



## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1 Анализ задания на ВКР .....	6
1.1 Анализ существующих аналогов.....	6
1.1.1 Intel 8080 CPU Emulator.....	6
1.1.2 Emu-I8080 .....	7
1.1.3 Rk86 .....	7
1.2 Разработка технического задания .....	8
1.2.1 Интерфейс программного эмулятора .....	9
1.2.2 Диаграмма вариантов использования и динамическая модель системы.....	11
1.3 Выводы по главе.....	24
2 Проектирование.....	25
2.1 Модули системы.....	25
2.2 Программный интерфейс приложения .....	29
2.3 Выводы по главе.....	30
3 Реализация .....	31
3.1 Используемые инструменты.....	31
3.2 Используемые шаблоны проектирования .....	31
3.2.1 Model-View-Presenter .....	31
3.2.2 Команда.....	32
3.2.3 Слушатель.....	33
3.3 Инструкции к программе .....	34
3.3.1 Инструкция по сборке .....	34
3.3.2 Инструкция по установке и запуску .....	34
3.3.3 Сборка API.....	34
3.3.4 Работа с API.....	34
3.4 Тестирование .....	36
3.4.1 Модульное тестирование .....	36
3.4.2 Ручное тестирование .....	36
3.5 Выводы по главе.....	38
ЗАКЛЮЧЕНИЕ .....	39
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	40

ПРИЛОЖЕНИЕ А .....	41
ПРИЛОЖЕНИЕ Б.....	42
ПРИЛОЖЕНИЕ В .....	43
ПРИЛОЖЕНИЕ Г.....	44

## ВВЕДЕНИЕ

При изучении дисциплин, связанных с микропроцессорными системами («ЭВМ и периферийные устройства», «Архитектура ЭВМ и систем»), первым этапом обучения студентов является процесс формирования понимания основных принципов функционирования микропроцессора. Студентам необходимо рассмотреть структуру микропроцессора, его функциональные блоки, механизмы их взаимодействия, а также его систему команд. Довольно часто в процессе изучения микропроцессоров используются их программные эмуляторы.

Несмотря на то, что на данный момент существует множество эмуляторов микропроцессора Intel 8080 [1, 2] каждый из них обладает какими-либо недостатками:

- отсутствуют внешние устройства, виртуально подключенные к микропроцессору (таймеры, дисплеи и т.д.);
- отсутствует переносимость между различными операционными системами;
- отсутствует возможность установки точек останова в программе;
- при разработке команды программы жестко привязываются к адресам в памяти (отсутствует транслятор из текста программы в машинный код), что снижает эффективность работы и затрудняет групповую работу студентов.

Отсутствие эмуляторов, удовлетворяющих всем потребностям преподаваемых дисциплин, подкрепляет **актуальность работы**.

**Целью** выпускной квалификационной работы является разработка программного эмулятора микропроцессора Intel 8080 не обладающего вышеперечисленными недостатками. Для достижения цели в работе решаются следующие **задачи**:

- 1) Составить техническое задание на разработку эмулятора, исходя из особенностей задач, решаемых студентами в рамках преподаваемых курсов.

- 2) Выполнить проектирование архитектуры программного эмулятора основываясь на составленном техническом задании.
- 3) Реализовать эмулятор, придерживаясь разработанных ранее архитектурных решений.
- 4) Опираясь на функциональные требования, провести тестирование разработанного программного эмулятора.
- 5) Составить инструкции по сборке и запуску программного эмулятора.

# 1 Анализ задания на ВКР

## 1.1 Анализ существующих аналогов

### 1.1.1 Intel 8080 CPU Emulator

Эмулятор микрокомпьютера на базе микропроцессора Intel 8080 под управлением операционной системы CP / M (Control Programs for Microcomputer) [1]. Эмулятор работает в браузере и позволяет монтировать до 4 образов восьмидюймовых дискет, с которых возможна загрузка заранее подготовленных программ в виде .hex файлов. Интерфейс эмулятора выполнен в виде консоли, как показано на рисунке 1. Управление осуществляется при помощи консольных команд. В эмуляторе есть возможность пошагового выполнения программы, просмотра/модификации значений регистров, флагов и ячеек памяти.

```
Intel 8080 CPU Emulator
2010 by Stefan Tramm
based on
  ShellInABox and its marvelous VT100 emulator by Markus Gutschke
  (http://code.google.com/p/shellinabox/)
  js8080 by Chris Double (http://www.bluishcoder.co.nz/js8080/)
  z80pack by Udo Munk (http://www.unix4fun.org/z80pack/)

Type HELP for a list of commands.

>>> help
r                               show listing of available files
r filename                       read Intel HEX file into memory
r d filename                     load disk image file into drive 0..3
r ptr:                           read Intel HEX from ptr: device into memory
r ptp:                           read Intel HEX from ptp: device into memory
d [address]                     dump memory
l [address]                     list memory
m [address]                     modify memory
g [address]                     run program
<CR>                             single step program
x [register]                   show/modify register (a b c d e h l af bc de hl sp pc)
x f<flag>                       modify flag (s z i h p c)
b [d [addr]]                   load bootsector from drive d to addr, default 0
io                              show current disc io, tape, and puncher parameters
rew                             rewind tape (mount tape by drag-n-drop)
ptp [name]                     send puncher content to a server or create new puncher (name)
dsk [d]                         show/mount disks or send disk 0..3 image to a server
z [n]                           show/modify clock cycle count
instr [n]                       show/modify instructions per JS timeslice
F d                             format disk drive 0..3
W                               wipe out diskdrives
Chrome users: enable popups, "dsk", "ptp", and line printer open a new window
Find more documentation at http://www.tramm.li/i8080/index.html
reach the author at mailto:stefan.tramm@me.com
>>>
```

Рисунок 1 – Интерфейс эмулятора Intel 8080 CPU Emulator

### 1.1.2 Emu-I8080

Эмулятор микропроцессора Intel 8080. Данный эмулятор имеет графический интерфейс пользователя (рисунок 2), что позволяет удобно просматривать/редактировать значения отдельных ячеек памяти, регистров и флагов. Присутствуют режимы полного и пошагового выполнения программы, реализована поддержка работы с виртуальным дисплеем разрешением 32x10 пикселей. Эмулятор предоставляет возможность для ввода значений с клавиатуры, а также вывод значений регистра-аккумулятора в соответствующую КОНСОЛЬ.

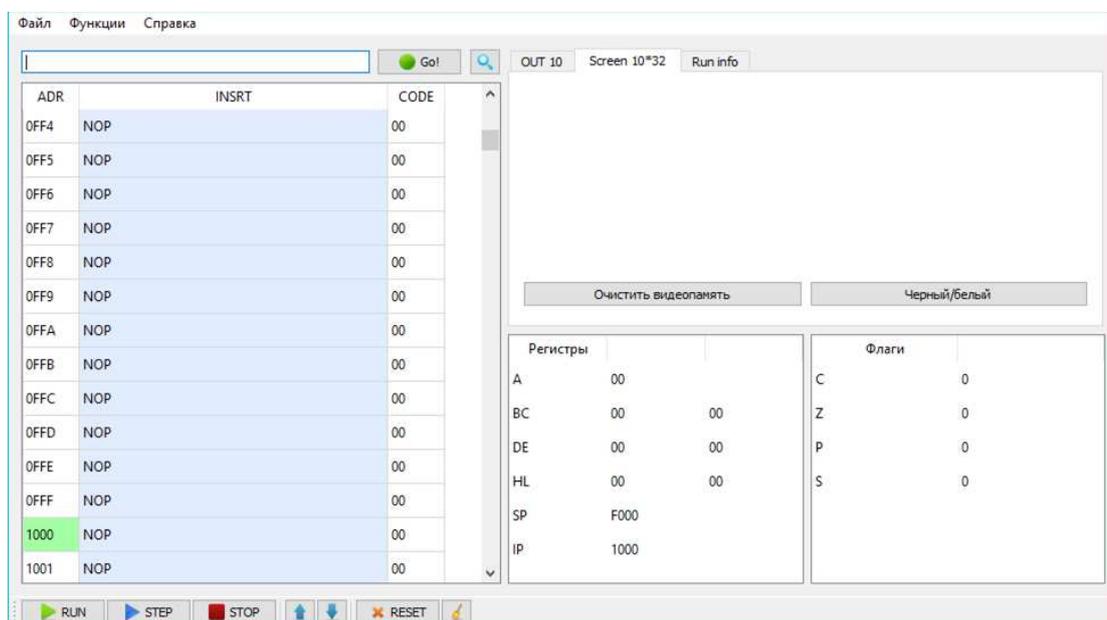


Рисунок 2 – Интерфейс эмулятора Emu-I8080

### 1.1.3 Rk86

Эмулятор 8-ми разрядного советского персонального компьютера Радио-86РК [2]. Данный эмулятор работает в браузере. Управление эмулятором происходит в режиме эмуляции работы монитора ПЗУ, что позволяет просматривать значения ячеек памяти, состояния регистров и флагов, а также вводить команды микропроцессора в виде шестнадцатеричных кодов. Данный

эмулятор позволяет писать программы на языке ассемблера и загружать транслированный машинный код в память. Однако, несмотря на наличие ассемблера, детальная эмуляция поведения компьютера Радио-86РК и его программного обеспечения делает процесс написания и отладки программ долгим и трудоёмким.



Рисунок 3 – Интерфейс эмулятора Rk86

## 1.2 Разработка технического задания

Функционал разрабатываемого программного эмулятора микропроцессора Intel 8080 формируется исходя из набора задач, стоящих перед студентами, в рамках курса лабораторных работ (таблица 1) по дисциплине ЭВМ и периферийные устройства.

Кроме того, часть лабораторных работ выполняется студентами в группах, а используемый в настоящее время эмулятор не удобен при групповой работе из-за отсутствия возможности использовать систему контроля версий, что связано со способом хранения кода программы в файле.

Таблица 1 – Список лабораторных работ по дисциплине ЭВМ и периферийные устройства

Номер	Наименование работы
1	Изучение арифметических и логических команд базового микропроцессора.
2	Изучение команд передачи управления в базовом микропроцессоре.
3	Изучение взаимодействия со стеком и вызов подпрограмм.
4	Моделирование процесса управления внешними устройствами.
5	Моделирование генерации случайной последовательности и оценка качества генератора.
6	Формирование динамического текста на виртуальном экране.
7	Моделирование терморегулятора.
8	Проектирование систем управления объектом. Аппаратная часть.
9	Проектирование систем управления объектом. Программная часть.

Таким образом в разрабатываемом эмуляторе должна быть реализована поддержка:

- работы с таймером;
- работы с пиксельным экраном;
- работы с символьным экраном;
- трансляции программы в машинный код.

### 1.2.1 Интерфейс программного эмулятора

На рисунках 4 и 5 приведен прототип интерфейса пользователя в соответствии с пожеланиями заказчика.

Главное окно эмулятора (рисунок 4) отображает значения ячеек памяти (1) и состояния регистров и флагов микропроцессора (2), поле ввода (3) и поле вывода (4), список подключенных периферийных устройств (5), а также пиксельный (6) и символьный (7) экраны.

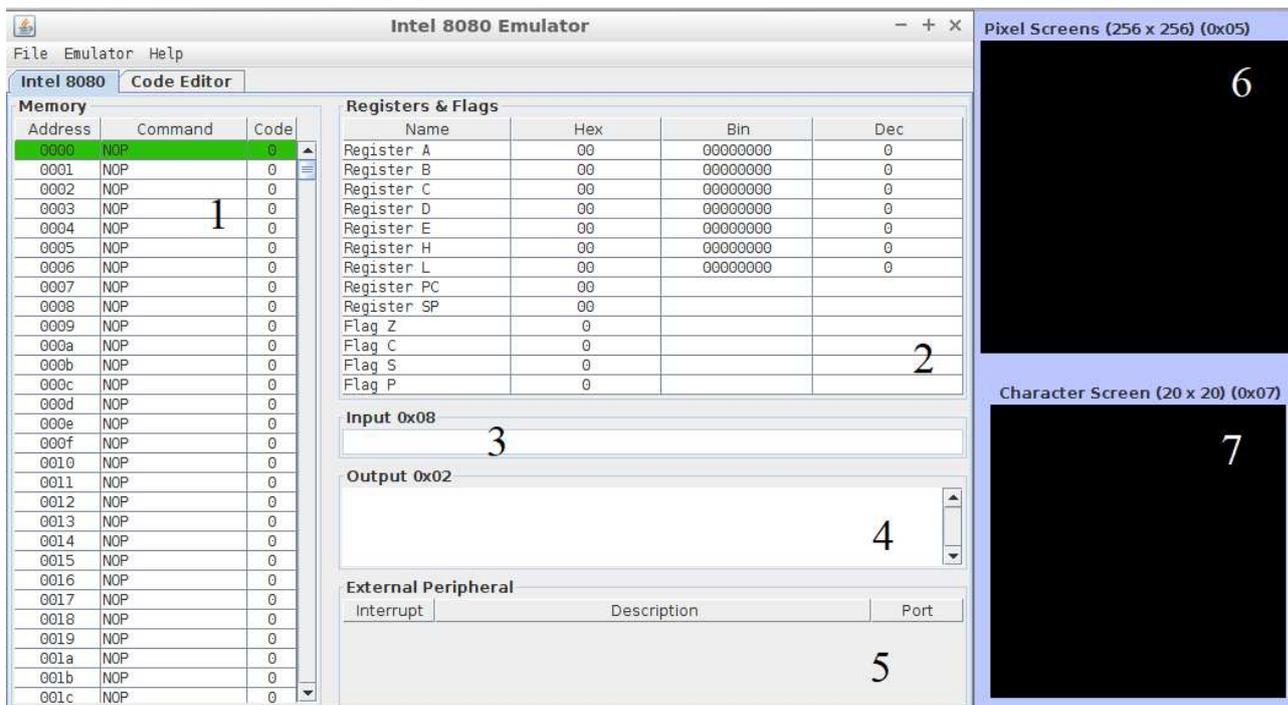


Рисунок 4 – Главное окно эмулятора

Вкладка редактора кода (рисунок 5) содержит поле для редактирования текста программы (1), поле для вывода результата трансляции программы (2) и таблицу соотношений меток и их адресов в памяти (3).

