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

Основы архитектуры ЦВМ

**Отчет о выполнении лабораторной работы №6**

Студент,

группы 5130201/30002 \_\_\_\_\_\_\_\_\_\_\_\_\_ Филиппов Г. М.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_ Вербова Н. М.

Санкт-Петербург - 2024 г.

# Листинг 1. Исходный вариант 2.8

Исходный код:

int cMas[10];

unsigned int i;

void main()

{

for (i = 0; i < 9; i++) // Заменить цикл на do......while

{

if (i >= 4)

cMas[i] = 13 \* i + 7;

else

cMas[i] = (i \* 8) ^ 0xC;

}

}

Дизассемблированный код:

1: int cMas[10];

2: unsigned int i;

3:

4: void main() { // Вызов функции main

00001119 <+0>: push %rbp // Сохранение текущего значения frame pointer (rbp) для

// возвращения к предыдущему контексту

0000111a <+1>: mov %rsp,%rbp // Сохранение frame pointer на текущую вершину стека, rbp

// используется для обращения к локальным переменным

5: for (i = 0; i < 9; i++) {

0000111d <+4>: movl $0x0,0x2f41(%rip) # 0x4068 <i> // Инициализация переменной (по адресу) i = 0

// расположение i определяется как 0x4068

00001127 <+14>: jmp 0x119d <main+132> // Переход к метке проверки условия цикла <main+132>

00001129 <+16>: mov 0x2f39(%rip),%eax # 0x4068 <i> // Загрузка i в регистр eax для

// последующей работы с переменной

6: if (i >= 4) // Проверка условия

0000112f <+22>: cmp $0x3,%eax // Сравнение значения i с 3 для проверки условия

00001132 <+25>: jbe 0x1166 <main+77> // Если i ≤ 3, переход к метке <main+77>, иначе далее

// jump if below or equal

7: cMas[i] = 13 \* i + 7; // Если i ≥ 4, то:

00001134 <+27>: mov 0x2f2e(%rip),%edx # 0x4068 <i> // Загрузка i в регистр edx для

// последующей работы с переменной

0000113a <+33>: mov %edx,%eax // Копирование i в регистр eax, чтобы вычислить 13 \* 0000113c <+35>: add %eax,%eax // Регистр eax складываем с самим собой, тем самым i

// мы умножаем на 2 -> 2\*i

0000113e <+37>: add %edx,%eax // Затем в регистр eax добавляем i из регистра edx,

// тем самым в eax будет лежать уже 3\*i

00001140 <+39>: shl $0x2,%eax // Сдвигаем значение в регистре eax на 2 влево, тем

// самым получая в eax из 3\*i -> 12\*i

00001143 <+42>: add %edx,%eax // Еще раз добавляем i к регистру eax, получая наконец,

// 12\*i + i = 13\*i

00001145 <+44>: add $0x7,%eax // Прибавляем 7 к регистру eax, тем самым имея

// окончательное значение 13\*i + 7

00001148 <+47>: mov 0x2f1a(%rip),%edx # 0x4068 <i> // Загружаем i в регистр edx

0000114e <+53>: mov %eax,%ecx // Сохраняем значение 13\*i + 7 в регистре ecx

00001150 <+55>: mov %edx,%eax // Загружаем в регистр eax значение i из edx

00001152 <+57>: lea 0x0(,%rax,4),%rdx // Теперь загружаем в регистр rdx адрес ячейки памяти,

// куда ссылается cMas[i], для этого относительно постоян.

// смещения 0x0, мы смешаемся на значение rax = eax = i // умноженного на 4 (размер int)

0000115a <+65>: lea 0x2edf(%rip),%rax # 0x4040 <cMas> // Записываем адрес начала cMas в rax

00001161 <+72>: mov %ecx,(%rdx,%rax,1) // Перемещаем сохраненное значение 13\*i + 7, по адресу

// начало массива сMas из регистра rax + смещение из

// регистра rdx на i int-ов вправо.

00001164 <+75>: jmp 0x118e <main+117> // Переход к метке <main+117>, где происходит увеличение i

9: cMas[i] = (i \* 8) ^ 0xC; // Если i < 3, то из метки <main+25> мы попадаем сюда

00001166 <+77>: mov 0x2efc(%rip),%eax # 0x4068 <i> // Загружаем i в регистр eax

0000116c <+83>: shl $0x3,%eax // Сдвигаем регистр eax на 3 влево, тем самым умножая

// значение i на 8 -> 8\*i

0000116f <+86>: xor $0xc,%eax // Выполняем побитовую операцию xor с значением в

// регистре eax -> 8\*i xor 0xC

00001172 <+89>: mov 0x2ef0(%rip),%edx # 0x4068 <i> // Загружаем i в регистр edx

00001178 <+95>: mov %eax,%ecx // Копируем значение 8\*i ^ 0xC в регистр ecx

0000117a <+97>: mov %edx,%eax // Копируем значение i из edx в eax

0000117c <+99>: lea 0x0(,%rax,4),%rdx // Аналогично метке <main+57>

00001184 <+107>: lea 0x2eb5(%rip),%rax # 0x4040 <cMas> // Аналогично метке <main+65>

