# Модульная система трансляции с возможностью обратимой трансляции для ЯВУ, машинно-независимым исполнением программ и поддержкой JIT-компиляции

*Данила Байгушев, 8-9 класс*[1]

*Научный руководитель: И.Р. Дединский, МФТИ*

**Введение**

Основная цель работы – разработка транслятора для преобразования языков программирования друг в друга, в том числе высокоуровневых (ЯВУ), с возможностью преобразовывать тексты программ с одного ЯВУ на другой и исполнять программы с использованием JIT-компиляции.

Среди языков программирования существуют как промышленные языки (`C, C++, Java, C#` и др.), так и экспериментальные, а также так называемые эзотерические языки программирования, разработанные ради шутки (`LOLcode, Chief, Shakespeare` и др.) [1] (рис. 1). На эзотерических языках зачастую трудно писать, что объясняется их специфическими особенностями. В этой связи актуальной становится задача автоматической трансляции между языками.

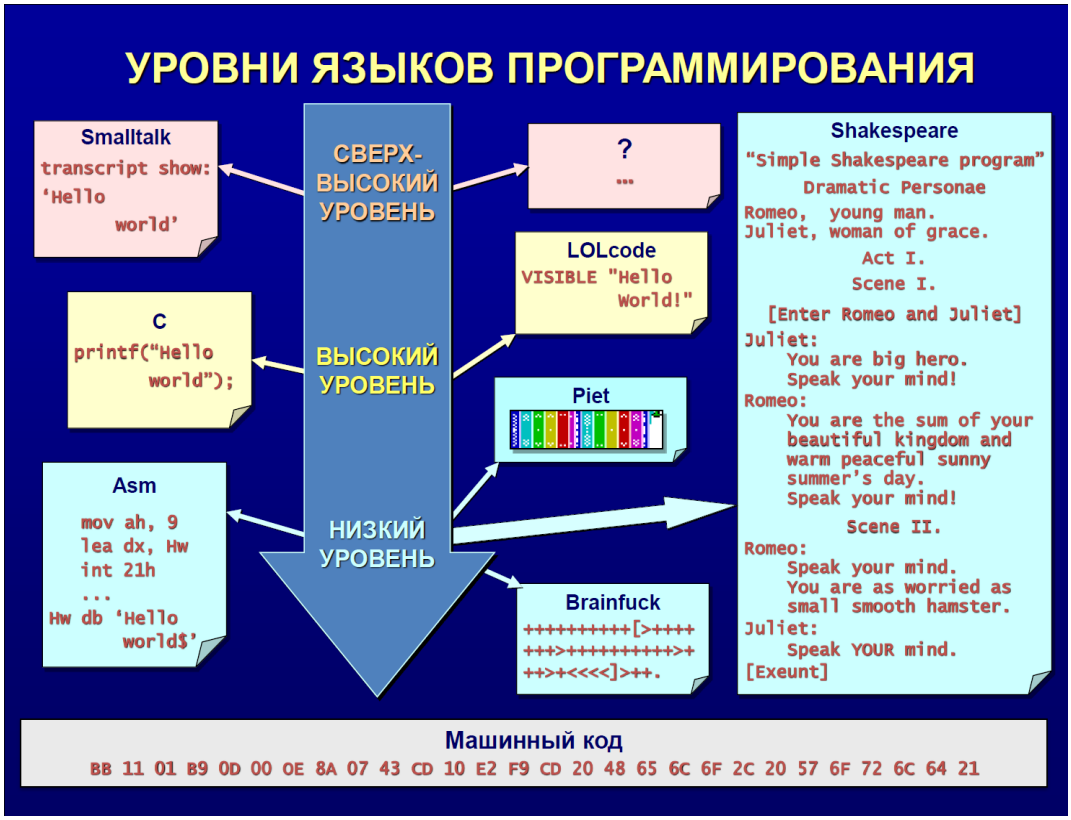



Рис. 1. Различные языки программирования.

[1] Электронная почта: idanila24@gmail.com.

**Система трансляции**

Разработанная система такой трансляции (`Small Compiler Collection, SCC`) имеет модульную структуру и напоминает известную систему компиляции `GCC` [2] (рис. 2). Это сделано для того, чтобы можно было легко добавлять в коллекцию новые языки. Основная идея – хранение всех программ в универсальном внутреннем представлении, абстрактном синтаксическом дереве (AST) [3]. Транслятор представляет собой систему модулей, позволяющих получать из программы такое дерево, и наоборот. Этот способ позволяет для добавления нового языка L всего лишь написать два модуля – фронт-энд (front-end, ответственен за преобразование из данного языка L в AST) и бэк-энд (back-end, ответственен за преобразование из AST в данный язык L) и таким образом получить возможность трансляции из любого языка $L_1$ в любой язык $L_2$, из имеющихся в коллекции.


