

Вариант №7

Разработать ассемблер и интерпретатор для учебной виртуальной машины (УВМ).

Спецификация УВМ

Загрузка константы

A	B
Биты 0—5	Биты 6—18
32	Константа

Размер команды: 3 байт. Операнд: поле В. Результат: новый элемент на стеке.

Тест (A=32, B=461):

0x60, 0x73, 0x00

Чтение значения из памяти

A	B
Биты 0—5	Биты 6—18
54	Адрес

Размер команды: 3 байт. Операнд: значение в памяти по адресу, которым является поле В. Результат: новый элемент на стеке.

Тест (A=54, B=480):

0x36, 0x78, 0x00

Запись значения в память

A	B
Биты 0—5	Биты 6—18
52	Смещение

Размер команды: 3 байт. Операнд: элемент, снятый с вершины стека. Результат: значение в памяти по адресу, которым является сумма адреса (элемент, снятый с вершины стека) и смещения (поле В).

Тест (A=52, B=627):

0xF4, 0x9C, 0x00

Унарная операция: sqrt()

A	B
Биты 0—5	Биты 6—18
31	Адрес

Размер команды: 3 байт. Операнд: значение в памяти по адресу, которым является поле В. Результат: новый элемент на стеке.

Тест (A=31, B=635):

0xDF, 0x9E, 0x00

Этап 1. Перевод программы в промежуточное представление

Цель: создать CLI-приложение ассемблера. Реализовать разбор текстового представления команд и трансляцию в промежуточное представление.

Требования:

1. Ассемблер должен принимать на вход аргументы командной строки:
 - Путь к исходному файлу с текстом программы.
 - Путь к двоичному файлу-результату.
 - Режим тестирования.
2. Спроектировать человекочитаемый язык ассемблера, используя формат YAML. Поддерживать все команды спецификации UVM.
3. Описать в документации (например, в README.md) спроектированный язык ассемблера.
4. Реализовать транслятор, который язык ассемблера преобразует во внутреннее представление (например, список кортежей, объектов или словарей).
5. (только для данного этапа) В режиме тестирования вывести на экран внутреннее представление ассемблированной программы в формате полей и значений, как в тесте из спецификации UVM.
6. Создать программу для тестов, приведенных в спецификации UVM. Продемонстрировать, что ассемблер генерирует идентичные последовательности полей и их значений.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 2. Формирование машинного кода

Цель: реализовать логику преобразования команд в их двоичное представление.

Требования:

1. Реализовать транслятор из промежуточного в машинное представление.
2. Записать результат ассемблирования в двоичный выходной файл.
3. Вывести на экран размер двоичного файла в байтах.

4. В режиме тестирования вывести результат ассемблирования на экран в байтовом формате, как в тесте из спецификации УВМ.
5. Создать файл на языке ассемблера, результат трансляции которого соответствует всем тестовым байтовым последовательностям из спецификации УВМ.
6. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 3. Интерпретатор и операции с памятью

Цель: создать цикл интерпретации, реализовать модель памяти УВМ и выполнить базовые команды.

Требования:

1. Интерпретатор должен принимать на вход аргументы командной строки:
 - Путь к бинарному файлу с ассемблированной программой.
 - Путь к файлу, куда будет сохранен дамп памяти после выполнения программы.
 - Диапазон адресов памяти для вывода дампа.
2. Для дампа с содержимым памяти должен использоваться формат XML.
3. Реализовать модель памяти УВМ (например, в виде массивов). Память команд и память данных должны быть разделены.
4. Реализовать основной цикл интерпретатора: чтение команды из бинарного файла, перевод команды в промежуточное представление, выполнение.
5. Реализовать команды загрузки константы, а также чтение и запись в память.
6. Написать и выполнить тестовую программу, которая копирует массив с одного адреса на другой, чтобы проверить корректность работы.
7. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 4. Реализация арифметико-логического устройства (АЛУ)

Цель: завершить реализацию интерпретатора, добавив поддержку вычислительных операций.

Требования:

1. Реализовать выполнение команды `sqrt()`.

2. Для реализованной команды написать и выполнить тестовую программу, которая демонстрирует корректные вычисления с сохранением результата в память для проверки.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 5. Выполнение тестовой задачи

Цель: использовать разработанные ассемблер и интерпретатор для решения тестовой задачи.

Требования:

1. Написать, скомпилировать и исполнить программу по тестовой задаче: выполнить поэлементно команду `sqrt()` над вектором длины 7. Результат записать в новый вектор.
2. Создать три примера программ с вычислениями над различными данными. Продемонстрировать, что дамп памяти соответствует требованиям задачи.
3. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.

Этап 6. Кроссплатформенное GUI-приложение

Цель: реализовать кроссплатформенную версию UBM.

Требования:

1. Разработать GUI-версию UBM. Поддержать следующие элементы GUI:
 - редактируемое окно программы на языке ассемблера.
 - окно вывода дампа памяти.
 - кнопка для ассемблирования и запуска интерпретатора.
2. Портить GUI-версию UBM на следующие платформы:
 - Windows.
 - Linux.
 - Web/WASM.
3. Создать единый сборочный скрипт для всех поддерживаемых платформ.
4. Результат выполнения этапа сохранить в репозиторий стандартно оформленным коммитом.