0000118b <+114>: mov %ecx,(%rdx,%rax,1) // Аналогично метке <main+72>

0000118e <+117>: mov 0x2ed4(%rip),%eax # 0x4068 <i> // Загружаем в регистр eax значение i

00001194 <+123>: add $0x1,%eax // Инкрементируем i из цикла for

00001197 <+126>: mov %eax,0x2ecb(%rip) # 0x4068 <i> // Загружаем значение i+1 по адресу i

0000119d <+132>: mov 0x2ec5(%rip),%eax # 0x4068 <i> // Загружаем обновленное значение i в

// регистр eax

000011a3 <+138>: cmp $0x8,%eax // Сравниваем значение i в регистре eax c 8

000011a6 <+141>: jbe 0x1129 <main+16> // Если значение i в регистре eax ≤ 8, то случится переход

// к метке <main+16> (начало тела цикла)

000011a8 <+143>: nop // Просто заглушка

000011a9 <+144>: nop // Просто заглушка

000011aa <+145>: pop %rbp // Удаляем адрес начала функции из стека, больше по нему

// нельзя обратиться к локальным переменным

000011ab <+146>: ret // Завершение функции и возвращение управления функции

// выше, в данном случае - завершение программы

# Листинг 2. Исходный вариант 2.8

Заменим оператор цикла на тот, что предлагается по заданию и исправим инициализатор итератора, также заменим постфиксный инкремент на префиксный.

Кроме того, сделаем массив локальным статическим, а итератор локальным. Заменим обращение к элементам массива по указателю, как требуется в одном из пунктов задания.

Исходный код:

void main() {

static int cMas[10];

unsigned int i = 0;

do {

if (i >= 4)

\*(cMas + i) = 13 \* i + 7;

else

\*(cMas + i) = (i \* 8) ^ 0xC;

++i;

} while (i < 9);

}

Дизассемблированный код:

1: int main() { // Вызов функции main

00001119 <+0>: push %rbp // Сохранение текущего значения frame pointer (rbp) для

// возвращения к предыдущему контексту

0000111a <+1>: mov %rsp,%rbp // Сохранение frame pointer на текущую вершину стека, rbp

// используется для обращения к локальным переменным

0000111d <+4>: movl $0x0,-0x4(%rbp) // Инициализация локальной переменной i нулем

00001124 <+11>: cmpl $0x3,-0x4(%rbp) // Сравнение переменной i и 3

00001128 <+15>: jbe 0x1156 <main+61> // Если 3 below or equal чем i, то идем к метке <main+61>

0000112a <+17>: mov -0x4(%rbp),%edx // Иначе сохраняем i в регистр edx

0000112d <+20>: mov %edx,%eax // Копируем из edx значение в регистр eax

0000112f <+22>: add %eax,%eax // Умножаем i в eax на 2 -> 2i

00001131 <+24>: add %edx,%eax // Добавляем к регистру eax еще одну i -> 3i

00001133 <+26>: shl $0x2,%eax // Сдвигаем значение регистра на 2 влево -> 12i

00001136 <+29>: add %edx,%eax // Добавляем еще одну i -> 13i

00001138 <+31>: lea 0x7(%rax),%ecx // Прибавляем к rax 7 и записываем в регистр ecx -> 13i+7

0000113b <+34>: mov -0x4(%rbp),%eax // Копируем лок. переменную i в регистр eax

0000113e <+37>: lea 0x0(,%rax,4),%rdx // Считаем смещение адреса на i позиция для int-ов

00001146 <+45>: lea 0x2ef3(%rip),%rax # 0x4040 <cMas.0> // Копируем адрес статического массива

// cMas в регистр rax

0000114d <+52>: add %rdx,%rax // К адресу rax добавляем наше смещение rdx и получаем

// итоговый адрес элемента cMas[i]

00001150 <+55>: mov %ecx,%edx // Копируем значение 13i+7 в регистр edx

00001152 <+57>: mov %edx,(%rax) // Копируем значение 13i+7 по адресу из регистра rax

00001154 <+59>: jmp 0x117a <main+97> // Прыгаем в метку <main+97> (после if)

00001156 <+61>: mov -0x4(%rbp),%eax // Копируем значение i в регистр eax

00001159 <+64>: shl $0x3,%eax // Сдвигаем регистр eax на 3 позиции влево -> 8i

0000115c <+67>: xor $0xc,%eax // Делаем xor с 0xc и заносим результат в eax

0000115f <+70>: mov %eax,%ecx // Копируем значение eax в ecx

00001161 <+72>: mov -0x4(%rbp),%eax // Копируем значение i в регистр eax

00001164 <+75>: lea 0x0(,%rax,4),%rdx // Аналогично метке <main+37>

0000116c <+83>: lea 0x2ecd(%rip),%rax # 0x4040 <cMas.0> // Аналогично метке <main+45>

00001173 <+90>: add %rdx,%rax // Аналогично метке <main+52>

00001176 <+93>: mov %ecx,%edx // Аналогично метке <main+55>

00001178 <+95>: mov %edx,(%rax) // Аналогично метке <main+57>

0000117a <+97>: addl $0x1,-0x4(%rbp) // Добавляем к нашей переменной i значение 1

