**Лабораторная работа №2**

**Защищенный режим процессора x86**

**Задание:** написать программу на языке ассемблера, работающую на нулевом уровне привилегий.

В программе необходимо реализовать:

1. загрузку данных с нужных секторов жесткого диска в память с помощью функций BIOS;
2. инициализацию программируемого контроллера прерываний;
3. корректной перевод процессора в защищенный режим;
4. вывод сообщения из защищенного режима, используя memory mapping;
5. подсчет доступной оперативной памяти, используя flat memory model;
6. обработчики прерываний системного таймера (IRQ0) и клавиатуры (IRQ1);
7. корректный возврат в реальный режим по нажатию клавиши enter;
8. вывод сообщения после возврата в реальный режим.

В данной лабораторной работе подразумевается, что программа работает без операционной системы. В связи с этим, нужно воспользоваться эмулятором аппаратного обеспечения. Рекомендуется воспользоваться эмулятором **QEMU**: [www.qemu.org](http://www.qemu.org). QEMU — это свободная кроссплатформенная программа с открытыми исходным кодом для эмуляции аппаратного обеспечения.

Рекомендуется использовать ассемблер **NASM**: [www.nasm.us](http://www.nasm.us). NASM — это свободный кроссплатформенный ассемблер для архитектуры Intel x86.

Справку по синтаксису языка ассемблера NASM можно найти в приложении.

**Выполнение лабораторной работы**

1. **Запуск программы без операционной системы**

При запуске компьютера, BIOS загружает 512 байт с загрузочного сектора жесткого или гибкого диска в оперативную память по адресу **0x7C00**. Если последние 2 байта равны **0xAA55**, тогда BIOS выполняет инструкцию **jmp 0x7C00**, передавая управление программе-загрузчику ОС. Программа, реализуемая в рамках данной лабораторной работы, по своей сути, является вариантом загрузчика ОС.

В качестве введения, рассмотрим следующий код (NASM).  
----------------------------------------------------------------------------

bits 16 ; указание NASM, что все команды далее 16-битные

org 0x7c00 ; указание NASM, что начальный адрес программы

; после загрузки в память будет 0000:0x7c00

cli ; запрет аппаратных прерываний

xor ax, ax

mov ds, ax ; для работы на эмуляторе, этой инструкции не нужно,

; но если запускать на реальном железе, то для доступа

; к данным, необходимо обнулить регистр ds.

; ПРЕДУПРЕЖДЕНИЕ:

; ЭТО ИНФОРМАЦИЯ БЫЛА ДАНА ДЛЯ ПОЛНОТЫ КАРТИНЫ,

; НЕ ЗАПУСКАЙТЕ СВОИ ПРОГРАММЫ НА РЕАЛЬНОМ КОМПЬЮТЕРЕ

; ВЫ МОЖЕТЕ ЕГО ПОВРЕДИТЬ!!!

mov si, msg ; загрузка адреса сообщения в регистр si

mov ah, 0x0e ; функция BIOS 'Write Character in TTY mode'

.loop:

lodsb

or al, al ; if al == 0

jz halt ; then

; else

int 0x10 ; запуск функции отображения видео BIOS

jmp .loop

halt:

hlt ; ожидание аппаратного прерывания (останов)

msg: db "Hello, World", 0

; Необходимы 2 байта сигнатуры загрузчика.

times 510 – ($ - $$) db 0 ; Справка по NASM находится в Приложении.

dw 0xaa55 ; Сигнатура

-------------------------------------------------------------------------

Пусть, данный код сохранен в файле boot.asm. Необходимо транслировать данный файл в бинарный формат. Делается это следующим образом:

nasm -f bin boot.asm -o boot.bin

Теперь можно запустить данную программу на виртуальной машине. Файл boot.bin будет представлять собой образ жесткого диска.

qemu-system-i386 -drive file=boot.bin,format=raw

Для выполнения лабораторной работы, 512 байт будет недостаточно. Недостающие сектора нужно загрузить самостоятельно. Будьте внимательны, загрузка бо́льшего количества секторов, чем есть на жестком диске, приведет к ошибке!

