Microsoft BASIC for 6502 Original Source Code [1978]

2015-01-13 by Michael Steil

This is the original 1978 source code of Microsoft BASIC for 6502 with all original comments, documentation and easter eggs:

Twitter:
@pagetable

Mastodon:
@pagetable@mastodon.social

Categories

6502 Amiga Apple
Archeology
BASIC Bildschirmtext
C64 C128

```
TITLE   BASIC M6502 8K VER 1.1 BY MICRO-SOFT
SEARCH  M6502
SALL
RADIX 10                        ;THROUGHOUT ALL BUT MATH-PAK.

$Z::                            ;STARTING POINT FOR M6502 SIMULATOR
        ORG     0               ;START OFF AT LOCATION ZERO.
SUBTTL  SWITCHES,MACROS.

REALIO=4                        ;5=STM
                                ;4=APPLE.
                                ;3=COMMODORE.
                                ;2=OSI
                                ;1=MOS TECH,KIM
                                ;0=PDP-10 SIMULATING 6502
INTPRC==1                       ;INTEGER ARRAYS.
ADDPRC==1                       ;FOR ADDITIONAL PRECISION.
LNGERR==0                       ;LONG ERROR MESSAGES.
TIME==  0                       ;CAPABILITY TO SET AND READ A CLK.
EXTIO== 0                       ;EXTERNAL I/O
```

M6502.MAC (1978-07-27, 6955 lines, 161,685 bytes)

This is currently the oldest publicly available piece of source written by Bill Gates.

Language

Like the 8080 version, the 6502 version was developed on a PDP-10, using the MACRO-10 assembler. A set of macros developed by Paul Allen allowed MACRO-10 to understand and translate 6502 assembly, albeit in a modified format to fit the syntax of macros, for example:

MOS 6502	MACRO-10
LDA #0	LDAI 0

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

`LDA (ADDR),Y`	`LDADY ADDR`

MACRO-10 did not support hex numbers, which is why most numbers are in decimal format. In the floating point code, all numbers are octal. The `RADIX` statement switches between the two. Octal can also be forced with a `^O` prefix.

Conditional translation is done using the `IFE` and `IFN` statements, which test whether the argument is zero. The following only adds the string to the binary if `REALIO` is equal to 4:

```
IFE     REALIO-4,<DT"APPLE BASIC V1.1">
```

Macros

The source defines many macros that make development easier. There are some examples:

Macro	Definition	Comment
`SYNCHK (Q)`	`LDAI <Q>` `JSR SYNCHR`	Get the next character and make sure it's `Q`, otherwise `SYNTAX ERROR`. This pattern is used a lot.
`LDWD (WD)`	`LDA WD` `LDY <WD>+1`	Most 16 bit constants are loaded into A/Y with this macro, but macros for A/X and X/Y also exist.
`LDWDI (WD)`	`LDAI <<WD>&^O377>` `LDYI <<WD>/^O400>`	This loads an immediate constant into A/Y.

PSHWD (WD)	LDA <WD>+1 PHA LDA WD PHA	This pushes a 16 bit value from memory (absolute or zero page) onto the stack.
JEQ (WD)	BNE .+5 JMP WD	A compact way to express out-of-bounds branches. Macros exist for all branches.
SKIP2	XWD ^O1000,^O054	This emits a byte value of 0x2C (BIT absolute), which skips the next instruction. (The ^O1000 part wraps the byte in a PDP-10 instruction – see below.)

