

Платформа для машинно-независимого исполнения программного кода с возможностью JIT-компиляции

Цель:

Разработка набора средств для работы с программным кодом и его исполнения в стековой виртуальной машине с возможностью JIT-компиляции в машинный код центрального процессора.

Автор:

Шаповалов Иван, 10 «В» класс

Научный руководитель:

Дединский Илья Рудольфович

Задачи

(принципиальные)

Обеспечение:

- Кроссплатформенность
- Простой и минималистичный API
- Работа с несколькими входными языками
- Возможность работы с загруженными данными
- Платформено-независимое исполнение кода
- Возможность JIT-компиляции для некоторых архитектур CPU

(функциональные)

Реализация:

- Абстракция от ОС
- Модульная конвейерная архитектура
- Наличие обобщённого внутреннего представления
- Поддержка объектных файлов
- Компоновщик
 - Раздельная компиляция
 - Динамическое связывание
- Стековая виртуальная машина
- JIT-компиляция для архитектуры Intel IA-32E

Архитектура системы

Модули (основная функциональность)

Front-end

MMU

Компоновщик

Логика
исполнения

Интерфейсы (неизменяемые)

Ядро системы

Типы данных
integer, float

Определения
байт-кода

Состояние VM
регистры

API

Слой взаимодействия с ОС

Обработчик
исключений

Абстракция
POSIX/Win32

Подсистема
отладки

Реализация

- Конвейерный принцип
- Каждый этап — отдельный модуль
- API — обёртка над системой модулей, вызывающая их в нужном порядке

Загрузка исходного текста

- Генерация байт-кода
- Абстракция от языка исходного текста

Хранение (контекст)

- Сериализация в объектный файл
- Загрузка из объектного файла
- Прямое изменение данных контекста

Компоновка

- Разрешение имён
- Объединение нескольких исходных файлов
- Повторная обработка ранее скомпонованных модулей