Рис. 2. Система трансляции SCC.

В качестве основы был разработан стандарт AST (`SCC_tree`), позволяющий хранить программу в удобной форме. В его основе – неполное бинарное дерево, так как оно достаточно быстрое, и, в то же время, не требует много памяти, и хорошо подходит для описания структуры программы. При разработке учитывалось два взаимосвязанных фактора: дерево должно быть достаточно простым, для того, чтобы было легко переводить из него в другой язык программирования, и оно должно быть достаточно «выразительным» (атрибутированным), чтобы программа, переведённая из языка высокого уровня в дерево была «узнаваема». Для хранения дерева был разработан определенный формат файла. Также для упрощения разработки был разработан визуализатор для синтаксических деревьев разработанного формата [4] на основе программы `Dot` [5], транслирующий AST в изображение в формате JPEG.

Также был разработан комплект модулей, выполняющих различные операции с AST.

Один из таких модулей предназначен для оптимизации программ. Он занимается вычислением выражений, результат которых известен до выполнения программы (сверткой констант) и рядом других оптимизаций. Работу этого модуля можно наглядно рассмотреть, произведя трансляцию из некоторого языка L в этот же язык L (см. пример 2).

Также был разработан редактор связей (linker) – модуль для упрощения перевода AST в программы. Он создаёт информацию (`SCC_tree_info`) об AST. Основная задача этого модуля – перейти от названий переменных и функций к номерам и распределить между переменными и массивами память. Он также упрощает процесс транслирования программы в дерево, т. к. при создании дерева не требуется нумеровать переменные и функции, распределять память. В отличие от полнофункционального линкера, он не предназначен для сборки программы из нескольких модулей.

Трансляция из одной программы в другую идёт в несколько этапов. Сначала по исходной программе строится дерево. Затем это дерево оптимизируется. После этого по ее синтаксическому дереву линкером строится информационный файл и с помощью этого файла и AST строится программа на языке, в который производится трансляция.

**Процедура трансляции**

В качестве «стандартного» языка программирования был выбран язык C. На основе его подмножества был создан скриптоподобный язык высокого уровня (`SCC_C`). В этом языке есть переменные, условия, циклы, массивы, функции с возвращаемыми значениями, ссылки (рис. 3). Для того, чтобы построить AST, исходный текст разбивается на лексемы и затем, при помощи рекурсивного спуска, строится дерево синтаксического разбора (см. рис. 4-7). Использование лексического анализа существенно ускоряет работу программы.

