

Министерство образования Республики Беларусь  
Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ  
КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №1

По дисциплине «Архитектура вычислительных систем»  
По теме «Защищённый режим 32-разрядных процессоров»

Выполнил:  
студент гр. 953501  
Кореневский С. А.

Проверил:  
старший преподаватель  
Шиманский В.В.

Минск 2021

## Содержание

1. Цель работы .....	3
2. Постановка задачи.....	4
3. Теоретические сведения .....	5
Обзор режимов 32-разрядных микропроцессоров (IA-32) .....	5
Адресация памяти в защищенном режиме. ....	5
Защита по привилегиям .....	10
Переключение в защищенный режим и обратно.....	14
4. Программная реализация .....	17
5. Выводы .....	20
Список литературы .....	21
Приложение 1. Текст программы.....	22

## **1. Цель работы**

Изучить особенности защищенного режима процессора. Получить практические навыки по программированию переключения процессора из реального в защищенный режим и обратно.

## **2. Постановка задачи**

Написать программу, переключающую процессор в защищенный режим, выводящую на экране сообщение и затем возвращающую процессор в реальный режим. Чтобы сравнить работу процессора в защищенном и реальном режиме, посчитаем ряд до сходимости и выведем время работы в каждом из режимов.

### 3. Теоретические сведения

#### Обзор режимов 32-разрядных микропроцессоров (IA-32)

32-разрядные процессоры Intel 80386, 80486 и Pentium с точки зрения рассматриваемых в данном разделе вопросов имеют аналогичные средства, поэтому для краткости в тексте используется термин "процессор i386", хотя вся информация этого раздела в равной степени относится к трем моделям процессоров фирмы Intel.

Процессор i386 имеет два режима работы - реальный (real mode, R-Mode) и защищенный (protected mode, P-Mode).

**Реальный режим.** Реальный режим – это режим, в котором процессор работает как быстрый процессор 8086, но позволяет пользоваться большинством своих технологий (MMX / SSE / SSE2, 32-разрядными регистрами общего назначения, регистрами управления и отладки и пр.). После аппаратного сброса процессор переходит в этот режим и начинает выполнять программную инициализацию из BIOS-а. Реальный режим в современных процессорах предназначен для запуска компьютера и подразумевается, что операционная система будет работать в защищённом режиме (поэтому оптимизация по производительности для процессоров IA-32 производится для защищённого режима).

В реальном режиме не доступны основные достоинства процессора – виртуальная память, мультизадачность, уровни привилегий, работа с кэшами, буферами TLB, буфером ветвлений и некоторыми другими технологиями, обеспечивающими высокую производительность.

**Защищённый режим.** Как утверждает Intel, это "родной" (native) режим 32-разрядного процессора. В защищённый режим процессор надо переводить специальными операциями над системными регистрами и войти в этот режим процессор может только из реального режима. При работе в защищённом режиме процессор контролирует практически все действия программ и позволяет разделить операционную систему, драйвера и прикладные программы разными уровнями привилегий. Благодаря этому ОС может ограничить области памяти, выделяемой программам, запретить или разрешить ввод/вывод по любым адресам, управлять прерываниями и многое другое. При попытке программы выйти за допустимый диапазон адресов памяти, выделенной ей, либо при обращении к "запрещённым" для неё портам процессор будет генерировать исключения – специальный тип прерываний. Грамотно оперируя исключениями, операционная система может контролировать действия программ, организовать систему виртуальной памяти, мультизадачность и другие программные технологии.

В защищённом режиме максимально доступны все ресурсы процессора. Например, в R-Mode максимальный диапазон адресов памяти ограничен одним мегабайтом, а в защищённом режиме он расширен до 4 Гб для процессоров 386 и 486 и 64 Гб для Pentium-ов.

#### Адресация памяти в защищенном режиме.

Физическое адресное пространство процессора i386 составляет 4 Гбайта, что определяется 32-разрядной шиной адреса. Физическая память является линейной с адресами от 00000000 до FFFFFFFF в шестнадцатеричном представлении. Процессор i80386 в защищённом режиме использует трёхступенчатую схему преобразования адреса.

Программы используют *логический адрес (виртуальный адрес)*, состоящий из селектора и смещения. Это первая ступень в схеме преобразования адресов. Смещение хранится в соответствующем поле команды, а номер сегмента – в одном из шести сегментных регистров процессора (CS, SS, DS, ES, FS или GS), каждый из которых является 16-битным. Компонента смещения является 32-азрядной, т. к. допустимый размер сегмента значительно превышает 64 Кбайта. Селектор – это индекс, с помощью которого процессор извлекает из специальной таблицы базовый адрес сегмента. В реальном режиме мы имеем дело с сегментным адресом и смещением, а в защищенном – с селектором и смещением.

Получение из логического адреса 32-разрядного *линейного адреса* с помощью механизма сегментации является второй ступенью в схеме преобразования адресов.

Далее с помощью страничного механизма линейный адрес преобразуется в 32-разрядный физический адрес. Это третья ступень в схеме преобразования адресов. Страничный механизм включается установкой специального бита PG в регистре CR0 при помощи привилегированной команды. При отключенном страничном механизме линейный адрес является физическим адресом сегмента.

Сначала рассмотрим механизм преобразования логического адреса в линейный при отключенном механизме управления страницами.

Рассмотрим структуру селектора. На самом деле не все 16 бит селектора используются для индексации по таблице базовых адресов. Формат селектора адреса приведен на рис.1.

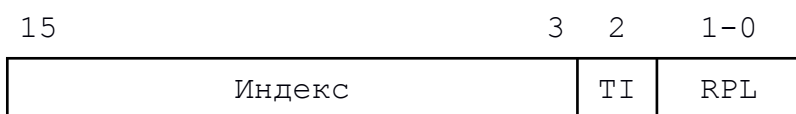


Рис. 1 – Полный формат селектора адреса

В качестве индекса выступают старшие 13 бит. Два младших бита используются системой защиты памяти. На рисунке они обозначены как RPL (Requested Privilege Level). Это поле является запрошенным программой уровнем привилегий и будет обсуждено позже. Поле TI (Table Indicator) состоит из одного бита. Если этот бит равен нулю, для преобразования адреса используется так называемая глобальная дескрипторная таблица GDT (Global Descriptor Table), в противном случае – локальная дескрипторная таблица LDT (Local Descriptor Table). Глобальная дескрипторная таблица предназначена для описания сегментов операционной системы и сегментов межзадачного взаимодействия, то есть сегментов, которые в принципе могут использоваться всеми процессами, а локальная дескрипторная таблица – для

сегментов отдельных задач. Таблица GDT одна, а таблиц LDT должно быть столько, сколько в системе выполняется задач, но не более чем 8К ( $2^{13}$ ). При этом активной в каждый момент времени может быть только одна из таблиц LDT. Разрядность поля индекса определяет максимальное число глобальных и локальных сегментов задачи – по 8К ( $2^{13}$ ) сегментов каждого типа, всего 16 К. С учетом максимального размера сегмента – 4 Гбайта – каждая задача при чисто сегментной организации виртуальной памяти работает в виртуальном адресном пространстве в 64 Тбайта.