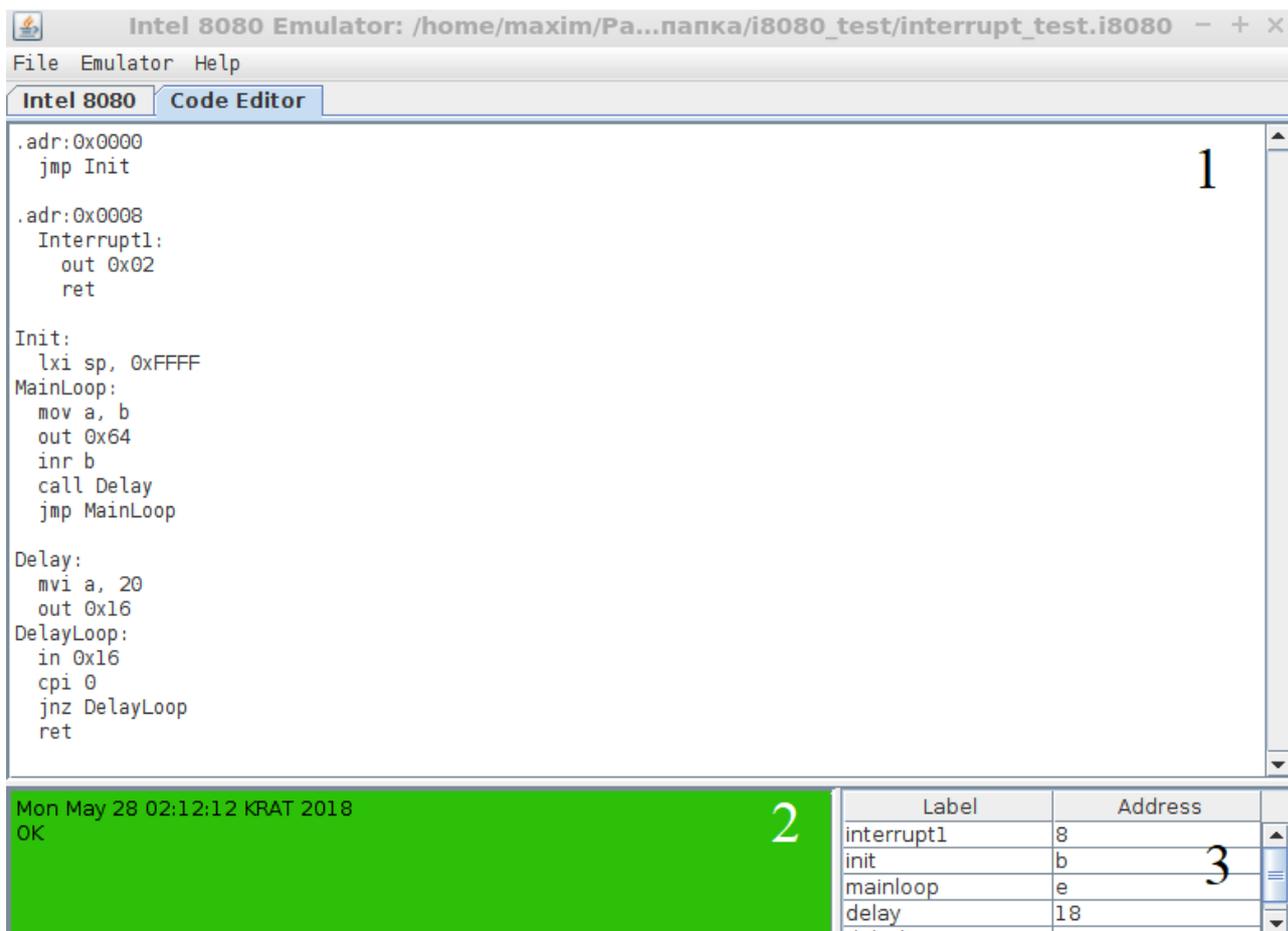


Рисунок 5 – Вкладка редактора кода

В ПРИЛОЖЕНИИ А приведена модель предметной области (словарь системы), в которой фиксированы основные термины (соответствующие элементам программы), используемые при описании технического задания.

### 1.2.2 Диаграмма вариантов использования и динамическая модель системы

Функциональные требования к новому эмулятору можно сформулировать в виде диаграммы вариантов использования [3], графическая часть которой приведена на рисунке 6.

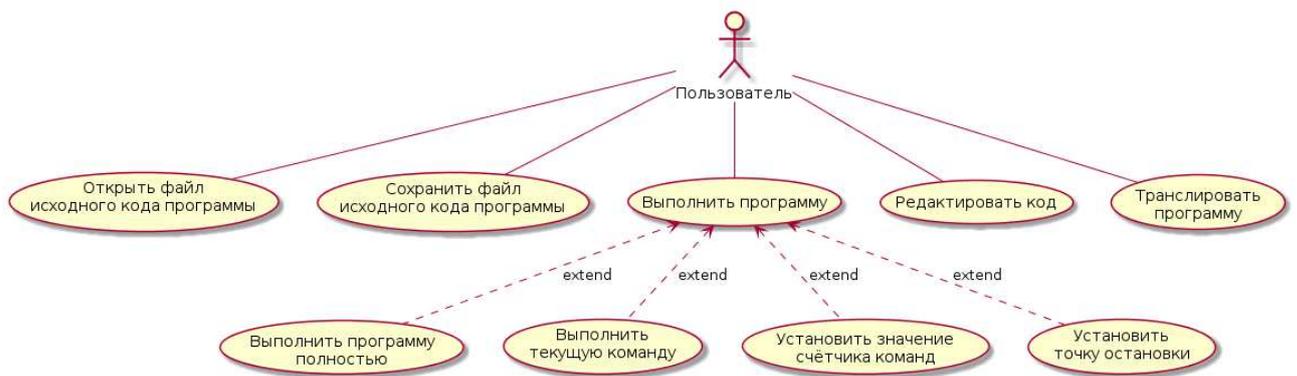


Рисунок 6 – Диаграмма вариантов использования

### Текстовое описание прецедентов:

**Название прецедента:** *открыть файл исходного кода программы.*

**Действующее лицо:** пользователь.

**Цель:** открыть файл программы.

**Предусловие:** процесс выполнения программы на эмуляторе остановлен.

### Главная последовательность:

- 1) Пользователь выбирает в меню «Файл» пункт «Открыть».
- 2) Эмулятор показывает пользователю окно выбора файла.
- 3) Пользователь выбирает нужный ему файл и нажимает кнопку «Открыть».
- 4) Эмулятор открывает файл и отображает его содержимое в окне редактора кода.

**Альтернативная последовательность** (попытка открытия несуществующего файла):

- 1) Пользователь выбирает в меню «Файл» пункт «Открыть».
- 2) Эмулятор показывает пользователю окно выбора файла.
- 3) Пользователь выбирает нужный ему файл и нажимает кнопку «Открыть».
- 4) Эмулятор закрывает окно выбора файла и показывает пользователю сообщение об ошибке открытия файла.

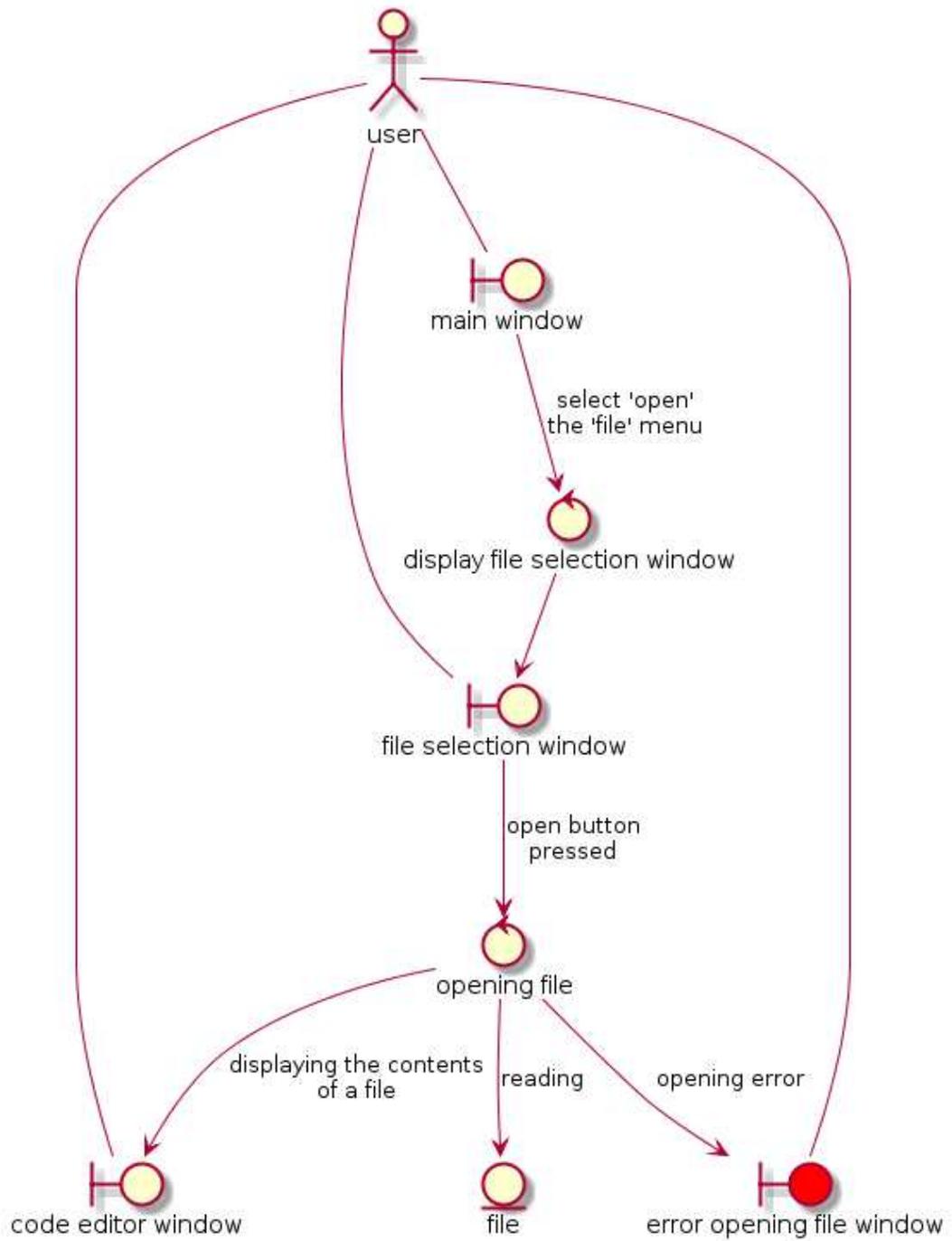


Рисунок 7 – Диаграмма пригодности прецедента «открыть файл исходного кода программы»

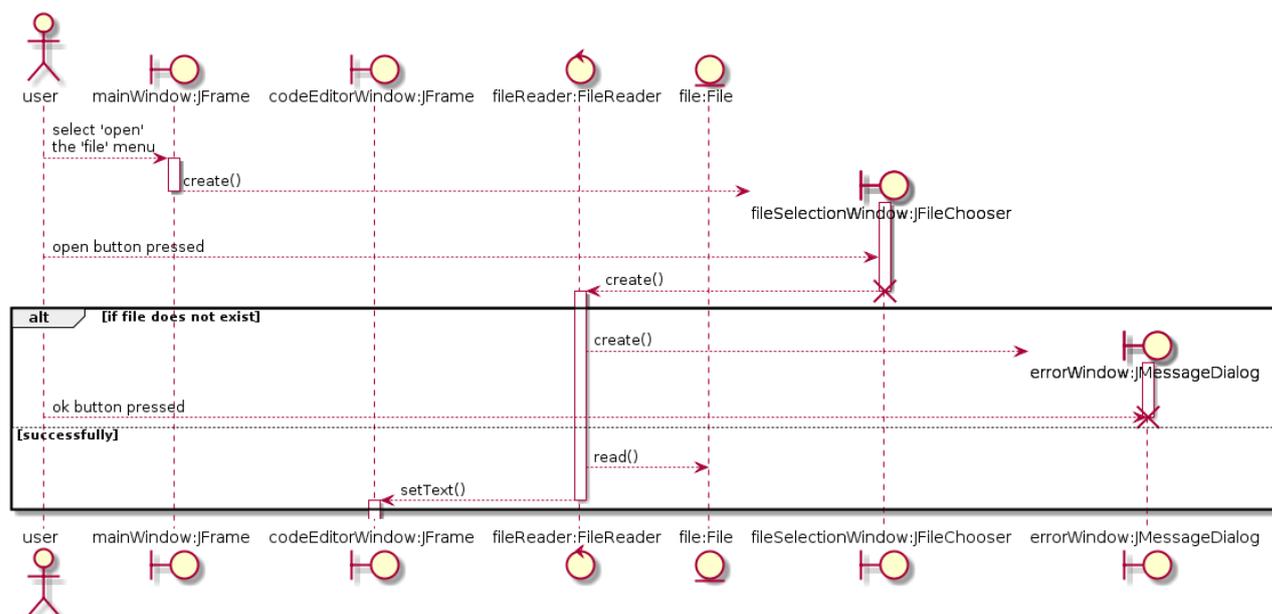


Рисунок 8 – Диаграмма последовательности прецедента «открыть файл исходного кода программы»

**Название прецедента:** *сохранить файл исходного кода программы.*

**Действующее лицо:** пользователь.

**Цель:** сохранить файл программы.

**Предусловие:** процесс выполнения программы на эмуляторе остановлен.

**Главная последовательность:**

- 1) Пользователь выбирает в меню «Файл» пункт «Сохранить».
- 2) Эмулятор показывает пользователю окно выбора файла.
- 3) Пользователь выбирает каталог назначения, вводит имя сохраняемого файла или выбирает существующий файл для сохранения и нажимает кнопку «Сохранить».
- 4) Эмулятор сохраняет файл и отображает сообщение об успешном сохранении файла.

