ДИСЦИПЛИНА **Разработка кроссплатформенных программных систем**

(полное наименование дисциплины без сокращений)

ИНСТИТУТ **Информационных технологий**

КАФЕДРА **Корпоративных информационных систем**

(полное наименование кафедры)

ВИД УЧЕБНОГО МАТЕРИАЛА **Самостоятельная работа**

(в соответствии с пп.1-11)

ПРЕПОДАВАТЕЛЬ **П.Н. Советов, А.В. Горчаков**

(фамилия, имя, отчество)

СЕМЕСТР **7 семестр (осенний) 2025/2026 учебного года**

(указать семестр обучения, учебный год)

Разработка кроссплатформенных программных систем

Сборник практических работ

ИКБО-21-22

РТУ МИРЭА – 2025

Оглавление

О практических работах

Практические работы (ПР) состоят из нескольких этапов. ПР выполняются очно и защита каждого этапа происходит на семинарских занятиях. Этапы работы над ПР сохраняются в публично доступном git-репозитории. Каждый этап разработки ПР должен быть отражен в истории коммитов с детальными сообщениями. Студент самостоятельно выбирает язык реализации.

Документация по ПР оформляется в виде readme.md, который содержит:

1. Общее описание.
2. Описание всех функций и настроек.
3. Описание команд для сборки проекта и запуска тестов.
4. Примеры использования.

Список публичных git-сервисов для репозиториев ПР:

1. github.com
2. gitea.com
3. gitlab.com
4. gitflic.ru
5. hub.mos.ru
6. gitverse.ru
7. gitee.com

Вариант №1

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—2	Биты 3—25	Биты 26—53
1	Адрес	Константа

Размер команды: 9 байт. Операнд: поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=1, B=158, C=285):

```
0xF1, 0x04, 0x00, 0x74, 0x04, 0x00, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—2	Биты 3—25	Биты 26—48
2	Адрес	Адрес

Размер команды: 9 байт. Операнд: значение в памяти по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=2, B=947, C=554):

```
0x9A, 0x1D, 0x00, 0xA8, 0x08, 0x00, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—2	Биты 3—25	Биты 26—33	Биты 34—56
7	Адрес	Смещение	Адрес

Размер команды: 9 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является сумма адреса (значение в памяти по адресу, которым является поле D) и смещения (поле C).

Тест (A=7, B=21, C=24, D=435):

```
0xAF, 0x00, 0x00, 0x60, 0xCC, 0x06, 0x00, 0x00, 0x00
```

Унарная операция: `sgn()`

A	**B**	**C**	**D**

A	B	C	D
Биты 0—2	Биты 3—10	Биты 11—33	Биты 34—56
0	Смещение	Адрес	Адрес

Размер команды: 9 байт. Операнд: значение в памяти по адресу, которым является поле D. Результат: значение в памяти по адресу, которым является сумма адреса (значение в памяти по адресу, которым является поле C) и смещения (поле B).

Тест (A=0, B=118, C=687, D=204):

```
0xB0, 0x7B, 0x15, 0x00, 0x30, 0x03, 0x00, 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 формат CSV. 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 тесте 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 число ассемблированных команд.
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 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 `sgn()`.
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 `sgn()` над вектором длины 6. 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 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.

Вариант №2

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

Спецификация УВМ

Загрузка константы

A	**B**
Биты 0—5	Биты 6—35
16	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=16, B=817):

```
0x50, 0xCC, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A
Биты 0—5
63

Размер команды: 1 байт. Операнд: значение в памяти по адресу, которым является регистр-аккумулятор. Результат: регистр-аккумулятор.

Тест (A=63):

```
0x3F
```

Запись значения в память

A	**B**
Биты 0—5	Биты 6—32
14	Адрес

Размер команды: 5 байт. Операнд: регистр-аккумулятор. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=14, B=502):

```
0x8E, 0x7D, 0x00, 0x00, 0x00
```

Бинарная операция: "<="

A	**B**
Биты 0—5	Биты 6—32
33	Адрес

Размер команды: 5 байт. Первый операнд: регистр-аккумулятор. Второй операнд: значение в памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=33, B=114):

```
0xA1, 0x1C, 0x00, 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 текстовое представление в духе традиционных ассемблеров (`мнемоника аргумент, аргумент, ...`). 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 тесте 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 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 `"<="`.

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 `"<="` над двумя векторами длины 5. 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.

Вариант №3

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—11	Биты 12—34
138	Адрес	Константа

Размер команды: 5 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=138, B=12, C=11):

```
0x8A, 0xBC, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—7	Биты 8—11	Биты 12—41
123	Адрес	Адрес

Размер команды: 6 байт. Операнд: значение в памяти по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=123, B=15, C=702):

```
0x7B, 0xEF, 0x2B, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—7	Биты 8—11	Биты 12—41
133	Адрес	Адрес

Размер команды: 6 байт. Операнд: регистр по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=133, B=5, C=43):

```
0x85, 0xB5, 0x02, 0x00, 0x00, 0x00
```

Унарная операция: побитовое "не"

A	**B**	**C**
Биты 0—7	Биты 8—11	Биты 12—41
217	Адрес	Адрес

Размер команды: 6 байт. Операнд: регистр по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=217, B=0, C=961):

```
0xD9, 0x10, 0x3C, 0x00, 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 формат JSON. 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 тесте 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 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 `побитовое "не"`.

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 `побитовое "не"` над вектором длины 8. 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 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.

Вариант №4

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—5	Биты 6—17	Биты 18—47
59	Адрес	Константа

Размер команды: 6 байт. Операнд: поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=59, B=449, C=937):

```
0x7B, 0x70, 0xA4, 0x0E, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—5	Биты 6—17	Биты 18—29	Биты 30—44
21	Адрес	Адрес	Смещение

Размер команды: 6 байт. Операнд: значение в памяти по адресу, которым является сумма адреса (значение в памяти по адресу, которым является поле B) и смещения (поле D). Результат: значение в памяти по адресу, которым является поле C.

Тест (A=21, B=589, C=802, D=62):

```
0x55, 0x93, 0x88, 0x8C, 0x0F, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—5	Биты 6—17	Биты 18—29	Биты 30—44
26	Адрес	Адрес	Смещение

Размер команды: 6 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является сумма адреса (значение в памяти по адресу, которым является поле C) и смещения (поле D).

