БГУИР

Кафедра ЭВМ

Отчет по лабораторной работе № 4

Тема: «Подсистема прерываний»

Выполнил: Чернявский Я.А.

студент группы 724402

Проверил:
к.т.н., доцент Селезнёв И.Л.

Минск

2019

1. Цель

Изучить организацию прерываний в IBM PC с использованием контроллера прерываний.

2. Задание

Под MS DOS написать программу, которая:

1) выполняет инициализацию контроллера прерываний;
2) выводит на экран содержимое регистров запросов, обслуживаний и масок для ведущего и ведомого контроллеров (через видеобуфер).

3. Теоретические сведения

Каждый момент времени центральный процессор может работать только с одним устройством. Циклический опрос каждого устройства с последующей обработкой запроса оказался неэффективным. Решение задачи оказался контроллер прерываний, который принимает запросы от устройств и в соответствии с приоритетом направляет их процессору, если прерывание от данного устройства не замаскировано (разрешено) в регистре масок. Если прерывание разрешено и устройство его запросило, то устанавливается соответствующий устройству бит в регистре запросов.

Контроллер прерываний состоит из двух микросхем, подключенных каскадно (ведущий и ведомый контроллеры), каждая из которых имеет по 8 линий прерываний (IRQ0-IRQ7, IRQ8-IRQ15). За каждой линией закреплено определенное устройство.

Когда процессор получает запрос на прерывание, он сохраняет свое текущее состояние и переключается на выполнение запрошенной операции. При этом устанавливается бит в регистре обслуживания (бит запроса сбрасывается). После обслуживания прерываний сбрасывается бит обслуживания, посылается сигнал EOI (endofinterrupt), процессор переключается на выполняемую ранее задачу.

Для доступа к контроллеру прерываний используются порты 20h и 21h (для ведущего), A0 иA1h (для ведомого). Регистр масок доступен через порт 21h /A1h. Чтобы изменить определенный бит, нужно считать значение из этого регистра, изменить нужный бит, записать значение обратно.

Чтобы считать регистр запросов, его нужно выбрать (записать в 20h/A0h значение 0Ah), а затем считать содержимое из порта 20h/A0h. Чтобы считать регистр обслуживания, его нужно выбрать (записать в 20h/A0h значение 0Bh), а затем считать содержимое из порта 20h/A0h.

Прерывание — сигнал, сообщающий процессору о наступлении какоголибо события. При этом выполнение текущей последовательности команд приостанавливается, и управление передаётся обработчику прерывания, который реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код. Прерывание связывают с числом, называемым номером типа прерывания или номером прерывания.

В зависимости от источника возникновения сигнала, прерывания делятся на асинхронные, синхронные и программные (рис. 3.1).

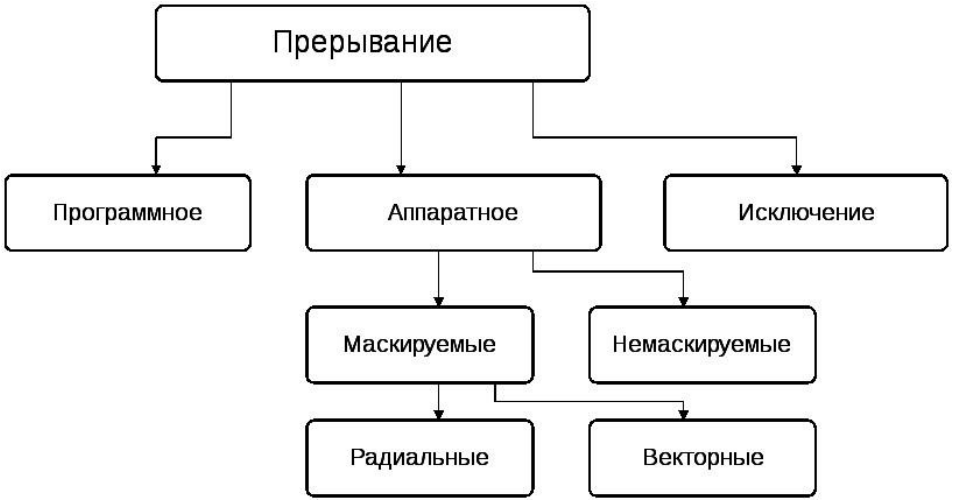

Рис. 3.1 – Виды прерываний

Асинхронные, аппаратные, или внешние — события, которые исходят от внешних источников (например, периферийных устройств) и могут произойти в любой произвольный момент. Эти прерывания информируют систему о событиях, связанных с работой устройств, например, о том, что наконец-то завершилась печать символа на принтере или что требуемый сектор диска уже прочитан его содержимое доступно программе. Факт возникновения в системе такого прерывания трактуется как запрос на прерывание IRQ;

Асинхронные прерывания делятся на маскируемые и немаскируемые.

Маскируемые — прерывания, которые можно запрещать установкой соответствующих битов в регистре маскирования прерываний.

Немаскируемые NMI — обрабатываются всегда, независимо от запретов на другие прерывания. К примеру, такое прерывание может быть вызвано сбоем в микросхеме памяти.

Синхронные, или внутренние — события в самом процессоре как результат нарушения каких-то условий при исполнении машинного кода:

деление на ноль или переполнение стека, обращение к недопустимым адресам памяти или недопустимый код операции;

Программные (частный случай синхронного прерывания) — инициируются исполнением специальной инструкции в коде программы. Программные прерывания как правило используются для обращения к функциям встроенного программного обеспечения (firmware), драйверов и операционной системы. Программные прерывания с заданным номером вызываются через команду INT. Прикладные программы могут сами устанавливать свои обработчики прерываний для их последующего использования другими программами. Для этого встраиваемые обработчики прерываний должны быть резидентными в памяти.

Заметим еще, что обработчики прерываний могут сами вызывать программные прерывания, например, для получения доступа к сервису BIOS или DOS.

