Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта
Направление: 02.03.01 «Математика и компьютерные науки»

Отчёт о выполнении лабораторной работы №6

«Команды и способы адресации для x86 в 32-битном режиме»

Вариант 2-2

Выполнил студент
группы 5130201/40002 _____ Семенов И. А.

Проверила
преподаватель _____ Веробова Н. М.

«____»_____ 2025г.

Санкт-Петербург
2025

1 Цель работы

Познакомиться с системой процессорных команд x86 в 32-битном режиме и способами использования этих команд. Изучить способы адресации, кодирование команд и организацию доступа к переменным в памяти.

2 Методика выполнения работы

1. Взять фрагмент программы на языке Си согласно варианту задания
2. Оттранслировать программу с в 32-битном режиме без оптимизаций
3. Получить дизассемблированный код функции `main`
4. Проанализировать соответствие инструкций Си и ассемблерного кода
5. Заменить обращение к элементам массива через индекс на обращение через указатель
6. Сравнить изменения в способах адресации

3 Исходный код программы

3.1 Вариант с индексацией массива

Листинг 1: Исходный код с индексацией массива

```
1  int cMas[10];
2  unsigned long i;
3
4  void main() {
5      for (i = 0; i < 9; i++) {
6          if (i != 6)
7              cMas[i] = (17 * i) & 0x0E;
8          else
9              cMas[i] = 0x1A * i / 4;
10     }
11 }
```

Переменные `cMas` и `i` объявлены глобально.

4 Дизассемблированный код функции main

Листинг 2: Дизассемблированный код функции main

```
0000118d <main>:
    118d:   55                      push   ebp
    118e:   89 e5                   mov    ebp,esp
    1190:   e8 76 00 00 00          call   120b <__x86.get_pc_thunk.ax>
    1195:   05 47 2e 00 00          add    eax,0x2e47
    119a:   c7 80 8c 00 00 00 00    mov    DWORD PTR [eax+0x8c],0x0
    11a1:   00 00 00
    11a4:   eb 56                   jmp    11fc <main+0x6f>
    11a6:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    11ac:   83 fa 06                cmp    edx,0x6
    11af:   74 21                   je     11d2 <main+0x45>
    11b1:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    11b7:   89 d1                   mov    ecx,edx
    11b9:   c1 e1 04                shl    ecx,0x4
    11bc:   01 ca                   add    edx,ecx
    11be:   89 d1                   mov    ecx,edx
    11c0:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    11c6:   83 e1 0e                and    ecx,0xe
    11c9:   89 8c 90 64 00 00 00    mov    DWORD PTR [eax+edx*4+0x64],ecx
    11d0:   eb 1b                   jmp    11ed <main+0x60>
    11d2:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    11d8:   6b d2 1a                imul   edx,edx,0x1a
    11db:   89 d1                   mov    ecx,edx
    11dd:   c1 e9 02                shr    ecx,0x2
    11e0:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    11e6:   89 8c 90 64 00 00 00    mov    DWORD PTR [eax+edx*4+0x64],ecx
    11ed:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    11f3:   83 c2 01                add    edx,0x1
    11f6:   89 90 8c 00 00 00       mov    DWORD PTR [eax+0x8c],edx
    11fc:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    1202:   83 fa 08                cmp    edx,0x8
    1205:   76 9f                   jbe    11a6 <main+0x19>
    1207:   90                      nop
    1208:   90                      nop
    1209:   5d                      pop    ebp
    120a:   c3                      ret
```

5 Разбор дизассемблера по байтам

Объяснение каждой инструкции для функции `main`:

- **0x0000118d** <+0>: `55 push ebp`
  Байт-код: 55
  Сохраняем указатель базы стека предыдущей функции.

- **0x0000118e** <+1>: `89 e5 mov ebp,esp`
  Байт-код: 89 e5

Устанавливает `ebp` на текущий верх стека (`esp`), чтобы использовать ebp как базу для локальных переменных.

- **0x00001190** <+3>: `e8 76 00 00 00 call 0x120b <__x86.get_pc_thunk.ax>`
  Байт-код: e8 76 00 00 00
  Сохраняет адрес следующей инструкции в регистре `eax` (служебная вставка).

- **0x00001195** <+8>: `05 47 2e 00 00 add eax,0x2e47`
  Байт-код: 05 47 2e 00 00
  Добавляет смещение к `eax` для вычисления адреса глобальных данных.

