Учебная виртуальная машина (УВМ) — Этап 1

1. Назначение

Этап 1 проекта посвящён созданию **ассемблера** для учебной виртуальной машины (УВМ).
Ассемблер переводит текстовую программу в **промежуточное представление (A/B/C)** и далее в **5-байтный машинный код**, строго соответствующий спецификации УВМ.

Инструмент реализован в Python в виде CLI-приложения.

---

2. Язык ассемблера

Ассемблер использует человекочитаемый синтаксис, похожий на высокоуровневые языки.

Формат инструкции

```
МНЕМОНИКА аргумент1, аргумент2
```

Допустимые аргументы

- десятичные числа: `123`
- шестнадцатеричные числа: `0xFF`
- константы, объявленные через `.set`

Комментарии

- `# комментарий`
- `; комментарий`
- inline: `LOAD_CONST 10, 20 # пример`

Поддерживаемые инструкции

Mnemonic	Описание	A	Формат
`LOAD_CONST value, addr`	загрузить константу в память	45	SHORT
`READ b_addr, c_addr`	чтение косвенным адресованием	55	FULL
`WRITE b_addr, c_addr`	запись значения в память	14	FULL
`BITREV b_addr, c_addr`	унарная операция bitreverse()	34	FULL

Директивы

```
.set NAME = VALUE
```

Пример:

```
.set X = 100
LOAD_CONST X, 500
```

---

3. Форматы команд УВМ

Каждая команда занимает **5 байт (40 бит)**.

Формат SHORT (LOAD_CONST)

Поле	Биты	Размер
A	0–5	6
B	6–21	16
C	22–38	17

Кодирование:

```
value = A | (B << 6) | (C << 22)
```

Формат FULL (READ, WRITE, BITREV)

Поле	Биты	Размер
A	0–5	6
B	6–22	17
C	23–39	17

Кодирование:

```
value = A | (B << 6) | (C << 23)
```

---

4. Использование ассемблера

Ассемблер — это Python CLI-приложение.

Запуск

```
python uvm_assembler.py input.asm output.bin [--test]
```

Аргументы

- `input.asm` — исходная текстовая программа
- `output.bin` — бинарный файл результата
- `--test` — вывод промежуточного представления (A/B/C и 5 байт)

---

5. Пример программы (test1.asm)

```
# Test program from specification
LOAD_CONST 146, 456
READ 679, 835
WRITE 716, 603
BITREV 313, 398
```

---

6. Пример запуска

```
python uvm_assembler.py test1.asm out.bin --test
```

Пример выводимых данных

```
Assembled program (intermediate representation):
Line   2: LOAD_CONST   A=45, B=146, C=456 -> 0xAD, 0x24, 0x00, 0x72, 0x00
Line   3: READ         A=55, B=679, C=835 -> 0xF7, 0xA9, 0x80, 0xA1, 0x01
Line   4: WRITE        A=14, B=716, C=603 -> 0x0E, 0xB3, 0x80, 0x2D, 0x01
Line   5: BITREV       A=34, B=313, C=398 -> 0x62, 0x4E, 0x00, 0xC7, 0x00
```

Все байты **полностью совпадают** с эталоном спецификации УВМ.

---

7. Результаты этапа

На этапе 1 выполнено:

- разработан человекочитаемый язык ассемблера;
- реализован корректный разбор команд и директив `.set`;
- создано промежуточное представление (A/B/C);
- реализовано точное побитовое кодирование команд УВМ;
- создан бинарный файл `output.bin`;
- реализован режим `--test` для отображения IR;
- подтверждено совпадение байтов с официальной спецификацией.

Понял! Давай продолжим **ваш README.md** и добавим раздел про Этап 2 **в том же стиле**, без переписывания всего предыдущего текста. Можно вставить прямо после раздела «7. Результаты этапа». Вот продолжение:

Этап 2. Интерпретатор и работа с памятью

1. Назначение

Этап 2 посвящён реализации **интерпретатора УВМ**, который выполняет инструкции из бинарного файла, управляет памятью и создаёт дамп памяти в формате XML. Интерпретатор работает с объединённой памятью команд и данных и поддерживает все инструкции УВМ.

