions.

## Commodore BASIC 2 (2.0a)

Just like Apple, Commodore went back to Microsoft for an updated version of BASIC, and integrated its changes into the new version. The version they got was slightly newer than Apple's, but the major difference was that Microsoft added the "WAIT 6502" easter egg. For this, they changed the encoding of the string "MICROSOFT!" that was hidden in every BASIC since 1.1 from XORed ASCII into PETSCII with the upper two bits randomly set – this way, the text would be just as obfuscated, but it the decoder would be shorter on PET systems. So Commodore BASIC 2 is the only version of Microsoft BASIC that ever accesses this hidden text.

Every version since 2.0a had the PETSCII version of the "MICROSOFT!" text in it – and so did every version of BASIC for 6809.

## Intellivision Keyboard Component BASIC (2.0b)

The Mattel Intellivision is a game console released in 1980 that contained a very nonstandard 16 bit “CP1610” CPU. After a series of delays, the “Keyboard Component“, an extension with its own 6502 CPU and Microsoft BASIC, was released in 1982, but canceled very soon. They are very rare today.

The BASIC in the Keyboard Component is the most custom of all known versions. It is based on a 6 digit version of BASIC 2 and younger than Commodore BASIC 2: It contains two bugfixes: One piece of code that pulled its caller’s address from the stack and normalized it by adding one, had forgotten to respect the carry, so this could fail if the caller sits just on a page boundary. The other fix changed the number of steps needed for normalizing a floating point number.

Intellivision BASIC replaced LOAD and SAVE by PLOD, PSAV, VLOD, VSAV and SLOD, PRT, GETC and VER were added, and PEEK, POKE and WAIT were removed. But the customizations were even more extensive: Instead of keeping the interface to library code, a lot of code was replaced inline, and the whole init code was rewritten. While most of the generic code, for example memory handling was unchanged across Commodore, Ohio, AppleSoft and KIM, making it easier to later integrate Microsoft’s fixes, some of even this code was altered on the Keyboard Component.

What is interesting about the strings in Intellivision BASIC is that they use both upper- and lower case. The start message is this:

```
INTELLIVISION BASIC
Copyright Microsoft, Mattel  1980
```

But upper-/lowercase support doesn't stop here: The complete code has been extended to be case insensitive, but case preserving. The CHRGET code, a super-optimized function living in the zeropage has been patched with a call to this function:

```
LF430:
        cmp     #'a'
        bcc     LF43A
        cmp     #'z'+1
        bcs     LF43A
        sbc     #$1F
LF43A:
        rts
```

This very unoptimized piece of code adds at least 17 cycles to every CHRGET, and will slow down execution measurably.

## Microtan BASIC (2.0c)

The version of BASIC that shipped on the Tangerine MICROTAN 65 is, like the Ohio and KIM versions, again a very authentic version with few changes. The updated BASIC 2 contained a single bug fix, which is the floating point constant of -32768 which hadn't been updated from 6 to 9 digits correctly and was missing a byte. The startup message looks like this:

```
MICROTAN BASIC
(C) 1980 MICROSOFT
```

Microtan BASIC contains the complete "memory size" and terminal width procedures and the "Weiland/Gates" easter egg.

Although the Microtan was introduced in 1980, its version of BASIC was, like the KIM version, assembled with code that worked around the ROR bug in 6502 chips until mid-1976. The I/O library on the other hand made use of ROR, suggesting that this compile time option was set in error.

## Bugs never fixed

As you can see, the first versions had many bugs that were quickly fixed, but fixed became less and less – simply because there were only very few bugs left. But still there are some bugs that never got fixed. The short copy of the random number seed for example, exists on all versions.

Similarly, the two extra constants used for generating random numbers (CONRND1, CONRND2) are 4 bytes in all versions, which is one byte short for 9 digit BASIC. But this is another bug that doesn't really matter, since the numbers will still be random enough.

The buggy check on large line numbers has also never been fixed. Typing 35072121 into any version of Microsoft BASIC will have the interpreter jump to a pseudo random memory address. The buggy code resides in "LINGET".

Something similar happens in the case of `PRINT 5+"A"+-5`: The interpreter will build up the formula on the CPU stack, but miss the string/float type mismatch because of the "+-", and messes up its stack when removing items. This bug is in "FRMEVL".

But the fact that Microsoft never fixed these bugs in their codebase doesn't mean none of the OEMs fixed them. While the LINGET and FRMEVL seem to have been unnoticed everywhere, at least the CONRND1/CONRND2 bug has been fixed by Commodore, at least as early as for the VIC-20 in 1980.

