MOVING FORTH

Part 8: CamelForth for the 6809

by Brad Rodriguez

This article first appeared in The Computer Journal #74 (July/August 1995).

Finally, the last installment of "Moving Forth!" Here is the long- promised ANSI CamelForth for the Motorola 6809, and specifically for the Scroungmaster II processor board.

Unlike the Z80 and 8051 CamelForth, the 6809 Forth was produced with my "Chromium 2" Forth metacompiler [ROD92]. Right away you'll notice two things: first, the metacompiler runs on an older Forth (F83), and so the source code is contained in 16x64 Forth "screens". I've converted these to an ASCII file for TCJ, but the original formatting is still evident.

Second, source code for a Forth metacompiler looks like ordinary Forth code (with a few changes, which I'll discuss shortly). Thus the definition of **1+** is given as

```
  CODE 1+   1 # ADDD,   NEXT   ;C
```

The assembler used is the 6809 assembler I've described previously in TCJ [ROD91].

I typed the high-level source code directly from the already-published listings (converting to the Forth syntax in the process). Unfortunately, this was done over the space of a few days, and sometimes I worked from the Z80 listing, and sometimes the 8051...with the result that the Harvard-architecture constructs (such as I@ and IALLOT) are not consistently used in the 6809 code. This is of no consequence for the non-Harvard 6809, but I'll have to correct this before porting the Forth code to a Harvard CPU.

Also, since I was working from published listings, I often neglected typing the detailed comments for the high-level words. For this I apologize. You can find how any word works by consulting the previous listings, but I shouldn't force you to do this.

6809 CAMELFORTH SOURCE CODE

The 6809 CamelForth model holds top-of-stack in D, and uses the S stack pointer for the Parameter Stack. The U stack pointer is the Return Stack Pointer, and Y is the Interpreter Pointer. X is the temporary register "W". The 6809 direct page pointer DPR holds the high byte of the User Pointer (the low byte is assumed to be zero).

The memory map for a Scroungemaster II with 8K of RAM and 8K of EPROM is as follows:

```
 6000-797Fh RAM dictionary (for new definitions)
 7980-79FFh Terminal Input Buffer
 7A00-7A7Fh User Area (USER variables)
 7A80-7AFFh Parameter Stack (grows downward)
 7B00-7B27h HOLD area (grows downward)
 7B28-7B7Fh PAD area (general purpose buffer)
 7B80-7BFFh Return Stack (grows downward)

 E000-FFFFh Forth kernel in EPROM
```

All of the RAM data areas are referenced to the User Pointer, whose starting value is given by UP-INIT: in this case, 7A00h. (Note the use of UP-INIT-HI for the high byte of this value.) When CamelForth starts, it will set its Dictionary Pointer to DP-INIT, which must be in RAM so you can add new definitions to the Forth dictionary. These are all specified with the metacompiler's EQU directive. An EQU is like a CONSTANT, except that it is *only* known to the metacompiler. These EQUates take up no space in the 6809 kernel, and will not appear in the 6809 Forth's dictionary.

DICTIONARY tells the metacompiler where to compile the code, in this case for an 8K EPROM from E000-FFFFh. The new dictionary is named "ROM", and then ROM is specified to select that dictionary. (If you're familiar with Forth vocabularies, you'll see a strong resemblance.)

AKA ("also known as") defines a synonym for a Forth word. Since the 6809 is a non-Harvard machine, we should compile @ wherever I@ appears in the source code, and likewise for the other "I-prefix" (instruction-space) words. AKA will do this.

These synonyms are like EQUates -- they don't appear in the 6809 dictionary.

The metacompiler allows you to use forward references, i.e., Forth words which haven't been defined yet. (You must of course define them before you finish!) Often this is automatic, but AKA requires you to explicitly declare a forward reference with PRESUME. Thus

```
PRESUME WORD   AKA WORD IWORD
```

is needed to create the IWORD synonym. @ ! HERE ALLOT and the others are PRESUMEd by the metacompiler, so we don't have to do so here.