Задача контроллера прерываний – прием запросов прерываний от вычислительных устройств, сравнение из приоритетов и посылки запросов прерывания в ЦП вместе с информацией о местоположении соответствующих подпрограмм. В функции контроллера прерываний также входит изменение дисциплины обслуживания вычислительного устройства, т.е. установка различных режимов, присвоение приоритетов исследуемого устройства в соответствии с общими требованиями к системе.

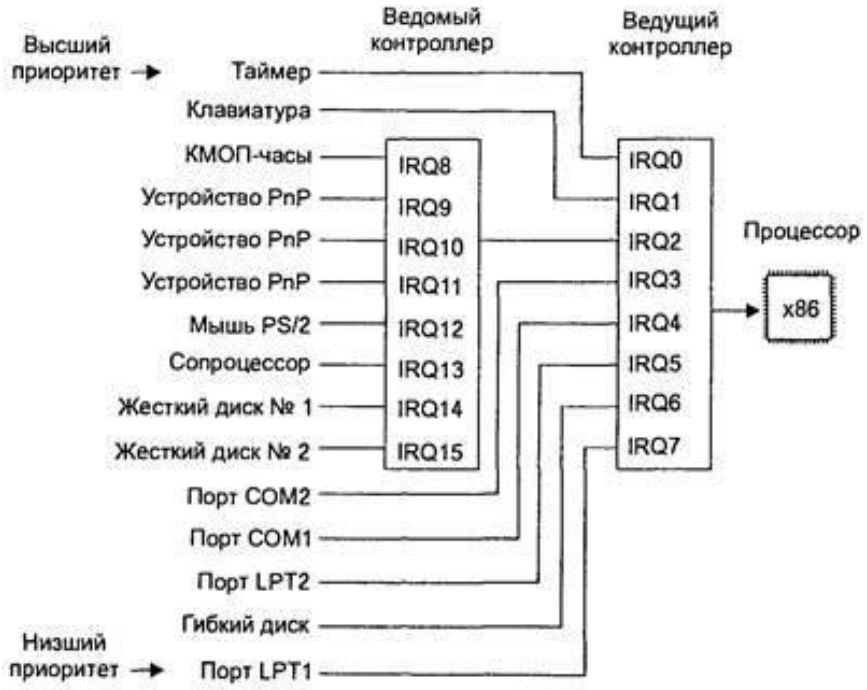

Рис. 3.2 – Схема контроллера прерываний

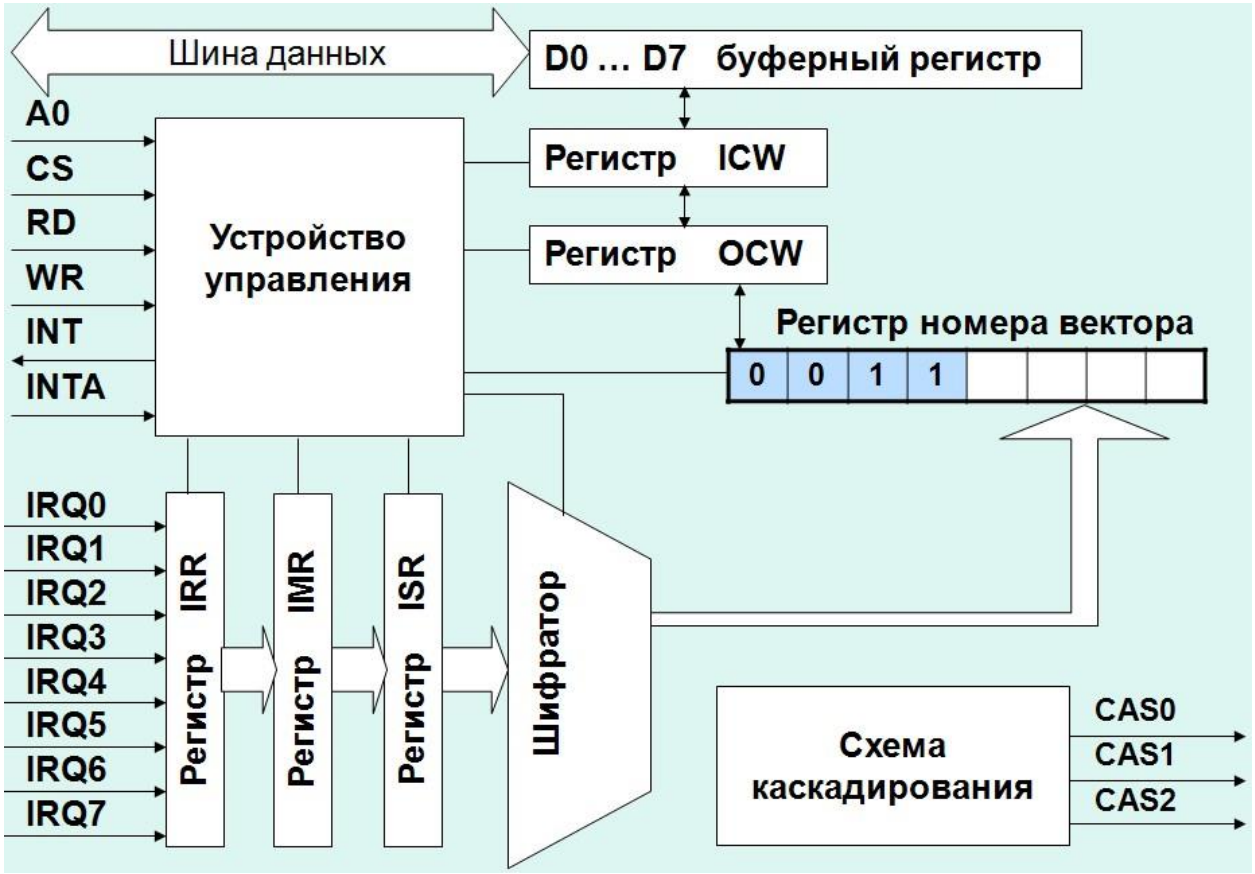

Рис. 3.3 – Регистры масок прерываний

Регистр Масок Прерываний – хранит маску, с помощью которой можно запретить обслуживание запросов по любому входу.

Регистр Запроса Прерываний – служит для запоминания всех запросов от вычислительного устройства по входам IR7-IR0.