## How to build your own BASIC

Now that you have the source that can build seven different OEM versions of Microsoft BASIC, and that you know about the differences between those, you might be interested in building your own version of BASIC 6502 for some 6502-based machine or customizing BASIC to build a bugfixed or extended version for some platform.

First duplicate one of the cfg files, and add it to make.sh. cbmbasic2 is a good start, as you can quite easily test the resulting images in the VICE emulator – CC65 can even provide symbol information for the VICE debugger. Add a case in defines.s to define one of CBM1, CBM2, APPLE etc., because you need one flavour of platform specific code, and include your own defines_*.s. For Commodore BASIC, you also need to define CONFIG_CBM_ALL.

If you are targeting a new type of computer, make sure to adjust the zero page locations in your defines_*.s file (ZP_STARTn) so that they don't clash with your I/O library. Also make sure that, in case you are compiling for RAM, the init code does not try to detect the memory size and overwrite itself.

The CONFIG_n defines specify what Microsoft-version the OEM version is based on. If CONFIG_2B is defined, for example, CONFIG_2A, CONFIG_2, CONFIG_11A, CONFIG_11 and CONFIG_10A will be defined as well, and all bugfixes up to version 2B will be enabled. The following symbols can be defined in addition:

| | |
|---|---|
| CONFIG_CBM1_PATCHES | jump out into CBM1's binary patches instead of doing the right thing inline |
| CONFIG_CBM_ALL | add all Commodore-specific additions except file I/O |

| | |
|---|---|
| CONFIG_DATAFLG | ? |
| CONFIG_EASTER_EGG | include the CBM2 “WAIT 6502” easter egg |
| CONFIG_FILE | support Commodore PRINT#, INPUT#, GET#, CMD |
| CONFIG_IO_MSB | all I/O has bit #7 set |
| CONFIG_MONCOUT_DESTROYS_Y | Y needs to be preserved when calling MONCOUT |
| CONFIG_NO_CR | terminal doesn’t need explicit CRs on line ends |
| CONFIG_NO_LINE_EDITING | disable support for Microsoft-style “@”, “_”, BEL etc. |
| CONFIG_NO_POKE | don’t support PEEK, POKE and WAIT |
| CONFIG_NO_READ_Y_IS_ZERO_HACK | don’t do a very volatile trick that saves one byte |
| CONFIG_NULL | support for the NULL statement (send sync 0s for serial terminals) |
| CONFIG_PEEK_SAVE_LINNUM | preserve LINNUM on a PEEK |

| | |
|---|---|
| CONFIG_PRINTNULLS | whether PRINTNULLS does anything |
| CONFIG_PRINT_CR | print CR when line end reached |
| CONFIG_RAM | optimizations for RAM version of BASIC, only use on 1.x |
| CONFIG_ROR_WORKAROUND | use workaround for buggy 6502s from 1975/1976; not safe for CONFIG_SMALL! |
| CONFIG_SAFE_NAMENOTFOUND | check both bytes of the caller's address in NAMENOTFOUND |
| CONFIG_SCRTCH_ORDER | where in the init code to call SCRTCH |
| CONFIG_SMALL | use 6 digit FP instead of 9 digit, use 2 character error messages, don't have GET |

Changing symbol definitions can alter an existing base configuration, but it is not guaranteed to assemble or work correctly.

I am very interested in your creations. Please add a comment to this article if you have made something new out of this source base!

## Using the Floating Point Library Standalone

The complete project has been split into many components, each in their own assembly source file. The core floating point library is in float.s, extra trigonometric functions are in trig.s. It should not be too hard to use this broken-out part (in a 6 digit or 9 digit version) standalone in your own creations. The 9 digit version is a little over 2 KB in size, the 6 digit version is a little smaller.

## Adding More Versions

If you want to add another version of BASIC into the source base, you can do it like this: Use "da65" from the CC65 package to dissemble your version of BASIC and all existing .bin files (with the correct base addresses), and run a "diff" command on the new disassembly and each of the disassemblies of the existing versions. The diff that contained the fewest changes (just look at the file size) is probably a good candidate to base your new version on. Or look at the release date or the family tree to find a version which is similar.

Now create a new version in the source base, as described earlier. Make sure the new version assembles; then compare the disassembly of your version with the disassembly of the original binary in a diff program, like the excellent Mac OS X FileMerge, to find the differences. In most cases, you will only have to adjust a few defines (CONFIG_* and zero page locations) in your defines_*.s file to get matching output. Otherwise, add ifdefs to the respective source files. Run regress.sh to verify that you didn't break the other versions.