**Альтернативная последовательность** (попытка сохранения файла в каталог без соответствующих прав доступа):

- 1) Пользователь выбирает в меню «Файл» пункт «Сохранить».
- 2) Эмулятор показывает пользователю окно выбора файла.

3) Пользователь выбирает каталог назначения, вводит имя сохраняемого файла или выбирает существующий файл для сохранения и нажимает кнопку «Сохранить».

4) Эмулятор закрывает окно выбора файла и показывает пользователю сообщение об ошибке сохранения файла.

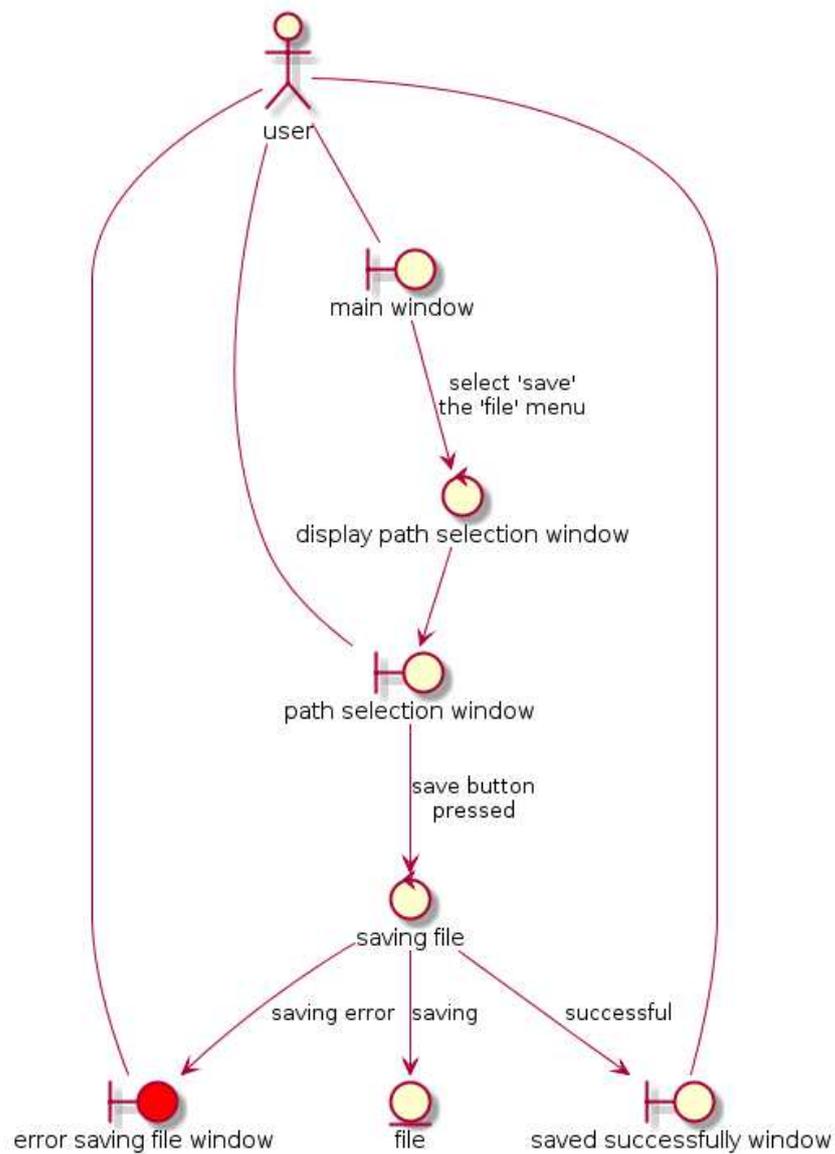


Рисунок 9 – Диаграмма пригодности прецедента «сохранить файл исходного кода программы»

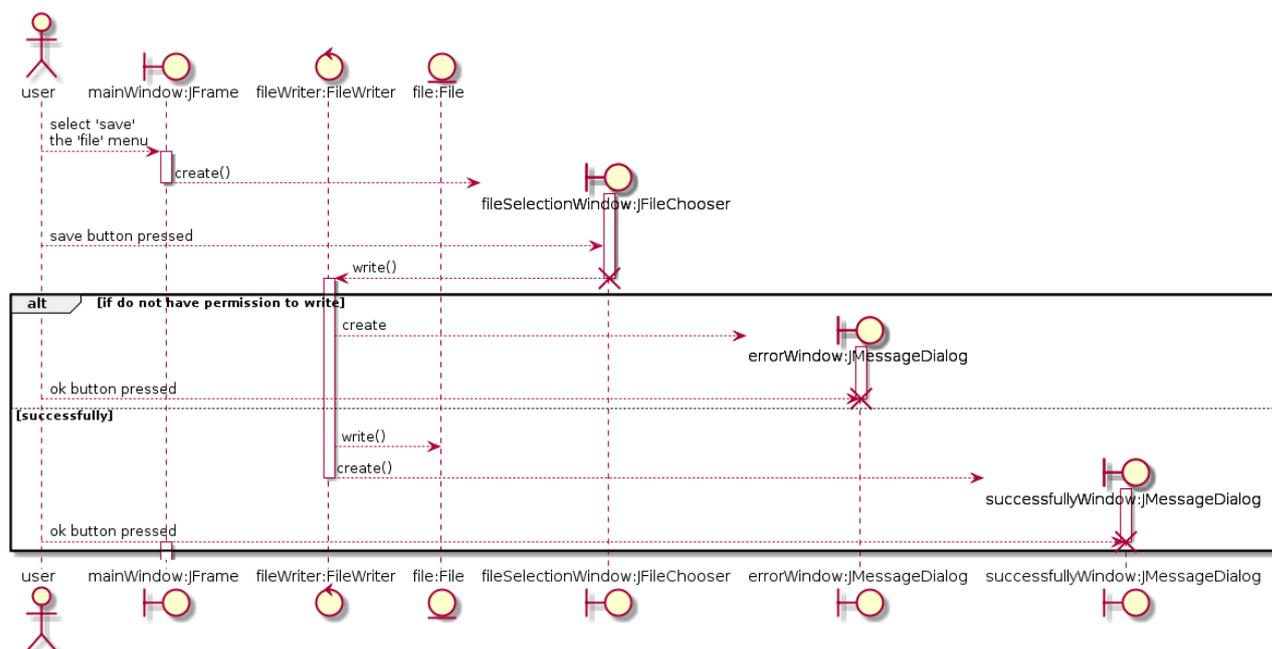


Рисунок 10 – Диаграмма последовательности прецедента «сохранить файл исходного кода программы»

**Название прецедента:** *транслировать программу.*

**Действующее лицо:** пользователь.

**Цель:** транслировать программу и загрузить её в память эмулятора.

**Предусловие:** процесс выполнения программы на эмуляторе остановлен.

**Главная последовательность:**

- 1) Пользователь выбирает в меню «Эмулятор» пункт «Трансляция».
- 2) Эмулятор проводит синтаксический анализ текста программы, выполняет трансляцию исходного текста программы в машинный код, загружает её в память эмулятора и выводит сообщение об успешной трансляции программы в консоль редактора кода.

**Альтернативная последовательность** (попытка трансляции исходного текста программы, содержащего синтаксические ошибки):

- 1) Пользователь выбирает в меню «Эмулятор» пункт «Трансляция».
- 2) Эмулятор проводит синтаксический анализ текста программы и выводит сообщение об ошибках в тексте программы в консоль редактора кода.

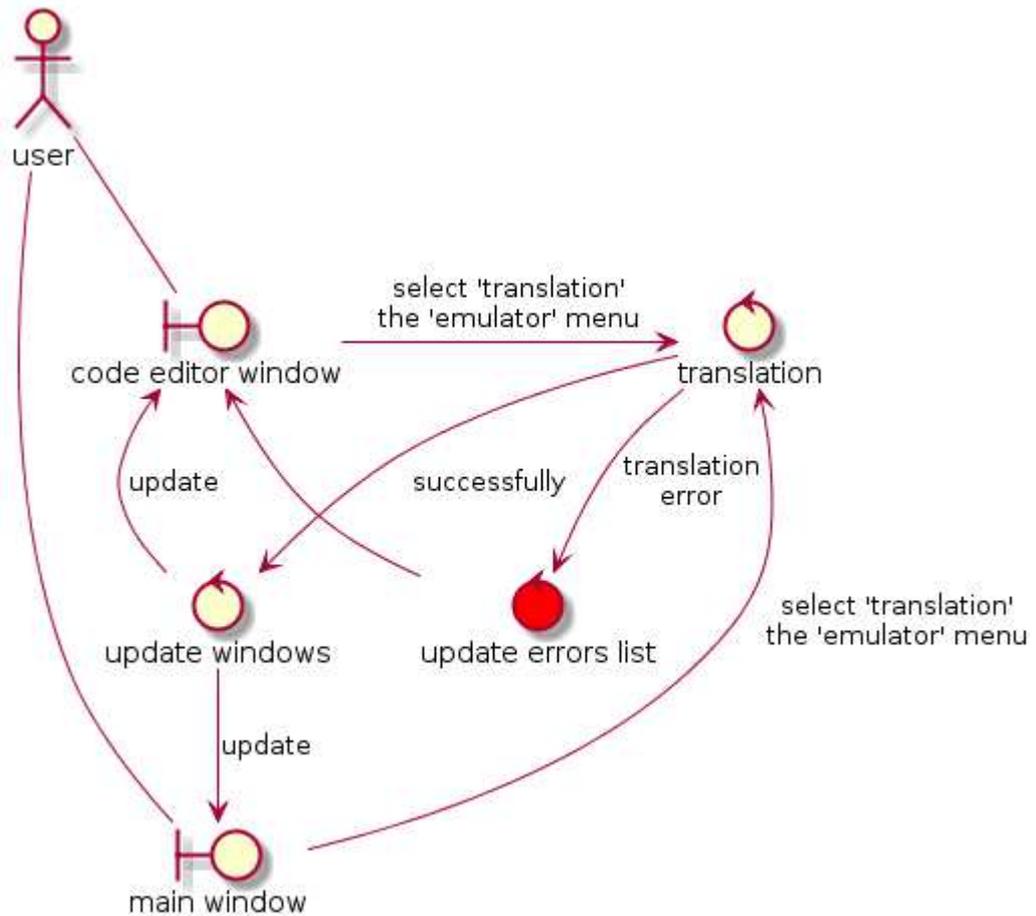


Рисунок 11 – Диаграмма пригодности прецедента «транслировать программу»

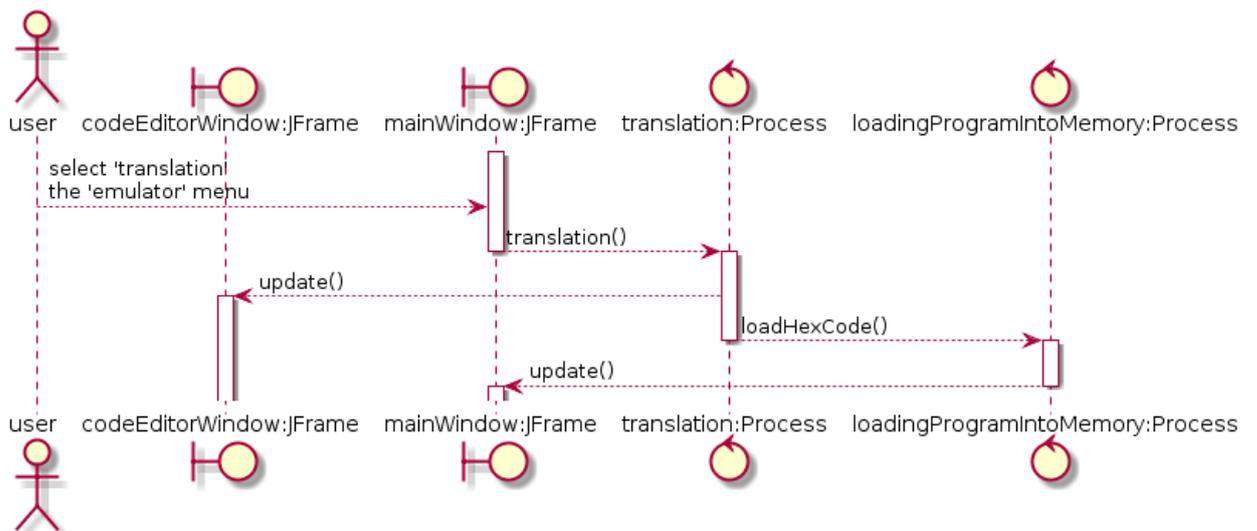


Рисунок 12 – Диаграмма последовательности для прецедента «транслировать программу»

**Название прецедента:** *выполнить программу.*

**Действующее лицо:** пользователь.

**Цель:** выполнить (частично или полностью) программу, находящуюся в памяти эмулятора.

**Предусловие:** процесс выполнения программы на эмуляторе остановлен, пользователь находится на главном экране эмулятора.

