**Министерство образования и науки Российской Федерации**

**федеральное государственное автономное образовательное учреждение высшего образования**

**“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,**

**МЕХАНИКИ И ОПТИКИ”**

Факультет Программной инженерии и компьютерной техники

Образовательная программа Вычислительные системы и сети

Направление подготовки (специальность) 09.03.01 – Информатика и вычислительная техника

О Т Ч Е Т

Об учебной практике по получению первичных профессиональных умений и навыков

Тема задания: Исследование и моделирование архитектуры набора команд х86

Обучающийся Давыдов И. Д., гр. P3200

Руководитель практики от университета: Логинов Иван Павлович, ассистент

Практика пройдена с оценкой \_\_\_\_

Подписи членов комиссии:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

Дата \_\_\_\_

Санкт-Петербург

2019

Оглавление

[1. Цель и задачи 3](#_Toc9907598)

[2. Краткое описание архитектуры х86 3](#_Toc9907599)

[2.1 Модель памяти 3](#_Toc9907600)

[2.2 Регистровая структура 4](#_Toc9907601)

[2.3 Формат машинных команд 5](#_Toc9907602)

[3. О предоставленном инструментарии 6](#_Toc9907603)

[4. Выполнение 6](#_Toc9907604)

[5. Тестирование 7](#_Toc9907605)

[6. Вывод 8](#_Toc9907606)

[7. Список литературы 9](#_Toc9907607)

# Цель и задачи

Все более и более популярными становятся retargetable компиляторы, это понятие означает заточен ли компилятор под конкретную архитектуру, или его реализацией предусмотрены несколько архитектур. Наиболее известными примерами retargetable компиляторов являются GCC и LLVM. GCC поддерживает более 100 процессоров, а LLVM в свою очередь, около 10. Очевидно, что подобные проекты тяжело поддерживать, и на это уходит много ресурсов, даже учитывая то, что разработчики принимают участие в создании процессоров.

Моей основной задачей в рамках данной практики, было написание определения архитектуры х86 в специальной нотации, для тестирования Architecture description language (ADL) предоставленного руководителем практики. Данное описание включает в себя регистровую структуру, модель памяти и инструкции. Такой подход позволяет вынести архитектурно-зависимые модули из компилятора, что значительно упрощает процесс сопровождения ПО.

# Краткое описание архитектуры х86

В данном разделе приведены фрагменты конспекта архитектуры х86.

## Модель памяти

При использовании средств управления памятью процессора, программы напрямую не обращаются к физической памяти. Вместо этого они получают доступ к памяти с помощью одной из трех моделей памяти: плоской, сегментной или в режиме реального адреса:

• Плоская модель— для программы память является единым и непрерывным адресным пространством. Этот пространство называется линейным адресным пространством. Код, данные и стеки содержатся в этом адресном пространстве. Линейное адресное пространство является байт-адресуемым. Адрес для любого байта в линейном адресном пространстве называется линейным адресом.

• Сегментная модель— механизм адресации, обеспечивающий существование нескольких независимых адресных пространств, как в пределах одной задачи, так и в системе в целом, для защиты задач от взаимного влияния. Память для программы делится на непрерывные области памяти, называемые ***сегментами***.

Сама программа может обращаться только к данным, которые находятся в этих сегментах. Операционная система размещает сегменты программы в ОП по определенным физическим адресам, а значения этих адресов записывает в определенные места, в зависимости от режима работы микропроцессора. Таким образом, для обращения к конкретному физическому адресу ОП необходимо определить адрес начала сегмента и смещение внутри сегмента.

• Модель режима реальной адресации— это модель памяти для процессора Intel 8086. Она поддерживается для обеспечения совместимости с существующими программами, написанными для работы на процессоре Intel 8086. В real-address mode используется специфическая реализация сегментированной памяти, в которой линейное адресное пространство для программы и операционный системы состоят из массива сегментов размером до 64 Кбайт каждый. Максимальный размер линейного адресного пространства в режиме реальной адресации 220 байт.

