

Вариант №24

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

Спецификация УВМ

Загрузка константы

A	B	C
Биты 0—5	Биты 6—12	Биты 13—38
35	Адрес	Константа

Размер команды: 5 байт. Операнд: поле С. Результат: регистр по адресу, которым является поле В.

Тест (A=35, B=81, C=368):

0x63, 0x14, 0x2E, 0x00, 0x00

Чтение значения из памяти

A	B	C
Биты 0—5	Биты 6—12	Биты 13—33
32	Адрес	Адрес

Размер команды: 5 байт. Операнд: значение в памяти по адресу, которым является поле С. Результат: регистр по адресу, которым является поле В.

Тест (A=32, B=13, C=48):

0x60, 0x03, 0x06, 0x00, 0x00

Запись значения в память

A	B	C	D
Биты 0—5	Биты 6—12	Биты 13—19	Биты 20—26
17	Адрес	Адрес	Смещение

Размер команды: 4 байт. Операнд: регистр по адресу, которым является поле В. Результат: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле С) и смещения (поле D).

Тест (A=17, B=6, C=127, D=15):

0x91, 0xE1, 0xFF, 0x00

Бинарная операция: min()

A	B	C	D
Биты 0—5	Биты 6—12	Биты 13—19	Биты 20—40
58	Адрес	Адрес	Адрес

Размер команды: 6 байт. Первый операнд: значение в памяти по адресу, которым является поле D. Второй операнд: значение в памяти по адресу, которым является регистр по адресу, которым является поле С. Результат: регистр по адресу, которым является поле B.

Тест (A=58, B=92, C=23, D=628):

0x3A, 0xF7, 0x42, 0x27, 0x00, 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 “алгебраический” синтаксис в духе языков высокого уровня. Поддержать все команды спецификации УВМ.
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 teste из спецификации УВМ.
6. Создать программу для тестов, приведенных в спецификации УВМ. 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 teste 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 CSV.
3. Реализовать модель памяти УВМ (например, в виде массивов). Память команд и память данных должны быть объединены.
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 `min()`.
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 `min()` над двумя векторами длины 8. Результат записать в первый 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 кросплатформенную версию УВМ.

Требования:

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