Configurations

The BASIC source supports several compile-time configuration options:

Name	Comment	Description
INTPRC	INTEGER ARRAYS	
ADDPRC	FOR ADDITIONAL PRECISION	40 bit (9 digit) vs 32 bit (7 digit) float
LNGERR	LONG ERROR MESSAGES	Error message strings instead of two-character codes
TIME	CAPABILITY TO SET AND READ A CLK	TI and TI$ support

EXTIO	EXTERNAL I/O	PRINT#, INPUT#, CMD, SYS (!), OPEN and CLOSE support
DISKO	SAVE AND LOAD COMMANDS	LOAD, SAVE (and on Commodore: VERIFY) support
NULCMD	FOR THE "NULL" COMMAND	NULL support, a command to configure the number of NUL characters to print to the terminal after each line break
GETCMD		GET support
RORSW		If 1, the ROR instruction is not used
ROMSW	TELLS IF THIS IS ON ROM	The RAM version can optionally jetison the SIN, COS, TAN and ATN commands at startup
CLMWID		Column width for TAB()“
LONGI	LONG INITIALIZATION SWITCH	
STKEND		The top of stack at startup
BUFPAG		Page of the input buffer; if 0, the buffer uses *parts* of the zero page
BUFLEN	INPUT BUFFER SIZE	
LINLEN	TERMINAL LINE LENGTH	

ROMLOC	ADDRESS OF START OF PURE SEGMENT	
KIMROM		KIM-specific smaller config

Targets

The constant REALIO is used to configure what computer system to generate the binary for. It has one of the following values:

Value	Comment	Banner	Machine
0	PDP-10 SIMULATING 6502	SIMULATED BASIC FOR THE 6502 V1.1	Paul Allen's Simulator on PDP-10
1	MOS TECH,KIM	KIM BASIC V1.1	MOS KIM-1
2	OSI	OSI 6502 BASIC VERSION 1.1	OSI Model 500
3	COMMODORE	### COMMODORE BASIC ###	Commodore PET 2001
4	APPLE	APPLE BASIC V1.1	Apple II
5	STM	STM BASIC V1.1	(unreleased)

All versions except Commodore also print "COPYRIGHT 1978 MICROSOFT" in a new line.

The target defines the setting of the configuration constants, but some code is also conditionally compiled depending on a specific target.

What is interesting is that initially it was Microsoft adapting their source for the different computers, instead of giving source to the different vendors and having them adapt it. Features like file I/O and time support seem to have been specifically developed for Commodore, for example. Later, the computer companies would get the source from Microsoft and develop themselves – source code of the Apple and Commodore derivatives is available; they both contain Microsoft comments.

By the way, the numbering of these targets probably indicated in which order Microsoft signed contracts with computer manufacturers. MOS was first (for the KIM), then OSI, then Commodore/MOS again (this time for the PET), then Apple.

The PDP-10 Target

Paul Allen's additional macros for 6502 development made the MACRO-10 assembler output one 36 bit PDP-10 instruction word for every 6502 byte. When targeting a real 6502 machine, the 6502 binary could be created by simply extracting one byte from every PDP-10 word.

In the case of targeting the simulator, the code created by the assembler could just be run without modification, since every emitted PDP-10 instruction was constructed so that it would trap – the linked-in simulator would then extract the 6502 opcode from the instruction and emulate the 6502 behavior.

While this trick was mostly abstracted by the (unreleased) macro package, its workings can be seen in a few cases in the BASIC source. Here, it defines `SKIP1` and `SKIP2`. Instead of just emitting 0x24 or 0x2C, respectively, it combines it with the octal value of 01000 to make it a PDP-10 instruction that traps:

```
DEFINE  SKIP1,  <XWD ^O1000,^O044>      ;BIT ZERO PAGE TRICK.
DEFINE  SKIP2,  <XWD ^O1000,^O054>      ;BIT ABS TRICK.
```

In the initialization code, it writes a `JMP` instruction into RAM. On the simulator, it has to patch up the opcode of JMP (0x4C, decimal 76) to be the correct PDP-10 instruction:

```
        LDAI    76              ;JMP INSTRUCTION.
IFE     REALIO,<HRLI 1,^O1000>  ;MAKE AN INST.
```

With this information, we can reconstruct what the set of 6502 macros, which is not part of this source, probably looked like. Here is `LDAI` (`LDA` immediate):

```
DEFINE  LDAI    (Q),<
        XWD ^O1000,^O251        ;EMIT OPCODE
        XWD ^O1000,<Q>          ;EMIT OPERAND
>
```

