Benchmarking CF83 Forth

by M. David Johnson

2019/04/26
Abstract

CF83 Forth was compared, for speed of operation, to Basic, Assembly Language, and four other Forths for the Radio Shack Color Computer.

With the exception of printing, CF83 Forth was found to be significantly faster than Basic, significantly slower than Assembly Language, and reasonably comparable to the four other Forths.

In a printing-intensive benchmark, CF83 Forth was found to be much slower than any of its six competitors in this investigation.

-----

Special Thanks to Stephen M. Periera for his guidance through the labyrinths of Talbot ColorForth.
Table of Contents

Abstract ................................................................. 2
Introduction ......................................................... 4
General Methodology ............................................... 6
Results ................................................................. 7
Conclusions and Future Work ................................. 9

-----

Appendix A: Equipment ........................................ 11
Appendix B: Languages ........................................ 12
Appendix C: Calibration and the CoCo Timer .......... 22
Appendix D: Brute Force Primes Benchmark ........ 24
Appendix E: Sieve of Eratosthenes Benchmark .......... 41
Appendix F: Add Loops Benchmark ...................... 60
Appendix G: Print Loops Benchmark .................. 72
Appendix H: New BDS Software License .............. 84
Appendix I: References .......................................... 85
Introduction

Speed is not the only factor which is important for selecting a programming language. Other factors include:

- Availability
- Cost
- Ease of Use
- Learning Curve
- Features Collection
- Suitability to a Specific Programming Task
- User/Language Interface
- Language/Machine Interface
- Extensibility
- etc.

However, speed is certainly one of the factors that go into making the selection, and it is therefore an important factor to measure.

In this paper, we investigate and compare the relative speeds of the following languages:

- Basic
- Assembly Language
- CF83 Forth
- Armadillo ColorForth 2.0
- pd10 Superforth
- Talbot ColorForth 1.1
- eForth

all for the Radio Shack Color Computer.

We make the comparisons on the basis of four Benchmarks:

- Brute Force Primes Benchmark
- Sieve of Eratosthenes Primes Benchmark
- Add Loops Benchmark
- Print Loops Benchmark

When I first completed CF83 Forth and its extensions back in 1991, The Rainbow Magazine had just died. Figuring that the CoCo would die with it (little did I know), I put CF83 in a drawer and mostly forgot about it. At the time, I never tested it in comparison to other languages available for the CoCo.
Now that I've discovered that the CoCo still lives, and have put CF83 back “out there”, it seemed like an appropriate time to explore how CF83 stacks up against some of the other players in the CoCo languages game.

I hope the current investigation will help you in assessing whether or not CF83 would be your choice for developing any of your own applications.
General Methodology

All benchmark tests were run on the Vcc Emulator as described in Appendix A. Before beginning any of the benchmark runs proper, the stopwatch was used to calibrate the CoCo Timer, as described in Appendix C, i.e. to determine how many ticks per second were actually occurring in the Vcc Emulator.

Each of the languages, even the Forths, are different from each other and have their own idiosyncrasies. For each of the four benchmarks, the code had to be independently developed for each language. This means that the code is not identical in any two given cases.

However, the sections of the code which were individualized in this manner were kept outside of the timed sequences. In all cases, the specialized setups were accomplished prior to the start of the timing, and the reporting mechanisms were delayed until after the end of the timing.

In each benchmark, with the exception of eForth, the methodology followed was:

1. Initial setup
   - Clear the CoCo Timer to Zero
   - Perform the benchmark
   - Get the CoCo Timer Number of ticks expired
   - Report the results.

Because eForth does not implement nor provide any access to the CoCo Timer, its methodology was similar, although not identical:

1. Initial setup
   - Sound the bell → User starts the stopwatch
   - Perform the benchmark
   - Sound the bell → User stops the stopwatch
   - Report the results.

In all cases, for each benchmark for each language (thus $4 \times 7 = 28$ cases in all), each given case was run ten times (thus there were a total of 280 runs) to smooth out any anomalies encountered.

(Calculator.net) was then used to obtain the mean and standard deviation for each case. Finally, except for the eForth runs, which were already reporting in terms of seconds of elapsed time, the results in number of ticks were converted to number of seconds elapsed.

Both the Brute Force Primes and the Sieve of Eratosthenes Primes Benchmarks’ results were verified for correct selection of primes against Professor Chris Caldwell’s list, “The First 10,000 Primes”, located at The University of Tennessee at Martin:

https://primes.utm.edu/lists/small/10000.txt
Results

In each case, the Brute Force Primes Benchmark calculated all the primes up to 290 by the simple mechanism of dividing each number N (from N = 3 to 290) by each lesser number N1 (from N1 = 2 to N1 = N-1). If the value of any of the divisions of a given N by a given N1 was a whole number (i.e. if the division was exact) then that N is not prime. See Appendix D.

<table>
<thead>
<tr>
<th>Language</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Language</td>
<td>5.80</td>
</tr>
<tr>
<td>eForth</td>
<td>50.35</td>
</tr>
<tr>
<td>pd10 SuperForth</td>
<td>98.65</td>
</tr>
<tr>
<td>CF83 Forth</td>
<td>113.85</td>
</tr>
<tr>
<td>Talbot ColorForth 1.1</td>
<td>130.18</td>
</tr>
<tr>
<td>Armadillo ColorForth 2.0</td>
<td>161.18</td>
</tr>
<tr>
<td>Basic</td>
<td>1056.34</td>
</tr>
</tbody>
</table>

In each case, the Sieve of Eratosthenes Primes Benchmark calculated all the primes up to 4,364 by the well-known Sieve of Eratosthenes. See Appendix E.

<table>
<thead>
<tr>
<th>Language</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Language</td>
<td>0.77</td>
</tr>
<tr>
<td>eForth</td>
<td>2.87</td>
</tr>
<tr>
<td>Talbot ColorForth 1.1</td>
<td>3.28</td>
</tr>
<tr>
<td>Armadillo ColorForth 2.0</td>
<td>3.97</td>
</tr>
<tr>
<td>pd10 SuperForth</td>
<td>4.50</td>
</tr>
<tr>
<td>CF83 Forth</td>
<td>4.70</td>
</tr>
<tr>
<td>Basic</td>
<td>126.34</td>
</tr>
</tbody>
</table>

In each case, the Add Loops Benchmark added 5 to 7 and stored the result in the variable AVAR 165,000 times. See Appendix F.

<table>
<thead>
<tr>
<th>Language</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Language</td>
<td>5.45</td>
</tr>
<tr>
<td>eForth</td>
<td>34.62</td>
</tr>
<tr>
<td>Talbot ColorForth 1.1</td>
<td>47.52</td>
</tr>
<tr>
<td>Armadillo ColorForth 2.0</td>
<td>42.66</td>
</tr>
<tr>
<td>CF83 Forth</td>
<td>62.61</td>
</tr>
<tr>
<td>pd10 SuperForth</td>
<td>69.62</td>
</tr>
<tr>
<td>Basic</td>
<td>1088.46</td>
</tr>
</tbody>
</table>
In each case, the Print Loops Benchmark printed the string:

“PRINTING LOOPS BENCHMARK ”

2,000 TIMES. See Appendix G.

<table>
<thead>
<tr>
<th>Language</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Language</td>
<td>16.31 s</td>
</tr>
<tr>
<td>pd10 SuperForth</td>
<td>25.21 s</td>
</tr>
<tr>
<td>Basic</td>
<td>34.66 s</td>
</tr>
<tr>
<td>Armadillo ColorForth 2.0</td>
<td>42.88 s</td>
</tr>
<tr>
<td>Talbot ColorForth 1.1</td>
<td>60.22 s</td>
</tr>
<tr>
<td>eForth</td>
<td>168.23 s</td>
</tr>
<tr>
<td>CF83 Forth</td>
<td>1006.55 s</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

In all of the Benchmarks, except for the Print Loops Benchmark, all five Forths were significantly faster than Basic, significantly slower than Assembly Language, and reasonably comparable to each other, except that eForth was consistently faster than the other Forths in all three of those benchmarks.

In the Print Loops Benchmark, all of the Forths except pd10 SuperForth were slower than Basic, with CF83 even being six times slower than its nearest competitor, eForth.

The slowness of eForth and CF83 Forth can be attributed to their using their own (PMODE 4) graphics screens for text, and thus having to draw their text characters rather than using the CoCo’s native text screen.

CF83 Forth should thus probably not be considered for “printing-intensive” applications.

CF83’s dismal printing performance could probably be significantly improved (when necessary for a given application) by introducing new, alternate, printing words loosely based on Extended Color Basic’s PUT mechanism (cf. Zydhek, p. B33, at memory location $9758).

Some preliminary tests have already indicated that a significant improvement in speed can be reasonably expected from such.

Alternatively, CF83 could be internally revised to use such a PUT mechanism universally. That would, however, be a seriously more extensive project.

Finally, in addition to such a change, for even more speed (with everything – not just printing) CF83 could also be completely rewritten to utilize a subroutine-threaded interpreter model instead of its current indirect threaded interpreter model. The time overhead of jumping back to the inner interpreter at the end of every colon word definition could thus be completely avoided.

However, such an increase in speed comes with a price: greater system memory use and, therefore, less user-dictionary space.

(Warren, p.76) tells us that the MC6809 LBSR ($17) instruction requires 3 bytes, and (p. 100) the RTS ($3B) instruction requires 1 byte.

The current indirect-threaded interpreter code layout for a CF83 colon word appears as follows:

Name Field = length varies
Link Fields = 4 bytes
Code Field = $0007 for colon definitions = 2 bytes.
Word 1 Code Field Address = 2 bytes
Word 2 Code Field Address = 2 bytes
Word N Code Field Address = 2 bytes  
Semi = $000D = 2 bytes  

So that the total length of the colon word, without the Name Field or Link Fields, is:

\[2 + 2 + 2N \text{ bytes}\]

But, for a subroutine-threaded Forth, the code layout would be:

Name Field = length varies  
Link Fields = 4 bytes  
Code Field = Not used = 0 bytes.  
LBSR Word 1 = 3 bytes  
LBSR Word 2 = 3 bytes  
LBSR Word N = 3 bytes  
RTS = 1 byte  

So that the total length of the subroutine-threaded colon word, without the Name Field or Link Fields, is:

\[0 + 1 + 3N \text{ bytes}\]

Setting \[2 + 2 + 2N = 0 + 1 + 3N \Rightarrow 4 + 2N = 1 + 3N \Rightarrow N = 3\]

Which means that any colon word definition comprising more than three words will take up more memory space in a subroutine-threaded environment than it currently does in the indirect-threaded environment.

And, we can also note that as the number of words in a colon definition increases, the definition length of the subroutine-threaded definition will approach 1.5 times the definition length of the indirect-threaded definition.

With only 64K to work with, this might easily be deemed prohibitive.
Appendix A -- Equipment

Our test suite consists simply of the Vcc Color Computer Emulator and a stopwatch.

Vcc 2.0.1 is running under Windows 7 Professional (64-bit) SP1 on a Hewlett-Packard p6774y with an AMD Phenom™ II X4 840T 2.90 GHz processor and 16.0 GB of RAM.

The stopwatch is a Cronus Survivor Series, Model C601-11.
Appendix B -- Languages

Seven different languages are compared in this investigation:

- Basic
- Assembly Language
- CF83 Forth
- Armadillo ColorForth 2.0
- pd10 SuperForth
- Talbot ColorForth 1.1
- eForth

**Basic**

“Basic” here simply means Radio Shack’s standard Disk Extended Color Basic 2.1.

**Assembly Language**

This is standard 6809 Assembly Language as embodied in Radio Shack Disk EDTASM+ 01.00.00 1983, running under TRSDOS 01.07.00 11/82.

**CF83 Forth**

This is the original CF83 Forth, Copyright 1991 by M. David Johnson, BDS Software, Glenview, IL, as contained on the base CF83-0 disk. It was used without any extensions, except that the CF83-3 Block Editor disk was used for development of the program blocks. Only the CF83-0 disk software and the program blocks themselves were used in the test runs of this investigation.

The CF83 Forth system is available (for free) directly from BDS Software at:

http://www.bds-soft.com/coco.html

It’s also available on the (CoCo Archive), where it can be downloaded from:

http://www.colorcomputerarchive.com/coco/Disks/Programming/ under the title:

CF83 Forth (BDS Software).zip

CF83 Forth uses its own (PMODE 4) text screen, which is 64 characters wide by 24 characters high.
Note that, when switching focus (e.g. when removing one diskette and inserting another), you may have to hit your next selected key twice in order to get the intended result.

Also note that, when removing a diskette, it’s usually a good idea to execute “flush” first.

**Armadillo ColorForth 2.0**

Armadillo’s ColorForth 2.0 is available on the (CoCo Archive). It was downloaded from:

[http://www.colorcomputerarchive.com/coco/Disks/Programming/](http://www.colorcomputerarchive.com/coco/Disks/Programming/) under the title:

Color Forth 2.0 (Armadillo Int'l Software).zip

Armadillo ColorForth uses the standard CoCo text screen; 32 characters wide by 16 lines high.