Тест (A=26, B=824, C=709, D=834):

```
0x1A, 0xCE, 0x14, 0x8B, 0xD0, 0x00
```

Бинарная операция: умножение

A	B	C	D
Биты 0—5	Биты 6—17	Биты 18—29	Биты 30—41
3	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 поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=3, B=490, C=19, D=801):

```
0x83, 0x7A, 0x4C, 0x40, 0xC8, 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 текстовое представление в духе традиционных ассемблеров (`мнемоника аргумент, аргумент, ...`). 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 тесте 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 число ассемблированных команд.
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 CSV.
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 умножение.
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 умножение над двумя векторами длины 7. Результат записать во второй 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.

Вариант №5

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—3	Биты 4—6	Биты 7—26
2	Адрес	Константа

Размер команды: 4 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=2, B=6, C=640):

```
0x62, 0x40, 0x01, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—3	Биты 4—6	Биты 7—18	Биты 19—21
7	Адрес	Смещение	Адрес

Размер команды: 3 байт. Операнд: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле B) и смещения (поле C). Результат: регистр по адресу, которым является поле D.

Тест (A=7, B=6, C=113, D=7):

```
0xE7, 0x38, 0x38
```

Запись значения в память

A	**B**	**C**
Биты 0—3	Биты 4—32	Биты 33—35
4	Адрес	Адрес

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

Тест (A=4, B=2, C=2):

```
0x24, 0x00, 0x00, 0x00, 0x04
```

Бинарная операция: `pow()`

A	**B**	**C**	**D**
Биты 0—3	Биты 4—6	Биты 7—18	Биты 19—21
0	Адрес	Смещение	Адрес

Размер команды: 3 байт. Первый операнд: регистр по адресу, которым является поле B. Второй операнд: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле D) и смещения (поле C). Результат: регистр по адресу, которым является поле B.

Тест (A=0, B=7, C=867, D=7):

```
0xF0, 0xB1, 0x39
```

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 JSON. 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 тесте 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 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 CSV.
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 `pow()`.

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 `pow()` над вектором длины 6 и числом 57. Результат записать в исходный 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.

Вариант №6

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—10	Биты 11—31
10	Адрес	Константа

Размер команды: 4 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=10, B=13, C=508):

```
0x8A, 0xE6, 0x0F, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—6	Биты 7—31	Биты 32—35
37	Адрес	Адрес

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

Тест (A=37, B=934, C=14):

```
0x25, 0xD3, 0x01, 0x00, 0x0E
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—6	Биты 7—10	Биты 11—14	Биты 15—20
83	Адрес	Адрес	Смещение

Размер команды: 3 байт. Операнд: регистр по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле B) и смещения (поле D).

Тест (A=83, B=13, C=6, D=7):

```
0xD3, 0xB6, 0x03
```

Бинарная операция: "<="

A	**B**	**C**
Биты 0—6	Биты 7—10	Биты 11—14
111	Адрес	Адрес

Размер команды: 2 байт. Первый операнд: регистр по адресу, которым является поле C. Второй операнд: значение в памяти по адресу, которым является регистр по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=111, B=6, C=4):

```
0x6F, 0x23
```

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 JSON. 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 тесте 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 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 JSON.
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 `"<="`.

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 `"<="` над вектором длины 5 и числом 73. Результат записать в исходный 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.

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 B. 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 B. 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 B).

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 B. 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. 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 тесте 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 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 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.

Вариант №8

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—5	Биты 6—8	Биты 9—38
51	Адрес	Константа

Размер команды: 5 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=51, B=0, C=330):

```
0x33, 0x94, 0x02, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—5	Биты 6—8	Биты 9—22	Биты 23—25
60	Адрес	Смещение	Адрес

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

Тест (A=60, B=4, C=951, D=2):

```
0x3C, 0x6F, 0x07, 0x01
```

Запись значения в память

A	**B**	**C**
Биты 0—5	Биты 6—8	Биты 9—11
20	Адрес	Адрес

Размер команды: 2 байт. Операнд: регистр по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является регистр по адресу, которым является поле B.

Тест (A=20, B=6, C=5):

```
0x94, 0x0B
```

Бинарная операция: `взятие остатка`

A	**B**	**C**	**D**
Биты 0—5	Биты 6—8	Биты 9—26	Биты 27—29

A	B	C	D
40	Адрес	Адрес	Адрес

Размер команды: 4 байт. Первый операнд: значение в памяти по адресу, которым является поле C. Второй операнд: регистр по адресу, которым является поле D. Результат: регистр по адресу, которым является поле B.

Тест (A=40, B=3, C=620, D=1):

```
0xE8, 0xD8, 0x04, 0x08
```

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 тесте 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 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 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 `взятие остатка`.
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 `взятие остатка` над вектором длины 9 и числом 25. Результат записать в исходный 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.

Вариант №9

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—5	Биты 6—12	Биты 13—41
10	Адрес	Константа

Размер команды: 6 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=10, B=5, C=887):

```
0x4A, 0xE1, 0x6E, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—5	Биты 6—12	Биты 13—19	Биты 20—24
35	Адрес	Адрес	Смещение

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

Тест (A=35, B=0, C=105, D=16):

```
0x23, 0x20, 0x0D, 0x01
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—5	Биты 6—12	Биты 13—19	Биты 20—24
36	Адрес	Адрес	Смещение

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

Тест (A=36, B=127, C=27, D=26):

```
0xE4, 0x7F, 0xA3, 0x01
```

Унарная операция: `sqrt()`

A	**B**	**C**
Биты 0—5	Биты 6—12	Биты 13—19

A	B	C
9	Адрес	Адрес

Размер команды: 3 байт. Операнд: значение в памяти по адресу, которым является регистр по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=9, B=22, C=46):

```
0x89, 0xC5, 0x05
```

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 JSON. 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 тесте 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 число ассемблированных команд.
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 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 `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 6. Результат записать в исходный 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.

Вариант №10

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—4	Биты 5—27	Биты 28—47
1	Адрес	Константа

Размер команды: 6 байт. Операнд: поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=1, B=855, C=994):

```
0xE1, 0x6A, 0x00, 0x20, 0x3E, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—4	Биты 5—27	Биты 28—50
30	Адрес	Адрес