1. **Вызов функций BIOS. Загрузка данных с секторов жесткого диска в оперативную память**

Для доступа к функциям BIOS, в общем случае, устанавливается регистр **AH** в определенное значение с последующим вызовом нужного прерывания инструкцией **INT**.

Функции BIOS являются подпрограммами реального режима, их можно использовать только в реальном режиме процессора (режим Virtual 8086, который позволил бы вызывать функции BIOS в защищенном режиме, в данной работе не рассматривается).

Существуют большое количество функций BIOS, они разделены на группы. В лабораторной работе потребуются только две группы:

* **INT 0x10** — функции отображения видео;
* **INT 0x13** — функции доступа к хранилищу (к жестким или флоппи дискам).

Прежде всего, для вывода информации на экран в реальном режиме процессора, нам необходимо воспользоваться прерыванием **INT 0x10**. Для консольного вывода символа, в регистр **AH** должно быть помещено значение **0x0E**, а в регистр **AL** ASCII-код нужного символа.

Также может быть полезно очистить экран. Это делается с помощью того же прерывания **INT 0x10**.  
При установке видеорежима, экран монитора очищается. Установить видеорежим можно занеся значение **0x00** в регистр **AH** и номер видеорежима в регистр **AL**. Список видеорежимов можно найти здесь: <http://www.columbia.edu/~em36/wpdos/videomodes.txt>.   
Кроме того, можно установить курсор в нужное положение на экране — для этого в регистр AH заносится значение **0x02**, в регистр **DH** – номер строки, в регистр **DL** – номер столбца.

Как уже было сказано, 512 байт не хватит для написания лабораторной работы. Для загрузки дополнительных данных с секторов жесткого диска, необходимо вызвать специальную функцию BIOS. Существует два способа адресации сектора на диске, с которыми работает BIOS:

* CHS — Cylinder-Head-Sector;
* LBA — Logical Block Addressing.

Адресация CHS основана на геометрии диска, где адрес сектора состоит из трех частей – цилиндра, головки и сектора. Данный способ устарел, он позволяет адресовать до 8 ГБ, что в настоящее время является достаточно малой величиной.

Все современные (и не только современные) контроллеры жестких дисков поддерживают адресацию LBA, которая присваивает каждому сектору собственный номер. Нумерация секторов начинается с нуля. Ранее, для задания адреса сектора, LBA использовала 48 бит, что позволяло адресовать до 128 ПБ, теперь используется 64 бит с лимитом в 8 зеттабайт.

Прерывание **INT 0x13** предоставляет сервис чтения данных с секторов диска с иcпользованием LBA.  
Для чтения сектора, перед вызовом прерывания **INT 0x13** нужно: занести в регистр **AH** значение **0x42**, в регистр **DL** — номер диска, в **DS:SI** — указатель вида *segment:offset* на структуру **DAP (Disk Address Packet).***Примечание: при передаче управления загрузчику, BIOS заносит в регистр* ***DL*** *номер диска, с которого был считан загрузчик.*

Структура **DAP** имеет следующий вид:

|  |  |
| --- | --- |
| **Размер элемента** | **Описание** |
| 1 байт | Размер структуры в байтах (это поле всегда равно 16) |
| 1 байт | Зарезервировано, всегда равно 0 |
| 2 байта | Количество секторов для чтения |
| 4 байта | Указатель вида *segment:offset* на область памяти, в которую будут записаны данные секторов |
| 8 байт | Номер сектора, с которого будет начато чтение |

Прерывание, в случае успеха, занесет в регистр **AH** значение **0** и сбросит флаг **CF.**В случае ошибки — установит флаг **CF**.