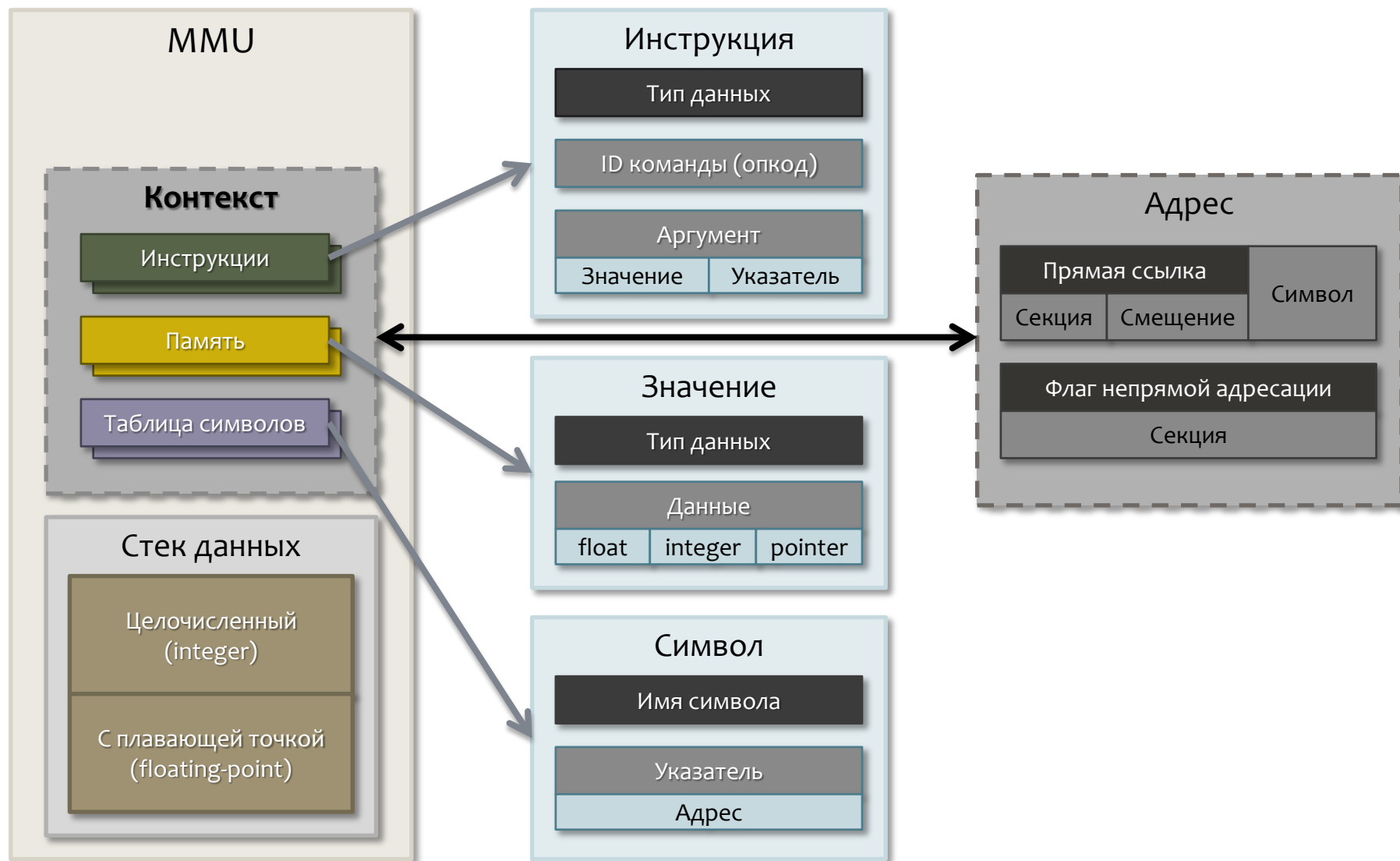
Исполнение

- Исполняемая единица — контекст
- По умолчанию используется виртуальная машина

Возможна
