МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический университет
Петра Великого»

Институт компьютерных наук и кибербезопасности

Направление: 02.03.01 Математика и компьютерные науки

Отчет по дисциплине: «Основы архитектуры ЦВМ»

Команды и способы адресации для x86 в 32-битном режиме» по дисциплине Основы архитектуры ЦВМ

Студент,
группы 5130201/40003 _____ Адиатуллин Т. Р

Руководитель,
Преподаватель _____ Вербова Н. М.

«____»_____ 2025 г.

Санкт-Петербург, 2025

Содержание

1 Цель работы

Познакомиться с системой процессорных команд и способами использования этих команд.

2 Методика

1. Взять фрагмент программы на языке Си, заданный преподавателем, или написать такой фрагмент самостоятельно по заданной спецификации.

2. Оттранслировать программу, содержащую этот фрагмент.

3. Перейти в режим отладки и изучить дизассемблированный код фрагмента.

4. Самостоятельно заменить обращение к элементам массива через индекс на обращение через указатель, перетранслировать программу, проанализировать и прокомментировать (описать), как компилятор реализовал обращение через указатель.

3 Порядок выполнения работы

1. Замените в программе-шаблоне участок кода «Циклический фрагмент на языке высокого уровня» на тот, который содержится в Вашем задании.

2. Оттранслируйте программу и перейдите в режим отладки.

3. Найдите и отметьте в результате трансляции (в последовательности команд процессора) участки, соответствующие отдельным инструкциям исходного текста на Си.

4. Опишите, как выбранные компилятором последовательности команд выполняют действия, заданные инструкцией языка Си.

5. Составьте список переменных, объявленных в программе, для каждой переменной отметьте, как она объявлена (локально, локально-static, глобально). После этого определите, где транслятор отвел место для каждой переменной. Нарисуйте «карту памяти» — расположение в памяти ваших переменных.

6. Найдите и отметьте в дизассемблере все команды, которые обращаются к этим переменным, для каждой команды укажите, к какой именно переменной она обращается.

7. Определите для каждой из команд, обращающихся к переменной:

(a) сколько операндов указано в адресной части,

(b) какие способы адресации использовал компилятор в каждом из адресных полей.

8. Найдите команды обращения к данным, в которых используется многокомпонентная адресация. Проинтерпретируйте значения отдельных компонент в этих командах.

9. Опишите способы адресации в командах перехода, которые использовал компилятор.

10. Замените в программе объявления глобальных переменных на локальные (либо, наоборот, в зависимости от того, что было в вашей программе). Проанализируйте и опишите, как изменились способы адресации в командах обращения к этим переменным.

11. Разобрать все команды по кодам байтов.

4 Исходный код на языке Си

```c
int Mas[10];
unsigned long i;

int main() {
    for (i = 0; i < 9; i++) {
        if (i != 6)
            Mas[i] = (17*i) & 0x0E;
        else
            Mas[i] = 0x1A * i/4;
    }
    return 0;
}
```

Листинг 1: Исходный код программы на языке C

Программа реализует следующую логику:

- Объявлен глобальный массив целых чисел `Mas` размером 10 элементов
- Объявлена глобальная переменная-счетчик `i` типа `unsigned long`
- В цикле for от 0 до 8 происходит заполнение массива по условию:
  - Если индекс не равен 6, то `Mas[i] = (17*i) & 0x0E`
  - Если индекс равен 6, то `Mas[i] = 0x1A * i / 4`

