В. М. Михальчук
А. А. Ровдо
С. В. Рыжиков

МИКРОПРОЦЕССОРЫ 80x86, Pentium

АРХИТЕКТУРА, ФУНКЦИОНИРОВАНИЕ, ПРОГРАММИРОВАНИЕ, ОПТИМИЗАЦИЯ КОДА

В. М. Михальчук
А. А. Ровдо
С. В. Рыжиков

МИКРОПРОЦЕССОРЫ 80x86, Pentium

АРХИТЕКТУРА, ФУНКЦИОНИРОВАНИЕ, ПРОГРАММИРОВАНИЕ, ОПТИМИЗАЦИЯ КОДА

Минск
« БИТРИКС »
1994

ББК 32.973.2
М69
УДК 681.32

Производственно-практическое издание
Михальчук Виталий Михайлович
Ровдо Алексей Александрович
Рыжиков Сергей Владимирович
Микропроцессоры 80x86, Pentium
Архитектура, функционирование, программирование, оптимизация кода

Подписано в печать с оригинал макета 1.08.94г.
Формат 60x84 1/16. Гарнитура таймс. Печать офсетная.
Усл.-печ. л. 25. Тираж 5000 экз. Заказ 5519.
ООО " Битрикс ". Лицензия ЛВ№ 876.
220103 Минск, а/я 37.

Типография "Победа", 222310, г. Молодечно, ул. В. Тавлая, 11.

Михальчук В.М. и др.
М69 Микропроцессоры 80x86, Pentium : Архитектура, функционирование, программирование, оптимизация кода /В.М. Михальчук, А.А. Ровдо, С.В. Рыжиков. - Мн.:Битрикс, 1994. - 400 с.
ISBN 985-6164-01-x.

М 68103200000
ISBN 985-6164-01-x **ББК 32.973.2**

Предисловие

Предлагаемая читателю книга содержит последовательное описание архитектуры, системы команд и тонкостей программирования наиболее популярных в настоящее время микропроцессоров: 8086, 80286, 80386, 80486, Pentium, а также математических сопроцессоров 8087, 80287, 80387, 80487.

Поскольку микропроцессоры серии 80x86, Pentium программно совместимы сверху вниз, то все они описываются параллельно, с указанием, где это необходимо, возможных отличий. Такое построение книги позволяет быстро и корректно решать вопросы, возникающие при разработке универсального - работающего на всех процессорах серии, программного обеспечения.

Наибольшее внимание уделено рассмотрению процессоров 80486 и Pentium, описанию их системы команд, используемых методов адресации и защиты информации.

Микропроцессор Pentium представляет собой не только очередной шаг в повышении производительности и степени интеграции, но и содержит целый ряд принципиально новых подходов, реализованных в его архитектуре. Профессиональные разработчики программного обеспечения найдут в этой книге самое подробное изложение этих вопросов.

Кроме описания деталей архитектуры в книге также содержится глава посвященная вопросам разработки наиболее эффективных программ для 32-х разрядных микропроцессоров 80386, 80486, Pentium. Читатели найдут в ней подробное изложение и примеры реализации методов оптимизации кодов программ для наиболее быстрого их исполнения на процессорах 80386, 80486, Pentium.

Данная книга может быть полезна, как на первоначальном этапе ознакомления с современными микропроцессорами, так и как удобный в использовании справочник при повседневной работе. Для быстрого поиска необходимого материала служат: алфавитный указатель и специальные заголовки в верхней строке.

Авторы выражают глубокую признательность Д.А. Бондарю и В.В. Шленскому, чья помощь сделала возможным выход данной книги.

1. Введение

В таблице 1-1 приведены наиболее распространенные в настоящее время микропроцессоры и математические сопроцессоры, которые частично или полностью совместимы с процессорами фирмы Intel.

Как правило, фирмы производители заявляют о полной совместимости своих изделий с микропроцессорами Intel, однако, как показывает практика, поведение клонов может отличаться от поведения приборов Intel (например, в случае конкурирующих прерываний). Тем не менее, в подавляющем большинстве случаев все что изложено в данной книге о процессорах Intel верно и для их аналогов производства других фирм.

Для математических сопроцессоров ситуация иная, степень сходства здесь уже несколько ниже, чем в случае с микропроцессорами. Основные различия математических сопроцессоров лежат в их внутренней архитектуре. Ниже приводится краткая характеристика сопроцессоров некоторых фирм.

Внутреннюю архитектуру сопроцессоров Intel можно представить в виде следующих независимо работающих функциональных блоков: блока интерфейса шины, интерфейса данных и управляющей логики, а также блока математических вычислений. Основной процессор может передавать сопроцессору коды и данные следующей операции еще до того, как закончилось выполнение текущей. Блок интерфейса данных сопроцессора имеет специальный буфер, работающий по принципу FIFO (первым вошел - первым вышел). Это дает возможность организовать между основным процессором и сопроцессором как синхронную, так и асинхронную работу без циклов ожидания. Кроме этого, такой буфер позволяет синхронизировать работу процессора и сопроцессора тактируемых разной частотой. Например, процессор Intel386 с тактовой частотой 25Мгц может работать совместно с сопроцессором Intel387, тактовая частота которого 12Мгц.

В настоящее время существует два поколения математических сопроцессоров фирмы Intel. К первому относятся Intel287, 8087, а ко второму все остальные - Intel387DX, Intel387SX, Intel80287A и т.д. Дело в том, что сопроцессор 8087 и по сути его копия Intel287 создавались тогда, когда стандарт IEEE-754, содержащий описание форматов чисел с плавающей точкой и определение правил работы с ними, находился еще в стадии разработки. Это и привело к некоторым отклонениям в функционировании выше названных сопроцессоров от ныне действующей версии стандарта.

Таб. 1-1 Основные типы и модификации микропроцессоров и математических сопроцессоров

Наименование	Фирма	Разрядность шины данных внутр./внеш.	Адресуемая память/разрядность шины адреса	Встроенный КЭШ	Сопроцессор	Пониженное потребление	Внутреннее удвоение частоты	Комментарий
i8086	Intel	16/16	1МБ/20	нет	нет	нет	нет	
i8088	Intel	16/8	1МБ/20	нет	нет	нет	нет	
V20	NEC	16/8	1МБ/20	нет	нет	нет	нет	аналог i8088
V30	NEC	16/16	1МБ/20	нет	нет	нет	нет	аналог i8086
i80186	Intel	16/16	1МБ/20	нет	нет	нет	нет	2 таймера, 2 ПДП
i80188	Intel	16/8	1МБ/20	нет	нет	нет	нет	2 таймера, 2 ПДП
V40	NEC	16/8	1МБ/20	нет	нет	нет	нет	аналог i80188
V50	NEC	16/16	1МБ/20	нет	нет	нет	нет	аналог i80186
Intel286	Intel	16/16	16МБ/24	нет	нет	нет	нет	
V33	NEC	16/16	16МБ/24	нет	нет	нет	нет	аналог i80286
Intel386SX	Intel	32/16	16МБ/24	нет	нет	нет	нет	
Intel386DX	Intel	32/32	4ГБ/32	нет	нет	нет	нет	
Intel386SL	Intel	32/16	16МБ/24	нет	нет	есть	нет	контроллер КЭШ, SMM
38600DX	C&T	32/32	4ГБ/32	нет	нет	нет	нет	10% быстрее i386DX
Am386DXL	Amd	32/32	4ГБ/32	нет	нет	есть	нет	
Am386SXL	Amd	32/16	16МБ/24	нет	нет	есть	нет	
38600SX	C&T	32/16	16МБ/24	нет	нет	нет	нет	10% быстрее i386SX
38605DX	C&T	32/32	4ГБ/32	512	нет	есть	нет	режим Super State
38605SX	C&T	32/16	16МБ/24	512	нет	есть	нет	режим Super State
386SLC	IBM	32/16	16МБ/24	8K	нет	есть	нет	совм. по выв. с i386SX
486SLC2	IBM	32/16	16МБ/24	16K	нет	есть	есть	совм. по выв. с i386SX
486SLC3	IBM	32/16	16МБ/24	16K	нет	есть	f*3	совм. по выв. с i386SX
Cx486DLC	Cyrix	32/32	4ГБ/32	1K	нет	нет	нет	совм. по выв. с i386DX
Cx486SLC	Cyrix	32/16	16МБ/24	1K	нет	нет	нет	совм. по выв. с i386SX
Cx486SLC2	Cyrix	32/16	16МБ/24	1K	нет	есть	есть	совм. по выв. с i386SX
Intel486DX	Intel	32/32	4ГБ/32	8K	встр.	нет	нет	
Intel486SX	Intel	32/32	4ГБ/32	8K	нет	нет	нет	сопроцессор Intel487SX
Intel486DX2	Intel	32/32	4ГБ/32	8K	встр.	нет	есть	аналог Over Drive
Intel486DX4	Intel	32/32	4ГБ/32	8K	встр.	нет	f*3	с утроением частоты
Over Drive	Intel	32/32	4ГБ/32	8K	встр.	нет	есть	аналог Intel486DX2
Intel486SL	Intel	32/32	4ГБ/32	8K	нет	есть	нет	встроен контроллер КЭШ
Pentium	Intel	64,32/64	4ГБ/32	2*8K	встр.	нет	нет	режим SMM
i8087	Intel	16/16	-	-	да	нет	-	
i80287	Intel	16/16	-	-	да	нет	-	
i80c187	Intel	16/16	-	-	да	нет	-	IEEE-754
i80287A	Intel	16/16	-	-	да	нет	-	IEEE-754
i80287XL	Intel	16/16	-	-	да	да	-	IEEE-754
Intel387DX	Intel	32/32	-	-	да	нет	-	IEEE-754
Intel387SX	Intel	32/16	-	-	да	нет	-	IEEE-754
Intel487SX	Intel	32/32	-	-	да	нет	-	IEEE-754
83D87	Cyrix	32/32	-	-	да	нет	-	25% быстрее Intel387DX
83S87	Cyrix	32/16	-	-	да	нет	-	25% быстрее Intel387SX
82S87	Cyrix	16/16	-	-	да	нет	-	быстрее i80287A
2C87	ITT	16/16	-	-	да	нет	-	расширен. i80287A
3C87	ITT	32/32	-	-	да	нет	-	расширен. i80387DX
83c87	ULSI	32/32	-	-	да	нет	-	аналог i80387DX
83c87SX	ULSI	32/16	-	-	да	нет	-	аналог i80387SX
82c287	Amd	16/16	-	-	да	нет	-	аналог i80287
38700DX	C&T	32/32	-	-	да	нет	-	500% быстрее i80387DX
38700SX	C&T	32/16	-	-	да	нет	-	500% быстрее i80387SX
WTL3167	Weitek	32/32	-	-	да	нет	-	не аналог i80387
WTL4167	Weitek	32/32	-	-	да	нет	-	не аналог i80487

Второе же поколение сопроцессоров фирмы Intel уже полностью соответствует стандарту IEEE-754-1985. Помимо этого в сопроцессоры были введены и несколько дополнительных команд для прямого вычисления значений синуса и косинуса. Одним из существенных отличий двух поколений математических сопроцессоров можно считать их различное поведе-

ние при обработке значения "бесконечность". Согласно предварительному стандарту IEEE в сопроцессорах 8087 и Intel287 для обработки этого значения было предусмотрено два режима. В первом режиме - "проективном", который устанавливается сразу после инициализации этих сопроцессоров, существует только одно значение бесконечности. В альтернативном же режиме - "аффинном" - напротив, значения "минус" и "плюс" бесконечности различаются. Именно этот режим является единственно возможным для нового поколения математических сопроцессоров фирмы Intel. Кроме выше описанного, более ранние сопроцессоры поддерживают ряд форматов представления данных ("псевдо-нуль", "псевдо-бесконечность", "ненормализованное"), которые в сопроцессорах нового поколения отсутствуют. Кое-какие характерные отличия имеются у 8087, их суть в способе обработки исключительных ситуаций сопроцессора. Дело в том, что сопроцессоры, предназначенные для работы с микропроцессорами Intel286, Intel386 и т.п. сообщают о возникновении исключительной ситуации через специальный вывод ERROR#, активизация сигнала на котором вызывает прерывание 16 микропроцессора. В процессоре же 8086(8088) прерывание ошибки сопроцессора отсутствует. Поэтому сопроцессор сообщает об ошибках выставляя активный сигнал на выводе INT запроса прерывания, который (запрос) в свою очередь передается на вход NMI (немаскируемые прерывания) микропроцессора (такая схема реализуется в XT совместимых машинах). Для реализации такой системы сигнализации об ошибках в 8087 предусмотрено два дополнительных флага в управляющих регистрах и две дополнительные команды.

Intel387DX предназначен для работы в компьютерах на процессоре Intel386DX. В машинах на базе Intel386SX применяется Intel387SX. Связь между Intel386 и Intel387 осуществляется в 32-битном формате. Сопроцессор Intel387 позволяет использовать гибкое управление памятью, распознавая функции защиты, присущие Intel386. Если декодирующее устройство основного процессора распознает код операции, предназначенной для сопроцессора, то этот код записывается в командный регистр Intel387.

Однако совместно с 32-разрядным процессором Intel386 может применяться 16-разрядный сопроцессор Intel287, при этом, весь обмен между основным процессором и сопроцессором осуществляется по 16-разрядной шине. Процессор Intel386 определяет тип математического сопроцессора сразу после аппаратного сброса по состоянию сигнала на выводе ERROR#.

Как уже отмечалось, в настоящее время микросхемы математических сопроцессоров для IBM совместимых персональных компьютеров предла-

гают несколько фирм. В основном архитектура этих сопроцессоров копирует классическую архитектуру фирмы Intel. Рассмотрим некоторые из них.

Итак, сопроцессоры фирмы Cyrix: 83D87, предназначенный для работы с микропроцессором 386DX, и 83S87 - для работы с 386SX. Фирма гарантирует абсолютную совместимость своих изделий с сопроцессорами Intel. Тем не менее, производительность сопроцессоров Cyrix приблизительно на 25% выше чем у прототипов. Следует сказать также, что сопроцессоры фирмы Cyrix работают не только быстрее, но и точнее сопроцессоров Intel. Кроме уже упомянутых сопроцессоров этой фирмы в компьютерах иногда применяется модель 82S87, используемая совместно с Intel286.

Математические сопроцессоры фирмы IIT: 2C87 и 3C87, предназначенные для работы в 286-х и 386-х системах соответственно. Эти сопроцессоры имеют несколько расширенную архитектуру по сравнению с сопроцессорами Intel. Тридцать два 80-разрядных регистра данных в этих сопроцессорах разделены на четыре банка, из которых программно доступными являются только три. Однако при вычислении трансцендентных функций возможно ухудшение точности получаемых результатов. Это связано обычно с накоплением аппаратных ошибок при использовании встроенного матричного умножителя 4x4. По скорости работы сопроцессоры IIT несколько уступают сопроцессорам Cyrix.

Имеет смысл особо отметить сопроцессоры 38700SX и 38700DX (серия Super Math). Разработчики фирмы CHIPS & Technologies особое внимание уделили эффективности интерфейсов математических сопроцессоров Super Math, которые тем не менее обладают полной совместимостью с аналогичными микросхемами фирмы Intel. Главным усовершенствованием Super Math является модернизация их внутренней структуры. Это позволяет им при выполнении некоторых операций работать в 6 раз производительнее.

Совместимые с приборами Intel сопроцессоры выпускают и некоторые другие фирмы (ULSI, Amd и т.д.).

В современных 386-х компьютерах кроме Intel подобных сопроцессоров иногда применяются сопроцессоры несколько иной архитектуры - сопроцессоры Weitek. Причем они могут применяться как отдельно, так и параллельно со стандартным сопроцессором Intel387.

Первое существенное отличие сопроцессоров Weitek от всех других математических сопроцессоров - это то, что они используют другую, так

называемую memory mapped архитектуру. То есть сопроцессор для основного процессора представляется некоторой ограниченной областью адресного пространства. По сути, обращение к такому сопроцессору происходит через оперативную память. Область адресов сопроцессора лежит между C000:0000h и C000:FFFFh. Если старшие 16 бит адреса не соответствуют C000h, то сопроцессор Weitek игнорирует это сообщение. Биты адреса с 10 по 15 определяют команду для сопроцессора, а биты с 9 по 0 определяют регистр приемник и регистр источник (в сопроцессоре WTL3167 используется 32 80 разрядных регистра вместо 8 у Intel387). Следует отметить, что регистр для второго операнда всегда совпадает с регистром приемником, то есть регистром результата. Таким образом, загрузив операнды и код операции в область памяти, соответствующую сопроцессору, следующей командой можно уже считывать результат.

2. Прикладная архитектура

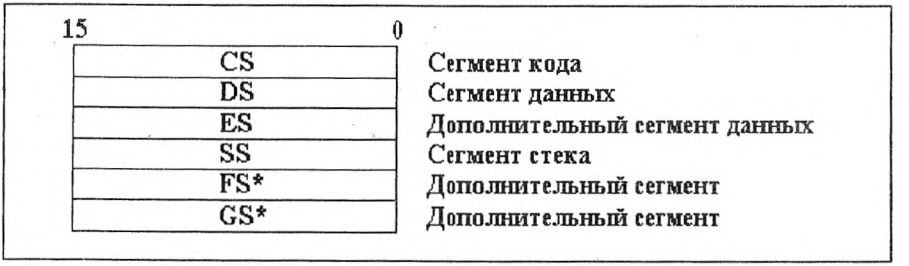
2.1 Регистры общего назначения

Рис. 2-1 Формат регистров общего назначения

* 32 разрядные регистры имеются только в микропроцессорах Intel386, Intel486 и Pentium, в 8086 (8088) и Intel286 все регистры 16 разрядные.

2.2 Сегментные регистры

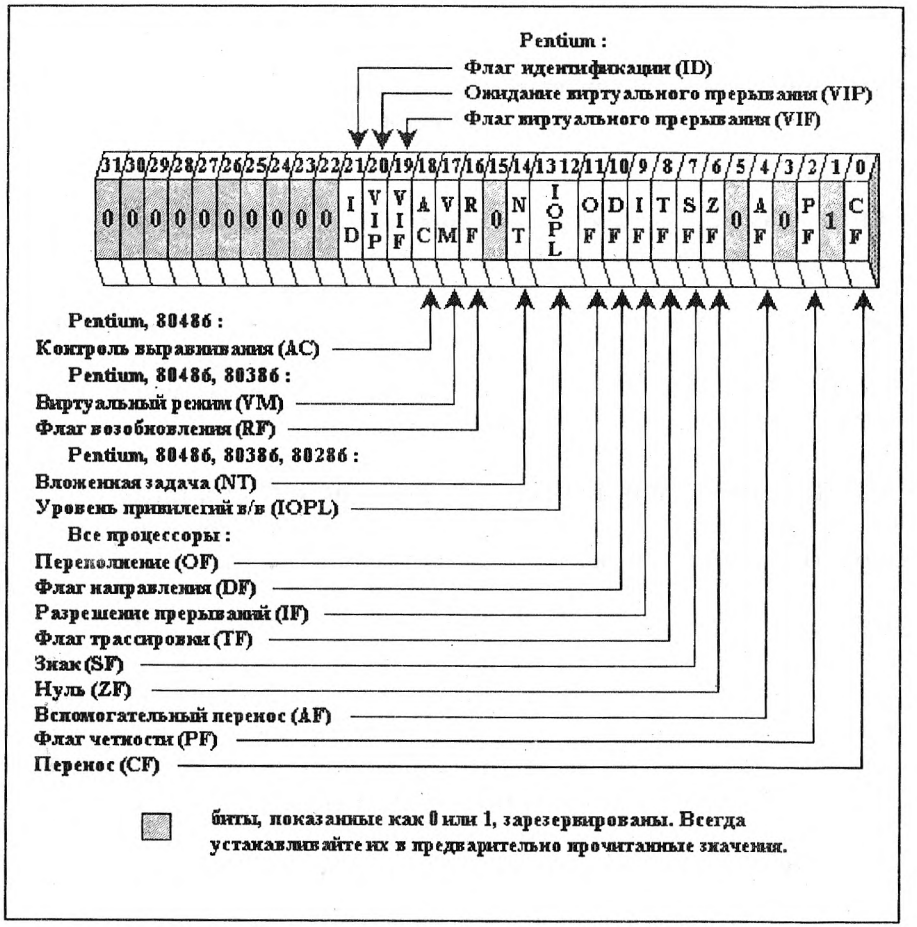
Рис. 2-2 Формат сегментных регистров



* Регистры FS и GS имеются только в Intel386, Intel486 и Pentium, в 8086 (8088) и Intel286 дополнительных сегментных регистров нет.

2.3 Регистр флагов (E)FLAGS

Рис. 2-3 Формат регистра флагов



ID (Флаг идентификации, бит 21) (Pentium)

Предназначен для проверки, поддерживается ли процессором команда CPUID. Если в программе можно установить и сбросить этот флаг, значит команда CPUID данным процессором поддерживается.

VIP (Ожидание виртуального прерывания, бит 20) (Pentium)

Флаг VIP, как и флаг VIF, позволяет каждой прикладной программе в многозадачном режиме иметь виртуальную версию флага IF. За дополнительной информацией по использованию этих флагов в режиме V86 и защищенном режиме обратитесь к Приложению H.

VIF (Виртуальное прерывание, бит 19) (Pentium)

Этот флаг является виртуальным подобием флага IF, используется совместно с VIP.

AC (Режим контроля выравнивания, бит 18) (Pentium, Intel486)

Установка флага AC и бита AM регистра CR0 включает контроль выравнивания при обращении к памяти. При этом генерируется особая ситуация контроля выравнивания, если происходит обращение к невыровненному операнду, например, к слову по нечетному адресу или к двойному слову по адресу не кратному четырем. Исключение контроля выравнивания генерируется только при уровне привилегий равном 3.

VM (Виртуальный режим, бит 17) (Pentium, Intel486, Intel386)

Установка флага VM переключает процессор в режим виртуального-8086 (специальное подмножество защищенного режима).

RF (Флаг возобновления, бит 16) (Pentium, Intel486, Intel386)

Флаг RF временно выключает обработку особых ситуаций отладки для того, чтобы команда, вызвавшая такую ситуацию, могла быть перезапущена и не стала бы причиной новой особой ситуации. Отладчик устанавливает этот флаг командой IRETD при возврате в прерванную программу. Команды POPF, POPFD и IRET на этот флаг не влияют.

NT (Вложенная задача, бит 14) (Pentium, Intel486, Intel386, Intel286)

Процессор устанавливает и проверяет этот флаг для контроля за прерванными задачами и при вызове процедур. Флаг NT влияет на действия, производимые командой IRET. Этот флаг может быть изменен командой POPF, POPFD, IRET. Некорректные изменения этого флага могут привести к возникновению различных особых ситуаций в прикладных программах.

IOPL (Уровень привилегий ввода/вывода, биты 13 и 12) (Pentium,..Intel286)

Уровень привилегий ввода/вывода используется в механизме защиты для управления доступом к адресному пространству ввода/вывода. Текущий уровень привилегий (CPL) совместно с IOPL определяет возможность по изменению поля IOPL командами POPF, POPFD и IRET.

OF (Флаг переполнения, бит 11)

Флаг переполнения сигнализирует о потере старшего бита результата в связи с переполнением разрядной сетки при работе со знаковыми числами. При сложении этот флаг устанавливается в 1, если происходит перенос в старший бит и нет переноса из старшего бита, или имеется перенос из старшего бита, но отсутствует перенос в него; в противном случае флаг OF устанавливается в 0. При вычитании он устанавливается в 1, когда возникает заем из старшего бита, но заем в старший бит отсутствует, либо имеется заем в старший бит, но отсутствует заем из него.

DF (Флаг направления, бит 10)

Флаг направления определяет порядок обработки цепочек в соответствующих командах (строковые команды: STOS, LODS, CMPS, OUTS и т.д.) - от меньших адресов к большим DF=0, от больших к меньшим DF=1.

IF (Флаг разрешения прерываний, бит 9)

При IF = 1, микропроцессор воспринимает (распознает) и соответственно реагирует на запрос прерывания по входу INTR; при IF = 0, прерывания по этому входу запрещаются и микропроцессор игнорирует поступающие запросы прерываний. Значение флага IF не влияет на восприятие внешних немаскируемых прерываний по входу NMI, а также внутренних программных прерываний, выполняемых по команде INT. В защищенном режиме CPL и IOPL определяют условия, при которых возможно изменение этого флага командами CLI, STI, POPF, POPFD и IRET.

TF (Флаг трассировки, бит 8)

При TF = 1, микропроцессор переходит в пошаговый режим работы, применяемый при отладке программ, когда автоматически генерируется внутреннее прерывание 1 после выполнения каждой команды. Прерывание отладки начнет генерироваться, если прикладная программа установит флаг TF с помощью команд POPF, POPFD или IRET.

SF (Флаг знака, бит 7)

Дублирует значение старшего бита результата, который при использовании дополнительного кода соответствует знаку числа.

ZF (Флаг нуля, бит 6)

Сигнализирует о получении нулевого результата операции.

AF (Флаг вспомогательного переноса, бит 4)

Фиксирует перенос (заем) из младшей тетрады, т.е. из бита 3 в старшую тетраду при сложении (вычитании). Используется только для двоично-десятичной арифметики, которая оперирует исключительно младшими байтами.

PF (Флаг четности, бит 2)

Фиксирует наличие четного числа единиц в младшем байте результата операции, может быть использован, например, для контроля правильности передачи данных.

CF (Флаг переноса, бит 0)

Флаг переноса фиксирует значение переноса (заема), возникающего при сложении (вычитании). Иногда используется и в других ситуациях.

2.4 Управляющие регистры (Pentium, Intel486, Intel386, Intel286)

Рис. 2-4 Формат управляющих регистров

Регистр CR0 (MSW в Intel286)

Первые 16 бит регистра CR0 называются словом состояния машины (MSW). Слово состояния машины впервые появилось в процессоре Intel286. В более поздних процессорах оно является лишь частью управляющего регистра CR0 и сохранено для совместимости.

PE (Разрешение защиты, бит 0) (Pentium, Intel486, Intel386, Intel286)

При сброшенном PE процессор работает в режиме реальной адресации. PE можно установить загрузкой MSW или CR0. Сбросить PE можно

только загрузкой CR0. В целях прямой совместимости с Intel286 бит PE не может быть сброшен командой LMSW.

MP (Слежение за сопроцессором, бит 1) (Pentium, Intel486, Intel386, Intel286)

В процессорах Intel286 и Intel386 используется совместно с битом TS для определения, будет ли команда WAIT генерировать особую ситуацию 7 ("сопроцессор отсутствует"); ловушка генерируется при установленных MP и TS. При запуске программ процессоров Intel286 и Intel386 на процессорах со встроенным сопроцессором (Intel486DX, Pentium) бит MP должен быть установлен; в случае процессора Intel486SX (нет встроенного сопроцессора) бит MP должен быть сброшен.

EM (Эмуляция сопроцессора, бит 2) (Pentium, Intel486, Intel386, Intel286)

Установка этого бита вызывает генерацию особой ситуации 7 ("сопроцессор отсутствует") всеми кодами команд сопроцессора. Значение EM не влияет на выполнение команды WAIT.

TS (Задача переключена, бит 3) (Pentium, Intel486, Intel386, Intel286)

Процессор устанавливает флаг TS при каждом переключении задач и анализирует его при поступлении команд сопроцессора. Команда CLTS очищает этот бит.

ET (Тип сопроцессора, бит 4) (Intel386)

При ET = 1, используется 32 битный протокол Intel387. При ET = 0, используется 16 битный протокол Intel287. При необходимости может быть сброшен или установлен загрузкой CR0. В целях прямой совместимости с Intel286 не изменяется командой LMSW.

NE (Ошибка сопроцессора, бит 5) (Intel486, Pentium)

Установка этого бита включает механизм сообщений об ошибках FPU. Если NE сброшен или на входе IGNNE# активный сигнал, то ошибки сопроцессора игнорируются. Если NE установлен и на входе IGNNE# сигнал не активен, то генерируется прерывание на выводе FERR#, который

может быть подключен к контроллеру прерываний (вывод FERR# аналогичен выводу ERROR# сопроцессоров Intel287 и Intel387).

WP (Защита записи, бит 16) (Intel486, Pentium)

Установка этого бита позволяет защитить страницы пользовательского уровня от изменения их программой супервизором, работающей на более высоком уровне привилегий. Если этот бит сброшен, то супервизор может осуществлять запись в страницы пользовательского уровня защищенные от записи.

AM (Маска выравнивания, бит 18) (Intel486, Pentium)

Этот бит позволяет осуществлять контроль выравнивания, если он установлен. Если бит AM сброшен, контроль выравнивания запрещен. Контроль выравнивания осуществляется только, если установлены бит AM, флаг AC, и $CPL_0 = 3$ (режим пользователя).

NW (Запрет сквозной записи, бит 29) (Intel486, Pentium)

Предназначен для управления встроенным КЭШ. Если этот бит сброшен, то сквозная запись и циклы недействительности КЭШ разрешены. Если бит установлен, то сквозная запись и циклы недействительности запрещены.

CD (Разрешение КЭШ, бит 30) (Intel486, Pentium)

Если CD = 0, работа внутреннего КЭШ разрешена, иначе - запрещена.

PG (Включение страничного механизма, бит 31) (Intel386, Intel486, Pentium)

Если PG = 1, страничный механизм включен, иначе - выключен.

Регистр CR2 (Pentium, Intel486, Intel386)

Если включен механизм страничной адресации (CR0.PG = 1), и генерируется особая ситуация 14 ("страничная ошибка"), то регистр CR2 будет содержать полный 32 разрядный линейный адрес, поступление которого в блок страничной адресации вызвало эту ошибку.

Регистр CR3 (Pentium, Intel486, Intel386)

Регистр содержит 20 старших бит физического адреса каталога страниц. Иногда CR3 называют регистром базы каталога страниц (PDBR). 12 младших бит регистра CR3 не используются в процессоре Intel386. В процессорах Pentium и Intel486 используется еще два следующих бита.

PWT (Сквозная запись страниц, бит 3) (Intel486, Pentium)

Бит используется для управления КЭШ. Программы могут управлять кэшированием как отдельных страниц, так и таблиц страниц посредством этого бита. При PWT=1, для текущей страницы обновление реализуется методом сквозной записи, при PWT = 0 - методом обратной записи.

PCD (Запрещение кэширования страниц, бит 4) (Intel486, Pentium)

Бит используется для управления КЭШ. Программы могут управлять кэшированием как отдельных страниц, так и таблиц страниц посредством этого бита. PCD=1 запрещает загрузку страницы в КЭШ память.

Регистр CR4 (Pentium)

VME (Расширение режима виртуального-8086, бит 0) (Pentium)

Установка в 1 этого бита включает поддержку флагов виртуальных прерываний VIF и VIP в режиме V86.

PVI (Виртуальные прерывания защищенного режима, бит 1) (Pentium)

Установка в 1 этого бита включает поддержку виртуальных прерываний в защищенном режиме. Это позволяет выполнять программы, предназначенные для уровня привилегий 0, на уровне привилегий 3.

TSD (Разрешение маркера времени, бит 2) (Pentium)

Установка TSD в 1 разрешает привилегированную команду RDTSC (Read from time stamp counter).

DE (Расширение отладки, бит 3) (Pentium)

Установка DE = 1 разрешает точки останова по вводу/выводу.

PSE (Расширение размера страниц, бит 4) (Pentium)

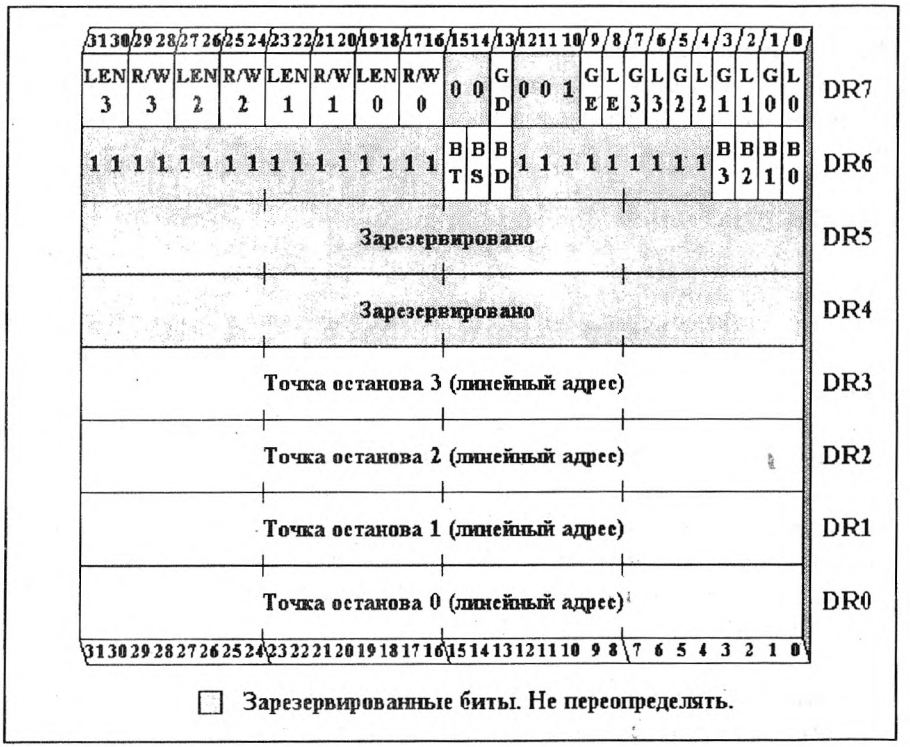
PSE = 1 разрешает использование страниц размером 4Мбайт.

MCE (Расширение контроля машины, бит 6) (Pentium)

MCE = 1 разрешает расширенный контроль.

2.5 Отладочные регистры (Pentium, Intel486, Intel386)

Рис. 2-5 Формат отладочных регистров



Регистры DR0, DR1, DR2, DR3 содержат линейные адреса четырех точек останова. Сравнение этих адресов производится до трансляции физического адреса. Каждая из четырех точек останова описывается отдельно в регистре DR7.

Регистры DR4 и DR5 зарезервированы. Процессоры Intel386 и Intel486 ассоциируют эти регистры с регистрами DR6 и DR7 соответственно. Когда расширение отладки выключено (CR4.DE = 0) процессор Pentium поступает аналогично. Но если включено расширение отладки (CR4.DE = 1), то все ссылки на регистр DR4 и DR5 приводят к генерации особой ситуации "неопределенный код операции" (#UD).

Регистр DR6

Регистр состояния отладки DR6 содержит информацию о том, какая из точек останова вызвала особую ситуацию отладки, и некоторую другую информацию.

B0..B3 (биты 0..3) (Pentium, Intel486, Intel386)

Устанавливаются при встрече соответствующей контрольной точки несмотря на то, разрешена для нее генерация особой ситуации отладки или нет.

BD (бит 13) (Pentium, Intel486, Intel386)

Устанавливается, если следующая команда будет читать или записывать один из отладочных регистров, если бит GD регистра DR7 установлен в 1, т.е. сигнализирует о том, что особая ситуация 1 ("прерывание отладки") имеет место из-за отказа при попытке обращения к регистрам отладки, при установленном бите DR7.GD = 1.

BS (бит 14) (Pentium, Intel486, Intel386)

Ассоциируется с флагом трассировки TF. Бит BS устанавливается, если особая ситуация отладки вызвана пошаговым режимом (EFLAGS.TF=1). Пошаговый режим является наиболее привилегированным. Когда BS установлен, некоторые из других битов состояния отладки также могут быть установлены.

BT (бит 15) (Pentium, Intel486, Intel386)

Ассоциируется с битом T в TSS (бит шлюза отладки). Процессор устанавливает бит BT перед входом в обработчик прерываний отладки, если происходит переключение задач, и в TSS новой задачи бит T установлен. Переключение происходит нормально, но перед выполнением первой ко-

манды новой задачи имеет место особая ситуация 1 ("прерывание отладки").

Регистр DR7

Регистр управления отладкой DR7 задает формы доступа к памяти или в/в для каждой точки останова.

Li/Gi (Pentium, Intel486, Intel386)

Биты разрешения, соответственно, локальных и глобальных контрольных точек. Li разрешают устанавливать контрольные точки для конкретных задач, не затрагивая другие. Gi влияют на все задачи. Биты Li автоматически очищаются при переключении задач.

LE, GE (биты 8, 9) (Pentium, Intel486, Intel386)

Управляют возможностью "точного соответствия контрольных точек данных". Если LE или GE установлен, сообщение о любой ловушке в контрольной точке по данным будет иметь место непосредственно после завершения команды, которая вызвала пересылку операнда. Если эти биты не установлены, сообщение об останове по данным может поступить спустя несколько команд или не поступить вообще. Рекомендуется устанавливать эти биты всякий раз, когда используются контрольные точки данных. При переключении задач LE очищается, GE - нет.

LENi (Pentium, Intel486, Intel386)

Поля LEN задают длину проверяемого интервала памяти для каждой контрольной точки (т.е. область, внутри которой условия останова могут сработать). Длина может составлять 1, 2 или 4 байта. Значение этих полей интерпретируется следующим образом:

00 - один байт;
01 - два байта;
10 - неопределено;
11 - четыре байта.

Если поле RWi = 00, то LENi должно быть равно нулю. В зависимости от LEN должно соблюдаться выравнивание контрольных точек по границам слов (LENi = 01) или двойных слов (LENi = 11).

RWi (Pentium, Intel486, Intel386)

Биты R и W задают, соответственно, для каждой контрольной точки в DR0 - DR3 тип доступа к памяти, при котором должна возникнуть особая ситуация.

00 - прерывание при выполнении команды;
01 - прерывание при записи данных;
10 - неопределено, если CR4.DE=0; если CR4.DE=1, прерывание при чтении/записи в/в (без вызова команды);(в Intel386, Intel486 неопределено);
11 - прерывание при чтении/записи данных (без вызова команды).

GD (бит 13) (Pentium, Intel486, Intel386)

Бит предназначен для защиты отладчика. Используется совместно с битом BD регистра DR6. Регистры отладки доступны только в реальном режиме или в защищенном режиме при уровне привилегий 0. Установка бита GD в реальном режиме и защищенном режиме на уровне привилегий 0 обеспечивает сверхзащиту всех обращений к регистрам отладки. При необходимости это гарантирует программному отладчику полный контроль над регистрами отладки. Когда бит GD установлен, попытка обратиться к любому из регистров отладки вызывает особую ситуацию 1 ("прерывание отладки"). При вызове процедуры обработки бит GD автоматически сбрасывается, что обеспечивает этой процедуре свободный доступ к регистрам отладки.

2.6 Тестовые регистры (Intel386, Intel486)

Рис. 2-6 Формат тестовых регистров TR3, TR4, TR5 (Intel486)

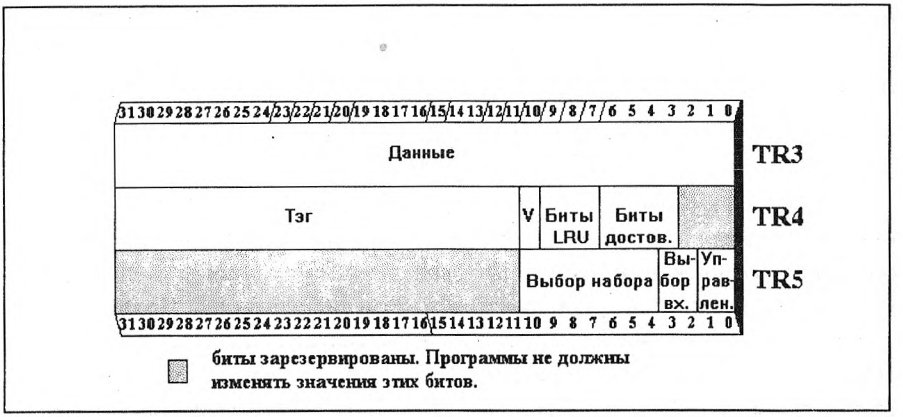
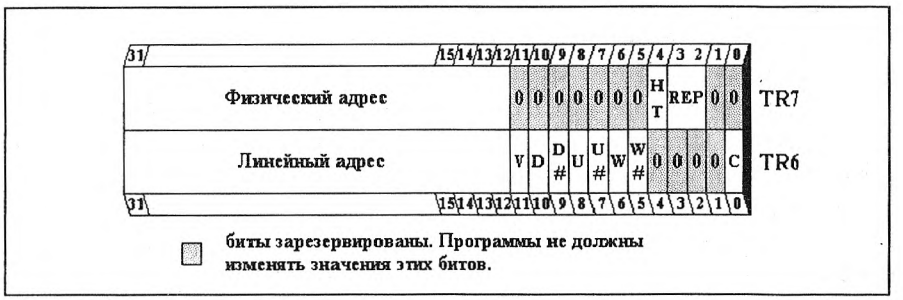


Рис. 2-7 Формат тестовых регистров TR6, TR7 (Intel386, Intel486)



Регистры TR3, TR4, TR5 используются в Intel486 при тестировании внутреннего КЭШ. (В процессоре Pentium механизм тестирования внутреннего КЭШ реализован иначе.)

Регистр TR3 (Intel486)

Регистр TR3 - регистр данных тестирования КЭШ, позволяет осуществить доступ к буферам чтения и заполнения. Данные, которые предполагается занести в буфер заполнения, должны быть вначале записаны в ре-

гистр TR3, а данные, читаемые из буфера чтения КЭШ, поступают в этот регистр. Регистр TR3 является 32 битным, в то время как буферы чтения и заполнения имеют размер 128 бит. В связи с этим, для загрузки всего буфера заполнения данные должны заносится в регистр TR3 и из него в буфер четырежды. Соответственно, для извлечения всего содержимого буфера чтения регистр TR3 должен быть прочитан четыре раза. Биты выбора входа в регистре TR5 определяют, которые 32 бита из 128 помещены в регистр TR3.

Регистр TR4 (Intel486)

Регистр TR4 - регистр состояния тестирования КЭШ, хранит информацию о значении тэга, битов LRU и битов достоверности строк в ходе операций тестирования. Перед тестовой записью данных в КЭШ в TR4 должны быть занесены значения тэга и бита верности (V). После чтения содержимого стороки КЭШ регистр TR4 содержит значения тэга и бита верности (V), связанных с выбранным входом, а также значения битов LRU и четырех битов достоверности, связанных с данным набором.

Регистр TR5 (Intel486)

Регистр TR5 - регистр управления тестированием КЭШ, определяет тип операции тестирования, набор и точку входа внутри набора, которые будут использованы. Биты выбора набора позволяют указать, к какому из 128 возможных наборов будет производиться обращение. Функционирование двух битов выбора входа зависит от состояния битов управления. Если выполняется обращение к буферам чтения или заполнения, то биты выбора точки входа определяют, с которой из 32 битовых областей буфера будет производиться операция. Когда область определена, эти биты указывают на один из четырех входов набора. Выбор определенной операции осуществляется каждый раз посредством указания нужной комбинации значений битов управления регистра TR5 в соответствии с приведенной ниже таблицей.

Биты управления		Операция	Функция битов выбора входов	Биты выбора набора
Бит 1	Бит 0			
0	0	Разрешение записи в буфер записи Разрешение чтения из буфера чтения	Выбор 32 битной области в буфере записи/чтения	
0	1	Выполнение записи в КЭШ	Выбор точки входа в наборе	Выбор набора для выполнения записи
1	0	Выполнение чтения из КЭШ	Выбор точки входа в наборе	Выбор набора для выполнения чтения
1	1	Выполнение очистки КЭШ		

При выполнении операций тестирования КЭШ ее обычная работа должна быть запрещена (CR0.CD = 1, CR0.NW = 1).

Регистр TR6 (Intel386, Intel486)

Регистр TR6 используется как командный регистр при тестировании TLB и содержит следующие поля.

C (бит 0) (Intel386, Intel486)

Командный бит, определяет один из двух видов команд: запись элементов в TLB (C = 0) или просмотр содержимого TLB (C = 1).

W#, W (биты 5, 6) (Intel386, Intel486)

Инверсный и прямой биты чтения/записи для/из элемента TLB.

U#, U (биты 7, 8) (Intel386, Intel486)

Инверсный и прямой биты доступности пользователю для/из элемента TLB.

D#, D (биты 9, 10) (Intel386, Intel486)

Инверсный и прямой биты мусора для/из элемента TLB.

V (бит 11) (Intel386, Intel486)

Бит достоверности для данного элемента TLB. TLB использует бит достоверности для идентификации элементов TLB, которые содержат достоверные данные (V = 1). Запись в CR3 приводит к сбросу всех битов достоверности.

Линейный адрес (биты 12..31) (Intel386, Intel486)

При записи в TLB, элемент TLB размещается по этому адресу. Этот элемент опирается на значение, установленное в TR7, и на величину, предварительно введенную в TR6 (команда чтения).

Регистр TR7 (Intel386, Intel486)

Регистр TR7 используется как регистр данных для TLB.

REP (биты 2, 3) (Intel386, Intel486)

При записи в TLB устанавливает, какой из четырех ассоциативных блоков используется. При чтении, если HT установился, REP указывает, в каком из четырех ассоциативных блоков найден признак; если HT не установился, то REP неопределен.

HT (бит 4) (Intel386, Intel486)

При чтении бит HT показывает, был ли поиск заданного элемента в TLB успешным (HT=1) или нет (HT=0). При записи в TLB необходимо HT=1.

Физический адрес (биты 12..31) (Intel386, Intel486)

Поле данных для TLB. При записи в TLB, элемент, определяемый линейным адресом в TR6, устанавливается в значение, содержащееся в этом поле. При чтении TLB, если HT установлен, в данное поле считывается значение физического адреса из элемента TLB; если HT не установлен, то значение этого поля неопределено.

Для записи элементов TLB необходимо:

1. Записать двойное слово в TR7, которое должно содержать корректное значение физического адреса, HT и REP полей (HT = 1, REP равен номеру ассоциативного блока).
2. Записать двойное слово в TR6, где указывается соответствующий линейный адрес и значения U, V, D и W (C = 0).

Примечание: не записывайте одинаковых признаков в разные элементы, результат неопределен.

Для чтения элементов TLB необходимо:

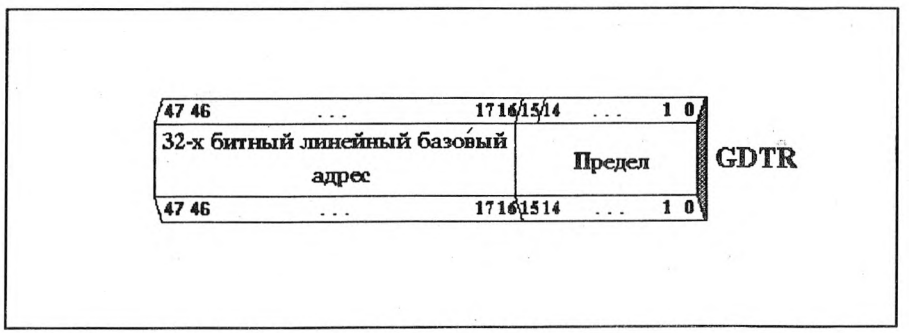
1. Записать двойное слово в TR6, где указать соответствующий линейный адрес и атрибут (C = 1).
2. Прочитать TR7. Если бит HT = 1, то остальные поля указывают состояние выбранного элемента TLB. Если HT = 0, то остальные поля не несут никакой информации, и их значения неопределены.

Примечание: при тестировании бит V используется подобно дополнительному адресному биту. Он обязательно должен быть установлен при чтении, чтобы не происходило обращения к недостоверным элементам. Результат чтения при V = 0 и наличии в TLB недостоверных элементов непредсказуем.

2.7 Системные регистры (Pentium, Intel486, Intel386, Intel286)

Регистр глобальной таблицы дескрипторов (GDTR)

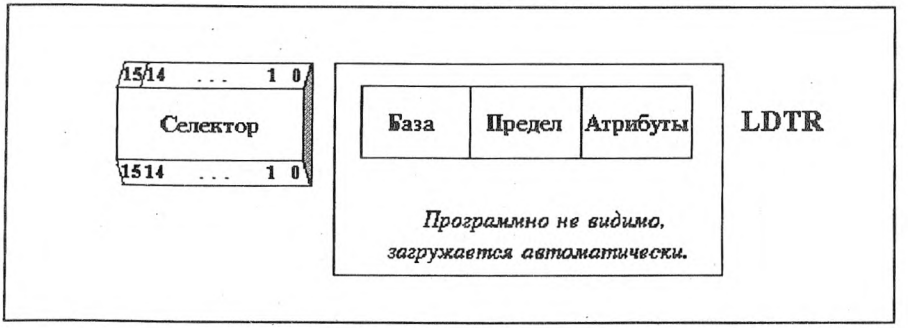
Рис. 2-8 Формат регистра глобальной таблицы дескрипторов



Регистр GDTR содержит 32 битный (24 битный для Intel286) базовый адрес и 16 битный предел глобальной таблицы дескрипторов (GDT).

Регистр локальной таблицы дескрипторов (LDTR)

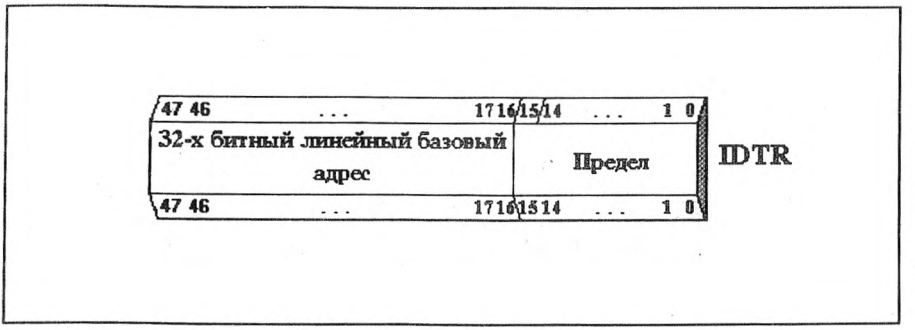
Рис. 2-9 Формат регистра локальной таблицы дескрипторов



Видимая часть регистра LDTR содержит только селектор дескриптора локальной таблицы дескрипторов (LDT). Сам дескриптор LDT загружается в скрытую часть LDTR из глобальной таблицы дескрипторов.

Регистр таблицы дескрипторов прерываний (IDTR)

Рис. 2-10 Формат регистра таблицы дескрипторов прерываний

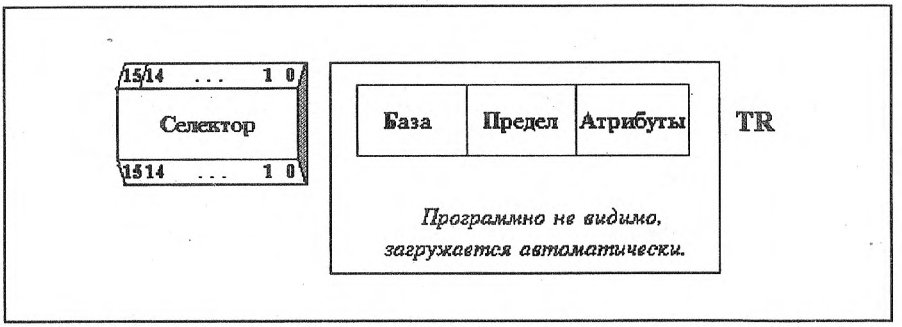


Регистр IDTR содержит 32 битный (24 битный для Intel286) базовый адрес и 16 битный предел таблицы дескрипторов прерываний (IDT). В ре-

альном режиме может быть использован для изменения местоположения таблицы векторов прерываний.

Регистр задачи (TR)

Рис. 2-11 Формат регистра задачи



Видимая часть регистра TR содержит селектор дескриптора сегмента состояния задачи (TSS). Сам дескриптор TSS загружается в скрытую часть TR из глобальной таблицы дескрипторов.

2.8 Специальные регистры процессора Pentium

Микропроцессор Pentium имеет несколько дополнительных, не имеющих аналогов в других процессорах регистров. Это так называемые особые регистры модели (Model Specific Registers). Они используются в некоторых специальных функциях процессора Pentium и в дальнейших модификациях процессоров могут не поддерживаться. Для чтения и записи этих регистров используются команды RDMSR и WRMSR. За информацией об этих регистрах обратитесь к техническому руководству фирмы Intel.

2.9 Состояние процессора после инициализации

Таб. 2-1 Состояние процессора после сброса

Регистр	Pentium		Intel486	Intel386	Intel286	8086
	RESET	INIT				
EFLAGS (FLAGS)[1]	00000002h	00000002h	00000046h	00000000h	(0002h)	
EIP(IP)	0000FFF0h	0000FFF0h	0000FFF0h	0000FFF0h	(FFF0h)	(FFF0h)
CR0(MSW)	60000010h	Прим.[2]	60000010h	7FFFFFE0h	(FFF0h)	-
CR2/CR3/ CR4	00000000h	00000000h	00000000h	00000000h	-	-
CS	F000h База = FFFF0000h Предел = FFFFh AR=93h	F000h База = FFFF0000h Предел = FFFFh AR=93h	F000h База = FFFF0000h Предел = FFFFh AR=93h	F000h База = FFFF0000h Предел = FFFFh AR=93h	F000h База = F0000h Предел = FFFFh AR=93h	F000h
SS, DS, ES, FS, GS	0000h База = 00000000h Предел = FFFFh AR=82h	0000h База = 00000000h Предел = FFFFh AR=82h	0000h База = 00000000h Предел = FFFFh AR=82h	0000h База = 00000000h Предел = FFFFh AR=82h	0000h База = 0000h Предел = FFFFh AR=82h	
EDX[3]	0000x5xxh	0000x5xxh	0000x4xxh	0000x3xxh		
EAX	0[4]	0[4]	0[4]	C51BB653h		
EBX, ECX, ESI, EDI, EBP, ESP	00000000h	00000000h	00000000h	00000000h		
GDTR, LDTR, IDTR	База = 00000000h Предел = FFFFh AR = 82h	База = 00000000h Предел = FFFFh AR = 82h	База = 00000000h Предел = FFFFh AR = 82h	База = 00000000h Предел = FFFFh AR = 82h	База = 0000h Предел = FFFFh AR=82h	-
DR0, DR1, DR2, DR3	0	0	0	0	-	-
DR6	FFFF0FF0h	FFFF0FF0h	FFFF1FF0h	FFFF1FF0h	-	-
DR7	00000400h	00000400h	00001400h	00001400h	-	-
Time Stamp Counter	0	не изменяется	-	-	-	-
Control and Event Select	0	не изменяется	-	-	-	-
TR12	0	не изменяется	-	-	-	-

Другие MSR	неопред.	не изменяются	-	-	-	-
Встроенный КЭШ данных, команд и TLB	недействит.	недействит.	недействит.	-	-	-

Примечания:

1. Старшие 10 (14 для Intel486 и Intel386) бит регистра EFLAGS неопределены. Неопределенные биты зарезервированы. Программы не должны изменять состояние этих битов.
2. В CR0 биты CD и NW не изменяются, бит 4 устанавливается в 1, остальные биты сбрасываются.
3. В процессорах Intel386 и старше регистр EDX содержит информацию о типе процессора. DH содержит 3, 4 или 5 в зависимости от процессора: Intel386, Intel486 или Pentium, соответственно. Несколько отличные значения могут быть в переработанных вариантах процессоров, например, для Intel386SX это 23h. Регистр DL содержит модификацию.
4. Регистр EAX содержит 0, только если все внутренние тесты прошли успешно, иначе EAX не нулевой.

2.10 Встроенный КЭШ. (Pentium, Intel486)

Таб. 2-2 Режимы работы КЭШ.

CD	NW	Pentium	Intel486
0	0	Нормальный, наиболее быстрый режим работы КЭШ: - информация, которая есть в КЭШ, выбирается только оттуда; - если информации нет в КЭШ, при чтении она туда заносится; - состояние линий КЭШ задается сигналом на выводе WB/WT#; - при записи модифицируется КЭШ; - команды недействительности поддерживаются.	Нормальный, наиболее быстрый режим работы КЭШ: - информация, которая есть в КЭШ, выбирается только оттуда; - если информации нет в КЭШ, при чтении она туда заносится; - при записи модифицируется КЭШ и внешняя память; - команды недействительности поддерживаются.
0	1	Неопределенная операция #GP(0).	Неопределенная операция #GP(0).

1	0	КЭШ выключен. Содержимое сохраняется заблокированным. Содержимое памяти сохраняется: - информация, которая есть в КЭШ, выбирается только оттуда; - если информации нет в КЭШ, при чтении она туда не заносится; - состояние линий КЭШ задается сигналом на выводе WB/WT#; - при записи модифицируется КЭШ; - команды недействительности поддерживаются.	КЭШ выключен. Содержимое сохраняется заблокированным. Содержимое памяти сохраняется: - информация, которая есть в КЭШ, выбирается только оттуда; - если информации нет в КЭШ, при чтении она туда не заносится; - при записи модифицируется КЭШ и внешняя память; - команды недействительности поддерживаются.
1	1	КЭШ выключен. Содержимое памяти не сохраняется: - информация, которая есть в КЭШ, выбирается оттуда; - если информации нет в КЭШ, при чтении она туда не заносится; - при записи модифицируется КЭШ, но не модифицируется память; - команды недействительности поддерживаются; - при записи, линии из состояния Exclusiv переключаются в состояние Modified; - циклы анализа и недействительности не влияют на состояние и содержимое КЭШ. Это состояние после сброса.	КЭШ выключен. Содержимое памяти не сохраняется: - информация, которая есть в КЭШ, выбирается оттуда; - если информации нет в КЭШ, при чтении она туда не заносится; - при записи модифицируется КЭШ, но не модифицируется память; - команды недействительности поддерживаются; - циклы анализа и недействительности не влияют на состояние и содержимое КЭШ. Это состояние после сброса.

Примечание: КЭШ процессора Pentium позволяет два режима работы: со сквозной записью или с обратной записью. КЭШ процессора Intel486 работает со сквозной записью.

3. Системная архитектура

3.1 Прерывания и особые ситуации

Существует два источника прерываний и два источника особых ситуаций.

Прерывания:

1. Маскируемые прерывания - принимаются на выводе INTR CPU. Если флаг IF установлен, маскируемые прерывания не проходят.
2. Немаскируемые прерывания - принимаются на выводе NMI CPU. Механизм запрета немаскируемых прерываний отсутствует.

Особые ситуации:

1. Определенные процессором особые ситуации. Разделяются на ошибки, ловушки и сбои.
2. Программные прерывания. Могут быть вызваны командами INTO, INT3, INTn, BOUND.

Таб. 3-1 Прерывания и особые ситуации.

Название	Номер	Тип	Источник	Возвращаемый адрес указывает на команду, вызвавшую особую ситуацию?	Генерируется ли код ошибки?	Процессор, в котором впервые появилось
Деление на нуль	0	Ошибка	Команды DIV и IDIV	Да	Нет	8086
Отладка	1	*[1]	Любые	*[1]	Нет	8086
Немаскируемое прерывание	2	Прер.	NMI	Да	Нет	8086
Точка останова	3	Лов.	Команда INT 3	Нет	Нет	8086
Переполнение	4	Лов.	Команда INTO	Нет	Нет	8086
Проверка границ	5	Ошибка	Команда BOUND	Да	Нет	80186

Неопределенная операция	6	Ошибка	Зарезервированные коды операций	Да	Нет	Intel286
Сопроцессор отсутствует	7	Ошибка	Команды ESC и WAIT	Да	Нет	Intel286
Двойная ошибка	8	Сбой	Любые команды	Да	Да (всегда нуль)	Intel286
Неправильный TSS[3]	10	Ошибка	Команды JMP, CALL IRET, прерывания и особые сит.	Да[2]	Да	Intel286
Сегмент не присутствует[3]	11	Ошибка	Любая команда, изменяющая сегмент	Да[2]	Да	Intel286
Ошибка стека	12	Ошибка	Стековые операции	Да	Да	Intel286
Общая защита	13	Ошибка /Лов.[4]	Любые ссылки на код или данные	Да	Да	Intel286
Страничная ошибка[3]	14	Ошибка	Любые ссылки на код или данные	Да	Да (спец. формат)	Intel386
Зарезервировано	15					
Превышение сегмента сопроцессором	9	Сбой	Команды ESC и WAIT	Нет	Нет	Intel286 Intel386 только
Ошибка сопроцессора	16	Ошибка[5]	Команды ESC и WAIT	Да	Нет	Intel286
Контроль выравнивания[3]	17	Ошибка	Любые ссылки на данные	Да	Да (всегда нуль)	Intel486
Контроль машины	18		Зависит от модели	-	Зависит от модели	Penium
Программные прерывания	0-225	Лов.	Команды INT n	Нет	Нет	8086

Примечания:

1. Особая ситуация отладки может быть как ошибкой, так и ловушкой. Отладчик должен определить это по значению в регистре DR6.
2. Команда, вызвавшая ошибку, перезапускаема не всегда.
3. Только для защищенного и V86 режимов. В реальном зарезервированы.
4. Все команды, вызвавшие ошибки общей защиты, перезапускаемы. Если ошибка вызвана прерыванием, то прерванная программа может быть продолжена, но возможна потеря прерывания.

5. Ошибки сопроцессора выявляются на первой команде ESC или WAIT после команды ESC, вызвавшей ошибку.

Дополнение:

Ошибка - сохраняются значения CS:IP, указывающие на команду, вызвавшую ошибку.

Ловушка - сохраняются значения CS:EIP, указывающие на команду, которая будет выполняться вслед за командой, вызвавшей ловушку; например, если ловушка произошла во время команды JMP, то сохраненные значения CS:EIP указывают на команду, являвшуюся целью команды JMP.

Сбой - это особая ситуация, которая также допускает точную локализацию вызвавшей ее команды, но не допускает перезапуска; сбои используются для сообщений о некоторых ошибках, таких как: технические неисправности и наличие некорректных значений в системных таблицах.

Прерывание 0 (Деление на нуль)

Возникает, если в командах DIV или IDIV делитель равен нулю.

Прерывание 1 (Прерывание отладки)

Генерируется при ряде условий. Может быть как ошибкой, так и ловушкой в зависимости от причины:

точка останова по команде	- ошибка;
точка останова по данным	- ловушка;
общий поиск	- ошибка;
пошаговое	- ловушка;
точка останова по переключению задач	- ловушка.

Процессор не сохраняет код ошибки для этого прерывания. Обработчик должен проверять отладочные регистры для выяснения причины прерывания.

Прерывание 3 (Точка останова)

Генерируется командой INT3. Сохраненные значения CS:EIP указывают на байт, следующий за INT3.

Прерывание 4 (Переполнение)

Генерируется при выполнении команды INTO, если установлен флаг OF.

Прерывание 5 (Нарушение границ) (Intel286, Intel386, Intel486, Pentium)

Генерируется, если при выполнении команды BOUND процессор обнаруживает, что операнд выходит за заданные границы.

Прерывание 6 (Неопределенный код операции) (#UD) (Intel286, Intel386, Intel486, Pentium)

Возникает, когда в управляющем блоке процессора обнаруживается некорректный код. Сохраненное значение CS:EIP указывает на недопустимый код или на его префикс. Ошибка может быть обработана в текущей задаче. Это прерывание также возникает при любом появлении неправильного операнда (например, межсегментный переход, использующий регистровый операнд). В процессорах Intel386, Intel486, Pentium существует еще одно условие возникновения этого прерывания - использование префикса LOCK с командами не допускающими блокировки. Кроме этого, коды D6h и F1h зарезервированы Intel и не вызывают ошибки.

Прерывание 7 (Сопроцессор отсутствует) (#NM) (Intel286, Intel386, Intel486, Pentium)

Генерируется в следующих ситуациях:

- процессор выполняет команду ESC, и CR0.EM=1;
- процессор выполняет команду WAIT (MP=1) или ESC, и CR0.TS=1.

т.е., если поступает команда ESC, а сопроцессор отсутствует или поступают команды ESC или WAIT, а состояние сопроцессора не отвечает необходимому для данной задачи (произошло переключение задач).

В процессорах Intel286 и Intel386 бит MP регистра CR0 используется совместно с битом TS для определения, должна ли команда WAIT генерировать особую ситуацию. Для программ, запускаемых на процессорах Pentium, Intel486DX и Intel486SX (при наличии сопроцессора Intel487SX), должно быть MP = 1. Для Intel486SX (когда сопроцессор Intel487SX отсутствует) - MP = 0.

Прерывание 8 (Двойная ошибка) (#DF) (Intel286, Intel386, Intel486, Pentium)

При выполнении одной команды может произойти две особых ситуации. (Примером такой ситуации может служить исключение общей защиты

на уровне 3, сопровождаемое исключением 11 из-за отсутствия требуемого сегмента.) При этом в некоторых случаях генерируется прерывание 8.

Таб. 3-2 Условия, при которых генерируется двойная ошибка

Особая ситуация, возникшая первой	Особая ситуация, возникшая второй		
	Безопасная (1,2,3,4,5,6,7,16)	Способствующая (0,10,11,12,13)	Страничная (14)
Безопасная(1,2,3,4,5,6,7,16)	Нет	Нет	Нет
Способствующая(0,10,11,12,13)	Нет	Да	Нет
Страничная(14)	Нет	Да	Да

Примечание: безопасные, способствующие, страничные - классы прерываний и особых ситуаций.

Код ошибки помещается в стек обработчика двойной ошибки; поэтому этот код всегда нулевой. Вызвавшая ошибку команда не перезапускаема. Если во время перехода на обработчик двойной ошибки возникает еще какая-либо особая ситуация, то процессор переходит в режим останова, откуда может быть выведен только сигналами NMI или RESET. (На практике оказывается, что даже не все процессоры фирмы Intel ведут себя именно так, некоторые просто сбрасываются, некоторые могут, как ни в чем не бывало, продолжить работу.)

Прерывание 9 (Превышение сегмента сопроцессором) (Intel286, Intel386)

Генерируется при проверке границ сегмента, производящейся при каждом обращении сопроцессора к памяти, в случае, если произошел выход за пределы сегмента при попытке чтения/записи второго или следующих слов операнда. Прерывание может обрабатываться в той же задаче, но команда, вызвавшая эту ошибку, не перезапускаема. Процессор не сохраняет никакой информации о вызвавшей прерывание команде. Для выяснения причины прерывания необходимо обратиться к специальным регистрам сопроцессора, содержащим последнюю выполненную команду и адрес запрошенного операнда. После возникновения этого прерывания не могут быть выполнены никакие команды типа WAIT или ESC (кроме FINIT) до тех пор, пока не будет устранена причина сбоя, или сопроцессор не будет инициализирован.

Прерывание 9 не поддерживается процессорами Intel486 и Pentium. В тех случаях, когда Intel287 и Intel387 генерируют прерывание, Intel486 и Pentium просто сбрасывают команду. Некоторые случаи, однако, требуют осторожности. Ошибки памяти (особенно страничные ошибки) могут потеряться, если они происходят в FLDENV или FRSTOR в то время, как процессор осуществляет переключение задач. Рекомендуется, чтобы область сохранения состояния сопроцессора находилась в той же странице, что и TSS.

Прерывание 10 (Неправильный TSS) (#TS) (Intel286, Intel386, Intel486, Pentium)

Генерируется при переключении задач, если новый TSS, на который указывает шлюз задачи, является недопустимым. Недопустимыми считаются следующие случаи:

Индекс кода ошибки	Описание
Сегмент TSS	Предел TSS меньше 67h (2Bh для Intel286)
Сегмент LDT	Неправильная LDT или LDT не присутствует
Сегмент стека	Селектор стекового сегмента выходит за границы таблицы дескрипторов
Сегмент стека	Сегмент стека не разрешен для записи
Сегмент стека	Поле RPL селектора стекового сегмента несовместимо с CPL
Сегмент стека	DPL стекового сегмента несовместим с CPL
Кодовый сегмент	Селектор кодового сегмента выходит за границы таблицы дескрипторов
Кодовый сегмент	Кодовый сегмент не является исполняемым
Кодовый сегмент	DPL несогласованного кодового сегмента не равен CPL
Кодовый сегмент	DPL согласованного кодового сегмента больше CPL
Сегмент данных	Селектор сегмента данных выходит за границы таблицы дескрипторов
Сегмент данных	Сегмент данных не доступен для чтения

Бит EXT кода ошибки показывает, была ли особая ситуация вызвана событием внешним по отношению к программе (т.е., если внешнее прерывание осуществляет переключение задач с некорректным TSS).

Это прерывание может быть вызвано, как в контексте старой задачи, так и уже в новой (т.е. регистр TR уже будет загружен новым селектором TSS). Первый случай возникает при проверке присутствия нового TSS.

Ошибка не может быть обработана в текущей задаче.

Прерывание 11 (Сегмент не присутствует) (#NP) (Intel286, Intel386, Intel486, Pentium)

Генерируется, если процессор обнаруживает, что бит присутствия в дескрипторе сегмента сброшен. Это может произойти в следующих случаях:

- при любой команде загрузки регистров CS, DS, ES, FS или GS (загрузка SS вызывает стековую ошибку);
- при загрузке регистра LDTR командой LLDT (загрузка LDTR во время переключения задач вызывает прерывание 10);
- при попытке использования дескриптора задачи, помеченного как неприсутствующий.

Эта ошибка рестартируема и может обслуживаться в рамках породившей ее задачи.

Код ошибки помещается в стек. Бит EXT кода ошибки устанавливается, если прерывание вызвано внешним событием. Бит IDT кода ошибки устанавливается, если код ошибки ссылается на элемент IDT (т.е. команда INT ссылается на неприсутствующий шлюз).

Прерывание 12 (Ошибка стека) (#SS) (Intel286, Intel386, Intel486, Pentium)

Генерируется в следующих случаях:

- при нарушении установленных границ в командах, использующих регистр SS (POP, PUSH, ENTER, LEAVE, а также команды типа MOV AX,[BP] MOV AX,SS:[EAX] и т.п.);
- при попытке загрузить SS дескриптором, который помечен как неприсутствующий (может случиться в командах CALL, IRET, LSS, MOV, POP).

При возникновении стековой ошибки, процессор помещает код ошибки в стек обработчика. Если прерывание произошло во время переключения задач посредством межуровнего CALL из-за того, что произошло переполнение стека, или сегмент стека не присутствует, то код ошибки содержит селектор сегмента, ставшего причиной ошибки; в других случаях код ошибки всегда нулевой.

Команда, вызвавшая это прерывание, рестартируема во всех случаях. (На практике, в реальном режиме при превышении границ стекового сегмента, разные процессоры могут сброситься, зависнуть, перейти в режим останова и пр.)

Прерывание 13 (Общая защита) (#GP) (Intel286, Intel386, Intel486, Pentium)

Все нарушения защиты, которые не учитываются в других особых ситуациях, генерируют прерывание 13. Такими нарушениями могут быть:

- выход за границы сегмента при использовании CS, DS, ES, FS или GS;
- выход за границы сегмента при ссылках на таблицу дескрипторов;
- передача управления в сегмент, который не является исполняемым;
- запись в только читаемый сегмент данных или кода;
- чтение из только исполняемого кодового сегмента;
- загрузка регистра SS селектором только читаемого сегмента (кроме случая, когда это происходит во время переключения задач);
- загрузка SS, DS, ES, FS или GS селектором системного сегмента;
- загрузка DS, ES, FS или GS селектором только исполняемого кодового сегмента;
- загрузка SS селектором исполняемого сегмента;
- использование DS, ES, FS или GS, когда в них загружен нулевой селектор;
- переключение на занятую задачу;
- нарушение правил привилегий;
- превышение командой размера 15 байт (Intel386, Intel486, Pentium), такое может случиться при неправильном использовании префиксов;
- прерывание через шлюз прерывания или задачи в режиме V86 на уровень привилегий отличный от 0 (Intel386, Intel486, Pentium);
- попытка записи единицы в зарезервированные биты CR4 (Pentium);
- если начальный адрес операндов сопроцессора выходит за границу сегмента (Intel486, Pentium).

При возникновении ошибки общей защиты процессор помещает код ошибки в стек обработчика. Если ошибка произошла при загрузке неправильного дескриптора, то код ошибки - это селектор этого дескриптора, во всех остальных случаях код ошибки нулевой.

В реальном режиме прерывание 13 генерируется при нарушении границы 0FFFFh в сегментах CS, DS, ES, FS, GS.

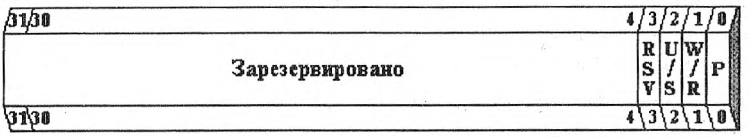
Прерывание 14 (Страничная ошибка) (#PF) (Intel386, Intel486, Pentium)

Генерируется, если страничный механизм активизирован (CR0.PG=1), и при трансляции линейного адреса в физический возникает одна из следующих ситуаций:

- элемент таблицы страниц или каталога страниц, используемый при трансляции адреса, имеет нулевой бит присутствия, т.е. нужная таблица страниц или страница не присутствует в физической памяти;
- процедура не располагает уровнем привилегий, достаточным для доступа к выбранной странице.

Обработчик страничной ошибки получает информацию о ее причине из двух источников: кода ошибки, помещаемого в стек (код страничной ошибки имеет специальный формат, описанный ниже (рис. 3-1), и содержимого регистра CR2, который содержит линейный адрес, вызвавший ошибку.

Рис. 3-1 Формат кода страничной ошибки



P — 0 - ошибка вызвана неприсутствием страницы
1 - ошибка вызвана нарушением защиты уровней привилегий

W/R — 0 - ошибка произошла при чтении
1 - ошибка произошла при записи

U/S — 0 - ошибка произошла, когда процессор находился в режиме супервизора
1 - ошибка произошла, когда процессор находился в режиме пользователя

RSV — 1 - ошибка произошла из-за обнаружения единицы в одном из зарезервированных битов элемента таблицы или каталога страниц (только Pentium, в других процессорах бит RSV неопределен)
0 - в остальных случаях

Прерывание 16 (Ошибка сопроцессора) (#MF) (Intel286, Intel386, Intel486, Pentium)

Это прерывание сигнализирует об ошибке, возникшей во время выполнения команды сопроцессором. Прерывание 16 генерируется только, если CR0.NE=1.

Если CR0.NE=1, то прерывание 16 генерируется непосредственно во время выполнения команды сопроцессора, следующей за командой, вызвавшей ошибку.

Если NE = 0 (и сигнал на входе IGNNE# неактивен), то незамаскированные ошибки сопроцессора (с помощью регистра управления сопроцессора можно замаскировать все исключения) приводят к остановке процессора перед выполнением следующей за вызвавшей ошибку команды плавающей арифметики (если это не управляющая команда) или команды WAIT. При этом, процессор ждет освобождения сопроцессора, а сопроцессор в свою очередь ожидает обслуживания исключения. Вывести процессор из этого режима возможно только внешним прерыванием, которое обычно формируется аппаратурой по появлению сигнала на выходе FERR# (Intel486, Pentium) или ERROR#(Intel387, Intel287). Сигнал на этом выводе появляется независимо от значения бита NE при возникновении исключения в сопроцессоре.

Если NE = 0 (но сигнал на входе IGNNE# активен), то процессор игнорирует ошибку и продолжает выполнение программы.

Прерывание 17 (Контроль выравнивания) (#AC) (Intel486, Pentium)

Генерируется при попытке доступа к невыровненным операндам. Выравнивание необходимо для следующих типов данных.

