Санкт-Петербургский государственный университет

Кафедра Системного Программирования

Болотов Сергей Сергеевич

Разработка компилятора для языка РуСи на платформу MIPS

Бакалаврская работа

Научный руководитель:
д. ф.-м. н., профессор Терехов А. Н.

Рецензент:
Тиунова А. Е.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Department of Software Engineering

Sergei Bolotov

Development of RuC compiler for MIPS platform

Graduation Thesis

Scientific supervisor:
professor Andrey Terekhov

Reviewer:
Anna Tiunova

Saint-Petersburg
2016

Оглавление

Введение **5**

1. Обзор **7**

1.1. Язык РуСи 7

1.2. Компилятор РуСи 7

1.3. LLVM 8

1.4. Компилятор РуСи в LLVM 9

1.5. Clang 9

2. Документация для платформы MIPS и таблица трансляций из РуСи в MIPS **10**

2.1. Общие сведения 10

2.2. АЛУ для работы с целочисленными регистрами 10

2.3. АЛУ для работы с переменными с плавающей запятой . 11

2.4. Таблица трансляции 11

3. Реализация компилятора **14**

3.1. Структура программы на языке РуСи 14

3.2. Структура компилятора 14

3.3. Работа с регистрами 16

3.4. Недостатки реализации 16

3.5. Поддержка РуСи 17

3.6. Отладка и симулятор MARS 17

3.6.1. Тестирование работоспособности 18

3.7. Дальнейшая работа над компилятором 18

4. Сравнение с аналогами **19**

4.1. Сравнение результирующего кода 19

4.1.1. Поиск элемента в массиве 19

4.1.2. Умножение матриц 19

4.1.3. Выводы 20

4.2. Сравнение времени работы компилятора 21

4.2.1. Поиск элемента в массиве 22
4.2.2. Умножение матриц 22
4.2.3. Выводы 22

Заключение **23**

Список литературы **24**

Введение

Создание компилятора для языка программирования — большая и сложная задача. Написание компилятора требует обращения к таким темам, как языки программирования, архитектура компьютера, теория языков, алгоритмы, разработка программного обеспечения[10].

Для упрощения реализации компилятора существует множество методов, одним из которых является создание промежуточных представлений. Такой подход является достаточно успешным в связи с тем, что, несмотря на разнообразие целевых платформ, множество частей нового компилятора можно взять из предыдущей разработки. При этом такую инфраструктуру необходимо дополнять недостающими частями — трансляторами из языка высокого уровня в промежуточное представление, а так же промежуточного представления в коды целевой платформы.

Язык РуСи является языком программирования, разрабатываемым на кафедре системного программирования математико-механического факультета Санкт-Петербургского государственного университета. РуСи предназначен для обучения программированию учеников школ и преподавания студентам СПбГУ курса по трансляции языков программирования. Данный проект находится в стадии развития и в данный момент имеет компилятор в коды виртуальной машины, а также интерпретатор данной виртуальной машины. Проекту недостаёт компиляторов в целевые платформы.

Одной из таких недостающих частей является компилятор для платформы MIPS[4][3]. MIPS — архитектура, получившая широкое распространение как одна из первых RISC[1]-архитектур, до сих пор активно используется, в том числе в качестве архитектуры для новых платформ. Создание компилятора языка РуСи в MIPS будет способствовать более широкому распространению языка, покажет на примере, как разрабатываются компиляторы, а также упростит создание дальнейших

[1]RISC — reduced instruction set computer, архитектура процессора, в котором быстродействие увеличивается за счёт упрощения инструкций.

компиляторов в платформы на RISC-архитектуре, такие как ARM[2] и SPARC[3].

Предметом данной работы является реализация компилятора языка РуСи в MIPS на основе существующего компилятора языка РуСи.

Постановка задачи

Целью данной работы является создание компилятора из языка РуСи в конечные коды платформы MIPS. Для достижения данной цели были сформулированы следующие задачи.

1. Разработать русскоязычную документацию платформы MIPS и таблицу трансляций из кодов РуСи в коды MIPS, достаточную для реализации компилятора.

2. Реализовать оптимизирующий компилятор из языка РуСи в платформу MIPS.

3. Провести сравнение компилятора с существующими аналогами.

[2]ARM — семейство лицензируемых 32-битных и 64-битных микропроцессорных ядер разработки компании ARM Limited.