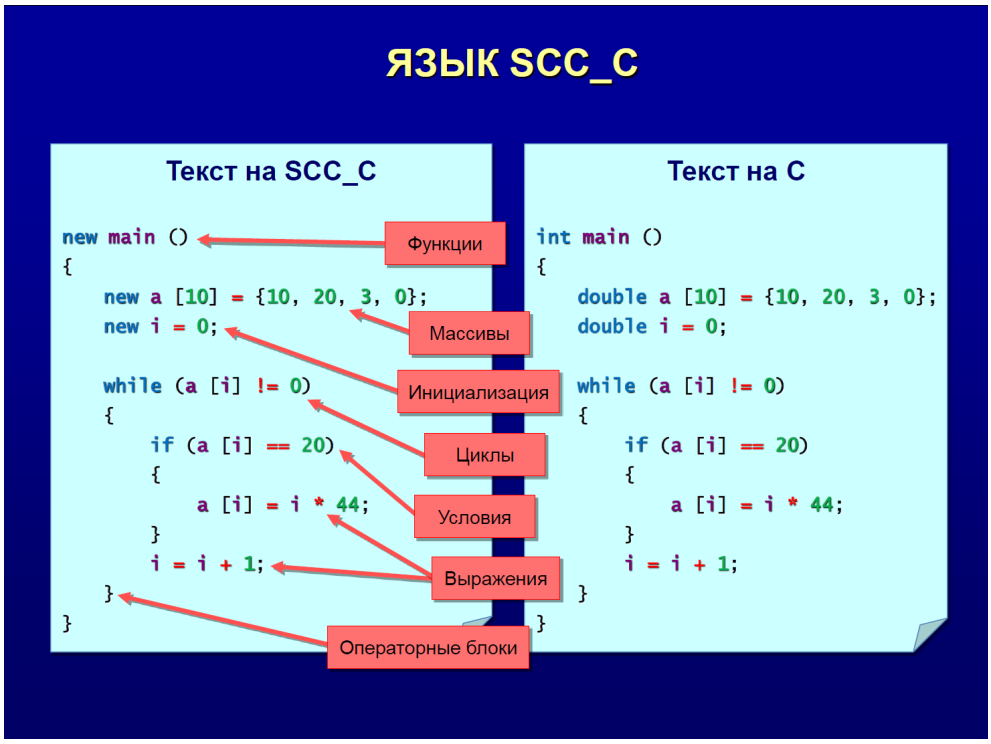



Рис. 3. Пример кода на языке `SCC_C`.

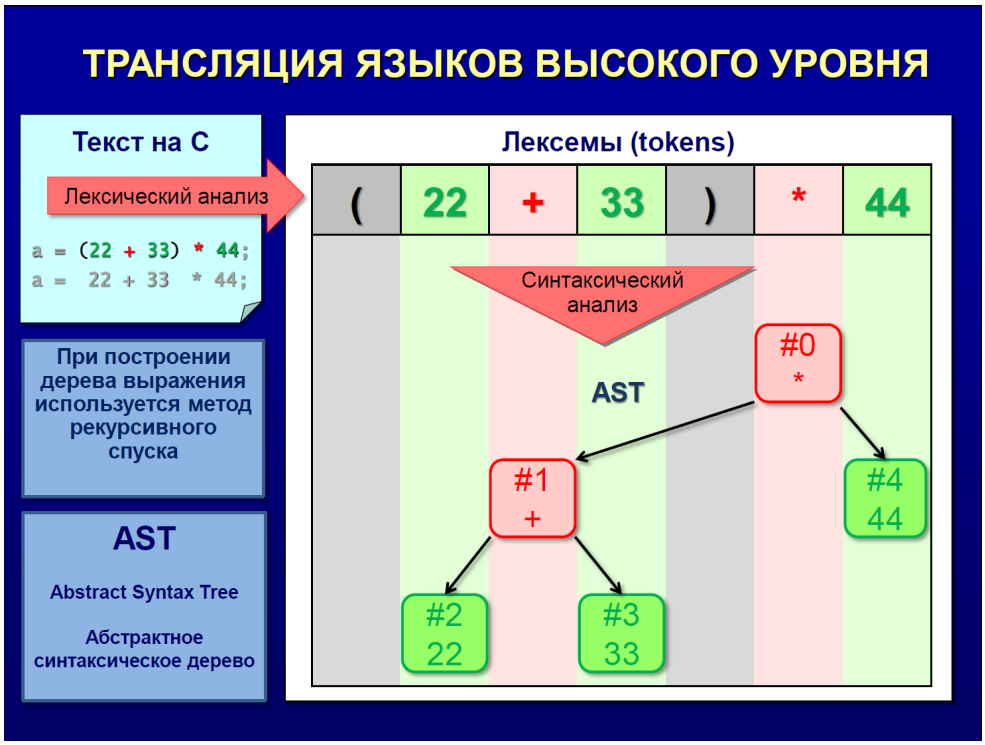



Рис. 4. Пример построения дерева простого выражения.

