Haste86: Graphic CLI debugger

Haste86: Debugger for x86 assembly with graphic CLI

Xiaojun Zhou
D11, Lund University, Sweden
xiaojnzh@gmail.com

Abstract

When developing a compiler that emits X86 code, there needs to be a way to debug the code to ensure correctness. GDB is a standard solution for this in industry. However, for students under time pressure in a course, the steep learning curve may be prohibitive. Therefore, we believe that there is room for a significantly simplified debugger that only supports a small subset of GDB, in order to help students get used to debugging faster.

In this paper, we present `Haste86`. It is a X86 debugger that supports basic arithmetic operations and branching, enough to cover the needs of a compiler course at Lund University. However, through a small user study, `Haste86` did not perform as expected and received an overall negative reception.

1 Introduction

Low-level programming languages, like assembly, are quite different from high-level languages like Python, Java, or C. The direct usage of registers or the manual management of the memory are some of the most common differences between low and high level that the user may find. For example, Figures 1 & 2 show the exact same implementation in Java and in Assembly respectively. Hence, it is not too far fetched to say that students who are used to these high-level programming languages may struggle with low-level ones.

```java
public class FibExit {
    public static int fib(int n) {
        if (n <= 1) return n;
        int a = 0, b = 1;
        for (int i = 2; i <= n; i++) {
            int temp = a + b;
            a = b;
            b = temp;
        }
        return b;
    }
```

Figure 1. Example of Fibonacci sequence in Java

Course paper, EDAN70, Lund University, Sweden
January 14, 2026.

```asm
fib:
    mov eax, edi
    cmp eax, 1
    jbe .ret
    mov ecx, 0
    mov edx, 1
    mov ebx, eax
.loop:
    mov eax, ecx
    add eax, edx
    mov ecx, edx
    mov edx, eax
    dec ebx
    cmp ebx, 1
    ja  .loop
    mov eax, edx
.ret:
    ret
```

Figure 2. Same example of Fibonacci sequence as 1 in Assembly x86-64, AT&T Syntax

A compilers course "EDAN65" is given each year in Lund University, Sweden, in which students build a compiler for a simple C-like language from scratch in the course of 6 weeks. A lab should be completed and presented each week for a total of 6 labs. Working in pairs and with an average of 10-15 hours/week put into them, these labs take up a significant amount of time to finish. In the last lab of this course, the pair of students must be able to produce correct and functional x86 code. To ensure the correctness of the code, a debugger can be used to tackle this. The course program actually recommends two of them: GDB (GNU Debugger) and DDD (Data Display Debugger).

GDB is a feature rich debugger [5] and can run on most modern operating systems. However, we hypothesize that the large number of features in GDB and the fact that it uses a terminal-based interface leads to a relatively steep learning curve [6]. Due to the time constraints of the lab, this can deter students from using GDB altogether. The students are expected to deliver a compiler that generates correct X86-64 code with AT&T syntax in a week. Students that chose to learn GDB may end up spending a large portion of the lab time in just understanding how to use GDB and learning assembly x86 instead of the actual lab.

DDD is an alternative for GDB, which is based on GDB but with a GUI [7]. And thanks in part to its GUI, it is fairly easier to use than a CLI-based program [6]. However, it is only able to run on computers with Linux OS, which only a small fraction of students use. Possible workarounds for students not using Linux include virtual machines and remote

desktops. This can perhaps become a hassle, since every time you needed to do a small change or check something, you would need to initialize the virtual machine again or have Internet connection to the remote desktop. However, for Linux, DDD with its GUI can be easier to pick up than GDB [6], but it is hard to use for other OS.

In this paper we present our own debugger, `Haste86`. `Haste86` is a debugger designed to have a low learning curve, with only the features necessary to debug the code produced in the Compilers course, and not limited in usage by the OS of the student's computer. The aim of `Haste86` is not to compete with GDB nor DDD, but to be a simpler alternative, which would let students, who want to focus on the lab, be able to debug X86 assembly code more easily. Through a small user study we found that our initial hypothesis of GDB not being beginner-friendly was quite accurate and that grapic CLI or TUI improved readability and ease of use for a new user. However, we also find out that `Haste86` still has a lot of things that it can be improved upon and the results were not as satisfactory as we would have wanted it to be.

2 Solution

`Haste86` is a debugger for a subset of x86 implemented on Java. To be more precise, the subset is x86-64 in AT&T syntax. An arquitecture overview is shown in Figure 3. In this section we describe the implementation and design choices behind `Haste86`

`Haste86` has a reduced scope focusing only on the instructions listed in the compiler course "EDAN65" given at Lund University, Lund, Sweden. It aims for a more graphical user interface, which in turn will make it less difficult to use as a typical CLI-based user interface [6].