**Главная последовательность:**

1) Если пользователь хочет установить адрес, с которого необходимо выполнять программу, то выполняет прецедент «установить значение программного счётчика».

2) Если пользователь хочет добавить точку остановки, то выполняет прецедент «установить точку остановки».

3) Если пользователь хочет выполнить текущую команду микропроцессора, то выполняет прецедент «выполнить текущую команду».

4) Если пользователь хочет выполнить программу полностью, то выполняет прецедент «выполнить программу полностью».

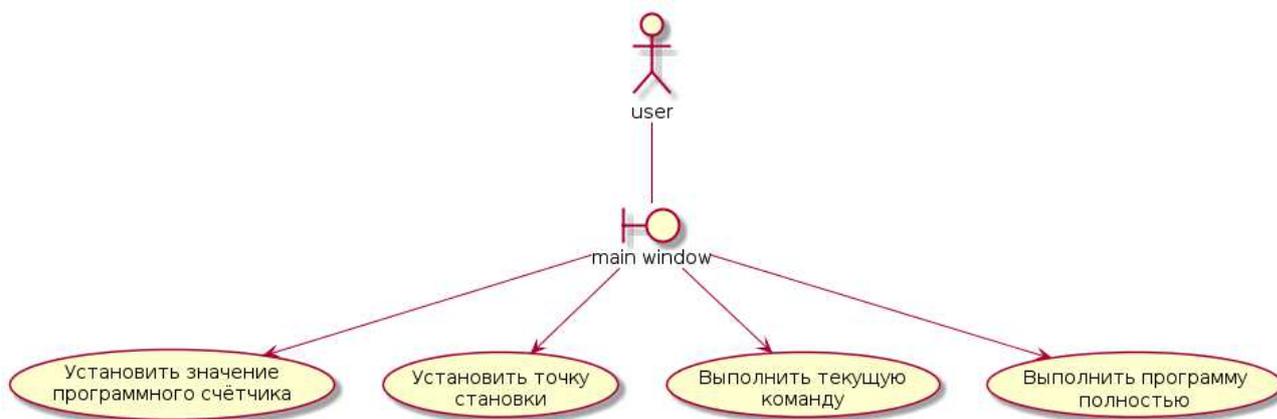


Рисунок 13 – Диаграмма пригодности прецедента «выполнить программу»

**Название прецедента:** *выполнить программу полностью.*

**Действующее лицо:** пользователь.

**Цель:** выполнить программу, находящуюся в памяти эмулятора.

**Предусловие:** процесс выполнения программы на эмуляторе остановлен, пользователь находится на главном экране эмулятора.

**Главная последовательность:**

1) Пользователь выбирает в меню «Эмулятор» пункт «Выполнить».

2) Эмулятор запускает процесс выполнения программы. Во время выполнения программы происходит обновление пиксельного и символьного экранов эмулятора. По достижению точки останова или конца программы, эмулятор завершает выполнение программы и отображает обновлённые значения регистров и памяти.

3) Если запущенная эмулятором программа содержит в себе бесконечный цикл, то её завершение выполняется пользователем принудительно выбором в меню «Эмулятор» пункта «Остановить».

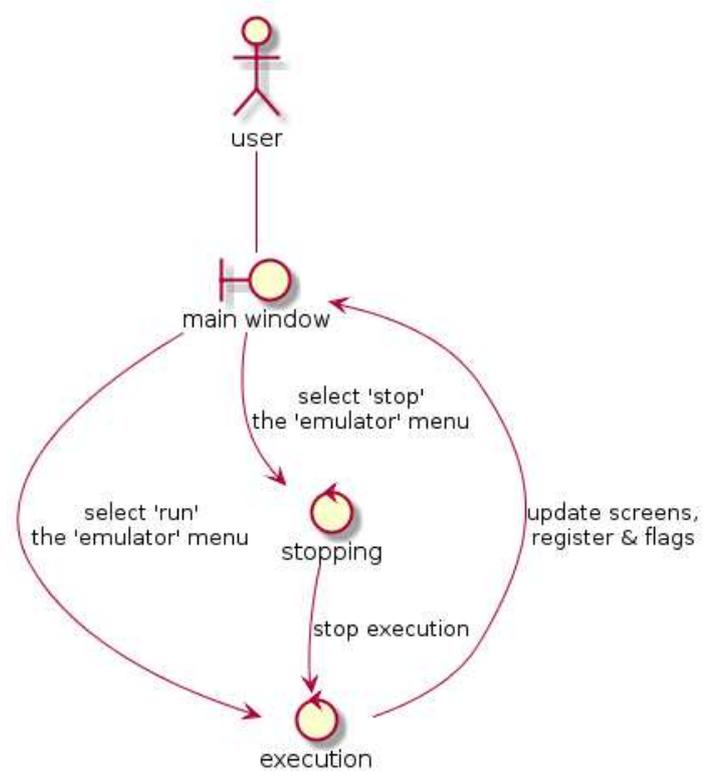


Рисунок 14 – Диаграмма пригодности прецедента «выполнить программу полностью»

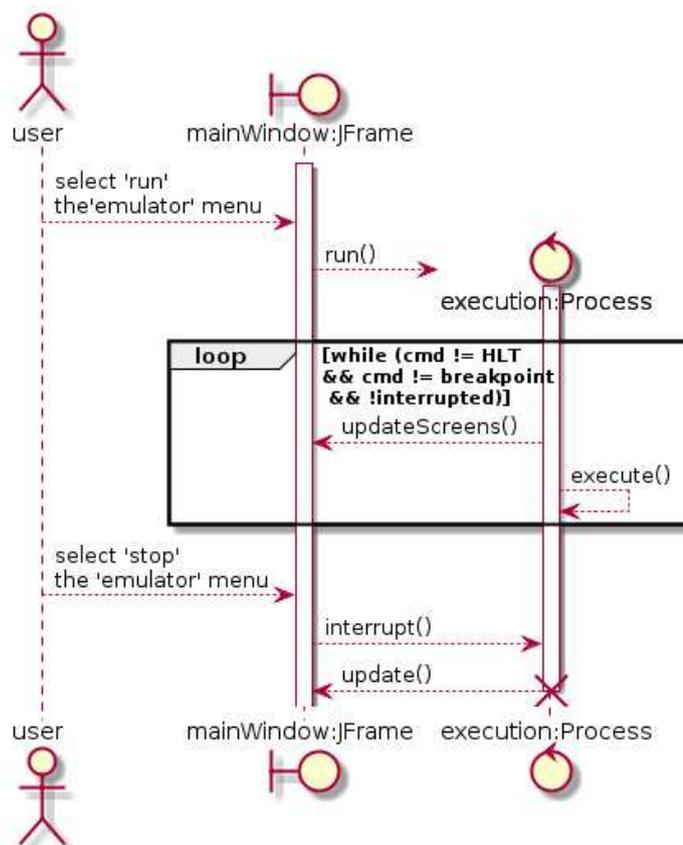


Рисунок 15 – Диаграмма последовательности для прецедента «выполнить программу полностью»

**Название прецедента:** *выполнить текущую команду.*

**Действующее лицо:** пользователь.

**Цель:** выполнить текущую команду микропроцессора.

**Предусловие:** процесс выполнения программы на эмуляторе остановлен, пользователь находится на главном экране эмулятора.

**Главная последовательность:**

- 1) Пользователь выбирает в меню «Эмулятор» пункт «Шаг».
- 2) Эмулятор выполняет текущую команду микропроцессора. Отображает обновлённые значения регистров и памяти, производит обновление пиксельного и символического экранов.

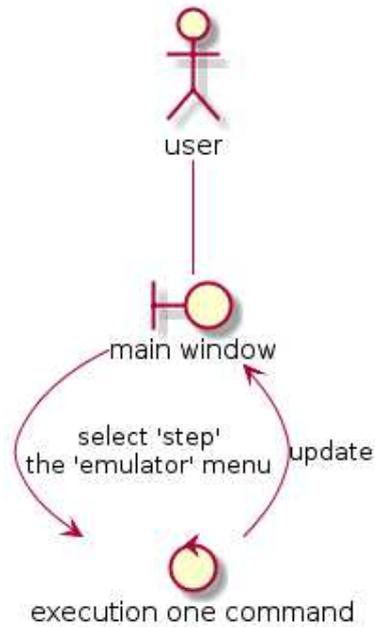


Рисунок 16 – Диаграмма пригодности прецедента «выполнить текущую команду»

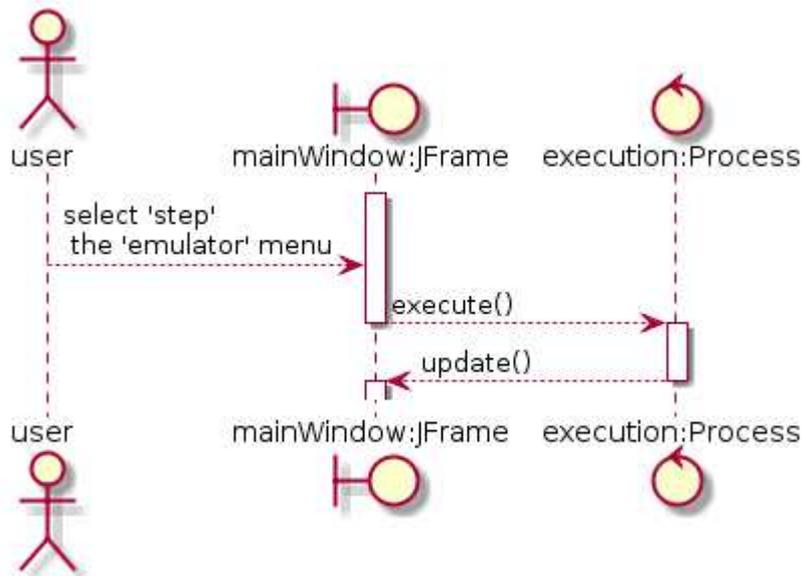


Рисунок 17 – Диаграмма последовательности для прецедента «выполнить текущую команду»

**Название прецедента:** *установить значение счётчика команд.*

**Действующее лицо:** пользователь.

**Цель:** установить нужное значение счётчика команд.

**Предусловие:** процесс выполнения программы на эмуляторе остановлен, пользователь находится на главном экране эмулятора.

### Главная последовательность:

1) Пользователь кликает левой кнопкой мыши по нужной строке таблицы памяти эмулятора.

2) Эмулятор устанавливает значение счётчика команд, согласно выбранному адресу памяти.

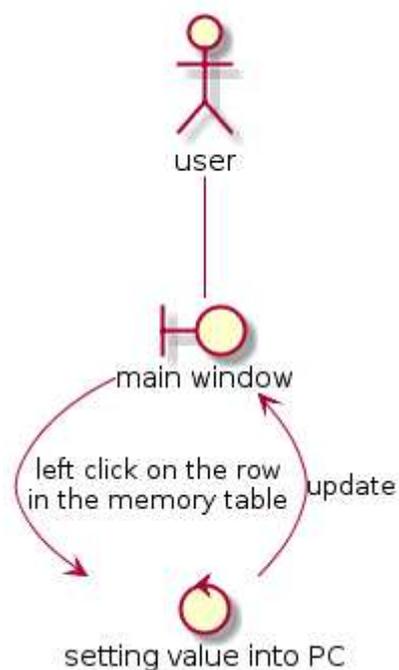


Рисунок 18 – Диаграмма пригодности прецедента «установить значение счётчика команд»

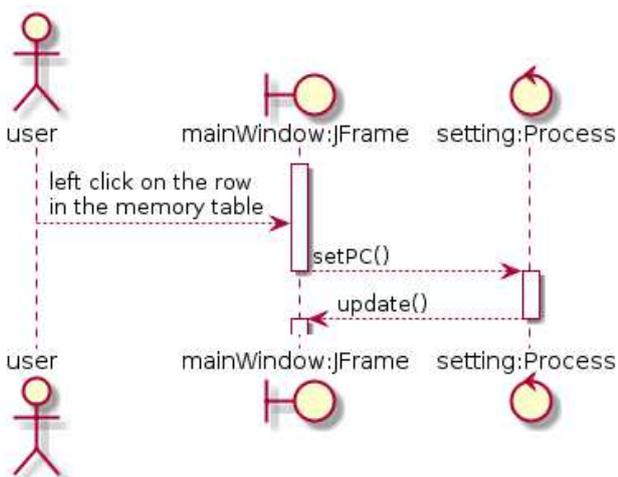


Рисунок 19 – Диаграмма последовательности для прецедента «установить значение счётчика команд»

**Название прецедента:** *установить точку останова.*

**Действующее лицо:** пользователь.

**Цель:** установить точку останова в программе.

**Предусловие:** процесс выполнения программы на эмуляторе остановлен, пользователь находится на главном экране эмулятора.

**Главная последовательность:**

1) Пользователь кликает правой кнопкой мыши по нужной строке таблицы памяти эмулятора.