Таб. 3-3 Условия контроля выравнивания

Тип данных	Адрес должен быть кратен
WORD	2
DWORD	4
Short-real (Короткое вещественное - 32 бита)	4
Long-real (Длинное вещественное - 64 бита)	8
Temp-real (Временное вещественное - 80 бит)	8
Селектор	2
32 битный частичный указатель	2
32 битный прямой указатель	4
48 битный псевдо-дескриптор	4

Область сохранения для FSTENV/FLDENV	4 или 2, определяется размером операнда
Область сохранения для FSAVE/FRSTOR	4 или 2, определяется размером операнда
Строка бит	4

Для активизации проверки выравнивания должны выполняться следующие условия:

CR0.AM = 1;
EFLAGS.AC = 1;
CPL = 3.

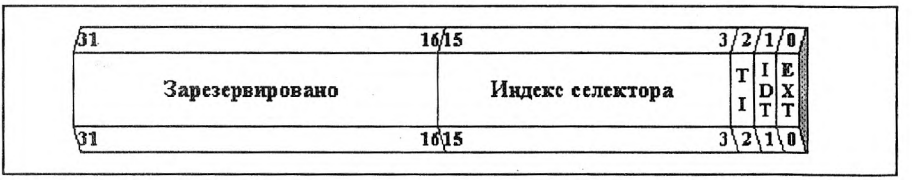
Прерывание 18 (Контроль машины) (Pentium)

Данное прерывание поддерживается только процессором Pentium и в дальнейших модификациях процессоров может быть исключено. Оно предназначено для выявления ошибок паритета и других неисправностей аппаратуры. Обработчик определяет причину прерывания по содержимому особого регистра Mashine Check Type. За дополнительной информацией обратитесь к техническому описанию Intel.

Код ошибки

Код ошибки заносится в стек обработчика особой ситуации при возникновении таковой.

Рис. 3-2 Формат кода ошибки



Код ошибки похож на селектор сегмента, однако вместо поля RPL, он содержит два однобитных поля EXT и IDT.

Бит EXT устанавливается, если причиной особой ситуации является внешнее по отношению к прерванной программе событие (внешние прерывания).

Бит IDT устанавливается, если индексная часть кода ошибки указывает на дескриптор шлюза прерывания. Если IDT = 0, то бит TI показывает, ссылается ли код ошибки на GDT (TI=0) или LDT (TI=1).

В некоторых случаях код ошибки может быть нулевым или вообще отсутствовать.

Процессоры Intel486 и Intel386 помещают в стек код ошибки как 16 битное значение, но изменяют ESP на 4. Pentium сохраняет 32 битное значение с нулевыми старшими битами.

Рис. 3-3 Стек после вызова прерывания или особой ситуации

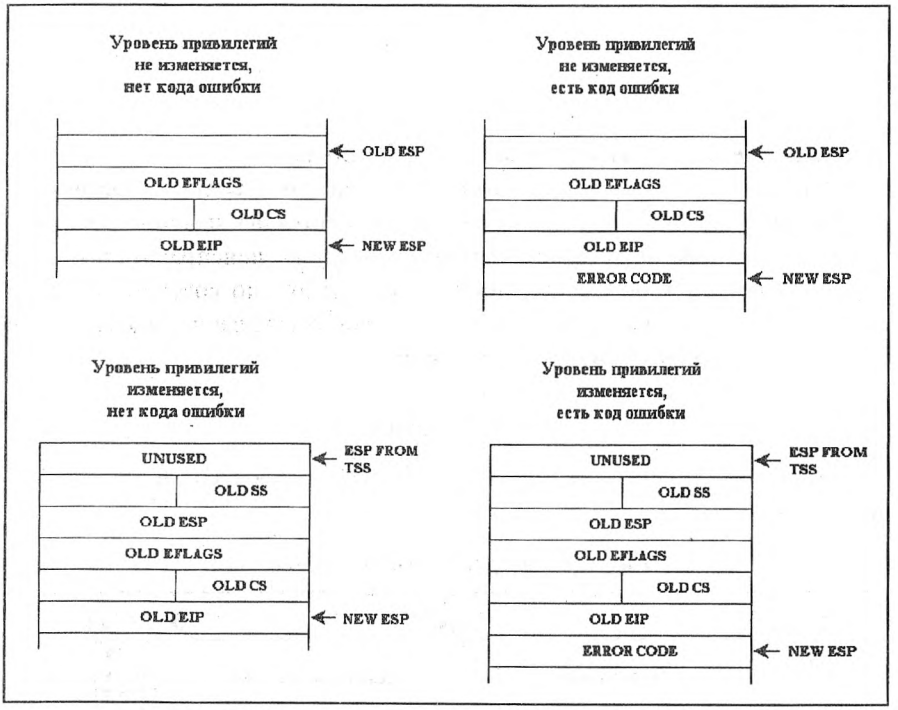
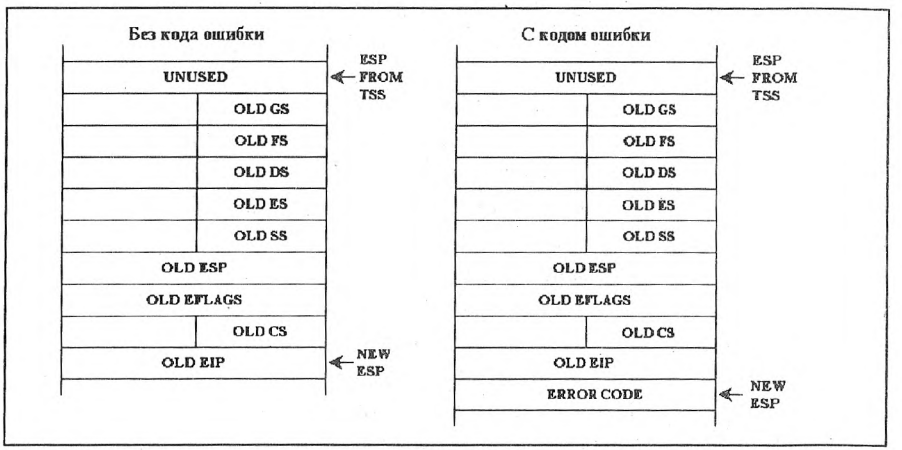


Рис. 3-4 Стек после вызова прерывания или особой ситуации в режиме V86



Приоритет прерываний и особых ситуаций

При возникновении конкуренции прерываний и особых ситуаций они обрабатываются в порядке наивысшего приоритета. (На практике, отложенные прерывания могут иногда просто сбрасываться, в зависимости от модели процессора.)

Таб. 3-4 Приоритет прерываний и особых ситуаций

Приоритет	Класс	Описание
Высший	1	Ловушки в предыдущей команде: - точки останова; - ловушки отладки (кроме описанных ниже).
	2	Внешние прерывания: - NMI; - маскируемые прерывания.
	3	Ошибки при вызове следующей команды: - точка останова для кода; - выход за границу сегмента; - страничная ошибка.

	4	Ошибки при декодировании следующей команды: - неопределенный код операции; - длина кода команды превышает 15 байт; - сопроцессор недоступен.
	5	Ошибки при выполнении команды: - общее определение; - ошибка сопроцессора (для предыдущей команды сопроцессора); - прерывание по переполнению; - границы (BOUND); - плохой TSS; - сегмент не присутствует; - стековая ошибка; - общая защита; - страничная ошибка по данным; - проверка выравнивания.
Низший		

3.2 Дескрипторы (Intel286, Intel386, Intel486, Pentium)

Ниже на рис. 3-5..3-12 представлены форматы всех возможных типов дескрипторов для Pentium, Intel486, Intel386. Дескрипторы процессора Intel286 аналогичны за исключением того, что старшее слово старшего двойного слова дескриптора для Intel286 описано как зарезервированное и при запуске программ Intel286 на 32 разрядных процессорах должно быть нулевым.

Рис. 3-5 Формат дескриптора

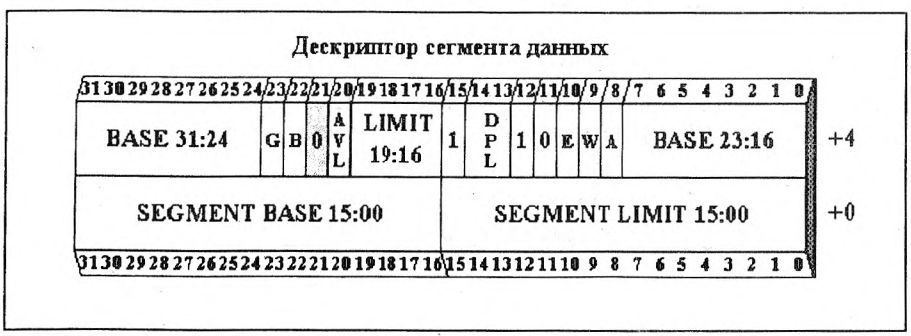


Рис. 3-6 Формат дескриптора

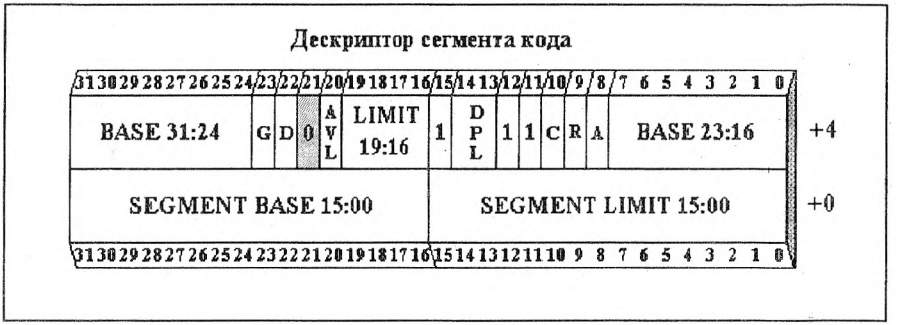


Рис. 3-7 Формат дескриптора

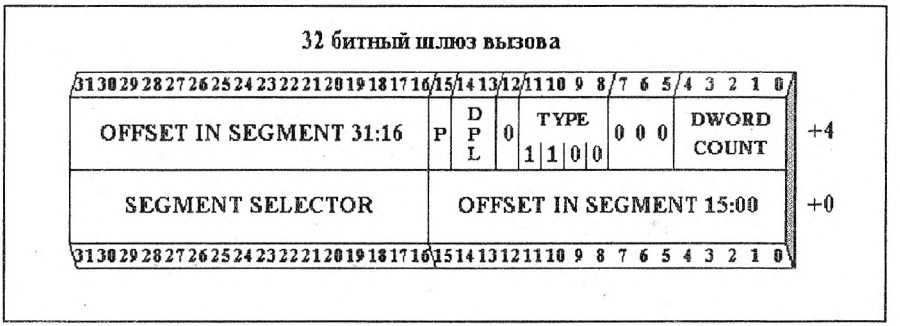
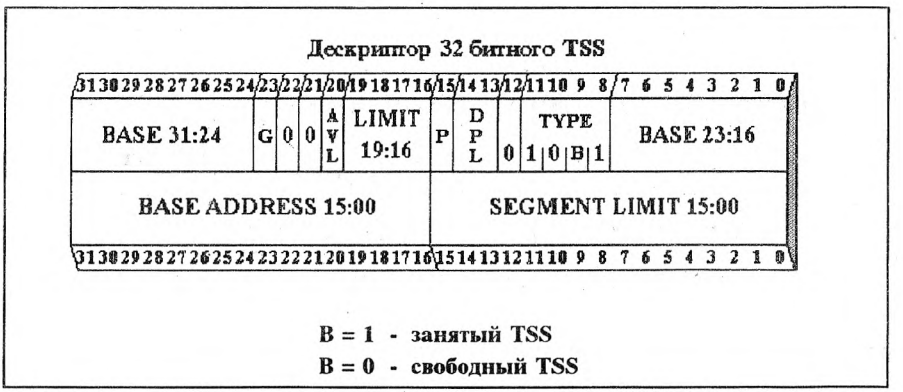


Рис. 3-8 Формат дескриптора



B = 1 - занятый TSS

B = 0 - свободный TSS

Рис. 3-9 Формат дескриптора

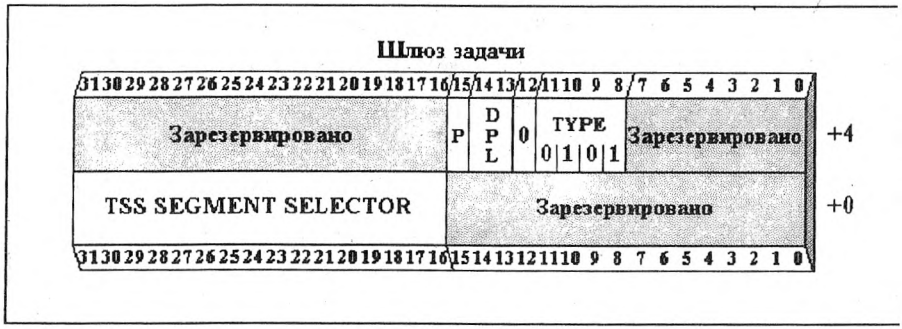


Рис. 3-10 Формат дескриптора

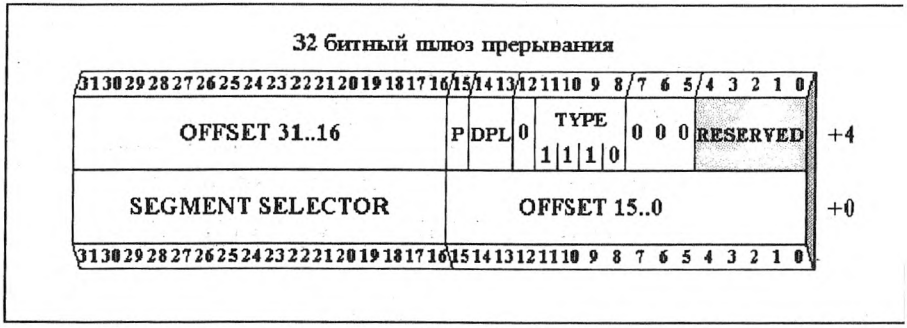
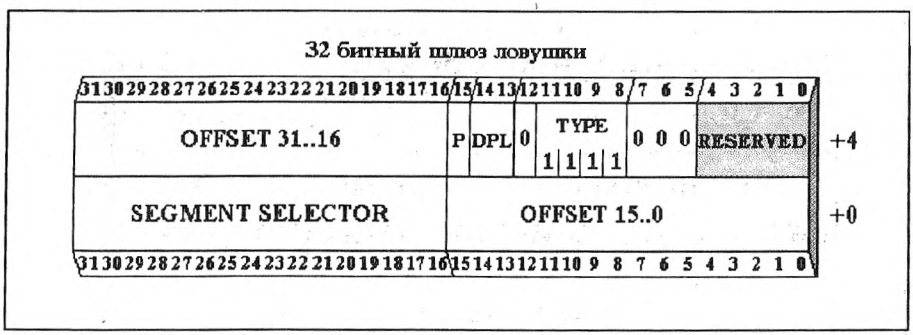


Рис. 3-11 Формат дескриптора



BASE (База) (Intel286 - 24 бита; Intel386, Intel486, Pentium - 32 бита)

Задает положение сегмента в 4 Гбайтном (16М для Intel286) адресном пространстве. Для максимального быстродействия рекомендуется, чтобы значение базы было выровнено по 16 байтным границам. Для дескрипторов Intel286 определены только младшие 24 бита этого поля.

LIMIT (Предел) (Intel286 - 16 бит; Intel386, Intel486, Pentium - 20 бит)

Задает размер сегмента. Процессор может интерпретировать это поле двумя способами в зависимости от значения поля G (гранулярность).

Если G = 0, то предел задает размер сегмента в байтах (от 1 байта до 1 Мбайта) (всегда для Intel286).

Если G = 1, то предел задает размер сегмента в 4 Кбайтных параграфах (от 4Кбайт до 4 Гбайт).

Для сегментов расширяющихся вверх логический адрес может принимать любое значение от 0 до LIMIT. Для сегментов расширяющихся вниз (стековые сегменты) смысл поля LIMIT переворачивается, т.е. логический адрес может принимать любые значения кроме значений от 0 до LIMIT.

Для дескрипторов Intel286 определены только младшие 16 бит этого поля.

G (Гранулярность) (Intel386, Intel486, Pentium)

Бит гранулярности задает масштаб поля LIMIT. Когда G=0, предел задается в байтах, если G=1 - в 4 Кбайтных блоках.

S (Системный, бит 12 старшего двойного слова дескриптора) ((Intel286, Intel386, Intel486, Pentium)

Задает тип дескриптора. Если S=0, то дескриптор считается системным. Если S=1, то это дескриптор сегмента кода или данных.

D (Разрядность по умолчанию) (Intel386, Intel486, Pentium)

Задает разрядность операндов и адресов, принимаемую по умолчанию для сегмента (разрядность по умолчанию может быть изменена префиксом размера операнда и префиксом размера адреса). Если D=1, то устанавливается режим 32 битных операндов и 32 битного адреса. Если D=0, то адреса и операнды 16 битные.

B (Размерность) (Intel386, Intel486, Pentium)

Управляет следующими аспектами стековых операций.

1. Размером регистра указателя стека. Если B=1, в стековых операциях используется 32 битный ESP. Если B=0 - 16 битный SP.
2. Верхней границей расширяемого вниз сегмента. В расширяемых вниз сегментах нижняя граница задается полем LIMIT, а верхняя равна FFFFFFFFh, если B=1, и FFFFh, если B=0.

DPL (уровень привилегий дескриптора) (Intel286, Intel386, Intel486, Pentium)

Задает уровень привилегий сегмента, который (уровень) используется механизмом защиты для контроля доступа к сегменту.

P (Присутствие) (Intel286, Intel386, Intel486, Pentium)

Бит присутствия сегмента в физической памяти. Если этот бит не установлен, процессор генерирует особую ситуацию 11 ("сегмент не присутствует"), когда селектор дескриптора загружается в сегментный регистр.

A (Доступ) (Intel286, Intel386, Intel486, Pentium)

Бит доступа устанавливается процессором для индикации того, что сегмент был загружен в сегментный регистр. Предварительно очистив бит A, программное обеспечение может позже проверить, осуществлялся ли доступ к сегменту.

Примечание: процессоры Intel486 и Pentium осуществляют цикл записи в память нового значения бита A=1, только если он не установлен; Intel286 и Intel386 делают это независимо от того, в каком значении бит A находился.

R (Разрешение чтения) (Intel286, Intel386, Intel486, Pentium)

R = 1 - чтение возможно;

R = 0 - чтение запрещено.

W (Разрешение записи) (Intel286, Intel386, Intel486, Pentium)

W = 1 - запись возможна;

W = 0 - запись запрещена.

C (Согласование) (Intel286, Intel386, Intel486, Pentium)

Кодовый сегмент может быть согласованным или несогласованным. Передача управления из сегмента с меньшим уровнем привилегий в сегмент с большим уровнем привилегий требует, чтобы сегмент, в который передается управление, был согласованным, иначе генерируется особая ситуация общей защиты (кроме случая использования ловушки задачи).

E (Расширяемость) (Intel286, Intel386, Intel486, Pentium)

Задает направление расширения сегмента.

E = 1 - сегмент расширяется вниз (стек);

E = 0 - сегмент расширяется вверх.

(D)WORD COUNT (Счетчик (двойных) слов) (Intel286, Intel386, Intel486, Pentium)

Для 32 битного шлюза задает количество двойных слов, копируемых из стека вызывающей процедуры в стек вызываемой. В случае 16 битного шлюза, задает количество слов, копируемых из стека вызывающей процедуры в стек вызываемой.

OFFSET (Смещение) (Intel286, Intel386, Intel486, Pentium)

Смещение точки входа в целевом сегменте.

SELECTOR (Селектор) (Intel286, Intel386, Intel486, Pentium)

Селектор целевого сегмента или (для шлюза задачи) селектор TSS целевой задачи.

TYPE (Тип биты 11..8 старшего двойного слова дескриптора) (Intel286, Intel386, Intel486, Pentium)

Поле тип содержит биты, которые задают назначение (тип) дескриптора.

TYPE	Описание
----------	Intel286, Intel386, Intel486, Pentium
0000	Зарезервировано
0001	Свободный 16 битный TSS
0010	Локальная таблица дескрипторов (LDT)
0011	Занятый 16 битный TSS
0100	16 битный шлюз вызова
0101	Шлюз задачи
0110	16 битный шлюз прерывания
0111	16 битный шлюз ловушки
----------	Intel386, Intel486, Pentium
1000	Зарезервировано
1001	Свободный 32 битный TSS
1010	Зарезервировано
1011	Занятый 32 битный TSS
1100	32 битный шлюз вызова
1101	Зарезервировано
1110	32 битный шлюз прерывания
1111	32 битный шлюз ловушки

3.3 Селекторы (Intel286, Intel386, Intel486, Pentium)

Рис. 3-12 Формат селектора сегмента

Индекс (биты 15..3) (Intel286, Intel386, Intel486, Pentium)

Указывает на нужный дескриптор в таблице дескрипторов. Процессор умножает значение этого поля на 8 (размер дескриптора) и прибавляет к базовому адресу таблицы дескрипторов из регистра GDTR или LDTR. Полученное значение и является адресом дескриптора.

TI (Бит указания таблицы) (бит 2) (Intel286, Intel386, Intel486, Pentium)

Задает, какую таблицу дескрипторов необходимо использовать. TI = 0 - GDT или TI = 1 - LDT.

RPL (Запрашиваемый уровень привилегий) (биты 1, 0) (Intel286, Intel386, Intel486, Pentium)

Определяет уровень привилегий, запрашиваемый селектором (наивысшие привилегии имеет уровень 0).

3.4 Сегмент состояния задачи (TSS)

Рис. 3-13 32-битный TSS (Pentium, Intel486, Intel386)

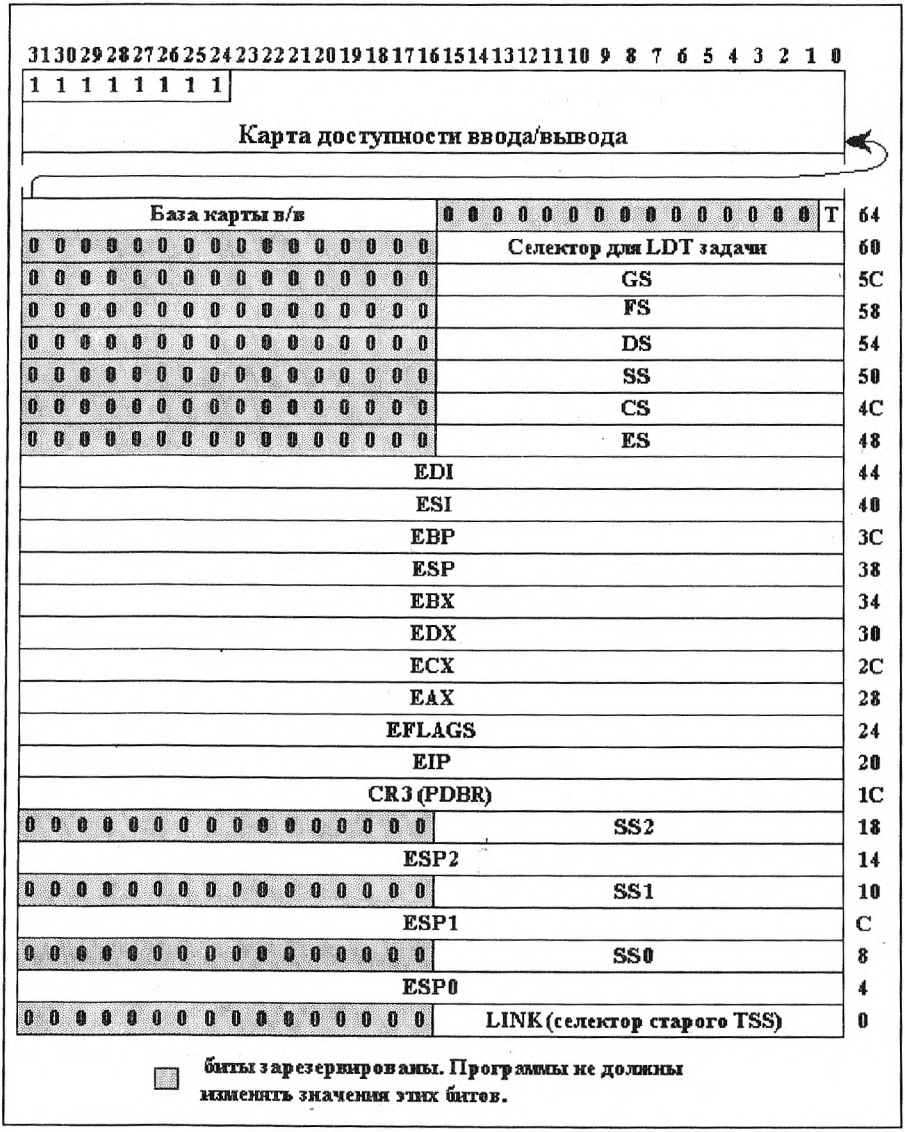


Рис. 3-14 16-битный TSS (Pentium, Intel486, Intel386, Intel286)

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
Селектор для LDT задачи	2A
DS	28
SS	26
CS	24
ES	22
DI	20
SI	1E
BP	1C
SP	1A
BX	18
DX	16
CX	14
AX	12
FLAGS	10
IP	E
SS2	C
SP2	A
SS1	8
SP1	6
SS0	4
SP0	2
LINK (селектор старого TSS)	0

T (Бит ловушки отладки) (Intel386, Intel486, Pentium)

Если T = 1, то при переключении задач генерируется особая ситуация отладки.

База карты ввода/вывода (Intel386, Intel486, Pentium)

Служит для указания положения битовой карты ввода/вывода и карты перенаправления прерываний (только Pentium). Если эта карта присутствует, то она расположена в верхней части TSS. Базовый адрес указывает на начало карты ввода/вывода и на конец 32 байтовой карты прерываний, ис-

пользуемой в режиме V86 (только Pentium). Информацию о карте прерываний смотрите в Приложении H.

Карта разрешения в/в (Intel386, Intel486, Pentium)

Определяет доступность портов ввода/вывода. Расположена в верхней части TSS и определяет его длину.

Если CPL ≤ IOPL в защищенном режиме, то проверки доступности ввода/вывода не производится. Если CPL > IOPL, или если процессор функционирует в режиме V86, то производится проверка доступности ввода/вывода при любых операциях с ним.

Каждый бит в карте соответствует своему адресу порта. Например, управляющий бит для порта 29h находится в позиции 1 шестого байта карты ввода/вывода. Процессор проверяет все биты, соответствующие используемому порту, т.е. если пересылается слово, то тестируется два бита, двойное слово - четыре бита, байт - один бит карты в/в. Если какой-либо из тестируемых битов установлен, то генерируется особая ситуация общей защиты. В случае, если используемый порт не отражен в карте ввода/вывода (т.е. имеет больший адрес), то генерируется особая ситуация общей защиты. Если базовый адрес карты ввода/вывода больше или равен пределу сегмента TSS, то любые обращения к портам ввода/вывода вызывают генерацию особой ситуации общей защиты. За последним байтом битовой карты ввода/вывода в TSS следует заключительный байт, содержащий 1 во всех разрядах. Адрес этого байта должен соответствовать границе сегмента, определенной дескриптором TSS.

LINK (Поле обратной связи) (Intel286, Intel386, Intel486, Pentium)

Содержит селектор TSS предыдущей задачи.

SSn, (E)SPn (Intel286, Intel386, Intel486, Pentium)

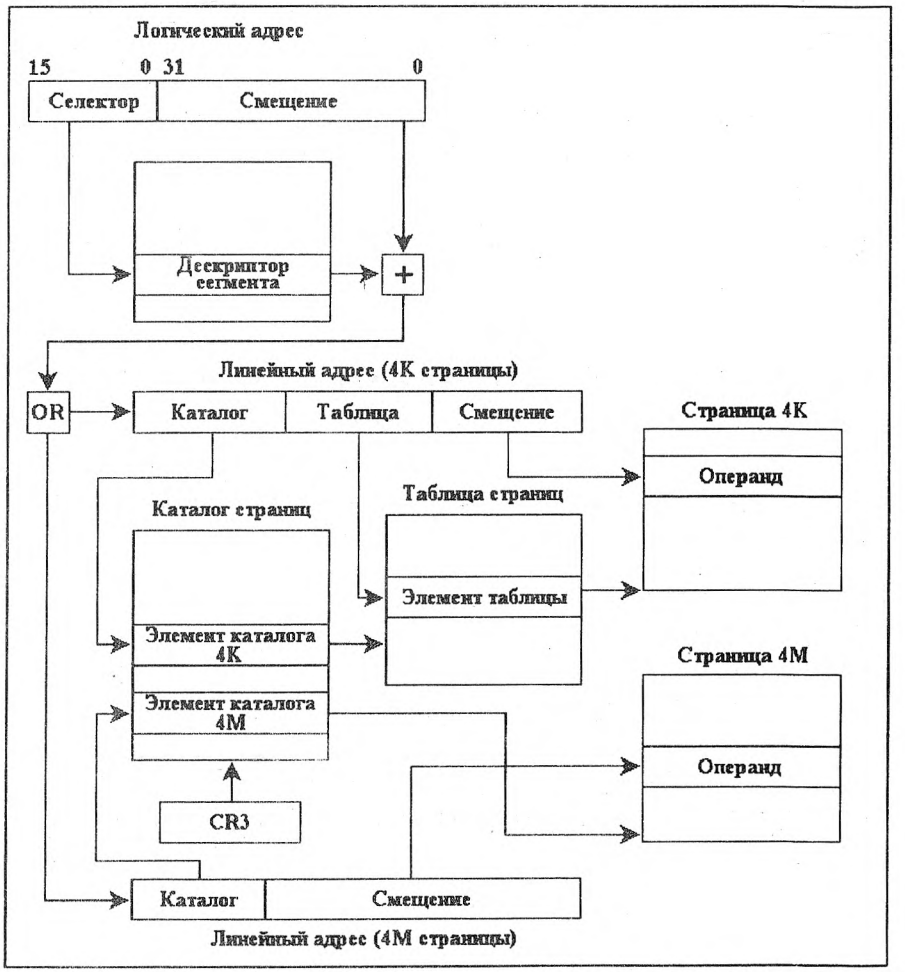
Копии указателя стека SS:ESP на уровне привилегий n.

Примечание: процессор Pentium записывает селектор TSS как 32 битное значение со старшими двумя байтами равными нулю. Intel386, Intel486 записывают только 16 бит.

3.5 Страничный механизм (Intel386, Intel486, Pentium)

3.5.1 Формирование физического адреса при работе со страничным механизмом

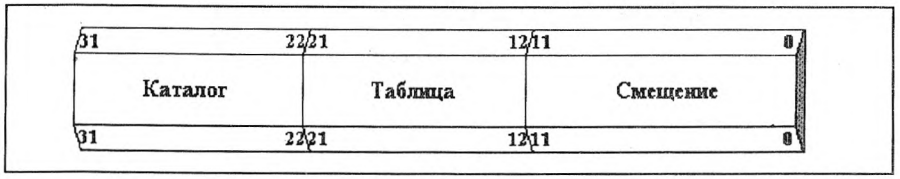
Рис. 3-15



Примечание: размер страниц 4M поддерживает только Pentium.

3.5.2 Линейный адрес (Intel386, Intel486, Pentium)

Рис. 3-16 Формат линейного адреса для 4K страниц



Поле "каталог" используется как индекс при выборе нужного элемента каталогов страниц. Поле "таблица" - для выбора элемента таблицы страниц.

Примечание: информацию об использовании 4M страниц смотрите в приложении H.

3.5.3 Каталоги и таблицы страниц (Intel386, Intel486, Pentium)

Рис. 3-17 Формат элементов каталога страниц

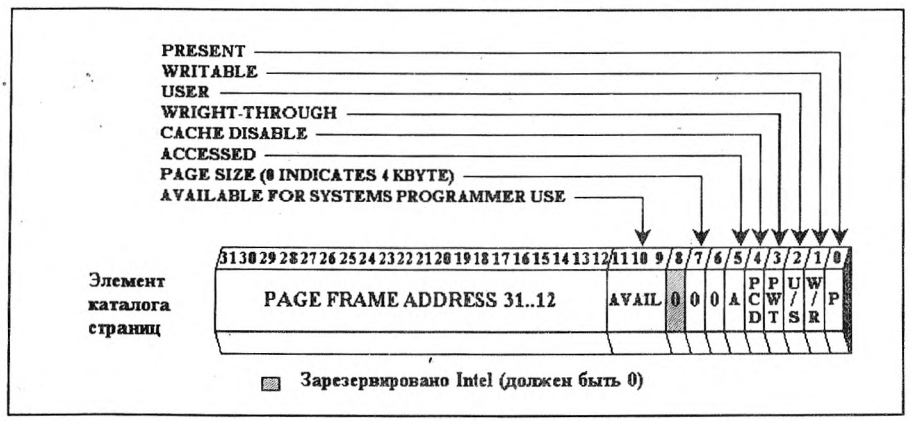
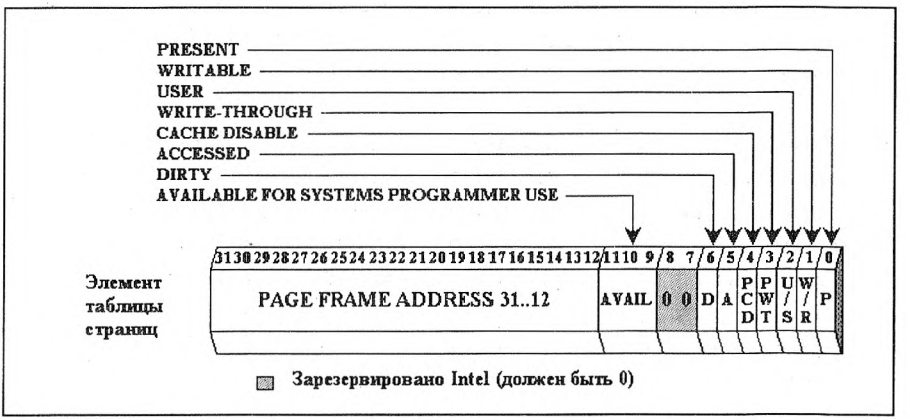


Рис. 3-18 Формат элементов таблицы страниц для 4K режима



PAGE FRAME ADDRESS (Базовый адрес страницы или таблицы страниц, биты 31..12)

Задает начальный физический адрес страницы или таблицы страниц (в элементе каталога страниц для 4K режима). Так как страницы локализованы в 4 Кбайтных границах, 12 младших бит адреса всегда нулевые.

AVAIL (Резерв ОС, биты 11, 10, 9) (Intel386, Intel486, Pentium)

Биты зарезервированы для операционной системы (ОС), которая может использовать их для своих потребностей. Например, в этих битах может размещаться информация о времени последнего обращения к данному каталогу или странице.

D (Бит модификации, бит 6) (Intel386, Intel486, Pentium)

Процессор устанавливает этот бит при записи в выбранную страницу. Устанавливается только бит элемента таблицы страниц. В каталоге страниц в 4K режиме этот бит не используется.

A (Доступ, бит 5) (Intel386, Intel486, Pentium)

Процессор устанавливает бит доступа в обоих таблицах страниц, если производится чтение или запись выбранной страницы.

PCD (Запрещение кэширования страниц, бит 4) (Intel486, Pentium)

Бит используется для управления КЭШ. Программы могут управлять кэшированием как отдельных страниц, так и таблиц страниц посредством этого бита. PCD=1 запрещает загрузку страницы в КЭШ память.

PWT (Сквозная запись страниц, бит 3) (Intel486, Pentium)

Бит используется для управления КЭШ. Программы могут управлять кэшированием как отдельных страниц, так и таблиц страниц посредством этого бита. При PWT=1, для текущей страницы обновление реализуется методом сквозной записи, при PWT = 0 - методом обратной записи.

U/S (Пользователь/Супервизор, бит 2) (Intel386, Intel486, Pentium)

Используется механизмом защиты. Смотрите пункт 3.5.4.

W/R (Доступность для записи/чтения, бит 1) (Intel386, Intel486, Pentium)

Используется механизмом защиты. Смотрите пункт 3.5.4.

P (Присутствие, бит 0) (Intel386, Intel486, Pentium)

Указывает, находится ли данная страница или таблица в физической памяти. Если P=1 - страница в памяти. Если P = 0, то при обращении к такой странице генерируется особая ситуация 14 ("страничная ошибка"). Для элементов с P = 0 все остальные поля не анализируются и могут быть использованы программой для своих нужд.

3.5.4 Механизм защиты страниц (Intel386, Intel486, Pentium)

Защитой страниц управляют биты U/S и W/R элементов каталога и таблицы страниц.

Таб. 3-5 Комбинация защиты в каталоге и в таблице страниц

Элемент каталога страниц		Элемент таблицы страниц		Итоговый эффект	
Привилегия	Доступ	Привилегия	Доступ	Привилегия	Доступ
U	R	U	R	U	R
U	R	U	W	U	R
U	W	U	R	U	R
U	W	U	W	U	W
U	R	S	R	S	W*
U	R	S	W	S	W*
U	W	S	R	S	W*
U	W	S	W	S	W*
S	R	U	R	S	W*
S	R	U	W	S	W*
S	W	U	R	S	W*
S	W	U	W	S	W*
S	R	S	R	S	W*
S	R	S	W	S	W*
S	W	S	R	S	W*
S	W	S	W	S	W*

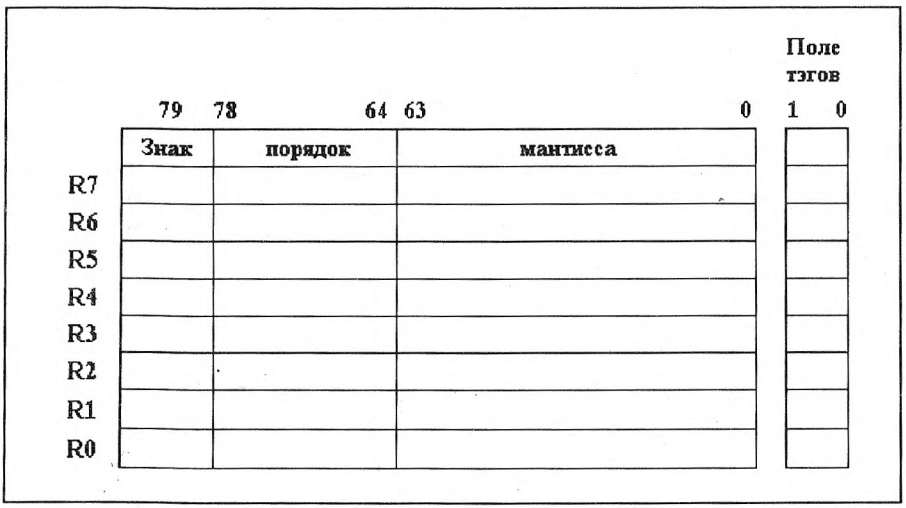
* CR0.WP = 0, то тип доступа - только чтение (Intel486, Pentium)

Примечание: U - пользователь (U/S = 1);
S - супервизор (U/S = 0);
R - только чтение (R/W = 0);
W - чтение/запись (R/W = 1).

4. Математический сопроцессор, прикладная архитектура

4.1 Регистры общего назначения FPU*

Рис. 4-1 Формат регистров общего назначения



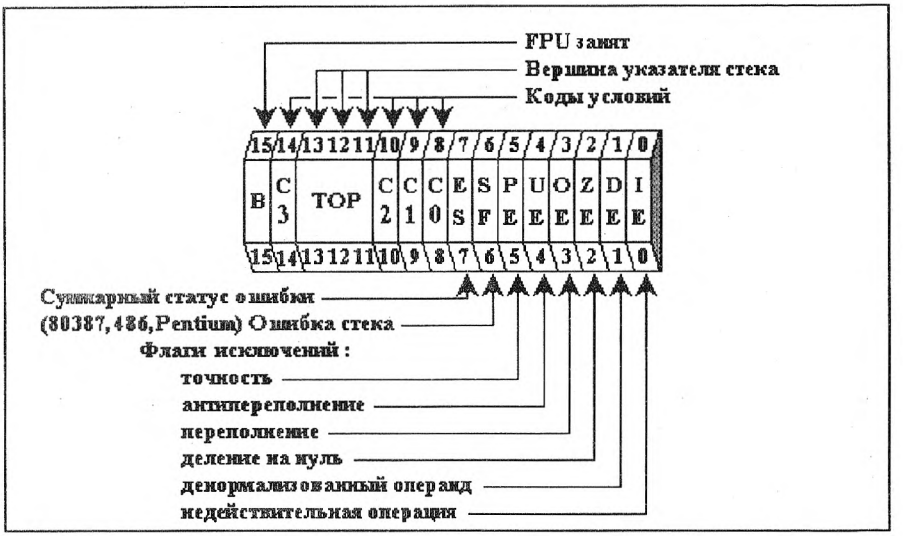
Математический сопроцессор содержит восемь 80 рязрядных регистров общего назначения R0..R7, которые образуют специальный регистровый стек FPU. Регистры стека обозначаются как ST(0)..ST(7) и ассоциируются с физическими регистрами R0..R7 в порядке возрастания номеров. Номер регистра, соответствующего вершине стека ST(0), определяется полем TOP регистра состояния SW. Кроме этого, каждому регистру R0..R7 соответствует двухбитовое поле в регистре тэгов TW.

Примечание: ST(0) также может обозначаться как ST.

* FPU (Floating-Point Unit) - обычно так обозначаются только встроенные сопроцессоры, однако и здесь и в дальнейшем, если это не указано особо, данное обозначение распространяется на все типы описываемых в данной книге математических сопроцессоров.

4.2 Регистр состояния FPU (SW)

Рис. 4-2 Формат регистра состояния



B (бит 15)

Бит занятости сохранен для совместимости с 8087 и Intel287, где он устанавливается в 1, когда сопроцессор выполняет команду, или происходит прерывание в сопроцессоре в результате исключительной ситуации. Если сопроцессор свободен, бит занятости установлен в 0. В процессорах Pentium, Intel486, Intel387 этот бит дублирует состояние бита ES.

TOP (биты 13, 12, 11)

Содержит номер общего регистра, являющегося вершиной стека.

000 - R0
001 - R1
...
111 - R7

C0, C1, C2, C3 (биты 8, 9, 10, 14)

Коды условий. Аналогичны флагам CPU. FPU устанавливает значения этих битов по результатам арифметических операций.

Таб. 4-1 Интерпретация кодов условий

Команда	C0	C3	C2	C1
FCOM, FCOMP, FCOMPP, FTST, FUCOMPP, FICOM, FICOMP	По результатам сравнения		Операнды не сравнимы	Нуль или O/U#
FXAM	Класс операнда			Знак или O/U#
FPREM, FPREM1	Q2	Q1	0 - преобразование завершено 1 - преобразование не завершено	Q0 или O/U#
FIST, FBSTP, FRINDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	Неопределено			Округление или O/U#
FPTAN, FSIN, FCOS, FSINCOS	Неопределено		0 - преобразование завершено 1 - преобразование не завершено	Округление или O/U# (Неопределено, если C2=1)
FCHS, FABS, FXCH, FINCSTP, FDECSTP, FLDconst, FXTRACT, FLD, FILD, FBLD, FSTP (extended-real)	Неопределено			Нуль или O/U#
FLDENV, FRSTOR	Биты загружаются из памяти			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX	Неопределено			
FINIT, FSAVE	Нуль	Нуль	Нуль	Нуль

Примечания:
O/U# - В случае возникновения стековой ошибки, когда устанавливаются биты IE и SF слова состояния SW, бит C1 определяет тип ошибки: переполнение (C1=0) или антипереполнение (C1=1).

Преобразование - Если в процессе выполнения FPREM или FPREM1 получается остаток меньший чем модуль, то функция считается выполненной и флаг C2 очищается. В противном случае, флаг C2 устанавливается, а результат в ST называется неполным остатком. Программа может перезапускать команду (используя неполный остаток в ST как делимое), пока флаг C2 не будет очищен. Для команд FPTAN, FSIN, FCOS и FSINCOS флаг C2 устанавливается, если значение операнда находится вне приемлемого диапазона.

Округление - В случае возникновения исключения "неточный результат", когда устанавливается бит PE слова состояния SW, бит C1 показывает, в какую сторону произведено округление: C1=1 - вверх, C2=0 - вниз.

Команда FSTSW копирует значение слова состояния сопроцессора в регистр AX CPU. Далее командой SAHF можно скопировать C3..C0 в регистр флагов.

Таб. 4-2 Соответствие между битами C3..C0 и битами регистра флагов CPU

Флаги FPU	Флаги EFLAGS
C0	CF
C1	Нет
C2	PF
C3	ZF

ES (бит 7) (IR для 8087)

В 8087 содержит флаг запроса прерывания (IR), который устанавливается в единицу при возникновении какой-либо незамаскированной исключительной ситуации. Значение флага IR повторяется на выводе INT сопроцессора. Флаг IR может быть замаскирован программно установкой соответствующего разряда регистра маски.

В Intel287, Intel387, Intel486, Pentium этот бит называется битом суммарной ошибки (ES) и устанавливается в 1 при возникновении какой-либо незамаскированной исключительной ситуации, иначе - очищается. При установленном бите ES, активизируется сигнал на выводе ERROR# (Intel287, Intel387) или FERR# (Intel486, Pentium).

SF (бит 6) (Intel387, Intel486, Pentium)

Стековая ошибка. В процессорах Pentium, Intel486 и сопроцессоре Intel387, если исключение "недействительная операция" возникает в результате выхода за верхнюю или нижнюю границы регистрового стека, то устанавливается не только бит 0 (IE) слова состояния, но и бит 6 (SF), при этом бит 9 (C1) указывает на выход за верхнюю границу (переполнение) стека (C1 = 1) или на выход за нижнюю границу (C1 = 0).

В сопроцессорах Intel287 и 8087 этот бит не используется и является зарезервированным.

PE, UE, OE, ZE, DE, IE (биты 5, 4, 3, 2, 1, 0)

Флаги исключительных ситуаций. Устанавливаются в случае возникновения одной из шести особых ситуаций сопроцессора (исключений). Каждому исключению соответствует свой флаг:

PE - неточный результат;
UE - антипереполнение;
OE - переполнение;
ZE - деление на нуль;
DE - денормализованный операнд;
IE - недействительная операция.

4.3 Регистр управления FPU (CW)

Рис. 4-3 Формат регистра управления

Управление округлением:

00 - округление к ближайшему или четному

01 - округление вниз (к -∞)

10 - округление вверх (к +∞)

11 - округление к нулю

Управление точностью:

00 - 24 бита мантисса, 7 бит порядок

01 - зарезервировано

10 - 53 (10), двойная точность

11 - 64 (15), расширенная точность

Управление бесконечностью (только Intel287 и 8087):

0 - проективная (∞)

1 - аффинная (±∞) (Всегда в Intel387, Intel486, Pentium)

IC (бит 12) (Intel287, 8087)

Управление бесконечностью. В сопроцессорах Intel287 и 8087 данный бит определяет, что +∞ и -∞ считаются одной беззнаковой бесконечностью (IC = 0) - "проективная арифметика"; или что +∞ и -∞ считаются двумя знаковыми бесконечностями (IC = 1) - "аффинная ариф-

метика". В более поздних сопроцессорах , поддерживающих стандарт IEEE-754 (Intel387, Intel486, Pentium), данный бит, хотя и может программироваться, но не влияет на работу FPU. Независимо от значения IC всегда применяется аффинная арифметика ($-\infty < +\infty$).

RC (биты 11, 10)

Управление округлением. Поле RC определяет выбор одного из четырех методов округления результата арифметических операций FADD, FSUB(R), FMUL, FDIV(R), а также команд FLD и FST(P).

PC (биты 9, 8)

Управление точностью. Указывает точность представления результата арифметических операций FADD, FSUB(R), FMUL и SQRT. На другие команды эти биты влияния не оказывают. Уменьшение точности может быть полезно для совместимости с ранними моделями арифметических сопроцессоров.

IEM (бит 7) (только 8087)

Маскирует флаг запроса прерывания IR, т.е. независимо от значений шести масок исключений сигнал IEM = 1 запрещает генерацию прерываний при возникновении ошибок сопроцессора.

PM, UM, OM, ZM, DM, IM (биты 5, 4, 3, 2, 1, 0)

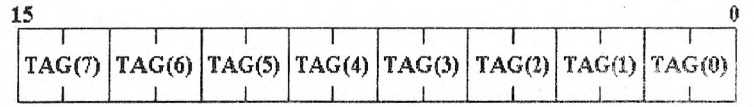
Маски исключений. Биты содержат индивидуальные маски для каждой из шести исключительных ситуаций сопроцессора. Установка бита маски запрещает прерывание при возникновении соответствующей ошибки FPU.

4.4 Регистр тэгов (TW)

Слово тэгов описывает содержимое всех регистров регистрового стека FPU.

Значения тэгов соответствуют физическим регистрам R0 - R7. Необходимо использовать поле TOP регистра состояния для определения соответствия тэгов к относительным регистрам стека ST(0) - ST(7).

Рис. 4-4 Формат регистра тэгов



Значение тэга (TAG):

00 - действительный;
01 - нуль;
10 - специальный: недействительный (NAN, неподдерживаемый), бесконечность или денормализованный;
11 - пустой.

Программы обработчики исключений могут использовать тэговую информацию для определения содержимого регистров стека без применения комплексного анализа непосредственных значений.

При загрузке слова тэгов командами FLDENV или FRSTOR процессоры Pentium, Intel486 и Intel387 разделяют приходящие значения тэгов на два типа: занятый (значения 00, 01 и 10) и незанятый (11). В последующих операциях с незанятыми регистрами всегда проверяется значение в регистре, но не значение тэга. Команды FSTENV и FSAVE проверяют незанятые регистры и устанавливают корректные значения тэгов перед их занесением в память.

В сопроцессорах Intel287 и 8087 при загрузке слова тэгов командами FLDENV или FRSTOR корректность заносимых значений тэгов также не проверяется, однако, в последующих операциях сопроцессор опирается не на конкретные значения в регистрах, а на тэговую информацию, корректируя ее в случае изменения содержимого регистров. Поэтому программное обеспечение может функционировать неправильно при занесении в слово тэгов (командами FLDENV или FRSTOR) информации, не соответствующей действительному содержимому регистров.

Еще одно отличие сопроцессоров Intel287 и 8087 состоит в том, что они интерпретируют значения "псевдо-нуль" и "ненормализованное" как действительные (00). Более поздние модели сопроцессоров, поддерживающие стандарт IEEE-754, интерпретируют эти значения как специальные (10).

4.5 Специальные регистры FPU

Регистры указателя команд (FIP) и указателя данных (FDP) предназначены для использования обработчиком исключений. Всегда при приходе команды ESC (кроме FINIT, FCLEX, FLDCW, FSTCW, FSTSW, FSTSW AX, FSTENV, FLDENV, FSAVE и FWAIT) процессор сохраняет адрес команды, код операции и адрес операнда (если присутствует) в регистрах указателях. Содержимое этих регистров не изменяется при выполнении последующих управляющих команд (перечислены выше).

Доступ к этим регистрам осуществляется с помощью команд FSTENV, FLDENV, FSAVE и FRSTOR.

При записи в память, указатели команды и данных могут представляться в четырех различных форматах в зависимости от режима работы процессора и атрибута размера адреса команды. На рис. 4-5...4-8 представлены эти форматы при использовании команд FSTENV, FLDENV.

Рис. 4-5 Формат данных FSTENV, FLDENV в 32 битном защищенном режиме (Pentium, Intel486, Intel387)

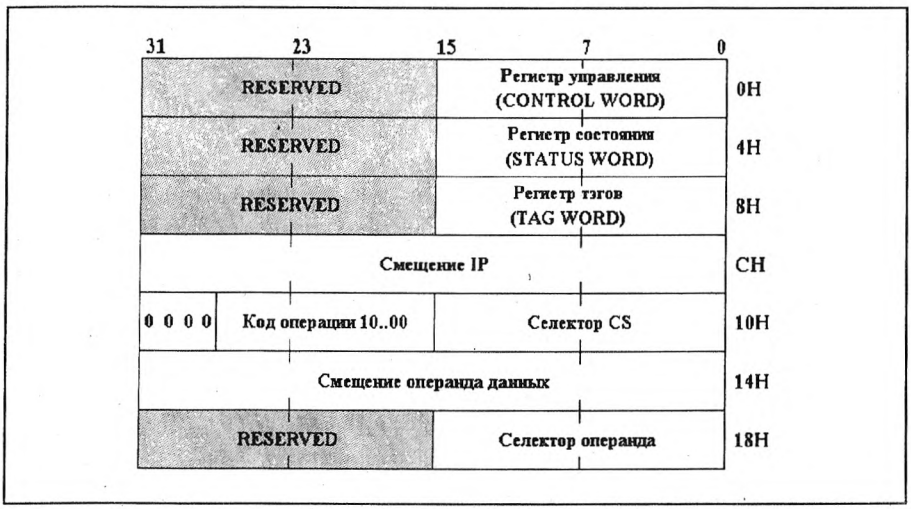


Рис. 4-6 Формат данных FSTENV, FLDENV в 32 битном режиме реальной адресации (Pentium, Intel486, Intel387)

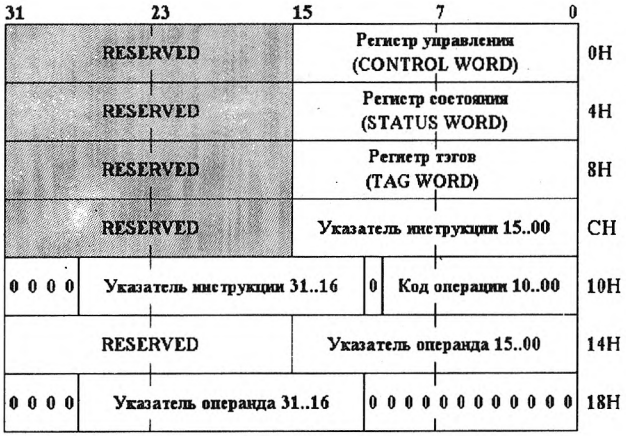


Рис. 4-7 Формат данных FSTENV, FLDENV в 16 битном защищенном режиме (Pentium, Intel486, Intel387, Intel287)

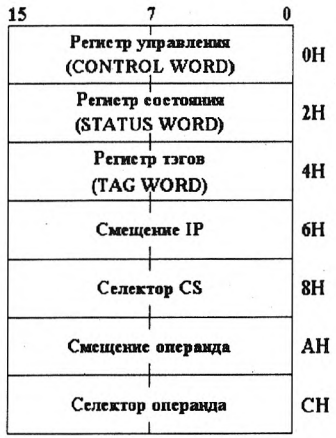


Рис. 4-8 Формат данных FSTENV, FLDENV в 16 битном режиме реальной адресации (Pentium, Intel486, Intel387, Intel287, 8087) и режиме V86 (Pentium, Intel486, Intel387)

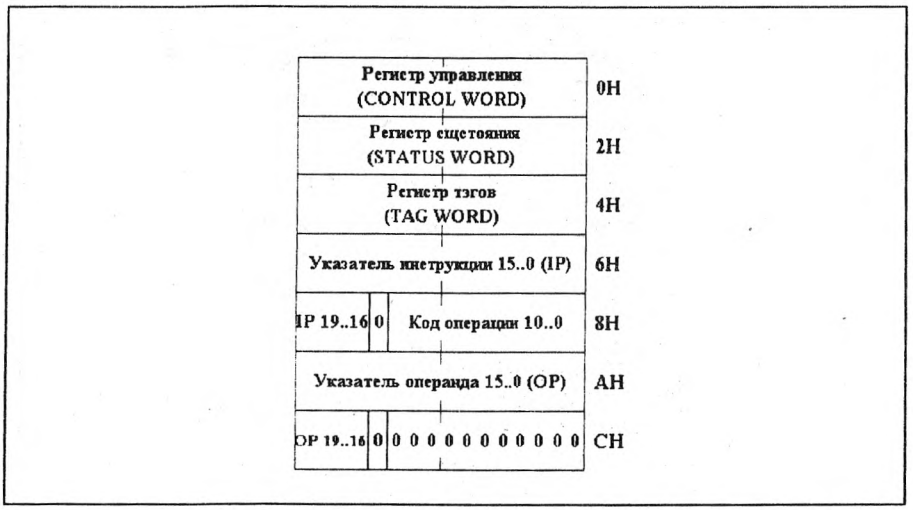
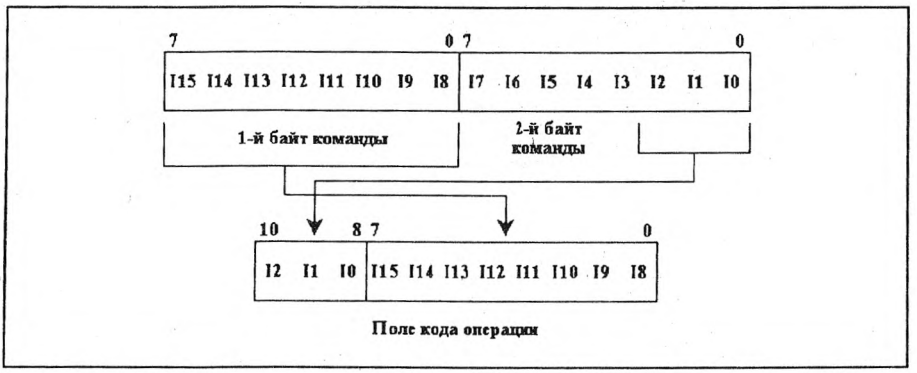


Рис. 4-9 Формат поля кода операции



Во всех сопроцессорах кроме 8087 адрес команды учитывает наличие префиксов. В 8087 префиксы не учитываются.

4.6 Состояние FPU после сброса и инициализации

Таб. 4-3 Состояние FPU после аппаратного сброса

Регистр	Pentium[1]	Intel486[2]	Intel387[3]	Intel287	8087
Управления (CW)	0040h	037Fh	037Fh	037Fh	037Fh
Состояния (SW)	0000h	0000h	0000h	0000h*	0000h*
Тэгов (TW)	FFFFh	FFFFh	FFFFh	FFFFh	FFFFh
Указатель команд (FIP)	00000000h	00000000h	00000000h	0000h	0000h
Указатель данных (FDP)	00000000h	00000000h	00000000h	0000h	0000h
Селектор CS (FIP)	0000h	0000h	0000h	0000h	0000h
Селектор операнда (FDP)	0000h	0000h	0000h	0000h	0000h
Стек (R0 - R7)	все 0	все 0	все 0	все 0	все 0

* После аппаратного сброса биты C0, C1, C2, C3 слова состояния 8087 и Intel287 не изменяются.

1 В Pentium состояние FPU по сигналу INIT не изменяется.

2 В Intel486 состояние FPU не изменяется при аппаратном сбросе без самотестирования.

3 В Intel387 состояние регистрового стека не изменяется, если процессор Intel386 сбрасывается без самотестирования.

После аппаратного сброса или команды FINIT сопроцессор Intel387 сигнализирует о состоянии ошибки (активный сигнал на выводе ERROR#). Процессоры Pentium, Intel486 и сопроцессоры Intel287, 8087 наоборот (неактивный сигнал на выводе ERROR# для Intel287). По этому признаку процессор Intel386 идентифицирует тип математического сопроцессора (Intel287 или Intel387).

5. Математический сопроцессор, системная архитектура.

5.1 Исключительные ситуации сопроцессора

Недействительная операция (IE)

Существует два класса операций, которые могут вызывать исключение "недействительная операция": стековые и арифметические.

1. Стековые операции.

Исключение возникает, если производится попытка занесения данных в занятый стековый регистр или попытка чтения из незанятого регистра. При этом в сопроцессорах, которые отвечают стандарту IEEE-754 (Intel387, Intel486, Pentium), устанавливается также флаг SF слова состояния, а флаг C1 идентифицирует тип стековой ошибки.

C1(O/U#) = 1 - запись в занятый регистр;

C1(O/U#) = 0 - чтение пустого регистра.

Когда исключение "недействительная операция" замаскировано, FPU возвращает неопределенность QNAN. Это значение помещается в регистр назначения, разрушая его содержимое. Когда исключение не замаскировано, то вызывается обработчик исключений, поле TOP и исходные операнды не изменяются.

2. Арифметические операции.

Таб. 5-1 Условия возникновения исключения "недействительная операция" и реакция сопроцессора, если исключение замаскировано.

Условия		Реакция (при IM=1)[1]
Для сопроцессоров отвечающих стандарту IEEE-754 (Intel386, Intel486, Pentium)[3]	Для сопроцессоров Intel287, 8087	
Один или оба операнда SNAN.	Один или оба операнда NAN.	Возвращает неопределенность QNAN (с большим абсолютным значением).
Сравнение или тестирование, если один или оба операнда NAN.	Сравнение или тестирование, если один или оба операнда NAN, или один операнд - проективная бесконечность.	Устанавливает код "несравнимы" (C3C0 = 11).
Сложение бесконечностей с различными знаками или вычитание бесконечностей с одинаковыми знаками.	Сложение бесконечностей с различными знаками или вычитание бесконечностей с одинаковыми знаками (аффинная арифметика). Сложение или вычитание, когда оба операнда - проективная бесконечность.	Возвращает неопределенность QNAN.
Умножение бесконечности на нуль или нуля на бесконечность.	Умножение бесконечности на нуль или нуля на бесконечность.	Возвращает неопределенность QNAN.
Деление бесконечности на бесконечность или нуля на нуль.	Деление бесконечности на бесконечность, нуля на нуль, нуля на псевдонуль, делитель не нормализован или денормализован.	Возвращает неопределенность QNAN.
Команды FPREM, FPREM1 - делитель нулевой, или делимое бесконечность.	Команды FPREM, FPREM1 - делитель нулевой, не нормализован, денормализован, или делимое бесконечность.	Возвращает неопределенность QNAN, устанавливает код "команда завершения" (C2=0).
Команда FSQRT - отрицательный операнд (кроме -0), команда FYL2X - отрицательный операнд (кроме -0), команда FYL2XP1 - операнд X < -1.	Команда FSQRT - операнд не нормализован, денормализован, отрицательное число (кроме -0), проективная бесконечность.	Возвращает неопределенность QNAN.

Команды FIST(P) - регистр источник пуст, NAN, бесконечность или непредставим в указанном формате.	Команды FIST(P) - регистр источник пуст, не нормализован, денормализован, NAN, бесконечность или непредставим в указанном формате.	Записывает целочисленную неопределенность.
Команды FBST(P) - регистр источник пуст, NAN, бесконечность или превышает 18 десятичных цифр.	Команды FBST(P) - регистр источник пуст, денормализован, не нормализован, NAN, бесконечность или превышает 18 десятичных цифр.	Записывает упакованную десятичную неопределенность.
	Команды FST(P) - регистр источник содержит не нормализованные значения, а регистр приемник определен форматом short-real или long-real.	Записывает код неопределенности в формате short-real или long-real.
Тригонометрические команды FCOS, FSIN, FPTAN, FSINCOS - операнд бесконечность.		Возвращает неопределенность QNAN, устанавливает код "команда завершения" (C2=0).
Команда FXCH - один или оба регистра пусты.	Команда FXCH - один или оба регистра пусты.	Заполняет пустой/пустые регистры кодом неопределенность QNAN и совершает обмен.
Операции с неподдерживаемыми форматами[2]		Возвращает неопределенность QNAN.

Примечания:

1. Процессоры Intel287, 8087 не поддерживают формат неопределенности QNAN, вместо этого они возвращают неопределенность в формате временного вещественного.
2. Форматы псевдонуль, псевдоNAN, псевдонеопределенность, и не нормализованный поддерживаются только сопроцессорами Intel287 и 8087. Процессоры Pentium, Intel486, Intel387 генерируют исключение "недействительная операция" при встрече данных форматов.
3. Процессоры Pentium, Intel486, Intel387 автоматически нормализуют встреченные денормализованные операнды, где это возможно.

Если исключение не замаскировано, то вызывается обработчик исключений, и операнды остаются неизмененными.

Денормализованный операнд (DE)

Во всех сопроцессорах, если DE не замаскировано, то при встрече денормализованного операнда происходит вызов обработчика исключений (операнды при этом не изменяются).

Если исключение замаскировано, то процессоры Pentium, Intel486, Intel387 производят преобразование встреченных денормализованных операндов к нормализованному виду. Сопроцессоры Intel287, 8087 оперируют с денормализованными операндами по следующим правилам:

Команда FLD - операнд источник денормализован;	загружает как обычный (никаких специальных действий).
Арифметические команды - один или оба операнда денормализованы;	преобразует операнд(ы) в эквивалентное ненормализованное значение и выполняет команду.

Примечание: сопроцессоры Intel287, 8087 не поддерживают исключение "денормализованный операнд" в команде FXTRACT.

Деление на нуль (ZE)

Исключение возникает при делении ненулевого операнда на нуль. Это возможно в командах F(I)DIV(R)(P), а также в использующих деление командах FYL2X и FXTRACT. Если исключение не замаскировано, то вызывается обработчик исключения, иначе происходит следующее:

- операции FDIV возвращают бесконечность со знаком равным исключающему ИЛИ (XOR) знаков операндов;
- FYL2X возвращает бесконечность со знаком противоположным знаку ненулевого операнда;
- для FXTRACT, ST(1) устанавливается равным $-\infty$, ST устанавливается в 0 со знаком исходного операнда (Pentium, Intel486, Intel387).

Переполнение (OE)

Исключение генерируется, если результат операции превышает максимально допустимое значение для положительных чисел или минимально допустимое для отрицательных. Это может происходить в командах FST(P), F(I)ADD(P), F(I)SUB(R)(P), F(I)MUL(P), FDIV(R)(P), FSCALE, FYL2X, FYL2XP1.

Если исключение "переполнение" замаскировано, то возвращается одно из следующих значений:

Режим округления	Знак истинного результата	Возвращаемое значение
К ближайшему	+	+∞
	-	-∞
К -∞	+	Наибольшее положительное
	-	-∞
К +∞	+	+∞
	-	Наименьшее отрицательное
К нулю	+	Наибольшее положительное
	-	Наименьшее отрицательное

Если исключение "переполнение" не замаскировано, реакция сопроцессора зависит от того, предполагает команда поместить результат в стек или в память.

Если назначением является стек, истинный результат делится на 2^{24576} и округляется (до заданной полем РС слова управления точности в командах, где задание точности возможно, и до расширенной точности в других командах). Бит С1 слова состояния устанавливается, если произошло округление вверх.

Если назначением является память, то в ней ничего не сохраняется. Обработчик исключения может изменить исходные операнды и перезапустить команду, а также принудительно записать нужный результат в память и продолжить выполнение.

Примечание:

В процессорах Pentium, Intel486, Intel387, если исключение "переполнение" замаскировано, и установлен режим округления к нулю, результатом операции, вызвавшей переполнение, является наибольшее положительное или наименьшее отрицательное число. Сопроцессоры Intel287 и 8087 не генерируют исключение "переполнение", если маскируемый отклик не является бесконечностью.

Если исключение "переполнение" не замаскировано, то в процессорах Pentium, Intel486, Intel387, когда результат запоминается в стеке, мантисса результата округляется в соответствии с полем РС управляющего слова или

в соответствии с кодом операции; при этом флаг исключения точности устанавливается по результатам операции. В сопроцессорах Intel287 и 8087 округление не производится, и флаг исключения точности не изменяется.

Антипереполнение (UE)

Антипереполнение может быть вызвано командами FST(P), FADD(P), FSUB(RP), FMUL(P), F(I)DIV(RP), FSCALE, FPREM(1), FPTAN, FSIN, FSINCOS, FPATAN, F2XM1, FYL2XP1.

Ситуации, которые могут привести к исключению "антипереполнение":

1. Очень маленький результат, который из-за своей малости может стать причиной некоторых других исключений впоследствии (например, исключение "переполнение" после деления);
2. Неточный результат, то есть ситуации, когда точный результат может быть получен при вычислении с неограниченным диапазоном экспоненты и точности. (Потеря точности при денормализации малых чисел.)

Исключение "антипереполнение" возникает:

1. Когда результат одновременно маленький и неточный, если маска установлена (OM=1).
2. Когда результат маленький, если маска не установлена (OM=0). Точность/неточность результата значения не имеет.

Реакция на исключение "антипереполнение":

1. Если маска установлена (OM=1), то возвращается денормализованный или нулевой результат. Исключение точности также может выставляться.
2. Если маска не установлена (OM=0), то результат операции, вызывающий антипереполнение зависит от того, используется в качестве назначения стек или память.

Если назначением является стек, истинный результат умножается на 2^{24576} и округляется до заданной полем PC слова управления точности в командах, где задание точности возможно и до расширенной точности в других командах. Бит C1 слова состояния устанавливается, если произошло округление вверх.

Если назначением является память, то в ней ничего не сохраняется. Обработчик исключения может изменить исходные операнды и перезапустить команду, а также принудительно записать нужный результат в память и продолжить выполнение.

Примечание:

В сопроцессорах Intel287 и 8087 когда округление производится к нулю, и маска установлена (UM=1), флаг антипереполнения выставляется по маленькому результату независимо от его точности/неточности. Когда маска неустановлена, и назначение - стек, то мантисса не округляется (то есть результат отличается от результата в Pentium, Intel486, Intel387 значением наименее значимого бита мантиссы).

Неточный результат (PE)

Это исключение возникает, если результат операции может быть точно представлен в формате назначения (например, 1/3 не может быть точно представлена в двоичном виде).

Бит C1 показывает в какую сторону произведено округление: C1=1 - вверх, C2=0 - вниз.

Данное исключение может вызываться если произошло исключение "антипереполнение" и при этом: OM=1 и имеет место потеря точности.

Приоритет исключений

При возникновении в одной операции сразу нескольких исключительных ситуаций, как правило (кроме оговоренных в описании исключений случаев), устанавливается только флаг исключения с большим приоритетом.

Приоритеты исключений распределены следующим образом:

1. "Недействительная операция":
   - обращение к пустому стеку;
   - переполнение стека;
   - неподдерживаемый формат операнда;
   - операнд SNAN.
2. Операнд QNAN. Данный случай не является исключением, но он может маскировать исключения с меньшим приоритетом. Например, деление QNAN на нуль не вызывает исключения "деление на нуль", а возвращает результатом QNAN.

3. Любые другие источники исключения "недействительная операция", кроме названных выше и деления на нуль.
4. "Денормализованный операнд", если замаскировано, то дает возможность к возникновению исключений с меньшим приоритетом.
5. "Переполнение" и "антипереполнение".
6. "Неточный результат".

5.2 Форматы данных сопроцессора

Таб. 5-2 Форматы данных FPU

Форматы данных	Число байтов	Примерный диапазон значений
Целое слово	2	$\pm 10^{4}$
Короткое целое	4	$\pm 10^{9}$
Длинное целое	8	$\pm 10^{18}$
Двоично-десятичное	10	$\pm 10^{18}$
Короткое вещественное	4	$\pm 10^{\pm 38}$
Длинное вещественное	8	$\pm 10^{\pm 308}$
Временное вещественное	10	$\pm 10^{\pm 4932}$

Таб. 5-3 Представление двоичных целых

Класс	Знак	Величина
Положительные		
(Больше)	0	11 .. 11
	...	...
(Меньше)	0	00 .. 01
Нуль	0	00 .. 00
Отрицательные		
(Меньше)	1	11 .. 11
	...	...
(Больше/неопределенность*)	1	00 .. 01
	Слово:	15 бит
	Короткое:	31 бит
	Длинное:	63 бита

* Если данное значение используется в качестве операнда-источника (команды загрузки целого или арифметические операции с целыми), FPU воспринимает его как наибольшее отрицательное значение, представленное в формате -2^{15}, -2^{31}, -2^{63}. FPU может возвращать данное значение в назначение в следующих двух случаях:

1. Если результат - наибольшее отрицательное число.
2. В качестве реакции на замаскированное исключение "недействительная операция", когда возвращается специальное значение "целочисленная неопределенность".

Таб. 5-4 Представление двоично-десятичных

Класс	Знак		Величина					
			digit	digit	digit	digit	...	digit
Положительные								
(Больше)	0	0000000	1001	1001	1001	1001	...	1001
	...	...	...	...	...	...	...	...
(Меньше)	0	0000000	0000	0000	0000	0000	...	0001
Нуль	0	0000000	0000	0000	0000	0000	...	0000
Отрицательные								
(Меньше)	1	0000000	0000	0000	0000	0000	...	0000
	...	...	...	...	...	...	...	...
(Больше)	1	0000000	1001	1001	1001	1001	...	1001
Неопределенность*	1	1111111	1111	1111	U**	U	...	U
	1 байт		9 байт					

* Упакованная десятичная неопределенность сохраняется FBSTP как реакция на замаскированное исключение "недействительная операция". Попытка загрузить это значение командой FBLD приведет к неопределенному результату.

