Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра информатики

Н.А. Волорова, А.С. Дурович

Архитектура вычислительных систем

Лабораторный практикум

для студентов специальности “Информатика”
всех форм обучения

Минск 2003

УДК 681.3 (075.8)
ББК 32.973 я 73
В 68

Волорова Н.А.

В 68 Архитектура вычислительных систем: Лаб. практикум для студ. спец. «Информатика» всех форм обучения / Н.А. Волорова, А.С. Дурович – Мн.: БГУИР, 2003. — 32 с.: ил.

ISBN 985-444-487-2

Лабораторный практикум предназначен для изучения архитектуры и программирования сопроцессора, защищенного режима процессора, мультизадачности в защищенном режиме.

УДК 681.3 (075.8)
ББК 32.973 я 73

ISBN 985-444-487-2

Содержание

Лабораторная работа №1

Программирование арифметического сопроцессора

1. Цель работы

Изучить назначение и архитектуру арифметического сопроцессора. Получить практические навыки по программированию сопроцессора.

2. Теоретические сведения

2. 1. Сопроцессорные конфигурации

Использование сопроцессора позволяет значительно ускорить работу программ, выполняющих расчеты с высокой точностью, тригонометрические вычисления и обработку информации, которая должна быть представлена в виде действительных чисел. Сопроцессор (FPU) подключается к системной шине параллельно с центральным процессором (CPU) и может работать только совместно с ним. Все команды попадают в оба процессора, но каждый процессор выполняет только свои команды. Сопроцессор не имеет своей программы и не может осуществлять выборку команд и данных. Это делает центральный процессор. Сопроцессор перехватывает с шины данные и после этого реализует конкретные действия по выполнению команды. Два процессора работают параллельно, что повышает эффективность системы. Однако возникают ситуации, когда их работа требует синхронизации (из-за разницы во времени выполнения команд).

Синхронизация по командам. Когда центральный процессор выбирает для выполнения команду сопроцессора, последний может быть занят выполнением предыдущей команды. Поэтому перед каждой командой сопроцессора в программе должна стоять специальная команда (wait), которая только проверяет текущее состояние сопроцессора и, если он занят, переводит центральный процессор в состояние ожидания. Соответствующую команду в программу может вводить либо ассемблер, либо компилятор языка без указаний программиста.

Синхронизация по данным. Если выполняемая в сопроцессоре команда записывает операнд в память перед последующей командой процессора, которая обращается к этой ячейке памяти, требуется команда проверки состояния сопроцессора. Если данные еще не были записаны, процессор должен переходить в состояние ожидания. Автоматически учесть такие ситуации довольно сложно, поэтому вводить команды, которые проверяют состояние сопроцессора и при необходимости заставляют центральный процессор ожидать, должен программист.

2.2. Программная модель сопроцессора

В программную модель любого процессора включаются только те регистры, которые доступны программисту на уровне машинных команд. Основу программной модели сопроцессора образует регистровый стек из восьми 80-битных регистров R0-R7. В них хранятся числа в вещественном формате. В любой момент времени 3-битное поле ST в слове состояния определяет регистр, являющийся текущей вершиной стека и обозначаемый ST(0). При занесении в стек (push) осуществляется декремент поля ST и загружаются данные в новую вершину стека. При извлечении из стека (pop) в получатель, которым чаще всего является память, передается содержимое вершины стека, а затем инкрементируется поле ST .

В организации регистрового стека сопроцессора есть следующие отличия от классического стека:

- стек имеет кольцевую структуру. Контроль за использованием стека должен осуществлять программист. Максимальное число занесений без промежуточных извлечений равно 8;
- в командах сопроцессора допускается явное или неявное обращение к регистрам с модификацией или без поля ST. Например, команда fsqrt замещает число из вершины стека значением корня из него. В бинарных операциях допускается явное указание регистров. Адресация осуществляется относительно текущей вершины стека и обозначается ST (i) $0<i<7$, считая от вершины;
- не все стековые команды автоматически модифицируют указатель вершины стека.

С каждым регистром стека ассоциируется 2-битный тег (признак), совокупность которых образует слово тегов. Тег регистра R0 находится в младших битах, R7 – в старших. Тег фиксирует наличие в регистре действительного числа (код 00), истинного нуля (код 01), ненормализованного или бесконечности (код 10) и отсутствие данных (код 11). Наличие тегов позволяет сопроцессору быстрее обнаруживать особые случаи (попытка загрузить в непустой регистр, попытка извлечь из пустого) и обрабатывать данные.

Остальными регистрами сопроцессора являются регистр управления, регистр состояния, два регистра состояния команды и два регистра указателя данных. Длина их всех 16 бит.

2.3. Форматы численных данных

Арифметический сопроцессор К1810ВМ87 оперирует семью форматами численных данных, образующих три класса: двоичные целые, упакованные десятичные целые и вещественные числа. Во всех форматах старший (левый) бит отведен для знака.

Форматы различаются длиной, следовательно, диапазоном допустимых чисел, способом представления (упакованный и неупакованный формат), способом кодировки(прямой и дополнительный код).

Двоичные целые числа. Три формата целых двоичных (целое слово (16 бит), короткое целое (32 бита), длинное целое (64 бита)) различаются длиной, следовательно, диапазоном чисел. Только в этих форматах применяется стандартный дополнительный код. “0” имеет единственное кодирование. Наибольшее положительное число кодируется как 011…1, а наибольшее по модулю отрицательное – как 100...0.

Упакованные десятичные целые. Числа представлены в прямом коде и упакованном формате, т.е. в байте содержится две десятичные цифры в коде 8421. Старший бит левого байта – знак, остальные игнорируются, но при записи в них помещаются нули. Но надо учитывать, что при наличии в тетраде запрещающих комбинаций 1010 – 1111, результат операции не определен, т.е. сопроцессор не контролирует правильность результата.