JIT-компиляция

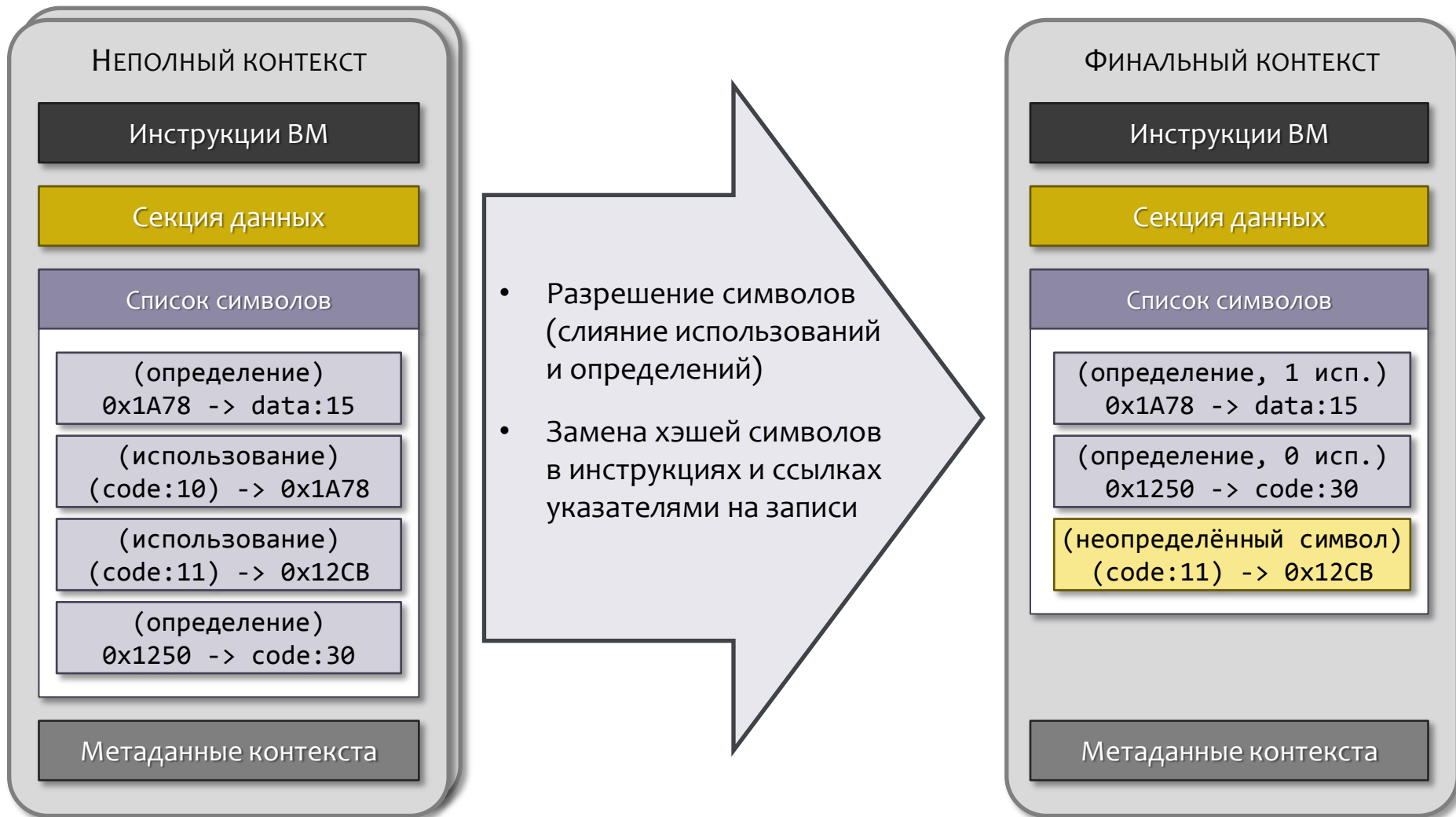
Байт-код и его структура

- Конвейер на всех шагах работает только с байт-кодом.