The CODE definitions are conventional. Note that you can use

```
HERE EQU labelname
```

to generate a label when metacompiling. (This is a function of the metacompiler, not the assembler.) Also, ASM: begins a "fragment" of assembler code (i.e., not part of a CODE word).

The phrase

```
HERE RESOLVES name
```

is used to resolve certain forward references which are made by the metacompiler (for example, the metacompiler has to know where the code for the DOCOLON action is). You should leave these alone. Otherwise, feel free to add any CODE definitions to the source code.

The code for defining words and control structures (IMMEDIATE words) is rather opaque. This is because these words *must also perform some action while metacompiling.* For example: the 6809 Forth includes the standard word CONSTANT, to define new constants. But CONSTANTs may also appear in the 6809 kernel; we may have to define a CONSTANT *while metacompiling*. The EMULATE: phrase instructs the metacompiler how to handle the word CONSTANT if it is encountered. This phrase is written entirely using metacompiler words, and so may appear to be total gibberish.

Likewise, IF THEN and their ilk include the metacompiler phrases to build and resolve branches in the 6809 image. Some Forth metacompilers bury this code inside the compiler. This makes for prettier target code, but if you change the way branches work (for example), you have to perform surgery on the metacompiler. I preferred to make these actions easily changeable, and so I designed Chromium to put them in the target source code. (The most horrific examples are the definitions of TENDLOOP and TS", which actually extend the metacompiler vocabulary in the middle of the target source code.)

If you're new to Forth and the metacompiler, it's best to just accept these as given. "Ordinary" colon definitions are easy to add. Just follow the example of the rest of the 6809 source code. You can even make CREATE..DOES> definitions, as long as you don't need to use them within the metacompiler.

FUTURE WORK

On a 1 MHz 6809, a line of text input takes a noticeable time to process (up to 1 second at a rough estimate). This is partly becuase so much of the interpreter is written in high-level Forth, and partly because CamelForth uses a single-linked-list dictionary. These handicaps only affect *compilation* speed, not execution speed, but the delays can be annoying. Maybe someday I'll do an article on "Accelerating Forth".

Currently, the User Pointer never changes. The reason we have a User Pointer is to support multitasking -- each task having separate user area, stacks, etc. I'll be working on this soon. I may also explore using the SM II's memory management to give each task a full 32K private dictionary. And of course, I intend to write a true *multiprocessor* Forth kernel using the shared bus. If I live long enough, a *distributed* Forth kernel using the serial ports (a la Transputer) is the logical next step.

The source code for 6809 CamelForth, version 1.0, is available on GEnie's Forth Roundtable in the file CAM09-10.ZIP. This file includes the Chromium 2 metacompiler, complete and ready to run. You'll need a copy of F83. Then you merely type

```
F83 CHROMIUM.SCR
1 LOAD
BYE
```

This will load the metacompiler, compile the 6809 CamelForth, and write the result to an Intel hex file 6809.HEX. Note: if you're using the CP/M or Atari ST versions of F83, you'll have to edit the load screen to delete the hex file utility, since this only

works under MS- DOS. I haven't yet tested Chromium 2 with CP/M or Atari ST, so if you need assistance, please contact me.

Which reminds me: I have a *new email address!* You can now reach me as bj@genie.com, or just BJ if you're a GEnie user. It's a lot easier to type. *[Note for web publication: my current email address is here.]*

ERRATA

There were some errors in the Harvard memory access in CamelForth/8051. The corrected file is on GEnie as CAM51-11.ZIP. I've also uploaded the current Z80 CamelForth, CAM80-12.ZIP, which incorporates all the fixes which have been published in TCJ.

REFERENCES

[ROD91] Rodriguez, B. J., "B.Y.O. Assembler," The Computer Journal #52 (Sep/Oct 1991) and #54 (Jan/Feb 1992).

[ROD92] Rodriguez, B. J., "Principles of Metacompilation," Forth Dimensions XIV:3 (Sep/Oct 1992), XIV:4 (Nov/Dec 1992), and XIV:5 (Jan/Feb 1993). Describes the "Chromium 1" metacompiler.

Source code for 6809 CamelForth is available on this site at http://www.camelforth.com/public_ftp/cam09-10.zip.

Return to publications page