Вещественные числа. Различают короткие вещественные (КВ)(мантисса – 24 бита, порядок – 8 бит), длинные вещественные (ДВ) (мантисса – 53 бита, порядок – 11 бит) и временные вещественные (ВВ) (мантисса – 64 бита, порядок – 15 бит). Для них применяется формат с плавающей точкой. Значащие цифры находятся в поле мантиссы, порядок показывает фактическое положение двоичной точки в разрядах мантиссы, бит знака S определяет знак числа. Порядок дается в смещенной форме :

$$E = \text{истинный порядок} + \text{смещение}.$$

Смещение для соответствующих форматов равно 127, 1023, 16383, это упрощает операцию сравнения. Операция с целыми числами быстрее операции над плавающей точкой. Это важно в алгоритмах.

Значение числа равно

$$(1)^S \times 2^{\text{E-смещение}} \times F_0F_1F_2\ldots F_n,$$

где n для разных форматов равно 23, 52 или 63.

Порядок имеет фиксированную длину, определяя один диапазон представимых чисел. Мантисса – правильная дробь. В коротком и длинном вещественном формате бит F_O при передаче чисел и хранении их в памяти не фигурирует. Это скрытый бит, который в нормализованных числах содержит “1”. Скрытый бит не дает представить в этих форматах нуль и он должен кодироваться как специальное значение.

Числа во временном вещественном формате имеют явный бит F_O. Формат повышает скорость выполнения операций благодаря простоте представления чисел, не являющихся ненормализованными. При загрузке из памяти в регистр FPU оно преобразуется во временный вещественный формат. А при записи в память – обратный формат. Временной вещественный формат – единственный, в котором абсолютно точно кодируются любые загружаемые из памяти числа.

2.4. Режимы работы. Состояние

Сопроцессор имеет 2 доступных 16-битных регистра, содержимое которых определяет его режим работы и текущее состояние. Форматы регистров содержат слово управления CW и слово состояния SW. Регистр управления содержит 6 бит масок особых случаев. Регистр состояния – 6 бит флажков.

Регистр управления:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	IC	RC		PC		IEM	X	PM	UM	OM	ZM	DM	IM

Регистр состояния:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	C3	ST			C2	C1	C0	IR	X	PE	UE	OE	ZE	DE	IE

Рис 1. Форматы слова управления и слова состояния

Регистр управления содержит 6 бит масок особых случаев, а регистр состояния - 6 бит флажков особых случаев:

- P – потеря точности;
- U – антипереполнение;
- O – переполнение;
- Z - деление на нуль;
- D - денормализованный операнд;
- I - недействительная операция.

Слово управления. Оно определяет для сопроцессора один из нескольких вариантов обработки численных данных. Программа центрального процессора может сформировать в памяти образ слова управления, а затем заставить сопроцессор загрузить его в регистр CW. Рассмотрим значение полей.

Шесть младших бит слова управления представляют собой индивидуальные маски особых случаев, т.е. особых ситуаций, обнаруженных сопроцессором при выполнении команд. Если любой из этих бит установлен в ”1”, то соответствующее этому биту прерывание не будет вызвано. В противном случае (соответствующий бит установлен в ”0”) сопроцессор устанавливает в “1” бит запроса прерывания в слове состояния и при общем разрешении прерываний генерирует сигнал int прерывания сопроцессора.

Бит 7 слова управления содержит маску управления прерыванием IEM, которая разрешает (IEM=0) или запрещает(IEM=1) прерывание центрального процессора.

2-битное поле управления точностью (PC) определяет точность вычислений в 24 бита (PC=00), 53 бита (PC=10) или 64 бита (PC=11). По умолчанию вводится режим с максимальной точностью в 64 бита.

2-битное поле управления округлением RC определяет один из четырех возможных вариантов округления результатов операций сопроцессора.

Бит 12 управляет режимом обработки специального значения бесконечности IC. Когда IC=0, сопроцессор обрабатывает два специальных значения “плюс бесконечность” и “минус бесконечность” как одно и то же значение “бесконечность”, не имеющее знака.

Слово состояния. В нем младшие 6 бит отведены для регистрации особых случаев. Бит 7 – запроса прерывания (IR) устанавливается в “1” при возникновении любого незамаскированного особого случая. Биты C3-C0 фиксируют код условия в операциях сравнения, проверки условия и анализа. Три бита ST – указатели стека. Стековые операции сопровождаются модификацией поля ST. Наконец, флажок занятости B устанавливается в состояние “1” когда численное операционное устройство выполняет операцию. Большую роль играют биты кода условия, которые фиксируют особенности результата (см. табл.). Коды условия привлекаются для реализации условных переходов. Сопроцессор самостоятельно не может влиять на ход выполнения программ. Поэтому для условных переходов по результатам операций сопроцессора приходится сначала передавать код условия в память, а затем загружать один из регистров центрального процессора. После этого код условия передается в регистр флагов, производится условный переход.

Интерпретация кода условия в операциях сравнения и проверки (FCOM, FCOMP, FTST)

C3	C2	C1	C0	Описание
0	X	X	0	(ST)>источника (src)
0	X	X	1	(ST)<источника (src)
1	X	X	0	(ST)=источнику (src)
1	X	X	1	Не сравнимы

2.5. Система команд

Система команд сопроцессора содержит 6 групп команд: команды передач данных, арифметические команды, команды сравнения, трансцендентных операций, команды загрузки констант и управления сопроцессором. Операнды некоторых команд определяются неявно. Другие команды допускают или требуют явного задания операнда.

При рассмотрении системы команд сопроцессора будем пользоваться следующими обозначениями: src обозначает источник, т.е. операнд, значение которого не модифицируется, dst – получатель, т.е. операнд, значение которого замещается результатом операции.

Особенности задания команд. Команды бинарных операций допускают несколько форм задания**.** При пустом поле операнда операция выполняется с двумя верхними элементами стека ST(источник) и ST(1)(получатель). После производства операции осуществляется инкрементирование указателя стека и результат помещаеться в новую вершину стека, заменяя исходное содержимое

ST(1). Когда в бинарной команде определен один операнд, операция выполняется с привлечением указанного в команде регистра или ячейки памяти и содержимого вершины стека. Результат загружается в старую вершину стека и указатель стека не модифицируется. Если в бинарной команде указаны два операнда, ими является содержимое двух регистров стека, причём одним из них будет ST, а вторым –ST(i).

