Аннотация

В данной пояснительной записке приводится описание разработки ассемблера и генератора детализированных листингов для микропроцессора Intel 8080 в виде интегрированного с текстовым процессором ConTeXt решения и составленного на языке программирования Lua. ПО предназначено для облегчения разработки и документации программ, написанных для учебных микропроцессорных стендов УМПК-80. Работа также преследует побочную цель демонстрации возможностей текстового процессора ConTeXt как средства составления сложной документации, в том числе с применением автоматизированной вёрстки.

Процесс разработки следует стандартной схеме и включает изучение языка ассемблера микропроцессоров Intel 8080, реализацию парсера, ассемблера и инструментов форматирования для генерации листинга. Для изучения языка ассемблера применялось выпущенное в 1977 году официальное руководство. Для составления парсера была применена библиотека LPEG для составления 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 Office Word.

Полученное на момент написания пояснительной записки программное решение имеет функционал, достаточный для использования его в указанных целях, но требует доработки в полноценный модуль 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

This explanatory note contains a description of the development process of an assembler and code listing generator for the Intel 8080 line of microprocessors in the form of a solution integrated with the ConTeXt text processor and written in the Lua programming language. This software is intended to simplify the development and documentation of programmes written for the UMPK-80 educational microprocessor system. This work also intends to demonstrate the typesetting capabilities of ConTeXt in the realm of documentation, including the usage of its automated typesetting instruments.

The development process follows standard procedure and includes studying the assembly language itself followed by developing an appropriate parser, assembler and formatter capable of typesetting a code listing. The official Intel 8080 assembly language manual is the main reference used. The parser has been written using the LPEG library for writing parsers based on parsing expression grammars. It comes by default with ConTeXt. A demonstrative formatter has been developed that depends mainly on an API that allows using ConTeXt functionality from Lua, however the final product will include a formatter that is more integrated with the text processor itself.

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 Office Word.

At the moment of writing this explanatory note the developed software solution is capable of achieving the given goals, however it requires further development to make it easier to use. It is planned that the solution will receive assembly directive support, literate programming style source code splitting and more flexible code listing typesetting capabilities.

Обозначения и сокращения

|  |  |  |
| --- | --- | --- |
| РВ‑грамматика | — | грамматика, разбирающая выражение. Один из типов формальной грамматики. Применяется для обработки компьютерных языков (см. также контекстно-свободную грамматику) [\in[item: peg]] |
| 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. [\in[item: umpk-80]] 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. [\in[item: introCTX]] 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 [\in[item: cld manual]] |
| 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 [\in[item: lpeg]] |
| 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 [\in[item: lua org]] |

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

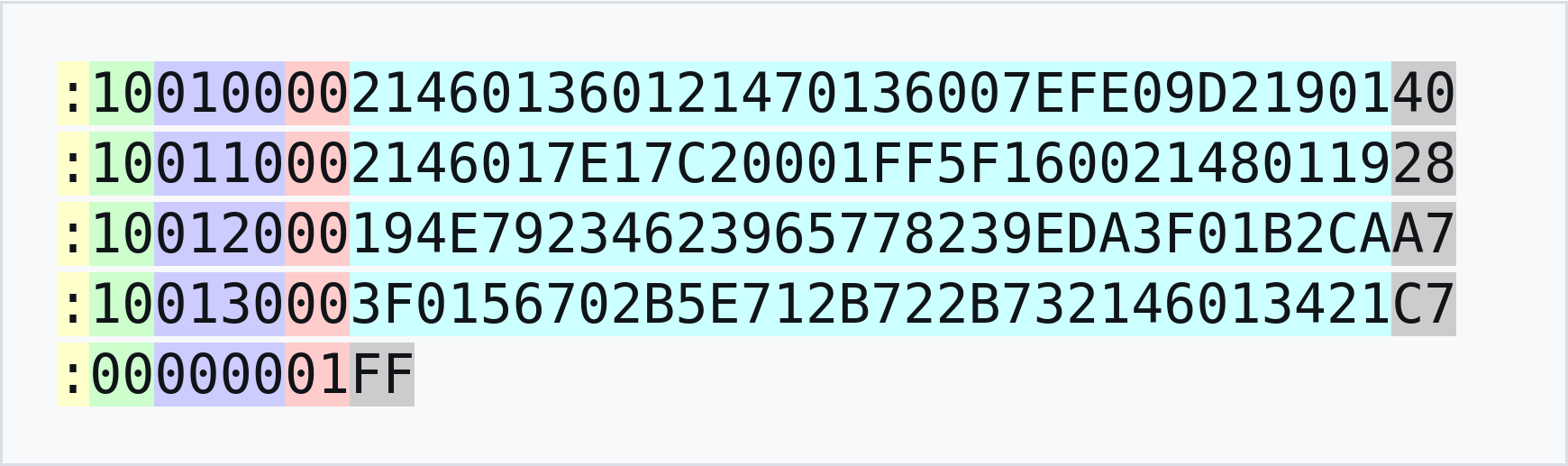
В лучшем случае с помощью них можно получить ассемблерный код в виде файла формата Intel HEX (рисунок 2). Как можно увидеть, он является достаточно неудобным для копирования. В частности, можно спутать контрольную сумму с байтом машинного кода программы; приведённые в файле (двухбайтовые) адреса представлены 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. В связи с этим данная работа выполняется скорее как демонстрация возможностей текстового процессора ConTeXt.

1. Анализ
2. 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. Функционал парсерной библиотеки LPEG
6. Составление парсера языка ассемблирования
7. Составление ассемблера
8. Составление и форматирование листинга
9. Заключение
10. Conclusion
11. Список использованных источников
12. Характеристика текстового процессора ConTeXt