## Регистровая структура

|  |  |  |
| --- | --- | --- |
| Функциональная группа | Имя | Описание |
| Общего назначения | EAX | Аккумулятор, используется в арифметических инструкциях |
| EBX | По умолчанию используется как базовая компонента эффективного адреса операнда, находящегося в памяти |
| ECX | Используется для циклов |
| EDX | Хранит данные во время i/o операций |
| ESI | По умолчанию используется для задания индексной компоненты ЕА, а также для адресации элементов строки-источника в командах обработки строк |
| EDI | По умолчанию используется аналогично SI для задания индексной компоненты ЕА, а также для адресации элементов строки-приемника в командах обработки строк |
| EBP | По умолчанию используется как базовая компонента эффективного адреса операнда в памяти, как EВХ |
| ESP | По умолчанию используется для адресации вершины стека |
| Контроля потока управления | EIP | Счетчик команд, содержит адрес следующей выполняемой инструкции |
| Состояния | EFLAGS | Отражает текущее состояние процессора |
| Сегментные регистры | CS | Содержит адрес сегмента с машинными командами, к которому имеет доступ микропроцессор |
| DS | Хранит адрес сегмента данных текущей программы |
| SS | Содержит адрес сегмента стека |
| ES | Хранят адреса дополнительных сегментов |
| FS |
| GS |
| Отладка | DR0 – DR5 | Содержат линейные адреса точек остнова |
| DR6 | Регистр состояния отладки |
| DR7 | Регистр управления отладкой |

Таблица . Перечень основных регистров, определяемых в составе кода инструкций

## Формат машинных команд

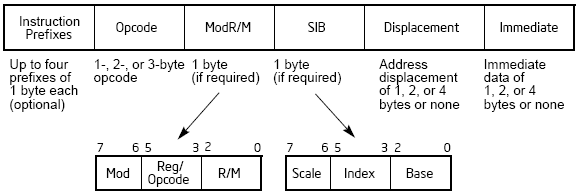


Рисунок формат машинных команд

Префиксы (каждый из них опционален):

* Однобайтовый префикс смены режима адресации AddressSize (значение 67h).
* Однобайтовый префикс изменения сегмента Segment (значения 26h, 2Eh, 36h, 3Eh, 64h и 65h).
* Однобайтовый префикс OperandsSize для изменения размера операнда (значение 66h).
* Префикс Escape. Всегда состоит как минимум из одного байта 0Fh. За этим байтом опционально идёт байт 38h или 3Ah. Предназначен для уточнения инструкции.

и т.д.

Байт ModR/M состоит из трех информационных полей:

* поле mod содержит два наиболее значимых бита и в комбинации с полем r/m определяет 32 возможных типа адресации ( 8 регистровых и 24 индексных)
* поле reg содержит либо номер регистра, либо три дополнительных бита опкода
* поле r/m может определять номер регистра, в котором находится операнд, или

обозначать различные режимы адресации, например 100 будет означать режим displacement-only

SIB байт, в свою очередь, определяет масштабный множитель, номера индексного и базового регистров.

# О предоставленном инструментарии

Нотация описывает архитектуру как набор следующих элементов:

* Регистров и их представлений отображения частей регистров или нескольких регистров под одним наименованием)
* Модели памяти
* Инструкций и их мнемоник
* Функций инструкций
* Форматов операндов инструкций

В ходе работы были выявлены следующие неудобства и баги текущей реализации приложения:

- Регистро-зависимость имен регистров и др. Так же невозможность использования в именах мнемоник слов and, or и т. д., по причине того, что они воспринимаются как ключевые слова

- Некорректная работа с immediate данными, которая заключается в том, что не определяется размер, в который может поместиться значение. Данная проблема не позволяет описать часть инструкций, в которых значения опкодов напрямую зависят от размеров операндов

- Невозможность задания нескольких мнемоник, соответствующих одному опкоду

* Для работы с приложением приходится писать скрипт, т.к. процесс выполнения и получения результатов команд включает в себя множество различных действий

- Вывод результатов напрямую на поток вывода требует дополнительного перенаправления потока

* Недостаточное покрытие текущей реализации документацией

# Выполнение

*Листинг 1. Описание регистровой структуры на примере EFLAGS*

storage EFLAGS\_S [32];

view CF = EFLAGS\_S[0];

view PF = EFLAGS\_S[2];

view AF = EFLAGS\_S[4];

view ZF = EFLAGS\_S[6];

view SF = EFLAGS\_S[7];

view TF = EFLAGS\_S[8];