Процесс преобразования логического адреса в линейный приведен на рис. 2.

Рис. 2 – Преобразование логического адреса в линейный

Рассмотрим сначала использование таблицы GDT. Значение из поля индекса селектора используется в качестве индекса в таблице GDT для выборки 32-разрядного базового адреса. Этот базовый адрес складывается со второй компонентой логического адреса – смещением. В результате получается 32-разрядный линейный адрес.

В дескрипторных таблицах хранятся так называемые дескрипторы сегментов. Дескриптор представляет собой 8-байтную структуру, которая содержит базовый адрес описываемого сегмента, предел сегмента и права доступа к сегменту. Формат дескриптора приведен на рис. 3.

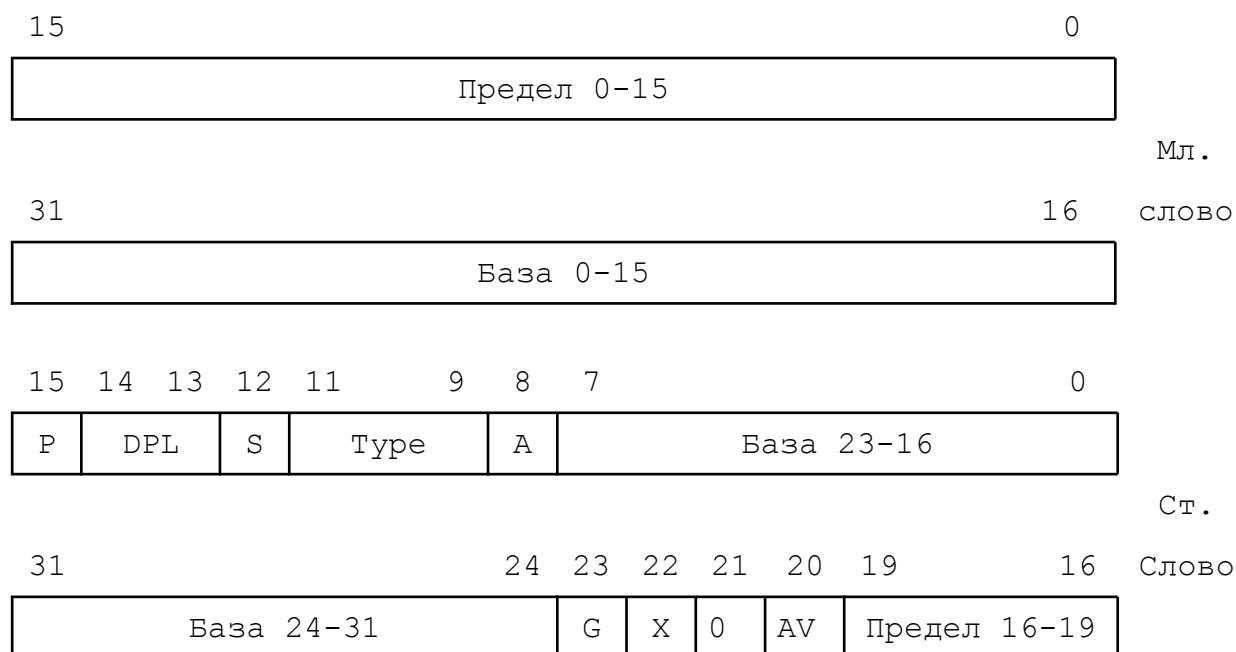


Рис. 3 – Дескрипторы для процессора i80386

Рассмотрим элементы дескриптора.

*Адрес сегмента* – также называется базовым адресом, – 32-разрядный адрес области памяти, с которой начинается сегмент.

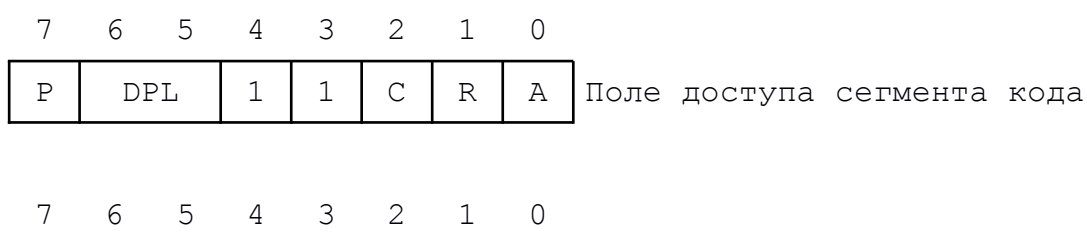
*Предел сегмента* – предельное значение смещения в сегменте; также можно рассматривать предел как размер сегмента минус один элемент размера – байт или страницу, смотря в чём измеряется сегмент.

Поля *A*, *Typ*, *S*, *DPL* и *P* образуют так называемое *поле доступа*. Биты *A* и *P* предназначены для организации виртуальной памяти. Бит *P* называется *битом присутствия* сегмента в памяти. Для тех сегментов, которые находятся в физической памяти, этот бит должен быть установлен в 1. Если программе понадобится память, то она сохранит содержимое какого-либо сегмента на диск и сбросит бит *P*. Если любая программа в дальнейшем обратится к этому сегменту, то процессор сгенерирует исключение не присутствующего сегмента и запустит обработчик этой ситуации, который должен будет подгрузить содержимое сегмента с диска и установить бит *P*. После этого управление снова передаётся команде, обратившейся к этому сегменту (производится повторное выполнение команды, вызвавшей сбой) и работа программы будет продолжена. Бит *P* устанавливается и сбрасывается программами, сам процессор его только считывает. Бит *A* (Accessed) – *бит доступа* в сегмент. Этот бит показывает, был ли произведен доступ к сегменту, описываемому этим дескриптором, или нет. Если процессор обращался к сегменту для чтения или записи данных, или для выполнения кода, размещённых в нём, то бит *A* будет установлен (равен 1), иначе – сброшен (0). С помощью бита *A* операционная система может определить, использовался ли за последнее время этот сегмент или нет и предпринять какие-либо действия. Бит *A* процессором только устанавливается, сбрасывать его должна операционная система. При создании нового дескриптора подразумевается, что бит *A* будет равен 0 (т.е. обращений к этому сегменту ещё не было).