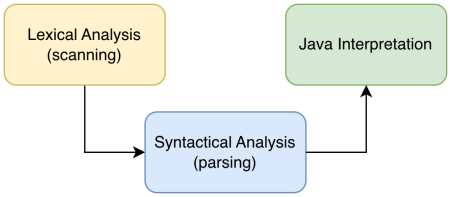

Figure 3. Overview of `Haste86`

`Haste86` implements a scanner using JFlex and a parser using NeoBeaver. It also uses JastAdd for modelling the AST and implementing the actual debugging functionality; but, its semantic analysis is very limited and most errors are encountered in runtime. In addition, it can detect some lexical and syntactical errors.

Since everything is interpreted on Java, there is no need to compile the input assembly file. `Haste86`'s compilation will only be needed the first time it is downloaded. This in turn speeds up the process of running the files. Another advantage of it being interpreted is that it is runnable in most OS's that can run Java, since it uses a virtual stack, registers, and whatnot.

We can see in Figure 4 how the graphic CLI of `Haste86` actually looks like. And because `Haste86` was designed with ease of use on mind, it resulted in some design choices that can be seen in Figure 4 and we think should be highlighted:

1. `Haste86` always prints the state of registers below R8 only, but only prints R8 to R15 if any of these registers have been modified. Since the intended solution to the labs do not require their usage, not showing these registers reduce information density in the UI. This allows students to visualize the registers that they usually use much easier.

2. `Haste86` always displays available actions at the bottom, inspired by text editor `Nano`; this may be helpful for students to pick up the debugger more easily and it also reduces the amount of information the user has to keep in their heads.

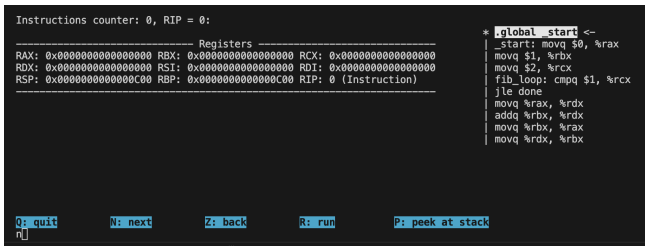

Figure 4. Graphic CLI of `Haste86`

3 Evaluation

In this section, we compare `Haste86` with GDB in terms of how easy it is to pick up and debug with.

3.1 Interview Design

We tested `Haste86` on 3 willing participants who had programmed and debugged in general before, and who had signed a consent form. Firstly we asked them some questions about how good they were at programming compared to their peers in a scale of -2 to +2, where -2 meant that they were much worse than their peers and +2 that they were much better than their peers, we repeated the same question with programming in assembly and with using debuggers. This was helpful to classify them and was inspired by Peitek et al [2], who found that the best indicator of "programming

efficiency" is to ask the programmer to rate themselves compared to their peers. We then asked them some questions about their experience with GDB, whether they had used it before, if they remember their first experience with it, and we took notes of their responses. Then they tested `Haste86`, debugging a buggy fibonacci sequence in assembly. Originally, they were supposed to test debugging the program in GDB too, however, due to time limitations, only one of the participants did it. Lastly, we discussed their experience with Haste86 and listened to the improvements they wanted to see implemented.

3.2 Results

	ASM Expertise	GDB first impression	Haste86	GDB
P0	-2	Didn't understand it	6	6
P1	0	Confusing	4	7
P2	+2	Really lost	0*	6

Figure 5. Summary table of the interview responses

The table shown in Figure 5 summarizes the interviews. The interviewees shown, named P0, P1 and P2, were not interviewed in this order, but we decided to sort the table like this to more easily see the trends in their responses. P0 is the beginner as shown with their -2 in ASM expertise compare to their peer, P1 is the average student and P2 is the expert among these three participants.

We can see that `Haste86` is more favored by beginners and becomes less favored the better they are at assembly. P1's rating of GDB was pretty high because they were using it mostly for their C programs. P2's rating was a 6 for GDB because they were used to other debuggers which implements graphic CLI such as `GDB-PEDA` and `PwnDBG`.

The results show that the initial hypothesis about GDB being beginner unfriendly was quite on point, since 2 of the interviewed people answered the question "Do you remember your impression the first time uning it?" using quotes: "I felt lost as s###" and "Pretty bad".