Альтернативные формы операндов условно показываются с помощью наклонной черты, причем черта без последующей спецификации означает отсутствие явно задаваемых операндов. Например, команда FADD имеет следующий общий вид:

FADD //src/dst,src

Такая запись подразумевает три возможных формы команд: без операндов, с одним источником, с получателем и источником.

В мнемониках команд сопроцессора приняты следующие соглашения: первая буква всегда F; вторая буква I обозначает операцию с целыми числами, буква B – операцию с десятичным целым операндом, а пустая – операцию с вещественными числами; предпоследняя или последняя буква R указывает обратную операцию (например, в обычной форме команды деления получатель делится на источник, а в обратной форме источник делится на получатель; в обоих формах результат помещается в получатель); последняя буква P идентифицирует команду, заключительным действием которой является извлечение из стека.

Команды передачи данных. Команды этой группы производят передачу данных между регистрами стека, а также между вершинами стека и памятью. Одной командой число из памяти преобразуется во временный вещественный формат и загружается в стек. Таким же образом, но в обратном порядке осуществляется передача числа в память.

Команды загрузки. 3 команды загрузки имеют следующий вид:

- вещественное: FLD src;
- двоичное целое FILD src;
- десятичное целое FBLD src.

Эти команды осуществляют декремент указания стека и передачу в новую вершину стека содержимого источника. В команде FLD источником может быть один из регистров стека или вещественное число. В командах FILD и FBLD – только операнд в памяти.

Команды запоминания

- вещественное: FST dst;
- двоичное целое: FIST dst.

Они производят передачу содержимого вершины стека в память без модификации указателя ST и содержимого ST(0). В команде FST получатель - регистр стека или вещественная переменная в памяти. В команде FIST получателем является переменная в памяти, имеющая формат короткого целого и

целого слова. Рассмотренные команды не допускают для получателя формат длинного целого, временного вещественного и неупакованного десятичного.

Команды запоминания с извлечением из стека. Три команды помимо передачи содержимого ST(0) осуществляют извлечение из стека. Регистр бывшей вершины стека отмечается как пустой и производится инкремент указателя стека:

- вещественное: FSTP dst;
- двоичное целое: FISTP dst;
- десятичное целое: FBSTP dst.

Действия команды FSTP очень похожи на действия FST с добавлением извлечения из стека. Однако FSTP может передать в память слово во временном вещественном формате, чего не может сделать FST. Команда FIST обеспечивает передачу в память числа в любом формате целого двоичного, включая длинное целое. Последний формат недопустим в формате FSTP. Команда FBST преобразует операнд из вершины стека в упакованное десятичное число, передает его в память и производит извлечение из стека.

Команда обмена содержимого регистров

FXCH//dst ST(0)<->(dst) обменивает содержимое получателя ST(i) с вершиной ST(0). При пустом поле операнда обменивается содержимое регистров ST(1) и ST(0).

Команды управления

- команда FINIT / FNINIT - инициализировать сопроцессор;
- команда FLDCW src – загрузить слово управления. Источником является целое число в памяти;
- команды FSTCW dst и FSTSW dst – запомнить слово управления и состояния в ячейке памяти, определяемой получателем dst;
- команда FSTENV dst – запомнить среду.

Под средой сопроцессора К1810ВМ87 понимается содержимое регистров управления, состояния, тегов, указателя команды и указателя операнда. Команда FSTENV передаёт его в область памяти с начальным адресом, указанным в команде. Формат хранения среды в памяти показан на рис. 2.

Начальный адрес →	15 … 0	Смещение
	Слово управления	+0
	Слово состояния	+2
	Слово тегов	+4
	Указатель команды	+6
		+8
	Указатель операнда	+10
		+12

Рис. 2. Формат хранения среды сопроцессора в памяти

До выборки из очереди следующей команды сопроцессора выполнение команды FSTENV должно закончиться.

Команда FLDENV src – загрузить среду. Парная предыдущей команде команда FLDENV src осуществляет загрузку среды сопроцессора из области памяти, определяемой src. После команды FLDENV не требуется команда FWAIT, так как сопроцессор автоматически контролирует завершение передачи всех слов среды до перехода к своей следующей команде.

Команда FSAVE dst - сохранить полное состояние сопроцессора. Полное состояние сопроцессора представляет собой содержимое всех регистров програмной модели – среды и восьми регистров стека. Размер полного состояния сопроцессора составляет 94 байта. Команда FSAVE передаёт его в область памяти с начальным адресом, указаным в команде. Формат размещения полного состояния сопроцессора в памяти (его иногда называют «образом в памяти») показан на рис.3.

Начальный адрес →	15 … 0	Смещение
	Среда сопроцессора	+0
	Верхний элемент стека st(0)	+14
	Следующий элемент стека st(1)	+24
	...	...
	Последний элемент стека st(7)	+12

Рис. 3. Формат хранения полного состояния сопроцессора в памяти

Команда FRSTOR src – восстановить полное состояние сопроцессора.

Для сохранения и восстановления состояния сопроцессора обычно применяется следующий ассемблерный фрагмент:

```
SUB   SP,94;Зарезервировать пространство в стеке
MOV   BP,SP;BP является базой для состояния
FSAVE [BP];Сохранить полное состояние
      ...
MOV   BP,SP;BP является базой для состояния
FRSTOR [BP];Восстановить состояние
ADD   SP,94;Освободить пространство в стеке
```

Арифметические команды

Необходимо отметить, что вещественные числа в памяти не могут быть в формате временного вещественного, а целые числа – в формате длинного целого. Здесь сказывается недостаточность наборов кодов операций.

Команды сложения. Операция сложения реализуется командами со следующими формами:

- вещественные числа FADD //src/dst,src;
- вещественные числа с извлечением из стека FADDP dst,src;
- целые числа FIADD src.

Отметим, что команда FADD ST,ST(0) удваивает содержимое вершины стека.

Команды вычитания. Обычное вычитание $dst \leftarrow (dst) - (src)$ осуществляют команды:

- вещественные числа FSUB //src/dst,scr;
- вещественные числа с извлечением из стека FSUBP dst,src;
- целые числа FISUB src.