5 Дизассемблированный код

Дизассемблированный код представлен ниже:

```
00000000 <_main>:
; int main() {
       0: 55                    push    ebp               ; сохранение базового указателя
       1: 89 e5                 mov     ebp, esp          ; установка нового базового указателя
       3: 50                    push    eax               ; выделение места на стеке
       4: c7 45 fc 00 00 00 00  mov     dword ptr [ebp - 0x4], 0x0 ; инициализация локальной переменной
;    for (i = 0; i < 9; i++) {
       b: c7 05 00 00 00 00 00 00 00 00  mov     dword ptr [0x0], 0x0 ; i = 0 (i находится по абсолютному адресу 0x0)
       ; -- Проверка условия цикла --
      15: 83 3d 00 00 00 00 09  cmp     dword ptr [0x0], 0x9 ; сравнить i с 9
      1c: 73 48                 jae     0x66 <_main+0x66>  ; если i >= 9, перейти к return 0
;    if (i != 6)
      1e: 83 3d 00 00 00 00 06  cmp     dword ptr [0x0], 0x6 ; сравнить i с 6
      25: 74 18                 je      0x3f <_main+0x3f>  ; если i == 6, перейти к блоку else
;           Mas[i] = (17*i) & 0x0E;
      27: 6b 0d 00 00 00 00 11  imul    ecx, dword ptr [0x0], 0x11 ; ecx = i * 17
      2e: 83 e1 0e              and     ecx, 0xe          ; ecx = ecx & 14
      31: a1 00 00 00 00        mov     eax, dword ptr [0x0] ; загрузить i в eax (как индекс)
      36: 89 0c 85 00 00 00 00  mov     dword ptr [4*eax], ecx ; Mas[i] = ecx (адресация Mas + i*4)
      3d: eb 16                 jmp     0x55 <_main+0x55>  ; переход к инкременту i (пропуск else)
;           Mas[i] = 0x1A * i/4; (Блок ELSE)
      3f: 6b 0d 00 00 00 00 1a  imul    ecx, dword ptr [0x0], 0x1a ; ecx = i * 26
      46: c1 e9 02              shr     ecx, 0x2          ; ecx = ecx / 4 (сдвиг вправо)
      49: a1 00 00 00 00        mov     eax, dword ptr [0x0] ; загрузить i в eax (как индекс)
      4e: 89 0c 85 00 00 00 00  mov     dword ptr [4*eax], ecx ; Mas[i] = ecx (адресация Mas + i*4)
;    } (Конец тела цикла)
      55: eb 00                 jmp     0x57 <_main+0x57>  ; переход к следующей инструкции
;    for (i = 0; i < 9; i++) { (Инкремент)
      57: a1 00 00 00 00        mov     eax, dword ptr [0x0] ; загрузить i в eax
      5c: 83 c0 01              add     eax, 0x1          ; eax = i + 1
      5f: a3 00 00 00 00        mov     dword ptr [0x0], eax ; сохранить новое i
      64: eb af                 jmp     0x15 <_main+0x15>  ; переход к проверке условия
;    return 0;
      66: 31 c0                 xor     eax, eax          ; eax = 0 (возвращаемое значение)
      68: 83 c4 04              add     esp, 0x4          ; очистка стека
      6b: 5d                    pop     ebp               ; восстановление ebp
      6c: c3                    ret                       ; возврат из функции
```

Рис. 1: Часть дизассемблированного фрагмента

6 Список переменных и способы адресации

6.1 Глобальные переменные

В программе объявлены две глобальные переменные:

Переменная	**Тип**	**Размер**	**Адрес**
Mas	int[10]	40 байт	0x00000000 (база)
i	unsigned long	4 байта	0x00000000 (база)

Таблица 1: Глобальные переменные программы

Примечание: В дизассемблированном коде используются релокационные адреса (0x00000000), которые будут заменены линкером на реальные адреса в секции данных (.data или .bss).

6.2 Карта памяти массива Mas

Элемент	Смещение	Адрес
Mas[0]	0x00	base + 0x00
Mas[1]	0x04	base + 0x04
Mas[2]	0x08	base + 0x08
Mas[3]	0x0C	base + 0x0C
Mas[4]	0x10	base + 0x10
Mas[5]	0x14	base + 0x14
Mas[6]	0x18	base + 0x18
Mas[7]	0x1C	base + 0x1C
Mas[8]	0x20	base + 0x20
Mas[9]	0x24	base + 0x24

Таблица 2: Карта памяти массива Mas (каждый элемент — 4 байта)