Регистр Обслуживаемых Прерываний – содержит единицу на позиции, соответствующей обслуживаемым запросам, причем каждая единица запрещает обслуживание запросов с меньшим приоритетом если отсутствует режим специального маскирования.

Управляющее Устройство – содержащее регистры команд инициализации (РКИ) и регистры рабочих команд (РРК), обеспечивает выработку внешних и внутренних управляющих сигналов.

Блок Каскадирования – осуществляет связь ведущего контроллера с ведомым при использовании нескольких контроллеров прерываний микропроцессорной системы.

Контроллер прерываний воспринимает запрос к прерыванию по входам IR от одного или нескольких вычислительных устройств. Записывает 1 соответственно РЗП. Проверяет маскирование. Проверяет запрос с наивысшим

приоритетом. Посылает сигнал прерывания INTA на соотв. вход ЦП. Если прерывания разрешены, ЦП заканчивает выполнение текущей команды и активизирует сигнал подтверждения прерывания INTA. По получении импульса INTA, ПКП устанавливает 1 в разряд регистра РОП соответствующий запросу, подтверждающему обслуживание. Одноименный разряд регистра РЗП сбрасывается в ноль. ЦП вырабатывает второй импульс INTA, с получением которого контроллер посылает по шине данных в ЦП восьмиразрядный указатель (вектор), используемый для определения начального адреса подпрограммы обслуживания прерывания.

ЦП переходит на выполнение подпрограммы обслуживания прерывания. В режиме автоматического окончания прерывания, цикл входа в прерывание завершается по окончанию второго импульса INT, по которому соответствующий разряд РОП сбрасывается в ноль. В режиме обычного окончания разряд регистра РОП соответствующий текущему уровню прерывания остается 1, до появления команды конца прерывания EOI. Для ввода информации в контроллер прерывания используются 2 порта ввода-вывода. Порт 20h/A0h и порт 21h/A1h. Через эти порты могут быть переданы 4 слова инициализации ICW1- ICW4, задающие режим работы контроллер прерывания, и 3 операционных управляющих слова рабочих приказов OCW1 - OCW3.

Биты	7	6	5	4	3	2	1	0
Описание	0	0	0	1	PT	Размер	Режим	ICW4

Таблица 3.1 – Формат команды ICW1

- Бит 0 управляет использованием ICW4. Если бит установлен в 1, команда будет вызвана.
- Бит 1 определяет использование ведомого контроллера (1 – не используется, 0 – используется).
- Бит 2 определяет размер вектора прерываний (1 – 4 байта, 0 – 8 байтов).
- Бит 3 определяет режим срабатывания триггера (1 – по фронту, 0 – по уровню).
- Бит 4 должен быть установлен в 1.
- Биты 5-7 должны быть установлены в 0.

Рабочие команды – OCW – для задания маски, задания различных режимов работы и для обеспечения контрольного считывания содержимого регистров контроллера прерываний.

Бит	7	6	5	4	3	2	1	0

Порт 21h	IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0
Порт A0h	IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8

Таблица 3.2 – Формат команды OCW1

Биты	7	6	5	4	3	2	1	0
Описание	Использование EOI			0	0	Приоритет		

Таблица 3.3 – Формат команды OCW2

- Биты 0-2 определяют номер линии (IRQ), которая будет использована, если бит 6 установлен в 1. Возможны следующие значение: 000b – выход 0, 001b – выход 1, 010b – выход 2, 011b – выход 3, 100b – выход 4, 101b – выход 5, 110b – выход 6, 111b – выход 7. Биты 3-7 должны быть установлены в 0.
- Биты 3-4 определяют код команды OCW2 (00b). Должны быть установлены в 0.
- Биты 5-7 определяют вариант использования команды окончания прерывания (EOI). Возможны следующие значения:
  - 000b – автоматический сдвиг приоритетов запрещён;
  - 100b – автоматический сдвиг приоритетов разрешён;
  - 001b – неспецифичная команда EOI;
  - 011b – специфичная команда EOI;
  - 101b – сдвиг приоритетов для неспецифичной команды EOI;
  - 111b – сдвиг приоритетов для специфичной команды EOI; 110b – использование сдвига приоритетов; 010b – нет никаких операций.

Биты	7	6	5	4	3	2	1	0
Описание	0	Маска		0	1	Опрос	Статус	

Таблица 3.4 – Формат команды OCW3

- Биты 0-1позволяют установить режим доступа к состоянию контроллера. Возможны следующие значения: 00b и 01b – не читать состояния, 10b – читать регистр состояния на следующем прерывании, 11b – читать регистр состояния.
- Бит 2 используется для метода поллинга (опроса). Если бит установлен в 1, режим полллинга используется.
- Биты 3-4 определяют код команды OCW2 (01b).

- Биты 5-6 определяют состояние специального режима маски (00b и 01b – не использовать, 10b – выключить специальный режим маски, 11b – включить специальный режим маски).
- Бит 7 зарезервирован и должен быть установлен в 0.

Контроллер прерываний имеет следующие режимы работы: Режим полной вложенности, при котором каждый вход запроса прерываний имеет фиксированный приоритет от высшего уровня IR0 до низшего IR7, является стандартным и не требует использования рабочих команд, остальные режимы, а также состояние регистра маски РМП, могут быть заданы с помощью специальных рабочих команд.

Режим полной вложенности устанавливается сразу после окончания инициализации. Приоритеты запросов прерываний упорядочиваются в стороны прерывания от 0 до 7.

В системе с каскадированием контроллеров полная вложенность обеспечивается только по входам ведущего контроллера. При обслуживании запроса по какому-либо входу ведомого контроллера, запросы по другим его входам, даже с более высоким приоритетом, не обслуживаются.