Для производства обратного вычитания $dst \leftarrow (src) - (dst)$ предназначены команды FSUBR, FSUBRP, FISUBR, имеющие аналогичные формы.

Команды умножения. Операция умножения реализуется следующими командами:

- вещественные числа FMUL //src/dst,scr;
- вещественные числа с извлечением из стека FMULP dst,src;
- целые числа FIMUL src.

Команды деления. Для выполнения обычной операции деления предусмотрены команды:

- вещественные числа FDIV //src/dst,scr;
- вещественные числа с извлечением из стека FDIVP dst,src;
- целые числа FIDIV src.

Соответствующие команды обратного деления FDIVR, FDIVRP, FIDIVR загружают в получатель частное от деление источника на получатель.

Приведём несколько примеров арифметических команд:

```
FADD ST, ST(5);Сложить содержимое регистров
FIADD WORD PTR COUNT [SI];Прибавить целое слово
FSUBP ST(2), ST ;Вычесть содержимое регистров
FDIVR DWORD PTR [SI];Разделить короткое вещественное
FIDIVR    DWORD PTR [BX+5];Разделить короткое целое
```

Команда извлечения квадратного корня. Команда FSQRT извлечения квадратного корня заменяет число, находящееся в вершине стека, значением квадратного корня:

FSQRT ST(0) <- $ST(0)^{1/2}$.

3. Лабораторное задание

Написать программу, находящую решение квадратного уравнения $ax^2 + bx + c = 0$ с помощью сопроцессора.

4. Контрольные вопросы

1. Могут ли процессор и сопроцессор работать параллельно?
2. В чем суть синхронизации по данным?
3. В чем суть синхронизации по командам?
4. Какие регистры образуют программную модель сопроцессора?
5. Какими форматами численных данных может оперировать сопроцессор?
6. Что подразумевается под средой сопроцессора?
7. Что подразумевается под состоянием сопроцессора?

Лабораторная работа №2

Защищенный режим 32-разрядных процессоров

1. Цель работы

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

2. Теоретические сведения

2.1. 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. Для краткости в тексте используется термин "процессор i86", хотя вся информация этого раздела в равной степени относится к трем моделям процессоров фирмы Intel.

Процессор i86 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-разрядные регистры общего назначения, регистры 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; подразумевается, что операционная система будет работать в защищённом режиме (поэтому оптимизация по критерию 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; 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, драйверы 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.

2.2. Адресация памяти в защищенном режиме

Физическое адресное пространство процессора i86 составляет 4 Гб, что определяется 32-разрядной шиной адреса. Физическая память является линейной с адресами от 00000000 до FFFFFFFF в шестнадцатеричном представлении. Процессор i86 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-разрядной, т.к. допустимый размер сегмента значительно превышает 64 Кб. 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.4.

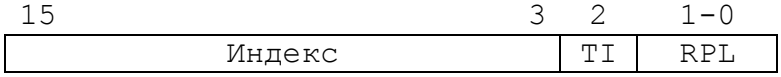

Рис.4. Полный формат селектора адреса

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, который будет обсужден 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, т.е. 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 8K (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 8K (2^{13}) сегментов каждого типа, всего 16 K. С учетом максимального размера сегмента - 4 Гб - каждая задача при чисто сегментной организации виртуальной памяти работает в виртуальном адресном пространстве в 64 Тб.

Процесс преобразования логического адреса в линейный приведен на рис.5.

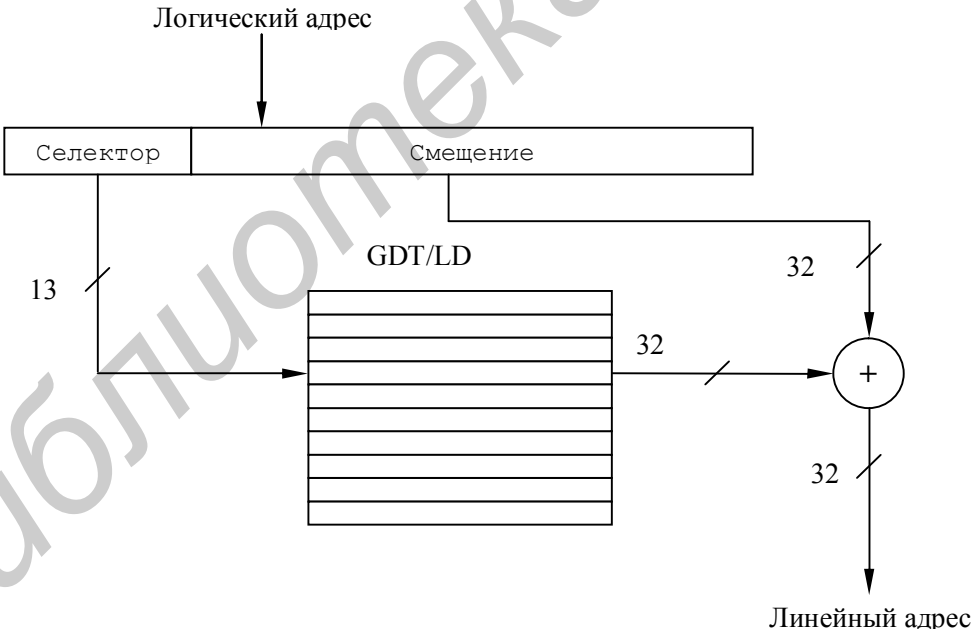

Рис. 5. Преобразование логического адреса в линейный

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. 6.