Repeat the last step until the assembly process outputs the same file. Send your changes to me. :-)

Note that the idea of all versions of BASIC in the current source code is that they are all direct forks from Microsoft's codebase. I chose not to include versions like Commodore BASIC 4, Commodore BASIC 2 for the VIC-20/C64 etc., and I wouldn't add very late

AppleSoft versions, because these are only extended versions of earlier forks and contain no extra code from the original Microsoft source base. Versions that would be very interesting to integrate would be AppleSoft II and Atari Microsoft BASIC, preferably the very first revisions of these.

## Credits

- Function names and all uppercase comments taken from Bob Sander-Cederlof's excellent AppleSoft II disassembly
- AppleSoft lite by Tom Greene helped a lot, too.
- Thanks to Joe Zbicak for his help with Intellivision Keyboard BASIC
- This work is dedicated to the memory of my dear hacking pal Michael "acidity" Kollmann.

6502, Archeology, BASIC, GitHub, Hacks

‹ "ROR" in Microsoft BASIC for 6502
› Transactor November 1987: Volume 8, Issue 3 (PDF)

## 57 thoughts on "Create your own Version of Microsoft BASIC for 6502"

**Marco**

2008-10-21 at 00:34 | Reply

Which version of cc65 are you using? The posted source contains c-style comments in inline.s, poke.s and init.s which are not supported by V2.12.9 (20080831 snapshot) or the V2.12.0 release. If I fix this, then the resulting cbmbasic2.bin in tmp is 8670 bytes in length and does not run in vice (x64 -basic tmp/cbmbasic2.bin hangs). Any hints?

**Michael Steil**
2008-10-21 at 01:25 | Reply

@Macro: I'm using a V2.12.9 that I compiled myself. Weird. I would be happy about your patches. :-)

Here are the MD5s of the original binaries:
applesoft.bin = 84b510725233ce4a9f12894d17c0c056
cbmbasic1.bin = 0aae8577c9e2d78a869f15fe73e3c0db
cbmbasic2.bin = 65fbddc1114c5ca4648cf31d6a9a2891
kb9.bin = 3272e79a92398bfceaab2f8178ce382f
kbdbasic.bin = 639f953ac2eb1d1ceb307eea843f3c27
microtan.bin = b28f19e5947bed5c771990ad0eb3fe36
osi.bin = 1f45d3c8c7e684901b453a443a93f2e9

Commodore BASIC 2 is the PET version and won't just work on a C64, for several reasons:

* It is compiled for $C000-$E000; the C64 has its own ROM and other parts there.
* It does a memory test by reading/writing memory (without preserving the data) until it hits unassigned memory or ROM – if you link cbmbasic2 to another address in RAM, it will overwrite itself.
* RAMSTART is $0400, while it should be $0800 on the C64.

These items can probably be fix pretty easily, but:

* Zero page layout is different on the PET and the C64: You must make sure that BASIC does not use zero page locations used by KERNAL. After all, BASIC on Commodore computers is not standalone, but requires the services of KERNAL, especially the screen editor.

Please note that the “Commodore BASIC V2” used in the VIC-20 and the C64 is unrelated to version 2 in the PET: C64 BASIC is based on the PET BASIC 4.0 source code, with all post-2.0 features stripped out again for space reasons. Among other things, C64 BASIC used CR instead of CR/LF everywhere, and it had some better abstractions between BASIC and the KERNAL: For example, on the C64, KERNAL detects the size of memory, and BASIC queries this information through a call into the KERNAL jump table.

So I suggest you replace the BASIC2 ($C000-$E000) ROM and the lower part of EDIT2G ($E000-E1DE) ROM in VICE with your assembled binary. Be careful that you don’t overwrite the screen editor code. This way, your binary will work without changes. Be careful though to configure VICE to emulate the correct version of the PET.

### Marco

2008-10-21 at 02:38 | Reply

– I still can't figure out, why you can use c-style comments in your source. But since putting a semicolon in front of /* suffices, I won't bother.
– My checksums are OK.
– Thanks for your suggestions regarding xpet. This seems to work. I've written a script to launch xpet for the current build of cbmbasic2:

```
#!/bin/sh
mkdir -p pet
vice_dir=$(dirname $(readlink -f $(which xpet)))/..
cp $vice_dir/lib/vice/PET/edit2g pet
cp $vide_dir/lib/vice/PET/basic2 pet
dd if=tmp/cbmbasic2.bin of=pet/basic2 bs=1 count=8192
dd if=tmp/cbmbasic2.bin of=pet/edit2g conv=notrunc bs=1 skip=8192
xpet -model 3032 -basic pet/basic2 -editor pet/edit2g
#—
```

Apart from that: great work!

