

# **ВАРИАНТ №4**

## **Разработка кроссплатформенных программных систем**

Учебная виртуальная машина (УВМ)  
Ассемблер и интерпретатор

## КРАТКОЕ ОПИСАНИЕ РЕШЕНИЯ

### Коротко про вариант

- 3-байтные инструкции, поля A/B/C/D как в спецификации
- Команды: `load_const`, `read_mem`, `write_mem`, `sub_mem` (вычитание)
- Ассемблер: алгебраический синтаксис ( $rB = 5$ ,  $rB = \text{mem}[rC]$ ,  $\text{mem}[rB] = rC$ ,  $\text{mem}[rC + k] -= rD$ )

### Как использовать

```
python3 uvm_asm.py -i emu-program.csv -o emu-output.bin -t 1
```

Сборка программы из исходного кода

```
python3 uvm_interp.py -i emu-output.bin -o dump.xml -r "0-35"
```

Запуск интерпретатора с выводом дампа памяти

```
python3 uvm-ui.py
```

Запуск GUI-версии (Textual)

### Что получается

- Тестовая программа `emu-program.csv` делает поэлементное вычитание двух векторов длины 6
- После исполнения в памяти по адресам 0-5 лежит результат (пример: 9, 18, 27, 36, 45, 54), исходный второй вектор остаётся в 20-25
- XML-дамп пишется в `dump.xml`, там видны регистры и выбранный диапазон памяти

# СПЕЦИФИКАЦИЯ УВМ

Учебная виртуальная машина поддерживает 4 команды с фиксированной длиной 3 байта.

## Команда 1: Загрузка константы

Поле	Биты	Значение	Описание
A	0-3	14	Код операции
B	4-10	Адрес	Адрес регистра
C	11-22	Константа	Значение для загрузки

- Размер команды: 3 байта
- Операнд: поле C (константа)
- Результат: регистр по адресу из поля B

Тест (A=14, B=34, C=686): 0x2E, 0x72, 0x15

## Команда 2: Чтение значения из памяти

Поле	Биты	Значение	Описание
A	0-3	4	Код операции
B	4-10	Адрес	Адрес регистра назначения
C	11-17	Адрес	Адрес регистра с адресом памяти

- Размер команды: 3 байта
- Операнд: значение в памяти по адресу из регистра (адрес регистра в поле C)
- Результат: регистр по адресу из поля B

Тест (A=4, B=103, C=29): 0x74, 0xEE, 0x00

### Команда 3: Запись значения в память

Поле	Биты	Значение	Описание
A	0-3	2	Код операции
B	4-10	Адрес	Адрес регистра с адресом памяти
C	11-17	Адрес	Адрес регистра с данными

- Размер команды: 3 байта
- Операнд: регистр по адресу из поля C
- Результат: значение в памяти по адресу из регистра (адрес регистра в поле B)

Тест (A=2, B=74, C=62): 0xA2, 0xF4, 0x01

### Команда 4: Бинарная операция (вычитание)

Поле	Биты	Значение	Описание
A	0-3	1	Код операции
B	4-8	Смещение	Смещение для адреса
C	9-15	Адрес	Адрес регистра с базовым адресом
D	16-22	Адрес	Адрес регистра с вычитаемым

- Размер команды: 3 байта
- Первый операнд: значение в памяти по адресу (регистр[C] + смещение[B])
- Второй операнд: регистр по адресу из поля D
- Результат: значение в памяти по адресу (регистр[C] + смещение[B])

Тест (A=1, B=3, C=13, D=94): 0x31, 0x1A, 0x5E

# ЭТАПЫ РАЗРАБОТКИ

## 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 аргументы командной строки (путь к исходному файлу, путь к результату, режим тестирования)
2. Спроектировать человекочитаемый язык ассемблера с алгебраическим синтаксисом
3. Описать язык ассемблера в документации (README.md)
4. Реализовать транслятор в промежуточное представление
5. В режиме тестирования вывести промежуточное представление
6. Создать программу для тестов из спецификации UVM

## 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 в двоичный выходной файл
3. Вывести на экран число ассемблированных команд
4. В режиме тестирования вывести результат в байтовом формате
5. Создать файл на языке ассемблера для всех тестовых последовательностей

## Этап 3: Интерпретатор и операции с памятью

Цель: Создать цикл интерпретации, реализовать модель памяти UVM и выполнить базовые команды.

Требования:

1. Интерпретатор должен принимать аргументы (путь к программе, путь к дампу, диапазон адресов)

2. Использовать формат XML для дампа памяти
3. Реализовать модель памяти УВМ (разделение памяти команд и данных)
4. Реализовать основной цикл интерпретатора
5. Реализовать команды загрузки константы, чтения и записи в память
6. Написать тестовую программу копирования массива

## Этап 4: Реализация АЛУ

Цель: Завершить реализацию интерпретатора, добавив поддержку вычислительных операций.

Требования:

1. Реализовать выполнение команды вычитания
2. Написать и выполнить тестовую программу с корректными вычислениями

## Этап 5: Выполнение тестовой задачи

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

Требования:

1. Написать программу: поэлементное вычитание двух векторов длины 6
2. Создать три примера программ с различными данными
3. Продемонстрировать соответствие дампа памяти требованиям задачи

## Этап 6: Кроссплатформенное GUI-приложение

Цель: Реализовать кроссплатформенную версию УВМ.

Требования:

1. Разработать GUI-версию УВМ с редактируемым окном программы, окном дампа памяти и кнопкой запуска
2. Портировать GUI-версию на платформы: Windows, Linux, Web/WASM
3. Создать единый сборочный скрипт для всех платформ

## ОБЩИЕ ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ

1. Практические работы выполняются очно, защита каждого этапа происходит на семинарских занятиях
2. Этапы работы сохраняются в публично доступном git-репозитории
3. Каждый этап должен быть отражен в истории коммитов с детальными сообщениями
4. Студент самостоятельно выбирает язык реализации

### Документация (README.md) должна содержать:

- Общее описание
- Описание всех функций и настроек
- Команды для сборки проекта и запуска тестов
- Примеры использования

### Рекомендуемые git-сервисы:

- [github.com](https://github.com)
- [gitlab.com](https://gitlab.com)
- [bitbucket.org](https://bitbucket.org)