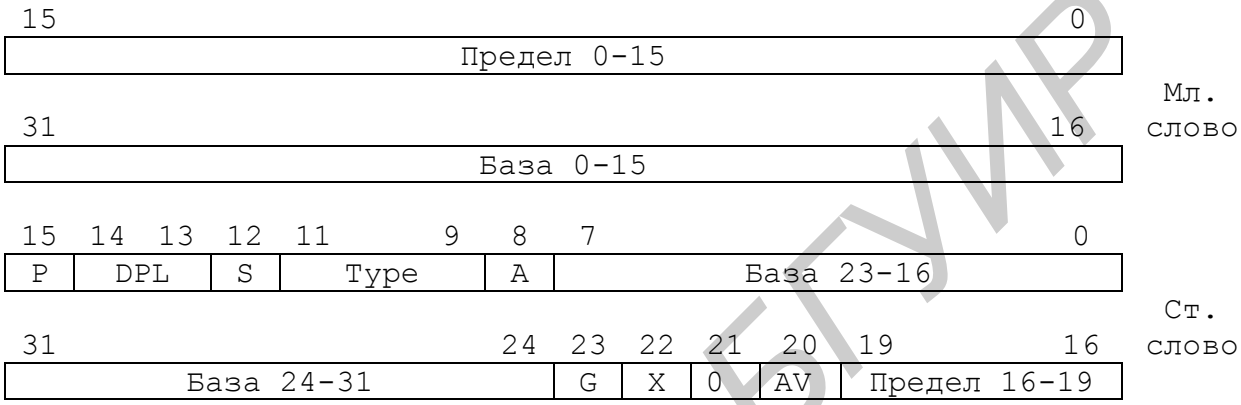

Рис. 6. Дескрипторы для процессора i80386

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, Тип, 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 неприсутствующего сегмента и запустит обработчик 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 (Acessed) - бит доступа в сегмент. Этот бит показывает, был произведен доступ к сегменту, описываемому этим дескриптором, или нет. Если процессор обращался к сегменту для чтения или записи данных или для выполнения кода, размещённого 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, 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 двухбитное поле, в него при создании дескриптора записывают значения от 0 до 3, определяющее уровень привилегий.

Поле Type - трёхбитное поле. Это поле интерпретируется по-разному, в зависимости от типа сегмента (см. рис.7.).

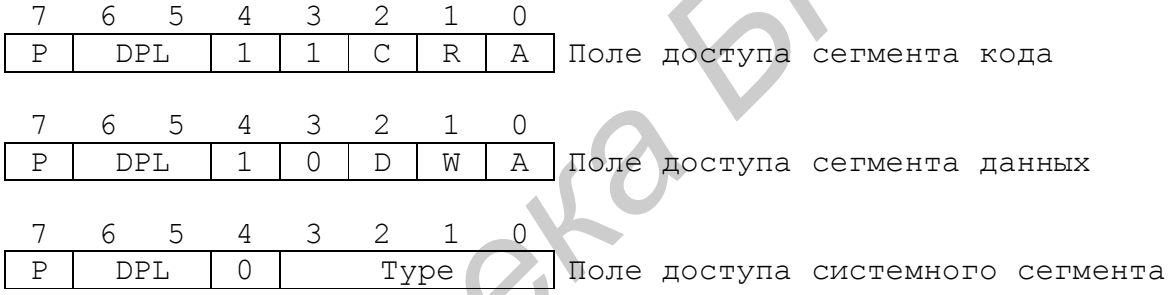

Рис.7. Формат поля доступа дескриптора

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 указывает, согласованный это сегмент (бит = 1) или нет (бит = 0). Этот бит будет рассмотрен позже. Поле R указывает: разрешено чтение сегмента кода (бит = 1) или - только выполнение (бит = 1).

Поле доступа сегмента кода имеет битовые поля D 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 содержат 32 бита линейного базового адреса дескрипторной таблицы и 16 бит предела таблицы. Формат регистра GDTR приведен на рис. 8.

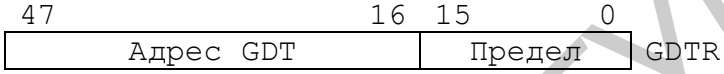

Рис.8. Формат регистра GDTR

Перед переходом в защищенный режим программа должна создать в оперативной памяти таблицу GDT и загрузить регистр GDTR при помощи команды LGDTR 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 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 генерирует 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. 9. 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. Всего может быть 1024 дескриптора. Самих 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 Кб, т. е. 4096 байт.

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

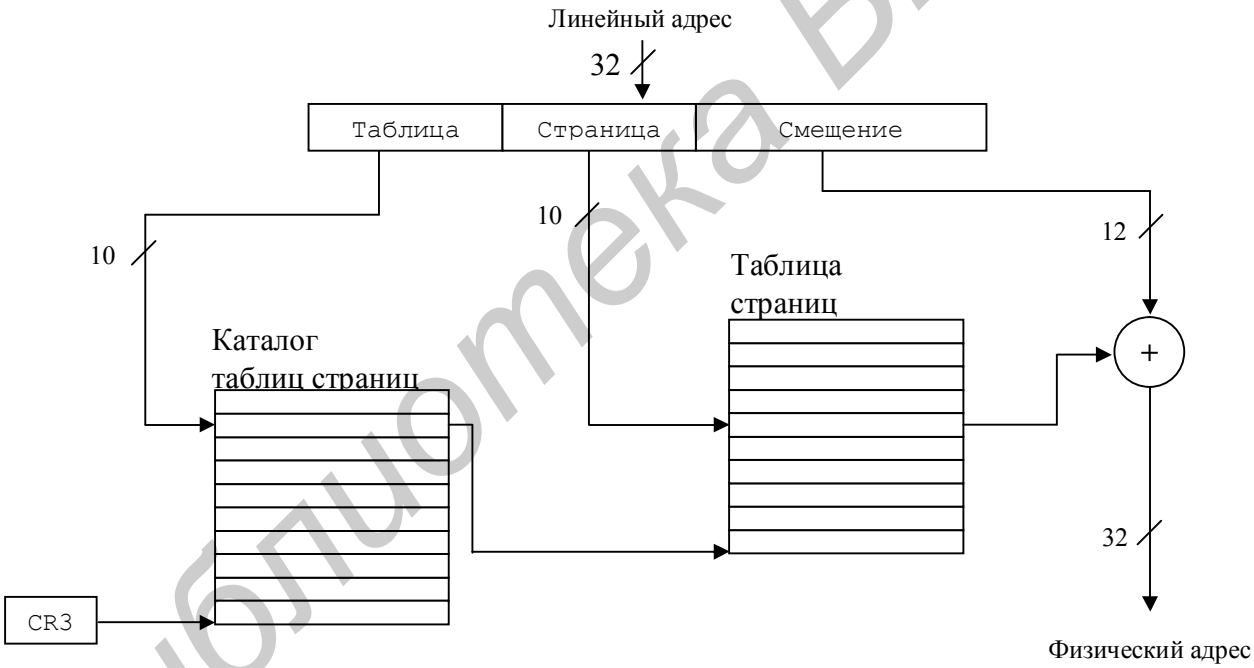

Рис.9. Преобразование линейного адреса в физический

2.3. 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 своего уровня привилегий при обращении к памяти. RPL заносится в младшие биты селектора.