---

2. Основные возможности

- Выполнение инструкций:
    - `LOAD_CONST` — загрузка константы в память
    - `READ` — чтение значения по косвенному адресу
    - `WRITE` — запись значения в память
    - `BITREV` — побитовое обращение значения
- Модель памяти: массив целых чисел фиксированного размера (по умолчанию 1024).
- Команды выполняются последовательно в порядке размещения в бинарном файле.

---

3. Дамп памяти

- Интерпретатор может сохранять содержимое памяти в **XML-файл**.
- Пользователь задаёт диапазон адресов для вывода дампа.
- Формат XML:

```
<memory>
  <cell addr="0">10</cell>
  <cell addr="1">20</cell>
  <cell addr="2">10</cell>
  <cell addr="3">20</cell>
  <cell addr="4">1342177280</cell>
</memory>
```

---

4. Использование интерпретатора

Запуск

```
python uvm_assembler.py --interpret <input.bin> <dump.xml> <start>-<end>
```

Аргументы

- `<input.bin>` — бинарный файл, созданный ассемблером (Этап 1)
- `<dump.xml>` — имя файла для сохранения дампа памяти
- `<start>-<end>` — диапазон адресов памяти для дампа (например, `0-16`)

Пример запуска

```
python uvm_assembler.py --interpret test2.bin dump.xml 0-10
```

Вывод:

```
Memory dumped to dump.xml (addresses 0-9)
```

Файл `dump.xml` будет содержать значения памяти с адресов 0–9.

---

5. Результаты этапа 2

- Реализован основной цикл интерпретатора.
- Создана модель объединённой памяти команд и данных.
- Реализованы команды `LOAD_CONST`, `READ`, `WRITE` и `BITREV`.
- Поддерживается дамп памяти в XML по заданному диапазону адресов.
- Проверено выполнение тестовой программы, которая копирует массив с одного адреса на другой и применяет `BITREV`.

Конечно! Вот продолжение README.md с описанием **Этапа 3**, в том же стиле, чтобы можно было вставить прямо после раздела «8.5 Результаты этапа 2»:

Этап 3. Реализация арифметико-логического устройства (АЛУ)

1. Назначение

Этап 3 посвящён расширению функциональности интерпретатора добавлением **арифметико-логических операций**. На данном этапе реализована команда `BITREV`, которая выполняет побитовое обращение числа в памяти.

---

2. Основные возможности

- **Команда BITREV**:
    - Читает значение из памяти по адресу C.
    - Выполняет побитовое обращение (реверс битов) для 32-битного числа.
    - Записывает результат в память по адресу B.
- Позволяет проверять работу АЛУ через тестовые программы и дамп памяти.

---

3. Пример программы для тестирования BITREV (`test3.asm`)

```
# Тест Этап 3: проверка BITREV
LOAD_CONST 10, 0       # Сохраняем 10 по адресу 0
BITREV 1, 0            # Побитовый реверс числа по адресу 0 -> адрес 1
```

Ожидаемый результат в памяти:

Адрес	Значение
0	10
1	1342177280

(побитовый реверс числа 10 для 32 бит)

---

4. Запуск теста

1. Ассемблируем программу:

```
python uvm_assembler.py --assemble test3.asm test3.bin --test
```

2. Выполняем интерпретацию и делаем дамп памяти:

```
python uvm_assembler.py --interpret test3.bin dump3.xml 0-2
```

Файл `dump_alu.xml` будет содержать:

```
<memory>
  <cell addr="0">10</cell>
  <cell addr="1">1342177280</cell>
</memory>
```

---

5. Результаты этапа 3

- Реализована команда BITREV в интерпретаторе.
- Создана тестовая программа, демонстрирующая корректность вычислений АЛУ.
- Результаты вычислений успешно сохраняются в память и проверяются через дамп XML.
- Подтверждена корректность побитового реверса 32-битных чисел.