### Mike Cohen

2008-10-21 at 07:14 | Reply

One interesting quirk of OSI basic was how it displayed the two-letter error codes. The codes were stored internally with the high bit set on the second character. Since OSI systems used the high bit for graphics characters, it displayed a graphic character for the second character. I remember seeing lots of ‘SËŠ Error’ messages :)

**Michael Steil**
2008-10-21 at 11:19 | Reply

@Mike: Yes, the source shows that this was fixed in Intellivision BASIC:

program.s:

```
      lda     ERROR_MESSAGES+1,x
.ifdef KBD
      and     #$7F
.endif
      jsr     OUTDO
```

The “.ifdef KBD” should probably be an “.ifdef CONFIG_2”.

**James Abbatiello**
2008-10-21 at 16:54 | Reply

CONFIG_DATAFLG is for machines with 8-bit character sets. Tokens are stored internally as bytes with the high bit set. When you LIST a program and it sees a byte with the high bit set it normally prints out the string representation of the token. With CONFIG_DATAFLG on it keeps track of whether you are in a quoted string and prints the token if you aren't or the literal character if you are. This lets you have literal strings containing characters with the high bit set and still have your programs list properly.

It also causes the character 0xFF to list literally whether it is in a string or not. On the Commodore 0xFF is the character for pi and you can use it as a constant. The program
10 PRINT Ď□
will print out 3.14159265 when run. When this program is LISTed we want the character to print literally and not be interpreted as a token (0xFF is not assigned to any token).

The OSI machine seems to have a small picture of a tank pointing up and to the left at 0xFF. The special case may have been included in the OSI due to an oversight. I don't know if you get 3.14159265 if you try to evaluate it.

**James Abbatiello**
2008-10-21 at 19:26 | Reply

Update: I tried an OSI C1P emulator and it doesn't allow you to enter the line "10 PRINT [FF]" (where [FF] is the character with code 0xFF). It just drops it silently and deletes any existing definition for line 10 if there is one. If I manually edit the memory to include that statement it produces an error at runtime. So it seems there is no need for the FF special case in the OSI.

**Joe Zbiciak**
2008-10-22 at 09:13 | Reply

I'm still musing over some of the unique aspects of the Intellivision Keyboard port. The Keyboard is such an interesting beast, to be sure. At least knowing the structure of Microsoft BASIC made it easy to intuit what the Keyboard-specific bits were trying to accomplish, even if I wasn't sure *how* they were accomplishing them. A prime example is in kbd_iscntc.s:

```
ISCNTC:
jsr LE8F3
bcc RET1
LE633:
jsr LDE7F
```

beq STOP
cmp #$03
bne LE633This code checks for a control-C, and is supposed to branch to an RTS instruction if no ^C is pending. A cursory inspection of the above code indicates that the function at LE8F3 returns w/ carry set if there is a ^C pending. But what does it do? The code is mysteriously tacked in the end of print.s:
LE8F3:
pha
lda $047F
clc
beq LE900
lda #$00
sta $047F
sec
LE900:
pla
rts
This checks location $047F to see if it's non-zero. If it's non-zero, then it zeros it and returns with carry set, indicating (and acknowledging) ^C or some other error. The rest of the ISCNTC code just drains the keyboard input FIFO either up until the ^C, or until it's empty. (Trust me on that one.)

But where does $047F get set? The only reference I can find is in a timer tick interrupt in the KC's 6502-side EXEC, where $047F gets counted down toward zero! Take a look:
C1CB AD 7F 04 LDA $047F ;
C1CE F0 03 BEQ $C1D3 ; |– If $047F isn't zero yet, decrement it
C1D0 CE 7F 04 DEC $047F ; /

So what the heck is going on here? I've seen lots of references to locations to either side of $047F, but never $047F itself. If this really does get decremented every tick, maybe it gets set to $FF on an error, and the error expires after ~4 seconds???

Clearly, I still haven't figured out all of the communication paths between the Master Component and the Keyboard Component, nor have I figured out where all the different flags get set from. It's definitely an adventure reverse engineering an OS that spans two different CPU types, and includes such bizarre things as 10-bit wide RAM shared between them. :-)

I really just ought to publish what I have reverse-engineering-wise and see what more eyes might bring.