`Haste86` received an overall low score, with an average of 3.33 out of 10. Its biggest flaws was that it wasn't fully developed, the expert interviewee felt that if it was finished they would have given it a 6 instead of a 0. Some praised the design choices we made as overall improvements. But it still lacked some features they might have liked.

3.3 Discussion

At the end, GDB was given a overall better score than `Haste86`. And although `Haste86` did not perform as we expected, the overall receiving of the product if it was finished could surpass GDB. However, we will like to say that the users did not feel lost with `Haste86` at any point while testing it. So we can at least conclude that graphic CLI augments ease of use in complex systems.

3.4 Threats to validity

It should be mentioned that some of these participants were friends and acquaintances. Therefore, they might have shown courtesy bias[1] in their responses. Also, the limited number of interviewees could be a threat to the validity of our research, as it may affect the findings and the conclusions arrived from these findings [4].

P0's score for `Haste86` and GDB should be taken with a grain of salt since they haven't used assembly before, and their struggles could easily be associated with difficulties understanding a new language as well as trouble using a debugger for a language they don't know.

4 Related work

Risberg Alaküla et al. (2025) developed a tool and performed semi-structured interviews on students in a course on program analysis at Lund University. Their study design inspired the evaluation of this paper [3].

Stallman et al. (1988) introduced the GNU Debugger (GDB), a foundational tool for low-level program inspection and debugging [5]. It provided the debugging approaches that our system is inspired by.

Antony Unwin and Heike Hofman (1999) made a comparison between the usability of GUI and CLI and whether they could be synergizing or conflicting [6]. A synergistic design philosophy has been translated into `Haste86`.

Andreas Zeller and Dorothea Lütkehaus (1996) introduced the Data Display Debugger (DDD) based on GDB but with a GUI [7]. The "real" original reference was Dorothea's thesis, sixth reference in their ACM paper. However, it is in german and not available online, so we used this one instead, providing us with a clear view from the authors about their motivations for building DDD and what they wanted to achieve with it.

[1] Courtesy bias is the tendency that some individuals have of not fully stating their unhappiness with a service or product because of a desire not to offend the person or organization that they are responding to [1].

5 Conclusion

In this report, we built a tool with easy UI. We did user study, showed it was usable. Future developers could port this UI to GDB itself, making the result powerful and retaining the simplicity of our implementation

Acknowledgments

First of all, I want to thank Görel Hedin and Johan Eker. I had made some serious scheduling mistakes and they were so kind to help me find a solution to a problem I myself caused.

Then, I want to thank all three of the interviewees. They voluntarily accepted my request to test the prototype and gave their feedback. Even though I asked for half an hour of their time, we ended up spending more than 1 hour in each interview, so I am very grateful for their consideration and patience.

Finally, the person who has helped me the most in this project and whose guidance was lifesaving, my supervisor, Anton Risberg Alaküla. Without his help week by week, I do not think I would have been able to do a project half as good as the one I ended up making. I had some troubles caused by my inexperience, since this was my first big project ever, but Anton was considerate with me. He was really helpful and understanding, since I may had some difficulties communicating my ideas in English, but in the end, thanks to his guidance on how should I approach this project it came to fruition.

To all these 6 people, whose participation was important in this project, I am very thankful.

References

[1] Alleydog.com. [n. d.]. Courtesy Bias. https://www.alleydog.com/glossary/definition.php?term=Courtesy+Bias. ([n. d.]). Alleydog.com's online glossary, n.d.

[2] Norman Peitek, Annabelle Bergum, Maurice Rekrut, Jonas Mucke, Matthias Nadig, Chris Parnin, Janet Siegmund, and Sven Apel. 2022. Correlates of programmer efficacy and their link to experience: A combined EEG and eye-tracking study. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 120–131.

[3] Anton Risberg Alaküla, Niklas Fors, and Emma Söderberg. 2025. Study of the Use of Property Probes in an Educational Setting. In *The Art, Science, and Engineering of Programming*.

[4] Margarete Sandelowski. 1995. Sample size in qualitative research. *Research in nursing & health* 18, 2 (1995), 179–183.

[5] Richard Stallman, Roland Pesch, and Stan Shebs. 1988. *Debugging with GDB*. Free Software Foundation. https://sourceware.org/gdb/current/onlinedocs/gdb/

[6] Antony Unwin and Heike Hofmann. 1999. GUI and Command-line-Conflict or Synergy? *Computing Science and Statistics* (1999), 246–253.

[7] Andreas Zeller and Dorothea Lütkehaus. 1996. DDD—a free graphical front-end for UNIX debuggers. *ACM Sigplan Notices* 31, 1 (1996), 22–27.