What Can Be Done With Meta-Compilers?

by Rieks Joosten
Pijnenburg Software Development B.V.
P.O. Box 82
5270 AB St. Michielsgestel
The Netherlands

Abstract

A Forth system has been modified and extended so that it became a tool with which compilers can be built; these unite the powers of classical compilers with the characteristic development facilities found in Forth systems. This Meta-tool (Meta as in Meta-Physics) not only allows for building powerful compilers for almost any machine, but also for building an integrated development system that includes debugging and simulation facilities. Also, the variety of tools that may be needed for a number of different target machines may look very much alike so that one can speak of an integrated tool kit, i.e. a complete software development environment that embraces a (rather large) class of machines.

Introduction

One of the major developments of Pijnenburg Software Development (PSD) last year, was the creation of the software for a Hand Held Computer (HHC) project, which was a joint enterprise of Matsushita Electrical Industries (MEI), Japan, and PSD. This project allowed testing of the concepts used in the Meta-tool kit [ROC]. The project consisted of the creation of a Hand Held Computer FH2000 by MEI, while the software was created in mutual cooperation. The goal of the project was to create a powerful machine for financial type purposes, in such a way that application programs serving such purposes will efficiently execute.

The hardware for the FH-2000 Hand Held Computer consisted of a 80C88 microprocessor running at 4.77 MHz, an 80*8 character LCD, full QWERTY keyboard, beeper, clock, 8kB ram, 64kB system rom. Optional hardware additions included extra ram, program roms, RS232, modem, and a printer.

The hardware on which the development tools were running was the IBM pc, supplemented with an (E)PROM burner (required to burn the programs into rom). Although additional hardware was not strictly necessary, utilizing a harddisk enhanced the performance of the development tools.

The development tools for this project were based on PSD's FysForth

'86 Forth system [FYS], which included standard facilities like a textfile editor, copy facilities, file handling, assembler etc. This Forth system is more or less 79 Standard with extensions such as error recovery, safe forgetting, and virtual memory handling. Exceptions to the standard were the mass storage facilities (using files instead of blocks), and the system variables, which apply the TO-concept.

From the HHC project it was learned that facilities such as extended buffer management, symbol tables handling, target dictionary handling, support for forward referencing, and logging needed to be available. These facilities were developed as separate support modules that, together with the modified host Forth system, would form an entity that was called MCS, the Meta Compilation System [ROC]. MCS was the basic tool kit, capable of extending itself in an interactive manner while having available the necessary hooks so that it could be turned into an integrated tool for sophisticated program development.

The Meta Compilation System Extended: Assembler Environments.

MCS would accept source texts that compiled an assembler on top of it, similar as is done in any Forth system. In fact, only an estimated 5-10% of source of a regular FIG-assembler needed to be rewritten to upgrade MCS to a full blown cross assembler with facilities such as the ability to use labels, synonyms, macroes, forward referencing, separated compilation for ROM and RAM, logging, reference counting, etc. Other extensions included debugging facilities (Dumping, Moving of memory blocks, Verifying memory, etc) and simulation of machine code execution (tracing, breakpoints, single stepping, disassembling etc).

In the same way, machine code development environments could be built for different machines. Having split extension modules in target-cpu dependent and independent ones, enabled the re-use of target-cpu independent code. As a consequence commands that would write partially compiled code to file, reading pre-compiled modules from mass storage, dumping of target memory, printing symbolic names and their associated target addresses etc., would have the same syntax and semantics whether they were used in an 8088 or a 6809 development environment. The general idea was that it would be sufficient to only once learn a machine code programming environment, and subsequently only learning different cpu's (mnemonics, registers etc), so that programmers could optimally use their time for the task they were intended to do.

Although such development environments for different machines look very much alike, it has shown to be practical to sometimes add dedicated software tools to the environment, so that specific project related tasks could be performed. Such tools would include downloading, linking, etc. All this resulted in an interactive debugging aid for target code, that had compiler and debugging powers beyond those of the classical dedicated tools.

The Concept: EVOLUTION Of Software

In a similar way as when upgrading MCS to an assembler development environment, the latter was extended with Forth cross compilers, decompilers, high-level tracing, stack dumping etc. so that a complete Forth development environments evolved. However, although extending the development system could be done before the actual target compilation process, these could be combined so that while accumulating object code, the power of the development environment would grow as well. This enabled tailoring the debugging aids to the object binary, and testing the routines as they were compiled. In such a way, the toolkit as well as the compiled code gradually evolved to an optimum.

The development environment was not only used for the creation of the targetted software, but also for building separate, stand-alone debugging aids. As an example, code that completely simulated the target computer was built so that the I/O parts of software could be tested. The simulated computer software was made even before the hardware was available, and it was changed rapidly when appropriate.

Not only was the development environment tailored for software testing at the hardware level, it was also tuned with respect to the user interface. A programmers interface identical with that of a classical tool was put on the environment so that programmers were not exposed to too many differences. As more power was required, the programmers were taught how to use the development environment itself, rather than the limited classical tool interface. In this way, programmers were gradually brought in contact with this different approach to program development.

Performance

The tools described above made it possible to implement many features into the operating system of the HHC in a relatively simple way. Such features included stringent error handling, 18 digit precise floating point (including all scientific functions), alarm- and time-handling, file manipulation, buffer management, hardware configuration and relocation, several I/O layers, exceptional event handling etc.

The development system included complete simulation of the target computer and debugging facilities, both for high level Forth code as well as for machine code (e.g. stepping, tracing, breakpoints, dumping of stacks, compiling, etc.). Arguments for debugger commands could be numeric, symbolic, or calculated from a mixture of those.

A limited number of people both from MEI and PSD have worked approximately 1 year to design and develop the hardware, system software and the development environment.

Timings of the compilation process in the development environment have indicated a compilation speed of approx. 200kB source text per minute (apart from reading/writing to disk and other I/O). This process generated very compact machine- and Forth code.

Conclusions

A complete development environment was built that unified the powers from interactive Forth and classical compilers. The integrated development environment would look similar for different target machines, while it still could be tailored for specific needs in a project or for a programmer. Also, it could be extended both interactively or in batch mode, either separate from actual target compilation (to create a specific tool), or by having it evolve with the target software.

Although the generality of the concepts seem to contradict the fact that a development tool like this can be tuned to a precision instrument, practice showed this not to be the case. Experience with the Meta Compilation System has shown a significant reduction in development and testing time, both for applications and tools.

Acknowledgements and References

I would like to thank Matsushita Electrical Industries (MEI), Japan, and their engineers, for their cooperation and the project in which the ideas of the Metacompilation System could be expanded. Also, I would like to thank Dr. Hans Nieuwenhuyzen for his advice and for his letting us try out ideas on him.

- [ROC] 1985 Rochester Forth Confèrence. The abstract of this paper can be found in the conference proceedings. The article itself is submitted for publication in the Journal of Applied Forth Research (yet to be published).

- [FYS] FysForth '86 User Manual, Mountain View Press, PO Box 4656, Mountain View, CA 94040.

- [EVO] EVOLUTION, Pijnenburg Software Development, P.O.Box 82, 5270 AB St. Michielsgestel, The Netherlands.