Бит *S* (System) – определяет системный объект. Если этот бит установлен, то дескриптор определяет сегмент кода или данных, а если сброшен, то системный объект (например, сегмент состояния задачи, локальную дескрипторную таблицу, шлюз).

Поле *DPL* (Descriptor Privilege Level) – *уровень привилегий*, который имеет объект, описываемый данным дескриптором. Это двухбитовое поле, в него при создании дескриптора записывают значения от 0 до 3, определяющее уровень привилегий.

Поле *Type* - трёхбитовое поле, которое интерпретируется по-разному, в зависимости от типа сегмента (см. рис. 4.).





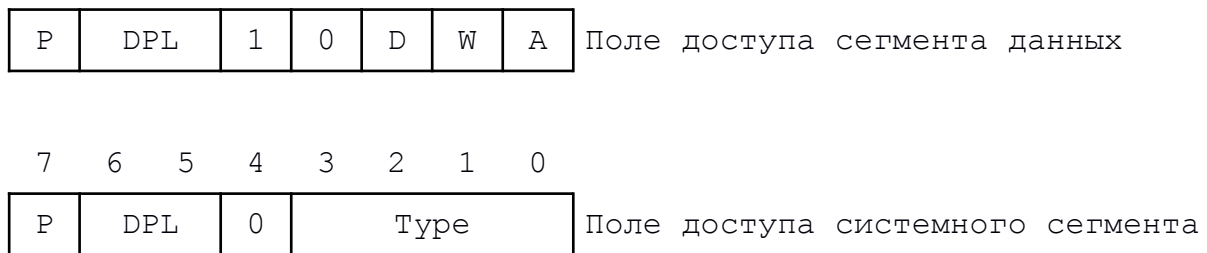


Рис. 4 – Формат поля доступа дескриптора

Для несистемных сегментов старший бит поля Type (3-й бит в байте прав доступа) имеет следующее значение: если он равен 1, то это сегмент кода, иначе это сегмент данных.

Поле доступа сегмента кода имеет битовые поля *C* и *R*. Поле *C* (conformance) указывает, согласован ли этот сегмент (бит = 1) или нет (бит = 0). Этот бит будет рассмотрен позже. Поле *R* указывает, разрешено ли чтение сегмента кода (бит = 1) или только выполнение (бит = 1).

Поле доступа сегмента кода имеет битовые поля *D* (direction) и *W*. Поле *W* называется битом разрешения записи в сегмент. Если этот бит установлен в 1, то наряду с чтением возможна и запись в данный сегмент. Поле *D* задает направление расширения сегмента. Обычный сегмент данных расширяется в область старших адресов (расширение вверх). Если же в сегменте расположен стек, расширение происходит в обратном направлении – в область младших адресов (расширение вниз). Для сегментов, в которых организуются стеки, необходимо устанавливать поле *D* равным 1.

Бит *G* (granularity) – гранулярность сегмента, т.е. единицы измерения его размера. Если бит *G* = 0, то сегмент имеет байтную гранулярность, иначе – страничную (одна страница – это 4Кб).

Бит *X* указывает разрядность выполняемых команд. Если этот бит установлен в 1, используются 32-разрядные команды, если сброшен в 0 – 16-разрядные.

Бит *AVL* используется системным программным обеспечением.

Расположение дескрипторных таблиц определяется регистрами процессора GDTR, LDTR. Регистры GDTR – 6-байтный, он содержит 32 бита линейного базового адреса дескрипторной таблицы и 16 бит предела таблицы. Формат регистра GDTR приведен на рис. 5.

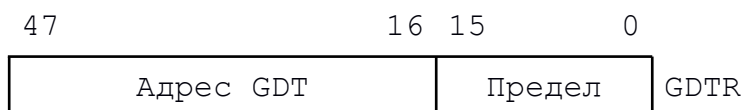


Рис. 5 – Формат регистра GDTR

Перед переходом в защищенный режим программа должна создать в оперативной памяти таблицу GDT и загрузить регистр GDTR при помощи

команды LGDT src, где src указывает на область памяти с адресом и пределом таблицы GDT.

В случае, когда селектор задачи указывает на таблицу LDT, виртуальный адрес преобразуется в физический адрес аналогичным образом, но для доступа к самой таблице LDT добавляется еще один этап, так как в процессоре регистр LDTR указывает на размещение таблицы LDT не прямо, а косвенно. Сам регистр LDTR имеет размер 16 бит и содержит селектор дескриптора таблицы GDT, который описывает расположение этой таблицы в физической памяти. Поэтому при доступе к элементу физической памяти через таблицу LDT происходит двукратное преобразование виртуального адреса в физический, причем оба раза по описанной выше схеме. Сначала по значению селектора LDTR определяется физический адрес дескриптора из таблицы GDT, описывающего начало расположения таблицы LDT в физической памяти, а затем с помощью селектора задачи вычисляется смещение в таблице LDT и определяется физический адрес нужного дескриптора. Далее процесс аналогичен преобразованию виртуального адреса с помощью таблицы GDT.

Следует заметить, что нулевой дескриптор, т.е. определённый в самом начале GDT, процессор не использует, обращение к такому дескриптору могло бы быть, когда поле Index селектора равно 0. Если всё же в программе встречается обращение к нулевому дескриптору, то процессор сгенерирует исключение и не позволит доступ к такому дескриптору. Первый дескриптор GDT описывает саму таблицу GDT.

Рассмотрим теперь страничный механизм. Преобразование линейного адреса в физический иллюстрируется рис. 6. Процесс вычисления адреса страницы часто называют трансляцией страниц. Старшие 10 бит линейного адреса используются как индекс в таблице, называемой каталогом таблиц страниц. Расположение каталога таблиц страниц в физической памяти определяется содержимым системного регистра процессора CR3.

Каталог таблиц страниц содержит дескрипторы таблиц страниц, определяющие физический адрес таблиц страниц. В каталоге всего может быть 1024 дескриптора. Самых же каталогов может быть сколько угодно, но в каждый момент времени используется только один – тот, на который указывает регистр CR3.