Размер команды: 7 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=30, B=569, C=386):

```
0x3E, 0x47, 0x00, 0x20, 0x18, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—4	Биты 5—11	Биты 12—34	Биты 35—57
10	Смещение	Адрес	Адрес

Размер команды: 8 байт. Операнд: значение в памяти по адресу, которым является поле D. Результат: значение в памяти по адресу, которым является сумма адреса (значение в памяти по адресу, которым является поле C) и смещения (поле B).

Тест (A=10, B=96, C=322, D=351):

```
0x0A, 0x2C, 0x14, 0x00, 0xF8, 0x0A, 0x00, 0x00
```

Унарная операция: `унарный минус`

A	**B**	**C**

A	**B**	**C**
Биты 0—4	Биты 5—27	Биты 28—50
6	Адрес	Адрес

Размер команды: 7 байт. Операнд: значение в памяти по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является значение в памяти по адресу, которым является поле B.

Тест (A=6, B=418, C=223):

```
0x46, 0x34, 0x00, 0xF0, 0x0D, 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 формат CSV. 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 тесте 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 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 `унарный минус`.
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 `унарный минус` над вектором длины 7. Результат записать в исходный 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.

Вариант №11

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—19	Биты 20—25
100	Константа	Адрес

Размер команды: 4 байт. Операнд: поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=100, B=455, C=33):

```
0x64, 0xC7, 0x11, 0x02
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—7	Биты 8—30	Биты 31—36
89	Адрес	Адрес

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

Тест (A=89, B=109, C=50):

```
0x59, 0x6D, 0x00, 0x00, 0x19
```

Запись значения в память

A	**B**	**C**
Биты 0—7	Биты 8—30	Биты 31—36
191	Адрес	Адрес

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

Тест (A=191, B=291, C=49):

```
0xBF, 0x23, 0x01, 0x80, 0x18
```

Бинарная операция: побитовое исключающее "или"

A	**B**	**C**	**D**
Биты 0—7	Биты 8—13	Биты 14—23	Биты 24—29
50	Адрес	Смещение	Адрес

Размер команды: 4 байт. Первый операнд: регистр по адресу, которым является поле D. Второй операнд: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле B) и смещения (поле C). Результат: регистр по адресу, которым является поле D.

Тест (A=50, B=35, C=37, D=55):

```
0x32, 0x63, 0x09, 0x37
```

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 JSON. 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 тесте 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 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 CSV.
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 `побитовое исключающее "или"`.

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 `побитовое исключающее "или"` над вектором длины 9 и числом 197. 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 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.

Вариант №12

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—10	Биты 11—32
46	Адрес	Константа

Размер команды: 5 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=46, B=4, C=1019):

```
0x2E, 0xDA, 0x1F, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—6	Биты 7—18	Биты 19—22
67	Адрес	Адрес

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

Тест (A=67, B=372, C=10):

```
0x43, 0xBA, 0x50, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—6	Биты 7—10	Биты 11—14
52	Адрес	Адрес

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

Тест (A=52, B=14, C=15):

```
0x34, 0x7F, 0x00, 0x00, 0x00
```

Унарная операция: унарный минус

A	**B**	**C**
Биты 0—6	Биты 7—10	Биты 11—14
6	Адрес	Адрес

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

Тест (A=6, B=15, C=2):

```
0x86, 0x17, 0x00, 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 формат JSON. 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 тесте 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 число ассемблированных команд.
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 JSON.
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 `унарный минус`.

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 `унарный минус` над вектором длины 8. Результат записать в исходный 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.

Вариант №13

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

Спецификация УВМ

Загрузка константы

A	**B**
Биты 0—6	Биты 7—15
34	Константа

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

Тест (A=34, B=391):

```
0xA2, 0xC3, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—6	Биты 7—20
33	Адрес

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

Тест (A=33, B=241):

```
0xA1, 0x78, 0x00
```

Запись значения в память

A	**B**
Биты 0—6	Биты 7—17
80	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 B).

Тест (A=80, B=301):

```
0xD0, 0x96, 0x00
```

Унарная операция: `popcnt()`

A	**B**
Биты 0—6	Биты 7—17
14	Смещение

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

Тест (A=14, B=746):

```
0x0E, 0x75, 0x01
```

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 текстовое представление в духе традиционных ассемблеров (`мнемоника аргумент, аргумент, ...`). 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 тесте 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 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 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 `popcnt()`.

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 `popcnt()` над вектором длины 9. 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 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.

Вариант №14

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

Спецификация УВМ

Загрузка константы

A	**B**
Биты 0—7	Биты 8—32
203	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=203, B=937):

```
0xCB, 0xA9, 0x03, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—7	Биты 8—22
0	Адрес

Размер команды: 5 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: новый элемент на стеке.

Тест (A=0, B=332):

```
0x00, 0x4C, 0x01, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—7	Биты 8—22
184	Смещение

Размер команды: 5 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 B).

Тест (A=184, B=774):

```
0xB8, 0x06, 0x03, 0x00, 0x00
```

Бинарная операция: `"=="`

A
Биты 0—7
86

Размер команды: 5 байт. Первый операнд: элемент, снятый с вершины стека. Второй операнд: элемент, снятый с вершины стека. Результат: значение в памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=86):

```
0x56, 0x00, 0x00, 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 формат YAML. 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 тесте 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 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 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 `"=="`.

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 "==" над двумя векторами длины 5. 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 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.

Вариант №15

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—3	Биты 4—7	Биты 8—16
9	Адрес	Константа

Размер команды: 3 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=9, B=11, C=439):

```
0xB9, 0xB7, 0x01
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—3	Биты 4—26	Биты 27—30
4	Адрес	Адрес

Размер команды: 4 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=4, B=122, C=6):

```
0xA4, 0x07, 0x00, 0x30
```

Запись значения в память

A	**B**	**C**
Биты 0—3	Биты 4—7	Биты 8—11
7	Адрес	Адрес

Размер команды: 2 байт. Операнд: регистр по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является регистр по адресу, которым является поле B.

Тест (A=7, B=13, C=0):

```
0xD7, 0x00
```

Бинарная операция: `побитовое исключающее "или"`

A	**B**	**C**
Биты 0—3	Биты 4—7	Биты 8—11
5	Адрес	Адрес

Размер команды: 2 байт. Первый операнд: регистр по адресу, которым является поле B. Второй операнд: значение в памяти по адресу, которым является регистр по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=5, B=1, C=8):