0000117e <+101>: cmpl $0x8,-0x4(%rbp) // Сравнение 8 и переменной i

00001182 <+105>: jbe 0x1124 <main+11> // Если 8 below or equal переменной i, то прыгаем на метку

// <main+11>, что соответствует началу тела цикла

00001184 <+107>: nop // Заглушка

00001185 <+108>: nop // Заглушка

00001186 <+109>: pop %rbp // Удаляем адрес начала функции из стека, больше по нему

// нельзя обратиться к локальным переменным

00001187 <+110>: ret // Завершение функции и возвращение управления функции

// выше, в данном случае - завершение программы

# Порядок выполнения по шагам

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.

**Для листинга 1**. И массив (Mas[]) и итератор (i) объявлены глобально.

int cMas[10];

unsigned short i;

| **Переменная** | **Адрес** |
| --- | --- |
| cMas[0] | 0x4040 |
| cMas[1] | 0x4044 |
| cMas[2] | 0x4048 |
| cMas[3] | 0x404c |
| cMas[4] | 0x4050 |
| cMas[5] | 0x4054 |
| cMas[6] | 0x4058 |
| cMas[7] | 0x405c |
| cMas[8] | 0x4060 |
| cMas[9] | 0x4064 |
| i | 0x4068 |

**Для листинга 2**. Массив (Mas[]) объявлен локально-static, в то время как итератор (i) объявлен локально.

static int cMas[10];

unsigned int i = 0;

| **Переменная** | **Адрес** |
| --- | --- |
| Mas[0] | 0x4040 |
| ... | ... |
| Mas[9] | 0x4064 |

| **Стек** | | | |
| --- | --- | --- | --- |
| **Указатель стека** | **Переменная** | **Значение переменной при инициализации** | **Адрес** |
| rsp → |  |  | 0x0018FEEC |
|  | ... | ... | ... |
| (rbp – 4) → | i (мл. байт) | 0x00 | 0x0018FF30 |
|  | i | 0x00 | 0x0018FF31 |
|  | i | 0x00 | 0x0018FF32 |
|  | i (ст. байт) | 0x00 | 0x0018FF33 |
| rbp → | old rbp | 0x0018FF80 | 0x0018FF34 |

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

**Для листинга 1**. Все команды, которые обращаются к переменной i, используют абсолютную адресацию и с помощью регистра %rip + адрес смещения, обращаются к переменной в памяти, которая лежит по адресу 0x4068. Аналогично для массива cMas[]

**Для листинга 2**.

Для обращения к итератору (i), расположенному в сетке, используется косвенно-регистровая адресация со смещением, и в поле одного из операндов используется запись [rbp-4].

Для обращения к массиву cMas, используется как в 1-ом листинге абсолютный адрес, который через регистр %rip + адрес смещения позволяет к нему обращаться

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

Например, 00402743 8B 45 F8 mov eax,dword ptr [ebp-8] ; заносим в регистр eax значение переменной i, расположенной в стеке

| | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | --- | --- | --- | --- | --- | --- | --- | --- | | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |   Opcode | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | | MOD | | REG | | | R/M | | | | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |   mod-reg-r/m | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | --- | --- | --- | --- | --- | --- | --- | --- | | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |   Displacement |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

1. **mov %ecx,(%rdx,%rax,1)** - Команда перемещает содержимое регистра ecx по адресу rdx + rax \* 1, т.е. по адресу rdx + rax
2. **movl $0x0,0x2f41(%rip)** - Команда перемещает значение нуля по адресу из регистра rip + 0x2f41, чтобы получить целевой адрес глобальной или статической переменной, относительно следующей ассемблерной команды
3. **mov 0x2efc(%rip), %eax** - Команда точно так же как и выше, находит адрес путем прибавления некой константы к адресу из регистра, только далее, полученное значение сразу загружается в регистр eax.
4. **lea 0x0(,%rax,4),%rdx** Аналогично первому пункту, только уже используется множитель 4, из чего получается 0 + rax \* 4, где rax содержит кол-во смещений, а 4 - величина шага, в данном случае для int, в данном случае адрес относительный, так как нет начала массива, поэтому в rdx мы просто загружаем адрес смещения для i-го элемента.

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

Все команды перехода, используемые в программе, используют относительную адресацию (относительно текущего значения счетчика команд IP)

Например, 0040277A 75 BB jne main+17h (0402737h)

| | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |   Opcode | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | --- | --- | --- | --- | --- | --- | --- | --- | | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |   Displacement |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

* **jbe 0x1156 <main+61>** - конструкция jbe дословно означает - «jump if below or equal». Данная конструкция используется только для беззнаковых чисел. И не смотря на то, что в исходном коде сигнатуры условий различны (i >= 4 и i < 9) тем не менее в обоих конструкциях используется jbe, как наиболее оптимизированный по мнению компилятора. После конструкции идет адрес перехода - 0x1156 и условная метка <main+61>, показывающая, что команда удалена от начала main на 61 байт.