Следующие 10 бит линейного адреса предназначены для индексации таблицы страниц, выбранной с помощью старших 10 бит адреса. Таблица страниц содержит 1024 дескриптора, определяющих физические адреса страниц памяти. Размер одной страницы составляет 4 Кбайта, т. е. 4096 байт.

Младшие 12 бит линейного адреса указывают смещение к адресуемому байту внутри страницы.

Рис.6 – Преобразование линейного адреса в физический

## **Защита по привилегиям**

Основой защищённого режима являются уровни привилегий (кольца защиты). **Уровень привилегий** – это степень использования ресурсов процессора. Всего таких уровней четыре, и они имеют номера от 0 до 3. Уровень номер 0 – самый привилегированный. Когда программа работает на этом уровне привилегий, ей "можно всё". Обычно такими привилегиями обладает ядро операционной системы. Уровень 1 – менее привилегированный и запреты, установленные на уровне 0 действуют для уровня 1. Уровень 2 – ещё менее привилегированный, а 3-й имеет самый низкий приоритет. Несложные системы могут использовать не все кольца, а только некоторые или даже одно. Например, в схеме “супервизор-пользователь” все программы операционной системы расположены в кольце 0, а пользовательские программы – в кольце 3.

Для описания механизма защиты пользуются следующими понятиями:

*Уровень привилегий дескриптора (Descriptor Privilege Level: DPL)* – уровень привилегий, на который помещен описываемый дескриптором объект. Поле DPL хранится в байте прав доступа дескриптора.

*Текущий уровень привилегий (Current Privilege Level: CPL)* – уровень привилегий выполняемого сегмента кода. Это значение соответствует DPL сегмента кода (кроме подчиняемых сегментов кода). Значение CPL хранится в поле RPL селектора сегмента кода, который помещен в регистр CS.

*Запрашиваемый уровень привилегий (Requested Privilege Level: RPL)* – используется для временного понижения уровня привилегий данного сегмента при обращении к памяти. RPL заносится в младшие биты селектора и выставляется, например, операционной системой для понижения привилегий сегмента, переданного непривилегированной стороной.

*Уровень привилегий ввода-вывода (Input/Output Privilege Level: IOPL)* – указывает какой уровень привилегирован для работы с портами ввода-вывода. Это значение хранится в регистре EFLAGS и может быть различным для разных задач.

В защищенном режиме реализованы: **защита от выполнения привилегированных команд, защита доступа к данным и защита сегментов кода.** Рассмотрим их.

В процессоре есть команды, которые могут кардинально изменить состояние всей системы. Такие команды выполняются только на нулевом уровне привилегий, а на всех других уровнях вызывают нарушение общей защиты (исключение №13). К этим командам относятся:

1. **HLT** – останов процессора;
2. **CLTS** – сброс флажка Task Switched (используется при управлении мультизадачностью);
3. **LIDT, LGDT, LLDT** – загрузка регистров дескрипторных таблиц;
4. **LTR** – загрузка регистра задачи;
5. **LMSW** – загрузка младшего слова регистра CR0;
6. **MOV CRx, reg32** – работа с управляющими регистрами;
7. **MOV DRx, reg32** – работа с регистрами отладки;

а также команды работы со специфическими регистрами (TRx – для 386,486; MCRs – для Pentium и P6; MTRRs – для P6). Следует отметить, что команда **POPF** также чувствительна к уровню привилегий. Она не изменяет состояние управляющих флажков IOPL, IF и др., если выполняется на уровне привилегий, отличном от нулевого. Кроме безусловно привилегированных команд есть команды чувствительные к уровню привилегий ввода-вывода. Это команды работы с портами (**IN, INS, OUT, OUTS**) и команды сброса/установки флажка разрешения прерываний (**CLI, STI**). Эти команды выполняются только в том случае, если  $CPL \leq IOPL$ . Если это условие не выполняется, то для команд ввода-вывода производится дополнительная сверка с картой разрешения портов ввода-вывода. Если код не имеет привилегий на выполнение команды, то возникает нарушение общей защиты (исключение №13).

Второй аспект защиты – защита доступа к данным. Код имеет право обратиться к данным, которые находятся на том же или на более низком уровне привилегий. При этом учитывается не только CPL, но и RPL. Данные доступны, если дескриптор сегмента данных имеет  $DPL \geq \max(CPL, RPL)$ . Такой контроль производится при загрузке селекторов в сегментные регистры (DS, ES, FS, GS). В сегментный регистр можно загрузить только селектор доступного с текущего уровня привилегий сегмента данных или, если сегментный регистр не будет использоваться, пустой селектор. Попытка нарушить правило привилегий или загрузить селектор системного дескриптора или дескриптора сегмента кода только для выполнения в сегментный регистр данных приведет к нарушению общей защиты (исключение №13). Кроме того, в командах изменения данных в памяти производится проверка на возможность записи в сегмент.

Особое правило привилегий для сегментов стека. Стек должен находиться строго на том же уровне привилегий, что и код программы ( $DPL = CPL$ ). При этом сегмент стека обязательно должен быть присутствующим ( $P=1$ ) и для него должны быть доступны операции и чтения, и записи.

Вызов подпрограмм без смены кодового сегмента в защищенном режиме процессора i386 производится аналогично вызову в реальном режиме с помощью команд JMP и CALL. Для вызова подпрограммы, код которой находится в другом сегменте (который может принадлежать библиотеке, другой задаче или операционной системе), процессор i386 предоставляет 2 варианта вызова, причем оба используют защиту с помощью прав доступа.

Первый способ состоит в непосредственном указании в поле команды JMP или CALL селектора сегмента, содержащего код вызываемой подпрограммы, а также смещение в этом сегменте адреса начала подпрограммы. Разрешение вызова происходит в зависимости от значения поля C в дескрипторе сегмента вызываемого кода. При  $C=0$  вызываемый сегмент не считается подчиненным (согласованным), и вызов разрешается, только если уровень прав вызывающего кода не меньше уровня прав вызываемого сегмента, т.е. при условии  $CPL \leq DPL$ . При  $C=1$  вызываемый

сегмент считается подчиненным и допускает вызов из кода с любым уровнем прав доступа, но при выполнении подпрограмма наделяется уровнем прав вызвавшего кода.