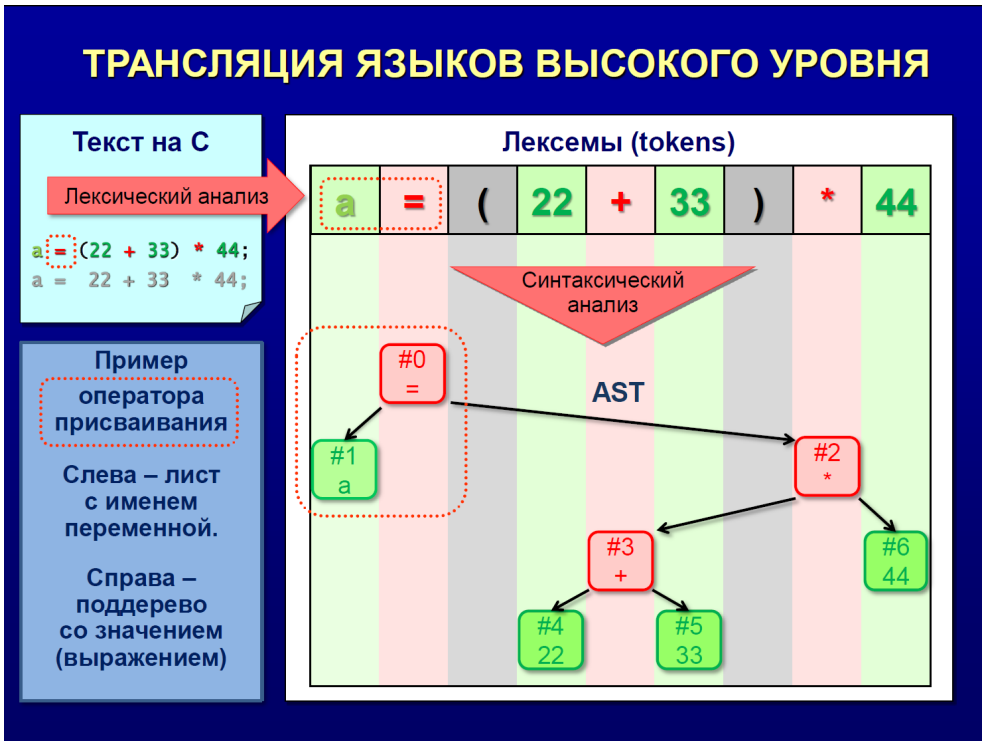



Рис. 5. Построение дерева после добавления в язык оператора присваивания.

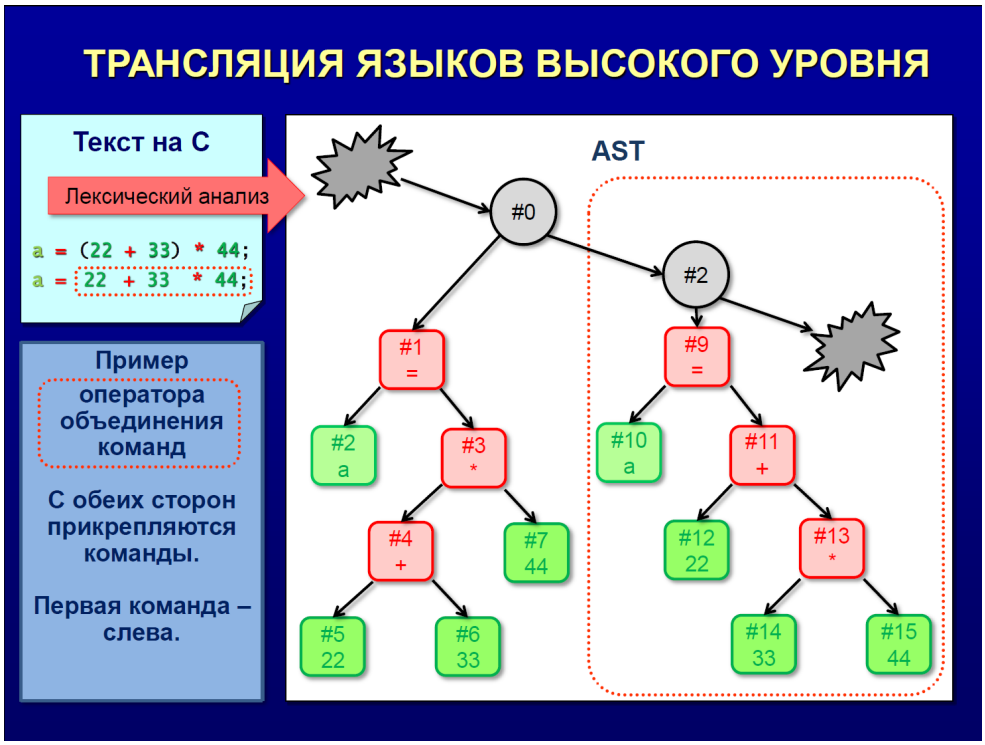



Рис. 6. Пример разбора оператора объединения команд.