- **0x0000119a** <+13>: `c7 80 8c 00 00 00 00 00 00 00 mov DWORD PTR [eax+0x8c],0x0`
  Байт-код: c7 80 8c 00 00 00 00 00 00 00
  Инициализация переменной `i = 0` по адресу `eax`+0x8c.

- **0x000011a4** <+23>: `eb 56 jmp 0x11fc <main+111>`
  Байт-код: eb 56
  Переход к проверке условия цикла.

- **0x000011a6** <+25>: `8b 90 8c 00 00 00 mov edx,DWORD PTR [eax+0x8c]`
  Байт-код: 8b 90 8c 00 00 00
  Загрузка значения переменной `i` в регистр `edx`.

- **0x000011ac** <+31>: `83 fa 06 cmp edx,0x6`
  Байт-код: 83 fa 06
  Сравнение значения `i` с константой 6.

- **0x000011af** <+34>: `74 21 je 0x11d2 <main+69>`
  Байт-код: 74 21
  Условный переход на else-ветку, если `i == 6`.

- **0x000011b1** <+36>: `8b 90 8c 00 00 00 mov edx,DWORD PTR [eax+0x8c]`
  Байт-код: 8b 90 8c 00 00 00
  Повторная загрузка `i` в `edx` для вычисления `17*i`.

- **0x000011b7** <+42>: `89 d1 mov ecx,edx`
  Байт-код: 89 d1
  Копирование значения `i` из `edx` в `ecx`.

- **0x000011b9** <+44>: `c1 e1 04 shl ecx,0x4`
  Байт-код: c1 e1 04
  Сдвиг `ecx` влево на 4 бита: `ecx = i * 16`.

- **0x000011bc** <+47>: `01 ca add edx,ecx`
  Байт-код: 01 ca
  Сложение: `edx = i + i*16 = i*17`. Оптимизация умножения на 17.

- **0x000011be** <+49>: `89 d1 mov ecx,edx`
  Байт-код: 89 d1
  Копирование результата `i*17` в `ecx`.

- **0x000011c0** <+51>: `8b 90 8c 00 00 00 mov edx,DWORD PTR [eax+0x8c]`
  Байт-код: 8b 90 8c 00 00 00
  Загрузка `i` в `edx` для использования как индекса массива.

- **0x000011c6** <+57>: `83 e1 0e and ecx,0xe`
  Байт-код: 83 e1 0e
  Побитовое И: `ecx = (i*17) & 0x0E`.

- **0x000011c9** <+60>: `89 8c 90 64 00 00 00 mov DWORD PTR [eax+edx*4+0x64`
  Байт-код: 89 8c 90 64 00 00 00
  Запись результата в `cMas[i]`. Многокомпонентная адресация: база `eax` + индекс `edx`*4 + смещение 0x64.

- **0x000011d0** <+67>: `eb 1b jmp 0x11ed <main+96>`
  Байт-код: eb 1b
  Безусловный переход к инкременту счётчика, минуя else-ветку.

- **0x000011d2** <+69>: `8b 90 8c 00 00 00 mov edx,DWORD PTR [eax+0x8c]`
  Байт-код: 8b 90 8c 00 00 00
  Начало else-ветки: загрузка `i` в `edx`.

- **0x000011d8** <+75>: `6b d2 1a imul edx,edx,0x1a`
  Байт-код: 6b d2 1a
  Умножение: `edx = i * 26` (0x1A = 26).

- **0x000011db** <+78>: `89 d1 mov ecx,edx`
  Байт-код: 89 d1
  Копирование результата `i*26` в `ecx`.

- **0x000011dd** <+80>: `c1 e9 02 shr ecx,0x2`
  Байт-код: c1 e9 02
  Сдвиг вправо на 2 бита: `ecx = (i*26) / 4`. Оптимизация деления на 4.

- **0x000011e0** <+83>: `8b 90 8c 00 00 00 mov edx,DWORD PTR [eax+0x8c]`
  Байт-код: 8b 90 8c 00 00 00
  Загрузка `i` в `edx` для индексации массива.

- **0x000011e6** <+89>: `89 8c 90 64 00 00 00 mov DWORD PTR [eax+edx*4+0x64`
  Байт-код: 89 8c 90 64 00 00 00
  Запись результата в `cMas[i]`.

- **0x000011ed** <+96>: `8b 90 8c 00 00 00 mov edx,DWORD PTR [eax+0x8c]`
  Байт-код: 8b 90 8c 00 00 00
  Загрузка текущего значения `i` для инкремента.

- **0x000011f3** <+102>: `83 c2 01 add edx,0x1`
  Байт-код: 83 c2 01
  Инкремент: `edx = i + 1`.

