Аннотация

Учебно-методический микропроцессорный стенд УМПК-80 до сих пор находит применение в учебных целях, в частности на нескольких направлениях обучения в Сургутском государственном университете. Выполнение лабораторных работ с его использованием предполагает ввод программ в машинном коде микропроцессора Intel 8080, а также составление отчёта с детальным листингом, включающего в себя используемые адреса, машинный и эквивалентный ассемблерный код, а также комментарии. Наличие листинга значительно упрощает и процесс разработки, но составление и коррекция программы ассемблирования и пересчёта адресов. Вручную или с помощью типичного ассемблера, составление листинга затруднительно и занимает много времени.

В связи с этим была поставлена цель написания ассемблера и генератора листингов в виде модуля для некого текстового процессора. Такой модуль должен позволить автору составить программу для стенда в ассемблерном коде в самом отчёте или в подгружаемых файлах и в результате генерировать детальный листинг прямо в документе. Исправления кода должны отражаться в листинге.

Процесс разработки следует стандартной схеме и включает анализ доступных расширяемых текстовых процессоров, изучение выбранного текстового процессора, изучение языка ассемблера микропроцессоров Intel 8080, написание парсера, ассемблера и инструментов форматирования для генерации листинга. Был выбран текстовый процессор ConTeXt, так как он использует язык Lua, имеет для него библиотеку для составления парсеров и имеет типографический функционал достаточный, чтобы заместить более распространённые текстовые процессоры, например Microsoft Word.

Для изучения языка ассемблера применялось выпущенное в 1977 году официальное руководство. Для составления парсера была применена библиотека LPeg для парсеров на основе РВ-грамматики, включённая в доступные по умолчанию в текстовом процессоре ConTeXt библиотеки. Демонстративные инструменты форматирования используют интерфейс для взаимодействия с ConTeXt через Lua, но в будущем предполагается интерфейс, более приближенный к самому текстовому процессору, и как следствие более удобный.

ConTeXt является весьма непопулярным текстовым процессором. Это можно объяснить отсутствием маркетинга, необходимостью составления документов на специальном (предметно-ориентированным) языке программирования, усиленным фокусом на качество вёрстки ценой сложности инструмента. В связи с этим возможности внедрения разработанного ассемблера практически отсутствуют, пока требования к качеству вёрстки в академической среде ограничены возможностями более популярных текстовых процессоров, прежде всего Microsoft Word.

Полученное на момент написания пояснительной записки программное решение имеет функционал, достаточный для использования его в указанных целях. Автор отчёта может использовать модуль для вызова ассемблера, указав исходный код программы, адрес начала и известные символы, и ассемблер соберёт программу и выдаст на месте вызова листинг. Цель была достигнута, но разработка на этом не заканчивается. Модуль требует доработки в полноценное дополнение для текстового процессора для более удобной работы. В частности, нужно реализовать поддержку ассемблерных директив, составления программ отдельными сегментами, а также предоставить возможность гибкой настройки вида листинга.

Annotation

The UMPK-80, an educational microprocessor system, is being used to this day, particularly within Surgut State University as part of several courses. Laboratory assignments using the system include program entry in Intel 8080 machine code and report preparation. The report requires a detailed code listing that includes addresses used, machine code and its assembly code equivalent, as well as comments. Having such a listing greatly simplifies the development process as well, however any changes to the program require assembly. Done manually or using a separate assembler, the process of making a code listing is time consuming and tedious.

As such, a goal has been set to develop an assembler that doubles as a code listing generator and works as a module for a text processor. Such a module must enable the author to provide a program in assembly code in the report or in auxiliary files and as a result the module must generate a code listing right in the document. Any changes in code must be reflected in the listing.

The development process follows standard procedure and includes the analysis of extensible text processors available, studying the chosen text processor, studying the Intel 8080 assembly language, writing a parser, assembler and formatter capable of generating a code listing. ConTeXt has become the text processor of choice given that it uses the Lua programming language, has a parser builder library for it and it is typographically capable enough to be a replacement for more common text processors such as Microsoft Word.

The official 1977 manual has been the only reference for the study of the Intel 8080 assembly language. The developed parser uses the LPeg library for making PEG parsers as it is included in the ConTeXt installation. A demonstrative formatter has been written which uses simplistic tools to interface with ConTeXt through Lua, however it is planned that it will be replaced with an interface more closely tied to the text processor itself and as such an easier one.