You can also see native `TJSR` PDP-10 assembly instructions for character I/O:

```
IFE     REALIO,<
        TJSR    INSIM##>        ;GET A CHARACTER FROM SIMULATOR



IFE     REALIO,<
        TJSR    OUTSIM##>       ;CALL SIMULATOR OUTPUT ROUTINE
```

The DDT command, which breaks into the PDP-10's DDT debugger, only exists in this config:

```
IFE     REALIO,<
DDT:    PLA                     ;GET RID OF NEWSTT RETURN.
        PLA
        HRRZ    14,.JBDDT##
        JRST    0(14)>
```

The KIM and OSI Targets

The KIM target is meant for the MOS KIM-1 and Ohio Scientific OSI Model 500 single-board computers. These are the first ports to specific computers, and also the cleanest, i.e. except for the character I/O interface and the very simple LOAD/SAVE implementation for the KIM, there is nothing specific about these targets.

The Commodore Target

The Commodore target is meant for the Commodore PET 2001. It includes LOAD/SAVE/VERIFY (the commands jump directly to outside "KERNAL" ROM code), the I/O commands (SYS, PRINT#, OPEN etc.), the GET command and the π, ST, TI and TI$ symbols. CLEAR is renamed to CLR, "OK" is renamed to "READY.", the BEL character is not printed, and character I/O code behaves differently to account for the more featureful screen editor of the PET.

Oh, and the Commodore version of course includes the Bill Gates WAIT 6502,1 easter egg! This is the WAIT instruction:

```
; THE WAIT LOCATION,MASK1,MASK2 STATEMENT WAITS UNTIL THE CONTENTS
; OF LOCATION IS NONZERO WHEN XORED WITH MASK2
```

```
; AND THEN ANDED WITH MASK1. IF MASK2 IS NOT PRESENT, IT
; IS ASSUMED TO BE ZERO.

FNWAIT: JSR     GETNUM
        STX     ANDMSK
        LDXI    0
        JSR     CHRGOT
        BEQ     ZSTORDO
        JSR     COMBYT          ;GET MASK2.
STORDO: STX     EORMSK
        LDYI    0
WAITER: LDADY   POKER
        EOR     EORMSK
        AND     ANDMSK
        BEQ     WAITER
ZERRTS: RTS                     ;GOT A NONZERO.
```

Note how the BEQ instruction references ZSTORDO, not STORDO – execution sneaks out of this function here.

Well, on non-Commodore machines, ZSTORDO is assigned to be the same as STORDO, so everything is fine:

```
IFN     REALIO-3,<ZSTORDO=STORDO>
```

But on Commodore, we have this code hidden near the top of the floating point math package – close enough so the BEQ can reach it, but inside code that is least likely to get touched:

```
IFE     REALIO-3,<
ZSTORD:!        LDA     POKER
        CMPI    146
```

```
        BNE     STORDO
        LDA     POKER+1
        SBCI    31
        BNE     STORDO
        STA     POKER
        TAY
        LDAI    200
        STA     POKER+1
MRCHKR: LDXI    12
IF1,<
MRCHR:  LDA     60000,X,>
IF2,<
MRCHR:  LDA     SINCON+36,X,>
        ANDI    77
        STADY   POKER
        INY
        BNE     PKINC
        INC     POKER+1
PKINC:  DEX
        BNE     MRCHR
        DEC     ANDMSK
        BNE     MRCHKR
        RTS
IF2,<PURGE ZSTORD>>
```

(`IF1` and `IF2` are true on the first and the second assembler pass, respectively, so the conditional there is to hint to the assembler in the first pass that `SINCON+36` is not a zero page address. Also note that all numbers here are octal, since this code is in the floating point package.)

First of all, the final line here removes `ZSTORD` from the list of symbols after the second pass, so that Commodore would not notice it in a printout of all symbols – very smart!