```
0x15, 0x08
```

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 CSV. 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 тесте 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 число ассемблированных команд.
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 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 `побитовое исключающее "или"`.

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 `побитовое исключающее "или"` над двумя векторами длины 9. 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 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.

Вариант №16

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

Спецификация УВМ

Загрузка константы

A	**B**
Биты 0—4	Биты 5—21
2	Константа

Размер команды: 3 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=2, B=142):

```
0xC2, 0x11, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—4	Биты 5—14
1	Адрес

Размер команды: 2 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=1, B=592):

```
0x01, 0x4A
```

Запись значения в память

A	**B**
Биты 0—4	Биты 5—14
18	Адрес

Размер команды: 2 байт. Операнд: регистр-аккумулятор. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=18, B=610):

```
0x52, 0x4C
```

Бинарная операция: побитовый циклический сдвиг влево

A	**B**
Биты 0—4	Биты 5—14
26	Адрес

Размер команды: 2 байт. Первый операнд: регистр-аккумулятор. Второй операнд: значение в памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=26, B=827):

```
0x7A, 0x67
```

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. 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 тесте 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 число ассемблированных команд.
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 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 `побитовый циклический сдвиг влево`.

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 `побитовый циклический сдвиг влево` над двумя векторами длины 9. Результат записать во второй 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.

Вариант №17

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

Спецификация УВМ

Загрузка константы

A	**B**
Биты 0—7	Биты 8—37
140	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=140, B=265):

```
0x8C, 0x09, 0x01, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—7	Биты 8—23
23	Адрес

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

Тест (A=23, B=53):

```
0x17, 0x35, 0x00
```

Запись значения в память

A
Биты 0—7
233

Размер команды: 1 байт. Операнд: элемент, снятый с вершины стека. Результат: значение в памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=233):

```
0xE9
```

Унарная операция: унарный минус

A
Биты 0—7
66

Размер команды: 1 байт. Операнд: элемент, снятый с вершины стека. Результат: значение в памяти по адресу, которым является элемент, снятый с вершины стека.

Тест (A=66):

```
0x42
```

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 CSV. 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 тесте 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 число ассемблированных команд.
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 `унарный минус`.

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 `унарный минус` над вектором длины 10. Результат записать в исходный 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.

Вариант №18

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—10	Биты 11—28
35	Адрес	Константа

Размер команды: 6 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=35, B=0, C=856):

```
0x23, 0xC0, 0x1A, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—6	Биты 7—10	Биты 11—20	Биты 21—24
91	Адрес	Смещение	Адрес

Размер команды: 6 байт. Операнд: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле D) и смещения (поле C). Результат: регистр по адресу, которым является поле B.

Тест (A=91, B=12, C=475, D=12):

```
0x5B, 0xDE, 0x8E, 0x01, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—6	Биты 7—10	Биты 11—14
36	Адрес	Адрес

Размер команды: 6 байт. Операнд: регистр по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является регистр по адресу, которым является поле B.

Тест (A=36, B=9, C=9):

```
0xA4, 0x4C, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `bswap()`

A	**B**	**C**	**D**
Биты 0—6	Биты 7—10	Биты 11—20	Биты 21—40

A	B	C	D
60	Адрес	Смещение	Адрес

Размер команды: 6 байт. Операнд: значение в памяти по адресу, которым является поле D. Результат: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле B) и смещения (поле C).

Тест (A=60, B=8, C=67, D=215):

```
0x3C, 0x1C, 0xE2, 0x1A, 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 формат CSV. 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 тесте 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 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 `bswap()`.
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 `bswap()` над вектором длины 8. Результат записать в исходный 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.

Вариант №19

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—2	Биты 3—17	Биты 18—43
0	Константа	Адрес

Размер команды: 12 байт. Операнд: поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=0, B=146, C=527):

```
0x90, 0x04, 0x3C, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—2	Биты 3—28	Биты 29—54
4	Адрес	Адрес

Размер команды: 12 байт. Операнд: значение в памяти по адресу, которым является значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=4, B=352, C=149):

```
0x04, 0x0B, 0x00, 0xA0, 0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—2	Биты 3—28	Биты 29—54
6	Адрес	Адрес

Размер команды: 12 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является значение в памяти по адресу, которым является поле C.

Тест (A=6, B=367, C=982):

```
0x7E, 0x0B, 0x00, 0xC0, 0x7A, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `abs()`

A	**B**	**C**
Биты 0—2	Биты 3—28	Биты 29—54

A	B	C
5	Адрес	Адрес

Размер команды: 12 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является значение в памяти по адресу, которым является поле C.

Тест (A=5, B=712, C=419):

```
0x45, 0x16, 0x00, 0x60, 0x34, 0x00, 0x00, 0x00, 0x00, 0x00, 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 текстовое представление в духе традиционных ассемблеров (`мнемоника аргумент, аргумент, ...`). 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 тесте 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 число ассемблированных команд.
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 `abs()`.
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 `abs()` над вектором длины 6. Результат записать в исходный 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.

Вариант №20

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

Спецификация УВМ

Загрузка константы

A	**B**
Биты 0—7	Биты 8—26
110	Константа

Размер команды: 6 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=110, B=681):

```
0x6E, 0xA9, 0x02, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—7	Биты 8—30
174	Адрес

Размер команды: 6 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=174, B=930):

```
0xAE, 0xA2, 0x03, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—7	Биты 8—30
22	Адрес

Размер команды: 6 байт. Операнд: регистр-аккумулятор. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=22, B=709):

```
0x16, 0xC5, 0x02, 0x00, 0x00, 0x00
```

Бинарная операция: ">"

A	**B**
Биты 0—7	Биты 8—30
50	Адрес

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

Тест (A=50, B=129):

```
0x32, 0x81, 0x00, 0x00, 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 тесте 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 число ассемблированных команд.
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 ">".

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 ">" над двумя векторами длины 6. 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 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.