The zip file expands to a single Clrforth.dsk file. That disk’s directory shows two files:

```
SCREENS  FTH  3 A 60
COLORFTH BIN  2 B 4
```

However, `SCREENS.FTH` appears to be an invalid directory entry. Although one might be tempted to guess that this file would provide the screens code (and perhaps some instructions) for Armadillo’s ColorForth, nothing labeled such seems to actually exist on the disk. In the areas where such a file might reside, all the bytes are uniformly $FF$.

As of this writing, I have been unable to discover this file or any other screens listing or instruction manual for Armadillo ColorForth anywhere else.

The `COLORFTH.BIN` file, however, is valid and `LOADM` and `EXEC` function as expected with it. This results in “COLORFORTH VERSION 2.0 (C) 1983 ARMADILLO INT’L SOFTWARE” being displayed at the top of the standard CoCo 32x16 screen.

I could discover no block editor mechanism for Armadillo ColorForth. However, blocks prepared with the CF83 Block Editor work just fine in Armadillo.

When the blocks load, they result in some messages which are not clear because of the lack of a manual. For example, from the Armadillo Benchmarks Disk, 1 LOAD produces the message “LIMIT MSG #4  OK”. I suspect it may indicate that LIMIT is a duplication of an existing word, but I can’t be sure. The blocks work properly anyway.

If it gives the message “? MSG #0”, however, that clearly means that the word preceding the message is not recognized by the system, i.e. it has not been defined.

Armadillo is a fig-FORTH. As such, variables must be defined with a leading zero, i.e.:
0 VARIABLE AVAR

rather than the Forth-83 style:

VARIABLE AVAR

When preparing blocks with the CF83 Block Editor, it’s important to remember that Armadillo ColorForth likes ALL CAPS.

Note that “J” (the Forth-83 outer loop counter) is not required by fig-FORTH and is not included in Armadillo ColorForth. Use something like 0 VARIABLE TEMP to store an outer loop counter as needed.

U. doesn’t work in Armadillo. For these test runs I just used . instead.

0> doesn’t work. I just used 0 > instead.

An interesting situation occurred with 1– which doesn’t work in Armadillo ColorForth. In these tests, 1 – didn’t work properly either. I also tried using 65535 + instead, but that wouldn’t go either. I finally predefined:

: 1– 1 – ;

and that worked properly.

NOT doesn’t work. Predefine : NOT IF 0 ELSE 65535 THEN ; instead.

Note also that Armadillo provides THEN as a clone of theENDIF which is fig_FORTH standard.

And finally, note that : TEST 10 0 DO I . LOOP ; works as expected in Armadillo, sequencing through loops 0 through 9 and stopping before 10.

pd10 SuperForth

d10 SuperForth is available on the (CoCo Archive). It was downloaded from:

http://www.colorcomputerarchive.com/coco/Disks/Programming/ under the title:

Forth.zip

pd10 SuperForth uses the standard CoCo text screen; 32 characters wide by 16 lines high.

The zip file expands to a single FORTH.dsk file. That disk’s directory shows thirteen files:

MENU BAS 0 B 1
WE.BAS gives an OS Error when you try to run it.

MENU.BAS purports to offer you the opportunity to read either the (1) FORTH MANUAL or the (2) FORTH DOCUMENTATION.

However, if you choose the FORTH MANUAL, the system presents not exactly gibberish, but the absence of any space between words and the presence of extraneous characters makes the resulting text essentially indecipherable.

If you choose the FORTH DOCUMENTATION, the system appears to present a sector-by-sector list of a portion of the disk’s contents, beginning at Track 9, Sector 10, apparently the code in the EDIT.DAT file. While this may be of use in tracing some of the code, it doesn’t appear to be the intended documentation.

FORTHMAN.UL1, .UL2, and .UL3 would appear (from the “2 B 7” and “2 B 1” directory line entries) to be machine language files, but a LOADM and EXEC just dumps you back to the Disk ECB opening prompt and leaves the system in a (not immediately visible) corrupt state.

FRTHDOC1.TXT, FRTHDOC2.TXT, FRTHDOC3.TXT, and FRTHDOC4.TXT are valid text files rather clearly intended to make up the Instruction Manual for pd10 SuperForth. However, FRTHDOC4.TXT appears to just be a second copy of FRTHDOC1.TXT, instead of what I suspect may have originally been a fourth part of the manual.

Neither FORTH.BIN nor 32KFORTH.BIN work in Vcc. But NEWFORTH.BIN does work.

pd10 SuperForth is a fig-FORTH. As such, variables must be defined with a leading zero, i.e.:

```
0 VARIABLE AVAR
```

rather than the Forth-83 style:

```
VARIABLE AVAR
```
pd10 SuperForth does not use 1024 byte blocks (a.k.a. screens) like the other Forth’s in this investigation. Instead, it uses .DAT files. Instead of performing a block load like:

```
1 LOAD
```
pd10 loads the files with commands like:

```
LOAD BFPRIME1.DAT
```
with the filename NOT enclosed in quotes.

In order to edit such a file, pd10 must first load the editor with:

```
LOAD EDIT
```

(pd10 SuperForth likes ALL CAPS.)

The editor doesn’t like any lines longer than 32 characters.

However, SuperForth will also accept any standard ASCII file as input to the LOAD command.

pd10 SuperForth likes ALL CAPS.

Use `ENDIF` at the end of `IF` constructs; don’t try to define `THEN`.

Apparently, pd10 doesn’t like indented lines or stack comments.

In this investigation, it also didn’t like the `REPORTRESULTS` word. I used `RR` instead and it worked fine.

`U.` works. `NOT` works. `J` works. `1-` works. `0>` does not work. Predefine: `0> 0 >`; instead.

Two apparent variations from the fig-FORTH standard:

1. `+LOOP` does not work. pd10 uses `LOOP+` instead.

2. `: TEST 10 0 DO I . LOOP ;` does not work as expected in pd10; it sequences all the way through loops 0 through 10 instead of stopping before 10. Therefore, in this investigation I used constructs similar to: `TEST 9 0 DO I . LOOP ;` instead.

**Talbot ColorForth**

Talbot ColorForth is available on the (CoCo Archive). It was downloaded from:

[http://www.colorcomputerarchive.com/coco/Disks/Programming/](http://www.colorcomputerarchive.com/coco/Disks/Programming/) under the title:

Color Forth (Talbot Microsystems).zip
Talbot ColorForth uses the standard CoCo text screen; 32 characters wide by 16 lines high.

The zip file expands to two .dsk files and two .txt files:

CCF-EXEC.DSK
CCF-SRC.DSK
ColorForthMemMap.txt
ColorForthUserNotes.txt

This is Talbot’s ColorForth 1.1 by TJZ (T. J. Zimmer) and RJT (R. J. Talbot), as modified by SMP (Stephen M. Pereira) in January 2015 for Disk Extended Color Basic.

The .txt files constitute Stephen’s Instruction Manual for his modifications.

The original MicroWorks Talbot Instruction Manual is also available on the (CoCo Archive). It was downloaded from:

http://www.colorcomputerarchive.com/coco/Documents/Manuals/Programming/ under the title:

ColorForth v1.0 (Talbot Microsystems).pdf

The CCF-EXEC.DSK directory shows four files:

CC-FORTH BIN  2 B 3
CCF-MISC BIN  2 B 4
CCF-ED BIN  2 B 4
CCF-DOLR BIN  2 B 4

These are the Machine Language files for the system and the editor, as well as the CCF-DOLR.BIN file which is an example demonstrating different ways to combine coins to result in an amount of $1.00 total.

The CC-FORTH.BIN file is the primary system file. It loads a subset of fig-FORTH which allows for many common operations, and also provides the most memory space for user programs.

To load and run the system, use the usual:

LOADM”CC-FORTH
EXEC

The CCF-MISC.BIN file includes the remainder of the figFORTH system as well as some additional utility words. Its addition allows user programs to utilize all of the features of fig-FORTH but takes up more memory space. It must also be loaded before the editor can be loaded.
The **CCF-ED.BIN** file is the Screen (a.k.a. Block) editor.

Except for **CC-FORTH.BIN**, although the files have the `.BIN` extension, they are not directly executable. Talbot ColorForth uses the `.BIN` extension for its screen files as well, a practice which is somewhat confusing. Attempting to EXEC any of these files just returns a deceptive “OK” result; deceptive because nothing seems to have actually executed.

These `.BIN` files each contain space for up to eight (8) screens. If you wish to load more than eight screens, you need to implement the excess in other file(s).

Also rather confusing are the places where one might expect to enter a space between words, but where a carriage return (i.e. the ENTER key) is required instead. In what follows, I will indicate such carriage returns with the symbol `<CR>`.

To load screens, you must first load the file itself, using **LSCR**, and then load the individual screens from that file. For example, to load the screens of the **CCF-MISC.BIN** file, the commands would be:

```
LSCR<CR>CCF-MISC
4 3 2 1 LOAD LOAD LOAD LOAD<CR>
```

Note that there is no `<CR>` after **CCF-MISC** – this is because if the filename is exactly eight characters long, the system performs the `<CR>` for you automatically.

This load gives you the full fig-FORTH system plus some utility words. Also note that **CCF-MISC.BIN** MUST be loaded this way BEFORE you try to load the editor. To load the editor at this point, execute the following:

```
LSCR<CR>CCF-ED<CR>
7 6 5 4 3 2 LOAD LOAD LOAD LOAD LOAD LOAD<CR>
```

Note that, here, the `<CR>` MUST follow the **CCF-ED** because it is less than eight characters long.

Then, after all this, you can load a screens file to be edited. Put the target disk into the drive and enter:

```
LSCR<CR>FILENAME
```

adding a `<CR>` if the filename is shorter than eight characters. You can leave off the `.BIN` extension: it is understood.

While loading a screen, ColorForth may report something like:

```
REDEF: LIMIT
```
This just means that your `LIMIT` word is a redefinition of an existing word named `LIMIT`. This is not a problem unless you intend to use the original `LIMIT` somewhere else in your code.

At the end of the block load, I’ve found that Talbot displays a red character-sized box immediately followed by “(-? 0” which apparently means that zero errors were encountered during the load. This is somewhat counter-intuitive since most Forths use the “?” character to indicate that some word is not recognized in this context, and instead use “OK” to indicate no errors.

After you have finished editing a set of eight (or fewer) screens, you must save them to a filename using:

```
SSCR<CR>FILENAME
```

You should note that Talbot ColorForth screens (a.k.a. blocks), instead of being the usual Forth 64 characters wide by 16 lines high, are each 32 characters wide by 32 lines long.

Once the file has been loaded, you can begin editing the screens with:

```
N1 CLEAR<CR>  (only if it is a new screen, or if you want to wipe it clean)
N1 EDIT<CR>
```

where `N1` is the screen number (from 0 to 7).

**Editing is line-by-line:** Talbot’s does not include a screen editor. To edit a line, enter:

```
N2 T<CR>  
P TEXT TO BE ENTERED<CR>
```

where `N2` is the line number (from 0 to 31) and “P” means to overwrite the line with the following text. To insert text, delete lines, or perform other line editing tasks, please refer to the manuals.

After you are finished editing, do:

```
SSCR<CR>FILENAME
```

To save the file which can thereafter be `LSCR`ed and its screens `LOAD`ed as indicated above.

It has been my experience that any `LSCR` or `SSCR` error (such as using a space instead of a `<CR>`, or forgetting to change the disk in the drive) will result in an `SN ERROR` and the system will stop working properly until you do a Cold Start on the CoCo, and begin all over again from scratch.

Talbot ColorForth is a fig-FORTH. As such, variables must be defined with a leading zero, i.e.
0 VARIABLE AVAR

rather than the Forth-83 style:

VARIABLE AVAR

While it’s true that loading CCF-MISC.BIN does indeed add the rest of the fig-FORTH words, it also takes up a lot of memory space. Stephen Pereira’s Memory Map shows that the memory space available for the Talbot ColorForth 1.1 User Dictionary Space only runs from $4380 to $5800 (5,249 bytes). In particular, our Sieve of Eratosthenes Benchmark exceeds the space available when CCF-MISC has been loaded.

So, instead of loading CCF-MISC, I just used the base CC-FORTH.BIN and predefined the following words where necessary:

Throughout, I used 0 > instead of the non-existant 0>

: NOT IF 0 ELSE 65535 ENDIF ;
: U. 0 <# #S #> TYPE SPACE ;
: 2DUP OVER OVER ;
: WHILE [COMPILE] IF 2+ ; IMMEDIATE
: AGAIN 1 ?PAIRS COMPILE BRANCH BACK ; IMMEDIATE
: REPEAT >R >R [COMPILE] AGAIN R> R> 2 – [COMPILE] ENDIF ; IMMEDIATE
: +LOOP 3 ?PAIRS COMPILE (+LOOP) BACK ; IMMEDIATE

I used ENDIF instead of THEN throughout.