Специальный режим полной вложенности применяется тогда, когда в системе используется каскадирование контроллеров и вложенность приоритетов должна сохраняться для каждого контроллера. Режим устанавливается командой ICW4 и отличается от обычного режима полной вложенности следующим: когда обслуживается запрос от некоторого ведомого контроллера, этот контроллер не закрыт от приоритетной логики ведущего контроллера и дальнейшие запросы на прерывание от входов с более высоким приоритетом в пределах ведомого контроллера будут распознаваться ведущим контроллером и инициировать запрос прерывания для ЦП. В обычном режиме полной вложенности ведомый контроллер маскируется если его запрос находится в обслуживании, так что другие запросы от него не воспринимаются. Если запускается подпрограмма обслуживания прерывания, то программным путем необходимо проверить является ли данное прерывание единственным от этого ведомого контроллера.

Режим автоматического сдвига приоритетов – Режим А, он устанавливается командой OCW2. В контроллере имеется также режим программного сдвига приоритетов и называется режим Б, в этом случае с помощью команды OCW2 можно задавать номер входа, которому будет присвоен низший приоритет. Приоритеты других входов будут зафиксированы по отношению к заданному.

Режим специального маскирования применяется для того, чтобы программы обслуживания прерываний могли динамически изменять структуру системы с приоритетом в процессе работы. Например, при выполнении какой-либо части подпрограммы обслуживания, необходимо запретить запросы более низких уровней, а при выполнении другой части – разрешить их. Трудность реализации таких действий состоит в том, что пока выполняется подпрограмма и соответствующий разряд в регистре РОП не сброшен, контроллер не реагирует на запросы с более низким приоритетом. Для разрешения прерываний со всех уровней, в т.ч. и с более низких, устанавливается режим специального маскирования (или затенения), при котором запрещается прерывание только на данном уровне. Он устанавливается командой OCW3.

Режим поллинга (опроса) применяется для организации обслуживания запросов прерываний по инициативе программы, выполняемой микропроцессором. При этом прерывания процессора запрещаются путем сброса флага IF и используется команда POLL, которая задается с помощью OCW3. Тогда контроллер воспринимает следующий импульс по входу RD как подтверждение прерывания и выдает на шину данных байт, в котором D7=1, если на вход данного контроллера поступил запрос, соответственно разряды D2- D0 задают двоичный код входа высшего уровня среди тех, по которым поступили запросы. С помощью команды POLL последовательно опрашиваются все контроллеры прерывания в системе, что позволяет легко наращивать число уровней приоритетов. В этом режиме таблица адресов подпрограмм обслуживания не используется, и он полезен в тех случаях, когда одна подпрограмма применяется для обслуживания нескольких уровней прерывания.

Контроллеры прерываний имеют встроенные средства их каскадирования с целью увеличения числа уровня прерываний. Когда активизируется вход запроса ведомого контроллера, ведущий контроллер разрешает ведомому выдать номер, соответствующий уровню прерывания по второму импульсу.

Для того чтобы связать адрес обработчика прерывания с номером прерывания, используется таблица векторов прерываний, занимающая первый килобайт оперативной памяти - адреса от 0000:0000 до 0000:03FF. Таблица состоит из 256 элементов - FAR-адресов обработчиков прерываний. Эти элементы называются векторами прерываний. В первом слове элемента таблицы записано смещение, а во втором - адрес сегмента обработчика прерывания.

Иногда может потребоваться организовать обработку некоторых прерываний. Для этого программа должна переназначить вектор на свой обработчик. Это можно сделать, изменив содержимое соответствующего элемента таблицы векторов прерываний.

Составление собственных программ обработки прерываний и замена стандартных обработчиков DOS и BIOS является ответственной и сложной работой. Необходимо учитывать все тонкости работы аппаратуры и взаимодействия программного и аппаратного обеспечения. При отладке возможно разрушение операционной системы с непредсказуемыми последствиями.

Очень важно не забыть перед завершением работы восстановить содержимое измененных векторов в таблице прерываний, т.к. после завершения работы программы память, которая была ей распределена, считается свободной и может быть использована для загрузки другой программы. Если вектор не был восстановлен и пришло прерывание, то система может разрушиться, так как вектор теперь указывает на область, которая может содержать что угодно.

Поэтому последовательность действий для нерезидентных программ, желающих обрабатывать прерывания, должна быть такой. Прочитать содержимое элемента таблицы векторов прерываний для вектора с нужным вам номером. Запомнить это содержимое, адрес старого обработчика прерывания, в области данных программы; Установить новый адрес в таблице векторов прерываний так, чтобы он соответствовал началу Вашей программы обработки прерывания; перед завершением работы программы прочитать из области данных адрес старого обработчика прерывания и записать его в таблицу векторов прерываний.

Кроме того, операция изменения вектора прерывания должна быть непрерывной в том смысле, что во время изменения не должно произойти прерывание с номером, для которого производится замена программы обработки.

Для облегчения работы по замене прерывания DOS предоставляет в распоряжение специальные функции для чтения элемента таблицы векторов прерывания и для записи в нее нового адреса. DOS гарантирует, что операция по замене вектора будет выполнена правильно. Нет необходимости заботиться о непрерывности процесса замены вектора прерывания. DOS функции: void setvect(int intr_num, void interrupt(*isr)()); для установки новой

программы обработки прерывания с номером intr_num, передавая ее адрес в параметре isr. Адрес подпрограммы можно передать только в том случае, если она объявлена как подпрограмма обработки прерывания (interrupt).

void interrupt(*getvect(int intr_num))(); для считывания значения вектора с номером intr_num и интерпретирует прочитанное , как "дальний" указатель (с атрибутом "far") на некоторую функцию прерывания. Параметр intr_num должен иметь значение от 0 до 255. Модификатор interrupt описывает функцию, которая является обработчиком прерывания. Такая функция завершается командой возврата из обработки прерывания IRET, и для нее автоматически генерируются команды сохранения регистров на входе и их восстановления при выходе из обработчика прерывания.

Функция обработки прерывания должна быть FAR-функцией, т.к. таблица векторов прерываний содержит полные адреса в виде сегмент:смещение. Ключевое слово interrupt используется также для описания переменных, предназначенных для хранения векторов прерываний.