**Joe Zbiciak**
2008-10-22 at 09:14 | Reply

Eeek, my formatting got eaten. To see it with proper formatting, look here:

http://community.livejournal.com/vintagecomputer/66963.html?thread=314771#t314771

**Frank Palazzolo**
2016-12-07 at 12:46 | Reply

> I've recently been working with Intellivision Keyboard BASIC with Joe Zbiciak as well. The mysterious VER function was added and is the only command which is not documented in the Keyboard BASIC manual. Looking at your code, it returns a 19 (0x13) when the argument is 0, which is hardcoded in the BASIC. If the argument is non-zero, it reads a byte from the Keyboard Component's ROM $DFF9, which happens to be 50 (0x32). So, it looks like it can return a version byte for either the BASIC or the Keyboard Component itself.
>
> How to interpret this? Well, it's possible the BASIC is self- labelled 19, 1.9 or 1.3 (from the hex interpretation)
>
> I am now working on restoring the original BASIC software written for that platform, and will keep an eye out for any code that actually uses the VER function.

Pingback: Microsoft Basic, Computer Archeology | Hans' blog

Pingback: Geeknews » Create Your Own Version of Microsoft BASIC for 6502

Pingback: bleuge - 76 tunguskas por minuto

**Harry Dodgson**
2008-10-30 at 02:50 | Reply

Here is a hybrid version:

AIM 65 BASIC V1.1 (C) 1978 MICROSOFT has 9 digit FP,GET, and ROR – but only 2 char error messages. It was shipped in 8K of ROM with the ATN() function left off for users to install to RAM if they wanted it. The message in caps is at the end of the ROM. Only the first part prints on startup.

**Hans Otten**
2008-10-30 at 09:45 | Reply

Great job!
I have been playing with KIM-1 Microsoft Basic long long ago, bought an original version on tape before 1980 and it still loads. (eh, it did two years ago :) .
With the aid of a commented disassembly of PET Basic I could understand the

workings, like the jumps into 3 byte instructions! and improve some nasty shortcomings in KIM Basic, like no working save and load.

What about SYM-1 and AIM-65 Basic? What I have seen of those versions I suppose it somewhere in the 1.1 branch, but much better integrated with the SYM-1 and AIM-65 operatings system.

**Bob Sander-Cederlof**
2008-10-30 at 16:41 | Reply

I thoroughly enjoyed reading this, and also the Easter Egg article. Thanks for all your work researching and writing.

Pingback: pagetable.com » Blog Archive » Commodore BASIC as a Scripting Language for UNIX and Windows - now Open Source

**Jeff Kingsley**
2008-11-04 at 23:18 | Reply

Wow, what a trip back in time! My very first exposure to computers was a KIM-1 while I was doing undergrad research for a chemistry professor. He handed me a book on assembly language for the 6502 and told me to write a monitor program that would get input from the user, dial up the campus mainframe (acoustic 300 baud modem) and upload gas phase electron diffraction data that the KIM-1 had just collected.

I was instantly hooked on programming these new fangled computer gizmos. I bought myself an AIM-65 and all the bells and whistles (16kB RAM, extender board, ROM basic and assembler, etc). I bought a suitcase and mounted the entire thing inside so that I had a ‘portable’ computer.

On the KIM-1, in addition to assembler, we also had MS Basic (on tape of course). There was some floating point math that the computer did prior to uploading the data. We wanted to speed this up so we interfaced an AMD9511 Math chip to the KIM-1 and then put hooks into MS Basic to use the chip instead of the FP routines. Worked great!

A couple of years later I did something similar when the 8087 first came out for the IBM-PC. I created an alternate object library for the MS compilers that used the 8087 instead of the FP routines. I had to convert back and forth between MS FP format and IEEE format, but with my experience on the KIM-1 it was no problem. I got my first pre-production 8087 from a company that had given up doing that very thing due to their frustration in trying to convert the FP formats.

Thanks for the memory jog!

### Steve Chamberlin

2008-11-14 at 21:28 | Reply

> This is fantastic! I'm finishing a homemade 8-bit computer with a homemade microcoded CPU, and planning to get BASIC running on it. My computer's instruction set is a superset of the 6502, so in theory I should be able to start with one of the plain vanilla 6502 BASICs here, plug in my own character read/write routines, and have something up and running. I had feared I'd need to reverse-engineer an exising BASIC, or write my own, but this will make my life much easier. I can't wait to get BASIC running on my machine… thanks!

Pingback: Big Mess o' Wires » BASIC?

### M-ko

2008-11-24 at 10:46 | Reply

> Working on figuring it out so I can have M$ BASIC on my virtual 6502 platforms. Haven't yet. :/

Pingback: 7 BASICs 6502 pour le prix d'un | hilpers

Pingback: Jeff Barr's Blog » Links for Monday, March 1, 2009

**Wim**
2009-03-16 at 05:59 | Reply

> Anybody already worked on a R65C02 source for CBM-BASIC, making use of the STZ, PHX, SMB, RMB etc opcodes ?