** U означает, что значение битов неопределено и может быть любым.

Таб. 5-5 Представление коротких (short-real) и длинных (long-real) вещественных

Класс	Знак	Порядок	Мантисса ff-ff*
Положительные "не числа" (NaN)			
Quiet (QNaN)	0 ... 0	11 .. 11 ... 11 .. 11	11 .. 11 ... 10 .. 00
Signaling (SNaN)	0 ... 0	11 .. 11 ... 11 .. 11	01 .. 11 ... 00 .. 01
Бесконечность (+INF)	0	11 .. 11	00 .. 00
Положительные вещественные			
Нормализованные	0 ... 0	11 .. 10 ... 00 .. 01	11 .. 11 ... 00 .. 00
Денормализованные	0 ... 0	00 .. 00 ... 00 .. 00	11 .. 11 ... 00 .. 01
Нуль	0	00 .. 00	00 .. 00
Отрицательные вещественные			
Нуль	1	00 .. 00	00 .. 00
Денормализованные	1 ... 1	00 .. 00 ... 00 .. 00	00 .. 01 ... 11 .. 11
Нормализованные	1 ... 1	00 .. 01 ... 11 .. 10	00 .. 00 .... 11 .. 11
Бесконечность (-INF)	1	11 .. 11	00 .. 00
Отрицательные "не числа" (NaN)			
Signaling (SNaN)	1 ... 1	11 .. 11 ... 11 .. 11	00 .. 01 ... 01 .. 11
(Неопределенность QNaN) Quiet (QNaN)	1 ... 1	1 .. 11 ... 11 .. 11	10 .. 00 ... 11 .. 11
	Короткое:	8 бит	23 бита
	Длинное:	11 бит	52 бита

Таб. 5-6 Представление временных вещественных (temp-real)

Класс	Знак	Порядок	Мантисса ff-ff*
Положительные "не числа" (NaN)			
Quiet (QNaN)	0 ... 0	11 .. 11 ... 11 .. 11	111 .. 11 ... 110 .. 00
Signaling (SNaN)	0 ... 0	11 .. 11 ... 11 .. 11	101 .. 11 ... 100 .. 01
Бесконечность (+INF)	0	11 .. 11	100 .. 00
Положительные вещественные			
Нормализованные	0 ... 0	11 .. 10 ... 00 .. 01	111 .. 11 ... 100 .. 00
Ненормализованные	0 ... 0	00 .. 00 ... 00 .. 00	111 .. 11 ... 100 .. 01
Денормализованные	0 ... 0	00 .. 00 ... 00 .. 00	011 .. 11 ... 000 .. 01
Нуль	0	00 .. 00	000 .. 00
Отрицательные вещественные			
Нуль	1	00 .. 00	000 .. 00
Денормализованные	1 ... 1	00 .. 00 ... 00 .. 00	000 .. 01 ... 011 .. 11
Ненормализованные	1 ... 1	00 .. 00 ... 00 .. 00	100 .. 00 ... 111 .. 11
Нормализованные	1 ... 1	00 .. 01 ... 11 .. 10	100 .. 00 ... 111 .. 11
Бесконечность (-INF)	1	11 .. 11	100 .. 00
Отрицательные "не числа" (NaN)			
Signaling (SNaN)	1 ... 1	11 .. 11 ... 11 .. 11	100 .. 01 ... 101 .. 11
(Неопределенность QNaN) Quiet (QNaN)	1 ... 1	1 .. 11 ... 11 .. 11	110 .. 00 ... 111 .. 11
		15 бит	64 бита

Таб. 5-7 Неподдерживаемые форматы Pentium, Intel486, Intel387 (эти форматы поддерживаются в Intel287 и 8087)

Класс	Знак	Порядок	Мантисса f_{Δ}ff-ff
Положительные псевдо "не числа" (Pseudo-Nan)			
Quiet	0 ... 0	11 .. 11 ... 11 .. 11	0_{Δ}11 .. 11 ... 010 .. 00
Signaling	0 ... 0	11 .. 11 ... 11 .. 11	001 .. 11 ... 000 .. 01
Псевдо-бесконечность	0	11 .. 11	100 .. 00
Положительные вещественные			
Ненормализованные	0 ... 0	11 .. 10 ... 00 .. 01	011 .. 11 ... 000 .. 00
Отрицательные вещественные			
Ненормализованные	1 ... 1	00 .. 01 ... 11 .. 10	000 .. 00 ... 011 .. 01
Псевдо-бесконечность	1	11 .. 11	100 .. 00
Отрицательные псевдо "не числа" (Pseudo-NaN)			
Signaling	1 ... 1	11 .. 11 ... 11 .. 11	000 .. 01 ... 001 .. 11
Quiet	1 ... 1	11 .. 11 ... 11 .. 11	010 .. 00 ... 011 .. 11
		15 бит	64 бита

6. Система команд

6.1 Атрибуты (признаки) размера операнда и размера адреса

Во время выполнения команды процессор может обращаться к памяти, используя 16 или 32 разрядные адреса. Поэтому, каждая команда, использующая обращение к памяти, ассоциируется с атрибутом 16 или 32 разрядного адреса. В 16 разрядном режиме подразумевается использование 16 бит смещения в команде и генерация 16 битного адресного смещения (относительный адрес в сегменте). В свою очередь 32 разрядная адресация подразумевает использование 32 бит смещения в команде и генерацию 32 битного адресного смещения. Подобным образом, команда, которая обращается к словам (16 бит) или двойным словам (32 бита), имеет атрибут, соответственно, 16 или 32 битного размера операнда.

Атрибуты полностью определяются комбинацией значения по умолчанию, префиксом команды и (для команд работающих в защищенном режиме) битами спецификации размера в дескрипторах сегментов.

6.1.1 Атрибуты размера операнда и размера адреса по умолчанию

Для программ, работающих в защищенном режиме, D бит в дескрипторах выполняемых сегментов определяет значение атрибута размера операнда и размера адреса по умолчанию. Это значение атрибута применяется ко всем командам, выполняемым в сегменте. Если D бит сброшен, то значения размера операнда и размера адреса по умолчанию устанавливаются равными 16 бит, а если D бит установлен - 32 бита.

Программы, выполняемые в реальном или виртуальном-8086 режимах, всегда имеют размеры операнда и адреса по умолчанию равными 16 бит.

6.1.2 Префиксы размера операнда и размера адреса

Внутренняя кодировка команды может включать в себя два однобайтных префикса: префикс размера операнда (67h), префикс размера адреса (66h). (Положение префиксов при кодировке команды описано в пункте "Формат команды".) Эти префиксы замещают значения атрибутов размера операнда и размера адреса принятые по умолчанию для команды следующей за префиксом. В табл. 6-1 показаны все возможные комбинации значений атрибутов по умолчанию и их замещения префиксами.

Таб. 6-1 Префиксы

Сегмент по умолчанию (D =)	0	0	0	0	1	1	1	1
Префикс размера операнда 66h	N	N	Y	Y	N	N	Y	Y
Префикс размера адреса 67h	N	Y	N	Y	N	Y	N	Y
Эффективный размер операнда	16	16	32	32	32	32	16	16
Эффективный размер адреса	16	32	16	32	32	16	32	16

6.1.3 Атрибут размера адреса для стека

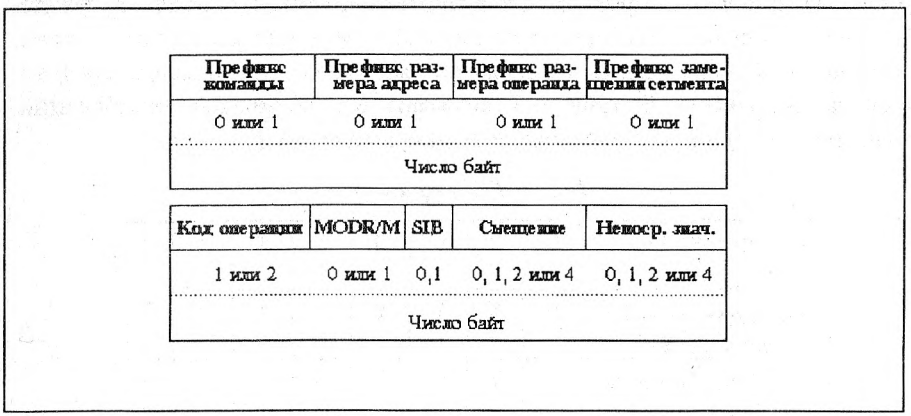
Команды, которые используют стек неявно (например: pop eax), также имеют атрибут размера адреса стека 16 или 32 бит. Команды с атрибутом 16 битного размера адреса стека используют 16 битный регистр указателя стека SP; команды с атрибутом 32 битного размера адреса стека используют 32 битный регистр указателя стека ESP.

Атрибут размера адреса стека управляется B битом дескриптора сегмента данных регистра SS. Нулевое значение B бита, устанавливает атрибут 16 битного размера адреса стека, единичное значение устанавливает атрибут 32 битного размера адреса стека.

6.2. Формат команды

Кодировки всех команд являются подмножеством общего формата команды, показанного на рис. 6-1.

Рис. 6-1. Формат команды



Команда содержит следующие поля:

- префиксы команды (необязательно присутствуют и могут следовать в произвольном порядке);
- одно или двухбайтный код операции;
- спецификатор адреса, состоящий из байта режима адресации ModR/M и байта масштабируемого индекса SIB (Scale Index Base) (необязательно присутствуют);
- смещение, если оно необходимо;
- поля непосредственного значения, если такое значение присутствует.

В коде операции могут быть определены более мелкие поля:

- направление операции;
- размер смещения;
- коды регистров;
- знак расширения.

Эти поля варьируются в зависимости от класса операции.

Большинство команд, которые могут ссылаться на операнд в памяти, содержат байт режима адресации, следующий за байтом (байтами) кода операции. Этот байт, называемый ModR/M байтом, определяет используемую форму (режим) адресации. Кодирование определенным образом байта ModR/M указывает на наличие второго байта спецификатора адреса - SIB байта, который следует сразу же за байтом ModR/M и необходим для полного описания формы адресации.

Формы адресации могут включать в себя смещение, следующее непосредственно за ModR/M байтом или SIB байтом. Присутствующее смещение может быть 8, 16 или 32 битным.

Если в команде присутствует непосредственный операнд, то он всегда следует за байтами смещения и является последним полем в команде.

Для каждой группы префиксов зарезервировано не более одного байта, т.е. префикс данной группы может присутствовать или нет. Префиксы сгруппированы следующим образом:

1. Префиксы команды REP, REPE/REPZ, REPNE/REPNZ, LOCK;
2. Префиксы замещения сегмента CS, SS, DS, ES, FS, GS;
3. Префиксы замещения размера операнда;
4. Префиксы замещения размера адреса.

В команде может присутствовать только по одному префиксу из каждой группы. Результат использования избыточного количества префиксов (более одного префикса из группы) неопределен и варьируется от процессора к процессору. Префиксы могут следовать в любом порядке.

Префиксы команды:

F3h - REP (используется только со строковыми командами);
F3h - REPE/REPZ (используется только со строковыми командами);
F2h - REPNE/REPNZ (используется только со строковыми командами);
F0h - LOCK.

Префиксы замещения сегмента:

2Eh - Префикс замещения сегмента по умолчанию сегментом CS;
36h - Префикс замещения сегмента по умолчанию сегментом SS;
3Eh - Префикс замещения сегмента по умолчанию сегментом DS;
26h - Префикс замещения сегмента по умолчанию сегментом ES;
64h - Префикс замещения сегмента по умолчанию сегментом FS;
65h - Префикс замещения сегмента по умолчанию сегментом GS;
66h - Префикс замещения размера операнда;
67h - Префикс замещения размера адреса.

6.2.1 Байты ModR/M и SIB

Во многих командах процессора байты ModR/M и SIB следуют за байтом (байтами) кода операции. Эти байты содержат следующую информацию:

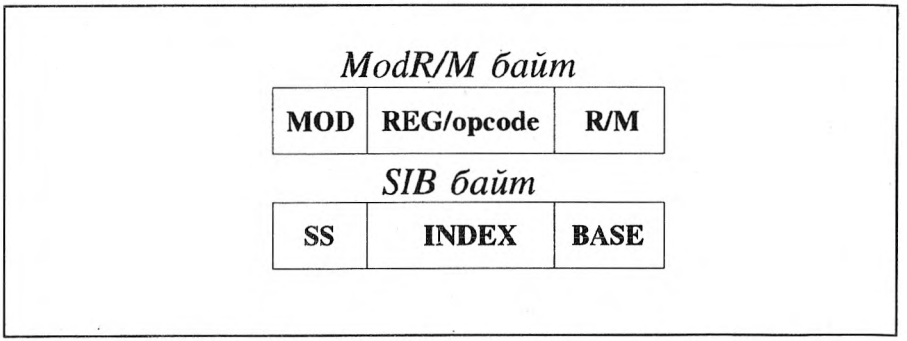
- Тип индексации или номер регистра, используемого в команде;
- Используемый регистр или дополнительная информация о коде операции;
- Информация о базе, индексе и масштабе.
- Байт ModR/M состоит из трех информационных полей:
- Поле mod , которое содержит два наиболее значимых бита (7 и 6 биты), в комбинации с полем r/m определяет 32 возможных типа адресации (8 регистровых и 24 индексных);
- Поле reg содержит следующие за полем mod три бита (5, 4 и 3 биты) и определяет, либо номер регистра, либо три дополнительных бита кода операции;
- Поле r/m содержит три последних значащих бита байта ModR/M (2, 1 и 0 биты) и может определять номер регистра, в котором находится операнд, или представлять совместно с полем mod, описанным выше, 32 возможных типа адресации.

32-х битные базовые и масштабируемые индексные формы адресации требуют наличия SIB байта. Определенная кодировка ModR/M байта сигнализирует о присутствии SIB байта. SIB байт состоит из следующих полей:

- Поле ss , занимающее два наиболее значимых бита (7 и 6 биты), определяет масштабный множитель;
- Поле index , занимающее следующие три бита за полем ss (5, 4, и 3 биты), определяет номер индексного регистра;
- Поле base , занимающее оставшиеся три бита (2, 1 и 0 биты), определяет номер базового регистра.

На рис. 6-2 показаны форматы ModR/M и SIB байтов.

Рис. 6-2 Формат байтов ModR/M и SIB



Значения ModR/M и SIB байтов и соответствующие им формы адресации показаны в таблицах 6-2, 6-3, 6-4. В таб. 6-2 представлены 16 битные формы адресации, определяемые байтом ModR/M. В таб. 6-3 представлены 32 битные формы адресации, определяемые байтом ModR/M. В таб. 6-4 представлены 32 битные формы адресации, определяемые байтом SIB.

Таб. 6-2 16-ти битные формы адресации, определяемые байтом ModR/M

r8(/r)			AL	CL	DL	BL	AH	CH	DH	BH
r16(/r)			AX	CX	DX	BX	SP	BP	SI	DI
r32(/r)			EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
/digit (Opcode)			0	1	2	3	4	5	6	7
REG =			000	001	010	011	100	101	110	111
Эффективный адрес	Mod	R/M	Значение ModR/M в шестнадцатеричной системе исчисления							
[BX+SI]	00	000	00	08	10	18	20	28	30	38
[BX+DI]		001	01	09	11	19	21	29	31	39
[BP+SI]		010	02	0A	12	1A	22	2A	32	3A
[BP+DI]		011	03	0B	13	1B	23	2B	33	3B
[SI]		100	04	0C	14	1C	24	2C	34	3C
[DI]		101	05	0D	15	1D	25	2D	35	3D
disp16		110	06	0E	16	1E	26	2E	36	3E
[BX]		111	07	0F	17	1F	27	2F	37	3F
[BX+SI]+disp8	01	000	40	48	50	58	60	68	70	78
[BX+DI]+disp8		001	41	49	51	59	61	69	71	79
[BP+SI]+disp8		010	42	4A	52	5A	62	6A	72	7A
[BP+DI]+disp8		011	43	4B	53	5B	63	6B	73	7B
[SI]+disp8		100	44	4C	54	5C	64	6C	74	7C
[DI]+disp8		101	45	4D	55	5D	65	6D	75	7D
[BP]+disp8		110	46	4E	56	5E	66	6E	76	7E
[BX]+disp8		111	47	4F	57	5F	67	6F	77	7F
[BX+SI]+disp16	10	000	80	88	90	98	A0	A8	B0	B8
[BX+DI]+disp16		001	81	89	91	99	A1	A9	B1	B9
[BP+SI]+disp16		010	82	8A	92	9A	A2	AA	B2	BA
[BP+DI]+disp16		011	83	8B	93	9B	A3	AB	B3	BB
[SI]+disp16		100	84	8C	94	9C	A4	AC	B4	BC
[DI]+disp16		101	85	8D	95	9D	A5	AD	B5	BD
[BP]+disp16		110	86	8E	96	9E	A6	AE	B6	BE
[BX]+disp16		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL		001	C1	C9	D1	D9	E1	E9	F1	F9
EDX/DX/DL		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AH		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/DH		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH		111	C7	CF	D7	DF	E7	EF	F7	FF

Примечание:

1. disp8 обозначает 8 битное смещение следующее за ModR/M байтом, которое должно быть знако-расширено до 16 бит и прибавлено к индексу.
2. disp16 обозначает 16 битное смещение следующее за ModR/M байтом, которое должно быть прибавлено к индексу.

Таб. 6-3 32-х битные формы адресации, определяемые байтом ModR/M

r8(/r)			AL	CL	DL	BL	AH	CH	DH	BH
r16(/r)			AX	CX	DX	BX	SP	BP	SI	DI
r32(/r)			EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
/digit (Opcode)			0	1	2	3	4	5	6	7
REG =			000	001	010	011	100	101	110	111
Эффективный адрес	Mod	R/M	Значение ModR/M в шестнадцатеричной системе исчисления							
[EAX]	00	000	00	08	10	18	20	28	30	38
[ECX]		001	01	09	11	19	21	29	31	39
[EDX]I]		010	02	0A	12	1A	22	2A	32	3A
[EBX]I]		011	03	0B	13	1B	23	2B	33	3B
[--][--]		100	04	0C	14	1C	24	2C	34	3C
disp32		101	05	0D	15	1D	25	2D	35	3D
[ESI]		110	06	0E	16	1E	26	2E	36	3E
[EDI]		111	07	0F	17	1F	27	2F	37	3F
[EAX]+disp8	01	000	40	48	50	58	60	68	70	78
[ECX]+disp8		001	41	49	51	59	61	69	71	79
[EDX]+disp8		010	42	4A	52	5A	62	6A	72	7A
[EBX]+disp8		011	43	4B	53	5B	63	6B	73	7B
[--][--]+disp8		100	44	4C	54	5C	64	6C	74	7C
[EBP]+disp8		101	45	4D	55	5D	65	6D	75	7D
[ESI]+disp8		110	46	4E	56	5E	66	6E	76	7E
[EDI]+disp8		111	47	4F	57	5F	67	6F	77	7F
[EAX]+disp32	01	000	40	48	50	58	60	68	70	78
[ECX]+disp32		001	41	49	51	59	61	69	71	79
[EDX]+disp32		010	42	4A	52	5A	62	6A	72	7A
[EBX]+disp32		011	43	4B	53	5B	63	6B	73	7B
[--][--]+disp32		100	44	4C	54	5C	64	6C	74	7C
[EBP]+disp32		101	45	4D	55	5D	65	6D	75	7D
[ESI]+disp32		110	46	4E	56	5E	66	6E	76	7E
[EDI]+disp32		111	47	4F	57	5F	67	6F	77	7F
EAX/AX/AL	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL		001	C1	C9	D1	D9	E1	E9	F1	F9
EDX/DX/DL		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AH		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/DH		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH		111	C7	CF	D7	DF	E7	EF	F7	FF

Примечание:
1. [-][-] обозначает, что за ModR/M байтом следует SIB байт.
2. disp8 обозначает 8 битное смещение следующее за SIB байтом, которое должно быть знако-расширено до 16 бит и прибавлено к индексу.
3. disp32 обозначает 32 битное смещение следующее за SIB байтом, которое должно быть прибавлено к индексу.

Таб. 6-4 32-х битные формы адресации, определяемые байтом SIB

r32			EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
Base =			0	1	2	3	4	5	6	7
Base =			000	001	010	011	100	101	110	111
Масштабный индекс	SS	Инд екс	Значение SIB в шестнадцатеричной системе исчисления							
[EAX]	00	000	00	01	12	13	24	25	36	37
[ECX]		001	08	09	1A	1B	2C	2D	3E	3F
[EDX]I]		010	10	11	12	13	14	15	16	17
[EBX]I]		011	18	19	1A	1B	1C	1D	1E	1F
нет		100	20	21	22	23	24	25	26	27
[EBP]		101	28	29	2A	2B	2C	2D	2E	2F
[ESI]		110	30	31	32	33	34	35	36	37
[EDI]		111	38	39	3A	3B	3C	3D	3E	3F
[EAX*2]	01	000	40	41	42	43	44	45	46	47
[ECX*2]		001	48	49	4A	4B	4C	4D	4E	4F
[EDX*2]		010	50	51	52	53	54	55	56	57
[EBX*2]		011	58	59	5A	5B	5C	5D	5E	5F
нет		100	60	61	62	63	64	65	66	67
[EBP*2]		101	68	69	6A	6B	6C	6D	6E	6F
[ESI*2]		110	70	71	72	73	74	75	76	77
[EDI*2]		111	78	79	7A	7B	7C	7D	7E	7F
[EAX*4]	01	000	80	81	82	83	84	85	86	87
[ECX*4]		001	88	89	8A	8B	8C	8D	8E	8F
[EDX*4]		010	90	91	92	93	94	95	96	97
[EBX*4]		011	98	99	9A	9B	9C	9D	9E	9F
нет		100	A0	A1	A2	A3	A4	A5	A6	A7
[EBP*4]		101	A8	A9	AA	AB	AC	AD	AE	AF
[ESI*4]		110	B0	B1	B2	B3	B4	B5	B6	B7
[EDI*4]		111	B8	B9	BA	BB	BC	BD	BE	BF
[EAX*8]	11	000	C0	C1	C2	C3	C4	C5	C6	C7
[ECX*8]		001	C8	C9	CA	CB	CC	CD	CE	CF
[EDX*8]		010	D0	D1	D2	D3	D4	D5	D6	D7
[EBX*8]		011	D8	D9	DA	DB	DC	DD	DE	DF
нет		100	E0	E1	E2	E3	E4	E5	E6	E7
[EBP*8]		101	E8	E9	EA	EB	EC	ED	EE	EF
[ESI*8]		110	F0	F1	F2	F3	F4	F5	F6	F7
[EDI*8]		111	F8	F9	FA	FB	FC	FD	FE	FF

Примечание:
[*] обозначает disp32 без базового регистра, если mod = 00, и [EBP] - в противном случае. Это обеспечивает следующие формы адресации:

disp32[index] (mod = 00)
disp8[EBP][index] (mod = 01)
disp32[EBP][index] (mod = 10)

6.2.2 Как читать описания команд

Формат описания команд процессоров:

Мнемоника Название команды

O D I T S Z A P C
Влияние на флаги

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	86
Информация о коде и времени выполнения команды						

После каждой такой таблицы следуют параграфы: "Операция", "Описание", "Особые ситуации защищенного режима", "Особые ситуации режима реальной адресации", "Особые ситуации виртуального-8086 режима" и иногда ,"Примечание". Ниже приведено объяснение принятых соглашений и аббревиатур, используемые для описания команд.

Влияние на флаги

В этой строке указано состояние битов регистра (E)FLAGS после выполнения операции. Используются следующие обозначения:

*	- флаг устанавливается по результатам операции;
пусто	- флаг не изменяется;
?	- флаг принимает неопределенное значение;
0	- флаг очищается;
1	- флаг устанавливается.

Колонка кода операции

В колонке кода операции содержится полный объектный код для всех возможных форм команды. По возможности, коды представлены шестнадцатеричной системой счисления в таком порядке, в котором они располагаются в памяти. Не шестнадцатеричные элементы определяются следующим образом.

/цифра

Цифра (от 0 до 7 включительно) сигнализирует, что ModR/M байт команды использует только r/m (регистр или память) операнд. Поле reg содержит цифру, используемую для расширения кода операции.

/r

Сигнализирует о том, что ModR/M байт содержит и регистровый операнд, и r/m операнд.

cb, cw, cd, cp

Однобайтная (cb), двухбайтная (cw), четырехбайтная (cd) или шестибайтная (cp) величина, следующая за кодом операции и определяющая значение смещения в кодовом сегменте или новое значение для регистра кодового сегмента.

ib, iw, id

Однобайтный (ib), двухбайтный (iw) или четырехбайтный (id) непосредственный операнд, следующий за кодом операции, ModR/M байтом или SIB байтом. Код операции определяет является ли операнд знаковой величиной. Все двухбайтные (word) и четырехбайтные (dword) значения приводятся начиная с младшего байта.

+rb, +rw, +rd

Код регистра (от 0 до 7), прибавляемый к байту, приведенному слева от знака плюс, для образования однобайтного кода. В следующей таблице приведены значения кодов.

rb	rw	rd
AL = 0	AX = 0	EAX = 0
CL = 1	CX = 1	ECX = 1
DL = 2	DX = 2	EDX = 2
BL = 3	BX = 3	EBX = 3
AH = 4	SP = 4	ESP = 4
CH = 5	BP = 5	EBP = 5
DH = 6	SI = 6	ESI = 6
BH = 7	DI = 7	EDI = 7

+i

Используется в командах с плавающей запятой, когда один из операндов - регистр стека сопроцессора ST(i). Число i (от 0 до 7) прибавляется к байту, приведенному слева от знака плюс, для получения однобайтного кода.

Колонка команды

В колонке команды приводится синтаксическое выражение команды (мнемокод). Список символов, используемых при представлении операндов в мнемокоде команды:

rel8

Относительный адрес в диапазоне от -128 байт до +127 байт от конца команды.

rel16, rel32

Относительный адрес внутри того же кодового сегмента, в котором команда ассемблировалась. rel16 применима к командам с атрибутом 16 битного размера операнда; rel32 применима к командам с атрибутом 32 битного размера операнда.

ptr16:16, ptr16:32

Дальний указатель (обычно в кодовый сегмент, отличный от кодового сегмента команды). Употребление ptr16:16 сигнализирует о том, что значение указателя состоит из двух частей. Значение, приведенное слева от двоеточия, это 16 битный селектор или значение для регистра кодового сегмента. Значение справа от двоеточия соответствует смещению в сегменте назначения. ptr16:16 используется в командах с атрибутом 16 битного размера операнда. ptr16:32 используется в командах с атрибутом 32 битного размера операнда.

r8

Один из однобайтных регистров AL, CL, DL, BL, AH, CH, DH или BH.

r16

Один из двухбайтных регистров AX, CX, DX, BX, SP, BP, SI или DI.

r32

Один из четырехбайтных регистров EAX, ECX, EDX, EBX, ESP, EBP, ESI или EDI.

imm8

Непосредственная однобайтная величина. imm8 - это знаковое число от -128 до +127 включительно. Для команд, в которых imm8 используется совместно с двухбайтным или четырехбайтным операндом, непосредственная величина imm8 знакорасширяется до слова или двойного слова. Старший байт слова заполняется старшим битом непосредственной величины.

imm16

Непосредственная двухбайтная величина, используемая в командах с атрибутом 16 битного размера операнда. Это число от -32768 до +32767 включительно.

imm32

Непосредственная четырехбайтная величина, используемая в командах с атрибутом 32 битного размера операнда. Она позволяет использовать числа от -2147483648 до +2147483647 включительно.

r/m8

Однобайтный операнд, который представляет собой содержимое байтового регистра (AL, CL, DL, BL, AH, CH, DH или BH) или байта памяти.

r/m16

Двухбайтный регистр или операнд в памяти, используемый в командах с атрибутом 16 битного размера операнда. Двухбайтные регистры: AX, CX, DX, BX, SP, BP, SI, DI. Содержимое памяти находится по адресу, получаемому при вычислении эффективного адреса.

r/m32

Четырехбайтный регистр или операнд в памяти, используемый в командах с атрибутом 32 битного размера операнда. Четырехбайтные регистры: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI. Операнд в памяти находится по адресу, получаемому при вычислении эффективного адреса.

r/m64

Восьмибайтный регистр или операнд в памяти, используемый в командах с атрибутом 64 битного размера операнда. Поле reg/opcode представляет собой код операции. Операнд в памяти находится по адресу, получаемому при вычислении эффективного адреса.

m

16 или 32 битный операнд в памяти.

m8

Байт памяти адресуемый DS:(E)SI или ES:(E)DI (используется только строковыми командами).

m16

Слово памяти адресуемое DS:(E)SI или ES:(E)DI (используется только строковыми командами).

m32

Двойное слово памяти адресуемое DS:(E)SI или ES:(E)DI (используется только строковыми командами).

m16:16, m16:32

Операнд в памяти, содержащий дальний указатель, состоящий из двух чисел. Число слева от двоеточия соответствует селектору сегмента указателя, число справа соответствует его смещению. Для команд с такими операндами разрешены все режимы адресации.

m16&32, m16&16, m32&32

Операнд в памяти, состоящий из пар элементов данных, размеры которых указаны слева и справа от знака амперсанда. Для команд с такими операндами разрешены все режимы адресации. m16&16 и m32&32 используются в команде BOUND для хранения верхней и нижней границ индексов массива. m16&32 используется командами LIDT и LGDT для обеспечения загрузки значения поля предела (из слова) и значения поля базы (из двойного слова) соответствующих регистров глобальной таблицы дескрипторов и таблицы дескрипторов прерываний (GDTR и IDTR).

moffs8, moffs16, moffs32

Простая переменная в памяти (смещение в памяти) типа BYTE, WORD или DWORD, используемая некоторыми вариантами команды MOV. Фактический адрес получается при помощи простого смещения относительно базы сегмента. ModR/M байт в этой команде не используется. Число, указанное в moffs, определяет его размер, который устанавливается атрибутом размера адреса команды.

Sreg

Сегментный регистр. Значения битов определяют сегментный регистр (ES=0, CS=1, SS=2, DS=3, FS=4, GS=5).

m32real, m64real, m80real

Вещественные операнды FPU в памяти (соответственно Short-real, Long-real, Temp-real).

m16int, m32int, m64int

Целочисленные операнды FPU в памяти (соответственно Word-integer, Short-integer, Long-integer).

m80dec

Двоично-десятичный операнд FPU в памяти.

mNbyte

N-байтный операнд FPU в памяти.

ST или ST(0)

Верхний элемент стека регистров сопроцессора.

ST(i)

i-ый элемент от вершины стека сопроцессора.

Колонка времени выполнения

Колонка времени выполнения содержит аппроксимированное число временных циклов (тактов), требующихся для выполнения команды. Подсчет числа необходимых на выполнение тактов основан на следующих предположениях.

- Выбор данных и команд осуществляется из КЭШ.
- Цель команды JUMP находится в КЭШ.
- Отсутствуют циклы, конкурирующие с командой за использование КЭШ.
- Транслируемая страница представлена в TLB.
- Операнды в памяти выровнены.
- При вычислении эффективного адреса используется базовый регистр, который не был назначением предшествующей команды.
- При выполнении команды не было исключений.
- Запись в буфер осуществляется без задержки.

Обозначения, используемые при описании количества тактов:

n

Представляет количество повторений.

m

Представляет собой количество компонент в следующей выполняемой команде, где все байты смещения (если таковое имеется) считаются одной компонентой, все непосредственные значения (если таковые имеются) считаются одной компонентой, любой другой байт команды и префикса(ов) считается за одну компоненту.

pm=

Количество тактов необходимое для выполнения команды в защищенном режиме. Если значение pm= не приводится, это означает, что оно точно такое, как и для реального режима.

rm=

Количество тактов необходимое для выполнения команды в режиме реальной адресации. Обычно символы rm= опускаются.

vm=

Количество тактов необходимое для выполнения команды в режиме V86. Если значение vm= не приводится, это означает, что оно точно такое, как и для реального режима.

Если во время выполнения команды возникает особая ситуация, а обработчик исключений это другая задача, то время выполнения команды увеличивается на количество тактов, необходимое для переключения задачи. Это время зависит от нескольких факторов:

- типа TSS, используемого для представления новой задачи (32 бит TSS или 16 бит TSS);
- выполняется ли текущая задача в режиме V86;
- выполняется ли новая задача в режиме V86;
- были ли промахи при обращении к КЭШ;
- используется ли в качестве шлюза задачи шлюз прерывания или шлюз ловушки.

Таблица 6-5 содержит времена переключения задач для особых ситуаций, предполагая попадание в КЭШ и использования шлюза задачи.

Таб. 6-5 Время переключения задач для особых ситуаций

Старая задача	Новая задача		
	для 32 битного TSS	для 16 битного TSS	для VM TSS
VM/32 битный/16 битный	85	87	71

Колонка примеров

Колонка примеров, следующая за колонкой времени выполнения команды, содержит возможную запись команды на языке ассемблера TASM.

Операция (функциональное описание)

Параграф "Операция" содержит алгоритм выполнения команды на языке подобном ALGOL или PASCAL. В алгоритмах используются следующие конструкции:

- Комментарии выделяются парами символов "(*" и "*)".

- Составные выражения заключаются между ключевыми словами "if" выражение (IF, THEN, ELSE, FI), или "do" выражение (DO, OD), или "case" выражение (CASE...OF, ESAC).
- Операция продолжается до тех пор, пока не встретится ключевое слово END.
- Под именем регистра подразумевается его содержимое. Имя регистра, заключенное в скобки, подразумевает содержимое ячейки памяти, адрес которой содержится в этом регистре. Например, ES:[DI] указывает на содержимое ячейки памяти сегмента ES с относительным адресом в регистре DI. [SI] указывает на содержимое ячейки памяти по адресу содержащемуся в регистре SI относительно его сегмента по умолчанию (сегмента DS) или замещающего сегмента.
- Квадратные скобки также используются для операндов в памяти, где они (скобки) подразумевают, что содержимое ячейки памяти является относительным смещением в сегменте. Например, [SRC] указывает на то, что содержимое операнда источника является относительным смещением в сегменте.
- A = B указывает, что величине A присваивается значение B.
- Символы =, ≠, ≥, ≤ являются операторами отношения, используемыми для сравнения двух величин. Выражение отношения, такое как A = B является истинным (TRUE), если значение A равно B, в противном случае - ложным (FALSE).

В алгоритмических описаниях используются следующие идентификаторы.

OperandSize

Представляет собой атрибут размера операнда команды, который может быть 16 или 32 битным.

AddressSize

Представляет собой атрибут размера адреса, который может быть 16 или 32 битным.

Например:

```
IF (команда CMPSW)
THEN OperandSize = 16;
ELSE
  IF (команда CMPSD)
  THEN OperandSize = 32;
  FI;
FI;
```

Пример указывает на то, что атрибут размера операнда зависит от используемой формы команды CMPS. Описание атрибутов размера адреса и размера операнда приведено выше.

StackAddrSize

Представляет собой соответствующий команде атрибут размера адреса стека, который может быть 16 или 32 битным.

SRC

Представляет собой операнд-источник. Из двух операндов команды, SRC является правым операндом.

DEST

Представляет собой операнд-назначение. Из двух операндов команды, DEST является левым операндом.

LeftSRC, RightSRC

Представляют собой операнды-источники. Применяются в случае, если в команде оба операнда являются источниками.

eSP

Представляет собой, либо регистр SP, либо регистр ESP, в зависимости от установки B-бита дескриптора текущего сегмента стека.

Функции, используемые в описании алгоритма.

Trunkate_to_16bits(величина)

Уменьшает размер величины до 16 бит, отбрасывая старшие биты, если это необходимо.

Addr(операнд)

Возвращает эффективный адрес операнда (вычисление эффективного адреса производится без добавления базы сегмента).

ZeroExtended(величина)

Возвращает величину, дополненную нулями до размера операнда команды. Например, если OperandSize = 32, тогда ZeroExtended от однобайтной величины -10 преобразует байт F6h в четырехбайтную величину 000000F6h. Эта функция оставляет величину без изменения в том случае, если атрибут размера операнда и размера величины совпадают.

SignExtended(величина)

Возвращает величину, знакорасширенную до размера операнда команды. Например, если OperandSize = 32, тогда SignExtended от однобайтной величины -10 преобразует байт F6h в четырехбайтную величину FFFFFFF6h. Эта функция оставляет величину без изменения в том случае, если атрибут размера операнда и размер величины совпадают.

Push(value)

Помещает величину в стек. Количество байтов, помещаемых в стек, определяется атрибутом размера операнда команды. Push действует следующим образом:

```
IF StackAddrSize = 16
THEN
  IF OperandSize = 16
  THEN
    SP = SP - 2;
    SS:[SP] = value;
  ELSE (* OperandSize = 32 *)
    SP = SP - 4;
    SS:[SP] = value;
  FI;
ELSE (* StackAddrSize = 32 *)
  IF OperandSize = 16
  THEN
    ESP = ESP - 2;
    SS:[ESP] = value;
  ELSE (* OperandSize = 32 *)
    ESP = ESP - 4;
    SS:[ESP] = value;
  FI;
FI;
```

Примечание: иногда, для указания того, что в стек помещается именно четырехбайтная величина, пишется не Push(value) а Pushd(value).

Pop()

Возвращает величину с вершины стека. Выражение EAX = Pop() помещает в EAX 32 битную величину, которую Pop() берет с вершины стека. Pop может возвращать как слова, так и двойные слова, в зависимости от значения атрибута размера операнда. Pop действует следующим образом:

```
IF StackAddrSize = 16
THEN
  IF OperandSize = 16
  THEN
    ret val = SS:[SP]; (* двухбайтная величина *)
    SP = SP + 2;
  ELSE (* OperandSize = 32 *)
    ret val = SS:[SP]; (* четырехбайтная величина *)
    SP = SP + 4;
  FI;
ELSE (* StackAddrSize = 32 *)
  IF OperandSize = 16
  THEN
    ret val = SS:[ESP]; (* двухбайтная величина *)
    ESP = ESP + 2;
  ELSE (* OperandSize = 32 *)
    ret val = SS:[ESP]; (* четырехбайтная величина *)
    ESP = ESP + 4;
  FI;
FI;
RETURN(ret val); (* возвращает слово или двойное слово *)
```

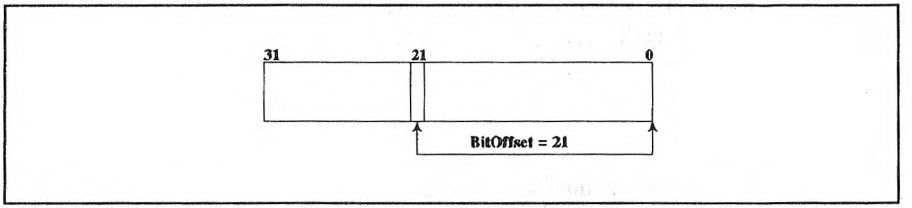
Pop(ST)

Используется при описании команд с плавающей запятой и обозначает выборку значения из стека регистров сопроцессора.

Bit[BitBase, BitOffset]

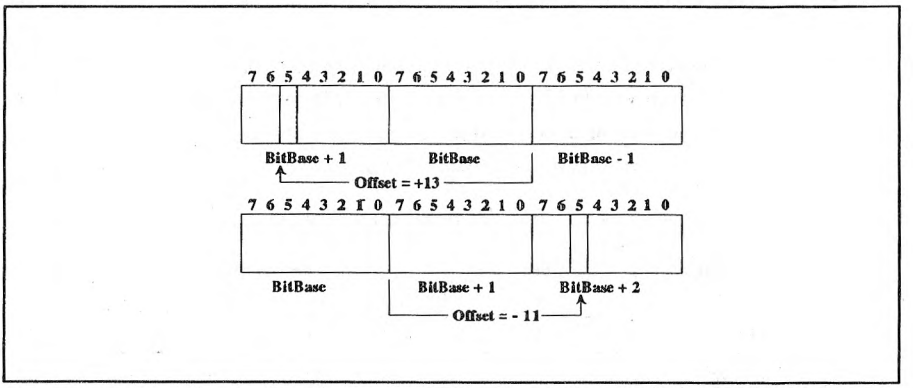
Возвращает значение бита из строки битов, которая является последовательностью битов в памяти или в регистре. Нумерация битов идет в порядке возрастания и начинается с самого младшего бита содержимого регистров или памяти. Слово в памяти хранится таким образом, что младший байт слова располагается по меньшему адресу, чем старший. Смещение бита (BitOffset) может принимать значения от 0 до 31, если в качестве базового операнда (BitBase) используется регистр. Это смещение адресует бит в указанном регистре. На рис. 6-3 приведен пример Bit[EAX,21].

Рис. 6-3. Смещение BitOffset для выражения Bit[EAX,21]



Смещение бита может принимать значения от -2 Гбит до +2 Гбит в том случае, если BitBase является адресом памяти. Номер адресуемого бита в байте - (Offset MOD 8), этот байт находится по адресу (BitBase + (BitOffset DIV 8)), где DIV - целочисленное деление со знаком и округлением в сторону -∞ , а MOD возвращает положительное число. Это показано на рис. 6-4.

Рис. 6-4 Индексация бита памяти



I-O-Permission(I-O-Address, width)

Возвращает значение TRUE или FALSE в зависимости от значения маски разрешения ввода-вывода и других факторов. Эта функция работает следующим образом:

```
IF (TSS 16 битный) THEN RETURN (FALSE); FI;
Ptr = [Tss + 66]; (* выбирается указатель битовой карты *)
BitStringAddr = CHR(I-O-Address, 3) + Ptr;
MaskShift = I-O-Address AND 7;
CASE width OF:
  BYTE: nBitMask = 1;
  WORD: nBitMask = 3;
```

```
  DWORDE: nBitMask = 15;
ESAC;
mask = SHL(nBitMask, MaskShift);
CheckString = [BitStringAddr] AND mask;
IF CheckString = 0;
THEN RETURN (TRUE);
ELSE RETURN (FALSE);
FI;
```

Switch-Tasks

Это функция переключения задачи.

Описание

Параграф "Описание" содержит дополнительную информацию о действии команды.

Особые ситуации защищенного режима

В этом параграфе описываются особые ситуации, которые могут иметь место при выполнении команды в защищенном режиме. Все особые ситуации обозначаются следующим образом: вначале идет знак #, затем две буквы английского алфавита, после чего в скобках может присутствовать код ошибки. Например, #GP(0) обозначает особую ситуацию - ошибка общей защиты с кодом ошибки равным нулю. В таб. 6-6 приведены обозначения особых ситуаций и соответствующие им номера прерываний.

Таб. 6-6 Особые ситуации

Мнемоника	Прерывание	Описание
#UD	6	Неопределенная операция
#NM	7	Сопроцессор отсутствует
#DF	8	Двойная ошибка
#TS	10	Неправильный TSS
#NP	11	Сегмент не присутствует
#SS	12	Ошибка стека
#GP	13	Общая защита
#PF	14	Страничная ошибка
#MF	16	Ошибка сопроцессора
#AC	17	Контроль выравнивания

Прикладные программисты могут обратиться к документации, прилагаемой к операционной системе, для уточнения действий ОС при возникновении особой ситуации.

Особые ситуации режима реальной адресации

В режиме реальной адресации процессор осуществляет меньшее число проверок, поэтому в этом режиме меньше условий для возникновения особых ситуаций.

Особые ситуации виртуального-8086 режима

Задачи режима V86 обеспечивают возможность эмулировать виртуальные машины 8086. Особые ситуации V86 подобны особым ситуациям процессора 8086, за некоторым исключением.

6.3 Описание команд процессора

AAA ASCII коррекция после сложения

O	D	I	T	S	Z	A	P	C
?				?	?	*	?	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
37	AAA	3	3	4	3	8	aaa

Операция

```
ALcarry = AL > 0F9h; (* 1 если истина *)
IF ((AL AND 0Fh) > 9) OR (AF = 1)
THEN
  AL = (AL + 6) AND 0Fh;
  AH = AH + 1 + ALcarry;
  AF = 1;
  CF = 1;
ELSE
  AF = 0;
  CF = 0;
  AL = AL AND 0Fh;
FI;
```

Описание

Выполняйте команду AAA только после выполнения команды ADD, которая помещает однобайтный результат в регистр AL. Младшая тетрада операнда команды ADD должна быть в диапазоне от 0 до 9 (BCD цифры). В этом случае команда AAA корректирует регистр AL таким образом, что он будет содержать правильную десятичную цифру. Если при сложении происходит десятичный перенос, тогда регистр AH увеличивается на единицу, а также устанавливаются флаги CF и AF. Если при этом же сложении в старшей тетраде AL получилось значение Fh, тогда регистр AH увеличивается на единицу еще раз. Если при сложении десятичного переноса не возникло, тогда сбрасываются флаги CF и AF, а регистр AH не изменяется. В любом случае, старшая тетрада регистра AL обнуляется. Для

того, чтобы преобразовать содержимое регистра AL к ASCII формату, необходимо после команды AAA выполнить команду OR AL,30h.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

AAD ASCII коррекция AX перед делением

O	D	I	T	S	Z	A	P	C
?				*	*	?	*	?

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
D5 0A	AAD	10	14	19	14	60	aad

Операция

```
regAL = AL;
regAL = AH;
AL = (regAH*imm8 + regAL) AND 0FFh;
AH = 0;
```

Примечание

imm8 является вторым байтом кода операции. При ассемблировании команды AAM imm8 устанавливается в значение равное 0Ah, тем не менее, возможно явное изменение этого байта, что может привести к другому результату, согласно пункту "Операция".

Описание

Команда AAD используется для подготовки двух неупакованных BCD цифр (наименее значащая цифра в регистре AL, наиболее значащая цифра

в регистре AH) для операции деления, которая возвращает неупакованный результат. Это осуществляется путем установки регистра AL в значение AL + (второй байт кода операции * AH) с последующей очисткой регистра AH. После команды AAD регистр AX будет равен двоичному эквиваленту оригинального неупакованного двухзначного числа.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

AAM ASCII коррекция AX после умножения

O	D	I	T	S	Z	A	P	C
?				*	*	?	*	?

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
D4 0A	AAM	18	15	17	16	83	aam

Операция

```
regAL = AL;
AH = regAL / imm8;
AL = regAL MOD imm8;
```

Примечание

imm8 является вторым байтом кода операции. При ассемблировании команды AAM imm8 устанавливается в значение равное 0Ah, тем не менее, возможно явное изменение этого байта, что может привести к другому результату, согласно пункту "Операция".

Описание

Выполняйте команду AAM только после выполнения команды MUL над двумя неупакованными BCD цифрами, результат которой помещается в регистр AX. Поскольку результат всегда меньше 100, то он полностью содержится в регистре AL. Команда AAM распаковывает результат умножения, содержащийся в регистре AL, деля его на второй байт кода операции. Частное (наиболее значащая цифра) помещается в регистр AH, а остаток (наименее значащая цифра) - в регистр AL.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

AAS ASCII коррекция после вычитания

O	D	I	T	S	Z	A	P	C
?				?	?	*	?	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
3F	AAS	3	3	4	3	8	aas

Операция

```
ALborrow = AL<6;   (* 1 если истина *)
IF (AL AND 0Fh) > 9 OR AF = 1
THEN
  AL = (AL - 6) AND 0Fh;
  AH = AH - 1 - ALborrow;
  AF = 1;
  CF = 1;
ELSE
  CF = 0;
  AF = 0;
  AL = AL AND 0Fh;
FI;
```

Описание

Выполняйте команду AAS только после выполнения команды SUB, которая помещает однобайтный результат в регистр AL. Младшая тетрада операнда команды SUB должна быть в диапазоне от 0 до 9 (BCD цифры). В этом случае команда AAS корректирует регистр AL таким образом, что он будет содержать правильную десятичную цифру. Если при сложении происходит десятичный перенос, тогда регистр AH увеличивается на единицу, а также устанавливаются флаги CF и AF. Если при этом же сложении в старшей тетраде AL получилось значение Fh, тогда регистр AH увеличивается на единицу еще раз. Если при сложении десятичного переноса не возникло, тогда сбрасываются флаги CF и AF, а регистр AH не изменяется. В любом случае, старшая тетрада регистра AL обнуляется. Для того, чтобы преобразовать содержимое регистра AL к ASCII формату, необходимо после команды AAS выполнить команду OR AL,30h.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

ADC Сложение с переносом

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
10 /r	ADC r/m8,r8	1/3	1/3	2/7	2/7	3/16+EA	adc ah,al
11 /r	ADC r/m16,r16	1/3	1/3	2/7	2/7	3/16+EA	adc word ptr m16,dx
11 /r	ADC r/m32,r32	1/3	1/3	2/7			adc ecx,ebx
12 /r	ADC r8,r/m8	1/2	1/2	2/6	2/7	3/9+EA	adc ah,sum
13 /r	ADC r16,r/m16	1/2	1/2	2/6	2/7	3/9+EA	adc di,ax
13 /r	ADC r32,r/m32	1/2	1/2	2/6			adc ecx,raznost
14 ib	ADC AL,imm8	1	1	2	3	4	adc al,4
15 iw	ADC AX,imm16	1	1	2	3	4	adc ax,03FDh
15 id	ADC EAX,imm32	1	1	2			adc eax,23456789h
80 /2 ib	ADC r/m8,imm8	1/3	1/3	2/7	3/7	4/17+EA	adc byte ptr [si],5
81 /2 iw	ADC r/m16,imm16	1/3	1/3	2/7	3/7	4/17+EA	adc di,0ABFh

```
81 /2 id     ADC r/m32,imm32 1/3     1/3   2/7                        adc ecx,0CAAAAh
83 /2 ib     ADC r/m16,imm8  1/3     1/3   2/7   3/7   4/17+EA        adc bx,0ABh
83 /2 ib     ADC r/m32,imm8  1/3     1/3   2/7                        adc edx,0Ah
```

Операция

DEST = DEST + SRC + CF;

Описание

Команда ADC осуществляет целочисленное сложение двух операндов DEST и SRC и флага переноса CF. Результат сложения помещается на место первого операнда (DEST), и соответственно результату устанавливаются флаги. Команда ADC обычно используется как часть в многобайтных или многословных (multi-word) операциях сложения. При сложении непосредственного однобайтного значения с двухбайтным или четырехбайтным операндом, непосредственная величина прежде всего знакорасширяется до размеров операнда, и только после этого выполняется сложение.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

ADD Сложить

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
04 ib	ADD AL,imm8	1	1	2	3	4	add al,17
05 iw	ADD AX,imm16	1	1	2	3	4	add ax,0AABh
05 id	ADD EAX,imm32	1	1	2			add eax,0AAC23h
80 /0 ib	ADD r/m8,imm8	1/3	1/3	2/7	3/7	4/17+EA	add bl,5
81 /0 iw	ADD r/m16,imm16	1/3	1/3	2/7	3/7	4/17+EA	add bx,0A4Fh
81 /0 id	ADD r/m32,imm32	1/3	1/3	2/7			add edx,0CAAAAh
83 /0 ib	ADD r/m16,imm8	1/3	1/3	2/7	3/7	4/17+EA	add word ptr [di],7
83 /0 ib	ADD r/m32,imm8	1/3	1/3	2/7			add ecx,0Fh
00 /r	ADD r/m8,r8	1/3	1/3	2/7	2/7	3/16+EA	add ch,al
01 /r	ADD r/m16,r16	1/3	1/3	2/7	2/7	3/16+EA	add word ptr m32,dx
01 /r	ADD r/m32,r32	1/3	1/3	2/7			add edx,ebx
02 /r	ADD r8,r/m8	1/2	1/2	2/6	2/7	3/9+EA	add ch,sum
03 /r	ADD r16,r/m16	1/2	1/2	2/6	2/7	3/9+EA	add si,ax
03 /r	ADD r32,r/m32	1/2	1/2	2/6			add edi,raznost

Операция

DEST = DEST + SRC;

Описание

Команда ADD осуществляет целочисленное сложение двух операндов (DEST и SRC). Результат сложения помещается на место первого операнда (DEST). Флаги устанавливаются в соответствии с полученным результатом. При сложении непосредственного однобайтного значения с двухбайтным или четырехбайтным операндом, непосредственная величина прежде всего знакорасширяется до размеров операнда, и только после этого выполняется сложение.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реа́льной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

AND Логическое "И"

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
20 /r	AND r/m8,r8	1/3	1/3	2/7	2/7	3/16+EA	and [bx],ah
21 /r	AND r/m16,r16	1/3	1/3	2/7	2/7	3/16+EA	and cx,di
21 /r	AND r/m32,r32	1/3	1/3	2/7			and memory,ecx
22 /r	AND r8,r/m8	1/2	1/2	2/6	2/7	3/9+EA	and ch,sum
23 /r	AND r16,r/m16	1/2	1/2	2/6	2/7	3/9+EA	and di,[si]
23 /r	AND r32,r/m32	1/2	1/2	2/6			and ecx,raznost
24 ib	AND AL,imm8	1	1	2	3	4	and al,4
25 iw	AND AX,imm16	1	1	2	3	4	and ax,03FDh
25 id	AND EAX,imm32	1	1	2			and eax,23456789h
80 /4 ib	AND r/m8,imm8	1/3	1/3	2/7	3/7	4/17+EA	and byte ptr [bx],5
81 /4 iw	AND r/m16,imm16	1/3	1/3	2/7	3/7	4/17+EA	and dx,0DBBh
81 /4 id	AND r/m32,imm32	1/3	1/3	2/7			and ebx,0CAAAAh
83 /4 ib	AND r/m16,imm8	1/3	1/3	2/7	3/7	4/17+EA	and cx,0ABh
83 /4 ib	AND r/m32,imm8	1/3	1/3	2/7			and edx,0Ah

Операция

```
DEST = DEST AND SRC;
CF = 0;
OF = 0;
```

Описание

Каждый бит результата устанавливается в 1, в том случае, если соответствующие биты операндов установлены в 1, в противном случае - в 0.

DEST	SRC	AND
0	0	0
0	1	0
1	0	0
1	1	1

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сег-

ментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

ARPL Коррекция запрашиваемого уровня привилегий (поля RPL селектора)

O	D	I	T	S	Z	A	P	C
					*			

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
63 /r	ARPL r/m16,r16	pm=7	9/9	pm=20/21	pm=10/11	arpl selector,dx

Операция

```
IF ((RPL биты 0..1 в DEST) < (RPL биты 0..1 в SRC))
THEN
  ZF = 1;
  (RPL биты 0..1 в DEST) = (RPL биты 0..1 в SRC);
ELSE
  ZF = 0;
FI;
```

Описание

У команды ARPL два операнда. Первый операнд - это 16 битная переменная в памяти или двухбайтный регистр, который содержит значение селектора. Второй операнд - двухбайтный регистр. Если поле RPL (запрашиваемый уровень привилегий - младшие два бита) первого операнда меньше, чем поле RPL второго операнда, то устанавливается флаг ZF, а

поле RPL первого операнда увеличивается до значения поля RPL второго операнда. В противном случае сбрасывается флаг ZF, и первый операнд не изменяется.

Команда ARPL используется в системном программном обеспечении. Эта команда позволяет убедиться в том, что селектор переданный процедуре не требует более высокого уровня привилегий, чем позволено вызывающей процедуре. Обычно, вторым операндом команды ARPL является регистр, содержащий значение селектора CS вызывающей процедуры.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 6, команда ARPL не распознается в режиме реальной адресации.

Особые ситуации режима V86

#UD, команда ARPL не распознается в режиме V86.

BOUND Проверить индекс на выход за границы массива

O D I T S Z A P C

Код операции	Команда	Такты Pentium	486	386	286	Пример
62 /r	BOUND r16,m16&16	8 (within bounds) int+32 (out of bounds)	7	10	13	bound ax,array
62 /r	BOUND r32,m32&32	8 (within bounds) int+32 (out of bounds)	7	10		bound edx,arr

Операция

```
IF (LeftSRC < [RightSRC] OR LeftSRC > [RightSRC + OperandSize/8])
  (* под нижней границей или над верхней границей *)
```

```
    THEN прерывание 5;
FI;
```

Описание

Команда BOUND позволяет убедиться в том, что знаковый индекс массива находится в пределах, определенных в блоке памяти, состоящем из верхней и нижней границ. Каждая граница занимает слово, при атрибуте размера операнда равном 16 бит, и двойное слово, при атрибуте размера операнда равном 32 бита. Первый операнд (всегда регистр) должен быть больше или равен первой границе в памяти (нижней границе) и меньше или равен сумме второй границы в памяти (верхней границы) и количества байт, отведенных размеру операнда. Если значение регистра не находится в пределах границ, то происходит прерывание 5, EIP возврата из которого указывает на команду BOUND.

Данные структуры пределов границ помещаются обычно непосредственно перед массивом, делая тем самым пределы адресуемыми при помощи постоянного смещения от начала массива.

Особые ситуации защищенного режима

Прерывание 5, если выше описанный тест на проверку границ потерпел неудачу. #GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Второй операнд должен быть операндом в памяти, но не регистром. #UD, если в команде BOUND поле ModR/M задает регистр в качестве второго операнда.

Особые ситуации режима реальной адресации

Прерывание 5, если выше описанный тест на проверку границ потерпел неудачу. Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh. Прерывание 6, если второй операнд - регистр.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

BSF Сканирование битов вперед

O	D	I	T	S	Z	A	P	C
					*			

Код операции	Команда	Такты			Пример
		Pentium	486	386	
0F BC	BSF r16,r/m16	6-34/6-35	6-42/7-43	10+3n	bsf ax,bx
0F BC	BSF r32,r/m32	6-42/6-43	6-42/7-43	10+3n	bsf eax,edx

Операция

```
IF r/m = 0
THEN
  ZF = 0;
  register = UNDEFINED;
ELSE
  temp = 0;
  ZF = 0;
  WHILE BIT[r/m, temp] = 0
  DO
    temp = temp + 1;
    register = temp;
  OD;
FI;
```

Описание

Команда BSF сканирует биты второго операнда (слово или двойное слово), начиная с бита 0. Если все биты равны нулю, то устанавливается флаг ZF, в противном случае флаг ZF сбрасывается, и в регистр назначения (первый операнд) помещается номер первого установленного бита.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегмен-

те SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

BSR Сканирование битов назад

O	D	I	T	S	Z	A	P	C
					*			

Код операции	Команда	Такты			Пример
		Pentium	486	386	
0F BD	BSR r16,r/m16	7-39/7-40	6-103/7-104	10+3n	bsr ax,bitmask
0F BD	BSR r32,r/m32	7-71/7-72	6-103/7-104	10+3n	bsr eax,edx

Операция

```
IF r/m = 0
THEN
  ZF = 1;
  register = UNDEFINED;
ELSE
  temp = OperandSize - 1;
  ZF = 0;
  WHILE BIT[r/m,temp] = 0
  DO
    temp = temp - 1;
    register = temp;
  OD;
FI;
```

Описание

Команда BSR сканирует биты второго операнда (слово или двойное слово), начиная с наиболее значащего бита, и заканчивая наименее значащим. Если все биты равны нулю, то устанавливается флаг ZF, в противном случае флаг ZF сбрасывается, и в регистр назначения (первый опе-

ранд) помещается номер первого установленного бита, найденного при сканировании в обратном направлении.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS; ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

BSWAP Перестановка байтов

O D I T S Z A P C

Код операции	Команда	Такты		Пример
		Pentium	486	
0F C8+rd	BSWAP r32	1	1	bswap ebx

Операция

```
TEMP = r32
r32(7..0) = TEMP(31..24)
r32(15..8) = TEMP(23..16)
r32(23..16) = TEMP(15..8)
r32(31..24) = TEMP(7..0)
```

Описание

Команда BSWAP меняет порядок следования байтов 32 битного регистра на противоположный. В случае использования команды BSWAP с ат-

рибутом 16 битного размера операнда, результат, помещаемый в регистр назначения, неопределен.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

BT Проверка бита

O	D	I	T	S	Z	A	P	C
								*

Код операции	Команда	Такты			Пример
		Pentium	486	386	
0F A3	BT r/m16,r16	4/9	3/8	3/12	bt [di],ax
0F A3	BT r/m32,r32	4/9	3/8	3/12	bt edi,eax
0F BA /4 ib	BT r/m16,imm8	4	3/3	3/6	bt bx,3
0F BA /4 ib	BT r/m32,imm8	4	3/3	3/6	bt edx,17

Операция

```
CF = BIT[LeftSRC, RightSRC];
```

Описание

Команда BT сохраняет значение одного бита из первого операнда во флаге CF. Позиция сохраняемого бита из первого операнда находится во втором операнде.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Примечание

Значение индекса выбираемого бита может быть представлено непосредственным значением в команде BT или значением в общем регистре. В этой команде используется только 8 битное непосредственное значение. Значение этого операнда берется по модулю 32, таким образом смещение битов находится в диапазоне от 0 до 31. Это позволяет выбирать любой бит внутри регистра. Для битовых строк в памяти это поле непосредственного значения дает только смещение внутри слова или двойного слова.

Некоторые ассемблеры поддерживают значения битовых смещений больше 31, используя поле непосредственного значения (imm8) в комбинации с полем смещения операнда в памяти(disp). В этом случае ассемблером младшие 3 или 5 битов (3 для 16 битных операндов, 5 для 32 битных операндов) смещения бита (второй операнд команды) сохраняются в поле непосредственного операнда, а старшие биты сдвигаются и комбинируются с полем смещения (disp). Процессор же игнорирует ненулевые значения старших битов поля imm8.

При доступе к памяти процессор обращается к четырем байтам, начинающимся по полученному следующим образом адресу:

Effective Address + (4 * (BitOffset DIV 32))

для размера операнда 32 бита, или к двум байтам, начинающимся по адресу:

Effective Address + (2 * (BitOffset DIV 16))

для 16 битного размера операнда. Такое обращение происходит, даже если необходим доступ только к одиночному байту. Поэтому, избегайте ссылок к областям памяти, близким к пустым адресным пространствам. В частности, избегайте ссылок на распределенные в памяти регистры ввода-вывода. Вместо этого используйте команду MOV для загрузки и сохранения значений по таким адресам и регистровую форму команды BT для работы с данными.

BTC Проверка и дополнение бита

O	D	I	T	S	Z	A	P	C
								*

Код операции	Команда	Такты			Пример
		Pentium	486	386	
0F BB	BTC r/m16,r16	7/13	6/13	6/13	btc [di],ax
0F BB	BTC r/m32,r32	7/13	6/13	6/13	btc [edi],eax
0F BA /7 ib	BTC r/m16,imm8	7/8	6/8	6/8	btc [bx],05
0F BA /7 ib	BTC r/m32,imm8	7/8	6/8	6/8	btc [ebx],02

Операция

```
CF = BIT[LeftSRC, RightSRC];
BIT[LeftSRC, RightSRC] = NOT BIT[LeftSRC, RightSRC];
```

Описание

Команда BTC сохраняет значение бита из первого операнда со смещением, указанным вторым операндом, во флаге CF, а затем дополняет (инвертирует) этот бит.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Примечание

Значение индекса выбираемого бита может быть представлено непосредственным значением в команде BTC или значением в общем регистре. В этой команде используется только 8 битное непосредственное значение. Значение этого операнда берется по модулю 32, таким образом смещение битов находится в диапазоне от 0 до 31. Это позволяет выбирать любой бит внутри регистра. Для битовых строк в памяти это поле непосредственного значения дает только смещение внутри слова или двойного слова.

Некоторые ассемблеры поддерживают значения битовых смещений больше 31, используя поле непосредственного значения (imm8) в комбинации с полем смещения операнда в памяти(disp). В этом случае ассемблером младшие 3 или 5 битов (3 для 16 битных операндов, 5 для 32 битных операндов) смещения бита (второй операнд команды) сохраняются в поле непосредственного операнда, а старшие биты сдвигаются и комбинируются с полем смещения (disp). Процессор же игнорирует ненулевые значения старших битов поля imm8.

При доступе к памяти процессор обращается к четырем байтам, начинающимся по полученному следующим образом адресу:

Effective Address + (4 * (BitOffset DIV 32))

для размера операнда 32 бита, или к двум байтам, начинающимся по адресу:

Effective Address + (2 * (BitOffset DIV 16))

для 16 битного размера операнда. Такое обращение происходит, даже если необходим доступ только к одиночному байту. Поэтому, избегайте ссылок к областям памяти, близким к пустым адресным пространствам. В частности, избегайте ссылок на распределенные в памяти регистры ввода-вывода. Вместо этого используйте команду MOV для загрузки и сохранения значений по таким адресам и регистровую форму команды BTC для работы с данными.

BTR Проверка и сброс бита

O	D	I	T	S	Z	A	P	C
								*

Код операции	Команда	Такты			Пример
		Pentium	486	386	
0F B3	BTR r/m16,r16	7/13	6/13	6/13	btr [di],ax
0F B3	BTR r/m32,r32	7/13	6/13	6/13	btr [edi],eax
0F BA /6 ib	BTR r/m16,imm8	7/8	6/8	6/8	btr [bx+8],05
0F BA /6 ib	BTR r/m32,imm8	7/8	6/8	6/8	btr [ebx],02

Операция

```
CF = BIT[LeftSRC, RightSRC];
BIT[LeftSRC, RightSRC] = 0;
```

Описание

Команда BTR сохраняет значение бита, из первого операнда со смещением, указанным вторым операндом, во флаге CF, а затем обнуляет этот бит.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Примечание

Значение индекса выбираемого бита может быть представлено непосредственным значением в команде BTR или значением в общем регистре. В этой команде используется только 8 битное непосредственное значение. Значение этого операнда берется по модулю 32, таким образом смещение битов находится в диапазоне от 0 до 31. Это позволяет выбирать любой бит внутри регистра. Для битовых строк в памяти это поле непосредственного значения дает только смещение внутри слова или двойного слова.

Некоторые ассемблеры поддерживают значения битовых смещений больше 31, используя поле непосредственного значения (imm8) в комбинации с полем смещения операнда в памяти(disp). В этом случае ассемблером младшие 3 или 5 битов (3 для 16 битных операндов, 5 для 32 битных операндов) смещения бита (второй операнд команды) сохраняются в поле непосредственного операнда, а старшие биты сдвигаются и комбинируются с полем смещения (disp). Процессор же игнорирует ненулевые значения старших битов поля imm8.

При доступе к памяти процессор обращается к четырем байтам, начинающимся по полученному следующим образом адресу:

Effective Address + (4 * (BitOffset DIV 32))

для размера операнда 32 бита, или к двум байтам, начинающимся по адресу:

Effective Address + (2 * (BitOffset DIV 16))

для 16 битного размера операнда. Такое обращение происходит, даже если необходим доступ только к одиночному байту. Поэтому, избегайте ссылок к областям памяти, близким к пустым адресным пространствам. В частности, избегайте ссылок на распределенные в памяти регистры ввода-вывода. Вместо этого используйте команду MOV для загрузки и сохранения значений по таким адресам и регистровую форму команды BTR для работы с данными.

BTS Проверка и установка бита

O	D	I	T	S	Z	A	P	C
								*

Код операции	Команда	Такты			Пример
		Pentium	486	386	
0F AB	BTS r/m16,r16	7/13	6/13	6/13	bts [di],bp
0F AB	BTS r/m32,r32	7/13	6/13	6/13	bts [edi],ebx

```
0F BA /5 ib   BTS r/m16,imm8  7/8      6/8      6/8          bts [bx+si],05
0F BA /5 ib   BTS r/m32,imm8  7/8      6/8      6/8          bts [edi+ebx],02
```

Операция

```
CF = BIT[LeftSRC, RightSRC];
BIT[LeftSRC, RightSRC] = 1;
```

Описание

Команда BTS сохраняет значение бита, из первого операнда со смещением, указанным вторым операндом, во флаге CF, а затем устанавливает этот бит в 1.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Примечание

Значение индекса выбираемого бита может быть представлено постоянным непосредственным значением в команде BTS или значением в общем регистре. В этой команде используется только 8 битное непосредственное значение. Значение этого операнда берется по модулю 32, таким образом смещение битов находится в диапазоне от 0 до 31. Это позволяет выбирать любой бит внутри регистра. Для битовых строк в памяти это поле непосредственного значения дает только смещение внутри слова или двойного слова.

Некоторые ассемблеры поддерживают значения битовых смещений больше 31, используя поле непосредственного значения (imm8) в комбинации с полем смещения операнда в памяти(disp). В этом случае ассемблером младшие 3 или 5 битов (3 для 16 битных операндов, 5 для 32 битных операндов) смещения бита (второй операнд команды) сохраняются в поле непосредственного операнда, а старшие биты сдвигаются и комбинируются с полем смещения (disp). Процессор же игнорирует ненулевые значения старших битов поля imm8.

При доступе к памяти процессор обращается к четырем байтам, начинающимся по полученному следующим образом адресу:

Effective Address + (4 * (BitOffset DIV 32))

для размера операнда 32 бита, или к двум байтам, начинающимся по адресу:

Effective Address + (2 * (BitOffset DIV 16))

для 16 битного размера операнда. Такое обращение происходит, даже если необходим доступ только к одиночному байту. Поэтому, избегайте ссылок к областям памяти, близким к пустым адресным пространствам. В частности, избегайте ссылок на распределенные в памяти регистры ввода-вывода. Вместо этого используйте команду MOV для загрузки и сохранения значений по таким адресам и регистровую форму команды BTS для работы с данными.

CALL Вызов процедуры

O D I T S Z A P C

Код	Команда	Такты					Описание типа вызова
		Pentium	486	386	286*	86	
E8 cw	CALL rel16	1	3	7+m	7	19	ближний, смещение указывается относительно следующей команды
FF /2	CALL r/m16	2	5/5	7+m/ 10+m	7/11	16/ 21+EA	ближний косвенный
9A cd	CALL ptr16:16	4	18, pm=20	17+m, pm=34+m	13, pm=26	28	межсегментный, указывается 4 байтный непосредственный адрес
9A cd	CALL ptr16:16	pm=22	pm=35	pm=52+m	41		через шлюз вызова, привилегии не изменяются
9A cd	CALL ptr16:16	pm=44	pm=69	pm=86+m	82		через шлюз вызова, привилегии увеличиваются, нет параметров
9A cd	CALL ptr16:16	pm=45+2x	pm=77 +4x	pm=94+ 4x+m	86+4x		через шлюз вызова, привилегии увеличиваются, x параметров
9A cd	CALL ptr16:16	pm=21+ts	pm=37 +ts	ts	177/ 182		к новой задаче
FF /3	CALL m16:16	5	17, pm=20	22+m, pm=38+m	16/29	37+EA	косвенный межсегментный
FF /3	CALL m16:16	pm=22	pm=35	pm=56+m	44		через шлюз вызова, привилегии не изменяются
FF /3	CALL m16:16	pm=44	pm=69	pm=90+m	83		через шлюз вызова, привилегии увеличиваются, нет параметров
FF /3	CALL m16:16	pm=45+2x	pm=77 +4x	pm=98+ 4x+m	90+ 4x+m		через шлюз вызова, привилегии увеличиваются, x параметров
FF /3	CALL m16:16	pm=21+ts	pm=37 +ts	5+ts	180/ 185		к новой задаче
E8 cd	CALL rel32	1	3	7+m			ближний, смещение указывается относительно следующей команды
FF /2	CALL r/m32	2	5/5	7+m/ 10+m			ближний косвенный
9A cp	CALL ptr16:32	4	18, pm=20	17+m, pm=34+m			межсегментный, указывается 6 байтный непосредственный адрес
9A cp	CALL ptr16:32	pm=22	pm=35	pm=52+m			через шлюз вызова, привилегии не изменяются
9A cp	CALL ptr16:32	pm=44	pm=69	pm=86+m			через шлюз вызова, привилегии увеличиваются, нет параметров
9A cp	CALL ptr16:32	pm=45+2x	pm=77 +4x	pm=94+ 4x+m			через шлюз вызова, привилегии увеличиваются, x параметров
9A cp	CALL ptr16:32	pm=21+ts	pm=37 +ts	ts			к новой задаче
FF /3	CALL m16:32	5	17, pm=20	22+m, pm=38+m			косвенный межсегментный
FF /3	CALL m16:32	pm=22	pm=35	pm=56+m			через шлюз вызова, привилегии не изменяются
FF /3	CALL m16:32	pm=44	pm=69	pm=90+m			через шлюз вызова, привилегии увеличиваются, нет параметров
FF /3	CALL m16:32	pm=45+2x	pm=77 +4x	pm=98+ 4x+m			через шлюз вызова, привилегии увеличиваются, x параметров
FF /3	CALL m16:32	pm=21+ts	pm=37 +ts	5+ts			к новой задаче

* При выполнении следующей команды добавляется по одному такту на каждый байт команды (только для 80286).

Значения ts задаются следующей таблицей:

Старая задача	Новая задача		
	для 32 битного TSS	для 16 битного TSS	для VM TSS
VM/32 битный/16 битный	85	87	71

Операция

```
IF (вызов с операндами типа rel16 или rel32)
THEN (* ближний относительный вызов *)
  IF OperandSize = 16
  THEN
    Push(IP);
    EIP = (EIP + rel16) AND 0000FFFFh;
  ELSE (* OperandSize = 32 *)
    Push(EIP);
    EIP = EIP + rel32;
  FI;
FI;
IF (вызов с операндами типа r/m16 или r/m32)
THEN (* ближний абсолютный вызов *)
  IF OperandSize = 16
  THEN
    Push(IP);
    EIP = [r/m16] AND 0000FFFFh;
  ELSE (* OperandSize = 32 *)
    Push(EIP);
    EIP = [r/m32];
  FI;
FI;
IF (PE = 0 OR (PE = 1 AND VM = 1)) (* реальный или виртуальный-8086 режим
      *)
  AND (команда типа far CALL)
    (* т.е. тип операнда m16:16, m16:32, ptr16:16 или ptr16:32 *)
THEN
  IF OperandSize = 16
  THEN
    Push(CS);
    Push(IP); (* адрес следующей инструкции; 16 бит *)
  ELSE (* OperandSize = 32 *)
    Push(CS); (* 32 бита *)
    Push(EIP); (* адрес следующей инструкции; 32 бита  *)
  FI;
  IF (операнды типа m16:16 или m16:32)
  THEN (* косвенный дальний вызов *)
    IF OperandSize = 16
    THEN
      CS:IP = [m16:16];
      EIP = EIP AND 0000FFFFh; (* очищает старшие 16 бит *)
    ELSE (* OperandSize = 32 *)
      CS:EIP = [m16:32];
    FI;
  FI;
  IF (операнды типа ptr16:16 или ptr16:32)
  THEN (* прямой дальний вызов *)
    IF OperandSize = 16
    THEN
      CS:IP = ptr16:16;
      EIP = EIP AND 0000FFFFh; (* очищает старшие 16 бит  *)
    ELSE (* OperandSize = 32 *)
      CS:EIP = ptr16:32;
    FI;
  FI;
FI;
IF (PE = 1 AND VM = 0) (* Защищенный режим, не V86 режим *)
  AND (команда типа far CALL)
THEN
  Если вызов косвенный, проверяется доступность двойного слова по
       эффективному адресу, #GP(0), если нарушен предел;
  Новый CS селектор должен быть не нулевым, иначе #GP(0);
```

```
    Проверяет, чтобы индекс нового CS селектора попадал в пределы таблицы
        дескрипторов, иначе #GP(новый CS селектор);
    Проверяет AR байт выбранного дескриптора для различных возможных
        значений, в зависимости от значения:
      GO TO CONFORMING-CODE-SEGMENT; (* согласованный кодовый
        сегмент*)
      GO TO NONCONFORMING-CODE-SEGMENT; (* несогласованный кодовый
        сегмент *)
      GO TO CALL-GATE; (* шлюз вызова *)
      GO TO TASK-GATE; (* шлюз задачи *)
      GO TO TASK-STATE-SEGMENT; (* TSS - сегмент состояния задачи *)
    ELSE #GP(CS селектор);
FI;

CONFORMING-CODE-SEGMENT:
    IF DPL > CPL THEN #GP(CS селектор); FI;
    Сегмент должен присутствовать, иначе #NP(CS селектор);
    Стек должен иметь достаточно места для адреса возврата, иначе #SS(0);
    Указатель инструкции должен быть в пределах кодового сегмента,
        иначе #GP(0);
    Загрузить дескриптор кодового сегмента в регистр CS;
    Загрузить CS с новым селектором кодового сегмента;
    Загрузить EIP с расширением нулями(новое смещение);
    IF OperandSize = 16 THEN EIP = EIP AND 0000FFFFh; FI;

NONCONFORMING-CODE-SEGMENT:
    IF RPL > CPL THEN #GP(CS селектор); FI;
    IF DPL ≠ CPL THEN #GP(CS селектор);
    Сегмент должен присутствовать, иначе #NP(CS селектор);
    Стек должен иметь достаточно места для адреса возврата, иначе #SS(0);
    Указатель инструкции должен быть в пределах кодового сегмента,
        иначе #GP(0);
    Загрузить дескриптор кодового сегмента в регистр CS;
    Загрузить CS с новым селектором кодового сегмента;
    Установить RPL в CS в значение CPL;
    Загрузить EIP с расширением нулями(новое смещение);
    IF OperandSize = 16 THEN EIP = EIP AND 0000FFFFh; FI;

CALL-GATE:
    DPL шлюза вызова должен быть ≥ CPL, иначе #GP(селектор шлюза вызова);
    DPL шлюза вызова должен быть ≥ RPL, иначе #GP(селектор шлюза вызова);
    Шлюз вызова должен присутствовать, иначе #NP(селектор шлюза вызова);
    Проверка селектора кодового сегмента в дескрипторе шлюза вызова:
      Селектор должен быть не нулевым, иначе #GP(0);
      Индекс селектора должен попадать в пределы таблицы дескрипторов,
        иначе #GP(CS селектор);
      AR байт выбранного дескриптора задает кодовый сегмент,
          иначе #GP(CS селектор);
      DPL выбранного дескриптора должен быть ≤ CPL, иначе #GP(CS селектор);
      IF (несогласованный кодовый сегмент) AND (DPL < CPL)
      THEN GO TO MORE-PRIVILEGE ELSE GO TO SAME-PRIVILEGE FI;

MORE-PRIVILEGE:
    Получить новый SS селектор для нового уровня привилегий из TSS;
    Проверка селектора и дескриптора для нового SS:
      Селектор должен быть не нулевым, иначе #TS(0);
      Индекс селектора должен попадать в пределы таблицы дескрипторов,
        иначе #TS(SS селектор);
      RPL селектора должен быть равен DPL кодового сегмента,
        иначе #TS(SS селектор);
      DPL стекового сегмента должен быть равен DPL кодового сегмента,
        иначе #TS(SS селектор);
      AR байт дескриптора должен задавать разрешенный для записи сегмент
        данных, иначе #TS(SS селектор);
      Сегмент должен присутствовать, иначе #SS(SS селектор);
```

```
  IF OperandSize = 32;
  THEN
    Новый стек должен иметь место для параметров плюс 16 байт,
      иначе #SS(SS селектор);
    EIP должен быть в пределах кодового сегмента, иначе #GP(0);
    Загрузить новое значение SS:eSP из TSS;
    Загрузить новое значение CS:EIP из шлюза;
  ELSE
    Новый стек должен иметь место для параметров плюс 8 байт,
      иначе #SS(SS селектор);
    IP должен быть в пределах кодового сегмента, иначе #GP(0);
    Загрузить новое значение SS:eSP из TSS;
    Загрузить новое значение CS:IP из шлюза;
  FI;
  Загрузить дескриптор CS;
  Загрузить дескриптор SS;
  Поместить длинный указатель старого стека в новый стек;
  Получить счетчик двойных слов из шлюза вызова, замаскировав 5 бит;
  Скопировать параметры из старого стека в новый стек;
  Поместить адрес возврата в новый стек;
  Установить CPL в значение DPL сегмента стека;
  Установить RPL в CS в значение CPL;

SAME-PRIVILEGE:
  IF OperandSize = 32
  THEN
    Стек должен иметь место для 6 байт адреса возврата (расширяемых до 8
      байт), иначе #SS(0);
    EIP должен быть в пределах кодового сегмента, иначе #GP(0);
    Загрузить новое значение CS:EIP из шлюза;
  ELSE
    Стек должен иметь место для 4 байт адреса возврата, иначе #SS(0);
    IP должен быть в пределах кодового сегмента, иначе #GP(0);
    Загрузить новое значение CS:IP из шлюза;
  FI;
  Поместить адрес возврата в стек;
  Загрузить дескриптор кодового сегмента в регистр CS;
  Установить RPL в CS в значение CPL;

TASK-GATE:
  DPL шлюза задачи должен быть ≥ CPL, иначе #TS(селектор шлюза);
  DPL шлюза задачи должен быть ≥ RPL, иначе #TS(селектор шлюза);
  Шлюз задачи должен присутствовать, иначе #NP(селектор шлюза);
  Проверка селектора TSS в дескрипторе шлюза задачи:
    Селектор должен задавать GDT (бит TI=0), иначе #TS(TSS селектор);
    Индекс селектора должен попадать в пределы таблицы GDT,
      иначе #TS(TSS селектор);
    AR байт TSS дескриптора должен задавать свободный TSS,
      иначе #TS(TSS селектор);
    TSS должен присутствовать, иначе #NP(TSS селектор);
  SWITCH-TASKS (с вложением) в TSS;
  IP должен быть в пределах кодового сегмента, иначе #TS(0);

TASK-STATE-SEGMENT:
  TSS DPL должен быть ≥ CPL, иначе #TS(TSS селектор);
  TSS DPL должен быть ≥ RPL, иначе #TS(TSS селектор);
  AR байт TSS дескриптора должен задавать свободный TSS,
    иначе #TS(TSS селектор);
  TSS должен присутствовать, иначе #NP(TSS селектор);
  SWITCH-TASKS (с вложением) в TSS;
  IP должен быть в пределах кодового сегмента, иначе #TS(0);
```

Описание

Команда CALL передает управление указанной операндом процедуре. После завершения процедуры (команда возврата выполняется внутри самой процедуры), выполнение продолжается с команды, следующей за командой CALL.