Уровень привилегий ввода-вывода (Input/Output Privilege Level: IOPL) – указывает, при каком уровне разрешена работа с портами ввода-вывода. Это значение хранится в регистре EFLAGS и может быть различным для разных задач.

В защищеном 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:

- **HLT** - останов процессора;
- **CLTS** - сброс флажка Task Switched (исп. при управлении мульти-задачностью);
- **LIDT, LGDT, LLDT** - загрузка регистров дескрипторных таблиц;
- **LTR** - загрузка регистра задачи;
- **LMSW** - загрузка младшего слова регистра CR0;
- **MOV CRx,reg32** - работа с управляющими регистрами;
- **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 **POPFD** 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<=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>=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 реализуется 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 i86 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 <= 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 (рис. 10). 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.

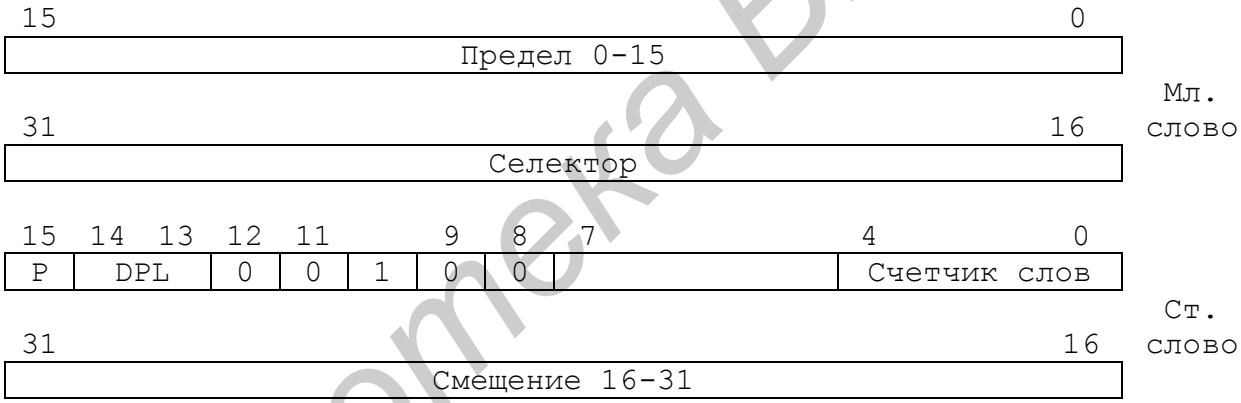

Рис. 10. Дескриптор шлюза вызова для процессора i80386

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 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.

Адресацию шлюза можно выполнить только командой FAR CALL, т.е. "насовсем" сменить уровень привилегий таким способом нельзя, всегда предполагается возврат на более низкий уровень привилегий.

2.4. Переключение в защищенный режим и обратно

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

- 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 переключется в защищенный режим. Впоследствии, находясь в защищенном режиме, программа может модифицировать имеющиеся дескрипторы или добавлять новые, загрузив заново регистр GDTR;
- 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 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 LGDTR 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, 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 т. к. систему прерываний мы для него пока не определяли, то он попросту зависнет либо произойдёт сброс - это уже зависит от "железа". Если вы не выполните команду дальнего перехода первой, а выполните 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 остальные сегментные регистры, которые будут использоваться, следует загрузить правильными селекторами соответствующих дескрипторов.

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

- передать управление сегменту кода с пределом 64Кб;
- 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;
- 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.

