Nearly every text has a throw-away chapter as Chapter One. Here's my version. Seriously, though, some important copyright, instructional, and support information appears in this chapter. So you'll probably want to read this stuff. Instructors will definitely want to review this material.

1.1 Foreword to the HLA Version of "The Art of Assembly..."

In 1987 I began work on a text I entitled "How to Program the IBM PC, Using 8088 Assembly Language." First, the 8088 faded into history, shortly thereafter the phrase "IBM PC" and even "IBM PC Compatible" became far less dominant in the industry, so I retitled the text "The Art of Assembly Language Programming." I used this text in my courses at Cal Poly Pomona and UC Riverside for many years, getting good reviews on the text (not to mention lots of suggestions and corrections). Sometime around 1994-1995, I converted the text to HTML and posted an electronic version on the Internet. The rest, as they say is history. A week doesn't go by that I don't get several emails praising me for releasing such a fine text on the Internet. Indeed, I only hear three really big complaints about the text: (1) It's a University textbook and some people don't like to read textbooks, (2) It's 16-bit DOS-based, and (3) there isn't a print version of the text. Well, I make no apologies for complaint #1. The whole reason I wrote the text was to support my courses at Cal Poly and UC Riverside. Complaint #2 is quite valid, that's why I wrote this version of the text. As for complaint #3, it was really never cost effective to create a print version; publishers simply cannot justify printing a text 1,500 pages long with a limited market. Furthermore, having a print version would prevent me from updating the text at will for my courses.

The astute reader will note that I haven't updated the electronic version of "The Art of Assembly Language Programming" (or "AoA") since about 1996. If the whole reason for keeping the book in electronic form has been to make updating the text easy, why haven't there been any updates? Well, the story is very similar to Knuth's "The Art of Computer Programming" series: I was sidetracked by other projects[1].

The static nature of AoA over the past several years was never really intended. During the 1995-1996 time frame, I decided it was time to make a major revision to AoA. The first version of AoA was MS-DOS based and by 1995 it was clear that MS-DOS was finally becoming obsolete; almost everyone except a few die-hards had switched over to Windows. So I knew that AoA needed an update for Windows, if nothing else.

I also took some time to evaluate my curriculum to see if I couldn't improve the pedagogical (teaching) material to make it possible for my students to learn even more about 80x86 assembly language in a relatively short 10-week quarter.

One thing I've learned after teaching an assembly language course for over a decade is that support software makes all the difference in the world to students writing their first assembly language programs. When I first began teaching assembly language, my students had to write all their own I/O routines (including numeric to string conversions for numeric I/O). While one could argue that there is some value to having students write this code for themselves, I quickly discovered that they spent a large percentage of their project time over the quarter writing I/O routines. Each moment they spent writing these relatively low-level routines was one less moment available to them for learning more advanced assembly language programming techniques. While, I repeat, there is some value to learning how to write this type of code, it's not all that related to assembly language programming (after all, the same type of problem has to be solved for *any* language that allows numeric I/O). I wanted to free the students from this drudgery so they could learn more about assembly language programming. The result of this observation was "The UCR Standard Library for 80x86 Assembly Language Programmers." This is a library containing several hundred I/O and utility functions that students could use in their assembly language programs. More than nearly anything else, the UCR Standard Library improved the progress students made in my courses.

1. Actually, another problem is the effort needed to maintain the HTML version since it was a manual conversion from Adobe Framemaker. But that's another story...

It should come as no surprise, then, that one of my first projects when rewriting AoA was to create a new, more powerful, version of the UCR Standard Library. This effort (the UCR Stdlib v2.0) ultimately failed (although you can still download the code written for v2.0 from http://webster.cs.ucr.edu). The problem was that I was trying to get MASM to do a little bit more than it was capable of and so the project was ultimately doomed.

To condense a really long story, I decided that I needed a new assembler. One that was powerful enough to let me write the new Standard Library the way I felt it should be written. However, this new assembler should also make it much easier to learn assembly language; that is, it should relieve the students of some of the drudgery of assembly language programming just as the UCR Standard Library had. After three years of part-time effort, the end result was the "High Level Assembler," or HLA.

HLA is a radical step forward in teaching assembly language. It combines the syntax of a high level language with the low-level programming capabilities of assembly language. Together with the HLA Standard Library, it makes learning and programming assembly language almost as easy as learning and programming a High Level Language like Pascal or C++. Although HLA isn't the first attempt to create a hybrid high level/low level language, nor is it even the first attempt to create an assembly language with high level language syntax, it's certainly the first complete system (with library and operating system support) that is suitable for teaching assembly language programming. Recent experiences in my own assembly language courses show that HLA is a major improvement over MASM and other traditional assemblers when teaching machine organization and assembly language programming.