Таблица прерываний:

Номер	**Описание**
0	Ошибка деления. Вызывается автоматически после выполнения команд DIV или IDIV, если в результате деления происходит переполнение (например, при делении на 0). Обычно при обработке этого прерывания MS-DOS выводит сообщение об ошибке и останавливает выполнение программы. При этом для процессора i8086 адрес возврата указывает на команду, следующую после команды деления, а для процессора i80286 и более старших моделей - на первый байт команды, вызвавшей прерывание
1	Прерывание пошагового режима. Вырабатывается после выполнения каждой машинной команды, если в слове флагов установлен бит пошаговой трассировки TF. Используется для отладки программ. Это прерывание не вырабатывается после пересылки данных в сегментные регистры командами MOV и POP

2	Аппаратное немаскируемое прерывание. Это прерывание может использоваться по-разному в разных машинах. Обычно оно вырабатывается при ошибке четности в оперативной памяти и при запросе прерывания от сопроцессора
3	Прерывание для трассировки. Генерируется при выполнении однобайтовой машинной команды с кодом CCh и обычно используется отладчиками для установки точки прерывания
4	Переполнение. Генерируется машинной командой INTO , если установлен флаг переполнения OF. Если флаг не установлен, команда INTO выполняется как NOP. Это прерывание используется для обработки ошибок при выполнении арифметических операций

5	Печать копии экрана. Генерируется, если пользователь нажал клавишу <PrtSc>. В программах MS-DOS обычно используется для печати образа экрана. Для процессора i80286 и более старших моделей генерируется при выполнении машинной команды BOUND, если проверяемое значение вышло за пределы заданного диапазона
6	Неопределенный код операции или длина команды больше 10 байт
7	Особый случай отсутствия арифметического сопроцессора
8	IRQ0 - прерывание интервального таймера, возникает 18,2 раза в секунду
9	IRQ1 - прерывание от клавиатуры. Генерируется, когда пользователь нажимает и отжимает клавиши. Используется для чтения данных из клавиатуры
A	IRQ2 - используется для каскадирования аппаратных прерываний

B	IRQ3 - прерывание асинхронного порта COM2
C	IRQ4 - прерывание асинхронного порта COM1
D	IRQ5 - прерывание от контроллера жесткого диска (только для компьютеров IBM PC/XT)
E	IRQ6 - прерывание генерируется контроллером НГМД после завершения операции ввода/вывода
F	IRQ7 - прерывание от параллельного адаптера. Генерируется, когда подключенный к адаптеру принтер готов к выполнению очередной операции. Обычно не используется
10	Обслуживание видеоадаптера

11	Определение конфигурации устройств в системе
12	Определение размера оперативной памяти
13	Обслуживание дисковой системы
14	Работа с асинхронным последовательным адаптером
15	Расширенный сервис
16	Обслуживание клавиатуры
17	Обслуживание принтера
18	Запуск BASIC в ПЗУ, если он есть

19	Перезагрузка операционной системы
1A	Обслуживание часов
1B	Обработчик прерывания, возникающего, если пользователь нажал комбинацию клавиш <Ctrl+Break>
1C	Программное прерывание, вызывается 18,2 раза в секунду обработчиком аппаратного прерывания таймера
1D	Адрес видеотаблицы для контроллера видеоадаптера 6845
1E	Указатель на таблицу параметров дискеты
1F	Указатель на графическую таблицу для символов с кодами ASCII 128-255
20-5F	Используется MS-DOS или зарезервировано для MS-DOS
60-67	Прерывания, зарезервированные для программ пользователя
68-6F	Не используются
70	IRQ8 - прерывание от часов реального времени
71	IRQ9 - прерывание от контроллера EGA
72	IRQ10 – зарезервировано
73	IRQ11 – зарезервировано
74	IRQ12 – зарезервировано

75	IRQ13 - прерывание от арифметического сопроцессора
76	IRQ14 - прерывание от контроллера жесткого диска
77	IRQ15 – зарезервировано
78 - 7F	Не используются
80-85	Зарезервировано для BASIC
86-F0	Используются интерпретатором BASIC
F1-FF	Не используются

Таблица 3.5 – Векторы прерываний

Часто при выполнении критических участков программ для того, чтобы гарантировать выполнение определенной последовательности команд целиком приходится запрещать прерывания. Это можно сделать DOS командой disable(). Ее нужно поместить в начало критической последовательности команд, а в конце расположить DOS команду enable(), разрешающую процессору воспринимать прерывания. Команда disable() запрещает только маскируемые прерывания, немаскируемые всегда обрабатываются процессором.

Если прерывания происходят часто, то их обработка может сильно замедлить работу прикладной программы. Поэтому обработчик прерывания должен быть короткой, быстро работающей программой, которая выполняет только самые необходимые действия. Например, считать очередной символ из порта принтера и поместить его в буфер, увеличить значение какого-либо глобального счетчика прерываний и т.п.