Действие различных форм команды описано ниже.

При ближних вызовах (вызовы с операндами типа r/m16, r/m32, rel16, rel32), изменение или сохранение сегментных регистров не обязательно. Формы команды CALL rel16 и CALL rel32 прибавляют знаковое смещение к адресу следующей за CALL команды, чтобы определить адрес перехода на вызываемую процедуру. Форма CALL rel16 используется с атрибутом размера операнда 16 бит, CALL rel32 - с атрибутом размера операнда 32 бита. Вычисленный таким образом адрес помещается в 32 битном регистре EIP. В случае CALL rel16 верхние 16 бит регистра EIP очищаются, приводя результат к значению, невыходящему за пределы 16 бит. Команды CALL r/m16 и CALL r/m32 задают регистр или местоположение ячейки памяти, из которых выбирается абсолютное смещение в сегменте. Смещение, выбранное из r/m, является 32 битным в случае 32 битного атрибута размера операнда (r/m32), или 16 битным для 16 битного атрибута размера операнда (r/m16). Смещение команды, следующей за командой CALL, помещается в стек. Оно будет использовано ближней командой RET для возврата из процедуры. Ни одна из этих форм команды CALL регистр CS не изменяет.

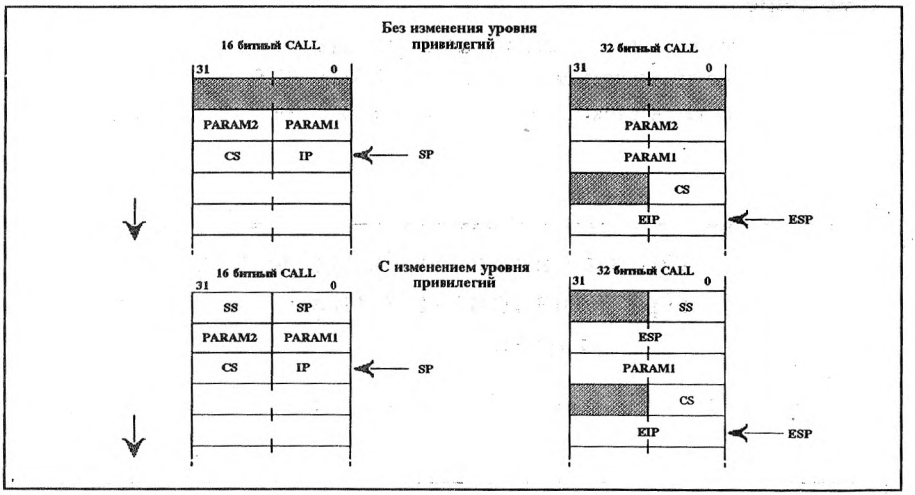
Дальние вызовы, (CALL ptr16:16 и CALL ptr16:32) используют четырех и шестибайтные операнды в качестве дальних указателей на вызываемую процедуру. Формы команды CALL m16:16 и CALL m16:32 выбирают дальний указатель из указанного местоположения в памяти (разрешены все режимы адресации). В режиме реальной адресации или в режиме V86 дальний указатель обеспечивает 16 бит для регистра CS и 16 или 32 бита для регистра EIP (в зависимости от атрибута размера операнда). Эти формы команды в качестве адреса возврата помещают в стек значения регистров CS и (E)IP.

В защищенном режиме обе формы с дальним указателем принимают во внимание значение байта AR в дескрипторе, указанном селекторной частью дальнего указателя. В зависимости от значения байта AR, происходит одно из следующих управляющих преобразований:

- дальний вызов на том же уровне привилегий;
- межуровневый дальний вызов;
- переключение задачи.

Если вызов осуществляется без переключения задачи, то флаги не изменяются. Переключение задачи вызывает перезагрузку регистра (E)FLAGS значением из TSS.

Стек после дальнего 16-ти или 32-х разрядного CALL в защищенном режиме



Особые ситуации защищенного режима

Для дальних вызовов: #GP, #NP, #SS и #TS, как указано в пункте "Операция".

Для ближних прямых вызовов: #GP(0), если местоположение процедуры не попадает в пределы кодового сегмента; #SS(0), если сохранение адреса возврата вызывает нарушение границ стекового сегмента; #PF(код ошибки), страничная ошибка.

Для ближних косвенных вызовов: #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #GP(0), если местоположение процедуры не попадает в пределы кодового сегмента; #PF(код ошибки), страничная ошибка.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка.

Замечание

Любые вызовы из 32 битного кодового сегмента к 16 битным сегментам должны осуществляться из первых 64Кбайт 32 битного сегмента, так как в этом случае в стеке сохраняется только 16 битное смещение адреса возврата.

CBW Преобразовать байт в слово
CWDE Преобразовать слово в двойное слово

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
98	CBW	3	3	3	2	2	cbw
98	CWDE	3	3	3			cwde

Операция

```
IF OperandSize = 16 (* команда типа CBW *)
THEN AX = SignExtend(AL);
ELSE (* OperandSize = 32, команда типа CWDE *)
  EAX = SignExtend(AX);
FI;
```

Описание

Команда CBW преобразует знаковый байт из регистра AL в знаковое слово в регистре AX путем расширения наиболее значащего бита регистра AL (знаковый бит) во все биты регистра AH. Команда CWDE преобразует знаковое слово в регистре AX в знаковое двойное слово в регистре EAX путем расширения наиболее значащего бита регистра AX (знаковый бит) в два наиболее значащих байта регистра EAX. Заметьте, что команда CWDE отличается от команды CWD тем, что CWD использует регистровую пару DX:AX, а не EAX в качестве назначения.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

CDQ Преобразовать двойное слово в учетверенное слово

Смотрите команду CWD.

CLC Очистить флаг переноса

O	D	I	T	S	Z	A	P	C
								0

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
F8	CLC	2	2	2	2	2	clc

Операция

CF = 0;

Описание

Команда CLC очищает флаг CF. Никакого другого влияния на флаги и регистры не имеет.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

CLD Очистить флаг направления

O	D	I	T	S	Z	A	P	C
	0							

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
FC	CLD	2	2	2	2	2	cld

Операция

DF = 0;

Описание

Команда CLD очищает флаг направления. Другие флаги и регистры не затрагиваются.

После выполнения команды CLD строковые операции будут увеличивать используемый ими индекс (SI и/или DI).

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

CLI Запретить внешние прерывания

O D I T S Z A P C
0

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
FA	CLI	7	5	3	3	2	cli

Операция

```
IF PE = 0;
THEN IF = 0;
ELSE
  IF VM = 0; (* Выполняется в защищенном режиме *)
  THEN
    IF IOPL = 3
    THEN IF = 0;
    ELSE IF CPL ≤ IOPL
        THEN IF = 0;
        ELSE #GP(0);
        FI;
    FI;
  ELSE (* Выполняется в виртуальном-8086 режиме *)
    IF IOPL = 3
    THEN IF = 0;
    ELSE #GP(0);
    FI;
  FI;
FI;
```

Таблица решений

Данная таблица решений указывает какое действие из нижней части таблицы произойдет при условиях, стоящих в верхней части.

PE =	0	1	1	1	1
VM =	-	0	-	0	1
CPL	-	≤ IOPL	-	> IOPL	-
IOPL	-	-	=3	-	< 3
IF ← 0	Y	Y	Y		
#GP(0)				Y	Y

Замечание:

	не влияет;
пусто	действие не происходит;
Y	произошло действие из первой колонки.

Описание

Команда CLI очищает флаг IF, если текущий уровень привилегий как минимум такой же, как и IOPL. Никакие другие флаги не изменяются. После команды CLI и до точки установки флага IF внешние прерывания не распознаются.

Особые ситуации защищенного режима

#GP(0), если текущий уровень привилегий по значению больше (имеет меньше привилегий), чем уровень привилегий ввода-вывода в регистре флагов. Уровень привилегий ввода-вывода определяет наихудший уровень привилегий, на котором возможен ввод-вывод.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

#GP(0), как и в защищенном режиме.

Замечание

Информацию об особых ситуациях виртуального режима для данной команды смотрите в Приложении H.

CLTS Очистить флаг переключения задач в CR0

```
O   D   I   T   S   Z   A   P   C

TS = 0 (* TS флаг в регистре CR0 *)
```

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
0F 06	CLTS	10	7	5	2	clts

Операция

```
TS Flag in CR0 = 0;
```

Описание

Команда CLTS очищает флаг переключения задач (TS) в регистре CR0. Этот флаг устанавливается процессором каждый раз, когда происходит переключение задач. Флаг TS используется для управления сопроцессором следующим образом:

- отлавливается каждое выполнение команды ESC, если установлен флаг TS;
- отлавливается команда WAIT, если установлен флаг MP и TS.

Таким образом, если произошло переключение задач после того, как началась команда ESC, возможно необходимо сохранить содержимое устройства с плавающей запятой перед тем, как будет извлечена новая команда ESC. Обработчик ошибок сохраняет содержимое FPU и очищает флаг TS.

Команда CLTS используется в системном программном обеспечении, а не в прикладных программах. Это привилегированная команда, и может выполняться при нулевом уровне привилегий.

Особые ситуации защищенного режима

#GP(0), если команда CLTS выполняется с отличным от нуля текущим уровнем привилегий.

Особые ситуации режима реальной адресации

Нет (допустима в режиме реальной адресации, чтобы позволить инициализировать защищенный режим).

Особые ситуации режима V86

Такие же, как и в защищенном режиме.

CMC Дополнить флаг переноса

O	D	I	T	S	Z	A	P	C
								*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
F5	CMC	2	2	2	2	2	cmc

Операция

CF = NOT CF;

Описание

Команда CMC инвертирует значение флага CF. Другие флаги не изменяются.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

CMP Сравнить два операнда

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
3C ib	CMP AL,imm8	1	1	2	3	4	cmp al,17
3D iw	CMP AX,imm16	1	1	2	3	4	cmp ax,0AABh
3D id	CMP EAX,imm32	1	1	2			cmp eax,0AAC23h
80 /7 ib	CMP r/m8,imm8	1/2	1/2	2/5	3/6	4/10+EA	cmp bl,5
81 /7 iw	CMP r/m16,imm16	1/2	1/2	2/5	3/6	4/10+EA	cmp bx,0A4Fh
81 /7 id	CMP r/m32,imm32	1/2	1/2	2/5			cmp edx,0CAAAAh
83 /7 ib	CMP r/m16,imm8	1/2	1/2	2/5	3/6	4/10+EA	cmp word ptr [di],7
83 /7 ib	CMP r/m32,imm8	1/2	1/2	2/5			cmp ecx,0Fh
38 /r	CMP r/m8,r8	1/2	1/2	2/5	2/7	3/9+EA	cmp ch,al
39 /r	CMP r/m16,r16	1/2	1/2	2/5	2/7	3/9+EA	cmp word ptr m32,dx
39 /r	CMP r/m32,r32	1/2	1/2	2/5			cmp edx,ebx
3A /r	CMP r8,r/m8	1/2	1/2	2/6	2/6	3/9+EA	cmp ch,sum
3B /r	CMP r16,r/m16	1/2	1/2	2/6	2/6	3/9+EA	cmp si,ax
3B /r	CMP r32,r/m32	1/2	1/2	2/6			cmp edi,raznost

Операция

LeftSRC - SignExtend(RightSRC);
(* CMP не сохраняет результат; цель команды - установление флагов *)

Описание

Команда CMP вычитает второй операнд из первого, но, в отличие от команды SUB, не сохраняет результат - изменяются только флаги. Команда CMP обычно используется вместе с условными переходами и командой SETcc. Если более чем однобайтный операнд сравнивается с однобайтным непосредственным значением, однобайтное значение сначала знакорасширяется.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

CMPS Сравнить строковые операнды
CMPSB
CMPSW
CMPSD

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
A6	CMPS m8,m8	5	8	10	8	22	cmpsb
A7	CMPS m16,m16	5	8	10	8	22	cmpsw
A7	CMPS m32,m32	5	8	10			cmpsd
A6	CMPSB	5	8	10	8	22	cmpsb

A7	CMPSW	5	8	10	8	22	cmpsw
A7	CMPSD	5	8	10			cmpsd

Операция

```
IF (команда типа CMPSD) OR (команда с двойным словом в качестве операнда)
THEN OperandSize = 32;
ELSE OperandSize = 16;
FI;
IF AddressSize = 16
THEN (использовать SI для source-index и DI для destination-index);
ELSE (* AddressSize = 32 *)
   использовать ESI для source-index и EDI для destination-index;
FI;
IF (команда с байтом в качестве операнда)
THEN
  установить ZF по результатам операции [destination-index] - [source-index];
     (* сравнение байтов *)
  IF DF = 0 THEN IncDec = 1; ELSE IncDec = -1; FI;
ELSE
  IF OperandSize = 16
  THEN
     установить ZF по результатам операции [destination-index] - [source-index];
        (* сравнение слов *)
    IF DF = 0 THEN IncDec = 2 ELSE IncDec = -2; FI;
  ELSE (* OperandSize = 32 *)
    установить ZF по результатам операции [destination-index] - [source-index];
        (* сравнение двойных слов *)
    IF DF = 0 THEN IncDec = 4 ELSE IncDec = -4; FI;
  FI;
FI;
source-index = source-index + IncDec;
destination-index = destination-index + IncDec;
```

Описание

Команда CMPS сравнивает байт, слово или двойное слово, указанное индексным регистром источником, с байтом, словом или двойным словом, указанным индексным регистром назначения. Если атрибут размера адреса этой команды равен 16 битам, то в качестве индексных регистров источника и назначения будут использованы SI и DI. Перед выполнением команды CMPS загрузите в регистры SI и DI (ESI и EDI) правильные индексные значения.

Сравнение выполняется путем вычитания операнда, указанного индексным регистром назначения из операнда, указанного индексным регистром источника.

Заметьте, что направление вычитания для команды CMPS это [SI] - [DI] или [ESI] - [EDI]. Левый операнд ([SI] или [ESI]) - это источник, правый операнд ([DI] или [EDI]) - это назначение. Это случай инверсного использования соглашения Intel о том, что левый операнд - это назначение, а правый операнд - это источник.

Результат вычитания не сохраняется, изменения отражаются только на флагах. Типы операндов определяются тем, сравниваются ли байты, слова или двойные слова. Для первого операнда ([SI] или [ESI]) используется регистр DS, если не присутствует префикс замены сегмента. Второй операнд ([DI] или [EDI]) адресуется регистром ES, замещение сегмента невозможно.

После сравнения оба индексных регистра (источник и назначение) автоматически изменяются. Если флаг DF равен 0 (была выполнена команда CLD), то регистры увеличиваются; если флаг DF равен 1 (была выполнена команда STD), то регистры уменьшаются. Регистры увеличиваются или уменьшаются на 1 при сравнении байтов, на 2 при сравнении слов, и на 4 при сравнении двойных слов.

Команды CMPSB, CMPSW, CMPSD являются синонимами команды CMPS для сравнения байтов, слов, или двойных слов, соответственно.

Команде CMPS может предшествовать префикс REPE или REPNE для сравнения блока из (E)CX байтов, слов или двойных слов. Обратитесь к описанию команды REP для более подробной информации по этой операции.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

CMPXCHG Сравнить и обменять

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Код операции	Команда	Такты Pentium	Такты 486		Пример
0F B0 /r	CMPXCHG r/m8,r8	6			cmpxchg al,dh
0F B1 /r	CMPXCHG r/m16,r16	6			cmpxchg ax,dx
0F B1 /r	CMPXCHG r/m32,r32	6			cmpxchg ecx,edi
0F A6 /r	CMPXCHG r/m8,r8		6/7 если сравнение удачно;6/10 если нет		cmpxchg al,dh
0F A7 /r	CMPXCHG r/m16,r16		6/7 если сравнение удачно;6/10 если нет		cmpxchg ax,dx
0F A7 /r	CMPXCHG r/m32,r32		6/7 если сравнение удачно;6/10 если нет		cmpxchg ecx,edi

Операция

```
IF (аккумулятор = DEST)
THEN
  ZF = 1;
  DEST = SRC;
ELSE
  ZF = 0;
  аккумулятор = DEST;
FI;
```

Описание

Команда CMPXHG сравнивает аккумулятор (регистр AL, AX или EAX) с DEST. В случае их равенства SRC загружается в DEST, иначе DEST загружается в аккумулятор.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Примечание

Эта команда может быть использована с префиксом LOCK. В порядке упрощения интерфейса с шиной процессора операнд назначения принимает цикл записи, не обращая внимания на результат выполнения. DEST записывается обратно, если сравнение неудачно, и SRC записывается в DEST в противном случае. (Процессор никогда не производит блокировки чтения без блокировки записи.)

CMPXCHG8B Сравнить и обменять 8 байт

O	D	I	T	S	Z	A	P	C
					*			

Код операции	Команда	Такты Pentium	Пример
0F C7 m64	CMPXCHG8B r/m64	10	cmpxchg8b [edi]

Операция

```
IF EDX:EAX = DEST
THEN
  ZF = 1;
  DEST = ECX:EBX;
ELSE
  ZF = 0;
  EDX:EAX = DEST;
FI;
```

Описание

Команда CMPXCHG8B сравнивает 64 битное значение EDX:EAX с операндом в памяти (DEST). EDX содержит старшие 32 бита, EAX содержит младшие 32 бита 64 битного значения. Если они равны, то 64 битное значение в ECX:EBX сохраняется в DEST. ECX содержит старшие 32 бита, а EBX младшие 32 бита. В противном случае DEST загружается в EDX:EAX.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Примечание

Эта команда может быть использована с префиксом LOCK. В порядке упрощения интерфейса с шиной процессора операнд назначения принимает цикл записи, не обращая внимания на результат выполнения. DEST записывается обратно, если сравнение неудачно, и SRC записывается в DEST в противном случае. (Процессор никогда не производит блокировки чтения без блокировки записи.)

Синтаксис "r/m64" ранее был использован только в контексте операций с плавающей запятой. Он указывает на 64 битное значение в памяти по адресу определенному байтом ModR/M.

CPUID Идентификация CPU

O D I T S Z A P C

Код операции	Команда	Такты Pentium	Пример
0F A2	CPUID	14	cpuid

Операция

```
switch (EAX)
  case 0:
    EAX = hv; (* hv = 1 для процессора Pentium *)
            (* hv - это максимально возможное входное значение, которое
               распознается командой CPUID *)
    EBX = строка идентификации продавца;
    EDX = строка идентификации продавца;
    ECX = строка идентификации продавца;
    break;
  case 1:
    EAX[3:0] = Номер поколения;
    EAX[7:4] = Модель;
    EAX[11:8] = Семейство;
    EAX[31:12] = Зарезервировано;
    EBX = зарезервирован; (* 0 *)
    ECX = зарезервирован; (* 0 *)
    EDX = флаг особенностей;
    break;
  default: (* EAX > hv *)
    EAX = зарезервирован; (* неопределен *)
    EBX = зарезервирован; (* неопределен *)
    ECX = зарезервирован; (* неопределен *)
    EDX = зарезервирован; (* неопределен *)
    break;
end-of-switch
```

Описание

Команда CPUID предоставляет программному обеспечению информацию о продавце, семействе, модели и поколении процессора, на котором она выполнена. Входное значение, загружаемое в регистр EAX, указывает какую информацию необходимо вернуть командой CPUID.

После выполнения команды CPUID со значением 0 в EAX, регистр EAX будет содержать максимальное значение, понимаемое командой CPUID. Для процессора Pentium значение в EAX будет 1. В выходное значение этой команды также включена (со входным значением 0 в EAX) строка идентификации продавца, содержащаяся в EBX, EDX и ECX регистрах. EBX содержит первых 4 символа, EDX содержит следующие 4 сим-

вола и ECX содержит последние 4 символа. Для процессоров Intel строка идентификации продавца следующая: "GenuineIntel".

После выполнения команды CPUID с входным значением 1, загруженным в регистр EAX, EAX[3:0] содержит номер разработки микропроцессора, EAX[7:4] содержит модель (первая модель будет указана 0001b в этих битах), и EAX[11:8] содержит семейство (5 для семейства процессоров Pentium). EAX[31:12], EBX и ECX зарезервированы. Процессор Pentium устанавливает регистр особенностей EDX в 1BFh, указывая какие свойства он поддерживает. Установленный флаг свойств указывает на соответствующее поддерживаемое свойство. Регистр свойств определен следующим образом:

EDX[0:0] FPU на кристалле;
EDX[6:1] смотрите в Приложении H;
EDX[7:7] особая ситуация проверки машины;
EDX[8:8] команда CMPXCHG8B;
EDX[31:9] зарезервировано.

Программное обеспечение должно сначала определить идентификатор продавца, в порядке точной интерпретации битов регистра особенностей.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

CWD Преобразовать слово в двойное слово
CDQ Преобразовать двойное слово в учетверенное слово

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
99	CWD	2	3	2	2	5	cwd
99	CDQ	2	3	2			cdq

Операция

```
IF OperandSize = 16 (* команда типа CWD *)
THEN DX = SignExtend(AX);
ELSE (* OperandSize = 32, команда типа CDQ *)
  EDX = SignExtend(EAX);
FI;
```

Описание

Команды CWD и CDQ удваивают размер операнда источника. Команда CWD копирует знак (бит 15) слова из регистра AX в каждую битовую позицию регистра DX. Команда CDQ копирует знак (бит 31) двойного слова из регистра EAX в каждую битовую позицию регистра EDX. Команда CWD может использоваться для получения четырехбайтного делимого и слова перед целочисленным делением слова, а команда CDQ может быть использована для получения восьмибайтного делимого из двойного слова перед делением двойных слов. Команды CWD и CDQ - это различные мнемоники одного кода операции. Какая из них будет выполняться, определяется 16 или 32 битным сегментом и присутствием какого-нибудь префикса замещения размера операнда.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

CWDE Преобразовать слово в двойное слово

Смотрите команду CBW.

DAA Десятичное выравнивание AL после сложения

O	D	I	T	S	Z	A	P	C
?				*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
27	DAA	3	2	4	3	4	daa

Операция

```
IF (((AL AND 0Fh) > 09h) or EFLAGS.AF = 1)
THEN AL = AL + 06h;
FI;
IF (((AL AND 0Fh) > 90h) or EFLAGS.CF = 1)
THEN
  AL = AL + 60h;
  CF = 1;
FI;
```

Описание

Выполняйте команду DAA только после выполнения команды сложения (ADD, ADC), которая оставляет однобайтный результат в регистре AL. Операнды команды сложения должны состоять из двух упакованных BCD цифр. Команда DAA подгоняет регистр AL так, чтобы он содержал корректный упакованный результат из двух BCD цифр.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

DAS Десятичное выравнивание AL после вычитания

O	D	I	T	S	Z	A	P	C
?				*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
2F	DAS	3	2	4	3	4	das

Операция

```
tmpCF = 0;
tmpAL = AL;
IF (((tmpAL AND 0Fh) > 09h) or AF = 1)
THEN
  AF = 1;
  AL = AL - 06h
  tmpCF = (AL < 0) OR CF;
FI;
IF ((tmpAL > 99h) > 09h) or CF = 1)
THEN
  AL = AL - 60h
  tmpCF = 1;
FI;
CF = tmpCF;
```

Описание

Выполняйте команду DAS только после выполнения команды вычитания, которая оставляет однобайтный результат в регистре AL. Операнды команды вычитания должны состоять из двух упакованных BCD цифр. Команда DAS подгоняет регистр AL так, чтобы он содержал корректный упакованный результат из двух BCD цифр.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

DEC Декремент

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
FE /1	DEC r/m8	1/3	1/3	2/6	2/7	3/15+EA	dec al
FF /1	DEC r/m16	1/3	1/3	2/6	2/7	3/15+EA	dec word ptr [di]
FF /1	DEC r/m32	1/3	1/3	2/6			dec dword ptr [ebx]
48+rw	DEC r16	1	1	2	2	3	dec dx
48+rd	DEC r32	1	1	2			dec ecx

Операция

```
DEST = DEST - 1;
```

Описание

Команда DEC вычитает 1 из операнда. Команда DEC не изменяет флаг CF. Чтобы воздействовать на флаг CF, используйте команду SUB с непосредственным операндом равным 1.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

DIV Беззнаковое деление

O	D	I	T	S	Z	A	P	C
?				?	?	?	?	?

Код	Команда	Такты					Действие	Пример
		Pentium	486	386	286	86		
F6 /6	DIV AL,r/m8	17	16/16	14/17	?	?	AL=AX/(r/m8) AH=AX MOD (r/m8)	div cl
F7 /6	DIV AX,r/m16	25	24/24	22/25	?	162	AX=DX:AX/(r/m16) DX=DX:AX MOD (r/m16)	div bx
F7 /6	DIV EAX,r/m32	41	40/40	38/41			EAX=EDX:EAX/(r/m32) EDX=EDX:EAX MOD (r/m32)	div edi

Операция

```
temp = dividend / divizor;
IF (temp не годится в качестве частного)     (* dividend - делимое *)
THEN Прерывание 0;                           (* divizor - делитель *)
ELSE                                         (* quotient - частное *)
  quotient = temp;                           (* remainder - остаток *)
  remainder = dividend MOD (r/m);
FI;
```

Замечание

Деление без знака. Делитель представляется r/m операндом. Делимое, частное и остаток используют неявные регистры.

Описание

Команда DIV выполняет целочисленное деление без знака. Делимое задается неявно, только делитель задается как операнд. Остаток всегда меньше делителя. Тип делителя определяет, какие регистры будут использованы. Если размер делителя 8 бит, в качестве делимого используется AX. При размере делителя 16 бит, делимое находится в паре регистров DX:AX. Целая часть частного помещается в первом случае в AL, остаток - в AH, во втором случае, соответственно, в AX и DX. Если размер делителя - 32 бита, аналогичным образом используются EDX и EAX.

Размер	Делитель	Частное	Остаток	Делимое
byte	r/m8	AL	AH	AX
word	r/m16	AX	DX	DX:AX
dword	r/m32	EAX	EDX	EDX:EAX

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

ENTER Создать кадр стека для параметров процедуры

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
C8 iw 00	ENTER imm16,0	11	14	10	11	enter 4,0
C8 iw 01	ENTER imm16,1	15	17	12	11	enter 5,1
C8 iw ib	ENTER imm16,imm8	15+2*imm8	17+3*imm8	15+4*(imm8-1)	12+4*(imm8-1)	enter 9,6

Операция

```
level = level MOD 32
2ndOperand = 2ndOperand MOD 32
IF OperandSize = 16 THEN Push(BP); ELSE Push(EBP); FI;
```

```
IF StkSize = 16 THEN FramePtr = SP; ELSE FramePtr = ESP; FI;
FOR i = 1 TO (2ndOperand - 1)
DO
  IF OperandSize = 16
  THEN
    IF StkSize = 16
    THEN
      BP = BP - 2;
      Push([BP]); (* помещает слово в стек *)
    ELSE (* StkSize = 32 *)
      EBP = EBP - 2;
      Push([EBP]); (* помещает слово в стек *)
    FI;
  ELSE (* OperandSize = 32 *)
    IF StkSize = 16
    THEN
      BP = BP - 2;
      Pushd([BP]); (* помещает двойное слово в стек *)
    ELSE (* StkSize = 32 *)
      EBP = EBP - 2;
      Pushd([EBP]); (* помещает двойное слово в стек *)
    FI;
  FI;
OD;
IF StkSize = 16
THEN Push(FramePtr); (* помещает слово в стек *)
ELSE Pushd(FramePtr); (* помещает двойное слово в стек *)
FI;
IF StkSize = 16
THEN
  BP = FramePtr;
  SP = SP - 1stOperand;
ELSE
  EBP = FramePtr;
  ESP = ESP - 1stOperand;
FI;
```

Описание

Команда ENTER создает кадр стека, необходимый большинству блочно-структурных языков программирования высокого уровня. Первый операнд указывает, какое количество байтов динамической памяти выделяется в стеке для процедуры, в которую осуществляется вход. Второй операнд дает лексический уровень вложенности (от 0 до 31) процедуры внутри исходного кода. Он определяет количество указателей кадра стека, скопированных в новый кадр стека из предыдущего кадра.

Атрибут размера операнда и атрибут размера стека используются, чтобы определить: BP или EBP используется для указателя кадра, и SP или ESP - для указателя стека.

Если атрибут размера операнда равен 16 бит, то процессор использует регистр BP в качестве указателя кадров, если только атрибут размера стека не равен 32 бита, тогда он использует EBP для указателя кадра и ESP для указателя стека. Если атрибут размера операнда равен 32 бит, то процессор использует регистр EBP в качестве указателя кадров и регистр

ESP в качестве указателя стека, если только атрибут размера стека не равен 16 бит, тогда он использует BP для указателя кадра и SP для указателя стека.

Если второй операнд равен нулю, тогда команда ENTER помещает в стек указатель кадра (регистр BP или EBP), затем команда ENTER вычитает первый операнд из указателя стека и устанавливает указатель кадра равным текущему значению указателя стека.

Например, подпрограмма, имеющая 12 байтов локальных переменных, должна иметь в своей точке входа команду ENTER 12,0 и команду LEAVE перед каждой командой RET. 12 локальных байт будут адресоваться отрицательными смещениями от указателя кадра.

Особые ситуации защищенного режима

#SS(0), если значение SP или ESP превысит предел стека в любой точке во время выполнения команды. #PF(код ошибки), при страничной ошибке.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

HLT Останов

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	86	
F4	HLT		4	5	2	2	hlt

Операция

Enter halt state;

Описание

Команда HLT останавливает выполнение команд и переводит процессор в состояние останова. Разрешенное прерывание, NMI или RESET возобновляет выполнение. Если прерывания (включая NMI) используются для возобновления выполнения после команды HLT, то сохраненное значение CS:IP (CS:EIP) указывает на команду, следующую за командой HLT.

Особые ситуации защищенного режима

Команда HLT - привилегированная команда. #GP(0), если текущий уровень привилегий не равен 0.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

#GP(0); Команда HLT - привилегированная команда.

IDIV Деление со знаком

O	D	I	T	S	Z	A	P	C
?				?	?	?	?	?

Код	Команда	Такты					Действие	Пример
		Pentium	486	386	286	86		
F6 /7	IDIV AL,r/m8	22	19/20	19	17/20	101-112/ 107-118+EA	AL=AX/(r/m8) AH=AX MOD (r/m8)	idiv dl
F7 /7	IDIV AX,r/m16	30	27/28	27	25/28	165-184/ 171-190+EA	AX=DX:AX/(r/m16) DX=DX:AX MOD (r/m16)	idiv cx
F7 /7	IDIV EAX,r/m32	46	43/44	43			EAX=EDX:EAX/(r/m32) EDX=EDX:EAX MOD (r/m32)	idiv esi

Операция

```
temp = dividend / divisor;
IF (temp не годится в качестве частного)  (*  dividend - делимое  *)
THEN Прерывание 0;                        (*  divizor - делитель *)
ELSE                                      (*  quotient - частное  *)
  quotient = temp;                        (*  remainder - остаток  *)
  remainder = dividend MOD (r/m);
FI;
```

Описание

Команда IDIV выполняет деление со знаком. Делимое, частное и остаток неявно расположены в фиксированных регистрах. Только делитель представлен как явный r/m операнд. Тип делителя определяет, какие из регистров будут использованы.

Размер	Делитель	Частное	Остаток	Делимое
byte	r/m8	AL	AH	AX
word	r/m16	AX	DX	DX:AX
dword	r/m32	EAX	EDX	EDX:EAX

Если результирующее частное слишком велико, чтобы помещаться по назначению, или если делитель равен нулю, то генерируется прерывание 0. Целочисленное частное округляется в сторону нуля. Остаток имеет тот же знак, что и делимое, и абсолютная величина остатка всегда меньше, чем абсолютная величина делителя.

Особые ситуации защищенного режима

Прерывание 0, если частное слишком велико, чтобы подходить регистру назначения (AL или AX), или если делитель равен нулю. #GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 0, если частное слишком велико, чтобы подходить регистру назначения (AL или AX), или если делитель равен нулю. Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

IMUL Знаковое умножение

O	D	I	T	S	Z	A	P	C
*				?	?	?	?	?

Код	Команда	Такты					Действие*	Пример
		Pentium	486	386	286	86		
F6 /5	IMUL r/m8	11	13-18/ 13-18	9-14/ 12-17	13/16	80-98/ 86-104 +EA	AX=AL*r/m8	imul dl
F7 /5	IMUL r/m16	11	13-26/ 13-26	9-22/ 12-25	21/24	128-154/ 134-160 +EA	DX:AX=EAX*r/m16	imul si

Код	Команда	Такты				Действие	Пример
		Pentium	486	386	286		
F7 /5	IMUL r/m32	10	12-42/ 13-42	9-38/ 12-41		EDX:EDX=EAX*r/m32	imul edi
0F AF /r	IMUL r16,r/m16	10	13-26/ 13-26	9-22/ 12-25		r16=r16*r/m16	imul si,[di]
0F AF /r	IMUL r32,r/m32	10	12-42/ 13-42	9-38/ 12-41		r32=r32*r/m32	imul edi,ecx
6B /r ib	IMUL r16,r/m16,imm8	10	13-26/ 13-26	9-14/ 12-17	21/24	r16=r/m16*imm8	imul ax,dx,4
6B /r ib	IMUL r32,r/m32,imm8	10	13-42	9-14/ 12-17		r32=r/m32*imm8	imul eax,edx,4
6B /r ib	IMUL r16,imm8	10	13-26	9-14	21	r16=r16*imm8	imul ax,4
6B /r ib	IMUL r32,imm8	10	13-26	9-14		r32=r32*imm8	imul edx,5
69 /r iw	IMUL r16, r/m16,imm16	10	13-26/ 13-26	9-22/ 12-25	21/24	r16=r/m16*imm16	imul ax,dx,1
69 /r id	IMUL r32, r/m32,imm32	10	13-42/ 13-42	9-38/ 12-41		r32=r/m32*imm32	imul eax,edx,8
69 /r iw	IMUL r16,imm16	10	13-26	9-22		r16=r/m16*imm16	imul ax,8F57h
69 /r id	IMUL r32,imm32	10	13-42	9-38		r32=r/m32*imm32	imul edx,9000

* Множитель (multiplier) предварительно знакорасширяется до размера множимого (multiplicand).

Операция

```
result = multiplicand*multiplier;
```

Описание

Команда IMUL выполняет умножение со знаком. Некоторые формы команды используют неявные операнды регистры (как указано в столбце "Действие").

Команда IMUL очищает флаги OF и CF при следующих условиях (в противном случае флаги OF и CF устанавливаются):

Форма команды	Условия очистки CF и OF
r/m8	AL = SignExtend(AL)
r/m16	AX = SignExtend(AX)
r/m32	EDX:EAX = SignExtend(EAX)
r16,r/m16	Результат помещается в r16
r32,r/m32	Результат помещается в r32
r16,r/m16,imm16	Результат помещается в r16
r32,r/m32,imm32	Результат помещается в r32

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Когда используются формы команды с аккумулятором (IMUL r/m8, IMUL r/m16 или IMUL r/m32), то результат умножения доступен, даже если установлен флаг переполнения, потому что результат в два раза больше по размеру, чем множимое и множитель. Этого достаточно, чтобы поддержать любой возможный результат.

IN Ввод из порта

O D I T S Z A P C

Код операции	Команда		Такты					Пример
			Pentium	486	386	286	86	
E4 ib	IN	AL,imm8	7,pm=4*/ 21**,vm=19	14,pm=8*/ 28**,vm=27	12,pm=6*/ 26**	5	10	in al,60h
E5 ib	IN	AX,imm8	7,pm=4*/ 21**,vm=19	14,pm=8*/ 28**,vm=27	12,pm=6*/ 26**	5	10	in ax,00h
E5 ib	IN	EAX,imm8	7,pm=4*/ 21**,vm=19	14,pm=8*/ 28**,vm=27	12,pm=6*/ 26**			in eax,81h
EC	IN	AL,DX	7,pm=4*/ 21**,vm=19	14,pm=8*/ 28**,vm=27	12,pm=6*/ 26**	5	8	in al,dx
ED	IN	AX,DX	7,pm=4*/ 21**,vm=19	14,pm=8*/ 28**,vm=27	12,pm=6*/ 26**	5	8	in ax,dx
ED	IN	EAX,DX	7,pm=4*/ 25**,vm=19	14,pm=8*/ 28**,vm=27	12,pm=6*/ 26**			in eax,dx

*если CPL ≤ IOPL
**если CPL > IOPL

Операция

```
IF (PE = 1) AND ((VM = 1) OR (CPL > IOPL))
THEN (* виртуальный-8086 режим, или защищенный режим с CPL>IOPL *)
  IF NOT I-O-Permission (SRC, width(SRC))
  THEN #GP(0);
  FI;
FI;
DEST = [SRC]; (* Читает из адресного пространства ввода-вывода*)
```

Описание

Команда IN пересылает байт, слово или двойное слово данных из порта, адресуемого при помощи второго операнда, в регистр (AL, AX или EAX), указанный первым операндом. Адрес порта может задаваться вторым байтом команды (imm8) или содержимым регистра DX. Таким образом, возможно обращение к 256 портам, адреса которых фиксированы в программе (байт imm8), или к 65536 портам, адреса которых могут изменяться путем перезагрузки содержимого регистра DX. При работе с 16 разрядными данными число адресуемых портов уменьшается вдвое, а их адресация должна производиться четными числами: 0, 2, 4 ... 254 для фиксированных портов; 0, 2, 4 ... 65534 для изменяемых портов. Соответственно для 32 разрядных данных число портов сокращается еще вдвое, а их адресация производится числами кратными четырем: 0, 4, 8 ... 252 ... 65532. Таким образом, номера портов должны быть выровнены по границам передаваемых данных согласно их разрядности: 1, 2 или 4 байта. В этом случае пересылка осуществляется за один цикл. Если заданный номер 16 или 32

разрядного порта имеет невыровненное значение, то для передачи данных требуется дополнительный цикл шины.

Особые ситуации защищенного режима

#GP(0), если значение текущего уровня привилегий больше (имеет меньше привилегий), чем уровень привилегий ввода-вывода, и любой из соответствующих битов разрешения ввода-вывода в TSS равен 1.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

#GP(0), если любой из соответствующих битов разрешения ввода-вывода в TSS равен 1.

INC Инкремент

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
FE /0	INC r/m8	1/3	1/3	2/6	2/7	3/15+EA	inc al
FF /0	INC r/m16	1/3	1/3	2/6	2/7	3/15+EA	inc word ptr [di]
FF /0	INC r/m32	1/3	1/3	2/6			inc dword ptr [ebx]
40+rw	INC r16	1	1	2	2	3	inc dx
40+rd	INC r32	1	1	2			inc ecx

Операция

```
DEST = DEST + 1;
```

Описание

Команда INC прибавляет 1 к операнду. Она не изменяет флаг CF. Используйте команду ADD со вторым операндом равным 1, чтобы повлиять на флаг CF.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

INS INSB INSW INSD — Ввод из порта в строку

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
6C	INS DX,r/m8	9,pm=6*/ 24**,vm=22	17,pm=10*/ 32**,vm=30	15,pm=9*/ 29**	5	insb
6D	INS DX,r/m16	9,pm=6*/ 24**,vm=22	17,pm=10*/ 32**,vm=30	15,pm=9*/ 29**	5	insw
6D	INS DX,r/m32	9,pm=6*/ 24**,vm=22	17,pm=10*/ 32**,vm=30	15,pm=9*/ 29**		insd
6C	INSB	9,pm=6*/ 24**,vm=22	17,pm=10*/ 32**,vm=30	15,pm=9*/ 29**	5	insb
6D	INSW	9,pm=6*/ 24**,vm=22	17,pm=10*/ 32**,vm=30	15,pm=9*/ 29**	5	insw
6D	INSD	9,pm=6*/ 24**,vm=22	17,pm=10*/ 32**,vm=30	15,pm=9*/ 29**		insd

*если CPL ≤ IOPL
**если CPL > IOPL

Операция

IF AdressSize = 16
THEN использовать DI для dest-index;

```
ELSE (* AdressSize = 32 *)
  использовать EDI для dest-index;
FI;
IF (PE = 1) AND ((VM = 1) OR (CPL > IOPL))
THEN (* виртуальный-8086 режим, или защищенный режим с CPL > IOPL *)
  IF NOT I-O-Permission (SRC, width(SRC))
  THEN #GP(0);
  FI;
FI;
IF (команда с байтом в качестве операнда)
THEN
  ES:[dest-index] = [DX];
    (* Чтение байта по DX из адресного пространства ввода-вывода *)
  IF DF = 0 THEN IncDec = 1 ELSE IncDec = -1; FI;
ELSE
  IF OperandSize = 16
  THEN
    ES:[dest-index] = [DX];
      (* Чтение слова по DX из адресного пространства ввода-вывода *)
    IF DF = 0 THEN IncDec = 2 ELSE IncDec = -2; FI;
  ELSE (* OperandSize = 32 *)
    THEN
    ES:[dest-index] = [DX];
      (* Чтение двойного слова по DX из адресного пространства ввода-вывода*)
    IF DF = 0 THEN IncDec = 4 ELSE IncDec = -4; FI;
  FI;
FI;
dest-index = dest-index + IncDec;
```

Описание

Команда INS пересылает данные из порта, занумерованного регистром DX в байт, слово или двойное слово в памяти по адресу ES:[dest-index]. Операнд в памяти должен адресоваться регистром ES. Замещение сегмента невозможно. Индексный регистр назначения (dest-index) - это регистр DI, если атрибут размера адреса команды равен 16 бит, или регистр EDI, если атрибут размера адреса равен 32 бита.

Команда INS не позволяет указывать номер порта в качестве непосредственного значения. Порт должен быть адресован через значение регистра DX. Загрузите корректное значение в регистр DX перед выполнением команды INS.

Адрес назначения определяется содержимым индексного регистра назначения. Загрузите корректное значение индекса в индексный регистр перед выполнением команды INS.

После завершения пересылки данных регистр DI или EDI автоматически изменяется. Если флаг DF равен нулю (была выполнена команда CLD), то регистр DI или EDI увеличивается, если флаг DF равен единице (была выполнена команда STD), то регистр DI или EDI уменьшается. Регистр DI уменьшается или увеличивается на 1, если был введен байт, на 2 - если слово, на 4 - если двойное слово.

Команды INSB, INSW, INSD - синонимы команды INS для байта, слова или двойного слова, соответственно. Команде INS может предшествовать префикс REP для ввода блока из CX байтов или слов. Обратитесь к описанию команды REP для подробного рассмотрения такой операции.

Особые ситуации защищенного режима

#GP(0), если значение текущего уровня привилегий больше (имеет меньше привилегий), чем уровень привилегий ввода-вывода, и любой из соответствующих битов разрешения ввода-вывода в TSS равен 1. #GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

#GP(0), если любой из соответствующих битов разрешения ввода-вывода в TSS равен 1. Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

INT Вызов процедуры прерывания
INTO

O	D	I	T	S	Z	A	P	C
		0	0					

Код операции	Команда	Такты					Пример
		Pentium	486	386	286*	86	
CC	INT 3	13	26	33	23	52	int3
CC	INT 3	27	44	pm=59	40		int3
CC	INT 3	44	71	pm=99	78		int3
CC	INT 3	56	82	pm=119			int3
CC	INT 3	19+ts	37+ts	ts	167		int3
CD ib	INT imm8	16	30	37	23	51	int13
CD ib	INT imm8	31	44	pm=59	40		int10
CD ib	INT imm8	48	77	pm=99	78		int13

CD ib	INT imm8	82	86	pm=119		int15
CD ib	INT imm8	23+TS	37+ts	ts	167	int8
CE	INTO	Pass: 13, Fail: 4	Pass: 28, Fail: 3	Fail: 3,pm=3; Pass: 35	Fail: 3, Fail: 4, Pass: 24 Pass: 53	into
CE	INTO	Pass: 27, Fail: 4	46	pm=59	41	into
CE	INTO	Pass: 44, Fail: 4	73	pm=99	79	into
CE	INTO	Pass: 56, Fail: 4	84	pm=119		into
CE	INTO	Pass: 19+TS, Fail: 4	39+ts	ts	168	into

* При выполнении следующей команды добавляется по одному такту на каждый байт команды (только для 80286).

Значения ts определяются следующей таблицей:

Старая задача	Новая задача		
	для 32 битного TSS	для 16 битного TSS	для VM TSS
VM/32 битный/16 битный	85	87	71

Операция

```
IF PE = 0
THEN CALL REAL-ADDRESS-MODE;
ELSE
  CALL PROTECTED-MODE;
  IF (шлюз задачи)
  THEN CALL TASK-GATE;
  ELSE
    CALL TRAP-OR-INT-GATE; (* PE = 1, шлюз прерывания или ловушки *)
    IF (кодовый сегмент несогласованный) AND (DPL < CPL)
    THEN
      IF VM = 0
      THEN CALL INT-TO-INNER-PRIV;
        (* PE = 1, шлюз прерывания или ловушки, DPL < CPL, VM = 0 *)
      ELSE CALL INT-FROM-V86-MODE;
        (* PE = 1, шлюз прерывания или ловушки, DPL < CPL, VM = 1 *)
      FI;
    ELSE (* PE = 1, шлюз прерывания или ловушки, DPL ≥ CPL *)
      IF (кодовый сегмент согласованный) AND (DPL < CPL)
      THEN CALL INT-TO-SAME-PRIV;
      ELSE #GP(CS selector + EXT);
        (* PE = 1, шлюз прерывания или ловушки, DPL > CPL *)
      FI;
    FI;
  FI;
FI;
END;

REAL-ADDRES-MODE PROC
  Push(FLAGS);
  IF = 0; (* Очищает флаг прерываний *)
  TF = 0; (* Очищает флаг ловушки *)
  Push(CS);
  Push(IP);
  (* В стек не заносятся коды ошибок *)
  CS = IDT[номер прерывания * 4].selector;
  IP = IDT[номер прерывания * 4].offset;
(* начать выполнение в режиме реальной адресации *)
REAL-ADDRESS-MODE ENDPROC
```

```
PROTECTED-MODE PROC
  Вектор прерывания должен быть в пределах таблицы IDT,
    иначе #GP(номер прерывания * 8 + 2 + EXT);
  Байт AR дескриптора должен задавать шлюз прерывания, шлюз ловушки или
       шлюз задачи, иначе #GP(номер прерывания * 8 + 2 + EXT);
  IF (прерывание программное) (* т.е. один из случаев INT n, INT 3 или INTO*)
  THEN
    IF (шлюз дескриптора DPL < CPL)
    THEN #GP(номер прерывания * 8 + 2 + EXT);
      (* PE =1, DPL < CPL, программное прерывание *)
    FI;
  FI;
  Шлюз должен присутствовать, иначе #NP(номер прерывания * 8 + 2 + EXT);
PROTECTED-MODE ENDPROC

TRAP-OR-INT-GATE PROC
  Проверка CS селектора и дескриптора заданного в дескрипторе шлюза:
    Селектор должен быть не нулевым, иначе #GP(EXT);
    Индекс селектора должен попадать в пределы таблицы дескрипторов,
      иначе #GP(селектор + EXT);
    AR байт выбранного дескриптора задает кодовый сегмент,
      иначе #GP(селектор + EXT);
    Сегмент должен присутствовать, иначе #NP(селектор + EXT);
TRAP-OR-INT-GATE ENDPROC

INT-TO-INNER-PRIV PROC
  (* PE = 1, шлюз прерывания или шлюз ловушки, DPL < CPL, VM = 0 *)
  Проверка селектора и дескриптора для нового стека в текущем TSS:
    Селектор должен быть не нулевым, иначе #TS(EXT);
    Индекс селектора должен попадать в пределы таблицы дескрипторов,
      иначе #TS(SS селектор + EXT);
    RPL селектора должен быть равен DPL кодового сегмента,
      иначе #TS(SS селектор + EXT);
    DPL стекового сегмента должен быть равен DPL кодового сегмента,
      иначе #TS(SS селектор + EXT);
    AR байт дескриптора должен задавать разрешенный для записи сегмент
       данных, иначе #TS(SS селектор + EXT);
    Сегмент должен присутствовать, иначе #SS(SS селектор + EXT);
  IF (32 битный шлюз)
  THEN новый стек должен иметь место для 20 байт, иначе #SS(0)
  ELSE новый стек должен иметь место для 10 байт, иначе #SS(0)
  FI;
  Указатель инструкции должен попадать в пределы сегмента CS, иначе #GP(0);
  Загрузить новые значения SS и eSP из TSS;
  IF (32 битный шлюз)
  THEN CS:EIP = селектор:смещение из шлюза;
  ELSE CS:IP = селектор:смещение из шлюза;
  FI;
  Загрузить дескриптор CS в скрытую часть регистра CS;
  Загрузить дескриптор SS в скрытую часть регистра SS;
  IF (32 битный шлюз)
  THEN
    Push(длинный указатель на старый стек) (* 3 слова дополняются до 4 *);
    Push(EFLAGS);
    Push(длинный указатель на точку возврата) (* 3 слова дополняются до 4 *);
  ELSE
    Push(длинный указатель на старый стек) (* 2 слова *);
    Push(FLAGS);
    Push(длинный указатель на точку возврата) (* 2 слова *);
  FI;
  Установить CPL равным DPL нового кодового сегмента;
  Установить RPL в CS равным CPL;
  IF (шлюз прерывания) THEN IF = 0 (* сбросить флаг прерывания *); FI;
  TF = 0;
```

```
  NT = 0;
INT-FROM-INNER-PRIV ENDPROC

INT-FROM-V86-MODE PROC
  Проверка селектора и дескриптора для нового стека в текущем TSS:
    Селектор должен быть не нулевым, иначе #TS(EXT);
    Индекс селектора должен попадать в пределы таблицы дескрипторов,
      иначе #TS(SS селектор + EXT);
    RPL селектора должен быть равен DPL кодового сегмента,
      иначе #TS(SS селектор + EXT);
    DPL стекового сегмента должен быть равен DPL кодового сегмента,
      иначе #TS(SS селектор + EXT);
    AR байт дескриптора должен задавать разрешенный для записи сегмент
       данных, иначе #TS(SS селектор + EXT);
    Сегмент должен присутствовать, иначе #SS(SS селектор + EXT);
  IF (32 битный шлюз)
  THEN новый стек должен иметь место для 20 байт, иначе #SS(0)
  ELSE новый стек должен иметь место для 10 байт, иначе #SS(0)
  FI;
  Указатель инструкции должен попадать в пределы сегмента CS, иначе #GP(0);
  IF IOPL < 3
  THEN
    #GP(0); (* ловушка монитора V86: PE = 1, шлюз прерывания или ловушки,
                DPL < CPL, VM = 1, IOPL < 3 *)
  ELSE (* IOPL = 3 *)
    IF GATE'S_DPL = 3
    THEN
      IF TARGET'S_CPL ≠ 0
      THEN #GP(0);
      ELSE
        TempEflags = EFLAGS;
        VM = 0;
        TF = 0;
        IF (обслуживание через шлюз прерывания)
        THEN IF = 0;
        FI;
        TempSS = SS;
        TempESP = ESP;
        SS = TSS.SS0; (* изменяется на стековый сегмент уровня 0 *)
        ESP = TSS.ESP0; (* изменяется на указатель стека уровня 0 *)
        Push(GS); (* расширяется до двух слов *)
        Push(FS); (* расширяется до двух слов *)
        Push(DS); (* расширяется до двух слов *)
        Push(ES); (* расширяется до двух слов *)
        GS = 0; (* сегментный регистр обнуляется - недопустимо в защищенном
                   режиме *)
        FS = 0;
        DS = 0;
        ES = 0;
        Push(TempSS); (* расширяется до двух слов *)
        Push(TempESP);
        Push(TempEflags);
        Push(CS); (* расширяется до двух слов *)
        Push(EIP);
        CS:EIP = селектор:смещение из шлюза прерывания;
        (* Начать выполнение новой процедуры в защищенном режиме *)
      FI;
    ELSE (* GATE'S_DPL ≠ 3*)
      #GP(0);
    FI;
  FI;
INT-FROM-V86-MODE ENDPROC

INT-TO-SAME-PRIV PROC
  (* PE = 1, DPL = CPL или согласованный сегмент *)
```

```
  IF (32 битный шлюз)
  THEN новый стек должен иметь место для 10 байт, иначе #SS(0)
  ELSE новый стек должен иметь место для 6 байт, иначе #SS(0)
  FI;
  IF (прерывание произошло по исключению с кодом ошибки)
  THEN В стеке должно быть место еще для двух байт, иначе #SS(0);
  FI;
  Указатель инструкции должен попадать в пределы сегмента CS, иначе #GP(0);
  IF (32 битный шлюз)
  THEN
    Push(EFLAGS);
    Push(длинный указатель на точку возврата) (* 3 слова дополняются до 4 *);
    CS:EIP = селектор:смещение из шлюза;
  ELSE
    Push(FLAGS);
    Push(длинный указатель на точку возврата) (* 2 слова *);
    CS:IP = селектор:смещение из шлюза;
  FI;
  Загрузить дескриптор CS в скрытую часть регистра CS;
  Установить RPL в CS равным CPL;
   Push(код ошибки); (* если есть *)
  IF (шлюз прерывания) THEN IF = 0; FI;
  TF = 0;
  NT = 0
INT-TO-SAME-PRIV ENDPROC

TASK-GATE PROC (* PE = 1, шлюз задачи *)
  Проверка селектора TSS в дескрипторе шлюза задачи:
    Селектор должен задавать GDT (бит TI=0), иначе #TS(TSS селектор);
    Индекс селектора должен попадать в пределы таблицы GDT,
      иначе #TS(TSS селектор);
    AR байт TSS дескриптора должен задавать свободный TSS (нижние
      биты 0001), иначе #TS(TSS селектор);
    TSS должен присутствовать, иначе #NP(TSS селектор);
  SWITCH-TASKS (с вложением) в TSS;
  IF (прерывание произошло по исключению с кодом ошибки)
  THEN
    В стеке должно быть место еще для двух байт, иначе #SS(0);
    Поместить код ошибки в стек;
  FI;
  Указатель инструкции должен попадать в пределы сегмента CS, иначе #GP(0);
TASK-GATE ENDPROC
```

Таблица действий

Следующая таблица показывает, какое действие из нижней части таблицы происходит в случае выполнения условий из верхней части таблицы. Каждый символ Y в нижней части указывает на то, что выполняется соответствующая процедура, определенная в пункте "Операция" (или имеет место GP(0)). Число следующее за Y указывает, в каком порядке выполняются процедуры.

PE	0	1	1	1	1	1	1	1
VM	-	-	-	-	-	0	1	1
IOPL	-	-	-	-	-	-	<3	=3
Соотношение DPL и CPL	-	DPL< CPL	-	DPL> CPL	DPL= CPL	DPL< CPL	-	-
Тип прерывания	-	S/W	-	-	-	-	-	-

Тип шлюза	-	-	Задачи	Задачи или прерывания	Задачи или прерывания	Задачи или прерывания	Задачи или прерывания	Задачи или прерывания
REAL-ADDRESS-MODE	Y							
PROTECTED-MODE		Y1	Y1	Y1	Y1	Y1	Y1	Y1
TRAP-OR-INT-GATE				Y2	Y2	Y2	Y2	Y2
INT-TO-INNER-PRIV						Y3		
INT-TO-SAME-PRIV					Y3			
INT-FROM-V86-MODE								Y3
TASK-GATE			Y2					
#GP		Y2		Y3			Y3	

Описание

Команда INT n генерирует вызов обработчика прерывания. Непосредственный операнд от 0 до 255 дает номер индекса вызываемого прерывания в таблице дескрипторов прерываний (IDT). В защищенном режиме IDT состоит из множества 8 байтовых дескрипторов. Дескриптор вызываемого прерывания указывает прерывание, ловушку или шлюз задачи. В режиме реальной адресации IDT является массивом 4 байтных дальних указателей. В защищенном режиме и режиме реальной адресации линейный базовый адрес IDT определяется содержимым IDTR. Первоначальное значение IDTR равно нулю при установке в режим реальных адресов.

Когда процессор находится в режиме V86 (VM = 1), IOPL определяет случаи, когда INT n вызывает особую ситуацию общей защиты (IOPL < 3), или выполняется прерывание защищенного режима на уровне привилегий 0. DPL шлюза прерываний должен быть установлен в 3, а целевой CPL процедуры обработки прерывания должен быть 0, чтобы выполнить прерывание защищенного режима на нулевом уровне привилегий.

Условно-выраженная команда программного обеспечения INTO идентична команде прерывания INT n за исключением того, что номер прерывания задается неявно равным 4, и прерывание выполняется только, если установлен флаг переполнения (OF).

Первые 32 прерывания зарезервированы Intel для системного использования. Некоторые из этих прерываний используются при генерации внутренних исключений.

Команда INT n в общем ведет себя как дальний вызов, за исключением того, что регистр флагов помещается в стек перед адресом воз-

врата. Процедуры прерывания завершаются командой IRET, которая выбирает из стека флаги и адрес возврата.

В режиме реальной адресации команда INT n помещает в стек флаги, регистр CS и IP возврата в этом порядке, затем переходит по дальнему указателю, индексированному номером прерывания.

Смотрите также рис. 3-3 и рис. 3-4 на стр. 43, 44, на которых показано содержимое стека после прерывания в защищенном режиме и в режиме V86.

Особые ситуации защищенного режима

#GP, #NP, #SS и #TS, как указано выше в пункте "Операция".

Особые ситуации режима реальной адресации

Нет. Если регистр SP или ESP равен 1, 3 или 5 перед выполнением команды INT или INTO, то процессор останавливается из-за недостаточного пространства стека.

Особые ситуации режима V86

#GP(0), если IOPL < 3 и только для команды INT n, чтобы разрешить эмуляцию прерывания 3, генерируется особая ситуация "точка останова"; команда INTO генерирует особую ситуацию "переполнение", если установлен флаг OF.

Для дополнительной информации об особых ситуациях при использовании этой команды в режиме V86 смотрите Приложение H.

INVD Аннулировать КЭШ

O D I T S Z A P C

Код операции	Команда	Такты		Пример
		Pentium	486	
0F 08	INVD	15	4	invd

Операция

Сделать недействительным (INVALIDATE) внутренний КЭШ
Выдать сигнал внешнему КЭШ о недействительности (INVALIDATE) данных

Описание

Команда INVD делает недействительным содержимое внутреннего КЭШ. Далее производится особый функциональный цикл шины, который указывает, что внешние КЭШ также должны стать недействительными. Для внешних КЭШ инструкция на обратную запись данных не выдается.

Особые ситуации защищенного режима

Команда INVD - привилегированная команда. #GP(0), если текущий уровень привилегий не равен 0.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

#GP(0); Команда INVD - привилегированная команда.

Замечание

INVD должна использоваться осторожно. Она не записывает обратно измененные линии КЭШ, поэтому она может вызвать несогласование данных КЭШ с любой другой памятью в системе. Если не имеется особой необходимости не допустить обратную запись измененных линий КЭШ перед их аннулированием (invalidate), то программа должна использовать команду WBINVD. Команда INVD используется при тестировании и устранении ошибок, когда соответствие с основной памятью не нужно.

Эта команда реализационно-зависима, т.е. эта функция может быть реализована иначе на будущих процессорах Intel.

Эта команда не ожидает, пока внешний КЭШ выполнит необходимые действия. На ответственности аппаратного обеспечения лежит обязанность отреагировать на указание недействительности внешнего КЭШ.

Эта команда поддерживается только процессорами Intel486 и Pentium.

INVLPG Аннулировать элемент TLB

O D I T S Z A P C

Код операции	Команда	Такты		Пример
		Pentium	486	
0F 01 /7	INVLPG m	25	12 for hit	invlpg [bx]

Операция

Сделать недействительной точку(и) входа в TLB
(* INVALIDATE RELEVANT TLB ENTRY(S) *)

Описание

Команда INVLPG используется для того, чтобы убедиться, что в TLB (буфер ассоциативной трансляции) нет ненужных элементов. Если TLB содержит входы, отражающие адрес операнда команды INVLPG, то все соответствующие элементы TLB помечаются как недействительные.

Особые ситуации защищенного режима

Команда INVLPG - привилегированная команда. #GP(0), если CPL не равен нулю. #UD, если команда используется с операндом регистром.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

#UD, если команда используется с операндом регистром. #GP(0), так как команда INVLPG - привилегированная команда.

Замечание

Эта команда поддерживается только процессорами i80486 и Pentium.

IRET Возврат из прерывания
IRETD

O	D	I	T	S	Z	A	P	C
*	*	*	*	*	*	*	*	*

Код операции	Команда	Такты					Описание
		Pentium	486	386	286	86	
CF	IRET	8	15	22, pm=60	17	32	Возврат из прерывания в реальном и V86 режимах (far return and pop flags)
CF	IRET	10	15	pm=38	pm=31		Возврат из прерывания в защищенном режиме (far return and pop flags)
CF	IRET	27	36	pm=82	55		Возврат из прерывания на более низкий уровень привилегий
CF	IRET	ts+10	ts+32	ts	169		Возврат из прерывания в другую задачу (NT = 1)
CF	IRETD	10	15	22, pm=38			Возврат из прерывания (far return and pop flags)
CF	IRETD	27	36	pm=82			Возврат из прерывания на более низкий уровень привилегий
CF	IRETD	ts+10	ts+32	ts			Возврат из прерывания в другую задачу (NT = 1)

Значение ts определяется следующей таблицей:

Старая задача	Новая задача		
	для 32 битного TSS	для 16 битного TSS	для VM TSS
VM/32 битный/16 битный	85	87	71

Операция

```
IF PE = 0
THEN GOTO REAL_ADDRESS_MODE;
ELSE GOTO PROTECTED_MODE;
FI;

REAL_ADDRESS_MODE:
  IF OperandSize = 32 (* команда типа IRETD *)
  THEN EIP = Pop();
  ELSE (* команда типа IRET *)
    IP = Pop();
  FI;
  CS = Pop();
  IF OperandSize = 32 (* команда типа IRETD *)
  THEN Pop(); EFLAGS = Pop();
  ELSE (* команда типа IRET *)
    FLAGS = Pop();
  FI;
END;

PROTECTED_MODE:
  IF VM = 1 (* виртуальный режим: PE = 1, VM = 1 *)
```

```
  THEN GOTO STACK_RETURN_FROM_V86; (* PE = 1, VM = 1 *)
  ELSE
    IF NT = 1
    THEN GOTO TASK_RETURN; (* PE = 1, VM = 1, NT = 1 *)
    ELSE
      IF (VM = 1 в образе флагов в стеке)
        THEN GOTO STACK_RETURN_TO_V86; (* PE = 1, VM = 1 в образе
                                          флагов в стеке *)
        ELSE GOTO STACK_RETURN; (* PE = 1, VM = 0 в образе флагов в
                                   стеке *)
      FI;
    FI;
  FI;
END;

STACK_RETURN_FROM_V86:
  IF IOPL = 3 (* виртуальный режим: PE = 1, VM = 1, IOPL = 3 *)
  THEN
    IF OperandSize = 16
      IP = Pop(); (* восстанавливает из стека 16 бит *)
      CS = Pop();
      FLAGS = Pop();
    ELSE (* OperandSize = 32 *)
      EIP = Pop(); (* восстанавливает из стека 16 бит *)
      CS = Pop();
      EFLAGS = Pop(); (* биты VM, IOPL, VIP и VIF регистра EFLAGS
                         командой IRETD не изменяются *)
    FI;
  ELSE #GP(0); (* ловушка в монитор V86: PE = 1, VM = 1, IOPL < 3 *)
  FI;
END;

STACK_RETURN_TO_V86: (* Прерывание произошло в режиме V86: PE = 1,
                        VM = 1 в образе флагов в стеке *)
  В пределы стека должны попадать верхние 36 байт, иначе #SS(0);
  Указатель инструкции должен попадать в пределы сегмента CS, иначе #GP(0);
  EFLAGS = SS:[ESP + 8]; (* Устанавливает VM в прерванной процедуре *)
  EIP = Pop();
  CS = Pop(); (* CS поступает как в 8086, при VM = 1 *)
  throwaway = Pop(); (* читает из стека, но не запоминает EFLAGS *)
  TempESP = Pop();
  TempSS = Pop();
  ES = Pop(); (* восстанавливает из стека 2 слова; старшее слово теряется *)
  DS = Pop(); (* восстанавливает из стека 2 слова; старшее слово теряется *)
  FS = Pop(); (* восстанавливает из стека 2 слова; старшее слово теряется *)
  GS = Pop(); (* восстанавливает из стека 2 слова; старшее слово теряется *)
  SS:ESP = TempSS:TempESP;
  (* Продолжить выполнение в виртуальном-8086 режиме *)
END;

TASK_RETURN: (* PE = 1, VM = 1, NT = 1 *)
  Проверка селектора обратной связи в TSS, адресуемого регистром задачи:
    Селектор должен задавать GDT (бит TI=0), иначе #TS(новый TSS селектор);
    Индекс селектора должен попадать в пределы таблицы GDT,
      иначе #TS(новый TSS селектор);
    AR байт должен задавать свободный TSS, иначе #TS(новый TSS селектор);
    TSS должен присутствовать, иначе #NP(новый TSS селектор);
  SWITCH-TASKS (без вложения) в TSS заданный селектором обратной связи;
  Пометить покинутую задачу, как не занятую;
  Указатель инструкции должен попадать в пределы сегмента CS, иначе #GP(0);
END;

STACK-RETURN: (* PE = 1, VM = 0 в образе флагов в стеке *)
  IF OperandSize = 32
  THEN В пределы стека должны попадать верхние 3 слова, иначе #SS(0);
```

```
    ELSE В пределы стека должны попадать верхние 2 слова, иначе #SS(0);
    FI;
    Для возвращаемого CS селектора RPL должно быть ≥ CPL,
      иначе #GP(возвращаемый селектор);
    IF (RPL возвращаемого селектора = CPL)
    THEN GOTO RETURN-SAME-LEVEL;
    ELSE GOTO RETURN-OUTER-LEVEL;
    FI;
  END;

  RETURN-SAME-LEVEL: (* PE = 1, VM = 0 в образе флагов в стеке ,
                        RPL = CPL *)
    IF OperandSize = 32
    THEN
      В пределы стека должны попадать верхние 12 байт, иначе #SS(0);
      Возвращаемый CS селектор (в [eSP + 4]) должен быть не нулевым,
        иначе #GP(0);
    ELSE
      В пределы стека должны попадать верхние 6 байт, иначе #SS(0);
      Возвращаемый CS селектор (в [eSP + 2]) должен быть не нулевым,
        иначе #GP(0);
    FI;
    Индекс селектора должен попадать в пределы таблицы дескрипторов,
        иначе #GP(возвращаемый селектор);
    AR байт должен задавать кодовый сегмент,
        иначе #GP(возвращаемый селектор);
    IF (несогласованный)
    THEN (DPL кодового сегмента = CPL);
    ELSE #GP(возвращаемый селектор); (* PE = 1, VM = 0 в образе флагов в
                                        стеке,
      RPL = CPL, несогласованный, DPL ≠ CPL *)
    FI;
    IF (согласованный)
    THEN
      IF (DPL > CPL) THEN #GP(возвращаемый селектор); FI;
      (* PE = 1, VM = 0 в образе флагов в стеке, RPL = CPL, согласованный,
        DPL > CPL *)
      Сегмент должен присутствовать, иначе #NP(возвращаемый селектор);
      Указатель инструкции должен попадать в пределы сегмента CS,
        иначе #GP(0);
    FI;
    IF OperandSize = 32
    THEN
      Загрузить CS:EIP из стека;
      Загрузить новый дескриптор регистра CS;
      Загрузить EFLAGS с третьим двойным словом из стека;
      Увеличить eSP на 12;
    ELSE
      Загрузить CS:IP из стека;
      Загрузить новый дескриптор регистра CS;
      Загрузить FLAGS с третьим словом из стека;
      Увеличить eSP на 6;
    FI;
  END;

  RETURN-OUTER-LEVEL:
    IF OperandSize = 32
    THEN В пределы стека должны попадать верхние 20 байт, иначе #SS(0);
    ELSE В пределы стека должны попадать верхние 10 байт, иначе #SS(0);
    FI;
    Проверка CS селектора и дескриптора кодового сегмента:
      Селектор должен быть не нулевым, иначе #GP(0);
      Индекс селектора должен попадать в пределы таблицы дескрипторов,
        иначе #GP(возвращаемый селектор);
      AR байт выбранного дескриптора задает кодовый сегмент,
```

```
      иначе #GP(возвращаемый селектор);
  IF (несогласованный)
  THEN
    DPL кодового сегмента = RPL CS селектора,
      иначе #GP(возвращаемый селектор);
  FI;
  Сегмент должен присутствовать, иначе #NP(возвращаемый селектор);
  Проверка SS селектора и дескриптора стека:
    Селектор должен быть не нулевым, иначе #GP(0);
    Индекс селектора должен попадать в пределы таблицы дескрипторов,
      иначе #GP(SS селектор);
    RPL селектора должен быть равен RPL возвращаемого CS селектора,
      иначе #GP(SS селектор);
    AR байт дескриптора должен задавать разрешенный для записи сегмент
        данных, иначе #GP(SS селектор);
    DPL сегмента стека должен быть равен RPL возвращаемого CS селектора,
      иначе #GP(SS селектор);
    Сегмент должен присутствовать, иначе #NP(SS селектор);
  Указатель инструкции должен попадать в пределы сегмента CS, иначе #GP(0);
  IF OperandSize = 32
  THEN
    Загрузить CS:EIP из стека;
    Загрузить EFLAGS величиной из [eSP + 8];
  ELSE
    Загрузить CS:IP из стека;
    Загрузить FLAGS величиной из [eSP + 4];
  FI;
  Загрузить SS:eSP из стека;
  Устанавливает CPL равным RPL в возвращаемом CS селекторе;
  Загрузить дескриптор CS в скрытую часть регистра CS;
  Загрузить дескриптор SS в скрытую часть регистра SS;
  DO (Для каждого из регистров ES, FS, GS и DS)
    IF (значение регистра недопустимо для текущего уровня)
    THEN обнулить регистр (селектор = AR = 0);
    FI;
    Быть допустимым для текущего уровня - значит отвечать следующим
                                                        условиям:
      Индекс селектора должен попадать в пределы таблицы дескрипторов;
      AR байт дескриптора должен задавать разрешенный для чтения сегмент;
      IF (сегмент это сегмент данных или несогласованный кодовый сегмент)
      THEN DPL должен быть > CPL, или DPL должен быть < RPL;
      FI;
  OD;
END;
```

Описание

В режиме реальной адресации команда IRET выбирает из стека указатель команд, регистр CS и регистр флагов и продолжает выполнение прерванной процедуры.

В защищенном режиме действие команды IRET зависит от установки бита (NT) флага вложенности задачи в регистре флагов. Когда из стека извлекается новое значение флага, то биты IOPL в регистре флагов изменяются только когда CPL = 0.

Если флаг NT сброшен, то команда IRET возвращает управление из процедуры прерывания без переключения задачи. Код, которому возвращается управление, должен быть равно или менее привилегирован, чем про-

цедура прерывания (как указывается битами RPL селектора CS, получаемого из стека). Если код назначения менее привилегирован, то команда IRET также получает из стека указатель стека и SS.