The introduction of HLA is bound to raise lots of questions about its suitability to the task of teaching assembly language programming (as well it should). Today, the primary purpose of teaching assembly language programming at the University level isn't to produce a legion of assembly language programmers; it's to teach machine organization and introduce students to machine architecture. Few instructors realistically expect more than about 5% of their students to wind up working in assembly language as their primary programming language[2]. Doesn't turning assembly language into a high level language defeat the whole purpose of the course? Well, if HLA lets you write C/C++ or Pascal programs and attempted to call these programs "assembly language" then the answer would be "Yes, this defeats the purpose of the course." However, despite the name and the high level (and very high level) features present in HLA, HLA is still assembly language. An HLA programmer still uses 80x86 machine instructions to accomplish most of the work. And those high level language statements that HLA provides are purely optional; the "purist" can use nothing but 80x86 assembly language, ignoring the high level statements that HLA provides. Those who argue that HLA is not *true* assembly language should note that Microsoft's MASM and Borland's TASM both provide many of the high level control structures found in HLA[3].

Perhaps the largest deviation from traditional assemblers that HLA makes is in the declaration of variables and data in a program. HLA uses a very Pascal-like syntax for variable, constant, type, and procedure declarations. However, this does not diminish the fact that HLA is an assembly language. After all, at the machine language (vs. assembly language) level, there is no such thing as a data declaration. Therefore, any syntax for data declaration is an abstraction of data representation in memory. I personally chose to use a syntax that would prove more familiar to my students than the traditional data declarations used by assemblers.

Indeed, perhaps the principle driving force in HLA's design has been to leverage the student's existing knowledge when teaching them assembly language. Keep in mind, when a student first learns assembly language programming, there is so much more for them to learn than a handful of 80x86 machine instructions and the machine language programming paradigm. They've got to learn assembler directives, how to declare variables, how to write and call procedures, how to comment their code, what constitutes good programming style in an assembly language program, etc. Unfortunately, with most assemblers, these concepts are completely different in assembly language than they are in a language like Pascal or C/C++. For example, the indentation techniques students master in order to write readable code in Pascal just don't apply to (traditional) assembly language programs. That's where HLA deviates from traditional assemblers. By using a

2. My experience suggests that only about 10-20% of my students will ever write *any* assembly language again once they graduate; less than 5% ever become regular assembly language users.

3. Indeed, in some respects the MASM and TASM HLL control structures are actually *higher* level than HLA's. I *specifically* restricted the statements in HLA because I did not want students writing "C/C++ programs with MOV instructions."

high level syntax, HLA lets students leverage their high level language knowledge to write good readable programs. HLA will not let them avoid learning machine instructions, but it doesn't force them to learn a whole new set of programming style guidelines, new ways to comment your code, new ways to create identifiers, etc. HLA lets them use the knowledge they already possess in those areas that really have little to do with assembly language programming so they can concentrate on learning the important issues in assembly language.

So let there be no question about it: HLA is an assembly language. It is not a high level language masquerading as an assembler[4]. However, it is a system that makes learning and using assembly language easier than ever before possible.

Some long-time assembly language programmers, and even many instructors, would argue that making a subject easier to learn diminishes the educational content. Students don't get as much out of a course if they don't have to work very hard at it. Certainly, students who don't apply themselves as well aren't going to learn as much from a course. I would certainly agree that if HLA's only purpose was to make it easier to learn a fixed amount of material in a course, then HLA would have the negative side-effect of reducing what the students learn in their course. However, the real purpose of HLA is to make the educational process more efficient; not so the students spend less time learning a fixed amount of material (although HLA could certainly achieve this), but to allow the students to learn the same amount of material in less time *so they can use the additional time available to them to advance their study of assembly language*. Remember what I said earlier about the UCR Standard Library- it's introduction into my course allowed me to teach even more advanced topics in my course. The same is true, even more so, for HLA. Keep in mind, I've got ten weeks in a quarter. If using HLA lets me teach the same material in seven weeks that took ten weeks with MASM, I'm not going to dismiss the course after seven weeks. Instead, I'll use this additional time to cover more advanced topics in assembly language programming. That's the real benefit to using pedagogical tools like HLA.