*О создании структур в NASM см. Приложение.*

Следует отметить, что данный сервис BIOS является частью **INT 13h Extensions**. Поддержка данных расширений присутствует во всех современных BIOS и контроллерах ЖД, но было бы хорошим тоном проверить наличие данных расширений. Для этого, перед вызовом прерывания **INT 0x13** нужно занести в регистр **AH** значение **0x41**, в регистр **DL** —номер диска, в регистр **BX** — магическое число **0x55AA**.   
По завершении прерывания флаг **CF** будет сброшен, если присутствует поддержка расширений **INT 13h**. В случае отсутствия расширений — устанавливается флаг **CF**.

1. **Инициализация программируемого контроллера прерываний.**

Программируемый контроллер прерываний (Programmable Interrupt Controller — PIC, 8259A Microcontroller, советский аналог — К1810ВН59) отвечает за прием запросов прерываний от различных устройств, их хранение в ожидании обработки, выделение наиболее приоритетного из одновременно присутствующих запросов и выдачу вектора прерывания. Контроллер называется «программируемым», т. к. режимы его работы устанавливаются программно. PIC может обрабатывать до 8 запросов прерываний (Interrupt Request — IRQ), что оказалось недостаточно, поэтому используется каскадное соединение двух PIC. Один из них называется ведущим (master/primary PIC), который в свою очередь соединен с процессором. Другой называется ведомым (slave PIC), он соединен с ведущим по ножке IRQ2.  
PIC обрабатывает **только** аппаратные прерывания.

Векторы прерываний находятся в таблице векторов прерываний (Interrupt Vector Table — IVT). Всего в IVT находится 256 адресов обработчиков прерываний (Interrupt Service Routines — ISR) в формате *сегмент:смещение*. В реальном режиме процессора, IVT располагается в первых 1024 байтах физической памяти (с адреса **0x0** по **0x3FF**). В защищенном режиме процессора используется таблица дескрипторов прерываний (Interrupt Descriptor Table — IDT), каждый дескриптор в которой содержит базовый адрес ISR, селектор сегмента памяти, один зарезервированный байт и байт атрибутов (см. семинары).

Опишем процесс возникновения и обработки прерывания системного таймера (IRQ0).  
Возникновение прерывания

1. системный таймер информирует PIC о прерывании с помощью изменения напряжения с низкого уровня на высокий на линии IR0;
2. PIC устанавливает бит прерывания IRQ0 в регистре запросов прерываний (Interrupt Request Register — IRR). В случае системного таймера будет установлен бит №0;
3. прерывание может быть замаскированно, в таком случае оно будет проигнорировано.  
   Для того, чтобы выяснить является ли прерывание замаскированным, PIC проверяет регистр замаскированных прерываний (Interrupt Mask Register — IMR). Если бит прерывания (в данном случае бит №0) установлен — прерывание замаскированно;
   1. если прерывание не замаскировано, PIC определяет есть ли более приоритетные прерывания для обработки. Если такие прерывания присутствуют, тогда текущий запрос игнорируется до тех пор, пока не будет обработано прерывание более высокого приоритета;
   2. если прерывание не замаскировано и нет более приоритетных прерываний, PIC переходит к следующему шагу;
4. PIC информирует процессор о том, что произошло прерывание посредством ножки INTA (Interrupt Acknowledge), маскируются все остальные прерывания до получения команды завершения прерывания (End of Interrupt — EOI).

Поступление прерывания на процессор

1. процессор завершает выполнение текущей инструкции;
2. процессор проверяет флаг Interrupt Flag (IF) в регистре флагов;
   1. если флаг IF установлен, процессор подтверждает готовность обработать прерывание посредством ножки INTR;
   2. Если флаг IF сброшен, запрос на прерывание игнорируется;
3. PIC получает подтверждение от процессора через ножку INTR;
4. PIC выставляет смещение относительно базового адреса IVT (IDT в защищенном режиме) на ножки D0-D7 (шина данных, 1 байт);
   1. смещение устанавливается во время инициализации контроллера, в реальном и защищенном режиме используются разные смещения (в виду наличия 32 исключений в защищенном режиме).
5. PIC выставляет номер прерывания на шину данных;
6. PIC устанавливает нужный бит в регистре In Service Register (ISR), который содержит информацию о том, какие прерывания в данный момент обрабатываются. В случае системного таймера в регистре ISR будет установлен бит №0.