As has been discussed before, this code writes the string “MICROSOFT!” into the PET’s screen RAM if the argument to WAIT is “6502”. The encoded string is hidden as two extra

40 bit floating point numbers appended to the coefficients used by the SIN function:

```
IFN     ADDPRC,<
SINCON: 5               ;DEGREE-1.
        204     ; -14.381383816
        346
        032
        055
        033
        206     ; 42.07777095
        050
        007
        373
        370
        207     ; -76.704133676
        231
        150
        211
        001
        207     ; 81.605223690
        043
        065
        337
        341
        206     ; -41.34170209
        245
        135
        347
        050
        203     ; 6.2831853070
        111
        017
        332
        242
        241     ; 7.2362932E7
        124
        106
        217
        23
```

```
217     ; 73276.2515
122
103
211
315>
```

These last ten bytes, nicely disguised as octal values of floating point constants, spell out "MICROSOFT!" backwards after clearing the upper two bits. What's interesting is that the floating point values next to them are actually incorrect: They should be 7.12278788E9 and 26913.7691 instead.

Also note that these constants are not conditionally assembled! All versions built since the Commodore easter egg was introduced also contained these 10 bytes – including BASIC for the Motorola 6800!

The Apple Target

The Apple target is meant for the Apple II, and contains no customizations other than some changes around I/O handling (which calls into the monitor ROM). Note that this is not yet the "AppleSoft" version of BASIC, which was a more customized version modified by Apple later.

The STM Target

"STM" most likely stands for "Semi-Tech Microelectronics" – a company that never shipped a 6502-based computer. Their first machine was the "Pied Piper", a Z80-based system, and they later made a PC clone. It seems they had a 6502-based computer in development that never shipped – or at least they were considering making one, and Microsoft added the target; this target doesn't actually change any of the defaults.

Organization of the Source

The source uses the PAGE and SUBTTL keywords for organization. Here are the headings:

```
SUBTTL  SWITCHES,MACROS.
SUBTTL  INTRODUCTION AND COMPILATION PARAMETERS.
SUBTTL  SOME EXPLANATION.
SUBTTL  PAGE ZERO.
SUBTTL  RAM CODE.
SUBTTL  DISPATCH TABLES, RESERVED WORDS, AND ERROR TEXTS.
SUBTTL  GENERAL STORAGE MANAGEMENT ROUTINES.
SUBTTL  ERROR HANDLER, READY, TERMINAL INPUT, COMPACTIFY, NEW, REINIT.
SUBTTL  THE "LIST" COMMAND.
SUBTTL  THE "FOR" STATEMENT.
SUBTTL  NEW STATEMENT FETCHER.
SUBTTL  RESTORE,STOP,END,CONTINUE,NULL,CLEAR.
SUBTTL  LOAD AND SAVE SUBROUTINES.
SUBTTL  RUN,GOTO,GOSUB,RETURN.
SUBTTL  "IF ... THEN" CODE.
SUBTTL  "ON ... GO TO ..." CODE.
SUBTTL  LINGET -- READ A LINE NUMBER INTO LINNUM
SUBTTL  "LET" CODE.
SUBTTL  PRINT CODE.
SUBTTL  INPUT AND READ CODE.
SUBTTL  THE NEXT CODE IS THE "NEXT CODE"
SUBTTL  DIMENSION AND VARIABLE SEARCHING.
SUBTTL  MULTIPLE DIMENSION CODE.
SUBTTL  INTEGER ARITHMETIC ROUTINES.
SUBTTL  FRE FUNCTION AND INTEGER TO FLOATING ROUTINES.
SUBTTL  SIMPLE-USER-DEFINED-FUNCTION CODE.
SUBTTL  STRING FUNCTIONS.
SUBTTL  PEEK, POKE, AND FNWAIT.
SUBTTL  FLOATING POINT ADDITION AND SUBTRACTION.
SUBTTL  NATURAL LOG FUNCTION.
SUBTTL  FLOATING MULTIPLICATION AND DIVISION.
SUBTTL  FLOATING POINT MOVEMENT ROUTINES.
SUBTTL  SIGN, SGN, FLOAT, NEG, ABS.
SUBTTL  COMPARE TWO NUMBERS.
```

```
SUBTTL  GREATEST INTEGER FUNCTION.
SUBTTL  FLOATING POINT INPUT ROUTINE.
SUBTTL  FLOATING POINT OUTPUT ROUTINE.
SUBTTL  EXPONENTIATION AND SQUARE ROOT FUNCTION.
SUBTTL  EXPONENTIATION FUNCTION.
SUBTTL  POLYNOMIAL EVALUATOR AND THE RANDOM NUMBER GENERATOR.
SUBTTL  SINE, COSINE AND TANGENT FUNCTIONS.
SUBTTL  ARCTANGENT FUNCTION.
SUBTTL  SYSTEM INITIALIZATION CODE.
```