2) Эмулятор устанавливает на выбранный адрес памяти точку останова.

**Альтернативная последовательность (удаление точки останова):**

1) Пользователь кликает правой кнопкой мыши по строке таблицы памяти эмулятора, содержащей точку останова.

2) Эмулятор снимает с выбранного адреса памяти точку останова.

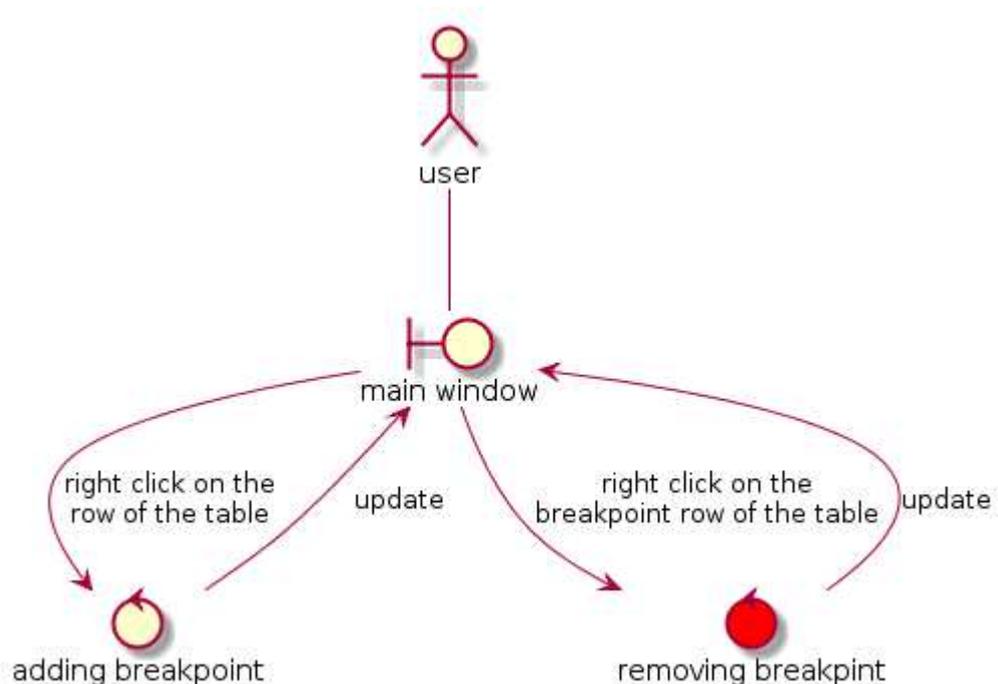


Рисунок 20 – Диаграмма пригодности прецедента «установить точку останова»

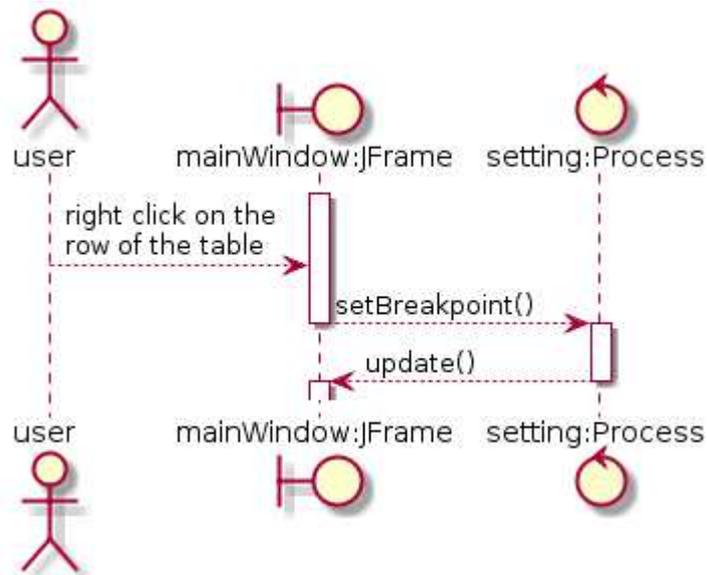


Рисунок 21 – Диаграмма последовательности для прецедента «установить точку останова»

### 1.3 Выводы по главе

- 1) Выполнен обзор предметной области и проведён анализ существующих аналогов разрабатываемого эмулятора.
- 2) Составлена модель предметной области.
- 3) Сформулированы функциональные требования к разрабатываемому эмулятору в виде диаграмм прецедентов, и их текстового описания. По каждому прецеденту составлены диаграммы пригодности и диаграммы последовательности.

## 2 Проектирование

### 2.1 Модули системы

Логику функционирования программного эмулятора микропроцессора Intel 8080 целесообразно разделить на два модуля:

- пользовательский интерфейс и презентационная логика;
- модель эмулятора (бизнес-логика).

Модель эмулятора содержит:

- транслятор;
- модуль исполнения пользовательской программы;
- модель микропроцессора;
- система команд микропроцессора;
- система ввода-вывода.

На рисунке 22 представлена схема взаимодействия модулей системы.

На рисунке 23 представлена сокращённая диаграмма классов эмулятора.

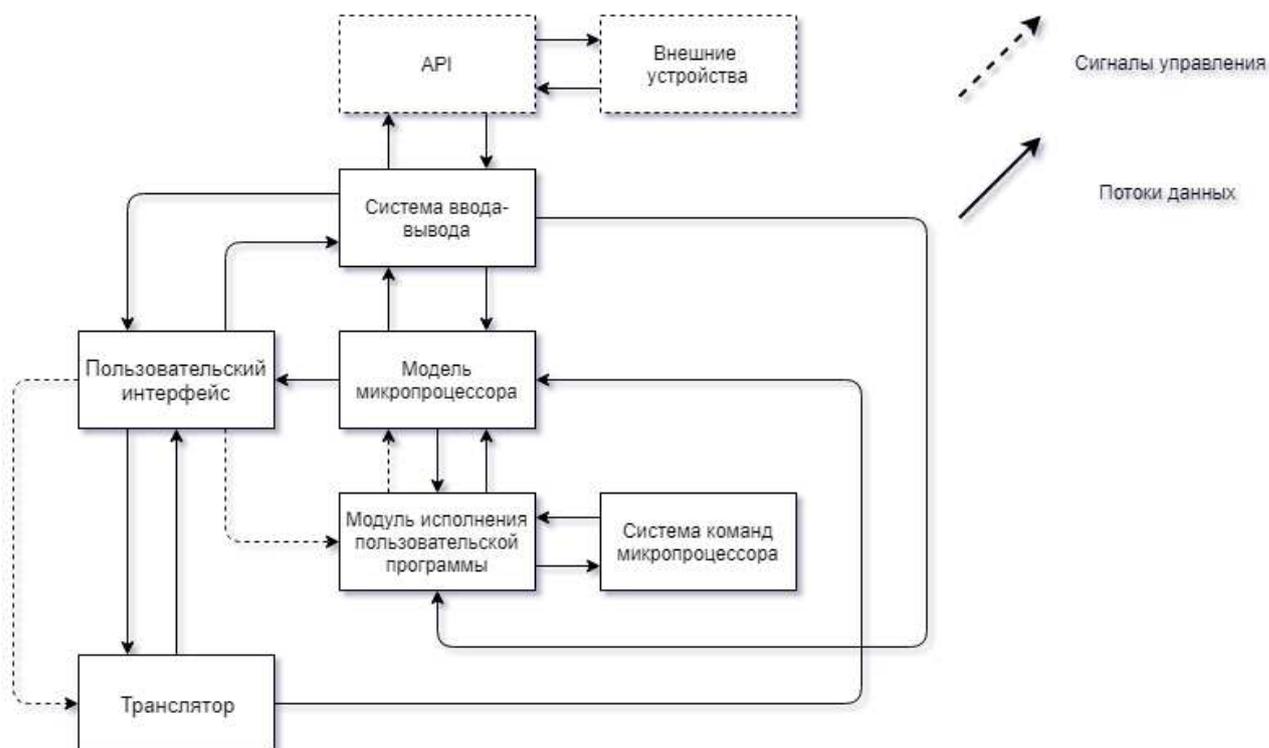


Рисунок 22 – Схема взаимодействия модулей системы

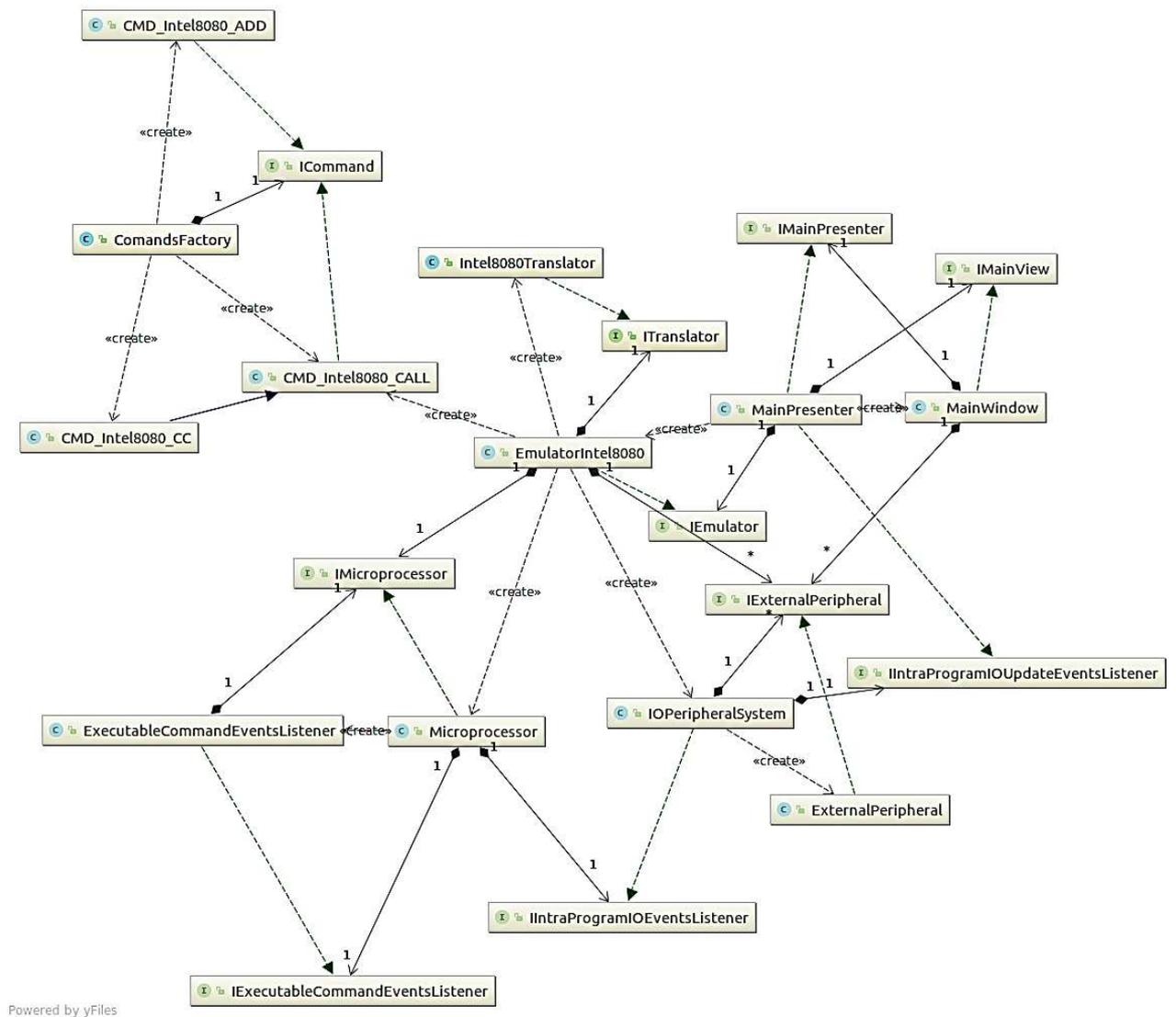


Рисунок 23 – Диаграмма основных классов эмулятора

На вход транслятора поступает текст пользовательской программы. Транслятор выполняет лексический анализ текста программы и, если в тексте программы отсутствуют не распознанные последовательности символов, то результатом работы транслятора является список лексем и шестнадцатеричных кодов команд микропроцессора Intel 8080.

Модуль исполнения пользовательской программы содержит в себе логику работы основных механизмов эмулятора:

- пошаговое выполнение программы;
- потоковое выполнение программы;
- точки останова;

- контроль прерываний.

Модуль микропроцессора описывает внутреннюю структуру микропроцессора и содержит в себе описание регистров, флагов и адресуемой микропроцессором памяти.

Система команд микропроцессора (рисунок 24) описывает набор доступных для выполнения команд и логику их исполнения.

Интерфейс *ICommand* определяет набор доступных для взаимодействия с командой методов.

Классы *CMD\_Intel8080\_ADD*, *CMD\_Intel8080\_CALL* и *CMD\_Intel8080\_CC* являются примером классов конкретных команд и содержат в себе информацию о конкретной инструкции микропроцессора.

Класс *ComandsFactory* представляет интерфейс для построения команды по её шестнадцатеричному коду, хранящемуся в памяти эмулятора.