[3]SPARC — архитектура RISC-микропроцессоров, первоначально разработанная в 1985 году компанией Sun Microsystems.

1. Обзор

1.1. Язык РуСи

Язык РуСи — разработка кафедры системного программирования СПбГУ, целью которой в первую очередь является помощь в преподавании программирования школьникам и студентам. Язык представляет собой подмножество языка программирования C с особенностями, упрощающими программирование на нём. Эти особенности включают в себя:

- функции print и printid, позволяющие печатать значение переменной и значение переменной вместе с идентификатором соответственно, вне зависимости от типа переменной.

- ключевые слова на русском языке, а также возможность объявления переменных на русском языке.

Пример такой программы:

```
цел мой[5] = {1, 2, 3, 4, 5};
пусто главная()
{
печать("мой12345");
печатьид(мой);
}
```

1.2. Компилятор РуСи

Основной работой по языку РуСи является компилятор РуСи в коды виртуальной машины. Это двухпроходный компилятор, результатом работы первого прохода является дерево лексического, синтаксического и видозависимого анализа, результатом работы второго прохода — коды виртуальной машины. Коды виртуальной машины исполняются интерпретатором, никаких оптимизаций результирующего кода или промежуточного дерева не производится. Компилятор изначально не

компилирует код ни в какую конечную платформу, например MIPS. На примере данного компилятора преподаётся курс трансляции языков программирования.

1.3. LLVM

Одной из существующих инфраструктур компиляции является LLVM[6]. LLVM состоит из промежуточного представления LLVM IR[5], а также компиляторов из LLVM IR в целевые платформы. Задачей проекта LLVM является упростить создание компиляторов, позволяя разработчикам компиляторов не создавать множество компиляторов в конечные платформы, а реализовать компилятор из языка высокого уровня в LLVM IR, после чего LLVM самостоятельно оптимизирует код и выполнит компиляцию для целевой платформы. Количество поддерживаемых в данный момент платформ достаточно велико и в том числе включает в себя архитектуру MIPS. Инфраструктура LLVM также позволяет добавлять свои оптимизации кода LLVM IR.

У данной концепции есть так же и недостатки. Во-первых, такая унификация приводит к потере качества оптимизации исходного кода на высокоуровневом языке программирования. При трансляции в промежуточное представление может быть потеряна важная для оптимизации информация. LLVM позволяет частично обходить данную сложность с помощью метаинформации и дополнительных проходов по результирующему коду LLVM IR. Во-вторых, транслятор в конечную платформу реализуется проектом LLVM, таким образом для невостребованных для проекта платформ реализации бывают недостаточно качественными, слишком медленные или с недостатками платформозависимых оптимизаций, а учитывая размеры проекта LLVM, вносить собственные изменения может стоить больших усилий. В-третьих, сам продукт имеет достаточно много зависимостей и внушительный размер после сборки, что может быть неудобно если требуется портативный компилятор.

1.4. Компилятор РуСи в LLVM

Одной из работ по языку РуСи является реализация компилятора для языка РуСи в промежуточное представление LLVM[8]. Компилятор транслирует промежуточное дерево, являющееся результатом первого прохода компилятора РуСи, в промежуточное представление LLVM IR. Результатом работы компилятора является код на языке LLVM IR, над которым LLVM может производить оптимизации и который может быть оттранслирован и скомпилирован в коды конечной архитектуры.

1.5. Clang

Самым популярным компилятором языка C, построенным на основе LLVM, является Clang[2]. Clang — транслятор языка C в LLVM. Изначально Clang задумывался как замена компилятора GCC с более высокой скоростью компиляции и лучшеми сообщениями об ошибках. Так как трансляция производится в промежуточное представление LLVM, то в результате Clang позволяет получить код на множестве конечных платформ из одного исходного кода, в том числе на платформе MIPS.

2. Документация для платформы MIPS и таблица трансляций из РуСи в MIPS

2.1. Общие сведения

MIPS — архитектура 32 и 64 битных RISC процессоров с 5-этапным конвейером, разработана в компании MIPS Technologies. АЛУ[4] процессора MIPS содержит 32 регистра общего назначения, 32 регистра для значений с плавающей запятой и 2 регистра для хранения результатов операций деления и умножения.

2.2. АЛУ для работы с целочисленными регистрами

Регистры общего назначения пронумерованы от $r0 до $r31, их мнемоника и назначение указаны в таблице 1.