And finally, note that : TEST 10 0 DO I . LOOP ; works as expected in Talbot, sequencing through loops 0 through 9 and stopping before 10.

eForth

eForth by Frank Hogg Laboratory is available on the (CoCo Archive). It was downloaded from:

http://www.colorcomputerarchive.com/coco/Disks/Programming/ under the title:

eFORTH (Keyboard Patch) (Frank Hogg Laboratory).zip
The original “eFORTH (Frank Hogg Laboratory).zip” (without the keyboard patch) is also available on the (CoCo Archive) but it does not appear to work.

eForth uses its own (apparently PMODE 4) text screen, which is 51 characters wide by 24 characters high.

eForth is an 83-Standard Forth. It uses its own line editor. No screen editor is provided. The line editor uses a mechanism similar to Talbot Colorforth’s line editor, e.g.

```
   n2 t to place the cursor on line number n2
   p Text to be Inserted to overwrite the text on the selected line.
```

Refer to the eForth manual for further details.

The line editor provides the `wipe` word to clear the block. But it can’t be used by itself. Instead, you have to use the “`editor wipe`” two-word command.

The first line of each block is intended (as is usual in Forth) to be a comment. But, you don’t add the closing parentheses yourself. eForth adds your initials, the date, and the closing parentheses automatically. Your initials are "cee" unless you’ve previously specified otherwise (read your manual!), and the date seems to be stuck at “23jan84” no matter what.

After you’ve finished editing a block, be sure to enter the “`flush`” command to save it to the disk.

eForth does not utilize nor provide any access to the standard CoCo Timer. For the eForth runs of the Benchmarks, it was thus necessary to use the stopwatch instead.

```
   . works. 1- works. j works. bell works.
```

In 83-Standard Forth, the word “`not`” does a one’s complement on the entry on the top of the stack. While functionally correct, I chose to use the (operationally identical in this instance) Logical not in the eForth applications, i.e.:

```
   lnot if 0 else 65535 then ;
```

Also, `0>` is not present, so I predefined:

```
   : 0> 0 > ;
```

And finally, note that `test 10 0 do i . loop ;` works as expected in eForth, sequencing through loops 0 through 9 and stopping before 10.
Appendix C  --  Calibration and the CoCo Timer

The (CoCo Manual, p. 221) tells us that:

Your computer has a built-in *timer* that measures time in sixtieths of a second (approximately). The moment you power-up the computer, the timer begins counting at zero. When it counts to 65535 (approximately 18 minutes later), the timer starts over at zero. It pauses during cassette and printer operations.

So the Color Computer Timer operates at a nominal rate of 60 ticks per second. Since the tests for this investigation are being performed on the Vcc Emulator, it’s important to calibrate the Emulator’s Timer.

We therefore run the Emulator against the stopwatch to determine the actual number of ticks per second we are encountering:

```
100 '**********
110 '* TIMTST01.BAS
120 '* TIMER TEST 01
130 '* MDJ 2019/01/16
140 '**********
150 'ZERO THE TIMER
160 POKE 274,0
170 POKE 275,0
180 'GET THE CURRENT TIMER VALUE
190 T1=PEEK(274)
200 T2=PEEK(275)
210 T=(T1*256)+T2
220 PRINT T;
230 GOTO 190
240 'TIME THE RUN WITH A
250 ' STOPWATCH.
260 'CLICK THE BREAK KEY ON THE
270 ' COCO AT THE SAME MOMENT
280 ' THAT YOU CLICK THE
290 ' STOPWATCH.
300 'DIVIDE THE LAST REPORTED
310 ' TIMER VALUE BY THE NUMBER
320 ' OF SECONDS ELAPSED TO GET
330 ' THE APPROXIMATION OF THE
340 ' NUMBER OF TICKS PER
350 ' SECOND.
```
Timer Test Results:

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
<th>Stopwatch</th>
<th>Ticks/Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3573</td>
<td>60.10</td>
<td>59.45</td>
</tr>
<tr>
<td>2</td>
<td>3583</td>
<td>60.20</td>
<td>59.52</td>
</tr>
<tr>
<td>3</td>
<td>3590</td>
<td>60.30</td>
<td>59.54</td>
</tr>
<tr>
<td>4</td>
<td>3574</td>
<td>60.16</td>
<td>59.41</td>
</tr>
<tr>
<td>5</td>
<td>3587</td>
<td>60.24</td>
<td>59.55</td>
</tr>
<tr>
<td>6</td>
<td>3580</td>
<td>60.12</td>
<td>59.55</td>
</tr>
<tr>
<td>7</td>
<td>3584</td>
<td>60.27</td>
<td>59.47</td>
</tr>
<tr>
<td>8</td>
<td>3575</td>
<td>60.16</td>
<td>59.42</td>
</tr>
<tr>
<td>9</td>
<td>3578</td>
<td>60.11</td>
<td>59.52</td>
</tr>
<tr>
<td>10</td>
<td>3587</td>
<td>60.28</td>
<td>59.51</td>
</tr>
</tbody>
</table>

Mean s          59.494
Mean s          0.0527

Therefore, in all our tests, when converting from reported Timer values to equivalent minutes and seconds, we assume the CoCo Timer is operating at a uniform rate of 59.494 ticks/second.
Appendix D -- Brute Force Primes Benchmark

Our Brute Force Primes Benchmark is an adaptation, in Basic, Assembly Language, and the various Forths, of Calmatory’s Basic Brute Force method in C, with no optimizations, as presented at:


Our method finds all the prime numbers up to 290. The number 290 was chosen because, in Basic, the timer values obtained approached the timer limit of 65535. Thus the timer would not roll over during the Basic runs and, simultaneously, the other runs would enjoy the greatest possible precision of results within the limit imposed by the Basic runs.

The Basic Program:

```
100 '**********
110 '* BFPRIMES.BAS
120 '* BENCHMARK TESTER
130 '* MDJ 2019/01/16
140 '**********
150 'ZERO THE COCO TIMER
160 POKE 274,0
170 POKE 275,0
180 '*****
190 'BRUTE FORCE PRIMES
200 '*****
210 'SET LIMIT
220 L=290
230 DIM P(L)
240 FOR I=1 TO L
250 P(L)=0
260 NEXT I
270 P(2)=1
280 'GET THE PRIMES
290 FOR I=3 TO L
300 Q3=1
310 FOR J=2 TO I-1
320 Q=I/J
330 Q1=FIX(Q)
340 Q2=Q-Q1
350 IF(NOT(Q2>0)) THEN Q3=0
360 NEXT J
370 P(I)=Q3
```
380 NEXT I
390 'GET THE COCO TIMER VALUE
400 T1=PEEK(274)
410 T2=PEEK(275)
420 T=(T1*256)+T2
930 '*****
940 'REPORT THE RESULTS
950 PRINT "LIMIT: ";L
960 PRINT "PRIMES: ";
970 FOR I=1 TO L
980 IF(P(I)=1) THEN PRINT I;", ";
990 NEXT I
1000 PRINT"TIMER = ";T

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>62844</td>
</tr>
<tr>
<td>2</td>
<td>62847</td>
</tr>
<tr>
<td>3</td>
<td>62882</td>
</tr>
<tr>
<td>4</td>
<td>62890</td>
</tr>
<tr>
<td>5</td>
<td>62844</td>
</tr>
<tr>
<td>6</td>
<td>62853</td>
</tr>
<tr>
<td>7</td>
<td>62814</td>
</tr>
<tr>
<td>8</td>
<td>62864</td>
</tr>
<tr>
<td>9</td>
<td>62876</td>
</tr>
<tr>
<td>10</td>
<td>62864</td>
</tr>
</tbody>
</table>

Mean 62857.8
s 22.2850
= 1056.54 seconds
= 17 minutes 36.54 seconds
The Assembly Language Program without the assembly:

```
00100  **********
00110  * BFPRIM.ASM
00120  * BRUTE FORCE PRIMES BENCHMARK
00130  * MDJ 2019/01/19
00140  **********
00150  ORG     $3200
00160  PSHS    A,B,X,Y
00170  JMP     GP
00180  Q3      RMB     1       PRIME FLAG: 1 = PRIME
00190  LIMIT   RMB     2       TEST 4 THROUGH 290
00200  L1      RMB     2       L1 = LIMIT + 1
00220  TEMP    RMB     2
00230  PARRAY  RMB     291     PRIMES ARRAY
00260  GP      LDX     #4      OUTER LOOP COUNTER
00270  GP1     CMPX    L1
00280  BEQ     GP6     GO IF OUTER LOOP COMPLETE
00290  LDA     #1      SET PRIME FLAG
00300  STA     Q3
00310  LDY     #2      INNER LOOP COUNTER
00320  GP2     STX     TEMP
00330  CMPY    TEMP
00340  BEQ     GP5     GO IF INNER LOOP COMPLETE
00350  TFR     X,D
00360  STY     TEMP
00370  GP3     SUBD    TEMP
00380  CMPD    #0
00390  BGT     GP3     GO CONTINUE CALCULATION
00400  BLT     GP4     MOD > 0 (DIVISION NOT EXACT)
00410  CLRA    MOD = 0 (DIVISION IS EXACT)
00420  STA     Q3      CLEAR PRIME FLAG
00430  GP4     LEAY    1,Y     INCREMENT INNER LOOP COUNTER
00440  BRA     GP2
00450  GP5     PSHS    X,Y     STORE THE PRIMES ARRAY ENTRY
00460  LDY     #$320C  START OF PARRAY
00470  TFR     Y,D
00480  STX     TEMP
00490  ADDD    TEMP
00500  TFR     D,X
00510  LDA     Q3      GET PRIME FLAG
00520  STA     ,X     PUT IT TO PRIMES ARRAY
00530  PULS    X,Y
00540  LEAX    1,X     INCREMENT OUTER LOOP COUNTER
00550  BRA     GP1
00580  GP6     PULS    A,B,X,Y
00590  RTS     RETURN TO BASIC
```
00600       END

100  '**********
110  '* BFPRIM.BAS
120  '* BASIC SUPERVISOR FOR
130  '*   BFPRIM.ASM
140  '*   BRUTE FORCE PRIMES BENCHMARK
150  '* MDJ 2019/01/19
160  '**********
170  CLEAR 1024, &H31FF
180  LOADM "BFPRIM.BIN"
200  POKE &H3206, &H1           ' LIMIT = 290
210  POKE &H3207, &H22
220  POKE &H3208, &H1           ' L1 = 291
230  POKE &H3209, &H23
240  ' ZERO THE PRIMES ARRAY
250  FOR I = 0 TO 290
260  I1 = &H320C + I
270  POKE I1, 0
280  NEXT I
290  ' SET THE FIRST TWO PRIMES
300  POKE &H320E, 1
310  POKE &H320F, 1
311  POKE 274, 0
312  POKE 275, 0
320  EXEC &H3200               ' GO GET THE PRIMES
322  T1 = PEEK(274)
323  T2 = PEEK(275)
324  T = (T1 * 256) + T2
330  'REPORT THE RESULTS
340  PRINT "PRIMES: ";
350  FOR I = 0 TO 290
360  I1 = &H320C + I
370  I2 = PEEK(I1)
380  IF (I2 = 1) THEN PRINT I",";
390  NEXT I
400  PRINT "TIMER = ";
420  PRINT T
430  END
The Assembly Language Program with the assembly, but without the comments:

```
00100 ************
00110 * BFPRIM.ASM
00120 * BRUTE FORCE PRIMES BENCHMARK
00130 * MDJ 2019/01/19
00140 ************
3200 00150 ORG $3200
3200 34 36 00160 PSHS A,B,X,Y
3202 7E 332F 00170 JMP GP
3205 00180 Q3 RMB 1
3206 00190 LIMIT RMB 2
3208 00200 L1 RMB 2
320A 00220 TEMP RMB 2
320C 00230 PARRAY RMB 291
332F 8E 0004 00260 GP LDX #4
3332 BC 3208 00270 GP1 CMPX L1
3335 27 46 00280 BEQ GP6
3337 86 01 00290 LDA #1
3339 B7 3205 00300 STA Q3
333C 108E 0002 00310 LDY #2
3340 BF 320A 00320 GP2 STX TEMP
3343 10BC 320A 00330 CMPY TEMP
3347 27 19 00340 BEQ GP5
3349 1F 10 00350 TFR X,D
334B 10BF 320A 00360 STY TEMP
334F B3 320A 00370 GP3 SUBD TEMP
3352 1083 0000 00380 CMPD #0
3356 2E F7 00390 BGT GP3
3358 2D 04 00400 BLT GP4
335A 4F 00410 CLRA
335B B7 3205 00420 STA Q3
335E 31 21 00430 GP4 LEAY 1,Y
3360 20 DE 00440 BRA GP2
3362 34 30 00450 GP5 PSHS X,Y
3364 108E 320C 00460 LDY #$320C
3368 1F 20 00470 TFR Y,D
336A BF 320A 00480 STX TEMP
336D F3 320A 00490 ADDD TEMP
3370 1F 01 00500 TFR D,X
3372 B6 3205 00510 LDA Q3
3375 A7 84 00520 STA ,X
3377 35 30 00530 PULS X,Y
3379 30 01 00540 LEAX 1,X
337B 20 B5 00550 BRA GP1
337D 35 36 00580 GP6 PULS A,B,X,Y
337F 39 00590 RTS
```
TOTAL ERRORS

