Вариант №10

Ассемблер для учебной виртуальной машины

Этап 1. Перевод программы в промежуточное представление

Описание

В рамках этапа 1 реализован CLI-ассемблер для учебной виртуальной машины (УВМ). Ассемблер принимает на вход текст программы, написанной на человекочитаемом языке ассемблера, и преобразует её во внутреннее (промежуточное) представление.

Промежуточное представление используется на следующем этапе для исполнения программы интерпретатором.

Поддерживаемые команды УВМ

Команда	Описание
`LOAD_CONST`	Загрузка константы в память
`READ`	Чтение значения из памяти
`WRITE`	Запись значения в память
`BITREV`	Унарная операция `bitreverse()`

Язык ассемблера

Используется алгебраический синтаксис, близкий к языкам высокого уровня.

Общий вид инструкции

```
ИНСТРУКЦИЯ(B, C)
```

где:

- `B` — первый операнд инструкции
- `C` — второй операнд инструкции

Пример программы

```
LOAD_CONST(146, 456)
READ(679, 835)
WRITE(716, 603)
BITREV(313, 398)
```

Промежуточное представление

Результатом работы ассемблера является список кортежей следующего вида:

```
[
    ("load_const", B, C),
    ("read", B, C),
    ("write", B, C),
    ("bitrev", B, C)
]
```

Пример промежуточного представления для тестовой программы:

```
[
    ('load_const', 146, 456),
    ('read', 679, 835),
    ('write', 716, 603),
    ('bitrev', 313, 398)
]
```

Использование

Аргументы командной строки

Аргумент	Описание
`--src`	Путь к файлу с исходным кодом на языке ассемблера
`--out`	Путь к файлу с промежуточным представлением
`--test`	Режим тестирования (вывод промежуточного представления на экран)

Пример запуска

Ассемблирование программы с выводом промежуточного представления:

```
python assembler.py --src test.asm --out program.ir --test
```

Тестирование

Для проверки корректности работы ассемблера используется программа, соответствующая тестам из спецификации УВМ.

Ассемблер корректно формирует промежуточное представление, совпадающее по структуре и значениям с ожидаемым результатом.

Результаты этапа

- Реализован CLI-ассемблер
- Спроектирован человекочитаемый язык ассемблера
- Реализован разбор команд и трансляция в промежуточное представление
- Добавлен режим тестирования
- Создан тест на основе спецификации УВМ

Интерпретатор учебной виртуальной машины

Этап 2. Интерпретатор и операции с памятью

Описание

В рамках этапа 2 реализован интерпретатор учебной виртуальной машины (УВМ). Интерпретатор выполняет программы, представленные в виде промежуточного представления, сформированного на этапе 1.

Реализована модель памяти УВМ, основной цикл интерпретации, а также базовые команды работы с памятью.

Модель памяти

Память УВМ представлена в виде одного линейного массива целых чисел.
Память команд и память данных объединены в единое адресное пространство.

Размер памяти по умолчанию — 1024 ячейки.

Поддерживаемые команды

Команда	Описание
`load_const`	Запись константы в память по адресу `C`
`read`	Чтение значения из памяти по адресу, хранящемуся в `mem[B]`, и запись результата в `mem[C]`
`write`	Чтение значения из памяти по адресу `B` и запись результата в `mem[C]`
`bitrev`	Побитовый реверс значения `mem[C]` и запись результата в `mem[B]`

Формат входных данных

Интерпретатор принимает на вход файл с промежуточным представлением, полученным на этапе 1.

Формат промежуточного представления — список кортежей:

```
[
    ('load_const', B, C),
    ('read', B, C),
    ('write', B, C),
    ('bitrev', B, C)
]
```

Использование

Аргументы командной строки

Аргумент	Описание
`--path`, `-p`	Путь к файлу с промежуточным представлением
`--dump`, `-d`	Путь к XML-файлу с дампом памяти
`--range`, `-r`	Диапазон адресов памяти для дампа (например, `0-32`)

Пример запуска

```
python interpreter.py --path program.ir --dump memory.xml --range 0-32
```

Формат дампа памяти

Результат выполнения программы сохраняется в XML-файл следующего вида:

```
<memory>
  <cell address="10">111</cell>
  <cell address="20">111</cell>
</memory>
```

Каждый элемент `<cell>` содержит адрес ячейки памяти и значение, хранящееся по этому адресу после выполнения программы.