Если флаг NT установлен, то команда IRET производит действие обратное действиям команды CALL или INT, которая вызвала переключение задачи. Обновленное состояние задачи при выполнении команды IRET сохраняется в сегменте состояния задачи.

Особые ситуации защищенного режима

#GP, #NP или #SS, как указано выше в пункте "Операция". #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда, извлекаемая из стека, лежит вне адреса 0FFFFh.

Особые ситуации режима V86

#GP(0), если уровень привилегий ввода-вывода меньше 3, для разрешения эмуляции. #PF(код ошибки), страничная ошибка. #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

За дополнительной информацией по исключениям, используемым этой командой в V86 режиме, обращайтесь к Приложению H.

Jcc Условный переход

O D I T S Z A P C

Код	Команда	Такты				Условие осуществления перехода
		Pentium	486	386	286	86	
77 cb	JA rel8	1	3,1	7+m,3	7,3	16,4	Если выше (CF=0 и ZF=0)
73 cb	JAE rel8	1	3,1	7+m,3	7,3	16,4	Если выше или равно (CF=0)
72 cb	JB rel8	1	3,1	7+m,3	7,3	16,4	Если ниже (CF=1)
76 cb	JBE rel8	1	3,1	7+m,3	7,3	16,4	Если ниже или равно (CF=1 или ZF=1)
72 cb	JC rel8	1	3,1	7+m,3	7,3	16,4	Если перенос (CF=1)
E3 cb	JCXZ rel8	6,5	3,1	9+m,5	8,4	18,6	Если CX=0
E3 cb	JECXZ rel8	6,5	3,1	9+m,5			Если ECX=0

74 cb	JE rel8	1	3,1	7+m,3	7,3	16,4	Если равно (ZF=1)
74 cb	JZ rel8	1	3,1	7+m,3	7,3	16,4	Если нуль (ZF=1)
7F cb	JG rel8	1	3,1	7+m,3	7,3	16,4	Если больше (ZF=0 и SF=OF)
7D cb	JGE rel8	1	3,1	7+m,3	7,3	16,4	Если больше или равно (SF=OF)
7C cb	JL rel8	1	3,1	7+m,3	7,3	16,4	Если меньше (SF<>OF)
7E cb	JLE rel8	1	3,1	7+m,3	7,3	16,4	Если меньше или равно (ZF=1 или SF<>OF)
76 cb	JNA rel8	1	3,1	7+m,3	7,3	16,4	Если не выше (CF=1 или ZF=1)
72 cb	JNAE rel8	1	3,1	7+m,3	7,3	16,4	Если не выше или равно (CF=1)
73 cb	JNB rel8	1	3,1	7+m,3	7,3	16,4	Если не ниже (CF=0)
77 cb	JNBE rel8	1	3,1	7+m,3	7,3	16,4	Если не ниже или равно (CF=0 и ZF=0)
73 cb	JNC rel8	1	3,1	7+m,3	7,3	16,4	Если не перенос (CF=0)
75 cb	JNE rel8	1	3,1	7+m,3	7,3	16,4	Если не равно (ZF=0)
7E cb	JNG rel8	1	3,1	7+m,3	7,3	16,4	Если не больше (ZF=1 или SF<>OF)
7C cb	JNGE rel8	1	3,1	7+m,3	7,3	16,4	Если не больше или равно (SF<>OF)
7D cb	JNL rel8	1	3,1	7+m,3	7,3	16,4	Если не меньше (SF=OF)
7F cb	JNLE rel8	1	3,1	7+m,3	7,3	16,4	Если не меньше или равно (ZF=0 и SF=OF)
71 cb	JNO rel8	1	3,1	7+m,3	7,3	16,4	Если не переполнение (OF=0)
7B cb	JNP rel8	1	3,1	7+m,3	7,3	16,4	Если нечетно (PF=0)
79 cb	JNS rel8	1	3,1	7+m,3	7,3	16,4	Если положительно (SF=0)
75 cb	JNZ rel8	1	3,1	7+m,3	7,3	16,4	Если не нуль (ZF=0)
70 cb	JO rel8	1	3,1	7+m,3	7,3	16,4	Если переполнение (OF=1)
7A cb	JP rel8	1	3,1	7+m,3	7,3	16,4	Если четно (PF=1)
7A cb	JPE rel8	1	3,1	7+m,3	7,3	16,4	Если четно (PF=1)
7B cb	JPO rel8	1	3,1	7+m,3	7,3	16,4	Если нечетно (PF=0)
78 cb	JS rel8	1	3,1	7+m,3	7,3	16,4	Если отрицательно (SF=1)

* Если переход осуществляется, при выполнении следующей команды добавляется по одному такту на каждый байт команды (только для Intel286).

Код	Команда	Такты			Условие осуществления перехода
		Pentium	486	386	
0F 87 cw/cd	JA rel16/32	1	3,1	7+m,3	Если выше (CF=0 и ZF=0)
0F 83 cw/cd	JAE rel16/32	1	3,1	7+m,3	Если выше или равно (CF=0)
0F 82 cw/cd	JB rel16/32	1	3,1	7+m,3	Если ниже (CF=1)
0F 86 cw/cd	JBE rel16/32	1	3,1	7+m,3	Если ниже или равно (CF=1 или ZF=1)
0F 82 cw/cd	JC rel16/32	1	3,1	7+m,3	Если перенос (CF=1)
0F 84 cw/cd	JE rel16/32	1	3,1	7+m,3	Если равно (ZF=1)
0F 84 cw/cd	JZ rel16/32	1	3,1	7+m,3	Если нуль (ZF=1)
0F 8F cw/cd	JG rel16/32	1	3,1	7+m,3	Если больше (ZF=0 и SF=OF)
0F 8D cw/cd	JGE rel16/32	1	3,1	7+m,3	Если больше или равно (SF=OF)
0F 8C cw/cd	JL rel16/32	1	3,1	7+m,3	Если меньше (SF<>OF)
0F 8E cw/cd	JLE rel16/32	1	3,1	7+m,3	Если меньше или равно (ZF=1 или SF<>OF)
0F 86 cw/cd	JNA rel16/32	1	3,1	7+m,3	Если не выше (CF=1 или ZF=1)
0F 82 cw/cd	JNAE rel16/32	1	3,1	7+m,3	Если не выше или равно (CF=1)
0F 83 cw/cd	JNB rel16/32	1	3,1	7+m,3	Если не ниже (CF=0)
0F 87 cw/cd	JNBE rel16/32	1	3,1	7+m,3	Если не ниже или равно (CF=0 и ZF=0)
0F 83 cw/cd	JNC rel16/32	1	3,1	7+m,3	Если не перенос (CF=0)
0F 85 cw/cd	JNE rel16/32	1	3,1	7+m,3	Если не равно (ZF=0)
0F 8E cw/cd	JNG rel16/32	1	3,1	7+m,3	Если не больше (ZF=1 или SF<>OF)
0F 8C cw/cd	JNGE rel16/32	1	3,1	7+m,3	Если не больше или равно (SF<>OF)
0F 8D cw/cd	JNL rel16/32	1	3,1	7+m,3	Если не меньше (SF=OF)
0F 8F cw/cd	JNLE rel16/32	1	3,1	7+m,3	Если не меньше или равно (ZF=0 и SF=OF)
0F 81 cw/cd	JNO rel16/32	1	3,1	7+m,3	Если не переполнение (OF=0)
0F 8B cw/cd	JNP rel16/32	1	3,1	7+m,3	Если нечетно (PF=0)
0F 89 cw/cd	JNS rel16/32	1	3,1	7+m,3	Если положительно (SF=0)
0F 85 cw/cd	JNZ rel16/32	1	3,1	7+m,3	Если не нуль (ZF=0)
0F 80 cw/cd	JO rel16/32	1	3,1	7+m,3	Если переполнение (OF=1)
0F 8A cw/cd	JP rel16/32	1	3,1	7+m,3	Если четно (PF=1)
0F 8A cw/cd	JPE rel16/32	1	3,1	7+m,3	Если четно (PF=1)
0F 8B cw/cd	JPO rel16/32	1	3,1	7+m,3	Если нечетно (PF=0)
0F 88 cw/cd	JS rel16/32	1	3,1	7+m,3	Если отрицательно (SF=1)

Значение ts определяется следующей таблицей:

Старая задача	Новая задача		
	для 32 битного TSS	для 16 битного TSS	для VM TSS
VM/32 битный/16 битный	85	87	71

Операция

IF condition

```
THEN
  EIP = EIP + SignExtend(r/m8/16/32);
  IF OperandSize = 16
  THEN EIP = EIP AND 0000FFFFh;
  FI;
FI;
```

Описание

Условные переходы (исключая команду JCXZ) проверяют флаги, которые были установлены предыдущей командой. Условия для каждой мнемоники даны выше. Термины "меньше" и "больше" используются для сравнения знаковых целых, "выше" и "ниже" используются для беззнаковых целых.

Если данное условие истинно, то происходит переход на местоположение, указанное операндом. Такие команды наиболее эффективны, когда цель условного перехода находится в текущем кодовом сегменте в диапазоне от - 128 до +127 байт относительно первого байта следующей команды. Переход также может направляться от -32768 до +32767 (атрибут размера сегмента 16) или от -2^{31} до $+2^{31}$ (атрибут размера сегмента 32) относительно первого байта следующей команды. Когда цель условного перехода находится в другом сегменте, тогда используйте противоположную форму команды перехода (т.е. команды JE и JNE) и получайте доступ к цели при помощи безусловного дальнего перехода в другой сегмент. Например, вы не можете запрограммировать:

```
JZ FARLABEL
```

Вместо этого вы должны программировать:

```
JNZ BEYOND
JMP FARLABEL
BEYOND:
```

Исходя из того, что может быть несколько путей интерпретации состояния флагов, ASM386 обеспечивает более чем одну мнемонику для большинства кодов операций условных переходов. Например, если вы сравниваете два символа в EAX и хотите перейти если они равны, то используйте команду JE, а если вы по битно логически умножили AX с маской битового поля и хотите перейти только если результат равен нулю, то используйте команду JZ - синоним команды JE.

JCXZ отличается от команды условного перехода тем, что она проверяет содержимое регистра CX или ECX на нуль, а не флаги. Команда JCXZ полезна вначале условного цикла, который завершается командой

условного цикла (такой как LOOPNE TARGETLABEL). Команда JCXZ предотвращает вхождение в цикл с регистром CX или ECX равным нулю.

Особые ситуации защищенного режима

#GP(0), если смещение перешло за пределы кодового сегмента.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

Замечание

Команда JCXZ выполняется дольше, чем последовательность из двух команд, которая сравнивает регистр счетчик с нулем и осуществляет переход, если счетчик равен нулю.

JMP Переход

O D I T S Z A P C

Код	Команда	Pentium	486	Такты 386	286	86	Описание типа перехода
EB cb	JMP rel8	1	3	7+m	7	15	короткий
E9 cw	JMP rel16	1	3	7+m	7	15	ближний, смещение указывается относительно следующей команды
FF /4	JMP r/m16	2	5,5	7+m, 10+m	7,11	11, 18+EA	ближний косвенный
EA cd	JMP ptr16:16	3	17, pm=19	12+m, pm=27+m	11, pm=23	15	межсегментный, указывается 4-байтный непосредственный адрес
EA cd	JMP ptr16:16	18	32	pm=45+m	38		на шлюз вызова, привилегии не изменяются
EA cd	JMP ptr16:16	19+ts	42+ts	ts	175		через TSS
EA cd	JMP ptr16:16	20+ts	43+ts	ts	180		через шлюз задачи
FF /5	JMP m16:16	4	13, pm=18	43+m, pm=31+m	15, pm=26	24+EA	косвенный межсегментный
FF /5	JMP m16:16	18	31	pm=49+m	41		на шлюз вызова, привилегии не изменяются
FF /5	JMP m16:16	19+ts	41+ts	5+ts	178		через TSS
FF /5	JMP m16:16	20+ts	42+ts	5+ts	183		через шлюз задачи
E9 cd	JMP rel32	1	3	7+m			ближний, смещение указывается относительно следующей команды
FF /4	JMP r/m32	2	5,5	7+m, 10+m			ближний косвенный
EA cp	JMP ptr16:32	3	13, pm=18	12+m, pm=27+m			межсегментный, указывается 6-байтный непосредственный адрес
EA cp	JMP ptr16:32	18	31	pm=45+m			на шлюз вызова, привилегии не изменяются
EA cp	JMP ptr16:32	19+ts	42+ts	ts			через TSS

EA cp	JMP ptr16:32	20+ts	43+ts	ts	через шлюз задачи
FF /5	JMP m16:32	4	13,	43+m,	косвенный межсегментный
			pm=18	pm=31+m	
FF /5	JMP m16:32	18	31	pm=49+m	на шлюз вызова, привилегии не изменяются
FF /5	JMP m16:32	19+ts	41+ts	5+ts	через TSS
FF /5	JMP m16:32	20+ts	42+ts	5+ts	через шлюз задачи

* При выполнении следующей команды добавляется по одному такту на каждый байт команды (только для 80286).

Операция

```
IF (команда относительного перехода) (* т.е. операнды rel8, rel16, или rel32 *)
THEN
  EIP = EIP + rel8/16/32;
  IF OperandSize = 16
  THEN EIP = EIP AND 0000FFFFh;
  FI;
FI;
IF (команда ближнего косвенного перехода) (* т.е операнды r/m16 или r/m32 *)
THEN
  IF OperandSize = 16
  THEN EIP = [r/m16] AND 0000FFFFh;
  ELSE EIP = [r/m32]; (* OperandSize = 32 *)
  FI;
FI;
IF (PE = 0 OR (PE = 1 AND VM = 1)) (* реальный режим или режим V86 *)
   AND (команда дальнего перехода)
     (* т.е операнды m16:16, m16:32, ptr16:16, ptr16:32 *)
THEN GO TO REAL-OR-V86-MODE;
  IF (операнды m16:16 или m16:32)
  THEN (* косвенный *)
    IF OperandSize= 16
    THEN
      CS:IP = [m16:16];
      EIP = EIP AND 0000FFFFh; (* очищает старшие 16 бит *)
    ELSE (* OperandSize = 32 *)
      CS:EIP = [m16:32];
    FI;
  FI;
  IF (операнды ptr16:16 или ptr 16:32)
  THEN
    IF OperandSize = 16
    THEN
    CS:IP = ptr16:16;
      EIP = EIP AND 0000FFFFh; (* очищает старшие 16 бит *)
    ELSE (* OperandSize = 32 *)
      CS:IP = ptr16:32;
    FI;
  FI;
FI;
IF (PE = 1 AND VM = 0) (* Защищенный режим, не V86 режим *)
   AND (команда дальнего перехода)
THEN
  IF (операнд m16:16 или m16:32)
  THEN (* косвенный *)
    Проверка доступности двойного слова по EA,
      #GP(0) или #SS(0) в случае превышения пределов;
  FI;
  Селектор назначения не нулевой, иначе #GP(0);
  Индекс селектора назначения должен попадать в пределы таблицы
       дескрипторов, иначе #GP(селектор);
  В зависимости от AR байта дескриптора назначения:
    GO TO CONFORMING-CODE-SEGMENT; (* согласованный кодовый
                                      сегмент *)
```

```
    GO TO NONCONFORMING-CODE-SEGMENT; (* несогласованный кодовый
                                          сегмент *)
    GO TO CALL-GATE; (* шлюз вызова *)
    GO TO TASK-GATE; (* шлюз задачи *)
    GO TO TASK-STATE-SEGMENT; (* TSS - сегмент состояния задачи *)
  ELSE #GP(CS селектор); (* неверный AR байт в дескрипторе *)
FI;

CONFORMING-CODE-SEGMENT:
  IF DPL > CPL THEN #GP(CS селектор); FI;
  Сегмент должен присутствовать, иначе #NP(CS селектор);
  Указатель инструкции должен быть в пределах кодового сегмента,
     иначе #GP(0);
  IF OperandSize = 32
  THEN Загрузить CS:EIP указателем назначения;
  ELSE Загрузить CS:IP указателем назначения;
  FI;
  Загрузить дескриптор кодового сегмента в регистр CS;
END;

NONCONFORMING-CODE-SEGMENT:
  IF (RPL селектора назначения) > CPL THEN #GP(CS селектор); FI;
  IF DPL ≠ CPL THEN #GP(CS селектор);
  Сегмент должен присутствовать, иначе #NP(CS селектор);
  Указатель инструкции должен быть в пределах кодового сегмента,
     иначе #GP(0);
  IF OperandSize = 32
  THEN Загрузить CS:EIP указателем назначения;
  ELSE Загрузить CS:IP указателем назначения;
  FI;
  Загрузить дескриптор кодового сегмента в регистр CS;
  Установить RPL в CS в значение CPL;
END;

CALL-GATE:
  DPL шлюза вызова должен быть ≥ CPL, иначе #GP(селектор шлюза вызова);
  DPL шлюза вызова должен быть ≥ RPL, иначе #GP(селектор шлюза вызова);
  Шлюз вызова должен присутствовать, иначе #NP(селектор шлюза вызова);
  Проверка селектора кодового сегмента в дескрипторе шлюза вызова:
    Селектор должен быть не нулевым, иначе #GP(0);
    Индекс селектора должен попадать в пределы таблицы дескрипторов,
      иначе #GP(CS селектор);
    AR байт выбранного дескриптора задает кодовый сегмент,
       иначе #GP(CS селектор);
    IF (несогласованный кодовый сегмент)
    THEN DPL выбранного дескриптора должен быть = CPL,
       иначе #GP(CS селектор);
    FI;
    IF (согласованный кодовый сегмент)
    THEN DPL выбранного дескриптора должен быть ≤ CPL,
       иначе #GP(CS селектор);
    FI;
    Сегмент должен присутствовать, иначе #NP(CS селектор);
    Указатель инструкции должен быть в пределах кодового сегмента,
       иначе #GP(0);
  IF OperandSize = 32
  THEN Загрузить CS:EIP из шлюза вызова;
  ELSE Загрузить CS:IP из шлюза вызова;;
  FI;
  Загрузить дескриптор кодового сегмента в регистр CS;
  Установить RPL в CS в значение CPL;
END;

TASK-GATE:
  DPL шлюза задачи должен быть ≥ CPL, иначе #TS(селектор шлюза);
```

```
    DPL шлюза задачи должен быть ≥ RPL, иначе #TS(селектор шлюза);
    Шлюз задачи должен присутствовать, иначе #NP(селектор шлюза);
    Проверка селектора TSS в дескрипторе шлюза задачи:
      Селектор должен задавать GDT (бит TI=0), иначе #TS(TSS селектор);
      Индекс селектора должен попадать в пределы таблицы GDT,
        иначе #TS(TSS селектор);
      AR байт TSS дескриптора должен задавать свободный TSS (младшие
        биты 00001), иначе #TS(TSS селектор);
      TSS должен присутствовать, иначе #NP(TSS селектор);
    SWITCH-TASKS (без вложения) в TSS;
    IP должен быть в пределах кодового сегмента, иначе #TS(0);
  END;

  TASK-STATE-SEGMENT:
    TSS DPL должен быть ≥ CPL, иначе #TS(TSS селектор);
    TSS DPL должен быть ≥ RPL, иначе #TS(TSS селектор);
    AR байт TSS дескриптора должен задавать свободный TSS,
        иначе #TS(TSS селектор);
    TSS должен присутствовать, иначе #NP(TSS селектор);
    SWITCH-TASKS (с вложением) в TSS;
    IP должен быть в пределах кодового сегмента, иначе #TS(0);
  END;
```

Описание

Команда JMP передает управление в другую точку потока команд без записи информации о возврате.

Переходы по назначению типа r/m16, r/m32, rel16 и rel32 являются ближними переходами и не приводят к изменению значения сегментного регистра.

Формы команды JMP rel16 и JMP rel32 прибавляют смещение к адресу команды следующей за JMP, чтобы определить назначение. Форма rel16 используется, когда атрибут размера операнда команды равен 16 бит (только 16 битный атрибут размера сегмента). rel32 используется, когда атрибут размера операнда равен 32 бита (только 32 битный размер сегмента). Результат сохраняется в 32 битном регистре EIP. В случае rel16, верхние 16 бит регистра EIP очищаются, в результате чего получается смещение, значение которого не превосходит границы 16 бит.

Формы JMP r/m16 и JMP r/m32 указывают на регистр или местоположение в памяти, откуда выбирается абсолютное смещение назначения. Смещение выбирается из r/m равного 32 битам, в случае 32 битного атрибута размера операнда (r/m32), и из r/m равного 16 битам, в случае 16 битного атрибута размера операнда (r/m16).

Формы команды JMP ptr16:16 и JMP ptr16:32 используют четырех или шестибайтный операнд в качестве длинного указателя на назначение. Формы JMP m16:16 и JMP m16:32 выбирают длинный указатель из указанного местоположения в памяти (разрешены все режимы адресации). В реальном и виртуальном режимах длинный указатель обеспечивает 16 бит для

регистра CS и 16 или 32 бита для регистра EIP (в зависимости от атрибута размера операнда). В защищенном режиме обе формы с длинным указателем согласуются с байтом прав доступа (AR) дескриптора, указанного при помощи индекса селекторной частью длинного указателя. В зависимости от значения байта AR, переход осуществляется с применением одного из следующих типов управляющих преобразований:

- переход в кодовый сегмент на том же уровне привилегий;
- переключение задач.

Особые ситуации защищенного режима

Дальние переходы: #GP, #NP, #SS и #TS, как указано в пункте "Операция".

Ближние прямые переходы: #GP(0), если местоположение процедуры вне пределов кодового сегмента; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Ближние косвенные переходы: #GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

LAHF Загрузить флаги в регистр AH

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
9F	LAHF	2	3	2	2	4	lahf

Операция

AH = SF:ZF:xx:AF:xx:PF:xx:CF;

Описание

Команда LAHF перемещает младший байт слова флагов в регистр AH. Биты от наиболее значащего к наименее значащему таковы: знак результата, признак нуля, неопределен, вспомогательный перенос, неопределен, четность, неопределен, перенос.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

LAR Загрузить байт прав доступа

O D I T S Z A P C
*

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
0F 02 /r	LAR r16,r/m16	8	11/11	pm=15/16	14/16	lar ax,dx
0F 02 /r	LAR r32,r/m32	8	11/11	pm=15/16		lar eax,edx

Описание

Команда LAR сохраняет частично замаскированную форму второго двойного слова дескриптора (не маскируется байт прав доступа AR) для селектора источника, если селектор видим при текущем уровне привилегий (измененном при помощи RPL селектора), и если дескриптор имеет допустимый тип.

Если указан 32 битный размер операнда, то 32 битный регистр назначения загружается старшим двойным словом дескриптора, маскированным 00FxFF00h, и устанавливается флаг ZF. x обозначает, что четыре бита, соответствующие четырем старшим битам предела, не определены в значении, загружаемом командой LAR.

Если указан 16 битный размер операнда, то 16 битный регистр назначения загружается младшими 16 битами старшего двойного слова дескриптора, маскированными FF00, и устанавливается флаг ZF.

Если селектор невидим или неверного типа, то флаг ZF сбрасывается.

Все дескрипторы сегментов кода и данных допустимы для команды LAR.

В следующей таблице даны допустимые для команды LAR типы дескрипторов специальных сегментов и шлюзов:

Тип (Type)	Имя	Допустим/ недопустим
0000	Зарезервировано	недопустим
0001	Свободный 16 битный TSS	допустим
0010	Локальная таблица дескрипторов (LDT)	допустим
0011	Занятый 16 битный TSS	допустим
0100	16 битный шлюз вызова	допустим
0101	Шлюз задачи	допустим
0110	16 битный шлюз прерывания	недопустим
0111	16 битный шлюз ловушки	недопустим
1000	Зарезервировано	недопустим
1001	Свободный 32 битный TSS	допустим
1010	Зарезервировано	недопустим
1011	Занятый 32 битный TSS	допустим
1100	32 битный шлюз вызова	допустим
1101	Зарезервировано	недопустим
1110	32 битный шлюз прерывания	недопустим
1111	32 битный шлюз ловушки	недопустим

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

LDS Загрузить полный указатель
LES
LFS
LGS
LSS

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
C5 /r	LDS r16,m16:16	4	6/12	7,pm=22	7,pm=21	16+EA	lds ax,wordX
C5 /r	LDS r32,m16:32	4	6/12	7,pm=22			lds eax,dwordZ
0F B2 /r	LSS r16,m16:16	4,pm=8	6/12	7,pm=22			lss di,vector
0F B2 /r	LSS r32,m16:32	4,pm=8	6/12	7,pm=22			lss esi,dwordBag
C4 /r	LES r16,m16:16	4	6/12	7,pm=22	7,pm=21	16+EA	les ax,wordX
C4 /r	LES r32,m16:32	4	6/12	7,pm=22			les eax,dwordZ
0F B4 /r	LFS r16,m16:16	4	6/12	7,pm=22			lfs ax,wordX
0F B4 /r	LFS r32,m16:32	4	6/12	7,pm=22			lfs eax,dwordZ
0F B5 /r	LGS r16,m16:16	4	6/12	7,pm=22			lgs ax,wordX
0F B5 /r	LGS r32,m16:32	4	6/12	7,pm=22			lgs eax,dwordZ

Операция

```
CASE (команда) OF
  LSS: Sreg is SS; (* загружает регистр SS *)
  LDS: Sreg is DS; (* загружает регистр DS *)
```

```
  LFS: Sreg is FS; (* загружает регистр FS *)
  LGS: Sreg is GS; (* загружает регистр GS *)
ESAC;
IF OperandSize = 16
THEN
  r16 = [Effective Address]; (* перенос 16 бит *)
  Sreg = [Effective Address + 2]; (* перенос 16 бит *)
  (* В защищенном режиме загружается дескриптор в скрытую часть
      сегментного регистра *)
ELSE
  r32 = [Effective Address]; (* перенос 32 бит *)
  Sreg = [Effective Address + 4]; (* перенос 16 бит *)
  (* В защищенном режиме загружается дескриптор в скрытую часть сегментно-
      го регистра *)
FI;
```

Описание

Команды LGS, LSS, LDS, LES и LFS читают полный указатель из памяти и сохраняют его в выбранной паре - сегментный регистр:регистр. Полный указатель загружает 16 бит в сегментный регистр SS, DS, ES, FS или GS. В другой регистр загружается 32 бита, если атрибут размера операнда равен 32 бита, и 16 бит, если атрибут размера операнда равен 16 бит. Другой 16 или 32 разрядный регистр определяется r16 или r32 операндом команды.

Когда присвоение производится одному из сегментных регистров, то дескриптор также загружается в скрытую часть этого сегментного регистра. Значение этого дескриптора выбирается из таблицы дескрипторов по значению селектора, загруженному в сегментный регистр.

Нулевой селектор (значение 0000 - 0003) может быть загружен в DS, ES, FS или GS регистры, не вызывая особых ситуаций защиты. (Любая последующая ссылка в сегмент, чей соответствующий сегментный регистр загружается нулевым селектором, вызывает исключение #GP(0). Никакого обращения в сегмент не происходит.)

Ниже представлен листинг проверок и действий, предпринимаемых при загрузке сегментного регистра в защищенном режиме:

```
IF (загружается SS)
THEN
  IF (селектор не нулевой) THEN #GP(0); FI;
  Индекс селектора должен попадать в пределы таблицы дескрипторов,
    иначе #GP(селектор);
  Поле RPL селектора должно быть равно CPL, иначе #GP(селектор);
  AR байт должен задавать сегмент данных, доступный для записи,
    иначе #GP(селектор);
  Поле DPL AR байта должно быть равно CPL, иначе #GP(селектор);
  Сегмент должен быть помечен, как присутствующий, иначе #SS(селектор);
  Загрузить SS селектором;
  Загрузить SS дескриптором;
FI;
IF (загружается DS, ES, FS или GS не нулевым селектором)
THEN
  Индекс селектора должен попадать в пределы таблицы дескрипторов,
```

```
    иначе #GP(селектор);
  AR байт должен задавать сегмент данных доступный для записи,
     иначе #GP(селектор);
  IF (данные или несогласованный код)
  THEN RPL и CPL должны быть меньше или равны DPL в AR байте,
     иначе #GP(селектор);
  FI;
  Сегмент должен быть помечен, как присутствующий, иначе #NP(селектор);
  Загрузить селектор в сегментный регистр;
  Загрузить дескриптор в сегментный регистр;
FI;
IF (загружается DS, ES, FS или GS нулевым селектором)
THEN
  Загрузить селектор в сегментный регистр;
  Очистить бит достоверности дескриптора (descriptor valid bit);
FI;
```

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; второй операнд должен быть операндом в памяти, а не регистром - если регистр, то #UD; #GP(0), если нулевой селектор загружается в SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 6, если второй операнд - регистр; Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

LEA Загрузить эффективный адрес

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
8D /r	LEA r16,m	1	1	2	3	2+EA	lea ax,driv
8D /r	LEA r32,m	1	1	2			lea edx,inv
8D /r	LEA r16,m	1	1	2			lea ax,driv
8D /r	LEA r32,m	1	1	2			lea eax,priv

Операция

```
IF OperandSize = 16 AND AddressSize = 16
THEN r16 = Addr(m);
ELSE
  IF OperandSize = 16 AND AddressSize = 32
  THEN r16 = Truncate_to_16bits(Addr(m)); (* 32 битный адрес *)
  ELSE
    IF OperandSize = 32 AND AddressSize = 16
    THEN
      r32 = Truncate_to_16bits(Addr(m)) and zero extend;
    ELSE
      IF OperandSize = 32 AND AddressSize = 32
      THEN r32 = Addr(m);
      FI;
    FI;
  FI;
FI;
```

Описание

Команда LEA вычисляет эффективный адрес (смещение) и сохраняет его в указанном регистре. Атрибут размера операнда команды определяется выбранным регистром. Атрибут размера адреса определяется атрибутом кодового сегмента. Атрибуты размера адреса и размера операнда воздействуют на операцию, выполняемую командой LEA, следующим образом:

Размер операнда	Размер адреса	Выполняемая операция
16	16	Вычисляется 16 битный эффективный адрес и сохраняется в 16 битном регистре назначения
16	32	Вычисляется 32 битный эффективный адрес его младшие 16 бит сохраняются в 16 битном регистре назначения

32	16	Вычисляется 16 битный эффективный адрес, затем знакорасширяется и сохраняется в 32 битном регистре назначения
32	32	Вычисляется 32 битный эффективный адрес и сохраняется в 32 битном регистре назначения

Особые ситуации защищенного режима

#UD, если второй операнд - регистр.

Особые ситуации режима реальной адресации

Прерывание 6, если второй операнд - регистр.

Особые ситуации режима V86

То же, что и в режиме реальной адресации.

Замечание

Различные ассемблеры могут использовать различные алгоритмы, основанные на атрибуте размера и символической ссылке второго операнда.

LEAVE Выход из процедуры высокого уровня

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
C9	LEAVE	3	5	4	5	leave
C9	LEAVE	3	5	4		leave

Операция

```
IF StackAddress = 16
THEN SP = BP;
ELSE (* StackAddrSize = 32 *)
  ESP = EBP;
FI;
```

```
IF OperandSize = 32
THEN BP = Pop();
ELSE (* OperandSize = 32 *)
  EBP = Pop();
FI;
```

Описание

Команда LEAVE производит действия обратные команде ENTER. Копируя указатель кадра в указатель стека, команда LEAVE освобождает пространство стека, использовавшееся для локальных переменных. Старый указатель кадра выбирается из стека в регистр BP или EBP, восстанавливая кадр вызывающей процедуры. Последующая команда RET nn удаляет любые аргументы, помещенные в стек входной процедуры.

Особые ситуации защищенного режима

#SS(0), если регистр BP не указывает на местоположение внутри пределов текущего сегмента стека.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда будет лежать вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

То же, что и в режиме реальной адресации.

LES Загрузить полный указатель

Смотрите LDS.

LFS Загрузить полный указатель

Смотрите LDS.

LGDT Загрузить регистр глобальной таблицы дескрипторов (GDTR)
LIDT Загрузить регистр таблицы дескрипторов прерываний (IDTR)

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
0F 01 /2	LGDT m16&32	6	11	11	11	lgdt globalT
0F 01 /3	LIDT m16&32	6	11	11	12	lidt globalT

Операция

```
IF (команда типа LIDT)
THEN
  IF OperandSize = 16
  THEN IDTR.Limit:Base = m16:24 (* загружается 24 бита  базы *)
  ELSE IDTR.Limit:Base = m16:32
  FI;
ELSE (* команда типа LGDT *)
  IF OperandSize = 16
  THEN GDTR.Limit:Base = m16:24 (* загружается 24 бита  базы *)
  ELSE GDTR.Limit:Base = m16:32
 FI;
FI;
```

Описание

Команды LGDT и LIDT загружают линейный базовый адрес и значение предела из шестибайтного операнда в памяти в GDTR или IDTR соответственно. Если используется 16 битный операнд с командами LGDT и LIDT, то регистр загружается 16 битным пределом и 24 битной базой, а старшие 8 бит шестибайтного операнда не используются. Если используется 32 битный операнд, то загружаются 16 битный предел и 32 битная база, а старшие 8 бит шестибайтного операнда используются как старшие биты базового адреса.

Команды SGDT и SIDT всегда сохраняют все 48 бит шестибайтного операнда. У 16 битных процессоров старшие 8 бит не определены после выполнения команды SGDT или SIDT, для 32 битных процессоров в старшие 8 бит записываются старшие 8 бит адреса для размеров операндов 16

и 32 бита. Если команда LGDT или LIDT используется с 16 битным операндом для загрузки регистра, сохраненного командой SGDT или SIDT, то старшие 8 бит сохраняются как нули.

Команды LGDT и LIDT используются в системном программном обеспечении, они не используются в прикладных программах. Только эти команды непосредственно загружают линейный адрес (т.е. не адрес относительно сегмента) в защищенном режиме.

Особые ситуации защищенного режима

#GP(0), если текущий уровень привилегий не равен 0; #UD, если операнд-источник - регистр; #GP(0), если недопустимый эффективный адрес операнда в памяти сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда будет лежать вне пространства эффективных адресов от 0 до 0FFFFh; прерывание 6, если второй операнд - регистр.

Замечание

Эти команды допустимы в реальном режиме, для обеспечения начальной загрузки при инициализации защищенного режима.

Особые ситуации режима V86

То же, что и в режиме реальной адресации; #PF(код ошибки), страничная ошибка.

LGS Загрузить полный указатель

Смотрите LDS.

LLDT Загрузка регистра локальной таблицы дескрипторов (LDTR)

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
0F 00 /2	LLDT r/m16	9	11/11	20	11/19	lldt [di]

Операция

LDTR = SRC;

Описание

Команда LLDT загружает регистр локальной таблицы дескрипторов (LDTR). Двухбайтный операнд (в памяти или регистр) команды LLDT должен содержать селектор глобальной таблицы дескрипторов (GDT). Элемент GDT должен быть дескриптором локальной таблицы дескрипторов. Если так оно и есть, то LDTR загружается этим элементом. Дескрипторные регистры DS, ES, SS, FS, GS и CS не изменяются. Поле LDT в сегменте состояния задачи (TSS) не изменяется.

Операнд-селектор может быть нулем, если это так, то LDTR помечается как недопустимый. Все дескрипторные ссылки (исключая ссылки через команды LAR, VERR, VERW или LSL) вызывают ошибку #GP.

Команда LLDT используется в системном программном обеспечении, она не используется в прикладных программах.

Особые ситуации защищенного режима

#GP(0), если текущий уровень привилегий не равен 0; #GP(селектор), если операнд-селектор не указывает на глобальную таблицу дескрипторов, или запись в GDT - не дескриптор локальной таблицы дескрипторов; #NP(селектор), если дескриптор LDT не присутствует; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка.

Особые ситуации режима реальной адресации

Прерывание 6, команда LLDT не распознается в реальном режиме.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации (из-за того, что команда не распознается, она не выполняется и не происходит обращения к памяти).

Замечание

Атрибут размера операнда не влияет на эту команду.

LIDT Загрузить регистр таблицы дескрипторов прерываний (IDTR)

Смотрите LGDT.

LMSW Загрузить слово состояния машины (MSW)

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
0F 01 /6	LMSW r/m16	8	13/13	10/13	3/6	lmsw cx

Операция

```
MSW = r/m16; (* 16 бит помещается в слово состояния машины (MSW) *)
```

Описание

Команда LMSW загружает слово состояния машины (часть регистра CR0) из операнда источника. Эта команда может быть использована для переключения в защищенный режим, в этом случае за ней должен следо-

вать межсегментный переход, чтобы очистить очередь команд. Команда LMSW не переключает обратно в реальный режим.

Команда LMSW используется в системном программном обеспечении, она не используется в прикладных программах.

Особые ситуации защищенного режима

#GP(0), если текущий уровень привилегий не равен 0; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS,DS,ES,FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации.

Замечание

Атрибут размера операнда не воздействует на эту команду. Эта команда обеспечивает совместимость с процессором Intel286, программы для процессоров Intel386, Intel486 и Pentium должны использовать вместо нее команду MOV CR0,... . Команда LMSW не изменяет биты PG, ET или NE и не может быть использована для очистки бита PE.

LOCK Префикс выдачи сигнала блокировки шины (LOCK#)

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
F0	LOCK	1	1	0	0	2	lock

Описание

Префикс выдачи сигнала LOCK# (префикс блокировки шины) заставляет процессор установить сигнал LOCK# во время выполнения следующей за ним команды. В многопроцессорной среде этот сигнал может быть использован для гарантии того, что процессор имеет исключительный доступ к любой памяти с распределенным доступом, пока установлен LOCK#. Последовательность чтение-модификация-запись обычно используется для осуществления операции "проверить-и-установить" в команде BTS.

Префикс LOCK функционирует только со следующими командами (Pentium, Intel486, Intel386 - для Intel286 и 8086 допустимы любые команды):

BTS, BTR, BTC	mem, reg/imm
XCHG	reg, mem
XCHG	mem, reg
ADD, OR, ADC, SBB, AND, SUB, XOR	mem, reg/imm
NOT, NEG, INC, DEC	mem
CMPXCHG, XADD	

Будет сгенерирована ловушка "неопределенный код операции", если префикс используется с любой другой командой (Pentium, Intel486, Intel386).

Команда XCHG всегда устанавливает LOCK#, несмотря на присутствие или отсутствие префикса LOCK.

Целостность префикса LOCK не подвергается воздействию выравниванием полей памяти. Блокировка памяти отслеживается для произвольных невыровненных полей.

Особые ситуации защищенного режима

#UD, если префикс LOCK используется с командой не из выше приведенного списка, другие исключения могут генерироваться последующей (блокированной) командой.

Особые ситуации режима реальной адресации

Прерывание 6, если префикс LOCK используется с командой не из выше приведенного списка, другие исключения могут генерироваться последующей (блокированной) командой.

Особые ситуации режима V86

#UD, если префикс LOCK используется с командой не из выше приведенного списка, другие исключения могут генерироваться последующей (блокированной) командой.

LODS Загрузить строковый операнд
LODSB
LODSW
LODSD

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
AC	LODS m8	2	5	5	5	12	lodsb
AD	LODS m16	2	5	5	5	12	lodsw
AD	LODS m32	2	5	5			lodsd
AC	LODSB	2	5	5	5	12	lodsb
AD	LODSW	2	5	5	5	12	lodsw
AD	LODSD	2	5	5			lodsd

Операция

```
IF AddressSize = 16
THEN использовать SI для sourse-index;
ELSE (* AddressSise = 32 *) использовать ESI для sourse-index;
FI;
IF (команда байтового типа)
THEN
  AL = [sourse-index]; (* загрузить байт *)
  IF DF = 0 THEN IncDec = 1; ELSE IndDec = -1; FI;
ELSE
  IF OperandSize = 16
  THEN
    AX = [sourse-index]; (* загрузить слово *)
    IF DF = 0 THEN IncDec = 2; ELSE IndDec = -2; FI;
  ELSE (* OperandSize = 32 *)
    EAX = [sourse-index]; (* загрузить двойное слово *)
    IF DF = 0 THEN IncDec = 4; ELSE IndDec = -4; FI;
  FI;
FI;
sourse-index = sourse-index + IncDec;
```

Описание

Команда LODS загружает в регистр AL, AX или EAX байт, слово или двойное слово из ячейки памяти, указываемой при помощи индексного регистра-источника. После передачи индексный регистр-источник автома-

тически изменяется. Если флаг DF равен 0 (была выполнена команда CLD), то индекс-источник увеличивается, если флаг DF равен 1 (была выполнена команда STD), то индекс-источник уменьшается. Увеличение или уменьшение происходит на 1, если загружается байт, на 2, если загружается слово, на 4, если загружается двойное слово.

Если атрибут размера адреса команды равен 16 бит, то для индексного регистра источника используется регистр SI, иначе, если атрибут размера адреса команды равен 32 бита, то для индексного регистра источника используется регистр ESI. Адрес данных источника однозначно определяется содержимым регистра SI или ESI. Загружайте корректное значение индекса в регистр SI перед выполнением команды LODS. Команды LODSB, LODSW и LODSD - синонимы команды LODS для байта, слова и двойного слова.

Команде LODS может предшествовать префикс REP, тем не менее, более типично использование команды LODS внутри LOOP-конструкций, потому что обычно необходима дальнейшая обработка данных в регистрах AL, AX или EAX.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

LOOP Управление циклом со счетчиком CX

LOOPcond

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
E2 cb	LOOP rel8	5,6	2,6	11+m	8,4	17,5	loop met2
E1 cb	LOOPE rel8	7,8	9,6	11+m	8,4	18,6	loope met2
E1 cb	LOOPZ rel8	7,8	9,6	11+m	8,4	18,6	loopz met2
E0 cb	LOOPNE rel8	7,8	9,6	11+m	8,4	19,5	loopne met2
E0 cb	LOOPNZ rel8	7,8	9,6	11+m	8,4	19,5	loopnz met2

Операция

```
IF Address = 16 THEN CountReg is CX; ELSE CountReg is ECX; FI;
CountReg = CountReg - 1;
IF (команда не LOOP)
THEN
  IF ( команда LOOPE) OR (команда LOOPZ))
  THEN BranchCond = (ZF = 1) AND (CountReg ≠ 0)
  FI;
  IF (команда LOOPNE) OR (команда LOOPNZ))
  THEN BranchCond = (ZF = 0) AND (CountReg ≠ 0)
  FI;
FI;
IF BranchCond
THEN
  IF OperandSize = 16
  THEN IP = IP + SignExtend(rel8);
  ELSE (* OperandSize = 32 *)
    EIP =EIP + SignExtend(rel8);
  FI;
FI;
```

Описание

Команда LOOP уменьшает регистр счетчик без изменения какого-либо из флагов. Затем проверяются условия, характерные для конкретной формы используемой команды LOOP. Если условия удовлетворяются, то происходит короткий переход на метку, заданную операндом команды LOOP. Если атрибут размера адреса равен 16 бит, то в качестве счетчика используется регистр CX, иначе - ECX. Операнд команды LOOP должен быть в диапазоне от -128 до +127 байт, относительно первого байта следующей команды.

Команды LOOP обеспечивают циклический контроль и объединяют индексное управление циклом с условным ветвлением. Используйте ко-

манду LOOP так: загрузите беззнаковое значение счетчика итераций в регистр счетчик, затем запрограммируйте команду LOOP в конце серии итерационных команд. Назначение команды LOOP - метка, которая указывает на начало итерации.

Особые ситуации защищенного режима

#GP(0), если смещение точки, к которой осуществляется переход, находится вне пределов текущего кодового сегмента.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

Замечание

Безусловная команда LOOP выполняется дольше, чем последовательность из двух команд, которые уменьшают счетчик, а затем осуществляют переход, если счетчик не равен нулю.

LSL Загрузить предел сегмента

O D I T S Z A P C
*

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
0F 03 /r	LSL r16,r/m16	8	10	pm=20/21	14/16	lsl ax,bx
0F 03 /r	LSL r32,r/m32	8	10	pm=20/21		lsl eax,ebx
0F 03 /r	LSL r16,r/m16	8	10	pm=25/26	14/16	lsl ax,bx
0F 03 /r	LSL r32,r/m32	8	10	pm=26/26		lsl eax,ebx

Описание

Команда LSL загружает регистр преобразованным значением предела сегмента и устанавливает флаг ZF при условии: селектор-источник видим при текущем уровне привилегий и RPL; селектор-источник попадает в пределы таблицы дескрипторов; дескриптор имеет допустимый для команды LSL тип (смотрите таблицу ниже). Иначе, флаг ZF очищается, а ре-

гистр-назначение остается неизменным. Если дескриптор имеет предел выраженный в страницах, тогда команда LSL переведет его к байтовому выражению перед загрузкой в регистр назначения (производится сдвиг влево на 12 бит двадцатиразрядного предела из дескриптора, затем OR 00000FFFFh).

32 битные формы команды LSL сохраняют 32 битный предел в 32 битный регистр назначения. Для размеров операнда 16 бит предел приводится к форме допустимого 32 битного предела (старшие 16 бит теряются, а младшие 16 бит загружаются в регистр назначения).

Ниже, в таблице приведены допустимые для команды LSL типы дескрипторов специальных сегментов и шлюзов.

Тип (TYPE)	Имя	Допустим/ недопустим
0000	Зарезервировано	недопустим
0001	Свободный 16 битный TSS	допустим
0010	Локальная таблица дескрипторов (LDT)	допустим
0011	Занятый 16 битный TSS	допустим
0100	16 битный шлюз вызова	недопустим
0101	Шлюз задачи	недопустим
0110	16 битный шлюз прерывания	недопустим
0111	16 битный шлюз ловушки	недопустим
1000	Зарезервировано	недопустим
1001	Свободный 32 битный TSS	допустим
1010	Зарезервировано	недопустим
1011	Занятый 32 битный TSS	допустим
1100	32 битный шлюз вызова	недопустим
1101	Зарезервировано	недопустим
1110	32 битный шлюз прерывания	недопустим
1111	32 битный шлюз ловушки	недопустим

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 6, команда LSL не распознается в режиме реальной адресации.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации (так как команда не распознается, она не выполнится и не произойдет обращения к памяти).

LSS Загрузить полный указатель

Смотрите LDS.

LTR Загрузить регистр задачи

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
0F 00 /3	LTR r/m16	10	20/20	pm=23/27	17/19	ltr di

Описание

Команда LTR загружает регистр задачи селектором из регистра источника или из ячейки памяти, указанной операндом. Загруженный TSS помечается как занятый. Переключения задачи не происходит.

Команда LTR обычно используется только в системном программном обеспечении.

Особые ситуации защищенного режима

#GP(0), если недопустимое значение эффективного адреса в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимое значение адреса в сегменте SS; #GP(0), если текущий уровень привилегий не 0; #GP(селектор), если объект, указываемый селектором источником не является TSS или уже занят; #NP(селектор), если TSS помечен как неприсутствующий; #PF(код ошибки), для страничной ошибки.

Особые ситуации режима реальной адресации

Прерывание 6, команда LTR не распознается в режиме реальной адресации.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации.

Замечание

На эту команду атрибут размера операнда не воздействует.

MOV Переслать данные

O D I T S Z A P C

Код операции	Команда	Pentium	486	Такты 386	286	86	Пример
88 /r	MOV r/m8,r8	1	1	2/2	2/3	2/9+EA	mov [di],dl
89 /r	MOV r/m16,r16	1	1	2/2	2/3	2/9+EA	mov [si],cx
89 /r	MOV r/m32,r32	1	1	2/2			mov [ebx],edi
8A /r	MOV r8,r/m8	1	1	2/4	2/5	2/8+EA	mov dh,cl
8B /r	MOV r16,r/m16	1	1	2/4	2/5	2/8+EA	mov dx,[di]
8B /r	MOV r32,r/m32	1	1	2/4			mov ebx,[edi]
8C /r	MOV r/m16,Sreg*	1	3/3	2/2	2/3	2/9+EA	mov dx,ds
8E /r	MOV Sreg,r/m16	2/3**	3/9	2/5, pm=1/198	2/5, pm=17/19	2/8+EA	mov ds,si
A0	MOV AL,moffs8	1	1	4	5	10	mov al,byteC
A1	MOV AX,moffs16	1	1	4	5	10	mov ax,wordR
A1	MOV EAX,moffs32	1	1	4			mov eax,dwd
A2	MOV moffs8,AL	1	1	4	3	10	mov byteC,al
A3	MOV moffs16,AX	1	1	4	3	10	mov wordR,ax
A3	MOV moffs32,EAX	1	1	4			mov dwd,eax
B0+rb	MOV reg8,imm8	1	1	2	2	4	mov dl,9
B8+rw	MOV reg16,imm16	1	1	2	2	4	mov cx,3212
B8+rd	MOV reg32,imm32	1	1	2			mov ebp,21367
C6 /0	MOV r/m8,imm8	1	1	2/2	2/3	4/10+EA	mov [di],2
C7 /0	MOV r/m16,imm16	1	1	2/2	2/3	4/10+EA	mov [di],2354
C7 /0	MOV r/m32,imm32	1	1	2/2			mov [esi],985

* В 32-битном режиме необходимо использование префикса переопределения размера операнда (байт со значением 67h предшествующий коду операции).
** В защищенном режиме команда MOV SS,r/m16/32 выполняется за 7 тактов.

Операция

DEST = SRC;

Описание

Команда MOV копирует второй операнд в первый операнд.

Если операнд назначение - сегментный регистр (DS, ES, SS и т.п.), тогда в скрытую часть этого регистра также загружаются данные из дескриптора. Эти данные для регистра извлекаются из элемента таблицы дескрипторов для данного селектора. Пустой селектор (значения 0000 - 0003) может быть загружен в регистры DS, ES, FS и GS не вызывая особой ситуации, тем не менее использование этих регистров вызывает особую ситуацию #GP(0), и никакого обращения к памяти не происходит.

Команда MOV в SS запрещает все прерывания до завершения выполнения следующей команды (которая может быть командой MOV в ESP).

Ниже представлен листинг проверок и действий, предпринимаемых при загрузке сегментного регистра в защищенном режиме:

```
IF (загружается SS)
THEN
  IF (селектор не нулевой) THEN #GP(0); FI;
  Индекс селектора должен попадать в пределы таблицы дескрипторов,
    иначе #GP(селектор);
  Поле RPL селектора должно быть равно CPL, иначе #GP(селектор);
  AR байт должен задавать сегмент данных, доступный для записи,
    иначе #GP(селектор);
  Поле DPL AR байта должно быть равно CPL, иначе #GP(селектор);
  Сегмент должен быть помечен, как присутствующий, иначе #SS(селектор);
  Загрузить SS селектором;
  Загрузить SS дескриптором;
FI;
IF (загружается DS, ES, FS или GS не нулевым селектором)
THEN
  Индекс селектора должен попадать в пределы таблицы дескрипторов,
    иначе #GP(селектор);
  AR байт должен задавать сегмент данных, или кодовый сегмент доступный
    для чтения, иначе #GP(селектор);
  IF (данные или несогласованный код)
  THEN RPL и CPL должны быть меньше или равны DPL в AR байте,
    иначе #GP(селектор);
  FI;
  Сегмент должен быть помечен, как присутствующий, иначе #NP(селектор);
  Загрузить селектор в сегментный регистр;
  Загрузить дескриптор в сегментный регистр;
FI;
IF (загружается DS, ES, FS или GS нулевым селектором)
THEN
  Загрузить селектор в сегментный регистр;
  Очистить бит правильности дескриптора (descriptor valid bit);
FI;
```

Особые ситуации защищенного режима

Если загружается сегментный регистр, #GP, #SS и #NP происходят в соответствии с алгоритмом, иначе #GP(0), если назначение находится в сегменте недоступном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная

ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

MOV Переслать в/из управляющего регистра

O D I T S Z A P C

Код операции	Команда	Такты			Пример
		Pentium	486	386	
0F 22 /r	MOV CR0,r32	22	16	10	mov cr0,eax
0F 22 /r	MOV CR2,r32	12	4	4	mov cr2,ebx
0F 22 /r	MOV CR3,r32	21	4	5	mov cr3,edx
0F 22 /r	MOV CR4,r32	14			mov cr4,ecx
0F 20 /r	MOV r32,CR0-CR4	4	4	6	mov eax,cr4

Операция

```
DEST = SRC;
```

Описание

Приведенная выше форма команды MOV сохраняет или загружает CR0, CR2, CR3, CR4 в или из регистра общего назначения.

С этими командами всегда используются 32 битные операнды независимо от атрибута размера операнда.

Особые ситуации защищенного режима

#GP(0), если текущий уровень привилегий не 0; #GP(0), если произведена попытка записать 1 в любой из зарезервированных битов регистра CR4.

Особые ситуации режима реальной адресации

Прерывание 13, если произведена попытка записать 1 в любой из зарезервированных битов регистра CR4.

Особые ситуации режима V86

#GP(0), при попытке выполнить команду.

Замечание

Поле reg внутри ModR/M байта указывает, какой из специальных регистров каждой категории привлекается. Два бита поля mod всегда равны 11. Поле r/m указывает на привлекаемый регистр общего назначения.

Всегда устанавливайте зарезервированные и неописанные биты управляющих регистров в прочитанное значение.

MOV Переслать в/из отладочного регистра

O D I T S Z A P C

Код операции	Команда	Такты			Пример
		Pentium	486	386	
0F 21 /r	MOV r32,DR0–DR3	11	10	22	mov eax,dr0
0F 21 /r	MOV r32,DR4–DR5	12			mov eax,dr5
0F 21 /r	MOV r32,DR6–DR7	11	10	14	mov eax,dr6
0F 23 /r	MOV DR0–DR3,r32	11	11	22	mov dr0,eax
0F 23 /r	MOV DR4–DR5,r32	12			mov dr5,eax
0F 23 /r	MOV DR6–DR7,r32	11	11	16	mov dr6,eax

Операция

```
IF (DE = 1) and (SRC or DEST = DR4 or DR5))
THEN #UD;
ELSE DEST = SRC;
FI;
```

Описание

Приведенная выше форма команды MOV сохраняет или загружает отладочные регистры DR0, DR1, DR2, DR3, DR6 и DR7 в или из регистра общего назначения. С этими командами всегда используются 32 битные операнды независимо от атрибута размера операнда.

Для процессора Pentium, когда бит DE (расширенная отладка) в CR4 очищен, команды MOV, использующие отладочные регистры, действуют аналогично командам процессоров Intel386 и Intel486. Ссылки в DR4 и DR5 направляются к DR6 и DR7 соответственно. Когда бит DE в CR4 установлен, попытки выполнить команду MOV, использующую DR4 и DR5, дают в результате особую ситуацию "неопределенный код операции" (#UD).

Особые ситуации защищенного режима

#GP(0), если текущий уровень привилегий не 0; #UD, если бит DE (расширенная отладка) в CR4 установлен, и команда MOV выполняется для DR4 и DR5.

Особые ситуации режима реальной адресации

#UD, если бит DE (расширенная отладка) в CR4 установлен, и команда MOV выполняется для DR4 и DR5.

Особые ситуации режима V86

#GP(0), при попытке выполнить команду.

Замечание

Эти команды должны выполняться при уровне привилегий 0 или в реальном режиме, иначе будет сгенерирована особая ситуация защиты.

Всегда устанавливайте зарезервированные и неописанные биты отладочных регистров в предварительно прочитанные значения.

MOV Переслать данные в/из тестового регистра (Intel486, Intel386)

O D I T S Z A P C

Код операции	Команда	Такты		Пример
		486	386	
0F 24 D8+rd	MOV r32,TR3	3		mov eax,tr3
0F 24 E0+rd	MOV r32,TR4	4		mov ebx,tr4
0F 24 E8+rd	MOV r32,TR4	4		mov edx,tr5
0F 24 F0+rd	MOV r32,TR6	4	12	mov eax,tr6
0F 24 F8+rd	MOV r32,TR7	4	12	mov eax,tr7
0F 26 D8+rd	MOV TR6,r32	4		mov tr3,ecx
0F 26 E0+rd	MOV TR6,r32	4		mov tr4,eax
0F 26 E8+rd	MOV TR6,r32	4		mov tr5,eax
0F 26 F0+rd	MOV TR6,r32	4	12	mov tr6,edx
0F 26 F8+rd	MOV TR7,r32	4	12	mov tr7,eax

Операция

DEST = SRC;

Описание

Приведенная выше форма команды MOV сохраняет или загружает TR3, TR4, TR5, TR6, TR7 в или из регистра общего назначения.

С этими командами всегда используются 32 битные операнды независимо от атрибута размера операнда.

Особые ситуации защищенного режима

#GP(0), если текущий уровень привилегий не 0;

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

#GP(0), при попытке выполнить команду.

Замечание

Эти команды должны выполняться при уровне привилегий 0 или в реальном режиме, иначе будет сгенерирована особая ситуация защиты.

Всегда устанавливайте зарезервированные и неописанные биты тестовых регистров в предварительно прочитанные значения.

MOVS Переслать данные из строки в строку

MOVSB
MOVSW
MOVSD

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
A4	MOVS m8,m8	4	7	7	5	18	movsb
A5	MOVS m16,m16	4	7	7	5	18	movsw
A5	MOVS m32,m32	4	7	7			movsd
A4	MOVSB	4	7	7	5	18	movsb
A5	MOVSW	4	7	7	5	18	movsw
A5	MOVSD	4	7	7			movsd

Операция

```
IF (команда MOVSD) OR (команда работающая с двойными словами)
THEN OperandSize = 32;
ELSE OperandSize = 16;
FI;
IF AddressSize = 16
THEN использовать SI для source-index и DI для destination-index;
ELSE (* AddressSize = 32 *)
  использовать ESI для source-index и EDI для destination-index;
FI;
IF (команда байтового типа)
THEN
  [destination-index] = [source-index]; (* перенос байта *)
  IF DF = 0 THEN IncDec = 1 ELSE IncDec = -1; FI;
ELSE
  IF OperandSize = 16
  THEN
    [destination-index] = [source-index]; (* перенос слова *)
    IF DF = 0 THEN IncDec = 2 ELSE IncDec = -2; FI;
  ELSE (* OperandSize = 32 *)
    [destination-index] = [source-index]; (* перенос двойного слова *)
    IF DF = 0 THEN IncDec = 4 ELSE IncDec = -4; FI;
  FI;
FI;
source-index = source-index + IncDec;
destination-index = destination-index + IncDec;
```

Описание

Команда MOVS копирует байт, слово или двойное слово по адресу [(E)SI] в байт или слово по адресу ES:[(E)DI]. Операнд назначения должен адресоваться из регистра ES, замещение сегмента для назначения невозможно. Замещение сегмента может быть использовано для операнда источника (сегментом по умолчанию является DS).

Адрес источника и назначения определяются исключительно через значения регистров (E)SI, (E)DI. Загружайте корректные значения индексов в регистры (E)SI, (E)DI перед выполнением команды MOVS.

Команды MOVSB, MOVSW и MOVSD являются синонимами команды MOVS, использующей однобайтные, двухбайтные и четырехбайтные операнды соответственно.

После пересылки данных оба регистра, (E)SI и (E)DI автоматически изменяются. Если флаг DF равен 0 (была выполнена команда CLD), то регистры увеличиваются, если флаг DF равен 1 (была выполнена команда STD), то регистры уменьшаются. Увеличение или уменьшение происходит на 1, если пересылается байт, на 2, если пересылается слово, на 4, если пересылается двойное слово. Команде MOVS может предшествовать префикс REP для перемещения блоков из (E)CX байтов, слов или двойных слов. Смотрите команду REP для более подробной информации.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

MOVSX Переслать с расширением знака

O D I T S Z A P C

Код операции	Команда	Такты			Пример
		Pentium	486	386	
0F BE /r	MOVSX r16,r/m8	3	3/3	3/6	movsx ax,cl
0F BE /r	MOVSX r32,r/m8	3	3/3	3/6	movsx eax,byte ptr memo
0F BE /r	MOVSX r32,r/m16	3	3/3	3/6	movsx ecx,word ptr [edi]

Операция

DEST = SignExtend(SRC);

Описание

Команда MOVSX читает содержимое операнда источника (второй операнд - регистр или ячейка памяти) как байт или слово, знакорасширяет значение до атрибута размера операнда (16 или 32 бита) и сохраняет результат в регистре назначения.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

MOVZX Переслать с расширением нулями

O D I T S Z A P C

Код операции	Команда	Такты			Пример
		Pentium	486	386	
0F B6 /r	MOVZX r16,r/m8	3	3/3	3/6	movzx bx,ah
0F B6 /r	MOVZX r32,r/m8	3	3/3	3/6	movzx ebx,cl
0F BE /r	MOVZX r32,r/m16	3	3/3	3/6	movzx ecx,word ptr [ebx]

Операция

DEST = ZeroExtend(SRC);

Описание

Команда MOVZX читает содержимое операнда источника (второй операнд - регистр или ячейка памяти) как байт или слово, расширяет нулями значение до атрибута размера операнда (16 или 32 бита) и сохраняет результат в регистре назначения.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0) недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

MUL Беззнаковое умножение AL, AX или EAX

O	D	I	T	S	Z	A	P	C
*				?	?	?	?	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
F6 /4	MUL AL,r/m8	11	13/18	9-14/12-17	13/16	70-77/76-83+EA	mul ch
F7 /4	MUL AX,r/m16	11	13/26	9-22/12-25	21/24	118-133/124-139+EA	mul bx
F7 /4	MUL EAX,r/m32	10	13/42	9-38/12-41			mul edi

Операция

```
IF (операция над байтами)
THEN AX = AL * r/m8
ELSE (* операция над словами и двойными словами *)
  IF OperandSize = 16
  THEN DX:AX = AX * r/m16
  ELSE (* OperandSize = 32 *)
    EDX:EAX = EAX * r/m32
  FI;
FI;
```

Описание

Команда MUL выполняет беззнаковое умножение. Ее действие зависит от размера операндов следующим образом:

- Однобайтный операнд умножается на значение AL, результат помещается в регистр AX, флаги CF и OF очищаются, если значение в AH равно нулю, иначе устанавливаются в 1.
- Двухбайтный операнд умножается на значение AX, результат оставляется в паре регистров DX:AX. Регистр DX содержит старшие 16 бит результата. Флаги CF и OF очищаются, если значение в DX равно нулю, иначе устанавливаются в 1.
- Четырехбайтный операнд умножается на значение EAX, результат оставляется в паре регистров EDX:EAX. Регистр EDX содержит старшие 32 бита результата. Флаги CF и OF очищаются, если значение в EDX равно нулю, иначе устанавливаются в 1.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

NEG Изменить знак

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
F6 /3	NEG r/m8	1/3	1/3	2/6	2/7	3/16+EA	neg byte ptr [bx+6]
F7 /3	NEG r/m16	1/3	1/3	2/6	2/7	3/16+EA	neg cx
F7 /3	NEG r/m32	1/3	1/3	2/6			neg eax

Операция

```
IF r/m = 0 THEN CF = 0 ELSE CF = 1; FI;
r/m = -r/m;
```

Описание

Команда NEG заменяет значение регистра или операнда в памяти, его точным дополнением в двоичной системе, т.е. числом, которое при сложении с исходным операндом дает 0 во всех разрядах и перенос из старшего разряда.

Флаг CF устанавливается, если исходный операнд не нуль, иначе флаг CF очищается. Операнд вычитается из нуля, результат помещается на ме-

сто операнда, и флаги OF, SF, ZF и PF устанавливаются по результату операции.

Если операнд имеет максимальное (по модулю) отрицательное значение (например, -127 для 8 битного операнда), то его величина не меняется, но устанавливаются флаги OF и CF.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

NOP Нет операции

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
90	NOP	1	1	3	3	3	nop

Описание

Команда NOP не выполняет никаких действий. Команда NOP - однобайтная команда, которая только занимает место, но не влияет на состояние процессора, за исключением регистра (E)IP.

Команда NOP является вымышленной мнемоникой для команды XCHG (E)AX,(E)AX.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

NOT Поразрядное дополнение в двоичной системе (логическое "НЕ")

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
F6 /2	NOT r/m8	1/3	1/3	2/6	2/7	3/16+EA	not ah
F7 /2	NOT r/m16	1/3	1/3	2/6	2/7	3/16+EA	not word ptr [bp+5]
F7 /2	NOT r/m32	1/3	1/3	2/6			not edi

Операция

r/m = NOT r/m;

Описание

Команда NOT инвертирует операнд. Каждая 1 становится 0 и наоборот.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегмен-

те SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

OR Логическое "ИЛИ"

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
0C ib	OR AL,imm8	1	1	2	3	4	or al,0AAh
0D iw	OR AX,imm16	1	1	2	3	4	or ax,0FODh
0D id	OR EAX,imm32	1	1	2			or eax,23456789h
80 /1 ib	OR r/m8,imm8	1/3	1/3	2/7	3/7	4/17+EA	or byte ptr [di],5
81 /1 iw	OR r/m16,imm16	1/3	1/3	2/7	3/7	4/17+EA	or dx,0DBBh
81 /1 id	OR r/m32,imm32	1/3	1/3	2/7			or edx,0CAAAAh
83 /1 ib	OR r/m16,imm8	1/3	1/3	2/7	3/7	4/17+EA	or cx,0AAh
83 /1 ib	OR r/m32,imm8	1/3	1/3	2/7			or ecx,02h
08 /r	OR r/m8,r8	1/3	1/3	2/6	2/7	3/16+EA	or [di],ah
09 /r	OR r/m16,r16	1/3	1/3	2/6	2/7	3/16+EA	or bx,si
09 /r	OR r/m32,r32	1/3	1/3	2/6			or memory,eax
0A /r	OR r8,r/m8	1/2	1/2	2/7	2/7	3/9+EA	or dl,sum
0B /r	OR r16,r/m16	1/2	1/2	2/7	2/7	3/9+EA	or di,[si+12]
0B /r	OR r32,r/m32	1/2	1/2	2/7			or ecx,raznost

Операция

```
DEST = DEST OR SRC;
CF = 0;
OF = 0;
```

Описание

Команда OR вычисляет включающее "ИЛИ" для двух операндов и помещает результат на место первого операнда. Каждый бит результата равен 0, если соответствующие биты операндов равны 0, иначе бит результата равен 1.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

OUT Вывод в порт

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
E6 ib	OUT imm8,AL	12,pm=9*/ 26**,vm=24	16,pm=11*/ 31**,vm=29	10,pm=4*/ 24**	3	10	out 60h,al
E7 ib	OUT imm8,AX	12,pm=9*/ 26**,vm=24	16,pm=11*/ 31**,vm=29	10,pm=4*/ 24**	3	10	out 00h,ax
E7 ib	OUT imm8,EAX	12,pm=9*/ 26**,vm=24	16,pm=11*/ 31**,vm=29	10,pm=4*/ 25**			out 81h,eax
EE	OUT DX,AL	12,pm=9*/ 25**,vm=24	16,pm=11*/ 31**,vm=29	11,pm=5*/ 25**	3	8	out dx,al
EF	OUT DX,AX	12,pm=9*/ 25**,vm=24	16,pm=11*/ 31**,vm=29	11,pm=5*/ 25**	3	8	out dx,ax
EF	OUT DX,EAX	12,pm=9*/ 25**,vm=24	16,pm=11*/ 31**,vm=29	11,pm=5*/ 25**			out dx,eax

*если CPL ≤ IOPL
**если CPL > IOPL

Операция

```
IF (PE = 1) AND ((VM = 1) OR (CPL > IOPL))
THEN (* виртуальный-8086 режим, или защищенный режим с CPL > IOPL *)
  IF NOT I-O-Permission (DEST, width(DEST))
  THEN #GP(0);
  FI;
FI;
[DEST] = SRC; (* использовано пространство ввода-вывода *)
```

Описание

Команда OUT пересылает байт, слово или двойное слово данных из регистра (AL, AX или EAX), заданного вторым операндом, в порт вывода, занумерованный первым операндом. Адрес порта может задаваться вторым байтом команды (imm8) или содержимым регистра DX. Таким образом, возможно обращение к 256 портам; адреса которых фиксированы в программе (байт imm8), или к 65536 портам, адреса которых могут изменяться путем перезагрузки содержимого регистра DX. При работе с 16 разрядными данными число адресуемых портов уменьшается вдвое, а их адресация должна производиться четными числами: 0, 2, 4 ... 254 для фиксированных портов; 0, 2, 4 ... 65534 для изменяемых портов. Соответственно, для 32 разрядных данных число портов сокращается еще вдвое, а их адресация производится числами кратными четырем: 0, 4, 8 ... 252 ... 65532. Таким образом, номера портов должны быть выровнены по границам передаваемых данных согласно их разрядности: 1, 2 или 4 байта. В этом случае пересылка осуществляется за один цикл. Если заданный номер 16 или 32 разрядного порта имеет невыровненное значение, то для передачи данных требуется дополнительный цикл шины.

Особые ситуации защищенного режима

#GP(0), если значение текущего уровня привилегий больше (имеет меньше привилегий), чем уровень привилегий ввода-вывода, и любой из соответствующих битов разрешения ввода-вывода в TSS равен 1.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

#GP(0), если любой из соответствующих битов разрешения ввода-вывода в TSS равен 1.

Замечание

После того как команда OUT выполнена, процессор Pentium проверяет, установлен ли сигнал EWBE# в активное состояние перед началом выполнения следующей команды. Заметьте, что команда может быть выбрана с упреждением, даже если EWBE# не активен, но она не выполняется до тех пор, пока сигнал отсутствует.

OUTS OUTSB OUTSW OUTSD — Вывод строки в порт

O D I T S Z A P C

Код операции	Команда	Такты Pentium	486	386	286	Пример
6E	OUTS DX,r/m8	13,pm=10*/ 27**,vm=25	17,pm=10*/ 32**,vm=30	14,pm=8*/ 28**	5	outsb
6F	OUTS DX,r/m16	13,pm=9*/ 27**,vm=25	16,pm=10*/ 32**,vm=30	10,pm=8*/ 28**	5	outsw
6F	OUTS DX,r/m32	13,pm=9*/ 27**,vm=25	16,pm=10*/ 32**,vm=30	10,pm=8*/ 28**		outsd
6E	OUTSB	13,pm=9*/ 27**,vm=25	16,pm=10*/ 32**,vm=30	11,pm=8*/ 28**	5	outsb
6F	OUTSW	13,pm=9*/ 27**,vm=25	16,pm=10*/ 32**,vm=30	11,pm=8*/ 28**	5	outsw
6F	OUTSD	13,pm=9*/ 27**,vm=25	16,pm=10*/ 32**,vm=30	11,pm=8*/ 28**		outsd

*если CPL ≤ IOPL
**если CPL > IOPL

Операция

```
IF AdressSize = 16
THEN использовать SI для source-index;
ELSE (* AdressSize = 32 *)
  использовать ESI для source-index;
FI;
IF (PE = 1) AND ((VM = 1) OR (CPL > IOPL))
THEN (* виртуальный-8086 режим, или защищенный режим с CPL > IOPL *)
  IF NOT I-O-Permission (DEST, width(DEST))
  THEN #GP(0);
  FI;
FI;
IF (команда байтового типа)
THEN
  [DX] = [source-index]; (* записать байт в DX адрес ввода-вывода *)
  IF DF = 0 THEN IncDec = 1 ELSE IncDec = -1; FI;
ELSE
  IF OperandSize = 16
  THEN
    [DX] = [source-index]; (* записать слово в DX адрес ввода-вывода *)
    IF DF = 0 THEN IncDec = 2 ELSE IncDec = -2; FI;
  ELSE (* OperandSize = 32 *)
    THEN
    [DX] = [source-index]; (* записать двойное слово в DX адрес ввода-вывода *)
    IF DF = 0 THEN IncDec = 4 ELSE IncDec = -4; FI;
  FI;
FI;
source-index = source-index + IncDec;
```

Описание

Команда OUTS пересылает байт, слово или двойное слово из ячейки памяти, адресуемой индексным регистром-источником, в порт вывода, адресуемый регистром DX. Если атрибут размера адреса для этой команды равен 16 бит, то в качестве индексного регистра-источника используется регистр SI. Если атрибут размера адреса равен 32 бита, то в качестве индексного регистра-источника используется регистр ESI.

Команда OUTS не позволяет указывать номер порта в качестве непосредственного значения. Порт должен адресоваться через регистр DX. Загружайте корректное значение в регистр DX перед выполнением команды OUTS.

Адрес источника данных определяется содержимым индексного регистра-источника. Загружайте корректное значение индекса в регистр (E)SI перед выполнением команды OUTS.

После пересылки индексный регистр-источник автоматически изменяется. Если флаг DF равен 0 (была выполнена команда CLD), то регистр увеличивается, если флаг DF равен 1 (была выполнена команда STD), то регистр уменьшается. Увеличение или уменьшение происходит на 1, если пересылается байт, на 2, если пересылается слово, на 4, если пересылается двойное слово.

Команды OUTSB, OUTSW и OUTSD являются синонимами команды OUTS для байта, слова и двойного слова соответственно.

Команде OUTS может предшествовать префикс REP для вывода блока из (E)CX байтов, слов или двойных слов. Смотрите команду REP для более подробной информации.

Особые ситуации защищенного режима

#GP(0), если значение текущего уровня привилегий больше (имеет меньше привилегий), чем уровень привилегий ввода-вывода, и любой из соответствующих битов разрешения ввода-вывода в TSS равен 1. #GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS,DS,ES,FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

#GP(0), если любой из соответствующих битов разрешения ввода-вывода в TSS равен 1. Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

После того, как команда OUTS выполнена, процессор Pentium проверяет, что сигнал EWBE# установлен в активное состояние перед началом выполнения следующей команды. Заметьте, что команда может быть выбрана с упреждением, даже если EWBE# не активен, но она не выполняется до тех пор, пока сигнал отсутствует.

POP Извлечь из стека

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
8F /0	POP m16	3	6	5	5	17+EA	pop word ptr [di]
8F /0	POP m32	3	6	5			pop dword ptr mem
58+rw	POP r16	1	4	4	5	8	pop cx
58+rd	POP r32	1	4	4			pop edi
1F	POP DS	3	3	7,pm=21	5,pm=20	8	pop ds
07	POP ES	3	3	7,pm=21	5,pm=20	8	pop es
17	POP SS	3	3	7,pm=21	5,pm=20	8	pop es
0F A1	POP FS	3	3	7,pm=21			pop fs
0F A9	POP GS	3	3	7,pm=21			pop gs

Операция

```
IF StackAddrSize = 16
THEN
  IF OperandSize = 16
  THEN
    DEST = (SS:SP); (* копировать слово *)
    SP = SP + 2;
  ELSE (* OperandSize = 32 *)
    DEST = (SS:SP); (* копировать двойное слово *)
    SP = SP + 4;
  FI;
```

```
ELSE (* StackAddrSize = 32 *)
  IF OperandSize = 16
  THEN
    DEST = (SS:ESP); (* копировать слово *)
    ESP = ESP + 2;
  ELSE (* OperandSize = 32 *)
    DEST = (SS:ESP); (* копировать двойное слово *)
    ESP = ESP + 4;
  FI;
FI;
```

Описание

Команда POP заменяет содержимое операнда памяти, регистра или сегментного регистра значением из вершины стека, адресуемой через SS:SP (атрибут размера адреса равен 16 бит) или SS:ESP (атрибут размера адреса равен 32 бит). Указатель стека SP увеличивается на 2 для 16 битного размера операнда или на 4 для 32 битного размера операнда, после чего он указывает на новую вершину стека.