Имя	Номер	Назначение	Сохраняет значение после вызова
$zero	r0	Постоянный 0	-
$at	r1	Переменная ассемблера	нет
$v0-$v3	r2-r3	Значение возврата функции	нет
$a0-$a3	r4-r7	Аргументы функции	нет
$t0-$t9	r8-r15, r24-r25	Временные регистры	нет
$s0-$s7	r16-r23	Сохранённые регистры	да
$k0-$k1	r26-r27	Используются в режиме ядра	-
$gp	r28	Указатель на глобальную память	да
$sp	r29	Указатель на стэк	да
$fp	r30	Указатель на фрэйм функции	да
$ra	r31	Адрес возврата	нет

Таблица 1: Таблица целочисленных регистров MIPS

Мнемоника, назначение регистра и сохранение значение после вызова является договорённостью среди разработчиков на платформе MIPS. Разработчик на языке ассемблера может не придерживаться данных правил, но в таком случае функции, реализованные им, не смогут быть переиспользованы в других приложениях. Таким образом, при разработке компилятора необходимо придерживаться указанных в таблице

[4]АЛУ — арифметико-логическое устройство.

1 мнемоник и назначений.

Регистры значения возврата функции хранят значения, возвращаемые функциями после исполнения. Регистры аргументов функции хранят аргументы, передаваемые в функцию при вызове. Временные регистры могут быть использованы разработчиком для вычисления. Сохранённые временные регистры сохраняют своё значение после вызова функции. Регистры $sp и $fp обновляются при инициализации функции и указывают на текущее положение стека и фрейм функции. Регистр $0 всегда возвращает значение ноль.

2.3. АЛУ для работы с переменными с плавающей запятой

Архитектура процессоров MIPS содержит 32 регистра значений с плавающей запятой, пронумерованы от $f0 до $f31. В ранних версиях процессоров каждый регистр имел размер 32 бита, в более новых версиях каждый регистр имеет размер 64 бита. В данной работе эта возможность рассматриваться не будет, так как язык РуСи не поддерживает операции с переменными с плавающей запятой размера 64 бита. На таблице 2 указано назначение регистров.

Номер	Назначение	Сохраняет значение после вызова
f0, f2	Регистры значения возврата функции	нет
f12, f14	Регистры аргументов функции	нет
Чётные f20-f30	Сохранённые регистры	да
Нечётные f1-f31, чётные f0-f10, f16, f18	Временные регистры	нет

Таблица 2: Таблица регистров MIPS значений с плавающей запятой

2.4. Таблица трансляции

Для реализации компилятора из языка РуСи в платформу MIPS необходимо предварительно составить таблицу трансляции. Таблица трансляции показывает, как конструкции ЯВУ[5] будут транслироваться

[5]ЯВУ — язык программирования высокого уровня.

в код на конечной платформе. Трансляция производится из промежуточного представления в виде дерева, которое является результатом выполнения первого прохода компилятора РуСи.

Промежуточное представление в виде дерева представляет собой граф исполнения программы в линейной развёртке. Узлами дерева являются операторы исходной программы и определения функций и переменных. Дерево отражает всю информацию о вычисляемых значениях, известную во время работы компилятора.

В таблице 3 указаны трансляции из дерева в код на платформе MIPS.

Узлы дерева компилятора РуСи	Код на платформе MIPS
Метка TLabel идентификатор label — имя метки	label:
Определение функции TFuncdef идентификатор_функции идентификатор_аргумента f — имя функции	f: ... # код_функции jr $ra Аргументы лежат в $a0-$a3, если более 4-х аргументов — они лежат на стэке. Результат функции хранится в регистре $v0
Определение переменной TDeclid размерность идентификатор начальное_значение начальное_значение — указатель на узел в дереве, где хранится выражение, равное начальному значению, или 0 иначе	Глобальные: строки: str: .asciiz str_value массивы: array: .word w1,..wn скаляры: x: .word x_value С помощью .word задаются как массивы, так и скалярные значения, в т.ч. с плавающей запятой Операции получения глобальных значений: la загружает адрес в регистр, lw загружает значение по адресу, sw сохраняет значение из регистра по адресу Локальные переменные могут храниться на стэке либо в регистрах $s0-$s7