ConTeXt является весьма непопулярным текстовым процессором. Это можно объяснить отсутствием маркетинга, необходимостью составления документов на специальном (предметно-ориентированным) языке программирования, усиленным фокусом на качество вёрстки ценой сложности инструмента. В связи с этим возможности внедрения разработанного ассемблера практически отсутствуют, пока требования к качеству вёрстки в академической среде ограничены возможностями более популярных текстовых процессоров, прежде всего Microsoft Word.

ConTeXt is largely unpopular. It may be caused by a lack of marketing, usage of a domain-oriented language being a requirement for writing documents, the complexity of the tool coming as a price for high quality typesetting. It comes as no surprise that there is little hope for integrating the developed software, not until academic typesetting requirements grow beyond the possibilities of much more popular text processing software such as Microsoft Word.

At the moment of writing this explanatory note the developed software solution has enough functionality to achieve the main goal. A report author can use the module to call the assembler with given source code, starting address and known symbols, and the assembler will build a program and generate a listing in its place. The goal has been reached, but the development process continues as the module requires more work to become easier to use. In particular, the module requires assembly directive support, literate programming style source code splitting and more flexible code listing typesetting settings.

Обозначения и сокращения

|  |  |  |
| --- | --- | --- |
| АСТ | — | абстрактное синтаксическое дерево. Структура данных, отражающая структуру (фрагмента) программы |
| РВ‑грамматика | — | грамматика, разбирающая выражение. Один из типов формальной грамматики. Применяется для обработки компьютерных языков (см. также контекстно-свободную грамматику) |
| Текстовый процессор | — | программа, предназначенная для работы над форматированным текстом (составление, печать, стилизация, структуризация и т. д.) |
| УМПК‑80 | — | учебно-методический стенд, предназначенный для практического изучения микропроцессорных систем на начальном уровне. В основе лежит микропроцессор Intel 8080. Используется в Сургутском государственном университете преподавателями и студентами на направлениях «Управление в технических системах» и «Программная инженерия» |
| ConTeXt | — | текстовый процессор на основе системы компьютерной вёрстки TeX. Разрабатывается нидерландской компанией PRAGMA ADE, занимающейся автоматизированной высококачественной вёрсткой. Использует собственно разработанную новейшую версию TeX под названием LuaMetaTeX, имеющую множество улучшений, в том числе встроенный интерпретатор Lua |
| Intel 8080 | — | второй восьмибитный микропроцессор от Intel, выпуск которого начался в апреле 1974 года |
| LPeg | — | Lua Parsing Expression Grammar, библиотека для составления парсеров на основе РВ-грамматики в Lua |
| Lua | — | легко встраиваемый и расширяемый интерпретируемый язык программирования. Нередко используется в полноценных программных решениях как способ пользователю расширить функционал программы. Отличается простотой, скоростью работы интерпретатора и широким применением хеш-таблиц |

Содержание

[Введение 5](#_Toc200472184)

[1 Синтаксис языка ассемблирования Intel 8080 7](#_Toc200472185)

[2 Функционал парсерной библиотеки LPeg 8](#_Toc200472186)

[3 Составление парсера языка ассемблирования 9](#_Toc200472187)

[4 Составление ассемблера 10](#_Toc200472188)

[5 Составление и форматирование листинга 11](#_Toc200472189)

[Заключение 12](#_Toc200472190)

[Conclusion 13](#_Toc200472191)

[Список использованных источников 14](#_Toc200472192)

[Приложение А Характеристика текстового процессора ConTeXt 15](#_Toc200472193)

1. Введение

Выполнение лабораторных работ с учебно-методическим микропроцессорным стендом УМПК-80 включает в себя анализ проблемы, проектирование программного решения, последующую реализацию и тестирование с возможными доработки по мере необходимости. Это проводится в целях закрепления учебного материала по дисциплине «Организация ЭВМ». Ввод программ осуществляется в машинном коде микропроцессора в выбранную студентом область памяти. [\in[item: umpk-80]] По окончании работы необходимо привести отчёт, одним из обязательных элементов которого является детальный листинг в виде таблицы, включающей в себя адреса каждого байта машинного кода программы, метки для наглядности, сам машинный код и эквивалентный ассемблерный код, а также комментарии (таблица 1)