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GP</td>
<td>332F</td>
<td></td>
</tr>
<tr>
<td>GP1</td>
<td>3332</td>
<td></td>
</tr>
<tr>
<td>GP2</td>
<td>3340</td>
<td></td>
</tr>
<tr>
<td>GP3</td>
<td>334F</td>
<td></td>
</tr>
<tr>
<td>GP4</td>
<td>335E</td>
<td></td>
</tr>
<tr>
<td>GP5</td>
<td>3362</td>
<td></td>
</tr>
<tr>
<td>GP6</td>
<td>337D</td>
<td></td>
</tr>
<tr>
<td>L1</td>
<td>3208</td>
<td></td>
</tr>
<tr>
<td>LIMIT</td>
<td>3206</td>
<td></td>
</tr>
<tr>
<td>PARRAY</td>
<td>320C</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>3205</td>
<td></td>
</tr>
<tr>
<td>TEMP</td>
<td>320A</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>345</td>
</tr>
<tr>
<td>2</td>
<td>345</td>
</tr>
<tr>
<td>3</td>
<td>345</td>
</tr>
<tr>
<td>4</td>
<td>345</td>
</tr>
<tr>
<td>5</td>
<td>345</td>
</tr>
<tr>
<td>6</td>
<td>345</td>
</tr>
<tr>
<td>7</td>
<td>345</td>
</tr>
<tr>
<td>8</td>
<td>345</td>
</tr>
<tr>
<td>9</td>
<td>345</td>
</tr>
<tr>
<td>10</td>
<td>345</td>
</tr>
</tbody>
</table>

Mean = 345 s

= 5.80 seconds
The CF83 Forth Program:

**BLOCK NUMBER 1**

( CF83 Brute Force Primes Benchmark Test - 1/2 )
variable q3
variable limit 290 limit !
( Make array with 291 byte entries, 0 through 290 )
variable primesArray 289 allot
variable timerValue
: zeroTheArray ( -- ) limit @ 0 do 0 primesArray i + c! loop ;
: getPrimes ( -- )
limit @ 4 do
1 q3 !
i 1- 2 do
j i mod 0> not ( if NOT 0> )
if 0 q3 ! then
loop
q3 @ primesArray i + c!
loop ;

**BLOCK NUMBER 2**

( CF83 Brute Force Primes Benchmark Test - 2/2 )
: reportResults ( -- ) ." Limit: " limit @ u. cr
." Primes: "
limit @ 2 do
primesArray i + c@
if i u. ." , " then
loop
." Timer = " timerValue @ u. ;
: run ( -- )
zeroTheArray
1 primesArray 2+ c! 1 primesArray 3 + c!
0 274 ! ( Zero the CoCo timer )
getPrimes
274 @ timerValue ! ( Get the CoCo timer value )
reportResults ;
<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6774</td>
</tr>
<tr>
<td>2</td>
<td>6774</td>
</tr>
<tr>
<td>3</td>
<td>6774</td>
</tr>
<tr>
<td>4</td>
<td>6773</td>
</tr>
<tr>
<td>5</td>
<td>6774</td>
</tr>
<tr>
<td>6</td>
<td>6773</td>
</tr>
<tr>
<td>7</td>
<td>6773</td>
</tr>
<tr>
<td>8</td>
<td>6774</td>
</tr>
<tr>
<td>9</td>
<td>6774</td>
</tr>
<tr>
<td>10</td>
<td>6773</td>
</tr>
</tbody>
</table>

Mean = 6773.6

s = 0.51640

= 113.85 seconds
The Armadillo ColorForth 2.0 Program:

BLOCK NUMBER 1

( ARMADILLO BRUTE FORCE PRIMES BENCHMARK - 1/2 )
0 VARIABLE Q3 0 VARIABLE TEMP : NOT IF 0 ELSE 65535 THEN ;
0 VARIABLE LIMIT 290 LIMIT !
( MAKE ARRAY WITH 291 BYTE ENTRIES, 0 THROUGH 290 )
0 VARIABLE PRIMESARRAY 289 ALLOT
0 VARIABLE TIMERVALUE
: ZEROTHEARRAY ( -- ) LIMIT @ 0 DO 0 PRIMESARRAY I + C! LOOP ;
: GETPRIMES ( -- )
LIMIT @ 4 DO I TEMP !
 1 Q3 !
TEMP @ 1 - 2 DO
  TEMP @ I MOD 0 > NOT ( IF NOT 0> )
  IF 0 Q3 ! THEN
LOOP
Q3 @ PRIMESARRAY I + C!
LOOP ;

BLOCK NUMBER 2

( ARMADILLO BRUTE FORCE PRIMES BENCHMARK - 2/2 )
: REPORTRESULTS ( -- ) " LIMIT: " LIMIT @ . CR
  " PRIMES: "
LIMIT @ 2 DO
  PRIMESARRAY I + C@
  IF I .." , " THEN
LOOP
  " TIMER = " TIMERVALUE @ . ;
: RUN ( -- )
ZEROTHEARRAY
1 PRIMESARRAY 2 + C! 1 PRIMESARRAY 3 + C!
0 274 ! ( ZERO THE COCO TIMER )
GETPRIMES
274 @ TIMERVALUE ! ( GET THE COCO TIMER VALUE )
REPORTRESULTS ;
<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9590</td>
</tr>
<tr>
<td>2</td>
<td>9589</td>
</tr>
<tr>
<td>3</td>
<td>9590</td>
</tr>
<tr>
<td>4</td>
<td>9589</td>
</tr>
<tr>
<td>5</td>
<td>9590</td>
</tr>
<tr>
<td>6</td>
<td>9589</td>
</tr>
<tr>
<td>7</td>
<td>9589</td>
</tr>
<tr>
<td>8</td>
<td>9589</td>
</tr>
<tr>
<td>9</td>
<td>9590</td>
</tr>
<tr>
<td>10</td>
<td>9590</td>
</tr>
</tbody>
</table>

Mean: 9589.5
s: 0.52705

= 161.18 seconds
The pd10 SuperForth Program:

```
( BFPRIME1.DAT )
( PD-10 SUPERFORTH - 1/4 )
( BRUTE FORCE PRIMES BENCHMARK )
( MDJ 2019-01-20 )

( WORD: NOT PRESENT IN PD-10 SUPERFORTH )
: 0> 0 > ;

( NOTE: FIG REQUIRES NUMBER BEFORE VARIABLE )

( NOTE: N1 N2 DO LOOP RUNS UP THROUGH N1 )
( INSTEAD OF JUST UP TO IT. )

0 VARIABLE Q3
0 VARIABLE LIMIT 290 LIMIT !
0 VARIABLE L1
0 VARIABLE TEMP
( MAKE ARRAY W/291 BYTE ENTRIES, 0 THRU 290 )
0 VARIABLE PRIMESARRAY 289 ALLOT
0 VARIABLE TIMervalue

: ZA 0 PRIMESARRAY TEMP @ + C! ;

: ZEROlHEARRAY LIMIT @ 1- 0 DO I TEMP ! ZA LOOP ;

: PA LIMIT @ 1- 0 DO 1 PRIMESARRAY I + C! LOOP ;
: PB LIMIT @ 1- 0 DO PRIMESARRAY I + C@ U. LOOP ;

( BFPRIME2.DAT )
( PD-10 SUPERFORTH - 2/4 )
( BRUTE FORCE PRIMES BENCHMARK )
( MDJ 2019-01-20 )

0 VARIABLE TEMP1
0 VARIABLE TEMP2

: GA 1 Q3 ! ;
: GB IF 0 Q3 ! ENDIF ;
: GC Q3 @ PRIMESARRAY TEMP1 @ + C! ;

: GD TEMP1 @ TEMP2 @ MOD 0> NOT ;
: GE TEMP1 @ 2 - 2 DO I TEMP2 ! GD GB LOOP ;

: GETPRIMES LIMIT @ 1- 4 DO I TEMP1 ! GA GE GC LOOP ;
```
0 VARIABLE TEMP3

: RA ." LIMIT: " LIMIT @ U. CR ." PRIMES: " ;
: RB LIMIT @ 1- 2 ;
: RC PRIMESARRAY TEMP3 @ + C@ ;
: RD IF TEMP3 @ U. ." , " ENDIF ;
: RE ." TIMER = " TIMERVALUE @ U. ;

: REPORTRESULTS RA RB DO I TEMP3 ! RC RD LOOP RE ;

( BFPRIME4.DAT )
( PD-10 SUPERFORTH - 4/4 )
( BRUTE FORCE PRIMES BENCHMARK )
( MDJ 2019-01-20 )

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5869</td>
</tr>
<tr>
<td>2</td>
<td>5869</td>
</tr>
<tr>
<td>3</td>
<td>5869</td>
</tr>
<tr>
<td>4</td>
<td>5869</td>
</tr>
<tr>
<td>5</td>
<td>5869</td>
</tr>
<tr>
<td>6</td>
<td>5869</td>
</tr>
<tr>
<td>7</td>
<td>5869</td>
</tr>
<tr>
<td>8</td>
<td>5869</td>
</tr>
<tr>
<td>9</td>
<td>5869</td>
</tr>
<tr>
<td>10</td>
<td>5869</td>
</tr>
</tbody>
</table>

Mean 5869
s 0

= 98.65 seconds
The Talbot ColorForth Program:

This listing has been modified to eliminate trailing blank lines.

```
SCR 1
0 ( BFPRJ.BIN )
1 ( TALBOT COLORFORTH 1.1 )
2 ( BRUTE FORCE PRIMES BNCHMRK )
3 ( MDJ 2019-01-21 )
4 : NOT IF 0 ELSE 65535 ENDIF ;
5 : U. 0 <# #S #> TYPE SPACE ;
6 0 VARIABLE Q3
7 0 VARIABLE TEMP
8 0 VARIABLE LIMIT 290 LIMIT !
9 0 VARIABLE PRIMESARRAY 289 ALLOT
10
11 0 VARIABLE TIMEVERVALUE
12 : ZERO THE ARRAY ( -- )
13 LIMIT @ 0 DO
14 0 PRIMESARRAY I + C!
15 LOOP ;
16 : GETPRIMES ( -- )
17 LIMIT @ 4 DO I TEMP !
18 1 Q3 !
19 TEMP @ 1 - 2 DO
20 TEMP @ I MOD 0 > NOT
21 IF 0 Q3 ! ENDIF
22 LOOP
23 Q3 @ PRIMESARRAY I + C!
24 LOOP ;

SCR 2
0 : REPORTRESULTS ( -- )
1 ." LIMIT: " LIMIT @ U. CR
2 ." PRIMES: "
3 LIMIT @ 2 DO
4 PRIMESARRAY I + C@
5 IF I U. ." , " ENDIF
6 LOOP
7 ." TIMER = "
8 TIMIVERVALUE @ U. ;
9 : RUN ( -- )
10 ZERO THE ARRAY
11 1 PRIMESARRAY 2 + C!
12 1 PRIMESARRAY 3 + C!
13 0 274 ! ( ZERO COCO TIMER )
GETPRIMES
```
15   274 @ ( GET COCO TIMER )
16   TIMERSVALUE !
17   REPORTRESULTS ;

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7745</td>
</tr>
<tr>
<td>2</td>
<td>7745</td>
</tr>
<tr>
<td>3</td>
<td>7745</td>
</tr>
<tr>
<td>4</td>
<td>7745</td>
</tr>
<tr>
<td>5</td>
<td>7745</td>
</tr>
<tr>
<td>6</td>
<td>7745</td>
</tr>
<tr>
<td>7</td>
<td>7745</td>
</tr>
<tr>
<td>8</td>
<td>7745</td>
</tr>
<tr>
<td>9</td>
<td>7745</td>
</tr>
<tr>
<td>10</td>
<td>7745</td>
</tr>
</tbody>
</table>

Mean 7745
s 0

= 130.18 seconds
The eForth Program:

This eForth printout was manually massaged a bit - but just to omit erroneous 23jan84 date and the blank lines at the end of each block.