**Ralf**
2009-11-27 at 12:37 | Reply

> Anyone knows of a similar endeavor for the x86 based BASIC(A)/GW-BASIC?

**Harry Dodgson**
2010-02-17 at 08:22 | Reply

I just tried:
35072121 PRINT “HI” on AIM-65 BASIC and all I get is an “?SN ERROR”
Does that mean it is fixed?

**goob**
2010-03-07 at 12:13 | Reply

You guys are demented.

The CC65 compiler + VICE C128 emulator is a great way to verify pure C code segments within limited hardware for designing compact platform independent embedded applications…

…but what the hell am I going to do with Microsoft BASIC coded in 6502 assembly!

You guys need something better to do with your time, seriously :-)

### Computer Settings

2010-11-14 at 01:14 | Reply

> Ralf, try to search on Google for the x86 based BASIC(A)/GW-BASIC

Pingback: A 256 Byte Autostart Fast Loader for the Commodore 64 « pagetable.com

### Jonno

2011-09-03 at 02:08 | Reply

> I have just been tracing through a c64 BASIC disassembly (“What’s really inside the Commodore 64”) and noticed a NOP at A7B7 that seems a waste of both space and time (particularly since it’s in the main ‘execute next statement’ loop of the runtime).
>
> Checking the source code (lines 87-89 of flow1.s), it appears this NOP was in the PET BASIC v2, but none of the other 6502 BASICs.
>
> Any idea why it was there?

Pingback: wow

Pingback: Office key

Pingback: Ovulating To Get Pregnant

Pingback: 4x4 articles

Pingback: Programming from Commodore to Intel « Harmonia Philosophica

**Dan Schwartz**
2012-10-21 at 11:19 | Reply

Here is another bug that's in every version of Microsoft Basic, which I have never seen mentioned anywhere. Type in the following program:

```
10 FOR I=1 TO 10
20 PRINT POS("P"+"")
30 NEXT
```

RUN it and you will see:
0
0
?ST (or ?FORMULA TOO COMPLEX) ERROR IN 20

Now, the argument of the POS function is a dummy argument, there is no reason to ever use a string, and certainly not a concatenated string, but still, I consider this a bug. What happens is that the POS function does not check whether its argument is numeric or string, and so in the event it is a string, it fails to clear the "temporary string stack", resulting in the above error. To get strings working again, you need to execute a CLEAR or NEW or RUN (of different program code).

**Johann E. K.**
2016-05-22 at 16:12 | Reply

There are several other missing checks for string type expressions glutting the string descriptor stack like:

print peek("")peek("")peek("")

or

if "" then
if "" then
if "" then

results also into *"?formula too complex"*.
If the error appears the string stack is already reset. If only one or 2 of them happens only a CLR is able to restore the string stack.
Depending on the Basic version the depth of the string stack may vary, e.g. Basic 7.0 on a C128 the 4th `if""then` raises the error.

**Dan Schwartz**
2012-10-21 at 11:45 | Reply

Jonno:

The NOP you refer to occurs immediately after a CPY #$02 instruction, which is to say, a check of whether Basic code is being executed out of the input buffer (in immediate mode). As the article mentions, the check of the input buffer page was quite confused and frequently patched, so apparently the code here was patched in the binary image shortly before being burned to ROM.

**Damian**
2013-01-24 at 17:59 | Reply

probably, a silly question, but when Microsoft was making this basic (OS) version 1,2,3 etc. what was the system they where running or tested the Basic (OS) on, before it got altered to the different machines Atari – Commodore PET (64) – AppleSoft, etc. (was it a big main-frame or (what would be call a emulator-simulator to day). or did they just, demonstratel idea, concept with no machine in mind.
ps. on second question/comment, is this MS BASIC the same TRS-80 (COLOR), – Extended BASIC, (COCO) and or DRAGON 32/64.
(is thee any connection MS-Dos QBasic QB, VB dos, versions for the x86 PC)

I'm only asking because this page makes a lot of sense, of things I have wondered about the different versions of basic, and there family history (tree), connection them. to master version made by Microsoft.
(not quite the same but seem like Linux is re-petting, in the same steps, with all the versions of almost the same software.)