Paul Allen vs. Bill Gates

The source of the 8080 version states:

```
PAUL ALLEN WROTE THE NON-RUNTIME STUFF.
BILL GATES WROTE THE RUNTIME STUFF.
MONTE DAVIDOFF WROTE THE MATH PACKAGE.
```

People have since wondered what runtime vs. non-runtime meant, especially since Paul Allen's recent debate on whether the company's ownership was faily split.

The BASIC for 6502 source sheds some light on this:

```
NON-RUNTIME STUFF
        THE CODE TO INPUT A LINE, CRUNCH IT, GIVE ERRORS,
        FIND A SPECIFIC LINE IN THE PROGRAM,
        PERFORM A "NEW", "CLEAR", AND "LIST" ARE
        ALL IN THIS AREA. [...]
```

So by “runtime” they just literally mean “at run time”: all code that is active when the program runs, as opposed to non-runtime, which is all code that assists editing the program.

By this understanding, we can assume this:

- Paul Allen wrote the macro package for the MACRO-10 assembler, the 6502 simulator, the tokenizer, the detokenizer, as well as finding, inserting and deleting BASIC lines.
- Bill Gates implemented all BASIC statements, functions, operators, expression evaluation, stack management for FOR and GOSUB, the memory manager, as well as the array and string library.
- Monte Davidoff wrote the floating point math package.

Version and Date

The last entry in the change log has a date of 1978-07-27. Both the comment in the first line of the file and the message printed at startup call it version 1.1.

What does this say about the version of the source? Is it the last version? Let’s look at the last bug fix and compare which BASIC binaries contain this fix, and let’s see whether there are fixes in BASIC binaries that are not in the source.

I have previously compared binaries of derivatives of BASIC for 6502 and compiled the information at github.com/mist64/msbasic. The last entry in the log of this source is about a bug that failed to correctly invalidate a pointer in the `RETURN` statement. According to my analysis of BASIC 6502 versions, this is fixed in the BASIC binaries for AIM-65, SYM-1,

Commodore v2, KBD BASIC and MicroTAN, i.e. on everything my previous analysis calls `CONFIG_2A` and higher.

The same analysis also came to the conclusion that there were two successors, `CONFIG_2B` and `CONFIG_2C`. At least the two `CONFIG_2B` fixes exist in two BASIC binaries: KBD BASIC and MicroTAN, but they don't exist in this source. It's very unlikely that both these bugs (and only these!) got fixed by the two computer manufacturers independently, so it's safe to assume that this source is not the final version – but pretty close to it!

Interesting Finds

- This code is comparing a keyboard input character to the BEL code. Bob Albrecht is a computer educator that "was instrumental in helping bring about a public-domain version of Basic (called Tiny Basic) for early microcomputers.".

```
CMPI    7               ;IS IT BOB ALBRECHT RINGING THE BELL
                        ;FOR SCHOOL KIDS?
```

- External documentation usually calls the conversion of ASCII BASIC text into the compressed format "tokenizing". The source calls this "crunching".
- Microsoft is still spelled "Micro-Soft".
- Apparently the multiplication function could use some performance improvements:

```
BNE     MLTPL2          ;SLOW AS A TURTLE !
```

- The `NEW` command is actually called `SCRATCH` in labels and comments – maybe other BASIC dialects called it that, and they decided to rename it to `NEW` later?