Block # 1
0 ( eForth Brute Force Primes Benchmark - 1/4 )
1 : 0> 0 > ;
2 : lnot if 0 else 65535 then ;
3 variable q3
4 variable limit 290 limit !
5 ( make array with 291 byte entries, 0 through 290 )
6 variable primesArray 289 allot
7 variable timerValue
8 : zeroTheArray ( -- ) limit @ 0 do
9    0 primesArray i + c! loop ;
10
11
12 : pa limit @ 0 do
13     1 primesArray i + c! loop ;
14 : pb limit @ 0 do
15     primesArray i + c@ u. loop ;

Block # 2
0 ( eForth Brute Force Primes Benchmark - 2/4 )
1 : getPrimes ( -- )
2     limit @ 4 do
3       1 q3 !
4       i 1- 2 do
5         j i mod 0> lnot ( if NOT 0> )
6         if 0 q3 ! then
7         loop
8       q3 @ primesArray i + c!
9       loop ;

Block # 3
0 ( eForth Brute Force Primes Benchmark - 3/4 )
1 : reportResults ( -- )
2    ." Limit : " limit @ u. cr
3    ." Primes: "
4    limit @ 2 do
5       primesArray i + c@
6       if i u. ." , " then
7       loop
8    ." Timer = " timerValue @ u. ;
Block # 4

0 ( eForth Brute Force Primes Benchmark - 4/4 )
1: run ( -- )
2 zeroTheArray
3 1 primesArray 2 + c!
4 1 primesArray 3 + c!
5 bell ( Signal user to start the stopwatch )
6 getPrimes
7 bell ( Signal user to stop the stopwatch )
8 reportResults ;

Run    Seconds

1  50.24
2  50.37
3  50.39
4  50.38
5  50.35
6  50.40
7  50.34
8  50.34
9  50.34
10 50.33

Mean  50.348
      s   0.04492

Say   50.35 seconds
The Brute Force Primes Recap:

<table>
<thead>
<tr>
<th>Language</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Language</td>
<td>5.80</td>
</tr>
<tr>
<td>eForth</td>
<td>50.35</td>
</tr>
<tr>
<td>pd10 SuperForth</td>
<td>98.65</td>
</tr>
<tr>
<td>CF83 Forth</td>
<td>113.85</td>
</tr>
<tr>
<td>Talbot ColorForth 1.1</td>
<td>130.18</td>
</tr>
<tr>
<td>Armadillo ColorForth 2.0</td>
<td>161.18</td>
</tr>
<tr>
<td>Basic</td>
<td>1056.34</td>
</tr>
</tbody>
</table>
Appendix E -- Sieve of Eratosthenes Benchmark

Our Sieve of Eratosthenes Benchmark is an adaptation, in Basic, Assembly Language, and the various Forths, of RosettaCode.org’s Sieve of Eratosthenes Benchmark in Forth, as presented at:

https://rosettacode.org/wiki/Sieve_of_Eratosthenes#Forth

Our method finds all the prime numbers up to 4364. The number 4364 was chosen because, in our Basic Program, any larger number results in an OM ERROR.

The Basic Program:

```
100 '**********
110 '* ERPRIMES.BAS
120 '* SIEVE OF ERATOSTHENES
130 '*   BENCHMARK
140 '* MDJ 2019/01/23
150 '**********
160 'SET LIMIT
170 L=4366
180 DIM P(L)     'PRIMES ARRAY
190 'SET THE ARRAY
200 FOR I=0 TO L 'OUTER LOOP COUNTER
210   P(I) = 1
220 NEXT I
230 P(0)=0
240 P(1)=0
250 'ZERO THE COCO TIMER
260 POKE 274,0
270 POKE 275,0
280 'GET THE PRIMES
290 FOR I = 2 TO L
300 IF (P(I) = 0) GOTO 400 'SKIP
310 PM = I * I 'SQUARE OF THE INDEX
320 IF (PM > L) GOTO 420 'DONE
330 P(PM) = 0
340 PS = PM 'INNER LOOP START INDEX
350 FOR J = PS TO L STEP I 'INNER LOOP COUNTER
360   PM = PM + I
370 IF PM>L GOTO400
380 P(PM) = 0
390 NEXT J
400 NEXT I
410 'GET THE COCO TIMER VALUE
420 T1=PEEK(274)
```
430  T2=PEEK(275)
440  T=(T1*256)+T2
450  'REPORT THE RESULTS
460  PRINT "PRIMES: ";
470  FOR I=1 TO L
480  IF(P(I)=1) THEN PRINT I;",";
490  NEXT I
500  PRINT "TIMER = ";T
510  END

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7523</td>
</tr>
<tr>
<td>2</td>
<td>7509</td>
</tr>
<tr>
<td>3</td>
<td>7515</td>
</tr>
<tr>
<td>4</td>
<td>7509</td>
</tr>
<tr>
<td>5</td>
<td>7520</td>
</tr>
<tr>
<td>6</td>
<td>7526</td>
</tr>
<tr>
<td>7</td>
<td>7509</td>
</tr>
<tr>
<td>8</td>
<td>7521</td>
</tr>
<tr>
<td>9</td>
<td>7519</td>
</tr>
<tr>
<td>10</td>
<td>7512</td>
</tr>
</tbody>
</table>

Mean 7516.3
s 6.3430

= 126.34 seconds
The Assembly Language Program without the assembly:

```
00100 **********
00110 * ERPRIM.ASM
00120 * SIEVE OF ERATOSTHENES
00130 * BENCHMARK
00140 * MDJ 2019/01/24
00150 **********
00160 ORG $3200
00170 PSHS A,B,U,X,Y
00180 JMP GP
00190 LIMIT RMB 2 TEST 4 THROUGH 4364
00200 L1 RMB 2 L1 = LIMIT + 1
00210 TEMP RMB 2
00220 PADDR RMB 2 START ADDRESS OF PARRAY
00230 OENTRY RMB 2 OFFSET OF PARRAY ENTRY
00240 SINDEX RMB 2 INNER LOOP START INDEX
00250 PARRAY RMB 4365 PRIMES ARRAY
00260 GP LDD #$3211 START OF PARRAY
00280 LDX #2 OUTER LOOP COUNTER
00290 GP1 CMPX L1 GO IF OUTER LOOP COMPLETE
00310 LDD PADDR GET PARRAY ENTRY
00320 STX TEMP
00330 ADDD TEMP
00340 TFR D,U
00350 LDA ,U
00360 CMPA #0
00370 BEQ GP5 SKIP IF ENTRY IS ZERO
00380 LDD TEMP SQUARE THE INDEX
00390 LDU TEMP SQUARING COUNTER
00400 LEAU -1,U
00410 GP2 CMPU #0
00420 BEQ GP3 GO IF SQUARING COMPLETE
00430 ADDD TEMP
00440 LEAU -1,U DECREMENT SQUARING COUNTER
00450 BRA GP2
00460 GP3 STD OENTRY PARRAY ENTRY OFFSET
00470 STD SINDEX INNER LOOP START INDEX
00480 CMPD LIMIT
00490 BHI GP6 EXIT IF DONE
00500 PSHS A,B,U,X,Y ZERO THE ENTRY
00510 LDD PADDR
00520 ADDD OENTRY
00530 TFR D,X
00540 CLRA
00550 STA ,X
```
00560         PULS    A,B,X
00570         LDY     SINDEX  INNER LOOP COUNTER
00580 GP4     CMPY    LIMIT
00590         BHI     GP5     GO IF INNER LOOP COMPLETE
00600         STX     TEMP
00610         ADDD    TEMP
00620         CMPD    LIMIT
00630         BHI     GP5     EXIT INNER LOOP IF DONE
00640         STD     OENTRY
00650         PSHS    A,B,X   ZERO THE ENTRY
00660         LDD     PADDR
00670         ADDD    OENTRY
00680         TFR     D,X
00690         CLRA
00700         STA     ,X
00710         PULS    A,B,X
00720         LEAY    1,Y     INCREMENT INNER LOOP COUNTER
00730         BRA     GP4
00740 GP5     LEAX    1,X     INCREMENT OUTER LOOP COUNTER
00750         BRA     GP1
00760 GP6     PULS    A,B,U,X,Y
00770         RTS
00780         END

100 '**********
110 '* ERPRIM.BAS
120 '* BASIC SUPERVISOR FOR
130 '*   ERPRIM.ASM
140 '*   SIEVE OF ERATOSTHENES
150 '*   BENCHMARK
160 '* MDJ 2019/01/25
170 '**********
180 CLEAR 1024, &H31FF
190 LOADM "ERPRIM.BIN"
200 POKE &H3205, &H11          ' LIMIT = 4364
210 POKE &H3206, &H0C
220 POKE &H3207, &H11          ' L1 = 4365
230 POKE &H3208, &H0D
240 ' SET THE PRIMES ARRAY
250 FOR I = 0 TO 4364
260 I1 = &H3211 + I
270 POKE I1, 1
280 NEXT I
290 ' CLEAR ENTRIES ZERO AND ONE
300 POKE &H3211, 0
310 POKE &H3212, 0
311 'ZERO THE COCO TIMER
320 POKE 274, 0
330 POKE 275, 0
340 EXEC &H3200               ' GO GET THE PRIMES
341 'GET THE COCO TIMER VALUE
350 T1 = PEEK(274)
360 T2 = PEEK(275)
370 T = (T1 * 256) + T2
380 'REPORT THE RESULTS
390 PRINT "PRIMES: ";
400 FOR I = 0 TO 4364
410 I1 = &H3211 + I
420 I2 = PEEK(I1)
430 IF (I2 = 1) THEN PRINT I;",";
440 NEXT I
450 PRINT "TIMER = ";
460 PRINT T
470 END
The Assembly Language Program with the assembly, but without the comments:

```
00100 **********
00110 * ERPRIM.ASM
00120 * SIEVE OF ERATOSTHENES
00130 * BENCHMARK
00140 * MDJ 2019/01/24
00150 **********

3200               00160 ORG $3200
3200 34 76       00170 PSHS A,B,U,X,Y
3202 7E 431E     00180 JMP GP
3205               00190 LIMIT RMB 2
3207               00200 L1 RMB 2
3209               00210 TEMP RMB 2
320B               00220 PADDR RMB 2
320D               00230 OENTRY RMB 2
320F               00240 SINDEX RMB 2
3211               00250 PARRAY RMB 4365
431E CC 3211     00260 GP LDD #$3211
4321 FD 320B     00270 STD PADDR
4324 8E 0002     00280 LDX #2
4327 BC 3207     00290 CMPX L1
432A 27 71       00300 BEQ GP6
432C FC 320B     00310 LDD PADDR
432F BF 3209     00320 STX TEMP
4332 F3 3209     00330 ADDD TEMP
4335 1F 03       00340 TFR D,U
4337 A6 C4       00350 LDA ,U
4339 81 00       00360 CMPA #0
433B 27 5C       00370 BEQ GP5
433D FC 3209     00380 LDD TEMP
4340 FE 3209     00390 LDU TEMP
4343 33 5F       00400 LEAU -1,U
4345 1183 0000   00410 GP2 CMPU #0
4349 27 07       00420 BEQ GP3
434B F3 3209     00430 ADDD TEMP
434E 33 5F       00440 LEAU -1,U
4350 20 F3       00450 BRA GP2
4352 FD 320D     00460 GP3 STD OENTRY
4355 FD 320F     00470 STD SINDEX
4358 10B3 3205   00480 CMPD LIMIT
435C 22 3F       00490 BHI GP6
435E 34 16       00500 PSHS A,B,X
4360 FC 320B     00510 LDD PADDR
4363 F3 320D     00520 ADDD OENTRY
4366 1F 01       00530 TFR D,X
4368 4F         00540 CLRA
```
4369 A7 84 00550 STA ,X
436B 35 16 00560 PULS A,B,X
436D 10BE 320F 00570 LDY SINDEX
4371 10BC 3205 00580 GP4 CMPY LIMIT
4375 22 22 00590 BHI GP5
4377 BF 3209 00600 STX TEMP
437A F3 3209 00610 ADDD TEMP
437D 10B3 3205 00620 CMPD LIMIT
4381 22 16 00630 BHI GP5
4383 FD 320D 00640 STD OENTRY
4386 34 16 00650 PSHS A,B,X
4388 FC 320B 00660 LDD PADDR
438B F3 320D 00670 ADDD OENTRY
438E 1F 01 00680 TFR D,X
4390 4F 00690 CLRA
4391 A7 84 00700 STA ,X
4393 35 16 00710 PULS A,B,X
4395 31 21 00720 LEAY 1,Y
4397 20 D8 00730 BRA GP4
4399 30 01 00740 GP5 LEAX 1,X
439B 20 8A 00750 BRA GP1
439D 35 76 00760 GP6 PULS A,B,U,X,Y
439F 39 00770 RTS
0000 00780 END

00000 TOTAL ERRORS

GP  431E
GP1 4327
GP2 4345
GP3 4352
GP4 4371
GP5 4399
GP6 439D
L1  3207
LIMIT 3205
OENTRY 320D
PADDR 320B
PARRAY 3211
SINDEX 320F
TEMP  3209
<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
</tr>
<tr>
<td>3</td>
<td>46</td>
</tr>
<tr>
<td>4</td>
<td>46</td>
</tr>
<tr>
<td>5</td>
<td>46</td>
</tr>
<tr>
<td>6</td>
<td>46</td>
</tr>
<tr>
<td>7</td>
<td>46</td>
</tr>
<tr>
<td>8</td>
<td>46</td>
</tr>
<tr>
<td>9</td>
<td>46</td>
</tr>
<tr>
<td>10</td>
<td>46</td>
</tr>
</tbody>
</table>

Mean 45.9
s 0.3162

= 0.77 seconds
The CF83 Forth Program:

**BLOCK NUMBER 5**

( CF83 Eratosthenes Sieve Primes Benchmark Test - 1/2 )

```
variable timerValue
: 2dup ( 32b -- 32b 32b ) over over ;
: primes? ( n -- flag ) here + c@ 0= ;
: erase ( addr u -- ) 0 fill ;
: composite! ( n -- ) here + 1 swap c! ;

: sieve ( n -- ) here over erase 2
    begin 2dup dup * >
    while dup primes? 
        if 2dup dup * do i composite! dup +loop
        then 1+repeat drop ;
```