Очевидно, что первый способ непригоден для вызова функций операционной системы, имеющей обычно нулевой уровень прав, из пользовательской программы, работающей, как правило, на третьем уровне. Поэтому процессор i386 предоставляет другой способ вызова подпрограмм, основанный на том, что заранее определяется набор точек входа в привилегированные кодовые сегменты, и эти точки входа описываются с помощью специальных дескрипторов – дескрипторов шлюзов вызова подпрограмм. Этот дескриптор принадлежит к системным дескрипторам, и его структура отличается от структуры дескрипторов сегментов кода и данных (рисунок 7). Селектор из поля команды CALL используется для указания на дескриптор шлюза вызова подпрограммы в таблицах GDT или LDT. Для использования этого дескриптора вызывающий код должен иметь не меньший уровень прав, чем дескриптор, но если дескриптор шлюза доступен, то он может указывать на дескриптор сегмента вызываемого кода, имеющий более высокий уровень, чем имеет шлюз, и вызов при этом произойдет. При определении адреса входа в вызываемом сегменте смещение из поля команды CALL не используется, а используется смещение из дескриптора шлюза, что не дает возможности задаче самой определять точку входа в защищенный кодовый сегмент.

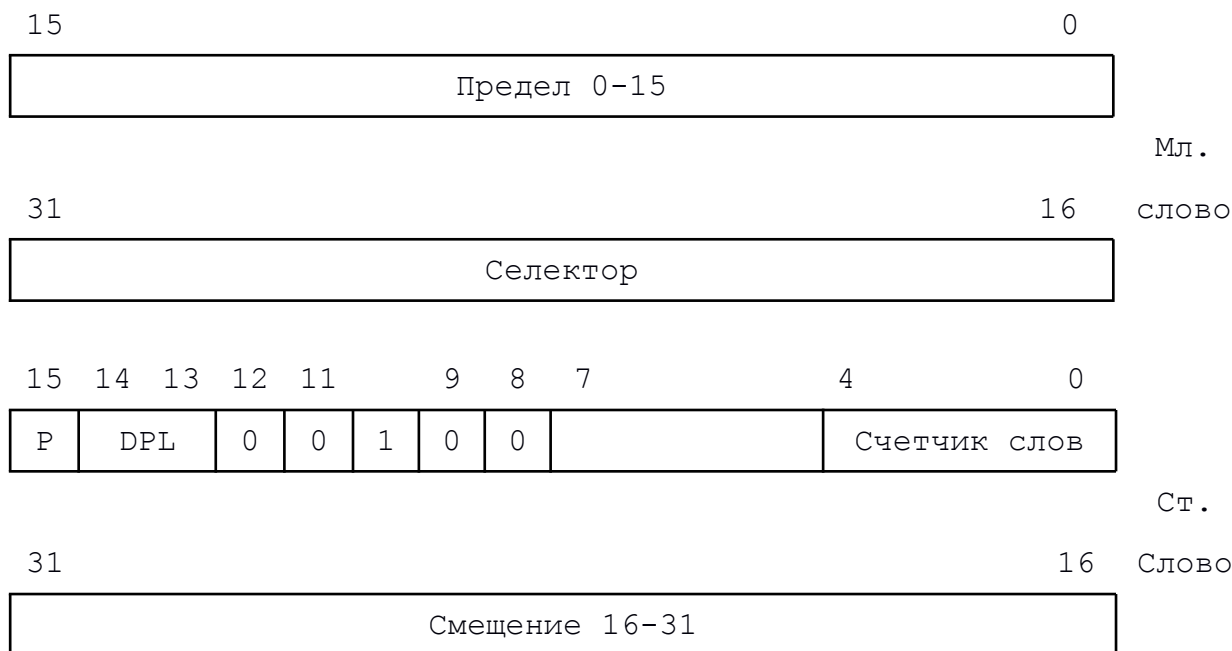


Рис. 7 – Дескриптор шлюза вызова для процессора i80386

Каждый уровень привилегий использует свой стек. При переключении уровней привилегий происходит переключение стека. При этом из текущего

стека в стек уровня доступа вызываемого сегмента копируется столько 32-разрядных слов, сколько указано в *поле счетчика слов* (WC) дескриптора шлюза. Перед этим во внутреннем стеке сохраняется указатель вершины внешнего стека. После копирования параметров во внутренний стек заносится адрес возврата. Привилегированная процедура должна заканчиваться инструкцией **RETF n**, где n - число байт, занимаемых параметрами в стеке.

Шлюзы можно адресовать только в команде FAR CALL, т. е. "наовсем" сменить уровень привилегий таким способом нельзя, всегда предполагается возврат на более низкий уровень привилегий.

### **Переключение в защищенный режим и обратно.**

Перед тем как переключить процессор в защищенный режим, надо выполнить некоторые подготовительные действия, а именно:

- 1 Подготовить в оперативной памяти глобальную дескрипторную таблицу. В этой таблице должны быть созданы дескрипторы для всех сегментов, которые будут нужны программе сразу после того, как она переключится в защищенный режим. Впоследствии, Находясь в защищенном режиме, программа может модифицировать имеющиеся дескрипторы или добавлять новые, загрузив заново регистр GDTR.
- 2 Запретить все маскируемые и немаскируемые прерывания.
- 3 Запомнить в оперативной памяти содержимое всех сегментных регистров, которые необходимо сохранить для возврата в реальный режим, в частности указатель стека реального режима.
- 4 Загрузить регистр GDTR.

Для того, чтобы запретить немаскируемые прерывания, следует в порт с адресом 70h записать байт, в котором старший бит установлен в 1.

Для загрузки регистра GDTR используется команда LGDT src, где src указывает на область памяти с адресом и пределом таблицы GDT. Образ регистра GDTR в оперативной памяти может быть представлен следующей структурой:

```
GDTR      label fword
GDT_Limit  DW  ?
GDT_Addr   DD  ?
```

Для переключения процессора из реального режима в защищенный надо установить бит PE системного регистра CR0 в 1. Для переключения можно использовать, например, такую последовательность команд:

```
Mov  AX, CR0
Or    AX, 1
Mov  CR0, AX
```

Первой командой после перехода в защищённый режим должна быть команда дальнего перехода (far jump), в которой будет указан селектор дескриптора сегмента кода и смещение в этом сегменте. При работе в



защищённом режиме процессор может использовать в сегментных регистрах только селекторы существующих дескрипторов, любые другие значения (например, сегментный адрес) использовать нельзя – процессор сгенерирует исключение общей защиты. Тем не менее, при переходе в защищённый режим регистр CS будет содержать сегментный адрес, который использовался в режиме реальных адресов, поэтому выполнение следующей команды, какой бы она ни была, должно было бы привести к генерации процессором исключения. На самом деле этого не происходит, так как эта команда не выбирается из памяти – она уже находится в конвейере процессора (даже в таком процессоре, как i386, есть конвейер) и поэтому вы можете выполнить эту команду.