- The math package documentation says:

```
MATH PACKAGE
        THE MATH PACKAGE CONTAINS FLOATING INPUT (FIN),
        FLOATING OUTPUT (FOUT), FLOATING COMPARE (FCOMP)
        ... AND ALL THE NUMERIC OPERATORS AND FUNCTIONS.
        THE FORMATS, CONVENTIONS AND ENTRY POINTS ARE ALL
        DESCRIBED IN THE MATH PACKAGE ITSELF.
```

Commodore's derived source changes this to:

```
; MATH PACKAGE
;       THE MATH PACKAGE CONTAINS FLOATING INPUT FIN, OUTPUT
;       FOUT, COMPARE FCOMP...AND ALL THE NUMERIC OPERATORS
;       AND FUNCTIONS.  THE FORMATS, CONVENTIONS AND ENTRY
;       POINTS ARE ALL DESCRIBED IN THE MATH PACKAGE ITSELF.
;       (HA,HA...)
```

- CHRGET is a central piece of BASIC for 6502. Here it is in its entirety:

```
; THIS CODE GETS CHANGED THROUGHOUT EXECUTION.
; IT IS MADE TO BE FAST THIS WAY.
; ALSO, [X] AND [Y] ARE NOT DISTURBED
;
; "CHRGET" USING [TXTPTR] AS THE CURRENT TEXT PNTR
; FETCHES A NEW CHARACTER INTO ACCA AFTER INCREMENTING [TXTPTR]
; AND SETS CONDITION CODES ACCORDING TO WHAT'S IN ACCA.
;       NOT C=  NUMERIC   ("0" THRU "9")
;       Z=      ":" OR END-OF-LINE (A NULL)
;
; [ACCA] = NEW CHAR.
; [TXTPTR]=[TXTPTR]+1
;
; THE FOLLOWING EXISTS IN ROM IF ROM EXISTS AND IS LOADED
; DOWN HERE BY INIT. OTHERWISE IT IS JUST LOADED INTO THIS
```

```
; RAM LIKE ALL THE REST OF RAM IS LOADED.
;
CHRGET: INC     CHRGET+7        ;INCREMENT THE WHOLE TXTPTR.
        BNE     CHRGOT
        INC     CHRGET+8
CHRGOT: LDA     60000           ;A LOAD WITH AN EXT ADDR.
TXTPTR= CHRGOT+1
        CMPI    " "             ;SKIP SPACES.
        BEQ     CHRGET
QNUM:   CMPI    ":"             ;IS IT A ":"?
        BCS     CHRRTS          ;IT IS .GE. ":"
        SEC
        SBCI    "0"             ;ALL CHARS .GT. "9" HAVE RET'D SO
        SEC
        SBCI    256-"0"         ;SEE IF NUMERIC.
                                ;TURN CARRY ON IF NUMERIC.
                                ;ALSO, SETZ IF NULL.
CHRRTS: RTS                     ;RETURN TO CALLER.
```

Did you ever wonder why all versions have $EA60 encoded into the `LDA` instruction that later gets overwritten? Because it's 60000 decimal. That's why! The source actually uses 60000 as a placeholder for 16 bit values in several places.

- The handling of π, `ST`, `TI` and `TI$` (all Commodore-specific) looks wonky: Instead of making them tokens, they are special cased in several places. I always assumed it was Commodore adding this without understanding (or wanting to disrupt) the existing code, but it was Microsoft adding these features. Maybe they were added by someone other than the original developers?