Вариант №21

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—4	Биты 5—9	Биты 10—29
1	Адрес	Константа

Размер команды: 4 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=1, B=20, C=110):

```
0x81, 0xBA, 0x01, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—4	Биты 5—9	Биты 10—14
22	Адрес	Адрес

Размер команды: 2 байт. Операнд: значение в памяти по адресу, которым является регистр по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=22, B=4, C=27):

```
0x96, 0x6C
```

Запись значения в память

A	**B**	**C**
Биты 0—4	Биты 5—9	Биты 10—14
3	Адрес	Адрес

Размер команды: 2 байт. Операнд: регистр по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является регистр по адресу, которым является поле C.

Тест (A=3, B=9, C=21):

```
0x23, 0x55
```

Бинарная операция: побитовое "или"

A	**B**	**C**
Биты 0—4	Биты 5—9	Биты 10—20

A	B	C
17	Адрес	Адрес

Размер команды: 3 байт. Первый операнд: регистр по адресу, которым является поле B. Второй операнд: значение в памяти по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=17, B=15, C=574):

```
0xF1, 0xF9, 0x08
```

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 JSON. 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 тесте 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 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 JSON.
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 `побитовое "или"`.
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 `побитовое "или"` над двумя векторами длины 9. 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.

Вариант №22

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—2	Биты 3—24	Биты 25—32
4	Адрес	Константа

Размер команды: 10 байт. Операнд: поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=4, B=554, C=82):

```
0x54, 0x11, 0x00, 0xA4, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—2	Биты 3—24	Биты 25—46
6	Адрес	Адрес

Размер команды: 10 байт. Операнд: значение в памяти по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=6, B=184, C=664):

```
0xC6, 0x05, 0x00, 0x30, 0x05, 0x00, 0x00, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—2	Биты 3—24	Биты 25—34	Биты 35—56
2	Адрес	Смещение	Адрес

Размер команды: 10 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является сумма адреса (значение в памяти по адресу, которым является поле D) и смещения (поле C).

Тест (A=2, B=207, C=67, D=874):

```
0x7A, 0x06, 0x00, 0x86, 0x50, 0x1B, 0x00, 0x00, 0x00, 0x00
```

Бинарная операция: побитовый арифметический сдвиг вправо

A	**B**	**C**	**D**	**E**

A	B	C	D	E
Биты 0—2	Биты 3—24	Биты 25—34	Биты 35—56	Биты 57—78
1	Адрес	Смещение	Адрес	Адрес

Размер команды: 10 байт. Первый операнд: значение в памяти по адресу, которым является поле E. Второй операнд: значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является сумма адреса (значение в памяти по адресу, которым является поле D) и смещения (поле C).

Тест (A=1, B=89, C=485, D=183, E=945):

```
0xC9, 0x02, 0x00, 0xCA, 0xBB, 0x05, 0x00, 0x62, 0x07, 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 CSV. 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 тесте 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 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 JSON.
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 `побитовый арифметический сдвиг вправо`.
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 `побитовый арифметический сдвиг вправо` над двумя векторами длины 6. 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 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.

Вариант №23

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—5	Биты 6—27	Биты 28—31
31	Константа	Адрес

Размер команды: 4 байт. Операнд: поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=31, B=457, C=15):

```
0x5F, 0x72, 0x00, 0xF0
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—5	Биты 6—9	Биты 10—20
0	Адрес	Адрес

Размер команды: 4 байт. Операнд: значение в памяти по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=0, B=13, C=12):

```
0x40, 0x33, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—5	Биты 6—9	Биты 10—13
5	Адрес	Адрес

Размер команды: 4 байт. Операнд: регистр по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является регистр по адресу, которым является поле B.

Тест (A=5, B=5, C=5):

```
0x45, 0x15, 0x00, 0x00
```

Унарная операция: `sqrt()`

A	**B**	**C**
Биты 0—5	Биты 6—9	Биты 10—13
59	Адрес	Адрес

Размер команды: 4 байт. Операнд: значение в памяти по адресу, которым является регистр по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=59, B=4, C=2):

```
0x3B, 0x09, 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 формат JSON. 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 тесте 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 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 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 `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 исходный 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.

Вариант №24

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—3	Биты 4—11	Биты 12—18
7	Константа	Адрес

Размер команды: 6 байт. Операнд: поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=7, B=209, C=94):

```
0x17, 0xED, 0x05, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—3	Биты 4—10	Биты 11—42
6	Адрес	Адрес

Размер команды: 6 байт. Операнд: значение в памяти по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=6, B=94, C=422):

```
0xE6, 0x35, 0x0D, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—3	Биты 4—10	Биты 11—17
12	Адрес	Адрес

Размер команды: 6 байт. Операнд: регистр по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является регистр по адресу, которым является поле B.

Тест (A=12, B=13, C=10):

```
0xDC, 0x50, 0x00, 0x00, 0x00, 0x00
```

Бинарная операция: деление

A	**B**	**C**
Биты 0—3	Биты 4—10	Биты 11—17
8	Адрес	Адрес

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

Тест (A=8, B=18, C=116):

```
0x28, 0xA1, 0x03, 0x00, 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 текстовое представление в духе традиционных ассемблеров (`мнемоника аргумент, аргумент, ...`). 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 тесте 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 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 JSON.
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 деление.
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 деление над вектором длины 10 и числом 77. 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 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.

Вариант №25

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—12	Биты 13—22
206	Адрес	Константа

Размер команды: 3 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=206, B=28, C=254):

```
0xCE, 0xDC, 0x1F
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—7	Биты 8—12	Биты 13—17
25	Адрес	Адрес

Размер команды: 3 байт. Операнд: значение в памяти по адресу, которым является регистр по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=25, B=22, C=2):

```
0x19, 0x56, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—7	Биты 8—12	Биты 13—17
114	Адрес	Адрес

Размер команды: 3 байт. Операнд: регистр по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является регистр по адресу, которым является поле B.

Тест (A=114, B=21, C=10):

```
0x72, 0x55, 0x01
```

Бинарная операция: побитовый циклический сдвиг влево

A	**B**	**C**	**D**
Биты 0—7	Биты 8—12	Биты 13—17	Биты 18—22

A	B	C	D
161	Адрес	Адрес	Адрес

Размер команды: 3 байт. Первый операнд: значение в памяти по адресу, которым является регистр по адресу, которым является поле C. Второй операнд: значение в памяти по адресу, которым является регистр по адресу, которым является поле D. Результат: регистр по адресу, которым является поле B.

Тест (A=161, B=29, C=18, D=8):

```
0xA1, 0x5D, 0x22
```

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 CSV. 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 тесте 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 число ассемблированных команд.
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 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 `побитовый циклический сдвиг влево`.
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 `побитовый циклический сдвиг влево` 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.