Pingback: Project:65 – Fully armed and operational « Coronax's Project Blog

**Peter Arnt**
2013-03-19 at 18:41 | Reply

A few notes on Ohio Scientific 6502 Basic: Ohio Scientific (OSI) actually had more than a few versions of 6502 basic. They had ROM BASIC for most of their desktop and superboard models. For disk-based systems they offered OS-65D and OS-65U disk operating systems. OS-65D tended to be offered on the lower end and hobbyist systems. OS-65U was offered on their more expensive Hard Disk-based systems geared towards small business. OS-65U was indeed their flagship OS. It did use 9-digit floating point math. 65U has special commands for working with ISAM files. The “FIND” command would perform a linear string search on the contents of a data file. Typically this was used for index/key files and would let the programmer know where to find a specific record within a master data file.

**Peter Arnt**
2013-03-20 at 13:23 | Reply

FYI:

On my Ubuntu 64-bit system using CC65 2.13.9-svn5323-1, msbasic will not build. The new version of the assembler is not processing constant expressions like it used to. Several statements with these expressions are generating “range” errors. Also, C-style comments at the end of .endif’s are being reported as junk characters.

-P.

**Bob Harper**
2013-08-08 at 17:11 | Reply

Thank-you for sharing this ‘archiology’ with the younger people, but especially with us older ‘hackers’ who knew hacking to mean cutting into PCBs to add feathures, i.e. later called ‘Hardware Hackers’. I had the 6th Commodore PET in Australia bought in 1977 at a Hamfest.It had 3.8kB memory, so we either learned machine code or wrote very small BASIC programs. The monitor/BASIC was hacked many times and in many ways but had to be able to be switched back to standard for some bought games, so the ROMs were double stacked, or more, and their enable pins were switched to select which ROM you used. i.e. Dual Boot before any DOS! Some of us had a lot of utilities, and Ham Code such as Morse Code Readers and Writers installed in place of the BASIC or as Plug-in memory that slotted into a socket on the Main PCB.
It was a great computer that weighed a lot due to it’s thick steel cabinet with a bolted on 9” Amber monitor. I donated it back to a Commodorte dealer when Amigas were in vogue, but I still have the manuals. Happy Hacking! (GOOD Hacking might be more appropriate!)

**Paul Londoner**
2013-12-09 at 06:04 | Reply

My ambition is to create a version of “Microsoft Extended BASIC” for the Commodore 64 platform. I’m making great progress learning 6502 Assembler, but I couldn’t achieve this task at the moment. I think it should be like the BASIC on the 6809 based Tandy “Coco”and Dragon 32/64, 8086 based GW BASIC, and/or Z80 based MSX BASIC. The 6502 based Atari Microsoft BASIC is available in an ATR disk image. The reason the Commodore 64 had no commands for colour, graphics, or sound was because Commodore founder and boss Jack Tramiel did a dirty deal with Microsoft that for $10,000 he could use the version of BASIC he bought in 1977 on as many computers as he liked, instead of the alternative deal of $3 per computer sold. I’m sure that Bill Gates, Ric Weiland, and Monte Davidoff never imagined in their worst nightmares that the same BASIC would be built in to a computer 5 years later (i.e. the Commodore 64), meaning that its graphics and sound hardware weren’t supported and that this computer would continue to be sold with the same crappy BASIC until 1994. People criticise Bill Gates, but I think he’s a saint compared to Jack Tramiel! I was unlucky enough to get a Commodore 64 as my first computer, because I hoped to play music on it. This was a few months after Jack Tramiel left Commodore. I sold it 10-11 months later because of the mind numbing stress that Commodore BASIC V2 caused me, and bought an Amstrad CPC. Later on, I bought a Yamaha CX5M Music Computer (MSX) mainly to play music on. If I succeed in porting Microsoft Extended BASIC to the C64 platform, then I think people should pay Microsoft $5 or $10 if they use it, because that’s what the penny pinching miser Jack Tramiel should have done! Read more about Commodore 64 BASIC V2 and comparing it to other computers on my blog http://www.commodore64crap.wordpress.com .

**Danny S.**
2015-04-16 at 16:01 | Reply

There were, of course, several enhanced versions of Basic sold for the Commodore 64. First there was the “Super Expander” cartridge, written by Commodore itself, and then “Simons’ Basic”, written by a British teenager (hence the COLOUR rather than COLOR command) but marketed by Commodore. And of course, Commodore’s later computers (like the C128, as well as the C16 and Plus/4) included versions of Basic with commands dedicated to the graphics and sound features found in those computers. So Commodore’s deal with Microsoft didn’t prevent them from implementing such commands at all. But the lack of such commands on the C64 undoubtedly taught a good number of people programming skills and knowledge of computer hardware that they would not have acquired if using the machine’s graphics and sound capabilities had been easier.