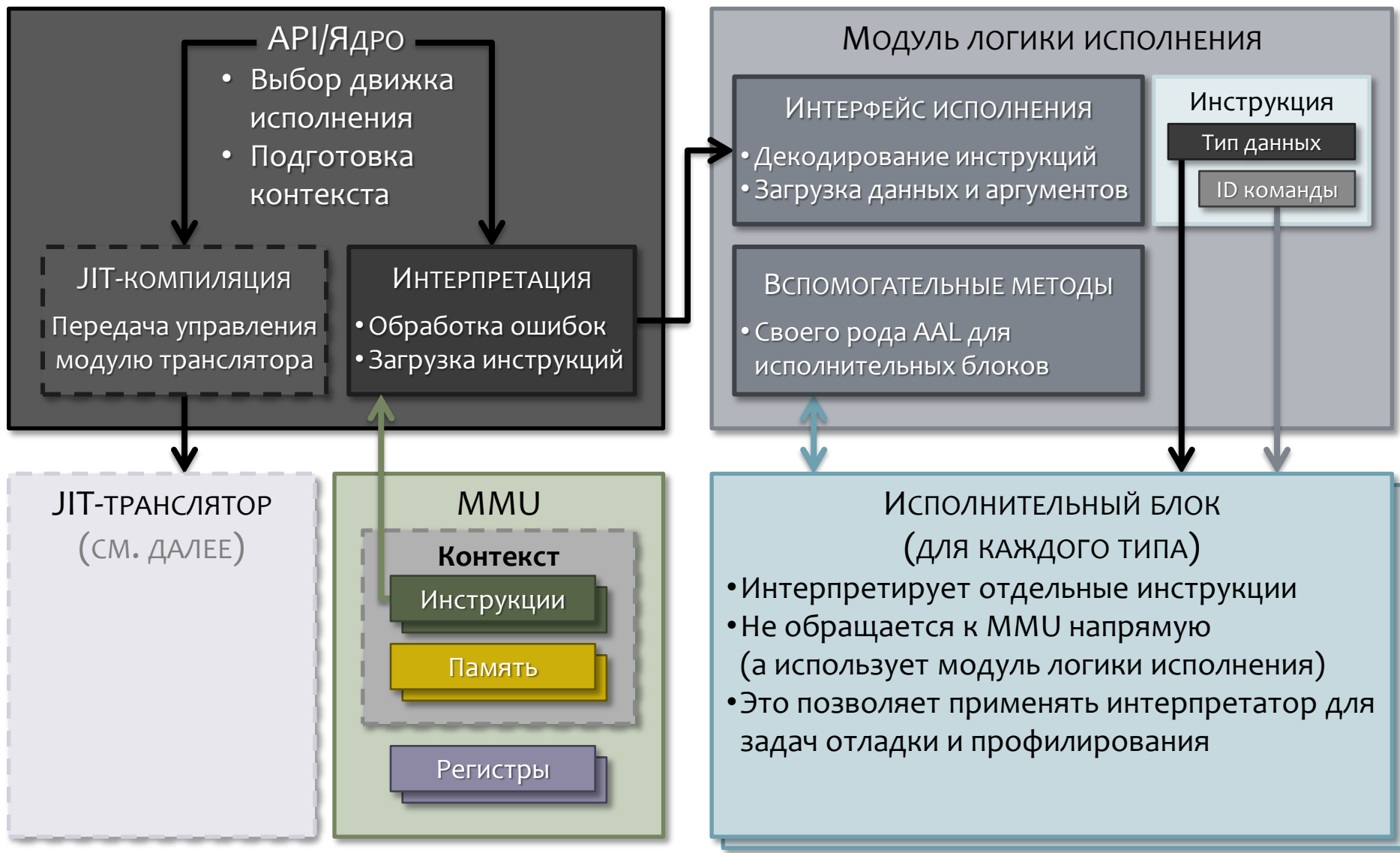
Компоновщик

- Подготовка контекста к исполнению
- Объединение нескольких контекстов



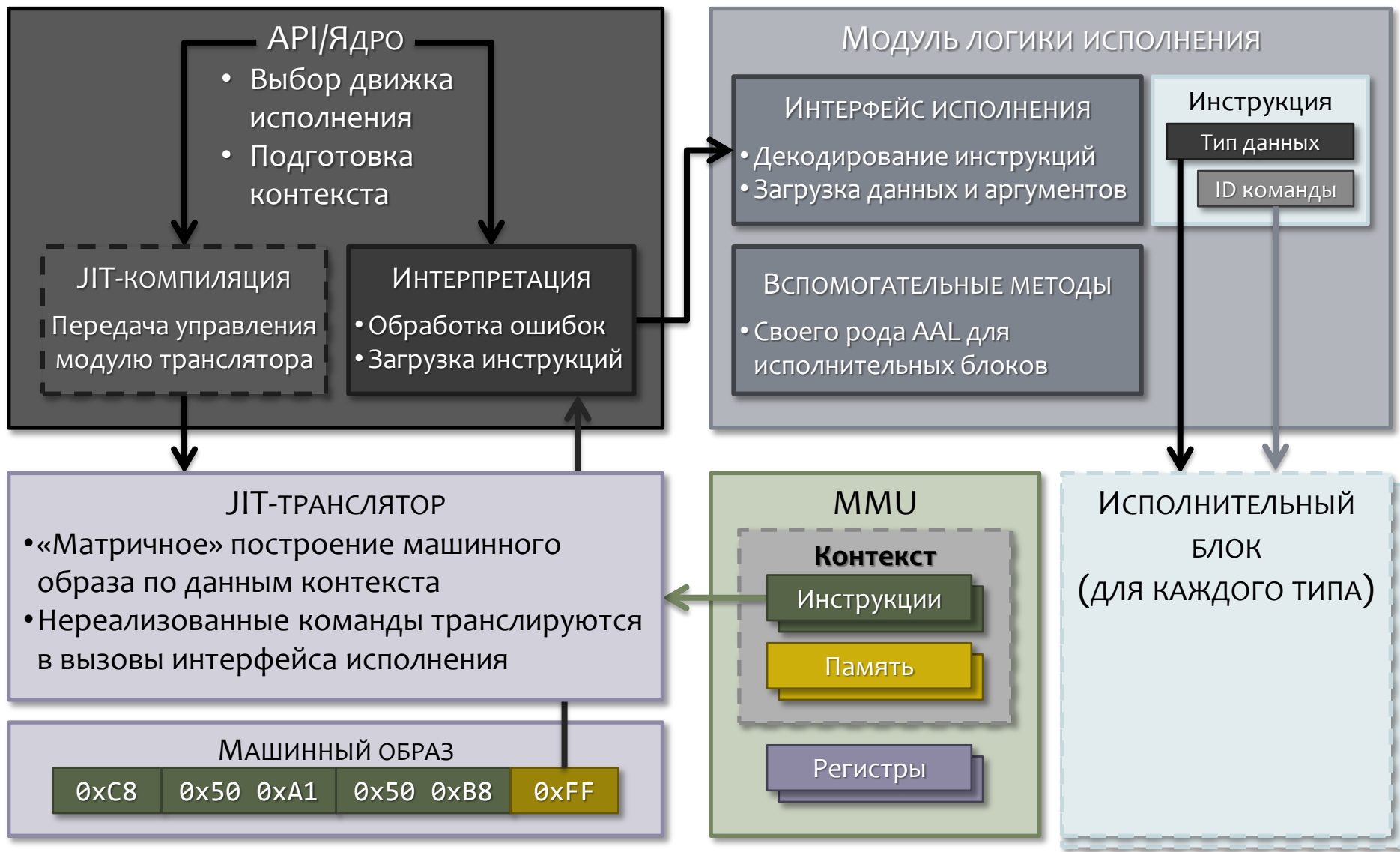
Исполнение

- Основной способ исполнения — интерпретация



JIT-компиляция

- JIT рассматривается как быстрая замена интерпретации.



Особенности генерации кода

- Тип генерируемого кода – стековый (без выделения регистров)
- Виртуальный стек моделируется аппаратным (вершина в **RAX**)
- Поддерживается платформа *Intel IA-32E (Long mode)* и *System V x86_64 ABI*
- Существует возможность вызова платформы с помощью функции-шлюза (исполнение пользовательских команд и разрешение не прямых ссылок)

Виртуальный стек

(по состоянию на инстр. 3)

offset 2: [RSP+4]

- [data+1]

offset 1: [RSP+0]

- 20

offset 0: RAX

- 10

Байт-код

```
1. ld.i data+1
2. push.i 20
3. push.i 10
4. div.i
5. add.i
6. user_instr
```

Машинный код (x86_64)