Безусловный переход TGoto идентификатор_метки, TBreak, TContinue label — имя метки	j label
Условный переход	Псевдоинструкции beq (переход, если значения в регистрах равны), bne (переход, если значения в регистрах не равны), bge, ble, bgt, blt. beq $1, $2, label
Оператор цикла TFor инициализация условие инкремент тело TWhile условие тело TDo тело условие В TFor сначала обрабатывается тело, затем инкремент	init_ref: ... # инициализация переменных cond_ref: ... # вычисление условия cond beq cond, $zero, exit ... # тело цикла incr: ... # инкремент счётчика exit:
Условный оператор TIf тело_если тело_иначе если тело_иначе равно 0, то у условного оператора нет метки else, вместо неё используется метка exit	... # вычисление условия condbeq cond, $zero, else_ref if_ref: ... j exit_ref else_ref: ... exit_ref:

Таблица 3: Таблица трансляции

3. Реализация компилятора

Компилятор языка РуСи в MIPS, представленный в данной работе, является оптимизирующим двухпроходным компилятором. Компиляция производится из промежуточного представления в виде дерева, получаемого в результате компилятора РуСи. Компилятор РуСи в MIPS реализован на языке C в виде модуля к компилятору РуСи в коды виртуальной машины, используя его определения, глобальные переменные, в особенности — видозависимое дерево промежуточного представления.

3.1. Структура программы на языке РуСи

Программа на языке РуСи состоит из списка определений и предописаний глобальных переменных и функций. Предописание функции состоит из названия, типа возвращаемого значения и типов принимаемых аргументов. Определение функции состоит из описания функции и следующего за описанием блока исполнения. Блок исполнения состоит из двух частей — списка определения локальных переменных и следующего за ним списка инструкций.

3.2. Структура компилятора

Компилятор РуСи в MIPS использует в процессе работы структуры, указанные ниже.

- Перечисление Registers — сопоставляет название и номер регистра.
- Перечисление Emplacement — сопоставляет значению его место в памяти процессора.
- Перечисление Instructions — содержит список поддерживаемых инструкций конечного процессора.
- Структура IdentEntry — содержит сведения об идентификаторе.

- Структура ValueEntry — содержит сведения о значении внутреннего представления.
- Структура Instruction — содержит сведения об инструкции конечного процессора.
- Структура Operation — содержит сведения об операции внутреннего представления.
- Структура ValueDiff — содержит сведения об изменении значения внутреннего представления.

Компиляция происходит в два прохода. На первом проходе компилятор разбирает промежуточное представление методом рекурсивного спуска. Компилятор разбирает определения глобальных переменных и функций до конца дерева промежуточного представления, после чего печатает файл на языке ассемблера MIPS. В отдельные функции вынесен разбор арифметических выражений, присваивания значений переменным, циклов, условных выражений, блоков исполнения[6], функций и определений. В процессе рекурсивного спуска компилятор разбирает определения и заполняет список идентификаторов в программе (структура IdentEntry). В процессе разбора выражений компилятор заполняет список сведений о значениях вычислений во внутреннем представлении (структура ValueEntry) и стэк производимых над ними операций (структура Operation) на основе сгенерированного списка идентификаторов.

На втором проходе по списку структур ValueEntry и стэку структур Operation компилятор генерирует список инструкций конечной архитектуры (структура Instruction), из которых затем прямым отображением строится код на языке ассемблера MIPS.

Список структур ValueEntry являет собой представление машинозависимого кода в SSA[7] форме, что позволяет производить множество

[6]Блок исполнения в языке РуСи, как и в языке C, начинается с левой фигурной скобки и заканчивается правой фигурной скобкой[9].

[7]SSA — Static Single Assignment, форма промежуточного представления, применяемая в современных оптимизирующих компиляторах.

оптимизаций над кодом [1]. Архитектура компилятора тесно связывает код на конечной платформе и представление в SSA форме, поэтому не представляется возможным в полной мере воспользоваться особенностями представления в виде списка ValueEntry.

3.3. Работа с регистрами

Компилятор хранит значения в переменных регистрах $t0-$t9, во время исполнения сохраняя сведения о текущей переменной, хранящейся в регистре, в массиве temp_regs. В начале цикла или после метки значения в регистрах сбрасываются, т.к. неизвестно, как изменятся значения и в каком состоянии придут по адресу данной метки.

Для выноса вычислений, независимых от исполнения итерации цикла, регистры, значения в которых не изменились, и инструкции, эти значения инициализирующие, во время обработки цикла сохраняются в специальный массив, из которого затем выносятся в область перед циклом.