Команда дальнего перехода обязательно очистит конвейер процессора и заставит его обратиться к таблице GDT, выбрать оттуда дескриптор, селектор которого указан в адресе команды, установить новое значение в регистр CS и начать выборку команд со смещения, также указанного в этом адресе. Это критический момент в работе программы. Если в GDT, селекторе, смещении или самой команде будет обнаружена ошибка, то процессор сгенерирует исключение, а так как система прерываний для него пока не определена, то он попросту зависнет либо произойдёт сброс – это уже зависит от "железа". Если вы не выполните первой команду дальнего перехода, а другую, которая не изменит содержимое регистра CS (а это все остальные команды), то процессор произведёт выборку в конвейер новой команды, используя текущие значения CS:IP, а так как в CS содержится не селектор (процессор уже в защищённом режиме!), то произойдёт исключение и зависание.

Команду `jmp far` можно закодировать, например, следующим образом:

```
db    0eah                ; jmp far Code_selector:P_Mode_entry
dw    P_Mode_entry
dw    Code_selector
```

Здесь `Code_selector` – селектор дескриптора сегмента кода, а `P_Mode_entry` – смещение в этом сегменте кода.

После этого следует загрузить остальные сегментные регистры, которые будут использоваться, правильными селекторами соответствующих дескрипторов.

Для переключения из защищенного режима в реальный режим программа должна выполнить следующие действия:

- 1 Передать управление сегменту кода с пределом 64Кбайта.
- 2 Загрузить в регистры SS, DS, ES, FS и GS селекторы дескрипторов, подготовленных для адресации памяти в реальном режиме и содержащих соответствующие реальному режиму значения. Дескрипторы должны иметь следующие параметры:
  - Предел = 64 Кб (FFFFh)
  - Байтная гранулярность (G = 0)
  - Расширяется вверх (E = 0)
  - Записываемый (W = 1)

- Присутствующий (P = 1)
- Базовый адрес = любое значение

Сегментные регистры должны быть загружены ненулевыми селекторами. Те сегментные регистры, в которые не будут загружены описанные выше значения, будут использоваться с атрибутами, установленными в защищённом режиме.

- 3 Запретить все маскируемые и немаскируемые прерывания.
- 4 Сбросить бит PE регистра CR0, переключив процессор в реальный режим.
- 5 Выполнить команду дальнего перехода для очистки внутренней очереди команд процессора.
- 6 Настроить систему прерываний для работы в реальном режиме (если она была изменена при входе в защищенный режим)
- 7 Разрешить прерывания.
- 8 Загрузить в сегментные регистры значения, необходимые для работы в реальном режиме.

Для разрешения всех прерываний можно воспользоваться следующим кодом:

```
Mov AL, 0dh ;разрешаем немаскируемые прерывания
Out 70h, AL
In AL, 21h ;разрешаем маскируемые прерывания
And AL, 0
Out 21h, AL
STI
```

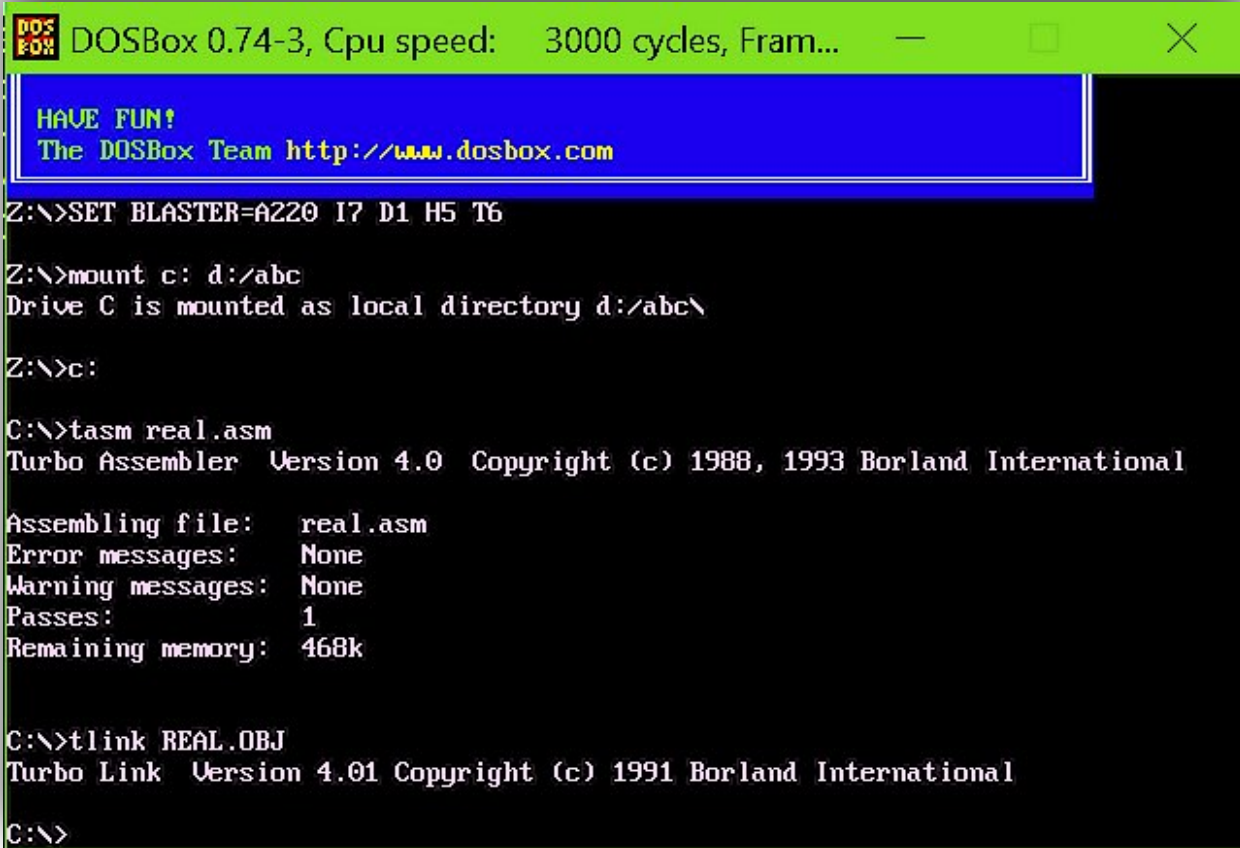


## 4. Программная реализация

В качестве средств для написания программы используются язык Assembler с компилятором TASM и средой исполнения DOSBox 0.74-3.