6.3 Команды обращения к переменной i

№	Адрес	Команда	Операция
1	0x0b	mov dword ptr [0x0], 0x0	Запись (i = 0)
2	0x15	cmp dword ptr [0x0], 0x9	Чтение (i < 9)
3	0x1e	cmp dword ptr [0x0], 0x6	Чтение (i != 6)
4	0x27	imul ecx, dword ptr [0x0], 0x11	Чтение (17*i)
5	0x31	mov eax, dword ptr [0x0]	Чтение (индекс)
6	0x3f	imul ecx, dword ptr [0x0], 0x1a	Чтение (26*i)
7	0x49	mov eax, dword ptr [0x0]	Чтение (индекс)
8	0x57	mov eax, dword ptr [0x0]	Чтение (i++)
9	0x5f	mov dword ptr [0x0], eax	Запись (i++)

Таблица 3: Команды обращения к переменной i

Статистика обращений к переменной i:

- Всего команд: 9
- Операций чтения: 7
- Операций записи: 2
- Способ адресации: прямая абсолютная (direct addressing)

№	Адрес	Команда	Операция
1	0x36	mov dword ptr [4*eax], ecx	Запись Mas[i] (if)
2	0x4e	mov dword ptr [4*eax], ecx	Запись Mas[i] (else)

Таблица 4: Команды обращения к массиву Mas

6.4 Команды обращения к массиву Mas

6.5 Многокомпонентная адресация

Команда 0x36: `mov dword ptr [4*eax], ecx`
Интерпретация компонентов:

- **База (база_Mas):** 0x00000000 — адрес начала массива Mas (будет заменен линкером)
- **Индекс (EAX):** содержит значение переменной i
- **Масштаб:** 4 — размер одного элемента массива в байтах (sizeof(int) = 4)
- **Итоговый адрес:** база_Mas + EAX × 4 = адрес элемента Mas[i]

Команда 0x4e: `mov dword ptr [4*eax], ecx`
Аналогична команде 0x36, но используется в ветке else для записи результата вычисления `0x1A * i / 4`.

6.6 Локальные переменные

Обращение к локальной переменной использует **косвенную адресацию со смещением**:

- **Формат:** `dword ptr [ebp + смещение]`
- **Пример:** `mov dword ptr [ebp - 0x4], 0x0`
- **Компоненты:**
  - Базовый регистр: EBP (указатель на базу стекового кадра)
  - Смещение: -0x4 (4 байта вниз от базы стека)
- **Вычисление адреса:** Адрес = EBP - 4

Адрес	Команда	Тип	Назначение
0x1c	jae 0x66	Беззнаковый $\geq$	Выход из цикла
0x25	je 0x3f	Равенство	Переход к else

Таблица 5: Условные переходы

6.7 Условные переходы

Все условные переходы используют **относительную адресацию**:
Способ адресации: относительная адресация с 8-битным смещением

- Команда содержит смещение относительно следующей инструкции
- Целевой адрес = адрес_следующей_команды + смещение
- Пример для 0x1c: целевой адрес = 0x1e + 0x48 = 0x66

6.8 Безусловные переходы

Адрес	Команда	Назначение
0x3d	jmp 0x55	К инкременту i
0x55	jmp 0x57	К инкременту i
0x64	jmp 0x15	К проверке условия цикла

Таблица 6: Безусловные переходы

Способ адресации: относительная адресация

- Используется короткая форма jmp с 8-битным смещением
- Целевой адрес вычисляется относительно следующей инструкции
- Пример для 0x3d: jmp 0x55 имеет смещение 0x16 (22 байта вперед)
- Пример для 0x64: jmp 0x15 имеет смещение 0xAF (отрицательное, назад)

7 Сравнение глобальных и локальных переменных

7.1 Модифицированная программа с локальными переменными

Для сравнения была создана версия программы, где глобальные переменные заменены на локальные:

```
int main() {
    int Mas[10];
    unsigned long i;

    for (i = 0; i < 9; i++) {
        if (i != 6)
            Mas[i] = (17*i) & 0x0E;
        else
            Mas[i] = 0x1A * i/4;
    }
    return 0;
}
```

Листинг 2: Модифицированный код с локальными переменными

Локальные переменные размещаются в стеке и адресуются через базовый указа- тель ebp:

- **Переменная i:** dword ptr [ebp - 0x34] вместо [0x0]
- **Массив Mas[i]:** (локальный): dword ptr [ebp + 4*eax - 0x2c] вместо [4*eax] base+offset).