Прерывание

1. процессор прерывает выполнение текущей программы, помещает значения регистров CS, EIP и регистра флагов на стек.
2. Процессор использует смещение, считанное с шины данных от PIC;
   1. Реальный режим
      1. процессор использует смещение относительно базового адреса IVT;
      2. процессор добавляет к смещению номер прерывания, переходя таким образом к нужному вектору прерывания в таблице IVT;
      3. процессор загружает адрес обработчика прерывания в CS:IP;
      4. выполняется обработчик прерывания;
   2. Защищенный режим
      1. процессор использует смещение относительно базового адреса IDT;
      2. процессор, согласно номеру прерывания, переходит к соответствующему шлюзу IDT;
      3. из шлюза IDT считывается селектор и загружается в регистр CS;
      4. из шлюза IDT считывается смещение и загружается в регистр EIP;
      5. выполняется обработчик прерывания.

*Примечание: в лабораторной работе не используется страничная организация памяти, поэтому в описании прерывания в защищенном режиме не описан шаг преобразования линейного адреса в физический.*

Регистры IRR и ISR контроллера прерываний не доступны для программы. Однако к регистрам IMR/Data Register и Command Register можно обращаться. Command Register, Data Register и Status Register используются для перепрограммирования PIC, IMR (он же Data Register) — для маскирования игнорируемых прерываний. Кроме того, посредством Command Register посылается команда завершения прерывания (EOI).

Данные регистры находятся внутри PIC, т. е. не являются частью центрального процессора. Таким образом, для того, чтобы модифицировать их содержимое необходимо использовать порты ввода-вывода.

**Порт ввода-вывода (I/O Port)** — технический термин для определенного адреса на шине ввода-вывода. Эта шина обеспечивает связь с устройствами в фиксированном порядке и размерах данных.

Для записи в порт используется инструкция **out**, для чтения — инструкция **in:**

in <регистр>, <номер-порта>

out <номер-порта>, <регистр>

где <номер-порта> это либо 8-рязрядная **константа**, либо регистр **DX** (в случае 16-разрядного адреса).

Таблица соответствия IO портов регистрам PIC.

|  |  |
| --- | --- |
| Адрес | Описание |
| **0x20** | Command Register и Status Register **ведущего** контроллера |
| **0x21** | IMR/Data Register **ведущего** контроллера |
| **0xA0** | Command Register и Status Register **ведомого** контроллера |
| **0xA1** | IMR/Data Register **ведомого** контроллера |

**Важно отметить**, что Command Register и Status Register — это **разные** регистры, хотя и разделяют один порт. Command Register является регистром **только для записи**, в том время как Status Register — **только для чтения**. Это критически важно, т. к. PIC различает к какому регистру осуществляется доступ в зависимости от того какая линия (записи или чтения) в данный момент используется.

**Процесс перепрограммирования PIC**

В начале выполнения вашей программы, PIC уже настроен на работу в реальном режиме процессора. Для перехода в защищенный режим процессора, необходимо перепрограммировать PIC. Данный процесс осуществляется с помощью слов управления инициализацией (Initialization Control Words — ICW).   
Если в системе присутствуют несколько PIC (как в нашем случае), необходимо передать каждому PIC нужные ICW.

**ICW 1** — главное управляющее слово, используемое при инициализации PIC. Это байтовое значение, которое должно быть помещено в Command Register PIC.