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
```

3. Лабораторное задание

Написать программу, переключающую процессор в защищенный режим, выводящую на экране сообщение и затем возвращающую процессор в реальный режим.

4. Контрольные вопросы

1. В чем преимущество защищенного режима по сравнению с реальным?
2. Какую структуру имеет селектор адреса?
3. Поясните процесс преобразования логического (виртуального) адреса в линейный.
4. Поясните процесс преобразования линейного адреса в физический.
5. Какая информация хранится в дескрипторах дескрипторных таблиц?
6. Какой максимальный размер сегмента в защищенном режиме?
7. Сколько в системе может быть глобальных и локальных дескрипторных таблиц?
8. Какие подготовительные действия нужно выполнить перед переключением в защищенный режим?
9. Какие подготовительные действия нужно выполнить перед возвратом в реальный режим?
10. Почему первой командой после переключения процессора в защищенный режим должна быть команда дальнего перехода?
11. Как запретить немаскируемые прерывания?
12. Какие существуют в защищенном режиме способы вызова подпрограммы, код которой находится в другом сегменте? Сравните их.

Лабораторная работа №3

Мультизадачность в защищенном режиме

1. Цель работы

Изучить принципы и средства реализации мультизадачности в защищенном режиме процессора. Получить практические навыки по программированию и использованию этих средств.

2. Теоретические сведения

2.1. Мультизадачность

Под мультизадачностью подразумевают способность компьютера выполнять несколько задач одновременно. На самом деле процессор некоторое время выполняет один командный поток, затем быстро переключается на второй и выполняет его, переключается на третий и т.д. При этом при каждом переключении сохраняется контекст прерываемого потока, так что потом процессор сможет "безболезненно" продолжить выполнение прерванного потока команд. Благодаря высокому быстродействию создается иллюзия того, что все задачи выполняются одновременно (параллельно).

Для управления мультизадачностью нет специальных команд. Задачи переключаются командами FAR CALL, FAR JMP, INT, IRET. Однако при этом участвуют специальные дескрипторы: дескриптор сегмента состояния задачи (Task State Segment) и дескриптор шлюза задачи. Когда управление передается на один из таких дескрипторов, происходит переключение задачи. При переключении задачи процессор сохраняет (восстанавливает) свой контекст в сегменте состояния задачи (TSS). Селектор TSS выполняемой задачи хранится в регистре задачи (Task Register). При переключении задачи процессор может сменить LDT, что позволяет назначить каждой задаче свое адресное пространство, недоступное для других задач. Можно также перегрузить CR3, что позволяет применить для изолирования задач механизм страничного преобразования.

Дескриптор TSS относится к системным дескрипторам. Поле Type дескриптора TSS может содержать код 1001, если это доступный TSS, или 1011, если это занятый TSS, т.е. если задача активна в настоящий момент.

На рис. 11 представлен формат сегмента TSS для процессора i80386. Из рисунка видно, что в TSS предусмотрены поля для хранения сегментных регистров GS, FS, DS, SS, CS, ES. Имеется поле для хранения содержимого регистра LDTR, указывающего на локальную таблицу дескрипторов, распределённую для данной задачи. Для хранения содержимого 32-разрядных регистров используются поля

TSS, обозначенные на рис.11 как EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX, EFLAGS, EIP.

Поле CR3 хранит содержимое системного регистра CR3. Этот регистр является указателем на каталог таблиц страниц. Таким образом, каждая задача может иметь свой собственный каталог таблиц страниц, что позволяет выполнить изоляцию задач не только на уровне сегментов, но и на уровне страниц.

Битовая карта ввода/вывода (БКВВ)			
Дополнительная информация ОС			
Относительный адрес БККВ	0	T	64
0	LDTR		60
0	GS		5C
0	FS		58
0	DS		54
0	SS		50
0	CS		4C
0	ES		48
EDI			44
ESI			40
EBP			3C
ESP			38
EBX			34
EDX			30
ECX			2C
EAX			28
EFLAGS			24
EIP			20
CR3			1C
0	SS2		18
ESP2			14
0	SS1		10
ESP1			C
0	SS0		8
ESP0			4
0	LINK		0

Рис. 11. Сегмент TSS процессора i80386

TSS процессора i80386 содержит указатели на стеки для второго, первого и нулевого приоритетных колец. Это поля SS2:ESP2, SS1:ESP1, SS0:ESP0.

Поле LINK используется для ссылки на TSS, вызвавшей задачи при вложенном вызове задач, аналогично тому, как это было в процессоре i80286.

Бит T используется для отладки. Если он установлен в “1”, при переключении на задачу возникает отладочное исключение, которое может быть использовано системным отладчиком.

Для обеспечения безопасной работы системы необходимо ограничить доступ программам пользователя ко всем или по крайней мере к некоторым портам ввода/вывода. Злонамеренная программа, имеющая доступ к портам контроллера прямого доступа к памяти, может выполнить с помощью этого контроллера чтение или запись информации по любым физическим адресам. Процессор i80286 хранит в регистре флагов уровень привилегий I0PL, на котором разрешено выполнять команды ввода/вывода. С помощью этого механизма можно запретить непривилегированным программам выполнять команды ввода/вывода.

Однако такой способ защиты не слишком удобен. Некоторые порты ввода/вывода не только безопасны для использования, но и весьма полезны для обычных программ (например, порт системного динамика или принтера).

Битовая карта ввода/вывода процессора i80386 позволяет для каждой задачи определить порты, которые эта задача может использовать. То есть операционная система имеет возможность санкционировать любую задачу для использования любого набора адресов портов ввода/вывода. Если задача попытается обратиться к несанкционированному порту ввода/вывода, произойдёт исключение.

Сегмент TSS содержит поле, обозначенное на рис.11 как база карты ввода/вывода. Оно служит для указания расположения битовой карты ввода/вывода задачи, использующей данный TSS. Поле базы карты ввода/вывода указывает 16-разрядное смещение начала битовой карты ввода/вывода относительно TSS. Предел TSS должен определяться с учётом карты. Каждый бит в карте ввода/вывода соответствует адресу байта порта ввода/вывода (карта состоит из 64 Кбит для описания доступа к 65536 портам). После битовой карты должен располагаться байт OFFh.