Основное отличие: глобальные переменные используют прямую адресацию, локальные — косвенную через базовый регистр.

8 Обращение к массиву через указатель

Для сравнения был реализован вариант программы с использованием указателя для доступа к элементам массива:

```
int Mas[10];
unsigned long i;
int *pMas = Mas;

int main() {
    for (i = 0; i < 9; i++) {
        if (i != 6)
            *(pMas + i) = (17*i) & 0x0E;
        else
            *(pMas + i) = 0x1A * i/4;
    }
    return 0;
}
```

Листинг 3: Код с использованием указателя

Перед расчетом адреса элемента pMas[i], компилятор должен загрузить базовый адрес массива (который хранится в переменной pMas) в регистр.

- **mov eax, dword ptr [0x74]** — загружает базовый адрес массива Mas из переменной-указателя pMas (которая находится по абсолютному адресу 0x74) в регистр eax (источник: прямая адресация, приёмник: регистровая).

Далее, регистры eax (базовый адрес) и ecx (индекс i) объединяются в одну команду, чтобы получить точный адрес элемента Mas[i].

- Используется многокомпонентная адресация вида dword ptr [eax+4*ecx], где eax хранит базовый адрес pMas, а 4*ecx — смещение до нужного элемента массива (источник: регистр, приёмник: память по косвенному адресу с базой и индексом).

8.1 Сравнение Подходов

Обращение по индексу:

- Компилятор использует прямую адресацию базового адреса массива (который является фиксированным абсолютным адресом, например, 0x0) и вычисляет смещение через индекс и масштаб.

Пример: mov dword ptr [4eax], edx (если базовый адрес Mas = 0) или mov dword ptr [Mas + 4eax], edx (источник: регистр edx, приёмник: память по абсолютному адресу с индексом).

Преимущество: Не нужна дополнительная инструкция для загрузки базового адреса в регистр.

Обращение через указатель:

- Сначала регистр eax загружает адрес массива из памяти: mov eax,dword ptr [0x74]. Это добавляет одну инструкцию на каждой итерации. Затем используется этот регистр eax как база для многокомпонентной адресации [eax+4ecx] (источник: регистр edx, приёмник: память по косвенному адресу с базой и индексом).

Недостаток: Дополнительная инструкция mov для получения базового адреса pMas из памяти.

При обращении по индексу (Mas[i]), компилятор может сразу использовать фиксированный базовый адрес массива (например, 0x0) в многокомпонентной адресации. При обращении через указатель ((pMas+i)), необходимо сначала загрузить значение указателя (pMas) в регистр (eax), что добавляет одну инструкцию на каждой итерации, но является необходимым, поскольку pMas — это переменная в памяти, а не константный адрес массива.

9 Ответ на вопросы:

9.1 Какие команды использовал транслятор для арифметических действий и почему?

- `imul ecx, dword ptr [0x0], 0x11` (адрес 0x27) — умножение i на 17.
- `and ecx, 0xe` (адрес 0x2e) — побитовое И с маской `0x0E`.
- `imul ecx, dword ptr [0x0], 0x1a` (адрес 0x3f) — умножение i на 26.
- `shr ecx, 0x2` (адрес 0x46) — логический сдвиг вправо на 2 для деления на 4.
- `add eax, 0x1` (адрес 0x5c) — инкремент i. Используется вместо `inc` для избежания ложных зависимостей от флагов в конвейере процессора.
- `xor eax, eax` (адрес 0x66) — возврат 0 из main.
- `add esp, 0x4` (адрес 0x68) — освобождение стека. Прямое увеличение указателя стека на 4 байта.