Команда POP CS не является командой процессора. Извлечение из стека в регистр CS осуществляется при помощи команды RET.

Если операнд назначения - сегментный регистр (DS, ES, FS, GS или SS), то извлекаемое значение должно быть селектором. В защищенном режиме загрузка селектора инициирует автоматическую загрузку информации дескриптора, ассоциирующегося с этим селектором, в скрытую часть сегментного регистра, а также проверку правильности информации селектора и дескриптора.

Пустое значение (0000 - 0003) может быть загружено в регистры DS, ES, FS или GS не вызывая исключений защиты. Попытка обратиться в сегмент, чей соответствующий сегментный регистр загружен пустым значением, приведет к исключению #GP(0). Обращение к памяти не происходит. Сохраненное значение сегментного регистра является пустым.

Команда POP SS запрещает все прерывания включая NMI до тех пор пока не выполнится следующая команда. Это позволяет последовательно выполнить команды POP SS и MOV (E)SP,(E)BP без опасности существования ошибочного стека во время прерывания. Тем не менее, использование команды LSS является предпочтительным методом загрузки регистров SS и (E)SP.

Команда POP-в-память, которая использует указатель стека в качестве базового регистра, обращается к памяти после извлечения данных из стека. Используемая база является значением ESP после выполнения команды.

Загрузка сегментного регистра в защищенном режиме вызывает специальные проверки и действия описанные ниже:

```
IF (загружается SS)
  IF (селектор нулевой) THEN #GP(0);
  Индекс селектора должен попадать в пределы таблицы дескрипторов,
   иначе #GP(селектор);
  RPL селектора должен быть равен CPL, иначе #GP(селектор);
  AR байт дескриптора должен задавать разрешенный для записи сегмент,
   иначе #GP(селектор);
  DPL в AR байте должен быть равен CPL, иначе #GP(селектор);
  Сегмент должен присутствовать, иначе #SS(селектор);
  Загрузить регистр SS селектором;
  Загрузить скрытую часть регистра SS дескриптором;

IF (загружается не нулевой селектор в регистр DS, ES, FS или GS)
  AR байт дескриптора должен задавать сегмент данных или разрешенный
     для чтения кодовый сегмент, иначе #GP(селектор);
  IF (сегмент данных или несогласованный кодовый сегмент)
  THEN RPL и CPL должны быть меньше или равны DPL в AR байте;
  ELSE #GP(селектор);
  FI;
  Сегмент должен присутствовать, иначе #NP(селектор);
  Загрузить сегментный регистр селектором;
  Загрузить скрытую часть сегментного регистра дескриптором;

IF (загружается нулевой селектор в регистр DS, ES, FS или GS)
  Загрузить сегментный регистр селектором;
  Очистить бит достоверности (valide bit) в скрытой части сегментного
   регистра;
```

Особые ситуации защищенного режима

#GP, #SS и #NP, если загружается сегментный регистр; #SS(0), если текущая вершина стека не попадает в пределы сегмента стека; #GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Пара команд PUSH/POP выполняются без привлечения дополнительных тактов (Pentium). Бит B дескриптора сегмента стека будет определять размер адреса стека (StackAddrSize).

Команда POP (E)SP увеличивает (E)SP перед тем, как данные из старой вершины стека будут записаны в назначение.

POPA Извлечь из стека все общие регистры

POPAD

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
61	POPA	5	9	24	19	popa
61	POPAD	5	9	24		popad

Операция

```
IF OperandSize = 16 (* команда POPA *)
THEN
  DI = Pop();
  SI = Pop();
  BP = Pop();
  increment SP by 2 (* пропустить следующие 2 байта в стеке *)
  BX = Pop();
  DX = Pop();
  CX = Pop();
  AX = Pop();
ELSE (* OperandSize = 32, команда POPAD *)
  EDI = Pop();
  ESI = Pop();
  EBP = Pop();
  increment SP by 4 (* пропустить следующие 4 байта в стеке *)
  EBX = Pop();
  EDX = Pop();
  ECX = Pop();
  EAX = Pop();
FI;
```

Описание

Команда POPA извлекает восемь 16 битных общих регистров. Однако значение SP отбрасывается вместо того, чтобы загрузиться в регистр SP. Команда POPA проделывает действия обратные предшествующей команде PUSHA, восстанавливая общие регистры к значениям, бывшим в них до

выполнения команды PUSHA. Первым извлекаемым регистром является регистр DI.

Команда POPAD извлекает восемь 32 битных общих регистров. Однако значение ESP отбрасывается вместо того, чтобы загрузиться в регистр ESP. Команда POPAD проделывает действия обратные предшествующей команде PUSHAD, восстанавливая общие регистры к значениям, бывшим в них до выполнения команды PUSHAD. Первым извлекаемым регистром является регистр EDI.

Особые ситуации защищенного режима

#SS(0), если текущая вершина стека не попадает в пределы сегмента стека; #PF(код ошибки), страничная ошибка.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка.

POPF Извлечь из стека в регистр FLAGS
POPFD Извлечь из стека в регистр EFLAGS

O	D	I	T	S	Z	A	P	C
*	*	*	*	*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
9D	POPF	6,pm=4	9,pm=6	5	5	8	popf
9D	POPFD	6,pm=4	9,pm=6	5			popfd

Операция

```
IF VM = 0 (* Не в виртуальном-8086 режиме *)
THEN
  IF OperandSize = 32;
  THEN EFLAGS = Pop() AND 277FD7h;
  ELSE FLAGS = Pop();
```

```
  FI;
ELSE (* В виртуальном-8086 режиме *)
  IF IOPL = 3
  THEN
    IF OperandSize = 32
    THEN
      TempEflags = Pop();
      EFLAGS = ((EFLAGS AND 1B3000h) OR (TempEflags AND !1B3000h));
              (* VM, RF, IOPL, VIP, и VIF биты EFLAGS не изменяются командой
                 POPFD *)
    ELSE
      FLAGS = Pop();
    FI;
  ELSE
    #GP(0); (* ловушка монитора V86 *)
  FI;
FI;
```

Описание

Команды POPF и POPFD извлекают слово или двойное слово из вершины стека и сохраняют значение в регистре (E)FLAGS. Если атрибут размера операнда этой команды равен 16 бит, извлекается слово из вершины стека, и его значение сохраняется в регистре FLAGS. Если атрибут размера операнда равен 32 бита, то извлекается двойное слово, и его значение сохраняется в регистре EFLAGS.

В режиме V86, когда IOPL < 3, команда POPF вызывает исключение #GP. Если IOPL = 3, при выполнении в режиме V86, POPF извлекает слово в регистр FLAGS.

Обратитесь к главе "Прикладная архитектура" за информацией о регистрах FLAGS и EFLAGS. Заметьте, что биты 16 (VM) и 17 (RF) регистра EFLAGS не изменяются командой POPFD.

Уровень привилегий ввода/вывода (IOPL) изменяется только при выполнении команды POPFD на уровне привилегий, по крайней мере таком же, как и текущий уровень привилегий ввода/вывода. (Режим реальной адресации эквивалентен уровню привилегий 0.) Если команда POPFD выполняется с недостаточным уровнем привилегий, то исключение не вызывается, а привилегированные биты не изменяются.

Особые ситуации защищенного режима

#SS(0), если текущая вершина стека не попадает в пределы сегмента стека.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

#GP(0), если уровень привилегий ввода/вывода меньше чем 3 (для разрешения эмуляции); #GP(0), если произведена попытка выполнить POPF с префиксом замещения размера операнда.

Замечание

Информацию о результатах POPF и POPFD при использовании исключений виртуального режима смотрите в Приложении H.

PUSH Поместить операнд в стек

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
FF /6	PUSH r/m16	1/2	4	5	5	16+EA	push word ptr [si]
FF /6	PUSH r/m32	1/2	4	5			push dword ptr [12]
50+rw	PUSH r16	1	1	2	3	11	push di
50+rd	PUSH r32	1	1	2			push esi
6A	PUSH imm8	1	1	2	3		push 0Ah
68	PUSH imm16	1	1	2	3		push 023Dh
68	PUSH imm32	1	1	2			push 012345FFh
0E	PUSH CS	1	3	2	3	10	push cs
16	PUSH SS	1	3	2	3	10	push ss
1E	PUSH DS	1	3	2	3	10	push ds
06	PUSH ES	1	3	2	3	10	push es
0F A0	PUSH FS	1	3	2			push fs
0F A8	PUSH GS	1	3	2			push gs

Операция

```
IF StackAddrSize = 16
THEN
  IF OperandSize = 16
  THEN
    SP = SP - 2;
    (SS:SP) = (SOURCE); (* пересылает слово *)
  ELSE (* OperandSize = 32 *)
    SP = SP - 4;
    (SS:SP) = (SOURCE); (* пересылает двойное слово *)
  FI;
ELSE (* StackAddrSize = 32 *)
  IF OperandSize = 16
  THEN
    ESP = ESP - 2;
```

```
    (SS:ESP) = (SOURCE); (* пересылает слово *)
  ELSE (* OperandSize = 32 *)
    ESP = ESP - 4;
    (SS:ESP) = (SOURCE); (* пересылает двойное слово *)
  FI;
FI;
```

Описание

Команда PUSH уменьшает указатель стека (E)SP на 2 для 16 битного размера операнда или на 4 для 32 битного размера операнда. Команда PUSH затем помещает операнд в новую вершину стека, адресуемую через SS:SP (атрибут размера адреса равен 16 бит) или SS:ESP (атрибут размера адреса равен 32 бита).

Команда PUSH (E)SP помещает в стек значение регистра (E)SP такое, каким оно было до команды. Для процессора 8086 (8088) команда PUSH SP помещает в стек новое значение SP (уменьшенное на 2).

Команда PUSH-из-памяти, которая использует указатель стека в качестве базового регистра, обращается к памяти перед занесением данных в стек. В качестве базы используется значение (E)SP перед выполнением команды.

Особые ситуации защищенного режима

#SS(0), если новая вершина стека не попадает в пределы сегмента стека; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Нет. Если регистр SP или ESP равен 1 или 3, то процессор останавливается из-за недостатка места в стеке. (Следует отметить, что в этом случае различные процессоры могут вести себя весьма не адекватно, это относится как к различным клонам, так и к процессорам Intel. Возможно жесткое зависание процессора, сброс, останов и пр.)

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Команда PUSH, использующаяся с операндом в памяти, выполняется дольше, чем последовательность из двух команд, пересылающих операнд через регистр.

Пара команд PUSH/POP выполняются без привлечения дополнительных тактов (Pentium).

Бит B дескриптора сегмента стека будет определять размер адреса стека (StackAddrSize).

PUSHA Поместить в стек все общие регистры

PUSHAD

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
60	PUSHA	5	11	18	17	pusha
60	PUSHAD	5	11	18		pushad

Операция

```
IF OperandSize = 16 (* команда PUSHA *)
THEN
  Temp = (SP);
  Push(AX);
  Push(CX);
  Push(DX);
  Push(BX);
  Push(Temp);
  Push(BP);
  Push(SI);
  Push(DI);
ELSE (* OperandSize = 32, команда PUSHAD *)
  Temp = (ESP);
  Push(EAX);
  Push(ECX);
  Push(EDX);
  Push(EBX);
  Push(Temp);
  Push(EBP);
  Push(ESI);
  Push(EDI);
FI;
```

Описание

Команды PUSHA и PUSHAD сохраняют в стеке восемь 16 битных (PUSHA) или 32 битных (PUSHAD) общих регистров. Команда PUSHA уменьшает указатель стека (SP) на 16. Команда PUSHAD уменьшает указатель стека (ESP) на 32. Последний помещаемый в стек регистр - регистр (E)DI.

Особые ситуации защищенного режима

#SS(0), если начальный или конечный адрес вершины стека не попадает в пределы сегмента стека; #PF(код ошибки), страничная ошибка.

Особые ситуации режима реальной адресации

Перед выполнением команды PUSHA или PUSHAD процессор останавливается, если регистр (E)SP равен 1, 3, 5. (Следует отметить, что в этом случае различные процессоры могут вести себя весьма не адекватно, это относится как к различным клонам, так и к процессорам Intel. Возможно жесткое зависание процессора, сброс, останов и пр.) Если регистр (E)SP равен 7, 9 ... 15 (... 29, 31) происходит прерывание 13.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка.

PUSHF Поместить регистр флагов в стек
PUSHFD

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
9C	PUSHF	4,pm=3	4,pm=3	4	3	10	pushf
9C	PUSHFD	4,pm=3	4,pm=3	4			pushfd

Операция

```
IF VM = 0 (* Не в виртуальном-8086 режиме *)
THEN
  IF OperandSize = 32;
```

```
  THEN Push(EFLAGS AND 0FCFFFFh); (* VM и RF биты EFLAG очищаются *)
  ELSE Push(FLAGS);
  FI;
ELSE (* В виртуальном-8086 режиме *)
  IF IOPL = 3
  THEN
    IF OperandSize = 32
    THEN Push(EFLAGS AND 0FCFFFFh); (* VM и RF биты EFLAG
                                       очищаются *)
    ELSE Push(FLAGS);
    FI;
  ELSE
    #GP(0); (* ловушка монитора V86 *)
  FI;
FI;
```

Описание

Команда PUSHF уменьшает указатель стека на 2 и копирует регистр FLAGS в новую вершину стека, команда PUSHFD уменьшает указатель стека на 4 и копирует регистр EFLAGS в новую вершину стека, указываемую SS:(E)SP.

Обратитесь к главе "Прикладная архитектура" за информацией о регистрах FLAGS и EFLAGS.

Особые ситуации защищенного режима

#SS(0), если новая вершина стека не попадает в пределы сегмента стека.

Особые ситуации режима реальной адресации

Нет. Процессор останавливается при недостатке пространства в стеке. (Следует отметить, что в этом случае различные процессоры могут вести себя весьма не адекватно, это относится как к различным клонам, так и к процессорам Intel. Возможно жесткое зависание процессора, сброс, останов и пр.)

Особые ситуации режима V86

#GP(0), если уровень привилегий ввода/вывода меньше чем 3 (для разрешения эмуляции).

Замечание

Информацию о результатах PUSHF и PUSHFD при использовании исключений виртуального режима смотрите в Приложении H.

RCL Циклический сдвиг влево через перенос (CF)
RCR Циклический сдвиг вправо через перенос (CF)
ROL Циклический сдвиг влево
ROR Циклический сдвиг вправо

O	D	I	T	S	Z	A	P	C
*								*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
D0 /2	RCL r/m8,1	1/3	3/4	9/10	2/7	2/15+EA	rcl ah,1
D2 /2	RCL r/m8,CL	7-24/9-26	8-30/9-31	9/10	5/8	8+4 per bit/(20 +4 per bit)+EA	rcl dh,cl
C0 /2 ib	RCL r/m8,imm8	8-25/10-27	8-30/9-31	9/10	5/8		rcl al,4
D1 /2	RCL r/m16,1	1/3	3/4	9/10	2/7	2/15+EA	rcl bx,1
D3 /2	RCL r/m16,CL	7-24/9-26	8-30/9-31	9/10	5/8	8+4 per bit/(20 +4 per bit)+EA	rcl di,cl
C1 /2 ib	RCL r/m16,imm8	8-25/10-27	8-30/9-31	9/10	5/8		rcl di,9
D1 /2	RCL r/m32,1	1/3	3/4				rcl esi,1
D3 /2	RCL r/m32,CL	7-24/9-26	8-30/9-31	9/10			rcl ecx,12
C1 /2 ib	RCL r/m32,imm8	8-25/10-27	8-30/9-31	9/10			rcl edx,8
D0 /3	RCR r/m8,1	1/3	3/4	9/10	2/7	2/15+EA	rcr bh,1
D2 /3	RCR r/m8,CL	7-24/9-26	8-30/9-31	9/10	5/8	8+4 per bit/(20 +4 per bit)+EA	rcr bh,cl
C0 /3 ib	RCR r/m8,imm8	8-25/10-27	8-30/9-31	9/10	5/8		rcr cl,4
D1 /3	RCR r/m16,1	1/3	3/4	9/10	2/7	2/15+EA	rcr cx,1
D3 /3	RCR r/m16,CL	7-24/9-26	8-30/9-31	9/10	5/8	8+4 per bit/(20 +4 per bit)+EA	rcr si,cl
C1 /3 ib	RCR r/m16,imm8	8-25/10-27	8-30/9-31	9/10	5/8		rcr di,6
D1 /3	RCR r/m32,1	1/3	3/4				rcr edi,1
D3 /3	RCR r/m32,CL	7-24/9-26	8-30/9-31	9/10			rcr ebx,12
C1 /3 ib	RCR r/m32,imm8	8-25/10-27	8-30/9-31	9/10			rcr ebp,8
D0 /0	ROL r/m8,1	1/3	3/4	3/7	2/7	2/15+EA	rol cl,1
D2 /0	ROL r/m8,CL	4	3/4	3/7	5/8	8+4 per bit/(20 +4 per bit)+EA	rol dh,cl
C0 /0 ib	ROL r/m8,imm8	1/3	2/4	3/7	5/8		rol ch,3
D1 /0	ROL r/m16,1	1/3	3/4	3/7	2/7	2/15+EA	rol bp,1
D3 /0	ROL r/m16,CL	4	3/4	3/7	5/8	8+4 per bit/(20 +4 per bit)+EA	rol ax,cl
C1 /0 ib	ROL r/m16,imm8	1/3	2/4	3/7	5/8		rol cx,7
D1 /0	ROL r/m32,1	1/3	3/4				rol ebx,1
D3 /0	ROL r/m32,CL	4	3/4	3/7			rol edx,11
C1 /0 ib	ROL r/m32,imm8	1/3	2/4	3/7			rol eax,9
D0 /1	ROR r/m8,1	1/3	3/4	3/7	2/7	2/15+EA	ror bl,1
D2 /1	ROR r/m8,CL	4	3/4	3/7	5/8	8+4 per bit/(20 +4 per bit)+EA	ror byte p tr [bx],cl
C0 /1 ib	ROR r/m8,imm8	1/3	2/4	3/7	5/8		ror al,4
D1 /1	ROR r/m16,1	1/3	3/4	3/7	2/7	2/15+EA	ror di,1
D3 /1	ROR r/m16,CL	4	3/4	3/7	5/8	8+4 per bit/(20 +4 per bit)+EA	ror dx,cl
C1 /1 ib	ROR r/m16,imm8	1/3	2/4	3/7	5/8		ror bx,3
D1 /1	ROR r/m32,1	1/3	3/4				ror ecx,1
D3 /1	ROR r/m32,CL	4	3/4	3/7			ror eax,20
C1 /1 ib	ROR r/m32,imm8	1/3	2/4	3/7			ror ebp,6

Операция

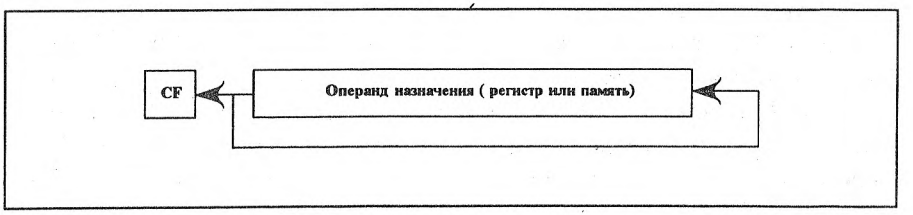
```
(* ROL - Циклический сдвиг влево *)
temp = COUNT;
WHILE (temp ≠ 0)
```

```
DO
  tmpcf = high-order bit of (r/m);
  r/m = r/m * 2 + (tmpcf);
  temp = temp - 1;
OD;
IF COUNT = 1
THEN
  IF high-order bit of r/m ≠ CF
  THEN OF = 1;
  ELSE OF = 0;
  FI;
ELSE OF = undefined;
FI;
(* ROR - Rotate Right *)
temp = COUNT;
WHILE (temp ≠ 0)
DO
  tmpcf = low-order bit of (r/m);
  r/m = r/m / 2 + (tmpcf*2^width(r/m));
  temp = temp - 1;
OD;
IF COUNT = 1
THEN
  IF (high-order bit of r/m) ≠ (bit next to high-order bit of r/m)
  THEN OF = 1;
  ELSE OF = 0;
  FI;
ELSE (OF неопределен);
FI;
```

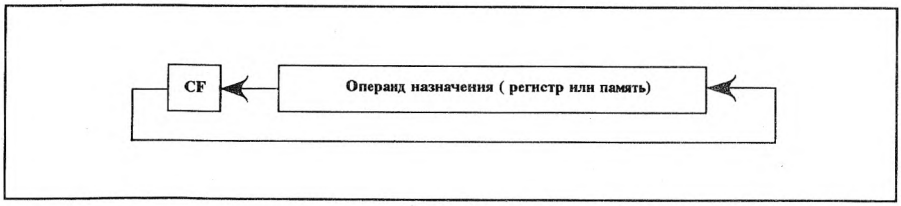
Описание

Команды циклического сдвига вращают биты регистра или операнда в памяти. Команды циклического сдвига влево сдвигают все биты вверх, за исключением старшего бита, который возвращается вниз. Команды циклического сдвига вправо делают обратное: биты сдвигаются вниз, а младший бит переносится в вершину. Для команд RCL и RCR флаг CF является частью вращаемого множества. Команда RCL сдвигает флаг CF в младший бит, а старший бит сдвигает во флаг CF. Команда RCR сдвигает флаг CF в старший бит, а младший бит сдвигает во флаг CF. Для команд ROL и ROR исходное значение флага CF не является частью результата, но флаг CF получает копию бита, который был сдвинут из одного конца в другой. Следующие рисунки поясняют выше сказанное.

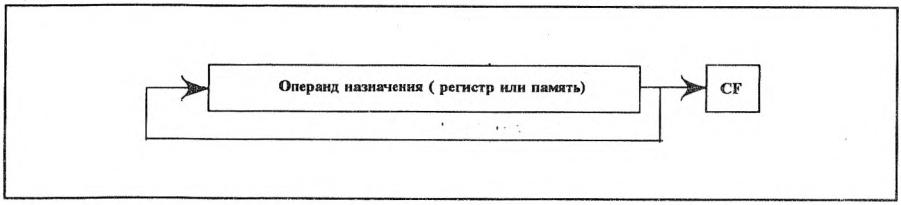
Команда ROL



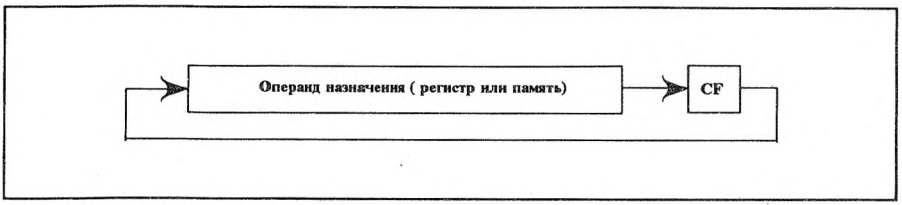
Команда RCL



Команда ROR



Команда RCR



Команда вращения повторяется столько раз, сколько указано вторым операндом, который может быть либо непосредственным числом, либо содержимым регистра CL. Чтобы уменьшить максимальное время выполнения команды процессор не позволяет счетчику вращений быть больше 31. Если счетчик вращений больше 31, используются только младшие 5 бит. Процессор 8086 (8088) не маскирует счетчик вращений. (Процессоры Intel386, Intel486 и Pentium во всех режимах, в том числе и в режиме V86 маскируют счетчик вращений.)

Флаг OF определен только для форм команд однократного вращения (вторым операндом является 1). Он неопределен во всех остальных случаях. Для сдвигов и циклических сдвигов влево - бит CF после сдвига складывается по модулю два (XOR) со старшим битом результата, чтобы получить флаг OF. Для сдвигов и циклических сдвигов вправо - старшие

два бита результата складываются по модулю два (XOR), чтобы получить флаг OF.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

RDMSR Чтение из особого регистра модели (Model Specific Register)

O D I T S Z A P C

Код операции	Команда	Такты Pentium	Пример
0F 32	RDMSR	20–24	rdmsr

Операция

```
EDX:EAX = MSR[ECX];
```

Описание

Значение в ECX указывает один из 64 битных особых регистров модели процессора Pentium. Содержимое особого регистра модели

копируется в EDX:EAX. EDX загружается старшими 32 битами, а EAX - младшими 32 битами.

Следующие значения используются для выбора особых регистров модели в процессоре Pentium:

Величина (Hex)	Название регистра	Описание
00h	Machine Check Address	Сохраняет адрес цикла, вызвавшего исключение
01h	Machine Check Type	Сохраняет тип цикла, вызвавшего исключение

Информацию о других величинах, использующихся при тестировании КЭШ, TLB и BTB, а также при осуществлении контроля, смотрите в Приложении H.

Особые ситуации защищенного режима

#GP(0), если текущий уровень привилегий не 0, либо значение в ECX не указывает особый регистр модели, который реализован в процессоре Pentium.

Особые ситуации режима реальной адресации

Прерывание 13, если значение в ECX не указывает особый регистр модели, который реализован в процессоре Pentium.

Особые ситуации режима V86

#GP(0), при попытке выполнить команду.

Замечание

Эта команда должна выполняться при уровне привилегий равном 0 или в реальном режиме, иначе будет сгенерирована особая ситуация защиты.

Если в особом регистре модели реализовано меньше чем 64 бита, то значение возвращаемое в EDX:EAX неопределено для нереализованных битов.

RDMSR используется для чтения содержимого особых регистров модели, которые контролируют функции тестирования, трассировки выполнения, слежения и проверки ошибок машины. За дополнительной информацией обратитесь к "Pentium™ Processor Data Book".

Значения 03h, 0Fh и значения выше 13h зарезервированы. Не выполняйте RDMSR с зарезервированными значениями в ECX.

REP Повторить следующую строковую операцию

REPE
REPZ
REPNE
REPNZ

O	D	I	T	S	Z	A	P	C
					*			

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
F3 6C	REP INS r/m8,DX	11+3*(E)CX, pm*1=8+ 3*(E)CX, pm*2=25+ 3*(E)CX, vm=23+3* (E)CX	16+8*E(CX), pm*1=10+ 8*(E)CX, pm*2=30+ 8(E)CX, vm=29+8* (E)CX	13+6*(E)CX, pm*1=7+ 6*(E)CX, pm*2=27+ 6*(E)CX	5+4*CX		rep insb
F3 6D	REP INS r/m16,DX	11+3*(E)CX, pm*1=8+ 3*(E)CX, pm*2=25+ 3*(E)CX, vm=23+3* (E)CX	16+8*E(CX), pm*1=10+ 8*(E)CX, pm*2=30+ 8(E)CX, vm=29+8* (E)CX	13+6*(E)CX, pm*1=7+ 6*(E)CX, pm*2=27+ 6*(E)CX	5+4*CX		rep insw
F3 6D	REP INS r/m32,DX	11+3*(E)CX, pm*1=8+ 3*(E)CX, pm*2=25+ 3*(E)CX, vm=23+3* (E)CX	16+8*E(CX), pm*1=10+ 8*(E)CX, pm*2=30+ 8(E)CX, vm=29+8* (E)CX	13+6*(E)CX, pm*1=7+ 6*(E)CX, pm*2=27+ 6*(E)CX			rep insd
F3 A4	REP MOVS m8,m8	6*3,13*4, 13+(E)CX*5	5*3,13*4, 12+3* (E)CX*5	5+4*(E)CX	5+4*CX	9+17*CX	rep movsb
F3 A5	REP MOVS m16,m16	6*3,13*4, 13+(E)CX*5	5*3,13*4, 12+3* (E)CX*5	5+4*(E)CX	5+4*CX	9+17*CX	rep movsw
F3 A5	REP MOVS m32,m32	6*3,13*4, 13+(E)CX*5	5*3,13*4, 12+3* (E)CX*5	5+4*(E)CX			rep movsd
F3 6E	REP OUTS DX,r/m8	13+4*(E)CX, pm*1=10+ 4*(E)CX, pm*2=27+ 4*(E)CX vm=25+4* (E)CX	17+5*(E)CX, pm*1=11+ 5*(E)CX, pm*2=31+ 5*(E)CX	5+12*(E)CX, pm*1=6+ 5*(E)CX, pm*2=26+ 5*(E)CX	5+4*CX		rep outsb
F3 6F	REP OUTS	13+4*(E)CX,	17+5*(E)CX,	5+12*(E)CX,	5+4*CX		rep outsw

		DX,r/m16	pm*1=10+ 4*(E)CX, pm*2=27+ 4*(E)CX vm=25+4* (E)CX	pm*1=11+ 5*(E)CX, pm*2=31+ 5*(E)CX	pm*1=6+ 5*(E)CX, pm*2=26+ 5*(E)CX			
F3	6F	REP OUTS DX,r/m32	13+4*(E)CX, pm*1=10+ 4*(E)CX, pm*2=27+ 4*(E)CX vm=25+4* (E)CX	17+5*(E)CX, pm*1=11+ 5*(E)CX, pm*2=31+ 5*(E)CX	5+12*(E)CX, pm*1=6+ 5*(E)CX, pm*2=26+ 5*(E)CX			rep outsd
F2	AC	REP LODS AL	7*3,7+ 3*(E)CX*6	5*3,7+ 4*(E)CX*6				rep lodsb
F2	AD	REP LODS AX	7*3,7+ 3*(E)CX*6	5*3,7+ 4*(E)CX*6				rep lodsw
F2	AD	REP LODS EAX	7*3,7+ 3*(E)CX*6	5*3,7+ 4*(E)CX*6				rep lodsd
F2	AA	REP STOS m8	6*3,9+ (E)CX*6	5*3,7+ 4*(E)CX*6	5+5*(E)CX	4+3*CX	9+10*CX	rep stosb
F2	AB	REP STOS m16	6*3,9+ (E)CX*6	5*3,7+ 4*(E)CX*6	5+5*(E)CX	4+3*CX	9+10*CX	rep stosw
F2	AB	REP STOS m32	6*3,9+ (E)CX*6	5*3,7+ 4*(E)CX*6	5+5*(E)CX			rep stosd
F3	A6	REPE CMPS m8,m8	7*3,9+ 4*(E)CX*6	5*3,7+ 7*(E)CX*6	5+9*N	5+9*N	9+22*N	repe cmpsb
F3	A7	REPE CMPS m16,m16	7*3,9+ 4*(E)CX*6	5*3,7+ 7*(E)CX*6	5+9*N	5+9*N	9+22*N	repe cmpsw
F3	A7	REPE CMPS m32,m32	7*3,9+ 4*(E)CX*6	5*3,7+ 7*(E)CX*6	5+9*N			repe cmpsd
F3	AE	REPE SCAS m8	7*3,9+ 4*(E)CX*6	5*3,7+ 5*(E)CX*6	5+8*N	5+8*N	9+15*N	repe scasb
F3	AF	REPE SCAS m16	7*3,9+ 4*(E)CX*6	5*3,7+ 5*(E)CX*6	5+8*N	5+8*N	9+15*N	repe scasw
F3	AF	REPE SCAS m32	7*3,9+ 4*(E)CX*6	5*3,7+ 5*(E)CX*6	5+8*N			repe scasd
F2	A6	REPNE CMPS m8,m8	7*3,8+ 4*(E)CX*6	5*3,7+ 7*(E)CX*6	5+9*N	5+9*N	9+22*N	repne cmpsb
F2	A7	REPNE CMPS m16,m16	7*3,8+ 4*(E)CX*6	5*3,7+ 7*(E)CX*6	5+9*N	5+9*N	9+22*N	repne cmpsw
F2	A7	REPNE CMPS m32,m32	7*3,8+ 4*(E)CX*6	5*3,7+ 7*(E)CX*6	5+9*N			repne cmpsd
F2	AE	REPNE SCAS m8	7*3,9+ 4*(E)CX*6	5*3,7+ 5*(E)CX*6	5+8*N	5+8*N	9+15*N	repne scasb
F2	AF	REPNE SCAS m16	7*3,9+ 4*(E)CX*6	5*3,7+ 5*(E)CX*6	5+8*N	5+8*N	9+15*N	repne scasw
F2	AF	REPNE SCAS m32	7*3,9+ 4*(E)CX*6	5*3,7+ 5*(E)CX*6	5+8*N			repne scasd

*1 если CPL ≤ IOPL
*2 если CPL > IOPL
*3 если (E)CX = 0
*4 если (E)CX = 1
*5 если (E)CX > 1
*6 если (E)CX > 0

Операция

```
IF AddresSize = 16
THEN использовать CX для CountReg;
ELSE (* AddresSize = 32 *) использовать ECX для CountReg;
FI;
WHILE CountReg ≠ 0
DO
  обслуживание прерываний (если они поступили);
  выполнение основной строковой команды;
  CountReg = CountReg - 1;
  IF (основная команда CMPSB, CMPSW, CMPSD, SCASB, SCASW или SCASD)
  THEN
```

```
    IF (команда REP/REPE/REPZ) AND (ZF = 0)
    THEN exit WHILE loop;
    ELSE
      IF (команда REPNZ или REPNE) AND (ZF = 1)
      THEN exit WHILE loop;
      FI;
    FI;
  FI;
OD;
```

Описание

Префиксы REP, REPE (повторять пока равно) и REPNE (повторять пока не равно) применяются со строковыми операциями. Каждый префикс заставляет строковую команду, которая следует за ним, повторяться указанное в регистре счетчике количество раз или, кроме этого, (для префиксов REPE и REPNE) пока не встретится указанное условие во флаге ZF.

Префиксы REPZ и REPNZ являются синонимами префиксов REPE и REPNE соответственно.

Префиксы REP применяются только к одной строковой команде за один раз. Чтобы повторить блок команд, используйте команду LOOP или другие циклические конструкции.

Ниже указано точное действие для каждой итерации.

1. Если атрибут размера адреса равен 16 бит, то использовать регистр CX в качестве регистра счетчика, если атрибут размера адреса равен 32 бита, то использовать регистр ECX в качестве регистра счетчика.
2. Проверить регистр счетчик. Если он равен 0, то выйти из итерации и перейти к следующей команде.
3. Выполнить любые ожидающие решения прерывания.
4. Исполнить строковую операцию один раз.
5. Уменьшить регистр счетчик на 1. Флаги не изменяются.
6. Проверить флаг ZF, если строковая операция - команда SCAS или CMPS. Если условие повторения не сохранилось, то выйти из итерации и перейти к следующей команде. Выйти из итерации, если префиксом является REPE, и флаг ZF равен 0 (последнее сравнение было неудачным), или если префикс REPNE, и флаг ZF равен 1 (последнее сравнение было удачным).
7. Вернуться к шагу 2 для следующей итерации.

Повторяющиеся команды CMPS и SCAS могут быть завершены, если счетчик исчерпан или флаг ZF не отвечает условию выполнения. Эти два случая могут быть различены ниже при использовании команды JCXZ или

при помощи условных переходов, которые проверяют флаг ZF (команды JZ, JNZ и JNE).

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

Замечание

Не все порты ввода/вывода могут поддерживать скорость, на которой выполняются команды REP INS и REP OUTS.

Не используйте префикс REP с командой LOOP. Правильное действие LOOP не гарантировано при использовании с префиксом REP, и эффект этой комбинации непредсказуем.

Поведение префикса REP при использовании с не строковыми командами неопределено.

Когда происходит страничная ошибка во время выполнения команд CMPS или SCAS с префиксом REPNE, значение EFLAGS восстанавливается в состояние предшествующее выполнению команды. Поскольку SCAS и CMPS не используют EFLAGS в качестве входного значения, то процессор может возобновить команду после обработки страничной ошибки.

RET Возврат из процедуры
RETN
RETF

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
C3	RET	2	5	10+m	11	16	retn
CB	RET	4	13,pm=18	18+m, pm=32+m	15,pm=25	26	retf
CB	RET	23	13,pm=33	10+m, pm=68+m	55		retf
C2 iw	RET imm16	3	5	10+m	11	20	retn 2
CA iw	RET imm16	4	14,pm=17	18+m, pm=32+m	15,pm=25	25	retf 6
CA iw	RET imm16	23	14,pm=33	10+m, pm=68+m	55		retf 16

Операция

```
IF (команда ближний RET)
THEN
  IF OperandSize = 16
  THEN
    IP = Pop();
    EIP = EIP AND 0000FFFFh;
  ELSE (* OperandSize = 32 *)
    EIP = Pop();
  FI;
  IF (команда имеет непосредственный операнд) THEN eSP = eSP + imm16; FI;
FI;
IF (PE = 0 OR (PE = 1 AND VM = 1)) (* реальный или виртуальный-8086
                                      режим *)
  AND (команда дальний RET)
THEN
  IF OperandSize = 16
  THEN
    IP = Pop();
    EIP = EIP AND 0000FFFFh;
    CS = Pop(); (* извлекает из стека 16 бит *)
  ELSE (* OperandSize = 32 *)
    EIP = Pop();
    CS = Pop(); (* извлекает из стека 32 бит , старшие 16 бит очищаются *)
  FI;
  IF (команда имеет непосредственный операнд) THEN eSP = eSP + imm16; FI;
FI;
IF (PE = 1 AND VM = 0) (* защищенный режим, не V86 режим *)
  AND (команда дальний RET)
THEN
  IF OperandSize = 32
  THEN В пределы стека должны попадать верхние 3 слова, иначе #SS(0);
  ELSE В пределы стека должны попадать верхние 2 слова, иначе #SS(0);
  FI;
  Для возвращаемого CS селектора RPL должно быть ≥ CPL,
    иначе #GP(возвращаемый селектор);
  IF (RPL возвращаемого селектора = CPL)
  THEN GOTO RETURN-SAME-LEVEL;
```

```
    ELSE GOTO RETURN-OUTER-LEVEL;
    FI;
  FI;

  SAME-LEVEL:
    Возвращаемый CS селектор должен быть не нулевым, иначе #GP(0);
    Индекс селектора должен попадать в пределы таблицы дескрипторов,
      иначе #GP(возвращаемый селектор);
    AR байт должен задавать кодовый сегмент,
      иначе #GP(возвращаемый селектор);
    IF (несогласованный)
    THEN DPL кодового сегмента = CPL, иначе #GP(возвращаемый селектор);
    FI;
    IF (согласованный)
    THEN
      IF (DPL > CPL) THEN #GP(возвращаемый селектор); FI;
      Сегмент должен присутствовать, иначе #NP(возвращаемый селектор);
      Верхнее слово в стеке должно попадать в пределы сегмента SS,
        иначе #SS(0);
      Указатель инструкции должен попадать в пределы сегмента CS,
        иначе #GP(0);
    FI;
    IF OperandSize = 32
    THEN
      Загрузить CS:EIP из стека;
      Загрузить новый дескриптор регистра CS;
      Увеличить eSP на 8 плюс непосредственное смещение, если оно задано;
    ELSE (* OperandSize = 16 *)
      Загрузить CS:IP из стека;
      Загрузить новый дескриптор регистра CS;
      Увеличить eSP на 4 плюс непосредственное смещение, если оно задано;
    FI;

  OUTER-PRIVILEGE-LEVEL:
    IF OperandSize = 32
    THEN В пределы стека должны попадать верхние 16 + imm16 байт,
      иначе #SS(0);
    ELSE В пределы стека должны попадать верхние 8 + imm16 байт,
      иначе #SS(0);
    FI;
    Проверка возвращаемого CS селектора и дескриптора кодового сегмента:
      Селектор должен быть не нулевым, иначе #GP(0);
      Индекс селектора должен попадать в пределы таблицы дескрипторов,
        иначе #GP(возвращаемый селектор);
      AR байт дескриптора задает кодовый сегмент,
        иначе #GP(возвращаемый селектор);
    IF (несогласованный)
    THEN
      DPL кодового сегмента должно быть = RPL CS селектора,
        иначе #GP(возвращаемый селектор);
    FI;
    IF (согласованный)
    THEN
      DPL кодового сегмента должно быть ≤ RPL возвращаемого CS селектора,
        иначе #GP(возвращаемый селектор);
    FI;

    Сегмент должен присутствовать, иначе #NP(возвращаемый селектор);
    Проверка возвращаемого SS селектора и дескриптора стека:
      Селектор должен быть не нулевым, иначе #GP(0);
      Индекс селектора должен попадать в пределы таблицы дескрипторов,
        иначе #GP(SS селектор);
      RPL селектора должен быть равен RPL возвращаемого CS селектора,
        иначе #GP(SS селектор);
```

```
    AR байт дескриптора должен задавать разрешенный для записи сегмент
       данных, иначе #GP(SS селектор);
    DPL сегмента стека должен быть равен RPL возвращаемого CS селектора,
      иначе #GP(SS селектор);
    Сегмент должен присутствовать, иначе #NP(SS селектор);
    Указатель инструкции должен попадать в пределы сегмента CS,
      иначе #GP(0);
  Установить CPL равным RPL в возвращаемом CS селекторе;
  IF OperandSize = 32
  THEN
    Загрузить CS:EIP из стека;
    Загрузить CPL равным RPL в CS селекторе;
    Увеличить eSP на 8 + imm16;
    Загрузить SS:eSP из стека;
  ELSE
    Загрузить CS:IP из стека;
    Загрузить CPL равным RPL в CS селекторе;
    Увеличить eSP на 4 + imm16;
    Загрузить SS:eSP из стека;
  FI;
  Загрузить дескриптор CS в скрытую часть регистра CS;
  Загрузить дескриптор SS в скрытую часть регистра SS;
  FOR каждого из регистров ES, FS, GS и DS
  DO
    IF (значение регистра недопустимо для текущего уровня)
    THEN обнулить регистр (селектор = AR = 0);
    FI;
    Быть допустимым для текущего уровня - значит отвечать следующим
                                                          условиям:
      Индекс селектора должен попадать в пределы таблицы дескрипторов;
      AR байт дескриптора должен задавать разрешенный для чтения сегмент;
      IF (сегмент это сегмент данных или несогласованный кодовый сегмент)
      THEN DPL должен быть ≥ CPL, или DPL должен быть ≥ RPL;
      FI;
  OD;
```

Описание

Команда RET передает управление по адресу возврата находящемуся в стеке. Этот адрес обычно помещается в стек командой CALL. Необязательный числовой параметр команды RET указывает количество байтов, которые освобождаются в стеке после извлечения адреса возврата.

Для внутрисегментного (ближнего) возврата адрес в стеке является смещением в сегменте, которое извлекается в указатель команд. Регистр CS не изменяется. Для межсегментного (дальнего) возврата адрес в стеке является дальним указателем. Смещение извлекается первым, затем следует селектор.

В реальном режиме регистры CS и IP загружаются непосредственно. В защищенном режиме межсегментный возврат заставляет процессор проверить дескриптор адресуемый селектором возврата. Байт AR дескриптора должен указывать кодовый сегмент с равным или меньшим уровнем привилегий (большим или равным числовым значением), чем текущий.

Возврат к меньшему уровню привилегий вызывает перезагрузку стека значением сохраненным вне блока параметров (значение берется из TSS).

Сегментные регистры DS, ES, FS и GS могут быть очищены командой RET во время возврата к меньшему уровню привилегий. Если значения селекторов в этих регистрах задают сегменты, которые не могут быть использованы на новом уровне привилегий (определяется путем анализа соответствующих дескрипторов), то регистры очищаются для предотвращения неразрешенного доступа с нового уровня привилегий.

Особые ситуации защищенного режима

#GP, #NP или #SS, как описано в разделе "Операция". #PF(код ошибки), для страничной ошибки.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка.

ROL Циклический сдвиг влево
ROR Циклический сдвиг вправо

Смотрите команду RCL.

RSM Выйти из режима управления системой (System Managment Mode)

O	D	I	T	S	Z	A	P	C
*	*	*	*	*	*	*	*	*

Код операции	Команда	Такты Pentium	Пример
0F AA	RSM	83	rsm

Описание

Состояние процессора восстанавливается из дампа созданного в момент входа в SMM (режим управления системой). Заметьте однако, что содержимое особых регистров модели (MSR) не задействуется. Процессор покидает режим SMM и возвращает управление прерванному приложению или операционной системе. Если процессор обнаруживает любую информацию о недопустимом состоянии, то он входит в состояние останова. Это происходит в любой из следующих ситуаций:

- Значение сохраненное в поле дампа не является адресом, выровненным по 32K.
- Любой из зарезервированных битов в CR4 установлен в 1.
- Комбинаций битов в CR0 является некорректной (PG = 1, PE = 0 или NW = 1, CD = 0).

Особые ситуации защищенного режима

#UD, если произведена попытка выполнить эту команду, когда процессор не находится в режиме SMM.

Особые ситуации режима реальной адресации

Прерывание 6, если произведена попытка выполнить эту команду, когда процессор не находится в режиме SMM.

Особые ситуации режима V86

#UD, если произведена попытка выполнить эту команду, когда процессор не находится в режиме SMM.

Замечание

Режим SMM предназначен для выполнения функций высокого уровня управления системой, таких как управление потреблением или безопасность. Процессор переходит в этот режим по специальному сигналу прерывания на выводе SMI#. Режим SMM во всем аналогичен режиму реальной адресации за исключением того, что возможна адресация всего 4 Гбайтного адресного пространства. За подробной информацией по этому режиму обращайтесь к документации Intel.

SAHF Записать содержимое AH в регистр FLAGS

O	D	I	T	S	Z	A	P	C
				*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
9E	SAHF	2	2	3	2	4	sahf

Операция

SF:ZF:xx:AF:xx:PF:xx:CF = AH;

Описание

Команда SAHF загружает флаги SF, ZF, AF, PF и CF значениями из регистра AH, из битов 7, 6, 4, 2 и 0 соответственно.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

SAL Арифметический сдвиг влево
SAR Арифметический сдвиг вправо
SHL Логический сдвиг влево
SHR Логический сдвиг вправо

O	D	I	T	S	Z	A	P	C
*				*	*	?	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
D0 /4	SAL r/m8,1	1/3	3/4	3/7	2/7	2/15+EA	sal al,1
D2 /4	SAL r/m8,CL	4	3/4	3/7	5/8	8+4 per bit/(20+4	sal ah,cl

						per bit)+EA	sal byteX,cl
C0 /4 ib	SAL r/m8,imm8	1/3	2/4	3/7	5/8		sal dh,4
D1 /4	SAL r/m16,1	1/3	3/4	3/7	2/7	2/15+EA	sal ax,1
D3 /4	SAL r/m16,CL	4	3/4	3/7	5/8	8+4 per bit/(2[illegible]	sal ax,cl
						per bit)+EA	sal wordZ,cl
C1 /4 ib	SAL r/m16,imm8	1/3	2/4	3/7	5/8		sal dx,4
D1 /4	SAL r/m32,1	1/3	3/4	3/7	2/7	2/15+EA	sal eax,1
D3 /4	SAL r/m32,CL	4	3/4	3/7	5/8	8+4 per bit/(20+4	sal eax,cl
						per bit)+EA	sal dwY,cl
C1 /4 ib	SAL r/m32,imm8	1/3	2/4	3/7	5/8		sal edx,4
D0 /7	SAR r/m8,1	1/3	3/4	3/7	2/7	2/15+EA	sal al,1
D2 /7	SAR r/m8,CL	4	3/4	3/7	5/8	8+4 per bit/(20+4	sal ah,cl
						per bit)+EA	sal byteX,cl
C0 /7 ib	SAR r/m8,imm8	1/3	2/4	3/7	5/8		sal dh,4
D1 77	SAR r/m16,1	1/3	3/4	3/7	2/7	2/15+EA	sal ax,1
D3 /7	SAR r/m16,CL	4	3/4	3/7	5/8	8+4 per bit/(20+4	sal ax,cl
						per bit)+EA	sal wordZ,cl
C1 /7 ib	SAR r/m16,imm8	1/3	2/4	3/7	5/8		sal dx,4
D1 /7	SAR r/m32,1	1/3	3/4	3/7	2/7	2/15+EA	sal eax,1
D3 /7	SAR r/m32,CL	4	3/4	3/7	5/8	8+4 per bit/(20+4	sal eax,cl
						per bit)+EA	sal dwY,cl
C1 /7 ib	SAR r/m32,imm8	1/3	2/4	3/7	5/8		sal edx,4
D0 /4	SHL r/m8,1	1/3	3/4	3/7	2/7	2/15+EA	shl al,1
D2 /4	SHL r/m8,CL	4	3/4	3/7	5/8	8+4 per bit/(20+4	shl ah,cl
						per bit)+EA	shl byteX,cl
C0 /4 ib	SHL r/m8,imm8	1/3	2/4	3/7	5/8		shl dh,4
D1 /4	SHL r/m16,1	1/3	3/4	3/7	2/7	2/15+EA	shl ax,1
D3 /4	SHL r/m16,CL	4	3/4	3/7	5/8	8+4 per bit/(20+4	shl ax,cl
						per bit)+EA	shl wordZ,cl
C1 /4 ib	SHL r/m16,imm8	1/3	2/4	3/7	5/8		shl dx,4
D1 /4	SHL r/m32,1	1/3	3/4	3/7	2/7	2/15+EA	shl eax,1
D3 /4	SHL r/m32,CL	4	3/4	3/7	5/8	8+4 per bit/(20+4	shl eax,cl
						per bit)+EA	shl dwY,cl
C1 /4 ib	SHL r/m32,imm8	1/3	2/4	3/7	5/8		shl edx,4
D0 /5	SHR r/m8,1	1/3	3/4	3/7	2/7	2/15+EA	shl al,1
D2 /5	SHR r/m8,CL	4	3/4	3/7	5/8	8+4 per bit/(20+4	shl ah,cl
						per bit)+EA	shl byteX,cl
C0 /5 ib	SHR r/m8,imm8	1/3	2/4	3/7	5/8		shr dh,4
D1 /5	SHR r/m16,1	1/3	3/4	3/7	2/7	2/15+EA	shr ax,1
D3 /5	SHR r/m16,CL	4	3/4	3/7	5/8	8+4 per bit/(20+4	shr ax,cl
						per bit)+EA	shr wordZ,cl
C1 /5 ib	SHR r/m16,imm8	1/3	2/4	3/7	5/8		shr dx,4
D1 /5	SHR r/m32,1	1/3	3/4	3/7	2/7	2/15+EA	shr eax,1
D3 /5	SHR r/m32,CL	4	3/4	3/7	5/8	8+4 per bit/(20+4	shr eax,cl
						per bit)+EA	shr dwY,cl
C1 /5 ib	SHR r/m32,imm8	1/3	2/4	3/7	5/8		shr edx,4

Операция

```
(* COUNT это второй операнд *)
(temp) = COUNT;
WHILE (temp ≠ 0)
DO
  IF (команда SAL или SHL)
  THEN CF = high-order bit of r/m;
  FI;
  IF (команда SAR или SHR)
  THEN CF = low-order bit of r/m;
  FI;
  IF (команда SAL или SHL)
  THEN r/m = r/m * 2;
  FI;
  IF (команда SAR)
  THEN r/m = r/m / 2;
    (* Знаковое деление с округлением в сторону минус бесконечности *)
  FI;
  IF (команда SHR)
  THEN r/m = r/m / 2; (* Беззнаковое деление *)
  FI;
  temp = temp - 1;
OD;
(* Установить переполнение для разных команд *)
IF COUNT = 1
```

```
THEN
  IF (команда SAL или SHL)
  THEN OF = high-order bit of r/m ≠ (CF);
  FI;
  IF (команда SAR)
  THEN OF = 0;
  FI;
  IF (команда SHR)
  THEN OF = high-order bit of operand;
  FI;
ELSE (OF неопределен);
FI;
```

Описание

Команда SAL (и ее аналог SHL) сдвигает биты операнда вверх. Старший бит сдвигается во флаг CF, а младший бит очищается.

Команды SAR и SHR сдвигают биты операнда вниз. Младший бит сдвигается во флаг CF. Получаемый эффект - деление операнда на два. Команда SAR выполняет знаковое деление с округлением в сторону -∞ (не тоже самое, что команда IDIV), старший бит остается неизменным. Команда SHR выполняет беззнаковое деление, старший бит очищается.

Сдвиг повторяется столько раз, сколько указано вторым операндом, который может быть либо непосредственным числом, либо содержимым регистра CL. Чтобы уменьшить максимальное время выполнения команды процессор не позволяет счетчику сдвигов быть больше 31. Если счетчик сдвигов больше 31, используются только младшие 5 бит. (Процессор 8086 (8088) не маскирует счетчик сдвигов. Процессоры Intel386, Intel486 и Pentium во всех режимах, в том числе в режиме V86 маскируют счетчик сдвигов.

Флаг OF определен только для форм команд однократного сдвига (вторым операндом является 1). Он неопределен во всех остальных случаях. Для сдвигов и циклических сдвигов влево - бит CF после сдвига складывается по модулю два (XOR) со старшим битом результата, чтобы получить флаг OF. Для команды SAR - флаг OF очищается для всех одиночных сдвигов. Для команды SHR - флаг OF устанавливается в значение старшего бита исходного операнда.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сег-

менте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

SBB Целочисленное вычитание с заемом

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
1C ib	SBB AL,imm8	1	1	2	3	4	sbb al,0AAh
1D iw	SBB AX,imm16	1	1	2	3	4	sbb ax,0FODh
1D id	SBB EAX,imm32	1	1	2			sbb eax,23456789h
80 /3 ib	SBB r/m8,imm8	1/3	1/3	2/7	3/7	4/17+EA	sbb ah,5
81 /3 iw	SBB r/m16,imm16	1/3	1/3	2/7	3/7	4/17+EA	sbb dx,0DBBh
81 /3 id	SBB r/m32,imm32	1/3	1/3	2/7			sbb edx,0CAAAAh
83 /3 ib	SBB r/m16,imm8	1/3	1/3	2/7	3/7	4/17+EA	sbb cx,0AAh
83 /3 ib	SBB r/m32,imm8	1/3	1/3	2/7			sbb ecx,02h
18 /r	SBB r/m8,r8	1/3	1/3	2/6	2/7	3/16+EA	sbb [di],ah
19 /r	SBB r/m16,r16	1/3	1/3	2/6	2/7	3/16+EA	sbb bx,si
19 /r	SBB r/m32,r32	1/3	1/3	2/6			sbb memory,eax
1A /r	SBB r8,r/m8	1/2	1/2	2/7	2/7	3/9+EA	sbb dl,sum
1B /r	SBB r16,r/m16	1/2	1/2	2/7	2/7	3/9+EA	sbb di,[si+12]
1B /r	SBB r32,r/m32	1/2	1/2	2/7			sbb ecx,raznost

Операция

```
IF (SRC байт и DEST слово или двойное слово)
THEN DEST = DEST - (SignExtend(SRC) + CF);
ELSE DEST = DEST - (SRC + CF);
FI;
```

Описание

Команда SBB складывает второй операнд (SRC) с флагом CF и вычитает результат из первого операнда (DEST). Результат вычитания при-

сваивается первому операнду (DEST) и флаги устанавливаются соответствующим образом.

Когда вычитается однобайтное непосредственное значение из двухбайтного операнда, тогда непосредственное значение сперва знакорасширяется.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

SCAS SCASB SCASW SCASD Сравнить строку данных

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
AE	SCAS m8	4	6	7	7	15	scasb
AF	SCAS m16	4	6	7	7	15	scasw
AF	SCAS m32	4	6	7			scasd
AE	SCASB	4	6	7	7	15	scasb
AF	SCASW	4	6	7	7	15	scasw
AF	SCASD	4	6	7			scasd

Операция

```
IF AddressSize = 16
THEN (использовать DI для dest-index);
ELSE (* AddressSize = 32 *) (использовать EDI для dest-index);
FI;
IF (команда байтового типа)
THEN
  AL - [dest-index]; (* Сравнивает байт в AL и dest *)
  IF DF = 0 THEN IncDec = 1; ELSE IncDec = -1; FI;
ELSE
  IF OperandSize = 16
  THEN
    AX - [dest-index]; (* Сравнивает слово в AX и dest *)
    IF DF = 0 THEN IncDec = 2; ELSE IncDec = -2; FI;
  ELSE
    EAX - [dest-index]; (* Сравнивает двойное слово в EAX и dest *)
    IF DF = 0 THEN IncDec = 4; ELSE IncDec = -4; FI;
  FI;
FI;
dest-index = dest-index + IncDec;
```

Описание

Команда SCAS вычитает указанный регистром назначения байт, слово или двойное слово в памяти из регистра AL, AX или EAX. Результат отбрасывается, устанавливаются только флаги. Операнд адресуется в сегменте ES, замещение сегмента невозможно.

Если атрибут размера адреса для этой команды равен 16 бит, то в качестве регистра назначения используется регистр DI, если атрибут размера адреса равен 32 бита, то используется регистр EDI.

Адрес сравниваемых данных определяется только содержимым регистра назначения, но не операндом команды SCAS. Операнд определяет тип данных (байт, слово, двойное слово). Загружайте корректное значение индекса в регистр (E)DI перед выполнением команды SCAS.

После завершения сравнения индексный регистр назначение автоматически изменяется. Если флаг DF равен 0 (была выполнена команда CLD), то регистр увеличивается; если флаг DF равен 1 (была выполнена команда STD), то регистр уменьшается. Регистр увеличивается или уменьшается на 1 при сравнении байтов, на 2 при сравнении слов, и на 4 при сравнении двойных слов.

Команды SCASB, SCASW, SCASD являются синонимами команды SCAS для байтов, слов или двойных слов соответственно. Они проще программируются на ассемблере, но не обеспечивают проверку типа данных или сегмента.

Команде SCAS может предшествовать префикс REPE или REPNE для сравнения блока из (E)CX байтов, слов или двойных слов. Обратитесь к

описанию команды REP для более подробной информации по этой операции.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегменте ES; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

SETcc Установка байта при условии

O D I T S Z A P C

Код операции	Команда	Такты			Условие установки байта
		Pentium	486	386	
0F 97	SETA r/m8	1/2	4/3	4/5	Если выше (CF=0 и ZF=0)
0F 93	SETAE r/m8	1/2	4/3	4/5	Если выше или равно (CF=0)
0F 92	SETB r/m8	1/2	4/3	4/5	Если ниже (CF=1)
0F 96	SETBE r/m8	1/2	4/3	4/5	Если ниже или равно (CF=1 или ZF=1)
0F 92	SETC r/m8	1/2	4/3	4/5	Если перенос (CF=1)
0F 94	SETE r/m8	1/2	4/3	4/5	Если равно (ZF=1)
0F 9F	SETG r/m8	1/2	4/3	4/5	Если больше (ZF=0 и SF=OF)
0F 9D	SETGE r/m8	1/2	4/3	4/5	Если больше или равно (SF=OF)
0F 9C	SETL r/m8	1/2	4/3	4/5	Если меньше (SF<>OF)
0F 9E	SETLE r/m8	1/2	4/3	4/5	Если меньше или равно (ZF=1 или SF<>OF)
0F 96	SETNA r/m8	1/2	4/3	4/5	Если не выше (CF=1 или ZF=1)
0F 92	SETNAE r/m8	1/2	4/3	4/5	Если не выше или равно (CF=1)
0F 93	SETNB r/m8	1/2	4/3	4/5	Если не ниже (CF=0)
0F 97	SETNBE r/m8	1/2	4/3	4/5	Если не ниже или равно (CF=0 и ZF=0)
0F 93	SETNC r/m8	1/2	4/3	4/5	Если не перенос (CF=0)
0F 95	SETNE r/m8	1/2	4/3	4/5	Если не равно (ZF=0)
0F 9E	SETNG r/m8	1/2	4/3	4/5	Если не больше (ZF=1 или SF<>OF)
0F 9C	SETNGE r/m8	1/2	4/3	4/5	Если не больше или равно (SF<>OF)
0F 9D	SETNL r/m8	1/2	4/3	4/5	Если не меньше (SF=OF)
0F 9F	SETNLE r/m8	1/2	4/3	4/5	Если не меньше или равно (ZF=0 и SF=OF)
0F 91	SETNO r/m8	1/2	4/3	4/5	Если не переполнение (OF=0)

0F 9B	SETNP r/m8	1/2	4/3	4/5	Если нечетно (PF=0)
0F 99	SETNS r/m8	1/2	4/3	4/5	Если положительно (SF=0)
0F 95	SETNZ r/m8	1/2	4/3	4/5	Если не нуль (ZF=0)
0F 90	SETO r/m8	1/2	4/3	4/5	Если переполнение (OF=1)
0F 9A	SETP r/m8	1/2	4/3	4/5	Если четно (PF=1)
0F 9A	SETPE r/m8	1/2	4/3	4/5	Если четно (PF=1)
0F 9B	SETPO r/m8	1/2	4/3	4/5	Если нечетно (PF=0)
0F 98	SETS r/m8	1/2	4/3	4/5	Если отрицательно (SF=1)
0F 94	SETZ r/m8	1/2	4/3	4/5	Если нуль (ZF=1)

Операция

IF condition THEN r/m8 = 1; ELSE r/m8 = 0; FI;

Описание

Команда SETcc записывает байт со значением 1 в назначение, указанное операндом, если условие выполнено, или байт со значением 0 - если условие не выполнено.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

SGDT Сохранить регистр глобальной таблицы дескрипторов (GDTR)
SIDT Сохранить регистр таблицы дескрипторов прерываний (IDTR)

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
0F 01 /0	SGDT m	4	10	9	11	sgdt [description]
0F 01 /1	SIDT m	4	10	9	12	sidt [bx+2]

Операция

DEST = 48 бит (база/предел) содержимого регистра;

Описание

Команды SGDT и SIDT копируют содержимое регистра таблицы дескрипторов в шесть байтов памяти, указанных операндом. Поле "Предел"(LIMIT) регистра присваивается первому слову, находящемуся по эффективному адресу. Если атрибут размера операнда равен 16 бит, то следующим трем байтам присваивается значение поля "База"(BASE), а четвертый байт неопределен. Если атрибут размера операнда равен 32 бита, то следующим четырем байтам присваивается значение 32 битного поля "База"(BASE) регистра.

Команды SGDT и SIDT используются только системным программным обеспечением и не используются в прикладных программах.

Особые ситуации защищенного режима

#UD, если операнд назначения является регистром; #GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 6, если операнд назначения является регистром. Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание о совместимости

16 битные формы команды SGDT и SIDT совместимы с процессором Intel286, если значение верхних восьми битах не используются. Процессор Intel286 сохраняет единицы в этих верхних битах, тогда как 32 битные процессоры сохраняют нули, если атрибут размера операнда равен 16 бит. Эти биты были специфицированы как неопределенные для команд SGDT и SIDT в "80286 Programming Reference Manual (Order No. 210498)".

SHL Логический сдвиг влево

Смотрите команду SAL.

SHLD Логический сдвиг влево двойной точности

O	D	I	T	S	Z	A	P	C
?				*	*	?	*	*

Код операции	Команда	Такты			Пример
		Pentium	486	386	
0F A4	SHLD r/m16,r16,imm8	4	2/3	3/7	shld ax,bx,10
0F A4	SHLD r/m32,r32,imm8	4	2/3	3/7	shld eax,ecx,4
0F A5	SHLD r/m16,r16,CL	4/5	2/3	3/7	shld [di],bx,cl
0F A5	SHLD r/m32,r32,CL	4/5	2/3	3/7	shld [ebx],eax,cl

Операция

```
(* Count представляет собой беззнаковое целое, согласованное с последним
      операндом команды (байтом imm8 или байтом в регистре CL) *)
ShiftAmt = count MOD 32;
InBits = register; (* Допускается частичное перекрытие операндов *)
IF ShiftAmt = 0
THEN (нет операции)
ELSE
  IF ShiftAmt ≥ OperandSize
  THEN (* плохие параметры *)
  r/m = UNDEFINED;
  CF, OF, SZ, ZF, AF, PF = UNDEFINED;
  ELSE (* Выполнить сдвиг *)
    CF = BIT[Base, OperandSize - ShiftAmt]; (* Последний бит выдвигается *)
    FOR i = OperandSize - 1 DOWNTO ShiftAmt
    DO BIT[Base,i] = BIT[Base,i-ShiftAmt]; OD;
    FOR i = ShiftAmt - 1 DOWNTO 0
    DO BIT[Base,i] = BIT[Base,i - ShiftAmt + OperandSize]; OD;
    Установить SF, ZF, PF (r/m);
       (* SF, ZF, PF  устанавливаются согласно значению результата *)
    AF = UNDEFINED;
  FI;
FI;
```

Описание

Команда SHLD сдвигает первый операнд, задаваемый полем r/m, влево на столько бит, на сколько указывает операнд счетчик. Второй операнд (r16 или r32) содержит биты, которые вдвигаются справа (начиная с бита 0). Результат сохраняется обратно в r/m операнд, а второй операнд регистр остается неизменным.

Операнд счетчик обеспечивается либо непосредственным байтом, либо содержимым регистра CL. Значение счетчика берется по модулю 32, чтобы обеспечить число между 0 и 31, на которое происходит сдвиг. Эта команда полезна для сдвигов двойной точности (64 бита или более).

Флаги SF, ZF и PF устанавливаются согласно значению результата. Флаг CF устанавливается в значение последнего выдвинутого бита. Флаги OF и AF оставляются неопределенными.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

SHR Логический сдвиг вправо

Смотрите команду SAL.

SHRD Логический сдвиг вправо двойной точности

O	D	I	T	S	Z	A	P	C
?				*	*	?	*	*

Код операции	Команда	Такты			Пример
		Pentium	486	386	
0F AC	SHRD r/m16,r16,imm8	4	2/3	3/7	shrd ax,bx,10
0F AC	SHRD r/m32,r32,imm8	4	2/3	3/7	shrd eax,ecx,4
0F AD	SHRD r/m16,r16,CL	4/5	3/4	3/7	shrd [di],bx,cl
0F AD	SHRD r/m32,r32,CL	4/5	3/4	3/7	shrd [ebx],eax,cl

Операция

```
(* Count представляет собой беззнаковое целое, согласованное с последним
      операндом команды (байтом imm8 или байтом в регистре CL) *)
ShiftAmt = count MOD 32;
InBits = register;  (* Допускается частичное перекрытие операндов *)
IF ShiftAmt = 0
THEN (нет операции)
ELSE
  IF ShiftAmt ≥ OperandSize
  THEN (* плохие параметры *)
   r/m = UNDEFINED;
   CF, OF, SZ, ZF, AF, PF = UNDEFINED;
  ELSE (* Выполнить сдвиг *)
    CF = BIT[r/m,ShiftArm - 1]; (* Последний бит выдвигается *)
    FOR i = 0 TO OperandSize - 1 - ShiftAmt
    DO BIT[r/m,i] = BIT[r/m,i-ShiftAmt]; OD;
    FOR i = OperandSize - ShiftAmt TO OperandSize - 1
    DO BIT[r/m,i] = BIT[InBits,i + ShiftAmt - OperandSize]; OD;
```

```
        (* SF, ZF, PF устанавливаются согласно значению результата *)
      Установить SF, ZF, PF (r/m);
      AF = UNDEFINED;
    FI;
  FI;
```

Описание

Команда SHRD сдвигает первый операнд, задаваемый полем r/m, вправо на столько бит, на сколько указывает операнд счетчик (третий операнд). Второй операнд (r16 или r32) содержит биты, которые вдвигаются слева (начиная с бита 31). Результат сохраняется обратно в r/m операнд, а второй операнд регистр остается неизменным.

Операнд счетчик (третий операнд) обеспечивается либо непосредственным байтом, либо содержимым регистра CL. Значение счетчика берется по модулю 32, чтобы обеспечить число между 0 и 31, на которое происходит сдвиг. Эта команда полезна для сдвигов двойной точности (64 бита или более).

Флаги SF, ZF и PF устанавливаются согласно значению результата. Флаг CF устанавливается в значение последнего выдвинутого бита. Флаги OF и AF оставляются неопределенными.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

SIDT Сохранить регистр таблицы дескрипторов прерываний (IDTR)

Смотрите команду SGDT.

SLDT Сохранить регистр локальной таблицы дескрипторов (LDTR)

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
0F 00 /0	SLDT r/m16	2	2/3	pm=2/2	2/3	sldt ax

Операция

r/m16 = LDTR;

Описание

Команда SLDT сохраняет регистр локальной таблицы дескрипторов (LDTR) в двухбайтном регистре или в слове памяти, указанном операндом. Этот регистр является селектором, указывающим на глобальную таблицу дескрипторов (GDT).

Команда SLDT используется только в системном программном обеспечении и не используется в прикладных программах.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0) если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 6, команда не распознается в реальном режиме.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации (из-за того, что команда не распознается, она не выполняется, и не производится обращений к памяти).

Замечание

Когда назначением является 32 битный регистр, тогда 16 битный операнд источник копируется в младшие 16 бит назначения, а старшие 16 бит регистра остаются неопределенными. С 16 битным регистровым операндом используются только младшие 16 бит назначения (старшие 16 бит остаются неизменными). С операндом в памяти источник записывается в память как 16 битная величина, несмотря на размер операнда. Как результат, 32 битное программное обеспечение должно всегда обращаться с этой величиной как с 16 битной и маскировать биты 16..31, если необходимо.

SMSW Сохранить слово состояния машины (MSW)

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
0F 01 /4	SMSW r/m16	4	2/3	2/3,pm=2/2	2/3	smsw ax

Операция

```
r/m16 = MSW;
```

Описание

Команда SMSW сохраняет слово состояния машины (часть регистра CR0) в двухбайтном регистре или слове памяти.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Этой командой обеспечивается совместимость с процессором Intel286. Программы для 32 битных процессоров должны использовать команду MOV ...,CR0.

Когда назначением является 32 битный регистр, тогда 16 битный операнд источник копируется в младшие 16 бит назначения, а старшие 16 бит регистра остаются неопределенными. С 16 битным регистровым операндом используются только младшие 16 бит назначения (старшие 16 бит остаются неизменными). С операндом в памяти источник записывается в память как 16 битная величина, несмотря на размер операнда. Как результат, 32 битное программное обеспечение должно всегда обращаться с этой величиной как с 16 битной и маскировать биты 16..31, если необходимо.

STC Установить флаг переноса

O	D	I	T	S	Z	A	P	C
								1

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
F9	STC	2	2	2	2	2	stc

Операция

CF = 1;

Описание

Команда STC устанавливает флаг CF.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

STD Установить флаг направления

O	D	I	T	S	Z	A	P	C
	1							

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
FD	STD	2	2	2	2	2	std

Операция

DF = 1;

Описание

Команда STD устанавливает флаг DF, заставляющий все последующие строковые операции уменьшать индексные регистры (E)SI и (E)DI, с которыми они оперируют.

Особые ситуации защищенного режима

Нет.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

Нет.

STI Разрешить внешние прерывания

O	D	I	T	S	Z	A	P	C
		1						

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
FB	STI	7	5	3	2	2	sti

Операция

```
IF PE = 0   (* Выполняется в режиме реальной адресации *)
THEN IF = 1; (* Установить флаг прерываний *)
ELSE (* Выполняется в защищенном режиме или виртуальном-8086 режиме *)
  IF VM = 0 (* Выполняется в защищенном режиме *)
  THEN
    IF IOPL = 3
    THEN IF = 1; (* Установить флаг прерываний *)
    ELSE
      IF CPL ≤ IOPL
        THEN IF = 1; (* Установить флаг прерываний *)
        ELSE #GP(0);
      FI;
    FI;
  ELSE (* Выполняется в виртуальном-8086 режиме *)
    #GP(0); (* ловушка монитора V86 *)
  FI;
FI;
```

Таблица решений

Данная таблица решений указывает какое действие из нижней части таблицы произойдет при условиях, стоящих в верхней части.

PE =	0	1	1	1	1
VM =	-	0	-	0	1
CPL	-	≤ IOPL	-	> IOPL	-
IOPL	-	-	= 3	-	< 3
IF ← 0	Y	Y	Y		
#GP(0)				Y	Y

Замечание:

-	не влияет;
пусто	действие не происходит;
Y	произошло действие из первой колонки.

Описание

Команда STI устанавливает флаг IF, если текущий уровень привилегий, как минимум, такой же, как и уровень привилегий ввода/вывода (IOPL). Никакие другие флаги не изменяются. После команды следующей за STI процессор начинает отвечать внешним прерываниям, если только эта команда не сбросит флаг IF. Если внешние прерывания были запрещены, и за командой STI следует команда RET, то команде RET позволяется быть выполненной перед тем, как внешние прерывания начнут распознаваться. Также, если внешние прерывания были запрещены и за командой STI следует команда CLI, которая очищает флаг IF, то внешние прерывания не распознаются.

Особые ситуации защищенного режима

#GP(0), если текущий уровень привилегий по значению больше (имеет меньше привилегий), чем уровень привилегий ввода-вывода в регистре флагов. Уровень привилегий ввода-вывода определяет наихудший уровень привилегий, на котором возможен ввод-вывод.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

#GP(0), как и в защищенном режиме.

Замечание

В случае немаскируемого прерывания (NMI), ловушки или ошибки, следующей за STI, прерывание будет иметь место перед выполнением следующей команды.

Информацию об особых ситуациях виртуального режима для данной команды смотрите в Приложении H.

STOS Сохранить строку данных
STOSB
STOSW
STOSD

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
AA	STOS m8	3	5	4	3	11	stosb
AB	STOS m16	3	5	4	3	11	stosw
AB	STOS m32	3	5	4			stosd
AA	STOSB	3	5	4	3	11	stosb
AB	STOSW	3	5	4	3	11	stosw
AB	STOSD	3	5	4			stosd

Операция

```
IF AddressSize = 16
THEN использовать ES:DI для DestReg;
ELSE (* AddressSise = 32 *) использовать ES:EDI для DestReg;
FI;
IF (команда байтового типа)
THEN
  (ES:DestReg) = AL;
  IF DF = 0
  THEN DestReg = DestReg + 1;
  ELSE DestReg = DestReg - 1;
  FI;
ELSE
  IF OperandSize = 16
  THEN
    (ES:DestReg) = AX;
    IF DF = 0
    THEN DestReg = DestReg + 2;
    ELSE DestReg = DestReg - 2;
    FI;
```

```
    ELSE (* OperandSize = 32 *)
      (ES:DestReg) = EAX;
      IF DF = 0
      THEN DestReg = DestReg + 4;
      ELSE DestReg = DestReg - 4;
      FI;
    FI;
FI;
```

Описание

Команда STOS пересылает содержимое регистра AL, AX или EAX в байт, слово или двойное слово в памяти, адресуемое регистром назначения относительно сегмента ES. Регистром назначения является регистр DI (для атрибута размера адреса 16 бит) или EDI (для атрибута размера адреса 32 бита).

Операнд назначения должен адресоваться регистром ES, замещение сегмента невозможно.

Адрес назначения определяется содержимым регистра назначения, а не явным операндом команды STOS. Операнд только определяет тип данных. Загружайте корректное значение индекса в регистр (E)DI перед выполнением команды STOS.

После завершения пересылки индексный регистр назначение автоматически изменяется. Если флаг DF равен 0 (была выполнена команда CLD), то регистр увеличивается; если флаг DF равен 1 (была выполнена команда STD), то регистр уменьшается. Регистр увеличивается или уменьшается на 1 при пересылке байта, на 2 при пересылке слова, и на 4 при пересылке двойного слова.

Команды STOSB, STOSW, STOSD являются синонимами команды STOS для байта, слова или двойного слова соответственно. Они проще программируются на ассемблере, но не обеспечивают проверку типа данных или сегмента.