**PeteArnt**

2014-08-17 at 10:24 | Reply

Minor correction to the section on Ohio Scientific:
OSI did have a high-end(i.e. business) BASIC called OS-65U. It used 9 digit precision FP math instead of the usual 6 digits. Also, garbage collection worked for the most part, but would cause programs using lots of string operations to periodically freeze for several seconds before continuing.

Kindest Regards,
P.

**Danny S.**
2015-04-16 at 16:18 | Reply

One more minor correction regarding Ohio Scientific: As I mentioned elsewhere, OSI's versions of Basic did NOT use the ROR instruction, but rather used the macro expansion workaround. Oh, and the input buffer in OSI Basic was definitely on zero page.

It is hard to see how something released in 1982 could be among the "first eight" versions, and also confusing why something from 1982 is listed in the chart given here before something from 1980. Anyway, OSI did have a 9-digit version of Basic in addition to the 6-digit ROM version. The copyright date displayed by the 9-digit Basic (when loaded from their OS-65D operating system) was also 1977. Possibly it was released between the KIM Basic and Applesoft version II (that is a guess based on the dates).

**Danny S.**
2015-06-19 at 06:28 | Reply

Well, I was sort-of right and sort-of wrong in the above comment. The ROM version of OSI Basic, which is what the article refers to, DID use the ROR instruction. The disk-based versions did not use it.

**MiaM**
2015-04-29 at 13:21 | Reply

There seems to be atleast one more version of Microsoft Basic for 6502:

http://www.atarimagazines.com/computeii/issue1/page3-c.php

COMPUTE II ISSUE 1 / APRIL/MAY 1980
"...HDE added the CHAINING function to their already much enhanced version of MICROSOFT Basic. This lets you run Basic programs which are actually too large to fit into your memory. According to HDE, a disk file system is next on the list for their not-so-basic BASIC."

Pingback: Create your own Version of Microsoft BASIC for 6502 | ExtendTree

**Peter Ferrie**
2017-02-03 at 14:19 | Reply

on the Apple II, the crashing line number is 437761, rather than 350721, because the compared token has a different value.

**doctor x**
2017-06-27 at 16:24 | Reply

i seek a minimal-chip sbc with VIC-20 6502 architecture. when i plug in a cartridge to the expand
port all i want to do is flip on the power switch & this sbc program goes right to work for any
particular application designed for. i am trying to keep this all simple but not able to grasp good
BASIC INTERPRETER as used by the VIC-20 computer. the BASIC STAMP is but one example but i
have never used since i don't know what microprocessor it uses (6502, 8080, etc). can you shed
some light on this subject for me? thanks, Dr. X

**Donald Nakano**
2017-08-24 at 15:06 | Reply

Thanks for posting this! I had a KIM-1 around 1980 and designed and built my own RAM boards from scratch. When I reached 5K of RAM, I bought Tom Pittman's Tiny BASIC. When I reached 13K, I bought Microsoft's Business BASIC. That software continued to be used until I got the HDE floppy drive system for my, by then, 32KB RAM KIM-1 system. I later designed my own CPU, EPROM, and I/O board which I put into a case with the floppy system.

That little system provided my income for several years as I developed software and hardware for the Commodore VIC20, 64, and Plus 4.

Eventually, I replaced it with an XT clone so I could develop for the IBM PC.

That was a long time ago and I'm still making my living working with microprocessors.

Thanks for the blast from the past!

Don

Pingback: Programming from Commodore to Intel | Harmonia Philosophica

**Maury Markowitz**
2018-03-10 at 07:47 | Reply

I am trying to find the origins of the integer-variable type, A%. I believe this only appeared in CBM versions after the PET machines I used.

Monte Davidoff mentioned integer variables were part of BASIC-PLUS that formed the pattern for MS BASIC. But he did not work on the 6502 versions so he did not know when it was added.

Looking through var.s I see what may be a reference to this; at line L2ED8 I see it looking for #$24, and at L2EE2 I see it looking for #$25, which I assume are the $ and % symbols?

If that is the case, it would appear this version has integer support. I believe this is MS version 2.x? Do you happen to know when this support first appeared?

I have always been curious to know why they added integer variables but did not include a separate integer math unit. It would seem that one could greatly increase performance if there was a *little* extra code to see if the variables and/or literals on either side of an operator were integers and then have code for the + and -, with the rest of the math routines falling back to the existing FP code.

**Carlos**
2018-04-27 at 11:30 | Reply

> Why no Microsoft basic M68 or M69, for Motorola 6800 and 6809 listed?

**Richard Findlay**
2019-06-21 at 02:52 | Reply

> Test Post

## Leave a Comment

Name *

Email *

Website

Post Comment