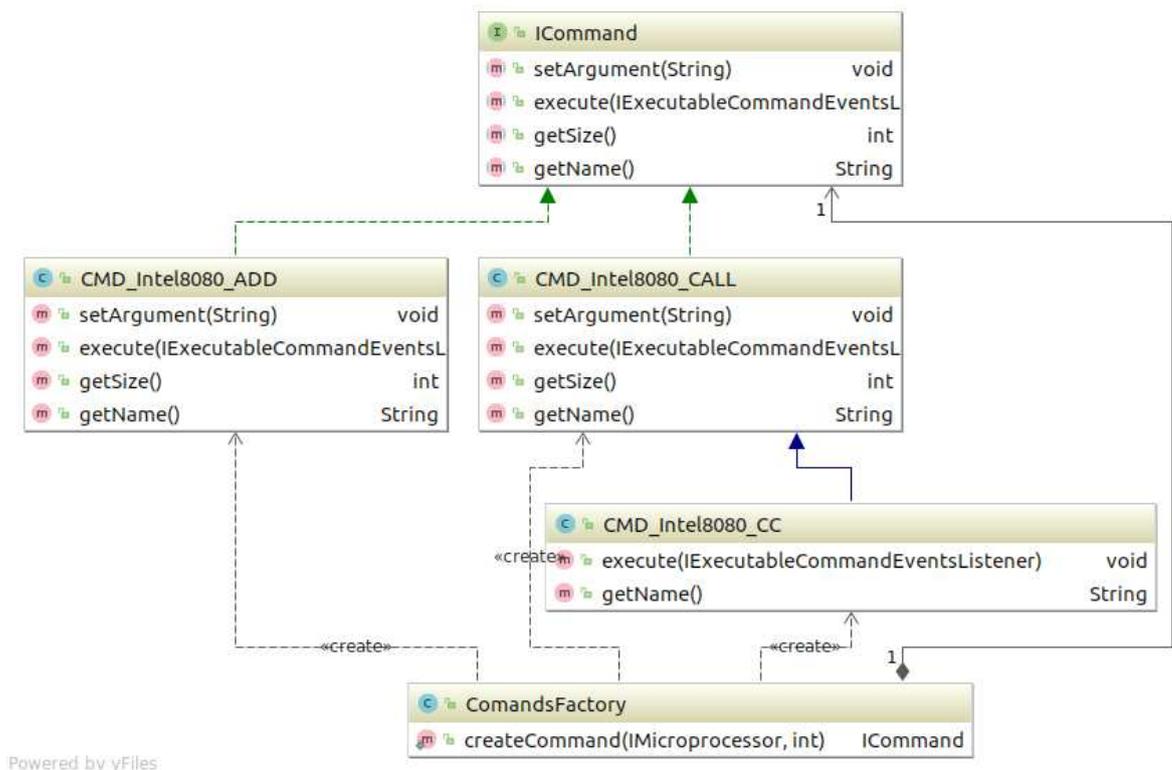


Рисунок 24 – Диаграмма классов системы команд

Система ввода-вывода (рисунок 25) содержит в себе логику работы портов микропроцессора и осуществляет информационный обмен микропроцессора с периферийными устройствами и регистрирует сигналы прерываний. Данная система также содержит в себе сервер внешних периферийных устройств, который отвечает за взаимодействие с периферийными устройствами, подключенными через API.

Интерфейс *IExternalPeripheral* описывает набор методов для взаимодействия с моделью внешнего периферийного устройства.

Класс *ExternalPeripheral* содержит в себе логику взаимодействия микропроцессора с периферийным устройством.

Класс *IOPeripheralSystem* является диспетчером периферийных устройств и отвечает за обслуживание подключений периферийных устройств.

Интерфейс *IntraProgramIOEventsListener* описывает методы для работы микропроцессора с системой ввода вывода. Класс, реализующий данный интерфейс прослушивает события ввода-вывода, возникающие при выполнении пользовательской программы.

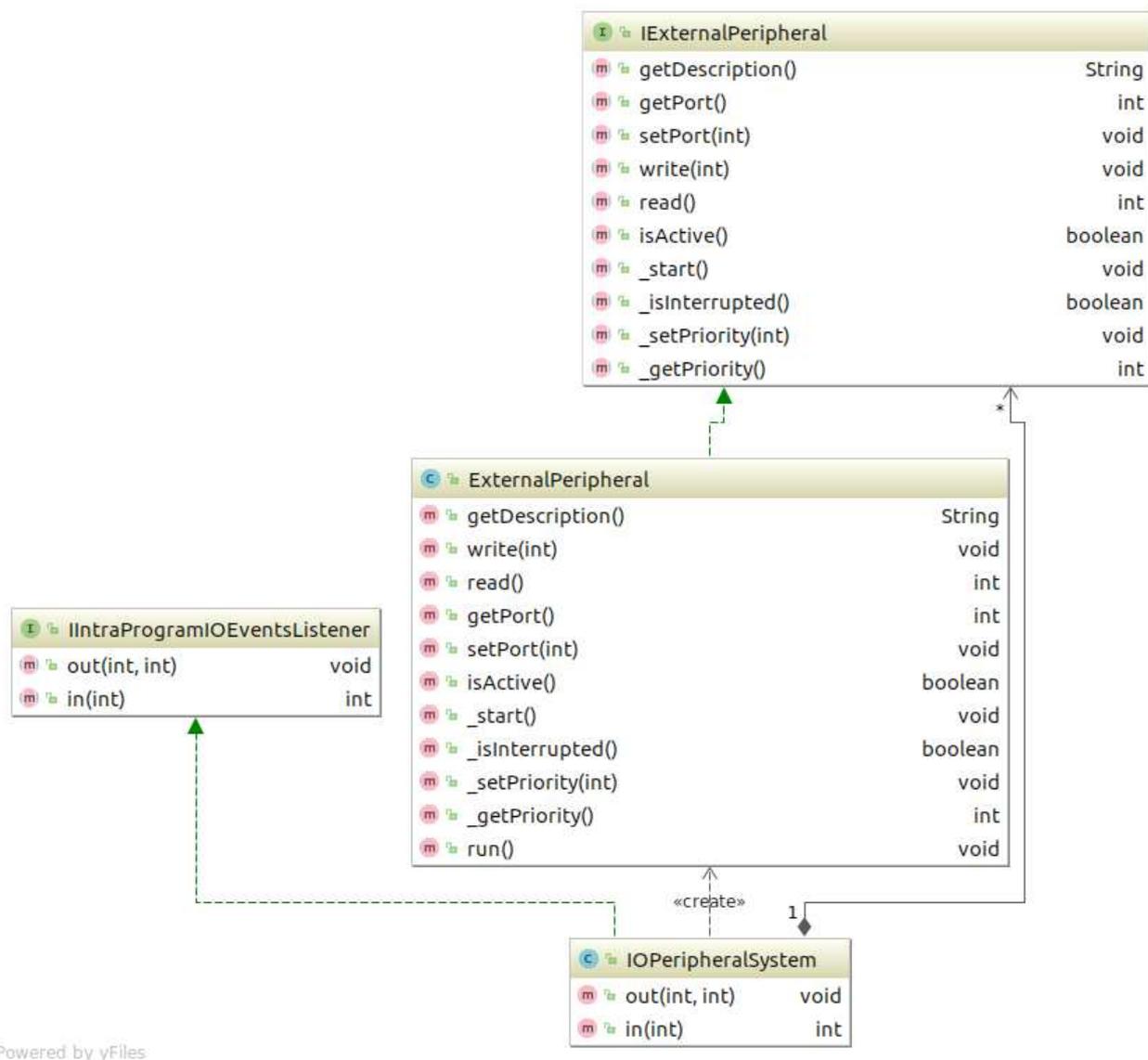


Рисунок 25 – Диаграмма классов системы ввода-вывода

## 2.2 Программный интерфейс приложения

Для расширения списка поддерживаемых эмулятором периферийных устройств было принято решение реализовать возможность разработки и подключения периферийных устройств, реализованных сторонними разработчиками.

API эмулятора (рисунок 26) предоставляет следующие функциональные возможности:

- подключение периферийного устройства к эмулятору;

- информационный обмен через 8-битный порт эмулятора;
- отправка сигнала прерывания.

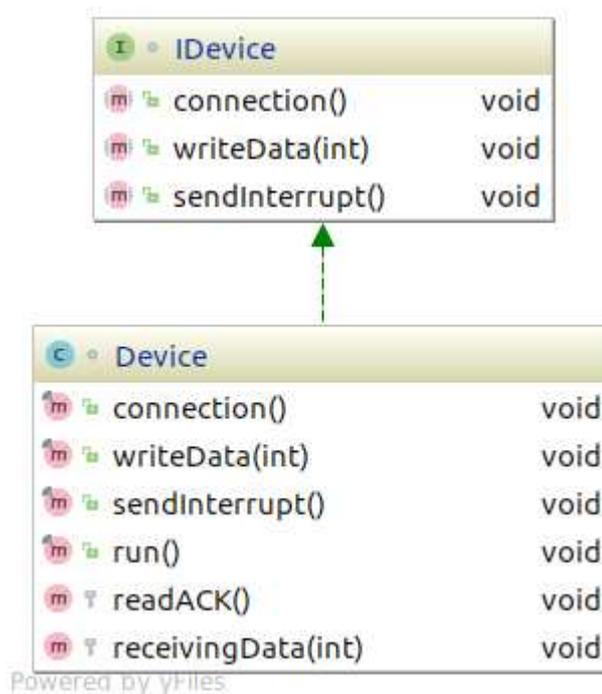


Рисунок 26 – Диаграмма классов API

### 2.3 Выводы по главе

- 1) Произведена декомпозиция логики работы эмулятора.
- 2) Определены обязанности каждого функционального модуля.
- 3) Для каждого функционального модуля составлена диаграмма классов.
- 4) Выполнено проектирование API для подключения периферийных устройств.

## 3 Реализация

### 3.1 Используемые инструменты

Java 8 – JDK 1.8.0\_171 [4].

В качестве среды разработки использовалась JetBrains IntelliJ IDEA Ultimate 2017.1 [5].

Для построения диаграмм на этапе разработки технического задания использовалась web версия PlantUML [6].

Система контроля версий – веб-сервис Github [7].

### 3.2 Используемые шаблоны проектирования

#### 3.2.1 Model-View-Presenter

Для разделения ответственности в презентационной логике приложения использовался паттерн MVP (Model-View-Presenter).

Диаграмма классов, которые непосредственно относятся к реализации шаблона MVP, представлена на рисунке 27.

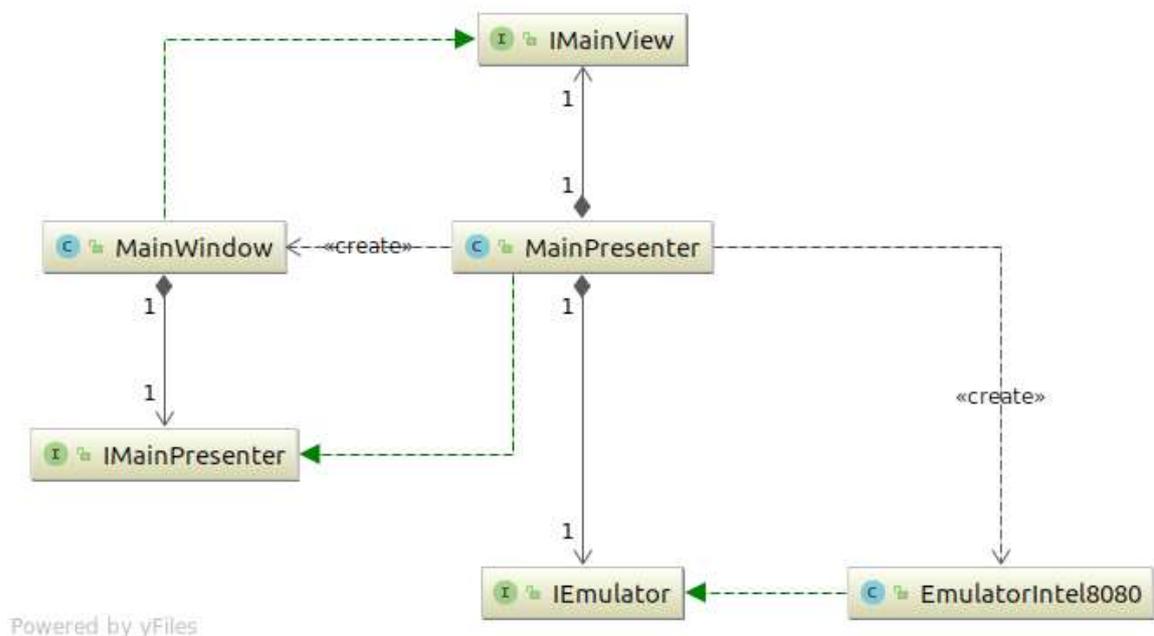


Рисунок 27 – Структура классов паттерна «MVP» в приложении

Вся бизнес-логика приложения заключена в модели системы – классе *EmulatorInte8080*. Класс *MainPresenter* производит контроль состояния пользовательского интерфейса – класс *MainWindow*, и осуществляет взаимодействие с моделью системы.

### 3.2.2 Команда

При реализации системы команд микропроцессора использовался паттерн «Команда» (рисунок 28). Каждая отдельная инструкция микропроцессора представлена в коде программы отдельным классом, инкапсулирующим внутри себя логику выполнения конкретной команды.