4. Листинг программы

```
//724402-2  L4  Chernyavsky Y.A.  Podsistema prerivaniy
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

char far *v = (char far *)0xB8000000;                    // Указатель на память видеобуфера
```

```
int quitFlag = 0, timerCounter = 0,                    // Флаг выхода, счётчик времени
colorId = 0, posX = 0, posY = 7;                       // ID цвета, координаты x и y
void interrupt far(*oldInt9)(void);                    // Указатель на обработчик прерывани Int9
void interrupt newInt9(void);                                  // Функция обработки прерывани Int9
void interrupt far(*oldInt8)(void);                    // Указатель на обработчик прерывани Int8
void interrupt newInt8(void);                                  // Функция обработки прерывани Int8
void interrupt far(*oldInt0)(void);                    // Указатель на обработчик прерывания Int0
void interrupt newInt0(void);                                  // Функция обработки прерывани Int0
void print(char*, int);                          // Функция вывода в консоль через видеопамять
void printTo(char*, int, int);           // Функция вывода в консоль через видеопамять в произвольное место
void printToWithColor(char*, int, int, int);   // Функция вывода в консоль через видеопамять в произвольное
место с заданным цветом
void setColorFor(int, int, int);                       // Функции установки цвета на ограниченную область
void intToHex(int, char*);                           // Функция конвертации в 16-ричную систему счисления
void intToBin(int, char*);                     // Функция конвертации в 2-ичную систему счисления
void movePointer(void);                                  // Функция смещения указателя на видеопамять
void setPointer(int, int);                               // Функция установки указателя на видеопамять
void scroll(void);                                                 // Функция прокрутки экрана
void clearWholeScreen(void);                             // Функция очистки всего экрана
void printRegisters(void);                               // Функция вывода на экран регистров
void fillWithZeroes(int, char*);                       // Функция дополнения десятичного числа нулями
void loading(void);                                              // Функция анимации загрузки
void clearScreen(void);                                  // Функция очистки экрана от скан-кодов
void printDivByZero(void);                             // Функция вывода ошибки при делении на 0


void intToBin(int scanCode, char *str)// Функция конвертации в 2-ичную систему счисления
{
        int rest, i, pos, value;
        pos = 8;
        value = scanCode;
        while (value > 1)
        {
                rest = value % 2;
                pos--;
                str[pos] = rest + '0';
                value = (value - rest) / 2;
        }
        pos--;
        str[pos] = value + '0';
        for (i = pos - 1; i > -1; i--)
                str[i] = '0'; str[8] = '\0';
}
void intToHex(int scanCode, char *str)// Функция конвертации в 16-ричную систему счисления
{
        char letters[6] = { 'A','B','C','D','E','F' };
        int rest, i, pos, value;
```

```
	pos = 2;
	value = scanCode;
	while (value > 15)
	{
		rest = value % 16;
		pos--;
		if (rest > 9)
		  str[pos] = letters[rest - 10];
		else
		  str[pos] = rest + '0';
		value = (value - rest) / 16;
	}
	pos--;
	if (value > 9)
	  str[pos] = letters[value - 10];
	else
	  str[pos] = value + '0';
	for (i = pos - 1; i > -1; i--)
		str[i] = '0';
	str[2] = '\0';
}
void printRegisters()//Функция вывода на экран регистров
{
	int reg;
	char strH[3], strB[9];
	reg = inportb(0x21);
	intToHex(reg, strH);
	intToBin(reg, strB);
	printToWithColor("MASTER:", 10, 0, 3);
	printToWithColor("Register mask (0x21):", 10, 0, 4);
	printToWithColor(strB, 10, 25, 4);
	printToWithColor("(", 10, 34, 4);
	printToWithColor(strH, 10, 35, 4);
	printToWithColor(")", 10, 37, 4);
	outportb(0x20, 0x0A);
	reg = inportb(0x20);
	intToHex(reg, strH);
	intToBin(reg, strB);
	printToWithColor("Register request (0x20):", 10, 0, 5);
	printToWithColor(strB, 10, 25, 5);
	printToWithColor("(", 10, 34, 5);
	printToWithColor(strH, 10, 35, 5);
	printToWithColor(")", 10, 37, 5);
	outportb(0x20, 0x0B);
	reg = inportb(0x20);
	reg = inportb(0x21);
	intToHex(reg, strH);
```

```
	intToBin(reg, strB);
	printToWithColor("Register service (0x20):", 10, 0, 6);
	printToWithColor(strB, 10, 25, 6);
	printToWithColor("(", 10, 34, 6);
	printToWithColor(strH, 10, 35, 6);
	printToWithColor(")", 10, 37, 6);
	reg = inportb(0xA1);
	intToHex(reg, strH);
	intToBin(reg, strB);
	printToWithColor("SLAVE :", 10, 42, 3);
	printToWithColor("Register mask (0xA1):", 10, 42, 4);
	printToWithColor(strB, 10, 67, 4);
	printToWithColor("(", 10, 76, 4);
	printToWithColor(strH, 10, 77, 4);
	printToWithColor(")", 10, 79, 4);
	outportb(0xA0, 0x0A);
	reg = inportb(0xA0);
	intToHex(reg, strH);
	intToBin(reg, strB);
	printToWithColor("Register request (0xA0):", 10, 42, 5);
	printToWithColor(strB, 10, 67, 5);
	printToWithColor("(", 10, 76, 5);
	printToWithColor(strH, 10, 77, 5);
	printToWithColor(")", 10, 79, 5);
	outportb(0xA0, 0x0B);
	reg = inportb(0xA0);
	intToHex(reg, strH);
	intToBin(reg, strB);
	printToWithColor("Register service (0xA0):", 10, 42, 6);
	printToWithColor(strB, 10, 67, 6);
	printToWithColor("(", 10, 76, 6);
	printToWithColor(strH, 10, 77, 6);
	printToWithColor(")", 10, 79, 6);
}
void clearScreen()//Функция очистки экрана от скан-кодов
{
	int i, j;
	for (i = 0; i < 80; i++)
	{
		for (j = 24; j > 6; j--)
		{
			v = (char far*)0xB8000000 + j * 160 + i * 2;
			*v = ' ';
		}
	}
	setPointer(0, 7);
}
```

```
void clearWholeScreen()//Функция очистки всего экрана
{
	int i, j;  v = (char far *)0xB8000000;
	for (i = 0; i < 25; i++)
	{
		for (j = 0; j < 80; j++)
		{
			v = (char far *)0xB8000000 + i * 160 + j * 2;
			*v = ' ';
			v++;
			*v = 0xf;
		}
	}
	setPointer(posX, posY);
}
void scroll()// Функция прокрутки экрана
{
	int i, j;
	char tmp;
	for (i = 7; i < 24; i++)
	{
		for (j = 0; j < 80; j++)
		{
