On the Need of Compilepretation for Legacy Languages

Vadim Zaytsev
Raincode
vadim@grammarware.net

Abstract

The context of this work is the research and development effort that went into creating a product called Raincode Assembler Compiler (Blagodarov et al. 2016), a reimplementation of the IBM mainframe assembler language retargeted and replatformed for .NET Framework (ECMA-335 June 2012). The assembler language for IBM System/360 and newer mainframes has existed and evolved since 1964 (Abel 1989; Carrano 1987; GC24-3414-7 1970; GC26-4943-06 2013; Kacmar 1988; McQuillen 1975; McQuillen and Prince 1987; O'Kane 2011; SA22-7832-09 2012; SC26-4940-06 2013; SC26-4941-06 2013). The current version is called High Level Assembler, or HLASM, and is widely accepted among mainframe developers and architects as a "second generation programming language", thus positioning it between raw machine code and proper third generation languages like COBOL and PL/I.

HLASM is used for a number of reasons (Blagodarov et al. 2016; Kornelis 2003), which fall roughly into three categories. A **first** classic use of HLASM is to basically write operating system components: you have full access to memory allocation, data handling, interrupts and whatnot. Porting applications like this to anything besides the exact same architecture for which they have been developed, is unhealthy fantasy. **Second**, there can be pure practical reasons of availability: HLASM developers were available while COBOL ones were not; language choice was due to the cost saving strategy for avoiding paying for the 3GL or 4GL compiler; fine-grained tuning of adjacent products was available only as assembler macros, etc. In practice, such programs gradually get rewritten in REXX, CLIST, PL/I or any other available languages since such short-term decisions are rarely profitable in the long run. However, the cost of rewriting *all* of the assembler code existing within a company as *just another subtask* of a bigger migration project is often unbearable—hence, if a company wants to migrate off the mainframe into a different platform, having HLASM sources as a part of their portfolio is one of the common showstoppers. If this is the case for the program at hand, we can imagine writing a compiler or interpreter that will handle it. The **third** family of reasons lies somewhere in between: system elements could have been written in HLASM for optimisation purposes, to reduce program size, running time, memory footprint, implement very sophisticated error recovery strategies, fine-tune memory allocation. On the grand scheme of things this may seem like a detail, but such details are what usually make grand schemes fall apart. In our case, such considerations demanded high efficiency and productivity of our "compiler".

Writing a language processor by the book seems to be straightforward (Grune et al. 2012): you parse the code, you handle contexts and static semantics, you generate some intermediate representation, optimise it, generate code, optimise it and finally interpret it or let it be interpreted by the target machine directly. A preprocessor (Favre 1996) is also dangling somewhere in there, with compiler book authors usually unsure where to put it since conceptually it is a compiler on its own right, but its tight integration into the base compilation chain makes separation challenging. There are many attempts in the middle, including compilation of classically interpreted, very dynamic languages (Rigo and Pedroni 2006), as well as building interactive REPL environments for compiled languages to boost language learnability and developer productivity (Imai et al. 2015).

We were faced with a barricade of challenges which I can go deep into detail during the presentation. In particular,

- **Preprocessing, Macros and Instructions** are so powerful, expressive and versatile that one needs a very good compiler architecture, extensible and non-trivial. As an example, some macros need to be expanded in exactly the same way as on the mainframe because the code they expand to, is being read later as data; while others need to be compiled directly into a call of the .NET API.
- **Self-modification and Code Representation** of memory contents that are manipulated as data and then executed as code, is widespread in HLASM code, which means one needs to have bit-honest representation of all data structures, including those exotic to the target platform (such as zoned decimals). As an example, HLASM contains an `EXECUTE` instruction that can take any bytes in memory, bit-or them with a given mask and then interpret them as code.

MoreVMs'17 April 3, 2017, Brussels, Belgium

- **Performance** is still being an issue at least to some extent, meaning that by-the-book interpretation/emulation is a viable technical solution but does not yield a proper product to satisfy the needs of the industry. As an example, approximately half of the instructions sets specific condition codes signifying overflows, zeroness and some other special circumstances — these need to be optimised to be evaluated lazily since in many cases subsequent instructions do not check them.

The combination of these three challenges makes a perfect implementation impossible — since for that one needs to implement a dead-honest emulator that covers all tiny features of the original, and yet optimise it to make the performance comparable to a compiler to native code. I can name a few design and implementation decisions that allowed us to do it anyway.

Acknowledgments

Several of my colleagues made significant contributions to this project, in particular Yves Jaradin was responsible for the original architecture of Raincode Assembler Compiler.

References

P. Abel. *Programming Assembler Language IBM 370*. Prentice Hall, 3rd edition, 1989.

V. Blagodarov, Y. Jaradin, and V. Zaytsev. Tool Demo: Raincode Assembler Compiler. In T. van der Storm, E. Balland, and D. Varr, editors, *Proceedings of the Ninth International Conference on Software Language Engineering (SLE)*, pages 221–225, 2016. doi: 10.1145/2997364.2997387.

F. M. Carrano. *Assembler Language Programming for the IBM 370*. Benjamin-Cummings Pub Co, 1987.

ECMA-335. *Common Language Infrastructure (CLI)*, 6th edition, June 2012. http://www.ecma-international.org/publications/standards/Ecma-335.htm.

J.-M. Favre. Preprocessors from an Abstract Point of View. In *Proceedings of the International Conference on Software Maintenance (ICSM'96)*, pages 329–338, Washington, DC, USA, 1996. IEEE Computer Society Press.

GC24-3414-7. *IBM System/360 Disk and Tape Operating Systems Assembler Language*. IBM, 8th edition, Jan. 1970.

GC26-4943-06. *High Level Assembler for z/OS & z/VM & z/VSE Version 1 Release 6 General Information*. IBM, 2013.

D. Grune, K. van Reeuwijk, H. E. Bal, C. J. Jacobs, and K. G. Langendoen. *Modern Compiler Design*. Springer, 2012.

T. Imai, H. Masuhara, and T. Aotani. Making live programming practical by bridging the gap between trial-and-error development and unit testing. In J. Aldrich and P. Eugster, editors, *Companion Proceedings of the 2015 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH)*, pages 11–12. ACM, 2015. doi: 10.1145/2814189.2814193. URL http://doi.acm.org/10.1145/2814189.2814193.

C. J. Kacmar. *IBM 370 Assembly Language With ASSIST. Structured Concepts and Advanced Topics*. Prentice Hall, 1988.

A. F. Kornelis. Why assembler? Bixoft, http://www.bixoft.nl/english/why.htm, 2003.

K. McQuillen. *System/360-370 Assembler Language (OS)*. Mike Murach, 1975.

K. McQuillen and A. Prince. *MVS Assembler Language*. Mike Murach & Associates, 1987.

K. C. O'Kane. *Basic IBM Mainframe Assembly Language Programming*. CreateSpace, 2011.

A. Rigo and S. Pedroni. Pypy's approach to virtual machine construction. In P. L. Tarr and W. R. Cook, editors, *Companion to the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 944–953. ACM, 2006. doi: 10.1145/1176617.1176753. URL http://doi.acm.org/10.1145/1176617.1176753.

SA22-7832-09. *z/Architecture Principles of Operation*. IBM, 10th edition, Sept. 2012.

SC26-4940-06. *High Level Assembler for z/OS & z/VM & z/VSE Version 1 Release 6 Language Reference*. IBM, 2013.

SC26-4941-06. *High Level Assembler for z/OS & z/VM & z/VSE Version 1 Release 6 Programmer's Guide*. IBM, 2013.