Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
Национальный Исследовательский Университет ИТМО
Факультет программной инженерии и компьютерной техники

Отчёт к практическому заданию №3

Выполнил:
Студент 1 курса магистратуры «Системное и прикладное программное обеспечение» группы P4115
Эрнандес Гарсия Даниэль Дельевич

Преподаватель:
к. т. н., доцент ФПИиКТ
Кореньков Юрий Дмитриевич

Санкт-Петербург
2026

Цели: реализовать формирование линейного кода в терминах некоторого набора инструкций посредством анализа
графа потока управления для набора подпрограмм. Полученный линейный код вывести в мнемонической
форме в выходной текстовый файл.

Задачи: составить описание виртуальной машины с набором инструкций и моделью памяти по варианту

a. Изучить нотацию для записи определений целевых архитектур

b. Составить описание ВМ в соответствии с вариантом

i. Описание набор регистров и банков памяти

ii. Описать набор инструкций: для каждой инструкции задать структуру операционного

кода, содержащего описание операндов и набор операций, изменяющих состояние ВМ

1. Описать инструкции перемещения данных и загрузки констант

2. Описать инструкции арифметических и логических операций

3. Описать инструкции условной и безусловной передачи управления

4. Описать инструкции ввода-вывода с использованием скрытого регистра в

качестве порта ввода-вывода

iii. Описать набор мнемоник, соответствующих инструкциям ВМ

c. Подготовить скрипт для запуска ассемблированного листинга с использованием описания ВМ:

i. Написать тестовый листинг с использованием подготовленных мнемоник инструкций

ii. Задействовать транслятор листинга в бинарный модуль по описанию ВМ

iii. Запустить полученный бинарный модуль на исполнение и получить результат работы

iv. Убедиться в корректности функционирования всех инструкций ВМ

2. Выбрать и изучить прикладную архитектуру системы команд существующей ВМ

a. Для выбранной ВМ:

i. Должен существовать готовый эмулятор (например qemu)

ii. Должен существовать готовый тулчейн (набор инструментов разработчика):

компилятор Си, ассемблер и дизассемблер, линковщик, желательно отладчик

b. Согласовать выбор ВМ с преподавателем

c. Изучить модель памяти и набор инструкций ВМ

d. Научиться использовать тулчейн (собирать и запускать программы из листинга)

e. Подготовить скрипт для запуска ассемблированного листинга с использованием эмулятора

i. Написать тестовый листинг с использованием инструкций ВМ

ii. Задействовать ассемблер и компоновщик из тулчейна

iii. Запустить бинарный модуль на исполнение и получить результат его работы

Порядок выполнения:

1. Описать структуры данных, необходимые для представления информации об элементах образа

программы (последовательностях инструкций и данных), расположенных в памяти

a. Для каждой инструкции – имя мнемоники и набор операндов в терминах данной ВМ

b. Для элемента данных – соответствующее литеральное значение или размер экземпляра типа

данных в байтах

2. Реализовать модуль, формирующий образ программы в линейном коде для данного набора

подпрограмм

a. Программный интерфейс модуля принимает на вход структуру данных, содержащую графы

потока управления и информацию о локальных переменных и сигнатурах для набора

подпрограмм, разработанную в задании 2 (п. 1.a, п. 2.b)

b. В результате работы порождается структура данных, разработанная в п. 1, содержащая

описание образа программы в памяти: набор именованных элементов данных и набор

именованных фрагментов линейного кода, представляющих собой алгоритмы подпрограмм

c. Для каждой подпрограммы посредством обхода узлов графа потока управления в порядке

топологической сортировки (начиная с узла, являющегося первым базовым блоком алгоритма

подпрограммы), сформировать набор именованных групп инструкций, включая пролог и

эпилог подпрограммы (формирующие и разрушающие локальное состояние подпрограммы)

d. Для каждого базового блока в составе графа потока управления сформировать группу

инструкций, соответствующих операциям в составе дерева операций

e. Использовать имена групп инструкций для формирования инструкций перехода между

блоками инструкций, соответствующих узлам графа потока управления, в соответствии с

дугами в нём

3. Доработать тестовую программу, разработанную в задании 2 для демонстрации работоспособности

созданного модуля

a. Добавить поддержку аргумента командной строки для имени выходного файла, вывод

информации о графах потока управления сделать опциональных

b. Использовать модуль, разработанный в п. 2 для формирования образа программы на основе

информации, собранной в результате работы модуля, созданного в задании 2 (п. 2.b)

c. Для сформированного образа программы в линейном коде вывести в выходной файл

ассемблерный листинг, содержащий мнемоническое представление инструкций и данных, как

они описаны в структурах данных (п. 1), построенных разработанным модулем (пп. 2.c-e)

d. Проверить корректность решения посредством сборки сгенерированного листинга и запуска

полученного бинарного модуля на эмуляторе ВМ (см. подготовка п. 1.c или п. 2.e)

Описание работы: Разработана программа, которая использует tree-sitter-cli (**tree-sitter** библиотеку) с описанием входной грамматики в файле grammar.js в корне проекта для синтаксического разбора исходного кода языка, соответствующего грамматике, которая генерирует код ассемблер на основе исходного кода языка указанной грамматики.

Аспекты реализации и результаты:

На основе получаемой структуры исходного кода в терминах грамматики происходит анализ основных терминов языка: блоков, выражений (логических, алгебраических, присваиваний значений выражений переменной

(новой или существующей), доступов к массиву по индексу), циклов, условных выражений, вызовов функций и на основе данного анализа через рекурсивные функции, таблицу переменных внутри каждой функции и таблицы функций в исходном файле, строится дерево IR инструкций (трехадресный код), которые далее преобразуются в ассемблер.

Выводы: в результате проделанной работы были получены навыки написания относительно простого компилятора, который генерирует код на языке ассемблер, который далее можно скомпилировать на основных операционных системах в исполняемый файл.