9.2 Где находятся операнды-источники и операнды-приемники для изучаемых команд

Расположение операндов-источников и операндов-приёмников:

Команда	**Источник**	**Приёмник**
`imul ecx, dword ptr [0x0], 0x11 (0x27)`	память [0x0] (значение i), immediate 0x11	регистр ecx
`and ecx, 0xe (0x2e)`	регистр ecx, immediate 0xe	регистр ecx
`imul ecx, dword ptr [0x0], 0x1a (0x3f)`	память [0x0] (значение i), immediate 0x1a	регистр ecx
`shr ecx, 0x2 (0x46)`	регистр ecx, immediate 0x2	регистр ecx
`mov dword ptr [4*eax], ecx (0x36, 0x4e)`	регистр ecx	память [4*eax]
`add eax, 0x1 (0x5c)`	регистр eax, immediate 0x1	регистр eax
`xor eax, eax (0x66)`	регистр eax, регистр eax	регистр eax
`cmp dword ptr [0x0], 0x9 (0x15)`	память [0x0], immediate 0x9	флаги процессора

Вывод: В Intel-синтаксисе операнды записываются как `dst, src`. Приёмник (destination) — всегда первый операнд, источник (source) — второй (и третий, если есть).

9.3 Какие регистры использовал компилятор и почему

- **EAX**(Extended Accumulator Register) — универсальный регистр, используется для:
  - загрузки адресов из GOT,
  - промежуточных вычислений (умножение, деление),
  - возвращаемого значения функции (`return 0`).
- **ECX**(Extended Count Register) — используется как:
  - индекс массива (`index` в `[base + 4*ecx]`),
  - счётчик/временное значение `i`,
  - делитель в `div ecx`.
- **EBP**, **ESP** — управление стековым фреймом (`base pointer`, `stack pointer`).

Почему операнды в регистрах, а не в памяти:

- Регистры — самая быстрая память процессора (0 тактов задержки).
- Операции с регистрами выполняются за 1 такт, обращение к памяти — десятки тактов (даже с кэшем L1).
- Компилятор размещает часто используемые переменные (индексы, временные результаты) в регистрах, а глобальные/большие структуры — в памяти

9.4 Как команды учитывают размеры и тип операндов

- **Явное указание размера:** конструкции вида `dword ptr` указывают, что операция производится над 32-битным (4-байтовым) операндом. Например:

```
mov dword ptr [ebp - 0x4], 0x0
cmp dword ptr [0x0], 0x9
```

  Это позволяет компилятору и процессору корректно интерпретировать объём памяти, задействованный в операции.

- **Тип операндов:** команды `mov`, `cmp`, `imul`, `shr`, `and`, `add` работают с регистрами общего назначения (`eax`, `ecx`, `esp`, `ebp`) и памятью. Тип операнда (регистр или память) влияет на форму машинной инструкции и её семантику.

- **Адресация с учётом размера:** при работе с массивом `Mas[i]` используется адресация `[4*eax]`, что соответствует смещению в байтах при доступе к элементам массива из 32-битных слов:

```
mov dword ptr [4*eax], ecx
```

  Здесь `eax` содержит индекс, а умножение на 4 обеспечивает переход к нужному элементу.

- **Сдвиги и маскирование:** команда `shr ecx, 0x2` реализует деление на 4, что допустимо только для целых типов. Команда `and ecx, 0xe` применяет маску к 32-битному значению.

- **Инициализация и арифметика:** команды `imul ecx, dword ptr [0x0], 0x11` и `imul ecx, dword ptr [0x0], 0x1a` используют 32-битные операнды, что соответствует типу переменной `i` в исходном C-коде.

- **Работа со стеком:** команды `push` и `pop` работают с 32-битными регистрами, что соответствует размеру слова в архитектуре IA-32.

Таким образом, каждая команда учитывает размер операндов либо явно (через `dword ptr`), либо неявно — по типу регистра. Это обеспечивает корректную работу с памятью, регистрами и арифметикой в рамках 32-битной архитектуры.