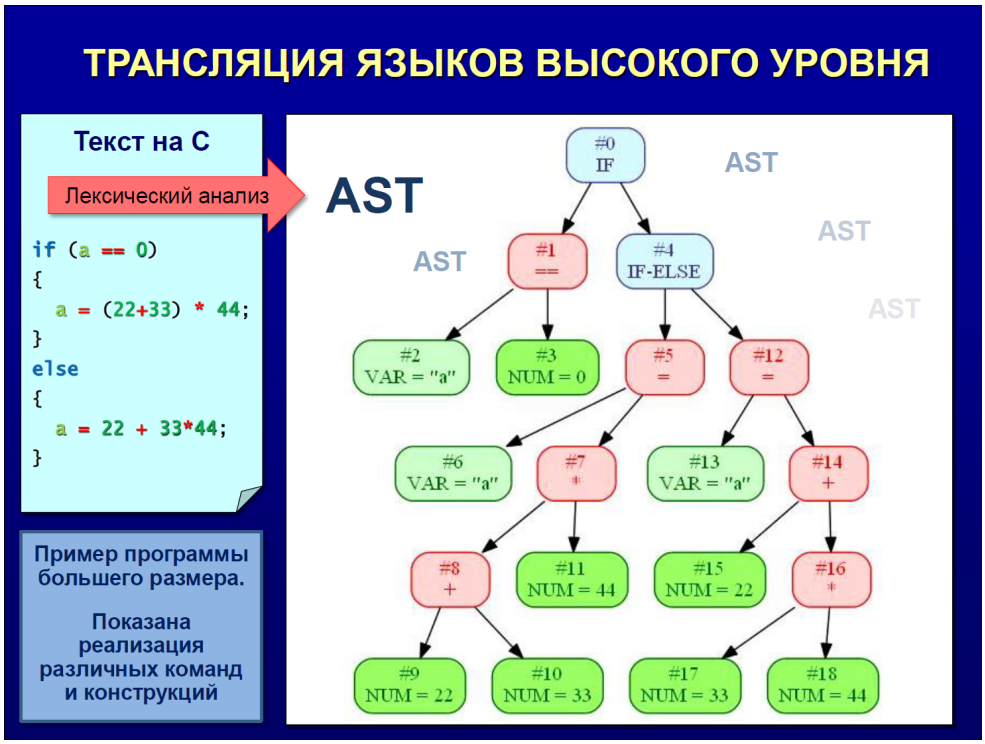



Рис. 7. Пример разбора более сложной программы.

Для него также был разработан независимый препроцессор, реализующий подключение библиотек и удаление комментариев.

В качестве другого языка в коллекции был выбран эзотерический язык программирования «Шекспир» [6]. Программы на нем напоминают литературные произведения (пьесы), но отличаются значительным многословием (рис. 8, 9). В оригинальной версии языка отсутствует поддержка вызова функций, но в рамках данной работы язык был несколько расширен и эта поддержка была реализована. Доработанная версия языка сохранила полную обратную полную совместимость с стандартным вариантом. Трансляции из AST в этот язык было уделено особое внимание.

Самим сложным оказалась поддержка «литературности» языка. Благодаря этому программы на конечном языке (`SCC_spl`), сгенерированные компьютером интересно читать (см. пример 1 в конце статьи). Для реализации языка были использованы базы слов, взятые из стандартной реализации языка «Шекспир».


Рис. 8. Пример синтаксиса языка «Шекспир». Синтаксис констант.