**BLOCK NUMBER 6**

( CF83 Eratosthenes Sieve Primes Benchmark Test - 2/2 )

```
: reportResults ( -- ) cr ." Primes: "
    4364 2 do i primes? if i . then loop
    ." Timer = " timerValue @ u. ;
: run ( -- )
    0 274 ! ( Zero the CoCo timer )
    4364 sieve
    274 @ timerValue ! ( Get the CoCo timer value )
    reportResults ;
```
<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>280</td>
</tr>
<tr>
<td>2</td>
<td>280</td>
</tr>
<tr>
<td>3</td>
<td>280</td>
</tr>
<tr>
<td>4</td>
<td>280</td>
</tr>
<tr>
<td>5</td>
<td>280</td>
</tr>
<tr>
<td>6</td>
<td>279</td>
</tr>
<tr>
<td>7</td>
<td>280</td>
</tr>
<tr>
<td>8</td>
<td>280</td>
</tr>
<tr>
<td>9</td>
<td>279</td>
</tr>
<tr>
<td>10</td>
<td>280</td>
</tr>
</tbody>
</table>

**Mean** 279.8 s 0.4216 = 4.70 seconds
The Armadillo ColorForth 2.0 Program:

**BLOCK NUMBER 4**

```
( ARMADILLO ERATOCTHENES SIEVE PRIMES BENCHMARK TEST - 1/2 )
0 VARIABLE TIMVALUE
: 2DUP ( 32B -- 32B 32B ) OVER OVER ;
: PRIMES? ( N -- FLAG ) HERE + C@ 0= ;
: ERASE ( ADDR U -- ) 0 FILL ;
: COMPOSITE! ( N -- ) HERE + 1 SWAP C! ;

: SIEVE ( N -- ) HERE OVER ERASE 2
  BEGIN 2DUP DUP * >
  WHILE DUP PRIMES?
    IF 2DUP DUP * DO I COMPOSITE! DUP +LOOP
    THEN 1+
  REPEAT DROP ;
```

**BLOCK NUMBER 5**

```
( ARMADILLO ERATOCTHENES SIEVE PRIMES BENCHMARK TEST - 2/2 )

: REPORTRESULTS ( -- ) CR ." PRIMES: "
  4364 2 DO I PRIMES? IF I . THEN LOOP
  ." TIMER = " TIMVALUE @ . ;

: RUN ( -- )
  0 274 ! ( ZERO THE COCO TIMER )
  4364 SIEVE
  274 @ TIMVALUE ! ( GET THE COCO TIMER VALUE )
  REPORTRESULTS ;
```
<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>236</td>
</tr>
<tr>
<td>2</td>
<td>236</td>
</tr>
<tr>
<td>3</td>
<td>236</td>
</tr>
<tr>
<td>4</td>
<td>236</td>
</tr>
<tr>
<td>5</td>
<td>236</td>
</tr>
<tr>
<td>6</td>
<td>236</td>
</tr>
<tr>
<td>7</td>
<td>236</td>
</tr>
<tr>
<td>8</td>
<td>236</td>
</tr>
<tr>
<td>9</td>
<td>236</td>
</tr>
<tr>
<td>10</td>
<td>236</td>
</tr>
</tbody>
</table>

Mean 236
s 0

= 3.97 seconds
The pd10 SuperForth Program:

\[
\text{( ERPRIME1.DAT )} \\
\text{( PD-10 SUPERFORTH - 1/2 )} \\
\text{( SIEVE OF ERATOSTHENES PRIMES BENCHMARK )} \\
\text{( MDJ 2019-03-24 )} \\
\]

\[(\text{NOTE: N1 N2 DO LOOP RUNS UP THROUGH N1 })\] 
\[(\text{INSTEAD OF JUST UP TO IT. })\] 

\[(\text{NOTE: PD-10 FILL DOES NOT APPEAR TO WORK WITH HERE})\] 
\[(\text{USE AHERE VARIABLE ARRAY INSTEAD})\]

\[0 \text{ VARIABLE AHERE 4364 ALLOT}\]
\[: \text{ERASE 0 FILL ;} \]
\[: \text{PRMQ AHERE + C@ 0= ;} \]
\[: \text{CMPST AHERE + 1 SWAP C! ;} \]
\[: \text{S1 IF 2DUP DUP * DO I CMPST DUP LOOP+ ENDIF 1+ ;} \]
\[: \text{S2 BEGIN 2DUP DUP * > WHILE DUP PRMQ S1 REPEAT ;} \]
\[: \text{SIEVE AHERE OVER ERASE 2 S2 DROP ;} \]

\[
\text{( ERPRIME2.DAT )} \\
\text{( PD-10 SUPERFORTH - 2/2 )} \\
\text{( SIEVE OF ERATOSTHENES PRIMES BENCHMARK )} \\
\text{( MDJ 2019-03-24 )} \\
\]

\[0 \text{ VARIABLE TIMERVALUE}\]
\[: \text{R1 4364 2 DO I PRMQ IF I . ENDIF LOOP ;} \]
\[: \text{R2 ." TIMER = " TIMERVALUE @ U. ;} \]
\[: \text{RR CR ." PRIMES: " R1 R2 ;} \]
\[: \text{U1 0 274 ! 4364 SIEVE ;} \]
\[: \text{U2 274 @ TIMERVALUE ! RR ;} \]
\[: \text{RUN U1 U2 ;} \]
<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>268</td>
</tr>
<tr>
<td>2</td>
<td>268</td>
</tr>
<tr>
<td>3</td>
<td>268</td>
</tr>
<tr>
<td>4</td>
<td>268</td>
</tr>
<tr>
<td>5</td>
<td>268</td>
</tr>
<tr>
<td>6</td>
<td>268</td>
</tr>
<tr>
<td>7</td>
<td>268</td>
</tr>
<tr>
<td>8</td>
<td>268</td>
</tr>
<tr>
<td>9</td>
<td>268</td>
</tr>
<tr>
<td>10</td>
<td>268</td>
</tr>
</tbody>
</table>

Mean 268

s 0

= 4.50 seconds
The Talbot ColorForth Program:

This listing has been modified to eliminate trailing blank lines.

SCR 1
0 ( ERPRI.BIN )
1 ( TALBOT COLORFORTH 1.1 )
2 ( ERATOSTHENES PRIMES BENCHMRK )
3 ( MDJ 2019-03-28 )
4 0 VARIABLE AHERE 4364 ALLOT
5 : 2DUP OVER OVER ;
6 : WHILE [COMPILE] IF 2+
7 ; IMMEDIATE
8 : AGAIN 1 ?PAIRS COMPILE
9 BRANCH BACK ; IMMEDIATE
10 : REPEAT >R >R [COMPILE] AGAIN
11 R> R> 2 - [COMPILE] ENDIF
12 ; IMMEDIATE
13 : +LOOP 3 ?PAIRS COMPILE (+LOOP)
14 BACK ; IMMEDIATE
15
16 0 VARIABLE TIMERVERVALUE
17 : PRIMES? AHERE + C@ 0= ;
18 : COMP! AHERE + 1 SWAP C! ;
19 : S1 IF 2DUP DUP * DO I COMP!
20 DUP +LOOP ENDIF 1+ ;
21 : SIEVE AHERE OVER ERASE 2
22 BEGIN 2DUP DUP * >
23 WHILE DUP PRIMES? S1
24 REPEAT DROP ;

SCR 2
0 ( ERPRI.BIN )
1 : R1 4364 2 DO I PRIMES?
2 IF I . ENDF LOOP ;
3 : REPORTRESULTS
4 CR ." PRIMES: " R1
5 ." TIMER = "
6 TIMERVERVALUE @ . ;
7 : RUN 0 274 ! 4364 SIEVE 274 @
8 TIMERVERVALUE !
9 REPORTRESULTS ;
<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>195</td>
</tr>
<tr>
<td>2</td>
<td>195</td>
</tr>
<tr>
<td>3</td>
<td>195</td>
</tr>
<tr>
<td>4</td>
<td>195</td>
</tr>
<tr>
<td>5</td>
<td>195</td>
</tr>
<tr>
<td>6</td>
<td>195</td>
</tr>
<tr>
<td>7</td>
<td>195</td>
</tr>
<tr>
<td>8</td>
<td>195</td>
</tr>
<tr>
<td>9</td>
<td>195</td>
</tr>
<tr>
<td>10</td>
<td>195</td>
</tr>
</tbody>
</table>

**Mean**

s 0

= **3.28 seconds**
The eForth Program:

This eForth printout was manually massaged a bit but just to omit erroneous 23jan84 date and the blank lines at the end of each block.

Block # 7
0 ( eForth Eratosthenes Sieve Primes Benchmark - 1/2 )
1
2 : primes? ( n -- flag ) here + c@ 0= ;
3
4 : composite! ( n -- ) here + 1 swap c! ;
5
6 : sieve ( n -- ) here over erase 2
7     begin 2dup dup * >
8     while dup primes?
9       if 2dup dup * do i composite! dup +loop
10       then 1+
11     repeat drop ;

Block # 8
0 ( eForth Eratosthenes Sieve Primes Benchmark - 2/2 )
1
2 : reportresults ( -- ) cr ." Primes: "
3     4364 2 do i primes? if i . then loop ;
4
5 : run ( -- )
6     bell ( Signal user to start the stopwatch )
7     4364 sieve
8     bell ( Signal user to stop the stopwatch )
9     reportresults ;
<table>
<thead>
<tr>
<th>Run</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.88</td>
</tr>
<tr>
<td>2</td>
<td>2.95</td>
</tr>
<tr>
<td>3</td>
<td>2.84</td>
</tr>
<tr>
<td>4</td>
<td>2.88</td>
</tr>
<tr>
<td>5</td>
<td>2.84</td>
</tr>
<tr>
<td>6</td>
<td>2.86</td>
</tr>
<tr>
<td>7</td>
<td>2.86</td>
</tr>
<tr>
<td>8</td>
<td>2.87</td>
</tr>
<tr>
<td>9</td>
<td>2.86</td>
</tr>
<tr>
<td>10</td>
<td>2.90</td>
</tr>
</tbody>
</table>

Mean 2.874
s 0.03239

Say 2.87 seconds
The Sieve of Eratosthenes Primes Recap:

<table>
<thead>
<tr>
<th>Language</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Language</td>
<td>0.77</td>
</tr>
<tr>
<td>eForth</td>
<td>2.87</td>
</tr>
<tr>
<td>Talbot ColorForth 1.1</td>
<td>3.28</td>
</tr>
<tr>
<td>Armadillo ColorForth 2.0</td>
<td>3.97</td>
</tr>
<tr>
<td>pd10 SuperForth</td>
<td>4.50</td>
</tr>
<tr>
<td>CF83 Forth</td>
<td>4.70</td>
</tr>
<tr>
<td>Basic</td>
<td>126.34</td>
</tr>
</tbody>
</table>
Appendix F -- Add Loops Benchmark

Our Add Loops Benchmark simply performs an addition and store 165,000 times. The number 165,000 was chosen because, in Basic, the timer values obtained approached the timer limit of 65535. Thus the timer would not roll over during the Basic runs and, simultaneously, the other runs would enjoy the greatest possible precision of results within the limit imposed by the Basic runs.