- **0x000011f6** <+105>: `89 90 8c 00 00 00 mov DWORD PTR [eax+0x8c],edx`
  Байт-код: 89 90 8c 00 00 00
  Сохранение нового значения: `i = i + 1`.

- **0x000011fc** <+111>: `8b 90 8c 00 00 00 mov edx,DWORD PTR [eax+0x8c]`
  Байт-код: 8b 90 8c 00 00 00
  Загрузка `i` для проверки условия цикла.

- **0x00001202** <+117>: `83 fa 08 cmp edx,0x8`
  Байт-код: 83 fa 08
  Сравнение `i` с 8.

- **0x00001205** <+120>: `76 9f jbe 0x11a6 <main+25>`
  Байт-код: 76 9f
  Условный переход: если `i <= 8` (беззнаковое сравнение), продолжить цикл. Условие `i < 9` реализовано как `i <= 8`.

- **0x00001207** <+122>: `90 nop`
  Байт-код: 90
  Пустая операция для выравнивания.

- **0x00001208** <+123>: `90 nop`
  Байт-код: 90
  Пустая операция для выравнивания.

- **0x00001209** <+124>: `5d pop ebp`
  Байт-код: 5d
  Восстановление указателя базы стека предыдущей функции.

- **0x0000120a** <+125>: `c3 ret`
  Байт-код: c3
  Возврат из функции.

6 Разбор двоичного кода команд

6.1 Команда mov DWORD PTR [eax+0x8c],0x0 (адрес 119a)

Машинный код: `c7 80 8c 00 00 00 00 00 00 00`

- `c7` — опкод MOV для записи непосредственного значения в память (r/m32, imm32)
- `80` — байт ModR/M: Mod=10 (смещение 32 бита), Reg=000, R/M=000 (EAX)
- `8c 00 00 00` — смещение 0x0000008c (little-endian)
- `00 00 00 00` — непосредственное значение 0

6.2 Команда mov DWORD PTR [eax+edx*4+0x64],ecx (адрес 11c9)

Машинный код: `89 8c 90 64 00 00 00`

- `89` — опкод MOV (r/m32, r32)
- `8c` — байт ModR/M: Mod=10 (смещение 32 бита), Reg=001 (ECX), R/M=100 (SIB follows)
- `90` — байт SIB: Scale=10 (*4), Index=010 (EDX), Base=000 (EAX)
- `64 00 00 00` — смещение 0x00000064 (little-endian)

6.3 Команда jbe 11a6 (адрес 1205)

Машинный код: `76 9f`

- `76` — опкод JBE (короткий переход)
- `9f` — относительное смещение (-97 в дополнительном коде)

Вычисление адреса перехода: 0x1207 + 0xFFFFFF9F = 0x1207 - 0x61 = 0x11a6

- Команда push ebp (адрес 118d)
  - Машинный код: 55

  - 55 — опкод PUSH для регистра EBP. Опкоды 50-57 кодируют push для регистров EAX-EDI соответственно (50+5=55 для EBP)

- Команда call 120b (адрес 1190)
  - Машинный код: e8 76 00 00 00
  - e8 — опкод CALL с относительным 32-битным смещением
  - 76 00 00 00 — смещение 0x00000076

- Команда add eax,0x2e47 (адрес 1195)
  - Машинный код: 05 47 2e 00 00
  - 05 — опкод ADD для пары (EAX)
  - 47 2e 00 00 — значение 0x00002e47

- Команда shl ecx,0x4 (адрес 11b9)
  - Машинный код: c1 e1 04
  - c1 — опкод для групповых сдвигов
  - e1 (11100001) — 11 (регистр-операнд), 100 (код операции SHL), 001 (ECX)
  - 04 — величина сдвига (4 бита)

- Команда and ecx,0xe (адрес 11c6)
  - Машинный код: 83 e1 0e
  - 83 — опкод для групповых операций
  - e1 — байт ModR/M 11 (регистр-операнд), 100 (код операции), 001 (ECX)
  - 0e — 0x0E маска

- Команда imul edx,edx,0x1a (адрес 11d8)
  - Машинный код: 6b d2 1a
  - 6b — опкод (трёхоперандное умножение с 8-битным множителем)
  - d2 (11010010) — 11 (регистр-операнд), 010 (приёмник), 010 (источник)
  - 1a — множитель 0x1A (26)

- Команда shr ecx,0x2 (адрес 11dd)
  - Машинный код: c1 e9 02

- c1 — опкод для групповых сдвигов r/m32 на imm8
- e9 (11101001) 11 (регистр-операнд), 101 (код операции SHR), 001 (ECX)
- 02 — величина сдвига (2 бита, эквивалент деления на 4)

- Команда ret (адрес 120a)
  - Машинный код: c3
  - c3 — опкод RET. Извлекает адрес возврата из стека.

