The Tali Forth 2 Manual

Scot W. Stevenson

2. March 2018

Abstract

Tali Forth 2 is a bare-metal ANSI(ish) Forth for the 65c02 8-bit MPU.

Contents

List of Tables

Part I

Introduction

Chapter 1

Overview

1.1 A (very) brief introduction to Forth

Part II

Using Tali Forth

Chapter 2

Installation

2.1 Downloading

Tali Forth was created to be easy to get started with. In fact, all you should need is the `ophis.bin` binary file and the `py65mon` simulator.

2.1.1 Downloading Tali Forth

Tali Forth 2 lives on GitHub at **FEHLT**.

2.1.2 Downloading the py65mon Simulator

Tali comes with an assembled version that should run out of the box with the `py65mon` simulator from https://github.com/mnaberez/py65. This is a Python program that should run on various operating systems.

To install py65mon with Linux, use the command `sudo pip install -U py65`. If you don't have PIP installed, you will have to add it first with `sudo apt-get install python-pip`. There is a `setup.py` script as part of the package, too.

2.2 Running the binary

To start the emulator, run: `py65mon -m 65c02 -r ophis.bin`.

Chapter 3

Running Tali Forth

3.1 Native compiling

In a pure subroutine-threaded Forth, higher-level words are merely a series of subroutine jumps. For instances, the Forth word `[char]`, formal Forth definition

```
: [char] char postpone literal ; immediate
```

in assembler is simply

```
                jsr xt_char
                jsr xt_literal
```

as an immediate, compile-only word. Theare are two obvious problems with this method: First, it is slow, because each `jsr/rts` pair consumes four bytes and 12 cycles overhead. Second, for smaller words, it uses far more bytes. Take for instance `drop`, which in its naive form is simply

```
                inx
                inx
```

for two bytes and four cycles. The jump to `drop` uses more space and takes far longer than the word itself. (In practice, `drop` checks for underflow, so the actual assembler code is

```
                cpx #dsp0-3
                bmi +
                lda #11         ; error code for underflow
                jmp error
*
                inx
                inx
```

for eleven bytes. We'll discuss the underflow check further below.)

To get rid of this problem, Tali Forth supports **native compiling**. The system variable `nc-limit` sets the threshhold up to which a word will be included not as a subroutine jump, but machine language. Let's start with an example where `nc-limit` is set to zero, that is, all words are compiled as subroutine jumps. Take a simple word such as

```
        : aaa 0 drop ;
```

and check the actual code with SEE:

```
        see aaa
          nt: 7CD  xt: 7D8
         size (decimal): 6

        07D8  20 52 99 20 6B 88  ok
```

(The actual addresses might be different, this is from the ALPHA release). Our word `aaa` consists of two subroutine jumps, one to zero and one to DROP. Now, if we increase the threshhold to 20, we get different code, as this console session shows:

```
20 nc-limit !  ok
: bbb 0 drop ;  ok
see bbb
  nt: 7DF  xt: 7EA
 size (decimal): 17

07EA  CA CA 74 00 74 01 E0 77  30 05 A9 0B 4C C7 AC E8
07FA  E8  ok
```

Even though the definition of `bbb` is the same as `aaa`, we have totally different code: The number 0001 is pushed to the Data Stack (the first six bytes), then we check for underflow (the next nine bytes), and finally we `drop` by moving X, the Data Stack Pointer. Our word is definitely longer, but have just saved 12 cycles.

To experiment with various parameters for native compiling, the Forth word `words&sizes` is included in `user_words.fs` (but commented out by default). The Forth is:

```
: words&sizes ( --)
        latestnt
        begin
                dup
        0<> while
                dup name>string type space
                dup wordsize u. cr
                2 + @
        repeat
        drop ;
```

Changing `nc-limit` should show differences in the Forth words.

3.2 Underflow stripping

Checking for underflow helps during the design and debug phases of writing Forth code, but once it ready to ship, those nine bytes per check hurt, as we see in the case above. To allow those checks to be stripped, we can set the system variable `uf-strip` to `true`.