Формат ICW 1:

|  |  |  |
| --- | --- | --- |
| № бита | Значение | Описание |
| 0 | IC4 | Если бит установлен — ожидается ICW 4. |
| 1 | SNGL | Если бит установлен — в системе только один PIC, иначе имеется каскадное соединение с ведомым контроллером. В случае каскадного соединения ожидается ICW 3. |
| 2 | 0 | Игнорируется на архитектуре x86, необходимо сбросить этот бит. |
| 3 | LTIM | Если бит установлен — прерывание возникает по установившемуся уровню сигнала (level triggered), иначе — по фронту сигнала. По умолчанию используется второй вариант, бит нужно сбросить. |
| 4 | 1 | Бит инициализации, ожидается ICW 2. |
| 5 | 0 | - |
| 6 | 0 | - |
| 7 | 0 | - |

В нашем случае в Command Register контроллеров должно быть передано значение **00010001 (0x11).**

**ICW 2** — это управляющее слово используется для установления смещения относительно базового адреса IVT или IDT.

|  |  |  |
| --- | --- | --- |
| № бита | Значение | Описание |
| 0-2 | 0 | - |
| 3-7 | T3 - T7 | Смещение относительно базового адреса IVT/IDT |

После передачи ICW 1 необходимо сразу передать ICW 2, в противном случае результат работы PIC не определен. Вероятнее всего, это приведет к несоответствию прерывания и его обработчика.

Кроме того, в отличие от ICW 1, которое помещалось в Command Register, ICW 2 помещается в Data Register (IMR).

Значение ICW 2 для ведущего контроллера в защищенном режиме очевидно, оно следует из того, что первые 32 прерывания, являющееся исключениями, зарезервированы. Значение ICW 2 для ведомого в таком случае ясно из того, что ведущий контроллер обрабатывает всего 8 различных прерываний.

**ICW 3** — управляющее слово, которое указывает контроллерам посредством какой линии IRQ они образуют каскадное соединение. **Важно**, что эти ICW различаются по формату для ведущего и ведомого контроллеров.

Формат ICW 3 для **ведущего** контроллера

|  |  |  |
| --- | --- | --- |
| № бита | Значение | Описание |
| 0-7 | S0-S7 | Определяет к какой линии IRQ присоединен ведомый контроллер |

Формат ICW 3 для **ведомого** контроллера

|  |  |  |
| --- | --- | --- |
| № бита | Значение | Описание |
| 0-2 | ID0 | **Номер** линии IRQ ведущего контроллера, к которой присоединен ведомый контроллер |
| 3-7 | 0 | **-** |

Т. к. в ICW 1 указан бит присутствия каскадного соединения, необходимо передать ICW 3.

Контроллеры PIC получают сигналы о прерываниях различных устройств (в том числе от ведомого контроллера) посредством линий IRQ IR0-IR7 (8 ножек), поэтому в ICW 3 для ведущего контроллера, номер линии IRQ, использующийся для создания каскадного соединения, задается соответствующим битом.

В тоже время, ведомый контроллер использует 3 ножки CAS0-CAS2 для коммуникации с ведущим контроллером. Поэтому, в ICW 3 для ведомого контроллера используется номер линии IRQ в бинарной форме.

На архитектуре x86 ведомый контроллер всегда присоединен к линии IRQ2 ведущего контроллера.

ICW 3 передается в Data Register PIC.

**ICW 4** — управляющее слово, которое имеет множество различных полей. Это связано с тем, что PIC изначально был спроектирован как микроконтроллер общего назначения еще до появления 8086. Таким образом, PIC содержит множество режимов работы, спроектированных для разных систем.

Для работы PIC с x86 нужно установить бит №0 данного управляющего слова (само значение имеет размер 1 байт).

ICW 4 передается в Data Register PIC.

После передачи ICW 4 PIC успешно перепрограммирован.

**Отметим**, что при обратном переходе из защищенного режима в реальный, необходимо снова перепрограммировать PIC. В этом случае ICW 2 будет иметь другое значение. Для **ведущего** контроллера смещение будет равно 8, в то время как для **ведомого** — 0x70. Кроме того, требуется восстановить значения регистров замаскированных прерываний.

Т. к. в лабораторной работе обрабатываются только прерывания от системного таймера и клавиатуры, нужно замаскировать остальные прерывания в защищенном режиме.