Вариант №26

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

Спецификация УВМ

Загрузка константы

A	**B**
Биты 0—3	Биты 4—25
15	Константа

Размер команды: 4 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=15, B=880):

```
0x0F, 0x37, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—3	Биты 4—17
7	Смещение

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

Тест (A=7, B=63):

```
0xF7, 0x03, 0x00
```

Запись значения в память

A	**B**
Биты 0—3	Биты 4—17
13	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 B).

Тест (A=13, B=382):

```
0xED, 0x17, 0x00
```

Бинарная операция: побитовое исключающее "или"

A	**B**
Биты 0—3	Биты 4—17
5	Смещение

Размер команды: 3 байт. Первый операнд: значение в памяти по адресу, которым является сумма адреса (элемент, снятый с вершины стека) и смещения (поле B). Второй операнд: элемент, снятый с вершины стека. Результат: новый элемент на стеке.

Тест (A=5, B=583):

```
0x75, 0x24, 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 CSV. 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 тесте 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 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 `побитовое исключающее "или"`.

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 `побитовое исключающее "или"` над двумя векторами длины 9. 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.

Вариант №27

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

Спецификация УВМ

Загрузка константы

A	**B**
Биты 0—7	Биты 8—36
87	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: новый элемент на стеке.

Тест (A=87, B=683):

```
0x57, 0xAB, 0x02, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—7	Биты 8—15
194	Смещение

Размер команды: 2 байт. Операнд: значение в памяти по адресу, которым является сумма адреса (элемент, снятый с вершины стека) и смещения (поле B). Результат: новый элемент на стеке.

Тест (A=194, B=96):

```
0xC2, 0x60
```

Запись значения в память

A	**B**
Биты 0—7	Биты 8—15
243	Смещение

Размер команды: 2 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 B).

Тест (A=243, B=74):

```
0xF3, 0x4A
```

Унарная операция: `побитовое "не"`

A	**B**
Биты 0—7	Биты 8—18
147	Адрес

Размер команды: 3 байт. Операнд: элемент, снятый с вершины стека. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=147, B=150):

```
0x93, 0x96, 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 тесте 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 число ассемблированных команд.
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 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 `побитовое "не"`.

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 `побитовое "не"` над вектором длины 7. Результат записать в исходный 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.

Вариант №28

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

Спецификация УВМ

Загрузка константы

A	**B**
Биты 0—2	Биты 3—19
4	Константа

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

Тест (A=4, B=24):

```
0xC4, 0x00, 0x00
```

Чтение значения из памяти

A
Биты 0—2
3

Размер команды: 1 байт. Операнд: значение в памяти по адресу, которым является элемент, снятый с вершины стека. Результат: новый элемент на стеке.

Тест (A=3):

```
0x03
```

Запись значения в память

A	**B**
Биты 0—2	Биты 3—9
0	Смещение

Размер команды: 2 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 B).

Тест (A=0, B=37):

```
0x28, 0x01
```

Унарная операция: `abs()`

A
Биты 0—2
6

Размер команды: 1 байт. Операнд: элемент, снятый с вершины стека. Результат: новый элемент на стеке.

Тест (A=6):

```
0x06
```

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 тесте 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 число ассемблированных команд.
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 JSON.
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 `abs()`.

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 `abs()` над вектором длины 7. Результат записать в исходный 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.

Вариант №29

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

Спецификация УВМ

Загрузка константы

A	**B**
Биты 0—3	Биты 4—33
4	Константа

Размер команды: 9 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=4, B=851):

```
0x34, 0x35, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—3	Биты 4—19
0	Смещение

Размер команды: 9 байт. Операнд: значение в памяти по адресу, которым является сумма адреса (регистр-аккумулятор) и смещения (поле B). Результат: регистр-аккумулятор.

Тест (A=0, B=60):

```
0xC0, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—3	Биты 4—34
15	Адрес

Размер команды: 9 байт. Операнд: регистр-аккумулятор. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=15, B=289):

```
0x1F, 0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `sqrt()`

A	**B**	**C**
Биты 0—3	Биты 4—34	Биты 35—65
13	Адрес	Адрес

Размер команды: 9 байт. Операнд: значение в памяти по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=13, B=162, C=902):

```
0x2D, 0x0A, 0x00, 0x00, 0x30, 0x1C, 0x00, 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 формат YAML. 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 тесте 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 число ассемблированных команд.
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 6. Результат записать в исходный 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.

Вариант №30

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

Спецификация УВМ

Загрузка константы

A	**B**
Биты 0—3	Биты 4—15
2	Константа

Размер команды: 3 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=2, B=121):

```
0x92, 0x07, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—3	Биты 4—16
0	Адрес

Размер команды: 3 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=0, B=687):

```
0xF0, 0x2A, 0x00
```

Запись значения в память

A	**B**
Биты 0—3	Биты 4—16
9	Адрес

Размер команды: 3 байт. Операнд: регистр-аккумулятор. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=9, B=848):

```
0x09, 0x35, 0x00
```

Унарная операция: `bitreverse()`

A	**B**
Биты 0—3	Биты 4—16
15	Адрес

Размер команды: 3 байт. Операнд: регистр-аккумулятор. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=15, B=631):

```
0x7F, 0x27, 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 JSON. 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 тесте 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 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 `bitreverse()`.
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 `bitreverse()` над вектором длины 8. 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 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.

Вариант №31

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—29	Биты 30—33
36	Константа	Адрес

Размер команды: 5 байт. Операнд: поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=36, B=637, C=4):

```
0xA4, 0x3E, 0x01, 0x00, 0x01
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—6	Биты 7—10	Биты 11—23
122	Адрес	Адрес

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

Тест (A=122, B=13, C=97):

```
0xFA, 0x0E, 0x03, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—6	Биты 7—19	Биты 20—23
79	Адрес	Адрес

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

Тест (A=79, B=242, C=14):

```
0x4F, 0x79, 0xE0, 0x00, 0x00
```

Бинарная операция: умножение

A	**B**	**C**
Биты 0—6	Биты 7—19	Биты 20—23
29	Адрес	Адрес

Размер команды: 5 байт. Первый операнд: значение в памяти по адресу, которым является поле B. Второй операнд: регистр по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=29, B=1012, C=13):

```
0x1D, 0xFA, 0xD1, 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 текстовое представление в духе традиционных ассемблеров (`мнемоника аргумент, аргумент, ...`). 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 тесте 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 число ассемблированных команд.
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 умножение.

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 умножение над вектором длины 9 и числом 219. Результат записать в исходный 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.

Вариант №32

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—27	Биты 28—55
18	Адрес	Константа

Размер команды: 7 байт. Операнд: поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=18, B=727, C=94):

```
0x92, 0x6B, 0x01, 0xE0, 0x05, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—6	Биты 7—13	Биты 14—34	Биты 35—55
120	Смещение	Адрес	Адрес

Размер команды: 7 байт. Операнд: значение в памяти по адресу, которым является сумма адреса (значение в памяти по адресу, которым является поле D) и смещения (поле B). Результат: значение в памяти по адресу, которым является поле C.