Of course, once I've addressed the concerns of assembly language instructors and long-time assembly language programmers, the need arises to address questions a student might have about HLA. Without question, the number one concern my students have had is "If I spend all this time learning HLA, will I be able to use this knowledge once I get out of school?" A more blunt way of putting this is "Am I wasting my time learning HLA?" Let me address these questions using three points.

First, as pointed out above, most people (instructors and experienced programmers) view learning assembly language as an educational process. Most students will probably never program full-time in assembly language, indeed, few programmers write more than a tiny fraction (less than 1%) of their code in assembly language. One of the main reasons most Universities require their students to take an assembly language course is so they will be familiar with the low-level operation of their machine and so they can appreciate what the compiler is doing for them (and help them to write better HLL code once they realize how the compiler processes HLL statements). HLA is an assembly language and learning HLA will certainly teach you the concepts of machine organization, the real purpose behind most assembly language courses.

The second point to ponder is that learning assembly language consists of two main activities; learning the assembler's syntax and learning the assembly language programming paradigm (that is, learning to *think* in assembly language). Of these two, the second activity is, by far, the more difficult. HLA, since it uses a high level language-like syntax, simplifies learning the assembly language syntax. HLA also simplifies the initial process of learning to program in assembly language by providing a crutch, the HLA high level statements, that allows students to use high level language semantics when writing their first programs. However, HLA does allow students to write "pure" assembly language programs, so a good instructor will ensure that they master the full assembly language programming paradigm before they complete the course. Once a student masters the semantics (i.e., the programming paradigm) of assembly language, learning a new syntax is relatively easy. Therefore, a typical student should be able to pick up MASM in about a week after mastering HLA[5].

As for the third and final point: to those that would argue that this is still extra effort that isn't worthwhile, I would simply point out that none of the existing assemblers have more than a cursory level of com-

4. The C-- language is a good example of a low-level non-assembly language, if you need a comparison.
5. This is very similar to mastering C after learning C++.

patibility. Yes, TASM can assemble most MASM programs, but the reverse is not true. And it's certainly not the case that NASM, A86, GAS, MASM, and TASM let you write interchangeable code. If you master the syntax of one of these assemblers and someone expects you to write code in a different assembler, you're still faced with the prospect of having to learn the syntax of the new assembler. And that's going to take you about a week (assuming the presence of well-written documentation). In this respect, HLA is no different than any of the other assemblers.

Having addressed these concerns you might have, it's now time to move on and start teaching assembly language programming using HLA.

1.2 Intended Audience

No single textbook can be all things to all people. This text is no exception. I've geared this text and the accompanying software to University level students who've never previously learned assembly language programming. This is not to say that others cannot benefit from this work; it simply means that as I've had to make choices about the presentation, I've made choices that should prove most comfortable for this audience I've chosen.

A secondary audience who could benefit from this presentation is any motivated person that really wants to learn assembly language. Although I assume a certain level of mathematical maturity from the reader (i.e., high school algebra), most of the "tough math" in this textbook is incidental to learning assembly language programming and you can easily skip over it without fear that you'll miss too much. High school students and those who haven't seen a school in 40 years have effectively used this text (and its DOS counterpart) to learn assembly language programming.

The organization of this text reflects the diverse audience for which it is intended. For example, in a standard textbook each chapter typically has its own set of questions, programming exercises, and laboratory exercises. Since the primary audience for this text is University students, such pedagogical material does appear within this text. However, recognizing that not everyone who reads this text wants to bother with this material (e.g., downloading it), this text moves such pedagogical material to the end of each volume in the text and places this material in a separate chapter. This is somewhat of an unusual organization, but I feel that University instructors can easily adapt to this organization and it saves burdening those who aren't interested in this material.

One audience to whom this book is specifically not directed are those persons who are already comfortable programming in 80x86 assembly language. Undoubtedly, there is a lot of material such programmers will find of use in this textbook. However, my experience suggests that those who've already learned x86 assembly language with an assembler like MASM, TASM, or NASM rebel at the thought of having to relearn basic assembly language syntax (as they would to have to learn HLA). If you fall into this category, I humbly apologize for not writing a text more to your liking. However, my goal has always been to teach those who don't already know assembly language, not extend the education of those who do. If you happen to fall into this category and you don't particularly like this text's presentation, there is some good news: there are dozens of texts on assembly language programming that use MASM and TASM out there. So you don't really need this one.

1.3 Teaching From This Text

The first thing any instructor will notice when reviewing this text is that it's far too large for any reasonable course. That's because assembly language courses generally come in two flavors: a machine organization course (more hardware oriented) and an assembly language programming course (more software oriented). No text that is "just the right size" is suitable for both types of classes. Combining the information for both courses, plus advanced information students may need after they finish the course, produces a large text, like this one.