Также, на время перепрограммирования контроллера прерываний необходимо **запретить** все маскируемые и немаскируемые прерывания. Немаскируемые прерывания (Non Maskable Interrupt — NMI) — это аппаратные прерывания, которые направляются напрямую к процессору. NMI могут возникнуть при наличии ошибок RAM или при критических неисправностях аппаратного обеспечения. Немаскируемые прерывания можно запретить, отправив байт с установленным 7-ым битом в порт CMOS (**0x70**).

После перехода в защищенный режим маскируемые прерывания можно разрешить.

1. **перевод процессора в защищенный режим**

Для перевода процессора в защищенный режим нужно:

1. запретить все прерывания;
2. загрузить GDT;
3. перепрограммировать PIC для защищенного режима;
4. загрузить IDT;
5. открыть линию A20 (бит №1 в порту **0x92**);
6. установить флаг защищенного режима;
7. совершить FAR jmp в сегмент кода.  
   *Примечание: jmp dword code\_selector:offset*

Информацию о GDT, IDT и линии A20 см. в конспектах семинаров.

1. **вывод сообщения из защищенного режима, используя memory mapping**

Для вывода информации на экран в защищенном режиме используется memory-mapping видео памяти VGA на RAM. Начальный адрес — **0xB8000**, размер области маппинга видео памяти равен 32 KiB. Каждые 2 байта этой памяти соответствуют одному символу на дисплее. Самый первый символ находится в левом верхнем углу экрана, последний — в правом нижнем. Первый из данных 2 байт кодирует атрибут символа, второй — код ASCII.

Напишите код вывода сообщения на экран о завершении перехода в защищенный режим.

1. **подсчет доступной оперативной памяти, используя flat memory model**

Для подсчета доступной оперативной памяти будем проверять каждую ячейку ОП. Для того, чтобы иметь возможность покрыть весь диапазон в 4 GiB, необходим сегмент данных, дескриптор которого имеет корректные значения байтов атрибутов.

*Примечание: когда процессор выполняет инструкцию записи в ячейку памяти, он не запишет новое значение напрямую в RAM сразу. Сначала новое значение помещается в кэш процессора. Для того чтобы записать новые данные в ОП и сбросить кэш, существует инструкция* ***wbinvd****.*

Для проверки, можно изменять количество доступной ОП на виртуальной машине. При запуске QEMU добавьте опцию **-m** <size>**M**, где M — это MiB

1. **обработчики прерываний системного таймера (IRQ0) и клавиатуры (IRQ1)**

Напишите 2 обработчика прерывания. В обработчике прерывания системного таймера выводите количество произошедших прерываний таймера на экран. В конце кода обработчика обязательно передайте команду End of Interrupt (EOI) контроллеру прерываний (почему это нужно см. в пункте 2) Команда EOI посылается в Command Register PIC. Команда EOI — это Operation Control Word 2 контроллера прерываний. Мы не будем приводить формат данного управляющего слова, т.к. это не относится напрямую к лабораторной работе. Значение OCW 2 для Non-Specific EOI равно **0x20**.

Также требуется реализовать обработчик прерывания клавиатуры. У клавиатуры имеется свой контроллер, который доступен по порту **0x60** и **0x61**. Скан-код нажатой клавиши можно считать из порта **0x60**. Перед отправкой EOI в коде обработчика прерывания клавиатуры нужно оповестить контроллер клавиатуры о том, что был обработан очередной скан-код, это делается сбрасыванием бита №7 в порту **0x61**.

Обработчик должен выводить на экран символ нажатой клавиши. По нажатию клавиши Backspace — затирать ранее введенные символы. По нажатию клавиши enter должен устанавливаться флаг выхода из защищенного режима процессора.

1. **возврат в реальный режим**Для возврата в реальный режим в первую очередь нужно:
2. перейти в 16-разрялный сегмент кода с помощью FAR jmp;
3. запретить все прерывания;
4. сбросить флаг защищенного режима;
5. совершить FAR jmp к коду реального режима;
6. перепрограммировать PIC для реального режима, загрузить IVT инструкцией lidt;
7. разрешить все прерывания;
8. **вывод сообщения после возврата в реальный режим**

См. пример в пункте **0**.