The Basic Program:

```
100 '**********
110 '* ADDLOOPS.BAS
120 '* ADDING LOOPS BENCHMARK
130 '* MDJ 2019/01/25
140 '**********
150 'ZERO THE COCO TIMER
160 POKE 274,0
170 POKE 275,0
180 'DO THE LOOPS
190 FOR I=1 TO 165
200 FOR J=1 TO 1000
210 A = 5 + 7
220 NEXT J
230 NEXT I
240 'GET THE COCO TIMER VALUE
250 T1=PEEK(274)
260 T2=PEEK(275)
270 T=(T1*256)+T2
280 'REPORT THE RESULTS
290 PRINT"TIMER = ";T
300 END
```
<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64712</td>
</tr>
<tr>
<td>2</td>
<td>64804</td>
</tr>
<tr>
<td>3</td>
<td>64781</td>
</tr>
<tr>
<td>4</td>
<td>64745</td>
</tr>
<tr>
<td>5</td>
<td>64773</td>
</tr>
<tr>
<td>6</td>
<td>64763</td>
</tr>
<tr>
<td>7</td>
<td>64750</td>
</tr>
<tr>
<td>8</td>
<td>64743</td>
</tr>
<tr>
<td>9</td>
<td>64743</td>
</tr>
<tr>
<td>10</td>
<td>64753</td>
</tr>
</tbody>
</table>

Mean: 64756.7

\[ 25.171 \text{ s} \]

= 1088.46 seconds

= 18 minutes 8.46 seconds
The Assembly Language Program without the assembly:

```
00100 **********
00110 * ADDLOOP.ASM
00120 * ADDING LOOPS BENCHMARK
00130 * MDJ 2019/01/26
00140 **********
00150         ORG     $3200
00160         PSHS    A,B,X,Y
00170         JMP     GP
00180 AVAR    RMB     2
00190 GP      LDX     #1      OUTER LOOP COUNTER
00200 GP1     CMPX    #166
00210         BEQ     GP4     EXIT IF OUTER LOOP COMPLETE
00220         LDY     #1      INNER LOOP COUNTER
00230 GP2     CMPY    #1001
00240         BEQ     GP3     GO IF INNER LOOP COMPLETE
00250         LDD     #5      PERFORM THE ADDITION
00260         ADDD    #7
00270         STD     AVAR
00280         LEAY    1,Y     INCREMENT INNER LOOP COUNTER
00290         BRA     GP2
00300 GP3     LEAX    1,X     INCREMENT OUTER LOOP COUNTER
00310         BRA     GP1
00320 GP4     PULS    A,B,X,Y
00330         RTS
00340         END
```

```
100 '**********
110 '* ADDLOOP.BAS
120 '* BASIC SUPERVISOR FOR
130 '*   ADDLOOP.ASM
140 '*   ADDING LOOPS BENCHMARK
150 '* MDJ 2019/01/26
160 '**********
170 CLEAR 1024, &H31FF
180 LOADM "ADDLOOP.BIN"
190 'ZERO THE COCO TIMER
200 POKE 274, 0
210 POKE 275, 0
220 'PERFORM THE ADDITION LOOPS
230 EXEC &H3200
240 'GET THE COCO TIMER VALUE
250 T1 = PEEK(274)
260 T2 = PEEK(275)
270 T = (T1 * 256) + T2
280 'REPORT THE RESULTS
```
290 PRINT "TIMER = ";
300 PRINT T
310 END
The Assembly Language Program with the assembly, but without the comments:

```
00100 **********
00110 * ADDLOOP.ASM
00120 * ADDING LOOPS BENCHMARK
00130 * MDJ 2019/01/26
00140 **********
3200               00150         ORG     $3200
3200 34   36       00160         PSHS    A,B,X,Y
3202 7E   3207     00170         JMP     GP
3205               00180 AVAR    RMB     2
3207 8E   0001     00190 GP      LDX     #1
320A 8C   00A6     00200 GP1     CMPX    #166
320D 27   1B       00210         BEQ     GP4
320F 108E 0001     00220         LDY     #1
3213 108C 03E9     00230 GP2     CMPY    #1001
3217 27   0D       00240         BEQ     GP3
3219 CC   0005     00250         LDD     #5
321C C3   0007     00260         ADDD    #7
321F FD   3205     00270         STD     AVAR
3222 31   21       00280         LEAY    1,Y
3224 20   ED       00290         BRA     GP2
3226 30   01       00300 GP3     LEAX    1,X
3228 20   E0       00310         BRA     GP1
322A 35   36       00320 GP4     PULS    A,B,X,Y
322C 39               00330         RTS
00000 TOTAL ERRORS
AVAR    3205
GP      3207
GP1     320A
GP2     3213
GP3     3226
GP4     322A
```
<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>324</td>
</tr>
<tr>
<td>2</td>
<td>324</td>
</tr>
<tr>
<td>3</td>
<td>325</td>
</tr>
<tr>
<td>4</td>
<td>324</td>
</tr>
<tr>
<td>5</td>
<td>325</td>
</tr>
<tr>
<td>6</td>
<td>325</td>
</tr>
<tr>
<td>7</td>
<td>324</td>
</tr>
<tr>
<td>8</td>
<td>324</td>
</tr>
<tr>
<td>9</td>
<td>325</td>
</tr>
<tr>
<td>10</td>
<td>324</td>
</tr>
</tbody>
</table>

Mean: 324.4 s

= 5.45 seconds
The CF83 Forth Program:

BLOCK NUMBER 3

( CF83 Adding Loops Benchmark - 1/1 )
variable aVar
variable timerValue
: doAdd ( -- )
  165 0 do
    1000 0 do
      5 7 + aVar !
    loop
  loop ;
: run ( -- )
  0 274 ! ( Zero the CoCo Timer )
doAdd
  274 @ timerValue ! ( Get the CoCo Timer Value )
." Timer = " timerValue @ u. ;

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3725</td>
</tr>
<tr>
<td>2</td>
<td>3725</td>
</tr>
<tr>
<td>3</td>
<td>3725</td>
</tr>
<tr>
<td>4</td>
<td>3725</td>
</tr>
<tr>
<td>5</td>
<td>3725</td>
</tr>
<tr>
<td>6</td>
<td>3725</td>
</tr>
<tr>
<td>7</td>
<td>3725</td>
</tr>
<tr>
<td>8</td>
<td>3725</td>
</tr>
<tr>
<td>9</td>
<td>3725</td>
</tr>
<tr>
<td>10</td>
<td>3725</td>
</tr>
</tbody>
</table>

Mean 3725
s 0

= 62.61 seconds
The Armadillo ColorForth 2.0 Program:

BLOCK NUMBER 3

( ARMAIDILLO ADDING LOOPS BENCHMARK - 1/1 )
0 VARIABLE AVAR : U. 0 <# #S #> TYPE SPACE ;
0 VARIABLE TIMERVALUE
: DOADDS ( -- )
  165 0 DO
    1000 0 DO
      5 7 + AVAR !
    LOOP
  LOOP ;
: RUN ( -- )
  0 274 ! ( ZERO THE COCO TIMER )
  DOADDS
  274 @ TIMERVALUE ! ( GET THE COCO TIMER VALUE )
  ." TIMER = TIMERVALUE @ U. ;

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2538</td>
</tr>
<tr>
<td>2</td>
<td>2538</td>
</tr>
<tr>
<td>3</td>
<td>2538</td>
</tr>
<tr>
<td>4</td>
<td>2537</td>
</tr>
<tr>
<td>5</td>
<td>2538</td>
</tr>
<tr>
<td>6</td>
<td>2538</td>
</tr>
<tr>
<td>7</td>
<td>2537</td>
</tr>
<tr>
<td>8</td>
<td>2538</td>
</tr>
<tr>
<td>9</td>
<td>2538</td>
</tr>
<tr>
<td>10</td>
<td>2538</td>
</tr>
</tbody>
</table>

Mean 2537.8
s 0.42164
= 42.66 seconds
The pd10 SuperForth Program:

( ADDLOOP1.DAT )
( PD-10 SUPERFORTH - 1/1 )
( ADDING LOOPS BENCHMARK )
( MDJ 2019-01-26 )

( NOTE: FIG REQUIRES NUMBER BEFORE VARIABLE )

( NOTE: N1 N2 DO LOOP RUNS UP THROUGH N1 )
( INSTEAD OF JUST UP TO IT. )

0 VARIABLE AVAR
0 VARIABLE TIMERVALUE

: DOINNER 999 0 DO 5 7 + AVAR ! LOOP ;
: DOADDS 164 0 DO DOINNER LOOP ;

: RPT ." TIMER = " TIMERVALUE @ U. ;
: RUN 0 274 ! DOADDS 274 @ TIMERVALUE ! RPT ;

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4142</td>
</tr>
<tr>
<td>2</td>
<td>4142</td>
</tr>
<tr>
<td>3</td>
<td>4142</td>
</tr>
<tr>
<td>4</td>
<td>4142</td>
</tr>
<tr>
<td>5</td>
<td>4142</td>
</tr>
<tr>
<td>6</td>
<td>4142</td>
</tr>
<tr>
<td>7</td>
<td>4142</td>
</tr>
<tr>
<td>8</td>
<td>4142</td>
</tr>
<tr>
<td>9</td>
<td>4142</td>
</tr>
<tr>
<td>10</td>
<td>4142</td>
</tr>
</tbody>
</table>

Mean 4142
s 0

= 69.62 seconds
The Talbot ColorForth Program:

This listing has been modified to eliminate trailing blank lines.

```
SCR 1
0 ( ADDLP.BIN )
1 ( TALBOT COLORFORTH 1.1 )
2 ( ADDING LOOP BENCHMARK )
3 ( MDJ 2019-01-26 )
4 : U. 0 <# #S #> TYPE SPACE ;
5 0 VARIABLE AVAR
6 0 VARIABLE TIMVALUE
7 : DOADDS ( -- )
8   165 0 DO
9     1000 0 DO
10      5 7 + AVAR !
11   LOOP
12   LOOP ;
13 : RUN ( -- )
14   ( ZERO THE COCO TIMER )
15 0 274 !
16   DOADDS
17   ( GET COCO TIMER VALUE )
18 274 @ TIMVALUE !
19   ." TIMER = 
20 TIMVALUE @ U. ;
```

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2827</td>
</tr>
<tr>
<td>2</td>
<td>2827</td>
</tr>
<tr>
<td>3</td>
<td>2827</td>
</tr>
<tr>
<td>4</td>
<td>2827</td>
</tr>
<tr>
<td>5</td>
<td>2827</td>
</tr>
<tr>
<td>6</td>
<td>2827</td>
</tr>
<tr>
<td>7</td>
<td>2827</td>
</tr>
<tr>
<td>8</td>
<td>2827</td>
</tr>
<tr>
<td>9</td>
<td>2827</td>
</tr>
<tr>
<td>10</td>
<td>2827</td>
</tr>
</tbody>
</table>

Mean  2827
s     0

= 47.52 seconds
The eForth Program:

This eForth printout was manually massaged a bit - but just to omit erroneous 23jan84 date and the blank lines at the end of the block.

Block # 5
0 ( eForth Adding Loops Benchmark - 1/1 )
1 variable aVar
2 variable timerValue
3 : doAdds ( -- )
4 165 0 do
5 1000 0 do
6 5 7 + aVar !
7 loop
8 loop ;
9 : run ( -- )
10 bell ( Signal user to start the stopwatch )
11 doAdds
12 bell ( Signal user to stop the stopwatch )
13." Done " ;

<table>
<thead>
<tr>
<th>Run</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34.78</td>
</tr>
<tr>
<td>2</td>
<td>34.67</td>
</tr>
<tr>
<td>3</td>
<td>34.67</td>
</tr>
<tr>
<td>4</td>
<td>34.55</td>
</tr>
<tr>
<td>5</td>
<td>34.57</td>
</tr>
<tr>
<td>6</td>
<td>34.64</td>
</tr>
<tr>
<td>7</td>
<td>34.64</td>
</tr>
<tr>
<td>8</td>
<td>34.63</td>
</tr>
<tr>
<td>9</td>
<td>34.60</td>
</tr>
<tr>
<td>10</td>
<td>34.45</td>
</tr>
</tbody>
</table>

Mean 34.620
s 0.087100

Say 34.62 seconds
The Add Loops Recap:

<table>
<thead>
<tr>
<th>Language</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Language</td>
<td>5.45 sec</td>
</tr>
<tr>
<td>eForth</td>
<td>34.62 sec</td>
</tr>
<tr>
<td>Talbot ColorForth 1.1</td>
<td>47.52 sec</td>
</tr>
<tr>
<td>Armadillo ColorForth 2.0</td>
<td>42.66 sec</td>
</tr>
<tr>
<td>CF83 Forth</td>
<td>62.61 sec</td>
</tr>
<tr>
<td>pd10 SuperForth</td>
<td>69.62 sec</td>
</tr>
<tr>
<td>Basic</td>
<td>1088.46 sec</td>
</tr>
</tbody>
</table>
Appendix G  -- Print Loops Benchmarks

Our Print Loops Benchmark simply prints the message “PRINTING LOOPS BENCHMARK” 2,000 times. The number 2,000 was chosen because, in CF83 Forth, the timer values obtained approached the timer limit of 65535. Thus the timer would not roll over during the CF83 Forth runs and, simultaneously, the other runs would enjoy the greatest possible precision of results within the limit imposed by the CF83 Forth runs.