If you're an instructor with a limited schedule for teaching this subject, you'll have to carefully select the material you choose to present over the time span of your course. To help, I've included some brief notes

at the beginning of each Volume in this text that suggests whether a chapter in that Volume is appropriate for a machine organization course, an assembly language programming course, or an advanced assembly programming course. These brief course notes can help you choose which chapters you want to cover in your course.

If you would like to offer hard copies of this text in the bookstore for your students, I will attempt to arrange with some "Custom Textbook Publishing" houses to make this material available on an "as-requested" basis. As I work out arrangements with such outfits, I'll post ordering information on Webster (http://webster.cs.ucr.edu). If your school has a printing and reprographics department, or you have a local business that handles custom publishing, you can certainly request copyright clearance to print the text locally.

If you're not taking a formal course, just keep in mind that you don't have to read this text straight through, chapter by chapter. If you want to learn assembly language programming and some of the machine organization chapters seem a little too hardware oriented for your tastes, feel free to skip those chapters and come back to them later on, when you understand the need to learn this information.

1.4 Copyright Notice

1.5 How to Get a Hard Copy of This Text

This text is distributed in electronic form only. It is not available in hard copy form nor do I personally intend to have it published. If you want a hard copy of this text, the copyright allows you to print one for yourself. The PDF distribution format makes this possible (though the length of the text will make it somewhat expensive).

If you're wondering why I don't get this text published, there's a very simple reason: it's too long. Publishing houses generally don't want to get involved with texts for specialized subjects as it is; the cost of producing this text is prohibitive given its limited market. Rather than cut it down to the 500 or so 6" x 9" pages that most publishers would accept, my decision was to stick with the full text and release the text in electronic form on the Internet. The upside is that you can get a free copy of this text; the downside is that you can't readily get a hard copy.

Note that the copyright notice forbids you from copying this text for anything other than personal use (without permission, of course). If you run a "Print to Order/Custom Textbook" publishing house and would like to make copies for people, feel free to contact me and maybe we can work out a deal for those who just have to have a hard copy of this text.

1.6 Obtaining Program Source Listings and Other Materials in This Text

All of the software appearing in this text is available from the Webster web site. The URL is

http://webster.cs.ucr.edu

The exact filename(s) of this material may change with time, and different services use different names for these files. Check on Webster for any important changes in addresses. If for some reason, Webster disappears in the future, you should use a web-based search engine like "AltaVista" and search for "Art of Assembly" to locate the current home site of this material.

1.7 Where to Get Help

If you're reading this text and you've got questions about how to do something, please post a message to one of the following Internet newsgroups:

```
comp.lang.asm.x86
alt.lang.asm
```

Hundreds of knowledgeable individuals frequent these newsgroups and as long as you're not simply asking them to do your homework assignment for you, they'll probably be more than happy to help you with any problems that you have with assembly language programming.

I certainly welcome corrections and bug reports concerning this text at my email address. However, I regret that I do not have the time to answer general assembly language programming questions via email. I do provide support in public forums (e.g., the newsgroups above and on Webster at http://webster.cs.ucr.edu) so please use those avenues rather than emailing questions directly to me. Due to the volume of email I receive daily, I regret that I cannot reply to all emails that I receive; so if you're looking for a response to a question, the newsgroup is your best bet (not to mention, others might benefit from the answer as well).

1.8 Other Materials You Will Need (Windows Version)

In addition to this text and the software I provide, you will need a machine running a 32-bit version of Windows (Windows 9x, NT, 2000, ME, etc.), a copy of Microsoft's MASM and a 32-bit linker, some sort of

text editor, and other rudimentary general-purpose software tools you normally use. MASM and MS-Link are freely available on the internet. Alas, the procedure you must follow to download these files from Microsoft seems to change on a monthly basis. However, a quick post to comp.lang.asm.x86 should turn up the current site from which you may obtain this software. Almost all the software you need to use this text is part of Windows (e.g., a simple text editor like Notepad.exe) or is freely available on the net (MASM, LINK, and HLA). You shouldn't have to purchase anything.

1.9 Other Materials You Will Need (Linux Version)

In addition to this text and the software I provide, you will need a machine running Linux (preferably Linux 2.4 or later), "as" and "ld" (if you can compile GCC programs, you've got these, they come standard with most distributions), some sort of text editor, and other rudimentary general-purpose software tools you normally use. Although not necessary, it helps if you've got superuser priviledges during installation so you can put the software in a reasonable spot.