7 Анализ переменных программы

Переменная	Тип	Размер	Объявление	Расположение
`i`	unsigned long	4 байта	глобально	`[eax+0x8c]`
`cMas[10]`	int[10]	40 байт	глобально	`[eax+0x64]`

8 Карта памяти переменных

Адрес (смещение)	Переменная	Размер	Тип
`[eax+0x64]`	`cMas[0]`	4 байта	int
`[eax+0x68]`	`cMas[1]`	4 байта	int
`[eax+0x6c]`	`cMas[2]`	4 байта	int
`[eax+0x70]`	`cMas[3]`	4 байта	int
`[eax+0x74]`	`cMas[4]`	4 байта	int
`[eax+0x78]`	`cMas[5]`	4 байта	int
`[eax+0x7c]`	`cMas[6]`	4 байта	int
`[eax+0x80]`	`cMas[7]`	4 байта	int
`[eax+0x84]`	`cMas[8]`	4 байта	int
`[eax+0x88]`	`cMas[9]`	4 байта	int
`[eax+0x8c]`	`i`	4 байта	unsigned long

8.1 Используемые способы адресации

1. **Непосредственная адресация** — операнд задан в команде:
   - `add edx,0x1` — константа 1
   - `cmp edx,0x6` — константа 6
   - `and ecx,0xe` — маска 0x0E
2. **Регистровая адресация** — операнд в регистре:

- `mov ecx,edx` — оба операнда в регистрах
- `add edx,ecx` — сложение регистров

3. **Базовая со смещением** — для доступа к переменной `i`:
   - `mov edx,[eax+0x8c]` — база EAX, смещение 0x8c
   - Формула: EA = EAX + 0x8c
4. **Многокомпонентная** — для доступа к `cMas[i]`:
   - `mov [eax+edx*4+0x64],ecx`
   - Формула: EA = EAX + EDX*4 + 0x64
   - EAX — база (адрес глобальных данных)
   - EDX — индекс (значение `i`)
   - 4 — масштаб (размер элемента int)
   - 0x64 — смещение массива

8.2 Адресация в командах перехода

- **Относительная короткая** (`jmp`, `je`, `jbe`): смещение 8 бит от адреса следующей команды
- Пример: `jbe 11a6` (код `76 9f`) — переход назад на 97 байт

9 Версия с указателем

9.1 Исходный код с указателем

Листинг 3: Исходный код с указателем

```
1  int cMas[10];
2  unsigned long i;
3
4  void main() {
5      int *p = cMas;
6      for (i = 0; i < 9; i++) {
7          if (i != 6)
8              *(p + i) = (17 * i) & 0x0E;
9          else
10             *(p + i) = 0x1A * i / 4;
11     }
12 }
```