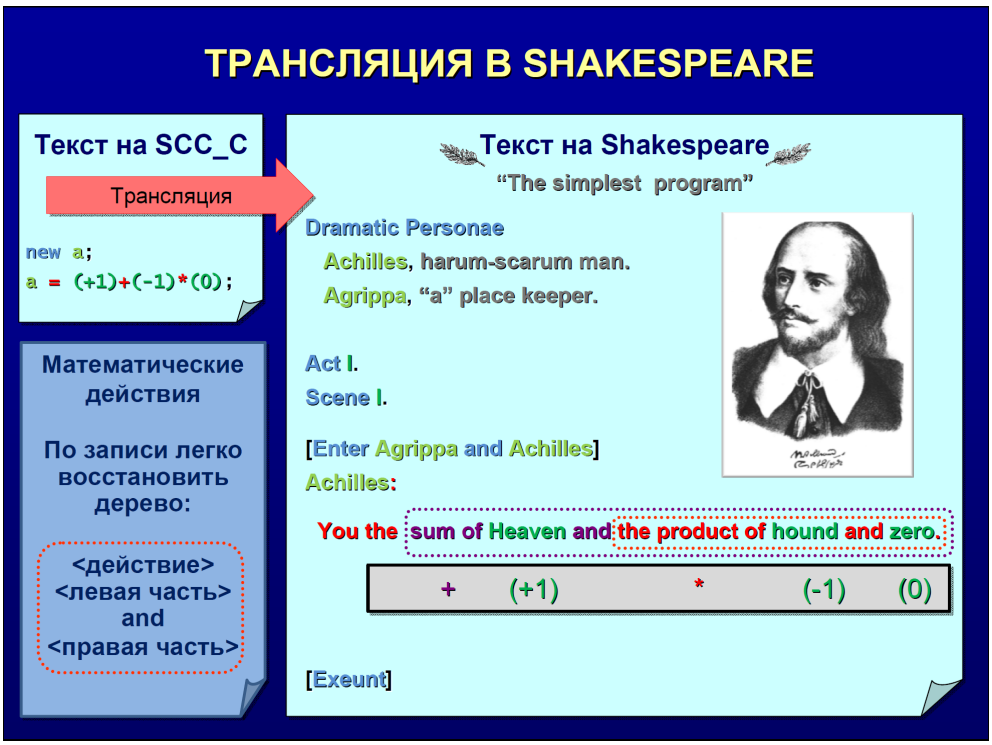



Рис. 9. Пример синтаксиса языка «Шекспир». Синтаксис выражений.

В качестве языка низкого уровня (ассемблер) был разработан собственный язык (`SCC_asm`), реализующий стековые операции, базовую арифметику, условные и безусловный переходы, вызовы функций, работу с памятью, ввод и вывод информации на экран.

**Исполнение кода**

Для исполнения программ был разработан программно реализованный процессор стеково-регистровой архитектуры. Этот процессор занимается покомандной интерпретацией `SCC_asm`, предварительно преобразованный в бинарные опкоды исполнимого файла (`SCC_exe`), для увеличения скорости работы разработанного процессора (см. рис. 10).