Origin of the File

The source was posted on the Korean-language blog 6502.tistory.com without further comment, in a marked-up format:

```
================================================================================
FILE: "david mac g5 b:m6502.asm"
================================================================================

000001  TITLE   BASIC M6502 8K VER 1.1 BY MICRO-SOFT
[...]
006955          END     $Z+START

End of File -- Lines: 6955 Characters: 154740

SUMMARY:

  Total number of files : 1
  Total file lines      : 6955
  Total file characters : 154740
```

This formatting was created by an unpublished tool by David T. Craig, who published a lot of Apple-related soure code (Apple II, Apple III, Lisa) in this format in as early as 1993, first anonymously, later with his name).

The filename "`david mac g5 b:m6502.asm`" (disk name "david mac g5 b", file name "m6502.asm", since it was a classic Mac OS tool) confirms David Craig's involvement, and it means the line numbers were added no earlier than 2003.

Given all this, it is safe to assume the file with the Microsoft BASIC for 6502 source originated at Apple, and was given to David Craig together with the other source he published.

The version I posted is a reconstruction of the original file, with the header, the footer and the line numbers removed, and the spaces converted back into tabs. I chose the name

"M6502.MAC" to be consistent with the MACRO-10 file extension used by the Microsoft BASIC for 8080 sources.

6502, Archeology, BASIC, PET

‹ Using the OS X 10.10 Hypervisor Framework: A Simple DOS Emulator
› Fully Commented Commodore 64 BASIC ROM Disassembly – based on Microsoft's Source

30 thoughts on "Microsoft BASIC for 6502 Original Source Code [1978]"

Pingback: Microsoft BASIC for 6502 Original Source Code [1978] | The WordPress C(h)ronicle

Pingback: Código fuente original de Microsoft BASIC del año 1978 [ENG]

atomz
2015-01-14 at 01:14 | Reply

> La verdad es que el código está de lujo. Excelentemente comentado y bastante claro y entendible (teniendo en cuenta la complejidad del lenguaje ensamblador). Hacer esto en los años 70 tiene muchísimo mérito principalmente debido a la falta de información y por ser un pionero en una tarea tan compleja. Los que hemos

programado en ese lenguaje sabemos la dificultad de realizar todo el código sin errores, ya que cualquier mínimo fallo desbarata todo el programa sin saber muy bien por qué. No tiene nada que ver con los lenguajes de alto nivel, como Java o C++.

Trond Nyløkken
2018-04-02 at 00:55 | Reply

I had a rockwell 6502 machine with basic emulator. I had also build a flopydrive with a controløer for it, and I make memoryswitch board for ekstra 32Mb memmory. That was very cool my firsr computer.

MagerValp
2015-01-14 at 02:43 | Reply

Commodore famously licensed Microsoft BASIC on a “pay once, no royalties” basis, and independently kept developing it for its 8-bit line of computers. If you’re interested in its history David Viner has posted the source for version 4.75 from 1982 with several enhancements:

http://www.davidviner.com/cbm9.html

peterpies
2017-10-20 at 11:02 | Reply

> Jack Tramiel the CEO of Commodore done the deal.

Pingback: Microsoft BASIC for 6502源代码泄露 | IT新闻 | html5tricks

Thomas
2015-01-14 at 23:49 | Reply

> My RSS reader stops rendering in the second macro definition. Maybe the WD in angle brackets is not being escaped correctly?

Pingback: Au Courant || WNBTv

Julian Skidmore
2015-01-17 at 10:40 | Reply

You wrote: “The NEW command is actually called SCRATCH in labels and comments – maybe other BASIC dialects called it that, and they decided to rename it to NEW later?”

Indeed, NorthStar Horizon Multiuser Basic called it scratch (or rather scr). I didn’t know that was an early Microsoftism, I just thought it was a quirk of NorthStar Horizon Multiuser Basic.

Gennaro Capuozzo
2015-02-17 at 08:05 | Reply

Several years later, the SCRATCH command was present again in CBM BASIC.
It had nothing in common with the NEW command: it was a file-oriented DELete (rm) operation.