Тест (A=120, B=60, C=679, D=510):

```
0x78, 0xDE, 0xA9, 0x00, 0xF0, 0x0F, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—6	Биты 7—27	Биты 28—48
62	Адрес	Адрес

Размер команды: 7 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является значение в памяти по адресу, которым является поле C.

Тест (A=62, B=972, C=832):

```
0x3E, 0xE6, 0x01, 0x00, 0x34, 0x00, 0x00
```

Унарная операция: `sqrt()`

A	**B**	**C**

A	B	C
Биты 0—6	Биты 7—27	Биты 28—48
81	Адрес	Адрес

Размер команды: 7 байт. Операнд: значение в памяти по адресу, которым является значение в памяти по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является значение в памяти по адресу, которым является поле B.

Тест (A=81, B=462, C=465):

```
0x51, 0xE7, 0x00, 0x10, 0x1D, 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 тесте 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 число ассемблированных команд.
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 6. Результат записать в исходный 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.

Вариант №33

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

Спецификация УВМ

Загрузка константы

A	**B**
Биты 0—4	Биты 5—36
28	Константа

Размер команды: 5 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=28, B=835):

```
0x7C, 0x68, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**
Биты 0—4	Биты 5—16
2	Смещение

Размер команды: 5 байт. Операнд: значение в памяти по адресу, которым является сумма адреса (регистр-аккумулятор) и смещения (поле B). Результат: регистр-аккумулятор.

Тест (A=2, B=601):

```
0x22, 0x4B, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**
Биты 0—4	Биты 5—32
4	Адрес

Размер команды: 5 байт. Операнд: регистр-аккумулятор. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=4, B=310):

```
0xC4, 0x26, 0x00, 0x00, 0x00
```

Бинарная операция: ">"

A	**B**
Биты 0—4	Биты 5—32
9	Адрес

Размер команды: 5 байт. Первый операнд: регистр-аккумулятор. Второй операнд: значение в памяти по адресу, которым является поле B. Результат: регистр-аккумулятор.

Тест (A=9, B=990):

```
0xC9, 0x7B, 0x00, 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 формат CSV. 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 тесте 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 число ассемблированных команд.
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 ">".

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 ">" над вектором длины 7 и числом 132. 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 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.

Вариант №34

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—25	Биты 26—42
168	Константа	Адрес

Размер команды: 6 байт. Операнд: поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=168, B=840, C=42):

```
0xA8, 0x48, 0x03, 0xA8, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—7	Биты 8—24	Биты 25—41
170	Адрес	Адрес

Размер команды: 6 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=170, B=1018, C=681):

```
0xAA, 0xFA, 0x03, 0x52, 0x05, 0x00
```

Запись значения в память

A	**B**	**C**
Биты 0—7	Биты 8—24	Биты 25—41
159	Адрес	Адрес

Размер команды: 6 байт. Операнд: значение в памяти по адресу, которым является поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=159, B=771, C=550):

```
0x9F, 0x03, 0x03, 0x4C, 0x04, 0x00
```

Унарная операция: `popcnt()`

A	**B**	**C**
Биты 0—7	Биты 8—24	Биты 25—41

A	B	C
135	Адрес	Адрес

Размер команды: 6 байт. Операнд: значение в памяти по адресу, которым является значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=135, B=732, C=382):

```
0x87, 0xDC, 0x02, 0xFC, 0x02, 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 текстовое представление в духе традиционных ассемблеров (`мнемоника аргумент, аргумент, ...`). 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 тесте 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 число ассемблированных команд.
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 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 `popcnt()`.
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 `popcnt()` над вектором длины 5. 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 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.

Вариант №35

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—6	Биты 7—34	Биты 35—50
80	Адрес	Константа

Размер команды: 12 байт. Операнд: поле C. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=80, B=989, C=12):

```
0xD0, 0xEE, 0x01, 0x00, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—6	Биты 7—34	Биты 35—62
51	Адрес	Адрес

Размер команды: 12 байт. Операнд: значение в памяти по адресу, которым является значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=51, B=590, C=65):

```
0x33, 0x27, 0x01, 0x00, 0x08, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Запись значения в память

A	**B**	**C**	**D**
Биты 0—6	Биты 7—34	Биты 35—45	Биты 46—73
126	Адрес	Смещение	Адрес

Размер команды: 12 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является сумма адреса (значение в памяти по адресу, которым является поле D) и смещения (поле C).

Тест (A=126, B=942, C=336, D=52):

```
0x7E, 0xD7, 0x01, 0x00, 0x80, 0x0A, 0x0D, 0x00, 0x00, 0x00, 0x00, 0x00
```

Унарная операция: `sqrt()`

A	**B**	**C**

A	B	C
Биты 0—6	Биты 7—34	Биты 35—62
68	Адрес	Адрес

Размер команды: 12 байт. Операнд: значение в памяти по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=68, B=618, C=747):

```
0x44, 0x35, 0x01, 0x00, 0x58, 0x17, 0x00, 0x00, 0x00, 0x00, 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 текстовое представление в духе традиционных ассемблеров (`мнемоника аргумент, аргумент, ...`). 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 тесте 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 число ассемблированных команд.
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 9. 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 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.