3.3 Gotchas

Tali has a 16-bit cell size (use `1 cells 8 * .` to get the cells size in bits with any Forth), which can trip up calculations when compared to the *de facto* standard Gforth with 64 bits. Take this example:

```
( Gforth )      decimal 1000 100 um* hex swap u. u.  186a0 0  ok
( Tali Forth)   decimal 1000 100 um* hex swap u. u.  86a0 1  ok
```

Tali has to use the upper cell of a double-celled number to correctly report the result, while Gforth doesn't. If the conversion from double to single is only via a `drop` instruction, this will produce different results.

3.4 Reporting a problem

Chapter 4

The Editor

(Currently, there is no editor installed.)

Part III

The Internals

Chapter 5

How Tali Forth works

Chapter 6

Developing

6.1 Adding new words

The easiest way to add new words to Tali Forth is to include them in the file `forth_code/user_words.fs`.

6.2 Deeper changes

Tali Forth was not only placed in the public domain to honor the tradition of giving the code away freely. It is also to let people play around with it and adapt it to their own machines. This is also the reason it is (perversely) overcommented.

To work on the internals of Tali Forth, you will need the Ophis assembler.

6.2.1 The Ophis Assembler

Michael Martin's Ophis Cross-Assember can be downloaded from http://michaelcmartin.github.io/Ophis/. It uses a slightly different format than other assemblers, but is in Python and therefore will run on almost any operating system. To install Ophis on Windows, use the link provided above. For Linux:

```
git clone https://github.com/michaelcmartin/Ophis
cd src
sudo python setup.py install
```

Switch to the folder where the Tali code lives, and assemble with the primitive shell script provided: `./assemble.sh` The script also automatically updates the file listings in the `docs` folder. Note that Ophis will not accept math operation characters in label names because it will try to perform those operations. Because of this, we use underscores for label names. This is a major difference to Liara Forth.

6.3 General notes

- The X register should not be changed without saving its pointer status.
- The Y register is free to be changed by subroutines. This means it should not be expected to survive subroutines unchanged.
- All words should have one point of entry – the `xt_word` link – and one point of exit at `z_word`. In may cases, this means a branch to an internal label `done` right before `z_word`.
- Because of the way native compiling works, the usual trick of combining JSR/RTS pairs to a single JMP (usually) doesn't work.

6.4 Coding style

Until I get around to writing a tool for Ophis assembler code that formats the source file the way gofmt does for Go (golang), I work with the following rules:

- Actual opcodes are indented by **two tabs**
- Tabs are **eight characters long** and converted to spaces
- Function-like routines are followed by a one-tab indented 'function doc' based on the Python 3 model: Three quotation marks at the start, three at the end it its own line, unless it is a one-liner. This should make it easier to automatically extract the docs for them at some point.
- The native words have a special commentary format that allows the automatic generation of word list by a tool in the tools folder, see there for details.
- Assembler mnenomics are lower case. I get enough uppercase insanity writing German, thank you very much.
- Hex numbers are also lower case, such as `$FFFE`
- Numbers in mnemonics are a stripped-down as possible to reduce visual clutter: `lda 0,x` instead of `lda $00,x`.
- Comments are included like popcorn to help readers who are new both to Forth and 6502 assembler.

6.5 Testing

There is no automatic or formal test suite available at this time, and due to space considerations, there probably never will be. The file `docs/testwords.md` includes a collection of words that will help with some general cases.

Appendix A

FAQ

A.1 Why does Tali Forth take so long to start up?

After the default kernel string is printed, you'll notice a short pause that didn't occur with Tali Forth 1. This is because Tali Forth 2 has more words defined in high-level Forth (see `forth-words.asm`) than Tali did. The pause happens because they are being compiled on the fly.

A.2 Why 'Tali' Forth?

I like the name, and we're probably not going to have anymore kids I can give it to.

(If it sounds vaguely familiar, you're probably thinking of Tali'Zorah vas Normandy, a character in the 'Mass Effect' universe created by EA / BioWare. This software has absolutely nothing to do with either the game or the companies and neither do I, expect that I've played the games and enjoyed them, though I do have some issues with *Andromeda*. Like what happened to the quarian ark?)

A.3 Then who is 'Lara'?

Liara Forth is a STC Forth for the big sibling of the 6502, the 65816. Tali 1 came first, then I wrote Liara with that knowledge and learned even more, and now Tali 2 is such much better for the experience. Oh, and it's another 'Mass Effect' character.

Appendix B

Thanks

Tali Forth 2 would not have been possible without the kind support of a very large number of individuals.

Index