Программа разбита на несколько логических частей, представляющих из себя процедуры, которые выполняют следующие задачи: подготовку ко входу в защищенный режим, вход в защищенный режим, вывод строки и возврат в реальный режим.

На рисунках представлен результат работы программы. Демонстрируется вывод строки в защищённом режиме и переход обратно в реальный режим.



```
DOS
BOX DOSBox 0.74-3, Cpu speed: 3000 cycles, Fram... — □ ×

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount c: d:/abc
Drive C is mounted as local directory d:/abc\

Z:\>c:

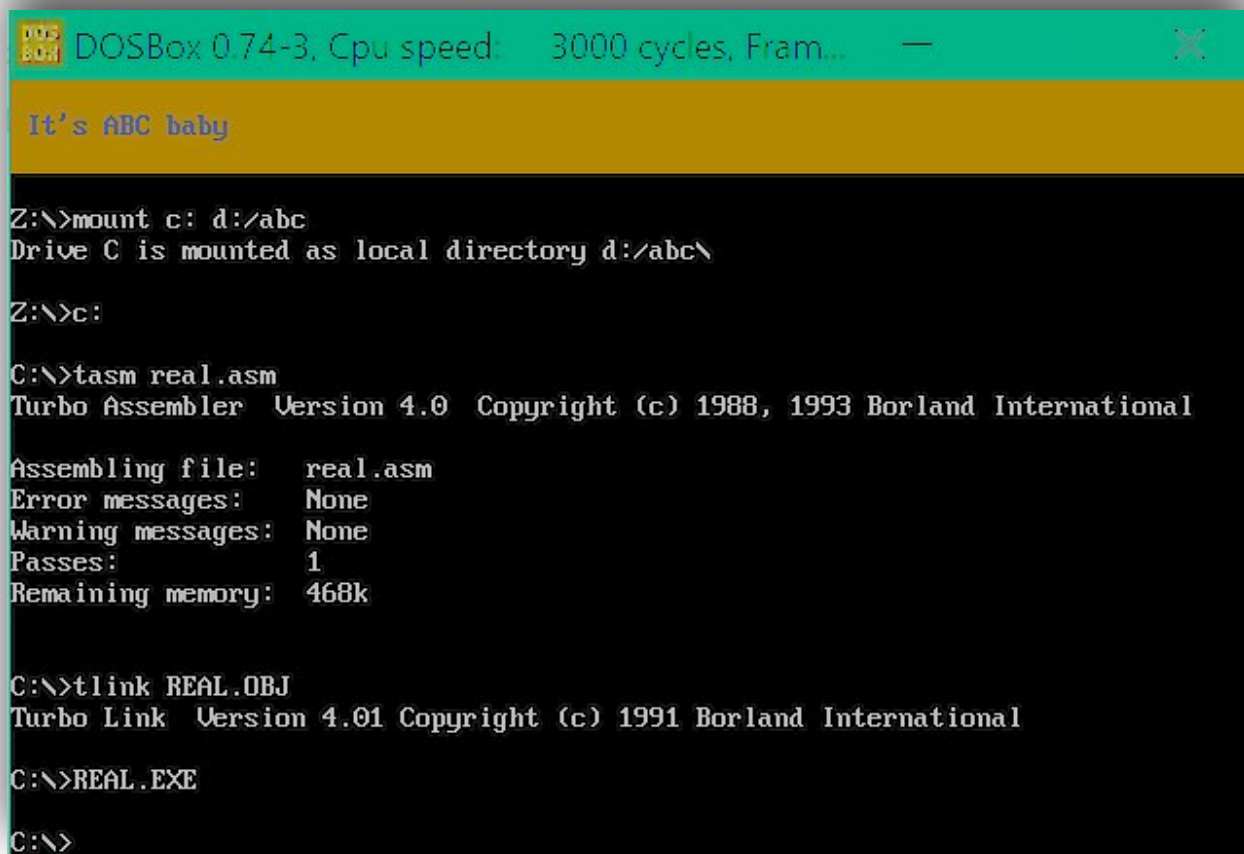
C:\>tasm real.asm
Turbo Assembler Version 4.0 Copyright (c) 1988, 1993 Borland International

Assembling file: real.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 468k

C:\>tlink REAL.OBJ
Turbo Link Version 4.01 Copyright (c) 1991 Borland International

C:\>
```

Рис.1 Подготовка к запуску программы



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Fram...
It's ABC baby

Z:\>mount c: d:/abc
Drive C is mounted as local directory d:/abc\

Z:\>c:

C:\>tasm real.asm
Turbo Assembler Version 4.0 Copyright (c) 1988, 1993 Borland International

Assembling file:   real.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  468k

C:\>tlink REAL.OBJ
Turbo Link Version 4.01 Copyright (c) 1991 Borland International