Таблица 1 — Пример листинга

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Адрес | Метка | Машинный  код | Ассемблерный код | Комментарии |
| 0990H | memcpy: | 1A | LDAX D | Байт считывается из источника… |
| 0991H |  | 77 | MOV M, A | … и он записывается по адресу назначения |
| 0992H |  | 13 | INX D | Следующая ячейка памяти источника… |
| 0993H |  | 23 | INX H | Следующая ячейка памяти назначения… |
| 0994H |  | 0D | DCR C | Одним байтом меньше, декремент счётчика |
| 0995H |  | C2 | JNZ memcpy | Если есть ещё байты, продолжить копирование |
| 0996H |  | 90 |
| 0997H |  | 09 |
| 0998H |  | C9 | RET | Выход из подпрограммы |

Как видно, каждый байт занимаемой (под)программой памяти имеет заранее определённый адрес, так как программист сам выбирает, где в адресном пространстве расположить программу. В листинге метка имеет ограниченную пользу, так как все адреса известны заранее, но с помощью них можно сделать переходы более наглядными. Ассемблерный код и комментарии делают листинг понятнее.

Несмотря на то, что формально листинг необходим лишь как элемент документации, особенности ввода программ на учебном стенде [\in[item: umpk-80]] делают его необходимым ещё на этапе реализации и даже реализации решения, прежде всего позволяя сопоставить машинный код и адресное пространство системы для последующего ввода программы в систему. Вариантов составления его не так много:

* Листинг можно составить вручную, что делает большинство студентов. Как правило, для этого используются электронные таблицы (в частности, Microsoft Office Excel). При аккуратном ведении листинга можно обеспечить правильную последовательность используемых адресов и даже простоту исправления неверных адресов и пропущенных команд, но процесс ассемблирования придётся выполнять вручную. По очевидным причинам это несёт за собой вероятность совершения ошибок.
* Можно воспользоваться существующими ассемблерами в дополнении к электронным таблицам. Как правило, для них нужно иметь исходный код в формате, представленном на рисунке 1. Для наглядности представлена программа из таблицы 1.  
  

Рисунок 1 — Пример исходного кода программы для типичного ассемблера для Intel 8080

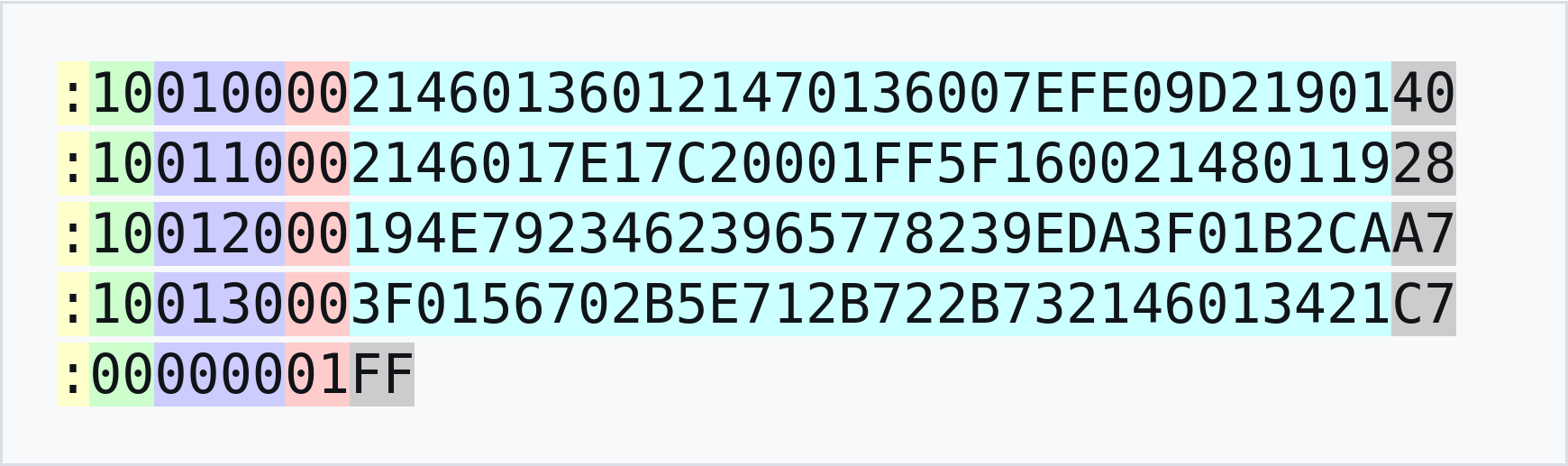
В лучшем случае с помощью них можно получить ассемблерный код в виде файла формата Intel HEX (рисунок 2). Как можно увидеть, он является достаточно неудобным для копирования. В частности, можно спутать контрольную сумму с байтом машинного кода программы; приведённые в файле (двухбайтовые) адреса представлены в порядке от старшего байта к младшему в отличие от используемого в микропроцессоре обратного порядка. Несмотря на это, ассемблеры значительно упрощают работы, самостоятельно генерируя машинный код и отсчитывая адреса. Сгенерировать файлы этого формата возможно, например, с помощью инструментов эмулятора emuStudio [\in[item: emuStudio]]. Есть и онлайн-инструменты, как Pretty Intel 8080 Assembler [\in[item: pretty assembler]].  