Тестирование

Для проверки корректности работы интерпретатора используется тестовая программа, копирующая значение из одной ячейки памяти в другую.

Пример тестовой программы

```
LOAD_CONST(111, 10)
WRITE(10, 20)
```

После выполнения программы значение из адреса `10` корректно копируется в адрес `20`, что подтверждается дампом памяти.

Результаты этапа

- Реализован интерпретатор УВМ
- Реализована модель памяти с единым адресным пространством
- Реализован основной цикл интерпретации
- Реализованы базовые команды работы с памятью
- Реализован вывод дампа памяти в формате XML
- Проведено тестирование корректности работы интерпретатора

Арифметико-логическое устройство учебной виртуальной машины

Этап 3. Реализация АЛУ

Вариант №10

Описание

В рамках этапа 3 завершена реализация интерпретатора учебной виртуальной машины путём добавления поддержки вычислительных операций, выполняемых арифметико-логическим устройством (АЛУ).

Реализована вычислительная команда BITREV, выполняющая побитовый реверс значения, хранящегося в памяти.

Реализованная операция АЛУ

BITREV

Команда выполняет побитовый реверс значения из памяти.

Формат команды в промежуточном представлении:

```
('bitrev', B, C)
```

Поведение команды:

- берётся значение из ячейки памяти C
- выполняется побитовый реверс (32 бита)
- результат записывается в ячейку памяти B

Использование

Интерпретатор используется аналогично этапу 2 и принимает на вход файл с промежуточным представлением программы.

Аргументы командной строки

Аргумент	Описание

`--path, -p` Путь к файлу с промежуточным представлением
`--dump, -d` Путь к XML-файлу с дампом памяти
`--range, -r` Диапазон адресов памяти для дампа (например, `0-32`)

Тестирование

Для проверки корректности работы АЛУ реализована тестовая программа, демонстрирующая работу операции BITREV.

Тестовая программа

```
LOAD_CONST(13, 10)
BITREV(20, 10)
```

В данной программе:

- в ячейку памяти 10 загружается значение 13
- выполняется побитовый реверс значения из ячейки 10
- результат сохраняется в ячейке памяти 20

Результаты выполнения

После выполнения тестовой программы был получен следующий дамп памяти:

```
<memory><cell address="0">0</cell><cell address="1">0</cell><cell
address="2">0</cell><cell
address="3">0</cell><cell address="4">0</cell><cell address="5">0</cell><cell
address="6">0</cell><cell address="7">0</cell><cell address="8">0</cell><cell
address="9">0</cell><cell address="10">13</cell><cell address="11">0</cell>
<cell address="12">0</cell><cell address="13">0</cell><cell
address="14">0</cell><cell address="15">0</cell><cell address="16">0</cell>
<cell address="17">0</cell><cell address="18">0</cell><cell
address="19">0</cell><cell address="20">2952790016</cell><cell
address="21">0</cell><cell address="22">0</cell><cell address="23">0</cell>
<cell address="24">0</cell><cell address="25">0</cell><cell
address="26">0</cell><cell address="27">0</cell><cell address="28">0</cell>
<cell address="29">0</cell><cell address="30">0</cell><cell
address="31">0</cell></memory>
```

Полученный результат подтверждает корректность работы операции побитового реверса.

Результаты этапа

- Реализована поддержка арифметико-логического устройства
- Реализована вычислительная операция BITREV
- Интеграция АЛУ в основной цикл интерпретатора
- Проведено тестирование корректности вычислений
- Результаты вычислений сохраняются в память и доступны для проверки