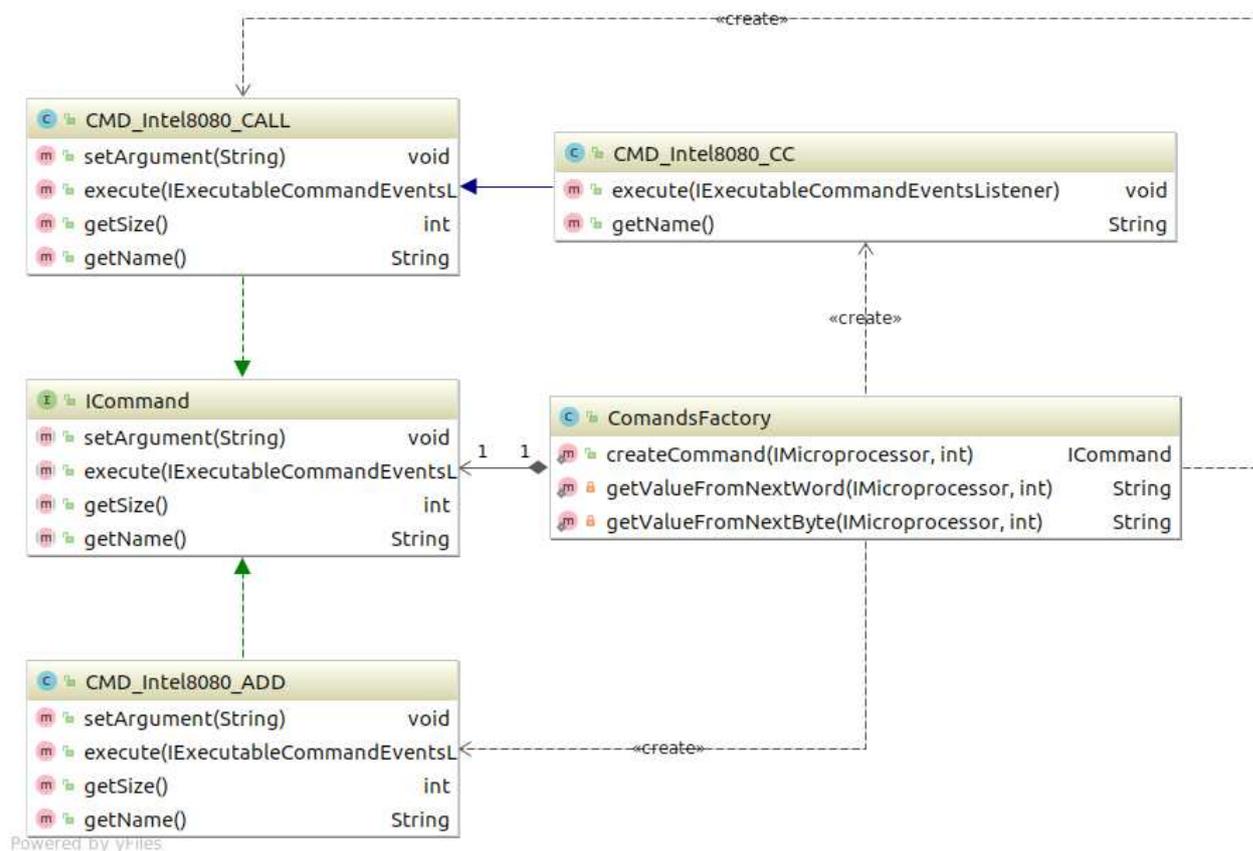


Рисунок 28 – Структура классов паттерна «Команда» в приложении

Подход с использованием данного паттерна позволяет повторно использовать написанный ранее код, что делает программное представление

системы команд более масштабируемым, а также позволяет эффективно проводить модульное тестирование системы.

### 3.2.3 Слушатель

В пространстве эмулятора выполняется пользовательская программа, поэтому в системе могут возникать события ввода-вывода, которые требуют соответствующей реакции на них. Примером может служить ситуация, когда пользовательская программа запрашивает ввод значения с клавиатуры – возникает ситуация, когда пользовательская программа должна отправить соответствующий запрос и получить на него ответ в виде введённого с клавиатуры значения. Для разрешения подобных ситуаций в приложении используется паттерн «Слушатель» (рисунок 29).

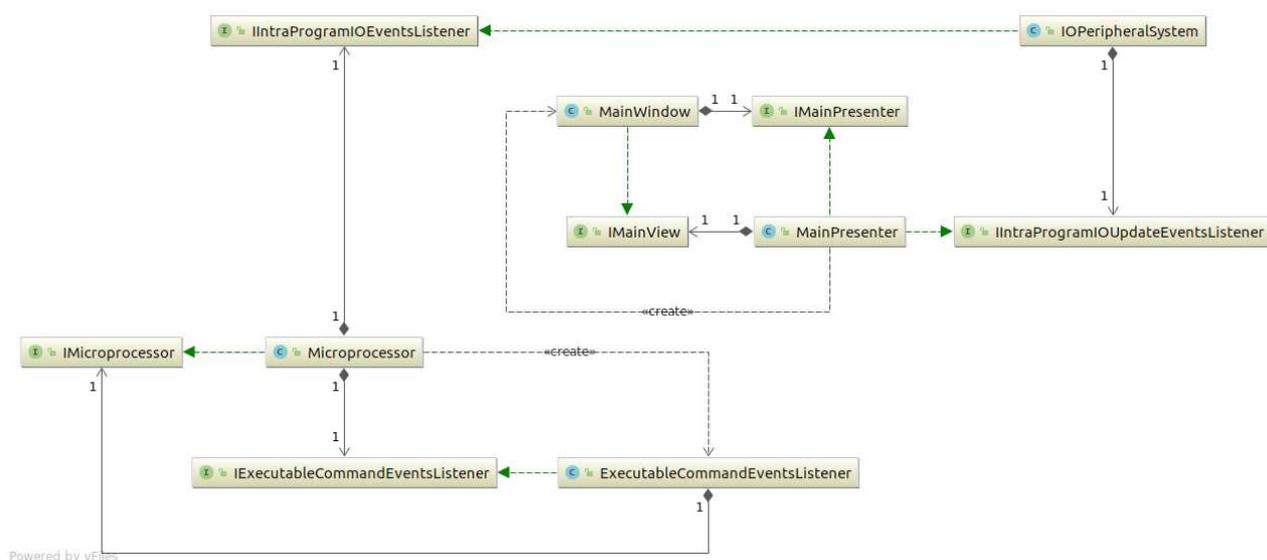


Рисунок 29 – Структура классов-слушателей и структура их использования в приложении

Запросы от пользовательской программы через каскад слушателей *IIntraProgramIOEventsListener*, реализуемый классом *IOPeripheralSystem*, и *IIntraProgramIOUpdateEventsListener*, реализуемый классом *MainPresenter*, передаются view компонентам, которые реагируют на него соответствующим образом.

### 3.3 Инструкции к программе

#### 3.3.1 Инструкция по сборке

Для сборки эмулятора необходимо иметь установленную JDK не ниже 1.6 версии.

1) Скопировать папку с исходным кодом эмулятора на компьютер вручную, либо выполнить: `git clone https://github.com/mozgolom/Intel_8080_microprocessor_emulator`

2) Выполнить: `sh ./i8080.sh` для Linux или `i8080.bat` для Windows

#### 3.3.2 Инструкция по установке и запуску

Для запуска эмулятора необходимо иметь установленную JVM не ниже 1.6 версии.

Данный программный эмулятор микропроцессора Intel 8080 не требует отдельной установки и состоит из одного файла `i8080.jar`

#### 3.3.3 Сборка API

Для сборки API необходимо иметь установленную JDK не ниже 1.6 версии.

Данный эмулятор микропроцессора Intel 8080 поддерживает возможность подключения внешних периферийных устройств с помощью специального API.

Для сборки библиотеки `api.jar` необходимо:

1) Скопировать папку с исходным кодом эмулятора на компьютер вручную, либо выполнить: `git clone https://github.com/mozgolom/Intel_8080_microprocessor_emulator`

2) Выполнить: `sh ./api.sh` для Linux или `api.bat` для Windows

#### 3.3.4 Работа с API

Исходный код API частично представлен ниже:

```
interface IDevice {
```

```

    void connection() throws UnknownHostException, IOException;
    void writeData(int value) throws IOException;
    void sendInterrupt() throws IOException;
}
class Device extends Thread implements IDevice {

    public Device(String description) {...}

    @Override
    final public void connection() throws UnknownHostException, IOException {...}
    @Override
    final public void writeData(int value) throws IOException {...}
    @Override
    final public void sendInterrupt() throws IOException {...}

    protected void readACK() {}
    protected void receivingData(int value) {}
}

```

Для реализации своего периферийного устройства необходимо выполнить наследования от класса *Device* и переопределить методы *readACK()* и *receivingData(int value)*.

Метод *connecton()* выполняет подключение устройства к серверу периферийных устройств эмулятора и запускает поток информационного обмена через порт эмулятора.

Метод *writeData(int value)* выставляет значение на порт устройства и оно становится доступным для считывания эмулятором.

Метод *sendInterrupt()* посылает эмулятору прерывание от периферийного устройства.

Метод *readACK()* вызывается, когда эмулятор прочитал значение из порта периферийного устройства. Вызов данного метода подтверждает чтение значения эмулятором.

Метод *receivingData(int value)* вызывается, когда эмулятор выполнил запись значения в порт периферийного устройства.

Для отключения периферийного устройства от эмулятора необходимо завершить его процесс.

## 3.4 Тестирование

### 3.4.1 Модульное тестирование

Модуль системы команд микропроцессора полностью покрыт unit-тестами. Для написания тестов использовалась библиотека модульного тестирования JUnit 4.

### 3.4.2 Ручное тестирование

Для проверки работоспособности эмулятора, а также интегрированных периферийных устройств (таймера, символьного и пиксельного экранов, консолей ввода и вывода) проводилось ручное тестирование приложения под операционными системами Ubuntu версии 17.10 и Windows 10. Весь функционал разработанного приложения корректно работает под обеими операционными системами.

Листинг программы находится в ПРИЛОЖЕНИИ Б. На рисунке 30 представлена работа эмулятора с пиксельным экраном.

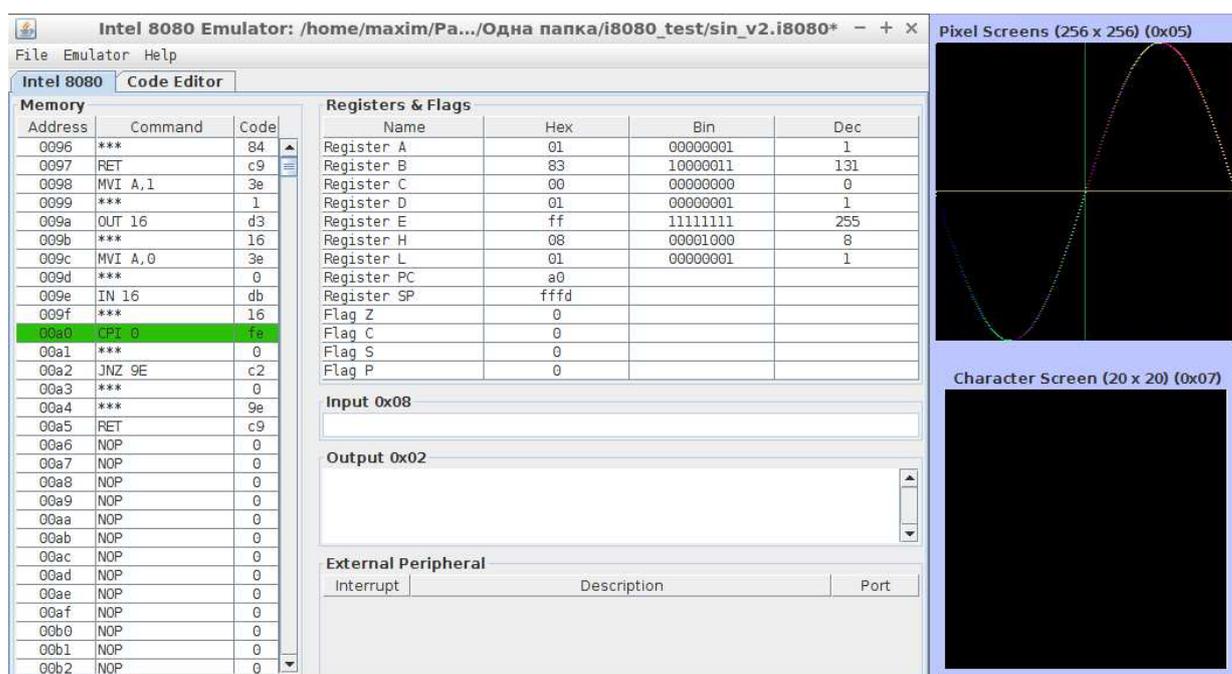


Рисунок 30 – Работа эмулятора с пиксельным экраном

Листинг программы находится в ПРИЛОЖЕНИИ В. На рисунке 31 представлена работа эмулятора с символьным экраном.