Рисунок 2 — Формат Intel HEX. Выделены по порядку: количество байт, адрес, тип записи, данные, контрольная сумма

* В редком случае можно найти инструменты, генерирующие листинг в неком виде. В частности, в работе name [\in[item: umpk emu]] представлен эмулятор, имеющий также функционал для генерации листингов в формате, приведённом в таблице 1. Это значительно облегчает составление отчёта, но требуют повторной генерации вне текстовых процессоров в случае изменения программы.

Необходимо отметить, что решения с применением ассемблера чаще всего предполагают наличие отдельного от текстового процессора ПО. В связи с этим была поставлена цель составить внедрённое в текстовый процессор решение, позволяющее генерировать листинги прямиком в документе. Такой подход позволяет легко составить листинг корректного вида (не требующий никаких стилистических изменений) для использования как при реализации и тестировании программы, так и при составлении документации к ней.

Проблема составления листинга является актуальной, но второстепенной — размер программ, необходимых для выполнения лабораторных работ, достаточно мал, что можно вручную выполнить ассемблирование.

1. Анализ инструментов и предметной области
   1. Доступные текстовые процессоры

Основным фактором при выборе текстового процессора в данном случае будет расширяемость за счёт встроенного в него языка программирования. Предпочтение отдаётся популярным языкам программирования, так как у тех больше пользовательских библиотек, в том числе и библиотек для составления парсеров. Это не единственный фактор — так как вероятно то, что пользователю придётся переключаться на новый текстовый процессор для того, чтобы он мог воспользоваться модулем, лучше выбрать наиболее подходящий для академических целей текстовый процессор.

Текстовые процессоры можно поделить на две группы по идеологии: более распространённую WYSIWYG (англ. *what you see is what you get*, «получишь, что видишь») и более редкую WYSIWYM (англ. *what you see is what you mean*, «имеешь в виду то, что видишь»).

Первая группа включает в себя более популярные опции, как Microsoft Word. Как следует из названия идеологии, текстовые процессоры данного вида представляют пользователю документ в том виде, в котором он будет представляться читателям. Они имеют собственные редакторы, позволяющие представить форматированный текст с различными комбинациями кеглей, шрифтов, стилей, отступов и т. д. Эти редакторы позволяют автору выбрать конкретную семью шрифтов и при вводе текста увидеть, как она будет смотреться в документе. [\in[item: wysiwyg]]

Из текстовых процессоров идеологии WYSIWYG были рассмотрены:

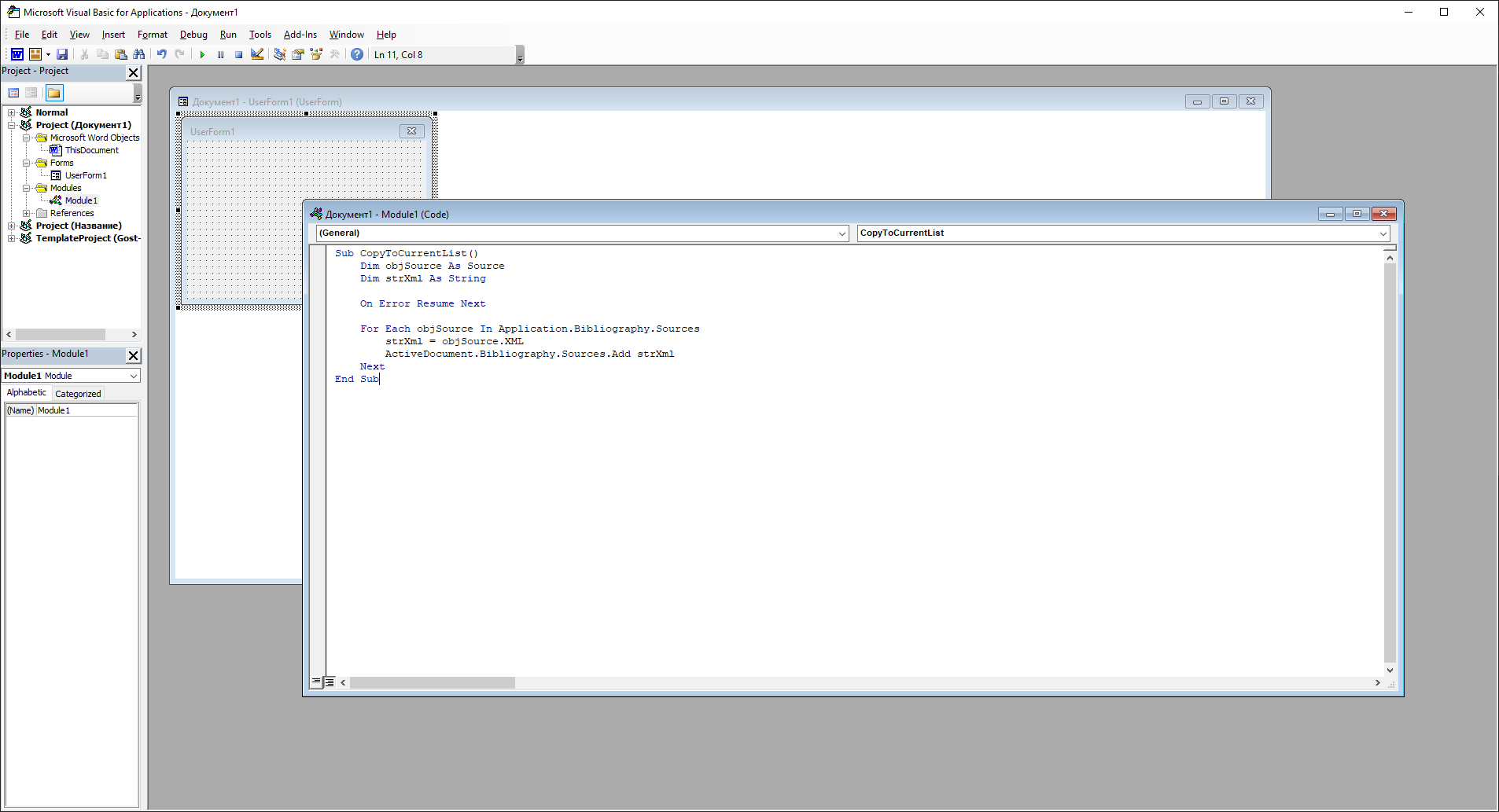
* Microsoft Word. Имеет поддержку макросов на собственном диалекте языка программирования Basic под названием Visual Basic for Applications. Он является упрощённой версией Visual Basic. На сайте Microsoft можно найти достаточно подробную документацию по языку. Язык имеет доступный даже непрограммистам синтаксис и имеет широкие возможности по взаимодействию с другим ПО от Microsoft. [\in[item: ms vba help]] На рисунке 3 можно увидеть окно редактора макросов Word.  
  

Рисунок 3 — Редактор позволяет создавать как модули кода, так и формы для взаимодействия с пользователем