9.5 Как компилятор организовал приведение типов

Компилятор свёл все операции к 32-битным целым (`dword`):

- Переменные и константы хранятся как 32-битные слова (`dword ptr`).
- Арифметика (`imul`, `add`, `shr`, `and`) выполняется над регистрами `eax`, `ecx` — всегда 32 бита.
- Сравнения (`cmp`) используют 32-битные операнды.
- Индексация массива учитывает размер элемента: `[4*eax]` для `int`.
- Возврат значения (`xor eax, eax`) также приведён к 32-битному `int`.

Вывод: все типы исходного кода приведены к единому формату `int32`.

9.6 Способы адресации при работе с памятью

[leftmargin=1.5em]

1. **Косвенно-регистровая (register indirect):**

```
mov edx, dword ptr [eax] ; адрес 0x63
```

Адрес операнда = содержимое регистра `eax`.

2. **Базовая со смещением (base + displacement):**

```
mov eax, dword ptr [eax + 0x4b4] ; адрес 0x18
```

Адрес = `eax + 0x4b4` (например, загрузка из GOT).

3. **Базовая относительно стека (base-pointer relative):**

```
mov dword ptr [ebp - 0x10], eax ; адрес 0xe
```

Адрес = `ebp - 0x10` (локальная переменная на стеке).

4. **Многокомпонентная (base + index*scale):**

```
mov dword ptr [eax + 4*ecx], edx ; адрес 0x85
```

Адрес = `eax + 4*ecx + disp`.

9.7 Способы адресации в командах перехода

В фрагменте используются два способа задания адреса перехода:

- **Относительная адресация (relative):** Команды `jmp`, `je`, `jae` используют смещение от текущего адреса:
  - `jmp 0x55`, `jmp 0x15` — безусловный переход.
  - `je 0x3f`, `jae 0x66` — условный переход.

  Смещение кодируется как байт/слово, интерпретируемое относительно текущего IP.

- **Прямая адресация (ret):** Команда `ret` извлекает адрес возврата из стека, не содержит явного адреса.

Вывод: переходы реализованы через относительные смещения, что упрощает линейную компоновку кода. Альтернатива — `jmp eax` (косвенная адресация), но она не используется здесь.

9.8 Типовые последовательности для конструкций языка C

- **Присваивание переменной константного значения:**

```
mov dword ptr [ebp - 4], 0    ; локальная переменная = 0
```

- **Обращение к переменной через указатель:**

```
mov eax, dword ptr [esi]      ; загрузить значение по адресу в
    esi
mov dword ptr [edi], eax      ; сохранить по адресу в edi
```

- **Циклическая конструкция (for/while):**

```
mov dword ptr [i], 0          ; инициализация
cmp dword ptr [i], 9          ; проверка условия
jae end_loop                  ; выход
; тело цикла
add dword ptr [i], 1          ; инкремент
jmp loop_start
```

- **Условный оператор (if):**

```
cmp dword ptr [i], 6
je else_block                 ; если i == 6     else
; then-блок
jmp after_if
```

```
else_block:
; else-блок
after_if:
```

- **Простое/сложное логическое условие:**

```
cmp eax, 5
jl less_than_5
cmp ebx, 10
jg greater_than_10
; условие (eax < 5) AND (ebx > 10)
```

- **Обращение к простой переменной по имени:**

```
mov eax, dword ptr [i]          ; загрузить i
```

- **Обращение к простой переменной по указателю:**

```
mov eax, dword ptr [ebx]        ; ebx хранит адрес переменной
```

- **Обращение к элементу массива по индексу:**

```
mov eax, dword ptr [i]          ; индекс
mov ecx, dword ptr [Mas + 4*eax] ; Mas[i]
```

Разбор команд

Обозначения

r/m32 - register / memory 32-bit, 32-битным регистром imm32 - immediate 32-bit, Это непосредственное значение, которое уже записано внутри инструкции

r32 - 32-битный регистр

Разбор инструкций x86

1. 55 — push ebp

- Байты: 55
- Разбор:
  - 0x55 — опкод инструкции PUSH EBP (сохранение базового указателя стека).
  - Формат: 50+rw, где rw=101 для EBP.
  - Итог: 01010101.