			v = (char far *)0xB8000000 + (i + 1) * 160 + j * 2;
			tmp = *v;
			v = (char far *)0xB8000000 + i * 160 + j * 2;
			*v = tmp;
		}
	}
	for (j = 0; j < 80; j = j + 1)
	{
		v = (char far *)0xB8000000 + 24 * 160 + j * 2;
		*v = ' ';
	}
}
void setPointer(int _posX, int _posY)// Функция установки указателя на видеопамять
{
	v = (char far *)0xB8000000 + _posY * 160 + _posX * 2;
	posX = _posX;
	posY = _posY;
}
void movePointer()// Функция смещения указателя на видеопамять
{
	posX++;
	if (posX >= 80)
	{
		posX = 0;
```

```
		posY = posY + 1;
	}
	if (posY >= 25)
	{
		posY = 24;
		scroll();
	}
	setPointer(posX, posY);
}
void setColorFor(int topBorder, int downBorder, int color)// Функции установки цвета на ограниченную область
{
	int i, j;
	for (i = topBorder; i <= downBorder; i++)
	{
		for (j = 0; j < 80; j++)
		{
			v = (char far *)0xB8000000 + i * 160 + j * 2;
			v++;
			*v = color;
			v--;
		}
	}
	setPointer(posX, posY);
}
void printTo(char *str, int posX_, int posY_)// Функция вывода в консоль через видеопамять в произвольное
место
{
	int i;
	v = (char far *)0xb8000000 + posY_ * 160 + posX_ * 2;
	for (i = 0; str[i] != '\0'; i++)
	{
		*v = str[i];
		v += 2;
	}
	setPointer(posX, posY);
}
void printToWithColor(char *str, int color, int posX_, int posY_)
// Функция вывода в консоль через видеопамять в произвольное место с заданным цветом
{
	int i;
	v = (char far *)0xb8000000 + posY_ * 160 + posX_ * 2;
	for (i = 0; str[i] != '\0'; i++)
	{
		*v = str[i];
		v++;
		*v = color;
		v--;
```

```
                v += 2;
        }
        setPointer(posX, posY);
}
void print(char *str, int color)// Функция вывода в консоль через видеопамять
{
        int i;
        for (i = 0; str[i] != '\0'; i++)
        {
                *v = str[i];
                v++;
                *v = color;
                v--;
                movePointer();
        }
}
void fillWithZeroes(int value_, char *str)// Функция дополнения десятичного числа нулями
{
        int rest, i, pos = 3, value = value_;
        while (value >= 10)
        {
                rest = value % 10;
                pos--;
                str[pos] = rest + '0';
                value = (value - rest) / 10;
        }
        pos--;
        str[pos] = value + '0';
        for (i = pos - 1; i >= 0; i = i - 1)
        {
                str[i] = '0';
        }
        str[3] = '\0';
}
void interrupt newInt0()// Функция обработки прерывани Int0 (деление на 0)
{
        disable();
        printDivByZero();
        printRegisters();
        outportb(0x20, 0x20);
        enable();
        delay(1000);
        setvect(9, oldInt9);
        setvect(8, oldInt8);
        setvect(0, oldInt0);
   quitFlag = 1;
}
```

```
void interrupt newInt8()// Функция обработки прерывани Int8 (таймер)
{
	char str[3];
	int value;
	disable();
	value = timerCounter / 18;
	timerCounter = timerCounter + 1;
	printToWithColor("TIMER", 0x0B, 70, 0);
	fillWithZeroes(value, str);
	printToWithColor(str, 0x0B, 76, 0);
	printRegisters();
	outportb(0x20, 0x20);
	enable();
}
void interrupt newInt9()// Функция обработки прерывани Int9 (клавиатура)
{
	char str[3];
	int scanCode, zero = 0;
	int colors[6] = { 0x0F,0x0E,0x0D,0x0C,0x0B,0x0A };
	disable();
	scanCode = inportb(0x60);
	if (scanCode == 0x05)
		quitFlag = 1;
	if (scanCode == 0x04)
	{
		printDivByZero();
		zero = zero / zero;
	}
	if (scanCode == 0x02)
		clearScreen();
	if (scanCode == 0x03)
	{
		colorId++;
		if (colorId >= 6)
			colorId = 0;
		setColorFor(7, 25, colors[colorId]);
	}
	intToHex(scanCode, str);
	print(str, colors[colorId]);
	print(" ", 0x0F);
	printRegisters();
	outportb(0x20, 0x20);
	enable();
}
int main()
{
	clearWholeScreen();// Функция очистки всего экрана
```

```
	printToWithColor("724402-2  L4  Chernyavsky Y.A.  Podsistema prerivaniy", 0x04, 0, 0);
// Функция вывода в консоль через видеопамять в произвольное место с заданным цветом
	printToWithColor("1. Clear screen  2. Change scancode color  3. Divide by zero  4. Exit", 0x0E, 0, 1);
// Функция вывода в консоль через видеопамять в произвольное место с заданным цветом
	setColorFor(3, 8, 0x04); // Функции установки цвета на ограниченную область
	printRegisters();// Функция вывода на экран регистров
	oldInt9 = getvect(9);
	setvect(9, newInt9);
	oldInt8 = getvect(8);
	setvect(8, newInt8);
	oldInt0 = getvect(0);
	setvect(0, newInt0);
	while (quitFlag != 1)
	{
		loading();// Функция анимации загрузки
	}
	setvect(9, oldInt9);
	setvect(8, oldInt8);
	setvect(0, oldInt0);
	clearWholeScreen();// Функция очистки всего экрана
	return;
}
void loading()// Функция анимации загрузки
{
	int i;
	int colors[6] = { 0x02,0x03,0x04,0x05,0x06,0x07 };
	for (i = 0; i < 6; i++)
	{
		printToWithColor("|", colors[i], 76, 1);
		if (quitFlag)
			return;
		delay(100);
		printToWithColor("/", colors[i], 76, 1);
		delay(100);
		if (quitFlag)
			return;
		printToWithColor("-", colors[i], 76, 1);
		if (quitFlag)
			return;
		delay(100);
		printToWithColor("\\", colors[i], 76, 1);
		if (quitFlag)
			return;
		delay(100);
		printToWithColor("|", colors[i], 76, 1);
		if (quitFlag)
			return;
```

```
                delay(100);
                printToWithColor("/", colors[i], 76, 1);
                if (quitFlag)
                        return;
                delay(100);
                printToWithColor("-", colors[i], 76, 1);
                if (quitFlag)
                        return;
                delay(100);
                printToWithColor("\\", colors[i], 76, 1);
                if (quitFlag)
                        return;
                delay(100);
                if (i == 5)
                        i = 0;
        }
}
void printDivByZero()// Функция вывода ошибки при делении на 0
{
        short value;
        char port61;
        clearScreen();
        printToWithColor("DIVISION BY ZERO!", 0x04, 30, 13);
        value = 1193180 / 600;
        outp(0x42, (char)value);
        outp(0x42, (char)(value >> 8));
        port61 = inp(0x61);
        port61 |= 3;
        outp(0x61, port61);
        delay(500);
        port61 &= 0xFFFC;
        outp(0x61, port61);
}
```