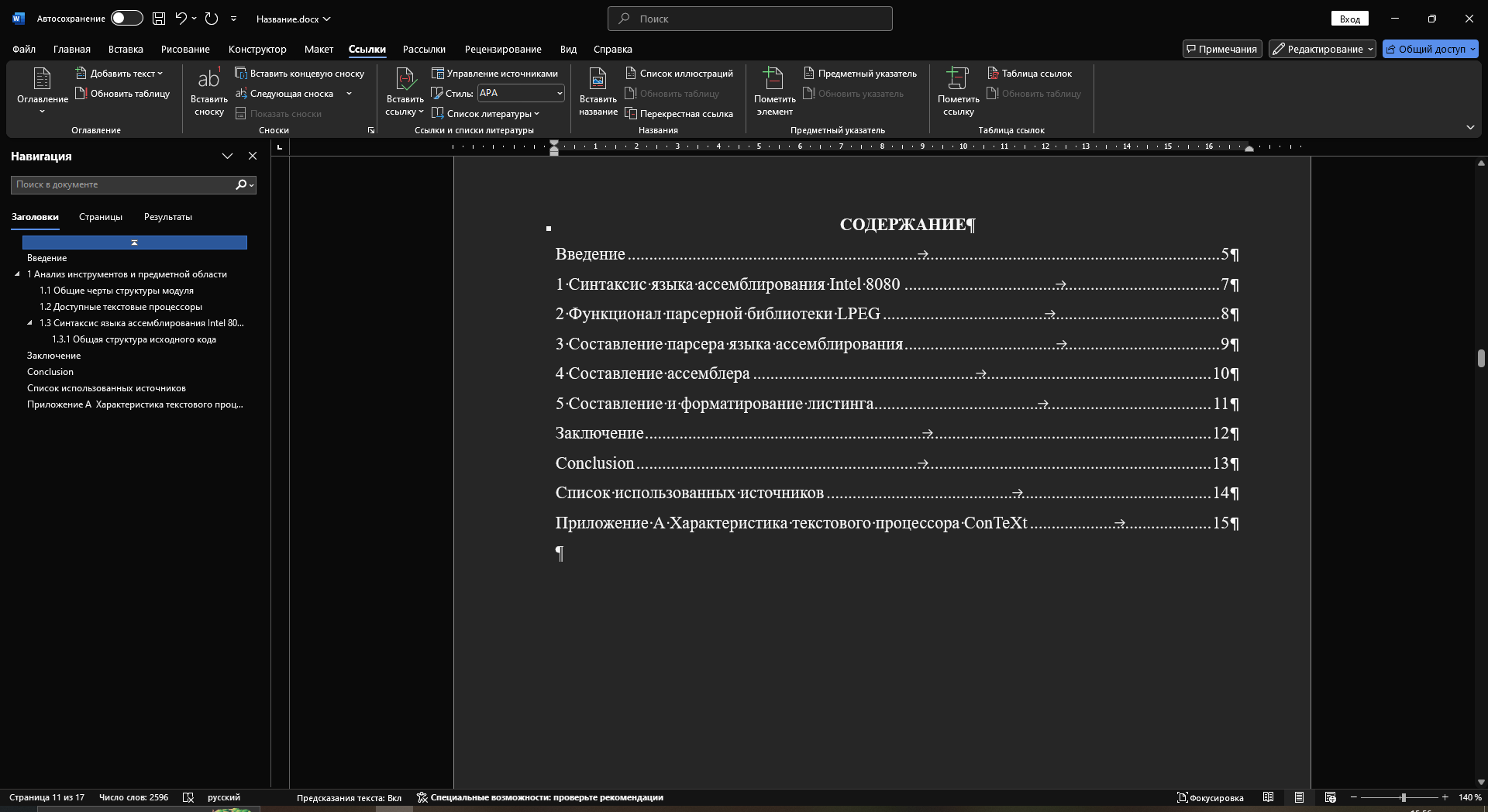
Представляется возможным написать ряд макросов для генерации листинга из исходного кода, но возникает ряд проблем. Во-первых, язык макросов VBA не обладает значительным объёмом библиотек для составления парсеров. В частности, была найдена лишь одна малоизвестная библиотека, последнее обновление которой было пять лет назад. [\in[item: vba peg]] В связи с этим представляется необходимым составлять парсер вручную. Во-вторых, в связи с упором на визуальную составляющую хранить ассемблерный код в тексте документа не представляется возможным — остаётся лишь располагать его в иных местах. Например, можно воспользоваться переменными документа [\in[item: vba doc vars]], или сохранять код в отдельных файлах. В-третьих, результирующий модуль будет требовать от пользователя обновлять таблицу вручную при каждом изменении исходного кода. Такое поведение аналогично тому у оглавлений в Word (рисунок 4); ручные обновления плохо сочетаются с подходом «немедленной готовности» документов после редактирования в текстовых процессорах WYSIWYG.  


Рисунок 4 — Содержание документа в тексте (справа) отстаёт от действительного (панель навигации справа). Необходимо обновить содержание в тексте через выпадающее меню

Стоит отметить, что составлять документы с достаточно сложной структурой в Microsoft Word представляется затруднительным. В частности, создание разделов, подразделов и т. д. требует использования многоуровневых списков с определёнными стилями на верхних уровнях, что порой мешает созданию произвольных списков внутри них. Это усложняет его использование в академических кругах.

* LibreOffice Writer. Текстовый процессор имеет два основных языка для написания макросов — Python и собственный диалект Basic, частично повторяющий функционал Visual Basic for Applications. [\in[item: libreoffice basic]] Скрипты Python в LibreOffice Writer могут использовать пользовательские библиотеки, которые можно установить как в общую папку, так и в пользовательскую папку. Также скрипты могут быть встроены в документ. [\in[item: libreoffice python]] Для Python легче найти библиотеки и инструменты для составления парсера, которые можно использовать для создания генератора листингов. [\in[item: pegen], \in[item: parsimonious]] Несмотря на это, проблема обновления листинга, хранения ассемблерного кода и неудобства структуризации документа всё так же применимы в связи с тем, как работает WYSIWYG. На рисунке 5 представлен редактор макросов LibreOffice Writer.