Команде STOS может предшествовать префикс REP для заполнения блока из (E)CX байтов, слов или двойных слов. Обратитесь к описанию команды REP для более подробной информации по этой операции.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегменте ES; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память при текущем уровне привилегий равном 3.

STR Сохранить регистр задачи

O D I T S Z A P C

Код операции	Команда	Такты				Пример
		Pentium	486	386	286	
0F 00 /1	STR r/m16	2	2/3	pm=23/27	2/3	str bx

Операция

```
r/m = TR; (* регистр задачи *)
```

Описание

Команда STR сохраняет регистр задачи (TR) в двухбайтном регистре или в слове памяти.

Команда STR используется только в системном программном обеспечении. Она не используется в прикладных программах.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 6, команда не распознается в реальном режиме.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации (из-за того, что команда не распознается, она не выполняется, и не производится обращений к памяти).

Замечание

Когда назначением является 32 битный регистр, тогда 16 битный операнд источник копируется в младшие 16 бит назначения, а старшие 16 бит регистра остаются неопределенными. С 16 битным регистровым операндом используются только младшие 16 бит назначения (старшие 16 бит остаются неизменными). С операндом в памяти источник записывается в память как 16 битная величина несмотря на размер операнда. Как результат, 32 битное программное обеспечение должно всегда обращаться с этой величиной как с 16 битным и маскировать биты 16..31, если необходимо.

SUB Целочисленное вычитание

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
2C ib	SUB AL,imm8	1	1	2	3	4	sub al,0AAh
2D iw	SUB AX,imm16	1	1	2	3	4	sub ax,0FODh
2D id	SUB EAX,imm32	1	1	2			sub eax,23456789h
80 /5 ib	SUB r/m8,imm8	1/3	1/3	2/7	3/7	4/17+EA	sub ah,5
81 /5 iw	SUB r/m16,imm16	1/3	1/3	2/7	3/7	4/17+EA	sub dx,0DBBh
81 /5 id	SUB r/m32,imm32	1/3	1/3	2/7			sub edx,0CAAAAh
83 /5 ib	SUB r/m16,imm8	1/3	1/3	2/7	3/7	4/17+EA	sub cx,0AAh
83 /5 ib	SUB r/m32,imm8	1/3	1/3	2/7			sub ecx,02h
28 /r	SUB r/m8,r8	1/3	1/3	2/6	2/7	3/16+EA	sub [di],ah
29 /r	SUB r/m16,r16	1/3	1/3	2/6	2/7	3/16+EA	sub bx,si
29 /r	SUB r/m32,r32	1/3	1/3	2/6			sub memory,eax
2A /r	SUB r8,r/m8	1/2	1/2	2/7	2/7	3/9+EA	sub dl,sum
2B /r	SUB r16,r/m16	1/2	1/2	2/7	2/7	3/9+EA	sub di,[si+12]
2B /r	SUB r32,r/m32	1/2	1/2	2/7			sub ecx,raznost

Операция

```
IF (SRC - байт, а DEST - слово или двойное слово)
THEN DEST = DEST - SignExtend(SRC);
ELSE DEST = DEST - SRC;
FI;
```

Описание

Команда SUB вычитает второй операнд (SRC) из первого операнда (DEST). Первому операнду присваивается результат вычитания, и соответствующим образом устанавливаются флаги.

Когда непосредственное однобайтное значение вычитается из двухбайтного операнда, тогда непосредственное значение сперва знакорасширяется до размера операнда назначения.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

TEST Сравнить логически

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
A8 ib	TEST AL,imm8	1	1	2	3	4	test al,0AAh
A9 iw	TEST AX,imm16	1	1	2	3	4	test ax,0FODh
A9 id	TEST EAX,imm32	1	1	2			test eax,2456789h
F6 /0 ib	TEST r/m8,imm8	1/2	1/2	2/5	3/6	5/11+EA	test ah,5
F7 /0 iw	TEST r/m16,imm16	1/2	1/2	2/5	3/6	5/11+EA	test dx,0DBBh
F7 /0 id	TEST r/m32,imm32	1/2	1/2	2/5			test edx,0CAAAAh
84 /r	TEST r/m8,r8	1/2	1/2	2/5	2/6	3/9+EA	test [di],ah
85 /r	TEST r/m16,r16	1/2	1/2	2/5	2/6	3/9+EA	test bx,si
85 /r	TEST r/m32,r32	1/2	1/2	2/5			test memory,eax

Операция

```
TEMP = LeftSRC AND RightSRC; (* Флаги SF, ZF, PF устанавливаются в
      соответствии с TEMP *)
CF = 0; OF = 0;
```

Описание

Команда TEST вычисляет поразрядное логическое "И" своих двух операндов. Каждый бит результата равен 1, если соответствующие биты операндов равны 1, иначе каждый бит равен 0. Результат нигде не запоминается, только соответствующим образом выставляются флаги.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

VERR Проверить сегмент на чтение
VERW Проверить сегмент на запись

O	D	I	T	S	Z	A	P	C
					*			

Код операции	Команда	Pentium	486	Такты 386	286	Пример
0F 00 /4	VERR r/m16	7	11	pm=10/11	14/16	verr cx
0F 00 /5	VERW r/m16	7	11	pm=15/16	14/16	verw dx

Операция

```
IF (сегмент, с указанным в (r/m) селектором, доступен с текущим уровнем
      привилегий) AND ((сегмент доступен по чтению для VERR) OR (сегмент
      доступен по записи для VERW))
THEN ZF = 1;
ELSE ZF = 0;
FI;
```

Описание

Двухбайтный регистр или операнд в памяти команд VERR и VERW содержит значение селектора. Команды VERR и VERW определяют доступен ли сегмент, указанный селектором, на текущем уровне привилегий и доступен ли этот сегмент для чтения (VERR) или для записи (VERW). Если сегмент доступен, то устанавливается флаг ZF, если же сегмент не доступен, то флаг ZF очищается. Для установки флага ZF необходимо выполнение следующих условий.

- Индекс селектора должен попадать в пределы таблицы дескрипторов (GDT или LDT); селектор должен быть определен.
- Селектор должен указывать на дескриптор сегмента кода или данных, но не на дескриптор сегмента состояния задачи, дескриптор LDT или дескриптор шлюза.
- Для команды VERR сегмент должен быть доступен для чтения, а для команды VERW сегмент должен быть доступным для записи сегментом данных.

- Если сегмент кода доступен для чтения и согласован, то уровень привилегий дескриптора (DPL) может быть любым для команды VERR, иначе DPL должен быть больше или равен (иметь меньше или столько же привилегий) и CPL (текущий уровень привилегий) и RPL селектора.

Производимая проверка является такой же, как если бы сегмент был загружен в регистр DS, ES, FS или GS, и был произведен указанный доступ (чтение или запись). Флаг ZF устанавливается по результату проверки. Значение селектора не может приводить к исключениям защиты, предоставляя возможность программному обеспечению предвидеть возможные проблемы доступа к сегменту.

Особые ситуации защищенного режима

Ошибки генерируются недопустимой адресацией операнда в памяти, который содержит селектор. Сам селектор не загружается ни в какой из сегментных регистров, и никаких ошибок, присущих операнду-селектору, не генерируется.

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 6, команды VERR и VERW не распознаются в реальном режиме.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

WAIT Ожидать

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
9B	WAIT	1	1-3	6	3	4+5n	wait

Описание

Команда WAIT заставляет процессор проверить заявленные немаскированные числовые исключения, прежде чем продолжить работу.

Особые ситуации защищенного режима

#NM, если в CR0 установлены и MP и TS.

Особые ситуации режима реальной адресации

Прерывание 7, если в CR0 установлены MP и TS.

Особые ситуации режима V86

#NM, если в CR0 установлены и MP и TS.

Замечание

Программирование WAIT после ESC команды позволяет убедиться, что любое немаскированное исключение с плавающей запятой, которое может вызывать команда, обработается прежде, чем процессор получит возможность изменить результаты команды (Pentium).

FWAIT - дополнительная мнемоника для WAIT.

В процессорах Pentium, Intel486, Intel387 команды сопроцессора автоматически синхронизируются (процессор автоматически ожидает окончания выполнения предыдущей команды сопроцессора перед запуском следующей). Для 8087, который используется совместно с 8086 или 8088, необходимо наличие команды WAIT перед каждой числовой командой для гарантии синхронизации. Программы 8087, имеющие такие синхронизирующие команды WAIT могут без реассемблирования выполняться на 32 разрядных процессорах, однако для сокращения кода и времени выполнения рекомендуется удаление лишних инструкций WAIT.

WBINVD Записать обратно и аннулировать КЭШ

O D I T S Z A P C

Код операции	Команда	Такты		Пример
		Pentium	486	
0F 09	WBINVD	2000+	5	wbinvd

Операция

Сбросить (FLUSH) внутренний КЭШ
Выдать сигнал внешнему КЭШ на обратную запись (WRITE-BACK)
Выдать сигнал внешнему КЭШ на сброс (FLUSH)

Описание

Команда WBINVD очищает содержимое внутреннего КЭШ. Далее производится особый функциональный цикл шины, который указывает, что внешний КЭШ должен записать свое содержимое в основную память (Write-Back). Другой специальный цикл шины, идущий следом, указывает, что внешний КЭШ должен быть очищен.

Особые ситуации защищенного режима

Команда WBINVD - привилегированная команда. #GP(0), если текущий уровень привилегий не равен 0.

Особые ситуации режима реальной адресации

Нет.

Особые ситуации режима V86

#GP(0); Команда WBINVD - привилегированная команда.

Замечание

INVD должна использоваться осторожно. Она не записывает обратно измененные линии КЭШ, поэтому она может вызвать несогласование данных КЭШ с любой другой памятью в системе. Если не имеется особой не-

обходимости не допустить обратную запись измененных линий КЭШ перед их аннулированием (invalidate), то программа должна использовать команду WBINVD. Команда INVD используется при тестировании и устранении ошибок, когда соответствие с основной памятью не нужно.

Эта команда реализационно-зависима. Эта функция может быть реализована иначе на будущих процессорах Intel.

Эта команда не ожидает, пока внешний КЭШ выполнит необходимые действия. На ответственности аппаратного обеспечения лежит обязанность отреагировать на указания обратной записи и недействительности внешнего КЭШ.

Эта команда поддерживается только процессорами i80486 и Pentium (с некоторыми отличиями из-за различной организации КЭШ).

WRMSR Записать в особый регистр модели (Model Specific Register)

O D I T S Z A P C

Код операции	Команда	Такты Pentium	Пример
0F 30	WRMSR	30-45	wrmsr

Операция

MSR[ECX] = EDX:EAX

Описание

Значение в ECX указывает один из 64 битных особых регистров модели процессора Pentium. Содержимое EDX:EAX копируется в особый регистр модели. Из EDX копируются старшие 32 бита, а из EAX - младшие 32 бита.

Следующие значения используются для выбора особых регистров модели в процессоре Pentium:

Величина (Hex)	Название регистра	Описание
00h	Machine Check Address	Сохраняет адрес цикла, вызвавшего исключение
01h	Machine Check Type	Сохраняет тип цикла, вызвавшего исключение

Информацию о других величинах, использующихся при тестировании КЭШ, TLB и BTB, а также при осуществлении контроля, смотрите в Приложении H.

Особые ситуации защищенного режима

#GP(0), если текущий уровень привилегий не 0, либо значение в ECX не указывает особый регистр модели, который реализован в процессоре Pentium.

Особые ситуации режима реальной адресации

Прерывание 13, если значение в ECX не указывает особый регистр модели, который реализован в процессоре Pentium.

Особые ситуации режима V86

#GP(0), при попытке выполнить команду.

Замечание

Всегда устанавливайте неопределенные и зарезервированные биты в прежде прочитанные значения.

Эта команда должна выполняться при уровне привилегий равном 0 или в реальном режиме, иначе будет сгенерирована особая ситуация защиты.

Если в особом регистре модели реализовано меньше чем 64 бита, то значение возвращаемое в EDX:EAX неопределено для нереализованных битов.

WRMSR используется для записи содержимого особых регистров модели, которые контролируют функции тестирования, трассировки выпол-

нения, слежения и проверки ошибок машины. За дополнительной информацией обратитесь к "PentiumTM Processor Data Book".

Значения 03h, 0Fh и значения выше 13h зарезервированы. Не выполняйте WRMSR с зарезервированными значениями в ECX.

XADD Обменять и сложить

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Код операции	Команда	Такты		Пример
		Pentium	486	
0F C0+rb	XADD r/m8,r8	3/4	3/4	xadd al,dl
0F C1+rw	XADD r/m16,r16	3/4	3/4	xadd ax,cx
0F C1+rd	XADD r/m32,r32	3/4	3/4	xadd ebx,ecx

Операция

```
TEMP = SRC + DEST;
SRC = DEST;
DEST = TEMP;
```

Описание

Команда XADD загружает назначение (DEST) в источник (SRC), а затем загружает сумму назначения и источника на место операнда назначения.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если в CR0 установлен бит EM или TS; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Эта команда может использоваться с префиксом LOCK.

XCHG Обмен

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
90+rw	XCHG AX,r16	2	3	3	3	3	xchg ax,bx
90+rw	XCHG r16,AX	2	3	3	3	3	xchg bx,ax
90+rd	XCHG EAX,r32	2	3	3			xchg eax,ebx
90+rd	XCHG r32,EAX	2	3	3			xchg edi,eax
86 /r	XCHG r/m8,r8	3	3/5	3/5	3/5	4/17+EA	xchg dh,dl
86 /r	XCHG r8,r/m8	3	3/5	3/5	3/5	4/17+EA	xchg cl,[bx]
87 /r	XCHG r/m16,r16	3	3/5	3/5	3/5	4/17+EA	xchg [di],dx
87 /r	XCHG r16,r/m16	3	3/5	3/5	3/5	4/17+EA	xchg ax,[bp]
87 /r	XCHG r/m32,r32	3	3/5	3/5			xchg ecx,edx
87 /r	XCHG r32,r/m32	3	3/5	3/5			xchg edi,[esi+5]

Операция

```
temp = DEST;
DEST = SRC;
SRC = temp;
```

Описание

Команда XCHG обменивает два операнда, которые могут присутствовать в любом порядке. Если используется операнд в памяти, то выставляется сигнал LOCK# на время выполнения обмена, несмотря на присутствие или отсутствие префикса LOCK или значения IOPL.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сег-

менте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

XLAT Табличное преобразование
XLATB

O D I T S Z A P C

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
D7	XLAT m8	4	4	5	5	11	xlatb
D7	XLATB	4	4	5	5	11	xlatb

Операция

```
IF AddresSize = 16
THEN
  AL = (BX + ZeroExtend(AL));
ELSE (* AddressSize = 32 *)
  AL = (EBX + ZeroExtend(AL));
FI;
```

Описание

Команда XLAT заменяет табличный индекс из регистра AL на элемент таблицы. Регистр AL должен быть беззнаковым индексом в таблице, адресуемой регистровой парой DS:BX (для атрибута размера адреса равного 16 бит), или регистровой парой DS:EBX (для атрибута размера адреса равного 32 бита).

Операнд команды XLAT допускает возможность замещения сегмента. Команда XLAT использует содержимое регистра (E)BX, даже если оно

отличается от смещения операнда. Смещение операнда должно быть загружено в регистр (E)BX предыдущей командой.

Безоперандная форма команды XLAT может быть использована, если таблица будет всегда находиться в сегменте DS.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память при текущем уровне, привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

XOR Логическое "Исключающее ИЛИ"

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

Код операции	Команда	Такты					Пример
		Pentium	486	386	286	86	
34 ib	XOR AL,imm8	1	1	2	3	4	xor al,0AAh
35 iw	XOR AX,imm16	1	1	2	3	4	xor ax,0F0Dh
35 id	XOR EAX,imm32	1	1	2			xor eax,23456789h
80 /6 ib	XOR r/m8,imm8	1/3	1/3	2/7	3/7	4/17+EA	xor byte ptr [di],5
81 /6 iw	XOR r/m16,imm16	1/3	1/3	2/7	3/7	4/17+EA	xor dx,0DBBh
81 /6 id	XOR r/m32,imm32	1/3	1/3	2/7			xor edx,0CAAAAh
83 /6 ib	XOR r/m16,imm8	1/3	1/3	2/7	3/7	4/17+EA	xor cx,0AAh
83 /6 ib	XOR r/m32,imm8	1/3	1/3	2/7			xor ecx,02h
30 /r	XOR r/m8,r8	1/3	1/3	2/6	2/7	3/16+EA	xor [di],ah
31 /r	XOR r/m16,r16	1/3	1/3	2/6	2/7	3/16+EA	xor bx,si
31 /r	XOR r/m32,r32	1/3	1/3	2/6			xor memory,eax
32 /r	XOR r8,r/m8	1/2	1/2	2/7	2/7	3/9+EA	xor dl,sum
33 /r	XOR r16,r/m16	1/2	1/2	2/7	2/7	3/9+EA	xor di,[si+12]
33 /r	XOR r32,r/m32	1/2	1/2	2/7			xor ecx,raznost

Операция

```
DEST = LeftSRC XOR RightSRC;
CF = 0;
OF = 0;
```

Описание

Команда XOR вычисляет "Исключающее ИЛИ" двух операндов и помещает результат на место первого операнда. Каждый бит результата равен 1, если соответствующие биты операндов различны, иначе бит результата равен 0.

Особые ситуации защищенного режима

#GP(0), если результат находится в сегменте, запрещенном для записи; #GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

6.4 Описание команд математического сопроцессора

Рис. 7-1 Общий формат команды FPU

	Команда										Опционные поля	
	Первый байт				Второй байт							
1	11011	OPA		1	mod		1	OPB		r/m	SIB	disp
2	11011	MF		OPA	mod		OPB			r/m	SIB	disp
3	11011	d	P	OPA	1	1	OPB		R	ST(i)		
4	11011	0	0	1	1	1	1	OP				
5	11011	0	1	1	1	1	1	OP				
	15-11	10	9	8	7	6	5	4	3	2 1 0		

MF - формат ячейки памяти:
- 00 - 32 битное вещественное;
- 01 - 32 битное целое;
- 10 - 64 битное вещественное;
- 11 - 16 битное целое.

P - извлечение из стека:
- 0 - не извлекать из стека;
- 1 - извлекать из стека.

d - назначение:
- 0 - ST(0);
- 1 - ST(i).

R XOR d = 0 - "Назначение" операция "Источник"
R XOR d = 1 - "Источник" операция "Назначение"
ST(i) - элемент регистрового стека:
- 000 - вершина стека;
- ...
- 111 - восьмой элемент стека.

OPA - первая часть кода операции.
OPB - вторая часть кода операции.

F2XM1 Вычислить 2^X - 1

IS	I	D	Z	O	U	P
*	*	*			*	*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 F0	F2XM1	13-57	242(140-279)	211-486	211-476	210-630	f2xm1

Операция

$ST = (2^{ST} - 1)$;

Описание

Команда F2XM1 замещает содержимое ST значением $(2^{ST} - 1)$. ST должно лежать в диапазоне $-1 < ST < 1$ для Pentium, Intel486 и Intel387 ($0 \leq ST \leq 0.5$ для Intel287 и 8087).

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Если операнд находится вне приемлемого диапазона, то результат F2XM1 неопределен.

Значения отличные от 2 могут быть потенцированы при помощи формулы $X^Y = 2^{(Y*\log_2 X)}$

Команды FLDL2T и FLDL2E загружают константы $\log_2 10$ и $\log_2 E$ соответственно. FYL2X может быть использована для вычисления $y*\log_2 X$ для произвольного положительного X.

FABS Абсолютное значение

IS I D Z O U P
*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 E1	FABS	1	3	22	10-17	10-17	fabs

Операция

Знаковый разряд ST = 0;

Описание

Команда FABS очищает знаковый бит ST. Эта операция оставляет положительное значение без изменения или замещает отрицательное значение положительным значением той же величины.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Исключение "недействительная операция" (I) возникает только при попытке извлечения из пустого стека. Никаких исключений не возникает, если операнд SNAN или представлен в неподдерживаемом формате.

FADD Сложение
FADDP Сложение с извлечением из стека FPU
FIADD Целочисленное сложение

IS	I	D	Z	O	U	P
*	*	*		*	*	*

Код	Команда	Такты					Пример
		Pentium	486	387	287	87	
D8 /0	FADD m32real	3/1	10(8-20)	24-32	90-120	90-120+EA	fadd dword ptr [si]
DC /0	FADD m64real	3/1	10(8-20)	29-37	95-125	95-125+EA	fadd qword ptr [bx]
D8 C0+i	FADD ST,ST(i)	3/1	10(8-20)	23-34	70-100	70-100	fadd ST,ST(4)
DC C0+i	FADD ST(i),ST	3/1	10(8-20)	23-34	70-100	70-100	fadd ST(2),ST
DE C0+i	FADDP ST(i),ST	3/1	10(8-20)	23-34	75-105	75-105	faddp ST(3),ST
DE C1	FADDP ST(1),ST	3/1	10(8-20)	23-34	75-105	75-105	faddp
DE /0	FIADD m16int	7/4	23(19-32)	71-85	102-137	102-137+EA	fiadd word ptr puls
DA /0	FIADD m32int	7/4	24(20-35)	57-72	108-143	108-143+EA	fiadd dword ptr memo

Операция

DEST = DEST + SRC;
IF (команда FADDP) THEN Pop(ST); FI;

Описание

Команды сложения прибавляет операнд-источник к операнду назначения и помещает сумму на место операнда назначения. (В однооперандных формах команд сложения в качестве операнда назначения выступает ST.) Команда FADDP, кроме этого, выталкивает из регистрового стека FPU верхний элемент (ST). Операнд в вершине стека может быть удвоен следующей командой: FADD ST,ST(0).

Особые ситуации защищенного режима

#GP(0),если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память при текущем уровне привилегий равном 3.

Замечание

Если операнд источник находится в памяти, то он автоматически приводится к формату временного вещественного (temp-real).

FBLD Загрузить двоично-десятичное

IS I D Z O U P
*

Код	Команда	Такты					Пример
		Pentium	486	387	287	87	
DF /4	FBLD m80dec	48-58	75(70-103)	5	290-310	290-310	fbld tbyte ptr [bx+di]

Операция

```
Декремент указателя вершины стека FPU;
ST(0) = SRC;
```

Описание

Команда FBLD преобразует BCD операнд-источник во временное вещественное (temp-real) и помещает его в стек FPU.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Источник загружается без ошибки округления. Знак источника сохраняется, включая случай, когда значение равно отрицательному нулю.

Подразумевается, что упакованные десятичные цифры находятся в диапазоне 0..9. Команда не проводит проверки на некорректные цифры (A..F), и попытка загрузить неверную кодировку приведет к неопределенному результату.

ST(7) должен быть пустым, во избежание исключения "недействительная операция" (I).

FBSTP Сохранить двоично-десятичное и извлечь из стека FPU

IS	I	D	Z	O	U	P
*	*					*

Код	Команда	Такты				Пример
		Pentium	486	387	287	87	
DF /6	FBSTP m80dec	148-154	175(172-176)	512-534	520-540	520-540+EA	fbstp tbyte ptr [bx]

Операция

```
DEST = ST(0);
Pop(ST);
```

Описание

Команда FBSTP преобразует значение в ST в упакованное десятичное число и сохраняет его в памяти по адресу указанному операндом, а затем

выталкивает ST из стека. Нецелочисленные значения сперва округляются согласно полю RC управляющего слова.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

FCHS Изменить знак

IS I D Z O U P
*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 E0	FCHS	1	6	24-25	10-17	10-17	fchs

Операция

```
Знаковый разряд ST = NOT(знаковый разряд ST);
```

Описание

Команда FCHS инвертирует знаковый бит ST.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Исключение "недействительная операция" (I) возникает только при попытке извлечения из пустого стека. Никаких исключений не возникает, если операнд SNAN или представлен в неподдерживаемом формате.

FCLEX Очистить исключения
FNCLEX

IS I D Z O U P

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
9B DB E2	FCLEX	9+at least 1 for FWAIT	7	11	2-8	2-8	fclex
DB E2	FNCLEX	9	7	11	2-8	2-8	fnclex

Операция

```
SW[0..7] = 0;
SW[15] = 0;
```

Описание

Команда FCLEX очищает флаг исключений (IE, DE, ZE, OE, UE, PE, SF), флаг состояния исключений (ES) и флаг занятости (B) в слове состояния FPU.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

FCLEX проверяет на наличие не замаскированных ошибок с плавающей запятой, перед очисткой флагов исключений. FNCLEX не проверяет.

FCOM Сравнить вещественные
FCOMP Сравнить вещественные и извлечь одно из стека FPU
FCOMPP Сравнить вещественные и извлечь оба из стека FPU

IS	I	D	Z	O	U	P
*	*	*				

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D8 /2	FCOM m32real	4/1	4	26	60-70	60-70+EA	fcom dword ptr [bp+12]
DC /2	FCOM m64real	4/1	4	31	65-75	65-75+EA	fcom qword ptr mtmor
D8 D0+i	FCOM ST(i)	4/1	4	24	40-50	40-50	fcom ST(2)
D8 D1	FCOM ST(1)	4/1	4	24	40-50	40-50	fcom
D8 /3	FCOMP m32real	4/1	4	26	63-73	63-73+EA	fcomp dword ptr [bp+12]
DC /3	FCOMP m64real	4/1	4	31	67-77	67-77+EA	fcomp dword ptr [ebp]
D8 D8+i	FCOMP ST(i)	4/1	4	26	45-52	45-52	fcomp ST(5)
D8 D9	FCOMP ST(1)	4/1	4	26	45-52	45-52	fcomp
DE D9	FCOMPP ST(1)	4/1	5	26	45-55	45-55	fcompp

Операция

```
CASE (отношение операндов) OF
  Несравнимые:  C3, C2, C0 = 111;
  ST > SRC: C3, C2, C0 = 000;
  ST < SRC: C3, C2, C0 = 001;
  ST = SRC: C3, C2, C0 = 100;
ESAC;
IF (команда FCOMP) THEN Pop(ST); FI;
IF (команда FCOMPP) THEN Pop(ST); Pop(ST); FI;
END;
```

Таблица соответствия флагов

Флаги FPU	Флаги EFLAGS
C0	CF
C1	Нет
C2	PF
C3	ZF

Описание

Команды сравнения действительных чисел сравнивают вершину стека с операндом источником, который может быть регистром или операндом в памяти (коротким или длинным вещественным). Команда FCOMP, кроме этого, выталкивает из стека верхний элемент (ST). Команда FCOMPP - два верхних элемента. Если в мнемонике не записано никакого операнда, то ST сравнивается с ST(1).

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Если операнд - NAN, представлен в неопределенном формате или произошла ошибка стека, возникает исключение "недействительная операция" (I), и биты условий устанавливаются в "не сравнимы" (C2=1).

Знак нуля игнорируется, так что -0.0 равен +0.0.

FCOS Косинус

IS	I	D	Z	O	U	P
*	*	*			*	*

Код операции	Команда	Такты			Пример
		Pentium	486	387	
D9 FF	FCOS	18-124	241(193-279)	123-772*	fcos

* Приведенное значение соответствует операндам в диапазоне |x|/4. В остальных случаях добавляется 76 тактов для преобразования операнда.

Операция

```
IF (операнды не выходят за рамки допустимого диапазона)
THEN
  C2 = 0;
  ST = cos(ST);
ELSE
  C2 = 1;
FI;
```

Описание

Команда FCOS замещает содержимое ST значением cos(ST). ST выражается в радианах и должно лежать в диапазоне $|X| < 2^{63}$.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Если операнд находится вне приемлемого диапазона, то устанавливается флаг C2, и ST остается неизмененным. На программисте лежит ответственность за уменьшение операнда до абсолютного значения, меньшего чем 2^{63}. Это можно сделать путем вычитания соответствующего целочисленного значения, умноженного на 2π.

FDECSTP Уменьшить указатель на вершину стека

IS I D Z O U P

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 F6	FDECSTP	1	3	22	6-12	6-12	fdecstp

Операция

```
IF TOP = 0
THEN TOP = 7;
ELSE TOP = TOP - 1;
FI;
```

Описание

Команда FDECSTP вычитает единицу (без переноса) из трехбитного поля TOP слова состояния FPU.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Эффект действия команды FDECSTP заключается во вращении стека. Она не изменяет ни регистров тегов, ни перемещает данные.

FDISI Запрет прерываний (8087)

IS I D Z O U P

Код операции	Команда	Такты 87	Пример
DB E1	FDISI	5(2-8)	fdisi

Описание

Команда FDISI устанавливает маску IEM = 1 в регистре состояния (SR), запрещая 8087 генерировать запросы на прерывание на выходе INT.

Замечание

Процессоры Pentium, Intel486 и математические сопроцессоры Intel287, Intel387, неподдерживающие команду FDISI, воспринимают ее аналогично FNOP, исключения "недействительная операция" не возникает. Проверка на незамаскированные исключения не осуществляется.

FDIV Деление
FDIVP Деление с извлечением из стека FPU
FIDIV Целочисленное деление

IS I D Z O U P
* * * * * * *

Код	Команда	Такты Pentium	486	387	287	87	Пример
D8 /6	FDIV m32real	39	73	89	215-225	215-225	fdiv dword ptr [si]
DC /6	FDIV m64real	39	73	94	220-230	220-230	fdiv qword ptr [bx]
D8 F0+i	FDIV ST,ST(i)	39	73	88-91	193-203	193-203	fdiv ST,ST(4)
DC F8+i	FDIV ST(i),ST	39	73	88-91	193-203	193-203	fdiv ST(2),ST
DE F8+i	FDIVP ST(i),ST	39	73	88-91	198-209	197-207	fdivp ST(3),ST
DE F9	FDIVP ST(1),ST	39	73	88-91	198-209	197-207	fdivp
DE /6	FIDIV m16int	42	73	136-140	224-238	224-238+EA	fidiv word ptr memo
DA /6	FIDIV m32int	42	73	120-127	230-243	230-243+EA	fidiv dword ptr [si]

Операция

DEST = DEST ÷ SRC;
IF (команда FDIVP) THEN Pop(ST); FI;

Описание

Команды деления делят операнд-назначение на операнд-источник (в однооперандных формах команд деления в качестве операнда-назначения используется ST) и возвращают частное на место операнда-назначения. Команда FDIVP, кроме этого, выталкивает из регистрового стека FPU верхний элемент (ST). Если в мнемонике команды FDIVP не записано никакого операнда, то ST(1) делится на ST.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Если операнд-источник находится в памяти, то он автоматически приводится к временному вещественному формату. Действие команды деления зависит от значения поля PC (управление точностью) управляющего слова FPU. Если PC определяет точность 53 бита, то команды деления выполняются за 33 такта (Pentium). Если установлена точность 24 бита, то команды деления выполняются только за 19 тактов (Pentium).

На процессорах, поддерживающих стандарт IEEE-754 (Pentium,Intel486, Intel387) операция над денормализованными операндами может привести к исключению "антипереполнение". Операция над денормализованными операндами приводит к исключению "недействительная операция" на математических сопроцессорах Intel287 и 8087, не

отвечающих стандарту IEEE-754, "антипереполнение" в этом случае невозможно.

Изменение обработчика исключения "антипереполнение" может потребоваться, только если необходимо по-разному обрабатывать различные коды операций. Возможно возникновение нескольких исключений "недействительная операция".

FDIVR Инверсное деление
FDIVRP Инверсное деление с извлечением из стека FPU
FIDIVR Инверсное целочисленное деление

IS	I	D	Z	O	U	P
*	*	*	*	*	*	*

Код	Команда	Такты					Пример
		Pentium	486	387	287	87	
D8 /7	FDIVR m32real	39	73	89	215-225	215-225	fdivr dword ptr [si]
DC /7	FDIVR m64real	39	73	94	220-230	220-230	fdivr qword ptr [bx]
D8 F8+i	FDIVR ST,ST(i)	39	73	88-91	198-208	198-208	fdivr ST,ST(4)
DC F0+i	FDIVR ST(i),ST	39	73	88-91	198-208	198-208	fdivr ST(2),ST
DE F0+i	FDIVRP ST(i),ST	39	73	88-91	198-208	198-208	fdivrp ST(3),ST
DE F1	FDIVRP ST(1),ST	39	73	88-91	198-208	198-208	fdivrp ST(1),ST
DE /7	FIDIVR m16int	42	73	135-141	224-238	224-238+EA	fidivr word ptr [si]
DA /7	FIDIVR m32int	42	73	121-128	230-243	230-243+EA	fidivr dword ptr [bp]

Операция

```
DEST = SRC ÷ DEST;
IF (команда FDIVRP) THEN Pop(ST); FI;
```

Описание

Команды инверсного деления делят операнд-источник на операнд-назначение (в однооперандных формах команд деления в качестве операнда-назначения используется ST) и возвращают частное на место операнда-назначения. Команда FDIVRP, кроме этого, выталкивает из регистрового стека FPU верхний элемент (ST). Если в мнемонике команды FDIVRP не записано никакого операнда, то ST делится на ST(1).

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сег-

менте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Если операнд-источник находится в памяти, то он автоматически приводится к временному вещественному формату. Действие команды деления зависит от значения поля PC (управление точностью) управляющего слова FPU. Если PC определяет точность 53 бита, то команды деления выполняются за 33 такта (Pentium). Если установлена точность 24 бита, то команды деления выполняются только за 19 тактов (Pentium).

На процессорах, поддерживающих стандарт IEEE-754 (Pentium,Intel486, Intel387) операция над денормализованными операндами может привести к исключению "антипереполнение". Операция над денормализованными операндами приводит к исключению "недействительная операция" на математических сопроцессорах Intel287 и 8087, не отвечающих стандарту IEEE-754, "антипереполнение" в этом случае невозможно.

Изменение обработчика исключения "антипереполнение" может потребоваться, только если необходимо по-разному обрабатывать различные коды операций. Возможно возникновение нескольких исключений "недействительная операция".

FENI Разрешение прерываний (8087)

IS I D Z O U P

Код операции	Команда	Такты 87	Пример
DB E0	FENI	5(2-8)	feni

Описание

Команда FENI устанавливает в нуль маску IEM = 0 в регистре состояния (SR), разрешая 8087 генерировать запросы на выходе INT.

Замечание

Процессоры Pentium, Intel486 и математические сопроцессоры Intel287, Intel387, неподдерживающие команду FENI, воспринимают ее аналогично FNOP, исключения "недействительная операция" не возникает. Проверка на незамаскированные исключения не осуществляется.

FFREE Освободить регистр с плавающей запятой

IS I D Z O U P

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
DD C0+i	FFREE ST(i)	1	3	18	9-16	9-16	ffree ST(1)

Операция

TAG(i) = 11b;

Описание

Команда FFREE помечает регистр назначение как пустой (Соответствующие поля в регистре тегов принимают значения 11b).

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

FFREE не влияет на содержимое регистра назначения. Указатель на вершину стека с плавающей запятой (поле TOP в регистре состояния) также остается неизменным.

FICOM Сравнить целое
FICOMP Сравнить целое и извлечь из стека FPU

IS	I	D	Z	O	U	P
*	*	*				

Код	Команда	Такты					Пример
		Pentium	486	387	287	87	
DE /2	FICOM m16int	8/4	18(16-20)	71-75	72-86	72-86+EA	ficom word ptr [eax]
DA /2	FICOM m32int	8/4	17(15-17)	56-63	78-91	78-91+EA	ficom dword ptr [si]
DE /3	FICOMP m16int	8/4	18(16-20)	71-75	74-88	74-88+EA	ficomp tbas
DA /3	FICOMP m32int	8/4	17(15-17)	56-63	80-93	80-93+EA	ficomp dword ptr [bp]

Операция

```
CASE (отношение операндов) OF
  Не сравнимые:  C3, C2, C0 = 111;
  ST > SRC: C3, C2, C0 = 000;
  ST < SRC: C3, C2, C0 = 001;
  ST = SRC: C3, C2, C0 = 100;
ESAC;
IF (команда FICOMP) THEN Pop(ST); FI;
```

Таблица соответствия флагов

Флаги FPU	Флаги EFLAGS
C0	CF
C1	Нет
C2	PF
C3	ZF

Описание

Команды целочисленного сравнения сравнивают содержимое вершины стека с операндом источником. Команда FICOMP, кроме этого, выталкивает из стека верхний элемент (ST).

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Операнд в памяти сперва преобразуется в формат временного вещественного, прежде чем произведется сравнение.

Если операнд - NAN, представлен в неопределенном формате или произошла ошибка стека, возникает исключение "недействительная операция" (I), и биты условий устанавливаются в "не сравнимо" (C2 = 1).

FIDIV Целочисленное деление

Смотрите FDIV.

FIDIVR Инверсное целочисленное деление

Смотрите FDIVR.

FILD Загрузить целое

IS I D Z O U P
*

Код	Команда	Такты					Пример
		Pentium	486	387	287	87	
DF /0	FILD m16int	3/1	12(9-12)	61-65	46-54	46-54+EA	fild word ptr [bx].sec
DB /0	FILD m32int	3/1	15(13-16)	45-52	52-60	52-60+EA	fild dword ptr [di+bx]
DF /5	FILD m64int	3/1	17(10-18)	56-67	60-68	60-68+EA	fild qword ptr [bx+12]

Операция

Декремент указателя вершины стека FPU;
ST(0) = SRC;

Описание

Команда FILD преобразует знаковый целочисленный операнд-источник в формат временного вещественного и заталкивает его в стек FPU.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Источник загружается без ошибки округления. ST(7) должен быть пуст, во избежание исключения "недействительная операция" (I).

FIMUL Целочисленное умножение

Смотрите FMUL.

FINCSTP Увеличить указатель вершины стека

IS I D Z O U P

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 F7	FINCSTP	1	3	21	6-12	6-12	fincstp

Операция

```
IF TOP = 7
THEN TOP = 0;
ELSE TOP = TOP + 1;
FI;
```

Описание

Команда FINCSTP добавляет единицу (без переноса) к трехбитному полю TOP слова состояния FPU.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлен. в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Эффект действия команды FINCSTP заключается во вращении стека. Она не изменяет ни регистров тегов, ни перемещает данные. Она не эквивалентна операции выталкивания из стека, потому что, она не устанавливает признак (TEG) старой вершины стека в значение пустой.

FINIT Инициализация FPU
FNINIT

IS I D Z O U P

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
9B DB E3	FINIT	16	17	33	2-8	2-8	finit
DB E3	FNINIT	12	17	33	2-8	2-8	fninit

Операция

```
CW = 037Fh;                    (* Управляющее слово *) (* Control word *)
SW = 0;                        (* Слово состояние *) (* Status word *)
TW = FFFFh;                    (* Слово признака *) (* Tag word *)
FEA = 0; FDS = 0;              (* Указатель данных *) (* Data pointer *)
FIP = 0; FOP = 0; FCS = 0;     (* Указатель команды *) (* Instruction pointer *)
```

Описание

Команды инициализации устанавливают FPU в определенное состояние, независящее от предыдущей деятельности.

Слово управления FPU устанавливается в 037Fh (округлять к ближайшему, все особые ситуации замаскированы, точность 64 бита). Слово состояния очищается (флагов особых ситуаций неустановлено, регистр стека R0 является вершиной стека). Все стековые регистры помечаются как пустые. Указатели ошибки (команды и данных) очищены.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

FINIT проверяет на условие незамаскированной ошибки с плавающей запятой, а FNINIT не проверяет. В процессорах Pentium и Intel486, в отличие от Intel387, Intel287 и 8087, FINIT и FNINIT очищают указатели ошибок.

FIST Сохранить целое
FISTP Сохранить целое и извлечь из стека FPU

IS	I	D	Z	O	U	P
*	*					*

Код	Команда	Такты					Пример
		Pentium	486	387	287	87	
DF /2	FIST m16int	6	33(29-34)	82-95	80-90	80-90+EA	fist word ptr [si+2]
DB /2	FIST m32int	6	32(28-34)	79-93	82-92	82-92+EA	fist dword ptr [bx]
DF /3	FISTP m16int	6	33(29-34)	82-95	82-92	82-92+EA	fistp word ptr [dx]
DB /3	FISTP m32int	6	33(29-34)	79-93	84-94	84-94+EA	fistp dword ptr [edi]
DF /7	FISTP m64int	6	33(29-34)	80-97	94-105	94-105+EA	fistp qword ptr [di]

Операция

```
DEST = ST(0);
IF (команда FISTP) THEN Pop(ST); FI;
```

Описание

Команда FIST преобразует значение в ST в знаковое целое в соответствии с полем RC управляющего слова и пересылает результат в память. ST остается неизменным. FIST может использовать слово или короткое целое в качестве назначения. FISTP также может использовать длинное целое. Команда FISTP, кроме этого, выталкивает из регистрового стека FPU верхний элемент (ST).

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки) страничная ошибка; #AC при невыровненной ссылке в память при текущем уровне привилегий, равном 3.

Замечание

Отрицательный нуль сохраняется с такой же кодировкой (00..00), как и положительный нуль. Если значение слишком велико, чтобы быть представленным как целое, то возникает исключение "недействительная операция". Маскированный отклик (когда в регистре FPU, бит IM = 1) - запись наибольшего отрицательного целого в память.

FISUB Целочисленное вычитание

Смотрите FSUB.

FISUBR Инверсное целочисленное вычитание

Смотрите FSUBR.

FLD Загрузить вещественное

IS	I	D	Z	O	U	P
*	*	*				

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 /0	FLD m32real	1	3	20	38-56	38-56+EA	fld dword ptr [bx]
DD /0	FLD m64real	1	3	25	40-60	40-60+EA	fld qword ptr [bp]
DB /5	FLD m80real	3	6	44	53-65	53-65+EA	fld tbyte ptr [si+12]
D9 C0+i	FLD ST(i)	1	4	14	17-22	17-22	fld ST(3)

Операция

Декремент указателя вершины стека FPU;
ST(0) = SRC;

Описание

Команда FLD заталкивает операнд-источник в стек FPU. Если источник это регистр, то номер регистра используется прежде, чем происходит уменьшение указателя вершины стека. В частности, код FLD ST(0) дублирует вершину стека.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM,

либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Если операнд-источник представлен коротким или длинным вещественным, то он автоматически преобразуется к формату временного вещественного. Загрузка временного вещественного операнда не требует преобразований, так что в этом случае исключения "недействительная операция" (I) и "денормализованный операнд" (D) не произойдут.

ST(7) должен быть пуст, во избежание исключения "недействительная операция" (I).

Команда FLD m80real на процессорах Pentium, Intel486 и Intel387 не может вызывать исключение "денормализованный операнд", потому что это не арифметическая команда. На сопроцессорах Intel287 и 8087 исключение "денормализованный операнд" возможно. Это не влияет на совместимость программного обеспечения и введено для расширения возможностей.

В командах FLD m32real и FLD m64real для процессоров Pentium, Intel486 и Intel387 загружаемое денормализованное число будет конвертировано к точному временному вещественному (так как оно помещается в стек). Для сопроцессоров Intel287 и 8087 загружаемое денормализованное число будет конвертировано к ненормализованному. Если в программе далее следует команда FXTRACT или FXAM, процессоры Pentium, Intel486 и Intel387 дадут результат отличный от сопроцессоров Intel287 и 8087. Данное изменение было введено для совместимости со стандартом IEEE-754.

В командах FLD m32real и FLD m64real, когда загружается SNAN, процессоры Pentium, Intel486 и Intel387 устанавливают исключение

"недействительная операция", а сопроцессоры Intel287 и 8087 - нет. Обработчик исключения необходимо скорректировать для обработки этого условия. Изменение было внесено для совместимости со стандартом IEEE-754.

FLD1 Загрузить константу +1.0
FLDL2T Загрузить константу $\log_2 10$
FLDL2E Загрузить константу $\log_2 e$
FLDPI Загрузить константу π
FLDLG2 Загрузить константу $\log_{10} 2$
FLDLN2 Загрузить константу $\log_e 2$
FLDZ Загрузить константу +0.0

IS I D Z O U P
*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 E8	FLD1	2/2	4	24	15-21	15-21	fld1
D9 E9	FLDL2T	5/3	8	40	16-22	16-22	fldе2t
D9 EA	FLDL2E	5/3	8	40	15-21	15-21	fldl2e
D9 EB	FLDPI	5/3	8	40	16-22	16-22	fldpi
D9 EC	FLDLG2	5/3	8	41	18-24	18-24	fldlg2
D9 ED	FLDLN2	5/3	8	41	17-23	17-21	fldln2
D9 EE	FLDZ	2/2	4	20	11-17	11-17	fldz

Операция

```
Декремент указателя вершины стека FPU;
ST(0) = CONSTANT;
```

Описание

Каждая из команд помещает в стек FPU часто используемые константы (во временном вещественном формате).

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

ST(7) должен быть пуст, во избежание исключения "недействительная операция" (I).

Используется внутренняя 66 битная константа, округляемая к временному вещественному формату (как указано в бите RC управляющего слова). Исключение "неточный результат" (P) не возникает.

Управление округлением реализовано для процессоров Pentium, Intel486 и Intel387. Управление округлением не выполняется для сопроцессоров Intel287 и 8087. Результат для FLDPI, FLDLN2, FLDG2 и FLDL2E для Pentium, Intel486, Intel387 такой же как и на Intel287 и 8087 в тех случаях, когда установлено округление к ближайшему или к $+\infty$. Результат также одинаковый для FLD2T, когда установлено округление к ближайшему, к $-\infty$ или к нулю. Результаты различны в наименьшем значащем бите мантиссы если для FLDPI, FLDLN2, FLDG2 и FLDL2E установлено округление к $-\infty$ или к нулю, и для FLDL2T если установлено округление к $+\infty$. Изменения внесены в соответствии с требованием стандарта IEEE-754.

FLDCW Загрузить управляющее слово

IS	I	D	Z	O	U	P
*	*	*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 /5	FLDCW m2byte	7	4	19	7-19	7-19+EA	fldcw con_word

Операция

CW = SRC;

Описание

Команда FLDCW замещает текущее значение управляющего слова FPU значением, содержащимся в указанном слове памяти.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

FLDCW обычно используется для установки или изменения режима работы FPU. Если бит исключения в слове состояния установлен, то загрузка нового управляющего слова, которое размаскирует эту особую ситуацию, приведет в результате к состоянию ошибки с плавающей запятой. При изменении режимов, рекомендуется очистить все ожидающие решения исключения, перед загрузкой нового управляющего слова.

FLDENV Загрузить среду FPU

IS	I	D	Z	O	U	P
*	*	*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 /4	FLDENV m14/ 28byte	37, 16 bit pm=32, 32 bit pm=33	44, pm=34	71	35-45	35-45+EA	fldenv [bp+6]

Операция

Задать среду FPU по значению в SRC;

Описание

Команда FLDENV перегружает среду FPU из области памяти, определенной операндом-источником. Данные должны быть записаны в память предшествующими командами, например, FSTENV или FNSTENV.

Среда FPU состоит из управляющего слова FPU, слова состояния, слова признаков (тегов) и указателей ошибки (как данных, так и команды). Расположение этих значений в памяти зависит, как от размера операнда, так и от текущего режима работы процессора и приведено на рис. 4-5..4-8 (стр. 68..70). Атрибут USE (размер операнда и размер адреса по умолчанию) текущего кодового сегмента определяет размер операнда: 14 байтный операнд применяется для сегмента USE16, и 28 байтный - для сегмента USE32. FLDENV должна выполняться в том же режиме работы, что и соответствующая ей FSTENV или FNSTENV.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Если образ среды содержит незамаскированные исключения, то загрузка его даст в результате состояние ошибки с плавающей запятой.

FMUL Умножение
FMULP Умножение с извлечением из стека FPU
FIMUL Целочисленное умножение

IS	I	D	Z	O	U	P
*	*	*		*	*	*

Код	Команда	Такты				Пример
		Pentium	486	387	287	87	
D8 /1	FMUL m32real	3/1	11	27-35	110-125	110-125+EA	fmul dword ptr [si]
DC /1	FMUL m64real	3/1	14	32-57	112-168	154-168+EA	fmul qword ptr [bx+mem]
D8 C0+i	FMUL ST,ST(i)	3/1	16	29-57	90-145	130-145	fmul ST,ST(4)
DC C8+i	FMUL ST(i),ST	3/1	16	29-57	90-145	130-203	fmul ST(2),ST
DE C8+i	FMULP ST(i),ST	3/1	16	29-57	198-208	134-148	fmulp ST(3),ST
DE C9	FMULP ST(1),ST	3/1	16	29-57	198-208	134-148	fmulp
DE /1	FIMUL m16int	7/4	8	76-87	124-138	124-138+EA	fimul word ptr [eax*2]
DA /1	FIMUL m32int	7/4	8	61-82	130-144	130-144+EA	fimul dword ptr [si+bx]

Операция

```
DEST = DEST * SRC;
IF (команда FMULP) THEN Pop(ST); FI;
```

Описание

Команды умножения умножают операнд-назначение на операнд-источник и помещают результат на место операнда назначения. (Однооперандные формы команд умножения используют в качестве операнда-назначения ST.) Команда FMULP, кроме этого, выталкивает из регистрового стека FPU верхний элемент (ST). Если в мнемонике команды FMULP не записано никакого операнда, то ST(1) умножается на ST.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Если операнд-источник находится в памяти, то он автоматически приводится к формату временного вещественного.

FNOP Нет операции

IS I D Z O U P

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 D0	FNOP	1	3	12	10-16	10-16	fnop

Описание

Команда FNOP не осуществляет никаких действий. Она не влияет ни на что, за исключением указателя команд.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

FPATAN Частичный арктангенс

IS	I	D	Z	O	U	P
*	*	*			*	*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 F3	FPATAN	13-173	5(2-17)	314-487	250-800	250-800	fpatan

Операция

```
ST(1) = arctan(ST(1) ÷ ST);
Pop(ST);
```

Описание

Команда FPATAN вычисляет арктангенс выражения ST(1) ÷ ST и возвращает вычисленное значение выраженное в радианах в ST(1). Результат имеет тот же знак, что и операнд из ST(1), и величину меньше π.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Тот факт, что FPATAN принимает два аргумента и возвращает арктангенс их отношения, упрощает вычисление других тригонометрических функций. Например, arcsin(x) (который является арктангенсом от $X \div \sqrt{(1-X^2)}$ может быть вычислен, при использовании следующей последовательности действий: поместить x в стек FPU; вычислить $\sqrt{(1-X^2)}$ и поместить значение результата в стек; выполнить FPATAN.

Для процессоров Pentium, Intel486 и Intel387 никаких ограничений на диапазон принимаемых FPATAN аргументов не налагается. Для сопроцессоров Intel287 и 8087 существует ограничение |ST(0)| < |ST(1)|.

Это не влияет на совместимость программного обеспечения и введено для расширения возможностей.

FPREM Нахождение частичного остатка деления

IS	I	D	Z	O	U	P
*	*	*			*	

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 F8	FPREM	16–64	2(2–8)	74–155	15–190	15–190	fprem

Операция

```
EXPDIF = exponent(ST) - exponent(ST(1));
IF EXPDIF < 64
THEN
  Q = целое, получаемое округлением ST ÷ ST(1) в сторону нуля;
  ST = ST - (ST(1) * Q);
  C2 = 0;
  C0, C1, C3 = три младших бита Q; (* Q2, Q1, Q0 *)
ELSE
  C2 = 1;
  N = число между 32 и 63;
  QQ = целое, получаемое округлением (ST ÷ ST(1)) ÷ 2^(EXPDIF-N) в сторону нуля;
  ST = ST - (ST(1) * QQ * 2^(EXPDIF-N);
FI;
```

Описание

Команда FPREM вычисляет остаток, получаемый от деления ST на ST(1), и помещает результат в ST. Знак остатка такой же, как и знак исходного делимого в ST. Величина остатка меньше чем величина модуля.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

FPREM получает точный результат; не происходит исключения "неточный результат" (P), управление округлением не влияет на результат.

Команда FPREM не является операцией остатка по стандарту IEEE-754. Необходимо использовать команду FPREM1, для получения такого рода остатка. FPREM поддерживается для совместимости с математическими сопроцессорами 8087, 80287.

FPREM работает методом итерационного вычитания и может уменьшить показатель степени ST не более чем на 63 за одно свое выполнение. Если в процессе выполнения FPREM получается остаток меньший чем модуль, то функция считается выполненной и флаг C2 очищается. В противном случае, флаг C2 устанавливается, а результат в ST называется неполным остатком. Показатель степени неполного остатка меньше, чем показатель степени исходного делимого, по крайней мере на 32. Программа может перезапускать команду (используя неполный остаток в ST как делимое), пока флаг C2 не будет очищен. Процедура прерывания с более высоким приоритетом, которое использует FPU, может изменить ход выполнения, переключаясь между командами и циклом вычисления остатка.

На процессорах, поддерживающих стандарт IEEE-754 (Pentium,Intel486, Intel387) операция над денормализованными операндами может привести к исключению "антипереполнение". Операция над денормализованными операндами приводит к исключению "недействительная операция" на математических сопроцессорах Intel287 и 8087, не отвечающих стандарту IEEE-754, "антипереполнение" в этом случае невозможно.

Изменение обработчика исключения "антипереполнение" может потребоваться, только если необходимо по-разному обрабатывать различные коды операций. Возможно возникновение нескольких исключений "недействительная операция".

Важное использование FPREM - это уменьшение аргументов периодических функций. Когда уменьшение выполнено, FPREM отражает три наименее значащих бита частного во флагах C3, C1, C0.

Биты C3, C1, C0 в слове состояния в процессорах Pentium, Intel486, Intel387 корректно отражают три младших бита частного. В сопроцессорах

Intel287 и 8087 биты частного некорректны, когда выполняется понижение 64^N+M, где N≥1 и M=1 или M=2.

FPREM1 Нахождение частичного остатка деления по стандарту IEEE-754

IS	I	D	Z	O	U	P
*	*	*			*	

Код операции	Команда	Такты			Пример
		Pentium	486	387	
D9 F5	FPREM1	70–20	94(72–167)	95–185	fprem1

Операция

```
EXPDIF = exponent(ST) - exponent(ST(1));
IF EXPDIF < 64
THEN
  QQ = целое, получаемое округлением ST ÷ ST(1);
    (* к ближайшему целому или к ближайшему четному целому, если результат
       точно по середине между двумя числами *)
  ST = ST - (ST(1) * Q);
  C2 = 0;
  C0, C1, C3 = три младших бита Q; (* Q2, Q1, Q0 *)
ELSE
  C2 = 1;
  N = число между 32 и 63;
  QQ = целое, получаемое округлением (ST ÷ ST(1)) ÷ 2^(EXPDIF-N) в сторону нуля;
  ST = ST - (ST(1) * QQ * 2^(EXPDIF-N);
FI;
```

Описание

Команда FPREM1 вычисляет остаток, получаемый при делении ST на ST(1), и оставляет результат в ST. Величина остатка меньше, чем половина величины модуля.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

FPREM1 получает точный результат; не происходит исключения "неточный результат" (P), управление округлением не влияет на результат.

Команда FPREM1 является операцией остатка по стандарту IEEE-754. Она отличается от FPREM методом округления частного (смотрите пункт "Операция"), когда разница показателей степени exp(ST) - exp(ST(1)) меньше чем 64.

FPREM1 работает методом итерационного вычитания и может уменьшить показатель степени ST не более чем на 63 за одно свое выполнение. Если в процессе выполнения FPREM1 получается остаток меньший чем модуль, то функция считается выполненной, и флаг C2 очищается. В противном случае, флаг C2 устанавливается, а результат в ST называется неполным остатком. Показатель степени неполного остатка меньше, чем показатель степени исходного делимого, по крайней мере на 32. Программа может перезапускать команду (используя неполный остаток в ST как делимое), пока флаг C2 не будет очищен. Процедура прерывания с более высоким приоритетом, которое использует FPU, может изменить ход выполнения, переключаясь между командами и циклом вычисления остатка.

Важное использование FPREM1 - это уменьшение аргументов периодических функций. Когда уменьшение выполнено, FPREM1 отражает три наименее значащих бита частного во флагах C3, C1, C0.

FPTAN Частичный тангенс

IS	I	D	Z	O	U	P
*	*	*			*	*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 F2	FPTAN	17-173	244(200-273)	191-573	30-540	30-540	fptan

Операция

```
IF (операнд в диапазоне)
THEN
  C2 = 0;
```

```
  ST = tan(ST);
Декремент указателя вершины стека FPU;
  ST = 1.0;
ELSE
  C2 = 1;
FI;
```

Описание

Команда FPTAN замещает содержимое ST значением tan(ST) и затем помещает 1.0 в стек FPU. ST выражено в радианах и должно лежать в диапазоне $|X| < 2^{63}$.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Если значение операнда находится вне приемлемого диапазона, то устанавливается флаг C2, а ST остается неизменным. На ответственности программиста лежит уменьшение операнда до абсолютного значения меньшего 2^{63}, путем вычитания подходящего целого, умноженного на 2π.

Тот факт, что FPTAN помещает 1.0 в стек FPU после вычисления tan(ST), поддерживает совместимость с математическими сопроцессорами 8087, Intel287 и упрощает вычисление других тригонометрических функций. Например, котангенс (который является обратной величиной тангенса) может быть вычислен при помощи выполнения FDIVR после FPTAN.

ST(7) должен быть пустым, во избежание исключения "недействительная операция" (I).

Для сопроцессоров Intel286 и 8087 диапазон операнда ограничен $|ST(0)| < \pi/4$, поэтому операнд предварительно должен быть уменьшен командой FPREM.

В случае переполнения стека, когда исключение "недействительная операция" замаскировано, процессоры Pentium, Intel486 и Intel387 помещают в ST и ST(1) QNAN (это обеспечивает совместимость со стандартом

IEEE-754), сопроцессоры Intel287 и 8087 помещают QNAN только в ST, а в ST(1) возвращается исходный операнд (бывший ранее в ST).

FRNDINT Округлить к целому

IS	I	D	Z	O	U	P
*	*	*				*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 FC	FRNDINT	9-20	29(21-30)	66-80	16-50	16-50	frndint

Операция

ST = округленное ST;

Описание

Команда FRNDINT округляет значение в ST к целому, в зависимости от поля RC в управляющем слове FPU.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

FRSTOR Восстановить состояние FPU

IS	I	D	Z	O	U	P
*	*	*	*	*	*	*

Код операции	Команда	Такты				Пример
		Pentium	486	387	287 87	
DD /4	FRSTOR m94/	16 bit rm or vm=75,	131,	308	205-215 197-207+EA	frstor mp

```
                108byte      32 bit rm or vm=95, pm=120
                                 pm=70
Замечание: Количество тактов в 80287 не является определяющим для вычисления полного
времени выполнения данной команды. Для обычного соотношения частот 80286 и 80287
выполнение команды в 80287 осуществляется параллельно с поступлением операндов. Полное
время выполнения команды определяется временем поступления (передачи) операндов. Для
соотношения частот 80286:80287 4:8, 1:1 и 8:5, полное количество тактов соответственно
составляет 490, 302 и 227 для 80287.
```

Операция

```
Состояние FPU = SRC;
```

Описание

Команда FRSTOR перезагружает состояние FPU (среду и регистровый стек) из области памяти, указанной операндом источником. Эти данные могут быть записаны предшествующими командами, например, FSAVE или FNSAVE.

Среда FPU состоит из управляющего слова FPU, слова состояния, слова признаков (тегов) и указателей ошибки (как данных, так и команды). Расположение окружения в памяти зависит, как от размера операнда, так и от текущего режима работы процессора и приведено на рис. 4-5..4-8 (стр. 68..70). Атрибут USE (размер операнда и размер адреса по умолчанию) текущего кодового сегмента определяет размер операнда: 94 байтный операнд применяется для сегмента USE16, и 108 байтный - для сегмента USE32. Регистры стека, начиная с ST и заканчивая ST(7), занимают 80 байт и следуют сразу же за образом среды. FRSTOR должна выполняться в том же режиме, что и соответствующая ей команда FSAVE или FNSAVE.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Если образ среды содержит незамаскированные исключения, то загрузка его даст в результате состояние ошибки с плавающей запятой.

FSAVE Сохранить состояние FPU
FNSAVE

IS I D Z O U P

Код операции	Команда	Такты Pentium	486	387	287	87	Пример
9B DD /6	FSAVE m94 /108byte	16 bit rm or vm=127, 32 bit rm or vm=151, pm=124; + at least 3 for FWAIT		375-376	205-215	197-207+EA	fsave mn1
DD /6	FNSAVE m94 /108byte	16 bit rm or vm=127, 32 bit rm or vm=151, pm=124		375-376	205-215	197-207+EA	fnsave mX

Замечание: Количество тактов в 80287 не является определяющим для вычисления полного времени выполнения данной команды. Для обычного соотношения частот 80286 и 80287 выполнение команды в 80287 осуществляется параллельно с поступлением операндов. Полное время выполнения команды определяется временем поступления(передачи) операндов. Для соотношения частот 80286:80287 4:8, 1:1 и 8:5, полное количество тактов соответственно составляет 490, 302 и 227 для 80287.

Операция

```
DEST = состояние FPU;
Инициализировать FPU; (* Эквивалентна FNINIT *)
```

Описание

Команда FSAVE записывает текущее состояние FPU (среду и регистровый стек) по указанному адресу, а затем инициализирует FPU. Среда FPU состоит из управляющего слова FPU, слова состояния, слова признаков (тегов) и указателей ошибки (как данных, так и команды).

Компоновка состояния в памяти зависит от размера операнда и текущего режима работы процессора. Атрибут USE (размер операнда и размер адреса по умолчанию) текущего кодового сегмента определяет размер опе-

ранда: 94 байтный операнд применяется для сегмента USE16, и 108 байтный - для сегмента USE32. Формат представления в памяти среды FPU приведен на рис. 4-5..4-8 (стр. 68..70). Регистры стека, начиная с ST и заканчивая ST(7), занимают 80 байт и следуют сразу же за образом среды.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

FSAVE и FNSAVE не сохраняют состояние FPU, пока вся деятельность FPU не завершится. Таким образом, сохраненный образ отражает состояние FPU после любой предварительно декодированной и выполненной команды. Если программа читает из памяти образ состояния сразу после сохранения, то она должна выполнить команду FWAIT, чтобы завершить сохранение.

Команды сохранения обычно используются, когда операционной системе необходимо осуществить контекстное переключение, или когда обработчику особых ситуаций необходимо использовать FPU, или когда прикладная программа хочет передать "чистый" FPU в подпрограмму.

FSCALE Масштабировать

IS	I	D	Z	O	U	P
*	*	*		*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 FD	FSCALE	20-31	31(30-32)	67-86	32-38	32-38	fscale

Операция

$$ST = ST * 2^{ST(1)};$$

Описание

Команда FSCALE интерпретирует величину в ST(1) как целое и добавляет это целое к показателю степени ST. Если значение не целочисленное, то FSCALE использует ближайшее целое меньшее по величине, т.е. обрезает значение в сторону нуля. Если получаемое целое равно нулю, то значение в ST не изменяется. Таким образом, FSCALE обеспечивает быстрое умножение или деление на целые степени двойки.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

FSCALE может быть использована как инверсия команды FXTRACT. В этом случае, так как FSCALE не выталкивает из стека показательную часть, то после FSCALE надо поставить FSTP ST(1).

На процессорах Pentium, Intel486, Intel387 диапазон масштабируемого операнда не ограничен. Если $0<|ST(1)|<1$, масштабным множителем будет нуль, поэтому ST(0) остается неизменным. Если округленный результат неточен, или если произошла потеря точности ("антипереполнение" замас-

кировано), устанавливается исключение "неточный результат". Диапазон масштабируемого операнда на Intel287 и 8087 ограничен. Если 0<|ST(1)|<1, результат неопределен и исключения не устанавливаются. Влияние на программное обеспечение заключается в том, что различные результаты получаются, когда 0<|ST(1)|<1. Это изменение было предпринято для расширения диапазона масштабирования.

FSETPM Установить защищенный режим работы (Intel287)

IS	I	D	Z	O	U	P

Код операции	Команда	Такты 287	Пример
DB E4	FSETPM	2-8	fsetpm

Описание

Команда FSETPM переводит сопроцессор в защищенный режим работы, возврат в реальный режим возможен только при сбросе сопроцессора.

Замечание

Процессоры Pentium, Intel486 и математические сопроцессоры Intel287, Intel387, неподдерживающие команду FSETPM, воспринимают ее аналогично FNOP, исключения "недействительная операция" не возникает. Проверка на незамаскированные исключения не осуществляется.

FSIN Синус

IS	I	D	Z	O	U	P
*	*	*			*	*

Код операции	Команда	Такты Pentium	486	387	Пример
D9 FE	FSIN	16-126	241(193-279)	122-771*	fsin

* Приведенное значение соответствует операндам в диапазоне |X|/4. В остальных случаях добавляется 76 тактов для преобразования операнда.

Операция

```
IF (значение операнда лежит в пределах допустимого диапазона)
THEN
  C2 = 0;
  ST = sin(ST);
ELSE
  C2 = 1
FI;
```

Описание

Команда FSIN заменяет содержимое ST значением sin(ST). ST выражается в радианах и должно лежать в диапазоне $|x| < 2^{63}$.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Если операнд находится вне приемлемого диапазона, то устанавливается флаг C2, а ST остается неизменным. На ответственности программиста лежит уменьшение операнда до абсолютного значения меньшего 2^{63}, путем вычитания подходящего целого, умноженного на 2π.

FSINCOS Синус и косинус

IS	I	D	Z	O	U	P
*	*	*			*	*

Код операции	Команда	Такты			Пример
		Pentium	486	387	
D9 FB	FSINCOS	17-137	291(243-329)	191-809*	fsincos

* Приведенное значение соответствует операндам в диапазоне $|X|/4$. В остальных случаях добавляется 76 тактов для преобразования операнда.

Операция

```
IF (значение операнда лежит в пределах допустимого диапазона)
THEN
  C2 = 0;
  TEMP = cos(ST);
  ST = sin(ST);
  Декремент указателя вершины стека FPU;
  ST = TEMP;
ELSE
  C2 = 1
FI;
```

Описание

Команда FSINCOS вычисляет sin(ST) и cos(ST), заменяя ST синусом, а затем помещает в стек FPU косинус. ST, выраженное в радианах, должно лежать в диапазоне $|x| < 2^{63}$.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Если операнд находится вне приемлемого диапазона, то устанавливается флаг C2, а ST остается неизменным. На ответственности программиста лежит уменьшение операнда до абсолютного значения меньшего 2^{63}, путем вычитания подходящего целого, умноженного на 2π.

FSINCOS выполняется быстрее, чем FSIN и FCOS.

FSQRT Квадратный корень

IS	I	D	Z	O	U	P
*	*	*				*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 FA	FSQRT	70	86(83-87)	122-129	180-186	180-186	fsqrt

Операция

ST = квадратный корень от ST;

Описание

Команда FSQRT замещает величину в ST квадратным корнем из нее.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Квадратный корень из -0 равен -0.

На процессорах, поддерживающих стандарт IEEE-754 (Pentium,Intel486, Intel387) операция над денормализованным операндом может привести к исключению "антипереполнение". Операция над денормализованным операндом приводит к исключению "недействительная операция" на математических сопроцессорах Intel287 и 8087, не отвечающих стандарту IEEE-754, "антипереполнение" в этом случае невозможно.

Изменение обработчика исключения "антипереполнение" может потребоваться, только если необходимо по-разному обрабатывать различные коды операций. Возможно возникновение нескольких исключений "недействительная операция".

FST Сохранить вещественное
FSTP Сохранить вещественное и извлечь из стека FPU

IS	I	D	Z	O	U	P
*	*	*		*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 /2	FST m32real	2	7	44	84-90	84-90+EA	fst dword ptr [di]
DD /2	FST m64real	2	8	45	96-104	96-104+EA	fst qword ptr rmdf1
DD D0+i	FST ST(i)	1	3	11	15-22	15-22	fst ST(3)
D9 /3	FSTP m32real	2	7	44	86-92	86-92+EA	fstp dword ptr [bx]
DD /3	FSTP m64real	2	8	45	98-106	98-106+EA	fstp qword ptr [bp]
DB /7	FSTP m80real	3	6	53	52-58	52-58+EA	fstp tbyte ptr [si]
DD D8+i	FSTP ST(i)	1	3	12	17-24	17-24	fstp ST(5)

Операция

```
DEST = ST(0);
IF (команда FSTP) THEN Pop(ST); FI;
```

Описание

Команда FST копирует текущее значение из регистра ST в назначение, которое может быть: другим регистром, коротким или длинным вещественным операндом в памяти. FSTP копирует, а затем выталкивает из стека ST; она принимает также временные вещественные операнды.

Если источник - это регистр, то номер регистра используется прежде, чем произойдет выборка из стека.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Если назначение короткое или длинное вещественное, то мантисса округляется до ширины назначения в зависимости от поля RC управляющего слова, а показатель преобразуется к ширине и смещению формата назначения. Проверяется состояние переполнения и потери значащей части.

Если ST содержит нуль, +∞ (+INF), -∞ (-INF) или NAN, то мантисса не округляется, а обрезается (справа), чтобы подходить назначению. Показатель не изменяется; он также обрезается справа. Эти действия сохраняют идентичность значений, как INF, так и NAN.

Исключение "недействительная операция" (I) не возникает, когда назначение является не пустым элементом стека.

Ненормализованный операнд в ST(0) вызывает потерю значимости. Исключение "денормализованный операнд" не взводится.

FSTCW Сохранить управляющее слово FNSTCW

IS I D Z O U P

Код операции	Команда	Такты				Пример
		Pentium	486	387	287	87	
9B D9 /7	FSTCW m2byte	2+at least 1 for FWAIT		15	12-18	12-18+EA	fstcw wordSA
D9 /7	FNSTCW m2byte	2		15	12-18	12-18+EA	fnstcw ssave

Операция

DEST = CW;

Описание

Команды FSTCW и FNSTCW записывают текущее значение управляющего слова (CW) по указанному назначению.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

FSTCW проверяет на состояние размаскированных ошибок с плавающей запятой перед сохранением управляющего слова. FNSTCW - нет.

FSTENV Сохранить среду FPU
FNSTENV

IS I D Z O U P

Код операции	Команда	Такты				Пример
		Pentium	486	387	287 87	
9B D9 /6	FSTENV m14/ 28byte	16 bit rm or vm=50, 32 bit rm or vm=48, 16 bit pm=49, 32 bit pm=50; +at least 3 for FWAIT	&	103-104	40-50	40-50+EA fstenv [bp]

```
D9 /6          FNSTENV m14/ 16 bit rm or vm=50, &   103-104 40-50 40-50+EA fnstenv [si]
               28byte       32 bit rm or vm=48,
                            16 bit pm=49,
                            32 bit pm=50
```

Операция

```
DEST = среда FPU;
CW[0..5] = 111111b;
```

Описание

Команды FSTENV и FNSTENV записывают текущую среду FPU по указанному назначению, а затем маскируют все исключения с плавающей запятой. Среда FPU состоит из управляющего слова FPU, слова состояния, слова признаков (тегов) и указателей ошибки (как данных, так и команды).

Компоновка состояния в памяти зависит от размера операнда и текущего режима работы процессора и приведена на рис. 4-5..4-8 (стр. 68..70). Атрибут USE (размер операнда и размер адреса по умолчанию) текущего кодового сегмента определяет размер операнда: 14 байтный операнд применяется для сегмента USE16, и 28 байтный - для сегмента USE32.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

FSTENV и FNSTENV не сохраняют среду пока вся деятельность FPU не прекратиться. Таким образом, сохраненная среда отражает состояние FPU после декодирования и выполнения предшествующей команды.

Команды сохранения среды обычно используются обработчиками исключений, потому что они обеспечивают доступ к указателям ошибки FPU. После сохранения среды, FSTENV или FNSTENV устанавливает маски всех исключений в управляющем слове FPU. Это предотвращает прерывание обработчика исключений при ошибках с плавающей запятой.

FSTENV проверяет состояние размаскированных ошибок с плавающей запятой перед сохранением среды FPU; FNSTENV - нет.

FSTP Сохранить вещественное и извлечь из стека FPU

Смотрите FST.

FSTSW Сохранить слово состояния FPU FNSTSW

IS I D Z O U P

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
9B DD /7	FSTSW m2byte	2 + at least 3 for FWAIT	3	15	12-18	12-18+EA	fstsw wwS
9B DF E0	FSTSW AX	2 + at least 3 for FWAIT	3	13	10-16		fstsw ax
DD /7	FNSTSW m2byte	2	3	15	12-18	12-18+EA	fnstsw wwS
DF E0	FNSTSW AX	2	3	13	10-16		fnstsw ax

Операция

```
DEST = SW;
```

Описание

Команды FSTSW и FNSTSW записывают текущее значение слова состояния FPU (SW) по указанному назначению, которое может быть, либо двухбайтным операндом в памяти, либо регистром AX.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

FSTSW проверяет состояние размаскированных ошибок с плавающей запятой перед сохранением среды FPU; FNSTSW - нет. FSTSW и FNSTSW в первую очередь используются в условном ветвлении (после команд сравнения, FPREM, FPREM1 или FXAM). Они также могут быть использованы при активизации обработчиков исключений (при упорядоченном опросе битов исключений) в среде не использующей прерывания.

Когда выполнена FNSTSW AX , регистр AX обновляется прежде, чем процессор Pentium выполнит следующую команду. Сохраняемое состояние такое же, как после выполнения предыдущей ESC команды.

FSUB Вычитание
FSUBP Вычитание с извлечением из стека FPU
FISUB Целочисленное вычитание

IS	I	D	Z	O	U	P
*	*	*		*	*	*

Код	Команда	Pentium	486	Такты 387	287	87	Пример
D8 /4	FSUB m32real	3/1	7(5-17)	24-32	90-120	90-120+EA	fsub dword ptr [si]
DC /4	FSUB m64real	3/1	7(5-17)	28-36	95-125	95-125+EA	fsub qword ptr [bx]
D8 E0+i	FSUB ST,ST(i)	3/1	7(5-17)	26-37	70-100	70-100	fsub ST,ST(4)
DC E8+i	FSUB ST(i),ST	3/1	7(5-17)	26-37	70-100	70-100	fsub ST(2),ST
DE E8+i	FSUBP ST(i),ST	3/1	7(5-17)	26-37	75-105	75-105	fsubp ST(3),ST
DE E9	FSUBP ST(1),ST	3/1	7(5-17)	26-37	75-105	75-105	fsubp
DE /4	FISUB m16int	7/4	23(19-32)	71-83	102-137	102-137+EA	fisub word ptr puls
DA /4	FISUB m32int	7/4	24(20-35)	57-82	108-143	108-143+EA	fisub dword ptr [si]

Операция

```
DEST = DEST - SRC;
IF (команда FSUBP) THEN Pop(ST); FI;
```

Описание

Команды вычитания вычитают операнд-источник из операнда-назначения и возвращают разность на место операнда-назначения. (Однооперандные формы команд вычитания используют в качестве операнда-назначения ST.) Команда FSUBP, кроме этого, выталкивает из регистрового стека FPU верхний элемент (ST). Если в мнемонике команды FSUBP не записано никакого операнда, то ST вычитается из ST(1).

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Если операнд-источник находится в памяти, то он автоматически приводится к формату временного вещественного.