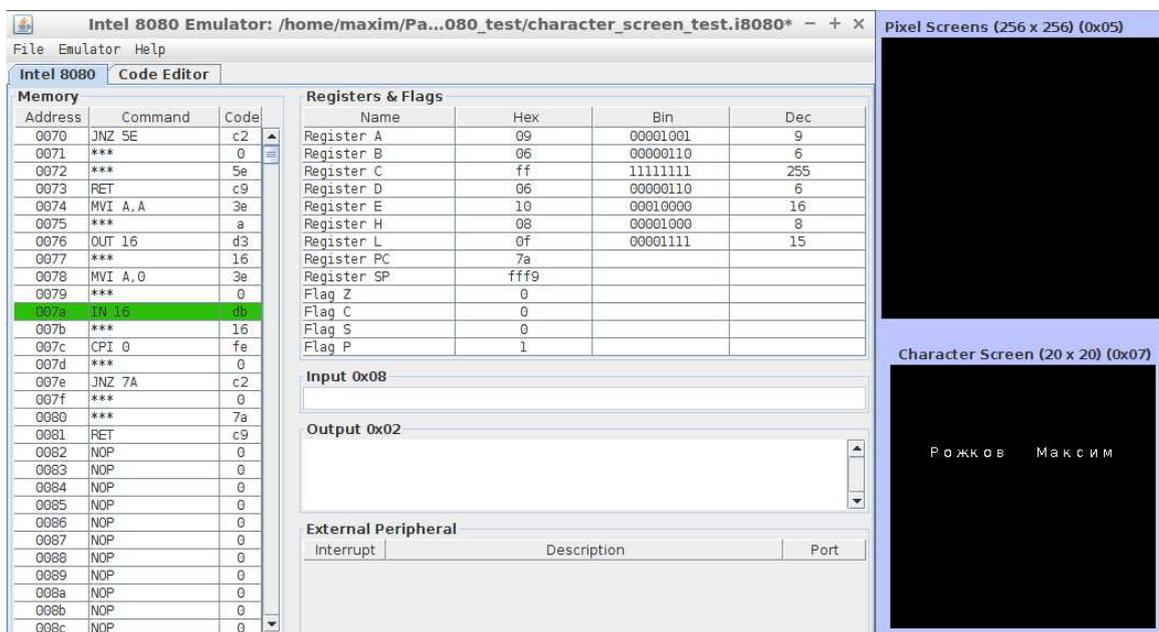


Рисунок 31 – Работа эмулятора с символьным экраном

Листинг программы находится в ПРИЛОЖЕНИИ Г. На рисунке 32 представлена работа эмулятора с внешним периферийным устройством (LED-индикацией).

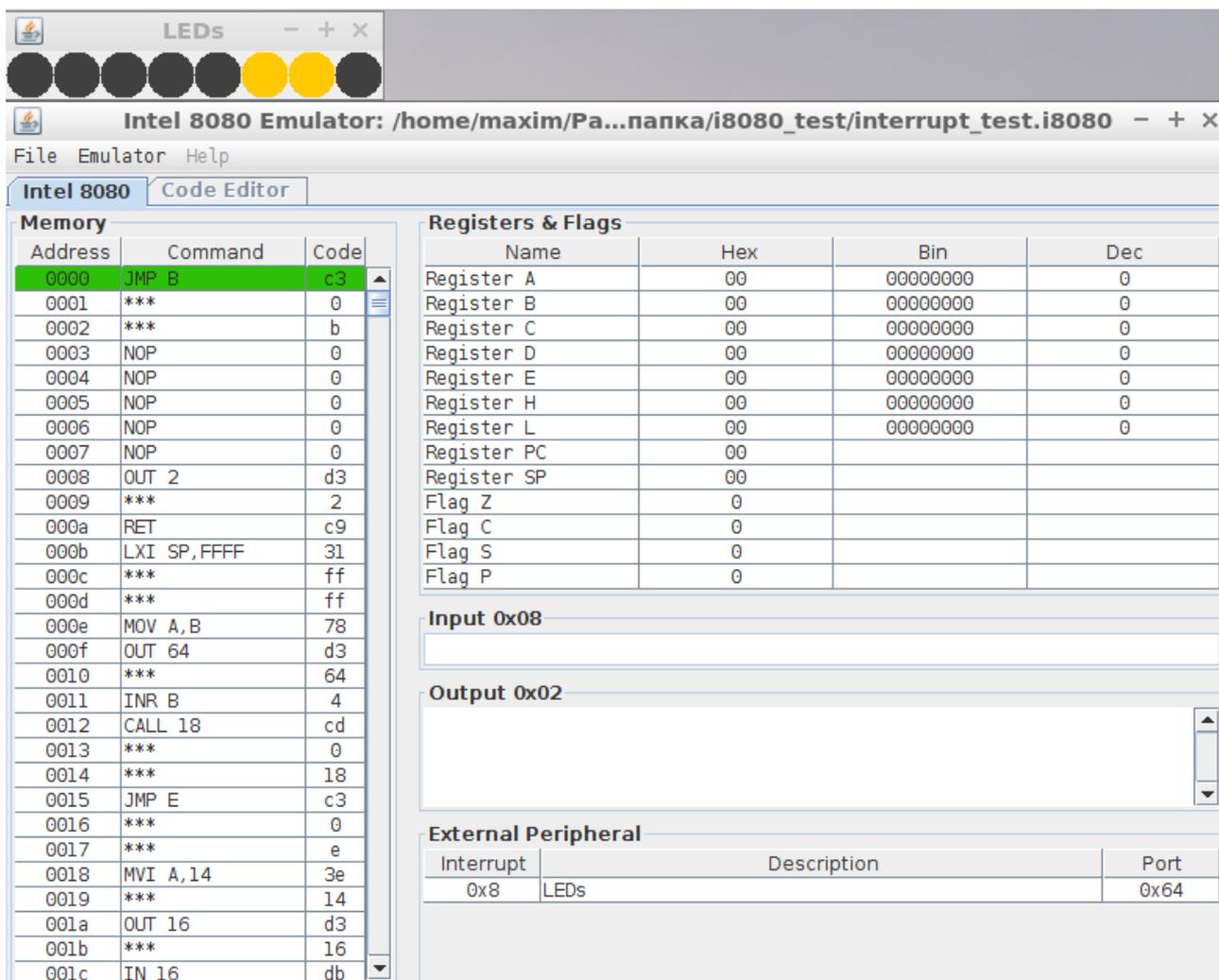


Рисунок 32 – Работа эмулятора с подключенным внешним периферийным устройством (LED-индикация)

### 3.5 Выводы по главе

- 1) Реализован программный эмулятор микропроцессора Intel 8080/8085.
- 2) Составлены инструкции по сборке, установке, запуску, а также написано руководство пользователя. Составлена документация к коду программы в HTML формате, которая доступна в репозитории проекта [8].
- 3) Проведено модульное тестирование системы команд микропроцессора и ручное тестирование всех модулей разработанного эмулятора под двумя операционными системами.

## **ЗАКЛЮЧЕНИЕ**

В результате проделанной работы был спроектирован и реализован программный эмулятор микропроцессора Intel 8080/8085. В разработанном эмуляторе реализована возможность подключения внешних устройств через API, также осуществляется эмуляция работы контроллера прерываний. В программном эмуляторе доступен редактор исходного кода программы со встроенным транслятором, выполняющим преобразование текста программы в машинный код.

Программный эмулятор микропроцессора Intel 8080/8085 доступен для скачивания с git-репозитория [8].

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Эмулятор микрокомпьютера на базе микропроцессора Intel 8080 [Электронный ресурс] – : Режим доступа: <https://www.tramm.li/i8080/>
2. Эмулятор персонального компьютера Радио-РК86 [Электронный ресурс] – : Режим доступа: <http://rk86.ru/>
3. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон. – Москва : ДМК, 2006. – 493 с.
4. Java [Электронный ресурс] – : Режим доступа: <https://java.com/>
5. JetBrains: Developer Tools for Professionals and Teams [Электронный ресурс] – : Режим доступа: <https://www.jetbrains.com/>
6. Plant UML [Электронный ресурс] : – Режим доступа: <http://plantuml.com/>
7. Github [Электронный ресурс] : – Режим доступа: <https://github.com/>
8. Git-репозиторий проекта [Электронный ресурс] : Intel 8080 Microprocessor Emulator – Режим доступа: [https://github.com/mozgolom/Intel\\_8080\\_microprocessor\\_emulator](https://github.com/mozgolom/Intel_8080_microprocessor_emulator)

# ПРИЛОЖЕНИЕ А

## Модель предметной области



## ПРИЛОЖЕНИЕ Б

### Листинг программы эмулятора для работы с пиксельным экраном

```
.adr:0x0000                                mvi a, 0xAA
Reset:                                      out 0x05
      lxi sp, 0xFFFF                       inr c
      mvi d, 0x00                           jnz DrawHLine
Init:                                       ret
      lxi h, 0x0800                          DrawVLine:
      mov a, l                               mov a, b
      add d                                  out 0x05
      mov l, a                               mov a, c
      inr d                                  out 0x05
      mvi e, 0xFF                           mvi a, 0x00
      mvi b, 0                               out 0x05
      mvi c, 127                             mvi a, 0x22
      call DrawVLine                         out 0x05
      mvi b, 127                             inr b
      mvi c, 0                               jnz DrawVLine
      call DrawHLine                         ret
      mvi c, 0x00                            ClearScreen:
      mvi b, 0x00                           lxi h, 0x0800
Loop:                                       mov a, l
      inx h                                  add d
      mov a, l                               dcr a
      cpi 255                               mov l, a
      jnz Skip                              mvi e, 0x00
      mvi l, 0x00                           mvi c, 0x00
Skip:                                       mvi b, 0x00
      mov a, c                               ClearLoop:
      mov b, m                               mov b, m
      call DrawPoint                         inx h
      inr e                                  mov a, l
      inr c                                  cpi 255
      jnz Loop                              jnz Skip2
      call Delay                             mov a, h
      call ClearScreen                       mvi l, 0x00
      jmp Init                               Skip2:
DrawPoint:                                  mov a, c
      mov a, b                               call DrawPoint
      out 0x05                              inr c
      mov a, c                               jnz ClearLoop
      sui 0                                  ret
      jnc Skip3                             Delay:
      mvi a, 0x00                           mvi a, 1
Skip3:                                       out 0x16
      out 0x05                              mvi a, 0
      mvi a, 0x00                           DelayLoop:
      out 0x05                              in 0x16
      mov a, e                              cpi 0x00
      out 0x05                              jnz DelayLoop
      ret                                    ret
DrawHLine:                                  mov a, b
      mov a, b                               out 0x05
      out 0x05                              mov a, c
      mov a, c                               out 0x05
      out 0x05                              mvi a, 0x00
      mvi a, 0x00                           out 0x05
```

## ПРИЛОЖЕНИЕ В

### Листинг программы эмулятора для работы с символьным экраном

```
.adr:0x0000
Init:
    lxi sp, 0xFFFF
    mvi b, 20
    push b
    mvi b, 0xFF
    mvi c, 0x02
    push b
Loop:
    pop b
    inr b
    push b
    push b
    lxi h, 0x0800
    call LoadString
    call Delay
    pop b
    push b
    call ClearRow
    pop d
    pop b
    dcr b
    push b
    push d
    jnz Loop
    jmp Init
    hlt
LoadString:
    pop b
    pop d
    push b
    push d
    mov e, m
    inx h
    push d
LoadLoop:
    pop b
    dcr c
    jm EndLoad
    pop d
    inr e
    push d
    push b
    mvi b, 0b00000111
    mov c, m
    inx h
    push b
    push d
    call LoadChar
    jmp LoadLoop
EndLoad:
    pop d
    ret
LoadChar:
    pop d
    pop b
    mov a, b
    out 0x07
    mov a, c
    out 0x07
    mvi a, 0x00
    out 0x07
    pop b
    mov a, b
    out 0x07
    mov a, c
    out 0x07
    push d
    ret
ClearRow:
    mvi c, 0x00
ClearRowLoop:
    mov a, b
    out 0x07
    mov a, c
    out 0x07
    mvi a, 0x00
    out 0x07
    out 0x07
    out 0x07
    inr c
    mov a, c
    cpi 19
    jnz ClearRowLoop
    ret
Delay:
    mvi a, 10
    out 0x16
    mvi a, 0
DelayLoop:
    in 0x16
    cpi 0x00
    jnz DelayLoop
    ret
.adr:0x0800
.set:14
.set:208
.set:238
.set:230
.set:234
.set:238
.set:226
.set:204
.set:224
.set:234
.set:241
.set:232
.set:236
```

## ПРИЛОЖЕНИЕ Г

### Листинг программы эмулятора для работы с LED-индикатором

```
.adr:0x0000
    jmp Init
.adr:0x0008
    Interrupt1:
        out 0x02
        ret
Init:
    lxi sp, 0xFFFF
MainLoop:
    mov a, b
    out 0x64
    inr b
    call Delay
    jmp MainLoop
Delay:
    mvi a, 20
    out 0x16
DelayLoop:
    in 0x16
    cpi 0
    jnz DelayLoop
    ret
```

Федеральное государственное автономное  
образовательное учреждение  
высшего образования  
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий  
институт

Вычислительная техника  
кафедра

УТВЕРЖДАЮ  
Заведующий кафедрой ВТ  
О. В. Непомнящий  
подпись инициалы, фамилия  
« 15 » 06 2018 г.

5, а

**БАКАЛАВРСКАЯ РАБОТА**

09.03.01 Информатика и вычислительная техника  
код и наименование направления

Программный эмулятор микропроцессора Intel 8080  
тема

Руководитель	<u>Васильев, 16.18</u> подпись, дата	ст. преподаватель должность, учёная степень	<u>В. С. Васильев</u> инициалы, фамилия
Выпускник	<u>М. Рожков, 01.06.18</u> подпись, дата		<u>М. В. Рожков</u> инициалы, фамилия
Консультант	<u>Л. И. Покидышева, 4.06.18</u> подпись, дата	доцент, канд. техн. наук должность, учёная степень	<u>Л. И. Покидышева</u> инициалы, фамилия
Нормоконтролер	<u>В. И. Иванов, 2.06.18</u> подпись, дата	доцент, канд. техн. наук должность, учёная степень	<u>В. И. Иванов</u> инициалы, фамилия

Красноярск 2018