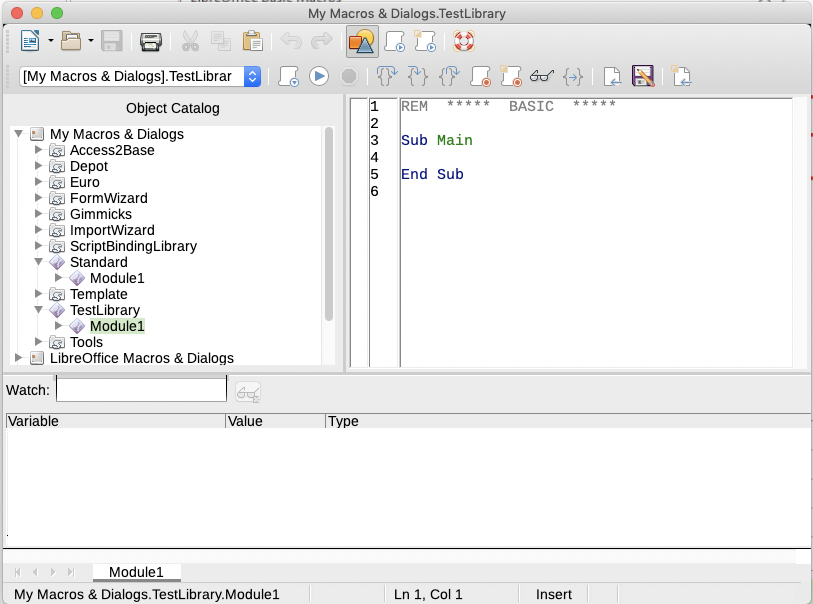


Рисунок 5 — Редактор макросов LibreOffice отчасти похож на тот у Microsoft Word

Альтернативу текстовым процессорам группы WYSIWYG составляют процессоры WYSIWYM. Вместо того, чтобы текстовый процессор немедленно отражал то, как выглядит документ, он позволяет лишь составить его структуру. Нередко подобные программы имеют собственный язык разметки или даже программирования для составления текстов. Такой подход требует конвертирования из исходного файла в формат для просмотра (HTML, PDF и др.), что является как положительной, так и отрицательной особенностью. Проверить состав текста, его структуру и стилизацию на правильность сложнее без экспорта, но становится гораздо проще автоматизировать процесс генерирования структуры и текста, например для составления документации из комментариев к коду. В случае более сложных текстовых процессоров данной группы могут быть доступны продвинутые инструменты для составления структуры в документе. [\in[item: wysiwym]]

При использовании текстовых процессоров WYSIWYM представляется возможным хранить ассемблерный код прямо в документе, так как финальный вид результирующего PDF-файла определяется выполняемыми командами при обработке.

Из текстовых процессоров данной идеологии были рассмотрены:

* LaTeX. Несмотря на то, что LaTeX (и остальные решения ниже) обычно не называют текстовыми процессорами, а, например, системами компьютерной вёрстки, функционал пересекается достаточно для рассмотрения. LaTeX является системой подготовки документов высокого типографического качества, являющейся пакетом макросов для TeX. Чаще всего она используется для составления средних или больших научно-технических документов. LaTeX поддерживает структуризацию документов на разделы с автоматическим нумерованием, вёрстку сложных математических формул, размещение рисунков и таблиц с автоматическим нумерованием, легко обновляемые перекрёстные ссылки на вышеописанные элементы. [\in[item: latex]]

Широкий список пакетов значительно расширяют LaTeX, позволяя автоматизировать ведение библиографий [\in[item: ctan biblatex]], составление графиков [\in[item: ctan pgfplots]], произвольной графики [\in[item: ctan pgf]] и т. д. LaTeX имеет собственный ненастраиваемый шаблон для статей и ожидается, что его не нужно менять; по необходимости можно скачать дополнительные, более гибкие шаблоны. Один из наиболее популярных наборов настраиваемых шаблонов — KOMA-script. [\in[item: ctan koma-script]]. LaTeX имеет активное сообщество, но страдает от малого объёма документации. Несмотря на сложность работы в LaTeX (на рисунке можно увидеть пример работы с исходным файлом), система является распространённой опцией в западных академических кругах.