FSUBR Инверсное вычитание
FSUBRP Инверсное вычитание с извлечением из стека FPU
FISUBR Целочисленное инверсное вычитание

IS	I	D	Z	O	U	P
*	*	*		*	*	*

Код	Команда	Такты				Пример
		Pentium	486	387	287	87	
D8 /5	FSUBR m32real	3/1	7(5-17)	25-33	90-120	90-120+EA	fsubr dword ptr[si]
DC /5	FSUBR m64real	3/1	7(5-17)	29-37	95-125	95-125+EA	fsubr qword ptr[bx]
D8 E8+i	FSUBR ST,ST(i)	3/1	7(5-17)	26-37	70-100	70-100	fsubr ST,ST(4)
DC E0+i	FSUBR ST(i),ST	3/1	7(5-17)	26-37	70-100	70-100	fsubr ST(2),ST
DE E0+i	FSUBRP ST(i),ST	3/1	7(5-17)	26-37	75-105	75-105	fsubrp ST(3),ST
DE E1	FSUBRP ST(1),ST	3/1	7(5-17)	26-37	75-105	75-105	fsubrp
DE /5	FISUBR m16int	7/4	23(19-32)	72-84	102-137	103-139+EA	fisubr word ptr[di]
DA /5	FISUBR m32int	7/4	24(20-35)	58-83	108-143	109-144+EA	fisubr memo

Операция

```
DEST = другой операнд - ST;
IF (команда FSUBRP) THEN Pop(ST); FI;
```

Описание

Команды инверсного вычитания вычитают операнд-назначение из операнда-источника и возвращают разность на место операнда-назначения. (Однооперандные формы команд инверсного вычитания используют в качестве операнда-назначения ST.) Команда FSUBRP, кроме этого, выталкивает из регистрового стека FPU верхний элемент (ST). Если в мнемонике команды FSUBRP не записано никакого операнда, то ST(1) вычитается из ST.

Особые ситуации защищенного режима

#GP(0), если недопустимый эффективный адрес операнда в памяти в сегментах CS, DS, ES, FS или GS; #SS(0), если недопустимый адрес в сегменте SS; #PF(код ошибки), страничная ошибка; #NM, если либо EM, либо TS установлены в CR0; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Особые ситуации режима реальной адресации

Прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh; Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

Такие же, как и в режиме реальной адресации. #PF(код ошибки), страничная ошибка; #AC, при невыровненной ссылке в память, при текущем уровне привилегий равном 3.

Замечание

Если операнд-источник находится в памяти, то он автоматически приводится к формату временного вещественного.

FTST Проверка

IS	I	D	Z	O	U	P
*	*	*				

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 E4	FTST	4/1	4	28	38–48	38–48	ftst

Операция

```
CASE (отношение операндов) OF
  Не сравнимые:   C3, C2, C0 = 111;
  ST > SRC: C3, C2, C0 = 000;
  ST < SRC: C3, C2, C0 = 001;
  ST = SRC: C3, C2, C0 = 100;
ESAC;
```

Таблица соответствия флагов

Флаги FPU	Флаги EFLAGS
C0	CF
C1	Нет
C2	PF
C3	ZF

Описание

Команда FTST сравнивает значение в вершине стека с 0.0.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Если операнд - SNAN, представлен в неопределенном формате, либо произошла стековая ошибка, то возникает исключение "недействительная операция" (I), и биты состояния устанавливаются в "несравнимо" (C2=1).

Знак нуля игнорируется, так что -0 равен +0.

FUCOM Сравнение неупорядоченных вещественных
FUCOMP Сравнение неупорядоченных вещественных с извлечением одного из стека FPU
FUCOMPP Сравнение неупорядоченных вещественных с извлечением обоих из стека FPU

IS	I	D	Z	O	U	P
*	*	*				

Код операции	Команда	Такты			Пример
		Pentium	486	387	
D0 E0+i	FUCOM ST(i)	4/1	4	24	fucom ST(3)
DD E1	FUCOM ST(1)	4/1	4	24	fucom
DD E8+i	FUCOMP ST(i)	4/1	4	26	fucomp ST(4)
DD E9	FUCOMP ST(1)	4/1	4	26	fucomp
DA E9	FUCOMPP ST(1)	4/1	5	26	fucompp

Операция

```
CASE (отношение операндов) OF
  Не сравнимые:   C3, C2, C0 = 111;
  ST > SRC: C3, C2, C0 = 000;
  ST < SRC: C3, C2, C0 = 001;
  ST = SRC: C3, C2, C0 = 100;
ESAC;
IF (команда FUCOMP) THEN Pop(ST); FI;
IF (команда FUCOMPP) THEN Pop(ST); FI;
```

Таблица соответствия флагов

Флаги FPU	Флаги EFLAGS
C0	CF
C1	Нет
C2	PF
C3	ZF

Описание

Команды сравнения неупорядоченных действительных сравнивают значение в вершине стека с источником, которой должен быть регистром. Если в мнемонике не указано никакого операнда, то ST сравнивается с ST(1). Команда FUCOMP, кроме этого, выталкивает из стека верхний элемент (ST). Команда FUCOMPP - два верхних элемента.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Если операнд - SNAN, представлен в неопределенном формате, либо произошла стековая ошибка, то возникает исключение "недействительная операция" (I), и биты состояния устанавливаются в "не сравнимо" (C2=1).

В случае QNAN операнда, биты состояния устанавливаются в "не сравнимо" (C2=1). В отличие от обычного сравнения (FCOM и т.п.), команды сравнения неупорядоченных вещественных не вызывают исключения "недействительная операция" (I) в случае QNAN операнда.

Знак нуля игнорируется, так что -0 равен +0.

FWAIT Ожидать

IS I D Z O U P

Код операции	Команда	Такты			Пример
		Pentium	486	387	
9B	FWAIT	1-3	1-3	3-5n*	fwait

* n = число проверок линии BUSY, производимых CPU, пока 80387 завершает выполнение предыдущей инструкции.

Описание

Команда FWAIT заставляет процессор проверять отложенные размаскированные числовые исключения, прежде чем продолжить выполнение.

В процессорах Pentium, Intel486, Intel387 команды сопроцессора автоматически синхронизируются (процессор автоматически ожидает окончания выполнения предыдущей команды сопроцессора перед запуском следующей). Для 8087, который используется совместно с 8086 или 8088, необходимо наличие команды WAIT перед каждой числовой командой для гарантии синхронизации. Программы 8087, имеющие такие синхронизирующие команды WAIT, могут без реассемблирования выполняться на 32 разрядных процессорах, однако для сокращения кода и времени выполнения рекомендуется удаление лишних инструкций WAIT.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Как показывает код операции, FWAIT на самом деле не ESC команда, а альтернативная мнемоника для WAIT. Кодируя FWAIT после ESC команды, убедитесь в том, что любое размаскированное исключение с плавающей запятой, вызываемое этой командой, обрабатывается прежде, чем процессор имеет шанс изменить результаты команды.

FXAM Проверить ST

IS I D Z O U P

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 E5	FXAM	21	8	30-38	12-23	12-23	fxam

Операция

```
C1 = знаковый разряд ST; (* 0 для положительного, 1 для отрицательного *)
CASE (тип объекта в ST) OF
  неподдерживаемый:  C3, C2, C0 = 000;
  NAN:               C3, C2, C0 = 001; (* не число *)
  нормальный:  C3, C2, C0 = 010;
  INF:               C3, C2, C0 = 011;
  нуль:              C3, C2, C0 = 100;
  пусто:             C3, C2, C0 = 101;
  ненормализованное:  C3, C2, C0 = 110;
ESAC;
```

Таблица соответствия флагов

Флаги FPU	Флаги EFLAGS
C0	CF
C1	Нет
C2	PF
C3	ZF

Описание

Команда FXAM сообщает тип объекта содержащийся в регистре ST путем установки флагов FPU.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Бит C1 представляет знак ST(0), не обращая внимания на то, пуст ST(0) или нет.

Сталкиваясь с пустым регистром на процессорах Pentium, Intel486 и Intel387 команда FXAM не будет генерировать комбинации в C3..C0 равные 1101 и 1111. Сопроцессоры Intel287 и 8087 могут генерировать эти комбинации в числе прочих. Это не влияет на программное обеспечение и обеспечивает расширение возможностей.

FXCH Поменять содержимое регистра

IS I D Z O U P
*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 C8+i	FXCH ST(i)	1	4	18	10-15	10-15	fxch ST(2)
D9 C9	FXCH ST(1)	1	4	18	10-15	10-15	fxch

Операция

```
TEMP = ST;
ST = DEST;
DEST = TEMP;
```

Описание

Команда FXCH меняет местами содержимое регистра назначения и вершины стека. Если назначение не указано явно, то используется ST(1).

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Многие числовые команды оперируют только с вершиной стека. FXCHG обеспечивает простое средство для использования этими командами нижних элементов стека. Например, следующая последовательность берет квадратный корень из третьего от вершины стека регистра (предполагается, что ST не пуст):

```
FXCH ST(3)
FSQRT
FXCH ST(3)
```

FXCH может быть объединена в пару с некоторыми командами с плавающей запятой (такими как: FADD, FSUB, FMUL, FLD, FCOM, FUCOM, FCHS, FTST, FABS, FDIV. Этот набор также включает FADDP, FSUBRP и т.п.). В паре FXCH выполняется параллельно и не требует дополнительных тактов (только Pentium).

FXTRACT Выделить показатель и мантиссу

IS	I	D	Z	O	U	P
*	*	*	*			

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 F4	FXTRACT	13	19(16-20)	70-76	27-55	27-55	fxtract

Операция

```
TEMP = значение ST;
ST = экспонента ST;
Декремент указателя вершины стека FPU;
ST = TEMP;
```

Описание

Команда FXTRACT разделяет значение в ST на его показатель и мантиссу. Показатель замещает исходный операнд в стеке, а мантисса помещается в стек. После выполнения FXTRACT ST (новая вершина стека) содержит значение исходной мантиссы, представленной вещественным числом: знак тот же, что и у операнда, его показатель равен истинному нулю (значение 16383 или 3FFFh), мантисса идентична мантиссе исходного операнда. ST(1) содержит значение несмещенного показателя исходного

операнда, выраженное вещественным числом. Чтобы проиллюстрировать действие FXTRACT, предположим, что ST содержит число с показателем равным +4 (т.е. поле показателя содержит 4003h). После выполнения FXTRACT, ST(1) будет содержать вещественное число +4.0 (знак положительный, показатель равен 4001h (+2 истинное), поле мантиссы содержит 1.00...00b). Другими словами значение в ST(1) будет равно $1.0 * 2^2 = 4$. Если ST содержит операнд, с истинным показателем равным -7 (т.е. поле показателя содержит 3FF8h), тогда FXTRACT возвратит показатель равный -7.0; после выполнения команды поля знака и показателя будут содержать C001h (отрицательный знак, истинный показатель равен 2), и мантисса будет равна 1.1100...00b. Другими словами значение в ST(1) будет равно $-1.75 * 2^2 = -7.0$. В обоих случаях после FXTRACT поля знака и мантиссы ST будут такими же, что и у исходного операнда, а поле показателя будет содержать 3FFFh (истинный 0).

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

FXTRACT представляет подмножество функций $\log_b(x)$, рекомендованных IEEE. Если исходный операнд равен нулю, то FXTRACT оставляет -INF в ST(1) (показатель), тогда как ST присваивается нуль со знаком равным знаку исходного операнда. В этом случае возникает исключение "деление на нуль" (Z) (смотрите замечание по совместимости).

ST(7) должен быть пустым, во избежание исключения "недействительная операция" (I).

FXTRACT полезна для операций возведения в степень и масштабирования. FXTRACT и команда потенцирования двойки F2XM1 необходимы для использования общей команды возведения в степень. Преобразование числа из временного вещественного формата к десятичному представлению (например, для печати или вывода на дисплей) требует не только FST, но

так же и FXTRACT, чтобы допустить масштабирование, которое не предполагает диапазон временного вещественного формата. FXTRACT также может быть полезна для отладки, потому что она позволяет раздельно наблюдать показатель и мантиссу вещественного числа.

FXTRACT на процессорах Pentium, Intel486 и Intel387, если операндом является нуль, вызывает исключение "деление на нуль" и в ST(1) заносится -INF. Если операндом является +INF, исключения не возникает. Для сопроцессоров Intel287 и 8087, если операндом является нуль, ST(1) обнуляется и исключения не возникает, а если операндом является +INF, возникает исключение "недействительная операция".

Это не влияет на программное обеспечение которое обычно обходит нуль и INF. Изменение было рекомендовано IEEE-754 для полной поддержки функции $\log_b(x)$.

FYL2X Вычислить y * $\log_2 x$

IS	I	D	Z	O	U	P
*	*	*	*	*	*	*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 F1	FYL2X	22-111	311(196-329)	120-538	900-1100	900-1100	fyl2x

Операция

```
ST(1) = ST(1) * log2ST;
Pop(ST);
```

Описание

Команда FYL2X вычисляет логарифм от ST по основанию 2, умножает логарифм на ST(1) и помещает результирующее значение в ST(1). Операнд в ST не может быть нулем или отрицательным.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Если операнд в ST отрицательный, то возникает исключение "недействительная операция".

Команда FYL2X разработана со встроенным умножением для оптимизации вычисления логарифмов с произвольным положительным основанием: $\log_b x = (\log_2 b)^{-1} * \log_2 x$.

Команды FLDL2T и FLDL2E загружают константы $\log_2 10$ и $\log_2 e$ соответственно.

FYL2XP1 Вычислить y * $\log_2(x+1)$

IS	I	D	Z	O	U	P
*	*	*			*	*

Код операции	Команда	Такты					Пример
		Pentium	486	387	287	87	
D9 F9	FYL2XP1	22-103	313(171-326)	257-547	700-1000	700-1000	fyl2xp1

Операция

```
ST(1) = ST(1) * log2(ST + 1.0);
Pop(ST);
```

Описание

Команда FYL2XP1 вычисляет логарифм от (ST + 1.0) по основанию 2, умножает логарифм на ST(1) и возвращает результирующее значение в ST(1), а затем выталкивает из стека ST. Операнд в ST должен быть в диапазоне: $(\sqrt{2} \div 2) - 1 \le ST \le \sqrt{2} - 1$.

Особые ситуации защищенного режима

#NM, если либо EM, либо TS установлены в CR0.

Особые ситуации режима реальной адресации

Прерывание 7, если либо EM, либо TS установлены в CR0.

Особые ситуации режима V86

#NM, если либо EM, либо TS установлены в CR0.

Замечание

Если операнд в ST находится вне приемлемого диапазона, то результат FYL2XP1 неопределен.

Команда FYL2XP1 обеспечивает более высокую точность, чем FYL2X при вычислении логарифмов близких к 1. Когда ε мало (ε - величина, логарифм которой необходимо найти, минус единица), большее количество значащих цифр может быть сохранено применением ε в качестве аргумента FYL2XP1, нежели применением 1 + ε в качестве аргумента FYL2X.

7. Оптимизация кода для 32-х разрядных процессоров

Цели этой главы - описать особенности архитектуры процессоров Intel486 и Pentium и предложить стратегию оптимизации, которая дает выигрыш для всех членов семейства.

Процессор Pentium способен выполнять все коды, созданные для Intel386, Intel486 без изменений. Однако, возможна оптимизация кода конкретно для каждого типа процессора, которая ускоряет выполнение кода. Большинство таких оптимизаций имеют дело с селекцией или перестановкой команд в программе в соответствии с архитектурой процессора.

7.1 Обзор процессоров Intel386, Intel486 и Pentium

Процессор Intel386

Intel386 содержит 32 битные шины данных, широкий набор 32 разрядных режимов адресации и внутрикристальное управление памятью.

Команды предвыбираются из внешней памяти и хранятся в буфере предвыборки, глубиной на 4 строки по 4 байта в каждой. Декодер инструкций питается из этого буфера.

Декодировщик инструкций размещает декодированную информацию в FIFO глубиной 3. Он связан как с предвыборкой, так и с исполнительным ядром отдельными протоколами.

Исполнительное ядро выполняет приходящие команды по одной за такт, но в некоторых случаях возможно перекрытие последнего цикла текущей команды с вычислением эффективного адреса следующей команды обращения к памяти.

Связь предвыборки /декодирования/ исполнения и последовательная природа исполнительного ядра выдвинули с точки зрения компиляции определенные требования к планированию команд. Например, следует избегать индексирования при вычислении эффективного адреса; следует выбирать короткие команды вместо сложных.

Процессор Intel486

В Intel486 был введен мощный целочисленный конвейер, обеспечивающий в пике исполнение одной команды за такт. Он имеет объединенный КЭШ кода и данных первого уровня и FPU на кристалле, кроме того на кристалле имеется то же устройство управления памятью, что и в Intel386.

В i486 имеется пятиступеньчатый целочисленный конвейер:

1. **Prefetch (PF)** - предвыборка. Команда извлекается из КЭШ и размещается в одном из двух 16 байтных буферов.
2. **Decode (D1)** - декодирование. Приходящий поток команд декодируется. Префиксные команды остаются в D1 два такта.
3. **Adress Generation (D2)** - генерация адреса. Эффективный и линейный адреса вычисляются параллельно. Обычно генерация адреса занимает один такт, кроме случаев, когда используется индексация, тогда нужно 2 такта.
4. **Execution (E)** - исполнение в АЛУ и доступ в КЭШ. Простые команды (с одной машинной операцией) выполняются за 1 такт, сложные - требуют несколько тактов.
5. **Writeback (WB)** - обратная запись. Производится коррекция содержимого регистров.

Выполненное ветвление прерывает конвейер команд и вызывает два такта штрафа всякий раз, когда конвейер бездействует в случае нереализованного перехода.

Внутрикристальный КЭШ объединен для команд и данных. Он имеет объем 8 Кбайт, 4 ассоциативных входа с псевдо-LRU (Least Recently Used) алгоритмом замещения. Обращение за данными в КЭШ имеет приоритет по сравнению с выборкой команд.

Плавающее АЛУ использует целочисленный конвейер для упреждающей выборки данных. Размер шин позволяет выполнять обмен 64 битными данными между КЭШ и FPU за 1 такт. Также возможно перекрытие в выполнении целых и плавающих операций.

Ядро Intel486 спроектировано для быстрого выполнения часто используемых команд. Значит надо тщательно выбирать команды для программы. Необходимо также планирование, чтобы избежать остановок конвейера.

Процессор Pentium

Это суперскалярный процессор, так как он может выполнять две целочисленные команды одновременно. Прозрачный для программного обеспечения механизм динамического предсказания ветвлений минимизирует простои из-за ветвлений.

Имеется два параллельных целочисленных конвейера, главный (U), который является усовершенствованным конвейером от МП Intel486 и вторичный конвейер (V), который подобен главному, но имеет некоторые ограничения на выполняемые команды. Эти ограничения будут описаны далее.

Pentium может запускать две команды за такт. Во время исполнения, следующие две команды проверяются и, если возможно, они запускаются так, чтобы первая команда выполнялась в U, а вторая - в V конвейере.

Когда команды выполняются в двух конвейерах, их исполнение точно такое же, как если бы они выполнялись последовательно.

На первой ступени (PF) инструкции выбираются из КЭШ или памяти. Поскольку имеется два независимых КЭШ, при выборке команд конфликты отсутствуют. Если в КЭШ нет нужной инструкции (промах), производится обращение в память. На PF ступени две пары независимых буферов выборки, каждый размером в одну строку (32 байта), работают совместно с буфером ветвления BTB. Это позволяет одному буферу выбирать инструкции последовательно, а другому - согласно предсказаниям BTB.

Следующая ступень D1 обеспечивает декодирование, чтобы запустить две следующие команды. На этом этапе решается, одна или две команды могут быть запущены.

На ступени D2 вычисляется адрес операндов, расположенных в памяти. В Intel486 команды, содержащие одновременно смещение и непосредственный операнд, или команды, содержащие базу и индекс, требуют дополнительного такта. В Pentium на ступени генерации адреса (D2) можно выполнять многовходовое сложение, поэтому индексация не добавляет такта, как в Intel486.

На ступени EX происходит исполнение в АЛУ и доступ в КЭШ данных. Поэтому те команды, которые выполняют АЛУ-операции и доступ в КЭШ данных будут требовать более одного такта на этой ступени. На ступени EX все команды U и V конвейеров, исключая условные переходы, проверяются для обеспечения правильного предсказания переходов.

На последней ступени WR осуществляется обратная запись, команды должны модифицировать состояние процессора и завершить исполнение. На этой ступени V конвейерные условные переходы проверяются на правильность предсказания.

На протяжении прохождения через конвейер команда может быть задержана по некоторым причинам, которые будут рассматриваться ниже. Инструкции поступают в U и V конвейеры и покидают их в унисон. Поэтому в обоих конвейерах переход на ступень EX происходит одновременно. Если в конвейере U команда задержана, то команда в конвейере V также задерживается. Если команда задержана в конвейере V, то спаренной команде в конвейере U можно продолжить выполнение. Никаким следующим командам не позволяется входить в ступень EX (исполнение) конвейера до тех пор, пока в обоих конвейерах команды не перейдут на ступень WB (обратная запись).

В Pentium имеется 2 КЭШ по 8 Кбайт по два ассоциативных входа со строками длиной 32 байта. Имеется 64 битной ширины интерфейс шины. КЭШ'и используют WR-механизм и LRU(Least Recently Used)-алгоритм замещения. КЭШ данных состоит из 8 банков, перекрывающихся по 4 битовым границам. В КЭШ данных может осуществляться одновременный доступ от обоих конвейеров, до тех пор, пока обращения происходят в разные банки. Минимальная задержка из-за промаха КЭШ составляет 3 такта.

В блоке предвыборки команд имеется 4 буфера, каждый длиной 32 байта. Предвыборщик может извлекать команду, которая разделена между двух строк, без потерь тактов. Ввиду разделенности КЭШ исчезли конфликты при обращении за данными и командами.

На ступени PF(предвыборка) два буфера выборки работают одновременно. Только один буфер активно требует выборки в любое заданное время. Выборки осуществляются последовательно до тех пор, пока не извлечена команда ветвления. Когда это происходит, буфер ветвления (BTB) предсказывает, будет ветвление иметь место или нет. Если предсказано, что ветвления не будет, требования на выборку продолжаются линейно. Если предсказано ветвление, другой буфер выборки включается и начинает выборку по адресу перехода. Если предсказанное ветвление не выполнилось, то конвейер очищается и предвыборка начинается сначала.

Pentium использует схему динамического предсказания переходов с 256 входовым буфером меток переходов (Branch Target Buffer). Если предсказание корректное, то штрафа при выполнении инструкции перехода

нет. Имеется 3 такта штрафа, если условный переход был выполнен в U-конвейере или 4 такта штрафа, если это было в V-конвейере. Неверно предсказанные вызовы и безусловные переходы имеют 3 такта штрафа в любом конвейере. В Intel486 осуществленные переходы вызывают 2 такта штрафа.

Конвейер FPU имеет 8 ступеней, первые пять из них являются общими с целочисленным конвейером.

1. **PF** - предвыборка;
2. **D1** - декодирование;
3. **D2** - генерация адреса;
4. **EX** - чтение памяти и регистра; преобразование данных FPU к формату внешней памяти и запись в память;
5. **X1** - исполнительная ступень 1 FPU; преобразование внешнего формата памяти во внутренний формат FPU и запись операнда в регистровый файл FPU; обход 1;
6. **X2** - исполнительная ступень 2 FPU;
7. **WF** - выполнение округления и запись FP результата в регистровый файл; обход 2;
8. **ER** - сообщения об ошибках (коррекция слова состояния).

Благодаря обходам данные становятся доступными даже перед действительной их записью в регистры.

Целочисленные команды проходят через 5 первых ступеней и используют пятую (X1) ступень как WB ступень.

Большинство наиболее часто используемых инструкций конвейеризовано так, чтобы конвейер мог принимать новую пару операндов каждый такт. Вследствие этого хороший генератор кода должен обеспечивать выход почти до 1 команды за такт (конечно, предполагается, что есть программа с нужной величиной естественного параллелизма).

Команда FXCH может выполняться параллельно с большинством команд FPU. Она позволяет генератору кода или программисту обрабатывать стек FPU как регулярный регистровый файл без деградации характеристик.

7.2 Примеры целочисленных программ

Предлагаемые ниже примеры позволяют подсчитать по тактам время выполнения программ. Примеры предполагают 100% удачное обращение к КЭШ и бесконфликтный доступ в память. Также предполагается 32 битное адресное пространство.

Имеется счет тактов, соответствующий каждой команде. Счет тактов может быть и не связан с командой, когда речь идет о простое конвейера. В примерах также предполагается, что предсказание ветвления является правильным.

Анализ этих примеров дает интуитивное ощущение того, как работают и почему простаивают спаренные конвейеры.

```
Исходный текст на Си:
  static int a[10],b[10];
  int i;
  for(i = 0; i < 10; i++)
     {
        a[i] = a[i] + 1;
        b[i] = b[i] + 1;
     }
```

Возможны различные правильные коды для этого текста, однако их эксплуатационные характеристики могут быть различны. Рассмотрим 3 примера:

Последовательность 1

```
    xor  eax,eax
TopOfLoop:
    mov  edx,eax
    shl  edx,2
    inc  dword ptr [edx+a]
    mov  edx,eax
    shl  edx,2
    inc  dword ptr [edx+b]
    inc  eax
    cmp  eax,10
    jl   TopOfLoop
```

Последовательность 2

```
    xor  eax,eax
TopOfLoop:
    inc  dword ptr [edx*4+a]
    inc  dword ptr [edx*4+b]
    inc  eax
    cmp  eax,10
    jl   TopOfLoop
```

Последовательность 3

```
    mov  eax,-40
TopOfLoop:
    mov  edx,[eax+40+a]
    mov  ecx,[eax+40+b]
    inc  edx
    inc  ecx
    mov  [eax+40+a],edx
    mov  [eax+40+b],ecx
    add  eax,4
    jnz  TopOfLoop
```

Код 1 имеет преимущество из-за удаления общих подвыражений. Это не оптимизированный код, скорее, это не полностью оптимизированный код. Неоптимизированный код не содержал бы счетчик (i) в регистре.

Код 2 - наиболее прямолинейный код.

Код 3 использует load/store модель. Он также включает некоторые оптимизации с исключением индуктивных переменных с тестовым перемещением. Счетчик циклов в eax считает до нуля. Когда 0, jnz не выполняется. Этот код избегает команд сравнения.

Далее характеристики каждого из этих трех кодов исследуются для Intel486 и Pentium.

Код 1 для Intel486

Команда SHL занимает 2 такта на i486. АЛУ операции (например, add) с памятью требует 3 такта: 1 - загрузка, 1 - сложение и 1 - запоминание.

```
mov  edx,eax              ; 1
shl  edx,2                ; 2
  (shl)                   ; 3
inc  dword ptr [edx+a]    ; 4
```

```
  (inc)                    ; 5
  (inc)                    ; 6
  (inc)                    ; 7
mov  edx,eax               ; 8
shl  edx,2                 ; 9
  (shl)                    ; 10
inc  dword ptr [edx+b]     ; 11
  (inc)                    ; 12
  (inc)                    ; 13
  (inc)                    ; 14
inc  eax                   ; 15
cmp  eax,10                ; 16
jl  TopOfLoop              ; 17
                           ; 18
                           ; 19
                           ; 20
mov  edx,eax               ; 21  следующая итерация

Всего: 20 тактов
```

Циклы 4 и 11 имели задержку из-за блокировки генерации адреса (Adress Generation Interlock). Регистр edx был записан в цикле 3 и использован как базовый в цикле 5. Когда регистр используется для вычисления эффективного адреса в цикле после того, как регистр записан в предыдущем цикле, имеет место штраф 1 такт. Это происходит, потому что вычисление адреса выполняется на ступени D2.

Для ассемблера, используемого для этого кода, команда jl была "jump near", но не "jump short". "Jump near" это команда с префиксом 0Fh. Префиксная команда требует дополнительного такта на Intel486 на ступени D1. i486 не имеет никакого механизма предсказания ветвлений. Когда переход имеет место, получается 2 такта штрафа (такты 19 и 20).

Код 1 для Pentium

```
U конвейер                V конвейер
mov  edx,eax                              ; 1
shl  edx,2                                ; 2
inc  dword ptr [edx+a]                    ; 3
  (inc)                                   ; 4
  (inc)                                   ; 5
  (inc)                   mov  edx,eax    ; 6 пара
shl  edx,2                                ; 7
inc  dword ptr [edx+b]                    ; 8
```

```
  (inc)                                ; 9
  (inc)                                ; 10
  (inc)                    inc  eax    ; 11 пара
cmp  eax,10                jl  TopOfLoop  ; 12
mov  edx,eax                ; 13 следующая итерация

Всего: 12 тактов
```

Заметим, что команда shl требует 2 такта на Intel486 и только один такт на Pentium. Pentium имеет специальную аппаратуру, чтобы избежать 0Fh префиксной задержки на jcc near командах. Он также может иметь пары из команд compare и jump, даже если cmp задает флаг условия, и jl читает его. Аппаратура предсказания ветвлений, когда она предсказывает переход, может выполнять команду по метке в цикле, следующем за jump. Когда многотактная команда в U - конвейере спаривается с другой командой, последняя операция памяти команды в U - конвейере образует пару с первой операцией команды V - конвейера (такты 6 и 11).

Код 2 для i486

```
inc  dword ptr [eax*4+a] ; 1
  (inc)                  ; 2
  (inc)                  ; 3
  (inc)                  ; 4
inc  dword ptr [eax*4+b] ; 5
  (inc)                  ; 6
  (inc)                  ; 7
  (inc)                  ; 8
inc  eax                 ; 9
cmp  eax,10              ; 10
jl  TopOfLoop            ; 11
                         ; 12
                         ; 13
                         ; 14
inc  dword ptr [eax*4+a] ; 15 следующая итерация

Всего: 14 тактов
```

На Intel486 из-за индексации имеется такт штрафа на ступени D2 (циклы 1 и 5). Это относится к базовым регистрам.

Код 2 для Pentium

```
U конвейер                  U конвейер
inc  dword ptr [eax*4+a]                          ; 1
  (inc)                                           ; 2
  (inc)                     inc dword ptr [eax*4+b] ; 3
                              (inc)                 ; 4
                              (inc)                 ; 5
inc  eax                                          ; 6
cmp  eax,10                 jl  TopOfLoop         ; 7
inc dword ptr [eax*4+a]                           ; 8

Всего: 7 тактов
```

Команда inc eax в такте 6 не образовывала пары с cmp, поскольку имеется регистровая зависимость. Кроме нескольких специальных случаев (как cmp-jmp) регистр не может быть доступен до такта его записи.

Код 3 для Intel486

```
mov  edx,[eax+40+a]      ; 1
                         ; 2
mov  ecx,[eax+40+b]      ; 3
inc  edx                 ; 4
inc  ecx                 ; 5
mov  [eax+40+a],edx      ; 6
mov  [eax+40+b],ecx      ; 7
add  eax,4               ; 8
jnz  TopOfLoop           ; 9
                         ; 10
                         ; 11
                         ; 12
mov  edx,[eax+40+a]      ; 13 следующая итерация

Всего: 12 тактов
```

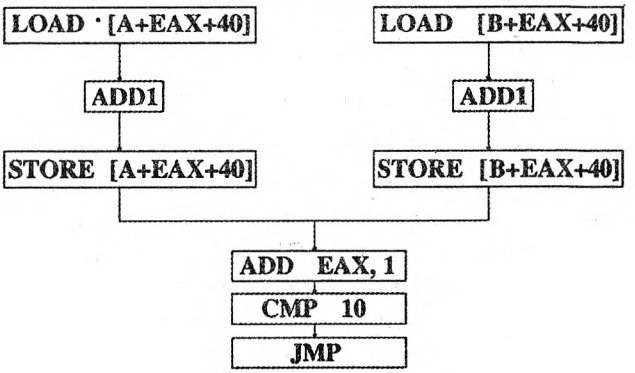
Задержка на такте 2 вызвана промахом буфера предвыборки Intel486. Предыдущие два примера имели подобный штраф, однако это было скрыто двумя тактами, использованными для команды shl в коде 1, и из-за индексного штрафа в коде 2.

Код 3 для Pentium

```
U конвейер                  V конвейер
mov  edx,[eax+40+a]         mov  ecx,[eax+40+b]    ; 1
inc  edx                    inc  ecx               ; 2
mov  [eax+40+a],edx         mov  [eax+40+b],ecx    ; 3
add  eax,4                  jnz  TopOfLoop         ; 4
mov  edx,[eax+40+a]         mov  ecx,[eax+40+b]    ; 5
  (mov)                       (mov)                ; 6

Всего: 5 тактов
```

Задержка буфера предвыборки Intel486 больше здесь не действует. Pentium имеет больше буферов предвыборки и различные возможности выравнивания. В выше приведенном примере управление циклом определяется командой add eax,4 , устанавливающей код нулевого условия, как только счет дойдет до 0. Имеется AGI (блокировка генерации адреса) на eax, поскольку add в такте 4 пишет в eax, и обе команды mov в такте 5 обращаются к нему, несмотря на ветвление между ними. На Intel486 AGI проявляется только между соседними командами. На Pentium может быть две команды в промежутке, и все еще будет AGI, например, между командой add в U-конвейере в такте n и командой mov в V-конвейере, которая использует результат add как базу в такте n+1. Общее правило состоит в том, что AGI будет иметь место, когда любая команда в такте n пишет в регистр, который используется для вычисления эффективного адреса в такте n+1. Это происходит, потому что вычисление адреса выполняется на ступени D2. В промежуточном представлении компилятора (перед планированием) программа 3 может иметь вид:



Упорядочение промежуточного кода, для получения ассемблерного кода, показанное ранее, применяет перемещение load из "b" перед store в "a". Перестановка команд требует знания, какие операнды независимы.

Парадигма load/store работает хорошо на Pentium, поскольку это обеспечивает больше возможностей для образования пар и планирования. Это, однако, не увеличивает числа тактов даже без планирования, хотя игнорируются возможные вторичные эффекты, такие как больший размер кода, что ведет к промахам КЭШ. Другой вторичный эффект - использование большего количества регистров, которые являются ограниченным ресурсом данного процессора. Компиляторные разработки должны уделить больше внимания распределению регистров.

В дальнейшем будем называть это как "load/store" стиль генерации кодов.

7.3 Стратегия генерации кодов

Хотя каждый член семейства 386 имеет различную микроархитектуру, однако это не сильно влияет на общую стратегию оптимизации. В самом деле, имеется ряд смешанных оптимизаций, которые будут создавать оптимальный код для всего семейства. Смешанные оптимизации включают.

1. Оптимизации, полезные всем членам семейства.
2. Оптимизации для выделенных членов семейства, но не для всех; при этом для остальных членов семейства не ухудшают характеристик.
3. Оптимизации полезные одному или двум членам семейства и вредные для остальных членов; в этом случае надо иметь возможность по желанию пользователя выключать оптимизацию.

7.3.1 Планирование целых команд

Планирование есть процесс перестановки команд с целью избежать остановок и задержек при сохранении логики программы.

Планирование целочисленных команд имеет две цели.

1. Исключить остановки в Intel486 конвейере и любом конвейере Pentium. Есть много ситуаций, когда встречаются задержки. Генеральная линия в этом случае - вставить независимую команду. Поскольку большинство часто используемых команд выполняется за один такт, нет особой необходимости скрывать

задержки. Наиболее общий вид задержек, которые можно избежать, это - задержки из-за блокировки генерации адреса (AGI).

2. Создание пар для использования суперскалярной архитектуры Pentium. Имеются некоторые ограничения на образование пар, и некоторые пары, даже уже запущенные не будут выполнятся в параллель. Детали образования пар описаны ниже. Дополнительная информация содержится в таб. 8-1.

Перестановка команд нужна, чтобы увеличить выпуск пар. Зависимые команды должны быть разделены по крайней мере одной независимой. Планирование пар имеет малое влияние на характеристики Intel486.

7.3.2 Выбор индексной версии базового регистра

Intel386 и Intel486 требуют дополнительного такта из-за индексации. Поэтому, если используется только один индексный компонент (т.е. не два: индекс и база) и индексирование не является необходимым, то быстрее использовать регистр как базу, чем как индекс. Пример:

```
mov  eax,[esi]   ; esi используется как база
mov  eax,[esi*]  ; esi используется как индекс,
                 ; 1 такт штрафа
```

Это вычисление адреса требует в Pentium одного такта, даже если используется индексация. Поэтому Pentium нейтрален по отношению к выбору индексной версии базового регистра.

7.3.3 Режим адресации и использование регистров

Для Intel486, когда регистр используется как базовый компонент, дополнительные такты появляются, если этот регистр - регистр назначения предыдущей команды (полагая, что все команды уже в очереди предвыборки).

Пример:

```
add  esi,eax   ;esi является регистром назначения
mov  eax,[esi] ;esi является базой, 1 такт штрафа
```

Поскольку Pentium имеет два целочисленных конвейера, и каждый из них организационно соответствует целочисленному конвейеру Intel486, то использование регистра как базовой или индексной компоненты при вычислении адреса (в любом конвейере) требует дополнительного такта (AGI имеет место), если этот регистр есть регистр назначения предшествующей команды. Чтобы избежать AGI (блокировка генерации адреса), команды должны быть отделены по крайней мере одним тактом, путем размещения другой команды между ними.

Заметим, что некоторые команды имеют неявные чтение/запись в регистры. Команды, которые генерируют адреса неявно через ESP (push, pop/ret/call) также страдают от штрафов из-за AGI. (Если явная запись предшествует явному или неявному чтению).

Пример:

```
sub  esp,24              ;
                         ; 1 цикл стоит
push ebx                 ;
mov  esp,ebp             ;
                         ; 1 цикл стоит
pop  ebp                 ;
```

Push и pop также неявно пишут в ESP. При этом, однако, не возникает AGI, если следующая команда адресует через ESP.

Пример:

```
push  edi                ; нет остановки
mov   ebx,[esp]          ;
```

На Intel486 есть такт потери при декодировании с индексом или непосредственным операндом. В Pentium комбинация с непосредственным операндом не может образовывать пары. Когда необходимо использовать константы, использование непосредственного операнда является более эффективным, чем загрузка константы в регистр, однако при многократном использовании константы будет быстрее загрузить ее из памяти в регистр для многоразового использования.

```
mov  result,555          ;555 является
                         ;непосредственным
                         ;значением, а result
                         ;является смещением
```

```
mov  dword ptr [esp+4],1 ;1 является
                         ;непосредственным
                         ;значением, а 4 является
                         ;смещением
```

Intel486 имеет такт потери, когда используется регистр непосредственно после того, как его подрегистр (sub-register) был записан. Pentium нейтрален в этом отношении.

Пример (Pentium):

```
mov  al,0                ;1
mov  [esp],eax           ;2
```

Пример (Intel486):

```
mov  al,0                ;1
                         ;2
mov  [esp],eax           ;3
```

7.3.4 Быстродействие предвыборки

Устройство предвыборки Intel486 будет обращаться к КЭШ на кристалле, чтобы заполнить очередь предвыборки, когда RI пуст и имеется достаточно места в очереди для другой КЭШ-строки (16 бит). Если очередь предвыборки становится пустой, то для запуска следующей команды может понадобится 3 дополнительных такта. Очередь предвыборки имеет размер 32 байта (2 строки КЭШ).

Поскольку выборка данных имеет приоритет над запросами предвыборки команд, это может заблокировать работу устройства предвыборки. Поэтому в оптимизированном коде следует располагать команды так, чтобы шины памяти не были заняты продолжительной серией обращения к памяти. Команды следует располагать так, чтобы команды без обращения к памяти (регистровые команды) шли по крайней мере за два такта до того, как очередь предвыборки останется пустой. Это позволяет устройству предвыборки переслать строку КЭШ в очередь.

Такая аранжировка команд не будет влиять на характеристики 386 и Pentium. В общем, для компилятора трудно смоделировать поведение предвыборки Intel486. Последовательность из четырех команд с обращением к памяти без задержки (то есть без потерь на индексацию) вероятно вызовет остановку, поскольку буфер предвыборки будет опустошен.

7.3.5 Выравнивание

Intel486 имеет строку КЭШ длинной 16 байт, а Pentium - 32 байта. Поскольку Intel486 имеет только два буфера предвыборки (16 байт каждый), выравнивание кода имеет прямое влияние на характеристики Intel486 как результат эффективности буфера предвыборки. Кодовое выравнивание мало влияет на Pentium из-за его способности осуществлять предвыборку через строковые границы без штрафа. Для Intel386 выравнивание так же безразлично. Для оптимизации характеристик всего семейства рекомендуется, чтобы метки были выровнены по 16 байтным границам.

Не выровненый доступ в КЭШ данных занимает по крайней мере два дополнительных такта для Intel486 и Pentium.

Двухбайтные данные

Двухбайтный объект должен полностью содержаться внутри выровненного четырехбайтного слова (то есть двоичный адрес должен быть xxxx00, xxxx01, xxxx10, но не xxxx11). В Intel486 двухбайтные объекты должны быть выровнены по двухбайтным границам во избежание штрафа.

Четырехбайтные данные

Выравнивать четырехбайтный объект следует по четырехбайтной границе.

Восьмибайтные данные

Восьмибайтные объекты (64 битные двойной точности реальные числа) должны быть выровнены по восьмибайтной границе. Штраф для невыровненных восьмибайтных данных в Intel486 и Pentium составляет 3 такта.

7.3.6 Префиксные коды операции

Для Intel386, Intel486 префиксные коды требуют дополнительного такта на декодирование. На Pentium команда с префиксом образует пару в U - конвейере (PU), если команда (без префикса) может быть в паре в обоих конвейерах (UV) или в U - конвейере (PU). Это специальный случай образования пар. Команды с префиксами запускаются в U - конвейере и могут быть спарены.

Все эти префиксы: lock (F0h), segment override (2Eh, 3Eh, 24h, 64h, 65h, 36h), address size (67h), second opcode map (0Fh) и operand size (66h)

декодируются за один такт для каждого префикса. Заметим, что это включает все 16 битные команды, исполняемые в 32 битном режиме, поскольку префикс размера операнда является необходимым (т.е. mov word ptr [..], add word ptr [..]). Команды jcc near, которые имеют префикс 0Fh, обрабатывается иначе, так как не требует дополнительного такта при декодировании и принадлежит к PV группе. Другие 0Fh коды операций ведут себя как обычные префиксные команды. Для оптимизации следует избегать префиксных кодов.

Когда же эти коды все же должны быть использованы, имеется два случая, в которых может быть достигнуто перекрытие между дополнительным тактом, нужным для декодирования префикса и тактом для предыдущей команды, исполняемой в том же конвейере:

1. Такты штрафа от использования регистров результата предыдущей команды как базы или индекса (AGI);
2. Последний такт обрабатывающейся многотактной команды.

7.4 Селекция целочисленных команд

Ниже даны рекомендации, какие последовательности команд следует использовать, а каких нужно избегать.

7.4.1 Команда lea

1. Lea может быть использована иногда как трех/четырех операндная дополнительная команда (lea ecx,[eax+ebx*4+a]).
2. Во многих случаях lea или последовательность lea, add, schift может заменять команды умножения констант.
3. Это может быть также использовано, чтобы избежать копирования регистра, когда оба операнда команды add все еще нужны после выполнения сложения, поскольку lea нуждается в переписывании операндов.

Недостаток lea состоит в том, что она увеличивает вероятность AGI (блокировка генерации адреса) предыдущей командой. Lea полезна для сдвигов на 2, 4, 8, поскольку сдвиг занимает три такта на Intel486, в то время как lea занимает два такта. На Pentium lea может выполняться в U или V конвейере, но сдвиг может выполняться только в U - конвейере.

7.4.2 Комплексные команды

Нужно избегать использования сложных команд (например: enter, leave, loop). Вместо них лучше использовать последовательность простых команд.

7.4.3 Расширение байта нулями

Команда movzx имеет префикс и занимает три такта при выполнении (всего четыре такта). Вместо нее для Intel486 лучше использовать такую последовательность:

```
xor  eax,eax
mov  al,mem
```

Если это происходит внутри цикла, может оказаться возможным вывести "xor" из петли, если единственное присвоение для eax есть mov al,mem. Это имеет величайшую важность для Pentium, поскольку movzx не парабельна и новая последовательность может дать возможность образования пар.

7.4.4 Push mem

Команда push mem требует для Intel486 четыре такта. Рекомендуется использовать следующую последовательность, поскольку она требует только два такта для i486 и увеличивает возможность образования пар для Pentium.

```
mov  reg,mem
push reg
```

7.4.5 Малый код операции

Нужно максимально использовать команды длинной в один байт. Это уменьшит объем кода и поможет повысить плотность команд в КЭШ. Наиболее общий пример состоит в использовании inc и dec вместо сложения или вычитания константы 1 с add или sub.

7.4.6 8/16 битные операнды

Для 8 битных операндов нужно пытаться использовать байтовые коды операции вместо 32 битных операций на байтовых расширениях по знаку или нулю. Префиксы размера операнда используйте с 16 битными, но не с 8 битными операндами.

Знаковое расширение обычно достаточно дорого. Часто семантика может быть сохранена за счет нулевого расширения 16 битных операндов. Специфично, что код на языке Си в следующих примерах не нуждается в знаковом расширении.

```
static short int a, b;
    ...
  if(a == b)
    {
        ...
    }
```

Код для сравнения этих 16 битных операндов может быть таким:

```
xor  eax,eax    ;1
mov  ax,[a]     ;2   1  (prefix) + 1
mov  bx,[b]     ;3   1  (prefix) + 1
cmp  eax,ebx    ;4
```

Прямолинейный метод может быть медленнее:

```
mov   ebx,[b]   ;5
mov   eax,[a]   ;1   1  (prefix) + 3
cmp   ebx,eax   ;9
```

Конечно это может быть сделано только при определенных условиях, но эти обстоятельства имеют тенденцию быть достаточно общими. Это не будет работать, если сравнение было для "больше", "меньше или равно" и так далее; или если значение в eax или ebx должны были использоваться в другой операции, где требовалось знаковое расширение.

7.4.7 Сравнение

Используйте test для сравнения значения в регистре с нулем. Команда test объединяет операнды по AND без записи в регистр назначения. Если вы объединили по AND значение с самим собой, и результат устанавливает

флаг нулевого условия, то значение было нулевым. Команда test предпочтительнее and, и поскольку and пишет в регистр результат, который может вызвать в свою очередь AGI (блокировка генерации адреса), test лучше, чем cmp ...,0, так как размер команды меньше.

Используйте test для сравнения результата булевого and с непосредственной константой на "равно" или "неравно", если используется регистр eax (if(avar & 8) { ... }).

Test есть однотактная парабельная команда, когда форма такова: test eax,imm или test reg,reg. Другие формы команды test требуют два такта и не парабельны.

7.4.8 Вычисление адресов

Включайте вычисление адресов в команды load и store. Команды обращения к памяти могут иметь 4 операнда: перемещаемая во время загрузки константа, непосредственный операнд, базовый регистр, индексный регистр. (В сегментной модели сегментный регистр может вводить дополнительный операнд при вычислении линейного адреса). Во многих случаях несколько целочисленных команд могут быть исключены благодаря полному использованию операндов с обращением к памяти.

Когда есть выбор, использовать базовый или индексный регистр, всегда выбирайте базовый, поскольку в Intel486 нужен один дополнительный такт на индексацию.

7.4.9 Очистка регистра

Предпочтительный способ заслать нуль в регистр есть xor reg,reg. Это экономит размер кода, но устанавливает коды условий. В случае, когда коды условий необходимо сохранить, используйте mov reg,0.

7.4.10 Целое деление

Типичным случаем является, когда перед целым делением идет команда cdq (деление использует edx:eax как делитель и cdq устанавливает edx). Лучше копировать eax в edx, а затем сдвинуть edx на 31 позицию к знаковому расширению. Команда copy/shift требует того же количества тактов, как и cdq на Intel486 и Pentium, но copy/shift схема позволяет двум другим командам выполняться в то же время на Pentium. Если известно, что значение положительно, используйте xor edx,edx.

7.4.11 Прологовая последовательность (группа команд, предшествующая вызовам процедур)

Будьте осторожны, чтобы избежать AGI (блокировка генерации адреса) в прологе из-за регистра esp. Поскольку push может образовывать пару с другой командой push, при вызове функции следует использовать эти команды. Если возможно, загружайте параметры перед декрементацией esp.

Избегайте сравнения с непосредственным нулем. Очень часто, когда значение сравнивается с нулем, операция производящая сравнение, устанавливает коды условий, которые могут быть проверены прямо командой jcc. Наиболее заметное исключение - mov и lea. В этих случаях используйте test.

В подпрограммах, которые не вызывают других подпрограмм, используйте esp как базовый регистр, чтобы освободить ebp. Если вы не используете 32 битную модель, помните, что ebp не может быть использован как общецелевой базовый регистр, поскольку он обращается к стековому сегменту.

7.4.12 Эпилоговая последовательность (группа команд, следующая после возврата из процедур)

Если только 4 байта были размещены в стековых границах для текущей функции, вместо инкрементации стекового указатель на 4 используйте команду pop. Это позволит избежать AGI и поможет как Intel486, так и Pentium. Pentium использует два pop для 8 байтов.

7.4.13 Целое умножение на константу

Такое умножение может быть заменено быстрой серией shifts, adds, subs и leas.

Бинарный метод

В общем, если в бинарном представлении имеется 8 или менее бит, лучше не делать умножения. Для i486 это выгодно, начиная с длины константы 6 и меньше. Нужно выполнять сдвиг и сложение для каждого бита.

Метод факторизации

Это делается путем факторизации (представления) константы степенью двойки, плюс или минус единица. Если число может быть факторизовано степенью двойки, тогда умножение может быть выполнено серией сдвигов. Если факторизация используется, могут быть сгенерированы сдвиги и сложение или вычитание предыдущего результата. Если данное число плюс или минус 1 может быть факторизовано степенью двойки, может быть сгенерирован сдвиг предыдущего результата и сложение или вычитание оригинального операнда.

Итерации для контрольных позиций двойки должны быть сделаны от 31 до 1. Необходимо знать число сдвигов, чтобы сгенерировать нужный код. Например:

```
imul   eax,217              ; 10 тактов
```

После проверки на степень двойки найдено, что 217 делится на 31

```
217/31 = 7,    31 = 2^5 - 1
save shift = 5  и  original = sub_previous_result
```

После проверки 217/31 или 7 найдено, что 7+1+ есть делимое 8

```
save shift = 3  и  original = sub_operand
```

После факторизации команды могут быть сгенерированы в обратном порядке.

```
mov   ecx,eax               ;1
shl   eax,3                 ;2
sub   eax,ecx               ;3
mov   ecx,eax               ;4
shl   eax,5                 ;5
sub   eax,ecx               ;6
```

Этот код позволяет планировать другие команды в V - конвейере Pentium.

7.5 Специфичные оптимизации для Pentium.

Образование пар не может быть произведено, если:

1. Следующие две команды относятся к неспариваемым командам (см. Таб. 8-1). В общем, большинство простых команд спариваемы (парабельны).
2. Две команды имеют явную или неявную зависимость по регистрам. Имеются некоторые специальные исключения из этого правила, где регистровая зависимость в паре допускается. Они будут описаны позже.
3. Обе команды не находятся в КЭШ команд. Исключением, которое позволяет получить пару, является условие, что первая команда есть однобайтная команда.

7.5.1 Парабельность команд

Непарабельные команды (NP):

1. Shift/rotate со счетчиком сдвига в CL.
2. Длинные арифметические команды, например: ret, enter, pusha, movs, rep stos, loopnz.
3. Некоторые специфичные команды, например: push sreg, call far.

Парабельные команды, запускаемые в U или V конвейерах (UV):

1. Большинство 8/32 битных операций АЛУ, например: add, inc, xor.
2. Все 8/32 битные команды сравнения, например: cmp, test.
3. Все 8/32 битные стековые операции, использующие регистры, например: push reg, pop reg.

Парабельные команды использующие конвейер U (PU):

1. Команды с перекосом и циклические команды, например: adc, sbb.
2. Префиксные команды.
3. Сдвиг с непосредственным счетом.
4. Некоторые плавающие операции, например: fadd, fmul, fld.

Эти команды должны быть запущены для U конвейера и могут образовывать пару с подходящей командой в V конвейере. Эти команды никогда не могут выполнятся в V - конвейере.

Парабельные команды, запускаемые в V - конвейере (PV):

1. Команды простых управляющих переходов, например: call near, jmp near, jcc. Это включает как jcc short, так и jcc near (которая имеет префикс 0Fh) версии команд условного перехода.

Эти команды могут выполнятся в U и V конвейерах, но оно образуют пару только находясь в V - конвейере. Поскольку эти команды изменяют

счетчик команд (EIP), они не могут образовывать пары в V - конвейере, так как следующая команда может быть не соседней. Даже когда ветвление в U - конвейере предсказано "не будет", они не будут образовывать пары с последующей командой.

Непарабельность из-за регистров

Парабельность команд также зависит от операндов. Ниже даны примеры этого:

1. Первая команда пишет в регистр, из которого вторая команда читает (потоковая зависимость). Пример:

```
mov  eax,8  mov  [ebp],eax
```

2. Обе команды пишут в тот же регистр (выходная зависимость). Пример:

```
mov  eax,8
mov  eax,[esp]
```

Эти ограничения не относятся к парам команд, которые пишут в EFLAGS регистр (т.е. две АЛУ операции, которые изменяют коды условий). Коды условий после выполнения пары команд будут иметь значение от команды из V - конвейера.

Заметим, что пара команд, в которой первая команда читает регистр, а вторая пишет в него (антизависимость), является парой. Пример:

```
mov  eax,ebx      mov  ebx,[ebp]
```

С целью определения регистровой зависимости, обращение к 8 битному или к 16 битному регистру обрабатывается как обращение к 32 битному регистру, отсюда:

```
mov  al,1
mov  ah,0
```

не образуют пары очевидной выходной зависимости по eax.

```
add  eax,mem1      add ebx,mem2     ;1
  (add)              (add)          ;2 2-такта
```

Специальные пары

Имеются команды, которые образуют пары, несмотря на общие запрещающие правила. Эти специальные пары обходят регистровые зависимости. Большая часть этих исключений использует неявное чтение/запись в ESP регистр или неявную запись в коды условий: Указатель стека:

1. push reg/imm ; push reg/imm
2. push reg/imm ; call
3. push reg ; pop

Коды условий:

1. cmp ; jcc
2. add ; jne

Заметим, что специальные пары, которые содержат только push/pop команды, могут иметь только непосредственные или регистровые операнды.

Ограничения на выполнение пар

Имеются некоторые пары, которые запускаются одновременно, но выполняться в параллель не могут:

1. Если две команды обращаются к одному и тому же банку памяти, то второе требование (V - конвейер) обязано ждать завершения первого сообщения. Конфликт по банку имеет место, когда биты 2 и 4 в физических адресах совпадают. Это происходит, потому что КЭШ организован как 8 банков с шириной входа 32 бита. Конфликт по банку обходится штрафом в один такт для V - конвейера.
2. Межконвейерная конкуренция должна сохранять, тем не менее, порядок обращения в память. Многотактная команда в U - конвейере будет выполняться в одиночестве до ее последнего обращения в память.

Вышеуказанные команды прибавляют содержимое регистра к ячейке памяти, затем помещают результат в регистр. Сложение с операндом из памяти требует два такта: первый такт - выборка из КЭШ, второй - непосредственно сложение. Поскольку имеется только одно обращение к памяти в команде U - конвейера, сложение в V - конвейере стартует в том же такте.

```
add  mem1,eax                    ;1
  (add)                          ;2
  (add)          add mem2,ebx    ;3
                   (add)         ;4
```

```
(add)          ;5
```

Выше приведенные команды прибавляют содержимое регистра к ячейке памяти и помещают результат в память. Такая операция требует три такта; первый - загрузка значения, второй - операция, третий - запоминание результата. В паре последний такт U - команды перекрывается с первым тактом V - команды.

Никакие другие команды не могут начать исполнение, пока исполняемые команды не будут завершены.

Чтобы показать возможность планирования и образования пар, лучше всего запустить последовательность простых команд вместо сложной команды, которая требует того же числа тактов. Последовательность простых команд имеет больше возможностей по образованию пар. Разработчики компиляторов и программисты могут тоже реконструировать комплексную форму, если не удается с ней образовать пару. Load/store стиль программирования требует много регистров и увеличивает объем кода. Это влияет на характеристики i486, хотя этот эффект вторичного порядка. Чтобы скомпенсировать нехватку регистров, нужны более совершенные средства распределения регистров и планирования команд так, чтобы дополнительные регистры использовались для повышения параллелизма.

7.5.2 Оптимизация для FPU Pentium

Для FPU Pentium нужна оптимизация, чтобы увеличить эффективность его конвейера.

Примеры программ для плавающей запятой

```
FORTRAN sourse:
  subroutine da(x, y, z, n)
  dimension x(n), y(n)
  do 10 i=1,n
10   x(i) = x(i) + y(i)*z
  return
end

Assembly code                    Pentium/i486
TopOfLoop:
   fld  dword ptr [esp+8]      ;1/1
   fmul dword ptr [ebx+eax*4] ;2/5
   fadd dword ptr [ecx+eax*4] ;5/16
```

```
    fstp dword ptr [ecx+eax*4] ;9/26
    inc  eax                   ;11/33
    cmp  eax,ebp               ;12/34
    jle  TopOfLoop             ;12/36+2 for branch

Всего: 12 тактов на итерацию
```

На Intel486 время, затрачиваемое на add и mul, зависит от значения. В этом примере, для умножения было получено 11, а для сложения - 10. Load требует 3 такта, store требует 7 тактов. Дополнительный такт перед fmul есть штраф за индексацию для fmull. Команды fadd и fstp не показывают индексного штрафа, поскольку штраф перекрывается с исполнением предыдущей команды. Такого перекрытия нет для fld и fxch.

На Pentium результат команды fadd и fmul может быть использован через три такта после старта, кроме случая, когда используется fst. Когда fst использует результат другой плавающей команды, необходим дополнительный такт. Команда fst выполняется за два такта и ничего нельзя выполнить в параллель с ней. Благодаря уменьшению числа тактов на плавающей операции, существенно улучшились характеристики Pentium, однако в этом примере нет перекрытия плавающих операций. Оно показано в следующей секции.

Чтобы добиться большего параллелизма, необходимо выполнить развертку независимых итерации:

```
TopOfLoop:                          ;Pentium/i486
   fld  dword ptr [esp+8]           ;1/1
   fmul dword ptr [ebx+eax*4]       ;2/5
   fadd dword ptr [ecx+eax*4]       ;5/16
   fstp dword ptr [ecx+eax*4]       ;9/26
   fld  dword ptr [esp+8]           ;11/33
   fmul dword ptr [ebx+eax*4+4]     ;12/37
   fadd dword ptr [ecx+eax*4+4]     ;15/48
   fstp dword ptr [ecx+eax*4+4]     ;19/58
   fld  dword ptr [esp+8]           ;21/65
   fmul dword ptr [ebx+eax*4+8]     ;22/69
   fadd dword ptr [ecx+eax*4+8]     ;25/80
   fstp dword ptr [ecx+eax*4+8]     ;29/90
   add  eax,3                       ;31/97
   cmp  eax,ebp                     ;32/98
   jle  TopOfLoop                   ;32/100+2 for branch

Всего: 32 такта (10.7/iteration)
```

Уменьшение числа тактов было получено при развертке за счет исключения некоторых операций управления циклом. Чтобы получить дальнейшее улучшение, необходимо перекрывать выполнение плавающих операций, чтобы устранять задержки.

Большинство плавающих операций требует, чтобы один операнд и результат были на верхушке стека. Это вводит зависимость смежных команд и препятствует их перекрытию.

Очевидный путь изменить дело - переделать архитектуру и вместо стека ввести плавающие регистры с произвольным доступом. Но при этом теряется программная совместимость вверх и вниз. Вместо этого сделали быстрой команду fxch. Это обеспечивает возможность избежать зависимости по вершине стека. Команда fxch может образовывать пару с большинством плавающих команд, поэтому нет никакого штрафа из-за нее в Pentium. Для Intel486 команда fxch требует 4 такта.

Чтобы получить выигрыш от выставленного при развертке параллелизма итераций, нужно применить планирование.

Ассемблерный код после перестановок и планирования:

```
                                  i486 P5 ST0      ST1      ST2
TopOfLoop:
  fld  dword ptr [esp+8]          ;1   1  z
  fmul dword ptr [ebx+eax*4]      ;5   2  y0*z
  fld  dword ptr [esp+8]          ;16  3  z        y0*z
  fmul dword ptr [ebx+eax*4+4];30  4  y1*z    y0*z
  fxch st(1)                      ;31  4  y0*z     y1*z
  fadd dwrod ptr [ecx+eax*4]      ;36  5  x0+y0*z y1*z
  fld  dword ptr [esp+8]          ;46  6  z        x0+y0*z y1*z
  fmul dword ptr [ebx+eax*4+8];50  7  y2*z    x0+y0*z y1*z
  fxch st(2)                      ;61  7  y1*z     x0+y0*z y2*z
  fadd dwrod ptr [ecx+eax*4+4];66  8  x1+y1*z x0+y0*z y2*z
  fxch st(1)                      ;76  8  x0+x1*z x1+y1*z y2*z
  fstp dword ptr [ecx+eax*4]      ;81  9  x1+y1*z y2*z
  fxch st(1)                      ;88  11 y2*z     x1+y1*z
  fadd dword ptr [ecx+eax*4+8];93  12 x2+y2*z x1+y1*z
  fxch st(1)                      ;103 12 x1+y1*z x2+y2*z
  fstp dword ptr [ecx+eax*4+4];108 13 x2+y2*z
  fstp dword ptr [ecx+eax*4+8];116 16
  add  eax,3                      ;123 18
  cmp  eax,ebp                    ;124 19
  jle  TopOfLoop                  ;126+2 19
                                  |
```

```
          (for branch)
Всего: 18 тактов (6.3/iteration)
```

На Intel486 индексный штраф и добавленная стоимость fxch очевидны. Индексный штраф не перекрывается с командой fxch.

На Pentium fxch образует пару с предыдущими командами fadd и fmul и выполняется параллельно с ними (такты 7, 8, 12). Команда fxch передает операнд в позицию для следующей плавающей команды. Есть потеря такта на такте 15 из-за операции store, идущей три такта после команды, определяющей ее операнд. Команда fxch не образует пару с fst и требует одного такта как отдельная команда (9-11).

Правила для FXCH

Команда fxch может выполняться "свободно" при следующих условиях:

- За командой FPU следует команда fxch.
- Команда FPU принадлежит к следующему списку команд: fadd, fsub, fmul, fld, fcom, fucom, fchs, ftst, fabs, fdiv.
- Эта fxch уже была выполнена раньше. Это означает, что в КЭШ отмечается первое исполнение команды, так что образование пары происходит только при втором обращении к команде. Это означает, что эта команда почти "свободна" и может быть использована для доступа к элементам нижних уровней стека FPU вместо занесения этих элементов в память.

Операнды из памяти

Выполнение плавающей операции с операндом из памяти, вместо использования стекового регистра, не требует такта. В целой части Pentium лучше избегать обращений к памяти. В плавающей части вас подстрекают использовать операнды из памяти.

Остановка в плавающей части

Имеются случаи, когда между двумя операндами имеется задержка. Между такими командами надо вставить другие команды. Эти команды могут быть целочисленными или плавающими, но не должны сами вызывать простоев. Число вставляемых команд зависит от длинны задержки.

Примером этого является случай, когда команда FPU зависит непосредственно от результата предыдущей команды FPU. В этом случае между этими командами хорошо вставить целочисленную команду, даже если эта команда выполняет управление циклом. Следующий пример реструктурирует цикл следующим образом:

```
for(i = 0; i < iSize; i++) array1[i] += array2[i];

                        Pentium          Intel486
TopOfLoop:
  flds   [eax+array2];2 - AGI            3
  fadds  [eax+array1];1                  3
  fstps  [eax+array1];5 - ждет fadds     14 - ждет fadds
  add     eax,4       ;1                 1
  jnc     TopOfLoop   ;0 - пара с add    3
      ...
                      ;9                 24

                        Pentium          Intel486
TopOfLoop:
  fstps  [eax+array1];4 - ждет fadds     10 - ждет fadds
                                              и AGI
LoopEntryPoint:
  flds   [eax+array2];1                  3
  fadds  [eax+array1];1                  3
  add    eax,4       ;1                 1
  jnc    TopOfLoop   ;0 - пара с add    3
      ...
                      7                  20
```

При размещении целых команд между fadds и fstps оба конвейера могут выполнять целые команды, пока fadds заканчивается в FPU и до начала исполнения fstps. Заметим, что это новая циклическая структура требует отдельной точки входа для первой итерации, поскольку цикл нужно начинать с flds. Итак, необходимо иметь дополнительный fstps после перехода, чтобы закончить последнюю итерацию.

1. Плавающие записи обязаны ждать дополнительного такта для получения плавающего операнда. После fld, fst необходимо ждать один такт. После распространенных арифметических операций fmul и fadd, которые обычно имеют задержку два, fst ждет дополнительный такт, поэтому

в общем требуется три такта. Этот ряд включает также команды faddp, fsubrp ... .

```
fld    mem1        ; 1   fld
                   ; 2   fst ждет
fst    mem2        ; 3   fst
fadd   mem1        ; 1   fadd
                   ; 2   fadd задержка
                   ; 3   fadd задержка
                   ; 4   fst ждет
fst    mem2        ; 5   fst
```

В следующем примере store не зависит от предыдущей загрузки:

```
fld    mem1        ; 1
fld    mem2        ; 2
fxch   st(1)       ; 2
fst    mem3        ; 3
```

2. Регистр может быть использован непосредственно после загрузки (fld):

```
fld    mem1        ; 1
fadd   mem2        ; 2,3,4
```

3. Использование операцией регистра FPU сразу после того, как он был записан другой fadd, fsub, или fmul вызывает два такта задержки. Если другие команды размещены между этими двумя, то задержки и потенциальные остановки могут быть исключены.

4. Есть многотактные команды (fdiv и fsqrt). Во время их выполнения в FPU, в параллель могут выполняться целые команды. Число целых команд зависит от длинны плавающих команд.

5. Целые операции mul и imul выполняются в FPU и поэтому не могут выполняться параллельно с командами FPU.

6. Команда fmul задерживается на один такт, если непосредственно предшествующий такт выполнил fmul или пару fmul/fxch. Умножитель может менять пару операндов только каждый второй такт.

7. Трансцендентные операции выполняются в U - конвейере и не допускают никакого перекрытия, значит, целая команда, следующая за такой операцией, будет ждать ее завершения.

8. Следует избегать плавающих операций, использующих целые операнды (fiadd или fisub). Эти команды следует разбивать на две: fild и плавающую команду. Число тактов до запуска другой команды (производительность) для fiadd составляет 4, в то время, как для fild и простой команды FPU требуется 1 такт, например:

```
fiadd  [ebp]     ; 4          fild  [ebp]     ; 1
                              faddp st(1)     ; 2
```

Использование команд fild-faddp оставляет два свободных такта для других команд.

9. Команда fstsw, которая обычно появляется после команды плавающего сравнения (fcom, fcomp, fcompp) задерживает на три такта. После команды сравнения можно поместить другие команды, чтобы убрать задержку.

10. Плавающие операции mov mem/imm to mem следует выполнять с помощью целых mov (если преобразование точности не требуется) вместо операции fld-fstp, например:

```
для величины double
   fld    [ebp]  ; 1          mov   eax,[ebp]       ; 1
                 ; 2          mov   edx,[ebp+4]     ; 1
   fstp   [edi]  ; 3          mov   [edi],eax       ; 2
                               mov   [edi+4],edx     ; 2
для величины float
   fld    [ebp]  ;1           mov   eax,[ebp]       ; 1
                 ;2           mov   [edi],eax       ; 2
   fstp   [edi]  ;3
```

Эта оптимизация также пригодна для Intel486.

11. Трансцендентные операции выполняются на Pentium много быстрее, чем на Intel486. Может оказаться заслуживающим внимания реализация некоторых функций математической библиотеки в виде встроенных в код (inline) процедур, поскольку затраты на вызовы и прологовые и эпило-

говые накладные расходы связанные с библиотекой не могут считаться пренебрежительными. Эмуляция этих операций программно не будет быстрее их аппаратной реализации, пока точность является удовлетворительной.

12. Целые команды в общем перекрываются с плавающими кроме случая, когда после команды FPU была fxch. В этом случае появляется задержка в один такт.

```
U конвейер        V конвейер
fadd              fxch          ; 1
                                ; 2 fxch задержка
mov  eax,1        inc  edx      ; 3
```

7.6 Обобщение

Следующая таблица суммирует различия в архитектуре для Intel386, Intel486 и Pentium и связанные с этим особенности генерации кода. Можно выделить ряд стратегий генерации кода, которые обеспечивают оптимальные характеристики для всего семейства, исключая использование команды FXCH, чтобы максимизировать производительность Pentium на плавающих операциях. Эта оптимизация должна быть управляема пользователем.

	Intel386	Intel486	Pentium
Внутренний КЭШ	Нет	8K совмещенный	8K код, 8K данные
Предвыборка	4*4 байта, заполняемые через доступ к внешней памяти	2*16 байта, заполняемые из КЭШ через общую шину	4*32 байта, заполняемые из КЭШ через выделенную шину
Декодирование	Очередь FIFO глубиной 3	Составная часть конвейера	Составная часть конвейера
Ядро	Перекрытие некоторых инструкций	Конвейер в 5 ступеней	Конвейер в 5 ступеней и суперскаляр
Математика	Сопроцессор	На кристалле	На кристалле и конвейеризован

	Процессор	Intel386	Intel486	Pentium
	Оптимизация			
КЭШ	Перекрытие mem с no-mem	Все равно	Перекрытие, если 4 подряд	Все равно
Предвыборка	Выравнивание	0-MOD-4	0-MOD-16	Все равно
Конвейериз. исполняющее ядро	Базовая версия индекса	Все равно	Использовать базу	Все равно
	Избегать AGI	Все равно	В следующей команде	В следующих трех командах
	Отбор команд	1 такт штрафа	Короткие команды	Короткие команды
Суперскаляр	Планирование	Все равно	Все равно	Пары
Конвейерез. FPU с FXCH	FP планирование	-	-	Некоторые команды

Рекомендации для семейства:

1. Перекрытие mem с no-mem : ничего не делать;
2. Выравнивание кодов : 0-MOD-16 в цикле;
3. Базовая версия индекса : использовать базу;
4. Избегать AGI : следующие три команды;
5. Отбор команд : короткая последовательность команд;
6. Образование пар : образовывать;
7. FP планирование : избегать FXCH;

Таб. 7-1 Парабельность команд

Команда	Формат	
ADC Сложение с переносом		PU
ADD Сложение		UV
AND Логическое "И"		UV
CALL Вызов процедуры (прямой в том же сегменте)	11101000:full displacement	PV
CMP Сравнить два операнда		UV
DEC Декремент		UV
INC Инкремент		UV
Jcc Условный переход		PV

JMP Переход (в том же сегменте): *короткий*	11101011:8-bit disp	PV
прямой	11101001:full displacement	PV
LEA Загрузить эффективный адрес		UV
MOV Переслать данные		UV
NOP Нет операции	10010000	UV
OR Логическое "ИЛИ"		UV
POP Извлечь из стека: *reg*	10001111:11000reg	UV
or	01011reg	UV
PUSH Поместить операнд в стек: *reg*	11111111:11110reg	UV
or	01010reg	UV
immediate	011010s0:immediate data	UV
RCL Циклический сдвиг влево через перенос:		
reg by 1	101000w:11010reg	PU
memory by 1	1101000w:mod010r/m	PU
reg by imm8	1100000w:11010reg:imm8	PU
memory by imm8	1100000w:mod010r/m:imm8	PU
RCR Циклический сдвиг вправо через перенос:		
reg by 1	1101000w:11011reg	PU
memory by 1	1101000w:mod011r/m	PU
reg by imm8	1100000w:11011reg:imm8	PU
memory by imm8	1100000w:mod011r/m:imm8	PU
ROL Циклический сдвиг влево:		
reg by 1	1101000w:11000reg	PU
memory by 1	1101000w:mod000r/m	PU
reg by imm8	1100000w:11000reg:imm8	PU
memory by imm8	1100000w:mod000r/m:imm8	PU
ROR Циклический сдвиг вправо:		
reg by 1	1101000w:11001reg	PU
memory by 1	1101000w:mod001r/m	PU
reg by imm8	1100000w:11001reg:imm8	PU
memory by imm8	1100000w:mod001r/m:imm8	PU
SAR Арифметический сдвиг вправо:		
reg by 1	1101000w:11111reg	PU
memory by 1	1101000w:mod111r/m	PU
reg by imm8	1100000w:11111reg:imm8	PU
memory by imm8	1100000w:mod111r/m:imm8	PU
SBB Целочисленное вычитание с заемом		PU
SHL Логический сдвиг влево: *reg by 1*	1101000w:11100reg	PU
memory by 1	1101000w:mod100r/m	PU
reg by imm8	1100000w:11100reg:imm8	PU
memory by imm8	1100000w:mod100r/m:imm8	PU

SHR Логический сдвиг вправо:		
reg by 1	1101000w:11110reg	PU
memory by 1	1101000w:mod110r/m	PU
reg by imm8	1100000w:11110reg:imm8	PU
memory by imm8	1100000w:mod110r/m:imm8	PU
SUB Целочисленное вычитание		UV
TEST Сравнить логически: *reg1 & reg2*	1000010w:11reg1reg2	UV
memory & reg	1000010w:modregr/m	UV
immediate & AC	1010100w:immediate data	UV
XOR Логическое "Исключающее ИЛИ"		UV
FABS Абсолютное значение		FX
FADD Сложение		FX
FADDP Сложение с извлечением из стека FPU		FX
FCHS Изменить знак		FX
FCOM Сравнить вещественные		FX
FCOMP Сравнить вещественные и извлечь из стека FPU		FX
FDIV Деление		FX
FDIVP Деление с извлечением из стека FPU		FX
FDIVR Инверсное деление		FX
FDIVRP Инверсное деление с извлечением из стека FPU		FX
FISUBR Инверсное целочисленное вычитание		FX
FLD Загрузить вещественное: *32-bit memory*	11011001:mod000r/m	FX
64-bit memory	11011101:mod000r/m	FX
ST(i)	11011001:11000ST(i)	FX
FMUL Умножение		FX
FMULP Умножение с извлечением из стека FPU		FX
FSUB Вычитание		FX
FSUBP Вычитание с извлечением из стека FPU		FX
FSUBR Инверсное вычитание		FX
FSUBRP Инверсное вычитание с извлечением из стека FPU		FX
FTST Проверка		FX
FUCOM Сравнение неупорядоченных вещественных		FX
FUCOMP Сравнение неупорядоченных вещественных с извлечением из стека FPU		FX

NP - команда не парабельна, выполняется в U конвейере;
PU - команда парабельна, если выполняется в U конвейере;
PV - команда парабельна, если выполняется в V конвейере;
UV - команда парабельна в любом конвейере;
FX - команда FPU парабельна с FXCH.

Приложение H

Некоторая специализированная информация о микропроцессоре Pentium не входит в стандартную документацию фирмы Intel. Данная информация считается конфиденциальной и может быть получена только при заключении соглашения о нераспространении. За подробностями обращайтесь на фиму Intel.

Содержание