2. 89 e5 — mov ebp, esp

- Байты: 89 e5
- Разбор:
  - 89 — опкод MOV r/m32, r32.
  - E5 — байт ModR/M:
    - 11 — режим register–register.
    - 100 — источник: ESP.
    - 101 — назначение: EBP.
- Итог: EBP = ESP.

3. 83 3d 00 00 00 00 09 — cmp dword ptr [0x0], 0x9

- Байты: 83 3d 00 00 00 00 09
- Разбор:
  - 83 — опкод CMP r/m32, imm8.
  - 3D — байт ModR/M:

    * 00 — режим с displacement.
    * 111 — код операции CMP (/7).
    * 101 — адрес через displacement.
  - 00 00 00 00 — смещение 0x0.
  - 09 — сравнение с константой 9.

4. 6b 0d 00 00 00 00 11 — **imul ecx, [0x0], 0x11**

- Байты: 6b 0d 00 00 00 00 11
- Разбор:
  - 6B — опкод IMUL r32, r/m32, imm8.
  - 0D — байт ModR/M:
    * 00 — режим displacement.
    * 001 — регистр назначения: ECX.
    * 101 — адрес через displacement.
  - 00 00 00 00 — смещение 0x0.
  - 11 — множитель 17.

5. 83 e1 0e — **and ecx, 0xe**

- Байты: 83 e1 0e
- Разбор:
  - 83 — опкод AND r/m32, imm8.
  - E1 — байт ModR/M:
    * 11 — режим register.
    * 100 — операция AND (/4).
    * 001 — регистр: ECX.
  - 0E — константа 14.

6. c1 e9 02 — **shr ecx, 0x2**

- Байты: c1 e9 02
- Разбор:
  - C1 — опкод SHR r/m32, imm8.
  - E9 — байт ModR/M:

    - `11` — режим register.
    - `101` — операция SHR (/5).
    - `001` — регистр: `ECX`.
  - `02` — сдвиг на 2 бита.

7. `83 c0 01` **— add eax, 0x1**

- Байты: `83 c0 01`
- Разбор:
  - `83` — опкод `ADD r/m32, imm8`.
  - `C0` — байт ModR/M:
    - `11` — режим register.
    - `000` — операция ADD (/0).
    - `000` — регистр: `EAX`.
  - `01` — константа `1`.

8. `31 c0` **— xor eax, eax**

- Байты: `31 c0`
- Разбор:
  - `31` — опкод `XOR r/m32, r32`.
  - `C0` — байт ModR/M:
    - `11` — режим register.
    - `000` — регистр-источник: `EAX`.
    - `000` — регистр-назначение: `EAX`.

9. `5d` **— pop ebp**

- Байты: `5d`
- Разбор:
  - `0x5D` — опкод `POP EBP`.
  - Формат: `58+rd`, где `rd=101`.

10. `c3` — **ret**

- Байты: `c3`
- Разбор:
  - `0xC3` — опкод `RET`.
  - Возврат из функции.

10 Результаты работы

В ходе выполнения лабораторной работы был изучен дизассемблированный код программы на языке C. Проанализированы способы адресации, используемые компилятором для доступа к глобальным и локальным переменным, а также к элементам массива.

Основные результаты:

- Изучены команды процессора x86 в 32-битном режиме и способы их использования для реализации конструкций языка высокого уровня.
- Проанализированы различные способы адресации: непосредственная, прямая, регистровая, косвенная и многокомпонентная.
- Составлена карта памяти для размещения глобальных переменных.
- Выявлены различия в способах адресации при обращении к глобальным и локальным переменным.
- Сравнены подходы к доступу к элементам массива через индексацию и через указатели.
- Изучены команды условных и безусловных переходов, используемые для реализации циклов и ветвлений.

Полученные знания позволяют лучше понимать, как компилятор транслирует программы на языках высокого уровня в машинный код, и как процессор выполняет эти программы на низком уровне.