The Basic Program:

```basic
100 '**********
110 '* PRTLOOPS.BAS
120 '* PRINTING LOOPS BENCHMARK
130 '* MDJ 2019/01/31
140 '**********
150 'ZERO THE COCO TIMER
160 POKE 274,0
170 POKE 275,0
180 'DO THE LOOPS
190 FOR I=1 TO 63
200 FOR J=1 TO 1000
210 PRINT "PRINTING LOOPS BENCHMARK ";
220 NEXT J
230 NEXT I
240 'GET THE COCO TIMER VALUE
250 T1=PEEK(274)
260 T2=PEEK(275)
270 T=(T1*256)+T2
280 'REPORT THE RESULTS
290 PRINT"TIMER = ";T
300 END
```
<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2062</td>
</tr>
<tr>
<td>2</td>
<td>2064</td>
</tr>
<tr>
<td>3</td>
<td>2062</td>
</tr>
<tr>
<td>4</td>
<td>2055</td>
</tr>
<tr>
<td>5</td>
<td>2066</td>
</tr>
<tr>
<td>6</td>
<td>2064</td>
</tr>
<tr>
<td>7</td>
<td>2058</td>
</tr>
<tr>
<td>8</td>
<td>2062</td>
</tr>
<tr>
<td>9</td>
<td>2058</td>
</tr>
<tr>
<td>10</td>
<td>2067</td>
</tr>
</tbody>
</table>

Mean 2061.8  
   s 3.7947  
   = 34.66 seconds
The Assembly Language Program without the assembly:

00100 **********
00110 * PRTLOOP.ASM
00120 * PRINTING LOOPS BENCHMARK
00130 * MDJ 2019/01/31
00140 **********
00150         ORG     $3200
00160         PSHS    A,B,X,Y
00170         JMP     GP
00180 MSG     FCC     'PRINTING LOOPS BENCHMARK '
00190         FCB     $00
00200 GP      LDX     #1      OUTER LOOP COUNTER
00210 GP1     CMPX    #64
00220         BEQ     GP6     EXIT IF OUTER LOOP COMPLETE
00230         LDY     #1      INNER LOOP COUNTER
00240 GP2     CMPY    #1001
00250         BEQ     GP5     GO IF INNER LOOP COMPLETE
00260         PSHS    X
00270         LDX     #MSG    START OF THE MESSAGE
00280 GP3     LDA     ,X+     LOAD CHARACTER
00290         BEQ     GP4     GO IF ZERO ( ==> END )
00300         JSR     $A30A   PUT CHARACTER TO SCREEN
00310         BRA     GP3
00320 GP4     PULS    X
00330         LEAY    1,Y     INCREMENT INNER LOOP COUNTER
00340         BRA     GP2
00350 GP5     LEAX    1,X     INCREMENT OUTER LOOP COUNTER
00360         BRA     GP1
00370 GP6     PULS    A,B,X,Y
00380         RTS
00390         END

100 '**********
110 '* PRTLOOP.BAS
120 '* BASIC SUPERVISOR FOR
130 '* PRTLOOP.ASM
140 '* PRINTING LOOPS BENCHMARK
150 '* MDJ 2019/01/31
160 '**********
170 CLEAR 1024, &H31FF
180 LOADM "PRTLOOP.BIN"
190 'ZERO THE COCO TIMER
200 POKE 274, 0
210 POKE 275, 0
220 'PERFORM THE PRINTING LOOPS
230 EXEC &H3200
240 'GET THE COCO TIMER VALUE
250 T1 = PEEK(274)
260 T2 = PEEK(275)
270 T = (T1 * 256) + T2
280 'REPORT THE RESULTS
290 PRINT "TIMER = ";
300 PRINT T
310 END
The Assembly Language Program without the assembly:

```
00100 **********
00110 * PRTLOOP.ASM
00120 * PRINTING LOOPS BENCHMARK
00130 * MDJ 2019/01/31
00140 **********
3200               00150         ORG     $3200
3200 34   36       00160         PSHS    A,B,X,Y
3202 7E   321F     00170         JMP     GP
3205      50       00180 MSG     FCC               52
49
4E
54
49
4E
47
20
4C
4F
4F
50
53
20
42
45
4E
43
48
4D
41
52
4B
20
321E  00   00190     FCB     $00
321F  8E  0001     00200 GP     LDX     #1
3222  8C  0040     00210 GP1    CMPX    #64
3225  27  22       00220     BEQ    GP6
3227 108E 0001    00230     LDY     #1
322B 108C 03E9    00240 GP2    CMPY    #1001
322F  27  14       00250     BEQ    GP5
3231  34  10       00260     PSHS    X
3233  8E  3205    00270     LDX    #MSG
3236 A6  80       00280 GP3    LDA    ,X+
3238  27  05       00290     BEQ    GP4
323A BD  A30A    00300     JSR    $A30A
323D  20  F7       00310     BRA    GP3
```
00000 TOTAL ERRORS

GP      321F
GP1     3222
GP2     322B
GP3     3236
GP4     323F
GP5     3245
GP6     3249
MSG     3205

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>968</td>
</tr>
<tr>
<td>2</td>
<td>971</td>
</tr>
<tr>
<td>3</td>
<td>970</td>
</tr>
<tr>
<td>4</td>
<td>971</td>
</tr>
<tr>
<td>5</td>
<td>970</td>
</tr>
<tr>
<td>6</td>
<td>971</td>
</tr>
<tr>
<td>7</td>
<td>970</td>
</tr>
<tr>
<td>8</td>
<td>970</td>
</tr>
<tr>
<td>9</td>
<td>970</td>
</tr>
<tr>
<td>10</td>
<td>970</td>
</tr>
</tbody>
</table>

Mean 970.1
s 0.8756

= 16.31 seconds
The CF83 Forth Program:

BLOCK NUMBER 4

( CF83 Printing Loops Benchmark - 1/1 )
variable timerValue
: doPrints ( -- )
   2 0 do
     1000 0 do
       ." PRINTING LOOPS BENCHMARK "
       loop
     loop ;
: run ( -- )
   0 274 ! ( Zero the CoCo Timer )
   doPrints
   274 @ timerValue ! ( Get the CoCo Timer Value )
   ." Timer = " timerValue @ u. ;

Run   Timer
  1   59712
  2   59702
  3   59928
  4   59928
  5   59928
  6   59928
  7   59927
  8   59928
  9   59928
 10   59928

Mean  59883.7
   s    93.1594

=  1006.55 seconds
=  16 minutes 46.55 seconds
The Armadillo ColorForth 2.0 Program:

**BLOCK NUMBER 6**

```forth
( ARMADILLO PRINTING LOOPS BENCHMARK - 1/1 )
: U. 0 <# #S #> TYPE SPACE ;
0 VARIABLE TIMERVALUE
: DOPRINTS ( -- )
  2 0 DO
    1000 0 DO
       ." PRINTING LOOPS BENCHMARK "
       LOOP
    LOOP ;
: RUN ( -- )
  0 274 ! ( ZERO THE COCO TIMER )
  DOPRINTS
  274 @ TIMERVALUE ! ( GET THE COCO TIMER VALUE )
  ." TIMER = " TIMERVALUE @ U. ;

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2548</td>
</tr>
<tr>
<td>2</td>
<td>2552</td>
</tr>
<tr>
<td>3</td>
<td>2551</td>
</tr>
<tr>
<td>4</td>
<td>2551</td>
</tr>
<tr>
<td>5</td>
<td>2552</td>
</tr>
<tr>
<td>6</td>
<td>2551</td>
</tr>
<tr>
<td>7</td>
<td>2552</td>
</tr>
<tr>
<td>8</td>
<td>2552</td>
</tr>
<tr>
<td>9</td>
<td>2551</td>
</tr>
<tr>
<td>10</td>
<td>2552</td>
</tr>
</tbody>
</table>

Mean 2551.2
s      1.2293

= 42.88 seconds
```
The pd10 SuperForth Program:

(PRTLOOP1.DAT )
(PD-10 SUPERFORTH - 1/1 )
(PRINTING LOOPS BENCHMARK )
(MDJ 2019-03-24 )

(NOTE: FIG REQUIRES NUMBER BEFORE VARIABLE )

(NOTE: N1 N2 DO LOOP RUNS UP THROUGH N1 )
( INSTEAD OF JUST UP TO IT. )

0 VARIABLE TIMERVALUE

: DOINNER 999 0 DO ." PRINTING LOOPS BENCHMARK " LOOP ;
: DOPRTS 1 0 DO DOINNER LOOP ;

: RPT ." TIMER = " TIMERVALUE @ U. ;
: RUN 0 274 ! DOPRTS 274 @ TIMERVALUE ! RPT ;

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1500</td>
</tr>
<tr>
<td>2</td>
<td>1500</td>
</tr>
<tr>
<td>3</td>
<td>1500</td>
</tr>
<tr>
<td>4</td>
<td>1500</td>
</tr>
<tr>
<td>5</td>
<td>1500</td>
</tr>
<tr>
<td>6</td>
<td>1500</td>
</tr>
<tr>
<td>7</td>
<td>1500</td>
</tr>
<tr>
<td>8</td>
<td>1500</td>
</tr>
<tr>
<td>9</td>
<td>1500</td>
</tr>
<tr>
<td>10</td>
<td>1500</td>
</tr>
</tbody>
</table>

Mean 1500
s 0

= 25.21 seconds
The Talbot ColorForth Program:

This listing has been modified to eliminate trailing blank lines.

```
SCR 1
  0 ( PRTL.P.BIN )
  1 ( TALBOT COLORFORTH 1.1 )
  2 ( PRINTING LOOP BENCHMARK )
  3 ( MDJ 2019-02-01 )
  4 : U. 0 <# #S #> TYPE SPACE ;
  5
  6 0 VARIABLE TIMERVVALUE
  7 : DOPRINTS ( -- )
  8     2 0 DO
  9       1000 0 DO
 10   ." PRINTING LOOPS BENCHMARK "
 11     LOOP
 12     LOOP ;
 13 : RUN ( -- )
 14   ( ZERO THE COCO TIMER )
 15     0 274 !
 16   DOPRINTS
 17   ( GET COCO TIMER VALUE )
 18     274 @ TIMERVVALUE !
 19   ." TIMER = "
 20     TIMERVVALUE @ U. ;
```

<table>
<thead>
<tr>
<th>Run</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3582</td>
</tr>
<tr>
<td>2</td>
<td>3582</td>
</tr>
<tr>
<td>3</td>
<td>3583</td>
</tr>
<tr>
<td>4</td>
<td>3583</td>
</tr>
<tr>
<td>5</td>
<td>3583</td>
</tr>
<tr>
<td>6</td>
<td>3583</td>
</tr>
<tr>
<td>7</td>
<td>3583</td>
</tr>
<tr>
<td>8</td>
<td>3583</td>
</tr>
<tr>
<td>9</td>
<td>3583</td>
</tr>
<tr>
<td>10</td>
<td>3583</td>
</tr>
</tbody>
</table>

Mean 3582.8
s 0.42164

= 60.22 seconds
The eForth Program:

This eForth printout was manually massaged a bit - but just to omit erroneous 23jan84 date and the blank lines at the end of the block.

Block # 6
0 ( eForth Printing Loops Benchmark - 1/1 )
1 : doPrints ( -- )
2  2 0 do
3    1000 0 do
4       ." PRINTING LOOPS BENCHMARK "
5     loop
6   loop ;
7 : run ( -- )
8    bell ( Signal user to start the stopwatch )
9    doPrints
10   bell ( Signal user to stop the stopwatch )
11  "." Done " ;

Run     Min:Sec

<table>
<thead>
<tr>
<th>Run</th>
<th>Min:Sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2:48.49</td>
</tr>
<tr>
<td>2</td>
<td>2:48.27</td>
</tr>
<tr>
<td>3</td>
<td>2:48.30</td>
</tr>
<tr>
<td>4</td>
<td>2:48.17</td>
</tr>
<tr>
<td>5</td>
<td>2:48.19</td>
</tr>
<tr>
<td>6</td>
<td>2:48.18</td>
</tr>
<tr>
<td>7</td>
<td>2:48.20</td>
</tr>
<tr>
<td>8</td>
<td>2:48.14</td>
</tr>
<tr>
<td>9</td>
<td>2:48.25</td>
</tr>
<tr>
<td>10</td>
<td>2:48.15</td>
</tr>
</tbody>
</table>

Mean  2:48.234
s      0.10405

Say    168.23 seconds
The Print Loops Recap:

<table>
<thead>
<tr>
<th>Language</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Language</td>
<td>16.31 s</td>
</tr>
<tr>
<td>pd10 SuperForth</td>
<td>25.21 s</td>
</tr>
<tr>
<td>Basic</td>
<td>34.66 s</td>
</tr>
<tr>
<td>Armadillo ColorForth 2.0</td>
<td>42.88 s</td>
</tr>
<tr>
<td>Talbot ColorForth 1.1</td>
<td>60.22 s</td>
</tr>
<tr>
<td>eForth</td>
<td>168.23 s</td>
</tr>
<tr>
<td>CF83 Forth</td>
<td>1006.55 s</td>
</tr>
</tbody>
</table>
Appendix H -- New BDS Software License

This New Software License applies to all software found on the BDS Software site, and supersedes all previous copyright notices and licensing provisions which may appear in the software itself or in any documentation therefor.

All software which has previously been placed in the public domain remains in the public domain.

All other software, programs, experiments and reports, documentation, and any other material on this site (other than that attributed to outside sources) is hereby copyright © 2018 (or later if so marked) by M. David Johnson.

All software, documentation, and other information on the BDS Software site is available for you to freely download without cost.

Whether you downloaded such items directly from this site, or you obtained them by any other means, you are hereby licensed to copy them, to sell or give away such copies, to use them, and to excerpt from them, in any way whatsoever, so long as nothing you do with them would denigrate the name of our Lord and Savior, Jesus Christ.

I make absolutely no warranty whatsoever for any of these items. You use them entirely at your own risk.

If they don't work for you, I commiserate.

If they crash your system, I sympathize.

But I accept no responsibility whatsoever for any such consequences. Under no circumstances will BDS Software or M. David Johnson be liable for any negative results of any kind which you may experience from downloading or using these items.

BDS Software's former mail address at P.O. Box 485 in Glenview, IL is no longer valid. Any mail sent to that address will be rejected by the U.S. Postal Service. See my Contact page.

M.D.J. 2018/06/08
Appendix I -- References

(Calculator.net). [https://www.calculator.net/standard-deviation-calculator.html](https://www.calculator.net/standard-deviation-calculator.html)


Unknown Author (Unknown Date). *PD-10 SUPERFORTH MANUAL*. Unknown online publisher.