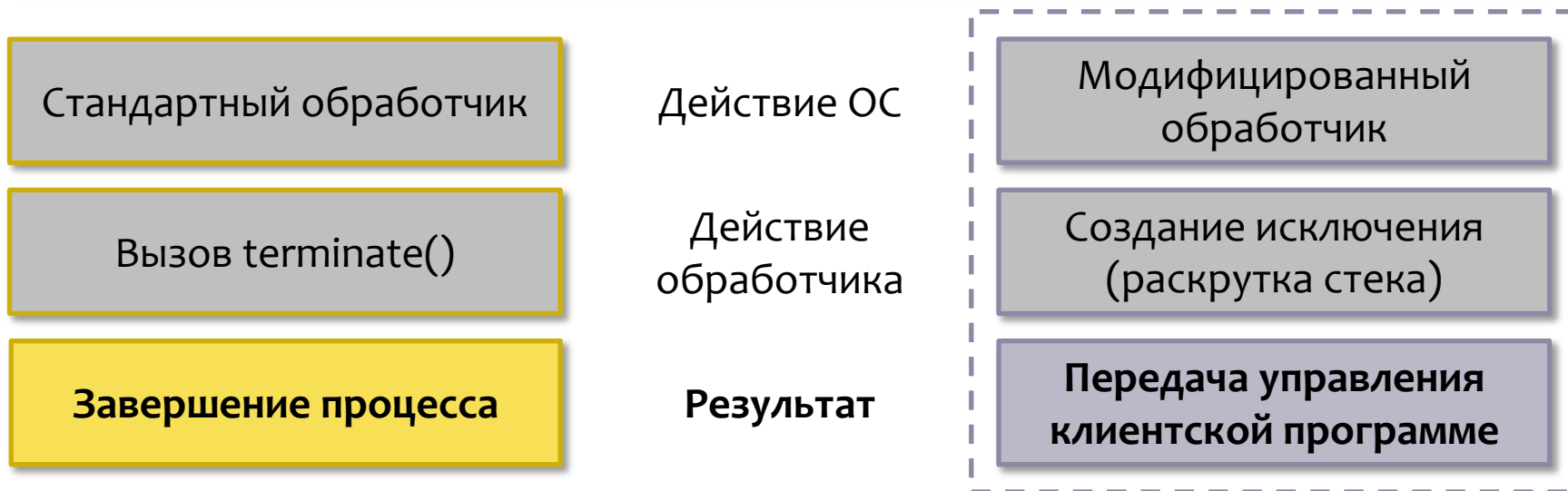
```
1. push [data+1]
2. push 20
3. mov rax, 10
4. cqo
   idiv [rsp]
   add rsp, 4
5. pop rdx
   add rax, rdx
6. lea rsi, [<inst. 6>]
   mov rdi, EXEC_INSTR
   call [platform_gate]
```

Подсистема обработки исключений

- Исполнение произвольного программного кода на центральном процессоре небезопасно
 - Возможны ошибки в коде или трансляторе

Сбой исполнения программы не должен приводить к аварийному завершению работы интерпретатора.

SIGSEGV (ошибка сегментации) — наиболее распространённый сбой



Результаты работы

Разработан комплекс средств для исполнения программного кода в стековой виртуальной машине с возможностью JIT-компиляции в машинный код CPU.

- Обеспечена возможность кроссплатформенной работы основных компонентов (кроме JIT-транслятора)
- Реализована абстракция от входного языка путём использования сериализуемого промежуточного представления (байт-кода)
- Поддерживается функционал компоновщика
- Разработана гибкая виртуальная машина со строгой типизацией и технической возможностью трассировки и управления исполнением
- Реализована JIT-трансляция байт-кода в машинный код Intel x86 для архитектуры Intel x86 с System V x86_64 ABI
- Реализована подсистема обработки исключительных ситуаций (сигналов)