При выполнении 16- или 32-разрядных операций ввода/вывода процессор проверяет все биты (2 или 4 бита), соответствующие адресу порта. Если проверяемый бит установлен в “1”, происходит исключение.

Для привилегированных программ, если уровень привилегий меньше или равен уровню I0PL, процессор не выполняет проверку битовой карты ввода/вывода. Чтобы полностью запретить задаче обращаться к портам ввода/вывода, достаточно установить базу карты ввода/вывода большей или равной пределу TSS. В этом случае любая команда ввода/вывода приведёт к генерации исключения.

Лишь значения первых 68h байт сегмента состояния задачи строго определены. Именно это число является минимальным размером TSS. Операционная система может по своему усмотрению устанавливать размер TSS и включать в сегмент TSS дополнительную информацию, необходимую для работы задачи и зависящую от конкретной операционной системы (например, контекст сопроцессора, указатели открытых файлов или указатели на именованные

конвейеры сетевого обмена). Включенная в этот сегмент информация автоматически заменяется процессором при выполнении команды CALL или JMP, селектор которой указывает на дескриптор сегмента TSS в таблице GDT (дескрипторы этого типа могут быть расположены только в этой таблице).

При переключении задачи с помощью прерывания или особого случая происходит автоматический возврат к прерванной задаче. Однако организуя вложение задач, необходимо помнить, что, в отличие от процедур, при переключении задачи в стек ничего не включается. Дескриптор TSS задачи, выполняемой в данный момент, помечается как занятый. При переключении на другую задачу с вложением (по INT или FAR CALL) дескриптор TSS остается помеченным. Переключиться на занятую задачу нельзя (возникает нарушение общей защиты - исключение №13).

Для переключения задач также действуют правила привилегий. По команде JMP или CALL можно переключиться на задачу, TSS которой менее привилегирован:

$$DPL_{TSS} >= \max(CPL,RPL).$$

Для особых случаев и прерываний это правило не действует. Если обработчик прерывания выполнен в виде отдельной задачи, то он может быть вызван независимо от значения CPL.

Не совсем удобно адресовать именно TSS для переключения задачи, т.к., во-первых, TSS могут быть размещены только в GDT (а в IDT или LDT - нет), а, во-вторых, если пользоваться только TSS, то каждую задачу мы "намертво" привязываем к определенному уровню привилегий (DPL_{TSS}), с которого она доступна для переключения. Этих недостатков лишены шлюзы задачи. Формат дескриптора шлюза задачи приведен на рис. 12.

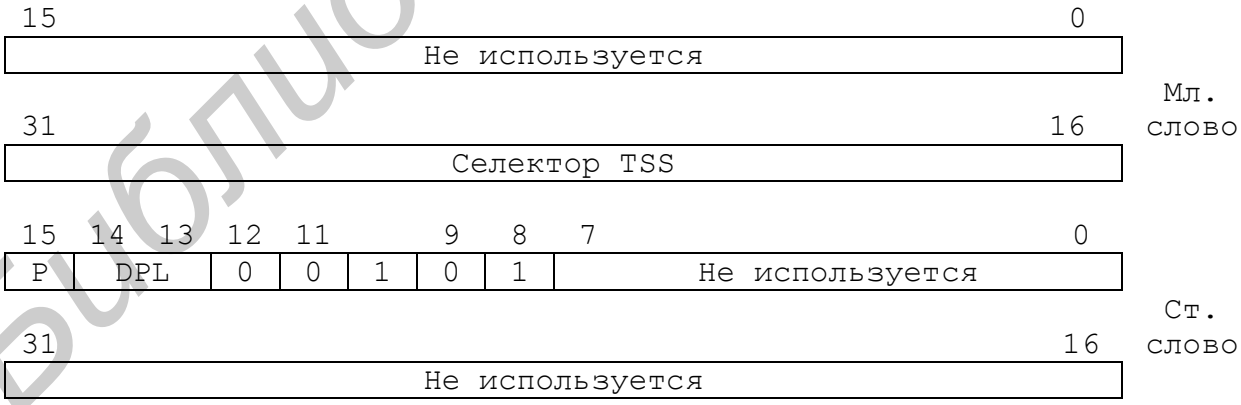

Рис. 12. Дескриптор шлюза задачи для процессора i80386

Шлюз задачи содержит селектор TSS. Шлюзы задач можно размещать и в IDT, что позволяет выполнять обработчики прерываний в виде отдельных задач, и

в LDT, что позволяет более гибко управлять переключением задач: для второй задачи первая может быть видна с одного уровня привилегий, а для третьей - с другого. Последняя возможность обеспечивается особым правилом привилегий: при переключении задачи через шлюз учитывается только $DPL_{шлюза}$, а DPL_{TSS} не играет роли, поэтому одной задаче может соответствовать множество шлюзов с различными DPL.

При переключении задач процессор выполняет следующие действия:

- выполняется команда CALL, селектор которой указывает на дескриптор сегмента типа TSS;
- в TSS текущей задачи сохраняются значения регистров процессора. На текущий сегмент TSS указывает регистр процессора TR, содержащий селектор сегмента;
- в TR загружается селектор сегмента TSS задачи, на которую переключается процессор;
- из нового TSS в регистр LDTR переносится значение селектора таблицы LDT в таблице GDT задачи;
- восстанавливаются значения регистров процессора (из соответствующих полей нового сегмента TSS);
- в поле селектора возврата заносится селектор сегмента TSS снимаемой с выполнения задачи для организации возврата к прерванной задаче в будущем.

Вызов задачи через шлюз происходит аналогично, добавляется только этап поиска дескриптора сегмента TSS по значению селектора дескриптора шлюза вызова.

3. Лабораторное задание

Написать программу, реализующую мультизадачность в защищенном режиме. Программа должна переключить процессор в защищенный режим, а затем запустить на выполнение 2-3 задачи, которые должны выполняться параллельно. Каждая задача выводит на экран свое сообщение. Задача выводит на экран часть сообщения, затем происходит переключение на другую задачу и т.д. Когда все задачи отработают, программа должна вернуть процессор в реальный режим.

4. Контрольные вопросы

1. Какая обязательная информация сохраняется в сегменте состояния задачи?
2. Какая дополнительная информация может быть сохранена в сегменте TSS?
3. Для чего используется битовая карта ввода/вывода?
4. Какие существуют в защищенном режиме способы переключения задач? Сравните их.
5. Какие действия выполняет процессор при переключении задач?

Литература

1. Григорьев В.Л. Архитектура и программирование арифметического сопроцессора. М.: Энергоатомиздат, 1991.
2. Фролов А.В., Фролов Г.В.. Защищенный режим процессоров Intel 80286, 80386, 80486: Практическое руководство по использованию защищенного режима. М.: Диалог-Мифи, 1993.
3. Григорьев В.Л. Микропроцессор i486. Архитектура и программирование. В 4 кн. Кн. 1: Программная архитектура. М.: ГРАНАЛ, 1993.
4. Григорьев В.Л. Микропроцессор i486. Архитектура и программирование. В 4 кн. Кн. 3: Устройство с плавающей точкой. М.: ГРАНАЛ, 1993.
5. Гук М. Процессоры Pentium II, Pentium Pro и просто Pentium. СПб.: Питер, 1999.
6. Ровдо А.А. Микропроцессоры от 8086 до Pentium III Xeon и AMD-K6K3. М.: ДМК, 2000.

Св. план, 2003 поз. 37

Учебное издание

Волорова Наталья Алексеевна
Дурович Александр Семенович

Архитектура вычислительных систем

Лабораторный практикум
для студентов специальности «Информатика»
всех форм обучения

Редактор Т.Н. Крюкова
Корректор Е.Н. Батурчик

Подписано в печать 02.05.2003. Формат 60x84 1/16.
Бумага офсетная. Печать ризографическая. Гарнитура «Таймс».
Усл. печ. л. 1,98. Уч.- изд. л. 1,8. Тираж 100 экз. Заказ 27.

Издатель и полиграфическое исполнение:
Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Лицензия ЛП № 156 от 30.12. 2002.
Лицензия ЛВ № 509 от 03.08. 2001.
220013, Минск, П. Бровки, 6