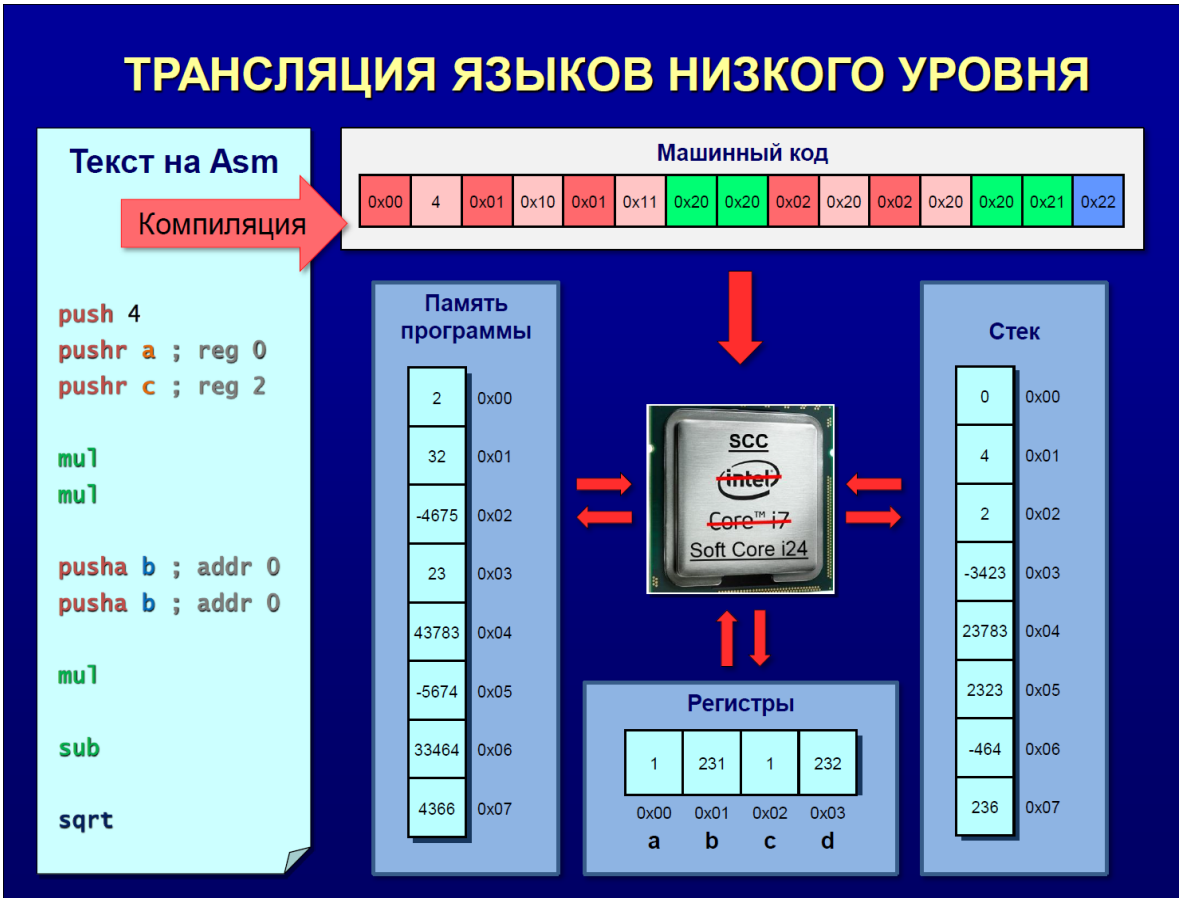



Рис. 10. Пример трансляции языков низкого уровня.

Для увеличения скорости исполнения программ был также разработан вариант софт-процессора с возможностью JIT-компиляции бинарного кода (`SCC_bin`), полученного из языка ассемблера софт-процессора, в машинные коды семейства процессоров Intel 80x86 (32-bit). При компиляции более 70% инструкций, включая команды математического сопроцессора (FPU), заменяются эквивалентными последовательностями машинных кодов, а сложные команды (такие как ввод, вывод и работа с памятью) заменяются на вызов функций, реализованных в `C++`.

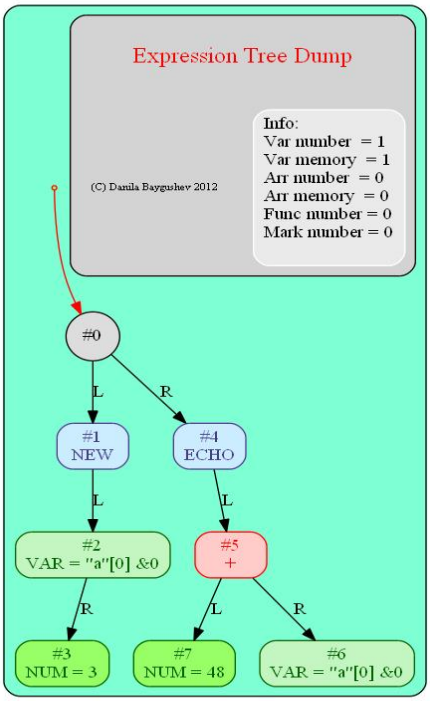
Для создания бинарных кодов `SCC_exe` и `SCC_bin` был разработан компилятор, использующий уже готовый модуль препроцессора `SCC_C`.

Также было написано несколько демонстрационных программ, которые показывают высокую эффективность JIT-компиляции над покомандной интерпретацией. Показан значительный прирост в скорости (2.5 – 3 раз, в отдельных случаях до 5 раз).

Работа выполнена на языке `C++` (GCC g++ v.4.7.2), в среде CodeBlocks [7], консольные модули написаны для операционной системы Windows, программные библиотеки разработаны как кросс-платформенные.

**Примеры трансляции**

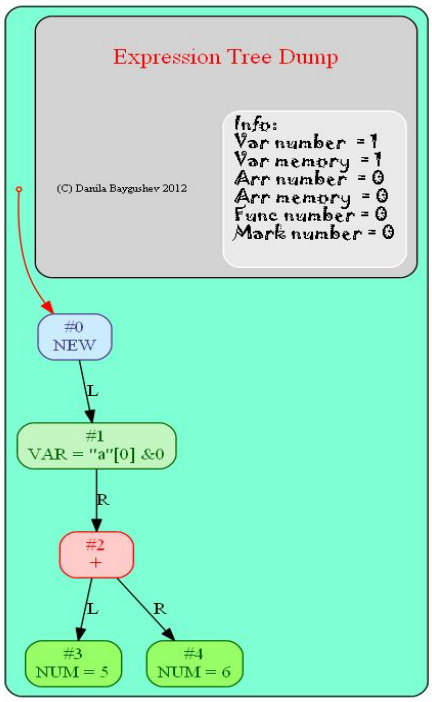
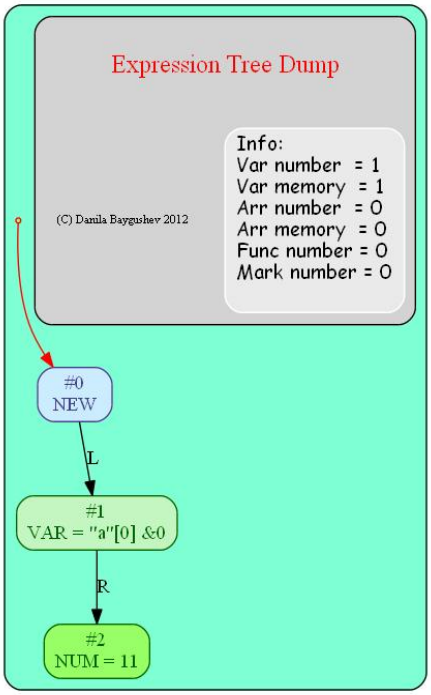
1. Трансляция из SCC_C в язык «Шекспир»:

| **Исходный текст на SCC_C** | **После препроцессора** | **Дерево из SCC_C после оптимизатора и линкера** |
|---|---|---|
| `new a = 3;`<br>`echo '0' + a;` | `new a = 3;`<br>`echo 48 + a;` |  |

Программа на языке «Шекспир», сгенерированная из AST:

```
/*{ This program was automatically generated from the tree.
    (c) Danila Baygushev, all rights reserved. }*/

Dramatic personae:

Achilles, harum-scarum man, Temp1.
Agrippa,  memory for variable #0.

Act I.
Scene I.

[Enter Agrippa and Achilles]

Achilles:
- You are as dirty as the difference between blue horrid Heaven and hamster.

Agrippa:
- You the sum of the difference between the square of the difference between
  distasteful fair hard cousin and town and rose and Agrippa.
- Speak your mind.

[Exeunt]
```

2. Трансляция из `SCC_C` в `SCC_C`:

| **Исходный текст `SCC_C`** | **Дерево из `SCC_C` после линкера** | **Дерево из `SCC_C` после оптимизатора** |
|---|---|---|
| `new a = 5 + 6;` |  |  |

Программа на языке `SCC_C`, сгенерированная из AST:

```
new a = 11;
```

**Заключение. Результаты работы**

Этот проект отличается от уже существующих решений тем, что он поддерживает трансляцию языков разных уровней и включает трансляцию в экзотерические языки.

Было продемонстрировано, что возможно производить эффективные трансляции из «обычных» языков в экзотерические. Также ещё раз было показано превосходство JIT-компиляции над обычным интерпретированием.

Результаты работы:

1) Разработан формат AST для единого внутреннего представления программ.
2) Разработан оптимизатор для AST.
3) Разработана независимая система визуализации AST.
4) Реализован высокоуровневый C-подобный скриптовый язык.
5) Реализован препроцессор для этого языка.
6) Реализован язык программирования «Шекспир», в него добавлена поддержка вызова функций.

7) Реализована модульная система транслятора, позволяющая легко расширять ее, добавляя новые модули и языки.
8) Реализован софт-процессор для исполнения программ.
9) Реализован вариант софт-процессора с возможностью JIT-компиляции в опкоды семейства процессоров Intel 80x86 (32-bit).
10) Проведено сравнение двух вариантов софт-процессоров.

Исходный код (65 файлов, более 17 тыс. строк на C++) и исполняемые файлы доступны в репозиториях автора на GitHub: http://github.com/discharged-spider/SCC и в Google Code: https://code.google.com/p/scc-small-compiler-collection.

Среди дальнейших возможностей развития проекта стоит выделить:

1) Расширение коллекции языков. Ведутся работы по добавлению языка LOLCode.
2) Разработка графического пользовательского интерфейса. Существующий набор консольных приложений, удобный опытным разработчикам, труден для понимания неискушенными пользователями.
3) Кроссплатформенная реализация консольных модулей. Так как все библиотеки не зависимы от ОС, стоит расширить количество поддерживаемых ОС.

**Литература**

1. http://en.wikipedia.org/wiki/Esoteric_programming_language
2. http://www.gnu.org/software/gcc
3. http://en.wikipedia.org/wiki/Abstract_syntax_tree
4. http://ded32.net.ru/news/2012-09-23-63
5. http://www.graphviz.org/pdf/dotguide.pdf
6. http://shakespearelang.sourceforge.net
7. http://www.codeblocks.org