<рисунок>

Наиболее продвинутым вариантом LaTeX является LuaLaTeX, использующий систему компьютерной вёрстки LuaTeX. [\in[item: lualatex]] Наиболее важным нововведением является встроенный интерпретатор Lua с API для взаимодействия с документом. Представляется возможным составить пакет для LuaLaTeX, использующий, например, библиотеку LPeg [\in[item: lpeg]] для реализации парсера.

* SILE. Эту систему вёрстки можно назвать современником TeX. Во многом повторяет функционал TeX и LaTeX, но также поддерживает Lua. SILE способен принимать файлы на собственном языке аналогично TeX/LaTeX, или же в формате XML. Не обладает широкой поддержкой ввиду своей новизны. Как и с LaTeX, представляется возможным использовать Lua для написания пакета.
* ConTeXt. Система вёрстки, которая разрабатывается параллельно с LaTeX. В отличие от LaTeX, который стремится избавить пользователя от необходимости настраивать стиль документа (если не использовать отдельные пакеты и шаблоны), ConTeXt пытается дать пользователю полный контроль над стилем собираемого текста. ConTeXt стремится быть функционально полным решением, замещая собой всё то, для чего бы в LaTeX пришлось использовать пакеты. ConTeXt имеет также и основную часть функционала LaTeX. К сожалению, всё это делает эту систему сложнее для изучения и использования. [\in[item: not so short context]] К счастью, помощь можно найти в официальной почтовой рассылке [\in[item: context mail]], в многочисленных справочниках в инсталляции и на сайте разработчика [\in[item: pragma ade]] и на официальной вики [\in[item: context garden]].

ConTeXt также построена вокруг LuaTeX (самые новые версии используют LuaMetaTeX) и в действительности эта система обладает более гибким API в Lua для работы с документом. Также ConTeXt включает вышеупомянутый пакет LPeg в инсталляции, что делает составление модуля проще.

После рассмотрения текстовых процессоров было решено, что целесообразнее выбрать одно из решений группы WYSIWYM. Предпочтение было отдано ConTeXt вместо LaTeX из-за наличия библиотеки LPeg и более гибкой настройки структуры документа, что лучше подходит для академических целей.

* 1. ConTeXt и Lua: CLD
  2. РВ-грамматики и LPeg
  3. Синтаксис языка ассемблирования Intel 8080
     1. Общая структура исходного кода

Исходный файл состоит из строк, каждая из которых может содержать до четырёх полей следующего формата в следующем порядке:

1. Метка или идентификатор. Метки и идентификаторы (вместе символы) мало отличаются друг от друга в плане технической реализации, представляя собой способ обращения к переменным, содержащим целочисленные именованные значения на 16 бит, доступные во время составления программы. В действительности названия меток и идентификаторов не могут пересекаться. Метки сохраняют за собой адрес, по которому будет размещена следующая команда или результат директивы. Это возможно, так как ассемблер имеет счётчик позиции, указывающий, где в адресном пространстве будет помещена следующая команда. Идентификаторы позволяют обращаться к произвольным численным значениям, которые можно присвоить с помощью директив EQU и SET. Чтобы пометить следующую команду или результат некоторых директив, необходимо указать название новой метки и поставить двоеточие. При использовании упомянутых директив идентификатор указывается аналогично, но без двоеточия. [\in[item: i8080]]
2. Команда или директива (без операндов). Указывается команда для микропроцессора или директива, побуждающая ассемблер изменить своё состояние или определить содержимое программы вне команд — изменить текущее значение счётчика позиции, установить значение переменной, выделить участок адресного пространства программы под произвольные данные и так далее. Все команды и директивы состоят лишь из латиницы. [\in[item: i8080]]
3. Операнды. Если команда или директива нуждается в операндах, то они идут после и разделяются запятыми. Операнды бывают нескольких видов: регистры, регистровые пары и выражения. Выражениям посвящён отдельный подраздел. [\in[item: i8080]]
4. Комментарии. Начинаются с символа точки с запятой и заканчиваются вместе со строкой. Игнорируются стандартными ассемблерами и поэтому могут содержать любые символы, кроме перехода на следующую строку. [\in[item: i8080]]
5. Проектирование модуля
6. Написание модуля
7. Работа с модулем
8. Пути дальнейшей разработки
9. Заключение
10. Conclusion
11. Список использованных источников
12. Характеристика текстового процессора ConTeXt