Lasse Karkkainen
2015-08-11 at 05:31 | Reply

Actually, The Scratch in later CBM Basics was directly taken from CBM-DOS. At least The 1541 (and later) disk drives used Scratch command for removing files. I am not sure if the PET drives used

same Disk Operating System? (and so Microsoft decided to change the name due possible confusion with Disk command?)

Pingback: Visto nel Web – 166 | Ok, panico

Pingback: Snippets: Io.js, FreeBSD in the Cloud and 6502 Basic | codescaling

Pingback: Les liens de la semaine – Édition #115 | French Coding

Ron
2015-01-20 at 01:27 | Reply

Certain characters in the headline of this entry causes most RSS readers on Mac to fail. (NSXMLParserErrorDomain 9)

Pingback: MSDN Blogs

Pingback: Compare in rete il codice sorgente del Basic Microsoft per 6502 (scritto da Bill Gates) | Archeologia Informatica

Pingback: Odd Lots – Jeff Duntemann's Contrapositive Diary

Pingback: Algunos programas que han hecho su código fuente público

Pingback: Algunos programas que han hecho su código fuente público | Entuespacio.com

Thiago
2015-03-13 at 23:36 | Reply

> Perhaps STM means STMicroelectronics? Did they make some kind of 6502 clone?

Pingback: Intervista a Chuck Peddle, il papà del 6502! | Archeologia Informatica

falken
2015-03-22 at 06:07 | Reply

> thanks, just cool ;-)

Pingback: نرم‌افزار قدیمی که امروزه آزاد هستند | سلام دنیا | رسانه تخصصی نرم‌افزارهای آزاد / ۱۸ متن‌باز

Pingback: Computer History: Links And Resources (19) | Angel "Java" Lopez on Blog

Pingback: Примеры классического кода, ставшего Open Source - itfm.pro

Pingback: Make Munich 2016 – das wars schon wieder | Steckschwein

Gordon
2017-03-05 at 01:28 | Reply

Hi Michael,
I stumbled across your blog trying to find out the difference between the NTSC/PAL Futurama DVDs.

I've since taken a look at some of your other posts. It's all really interesting stuff. I watched the talk on 6502 which was fascinating and hilarious. I'll definitely be following this blog.

Also, you might wanna check out some of the comments. You seem to be getting fucked hard by spam.

Check out the most recent comment by "organic living". What the hell? I don't even get why someone would make a program that makes such bizarre comments. The use of non-ASCII characters is the icing on the WTF-cake for me — e.g. "diɗ" instead of "did".

DAVID CRAIG
2019-07-08 at 08:53 | Reply

Hi Michael,

Thanks for the informative article about the origin of AppleSoft BASIC.

“it is safe to assume the file with the Microsoft BASIC for 6502 source originated at Apple, and was given to David Craig together with the other source he published”

If my recollection is correct, I recall finding long ago file “m6502.asm” on the internet. No one provided me with the file.

I believe you are correct about Microsoft creating the original 6502 version from their MITS ALTAIR BASIC whch Apple licensed and later maintained. I have the actual Apple source for what seems to be the last version of AppleSoft BASIC. If you want a copy, I can email to you. A non-Apple person provided this to me and believe he/she received this from Apple under a business license.

I also have the source for the Apple /// computer BASIC called BUSINESS BASIC. This itself seems to have been based on Apple’s interim BASIC 3 interpreter which was itself based on the AppleSoft BASIC source. BASIC 3 was never released as far as I recall. From what I understand, Apple wanted to create their own floating point BASIC without a Microsoft connection. The target platform was to be the Apple II line, but instead this source with many improvements (e.g. invokable modules) was used for the more powerful Apple /// instead.

Thanks for your well written article and software history efforts.

Also, I have no idea about the Korean blog you mention and this source. All I can say is once something is placed on the internet that something can easily spread all over the place.

Best in 2019.

~ David Craig | New Mexico
~ dtc.bayern@gmail.com
~ July 8, 2019

Leave a Comment

Name *

Email *

Website

Post Comment