Вариант №36

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—4	Биты 5—10	Биты 11—27
12	Адрес	Константа

Размер команды: 4 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=12, B=5, C=619):

```
0xAC, 0x58, 0x13, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—4	Биты 5—10	Биты 11—20	Биты 21—26
28	Адрес	Смещение	Адрес

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

Тест (A=28, B=18, C=458, D=38):

```
0x5C, 0x52, 0xCE, 0x04
```

Запись значения в память

A	**B**	**C**
Биты 0—4	Биты 5—10	Биты 11—29
11	Адрес	Адрес

Размер команды: 4 байт. Операнд: регистр по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является поле C.

Тест (A=11, B=18, C=216):

```
0x4B, 0xC2, 0x06, 0x00
```

Унарная операция: `sqrt()`

A	**B**	**C**
Биты 0—4	Биты 5—10	Биты 11—16
25	Адрес	Адрес

Размер команды: 4 байт. Операнд: значение в памяти по адресу, которым является регистр по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=25, B=59, C=6):

```
0x79, 0x37, 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 формат JSON. 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 тесте 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 число ассемблированных команд.
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 JSON.
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 `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 9. Результат записать в исходный 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.

Вариант №37

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—5	Биты 6—22	Биты 23—26
49	Константа	Адрес

Размер команды: 4 байт. Операнд: поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=49, B=245, C=3):

```
0x71, 0x3D, 0x80, 0x01
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—5	Биты 6—9	Биты 10—13
9	Адрес	Адрес

Размер команды: 2 байт. Операнд: значение в памяти по адресу, которым является регистр по адресу, которым является поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=9, B=9, C=9):

```
0x49, 0x26
```

Запись значения в память

A	**B**	**C**
Биты 0—5	Биты 6—9	Биты 10—33
4	Адрес	Адрес

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

Тест (A=4, B=9, C=204):

```
0x44, 0x32, 0x03, 0x00, 0x00
```

Унарная операция: `побитовое "не"`

A	**B**	**C**
Биты 0—5	Биты 6—9	Биты 10—13
30	Адрес	Адрес

Размер команды: 2 байт. Операнд: регистр по адресу, которым является поле B. Результат: регистр по адресу, которым является поле C.

Тест (A=30, B=3, C=3):

```
0xDE, 0x0C
```

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 JSON. 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 тесте 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 число ассемблированных команд.

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 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 `побитовое "не"`.

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 `побитовое "не"` над вектором длины 5. Результат записать в исходный 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.

Вариант №38

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—7	Биты 8—13	Биты 14—35
241	Адрес	Константа

Размер команды: 5 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=241, B=21, C=39):

```
0xF1, 0xD5, 0x09, 0x00, 0x00
```

Чтение значения из памяти

A	**B**	**C**
Биты 0—7	Биты 8—30	Биты 31—36
127	Адрес	Адрес

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

Тест (A=127, B=129, C=32):

```
0x7F, 0x81, 0x00, 0x00, 0x10
```

Запись значения в память

A	**B**	**C**
Биты 0—7	Биты 8—13	Биты 14—36
84	Адрес	Адрес

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

Тест (A=84, B=44, C=97):

```
0x54, 0x6C, 0x18, 0x00, 0x00
```

Бинарная операция: "<="

A	**B**	**C**	**D**
Биты 0—7	Биты 8—13	Биты 14—36	Биты 37—42
215	Адрес	Адрес	Адрес

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

Тест (A=215, B=40, C=490, D=38):

```
0xD7, 0xA8, 0x7A, 0x00, 0xC0, 0x04
```

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 JSON. 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 тесте 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 число ассемблированных команд.
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 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 `"<="`.

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 `"<="` над вектором длины 6 и числом 24. 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 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.

Вариант №39

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

Спецификация УВМ

Загрузка константы

A	**B**	**C**
Биты 0—4	Биты 5—9	Биты 10—31
1	Адрес	Константа

Размер команды: 4 байт. Операнд: поле C. Результат: регистр по адресу, которым является поле B.

Тест (A=1, B=21, C=155):

```
0xA1, 0x6E, 0x02, 0x00
```

Чтение значения из памяти

A	**B**	**C**	**D**
Биты 0—4	Биты 5—9	Биты 10—14	Биты 15—23
27	Адрес	Адрес	Смещение

Размер команды: 3 байт. Операнд: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле C) и смещения (поле D). Результат: регистр по адресу, которым является поле B.

Тест (A=27, B=4, C=28, D=174):

```
0x9B, 0x70, 0x57
```

Запись значения в память

A	**B**	**C**
Биты 0—4	Биты 5—9	Биты 10—14
11	Адрес	Адрес

Размер команды: 2 байт. Операнд: регистр по адресу, которым является поле B. Результат: значение в памяти по адресу, которым является регистр по адресу, которым является поле C.

Тест (A=11, B=7, C=13):

```
0xEB, 0x34
```

Унарная операция: `sqrt()`

A	**B**	**C**	**D**	**E**
Биты 0—4	Биты 5—13	Биты 14—18	Биты 19—23	Биты 24—32

A	B	C	D	E
18	Смещение	Адрес	Адрес	Смещение

Размер команды: 5 байт. Операнд: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле C) и смещения (поле B). Результат: значение в памяти по адресу, которым является сумма адреса (регистр по адресу, которым является поле D) и смещения (поле E).

Тест (A=18, B=242, C=25, D=28, E=497):

```
0x52, 0x5E, 0xE6, 0xF1, 0x01
```

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 JSON. 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 тесте 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 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 JSON.
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 `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 5. 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 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.

Вариант №40

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

Спецификация УВМ

Загрузка константы

A	**B**
Биты 0—3	Биты 4—22
8	Константа

Размер команды: 3 байт. Операнд: поле B. Результат: регистр-аккумулятор.

Тест (A=8, B=285):

```
0xD8, 0x11, 0x00
```

Чтение значения из памяти

A
Биты 0—3
10

Размер команды: 1 байт. Операнд: значение в памяти по адресу, которым является регистр-аккумулятор. Результат: регистр-аккумулятор.

Тест (A=10):

```
0x0A
```

Запись значения в память

A	**B**
Биты 0—3	Биты 4—32
9	Адрес

Размер команды: 5 байт. Операнд: регистр-аккумулятор. Результат: значение в памяти по адресу, которым является поле B.

Тест (A=9, B=150):

```
0x69, 0x09, 0x00, 0x00, 0x00
```

Унарная операция: побитовое "не"

A
Биты 0—3
14

Размер команды: 1 байт. Операнд: значение в памяти по адресу, которым является регистр-аккумулятор. Результат: регистр-аккумулятор.

Тест (A=14):

```
0x0E
```

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 тесте 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 число ассемблированных команд.

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 JSON.
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 `побитовое "не"`.

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 `побитовое "не"` над вектором длины 5. Результат записать в исходный 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.