…

*Листинг 2. Описание набора инструкции на примере ADD*

mnemonic add for add3pref (...) mem-to-reg;

instruction add3pref = {01100110, 0000 0011, sequence rmModByte16};

encode rmModByte16 sequence = alternatives {

nobase = { 00, reg16W1 as reg1, 100, 11, sibIndex as index, 101, disp32 as disp},

scale4WithoutBase = { 00, reg16W1 as reg1, 100, 10, sibIndex as reg2, 101, 0x0000},

scale8WithoutBase = { 00, reg16W1 as reg1, 100, 11, sibIndex as reg2, 101, 0x0000 },

rm00sib = { 00, reg16W1 as reg1, 100 , sequence sibByteWithoutEBP },

rm01sib = { 01, reg16W1 as reg1, 100 , sequence sibByte, disp8 as disp8 },

rm10sib = { 10, reg16W1 as reg1, 100 , sequence sibByte, disp32 as disp32 },

rm00reg = { 00, reg16W1 as reg1, rmMod00AsReg as reg2 },

rm01reg = { 01, reg16W1 as reg1, rmModXXAsReg as reg2, disp8 as disp8 },

rm10reg = { 10, reg16W1 as reg1, rmModXXAsReg as reg2, disp32 as disp32 }

};

# Тестирование

*Листинг 3. Часть теста описания архитектуры*

*aaa*

*aad 255*

*aam 7*

*aas*

*adc AX, [EAX]*

*adc AH, [EAX]*

*adc EAX, [EAX + 100]*

*adc [EAX], EAX*

*adc [100 + EAX+EAX\*8], EAX*

*adc [100 + EAX+EAX\*8], AX*

*adc [100], EAX*

*adc [EAX \* 8], EDX*

*add EAX, EAX*

*add EAX, [EBP]*

*add EDX, [100 + EAX + EAX \*8]*

*add EAX, [EAX\*8]*

Основная часть тестирования проводилась с использованием ассемблера NASM и дизассемблера OBJDUMP. Машинные команды тестового листинга, сгенерированные с помощью предоставленного консольного приложения, полностью совпадают с результатами работы NASM, так же как и мнемоники после дизассемблирования. Результаты тестирования приведены ниже:

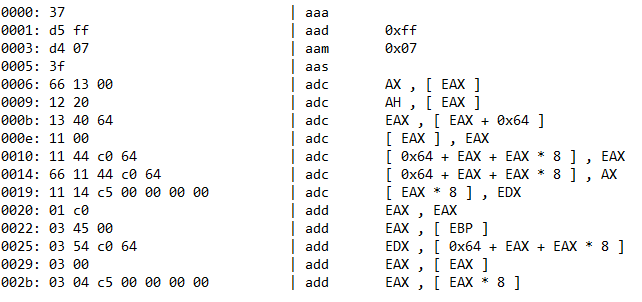


Рисунок . Результат тестирования описания

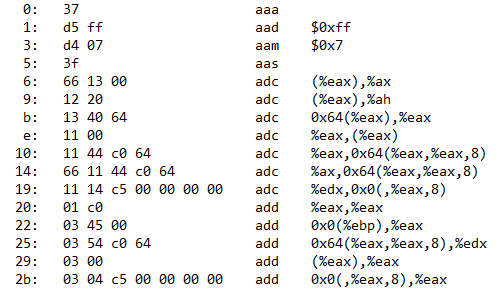


Рисунок . Резлуьтат работы NASM

Дополнительно, результат задачи assembleRaw был дизассемблирован с помощью OBJDUMP. Все инструкции были распознаны верно, включая операнды.

По результатам тестирования можно сделать вывод, что определение составлено верно.

# Вывод

В ходе выполнения работы была изучена архитектура x86, составлен её конспект и описание в специальной нотации, данной руководителем практики. Было проведено успешное тестирование описания. Была изучена официальная документация архитектуры x86, приобретены навыки анализа крупного объёма информации. Были найдены недоработки в текущей реализации инструментария, и предложены некоторые способы их решения.

Данный подход к построению retargetable компиляторов может оказаться весьма востребованным при должной реализации, хоть и много важных вопросов, например, оптимизация, остаются открытыми.

# Список литературы

1. Korenkov I. et al. DECLARATIVE TARGET ARCHITECTURE DEFINITION FOR DATA-DRIVEN DEVELOPMENT TOOLCHAIN //International Multidisciplinary Scientific GeoConference: SGEM: Surveying Geology & mining Ecology Management. – 2018. – Т. 18. – С. 271-278.
2. Intel® 64 and IA-32 Architectures Software Developer Manuals, volumes 1,2

URL: https://software.intel.com/en-us/articles/intel-sdm

1. Developer Guides, Manuals & ISA Documents for AMD processors

URL: [AMD64 Architecture Programmer’s Manual Volume 3: General-Purpose and System Instructions Rev 3.14](https://web.archive.org/web/20030802032632/http:/www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24594.pdf)

**Интернет ресурсы:**

1. <http://ref.x86asm.net/coder32.html> - описание набора команд x86
2. <http://sandpile.org> – краткий сборник информации по прикладной архитектуре х86