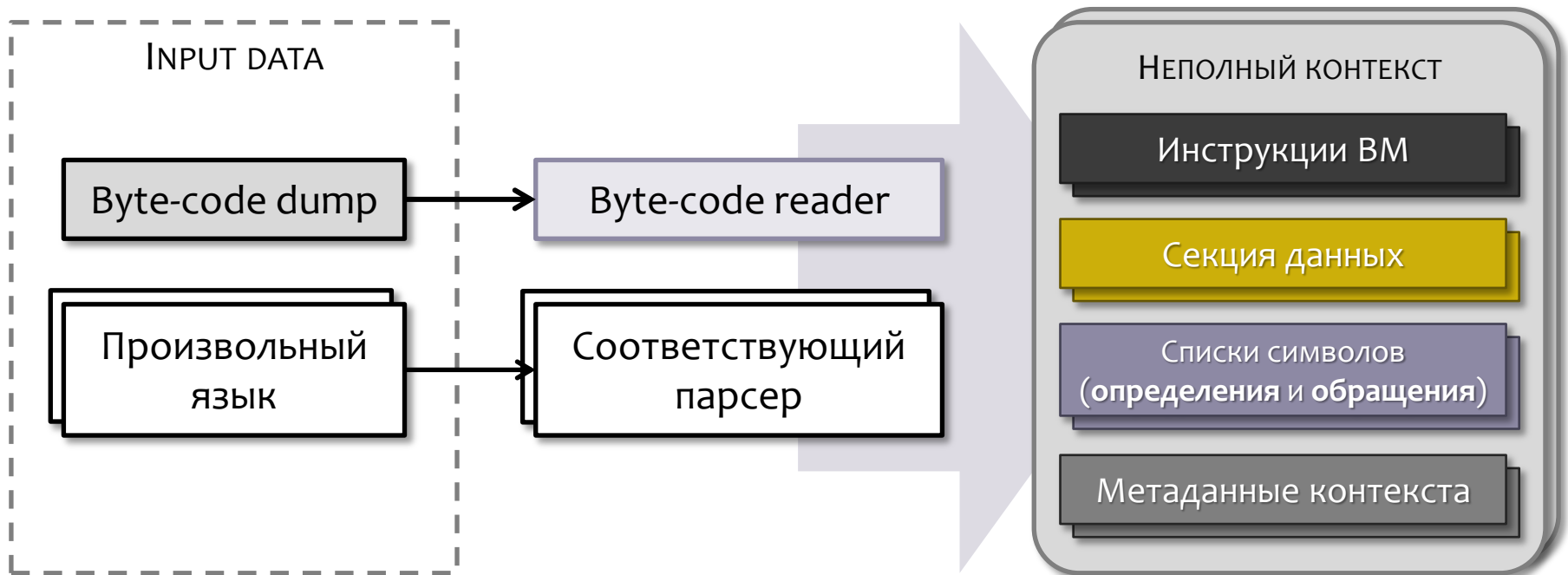
Объём кода:	9,5 KLOC
Количество файлов:	40 файлов
Адрес репозитория:	git://github.com/intelfx/Homework_2011.git

Благодарности

- Работа была выполнена в рамках учебного задания экспериментальной площадки по проектной деятельности в программировании И. Р. Дединского
- В процессе работы были использованы материалы Интернет-сайтов *stackoverflow.com* и *gotw.ca/gotw*
- При разработке JIT-компилятора была использована техническая документация для архитектуры Intel x86 (*Intel IA-32 Software Programming Guide*)

Front-end

- Абстракция от входного языка
- Используется принцип чёрного ящика
 - Отдельный модуль
 - Вход — файл
 - Выход — байт-код в виде «контекста» (объектный файл)



Объединение контекстов

- Шаг 1: слияние секций
- Шаг 2: распределение адресов символов (relocation)

КОНТЕКСТ 1

Инструкции ВМ (50)

Секция данных (20)

Список символов

(определение)
0x1A78 -> data:15

(определение)
0x1250 -> code:30

КОНТЕКСТ 2

Инструкции ВМ (20)

Секция данных (10)

Список символов

(определение)
0x6128 -> code:10

(определение)
0xA012 -> data:10

ИТОГОВЫЙ КОНТЕКСТ

Инструкции ВМ 1+2 (70)

Секция данных 1+2 (30)

Список символов

(определение)
0x1A78 -> data:15

(определение)
0x1250 -> code:30

(определение)
0x6128 -> **code:60**

(определение)
0xA012 -> **data:30**