9.2 Дизассемблированный код версии с указателем

Листинг 4: Дизассемблированный код с указателем

```
0000118d <main>:
    118d:   55                      push   ebp
    118e:   89 e5                   mov    ebp,esp
    1190:   53                      push   ebx
    1191:   83 ec 10                sub    esp,0x10
    1194:   e8 90 00 00 00          call   1229 <__x86.get_pc_thunk.ax>
    1199:   05 43 2e 00 00          add    eax,0x2e43
    119e:   8d 90 64 00 00 00       lea    edx,[eax+0x64]
    11a4:   89 55 f8                mov    DWORD PTR [ebp-0x8],edx
    11a7:   c7 80 8c 00 00 00 00    mov    DWORD PTR [eax+0x8c],0x0
    11ae:   00 00 00
    11b1:   eb 64                   jmp    1217 <main+0x8a>
    11b3:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    11b9:   83 fa 06                cmp    edx,0x6
    11bc:   74 28                   je     11e6 <main+0x59>
    11be:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    11c4:   89 d1                   mov    ecx,edx
    11c6:   c1 e1 04                shl    ecx,0x4
    11c9:   01 ca                   add    edx,ecx
    11cb:   89 d1                   mov    ecx,edx
    11cd:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    11d3:   8d 1c 95 00 00 00 00    lea    ebx,[edx*4+0x0]
    11da:   8b 55 f8                mov    edx,DWORD PTR [ebp-0x8]
    11dd:   01 da                   add    edx,ebx
    11df:   83 e1 0e                and    ecx,0xe
    11e2:   89 0a                   mov    DWORD PTR [edx],ecx
    11e4:   eb 22                   jmp    1208 <main+0x7b>
    11e6:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    11ec:   6b d2 1a                imul   edx,edx,0x1a
    11ef:   89 d1                   mov    ecx,edx
    11f1:   c1 e9 02                shr    ecx,0x2
    11f4:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    11fa:   8d 1c 95 00 00 00 00    lea    ebx,[edx*4+0x0]
    1201:   8b 55 f8                mov    edx,DWORD PTR [ebp-0x8]
    1204:   01 da                   add    edx,ebx
    1206:   89 0a                   mov    DWORD PTR [edx],ecx
    1208:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    120e:   83 c2 01                add    edx,0x1
    1211:   89 90 8c 00 00 00       mov    DWORD PTR [eax+0x8c],edx
    1217:   8b 90 8c 00 00 00       mov    edx,DWORD PTR [eax+0x8c]
    121d:   83 fa 08                cmp    edx,0x8
    1220:   76 91                   jbe    11b3 <main+0x26>
    1222:   90                      nop
    1223:   90                      nop
    1224:   8b 5d fc                mov    ebx,DWORD PTR [ebp-0x4]
    1227:   c9                      leave
    1228:   c3                      ret
```

9.3 Анализ изменений при переходе на указатель

9.3.1 Новые инструкции в прологе

Листинг 5: Инициализация указателя p

```
1190:   53                      push   ebx
1191:   83 ec 10                sub    esp,0x10
...
119e:   8d 90 64 00 00 00       lea    edx,[eax+0x64]
11a4:   89 55 f8                mov    DWORD PTR [ebp-0x8],edx
```

- `push ebx` — сохранение регистра EBX (будет использоваться для вычислений)
- `sub esp,0x10` — выделение 16 байт в стеке для локальных переменных
- `lea edx,[eax+0x64]` — загрузка адреса массива `cMas` в EDX
- `mov [ebp-0x8],edx` — сохранение указателя `p` в локальную переменную на стеке

9.3.2 Изменения в обращении к элементам массива

Версия с индексом (одна команда с многокомпонентной адресацией):

Листинг 6: Обращение через индекс

```
mov    DWORD PTR [eax+edx*4+0x64],ecx    ; cMas[i] = ecx
```

Версия с указателем (несколько команд):

Листинг 7: Обращение через указатель

```
11d3:   lea    ebx,[edx*4+0x0]        ; ebx = i * 4 (смещение в байтах)
11da:   mov    edx,DWORD PTR [ebp-0x8]  ; edx = p (загрузка указателя)
11dd:   add    edx,ebx                ; edx = p + i*4
...
11e2:   mov    DWORD PTR [edx],ecx    ; *(p + i) = ecx
```

9.3.3 Изменения в эпилоге

Листинг 8: Эпилог версии с указателем

```
1224:   8b 5d fc                mov    ebx,DWORD PTR [ebp-0x4]
1227:   c9                      leave
1228:   c3                      ret
```

Добавлено восстановление регистра EBX из стека перед возвратом.

9.4 Сравнение способов адресации

1. При использовании указателя компилятор создаёт **локальную переменную** в стеке для хранения адреса массива.

2. Обращение через указатель требует **больше инструкций**: вычисление смещения (`lea`), загрузка указателя из стека (`mov`), сложение (`add`), и только потом запись (`mov`).

3. Версия с индексом использует **многокомпонентную адресацию** x86 (база + индекс*масштаб + смещение) в одной команде, что эффективнее.

4. Версия с указателем использует **косвенно-регистровую адресацию** `[edx]` после предварительного вычисления адреса.

5. Код с указателем на **30 байт больше** (156 против 126 байт) и требует дополнительный регистр (EBX).

10 Выводы

В ходе выполнения лабораторной работы были изучены:

1. Система команд x86 в 32-битном режиме и их кодирование

2. Способы адресации: непосредственная, регистровая, базовая со смещением, многокомпонентная

3. Оптимизации компилятора: замена умножения на сдвиги и сложения, замена деления на сдвиги

4. Организация доступа к глобальным переменным

5. Различия в генерируемом коде при использовании индексации массива и указателей

Компилятор эффективно использует возможности архитектуры, применяя многокомпонентную адресацию для доступа к элементам массива и оптимизируя арифметические операции.