Для расчёта различных ветвей исполнения в операторе выбора и условном операторе изменения значений, хранимых в регистрах записываются в специальный массив структур ValueDiff, из которого затем обратно преобразуются в первоначальные.

3.4. Недостатки реализации

Основным недостатком архитектуры данной реализации является сильная связанность промежуточного представления и конечной платформы, что не позволяет производить оптимизации над промежуточным представлением, только над кодом конечной платформы. Также реализация предоставляет ограниченные возможности для расширения числа оптимизаций — все оптимизации над промежуточным представлением должны производиться внутри кода разбора дерева.

3.5. Поддержка РуСи

Компилятор РуСи в MIPS поддерживает следующие конструкции РуСи:

- определения функций,
- определения глобальных и локальных переменных,
- конструкции циклов for, while и do-while,
- условный оператор if,
- оператор выбора switch,
- операции над целочисленными значениями, в том числе операторы && и ||,
- операции над значениями с плавающей запятой,
- присвоения значений в переменные и элементы массивов,
- вызовы функций, в том числе поддерживаются функции как аргументы,
- функцию print для скалярных значений.

Компилятор РуСи в MIPS не поддерживает следующие конструкции РуСи:

- функцию print для массивов,
- функцию printid.

3.6. Отладка и симулятор MARS

Для отладки кода на языке ассемблера использовался симулятор платформы MIPS MARS[7] — графический симулятор языка ассемблера MIPS. Особенностями данного симулятора являются счётчик числа выполненных инструкций, возможность пошагового исполнения кода

на языке ассемблера и табличное представление текущих значений в памяти и в регистрах.

3.6.1. Тестирование работоспособности

Работоспособность тестировалась отдельно для каждой из конструкций, указанных в пункте "Поддержка РуСи" — создавалась программа на языке РуСи, а также эквивалентная данной программа на языке C, покрывающая некоторое множество различных случаев использования конкретной конструкции, программа компилировалась и исполнялась на симуляторе, результат её работы сравнивался с результатом работы эквивалентной программы на языке C.

3.7. Дальнейшая работа над компилятором

Дальнейшая работа над компилятором РуСи в MIPS состоит в:

- разделении промежуточного представления в SSA форме и кода на конечной платформе для реализации машинно-независимых оптимизаций,
- дополнении работы с регистрами для уменьшения числа операций перемещения значений между регистрами,
- расширении реализации для поддержки различных версий архитектуры MIPS,
- тестировании реализации на реальном аппаратном обеспечении (вместо существующего тестирования на эмуляторе).

4. Сравнение с аналогами

4.1. Сравнение результирующего кода

Важнейшим показателем в оценке качества компилятора является скорость исполнения компилируемой программы. Одной из самых точных оценок данной характеристики является количество инструкций процессора, потраченных на исполнение компилируемой программы.

Сравнение производилось на симуляторе MARS, который позволяет подсчитать инструкции, необходимые для выполнения инструкций до точки прерывания или до конца файла. Для сравнения использовались программы, приведённые в листингах ниже.

4.1.1. Поиск элемента в массиве

Для поиска элемента в массиве исполнялся код из листинга 1. Код компилируется как в языке РуСи, так и в языке C. Результат исполнения кода в обоих языках одинаковый. Результат сравнения приведён в таблице 4

```
int x[50] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
             11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
             21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
             31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
             41, 42, 43, 44, 45, 46, 47, 48, 49, 50};
void main()
{
    int i = 0;
    while(i < 50 && x[i] != 43)
        ++i;
    return;
}
```

Рис. 1: Код поиска элемента в массиве

4.1.2. Умножение матриц

Для умножения матриц исполнялся код из листинга 2. Код компилируется как в языке РуСи, так и в языке C. Результат исполнения кода

	Количество затраченных на исполнение инструкций
clang -O0	1291
компилятор РуСи в LLVM -O0	695
компилятор РуСи в MIPS	821
clang -O2	305
компилятор РуСи в LLVM -O2	276

Таблица 4: Количество затраченных инструкций для поиска элемента в массиве

в обоих языках одинаковый. Результат сравнения приведён в таблице 5.

```
int x[50][50];
int y[50][50];
int z[50][50];
int multiply(int i, int j)
{
    int ret = 0, it = 0;
    for(it; it < 50; ++it)
        ret += x[i][it] * y[it][j];
    return ret;
}
void main()
{
    int i, j;
    for(i = 0; i < 50; ++i)
        for(j = 0; j < 50; ++j)
            z[i][j] = multiply(i, j);
    return ;
}
```