C:\>REAL.EXE

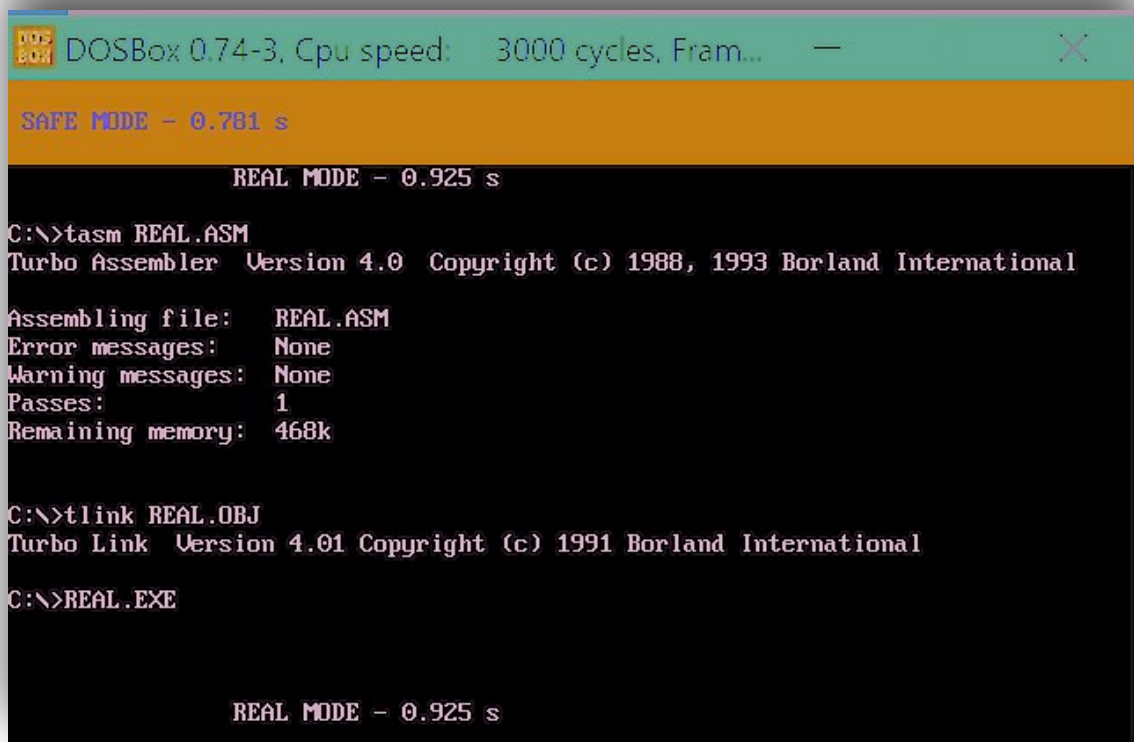
C:\>
```

Рис.2 Внешний вид консоли после успешно отработавшей программы

Теперь посчитаем ряд в реальном и защищенном режиме.

$$S(x) = \sum_{k=0}^n \frac{\cos\left(\frac{k\pi}{4}\right)}{k!} x^k,$$

На рисунке представлен результат работы программы. Демонстрируется расчет ряда в защищённом режиме и реальном режиме и время работы.



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Fram...  
SAFE MODE - 0.781 s  
REAL MODE - 0.925 s  
C:\>tasm REAL.ASM  
Turbo Assembler Version 4.0 Copyright (c) 1988, 1993 Borland International  
Assembling file: REAL.ASM  
Error messages: None  
Warning messages: None  
Passes: 1  
Remaining memory: 468k  
C:\>tlink REAL.OBJ  
Turbo Link Version 4.01 Copyright (c) 1991 Borland International  
C:\>REAL.EXE  
REAL MODE - 0.925 s
```

Рис.3 Внешний вид консоли после расчета ряда в каждом из режимов

Можем наблюдать, что защищенный режим работает быстрее. (В первой строке – защищенный, в последней - реальный)

## 5. Выводы

Перед тем как переключить процессор в защищенный режим, надо выполнить некоторые подготовительные действия, а именно:

- подготовить в оперативной памяти глобальную дескрипторную таблицу.
- запретить все маскируемые и немаскируемые прерывания;
- запомнить в оперативной памяти содержимое всех сегментных регистров, которые необходимо сохранить для возврата в реальный режим, в частности указатель стека реального режима;
- загрузить регистр GDTR.

Для переключения из защищенного режима в реальный режим программа должна выполнить следующие действия:

- передать управление сегменту кода с пределом 64Кбайта;
- загрузить в регистры SS, DS, ES, FS и GS селекторы дескрипторов, подготовленных для адресации памяти в реальном режиме и содержащих соответствующие реальному режиму значения.
- запретить все маскируемые и немаскируемые прерывания;
- сбросить бит PE регистра CR0, переключив процессор в реальный режим;
- выполнить команду дальнего перехода для очистки внутренней очереди команд процессора;
- настроить систему прерываний для работы в реальном режиме. (если она была изменена при входе в защищенный режим);
- разрешить прерывания;
- загрузить в сегментные регистры значения, необходимые для работы в реальном режиме.

В ходе выполнения лабораторной работы были изучены особенности защищенного режима процессора: произведён обзор режимов 32-разрядных микропроцессоров (IA-32), рассмотрены вопросы адресации памяти в защищённом режиме, защиты по привилегиям, переключения в защищённый режим и обратно.

Получены практические навыки по программированию переключения процессора из реального в защищенный режим и обратно: написана программа, переключающая процессор в защищенный режим, выводящая на экране сообщение и затем возвращающая процессор в реальный режим.

## Список литературы

1. Волорова Н. А. Лабораторный практикум по курсу «Архитектура вычислительных систем» для студентов специальности «Информатика» / 93-444-487-2- Мн.: БГУИР, 2003. – 32с.:ил.
2. Таненбаум, Э. С. Архитектура компьютера / Э. С. Таненбаум, Т. Остин ; пер. Е. Матвеев. - 6-е изд. - Санкт-Петербург [и др.] : Питер, 2018. - 811, [1] с. : ил., табл. - (Классика Computer science).
3. Фролов А.В., Фролов Г.В.. Защищенный режим процессоров Intel 80286, 80386, 80486: Практическое руководство по использованию защищенного режима. М.: Диалог-Мифи, 1993.

## Приложение 1. Текст программы

```
.model      tiny
.stack     100h
.code
.386p
org        100h

start:
    push    cs
    pop     ds                ; DS - сегмент данных (и
кода) нашей программы
    push    0B800h
    pop     es                ; ES - сегмент видеопамати

    cli
    in      al,70h            ; индексный порт CMOS
    or      al,80h            ; установка бита 7 в
нем запрещает NMI
    out     70h,al
    mov     eax,cr0            ; прочитать регистр CRO
    or      al,1              ; установить бит PE,
    mov     cr0,eax            ; с этого момента мы в
защищенном режиме

    mov     di,0                ; ES:DI - начало
видеопамати

    mov     si,offset message
    mov     cx,message_length

    mov     ah, 69h            ; атрибут внешнего вида текста

wxy_write:
    lodsb    ; очередной символ в al
    stosw    ; записываем его в видеопамать
    loop     wxy_write          ; цикл до конца строки

    push     cx
    mov     cx,50
ploop0:
    push     cx
    xor      cx,cx
ploop1:
    loop     ploop1
    pop     cx
    loop     ploop0

    pop     cx
```

```

        mov     eax,cr0                ; прочитать CR0
        and     al,0FEh                ; сбросить бит PE
        mov     cr0,eax                ; с этого момента
процессор работает в реальном режиме

        in      al,70h                 ; индексный порт CMOS
        and     al,07Fh                ; сброс бита 7 отменяет
блокирование NMI
        out     70h,al
        sti
        mov     ah,0
        int     16h
        mov     ax, 4c00h
        int     21h

message    db     80 dup(' ')
           db     "
"
           db     " It's ABC baby
"
           db     "
"
message_length = $ - message

end        start

```