5. Результаты работы программы

Работа прерываний 8 и 9 (IQR0 и IQR1 соответсвенно):

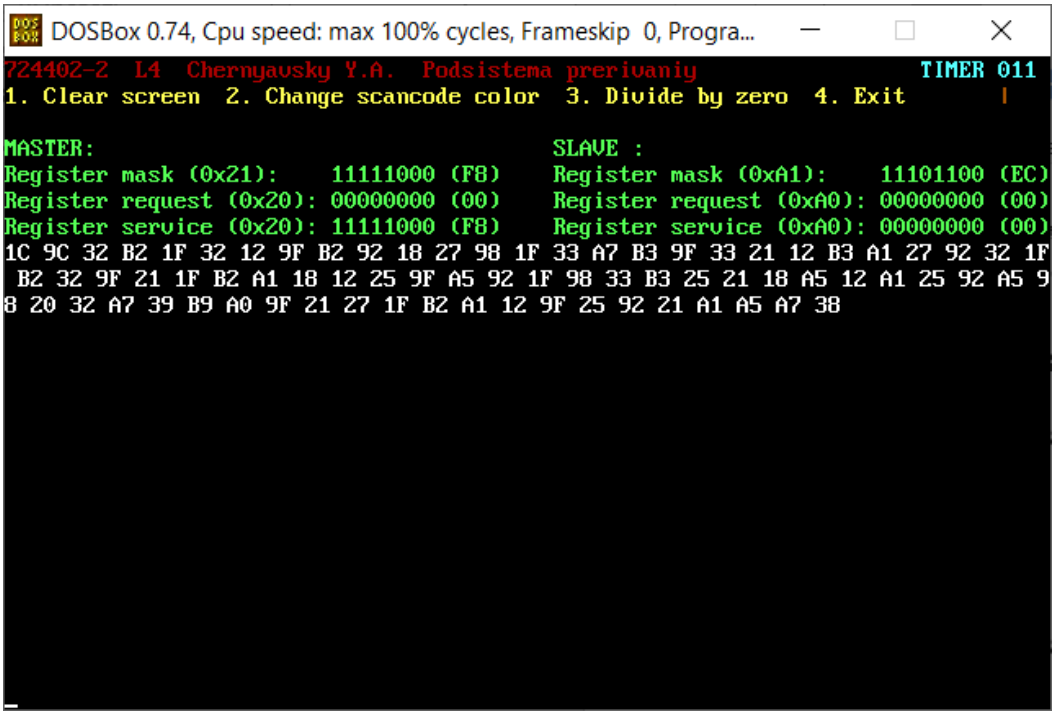

Рис. 5.1. – Работа прерываний 8 и 9

Работа прерывания 0:

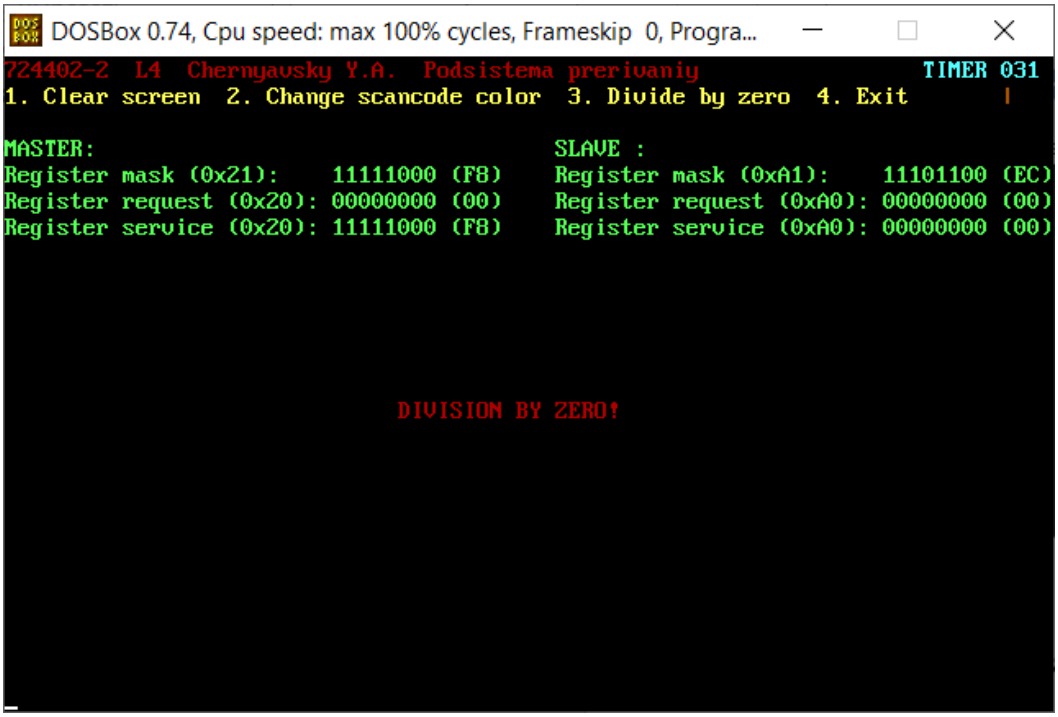

Рис. 5. 2 – Работа прерывания 0

6. Вывод

В ходе выполнения работы были получены практические навыки программирования контроллеров клавиатуры и системного таймера, был определён новый обработчик прерывания деления на ноль. Для того, чтобы связать адрес обработчика прерывания с номером прерывания, используется таблица векторов прерываний.