Рис. 2: Умножение матриц

4.1.3. Выводы

Приведённые результаты показывают, что, хотя компилятор показывает хорошие результаты в сравнении с компилятором clang без оптимизаций, отстаёт от компилятора clang с оптимизациями и от компилятора в представление LLVM. Это происходит вследствие того, что clang и LLVM имеют:

	Количество затраченных на исполнение инструкций
clang -O0	46839
компилятор РуСи в MIPS	38160
clang -O2	4023

Таблица 5: Количество затраченных инструкций для умножения матриц

- более строгую архитектуру промежуточного представления — LLVM, на основе которого построен clang, имеет строготипизированную трёхадресную SSA форму, над которой возможно производить множество независимых друг от друга операций.
- большее количество оптимизаций — проект LLVM развивается с 2004 года, имеет открытый исходный код и множество разработчиков, что позволяет поддерживать большее число оптимизаций промежуточного представления, а также постоянно разрабатывать новые. Примеры оптимизаций LLVM, не поддерживающихся в компиляторе РуСи:
  - удаление циклов,
  - удаление неиспользуемого кода,
  - обнаружение хвостовой рекурсии и пр.
- лучшая работа с возможностями конечной платформы.

4.2. Сравнение времени работы компилятора

Важной характеристикой компилятора является скорость компиляции файла исходного кода. Большее число оптимизаций ведёт к замедлению скорости компиляции, в то же время компилятор с большим числом библиотек, от которых он зависит, будет работать медленнее и без оптимизаций.

4.2.1. Поиск элемента в массиве

В таблице 6 указано время, затраченное на компиляцию файла, содержащего программу, исполняющую поиск элемента в массиве.

	Затраченное на компиляцию время, 100 запусков, с
clang -O0	3,8
компилятор РуСи в LLVM -O0	8,5
компилятор РуСи в MIPS	0,7
clang -O2	4,1
компилятор РуСи в LLVM -O2	9,0

Таблица 6: Затраченное на компиляцию время

4.2.2. Умножение матриц

В таблице 7 указано время, затраченное на компиляцию файла, содержащего программу, исполняющую умножение матриц.

	Затраченное на компиляцию время, 100 запусков, с
clang -O0	4,0
компилятор РуСи в MIPS	0,6
clang -O2	4,9

Таблица 7: Затраченное на компиляцию время

4.2.3. Выводы

Компилятор РуСи в MIPS, не имеющий внешней зависимости от большой библиотеки LLVM, в отличие от clang и компилятора РуСи в LLVM, работает на порядок быстрее.

Заключение

В рамках данной квалификационной работы были достигнуты следующие результаты.

1. Разработана документация платформы MIPS и приведены примеры трансляции кода на языке РуСи на платформу MIPS.

2. Реализован компилятор языка РуСи для платформы MIPS.

3. Произведено сравнение компилятора с аналогами.

Код проекта доступен на сайте:
`https://github.com/CepGamer/RuC-MIPS`.

Список литературы

[1] Appel Andrew W. Modern compiler implementation in C. — Cambridge university press, 2004.

[2] C. Lattner. C Language Family Frontend for LLVM. — URL: `http://clang.llvm.org`.

[3] D. Sweetman. See MIPS run. — Morgan Kaufmann, 2007.

[4] J. Heinrich. MIPS R4000 Microprocessor User's manual. — MIPS technologies, 1994.

[5] Lattner C. Adve V. LLVM Language Reference Manual. — URL: `http://llvm.org/docs/LangRef.html`.

[6] Lattner C. Adve V. LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation. — University of Illinois at UrbanaChampaign, 2004.

[7] Vollmar K., Sanderson P. MARS: an education-oriented MIPS assembly language simulator. — 2006. — Vol. 38, no. 1. — P. 239–243.

[8] Болотов С. Создание транслятора из РуСи в LLVM IR. — 2015. — URL: `http://se.math.spbu.ru/SE/YearlyProjects/spring-2015/list`.

[9] Керниган Б., Ритчи Д. Язык программирования С. — Невский диалект СПб., 2001.

[10] Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Д. Ульман, М. Лам. — Вильямс, 2008.